Text data & sentiment analysis

Lecture 6

Louis SIRUGUE

CPES 2 - Fall 2022

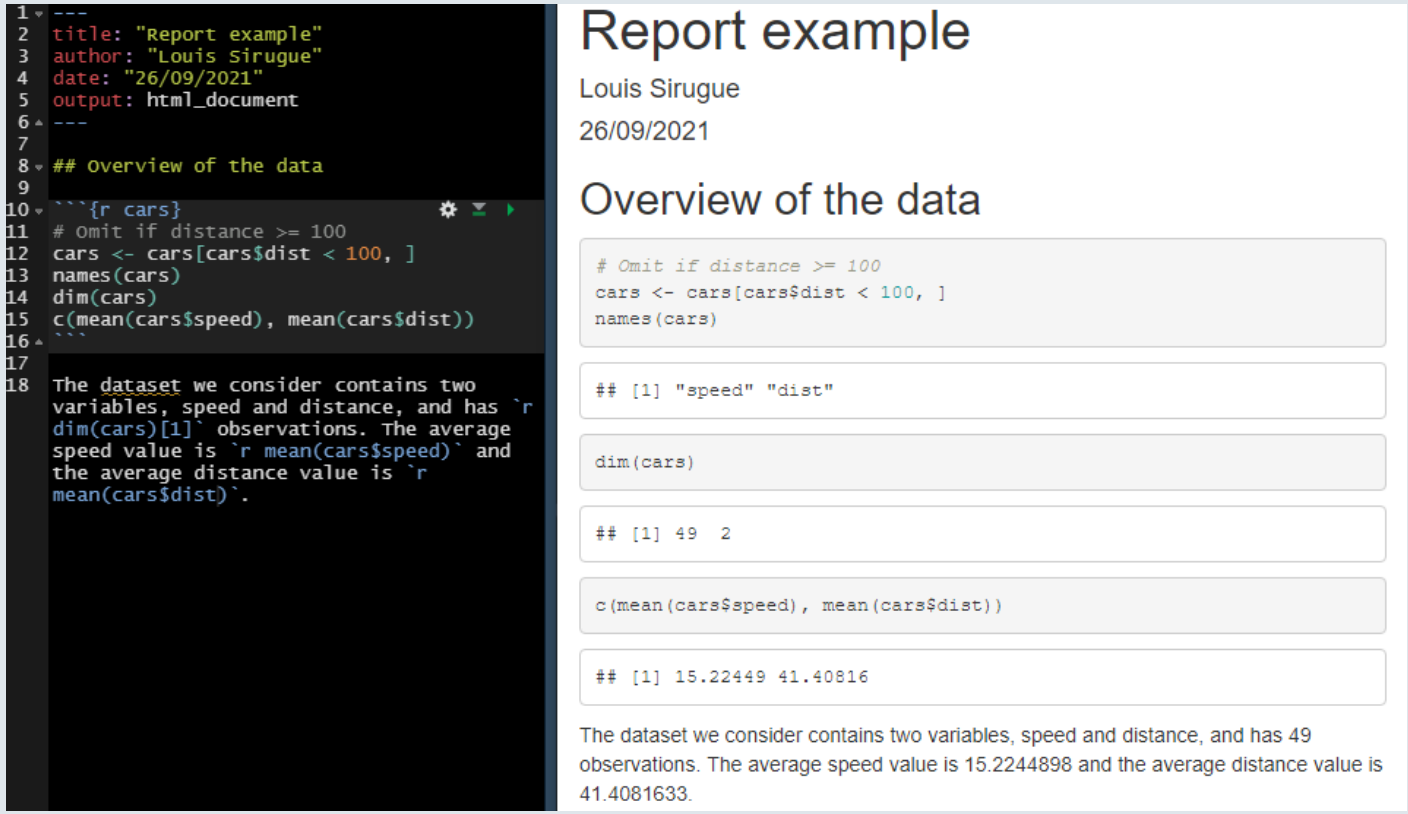
Last time we saw

1. Three types of contents

YAML header →

Code chunks →

Text →



```
---
title: "Report example"
author: "Louis Sirugue"
date: "26/09/2021"
output: html_document
---

## Overview of the data

```{r cars}
# Omit if distance >= 100
cars <- cars[cars$dist < 100, ]
names(cars)
dim(cars)
c(mean(cars$speed), mean(cars$dist))
```

The dataset we consider contains two variables, speed and distance, and has `r dim(cars)[1]` observations. The average speed value is `r mean(cars$speed)` and the average distance value is `r mean(cars$dist)`.
```

Last time we saw

2. Useful features

➜ **Inline code** allows to include the output of some **R code within text areas** of your report

Syntax	Output
`` `paste("a", "b", sep = "-")` ``	`paste("a", "b", sep = "-")`
`` `r paste("a", "b", sep = "-")` ``	a-b

➜ **`kable()`** for clean **html tables** and **`datatable()`** to navigate in **large tables**

```
kable(results_table)
datatable(results_table)
```

Last time we saw

3. LaTeX for equations

- $\LaTeX$ is a convenient way to display **mathematical** symbols and to structure **equations**
  - The **syntax** is mainly based on **backslashes \ and braces {}**

➜ What you **type** in the text area: `$x \neq \frac{\alpha \times \beta}{2}$`

➜ What is **rendered** when knitting the document: $x \neq \frac{\alpha \times \beta}{2}$

To **include** a **LaTeX equation** in R Markdown, you simply have to surround it with the **$ sign**

The mean formula with one `$` on each side

➜ For inline equations

$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$

The mean formula with two `$` on each side

➜ For large/emphasized equations

$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Today: Text data and sentiment analysis

1. Cleaning text data

- 1.1. Exploring the data
- 1.2. Regular expressions
- 1.3. Tokenization

2. Sentiment analysis

- 2.1. Stopwords
- 2.2. Sentiments
- 2.3. Analysis

3. Wrap up!

Today: Text data and sentiment analysis

1. Cleaning text data

- 1.1. Exploring the data
- 1.2. Regular expressions
- 1.3. Tokenization

1. Cleaning text data

1.1. Exploring the data

- Being able to **handle** strings and **text** data can be very useful
  - For **webscrapping**
  - To enlarge your set of observable (from **tweets**, reviews, political speeches/brochures)
  - Even with **standard data** containing character variables

- But text data is **quite complicated** to handle
  - **Not as codified** as conventional datasets
  - Can take **various formats**
  - Usually **very messy**

- The most **tedious** part of text-data analysis is **data cleaning**
  - The key tool for that purpose is **regular expressions**
  - Today we're giving it a go by doing a **sentiment analysis**

➜ Let's do a sentiment analysis on Romeo and Juliet by Shakespeare

1. Cleaning text data

1.1. Exploring the data

- The first step is to have a **look at the data**
  - For the type of data we work with today it is particularly easy: just **open the .txt** file in a notepad

➜ Open `romeo_and_juliet.txt` (you can view it here)

- The .txt file is organized as follows
  - Some general information about the file
  - The table of contents
  - The *dramatis personæ*
  - **The play**
  - Some copyright considerations
- Note that **stage directions** are mentioned **[within brackets]**

➜ To start working on it we should import the text in R

1. Cleaning text data

1.1. Exploring the data

- The function to read .txt files is **readLines()**
  - Don't forget to specify the correct **encoding**
  - At the beggining of the file is indicated *"Character set encoding: UTF-8"*

```
raj <- readLines("shakespeare/romeo_and_juliet.txt", encoding = "UTF-8")
```

Let's take a look at how this file is stored:

```
summary(raj)
```

```
##    Length     Class      Mode
##      5640 character character
```

```
raj[1]
```

```
## [1] "<U+FEFF>The Project Gutenberg eBook of Romeo and Juliet, by William Shakespeare"
```

1. Cleaning text data

1.1. Exploring the data

- readLines() stored the data as a **vector** containing **5,640** strings, one for every **line** of the file
  - To handle the data conveniently, we should put it in a **database** format

```
raj <- tibble(line = raj)
head(raj, 10)
```

```
## # A tibble: 10 x 1
##    line
##    <chr>
##  1 "<U+FEFF>The Project Gutenberg eBook of Romeo and Juliet, by William Shakespeare"
##  2 ""
##  3 "This eBook is for the use of anyone anywhere in the United States and"
##  4 "most other parts of the world at no cost and with almost no restrictions"
##  5 "whatsoever. You may copy it, give it away or re-use it under the terms"
##  6 "of the Project Gutenberg License included with this eBook or online at"
##  7 "www.gutenberg.org. If you are not located in the United States, you"
##  8 "will have to check the laws of the country where you are located before"
##  9 "using this eBook."
## 10 ""
```

