QUANTITATIVE TEXT ANALYSIS USING R

Scraping, Preparing, Visualising and Modelling Data

JULIAN BERNAUER

ANNA WOHLMANN

Sage

1 Oliver's Yard
55 City Road
London EC1Y 1SP

2455 Teller Road
Thousand Oaks
California 91320

Unit No 323-333, Third Floor, F-Block
International Trade Tower
Nehru Place, New Delhi – 110 019

8 Marina View Suite 43-053
Asia Square Tower 1
Singapore 018960

Editor: Jai Seaman
Editorial assistant: Becky Oliver
Production editor: Ian Antcliff
Cover design: Shaun Mercier
Typeset by: C&M Digitals (P) Ltd, Chennai, India
Printed in the UK

Library of Congress Control Number: 2024948533

British Library Cataloguing in Publication data

A catalogue record for this book is available from the British Library

ISBN 978-1-5264-6701-0
ISBN 978-1-5264-6700-3 (pbk)

CONTENTS

PREFACE

This book can be directly traced back to the 2015 European Political Science Association's conference in Vienna. There, one of the authors was incidentially approached by a Sage representative and had a conversation about his potential writing plans. It took another few years, until 2017, when that author started a job in the Data and Methods Unit of the Mannheim Centre of European Social Research (MZES) and found the business card he got back then at the conference, contacted Sage and inquired about potentially writing a book on quantitative text analysis (QTA). Charmingly, this initiative actually translated into a book contract.

After receiving a contract offer from Sage in early 2018, in an impressively swift and professional process managed by the publisher, the book still had a bumpy road to take. Julian Bernauer had envisioned teaching a course on QTA at the Graduate School of Economics and Social Sciences (GESS) in Mannheim, and writing the chapters at the same time as teaching them. This looked like a clever idea then but was rather naïve. Not only could he not finish chapter drafts at such a high rate, but also life happened. A third child was welcomed to his family, everybody had to move to another town to align home and job, only to be greeted by a pandemic taking over most of 2020 to 2022 which made adjusting to the new environments more difficult.

The incredible professionalism, patience and leadership, especially of Sage's Jai Seaman, as well as her team, has kept the book project alive over these periods. What finally brought about a decisive turn towards finishing the manuscript was the start of the co-authorship with Anna Wohlmann, in early 2023. Long story short, we're tremendously happy that the book project has survived these circumstances and that so many people at Sage and in our personal and professional environments provided their support and, especially, their patience. We hope this book is of some use to some people.

ACKNOWLEDGEMENTS

This textbook is a classic example of the phrase 'standing on the shoulders of giants'. Substantially, it only exists because of the efforts by people such as Kenneth Benoit, Will Lowe and Julia Silge. We would especially like to credit Kenneth Benoit and his team for developing the `quanteda` R package (https://quanteda.io), which took QTA to new standards even beyond the scientific community. We merely await their innovations.

The book's materialisation has been tremendously supported by the great professionals at Sage, and especially Jai Seaman (supported by Charlotte Bush, Rhiannon Holt and Becky Oliver), who have both managed and endured the whole process excellently. The contribution of Sage to the success of this book project has been immense and decisive. Thank you very much, and one last time, sincere apologies for the delay. All remaining errors are ours.

The authors thank the case study contributors for bringing life and diversity to the book with their excellent contributions.

Julian Bernauer especially thanks Thomas Bräuninger and Marc Debus for sparking his interest in QTA, Adrian Vatter for his mentoring, the participants of the 2019 Mannheim GESS QTA course class for providing feedback on the very early drafts of the book, Anna Wohlmann for her decisive help in getting the book over the finishing line, the MZES, and especially Denis Cohen, Alejandro Ecker, Valentin Kalaev and Philipp Heldmann, for the stimulating and enabling work environment, and his family for everything else. Anna Wohlmann thanks Marius Sältzer for establishing initial contact with R and QTA, Julian Bernauer for his trust and advise, the lecturers who encouraged the research path, and finally Anna's family and friends, especially Eva, Frank and Maya, for being the best cheerleaders.

ABOUT THE AUTHORS

Julian Bernauer's interest in QTA started in 2006, when he wrote a master's thesis supervised by Prof Thomas Bräuninger at the University of Konstanz (where he studied politics and public management) on analysing speeches by members of the German parliament. The topic of both this research and his doctoral studies at the University of Konstanz (finished in 2012) was political representation of different sorts, and he moved on to lecture and study comparative political institutions as a postdoctoral researcher and lecturer (*Oberassistent*) at the University of Bern. Since 2017, he has been a member of the Mannheim Centre of European Social Research (MZES), first in the Data and Methods Unit, and since 2020 as permanent scientific staff and researcher in the institute's computer department. He is also involved in the management of the MZES.

Anna Wohlmann is a doctoral candidate at the Technical University of Munich, holding a Master's degree in Politics & Technology from the Technical University of Munich and a Bachelor's degree in Political Science with a minor in Psychology from the University of Mannheim. When introduced to QTA with R in a bachelor-seminar, Anna immediately knew this would be their method of choice. The analysis of social media content created by social movements is the main focus of Anna's research.

3

TEXT AS DATA: OBTAINING, PREPARING AND CLEANING

The chapter describes three examples of how to get text into R (using party manifestos, Wikipedia articles and song lyrics), with an emphasis on scraping via APIs, along with some data cleaning and descriptive analysis of various text sources. We also discuss ethics and privacy in web scraping in an interlude within this chapter. A case study by Carlos Gueiros on the analysis of YouTube captions complements the gallery of examples handling text data.

Contents

- How to get text files into R and clean them
- Retrieving Wikipedia data
- Using social media data and its special characters
- Extracting song lyrics through an API
- Interlude: The ethics of web scraping and data privacy
- Case study (Carlos Gueiros): Labelling political topics with transformer models using YouTube text data

NO TEXT ANALYSIS WITHOUT TEXT

Chapter 2 provided an introduction to QTA using R, employing Kendrick Lamar's lyrics as an example. The current chapter builds upon previous discussions and widens and deepens the scope for treating text as data. A major barrier to performing QTA – or any type of statistical analysis, for that matter – is getting the data ready and into the system. Using text as data aggravates the hurdles posed, as text takes up more storage and comes with elements in various formats, for example, social media data with its unique format and special characters. With a few tricks, though, we can efficiently obtain text, avoid errors, isolate and clean the parts of interest, and turn them into meaningful data. Hence what you will learn in this chapter is the preparation of text data.

By the end of this chapter, you will have learned about three ways to obtain, clean and prepare data based on the running examples of the book. First, we go back to a classic use case of QTA: party manifestos (see, for instance, Laver, Benoit and Garry 2003). We read a selection of the party manifestos of European parties from the political documents archive polidoc.net into R (also available via shiny.mzes.uni-mannheim.de/polidoc), stored as plain UTF-8 text files. Based on this example, the basic `readtext` and `quanteda` functions are discussed in more detail. Next, we introduce a way to download Wikipedia data. This is followed by preprocessing of social media data, where we use data from what then had been Twitter at the time of data collection with the #fridaysforfuture hashtag as an example. The text is cleaned of elements such as URLs and emoticons, and the most frequent words are extracted. Lastly, the rap lyrics example from Chapter 2 is revisited to enable us to dig deeper into the use of a proper API and web scraping. Data from the website genius.com is converted into R objects. We also touch upon web scraping methods and related ethical concerns.

Throughout this section, several info boxes provide explanations of topics that might be of interest to some readers. These include data frames, loops, the `View()` function, the notorious problem of character encoding in text analysis (and computing in general) and the even more notorious regex (regular expressions) used to find arbitrarily complex patterns in text.

READING IN PARTY MANIFESTO TEXT FILES

Classic examples of texts used for QTA are party manifestos and other political texts. In a 2002 *American Journal of Politics* piece, Michael Laver, Ken Benoit (who later became the leading creator of `quanteda`) and John Garry analysed British manifestos and Irish parliamentary speeches to extract ideological positions from word frequencies (Laver, Benoit and Garry 2003). In this spirit, here we cover manifestos from the 2017 UK election.

This example assumes a very basic starting scenario: having actual preprocessed text files at hand. The manifestos are downloaded manually from the political documents archive polidoc.net, which houses several thousand party manifestos and other political documents from Europe and is maintained at the Mannheim Centre for Social Research in Europe (MZES) at the University of Mannheim, Germany. The texts of the manifestos are preprocessed to facilitate their quantitative analysis, and are stored as raw text in UTF-8 format while removing page numbers, headers and footnotes, donation and membership calls (which sometimes appear at the end of the manifestos), as well as indexes and registers, while keeping titles and forewords. Hence this is an old-school example of picking up text for QTA, which helps us to focus on the actual transformation of input text into a text data corpus in R. An alternative data source would be the Comparative Manifesto Project (CMP: https://manifesto-project.wzb.eu), which contains manifestos and the encoding of text units in line with the CMP coding scheme.

For an illustration, consider the 2017 UK general election. This example is chosen because the election took place in the context of the highly polarising Brexit negotiations between the UK and the EU. It was a snap election held in June of that year, triggered by Prime Minister Theresa May's desire to strengthen her position in the Brexit negotiations. Failing in this goal, the Tories lost their absolute majority of seats in parliament and ended up in a minority government. In 2019, another snap election brought Boris Johnson, who had then taken over from Theresa May, back to an absolute majority in parliament.

The manifestos covered have been issued by the Green Party (GP), the United Kingdom Independence Party (UKIP), Labour (Lab), the Conservatives (Con), the Liberal Democrats (LD), the Democratic Unionist Party (DUP), the Scottish National Party (SNP) and Plaid Cymru (PC).

Definition Box 3.1

Data frames (df) are a tabular data structure that you will come across a lot in R. The data is arranged in rows and columns and can be of different types within one data frame, for example, numeric or characters. The columns of a data frame carry the variable names and can be accessed through `df$variablename`. Rows can be labelled or have an index number, and therefore all entries can be accessed easily. There are a lot of functions that work on data frames and other tabular datasets. CSV and Excel can be imported into R as a data frame and, similarly, a data frame can be exported into those formats.

We use the `readtext` package to go from a directory with the preprocessed text files to a corpus in R. As a first step, a data frame (df) is generated using the `readtext()` command. The directory (variable `datadir`) is a character object defined in advance with the path to the project within which the folder 'UK17' exists. This is specific to the user. To move within folders in the project directory, additional text describing the location of, for instance, a data folder can be pasted together with the directory variable. The specification in brackets first exercises the pasting to obtain the correct directory without a blank (R command `paste0`), and then the text from that directory is read into R. We also specify UTF-8 as the encoding, which is a good choice and must match the actual encoding of the text to avoid errors in the recognition of text elements. UTF-8 is the standard used for the texts on polidoc.net. For more information on what UTF-8 is and how to deal with wrongly encoded text, see Advanced Knowledge Box 3.1.

```
df_uk <- readtext(paste0(datadir,"/UK17"), encoding="UTF-8")
```

Note that `readtext()` creates a data frame but not yet a corpus object. A data frame is more general (corpus objects are also data frames). The command finds all text files in the directory (so make sure there is nothing unwanted in there) and creates two variables: `doc_id` with the document names, which equal the file names (so they are a relevant choice), and `text` with the full text of the single files.

