Big Data and Data Mining (including Text Mining) | Data analysis practice

Table of contents

This document is a brief companion to the practical part of the course *Big Data and Data Mining (including Text Mining).*

It is mainly based on the following resources that you can check for further references:

- Wickham, H., & Grolemund, G. (2016). R for Data Science. O'Reilly Media. http://r4ds.had.co.nz/
- Silge, J., & Robinson, D. (2017). Text Mining with R. A Tidy Approach. O'Reilly. https://www.tidytextmining.com/

We'll use the **R** statistical/programming language, **RStudio** as development environment (you can also use Visual Studio Code), and some **packages** that will extend some handful functions for us.

To download and install R and RStudio, follow these instructions: https://r4ds.had.co.nz/introduction.html#prerequisites

(1) Data analysis with R

The goal of this lesson is to introduce the **data analysis workflow**. In particular, we will learn to:

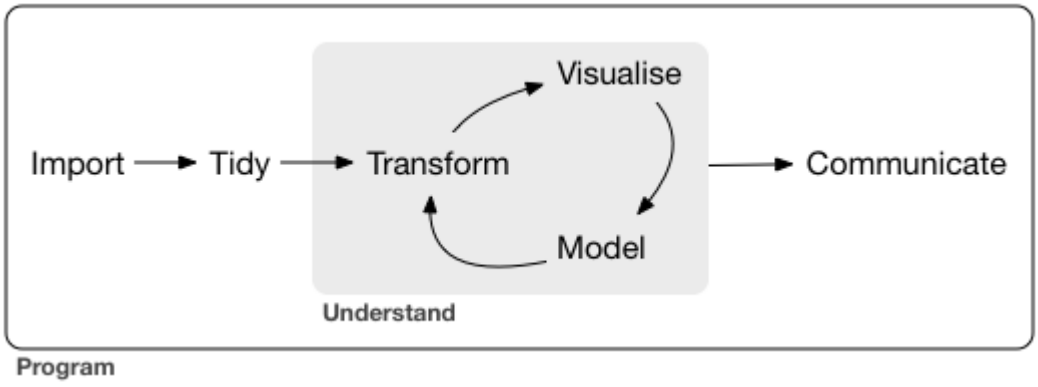
1. load data and resources;
2. explore and become familiar with the data and its structure;
3. use the *tidy* structure for our tables;
4. transform the data to clean it and have the optimal structure for our analyses.

You can check more on these topics here:

https://r4ds.had.co.nz/workflow-basics.html

https://r4ds.had.co.nz/transform.html

The general workflow of the data analysis is as follows:



(diagram taken from https://r4ds.had.co.nz/introduction.html)

We will work with a dataset of free associations to the word "Big data". This dataset was built from a brief survey with the following instructions:

> *Please tell us what words or phrases come to mind when you think of "big data". We also ask you to please tell us if these ideas that you have just introduced correspond to something that you value positively (something you like) or negatively (something you dislike)*

In addition, we have recorded the order in which each word was entered by the participant (a value usually between 1 to 5, although participants could enter more words).

A (simpler) form of this survey can be found here: https://forms.gle/ysyRxvsDHA9r47NH8

With this dataset, and through the indicated tasks, we are going to try to answer the following question: **what is the common sense about big data?**

The way in which we will analyze the data loosely follows the steps of the "prototypical analysis" from the Social representations theory by S. Moscovici and J. C. Abric.

Load data and resources

The first thing we will do is load some **packages** that will provide us with a set of functions that we will use throughout the exercise. Remember that a function is a sequence of commands that are applied to an object that is passed to the function, referencing it between its parentheses. For example, we will use the `library()` function and library names to enable the `readr` functions to import data, and the `tidyverse` set of libraries to manipulate and display.

```
install.packages("readr")
install.packages("tidyverse")
```

```
library(readr)
library(tidyverse)
```

We will then import the data with the `read_csv()` function from the `readr` package. In RStudio you can list a package's functions by typing their name followed by `::`.

We are interested in keeping this data as an object in memory, since we will be working with it in what follows. To do this, we use an assignment operator `<-` preceded by the name that we will give to the object.

```
word_associations <- readr::read_csv(file = "https://raw.githubusercontent.com/gastonbecer
```

```
Rows: 1704 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (2): id, word
dbl (2): order_of_evocation, evaluation

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Explore the data

The goal of **data exploration** is to familiarize ourselves with the structure of the data, and transform it in a way that allow us to carry out our analyses. Usually, this is where the cleaning steps and pre-processing are decided for further analysis, like modeling.

The first thing we are going to do check the size and structure of our dataset with `glimpse()`, and the first records with `head()`, and the basic stats with `summary()`.

This will allow us to know:

- the number of records and columns;
- the names of the columns and their data type;
- the content of the first records;
- the range of the numeric fields

```r
glimpse(word_associations)
```

```
Rows: 1,704
Columns: 4
$ id                <chr> "-M0-9OQkabuGoSmceB5E", "-M0-9OQkabuGoSmceB5E", "-M~
$ word              <chr> "information", "analysis", "research", "commercial"~
$ order_of_evocation <dbl> 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, ~
$ evaluation        <dbl> 5, 5, 5, 4, -5, 5, 5, 5, 0, -3, 2, 4, 4, 3, 2, 5, 4~
```

```r
head(word_associations, n = 10)
```

```
# A tibble: 10 x 4
   id                     word        order_of_evocation evaluation
   <chr>                  <chr>                    <dbl>      <dbl>
 1 -M0-9OQkabuGoSmceB5E information                    1          5
 2 -M0-9OQkabuGoSmceB5E analysis                       2          5
 3 -M0-9OQkabuGoSmceB5E research                       3          5
 4 -M0-9OQkabuGoSmceB5E commercial                     4          4
 5 -M0-9OQkabuGoSmceB5E filtration                     5         -5
 6 -M0U7_pJAU9EhgaOLIWq information                    1          5
 7 -M0U7_pJAU9EhgaOLIWq technology                     2          5
 8 -M0U7_pJAU9EhgaOLIWq systems                        3          5
 9 -M0U7_pJAU9EhgaOLIWq computer                       4          0
10 -M0U7_pJAU9EhgaOLIWq freaks                         5         -3
```

```
summary(word_associations)
```

```
      id                word           order_of_evocation   evaluation
 Length:1704        Length:1704        Min.   : 1.000     Min.   :-5.000
 Class :character   Class :character   1st Qu.: 2.000     1st Qu.:-1.000
 Mode  :character   Mode  :character   Median : 3.000     Median : 1.000
                                       Mean   : 3.058     Mean   : 1.055
                                       3rd Qu.: 4.000     3rd Qu.: 5.000
                                       Max.   :14.000     Max.   : 5.000
```

You can also explore the entire dataset using `view()`. Although not needed for this exercise, in larger datasets you could use some packagess to help you on the exploratory part. I recommend to take a look at `skimr`.

Now, let's check our dataset structure with a `glimpse`. Our table has the words that the participants responded to in the `word` column, the order in which those words were entered by the participant in `order_of_evocation`, and a valuation of that idea in `evaluation`. The `id` column allow us to bind the records together vertically because our dataset has several records (usually 5) per participant: we have 1 record per word associated to big data. This is called a **longer** (rather than **wider**) dataset, and the action to transform this structure is called **pivoting**. More on this here: https://r4ds.had.co.nz/tidy-data.html#pivoting

Transform, visualize and clean

Now let's see how we can transform the data to get answers to the following questions:

1. What are the most frequent words?

2. What are the words with the most extreme evaluations?
3. What are the words that were evoked faster?

To answer this we are going to use **manipulation verbs** (from `dplyr`, a library included in `tidyverse`) on our table. Some of these verbs are:

- `filter()` to keep only some records by setting a condition;
- `mutate()` to add a column/variable with the result of some operation on other columns;
- `group_by()` and `summarise()` to perform some operation on the data of different records, *reducing* them by only one by groups;
- `count()` which returns the number of records in a group; `n()` does the same when within a group operation;
- `arrange()` sorts the data ascending or descending;

Then, to chain these actions we are going to use an operator called **pipe `%>%`** that takes the object on the left and makes it go through the function on its right, returning the result. This makes it easier to think and write code to manipulate an object, since it resembles how we naturally handle objects (we first think of the object, then what we want to do with it).

Now, we have all the elements to answer our questions. It only remains to design a path of operations to make the response visible:

- (Step1) We are going to take our table and…
- (Step2) …group records that share the associated word (`word`), and for each one we will:
- (Step 3) count the number of records (giving us the frequency);
- (Step4) and calculate the mean of their valuations;
- (Step5) as well as the mean of the order in which it was evoked;
- (Step0) … the result of this operation we are going to store in a new table, which we will then operate to answer our questions.

For these operations we are going to use the commands just seen:

```
word_associations_frequency <- word_associations %>% # step 0 and 1
  group_by(word) %>% # (step 2)
  summarize(
    freq = n(), # (step 3)
    mean_score = mean(evaluation), # (step 4)
    mean_order_of_evocation = mean(order_of_evocation) #(step 5)
  )
```

```r
glimpse(word_associations_frequency)
```

```
Rows: 817
Columns: 4
$ word                   <chr> ".", "1984", "4 v", "4th generation human righ~
$ freq                   <int> 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 9, 1, 1, 2, 1~
$ mean_score             <dbl> 0.0000000, -5.0000000, 5.0000000, 5.0000000, 5~
$ mean_order_of_evocation <dbl> 4.000000, 2.000000, 3.000000, 6.000000, 1.0000~
```

If we just order/arrange this table we are already in a position to indicate which are the most/least frequent words.

For this we are going to use `slice_max()`, which sorts the data and slices it at some position.

```r
word_associations_frequency %>%
  slice_max(order_by = freq, n = 10)
```

```
# A tibble: 10 x 4
   word         freq mean_score mean_order_of_evocation
   <chr>       <int>      <dbl>                   <dbl>
 1 information   130       2.52                    2.18
 2 data           83       2.13                    1.94
 3 technology     44       2.34                    2.98
 4 control        35      -2.31                    2.57
 5 internet       34       2.94                    2.91
 6 analysis       25       3.08                    2.92
 7 great          25       2.64                    1.8
 8 handling       21      -3.52                    2.71
 9 privacy        19      -2.95                    3.53
10 statistics     19       2.37                    2.95
```

The most frequently mentioned word was “information”, along with a set of other words that we can say refer to the handling of data mediated by technology, with various products, such as the analysis of information and the generation of knowledge, or the manipulation and control (the only words that have a negative evaluation).

To know the most/least valued words, we must generate other cuts:

```r
word_associations_frequency %>%
  slice_max(order_by = mean_score, n=5) # most likeable words
```

```
# A tibble: 162 x 4
   word                                          freq mean_score mean_order_of_e~1
   <chr>                                        <int>      <dbl>              <dbl>
 1 4 v                                              1          5                  3
 2 4th generation human rights                      1          5                  6
 3 a contained sphere of data and information       1          5                  1
 4 A great fact                                     1          5                  1
 5 A type of problematic                            1          5                  5
 6 ability to analyze and relate D                  1          5                  5
 7 abundant                                         1          5                  2
 8 actuate                                          1          5                  3
 9 advantage                                        1          5                  4
10 all                                              1          5                  5
# ... with 152 more rows, and abbreviated variable name
#   1: mean_order_of_evocation
```

```
word_associations_frequency %>%
  slice_min(order_by = mean_score, n=5) # least likeable words
```

```
# A tibble: 106 x 4
   word                 freq mean_score mean_order_of_evocation
   <chr>               <int>      <dbl>                   <dbl>
 1 1984                    1         -5                       2
 2 a great change          1         -5                       5
 3 a hamburger             1         -5                       2
 4 absence of intimacy     1         -5                       4
 5 Access inequality       1         -5                       2
 6 action                  1         -5                       4
 7 almism                  1         -5                       4
 8 amazing                 1         -5                       3
 9 an important date       1         -5                       4
10 attract atention        1         -5                       4
# ... with 96 more rows
```

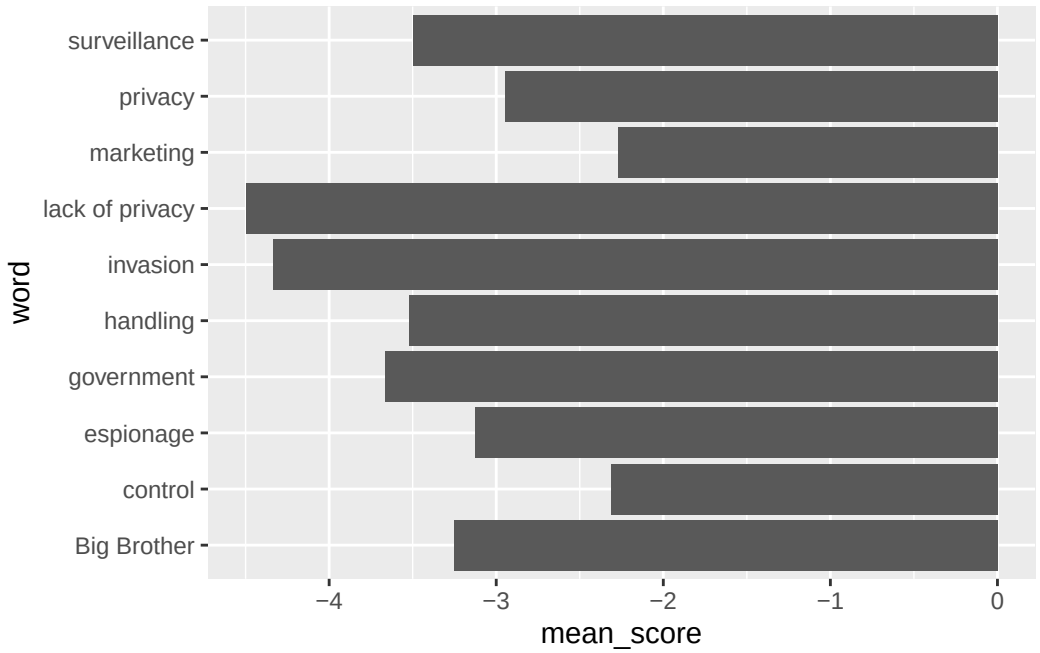
Beyond the fact that certain themes could be inferred in the words (especially the negative ones), we must ask ourselves if it makes sense to work with idiosyncratic ideas and expressions, introduced by a single participant. Ultimately, the question that guides all our exploration is about the *common* senses. So, let's run our analysis again using a `filter`. Repeating tasks is a scenario that we will have to get used to: the transformation-visualization-cleaning process is iterative.

Also, let's create our first chart! For this we are using `ggplot()`, a package and function that follow the grammar of graphics, which states that charts are composed by (at least) 3 elements:

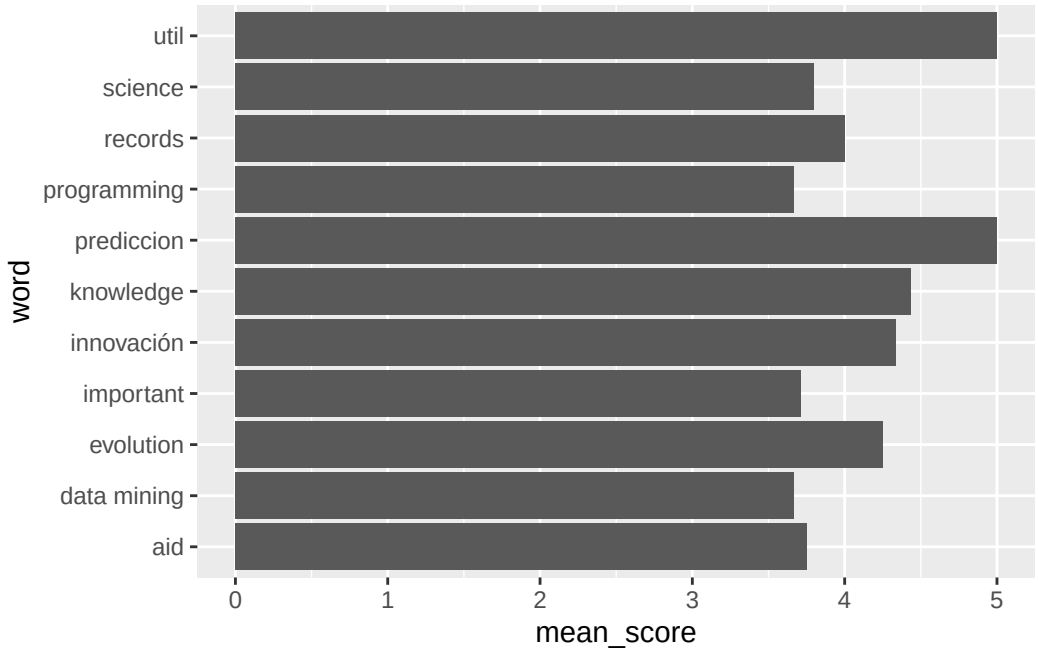
- data …
- … that we somehow map to visual properties of the chart or "aesthetics" (such as a certain column/variable for a chart axis),
- and a representation system or "geometry" (points, bars, areas, etc.)

More on `ggplot` here: https://r4ds.had.co.nz/data-visualisation.html

