Visual Linguistics with R

A practical introduction to quantitative Interactional Linguistics

Christoph Rühlemann

John Benjamins Publishing Company

Visual Linguistics with R

Visual Linguistics with R

A practical introduction to quantitative Interactional Linguistics

Christoph Rühlemann
University of Freiburg

John Benjamins Publishing Company
Amsterdam / Philadelphia

The paper used in this publication meets the minimum requirements of the American National Standard for Information Sciences – Permanence of Paper for Printed Library Materials, ANSI Z39.48-1984.

DOI 10.1075/z.228

Cataloging-in-Publication Data available from Library of Congress:
LCCN 2020015567 (PRINT) / 2020015568 (E-BOOK)

ISBN 978 90 272 0709 8 (HB) / ISBN 978 90 272 0710 4 (PB)
ISBN 978 90 272 6098 7 (E-BOOK)

Table of contents

VLR Data and Code files can be found here:
https://doi.org/10.1075/z.228.resources

Acknowledgments

This book has a long history. It started many years ago in Munich where Franca Kirchberg taught me the first steps in R. My relationship with R was greatly deepened in Texas in Stefan Th. Gries's R bootcamp. It finally caught fire when I realized that using R for visualization allowed the combination of linguistic research and artistic creation.

I'm indebted to Florian Dolberg, and his detailed comments on the first few chapters of an earlier draft. Another debt is owed to Martin Schweinberger, who cast a critical eye on one chapter. The biggest thank you is due to Mark Dingemanse, who delivered the ideal review of the pre-final manuscript: abundantly rich in detail, invariably constructive in his critique, and most helpful in making excellent suggestions for improvement.

A warm thank you goes to editor Kees Vaes and his remarkable patience in the face of multiple deadlines the author did not meet, and to Susan Hendriks of the production team at Benjamins who ensured that the passage from manuscript submission through production to publication was an entirely pleasant experience.

With all this assistance I hasten to say that all the faults and flaws that may have remained in the book are entirely mine.

Chapter 1

Introduction

1.1 On the design of this book

This book is primarily a textbook on R, a programming language and environment for statistical analysis and visualization. Its primary aim is to introduce R as a research instrument in quantitative Interactional Linguistics. Focusing on *visualization* in R, the book takes a how-to approach to the case studies explaining in good detail how key graphs in the case studies were coded. It also includes task sections to enable readers to conduct their own research in R.

But it's more than just a textbook. The book also provides original analyses on conversational talk-in-interaction based on corpus data from the conversational component of the British National Corpus (e.g., Hoffmann et al. 2008) and the Narrative Corpus (e.g., Rühlemann & O'Donnell 2012). Seen from this secondary aim, the book is also a research volume.

1.2 On Interactional Linguistics, Conversation Analysis, and Corpus Linguistics

Interactional Linguistics is heavily indebted to Conversation Analysis from which it spun off. The two fields share the occupation with talk-in-interaction in conversation, which is widely seen as the "core ecology for language use" (Levinson & Holler 2014: 1). Talk-in-interaction in conversation is conceptualized as "the contingently connected sequences of turns in which we each 'act', and which the other's – our recipient's – response to our turn relies upon, and embodies, his/her understanding of what we were doing and what we meant to convey in our (prior) turn" (Drew 2013: 131). The perspectives from which the two fields approach talk-in-interaction vary: Conversation Analysis looks at talk-in-interaction from the point of view of interaction: its main interest is in uncovering the basic 'machinery' (Sacks 1984) of how social interaction works at large; Interactional Linguistics approaches talk-in-interaction from the point of view of talk: its main interest is in "conceptualizing linguistic structure as resources for social interaction" (Couper-Kuhlen & Selting 2018: 4). Moreover, Conversation Analysis and Interactional Linguistics share the methodical attention to detail in carefully crafted transcripts of talk-in-interaction as well as a preference for qualitative analysis of select instances of features of interest considered in the sequential context of their occurrence. Corpus Linguistics, on the other hand, is a primarily quantitative method. Its basic assumption is that by ploughing through vast quantities of machine-readable text, "patterns emerge that could not be seen before" (Tognini Bonelli 2010: 18). In this respect, Corpus Linguistics works like areal archeology: by surveying masses of data 'from above', that is, at a remove from their use in situated talk-in-interaction, it can discover structures that are hidden to the observer (and the participant) 'below'. In recent years, quantification has come to be accepted as an addition to the methodological toolbox in Interactional Linguistics (cf. Couper-Kuhlen & Selting 2018: 13). Even in Conversation Analysis, the more conservative of the two fields, the traditional insistence on the qualitative method alone has been losing ground since Heritage's (1999) principled critique (cf. also Stivers 2015) such that, a good twenty years on, for Stokoe (2018) "CA is both [qualitative and quantitative]" (Stokoe 2018: 127).

Given that the case studies assembled in this voume aim to integrate methods of Interactional Linguistics and Conversation Analysis with methods of Corpus Linguistics, this book is also a methods straddler, building on, and adding to, the dynamic growth of the field of Corpus Pragmatics (cf. Aijmer & Rühlemann 2015).

1.3 On the book's two languages

Consistent with its double focus as primarily a textbook for visualization with R and secondarily a research volume, this book is about two languages. The one is the natural language of face-to-face conversation. This language is examined in the case studies. Its origin "would seem to lie somewhere between 1.4 Ma and ca 600,000 years ago" (Levinson & Holler 2014: 2). The other language the book is concerned with is R, a powerful, non-natural programming language. It has come into existence just very recently; in fact, R is a spin-off of S, an earlier programming language created in the 1970s (cf. Crawley 2007: vii). Despite its recency, R has made major inroads in the sciences across the board; where linguists embrace quantitative methods, "R seems to have become the *de facto* standard tool" (Levshina 2015: 21).

It is worth briefly contemplating what commonalities natural and non-natural languages such as R share and how they are distinguished, sometimes quite fundamentally.

Both languages have in common that they are built on lexis and syntax: you need a certain amount of vocabulary to address things of interest to you in either language and you need to order that vocabulary in ways to ensure well-formedness. For example, if you talk about bread, it will be helpful if you have the respective word for it at your disposal: 'bread' in English, 'pane' in Italian, 'psomí' in Greek and so forth. In R, if you want to define a vector with the values 1, 3, and 5 you need to have the function `c()` (for 'combine' or 'concatenate') in your vocabulary. If you want to specify the type of bread, you will have to place the adjective 'white' in front of 'bread' in English, while in Greek the adjective 'levkó' can go behind the noun for emphasis. In R, you wrap `c()` around the values, thus: `c(1, 3, 5)`; if you put the `c` behind the closing bracket – `(1, 3, 5)c` – R will return an error.

R and conversation also have in commmon that there are numerous ways to perform one and the same action. For example, to order a pint of beer in a bar, you can communicate this to the bartender in various ways, for example, by saying "a pint of lager, please", or just "lager", or by saying "this one" and pointing to the tap. Which of these (or other) actions you choose will depend on a myriad of situated variables. Similarly, in R, if you need to transform measurements in seconds stored in one column of a data frame into milliseconds stored in another column, there are many ways to achieve this: you can define that new column using a formula such as `df$milliseconds ← df$seconds*1000`, a command that will yield the desired output for each row in the data frame; alternatively, you can use an existing function such as `lapply()` and, inside `lapply`, define a function for that transformation (a more coding-intensive method); or you might go for a `for` loop iterating over all existing measurements in seconds and transforming them into milliseconds (a computationally expensive method). While in face-to-face communication the choice of wording may depend on numerous variables, the choice of code in R is governed by two, often conflicting, principles: economy and transparency. The economy principle favors sparse solutions over solutions that require large amounts of code or processing time by the computer. The transparency principle favors code that is simple and easy to process by the user. Obviously, the two principles often clash: one-liners may be sparse in terms of code but their structure may be hard to grasp for novices.

The most obvious difference between R and conversation is the fact that the latter is inherently multimodal. This is because "language production always occurs with the involvement of not only the vocal tract and lungs, but also the trunk, the head, the face, the eyes and, normally, the hands" (Levinson & Holler 2014: 1) and because conversationalists use these physical resources as communicative resources. For example, hand gestures can add semantic and pragmatic information to verbal information (e.g., Holler et al. 2009; Holler & Levinson 2019). The contrast with R, as any other programming language, could not be starker: it recognizes only (written) verbal input.

The differences continue with the ways that inadequate lexical choices or faulty syntax are dealt with. Natural language in interaction is highly forgiving. An ill-formed utterance such as "I bread" said in a bakery will move the interaction with the salesperson in the intended direction, and you will eventually leave the shop with something to satisfy your hunger. In cases where an ill-formed locution does pose a problem, natural language users can repair the trouble. They can use "practices designed for dealing with the sorts of difficulties which emerge in talk" (Liddicoat 2007: 171). These have evolved over millenia and are widely used (on average, once every 1.4 minutes across typologically diverse languages; cf. Dingemanse et al. 2015) ensuring that mistakes and ambiguities of verbal communication are ironed out and interaction can progress beyond the trouble.

Non-natural languages, by contrast, are much more unforgiving. Either a line of code is well-formed, in which case you will get the desired result, or it is ill-formed and you won't get the result and will have to start all over again. Typically, the maximum amount of 'repair' you can get is an error message or a warning; only in very few cases will R make a suggestion as to how to address the problem. So, the onus of working out what went wrong, why it went wrong, and how to 'put' it right will be on you alone.

Another difference relates to the degree to which uncertainty, vagueness, and indirectness are admitted. Uncertainty is a key semantic domain in natural language, covering the intermediate degrees between 'yes' and 'no' (Halliday & Matthiessen 2004: 147). Thus, it makes a huge difference whether you can, could, might, should, or will do X. Also, speakers often use so-called category markers, such as 'and stuff' and 'or something' thereby marking the actual choice as just one possible actualization of a broader category. For example, a waiter's question "Some beer or something?" is fully legally answered by "A glass of wine please" as 'wine' is contained in the category 'drinks' just as 'beer' is. Finally, as is well known from speech act theory, you can say one thing to mean another (Searle 1975). For example, "It's cold in here" may be taken as a request to turn up the heating. None of this is possible in non-natural languages,: they leave no room for uncertainty – any piece of code is an unhedged instruction to do X – and they process language literally only – anything you say is what you mean and anything you don't say cannot be meant.

Yet probably the starkest difference lies in the overall purpose for which the languages are used. We use natural language to achieve goals in interaction with others. That is, we say something to someone to achieve a particular purpose related to that someone: to entertain, to instruct, to describe, to assert, to affiliate, to insult, to charm, to persuade, and so forth. That purpose is only achieved in interaction with a communicative partner, the hearer. It is not enough to say something in order for that act of speech to represent a speech act, i.e. to do something or achieve something. In order for words to do and achieve something the words need to be heard, processed and, in some way or other, acted upon by a hearer. Natural language use is fundamentally interactional.

Superficially, the way a non-natural language such as R functions seems not dissimilar – we use a piece of code to achieve a purpose: to do a calculation, to evaluate a distribution, to draw a graph. But these achievements are achievements in themselves, they do not require an interlocutor to be ratified and, ultimately, consummated. Anything you 'say' in R is said to a machine that is blind, deaf, and – despite the massive amount of expertise programmed into it – ultimately dumb as it cannot create any new idea out of itself. Anything you achieve in R – a calculation, a significance test, a chart – is complete in itself; it requires no uptake or ratification, it is what it is and what you designed it to be in and of itself. However, there is a serious downside to this independence: we cannot be sure a priori whether what we have instructed R to do is what we had in mind for it to do. In natural conversation, the hearer's uptake will provide clear cues whether we have made ourselves understood in the way we intended to. In R, if the code is intact ('well-formed') and there are hence no warnings and no error, there is still no guarantee that it will actually produce what you had in mind for it to produce. You might have made a minute mistake somewhere in the run-up to the eventual code – a faulty retrieval of data points, a misguided transformation, a wrong match-up of data points, etc.

Once you start programming graphics in R you will discover that the code for them is actually quite trivial: in most cases, simple graph types require not even a handful of lines of code. The devil is in the run-up, that is, in how you prepare your data in such a way that it can be visualized in the graph type of your choice. This run-up may be relatively smooth or, perhaps more often than not, bumpy and longwinded, involving many steps and stops in between the initial data read-in and the graphical output.

1.4 Aims of this book

As part of their course work, graduate and undergraduate students of linguistics and related fields are generally encouraged to carry out small-scale original research. Today the opportunities to carry out such original research have grown exponentially. Students have vast amounts of authentic data literally at their fingertips, be it, for example, as raw web data or data they can source from online corpora. While the data resources have grown immensely, it is a common observation that the students' skills at processing these amounts of data have not kept apace. Not only do students generally not know how to perform significance tests on data, but their abilities of describing data are severely limited too. All students often know is how to make a bar plot or a line plot using Excel or any other commercial spreadsheet software. That's a serious shortcoming as visualizing data in research is "important, in fact indispensable" (Gries 2009: 146). It is important for three reasons. First, it "allows the viewer to process complex information in an intuitive way" (Hilpert 2011: 435). Second, it "will reveal any peculiarities of the data that will shape further analysis" (Johnson 2013: 288). And third, some visualizations are not merely a preparatory step *prior* to analysis; rather, the interpretation of the graphic *is* the analysis.

The first aim, then, in this book is to guide university students of linguistics in using R to visualize data in a variety of formats. While the main focus is clearly on visualization formats for descriptive statistics, some common inferential tests will also be introduced (cf. Chapters 10 and 12). The book also serves as a *general* introduction to R. This is because, as noted earlier, meaningful graphics ride, as it were, on the tip of an iceberg of code piled up to stream-line the data available into a format that can be processed by the graphic. So to arrive at graphics in R in most cases a significant amount of *data management* is required. How to manage data is dealt with in detail, not only in Chapter 2, which tackles data management directly, but also throughout Chapters 4–12 in the sections dedicated to explaining the code underlying the central graphic in the case study (see below) for each chapter.

The second aim is for the book to serve as an introduction to original quantitative research in Interactional Linguistics based on corpora. Chapters 4–12 each contain an original case study. They are centred around the two fundamental structures of conversational organization: the turn and the sequence. The case studies are ordered in such a way as to mirror a speaker's linear progression from turn inception (Chapter 4) to turn execution (Chapters 5 through 7) to turn completion (Chapter 11) and turn transition (Chapter 12). In Chapters 8 through 10 the focus of attention is the sequence, specifically the storytelling sequence.

There are, of course, limits to how one thing can effectively do two things at the same time, and these limits make sacrifices necessary. The sacrifices on the R side concern not only the huge realm of inferential statistics, which will be dealt with only passingly. The book will also focus on the most common visualization types and leave out discussion of less widely-used ones. Finally, at the expense of ggplot2 (check out `?ggplot2` at the R console), a package that claims to provide 'elegant' visualizations and has gained some traction recently, the book will concentrate largely, but not exclusively, on the visualizations facilitated by base R; as the book will show, base R graphics need not be aesthetically less pleasing. Also, graphics in base R are an appropriate starting point in a book geared toward novices to R. However, ggplot2 is an important development that should not be omitted entirely, so it will be introduced in Chapter 8 when we discuss the code underlying the violin plot – a plot type that can hardly be programmed in base R but is nicely implemented in ggplot2. As to the sacrifices made on the side of the case studies, the descriptions are governed by the principle of brevity. That is, introductory sections may, at times, provide the research background rather crudely and discussion sections may be restricted to discussing only the most immediate results in rather broad terms and refrain from digging deeper into less immediate aspects and implications. Still, it is hoped, the lack of fine detail will be made up for by the novelty of the research qestions asked.

Finally, the two aims of this book – teaching how to visualize data in R and presenting quantitative research in Interactional Linguistics – are not pursued separately, side by side, but in close integration with one another: each chapter introduces a new research topic *and* a new graphical format.

1.5 Chapter structure

The book is divided into 13 chapters. The present introductory chapter, Chapter 1, is followed by an introduction to the basics of data management in R in Chapter 2. Chapter 3 introduces the most common graphical parameters, based on an illustrative case study. The remaining chapters, Chapters 4–12, each present a case study in a key area of interaction research as well as a new graph type. These chapters also share the same three-fold structure:

- Case study:
  the first section presents a case study in which the graphic plays a central role.
- The graphic in the case study:
  this section explains how the plot was arrived at in the case study. The account is detailed and comprehensive to enable the reader not only to replicate the study but also to use the graph type in other research scenarios.
- Task:
  this section provides a practical task closely modeled on the case study.

The ultimate chapter, Chapter 13, aims to take stock of the lessons learned throughout the analytic chapters.

If you are a beginner in R, the following chapter, Chapter 2, is essential: it will help you make the first steps in R and get to know and understand basic functions and structures of R.

1.6 Installing R and R Studio

Installation of R cannot be easier: go to <https://cran.r-project.org> and download the latest version suitable for your operating system. You will be asked to also choose a CRAN (Comprehensive R Archive Network) mirror (CRAN is a network of servers around the world that store R code and documentation); choose the mirror nearest to you to minimize network load.

The initial download of R will also install a number of 'packages'; a package contains pre-defined functions, data, documentation, and tests, and is easy to share with others. At the time of writing there were thousands of additional packages for R available; to learn more, go to <http://r-pkgs.had.co.nz/intro.html>.

Once the install is complete, you could immediately start working with R. However, it is recommended that you also download an R editor. R editors allow you to store R code in files and they provide syntax highlighting and offer functionality for downloading packages of code. Many R editors are available; probably the most commonly used one is RStudio. For download go to <https://www.rstudio.com/products/rstudio/download/> and choose one of the freely downloadable versions. RStudio automatically connects to R so to start working in R it is sufficient to start RStudio.

1.7 How to get help with R

Once you start analyzing your own data, you will probably require some help at some point. Where to get help from?

There are a number of possibilities. In R itself, you can obtain help on specific functions or arguments for functions by using `?` followed by the function name. For example, `?mean` will open the R documentiation for this function to compute the arithmetic mean; alternatively, you can use `help(mean)`. If you don't know the function name, you can use two contiguous question marks followed by a keyword, for example `??arithmetic`, which will prompt R to open up a list of all functions available so you can choose the right one from it.

You can also search the internet by googling keywords followed by something like "in R". It is often amazing to see how many times your question, or questions similar to yours, have already been asked before, or that there exist useful video tutorials on more or less exactly your topic of interest!

Another source of assistance for specific queries are the dedicated online forums for R. Two such forums that specifically cater for *linguists* working with R are the Google groups CorpLingwithR at https://groups.google.com/forum/#!forum/corpling-with-r and StatForLingwithR at <https://groups.google.com/forum/#!forum/statforling-with-r> created and maintained by Stefan T. Gries, the author of immensely useful introductions for linguists to R (Gries 2009, 2017).

Finally, perhaps the most useful forum for R (and many other programming languages) is Stack Overflow at <https://stackoverflow.com/>. While the above-mentioned forums are relatively quiet, this forum buzzes with activity. The number of contributers, many of whom are professional programmers, is awe-inspiring: as of August 2018 it had more than 9 million registered users. The number of regular users specifically interested in R is also impressive: at the time of writing, the tag for R had more than 314,832 'watchers'! It is not unusual for queries posted to this forum to be answered, authoritatively, within minutes. A prerequisite for such a quick and effective response is that you post a question that can actually be answered; key is the provision of a reproducible example and a clear specification of the problem. Before posting queries to Stack Overflow, do check out this help page: <https://stackoverflow.com/questions/5963269/how-to-make-a-great-r-reproducible-example>.

Chapter 2

Data management in R

2.1 Introduction

In this chapter the aim is to lay the foundations to working with R by looking at how to manage data. Data management is an essential prerequisite for making graphics in that, for example, certain diagram types demand that the data be available in certain data structures. Also, the data you wish to plot are often not yet available in the initial data on hand but need to be gained through extracting and/or transforming available data. R offers near-endless ways of managing data. The techniques shown in this chapter represent a small cross-section only, selected with a view to enabling the reader to follow the code used for the case studies and to perform the tasks in this book.

2.2 Basics

The simplest use of R is as a calculator. Mathematical *operators* available in R include +, −, / for division, * for multiplication, and ^ for exponentiation.
In the following illustrative examples, use is made of the semi-colon and #: the semi-colon allows you to perform separate commands in one line, while # separates code to be processed by R (on the left of #) from comments that are not to be processed by R (on the right of #):

```
> 5 + 7; 5 - 7; 5/7; 5*7; 5^7
[1] 12          # addition
[1] -2          # subtraction
[1] 0.7142857   # division
[1] 35          # multiplication
[1] 78125       # exponentiation
```

A convenient way to 'shortcut' the definition of sequences is by using the colon between the start and end point of the sequence:

```
> 1:7
[1]1 2 3 4 5 6 7
```

It goes without saying that mathematical operations can be combined in R. Here, for example, we divide the sum of 5 + 7 by the sequence from 5 to 7 to the power of 7, thus producing three results (one for each number in the sequence):

```
> (5 + 7)/(5:7)^7
[1] 1.536000e-04 4.286694e-05 1.457119e-05
```

R has mathematical *functions* too. For example, `log()` takes the logarithm, `exp()` returns the exponential, and `sqrt()` gives the square root:

```
> log(5, 2); exp(2); sqrt(17)
[1] 2.321928   # logarithm of 5 to the base of 2
[1] 7.389056   # the exponential of 2
[1] 4.123106   # the square root of 17
```

2.3 Vectors

The beauty of R (and its infinite superiority to calculators) begins to show with vectors, the most fundamental data structure in R. A vector typically contains (or is interpreted by R as containing) same-type elements, for example, numbers or characters. To string the elements of the vector together the `c()` function is used; the vector's elements are separated by commas. Also, vectors are given names by using the assignment operator `<-`. If you have a contiguous range of numbers you can use the colon between the extreme values of the range. For example, the vectors `x` and `y` return the same elements:

```
> x <- c(1, 2, 3, 4, 9); y <- c(1:4, 9)
> x; y
[1] 1 2 3 4 9
[1] 1 2 3 4 9
```

If you perform operations on vectors, the operation is performed for every element in the vector. Here, for example, we define a sequence from 1 to 10 and compute the values of an exponential function for the sequence:

```
> x <- c(1:10) # define sequence
> 2^x          # 2 to the power of every element in the sequence
[1]    2    4    8    16    32    64    128    256    512    1024
```

As noted, vectors can contain character elements too. Let's take an attested utterance as illustration: "Ah well if not we 'll try the car", a nine-word utterance from the BNC that is part of the dataset underlying the analysis in Section 11.3 . The utterance can be rendered as a vector in two ways; (i) by defining the whole utterance as one element (`utt1`):

```
> utt1 <- "Ah well if not we 'll trythe car"
> utt1
[1] "Ah well if not we 'll try the car"
```

or (ii) by defining each word in the utterance as a separate element (`utt2`), in which case we need to concatenate the elements using `c()`:

```
> utt2 <- c("Ah", "well", "if", "not", "we", "'ll", "try", "the", "car")
> utt2
[1]"Ah" "well" "if" "not" "we" "'ll" "try" "the" "car"
```

The structural difference between `utt1` and `utt2` is by no means negligible, as can be seen from calling the function `length()`, which counts the number of elements in a vector:

```
> length(utt1); length(utt2)
[1] 1
[1] 9
```

The way elements in a vector are stringed together can be changed. If you want `utt1` to be divided into separate word elements, you can use the function `strsplit()` with the argument `split = " "` (i.e., you split the string on the white space between the words).

```
> strsplit(utt1, split = " ")
[[1]]
[1]"Ah" "well" "if" "not" "we" "'ll" "try" "the" "car"
```

Alternatively, if you have separate elements, as in `utt2`, but need the words as separate elements you can use the function `paste()` with its argument `collapse = " "` (i.e., with a space between the quote marks):[1]

```
> paste(utt1, collapse = " ")
[1] "Ah well if not we 'll try the car"
```

Often, however, data are not of the same kind, but mixed. For example, say, you have a vector z that combines a decimal number, a character element, and an integer number:

```
> z <- c(1.2, "ah", 5)
> z
[1]"1.2" "ah" "5"
```

How does R handle such a mixture? R will coerce the elements in the vector to the type that is easiest to coerce to. To find out what vector type R has coerced the elements to, use `typeof()`:

1. The output of this call, however, is a list, yet another data structure type (as indicated in the use of double square brackets in `[[1]]`; if you want a vector rather than a list, use the `unlist()` function, thus:

```
unlist(strsplit(utt1, split = " "))
[1] "Ah" "well" "if" "not" "we" "'ll" "try" "the" "car"
```

```
> typeof(z)
[1]"character"
```

So, R has coerced the elements to character elements! If you have a vector that contains mostly numerical elements, such a conversion does not make much sense; you will want a coercion toward numerical. This can be accomplished by `as.numeric()`:

```
> zz <- as.numeric(z)
Warning message:
NAs introduced by coercion
```

But this operation triggers a warning: "NAs introduced by coercion". NAs are 'not available' values. They are introduced in the conversion of `zz` to numeric because R cannot translate "ah" into a number – how should it? R 'solves' this problem by defining "ah" as a missing value:

```
> zz
[1]1.2  NA 5.0
```

`NA`s can be "a real source of irritation" (Crawley 2007: 14) and therefore require extra attention. The calculation of the mean, using the function `mean()`, or the median, using `median()`, are cases in point. For example, calling the two functions on the vector `zz` does not yield the expected output:

```
> mean(zz); median(zz)
[1] NA
[1] NA
```

`NA`s need to be removed from the calculation, which is achieved by the argument `na.rm = TRUE`:

```
> mean(zz, na.rm = TRUE); median(zz, na.rm = TRUE)
[1] 3.1
[1] 3.1
```

A vector type suitable for categorical data is factor. For example, the utterance "you did n't grumble about it did you this time" can be, and in linguistics often is, represented by the word tags assigned to each word: PNP VDD XX0 VVI PRP PNP VDD PNP DT0 NN1. Note these are the PoS tags used in the British National Corpus [BNC], from which most of the data in this volume have been taken. A complete description of the tagset is available on the website for the BNCweb, an interface for the BNC at http://bncweb.lancs.ac.uk/bncwebXML/Simple_query_language.pdf.

Suppose you have a vector with the tags as elements:

```
> tags <- c("PNP","VDD","XX0","VVI","PRP","PNP","VDD","PNP","DT0",
"NN1")
> tags
[1] "PNP" "VDD" "XX0" "VVI" "PRP" "PNP" "VDD" "PNP" "DT0" "NN1"
```

At this point, R does not 'know' that the tags represent categories; it just treats them as characters:

```
> typeof(tags)
[1] "character"
```

You can convert the elements of `tags` to factors with `as.factor()`:

```
> tags <- as.factor(tags)
> tags
[1] PNP VDD XX0 VVI PRP PNP VDD PNP DT0 NN1
Levels: DT0 NN1 PNP PRP VDD VVI XX0
```

Factors have 'levels' – that is, categories into which the elements fall; accordingly, the number of levels in `tags` is smaller than the number of tag tokens: 7 < 10. The same information is obtained if you query for the unique values of a vector, which is done by the function `unique()`:

```
> unique(tags)
[1] "PNP" "VDD" "XX0" "VVI" "PRP" "DT0" "NN1"
```

These are, then, the unique tags, or tag *types*. But surely you will also want to know in how many *tokens* the types each occur. This question can be answered by `table()`, an important function in linguistic research:

```
> table(tags)
tags
DT0 NN1 PNP PRP VDD VVI XX0
  1   1   3   1   2   1   1
```

R automatically orders the levels alphabetically. You can override that order and impose a different one, for example, by frequency. This is achieved by `sort()`:

```
> sort(table(tags))
tags
DT0 NN1 PRP VVI XX0 VDD PNP
  1   1   1   1   1   2   3
```

The default for `sort()` is to order in increasing order; if you want the most frequent levels to be shown first, use the argument `decreasing = TRUE` inside the function call:

```
> sort(table(tags), decreasing = TRUE)
tags
PNP VDD DT0 NN1 PRP VVI XX0
  3   2   1   1   1   1   1
```

2.4 Subsetting data

As will become clear in the case studies in this book, there are many circumstances where you will want to select only some elements of a vector; that is, you will want to work on a data *subset*. Subsetting is done by using what are called 'subscripts'. Subscripts are indices of the location of elements in a vector. They are defined in square brackets [ ]. For example, say, you want to identify the first element in `zz` and the third element in `tags`:

```
> zz[1]
[1] 1.2
> tags[3]
[1] XX0
Levels: DT0 NN1 PNP PRP VDD VVI XX0
```

Subscripts are also useful in defining *logical* subsets of data, that is, subsets that fullfil one (or more) logical criteria. Say, you are interested in only those values in `zz` that are greater than 3, you use the 'greater than' operator > (note that the vector name must be included in the square brackets too):

```
> zz[zz > 3]
[1] NA 5
```

Other logical operators include == (equal to), < (less than), <= (less than or equal to), >= (greater than or equal to), `&` (and), `|` (or), `!` (not), and `!=` (not equal); for a complete list see Crawley (2007: 27).

To illustrate logical NOT, let's say you want to define a subset of `zz` that does not include any `NA`s. You can do this by using `!` and the function `is.na()` (note the placement of `!` *before* `is.na()`):

```
> zz[!is.na(zz)]
[1] 1.2 5.0
```

Or you want to define a subset of `tags` that does not include any personal pronouns (tagged PNP):

```
> tags[tags! = "PNP"]
[1] VDD XX0 VVI PRP VDD DT0 NN1
Levels: DT0 NN1 PNP PRP VDD VVI XX0
```

Logical operators can be combined to define several conditions at once. For example, this code specifies a subset consisting of those values that are greater than 3 and that are not NA.

```
 > zz[zz > 3 & !is.na(zz)]
[1] 5
```

Or, in our vector tags, the following selects all tags except PRP or PNP:

```
> tags[!(tags == "PRP"|tags == "PNP")]
[1] VDD XX0 VVI VDD DT0 NN1
Levels: DT0 NN1 PNP PRP VDD VVI XX0
```

This last subset could be written much more elegantly using grep(), a function that searches for matches to a pattern. The pattern in the elements PRP and PNP – i.e., what they have in common – is obvious: the starting letter is capital P. While P can be matched by itself, the condition that P be the first letter in the string can be matched by the caret ^, a metacharacter used in regular expressions:

```
> tags[!grep("^P", tags)]
[1] VDD XX0 VVI VDD DT0 NN1
Levels: DT0 NN1 PNP PRP VDD VVI XX0
```

Regular expressions are extremely useful in finding patterns in language data or metalinguistic annotations and are therefore indispensable for linguistic research. A special section will introduce them in more detail below.

2.5 Dataframes and matrices

So far we have looked at single vectors. More often than not, however, linguists work with (much) larger data structures. Two such are introduced in this section: dataframes and matrices.

We'll begin with familiar data, the vector utt2, called here Turn:

```
> Turn <- c("Ah", "well", "if", "not", "we", "'ll", "try", "the", "car")
> Turn [1]
"Ah" "well" "if" "not" "we" "'ll" "try" "the" "car"
```

As linguists, we know, or can find out, a lot about these simple words. For example, we can identify the grammatical word class they belong to and assign tags to the words accordingly. Further, we can count the number of phonemes in each word and measure how long it took the speaker to articulate the words. Also, we can analyze the utterance, or turn, in terms of turn structure; that is, we can determine the words that relate back to the previous turn and thus belong to the pre-start component and those words in which the speaker performs their main action and which thus belong to the turn-constructional unit (TCU) (more on turn structure in Chapter 4). Rather than keeping these variables separate we will want to have them assembled in a single structure, a dataframe. So we define a dataframe `df` by calling the function `data.frame()` and defining the variables, this time using the simple equals sign =:

```
> df <- data.frame(
+ Turn = c("Ah", "well", "if", "not", "we", "'ll", "try", "the", "car"),
+ Tags = c("ITJ","AV0","CJS", "XX0", "PNP", "VM0", "VVI", "AT0", "NN1"),
+ Durations = c(0.175, 0.116, NA, NA, 0.110, NA, 0.184, 0.169, 0.225),
+ Structure = c(rep("pre", 2), rep("tcu", 7))
+ )
```

At first, it is useful to take a look at the dataframe. This can be done by simply typing the dataframe's name, which will produce the dataframe in the console, or by calling `View()`. This function will present the dataframe in a new window; if the dataframe is large, `View()` is quite handy in that it allows scrolling up and the down the dataframe and it delineates the cells in the dataframe, whereas the console display may get clipped and does not draw cell boundaries.

```
> df
   Turn  Tags  Durations  Structure
1    ah   ITJ      0.175        pre
2  well   AV0      0.116        pre
3    if   CJS         NA        tcu
4   not   XX0         NA        tcu
5    we   PNP      0.110        tcu
6   'll   VM0         NA        tcu
7   try   VVI      0.184        tcu
8   the   AT0      0.169        tcu
9   car   NN1      0.225        tcu
```

First, inspect your dataframe; specifically, check its structure: how is each variable defined? This check is done by `str()`:

```
> str(df)
'data.frame': 9 obs. of 4 variables:
$ Turn : Factor w/ 9 levels "'ll","Ah","car",..: 2 9 4 5 8 1 7 6 3
$ Tags : Factor w/ 9 levels "AT0","AV0","CJS",..: 4 2 3 9 6 7 8 1 5
$ Durations: num 0.175 0.116 NA NA 0.11 NA 0.184 0.169 0.225
$ Structure: Factor w/ 2 levels "pre","tcu": 1 1 2 2 2 2 2 2 2
```

The first line reveals that R has, correctly, identified `df` as a dataframe, with 9 observations in each column or variable. Note that the variable `Turn` is defined as a factor with 9 levels. You may remember that the vector `utt2`, which contained exactly the same words, was defined above as a character vector. Now that this data is part of a dataframe, R treats it as a factor, which is R's *default* structure for textual elements in dataframes. Note that the variables `Tags` and `Structure` are also factors. The variable `Durations,` on the other hand, is treated as numeric.

R's default transformation of textual elements to factors may not always be convenient. In this case, the data type can be reversed by `as.character();`note also that now that the variable `Turn` is part of a dataframe, calling `Turn` alone is insufficient. To address this variable, its location within `df` needs to be acknowledged; this is can be done by calling `df` followed by the dollar sign `$` followed by the variable name:

```
> df$Turn <- as.character(df$Turn)
```

Was the conversion successful? To find out call `typeof()`:

```
> typeof(df$Turn)
[1] "character"
```

Variables in dataframes can also be accessed via subscripts. For two-dimensional objects such as dataframes, there are subscripts for *rows* and subscripts for *columns* – in that order. For example, to select `Turn`, the variable stored in column #1, you can use:

```
> df[,1]
"Ah" "well" "if" "not" "we" "'ll" "try" "the" "car"
```

Note that the subscript for rows is missing, in which case R selects only the column specified after the comma. If you are interested in a specific cell, that is, an intersection of row and column, you need to specify both row and column. For example,

```
> df[2,3]
[1] 0.116
```

returns the value at the intersection of row #2 and column #3. If you are interested in a specific row only, you leave the space *behind* the comma blank. For example,

```
> df[3,]
  Turn Tags Durations Structure
3   if  CJS        NA       tcu
```

selects all elements in row #3 across all columns. To subset dataframes you can also apply logical conditions. For example, if you want a subset of the dataframe including all its columns but restricted to those words that fall into the turn's TCU, you select those rows in the dataframe that have the value `tcu` in the column `Structure`, which is column #4:

```
> df[df[,4] == "tcu",]
   Turn  Tags  Durations  Structure
3    if   CJS         NA        tcu
4   not   XX0         NA        tcu
5    we   PNP      0.110        tcu
6   'll   VM0         NA        tcu
7   try   VVI      0.184        tcu
8   the   AT0      0.169        tcu
9   car   NN1      0.225        tcu
```

Often, especially with larger data sets, you do not know the position of a variable in a dataframe, so using indices to address them is not convenient. An alternative route to addressing variables is by using their variable names. For example, the same subset as `df[df[,4]=="tcu",]` is obtained from `df[df$Structure=="tcu",]`.

To use another illustration of the utility of using variable names rather than variable indices to define subsets, say, you need to check which tag is used for the word "well", which is, after all, a highly multi-functional item that might easily be falsely tagged. To this end, you can call the variable `Tags` specifying that you are just interested in those `Tags` elements that, on the same row(s), have the value `"well"` in the `Turn` column:

```
> df$Tags[df$Turn=="well"]   # NB: no comma inside [] as the column is …
[1] AV0                      # … already selected in df$Tags
Levels: AT0 AV0 CJS ITJ NN1 PNP VM0 VVI XX0
```

We can see that 'well' is tagged as if it were an adverb, as in "well done" – clearly an erroneous tag, as 'well' is used in the turn as a pragmatic marker indicating some (problematic) relationship between the incipient turn and the preceding one. The inaccuracy can be fixed by assigning the correct tag, wrapped in quote marks, to the cell in question using the assignment operator:

```
> df$Tags[df$Turn=="well"] <- "ITJ"
```

If we now call `df`, this time for the sake of brevity just addressing the second row, we can see that R has indeed changed the element:

```
> df[2,]
  Turn Tags Durations Structure
2 well  ITJ     0.116       pre
```

Or, to provide another illustration of subsetting, a situation often arises where you find it desirable or necessary to exclude those rows from a dataframe that contain `NA`s. To obtain such an `NA`-free subset, use `!` to signal exclusion and `is.na()` to test for `NA`, and specify the column that contains the `NA`s, namely `Durations` in `df`:

```
> df[!is.na(df$Durations),]
   Turn  Tags  Durations  Structure
1    ah   ITJ      0.175        pre
2  well   ITJ      0.116        pre
5    we   PNP      0.110        tcu
7   try   VVI      0.184        tcu
8   the   AT0      0.169        tcu
9   car   NN1      0.225        tcu
```

If, for some reason, you decide that a variable/column is no longer needed, you can subset by using *negative* subscripts. For example, you can clip `df` by cutting off several columns at once:

```
> df[,-c(3:4)]
   Turn  Tags
1    ah   ITJ
2  well   ITJ
3    if   CJS
4   not   XX0
5    we   PNP
6   'll   VM0
7   try   VVI
8   the   AT0
9   car   NN1
```

Note that to make this clipping permanent you would have to store the clipped dataframe in a new vector, for example, thus:

```
> df_clipped <- df[,-c(3:4)]
```

Since we have not done this in the command `df[,-c(3:4)]` the dataframe `df` still contains all original variables.

As with atomic vectors, subsetting conditions can be compounded. Say, you want to select those rows where (i) `Durations` is greater than 0.15 and (ii) not `NA` and where (iii) the number of characters in `Turn` is 2; this latter information is provided by the function `nchar()`:

```
> df[df$Durations > 0.15 & !is.na(df$Durations) & nchar(df$Turn) == 2,]
  Turn  Tags  Durations  Structure
1   ah   ITJ      0.175        pre
```

A more complex scenario arises if you wish to subset on information that is not directly available in your dataframe. In Chapter 5, for example, we will be faced with the task of subsetting a dataframe on content words. Content words are normally made up of nouns, (full) verbs, adverbs, and adjectives (Biber et al. 1999). In our present dataframe df, information on these word classes is missing. However, we have the grammatical word class tags, and based on this information, content words can be identified. Let's check with the variable Turn in df: which of the nine words would be considered content words?

Regarding 'well', we already got rid of the misleading classification as AV0 for adverb and replaced it with ITJ; so 'well' is not a content word. The next candidate word is "'ll" tagged VM0 for modal verb. Modal verbs are a problematic boundary case (cf. Stubbs 2002: 40) in that on the one hand they express obligation, volition, probability, etc. – ie. 'content' – but on the other hand lack the formal characteristic of being open classes that characterize prototypical content word classes; for this illustrative task, we will include "'ll" as a content word. Finally, there are the 'classic' cases of lexical verbs, like "try" tagged VVI, and nouns, like "car" tagged NN1, that are more readily recognizable as content words. Based on this manual analysis, we can define a new variable Content for our dataframe df:

```
> df$Content <- c("no", "no", "no", "no", "no", "yes", "yes", "no", "yes")
```

Once we call df, or view it in View(df), we see that R has added the variable Content to df:

```
> df
  Turn  Tags  Durations  Structure  Content
1   ah   ITJ      0.175        pre       no
2 well   ITJ      0.116        pre       no
3   if   CJS         NA        tcu       no
4  not   XX0         NA        tcu       no
5   we   PNP      0.110        tcu       no
6  'll   VM0         NA        tcu      yes
7  try   VVI      0.184        tcu      yes
8  the   AT0      0.169        tcu       no
9  car   NN1      0.225        tcu      yes
```

Now we are in a position to subset df on content words:

```
> df[df$Content=="yes",]
   Turn  Tags  Durations  Structure  Content
6   'll   VM0         NA        tcu      yes
7   try   VVI      0.184        tcu      yes
9   car   NN1      0.225        tcu      yes
```

We have done this little analysis manually because manual coding is feasible with nine data points to analyze. But what about re-analyzing data sets that are larger, with observations counting in the thousands or more? There manual re-analysis is clearly no option. What else can be done? The answer is using regular expressions, a.k.a regex. For illustration, let's use regex for df to achieve the same re-analysis *automatically*.

In df, we know that the only content words are two verbs and a noun; there are no adjective and adverb tokens. For the sake of providing an illustration that is more widely applicable, let's assume a situation where we don't know which content word classes are attested in the data – if you have large amounts of data, this is a realistic assumption. So we need to set up a regex that caters for all four major classes: nouns, adjectives, adverbs, and lexical verbs (but not the primary verbs BE, HAVE, and DO, which are typically auxiliary verbs and hence function words); for consistency, we will also include modal verbs.

The first thing to determine are the patterns. These can only be discerned by looking at the tagset used.

Let's start with nouns: in the BNC, nouns can be tagged NN0 (noncountable common noun, as in "information"), NN1 (singular common noun, as in "car"), NN2 (plural common noun, as in "cars"), and NP0 (proper noun, as in "Sue"). What do these tags have in common? The first letter is invariably a capital N.

Lexical verb tags include VVI (infinitive, as in "go"), VVB (base, as in "go"), VVZ (3rd person singular present tense, as in "goes"), VVG (progressive, as in "going"), VVD (past tense, as in "went"), and finally VVN (past participle, as in "gone"). Here the pattern is clear-cut too: the tags all share the double "VV". This pattern is not shared by the tag for modal verbs VM0, where the second letter is "M".

For adjectives, the BNC tagset provides three tags: AJ0 (positive, as in "good"), AJC (comparative, as in "better"), and AJS (superlative, as in "best"). Thus, adjective tags all start with AJ. Similarly, adverbs can be AV0 (general, as in "often"), AVP (particle, as in "up"), and AVQ (wh-question words, as in "why"). Thus, they share AJ.

The combinations VV, AJ, and AV are unique for lexical verbs, adjectives, and adverbs, respectively; that is, they only occur in tags designating these classes; the letter N, by contrast, which is shared by all noun tags is not unique as it also occurs, for example, in VVN, the tag used for the past participle as in "taken". So to match nouns, the *position* of N at the beginning of the tag string needs to be incorporated into the regex; this is achieved by using the caret ^ in front of N.

Now we are almost ready to define the new variable, which we will call Content_rex; we also know that the function grepl(), which we already encountered earlier, will be necessary for this task. You may also remember that the syntax of grepl() is *grepl(pattern, variable)*. We can define the pattern as a vector:

```
> pattern <- "^N|VV|VM|AJ|AV"
```

Let's investigate the syntax of this pattern.[2] First, as noted, we used the caret ^ to signify that the pattern to match here is an N at the beginning of a string. Second, N is

2. The pattern can be refined, as there is repetition in VV and VM as well as AJ and AV. To get rid of the repetition, two more metacharacters can be used, namely (and):

```
> pattern <- "^N|V(V|M)|A(J|V)"
```

separated from `VV` by `|`, another metacharacter whose meaning is "or". We find this metacharacter three more times: once between `VV` and `VM`, `VM` and `AJ`, and `AJ` and `AV` – that is, between all alternative patterns.[3]

We can check whether the pattern matches the right tags by calling `grepl()`, which evaluates to true or false:

```
> grepl(pattern, df$Tags)
FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE TRUE
```

Indeed, the pattern is true for the expected words, namely the modal verb, the full verb, and the noun. This can more easily be seen if we call, not `grepl()`, but `grep()` (with out the 'l' for 'logical') and setting the parameter `value = T` (`T` is shorthand for `TRUE` and will be understood by R in either form):

```
> grep(pattern, df$Tags, value = T)
[1] "VM0" "VVI" "NN1"
```

So, now all that is missing to define the data in the new column is `ifelse()`, an immensely useful function that can be used if "you want to do one thing if a condition is true and another thing if the condition is false" (Crawley 2007: 62). In the context of the present task, the 'thing we want to do' is assign the label "yes" if a word is a content word; to all remaining words, which do not meet this condition, we want to assign the label "no". The syntax of `ifelse()` is *ifelse(condition, true, false)*: that is, the first argument specifies the condition, the second determines the action to be taken if the condition is true, the third states what is to be done if the condition is false. In setting up the code, we can re-use the vector `pattern`:[4]

```
> df$Content_rex <- ifelse(grepl(pattern, df$Tags),"yes","no")
> df
  Turn  Tags Durations  Structure Content  Content_rex
1   ah   ITJ     0.175        pre      no           no
2 well   ITJ     0.116        pre      no           no
3   if   CJS        NA        tcu      no           no
4  not   XX0        NA        tcu      no           no
5   we   PNP     0.110        tcu      no           no
6  'll   VM0        NA        tcu     yes          yes
```

3. The reader may wonder why adverbs and adjectives are not simply matched by `^A` since the two tags start with the letter A. Granted, `^A` would match adverbs and (positive) adjectives but it would also match the tag AT0, used for articles such as "a" and "the" – definitely not content words!

4. If a one-liner is preferred, the pattern can also be defined inside the whole expression:

```
df$Content_regex <- ifelse(grepl("^N|V(V|M)|A(J|V)",df$Tags),"yes","no")
```

```
7  try   VVI      0.184        tcu     yes          yes
8  the   AT0      0.169        tcu      no           no
9  car   NN1      0.225        tcu     yes          yes
```

We see from the output that `Content_rex` contains exactly the same elements as `Content`: That is, our automatic re-analysis of the information in `Tags` has been successful.

You may ask at this point what to do in a situation where there is not just one condition to satisfy but two or more conditions: can `ifelse()` be helpful too? The answer is yes because you can *nest* as many `ifelse()` statements as you want, thus defining processing instructions for as many conditions as you have. For example, to stick to our illustrative example, what we have done so far – namely distinguish content words from non-content words – is rather crude. A classification that takes into account more classes would make better linguistic sense. Such a more comprehensive classification is Biber et al.'s (1999) tripartite classification of major word classes; this classification includes not only content words, but also function words, and most notably, inserts. The three major word classes are described in more detail in the case study in Chapter 5. For present purposes, suffice it to say that inserts are, by and large, those words that in the BNC tagset are classified as interjections (ITJ); the only interjection that is both a prototypical insert and highly frequent in conversation but not recognized by the tagset is 'well' used as a pragmatic marker (see above). In other words, if we wish to categorize words according to Biber et al.'s (1999) three major word classes, we will need to identify content words, function words, and interjections. To discover content words in `Turn`, we can use the above regex; interjections are identifiable via their tag ITJ; and to determine function words, we will use the 'else' part of the `ifelse()` statement: anything that does not satisfy the two criteria for content words and interjections will be coded as a function word. This brings us to the following code:

```
> df$Major <- ifelse(grepl(pattern, df$Tags),"content",
+ ifelse(df$Tags=="ITJ", "insert", "function"))
> df
  Turn  Tags  Durations Structure  Content  Content_rex     Major
1   ah   ITJ      0.175       pre       no           no    insert
2 well   ITJ      0.116       pre       no           no    insert
3   if   CJS         NA       tcu       no           no  function
4  not   XX0         NA       tcu       no           no  function
5   we   PNP      0.110       tcu       no           no  function
6  'll   VM0         NA       tcu      yes          yes   content
7  try   VVI      0.184       tcu      yes          yes   content
8  the   AT0      0.169       tcu       no           no  function
9  car   NN1      0.225       tcu      yes          yes   content
```

It is not uncommon that a research projects requires data from more than a single dataframe. In the following we will look at functions that are key in managing 'traffic' between two dataframes.

For illustration we will re-use `df` but, to enhance legibility, clip it to the first three columns:

```
> df1 <- df[,c(1:3)]
> df1
  Turn  Tags  Durations
1   ah   ITJ      0.175
2 well   ITJ      0.116
3   if   CJS         NA
4  not   XX0         NA
5   we   PNP      0.110
6  'll   VM0         NA
7  try   VVI      0.184
8  the   AT0      0.169
9  car   NN1      0.225
```

The second dataframe, `df2`, contains some information on another 9-word turn and has exactly the same internal structure as `df1`:

```
> df2 <- data.frame(
+ Turn = c("no", "oh", "well", "let's", "hope", "he", "'ll", "get",
           "better"),
+ Tags = c("ITJ", "ITJ", "ITJ", "VM0", "VVI", "PNP", "VM0", "VVI",
           "AV0"),
+ Durations = c(0.210, 0.175, 0.123, 0.238, 0.194, 0.053, 0.069,
                0.170, 0.460)
+ )

> df2
    Turn  Tags Durations
1     no   ITJ     0.210
2     oh   ITJ     0.175
3   well   ITJ     0.123
4  let's   VM0     0.238
5   hope   VVI     0.194
6     he   PNP     0.053
7    'll   VM0     0.069
8    get   VVI     0.170
9 better   AV0     0.460
```

The first function to be introduced is `cbind()`, which binds dataframes column-wise together, i.e., side by side. The function takes as arguments simply the names of the dataframes:

```
> df3 <- cbind(df1, df2)
> df3
  Turn  Tags Durations    Turn  Tags Durations
1   ah   ITJ     0.175      no   ITJ     0.210
2 well   ITJ     0.116      oh   ITJ     0.175
3   if   CJS        NA    well   ITJ     0.123
4  not   XX0        NA   let's   VM0     0.238
5   we   PNP     0.110    hope   VVI     0.194
6  'll   VM0        NA      he   PNP     0.053
7  try   VVI     0.184     'll   VM0     0.069
8  the   AT0     0.169     get   VVI     0.170
9  car   NN1     0.225  better   AV0     0.460
```

While the join has worked out nicely, what is problematic in the new dataframe `df3` is that we now have identical column names, which will cause problems in addressing the columns. To work around this, we can change the column names using the functions `colnames()`, `rep()`,[5] and `paste()`; the latter function takes as many objects as you want to concatenate – in the present case, the column names and an index number to distinguish identical names – as well as the argument `sep`, which defines how the terms are to be joined; we will set it to `sep = ""`, meaning that we attach the index number *directly* to the column names:

```
> colnames(df3) <- paste(colnames(df3), c(rep(1,3), rep(2,3)), sep
= "")
```

Now the column names are distinct:

```
> colnames(df3)
[1] "Turn1" "Tags1" "Durations1" "Turn2" "Tags2" "Durations2"
```

A related (and in the present connection more suitable) way of joining two dataframes is `rbind()`. It row-binds (rather than column-binds) dataframes provided they have the same number of columns and the same column names (if they don't, `rbind()` will not work):

5. The function `rep()` takes two arguments: one for the object to be repeated, and another for how often it is to be repeated.

```
> df3 <- rbind(df1, df2)
> df3
      Turn  Tags Durations
1       ah   ITJ     0.175
2     well   ITJ     0.116
3       if   CJS        NA
4      not   XX0        NA
5       we   PNP     0.110
6      'll   VM0        NA
7      try   VVI     0.184
8      the   AT0     0.169
9      car   NN1     0.225
10      no   ITJ     0.210
11      oh   ITJ     0.175
12    well   ITJ     0.123
13   let's   VM0     0.238
14    hope   VVI     0.194
15      he   PNP     0.053
16     'll   VM0     0.069
17     get   VVI     0.170
18  better   AV0     0.460
```

The most important function to manage data traffic is arguably `match()`. This function "answers the question 'Where do the values in the second vector appear in the first vector'" (Crawley 2007: 47). To make clear what this means, let's start with a very simple example; we have two vectors, `a` and `b`:

```
> a <- c(LETTERS[1:5], "A", LETTERS[1:5])
> a
[1] "A" "B" "C" "D" "E" "A" "A" "B" "C" "D" "E"
> b <- c(LETTERS[1:2])
> b [1] "A" "B"
```

If we now call `match()` taking `a` and `b` as arguments, we get:

```
> match(a, b)
[1] 1 2 NA NA NA 1 1 2 NA NA NA
```

That is, `match()` creates a vector containing index numbers for the values in b (`1` corrresponds to `"A"` and `2` to `"B"`) and marks non-matches by `NA`: for example, there is no match in `b` for `"C"`, `"D"`, and `"E"` in `a`. How is this useful for managing larger and multiple dataframes?

The usefulness of `match()` lies in its capacity to facilitate the transfer of values from one dataframe to another dataframe based on *matching conditions*. For illustration, let's return to dataframe `df3`. The data assembled there is just a minute snippet of the data underlying the case study on speech rate reported in Chapter 7. A critical factor in the analysis of speech rate is the phonetic size of words, that is, the number of phonemes making up the words. This is because if you compare the durations of words, the variation you encounter will be heavily influenced by the words' physical lengths: a noun with seven or more phonemes will inevitably take longer to articulate than a function word with one or two phonemes.

Now, to illustrate the usefulness of `match()`, let's assume that you have a list of all words occurring in a dataset as well as the words' phonetic transcription. This list is stored in a (large) dataframe called `Phon`; its first six rows are shown by calling the function `head()`:

```
> head(Phon)
     Word  Transcription
1   admit         ədˈmɪt
2  aerial        ˈeərɪəl
3 aerobic      eəˈrəʊbɪk
4   after         ˈɑːftə
5     age            eɪʤ
6     ago          əˈgəʊ
```

Now what you want to do is transfer the transcription from `Phon` to `df3` in such a way that a word's transcription is put not just anywhere in `df3` but side-by-side that word, i.e., in the same row. To accomplish this, you define a new variable `df3$Trans` and assign values from `Phon$Transcription` to it *if* the values in `df$Turn` (the column with the words) and `Phon$Words match`:[6]

```
df3$Trans <- Phon$Transcription[match(df3$Turn, Phon$Word)]
```

If you now call `df3` you can see that the match has been successful:

6. Order matters in that `match()` assigns values from the second vector to the first vector (i.e., from right to left). For example, a different order may result in an error:

```
> df3$Trans <- Phonemes$Transcription[match(Phonemes$Word, df3$Turn)]
Error in `$<-.data.frame`(`*tmp*`, Trans, value = c(NA, NA, NA, NA, NA, :
replacement has 742 rows, data has 18
```

```
> df3
     Turn  Tags  Durations  Trans
1      ah   ITJ      0.175     ɑː
2    well   ITJ      0.116    wɛl
3      if   CJS         NA     ɪf
4     not   XX0         NA    nɒt
5      we   PNP      0.110    wiː
6     'll   VM0         NA      l
7     try   VVI      0.184   traɪ
8     the   AT0      0.169    ðiː
9     car   NN1      0.225    kɑː
10     no   ITJ      0.210    nəʊ
11     oh   ITJ      0.175     əʊ
12   well   ITJ      0.123    wɛl
13  let's   VM0      0.238   lɛts
14   hope   VVI      0.194   həʊp
15     he   PNP      0.053    hiː
16    'll   VM0      0.069      l
17    get   VVI      0.170    gɛt
18 better   AV0      0.460  ˈbɛtə
```

Now let’s turn to matrices, another core data structure. Unlike dataframes, which, as we have seen in the above examples, can accomodate variables of different types (e.g., numeric and character), matrices contain same-type data; that is, all data assembled in them should be either numeric, or integer, or character.

We can create a matrix `mtx` from scratch by defining the number of rows and columns thus:

```
> mtx <- matrix(NA, nrow = 8, ncol = 4)
> mtx
     [,1]  [,2] [,3] [,4]
[1,]   NA    NA   NA   NA
[2,]   NA    NA   NA   NA
[3,]   NA    NA   NA   NA
[4,]   NA    NA   NA   NA
[5,]   NA    NA   NA   NA
[6,]   NA    NA   NA   NA
[7,]   NA    NA   NA   NA
[8,]   NA    NA   NA   NA
```

You can check that the structure is indeed a matrix by using `class()`:

```
> class(mtx)
[1] "matrix"
```

The matrix above does not yet contain values. To fill the matrix we will use random deviates from the normal distribution generated by `rnorm()`[7] and we will use a `for` loop. In a `for` loop "you request that an index, *i*, takes on a sequence of values, and that one or more lines of command are executed as many times as there are different values in *i*" (Crawley 2007: 58). For example, if you define the sequence from 1 to the total number of rows in `mtx` as your index *i*, you can request that the command line inside the curly brackets `{ }` is executed anew for each row in `mtx`:

```
for(i in 1:nrow(mtx)){
   mtx[i,] <- rnorm(4, 1, 0.5)
   }
```

The result of the loop is this matrix (note that each time you execute the `for` loop `mtx` will contain slightly different numbers as `rnorm()`, like for example `sample()`, generates random numbers *ad hoc*! To avoid that variation, you can use the function `set.seed()`, and input random numbers such as `set.seed(123)`; this will ensure that the same numbers are generated upon each call to `rnorm()` and even across different computers):

```
> mtx
          [,1]      [,2]      [,3]       [,4]
[1,] 0.3185526 1.1064500 1.1118180 0.99079679
[2,] 0.1673188 1.9577960 0.6519454 0.87897832
[3,] 1.1265918 0.3630504 0.4120163 0.20233971
[4,] 0.6035189 0.9400921 1.1670354 1.33095974
[5,] 1.3572709 0.8692920 0.5234354 0.78606335
[6,] 0.4265050 1.4344361 0.8462145 1.13144930
[7,] 0.9547759 1.1859968 1.0068117 0.05911865
[8,] 1.8459110 1.5804917 1.2126679 0.69621761
```

Helpful functions to analyze data in matrices (and dataframes) include the functions of the `apply()` family of functions (including not only `apply()` but also `lapply()`, `tapply()` and others; see, for example, `?lapply`). For illustration, we will take a look at `apply()`, which applies a function to margins of a matrix.

Its syntax is simple: the first argument is the data, the second is either `1` (indicating that the function is to be applied row-wise) or `2` (indicating column-wise application of the function), and the third is the function itself. R provides a large number of built-in vector functions, including `sum()` for total, `min()` for minimum value, `mean()` for arithmetic average, `sort()` for sorting and many others (cf. Crawley 2007: 17–18). Here we use `max()`, which gives the maximum value, and `median()`, which computes

7. `rnorm()` requires three arguments: the number of observations (4 in the above example), the mean (here 1), and the standard deviation (here 0.5).

the median, the middle value of a sorted distribution and hence provides a more robust measure of central tendency than the mean (cf. Woods et al. 1986: 30–32).

```
> apply(mtx, 1, max)       # maximum by rows
[1]  1.111818  1.957796  1.126592  1.330960  1.357271  1.434436
1.185997  1.845911

> apply(mtx, 2, median)    # median by columns
[1]  0.7791474  1.1462234  0.9265131  0.8325208
```

Matrices can also contain characters. For illustration, let's fill `mtx` with letters from A to Z, using the built-in constant `LETTERS` , which contains the 26 upper-case letters of the Roman alphabet (`letters` is the equivalent for lower-case letters). To fill it with a *random sample* of letters we use `sample()`. Its arguments include the data to be sampled from and the number of draws; if the number of draws is greater than the number of values in the data, then the third argument `replace = T` allows for sampling *with replacement*.

We can do it in one line or with a `for` loop; note that the one-liner requires that the draw is executed for the whole length of the matrix, which is 32 (as there are 32 cells in the matrix) whereas the `for` loop – which iterates over the rows, each of which has 4 values – requires just four draws:

```
> mtx[] <- sample(LETTERS[1:3], length(mtx), replace = T)
> mtx
     [,1] [,2] [,3] [,4]
[1,] "B"  "A"  "B"  "C"
[2,] "A"  "B"  "C"  "B"
[3,] "B"  "B"  "A"  "C"
[4,] "C"  "B"  "B"  "C"
[5,] "A"  "B"  "A"  "A"
[6,] "B"  "B"  "B"  "A"
[7,] "A"  "B"  "A"  "C"
[8,] "A"  "A"  "B"  "B"

> for(i in 1:nrow(mtx)){
mtx[i,] <- sample(LETTERS[1:3], 4, replace = T)
}
> mtx
    [,1] [,2] [,3] [,4]
[1,]"A"  "A"  "C"  "B"
[2,]"A"  "A"  "B"  "B"
[3,]"B"  "B"  "C"  "C"
[4,]"A"  "B"  "A"  "B"
[5,]"C"  "B"  "B"  "A"
```

```
[6,] "A"  "C"  "C"  "B"
[7,] "B"  "B"  "B"  "B"
[8,] "C"  "A"  "B"  "C"
```

The example is so far not particularly meaningful in linguistic terms. To increase its meaningfulness, let's assume that the data in the matrix are words occurring in turns and that we have a matrix not of letters but of codes denoting the major word class of the words in the turns. We would then first label the matrix rows and columns accordingly, namely `w1`, `w2`, `w3`, and `w4` for columns and `Turn_1`, `Turn_2`, etc. for rows; the labels are assigned using `rownames()` and `colnames()`:

```
> rownames(mtx) <- c(paste("Turn", 1:8, sep = "_"))
> colnames(mtx) <- c("w1", "w2", "w3", "w4")
> mtx
       w1  w2  w3  w4
Turn_1 "A" "A" "C" "B"
Turn_2 "A" "A" "B" "B"
Turn_3 "B" "B" "C" "C"
Turn_4 "A" "B" "A" "B"
Turn_5 "C" "B" "B" "A"
Turn_6 "A" "C" "C" "B"
Turn_7 "B" "B" "B" "B"
Turn_8 "C" "A" "B" "C"
```

Now we define a vector `wordclass` that contains the labels for the three major word classes:

```
> wordclass <- c("Function", "Content", "Insert")
```

and sample these labels into the matrix:

```
> mtx[] <- sample(wordclass, length(mtx), replace = T)
> mtx
     [,1]       [,2]       [,3]       [,4]
[1,] "Function" "Function" "Insert"   "Function"
[2,] "Insert"   "Function" "Insert"   "Content"
[3,] "Function" "Content"  "Insert"   "Function"
[4,] "Content"  "Insert"   "Content"  "Insert"
[5,] "Content"  "Content"  "Function" "Function"
[6,] "Function" "Content"  "Function" "Function"
[7,] "Function" "Function" "Function" "Content"
[8,] "Content"  "Content"  "Content"  "Content"
```

This looks a little more linguistic-y. We can perform many useful analyses with this kind of data. For example, we may be interested in the frequencies of occurrence of the different word classes and in the percentages the classes account for. For the matrix as a whole, frequencies can be obtained from `table()`, and percentages from `prop.table(table())`:[8]

```
> frequencies <- table(mtx)
> frequencies
mtx
Content Function Insert
     12       14      6
```

```
> proportions <- prop.table(table(mtx))
> proportions
mtx
Content Function Insert
 0.3750   0.4375 0.1875
```

As will be shown in the case study in Chapter 5, frequencies and proportions of word classes are not distributed uniformly across positions in turns. Therefore it will be desirable to compute frequencies and proportions for each turn position, that is, each column in `mtx`. We can easily compute frequencies and proportions for the matrix columns separately. For example, using `prop.table()` and subsetting on the columns in `mtx` we get the proportions of the word classes in the first column, i.e., for the turn-initial words:

```
> prop.table(table(mtx[,1]))
Content Function Insert
  0.375    0.500  0.125
```

But what if, as in the present case, we have multiple columns, perhaps even many more than the four we presently have? Then it would violate the above mentioned economy principle if we repeated the code multiple times (and repeating code is not only cumbersome but also error-prone!). How can we get the desired output for all columns in *one* piece of code? For illustration, let's use a `for` loop (but note that there are shorter ways to do it)[9] and let's focus on proportions (the respective code for frequencies is essentially the same, just without `prop.table()`).

First we will set up a new matrix, `prop`, to feed the proportions in. What should that new matrix look like? We know that the source matrix, `mtx`, has four columns, each representing a word slot in the turn; so we will set the number of columns in

8. `prop.table()` cannot process character data, which we have in `mtx`; therefore `prop.table()` needs to be wrapped around `table()`.

9. For example, using `apply()` and `factor()` (in case not all three factor levels are represented in each column) which gets exactly the same results as the for loop:

```
apply(mtx, 2, function(x) prop.table(table(factor(x, levels = wordclass))))
```

prop to 4 too. We further know that each column in mtx has three unique values (Content, Function, and Insert); so we will set the number of rows in prop to 3. Finally, we label the rows and columns in the matrix:

```
> prop <- matrix(NA, ncol = 4, nrow = 3)
> colnames(prop) <- c("w1", "w2", "w3", "w4")
> rownames(prop) <- c("Content", "Function", "Insert")
> prop
         w1 w2  w3 w4
Content  NA NA  NA NA
Function NA NA  NA NA
Insert   NA NA  NA NA
```

This was the easy part. The slightly harder part is defining a for loop to fill in the proportions in one go. The first task is to define the index i: what we want to achieve is fill the four columns in prop. So what we could do is one of two things: we could either, quite simply, define i as the sequence 1:4, or define it as 1 to as many columns as there are to be filled, which we would express in 1:ncol(prop); obviously, the latter version is more flexible and will accomodate any number of columns. Once we have, then, set up the for clause in for(i in 1:ncol(prop)) we open curly brackets { }. Inside the brackets we formulate the command that we instruct R to execute as many times as there are values in i. We subset prop on its columns by the index i and instruct R to loop over the columns in mtx, also subsetting it on its columns with i, and to compute the proportions of the three word classes using prop.table(table()), extract them and assign them to prop[,i]:

```
> for(i in 1:ncol(prop)){
+     prop[,i] <- prop.table(table(mtx[,i]))
+ }
```

And this is the result – a neat little matrix in tabular form. If the matrix were much larger and contained real language data, it could serve as the starting point to a quantitative investigation of how word classes distribute across positions in four-word turns:

```
> prop
            w1    w2    w3    w4
Content  0.375 0.500 0.250 0.375
Function 0.500 0.375 0.375 0.500
Insert   0.125 0.125 0.375 0.125
```

2.6 Regular expressions

We already touched upon regular expression above. In this section, the aim is to provide a brief but more systematic introduction.

Regular expression deserves good mention in a book that is dedicated to research in Interactional Linguistics (and Conversation Analysis). This is because this language could greatly benefit analyses of CA transcriptions. Typically, CA transcripts are formatted as Word documents, a format which is only minimally machine-readable. For instance, searches for literal words are enabled but searches for more abstract patterns are impossible; also search results cannot be extracted. Unlike orthographic transcripts, CA transcripts encode a wealth of information related not only to the verbal content of face-to-face interaction but also to delivery aspects, including, for instance, tempo, emphasis, prosody, temporal aspects such as the length of pauses, sequential aspects such as overlap, and aspects of non-verbal and non-vocal conduct such as meaningful gestures, facial expressions, body posture etc. Certain widely accepted standards govern how a CA transcript encodes such aspects. For example, overlap is marked by square brackets [ ], intonational contours are denoted by punctuation marks, and changes in tempo are signalled by the 'greater than' and 'smaller than' symbols > and < . In other words: a wealth of multimodal information is encoded in a wealth of character symbols. Now, character symbols are the very stuff of regular expression: this language was invented to navigate through texts and extract patterns of character strings from them. In other words, regex makes CA transcripts machine-readable and can thus open up new ways of analyzing CA transcripts.

Regular expression is a language in its own right. Its purpose is to parse, extract, and manipulate text (Stubblebine 2007). It is implemented in many programming languages such as Perl, Java, Ruby, C+, Python, and of course R, where a number of extremely useful and important functions are used to implement the language.[10]

As any language, regular expression (often abbreviated to regex) requires a learning curve, and help will be appreciated especially at early stages. The first notable source of help is found in R itself, where the command `?regex` opens R's detailed documentation of how regular expression is implemented in R. Further, there are a large number of useful websites, including, for example, <https://www.regular-expressions.info/quickstart.html> and <http://stat.ethz.ch/R-manual/R-devel/library/base/html/regex.html> . Recommended are also Crawley (2007), who discusses regex in a very reader-friendly way, and Gries (2017), where regex is covered in very good detail and with practical tasks for readers. A very useful webpage is <https://jex.im/regulex/>, which *visualizes* regex in detail.

For illustration, we will work with the CA transcript of a storytelling interaction; let's call the transcript `CAt`:[11]

10. Although basically the same language, regex is implemented in subtly different ways across programming languages.

11. This storytelling is part of the storytelling sample underlying the case study in Chapter 10.

```
> CAt
    who                                                                   story
1    Sue:  That's like your grand:ma. did that with::=erm
2    <NA>  with Ju:ne (.) once or [ twice.]
3   Carl:                         [ Yeah. ]
4    Sue:  And June wanted to go out and yo- your granny said (0.8)
5    <NA>  "make sure you're ba(hh)ck before midni(hh)ght."
6    <NA>  [Mm.]
7    <NA>  [There] she was (.) a ma(h)rried woman with a(h)-
8   Carl:  She's a right wally.
9    Sue:  mm [kids  as well ]
10  Carl:     [They   assume] an awful lot man¿
11   Sue:  °°ye:ah,°°
12  Carl: °°the elderly do.°°
```

Regex matches patterns. The simplest 'pattern' is a literal pattern. For example, let's assume you want to know how many utterances Sue uses to tell the story. So you can specify the pattern `Sue:` (including the colon might be important as `Sue` alone could also be used as a word). To extract all occurrences of the pattern `Sue:` we will use the function `grep()`: this function retrieves any vector element that contains the match. Its first argument is the pattern, and the second is the variable name in which the pattern is being searched for:

```
> grep("Sue:", CAt$who)
[1] 1 4 9 11
```

The output may look confusing at first glance: why a series of numbers? The numbers are simply the indices of the matches in the vector `CAt$who`: so the first element is a match, as are the fourth, the nineth, and the eleventh. You can get the matches not only as indices but also as text, by adding the argument `value = T`:

```
> grep("Sue:", CAt$who, value = T)
[1] "Sue:" "Sue:" "Sue:" "Sue:"
```

So you can easily see that there are four distinct utterances by Sue in this storytelling. If you prefer to obtain a number, you can store the matches in a vector, call it for example `sue`, and use `length()` to determine the number of elements it contains:

```
> pattern <- "Sue:"
> sue <- grep(pattern, CAt$who)
> length(sue)
[1] 4
```

Now let's assume you wish to find out how many utterances the participants to this storytelling use altogether. This could be achieved in at least two ways. One way is by using two literal patterns conjoined by the metacharacter `|`, which marks alternatives:

```
> grep("Sue:|Carl:", CAt$who, value = T)
[1] "Sue:" "Carl:" "Sue:" "Carl:" "Sue:" "Carl:" "Sue:" "Carl:"
```

Again you can determine the number of utterances by using `length()`, this time directly without storing the matches in a vector:

```
> length(grep("Sue:|Carl:", CAt$who, value = T))
[1] 8
```

But in lengthy transcripts, or if you work with many transcripts at the same time, it will not be possible to specify each speaker in advance. Then you will prefer to use a more abstract pattern. What is it that the character strings `Sue:` and `Carl:` have in common? They share three properties: (i) they start with an uppercase letter, (ii) they are followed by a number of letters, and (iii) they share the colon at the end of the string. To capture these shared properties we will use *metacharacters*, that is, symbols with specialized meanings (for an overview see Table 2.1).

