Group Project Report

Group #5 - Saakshi, My, Lachlan, Sarah, Nazeef

21 October 2022

Preliminary Tasks

```r
# import required packages, supress any warnings shown

library("rtweet") # access Twitter API in R
library("tm") # used to convert tweets to a term frequency matrix
library("SnowballC") # used to stem the tweet text
library("dplyr") # need for distinct function

library("wordcloud") # word cloud library for visualisation
library("igraph") # used to build a mention network
library("dendextend") # used for extending dendrogram objects
library("tidyverse") # used to wrangle data for time-series plot
```

Project Tasks

Suppose we own a business called "AnalysisInMordor" which helps marketing and analytics teams build better businesses with data. We are trying to find out how Twitter data can provide insights about a **public figure - @BillGates** and planning to do the specified project tasks on a test user.

Search for 1600 tweets about the twitter handle we selected, and save it in a variable named "comp3020.dataset."

This is done in the below code chunk but the output is not printed. Instead, the **saved data file is loaded and will be used throughout this report** for each questions' analysis.

```r
# record your authentication keys to create token as an environment variable
app = ""
key = ""
secret = ""
access_token = ""
access_secret = ""

# perform OAuth authentication
twitter_token = create_token(app, key, secret, access_token, access_secret,
                             set_renv= FALSE)

# retrieve 1,600 raw tweets and save object
tweets = search_tweets('Bill Gates', n = 1600, type = "recent",
                       token = twitter_token, include_rts= TRUE)

save(tweets, file="comp3020.dataset.RData")
```

```r
# prepare data, by loading in the sourced data file
load("comp3020.dataset.RData") # use the set of tweets obtained

class(tweets) # quick check of loaded data
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```r
names(tweets) # examine the column headings
```

```
##  [1] "created_at"                    "id"
##  [3] "id_str"                        "full_text"
##  [5] "truncated"                     "display_text_range"
##  [7] "entities"                      "metadata"
##  [9] "source"                        "in_reply_to_status_id"
## [11] "in_reply_to_status_id_str"     "in_reply_to_user_id"
## [13] "in_reply_to_user_id_str"       "in_reply_to_screen_name"
## [15] "geo"                           "coordinates"
## [17] "place"                         "contributors"
## [19] "is_quote_status"               "retweet_count"
## [21] "favorite_count"                "favorited"
## [23] "retweeted"                     "lang"
## [25] "retweeted_status"              "quoted_status_id"
## [27] "quoted_status_id_str"          "possibly_sensitive"
## [29] "quoted_status"                 "text"
## [31] "favorited_by"                  "scopes"
## [33] "display_text_width"            "quoted_status_permalink"
## [35] "quote_count"                   "timestamp_ms"
## [37] "reply_count"                   "filter_level"
## [39] "query"                         "withheld_scope"
## [41] "withheld_copyright"            "withheld_in_countries"
## [43] "possibly_sensitive_appealable"
```

Statistical Analysis

Question 1

You want to know if there is a relationship between the followers' count and the favourites count, so that AnalysisInMordor could provide advice to their customers to increase their follower count.

Since followers count and favourites count are related to the user of the tweet, you should find out the unique users in your tweet sample (comp3020.dataset.). Then group up both variables and create a contingency table from both variables as follows:

- For **followers count**, use the grouping below and label each user with their category as either *Low, Average or High*:

$$0 \leq count < 1000\text{: Low}$$
$$1000 \leq count < 5000\text{: Average}$$
$$5000 \leq count \leq \infty\text{: High}$$

- For **favourites count**, use the grouping below and label each user with their category as either *Low, Average or High*:

$$0 \leq count < 300\text{: Low}$$
$$300 \leq count < 1000\text{: Average}$$
$$1000 \leq count \leq \infty\text{: High}$$

After labelling each user, create a contingency table of the variables.

```
users <- users_data(tweets) # gets user ID so you can extract follower and favorite count
```

```
# favorite count keep consistent throughout analysis: low favourite
  # the followers count goes through each respective grouping
l_l=users[users$followers_count < 1000
          & users$favourites_count < 300, ] # extracts requested data for all variables
l_l=distinct(l_l, l_l$id, .keep_all = TRUE) # get unique user data, removes duplicates
a_l=users[users$followers_count >= 1000 & users$followers_count < 5000
          & users$favourites_count < 300, ]
a_l=distinct(a_l, a_l$id, .keep_all = TRUE)
h_l=users[users$followers_count >= 5000 & users$favourites_count < 300, ]
h_l=distinct(h_l, h_l$id, .keep_all = TRUE)

# favorite count keep consistent throughout analysis: average favourite
  # the followers count goes through each respective grouping
l_a=users[users$followers_count < 1000
          & users$favourites_count >= 300 & users$favourites_count < 1000, ]
l_a=distinct(l_a, l_a$id, .keep_all = TRUE)
a_a=users[users$followers_count >= 1000 & users$followers_count < 5000
          & users$favourites_count >= 300 & users$favourites_count < 1000, ]
a_a=distinct(a_a, a_a$id, .keep_all = TRUE)
h_a=users[users$followers_count >= 5000
```

```
          & users$favourites_count >= 300 & users$favourites_count < 1000, ]
h_a=distinct(h_a, h_a$id, .keep_all = TRUE)

# favorite count keep consistent throughout analysis: high favourite
  # the followers count goes throughe each respective grouping
l_h=users[users$followers_count < 1000 & users$favourites_count >= 1000, ]
l_h=distinct(l_h, l_h$id, .keep_all = TRUE)
a_h=users[users$followers_count >= 1000 & users$followers_count < 5000
          & users$favourites_count >= 1000, ]
a_h=distinct(a_h, a_h$id, .keep_all = TRUE)
h_h=users[users$followers_count >= 5000 & users$favourites_count >= 1000, ]
h_h=distinct(h_h, h_h$id, .keep_all = TRUE)
```

```
# create data frame with columns being favorite count and use nrow() to find count
count = data.frame(
  "Low" = c(nrow(l_l), nrow(a_l), nrow(h_l)),
  "Average" = c(nrow(l_a), nrow(a_a), nrow(h_a)),
  "High" = c(nrow(l_h), nrow(a_h), nrow(h_h)))

rownames(count) = c("Low", "Average", "High") # add rownames to columns based on follower count
print(count)
```

```
##         Low Average High
## Low      90      67  756
## Average   6       4  297
## High      6       5  101
```

```
conTable <- as.matrix(count) # convert data frame to a matrix
names(dimnames(conTable)) <- c("Favourites Count", "Followers Count") # name the dimnames

conTable <- as.table(conTable) # create a table to hold our data
class(conTable) # verify data has "table" class
```

```
## [1] "table"
```

```
print(conTable) # print contingency table of the variables
```

```
##                 Followers Count
## Favourites Count Low Average High
##          Low      90      67  756
##          Average   6       4  297
##          High      6       5  101
```