```
word_associations_frequency %>%
    filter(freq > 2) %>% # let's filter non-shared words
    slice_min(order_by = mean_score, n=10) %>% # 10 least likeable words
  ggplot(
    data = ., # since the data was passed through the pipe
    aes(
      y = word, # y axis will have the evoked word
      x = mean_score, # x axis will have their score
    )
  ) +
  geom_col()
```



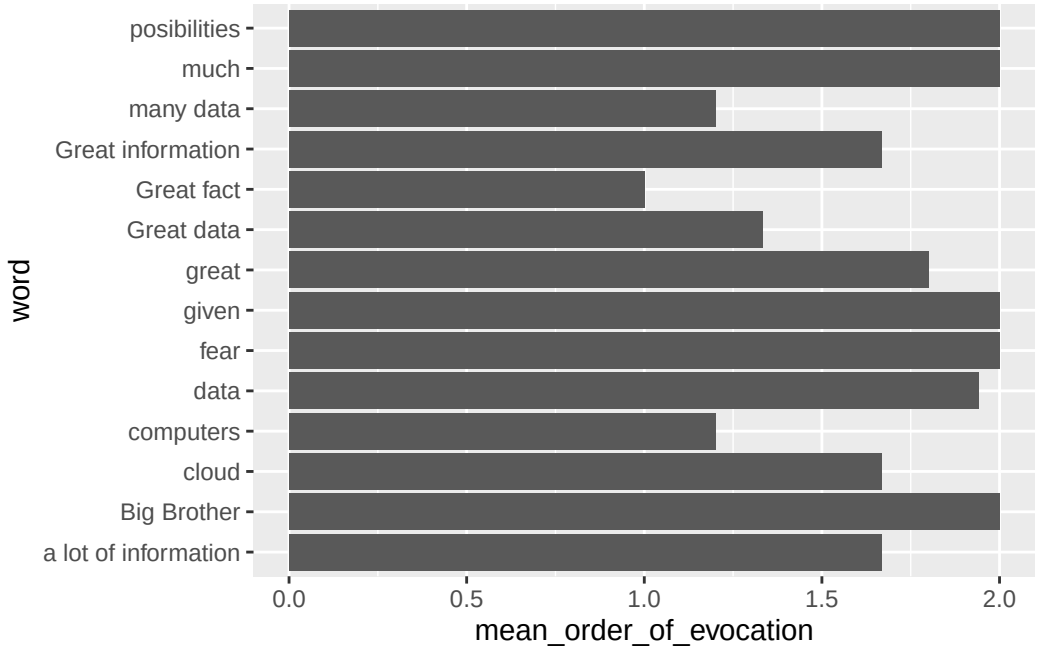
```r
word_associations_frequency %>%
    filter(freq > 2) %>% # let's filter non-shared words
    slice_max(order_by = mean_score, n=10) %>% # 10 most likeable words
  ggplot(
    data = ., # since the data was passed through the pipe
    aes(
      y = word, # y axis will have the evoked word
      x = mean_score, # x axis will have their score
    )
  ) +
  geom_col()
```



Let's repeat these analysis, now querying the words that were evoked the fastest, meaning the ones with the lower `mean_order_of_evocation`

```r
word_associations_frequency %>%
    filter(freq > 2) %>% # let's filter non-shared words
    slice_min(order_by = mean_order_of_evocation, n=10) %>% # 10 most salient words
  ggplot(
    data = ., # since the data was passed through the pipe
    aes(
```

```
    y = word, # y axis will have the evoked word
    x = mean_order_of_evocation, # x axis will have their mean OfE
  )
) +
geom_col()
```

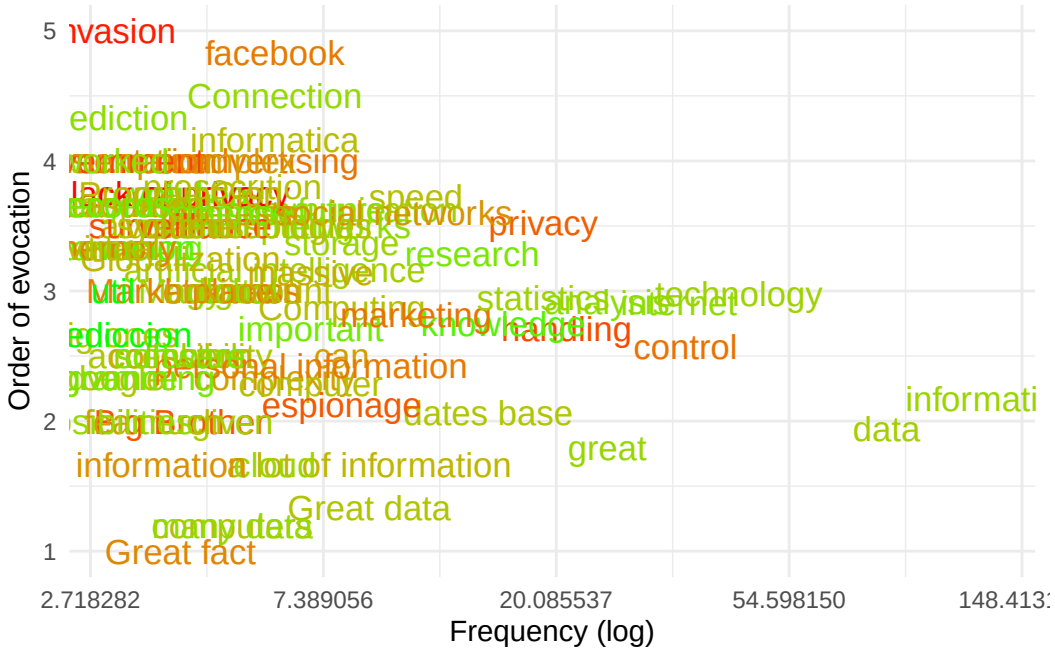


Final step! Let's bind all these analysis together to chart the evocations of "big data" in a way that allow us to visualize its "representational field". Specifically, we want to:

- draw points at the crossing of frequency (X) and order of evocation (Y) (for X we will apply logarithmic transformation, so we can see better the points);
- include the words in the graph, so that we can read them;
- show the word/idea valuation with fill colors, to quickly identify positive and negative senses (for this we are setting a green/red color palette).