Table 2.1 Regex metacharacters

Description	Meta-character	Function
backslash	\	double backslash \\ 'escapes' metacharacters, i.e., \\ marks them as to be processed *literally* ; e.g., \\. matches full stops
caret	^	marks beginning of string; inside character class [ ] it negates the character class; e.g. [^a-z] means '*not* lower case letters'
dollar sign	$	matches end of string
period or dot	.	matches 'anything'
vertical bar	|	marks alternatives
question mark	?	marks optional item (but used differently in look-arounds)
asterisk	*	matches preceding item zero or more times
plus sign	+	matches peceding item once or more times
parentheses	( )	delimit regex parts to be processed as a group, e.g. alternatives
square brackets	[ ]	delimit character class, e.g., [a-zA-Z] (all letters)
curly brackets	{ }	delimit quantifying expression, e.g., {1,5} (at least once, at most five times)

We can define the following pattern, where ^ denotes the beginning of a string, `[A-Z]` captures the class of upper case letters, `\\w` matches any alphanumeric characters, and + matches the preceding item (in this case `\\w`) at least once; the only literal element in the regex is the colon:

```
> grep(("^[A-Z]\\w+:", CAt$who, value = T)
[1] "Sue:" "Carl:" "Sue:" "Carl:" "Sue:" "Carl:" "Sue:" "Carl:"
```

Note that `\\w` is a *shorthand* character class, the equivalent of `[A-Za-z0-9_]`, which matches any alphanumeric character plus the underscore. To match only numbers one can use the class `[0-9]` or the shorthand `\\d` (d for digit). Another useful shorthand character class is `\\s` for any whitespace character, including white space, tabs, new lines, etc. If only the tab character is to be matched, the shorthand `\\t` can be used; similarly `\\n` matches new lines.

The above pattern also includes the caret `^`. This metacharacter is an 'anchor'. That is, it does not match a character but a character's *position* in the string. The anchor `^` matches a string's beginning. The opposite position, at the end of a string, is matched by the anchor `$`; for example, the pattern `"\\d$"` applied to the PoS tags `"VM0"` (for modal verb) and `"VVZ"` (for the 3rd person singular form of lexical verbs) matches `"VM0"` but not `"VVZ"`. Finally, another useful anchor is `\\b`. It allows you to match whole words, which is handy in linguistic research. For example, `"\\bwell\\b"` captures the adverb 'well' but not, for instance, 'unwell' or 'well-known'.

Let's return to our transcript `CAt` to address a different question: how to extract instances of constructed dialog, or direct speech, indicated in the transcript through quote marks? Again, think of what these instances have in common: in the extract there is a single instance, identifiable via the (bent) quote marks around it. So the pattern would be anything between " and ". This would suggest this pattern:

```
> pattern <- "“.*”"
> grep(pattern, CAt$story, value = T)
[1] "“make sure you're ba(hh)ck before midni(hh)ght.”           "
```

In the case of the short transcript with a single instance this pattern seems to work fine. But imagine working on larger transcripts with multiple instances of constructed dialog. Then the regex would match anything that is between the first opening quote mark " and the last closing quote mark " in the same vector element; that is, within vector elements, it would find, not bounded instances of constructed dialog, but the whole text from the first instance to the very last with all the intervening text as well! To make sure that all matches *stop* at each closing quote mark, you will have to *exclude* quote marks from those characters that are allowed to intervene between the quote opening and its closing. As noted above, exclusion is achieved by means of the caret `^` used inside the square brackets `[ ]`, thus:

```
> pattern <- "“[^”]*”"
> grep(pattern, CAt$story, value = T)
[1] "“make sure you’re ba(hh)ck before midni(hh)ght.”           "
```

Looking more closely at the match you may notice the lengthy stretch of white space following the closing quote mark – why is this included although it is clearly not part of the quote? The inclusion is because, as said above, `grep()` finds vector elements that *contain* matches. In other words: it not only finds the pattern match but anything else that may be part of the element – in the case of the element in question, a lot of white space. Consider this example from another storytelling:

```
> ex <- c("So he goes "↑beep beep↑" on the horn", "this bloke went
"HUh HUh,"")
> grep(pattern, ex, value = T)
[1] "So he goes "↑beep beep↑" on the horn" "this bloke went "HUh
HUh,""
```

Here, we get, not only "↑beep beep↑" and "HUh HUh,", the only true quotes, but the complete two lines. To extract the quotes from any surrounding context, a slightly more complex procedure is necessary, involving two more functions, `gregexpr()` and `regmatches()`: `gregexpr()` returns a list of the starting positions of each match and `regmatches()` produces a list of the texts of each match; to get the raw matches, use `unlist()`:[12]

```
> matches <- gregexpr(pattern, ex, perl = T)
> quotes <- regmatches(ex, matches)
> unlist(quotes)
[1] ""↑beep beep↑"" ""HUh HUh,""
```

Let's check with the CA transcript: do we get rid of all extra material surrounding the quote by using `gregexpr()` and `regmatches()`? For brevity sake, let's collapse the three seperate steps into a one-liner:

```
> unlist(regmatches(CAt$story, gregexpr(pattern, CAt$story, perl = T)))
[1] ""make sure you're ba(hh)ck before midni(hh)ght.""
```

Yes, the trailing space behind the quote has disappeared. And, since we are going to use the one-liner several times, let's simplify the code even further by defining a *function* for it so we don't always have to repeat the code; let's call this function simply `extract`:

12. Alternatively, and more simply, you can use the function `str_extract` from the `stringr` package:

```
> install.packages("stringr")        # install the package
> library(stringr)                   # call the package
> str_extract(CAt$story,pattern)     # extract raw matches
```

```
extract <- function(x) unlist(regmatches(x, gregexpr(pattern, x,
perl = T)))
```

So, next time we perform the extraction of the raw matches we simply call the function to get them.

For another illustration of the usefulness of regex for CA research, let's assume we are interested in extracting pauses. As always, the first step is to determine the shared properties: in the story there are two short pauses, `(.)`, and one longish pause, `(0.8)`. The two strings share the round opening and closing brackets as well as the period; the numbers are optional elements. Both round brackets and the period are metacharacters. This means we have to 'escape' them; escaping is achieved in R by using the double backslash `\\`. Let's approach the task of formulating the pattern in two steps.

```
> pattern <- "\\(\\.\\)"
> extract(CAt$story)
[1] "(.)" "(.)"
```

We can see that this regex did not find all pauses, only the short ones. To see why, let's break the regex into meaningful chunks: the first chunk is `\\(`, which matches opening brackets; the next is `\\.`, which matches the period; and, finally, `\\)` finds closing brackets. What is missing is the option that the pause is expressed as a decimal number. To account for this possibility, we will use `(…)?`, the syntactic structure to denote optional characters. To target optional numbers we will use `\\d` (`d` for 'digits'); alternatively, we could use `[0-9]`:

```
> pattern <- "\\((\\d+)?\\.(\\d+)?\\)"
> extract(CAt$story)
[1] "(.)"    "(0.8)" "(.)"
```

This now finds all pauses. Note the regex chunks: beside the already known chunks `\\(` for the opening brackets, `\\.` for the period, and `\\)` for the closing brackets, the new regex contains `(\\d+)?` twice, which expresses the optional occurrence of one or more digits.

Obviously, you would want to extract pauses in order to process them further (for example, in order to find out how long pauses are in certain sequential positions). But at this point such further processing would not yet be possible, for two reasons. The first is that R does not know that the results of the above operation represent numbers. Instead it treats it, not surprisingly, as characters. This can be seen if you store the results in a vector, called `pauses`, and query for its structure using `str()`:

```
> pauses <- extract(CAt$story)
> str(pauses)
chr [1:3] "(.)" "(0.8)" "(.)"
```

And why should R treat the results as numbers as they contain a lot of material that patently is not numeric, such as brackets and brackets without any numbers? So, the first task would be to deal with short pauses, transcribed as "(.)". It is customary in CA to transcribe pauses in this way when they are shorter than 0.3 seconds. On the other hand, we know that there is a perceptual gap of 120 ms (Heldner 2011); that is, anything shorther than that gap cannot be perceived as a gap in the first place. So, a reasonable suggestion would be to set all short gaps to the arithmetic mean of 120 ms and 300 ms, which is 210 ms, or 0.2 s. This transformation can be done with `gsub()`, a function that takes as its first argument a pattern and as its second argument the string into which the pattern is to be transformed; the third argument is the vector in which the matches are located:

```
> pauses <- gsub("\\(\\.\\)", "(0.2)", pauses)
> pauses
[1] "(0.2)" "(0.8)" "(0.2)"
```

Now we have decimal numbers in all matches. But we also need to get rid of the brackets; again `gsub()` is useful here:

```
> pauses <- gsub("(\\(|\\))", "", pauses)
> pauses
[1] "0.2" "0.8" "0.2"
```

Now the brackets are gone. But for R, `pauses` is still a character vector. To change it into numeric, we transform it using, as above, `as.numeric()`:

```
> pauses <- as.numeric(pauses)
> str(pauses)
num [1:3] 0.2 0.8 0.2
```

Now we could start working with the pauses as *numbers*.

For another illustration let's work with laughter. It is conventional in CA to transcribe within-word laughter by laughter particles wrapped into round brackets. The pattern to match words containing laughter particles can be formulated thus, this time making use of a quantifying expression in curly brackets `{ }`: the first number after the opening curly bracket specifies the minimum number of times an item is to occur; the second number, after the comma, specifies the maximum number of times an item occurs; if either the first or the second number are *not* provided, then the minimum or, respectively, maximum number of occurrences are left undefined:

```
> pattern <- "\\w*\\(h{1,}\\)\\w*"
```

The components are as follows: `\\w*` looks for zero or more alphanumeric characters, `\\(` targets the opening round bracket, `h{1,}` matches at least one `h` (the maximum number of expected tokens is undefined), `\\)` matches the closing bracket, and `\\w*` looks for letters again. If we use our function `extract()` as above we get an exhaustive list of all the words with laughter particles in them:

```
> extract(CAt$story)
[1] "ba(hh)ck" "midni(hh)ght" "ma(h)rried" "a(h)"
```

Overlap is yet another critical topic in CA. How can stretches of overlapped speech be extracted? The phenomenon is commonly transcribed via square brackets. So we could target overlap thus:

```
> pattern <- "\\[[^]]*]"
> extract(CAt$story)
[1] "[ twice.]""[ Yeah. ]" "[Mm.]""[There]""[kids as well ]""[They
assume]"
```

The pattern is composed of `\\[`, which escapes the opening square bracket (the closing square bracket, by contrast, does not need escaping); `[^]]*`, a character class which includes any character occurring zero or more times except the closing square bracket (see the similar problem described above in matching quotes); and `]`, which matches the closing square bracket.

Now, let's take this pattern for overlap one step further to explore yet another syntactic resource of regex: lookarounds. Let's assume that we want to get at the words immediately preceding the overlapped words. We could easily define a pattern to match *both* the prior word *and* the overlapped words, by adding `\\w+` for the prior word and `\\s` for the white space betweeen the two words:

```
> pattern <- "\\w+\\s\\[[^]]*]"
 > extract(CAt$story)
[1] "or [ twice.]"      "mm [kids as well ]"
```

But that's not the desired output: what we want to obtain is *just* the word prior to the overlap, that is, in this case, "or" and "mm". So we need to formulate the pattern in such a way that the part that matches the overlap is denoted merely as a co-occurrence phenomen but does not get *consumed* by the pattern. Matching patterns via co-occurring patterns is achieved by lookarounds (cf. Gries 2017). There are two major types of lookarounds: lookaheads and lookbehinds. Lookaheads encode the instruction: "match X if you see Y on the right", lookbehinds take the reverse direction: "match X if you see Y on the left". To target the word that immediately precedes overlap we will use lookahead – the word prior to overlap represents X while the overlap represents Y. Using the 'grammar' for lookahead, which is `(? = …)`, we can formulate this pattern:

```
 > pattern <- "\\w+(?=\\s\\[[^]]*])"
```

We have not changed much compared to the above pattern: we have defined `\\s\\[[^]]*]` as the co-occurrence pattern that we wish *not* to be extracted along with the target word by putting it into the lookahead syntax: `(?=\\s\\[[^]]*])`. If we now call `extract`, we get the exact matches:

```
> extract(CAt$story)
[1] "or" "mm"
```

Sticking with overlap and the words occurring in their vicinity, we could attempt to extract those words that *follow* overlap,. In this case, we would use lookbehind. Unfortunately, some engines do not support use of quantifier expressions such as `*` within lookbehinds, so we need to reformulate that part of the regex that addresses the overlap. Since only overlap is wrapped into square brackets and we are interested in what follows the closing square bracket of overlap, we could simply define that character as the anchor point:

```
> pattern <- "(?<=]\\s)\\w+"
```

This pattern instructs the engine to look for a word (`\\w+`) if that word is preceded by a closing square bracket and a space (`(?<=]\\s)`). And, indeed, the lookbehind finds the two correct matches:

```
> extract(CAt$story)
[1] "she" "an"
```

Lookarounds can also be negated. That is, instead of matching a pattern if there is a co-occurrence pattern on the right, you can tackle a pattern if, and only if, there is *no* such co-occurrence to the right. And, instead of matching a pattern that is defined by a preceding pattern you can query for a pattern that is *not* preceded by such pattern. The syntax is `(?!…)` for negative lookahead and `(?<!…)` for negative lookbehind. For example, to return to the above examples of words co-occurring with overlap, you could address and extract all words that are *not* followed by overlap, by adapting either the more complex pattern used above or the shorter one; they find exactly the same matches:

```
> pattern <- "(?!\\[[^]]*]\\s)\\w+"
> pattern <- "(?!]\\s)\\w+"
> extract(CAt$story)[1:10]      # output clipped to the first 10 matches
[1] "That" "s" "like" "your" "grand" "ma" "did" "that" "with" "erm"
```

Note that in defining the pattern we have switched the order of components: although we are looking for X preceding Y, we can put the pattern for Y in front of the pattern for X. R will still process the regex in the right way, as a lookahead.

Conversation Analysts have not yet taken advantage of regex to any large extent. To my knowledge, regex has only very recently been exploited in CA-related work (e.g., Haugh & Musgrave 2019). But regex offers great potential for analyzing CA transcripts. What has been demonstrated in this small section was based on a mini transcript. But that size restriction was owed to illustrative reasons, not to regex. Regex does not care as to how many transcripts it is used on: it could be used to match and extract patterns from thousands of transcripts in a single query. So what regex can do for CA is get a handle on *data of scale* – provided that codings, such as those underlying CA transcripts, remain true to CA principles (cf. Stivers 2015). In other words: it can help add to CA's methodological toolbox a serious quantitative component.

2.7 Importing and exporting data

Two essential operations are importing data from files into R and, conversely, exporting data from R into files.

Before importing data to R, it is important to store data in the right way. This 'right way' is governed by one principle: "all the values of the same variable must go in the same column" (Crawley 2007: 108). So, for example, if you had response times for different types of backchannels (cf. the case study in Chapter 12), it's not a good idea to devise one column for the response times for continuers, one for assessments, and so forth; this would scatter the values of the dependent variable (response time) across different columns. Rather, all the response times should be collected in one column and their respective backchannel types in another column.
Once your data is in the right layout, you can load it into R for further processing. For illustration, let's use a short excerpt from a larger dataset we will work with in Chapter 6 on nucleus placement (cf. column `Nucleus`) and surprisal (column `s1`); the file, called "Chapter2_ImportExample.txt", can be downloaded from the companion website.

```
   Speaker                                             Turn    s1  Nucleus
1              we bought the thomas second and then we  2.17        4
2    PS1BT           enough to put put , put gordon on  3.14
3    PS1BS       the original pop goes the weasel song              9
4    PS1BT  what you 're thinking there is china clay  0.69        9
5    PS1BS              sure it was n't chris as a baby  3.04       10
```

Note at this point two things: first, the first cell in the `Speaker` column is empty; and there are more empty cells in the last two columns. These are features that need to be considered when loading the data (see below).

The function to use for importing data depends on the format of the file you wish to read in. If your data is in .txt format, where the columns are separated by tabs, the `read.table()` function is key. If the data is stored in a csv-formatted file, then the function `read.csv()` is used.

In the following we'll assume that the file is stored as .txt. The first argument for both `read.table()` and `read.csv()` is the path to the file on your computer. Note that in Windows, that path normally starts with `"C:/Users/`… whereas it will start with `"/Users/`… on a Mac. Following the path, the first argument is `header`: it is set to `TRUE` (or simply `T`) if the columns have headers (which they usually, and ideally, do) and to `FALSE` (or `F`) if they don't.

Now let's try and read in the file:

```
> import <- read.table("[Your path]/ImportExample.txt",
+                     header = T)
Error in read.table("[Your path]/ImportExample.txt", :
  more columns than column names
In addition: Warning message:
In read.table("[Your path]/ImportExample.txt", :
  incomplete final line found by readTableHeader on '[Your Path]/
Import_example.txt''
```

We get an error and a warning – apparently, more parameters need to be specified to upload the data. In addition to `header = T`, we will use `quote = ""`, and `sep = "\t"`: the argument quote `= ""` ensures that no special characters will be used for quoted strings, `sep = "\t"` specifies that we have tab-delimited data fields:

```
> import <- read.table("[Your path]/ImportExample.txt",
+ header = T, sep = "\t", quote = "")
> import
  Speaker                                   Turn   s1 Nucleus
1           we bought the thomas second and then we 2.17       4
2   PS1BT         enough to put put , put gordon on 3.14      NA
3   PS1BS       the original pop goes the weasel song   NA       9
4   PS1BT what you 're thinking there is china clay 0.69       9
5   PS1BS               sure it was n't chris as a baby 3.04      10
```

This time the read-in worked. Looking more closely, though, we see that R has made some significant changes. The first change is the addition of `NA` to the empty fields under `s1` and `Nucleus`; by contrast, R has not added `NA` to the empty field under `Speaker` – why these changes?[13]

R does not import data blindly; it analyzes them in terms of structure. This emerges clearly from calling `str()`:

13. It might be useful to set blank cells from the start to NA. This can be achieved by including the argument `na.string = ""` in the call to `read.table()`.

```
> str(import)
'data.frame':     5 obs. of 4 variables:
$ Speaker: Factor w/ 3 levels "","PS1BS","PS1BT": 1 3 2 3 2
$ Turn    : Factor w/ 5 levels "enough to put put , put gordon on",..:
           4 1 3 5 2
$ s1: num 2.17 3.14 NA 0.69 3.04
$ Nucleus: int 4 NA 9 9 10
```

Not only has R analyzed the data as a dataframe (see first line) but it has 'decided' that the third column represents numerical data and the forth column integers while the first two columns represent factors. We will agreee that the informativity, or surprisal, values stored under `s1` represent numerical values and the `Nucleus` column contains integers; but the data under `Turns` are strings of characters rather than factors (as each turn is unique). The reason for this latter conversion is that `read.table()` *by default* converts character variables to factors. To prevent this, the argument `as.is` can be used: to keep the column `Turn` as a character variable, you could alternatively specify `as.is = "Turn"` or `as.is = 2`:

```
> import <- read.table("[Your path]/ImportExample.txt",
+                      header = T, quote = "", sep = "\t", as.is = 2)
```

```
> str(import)
'data.frame':     5 obs. of 4 variables:
 $ Speaker: Factor w/ 3 levels "","PS1BS","PS1BT": 1 3 2 3 2
 $ Turn    : chr "we bought the thomas second and then we" "enough
to put put , put gordon on" "the original pop goes the weasel song"
"what you 're thinking there is china clay" ...
 $ s1      : num 2.17 3.14 NA 0.69 3.04
 $ Nucleus: int 4 NA 9 9 10
```

Now `Turn` is read-in as a character variable and the data has been imported successfully. Importing data can at times require quite some thought, (and for beginners often creates head-aches), exporting data from R is much less demanding. Again, there are two major functions, `write.table()` and `write.csv()`. Their first argument is the name of the data to be exported, called `X` below. The second argument is the path to a dummy file to export the data into (obviously, this dummy file needs to be created and stored beforehand; in the code below, this file is called `Export_R.txt`). Note that the two functions will by default wrap quote marks around the data points and include the row names. To prevent this, set `quote = F` and `row.names = F`:

```
> write.table(X, "[Your path]/Export_R.txt", sep="\t", quote = F,
row.names = F)
```

The dataset generated in R will appear in `Export_R.txt` and can be further processed from there.

Chapter 3

Graphical parameters

3.1 Introduction

Suppose that, perhaps after a long and arduous process of carefully transforming your data (cf. Chapter 2), you have your data all set up for plotting. You may feel a sense of relief at this point, and deservedly so. But often the real work has only just begun. The reason is that while there are ready-made functions for most plot types introduced in this volume (the one exception being the location plot in Chapter 4), there may be a vast number of decisions to be taken not only as to what exactly to plot but, perhaps even more so, *how* to plot it. One and the same data can be plotted in innumerable ways, depending on which graphical parameters you choose and how you implement them. In other words, making an informative and aesthetically pleasing graphic involves a wealth of *design* questions. For example, are you going to make a plot with a single visualization or will there be two or three 'panels' in the plot? If the latter is the case, then how do you wish to arrange the panels: horizontally or vertically? Further, color is an obvious design feature, arguably, of every plot: does black-and-white suffice or do you require color? If the data suggest the use of colors, there's the question of which colors to use – just random ones or ones from the numerous palettes R offers? If you need to distinguish different types of data (e.g., two distinct factor levels) by using different point characters, you will have to decide which ones to use, and possibly also, which size and even which color? If you have a large dataset, the problem of overplotting will inevitably arise – how to minimize that? How many axes do you need and how do you want to label them? How 'bulky' is your legend and how can it be placed into the plot so that it does not overlay important data points? This list of design questions could be continued endlessly.

The aim in this chapter is to introduce you to the main recurrent graphical parameters. The total number of graphical parameters in R is overwhelming. Some graphic types have special parameters unique to them, many however are shared by all graphic types. We will focus here on the ones that are foundational in the sense that any graphical representation will make some use of them. Call `?par` to see the wealth of parameters R offers. In Sections 3.2 to 3.6 the main parameters are introduced; Section 3.7, then, serves to illustrate their use based on a case study.

3.2 Plot layout

The first consideration when plotting a graphic relates to layout. Layout involves two major aspects: how many graphics, called 'panels', are to be combined in the plot, and in which order, and how wide you wish to leave the margins around the plotting regions.

The number and order of panels in a plot is determined by the argument `mfrow = c( …, …)`, a vector whose first element determines the number of rows, the second the number of columns.If set to `mfrow = c( 1, 1)`, which is the default, R will produce a single panel; if set, say, to `mfrow = c( 2, 4)`, altogether eight panels will be produced arranged in two rows and four columns.

The margins around the panel(s) are set by `mar = c( …, …, …, …,)`, a vector whose four elements determine the margins – in that order – bottom, left, top, right of the panel.

3.3 Point characters and line types

There is a wide range of point characters to represent data points. Point characters are defined as arguments to graphics, that is, inside the call for a specific graphic; the argument takes the form `pch = `…. For example, `pch = 1` selects the empty circle (the default point character in R). The first 25 point characters are shown in Figure 3.1.

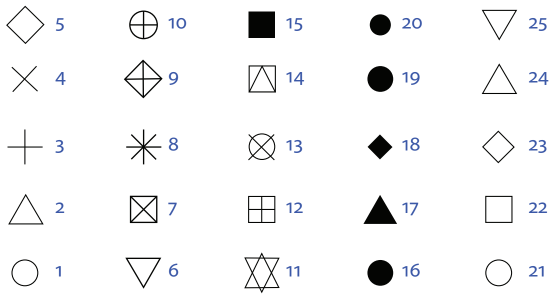


Figure 3.1 Main point characters

You can combine point characters in one and the same plot easily, for example, when you have two or more levels of a variable and want to distinguish them visually. This is accomplished by combining as many point characters in the function `c()` and by defining the argument `pch` through that vector: `pch = c()`. For example, `pch = c(1:5,10)`, selects the first five point characters and the tenth (cf. Figure 3.1).

Another important graphical parameter are lines. Lines can indicate two types of information: they can either connect data points or they can provide additional information, such as the mean of a distribution or density (cf. Chapter 12). Some graphic types, such as the scatter plot, standardly offer various options for connecting data points; the argument to define line type is `type`. You can set `type` to `type = "p"` (for unconnected points, the default), `type = "l"` (for lines), `type = "b"` (for individual points connected by lines), or `type = "h"` (for histogram-like lines), to name only four options. Another important specification is `type = "n"`, which is handy when you do not wish to actually plot data points (which may indeed be necessary, as will be seen in a number of the case studies).

A number of additional line types are available. For example the function `abline()` can be used to draw, for example, linear regression lines or other straight lines. To accomplish this you would use the function `lm()` (for linear model), which takes two variables `x` and `y`, inside `abline()`, thus: `abline(lm(x ~ y))`. The function `abline()` can also be used to draw lines for the arithmetic mean or the median, either vertically, `abline(v = mean(x))`, or horizontally, `abline(h = median(x))`. Another useful resource for adding lines to plots is `segments()`; it takes four arguments: the coordinates `x0` and `y0` from which to draw the line, and the coordinates `x1` and `y1` to which to draw. For example, `segments(x0 = 1, y0 = 1, x1 = 5, y1 = 5)`[14] draws a diagonal from the coordinates (1,1) to (5,5).

The look of lines can be changed in many ways. Line width can be varied by `lwd`; `lwd` defaults to 1, so larger positive numbers will increase line width. Among line types you have the choice between solid, `lty = 1`, dotted, `lty = 2`, dashed, `lty = 3`, and a few others. Another important parameter for changing the look of lines is `col` (for color; see below).

3.4 Adding text

The function `text()` allows you to add text to diagrams; the text added can be descriptive (for example, to designate a line as representing the mean) or, if you work with textual data such as words or phonetic transcriptions and some numeric variable for them (e.g., word length), that data itself can be printed into the graphic (see the illustrative case study below in Section 3.7 on duration , word class, and phonetic size).

The function `text()` takes essentially three arguments: the location of the text on the x-axis, the location on the y-axis, and the text itself, in straight quote marks. The syntax, then, is `text(x, y, "…")`. Text size can be changed by the argument `cex`, which defaults to 1; so `cex = 0.5` decreases the font whereas `cex = 1.2` increases it.

14. The coordinate labels are optional. Thus, `segments(1, 1, 5, 5)` works just as well.

3.5 Legends

Only few graphics are complete without a legend.

Legends are drawn with the function `legend()`. The function accommodates a large number of arguments, only some of which can be introduced here (more will be described in the case studies). The first argument to `legend()` concerns its location in the plot, which can be indicated via values on the x- and, respectively, the y-axis, or by keywords such as `"top"`, `"topright"`, `"topleft"`, etc. The next is the argument `legend` itself, which mostly takes a character vector of the labels for the variables in the plot. In legends, typically, a box appears before the variable label; the argument `fill` serves to set the colors for these boxes. The argument `border` determines the color of the border line around the boxes (if no such border is wanted, then this argument can be set to `border = "white"`). If point characters are to precede the legend labels instead of boxes, then `col` is used instead of `fill` to determine the colors of the point characters. Finally, the argument `bty` determines the type of box around the whole legend; if it is set to `bty = "n"`, no box will be drawn.

Finally, we turn to perhaps the most fundamental graphical parameter, colors.

3.6 Colors

R offers both built-in colors as well as color sets.

There are 657 built-in colors. Their names can be inspected by calling `colors()`. For space considerations, we will just look at the first six and the last six ones, by calling the `head()` function and, respectively, the complementary function `tail()`:

```
> head(colors())
  [1] "white""aliceblue""antiquewhite""antiquewhite1""antiquewhite
  2""antiquewhite3"
> tail(colors())
  [1] "yellow""yellow1""yellow2""yellow3""yellow4""yellowgreen"
```

From this small selection alone you get an impression of how broad a palette of colors R offers; for example, note that among the first six colors there are four different types of antiquewhite. If this selection still appears limited check out random samples of colors, using the `sample()` function:

```
> sample(colors(),10)
  [1] "grey82" "lightsteelblue""gray83""honeydew1""gray96""grey91"
  [7] "seagreen1" "lemonchiffon4""grey99""grey53"
```

As `colors()` is a vector, any elements of it can be accessed using (sequences of) indices:

```
> colors()[111:114]
[1] "darkslategray3""darkslategray4""darkslategrey""darkturquoise"
```

To compare the built-in colors not only by name but actually see them side-by-side, use this command:

```
> demo("colors")
```

To add to the already huge variability of the built-in colors, you can additonally *scale* them to make them more or less transparent. This is achieved by the function `adjust-color()`, where the parameter `alpha.f` can be set to any value between 0 (complete transparency) and 1 (no transparency). For example, this code sets five colors accessed by indices to semi-transparent:

```
> adjustcolor(colors()[100:104], alpha.f = 0.5)
```

Beside the built-in colors, R makes color *sets* available, including the following:

```
rainbow()          # colors from the rainbow palette
heat.colors()      # colors ranging from white to orange and red
terrain.colors()   # earthly colors from white to green and brown
cm.colors          # colors in the blue and magenta spectrum
gray.colors()      # shades of gray
```

As with built-in colors, colors from color sets can be scaled by using the `alpha` parameter. For example, this code increases the transparency of three rainbow colors massively:

```
> rainbow(3, alpha = 0.1)
```

Figure 3.2 illustrates how both built-in colors and color sets can be used in plots: the figure shows a series of nine piecharts arranged in three rows and three columns. This arrangement is achieved by `par(mfrow=c(3,3))`, where the first number defines the number of rows, and the second number determines the number of columns. Also note the use of the `mar` parameter, which is set here to 0.5 (the function `rep()` repeats an object as many times as specified after the comma); in all pie charts the radius is extremely large, simply to zoom in on the colors (and avoid the 'pie look', which is considered a poor visualization for data).

Starting from the top and going from left to right, the first pie chart specifies the built-in colors to be used by name (`c("red","royalblue3","yellow")`), the second specifies them by indices (`colors()[503:506]`), the third draws a random sample of 24 colors (`sample(colors(),24)`), and the fourth accesses colors via indices and instructs R to increase their transparency (`adjustcolor(colors()[100:124], alpha.f = 0.5)`). The fifth chart uses the `rainbow()` palette, with three colors set to semi-transparent by the argument `alpha = 0.5`. The remaining charts illustrate the colors of `heat.colors()`, `cm.colors()`, and `gray.colors()`.

```
> par(mfrow=c(3,3), mar=c(rep(0.5,4)))

> pie(rep(1, 24), col = c("red","royalblue3","yellow"), radius = 4)
> pie(rep(1, 24), col = colors()[503:506], radius = 4)
> pie(rep(1, 24), col = sample(colors(),24), radius = 4)
> pie(rep(1, 24), col = adjustcolor(colors()[100:124],alpha.f=0.5),
                  radius = 4)
> pie(rep(1, 24), col = rainbow(3, alpha = 0.5), radius = 4)
> pie(rep(1, 24), col = heat.colors(8), radius = 4)
> pie(rep(1, 24), col = topo.colors(6), radius = 4)
> pie(rep(1, 24), col = cm.colors(3), radius = 4)
> pie(rep(1, 24), col = gray.colors(24), radius = 4)
```

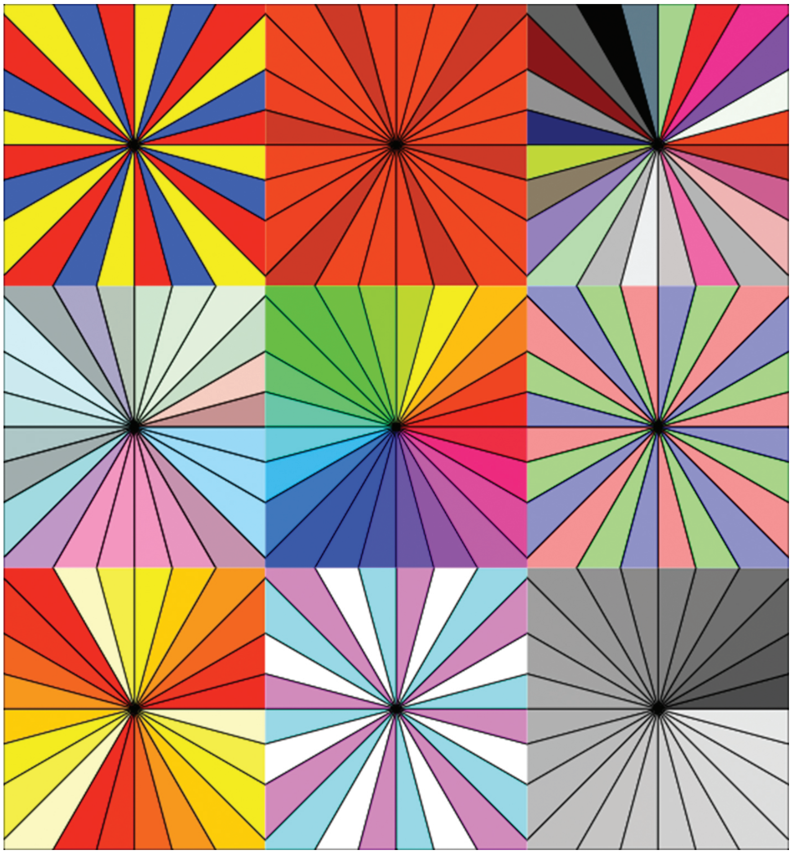


Figure 3.2 Pie charts to illustrate built-in colors and color sets

3.7 An illustration: Duration, word class, and phonetic size

Let's illustrate what has been said so far by a quick case study. The data for this case study, called "Chapter3_Casestudy.txt", as well as the code file, called "Chapter3_Casestudy_Code.R" can be downloaded from the companion website.

The case study addresses two questions. The first question is the correlation between a word's duration and its phonetic size (i.e., the number of phonemes making up the word): both research (e.g., Zipf 1949) and intuition suggest that a word will be longer in duration the longer it is in terms of the phonemes a speaker must produce to articulate the word. The second question is whether word class is involved in this correlation. Specifically, we will assume that function words, which are the most frequently used tokens, will be short both in terms of phonetic make-up and discourse duration and, conversely, that content words will be longer on both counts.

The aim here is to visualize these potential correlations based on a sample of word tokens pulled from the conversational subcorpus of the BNC and measured in Praat (Boersma & Weenik 2012) for duration. If a word type occurred more than once, each time inevitably with a slightly different duration in the discourse, *one* of the word's tokens was randomly selected. This filtering process led to a subsample of 1,329 unique word tokens and the duration they had upon one instance of use in conversational interaction. Also, following the procedure described in detail in Section 7.1.2, the words were phonetically transcribed and the number of phonemes making up the words were counted.

The data are organized in six variables: the word tokens (called `AllWords`), their phonetic transcriptions (`AllTrans`), their part-of-speech tags (`Allc5`), their durations in discourse (`AllDurs`), and their phonetic sizes (`AllPhons`).

The variables are stored in a dataframe called `gp_s` (for graphical parameters sample); the first ten rows of `gp_s` look like this

```
> gp_s[1:10,]
   AllWords  Allc5  AllDurs  AllPhons  AllTrans    Class
1        'd    VHD    0.075         1         d function
2       'em    PNP    0.168         2        ɛm function
3       'll    VM0    0.056         1         l function
4        'm    VBB    0.048         1         m function
5       're    VBB    0.084         1         r function
6        's    VBZ    0.063         1         z function
7       've    VHB    0.044         1         v function
8         a    AT0    0.051         1         ə function
9  abington    NP0    0.446         7   æbiŋtən  content
10     able    AJ0    0.149         3      eɪbl  content
```

Further, to illustrate how the text() function can be exploited to print text into plots, a subsample is drawn, called gp_s2, containing just 50 word tokens and their associated variables.

The relation of duration, word class, and phonetic size in the two data sets are shown in Figure 3.3, with the two upper and the left lower panel based on the larger sample gp_s and the lower right panel based on the small sample gp_s2. The panels are arranged in two rows and two columns (the argument mar sets the plot margins around the panels):

```
> par(mfrow = c(2, 2), mar = c(4, 4, 2, 1))
```

All four scatter plots are set up using the tilde format *y* ~ *x*, that is, in our case *duration* ~ *phonetic size*. The range of values for phonetic size is tightly restricted; as a result, the more than 1,000 token values stored in gp_s would get heavily overplotted. To spread the values on the x-axis and thus reduce overplotting, the data is 'jittered'; that is, a little noise is added to each data point. This is achieved by the function jitter(); to increase the visual spread on the x-axis even slightly more, the factor argument, which defaults to 1, is set to 1.7. In the panel for gp_s2, with its merely 50 values, jittering is not deployed.

A basic first thing to do is add a title, as well as descriptive labels for the axes; the title is added in the argument main, the axes are labeled in the xlab and, respectively, ylab arguments. Further, initial inspection of the graph shows that there are a handful of extreme values (potential 'outliers'). While these need to be dealt with carefully in a serious analysis, they may stretch the axes in such a way that the bulk of data points is not shown in fine enough detail. You may therefore wish to exclude them from the plot. This is accomplished by delimiting the range of values to be shown on the axes; ylim = c(0, 1.1) restricts the upper limit of the y-axis, which shows the durations, to 1.1 seconds, and xlim = c(1, 11) cuts off the x-axis, which shows phonetic size, at 11 phonemes. For the small sample gp_s2, axis size is not restricted.

Next, the arguments cex.main, cex.axis, and cex.lab determine the font sizes of title and axes.

The parameters we have set so far only affect the data visualization marginally. The most important parameters are colors, expressed in the argument col, and point characters, expressed in pch.

In the three panels for gp_s, the col argument makes use of the rgb() function, which allows manipulation of both intensity and transparency of the red, green and blue primaries. Its syntax is rgb(…, …, …, alpha = …, maxColorValue = …), where the first three slots take the desired color intensities for the three primary colors, alpha determines the transparency level, and maxColorValue defines the maximum of the color values range (any integer between 0 and 255). This is a useful function for large data sets where many data points would visually get drowned due to overplotting (even despite jittering). In the plot in the upper left panel all three primaries are set to `0`, in which case R defaults to black; `alpha` is set to `80`, a low-intermediate transparency value (with `alpha = 0` meaning 100% transparency and `alpha = 255` meaning no transparency at all). Further, in the upper left panel, no point character is selected, so R's default character comes into play, namely `pch = 1`. To increase the size of the character, the argument `cex` is set to `cex = 1.3`. This is the code for the upper left panel so far (the last argument `frame = F` suppresses the default box around the plot):

```
> plot(jitter(gp_s$AllDurs, factor = 1.7) ~ jitter(gp_s$AllPhons, 1.7),
+      main = "Duration, Word class, Phonetic size",
+      xlab = "Phonetic size",
+      ylab = "Duration",
+      ylim = c(0,1), xlim = c(0,10),
+      cex.main = 0.9, + cex.axis = 0.8,
+     cex.lab = 0.8, + col = rgb(0, 0, 0, 80, maxColorValue = 255),
+      cex = 1.3
+      frame = F)
```

The two other panels representing data in `gp_s` differ by their color and point character parameters. In the upper right panel, the three primaries red, green, and blue are used to differentiate the three word classes content words, function words, and inserts. This is achieved by defining the argument `col` through subsetting `gp_s` on the variable `Class` and using nested `ifelse()` statements: as inserts are expected to be few in numbers and overplotting will not present an issue, `"green"` is selected without changes to intensity and transparency. For content words and function words, overplotting *is* an issue. Therefore, content words are to be shown in red in full intensity – with the maximum value `255` given in the first argument to `rgb()` – but a low-intermediate transparency level of `85`, whereas function words are shown in blue – the second argument to `rgb()` – and an intermediate transparency level of `120`:

```
+ col = ifelse(gp_s$Class=="insert", "green",
+              ifelse(gp_s$Class=="content", rgb(255,0,0,85, max-
               ColorValue = 255),
+                     rgb(0,0,255,120, maxColorValue = 255)))
```

In the lower left panel, this color coding is left untouched but the word classes are further distinguished by point characters. The characters are assigned using the same logic as the color codings, namely by subsetting on `Class` and using `ifelse()` clauses:

```
+      pch = ifelse(gp_s$Class=="insert", 18,
+                   ifelse(gp_s$Class=="content", 1, 5)),
```

This selects the point character `18` for all inserts, the character `1` for content words, and the character `15` for all function words.

In the lower right panel, the phonetic transcriptions of each token in the sub-sample in `gp_s2` is printed to the left of the (unique) point character. In the call to `plot()`, the parameters `col` and `pch` are defined thus:

```
+    col = ifelse(gp_s2$Class=="insert", "green",
+                  ifelse(gp_s2$Class=="function", "blue", "red")),
+    pch = 20,
```

That is, as before, depending on the values in `Class`, the three primaries are set as colors (in their simple form) and the default point character, which is `1`, is replaced by `20`.

To print the transcriptions next to the point character, the function `text()` is used, with `gp_s2$AllPhons` on the x-axis – the first argument to `text()`, `gp_s2$AllDurs` on the y-axis – the second argument to `text()`, and `gp_s2$AllTrans` as the vector of values to be printed – the third argument to `text()`. Moreover, while `cex` determines the character size, the argument `pos` determines the side to which the text is to be placed: `1` for underneath, `2` for to the left, `3` for above, and `4` to the right:

```
> text(gp_s2$AllPhons,
+      gp_s2$AllDurs,
+      gp_s2$AllTrans,
+      col = ifelse(gp_s2$Class=="insert", "green",
+                  ifelse(gp_s2$Class=="function", "blue", "red")),
+      cex = 0.8,
+      pos = 2)
```

Figure 3.3 also contains a number of lines. In the upper left panel, a regression line is included depicting the slope of the correlation for `gp_s$AllDurs ~ gp_s$AllPhons`. The slope clearly climbs upward, indicating that duration and phonetic size are directly correlated: the longer the duration the longer the word token in terms of its phonetic make-up.[15] This graphical element is produced using the `abline()` function as well as, inside it, the `lm()` function, which fits a linear model:

15. To establish in unmistakable terms whether two variables are correlated, it is customary to perform a correlation test with the function `cor.test()`; it takes as minimum input the two variables (in any order and separated not by the tilde ~ but by a comma) and, if no specific test is selected, it defaults to the Pearson correlation:

```
< cor.test(gp_s$AllDurs, gp_s$AllPhons)
      Pearson's product-moment correlation
data: gp_sample$AllDurs and gp_sample$AllPhons
t = 29.479, df = 1327, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
```

```
> abline(lm(gp_s$AllDurs ~ gp_s$AllPhons))
```

In the upper right panel, the grey dashed line is a locally-weighted regression line, also referred to as a smoother (cf. Chapter 9). It is almost as straight as the regression line in the upper right panel, suggesting that the correlation is quite robust.[16] The line is drawn using the `lines()` function and, inside it, `lowess()`, which takes the same input as the `lm()` function. The arguments `lty` and `lwd` determine the line type and the line width:

```
> lines(lowess(gp_s$AllDurs ~ gp_s$AllPhons), lty = 2, col = "grey",
lwd = 2)
```

This panel also contains dashed lines depicting the mean values for `gp_s$AllPhons` and `gp_s$AllDurs`: as shown in Figure 3.3, the average duration of a word in discourse is 0.3221761 seconds while the average word length is between 4 and 5 phonemes. The average lines are drawn using the `abline()` function and the arguments `v` for vertical and `h` for horizontal as well as the `means()` function:

```
> abline(v = mean(gp_s$AllPhons), lty = 3, lwd = 1)
> abline(h = mean(gp_s$AllDurs), lty = 3, lwd = 1)
```

Means lines are also included in the lower right panel. Here, however, they depict not the overall means but the means for the three word classes. To delimit the computation of the means to a word class, the sample is subset to that word class. Only the code for content words is given here:

```
> abline(v = mean(gp_s$AllPhons[gp_s$Class=="content"]), lty = 3,
col = "red", lwd = 2)
> abline(h = mean(gp_s$AllDurs[gp_s$Class=="content"]), lty = 3,
col = "red", lwd = 2)
```

```
 0.5954408 0.6605009
sample estimates:
      cor
0.6290712
```

Perhaps the most critical pieces of information are the p-value, which is < 2.2e-16, a value far smaller than 0.001, indicating that the correlation is very highly significant, and the value at the very bottom, which indicates the strength of the correlation: as the value of 0.6290712 is closer to 1, which indicates a perfect correlation, than to 0, which indicates no correlation at all, duration and phonetic size can be considered solidly correlated.

16. See the high correlation coefficient 0.6290712.

As shown in Figure 3.3, the word class means are clearly distinct, with content words scoring highest both in terms of phonetic size and duration, function words scoring lowest on both counts, and inserts located inbetween.[17]

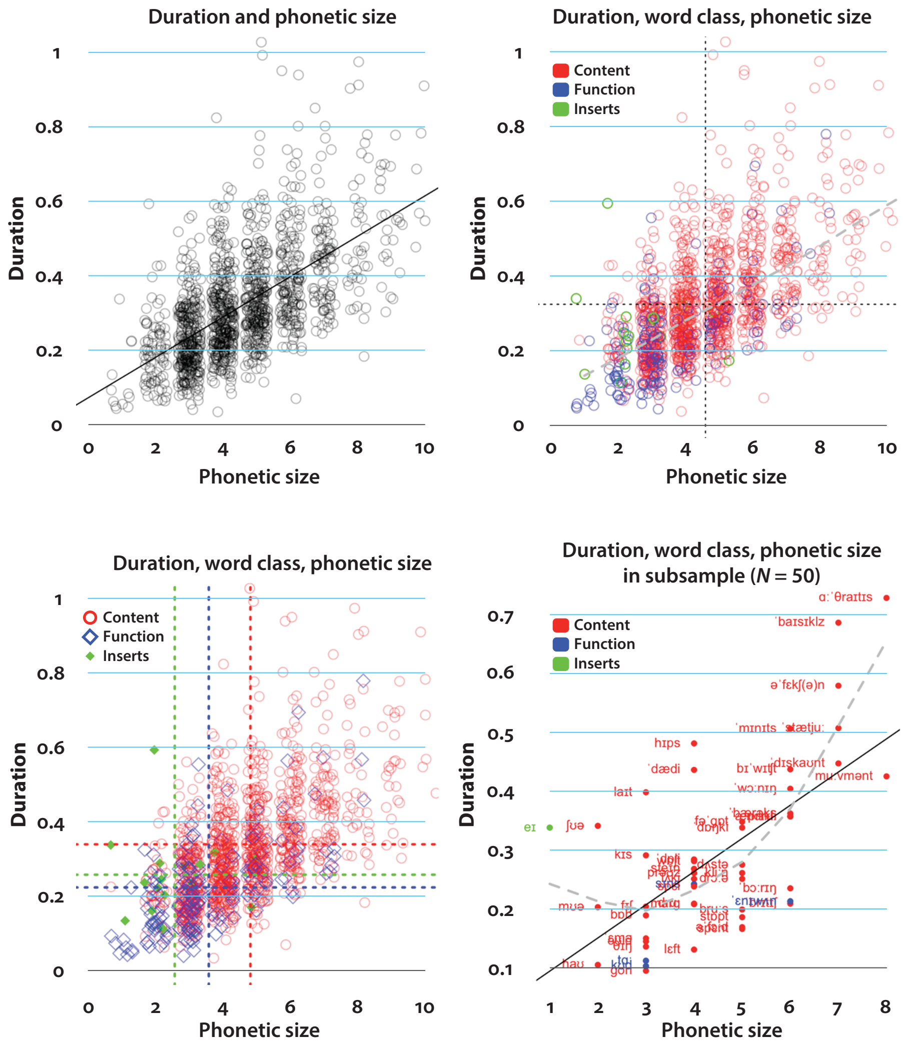


Figure 3.3 Duration, word class, and phonetic size

17. These averages are independent of the sample: they are confirmed across multiple samples.

The lower right panel includes the regression line and the smoother for the subsample. As before, the regression line rises steeply, suggesting that duration and phonetic size are in correlation even when only 50 data points are investigated. However, as can be seen from the smoother, this correlation is much less robust as the smoother forms a curve rather than a straight line.

To conclude, a quick glance at the legends. The code for the legend in the upper right panel is this:

```
> legend(0,1,
+         legend = c("Content", "Function", "Inserts"),
+         cex = 0.7,
+         fill = c("red", "blue", "green"),
+         border = "white", + bty = "n")
```

The legend is placed at `0` on the x-axis and `1` at the y-axis (that is, in the upper left area of the plot), the legend labels are given in the `legend` argument, the character extension is reduced with `cex = 0.7`; further, the argument `fill` determines the colors of the boxes to the left of the labels, `border = "white"` renders the border around the label boxes invisible, and `bty = "n"` suppresses the default box around the legend as a whole.

The code for the legend in the lower left panel differs by two parameters. First, `pch = c(1, 5, 18)` determines the point characters to be used instead of the squares to the left of the legend labels; second, `col = c("red","blue","green")` defines the colors of these point characters. In the lower right panel the only difference in the code for the legend is that the argument `border` is left unspecified, which is why, by default, black squares are drawn around the label boxes.

In this short illustration only a few parameters have been introduced. Many more will be introduced in the remaining chapters, Chapters 4–12, in which case studies are outlined and the central diagrams of the case studies are discussed in good detail.

As said earlier, the first case study examines what is done first in the turn and thus commences our journey from turn inception to turn transition that will keep us on our toes throughout the remainder of this book.

Chapter 4

Location plots

4.1 *Case study*: Turn structure and inserts

4.1.1 Introduction

Turn-taking is fundamental to human communication. As Levinson & Torreira put it:

> One of the most distinctive ethological properties of humans is that they spend considerable hours in the day in a close (often face-to-face) position with others, exchanging short bursts of sound in a human-specific communication pattern: (…) we may each produce about 1200 of these bursts a day, for a total of 2–3 h of speech. The bursts tend to involve a phrasal or clausal unit, but can be longer or shorter. At the end of such bursts, a speaker stops, and another takes a turn. This is the prime ecological niche for language, the context in which language is learned (…), in which the cultural forms of language have evolved, and where the bulk of language usage happens. (Levinson & Torreira 2015: 1)

The 'bursts' described are turns-at-talk. We produce, on average, 1,200 of them per day, containing, in total, between 15,000 (Roberts & Levinson 2017: 403) and 16,000 words (Mehl et al. 2007: 82). The average turn is between one and a half and two seconds (Levinson & Torreira 2015; Levinson 2016). Turns follow each other quickly, with a modal gap of just 200 ms (Stivers et al. 2009; Roberts et al. 2015).[18]

More importantly, though, turns-at talk are not only the prime ecological niche for language, they are also the prime units of social action: a turn "is constructed and is attended by its recipients for the action or actions it may be doing" (Schegloff 2001: 231). In a single turn, speakers can perform more than one action as turns are "multi-action compatible" (Levinson 2013: 108).

As the name suggests, turns-at-talk do not normally occur alone; rather, most turns are a 'turn-in-a-series' (Sacks et al. 1974: 722) interfacing with a prior turn and a next turn, thus potentially forming a 'sequence' (cf. Schegloff 2007). This seriality of turns is reflected in the potential three-part structure of turns: turns may have "one

18. Based on a sample of 50,000 utterances extracted from the conversational subcorpus of the BNC, Rühlemann (2018a) reported a mean number of 10.85 words per turn, which multiplied with 1,200 (see above), gives an estimate of around 13,000 words spoken per speaker per day.

[part] which addresses the relation of a turn to a prior, one involved with what is occupying the turn, and one which addresses the relation of the turn to a succeeding one" (Sacks et al. 1974: 722). Sacks et al. (1974) refer to the three parts as (i) pre-start, (ii) turn-constructional unit (TCU), and (iii) post-completer.

The three turn parts attend to different turn jobs. A pre-start indicates how the upcoming turn relates to the prior turn; its typical realization is through a pragmatic marker (e.g., 'well', 'so'). The turn-constructional unit (TCU) is "a coherent and self-contained utterance, recognizable in context as 'possibly complete'" (Clayman 2013: 151). A post-completer, finally, serves as a "locus of 'current selects next'" (Sacks et al. 1974: 718); typical occupants of the post-completer slot(s) are address terms and question tags. An example featuring all three turn components is (4.1):

(4.1) Alec: Well, you should be able to get one, Joyce
(BNC: KB2 2397; corrected transcription)

In (4.1), the turn-initial marker "well" foreshadows a somewhat non-straightforward relationship of the upcoming turn with the course of action initiated by the previous turn (not shown) (cf. Heritage 2013, 2015); the core of the turn (or the TCU) is Alec's statement "you should be able to get one". Finally, the address term "Joyce" bridges over to the next turn by selecting Joyce as the next speaker.

An important strand of research in Interactional Linguistics is concerned with the question as to how resources of a language are bound up with and shaped by the organization of turn-taking (e.g., Ochs et al. 1996). The goal in this case study is to contribute to an interactionally-informed understanding of how words and word classes relate to turn-taking and the structure of turns.

The lexical class in the focus are inserts, a newly recognized broad word class, alongside function words (e.g., 'the', 'of', 'under') and lexical words (e.g., 'car', 'pathetic', 'trundle') (more on these latter classes in Chapter 5). The defining features of inserts are the following: they "do not form an integral part of syntactic structure, but are inserted rather freely in the text. They are often marked off by intonation, pauses, or by punctuation marks in writing. They characteristically carry emotional and interactional meaning and are especially frequent in spoken texts" (Biber et al. 1999: 56).

The class of inserts is an unruly class as it includes a large number of functionally diverse categories, specifically (i) interjections (both primary and secondary; see below)), (ii) greetings and farewells ('hi', 'good morning', 'bye bye'), (iii) pragmatic markers ('right', 'well', 'so'), attention signals ('hey', 'ey'), (iv) response elicitors ('alright?', 'okay?'), (v) responses ('mm', 'uh-huh', 'yeah'), (vi) hesitators ('er', 'erm', 'uh', 'um'), (vii) thanks ('thanks', 'cheers'), (viii) the politeness marker 'please', (viiii) apologies ('sorry', 'excuse me', 'pardon'), and (x) expletives ('jeez', 'jesus') (Biber et al. 1999: 93–94).

By far the largest sub-category of inserts are interjections, that is, "a class of words which are unproductive, which do not enter into syntactic relationships with other classes, and whose function is purely emotive, e.g., *Yuk!, Strewth!, Blast!, Tut tut!*" (Crystal 2003: 239). Ameka (1992) distinguishes two classes: primary interjections, that is those interjections that are "not used otherwise" (Ameka 1992: 105) such as 'Yuk' or 'ah', and secondary interjections, a category including forms that belong to other word classes and can enter into syntactic relations; for example, 'Holy Christ', 'shit', and also 'well' (which can be used syntactically; cf. Rühlemann 2018c).

With that said, what we are going to tackle in this case study is how inserts, thus defined, relate to pre-starts, that is, to those turn parts whose job it is to provide a hinge between the incipient turn and the prior turn.

4.1.2 Data and methods

We are going to use two sets of data. The first set comprises random samples of 1,000 5- to 12-word turns from BNC-C (the conversational subcorpus of the British National Corpus, which totals 4.2 m words). This first set includes 8,000 turns and 68,000 words. To analyze it, we're going to use the BNC's elaborate tagging scheme, which provides Part-of-Speech (PoS) tags for word elements in the corpus. of central interest in the present connection is the tag ITJ. This tag is assigned in the scheme to "[i]nterjection *or other isolate*" (BNCweb: <http://bncweb.lancs.ac.uk/bncwebXML/Simple_query_language.pdf>; my emphasis). That is, the tag ITJ captures primary interjections and other isolates that would not be classified as interjections but fall under the purview of inserts.

For example, as shown in Table 4.1,[19] the item 'yeah' (ranked first) would fall under Biber et al.'s (1999) subcategory of 'responses'; the item 'oh' (ranked second) would be classified as a pragmatic marker; 'hello' (ranked 12th) represents a 'greeting' and 'bye' (ranked 15th) a 'farewell'; 'hey' is used as an 'attention-getter'; and, finally, 'urgh' (ranked 19th) is a typical (emotive) interjection; note also that the least frequent ITJ are mostly non-codified ad-hoc vocalizations and would hence be classed as primary interjections.

19. The table was generated using XQUery, a programming language designed specifically for XML documents such as the BNC; for an introduction see Rühlemann et al. (2015).

Table 4.1 25 top most frequent items tagged ITJ in BNC-C

Rank	ITJ	Freq	%
1	yeah	58706	28.5374
2	oh	41555	20.2002
3	no	32988	16.0357
4	mm	21888	10.6399
5	yes	17866	8.6848
6	ah	5843	2.8403
7	ooh	3856	1.8744
8	aye	2271	1.1039
9	ha	2098	1.0199
10	eh	1936	0.9411
11	mhm	1652	0.8030
12	hello	1641	0.7977
13	dear	1356	0.6592
14	aha	1137	0.5527
15	bye	1110	0.5396
16	yep	740	0.3597
17	hey	637	0.3097
18	cor	487	0.2367
19	urgh	438	0.2129
20	hm	385	0.1872
21	ee	336	0.1633
22	tt	306	0.1487
23	hi	306	0.1487
24	hmm	303	0.1473
25	oi	279	0.1356

One glaring omission in Table 4.1 is 'well', one of the most important and most common 'interjections' or, in functional terms, pragmatic markers (in BNC-C 'well' is invariably tagged as AV0, that is, as an adverb). Based on research indicating that more than 80% of all occurrences of 'well' in conversation are instances of the pragmatic marker rather than the adverb (or adjective or 'as well') (Aijmer 2013; Romero-Trillo 2018; Rühlemann 2018c) and the afore-mentioned intrinsic attraction of pragmatic marker 'well' to turn-initial positions, the tag for 'well' in the first two word slots in the turns was corrected to ITJ; the same correction was implemented for the pragmatic marker 'so'.

The second data set analyzed is, again, the 10-word sample from BNC-C used in the first sample collection but this time the 1,000 turns were inspected line by line and, if necessary, context by context and manually coded for pre-starts (and, for sake of completion, post-completers). The 10-word turn length was selected for this procedure as 10 words is likely the average length of turns in conversation (cf. Rayson 1997; Rühlemann 2018a). In annotating this subsample, not only concordance lines were inspected in their larger sequential contexts but also, where available, audio files of the larger contexts were accessed.

What was coded as pre-start, i.e., as "relating to the prior turn"? This is a critical question that needs to be addressed in good detail.

To start with, what was *not* coded as pre-start? Sacks et al. (1974) note that a turn can display a relationship with the prior turn through 'tying' devices, more commonly known in linguistics as 'cohesive' devices (Halliday & Hasan 1976) such as, for example, co-reference as instantiated by anaphoric pronouns; other types of cohesive ties include substitution, ellipsis, and lexical cohesion (cf. Halliday & Hasan 1976). These ties have in common that their "interpretation is dependent on that of another [element]. The one presupposes the other in the sense that it cannot effectively be decoded except by recourse to it" (Halliday & Hasan 1976: 4). Co-reference, substitution,ellipsis, or lexical cohesion were not used as criteria for coding pre-starts. The reasons are two-fold: these ties are syntactically integrated into the turn and they are positionally rather versatile. Personal pronouns, for instance, can, in principle, be placed anywhere in the turn. Pre-starts, by contrast, normally occur in initial turn positions preceding the TCU.[20] Second, as a turn *part* that is claimed to do a separate turn job it cannot be intertwined syntactically in the realization of the turn. To do a job in its own right, it needs to be syntactically free, i.e., detached from the syntax of the turn or at least detachable from it.

So, what *was* coded as pre-start in the present analysis? One type of cohesion that was not mentioned in the above list of types of cohesion is conjunction. Conjunctions do meet the criteria for pre-starts: they tend to occur in an initial position in the clause and, while not detached from it, they *can* be decoupled from the clause they introduce without altering the proposition expressed in the clause. Their detachability derives from the fact that they are in fact discourse-deictics: "[w]hat they seem to do is indicate, often in intricate ways, just how the utterance that contains them is a response to, or a continuation of, some portion of the prior discourse" (Levinson 1983: 88). More recently, conjunctions have been recognized as turn beginnings in CA work (Heritage & Sorjonen 2018: 2). So the first class of items to

20. Note that some pre-starts did occur in late positions in the turn. A case in point is the marker 'though', as in "i can never understand why you took that part though", which reflects a somewhat contrastive stance toward the previous turn but is normally positioned in turn-final positions.

be coded as pre-starts involves conjunctions such as 'and' signaling continuation, as in (4.2), 'but' introducing some contrast, as in (4.3), and 'cos' indicating, in most cases in conversation, not reason but continuation and elaboration, as in (4.4) (cf. Schleppgrell 1991).

(4.2) **and** that i 'm in love with him or something!

(4.3) **but** it does n't tell you what kind it is

(4.4) **cos** if it rains that wo n't come on outside.[21]

The second class of items identied as pre-starts includes those pragmatic markers that index, in some way, a relation to the prior turn(s). These markers include, in the present dataset, the marker 'so', often used to introduce not a consequence but a new sequence or to close down a sequence, as in (4.5) (cf. Fraser 1990; Buysse 2012). Also, perhaps most prominently, the marker 'well' features among pre-starts; in initial position, it serves as a 'warning particle' (Levinson 2013: 108) foreshadowing a 'dispreferred', that is, a turn that, to some degree, falls short of aligning with the course of action implemented in the prior turn (Schiffrin 1985; Schegloff & Lerner 2009; Heritage 2013, 2015). Another frequently occurring marker is 'oh', a (cognitive) change-of-state token marking the reception of unknown or inapposite information (Heritage 1998). Also included, but far less prominent in the data, is the phrase 'I dunno', a reduced form of 'I don't know'; research has shown that rather than admit the speaker's insufficient knowledge, 'I dunno' frequently prefaces disagreements (cf. Diani 2004; Sacks 1987); by contrast, when 'I don't know' was used syntactically, as in (4.9), it was not coded as a pre-start.[22]

---

21. To see how the transcripts were coded for turn structure, consider this example turn: "cos if it rains that wo n't come on outside." In the dataframe from which the case study was elaborated, each word in the ten-word turn was captured by its PoS tag, that is, in this case: `CJS CJS PNP VVZ CJT-DT0 VM0 XX0 VVI AVP-PRP AV0`. Each tag was stored in a separate column, labeled `c5_w1, c5_w2`, and so forth until `c5_w10`. Now, the conjunction "cos" in this turn does the job of a pre-start. To reflect the role of "cos" (as of any other items that were identified as pre-starts or post-completers), a new set of ten columns was added to the dataframe, labeled `struct1`, `struct2`, etc. up to `struct10`. In the case of our example turn, "cos" was coded as `"conjunction"` in column `struct1`, whereas all other PoS tags were copied unaltered into the remaining `struct*` columns.

22. Post-completers in the sub-sample include the following types: canonical question tags (e.g., 'doesn't she', 'is there', etc.), invariant tags ('you know', 'you see', 'eh', 'yeah?'), address terms such as first names and kinship terms, tails (e.g., they 're easy to steal though are n't they paintings?), the request marker 'please', and dangling 'so'.

(4.5) **so** whether she 'd like some , i do n't know.

(4.6) **well** that would n't please them next door would it ?

(4.7) **ooh** that 's geoff 's mummy 's birthday this week

(4.8) **i du n no**, you ca n't see his head

(4.9) **i do n't know** whether she was just finishing then

Another class of items counted as pre-starts are, in Biber et al.'s (1999) terminology, response items, including 'yeah', 'mm', and 'I know':

(4.10) **yeah**, it ai n't worth him passing his sodding test

(4.11) **mm**, some of, a lot of them are like that

(4.12) **i know**, he said i dare n't go near that

Quite often, pre-starts exhibit a complex structure as they consist of two or even more items, thus forming sequences of markers (cf. Heritage 2015):

(4.13) **so, yeah** i 'll go , i 'll go and check.

(4.14) **well i do n't know**, it 's fairly important geoff

(4.15) **oh aye yeah, well** i am rough are n't i?

Table 4.2 lists the top 15 most frequent pre-starts in the subsample.

Table 4.2 shows that the marker 'well' accounts for 16% of all pre-starts in the sample; further, the vast majority of the top most frequent pre-starts are interjections, both of the primary (e.g., 'oh') and the secondary type (e.g., 'well'); the table also attests to how commonly pre-starts are realized through sequences of inserts (e.g., 'yeah well', 'yeah but', etc.).

The graphic used to visualize the location of ITJ and, respectively, pre-starts in turns is based on the scatter plot, a plot type already encountered in Chapters 3 and described in more detail in Chapter 9. Unlike in typical scatter plots, the data to be plotted will not be numeric. Rather, to visualize ITJ locations in turns, we will plot character symbols. As the symbols designate the location of an item of interest (ITJ and, respectively, pre-start) within a unit of observation (the turn), thus allowing the observer to see how the item of interest is dispersed across the unit, I refer to this plot type as location plot. I am going to present two figures, one for each dataset. The first figure will have multiple panels, the second a single panel. The first figure locates ITJs in each of the 1,000 turns in each sub-sample of various turn lengths, the second highlights the location of pre-starts and post-completers in each of the 1,000 turns of

Table 4.2 15 most frequent pre-start types in subsample (inserts marked in blue color)

	Pre-start type	%
1	well	16.23
2	and	14.14
3	no	10.73
4	but	9.69
5	yeah	6.54
6	oh	5.50
7	cos	4.97
8	so	4.97
9	yes	2.36
10	yeah but	2.09
11	yeah well	1.83
12	i know	1.57
13	ah	1.31
14	oh well	1.31
15	oh yeah	1.31

the manually annotated sub-sample. In the multiple-panel plot I am going to deploy the colors used in X-rays, black for ITJ and grey for any other words. The second plot, by contrast, contains more than two variables, which is why a larger and more colorful palette will be used.

4.1.3 Results

The following series of panels shows the location of ITJ in turns in eight samples with distinct turn lengths (from five words to 12 words). As noted, black stripes denote ITJ.

The picture is consistent for each of the eight graphs in Figure 4.1: there are far more black stripes in the first 'bar' – the first position in the turn – than in any other 'bar', i.e., any other position in the turn. The meaning of the graphs is straightforward: items tagged ITJ overwhelmingly occur in the very first position in the turn. As noted, this is somewhat to be expected given the body of research on positioning of interjections and pragmatic markers (cf. above).

So, let's move to address our research question: how are inserts correlated with turn structure? Again, being items preceding the core of the turn, pre-starts will be expected to occur turn-initially. The question, then, is not *where* will pre-starts occur in the turn

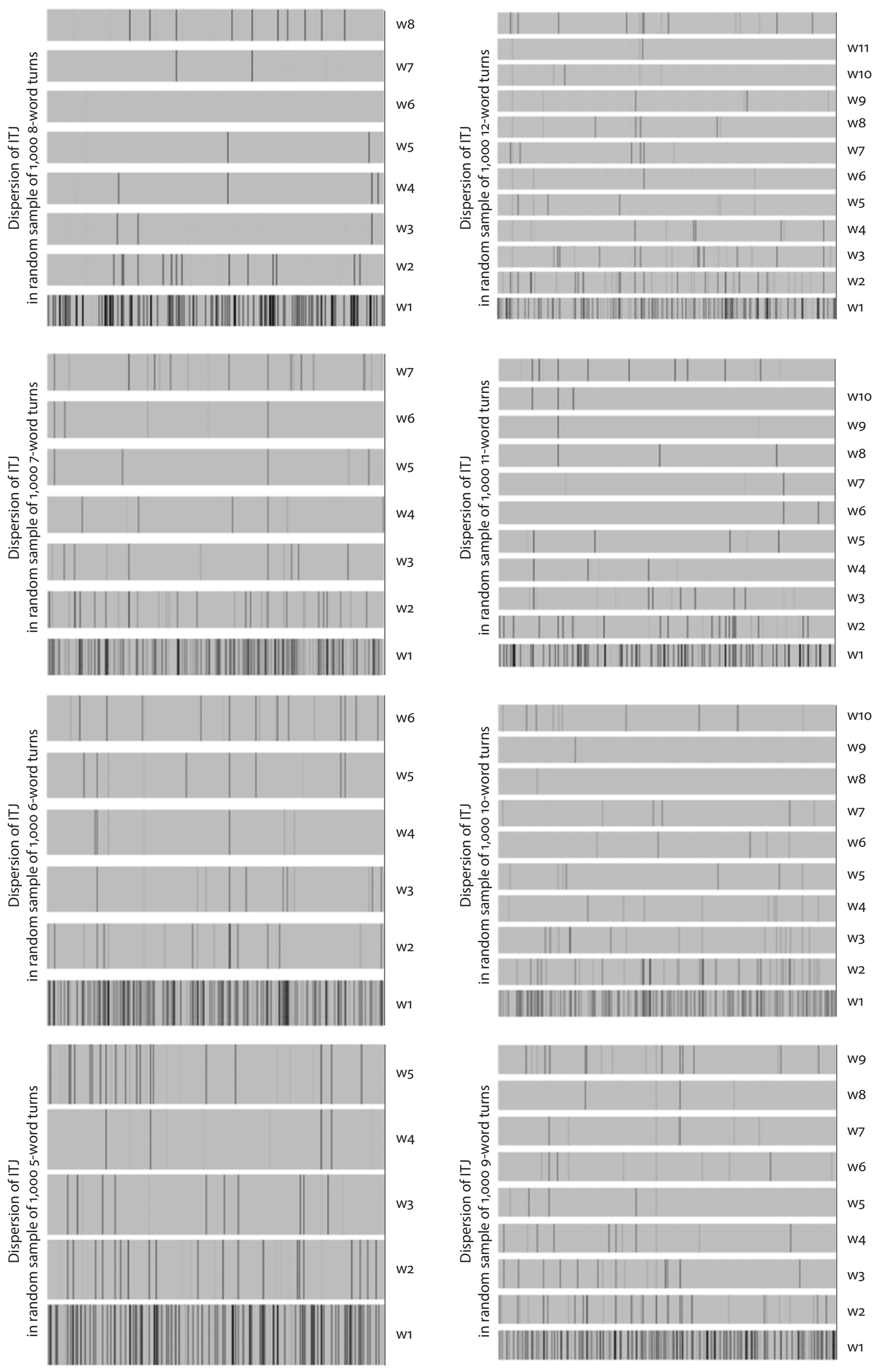


Figure 4.1 Positions of ITJ in randomly sampled 5- to 12-word turns

but rather, *how many inserts will remain unaccounted for once pre-starts are taken into account?*

To start with, the following plot shows the same distribution of ITJ in a 10-word turn sample that was part of Figure 4.1; just the colors have been changed from black (for ITJ) and grey (for any other item) to red and, respectively, white.

