# Caparezza Sentiment Analysis

# Caparezza Sentiment Analysis

In this notebook, a text mining process on the works of the Italian singer **Caparezza** will be performed, with the intention of finding recurring themes throughout his albums and songs.

Much of the inspiration for this project comes from this repository (https://tm4ss.github.io/docs/index.html).

# Get the lyrics

```
library(geniusr)
library(tidyverse)
```

```
## Warning: il pacchetto 'lubridate' è stato creato con R versione 4.2.3
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.0     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
## ✓ ggplot2   3.4.1     ✓ tibble    3.2.0
## ✓ lubridate 1.9.2     ✓ tidyr     1.3.0
## ✓ purrr     1.0.1
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force all conflict
s to become errors
```

```
library(tidytext)
```

```
## Warning: il pacchetto 'tidytext' è stato creato con R versione 4.2.3
```

```
library(quanteda)
```

```
## Warning: il pacchetto 'quanteda' è stato creato con R versione 4.2.3
```

```
## Package version: 3.3.1
## Unicode version: 13.0
## ICU version: 69.1
## Parallel computing: 8 of 8 threads used.
## See https://quanteda.io for tutorials and examples.
```

```
library(quanteda.textstats)
```

```
## Warning: il pacchetto 'quanteda.textstats' è stato creato con R versione 4.2.3
```

```r
library(udpipe)
```

```
## Warning: il pacchetto 'udpipe' è stato creato con R versione 4.2.3
```

```r
library(wordcloud)
```

```
## Warning: il pacchetto 'wordcloud' è stato creato con R versione 4.2.3
```

```
## Caricamento del pacchetto richiesto: RColorBrewer
```

```r
library(textdata)
```

```
## Warning: il pacchetto 'textdata' è stato creato con R versione 4.2.3
```

```r
library(reshape2)
```

```
## Warning: il pacchetto 'reshape2' è stato creato con R versione 4.2.3
```

```
##
## Caricamento pacchetto: 'reshape2'
##
## Il seguente oggetto è mascherato da 'package:tidyr':
##
##     smiths
```

```r
library(igraph)
```

```
## Warning: il pacchetto 'igraph' è stato creato con R versione 4.2.3
```

```
##
## Caricamento pacchetto: 'igraph'
##
## I seguenti oggetti sono mascherati da 'package:lubridate':
##
##     %--%, union
##
## I seguenti oggetti sono mascherati da 'package:dplyr':
##
##     as_data_frame, groups, union
##
## I seguenti oggetti sono mascherati da 'package:purrr':
##
##     compose, simplify
##
## Il seguente oggetto è mascherato da 'package:tidyr':
##
##     crossing
##
## Il seguente oggetto è mascherato da 'package:tibble':
##
##     as_data_frame
##
## I seguenti oggetti sono mascherati da 'package:stats':
##
##     decompose, spectrum
##
## Il seguente oggetto è mascherato da 'package:base':
##
##     union
```

Lyrics for the songs can be obtained using the *geniusr* package, available at https://github.com/ewenme/geniusr (https://github.com/ewenme/geniusr). Follow the instructions at that link to set up an API key and use it to query the Genius lyrics database.

```r
Sys.setenv(GENIUS_API_TOKEN = "mjJVgXu4_yQNgYXFYV72Q9uXCIdTFFq3s3c8PPcUT6G1tlAedbpg0dpdZv6KJQ
W7")

artist_id <- search_artist("Caparezza")$artist_id
artist_songs <- get_artist_songs(artist_id)

songs_ids <- c()
for (song in artist_songs$content) {
  song_id <- song$id
  songs_ids <- songs_ids %>% append(song_id)
}

songs_titles <- c()
songs_albums <- c()
songs_lyrics <- c()
for (i in 1:length(songs_ids)) {
  cat("Getting", i, "of", length(songs_ids), "id:", songs_ids[i], "\n")
  song <- get_song(songs_ids[i])$content
  song_title <- song$title
  song_album <- song$album$name
  song_lyrics <- list(get_lyrics_id(songs_ids[i]))
  # some songs have no album associated with them, so we won't consider them
  # additionally, some songs have no lyrics, we are excluding them too
  if (!is.null(song_album) & nrow(song_lyrics[[1]]) != 0){
    songs_titles <- songs_titles %>% append(song_title)
    songs_albums <- songs_albums %>% append(song_album)
    songs_lyrics <- songs_lyrics %>% append(song_lyrics)
  }
}
```

```
## Getting 1 of 220 id: 1150532
## Getting 2 of 220 id: 4699285
## Getting 3 of 220 id: 3125243
## Getting 4 of 220 id: 5282824
## Getting 5 of 220 id: 2844092
## Getting 6 of 220 id: 3258705
## Getting 7 of 220 id: 1742623
## Getting 8 of 220 id: 1567047
## Getting 9 of 220 id: 3169827
## Getting 10 of 220 id: 5211044
## Getting 11 of 220 id: 1465254
## Getting 12 of 220 id: 5064919
## Getting 13 of 220 id: 6655588
## Getting 14 of 220 id: 87063
## Getting 15 of 220 id: 1764703
## Getting 16 of 220 id: 1660583
## Getting 17 of 220 id: 6655582
## Getting 18 of 220 id: 6655578
## Getting 19 of 220 id: 1551384
## Getting 20 of 220 id: 1574292
## Getting 21 of 220 id: 1720346
## Getting 22 of 220 id: 3259796
## Getting 23 of 220 id: 1697016
## Getting 24 of 220 id: 3259788
## Getting 25 of 220 id: 1629161
## Getting 26 of 220 id: 5064899
## Getting 27 of 220 id: 1653990
## Getting 28 of 220 id: 6655591
## Getting 29 of 220 id: 508444
## Getting 30 of 220 id: 1765924
## Getting 31 of 220 id: 3169738
## Getting 32 of 220 id: 5206097
## Getting 33 of 220 id: 3258704
## Getting 34 of 220 id: 6655584
## Getting 35 of 220 id: 1301750
## Getting 36 of 220 id: 4699288
## Getting 37 of 220 id: 477158
## Getting 38 of 220 id: 1695687
## Getting 39 of 220 id: 1780748
## Getting 40 of 220 id: 3206374
## Getting 41 of 220 id: 1179348
## Getting 42 of 220 id: 3259783
## Getting 43 of 220 id: 1468333
## Getting 44 of 220 id: 6655581
## Getting 45 of 220 id: 3258706
## Getting 46 of 220 id: 1020332
## Getting 47 of 220 id: 1956985
## Getting 48 of 220 id: 138620
## Getting 49 of 220 id: 1972029
## Getting 50 of 220 id: 1808413
## Getting 51 of 220 id: 6655585
## Getting 52 of 220 id: 6653350
## Getting 53 of 220 id: 6655589
## Getting 54 of 220 id: 1818868
## Getting 55 of 220 id: 1129028
```

```
## Getting 56 of 220 id: 1666883
## Getting 57 of 220 id: 2303076
## Getting 58 of 220 id: 3169733
## Getting 59 of 220 id: 4082891
## Getting 60 of 220 id: 1567704
## Getting 61 of 220 id: 3259784
## Getting 62 of 220 id: 6655579
## Getting 63 of 220 id: 157221
## Getting 64 of 220 id: 4699239
## Getting 65 of 220 id: 2008208
## Getting 66 of 220 id: 6655590
## Getting 67 of 220 id: 1627111
## Getting 68 of 220 id: 277931
## Getting 69 of 220 id: 1743211
## Getting 70 of 220 id: 1370608
## Getting 71 of 220 id: 1431543
## Getting 72 of 220 id: 1273049
## Getting 73 of 220 id: 4699275
## Getting 74 of 220 id: 1722122
## Getting 75 of 220 id: 1496196
## Getting 76 of 220 id: 1728625
## Getting 77 of 220 id: 4699287
## Getting 78 of 220 id: 444103
## Getting 79 of 220 id: 6655592
## Getting 80 of 220 id: 563439
## Getting 81 of 220 id: 4699242
## Getting 82 of 220 id: 4781201
## Getting 83 of 220 id: 1781729
## Getting 84 of 220 id: 3169743
## Getting 85 of 220 id: 1476014
## Getting 86 of 220 id: 4699280
## Getting 87 of 220 id: 4699270
## Getting 88 of 220 id: 674039
## Getting 89 of 220 id: 4416376
## Getting 90 of 220 id: 4146704
## Getting 91 of 220 id: 1323725
## Getting 92 of 220 id: 1979888
## Getting 93 of 220 id: 1944824
## Getting 94 of 220 id: 436581
## Getting 95 of 220 id: 4699276
## Getting 96 of 220 id: 1078236
## Getting 97 of 220 id: 3169731
## Getting 98 of 220 id: 6655595
## Getting 99 of 220 id: 172270
## Getting 100 of 220 id: 4699274
## Getting 101 of 220 id: 1080433
## Getting 102 of 220 id: 4416273
## Getting 103 of 220 id: 1941077
## Getting 104 of 220 id: 2029154
## Getting 105 of 220 id: 3259787
## Getting 106 of 220 id: 1145167
## Getting 107 of 220 id: 649250
## Getting 108 of 220 id: 1289646
## Getting 109 of 220 id: 1609006
## Getting 110 of 220 id: 6655583
## Getting 111 of 220 id: 1830824
```

```
## Getting 112 of 220 id: 4656924
## Getting 113 of 220 id: 5077619
## Getting 114 of 220 id: 4699250
## Getting 115 of 220 id: 5077702
## Getting 116 of 220 id: 5077650
## Getting 117 of 220 id: 87059
## Getting 118 of 220 id: 3169803
## Getting 119 of 220 id: 5210744
## Getting 120 of 220 id: 6655587
## Getting 121 of 220 id: 3512668
## Getting 122 of 220 id: 3258698
## Getting 123 of 220 id: 2891692
## Getting 124 of 220 id: 3259795
## Getting 125 of 220 id: 1631560
## Getting 126 of 220 id: 4699279
## Getting 127 of 220 id: 5064837
## Getting 128 of 220 id: 1351088
## Getting 129 of 220 id: 1057320
## Getting 130 of 220 id: 3169823
## Getting 131 of 220 id: 1062435
## Getting 132 of 220 id: 3169812
## Getting 133 of 220 id: 5210826
## Getting 134 of 220 id: 3259797
## Getting 135 of 220 id: 684939
## Getting 136 of 220 id: 3259786
## Getting 137 of 220 id: 6655586
## Getting 138 of 220 id: 674050
## Getting 139 of 220 id: 3258695
## Getting 140 of 220 id: 1295771
## Getting 141 of 220 id: 416170
## Getting 142 of 220 id: 1175436
## Getting 143 of 220 id: 3258697
## Getting 144 of 220 id: 3169754
## Getting 145 of 220 id: 5210740
## Getting 146 of 220 id: 3169816
## Getting 147 of 220 id: 1742619
## Getting 148 of 220 id: 5658513
## Getting 149 of 220 id: 1318750
## Getting 150 of 220 id: 3258699
## Getting 151 of 220 id: 1784240
## Getting 152 of 220 id: 3259798
## Getting 153 of 220 id: 1781873
## Getting 154 of 220 id: 4699269
## Getting 155 of 220 id: 536128
## Getting 156 of 220 id: 1286708
## Getting 157 of 220 id: 958187
## Getting 158 of 220 id: 6655593
## Getting 159 of 220 id: 168549
## Getting 160 of 220 id: 3258700
## Getting 161 of 220 id: 3169724
## Getting 162 of 220 id: 5206079
## Getting 163 of 220 id: 1818950
## Getting 164 of 220 id: 3258694
## Getting 165 of 220 id: 5008073
## Getting 166 of 220 id: 3169708
## Getting 167 of 220 id: 3169833
```

```
## Getting 168 of 220 id: 5206057
## Getting 169 of 220 id: 5211063
## Getting 170 of 220 id: 2844119
## Getting 171 of 220 id: 4679008
## Getting 172 of 220 id: 1645904
## Getting 173 of 220 id: 3169808
## Getting 174 of 220 id: 444105
## Getting 175 of 220 id: 4699283
## Getting 176 of 220 id: 1726960
## Getting 177 of 220 id: 3258703
## Getting 178 of 220 id: 1805012
## Getting 179 of 220 id: 1979649
## Getting 180 of 220 id: 3258696
## Getting 181 of 220 id: 1323083
## Getting 182 of 220 id: 5064855
## Getting 183 of 220 id: 5064860
## Getting 184 of 220 id: 1338478
## Getting 185 of 220 id: 4431339
## Getting 186 of 220 id: 1296685
## Getting 187 of 220 id: 3258702
## Getting 188 of 220 id: 3169749
## Getting 189 of 220 id: 5210729
## Getting 190 of 220 id: 1219183
## Getting 191 of 220 id: 4222510
## Getting 192 of 220 id: 1082850
## Getting 193 of 220 id: 4699277
## Getting 194 of 220 id: 1299232
## Getting 195 of 220 id: 1743210
## Getting 196 of 220 id: 4699268
## Getting 197 of 220 id: 1408167
## Getting 198 of 220 id: 2891714
## Getting 199 of 220 id: 1481189
## Getting 200 of 220 id: 674415
## Getting 201 of 220 id: 5077775
## Getting 202 of 220 id: 3055453
## Getting 203 of 220 id: 4699241
## Getting 204 of 220 id: 5077754
## Getting 205 of 220 id: 5077809
## Getting 206 of 220 id: 2053555
## Getting 207 of 220 id: 1036148
## Getting 208 of 220 id: 3169744
## Getting 209 of 220 id: 5210722
## Getting 210 of 220 id: 6655580
## Getting 211 of 220 id: 1113446
## Getting 212 of 220 id: 1336126
## Getting 213 of 220 id: 3259785
## Getting 214 of 220 id: 157178
## Getting 215 of 220 id: 4699210
## Getting 216 of 220 id: 4699281
## Getting 217 of 220 id: 87062
## Getting 218 of 220 id: 4699284
## Getting 219 of 220 id: 3258701
## Getting 220 of 220 id: 6655594
```

```r
# collapse all lyrics lines into a single text
for (i in 1:length(songs_lyrics)){
  songs_lyrics[[i]] <- songs_lyrics[[i]]$line %>% paste(collapse = "\n")
}

songs_lyrics <- unlist(songs_lyrics)

songs <- data.frame(title = songs_titles, album = songs_albums, lyrics = songs_lyrics)
```