By using an appropriate statistical test, test whether the followers' count and favourites count are related.

```
chisq.test(conTable)
```

```
##
##  Pearson's Chi-squared test
##
## data:  conTable
## X-squared = 39.912, df = 4, p-value = 4.513e-08
```

```
chisq.test(conTable, simulate.p.value = TRUE) # when simulating the p-value
```

```
##
##  Pearson's Chi-squared test with simulated p-value (based on 2000
##  replicates)
##
## data:  conTable
## X-squared = 39.912, df = NA, p-value = 0.0004998
```

Question 2

Interpret the result of the above test. Is the favourites count related to the follower count? What does being related mean in this context.

Intepret your findings.

First we define the null and alternative hypothesis as the following:

- H_0 **(null hypothesis)** = the favourite count is not related (independent) to the followers' count
- H_1 **(alternative hypothesis)** = the favourite count is related (dependent) to the followers' count

For the variables to be related, it means that the favourites count is dependent on the followers' count. This means that a change in one variable, i.e. favourites count will directly (either proportionally or with inverse proportion) effect the result of the other variable, i.e. followers' count.

When performing the chi-squared test as per normal conditions, the p-value found is extremely small, so we can successfully reject the null hypothesis. This means we have found significant evidence that the alternative hypothesis is true, and can conclude that the favourites count is related to the followers' count.

Furthermore, when simulating the p-value based on 2000 replicates, the p-value continues to produce a small result. This reaffirms that we can successfully **reject the null hypothesis and have found significant evidence in favour of the alternative hypothesis** - the followers' count and favourites count are related.

This analysis and results for p-value is further supported by the summary function below.

```
summary(conTable)
```

```
## Number of cases in table: 1332
## Number of factors: 2
## Test for independence of all factors:
##  Chisq = 39.91, df = 4, p-value = 4.513e-08
```

Text Mining

Get the **tweet text from the tweet object** of tweets stored in comp3020.dataset.RData

Question 3

Pre-process your text data and construct a Document Term Matrix by using TF-IDF weights.

Pre-process our text data, by creating Document Term Matrix.

Also note that TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. Using manual calculation:

- **TF** = log(frequency of term + 1)
- **IDF** = log(number of documents / number of documents containing term t)

However, for the purpose of our analysis, we used the built in function.

```
# create a corpus from tweet text - this will convert all the tweets into
  # a vector containing each tweet with each new element being a new tweet
tweet.corpus = Corpus(VectorSource(tweets$text))

# convert the characters to ASCII
corpus = tm_map(tweet.corpus, function(x) iconv(x, to='ASCII'))

corpus = tm_map(corpus, removeNumbers) # remove numbers
corpus = tm_map(corpus, removePunctuation) # remove punctuation
corpus = tm_map(corpus, stripWhitespace) # remove whitespace
corpus = tm_map(corpus, tolower) # convert all to lowercase
corpus = tm_map(corpus, removeWords, stopwords()) # remove stopwords
corpus = tm_map(corpus, stemDocument) # convert all words to their stems

# to perform clustering, obtain document vectors contained in the
  # document term matrix, and extract the matrix and apply TF-IDF weighting

tweet.tdm = TermDocumentMatrix(tweet.corpus) # create a term document matrix
tweet.wtdm = weightTfIdf(tweet.tdm) # TDM using TF-IDF weights (using R function)
tweet.matrix = t(as.matrix(tweet.wtdm)) # change TDM to a document term matrix

# remove empty tweets so they don't effect the calculations
empties = which(rowSums(abs(tweet.matrix)) == 0)
empties # returns integer(0) as there are no empty tweets
```

```
## named integer(0)
```

```
length(empties) # return length of empties vector
```

```
## [1] 0
```

```
# create a conditional statement, where if length of empties is 0 return
  # the tweet.matrix as it is, otherwise remove empty tweets in matrix
if (length(empties) == 0) {
  tweet.matrix = tweet.matrix # return unchanged matrix
} else {
  tweet.matrix = tweet.matrix[-empties, ] # remove empty tweets
}
```

```
dim(tweet.matrix) # view the dimensions of the document term matrix (DTM)
```

```
## [1] 1431 6200
```

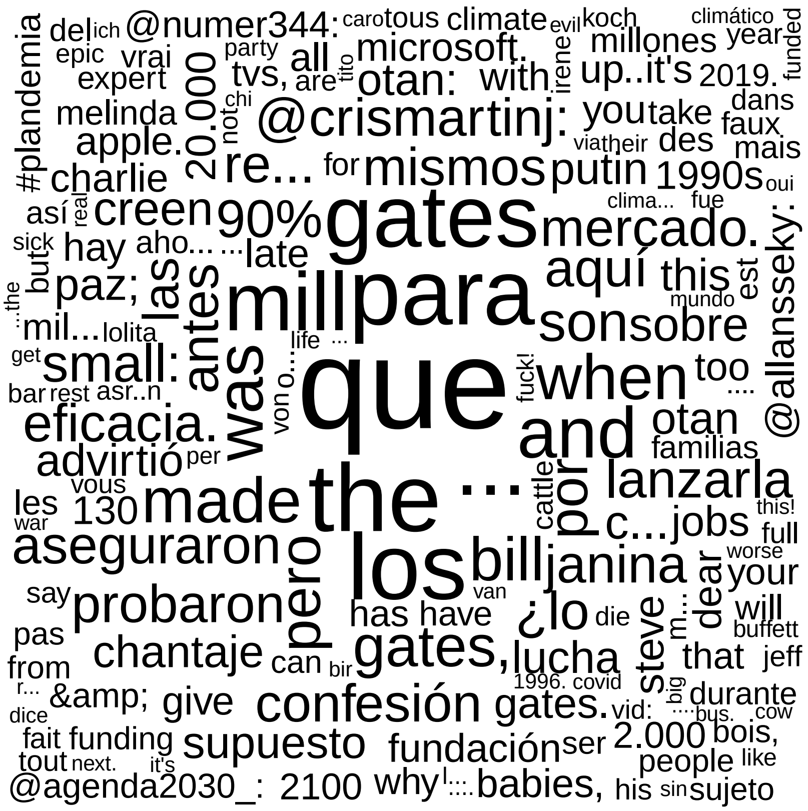
Question 4

Construct a word cloud of the words in your Document Term Matrix by using TF-IDF weights.