Advanced Knowledge Box 3.1

UTF-8, which stands for 'Unicode Transformation Format–8-bit', is a character encoding standard that assigns a unique binary code to each character in most of the world's writing systems. UTF-8 uses variable-length encoding, allowing it to efficiently represent both common and less frequently used characters. To make sure that all the symbols in the texts you are analysing in R are displayed correctly, independent of their language or special characters (including emojis), you should import them in UTF-8. There are many ways to check and change the encoding of text. For a pragmatic and convenient solution, we use the free editor Notepad++, where there is a menu entry 'Encoding' to check and alter the encoding. Besides our text input, to show UTF-8 files correctly, we also want our outputs in UTF-8. To change the standard encoding of R files, go to Tools > Global Options > Code > Saving and set the default text encoding to UTF-8. If you want to change a file you worked with prior to UTF-8 encoding, go to File, click on 'reopen with encodings' and choose UTF-8 from the list.

The `head()` command from `quanteda` shows the entries for the first six of the eight manifestos.

```
head(df_uk)
## readtext object consisting of 6 documents and 0 docvars.
## # A data frame: 6 × 2
##   doc_id                 text
## * <chr>                  <chr>
## 1 51101.000.2017.1.1.txt "\"The green \"..."
## 2 51301.000.2017.1.1.txt "\"ACTION PLA\"..."
## 3 51320.000.2017.1.1.txt "\"FOR THE MA\"..."
## 4 51421.000.2017.1.1.txt "\"[only Engl\"..."
## 5 51620.000.2017.1.1.txt "\"FORWARD, T\"..."
## 6 51902.000.2017.1.1.txt "\"STRONGER F\"..."
```

The doc_id is not very informative until you read the codebook of polidoc.net and learn that the numbers indicate the country of the manifesto (51), the party (101), the political level, here national (000), the year of the manifesto (2017) and its version (1.1). In other words, the logic of file names on polidoc.net is: `country_ID.party_ID.subnational_ID.year.document_year.version.txt`. This allows us to identify manifestos precisely. We can later define more directly accessible document names. Also note that encodings for the end of a line (`\n`) and the like are preserved, which can be useful for the further transformation of text. In the next step, we create a text corpus from the data frame using `quanteda`'s `corpus()` command. For help on the command, type '`?corpus()`' in R.

```
corpus_uk <- corpus(df_uk)
summary(corpus_uk,n=5)
## Corpus consisting of 8 documents, showing 5 documents:
##
##                    Text  Types  Tokens Sentences
## 51101.000.2017.1.1.txt   1108    3332        66
## 51301.000.2017.1.1.txt   1663    7021       307
## 51320.000.2017.1.1.txt   4292   25798      1033
## 51421.000.2017.1.1.txt   4081   23464       464
## 51620.000.2017.1.1.txt   4177   32657      1184
```

As the line `summary(corpus_uk)` shows (`n=5` restricts the number of lines displayed), at the document level, the corpus again features the names of the texts (variable `Text`) as

well as meta-information in the form of counts of types, tokens and sentences. Types are typically words (but could also be emoticons or some other unit) and are counted only once if they appear in a text, regardless of frequency ('Harry and Meghan and Archie' would give a count of four). Tokens equal the total number of appearances of different or the same types ('Harry and Meghan and Archie' would give a count of five). Sentences are defined via stop symbols such as '.', '?' or '!'.

As already mentioned, manifesto names are too cumbersome to serve, for instance, as labels in graphs, so we provide an additional variable with party abbreviations. This can be done through document variables, a separate set of meta-information contained in a corpus at the text level. New document variables are added with the `quanteda` command `docvars()`. We generate party labels and store them in the document variable `party` using:

```
docvars(corpus_uk, "party") <- c("GP","PC","Lab","LD","Con",
"SNP","DUP","UKIP")
summary(corpus_uk, n=5)
## Corpus consisting of 8 documents, showing 5 documents:
##
##                     Text  Types  Tokens Sentences party
##   51101.000.2017.1.1.txt   1108    3332         66    GP
##   51301.000.2017.1.1.txt   1663    7021        307    PC
##   51320.000.2017.1.1.txt   4292   25798       1033   Lab
##   51421.000.2017.1.1.txt   4081   23464        464    LD
##   51620.000.2017.1.1.txt   4177   32657       1184   Con
```

When we use `summary()`, we observe that the new document variable has been added to the corpus meta-information. Note that this act of manual labelling requires knowledge of the order of the manifestos in the data frame. As discussed in Chapter 2, we can also extract parts of the file names to create document variables. Of course, the corpus also contains the actual text in the manifestos, which could be displayed using the `print()` command. We restrict the display to the first three texts, the Green Party, Plaid Cymru and Labour manifestos, with `max_ndoc=3` and the number of characters displayed each with `max_nchar=100`:

```
print(corpus_uk, max_ndoc=3, max_nchar=100)
## Corpus consisting of 8 documents and 1 docvar.
## 51101.000.2017.1.1.txt :
## "The green party for a confident and caring britain 2017
our guarantee to you The Green Party has big..."
```

```
##
## 51301.000.2017.1.1.txt :
## "ACTION PLAN 2017 Party of Wales Plaid Cymru Wales faces
grave risks as we head into this election. O..."
##
## 51320.000.2017.1.1.txt :
## "FOR THE MANY NOT THE FEW THE LABOUR PARTY MANIFESTO 2017
FOREWORD A big part of being the leader of ..."
##
## [ reached max_ndoc ... 5 more documents ]
```

We see a string of text with the manifesto headings, slogans and some further text. To display basic information about a specific element (or define a sub-corpus), you can use:

```
corpus_uk["51101.000.2017.1.1.txt"]
## Corpus consisting of 1 document and 1 docvar.
## 51101.000.2017.1.1.txt :
## "The green party for a confident and caring britain 2017
our ..."
```

If you wish to access the full text of a specific element of the corpus directly, you could use double brackets (we refrain from printing the long result here):

```
corpus_uk[["51101.000.2017.1.1.txt"]]
```

Above, you learned that a corpus object has many unique properties, making it useful for text analysis. Besides the discussed elements, a corpus can store information about the computer system, the R version and the corpus creation date. If you expand the corpus in the Global Environment view in RStudio and click 'show attributes', you can see all these elements. See https://quanteda.io/reference/corpus.html for further details.

Knowledge Box 3.1

To look at a whole data frame, corpus or other element type, you can use the `View(document name)` function. Alternatively, you can click on the table symbol next to the document name in the 'environment' section of RStudio. You should see that section to the right of the script you are working on. This click triggers the same command; check your console to see.

The tools from `readtext()` and `quanteda` nicely prepare the data for further QTA, combining raw text with meta-information, a powerful combination allowing a wide range of data manipulations. In the next section, let's suppose we wish to perform some BOW (bag-of-words) type study, which assumes that political messages can be carried by key words. This requires a document-feature matrix (dfm), reporting the count of words per text. The dfm should not be confused with a data frame (df), which here simply stores the full texts and their names after obtaining them using `readtext()`. In other words, the dfm captures the distribution of words (more generally, tokens or features) across texts. It should be noted that while this provides the starting point for BOW approaches, it omits the position of words in texts. Other types of QTA, especially of the machine learning kind, acknowledge the context of words and are discussed in depth in Chapters 5 and 6.

The `tokens()` command is used to extract some of the words from the manifestos (see https://quanteda.io/reference/tokens.html for all options). It lets us remove punctuation (`remove_punct = TRUE`), symbols (`remove_symbols = TRUE`), numbers (`remove_numbers = TRUE`) and URLs (`remove_url = TRUE`). We can also choose to split hyphens and pass on corpus document variables (default) if one is used as an input.

```
tok_uk <- tokens(corpus_uk, remove_punct = TRUE, remove_
symbols = TRUE, remove_numbers = TRUE, remove_url = TRUE)
head(tok_uk, 3)
## Tokens consisting of 3 documents and 1 docvar.
## 51101.000.2017.1.1.txt :
##   [1] "The"   "green"     "party"     "for"      "a"
"confident"
##   [7] "and"   "caring"    "britain"   "our"   "guarantee"   "to"
## [ ... and 2,880 more ]
##
## 51301.000.2017.1.1.txt :
##   [1] "ACTION" "PLAN"   "Party"   "of"   "Wales"   "Plaid"
"Cymru"   "Wales"
## [9] "faces"   "grave"   "risks"   "as"
## [ ... and 6,288 more ]
##
## 51320.000.2017.1.1.txt :
## [1] "FOR"     "THE"     "MANY"     "NOT"       "THE"   "FEW"
## [7] "THE"     "LABOUR"  "PARTY"    "MANIFESTO" "FOREWORD""A"
## [ ... and 23,281 more ]
```

The overview of the first three manifestos shows that more than 23,000 words are extracted from the Labour manifesto. The Green Party's manifesto is much shorter, with less than 3000 words extracted. We also note typically uninformative stopwords such as 'the' or 'and'. These can be removed using the separate `stopwords` package (Muhr, Benoit and Watanabe 2021), which is loaded using:

```
library(stopwords)
stopwords::stopwords_getlanguages("snowball")
## [1] "da" "de" "en" "es" "fi" "fr" "hu" "ir" "it" "nl"
"no" "pt" "ro" "ru" "sv"
head(stopwords::stopwords("en", source = "snowball"), 20)
## [1]  "i"      "me"       "my"          "myself"    "we"
## [6]  "our"    "ours"     "ourselves"   "you"       "your"
## [11] "yours"  "yourself" "yourselves"  "he"        "him"
## [16] "his"    "himself"  "she"         "her"       "hers"
```

The second line of code chooses one of the available stopword lists, where the default is the 'Snowball' stopword list, and displays the languages available from it (see www.rdocumentation.org/packages/stopwords/versions/2.3 or a newer version for more details). Other sources cover languages not featured on the list but might be of poorer quality. Make sure to specify the correct language, as otherwise, you might just think that stopwords have been removed. In our case, we select 'en' for English text. The third line of code shows some of the stopwords covered. Note that you could alter the stopword list if there are, for instance, theoretical reasons to do so.