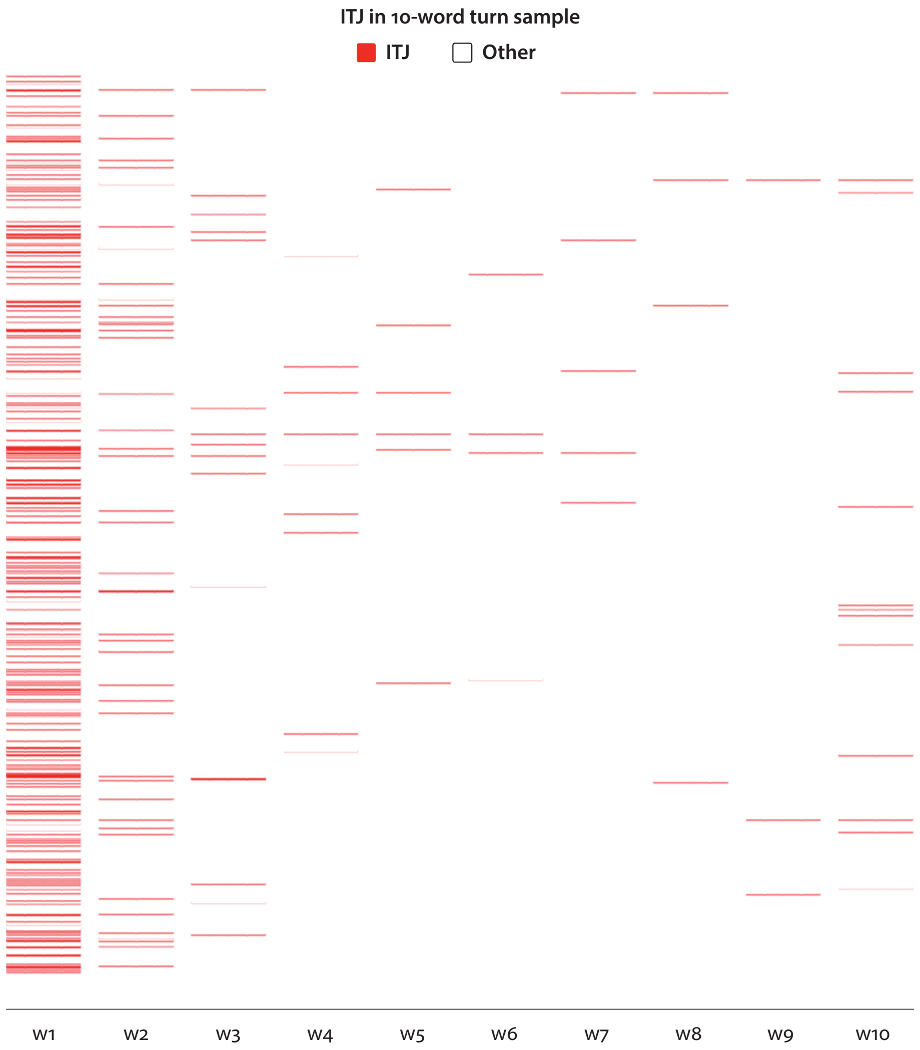


Figure 4.2 Positions of ITJ in turns in 10-word turn subsample

Now, observe what happens when pre-starts (and also post-completers) are plotted into the graph:

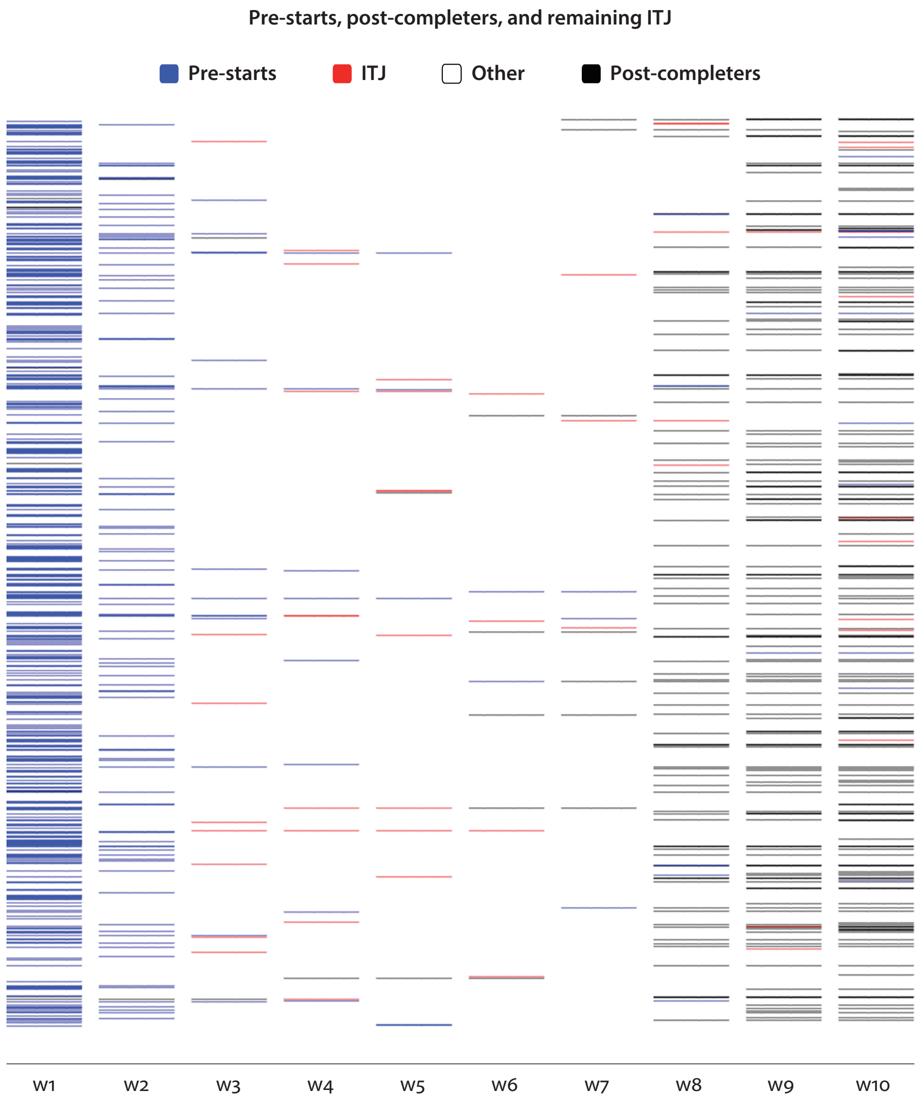


Figure 4.3 Positions of pre-starts, post-completers, and remaining inserts in turns in 10-word turn subsample

How many red stripes have survived? That is, how many inserts are not at the same time pre-starts? The plot shown in Figure 4.3 suggests the number of survivors is exceedingly small: almost all red color has disappeared from the turn-first and the turn-second positions; those few red lines that have remained in the plot are found in turn-medial and turn-final positions. This is clear visual evidence that inserts have an intrinsic association with pre-starts. This visual impression is conclusively confirmed by numerical descriptive statistics.

As can be seen from Table 4.3, roughly 40% of all turns in the sample have a pre-start; that is, almost every second turn is prefaced by a pre-start indicating how the incipient turn relates to the prior turn; post-completers are far less frequent (with 15%) and the proportion of turns with both peripheral turn parts is 6.6%.

Table 4.3 Frequencies and proportions of turn parts in subsample

	Turns with	**%**
Pre-starts	403	40.3
Post-completers	153	15.3
Pre-starts AND Post-completers	66	6.6

The most important piece of information, given our research question – how are inserts correlated with turn structure? –, is shown in Table 4.4: 85.88% of all inserts in the sample are consumed by pre-starts; by contrast, the respective proportion for post-completers is just 0.87%. That is, while inserts are virtually absent from post-completers, and only occasionally occur in turn-medial positions – that is, as part of the TCU – inserts thrive in pre-starts.

Table 4.4 Frequencies and proportions of ITJ in turn parts

	Freq	**%**
ITJ in Pre-starts	298	85.88
ITJ in TCU	46	13.25
ITJ in Post-completer	3	0.87
total	**347**	**100**

4.1.4 Discussion and conclusions

This analysis produced a clear finding: the overwhelming majority (85+%) of inserts are (part of) a pre-start. This finding is non-trivial: it suggests that a whole class of words – *inserts* – exists because it plays a specific role in turn-taking. How to make sense of this? To address this question, it is instructive to consider the notion of preference.

Turns-at-talk "have built into their design an approximate trajectory for the sequence of which they are a part" (Liddicoat 2007: 112; cf. Sacks 1987). That is, they do not merely perform an action, such as asking for information, but also set up a bias (an expectation) for how participants should respond (cf. Bögels et al. 2015: 2). This is where preference comes into play. A response (or next) turn that is aligned with the bias of the prior is *preferred* whereas a turn that disaligns, in some way and to some degree, with the prior turn is *dispreferred* (for discussions of preference, cf. Pomerantz & Heritage 2013; Rühlemann 2018a: Chapter 6). So, in essence, a next turn can respond, or relate, to the prior in two basic ways:

> [i]n 'unmarked' movement [from prior turn to next turn], next turns are congruent with the understandings, expectations and projections that were established in the previous turn, or sequence of turns. In 'marked' movement, there are departures from some of these understandings, expectations and projections (…). Turn-initial particles are often implicated in these departures because (…) they represent a first possible opportunity to indicate that a departure is underway. (Heritage 2015: 89)

In unmarked movement, then, the next turn is preferred as it aligns with the bias of the prior whereas in marked movement the next turn is dispreferred as it departs from the bias of the prior. The 'turn-initial particles' that are often implicated in these departures are Sacks et al.'s (1974) pre-starts. So, speakers begin turns with a pre-start to signal (degrees of) alignment/disalignment with the course of action taken by the prior turn. This is perhaps most obvious for the insert 'well', the most frequent pre-start in the data. It issues a warning that what is underway is a movement that is not, or not fully, 'congruent with the understandings, expectations and projections that were established in the previous turn'. Schegloff & Lerner refer to this departure from the understandings, expectations and projections of the prior turn as "nonstraightforwardness in responding" (Schegloff & Lerner 2009: 91). Schiffrin analyzes the departure as "a temporary suspension (…) for immediate coherence of a response" (Schiffrin 1985: 648; see also Schiffrin 1987: 323). The departure manifests in many different ways. In extracts (4.16)–(4.17), two of these manifestations will be illustrated.

In (4.16), the participants are talking about Betty's 10-year-old daughter. In lines 1–3, Betty and Rose note that Betty's daughter pinched some biscuits. This action is noted in remarkable unison, with Rose precisely echoing Betty's words and the echo onset timed as in a musical canon. So the two women have a firmly established shared understanding of the girl's 'deed'. Following a pause Betty produces the 'well'-prefaced turn "we' she didn't pinch them but she ate them". She thereby explicitly contradicts their mutual agreement that the girl had pinched the biscuits and replaces 'pinch' with 'ate', indicating that the girl not only stole the biscuits, but ate them. That is, Betty's 'well'-prefaced turn departs from their previously established mutual understanding to proffer an updated understanding of the girl's action.

(4.16)

```
1    Betty:   Ten years old [pinching the biscuits:::]
```

```
2     Rose:                  [ten   years   old   pin]ching the biscuits.
3             he he he.
4             (1.4)
5     Betty:  we' she didn't pinch them but she ate them
6     Rose:   hhhe he he
                              (BNC-C: KBE, 6542-6545; corrected transcription)
```

In extract (4.17), a piano teacher praises his student Caroline for practicing despite difficult circumstances: "think (it's) admirable that you're still doing it". This puts Caroline into a dilemma; agreement would effectively amount to self-praise, which would collide with a constraint against self-praise. Disagreement would be a stark departure from her teacher's assessment. The solution she opts for is praise-downgrade (cf. Brown & Levinson 1987: 39); that is, by responding "we:ll (it's) I HAVe to have (.) something (.) to do by myself." she casts her practicing as something she just can't help doing:

```
(4.17)
1     UNK:    No I, no, I mean you REAlly respond very well
2             that Tchaikovsky was lovely
3             I understand at the present (.) you:u (.) can't do as much
4             I think (it's) admirable that you're still doing it
5     Caro:   we:ll (it's) I HAVe to have (.) something (.) to do by myself.
                              (BNC-C: KBH, 5036-5037; corrected transcription)
```

Extracts (4.16)–(4.17) illustrate just some ways in which 'well' may indicate 'marked movement' from one turn to the next, initiating interactional departures (or non-straightforwardness or lack of coherence) in response turns. The marker's work as a warning that such movement is underway manifests in many other ways too (for examples, cf. Heritage 2015; Rühlemann 2018a). The important point in the present connection is that the *insert* 'well' does this work as a *pre-start*.

So pre-starts provide an early indication of (degrees of) alignment/disalignment with the course of action taken by the prior turn. This has implications for action ascription, that is, for how the interlocutor interprets the pre-start-prefaced turn: the pre-start "invoke[s] a frame of interpretation for the rest of the linguistic content of the utterance" (Gumperz 1996: 379). That interpretative frame may be quite wide, not delineating a specific action but rather narrowing down the range of possible actions. Seen in this way, pre-starts implement the "front-loading bias" (Levinson 2013: 112), a fundamental bias toward inserting cues to action type early in the turn to facilitate action ascription.

In conclusion, pre-starts are highly common in turns-at-talk; almost every second turn is prefaced by a pre-start. Moreover, pre-starts provide the prime ecological niche for inserts, one of three major word classes of the lexicon: more than 85% of all inserts are consumed by pre-starts. This suggests that inserts exist as an adaptation to turn-taking, lending support to Schegloff's observation that "some of the most fundamental features of natural language are shaped in accordance with their home environment in co-present interaction, as adaptations to it, or as part of its warp and weft" (Schegloff 1996: 54).

4.2 The location plot in the case study

The location plot in Figure 4.3 visualized the location of pre-starts, post-completers and all remaining inserts not consumed by pre-starts and/or post-completers. How was the figure programmed? The data underyling the figure contained a lot of manual annotation. The dataset we are going to start out from will contain this annotation already (whereas in the actual case study, it needed to be implemented there). The data called "Chapter4_Casestudy.txt" can be downloaded from the companion website; see "Chapter4_Casestudy_Code.R" for the code.

As always, we'll start by reading-in the data into R and store it as `ts` (for turn structure):

```
> ts <- read.table("[Your path]/Chapter4_Casestudy.txt", header=T,
quote="", sep="\t", fill=F)
```

Let's inspect this dataset; to do so we draw a random sample of five rows from the 1,000 rows using the `sample()` function:

```
> ts[sample(1:nrow(ts), 5),]
   File  Speaker                                                  Turn
146  KBF    PS04U         but we got , we got what was on our list .
594  KD2  KD2PSUNK  yeah , i think , well that could be a doggy there .
859  KP1    PS50U those are n't ready , they 've only just gone out
559  KD0    PS0HV there 's no raw plugs in that packet , is there ?
906  KPF    PS54T really , cos what time is the train that you got ?

        w1   w2    w3     w4     w5    w6     w7     w8    w9    w10
146    but   we   got     we    got  what    was     on   our   list
594   yeah    I think   well   that could     be      a doggy  there
859  those  are   n't  ready   they   've   only   just  gone    out
559  there   's    no    raw  plugs    in   that packet    is  there
906 really  cos  what   time     is   the  train   that   you    got
```

```
         struct1     struct2  struct3  struct4  struct5  struct6
146  conjunction         PNP      VVD      PNP      VVD      DTQ
594  marker              PNP      VVB      AV0      DT0      VM0
859  DT0                 VBB      XX0      AJ0      PNP      VHB
559  EX0                 VBZ      AT0      AJ0      NN2      PRP
906  AV0         conjunction      DTQ      NN1      VBZ      AT0
    struct7 struct8 struct9 struct10
146     VBD     PRP     DPS      NN1
594     VBI     AT0     NN1      AV0
859     AV0     AV0     VVN      AVP
559     DT0     NN1     tag      tag
906     NN1     CJT     PNP      VVD
```

We see that `ts` contains 23 columns, one for `File`, `Speaker`, and `Turn` each, and 10 for the words per turn position (`w1`, `w2`, etc.); the last ten columns are the ones of central interest, those labeled `struct1`, `struct2`, etc.: they contain the Part-of-Speech (PoS) tags for the respective words in `w1`, `w2`, etc. but also also other tags, namely, in row 594 in column `struct1`, the code `marker` used for the word "yeah" in first position in the turn "yeah , i think , well that could be a doggy there ." Or, to cite another example, you find the code `tag` twice in line 559 in columns `struct9` and `struct10` used for the question tag "is there?" in the turn "there 's no raw plugs in that packet , is there ?". These codes designate pre-starts and, respectively, post-completers, whose location in turns we aim to show in the location plot.

When uploading files with lots of variables it is always advisable to check how R has read them in. Let's do this for the columns of central interest (`struct1`, `struct2`, etc.:

```
> str(ts[,14:23])
'data.frame': 1000 obs. of 10 variables:
$ struct1  : Factor w/ 51 levels "address","AJ0",..: 30 30 6 20 12 5 13 30 20 30 ...
$ struct2  : Factor w/ 57 levels "address","AJ0", ..: 41 35 28 11 28 2 11 40 16 44 ...
$ struct3  : Factor w/ 66 levels "address","AJ0",..: 65 30 42 30 37 23 30 65 42 62 ...
$ struct4  : Factor w/ 68 levels "AJ0","AJ0-AV0",..: 63 56 6 56 6 45 66 63 1 39 ...
$ struct5  : Factor w/ 70 levels "address","AJ0", ..: 34 65 5 65 24 34 17 29 26 65 ...
$ struct6  : Factor w/ 69 levels "address","AJ0",..: 40 3 21 14 17 46 57 57 29 3 ...
$ struct7  : Factor w/ 69 levels "address","AJ0",..: 62 25 25 34 48 65 68 65 58 25 ...
$ struct8  : Factor w/ 66 levels "AJ0","AJ0-AV0",..: 39 34 35 46 5 11 61 39 61 38 ...
$ struct9  : Factor w/ 70 levels "address","AJ0",..: 45 24 34 69 17 6 9 65 32 38 ...
$ struct10 : Factor w/ 73 levels "address","AJ0",..: 8 8 32 68 8 17 8 8 10 44 ...
```

As is frequently the case when R encounters characters variables, it treats them as factors. For the next steps, however, it is necessary to convert them to character variables, which is done with the help of `lapply()` and, inside it (as its second argument) the function `as.character()`:

```
> ts[,14:23] <- lapply(ts[,14:23], as.character)
```

Let's check again with `str()` whether the conversion was successful:

```
> str(ts[,14:23])
'data.frame':   1000 obs. of 10 variables:
$ struct1 : chr "PNP" "PNP" "AV0" "marker" …
$ struct2 : chr "VDD" "UNC" "PNP" "CJS" …
$ struct3 : chr "XX0" "PNP" "VBZ" "PNP" …
$ struct4 : chr "VVI" "VM0" "AV0" "VM0" …
$ struct5 : chr "PNP" "VVI" "AT0" "VVI" …
$ struct6 : chr "VBD" "AT0" "NN1" "DT0" …
$ struct7 : chr "VVG" "NN1" "NN1" "PNP" …
$ struct8 : chr "TO0" "PRF" "PRP" "VDB" …
$ struct9 : chr "VBI" "NN1" "PRF" "XX0" …
$ struct10: chr "AV0" "AV0" "NP0" "VVI" …
```

It was. So we can proceed to the next step; we will need to re-organize the data such that all variables required for the plot are in one column. One variable required but not (directly) contained in `ts` is the position of the words in the turns – that variable we will have to create from scratch. We know we have ten unique positions in the 10-word turns, one for each word, so we can identify the positions using a sequence of integers from 1 to 10 and simply repeat these integers 1,000 times to obtain the variable `position`:

```
> position <- c(rep(1,1000),rep(2,1000), rep(3,1000), rep(4,1000),
+              rep(5,1000), rep(6,1000),rep(7,1000), rep(8,1000),
+              rep(9,1000), rep(10,1000))
```

Alternatively, and more elegantly, we can define `positions` with a `for` loop: here it's important to set up the vector `position`*before* the `for` loop (although, at that point, it evaluates to `NULL`) so R knows where to store the data in:

```
position <- c()
for(i in 1:10){
  position <- c(position, rep(i, 1000))
}
```

Next, we need a variable for the turn structural elements in `struct1`, `struct2`, etc.:

```
> element <- c(ts$struct1, ts$struct2, ts$struct3, ts$struct4, ts$struct5,
+ ts$struct6, ts$struct7, ts$struct8, ts$struct9, ts$struct10)
```

To achieve the same result in a much simpler fashion, we can use `unlist()`:

```
> element <- as.character(unlist(ts[14:23]))
```

We combine these two variables in a dataframe:

```
> df <- data.frame(position, element)
```

And yet a third variable will be handy for the plot: a variable indicating whether a token in the vector `df$element` is (part of) a pre-start, the TCU, or a post-completer and whether the token is an insert (ITJ) not consumed by a pre-start or a post-completer. In other words, we want to group all values in `df$element` into four larger groups: (i) pre-start, (ii) TCU, (iii) post-completer, or (iv) ITJ. To do this grouping, we will use three nested `ifelse()` statements. We know that the coding for pre-starts has two levels: `marker` and `conjunction`, so we will put these two levels separated by the alternative marker `|` into the first `ifelse()` clause. Post-completers have six levels: `address`, `tag`, `trans_marker`, `requ_marker`, `looseness_marker`, and `tail` so we will put these levels into the second `ifelse()` clause. Inserts have a single level, namely `ITJ`, for which we will test in the third `ifelse()` clause; all values in `element` for which none of these conditions hold will be labeled `TCU`:

```
df$turnpart <- ifelse(df$element=="marker" |
                      df$element=="conjunction", "pre",
                      ifelse(df$element=="address" |
                             df$element=="tag" |
                             df$element=="trans_marker" |
                             df$element=="requ_marker" |
                             df$element=="looseness_marker" |
                             df$element=="tail", "post",
                             ifelse(df$element=="ITJ","ITJ","TCU")))
```

What does this dataframe look like?

```
> head(df)
  position elements turnpart
1        1      PNP      TCU
2        1      PNP      TCU
3        1      AV0      TCU
4        1   marker      pre
5        1      CJT      TCU
6        1      AT0      TCU
```

We see that the value `marker` in `df$element` has been re-labeled as `pre` in `df$turnpart` and the other five `df$element` values as `TCU`. So far, so good: we can now turn to the plot.

Some preparation is required: we have to define a function that determines the plotting dimensions, that is, the number of rows and the number of columns in the plot. We will call this function `plotdim`; it takes two arguments, `N_Slots` (number of slots) and N_Rows (number of rows). Inside the function we compute the number of repeats, N_Repeats, and the total number of rows, N_total_Rows, as well as a dummy dataframe in which the first variable, rows, is the sequence from 1 to N_ total_Rows, and the second variable is the sequence from 1 to N_Slots:

```
> plotdim <- function(N_Slots, N_Rows){
+
+     # Compute number of repeats:
+     N_Repeats <- round(N_Rows / N_Slots, 0)
+     N_total_Rows <- N_Repeats * N_Slots
+
+     # Create dummy data:
+     dummy <- data.frame(rows = 1:N_total_Rows, slots = 1:N_Slots)
+     head(dummy); tail(dummy)
+
+     return(dummy)
+     }
```

This is a useful function in that it accommodates any number of rows and columns and can thus be used for any other dataset, even with odd numbers. We have 10 columns and 1,000 rows, so we will input these numbers into our function and store them in the vector dimensions:

```
> dimensions <- plotdim(10, 1000)
```

Now, we seek to make the plot as large as possible so we set small margins using the function par() and its argument mar:

```
> par(mar = c(2,1.5,1,1.5))
```

Next we use the vector dimensions to set up the plot's architecture (but not yet any data) by plotting dimensions$rows against dimensions$slots; the argument type = "n" embodies the instruction not to plot any data. In frame = F we suppress the default frame around the plot. Also, we do not wish to plot any axes, which is achieved by axes = F. Further, we define the title of the plot in main and reduce, for space reasons, the font size with cex.main = 0.9 (1 being the default):

```
> plot(dimensions$rows ~ dimensions$slots,
+      type = "n",
+      frame = F,
+      axes = F,
+      ylab = "", xlab = "",
+      main = "Pre-starts, post-completers, and remaining ITJ",
+      cex.main = 0.9)
```

Next we define the x-axis using axis(); we use 1 to select the x-axis (2 would be for the y-axis, 3 for a secondary x-axis above the plot, and 4 for a secondary y-axis on the right of the plot); the argument `at` determines where to place the ticks and the labels (namely, here, at each index of the sequence 1:10); also, we define the labels for the axis using `paste()` to attach the letter `w` to each integer in the sequence `1:10` without any separation (`sep = ""`) between letter and number:

```
> axis(1,
+       at = seq(1:10),
+       labels = paste("w",1:10, sep = ""),
+       cex.axis = 0.9)
```

Now we can fill in the data. We do this by defining a `for()` loop for each unique value in `df$position` (of which there are ten) and instruct R to execute the `text()` function in the following way. First we determine *where* we want to have text printed; the location of `text()` is defined by values on the x-axis and values on the y-axis. As we want R to loop over the ten positions on the x-axis, we use the index `i` as the set of values for the x-axis; further, we define the total number of rows as the values on the y-axis. What text we want is defined in quotation marks; here, we use a set of underscores to obtain a straight line for all items to be printed. Finally, we need to distinguish the items of interest – pre-starts, post-completers, and remaining ITJ, which are stored in `df$turnpart` – and we do so by assigning different colors to each of them in nested `ifelse()` clauses; importantly, we need to subset `df$turnpart` by the index `i` so that the target items get printed in the right position:

```
> for(i in unique(df$position)){
+    text(i, 1:length(dimensions$rows), "_____",
+          col = ifelse(df$turnpart[df$position==i]=="ITJ", "red",
+                ifelse(df$turnpart[df$position==i]=="pre", "blue",
+                ifelse(df$turnpart[df$position==i]=="post",
               "black","white"))))
+ }
```

Don't expect this step to materialize on your screen in split seconds – the `for` loop and the `text()` function take time so the plot will build up slowly, position by position.

Finally, we have to add a descriptive legend. We use `legend()`, first defining its x- and y-coordinates, then ordering its elements horizontally with `horiz = T`, defining the labels to be used in the argument `legend`, and using the argument `fill` to insert the desired colors into the legend boxes; finally, we suppress the default box around the legend with `bty = "n"` – and we're done!

```
> legend(1.5, 1050,
+         horiz = T,
+         legend = c("Pre-starts", "ITJ", "other", "Post-completers"),
+         cex = 0.7,
+         fill = c("blue", "red", "white", "black"),
+         bty = "n")
```

4.3 Task: Internal structure of complex pre-starts

Pre-starts are often composed of more than one item: they are compositions of inserts. In this Section the task is to investigate whether this internal composition of complex pre-starts is random or whether complex pre-starts "participate in a canonical ordering – a linear syntax" (Heritage 2015) if the cluster occurs turn- initially and involves the marker 'well'. An example is (4.18), where the turn starts with the cluster "No, oh well":

(4.18) **No, oh well** let's hope he 'll get better (BNC: KB0118)

In this Task, we will focus on turns in which by far the most frequent insert 'well' and the fourth most frequent insert 'oh' *co-occur*. The question to be addressed is in what particular order do the two inserts occur when they co-occur in a turn?

On the companion website, you find a very large dataset comprising all 77,850 turns with 7–12 words from the BNC-C; the dataset is called "Chapter5_Task.txt". The steps to take are the following:

1. Read-in the data and inspect the structure of the variables using `View()` and `str()`.
2. Set all relevant variables to lower character using `tolower()`.
3. Make a subsample containing only those turns in which both 'well' and 'oh' co-occur. Subsample in two steps, each time using the function `grepl()` to define the pattern to match: first select only those turns that contain 'well' defining this pattern thus: `"\\bwell\\b."` (`\\b` marks word boundaries and is used to make sure you don't match fragments of longer words, such as 'well-behaved'). Then, filter this subsample even further by selecting only those turns that contain the pattern `"\\boh\\b."` for 'oh'. At this point your dataset will have shrunk considerably: there should be only 376 turns left!
4. Reorganize the data: (i) put all the words in one column `word`, (ii) create a column for the words' turn position, calling it `position`, and (iii) bind the two vectors together in a dataframe `df`.
5. In that dataframe create a new variable `col` for the colors used in the location plot: assign suitable colors for 'oh', 'well', and any other insert you may be interested in by using nested `ifelse()` statements.
6. Care needs to be taken with the numerous `NA` values (these are a result of the fact that the dataset comprises turns of different sizes). It is advisable to set `NA` to a color of your choosing in a separate step, following this structure: `df$col[is.na(df$word)] <- "white"`.
7. Next determine the plotting dimensions using the above illustrated function `plotdim` and defining the dimensions using the correct numbers , namely 12 (for the maximally 12 words on the x-axis) and 376 (for the 376 turns in your subsample on the y-axis) thus: `dimensions <- plotdim(12, 376 )`.
8. Now you're ready to start plotting. First set up the plotting region along with the title suppressing the x-axis and the default frame with `axes = F`and `frame = F`.

Then define the x-axis with labels for the word slots, e.g., "w1", "w2", etc. Third, visualize the location of the inserts into the 12 slots in the plot in a `for()` loop for the unique values in `df$position` in this way:

```
for(i in unique(df$position)){
  text(i, 1:length(df$word), "____", cex = 1,
         col = df$col[df$position==i]
)
```

9. Finally, include a legend to provide the key to your color codings.

This procedure should produce a graph that, depending on what graphical parameters and inserts you have chosen to highlight, should roughly look like this:

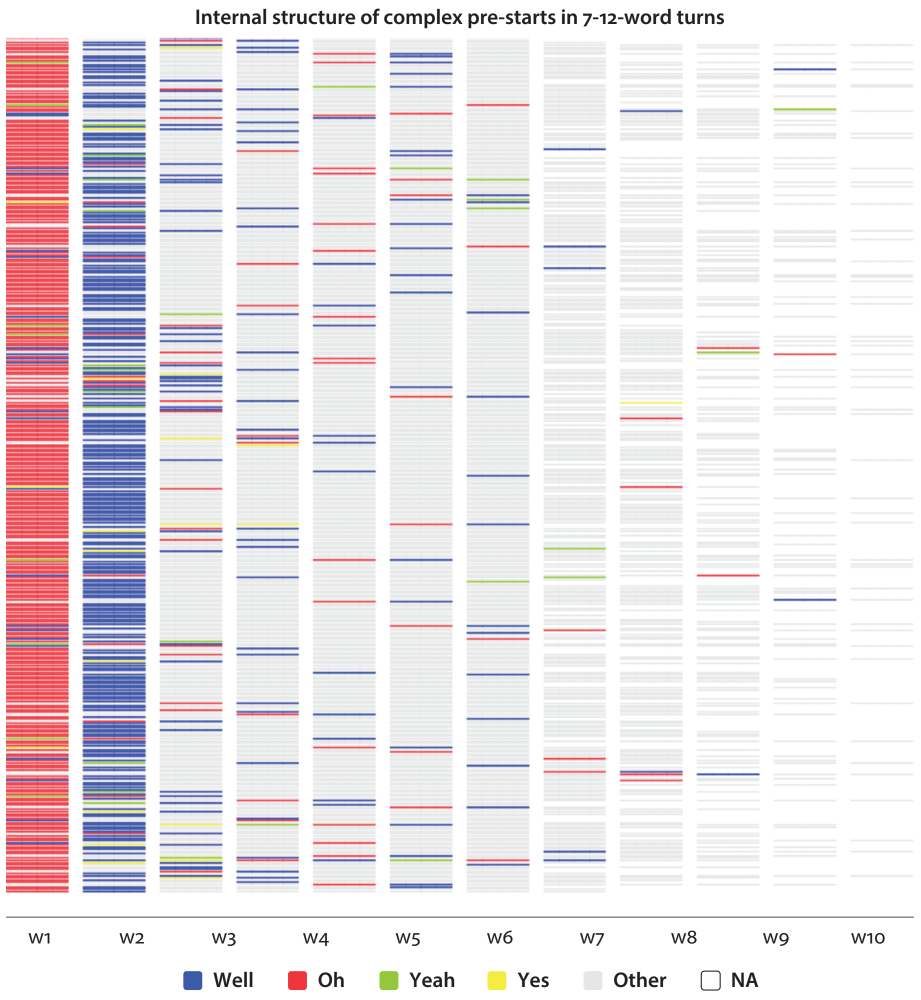


Figure 4.4 Location of some inserts in turns where 'oh' and 'well' co-occur

There is obviously a lot of detail in the graph. But one striking regularity jumps out: in complex pre-starts, 'oh' virtually always precedes 'well'. This order may reflect what may be called the indexical direction of 'oh' and 'well': 'oh' is a change-of-state token indicating that the speaker "has undergone a change in his or her locally current state of knowledge, information, orientation or awareness" (Heritage 1984: 299). Its indexical direction is hence "backward looking" (Heritage 2018: 156), that is, directed toward the previous turn and its effect on the speaker's state of 'knowledge, information, orientation or awareness'. 'Well', by contrast, is "largely forward looking" (Heritage 2018: 156) in direction, that is, toward the speaker's incipient turn, by alerting the hearer that this turn will not nonstraightforwardly align with the previous turn(s). If, then, 'oh' and 'well' co-occur to form a complex pre-start, their linear ordering – first 'oh', then 'well' – reflects the speaker's cognitive progress from first taking in and reacting to what was just said to then producing a response to it. Under this interpretation, the two items retain the indexical potential they convey individually. Note that this interpretation is not undisputed: others have argued that the meaning of insert clusters may conventionalize into idioms whose meaning is more than or different from the meaning of the sum of the composite parts (for a discussion of the issue of compositionality of complex inserts, see Haselow 2019). Schourup (2001: 1031), for example, argues that the combination 'oh well' has a non-compositional meaning and indicates resignation. My hunch is that the two views are not not mutually exclusive: 'oh well' can be used either compositionally, as an idiom, or non-compositionally, in which case the two inserts keep their individual meanings, and the different uses are likely marked prosodically, with compositional 'oh well' falling under a single intonational contour and spoken with a falling tone, and non-compositional 'oh well' spoken with two separate intonation contours and two different tones – but that's a hypothesis that needs to be tested in future work.

Chapter 5

Barplots

5.1 *Case study*: Turn structure – Turns and lexis

5.1.1 Introduction

The case study presented in this chapter is an exploratory one. The overall thrust is to use the corpus method to explore lexical patterns in turns, and merely tentatively or even speculatively point out directions for further analyses.

The case study connects immediately to the major theme of the previous case study: the ways that lexis is shaped by and bound up with turn structure. The previous case study demonstrated that one major class of words, inserts, are intrinsically involved, indeed have their functional habitat, in one turn part, the pre-start, whose overarching task is to relate the incipient turn back to the prior turn. In the present case study, we will extend this line of inquiry by asking to what extent not only inserts but all three major word classes – inserts, function words and content words – are correlated with turn structure. In the case of inserts, we saw that their distribution in the turn is heavily skewed to the left-hand periphery of the turn: almost all inserts occur in the turn-first (or, to a much lesser extent, turn-second) position in the turn. The question asked here is this: Are there similar distributional preferences/patterns for positions of function words and content words in the turn?

Why this question? Should we expect to find skewed positioning of function words and/or content words and, if so, why? The rationale for the question has two sources: what we know about the principal functions and frequencies of members of these word classes and what we can, at least initially, *infer* about their positional behavior from previous research.

What we know about the functions and frequencies of inserts, function words, and content words is, briefly, this. To start with, inserts, as noted in Chapter 4, include a 'ragbag' of items that fall into many heterogeous functional categories including interjections, hesitators, backchannels, pragmatic markers, and so forth. Their primary function is interactional, providing a 'hinge' between two turns by relating the incipient turn to the prior turn and often adumbrating the action to be done in the turn-in-progress. Inserts can be either highly infrequent or count among the top most frequent words. They are infrequent as far as ad-hoc vocalizations are concerned, such as 'whoopee', 'whoa', 'yippee-ai-ay', and 'vroom'. They are high-frequent when it comes to established forms such 'well', 'oh', 'yeah', 'mm', etc.: these forms are found among the top most frequent words in any spoken corpus.

Function words include articles, prepositions, helping verbs, conjunctions, and pronouns, to name only a few sub-classes. Most members of these classes occupy the top ranks in frequency lists in any corpus; e.g., 'the' is by far the most frequent word in any general corpus of English, followed by the preposition 'of'. Function words represent closed classes with relatively few members. Their primary function is 'language-internal' in the sense that they serve to make "explicit the relation of lexical words to each other" (Stubbs 2002: 40).

Content words, on the other hand, include four main sub-classes: nouns, adjectives, adverbs, and lexical verbs (but not auxiliary verbs) (cf. Biber et al. 1999). Their primary function is referential in that they "carry most of the lexical content, in the sense of being able to make reference outside language" (Stubbs 2002: 40), with the referential potential greatest in nouns (Biber et al. 1999: 232). They are open classes with thousands of members and counting. Content word types often have very low token numbers or merely occur just once in a corpus; in this case, they are called 'hapax-legomena'.

Consider the frequency distribution of the three word classes in the conversational sub-corpus of the BNC, shown in Figure 5.1.

As can be seen from Figure 5.1, *most* of the top most frequent lemmas in conversation, as in any corpus, are function words; the most frequent function word lemma in the BNC-C is the pronoun 'i'. It is outdone only by the lemma 'be' with almost 250,000 occurrences, coded here as a content word (although, obviously, forms of the lemma 'be' predominantly act as auxiliary verbs, in which case they are better classed as function words). Next, there are a number of insert lemmas of medium token frequencies. Finally, there is a very long series of many thousands of content word types cascading from the upper left corner toward the bottom right end of the figure. As noted, some do have very high token frequencies: for example, the second most frequent content word after the pronoun 'i' is the lemma 'thing'. Those content word lemmas located in the middle of the cascade still have considerable token frequencies numbering in the thousands. But note the large tail of content *hapaxes* (that is, word types occurring just once) in the whole subcorpus of over 4 million tokens! Their number is impressive: 9828. Considering that, as shown in Table 5.1, only 22 function word lemmas and 36 interjection lemmas are hapaxes the 9828 content word lemmas that occur just once are a proportion of 99%!

So function words and also inserts, on the one hand, and content words, on the other, exhibit contrary tendencies in terms of types and token frequency. The difference is greatest for function words and content words: function words comprise few types but the vast majority of these have exceedingly high frequencies; content words, conversely, comprise exceedingly many types, but their token frequencies cover the whole spectrum from very high to very low, including thousands of hapax legomena. We will come back to these frequency differences shortly.

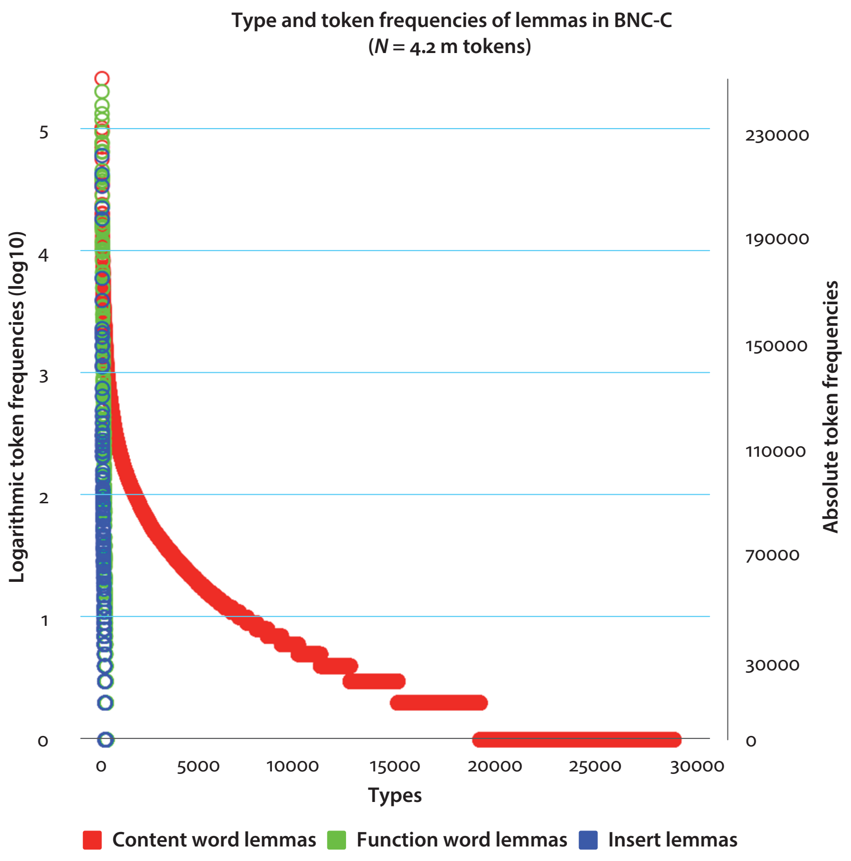


Figure 5.1 Frequency distribution of inserts, function words, and content words in the conversational sub-corpus of the BNC; the figure is based on a frequency table obtained from BNCweb (cf. Hoffmann et al. 2008); content word lemmas comprise nouns (tagged SUBST), verbs (VERB), adjectives (ADJ), and adverbs (AV0); function words lemma classes include pronouns (PRON), articles (ART), prepositions (PREP), and conjunctions (CONJ); inserts comprise those lemmas tagged as interjections (INTERJ).

Table 5.1 Frequencies of hapax legomena across the three word classes

					total
Function word hapaxes	CONJ	ART	PREP	PRON	
	3	4	7	8	22
Content word hapaxes	ADV	VERB	ADJ	SUBST	
	327	1388	2171	5942	9828
Insert hapaxes	INTERJ				
	36				36
					9886

What do we know, or can infer from prior research, about the word classes' positional preferences? Inserts, for sure, have a decidedly strong such preference: as we have seen in Chapter 4, their positional niche is the turn-first slot(s). But function words and/or content words? Here, the answer is less straightforward.

We can begin to address the question by considering the maxim of end-focus, also referred to as 'information packaging' (cf. Kaltenböck 2015). In a nutshell, this maxim ensures that speakers order "the information in a message so as to achieve a linear presentation from low to high information value" (Quirk et al. 1985: 1357); cf. also the notion of 'communicative dynamism' proposed by the Prague School (e.g., Jakobson 1960; Vachek 1964). Quirk et al. refer to it as 'information climax' (Quirk et al. 1985: 1357). In other words, we will expect that the information a speaker conveys in the turn to achieve an action will not be distributed in equal measures across a turn; rather, there will be "an Informational Asymmetry" (Prince 1981: 224) from low to high informational values.

What is low, what high information value? This is a complex question that we will not be able to address fully in this case study. All we can do is provide a rough sketch.

Low in informational value is information that can be considered *given* in various ways. For example, information can be considered given if it has been mentioned in the preceding context (in which case it is discourse-old), or because it can be assumed to be known to members of a group, or information is given if it can be inferred pragmatically via implicature or if it is immediately matched by the speech situation (such as the referent of the pronoun 'I'). The hallmark of given information in speech is that it is prosodically not made prominent.[23] Given information has "a facilitation function in the sense that it provides an anchor for the integration of new information into the memory structure" (Kaltenböck 2015: 122).[24]

By contrast, information is high in value when it is new, in the sense of not being retrievable from the preceding context (discourse-new), not inferable from the pragmatic context, and not evident in the speech situation. New information is made

23. Low-value information is also correlated with the 'theme', that is, usually, "the first element of a clause" (Quirk et al. 1985: 1361), which in many cases in an S-V-O language such as English is the subject.

24. Note that givenness plays a major role in ellipsis. For example, 'situational ellipsis' of the grammatical subject is unproblematic in conversation because the subject is the speaker him/herself. Also, 'structural ellipsis' too has its base in givenness. Wh-questions rarely get a structurally complete answer; rather answers tend to by 'symbiotic' providing merely the asked-for element and presupposing the question's other elements (cf. also Halliday & Hasan 1976):

Sand:	What did you have for dinner?
Alex:	Sausages (BNC: KDW 4432-4433)

prosodically prominent in that it carries the 'nuclear stress' (Leech 1983) and it is what Quirk et al. (1985: 1361) refer to as the 'focus', that is, "the most relevant part of a message" (Quirk et al. 1985: 1362).

The informational asymmetry that holds between given and new information is, however, not a categorical distinction: there is a continuum from given to new with intermediate information values. That is, the notion of communicative dynamism entails that information is climactically distributed in turns: there is a *continuous increase in information* from low to high as the speaker progresses from turn beginning to turn completion.[25,26,27] Let's call this the 'information climax hypothesis'. Note that this hypothesis already incorporates a positional element: if information in the turn begins low and, through intermediate values, ends high, we get this positional ordering:

→ high at turn completion
→ intermediate in medial turn postions
→ low at turn beginning

Now let's get back to word classes: can we make any educated guesses at how the members of the three classes will tend to be positioned in this progression from low to high information? One such positional guess picks up on the frequency distributions of the three classes, which we have seen are strikingly different. Based on these differences and based on simple probabilistic reasoning, it seems permissible to say: if a word is very high-frequent in a corpus it is less likely to convey new information in any given discourse situation because it stands a higher chance of having been used before in the discourse or of being more readily accessible to members of a discourse community. By contrast, if a word is very rare in a corpus, it stands a higher chance of

25. The 'information climax' hypothesis is "cognitively and pragmatically highly plausible as 'given' information anchors the sentence to already existing knowledge and thus facilitates the integration of new information. At the same time, the convention of placing new information late in the clause can make communication more effective, as it enables the addressee to recognize what is seen as the highpoint of the message while attaching it to sufficient old information to ensure correct processing" (Kaltenböck 2015: 120).

26. The 'information climax' hypothesis also provides an elegant explanation for the existence of so-called information packaging constructions such as fronting, existential 'there', cleft-structures, the passive, to name only a few, "all of which allow the speaker to present information in line with the given- before-new principle" (Kaltenböck 2015: 120).

27. For the crosslinguistic implications and complications of the 'information climax hypothesis', see Roberts & Levinson (2017), who compare information structure in turns in languages with verb-initial position and those with verb-final position and discuss these position preferences vis-à-vis the 'crunch zone' at turn turn ends in which production (by the speaker) and comprehension (by the recipient) overlap.

occurring in the speech situation for the first time and/or of being less easily accessible to members of a speech community. For example, if a word is a hapax legomenon, it inevitably must be new to the discourse; if it is attested just once it cannot have been used anywhere else in the corpus. Obviously, a corpus – even a very large one – is just a (minute) sample of a language or a language variety. Therefore, it is quite possible that what is a hapax in the corpus isn't a hapax in the language (variety) but is attested there more than just once. But if, as it should be, the corpus approximates the ideal of being a representative sample of the language (variety), the *proportions* of words in the corpus and in the language (variety) will be the same. That is, if a word is very rare in the corpus it should be very rare in the language (variety) too, and the inverse, if it is (very) common in the corpus, it will be (very) common in the language (variety).

If these – rather speculative – premises are accepted, we can make a clear positional prediction: function words, which are (very) high in frequency, (and also, to a lesser degree, inserts) will be found more commonly at turn beginnings, where, according to the information climax hypothesis, information values are low, and even more commonly in turn-medial positions, where information values are intermediate; content words, by contrast, which are (very) low in frequency, will be used more commonly in turn-medial positions and much more commonly in turn-final positions, where according to the information-climax hypothesis, information is high.

So, in other words, what we have called the information-climax hypothesis could be operationalized as the 'content-word climax' hypothesis: if information is ordered asymmetrically from high to low in turns, then we should find a continuous increase of content words across positions in turns. The aim in this case study is to test the content-word hypothesis.

5.1.2 Data and methods

Unlike other case studies in this book where the analysis is based on small manually annotated data samples, we are going to use in this case study large data sets pulled from BNC-C and use the Part-of-Speech (PoS) annotations readily available in the corpora to identify the three broad word classes in whose distribution across turns we are interested.

Specifically, we will examine the content-word climax hypothesis based on 7 to 10-word turns. The total number of turns will be 69,449, the total number of words will be almost half a million (495,395 words).

How to test the hypothesis methodologically? The first major step is to re-classify all words as 'content word', 'function word', or 'insert' depending on their PoS tag. Thus, the broad descriptor 'content word' is associated with all tags for adjectives, adverbs, lexical verbs, and nouns. The label 'insert' is assigned to tokens tagged ITJ (for inserts cf. the case study in Chapter 4). All remaining words were classified as 'function words'. Second, we compute the frequencies of occurrence of the word class tokens and determine their proportions per position in the turn.

The most handy graph type to visualize changes in proportions is the barplot.

5.1.3 Results

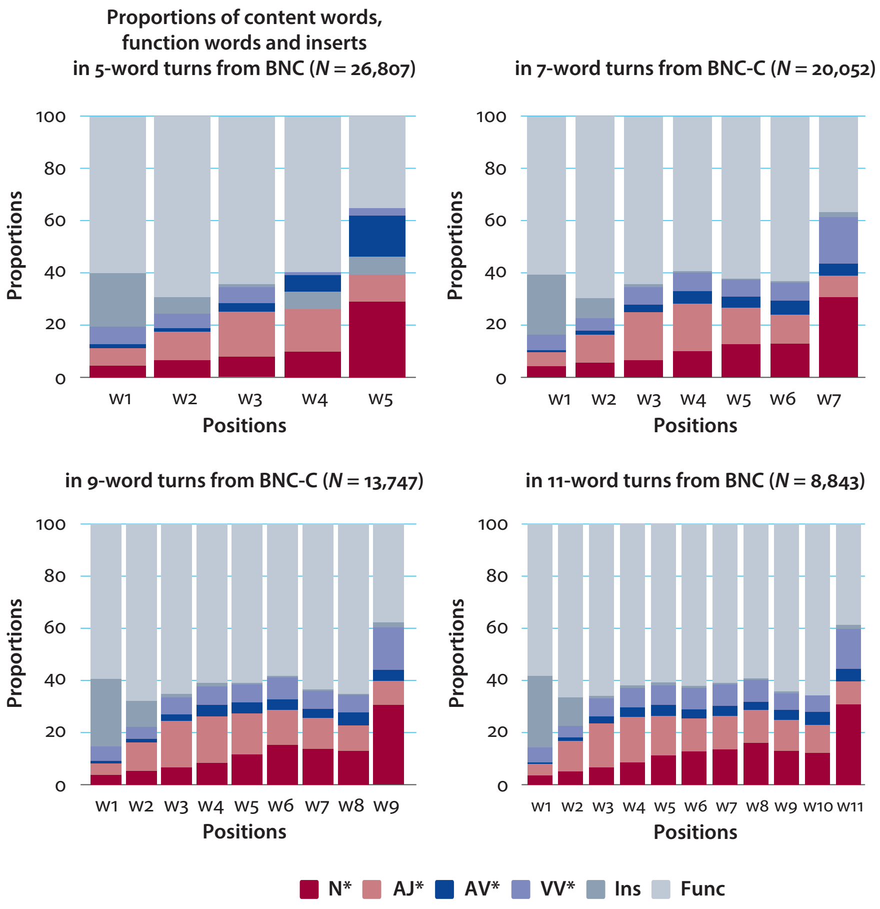


Figure 5.2 Proportions of Content words, function words, and inserts in 5-, 7-, 9-, and 11- word turns in BNC-C

The barplots in Figure 5.2 provide substantial support for the content-word climax hypothesis: the proportions of content words exhibit a clear overall rise across turn positions in all four samples investigated. The greatest single increases in proportion are invariably found for nouns; for example, in the 5-word turn sample, they start out with a modest share of 4.47% but gain a proportion of almost 29.10% upon turn completion (a more than six-fold gain) (Table 5.2).

The increase for content words as a group is continuous in the 5-word turn sample, starting at 24.19% in the turn-first slot and leading up to 62% in the turn-last position (Table 5.2). The increase in the other three samples is bi-modal, meaning that after a first rise across turn-early positions, there is a drop in content word proportions before a second maximum is reached in the turn-final position. In all four samples, however, the proportions reached in the peripheral turn positions (turn start and turn end) are comparable: content words have a share of slightly under 20% in the turn-first position, whereas their share is around 60% in the turn-last slot: almost two out of three turns end on a content word.

Table 5.2 Proportions of (sub-classes of) content words, inserts, and function words in 5- word turns in BNC-C

Broad class	Sub-class	w1	w2	w3	w4	w5
Content words	Noun	4.47	6.69	8.03	9.82	29.10
	Verb	6.68	11.04	17.25	16.21	10.24
	Adjective	1.35	2.02	3.20	6.68	6.74
	Adverb	11.69	5.80	6.28	6.62	15.92
	subtotal	*24.19*	*25.55*	*34.76*	*39.33*	*62.00*
Inserts	-	15.70	2.31	0.89	0.90	2.88
Function words	-	60.11	72.14	64.36	59.76	35.11

5.1.4 Discussion and concluding remarks

The analysis provides support for the content word climax hypothesis: overall, the proportions of content words increase over positions in the turn; specifically, they peak in the very last position, that is, at turn completion. This finding may be taken as initial evidence that the broader information climax hypothesis may be true.

Note the qualification as 'initial' evidence. The evidence provided by the above analysis and visualized in the barplot in Figure 5.2 is initial for at least two reasons. First, we did not factor in turn structure, as discussed in Chapter 4. That is, we did not distinguish turns that end on the transition-relevance point (TRP) from those that end on a post-completer (which by definition occurs post-TRP). Post-completers are quite unlikely candidates for focal information; rather, given their interactional function of bridging over from the current turn to the next, they are likely not integrated into the information structure of the TCU. Second, the evidence presented is initial because whether or not a word carries new focal information cannot be determined on lexical grounds only. This is for at least two reasons. The first is that the information highpoint need not necessarily be placed late in the turn. This is most obvious in the case of *wh*-questions, which typically front-load new focal information following the 'front-loading bias' (Levinson 2013: 112), as illustrated in (5.1):

(5.1) Clare: **why** have you got to be home so early then?
Mel: because I've got work to do (BNC: KBN 730-731)

Second, while in the unmarked case it is through content words that new focal information is expressed, function words and even inserts, too, can carry this information, and, in fact, they do "when special emphasis is required" (Quirk et al. 1985: 1365). There are numerous contexts where speakers seek such special emphasis. For illustration consider extracts (5.2) through (5.4). In (5.2) Ann places heavy emphasis on "you", a function word occurring as the first word in her turn, to express her horror at the idea that James should be driving a bus; in (5.3) Joanne stresses how much she felt annoyed by emphasizing the auxiliary verb "has"; and in (5.4), Terence highlights the insert "yes" to confirm that he did put the chicken in the freezer thus ending a lengthy stretch of hesitation.

(5.2)
James: if they (> can actually <) let me kno:w (0.3)
whether that bus is available or not.
(1.3)
Ann: **YO::** 're not gonna drive that bus are you?
(0.7)
James: I didn't say I wa:s.
(BNC: KB8 6858-6860; corrected transcription)

(5.3)
Joanne: he **ha:s** got me annoyed now, he really ha:s done.
(BNC: KCE 3375; corrected transcription)

(5.4)
Margaret: presu:mably you took those chicken:: (1.0)
things downstairs for me
(0.3) yesterday,
(2.1)
Terence: presumably I did.
(1.8)
Margaret: you know the [( )]
Terence: [yes i think i] di:d yeah.
(0.4)
the what?
(0.6)
Margaret: that [( )]
Terence: [oh **YES** i put them in the] freezer yes yes.
(BNC: KB2 9574-9579; corrected transcription)

Many more such examples of marked focus could be collected. Therefore, caution is due with regard to the information climax hypothesis, according to which the informational high point tends to occur at or toward the end of turns. However, what is marked is, by definition, not the default or 'normal' case. This leaves the possibility that unmarked focus where the informational highpoint comes late in the turn is the 'norm'.

In sum, we have substantial evidence for the content word climax hypothesis but at best preliminary evidence to suppport the information climax hypothesis. To examine this latter hypothesis more rigorously, a qualitative analysis is required of how speakers package information in turns within the larger sequential contexts. Under most circumstances, qualitative analyses are limited in terms of number of analyzed units. In this case study, we have taken another route, by investigating large numbers of turns drawn from corpora. This corpus method is admittedly less exact, looking at turns in isolation from their sequential context and building on the premise that different word classes correlate with different probabilities of carrying 'low' or 'high' information. But the reduced exactness of the method deployed here is at least partially compensated for by the bird's eye perspective it provides. This perspective can discover the outlines of an underlying structure (like the forgotten remains of archeological sites that are discernable only from above), a structure, in this case, for how speakers distribute low and high information in turns. While this structure may get undercut, overlaid, or overridden by moment-to-moment concerns arising in situated talk-in-interaction, it may yet be present and active as a *fundamental orientation* for how to design and distribute information in turns.

5.2 The barplot in the case study

What were the major steps in programing the barplots in the case study? In this section we will examine only the code underlying the barplot for 5-word turns; the other panels in Figure 5.2 were programmed accordingly.

As always, the first step is to read-in the data (cf. the file "Chapter5_Casestudy.txt"), store it as `t5`, and inspect it using `View()`:

```
t5 <- read.table("[Your path]/Chapter5_Casestudy.txt", header=T,
quote="", sep="\t", fill=F)
```

A quick look at the dataset reveals it has 13 columns:

```
> head(t5)
  file   who                         turn   w1  w1_c5      w2
1 KB0 PS002 You enjoyed yourself in America  You    PNP enjoyed
2 KB0 PS006          saw Mary and Andrew and  saw    VVD    Mary
3 KB0 PS006          It is horrible is n't ?   It    PNP      is
4 KB0 PS002              And you 're busy at  And    CJC     you
5 KB0 PS006         I think it was yesterday    I    PNP   think
6 KB0 PS006     They have there , yes , and They    PNP    have
```

```
  w2_c5        w3 w3_c5      w4  w4_c5        w5  w5_c5
1   VVD  yourself   PNX      in    PRP   America    NP0
2   NP0       and   CJC  Andrew    NP0       and    CJC
3   VBZ  horrible   AJ0      is    VBZ       n't    XX0
4   PNP       're   VBB    busy    AJ0        at    PRP
5   VVB        it   PNP     was    VBD  yesterday   AV0
6   VHB     there   AV0     yes    ITJ       and    CJC
```

We will mostly be concerned with the columns for Part-of-Speech (PoS) tags; their names all end in c5 (the name of the automatic tagger used in the BNC).

Before we start grouping the PoS tags in larger groups we need to implement one important correction: as noted before, the BNC's tagger consistently mistagged as adverbs (AV0) pragmatic uses of 'well' (most of them functioning as pre-starts; cf. Chapter 4); the same error occurred with regard to pragmatic 'so'. We can easily correct the tag in these two turn positions, by overriding the tag with the correct label, ITJ (for interjection):[28]

```
t5$w1_c5[t5$w1=="Well"|t5$w1=="well"|t5$w1=="So"|t5$w1=="so"] <- "ITJ"
t5$w2_c5[t5$w1=="Well"|t5$w2=="well"|t5$w2=="So"|t5$w2=="so"] <- "ITJ"
```

As we are interested only in the PoS tag columns and later will have to plot the data from a matrix (a data format that requires same-type data), we separate these columns into a new dataset, called c5:

```
c5 <- t5[, c(5,7,9,11,13)]
```

Let's check what's in this dataframe:

```
> head(c5)
  w1_c5  w2_c5  w3_c5  w4_c5  w5_c5
1   PNP    VVD    PNX    PRP    NP0
2   VVD    NP0    CJC    NP0    CJC
3   PNP    VBZ    AJ0    VBZ    XX0
4   CJC    PNP    VBB    AJ0    PRP
5   PNP    VVB    PNP    VBD    AV0
6   PNP    VHB    AV0    ITJ    CJC
```

Now we can group the PoS tags into broad classes. We do it using lapply(), a function that allows you to execute an action across all columns of a dataframe via

28. An alternative, and more economical, solution is this:

```
> well_so <- c("well", "so","Well", "So")
> ts$c5_w1[ts$w1 %in% well_so] <- "ITJ"
> ts$c5_w2[ts$w2 %in% well_so] <- "ITJ"
```

a self-defined function; in defining that function we make use of nested `ifelse()` statements. Specifically, we group all ITJ tags into a group called `Insert`, all tags starting with capital N (in regex `^N`) as `Noun`, all lexical verbs, whose tags start with VV, as `Verb`, all adjectives as `Adjective`, all adverbs as `Adverb`, and, finally, all words that do not satisfy these criteria will be labeled `Function`:

```
> c5_broad <- as.data.frame(lapply(c5,
+                           function(x)
+                           ifelse(grepl("ITJ", x), "Insert",
+                                  ifelse(grepl("^N", x), "Noun",
+                                  ifelse(grepl("^VV", x), "Verb",
+                                  ifelse(grepl("^AJ",x),"Adjective",
+                                  ifelse(grepl("^AV", x), "Adverb",
                                   "Function")))))))
```

Let's inspect the first six lines:

```
> head(c5_broad)
      w1_c5    w2_c5     w3_c5     w4_c5    w5_c5
1  Function     Verb  Function  Function     Noun
2      Verb     Noun  Function      Noun Function
3  Function Function Adjective  Function Function
4  Function Function  Function Adjective Function
5  Function     Verb  Function  Function   Adverb
6  Function Function    Adverb    Insert Function
```

The first word in column `w1_c5` has been labeled `Function`. If we look back to the first six lines of `c5` shown above, we see that the word's original tag was `PNP` (for personal pronoun). Pronouns count as function words, so the broad classification as a function word is correct. The second word in column `w1_c5` records `Verb`. Let's check again with `c5`: the tag there was `VVD`, so the broad class it belongs to is indeed `Verb`.

The next major step is to compute the proportions of the six broad word classes we have defined per turn position. To achieve this we will use `sapply()`, this time using the `table()` function nested inside the `prop.table()` function:

```
> prop5 <- as.data.frame(sapply(c5_broad,
+                               function(x)
+                               prop.table(table(x))*100))
```

Let's check what this has produced:

```
> prop5
              w1_c5     w2_c5      w3_c5      w4_c5     w5_c5
Adjective  1.354124  2.018130  3.1969262  6.6848211  6.744507
Adverb    11.690976  5.800724  6.2819413  6.6176745 15.921215
Function  60.107435 72.137874 64.3563248 59.7642407 35.110232
```

```
Insert    15.697392  2.309098  0.8878278  0.9027493  2.879845
Noun       4.472712  6.692282  8.0277539  9.8220614 29.104338
Verb       6.677360 11.041892 17.2492259 16.2084530 10.239863
```

We see that R has produced the proportions but it has also ordered the rows alphabetically, thus we find `Function` and `Insert` surrounded by content word classes. To arrange the rows in a more sensible order we can switch them:

```
> prop5 <- prop5[c(5,6,1:2,4,3),]
> prop5
             w1_c5     w2_c5      w3_c5      w4_c5     w5_c5
Noun       4.472712  6.692282  8.0277539  9.8220614 29.104338
Verb       6.677360 11.041892 17.2492259 16.2084530 10.239863
Adjective  1.354124  2.018130  3.1969262  6.6848211  6.744507
Adverb    11.690976  5.800724  6.2819413  6.6176745 15.921215
Insert    15.697392  2.309098  0.8878278  0.9027493  2.879845
Function  60.107435 72.137874 64.3563248 59.7642407 35.110232
```

This order makes more sense in that all content word classes are together, followed by inserts and function words. Just one more step is required before we plot the proportions: we need to re-format `prop5` as a matrix. This is because the type of barplot we want to produce, the stacked barplot, requires this data format. The re-format is done using `as.matrix()`:

```
> prop5 <- as.matrix(prop5)
```

One more preparatory step: if we wish to produce a plot with several panels we will have to determine the number of rows and columns in the `mfrow` argument to `par()`. For example, if we want to get a 2 × 2 figure as in the case study, we set `mfrow = c(2,2)`. In the same call to `par()` we can also determine the size of the margins around the plot using the `mar` argument:

```
> par(mfrow=c(2,2), mar=c(4,4,4,1))
```

Now we can start plotting. The function for barplots is `barplot()`; its first argument is the data, in our case the matrix `prop5`; next we have the usual parameters including `main` for the title (which we will suppress here for space reasons), the labels `xlab` and `ylab` for the x- and the y-axis. A major question to be addressed is how to visualize the word classes in such a way that those classes that belong together – namely `Noun`, `Verb`, `Adjective`, and `Adverb`, as these are the content word classes – can be distinguished from the other two major classes but also seen as related to one another. The solution to this issue proposed here (one out of many possible ones) is to use the same color for the four content word classes but to distinguish them by using different degrees of `density`. This argument draws lines at an angle defined in the argument `angle`, which we will set here to `45`. Finally, the five bars should be labeled. Labeling bars in barplots works with the argument `names,` as in most other graphics, but also with the argument `names.arg`. In our case, we could specify the labels individually as "w1", "w2" etc. but will prefer a more synthetic way, by using the `paste()` function. Finally, we set `las = 2` to rotate the labels by 90 degrees.

```
> barplot(prop5,
+          main = "",
+          ylab = "Proportions", xlab = "Positions",
+          col=c("red", "red", "red", "red", "blue", "green"),
+          density = c(80, 60, 40, 20, 50, 50),
+          angle = 45,
+          names.arg = paste("w",1:5, sep = ""), las = 2)
```

As we have not defined `main` so far, we will have to define the title for the barplot using the `title()` function, thus:

```
> title("Proportions of Content words,\nFunction words & Inserts",
+        line = 2.4,
+        cex.main = 0.9)
```

The argument `line` determines at which line the title is to be set in the plot margin, `cex.main` manipulates the title's font size, and `\n` instructs R to set what follows it on a new line. Next, we want to include a subtitle, using `title()` again:

```
> title("in 5-word turns from BNC (N = 26,807)",
+        line = 1.7,
+        cex.main = 0.7)
```

Finally, a legend is needed to provide a key for the barplot. As there is not enough space for the legend in the plotting region, we have to allow R to plot the legend outside the plotting region. This permission is granted by setting the `par()` argument `xpd` to `T` (or `TRUE`), thus:

```
> par(xpd = TRUE )
```

The function `legend()` takes a large number of arguments. Obviously, what must be included is an indication where to place the legend. This can either be done by ready-made expressions such as `"top", "topleft", "bottomright"` or by specifying the x- and y-coordinates of the legend's starting point; in our case `-1, 115`. Next, we set the argument `horiz` to `T` to display the labels horizontally. Then using the argument `labels` we define the labels for the six word classes we distinguish. Legends typically provide small boxes with different colors to distinguish the labels and relate them to the bar segments bearing the same colors; this is achieved by the arguments `col` and `fill`. We also repeat the `density` and `angle` details we already used in the barplot. We determine the font size by `cex` and the distance between the labels and the label boxes by `x.intersp`. Finally, we avoid drawing a frame around the whole legend (as this may interfere with the bars) by `bty = "n"`.

```
> legend(-1, 115,
+        horiz = T,
+        labels = c("N*", "AJ*", "AV*", "VV*", "Ins", "Func"),
+        col = c("red", "red", "red", "red", "blue", "green"),
+        fill = c("red", "red", "red", "red", "blue", "green"),
+        density = c(80,60,40,20, 50, 50),
+        angle = 45,
+        cex = 0.65,
+        x.intersp = 0.4
+        bty = "n")
```

In the case study, we continue plotting the other turn sizes. For this section however, we will consider the plot done.

5.3 Task: Proportions of frequency groups in 7-word turns

The above tested content word hypothesis was partly based on the observation that content words generally have much lower frequencies than function words and also inserts. As the content word hypothesis was confirmed by the analysis – content words increase across the word positions in turns –, it should follow that the proportions of low frequency words also increase from the beginning of the turn to its end. In this Task section we are going to test this assumption. The analysis will be based on a sizable dataset, namely all 20,000 or so 7-word turns in BNC-C.

The major steps in the coding process are the following:

1. Download the file "Chapter5_Task.txt" and read it into R, storing it as `t7`. You will see the dataset contains all the words' PoS tags, the words themselves, and, importantly, their absolute frequencies in the whole BNC-C (which comprises 4.2 m words).
2. Then subset `t7` to separate the data of interest – the frequencies – in a new dataframe called `t7f`.
3. The next step is crucial: to be able to compute proportions, you need to group the frequencies into larger frequency groups. Use the `lapply()` function as well as nested `ifelse()` statements:

```
> t7f_groups <- as.data.frame(lapply(t7f,
+              function(x)   ifelse(x <= 1, "1",
+                            ifelse(x >= 2 & x <= 3, "2-3",
+                            ifelse(x >= 4 & x <= 6, "4-6",
+                            ifelse(x >= 7 & x <= 11, "7-11",
+                                   ifelse(x >= 12 & x <= 20, "12-20",
+                                   ifelse(x >= 21 & x <= 37, "21-37",
+                                   ifelse(x >= 38 & x <= 70, "38-70",
+                                  ifelse(x >= 71 & x <= 135, "71-135",
+                                  ifelse(x >= 136 & x <= 264, "136-264",
+                                          ">264")))))))))))
```

4. From this point on you can pretty much proceed as was demonstrated for the barplot in the case study: (i) compute the proportions of the frequency groups in all seven word slots (with `sapply()`, storing them in a dataframe called, for example, `t7f_groups_props`, (ii) switching the rows so that they reflect the order in which you want the proportions to be plotted, and (iii) re-defining `t7f_groups_props` as a matrix.

The proportions you should then find should be the same as the ones depicted in this barplot (the graphical details will depend on your chosen parameters):

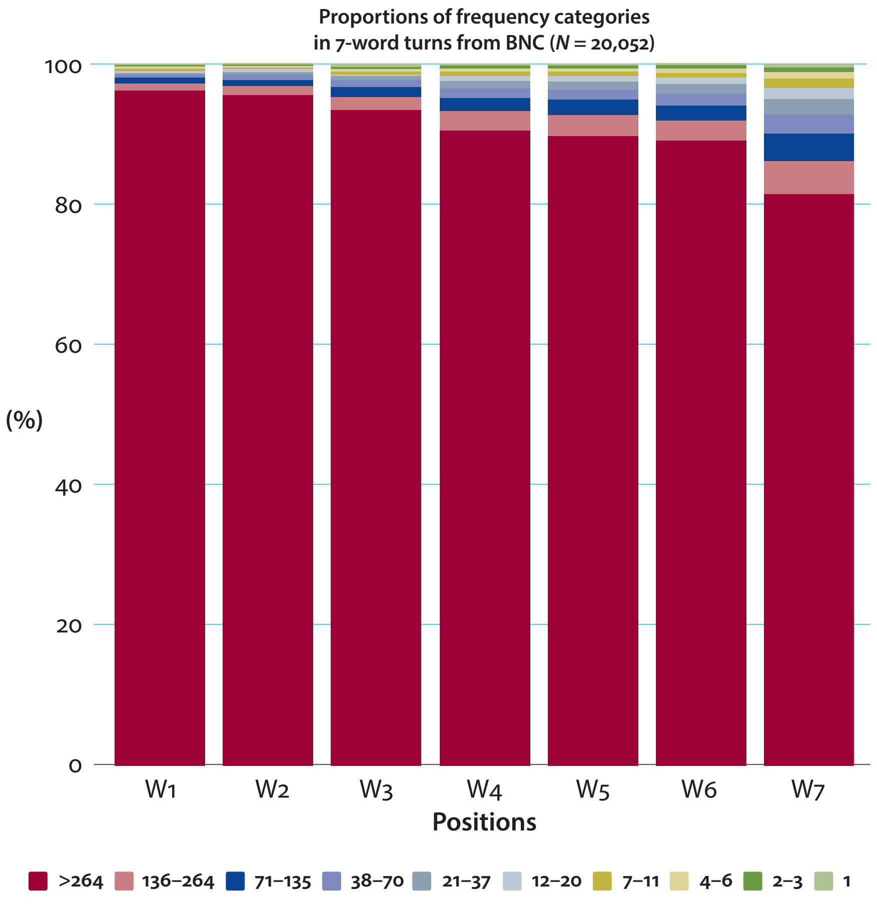


Figure 5.3 Proportions of frequency categories in 7-word turns in the BNC

Clearly, the words in the group with the greatest frequencies, namely frequencies greater than 264 occurrences in the whole BNC, get the lion's share in *all* seven word slots. But still, the proportions of words in low-frequency groups keep continuously climbing up from low levels at turn beginning to higher levels at turn end. The barplot in Figure 5.3 thus nicely confirms the assumption we set out to test: infrequent words do take up larger and larger proportions as we move from turn beginning to turn completion.

Chapter 6

Dotcharts

6.1 *Case study*: Turns and information structure

6.1.1 Introduction

Turns perform actions. Precisely *how* a turn embodies action varies from turn to turn. But what must be involved invariably is information of some sort – in most cases, a *gestalt* composed of information from different modalities, be they verbal, vocal, or gestural (Holler & Levinson 2019).