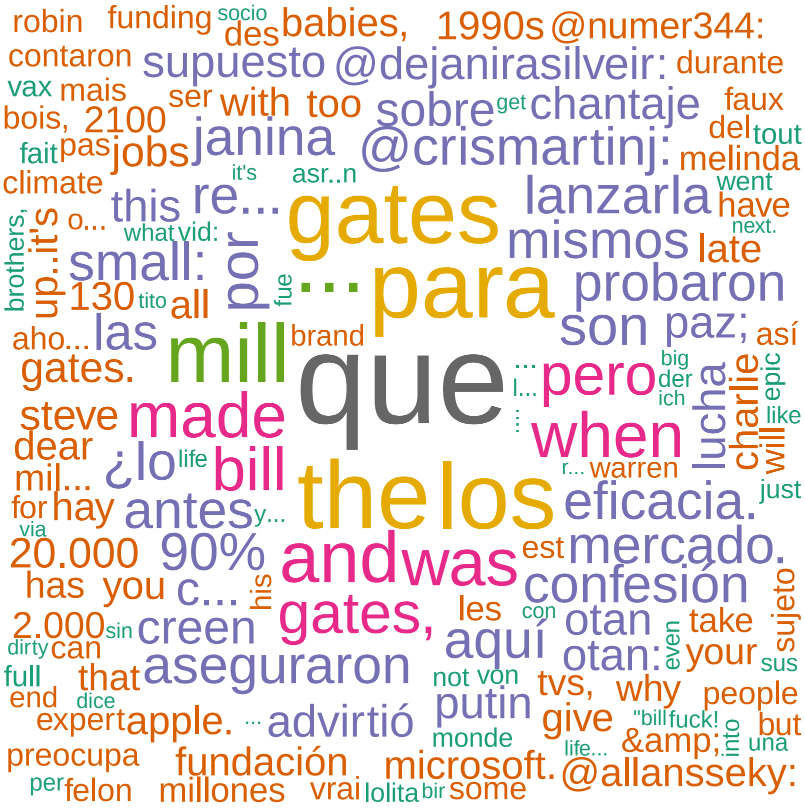
```
# sum tweets term frequencies (remember in DTM, terms are columns)
freqs = colSums(tweet.matrix)

# remove any words that have count "NA" (named vector)
  # as these values are not useful for calculation
freqs = freqs[!is.na(freqs)]

head(names(freqs)) # showing names from frequencies
```

```
## [1] "about"    "acquire." "all"      "and"      "biggest"  "bill"
```

```
# build the word cloud (needs term names and frequencies)
wordcloud(names(freqs), freqs, random.order = FALSE)
```



```r
# modifying the output - with minimum frequency of 3
wordcloud(names(freqs), freqs, random.order = FALSE, min.freq = 3,
          colors = brewer.pal(8, "Dark2"))
```



Intepret your plot.

As a general overview, word clouds are graphical representations of word frequency that give greater prominence to words that appear more frequently in a source text. The larger the word in the visual the more common the word was in the document(s).

Looking at the word cloud diagram, it can be seen that apart from **gates**, which represents our twitter handles last name, **que** and **para** are the two most prominent words. The first word stems from words like question and query to name a few, but is also a multi-functional word that is used in various languages to signify everything from “that”, “which”, “what” or “whom”. The second most frequent word follows a similar approach, where it can be seen that the word cloud is composed of not only English words, but also those originating from various cultures.

Furthermore, looking at the minimum frequency of 3 word cloud, the frequency of words are categorised by colour as well to show the relevance of each word to its particular category. For Bill Gates, prominent words like jobs, foundation, climate and Microsoft represent his company and ambition for the future, as depicted by his tweets and those that tweet about him.

Also, visualising the word cloud, it shows the trends of frequent words and how these are impacted by external events. For example, in the peak of political debates, the frequency of offensive words increased and as well the rise of climate and digital discussion. This is observed within the outer spiral of the word clouds above.

Question 5

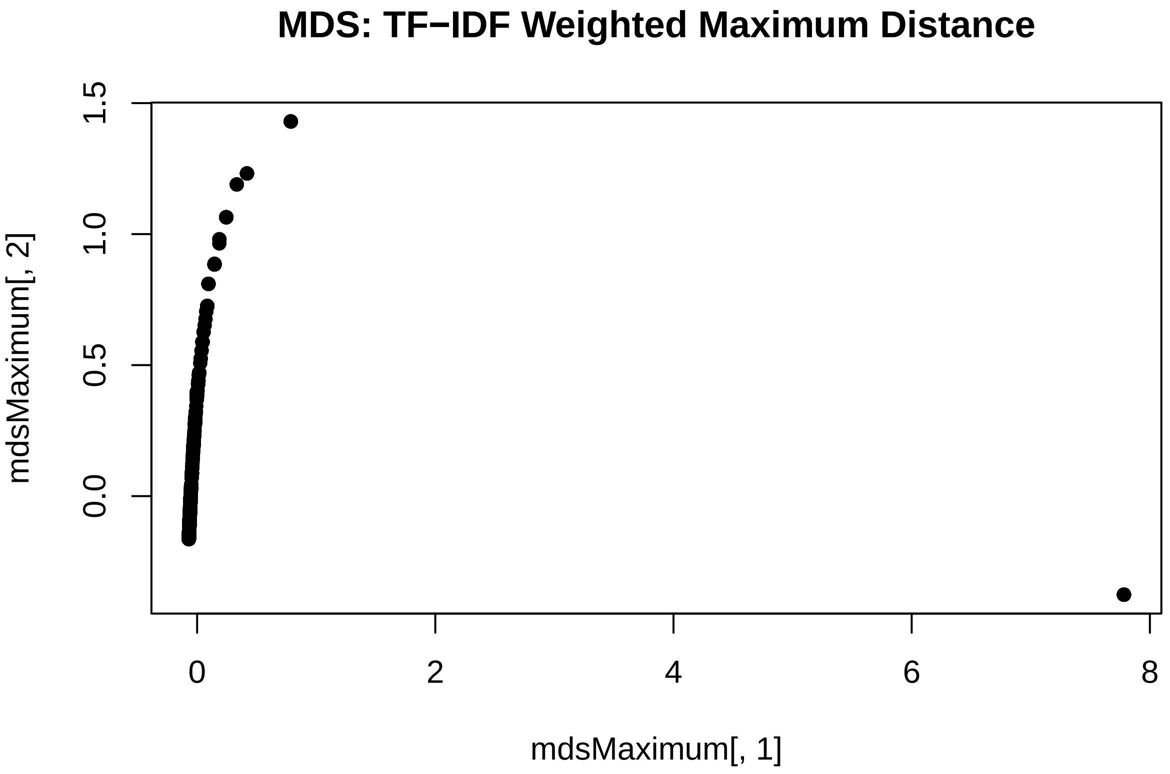
Create two distance matrices of documents using the Maximum and the Manhattan distance.

```
# matrix is already in document term (DTM) form, so no need to transpose

dim(tweet.matrix)[1] # check how many rows/documents we have, to set max bound
```