The removal itself is done by `tokens_select()`, specifying the pattern to match and what to do with it:

```
tok_uk_stop <- tokens_select(tok_uk, pattern = stopwords("en"),
selection = "remove")
head(tok_uk_stop, 3)
## Tokens consisting of 3 documents and 1 docvar.
## 51101.000.2017.1.1.txt :
## [1] "green"   "party"   "confident"   "caring"   "britain"
"guarantee"
## [7] "Green"   "Party"   "big"         "bold"     "ideas"
"create"
## [ ... and 1,711 more ]
```

```
##
## 51301.000.2017.1.1.txt :
## [1] "ACTION"     "PLAN"     "Party"     "Wales"     "Plaid"
"Cymru"
## [7] "Wales"      "faces"    "grave"     "risks"     "head"
"election"
## [ ... and 3,573 more ]
##
## 51320.000.2017.1.1.txt :
## [1] "MANY"   "LABOUR"   "PARTY"      "MANIFESTO"   "FOREWORD"
"big"
## [7] "part"   "leader"   "political"   "party"      "meet"
"people"
## [ ... and 13,586 more ]
```

Note that in addition to stopwords, we could also remove user-defined further words depending on our analytical needs. In this case, we could remove some party names that appear in the texts but only carry tautological meaning given that we are, for example, interested in deriving political positions from word usage:

```
tok_uk_stop2 <- tokens_select(tok_uk_stop, pattern = c("DUP", "Plaid",
"Cymru",
                                        "Conservatives", "Labour",
                                        "UKIP", "Green"),
                                  selection = "remove")
head(tok_uk_stop2, 3)
## Tokens consisting of 3 documents and 1 docvar.
## 51101.000.2017.1.1.txt :
##   [1] "party"  "confident"  "caring"  "britain"   "guarantee"
"Party"
##   [7] "big"    "bold"      "ideas"   "create"    "confident"
"caring"
## [ ... and 1,689 more ]
##
## 51301.000.2017.1.1.txt :
##   [1] "ACTION"   "PLAN"      "Party"     "Wales"   "Wales"
##   [6] "faces"    "grave"     "risks"     "head"    "election"
```

```
##   [11] "economy"   "communities"
## [ ... and 3,369 more ]
##
## 51320.000.2017.1.1.txt :
##   [1] "MANY"    "PARTY"     "MANIFESTO" "FOREWORD" "big"
"part"
##   [7] "leader"  "political" "party"     "meet"     "people"
"across"
## [ ... and 13,218 more ]
```

The resulting samples already contain more clear political buzzwords, such as 'Britain' and 'People'. Such word bags extracted from party manifestos can be used for further descriptive analysis or statistical modelling. Often, the distribution of the words across texts is of interest, which relates to the dfm (document-feature matrix). To create a dfm from the tokens object, we use:

```
dfm_uk <- dfm(tok_uk_stop2)
head(dfm_uk)
## Document-feature matrix of: 6 documents, 8,788 features
(70.52% sparse) and 1 docvar.
##                          features
##   docs                    party confident caring   britain
guarantee big bold
##   51101.000.2017.1.1.txt    12         4      4         5
        4   1    1
##   51301.000.2017.1.1.txt    19         0      2         0
        4   1    0
##   51320.000.2017.1.1.txt    16         0      0        47
       17   2    1
##   51421.000.2017.1.1.txt     9         2      0        23
        6   2    1
##   51620.000.2017.1.1.txt    10         1      4        96
        3   5    1
##   51902.000.2017.1.1.txt     2         0      1         0
        5   1    0
##                   features
## docs                      ideas create can
##   51101.000.2017.1.1.txt      1      5   8
```

```
##   51301.000.2017.1.1.txt      0      7  74
##   51320.000.2017.1.1.txt      1     21  40
##   51421.000.2017.1.1.txt      0     15  32
##   51620.000.2017.1.1.txt      2     29  99
##   51902.000.2017.1.1.txt      1      5  28
## [ reached max_nfeat ... 8,778 more features ]
```

A glance at the first manifestos reveals that 8793 types (unique tokens or features) appear in the eight manifestos covered. `quanteda` kindly provides further information on the number of documents covered and the presence of a document variable. Also, we learn that the matrix is 70.53 per cent sparse, which means that, often, a word does not appear in a manifesto (entries for word–manifesto combinations are 0). As a quick descriptive-analytical demonstration, a word cloud (Figure 3.1) is computed for all words in the manifestos:

```
library(quanteda.textplots) #load necessary package
textplot_wordcloud(dfm_uk, max_words = 100, min_count = 3,
                   color = "black")
```

Figure 3.1 Word cloud of 2017 British party manifestos

This requires the separate `quanteda.texplots` package. In the `textplot_wordcloud()` command, specifications include that the 100 most frequent words (`max_words`) with a minimum count of three (`min_count`) should be used. See https://quanteda.io/reference/textplot_wordcloud.html for details. The cloud (Figure 3.1) gives a good representation of the themes in the 2017 UK election, apparently spearheaded by an effort by parties to put people first. Government, the economy and health also appear to be central topics. To be sure, and to reiterate, a word cloud will only take us so far analytically. For a critical assessment of word clouds and alternatives, see Chapter 4.

To explore differences between parties, we can produce two more word clouds, one each for the Green Party and UKIP:

```
dfm_green <- dfm_subset(dfm_uk, party == "GP")
textplot_wordcloud(dfm_green, max_words = 100, min_count = 3,
                   color = "black")
```

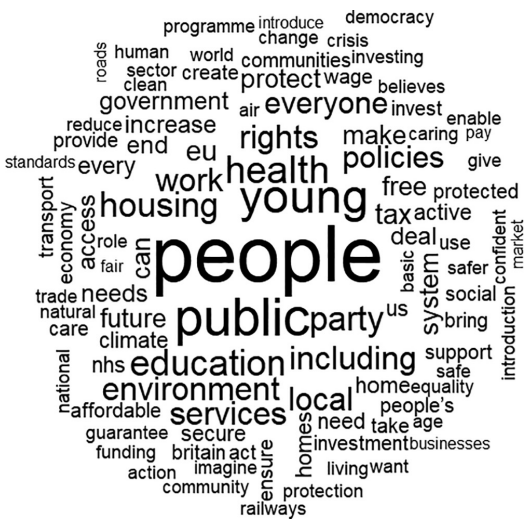

Figure 3.2 Word cloud of the Green Party manifesto

```
dfm_ukip <-dfm_subset(dfm_uk, party == "UKIP")
textplot_wordcloud(dfm_ukip, max_words = 100, min_count = 3,
                   color = "black")
```

Again, the clouds (Figures 3.2 and 3.3) give a good first idea of what the texts are about. It comes as no surprise that both manifestos talk about the 'people' a lot, as

Figure 3.3 Word cloud of the UKIP manifesto

would be expected in a representative democracy. The differences lie in the question of who is meant and what should be done for them. The Green Party appears to emphasise public services such as 'housing' and 'education', as well as 'health' and the 'environment', and talks about the 'young'. Note that we cannot infer from the word cloud whether the party actually advocates for young peoples' matters or whether it, for instance, wants to provide housing and education for them – these are the limitations of the BOW approach. UKIP has a different focus on 'britain' and the 'british' (words are all converted to lowercase) people or matters, as well as the 'uk', 'brexit' and the 'nhs'. A typical problem with the BOW approach is again observed in the fact that 'eu' is very frequently contained in the manifesto, but we can make the likely assumption that the party does not talk fondly about the European Union.

WIKIPEDIA DATA

Another data source often used for QTA is Wikipedia. Luckily, Corydon Baylor created an R package to import Wikipedia texts into R (Baylor 2022). Therefore, we can use one simple function from the `getwiki` package:

```
#install.packages("getwiki")
library("getwiki")
diwali <- get_wiki("Diwali",clean = T)
class(diwali)
## [1] "character"
```

The `get_wiki()` function only requires a search term and `clean`, which, when TRUE, removes HTML tags from the text. The Wikipedia text is returned as a character vector.

Let us download a few more texts through a loop. If you are searching for multiple words, make sure you connect them with an underscore.

```
holidays <- c("Diwali", "Holi", "Christmas", "Easter", "Eid_al_
Fitr", "Eid_al_Adha", "Kathina", "Vesakh", "Pessach", "Jom_Kippur")
holiday_texts <- c() #empty
for (holiday in holidays){ #for loop
  holiday_texts <- c(holiday_texts, get_wiki(holiday, clean = T))
  Sys.sleep(3) #take a break
}
```

Now we can create a corpus of texts explaining the most important holidays of the biggest religions:

```
holiday_corpus <- corpus(holiday_texts)
head(holiday_corpus,5) #only show first 5
## Corpus consisting of 5 documents.
## text1 :
## "Diwali (English: ; Deepavali, IAST: Dīpāvalī) is the Hindu ..."
##
## text2 :
## " Holi (Hindi pronunciation: ['ho:li:]) is a popular and sign..."
##
## text3 :
## "Christmas is an annual festival commemorating the birth of J..."
```

```
##
## text4 :
## "Easter, also called Pascha (Aramaic, Greek, Latin) or Resurr..."
##
## text5 :
## "Eid al-Fitr ( EED əl FIT-ər, - rə; Arabic: عيد الفطر, romani..."
```

Knowledge Box 3.2

Loops in R, such as for-loops, are invaluable tools for executing repetitive tasks across all elements of a vector. The structure of a for-loop is defined as follows: for (variable in vector) {do this}. The loop iterates over each element of the vector until it reaches the end. Within the curly brackets, the 'variable', often referred to as 'i', is utilised to represent each element in the vector sequentially. It is essential to note that if you intend to create a list within the loop, you must ensure that you append elements to the list to avoid overwriting existing entries. Additionally, aside from the for-loop, R offers other looping constructs such as the while-loop and the repeat-loop, each serving a specific purpose.

PREPROCESSING SOCIAL MEDIA DATA

The availability of social media data for researchers varies. At the time of writing, access to X, former Twitter has become restricted. Nevertheless, we want to give you the tools to remove special symbols in social media data or other text. We use Twitter data carrying the hashtag #fridaysforfuture. A sample of 6,353 tweets was captured on 24 January 2020, a few days after Greta Thunberg (who then focused on climate change) participated in the World Economic Forum in Davos, Switzerland. This period can be considered to be the peak of the movement thus far, as the SARS-CoV-2 virus began causing the illness Covid-19 in people at that time, which dominated both the world and Twitter in the following months and years. Retrieving tweets used to be done through the Twitter API (application programming interface). This access has not been available for free since the end of March 2023 (Twitter Dev. 2023).