Now we have got a dataframe with data about each individual song. Let's look at the first element.

```r
songs %>% head(1)
```

```
##          title                     album
## 1 Abiura Di Me Le Dimensioni Del Mio Caos
##
lyrics
## 1 Se pensi che possa cambiare il mondo, ti sbagli alla grande\nÈ già tanto se mi cambio le
mutande\nVoglio solo darti un'emicrania lancinante\nFino a che non salti nel vuoto come uno s
tuntman\nPensavi che sparassi palle? Bravo\nIo sono il drago di Puzzle Bobble\nCome Crash mi
piace rompere le scatole\nMa rischio le mazzate che nemmeno Double Dragon\nSarà per questo ch
e c'è sempre qualche blogger\nChe mi investirebbe come a Frogger\nGli bucherò le gomme e bye
bye\nAl limite può farmi una Sega Mega Drive\nNon mi vedrai salvare un solo lemming\nNé stare
qui a fare la muffa come Fleming\nNon darmi Grammy né premi da star\nMa giocati il tuo penny
e premi start\nIo voglio passare ad un livello successivo\nVoglio dare vita a ciò che scrivo
\nSono paranoico ed ossessivo fino all'abiura di me\nVado ad un livello successivo\nDove dare
vita a ciò che scrivo\nSono paranoico ed ossessivo fino all'abiura di me\nIo faccio politica
pure quando respiro\nMica scrivo musica giocando a Guitar Hero\nQuesti argomenti mi fanno sen
tire vivo\nIn mezzo a troppi zombi da Resident Evil\nMacché divo, mi chiudo a riccio più di S
onic\nFino a che non perdo l'armatura come a Ghost 'n Goblins\nMi metto a nudo io\nNon mi nas
condo come Snake in Metal Gear Solid\nHo 500 Amighe, intesi?\nFaccio canzoni, mica catechesi
\nPrendo soldi con il pugno alzato come Super Mario\nMa non li ho mai spesi per farmi le righ
e come a Tetris\nLa scena rap è controversa\nSfuggo con un salto da Prince of Persia\nIo non
gioco le Olimpiadi Konami\nSe stacco le mani l'agitazione mi resta\nIo voglio passare ad un l
ivello successivo\nVoglio dare vita a ciò che scrivo\nSono paranoico ed ossessivo fino all'ab
iura di me\nVado ad un livello successivo\nDove dare vita a ciò che scrivo\nSono paranoico ed
ossessivo fino all'abiura di me\nAbiura di me, abiura di me\nAbiura di me, di me, di me\nAbiu
ra di me, abiura di me\nAbiura di me, di me\nIo non vengo dalla strada, sono troppo nerd\nNon
sposo quella causa, ho troppi flirt\nVivo tra gente che col Red Alert\nPassa la vita su cubi
come Q*bert\nHo visto pazzi rievocare vecchi fantasmi\nCome Pac-Man e Dan Aykroyd\nHo visto d
uri che risolvono problemi alzando muri\nChe abbatto come ho fatto in Arkanoid\nNemmeno Freud
saprebbe spiegarmi\nPerché la notte sogno di aumentare le armi\nPerché la Terra mi pare talme
nte maligna\nChe in confronto Silent Hill assomiglia a Topolinia\nIo devo scrivere perché se
no sclero\nNon mi interessa che tu condivida il mio pensiero\nNon cammino sulle nubi come a W
onder Boy\nMi credi il messia? Sono problemi tuoi\nIo voglio passare ad un livello successivo
\nVoglio dare vita a ciò che scrivo\nSono paranoico ed ossessivo fino all'abiura di me\nVado
ad un livello successivo\nDove dare vita a ciò che scrivo\nSono paranoico ed ossessivo fino a
ll'abiura di me\nAbiura di me, abiura di me\nAbiura di me, di me, di me\nAbiura di me, abiura
di me\nAbiura di me, di me\nStai calmo\nChe punteggio basso\nLa corte condanna il signor Rezz
a Capa ad anni dieci di lavori socialmente utili come spalatore di cacca di elefanti circensi
\nLa seduta è sciolta, viva lo spazioporto!
```

Songs are ordered alphabetically by the title, and the first one appears to be *Abiura Di Me*.