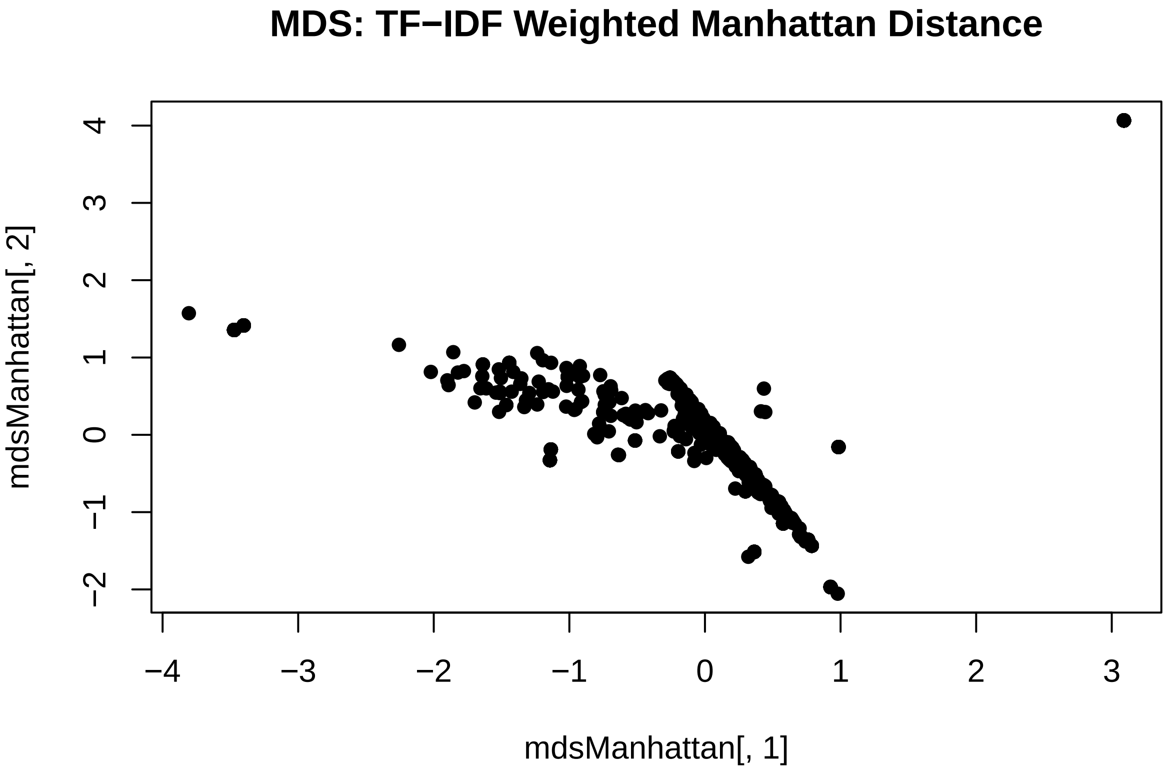
```
## [1] 1431
```

```
# perform MDS using 1000 dimensions (best bet for dimension choice)
  # note, only 1051 of the first 1400 eigenvalues are > 0
```

```
maxD = dist(tweet.matrix, method = "maximum") # maximum distance
mdsMaximum <- cmdscale(maxD, k=1000)

plot(mdsMaximum[, 1], mdsMaximum[, 2], pch = 16,
     main = "MDS: TF-IDF Weighted Maximum Distance")
```



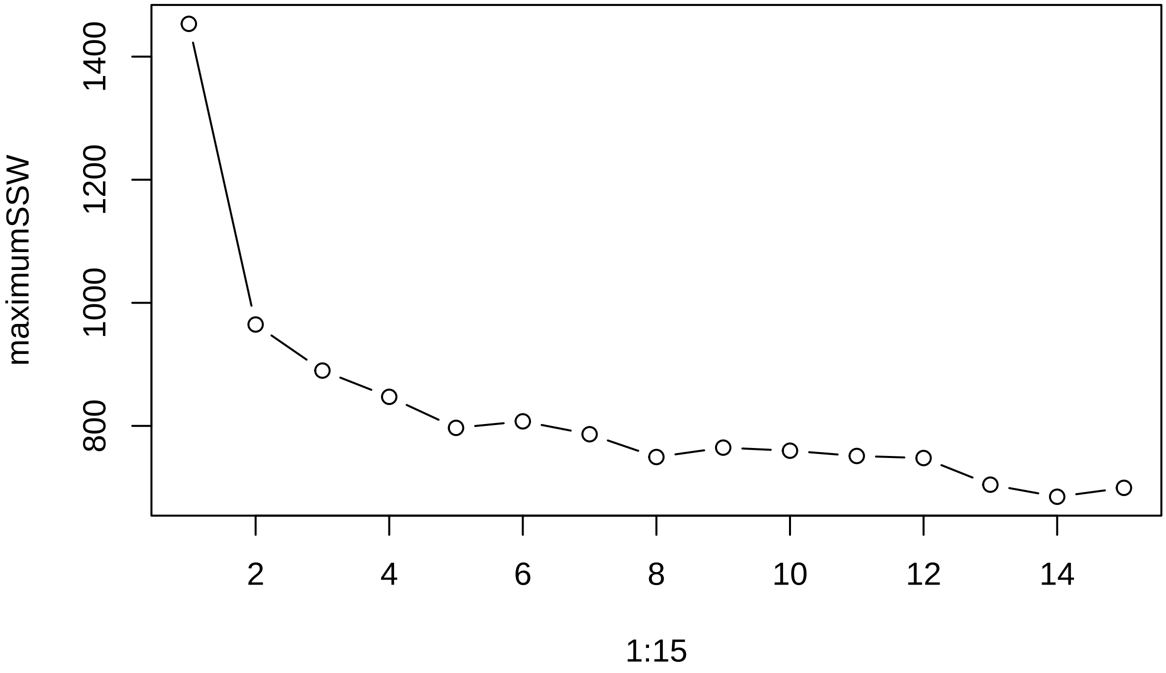
```
manhatD = dist(tweet.matrix, method = "manhattan") # manhattan distance
mdsManhattan <- cmdscale(manhatD, k=1000)

plot(mdsManhattan[, 1], mdsManhattan[, 2], pch = 16,
     main = "MDS: TF-IDF Weighted Manhattan Distance")
```