In this case study we will approach information in turns by investigating a key prosodic resource often thought to be indicative of information structure, what is called the nuclear tone, also known as nuclear stress or nucleus. Nuclear stress is defined as the syllable or word in a tone unit "which carries maximal prominence, usually due to a major pitch change" (Crystal 2003: 321). An example is presented in Figure 6.1, showing how Praat, a phonetic analysis software, traces the pitch contour of the turn "But why didn't she go an a Saturday then?":

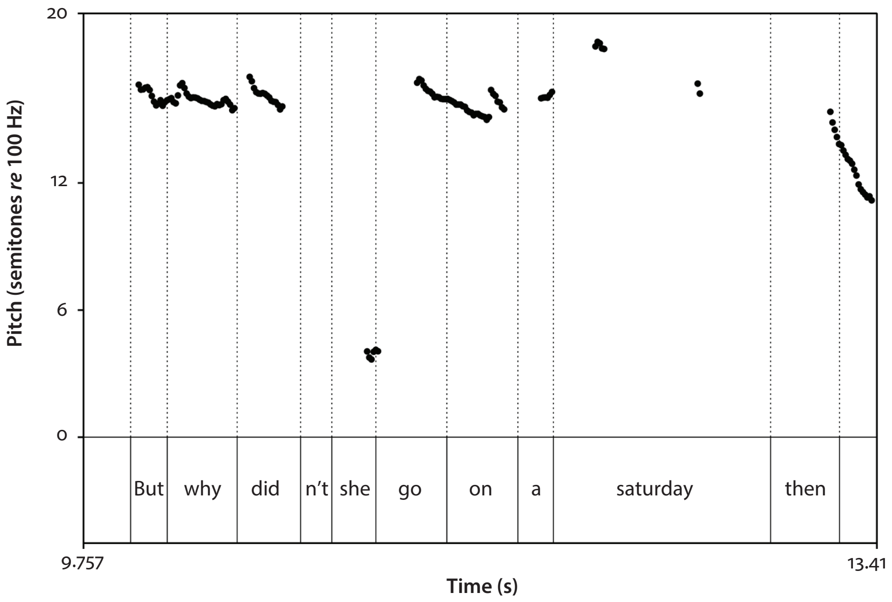


Figure 6.1 Pitch contour of turn "But why didn't she go an a Saturday then?"

There is a stepup in pitch of about 2.4 semitones on the word “Saturday”. The stepup makes the word “Saturday” literally ‘stand out’ from the turn and marks it as informationally key, or, as Quirk et al. (1985: 1357) put it, as the turn’s ‘information climax’. What can be observed in this example is a general pattern in English, where the nuclear stress at large identifies “the point of information in the sentence that is deemed most valuable or relevant from the speaker’s point of view” (Rochemont & Cullicover 1990: 18). Inasmuch as most ‘valuable or relevant’ information is at the same time ‘new’ information – in the sense of being new to the hearer or not retrievable from the preceding context, or not inferable from the pragmatic context, or not evident in the speech situation – researchers have posited a “fundamental association between high pitch and new information in English” (Wennerstrom 2001: 34).

Speakers decide online, in situ, as to which word to place the nuclear stress on. And decisions must be taken as there is, in principle, ample room for variation. For example, in the turn “But why didn’t she go an a Saturday then?” there is little – by way of grammar – to prevent a speaker from making prominent, not “Saturday”, but, for instance, the question word “why”, as in “But WHY didn’t she go on a Saturday then?” or the pronoun “she”, as in “But why didn’t SHE go an a Saturday then?” and so on. In deciding which word to stress speakers look to what is going on right now in the interaction on hand. So nuclear stress is an online resource.

The nucleus is, then, related to information structure and where it is placed is context-dependent. In this case study, the primary aim is to investigate whether nucleus placement, as an indication of information structure, is correlated to another measure of information structure, surprisal.

Surprisal is a notion developed in the field of information theory and based on predictability.

Predictability arises from the fact that language production is inevitably linear: we cannot but produce a sequence of words in which one word is followed by another word, which is followed by yet another, and so forth. Predictability further arises from the ‘idiom principle’. This principle holds that when we choose one word, this choice commonly co-selects one or more other words, called ‘collocates’, that is, the company that initially chosen word keeps (Sinclair 2000: 197). If a speaker utters one word and the next word happens to be such a common collocate then this next word will be little surprising: based on our knowledge of the language and of how words collocate we could guess the word. Obviously, not every word combination is easily predictable. Often an initial choice will be followed by a word (or words) that are less easily predicted – that are hence more suprising.

To measure the surprisal value of a word, one looks at the word that preceeds it. Let’s label the first word A and the second word B. Words A+B form what is called a bigram. Now, surprisal of B given A is calculated in three steps.

First, the frequency of bigram A+B in a large, hopefully representative, corpus is determined. Second, the frequency of bigram A+B is divided by the frequency of B alone in the corpus. This operation gets you what is called the Conditional Probability of B given A. The formula is this:

Conditional Probability B = Corpus frequency of A+B / corpus frequency of A

The more commonly and the more exclusively B follows A, then the more probable, and more predictable, B will be. For example, it turns out that in the BNC the clitic ''m' almost always follows the pronoun 'I'; the frequency of the bigram and the frequency of ''m' alone are hence quite similar and the conditional probability of ''m' given 'I' is close to 1 (namely 0.999786).

Conditional Probability is a measure of the predictability of a word given a preceding word: the closer the value to 1, the more easily predictable B is given A. *Surprisal* is the flip side of predictability: only something that is hard to predict will be considered surprising. So the third step in determining surprisal is to take the negative logarithm of the Conditional Probability value of B given A to the base of 2:

Surprisal B = –log2(Conditional Probability B)

To return to the combination 'I'm', as ''m' is very highly predictable given 'I', it is at the same time very little surprising. This is reflected in the fact that the surprisal value of ''m' given 'I' is very close to 0, in fact 0.0003088291.

We are interested here in the surprisal of words within the sequence of words in turns-at-talk: that is, we are interested in how surprisal develops from the turn's beginning to its end point. To get there, we need to establish the surprisal of B given A (or w2 given w1), C given B (w3 given w2), D given C (w4 given w3), and so forth. Consider this illustrative example in (6.1):

(6.1) God it all happens in your bathroom does n't it?

The turn contains the following bigrams:

w1w2	w2w3	w3w4	w4w5	w5w6	w6w7	w7w8	w8w9	w9w10
God it	it all	all happens	happens in	in your	your bathroom	bathroom does	does n't	n't it

Which combinations will intuitively contain more surprisal? The most easily 'guessable' combinations seem to be, perhaps, 'your bathroom', where 'bathroom' may be surprising given that 'your' can collocate with so many other words. On the other hand, 'n't' in 'does n't' will be easy to predict as it frequently forms, along with 'it', a question tag; hence, 'n't' will be little informative or surprising. As shown in Figure 6.2, these intuitions are largely borne out by corpus data:

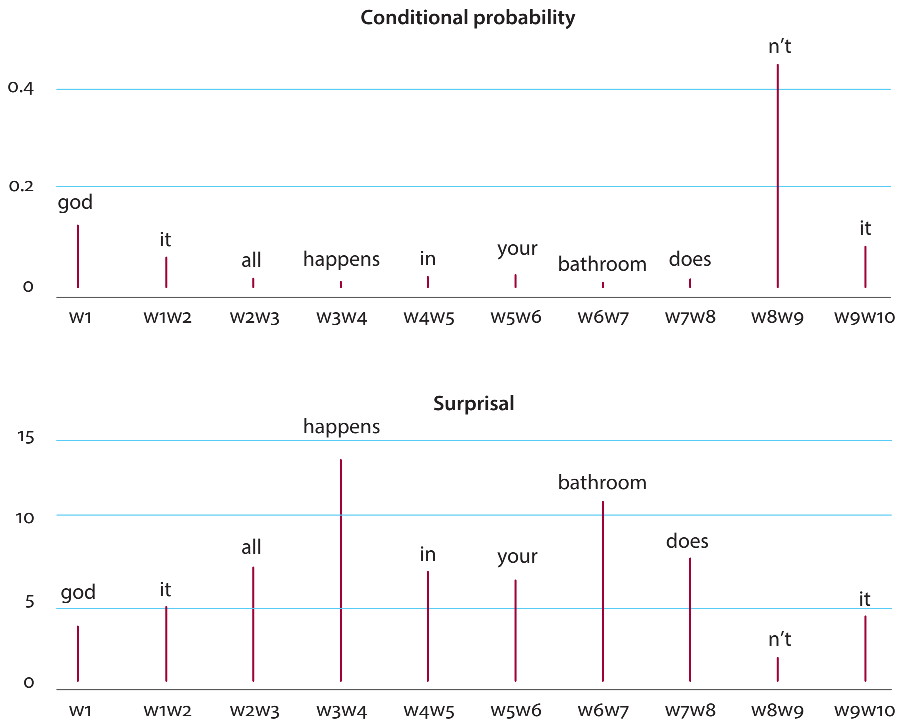


Figure 6.2 Conditional probability and surprisal in the turn "God, it all happens in your bathroom doesn't it?"

While 'happens' and 'bathroom' have the lowest predictability in this turn but the highest surprisal value, 'n't' has the highest predictability but the lowest surprisal value.

Surprisal, conceptualized in this way, ultimately relies on frequency and probability distributions of words and their combinations, that is, on how rare or common a word is in the language variety on hand and how rare or common a combination of certain words is. Do speakers *know* how words are distributed in terms of their relative frequencies and probabilities in a language variety? Our little intuition experiment above seemed to suggest that we can indeed gauge, to an extent, what collocations are more or less probable. In information-theoretic research it is a widely accepted observation that speakers do have access to frequency and probability distributions (e.g., Jaeger 2010). These distributions are believed to be stored in the mental lexicon and made use of in language production, thus providing an offline resource available to speakers anytime, independently of the particulars of the interaction on hand.

So our prediction will be that nuclear placement tends to fall on surprising words in the turn. In other words, we hypothesize that the position of the nucleus will be the position of high-surprisal words. To test this hypthesis the data and methods outlined in the following section were used.

6.1.2 Data and methods

The data for the case study come, again, from the conversational subcorpus of the BNC. To calculate surprisal, all 4.2 m bigrams were extracted from BNC-C using XQuery, a querying language designed for XML documents (such as the BNC) (cf. Rühlemann et al. 2015); then, based on the procedure outlined above, surprisal was calculated for each word except for words in turn-first position. The surprisal of turn-initial words was based on their frequency in turn-first position in the whole BNC-C divided by their total frequency (in any turn position) in the BNC-C. Thus, each word in the turns was assigned a surprisal value.

Facilitated by the Audio BNC (Coleman et al. 2012) a randomly sampled set of 1,000 ten-word turns were analyzed in Praat, a phonetic analysis tool (Boersma & Weenick 2012), both for nuclear stress and word duration. The identification of nuclear stress was conservative: turns were discarded when there was no clear single nucleus but a flat intonation contour or when there were multiple nuclei. The number of turns included in the final sample was 542.

The graphic used to visualize the relation of nucleus and surprisal is the dotchart. While this plot type is usually used for plots of a single variable, we are going to tweak it to plot the two variables by using it together with the `text()` function.

6.1.3 Results

We have two variables: position of nucleus and surprisal. While there is only a single word that carries the nuclear stress there are ten surprisal values in each turn, one for each word. What we want to find out is how the position of the nucleus relates to how surprisal is distributed across the turn. How to visualize the relation of these variables?

As to nucleus positions, the visualization is straightforward: all we have to do is plot the nucleus in the turn slot in which it appears. The nuclei will fall, in principle, into any slot. So we would see them scattered across the whole dotchart, which would make it hard to see any patterns in their positioning both vis-à-vis the turn and the surprisal distribution. To impose some order on how the nuclei will be plotted into the graph we will simply sort the whole dataset by nucleus position. That is, we put all those turns together where the nucleus falls on the first word, followed by all turns where the nucleus is on the second word, and so forth.

As regards surprisal, we are specifically interested in differences in surprisal *within the turn*, that is, how surprisal values relate to one another in one and the same turn. One way to get a handle on this turn-internal relationship is to *rank* the surprisal values, assigning the ordinal value 1 for the greatest surprisal value per turn, 2 for the second-greatest, 3 for the next-greatest and so forth. To visualize the rankings and their gradual development from low to high, we will use a heatmap-like palette of colors, ranging from light yellow (for the lowest-ranked value) to hot red (for the highest-ranked value).

The dotchart in Figure 6.3 allows for at least two observations. First, we can see a tendency for nucleus frequencies to increase across word positions in the turn: in very first position, there is a small number of nuclei (18, to be precise; cf. Table 6.1) but that number gets larger as we move toward the end of the turn, where the number of nuclei is greatest (namely 81). So there seems to be a tendency for speakers to place the nuclear stress on words later rather than early in the turn.

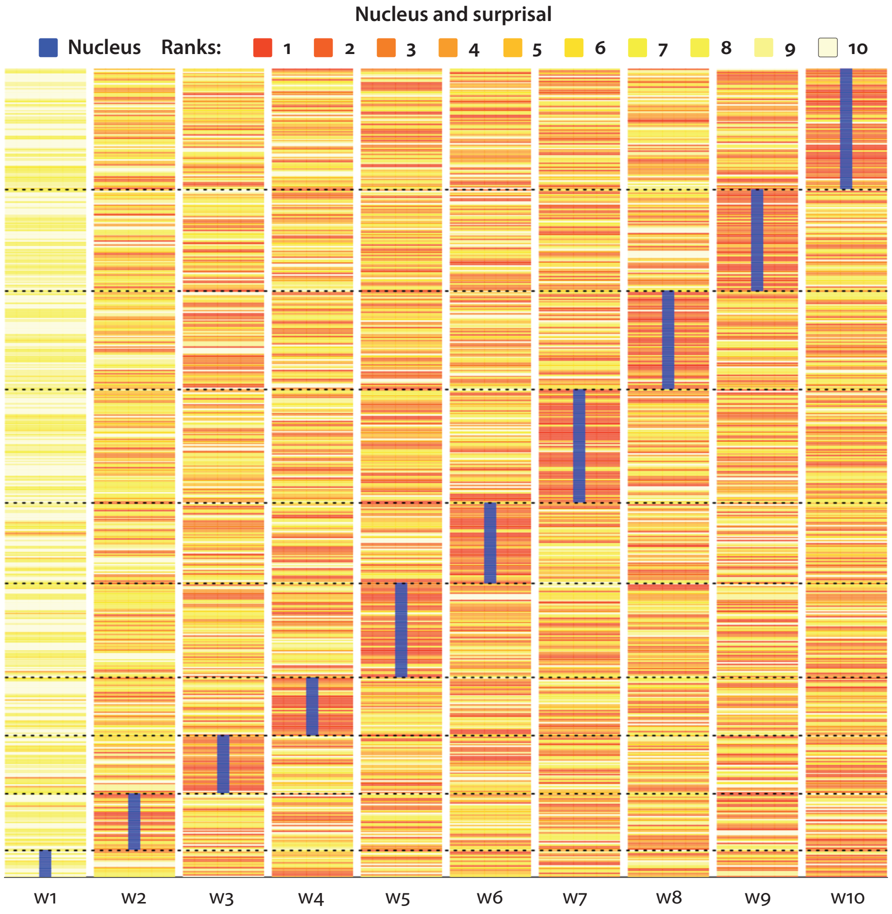


Figure 6.3 Nucleus position and surprisal

Table 6.1 Nuclei per position in turn

	w1	w2	w3	w4	w5	w6	w7	w8	w9	w10
Freq	18	38	39	39	63	54	76	66	68	81
%	3.32	7.01	7.20	7.20	11.62	9.96	14.02	12.18	12.55	14.94

Second, the colors from the red spectrum cluster a great deal around the nuclei in all but the first position. This is remarkable as the red spectrum is for high surprisal values: a great many words whose surprisal value is ranked first, second, or third are at the same time the words carrying the nucleus. In other words, the dotchart suggests that the nucleus tends to fall on high surprisal words. For example, as shown in Table 6.2, in more than a third of all turns (34.50%), the word carrying the nucleus is at the same time the word, called MaxInf, that is maximally surprising in the turn.

Table 6.2 Nucleus position and position of the maximally surprising word in the turn (MaxInf)

	Freq	%
No. of turns where position of MaxInf equals position of nucleus	187 (out of 542)	34.50

6.1.4 Discussion and conclusion

We have observed that speakers tend to make those words maximally prominent that are surprising in the turn. Two canonical examples are found in extract (6.2):

```
(6.2)
1   Bev:     when it gets the TOggles on it should be okay
2   Wendy:   the wha'?
3   Bev:     when it gets the TOggles on it sh[ould be] okay
4   Wendy:                                    [ yeah   ]
                  (BNC: KE6 1677-1680; corrected transcription)
```

In (6.2), the plural noun "TOggles" (a kind of button) is used in two turns by Bev, the second being a repeat of the first. The repeat is initiated by Wendy's response "the wha'?", indicating a comprehension problem and requesting a repair. Wendy's use of the article "the" locates the 'repairable' (the item causing the comprehension problem) as the word following the article. This is surprising as even upon its first utterance the word "TOggles" is made clearly prominent by the nuclear tone on it. The fact that "TOggles" creates a comprehension problem for Wendy is hence probably due to the word's rarity: the noun 'toggles' occurs merely three times in the whole BNC-C. Since the definite article preceding it occurs 115,359 times, its surprisal value is very high, namely 15.816, which is the highest surprisal in the turn.

The correlation we observed between nucleus placement and surprisal aligns well with research in information theory (e.g., Aylett & Turk 2006; Jaeger 2010). There, a common observation is that speakers invest those words they deem predictable and thus informationally redundant with significantly less vocal effort, producing them "with shorter duration and with less phonological and phonetic detail" (Jaeger 2010: 24). It is the flip side of this observation that is key here: if predictable words are produced with less vocal effort, it follows that surprising words will be produced with more such effort, that is, more 'phonological and phonetic detail'. Now, marking a word by raised pitch clearly requires and manifests this effort. So the dotchart we have inspected is further evidence to support the asymmetry observed in information-theoretic research of the reduced vocal effort with which predictable words are produced and the increased vocal effort with which surprising words are produced.

The surprising correlation of nucleus and surprisal shown in the dotchart also reveals a surprising connection of two resources that could hardly seem less connected. As noted above, *where* to place the nucleus depends on what action the speaker wishes to accomplish in the turn. Nucleus placement is an online decision. But the evidence we have discovered suggests that this decision is also informed by offline resources – namely the frequency and probability distributions, which are the 'data base' on which predictability and surprisal are computed and which are resources stored in the mental lexicon. In other words, the online decision related to action formation may have deep mental corrrelates, correlates speakers are little aware of but that nonetheless leave an imprint on talk in social interaction.

We have also seen that the correlation between nucleus and surprisal is not perfect. If it were perfect the rate of nuclei that fall on the maximally surprising word would not have been 34.50% but 100%. There are, then, lots of deviant cases. Consider, for example, extract (6.3), in which Bob and Alan are talking about African news sheets:

(6.3)

1	Bob:		Yes south africa:: has erm (0.63)
2			i i- we have a (1.29) regular (0.95) erm (1.80)
3			a four sheet two s- two sides an' a four sheet (1.16)
4			erm (0.35) it was weekly but
6			no:w it's erm (1.22) a fortnightly
7	Alan:		mm.
8	Bob:		news (.) letter
9			(0.47)
10	Bob:	→	**well (0.36) that's just a news sheet (1.06) from (1.00) OF zambia**
11	Alan:		mm,

(BNC: KB0 613-614; corrected transcription)

In Bob's turn in line 10, "Zambia" is the most surprising item in the turn (its surprisal value is 15.368097) but the nuclear stress is clearly on the preposition "of". The reason is that Bob treats the preposition "from" as problematic and self-repairs it into "of"; that self-repair is invested with the nuclear stress, arguably to ensure it is noticed by the recipient.

Another, perhaps more frequent, factor causing a split of nucleus and surprisal is illustrated in fragment (6.4). Ann has been relating how she had been considering buying a house and has talked about the financial advice some real-estate agents gave her. In line 2 she reports that she ended up with a deal proposed by "bradford and BI:ng:ley", noticeably stressing and drawling "BI:ng:ley". In Stuart's response "yeah well they're Agents for bradley, bradford and bingley" the word "Agents" gets the nuclear tone. Its surprisal value is high (14.265) but not as high as the surprisal for "bradley" (14.488) or "bingley" (14.886), which get the highest values in the turn. The reason why "Agents" is invested with the nuclear tone rather than the more surprising "bradley" or "bingley" is obvious: the names have been mentioned in Ann's prior turn already and hence no longer present any new information. The new and indeed focal information in Stuart's turn is the fact that the people Ann was given the advice to buy a certain house were employees of the real-estate company that represents the owner of that house. The implicature is clear: their advice may not have been impartial (note also the use of 'well' in second position in the turn: it provides advance-notice that the upcoming turn will not convey a simple confirmation of Ann's decision to go for the Bradford and Bingley deal):

(6.4)

1	Ann:		and (0.3) and one thing and another but er (.) it (.)
2			i mean (my mind) ended up with the bradford and BI:ng:ley (i mean)
3	Stuart:	→	**yeah well they're Agents for bradley, bradford and bingley**

(BNC: KB7 3495-3496; corrected transcription)

Extracts (6.3) and (6.4) are useful reminders that surprisal is operationalized here in a rather crude way (namely based on bigrams and corpus frequencies) and that it will therefore not always reflect what *surprises*, or fails to surprise, us in interaction. What it is unable to take into account are factors such as self-repair or prior mention. But even these two are not the only confounding factors impacting a speaker's decision where to place the nucleus; as shown in a multi-variate regression analysis (Rühlemann & Schweinberger, in preparation) nuclear placement is indeed correlated with a large number of factors. But the factor discussed here, surprisal, did prove to be a significant main effect in that model. So our discussion in this chapter rightly suggests that surprisal does co-determine nucleus placement in turns-at-talk.

6.2 The dotchart in the case study

We upload the data for this case study, stored as "Chapter6_Casestudy_Task.txt" on the companion website, and store it in R as `NUC`:

```
NUC   <-   read.table("[Your   path]/Chapter6_Casestudy_Task.txt",
header=T, quote="", sep="\t", fill=F)
```

This is a large dataset, as we can see from calling `head(NUC)`:

```
> head(NUC)
                                                       Turn  File
1 claims suspended for twenty six weeks by our good selves   KE3
2  elliel 's our discount because you know they 're doing   KBD
3             next time it wo n't be monday when we go .   KBW
4           no , but like bill , he 'll go down to tescos   KB2
5      no , they 've put them down in third division now .   KBD
6          no it 's alright i 'll probably manage with it   KC1

   Speaker        s1       s2        s3        s4         s5
1 KE3PSUNK 3.9068906 4.906891  2.90689   7.858860 4.70711433
2    PS03W 2.0000000 3.000000 10.290782 11.454813 5.64385619
3    PS089 3.9869043 3.776981  5.246336  8.220917 0.08361612
4    PS01U 0.7856258 5.331017  7.289107 13.419697 5.96413455
5    PS03W 0.7856258 5.499289  3.845281  6.386932 3.88398770
6    PS09E 0.7856258 3.894631  1.897266  6.259666 3.37149826

        s6        s7        s8        s9       s10 Nucleus    d1
1 5.095397  8.845490  7.556719 11.454813 12.998767       1 0.265
2 3.572999  3.117821  5.632733  2.452716  5.188287       1 0.453
3 5.436599 12.649032  5.952405  3.432537  5.489014       1 0.459
4 5.578019  4.245372  5.015054  3.623643 14.580043       1 0.259
5 6.409574  5.385654 13.467446  6.639039  5.727920       1 0.223
6 4.279486  5.905011 10.491352  4.721099  3.917913       1 0.246

     d2    d3    d4    d5    d6    d7    d8    d9   d10
1 0.402 0.108 0.241 0.343 0.255 0.186 0.169 0.175 0.687
2 0.044 0.318 0.447 0.203 0.186 0.239 0.062 0.051 0.244
3 0.517 0.152 0.132 0.134 0.131 0.575 0.157 0.109 0.339
4 0.203 0.243 0.212 0.062 0.055 0.113 0.205 0.052 0.728
5 0.081 0.089 0.159    NA 0.228 0.091 0.204 0.306 0.199
6 0.048 0.044 0.125 0.037 0.031 0.201 0.211 0.161 0.066
```

Beside some reference variables such as `Turn`, `File`, and `Speaker`, it contains not only the surprisal values in `s1`, `s2`, etc., as well as the central variable `Nucleus` given as a word's position in the turn (for example, `Nucleus==1` means that the nuclear tone fell on the first word in the turn), but also the word durations in `d1`, `d2`, etc. We see that `Nucleus` evaluates to `1` in all the first six rows. This is not by chance but because the dataframe has been ordered by `Nucleus`. The duration columns will be needed, not in the case study, but later in the Task section.

The goal in the case study is to plot the placement of nuclear stress against the ranked surprisal values in each slot. The values for nuclear stress are all readily available in `Nucleus`, the surprisal values however are still raw and unranked.

To rank them, we first extract them and store them separately, in `S`. Let's inspect the first row:

```
> S[1,]
        s1       s2       s3      s4       s5       s6       s7       s8
1  3.906891 4.906891 2.906891 7.85886 4.707114 5.095397 8.845490 7.556719

         s9       s10
1  11.454813 12.998767
```

Which value is the greatest, which the smallest? In `S[1,]`, the maximal surprisal value is 12.998767 in column `s10`, the minimal value is 2.906891 in column `s2`. So the former should get ranked first (i.e., 1), the latter last (i.e., 10). How is the ranking computed? It could not be easier: we use `apply()`, which takes as input the data, `S`, and `1` if the function should be applied to rows (`2`, if the function is to be applied to columns), and finally the function itself, `rank`. Two tweaks, however, are necessary. First, the function `rank()` by default works in ascending order. We prefer the (more intuitive) descending order. The function `rank()` itself cannot be set to decreasing. What we can do though is give a minus to the whole dataset `S`, thus effectively reversing the value relation across the rows. Second, we need to transpose the results using the function `t()`, which turns rows into columns and vice versa:

```
> SR <- t(apply(- S, 1, rank))
```

Has the ranking been successful? Upon inspecting the first row of `SR` we see that, as expected, the value in `s10` is ranked 1, that is, greatest – whereas the value in s3 is ranked 10, that is, smallest:

```
> SR[1,]
 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
  9  7 10  4  8  6  3  5  2   1
```

Now we have the data we wish to plot in the dotchart: the nucleus positions and the surprisal ranks. We could start the dotchart right away. However, for our dotchart it will be necessary to manipulate the code underlying the `dotchart()` function. The necessary manipulations concern the x-axis and the frame: we do not want the x-axis `dotchart()` produces by default and we do not want the default frame around the chart. Unfortunately, unlike in other graphics, `dotchart()` does not allow these features to be disabled by setting appropriate values to arguments. But we can disable them in the code for `dotchart()` itself. If we call just `dotchart`, without preceding question mark and without succeeding `()`, the full code underlying the function is made available (only a small portion is printed here for space considerations):

```
> dotchart
function (x, labels = NULL, groups = NULL, gdata = NULL, cex = par("cex"),
   pt.cex = cex, pch = 21, gpch = 21, bg = par("bg"), color = par("fg"),
   gcolor = par("fg"), lcolor = "gray", xlim = range(x[is.finite(x)]),
   main = NULL, xlab = NULL, ylab = NULL, ...)
{
(omitted code)
     axis(1)                                                 # x-axis
     box()                                                   # box
title(main = main, xlab = xlab, ylab = ylab, ...)
invisible()
}
<bytecode: 0x1154e64c0>
<environment: namespace:graphics>
```

Toward the end of a very long piece of code, we find the instruction to plot the x-axis, in `axis(1)`, and to draw a box, in `box()`. We can disable these features by copying the whole code, defining it as a new function, say, `dotchart.new`, and commenting out the two pieces of code `axis(1)` and `box()` by putting `#` in front of them:

```
> dotchart.new <- function (x, labels = NULL, groups = NULL, gdata
= NULL, cex = par("cex"),
(code omitted)
+ # axis(1)                                        # disables x-axis
+ # box()                                          # disables box
+ title(main = main, xlab = xlab, ylab = ylab, ...)
+ invisible()
+ }
```

We have finally completed the preparations for the dotchart. We start by setting the parameters for the plot layout calling `par()` with its arguments `mfrow`, `mar`, and also `xpd`, which we set to `T` or `TRUE` (it defaults to `F` or `FALSE`). This is to avoid that the text symbols we want to print, get cut off on the left and the right of the plot; this is also one of the reasons why we have disabled the box above (the other being that the graph looks nicer without a box).

```
> par(mfrow = c(1,1), mar = c(2,1,2,1), xpd = T)
```

Now we use `dotchart.new()`, feeding into it `NUC$Nucleus` as data input, defining the title with `main` and setting its font size with `cex.main`; the next argument, `pch = ""`, is critical as it makes sure that the nucleus positions do *not* get plotted – why? The reason is simple: if we printed the nuclei positions now and the surprisal ranks later, the ranks would overplot the nuclei dots and greatly reduce the clarity of the plot.

Dotcharts characteristically draw horizontal lines; these are useful when the dataset is small but distracting when the dataset is large, as in our case. We therefore set the line color to white with `lcolor = "white"`. Finally, `xlim = c(1,10)` defines the range of the x-axis.

```
> dotchart.new(NUC$Nucleus,
+              main = "Nucleus and Surprisal",
+              cex.main = 0.90,
+              pch = "", # plots nothing
+              lcolor = "white",
+              xlim = c(1,10))
```

We add the x-axis labels, using `axis()`, selecting `1` for the x-axis, placing the ticks at a sequence from 1 to 10 in steps of 1 with `at = seq(from = 1, to = 10, by = 1)`,[29] and reducing the font size of the axis with `cex.axis = 0.85`:

```
> axis(1,
+      at = seq(from = 1, to = 10, by = 1),
+      labels = c(paste("w", 1:10, sep ="")),
+      cex.axis = 0.85)
```

In the plot and the legend we will need suitable colors for the ten surprisal ranks. We cherry-pick colors which nicely form a continuum from very light yellow to very hot red from the `heat.colors()` palette:

```
> hc <- c("#FF4000FF", "#FF6000FF", "#FF8000FF", "#FF9F00FF", "#FFBF00FF",
+        "#FFDF00FF","#FFFF00FF", "#FFFF2AFF", "#FFFF80FF", "#FFFFD5FF")
```

To finish off the preliminaries, we place the legend. The first pair of arguments to `legend()` are the start points on the x- and the y-axis: here, we select `0.35` on the x-axis and, to make sure that the legend gets printed *above* the color cells, we select the whole `length(NUC$Nucleus)` and add a little more, `+ 20`. Since we have numerous legend labels, which it will be hard to cram into a single line, we reduce the distances between them, by setting `x.intersp = 0.4`. We define the 12 legend labels with `c()`, determine their font size with `cex` and their colors with `text.col`. Further, we align the labels horizontally (the default being vertical) with `horiz = T` and get rid of the default box around the legend as a whole with `bty = "n"`.

The next few arguments all concern the legend boxes: we set `fill = c("blue", NA, hc)` to fill them with colors, choosing `"blue"`for the first, `NA` for the second (as `"Ranks:"` is just a descriptive term, not a variable), and the ten colors from the `hc` vector defined above. Finally, the colors of the borders around the legend boxes are

29. Note that `seq()` works equally well without the words in front of the arguments.

defined; this becomes necessary as we do not want to have a (visible) box in front of "Ranks:":

```
> legend(0.35, length(NUC$Nucleus) + 20,
+         x.intersp = 0.4,
+         c("Nucleus", "Ranks:", "1", "2", "3", "4", "5", "6", "7",
          "8", "9", "10"),
+         cex = 0.55,
+         text.col = c("blue", rep("black",11)),
+         horiz = T,
+         bty = "n",
+         fill = c("blue", NA, hc),
+         border = c("black", "white", rep("black", 10)))
```

We move on the the meat of the dotchart: the surprisal ranks and the nucleus positions. For the surprisal ranks we define a `for` loop with nested `ifelse()` statements. As the data are 10-word turns, with one word and one rank per position in the turn, we start the loop with `for(i in 1:10)`. The ranks are inserted using the `text()` function. The first two arguments to `text()` are the location on the x-axis and the location on the y-axis. The x-axis location is one of ten word positions, so we set `i`; the y-axis location is the sequence of indices from the first `NUC$Nucleus` value to the last, so we use `seq(NUC$Nucleus)` (we could equally well put `1:542` as there are 542 `Nucleus` values). The third argument to `text()` is the text to be printed: we choose a simple straight line. The most important element in the graph are the colors for the suprisal ranks, which we have defined in `hc`. We assign these colors in `ifelse()` statements. The first `ifelse()` statement could be glossed thus: if the surprisal rank in any of the ten columns in `SR` equals `1`, then use the first color in `hc`; the instruction what to do when the test fails is passed on to the next `ifelse()` statement: if the rank in any of the ten columns in `SR` equals `2`, then use the second color in `hc`, and so forth until rank `9`; all remaining values – that is, rank 10 – are assigned the tenth color in `hc`.

```
> for(i in 1:10){
+   text(x = i, y = seq(NUC$Nucleus), "________",
+         col = ifelse(SR[,i] == 1, hc[1],
+                 ifelse(SR[,i] == 2, hc[2],
+                   ifelse(SR[,i] == 3, hc[3],
+                     ifelse(SR[,i] == 4, hc[4],
+                       ifelse(SR[,i] == 5, hc[5],
+                         ifelse(SR[,i] == 6, hc[6],
+                           ifelse(SR[,i] == 7, hc[7],
+                             ifelse(SR[,i] == 8, hc[8],
+                               ifelse(SR[,i] == 9, hc[9],
+                                                 hc[10]))))))))),
                                                  cex=0.8)
+ }
```

We have now visualized the surprisal ranks. The nuclei are still missing. We insert them into the chart again using `text()`, with the values stored in `NUC$Nucleus` on the x-axis and `seq(NUC$Nucleus)` on the y-axis; we choose a small underscore as text symbol in blue color and reduced font size:

```
> text(x = NUC$Nucleus, y = seq(NUC$Nucleus), "_", col = "blue",
cex = 0.8)
```

We're almost done. To increase the legibility of the graph, we want to insert segments separating the distinct nucleus positions 1, 2, 3, etc. First, we make a frequency list of the nucleus positions:

```
> f <- table(NUC$Nucleus); f
1   2  3  4  5  6  7  8  9 10
18 38 39 39 63 54 76 66 68 81
```

From that list we need to compute the cumulative sums, that is, the sequence from the first value in `f`, `f[1]`, to the sum of `f[1] + f[2]` to the sum of `f[1] + f[2] + f[3]` and so forth. The function `cumsum()` does just this:

```
> cum <- cumsum(f); cum
 1  2  3   4   5   6   7   8   9  10
18 56 95 134 197 251 327 393 461 542
```

Based on the cumulative sums we can draw the segments. First, with `par(xpd = F)` we set `xpd` back to its default to avoid that the segments extend beyond the colored cells. Then, we define a `for` loop `for(i in cum[1:9])`, that is, for the all the values in `cum` but the last (the last segment is omitted simply for aesthetic reasons) and insert the command that starting from `x0 = 0.55` to `x1 = 10.45` on the x-axis and for every `i`-th value on the y-axis a dotted black segment be drawn:

```
> par(xpd = F)
> for(i in cum[1:9]) {
+ segments(x0 = 0.55, x1 = 10.45, y0 = i, lty = 'dotted', col= "black")
+ }
```

6.3 Task: Nucleus placement and duration

Above we cited research showing that predictability goes hand-in-hand "with shorter duration and with less phonological and phonetic detail" (Jaeger 2010: 24). In this task we will focus on the first part of the quote, duration, and its relation to nucleus placement. The previous analysis suggested that the nucleus does not tend to fall on predictable words but on surprising words. So, putting one and one together, we can hypothesize that the nucleus will fall on words that are not only surprising but also long in duration.

The dataset you can test this hypothesis on is the same as the one underlying the case study in Section 6.1 (called "Chapter6_Casestudy_Task.txt"). The durations are stored in columns `d1`, `d2`, and so forth until `d10`. The nucleus positions are stored in column `Nucleus`.

To code the dotchart you can adapt the code underlying the case study in 6.1. There are only two differences to take note of. First, beware of data structure: after uploading the data, do check which format the variables are in. Since you are going to work with the `Nucleus` column and the ten `duration` columns, focus your attention on them: if, for example, the duration columns are not all numeric then you will have to convert

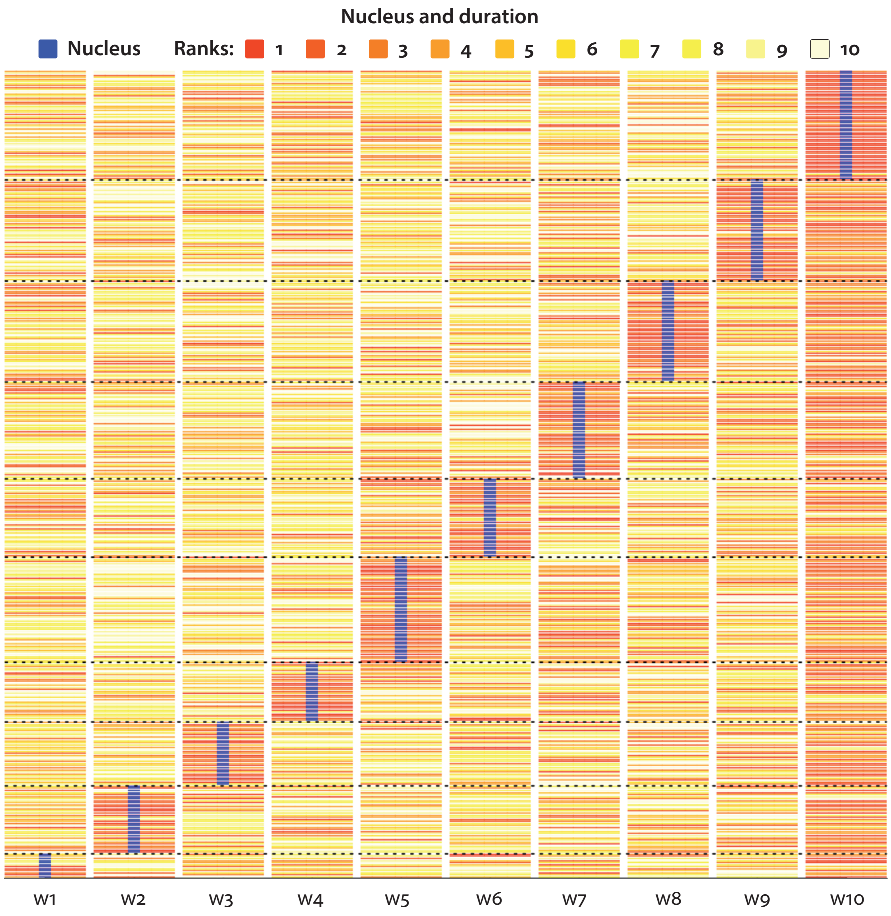


Figure 6.4 Nucleus and duration

them accordingly. Remember that conversion from factor to numeric requires nesting – first to character then to numeric: `as.numeric(as.character(x))`.
Second, not not every word in the dataset could be measured reliably, in which case its duration was set to NA. It is advisable to get rid, not only of the individual cell where duration is `NA`, but, as we are computing ranks, a relative measure *comparing* a set of values, the whole row in which one or more than one cell are `NA`. This is accomplished using the `na.omit()` function.

The rest of the code is analogous to the code in the case study.

Depending on the graphical parameters you choose, your plot would look roughly like the one shown in Figure 6.4.

The plot reveals two things. First, most turns end on either relatively long or even the longest word in the turn; this is shown by the massive amount of color from the orange/red spectrum in position w10. This tendency for speakers to elongate turn-final words or to use words in turn-final position that are long in terms of phonemic size will concern us in more detail in Chapter 11.

Second, as with nucleus and surprise, we see a correlation between nucleus and duration: many nuclei are at the same time the single longest word in the turn and, where this is not the case, the nucleus will often rest on the second- or third-longest word. So duration is quite likely to also influence a speaker's choice as to which word to invest with the nuclear stress and it will have to be included among the predictors in a multifactorial model of nuclear placement (cf. Rühlemann & Schweinberger, in preparation).

Chapter 7

Heatmaps and dendrograms

7.1 *Case study*: Turns and speech rate

7.1.1 Introduction

One of the paralinguistic design features of turns is the speed of speaking. How fast a speaker speaks depends on a number of factors. One factor is demographic: elderly speakers tend to speak more slowly than younger speakers (Amerman & Parnell 1992). Another is the action a speaker is performing: an informing turn may be delivered more slowly than the acceptance of an invitation. Also, speed depends on the size of the turn: short turns are spoken with greater speed than long turns (Yuan 2006). Speed may vary with the speaker-language relation: non-native speakers have a slower speaking rate than native speakers (Riggenbach 1991).

These factors all concern speed variation *across* turns. But what about speed variation *within* turns? One obvious source of tempo variation within turns are hesitations: when we search for a word we will slow down or halt altogether. But hesitations are not related to turn structure: they can occur pretty much anywhere in the turn. There is substantial research to suggest that speakers tend to slow down toward the end of turns; we will review this research in more detail in Chapter 11. But is the turn end the only structural place where speakers decelerate? One of the few studies addressing this question is Yuan et al. (2006). He observes that turn-first words are spoken more slowly than all other words in the turn except the last word; on articulating that turn-last word speakers decelerate again.

In this case study we will address speech rate variation as a function of turn position. That is, the research question asked is, *Does speech rate vary across positions in turns?*

7.1.2 Data and methods

The data we will use is the same 10-word turn sample from the BNC we analyzed in the previous chapter.

The variable turn position can easily be derived from the data available. We are interested in how speed of speaking develops from the beginning of the turn to its end. We can thus define each and every word in the turn as a separate position. Under this definition, a turn has as many positions as it has words and a word's position will be 1 if the word is the first word in the turn, 2 if the word is the turn-second word, and so forth. As the words in the sample are all ten words long, we will thus have ten positions.

The variable speech rate is harder to come by. Speech rate refers to "the speed of speaking" (Crystal 2003: 386). It is here operationalized as duration per phoneme. You might think: why not take the durations of the words used in a turn and compare them? Duration per word won't work as words have different intrinsic lengths given their different phonemic make-up; e.g., 'it' /ɪt/ has two phonemes, whereas 'tomorrow' /təˈmɒrəʊ/ has six phonemes (diphthongs such as əʊ count as a single phoneme); that is, 'tomorrow' will, to an extent, *inevitably* have longer duration in discourse than 'it'. Taking duration per phoneme neutralizes differences in phonetic size. (But obviously, even duration per phoneme is an approximate measure as phonemes too have intrinsic durational differences, with diphthongs being longer than monophthongs, affricates longer than plosives, etc.)

While our data does include word durations, it fails to include information on the words' phonetic sizes (i.e., the number of phonemes). How to get at that information? The steps taken to determine the phonetic size of the word forms in the 10-word turn sample were the following:

i. first a frequency list of all the word types in the sample was created in R and stored as a data frame,
ii. the word types in the frequency list were read into <https://tophonetics.com/>, a website offering free conversion of English text to IPA phonetic transcription, and converted to phonetic transcription,
iii. the transcriptions were added as a new column to the data frame containing the word types and their frequencies,
iv. the transcriptions were checked and corrected manually where the automatic conversion had failed (e.g., in case of infrequent words, such as names, or non-canonical spellings)
v. phonetic stress symbols were removed and following common phonetic practice to treat diphthongs as a single phoneme, diphthongs were collapsed into a single character,
vi. to establish phonetic sizes, the number of phonemes in each word type was counted by counting the number of characters, and finally,
vii. each phonetic transcription as well its phonetic size was matched to its respective word token in the sample.

To calculate speech rate, the duration of any word token in any turn was divided by its phonetic size. For example, to return to 'tomorrow', the first occurrence of 'tomorrow' is in turn #47: "With a bit of luck we'll know tomorrow"; that token's duration as a whole is 398 ms. To determine its duration per phoneme, 398 ms was divided by 6; the resulting speech rate per phoneme is 66.33 ms.

We will use two closely related visualizations to 'see' how speech rates develop across turn positions, one building on the back of the other.

The first is the heatmap, a highly intuitive graphic that substitutes numeric values with colored cells. Unlike heatmaps in general use, heatmaps in R include dendrograms as used in cluster analysis. Cluster analysis is an *exploratory* method in statistics; unlike most others, it does not serve to test but to *generate* a hypothesis (cf., for example, Rühlemann & Hilpert 2017).

The aim in doing cluster analysis is to find groups in data by showing "which of a (potentially large) set of samples are most similar to one another, and to group these similar samples in the same limb of a tree" (Crawley 2007: 742; cf. also Levshina 2015: 309 ff.). The grouping is based on a dissimiliarity matrix and executed iteratively, beginning with the most similar samples leading up to the most dissimilar ones: initially, the algorithm treats each sample as its own cluster, then merges the two most similar samples, compares the similarities of all remaining samples, merges the next two most similar samples in a cluster, and continues in this way until there is just a single complex cluster. As a result, dendrograms – as well the heatmap associated with them – *re*-group samples to reflect the (dis)similarity between them; this may be significant if a sequential order is inherent to the data getting clustered – as is the case with speech rates across turn positions. In dendrograms, an important visual diagnostic is the height at which two or more samples are merged into a cluster: the lower the height of the merge on the y-axis the more similar the merged samples, and conversely, the higher the merge, the more dissimilar the merged elements (cf. Levshina 2015: 309).

In the heatmaps in R, the dendrogram is an additional visual element placed above the heatmap proper. This placement may restrict the visibility of the branches of the dendrogram, specifically of their height relations. This is a non-trivial restriction because, as noted, the height of the merge points is an important indication of (dis)similarity. We are therefore also going to draw a dendrogram in an independent graph – our second visualization. The free-standing dendrogram not only allows a better view of how the clusters relate to one another but also allows the analyst to locate in the dendrogram *what he or she hypothesizes are the main clusters*.

That is, based on the clusters in a dendrogram and on prior inspection of the data in the heatmap an analyst can form a hypothesis and, in a second step, test that hypothesis in an analytic-statistical test. In the present connection, the cluster analysis will suggest the hypothesis that the speech rates fall into *n* main clusters. To assist the analyst in the task of hypthesizing the number of main clusters, the so-called 'average silhouette width' is a useful statistic (cf. Levshina 2015: 311). The statistical test administered to test that hypothesis will serve to determine whether the samples subsumed in the *n* main clusters are significantly different. Which test to use for this scenario depends on whether the data are normally distributed. As the data turn out to violate normality,[30] the appropriate test is the Wilcoxon rank sum test, which is a 'distribution-free' test assuming no particular distribution (Dalgaard 2008: 99).[31]

So the methodological road map in this case study starts with inspection of the heatmap, leads to inspection of the dendrogram, which leads to formation of a hypothesis, and, finally, culminates in an statistical examination of that hypothesis.

7.1.3 Results

The heatmap in Figure 7.1 presents a lot of information at once.

To start with, the color key is shown in the left upper corner, featuring five colors representing the five ranges the speech rates were grouped into: from white for the shortest range of speech rates (7 ms–137.2 ms) to darkgrey for the longest range of speech rates (527.8 ms–658 ms). The colors are overlaid with a density line; its peak is at the boundary of the second (137.2 ms–267.4 ms) and third range (267.4 ms–397.6 ms), suggesting that the bulk of the speech rates are found around this point.

Above the heatmap we find the dendrogram. Looking at the x-axis labels we instantly see that the speech rate samples have indeed been regrouped: most notably, `srate10` is now a neighbor to `srate1`. As noted, the build-up of dendrograms starts from the bottom; that is, the first cluster is formed between the two most similar objects, in our case `srate8` and `srate7`. Moving upward from this initial most-similar cluster we find a number of other, also relatively similar clusters, until we arrive at the very last cluster, which links `srate1` to `srate10`. This grouping indicates that `srate1` and `srate10` are the most dissimilar from the rest and yet relatively similar to each other.

30. To determine non-normality, a visual means was used: the Q-Q (quantile-quantile) plot. This plot shows the ordered sample values on the y-axis against the quantiles of the standard normal distribution (cf. Levshina 2015: 53). The analysis of the plot is straightforward: if the data line up straight, the data can be assumed to come from a normal distribution (Dalgaard 2008: 74); if there are bumps or breaks, the data should be considered non-normally distributed.

Obviously, normality can also be tested analytically. Normally, the Shapiro-Wilk test is used (check out `?shapiro.test` at the R console). In the present case, this test cannot be used as one of the samples is larger than 5,000 data points. An alternative test would be the Kolmogorov-Smirnov test. Underlying both tests is the null hypothesis that the data conform to normality. To reject that null hypothesis (and, hence, assume non-normality) the test must produce a p-value smaller than the commonly accepted threshold of 0.05.

31. The appropriate test for normally-distributed samples is the t test (e.g., Dalgaard 2008: 95 ff.); check out `?t.test` in R.

The relative similarity between `srate1` and `srate10` can also be read off the color codings in the heatmap, where the black and green colors, which indicate slow speech rates, are clearly much more numerous in these two positions than for any other turn position. However, while these slow rates are fairly evenly distributed across `srate10`, they are much less evenly dispersed in `srate1`: here, the slow rates are much fewer (but still more numerous than in `srate2` to `srate9`) and thus present a stark contrast to many very short rates (shown in the blue and white color cells).

The heatmap also features trace lines (in orange); the distance of the line from the center of each cell (marked by the stitched red line) is proportional to the size of the measurement. It is obvious that in `srate1` and `srate10`, the trace lines more often than not veer toward the right end of the cell, meaning that the speech rate is slower than the grand median, which is shown in the red line; the trace lines for the other turn positions are much more aligned with the median.

In sum, what the heatmap suggests is a separation of the data into three groups – or clusters: `srate1`, `srate10`, and the remaining speech rate samples taken together.

Is this confirmed in the separate dendrogram?

As can be seen in the separate dendrogram in Figure 7.2, where the heights are uncompressed, the distance on the y-axis between `srate10` and `srate1` is the largest in the whole dendrogram, which indicates that the dissimilarity is greatest between these two samples. This is in contrast to the visual impression we formed upon inspecting `srate1` and `srate10` in the heatmap, where the similarity beween `srate1` and `srate10` seemed noticeable. The dendrogram suggests that that similarity, though true, should not be overstated: it will probably not suffice to claim that the two samples together form a main cluster or that we are dealing with three main clusters (one for `srate1`, one for `srate10`, and one for all non-extreme positions in the turn the 'rest'). Rather, the dendrogram suggests that the data fall into two groups: a main cluster for `srate1` and another main cluster for `srate2` to `srate10`.

As the visual impressions from the heatmap and the dendrogram seem to be in conflict with one another it is advisable to consult the afore-mentioned average silhouette width. That statistic shows the 'well-formedness', or closeness, of the clusters in a given dendrogram. Its values range from 0 to 1, with values close to 0 indicating that the clusters in a dendrogram are not sufficiently distinct and those close to 1 indicating clear separation; widths smaller than 0.2 indicate insufficient cluster structure (cf. Levshina 2015: 311). Computing the average silhouette width for the present speech rate data produces a clear result, which unequivocally supports the visual impression gained from inspecting the dendrogram: the only width greater than 0.2 is for a two main-cluster solution, namely `srate1` and all other speech rate samples taken together. The two main clusters are highlighted by the rectangles in Figure 7.2.

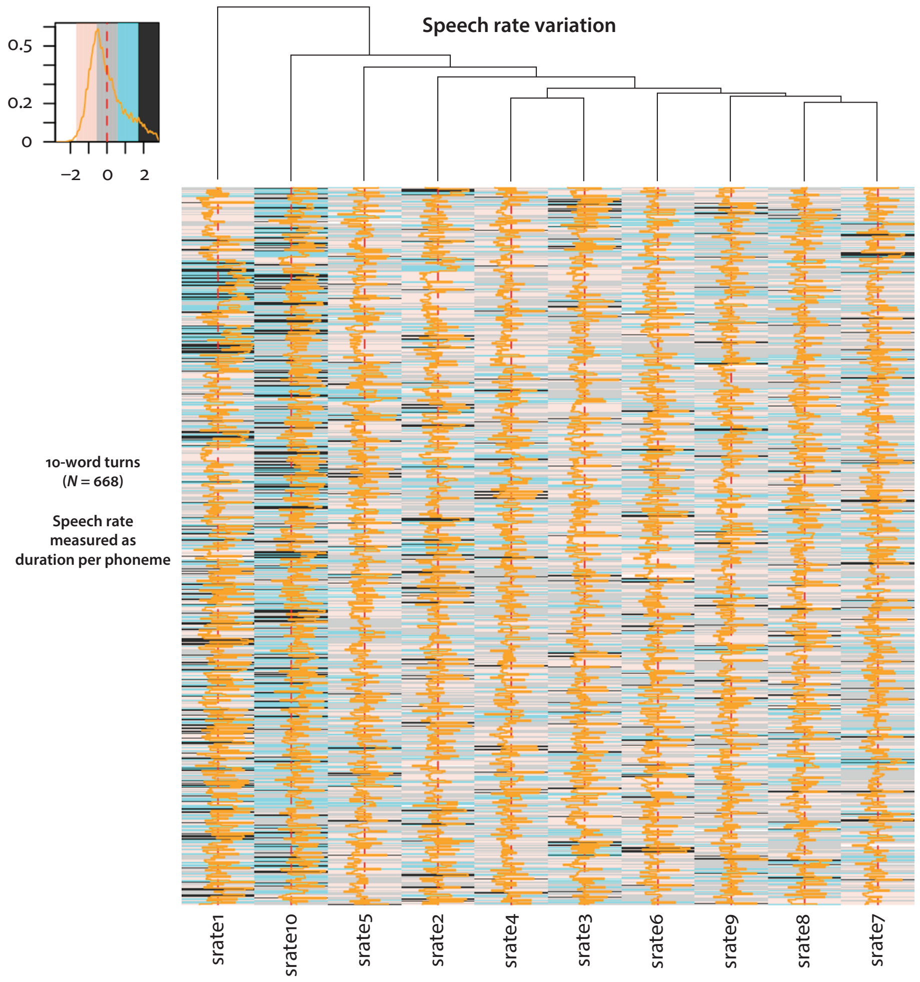


Figure 7.1 Heatmap with dendrogram of speech rates in 10-word turns; dark colors indicate slow speech rates, light colors indicate fast speech rates; vertical red lines represent grand-median speech rate; orange zig-zag lines trace the proportional distance of the speech rates from the grand median speech rate

We take this two-way division as the basis for the statistical test and hypothesize that the speech rates in the two main clusters are (significantly) different. The results of the Wilcoxon rank sum test conclusively support the hypothesis: the p-value is far smaller than the 0.05 threshold; we can accept our hypothesis: speech rates in the two main clusters vary very highly significantly (W = 2212172, $p < 0.001$).

In sum, we can now answer our research question, *Does speech rate vary across positions in turns?* The answer is yes. Speech rates fall into two main clusters: the speech

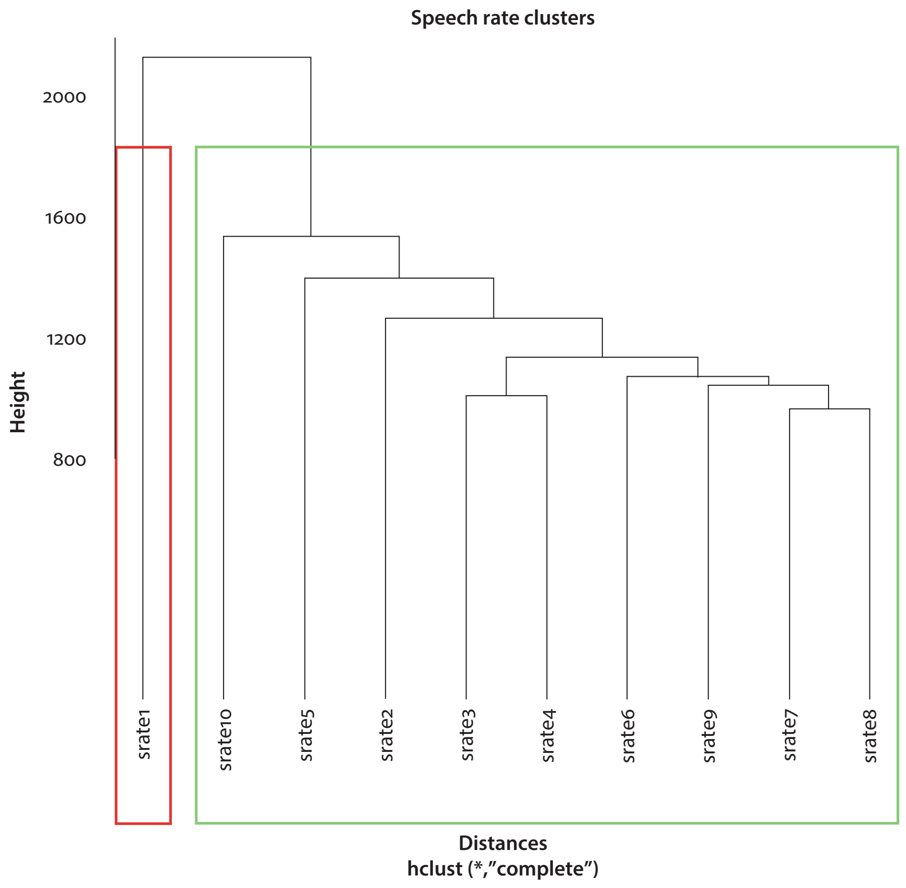


Figure 7.2 Cluster dendrogram of speech rates in 10-word turns; color rectangles indicate corresponding main clusters

rates (i) on the turn-first word and (ii) on the words in all remaining turn positions, including the turn-final word. The speech rates in these clusters are significantly different from one another: speech rate is, on average, slower on the turn-first word and faster thereafter. In a nutshell, speakers start slowly and speed up through the turn; the slow-down again upon turn-completion is relatively minor.

7.1.4 Discussion and concluding remarks

The observations made on the heatmap and the dendrograms can be summarized as follows: speech rates on the turn-initial word are slower than on any other word in the turn including the turn-last word. How can it be understood in a wider sense?

The finding that turn-initial speech rate is slow is consistent with Yuan et al.'s (2006) finding reported above. Two possible interpretations come to mind. First, speech planning may be involved. While there is wide consensus that next-speakers in face-to-face communication start planning their speech *during* the current-speaker's turn-in-progress (e.g., Magyari et al. 2014; Levinson & Holler 2014; Levinson & Torreira 2015; Roberts et al. 2015; Levinson 2016; Holler et al. 2018; Roberts & Levinson 2017), there is slightly less consensus as to whether next-speakers have already a *complete* turn plan when they actually take it. For example, Levinson & Torreira (2015) claim that, following "early gist comprehension with speech act apprehension […] the production system may automatically begin to formulate right down to the phonology […] with the actual articulation held in buffer until the comprehension system signals an imminent completion of the incoming turn" (Levinson & Torreira 2015: 12). This seems to imply that speakers start their turn with a complete turn plan on hand. Others, contrarily, leave room for continued, in-turn planning: Sacks et al. (1974: 719), for example, concede that a speaker may start to speak without requiring "that the speaker have a plan in hand as a condition for starting" and Roberts et al. (2015) acknowledge that turn-inital disfluencies may indicate that speakers begin a turn at talk without having fully planned it "in order to 'buffer' their comprehension or planning" (Roberts et al. 2015: 2). The relative deceleration of speech rate we observe in the present analysis on the turn-first word may serve to buy the speaker more in-turn planning time. Indirect evidence to back up this claim is the fact that turn-initial words are among the most frequent words in a language at all (Rühlemann, forthcoming b). Frequency, in turn, reduces planning costs and production time: for example, in Jescheniak & Levelt's (1994) picture naming experiment, high-frequent words were articulated 62 ms faster than low-frequency words (cf. also Levelt et al. 1999; Indefrey & Levelt 2004). So, a paradox arises: speakers spend more time on words that actually cost them less. It seems that a plausibel explanation for the paradox is that speakers use the turn-initial slot as an in-turn planning hub.

Second, it seems that the slow speech rate in turn-initial position is a reflection of the special role of the turn-initial item, which, as we have seen in Chapter 4, is in many cases a pre-start, that is, an item, that functionally relates the incipient turn to the prior turn. As has been argued recently, the precise action adumbrated by the pre-start may be subtly correlated with the pre-start's phonetic design, including its duration. A prime example is 'well', which can be drawled considerably: in Rühlemann (2018c), the maximum duration was almost 700 ms. In that study, the author hypothesized that elongated 'well' (fully articulated and typically separated from the turn-second word by a pause) prefaces *more substantial* disagreements.

To illustrate this point, consider (7.1), where Albert and J are exchanging assessments of Phil (who is not co-present):

(7.1)

1	Alb:	Is that June's boyfriend then.
2	J:	**we:ll** (0.601) that Phil, (.) he's a (0.454) screw I think.

(BNC: KB1 5128-5129; corrected transcription)

Albert's question in line 1 "Is that June's boyfriend then." is a simple yes/no question; it is delivered with statement intonation, suggesting that he is expecting a positive answer. However, instead of the preferred positive response J in line 2 provides a scathing critique of Phil: "he's a (0.454) screw I think" – this not only does not answer the question but it also makes Albert look utterly naive for unsuspiciously assuming Phil might be June's boyfriend when he is in actual fact a screw (which is why, by implication, he cannot or at least should not be June's boyfriend). Also, it undercuts his implicit assessment of Phil of a 'normal' person, that is, someone who might qualify as June's boyfriend. Clearly, J's response runs against the bias of Alb's innocent question: not only does it disagree with the course of action initialized by Albert's yes/no question but it also disagrees with his implicit assessment of Phil. In this extract, then, we find a 'well' that is greatly decelerated, fully articulated, and followed by a substantial pause and that is used to preface a highly dispreferred speech act.

However, as always, there are counter examples: 'well' can also be speeded up (and articulatorily reduced and phonetically integrated with the TCU); if it is, Rühlemann hypothesizes that 'well' indicates minor disagreements, as in extract (7.2):

(7.2)

1	Vicki:	Northampton, Geoff was born in [Wiltshire].
2	Heidi:	[Where] would you say I was bred.
3		Here¿
4		(0.8)
5	Vicki:	mhm.
6		(12.1)
7	Joan:	**we'** you spent your (.) <u>first</u> four years in Malton.

(BNC: KC3: 1311-1315; corrected transcription)

In the extract in (7.2) 'well' is fleetingly short with 59 ms, heavily reduced, and, thus, prosodically fully integrated into the subsequent TCU. The speakers are discussing where family members were born and raised. Heidi in line 2 inquires about where she was bred, and in line 3, offers the deictic reference "here" as a candidate location. Vicki's "mhm" in line 5 confirms the location. This Q-A sequence is followed by a very long pause of more than 12 seconds. The silence is broken when Joan expands the sequence with "we' you spent your (.) <u>first</u> four years in Malton." The nuclear stress is audibly on the adjective "first", suggesting that Joan treats the noun phrase "your <u>first</u> four years" as the focal information, not the place reference "Malton". On this interpretation, Joan's turn provides, not a disagreement with what was established so far – that Heidi was raised in Malton – but a restriction to that information, namely that she was raised "here" only the first four years of her life. This does not fully upset what the participants had agreed on – that Heidi was bred in Malton – but merely refines the informational status quo. The 'well'-prefaced turn in line 7, then, represents a weak dispreferred introduced by 'well' spoken with very fast speed.

So, to judge from these examples (and others discussed in Rühlemann 2018c), one reason for the noticeable variance in speech rate on the turn-initial word may be due to function-related differences in the phonetic design of pre-starts. There may also be others reasons. A full exploration of all reasons is far beyond this case study and must await future research.

7.2 The heatmap and the dendrogram in the case study

How were the heatmap and the dendrogram made in R?

Space considerations prevent going into details on the process of transcribing the words in the data phonetically (which was done mostly automatically but also involved manual post-editing; see above) as well as the alignment of words and their phonetic transcriptions. The data we are going to start from already contain the results of these processes; that is, the file "Chapter7_Casestudy.txt" to be downloaded from the companion website includes not only the turns and the individual words but also the phonetic sizes and phonetic transcriptions; the code is found in file "Chapter7_Casestudy_Code.R".

We will read-in the file and store it as `speed`:

```
> speed <- read.table("[Your path]/Chapter7_Casestudy.txt", header=T,
quote="", sep="\t", fill=T)
> head(speed)
  File Speaker                                                   Turn
1 KCH    PS1BT Ah , but what you 're thinking there is china clay .
2 KBE    PS04H  And Clayton 's not allowed to open his big mouth .
3 KBK    PS05X         And I 'm not a great lover of weeding , cos
4 KBW    PS088        And I bring my work home cos it 's finished .
5 KBP    PS066        And I ca n't remember what she said it was .
6 KBW    PS087          And I thought , oh no Sarah do n't do it !

   d1  d2  d3  d4  d5  d6  d7  d8  d9 d10 phon1 phon2 phon3 phon4
1 136 115  81  84 109 373 254 157 426 231     2     3     3     3
2 177 282  77 183 164 111 148 153 137 265     3     5     1     3
3 262 156  91 153  62 219 303 171 316 302     3     1     1     3
4 132 239 214 234 258 381 171  64 105 644     3     1     4     2
5 186  89  79 101 387 106 139 169 122 241     3     1     2     2
6 103 168 198 188 359 343  64  75  95 367     3     1     4     1
```

```
  phon5 phon6 phon7 phon8 phon9 phon10 trans1 trans2 trans3 trans4
1     1     6     2     2     4      3     ɑː    bʌt    wɒt    juː
2     4     3     4     3     3      3   ænd kleɪtn      z    nɒt
3     1     4     4     2     6      3   ænd     aɪ      m    nɒt
4     4     3     3     2     1      6   ænd     aɪ   brɪŋ    maɪ
5     7     3     3     3     2      3   ænd     aɪ     ca     nt
6     2     4     3     2     3      2   ænd     aɪ   θɔːt     əʊ
     trans5   trans6 trans7 trans8  trans9 trans10
1         r ˈθɪŋkɪŋ    ðeə     ɪz ˈʧaɪnə    kleɪ
2    əˈlaʊd      tuː ˈəʊpən    hɪz    bɪg    maʊθ
3         ə    greɪt  ˈlʌvə     ɒv ˈwiːdɪŋ     kɒs
4      wɜːk     həʊm    kɒs     ɪt       z ˈfɪnɪʃt
5 rɪˈmɛmbə      wɒt     ʃiː    sɛd      ɪt     wɒz
6       nəʊ   ˈseərə     duː     nt     duː     ɪt
```

As can be seen from calling `head(speed)`, the file contains more information than is needed for the heatmap and the dendrogram. This essential information is stored in the ten columns in which word durations are recorded, that is, in `d1`, `d2`, etc., as well as the ten columns in which the phonetic sizes are recorded, `phon1`, `phon2`, etc. Let's reduce the data so that it will contain only those twenty columns and store it in a new dataframe `speed2`:

```
> speed2 <- speed[, 4:23]
> head(speed2)
   d1  d2  d3  d4  d5  d6  d7  d8  d9 d10 phon1 phon2 phon3 phon4
1 136 115  81 84  109 373 254 157 426 231     2     3     3     3
2 177 282  77 183 164 111 148 153 137 265     3     5     1     3
3 262 156  91 153  62 219 303 171 316 302     3     1     1     3
4 132 239 214 234 258 381 171  64 105 644     3     1     4     2
5 186  89  79 101 387 106 139 169 122 241     3     1     2     2
6 103 168 198 188 359 343  64  75  95 367     3     1     4     1
  phon5 phon6 phon7 phon8 phon9 phon10
1     1     6     2     2     4      3
2     4     3     4     3     3      3
3     1     4     4     2     6      3
4     4     3     3     2     1      6
5     7     3     3     3     2      3
6     2     4     3     2     3      2
```

While this data gets us closer to the heatmap and the dendrogram, an important intermediary step is still missing: the calculation of speech rate. As said earlier, speech rate is computed as duration divided by phonetic size. We perform this division in one go by subsetting the whole dataframe and using the function `paste0()`, which allows us to target the twenty columns without referring to each of them separately:

```
> speed2[, paste0("srate",1:10)] <- speed2[, paste0("d",1:10)] /
speed2[, paste0("phon",1:10)]
> head(speed2)
   d1  d2  d3  d4  d5  d6  d7  d8  d9 d10 phon1 phon2 phon3 phon4
1 136 115  81 84  109 373 254 157 426 231     2     3     3     3
2 177 282  77 183 164 111 148 153 137 265     3     5     1     3
3 262 156  91 153  62 219 303 171 316 302     3     1     1     3
4 132 239 214 234 258 381 171  64 105 644     3     1     4     2
5 186  89  79 101 387 106 139 169 122 241     3     1     2     2
6 103 168 198 188 359 343  64  75  95 367     3     1     4     1

  phon5 phon6 phon7 phon8 phon9 phon10
1     1     6     2     2     4      3
2     4     3     4     3     3      3
3     1     4     4     2     6      3
4     4     3     3     2     1      6
5     7     3     3     3     2      3
6     2     4     3     2     3      2

     srate1    srate2 srate3 srate4    srate5    srate6    srate7
1 68.00000  38.33333   27.0   28.0 109.00000  62.16667 127.00000
2 59.00000  56.40000   77.0   61.0  41.00000  37.00000  37.00000
3 87.33333 156.00000   91.0   51.0  62.00000  54.75000  75.75000
4 44.00000 239.00000   53.5  117.0  64.50000 127.00000  57.00000
5 62.00000  89.00000   39.5   50.5  55.28571  35.33333  46.33333
6 34.33333 168.00000   49.5  188.0 179.50000  85.75000  21.33333

     srate8    srate9   srate10
1 78.50000 106.50000  77.00000
2 51.00000  45.66667  88.33333
3 85.50000  52.66667 100.66667
4 32.00000 105.00000 107.33333
5 56.33333  61.00000  80.33333
6 37.50000  31.66667 183.50000
```

Now we have the data to be plotted: this data is contained in the 10 columns `srate1`, `srate2`, etc. But we don't have the format yet: heatmaps require data in matrix format. So we need to re-format the ten speech rate columns. We do it by using `as.matrix()` and storing the matrix as `srate`:

```
> srate <- as.matrix(speed2[,21:30])
```

To check whether the conversion to matrix was successful, we use `class()`:

```
> class(srate)
[1] "matrix"
```

The heatmap discussed above is not the heatmap available from base R. It is contained in the package `gplots`, so we need to install the package first using `install.packages()` and making it available for the session using `library()` (NB: `install.packages()` requires quote marks around the package name, `library()` does not!):

```
> install.packages("gplots")
> library(gplots)
```

The heatmap function is called `heatmap.2()`. Its first argument is the matrix, `srate`. Next, we set the parameters for the key (if no key is sought, set `key = F`), by defining `keysize` and setting `key.title`, `key.xlab`, and `key.ylab` to `NA`; further, we set `density.info` to `"density"` to obtain a density line (other options are `"histogram"` or `"none"`).

The default setting for margins in `heatmap.2` is (5,5), which leaves (too) much space around the heatmap empty. We therefore decrease the margins with `margins = c(4.5,1)`. Also, we do not seek row labels, so we suppress them with `labRow = ""`.

The next few lines concern the dendrogram above the heatmap. `Rowv = F` instructs R not to draw a dendrogram for rows. As we want the dendrogram to be computed for columns we write `dendrogram = "column"`. The argument `scale = "row"` indicates that the values should be centered and scaled in the row direction.