```
word_associations_frequency %>%
  filter(freq > 2) %>%
  ggplot(
    data = . ,
    aes(
```

```
    x = freq,
    y = mean_order_of_evocation,
    label=word)) +
scale_x_continuous( trans='log' ) +
scale_colour_gradient(low = "red", high = "green", na.value = NA) +
geom_text( aes(size=7 , colour= mean_score),
           show.legend = FALSE, check_overlap = FALSE) +
labs(y="Order of evocation", x = "Frequency (log)") +
theme_minimal()
```



(2) Introduction to NLP and sentiment (lexicon) analysis

The goal of these lessons is to introduce ourselves to natural language processing (NLP), and to a particular task in that field: sentiment or polarity analysis.

Specifically, we will learn to:

1. pre-process text for further analysis;
2. cross tables (in our case, sentences and dictionaries);
3. perform basic sentiment analysis, preparing the data for the use of specific libraries.

You can check more on these tasks and topics here:

- https://www.tidytextmining.com/tidytext.html
- https://www.tidytextmining.com/sentiment.html
- https://r4ds.had.co.nz/relational-data.html#understanding-joins

We will work with a corpus of sentences, extracted from UK newspapers that include the words "big data."

The goal of our analysis will be to determine if big data is valued as a positive or negative phenomenon, according to the words in its co-text (sentences). Let's keep in mind that, as Paganoni (2019) suggests,

> Big data appears to be framed between two poles—data and information as opposed to rights and privacy—whose gap has of late been emphasized by a number of data scandals affecting business, health and politics, and culminating in the major unforeseen event of Cambridge Analytica and Facebook.

Throughout this tutorial we will work with several libraries, which we can install with the following code:

```
install.packages(c("readr", "tidyverse", "tidytext"))
install.packages(c("textdata"))
```

Let's load the data and filter sentences including "big data". The table includes a column `key` indicating if our interest keyword is present.

```
library(tidyverse)
library(readr)

sentences <- read_csv('https://raw.githubusercontent.com/gastonbecerra/textminingcourse/ma

sentences_bd <- sentences %>%
  filter(key==TRUE) %>%
  mutate(sentence_id = row_number()) # inserting an ID for each sentence
```

Preparing materials: corpus and lexicon

The goal of pre-processing text is to create a structured dataset that can be computed from an unstructured source, such as natural text. This can be achieved in several ways. In this exercise, we are trying a **dictionary-based approach** that will consist in looking for a *sentiment score* for the words of our sentences within a *lexicon*,and create a general score for our sentences.

In order to do so, we are going to:

- tokenize sentences, creating a *tibble* of their words;
- join tables (our sentences and lexicons);
- summarize a score for our sentences.

First, let's decompose our sentences in word *tokens*. These can be bind back because we are introducing an ID column referencing the sentence (row number) in the original dataset.

```
library(tidytext)

words_bd <- sentences_bd %>%
  unnest_tokens(output = word, input = sentence, token = "words") %>%
  select(-key) # we're dropping the key column
```

Second, let's prepare our *lexicon*. We are using generic sentiment lexicons from `tidytext` package. More info on these can be found here: https://www.tidytextmining.com/sentiment.html#the-sentiments-datasets

```
library(textdata)

affin <- get_sentiments("afinn") # import the lexicon

glimpse(affin)
```

```
Rows: 2,477
Columns: 2
$ word  <chr> "abandon", "abandoned", "abandons", "abducted", "abduction", "ab~
$ value <dbl> -2, -2, -2, -2, -2, -2, -3, -3, -3, -3, 2, 2, 1, -1, -1, 2, 2, 2~
```

Crossing / Joining tables

Now, let's cross tables! In particular, we are interested in seeing if the words that we extracted from our sentences match the words in the lexicons. For this, we will be using `_join` verbs from the `dplyr` package. More info on joins: https://r4ds.had.co.nz/relational-data.html#understanding-joins

We will `left_jpin()` our `words_bd` and `affin` (both have the `word` column). We will include the `value` column from the latter into the former (and leave blank if absent). Then, we are summarizing these values by sentence.

```
sentiment_bd <- words_bd %>%
  left_join( affin, by = "word" ) %>% # matching by word
  group_by( sentence_id ) %>%
  summarise(
    sentence_value = mean(value, na.rm = TRUE),
    sentence_found_words = paste(word[!is.na(value)], collapse = " ")
  )

glimpse(sentiment_bd)
```

```
Rows: 45
Columns: 3
$ sentence_id          <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15~
$ sentence_value       <dbl> 1.250000, 1.000000, -1.000000, 1.500000, -0.25000~
$ sentence_found_words <chr> "growing reach big sophisticated", "big big", "ag~
```

```
summary(sentiment_bd)
```

```
  sentence_id sentence_value    sentence_found_words
 Min.   : 1   Min.   :-1.0000   Length:45
 1st Qu.:12   1st Qu.: 0.4000   Class :character
 Median :23   Median : 1.0000   Mode  :character
 Mean   :23   Mean   : 0.8033
 3rd Qu.:34   3rd Qu.: 1.3333
 Max.   :45   Max.   : 2.0000
```

Let's explore these results a bit. Let's find sentences with the highest and lowest ratings. In order to better understand what we are evaluating, let's re-include the sentences, prior to our pre-processing.

```r
sentiment_bd %>% slice_max(order_by = sentence_value, n = 5) %>%
  inner_join(sentences_bd, by="sentence_id")
```

```
# A tibble: 5 x 6
  sentence_id sentence_value sentence_found_words           doc_id sente~1 key
        <int>          <dbl> <chr>                          <chr>  <chr>   <lgl>
1          27            2   big wealth                     BBDGU~ the gu~ TRUE
2          16            1.75 big care successful big       BBDFT~ ft.com~ TRUE
3          13            1.67 dedicated big attracting      BBDFT~ accord~ TRUE
4          42            1.67 care big success care big care 20161~ health~ TRUE
5          43            1.67 care big success care big care 20171~ priori~ TRUE
# ... with abbreviated variable name 1: sentence
```

```r
sentiment_bd %>% slice_min(order_by = sentence_value, n = 5) %>%
  inner_join(sentences_bd, by="sentence_id")
```

```
# A tibble: 6 x 6
  sentence_id sentence_value sentence_found_words          doc_id  sente~1 key
        <int>          <dbl> <chr>                         <chr>   <chr>   <lgl>
1           3           -1   aggressive big risk           BBDFT1~ "prope~ TRUE
2           9           -1   big catastrophe               BBDFT1~ "\"big~ TRUE
3          12           -1   big suffering imposing burden BBDFT1~ "a new~ TRUE
4          29           -0.5 big risks                     BBDIN1~ "indep~ TRUE
5          33           -0.5 big problem                   BBDIN1~ "big d~ TRUE
6          44           -0.5 big burdens                   201802~ "healt~ TRUE
# ... with abbreviated variable name 1: sentence
```

Let's evaluate the results and make decisions: Are they satisfactory to us, considering our objectives and the use that we will give to this data later? Do we want to introduce ad-hoc rules to improve these results? How many cases are lost by introducing rules? An ad-hoc rule seems to be required: perhaps "big" should not have intrinsic value... let's repeat filtering this.

