R Notebook

Part-A

This chunk help us to load the dataset.

```
myFile <- 'D:/ahmed al/shz.txt' # the local file path to my research prospect
us
# fill = TRUE b/c rows are of unequal length
#fill: Sometimes, we may get a file that contains the
# unequal length of rows, and we have to add blank spaces to that missing val
ues.
dat <- read.table(myFile, header = FALSE, fill = TRUE)
```

Loading our required packages.

```
library(dplyr) # for data wrangling

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(tidytext) # for Natural Language Processing

## Warning: package 'tidytext' was built under R version 4.1.2

library(stringr) # to deal with strings
library(wordcloud) # to for generating word clouds

## Warning: package 'wordcloud' was built under R version 4.1.2

## Loading required package: RColorBrewer

library(knitr) # for tables, It combines many features into one package

## Warning: package 'knitr' was built under R version 4.1.2

library(DT) # for dynamic tables

## Warning: package 'DT' was built under R version 4.1.2

library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 4.1.2

#Tools to help to create tidy data, where each column is a variable,
#each row is an observation, and each cell contains a single value.
```

Gather help us to create a single colounm of all the words.

```
tidy_dat <- tidyr::gather(dat, key, word) %>% select(word)
head(tidy_dat)

##              word
## 1           Zyaed
## 2 Understanding
## 3            Born
## 4           Life,
## 5         Through
## 6             The
```

length tells us about the total number of words(also known as tokens in NLP)

```
tidy_dat$word %>% length() #there are 2832 tokens in my document
```

```
## [1] 2832
```

Unique help us to identify the unique words.

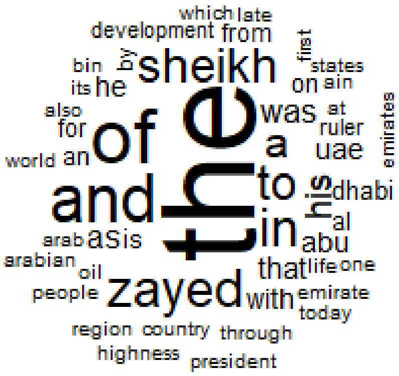
```
unique(tidy_dat$word) %>% length() # 695 words are unique

## [1] 695
```

Following commands help us to count the total number of words.

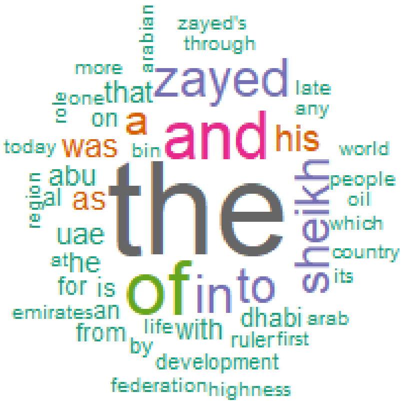
```
tokens <- tidy_dat %>%
unnest_tokens(word, word) %>%
dplyr::count(word, sort = TRUE) %>%
ungroup()
tokens %>% head(10)

##      word   n
## 1     the 137
## 2      of  73
## 3     and  62
## 4      in  45
## 5   zayed  45
## 6      to  40
## 7  sheikh  39
## 8       a  32
## 9     his  25
## 10    was  22
```

Word cloud is graph that shows the most frequent words by there size. Following commands use the total number of repeatitions to create the word cloud.

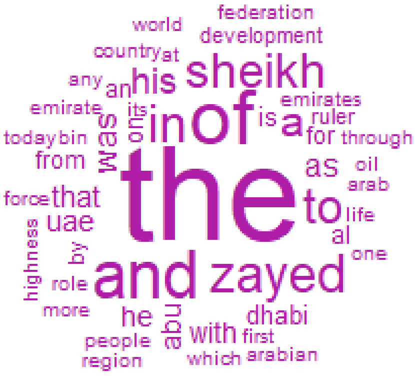
```
tokens %>% with(wordcloud(word, n, random.order = FALSE, max.words = 50))
```



```
# Module #: Module Name 61
# other choices
#colors=brewer.pal(8, “Dark2”)
tokens %>% with(wordcloud(word, n,
random.order = FALSE, max.words = 50,
colors=brewer.pal(8, "Dark2")))
```



```
tokens %>% with(wordcloud(word, n,
random.order = FALSE, max.words = 50,
colors="#AD1DA5"))
```



It removes the stopping words like is,or,am etc from the text data.

```
# remove stop words
data("stop_words")
tokens_clean <- tokens %>%
 anti_join(stop_words,by = "word")
```

It removes the numbers from the text data.