We have a quick closer look at four exemplary tweets from the dataset:

```
fff$text[c(28,766, 799, 814)]
## [1] "I finally finished this portrait of Greta. 🌍🔥 Which
version is your favourite? \n\n#FridaysForFuture
#schoolstrike4climate @GretaThunberg #illustration https://t.
co/rN66K1xtYx"
```

```
## [2] "There is no planet B! 🌍❌ \n\nSo lets make a
change! \n\nSign the ECI➡️☐📝https://t.co/pLXOs5XLWe\n\nTag
some of your friends! 👪\n\nLet's get that million
signatures!\n\n#fridaysforfuture #climateaction
#climatejustice #climateemergency #climatechange #forfuture
#thereisnoplanetb #NowECI https://t.co/lG2q0fTYXt"

## [3] "Die Menschheit rottet eher an Verblödung aus als dass
sie durch den Klimawandel ausstirbt.\n\nErste Anzeichen der
Verblödung erkennen sie an #GretaThunberg, #LuisaNeubauer und
#FridaysForFuture https://t.co/pi3kAqHiz5"

## [4] "@FFF_Frankfurt Wenn man freitags in die Schule geht,
muss man solche dämlichen Fragen nicht
stellen.\n\n#Bildungslücke\n#FridaysForFuture"
```

The first tweet displays the popularity of Greta Thunberg and her call for action against climate change. The English translation of the two German tweets, using the impressive DeepL translation software (www.deepl.com/de/translator), gives:

```
"Mankind is more likely to become extinct through
stupefaction than to die out as a result of climate change.
\n\nThe first signs of stupefaction can be seen in
#GretaThunberg, #LuisaNeubauer and #FridaysForFuture
https://t.co/pi3kAqHiz5"

"@FFF_Frankfurt If you go to school on Fridays, you don't have
to ask such stupid questions.
\n\n#educationalgap\n#FridaysForFuture"
```

[Translated using www.DeepL.com/Translator (free version)]

The translation reveals that in the afternoon of 24 January 2020, among German-speaking Twitter users, derogatory language was used against Greta Thunberg and Luisa Neubauer. Hence, the analysis of the medium immediately adds evidence to the criticism that social media is prone to hate speech.

In the sample we also have a tweet example of Urdu ('ur'), a language spoken in Pakistan and parts of India:

```
fff$text[fff$lang=="ur"]

## [1] "جمعہ مبارک 😇✨\n#JummahMubarak\n#Friday
#FridaysForFuture #FridayMotivation"
```

According to Google Translate (https://translate.google.com), the phrase 'JummahMubarak' translates to 'Happy Friday'. The Unicode representation of the smiley with halo emoticon would be `\U0001f607` and the sparkles would be `\u2728`, if with correct encoding. More information about this can be found in Advanced Knowledge Box 3.1. Whether these elements are actually related to the Fridays for Future movement cannot be inferred.

Suppose that we are interested in English tweets and a few variables only, including a label for users, the text of the tweets and the time of creation. To protect users' privacy, we replace the screen names with an ID made of the row numbers in the dataset. The second line introduces you to the `subset()` function, which you will use a lot when manipulating data frames. The third line also subsets the data frame, accessing the fields through square brackets. The structure is [rows, columns], therefore we have to add a comma before the code to specify all rows. This leaves a subset of 2,373 tweets.

```
fff$our_name <- paste("user", seq(1, nrow(fff), 1), sep = "_")
#change names
fff_en <- subset(fff, lang == "en") #only english tweets
fff_en <- fff_en[, c("created_at", "our_name", "text")]
#remove other variables
```

We will use these English tweets for some basic analysis. But first, the tweets should be cleaned and prepared. To this end, we rely on a few `quanteda` functions and turn the data frame of tweets into a corpus:

```
fff_corpus <- corpus(fff_en)
summary(fff_corpus, n=10)
## Corpus consisting of 2373 documents, showing 10 documents:
##
##    Text Types Tokens Sentences        created_at our_name
##   text1    19     20         1 2020-01-24 13:15:34 user_2
##   text2    17     17         1 2020-01-24 13:14:26 user_3
##   text3    24     24         1 2020-01-22 13:06:06 user_4
##   text4    20     20         1 2020-01-16 13:06:09 user_5
##   text5    20     20         1 2020-01-17 13:04:08 user_6
##   text6    26     26         1 2020-01-21 13:05:52 user_7
##   text7    21     21         1 2020-01-23 13:14:27 user_8
```

```
##   text8    17     17         1 2020-01-23 17:18:09 user_15
##   text9    22     22         1 2020-01-24 13:12:29 user_16
##   text10   11     12         3 2020-01-24 13:11:47 user_17
```

Suppose we want to have a look at the most frequent words in the tweets. We move to the token level to remove URLs, hashtags and handles, relying on the `tokens()` function from `quanteda` and the options offered by it or our own code. For URLs, the package offers a function `remove_url` at the level of its tokeniser, which is deactivated by default:

```
as.character(tokens(fff_corpus[14]))
##    [1] "I"                     "finally"
##    [3] "finished"              "this"
##    [5] "portrait"              "of"
##    [7] "Greta"                 "."
##    [9] "🌍"                    "🔥"
##   [11] "Which"                 "version"
##   [13] "is"                    "your"
##   [15] "favourite"             "?"
##   [17] "#FridaysForFuture"     "#schoolstrike4climate"
##   [19] "@GretaThunberg"        "#illustration"
##   [21] "https://t.co/rN66K1xtYx"
as.character(tokens(fff_corpus[14], remove_url=TRUE))
##    [1] "I"              "finally"            "finished"
##    [4] "this"           "portrait"           "of"
##    [7] "Greta"          "."                  "🌍"
##   [10] "🔥"             "Which"              "version"
##   [13] "is"             "your"               "favourite"
##   [16] "?"              "#FridaysForFuture"  "#schoolstrike
4climate"
##   [19] "@GretaThunberg" "#illustration"
```

This gets rid of 'https://t.co/rN66K1xtYx'. Further functions might be needed if the URL is written in a different way, for instance if it starts with 'www'. Note that we have made no changes to the corpus yet but have only shown the result of applying the function for URL removal. We will perform the full removal, along with hashtags and handles.

To this end, a little piece of regex is used, as these are systematically introduced by '#' or '@'. We create a tokens object `fff_toks` from the tweets corpus, removing punctuation and URLs at the same time. In a further step, the command `tokens_remove()` allows the application of the regex search and delete exercise for the hashtags and handles, with the result being stored in the object `fff_toks_txt`.

```
fff_toks <- tokens(fff_corpus, what = "word", remove_punct =
TRUE, remove_url=TRUE)
fff_toks_txt <- tokens_remove(fff_toks, pattern = "^[#@].+$",
valuetype = "regex")
fff_toks_txt[14]
## Tokens consisting of 1 document and 2 docvars.
## text14 :
## [1] "I"     "finally" "finished" "this"   "portrait" "of"
## [7] "Greta" "🌍"       "🔥"        "Which"  "version"  "is"
## [ ... and 2 more ]
```

This allows us to isolate the text in a tweet. We could also select only hashtags and handles by using:

```
fff_toks_haha <- tokens_select(fff_toks, pattern = "^[#@].+$",
valuetype = "regex")
fff_toks_haha[14]
## Tokens consisting of 1 document and 2 docvars.
## text14 :
## [1] "#FridaysForFuture"   "#schoolstrike4climate"
"@GretaThunberg"
## [4] "#illustration"
```

Advanced Knowledge Box 3.2

Regex, short for regular expression, is like a supercharged find-and-replace tool. It's a special sequence of characters that helps you search for patterns in text. You can use regex to find specific words, numbers or even complex patterns in a document. To use regex, you need to know what each symbol means. You can learn that from this cheat sheet: https://cheatography.com/davechild/cheat-sheets/regular-expressions/.

To prepare for a plot, we create a dfm of the tweet text while furthermore removing stopwords (see Definition Box 2.2 in Chapter 2), displaying the head of the dfm as well as the top features:

```
fff_dfm <- dfm(fff_toks_txt)
fff_dfm <- dfm_remove(fff_dfm, stopwords("english"))
head(fff_dfm, 10, n = 10)
## Document-feature matrix of: 10 documents, 6,503 features
(99.88% sparse) and 2 docvars.
##   features
## docs despite going keep hopealive stay strong shall pass crude oil
##   text1    1     1    1         1    1      1     1    1     0  0
##   text2    0     0    0         0    0      0     0    0     1  1
##   text3    0     0    0         0    0      0     0    0     0  0
##   text4    0     0    0         0    0      0     0    0     0  0
##   text5    0     0    0         0    0      0     0    0     0  0
##   text6    0     0    0         0    0      0     0    0     1  1
## [ reached max_ndoc ... 4 more documents, reached max_nfeat ...
6,493 more features ]
topfeatures(fff_dfm, n = 10)
## climate people change world can today now future strike  ❤
##     447    197    158   156 149   141 130    129    124 113
```

Compared to the sparseness of 70.53 per cent of the dfm of party manifestos, we can see that this one is 99.88 per cent sparse, speaking to a more diverse use of words. In social media texts, you can expect a high level of spareness.

Finally, we can plot the most frequent words using `ggplot()` from the `tidyverse` package (you can find more on `ggplot` in Chapter 4):

```
frequ <- as.numeric(topfeatures(fff_dfm,20))

word <- as.character(names(topfeatures(fff_dfm, 20)))
fff_plot <- as.data.frame(frequ,word)
ggplot(fff_plot, aes(x = reorder(word, frequ), y = frequ)) +
```

```
  geom_point() +
  coord_flip() +
  labs(x = NULL, y = "Frequency")
```

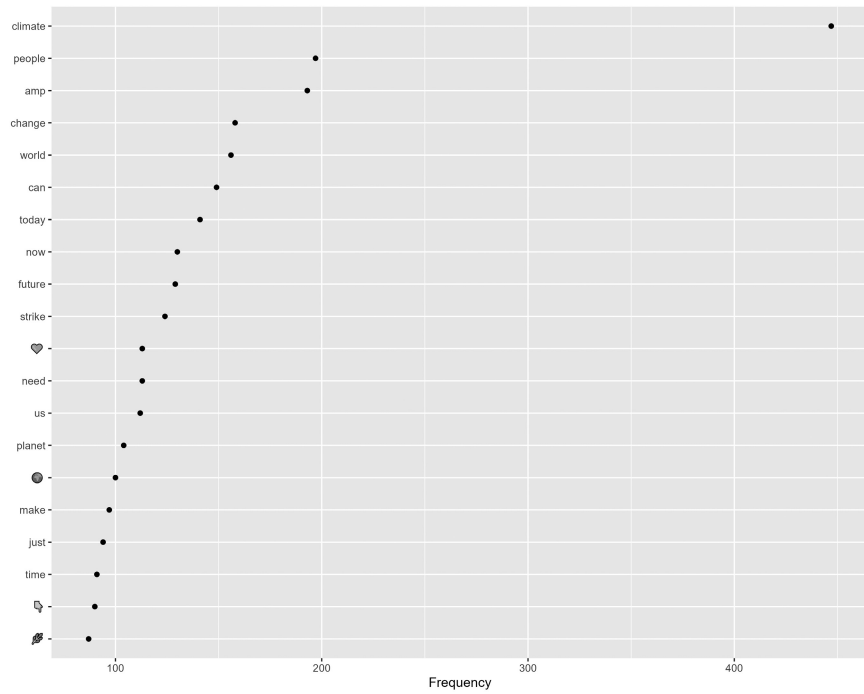

Figure 3.4 Fridays for Future Twitter data – most frequent words

In Figure 3.4, we can see that it is all about the climate! Once more, it is impressive how the list of frequently used words tells a story: After the word 'climate', the next few most frequent words are 'people', 'change', 'world', 'can' and 'today', and our brains immediately form a sentence out of them, as in 'people can change the world today'. While likely, we cannot be sure whether this is actually what is in the tweets in a BOW approach. A quick word cloud (Figure 3.5) underpins the impression from the frequency dot chart:

```
textplot_wordcloud(fff_dfm, max_words = 100, min_count = 3,
                   color = "black")
```

We will come back to the Fridays for Future tweets in Chapter 5. The chapter now returns to the data from Chapter 2, the rap lyrics by Kendrick Lamar.