```r
sentiment_bd <- words_bd %>%
  left_join( affin %>%
               filter(word != "big"), # filter "big" from affin
             by = "word" ) %>% # matching by word
  group_by( sentence_id ) %>%
  summarise(
```

```
    sentence_value = mean(value, na.rm = TRUE),
    sentence_found_words = paste(word[!is.na(value)], collapse = " ")
  )

glimpse(sentiment_bd)
```

```
Rows: 45
Columns: 3
$ sentence_id          <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15~
$ sentence_value       <dbl> 1.3333333, NaN, -2.0000000, 2.0000000, -0.6666667~
$ sentence_found_words <chr> "growing reach sophisticated", "", "aggressive ri~
```

```
summary(sentiment_bd)
```

```
  sentence_id sentence_value    sentence_found_words
 Min.   : 1   Min.   :-3.0000   Length:45
 1st Qu.:12   1st Qu.:-0.8333   Class :character
 Median :23   Median : 1.0000   Mode  :character
 Mean   :23   Mean   : 0.4489
 3rd Qu.:34   3rd Qu.: 1.8750
 Max.   :45   Max.   : 3.0000
              NA's   :14
```

(3) Topic modeling

In this exercise we will focus on topic modeling, an *unsupervised learning* technique that seeks to build topics or themes based on the distribution and correlation of words in a set of documents.

Throughout this exercise we will see:

- how to pre-process text for further analysis;
- how to build document/terms vectors;
- how to model topics;
- how to interpret a model.

We will work with a small corpus of news about big data. We will try to see how big data is portrayed and contextualized in news, so topic modeling can assist us in the analysis of discursive frames.

You can check more on these tasks and topics here:

- https://www.tidytextmining.com/topicmodeling.html
- https://bnosac.github.io/udpipe/docs/doc6.html

Throughout this tutorial we will work with several libraries, which we can install with the following code:

```
install.packages(c("readr", "tidyverse", "tidytext", "udpipe"))
install.packages(c("topicmodels", "stopwords" ))
```

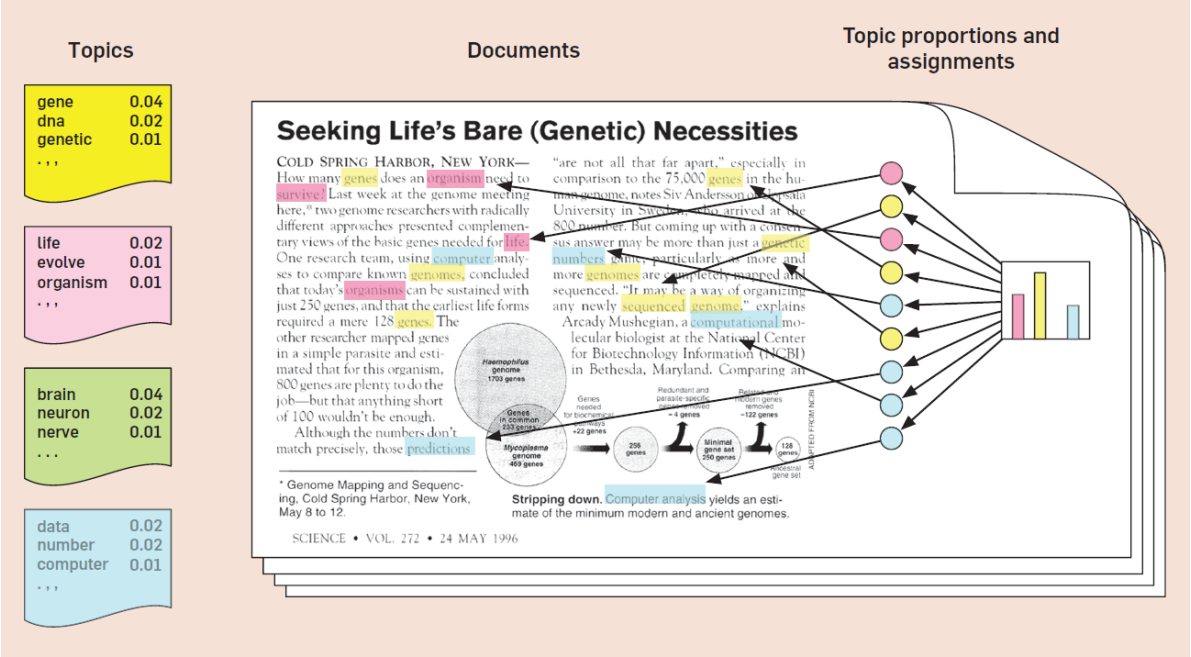
About the algorithm: LDA

There are several implementations of topic modeling. We will be using the most common (and basic) algorithm: Latent Dirichlet allocation, via the `topicmodels` pack.

> Topic models draw on the notion of distributional semantics and make use of the so-called bag of words assumption, i.e., the ordering of words within each document is ignored. To grasp the thematic structure of a document, it is sufficient to describe its distribution of words

Maier, Daniel, A. Waldherr, P. Miltner, G. Wiedemann, A. Niekler, A. Keinert, B. Pfetsch, et al. 2018. "Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology." *Communication Methods and Measures* 12 (2-3): 93–118. https://doi.org/10.1080/19312458.2018.1430754.

This model generates topics by proposing a certain distribution of all the words in the corpus, and calculating the distribution of these topics in each document.



(taken from Blei, David. 2012. "Probabilistic topic models." Communications of the ACM 55 (4): 77. https://doi.org/10.1145/2133806.2133826)

What is interesting about this way of understanding topics operational is that each topic can be understood as a semantic field, a set of words that are usually correlated in different documents. Then, when analyzing these results, we will try to infer a theme from the words that contribute the most to each topic. E.g., we could infer from a topic that the terms "sale", "product" and "buyer" contribute strongly to the topic "trade". According to one of the model's authors, the interpretability of most topics is a result of "the statistical structure of the language and how it interacts with LDA-specific probabilistic assumptions" (D. Blei, 2012, p. 79).

At the same time, the words are not exclusive to one topic but cross all topics with a relative contribution. This is precisely what interests us since we want to compare different ways of "contextualizing" the same term ("big data") across different topics, characterized by the use of certain other words.

Text pre-processing

As in any natural language processing task, we will start by loading the corpus and pre-processing text.

```
library(tidyverse)
library(tidytext)
```

```
library(readr)

corpus <- readr::read_csv(file = 'https://raw.githubusercontent.com/gastonbecerra/textmini
```

```
Rows: 57 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (2): doc_id, text

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
glimpse(corpus)
```

```
Rows: 57
Columns: 2
$ doc_id <chr> "BBDFT160304.txt", "BBDFT160401.txt", "BBDFT160516.txt", "BBDFT~
$ text   <chr> "\n                                        FT.com\n\n             ~
```

In order to complete our first analyses, we will perform several pre-processing tasks:

- We will do a morphosyntatic analysis to determine the different components of the sentence;
- We will reduce the words to their *lemmas*, basic word forms, without gender or conjugation;
- We will discard some common words, keeping only the most significant ones.

Lemmatization is a procedure that seeks to reduce words to their non-inflected or conjugated form. It is an alternative to *stemmization*, which attempts to heuristically and iteratively reduce the length of words, removing characters, until they are reduced to their root. Thus, the expression "`Google analyzes big data to infer the rate of contagion of the H1N1 flu`", is lemmatized as "`google analyze big data to infer the rate of contagion of the H1N1 flu`".

For these tasks we will work with `udPipe` package, and the `udpipe_annotate` function.

