Text manipulation and text preprocessing

Day 11

Solveig Bjørkholt

10. June

Plan for today

What we will learn today:

- What “text as data” means
- How to work with `stringr`
- What *regex* is and how to use it

Text as data

A lot of information is stored in text, be it reports, books, archives, mails, or whatever. This is one of the reasons why analyzing text has become so popular. It opens up a whole new window of opportunities when it comes to managing information and creating insight from it.

When working with text, the stages of the work can loosely be divided into three:

- Text manipulation
- Text preprocessing
- Text tokenization
- Text vectorization
- Text modelling

The first is all about managing and cleaning text to create something that is useful for our purposes. Then, we need to break the text into units and convert it to something the computer can work with - preferably numbers of some sort. At last, there are various modelling techniques we can use to analyze the text. We’ll go through each of these stages in these lectures, showing some basic examples of how to do each step.

Text manipulation

Text is unstructured data – there is no pre-defined model for how the computer should distinguish between variables, observations, and so on. The text is not made for the computer to make sense of it, and this means that we often have to do quite a lot of preparatory work on text to make it ready for computer processing.

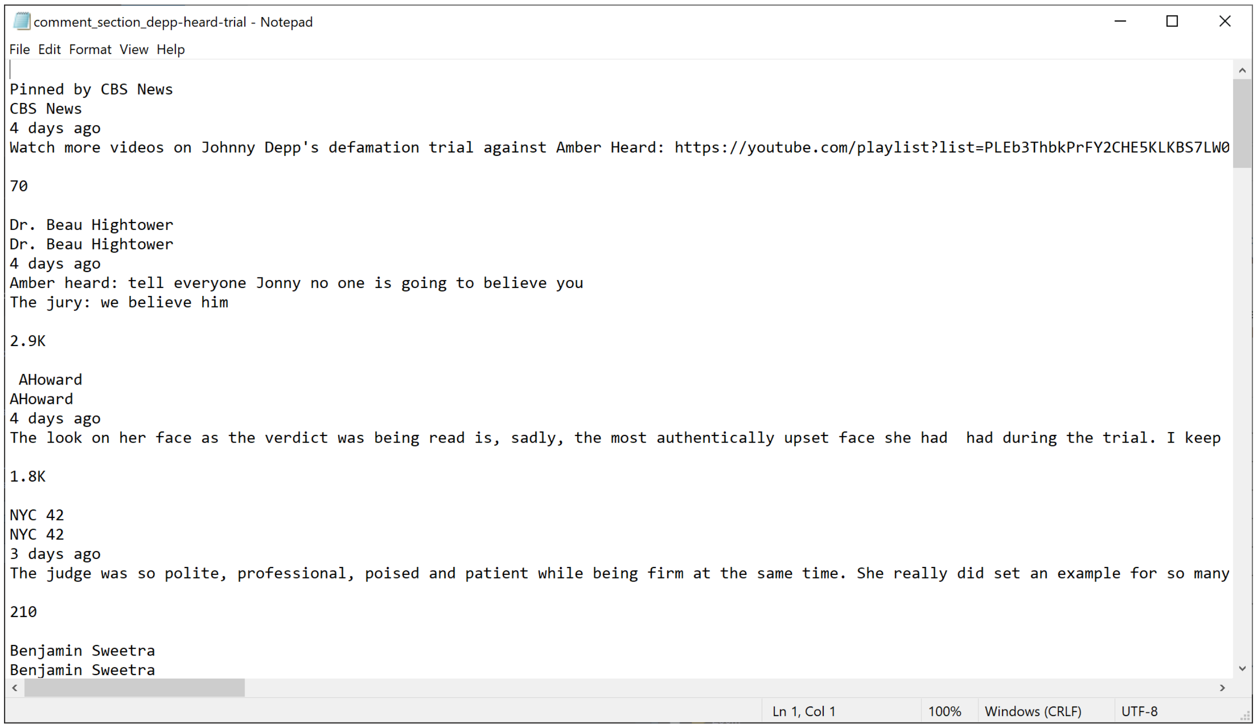
Text in programming language is often referred to as “strings”. A string is basically a sequence of text. They are always wrapped in `""`. For example, the vector below is a string.

```
string <- "A string is a sequence of text."
```

