

Tidytext

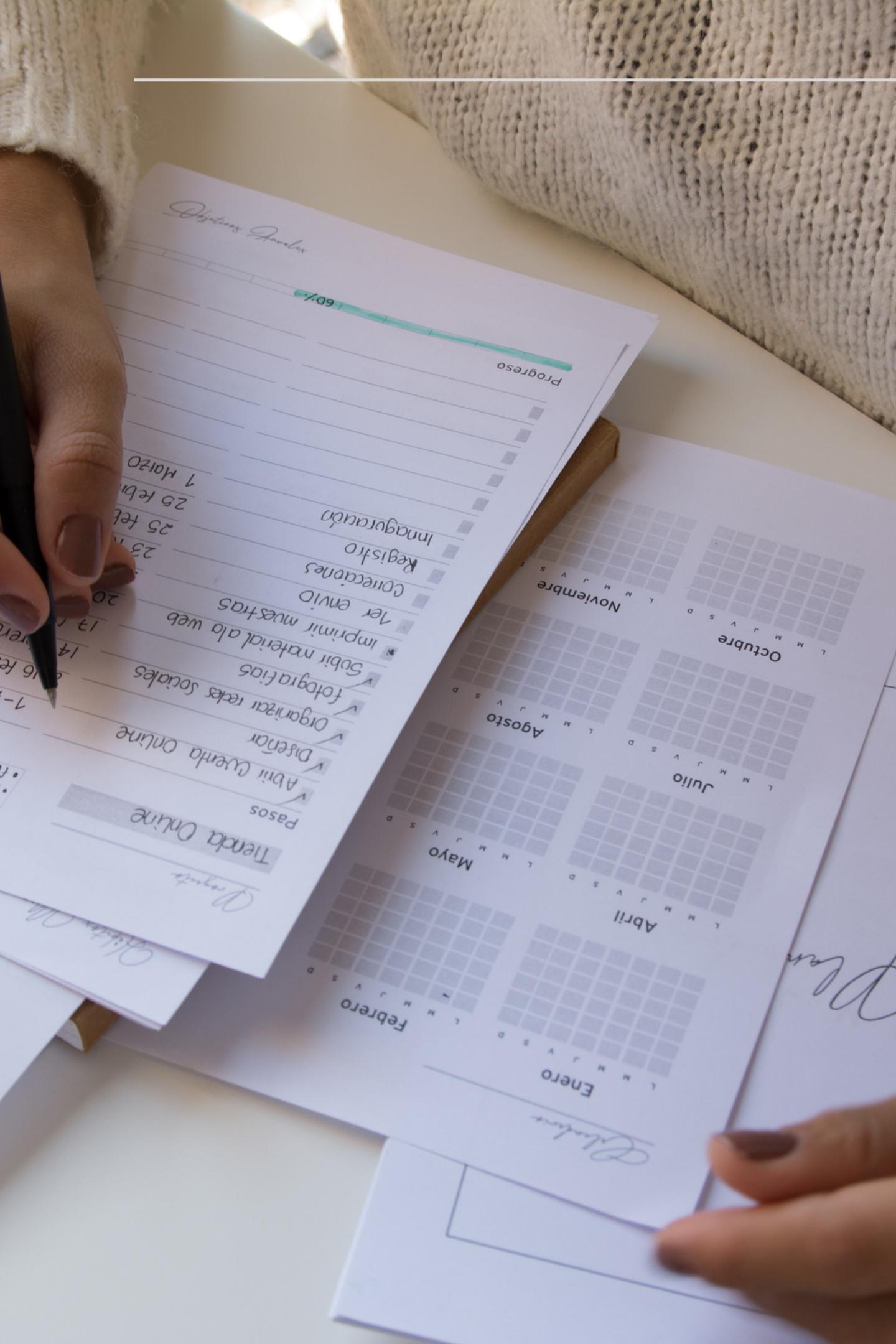


Biblioteka tidytext i macierze wystąpień w języku R



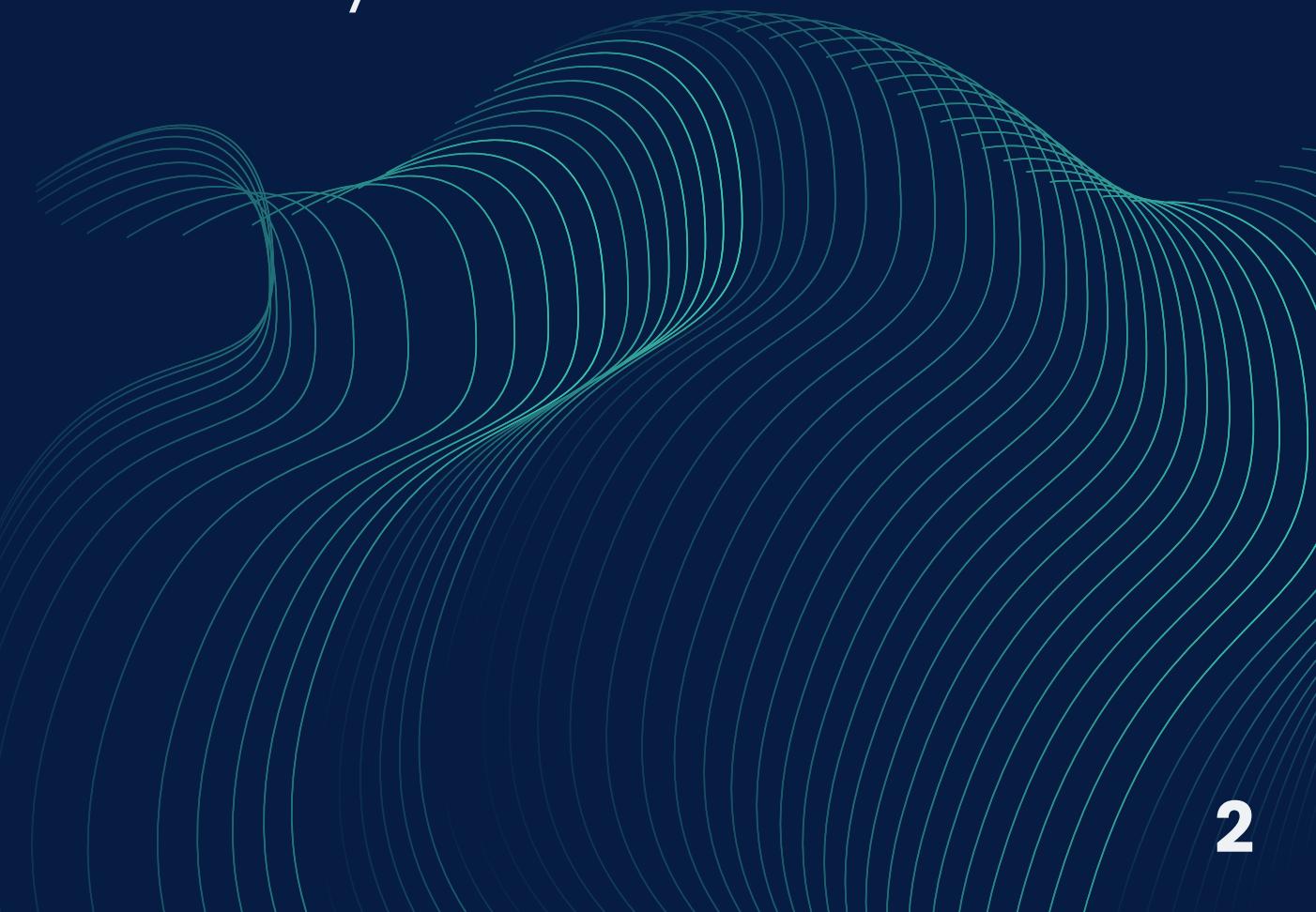
Kod wykorzystany w projekcie można znaleźć pod linkiem:

<https://github.com/zuzannapiekarczyk/Tidytext>



AGENDA

1. Biblioteka tidytext
2. Pozostałe biblioteki
3. Proces przygotowania danych
4. Proces tworzenia macierzy
5. Wizualizacja



1. Biblioteka tidytext

Wprowadzenie

Biblioteka tidytext to biblioteka programu R. Została **stworzona do pracy z tekstem w formie tabelarycznej**.



Biblioteka ta zawiera wiele narzędzi do analizy tekstu, takich jak:

- przetwarzanie tekstu,
- tokenizację,
- lematyzację.

Jest szczególnie przydatna dla osób pracujących z danymi tekstowymi, ponieważ umożliwia przetwarzanie tekstu w sposób **zorganizowany i łatwy do zrozumienia**.

Biblioteka ta jest oparta na idei *tidy data* (czyli danych uporządkowanych), która została zaproponowana przez Hadleya Wickhama.

INSTALACJA:

Biblioteka tidytext jest łatwa w użyciu i posiada bogatą dokumentację. Można ją zainstalować w R za pomocą polecenia **install.packages("tidytext")**.

Biblioteka tidytext jest zazwyczaj aktualizowana na bieżąco i jest kompatybilna z najnowszymi wersjami języka R.

2. Pozostałe biblioteki

Pozostałe biblioteki użyte do analizy

Poza biblioteką *tidytext* do analizy tekstu zostały użyte również poniższe biblioteki.

`library(stringr)`

Pakiet ten był niezbędny przy funkcji czyszczącej tekst, ponieważ zostały tam użyte funkcje:
`str_remove_all()`,
`str_replace_all()`,
`str_trim()`, czyli funkcje operujące na tekście.

`library(dplyr)`

Z powyższej biblioteki została użyta funkcja **`tibble()`**, służąca do zapisania tekstu w formie odpowiedniej tabeli. Ponadto pochodzą również z niej funkcje: **`anti_join()`**, **`left_join()`**, czyli funkcje łączące tabele; **`as_tibble()`**, oraz **`count()`**, służąca do zliczeń.

`library(textstem)`

Do lematyzacji tekstu została użyta funkcja **`lemmatize_words()`**, pochodząca z biblioteki *textstem*.

`library(quanteda)`

Pakiet *quanteda* był użyty głównie do generowania macierzy potrzebnych do analizy tekstu. Zostały wykorzystane z niej następujące funkcje:
`cast_dtm()`,
`cast_sparse()`,
`cast_dfm()`,
`as.matrix()`,
`apply()`,
`log()`.

`library(ggplot2)`

Ostatnia biblioteka została użyta do wykonania wizualizacji liczby wystąpień słów w formie wykresu z pomocą funkcji **`ggplot()`**.