Let’s do some cleaning: we are going to ignore alternative versions of the same song, such as live versions, radio edits and remixes. We are also keeping only the main albums, ignoring specials or compilations.

```
albums <- c("?! (Caparezza ?!)", "Verità Supposte", "Habemus Capa",
            "Le Dimensioni Del Mio Caos", "Il Sogno Eretico", "Museica",
            "Prisoner 709", "Exuvia")
songs <- songs %>%
  filter(!grepl("Live|Radio Edit|Radio Version|Remix|RMX", title) & album %in% albums)
```

Finally, we order songs chronologically by the album, then add an id for each song.

```
songs <- songs %>% arrange(match(album, albums), title)
doc_ids <- vector()
for(i in 1:nrow(songs)){
  id <- paste("doc", toString(i), sep = "")
  doc_ids <- doc_ids %>% append(id)
}
songs <- songs %>% mutate(doc_id = doc_ids, .before = 1)
```

Let’s check the number of songs for each album.

```
table(songs$album) %>% as.data.frame() %>% arrange(-Freq)
```

```
##                          Var1 Freq
## 1                      Exuvia   19
## 2                     Museica   19
## 3                Habemus Capa   18
## 4            Il Sogno Eretico   16
## 5                Prisoner 709   16
## 6             Verità Supposte   15
## 7           ?! (Caparezza ?!)   14
## 8 Le Dimensioni Del Mio Caos   14
```

The frequencies are correct, as every Caparezza fan will recognize.

# Text pre-processing

Now we start with some text pre-processing: let’s create a corpus from the songs’ lyrics.

```
caparezza_corpus <- corpus(songs$lyrics, docnames = songs$id)
summary(caparezza_corpus)
```

```
## Corpus consisting of 131 documents, showing 100 documents:
##
##      Text Types Tokens Sentences
##     text1   298    551         3
##     text2   322    744         8
##     text3   321    643         7
##     text4   251    585         1
##     text5   279    573         7
##     text6    49     57         1
##     text7   237    484         1
##     text8   270    584        10
##     text9   314    703         3
##    text10   206    486         1
##    text11   205    413         1
##    text12   232    709         4
##    text13   195    499        13
##    text14   262    607         2
##    text15   244    440         2
##    text16   181    392         7
##    text17   184    437         2
##    text18   258    500         4
##    text19   235    585         2
##    text20   233    566         4
##    text21   261    416         3
##    text22   273    609        14
##    text23   291    646         5
##    text24   286    556         2
##    text25   281    475         5
##    text26   323    645         6
##    text27   255    543         3
##    text28   220    404         3
##    text29   234    569         4
##    text30   240    525         2
##    text31   258    648        31
##    text32   232    633        19
##    text33   403    713        15
##    text34   270    537         1
##    text35   278    521        13
##    text36   302    593         2
##    text37   305    568         4
##    text38   300    518         3
##    text39    12     13         1
##    text40   346    773        25
##    text41     4    107         1
##    text42   265    628        12
##    text43    10     17         6
##    text44   308    528         4
##    text45   268    593        13
##    text46   265    515         4
##    text47   246    431         6
##    text48   299    582         4
##    text49   174    319         2
##    text50   277    495         7
##    text51   294    597         1
##    text52   267    617        19
```

```
##    text53   208    413         17
##    text54   275    553          7
##    text55   288    576         12
##    text56   325    615         41
##    text57   281    625          3
##    text58   319    598         19
##    text59   217    415          7
##    text60   160    538          4
##    text61   283    567          4
##    text62   289    714          8
##    text63   242    586         20
##    text64   228    500          6
##    text65   253    572          6
##    text66   286    703          5
##    text67   279    546         32
##    text68   207    470         16
##    text69   260    615         10
##    text70   281    607         20
##    text71   346    827          3
##    text72   247    507         13
##    text73   293    539         15
##    text74   217    574          1
##    text75   232    461          4
##    text76   169    322         12
##    text77    80    124          7
##    text78   318    667         19
##    text79   280    643          7
##    text80   167    384          1
##    text81   183    291          5
##    text82   177    250          1
##    text83   271    509          1
##    text84   256    488         16
##    text85   213    433          3
##    text86   293    465          2
##    text87   194    498          3
##    text88   186    385          1
##    text89   243    578         24
##    text90   228    405          2
##    text91   225    572          1
##    text92   196    436         11
##    text93   261    503          8
##    text94   236    414         18
##    text95   252    625          6
##    text96   312    732          5
##    text97   227    409          2
##    text98   263    607          2
##    text99   292    508          3
##   text100   272    618          2
```

There is a total of 129 documents and for each one the total number of tokens is displayed. A *token* is a single occurrence of a word in the document.

```
cat(as.character(caparezza_corpus[1]))
```

```
## Non rappresento che me stesso perché questo sono
## Se sbaglio mi perdono
## Prima di essere MC sii uomo mi ripeto
## Fa' mille passi indietro e il risultato
## È che non mi sento per niente arrivato
## Anzi sto bene anche a cibarmi degli avanzi dei padroni sazi
## E mi piglio spazi se me li concedono
## Sennò me li lascio fottere
## Detesto combattere, che vuoi farci? È carattere
## Sbattere testa contro le porte è il mio forte
## Sono il gallo da spennare per chi bara alle carte
## Giullare di corte messo a morte e poi salvato da una chance
## Lascerei la musica, ma 'sta stronza mi fa le avances
## E non resisto, mi do in pasto alla lingua che mastico
## Investo in testi che vesto di stracci e mi riduco al lastrico
## Nella testa un mistico richiamo, poema indiano
## Che mi prende per mano e mi dice: "Andiamo!"
## Se non rispondono al tuo appello, cammina solo, cammina solo
## Se non rispondono al tuo appello, cammina solo, cammina solo
## Detesto l'odio ma l'ho visto venir fuori
## Dagli occhi di alcuni interlocutori
## Hanno motivi loro e i loro sguardi sono come lastre di ghiaccio
## Si scioglieranno a poco a poco al fuoco di ciò che faccio
## Se il rancore resta onestamente non mi resta niente da fare
## Che alzare i tacchi e andare, menare via
## Cullarmi nel tepore di ogni mano che ha stretto la mia
## Avere Dio come terapia
## Sarà la miopia ma faccio fatica a inquadrare la retta via
## Voglio te per compagnia
## Portami in balia della gente, dove c'è amore
## Lì sarò presente anch'io
## Ti cedo il posto mio
## Non è per vincere che vivo ma per ardere
## Perciò se dovrò perdere lasciatemi perdere e avrò perso
## Cosciente che non sono né peggiore né migliore di nessuno
## Finchè sarò diverso
## Se non rispondono al tuo appello, cammina solo, cammina solo
## Se non rispondono al tuo appello, cammina solo, cammina solo
## Se mi ritrovo sull'incudine, sotto un martello di solitudine
## Colpo su colpo come un polpo sullo scoglio muoio, ma ci farò l'abitudine
## Se non lo sai cominciai per scherzo
## Come un bimbo immobile nell'automobile con le mani sullo sterzo
## Verso nuovi orizzonti, sopra e sotto i ponti
## Davanti a piatti pronti, pagato con assegni fatti di saldi e sconti
## Tra re, regine e fanti cercai clemenza
## Mò non vado in vacanza prima di aver lasciato una testimonianza di ciò che sono
## Coi miei tanti nomi, le contraddizioni
## Appartengo ad una strana scena, quella degli esseri umani
## Credo ai meriti che conquisto, credo in Cristo perché l'ho visto
## Credo al rischio dell'incomprensione, credo nelle persone
## Nella consolazione, nella mia devozione, in ogni azione pacifica
## Detesto l'astio che ramifica, la cassa che lo amplifica
## Canto il mio Magnificat come un pazzo a mare e monti
## Ignoranti e colti, sperando che qualcuno ascolti
## Se non rispondono al tuo appello, cammina solo, cammina solo
```

```
## Se non rispondono al tuo appello, cammina solo, cammina solo
## Se non rispondono al tuo appello, cammina solo, cammina solo
## Se non rispondono al tuo appello, cammina solo, cammina solo
```

The first document of the corpus is the song “Cammina Solo” from *?!*, since we have ordered songs by album and then by title.

One of the main steps of lexical analysis is the removal of punctuation marks, numbers and symbols which are not useful towards the text interpretation.

The *quanteda* package offers some useful tools for these operations.

```
corpus_tokens <- caparezza_corpus %>%
  quanteda::tokens(remove_punct = TRUE, remove_numbers = TRUE, remove_symbols = TRUE) %>%
  tokens_tolower()
```