Strings can be long (pages of documents) or short (a sentence or part of a sentence), and they can contain capital letters, small letters, punctuation, symbols, numbers, emojis, foreign letters, and so on. We need to manage this string variation, or else it will create noise when we analyze the text. There are many ways of doing this in R, we'll use the `stringr` package along with some *regex*. *Regex* is not an R package. It stands for "regular expression", and it is a method to specify patterns in text according to the patterns they display.

String manipulation using *regex* and stringr

To make everything a bit clearer, let's start with some really dirty text – the comment section on YouTube. Here, I have simply copied some of the comments from the video when the judge reads the verdict on the dispute between Johnny Depp and Amber Heard. If you're curious, you can see the video here. I pasted the text into a txt-file and saved it on my computer.


comment_section_depp-heard-trial - Notepad
File Edit Format View Help
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

```
Pinned by CBS News
CBS News
4 days ago
Watch more videos on Johnny Depp's defamation trial against Amber Heard: https://youtube.com/playlist?list=PLEb3ThbkPrFY2CHE5KLKBS7LW0

70

Dr. Beau Hightower
Dr. Beau Hightower
4 days ago
Amber heard: tell everyone Jonny no one is going to believe you
The jury: we believe him

2.9K

 AHoward
AHoward
4 days ago
The look on her face as the verdict was being read is, sadly, the most authentically upset face she had  had during the trial. I keep

1.8K

NYC 42
NYC 42
3 days ago
The judge was so polite, professional, poised and patient while being firm at the same time. She really did set an example for so many

210

Benjamin Sweetra
Benjamin Sweetra
```

Now, to read this text into R, we can use the package `readtext`. This package also works on many other document types such as .doc and .pdf, and we can use it on complete folders containing text. This means that if you want to read in many documents at once, just place the documents in one folder and specify this folder in the "folder" argument:

`readtext("/path/folder/")`

The function `readtext` gives you a dataframe with two variables called `doc_id` and `text`. The `doc_id` variable has the name of the document, and the `text` variable contains the contents of the document.

```
library(readtext)

comments_trial <- readtext("../../datafolder/comment_section_depp-heard-trial.txt")

glimpse(comments_trial)
```

```
## Rows: 1
## Columns: 2
## $ doc_id <chr> "comment_section_depp-heard-trial.txt"
## $ text   <chr> "\nPinned by CBS News\nCBS News\n4 days ago\nWatch more videos ~
```

We fetch the information from the `text` variable using `select` and `pull`. Then, using the function `substr`, I show the 100 first characters in the string. The text is pretty noisy. We have:

- Big letters and small letters
- URLs
- Line shifts (the symbol for line shift is `\n` and `\n\n`)
- Numbers
- Foreign signs such as â and €™.