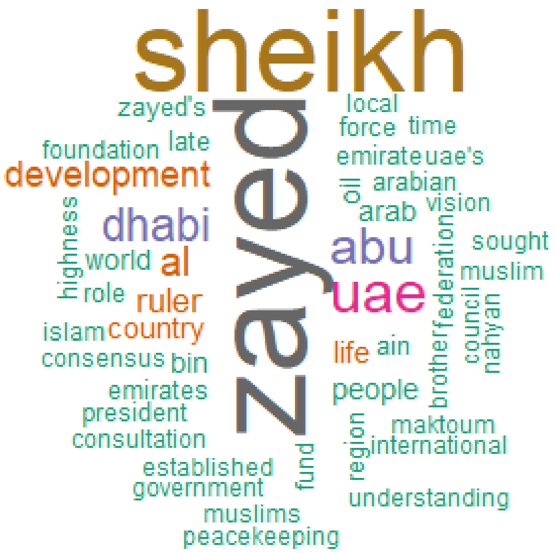
```
# remove numbers
nums <- tokens_clean %>% filter(str_detect(word, "^[0-9]")) %>%
select(word) %>% unique()
tokens_clean <- tokens_clean %>%
 anti_join(nums, by = "word")
```

Following code removes the i.e and e.g from the data.

```
# remove other stop words
uni_sw <- data.frame(word = c("i.e", "e.g."))
tokens_clean <- tokens_clean %>%
 anti_join(uni_sw, by = "word")
```

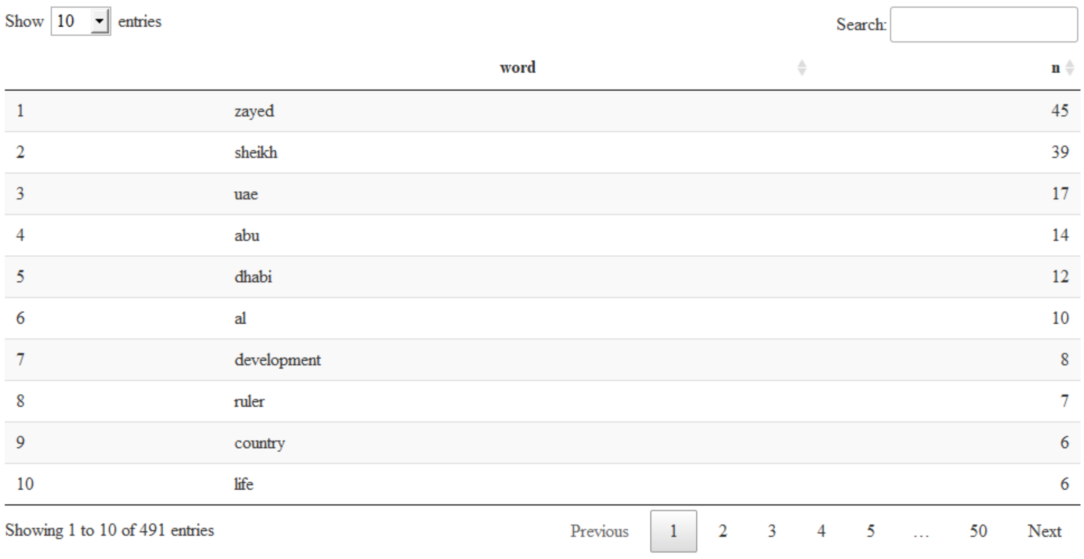
Creating the word cloud again without the numbers and stop words.

```
# define a nice color palette
pal <- brewer.pal(8,"Dark2")
# plot the 50 most common words
tokens_clean %>%
 with(wordcloud(word, n, random.order = FALSE, max.words = 50,
colors=pal))
```



Following commands show the clean data with word frequencies.

```
tokens_clean %>%
 DT::datatable()
```



Show 10 ▾ entries

Search:

	word	n
1	zayed	45
2	sheikh	39
3	uae	17
4	abu	14
5	dhabi	12
6	al	10
7	development	8
8	ruler	7
9	country	6
10	life	6

Showing 1 to 10 of 491 entries

Previous 1 2 3 4 5 … 50 Next

```
head(tokens_clean)
```

```
##     word  n
## 1  zayed 45
## 2 sheikh 39
## 3    uae 17
## 4    abu 14
## 5  dhabi 12
## 6     al 10
```

R sentiment packages help us to identify the sentiments from the text data. We created text and checked the sentiments with Rsentiments packages.

```
# Load the library
library(RSentiment)

## Warning: package 'RSentiment' was built under R version 4.1.3

 #: Module Name 69
calculate_total_presence_sentiment(c("This is a good text", "This is a bad te
xt",
"This is a really bad text", "This is horrible"))

## [1] "Processing sentence: this is a good text"
## [1] "Processing sentence: this is a bad text"
## [1] "Processing sentence: this is a really bad text"
## [1] "Processing sentence: this is horrible"

##      [,1]      [,2]       [,3]            [,4]      [,5]       [,6]
## [1,] "Sarcasm" "Negative" "Very Negative" "Neutral" "Positive" "Very Posit
ive"
## [2,] "0"       "3"        "0"             "0"       "1"        "0"
```