```
library(udpipe) # load package

model_en <- udpipe::udpipe_download_model('english') # download language model
```

```
model_en$file_model # reference to downloaded model
model_en <- udpipe_load_model(file = model_en$file_model) # load language model
```

```
corpus_anotated <- udpipe_annotate(
  object = model_en,
  x = corpus$text,
  doc_id = corpus$doc_id,
  trace = 10
  ) %>% as.data.frame(.)
```

```
2022-10-03 15:37:41 Annotating text fragment 1/57
2022-10-03 15:37:46 Annotating text fragment 11/57
2022-10-03 15:37:52 Annotating text fragment 21/57
2022-10-03 15:37:53 Annotating text fragment 31/57
2022-10-03 15:37:57 Annotating text fragment 41/57
2022-10-03 15:38:00 Annotating text fragment 51/57
```

We will use the `upos` information to filter the words that could be more significant: adjectives, verbs, and nouns. Here we omit the adverbs, since we are not interested in the possible modifications of the meaning between close words, such as negations or amplifications.

We will also introduce another filter: we will eliminate very common words in the language, which may not help us to identify a semantic field. For that we use a dictionary of common words, from the `stopwords` package, and we will eliminate those records with `filter()`.

```
library(stopwords)

corpus_anotated2 <- corpus_anotated %>%
  filter(upos=="ADJ"| upos=="VERB"| upos=="NOUN") %>%
  select( doc_id, lemma ) %>%
  filter(!lemma %in% stopwords::stopwords(language = "en"))

glimpse(corpus_anotated2)
```

```
Rows: 9,592
Columns: 2
$ doc_id <chr> "BBDFT160304.txt", "BBDFT160304.txt", "BBDFT160304.txt", "BBDFT~
$ lemma  <chr> "pm", "grow", "medical", "tech", "sector", "look", "rude", "hea~
```

Text vectors

Usually machine learning models are trained with data structured in the form of tables. When we work with text we must build these tables from the words of the document with which we are working. We do this with vectoring.

Let's suppose we have two documents with one sentence each:

1. `Big data is a the set of techniques to analyze and manipulate our thinking`
2. `Google analyzes big data to infer the rate of contagion of flu.`

If we convert the words on these sentences to *lemmas* and filter important words, they would read as:

1. `bigdata being technical analyze manipulate thought`
2. `google analyze bigdata infer rate contagion flu`

Let's see what these vectorized sentences would look like:

Doc	bigdata	being	technical	analyze	manipulate	infer	rate	contagion	flu
1	1	1	1	1	1	0	0	0	0
2	1	0	0	1	0	1	1	1	1

Here we have reduced each sentence to a "bag of words", which lost the context of formulation of verbal expressions, losing order. We are left with a general vocabulary that, for each sentence, notes the frequency of appearance with `1` and `0`. This data that is interpretable by a computer and can be used to train a machine learning model.

With the `count()` function it is very easy to build a vector, if we use the document id and the words as inputs. We can then convert our word distribution table to this type of object (document text matrix) using the `tidytext`'s `cast_dtm()` function.

```
corpus_dtm <- corpus_anotated2 %>%
  count(doc_id, lemma, sort = TRUE) %>%
  cast_dtm(doc_id, lemma, n)

corpus_dtm
```

```
<<DocumentTermMatrix (documents: 57, terms: 2895)>>
Non-/sparse entries: 6456/158559
Sparsity           : 96%
Maximal term length: 19
```

```
Weighting          : term frequency (tf)
```

The `DocumentTermMatrix` object shows us of the number of documents and unique words, and indicates a % of words that appear 0 times in a document (Sparsity).

Topic model with LDA

We are going to build the model with the `LDA()` function.

An important decision, which must be entered as a parameter to perform the analyses, is the number of topics to generate. Let's start with a judicious number, quick to test, and easy to examine, and come back to this problem later.

```r
library(topicmodels)

k_topics <- 5 # number of topics

corpus_tm <- topicmodels::LDA(
  corpus_dtm,
  k = k_topics,
  method = "Gibbs", # sampling method
  control = list(seed = 1:5, nstart=5, verbose=1000))
```

```
K = 5; V = 2895; M = 57
Sampling 2000 iterations!
Iteration 1000 ...
Iteration 2000 ...
Gibbs sampling completed!
K = 5; V = 2895; M = 57
Sampling 2000 iterations!
Iteration 1000 ...
Iteration 2000 ...
Gibbs sampling completed!
K = 5; V = 2895; M = 57
Sampling 2000 iterations!
Iteration 1000 ...
Iteration 2000 ...
Gibbs sampling completed!
K = 5; V = 2895; M = 57
Sampling 2000 iterations!
Iteration 1000 ...
```

```
Iteration 2000 ...
Gibbs sampling completed!
K = 5; V = 2895; M = 57
Sampling 2000 iterations!
Iteration 1000 ...
Iteration 2000 ...
Gibbs sampling completed!
```

```
corpus_tm
```

```
A LDA_Gibbs topic model with 5 topics.
```

Now let's export these results in 2 formats that we are interested in exploring, using the `tidy` function, and specifying which probabilities we are interested in:

- **beta**: topic x term probability;
- **gamma**: topic x document probability;

```
tm_beta <- tidy(corpus_tm, matrix = "beta")
glimpse(tm_beta)
```

```
Rows: 14,475
Columns: 3
$ topic <int> 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2~
$ term  <chr> "fat", "fat", "fat", "fat", "fat", "data", "data", "data", "data~
$ beta  <dbl> 4.822763e-05, 5.180005e-05, 3.866228e-05, 4.687134e-05, 1.688620~
```

```
tm_gamma <- tidy(corpus_tm, matrix = "gamma")
glimpse(tm_gamma)
```

```
Rows: 285
Columns: 3
$ document <chr> "BBDGU160408.txt", "BBDFT160516.txt", "BBDFT160401.txt", "BBD~
$ topic    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ gamma    <dbl> 0.03975364, 0.10447761, 0.09486166, 0.55517241, 0.11896552, 0~
```

The results produced by the model can be useful to infer topics. However, this implies an iterative process of interpretation by the researcher, which includes several moments:

1. labeling and organizing topics;
2. content analysis;
3. validation;

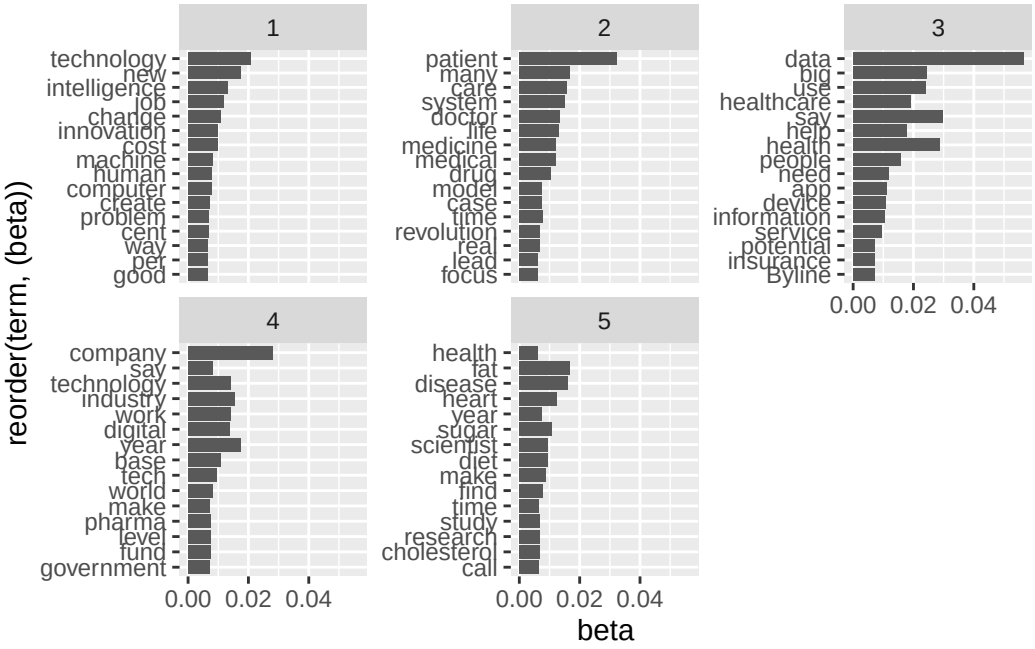
As in qualitative designs, we must take into consideration 2 issues: that the different tasks and moments of the analysis are not sequential but rather iterative, and that we will constantly make decisions that affect (forward) and inform (backwards) to other times; that all these decisions will be clearer and more robust if they are the product of consensus between different analysts who work autonomously and who document and exchange the reasons for their decisions.

Labeling and analyzing the model

Labeling is not a different process from qualitative coding, that is, the interactive interpretation of repeated ideas and expressions and the imputation of a code or label that identifies it. In terms of coding, preparing the data for this task is very easy: we simply list the terms that contribute the most to each topic.