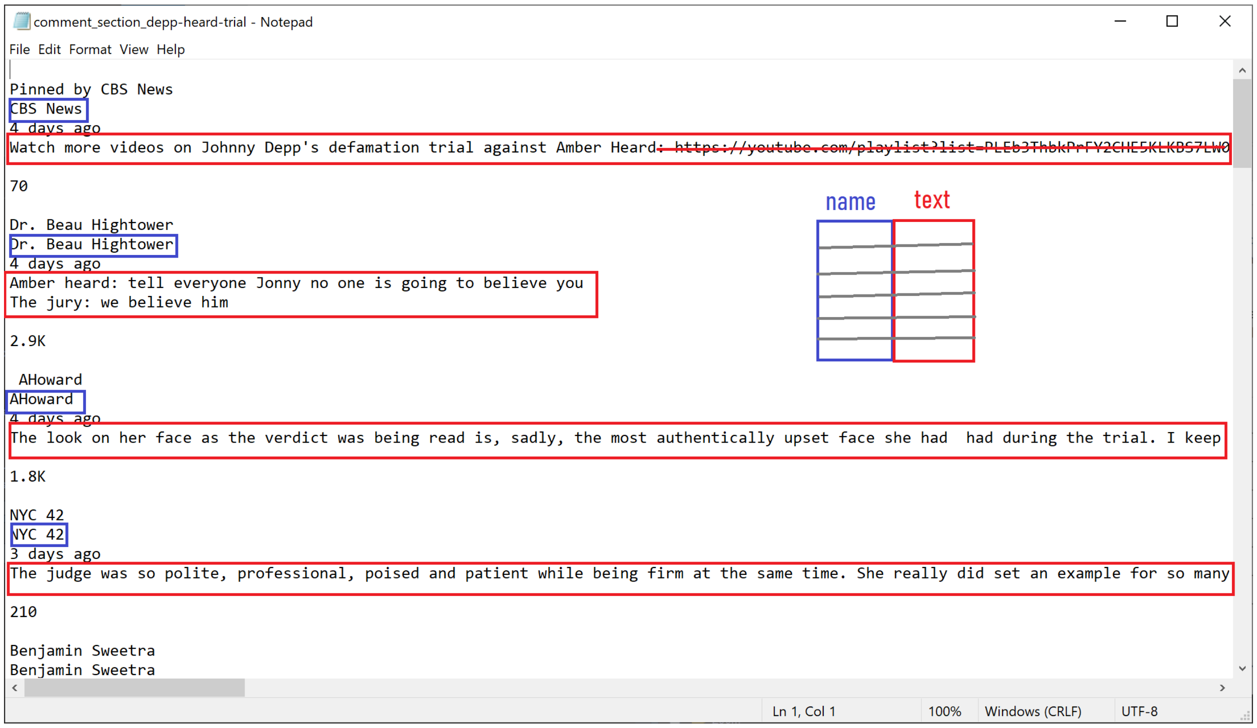
```r
comments_trial_text <- comments_trial %>%
  select(text) %>% # Getting only the text variable
  pull() # Making the variable into a single character vector

comments_substring <- substr(comments_trial_text, 1, 100)

comments_substring # Showing character 1 to 100 of the text.
```

```
## [1] "\nPinned by CBS News\nCBS News\n4 days ago\nWatch more videos on Johnny Depp’s defamation trial
```

Often, what we want is to clear out the most important parts of the information and move it into a dataframe. We might for example want to fetch the names of who posted a comment and the text from the comment, arranging them in two different columns in a dataframe, as shown in the picture below[1]. Also, typically we want to remove the parts of the text that is not important for us, for example URLs.


comment_section_depp-heard-trial - Notepad
File Edit Format View Help
name
text
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

```

Pinned by CBS News
CBS News
4 days ago
Watch more videos on Johnny Depp's defamation trial against Amber Heard~~: https://youtube.com/playlist?list=PLEb3ThbkPrFY2CHE5KLKBS7LW0~~

70

Dr. Beau Hightower
Dr. Beau Hightower
4 days ago
Amber heard: tell everyone Jonny no one is going to believe you
The jury: we believe him

2.9K

 AHoward
AHoward
4 days ago
The look on her face as the verdict was being read is, sadly, the most authentically upset face she had  had during the trial. I keep

1.8K

NYC 42
NYC 42
3 days ago
The judge was so polite, professional, poised and patient while being firm at the same time. She really did set an example for so many

210

Benjamin Sweetra
Benjamin Sweetra
```