Now we set the colors for the heatmap. A basic principle in choosing colors for plots is to achieve sufficient contrast. One way is by manipulating color transparency; as will be remembered from Chapter 2, this can be accomplished by `adjustcolor()` and its argument `alpha.f`. Now we could use, for example, a single color and define different hues (levels of transparency) for it. In the code below, a different color scheme is chosen: using `sample()` and drawing one color randomly for each transparency level, we can change the look of the graph drastically in a single click on the 'run' button of the R editor (provided that the whole code for the heatmap is marked) and can do so endless times until we get a fully pleasing plot. Note also that the five colors thus obtained obviate the need to define the number of `breaks`, that is, those points at which the values should be split into colors (if defined, the number of `breaks` must equal the number of colors +1).

Finally, we arrive at the settings for the trace line showing the degree to which a value deviates from the median. First, `trace = "col"` determines that the trace line should be applied to columns; next, the argument `tracecol` defines the trace line color and `linecol` defines the color for the dashed center line.

```
par(mfrow = c(1,1), mar = c(4,4,4,4))
> heatmap.2(srate,
+           keysize = 0.9,
+           key.title = NA, key.xlab = NA, key.ylab = NA,
+           density.info = "density",
+           margins = c(4.5,1),
+           labRow = "",
+           Rowv = F,
+           scale= "row",
+           dendrogram = "column",
+           col = c(adjustcolor(colors()[sample(1:650,1)], alpha.f
                    = 0.01),
+                   adjustcolor(colors()[sample(1:650,1)], alpha.f
                    = 0.25),
+                   adjustcolor(colors()[sample(1:650,1)], alpha.f
                    = 0.5),
+                   adjustcolor(colors()[sample(1:650,1)], alpha.f
                    = 0.75),
+                   adjustcolor(colors()[sample(1:650,1)], alpha.f
                    = 1)),
+#          breaks = 6,
+           trace = "col",
+ tracecol = "orange",
+ linecol = "red" )
```

We're almost done. What is still missing are the title and some descriptive text. The title is set using `title()`, placing it on a suitable `line` and adjusting the character size with `cex.main`:

```
> title(main = "Speech rate variation", adj = 1,
+       line = 3,
+       cex.main = 0.9)
```

Finally, we add a description to the left of the heatmap. Plotting anything *outside* the plotting area requires special permission; we grant this permission by setting in `par()` the argument `xpd` to `T`. To break the text into smaller segments that will fit the margin size, we use `\n` (for new line) at appropriate points in the text:

```
> par(xpd = T)
> text(0.2, 450,
+      "10-word turns\n(N = 668)\n\nspeech rate\nmeasured as\n
      duration per phoneme",
+      cex = 0.5)
```

The dendrogram, drawn as a separate graph as in Figure 7.2 above, is comparably easy to obtain. As noted before, dendrograms depict dissimilarities between samples. These are computed using the function `dist()`. Dissimilarity is operationalized in R as distance. There are different ways to measure distance. The default one, and the most commonly used, is the euclidian distance (for a useful discussion of this measure and other distance measures, cf. Levshina 2015: 309 ff.). By default, `dist()` computes the distances between the *rows* of a matrix. What we need, however, are the distances between the *columns* in `srate`. So we transpose the matrix, using the function `t()`:

```
> srate_t <- t(srate)
```

To compute the column distances, we call `dist()` for the transposed matrix and store the distances in a new vector `distances`:

```
> distances <- dist(t(srate), method = "euclidian")
```

If we inspect `distances` we can make a clear prediction: the distance between `srate7` and `srate8` is the smallest. Bearing in mind that clustering starts from the most similar samples leading up to the most dissimilar ones, these two samples will then be the first to get merged in a cluster. We also see that the distance is the largest between `srate1` and `srate10`, so these two will get merged last:

```
> distances
            srate1    srate2    srate3    srate4    srate5
srate2  1894.2509
srate3  2054.2973 1247.3219
srate4  2003.5988 1205.2017 1009.2812
srate5  2130.9504 1400.0916 1269.5916 1166.2709
srate6  2026.4913 1244.6610 1134.9720 1036.0755 1179.6798
srate7  1998.5535 1256.3724 1097.7831 1050.8472 1213.2253
srate8  1970.8385 1213.9076 1114.6530 1090.8353 1238.1394
srate9  2022.6206 1267.1548 1137.3140 1086.6513 1243.3356
srate10 2049.6332 1525.7504 1510.5942 1437.5199 1536.7883

            srate6    srate7    srate8    srate9
srate2
srate3
srate4
srate5
srate6
srate7  1072.0629
srate8  1067.6023  964.4838
srate9  1046.0096  999.0200 1044.9260
srate10 1455.1625 1441.6043 1430.9277 1370.5236
```

To finally plot the dendrogram we use the function `hclust()`, which takes the distance matrix `distances` as its first argument. There are several cluster methods, again the default one and the most commonly used is the method `"complete"` (see, again, Levshina 2015: 309 ff. for a good discussion of alternative methods). First, to make the code more compact for later on, we store the dendrogram as an object, `srate_dg`:

```
> srate_dg <- hclust(distances, method = "complete")
```

Then we set the stage for the plot by determining the layout, for example, thus:

```
> par(mfrow = c(1,1), mar = c(5.5, 4, 2, 2))
```

To actually obtain the dendrogram we wrap the `plot()` function around the object `srate_dg`. Inserting the argument `hang = -1` ensures that all clusters labels 'hang' from the same height at the y-axis.Additionally, we specify the heading, the label on the x-axis and the font sizes:

```
> plot(srate_dg,
+      cex = 0.9,
+      hang = -1,
+      main = "Speech rate clusters",
+      xlab = "Distances",
+      cex.main = 0.95, cex.lab = 0.9, cex.axis = 0.9)
```

The next step is critical: we want to know which clusters can be grouped into main clusters. The package `cluster` allows you to compute what is called average silhouette widths, a measure to identify the optimal cluster solution. So we first install the package and load it:

```
> install.packages("cluster")
> library(cluster)
```

To understand what's behind the computation of average silhouette widths let's take it in steps. Let's assume we hypothesize that the optimal cluster solution will be three clusters. To compute the silhouette widths for three clusters we will use two functions, `silhouette()` and `cutree()`. The latter takes as input our speech rate dendrogram `srate_dg` as well as the hypothesized number of clusters, `3`; `silhouette()` in turn takes as input `cutree()` as well as the distance matrix `distances`. We will store the result of this operation as `sil.k3`:

```
> sil.k3 <- silhouette(cutree(srate_dg, k = 3), distances)
```

What is the average silhouette width for a three-cluster solution? To address this question we can do a `summary()` of `sil.k3` and specifically target the value of interest, namely `avg.width`:

```
> summary(sil.k3)$avg.width
[1] 0.1735641
```

The `summary` output shows that the silhouette width for a three-cluster solution is suboptimal: 0.1735641 is smaller than the threshold value of 0.2 (see above). So we will want to look for a better cluster solution. We can compute the average silhoutte widths for all possible cluster solutions in one go using `sapply()`. A one-cluster solution does not make sense at all, neither does a 10-cluster solution. Therefore we exclude these solutions and define the sequence 2:9 as the first argument to `sapply()`. As the second argument we define a `function(x)` for which we adapt the code used for the three-cluster solution using `summary()` and `cutree()` and simply replace `3` with `x`:

```
> sil.k2to9 <- sapply(2:9,
                      function(x)
                       summary(silhouette(cutree(srate_dg, k = x),
                                          distances))$avg.width)
```

The mean silhouette widths for the eight possible cluster solutions are these:

```
> sil.k2to9
[1]  0.35669702  0.17356414  0.07300514  0.07268100  0.03766443
0.02880942 0.02250442 0.01115538
```

Which value is, which values are, greater than 0.2? Only the first – the average silhouette width for the two-cluster solution, so we'll go with that.

Next, we wish to draw rectangles into the dendrogram to visualize which branches form the main clusters. This is achieved by `rect.hclust()` wrapped around `srate_dg`. Also, we set the argument `k = 2` to obtain two rectangles designating the two main clusters; finally we set `border = 2:3` to color the rectangles red (`2`), green (`3`) (use `1` for black and `4` for blue):

```
> rect.hclust(srate_dg, k = 2, border = 2:3)
```

The test, finally, requires some preparation. First, we define vectors for each of the two main clusters that we believe speech rates fall into:

```
> mc1 <- srate[,1]
> mc2 <- srate[,2:10]
```

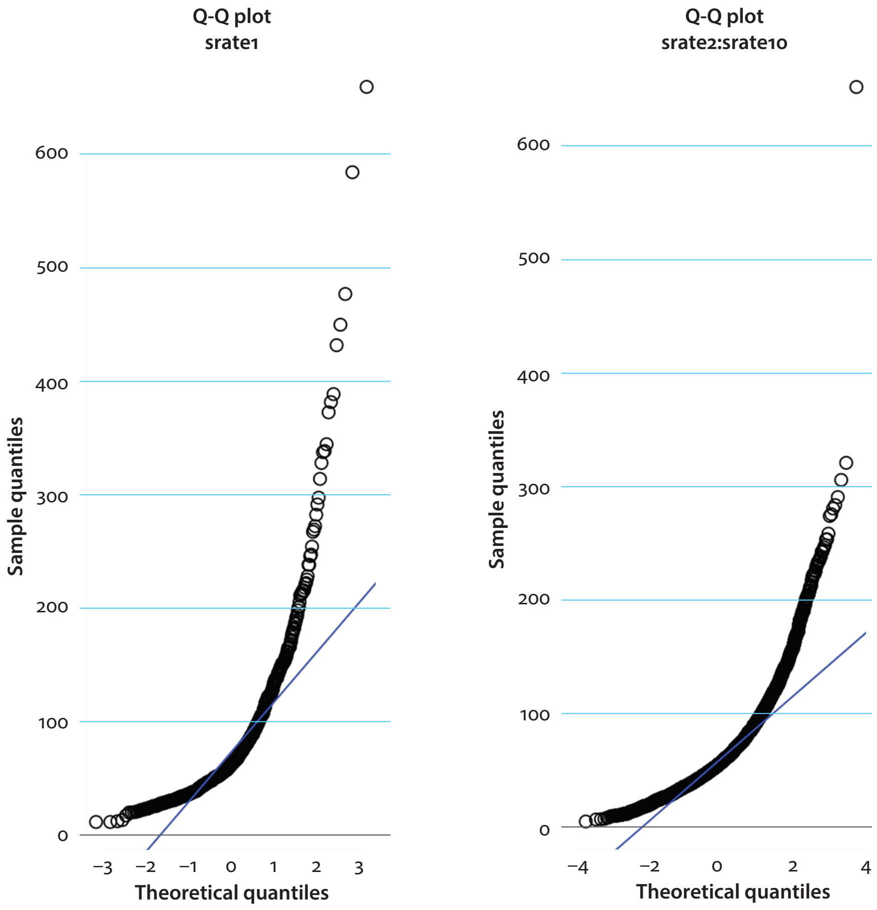
We perform the test to establish whether the data in the two vectors are significantly different. However, which test to use depends on which distribution the data come from. If they come from a normal distribution, then the appropriate test is the *t* test; if not, then a distribution-free test is warranted – such as the Wilcoxon rank sum test. How to find out whether the data are normally distributed? As noted earlier, one, visual, means is the Q-Q-plot. The code to produce the plots is simple. First, we allow for a two-panel plot in which the Q-Q plots are arranged side-by-side:

```
> par(mfrow = c(1,2))
```

Next, we produce for each of the three vectors both the Q-Q plot, using `qqnorm()`, as well as the appropriate straight line to compare the distribution against, using `qqline()`:

```
> qqnorm(mc1, main = "Q-Q plot\nsrate1",
+        cex.main = 0.9, cex.axis = 0.9, cex.lab = 0.9)
> qqline(mc1, col = "blue")
> qqnorm(mc2, main = "Q-Q plot\nsrate2:srate10”,
+        cex.main = 0.9, cex.axis = 0.9, cex.lab = 0.9)
> qqline(mc2, col = "blue")
```

The resulting graphs unequivocally indicate that the data in the two vectors do not come from a normal distribution as they deviate clearly from the straight line:



Having established that `mc1` and `mc2` are non-normally distributed, the data is all set up for testing using the `wilcox.test()` function. It takes as minimum input the two variables `mc1` and `mc2`:

```
> wilcox.test(mc1, mc2)
      Wilcoxon rank sum test with continuity correction
data: mc1 and mc2
W = 2212172, p-value = 1.577e-05
alternative hypothesis: true location shift is not equal to 0
```

The test result is reassuring: the speech rates in the two main clusters are very highly significantly different (W = 2212172, $p < 0.001$).

7.3 Task: Frequency structure in turns

In Section 5.3 we saw that word frequency is not uniformly distributed in turns but that there is a tendency for infrequent words to take up larger and larger proportions as speakers progress from a turn's beginning to the turn's end. Is this a continuous decrease or is there 'structure' in the decrease in the sense that, at some position in the turn, word frequency drops to become *substantially lesser*?

In this Task section, we are going to deploy the heatmap and the dendrogram to address this question.

The data is, again, 10-word turns form the conversational subcorpus of the BNC. This time, however, it is not a smallish subset but it is the full set – all 10,696 turns that have 10 words in the subcorpus.

How to proceed with the data to produce the heatmap and the dendrogram?

The data can be downloaded from the companion website, the file is called "Chapter7_Task.txt". On reading-in the data as, say, `t10`, you will see the file contains two reference columns, `File` and `Speaker`, the turns as strings in column `Turn` as well as the words per turn position, labeled `w1`, `w2`, etc.:

```
> head(t10)
  File Speaker                                                  Turn
1 KD9    PS1G2    That 's what he was doing , hoping I was asleep .
2 KD9    PS1G3      I 'll not get anything else done out the back .
3 KD9    PS1G3  Mark whenever you start drag him off into the cupboard
4 KD9    PS1G3 No the one downstairs has been there quite a while .
5 KD9    PS1G3 Sure wreck it , round here , do n't leave nothing alone .
6 KD9    PS1G2 Him there , his second time , first time he failed it .

    w1        w2  w3          w4       w5    w6      w7    w8
1 That        's what         he      was doing hoping     I
2    I       'll  not        get anything  else   done   out
3 Mark whenever  you       start     drag   him     off  into
4   No       the  one downstairs      has  been there quite
```

```
5 Sure     wreck  it       round     here    do    n't leave
6  Him     there  his     second     time  first  time    he
        w9      w10
1      was   asleep
2      the     back
3      the cupboard
4        a    while
5 nothing    alone
6  failed       it
```

We see that the first letter of the word tokens in `Turn` and `w1`, `w2`, etc. is sometimes upper case. This is unfortunate as the distinction between upper and lower case creates an artificial distinction between two orthographic forms of one and the same word. So we better set all text strings to lower case:

```
> t10[,3:13] <- lapply(t10[,3:13], tolower)
```

There's no frequency information in `t10` – where do we get that from? It is stored in another file, called "Chapter7_Task.BNCfreq.txt". Let's suppose you store the file as `bnc`. The number of rows it contains is impressive, with frequencies for 34,322 distinct word types:

```
> nrow(bnc)
[1] 34322
```

The first six lines look like this:

```
> head(bnc)
  Word Frequency
1 i    169024
2 you  134932
3 it   128004
4 the  115361
5 's   107729
6 and  90843
```

The first task now is to match the corpus frequencies to the words in `w1`, `w2`, etc. of dataframe `t10`. This can be done in one go by using `sapply()` and `match()`:

```
> t10[, paste0("f", 1:10)] <- sapply(t10[, 4:13],
+                                    function(x)
+                                    bncfreq$Frequency[match(x,
                                     bncfreq$Word)])
```

The next step is to transform the frequencies logarithmically, for example thus:

```
> t10[, paste0("f", 1:10,"log")] <- log2(t10[, paste0("f", 1:10)])
```

Now you have a very large dataframe:

```
> ncol(t10)
[1] 33
```

For the heatmap and the dendrogram you only need the ten columns you have last created, the logged frequencies. Isolate them and transform them to a matrix:

```
> f <- t10[, paste0("f", 1:10,"log")]
> f <- as.matrix(f)
> class(f)
[1] "matrix"
```

Plotting the heatmap with all 10,696 rows will be demanding too much – therefore take a random sample of just 500 turns:

```
> f_s <- f[sample(1:nrow(f), 500),]
```

There's no need to sample for the dendrogram: in plotting it, do use matrix `f` with all its 10,696 rows, not matrix `f_s`!

Now you're all set for the visualizations. The steps to take to produce the heat map and the dendrogram are all analogous to the ones explained in detail in Section 7.2.

Once you've managed to draw the graphs they should in principle look like the two on pages 141 and, respectively, 142.

The heatmap shows relatively little contrast. This is indication that the samples are not utterly dissimilar. Upon careful inspection we see that, as expected, frequencies start high at turn beginning and rather gradually decrease until turn completion. The heatmap then confirms the results of the case study in Chapter 5, where we observed a similar decline in frequency over the turn. But our research question in this present case study is whether there are *leaps* in rarity over the turn – that question we cannot answer based on the heatmap; we will need the dendrogram to penetrate more deeply into the frequency structure in turns.

The first thing to note in the dendrogram is that there are no large height differences indicating that all clusters are relatively similar to each other. This was to be expected given that the heatmap showed only slight color contrasts between the samples. The relative lack of dissimilarity also emerges from computing the average silhouette widths, where the largest value is slightly below the 0.2 limit. That value is for the two main-cluster solution, separating frequencies on turn completion from all other frequencies in the turn. The Wilkoxon rank sum test confirms that the difference between these two main clusters is very highly significant (W = 377557928, p-value < 2.2e-16).

So, while frequencies generally decrease slightly from turn inception to turn completion, they significantly drop upon turn completion. There is, then, structure in the way that speakers vary frequency in turns. A full appreciation of this fnding is well beyond the scope of this Task section; but some preliminary observations may be worth noting.

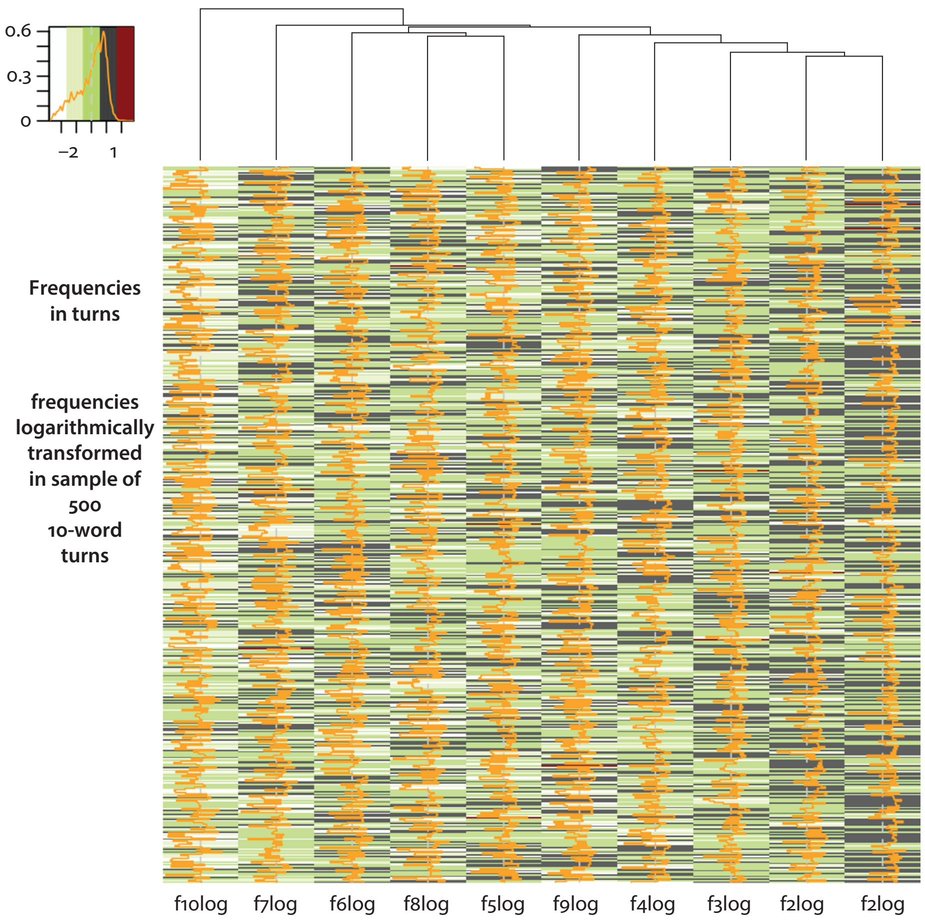


Figure 7.3 Heatmap of logarithmically transformed frequencies in random sample of 500 turns from all 10,969 10-word turns from BNC-C; dark colors indicate high frequencies, light colors low frequencies; vertical grey lines represent grand-median frequency; orange zig-zag lines trace the proportional distance of the frequencies from the grand median frequency

To start with, the finding is not restricted to 10-word turns; the same drop in frequency can be observed in turns of other sizes as well, as shown in the scatter plot in Figure 7.5, which represents average corpus frequencies per position in more than 77,000 turns ranging in size from 7 to 12 words.

Second, preliminary inspection of those turns that exhibit a sharp frequency drop in turn-last position suggests that one of the factors to explain the phenomenon are names – names of co-present interlocutors used as address forms, as in (7.3) and (7.4), or names by which speakers refer to non-present referents including people, as in (7.5) to (7.6), or places, as in (7.7) and (7.8):

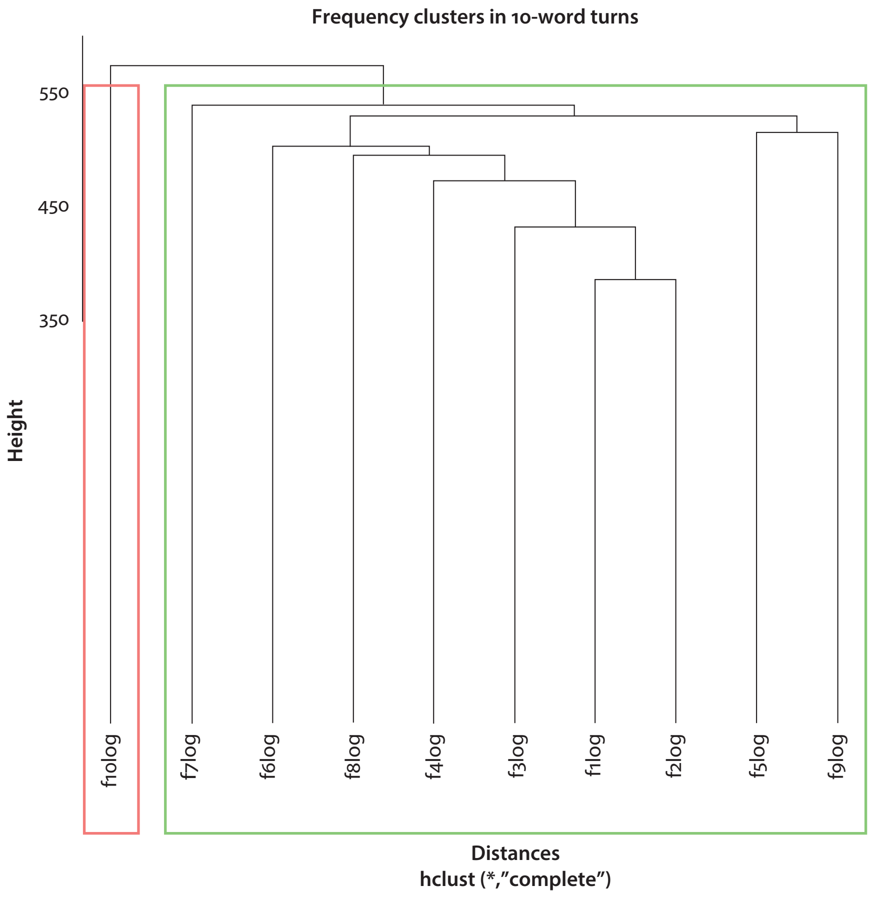


Figure 7.4 Cluster dendrogram of logarithmically transformed frequencies in all 10-word turns in BNC-C (*n* = 10,696 turns); color rectangles indicate main clusters

(7.3) so you 've got a big garden have you arth ?

(7.4) yeah but you 'd mark the ones you know , ruthy !

(7.5) er , i sent some money down for bethan and kaylie

(7.6) he was a big name at the time , laepol stokowski

(7.7) whether they 'll be in harden newlow or , or consaquay ?

(7.8) well you can spend some days on the cote d'azure

Place names and people's names generally have low frequencies (Scott & Tribble 2006: 70) as the two word classes are virtually unlimited in terms of number of members. Further, names used between close friends and family are prime targets for

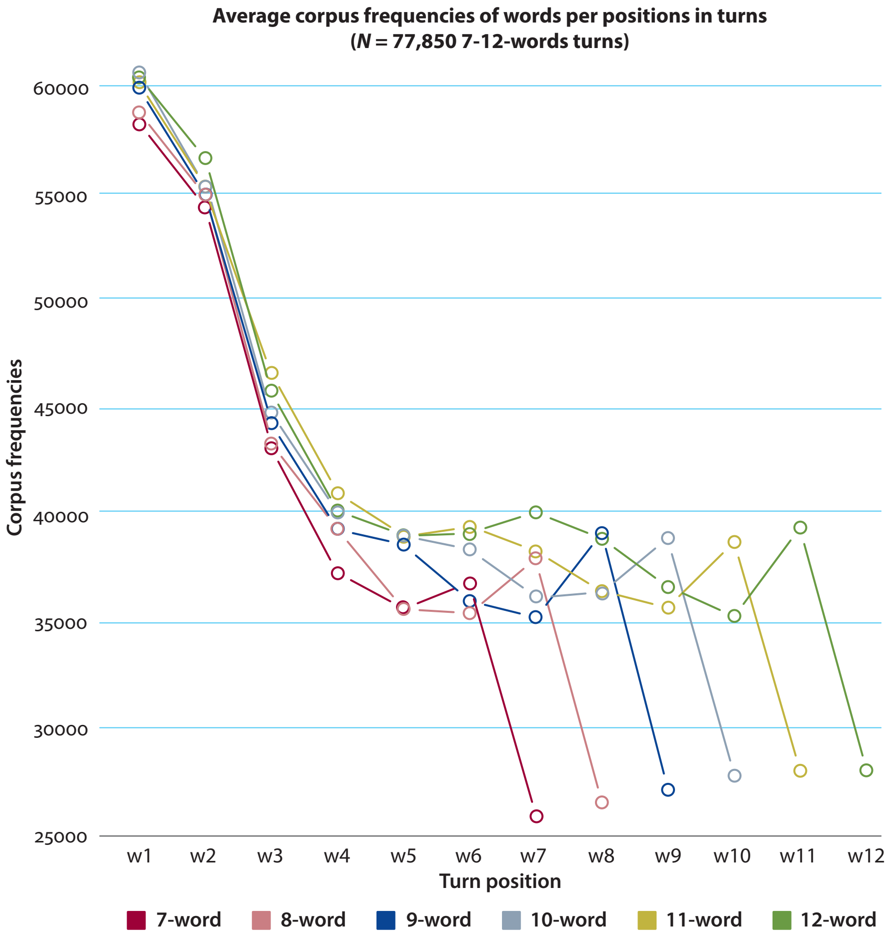


Figure 7.5 Average corpus frequencies of words in 7–12-word turns (*N* = 77,850 turns)

'morphological creativity' to express "affective connection and convergence" (Carter 2004: 101). For example, "Arth" and "Ruthie" are likely deviant forms of 'Arthur' and 'Ruth' and therefore even less frequent than the (already infrequent) standard forms. In interactional terms, the use of names in *address* forms – i.e., as post-completers (cf. Chapter 4) – serves the purpose of selecting the next speaker (Sacks et al. 1974). Seen from this perspective, the frequency drop in turn-last position may be related to turn-taking.

Perhaps more generally though the overall downward trend and its culmination in the steep slide in frequency in last position may relate to information packaging. In Chapter 5 we tested the content word climax hypothesis, and found that content words not only generally increase over the turn but spike in turn-last position. Given that content words mostly have much lower frequencies than inserts and function words we also speculated that the content word climax might be evidence of an information climax, a tendency for speakers to distribute information asymmetrically in the turn, with a bias for inserting new and focal information toward the end of the turn. It appears that the finding made in this Task section is consistent with the information climax hypothesis and provides additional evidence to support it. Consider the role of the low-frequency content word "optician" in extract (7.9):

(7.9)
1 John: It feel = feels (0.36) it- I feel as if I've got something
2 in ↓me eye all the time (1.14)
3 Arth: like sand?
4 John: [yeah.]
5 Arth: [like a like a rough] °yeah°
6 John: [yeah yes yeah]
7 (1.87)
8 → Arth: when was the last time you went to the optICian?
9 (1.10)
10 John: (jus') before you came here (°I think°) (0.77)
11 Arth: over two years ago °then°¿
12 John: yeah.
13 (1.45)
14 I'm due to go =
15 Arth: = I think you need to go (BNC: KP1 5245-5255; corrected transcription)

The talk between John and Arthur preceding the extract revolves around ailments and aches. When John complains about his eye trouble in line 1, thereby creating for Arthur an opportunity to volunteer assistance (cf. Kendrick & Drew 2016: 6), Arthur first expands the sequence by eliciting a more detailed description of the pain (lines 3–7). Then, in line 8, his question "when was the last time you went to the optICian?" offers assistance by implicitly suggesting that John consult an optician to cure his problem. There has been no mention of opticians in the preceding talk, so the reference to "optician" provides new information; that "optician" is indeed not just any new information but focal – that is, the informational climax of the turn – transpires from the nuclear stress on it (cf. Chapter 6).

The frequencies in the turn are strikingly unevenly distributed, as shown in Figure 7.6.

While the first nine words in the turn have very high frequencies – indeed "you" and "the" are among the top five most frequent words in conversation – , "optician" is very rare: it occurs merely four times in the conversational subcorpus of the BNC.

So frequency may be down at turn ends because it is at this point in the turn that speakers 'make a point', that is, get to the meat of their message, for example, by referring the recipient to some referent outside the discourse-so-far, and that 'point' is typically expressed by a low-frequency content word.

But we can here merely touch upon this potential connection between the frequency drop in turn-last position and the information climax. Much more in-depth

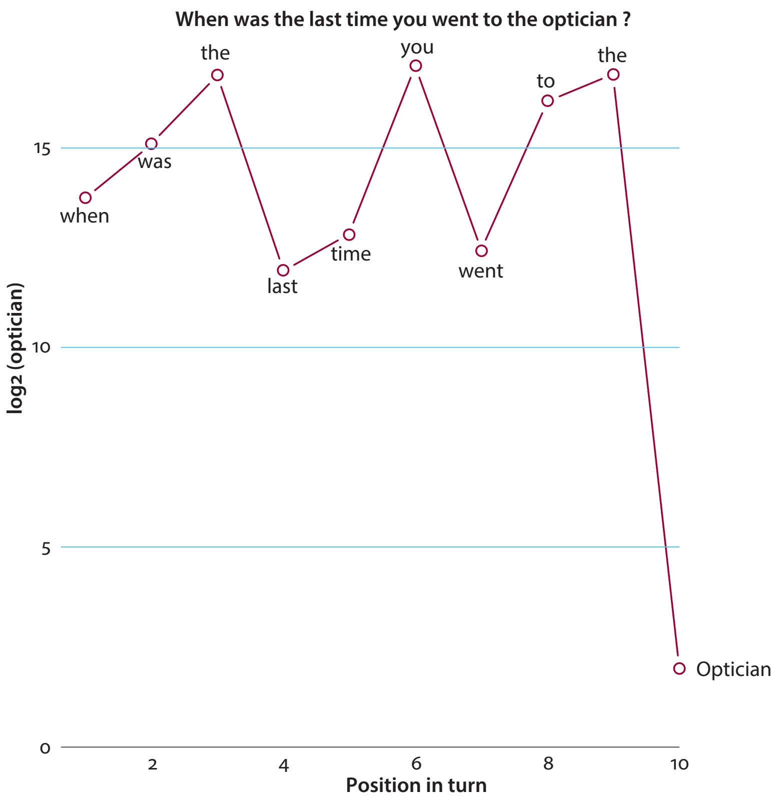


Figure 7.6 Frequencies (log-transformed) in turn "When was the last time you went to the optician?"

analysis of many more cases is required to get anywhere close to confirming the connection as a fact. What we can say with more certainty at this point is that word frequency is not uniformly distributed in turns but there is structure in how speakers use frequency to distribute words across positions in turns.[32]

We will now take a leap from the turn to the sequence, considering large stretches of thematically and interactionally related talk. Specifically we will explore the storytelling sequence, from the point of view of how speakers orient recipients to the situational parameters of the story world (Chapter 8), how they guide them prosodically toward the story highpoint (Chapter 9), and how they use paralinguistic resources to infuse constructed dialog (or 'direct speech') with stance, the lifeblood of storytelling (Chapter 10).

32. Clearly, syntax plays a role too: in an SVX language such as English, word order is heavily constrained. However, considering that the subject mostly represents discourse-old or contextually accessible information (such as 'I' or 'you'), syntax itself may be seen as an adaptation to constraints arising from turntaking (cf. Ochs et al. 1996).

Chapter 8

Strip charts and violin plots

8.1 *Case study*: Storytelling structure – The role of adnominal this in Orientation

8.1.1 Introduction

In this chapter we take a leap from the turn to the sequence. The sequence type we will be concerned with is storytelling. Storytelling is a complex task: it takes place in a telling situation, with the teller and their interlocutors talking some time some place, and revolves around a told situation made up of the events that happened to some agents some other time some other place. So storytelling intertwines two situations, the telling situation in which the telling takes place and the told situation in which the events took place.

To achieve this intertwining of two situations, storytellers need to 'orient' the recipient to the basic coordinates of the told situation; these basic coordinates include not only the spatio-temporal coordinates place and time but also the personal, or agent, dimension,[33] that is, the characters populating the story. These coordinates configure the 'ground' against which the 'figure' – the temporal sequence of events, which is "the defining characteristic of narrative" (Labov & Waletzky 1967/1997: 15) – can play out and be distinguished.[34]

It is generally assumed that the situational coordinates are laid out in what CA researchers refer to as Background (e.g., Goodwin 1986) or what Labov (1972) calls the Orientation section. This section, in Labov's model (Labov 1972: 363 ff.), succeeds the optional Abstract (a pre-view on what the story is going to be about) and precedes the Complicating action (the 'plot'), Evaluation (the 'point'), Resolution (how the events sort themselves out), and finally the Coda (which bridges over from the narrative to the present speech situation).

Obviously, for the Orientation section to provide a ground against which the events figure can be assessed, the Orientation should be sequentially early, that is, it should

33. Agents need not necessarily be persons. As discussed in Ervin-Tripp & Küntay (1997), non-human agents can be made the protagonists of stories; the example they discuss is an earthquake.

34. Note that in addition to sketching out agents, space, and time of the told situation, Orientation/Background sections can also *foreshadow* important elements/events that will be tracked and developed in later sections (cf. Baynham 2003; Dingemanse et al. 2017)

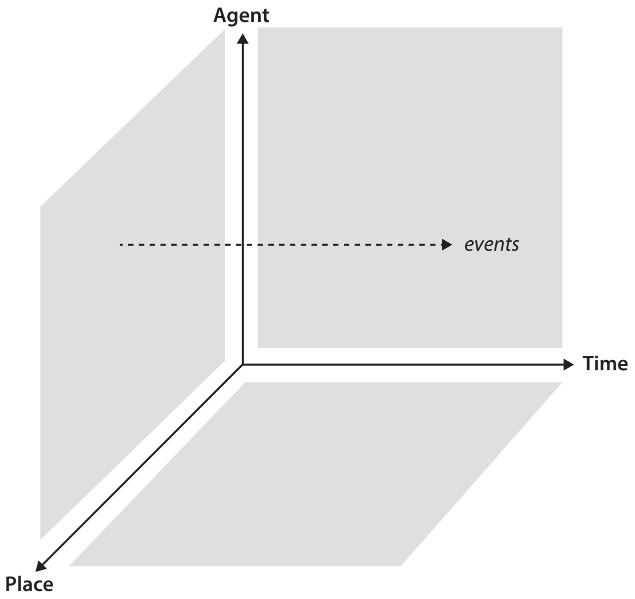


Figure 8.1 Dimensions of the told situation

precede other narrative sections. In this short case study, as well as in the Task section, we will examine linguistic means by which speakers actualize that early Orientation. In the case study we look into 'adnominal this', that is, use of 'this' as a determiner of a noun, in the Task section we will investigate the distribution of references to time in storytelling.

Consider extracts (8.1) and (8.2) as illustrations of 'adnominal this'.

(8.1)
1 Ang: >someone gave me a microwave oven **this morning** (0.4) for nothing. <
2 Sue: (what)?
3 Ang: >she rung me up and asked (got a microwave < oven)?
4 Sue: yeah¿
5 Ang: i said- (0.3)
6 (she says) "have you got one"
7 i said° no.
8 but i was gonna say Ange (has) cos i thought she wanted to defrost someth-'
9 Sue: yeah
10 Ang: she s'd well our (mother's) (0.3) got one you can have.
11 she- she s'd "you can have." =
12 → i s'd no i'll buy it° off" =
13 = she said "no our mother want no money.
14 so i spoke to her mum. <
15 (0.3) h 'n' she s'd > no you have it my love. =
16 = i've just bought a new one you can have it.
17 (0.9)

so i [(gonna get)] =
Sue: [who's this then.]
Ang: zoe.
y' know zoe?
Sue: oh yeah,
Ang: her mu'.
(1.7)
giving (away) a microwave oven for nowt.
( ): °i thought that's alright,°

(BNC: KB6 1610-1629; corrected transcription)

In (8.1), 'adnominal this' is found in the referring expression "this morning". It points to the early part of the day the story is being told; that is, it relates the told time exophorically to the telling time. The reference to "this morning" is the only mention of that time point in the story; it is not taken up again in this or any other form. By contrast, the type of 'adnominal this' used in (8.2) is of an altogether different nature:

(8.2)
Jud: °No they don't do any takeaways°.
(3.5)
Eh = Our (Arthur) (w's) sat there
and(.) **this** (.) **girl** comes clear the pots away (.)
and **she**, (.) been round lots o' tables, you know,
> collecting the cups up together and
she comes an' she goes < (.) ↑o:h **she** says you smell
Heh [heh heh]
Dor: [That's very] [(mea::n)]
Jud: [AND I LO]oked at **her**
and I thought you can't say tha::t
and Alan looked at **her** and then **she** looked
> **she** goes < ↑oh I meant **she** says you smell nice¿
H(hh)e said (1.4) he said don't get close, =
Dor: = Ye:ah =
Jud: = ↑Ooh↑ you smell
(1.5)
Dor: We're off though::: = uh (0.4) well York (0.9)

Judith is relating an incident in a restaurant where a waitress made an inappropriate remark about Arthur's smell. The waitress is referred to as "this (.) girl" as if she had just been mentioned in the preceding discourse or as if the reference could be resolved exophorically. Neither is the case: she is mentioned here for the first time and the reference cannot be decoded with recourse to entities outside of the text. This usage has been termed 'new-this' (Wald 1983) and 'introductory this' (Biber et al. 1999: 274). The usage is widespread across varieties of English (Wald 1983: 94) and particularly common in storytelling (Halliday and Hasan 1976: 61).

Not every 'adnominal this', then, is also an instance of 'introductory this.' The defining feature of 'introductory this' is functional: the referent it marks out is the story protagonist. Reconsider extract (8.1): that the waitress is the key player of the story is evidenced by the multiple anaphoric references to her using personal pronouns (highlighted in bold in the extract) following the initial mention. Given that 'introductory this' 'points' to who or what is going to be a central agent in the unfolding story, Rühlemann & O'Donnell (2015) interpret it as a type of theme marker and thus as a form of discourse deixis (see Levinson 1983: 89).

In this case study, we will explore 'adnominal this' in storytelling. The aim is to assess whether forms of 'adnominal this', including but not restricted to its incarnations as 'introductory this', serve as means for storytellers to provide the orientation story recipients need to assess the story events.

8.1.2 Data and methods

The case study is based on the c. 500 stories assembled in the *Narrative Corpus*, or NC (cf. Rühlemann & O'Donnell 2012).

Using XQuery (cf. Rühlemann et al. 2015), n-grams and c5-grams were extracted; n-grams are contiguous strings of words of variable length in running text and c5-grams are the corresponding strings of Part-of-Speech tags. For this analysis both two-grams (strings of two words), and three-grams (strings of three words) were retrievied. N-grams are computed iteratively, taking the endpoint of one n-gram as the startpoint of the next n-gram. For example, the utterance "I've tried to learn it" is analyzed thus in terms of n-grams and c5-grams:

```
       twogram  c5_twogram        threegram c5_threegram
1         i 've    PNP VHB        i 've been  PNP VHB VBN
2      've been    VHB VBN 've been trying  VHB VBN VVG
3  been trying     VBN VVG  been trying to  VBN VVG TO0
4    trying to     VVG TO0 trying to learn  VVG TO0 VVI
5     to learn     TO0 VVI      to learn it  TO0 VVI PNP
6     learn it     VVI PNP    learn it but  VVI PNP CJC
```

N-grams and c5-grams were extracted for the narrative component of the NC (the NC also contains non-narrative components, namely the conversational contexts in which the storytellings were embedded; cf. Section 8.3). The total number of n-grams extracted and underlying this analysis is 79,075.

Further, building on recent work on positioning in Discourse Analysis (e.g., Scott & Tribble 2006; O'Donnell et al. 2012), an n-gram's position is calculated as the proportion of the number of words preceding the n-gram out of the total number of words in the text. Based on this formula, positional values come to range between 0 and < 1. For example, the utterance "I've tried to learn it" is the first bit of a story counting 41 words. So the position of the first two-gram "i 've" is 0 / 41 = 0, the position of the second two-gram "'ve been" is 1 / 41 = 0.024390244, the position of "been trying" is 2 /41 = 0.048780488, and so forth.

Using regex, a subsample of the data is computed on the condition that the three-grams match the pattern 'this' plus optional adjective plus Noun. Next, a frequency list is compiled for the nouns in the n-grams in the subsample. The nouns are assigned to five semantic groups: 'person' (e.g., "this mad chap"), 'time' (e.g., "this afternoon"), 'place' (e.g., "this chalet"), 'object' (e.g., "this passport"), and 'other' used for this+N combinations that did not fit any of the four semantic groups.

To visualize the positions across semantic groups of 'adnominal this' we are going to use two related graph types, the stripchart and the violin plot. The two graph types overlap functionally to a large extent; the one function the violin plot alone brings to the table is that is also shows the kernel probability density of the data. Unlike most visualization techniques in this book, which are descriptive-statistic methods, the kernel density estimation is an analytic-statistic method, intended to make a statement not about the necessarily limited sample on hand – in our case, a few hundred instances of 'adnominal this' in storytelling in English – but about the 'population' from which the sample is taken, i.e., the entirety of uses of 'adnominal this' in storytelling in English.[35,36]

8.1.3 Results

Table 8.1 shows that the largest semantic group of 'adnominal this' is for 'this+N_person', which accounts for roughly a third of all instances. The close second is the 'this+N_time' group. These two largest groups account for more than 60% of all observations.

As depicted in the stripchart in Figure 8.2, 'adnominal this' consistently occurs in early positions in storytelling: the positional values show clear tendencies to cluster toward the beginning of the positional spectrum and the medians lines for any of the five semantic groups lie clearly left of the black dotted line showing the median position for all words in the sample (which is, obviously, 0.5). The earliest positions are for the groups 'this+N_time' and 'this+N_person'; cf. the median positions in Table 8.1.

35. Note that the skewness of the position data could equally well be shown, for example, in boxplots (cf. Chapter 10).

36. A very useful website introducing `ggplot2`'s functionality for violin plots is <http://www.sthda.com/english/wiki/ggplot2-violin-plot-quick-start-guide-r-software-and-data-visualization>

Table 8.1 Frequencies and median positions of semantic classes of 'adnominal this'

	this+N_object	this+N_other	this+N_person	this+N_place	this+N_time
frequency	30	18	49	12	43
%	19.74	11.84	32.24	7.89	28.29
median	0.3855727	0.3575189	0.2156863	0.2785436	0.1866197

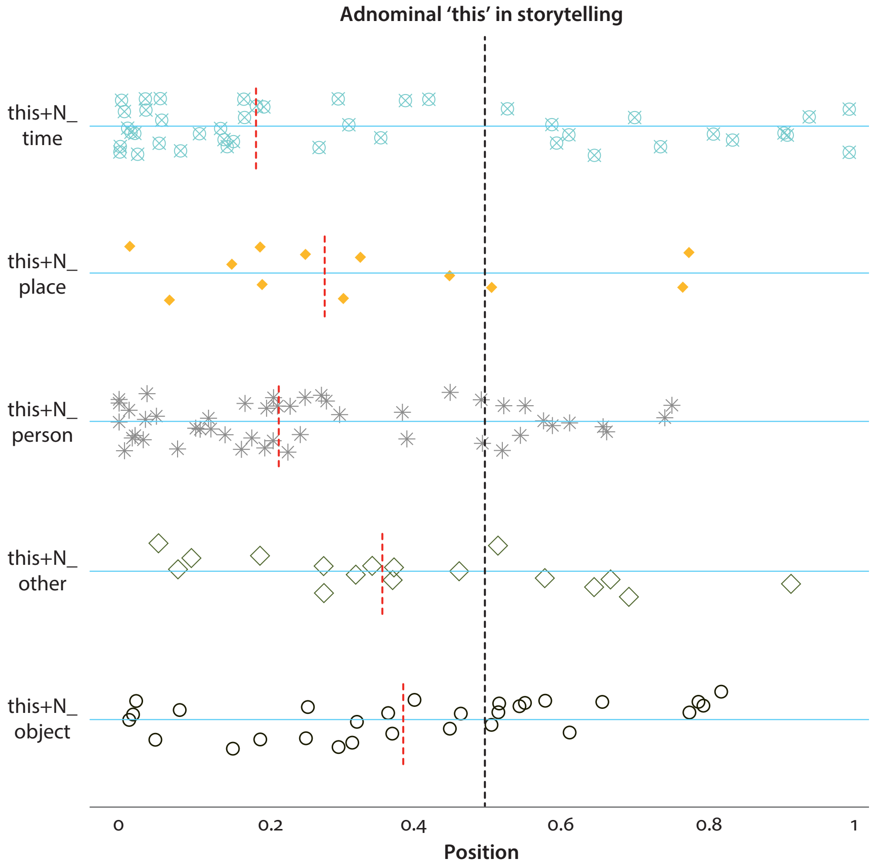


Figure 8.2 Stripchart of positions of semantic groups of 'adnominal this'; as shown by the clustering of the data points and the position of the median lines on the left side of the plot, the distribution of 'adnominal this' in any semantic class is skewed toward the story beginning

As noted, the additional information provided by the violin plot in Figure 8.3 is contained in the colored area circumscribed by the kernel density lines showing the probability estimates of positions of 'adnominal this'. The densities for most semantic

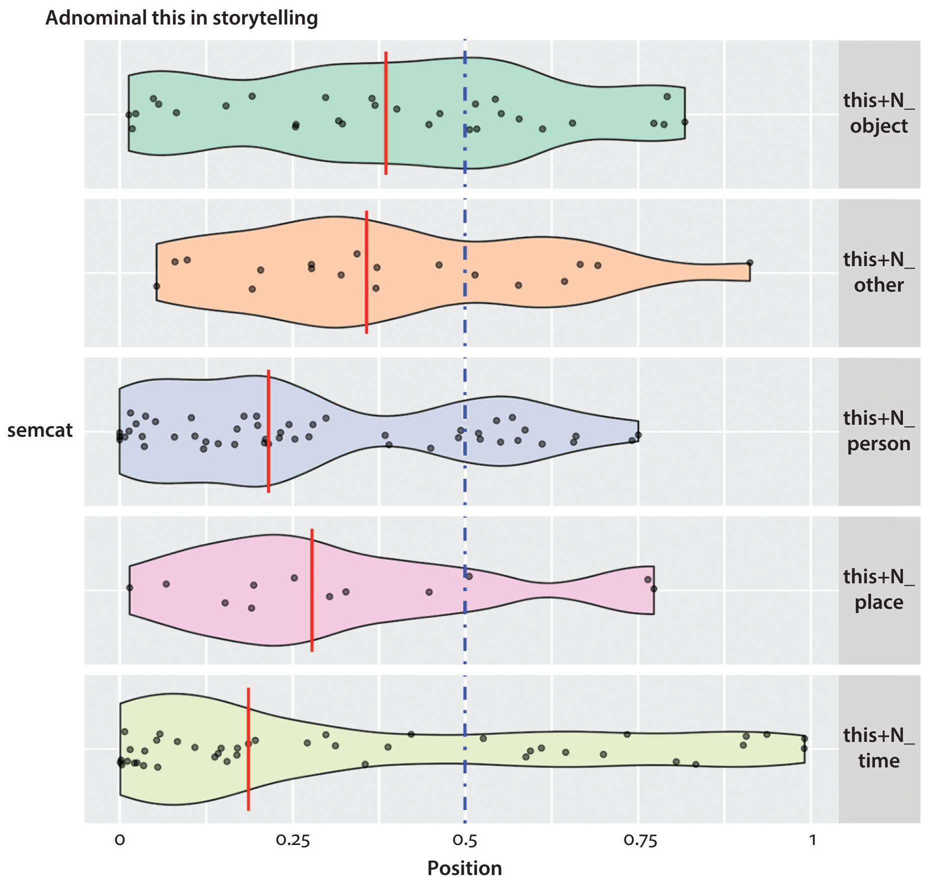


Figure 8.3 Violin plot of positions of semantic groups of 'adnominal this'; the skewness of the positions of 'adnominal this' is shown here not only by the clustering of the data points and the leaft-leaning median lines but also by the fact that the biggest 'humps' of the density lines are all far to the left of the 0.5 demarcation line

categories literally balloon even further to the left of the positional spectrum than the medians; the only category defying this trend is this+N_object, where the biggest hump forms after the median.

8.1.4 Discussion and concluding remarks

The evidence presented in the stripchart corroborates the (trivial) fact that "[p] eople often *begin* stories in conversation by referring to person, time, and place" (Dingemanse et al. 2017: 1; added emphasis). The stripchart also provides initial indication that people often begin stories by referring to *objects*. Based on the findings of the case study we can say that people begin stories by using 'adnominal this' to refer mostly to person and time but also to place and even objects. 'Adnominal this' can thus be seen as a semantically versatile means for speakers to set the scene for the benefit of the story recipients.

It is also tempting to view the correlation of 'adnominal this' and the early section of storytellings in light of Hoey's (2005) theory of 'lexical priming', which is based on the assumption is that "[e]very word is primed for use in discourse as a result of the cumulative effects of an individual's encounters with the word" (Hoey 2005: 13). The theory argues, inter alia, for a close association of lexis with *positions* in texts claiming that "every word is primed to occur in, or avoid, certain positions within the discourse" (Hoey 2005: 13); the claim is referred to as 'textual colligation.' Given its consistent occurrence in story-early positions, speakers may well be primed to use 'adnominal this' in the initial portions of storytellings.

This priming is quite likely for those instances that are introductory in the sense outlined above: when speakers use 'adnominal this' to introduce the protagonist of a story they will do so at the beginning of the story, often in the very first line, (as part of) the preface. While protagonists are mostly human agents, that is, from the 'this+N_person' group, the use of 'introductory this' also "extends to inanimates" (Wald 1983: 102), that is, to referents that would fall into the 'this+N_object' domain. In fragment (8.3), for example, Alan's story pivots around the octopus soup he once had in Germany:

(8.3)

1	Alan:	I had **this octopus** in Germany (.)
2		and it, we'd gone out for a meal and
3		(0.9)
4		I was gonna have steak and mushrooms n'
5		(0.4)
6	Barry:	mm
7	Alan:	lads said what we're havin' for starter like? =
8		= I said, oh (.) I'll have what you're ordering.
9		Well they ordered soup of the day you see (.)
10	Barry:	mm
11	Alan:	so they brought ( ) tasted **it** (.)
12		had a real funny- a <u>weird</u> taste
13	Barry:	mm
14	Alan:	anyway, and I sai- said to waitress, (why) what was **it**?
15		Sh' said **octopus**.
16		(1.1)
17		°urgh° =
18	Barry:	= you wouldn't have had **it** if you'd 've known?
19	Alan:	NO!

The notion of protagonist does not entail that the referent's centrality is necessarily manifest through its being mentioned anaphorically multiple times in the storytelling, as the reference to "octopus" in (8.3), which comes up at least four more times after its initial mention. Referents can acquire protagonistic qualities if they occur just once in the story. An excerpt to illustrate this claim is (8.4), where "this almighty swing", assigned to the 'this+N_object' group, occupies an initial position. The story is about a golfer who lost control over his golf club due to an over-agressive swing. While the swing itself is only mentioned at the beginning of the story, it can still be seen as the crucial semantic element because the subsequent events – the loss of the club, its teetering up the top of a tree and its crashing return to the ground – are all *consequences* of it:

(8.4)

1	Fra:	(had t') laugh yesterday towards the end of the e:r round.
2		it was really tipping do:wn misty fog.
3		couldn't see a hundred yards_
4		(1.3) h i had **this almighty sw-**
5		i said well i'll let the six iron,
6		i did **this almighty swing**, (0.6)
7		h the ball went one way,
8		and the club went right up the top of a tree.
9		(1.1)
10		it must've been fifteen foot at least up in the air.
11		(0.5) t- (0.3)
12		teetering on a top bra:nch.
13	Ste:	°he°
14	Fra:	i thought "oh my go:d lost me six iron"
15		a:ll of a sudden CRAsh it came down.
16	Ste:	( )
17	Fra:	old louis he didn't know which way anything had gone.
18		he said well i couldn't see any of that, (0.7)
19		didn't even see the club (go up th°) tree.

(BNC: KCI 1817 – 1821; corrected transcription)

So, 'adnominal this', including its critical incarnation as 'introductory this', is a primary linguistic resource by which storytellers accomplish the task of orienting the listener to the coordinates of the told situation. In the Task section we will examine another likely orientation-providing resource, time references.

8.2 The stripchart and the violin plot in the case study

Let's start with the stripchart in the case study: how was it configured in R?

The data, available for download on the companion website as "Chapter8_Casestudy.txt", is uploaded the usual way; the code is available as "Chapter8_Casestudy_Code.R". We will call the dataset `ngrams`:

```
> ngrams <- read.table("[Your path]/Chapter8_Casestudy.txt",header
= TRUE, sep = "\t", quote = "", comment.char = "")
```

Calling `head()` reveals that we have a `title` column with the story titles, the column `position` with the positional values and the columns `twogram` and `threegram` with the two-grams and respectively three-grams, as well as `c5_twogram` and `c5_threegram` with the respective PoS tag strings:

```
> head(ngrams)
                   title   position     twogram c5_twogram
1 Learning Welsh at age 76 0.00000000        i 've    PNP VHB
2 Learning Welsh at age 76 0.02439024    've been    VHB VBN
3 Learning Welsh at age 76 0.04878049 been trying    VBN VVG
4 Learning Welsh at age 76 0.07317073   trying to    VVG TO0
5 Learning Welsh at age 76 0.09756098    to learn    TO0 VVI
6 Learning Welsh at age 76 0.12195122    learn it    VVI PNP

        threegram c5_threegram
1       i 've been  PNP VHB VBN
2 've been trying  VHB VBN VVG
3  been trying to  VBN VVG TO0
4 trying to learn  VVG TO0 VVI
5     to learn it  TO0 VVI PNP
6    learn it but  VVI PNP CJC
```

We also see that the data contains a lot of rows without any occurrence of 'this', adnominal or other: e.g., none of the first six lines contain it. So we face the first challenge: how to subset the data so that only rows are left over that contain 'adnominal this'? This question amounts to the question, How to define a regex to match 'adnominal this'?

'Adnominal this' is, by definition, a determiner; that is, it precedes a noun. Determiners in the BNC are tagged DT0, noun tags all start with capital N. That's already a useful starting point to define the patterns, but no more than that. DT0 is used for all other determiners as well, including 'some', 'any', 'own', and many more in which we are not interested. To exclude them, we also have to match 'this' literally. One last critical consideration concerns the iterative nature of n-grams, where the same word occurs twice in two-grams, thrice in three-grams, and so forth (for example, one and the same item is both the endpoint of one two-gram and the startpoint of the next two-gram). If we do not control for this reduplication we end up matching twice the number of actual matches. We can exercise control by adding the restriction that 'this' occur as the first item in the three-gram (remember from Section 2.6 that in regex, first position in a string is expressed by the anchor `^`):

```
> this <- ngrams[grepl("^this\\s", ngrams$threegram)
+                 & grepl("DT0\\s(AJ0\\s)?N", ngrams$c5_threegram),]
```

The subset is, then, defined via two `grepl()` clauses defining two patterns that *both* need to be matched; see the ampersand (`&`) conjoning the two clauses. The first says, "Match all three-grams that start with the literal match 'this' followed by white space (`^this\\s`)", the second says, "Match all those determiner tags followed by white space (`DT0\\s`) followed by an optional adjective plus white space (`(AJ0\\s)?`) followed by a noun (`N`). Was the subset successful? Let's sample 10 rows from the subset:

```
> this[sample(1:nrow(this), 10),]
                        title   position     twogram c5_twogram
17014                    Snow 0.10909091 this morning DT0 NN1
44426   Nearly in an accident 0.39004149 this young   DT0 AJ0
28786       Doing the fencing 0.17073171 this week    DT0 NN1
25879 Waching machine advert 0.99115044 this week    DT0 NN1
42107               Moving in 0.01145038 this morning DT0 NN1
45058              Poor bloke 0.21138211 this guy     DT0 NN1
15322         Fire at Banbury 0.50608273 this seat    DT0 NN1
49750         Raymond is sick 0.58823529 this morning DT0 NN1
8200                    Conga 0.44811321 this pool    DT0 NN1
3062                Microwave 0.03488372 this morning DT0 NN1

               threegram c5_threegram
17014  this morning and  DT0 NN1 CJC
44426    this young lad  DT0 AJ0 NN1
28786       this week i  DT0 NN1 PNP
25879   this week silly  DT0 NN1 AJ0
42107   this morning no  DT0 NN1 ITJ
45058      this guy was  DT0 NN1 VBD
15322     this seat and  DT0 NN1 CJC
49750 this morning some  DT0 NN1 DT0
8200        this pool it  DT0 NN1 PNP
3062   this morning for  DT0 NN1 PRP
```

There seem to be no duplicate hits and all instances of 'this' are followed either directly by a noun or by an intervening adjective plus a noun. The next step in the analysis is the semantic categorization of the nouns that 'this' determines. For this task, we need to have a complete list of the nouns. At present, all we have are the n-grams containing them. So how to extract them from there? Again by using regex and again by using our function `extract` (cf. Chapter 2):

```
> extract <- function(x) unlist(regmatches(x, gregexpr(pattern, x,
perl = T)))
```

Now let's define the patterns to match. The first pattern is for the nouns following 'this' immediately. They are found in the `twogram` column, so we will target the pattern to that column. The pattern to match in that column is this:

```
> pattern <- "(?<=this\\s)\\w+"
```

You may notice that this pattern uses positive lookbehind; it says, "Match the word-like string (`\\w+`) if you see the literal pattern 'this' followed by white space on the left (`(? <= this\\s)`). We apply the pattern to `extract` and store the result in a vector called `R1` (as we are looking for the item occurring to the right in position 1). However, we need to place the constraint that `extract` work only on those rows in the subset where the value in `c5_twogram` matches the pattern `"DT0 N"`, otherwise we'd end up also including, for example, the adjectives prior to the nouns.

```
> R1 <- extract(this$twogram[grepl("DT0 N", this$c5_twogram)])
```

If we inspect the first few items in `R1`, the list looks good:

```
> R1
[1] "stuff" "woman" "stew" "morning" "sort" "quarter" "dorothy"
[8] "wool" "cotton" "morning" "kid" "bloke" "octopus" "sea"
```

Now for the nouns in R2 position in `threegram`. They have in common that they are the string *at the end* of each element. The notation in regex for end position is the dollar sign `$`. So we can formulate the pattern thus, again using positive lookbehind to match any word (`\\w+`) occurring after white space (`(?<=\\s)`) and at the very end of the element (`$`):

```
> pattern <- "(?<=\\s)\\w+$"
```

Before we pass the pattern to `extract`, we need to be aware that the pattern matches any element-final word. What we want to extract is much more restricted, namely just those element-final words that follow an adjective. To satisfy this condition we apply the pattern to a subset of `this` – namely those rows that have the value `"DT0 AJ0"` in the `c5_twogram` column :

```
> R2 <- extract(this$threegram[this$c5_twogram=="DT0 AJ0"])
> R2
[1] "factory" "bath" "farmer" "guy" "rope" "girl" "swing" "shape"
    "bloke"
[10] "lad" "machine" "girl" "pancakes" "thing" "lad" "lad" "way"
     "dream"
```

The output of R2 indeed contains only nouns. We now have the nouns collocating with 'this' in two separate vectors, R1 and R2. We simply combine them in a single vector using the concatenation function `c()`:

```
> R1R2 <- c(R1, R2)
```

The new vector `R1R2` contains all the nouns 'adnominal this' premodifies in the NC. At this point in the analysis, a manual analysis follows: we need to assign the collocating nouns to the five semantic groups we have devised. The assigment decisions cannot be made by R, but must be taken by the researcher. To make this task a little easier (by avoiding to classify the same items multiple times), we strip the vector of all duplicates by using the `unique()` function, which returns the unique forms:

```
> unique(R1R2)
[1]  "stuff" "woman"  "stew"  "morning" "sort"    "quarter" "dorothy"
[8]  "wool"  "cotton" "kid"   "bloke"   "octopus" "sea"     "area"
[15] "guy"   "pool"   "thing" "company" "car"     "chap"    "time"
(clipped output)
```

Upon careful consideration we formulate four patterns that will allow us to divide the nouns into four different semantic domains (the fifth domain will be for items that do not match any of the four patterns):

```
pattern_person  <-  "chappie|dorothy|farmer|fellow|intruder|jack|la
ura|mrs|paul|richard|shrewsbury|man|woman|chap|kid|lad|bloke|guy|
girl"
```

```
pattern_time <- "party|period|summer|afternoon|time|week|morning"
```

```
pattern_place <- "area|bath|chalet|company|factory|farm|house|pool|
sea|place"
```

```
pattern_object <- "bird|bus|cat|chair|cotton|dictionary|game|machine|
octopus|pancakes|paper|passport|rope|seat|stew|swing|wool|letter|
train|car|thing"
```

Using these patterns and nested `ifelse()` statements we can assign each instance of this+(AJ0)?+N to one of these categories in a new column, which we will call `semcat` (for semantic categories):

```
> this$semcat <- ifelse(grepl(pattern_person, this$twogram)
+                      | grepl(pattern_person, this$threegram),
+                        "this+N_person",
+                 ifelse(grepl(pattern_time, this$twogram)
+                          | grepl(pattern_time, this$threegram),
+                                "this+N_time",
+                   ifelse(grepl(pattern_place, this$twogram)
+                          | grepl(pattern_place, this$threegram),
+                                    "this+N_place",
+                       ifelse(grepl(pattern_object, this$twogram)
+                              | grepl(pattern_object,
                                      this$threegram),
+                                        "this+N_object",
+                                        "this+N_other"))))
```

As we are looking for matches in `twogram` *or* `threegram` (see the alternative marker `|`), we find matches even where one of the two variables doesn't provide a match; see, for example the last line of the output of `head()`, where the value in `twogram` fails to match but the value in `threegram` does match:

```
> head(this[,c(3,5,7)])
          twogram             threegram        semcat
1599    this stuff        this stuff and  this+N_other
1953    this woman        this woman had this+N_person
2169     this stew         this stew you this+N_object
3062  this morning      this morning for   this+N_time
3322     this sort          this sort of  this+N_other
4125 this fucking this fucking factory  this+N_place
```

Before we continue let's check what variable structure R has assigned the new variable `semcat`:

```
> str(this[,7])
chr [1:152] "this+N_other" "this+N_person" "this+N_object" "this+N_
            time" "this+N_other" …
```

`semcat` is a character variable. That's not the format we want: for the stripchart we need the semantic classes as *factors*. So we convert `semcat` using `as.factor()`:

```
> this$semcat <- as.factor(this$semcat)
```

Now we can plot the stripchart. As always, we begin by setting the plotting layout with `par()`:

```
> par(mfrow = c(1,1), mar = c(4,3,2.5,1))
```

We need to be clear what we want to visualize: if it were just the positions of 'adnominal this' whatever the semantic class, the data to input into the function `stripchart()` would be just `this$position`. But our goal is to visualize the positional distribution by semantic class. Therefore we choose as data input the formula notation `this$position ~ this$semcat`, that is, the numeric variable `this$position` grouped by the factor variable `this$semcat`. Moreover, we opt for `method = "jitter"`, which adds a little 'noise' to each positional value and thus helps to avoid overplotting of coincident values (an alternative is `method = "stacked"`, which aligns coincident values vertically; the default is `method = "overplot"`). There is also the *argument*`jitter`, which gives the *amount* of 'noise' to be applied; it defaults to 0.1. For this plot we increase it to`jitter = 0.2`. We also want to distinguish the classes visually, both by point character and color. We select `pch = c(1, 5, 8, 18, 13)` as the five point chacracters for the five classes. In terms of color, we're not committed to any specific combination so we sample five colors from the built-in set of `colors()`, thus: `col = sample(colors(), 5)`. The rest is as usual: the plot title, the label for the x-axis, the font sizes for title and axis, and the suppression of the default frame:

```
> stripchart(this$position ~ this$semcat,
+            method = "jitter",
+            jitter = 0.2,
+            pch = c(1, 5, 8, 18, 13),
+            col = sample(colors(), 5),
+            main = "Adnominal 'this' in storytelling",
+            xlab = "Position",
+            cex.main = 0.9,
+            cex.axis = 0.8,
+            frame = F)
```

To see more clearly how far the positional values stray away from the central unmarked position we plot the median position of the original dataset `ngrams` into the plot by using `abline()` and setting the argument `v` (for vertical) to `median(ngrams$position)`:

```
> abline(v = median(ngrams$position), lty = 3, lwd = 2)
```

Inserting the medians for each semantic category in `this` is one of the more challenging parts of the plot. First we need to access the five levels of the `semcat` variable using the function `levels()`, and store them in a vector, say, `cats`:

```
> cats <- levels(this$semcat)
> cats
[1] "this+N_object" "this+N_other" "this+N_person" "this+N_place"
    "this+N_time"
```

Now we define a `for` loop for each of the levels in `cats`. But `cats` is non-numeric, so simply saying `for(i in cats)` will produce an error; what we can do is grab `cats` by its subscripts, which are numerical, using `seq()`:

```
> seq(cats)
[1] 1 2 3 4 5
```

That we can input into the `for` loop:`for(i in seq(cats))`. What command is R to execute for each level in `cats`? We open curly brackets `{ }` to phrase the command. We begin by instructing R to draw vertical lines to represent the medians using the `segments()` function. We define the lines' starting points on the x-axis, `x0`, as the median position values of the five semantic categories; to iterate over the distinct categories, the argument `x0 = median(this$position[this$semcat==cats[i]])` tells R to treat as `x0` the median `position` where the `semcat` value equals the `cats` value (*this* pairing works as both `semcat` and `cats` are non-numerical vectors). Also, we define the starting points for the y-axis, `y0`, as well as the end points, `y1`, as the index `i` subtracting and, respectively, adding a little distance between them, say 0.3, and set the graphical parameters `col` (color), `lty` (line type), and `lwd` (line width):

```
> for(i in seq(cats)){
+   segments(x0 = median(this$position[this$semcat==cats[i]]),
+            y0 = i - 0.3,
+            y1 = i + 0.3,
+            col = "red", lty = 3, lwd = 2)
+ }
```

We attach a grid to the stripchart, using all its defaults:

```
> grid()
```

Finally, to check whether the iteration has proceeded in the right order and produced the right results, we can compute the median values by using `tapply()`:

```
> medians <- tapply(this$position, this$semcat, FUN = median)
> medians
this+N_object this+N_other this+N_person this+N_place this+N_time
    0.3855727    0.3575189     0.2156863    0.2785436   0.1866197
```

Indeed, the median lines the `for` loop has inserted in the stripchart are assigned to the right semantic class and sit at the right places on the x-axis.