Additionally, we would like to lemmatise: that is, to consider the *lemma* from which a word originates, instead of the world itself. For example, the infinitive forms of the verbs or the standard singular masculine adjective form are sufficient to express a concept, so we can safely use those instead of their derivatives.

```
txt <- sapply(corpus_tokens, FUN=function(x) paste(x, collapse = "\n"))
udpipe_download_model(language = "italian-isdt", model_dir = "resources/")
```

```
## Downloading udpipe model from https://raw.githubusercontent.com/jwijffels/udpipe.models.u
d.2.5/master/inst/udpipe-ud-2.5-191206/italian-isdt-ud-2.5-191206.udpipe to resources//italia
n-isdt-ud-2.5-191206.udpipe
```

```
##  - This model has been trained on version 2.5 of data from https://universaldependencies.o
rg
```

```
##  - The model is distributed under the CC-BY-SA-NC license: https://creativecommons.org/lic
enses/by-nc-sa/4.0
```

```
##  - Visit https://github.com/jwijffels/udpipe.models.ud.2.5 for model license details.
```

```
##  - For a list of all models and their licenses (most models you can download with this pac
kage have either a CC-BY-SA or a CC-BY-SA-NC license) read the documentation at ?udpipe_downl
oad_model. For building your own models: visit the documentation by typing vignette('udpipe-t
rain', package = 'udpipe')
```

```
## Downloading finished, model stored at 'resources//italian-isdt-ud-2.5-191206.udpipe'
```

```
##       language                                           file_model
## 1 italian-isdt resources//italian-isdt-ud-2.5-191206.udpipe
##
url
## 1 https://raw.githubusercontent.com/jwijffels/udpipe.models.ud.2.5/master/inst/udpipe-ud-
2.5-191206/italian-isdt-ud-2.5-191206.udpipe
##   download_failed download_message
## 1           FALSE               OK
```

```
lang_model <- udpipe_load_model(file = "resources/italian-isdt-ud-2.5-191206.udpipe")
outL <- udpipe_annotate(lang_model, x = txt, tokenizer = "vertical", trace = TRUE) %>%
  as.data.frame()
```

```
## 2023-05-31 09:05:05 Annotating text fragment 1/131
## 2023-05-31 09:05:06 Annotating text fragment 2/131
## 2023-05-31 09:05:07 Annotating text fragment 3/131
## 2023-05-31 09:05:08 Annotating text fragment 4/131
## 2023-05-31 09:05:09 Annotating text fragment 5/131
## 2023-05-31 09:05:09 Annotating text fragment 6/131
## 2023-05-31 09:05:09 Annotating text fragment 7/131
## 2023-05-31 09:05:10 Annotating text fragment 8/131
## 2023-05-31 09:05:11 Annotating text fragment 9/131
## 2023-05-31 09:05:12 Annotating text fragment 10/131
## 2023-05-31 09:05:12 Annotating text fragment 11/131
## 2023-05-31 09:05:13 Annotating text fragment 12/131
## 2023-05-31 09:05:14 Annotating text fragment 13/131
## 2023-05-31 09:05:14 Annotating text fragment 14/131
## 2023-05-31 09:05:15 Annotating text fragment 15/131
## 2023-05-31 09:05:16 Annotating text fragment 16/131
## 2023-05-31 09:05:16 Annotating text fragment 17/131
## 2023-05-31 09:05:17 Annotating text fragment 18/131
## 2023-05-31 09:05:17 Annotating text fragment 19/131
## 2023-05-31 09:05:18 Annotating text fragment 20/131
## 2023-05-31 09:05:19 Annotating text fragment 21/131
## 2023-05-31 09:05:19 Annotating text fragment 22/131
## 2023-05-31 09:05:20 Annotating text fragment 23/131
## 2023-05-31 09:05:21 Annotating text fragment 24/131
## 2023-05-31 09:05:21 Annotating text fragment 25/131
## 2023-05-31 09:05:22 Annotating text fragment 26/131
## 2023-05-31 09:05:23 Annotating text fragment 27/131
## 2023-05-31 09:05:24 Annotating text fragment 28/131
## 2023-05-31 09:05:24 Annotating text fragment 29/131
## 2023-05-31 09:05:25 Annotating text fragment 30/131
## 2023-05-31 09:05:25 Annotating text fragment 31/131
## 2023-05-31 09:05:26 Annotating text fragment 32/131
## 2023-05-31 09:05:27 Annotating text fragment 33/131
## 2023-05-31 09:05:28 Annotating text fragment 34/131
## 2023-05-31 09:05:29 Annotating text fragment 35/131
## 2023-05-31 09:05:30 Annotating text fragment 36/131
## 2023-05-31 09:05:31 Annotating text fragment 37/131
## 2023-05-31 09:05:31 Annotating text fragment 38/131
## 2023-05-31 09:05:32 Annotating text fragment 39/131
## 2023-05-31 09:05:32 Annotating text fragment 40/131
## 2023-05-31 09:05:33 Annotating text fragment 41/131
## 2023-05-31 09:05:33 Annotating text fragment 42/131
## 2023-05-31 09:05:34 Annotating text fragment 43/131
## 2023-05-31 09:05:34 Annotating text fragment 44/131
## 2023-05-31 09:05:35 Annotating text fragment 45/131
## 2023-05-31 09:05:35 Annotating text fragment 46/131
## 2023-05-31 09:05:36 Annotating text fragment 47/131
## 2023-05-31 09:05:37 Annotating text fragment 48/131
## 2023-05-31 09:05:37 Annotating text fragment 49/131
## 2023-05-31 09:05:38 Annotating text fragment 50/131
## 2023-05-31 09:05:38 Annotating text fragment 51/131
## 2023-05-31 09:05:39 Annotating text fragment 52/131
## 2023-05-31 09:05:40 Annotating text fragment 53/131
## 2023-05-31 09:05:40 Annotating text fragment 54/131
## 2023-05-31 09:05:41 Annotating text fragment 55/131
```

```
## 2023-05-31 09:05:42 Annotating text fragment 56/131
## 2023-05-31 09:05:42 Annotating text fragment 57/131
## 2023-05-31 09:05:43 Annotating text fragment 58/131
## 2023-05-31 09:05:44 Annotating text fragment 59/131
## 2023-05-31 09:05:45 Annotating text fragment 60/131
## 2023-05-31 09:05:45 Annotating text fragment 61/131
## 2023-05-31 09:05:46 Annotating text fragment 62/131
## 2023-05-31 09:05:47 Annotating text fragment 63/131
## 2023-05-31 09:05:48 Annotating text fragment 64/131
## 2023-05-31 09:05:48 Annotating text fragment 65/131
## 2023-05-31 09:05:49 Annotating text fragment 66/131
## 2023-05-31 09:05:50 Annotating text fragment 67/131
## 2023-05-31 09:05:50 Annotating text fragment 68/131
## 2023-05-31 09:05:51 Annotating text fragment 69/131
## 2023-05-31 09:05:52 Annotating text fragment 70/131
## 2023-05-31 09:05:52 Annotating text fragment 71/131
## 2023-05-31 09:05:53 Annotating text fragment 72/131
## 2023-05-31 09:05:54 Annotating text fragment 73/131
## 2023-05-31 09:05:55 Annotating text fragment 74/131
## 2023-05-31 09:05:55 Annotating text fragment 75/131
## 2023-05-31 09:05:56 Annotating text fragment 76/131
## 2023-05-31 09:05:56 Annotating text fragment 77/131
## 2023-05-31 09:05:56 Annotating text fragment 78/131
## 2023-05-31 09:05:57 Annotating text fragment 79/131
## 2023-05-31 09:05:58 Annotating text fragment 80/131
## 2023-05-31 09:05:58 Annotating text fragment 81/131
## 2023-05-31 09:05:59 Annotating text fragment 82/131
## 2023-05-31 09:05:59 Annotating text fragment 83/131
## 2023-05-31 09:06:00 Annotating text fragment 84/131
## 2023-05-31 09:06:00 Annotating text fragment 85/131
## 2023-05-31 09:06:01 Annotating text fragment 86/131
## 2023-05-31 09:06:01 Annotating text fragment 87/131
## 2023-05-31 09:06:02 Annotating text fragment 88/131
## 2023-05-31 09:06:02 Annotating text fragment 89/131
## 2023-05-31 09:06:03 Annotating text fragment 90/131
## 2023-05-31 09:06:03 Annotating text fragment 91/131
## 2023-05-31 09:06:04 Annotating text fragment 92/131
## 2023-05-31 09:06:05 Annotating text fragment 93/131
## 2023-05-31 09:06:05 Annotating text fragment 94/131
## 2023-05-31 09:06:06 Annotating text fragment 95/131
## 2023-05-31 09:06:06 Annotating text fragment 96/131
## 2023-05-31 09:06:07 Annotating text fragment 97/131
## 2023-05-31 09:06:08 Annotating text fragment 98/131
## 2023-05-31 09:06:09 Annotating text fragment 99/131
## 2023-05-31 09:06:09 Annotating text fragment 100/131
## 2023-05-31 09:06:10 Annotating text fragment 101/131
## 2023-05-31 09:06:11 Annotating text fragment 102/131
## 2023-05-31 09:06:12 Annotating text fragment 103/131
## 2023-05-31 09:06:12 Annotating text fragment 104/131
## 2023-05-31 09:06:13 Annotating text fragment 105/131
## 2023-05-31 09:06:14 Annotating text fragment 106/131
## 2023-05-31 09:06:14 Annotating text fragment 107/131
## 2023-05-31 09:06:14 Annotating text fragment 108/131
## 2023-05-31 09:06:15 Annotating text fragment 109/131
## 2023-05-31 09:06:15 Annotating text fragment 110/131
## 2023-05-31 09:06:16 Annotating text fragment 111/131
```

```
## 2023-05-31 09:06:17 Annotating text fragment 112/131
## 2023-05-31 09:06:17 Annotating text fragment 113/131
## 2023-05-31 09:06:18 Annotating text fragment 114/131
## 2023-05-31 09:06:19 Annotating text fragment 115/131
## 2023-05-31 09:06:19 Annotating text fragment 116/131
## 2023-05-31 09:06:20 Annotating text fragment 117/131
## 2023-05-31 09:06:20 Annotating text fragment 118/131
## 2023-05-31 09:06:21 Annotating text fragment 119/131
## 2023-05-31 09:06:22 Annotating text fragment 120/131
## 2023-05-31 09:06:22 Annotating text fragment 121/131
## 2023-05-31 09:06:23 Annotating text fragment 122/131
## 2023-05-31 09:06:23 Annotating text fragment 123/131
## 2023-05-31 09:06:24 Annotating text fragment 124/131
## 2023-05-31 09:06:24 Annotating text fragment 125/131
## 2023-05-31 09:06:25 Annotating text fragment 126/131
## 2023-05-31 09:06:25 Annotating text fragment 127/131
## 2023-05-31 09:06:26 Annotating text fragment 128/131
## 2023-05-31 09:06:26 Annotating text fragment 129/131
## 2023-05-31 09:06:26 Annotating text fragment 130/131
## 2023-05-31 09:06:26 Annotating text fragment 131/131
```

```
it_stopwords <- readLines("https://raw.githubusercontent.com/stopwords-iso/stopwords-it/maste
r/stopwords-it.txt")
```

```
## Warning in
## readLines("https://raw.githubusercontent.com/stopwords-iso/stopwords-it/master/stopwords-i
t.txt"):
## riga finale incompleta in
## 'https://raw.githubusercontent.com/stopwords-iso/stopwords-it/master/stopwords-it.txt'
```

```
outL <- outL %>% filter(!(token %in% it_stopwords) & !(lemma %in% it_stopwords))
```

We have made use of the *UDPipe* library annotation function.