[1]It could also have been useful to extract the time when the comment was made, but it all depends on what kind of analysis you want to do

stringr

stringr is a package within the tidyverse that offers a handful of functions for text manipulation.

```
substr(comments_substring, 1, 5) # Fetch parts of a string, here characters from position 1 to position
```

```
## [1] "\nPinn"
```

```
str_length(comments_substring) # Check how many characters a present in a string
```

```
## [1] 100
```

```
str_c(comments_substring, ". The end.") # Put together two strings. (Equivalent to paste and paste0)
```

```
## [1] "\nPinned by CBS News\nCBS News\n4 days ago\nWatch more videos on Johnny Depp's defamation trial
```

```
str_to_upper(comments_substring) # Make all characters into upper case
```

```
## [1] "\nPINNED BY CBS NEWS\nCBS NEWS\n4 DAYS AGO\nWATCH MORE VIDEOS ON JOHNNY DEPP'S DEFAMATION TRIAL
```

```
str_to_lower(comments_substring) # Make all characters into lower case
```

```
## [1] "\npinned by cbs news\ncbs news\n4 days ago\nwatch more videos on johnny depp's defamation trial
```

```
str_to_title(comments_substring) # Give all words a start capital letter and make the rest lower case
```

```
## [1] "\nPinned By Cbs News\nCbs News\n4 Days Ago\nWatch More Videos On Johnny Depp's Defamation Trial
```

```
str_to_sentence(comments_substring) # Give all sentences a start capital letter and make the rest lower
```

```
## [1] "\nPinned by cbs news\nCbs news\n4 days ago\nWatch more videos on johnny depp's defamation trial
```

```
str_detect(comments_substring, "Johnny Depp") # Check if a pattern is part of a string
```

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

```
str_subset(comments_substring, "Johnny Depp") # Subset the strings that contain a specific pattern
```

```
## [1] "\nPinned by CBS News\nCBS News\n4 days ago\nWatch more videos on Johnny Depp's defamation trial
```

```
str_locate(comments_substring, "Johnny Depp") # Check where in the string a specific pattern is located
```

```
##      start end
## [1,]    62  72
```

```
str_extract(comments_substring, "Johnny Depp") # Extract only the part of a string that matches a speci
```

```
## [1] "Johnny Depp"
```

```
str_remove(comments_substring, "Johnny Depp") # Remove from the string all instances that match a speci
```

```
## [1] "\nPinned by CBS News\nCBS News\n4 days ago\nWatch more videos on 's defamation trial against "
```

```
str_replace(comments_substring, "Johnny Depp", "Jack Sparrow") # Replace patterns of a string with anot
```

```
## [1] "\nPinned by CBS News\nCBS News\n4 days ago\nWatch more videos on Jack Sparrow's defamation trial
```

```
str_trim(comments_substring) # Remove whitespace from the start and the end of a string
```

```
## [1] "Pinned by CBS News\nCBS News\n4 days ago\nWatch more videos on Johnny Depp's defamation trial ag
```

```
str_squish(comments_substring) # Remove whitespace from the start and end of a string, and inside a str
```

```
## [1] "Pinned by CBS News CBS News 4 days ago Watch more videos on Johnny Depp's defamation trial again
```

```
str_split(comments_substring, "\n") # Make a list of strings by splitting them on a certain pattern
```

```
## [[1]]
## [1] ""
## [2] "Pinned by CBS News"
## [3] "CBS News"
## [4] "4 days ago"
## [5] "Watch more videos on Johnny Depp's defamation trial against "
```

```
str_conv(comments_substring, "UTF-8") # Convert the encoding of a string - useful if you get a lot of w
```

```
## [1] "\nPinned by CBS News\nCBS News\n4 days ago\nWatch more videos on Johnny Depp's defamation trial
```

Some of these functions also allow us to add an `_all`, which would match all instances of a pattern.

```
str_locate_all()
str_extract_all()
str_remove_all()
str_replace_all()
```

These functions are excellent when working with strings. Yet, notice that they often require a *pattern*. This pattern can be a string that matches the text perfectly, for example the name "Johnny Depp". However, if we want to do large-scale cleaning of a text, we should work to find more general patterns, and use these to decide which parts of the string that should be kept, removed, modified, and so on. To specify patterns, we use *regex*.