```
tm_beta %>% # terms
  group_by(topic) %>%
  top_n(15) %>%
  ungroup() %>%
  arrange(topic, -beta) %>% # vamos a mostrarlo como grafico
  ggplot(aes(x=reorder(term, (beta)),y=beta)) +
    geom_col() +
    facet_wrap(~topic, scales = "free_y") +
  coord_flip()
```

```
Selecting by beta
```



The goal of this analysis is to (manually) evaluate if there is a coherent field of words in each topic, and assign a label that describes it.

```
topic_names <- rbind(
  c(topic = 1 , nombre = "1. technology issues"),
  c(topic = 2 , nombre = "2. medicine"),
  c(topic = 3 , nombre = "3. health apps?"),
  c(topic = 4 , nombre = "4. industry and companies"),
  c(topic = 5 , nombre = "5. health and care")
) %>% as_tibble() %>% mutate(topic=as.integer(topic))
```

It is important to bear in mind that not all topics will always present a coherent semantic field: in many cases they can refer to regularities typical of the type of communication that we are analyzing, or a mixture of words such that instead of allowing us to infer a univocal field (incoherent).

Next, we need to organize our topics:

- Should we discard irrelevant topics?: We can decide to filter out other topics that are irrelevant for our research purposes.
- Should we group topics?: Generally, in qualitative coding, this process is done in iterations, making more abstract and coherent inferences, allowing us to move from the codes to themes and arguments. The LDA model does not have this hierarchical structure,

but we can group or collapse topics into more general themes. This is almost always necessary when working with higher Ks.

Content analysis

One of the main benefits of TM is that it allows the researcher to perform a quick exploration of the corpus, inferring a probable way to organize and classify the documents, thus facilitating subsequent tasks, such as comparisons between documents, or between other corpus. Certainly **these types of automatic techniques do not replace content analysis and the researcher's interpretation,** but they can, however, **complement them in a mixed research design**. TM can be useful either in an inductive phase, contributing to the first explorations of the corpus, or by triangulating results to support the researcher's hypotheses.

In any cases, we should select a few representative documents to perform qualitative analysis.

Let's create a small sample, by identifying documents with highest probability in each topic using `tm_gamma`. Let us assume that with 5 documents we can perform our analyses (in qualitative samples, this number is not an *a priori* decision).

```
tm_gamma %>%
  group_by(topic) %>%
  slice_max(gamma, n=5)
```

```
# A tibble: 25 x 3
# Groups:   topic [5]
   document                   topic gamma
   <chr>                      <int> <dbl>
 1 BBDIN161127 BIS.txt            1 0.563
 2 BBDIN170410.txt                1 0.555
 3 BBDTL170504.txt                1 0.291
 4 BBDTI160620.txt                1 0.258
 5 BBDTL170924.txt                1 0.233
 6 20170308_FORBES.txt            2 0.502
 7 20180201_HiTAnalytics.txt      2 0.491
 8 BBDGU170705.txt                2 0.379
 9 BBDTI170819BIS.txt             2 0.343
10 BBDGU170505.txt                2 0.271
# ... with 15 more rows
```

Model validation

At the time of validation, we seek to find out how solid our interpretation of the model is.

One way to validate our model and inferences, we could propose some **content and metadata hypothesis** for further exploration. This could be data not used in the model construction, e.g., document publication date, or publisher information).

For its part, **statistical validation** seeks to measure how reliable the model is, in terms of how consistent its results are.

Perplexity is the most used measure in this type of tests. This is a metric that results from a held-out likelihood test in which, once the model has been trained with certain parameters, it is used to predict the topics of "new" documents for the model, that is, documents that were not part of the model construction. of the corpus with which it was trained. This generally is done for estimating different models with different parameters, like the number of topics (K). We could separate some documents from our corpus to have "new" documents, and train several TM models with different values of K; finally, we are going to plot the results of the perplexity tests. Since perplexity is a measure of inconsistency, a lower value is appropriate. Generally these values are achieved at a higher K.

(4) A few notes on Corpus building

There are several ways to create a corpus: using databases, querying API, scrapping content from web. We'll focus on the later:

Web scrapping

The process of automatically retrieving content from web pages is known as Web Scraping. In order to retrieve content from web pages there are 2 requirements to find out:

- the URLs of the pages we want to read: in many cases they can be dynamic URLs, with parameters and values, that we can manipulate.
- the HTML structure of the page where the content is: since we are probably not interested in all the content of the page (logos, texts, buttons) but only some texts, we have to be able to identify in which part of the page template the contents are inserted that interests us, in order to map them.

Both pieces of information vary from site to site and from system to system. For the first thing, it is convenient to understand the basics of the URL syntax: https://en.wikipedia.org/wiki/URL#Syntax; for the second, understand the basics of HTML markup https://en.wikipedia.org/wiki/HTML#Markup, including how to grab identifiers from a piece

of site, something for which the Chrome Dev.Tools (F12 navigating the page) can be very useful.

More info: https://towardsdatascience.com/web-scraping-in- r-using-rvest-and-selectorgadget-5fc5124547e

Parsing OJS

Open Journal Systems (OJS) is an editorial management system for academic journals. It is used mostly by universities and journals that do not have publishing agreements with large companies.

In this exercise we are going to try to recover the links to the articles on "big data" that are published in a list of journals that use the OJS system:

```
ojs_sites = c(
  "http://ojs.sociologia-alas.org/index.php/CyC/",
  "http://ediciones.ucsh.cl/ojs/index.php/TSUCSH/",
  "http://revistamexicanadesociologia.unam.mx/index.php/rms/"
  )
```

Our first requirement is to find out the URLs of the pages we want to read.

Remember that we want to find only articles that contain "big data", so we need to use the search function of each OJS. After doing some searching "by hand" and seeing how OJS builds its URLs (https://docs.pkp.sfu.ca/dev/documentation/en/architecture-routes), we can put together the ones for the search results:

```
criteria <- "%22big+data%22"
urls_to_scrape <- paste( ojs_sites , # vamos a contatenar pedazos de textos
                         "search/search?query=",
                         criteria,
                         sep = "")
urls_to_scrape
```

```
[1] "http://ojs.sociologia-alas.org/index.php/CyC/search/search?query=%22big+data%22"
[2] "http://ediciones.ucsh.cl/ojs/index.php/TSUCSH/search/search?query=%22big+data%22"
[3] "http://revistamexicanadesociologia.unam.mx/index.php/rms/search/search?query=%22big+data
```

We already have the first requirement ready: the URLs of the pages to read!

Our second requirement is to find some pattern to identify the piece of content that interests us. In our case, it is the links to the articles, so, again, we have to see how the OJS URLs

are composed. According to its convention, articles in OJS have a URL with this structure: site_url + "/article/view/" + article_id. With XPath, we can write an expression that filters this type of link: //a[contains(@href, "/article/view/")].

So, we have the second requirement ready: the pattern to identify the content!

Now to do the task we will create our own function, which will go through the `urls_to_scrape` and read and parse their content, with the help of the `rvest` package.