1. Cleaning text data

1.1. Exploring the data

- Now we need to **get rid of** what comes **before and after the play**
  - The play starts at the second occurrence of "ACT I" (the first one being in the contents)
  - Let's identify the corresponding line and remove everything before that
- First, let's store the row numbers of every line that states `"ACT I"`:

```
beginning <- raj %>%
  mutate(line_number = row_number()) %>%
  filter(line == "ACT I")
```

```
beginning
```

```
## # A tibble: 2 x 2
##   line  line_number
##   <chr>       <int>
## 1 ACT I          40
## 2 ACT I         144
```

- There are indeed 2 occurrences of "ACT 1":
  - One at line 40 in the table of contents
  - And one at **line 144** where the **play starts**

1. Cleaning text data

1.1. Exploring the data

- We can thus get rid of every line whose row number is below that of the second occurrence of `"ACT 1"`:

```
raj <- raj %>% filter(row_number() >= beginning$line_number[2])
head(raj, 10)
```

```
## # A tibble: 10 x 1
##    line
##    <chr>
##  1 "ACT I"
##  2 ""
##  3 "SCENE I. A public place."
##  4 ""
##  5 " Enter Sampson and Gregory armed with swords and bucklers."
##  6 ""
##  7 "SAMPSON."
##  8 "Gregory, on my word, we'll not carry coals."
##  9 ""
## 10 "GREGORY."
```

1. Cleaning text data

1.1. Exploring the data

- Note that proceeding this ways allows to **automatize the process** for other plays
  - Looking at the line number in the data to remove what's before wouldn't be transposable
  - But this code can be applied directly to other plays (see macbeth, othello, ...)
  - We'll do so **for the whole data cleaning** so that we can clean other plays with virtually no additional code

- What all plays have in **common** at the end is a **final stage direction**
  - Romeo and Juliet: [_Exeunt._]
  - Macbeth: [_Flourish. Exeunt._]
  - Othello: [_Exeunt._]
  - A midsummer night's dream: [_Exit._]
  - ...

- But how to get the line number of the final stage direction?
  - The last stage direction is not always the same
  - We need to use **regular expressions**!

1. Cleaning text data

1.2. Regular expressions

- **Regular expressions** are used to identify strings that **match a given pattern**
  - Extremely useful tool when analyzing **text data**
  - Used in most programming languages, **not specific to R**

- In practice regular expressions are **strings of codified characters** describing a pattern
  - For instance the character **"^"** indicates the **start of the string**
  - So the regular expression "^a" would match any "a" that is a the beginning of a string

- Regular expressions in R can be used in different functions with different purposes:
  - **grep:** returns elements that match the regexp
  - **grepl:** returns TRUE for elements that match the regexp and FALSE otherwise
  - **gsub:** replaces the elements that match the regexp with what you want
  - ...

➜ Let's play around with regexp to get the idea

1. Cleaning text data

1.2. Regular expressions

- Consider the following vector:

```
txt <- c("One", "two", "three", "four", "5", "6", "7even", "Eight")
```

- How to **find** all the elements that **start with "t"?**
  - We can use the regular expression **"^t"**
  - And use **grepl** to know for every element whether it matches this pattern or not

```
grepl("^t", txt)
```

```
## [1] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
grepl("^th", txt)
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

1. Cleaning text data

1.2. Regular expressions

- Using **`grep`** instead or `grepl` will **return the indices** of the strings that match the pattern

```
grep("^f", txt)
```

```
## [1] 4
```

- Specifying `value = TRUE` will **return the values** instead of the indices

```
grep("^f", txt, value = T)
```

```
## [1] "four"
```

- Using **`gsub`** allows to **replace the pattern** by something else

```
gsub("^f", "4", txt)
```

```
## [1] "One"   "two"   "three" "4our"  "5"     "6"     "7even" "Eight"
```

1. Cleaning text data

1.2. Regular expressions

Regexp	Meaning
^	Start of string (or 'not')
$	End of string
.	Any character
*	0 or more occurences
+	1 or more occurences
{n}	n occurences
{n,}	n or more occurences
{n,m}	between n and m occurences

Regexp	Meaning
[]	A range of characters
[^abc]	Not a, b or c
[a-z]	Any lowercase letter from a to z
[A-Z]	Any capital letter from A to Z
[0-9]	Any digit from 0 to 9

1. Cleaning text data

1.2. Regular expressions

- Thus, if we do not want to replace only any "f" that is in first position but any string starting with "f"
  - **"^f":** f in first position
  - **"^f.":** f in first position followed by any character
  - **"^f.+":** f in first position followed by any occurrence of any character

```
gsub("^f.+", "4", txt)
```

```
## [1] "One"   "two"   "three" "4"     "5"     "6"     "7even" "Eight"
```

- Other examples

```
txt <- c("One", "two", "three", "four", "5", "6", "7even", "Eight")
```

```
grep("e$", txt)
```

```
## [1] 1 3
```

```
grep(".e", txt)
```

```
## [1] 1 3 7
```

```
grep(".e.", txt)
```

```
## [1] 3 7
```

```
grep("[0-9]e", txt)
```

```
## [1] 7
```

1. Cleaning text data

1.2. Regular expressions

- It is also possible to use the **logical operators** `&` (and) and `|` (or)

```
grep("e$|o$", txt, value = T)
```

```
## [1] "One"   "two"   "three"
```

- To use **symbols** such as `^`, `$`, `.`, `&` as characters instead of operators, they **should be preceded by `\\`**

```
grep("^^", c("^a", "b", "^c", "^d", "e", "f"), value = T)
```

```
## [1] "^a" "b"  "^c" "^d" "e"  "f"
```

```
grep("^\\^", c("^a", "b", "^c", "^d", "e", "f"), value = T)
```

```
## [1] "^a" "^c" "^d"
```

Practice

➜ Use `grepl` to create a variable that identifies every line that contains a (complete) stage direction

```
raj <- raj %>%
  mutate(direction = grepl("....", line))
```

- Remember that stage directions are in brackets `[ ]`

```
# Read the play
raj <- tibble(line = readLines("shakespeare/romeo_and_juliet.txt", encoding = "UTF-8"))

# Identify the lines "ACT I"
beginning <- raj %>%
  mutate(line_number = row_number()) %>%
  filter(line == "ACT I")

# Remove everything before the second occurence
raj <- raj %>% filter(row_number() >= beginning$line_number[2])
```

You've got 10 minutes!

Solution

- Basically we're looking for strings containing **"[something]"**
  - The **"[" and "]"** symbols should be **preceded by "\\"**
  - And the "something" translates into **".+"**, *i.e.*, any character any number of times

```
raj <- raj %>%
  mutate(direction = grepl("\\[.+\\]", line))
```

```
head(raj %>% filter(direction), 8)
```

```
## # A tibble: 8 x 2
##   line                                                                direction
##   <chr>                                                               <lgl>
## 1 " [_They fight._]"                                                  TRUE
## 2 " [_Beats down their swords._]"                                     TRUE
## 3 " [_They fight._]"                                                  TRUE
## 4 " [_Exeunt Montague and Lady Montague._]"                           TRUE
## 5 " [_Going._]"                                                       TRUE
## 6 " [_Exeunt._]"                                                      TRUE
## 7 "Whose names are written there, [_gives a paper_] and to them say," TRUE
## 8 " [_Exeunt Capulet and Paris._]"                                    TRUE
```

1. Cleaning text data

1.2. Regular expressions

- We can now find the last stage direction

```
end <- raj %>% # Do the computations separately for stage direction lines and other lines
  group_by(direction) %>%

  mutate(last_obs = row_number() == n()) %>% # Mark the last row of each group with TRUE

  ungroup() %>% # Ungroup the data

  mutate(line_number = row_number()) %>% # Create a line_number variable

  filter(direction & last_obs) # Keep the last stage direction
```

```
end
```

```
## # A tibble: 1 x 4
##   line           direction last_obs line_number
##   <chr>          <lgl>     <lgl>          <int>
## 1 " [_Exeunt._]" TRUE      TRUE            5141
```

1. Cleaning text data

1.2. Regular expressions

```
raj <- raj %>% filter(row_number() <= end$line_number)
```

- The play is now properly delimited:

```
kable(head(raj, 5), "Start of the play")
```

Start of the play

line	direction
ACT I	FALSE
	FALSE
SCENE I. A public place.	FALSE
	FALSE
Enter Sampson and Gregory armed with swords and bucklers.	FALSE

1. Cleaning text data

1.2. Regular expressions