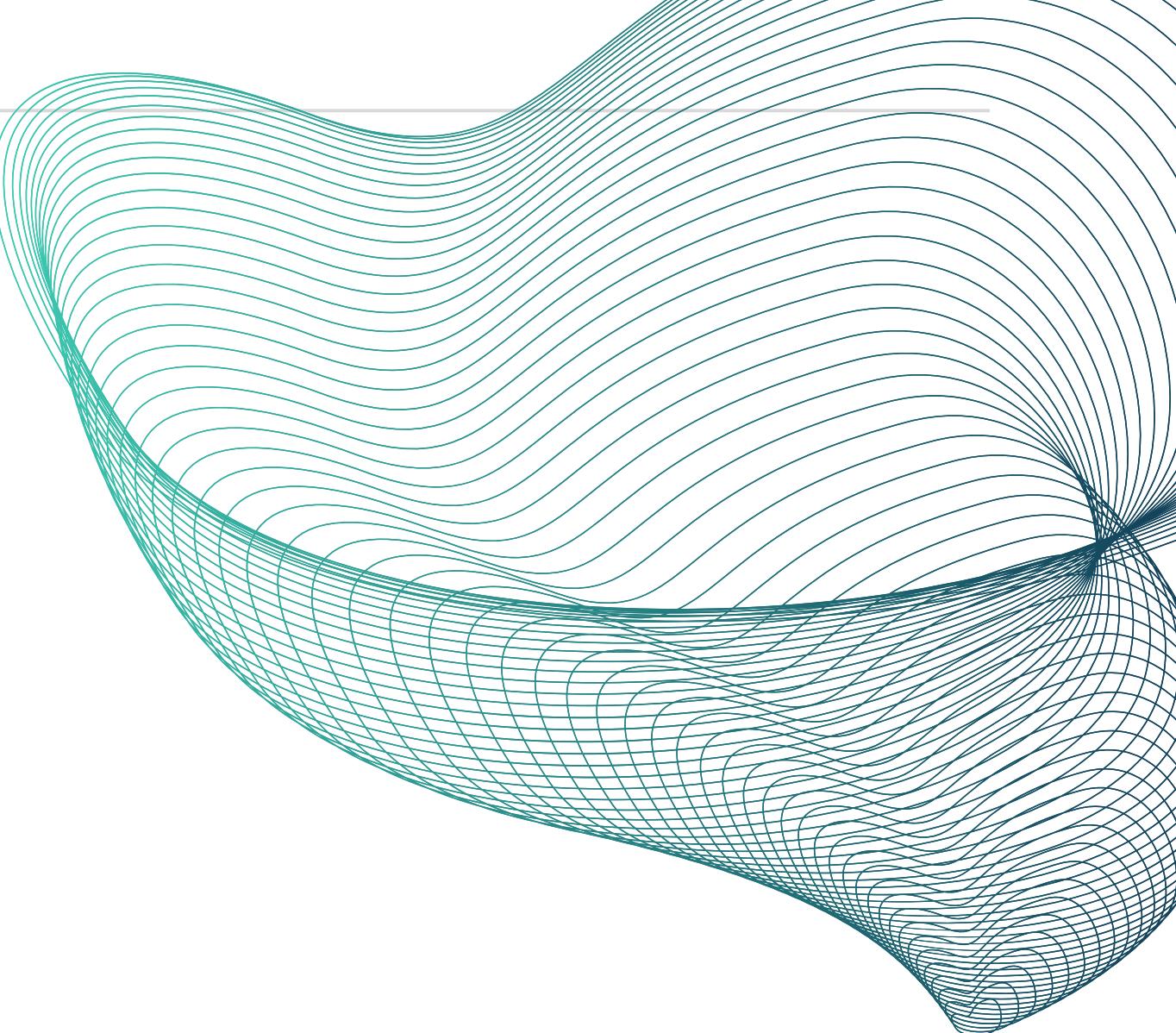
3. Proces przygotowania danych

Opis zbioru danych

W projekcie zostały wykorzystane **dane z polskimi przysłowiami**, które zostały udostępnione w trakcie laboratoriów. Dzięki ponownemu wykorzystaniu znanego zbioru możliwa będzie **lepsza ocena wyników uzyskanych podczas pracy z pakietem tidytext** w porównaniu z metodami zaprezentowanymi na zajęciach.

Dane zawierają 10 rekordów, gdzie każdy rekord to jedno przysłowie.

- [1] "Jesień tego nie zrodzi, czego wiosna nie zasiała"
- [2] "Czego wiosna nie zasiała – jesień nie urodzi"
- [3] "Św. Bartłomiej pogodny, jesień pogodna"
- [4] "Bartłomiej zwiastuje, jaka jesień następuje, i czy w przyszłym latku dożyjesz dostatku"
- [5] "Bartłomiej zwiastuje, jaka jesień następuje"
- [6] "Bartłomieja cały wrzesień naśladuje, i z nim jesień"
- [7] "Bartłomieja świętego dzień w jakiej zostaje porze, taką jesień bez ochyby daje"
- [8] "Jaki Bartek niesie dzień, taka będzie i jesień"
- [9] "Jaki Bartek niesie dzień, taka będzie i jesień"
- [10] "Jaki Bartek, taki wrzesień, jaki Marcin, taka zima"



3. Proces przygotowania danych

Czyszczenie tekstu

Aby wyczyścić tekst ze zbędnych znaków stworzono zmienną `clean`, która wykorzystuje biblioteki `stringr` oraz `dplyr`. Wykonuje ona następujące operacje:

1. Zamienia dwa lub więcej wystąpienia białego znaku na pojedynczy biały znak.
2. Usuwa znaki kontrolne z tekstu.
3. Zamienia wszystkie litery na małe.
4. Usuwa wszystkie znaki interpunkcyjne.
5. Usuwa cyfry.
6. Usuwa białe znaki z początku i końca tekstu.

Wszystkie te operacje służą do **usunięcia niepotrzebnych znaków i wyrazów z tekstu**, które mogą zakłócać analizę tekstu lub utrudniać jego przetwarzanie.

INPUT:

```
clean <- function(text){  
  require(stringr)  
  tekst_templ <- str_replace_all(text, "\\s{2,}", " ") %>%  
    str_replace_all("[cntrl]", " ") %>%  
    tolower() %>%  
    str_remove_all(":punct:") %>%  
    str_remove_all("\\d") %>%  
    str_trim()  
}  
clean_text <- clean(text)
```

3. Proces przygotowania danych

Tabela typu tibble

W kolejnym kroku przekształcono dane na tabelę typu tibble, która jest zgodna z ideą tidy.

Najpierw utworzono dataframe o nazwie `text_df` z dwoma kolumnami:

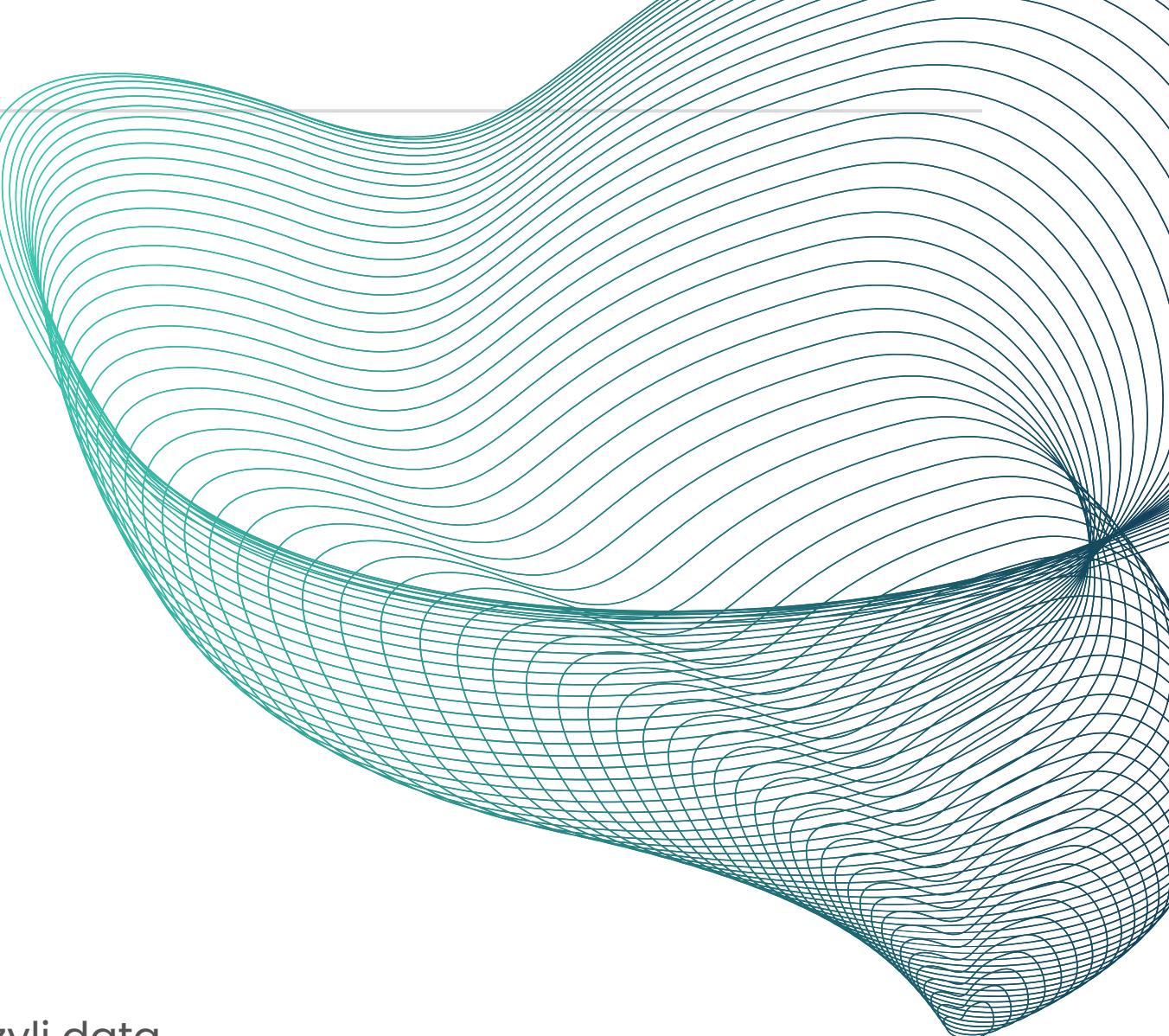
- 1.kolumna `line` zawiera numery linii tekstu,
- 2.kolumna `text` zawiera wyczyszczony tekst.