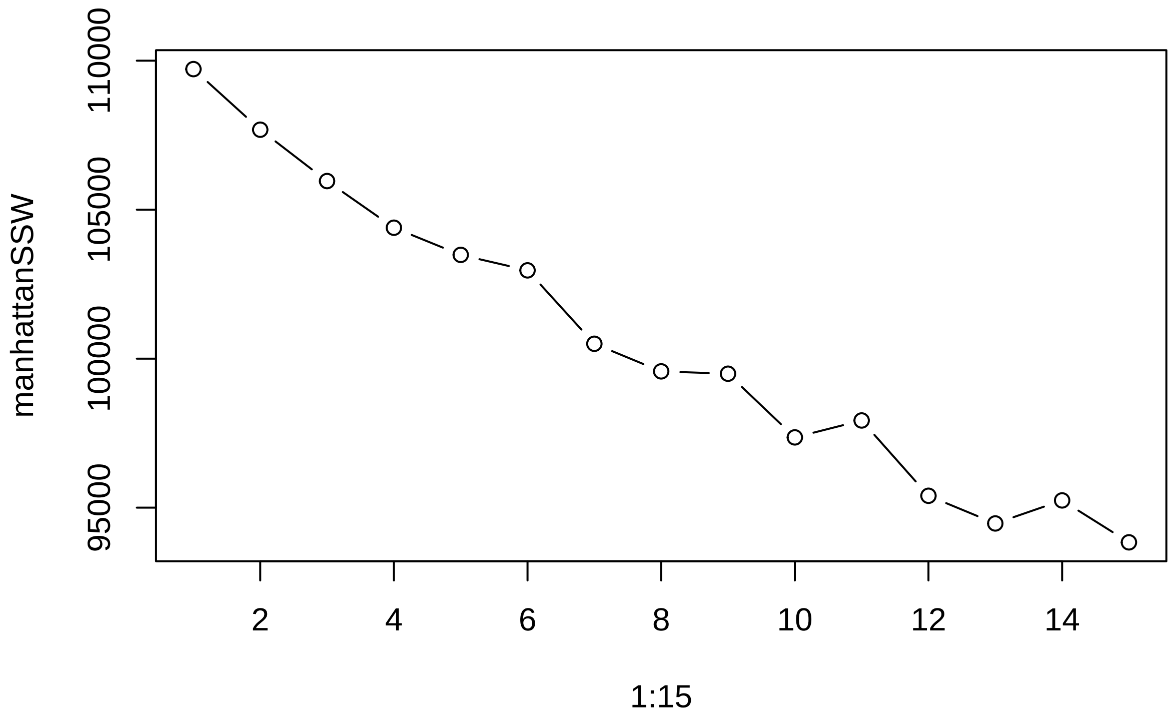
Question 6

Find the most appropriate number of clusters using the elbow method by using the two distance matrices you created above.

```
# clustering for maximum distance matrix

# use the elbow method to identify an appropriate number of clusters,
  # then compute the clusters and store them in K

set.seed(123) # to reproduce a particular sequence of 'random' numbers
n = 15  # starting with 15 clusters
maximumSSW = rep(0, n) # prepare SSW vectors (values to record)

for (a in 1:n) {
  # nstart is the number of random sets in each cluster
  K = kmeans(mdsMaximum, a, nstart = 20, iter.max = 50)
  maximumSSW[a] = K$tot.withinss # total within cluster sum of squares
}

plot(1:15, maximumSSW, type="b") # view the elbow plot

# we can see an elbow at 5 clusters ??
```



```
# clustering for manhattan distance matrix

# use the elbow method to identify an appropriate number of clusters,
  # then compute the clusters and store them in K

set.seed(123) # to reproduce a particular sequence of 'random' numbers
n = 15  # starting with 15 clusters
manhattanSSW = rep(0, n) # prepare SSW vectors (values to record)

for (a in 1:n) {
  # nstart is the number of random sets in each cluster
  K = kmeans(mdsManhattan, a, nstart = 20, iter.max = 50)
  manhattanSSW[a] = K$tot.withinss # total within cluster sum of squares
}

plot(1:15, manhattanSSW, type="b") # view the elbow plot

# We can see an elbow at 8 clusters ??
  # then a further drop indicating that there may be clusters in clusters
  # (a hierarchy of clusters), so we will compute k-means using 8 clusters
```



Question 7

Cluster the documents using k-means clustering.

```
# k-means clustering for maximum distance matrix
maxK = kmeans(mdsMaximum, 5, nstart = 20, iter.max = 50) # compute using 5 clusters
summary(maxK)
```

```
##              Length Class  Mode   
## cluster      1431   -none- numeric
## centers      5000   -none- numeric
## totss           1   -none- numeric
## withinss        5   -none- numeric
## tot.withinss    1   -none- numeric
## betweenss       1   -none- numeric
## size            5   -none- numeric
## iter            1   -none- numeric
## ifault          1   -none- numeric
```

```
table(maxK$cluster)
```

```
## 
##   1   2   3   4   5 
## 943 338 132  10   8 
```

```
# k-means clustering for manhattan distance matrix
manhatK = kmeans(mdsManhattan, 8, nstart = 20, iter.max = 50) # compute using 8 clusters
summary(manhatK)
```

```
##              Length Class  Mode   
## cluster      1431   -none- numeric
## centers      8000   -none- numeric
## totss           1   -none- numeric
## withinss        8   -none- numeric
## tot.withinss    1   -none- numeric
## betweenss       1   -none- numeric
## size            8   -none- numeric
## iter            1   -none- numeric
## ifault          1   -none- numeric
```

```
table(manhatK$cluster)
```