In the process, we have also removed *stopwords*, words that do not bring any meaningful addition to our texts, but are only necessary to connect other words and respect syntactic rules.

Let's have a look at random sample of five elements of the output.

```
outL %>% select(doc_id, token, lemma, upos) %>% sample_n(5)
```

```
##   doc_id     token     lemma upos
## 1  doc73   tortura   tortura NOUN
## 2 doc110       via       via  ADV
## 3 doc110       der       der  ADP
## 4  doc50 discovery discovery NOUN
## 5  doc54    urlava    urlare VERB
```

To further enhance our analysis, let's focus only on nouns, proper nouns, adjectives and verbs.

```
outL_reduced <- outL %>% filter(upos %in% c("NOUN", "PROPN", "ADJ", "VERB"))
```

Now we create a new corpus with the lemmatized lyrics.

```
# fct_inorder preserves original order of the column
lemmatized_lyrics <- outL_reduced %>% group_by(doc_id = fct_inorder(doc_id)) %>%
  summarise(lemmatized = paste(lemma, collapse = " "))
songs <- songs %>% right_join(lemmatized_lyrics, by = "doc_id")

caparezza_corpus <- songs$lemmatized %>% corpus(docnames = songs$id)
```

One final step for text-processing consists in handling collocations, or multi-word units(MWUs). A collocation is a set of two or more words which are closely related and are often used together to express a single concept.

They can be identified through statistical, objective methods, like in this case.

```
collocations <- caparezza_corpus %>% tokens() %>% textstat_collocations %>%
  arrange(-count) %>% head(10)
```

Looking at the collocations found by the function, we decide to take action on two of them, which seem to be clearly meant to be used together.

```
DTM <- caparezza_corpus %>% tokens() %>% tokens_compound(collocations[c(4,6),]) %>%
  tokens_remove("") %>% dfm()
```

# Lexical Analysis

In the last step, we have created an object called DTM. This is the document-term matrix of our corpus: it's a matrix which contains documents on its rows and terms on its columns. Every element of the matrix describes the number of occurrencies of the corresponding term inside the corresponding document.

```
DTM
```

```
## Document-feature matrix of: 129 documents, 8,679 features (98.55% sparse) and 0 docvars.
##        features
## docs    rappresentare perdere mc ripetere passo risultato sentire arrivato arma
##   text1             1       4  1        1     1         1       1        1    2
##   text2             0       0  0        0     0         0       0        0    0
##   text3             0       0  0        0     0         0       0        0    0
##   text4             0       0  0        0     0         0       1        0    0
##   text5             0       0  0        0     0         0       0        0    0
##   text6             0       0  0        0     0         0       0        0    0
##        features
## docs    |
##   text1 2
##   text2 0
##   text3 0
##   text4 0
##   text5 0
##   text6 0
## [ reached max_ndoc ... 123 more documents, reached max_nfeat ... 8,669 more features ]
```

```
DTM %>% dim()
```

```
## [1]  129 8679
```

We have got a total of 129 documents and 8679 unique terms in our corpus.

Let's build a frequencies table for our terms and look at the most recurring ones.

```
words <- colnames(DTM)
freqs <- colSums(DTM)
wordlist <- data.frame(words, freqs)
wordlist %>% arrange(-freqs) %>% head()
```

```
##           words freqs
## sapere   sapere   152
## vedere   vedere   151
## andare   andare   125
## mano       mano   116
## vivo       vivo   108
## parlare parlare   102
```

And now let's look at some useful quantities.

```
corpus_size <- sum(wordlist$freqs)
corpus_size
```

```
## [1] 22985
```

```
vocabulary_size <- nrow(wordlist)
vocabulary_size
```

```
## [1] 8679
```

```
words_occurrencies <- wordlist %>% group_by(freqs) %>% summarise(vK = n()) %>% arrange(-vK)
words_occurrencies
```

```
## # A tibble: 67 × 2
##    freqs    vK
##    <dbl> <int>
##  1     1  5708
##  2     2  1172
##  3     3   541
##  4     4   293
##  5     5   206
##  6     6   138
##  7     7    90
##  8     8    89
##  9     9    62
## 10    10    44
## # ℹ 57 more rows
```

```
lexicon_width <- vocabulary_size/corpus_size
lexicon_width
```

```
## [1] 0.3775941
```

```
language_refinement <- words_occurrencies$vK[1] / colSums(words_occurrencies)[2]
language_refinement
```

```
##        vK
## 0.6576795
```

Words that appear only once in the entire corpus are known as *hapaxes*. *Lexicon width* is the percentage of unique words in the total number of words used in the corpus. *Language refinement* is the percentage of hapaxes in the total number of words used in the corpus.

# Data visualization

In order to visualize the most frequent terms, we can use a *word cloud*, which portrays the terms in the corpus with different sizes depending on their relative frequency.

```
par(mar=c(1,1,0.5,1))
wordcloud(words = wordlist$words, freq = wordlist$freqs, scale = c(3.5, 0.35),
          max.words = 50, random.order = F,
          colors = RColorBrewer::brewer.pal(name = "Dark2", n = 4))
text(0.5, 1, "wordcloud with TF ponderation", font = 2)
```


It could be interesting to know in which songs a given word appears, especially if it seems an odd one. We can use this function.

```
count <- 1
for(i in songs$lyrics){
  if(grepl("mamma", i)){
    print(songs$title[[count]])
  }
  count <- count + 1
}
```

```
## [1] "Chi c*zzo me lo"
## [1] "Mammamiamamma"
## [1] "Limiti"
## [1] "Nel paese dei balordi"
## [1] "Felici ma trimoni"
## [1] "Sono troppo stitico"
## [1] "Io Diventerò Qualcuno"
## [1] "La Fine Di Gaia"
## [1] "Messa In Moto"
## [1] "Fugadà"
```

# TF-IDF ponderation

It is often necessary to weight terms in the corpus on the basis of their relative importance. For example, a term which occurs frequently in a document has a strong relevance for that document, but a term which occurs frequently in many different documents is less informative for any single document and should be treated as less relevant.

To account for this situation, we are going to use the term frequency-inverse document frequency (TF-IDF) weighting.

```
tf_idf <- dfm_tfidf(DTM)
freqs_tf_idf <- colSums(tf_idf)
words_tf_idf <- colnames(tf_idf)
wordlist_tf_idf <- data.frame(words = words_tf_idf, freqs = freqs_tf_idf)
wordlist_tf_idf %>% arrange(-freqs) %>% head(10)
```

```
##                             words    freqs
## mamma                       mamma 86.63831
## zicare                     zicare 86.53418
## fuga                         fuga 84.94036
## politico                 politico 76.63069
## van_gogh                 van_gogh 71.76005
## don't_recogniza don't_recogniza 68.76327
## tasca                       tasca 67.75775
## meghino                   meghino 67.53887
## ye-ye                       ye-ye 67.53887
## vivo                         vivo 65.38749
```

These are the most common terms after the applied weighting.

Let’s have a look at the new word cloud.

```
par(mar=c(1,1,0.5,1))
wordcloud(words = wordlist_tf_idf$words, freq = wordlist_tf_idf$freqs,
          scale = c(3.5, 0.35), max.words = 50, random.order = F,
          colors = RColorBrewer::brewer.pal(name = "Dark2", n = 4))
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : secessionista could not be fit on page. It will not be
## plotted.
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : sfogare could not be fit on page. It will not be
## plotted.
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : storia could not be fit on page. It will not be
## plotted.
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : campione could not be fit on page. It will not be
## plotted.
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : sapere could not be fit on page. It will not be
## plotted.
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : catalesso could not be fit on page. It will not be
## plotted.
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : caminare could not be fit on page. It will not be
## plotted.
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : problema could not be fit on page. It will not be
## plotted.
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : culpa could not be fit on page. It will not be
## plotted.
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : paradosso could not be fit on page. It will not be
## plotted.
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : metà could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = wordlist_tf_idf$words, freq =
## wordlist_tf_idf$freqs, : chiedere could not be fit on page. It will not be
## plotted.
```

```
text(0.5, 1, "wordcloud with TF-IDF ponderation", font = 2)
```