This is telling us about the positive and negative sentiments of the sentences.

```
calculate_sentiment(c("This is a good text",
 "This is a bad text",
 "This is a really bad text", "This is horrible"))

## [1] "Processing sentence: this is a good text"
## [1] "Processing sentence: this is a bad text"
## [1] "Processing sentence: this is a really bad text"
## [1] "Processing sentence: this is horrible"

##                         text sentiment
## 1       This is a good text  Positive
## 2        This is a bad text  Negative
## 3 This is a really bad text  Negative
## 4          This is horrible  Negative

calculate_score(c("This is a good text",
 "This is a bad text",
"This is a really bad text",
"This is horrible"))
```

```
## [1] "Processing sentence: this is a good text"
## [1] "Processing sentence: this is a bad text"
## [1] "Processing sentence: this is a really bad text"
## [1] "Processing sentence: this is horrible"
```

```
## [1]  1 -1 -1 -1
```

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

```
## Warning: package 'tidyverse' was built under R version 4.1.3
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.
3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.2     v forcats 0.5.1
## v readr   2.1.1
```

```
## Warning: package 'ggplot2' was built under R version 4.1.1
```

```
## Warning: package 'readr' was built under R version 4.1.2
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflict
s() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(janeaustenr)
```

```
## Warning: package 'janeaustenr' was built under R version 4.1.2
```

```
library(stringr)
library(wordcloud)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(textdata)
```

```
## Warning: package 'textdata' was built under R version 4.1.2
```

There are dictionaries to identify the sentiments of the text data. There are three different libraries, affin, buing and nrc. Affin package gives sentiments in the numeric form and bing identify the positive and negative sentiments. and nrc can identify the different kind of the sentiments like trust, fear, sadness, anger, etc..

```
get_sentiments("afinn")
```

```
## # A tibble: 2,477 x 2
##    word       value
##    <chr>      <dbl>
##  1 abandon       -2
##  2 abandoned     -2
##  3 abandons      -2
##  4 abducted      -2
##  5 abduction     -2
##  6 abductions    -2
##  7 abhor         -3
##  8 abhorred      -3
##  9 abhorrent     -3
## 10 abhors        -3
## # ... with 2,467 more rows
```

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

```
## # A tibble: 6,786 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 2-faces     negative
##  2 abnormal    negative
##  3 abolish     negative
##  4 abominable  negative
##  5 abominably  negative
##  6 abominate   negative
##  7 abomination negative
##  8 abort       negative
##  9 aborted     negative
## 10 aborts      negative
## # ... with 6,776 more rows
```

```
get_sentiments("nrc")
```

```
## # A tibble: 13,875 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 abacus      trust
##  2 abandon     fear
##  3 abandon     negative
##  4 abandon     sadness
##  5 abandoned   anger
##  6 abandoned   fear
##  7 abandoned   negative
##  8 abandoned   sadness
##  9 abandonment anger
## 10 abandonment fear
## # ... with 13,865 more rows
```

```
data(sentiments)
#dataset structure
str(sentiments)
```

```
## tibble [6,786 x 2] (S3: tbl_df/tbl/data.frame)
##  $ word     : chr [1:6786] "2-faces" "abnormal" "abolish" "abominable" ...
##  $ sentiment: chr [1:6786] "negative" "negative" "negative" "negative" ...
```

We are saving the affins as the affin_lexiocon variable. and we are doing same thing for the afinn, bing, nrc.

```
afinn_lexicon <- get_sentiments("afinn")
head(afinn_lexicon)
```

```
## # A tibble: 6 x 2
##   word       value
##   <chr>      <dbl>
## 1 abandon       -2
## 2 abandoned     -2
## 3 abandons      -2
## 4 abducted      -2
## 5 abduction     -2
## 6 abductions    -2
```

```
#NRC
nrc_lexicon <- get_sentiments("nrc")
head(nrc_lexicon)
```

```
## # A tibble: 6 x 2
##   word      sentiment
##   <chr>     <chr>
## 1 abacus    trust
## 2 abandon   fear
## 3 abandon   negative
## 4 abandon   sadness
## 5 abandoned anger
## 6 abandoned fear
```

```
#BING
bing_lexicon <- get_sentiments("bing")
head(bing_lexicon)
```