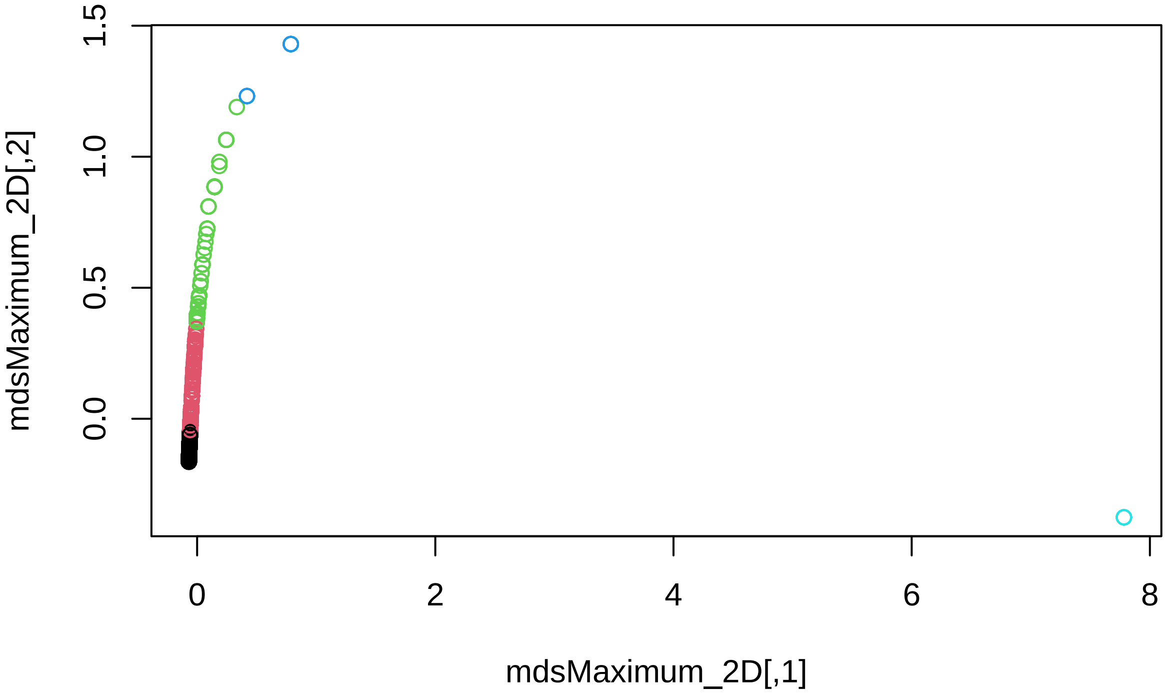
```
## 
##   1   2   3   4   5   6   7   8 
## 971  31  26 104   2 166  38  93 
```

It is to note though that for the different distance matrix clusterings', both have the same SSW and SSB value, where the best number of clusters for each matrix is provided where the reduction of SSW slows (the elbow bend).

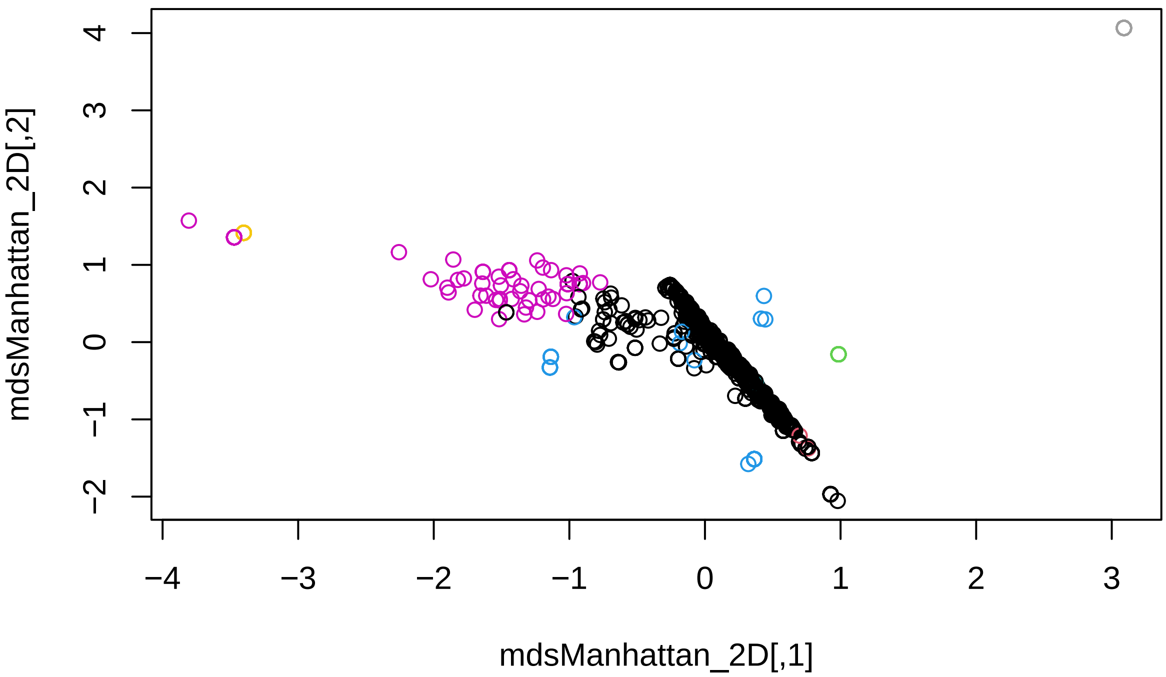
Question 8

Visualise your both clustering results in 2-dimensional vector space.

```
# k-means clustering visualisation for maximum distance matrix

mdsMaximum_2D <- cmdscale(maxD, k=2) # perform MDS using 2 dimensions
plot(mdsMaximum_2D, col = maxK$cluster) # plot and colour points using cluster number
```



```
# k-means clustering visualisation for manhattan distance matrix

mdsManhattan_2D <- cmdscale(manhatD, k=2) # perform MDS using 2 dimensions
plot(mdsManhattan_2D, col = manhatK$cluster) # plot and colour points using cluster number
```



Question 9

Comment on your results.