```
library(rvest)
```

```
Attaching package: 'rvest'
```

```
The following object is masked from 'package:readr':

    guess_encoding
```

```
get_articles <- function( url ) {
  url_con <- url(url, "rb") # open a connection
  webpage <- xml2::read_html(url_con) # read the html page
  close(url_con) # close connection
  xpath <- '//a[contains(@href, "/article/view/")]' # patter to identify links directing t
  links <- rvest::html_nodes(webpage, xpath = xpath) %>% # look for links
    html_attr('href') %>% # keep only URL
    return()
}

article_urls <- character()

for (i in 1:length(urls_to_scrape)) { # loop for urls_to_scrape
  message("scraping ",urls_to_scrape[i]) # show in screen which url about to parse
  article_urls <- c(article_urls, get_articles(urls_to_scrape[i]))
}
```

```
scraping http://ojs.sociologia-alas.org/index.php/CyC/search/search?query=%22big+data%22
```

```
scraping http://ediciones.ucsh.cl/ojs/index.php/TSUCSH/search/search?query=%22big+data%22
```

```
scraping http://revistamexicanadesociologia.unam.mx/index.php/rms/search/search?query=%22big
```

```
article_urls
```

```
[1] "http://ojs.sociologia-alas.org/index.php/CyC/article/view/245"
[2] "http://ojs.sociologia-alas.org/index.php/CyC/article/view/219"
[3] "http://ojs.sociologia-alas.org/index.php/CyC/article/view/213"
[4] "http://ojs.sociologia-alas.org/index.php/CyC/article/view/151"
[5] "http://ojs.sociologia-alas.org/index.php/CyC/article/view/131"
[6] "http://ediciones.ucsh.cl/ojs/index.php/TSUCSH/article/view/268"
[7] "http://revistamexicanadesociologia.unam.mx/index.php/rms/article/view/57723"
[8] "http://revistamexicanadesociologia.unam.mx/index.php/rms/article/view/57723/51185"
```

We have the links of the articles where it says "big data" in different journals. Now, we will surely be interested in doing a new scraping process to recover some metadata of those articles, or retrieving the full-content from a specific *galley* (publishing elements and options, like article in PDF, or auxiliary tables).

E.g., we could look for *keywords* in the metadata of the articles. Let's use the `ojsr` package instead.

```
library(ojsr)

metadata <- ojsr::get_html_meta_from_article(input_url = article_urls, verbose = TRUE)
```

```
trying url 1/8 http://ojs.sociologia-alas.org/index.php/CyC/article/view/245

scrapped http://ojs.soci ... found 53 elements using criteria .//meta

trying url 2/8 http://ojs.sociologia-alas.org/index.php/CyC/article/view/219

scrapped http://ojs.soci ... found 61 elements using criteria .//meta

trying url 3/8 http://ojs.sociologia-alas.org/index.php/CyC/article/view/213

scrapped http://ojs.soci ... found 54 elements using criteria .//meta

trying url 4/8 http://ojs.sociologia-alas.org/index.php/CyC/article/view/151

scrapped http://ojs.soci ... found 52 elements using criteria .//meta
```

```
trying url 5/8 http://ojs.sociologia-alas.org/index.php/CyC/article/view/131

scrapped http://ojs.soci ... found 53 elements using criteria .//meta

trying url 6/8 http://ediciones.ucsh.cl/ojs/index.php/TSUCSH/article/view/268

scrapped http://edicione ... found 48 elements using criteria .//meta

trying url 7/8 http://revistamexicanadesociologia.unam.mx/index.php/rms/article/view/57723

scrapped http://revistam ... found 51 elements using criteria .//meta

trying url 8/8 http://revistamexicanadesociologia.unam.mx/index.php/rms/article/view/57723

scrapped http://revistam ... found 51 elements using criteria .//meta
```

```
glimpse(metadata)
```

```
Rows: 423
Columns: 5
$ input_url         <chr> "http://ojs.sociologia-alas.org/index.php/CyC/articl~
$ meta_data_name    <chr> NA, "viewport", "generator", "DC.Creator.PersonalNam~
$ meta_data_content <chr> NA, "width=device-width, initial-scale=1.0", "Open J~
$ meta_data_scheme  <chr> NA, NA, NA, NA, NA, "ISO8601", "ISO8601", "ISO8601",~
$ meta_data_xmllang <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, "en", "es", "pt"~
```

```
metadata %>%
  filter(
      meta_data_name=="citation_keywords",
      trimws(meta_data_content)!=""
    ) %>% # filtering keywords
  mutate(keyword = trimws(tolower(meta_data_content))) %>%
  count(keyword, sort = TRUE)
```

```
                                                                                keyword n
1  big data, epistemología, metodología, ciencias sociales, redes sociales. 2
2    big data, epistemology, methodology, social sciences, social networks. 2
3                                                          relaciones públicas 2
4                                                               automatización 1
5                                                                     big data 1
6                                                  civilización transcultural 1
7                                                               colonialidad 1
8                                                  competencias profesionales 1
9                                                               comunicación 1
10                                               comunicación estratégica 1
11                                                                  covid-19 1
12                                                            crisis raigal 1
13                                                 cultura del trabajo 1
14                                                     cyberdemocracia 1
15                                                          empirismo. 1
16                                                   huellas digitales 1
17                                                   industria láctea 1
18                                                investigación social 1
19                                               métodos deinvestigación 1
20                                                          modernidad 1
21                                                    perfil de egreso 1
22                                                   políticas públicas 1
23                                                        sindicalismo 1
24                                          tecnologías de la información 1
25                                                  tecnologías digitales 1
26                                                                  vida 1
```

In addition to retrieving metadata from OJS, `ojsr` package allows:

```
?ojsr::get_issues_from_archive() # retrieve issues
?ojsr::get_articles_from_issue() # retrieve articles
?ojsr::get_galleys_from_article() # retrieve galleys
```

Some other resources

https://eur-lex.europa.eu/homepage.html?locale=pl

https://curia.europa.eu/jcms/jcms/Jo1_6308/

https://www.echr.coe.int/Pages/home.aspx?p=caselaw/HUDOC&c=

https://uncitral.un.org/sites/uncitral.un.org/files/media-documents/uncitral/en/facts_about_clout_eng_ebook.pdf

https://www.ilo.org/inform/online-information-resources/databases/terminology/lang--en/index.htm

https://op.europa.eu/pl/web/eu-vocabularies/det

https://op.europa.eu/pl/web/eu-vocabularies/dataset/-/resource?uri=http://publications.europa.eu/resource/dataset/eurovoc

https://unimelb.libguides.com/c.php?g=929605&p=6716619

https://www.icj-cij.org/en/cases

https://www.icc-cpi.int/cases