Figure 3.5 Fridays for Future word cloud

SCRAPING RAP SONGS

This chapter moves from reading ready text files (manifestos) to using a convenient function for Wikipedia data retrieval via Twitter data to scraping more directly. We still use an API provided by genius.com to get hold of a larger selection of song texts and extend on the introductory example used in Chapter 2. 'Full' scraping is a further step that is somewhat outside the scope of this book, as it involves more specialised programming and easily leads users to violate data providers' terms of use.

To web scrape is to legally access the content of web databases in an automated fashion using programming code (Munzert et al. 2015). More information is available in the ethics and web scraping section of this chapter. Polite scraping is important, and you might even break some laws if you don't follow the guidelines, so reading the following section of this chapter is highly recommended if you want to scrape data yourself.

For the song lyrics on Genius, there is an API available. To make sure, we still check `robotstxt`, an R package that can help you decide if you are allowed to scrape or not. In the genius.com case, the file (stored under https://genius.com/robots.txt) contains these lines:

```
User-Agent: *
 Disallow: /songs/*/metadata
 Disallow: /songs/*/comments
 Disallow: /users/*/top_artist_iq_rank
 Disallow: /artists/*/top_iq_earners
 Disallow: /comments/for_profile_page
```

```
Disallow: /bagon/*
Disallow: /mecha/*
Disallow: /account/unsubscribe/*
Disallow: /email_notifications/*/unsubscribe
Disallow: /annotations/*/report_as_abusive
Disallow: /songs/for_profile_page
Disallow: /api$
Disallow: /api/*
Disallow: /search?*
Disallow: /*/*/leaderboard

User-agent: Uniscan
Disallow: /

User-agent: Searchie
Disallow: /
```

We can suspect that genius.com's *robot.txt* reflects that the site's operators do not want us (any user agent, marked by '*') to extract comments or account subscription details, for example. If we scrape the website, we should avoid these directories even when they are accessible and discard the scraping in case the data of interest is stored there. Also, the user-agents 'Uniscan' and 'Searchie', which are specific web crawlers, are completely banned from scraping for unknown reasons. When we check the terms of service, we see that they disallow scraping in general; therefore, we used the API and did not scrape from the website. We retrieve a sample of rap songs to demonstrate QTA on a set of medium-length, informal texts from an internet source. To motivate the exercise, we pursue a little ad hoc research question. The working 'hypothesis' is that early in the 1980s and 90s, rap was both 'gangsta' and 'conscious' (political) at the same time, reflecting the complexities of African American culture. Gangsta Rap lyrics often depict urban crime and violence, often based on the harsh realities of inner-city life. This style emerged in the late 80s alongside political rap, which, frustrated with old-school rap lyrics, focused on social and political topics. It is important to recognise that rap music encompasses a wide range of perspectives and experiences. For this QTA example, we simplify and focus only on some characteristics. In gangsta rap lyrics, we would expect words indicating violence, crime, drugs and sexism. In political rap, themes could overlap, featuring violence and drugs as well, but with a different connotation, seeking to critique societal injustices and advocate for change, and potentially less sexism. Caldwell (2008) uses a framework to investigate specific differences in the characteristics of those two types of rap music. Some hints on such a relationship can be found in Dawson (1999, 334). In this book, we just want to make an example of QTA and don't go as deep.

For this exploratory analysis and a demonstration of web scraping, five songs from three suspected classes are selected. Table 3.1 provides an overview. The sample is selective, and the allocation of songs to classes as well as the classes themselves are rather crude, but the purpose is to test some tendencies and possibly detect outliers.

Table 3.1 Songs selected for scraping

Artist	Song	Class (suspected)
Grandmaster Flash and the Furious Five	The Message	Oldskool
NWA	Straight Outta Compton	Oldskool
Ice-T	6 n the Mornin	Oldskool
KRS-One	Sound of da Police	Oldskool
A Tribe Called Quest	Check the Rhime	Oldskool
Rick Ross	Hustlin	New Gangsta
Bobby Shmurda	Hot ***	New Gangsta
Drakeo the Ruler	Flu Flamming	New Gangsta
Lil Wayne	A Milli	New Gangsta
Wiz Khalifa	We Dem Boyz	New Gangsta
Eminem	The Storm	New Conscious
Wale	Ambition	New Conscious
Tyler the Creator	Yonkers	New Conscious
Kendrick Lamar	Humble	New Conscious
Logic	Everybody	New Conscious

As mentioned, fortunately, genius.com has an API, so we can both expect good results and be polite visitors to their server. As a plus, someone has created an R package for us to use the API in R (Henderson 2022). More information about the package can be found on the GitHub page of the creator: https://ewenme.github.io/geniusr/. Together, this provides all the tools needed to quickly and cleanly scrape song lyrics from genius.com. We continue by loading the `geniusr` package:

```
library(geniusr)
```

The API itself requires a few steps for registration at https://genius.com/developers. After registration, you end up with an user agent name, an ID, a secret code and an access token, which will be requested by the `geniusr` package and can be stored as a system

environment variable interactively using the `genius_token()` command. This function outputs your personal token, which is not printed here.

```
genius_token()
```

After having a look at the `geniusr` documentation, we can identify a number of useful functions that take care of the tasks we are looking for. In particular, a combination of `search_song()`, `get_song()` and `get_lyrics_id()` lets us find the genius.com IDs from song names, which in turn allows us to retrieve sample information on the song as well as the lyrics themselves, notably line-by-line, along with useful additional document-level variables.

With `search_song()` followed by `get_song()`, we can, for example, find all information on (if not yet the lyrics) of Kendrick Lamar's song 'DNA.' (with full stop). The first command provides us with a list of possible songs matching the search, including the respective genius.com song ID, while the second function retrieves the full list of meta-information based on the ID, of which we display the full title and release date.

```
search_song("DNA") #the correct id is 3035222
## # A tibble: 10 × 5
##   song_id   song_name                        song_lyrics_url
artist_id   artist_name
##     <int>      <chr>                          <chr>
<int> <chr>
##   1 3035222 DNA.                             https://genius...
1421    Kendrick L...
##   2 3233281   DNA                            https://genius...
70113   BTS
##   3 3804151 BTS - DNA (English Translation)  https://genius...
196943   Genius Eng...
##   4 3804152 BTS - DNA (Romanized)            https://genius...
1507735 Genius Rom...
##   5 729535  DNA (Ft. Na'kel Smith)           https://genius...
686     Earl Sweat...
##   6 2876029 DNA                              https://genius...
152808  Lia Marie ...
##   7 145033  D.N.A.                           https://genius...
15329   Genetikk
##   8 4313700  DNA (Ft. Capital Bra & Summer ... https://genius...
20442   KC Rebell
```

```
##   9 149911   Dizaster vs. DNA (Ft. Dizaster... https://genius...
117146   King of th...
##   10 3331753   DNA (Japanese Ver.)           https://genius...
70113     BTS

dna <- get_song(song_id = 3035222)
dna$content$full_title
dna$content$release_date_for_display
```

Next, let us get all the songs we listed in Table 3.1. To prepare the scraping exercise, the URLs of the songs we are interested in are stored in a character vector. Note that usually, we would not type them out but start from a list to import with R or search for songs by a certain artist.

```
oldskool <- c("Grandmaster-flash-and-the-furious-five-the-
message-lyrics",
              "Nwa-straight-outta-compton-lyrics",
              "Ice-t-6-n-the-mornin-lyrics",
              "Krs-one-sound-of-da-police-lyrics",
              "A-tribe-called-quest-check-the-rhime-lyrics")
new_gangsta <- c("Rick-ross-hustlin-lyrics",
                 "Bobby-shmurda-hot-nigga-lyrics",
                 "Drakeo-the-ruler-flu-flamming-lyrics",
                 "Lil-wayne-a-milli-lyrics",
                 "Wiz-khalifa-we-dem-boyz-lyrics")
new_conscious <- c("Eminem-the-storm-2017-bet-hip-hop-awards-
cypher-verse-lyrics",
                 "Wale-ambition-lyrics",
                 "Tyler-the-creator-yonkers-lyrics",
                 "Kendrick-lamar-humble-lyrics",
                 "Logic-everybody-lyrics")
lyrics <- c(oldskool,new_gangsta,new_conscious)
genius_urls <- paste0("https://genius.com/",lyrics)
```

Thanks to the API, the web scraping itself is done within a few lines of code, utilising the `get_lyrics_url()` command from the `geniusr` package. The loop draws on the list of song URLs. It creates a series of variables `lyr_1` to `lyr_i`, one for each song (where i is the maximum), with the lyrics in them. This operation is somewhat unstable and

vulnerable to disruptions, making it a true scraping experience. Adding the option `Sys.sleep(11)` gives the server an 11-second pause between each request, trying to reduce the server load and improving our chances of obtaining the desired data. In our tries, the text retrieval still fails in about half of the cases, at least most of the time.

```
for (i in 1:length(genius_urls)) {
  nam <- paste("lyr",i, sep = "_")
  assign(nam, get_lyrics_url(genius_urls[i]))
  Sys.sleep(11)
  }
```

After checking whether the variables created contain any data, it may be necessary to fill in the missing lyrics. For example, to get the lyrics of the second song, the command for another try would be:

```
lyr_2 <- get_lyrics_url(genius_urls[2])
```

Without the loop, the data is filled in after one or multiple tries. We can now move on and combine the single lyrics into a tibble, which is a tidyverse-style data frame.

```
rap_lyrics <- tibble()
rap_lyrics <- rbind(lyr_1,lyr_2,lyr_3,lyr_4,lyr_5,lyr_6,lyr_7,l
yr_8,lyr_9,lyr_10,lyr_11,lyr_12,lyr_13,lyr_14,lyr_15)
```

We learn that `get_lyrics_url()` conveniently returns not only the lyrics of each line but further variables, including the section name (e.g. intro or numbered verse), section artist and song name. The resulting data set contains 1179 lines of lyrics.

```
head(rap_lyrics, 5)
## # A tibble: 5 × 6
##   line         section_name section_artist song_name artist_name
song_lyrics_url
##   <chr>        <chr>        <chr>          <chr>     <chr>
<chr>
```

```
## 1 It's like a... Intro        Duke Bootee    The Mess... Grandmaste...
https://genius...
## 2 It makes me... Intro        Duke Bootee    The Mess... Grandmaste...
https://genius...
## 3 It's like a... Intro        Duke Bootee    The Mess... Grandmaste...
https://genius...
## 4 It makes me... Intro        Duke Bootee    The Mess... Grandmaste...
https://genius...
## 5 Broken glas... Verse 1      Melle Mel      The Mess... Grandmaste...
https://genius...
```

What we see here are some lines from one of the first commercially successful rap songs, 'The Message', describing social inequality, poverty and drug addiction in New York in the 1980s. It's relatively safe to say that the song can be classified as old-school rap, which we expect to be rather 'conscious'.