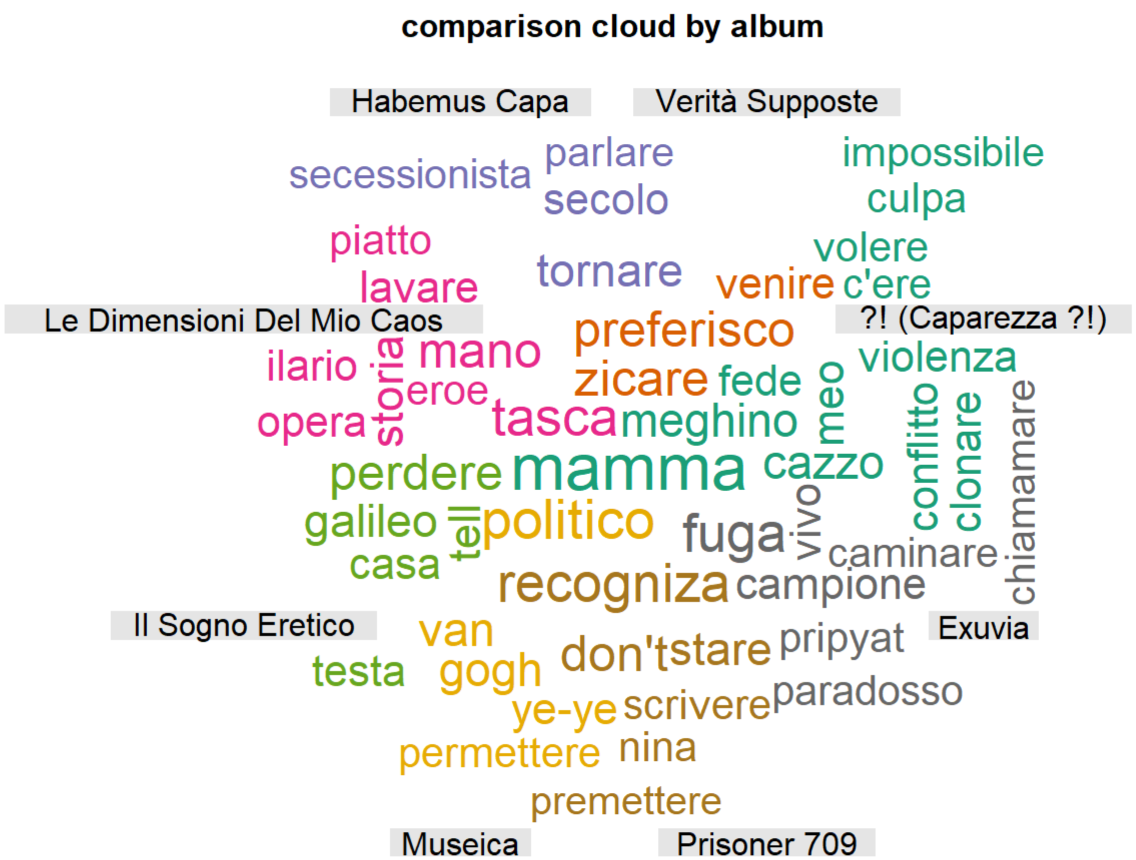

# Group by albums

Let's redo some of our previous analysis, this time considering the entire albums as documents, rather than individual songs.

```
lemmatized_lyrics_by_album <- songs %>% group_by(album) %>%
  summarise(lemmatized = paste(lemmatized, collapse = " ")) %>% arrange(match(album, albums))
corpus_album <- lemmatized_lyrics_by_album$lemmatized %>% corpus()

DTM_album <- corpus_album %>% tokens() %>% dfm()
DTM_album
```

```
## Document-feature matrix of: 8 documents, 8,677 features (80.97% sparse) and 0 docvars.
##        features
## docs    rappresentare perdere mc ripetere passo risultato sentire arrivato arma
##   text1             1       5  1        2     1         1       7        1    8
##   text2             0       2  0        0     4         0       8        0    4
##   text3             0       4  0        0     1         1      18        0    1
##   text4             0       2  0        1     3         1       6        0    3
##   text5             0      42  0        1     2         0       7        0    2
##   text6             1       1  0        1     6         0       5        0    7
##        features
## docs    |
##   text1 8
##   text2 5
##   text3 1
##   text4 2
##   text5 2
##   text6 7
## [ reached max_ndoc ... 2 more documents, reached max_nfeat ... 8,667 more features ]
```

```
docnames(DTM_album) <- lemmatized_lyrics_by_album$album
wordlist_album <- DTM_album %>% as.matrix() %>% t()
par(mar=c(0,0,0,0))
comparison.cloud(wordlist_album, scale = c(2, 1), max.words = 50, random.order = F,
                 title.size = 1, colors = RColorBrewer::brewer.pal(name = "Dark2", n = 8))
text(0.5, 1, "comparison cloud by album", font = 2)
```




We can recognize some iconic songs from each album by looking at the words: *conflitto* from *Il Conflitto* (*?!*), *secessionista* from *Inno Verdano* (*Habemus Capa*), *campione* from *Campione dei Novanta* (*Exuvia*), and more.

# Co-occurrence analysis

Next, we are going to analyze the co-occurrence of words in order to find terms which tend to appear in the same documents.

```
binDTM <- DTM %>% dfm_trim(min_docfreq = 10) %>% dfm_weight("boolean")
coocCounts <- t(binDTM) %*% binDTM
as.matrix(coocCounts[100:102, 100:102])
```

```
##         libro resto leggere
## libro      15     2       4
## resto       2    12       1
## leggere     4     1      14
```

For example, the words *libro* and *leggere* appear together in four documents. The diagonal elements of the matrix are the total occurrencies of the word.

Let's calculate some co-occurrence measurements for the words *lavoro* and then *leggere*: Mutual Information, Dice, and Log-Likelihood.

```
coocTerm <- "lavoro"
k <- nrow(binDTM)
ki <- sum(binDTM[, coocTerm])
kj <- colSums(binDTM)
names(kj) <- colnames(binDTM)
kij <- coocCounts[coocTerm, ]

mutualInformationSig <- log(k * kij / (ki * kj))
mutualInformationSig <- mutualInformationSig[order(mutualInformationSig, decreasing = TRUE)]

dicesig <- 2 * kij / (ki + kj)
dicesig <- dicesig[order(dicesig, decreasing=TRUE)]

logsig <- 2 * ((k * log(k)) - (ki * log(ki)) - (kj * log(kj)) + (kij * log(kij))
               + (k - ki - kj + kij) * log(k - ki - kj + kij)
               + (ki - kij) * log(ki - kij) + (kj - kij) * log(kj - kij)
               - (k - ki) * log(k - ki) - (k - kj) * log(k - kj))
logsig <- logsig[order(logsig, decreasing=T)]

resultOverView <- data.frame(
  names(sort(kij, decreasing=T)[1:10]), sort(kij, decreasing=T)[1:10],
  names(mutualInformationSig[1:10]), mutualInformationSig[1:10],
  names(dicesig[1:10]), dicesig[1:10],
  names(logsig[1:10]), logsig[1:10],
  row.names = NULL)
colnames(resultOverView) <- c("Freq-terms", "Freq", "MI-terms", "MI", "Dice-Terms", "Dice",
"LL-Terms", "LL")
print(resultOverView)
```

```
##    Freq-terms Freq MI-terms        MI Dice-Terms      Dice LL-Terms        LL
## 1      lavoro   12   lavoro 2.3749058     lavoro 1.0000000 politico 12.737294
## 2     mettere    7 politico 1.6817586   politico 0.4545455  giocare  8.124581
## 3      sapere    7  servire 1.2762935    giocare 0.3703704  servire  6.205002
## 4        mano    6  giocare 1.2762935    servire 0.3333333 arrivare  5.602518
## 5     parlare    6   facile 1.1709330    restare 0.3125000  restare  5.375092
## 6    arrivare    6  vecchio 1.0756228   scrivere 0.3125000 scrivere  5.375092
## 7     passare    6    donna 1.0531499   arrivare 0.3076923    donna  4.514591
## 8      vedere    5  restare 0.9886114      donna 0.2962963    gioco  4.063507
## 9     prendere   5    gioco 0.9886114      gioco 0.2857143    punto  4.063507
## 10     andare    5    punto 0.9886114      punto 0.2857143   facile  3.852185
```

```
coocTerm <- "leggere"
k <- nrow(binDTM)
ki <- sum(binDTM[, coocTerm])
kj <- colSums(binDTM)
names(kj) <- colnames(binDTM)
kij <- coocCounts[coocTerm, ]

mutualInformationSig <- log(k * kij / (ki * kj))
mutualInformationSig <- mutualInformationSig[order(mutualInformationSig, decreasing = TRUE)]

dicesig <- 2 * kij / (ki + kj)
dicesig <- dicesig[order(dicesig, decreasing=TRUE)]

logsig <- 2 * ((k * log(k)) - (ki * log(ki)) - (kj * log(kj)) + (kij * log(kij))
              + (k - ki - kj + kij) * log(k - ki - kj + kij)
              + (ki - kij) * log(ki - kij) + (kj - kij) * log(kj - kij)
              - (k - ki) * log(k - ki) - (k - kj) * log(k - kj))
logsig <- logsig[order(logsig, decreasing=T)]

resultOverView <- data.frame(
  names(sort(kij, decreasing=T)[1:10]), sort(kij, decreasing=T)[1:10],
  names(mutualInformationSig[1:10]), mutualInformationSig[1:10],
  names(dicesig[1:10]), dicesig[1:10],
  names(logsig[1:10]), logsig[1:10],
  row.names = NULL)
colnames(resultOverView) <- c("Freq-terms", "Freq", "MI-terms", "MI", "Dice-Terms", "Dice",
"LL-Terms", "LL")
print(resultOverView)
```

```
##    Freq-terms Freq    MI-terms        MI  Dice-Terms      Dice    LL-Terms
## 1     leggere   14     leggere 2.2207551     leggere 1.0000000      facile
## 2      sapere    9      facile 1.3044643      facile 0.3333333 interessare
## 3     mettere    8 interessare 1.2091542 interessare 0.3200000      bianco
## 4      vedere    8      bianco 1.0421001      bianco 0.2962963        vero
## 5      andare    7      voglia 1.0167823        vero 0.2926829       libro
## 6      volere    7      verità 1.0167823       libro 0.2758621      voglia
## 7        mano    6       libro 0.8989992      capire 0.2702703      verità
## 8     parlare    6      lavoro 0.8344607      volere 0.2592593      capire
## 9        vero    6        nero 0.7738361        nero 0.2580645        nero
## 10    sentire    5        vero 0.7166777      voglia 0.2500000    guardare
##          LL
## 1  6.477521
## 2  5.693482
## 3  4.423185
## 4  3.915501
## 5  3.437913
## 6  3.048969
## 7  3.048969
## 8  2.913518
## 9  2.655799
## 10 2.626559
```