The **best distance matrix for analysis of the data when visualising on a 2-dimensional space is the manhattan distance**. This is due to the largely spread out data in comparison to the maximum distance which only covers very limited points.

When looking at the k-means clusters for each distance matrix, it is evident that manhattan distance has a better distribution of values, hence attributing to the more accurate spread of data points.

However, in respect to the elbow distance the maximum distance has a better output as the elbow is much easier to visualise in comparison to the manhattan distance where it rather hard to define the elbow.

Question 10

Choose the clustering that you think worked better (from the clusterings you have done above).

Create a dendogram and a word cloud of the words in each cluster and interpret your results/topics for each cluster.

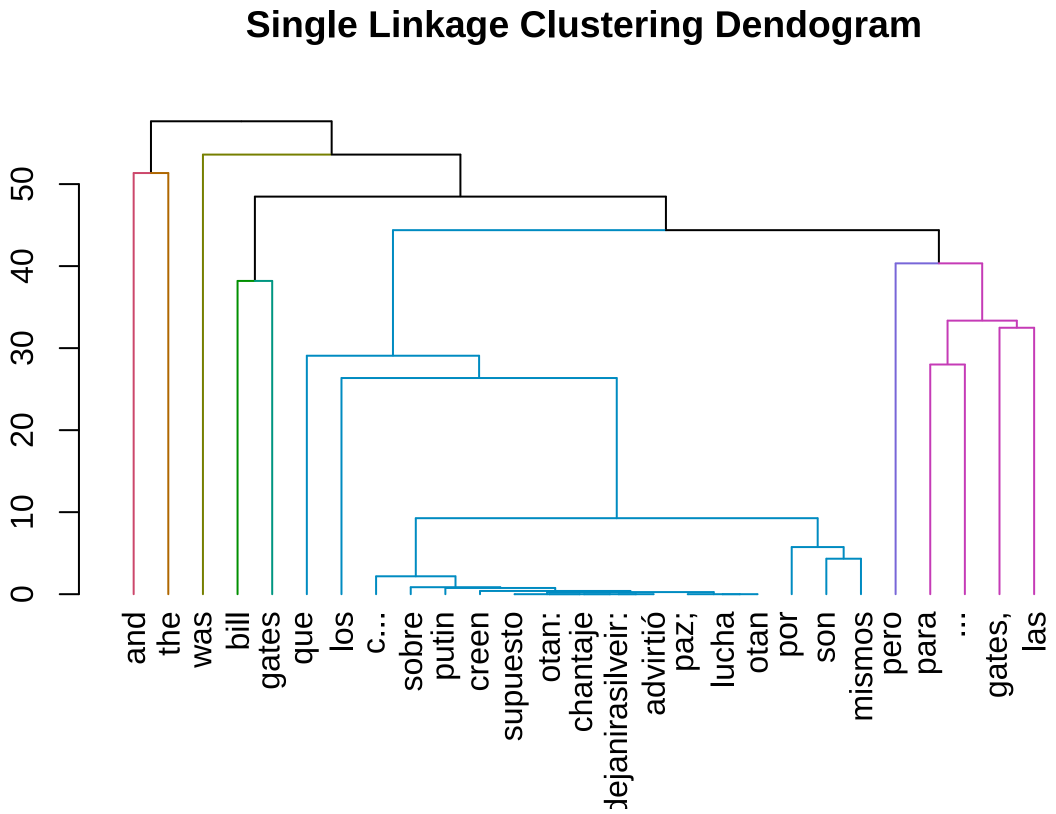
```
# select words that appear most often and examine how they are clustered
  # choose words that appear in at least 100 tweets and store their index
```

```
frequent.words = which(apply(tweet.matrix > 0, 2, sum) > 100)

# extract only those columns from the tweet matrix
term.matrix = tweet.matrix[,frequent.words]

dim(term.matrix) # check new term document matrix size
```

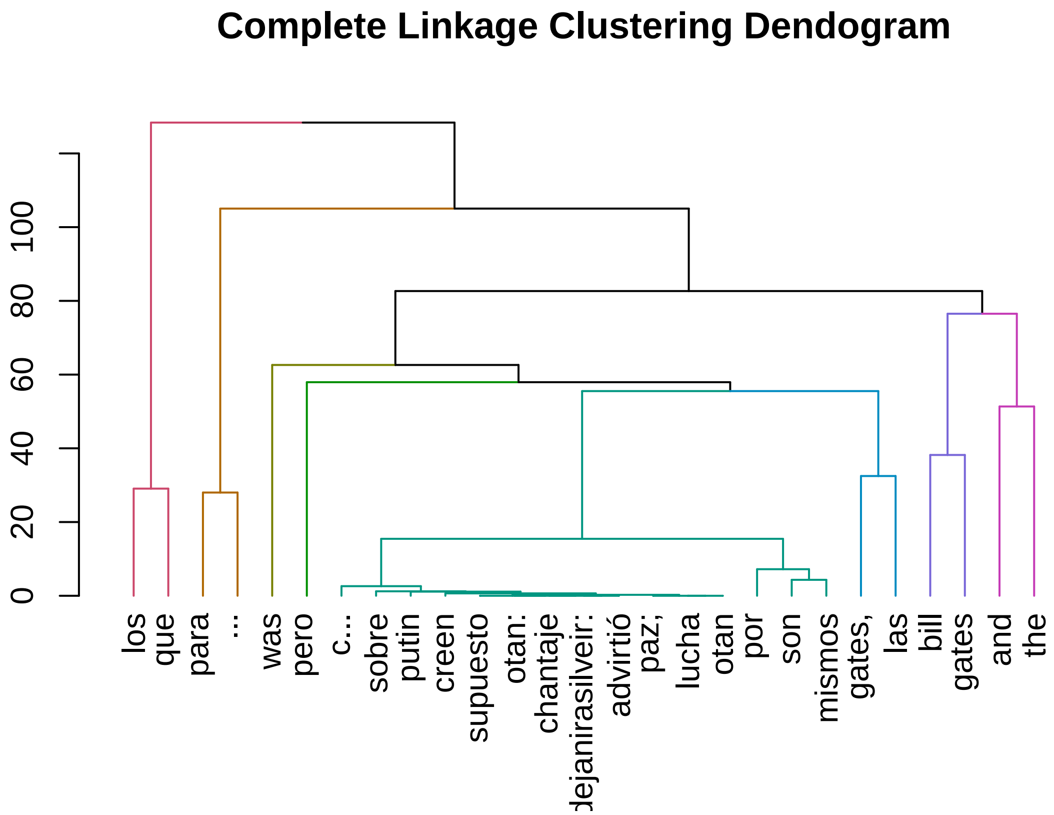
```
## [1] 1431   27
```

```
# compute the distance matrix containing the manhattan distance between
  # all chosen terms, then compute and plot the hierarchical clustering

# transpose matrix as dist() needs DTM as input - terms must be in the rows
D = dist(t(term.matrix), method = "manhattan")
```