The violin plot in Figure 8.2 is coded as follows. Luckily, we need not streamline our raw data `ngrams` in any other way than we did for the stripchart but can re-use our dataframe `this` as data input and start assembling the plot code almost immediately. Just one preparatory step is needed: we will have to store the median positions in a dataframe; so we re-define `medians`, which is a list, as a dataframe and call it `mediansdf`:

```
> mediansdf <- data.frame(semcat = names(medians), position = medians)
```

`ggplot2` is a non-base R package that needs to be installed:

```
> install.packages("ggplot2")
> library(ggplot2)
```

In ggplot2, it is customary to build up a plot in increments, to connect the increments with +, and to store them in a plot *object*, say, p. Then, before specifying the plot type you have in mind, you first lay the foundations for the plot by naming the dataset, here this, and, using the function aes() to define the x and y variables, here semcat and position, as well as determining by which variable the colors are to be filled in, here (obviously) the factor variable semcat. Since we are by no means finished coding the graph but want to add more, we must not forget to put a plus sign at the end of this preliminary chunk of code:

```
> p <-
+ ggplot(data = this,
+ aes(x = semcat, y = position, fill = semcat)) +
```

Next we give the plot a title, using the function labs():

```
+ labs(title = 'Adnominal this in storytelling') +
```

At this point R does not yet know which graph type we want to produce; we could veer off into all sorts of directions. The function that specifies the violin plot is geom_violin(). As other ggplot2 graphs, geom_violin() offers a vast array of functions and parameters; check out ?geom_violin at the console. We use scale, trim, and adjust, as well as scale_fill_manual, which is used outside the call to geom_violin. The argument scale is set to scale = "count" as we wish to scale the violin areas proportionally to the number of observations. The logical argument trim determines whether or not the tails of the violins are cut to the range of the data; as the cut is made by default, we do not *have to* set the argument to trim = TRUE but do it anyway to make the reader aware of this visually important argument (note that in the violin plot in Figure 8.5 the default is overridden). The related arguments adjust and kernel are critical too: they determine the degree of smoothing of the density line and the kernel function underlying the smoothing. The argument adjust defaults to 1; so values smaller than 1 will reduce smoothing (leading to more humps), values greater than 1 will increase it (leading to fewer humps). The argument kernel defaults to "gaussian" but we choose the "cosine" function, purely for aesthetic reasons. The function scale_fill_brewer() can be added if you wish to override the default color settings. This function offers a large number of palettes (cf. ?scale_fill_brewer); here, we select "Pastel2" (plot colors can also manually be selected with scale_fill_manual):

```
+ geom_violin(scale = "count",
+             trim = T,
+              adjust = 0.6, kernel = "cosine") +
+ scale_fill_brewer(palette = "Pastel2") +
```

The beauty of `ggplot2` is how seamlessly it allows you to combine different plotting (or, in `ggplot2` speak, geometric) objects in a single graph. Here, for example, we wish to also plot the equivalent of a stripchart, namely the distribution of the observed positions of 'adnominal this'. To do so we add the function `geom_point()` to the object `p`. Using the argument `position` allows us to make adjustments to where and how the the points will be plotted; here we use the function `position_jitter()` and its argument `width` to jitter the points, so that ties (i.e., coincidental values) become distinct points. Moreover, we determine the size of the points in the argument `size` and their transparency factor in `alpha`:

```
+ geom_point(aes(y = position),
+           position = position_jitter(width = .15), size = 1.1, alpha
            = 0.6)
```

Next we add another geometric object, namely a straight line to divide the early positions from the late positions. To locate the line on the y-axis we set `yintercept = 0.5` and we determine its `color`, `size`, and `linetype`:

```
+ geom_hline(yintercept = 0.5, color = "blue", size = 0.8, linetype
= 4) +
```

There is yet another important geometric element we want to insert: the line segments for the medians. To this end we use the function `facet_grid()`, which allows us to create 'facets', that is, panels for each subgroup. The obvious variable on which to split the position data into groups is `semcat`, our five semantic classes. The syntax for `facet_grid()` is fixed: the first element is the variable on which to split, the second is the tilde, and the third is a period. The additional argument `scales = "free"` ensures that the scales vary across both rows and columns:

```
+ facet_grid(semcat ~ ., scales = "free") +
```

Now we can use the function `geom_segment()` to inscribe the medians into the panels. Note that the data to plot in this case are no longer the data in dataframe `this`; the data are the medians stored in dataframe `mediansdf`. So we assign this data, `mediansdf`, to `geom_segment()`. Then we define the 'aesthetics', that is, how we want the segments to be drawn: we determine a start point in `x = 0.5`, an end point in `xend = 1.5`, and, most important, the location of the segment on the y-axis (that is, the actual median value stored in column `position` in `mediansdf`) in `yend = position`. Further, we set `color`, `linetype`, and `size` (that is, width) of the line segments:

```
+ geom_segment(data = mediansdf,
+              aes(x = 0.5, xend = 1.5, yend = position),
+              color = "red", linetype = 1, size = 0.8) +
```

By default, `ggplot2` plots contain legends, called 'guides'. While these are obviously useful in other cases, they would be redundant in our case as the information they convey – the names of the `semcat` levels – is already conveyed in the facet labels. We therefore disable `guides`:

```
+ guides(fill = F, color = F) +
```

However, the axis labels and the axis ticks for the `semcat` levels on the y-axis still remain visible in the plot. We don't need them either, as the `semcat` names are already provided by the facet labels. Parameters related to axis specifications are controled by the `theme()` function; to disable `theme()` parameters, we use `element_blank()`:

```
+theme(axis.text.y=element_blank(),axis.ticks.y=element_blank())
```

The violin plot could be considered complete at this point: we would have the violins representing the semantic categories lined up on the x-axis and the positions of 'adnominal this' shown on the y-axis. But there is yet another tweak to be implemented: the variable `positions` (in the dataframe `this`) reflects a temporal order – the occurrence of 'adnominal this' early or late in stories – and timelines are normally represented extending from left to right. To reflect that temporal order, we will want to position the violins horizontally rather than vertically. To achieve this, we have to rotate the coordinates of the whole plot, thus turning the x-axis into the y-axis and vice versa. The code for the rotation could not be simpler:

```
+ coord_flip()
```

Now the violins are on the y-axis and the positions on the x-axis, and the object `p`, which contains the code for the violin plot, is finally complete – which is why we do *not* put a plus sign behind `coord_flip()`. We call `p` to actually produce the plot:

```
> p
```

8.3 Task: Storytelling structure – The role of time expressions in Orientation

Time is an essential coordinate of the told situation – in order to form a representation of the told situation story recipients are eager to get to know *when* the events of a story happened: did it happen just now or this morning or was it yesterday or a week or a year or even longer ago? Knowing this matters, as the relevance of a story may vary with the recency of the events. We would assume that storytellers will make it their business to let the recipients know early on in the Orientation section. An illustrative example is (8.5), where the incipient storyteller tries hard to locate the precise day she detected her broken bra strap during P E:

(8.5)

Emm:	but° (.) i tell you what (.) it w's so embarrassin'
	the, it was the **Mo-** , the **Monday after the Saturday** got,
	the **Saturday** it was done, I had p e (0.6) o:h great
Hel:	uh hh
Emm:	so there's me like (0.8)
	i'm SAt in first lesson
	we have (0.3) p e last lesson (0.6)
	sat in the first lesson (0.8)
	'nd i'm th- I've thought (1.1)
	°↑go:d my bra strap feels really ↑lo:se!°
	(0.8) so I'm right (0.4)
	I thought oh my go::d!
	There's a ga:p in betwee:n,
	[they're not atta:ched any mo:re!
Hel:	[hh hu hh hh hh hh]

((story continues))

In this storytelling, then, the time coordinate is established early on, as one of the earliest elements to set the scene (the only element preceding the time is the storyteller's stance toward the events "it w's so embarrassin'"). But is this generally so? Do storytellers generally insert time references story-initially? The case study in Section 8.1 did point in this direction, showing that time references realized through 'adnominal this' were skewed toward the beginning of stories. But 'adnominal this' covers a just a small part of the rich pool of resources speakers can choose from to refer to time. This pool also includes time adverbs (e.g., 'today', 'yesterday', 'tomorrow'), time nouns (e.g., 'year', 'autumn'), calender nouns (e.g., 'Monday', 'April') and the ways these nouns can be premodified, not only by 'adnominal this', but also adjectives such as 'next', 'last', etc. thus achoring the time reference deictically to the time the utterance is made (cf. Rühlemann 2018a: Chapter 3). This Task section offers the opportunity to probe into that much wider range of resources to configure time and, possibly, temporal orientation for the story recipient.

The analysis is based on a large dataset pulled from the Narrative Corpus (NC; cf. Rühlemann & O'Donnell 2012). The NC has three components: turn-by-turn conversation preceding the stories, the storytellings themselves, and turn-by-turn conversation succeeding the stories. The three components are termed CPR (for pre-narrative component), CNN (for narrative component), and CPO (for post-narrative component). The narrative component CNN and the two non-narrative components taken together are of equal size, roughly 75,000 words. In this Task, we will make use of this three-fold structure by comparing the positions of time references across the three components. What we would expect to find is that time references are roughly equally distributed positionally in the non-narrative components but skewed to the left – that is, to early positions – in the storytelling component.

How can time references be retrieved from the NC? A major chunk can be extracted from the corpus by the PoS tag NP0: this tag captures proper names, including not only person names and place names but also the names of the days of the week and the names of the months. As for other references to time, most of them deictic in nature, such as 'last year' or 'today', tokens can be retrieved through literal searches. The time-deictic expressions included in the dataset are the following:

day, night, morning, afternoon, evening, week, month, year, today, tonight, yesterday, tomorrow

All word tokens either tagged as NP0 or matching the above literal strings were retrieved from the three components of the NC, totaling roughly 3,500 tokens. Crucially, the position in each story was calculated for each token using the formula discussed in Section 8.1.2. The data is available on the companion website as "Chapter8_Task.txt". Here's a random selection of ten rows from the dataset, called here `np0`:

```
> np0[sample(1:nrow(np0), 10),]
      component                  title    position  time_ref
3255        cnn             Ali Caver 0.008917197   morning
2729        cpo            Fire alarm 0.777777778   Kilburn
3360        cnn          Rugby scrum 0.806818182       day
2724        cpo        Russian winter 0.674311927  Thursday
1849        cpr      Thelma and Louise 0.000000000   Malcolm
2055        cpo Men don't have babies 0.793478261      Luke
3296        cnn         Darryl's phone 0.911764706 yesterday
550         cnn          Gassy custard 0.073170732    Friday
1188        cnn          Cheap buckets 0.194690265     Marks
3213        cnn          Black T-shirt 0.184000000 yesterday
```

We have the `component` variable with its three levels `cnn`, `cpr`, and `cpo`, an additional variable `title`, the central variable `position`, and finally the `time_ref` variable containing the word tokens. However, tokens such as Kilburn, Luke, or Marks need to be discarded as they are names for people and places (as noted, the tag NP0 is used for all kinds of proper names). So the first step (after uploading the data, checking its structure, and – if necessary – converting the structure of variables) is to subset the data on those tokens in the `time_ref` column that are indeed time expressions.

The subset can be obtained in various ways. One easy way is to define patterns, for example thus:

```
> days <- c("Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sunday")
> months <- c("January|February|March|April|May|June|July|August|S
eptember|October|November|December")
> deictics <- c("^day|night|morning|afternoon|evening|week|month|y
ear|today|tonight|tomorrow|yesterday")
```

To define the sought subset, say `time`, containing only the rows with the time expressions, you can use the three patterns, marked as alternatives by `|`, with `grepl()`:

```
> time <- np[grepl(days, np$time_ref)|grepl(months, np$time_
ref)|grepl(deictics, np$time_ref), ]
```

The subset you obtain should feature only tokens in the `time_ref` column that can be used as (part of) time references; let's check:

```
> time[sample(1:nrow(time),10),]

     component                  title    position  time_ref
3074       cnn                   Dark 0.006644518   morning
3547       cpo                    Row 0.094339623     today
2496       cpo        Darrel's coming 0.994897959    Sunday
3180       cnn     A corn on every toe 0.406474820       day
3067       cnn Roaring out the factory 0.144578313     night
521        cnn              Two calls 0.093167702    Friday
3402       cnn            Sundays off 0.254143646     night
3157       cnn    Richard's helicopter 0.890243902 afternoon
2886       cpo                   Gala 0.902985075     night
2947       cpr           Lazy teacher 0.497005988      year
```

Now you are ready to produce the stripchart and the violin plot with the median positions inscribed into them. The code for the plots is analogous to the code explained in Section 8.2. Depending on the parameters you have selected, your stripchart and violin plot should be like the graphs in Figure 8.4.

Both figures show that time references in storytelling (i.e., in the CNN component) heavily cluster toward the very beginning of the positional spectrum exhibiting a clear skew to the left, as indicated by the median line at 0.3. By contrast, no skew is discernible for time references in turn-by-turn talk (that is, in the CPR and CPO components): their medians hover closely around the 0.5 demarcation, which indicates no skew at all. The skewed distribution in storytelling is additionally shown by the density estimations, which show a big hump in early position for CNN which gradually flattens out toward later positions; no such development is

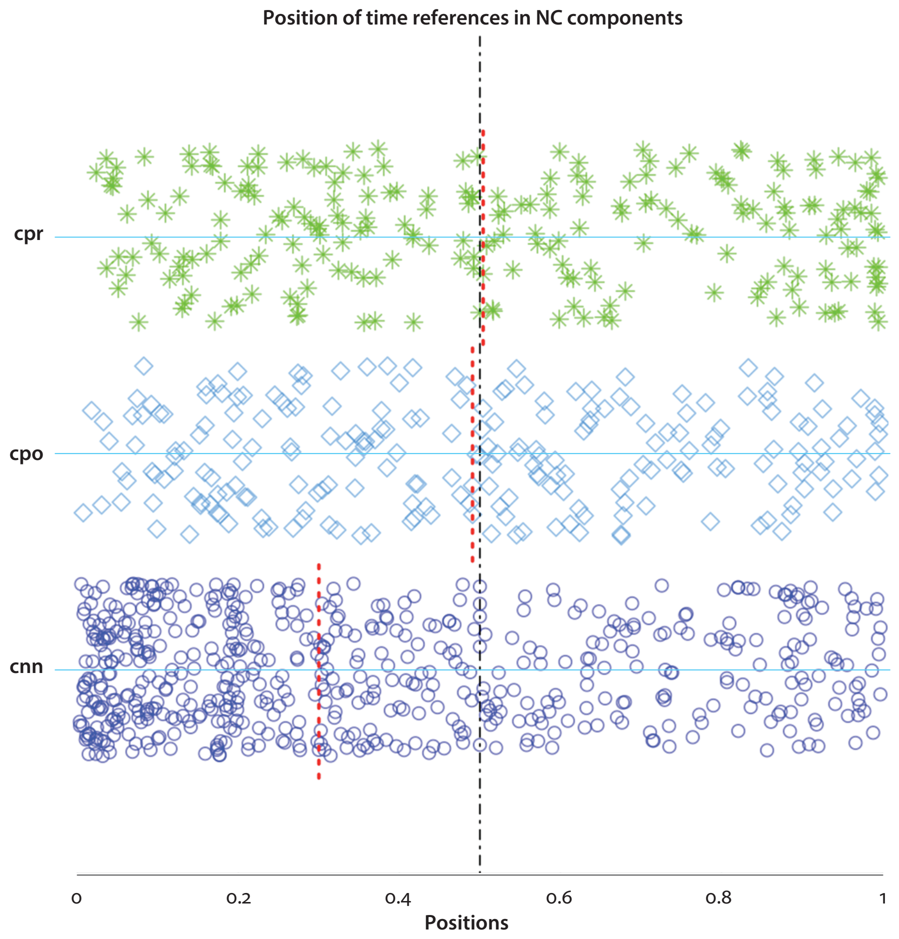


Figure 8.4 Stripchart of positions of time references in storytelling (CNN) and turn-by-turn talk (CPR and CPO)

discernible for the non-narrative components: the accidental bumps and valleys do not show a skewed distribution.

The stripchart and the violin plot then confirm our assumption that storytellers make it their business to provide the temporal coordinate, a crucial component of the told situation, at the beginning of stories, to enable story recipients to form a semantic ground against which to evaluate the events figure.

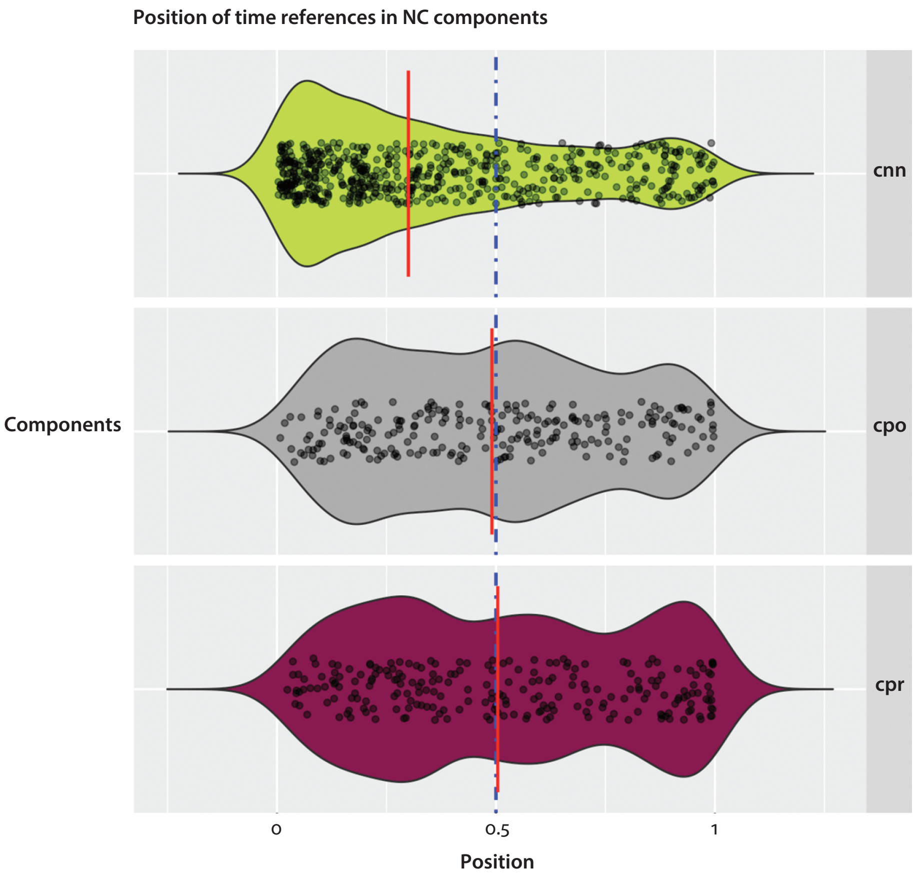


Figure 8.5 Violin plot of positions of time references in storytelling (CNN) and turn-by-turn talk (CPR and CPO); the argument `trim = F` ensured that violin tails were not cut off

Chapter 9

Scatter plots

9.1 *Case study*: Storytelling – The role of intensity and pitch in advance-projecting the Climax

9.1.1 Introduction

Storytelling is a key activity in conversation. There are at least three defining features to it. Storytelling constitutes an extended sequence as "stories take more than an utterance to produce" (Sacks 1992: 223) and they are "built from many turn-constructional units" (Goodwin & Heritage 1990: 299). Further, storytelling progresses in a structured way: "a story is not, in principle, a block of talk" (Jefferson 1978: 245); rather, stories are composed of "larger structures of talk" (Goodwin 1984: 241) referred to as story 'segments' (Jefferson 1978) or 'components' (Goodwin 1984). The components acknowledged in CA include Preface, Background, and Climax as well as Post-completion sequences; in Discourse Analysis, the most widely used structural model is Labov's model (1972); it provides for (the optional) Abstract, Orientation, Complication, Resolution, Evaluation, and (the optional) Coda. Finally, storytelling is 'powered' by stance (or affect): it can be conceptualized as "an activity that both takes a stance toward what is being reported and makes the taking of a stance by the recipient relevant" (Stivers 2008: 32). While storyteller can make their stance available throughout the whole storytelling process, often already in the Preface, the stance display by the recipient is relevant late in the storytelling, at or around the Climax. The stance taken by the recipient(s) preferably "*mirrors the stance* that the teller conveys having (often in the story preface) whether that is funny, sad, fabulous, or strange" (Stivers 2008: 33; added emphasis). So, in a nutshell, storytelling is not (only) about sharing events but (more) about affiliating and "sharing the emotional load" (Peräkylä et al. 2015: 301).

On this understanding the Climax is critical not only because it is the most 'tellable' event (Sacks 1992), but, more importantly, because it is that point at which the recipients' affiliation is expected and, thus, the point of "emotional contagion" (Hatfield et al. 1994).

If the inherent aim in storytelling is, then, to bring about this contagion and "share the emotional burden" (Peräkylä et al. 2015: 301) at or around the Climax, storytellers should deploy great care in making sure story recipients won't miss that point. In other words: storytellers will likely project its occurrence in advance or use signposts to mark the Climax upon its occurrence (for example, through constructed dialog). Some research suggests that storytellers subtly guide their hearers to the Climax using an orchestrated *crescendo* of vocal and bodily resources. While the bodily resources deployed for advance-projecting the Climax include co-speech gesturing (e.g., Holt 2007) and, in multi-party storytelling, accelerated gaze alternation (Rühlemann et al. 2019), we will be concerned here with *vocal* resources for Climax advance-projection.

In this case study, the aim is to investigate what I will call the crescendo hypothesis. This hypothesis predicts that vocal resources increase in synchrony with the storyteller's progression from early story segments to the Climax as the key segment in storytelling interaction. Specifically, the case study will investigate two vocal resources, intensity ('loudness') and pitch, and their relation with the Climax.[37]

9.1.2 Data and methods

Unlike in most other case studies in this book, the data used here will be a single case: intensity and pitch will be examined in a single storytelling. Needless to say that the results of this single-case study cannot, at this point, be generalized.

The storytelling under examination is a story from the Narrative Corpus, a corpus of storytellings occurring in natural conversation, extracted from the BNC, stored and re-annotated in the Narrative Corpus (NC; Rühlemann & O'Donnell 2012). The story, titled "Drained canal", is 47 seconds long.

Based on the story's availability in the audio files released by the Audio BNC, the story as presented here was re-transcribed using CA conventions. The story is told by Barry to Alan. The two men are long-time friends and passionate about fishing. At the point in the conversation where the story occurs, they are in the middle of sharing stories about bad luck at fishing. The story "Drained canal" continues this theme:

```
(9.1)
1    Alan:    Well it's, it's (.) luck innit [(I don' know),]
2    Barry:                                  [I remember    ] once go:n' on,
3             I got- (0.4) we got up 'bou' three three thirty in the morning
4             ( ) went out to er (0.9) canal somewhere up
5             (1.3)
6             Dulga' area past Dulgate
7             (1.3)
8             we set up
9             and we'd we'd been fishing for about two and half hours
```

37. For a detailed description of the acoustic concepts of intensity and pitch, see Boersma (2013).

10 it's aba- about six thirty in the morning
11 this old farmer comes up
12 says er (1.1) ↑Aye aye lads,
13 he said er (0.7) I wou' n' bother it
14 → **they > drained this area of the canal a few months aG(h)↑O < Hhh::::,**
15 [hh::: GGAeehh::: he] he he
16 Allan: [huh huh huh huh huh .]
17 Barry: S(h)*at* there watching our floats for hours *uh*heh heh:
18 I mean *luck*ily you- you know, you'd gone on- with a car
19 so it's a ma'er o' throw'n ev'ryth'n in th' back
20 ['n' j's go:n'] s'm'ere else sort of (ay) (1.5)
21 Alan: [° ye:: °]
22 Barry: could've sat there all bleed'n' day!
23 (1.00)
24 °°an' not known anythin' about it°°.
25 (4.4)
26 Alan: aye

['Drained canal', BNC: KBD 1790-1801]

The storytelling, arguably, reaches its Climax in line 14: "they > drained this area of the canal a few months aG(h)↑O < ". One important indicator that this is the Climax is the use of constructed dialog. Other indicators include accelerated tempo (expressed through > and <), increased volume (shown through capitalization), and rising pitch (indicated by ↑). Finally, the shared laughter between Alan and Barry in lines 15–16 indexes the stance convergence that storytelling is designed to achieve at the Climax.

The availability in audio format allowed both auditory and acoustic analysis in Praat (Boersma & Weenik 2012). Using a script specifically written for this case study (Weenik, p.c.) intensity, in decibel, and pitch, in Hertz, were measured and synchronized at 0.01 second intervals. Thus, three variables were extracted: time, intensity, and pitch. Further, based on auditory analysis, the Climax was identified in the Praat spectrogram as shown in Figure 9.1 (see the blue-shaded area). A new variable was created, `Segment`. Under that variable, the value "Climax" was entered at the time interval consumed by the Climax (shown in red numbers in the spectrogram); all other cells were set to NA.

9.1.3 Results

The scatter plot in Figure 9.2 visualizes the results of the acoustic and auditory analyses. The figure has two y-axes, one for intensity (on the left), another for pitch (on the right), each with a scale of its own. The green rectangle marks the extent of the Climax on the x-axis, which records time. The figure also shows locally-weighted regression lines, so-called smoothers (cf. Cleveland 1979). The advantage of the smoother is that it does not attempt to impose a single model on all of the data in the sample but that it

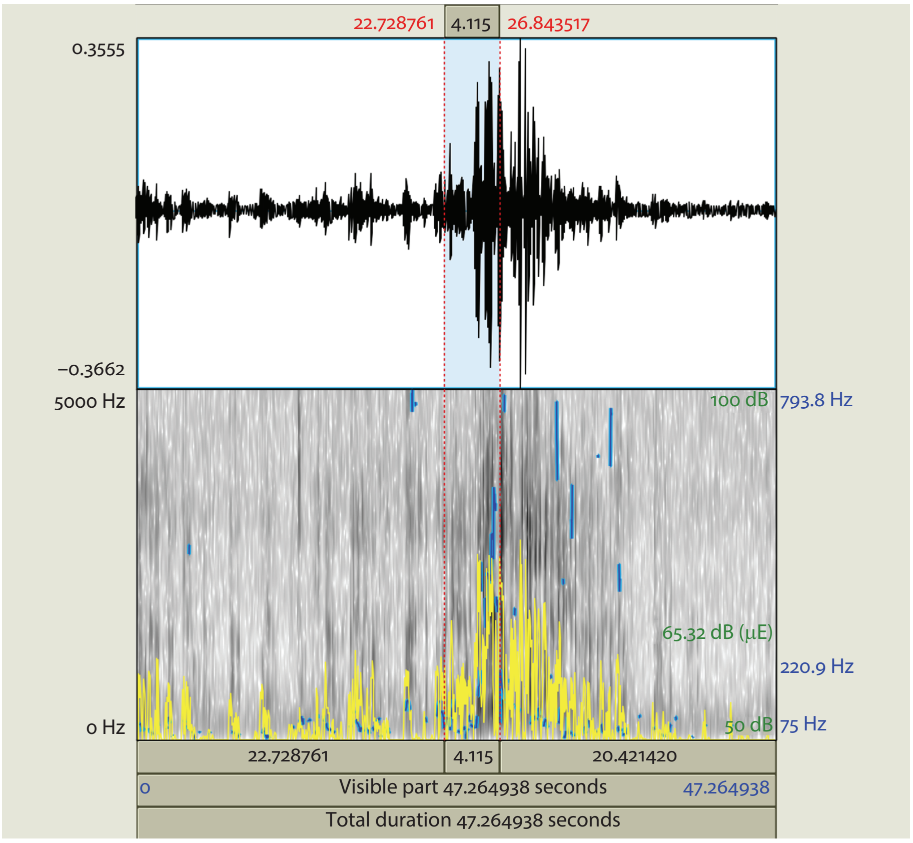


Figure 9.1 Praat spectrogram of storytelling "Drained canal"; the blue-shaded area in the upper panel delineates the Climax

models small local segments of the data. Smoothers are especially useful for detecting and comparing trends in noisy variables.

We can see from Figure 9.2 that pitch and intensity pretty much work in lockstep with one another throughout the whole storytelling: both measures see a slight drop right before the Climax, both reach their highpoint when the telling reaches its highpoint, and both fall back to low levels in post-Climax positions. The three variables, then, are fully synchronized.

9.1.4 Discussion and concluding remarks

Is the synchrony in "Drained canal" sufficient evidence to support the crescendo hypothesis, according to which vocal resources increase in sync with the storyteller's progression from early story segments to the Climax? Of course not. Before we can confidently claim that the multimodal correlation found in this single clase reflects a

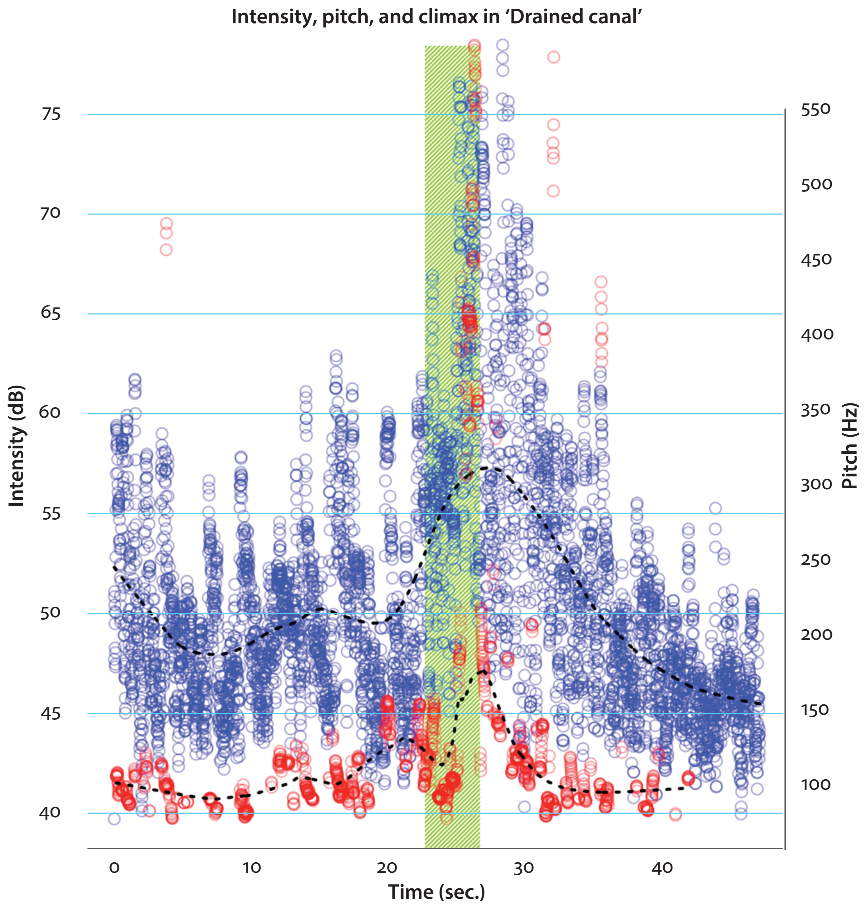


Figure 9.2 Intensity, pitch, and climax in "Drained canal"; the dashed lines are smoothers

multimodal design pattern underlying stories in general we have to find the same correlation in a large collection of stories. This large-scale analysis represents an exciting avenue for future research.

What little we can do in this book is perform the same acoustic and auditory analysis for another randomly selected storytelling from the BNC. This will be undertaken in the Task section below.

Also, the single-case study by no means allows us to know whether the paralinguistic crescendo observed is *used by the storyteller* to advance-project the Climax. Advance-projection is a design feature actively deployed by speakers for an interactional purpose, namely to guide the hearer toward a point at which some action by them becomes relevant, be it clapping in political speeches (cf. Atkinson 1984), taking the floor (cf. the case study in Section 11.1), or, as in the present case, affiliating with the teller's stance in storytelling (cf. Stivers 2008).

There is at least one alternative interpretation[38] (other than the interpretation as mere coincidence): the crescendo in pitch and intensity toward the Climax could index the teller's increasing *arousal*, that is, the intensifying activation of the autonomic nervous system associated with emotion (Peräkylä et al. 2015: 302). Emotions do not only involve changes in physiological responses such as heart rate, blood flow, and changes in skin conductance (e.g. Peräkylä et al. 2015) but these psychophysiological changes, in turn, also impact the speaker's voice, modulating it "as a function of a person's emotional state" (Liebenthal et al. 2016: 6). This line of thought provides an plausible explanation not only for the rise in pre-Climax pitch and intensity but also for the subsequent drop in paralingustic activity in post-Climax position. Research at the interface of CA and psychobiology discovered that displays of affiliation by recipients have the effect of reducing the storyteller's arousal: "affiliation calms down the storyteller" (Peräkylä et al. 2015: 302). In "Drained canal", the recipient's laughter at the Climax is a clear affiliation with the storyteller's stance toward the events. In other words: the changes we observe in pitch and intensity over the course of the storytelling could well be indicative of the storyteller's psychophysiological processes: increased arousal in reliving the story events and the emotions associated with them (in pre-Climax position) and decreased arousal in accomplishing the recipient's display of affiliation (in post-Climax position).

Intriguingly, the story analyzed in the Task section below will provide (preliminary) evidence to support this line of reasoning precisely because pitch and intensity do *not* peak at Climax.

9.2 The scatter plot in the case study

The scatter plot discussed in the case study in Section 9.1 is relatively straightforward to produce as most variables already come in a form that requires only a little tweaking before they can be fed into the plot. The data is available on the companion website as "Chapter9_Casestudy.txt"; the code is stored as file "Chapter9_Casestudy_Code.R". We will call the data `ip` (for intensity and pitch):

```
> ip <- read.table("[Your path]/Chapter9_Case study.txt", header =
T, quote = "",sep = "\t", fill = T)
```

The tweaks concerns the way the Praat script used to extract both intensity and pitch at equi-distant time intervals handles missing values:

38. Note that the two interpretations are by no means mutually exclusive.

```
> ip[1:10,]
   Time_s      Pitch_Hz  Intensity_dB  Segment
1    0.00 --undefined-- --undefined--       NA
2    0.01 --undefined-- --undefined--       NA
3     0.0 --undefined-- --undefined--       NA
4    0.03 --undefined-- --undefined--       NA
5    0.04 --undefined-- --undefined--       NA
6    0.05 --undefined--          39.7       NA
7    0.06 --undefined--         47.69       NA
8    0.07 --undefined--         50.82       NA
9    0.08 --undefined--         51.08       NA
10   0.09 --undefined--         50.41       NA
```

The Praat script assigns to missing values the label `"--undefined--"`. This is significant as R reads these variables as factors, as shown by calling `str()`:

```
> str(ip)
'data.frame'   : 4727 obs. of 4 variables:
$ Time_s       : num 0 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 …
$ Pitch_Hz     : Factor w/ 666 levels "--undefined--",..: 1 1 1 1 1
                 1 1 1 1 1 …
$ Intensity_dB: Factor w/ 1905 levels "--undefined--",..: 1 1 1 1 1
                 2 540 867 804 …
$ Segment      : logi NA NA NA NA NA NA …
```

That is, the `"--undefined--"` values need to be replaced by `NA` and the variables `Pitch_Hz` and `Intensity_dB` need to be converted to numeric. Also, `Segment` is treated as logical; we'd rather have it as factor. We set any cell in the dataframe that contains the unwanted value to `NA`:

```
> ip[ip == "--undefined--"] <- NA
```

Now we convert `Segment` to character and `Pitch_Hz` and `Intensity_dB` to numeric; note that conversion from factor to numeric requires a detour via conversion to character; so we convert columns #2 and #3 twice:

```
> ip[,2:4] <- lapply(ip[,2:4], as.character)
> ip[,2:3] <- lapply(ip[,2:3], as.numeric)
```

Now the variables have the right formats:

```
> str(ip)
'data.frame'   : 4718 obs. of 4 variables:
$ Time_s       : num 0.05 0.06 0.07 0.08 0.09 0.1 0.11 0.12 0.13 0.14 …
$ Pitch_Hz     : num NA NA NA NA NA NA NA NA NA NA …
$ Intensity_dB: num 39.7 47.7 50.8 51.1 50.4 …
$ Segment      : chr NA NA NA NA …
```

Finally, inspection reveals there are a small number of rows where both `Pitch_Hz` and `Intensity_dB` are `NA`:

```
> ip[(is.na(ip$Pitch_Hz) & is.na(ip$Intensity_dB)) ,]
      Time_s  Pitch_Hz  Intensity_dB  Segment
1       0.00        NA            NA     <NA>
2       0.01        NA            NA     <NA>
3       0.02        NA            NA     <NA>
4       0.03        NA            NA     <NA>
5       0.04        NA            NA     <NA>
4724   47.23        NA            NA     <NA>
4725   47.24        NA            NA     <NA>
4726   47.25        NA            NA     <NA>
4727   47.26        NA            NA     <NA>
```

These are not useful for the analysis. We remove them by subsetting `ip` for those rows in which both `Pitch_Hz` *and* (`&`) `Intensity_dB` are *not* (`!`) `NA` (the function `is.na()` tests for `NA` and returns `FALSE` and `TRUE` for each cell):

```
> ip <- ip[!(is.na(ip$Pitch_Hz) & is.na(ip$Intensity_dB)) ,]
```

Now, the clean-up is complete. Before we can configure the plot, though, we need to complete the variable `Segment`: the values of interest are of course the time interval covered by the Climax. We know from the auditory analysis based on the Praat spectrogram (cf. Figure 9.1) when the Climax starts and when it ends, namely 22.73 and, respectively, 26.83 seconds into the storytelling. So we set those cells of `Segment` that lie in this interval on the `Time_s` variable to `"Climax"`:

```
> ip$Segment[ip$Time_s >= 22.73 & ip$Time_s <= 26.83] <- "Climax"
```

Now we can start building the scatter plot, or, as I should say, the scatter plots. This is because the graph shown in Figure 9.2 is really two plots in one: one for intensity against time and one for pitch against time. Merging the two plots in one and providing separate y-axes for each of them will require some attention.

We start by setting the basic plotting parameters like plot layout and margins:

```
> par(mfrow=c(1,1), mar = c(4.1, 4, 2, 4))
```

At this point we can code the two y-axes. We do it by defining a sequence that starts from 0 and leads up to the maximum value in each variable and is divided by certain intervals; axis ticks will be placed at these points. To identify the maximum value we use the function `max()`. Note that `max()` cannot handle `NA`; to make it immune to them, we include the argument `na.rm = T`:

```
> ticks_y1 <- seq(from = 0, to = max(ip$Intensity_dB, na.rm = T),
by = 5) >
ticks_y2 <- seq(from = 0, to = max(ip$Pitch_Hz, na.rm = T), by = 50)
```

The plot for intensity against time is set up with the usual parameters such as *y* and *x* variables, the title, a label for the x-axis (which is obviously the same in either plot). We also determine that the y-axis be suppressed with `yaxt = "n"` and that no data be plotted with `type = "n"`.

```
> plot(ip$Intensity_dB ~ ip$Time_s,
+      main = "Intensity, pitch, and climax in 'Drained canal'",
       cex.main = 0.9,
+      xlab = "Time (sec.)",
+      ylab = "",
+      yaxt = "n",
+      frame = F,
+      type = "n")
```

Next we draw the primary y-axis: the argument `2` identifies the axis to be drawn as the second axis moving clockwise (from the x-axis); as input to the `at` argument we select our previously defined vector `ticks_y1`; we take blue color for the axis ticks (`col.ticks`) and axis labels (`col.axis`) and reduce the label size slightly (`cex.axis`):

```
> axis(2, at = ticks_y1, col.ticks = "blue", col.axis = "blue", cex.
       axis = 0.9)
```

To label the y-axis we use mtext():

```
> mtext("Intensity (dB)", side = 2, line = 2.5, col = "blue")
```

Why did we choose `type = "n"` in calling `plot()`? The reason is the Climax rectangle. If we printed the intensity values first, the rectangle would come to overplot them. We want the rectangle to remain in the background, so we will print it first. The function for rectangles is `rect()`. The input it requires are the two points on the x-axis and the two points on the y-axis, in that order: `xleft`, `ybottom`, `xright`, `ytop`. The point `xleft` is the minimum value on that subset of `ip$Time_s` that has the value `"Climax"` in the `Segment` column whereas `xright` is the opposite: the maximum value on the same `ip$Time_s` subset. The `ybottom` point is simply `0` and `ytop` is the maximum value of the variable depicted on the y-axis, that is, `ip$Intensity_dB`:

```
> rect(xleft = min(ip$Time_s[ip$Segment=="Climax"], na.rm = T),
+       ybottom = 0,
+       xright = max(ip$Time_s[ip$Segment=="Climax"], na.rm = T),
+       ytop = max(ip$Intensity_dB, na.rm = T),
+      density = 50, col = "lawngreen", angle = 45, border = "white")
```

In addition, `rect()` allows for a number of graphical parameters, from which we choose (i) `density`, the argument which controls the density of shading lines, measured in lines per inch, (ii) `angle`, the argument for the angle of the shading lines, (iii) `border`, for the color of the outline around the rectangle, and (iv) `col` for the color of the shading lines.

Now, and only now, will we print the values for intensity against time, by using `lines()`; this function requires the same data input as `plot()`, so we repeat `ip$Intensity_dB ~ ip$Time_s`. Further, we choose the type of plot we want, namely `type = "p"` for points (alternative values are, for example, `"l"` for lines, `"b"` for points and lines, `"h"` for histogram-like lines, etc.). We also select the point character for the points, with `pch = 1` (the default). As we have thousands of values, there will inevitably be a massive amount of overplotting, resulting in reduced legibility of the distribution of data points. To reduce the negative effect, we use the function `rgb()` (for 'red', 'green', 'blue'); it allows color variation both in terms of intensity and transparency; the argument `maxColorValue` gives the maximum of the values range.

For example, in the code below, `rgb(0, 0, 255, 100, maxColorValue = 255)` sets red and green to `0` but blue to `255` (the maximal value), as well as transparency to `100` (a value in the lower half of the possible range):

```
> lines(ip$Intensity_dB ~ ip$Time_s,
+       type = "p",
+       pch = 1,
+       col = rgb(0, 0, 250, 100, maxColorValue = 255))
```

The only thing missing from the first plot is the smoother line; to insert it into the plot we use the function `lowess()` inside the function `lines()` and set parameters such as color, `col`, linetype, `lty`, and line width, `lwd`:

```
> lines(lowess(ip$Intensity_dB ~ ip$Time_s, f = .2), col = "black",
lty = 3, lwd = 2)
```

Now for the second plot, for pitch against time. The 'trick' to print a second plot into an existing plot is `par(new = T)`: this will allow a new plot to be added to an existing one:

```
> par(new = T)
```

To plot the values we use mostly the same code as for the first plot; the difference is, obviously, the input data `ip$Pitch_Hz ~ ip$Time_s` and also `type = "p"`, that is, the instruction to actually make the values visible (rather than keep them invisible, as in the first plot). Another obvious difference is that here we use red but not green or blue:

```
> plot(ip$Pitch_Hz ~ ip$Time_s,
+      xlab = "",
+      ylab = "",
+      yaxt = "n",
+      type = "p",
+      frame = F,
+      col = rgb(255, 0, 0, 100, maxColorValue=255))
```

Also the procedure for the secondary y-axis is pretty much the same as for the primary y-axis, the main difference being that we select `4` (instead of `2`) and set the axis colors to red. The `side` argument is also key in configuring the input for the `mtext()` function to print the axis header:

```
> axis(4, at = ticks_y2, col.ticks = "red", col.axis = "red", cex.
       axis = 0.9)
> mtext("Pitch (Hz)", side = 4, line = 2.5, col = "red")
```

Finally, we have arrived at the last element, the smoother for pitch against time. Again the code is essentially the same as before; for sake of consistency, we will have to use the same amount of smoothing by setting `f` to the same value as we did in drawing the smoother for intensity against time:

```
> lines(lowess(ip$Pitch_Hz ~ ip$Time_s, f = 0.2), col = "black",
             lty = 3, lwd = 2)
```

And, voilà, the graphic is complete.

9.3 Task: Intensity, pitch and climax in "Non-smoking area"

In this Task section, the aim is to perform the same type of analysis that was conducted in the case study on another storytelling from the BNC, a 39.5 second story titled "Non-smoking area".

The telling's transcript is shown in (9.2). Sue, who is a shop assistant in a department store, is relating an incident with a customer who came in with a cigarette despite the no-smoking signs. Sue reveals her stance toward the events in the Preface: she was annoyed by the customer's 'cockyness' (line 2), which "really go' [her] back up¿" (line 4). The preface is followed by Background information relating to the customer's age (lines 6–7) and the first few verbal exchanges between him and Sue. The story arguably reaches its Climax in line 15 with Sue reporting her internal dialog "And I felt like saying fuck off and get out!". This is a likely candidate utterance for the story highpoint as the annoyance that can be inferred from the constructed dialog matches the annoyance with which the story was billed in the Preface. The preferred response at this point for the recipient would be the display of a stance that mirrors the stance by the storyteller. That display, however, is not proffered audibly. Instead there is a pause of 0.8 seconds. In pursuit of that display Sue starts a post-musing sequence (Schegloff 2007). Part of that pursuit is the hyperbolic simile of "going to a firework factory and (.) sparking up a cigarette" in line 17, followed by her assessment "you just wouldn't do i'," and her pleading question tag "would ya." in line 18. As there is still no audible response, let alone an agreement with her assessment, she provides the agreement herself in "°You wouldn't¿°" in line 20.

(9.2)
1 Sue: Did I tell you about that bloke the other week who got,
2 the one who go' cocky with me?
3 ↑Ooh↓ go' =
4 = he really got my back up¿
5 (0.6)
6 This young lad about (0.8) probably about twenty three, twenty four,
7 and he came in with a cigarette
8 and I said excu- (0.4) excuse me but I'm afraid it's a no-smoking area.
9 And he looked at me
10 and he said well have you got any",
11 he go-, started like tutting and said "well have you got any sig:ns up?
12 And I must have been in a really funny mood,
13 and I said °yes we have, it's on the till°
14 and he went (.) °well I suggest you get them in the windows then¿°
15 And I felt like saying fuck off and get out!
16 (0.8)
16 but I ↑thought (.) they're obvious, you know what I mean?
17 It's like (.) going to a firework factory and (.) sparking up a cigarette
18 you just wouldn't do i', would ya.
19 (1.8)
20 °You wouldn't¿°

(BNC: KC6 2010-2019; corrected transcription)

How do intensity and pitch intertwine with that story's structure?

The intensity, pitch, and segment data are available on the companion website under "Chapter9_Task.txt". To code the scatter plot, you can essentailly follow the coding underlying the plot in the case study:

– Read-in the file.
– Prepare the data by (i) converting "--undefined--" and empty cells to `NA`, (ii) cutting off rows where intensity *and* pitch are not available, and (iii) converting factor to numeric.

- Set up the basic plotting parameters with `par()`.
- Define the ticks for the primary y-axis and the secondary x-axis.
- Start plotting the first set of data, namely intensity as a function of time (making sure nothing is actually plotted with `type = "n"`).
- Plot the x-axis ticks and the x-axis label.
- Compute the coordinates for the Climax rectangle.
- Insert the rectangle into the plot.
- Now use `lines()` to actually plot intensity.
- Add the smoother with `lowess()`; experiment with the argument `f`: remember the larger `f`, the larger the amount of smoothing; also, choose a line type and line width that allow the smoother to become clearly visible.
- Now plot the second set of data, pitch as a function of time.
- Print the secondary y-axis and add its label with `mtext()`.
- Finally, add the second smoother.

Depending on the colors you have chosen and the degree of smoothing, your scatter plot will roughly look like the one in Figure 9.3.

One can easily see that intensity, pitch, and Climax do not match the way they did in the case study in Section 9.1: while the two acoustic measures develop largely in lockstep across the whole storytelling (except for the early part), they peak, at the same time, *after* the Climax, with pitch peaking even more strongly than intensity.

The reason for this postponement of the peak is, arguably, the interactional issue arising from the story recipient's reluctance to respond as expected by the storyteller. That is, noticing that the recipient fails to proffer the sought affiliation with the teller's stance of the events as annoying, the teller steps up her effort at pulling off that affiliation in the Post-sequence musing (cf. Couper-Kuhlen 2012). That increased effort is not only manifest verbally, in her hyperbolic comparison of the department store with a fireworks factory, but also, and perhaps more importantly, vocally, in her elevated pitch and the increase in loudness. The increase in post-Climax paralinguistic activity may also be a physiological reflection of the absent affiliation by the recipient. As Peräkylä et al. (2015: 302) have shown, "increased recipient affiliation is associated with a decrease in the physiological arousal in the storyteller: affiliation calms down the storyteller." As the affiliation is still pending at and after Climax, the calm-down has not yet taken effect; on the contrary: the teller is further aroused, this time not due to reliving the story events and the emotions associated with them but due to the imminent failure to bring about the recipient's stance convergence. (Note that this interpretation is at this point fully speculative; to substantiate it, pitch and intensity measures would have to be triangulated with psychophysiological data, such as skin conductance or heart rate.)

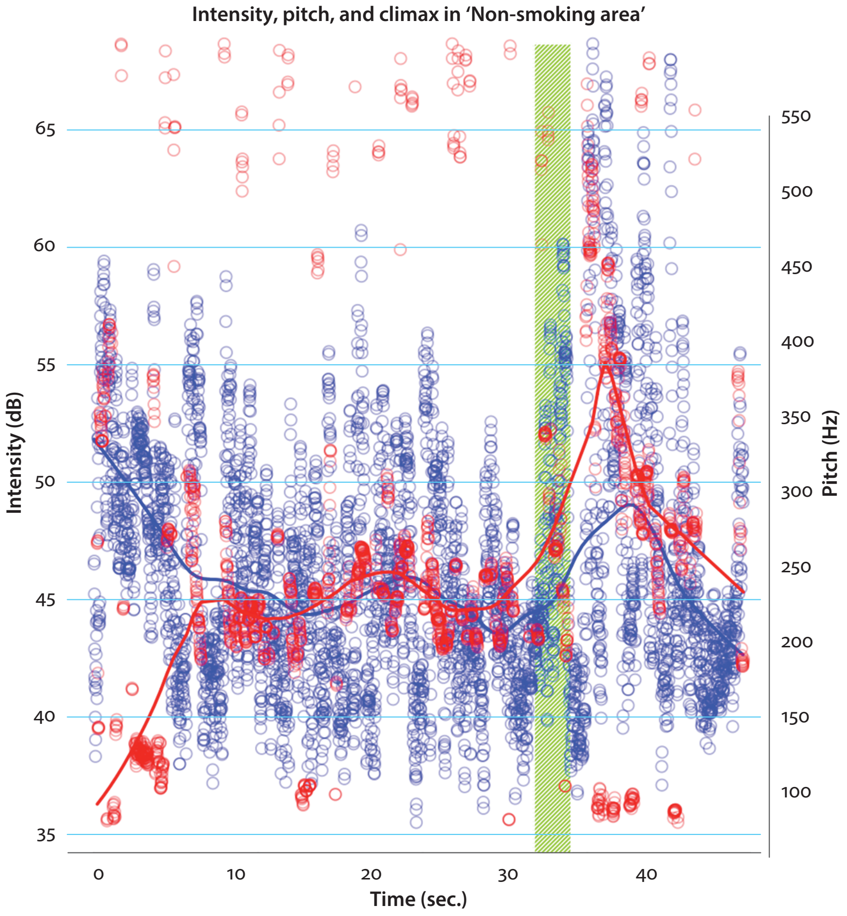


Figure 9.3 Intensity, pitch, and Climax in "Non-smoking area"; lines represent smoothers

So the two vocal resources intensity and pitch are used differently with regard to storytelling structure in this storytelling than in the story analyzed in the case study. It is, then, clearly far too early to make sweeping claims. However, what the two analyses do suggest is the possibility that pitch and intensity are *sensitive* to story structure. Exactly how they respond to internal divisions of storytellings needs yet to be examined in larger and more diverse data sets. Another intriguing avenue for future research is the question how, and to what extent, paralinguistic resources such as pitch and intensity are correlated not only with storytelling structure but with arousal.

Chapter 10

Association plots and mosaic plots

10.1 *Case study*: Paralinguistic prosody in constructed dialog in storytelling

10.1.1 Introduction

Among the complexities of conversation is what could be called 'stacked conversation' – conversation about conversation, a practice referred to as speech reporting (e.g., Koester & Handiford 2018) or discourse presentation (e.g., McIntyre et al. 2004). To converse about conversation, conversationalits have a large repertoire of forms available to them – McIntyre et al. (2004) discuss eight different types. In conversation, they predominantly choose one major form, constructed dialog.

Constructed dialog manifests in two forms, (i) as direct speech, where the discourse presented is introduced by a reporting clause, that is, a noun or pronoun and a verb of saying (e.g., 'she said', 'he goes', etc.), and as (ii) free direct speech, where the discourse is presented without any reporting clause; this latter type is also referred to as 'zero quotative' (Mathis & Yule 1994) and 'reenactment' (e.g., Sidnell 2006). The two subtypes of constructed dialog are illustrated in extract (10.1); there is one instance of free direct speech in line 7, indicated by " > > ", all other instances of constructed dialog are introduced by a reporting clause, and are hence direct speech, indicated by " > ":

(10.1)

1	Gail:	Arf was saying for, erm to Nat °the other night°
2>		he said **"↑oh look I got a baby inside me Nat."**
3>		She said **"no you can't (have) babies, ladies can."**
4		(0.9)
5	UNK:	°mm°
6>	Gail:	**"d'you know how they get there?"**
7>>		**"Yeah but I ain't telling you!"**
8	UNK:	I kno:w↑

It has traditionally been assumed that constructed dialog is a verbatim rendition of anterior discourse: the reporting speaker "commits himself to faithfully rendering form and content of what the original speaker said" (Coulmas 1985, quoted in Wade & Clark 1993: 806). This assumption has more recently been abandoned as a fallacy and replaced by Clark & Gerrig's (1990) demonstration theory; according to this theory quoters *demonstrate* an utterance (if there has been an 'original' one in the first place) by making *selections* from it and manipulating it on a large range of parameters such as pitch, range, quality, gestures, emotional state, etc. Why is this demonstration theory more plausible? The reasons are numerous (for a more detailed account see Rühlemann, forthcoming a). First, memory experiments found that "people generally cannot recall an utterance verbatim unless it was the last utterance heard" (Wade & Clark 1993: 805). Second, faithfulness presupposes that there was an original utterance to be faithful to; often, however, speakers "'quote' utterances that have never happened, but are projected as hypothetical in an imaginary world or as possible in a future situation" (Koester & Handiford 2018: 67). Third, and perhaps most importantly, conversationalists deploy constructed dialog for purposes altogether different than faithfulness.

In storytelling, constructed dialog is a primary means by which storytellers express their stance toward events (Stivers 2008, 2013) typically at or around the story climax (e.g., Drew 1998; Mayes 1990; Holt 2000; Stivers 2008). In 'animating' the voices of protagonists (Goffman 1981), storytellers inject into the constructed dialog subtle cues for the story recipients as to how to *evaluate* that utterance. These cues draw on the whole multimodal spectrum of expressive resources, including words (which may never have occurred), co-speech gestures, and vocal resources. Research by Holt (2007), Blackwell et al. (2015), Stec et al. (2016), and Soulaimani (2018) have shown high rates of non-verbal co-speech activity both in the gestural and the vocal modalities in the use of constructed dialog. Speaking of vocal resources, Wennerstrom observes that they are deployed to "achieve an infinite variety of emotional, attitudinal, and stylistic effects" (Wennerstrom 2001: 200).

In this case study, the aim is to examine whether these vocal resources are deployed in constructed dialog to a larger extent than they are deployed in speech outside constructed dialog. Let's call this the 'animation hypothesis': it predicts that constructed dialog is richer in paralinguistic prosody than non-constructed dialog.

10.1.2 Data and methods

The case study is based on a subsample of 50 randomly selected storytellings extracted from the *Narrative Corpus* (Rühlemann & O'Donnell 2012), a corpus of conversational storytellings.[39] Based on the audio recordings made available through the Audio BNC (Coleman 2012), these storytellings were re-transcribed using CA transcription conventions (e.g., Jefferson 2004, Hepburn & Bolden 2013), thereby correcting erroneous renditions in the orthographic transcripts and enriching the transcripts by adding paralinguistic detail of delivery (changes in intensity, pitch, stress, etc.) (for details

39. The NC contains roughly 150,000 words and 531 storytellings embedded in their conversational surroundings. For details on the construction and annotation of the corpus see Rühlemann & O'Donnell (2012).

on this re-transcription process, cf. Rühlemann & Gee 2017). The re-transcriptions were made by researchers unaware of the research question underlying this case study. Instances of constructed dialog were marked in the transcripts by use of quote marks, as shown in excerpt (10.2):

(10.2)
01 (UNK1): ( ) the other (.) the other day
02 we was on about nicola said about
03 oh she goes like that to luke. =
04 = i said "**oh you can't do that**
05 **otherwise he won't be able to have babies.**"
06 (0.5) so nicola said (0.2)
07 "**[but he] doesn't have babies** =
08 (UNK2): [hello:.]
09 (UNK1): =**women have babies.**"
10 i said "**w'll (0.3) it comes from a ↑man.**"
11 (0.6)
12 (UNK1): so (0.5) luke said = "**w' what are they like.**" =
13 = so i said "**well they're like little tadpoles.**"
14 and he went_ (0.7)
15 "**well↓ i ↓can't feel↑[any?]**"
16 (UNK2): [uhu hu hu] hu hu hu hu h
17 (UNK1): £a::nd I cou(h)ldn't help laugh'n£. =
18 =.h an' y' know what he said? (0.4)
19 luke turned round=he said (0.5)
20 "**GHO::°d° (.) mummy's ↑laughin.**"

In CA transcription there are standardized conventions for interactionally relevant details of paralinguistic phonology. These conventions are given in Table 10.1.

As noted, the aim in this case study is to find out whether constructed dialog is more animated than non-constructed dialog. To test it, we proceed as follows. First, two subsamples were formed from the 50-story sample: one for all speech outside constructed dialog, and one for all speech occurring inside constructed dialog. We will call the former subsample 'nonquoted speech' and the latter subsample 'quoted speech'. For illustration, reconsider the last four lines from extract (10.1), given here as (10.3):

(10.3)
17 (UNK1): £a::nd I cou(h)ldn't help laugh'n£. =
18 = .han' y' know what he said? (0.4)
19 luke turned round = he said (0.5)
20 "**GHO::°d° (.) mummy's ↑laughin.**"

The speech in lines 17–19 goes into the subsample for nonquoted speech whereas the speech in line 20 "GHO::°d° (.) mummy's ↑laughin." is assigned to the subsample for quoted speech.

Table 10.1 CA conventions for transcribing paralinguistic phonology

Sub-category	CA symbol	Description
intonation:	**?**	question(-like) rise
	¿ or **?,**	weakly rising intonation
	.	falling intonation
	,	continued intonation
	_	level intonation
	!	animated tone, not necessarily an exclamation
pitch change:	↑	sharp rise in pitch
	↑↓	sharp risefall in pitch
	↓ or |	sharp fall in pitch
volume:	**A** or bold formatting	loud voice
	°a	soft voice
stretching:	**a::**	lengthened sound
stress:	***a*** or ***a*** or ***A*** or bold formatting	stressed or heavily stressed or very heavily stressed sound
truncation:	**-**	cut-off in mid-word
aspiration:	**.h** or **h.**	inhalation or exhalation
	hh	extent of aspiration
smile voice:	**£**	talk produced while smiling
creaky voice:	*****	words pronounced with a creak
tremulous voice:	**~**	tremulous speech

Then, in a second step, in each subsample the number of words transcribed with CA symbols encoding paralinguistic prosody was determined in the subsamples. To illustrate, in the nonquoted subsample for (10.3) there are two tokens of £, indicating the start and end of smiley voice. In the subsample for quoted speech there are two word tokens with CA symbols for paralinguistic prosody: *GHO::°d°* (capitalization for loudness, : for lengthening, and ° for low intensity) and ↑*laughin'* (↑ for sudden pitch rise and underlining for emphasis). As this example illustrates, one and the same token can become the object of prosodic variation on multiple levels. However, to ensure independence of the observations, each word was counted just once no matter the number of paralinguistic resources used on it.[40]

40. It may well be argued that the 'amount' of paralinguistic prosody on a word will interactionally make a difference: as a reviewer notes, "↑THI:S" (3 points) is different in degree from "this" (1 point). Taking this cumulative dimension on board has not been possible here (but see Dingemanse & Akita 2017, where it has been) as the aim in the chapter is to demonstrate the use

To extract paralinguistic CA symbols from the transcripts, regex was deployed (cf. Section 2.6). To make the transcripts fully amenable to regex extraction the following changes were made to them. First, response tokens occurring in interjacent overlap with constructed dialog were manually wrapped into curly brackets so they could be matched and removed. Further, regex can match CA *symbols* used to encode paralinguistic delivery; what it cannot match are formattings, such as underlines and italics (both used in CA transcripts for degrees of emphasis). To also match instances of underlines and/or italics, these instances were marked by € (for emphasis) in the transcripts.

To examine the animation hypothesis, a chi-squared test was performed. This test examines the association of categorical variables, commonly arranged in a 2 × 2 contingency table (larger than 2 × 2 tables are possible too). The test is based on the null hypothesis that the variables are independent. That is, according to the null hypothesis it does not matter in which rows and columns the observations are grouped as they are not associated with their categories (cf. Levshina 2015: 210). The test computes *expected* frequencies based on the observed frequencies and the assumption of the variables' mutual independence and compares them to the observed frequencies (cf. Crawley 2007: 301 ff.). The graphics used for visualizing the test results are the association plot and the mosaic plot. These plots show *residuals*, that is, "the differences between the observed and expected frequencies divided by the squared root of the expected value" (Levshina 2015: 218).

10.1.3 Results

The animation hypothesis holds that constructed dialog is richer in paralinguistic prosody than non-constructed dialog. That is, it claims that the use of paralinguistic resources in speech depends on whether it is quoted speech or nonquoted speech. To test this claim using a chi-squared test, what we need are exactly four counts: the number of words in quoted speech (i) with and (ii) without paralinguistic codings, and the number of words in nonquoted speech (iii) with and (iv) without paralinguistic codings. The numbers discovered in the analysis are given in Table 10.2.

Table 10.2 2 × 2 contingency table for number of words with and without paralinguistic variation in quoted and nonquoted speech in storytelling

	Quoted speech	Nonquoted speech
With paralanguage	194	2115
Without paralanguage	317	6353

of association plots and mosaic plots, two visualizations for the chi-squared test. This test requires categorical variables that need to be mutually exclusive.

Based on the data in this table, the test returns a very highly significant result: X-squared = 41.882, df = 1, p-value = 9.696e-11. The residuals, the test's most important diagnostic, are depicted in the association plot in Figure 10.1 and the mosaic plot in Figure 10.2.

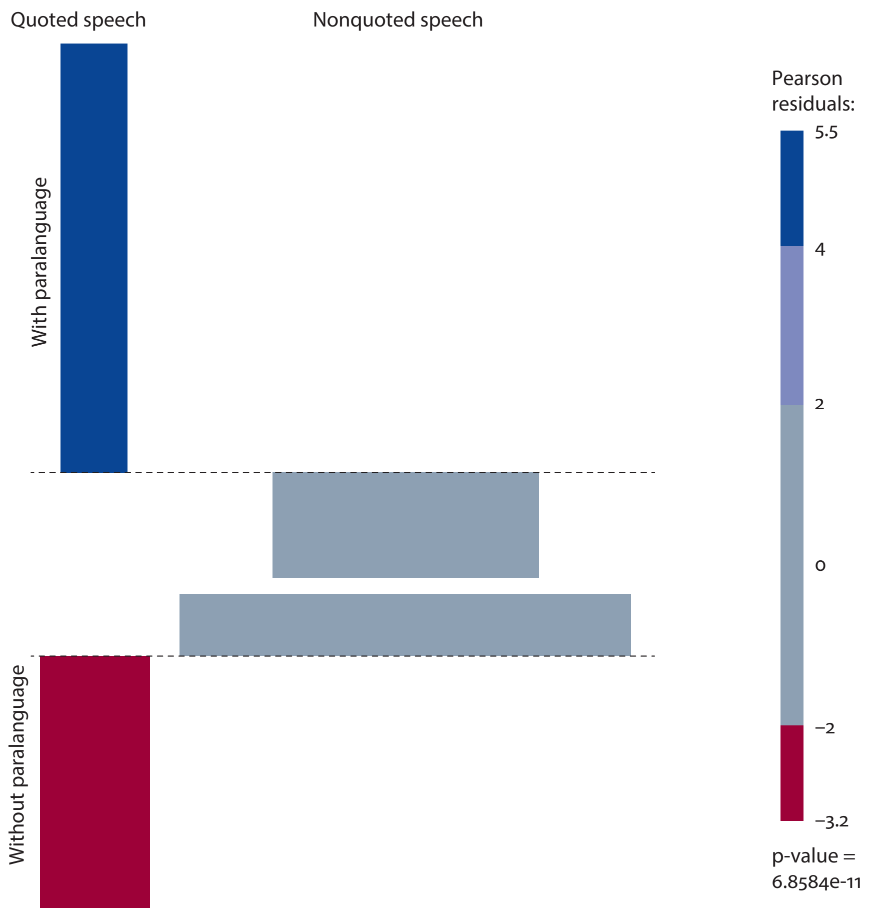


Figure 10.1 Association plot of paralinguistic prosody in the use of quoted and nonquoted speech in storytelling; height of rectangles represents size of residuals (difference between observed and expected frequencies); width of rectangles is proportional to size of counts

The rectangles are shaded depending on the size of the residuals: large positive residuals (> 4) are blue, large negative residuals are red; where the residuals are small, the rectangles are grey. Further, the dotted line (shown only in the association plot) represents a baseline indicating independence (that is, the null hypothesis). Positive residuals are given in rectangles rising, negative residuals falling, from

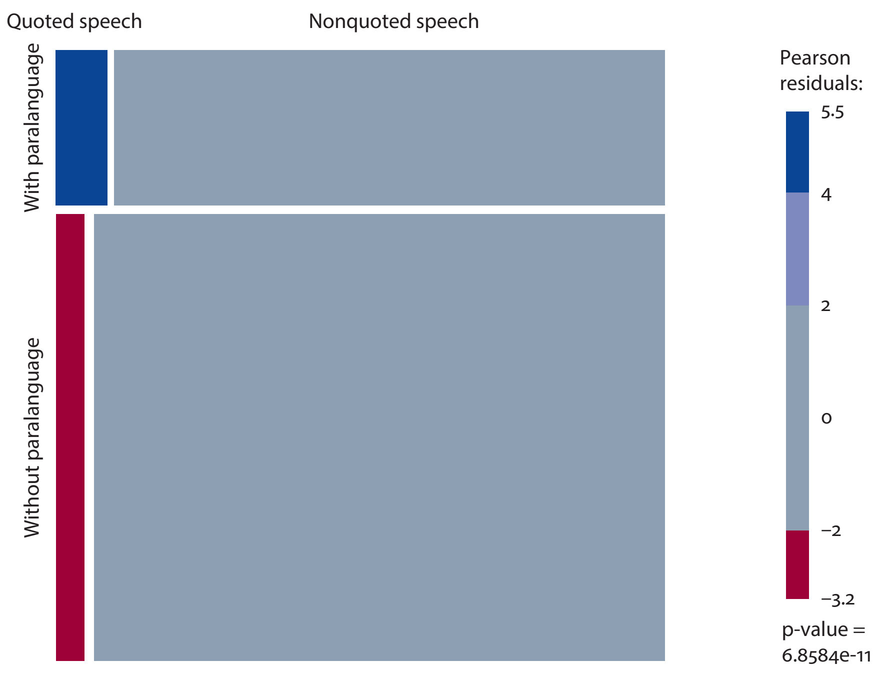


Figure 10.2 Mosaic plot of paralinguistic prosody in the use of quoted and nonquoted speech in storytelling

this baseline. Finally, the width of the rectangles is proportional to the size of the counts.

It can be seen very clearly that the differences between observed and expected frequencies are minimal for nonquoted speech (cf. the grey boxes), whereas the differences for quoted speech are marked: quoted speech contains far more paralinguistic codings than expected and also far fewer words without paralinguistic codings than expected.

The analysis, then, confirms the animation hypothesis: quoted speech is richer in paralinguistic variation than nonquoted speech.[41]

10.1.4 Discussion and concluding remarks

Why do speakers make more use of their vocal resources when they construct dialog than when they produce their own words? Two answers seem in order.

41. The effect size, however, is disappointing: Cramer's V is just 0.069.

One reason for greater paralinguistic variation is the fact that speakers anticipate the listeners' difficulty at distinguishing the speaker's own words (where, in Goffman's [1981] view, the speaker is both animator and author at the same time) and the constructed words (where the speaker is only animator): speakers modulate their voice to flag the beginning of constructed dialog thus aiding the listeners in recognizing what is constructed and what is not constructed. In other words, paralinguistic variation serves as auditory quote marks (cf. Holt 1996; Golato 2000; Bolden 2004; Rühlemann 2013). An example is extract (10.4):

(10.4)
1 Gail: Just sat there with Kate and then suddenly it was erm (1.1)
2 **"↑mum what's ↑hooker?"** (0.8)
3 **"°oh god.°"** ( )

The sudden rise in pitch on which the quote-initial word "mum" is intoned in line 2 sets the word apart from Gail's own words, flagging the word as authored by another, non-present speaker. Similarly, the reduced intensity of "°oh god.°" in line 3 forms a paralinguistic contrast with the preceding words, thus again signalling again signaling another speaker's words.

So, paralinguistic contrasts in the use of constructed dialog provide a service to the listener in aiding them with the 'boundary issue' (Rühlemann 2013) of telling apart non-quoted speech from quoted speech.

A second reason why constructed dialog is a prime site of paralinguistic activity relates to the function of constructed dialog in storytelling. As noted, storytelling essentially serves to bring about the story recipient's contagion with the storyteller's stance toward the events. Storytellers use constructed dialog *strategically* in the service of this goal, that is, with a view to accomplishing the stance contagion by infusing constructed dialog with stance. This becomes clearest in the use of 'mimicry', "a caricatured re-presentation" (Culpeper 2011: 161) or 'echo' of anterior discourse, "reflect[ing] the negative attitude of the echoer towards the echoed person" (Culpeper 2011: 165). In fragment (10.5), for instance, Jackie caricatures her mother's complaining about her absence from home by using mimicking voice quality, elongation, and elevated pitch:

(10.5)
1 Jackie: uhh I better <u>go</u> actually (.)
2 mother will be er complaining¿
3 Tony: yeah okay
4 Jackie: ((mimicks)) **"↑where have you bee:::n, you said half past ↑three::"**

So, paralinguistic activity looms large in constructed dialog because constructed dialog is a carrier of stance: it can "convey both the attitude of the reported speaker and, more implicitly, the attitude of the current speaker" (Holt 2000: 438).

10.2 The association plot and the mosaic plot in the case study

How were the association plot and the mosaic plot programmed?

The two plots were made on a 2 × 2 table containing exactly four counts: the number of (i) quoted words, with paralinguistic variation, (ii) quoted words without paralinguistic variation, (iii) nonquoted words with paralinguistic variation, and (iv) nonquoted words without paralinguistic variation. Just four numbers – that seems doable. But bear in mind that the data we start out from is pure text data, completely non-numeric, and CA transcripts feature not only speech but all sorts of nonspeech too (comments, pauses, etc.). So, to get to the four numbers, we need to prepare ourselves for a rocky road lying ahead.