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

We are using Jane Austens data of books for the further analysis.

```
tidy_books <- austen_books() %>%
 filter(book == "Emma") %>%
 group_by(book) %>%
## Using row_number() with mutate() will create a column of consecutive numbe
rs. The row_number() function
#is useful for creating an identification number (an ID variable). It is also
useful for labeling each observation by a
#grouping variable.
##cumsum() function in R Language is used to calculate the cumulative sum of
the vector passed as
## argument
# str_detect function returns a logical value (i.e. FALSE or TRUE),
 mutate(linenumber = row_number(),
 chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]",

 ignore_case = TRUE)))) %>% ungroup() %>%
 unnest_tokens(word, text)
```

following codes separate the words with joy only in nrc library.

```
nrc_joy <- get_sentiments("nrc") %>%
 filter(sentiment == "joy")
#Summarize the usage of `joy` words
tidy_books %>%
 semi_join(nrc_joy) %>%
 count(word, sort = T)

## Joining, by = "word"

## # A tibble: 301 x 2
##    word          n
##    <chr>     <int>
##  1 good        359
##  2 friend      166
##  3 hope        143
##  4 happy       125
##  5 love        117
##  6 deal         92
##  7 found        92
##  8 present      89
##  9 kind         82
## 10 happiness    76
## # ... with 291 more rows

tidy_books <- austen_books() %>%
 group_by(book) %>%
 mutate(linenumber = row_number(),
 chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]",
 ignore_case = TRUE)))) %>%

 ungroup() %>%
 unnest_tokens(word, text)
```

In the following codes we are generateing the random data and using it for the sentimental analysis.

```
## Creating identification number to represent 50 individual people
ID <- c(1:20)
## Creating sex variable (10 males/10 females)
Sex <- rep(c("male", "female"), 10) # rep stands for replicate
## Creating age variable (20-39 year olds)
Age <- c(26, 25, 39, 37, 31, 34, 34, 30, 26, 33,
 39, 28, 26, 29, 33, 22, 35, 23, 26, 36)
## Creating a dependent variable called Score
Score <- c(0.010, 0.418, 0.014, 0.090, 0.061, 0.328, 0.656, 0.002, 0.639, 0.1
73,
 0.076, 0.152, 0.467, 0.186, 0.520, 0.493, 0.388, 0.501, 0.800, 0.482)
## Creating a unified dataset that puts together all variables
## tibble is a simple dataframe
data <- tibble(ID, Sex, Age, Score)
## group by sex
data %>%
 group_by(Sex) %>%
 summarize(m = mean(Score), # calculates the mean
 s = sd(Score), # calculates the standard deviation
 n = n()) %>% # calculates the total number of observations
 ungroup()
```

```
## # A tibble: 2 x 4
##   Sex        m     s     n
##   <chr>  <dbl> <dbl> <int>
## 1 female 0.282 0.184    10
## 2 male   0.363 0.300    10
```

```
##`summarise()` ungrouping output (override with `.groups` argument)
# A tibble: 2 x 4
##x m s n
##hr> <dbl> <dbl> <int>
##1 female 0.282 0.184 10
##2 male 0.363 0.300 10
## mutate() and group_by()
data %>%
 group_by(Sex) %>%
 mutate(m = mean(Score)) %>% # calculates mean score by Sex
 ungroup()
```

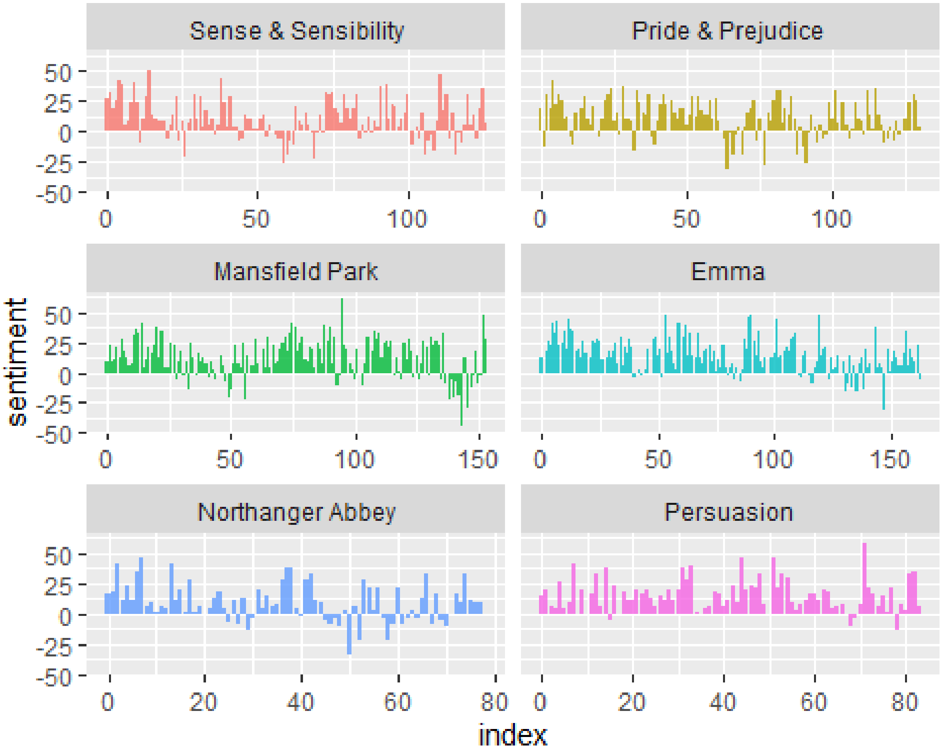
```
## # A tibble: 20 x 5
##       ID Sex      Age Score     m
##    <int> <chr>  <dbl> <dbl> <dbl>
##  1      1 male      26 0.01  0.363
##  2      2 female    25 0.418 0.282
##  3      3 male      39 0.014 0.363
##  4      4 female    37 0.09  0.282
##  5      5 male      31 0.061 0.363
##  6      6 female    34 0.328 0.282
##  7      7 male      34 0.656 0.363
##  8      8 female    30 0.002 0.282
##  9      9 male      26 0.639 0.363
## 10     10 female    33 0.173 0.282
## 11     11 male      39 0.076 0.363
## 12     12 female    28 0.152 0.282
## 13     13 male      26 0.467 0.363
## 14     14 female    29 0.186 0.282
## 15     15 male      33 0.52  0.363
## 16     16 female    22 0.493 0.282
## 17     17 male      35 0.388 0.363
## 18     18 female    23 0.501 0.282
## 19     19 male      26 0.8   0.363
## 20     20 female    36 0.482 0.282
```

```
janeaustensentiment <- tidy_books %>%
 inner_join(get_sentiments("bing")) %>%
 count(book, index = linenumber %/% 100, sentiment) %>%

 spread(sentiment, n, fill = 0) %>%

 mutate(sentiment = positive - negative)
```

```
## Joining, by = "word"
```