```
kable(tail(raj, 8), "End of the play")
```

End of the play

line	direction
A glooming peace this morning with it brings;	FALSE
The sun for sorrow will not show his head.	FALSE
Go hence, to have more talk of these sad things.	FALSE
Some shall be pardon’d, and some punished,	FALSE
For never was a story of more woe	FALSE
Than this of Juliet and her Romeo.	FALSE
	FALSE
[_Exeunt._]	TRUE

- We should also remove empty lines:

```
raj <- raj %>% filter(line != "")
```

1. Cleaning text data

1.3. Tokenization

- But the data is not ready yet, we need to **tokenize** it first
  - **Tokenization** is the fact of cleaning the data so that there is **one unit of text per row**
  - Like in a regular database where each row corresponds to an observation

- A token (unit of text) can be:
  - A character
  - A letter
  - A word
  - A sentence
  - etc.

- In our case it would be great to **tokenize** the data at the **line level**, documenting for each line:
  - The corresponding act
  - The corresponding scene
  - The corresponding character

1. Cleaning text data

1.3. Tokenization

- We can start by identifying the **act and scene delimiters**

```
raj <- raj %>%
  mutate(act_delim = grepl("^ACT", line),
         scene_delim = grepl("^SCENE", line))
```

- Identifying the **line delimiters** is more complicated:
  - There's no systematic word like "ACT" or "SCENE"
  - But they have the specificity to be in **uppercase** and to **end with a dot**
  - They can also contain a space and the character '

```
raj <- raj %>%
  mutate(line_delim = grepl("^[A-Z ']*\\.$", line))
```

➜ We should check it worked

1. Cleaning text data

1.3. Tokenization

```
library("DT")
datatable(raj %>% filter(act_delim|scene_delim), options = list(pageLength = 6))
```

Show 6 entries Search:

	line	direction	act_delim	scene_delim	line_delim
1	ACT I	false	true	false	false
2	SCENE I. A public place.	false	false	true	false
3	SCENE II. A Street.	false	false	true	false
4	SCENE III. Room in Capulet’s House.	false	false	true	false
5	SCENE IV. A Street.	false	false	true	false
6	SCENE V. A Hall in Capulet’s House.	false	false	true	false

Showing 1 to 6 of 29 entries Previous 1 2 3 4 5 Next

1. Cleaning text data

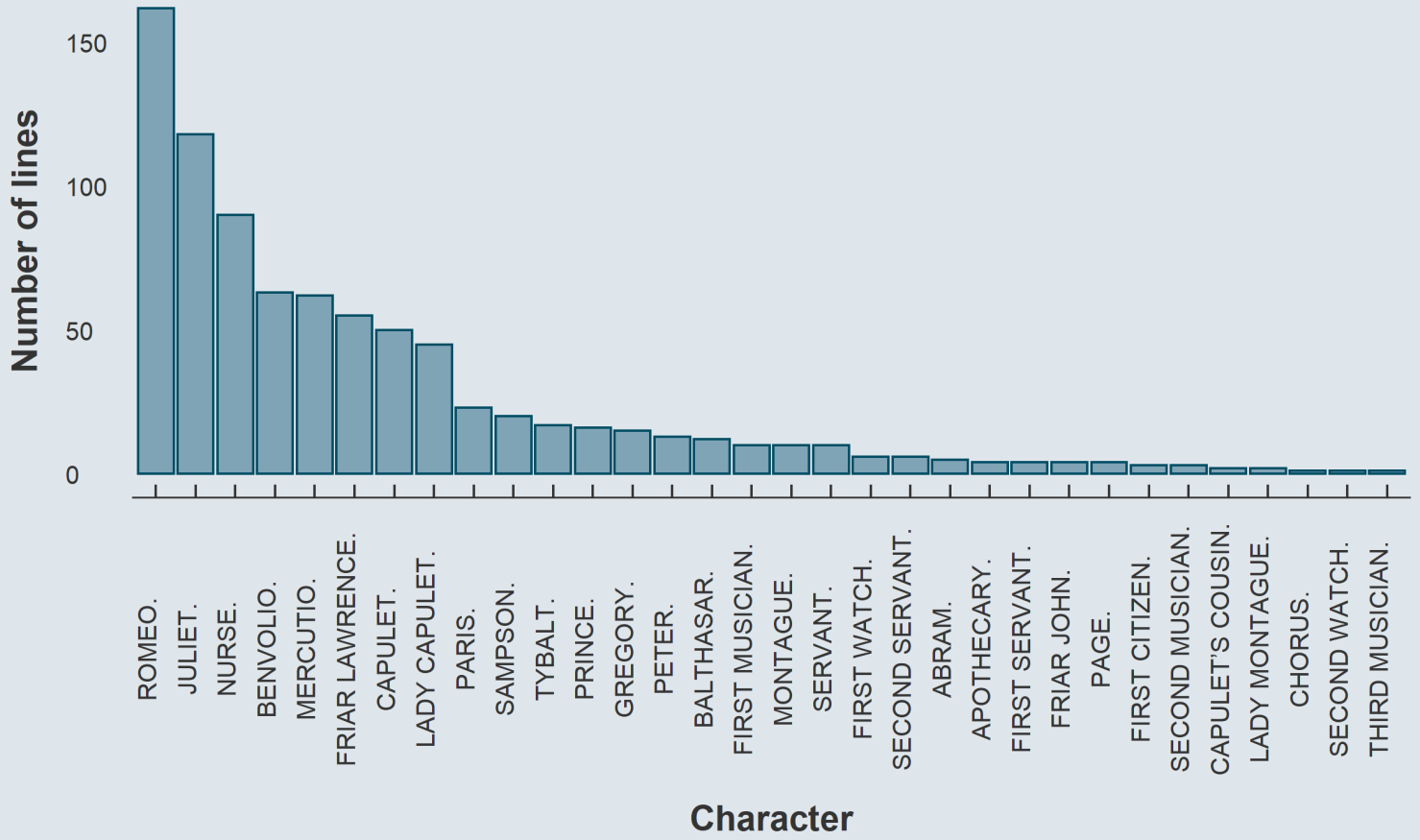
1.3. Tokenization

- We indeed observe the same table of contents as in the preamble here
- What about the characters?
  - Let's compute the number of **lines per character**

```
raj %>%
  # Keep only the line delimiters (character names)
  filter(line_delim) %>%
  # Group by character
  group_by(line) %>%
  # Count the number of line (creates variable n)
  tally() %>%
  # Plot it
  ggplot(., aes(x = reorder(line, -n), y = n)) +
  geom_bar(stat = "identity") +
  xlab("Character") + ylab("Number of lines") +
  theme(axis.text.x = element_text(angle = 90))
```

1. Cleaning text data

1.3. Tokenization



1. Cleaning text data

1.3. Tokenization

```
kable(head(raj, 8), "Head of the data")
```

Head of the data

line	direction	act_delim	scene_delim	line_delim
ACT I	FALSE	TRUE	FALSE	FALSE
SCENE I. A public place.	FALSE	FALSE	TRUE	FALSE
Enter Sampson and Gregory armed with swords and bucklers.	FALSE	FALSE	FALSE	FALSE
SAMPSON.	FALSE	FALSE	FALSE	TRUE
Gregory, on my word, we’ll not carry coals.	FALSE	FALSE	FALSE	FALSE
GREGORY.	FALSE	FALSE	FALSE	TRUE
No, for then we should be colliers.	FALSE	FALSE	FALSE	FALSE
SAMPSON.	FALSE	FALSE	FALSE	TRUE

1. Cleaning text data

1.3. Tokenization

- We managed to **identify** the indicators of **act/scene/line**
  - But the data is **not tokenized**
  - We want one row per line
  - And the corresponding act/scene/character of each line

- One way to do that would be to:
  - Start counters of act/scene/line
  - **Go through each row** of the data
  - Each time we cross a marker, increase the counter

- We can create **empty variables** that we will **fill in progressively**:

```
raj <- raj %>%
  mutate(id_act = NA,
         id_scene = NA,
         id_line = NA,
         id_char = NA)
```

1. Cleaning text data

1.3. Tokenization

- These vectors should be filled **row** after row with the corresponding values
  - We should first **initialize** the counters that we will **update** each time we pass a marker

```
temp_act <- 0
temp_scene <- 0
temp_line <- 0
temp_char <- ""
```

- We're all set to **start the loop**