# Co-occurrence visualization

To visualize co-occurrence for a single term, we may use a graph which displays the terms co-occurrence network (including secondary co-occurrence levels).

```
source("resources/calculateCoocStatistics.R")
numberOfCoocs <- 10
coocTerm <- "libro"
coocs <- calculateCoocStatistics(coocTerm, binDTM, measure="LOGLIK")
```

```
## Caricamento del pacchetto richiesto: Matrix
```

```
##
## Caricamento pacchetto: 'Matrix'
```

```
## I seguenti oggetti sono mascherati da 'package:tidyr':
##
##     expand, pack, unpack
```

```
print(coocs[1:numberOfCoocs])
```

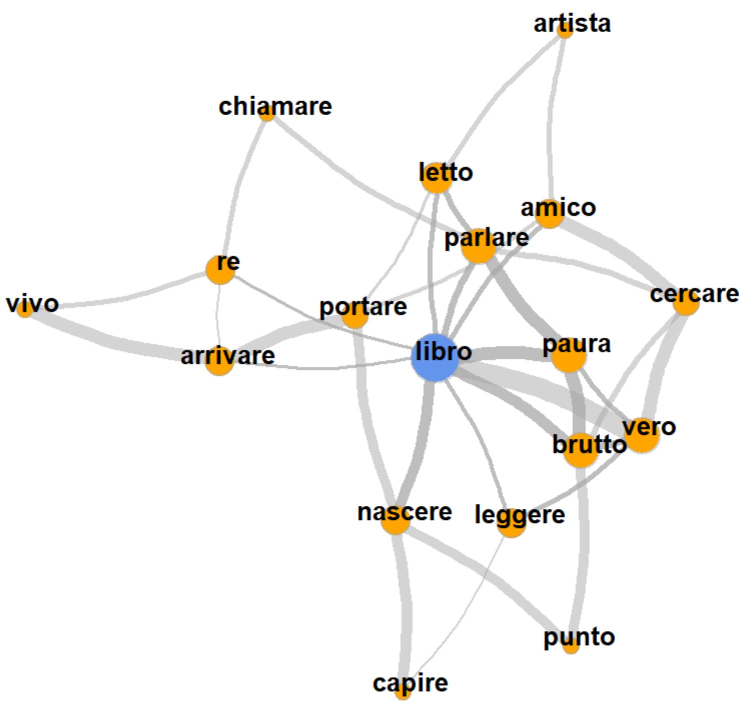
```
##     vero    paura  nascere   brutto  parlare    letto    amico  leggere
## 8.906301 6.536650 5.939264 5.939264 4.471098 3.941007 3.816210 3.437913
## arrivare       re
## 3.251808 2.994014
```

```
resultGraph <- data.frame(from = character(), to = character(), sig = numeric(0))
tmpGraph <- data.frame(from = character(), to = character(), sig = numeric(0))

# Fill the data.frame to produce the correct number of lines
tmpGraph[1:numberOfCoocs, 3] <- coocs[1:numberOfCoocs]
# Entry of the search word into the first column in all lines
tmpGraph[, 1] <- coocTerm
# Entry of the co-occurrences into the second column of the respective line
tmpGraph[, 2] <- names(coocs)[1:numberOfCoocs]
# Set the significances
tmpGraph[, 3] <- coocs[1:numberOfCoocs]

# Attach the triples to resultGraph
resultGraph <- rbind(resultGraph, tmpGraph)

# Iteration over the most significant numberOfCoocs co-occurrences of the search term
for (i in 1:numberOfCoocs){

  # Calling up the co-occurrence calculation for term i from the search words co-occurrences
  newCoocTerm <- names(coocs)[i]
  coocs2 <- calculateCoocStatistics(newCoocTerm, binDTM, measure="LOGLIK")

  #print the co-occurrences
  coocs2[1:10]

  # Structure of the temporary graph object
  tmpGraph <- data.frame(from = character(), to = character(), sig = numeric(0))
  tmpGraph[1:numberOfCoocs, 3] <- coocs2[1:numberOfCoocs]
  tmpGraph[, 1] <- newCoocTerm
  tmpGraph[, 2] <- names(coocs2)[1:numberOfCoocs]
  tmpGraph[, 3] <- coocs2[1:numberOfCoocs]

  #Append the result to the result graph
  resultGraph <- rbind(resultGraph, tmpGraph[2:length(tmpGraph[, 1]), ])
}

# Sample of some examples from resultGraph
resultGraph[sample(nrow(resultGraph), 6), ]
```

```
##         from      to      sig
## 98   leggere    nero 2.655799
## 1      libro    vero 8.906301
## 78   leggere  verità 3.048969
## 22     paura parlare 6.624673
## 39  arrivare    vivo 6.471441
## 86     letto  venere 3.426855
```

```r
# set seed for graph plot
set.seed(1)

# Create the graph object as undirected graph
graphNetwork <- graph.data.frame(resultGraph, directed = F)

# Identification of all nodes with less than 2 edges
verticesToRemove <- V(graphNetwork)[degree(graphNetwork) < 2]
# These edges are removed from the graph
graphNetwork <- delete.vertices(graphNetwork, verticesToRemove)

# Assign colors to nodes (search term blue, others orange)
V(graphNetwork)$color <- ifelse(V(graphNetwork)$name == coocTerm, 'cornflowerblue', 'orange')

# Set edge colors
E(graphNetwork)$color <- adjustcolor("DarkGray", alpha.f = .5)
# scale significance between 1 and 10 for edge width
E(graphNetwork)$width <- scales::rescale(E(graphNetwork)$sig, to = c(1, 10))

# Set edges with radius
E(graphNetwork)$curved <- 0.15
# Size the nodes by their degree of networking (scaled between 5 and 15)
V(graphNetwork)$size <- scales::rescale(log(degree(graphNetwork)), to = c(5, 15))

# Define the frame and spacing for the plot
par(mai=c(0,0,1,0))

# Final Plot
plot(
  graphNetwork,
  layout = layout.fruchterman.reingold, # Force Directed Layout
  main = paste(coocTerm, ' Graph'),
  vertex.label.family = "sans",
  vertex.label.cex = 0.8,
  vertex.shape = "circle",
  vertex.label.dist = 0.5,          # Labels of the nodes moved slightly
  vertex.frame.color = adjustcolor("darkgray", alpha.f = .5),
  vertex.label.color = 'black',     # Color of node names
  vertex.label.font = 2,            # Font of node names
  vertex.label = V(graphNetwork)$name,      # node names
  vertex.label.cex = 1 # font size of node names
)
```