Following plot shows the data for the sentiments for different chapters of the Jane book.

```
ggplot(data = janeaustensentiment, mapping = aes(x = index, y =
sentiment, fill = book)) +
 geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
 facet_wrap(facets = ~ book, ncol = 2, scales = "free_x")
```



Part-B

I am using texts available on this website. https://www.toppr.com/guides/essays/freedom-essay/ . Lets load the dataset.

```
myFile <- 'D:/ahmed al/txt.txt'
dat <- read.table(myFile, header = FALSE, fill = TRUE)
head(dat)

##        V1    V2     V3       V4      V5       V6  V7       V8  V9      V10
## 1  SIGNUP
## 2   LOGIN
## 3  Search   for      a    topic
## 4 English     > Essays        > Freedom    Essay for Students and Children
## 5  Essays
## 6 Freedom Essay    for Students     and Children
```

let’s reshape the dataset for the sentimental analysis.

```
tidy_dat <- tidyr::gather(dat, key, word) %>% select(word)
head(tidy_dat)

##      word
## 1  SIGNUP
## 2   LOGIN
## 3  Search
## 4 English
## 5  Essays
## 6 Freedom
```

Counting the words in the text file.

```
tokens <- tidy_dat %>%
unnest_tokens(word, word) %>%
dplyr::count(word, sort = TRUE) %>%
ungroup()
tokens %>% head(10)
```

```
##       word  n
## 1  freedom 37
## 2      the 35
## 3       of 34
## 4      and 28
## 5       is 19
## 6    essay 17
## 7    right 17
## 8       to 17
## 9      for 16
## 10      it 14
```

Let's remove the stop words from the dataset.

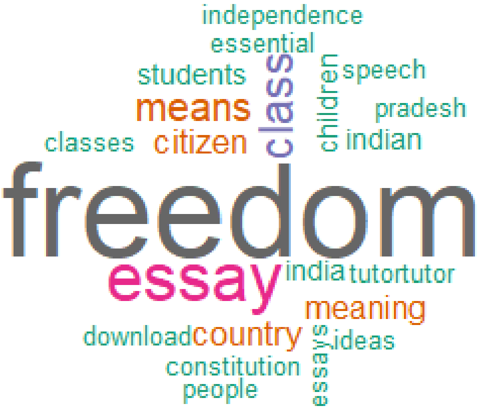
```
# remove stop words
data("stop_words")
tokens_clean <- tokens %>%
 anti_join(stop_words,by = "word")
```

Let's remove the numbers from the dataset.

```
# remove numbers
nums <- tokens_clean %>% filter(str_detect(word, "^[0-9]")) %>%
select(word) %>% unique()
tokens_clean <- tokens_clean %>%
 anti_join(nums, by = "word")
```

Let's create the word cloud of the data.

```
pal <- brewer.pal(8,"Dark2")
# plot the 50 most common words
tokens_clean %>%
 with(wordcloud(word, n, random.order = FALSE, max.words = 50,
colors=pal))
```