Funkcja tibble pochodzi z pakietu `tibble` i jest używana do tworzenia tzw. "tidy data frames", czyli data frame'ów, w których każda kolumna odpowiada jednej zmiennej, a każdy wiersz odpowiada jednej obserwacji.

W tym przypadku każda obserwacja to jedna linia tekstu, a każda zmienna to numer linii i tekst.

INPUT:

```
text_df <- tibble(line = 1:length(clean_text),  
text=clean_text)
```



3. Proces przygotowania danych

Tokenizacja

Następnie przeprowadzono tokenizację, czyli **podział przysłów na słowa/tokeny**.

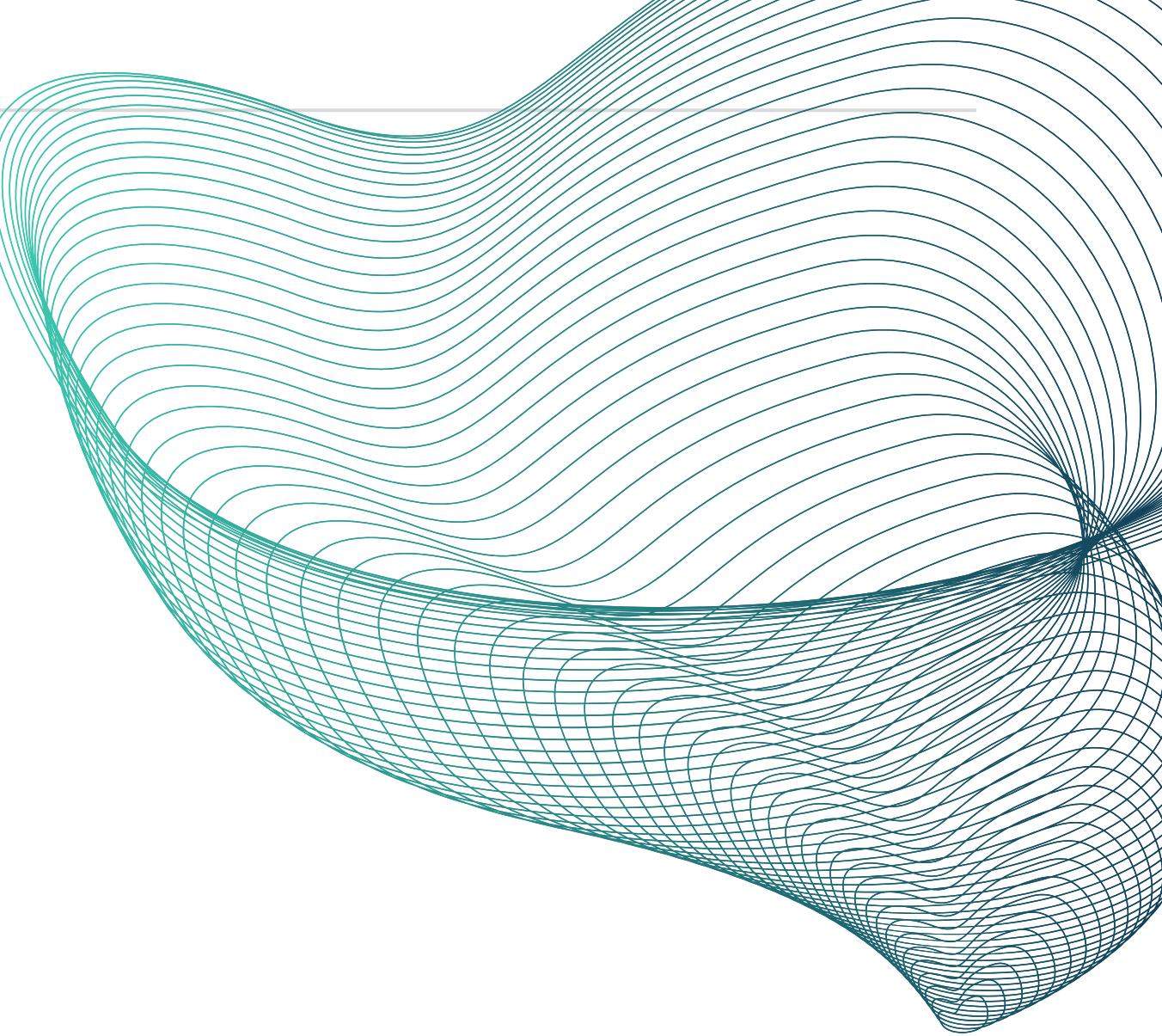
Funkcja unnest_tokens() z biblioteki `tidytext` dzieli tekst na pojedyncze słowa, a następnie umieszcza każde słowo w osobnym wierszu. Funkcja ta jest przydatna w analizie tekstu, ponieważ **umożliwia łatwe zliczanie wystąpień poszczególnych słów i ich analizę**.

Wynikowy dataframe o nazwie `text_tok` zawiera dwie kolumny:

1. `line`, zawierającą numer linii tekstu,
2. i `word`, zawierającą pojedyncze słowa z tej linii.

INPUT:

```
text Tok <- text_df %>%  
  unnest_tokens(word, text)
```



3. Proces przygotowania danych

Stopwords

W kolejnym kroku zostaje stworzona lista **stopwords** (słów-kluczy, które mają zostać usunięte z tekstu), która zostaje przekonwertowana na format tibble charakterystyczny dla pakietu **tidytext**.

Działanie kodu można podzielić na trzy etapy:

1. **Utworzenie listy "stopwords"** zawierającej słowa, które mają zostać usunięte z tekstu. Lista ta jest utworzona ręcznie i zawiera słowa takie jak "tak", "to", "i", "w" itp., które są bardzo często występującymi słowami i nie niosą dużo znaczenia w kontekście analizy tekstu.
2. Kod używa funkcji **stri_encode** z pakietu **stringi**, aby przekonwertować listę "stopwords" do formatu UTF-8.
3. Ostatecznie kod tworzy **tibble z kolumną word**, zawierającą listę słów z listy stopwords.