```
for (i in 1:nrow(raj)) {

  # Update counters
  if (raj[i, "act_delim"] == TRUE) { }
  if (raj[i, "scene_delim"] == TRUE) { }
  if (raj[i, "line_delim"] == TRUE) { }

  # Fill the vectors
```

1. Cleaning text data

1.3. Tokenization

- Each time we pass an act marker we should
  - Increase the act counter
  - Reset the scene counter
  - Reset the line counter

```
for (i in 1:nrow(raj)) {

  if (raj[i, "act_delim"] == TRUE) {
    temp_act <- temp_act + 1
    temp_scene <- 0
    temp_line <- 0
  }
```

- The same applies to the scene/line/character counters
- After what every updated counter should be stored in its vector

1. Cleaning text data

1.3. Tokenization

- Update counters each time we pass a scene/line marker and store all counters

```
  if (raj[i, "scene_delim"] == TRUE) {
    temp_scene <- temp_scene + 1
    temp_line <- 0
  }
  if (raj[i, "line_delim"] == TRUE) {
    temp_line <- temp_line + 1
    temp_char <- gsub(pattern = "\\.$", "", raj[i, "line"])
  }

  raj[i, "id_act"] <- temp_act
  raj[i, "id_scene"] <- temp_scene
  raj[i, "id_line"] <- temp_line
  raj[i, "id_char"] <- temp_char
}

kable(head(raj, 7), caption = "")
```

1. Cleaning text data

1.3. Tokenization

line	direction	act_delim	scene_delim	line_delim	id_act	id_scene	id_line	id_char
ACT I	FALSE	TRUE	FALSE	FALSE	1	0	0	
SCENE I. A public place.	FALSE	FALSE	TRUE	FALSE	1	1	0	
Enter Sampson and Gregory armed with swords and bucklers.	FALSE	FALSE	FALSE	FALSE	1	1	0	
SAMPSON.	FALSE	FALSE	FALSE	TRUE	1	1	1	SAMPSON
Gregory, on my word, we’ll not carry coals.	FALSE	FALSE	FALSE	FALSE	1	1	1	SAMPSON
GREGORY.	FALSE	FALSE	FALSE	TRUE	1	1	2	GREGORY
No, for then we should be colliers.	FALSE	FALSE	FALSE	FALSE	1	1	2	GREGORY

1. Cleaning text data

1.3. Tokenization

- We can now keep only the rows whose line id is positive
  - It removes everything that comes before the first line of a scene such as act and scene indicators
- And remove all the rows indicating the characters
  - Because we now have a column indicating the corresponding character for each line

```
raj <- raj %>% filter(id_line > 0 & !line_delim)
kable(head(raj, 3), "")
```

line	direction	act_delim	scene_delim	line_delim	id_act	id_scene	id_line	id_char
Gregory, on my word, we'll not carry coals.	FALSE	FALSE	FALSE	FALSE	1	1	1	SAMPSON
No, for then we should be colliers.	FALSE	FALSE	FALSE	FALSE	1	1	2	GREGORY
I mean, if we be in choler, we'll draw.	FALSE	FALSE	FALSE	FALSE	1	1	3	SAMPSON

1. Cleaning text data

1.3. Tokenization

- But there are still **lines spanning on multiple rows**
  - We need to **paste together** all the rows that correspond to a same line
  - We can use **group_by(id_act, id_scene, id_line)** to do the operation **for each line**
  - And use **paste()** in the **summarise()** function to paste all the rows of a given line

```
raj <- raj %>%
  # Do the computations separately for each line
  group_by(id_act, id_scene, id_line, id_char) %>%
  # Paste together all the rows of each line
  summarise(line = paste(line, collapse = " ")) %>%
  # Ungroup the data for future computations
  ungroup()
```

- Let's browse the data

```
datatable(raj, options = list(pageLength = 5))
```

1. Cleaning text data

1.3. Tokenization

Show 5 entries Search:

	id_act	id_scene	id_line	id_char	line
1	1	1	1	SAMPSON	Gregory, on my word, we'll not carry coals.
2	1	1	2	GREGORY	No, for then we should be colliers.
3	1	1	3	SAMPSON	I mean, if we be in choler, we'll draw.
4	1	1	4	GREGORY	Ay, while you live, draw your neck out o' the collar.
5	1	1	5	SAMPSON	I strike quickly, being moved.

Showing 1 to 5 of 837 entries Previous 1 2 3 4 5 ... 168 Next

1. Cleaning text data

1.3. Tokenization

- The last thing to do is to **remove stage directions**

Example:

```
kable(raj %>% filter(id_act == 1 & id_scene == 2 & id_line == 18), caption = "")
```

id_act	id_scene	id_line	id_char	line
1	2	18	ROMEO	Stay, fellow; I can read. [_He reads the letter._] _Signior Martino and his wife and daughters; County Anselmo and his beauteous sisters; The lady widow of Utruvio; Signior Placentio and his lovely nieces; Mercutio and his brother Valentine; Mine uncle Capulet, his wife, and daughters; My fair niece Rosaline and Livia; Signior Valentio and his cousin Tybalt; Lucio and the lively Helena. _ A fair assembly. [_Gives back the paper_] Whither should they come?

1. Cleaning text data

1.3. Tokenization

- We can do it using the **gsub()** function
  - Let's try with the regexp we used to detect stage directions

```
raj %>%
  mutate(line = gsub("\\[.+\\]", "", line)) %>%
  filter(id_act == 1 & id_scene == 2 & id_line == 18) %>%
  kable(., caption = "")
```

id_act	id_scene	id_line	id_char	line
1	2	18	ROMEO	Stay, fellow; I can read. Whither should they come?

- It **removed everything** between the first [and the last] of the line
  - But we want it to remove the two **stage directions separately**

- We should change *"any character"*: **"."**
  - By *"not [nor]"*: **"[^\\[\\]]"**

1. Cleaning text data

1.3. Tokenization

```
raj <- raj %>%
  mutate(line = gsub("\\[[^\\[\\]+\\]", "", line))
```

```
kable(raj %>% filter(id_act == 1 & id_scene == 2 & id_line == 18), caption = "")
```

id_act	id_scene	id_line	id_char	line
1	2	18	ROMEO	Stay, fellow; I can read. _Signior Martino and his wife and daughters; County Anselmo and his beauteous sisters; The lady widow of Utruvio; Signior Placentio and his lovely nieces; Mercutio and his brother Valentine; Mine uncle Capulet, his wife, and daughters; My fair niece Rosaline and Livia; Signior Valentio and his cousin Tybalt; Lucio and the lively Helena. _ A fair assembly. Whither should they come?

➜ It worked, we're finally done

Overview

1. Cleaning text data ✓

- 1.1. Exploring the data
- 1.2. Regular expressions
- 1.3. Tokenization

2. Sentiment analysis

- 2.1. Stopwords
- 2.2. Sentiments
- 2.3. Analysis

3. Wrap up!

Overview

1. Cleaning text data ✓

- 1.1. Exploring the data
- 1.2. Regular expressions
- 1.3. Tokenization

2. Sentiment analysis

- 2.1. Stopwords
- 2.2. Sentiments
- 2.3. Analysis

2. Sentiment analysis

2.1. Stopwords

- We **now** have clean data at the **line level**
  - But **sentiment analyses** are usually performed at the **word level**
  - The idea is to use a **dictionary** that attributes a **sentiment** to each (some) words

➜ To **tokenize** our data at the word level, we can use the **`unnest_token()`** function from the `tidytext` package
  - It will attribute one row to each word of each line
  - Put everything in lower case
  - And remove punctuation

```
library("tidytext")
raj <- raj %>%
  mutate(to_unnest = line) %>%
  unnest_tokens(token = "words", input = to_unnest, output = word)
```

- Let's have a look

```
kable(head(raj, 9), "Unnested data")
```

2. Sentiment analysis

2.1. Stopwords

Unnested data

id_act	id_scene	id_line	id_char	line	word
1	1	1	SAMPSON	Gregory, on my word, we'll not carry coals.	gregory
1	1	1	SAMPSON	Gregory, on my word, we'll not carry coals.	on
1	1	1	SAMPSON	Gregory, on my word, we'll not carry coals.	my
1	1	1	SAMPSON	Gregory, on my word, we'll not carry coals.	word
1	1	1	SAMPSON	Gregory, on my word, we'll not carry coals.	we'll
1	1	1	SAMPSON	Gregory, on my word, we'll not carry coals.	not
1	1	1	SAMPSON	Gregory, on my word, we'll not carry coals.	carry
1	1	1	SAMPSON	Gregory, on my word, we'll not carry coals.	coals
1	1	2	GREGORY	No, for then we should be colliers.	no

2. Sentiment analysis

2.1. Stopwords

- The **first step** of a sentiment analysis is usually to **get rid of *stopwords***
  - Stopwords are common words that do not carry much semantic meaning
  - These words take space and computing time without adding to the analysis, so we drop them

➜ We can use the list of stopwords from the `tidytext` package with **`get_stopwords()`**