Part-C

```
#install.packages("SentimentAnalysis")
library(SentimentAnalysis)

## Warning: package 'SentimentAnalysis' was built under R version 4.1.2

##
## Attaching package: 'SentimentAnalysis'

## The following object is masked from 'package:base':
##
##     write

sentiment <- analyzeSentiment("Yeah, this was a great soccer game for the Ger
man team!")
sentiment$PositivityGI

## [1] 0.3333333

#In case you had any errors, run this line of code:
#install.packages("SnowballC")
library(SnowballC)

## Warning: package 'SnowballC' was built under R version 4.1.1

sentiment$NegativityGI
```

```
## [1] 0

sentiment$WordCount

## [1] 6
```

Positive score is 0.3333 and negative score is zero. The word count is 6.

```
documents <- c ("Wow, I really like the new light sabers!",
                "That book was excellent.",
                "R is a fantastic language.",
                "The service in this restaurant was miserable.",
                "This is neither positive or negative.",
                "The waiter forget about my dessert -- what poor service!")
sentiment <- analyzeSentiment(documents)
sentiment$SentimentQDAP

## [1]  0.3333333  0.5000000  0.5000000 -0.3333333  0.0000000 -0.4000000

convertToDirection(sentiment$SentimentQDAP)

## [1] positive positive positive negative neutral  negative
## Levels: negative neutral positive

convertToBinaryResponse(sentiment$SentimentQDAP)

## [1] positive positive positive negative positive negative
## Levels: negative positive
```

This code gets the sentiment score for the sentences and convert the sentiment scores into the binary responses, and converts it into factors.

```
response <- c(+1, +1, +1, -1, 0, -1)
compareToResponse(sentiment, response)

## Warning in cor(sentiment, response): the standard deviation is zero

## Warning in cor(x, y): the standard deviation is zero

## Warning in cor(x, y): the standard deviation is zero

## Warning in cor(sentiment, response): the standard deviation is zero

##                               WordCount  SentimentGI  NegativityGI Positivi
tyGI
## cor                         -0.18569534  0.990011498 -9.974890e-01  0.94295
4167
## cor.t.statistic             -0.37796447 14.044046450 -2.816913e+01  5.66470
5543
## cor.p.value                  0.72465864  0.000149157  9.449687e-06  0.00478
8521
## lm.t.value                  -0.37796447 14.044046450 -2.816913e+01  5.66470
5543
```

```
## r.squared                  0.03448276  0.980122766  9.949843e-01  0.88916
2562
## RMSE                       3.82970843  0.450102869  1.186654e+00  0.71362
4032
## MAE                        3.33333333  0.400000000  1.100000e+00  0.66666
6667
## Accuracy                   0.66666667  1.000000000  6.666667e-01  0.66666
6667
## Precision                         NaN  1.000000000           NaN
NaN
## Sensitivity                0.00000000  1.000000000  0.000000e+00  0.00000
0000
## Specificity                1.00000000  1.000000000  1.000000e+00  1.00000
0000
## F1                                NaN  1.000000000           NaN
NaN
## BalancedAccuracy           0.50000000  1.000000000  5.000000e-01  0.50000
0000
## avg.sentiment.pos.response 3.25000000  0.333333333  8.333333e-02  0.41666
6667
## avg.sentiment.neg.response 4.00000000 -0.633333333  6.333333e-01  0.00000
0000
##                           SentimentHE NegativityHE PositivityHE Sentiment
LM
## cor                          0.4152274 -0.083045480    0.3315938   0.73704
55
## cor.t.statistic              0.9128709 -0.166666667    0.7029595   2.18111
42
## cor.p.value                  0.4129544  0.875718144    0.5208394   0.09462
66
## lm.t.value                   0.9128709 -0.166666667    0.7029595   2.18111
42
## r.squared                    0.1724138  0.006896552    0.1099545   0.54323
61
## RMSE                         0.8416254  0.922958207    0.8525561   0.72341
78
## MAE                          0.7500000  0.888888889    0.8055556   0.63333
33
## Accuracy                     0.6666667  0.666666667    0.6666667   0.83333
33
## Precision                          NaN          NaN          NaN   1.00000
00
## Sensitivity                  0.0000000  0.000000000    0.0000000   0.50000
00
## Specificity                  1.0000000  1.000000000    1.0000000   1.00000
00
## F1                                 NaN          NaN          NaN   0.66666
67
## BalancedAccuracy             0.5000000  0.500000000    0.5000000   0.75000
00
```

```
## avg.sentiment.pos.response   0.1250000  0.083333333    0.2083333   0.25000
00
## avg.sentiment.neg.response   0.0000000  0.000000000    0.0000000  -0.10000
00
##                            NegativityLM PositivityLM RatioUncertaintyLM
## cor                         -0.40804713    0.6305283                 NA
## cor.t.statistic             -0.89389841    1.6247248                 NA
## cor.p.value                  0.42189973    0.1795458                 NA
## lm.t.value                  -0.89389841    1.6247248                 NA
## r.squared                    0.16650246    0.3975659                 NA
## RMSE                         0.96186547    0.7757911          0.9128709
## MAE                          0.92222222    0.7222222          0.8333333
## Accuracy                     0.66666667    0.6666667          0.6666667
## Precision                           NaN          NaN                NaN
## Sensitivity                  0.00000000    0.0000000          0.0000000
## Specificity                  1.00000000    1.0000000          1.0000000
## F1                                  NaN          NaN                NaN
## BalancedAccuracy             0.50000000    0.5000000          0.5000000
## avg.sentiment.pos.response   0.08333333    0.3333333          0.0000000
## avg.sentiment.neg.response   0.10000000    0.0000000          0.0000000
##                            SentimentQDAP NegativityQDAP PositivityQDAP
## cor                          0.9865356369   -0.944339551    0.942954167
## cor.t.statistic             12.0642877257   -5.741148345    5.664705543
## cor.p.value                  0.0002707131    0.004560908    0.004788521
## lm.t.value                  12.0642877257   -5.741148345    5.664705543
## r.squared                    0.9732525629    0.891777188    0.889162562
## RMSE                         0.5398902495    1.068401367    0.713624032
## MAE                          0.4888888889    1.011111111    0.666666667
## Accuracy                     1.0000000000    0.666666667    0.666666667
## Precision                    1.0000000000            NaN            NaN
## Sensitivity                  1.0000000000    0.000000000    0.000000000
## Specificity                  1.0000000000    1.000000000    1.000000000
## F1                           1.0000000000            NaN            NaN
## BalancedAccuracy             1.0000000000    0.500000000    0.500000000
## avg.sentiment.pos.response   0.3333333333    0.083333333    0.416666667
## avg.sentiment.neg.response  -0.3666666667    0.366666667    0.000000000
```