In an intermediate step, the full songs are reassembled based on the variable containing the song title:

```
rap_songs <- aggregate(rap_lyrics$line,
                       list(rap_lyrics$song_name),
                       paste, collapse=" ")
```

In the background, shorter song titles are added for later illustration, and offensive language is changed to 'EXPL', similar to Chapter 2, but this time with some analytical interest in its frequency.

With a set of 15 prepared songs at our disposal, we can now delve into some basic QTA. For this exercise, we classify all songs into 'oldschool', 'gangsta' or 'conscious', where the latter are more recent songs known to be political. Note that the classification of songs into classes is fixed for didactic purposes and would ideally be the result of careful and validated manual coding or a classification algorithm (see Chapter 6). We append a variable with the values to the data set based on the short song title added silently (variable `sosho`):

```
rap_songs$class <- ""
rap_songs$class[rap_songs$sosho=="Message"] <- "old"
rap_songs$class[rap_songs$sosho=="Compton"] <- "old"
rap_songs$class[rap_songs$sosho=="Mornin"] <- "old"
```

```
rap_songs$class[rap_songs$sosho=="Police"] <- "old"
rap_songs$class[rap_songs$sosho=="Check"] <- "old"
rap_songs$class[rap_songs$sosho=="Hustlin"] <- "gang"
rap_songs$class[rap_songs$sosho=="Hot"] <- "gang"
rap_songs$class[rap_songs$sosho=="Flu"] <- "gang"
rap_songs$class[rap_songs$sosho=="Milli"] <- "gang"
rap_songs$class[rap_songs$sosho=="Boyz"] <- "gang"
rap_songs$class[rap_songs$sosho=="Storm"] <- "cons"
rap_songs$class[rap_songs$sosho=="Ambition"] <- "cons"
rap_songs$class[rap_songs$sosho=="Yonkers"] <- "cons"
rap_songs$class[rap_songs$sosho=="Humble"] <- "cons"
rap_songs$class[rap_songs$sosho=="Everybody"] <- "cons"
```

We turn the tibble into a corpus:

```
options(width = 70)
rap_corpus <- corpus(rap_songs, text_field="x")
summary(rap_corpus, n=6)
## Corpus consisting of 15 documents, showing 6 documents:
##
##    Text Types Tokens Sentences          Group.1     sosho class
##   text1   528   1180         2 6 'N the Mornin'    Mornin   old
##   text2   295    676        13          A Milli     Milli  gang
##   text3   293    747         4        Ambition  Ambition  cons
##   text4   314    764        13 Check the Rhime     Check   old
##   text5   208    525        13       Everybody Everybody  cons
##   text6   162    480         3    Flu Flamming       Flu  gang
```

We can now create a dfm while removing stopwords, numbers and punctuation, as well as subsets for the three groups.

```
options(width = 80)
rap_songs_dfm <- dfm_remove(dfm(tokens  #create token object,
then dfm
                            rap_corpus, #from the corpus
                            remove_punct = TRUE,
                            remove_numbers = TRUE)),
```

```
                              stopwords(language = "en") #remove
                              )
rap_dfm_old <- dfm_subset(rap_songs_dfm,
                                 class=="old") #subset to only
have oldschool rap
rap_frequ_old <- textstat_frequency(rap_dfm_old) #term
frequency
rap_dfm_cons <- dfm_subset(rap_songs_dfm,
                                  class=="cons")
rap_frequ_cons <- textstat_frequency(rap_dfm_cons)
rap_dfm_gang <- dfm_subset(rap_songs_dfm,
                                  class=="gang")
rap_frequ_gang <- textstat_frequency(rap_dfm_gang)
```

A first impression of the words used in all songs is given by a word cloud, using a maximum of the 100 most frequent words appearing at least three times. All explicit words have been removed, with the same arguments as in Chapter 2. This allows for discerning the remaining non-stopwords.

```
textplot_wordcloud(rap_songs_dfm , max_words = 100, min_count = 3,
                   color = "black")
```

Figure 3.6 Word cloud of all songs

We learn that explicit language makes up the vast majority of words (Figure 3.6), stored as `expl`. Also, the overall cloud reveals that 'hol' is a common word, which is part of 'hol'' in the original lyrics and appears in the context of 'hol' up', meaning 'hold up, wait a minute'. The reason for its prominence in the sample is that it appears in two hooks ('We Dem Boyz' and 'Humble'), where words are often repeated excessively. A possible variation of the analysis would be to exclude the hooks and focus on the verses, as is done in Chapter 2.

To get some more intuition about the accuracy of the working hypothesis that old-school rap is gangsta and political, while new songs tend to fall into either category, the most frequent words in the three classes are displayed, starting with class-specific word clouds.

```
textplot_wordcloud(rap_dfm_old, color="black", max_words = 150)
textplot_wordcloud(rap_dfm_cons, color="black", max_words = 150)
textplot_wordcloud(rap_dfm_gang, color="black", max_words = 150)
```

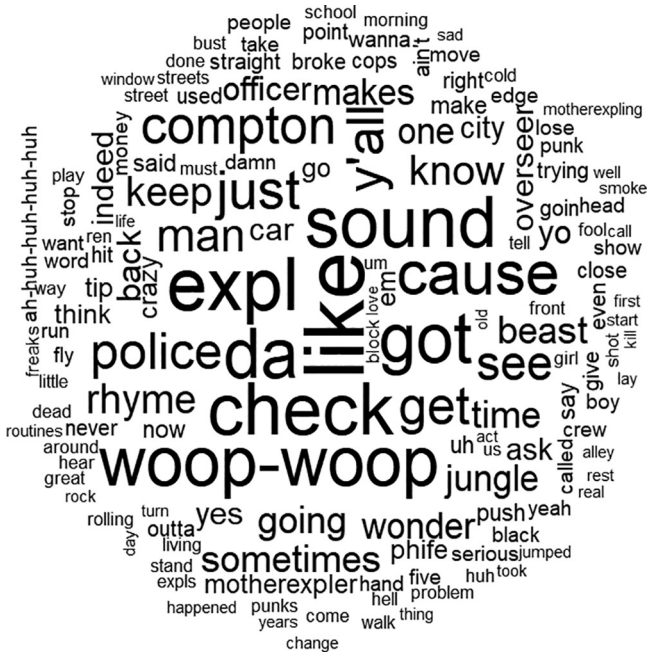

Figure 3.7 Old-school rap

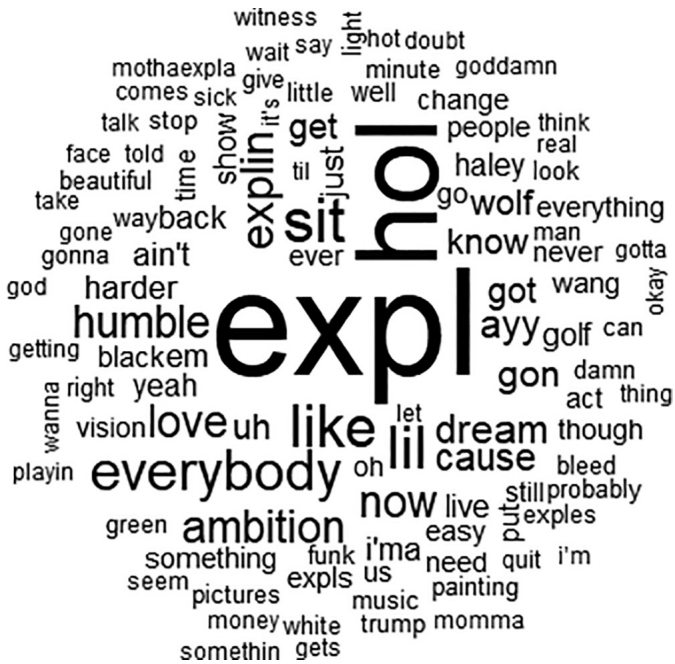

Figure 3.8 Conscious rap

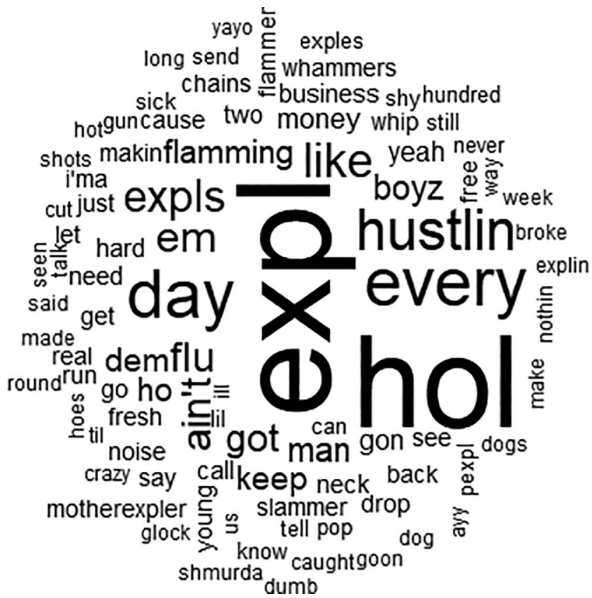

Figure 3.9 Gangsta rap

From this, we learn a little more. The songs labelled 'oldschool' (Figure 3.7) feature explicit language a bit less frequently than both 'conscious' (Figure 3.8) and 'gangsta' (Figure 3.9) rap. Obviously, this is also dictated by the selection of songs (small sample size), and as expected, 'gangsta' rap talks more about 'hustlin' (from the song by Rick Ross), the 'boys', the 'slammer' (prison) and 'flamming' (flu drug abuse). 'Conscious' rap features 'love', 'ambition', 'humble' and 'dream', hence pointing at emotion-related topics. 'Oldschool' rap fits the idea of political rap more closely, talking about 'police' (especially KRS-One), 'jungle' (Grandmaster Flash) and 'compton' (NWA), hence the outer world and its problems.