INPUT:

```
stopwords <- c('tak', 'to', 'tego', 'czego', 'jaka',  
'taka', 'i', 'czy', 'w', 'z', 'nim', 'jakiej', 'taką', 'jaki',  
'takać', 'takąż', 'taki', 'św')
```

```
stopwords <- stringi::stri_encode(stopwords, to  
= "UTF-8")
```

```
stopwords <- tibble(word=stopwords)
```

3. Proces przygotowania danych

Stopwords

Po stworzeniu listy stopwords **wykorzystana zostaje funkcja `anti_join()` do wyeliminowania stopwords z tekstu.** Operacja ta polega na zwróceniu tylko tych wierszy z ramki `text_tok`, dla których nie ma dopasowania z ramką `stopwords` w kolumnie `word`. W tym przypadku chodzi o usunięcie słów, które znajdują się na liście stopwords i są nieistotne z perspektywy analizy tekstu.

INPUT:

```
text_stop <- anti_join(text Tok, stopwords, by =  
join_by(word == word))
```



3. Proces przygotowania danych

Lematyzacja

Lematyzacja to **proces sprowadzenia słowa do jego podstawowej formy**, zwanej lematem. Lematyzacja jest podobna do stemmingu, ale zamiast obcinania końcówek słów, lematyzacja korzysta z reguł gramatycznych danego języka – w tym przypadku reguł j. polskiego.

Niestety, funkcja **lematize_words()** nie poradziła sobie z lematyzacją w j. polskim i słowa pozostały w niezmienionej formie. Dlatego też zdecydowano się na dokonanie lematyzacji w oparciu o słownik wykorzystywany podczas zajęć.

Wczytany został uproszczony słownik z pliku **slownik.txt** do zmiennej **dictionary**.

Wykorzystano pakiet **dplyr** oraz funkcję **left_join()**, która po dopasowaniu zmiennych zmieniła słowa na ich podstawowe formy.

INPUT:

```
dictionary <- read.csv2("slownik.txt", header=F,  
fileEncoding = "")  
  
dictionary <- as_tibble(dictionary, rownames =  
NULL)  
  
dictionary<- rename(dictionary, word = V2)  
  
text_lem <- text_stop %>%  
  left_join(dictionary, by = c("word" = "word"))  
  %>%  
  mutate(word = if_else(!is.na(V1), V1, word))  
text_lem <- text_lem %>% select(line, word)  
text_lem
```

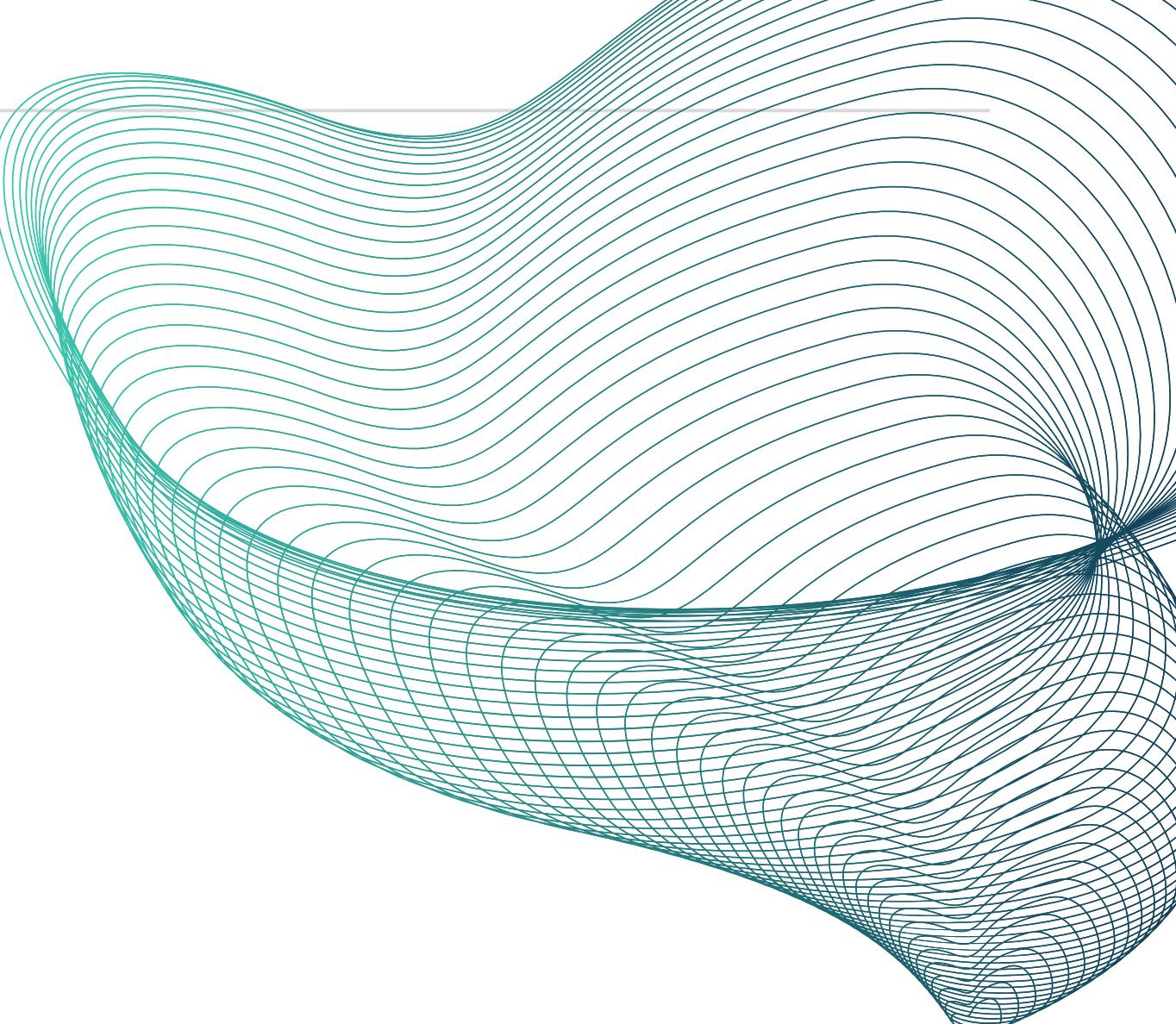
3. Proces przygotowania danych

Document-term tibble

Ostatnim elementem etapu przygotowania danych było **zliczenie ilości wystąpień poszczególnych termów** w obrębie danego przysłownia przy użyciu funkcji `count()` z pakietu `dplyr`.

W rezultacie powstał obiekt tibble o wymiarach 53x3.