Regex

Regex stands for "Regular expressions", and it is a method to match patterns in text. This allows us to handle many instances of the same thing in large volumes of text.

To understand what we mean by pattern, consider an email address. Some collections of email addresses can be quite standardized, for example emails within a company. Imagine a company by the name ThreePods with the abbreviation *TP* and the domain *.com*, giving all their employees email addresses with their surname. This collection of email addresses might look something like this:

`smith@tp.com`

`johnson@tp.com`

`brown@tp.com`

`perez@tp.com`

`young@tp.com`

To match these strings, we can use *regex* and specify that we want to match all strings that begin with a sequence of characters, followed by an @, then `tp` and `.com`. The *regex* string would look like this:

`"[a-z]+@tp.com"`

So what would this look like with the `stringr` functions above? Well, imagine that we have a bunch of text and want to only extract the email addresses. Then we could use `str_extract_all` (to get all five email addresses, not just one) together with our *regex* pattern and do this:

```
text <- "Here in the company ThreePods, we deliver many services of great quality that make important c

str_extract_all(text, "[a-z]+@tp.com")
```

```
## [[1]]
## [1] "smith@tp.com"   "johnson@tp.com" "brown@tp.com"   "perez@tp.com"
## [5] "young@tp.com"
```

The *regex* pattern in the string above is the `[a-z]+` part. `[a-z]` means "match any lower case character, and `+` means that the pattern can include both one or several of these. In other words; match any single or group of lower case characters. The table below shows an overview of some of the *regex* patterns, along with some examples of how they can be used in practice.

In particular, notice the `\`. Whenever you in *regex* want to specify for example that the pattern should contain a question mark, use two times backslash first; `\\?`.

Char	Description	Meaning
\	Backslash	Used to escape a special character
ˆ	Caret	Beginning of a string
$	Dollar sign	End of a string
.	Period or dot	Matches any single character
|	Vertical bar or pipe symbol	Matches previous OR next character/group
?	Question mark	Match zero or one of the previous
*	Asterisk or star	Match zero, one or more of the previous
+	Plus sign	Match one or more of the previous
( )	Opening and closing parenthesis	Group characters
[ ]	Opening and closing square bracket	Matches a range of characters

Char	Description	Meaning
{ }	Opening and closing curly brace	Matches a specified number of occurrences of the previous