As always, the first step, and already the first difficulty, is reading-in the data, stored as "Chapter10_Casestudy.txt" on the companion website; the code is stored there as "Chapter10_Code.R":

```
> stories <- read.table("[Your path]/Chapter10_Case study.txt",
+                      header = T, fill = T, quote = "", sep = "\n")
```

Note that the `sep` argument is set to `sep = "\n"` (for new line), rather than `"\t"` as before, as the lines in the story transcripts are not separated by tabs but merely by new lines. Calling `head(stories)`, we get the first six lines of the data:

```
> head(stories)
                                                          Story
1 "Kar:\tMind you our Colin's getting more like your dad every day
2                                             June:\tI €know he is.
3                                   Kar:\tblack welding glasses on,
4                         \tand he turned round and he made me jump
5                                                    \t"O:h, Colin",
6                                                     \tand then ( )
```

This is not yet the format we can readily work with. Rather, we will want to split the lines into two columns: one for the speaker, such as `Kar:` in line 1, and one for the speech they produce. A highly useful package that allows you to perform such splits is `stringr`. We install it and activate it for the current session:

```
> install.packages("stringr")
> library(stringr)
```

The package includes the function `str_extract()`; it takes two arguments, the input vector (in our case `stories$Story`) and the pattern to match. How to define that pattern, that is, where and how to locate the splitting point? As we can see from `head(stories)` shown above, the speaker name is followed by the colon : and a tab stop, whereas the speaker's speech is preceded by these two elements. So we can use positive lookahead and positive lookbehind to formulate these two patterns:

```
"\\w+:(?=\\t)"
```

This pattern can be glossed as the instruction to match any word-like character one or more times followed by a colon if there is a tab stop on the right.

```
"(?<=\\t).+\\w.*"
```

This pattern says, "Match any word-like character preceded by anything zero or more times and followed by anything zero or more times (`.+\\w.*`)if you see a tab stop on the left (`(? <= \\t)`)". Based on these patterns we can define a new dataframe, `newStories`, that contains the desired two columns, `speaker` and `text`:

```
> newStories <- data.frame(
+ speaker = str_extract(stories$Story, "\\w+:(?=\\t)"),
+ text = str_extract(stories$Story, "(?<=\\t).+\\w.*")
+ )
```

A look at the dataframe reveals that we got what we wanted:

```
> head(newStories)
  speaker                                              text
1    Kar:  Mind you our Colin's getting more like your dad every day
2   June:                                            I €know he is.
3    Kar:                               black welding glasses on,
4    <NA>                   and he turned round and he made me jump
5    <NA>                                            "O:h, Colin",
6    <NA>                                              and then ( )
```

At this point some housekeeping is in order. First, calling `str(newStories)` we see that , as is customary, R treats all character data as factors:

```
> str(newStories)
'data.frame': 1546 obs. of 2 variables:
$ speaker: Factor w/ 82 levels "Al:","Alan:",..: 43 42 43 NA NA NA
NA 3 43 NA ...
$ text : Factor w/ 1442 levels "\t\t\t\t\t\t[  yeah  ]",..: 1054 887
690 529 126 629 1439 1422 888 638 ...
```

To convert to character, we use `lapply()` on the whole dataframe:

```
> newStories[] <- lapply(newStories[], as.character)
```

Next, we need to get rid of rows with the above mentioned comment lines contained in double round brackets, identifiable through their value `"NA:"` in the `speaker` column as well as those rows with response tokens occurring within an uninterrupted string of constructed dialog, identifiable through `{…}`. We use subsetting, `grepl()` and the alternative marker `|` to check which and how many rows of the dataframe contain this material:

```
> newStories[grepl("NA:", newStories$speaker)
+            | grepl("\\{.*\\}", newStories$text),]
    speaker                                                  text
29      NA:        ((speakers have been discussing driving tests))
172     NA:                             ((response-story follows))"
173     NA: ((speaker is relating the content of an educational
                                                video about refugees))
184    Aud:                                           {((laughs))}
194    Aud:                                                  {Mm.}
268   Alan:                                          \t\t {[°ye::]}
286     NA: ((speakers have been talking about food they did not
                                                       want to eat))
726     NA: ((speakers have been talking about salt as a means to
                                  get rid of humidity in a caravan))
851   UNK2:                                            {[hello:.]}
```

There are altogether nine such rows. We exclude them from `newStories`using the negator `!`:

```
> newStories <- newStories[!(grepl("NA:", newStories$speaker)
+                           | grepl("\\{.*\\}", newStories$text)),]
```

Then, we collect all text in a single vector uninterrupted by line breaks; this will be the vector to extract quoted speech and nonquoted speech from:

```
> allSpeech <- paste(newStories$text, sep = "", collapse = "")
```

The data in this vector look like this:

```
> allSpeech
[1] "Mind you our Colin's getting more like your dad every day I
€know he is. black welding glasses on, and he turned round and he
made me jump “O:h, Colin”, and then (   ) You know, he'll come run-
ning (   ) glasses. Yeah. I can't do it. And then me heart jumped,
cos he looked just like your dad, and then a few days later, he's in
garage again, and Johnny come in and he said, “God” he said, “(it
did give) me a flaming heart attack out there” I said, “why” and he
said, “he turned round with these glasses on” then he said “it was
just like John's stood there” I says, “don't tell me” I said, “he
did the same thing to me the other day” and it were only within a
couple of days of each other. When he puts those glasses on, honestly
he's his double. (2.6) ((baby babbling)) i- it's jus’ … <truncated>
```

We now have a single string containing all speech by all storytellers in the sample. However, the string also hosts a lot of nonspeech, such as empty round brackets (for unintelligible words), transcriber comments in double round brackets, pauses in single round brackets, overlap delimitors, to name only the most important items. These items may get in the way when we start counting the words in quoted speech and nonquoted speech. So we need to remove them. We do so by using `gsub()` on `allSpeech` and we decide to do it bit by bit (rather than, as would be possible, in a single large pattern):

```
> # remove short pauses:
> allSpeech <- gsub("\\(\\.\\)", "", allSpeech)

> # remove long pauses:
> allSpeech <- gsub("\\(\\d\\.(\\d{1,})\\)", "", allSpeech)

# remove unintelligible speech:
> allSpeech <- gsub("\\(\\s{1,}\\)", "", allSpeech)

> # remove transcriber comments:
> allSpeech <- gsub("\\({2}[^(\\){2})]*\\){2}", "", allSpeech)

# replace multiple spaces by a single space:
> allSpeech <- gsub("\\s{2,}", " ", allSpeech)

> # remove overlap delimitors [ and ]:
> allSpeech <- gsub("(\\[( )?|( )?\\])", "", allSpeech)

> # remove comments (including the double brackets and their content):
> allSpeech <- gsub("\\({2}[^(\\){2})]*\\){2}", "", allSpeech) >

# remove select punctuation marks:
> allSpeech <- gsub("\\.|\\,|\\?", "", allSpeech)

> # add missing space after period:
> allSpeech <- gsub("\\.\\w", "\\. \\w", allSpeech)
```

Let's check what `allSpeech` now looks like:

```
> allSpeech
[1] "Mind you our Colin's getting more like your dad every day I
€know he is. black welding glasses on, and he turned round and he
made me jump “O:h, Colin”, and then You know, he'll come running
glasses. Yeah. I can't do it. And then me heart jumped, cos he
looked just like your dad, and then a few days later, he's in garage
again, and Johnny come in and he said, “God” he said, “it did give
me a flaming heart attack out there” I said, “why” and he said, “he
turned round with these glasses on” then he said “it was just like
```

```
John's stood there” I says, “don't tell me” I said, “he did the same
thing to me the other day” and it were only within a couple of days
of each other. When he puts those glasses on, honestly he's his
double. i- it's jus'… <truncated>
```

We seem to have cleaned the text sufficiently so that now we can finally form our first subsample, consisting of quoted speech only, identifiable in `allSpeech` by the (bent) quote marks. To pull out the raw matches, we can re-activate the `extract` function defined in Chapter 2:

```
> extract <- function(x) unlist(regmatches(x, gregexpr(pattern, x,
perl = T)))
```

Then we define the pattern along the lines we did in Chapter 2: to ensure that the match stops exactly at the end of each quote, we determine that the quote mark be disallowed from appearing between two quote marks by putting the quote mark into a character class (denoted by `[ ]`)and using the caret ^ inside the class to negate this class:

```
> pattern <- "“[^”]*”"
```

We apply the function to `allSpeech` and store the results in `qu`:

```
> qu <- extract(allSpeech)
```

Did we get the desired quotes?

```
> qu[1:10]
  [1] "“O:h, Colin”"
  [2] "“God”"
  [3] "“it did give me a flaming heart attack out there”"
  [4] "“why”"
  [5] "“he turned round with these glasses on”"
  [6] "“it was just like John's stood there”"
  [7] "“don't tell me”"
  [8] "“he did the same thing to me the other day”"
  [9] "“ya doin’ really well, I didn't tell you noth’n’.”"
 [10] "“you're doing ↑€really really well.”"
```

It looks like we did. How many quotes have we found? Calling `length(qu)` we find there are 319:

```
> length(qu)
[1] 319
```

But that’s not yet the number we are after. As regards quoted speech, there are two numbers we are after: how many words are there altogether and how many words carry CA symbols for paralinguistic activity? We get the first number by using the `strsplit()` function nested inside the `unlist()` function:

```
> qu_w <- unlist(strsplit(qu, " "))
> length(qu_w)
[1] 2309
```

There are 2309 words altogether in `qu`. How many of them are transcribed using CA codes for paralinguistic activity? Here we need to define a pattern again. This pattern is complex as it must match both the various CA codes and the positional variability in how they are distributed across the words: the relevant symbols can appear at the start of the word, inside the word itself, and at the end of the word.

We will focus on the following paralinguistic codes, of which we know from the transcripts they are by far the most common: loud voice (upper-case), soft voice (°), pitch rise (↑), pitch fall (↓), stretching (:), and emphasis (€). The pattern that matches these codes in any position vis-à-vis the word is this (note that for upper-case to be included as signifying loud intensity we set the condition that there be at least two consecutive capitals to rule out that, for example, proper names, which are typically capitalized, are matched):

```
> pattern <- "([€°↑↓:\\w]*[€°↑↓:]+[€°↑↓:\\w]*)|[A-Z]{2,}"
```

The pattern consists of two major chunks: `([€°↑↓:\\w]*[€°↑↓:]+[€°↑↓:\\w]*)` and `[A-Z]{2,}` separated by the alternative marker `|`. The first chunk defines a sequence of three character classes, in which order is not an issue: any character in a character class can appear in any combination with the other members of the class. All three classes contain the CA symbols we are specifically interested in. Additionally, the first and the third class contain `\\w`, which matches alphanumeric strings, that is, in our case, letters. The first chunk, then, matches words that contain, anywhere in the string and in any combination, the symbols `€`, `°`, `↑`, `↓`, or `:`. The second chunk matches any word that contains at least two uppercase letters.

We run the function `extract` again and store the result in `qu_para`:

```
> qu_para <- extract(qu)
> qu_para[1:10]
 [1] "O:h" "↑€really" "€that" "o::h" "°fuck°" "pyja:mas" "↑Aye"
"aGh↑O" "°ah°"
```

How many quoted words carry any of the above CA codes?

```
> length(qu_para)
[1] 194
```

This now is the first number we will use directly in the 2 × 2 contingency table for the chi-squared test and the graphics. If we substract the number from the total number of words in quoted speech, we get the second crucial number – the number of words in quotes without any paralinguistic coding: 2309 – 194 = 2015.

Now for the remaining two numbers: the number of words with, and the number of words without such codings in nonquoted speech. First, we return to `allSpeech`, the vector containing quoted and nonquoted speech and remove all quotes from it and store the result in `nonqu`:

```
> nonqu <- gsub("“[^”]*”", "", allSpeech)
```

Then we split `nonqu` into words:

```
> nonqu_w <- unlist(strsplit(nonqu, "\\s{1,}"))
```

and count how many nonquoted words we have altogether:

```
> length(nonqu_w)
[1] 6670
```

How many of these carry paralinguistic coding? We re-use the same pattern as for quoted words, apply the function `extract`, store the result in `nonqu_para`, and count the elements:

```
> pattern <- "([€°↑↓:\\w]*[€°↑↓:]+[€°↑↓:\\w]*)|[A-Z]{2,}"
> # extract <- function(x) unlist(regmatches(x, gregexpr(pattern,
                x, perl = T)))
> nonqu_para <- extract(nonqu)
> length(nonqu_para)
[1] 317
```

This is the third number required for the contingency table: there are 317 words with paralinguistic coding in nonquoted speech. Subtracting this number from the total number of words in nonquoted speech yields the number of nonquoted words without paralinguistic coding: 6670 – 317 = 6353.

We can finally assemble the contingency table. We start with the two counts for quoted speech and, respectively, nonquoted speech:

```
> qu_counts <- c(194, 2115)
> nonqu_counts <- c(317, 6353)
```

We bind the two vectors together by row, using the `rbind()` function and add appropriate column names and rownames:

```
> counts <- rbind(qu_counts, nonqu_counts)
> colnames(counts) <- c("Quoted speech", "Nonquoted speech")
> rownames(counts) <- c("With paralanguage", "Without paralanguage")
```

Thus we obtain this 2 × 2 contingency table:

```
> counts
                      Quoted speech  Nonquoted speech
With paralanguage               194              2115
Without paralanguage            317              6353
```

We can now perform the test using `chisq.test()`:

```
> test <- chisq.test(counts)
> test
      Pearson's Chi-squared test with Yates' continuity correction
data:   counts
X-squared = 41.882, df = 1, p-value = 9.696e-11
```

The vector `test` contains useful test statistics, such as the expected frequencies and the residuals:

```
> test$expected
                     Quoted speech  Nonquoted speech
With paralanguage         131.4065          2177.593
Without paralanguage      379.5935          6290.407

> test$residuals
                     Quoted speech  Nonquoted speech
With paralanguage         5.460354         -1.341346
Without paralanguage     -3.212697          0.789205
```

The association plot and the mosaic plot used in the case study are contained, not in base R (the association plot and mosaic plot available there are less advanced), but in the package `vcd`. We install this package and load it into the current session:

```
> install.packages("vcd")
> library(vcd)
```

The codes for the two plots could hardly be simpler. The two critical arguments in both plots are the first, the contingency table, and `shade`: if set to `T`, the rectangles in the plots will be colored depending on the size and polarity of the residuals:

```
> assoc(counts, shade = T, varnames = F)        # association plot
> mosaic(counts, shade = T, varnames = F)       # mosaic plot
```

Finally, to obtain the effect size, we call `assocstats()`, a function included in the `vcd` package:

```
> assocstats(counts)
                   X^2 df   P(> X^2)
Likelihood Ratio 39.341  1 3.5580e-10
Pearson          42.559  1 6.8584e-11
```

```
Phi-Coefficient     : 0.069
Contingency Coeff.: 0.069
Cramer's V          : 0.069
```

10.3 Task: Paralinguistic prosody in the use of backchannels in storytelling

In this Task section we will stick closely to paralinguistic prosody and also regex as a method to extract relevant data. The topic though is shifted: we will be looking at backchannels. These are introduced in more detail in the case study in Chapter 12. For the present connection a short sketch must suffice.

Backchannels are small, unobtrusive signals (of any modality – verbal, vocal, or bodily) by which current non-speakers register listenership and understanding (Gardner 1998).

Beside this most basic function, backchannels can take on a range of more specialized functions in context. The most common function is as continuers. Continuers are frequent in multi-unit turns such as storytellings. They index the recipient's alignment with the "structural asymmetry of the storytelling activity" (Stivers 2008: 34) and essentially signal the listener's willingness to pass up the turn until the multi-turn-in-progress is complete (Schegloff 1982). Another group of backchannels are assessment backchannels. This class of backchannels comprises tokens that express emotional responses rather than just structural alignment (Goodwin 1986). A third functional class is formed by convergence tokens (O'Keeffe & Adolphs 2008). They typically convey the listener's convergence on the speaker's opinions (O'Keeffe & Adolphs 2008). (For more detail and examples, see Chapter 12.)

The question asked in this Task section is whether the three classes vary in terms of paralinguistic prosody. In other words: do speakers modulate their voice more when producing continuers, assessments, or convergence tokens?

To address this question, we use the same dataset that will be used for the case study in Chapter 12. The data is described in more detail there. It includes 300 backchannels collected in storytellings from the Narrative Corpus (NC) and re-transcribed in CA style.

The steps to take are the following.

- Read-in the data, available on the companion website as "Chapter10_Task.txt".
- Make a frequency list of all functional codings, stored in column `fun`.
- Store this frequency list as a vector, say, `f_all`.
- Proceed to make a subsample from the dataset containing only those rows where the backchannel tokens contain CA symbols for paralinguistic activity; as you will see from browsing the full dataset, the symbols used in the backchannel transcription are those for loud and soft voice, pitch rise and fall, stretching, and creaky

voice (transcribed by *). So you can use this pattern (note that * need not be escaped here as it is included in the character class, which treats elements *per se* as literal elements):

```
"[°↑↓:*]|[A-Z]{2,}"
```

- You can then subset like this (assuming you have called the dataset `bcfun`):

```
bcfun[grepl("[°↑↓:*]|[A-Z]{2,}", bcfun$bc),]
```

- Next you need to make a frequency list for the backchannel functions in the subsample; store this frequency list in yet another vector, say `f_para`.
- Set up the matrix for the counts. Limit the data to be included to the first three levels of the frequency vectors; these levels are `"assess"` (for assessment backchannels), `"continue"` (for continuers) and `"converge"` (for convergence tokens) – exactly the ones of interest to us (all others have far too small frequencies). To calculate the frequencies of the function codings of backchannel tokens without paralinguistic variation, subtract `f_para[1:3]`from `f_all[1:3]`:

```
with_para <- f_para[1:3]
without_para <- f_all[1:3] - f_para[1:3]
```

- Rowbind them:

```
counts <- rbind(with_para, without_para)
```

- Install the package `vcd`; now you are ready to run the chi-squared test and produce the graphs.

As shown by the two graphs, the p-value is vanishingly small ($p < 0.001$), so we can discard the null hypothesis of no association: the use of paralinguistic resources does depend on whether speakers produce assessments, continuers, or convergence tokens. (Note also that Cramer's V, a measure of effect size, is very solid: 0.493.) While the residual sizes for convergence tokens are minimal, indicating that the differences in use or nonuse of paralinguistic prosody are very small, the residual sizes for assessments and continuers are substantial: assessments are realized with far greater voice modulation than would be expected under the null hypothesis of independence and continuers are realized with much less prosodic variability.

That assessments significantly vary prosodically is a reflection of their being expressive of affect. That continuers show minimal prosodic variation reflects their phonetic minimalness, as they are typically shorter than assessments; more importantly, it reflects their role as indices of the listener's *structural* alignment with the telling activity, a mere 'carry-on' signal that can be given without any affective involvement.

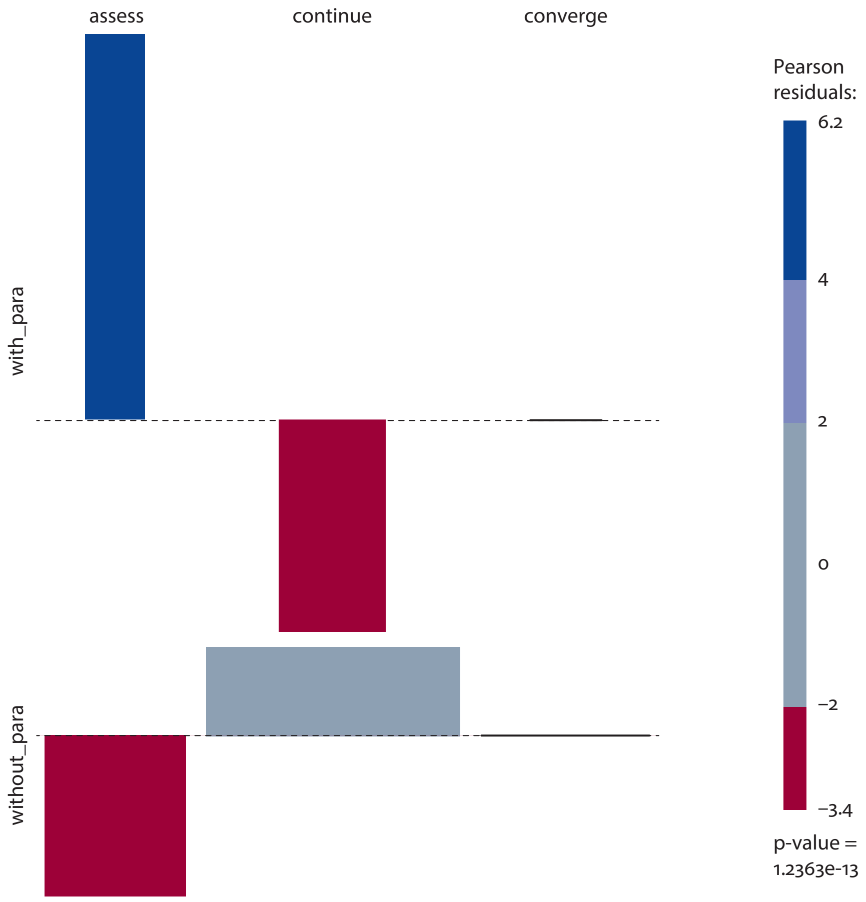


Figure 10.3 Association plot of paralinguistic prosody in the use of three functional classes of backchannels (assessments, continuers, and convergence tokens); height of rectangles represents size of residuals (difference between observed and expected frequencies); width of rectangles is proportional to number of observations

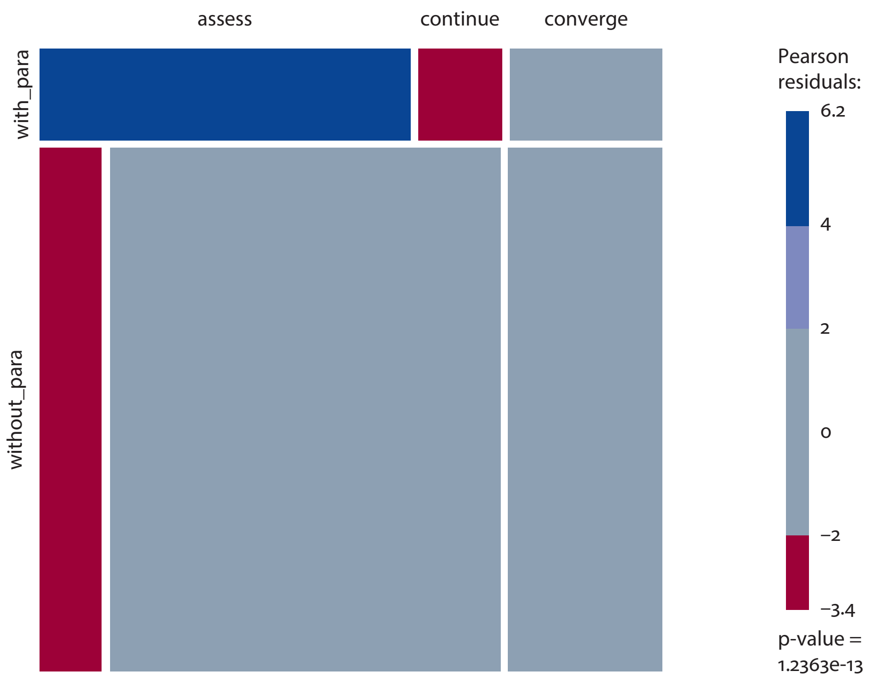


Figure 10.4 Mosaic plot of paralinguistic prosody in the use of three functional classes of back-channels (assessments, continuers, and convergence tokens)

We have now completed our loop through matters related to the sequence and will return to the turn, specifically to the ways speakers end them (Chapter 11) and how co-participants time the transition from one turn to another (Chapter 12).

Chapter 11

Box plots

11.1 *Case study*: Turn completion – Turn-final lengthening

11.1.1 Introduction

A foundational observation is that turn transitions are precision-timed: "[t]ransitions (from one turn to a next) with no gap and no overlap between them are common" (Sacks et al. 1974, p. 700).

The current consensus model to explain that precision timing is a combination of two factors: current non-speakers *predict* when the turn will come to its end and current speakers *project* when they will be ending (Levinson & Torreira 2015). A wide range of resources are used for prediction and projection. The resources could not be more varied and more multimodal altogether forming a *gestalt* that is the basis for pragmatic speech act prediction (Wells & Macfarlane 1998; Levinson & Torreira 2015; Holler & Levinson 2019). Among the linguistic resources, syntax is key (e.g., Duncan 1972; Sacks et al. 1974; Wells & Macfarlane 1998; de Ruiter et al. 2006; Magyari et al. 2014; Levinson & Torreira 2015), but also the use of post-completers such as question tags (Sacks et al. 1974), address terms (Jefferson 1973; Sacks et al. 1974), and 'sociocentric sequences' (Duncan (1972: 287), that is, "stereotyped expressions, typically following a substantive statement" such as 'you know', 'or something', etc. Further, bodily resources serve prediction and projection too. Probably the most eminent such resource is gaze, where current speakers, during the bulk of their turn, typically withdraw their gaze from the interlocutor but return it to them to enter into mutual gaze once they start nearing the turn end (Kendon 1967; Bavelas 2002). Finally, also involved in predicting and projecting turn completion is the huge inventory of vocal resources. Common cues that the speaker is coming to an end include audible out-breath (Ogden 2001), aspiration of word-final plosives (Local & Walker 2012), or a drop in pitch (Beattie et al. 1982) or intensity ('diminuendo') (e.g., Duncan 1972, 1974; Bögels & Torreira 2015). Finally, there is what has been called the drawl or turn-final lengthening, that is the elongation of the last syllable or word in the turn (e.g., Duncan 1972, 1974; Local & Walker 2012; Bögels & Torreira 2015). In this case study we will 'listen' in to that drawl in some more detail.

11.1.2 Data and methods

As in previous chapters, the case study is based on the 'demographically-sampled' subcorpus of the British National Corpus (BNC) featuring natural conversation between intimates and familiars, and the audio files released in the Audio BNC (Coleman et al. 2002).

Using XQuery (cf. Rühlemann et al. 2015) a random sample of 1,000 10-word turns – as noted, likely the average turn length (Rayson et al. 1997; Rühlemann 2018a) – for which audio recordings were available was extracted from the BNC's conversational subcorpus.

To measure the word durations, the recordings for each turn in the sample were accessed through BNCweb (Hoffmann et al. 2008), exported and read into Praat (Boersma & Weenink 2012). In each sound file, the spectral waveforms ('sonograms') were zoomed-in on to determine word boundaries based on 'valleys' in the waveform. Obviously, not all words could reliably be measured: the analysis in Praat produced 7,849 durations.

An established method to measure turn-final drawls has been to compare the durations of one and the same word in two conditions: (i) when the word occurs inside a turn, that is, *not* as the turn-final word, and (ii) when it is used as the turn-final word; cf., for example, Bögels and Torreira's (2015) experiment with a small set of utterance pairs, e.g., "So you're a student?" and "So you're student at Radbout University?", where the word 'student' is used turn-finally and, respectively, turn-medially.

A downside to this method is that in any corpus of natural conversation the number of word types and word tokens occurring sufficiently frequently in both conditions is likely limited to high-frequency word types, that is, essentially to function words (but less so to inserts, of which we know from Chapter 4 that they do not tend to occur at turn ends); lexical words, by contrast, which tend to have very few tokens in any corpus, are unlikely to occur in both conditions *en masse*. Therefore, a sweeping case for turn-final lengthening as a massively used turn-final cue *whatever the turn-final word* will be hard to make. Despite this shortcoming and other shortcomings not mentioned here but elaborated on in Rühlemann and Gries (2020), in this case study we will use just that method.

The graphic used for visualization is the boxplot. While still used rather rarely in linguistics, the boxplot is in fact an unusually useful graphic in that is depicts a lot of information at once.

11.1.3 Results

The subset of 1,000 10-word turns from the BNC contains 5,743 word tokens falling into 226 word types that occur *both* in pre-final positions (i.e., in word slots 1 to 9) and in the final position (slot 10). Note that these numbers are far in excess of the 25 types that occured in both positions in Local and Walker (2012: 260).

The word types in that set are highly varied, ranging from function words (e.g., 'that', 'would'), to inserts (e.g., 'oh', 'er') and lexical words (e.g., 'jumper', 'granny').

A first approximation to the question whether words are longer in the turn-final slot than in pre-final positions is by taking all 5,743 tokens into account regardless of their type frequency and by calculating mean durations for pre-final and final position. The mean duration for all 5,743 tokens in prefinal position is 0.1536376 s, the mean duration in final position is 0.2554404 s. This difference amounts to an impressive overall percentage change from pre-final to final condition of 66.26%. This percentage is a near-perfect replication of the overall change rate found in Local and Walker (2012: 260), which was 65%.

Obviously, this percentage is just the 'big picture.' A lot of variation is still to be discovered inside it. To get closer to that variation we can calculate averages, not for all tokens together, but for each of the 226 types by position and compare the averages by asking, 'By what percentage does *type duration* change from pre-final to final position?'. To address this question, the percentage changes in median duration are calculated for each of the 226 types. The percentage changes are displayed in the scatter plot in Figure 11.1.

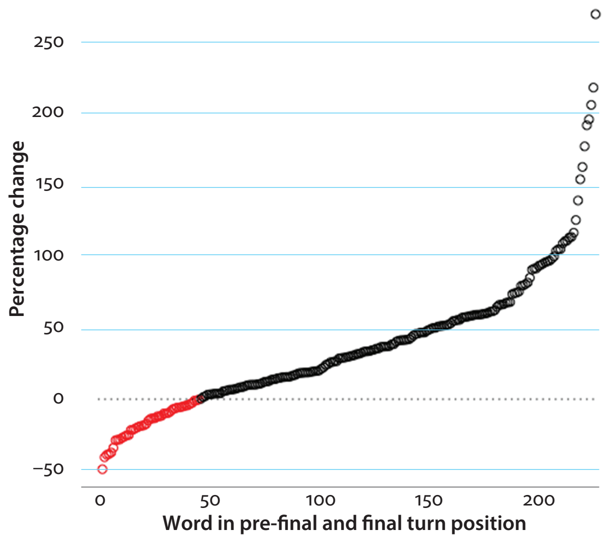


Figure 11.1 Percentage change in median duration of words occuring both in pre-final and final positions in 800 10-word turns in BNC (*N* = 226)

The red circles below the 0 mark on the y-axis represent word types with negative percentage change. These are word types whose median duration is greater in pre-final position than in final position. As can be seen from the graph, their number is quite limited: in fact, there are just 45 out of 226; (i.e., 20%). They form a stark contrast to the 181 words (80%) with positive percentage change represented in the black circles. Also note that the maximum negative percentage change is just −50%, whereas there is a large number of words with far more than +50% positive percentage change; in fact, the greatest positive percentage change is more than 250%.

This is already some clear initial indication that indeed words may be lengthened when used at the end of a turn as opposed to when used in the middle of the turn. But the number of words occurring just once in both positions is large and such low frequencies do not provide a sufficient basis for reliable estimates. Therefore, in a second step, we exclude all those words that occur infrequently in both positions, and focus on words for which there are more observations available by setting the minimum threshold for occurrence in both conditions to 5. As it turns out, the sample contains 31 word types that satisfy this criterion:

's, but, do, er, he, her, here, i, in, it, know, me, n't, now, off, on, one, out, see, she, that, them, then, there, they, think, this, to, up, was, you

Most lexical words have disappeared; nouns and adjectives are gone completely. Most types are function words or items that form part of larger 'idioms', namely 'see' and 'know', which are mainly used in 'you see' and 'you know', as well as 'think' occurring in 'I (don't) think'. With this smaller set with at least five occurrences in both positions, rather than investigating percentage change we can look at the durations in both conditions – pre-final v. final position – directly and compare them. The visualization is provided by boxplots.

Boxplots are an extremely instructive type of graphic: they "not only show the location and spread of data but also indicate skewness" (Crawley 2007: 155). The interpretation of the boxplot depends on its 'syntax', i.e., its main graphical elements. There are five such elements:

- the bold horizontal line
- the box
- the whiskers
- the empty circles
- the notches

The bold horizontal line depicts the median, the middle value of a sorted distribution (cf. Woods et al. 1986: 30–32), a measure of central tendency that is more robust than, for example, the arithmetic mean, which is more easily affected by extreme values.

The box (or 'hinge') represents the interquartile range (IQR): it ranges by default from the first to the third quartile of the data (also referred to as the 25th and 75th percentiles); that is, the box circumscribes the middle 50% of the data around the median. This is useful in indicating skewness. If the data are evenly spread, the median will cut the box into two same-size halves. If the data are skew, one of the two halfes will be larger than the other.

The whiskers can be thought of as 'fences' separating observations lying outside of the IQR (but still somewhat typical of the data) from outliers.

The empty circles designate outliers, that is, "data points with values that are surprisingly large or small given all data points considered jointly" (Baayen 2008: 27). Knowing which values are outliers in this sense may be useful as they may represent erroneous values that have to be removed from the data (cf. Crawley 2007: 157; Levshina 2015: 59).

The notches are an optional feature: the 'waist' around the medians is "intended to give a rough impression of the significance of the difference between two medians. Boxes in which the medians do not overlap are likely to prove to have significantly different medians under an appropriate test. Boxes with overlapping notches probably do not have significantly different medians" (Crawley 2007: 157). That is, the notch depicts the range of values in which you may assume with 95% confidence the 'true' median will lie. If the notch is narrow the median depicted in the box is a reliabale estimate of that true median, whereas a very wide notch indicates the given median is an unreliable indication of the true median.

A curious visual impression is formed when the sample size is too small: the notches will extend beyond the horizontal outlines of the box (the interquartile range, IQR), indicating that the 'true' median covers a range of values that is even larger than the IQR calculated on the basis of the values available. As Crawley (2007: 157) remarks, this "looks odd, but it is an intentional feature, supposed to act as a warning of the likely invalidity of the test."

With this introduction to the syntax of boxplots, we are now ready to inspect the plot in Figure 11.2.

In Figure 11.2, blue boxes show the durations in prefinal position, red boxes depict the durations in final positions; the boxes are sorted in ascending order of the median durations in prefinal position.

For a start, let's inspect the medians (depicted in the horizontal line cutting across the notches): the median duration is higher for all word types in final position but one – only 'her' has a longer median duration in the prefinal condition. At first glance, then, 30 cases – out of 31 – where the word in turn-final position is longer than the same word in prefinal position looks like conclusive evidence that the same words occurring in final positions in turns are lengthened compared to earlier positions. But the picture is more nuanced. The notches extend *beyond* the IQR for a number of

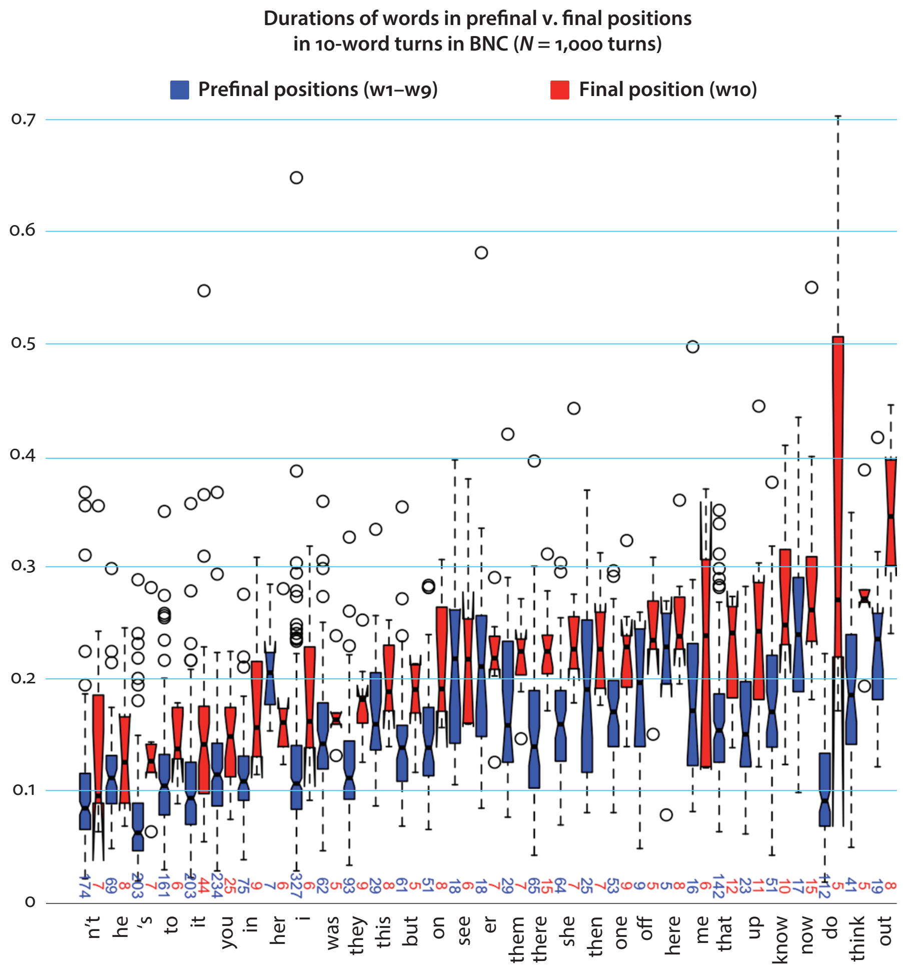


Figure 11.2 Boxplots of durations of 31 word types occurring at least 5 times both in pre-final and final positions in 800 10-word turns in BNC; frequencies of occurrence in both positions are indicated above the x-axis

items, indicating a lack of reliability of the estimate. The most extreme case is the notch for 'do' in final position: the overly wide range covered by the notch is due both to the limited number of observations, namely just 5, and the wide spread of the observed durations, which range from just above 0.2 seconds to over 0.7 seconds. In this case, as well as in similar cases, the median duration in final position turns out to be higher than in pre-final position in the sample but that comparison may or may not generalize to the wider population of these words in the two conditions – we simply have little robust evidence to make that generalization.

However, the notches for the two conditions do differ starkly for the overwhelming majority of the word types. This is the case for as many as 18 types including ''s', 'but', 'in', 'it', 'know', 'one', 'out', 'she', 'that', 'them', 'there', 'they', 'think', 'to', 'up', 'was', 'you'. For this large group of words we can be rather confident that under an appropriate test the median durations for final positions will be larger than for pre-final positions.

11.1.4 Discussion and concluding remarks

The analysis produced two main findings: (i) most words that occur with sufficient frequency in pre-final and final positions in turns are function words and inserts and (ii) words used in turn-final position are mostly longer than the same words used in pre-final positions in the turn.

Based on the two observations we have evidence that *inserts and function words* used in turn-final position are lengthened when compared to the durations the same words have in pre-final positions. We do not have sufficient evidence that lexical words get lengthened in the same way. This is in mild contrast to previous work on turn-final drawls which focused on durations of nouns in prefinal and final positions (e.g., Bögels & Torreira 2015; Local & Walker 2012). Thus, turn-final lengthening does seem to be confirmed but based on the (limited) evidence we have discovered in this case study we cannot claim that lengthening concerns any type of word – it might be restricted to inserts and function words.

But before we dismiss the idea that speakers use turn-final lengthening as a turn-yielding cue *whatever the word* let's bear in mind that the method we used was simplistic: we only looked at how durations vary in two positional conditions, pre-final and final. The duration of a word in discourse depends on a far larger set of factors (e.g., Local & Walker 2012). It is obviously beyond the scope of this case study to describe, let alone explore this set as a whole; the reader is referred to the multifactorial analysis in Rühlemann and Gries (2020).

11.2 The boxplot in the case study

The data underlying the boxplot in the case study, stored on the companion website as "Chapter11_Casestudy.txt", is uploaded here as `dur`; the code is stored on the companion website as "Chapter11_Code.R".

```
> dur <- read.table("[Your path]/Chapter11_Casestudy.txt",
+                         header=T, quote="", sep="\t", fill=T)
```

Calling `str(dur)` reveals a number of things: there are 1,000 observations and 23 columns, the duration columns `d1`, `d2`, etc. contain many `NA`s, and, as usual, R treats the word columns `w1`, `w2`, etc. as factors; the latter two aspects will require our attention at a later stage:

```
> str(dur)
'data.frame': 1000 obs. of 23 variables:
 $ file    : Factor w/ 56 levels "KB0","KB1","KB2",..: 40 47 48 47
             14 19 28 23 28 7 …
$ speaker : Factor w/ 263 levels "KB1PSUNK","KB9PSUNK",..: 137 221
             235 221 59 78 96 83 95 263 …
$ turn    : Factor w/ 998 levels ", and I ended up virtually people
             in a fish tank .",..: 4 5 12 14 24 27 30 31 35 21 …
$ d1      : num NA NA 0.136 0.421 0.177 0.262 0.132 0.186 0.103 0.316 …
$ d2      : num NA NA 0.115 0.054 0.282 0.156 0.239 0.089 0.168 0.112 …
$ d3      : num NA NA 0.081 0.084 0.077 0.091 0.214 0.079 0.198 0.204 …
$ d4      : num NA NA 0.084 0.217 0.183 0.153 0.234 0.101 0.188 0.113 …
$ d5      : num NA NA 0.109 0.346 0.164 0.062 0.258 0.387 0.359 0.188 …
$ d6      : num NA NA 0.373 0.248 0.111 0.219 0.381 0.106 0.343 0.067 …
$ d7      : num NA NA 0.254 0.146 0.148 0.303 0.171 0.139 0.064 0.228 …
$ d8      : num NA NA 0.157 0.099 0.153 0.171 0.064 0.169 0.075 0.317 …
$ d9      : num NA NA 0.426 0.081 0.137 0.316 0.105 0.122 0.095 0.142 …
$ d10     : num NA NA 0.231 NA 0.265 0.302 0.644 0.241 0.367 0.299 …
$ w1      : Factor w/ 180 levels "a","actually",..: 166 1 5 5 10 10
             10 10 10 10 …
$ w2      : Factor w/ 228 levels "'d","'ll","'m",..: 87 70 27 91 34
             87 87 87 87 7 …
$ w3      : Factor w/ 284 levels "'d","'ll","'m",..: 135 113 268 5
             5 3 38 45 244 246 …
$ w4      : Factor w/ 336 levels "'d","'ll","'m",..: 118 293 335 336
             191 191 182 183 197 155 …
$ w5      : Factor w/ 375 levels "'","'ll","'m",..: 158 157 4 121
             15 7 367 262 218 178 …
$ w6      : Factor w/ 386 levels "'d","'em","'ll",..: 161 8 327 378
             339 119 138 365 264 8 …
$ w7      : Factor w/ 388 levels "'d","'ll","'m",..: 321 343 325 320
             222 190 76 284 90 184 …
$ w8      : Factor w/ 372 levels "'d","'ll","'m",..: 215 262 151 87
             138 210 152 265 196 241 …
$ w9      : Factor w/ 369 levels "'m","'re","'s",..: 16 176 62 208
             40 350 3 159 80 368 …
$ w10     : Factor w/ 470 levels "'ll","'s","'ve",..: 285 390 78 57
             252 88 137 435 208 218 …
```

The data format is unfortunate: the durations, which are stored across ten columns, are in fact one variable, and the same can be said of the ten word columns. So we will convert the format into a long format, containing only variables of interest and concatenating them in one column each. We start with the durations. Concatenating them in a single vector, called `d`, can be achieved in multiple ways. An economical way is subsetting `dur` on the relevant columns, columns #4 through #13, and unlisting the result with the function `unlist()`:

```
> d <- unlist(dur[4:13])
```

To collect the word tokens in a single vector, called `w`, we apply the same logic:

```
> w <- unlist(dur[14:23])
```

We can combine the two vectors in a dataframe called `df`, and call `str(df)` to inspect its structure:

```
> df <- data.frame(d, w)
> str(df)
'data.frame': 10000 obs. of 2 variables:
 $ d: num NA NA 0.136 0.421 0.177 0.262 0.132 0.186 0.103 0.316 …
 $ w: Factor w/ 1535 levels "a","actually",..: 166 1 5 5 10 10 10
      10 10 10 …
```

We have now assembled all 10,000 duration and word values in separate columns, `d` and `w`. Yet, we need a third, *positional* variable to assign the word tokens to prefinal or final position. This variable is easy to obtain: as we have exactly 10,000 rows, the first 9,000 are for prefinal words and the last 1,000 for final words. Based on this knowledge, we define the column `df$p` using `c()` and `rep()`:

```
> df$p <- c(rep("prefinal", 9000), rep("final", 1000))
```

We still have many durations that are `NA` – we can now remove them. We do it by using `na.omit()`, which deletes any *row* in a dataframe that contains `NA` in any cell:

```
> df <- na.omit(df)
```

If we now call `nrow()` to count the number of rows remaining, we get 7,849: the number of words with a durational value:

```
> nrow(df)
[1] 7849
```

The next step is crucial: we need to identify the word *types* that occur with sufficient frequency in both positions, prefinal and final. To obtain frequencies, the `table()` function is handy. We start with words in prefinal position, using `sort()` and its argument `decreasing = T` to sort the frequency table in descending order, `table()` to calculate the table, and subsetting `df$w` on the value `"prefinal"` in `df$p`:

```
> prefinal <- sort(table(df$w[df$p=="prefinal"]), decreasing = T)
```

At this point `prefinal` is a `"table"` with a complex structure:

```
> str(prefinal)
 'table' int [1:1535(1d)] 327 234 213 203 203 174 161 147 142 122 …
 - attr(*, "dimnames")=List of 1
  ..$ : chr [1:1535] "i" "you" "the" "it" …
```

To obtain a more easy-to-handle structure, we store `prefinal` as a dataframe using `as.data.frame()`, and inspect the first five lines:

```
> prefinal     <- as.data.frame(prefinal)
> prefinal[1:5,]
   Var1  Freq
1     i   327
2   you   234
3   the   213
4    's   203
5    it   203
```

Before we move on let's convert `Var1` from factor to character, as we will need this later on:

```
> prefinal$Var1 <- as.character(prefinal$Var1)
```

Then we execute the same procedure for the word tokens in final position:

```
> final <- sort(table(df$w[df$p=="final"]), decreasing = T)
> final <- as.data.frame(final)
> final$Var1 <- as.character(final$Var1)
> final[1:5,]
   Var1 Freq
1    it   44
2   you   25
3   now   15
4 there   15
5  that   12
```

On comparing the two frequency lists we notice that the frequencies are much greater in `prefinal` (which is not surprising as this list was computed on nine times as many word tokens as `final`) and that the most frequent word types are quite different: for example, the 1st and 2nd personal pronouns are among the top most frequent items in `prefinal` but not in `final`.

The next step is to filter the two frequency lists on two conditions: we want to single out those items that (i) have a minimum frequency of five occurrences (ii) in *both* lists.

To get at the intersecting items we use `intersect()` and to filter out frequencies below five occurrences we subset using the >= operator, storing the result in the vector `BOTH`:

```
>BOTH<-intersect(prefinal$Var1[prefinal$Freq>=5],final$Var1[final$Freq
>= 5])
```

How many items fullfil the two criteria?

```
> length(BOTH)
[1] 31
```

Which words are these 31 types?

```
> BOTH
 [1] "i" "you" "'s" "it" "n't" "to" "that" "do" "they" "in" "he"
"there"
[13] "she" "was" "but" "one" "know" "on" "think" "them" "this"
"then" "up" "out"
[25] "er" "see" "now" "me" "off" "her" "here"
```

We now need to collect the durations of all the *tokens* of these types and we need to make a distinction between the token-duration pairings depending on their position in the turn.

Let's start by extracting all those tokens of the words in `BOTH` that occur in *prefinal* position. We subset `df` on those word tokens in `w` that are assigned the `"prefinal"` value in `d$p` and (`&`) that are contained in (`%in%`) the vector `BOTH`:

```
> PrefinalTokens <- df$w[df$p=="prefinal" & df$w %in% BOTH]
```

To get the durations of these prefinal tokens, we proceed similiarly, by subsetting `df` on those durations in `d` that are `"prefinal"` in `p` and where the word types are contained in `BOTH`:

```
> PrefinalTokensDur <- df$d[df$p=="prefinal" & df$w %in% BOTH]
```

We also need the same type of data for the *final* tokens of the words contained in `BOTH`. We get it by adapting the methods used for prefinal tokens:

```
> FinalTokens <- df$w[df$p=="final" & df$w %in% BOTH]
> FinalTokensDur <- df$d[df$p=="final" & df$w %in% BOTH]
```

This gets us a total of four vectors, two for each condition (prefinal and final position). Let's define two like-structured dataframes, one for each condition, and, in each dataframe, include a third variable, namely `group`, defining it as `"prefinal"` in the prefinal-position data, and as `"final"` in the final-position data:

```
> DataPre <- data.frame(
+   group = "prefinal",
+   word = PrefinalTokens,
+   duration = PrefinalTokensDur)
```

```
> DataFin <- data.frame(
+   group = "final",
+   word = FinalTokens,
+   duration = FinalTokensDur)
```

The two dataframes have the same structure and the same column headings, so we can rowbind them into one dataframe using the `rbind()` function:

```
> DATA <- rbind(DataPre, DataFin)
> head(DATA)
     group word duration
1 prefinal  but    0.153
2 prefinal  but    0.159
3 prefinal  but    0.164
4 prefinal  but    0.073
5 prefinal  but    0.158
6 prefinal  but    0.172
```

We could plot the boxplots now. If we want to order the boxes, however, one more intermediary step is required. In Figure 11.2 in the case study we chose to sort the boxes in ascending order of the median in the `"final"` group. To order the boxes this way we use `levels()`, a function which accesses the levels attribute of a variable, and `reorder()` (for more detail see `?levels` and `?reorder`):

```
> ordered <- levels(reorder(DATA$word[DATA$group == "final"],
+                          DATA$duration[DATA$group == "final"],
+                          FUN = median))
> DATA$word <- factor(DATA$word, levels = ordered)
```

Now we can start! As always, we first set up the basic parameters of layout and plotting margins:

```
> par(mfrow=c(1,1), mar=c(4,4,4,0))
```

As we have three variables – durations, words, and groups – we need to use this format for the data input into the `boxplot()` function: *var_1 ~ var_3 + var_2*. In our case: `DATA$duration ~ DATA$group + DATA$word`.

From this point onward, the plot itself is straightforward: we set the title and its character size, the x- and y-axis labels, enable the notches in `notch = T` (nevermind the warning issued at this point that “some notches went outside hinges”!), choose the colors for the two conditions, suppress the frame with `frame = F` and the x-axis in `xaxt = "n"`.

```
> boxplot(DATA$duration ~ DATA$group + DATA$word,
+ main = "Durations of words in pre-final v. final positions\nin
10-word turns in BNC (N = 1,000 turns)",
+          cex.main = 0.9,
+          ylab = "Durations (sec.)",
+          xlab = "",
+          notch = T,
+          col = c("blue", "red"),
+          frame = F,
+          xaxt = "n")
```

We select the x-axis by inputting `1` into `axis()`. Where to place the axis ticks and the labels? We know we have 62 boxes (as we have 31 word types in `BOTH`) and we want to place the tick exactly *between* each pair of boxes. Using the argument `at` we define a sequence ranging from 1.5 to 61.5 and divide it by 2 to obtain 31 ticks. For the axis labels we make the word types available that are stored in `DATA$word` using `levels()`. Finally, we set character size and character orientation: `las = 2` will rotate the labels by 90°.

```
> axis(1,
+       at = seq(from = 1.5, to = 61.5, by = 2),
+       labels = levels(DATA$word),
+       cex.axis = 0.9, las = 2)
```

The legend is set to some location in the plot where it does not interfere with the whiskers and/or outliers (using trial and error!); the legend text is set in `legend` as well its character size in `cex`; the legend boxes are filled with the colors chosen for the boxes with `fill = c("blue", "red")`, the border around these legend boxes is set to white with `border = "white"`, the default frame around the legend as a whole is suppressed through `bty = "n"`:

```
> legend(1, 0.65,
+        legend = c("Pre-final positions (w1-w9)", "Final position
                   (w10)"),
+        cex = 0.8,
+        fill = c("blue", "red"),
+        border = "white",
+        bty = "n")
```

All that is missing at this point are the frequencies for the word types in the two conditions, which we will print underneath the boxes, just above the x-axis. We table `DATA$word` against `DATA$group` to obtain the frequencies of all word types by group.

```
> f <- table(DATA$word, DATA$group)
> head(f)
      prefinal final
n't        174     7
he          69     8
's         203     7
to         161     6
it         203    44
you        234    25
```

The data structure we now have is a table. As above, to be able to address the frequencies more easily, we convert the table to a dataframe using `as.data.frame()`:

```
> f <- as.data.frame(f)
```

To print the frequencies onto the plot, we need to determine where to plot them on the x- and y-axis: for the x-axis we define a sequence from 1 to 62 (the total number of boxes) divided by 2 and, on the y-axis, place it sharply above the axis itself. The data to print is found in `f$Freq[f$Var2 == "prefinal"]`, that is, the frequency values for the word types in the `"prefinal"` group. The prefinal boxes in the boxplot were blue, so we make the frequencies blue too. Finally, `srt = 270` rotates the labels.

```
> text(seq(from = 1, to = 62, by = 2), 0.018,
+       f$Freq[f$Var2=="prefinal"],
+       cex = 0.6,
+       col = "blue",
+       srt = 270)
```

We take similar steps for the frequencies in the `"final"` group; the differences are: (i) we start `seq` at 2, (ii) we subset the frequencies in `f$Freq` on the condition that the level in `f$Var2` be `"prefinal"`, and (iii) we select `"red"` for the numbers:

```
> text(seq(from = 2, to = 62, by = 2), 0.018,
+       f$Freq[f$Var2=="final"],
+       cex = 0.6,
+       col = "red",
+       srt = 270)
```

Now the plot is complete.

11.3 Task: Turn-final lengthening in 9-word turns in the BNC

The task in this section is straightforward: it asks you to compare pre-final and final durations in boxplots on a different dataset. This dataset is comparable in many ways to the dataset of the case study: it contains turns from the conversational subcorpus of the BNC, as well as durations measured in Praat for all words in the sample (unless unavailable due to poor audio quality). The only major difference is that the turns are nine words long. The data is stored on the companion website as file "Chapter11_Task.txt".

Also, it should be noted that the data were underlying a study on the durations of syntactic and pragmatic 'well' (Rühlemann 2018c); therefore all turns contain the word 'well'. 'Well' is not only highly multi-functional performing multiple syntactic and pragmatic sub-functions but also versatile with regard to position in the turn. It will, then, not be surprising that 'well' features among those word types that occur both in pre-final and final positions in the dataset.

The major steps in programming the boxplots are the following:

- convert the data to the long format, with each variable stored in a single column
- determine which word types occur both in prefinal and final positions
- select those word types that have a minimum token frequency of 5 occurrences in both conditions
- match the prefinal and final durations to the tokens of these word types
- make sure the boxes get sorted in ascending order of the medians in final position
- plot the durations in boxplots
- print the number of tokens of each word type in either position at the bottom of the graph
- once you're done, you should have a graphic that looks roughly like the one in Figure 11.3.

The boxplot shows that six items satisfy the criterion that they occur at least five times in both conditions (pre-final and final position); also, it shows that five out of these, or 80%, have greater durations in final position: 'well', 'it', 'they', 'she', and 'there'. The only item defying the trend is 'you'.

As far as the notches are concerned, it seems that the differences in duration may be significant for all five items with longer durations in final position. The difference between pre-final and final 'you' by contrast is unlikely to be statistically significant. So, on the whole, in this dataset too words do get lengthened in turn-final position.

A final and cautionary word relates to 'well': when 'well' occurs in final position in a turn it is most of the time fairly distinct from when it occurs in the first or second

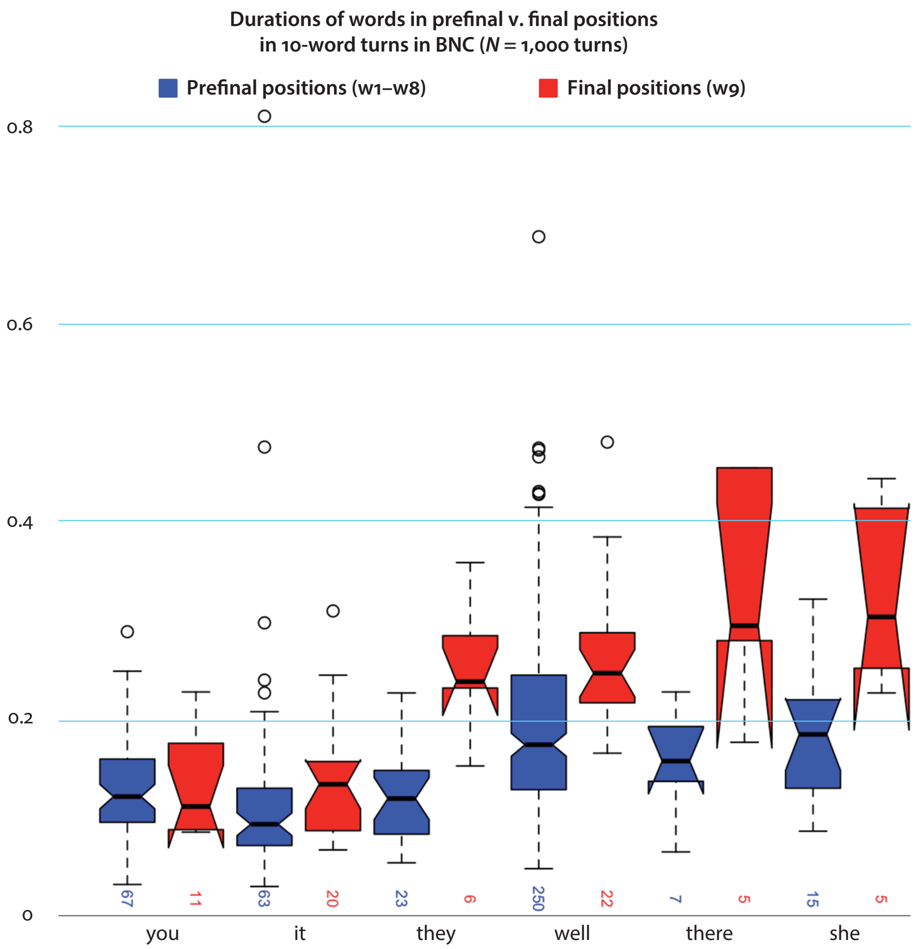


Figure 11.3 Boxplots of durations of words in pre-final and final positions in 9-word turns

position in the turn – not only in durational terms. We know that turn-initial 'well' is virtually always pragmatic 'well', a pre-start in Sacks et al.'s (1974) sense relating the developing turn to the prior turn. In turn-final position, by contrast, it virtually always co-occurs with 'as' in 'as well', a phrase which serves as an additive subjunct. For example:

(11.1) I meant to put this one out as **well** (BNC: KB0 442)

We get a sense that this 'well' is lexically and functionally quite at a remove from pragmatic 'well'. So the question arises: Are pragmatic 'well' and additive subjunct 'well' one and the same *word*? They would have to be if we take the present analysis to indicate that even 'well' gets lengthened in turn-final position. Alternatively, additive subjunct ‘well’ could be considered a lexeme in its own right, with its own distinct durational behavior. And as it overwhelmingly only occurs at turn ends, we could not even be sure whether the greater length of ‘as well’ is owed to turn-final lengthening or whether the length is just part of the lexeme itself. To examine ‘well’ in the context of the turn-final drawl, a larger database would be required, one in which ‘as well’ occurs in larger numbers and in more positions than just the final one. Only then could we be certain that we are comparing the same word in two different turn positions. For the time being we are probably better off taking ‘well’ out of the equation of turn-final lengthening.

Chapter 12

Histograms and density plots

12.1 *Case study*: Turn transition – Backchannel response time

12.1.1 Introduction

The case studies so far have all dealt with phenomena *inside* turns and sequences. In this last chapter, the spotlight moves to the space *between* turns, the transition space separating the just-completed turn from the just-commencing one.

The way people, across languages and cultures, manage this transition seems to be "strongly universal, with only slight variations in timing" (Levinson 2016: 7): transition commonly happens within just a 'slight gap' (Heldner & Edlund 2010: 557; cf. also Sacks et al. [1974: 700–701]). This slight gap is roughly the length of a single syllable (Levinson 2016: 7), namely 200 ms – that is, "faster than the average time it takes to blink the eye" (Enfield 2017: 41; see also Stivers et al. 2009; Heldner & Edlund 2010; Dingemanse & Enfield 2014; Riest et al. 2015; Roberts et al. 2015). The contexts in which this 200 ms response time was observed were responses to yes/no questions (Stivers et al. 2009) and also functionally unrestricted contexts (Heldner & Edlund 2010; Riest et al. 2015; Roberts et al. 2015).

The 200 ms benchmark is obviously just a central tendency hiding considerable variation. For example, transition may accelerate when speakers compete to get the next turn (Enfield 2017); it is also faster when the medium of conversation is the telephone rather than face-to-face conversation (Levinson 1983; ten Bosch et al. 2005; Levinson & Torreira 2015). Conversely, transitions may be slower following longer and/or more complex previous turns (Roberts et al. 2015: 2) or following turns that end in less predictable ways (Magyari et al. 2014). Finally, deceleration may be built into the design of 'dispreferreds', that is, responses not, or not straightforwardly, aligned to the course of action initiated in the prior turn (Levinson & Torreira 2015: 12). So, clearly the time that elapses between one and the next turn may vary due to "the functional role of each turn in communication" (Roberts et al. 2015). Another, complex, functional context in which transition time is likely to vary considerably is that of backchannels.

Backchannels are universally occurring multimodal displays by non-current speakers of listening to and understanding the current speaker's turn, and of their willingness to continue listening (cf. Schegloff 1982). In English, the most frequent vocal backchannels *mm*, *yeah*, *oh*, *mhm*, and *uh huh* are widely shared across varieties such as American English (White 1989), Australian English (Wong & Peters 2007), New Zealand English (Wong & Peters 2007) and British English (Rühlemann 2017). Backchannels have very high rates of occurrence in natural conversation: Gardner (1998: 205) estimates that backchannels "occur more than a thousand times in a single hour of talk", an estimate corroborated in recent quantitative research (Wesseling & van Son's 2005; Bavelas et al. 2000).

Backchannels forms do not only include vocal forms; they also include bodily forms such as gestures, head nods, extended blinks, and even eyebrow flashes (e.g., Goodwin 1986; Stivers 2008; Stivers et al. 2009; McCarthy 2003; Hömke et al. 2017). There is some agreement that backchannels do not constitute full turns but rather serve as signs that the listener passes up the turn to the current speaker; in Yngve's words, they are talk "in the back channel, over which the person who has the turn receives short messages such as *yes* and *uh-huh* without relinquishing the turn" (Yngve 1970: 568).

Listeners do not produce backchannels anywhere in the current speaker's turn but most commonly at "TCU completions" (Stivers 2013: 201), that is, "at the boundaries of turn-constructional units, precisely the sequential position that is able to demonstrate both that one unit has been received and that another is now awaited" (Goodwin 1986: 208).[42] Visual backchannels, by contrast, such as head nods, blinks, gestures, etc. are not turn-organized and their position may not be aligned with the TCU (Stivers 2008).[43]

The most common environment for backchannels is in 'tellings' (cf. Schegloff 1993: 105), that is, expanded sequences characterized by an epistemic asymmetry with one participant imparting a chunk of knowledge to the other and a structural asymmetry with the 'teller' taking multi-unit turns built out of *multiple* successions of TCUs that offer "places in it for others' talk" (Sacks 1992: 526). In storytelling, a key type of telling sequence, backchannels represent by far the most common response type produced by story recipients (Rühlemann 2013).

While the function underlying all backchannels is signaling understanding (cf. Gardner 1998), a number of additional functions can be distinguished, some of them so distinct that "the brief recipient vocalizations that occur during ongoing talk might in fact be divided into different classes" (Goodwin 1986: 207). A critical distinction is

42. A much less frequent backchannel placement is 'interjacent' (Jefferson 1986: 158), i.e., mid-way into the TCU (cf. Rühlemann 2018b).

43. Stivers (2008: 43) observes that nods are used in different sequential environments than vocal continuers: "nods are typically positioned following telling elements that provide the recipient with access to the events being reported in the telling or directly to the teller's stance toward those events."

between continuers (or generic backchannels) and assessments (or specific backchannels) (Goodwin 1986). Continuers are responses that exhibit "an understanding that an extended unit of talk is underway by another [speaker] and that it is not yet, or may not yet be (…) complete" (Schegloff 1982: 81). Continuers serve a 'bridging' function by treating the speaker's current "[turn constructional] unit as a preliminary to another" (Goodwin 1986: 210) or "as a prelude to further talk" (Goodwin 1986: 214). Assessments, by contrast, serve to comment "on the specifics of what is being said in the current unit" (Goodwin 1986: 210) by evaluating the speaker's ongoing talk "in a particular way as remarkable" (Goodwin 1986: 211) and potentially "embody[ing] elaborated participation display" (Goodwin 1986: 211). These displays are characteristically of an evaluative nature. Speaking of what they call 'engaged response tokens', a class matching Goodwin's class of assessment backchannels, O'Keeffe and Adolphs (2008: 84) note that they "express genuine emotional responses such as surprise, shock, horror, sympathy, empathy and so on".

Continuers and assessments seem to occupy the extreme ends of the functional range of backchannels. The functional middle ground is covered by backchannels that do more than just signal continued listenership but less than express strong affect. A large class performing this function is the class of what O'Keeffe and Adolphs (2008) refer to as 'convergence tokens'; their main 'job' is to signal agreement on factual information conveyed or opinions expressed by the main speaker rather than their affective stance. In turn-by-turn talk, O'Keeffe & Adolphs note, convergence tokens often occur at topic boundaries. Prime examples of convergence tokens include 'yeah', 'that's right', and also 'no' said to affirm negated information. Given their capacity to signal agreement on, and recognition of, specific information and also to convey convergence on opinions, convergence tokens can be seen as forms of interactional bonding between speaker and addressee helping to "maintain good relations between speakers by reinforcing commonality between them" (O'Keeffe & Adolphs 2008: 87). Continuers, convergence tokens, and assessments are thus best understood as presenting a cline from least affective (continuers) to more affective (convergence tokens) to strongly affective (assessments). Finally, a fourth functional class accommodates backchannel forms that have their locus in sequence-recompletion, a practice "whereby participants reoccasion the relevance of sequence completion after the sequence was already treated as complete (i.e., after a lapse)" (Hoey 2017: 49).

Fragment (12.1) illustrates both continuers, convergence tokens, and assessments in storytelling. The storyteller Martine is relating a near-accident she had. She introduces the story by a preface in line 1: "Go::', I was nearly in an <u>ac</u>cident last night, °that reminds me.°". The preface is indicative of the teller's stance in two ways: the reference to "<u>ac</u>cident", highlighted by emphasis on the first syllable, relates the story to a life-threatening incident, comparable to Labovian danger-of-death stories (cf. Labov 1972). Similarly, the underlying semantics of the protracted interjection "Go::", a clipped form of 'oh my god', also relates the upcoming story to an extreme existential experience causing distress. Her stance toward the event is, then, that of being terrified. In line 2, Martine starts to develop the story background, providing a first place reference in "off the (0.4) Wrexham road" and, after a 1.3 s silence, a more specific reference in "along the Paisword one" in line 3. Merielle's backchannel "[mm]" indexes recognition of the location and thus allows the teller to continue with the story by providing more background information. First, in lines 5–7, she names the three protagonists of the story, namely the three cars and drivers involved in the near-accident (the pick-up, the Escort van, and herself). Then, in lines 8–9, Martine notes that "there's not many places to overtake down there"; this provides a first-mention of overtaking, the manouvre that is going to lead to the near-accident. The scarcity of space for overtaking is confirmed by Merielle's convergence token "[no]" in line 10, by which Merielle actively confirms Martine's observation that at the location indicated "there's not many places to overtake". Once this confirmation is given, Martine swiftly, as indicated by the overlapped marker "[so]" in line 11, moves the story closer to the Climax by describing in lines 12–17 how things got dramatic when one of the two other drivers started to overtake the slow pick-up just as Martine "could see a car °coming the other way°". To make matters worse, the driver got stuck "running alongside this pick-up" unable to drop back and unable to overtake. This description, and the clear danger that transpires from it, is acknowledged by Merielle's "M[m]" in line 18, spoken partly in overlap and a slight increase in intensity (indicated by the capital). At this point the Climax is reached: to escape the collision the overtaking driver very nearly manages to squeeze in behind the slow pick up "jamm[ing] its anchors" (line 20) just as the car coming the other way speeds past him. That climactic moment is depicted by Martine's non-verbal "↑we:o:w:↑", imitating the sound of the car coming the other way rushing past at high speed. Just a split second before Martine produces that depiction, Merielle utters "Oh m[y god]" in line 23, partly overlapping Martine's "[erm]". Merielle's backchannel does more than just bridge over from one unit to the next, providing instead an empathic assessment of the near-accident and the predicament Martine was in (her car being in close vicinity to the two cars that nearly escaped each other). Note also that Merielle's "Oh my god" is a full rendition of Martine's "Go::" by which she launched the whole telling – the stance convergence could hardly be more perfect:

(12.1)

1	Martine:	Go::', I was nearly in an <u>ac</u>cident last night, °that reminds me.°
2		Coming- I'd just turned off the (0.4) Wrexham road, (1.3)
3		er going along the Paisword one (.)
4	Merielle:	**[mm]**
5	Martine:	[and] there's like an old pick-up-ey type thing (0.6)
6		and then there w's this (0.3) young lad in a (1.4) in a work's Escort van
7		and then there was me (1.0)

8 erm (1.0) I thought (.) oh bit of a blow being behind this vehicle,
9 but there';s not many places to overtake down there =
10 Merielle: **[no]**
11 Martine: = [so] I was jus' gonna have to, (.) y' know, be stopped there (0.3)
12 anyway this young lad (0.9) after the first (part) bend
13 he went to overta'
14 an' I could see a car °coming the other way° (0.7)
15 erm he was like er running alongside this pick-up (0.5)
16 bu' i' was like as if he couldn't drop back
17 and he couldn't make it either?
18 Merielle: **M[m]**
19 Martine: [so] he j's kept going, so the (.) pick-up
20 jammed its anchors on,
21 the' w's a, the road that was coming the other way,
22 the road that went up the verge (0.4)
23 Merielle: **Oh m[y god]**
24 Martine: [erm] the- ↑we:o:w:↑ (0.4) just squeezed in
25 and this lad sort of shit himself completely
26 and VOO::m right up the road then, (0.5)
27 well I (0.4) I knew what was gonna 'appen
28 so I dropped right back a:n:d er (1.1)
29 how he missed him I don't know,
30 must 'u been inches
31 (2.4)
32 or I(h)TCH(h)es
33 Mike: ((laughs))
34 (10.1)

(BNC KD8: 7592-7600; corrected transcription)

The question we are going to address in this short case study relates to the three functions as well as transition time: do continuers, convergence tokens, and assessments differ in terms of response time?

12.1.2 Data and methods

The case study is based on the *Narrative Corpus* (NC; Rühlemann & O'Donnell 2012) introduced earlier in Chapter 10. As noted, the NC contains more than 500 storytellings and contains exhaustive annotation. Key in the present connection is the annotation of backchannel utterances.

Based on the recent release of the Audio BNC (Coleman et al. 2012), backchannels in the NC were analyzed phonetically. The phonetic analysis involved the following procedure. First, all vocal backchannels were retrieved (as well as the storytelling TCUs to which they were a response). Second, using phonetic analysis software, response times for the retrieved backchannels were measured. In a first pass, all backchannel response times were measured in Audacity <http://www.audacityteam.org/> ; in a second pass, response times between 0 and 200 ms were re-measured in Praat (Boersma & Weenink 2012). After filtering out false positives and unintelligible backchannels,[44] the total number of tokens available was 609.

Given the aim in this case study to compare the response times for different functional classes of backchannels, a random subset of 300 backchannels was manually coded for function type by taking the whole storytelling sequence into account as well as repeatedly listening to the relevant context.[45]

The graph type used to visualize response times is the histogram. This plot type is "excellent for showing the mode, the spread and the symmetry (skew) of a set of data" (Crawley 2007: 162) and it is a "popular way of visualizing a univariate continuous distribution" (Levshina 2015: 50) such as the distribution of response times, measured in milliseconds.

Histograms divide data into equi-spaced intervals, called 'bins', and draw rectangles the heights of which are proportional to the number of values in the bin. They are often overlaid by a 'densitiy curve', which "smoothes discrete jumps of the histogram" (Baayen 2008: 25–26). Thus, for continuous data, "the density curve is both more appropriate and more accurate" (Baayen 2008: 26)

12.1.3 Results

The distribution of all 609 backchannel response times available in audio in the NC is shown in Figure 12.1; the horizontally aligned 'rugs' indicate the backchannels' precise location on the duration cline:

44. These included backchannels not responding immediately to the storyteller's TCU but to another recipient's response (i.e., a backchannel by one recipient following or in overlap with another recipient's backchannel or another recipient's more substantial contribution). Further discarded were response tokens that functioned as responses to yes/no questions, such as 'yeah' in response, for example, to "Zoe you know Zoe". Also excluded were responses consisting entirely of laughter tokens as well as backchannels whose temporal properties could not be measured with sufficient accuracy (due, for example, to interfering noises, distance from the microphone, etc.).

45. Functional classification may be influenced by subjectivity and/or inconsistency. In Rühlemann & Dingemanse (in preparation), to ensure coding reliability, the data were coded independently by two researchers and the amount of inter-rater agreement was determined using Cohen's kappa.