W podejściu tidy nie ma potrzeby tworzenia bag of words. W nomenklaturze tidytext obiekt ten można nazwać tabelą document-term, która posłuży do stworzenia m.in. macierzy document-term.



INPUT:

```
text_lem <- text_lem %>%  
  select(line, word) %>%  
  count(line, word) %>%  
  rename(count = n, word = word)
```

4. Proces przygotowania macierzy

Macierz document-term DTM

Jedną z najczęstszych struktur, z którymi współpracują pakiety do text mining, jest macierz document-term (DTM) przedstawiająca **częstotliwość występowania każdego słowa w każdym dokumencie.**

- każdy wiersz reprezentuje jeden dokument (w tym przypadku jedno przysłowie),
- każda kolumna reprezentuje jeden termin,
- każda wartość zawiera liczbę wystąpień tego terminu w tym dokumencie.

INPUT:

```
dtm <- text_lem %>%  
  cast_dtm(document = line, term = word, value = count)  
  dtm
```

OUTPUT:

```
<<DocumentTermMatrix (documents: 10, terms: 28)>>  
Non-/sparse entries: 53/227  
Sparsity : 81%  
Maximal term length: 10  
Weighting : term frequency (tf)
```

WNIOSKI:

- 10 dokumentów,
- 28 unikalnych termów,
- 53 wpisy (wiele kombinacji dokument-słowo jest pusta),
- wartość rzadkości: 81% (większość kombinacji dokument-słowo nie występuje w danych),
- najdłuższe słowo: 10 liter.

4. Proces przygotowania macierzy

Macierz document-feature (DFM)

Typowym dla podejścia tidy są macierze document-feature (DFM), które również należą do pakietu [quantada](#). Funkcja `cast_dfm()` przekształca obiekt tibble w macierz DFM, która wygląda podobnie jak macierze DTM tworzone w innych pakietach/językach programowania.

Wynikiem działania kodu jest tabela, która w miejscach, gdzie kombinacja dokument-feature jest pusta – wyświetla zera.

INPUT:

```
dfm <- text_lem %>%  
  cast_dfm(document = line, term =  
  word, value = count)  
dfm
```

OUTPUT:

```
Document-feature matrix of: 10 documents, 28 features (81.07% sparse) and 0 docvars.  
  features  
  docs jesień nie wiosna zasiać zrodzić urodzić bartłomiej pogodny dostatek dożyć  
  1     1   2     1     1     1     0     0     0     0     0     0  
  2     1   2     1     1     0     1     0     0     0     0     0  
  3     1   0     0     0     0     0     0     1     2     0     0  
  4     1   0     0     0     0     0     0     1     0     1     1  
  5     1   0     0     0     0     0     0     1     0     0     0  
  6     1   0     0     0     0     0     0     1     0     0     0  
[ reached max_ndoc ... 4 more documents, reached max_nfeat ... 18 more features ]
```

4. Proces przygotowania macierzy

Macierz sparse document-term SDTM

W kolejnym kroku utworzono macierz rzadką (SDTM) przy użyciu funkcji **`cast_sparse()`** z pakietu [quanteda](#), w której każdy wiersz odpowiada jednemu dokumentowi (przysłowiowi), a każda kolumna reprezentuje dane słowo (token). Na przecięciach wierszy i kolumn wyliczone zostały liczebności tokenów w obrębie poszczególnych dokumentów.

Macierz typu sparse charakteryzuje się tym, że w miejscach, **gdzie kombinacja dokument-słowo jest pusta, wyświetdana jest kropka zamiast zera** i przypomina ona bardziej typową macierz DTM.

INPUT:

```
dfm <- text_lem %>%
  cast_dfm(document = line, term =
word, value = count)
dfm
```

OUTPUT:

```
10 x 28 sparse Matrix of class "dgCMatrix"
[[ suppressing 28 column names 'jesień', 'nie', 'wiosna' ... ]]

 1  1 2 1 1 1 . . . . . . . . . . . . . . . . . . . .
 2  1 2 1 1 . 1 . . . . . . . . . . . . . . . . . . . .
 3  1 . . . . . 1 2 . . . . . . . . . . . . . . . . . .
 4  1 . . . . . 1 . 1 1 1 1 1 1 1 . . . . . . . . . .
 5  1 . . . . . 1 . . . . 1 . 1 . . . . . . . . . . .
 6  1 . . . . . 1 . . . . . . 1 1 1 . . . . . . . . .
 7  1 . . . . . 1 . . . . . . . 1 1 1 1 1 1 1 . . .
 8  1 . . . . . 1 . . . . . . . . 1 . . . . 1 1 . .
 9  1 . . . . . 1 . . . . . . . . . 1 . . . . 1 1 . .
10 . . . . . 1 . . . . . . . . . . 1 . . . . . . 1 1
```


4. Proces przygotowania macierzy

Term Frequency-Inverse Document Frequency

- usuwa termy, które są charakterystyczne dla wszystkich dokumentów i zostawia tylko te unikalne/różnicujące)

Aby przekształcić macierz FDM na macierz TF-IDF, należy pomnożyć każdy element macierzy TF przez wagę IDF dla danego termu:

$$idf(t) = \log(N/df(t))$$

gdzie N to liczba wszystkich dokumentów, a df(t) to liczba dokumentów zawierających term t.

W wyniku tej sekwencji kodu, zmienna `tfidf_matrix` będzie zawierała macierz TF-IDF, w której każdy element będzie wynikiem pomnożenia odpowiadającego mu elementu w macierzy TF przez wagę IDF dla danego termu.

Ta operacja pozwala na **wyodrębnienie najistotniejszych słów (termów) w danym dokumencie**.

INPUT:

```
tf_matrix <- t(apply(fdm, 1, function(x) x/sum(x)))
idf_weights <- log(nrow(fdm)/rowSums(fdm > 0))
tfidf_matrix <- tf_matrix * idf_weights
```