Examples

`Finished\? matches "Finished?"`

`^http matches strings that begin with http`

`[^0-9] matches any character not 0-9`

`ing$ matches "exciting" but not "ingenious"`

`gr.y matches "gray", "grey"`

`Red|Yellow matches "Red" or "Yellow"`

`colou?r matches colour and color`

`Ah? matches "Al" or "Ah"`

`Ah* matches "Ahhhhh" or "A"`

`Ah+ matches "Ah" or "Ahhh" but not "A"`

`[cbf]ar matches "car", "bar", or "far"`

`[a-zA-Z] matches ascii letters a-z (uppercase and lower case)`

Example: string and *regex* for comments section cleaning

Lets look at how `stringr` and *regex* can be used for cleaning of our comments section string. As mentioned above, a typical problem is that we start with a chunk of text, but what we really want is a dataframe with tidy variables. The job of cleaning text can be very time consuming and nitpicky, and I am not going to delve into the details here. However, here is an example of a code to go from a chunk of text to a dataframe using `stringr` and *regex*. The dataframe has two variables - one with who has posted a comment (`name`) and one with the content of their post (`text`).

```
comments <- comments_trial_text %>%
  str_split("\n\n") %>% # Split the text on double lineshift, \n\n
  .[[1]] %>% # Pick the first element of the list, as we only have one
  str_remove_all("[0-9]+(\\.)?(K)?") %>% # Remove all parts of the string containing numbers, possiblu
  str_remove_all("days ago|hours ago|\\(edited\\)|Pinned by") %>% # Removing these strings from the tex
  stringi::stri_remove_empty() %>% # Removing all empty character vectors
  str_split("\n") %>% # Splitting again on single lineshift
  map(., ~ str_squish(.)) %>% # Removing whitespace from between and within the string, using map becau
  map(., ~ stringi::stri_remove_empty(.)) %>%  # Removing empty character vectors again
  map(., ~ unique(.)) # Picking only the unique values

name <- list() # Make a list where we can store the names
text <- list() # Make a list where we can store the text

for (i in 1:length(comments)) { # Creating a for-loop
  name[[i]] <- comments[[i]][1] # Picking the first vector element in every list element and store it i
  text[[i]] <- str_c(comments[[i]][-1], collapse = "") # Picking the rest of the elements from the list
}
```

```
dataframe <- as_tibble(list(name = unlist(name), # Putting the two vectors together into a dataframe
                            text = unlist(text)))


dataframe <- dataframe %>%
  mutate(text = str_remove_all(text, "https://.*"), # In the dataframe, I change the text variable and
         text = str_conv(text, "ASCII"), # The text has emojis, to remove them I convert the text to AS
         text = str_replace_all(text, "\uFFFD", "")) # Now all emojis are called UFFFD, and I remove th

dataframe <- dataframe %>%
  mutate(name = ifelse(str_length(name) >= 50, NA, name)) %>% # One text ended up in the name columns,
  na.omit() # And remove NA from the dataframe

dataframe
```

```
## # A tibble: 27 x 2
##    name                text
##    <chr>               <chr>
##  1 CBS News            "Watch more videos on Johnny Depp's defamation trial agai~
##  2 Dr. Beau Hightower "Amber heard: tell everyone Jonny no one is going to beli~
##  3 AHoward             "The look on her face as the verdict was being read is, s~
##  4 NYC                 "The judge was so polite, professional, poised and patien~
##  5 Benjamin Sweetra    "This is a big win because it shows we cant automatically~
##  6 Mary Gaia           "Proud of our jury system. They did the job that was entr~
##  7 Jacy LB             "Shes an embarrassment to DV abuse victims. Her statement~
##  8 Artsy               "I honestly didnt think Depp would win because defamation~
##  9 Welly Sanusi        "So happy for Johnny Depp. Through the trial, everyone ca~
## 10 Kimber Doe          "Imagine being someone in this situation who was actually~
## # ... with 17 more rows
```

Text preprocessing

Text manipulation and text preprocessing is closely related. However, while text manipulation considers the general act of juggling text and forming it into lists, dataframes and so on, text preprocessing is all about cleaning the text to make it ready for analysis.

Typical preprocessing units

Which parts of a text you decide to keep, modify or remove, depends on the type of analysis you're doing. For example, sometimes keeping names of people might be essential, then you might want to keep capital letters. If you're working with laws, you might want to remove most symbols except §. However, that being said, there are a few things that analysists usually consider removing from the text to reduce noise.

Whitespace

Whitespace occurs in strings with more than one space between units. To remove, use `str_trim` or `str_squish`.

```
string <- "  This   is a    string with  whitespace    .  "

string
```

```
## [1] "  This   is a    string with  whitespace    .  "
```

```
str_trim(string, side = "both") # Remove whitespace from ends of strings
```

```
## [1] "This   is a    string with  whitespace    ."
```

```
str_squish(string) # Remove whitespace from ends of strings and in the middle
```

```
## [1] "This is a string with whitespace ."
```

Stopwords

Stopwords include all types of words that do not carry any meaning in themselves, for example *for*, *a*, *the* and *but*. Words such as prepositions and conjunctions are typically regarded as stopwords.