We created the exact responses for the sentences and compared them with the sentiment score by our analysis and got all the scores by comparison.

PArt-D

```
##1.2 The unnest_tokens function
#Emily Dickinson wrote some lovely text in her time.
text <- c("Because I could not stop for Death -",
          "He kindly stopped for me -",
          "The Carriage held but just Ourselves -",
          "and Immortality")
text

## [1] "Because I could not stop for Death -"
## [2] "He kindly stopped for me -"
## [3] "The Carriage held but just Ourselves -"
## [4] "and Immortality"

##This is a typical character vector that we might want to analyze.
##To turn it into a tidy text dataset,
##we first need to put it into a data frame.
library(dplyr)
text_df <- tibble(line = 1:4, text = text)
text_df

## # A tibble: 4 x 2
##    line text
##   <int> <chr>
## 1     1 Because I could not stop for Death -
## 2     2 He kindly stopped for me -
## 3     3 The Carriage held but just Ourselves -
## 4     4 and Immortality

#Within our tidy text framework, we need to both break the text into
#individual tokens (a process called tokenization) and transform it to
#a tidy data structure. To do this, we use tidytext’s unnest_tokens() functio
n.
library(tidytext)
text_df %>%
  unnest_tokens(word, text)

## # A tibble: 20 x 2
##     line word
##    <int> <chr>
##  1     1 because
##  2     1 i
##  3     1 could
##  4     1 not
##  5     1 stop
##  6     1 for
##  7     1 death
##  8     2 he
##  9     2 kindly
## 10     2 stopped
## 11     2 for
## 12     2 me
## 13     3 the
## 14     3 carriage
## 15     3 held
## 16     3 but
## 17     3 just
## 18     3 ourselves
## 19     4 and
## 20     4 immortality
```

Just appear at the line 3.

```
##1.3 Tidying the works of Jane Austen published novels
library(janeaustenr)
library(dplyr)
library(stringr)
original_books <- austen_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number(),
         chapter = cumsum(str_detect(text,
                                     regex("^chapter [\\divxlc]",
                                           ignore_case = TRUE)))) %>%
  ungroup()
original_books

## # A tibble: 73,422 x 4
##    text                  book                linenumber chapter
##    <chr>                 <fct>                    <int>   <int>
##  1 "SENSE AND SENSIBILITY" Sense & Sensibility          1       0
##  2 ""                    Sense & Sensibility          2       0
##  3 "by Jane Austen"      Sense & Sensibility          3       0
##  4 ""                    Sense & Sensibility          4       0
##  5 "(1811)"              Sense & Sensibility          5       0
##  6 ""                    Sense & Sensibility          6       0
##  7 ""                    Sense & Sensibility          7       0
##  8 ""                    Sense & Sensibility          8       0
##  9 ""                    Sense & Sensibility          9       0
## 10 "CHAPTER 1"           Sense & Sensibility         10       1
## # ... with 73,412 more rows
```