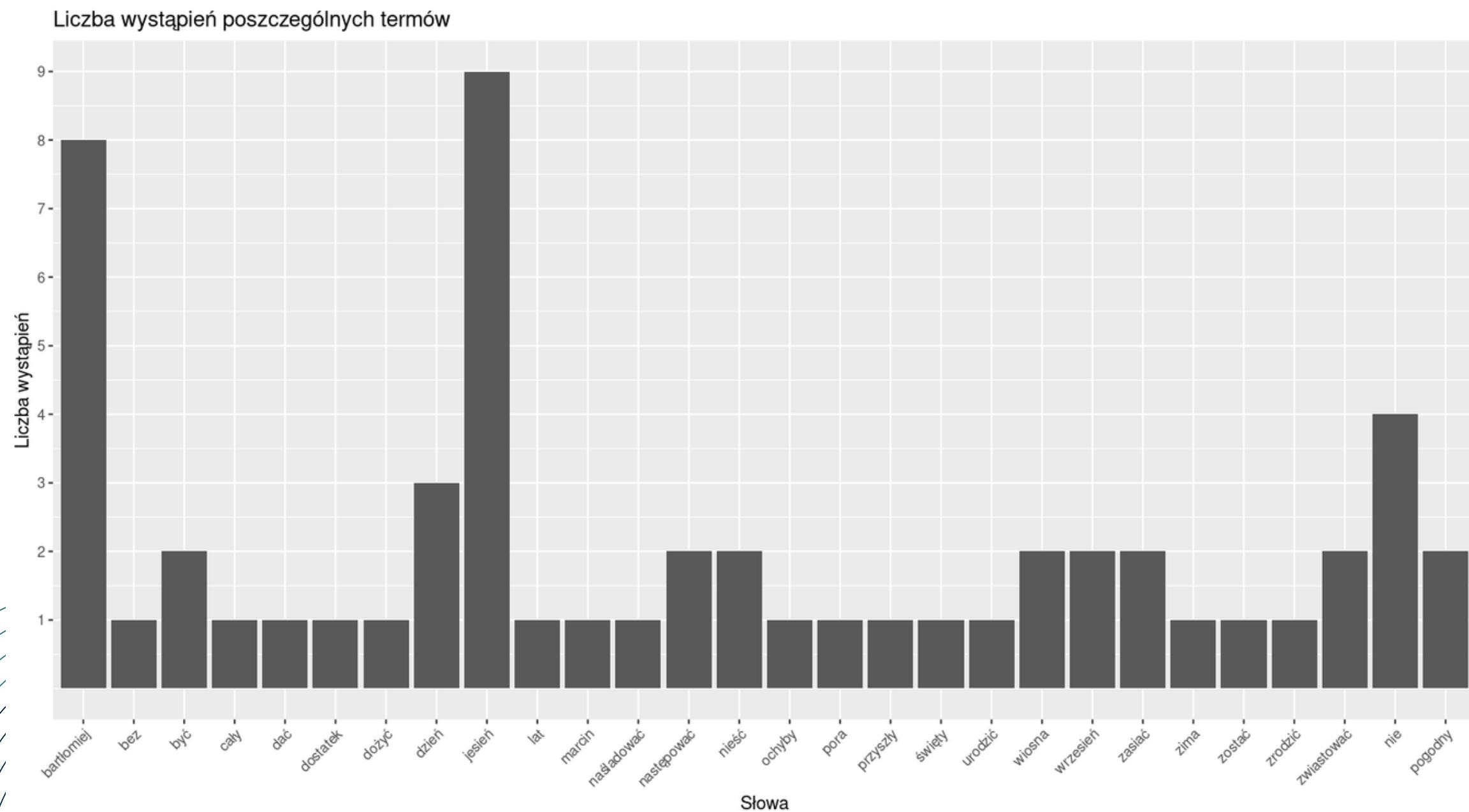
OUTPUT:

Terms	Docs	1	2	3	4	5
jesień		0.1261089	0.1261089	0.1261089	0.1261089	0.1261089
nie		1.3195287	1.3195287	0.0000000	0.0000000	0.0000000
wiosna		1.3195287	1.3195287	0.0000000	0.0000000	0.0000000
zasiać		1.3195287	1.3195287	0.0000000	0.0000000	0.0000000
zrodzić		3.3322045	0.0000000	0.0000000	0.0000000	0.0000000
urodzić		0.0000000	3.3322045	0.0000000	0.0000000	0.0000000
bartłomiej		0.0000000	0.0000000	0.1565954	0.1565954	0.1565954
pogodny		0.0000000	0.0000000	3.3322045	0.0000000	0.0000000
dostatek		0.0000000	0.0000000	0.0000000	3.3322045	0.0000000
dożyć		0.0000000	0.0000000	0.0000000	3.3322045	0.0000000
lat		0.0000000	0.0000000	0.0000000	3.3322045	0.0000000
następować		0.0000000	0.0000000	0.0000000	1.3195287	1.3195287
przyszły		0.0000000	0.0000000	0.0000000	3.3322045	0.0000000
zwiastować		0.0000000	0.0000000	0.0000000	1.3195287	1.3195287
cały		0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