Therefore, and somewhat in line with intuition, new conscious rap songs contain references to emotions and morale. What emerges in the 'new gangsta' songs are potential references to street life. There is generally not much direct reference to violence in the sample (potentially indicated by words like 'dead' or 'gun'), resonating with Chapter 2. The word patterns observed let 'oldschool' rap appear to feature a more direct social tone. A further observation is that 'oldschool' rap is much more verbose and especially balanced, unlike more recent songs, which often repeat the same line or word. 'The Message' is a good example, narrating several stories from the streets. Also, due to this reason, we might be interested in the relative frequencies of words by group:

```
rap_dfm_grouped <- dfm_group(rap_songs_dfm, groups = class)
#group by class
rap_dfm_rel <- dfm_weight(rap_dfm_grouped, scheme = "prop")
#relative frequencies
#proportions: freq of term/total nr of terms in document
relfreq <- textstat_frequency(rap_dfm_rel, n = 10, groups =
class) #term frequency

ggplot(data = relfreq, aes(x = nrow(relfreq):1, y = frequency)) +
#create plot
  geom_point() +
  facet_wrap(~ group, scales = "free") + #adds facets
  coord_flip() +
  scale_x_continuous(breaks = nrow(relfreq):1,
                     labels = relfreq$feature) +
  labs(x = NULL, y = "Relative frequency") +
  scale_y_continuous(labels = scales::number_format(accuracy =
0.01))
```

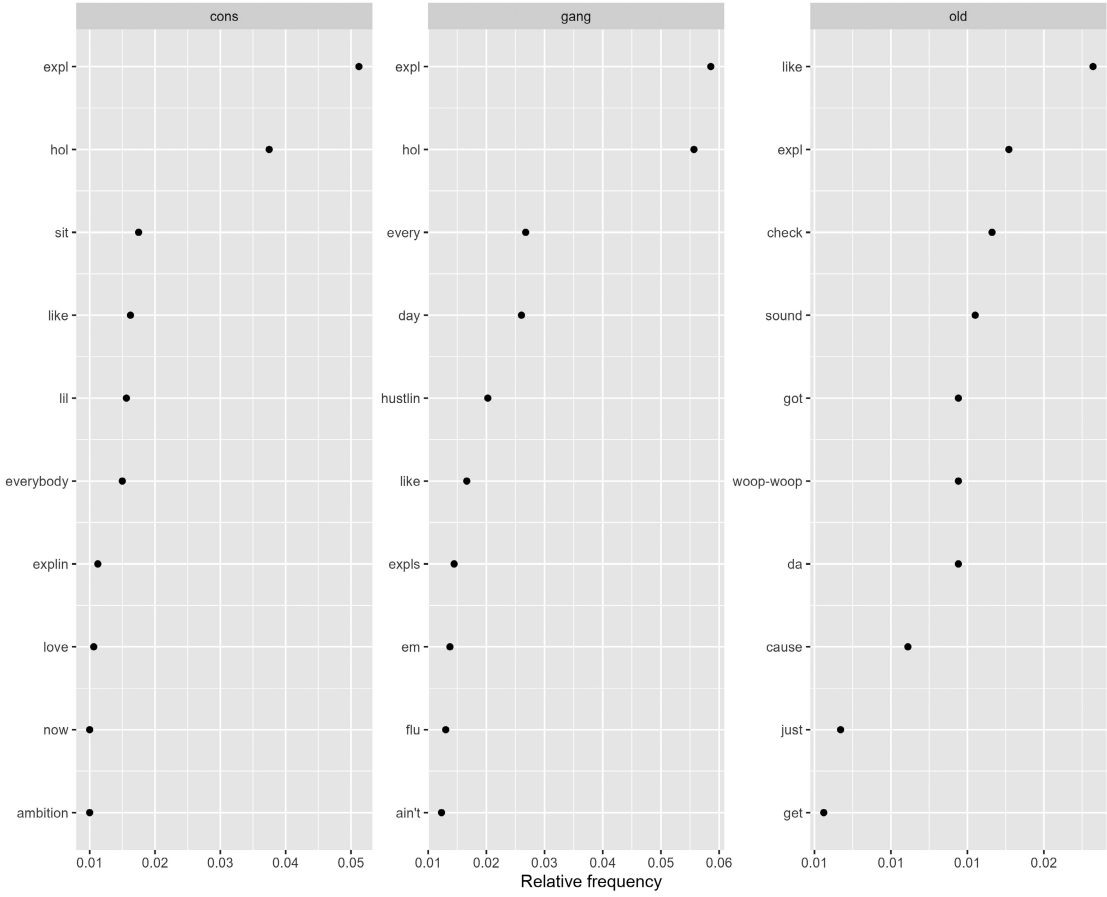

Figure 3.10 Relative word frequency of the different types of rap

Be careful when interpreting the plots in Figure 3.10; the x-axis scale is different on each one! The patterns underpin the observations made. 'Conscious' and 'gangsta' rap are similar in their usage of explicit language and lower linguistic variety. Also, all groups heavily rely on 'like', which serves well for comparisons appearing frequently in the lyrics. At the same time, the vocabulary differs to some extent.

In sum, we cannot derive a strong conclusion from this cursory analysis. In addition to the limited theoretical background initiated and limitations in the tools used, this is also a warning that not everything can be automated. To understand the meaning of a text, we might still need to read and interpret it. But we can get a descriptive impression.

INTERLUDE: RESEARCH ETHICS AND WEB SCRAPING

With the internet, data availability has changed. The additional source of data is useful for social scientists. This chapter is all about how to receive text from websites or social media. When using data, especially data about people, we have to keep in mind whether it is ethical to use it. We adress that issue in this interlude.

What is Ethical and What is Not?

With big data, research changes: it is possible to gather much more data without the consent of the people the data belongs to and without institutional oversight. This data might be available online, but people might not want it to be used for research (Zimmer 2018). Big data researchers disagree on how this data should be used due to the conceptual gaps that come with the new technology. Many researchers lack ethical training, and evaluation frameworks for research projects are limited (Zimmer 2018).

In the following, we introduce you to fundamental theories of ethics. This section is based on the structure of the computational methods course taught by Nikolai Gad (2021). Consequentialism, based on Bentham and John Stuart Mill, focuses on ethical ends, meaning the consequences of one's action. Deontology, going back to Kant, lies in 'ethical duties, independent of their consequences' (Salganik 2018), so focusing on ethical means.

Salganik describes four principles derived from the Belmont Report and the Menlo Report, American ethical guidelines involving human subjects, with the latter specifically addressing information and communication technologies (Salganik 2018). Those can be used as guidance for ethical decision-making.

1 Respect for the person. Addressing the autonomy of people, which makes informed consent necessary even if research does no harm.
2 Beneficence. The harm of research should be reduced while the benefits are maximised.

3 Justice. The risks and benefits for participants and the public have to be in an ethical balance. Vulnerable groups should be protected even more, but not excluded from beneficial research.
4 Respect for law and public interest. Obey the law and terms of service of websites. Be transparent about your goals, methods and results.

Zimmer defines the core concepts as 'minimizing harm, obtaining informed consent and protecting subject privacy and confidentiality' (Zimmer 2018). Those are comparable to Salganik's (2018) concepts.

What is Privacy?

A question that is not easy to answer. Solove offers a taxonomy of privacy after arguing that 'the right to be let alone' is too broad a definition because it is a 'plurality of things' (Solove n.d.). Solove places the focus on an overview of possible privacy problems so that they can be addressed under the name of privacy instead of a strict definition. For researchers, the most important part of this taxonomy is 'information processing'. The privacy of individuals is endangered when the collected information is abused or used without their consent (Solove 2007).

Nissenbaum offers a new theory for the definition of privacy, called 'privacy as contextual integrity' (Nissenbaum 2010). In contextual integrity, you ask if informational norms are violated by the use of data, depending on the actors, attributes and transmission principles (Salganik 2018). Meaning, 'information collection and transmission must be appropriate within a given context' (Zimmer 2018). She offers a process to check if given research fulfils those requirements in its context (Nissenbaum 2010, 182–83).

The GDPR law, described in the following section, defines personal data as 'any information that relates to an individual who can be directly or indirectly identified' ("What Is GDPR, the EU's New Data Protection Law?" 2018). It lists examples like name, email address, location information, ethnicity, gender, bio-metric data, religious belief, web cookies and political opinion, as well as pseudonymous data if it is easy to identify someone with it ("What Is GDPR, the EU's New Data Protection Law?" 2018).

Privacy Laws

The idea of open science, to share information freely for research, is a goal the science community works towards (Mahomed and Labuschaigne 2022). An equally important goal is to protect individuals' privacy.

If you work in Europe or with European data, it is good to know about the GDPR, Europe's data privacy and security law, which applies to people who 'process the personal data of EU citizen or residents' ("What Is GDPR, the EU's New Data Protection Law?" 2018). The GDPR website offers a checklist (https://gdpr.eu/checklist/) to help with compliance. There are different laws in place for countries outside the EU. In the digital age, data can be collected across borders easily. You should know the law in the

places you study, even if that is a difficult task. Acting against the terms and conditions in the USA can have legal consequences (due to, for example, the Computer Fraud and Abuse Act); those terms and conditions might include forbidding you to scrape a webpage (Krotov and Silva 2018). Besides, republishing scraped data can be against a website's copyright. Damage to a website through overloading while scraping can also lead to prosecution. South Africa has the Personal Information Act No. 4 (POPIA) in place, which regulates the 'use, management and transfer of personal information' (Mahomed and Labuschaigne 2022, 80).

If you are uncertain whether your research complies with the privacy laws of your country, you should get legal advice.

Institutional Review Boards

Institutional review boards are responsible for checking the researcher's compliance with research ethics (Von Unger and Simon 2016). They originate from the medical field following unethical experiments in the Second World War and the Tuskegee study in the 1970s, where African American men were denied existing treatment to study the progression of syphilis (Von Unger and Simon 2016). The social sciences developed an ethical codex in the 1990s, including political scientists ("Ethik-Kodex" n.d.). Your university should have a review board. For instance, information about the 'Ethikkommission' of the University of Mannheim can be found here: www.uni-mannheim.de/en/about/organization/bodies-and-committees/committees-and-councils/ethics-committee.