```
get_stopwords()[["word"]][1:50]
```

```
##  [1] "i"          "me"         "my"         "myself"     "we"
##  [6] "our"        "ours"       "ourselves"  "you"        "your"
## [11] "yours"      "yourself"   "yourselves" "he"         "him"
## [16] "his"        "himself"    "she"        "her"        "hers"
## [21] "herself"    "it"         "its"        "itself"     "they"
## [26] "them"       "their"      "theirs"     "themselves" "what"
## [31] "which"      "who"        "whom"       "this"       "that"
## [36] "these"      "those"      "am"         "is"         "are"
## [41] "was"        "were"       "be"         "been"       "being"
## [46] "have"       "has"        "had"        "having"     "do"
```

2. Sentiment analysis

2.1. Stopwords

- We want to **remove** every row that corresponds to a **stopword** to **reduce** the **dimensionality** of the data

```
nrow(raj)
```

```
## [1] 24156
```

- We can do so using the `anti_join()` function:

```
raj <- raj %>%
  anti_join(get_stopwords())

nrow(raj)
```

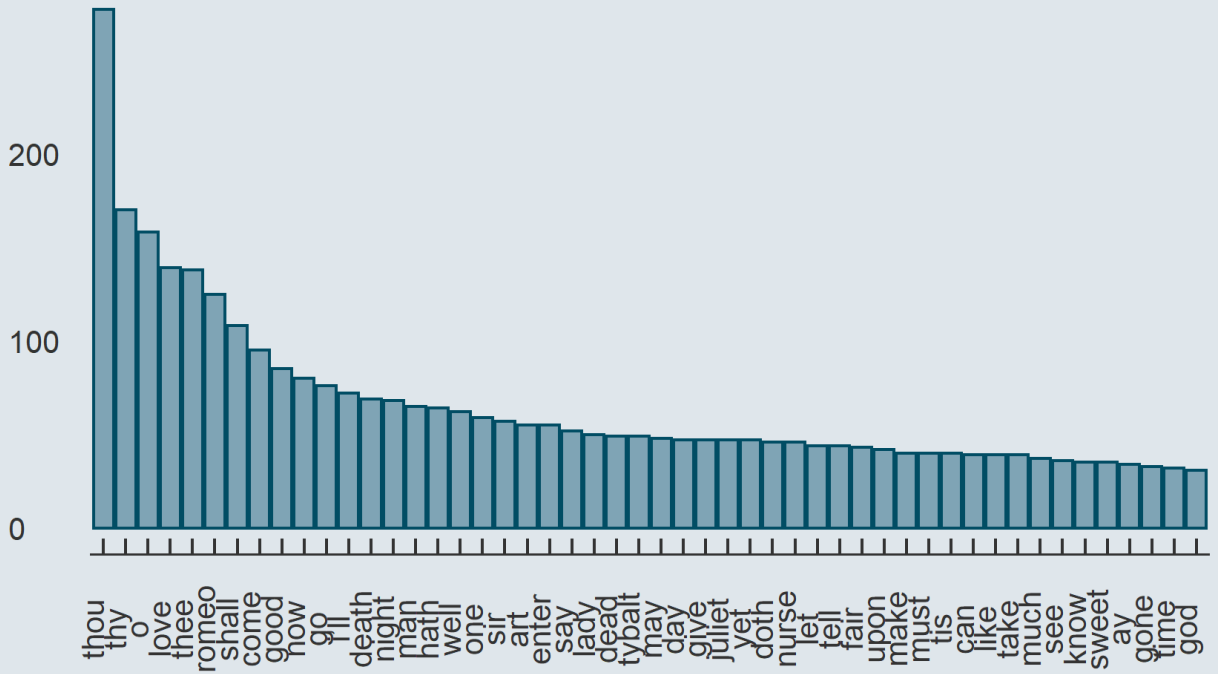
```
## [1] 13037
```

➜ It reduced the number of rows by almost half!

2. Sentiment analysis

2.1. Stopwords

- Here are the 50 **most common words** in the piece after removing the stopwords from the list



- **Some stopwords** remained, in particular archaic pronouns that were **not in our list** such as thou, thy, thee, ...

- But we can already see that **love**, Romeo, night, death, are among the **most frequent words** in the play

2. Sentiment analysis

2.2. Sentiments

- The next step is to **join** the words to their **corresponding sentiments** using a **dictionary**
  - Some dictionaries are very simple: positive/negative
  - And some are more elaborate: trust/fear/sadness/anger/...
- The `tidytext` packages contains several sentiment dictionaries:

```
head(get_sentiments("bing"))
```

```
## # A tibble: 6 x 2
##   word       sentiment
##   <chr>      <chr>
## 1 2-faces    negative
## 2 abnormal   negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
## 6 abominate  negative
```

```
unique(get_sentiments("bing")[["sentiment"]])
```

```
## [1] "negative" "positive"
```

```
unique(get_sentiments("nrc")[["sentiment"]])
```

```
##  [1] "trust"        "fear"         "negative"
##  [6] "surprise"     "positive"     "disgust"
```

2. Sentiment analysis

2.2. Sentiments

- We're gonna use the **`afinn`** dictionary that rates words with integers from **-5 (negative) to 5 (positive)**

```
raj <- raj %>% left_join(get_sentiments("afinn"))
summary(raj$value)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -5.000  -2.000   1.000   0.258   2.000   4.000   11172
```

- *Notice that most words have no associated sentiment*

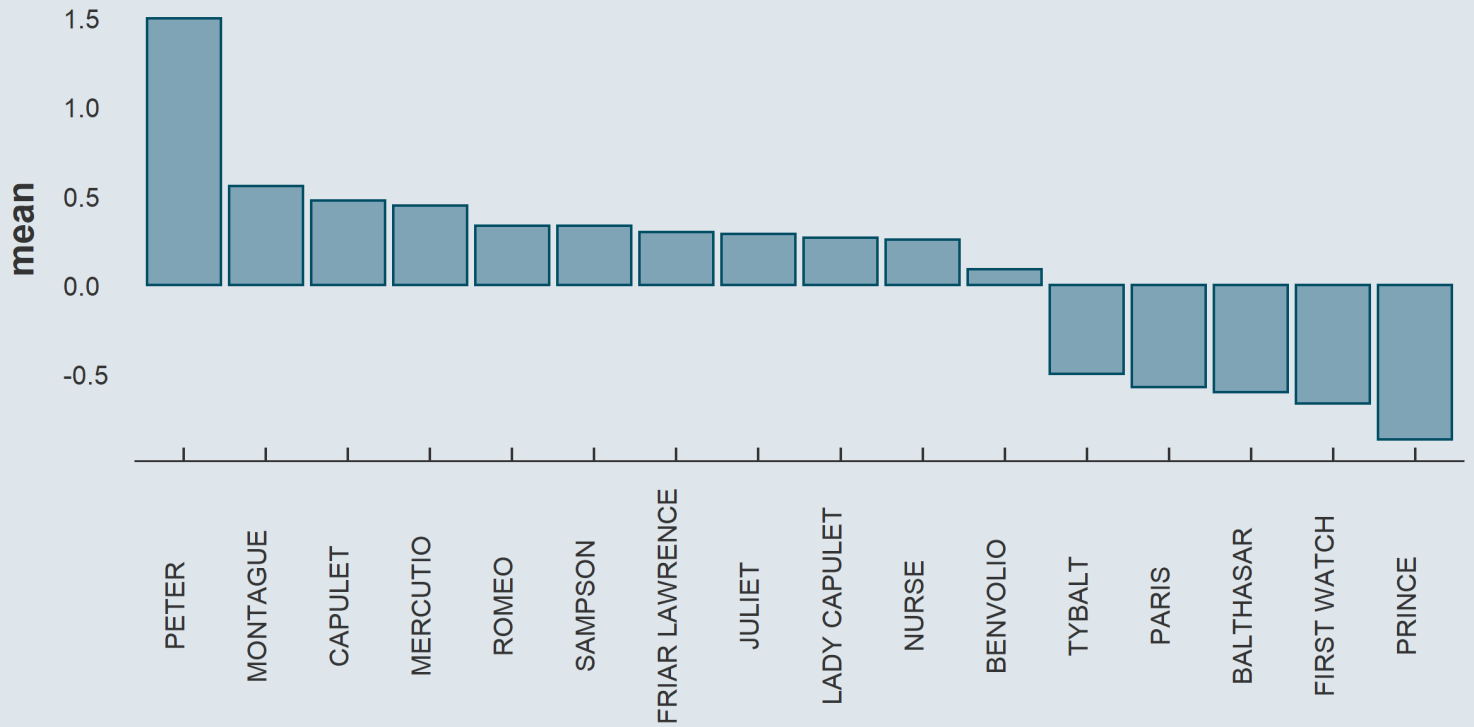
➜ Let's start by computing the average sentiment for the main characters