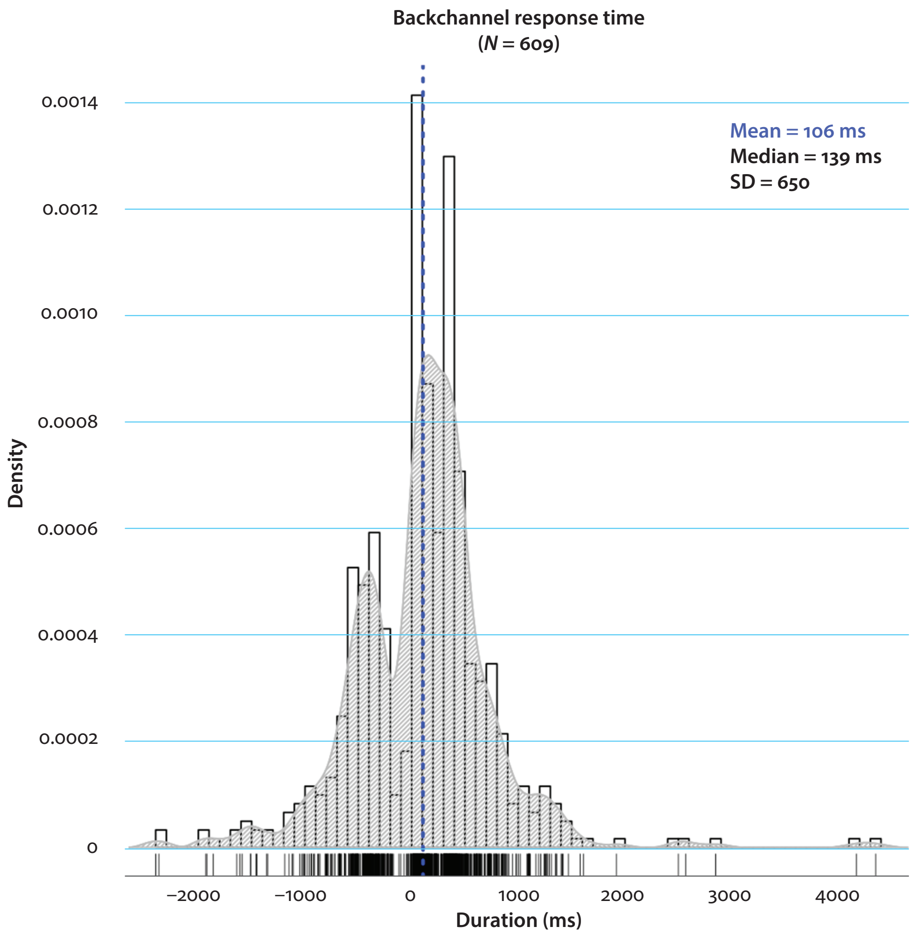


Figure 12.1 Backchannel (BC) response time in the Narrative Corpus

The histogram in Figure 12.1 provides a number of useful pieces of information. First, we see large 'tails' both to the left, where negative response times for backchannels in overlap are plotted, and, even more so, to the right, where positive response times for backchannels occurring after gaps are plotted; note, for example, the two backchannels with response times longer than 4 seconds, indicated by the two rug stripes at the extreme end of the duration axis. Even if we disregard these extreme values, the distribution is slightly asymmetrically divided between overlap and gap: 199 values, or 32.68%, are greater than 0, meaning there are more free-standing backchannels than overlapped ones; cf. the proportions of overlap in turn-by-turn talk, which range between 44% (ten Bosch et al. 2005) 40% (Heldner & Edlund 2010), and 30% (Levinson & Torreira 2015).

Second, the tallest bin is for response times between 0 ms and 100 ms. This indicates that the largest group of response times fall into this range. The modal response times for backchannels, then, are between 0–100 ms – exactly half of the mode reported in Stivers et al. (2009) for answers to polar questions. Second, the mean value is 106 ms, while the median response time is 139 ms – again clearly shorter values of central tendency than the respective values reported in previous research. Third, the distribution violates normality.[46] This transpires clearly from the sharp drop of the density curve between −400 ms to 0 ms. This, together with the large Standard Deviation, suggests that treating backchannels as a unified group is only a first step, as there is likely a lot of internal functional variation.

To approach this functional variation, let's first check the frequencies and proportions of the functional categories coded in the subsample:

Table 12.1 Frequencies and proportions of functional categories in subsample

	Continuers	Convergence tokens	Assessments	Recompleters	Other	Unclear
Freq	140	62	43	3	36	16
%	46.67	20.67	14.33	1.00	12.00	5.33

In the 300-token sample, the most common functional category are continuers, with 140 examples and 46.67%. The next two largest groups are convergence tokens with 20.67% and assessments accounting for 14.33%. Recompleters, with only 3 instances, are fairly infrequent. Another group, of considerable size, are backchannels classed as "Other", a label used for response tokens that clearly served to register listenership but whose function did not match any of the four categories continuers, convergence tokens, assessments, or recompleters. This suggests that the functional spectrum of backchannels may be even more comprehensive than is commonly acknowledged.

The following plot focuses on the response times for the three major classes continuers, convergence tokens, and assessments, showing their histogram, densities, means and standard deviations:

46. Data is considered normally distributed if it spreads symmetrically around a central value, thus forming a bell curve (cf. Levshina 2015: 10). Other distributions include the uniform distribution, the bimodal distribution, the Poisson distribution, to name only a few (cf. Crawley 2007: 210 ff.).

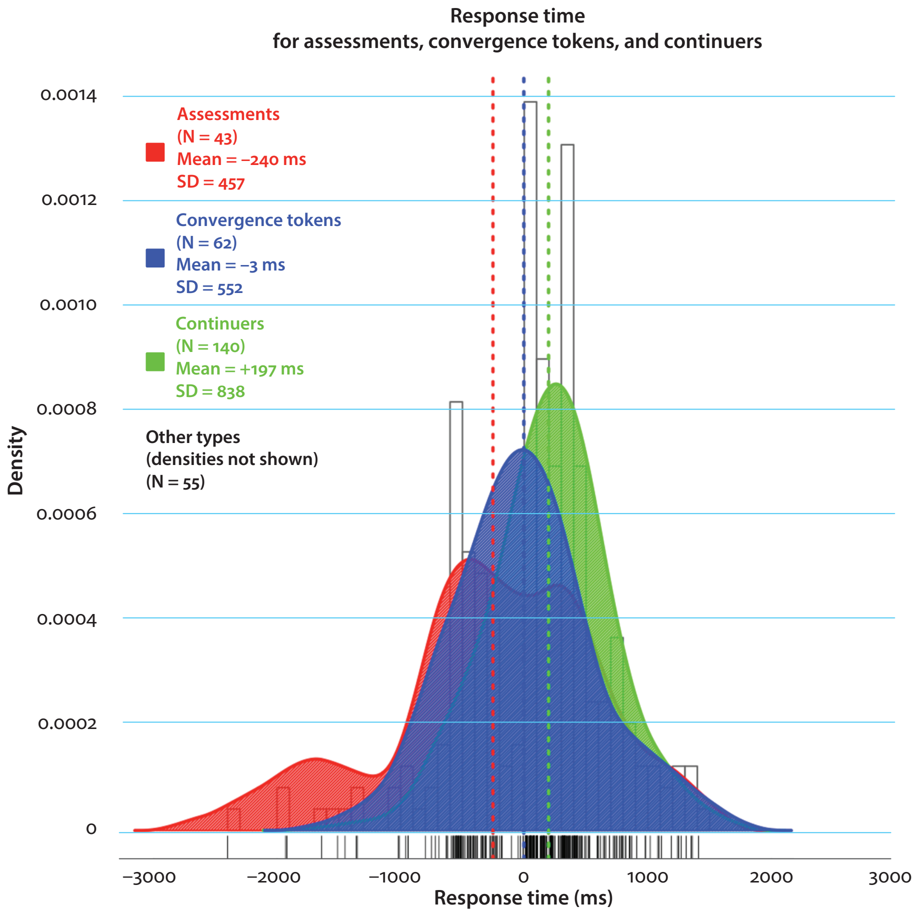


Figure 12.2 Response times for continuers, convergence tokens, and assessments

As can be seen in Figure 12.2, the three functional types are clearly distinguished in terms of response time: assessments typically occur in overlap (mean response time = −240 ms), continuers normally follow after a slight gap (mean response time = 104 ms), whereas convergence tokens tend to occupy the middle ground (mean response time = −3 ms). While the density curves for continuers and convergence tokens somewhat approximate the bell shape typical of normal distribution, the curve for assessments clearly suggests nonnormality, with its primary maximum on the overlap side but two secondary maxima, one in overlap and the other on the right of the 0 ms demarcation. The irregularity of response times for assessments can also be read off the Standard Deviation: it is almost twice as large as the SD values for continuers and convergence tokens.

12.1.4 Discussion and concluding remarks

The histograms and density plots shown above suggest that the response time for backchannels generally is shorter than the 200 ms observed in research so far (see above): the average response time was 106 ms, a value closely mirroring previous response times obtained from experimental research (Wesseling & van Son 2005a, 2005b), where the measured interval was 102 ms.

The main reason for this speediness is probably reduced production latency. This reduction is facilitated by multiple factors. First, backchannels do not express concepts and do not have lexical content (Müller 1996: 132); that is, production time for backchannels is reduced by the 150–200 ms (Indefrey & Levelt 2004: 104) it takes to retrieve, for example, the concept SHEEP as well as up to 150 ms spent on lemma retrieval (for a content word such as 'sheep') (Indefrey & Levelt 2004: 106). Second, backchannels are phonologically minimal, mostly just a single syllable long or, in the case of 'mm', just a single phoneme. The minimalness means that retrieval time for instructions how to articulate them will be equally minimal, thus further reducing production latency. Third, as mentioned, backchannels are among the most frequent words in conversation. Frequency too impacts on production time; high-frequency words are produced up to 60 ms faster than low-frequency words (Indefrey & Levelt 2004: 106). Forth, many backchannels, especially the most frequent, are non-verbal – 'noises', as Carter & McCarthy (1997: 12) call them. As we know from Stivers et al. (2009), nonverbal responses such as head shakes, nods, or shrugs are produced "faster than speech" (Stivers et al. 2009: 10588). So non-verbal backchannels are likely faster too. Fifth, backchannels may not require inbreath for articulation. As shown by Torreira et al. (2015), speakers preparing to answer a question breath differently depending on the length of the response: long responses are preceded by long inbreaths of up to 800 ms, whereas minimal responses are produced on residual breath (i.e., without inbreath). Due to their minimalness, backchannels may not require inbreath either.

The case study also showed important differences in response time between functional categories, with continuers typically occurring after a slight gap, convergence tokens requiring no gap and no overlap, and assessment tokens tending to be produced early, that is, during the current turn. That continuers typically occur without, and assessments within, overlap is to be expected given previous research. For example, Goodwin noted that continuers occur *between* the speaker's turn-constructional units, whereas assessments occur *within* the units, that is, in overlap with them (Goodwin 1986). Also, speaking of assessment sequences, Pomerantz observes that "strong or upgraded assessments are performed with a minimization of gap (in fact, frequently in slight overlap)" (Pomerantz 1984: 69; cf. also Vatanen 2018).

As noted previously, in storytelling, which can be conceptualized as "an activity that both takes a stance toward what is being reported and makes the taking of a stance by the recipient relevant" (Stivers 2008: 32), the critical point for the recipient to produce an assessment backchannel is the story Climax. Stivers noted that assessment backchannels ('affiliative tokens' in her terminology) such as 'ah', 'wow', 'oh my god', are relevant responses "as tellers near the high point of the telling" (Stivers 2013: 201) to display the recipient's agreement with the teller's stance toward the recounted events (Stivers 2008, 2013; see also Schegloff 1993: 106).

For illustration, consider extract (12.2), the closing part of an extended storytelling by Betty about a friend of hers who forgets time and again to put her lights on while driving. The constructed dialog in lines 1–2 "STUPID COW you ain';t got your fucking lights on again have you? IT'S THAT SWITCH THERE", which is ascribed to the friend's husband, indicates that the telling is approaching the Climax (cf. Mayes 1990). The Climax is finally reached in Betty's description that the friend did "the self same thing" (line 4), namely forget to put her lights on while pulling "into the shop out of the shop no lights" (line 6). Julie's backchannel "[↑oh][:↓:: my: ↑go:::d]" is in double overlap: it is in TCU-terminal overlap with Betty's "the next day she did the self same thing" and in TCU-initial overlap with Betty's next TCU "pull into the shop out of the shop" (cf. Rühlemann 2018b). Thus, the backchannel stretches over the whole Climax. Note also the backchannel's phonetic realization: its extreme lengthening and pitch movement indicate heavy affect – the affect of being shocked by the dangerous forgetfulness of Betty's friend.

```
(12.2)
1  Betty:  ( ) STUPID COW you ain';t got your fucking lights on again have you?
2          IT'S THAT SWITCH THERE,
3          °so she puts her lights on° ( ) same day.
4          the next day she did the self same [thing]
5  Julie:                                     [↑oh ][:↓::       my:       ↑go:::d]
6  Betty:                                          [pull into the shop out of the shop]
7          no lights.
8          (0.41)
                                   (BNC: KBE 1827-1896; corrected transcription)
```

In many other contexts, extended overlaps such as the one in extract (12.2), might be considered interruptions and require mechanisms for repair (cf. Schegloff 2000). Here, by contrast, the fact that two speakers talk in unison is not only tolerated but sought and valued, as an external manifestation of the 'emotional contagion' (Doherty 1997; Peräkylä et al. 2015) that is, ultimately, the raison d'être for stories in talk-in-interaction.

12.2 The histogram and density plot in the case study

How were the histogram and density plot in Figure 12.2 arrived at?

First, read-in the data, available on the companion website as "Chapter12_Casestudy.txt":

```
> BC <- read.table("[Your path]/Chapter12_Casestudy.txt", header=T,
quote="", sep="\t", fill=F)
```

As can be seen from calling `head(BC)`, there are just three variables: (i) `bc` containing the backchannel forms in CA transcription, (ii) `responsetime`, the response times measured in ms, and `fun`, the functional classifications:

```
> head(BC)
       bc  responsetime      fun
1  ah yes           730 continue
2      hm           276 continue
3      mm           370 continue
4   ↑mm:↓           882    other
5    yeah           460 continue
6      mm           650 continue
```

As always, we check the data structure with `str()`:

```
> str(BC)
'data.frame': 300 obs. of 3 variables:
$ bc : Factor w/ 114 levels "(did she)","(did they)",..: 37 48 59 …
$ responsetime: int 730 276 370 882 460 650 250 1165 450 910 …
$ fun : Factor w/ 6 levels "assess","continue",..: 2 2 2 4 2 2 2 5 …
```

`str(BC)` reveals that `responsetime` contains integers and the variables `bc` and `fun` are factors. If we worked with the elements in `bc` we would probably have to convert the variable to character. But in the case study the relevant variables are just `responsetime` and `fun`, so we are fine with the dataframe's data structure.

We are, then, in a luxurious position: no further adjustments are necessary, we can start plotting the response times right away.

As always, we begin by setting the plotting layout:

```
> par(mfrow = c(1,1), mar = c(4.1,4,2.7,0.5))
```

Next, we set up the histogram. As we want to plot only the response times for the three major classes assessment, continuer, and convergence tokens, we need to exclude the response times of the backchannels classified otherwise, namely `"recomplete"`, `"other"`, or `"unclear"`. We do so by subsetting `BC` and using the negative operator

`!` as well as the ampersand `&`. The title, defined in `main`, is quite long, so we include `\n` to insert a line break. Further, the font size of the title, axes, and axis labels is reduced in `cex.main`, `cex.axis`, and `cex.lab`. An important argument is `freq`. It defaults to `TRUE`, but needs to be set to `FALSE`; this is required as we intend to plot densities, which have a completely different scale on the y-axis. The argument `border` controls the color of the border delineating the histogram's bins. As we suspect that some of the densities will cover a large range, we extend the range of the x-axis with `xlim = c(-3000, 3000)`. Finally, the label for the x-axis is included with `xlab = "Response time (ms)"`:

```
> hist(BC$responsetime[!BC$fun=="recomplete" &
+                      !BC$fun=="other" &
+                      !BC$fun=="unclear"],
+      main = "Response time\nfor assessments, convergence tokens,
              and continuers",
+      cex.main = 0.8,
+      cex.axis = 0.7,
+      cex.lab = 0.8,
+      breaks = 50,
+      freq = F,
+      border = "grey40",
+      xlim = c(-3000, 3000),
+      xlab = "Response time (ms)")
```

To see the exact location of the individual response times, we include a 'rug', using the function `rug()`, where the argument `side = 1` determines the placement of the rug at the x-axis (use `side = 2` for the y-axis and so on clockwise):

```
> rug(BC$responsetime[!BC$fun=="recomplete"
+                     & !BC$fun=="other"
+                     & !BC$fun=="unclear"], side = 1)
```

Now for the main part, the density plots for the three functional classes continuers, convergence tokens, and assessments. The key function is `density()`, the function to compute kernel density estimates, a data smoothing procedure used in inferential statistics (that is, that sub-field of statistics that aims to make inferences from a sample of a population to that population itself). The function's main arguments are `kernel` and `bw` (for bandwith) (cf. `?density`). The argument `kernel` specifies the type of smoothing, for example, `"gaussian"` (the default), `"epanechnikov"`, or `"cosine"`. The argument `bw` is the smoothing parameter. Smoothing is an inferential procedure intended to abstract from the known (sample) data to the unknown (population) data. As there is inevitably variation and deviation from the 'true' distribution in the sample data, an important question is how much we should abstract away from that variation. If too much is filtered out, the result is oversmoothing, which hides underlying structure; if too little is filtered out, the result is undersmoothing, with too many spurious artifacts distracting from the underlying structure. Obviously, the ideal smoothing parameter neither undersmoothes nor oversmoothes. Choosing the right parameter is a difficult task. In this case study, it is chosen on a trial and error basis, by testing different parameters and comparing the resulting estimates.

After mutiple such trials, the kernel selected for Figure 12.2 was `"cosine"` and the bandwidth parameter `250`. With these decisions made, we could now plot both the density lines and shade the area under the lines for each functional level separately. For example, we could use this code for response times of backchannels coded as continuers:

```
> lines(density(BC$responsetime[BC$fun=="continue"],
+               kernel = "cosine", bw = 250),
+       col = "green",
+       lwd = 1)
```

Here, we make use of the function `lines()` wrapped around `density()` and target that subset of `BC$responsetime` that satisfies the criterion that the value on `BC$fun` is`"continue"`.

To shade the area underneath the density line we can wrap the `polygon()` function around the `density()` function and apply the same subsetting logic as above:

```
> polygon(density(BC$responsetime[BC$fun=="continue"],
+                 kernel = "cosine", bw = 250),
+         col = "green",
+         density = 50,
+         angle = 90)
```

And we could repeat these commands for assessments and convergence tokens. But a more elegant way is to execute it all in one piece of code, namely in a `for` loop (not dissimilar to the `for` loop used in Section 8.2). First we access the three function levels of interest in `BC$fun`; as the levels are ordered alphabetically, these happen to be the first three so that we can subset `levels(BC$fun)` on those three:

```
> cats <- levels(BC$fun)[1:3]
> cats
[1] "assess" "continue" "converge"
```

Next, we define a vector `cols` containing a color for each of the three levels:

```
> cols <- c("red", "green", "blue")
```

Now we set up the `for` loop: `for(i in seq(cats))`. Here, as `cats` is a character vector we use `seq()` to access the vector's indices, which are numeric.Inside the `for` loop we embed two commands, one for the density line using `lines()`, one for the shaded area using `polygon()`. All we really have to change compared to the code for each functional level plotted separately (discussed above), is that we specify `BC$fun` with `i` for `cats` and that we specify the colors via the (same) `i` for `cols`. The rest, including the `density()` arguments `kernel`, `bw`, and `lty` as well as the `polygon()` arguments `density` and `angle`, remains the same:

```
> for(i in seq(cats)){
+   lines(density(BC$responsetime[BC$fun==cats[i]],
+                  kernel="cosine", bw = 250),
+         col = cols[i],
+         lwd = 2)
+   polygon(density(BC$responsetime[BC$fun==cats[i]],
+                    kernel="cosine", bw = 250),
+           col = cols[i],
+           density = 75,
+           angle = 45)
+ }
```

Figure 12.2 also displays lines for the mean response times of the three function levels. To draw them we can recycle the vectors `cats` and `cols` in a new `for` loop. This loop is also `for(i in seq(cats))` but embodies the instruction to use the `abline()` function to draw a straight vertical line to represent the mean response time of each function type:

```
> for(i in seq(cats)){
+ abline(v = mean(BC$responsetime[BC$fun==cats[i]]),
+ col = cols[i], lty = 3, lwd = 2)
+ }
```

Finally, we include a legend. As we want to display the number of instances per function, the means and the Standard Deviations, we first compute these measures, using `tapply()`:

```
> tapply(BC$responsetime, BC$fun, FUN = length)
     assess  continue  converge  other  recomplete  unclear
         43       140        62     36           3       16

> tapply(BC$responsetime, BC$fun, FUN = mean)
    assess    continue  converge      other  recomplete    unclear
-253.093023 196.514286 -3.645161 224.138889 2671.666667 22.500000

> tapply(BC$responsetime, BC$fun, FUN = sd)
     assess  continue  converge    other  recomplete  unclear
   837.5415 457.5367  552.4383 430.1897  1599.4244 582.3701
```

From this output we can copy the values of interest into the legend text. We insert the legend `"topleft"`, that is, at a predefined location in the left upper corner of the plotting region. Next, the legend text is defined in `c()`, by using the information gained beforehand (the number of instances per function, the mean, the SD); note we place each bit of information separately on a new line by inserting `\n` and we include empty elements `""` to separate the texts blocks. The argument `cex` specifies the sought character extension (i.e. font size), `text.col` specifies the colors for the legend texts (note the use of `"white"` for the empty text elements). Further, in `fill` we define the colors for the little squares to the left of the legend texts (note again the color `"white"` for the empty text elements); `border = "white"` sets the line around the squares to white and `bty = "n"` suppresses the default box around the legend as a whole:

```
> legend("topleft",
+         c("Assessments\n(N = 43)\nMean = -240 ms\nSD = 457",
+           "",
+           "Convergence tokens\n(N = 62)\nMean = -3 ms\nSD = 552",
+           "",
+           "Continuers\n(N = 140)\nMean = +197 ms\nSD = 838",
+           "",
+           "Other types\n(densities not shown)\n(N = 55)"),
+         cex = 0.75,
+         text.col = c("red", "white", "blue", "white", "green",
                      "white", "black"),
+         fill = c("red", "white", "blue", "white", "green", "white",
                  "white"),
+         border = "white",
+         bty = "n")
```

12.3 Task: Response time for laughter

Let's end this volume with laughter.

Laughter is an inherently responsive form of conduct in interaction. Among the environments in which laughter is a *relevant* form of response, storytelling figures high (Schegloff 1993: 104) – as does another speaker's laughter: as Jefferson (1979) has shown, speakers rarely laugh alone but share laughter with others in overlap. Laughter is treated as an activity that is "*not* to be done serially, not one *after* the other, but to be done simultaneously" (Schegloff 2000: 6; emphasis in original).

So laughter triggers simultaneous laughter. But does laughter also occur simultaneously with *talk*? That is, does (the onset of) laughter precede the completion of the turn or TCU to which it is a response?

In this task, you will be able to address this question.

On the companion website you find data from the Narrative Corpus: response times of laughter produced by story recipients; the file is called "Chapter12_Task.txt".; the response times are stored in column `laughter_rt`.

Use a histogram and a density plot to visualize the distribution of laughter response times across the overlap/gap spectrum; also calculate and visualize central tendencies, such as the mean response time and the Standard Deviation.

The steps to code the plot include the following:

- plot a histogram for response times with a suitable number of bins
- plot a density line with a suitable kernel and bandwidth
- shade the area under the density line using the `polygon()` function
- calculate the mean
- plot the mean into the graphic
- include a legend

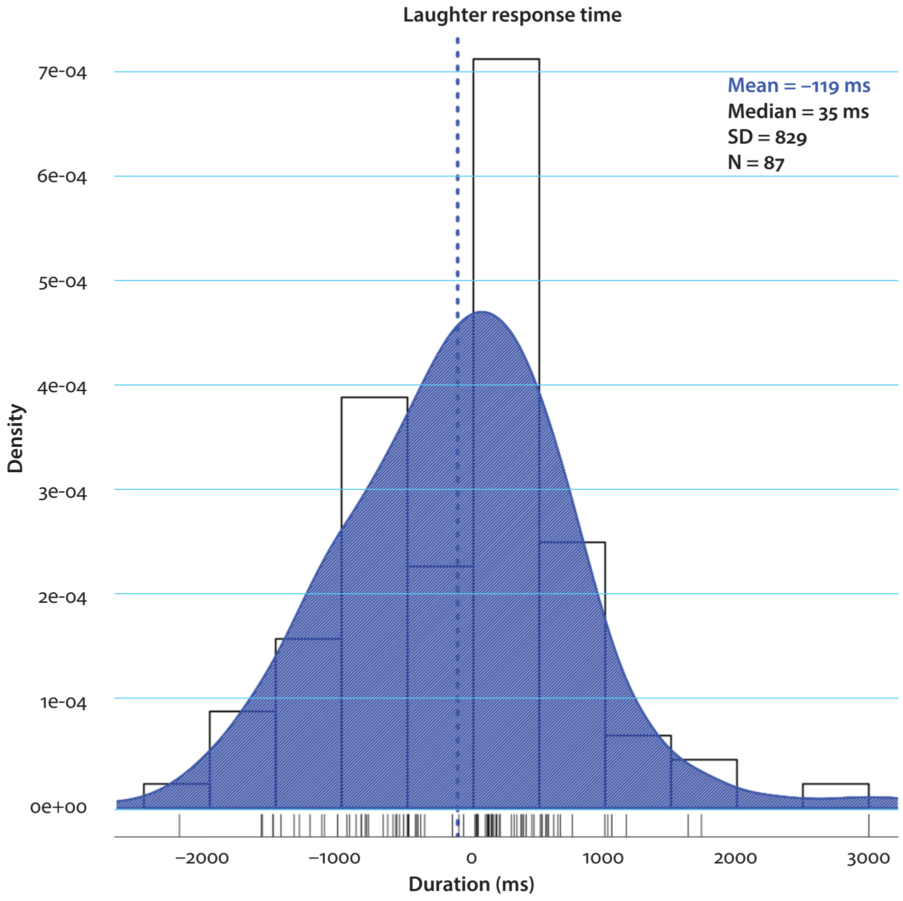


Figure 12.3 Response time for laughter in storytelling

Depending on the graphical parameters as well as the kernel and the bandwith chosen by you, your plot will be similar to the one shown in Figure 12.3.

As shown in Figure 12.3, the central tendency for laughter in storytelling is to occur in overlap: the mean response time is −119 ms. However, as indicated by the wide spread of the density curve and the high SD value (given in the legend), there is considerable variation in how laughter is timed vis-à-vis the speaker's TCU. In a serious analysis, that variation would warrant close case-by-case analysis, with a special emphasis on the most deviant response times, indicated in the rugs on the x-axis: this deviant-case analysis would certainly have to include the one story where laughter is delayed by 3 seconds and also thoses cases where laughter occurs after a (more than) 1-second gap.

Given that laughter is a nonverbal vocalization, the central tendency of laughter to occur in overlap aligns well with research indicating that nonverbal responses are quicker than verbal responses (Stivers et al. 2009: 10588). As far as laughter is concerned the explanation is readily at hand: unlike lexical words encoding concepts, laughter does not have lexical meaning. As a result, all the retrieval and articulation processes for lexical words briefly described in Section 12.1.4 do not apply. If we accept the premise that laughter, one of the "most inescapably *responsive* forms of conduct" (Schegloff 1993: 104; added emphasis), indexes, among other things, understanding, laughter may be among the responses that are temporally most closely associated with comprehension: we laugh *as soon as* we have understood the laughable TCU/turn. Its occurrence prior to TCU/turn completion suggests that that understanding has been reached before the turn's/TCU's end point has been reached. Thus, the fact that laughter is typically in overlap conclusively supports the notion that the precision timing of turn transition in conversation is indeed due to successful anticipation and prediction.

Chapter 13

Concluding remarks

We have reached the end of our journey through turn and sequence. All along the way R has been our travel guide. What has R shown us? Let's take stock of the most salient lessons taught.

13.1 Single-case analysis and visualization

Both Interactional Linguistics and Conversation Analysis stress the importance of careful single-case analysis (e.g., Schegloff 1993). This would seem to prevent any utility for visualization – for what to show visually if there is just a single case to report? This is the point where the different approaches of Interactional Linguistics and Conversation Analysis to the 'talk' component in talk-in-interaction come into play. As noted earlier, Conversation Analysis has no large interest in language whereas Interactional Linguistics, as a subfield of *linguistics*, has (Couper-Kuhlen & Selting 2018). It can thus take advantage of the large array of linguistic techniques to penetrate into microscopic details of talk in interaction. A visual representation of these micro-details can be insightful too if it is coupled to rigorous sequential analysis. For example, as we have seen in Section 9.1, a single storytelling of roughly half a minute duration yielded many thousands of phonetic data points on the scales of pitch and intensity. We combined the phonetic analysis with a sequential analysis of the storytelling focusing on its progression toward the Climax, depicted the relation of the phonetic data and the sequential data in a scatter plot, and *saw* that pitch and intensity peaked where the telling peaked. In Section 9.3, by contrast, in a related analysis, we saw that the high-point was reached prior to the phonetic maxima. Subsequent scrutiny showed that the disalignment was due to a lack of recipient affiliation and the teller's pursuit of affiliation in post-Climax position. So, visualization may benefit even single-case analyses provided quantifiable linguistic detail is taken into account.

13.2 Regular expression and transcription

Analyses in Interactional Linguistics and Conversation Analysis are typically based on rigorous transcription conventions, typically in accordance with the Jeffersonian system (e.g., Jefferson 2004). A drawback of the Jeffersonian transcript is that it is not machine-readable per se. As long as the method of choice is the single-case analysis or analysis of a few select transcripts, this drawback does not make itself felt. The analyst only starts to sense a feeling of missingness as soon as the aim is to get a handle on more, potentially many more, instances of a feature of interest. As has been shown in Chapters 2 and 10, R's implementation of regular expression offers a solution: regular expressions allow the analyst to define any – simple or complex – feature of interest as a *pattern* and to match the pattern in dozens, hundreds, or thousands of transcripts – the number of transcripts to be processed by regex is unbounded. Thus, regular expressions open up transcripts – elaborated based on rigorous attention to interactionally relevant detail and thus true to methodological principles – to large-scale quantitative analysis.

13.3 Integrating quantitative and qualitative analysis

Quantification is by no means a fool-proof method: just because we can count something doesn't mean the count is meaningful *per se*. As Schegloff demonstrates compellingly, investigating laughter per minute is meaningless unless the 'environments of possible relevant occurrence' of laughter are taken into account (Schegloff 1993: 103).[47] Quantitative analysis is therefore not an alternative to qualitative analysis but a complement to it. There are two ways to make quantification fruitful for qualification, and the other way round. One is to start with rigorous qualitative analysis of, potentially many, single cases and build quantification on its back (Schegloff 1993: 102). That's the methodological route we took, for example, in Chapter 10: first the uses of paralinguistic prosody and the occurrences of constructed dialog were identified in Jeffersonian transcripts based on case-by-case analysis; then the quantitative analysis was carried out testing whether expressive prosody was significantly associated with its occurrence inside and outside constructed dialog in storytelling. Alternatively, one can start with the 'areal view' facilitated by corpus analysis, looking for quantifiable patterns in large amounts of data, and in a second step return to 'the ground', that is, to single-case analysis, to examine canonical and also deviant cases that buck the trend discerned from 'above'. That's the route taken, for example, in Chapter 6: first we investigated surprisal in the context of the nuclear tone in turns, discovering a (surprising) correlation of the two, then we examined deviant cases, where the nuclear tone had been placed

47. As noted in Section 12.3, laughter's 'environments of possible relevant occurrence' include stories and another speaker's laughter; alternatively, as noted by Schegloff (1993), laughter may be relevant in potential offenses (Jefferson et al. 1987) or in telling one's own troubles (Jefferson 1984).

on a less surprising word. Or, to cite another example, the analysis in Section 12.3 suggested that the central tendency for laughter in storytelling was to occur far into the storyteller's as-yet-incomplete TCU. But it was also shown that the spread of response times was remarkable including no small number of positive response times, that is, cases where laughter occurred with substantial delay. Investigating these deviant cases in due sequential detail would be a 'must' in any serious examination of laughter in storytelling.

13.4 Multimodal enhancement

Recent research into multimodality suggests that "gesture and speech form an integrated system in language comprehension" (Kelly et al. 2010: 260) and that multimodally enhanced communication facilitates information processing. This is evidenced by faster reaction times to combinations of speech and iconic gestures compared to speech-only (e.g., Holler et al. 2018). Unlike other types of gestures, such as deictic pointing, iconic gestures are "closely linked to the semantic contents of the speech that they accompany" (Holler et al. 2009: 74) by *depicting* aspects of what a speaker is verbally referring to (Kendon 1986). However, (gestural) depiction and (verbal) description do not convey exactly the same information; rather a complimentary relationship obtains (Kendon 1986: 12; Dingemanse & Akita 2017: 519): iconic gestures provide "a visual representation of aspects of content that are not referred to in the verbal component of the utterance" (Kendon 1986: 13). A large number of studies have addressed the question as to which of these gesture-specific meanings contribute most to effective communication (for references, see Holler et al. 2009). It appears that iconic gestures are particularly effective at communicating information "about the *size* of objects and the *relative position* of objects with respect to one another" (Holler et al. 2009: 74, emphasis in original).

Visualizations of data in academic texts are the equivalent of iconic gestures in speech: the information they convey is, obviously, closely linked to the description in the text and yet they convey more than the words and they do it in a dfferent way. Information is expressed differently because words organize it in a temporal linear order, whereas visualizations, just as iconic gestures, represent it "'all at once' or 'globally'" (Kendon 1986: 9). What 'more' information visualizations convey is related to size and position. Barcharts, for example, depict size differences of count data in exact detail whereas descriptions resort to relative quantification such as 'more than' or 'less than' and vague qualifications such as 'slightly' or 'massively'. Size as a graphical parameter is used not only for frequency differences but also to depict differences of continuous variables. For example, as seen in Chapter 7, dendrograms translate a distance matrix into branches of different sizes, with the size differences indicating different degrees of dissimilarity. Position too is a key element in data visualization – just not physically locational position that we refer to in speech but metaphorical position. For example, positions in turns or sequences metaphorically refer to time points: the word in initial position in a turn is the temporally first word to occur, 'adnominal this' occurs 'early' in storytellings, while assessment tokens appear 'late' in the story. Just how early and, respectively, late is shown precisely, with the whole range of gradience laid out before the observer, for example, in stripcharts and violinplots, whereas words, restricted to discrete categories (such as 'early' or 'late'), fail to convey the same precision.

So, visualizations in research excel at the same semantic categories as iconic gestures in speech. And, as iconic gestures, visualizations boost comprehension thus contributing to the research report's communicative effectiveness.

13.5 To conclude

This book set out to demonstrate and teach the use of R, quantification, and visualization for research on talk-in-interaction in conversation. Given this, perhaps unorthodox, combination, it may be the first of its kind. Given the combination's considerable potential, it may not, and hopefully will not, be the last.

References

Aijmer, K. 2013. *Understanding Pragmatic Markers. A Variational Pragmatic Approach*. Edinburgh: EUP.

Aijmer, K. & Rühlemann, C. 2015. *Corpus Pragmatics. A Handbook*. Cambridge: CUP. https://doi.org/10.1017/CBO9781139057493

Ameka, F. 1992. Interjections: The universal yet neglected part of speech. *Journal of Pragmatics* 18: 101–118. https://doi.org/10.1016/0378-2166(92)90048-G

Amerman, J. D. & Parnell, M. M. 1992. Speech timing strategies in elderly adults. *Journal of Phonetics* 20: 65–76. https://doi.org/10.1016/S0095-4470(19)30254-2

Atkinson, J. M. 1984. Public speaking and audience response: Some techniques for inviting applause. In *Structures of Social Action: Studies in Conversation Analysis*, J. M. Atkinson & J. Heritage (eds), 370–409. Cambridge: CUP.

Bavelas, J. B., Coates, L. & Johnson, T. 2000. Listeners as co-narrators. *Journal of Personality and Social Psychology* 79: 941–952. https://doi.org/10.1037/0022-3514.79.6.941

Bavelas, J.B., Coates, L. & Johnson, T. 2002. Listener responses as a collaborative process: The role of gaze. *Journal of Communication* 52: 566–580.

Baynham, M. 2003. Narratives in space and time: Beyond 'backdrop' accounts of narrative orientation. *Narrative Inquiry* 13(2): 347–366. https://doi.org/10.1075/ni.13.2.07bay

Beattie, G., Cutler, A. & Pearson, M. 1982. Why is Mrs. *Thatcher interrupted so often? Nature* 300: 744–747.

Biber, D., Johansson, S., Leech, G., Conrad, S. & Finegan, E. 1999. *Longman Grammar of Spoken and Written English*. Harlow: Pearson Education.

Blackwell, N. L., Perlman, M. & Fox Tree, M. E. 2015. Quotation as multi-modal construction. *Journal of Pragmatics* 81: 1–7. https://doi.org/10.1016/j.pragma.2015.03.004

Boersma, P. 2013. Acoustic analysis. In *Research Methods in Linguistics*, R. J. Podesta & D. Sharma (eds), 375–396. Cambridge: CUP.

Boersma, P. & Weenink, D. 2012. *Praat: Doing Phonetics by Computer* [Computer program]. <http://www.praat.org/>

Bögels, S. & Torreira, F. 2015. Listeners use intonational phrase boundaries to project turn ends in spoken interaction. *Journal of Phonetics* 52: 46–57. https://doi.org/10.1016/j.wocn.2015.04.004

Bögels, S., Kendrick, K. H. & Levinson, S. C. 2015. Never say no … How the brain interprets the pregnant pause in conversation. *PLoS ONE* 10(12): e0145474. https://doi.org/10.1371/journal.pone0145474

Bolden, G. 2004. The quote and beyond: Defining boundaries of reported speech in conversational Russian. *Journal of Pragmatics* 36: 1071–1118. https://doi.org/10.1016/j.pragma.2003.10.015

ten Bosch, L., Oostdijk, N. & Boves, L. 2005. On temporal aspects of turn taking in conversational dialogues. *Speech Communication* 47: 80–86. https://doi.org/10.1016/j.specom.2005.05.009

Brinton, L. J. 2010. Discourse markers. In *Historical Pragmatics* [Handbooks of Pragmatics 8], A. H. Jucker & I. Taavitsainen (eds), 285–314. Berlin: D. Gruyter Mouton.

Brown, P. &. Levinson, S.C. 1987. *Politeness. Some Universals in Language Usage*. Cambridge: CUP.

Buysse, L. 2012. *So* as a multifunctional discourse marker in native and learner speech. *Journal of Pragmatics* 44: 1764–1782. j.pragma.2012.08.012. https://doi.org/10.1016/j.pragma.2012.08.012

Carter, R. 2004. *Language and Creativity. The Art of Common Talk*. Abingdon: Routledge.

Carter, R. A., Hughes, R. & McCarthy, M.J. 2000. *Exploring Grammar in Context*. Cambridge: CUP.

Carter, R. & McCarthy, M. 1997. *Exploring Spoken English*. Cambridge: CUP.

Clark, H. H. & Gerrig, R. J. 1990. Quotations as demonstrations. *Language* 66(4): 764–805. https://doi.org/10.2307/414729

Clayman, S. E. 2013. Turn-constructional units and the transition-relevance place. In Sidnell & Stivers (eds), 150–166.

Cleveland, W. S. 1979. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association* 74: 829–836. https://doi.org/10.1080/01621459.1979.10481038

Coleman, J., Baghai-Ravary, L., Pybus, J. & Grau, S. 2012. *Audio BNC. The Audio Edition of the Spoken British National Corpus*. Phonetics Laboratory, University of Oxford. <http://www.phon.ox.ac.uk/AudioBNC> (29 April 2020).

Coulmas, F. 1985. Direct and indirect speech: General problems and problems of Japanese. Journal of Pragmatics 9: 41–63.

Couper-Kuhlen, E. 2012. Exploring affiliation in the reception of conversational complaint stories. In *Emotion in Interaction*, A. Peräkylä & M.-L. Sorjonen (eds), 113–146. Oxford: OUP. https://doi.org/10.1093/acprof:oso/9780199730735.003.0006

Couper-Kuhlen, E. & Selting, M. 2018. *Interactional Linguistics. Studying Language in Social Interaction*. Cambridge: CUP.

Crawley, M. J. 2007. *The R book*. Chichester: John Wiley & Sons.

Crystal, D. 2003. *A Dictionary of Linguistics and Phonetics*, 5th edn. Oxford: Blackwell.

Culpeper, J. 2011. *Impoliteness: Using Language to Cause Offence*. Cambridge: CUP. https://doi.org/10.1017/CBO9780511975752

Dalgaard, P. 2008. *Introductory Statistics with R*, 2nd edn. Berlin: Springer. https://doi.org/10.1007/978-0-387-79054-1

Diani, G. 2004. The discourse functions of I don't know in English conversation. In *Discourse Patterns in Spoken and Written Corpora* [Pragmatics & Beyond New Series 120], K. Aijmer & B. Stenström (eds), 157–171. Amsterdam: John Benjamins. https://doi.org/10.1075/pbns.120.11dia

Dingemanse, M. & Enfield, N. J. 2014. Let's talk: Universal social rules underlie languages. *Scientific American Mind*, 25: 64–69. https://doi.org/10.1038/scientificamericanmind0914-64

Dingemanse, M. & Akita, K. 2017. An inverse relation between expressiveness and grammatical integration: On the morphosyntactic typology of ideophones, with special reference to Japanese. *Journal of Linguistics* 53(3): 501–532. https://doi.org/10.1017/S002222671600030X

Dingemanse M., Roberts, S.G., Baranova, J., Blythe, J., Drew, P., Floyd, S., Gisladottir, S.R., Kendrick, K.H., Levinson, S.C., Manrique, E., Rossi, G. & Enfield, N.J. 2015. Universal principles in the repair of communication problems. *PLoS ONE* 10(9): e0136100. https://doi.org/10.1371/journal.pone.0136100

Dingemanse, M., Rossi, G. & Floyd, S. 2017. Place reference in story beginnings: A cross-linguistic study of narrative and interactional affordances. *Language in Society* 46(2): 129–158. https://doi.org/10.1017/S0047404516001019

Doherty, W. 1997. The emotional contagion scale: A measure of individual differences. *Journal of Nonverbal Behavior* 21(2): 131–154. https://doi.org/10.1023/A:1024956003661

Drew, P. 1998. Complaints about transgressions and misconduct. *Research on Language and Social Interaction* 31: 295–325. https://doi.org/10.1080/08351813.1998.9683595

Drew, P. 2013. Turn design. In Sidnell & Stivers (eds), 131–149.

Duncan, S. 1972. Some signals and rules for taking speaking turns in conversations. *Journal of Personality and Social Psychology* 23: 283–292. https://doi.org/10.1037/h0033031

Duncan, S. 1974. On the structure of speaker-auditor interaction during speaking turns. *Language in Society* 3: 161–180. https://doi.org/10.1017/S0047404500004322

Enfield, N.J. 2017. *How we Talk. The Inner Workings of Conversation*. New York NY: Basic Books.

Ervin-Tripp, S. M. & Küntay, A. 1997. The occasioning and structure of conversational stories. In *Conversation: Cognitive, Communicative and Social Perspectives* [Typological Studies in Language 34], T. Givón (ed.),133–166. Amsterdam: John Benjamins. https://doi.org/10.1075/tsl.34.06erv

Fraser, B. 1990. An approach to discourse markers. *Journal of Pragmatics* 14: 383–395. https://doi.org/10.1016/0378-2166(90)90096-V

Gardner, R. 1998. Between speaking and listening: The vocalisation of understandings. *Applied Linguistics* 19(2): 204–224. https://doi.org/10.1093/applin/19.2.204

Goffman, E. 1981. *Forms of Talk*. Philadelphia PA: University of Pennsylvania Press.

Goodwin, C. 1984. Notes on story structure and the organization of participation. In *Structures of Social Action: Studies in Conversation Analysis*, J. M. Atkinson & J. Heritage (eds), 225–246. Cambridge: CUP

Goodwin, C. 1986. Between and within alternative sequential treatments of continuers and assessments. *Human Studies* 9: 205–217. https://doi.org/10.1007/BF00148127

Goodwin, C. & Heritage, J. 1990. Conversation analysis. *Annual Review of Anthropology* 19: 283–307. https://doi.org/10.1146/annurev.an.19.100190.001435

Golato, A. 2000. An innovative German quotative for reporting embodied actions: Und ich so/und er so 'and I'm like/and he's like'. *Journal of Pragmatics* 32: 29–54. https://doi.org/10.1016/S0378-2166(99)00030-2

Gravano, A. & Hirschberg, J. 2011. Turn-taking cues in task-oriented dialogue. *Computer Speech and Language* 25(3): 601–634. https://doi.org/10.1016/j.csl.2010.10.003

Gries, S.T. 2009. *Statistics for Linguistics with R*. Berlin: De Gruyter.

Gries, S.T. 2017. *Quantitative Corpus Linguistics with R. A Practical Introduction*. London: Routledge.

Gumperz, J. J. 1996. The linguistic and cultural relativity of inference. In *Rethinking Linguistic Relativity*, J. J. Gumperz & S. C. Levinson (eds), 347–406. Cambridge: CUP.

Halliday, M. A. K. & Hasan, R. 1976. *Cohesion in English*. London: Longman.

Halliday, M. A. K. & Matthiessen, M. I. M. 2004. *A. Introduction to Functional Grammar*, 2nd edn. London: Edward Arnold.

Haselow, A. 2019. Discourse marker sequences: Insights into the serial order of communicative tasks in real-time turn production. *Journal of Pragmatics* 146: 1–18. https://doi.org/10.1016/j.pragma.2019.04.003

Haugh, M. & Musgrave, S. 2019. Conversational lapses and laughter: Towards a combinatorial approach to building collections in conversation analysis. *Journal of Pragmatics* 143: 279–291. https://doi.org/10.1016/j.pragma.2018.09.005

Hatfield, E., Cacioppo, J. & Rapson, R. 1994. *Emotional Contagion*. Cambridge: CUP.

Heldner, M. 2011. Detection thresholds for gaps, overlaps, and no-gaps-no-overlap. *Journal of the Acoustical Society of America* 130(1): 508–513.

Heldner, M. & Edlund, J. 2010. Pauses, gaps and overlaps in conversations. *Journal of Phonetics* 38: 555–568. https://doi.org/10.1016/j.wocn.2010.08.002

Heritage, J. 1984. A change-of-state token and aspects of its sequential placement. In *Structures of Social Action*, J. Atkinson & J. Heritage (eds), 299–345. Cambridge: CUP.

Heritage. J. 1998. Oh-prefaced responses to inquiry. *Language in Society* 27(3): 291–334.

Heritage, J. 1999. Conversation analysis at century's end: Practices of talk-in-Interaction, their distributions, and their outcomes. *Research on Language & Social Interaction* 32(1–2): 69–76. https://doi.org/10.1080/08351813.1999.9683609

Heritage. J. 2013. Turn-initial position and some of its occupants. *Journal of Pragmatics* 57: 331–337. https://doi.org/10.1016/j.pragma.2013.08.025

Heritage, J. 2015. Well-prefaced turns in English conversation: A conversation analytic perspective. *Journal of Pragmatics* 88: 88–104. https://doi.org/10.1016/j.pragma.2015.08.008

Heritage, J. 2018. Turn-initial particles in English: The cases of of and well. In *Between Turn and Sequence. Turn-initial Particles across Languages* [Studies in Language and Social Interaction, 31], J. Heritage & M.L. Sorjonen (eds), 155–189. Amsterdam: John Benjamins.

Heritage, J. & Sorjonen, M.L. 2018. Analyzing turn-initial particles. In *Between Turn and Sequence. Turn-initial Particles across Languages* [Studies in Language and Social Interaction, 31], J. Heritage & M.L. Sorjonen (eds), 1–22. Amsterdam: John Benjamins.

Hilpert, M. 2011. Dynamic visualizations of language change. Motion charts on the basis of bivariate and multivariate data from diachronic corpora. *International Journal of Corpus Linguistics* 16(4): 435–461.

Hoey, M. 2005. *Lexical Priming. A New Theory of Words and Language*. London: Routledge

Hoey, E. M. 2015. Lapses: How people arrive at, and deal with, discontinuities in talk. *Research on Language and Social Interaction* 48(4): 430–453. https://doi.org/10.1080/08351813.2015.1090116

Hoey, E. M. 2017. Sequence recompletion: A practice for managing lapses in conversation. *Journal of Pragmatics* 109: 47–63. https://doi.org/10.1016/j.pragma.2016.12.008

Hoffmann, S., Evert, S., Smith, N., Lee, D. & Berglund Prytz, Y. 2008. *Corpus Linguistics with BNC-web – A Practical Guide*. Frankfurt: Peter Lang.

Holler, J. & Levinson, S. C. 2019. Multimodal language processing in human communication. *Trends in Cognitive Sciences* 23(8): 639–652. https://doi.org/10.1016/j.tics.2019.05.006

Holler, J., Kendrick, K. H. & Levinson, S. C. 2018. Processing language in face-to-face conversation: Questions with gestures get faster responses. *Psychonometric Bulletin Review* 25: 1900–1918. https://doi.org/10.3758/s13423-017-1363-z

Holler, J., Shovelton, H. & Beattie, G. 2009. Do iconic gestures really contribute to the semantic information communicated in face-to-face interaction? *Journal of Nonverbal Behavior* 33: 73–88. https://doi.org/10.1007/s10919-008-0063-9

Hömke, P., Holler, J. & Levinson, S. C. 2017. Eye blinking as addressee feedback in face-to-face conversation. *Research on Language and Social Interaction* 50(1): 54–70. https://doi.org/10.1080/08351813.2017.1262143

Holt, E. 1996. Reporting talk: The use of direct reported speech in conversation. *Research on Language and Social Interaction* 29(3): 219–245. https://doi.org/10.1207/s15327973rlsi2903_2

Holt, E. 2000. Reporting and reacting: Concurrent responses to reported speech. *Research on Language and Social Interaction* 33(4): 425–454. https://doi.org/10.1207/S15327973RLSI3304_04

Holt, E. 2007. 'I'm eying your chop up mind': Reporting and enacting. In *Reporting Talk. Reported Speech in Interaction*, E. Holt & R. Clift (eds), 47–80. Cambridge: CUP.

Indefrey, P. & Levelt, W. J. M. 2004. The spatial and temporal signatures of word production components. *Cognition* 92: 101–144. https://doi.org/10.1016/j.cognition.2002.06.001

Jaeger, T. F. 2010. Redundancy and reduction: Speakers manage syntactic information density. *Cognitive Psychology* 61(1): 23–62. https://doi.org/10.1016/j.cogpsych.2010.02.002

Jakobson, R. 1960. Linguistics and poetics. In *Style in Language*, T. A. Sebeok (ed.), 351–377. Cambridge, MA: The MI. Press.

Jefferson, G. 1973. A case of precision timing in ordinary conversation: Overlapped tag-positioned address terms in closing sequences. *Semiotics* 9: 47–96.

Jefferson, G. 1978. Sequential aspects of storytelling in conversation. In *Studies in the Organization of Conversatonal Interaction*, J. Schenkein (ed.), 219–248. New York NY: Academic Press.

Jefferson, G. 1979. A technique for inviting laughter and its subsequent acceptance declination. In *Everyday Language – Studies in Ethnomethodology*, G. Psathas (ed.), 79–95. New York NY: Irvington Publishers.

Jefferson, G. 1984. On the organization of laughter in talk about troubles. In *Structures of Social Action: Studies in Conversation Analysis*, J. M. Atkinson & J. Heritage (ed.), 191–222. Cambridge: CUP.

Jefferson, G. 1986. Notes on 'latency' in overlap onset. *Human Studies* 9: 153–183. https://doi.org/10.1007/BF00148125

Jefferson, G. 2004. Glossary of transcript symbols with an introduction. In *Conversation Analysis. Studies from the First Generation*, G. H. Lerner (ed.), 13–31. Amsterdam: John Benjamins. https://doi.org/10.1075/pbns.125.02jef

Jefferson, G., Sacks, H. & Schegloff, E. A. 1987. Notes on laughter in the pursuit of intimacy, G. Button & J. R. E. Lee (eds), 152–205. Clevedon: Multilingual Matters.

Jescheniak, J. D. & Levelt, W. J. M. 1994. Word frequency effects in speech production: Retrieval of syntactic information and of phonological form. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 20: 824–843.

Johnson, K. 2013. *Quantitative Methods in Linguistics*. Malden MA: Blackwell.

Kelly, S. D., Özyürek, A. & Maris, E. 2010. Two sides of the same coin: Speech and gesture mutually interact to enhance comprehension. *Psychological Science* 21(2): 260–267. https://doi.org/10.1177/0956797609357327

Kaltenböck, G. 2015. Processibility. In *Corpus Pragmatics: A Handbook*, K. Aijmer & C. Rühlemann (eds.), 117–142. Cambridge: CUP.

Kendon, A. 1967. Some functions of gaze-direction in social interaction. *Acta Psychologica* 26: 22–63. https://doi.org/10.1016/0001-6918(67)90005-4

Kendon, A. 1986. Some reasons for studying gesture. *Semiotica* 62(1–2): 3–28.

Kendrick, K. & Drew, P. 2016. Recruitment: Offers, requests, and the organization of assistance in interaction. *Research on Language and Social Interaction* 49(1): 1–19. https://doi.org/10.1080/08351813.2016.1126436

Koester, A. & Handford, M. 2018. It's not good saying "Well it it might do that or it might not"': Hypothetical reported speech in business meetings. *Journal of Pragmatics* 130: 67–80. https://doi.org/10.1016/j.pragma.2018.03.005

Labov, W. 1972. *Language in the Inner City*. Oxford: Basil Blackwell.

Labov, W. & Waletzky. J. 1967/1997. Narrative analysis: Oral versions of personal experience. In *Essays on the Verbal and Visual Arts*, J. Helms (ed.), 12–44. Seattle WA: University of Washington Press. Reprinted in *Journal of Narrative Inquiry and Life History* 7(1–4): 3–38.

Leech, G. 1983. *Principles of Pragmatics*. London: Longman.

Levelt, W. J., Roelofs, A. & Meyer, A. S. 1999. A theory of lexical access in speech production. *Behavioral and Brain Sciences* 22(1): 1–38. https://doi.org/10.1017/S0140525X99001776

Levinson, S. C. 1983. *Pragmatics*. Cambridge: CUP. https://doi.org/10.1017/CBO9780511813313

Levinson, S. C. 2013. Action formation and ascription. In Sidnell & Stivers (eds), 103–130.

Levinson, S. C. 2016. Turn-taking in human communication–Origins and implications for language processing. *Trends in Cognitive Sciences* 20(1): 6–14. https://doi.org/10.1016/j.tics.2015.10.010

Levinson S. C. & Holler, J. 2014. The origin of human multi-modal communication. *Philosophical Transactions of the Royal Society B* 369: 20130302. https://doi.org/10.1098/rstb.2013.0302

Levinson, S. C. & Torreira, F. 2015. Timing in turn-taking and its implications for processing models of language. *Frontiers in Psychology* 6: 731. https://doi.org/10.3389/fpsyg.2015.00731

Levshina, N. 2015. *How to do Linguistics with R. Data Exploration and Statistical Analysis*. Amsterdam: John Benjamins. https://doi.org/10.1075/z.195

Liebenthal, E., Silbersweig, D. A. & Stern, E. 2016. The language, tone and prosody of emotions: Neural substrates and dynamics of spoken-word emotion perception. *Frontiers of Neuroscience* 10: 506. https://doi.org/10.3389/fnins.2016.00506

Liddicoat, A. J. 2007. *A. Introduction to Conversation Analysis*. London: Continuum.

Local, J., & Walker, G. 2012. How phonetic features project more talk. *Journal of the International Phonetic Association* 42: 255–280. https://doi.org/10.1017/S0025100312000187

Magyari, L., Bastiaansen, M. C. M., de Ruiter, J. P. & Levinson, S. C. 2014. Early anticipation lies behind the speed of response in conversation. *Journal of Cognitive Neuroscience* 26(11): 2530–2539. https://doi.org/10.1162/jocn_a_00673

Mathis, T. & Yule, G. 1994. Zero quotatives. *Discourse Processes* 18(1): 63–76. https://doi.org/10.1080/01638539409544884

Mayes, P. 1990. Quotation in spoken English. *Studies in Language* 14: 325–363. https://doi.org/10.1075/sl.14.2.04may

McCarthy, M. 2003. Talking back: 'Small' interactional response tokens in everyday conversation. *Research on Language and Social Interaction* 36(1): 33–63. https://doi.org/10.1207/S15327973RLSI3601_3

McIntyre, D., Bellard Thomson, C., Heywood, J., McEnery, T., Semino, E. & Short, M. 2004. Investigating the presentation of speech, writing and thought in spoken British English: A corpus-based approach. *ICAME Journal* 28: 49–76.

Mehl, M. R., Vazire, S., Ramirez-Esparza, N., Slatcher, R. B. & Pennebaker, J. W. 2007. Are women really more talkative than men? *Science* 317: 82. https://doi.org/10.1126/science.1139940

Müller, F. E. 1996. Affiliating and disaffiliating with continuers: prosodic aspects of recipiency. In *Prosody in Conversation*, E. Couper-Kuhlen & M. Selting (eds), 131–176. Cambridge: CUP. https://doi.org/10.1017/CBO9780511597862.006

Norrick, N. 2009. Interjections as pragmatic markers. *Journal of Pragmatics* 41: 866–891.

Ochs, E., Schegloff, E.A. & Thompson, S.A. (eds). 1996. *Interaction and Grammar*. Cambridge: CUP.

O'Donnell, M. B., Scott, M., Mahlberg, M. & Hoey, M. 2012. Exploring text-initial words, clusters and concgrams in a newspaper corpus. *Corpus Linguistics and Linguistic Theory* 8(1): 73–101.

Ogden, R. 2001. Turn transition, creak and glottal stop in Finnish talk-in-interaction. *Journal of the International Phonetic Association* 31(1): 139–152. https://doi.org/10.1017/S0025100301001116

O'Keeffe, A. & Adolphs, S. 2008. Response tokens in British and Irish discourse. Corpus, context and variational pragmatics. In *Variational Pragmatics* [Pragmatics & Beyond New Series 178], K. P. Schneider & A. Barron (eds), 69–98. Amsterdam: John Benjamins. https://doi.org/10.1075/pbns.178.05ok

Peräkylä, A., Henttonen, P., Voutilainen, L., Kahri, M., Stevanovic, M., Sams, M. & Ravaja, N. 2015. Sharing the emotional load: Recipient affiliation calms down the storyteller. *Social Psychology Quarterly* 78(4): 301–323. https://doi.org/10.1177/0190272515611054

Pomerantz, A. M. 1984. Agreeing and disagreeing with assessments: Some features of preferred/dispreferred turn shapes. In *Structures of Social Action: Studies in Conversation Analysis*, J. M. Atkinson & J. Heritage (eds), 57–101. Cambridge: CUP.

Pomerantz, A. & Heritage, J. 2013. Preference. In Sidnell & Stivers (eds), 210–228.

Prince, E. F. 1981. Toward a taxonomy of given/new information. In *Radical Pragmatics*, P. Cole (ed.), 223–255. New York, NY: Academic Press.

Quirk, R., Greenbaum, S., Leech, G. & Svartvik, J. 1985. *A Comprehensive Grammar of the English Language*. London: Longman.

Rayson, P., Leech, G. & Hodges. M. 1997. Social differentiation in the use of English vocabulary: Some analyses of the conversational component of the British National Corpus. *International Journal of Corpus Linguistics* 2(1): 133–152. https://doi.org/10.1075/ijcl.2.1.07ray

Riest, C., Jorschnik, A.B. & de Ruiter, J. P. 2015. Anticipation in turn-taking: Mechanisms and information sources. *Frontiers in Psychology* 6: 1–14. https://doi.org/10.3389/fpsyg.2015.00089

Riggenbach, H., 1991. Toward an understanding of fluency: A microanalysis of nonnative speaker conversations. *Discourse Processes* 14: 423–441. https://doi.org/10.1080/01638539109544795

Rochemont, M. & Cullicover, P. 1990. *English Focus Constructions and the Theory of Grammar*. Cambridge: CUP.

Roberts, S. G. & Levinson, S. C. 2017. Conversation, cognition and cultural evolution: A model of the cultural evolution of word order through pressures imposed from turn takingin conversation. *Interaction Studies* 18(3): 402–429. https://doi.org/10.1075/is.18.3.06rob

Roberts, S. G., Torreira, F. & Levinson, S. C. 2015. The effects of processing and sequence organization on the timing of turn taking: A corpus study. *Frontiers of Psychology* 6: 509. https://doi.org/10.3389/fpsyg.2015.00509

Romero-Trillo, J. 2018. Prosodic modeling and position analysis of pragmatic markers in English conversation. *Corpus Linguistics and Linguistic Theory*.

Rühlemann, C. 2013. *Narrative in English Conversation. A Corpus Analysis*. Cambridge: CUP. https://doi.org/10.1017/CBO9781139026987

Rühlemann, C. 2017. Integrating corpus-linguistic and conversation-analytic transcription in XML. The case of backchannels and overlap in storytelling interaction. *Corpus Pragmatics* 1(3): 201–232. https://doi.org/10.1007/s41701-017-0018-7

Rühlemann, C. 2018a. *Corpus Linguistics for Pragmatics*. London: Routledge. https://doi.org/10.4324/9780429451072

Rühlemann, C. 2018b. TCU-initial backchannel overlap in storytelling. *Narrative Inquiry* 28(2): 257–279. https://doi.org/10.1075/ni.17060.ruh

Rühlemann, C. 2018c. How long does it take to say 'well'? Evidence from the Audio BNC. *Corpus Pragmatics* 3(1): 49–66. https://doi.org/10.1007/s41701-018-0046-y

Rühlemann, C. Forthcoming a. What dialog is absent from constructed dialog? *English Text Construction*.

Rühlemann, C. Forthcoming b. Turn structure and inserts. *International Journal of Corpus Linguistics*.

Rühlemann, C. & Dingemanse, M. In preparation. Response time for backchannels.

Rühlemann, C. & Gee, M. 2017. Conversation analysis and the XML method. *Gesprächsforschung* 18: 274–296.

Rühlemann, C. & Gries, S. T. 2020. Speakers advance-project turn completion by slowing down – A multifactorial corpus study.

Rühlemann, C. & Hilpert, M. 2017. Colloquialization in journalistic writing: Investigating inserts in TIME magazine with a focus on well. *Journal of Historical Pragmatics* 18(1): 102–135. https://doi.org/10.1075/jhp.18.1.05ruh

Rühlemann, C. & O'Donnell, M.B. 2012. Introducing a corpus of conversational narratives. Construction and annotation of the Narrative Corpus. *Corpus Linguistics and Linguistic Theory* 8(2): 313–350.

Rühlemann, C. & Schweinberger, M. In preparation. Nucleus placement – A multifactorial corpus analysis.

Rühlemann, C., Bagoutdinov, A. & O'Donnell, M. B. 2015. Modest XPath and XQuery for corpora: Exploiting deep XML annotation. *ICAM. Journal* 39: 47–84. https://doi.org/10.1515/icame-2015-0003

de Ruiter, J. P., Mitterer, H. & Enfield, N. J. 2006. Projecting the end of a speaker's turn: A cognitive cornerstone of conversation. *Language* 82(3): 515–535. https://doi.org/10.1353/lan.2006.0130

Sacks, H. 1984. Notes on methodology. In *Structures of Social Action*, J. M. Atkinson & J. Heritage (eds), 21–27. Cambridge: CUP.

Sacks, H. 1987. On the preference for agreement and contiguity in sequences in conversation. In *Talk and Social Organisation*, G. Button & J. R. E. Lee (eds). *Clevendon: Multilingual Matters*.

Sacks, H. 1992. *Lectures on Conversation*, Vols. I & II. Oxford: Blackwell.

Sacks, H., Schegloff, E. A. & Jefferson, G. 1974. A simplest systematics for the organisation of turn-taking for conversation. *Language* 50(4): 696–735. https://doi.org/10.1353/lan.1974.0010

Scott, M. & Tribble, C. 2006. *Textual Patterns: Key Words and Corpus Analysis in Language Education* [Studies in Corpus Linguistics 22]. Amsterdam: John Benjamins. https://doi.org/10.1075/scl.22

Schegloff, E. A. 1982. Discourse as an interactional achievement: Some uses of 'uh huh' and other things that come between sentences. In *Georgetown University Round Table on Languages and Linguistics Analyzing Discourse: Text and Talk*, D. Tannen (ed.), 71–93. Washington DC: Georgetown University Press.

Schegloff, E. A. 1993. Reflections on quantification in the study of conversation. *Research on Language and Social Interaction* 26: 99–128.

Schegloff, E. A. 1996. Turn organization: One intersection of grammar and interaction. In *Interaction and Grammar*, E. Ochs, E. A. Schegloff & S. A. Thompson (eds), 53–133. Cambridge: CUP.

Schegloff, E. A. 2000. Overlapping talk and the organization of turn-taking for conversation. *Language in Society* 29: 1–63. https://doi.org/10.1017/S0047404500001019

Schegloff, E. A. 2001. Discourse as an interactional achievement III: The omnirelevance of action. In *The Handbook of Discourse Analysis*, D. Schiffrin, D. Tannen & H. E. Hamilton (eds), 229–249. Oxford: Blackwell.

Schegloff, E. A. 2007. *Sequence Organisation in Interaction: A Primer in Conversation-Analysis*. Cambridge: CUP. https://doi.org/10.1017/CBO9780511791208

Schegloff, E. A. & Lerner, G. H. 2009. Beginning to respond: "well"-prefaced responses to WH-questions. *Research on Language and Social Interaction* 42(2): 91–115. https://doi.org/10.1080/08351810902864511

Schiffrin, D. 1985. Conversational coherence: The role of well. *Language* 61: 640–667.

Schiffrin, D. 1987. *Discourse Markers*. Cambridge: CUP.

Schleppgrell, M. 1991. Paratactic because. *Journal of Pragmatics* 16: 323–337. https://doi.org/10.1016/0378-2166(91)90085-C

Schourup, L. 2001. Rethinking well. *Journal of Pragmatics* 33: 1025–1060. https://doi.org/10.1016/S0378-2166(00)00053-9

Searle, J. R. 1975. Indirect speech acts. In *Syntax and Semantics III*, P. Cole & J. L. Morgan (eds), 59–82. New York NY: Academic Press.

Sidnell, J. 2006. Coordinating gesture, talk, and gaze in reenactments. *Research on Language and Social Interaction* 39(4): 377–409. https://doi.org/10.1207/s15327973rlsi3904_2

Sinclair, J. M. 2000. Lexical grammar. *Naujoji Metodologija* 24: 191–203.

Soulaimani, D. 2018. Talk, voice and gestures in reported speech: Toward an integrated approach. *Discourse Studies* 20(3): 361–376. https://doi.org/10.1177/1461445618754419

Stec, K., Huiskes, M. & Redeker, G. 2016. Multimodal quotation: Role shift practices in spoken narratives. *Journal of Pragmatics* 104: 1–17. https://doi.org/10.1016/j.pragma.2016.07.008

Stivers, T. 2008. Stance, alignment, and affiliation during storytelling: When nodding is a token of affiliation. *Research on Language and Social Interaction* 41(1): 31–57. https://doi.org/10.1080/08351810701691123

Stivers, T. 2013. Sequence organization. In Sidnell & Stivers (eds), 191–209.

Stivers, T. 2015. Coding social interaction: A heretical approach in Conversation Analysis? *Research on Language and Social Interaction* 48(1): 1–19. https://doi.org/10.1080/08351813.2015.993837

Stivers, T. & Robinson, J. D. 2006. A preference for progressivity in interaction. *Language in Society* 35: 367–392. https://doi.org/10.1017/S0047404506060179

Stivers, T. & Sidnell, T. 2013. Introduction. In Sidnell & Stivers (eds), 1–8.

Stivers, T., Enfield, N. J., Brown, P., Englert, C., Hayashi, M., Heinemann, T., Hoymann, G., Rossano, F., de Ruiter, J. P., Yoon, K.-E. & Levinson, S. C. 2009. Universals and cultural variation in turn-taking in conversation. *Proceedings of the National Academy of the Sciences U.S.A.* 106(26): 10587–10592. https://doi.org/10.1073/pnas.0903616106

Stokoe, E. 2018. *Talk: The Science of Conversation*. London: Robinson.

Stubbs, M. 2002. *Words and Phrases. Corpus Studies of Lexical Semantics*. Malden MA: Blackwell.

Stubblebine, T. 2007. *Regular Expression*, 2nd edn. Sebastopol CA: O'Reilly.

Tognini Bonelli, E. 2010. The evolution of corpus linguistics. In *The Routledge Handbook of Corpus Linguistics*, A. O'Keeffe & M. McCarthy (eds), 14–27. London: Routledge. https://doi.org/10.4324/9780203856949.ch2

Torreira, F., Bögels, S. & Levinson, S. C. 2015. Breathing for answering: The time course of response planning in conversation. *Frontiers in Psychology*. https://doi.org/10.3389/fpsyg.2015.00284

Vachek, J. (ed.). 1964. *A Prague School Reader*. Bloomington, IN: Indiana University Press.

Vatanen, A. 2018. Responding in early overlap: Recognitional onsets in assertion sequences. *Research on Language and Social Interaction*. https://doi.org/10.1080/08351813.2018.1413894

Wade, E. & Clark, H. H. 1993. Reproduction and demonstration in quotations. *Journal of Memory and Language* 32(6): 805–819. https://doi.org/10.1006/jmla.1993.1040

Wald, B. 1983. Referents and topic within and across discourse units: Observations from current vernacular English. In *Discourse Perspectives on Syntax*, F. Klein-Andreu (ed.), 91–116. New York NY: Academic Press.

Wells, B. & Macfarlane, S. 1998. Prosody as an interactional resource: Turn-projection and overlap. *Language and Speech* 41(3–4): 265–294. https://doi.org/10.1177/002383099804100403

Wennerstrom, A. 2001. *The Music of Everyday Speech. Prosody and Discourse Analysis*. Oxford: OUP.

Wesseling, W. & van Son, R. J. J. H. 2005. Timing of experimentally elicited minimal responses as quantitative evidence for the use of intonation in projecting trps. *Interspeech* 6: 3389–3392.

White, S. 1989. Backchannels across cultures: A study of Americans and Japanese. *Language in Society* 18(1): 59–76. https://doi.org/10.1017/S0047404500013270

Wong, D. & Peters, P. 2007. A study of backchannels in regional varieties of English, using corpus mark-up as the means of identification. *International Journal of Corpus Linguistics* 12(4): 479–509. https://doi.org/10.1075/ijcl.12.4.03won

Woods, A., Fletcher, P. & Hughes A. 1986. *Statistics in Language Studies*. Cambridge: CUP.

Yngve, V. 1970. On getting a word in edgewise. In *Papers from the Sixth Regional Meeting of the Chicago Linguistic Society*, Robert I. Binnick (ed.), 567–77. Chicago IL: CLS.

Yuan, J., Liberman, M. & Cieri, C. 2006. Towards an integrated understanding of speaking rate in conversation. *Interspeech* 2006.

Zipf, G. K. 1949. *Human Behavior and the Principle of Least Effort. A. Introduction to Human Ecology*. Cambridge, MA: Addison-Wesley Press.

Function index

Conceptual index

This book is a textbook on R, a programming language and environment for statistical analysis and visualization. Its primary aim is to introduce R as a research instrument in quantitative Interactional Linguistics. Focusing on *visualization* in R, the book presents original case studies on conversational talk-in-interaction based on corpus data and explains in good detail how key graphs in the case studies were programmed in R. It also includes task sections to enable readers to conduct their own research and compute their own visualizations in R. Both the code underlying the key graphs in the case studies and the datasets used in the case studies as well as in the task sections are made available on the book's companion website.

ISBN 978 90 272 0710 4

9 789027 207104

John Benjamins Publishing Company