Using an API

The following information is important to know when using an API. You need to check the information for the specific API you are utilising, but a lot of the information is transferable to other APIs. X (formerly Twitter) makes its data available via an API, which now costs money. The Genius API is easy to register for and is free. TikTok has a research API, but access is restricted; as a student, you will not get access.

To ensure the safety of platform users and a beneficial use of data, platforms make a 'Developer Policy', 'Developer Agreement' or 'API Terms of Service' available (TikTok 2023). If you are using an API, it is important to check this policy because you have to agree to its terms. When applying for the TikTok Research API, for example, you have to truthfully say what you will use the API for.

When registering for an API, you will get something like an API key. These are like passwords. They are only for your own use; do not share them. For many often-used APIs, R packages are available. They often allow you to save your token in the R environment. You are responsible for ensuring that the data does not get into the hands of others and is stored safely.

The privacy of the platform users is important, and their privacy has to be ensured. TikTok states that research output cannot be published if it can be linked back to an individual, and you cannot distribute any data they make available for you (TikTok 2023).

X places specific data under stricter conditions (note that these regulations can change quickly), which might be relevant for research ("More on Restricted Use Cases – Twitter Developers" n.d.). This includes:

- Health (including pregnancy)
- Negative financial status or condition
- Political affiliation or beliefs
- Racial or ethnic origin
- Religious or philosophical affiliation or beliefs
- Sex life or sexual orientation
- Trade union membership
- Alleged or actual commission of a crime

This data can only be used without personal data, which includes user IDs, names and other identifiers in an aggregated analysis. Other laws have to be taken into account as well.

You should never use the API in a way that decreases its functionality. Some platforms might set rate limits, which you are not allowed to circumvent. Additionally, to make sure you are not sending too many requests in a loop, it is a good idea to set a pause between the requests:

```
Sys.sleep(time = 0.2)
```

A study looking at the Twitter (as it was then) user perspective (Fiesler and Proferes 2018) showed that 61 per cent of the respondents (368 respondents in total) did not know that researchers can use their tweets, even though this is stated in the privacy policy. Of the people questioned, 27 per cent said that they feel uncomfortable if their tweets are used for research. If their whole Twitter history were hypothetically used for research, 21.3 per cent of the respondents felt very uncomfortable, and another 27.2 per cent felt uncomfortable. Important factors that make users accept that their tweets are being used are if they were asked beforehand (67.4 per cent), the topic of the study (56.6 per cent), the size of the dataset (45.7 per cent) and whether other information like profile information or geolocation was not analysed (48.3 per cent). In conclusion, if you are using data through an API, stay within the terms set by the platform and take additional measures to protect the privacy of your subjects. Just because it is legal does not mean your research is ethical.

Scraping Allowed?

To find out if you are allowed to scrape a page, check the terms and conditions. A simple tool to check for each specific webpage is the *robots.txt* files (Meissner 2024). Many pages have them integrated in the domain of a website. With the `robotstxt` package that can be found on cran, you can easily check the existence of such a file:

```
library(robotstxt)
```

It only checks the specific page:

```
paths_allowed("http://google.com/")
## google.com
## [1] TRUE
paths_allowed("http://google.com/search")
## google.com
## [1] FALSE
```

Still check terms and conditions, especially if there is no *robots.txt* file available; it is not required and is not guaranteed to be correct.

Case Study (Carlos Gueiros)

Labelling Political Topics With Transformer Models Using Youtube Text Data

This case study focuses on applying advanced language models to analyse text data from YouTube videos, demonstrating how machine learning can predict topics based on contextual information.

Text Analysis Task

The task at hand is to use text data, such as captions and descriptions, from YouTube videos to predict the topic of the text using a pre-trained language model that takes into account the context of the text. The standard way to get YouTube data on videos, captions, playlists and channels is via 'YouTube Data API v3', which requires a Google account to access the API via R or Python (see https://paulcbauer.github.io/apis_for_social_scientists_a_review and https://developers.google.com/youtube/v3/getting-started for more details). In this example, however, we use an online platform, 'YouTube Data Tools' (Rieder 2015), which facilitates data collection from the YouTube API v3. From there, we can extract all video IDs and descriptions from a channel as a text file, which we can then use to analyse data and retrieve captions using the information in Seo and Choi (2020).

QTA Method Used

We used ParlBERT-Topic-German, which is an adapted version of BERT for the German language fine-tuned on German parliamentary text, following the work of Klamm, Rehbein and Ponzetto (2022). The transformer model serves as the foundation for BERT, which is a neural network that operates on sequences, such as a sequence of words, to generate predictions, such as the topic of the sequence. The following steps were used to label the text:

1 Define a variable with relevant text that contains whole sentences or paragraphs of running text;
2 Apply the pre-trained model to predict text labels among 20 categories (Klamm, Rehbein and Ponzetto 2022).

The full code is available on the GitHub repository for the book: https://github.com/julianbernauer/SAGE-QTAUR.

Illustration of Results

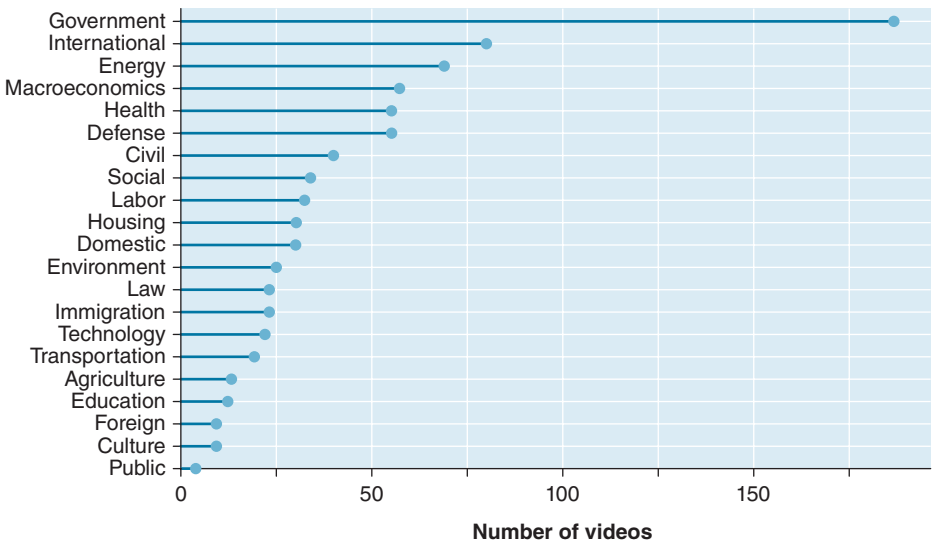

Figure 3.11 Illustration of results

Figure 3.11 illustrates an application of the method described above to the text descriptions of 828 German-language YouTube videos from the German Bundestag YouTube channel in 2022 (data collected using Rieder (2015) and Seo and Choi (2020)). The most common topic, 'Government,' is disproportionately larger than other topics, which may indicate that discussions of the government and internal affairs dominate the videos. It should be noted that the model may overestimate the occurrence of this topic given the political language and context present in the text. Other topics of discussion, such as international affairs, health and defence, were also widely discussed, and this classification is reasonable given the events related to these topics in 2022.

(Continued)

Discussion of Strength and Weaknesses

The method can classify the topics of texts related to YouTube videos from the German Bundestag. However, the results should be viewed primarily as a description of the topics discussed. As with traditional topic modelling, further validation is required to use these topics as measures in social research (Vázquez et al. 2022; Ying, Montgomery and Stewart 2022). Pre-trained transformer models offer a direct and out-of-the-box way to classify and analyse text without much preprocessing and can perform better than other traditional unsupervised topic modelling approaches with word vectors, as they consider the context of texts. However, this approach is more computationally intensive compared to word vectors, and careful consideration of the similarity between the text being classified and the text on which the model was trained is necessary.

CHAPTER SUMMARY

In this chapter, you learned to load different kinds of text data into R, including from *txt* files, from Wikipedia and through the Genius API. We showed you what to look out for when using APIs. Loading data into R is the necessary first step for QTA. We repeated the actions of turning text data from a data frame into a corpus, a token object and a dfm. Now you also know how to remove symbols you might not want in your data, like numbers, URLs or emojis, and you know what regex is. Moreover, you understand how to remove words you don't want and what stopwords are. Further, you learned what web scraping is and how to keep your scraping ethical while respecting privacy laws. Finally, you saw a case study using YouTube data.

Data Sources

- Party manifestos: shiny.mzes.uni-mannheim.de/polidoc and https://manifesto-project.wzb.eu/.
- Genius API/geniusr: https://ewenme.github.io/geniusr/.
- X (Twitter) API: https://developer.x.com/en/docs/x-api.
- Wikipedia data: https://cran.r-project.org/web/packages/getwiki/vignettes/getwiki.html.

Further Reading

See the Quick Start Guide to `quanteda` for more code examples and information on the package developed by Kenneth Benoit, Kohei Watanabe, Haiyan Wang, Paul Nulty, Adam Obeng, Stefan Müller, Akitaka Matsuo, William Lowe and the European Research Council: https://quanteda.io/articles/quickstart.html.

For using text as data for analysing policy positions, see: Laver, Michael, Kenneth Benoit and John Garry. 2003. 'Extracting Policy Positions from Political Texts Using Words as Data'. *American Political Science Review* 97 (2): 311–31.

More information and guidance on web scraping can be found at: Munzert, Simon, Christian Rubba, Peter Meißner and Dominic Nyhuis. 2015. *Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining*. John Wiley & Sons.

For more on the ethics of web scraping, see: Salganik, Matthew J. 2018. '6 Ethics'. In *Bit by Bit – Social Research in the Digital Age*. Princeton University Press. www.bitbybitbook.com/.

Questions

1 Which R package can be helpful to make sure you are allowed to scrape a webpage?

2 Why do we use `paste0` when loading in data based on a set directory and a string?

```
data <- readtext(paste0(datadir,"/exercise3"), encoding="UTF8")
```

3 Sören loads a text file into R with `readtext` and prints the content; this is what he sees:

print("Hi everÿone :Ð My nàme is SÃ¶ren, Î am 24 yeárs old ând I lÔve skiing!")

What did he do wrong?

Exercises

1 Create a corpus from a character vector that consists of multiple texts; create the character vector yourself. Hint: You can link character vectors together into one character vector using `c()`.

2 Go to the polidoc page (shiny.mzes.uni-mannheim.de/polidoc), register (if you like) and download a few party manifestos of your choice (as *.txt* files). Read them into R with the `readtext` package, and create a corpus. Now you can play with that data: add document-level variables, tokenise and create plots. Use the examples from Chapters 2 and 3 as a guide.

3 Sign up for a Genius API (or another API of your choice). Check the terms of use to make sure what you want to do is legal. Store your genius token in R. Now load the lyrics to your favourite song in R.