```
raj %>% group_by(id_char) %>%
  summarise(mean = mean(value, na.rm = T), n_words = n()) %>% filter(n_words > 100) %>%
  ggplot(., aes(x = reorder(id_char, -mean), y = mean)) +
  geom_bar(stat = "identity", fill = "#6794A7", color = "#014D64", alpha = .8) +
  theme(axis.text.x = element_text(angle = 90)) + xlab("")
```

2. Sentiment analysis

2.3. Analysis

- Average sentiment for the main characters



2. Sentiment analysis

2.3. Analysis

- We can also look at the sentiment of the lines of the main characters when they mention other characters

```
raj %>%
  filter(id_char %in% c("ROMEO", "JULIET", "NURSE")) %>%
  group_by(id_char) %>%
  summarise(about_romeo = mean(ifelse(grepl(pattern = "Romeo", line), value, NA), na.rm = T),
            about_juliet = mean(ifelse(grepl(pattern = "Juliet", line), value, NA), na.rm = T),
            about_nurse = mean(ifelse(grepl(pattern = "Nurse", line), value, NA), na.rm = T)) %>%
  kable(., caption = "Crossed sentiments")
```

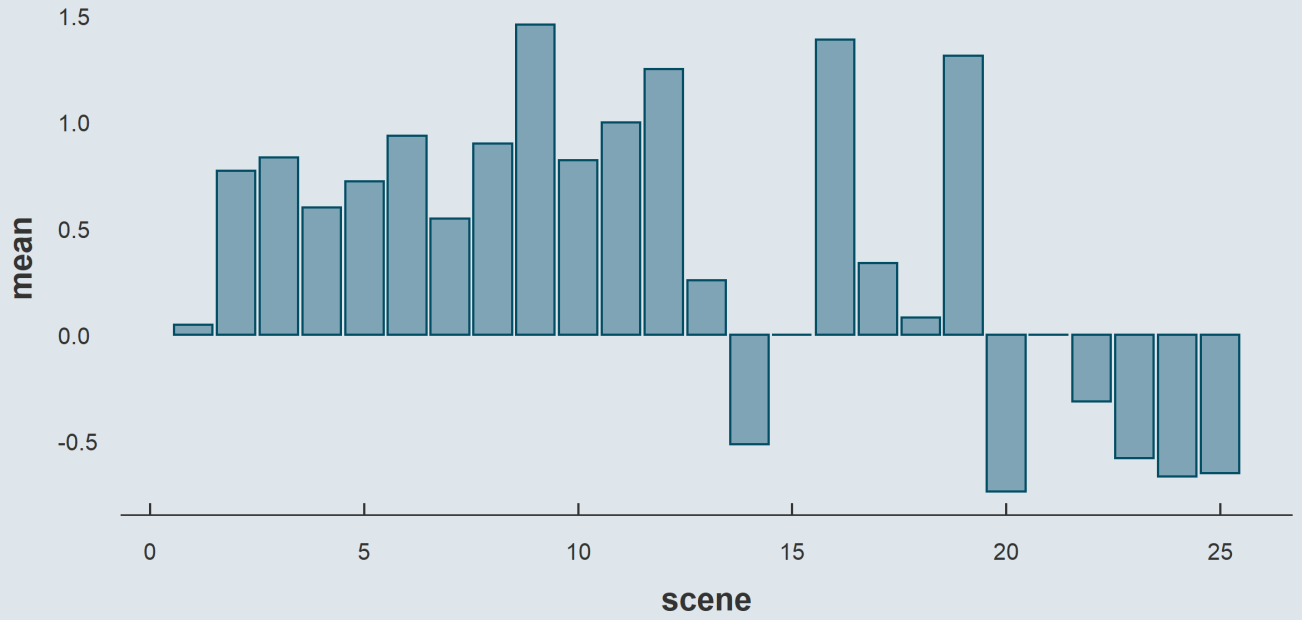
Crossed sentiments

id_char	about_romeo	about_juliet	about_nurse
JULIET	-0.26	-1.50	0.25
NURSE	0.83	-0.27	0.11
ROMEO	-0.74	-0.07	2.40

2. Sentiment analysis

2.3. Analysis

- We can also look at the evolution of the sentiment over the play




```
raj %>%
  group_by(id_act, id_scene) %>%
  summarise(
    mean = mean(value, na.rm = T)
    ) %>% ungroup() %>%
  mutate(scene = row_number()) %>%
  ggplot(aes(x = scene,y = mean))+
  geom_bar(stat = 'identity',
           fill = "#6794A7",
           color = "#014D64",
           alpha = .8)
```

➜ Not a happy end

2. Sentiment analysis

2.3. Analysis

- We can put our code into a **function** and **apply** it to **other plays**
  - Create a function sentiment_evolution() that takes the file name as an argument
  - And that return the evolution of positivity over the play as the output
  - See the code here

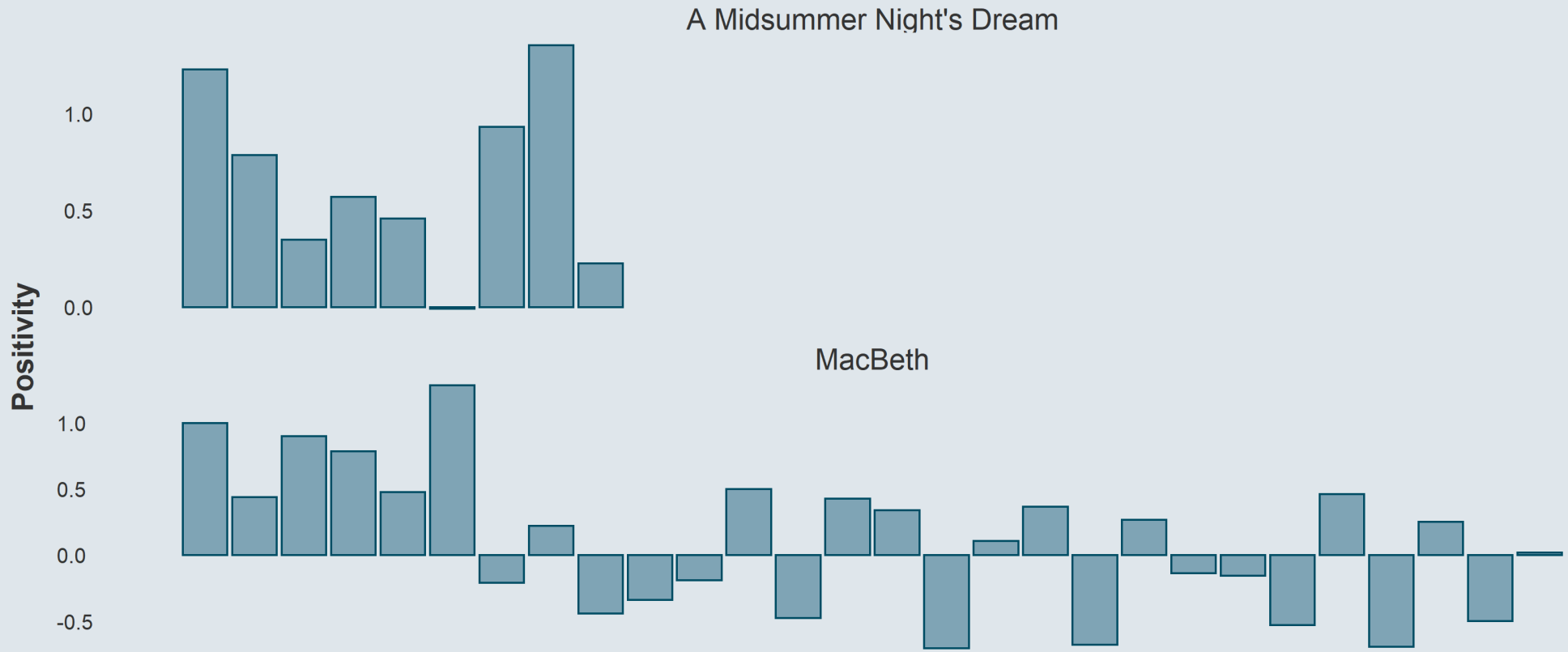
- This function can then be applied to different plays of Shakespeare:

```
plays <- c("a_midsummer_nights_dream.txt", "macbeth.txt",
           "othello_the_moor_of_venice.txt", "romeo_and_juliet.txt",
           "the_merchant_of_venice.txt", "the_taming_of_the_shrew.txt",
           "the_tragedy_of_king_lear.txt", "the_winters_tale.txt")

for (file in plays) {
  sentiment_evolution(file)
}
```

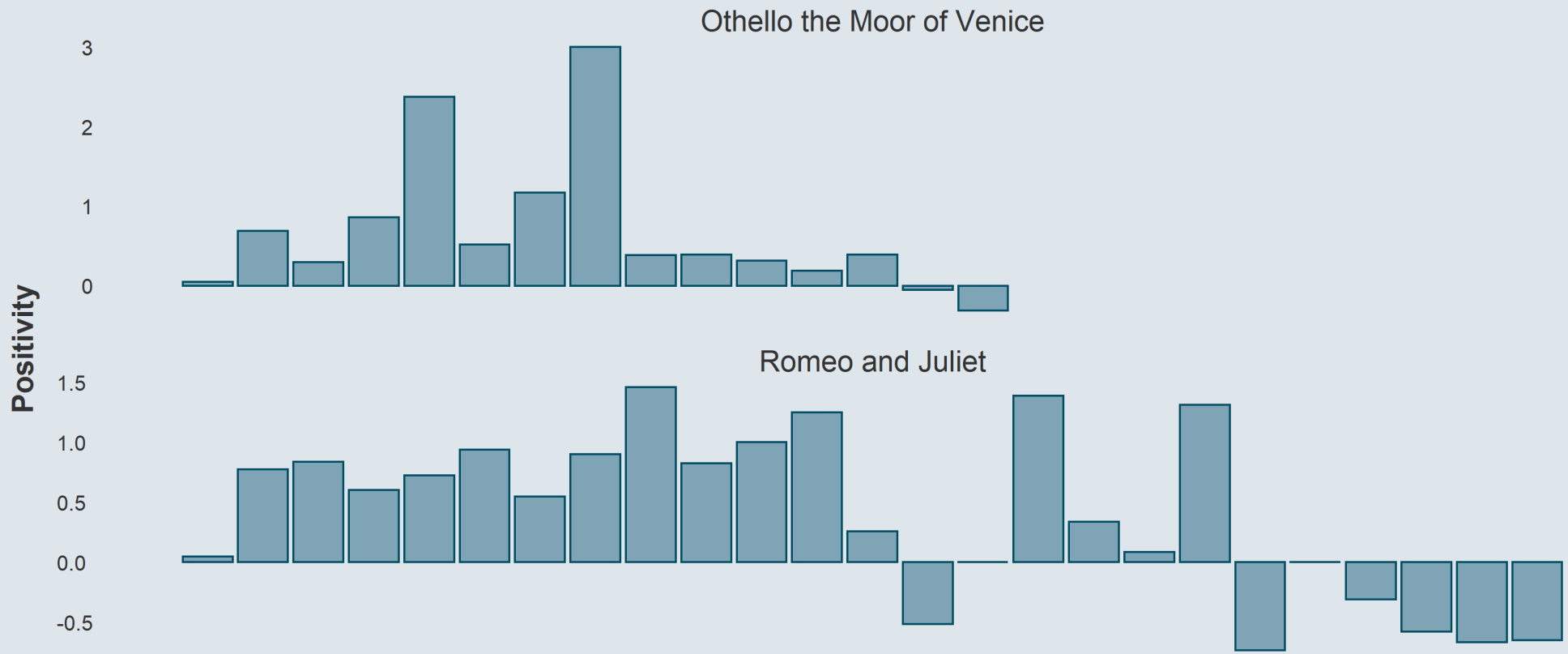
2. Sentiment analysis

2.3. Analysis



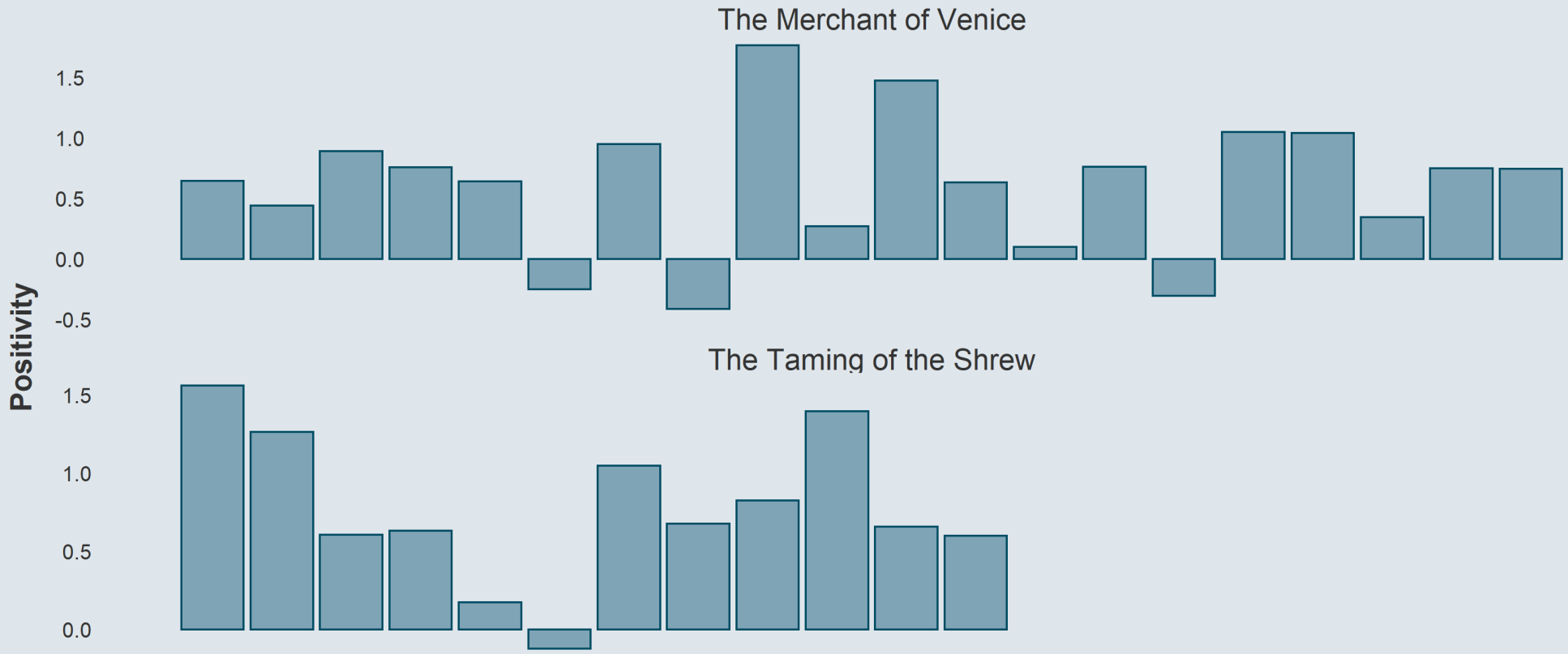
2. Sentiment analysis

2.3. Analysis



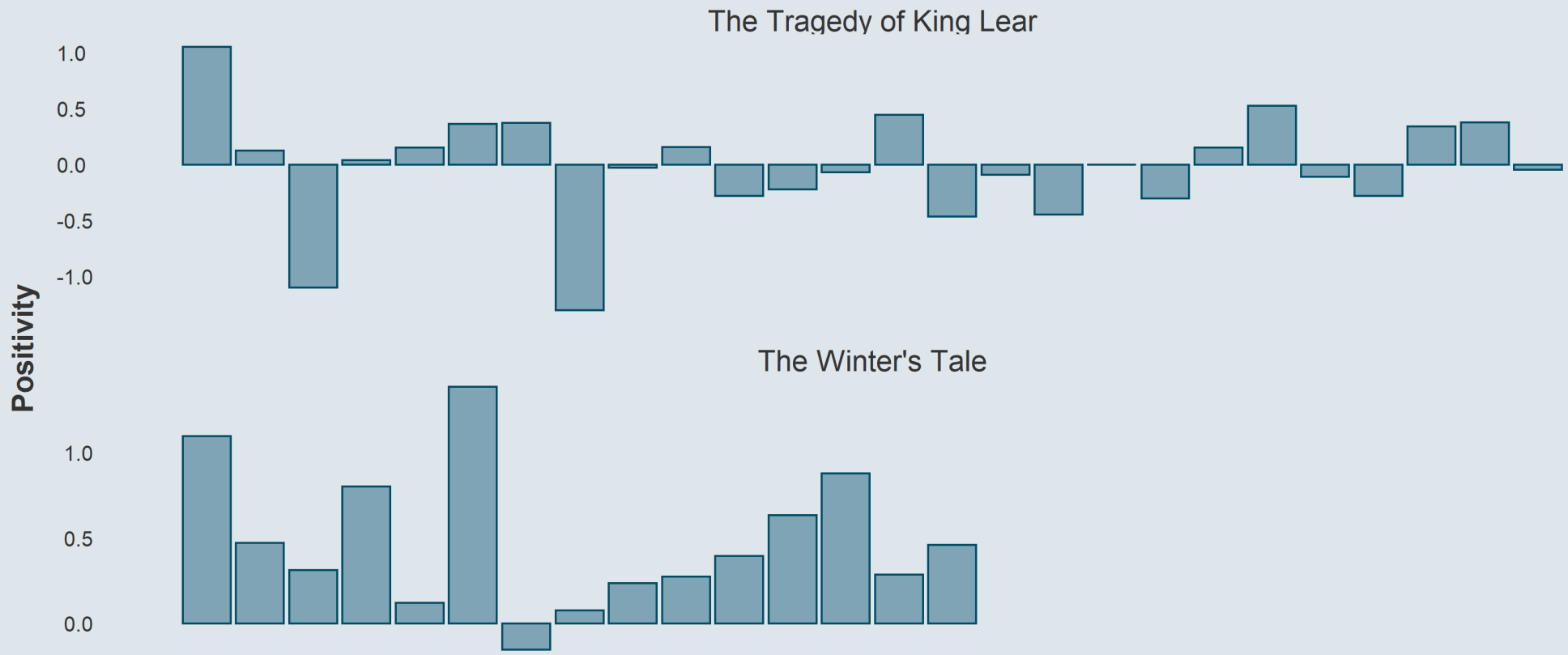
2. Sentiment analysis

2.3. Analysis



2. Sentiment analysis

2.3. Analysis



Overview

1. Cleaning text data ✓

- 1.1. Exploring the data
- 1.2. Regular expressions
- 1.3. Tokenization

2. Sentiment analysis ✓

- 2.1. Stopwords
- 2.2. Sentiments
- 2.3. Analysis

3. Wrap up!

3. Wrap up!

1. Regular expressions

- **Regular expressions are strings of codified characters describing a pattern**
  - For instance the character "^" indicates the start of the string
  - So the regular expression "^a" would match any "a" that is a the beginning of a string

- Regular expressions in R can be used in different functions with different purposes:
  - **grep:** return elements that match the regexp
  - **grepl:** return TRUE for elements that match the regexp and FALSE otherwise
  - **gsub:** replace the elements that match the regexp with what you want

Regexp	Meaning
^	Start of string (or 'not')
$	End of string
.	Any character
*	0 or more occurences
+	1 or more occurences
[^abc]	Not a, b or c
[a-z]	Any lowercase letter from a to z
[A-Z]	Any capital letter from A to Z
[0-9]	Any digit from 0 to 9

3. Wrap up!

2. Tokenization

- **Tokenization** is the fact of cleaning the data so that there is **one unit of text per row**
  - A unit of text (token) can be a character, a letter, a word, a sentence, etc.

line	direction
ACT I	FALSE
SCENE I. A public place.	FALSE
Enter Sampson and Gregory armed with swords and bucklers.	FALSE
SAMPSON.	FALSE
Gregory, on my word, we’ll not carry coals.	FALSE
GREGORY.	FALSE
No, for then we should be colliers.	FALSE

id_act	id_scene	id_line	id_char	line
1	1	1	SAMPSON	Gregory, on my word, we’ll not carry coals.
1	1	2	GREGORY	No, for then we should be colliers.
1	1	3	SAMPSON	I mean, if we be in choler, we’ll draw.

3. Wrap up!

3. Stopwords and sentiments

- First step: **get rid of *stopwords***
  - Stopwords are common words that do not carry much semantic meaning but take space and computing time