## libro Graph



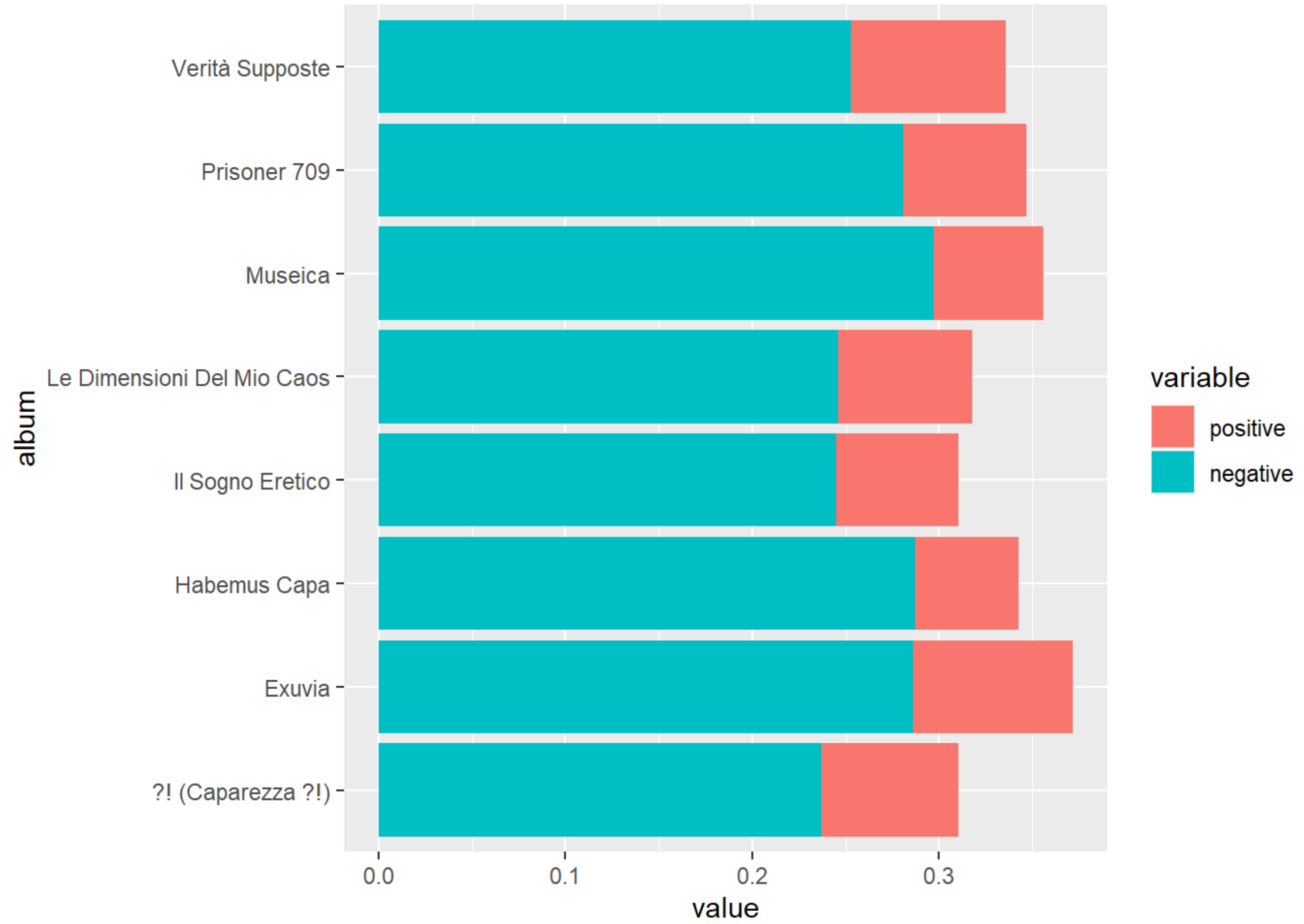

# Sentiment analysis

We are now going to perform a sentiment analysis on the lyrics of the songs. At this link (https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm), you can download NRC emotion lexicons in different languages. It allows you to catalogue words in positive / negative classes and even emotions, as we will see later.

```
sentiment_lexicon <- read.table("resources/Italian-NRC-EmoLex.txt",
                                header = TRUE, sep = "\t")
sentiment_lexicon_corpus <- sentiment_lexicon %>% filter(Italian.Word %in% colnames(DTM))
positive_terms <- sentiment_lexicon_corpus %>% filter(positive == 1) %>%
  select(Italian.Word) %>% pull()
negative_terms <- sentiment_lexicon_corpus %>% filter(positive == 0) %>%
  select(Italian.Word) %>% pull()
counts_positive <- rowSums(DTM[, positive_terms])
counts_negative <- rowSums(DTM[, negative_terms])
counts_all_terms <- rowSums(DTM)
relative_sentiment_frequencies <- data.frame(
  positive = counts_positive / counts_all_terms,
  negative = counts_negative / counts_all_terms
)
sentiments_by_album <- aggregate(relative_sentiment_frequencies,
                                 by = list(album = songs$album), mean)

head(sentiments_by_album)
```

```
##                        album   positive  negative
## 1          ?! (Caparezza ?!) 0.07331877 0.2368422
## 2                     Exuvia 0.08534713 0.2859161
## 3               Habemus Capa 0.05526262 0.2870167
## 4           Il Sogno Eretico 0.06532196 0.2448593
## 5 Le Dimensioni Del Mio Caos 0.07163986 0.2457957
## 6                    Museica 0.05849356 0.2971109
```

```
df_sentiment <- melt(sentiments_by_album, id.vars = "album")
ggplot(data = df_sentiment, aes(x = album, y = value, fill = variable)) +
  geom_bar(stat="identity", position="stack") + coord_flip()
```




Here we have catalogued every single word, then grouped words by albums and visualized the sentiment distribution for each album. We see that the sentiment is mostly negative for every single one. It has probably to do with the topics that appear in Caparezza's songs, who is notoriously a socially engaged singer and therefore often criticizes the hypocrisy of society.

It would be interesting to know if there is at least one song with more positive words than negative ones. Let's find out.

```
positive_songs <- aggregate(
  relative_sentiment_frequencies, by = list(album = songs$title),
  mean) %>% filter(positive > negative)
positive_songs
```

```
##                          album  positive  negative
## 1 Chi Se Ne Frega Della Musica 0.2097561 0.2000000
## 2                       Fugadà 0.2131783 0.1627907
## 3         Uomini di molta fede 0.1634615 0.1250000
```

Here they are. *Chi se ne frega della musica*, *Fugadà* and *Uomini di molta fede* have been classified as mostly positive songs.

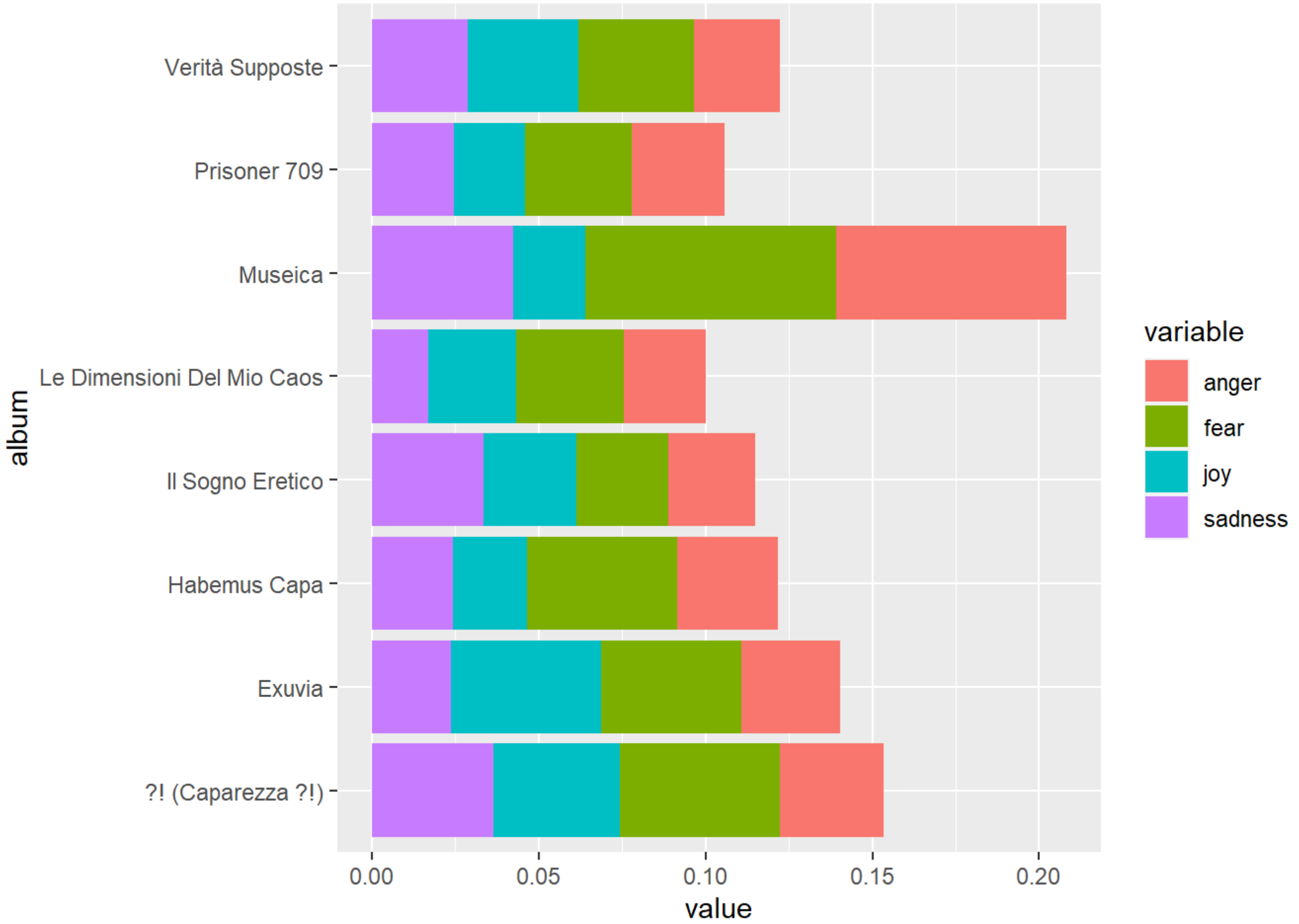
# Emotion analysis

For the last part of our analysis, we are going to look at the emotions, specifically anger, fear, joy and sadness, using the same NCR lexicon as before.

```
anger_terms <- sentiment_lexicon_corpus %>% filter(anger == 1) %>%
  select(Italian.Word) %>% pull()
fear_terms <- sentiment_lexicon_corpus %>% filter(fear == 1) %>%
  select(Italian.Word) %>% pull()
joy_terms <- sentiment_lexicon_corpus %>% filter(joy == 1) %>%
  select(Italian.Word) %>% pull()
sadness_terms <- sentiment_lexicon_corpus %>% filter(sadness == 1) %>%
  select(Italian.Word) %>% pull()

counts_anger <- rowSums(DTM[, anger_terms])
counts_fear <- rowSums(DTM[, fear_terms])
counts_joy <- rowSums(DTM[, joy_terms])
counts_sadness <- rowSums(DTM[, sadness_terms])

relative_emotion_frequencies <- data.frame(
  anger = counts_anger / counts_all_terms,
  fear = counts_fear / counts_all_terms,
  joy = counts_joy / counts_all_terms,
  sadness = counts_sadness / counts_all_terms
)

emotions_by_album <- aggregate(relative_emotion_frequencies,
                                 by = list(album = songs$album), mean)

head(emotions_by_album)
```

```
##                        album      anger       fear        joy    sadness
## 1          ?! (Caparezza ?!) 0.03095043 0.04822721 0.03773830 0.03649381
## 2                     Exuvia 0.02946261 0.04224000 0.04497954 0.02364693
## 3               Habemus Capa 0.03003343 0.04499670 0.02233911 0.02429495
## 4            Il Sogno Eretico 0.02612953 0.02751830 0.02772947 0.03357154
## 5 Le Dimensioni Del Mio Caos 0.02456764 0.03229070 0.02613714 0.01702260
## 6                    Museica 0.06870738 0.07543487 0.02139712 0.04249225
```

```
df_emotions <- melt(emotions_by_album, id.vars = "album")
ggplot(data = df_emotions, aes(x = album, y = value, fill = variable)) +
  geom_bar(stat="identity", position="stack") + coord_flip()
```




Anger and fear are the prevailing emotions and they are mostly noticeable in the *Museica* album. The album where joy shares a larger percentage is *Exuvia*.