5. Wizualizacja

Wizualizacja

Liczba wystąpień poszczególnych termów

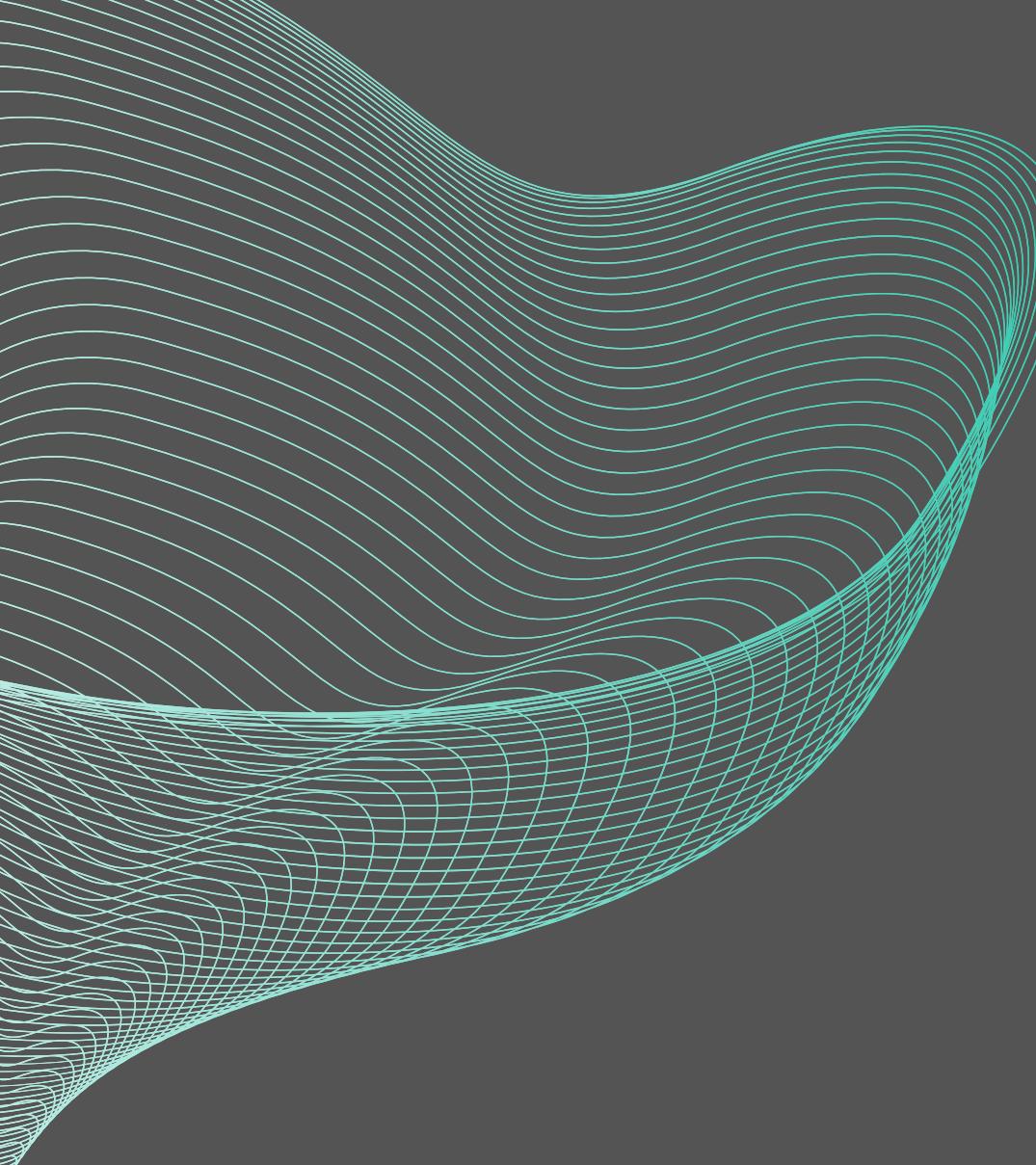


INPUT:

```
ggplot(text_lem, aes(x = reorder(word, count),  
y = count)) +  
  geom_bar(stat = "identity") +  
  theme(axis.text.x = element_text(angle = 45,  
hjust = 1)) +  
  labs(x = "Słowa", y = "Liczba wystąpień") +  
  ggtitle("Liczba wystąpień poszczególnych  
termów") +  
  scale_y_continuous(breaks =  
c(1,2,3,4,5,6,7,8,9,10))
```

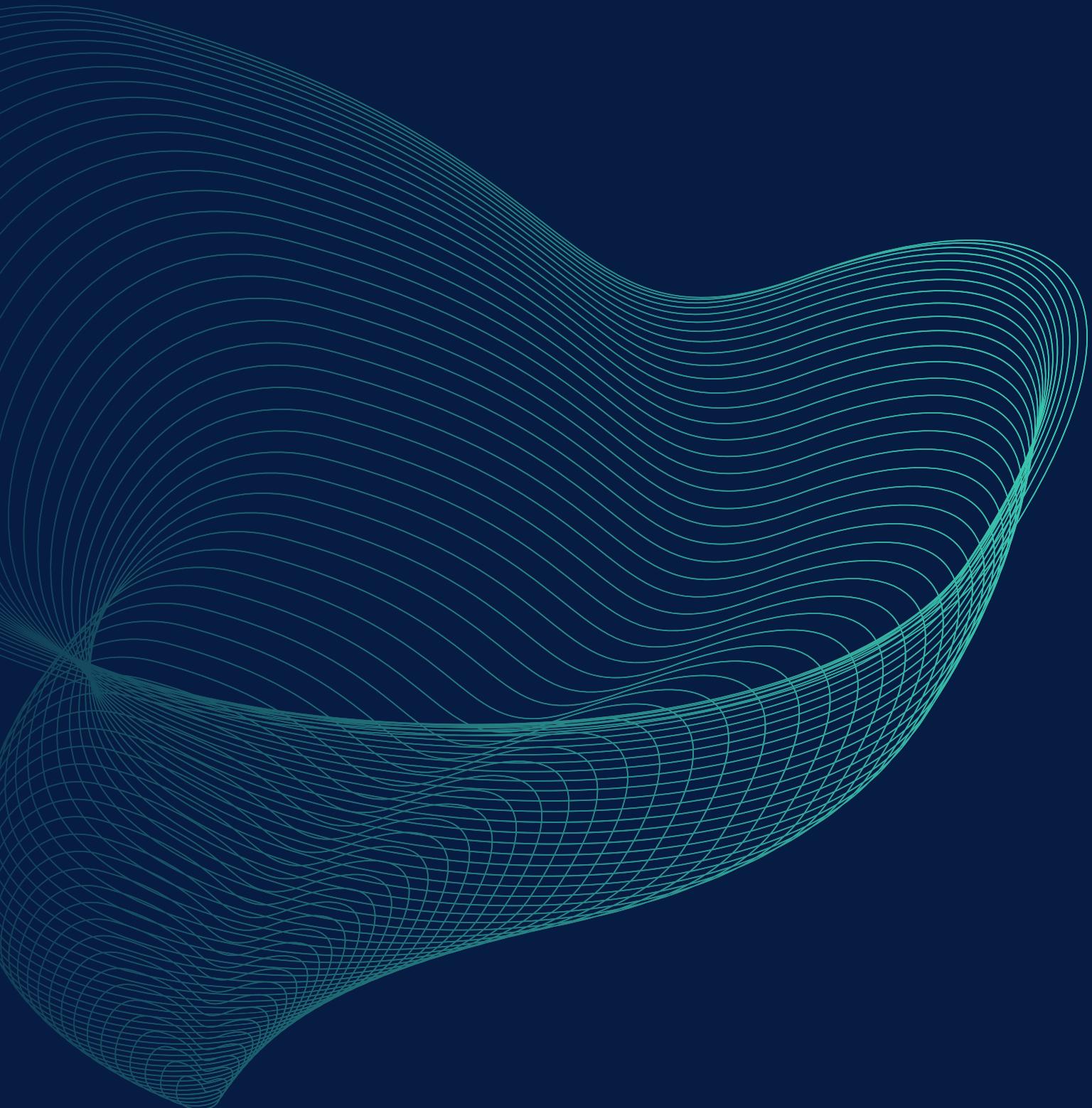
W wizualizacji wykorzystano pakiet ggplot2 w R do tworzenia wykresu słupkowego liczby słów w zestawie danych tekstowych.

Oś X ukazuje unikalne słowa w zbiorze danych, a **oś Y** – ile razy każde słowo pojawia się w zbiorze danych.



Źródła:

- "Text Mining with R", Julia Silge & David Robinson, O'Reilly, 2017
- <https://github.com/zuzannapiekarczyk/Tidytext>
- <https://github.com/dgrtwo/tidy-text-mining/blob/master/index.Rmd>
- <https://www.tidytextmining.com/tidytext.html>
- <https://bookdown.org/Maxine/tidy-text-mining/tidy-text-format.html>
- <https://github.com/rstudio/rmarkdown>
- <https://github.com/juba/rmdformats>
- <https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>
- <https://holtzy.github.io/Pimp-my-rmd/>
- <https://www.tidytextmining.com/dtm.html>
- <https://www.unsplash.com>



Dziękujemy
za uwagę!