CHAPTER 1 appears at the line 10 and chapter 1.

```
#To work with this as a tidy dataset, we need to restructure it in the one-to
ken-per-row format,
#which as we saw earlier is done with the unnest_tokens() function.
library(tidytext)
tidy_books <- original_books %>%
  unnest_tokens(word, text)
tidy_books

## # A tibble: 725,055 x 4
##    book                linenumber chapter word
##    <fct>                    <int>   <int> <chr>
##  1 Sense & Sensibility          1       0 sense
##  2 Sense & Sensibility          1       0 and
##  3 Sense & Sensibility          1       0 sensibility
##  4 Sense & Sensibility          3       0 by
##  5 Sense & Sensibility          3       0 jane
##  6 Sense & Sensibility          3       0 austen
##  7 Sense & Sensibility          5       0 1811
##  8 Sense & Sensibility         10       1 chapter
##  9 Sense & Sensibility         10       1 1
## 10 Sense & Sensibility         13       1 the
## # ... with 725,045 more rows
```

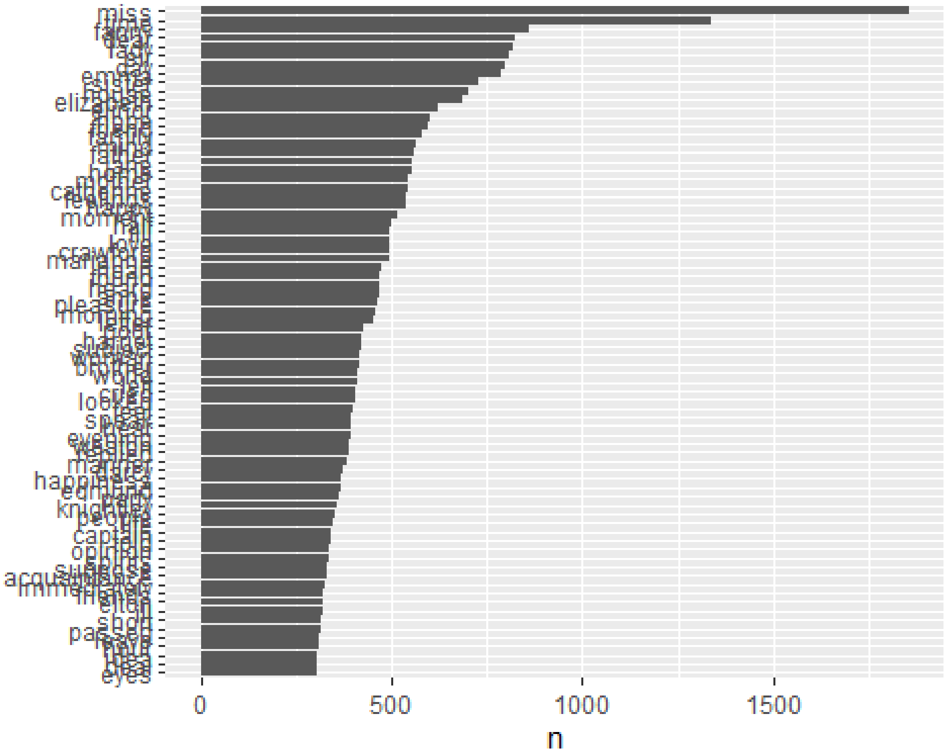
```
#This function uses the tokenizers package to separate each line of text in t
he original data frame into tokens.
## remove stop words (kept in the tidytext dataset stop_words)
data(stop_words)
tidy_books <- tidy_books %>%
  anti_join(stop_words)

## Joining, by = "word"
```

Tokenization means splitting the sentences into words and counting the appearance of the words. It remove all the stop words from the data.

```
##use dplyr’s count() to find the most common words in all the books as a who
le
tidy_books %>%
  count(word, sort = TRUE)

## # A tibble: 13,914 x 2
##    word       n
##    <chr>  <int>
##  1 miss    1855
##  2 time    1337
##  3 fanny    862
##  4 dear     822
##  5 lady     817
##  6 sir      806
##  7 day      797
##  8 emma     787
##  9 sister   727
## 10 house    699
## # ... with 13,904 more rows
```

These are the top 5 common words. miss time fanny dear and lady.

```
##create a visualization of the most common words
library(ggplot2)

tidy_books %>%
  count(word, sort = TRUE) %>%
  filter(n > 300) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word)) +
  geom_col() +
  labs(y = NULL)
```



The above code counts the words frequency, and filter the words that occur more than 300 times. After that we reorder the data and plot them on the y axis in the graph. The word ‘miss’ is the most occuring and ‘eyes’ is the least occuring.