```
matrix(get_stopwords()[["word"]][1:24],ncol=3)
```

```
##       [,1]        [,2]         [,3]
## [1,] "i"         "you"        "himself"
## [2,] "me"        "your"       "she"
## [3,] "my"        "yours"      "her"
## [4,] "myself"    "yourself"   "hers"
## [5,] "we"        "yourselves" "herself"
## [6,] "our"       "he"         "it"
## [7,] "ours"      "him"        "its"
## [8,] "ourselves" "his"        "itself"
```

- Second step: **join sentiments dictionary**
  - Some dictionaries are very simple: positive/negative
  - And some are more elaborate: trust/fear/sadness/anger/...

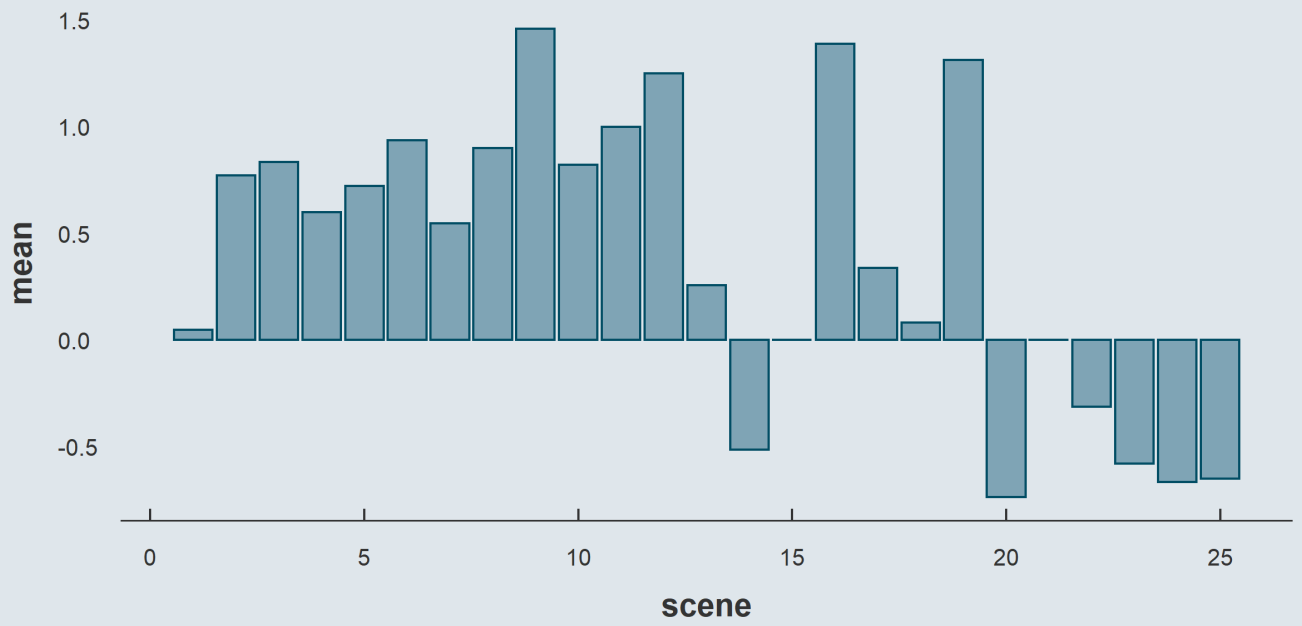
```
head(get_sentiments("bing"), 5)
```

```
## # A tibble: 5 x 2
##   word       sentiment
##   <chr>      <chr>
## 1 2-faces    negative
## 2 abnormal   negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
```

3. Wrap up!

4. Analysis

- Evolution of the average sentiment **over the play**



- Sentiment of characters (rows) **when mentioning other characters** (columns)

```
## # A tibble: 3 x 4
##   id_char  ROMEO  JULIET NURSE
##   <chr>    <dbl>   <dbl> <dbl>
## 1 JULIET  -0.255 -1.5    0.246
## 2 NURSE    0.826 -0.267  0.111
## 3 ROMEO   -0.737 -0.0746 2.4
```