In the code below, first make a character vector called "stopwords" with several stopword specified. Then, I use `str_c` to paste `\\b` and `\\b` before and after each word. This is to create a so-called "word boundary", so that the stopwords will only match for example "i", and not also the i in containing. The `collpase = "|"` argument divides each word by |, which means "or". Then, I use `str_replace_all` to replace all stopwords with nothing, `""`.

```
stopwords <- c("i", "is", "are", "we", "him", "her", "such", "they", "and", "a", "for", "be", "as", "wi

stopwords_boundary <- str_c("\\b", stopwords, "\\b", # Creating word boundaries by adding \\b in front 
                            collapse = "|") # Dividing the words by |

string <- "this is a string containing a few stopwords such as a, for and i, and to our common benefit,

str_replace_all(string, stopwords_boundary, "")
```

```
## [1] "   string containing  few stopwords   ,   ,    common benefit,    removed"
```

Punctuation

It's not unusual to want to remove punctuation such as dots, commas and question marks from the text. One way of doing this, is to make a vector specifying the punctuation marks you want to remove, divided by a |. Recall that in *regex*, letters such as `?` and `.` are used to indicate sentence patterns. To make it clear that we are talking about the *letter* and not the pattern indicator, add two backslashes first.

```
string <- "Here we have a string. Is it useful for you? Maybe not... But! With time, it might be."

punctuation <- c("\\!|\\?|\\.|\\,")

str_replace_all(string, punctuation, "")
```

```
## [1] "Here we have a string Is it useful for you Maybe not But With time it might be"
```

If you are not too picky about which punctuation you remove, you can also use `[:punct:]`.

```r
str_replace_all(string, "[:punct:]", "")
```

```
## [1] "Here we have a string Is it useful for you Maybe not But With time it might be"
```

Numbers

Numbers are seldom easy to analyze in a text-based setting, because they do not in themselves carry any meaning without knowing what they refer to. And if we want to analyze numbers, we might as well create a standard dataframe with variables and numbers and analyze it the old-fashioned way. Because numbers are not easy to use in text analysis, we often remove them.

Here, I use `str_remove_all` together with the *regex* pattern `[0-9]+`. `[0-9]` means any number and `+` means that there can be one or more of them.

```r
string <- "I have 50 likes on my post. My friends have 100 likes. It's not fair. I want 1000000 likes!"

str_remove_all(string, "[0-9]+")
```

```
## [1] "I have  likes on my post. My friends have  likes. It’s not fair. I want  likes!"
```

If I want to extract only numbers of a specific length, I could use curly parentheses wrapping how many units I want to extract. A common thing is for example to want to extract years from a string. In the example below, we extract only numbers of four, leaving the 1st aside.

```r
string <- "I was born on the 1st of October 1993."

str_extract(string, "[0-9]{4}")
```

```
## [1] "1993"
```

Symbols

Symbols can also be useful to remove, whether they are /, {}, (), §, +, - or any other. One method is, as shown above, to make a vector with the symbols, divide them by the | operator and make sure you allude to the symbol and not the *regex* pattern by adding two backslash first.

```r
symbols <- str_c("\\!|\\@|\\#|\\$|\\%|\\^|\\&|\\*|\\(|\\)|\\{|\\}|\\-|\\=|\\_|\\+|\\:|\\|\\<|\\>|\\?|\\

string <- "A string must be written in [code] #code-string. Because string - code = just normal text &

str_remove_all(string, symbols)
```

```
## [1] "A string must be written in code codestring Because string  code  just normal text  that we can
```

Another possibility is to replace the non-alphabetic characters with space using `str_replace_all` and `[[:alnum:]]`. The hat, ^, means “beginning of”.

```r
string <- "A string must be written in [code] #code-string. Because string - code = just normal text &

str_replace_all(string, "[^[:alnum:]]", " ")
```

```
## [1] "A string must be written in  code   code string  Because string   code   just normal text   tha
```