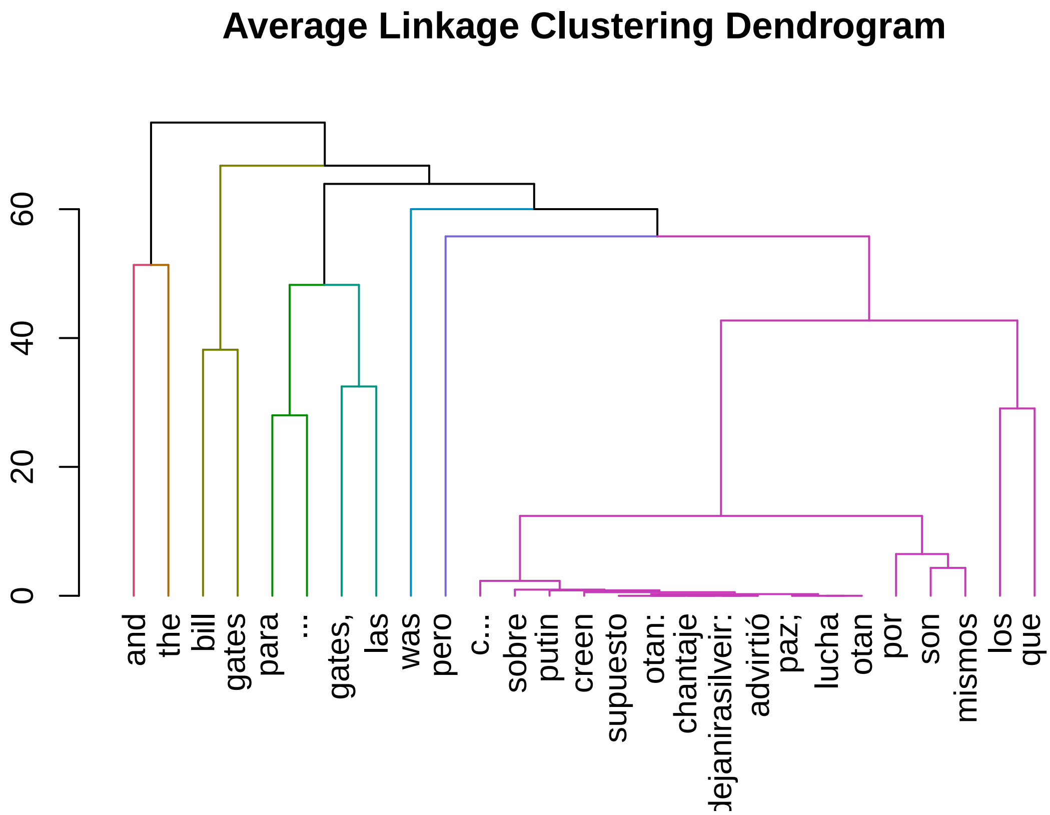
```
singleD = hclust(D, method = "single") # single linkage clustering
sDend = as.dendrogram(singleD) # cast clustering into dendrogram structure
singleClusterColour = color_branches(sDend, k = 8) # colour by cluster
plot(singleClusterColour, main = "Single Linkage Clustering Dendogram")
```



```
completeD = hclust(D, method = "complete") # complete linkage clustering
cDend = as.dendrogram(completeD) # cast clustering into dendrogram structure
completeClusterColour = color_branches(cDend, k = 8) # colour by cluster
plot(completeClusterColour, main = "Complete Linkage Clustering Dendogram")
```



```
averageD = hclust(D, method = "average") # average linkage clustering
aDend = as.dendrogram(averageD) # cast clustering into dendrogram structure
averageClusterColour = color_branches(aDend, k = 8) # colour by cluster
plot(averageClusterColour, main = "Average Linkage Clustering Dendrogram")
```



Interpretation

Dendrograms are branching diagrams that represent the relationships of similarity among a group of objects. The height of the nodes tell us how similar or different the words are from each other. The greater the height, the greater the difference. Furthermore, the closer the path between, the closer the cluster is.

Particularly focusing on the complete and average linkage clustering dendrograms, observation from the plot has shown that the method of hierarchical clustering greatly reduced the dimension of our term matrix. A number of terms like “los”, “que” and “para” are moving up the hierarchy, which indicates that such terms could be closely related to one another, and they can potentially be around one topic that includes Bill Gates.

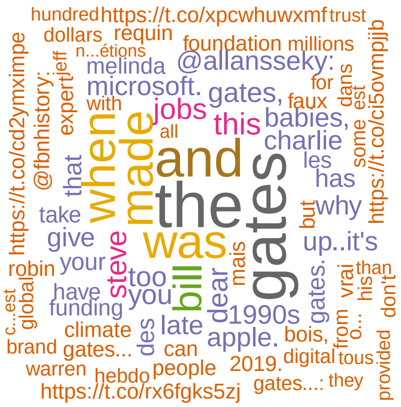
Furthermore, the height values on the dendrogram are evenly spread out as we move along the hierarchy, which could mean that a variety of equivalent words are being used in the tweets; note that the tweets can linguistically differ. This may also explain why a number of terms are based closer to the x-axis, i.e. multiples of nodes have closer connections.

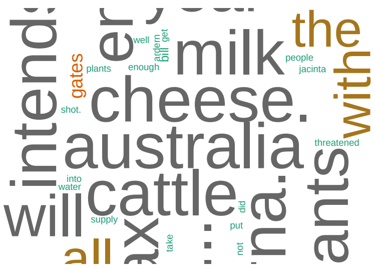
```r
# generate a word cloud of the words in each cluster
table(manhatK$cluster)
```

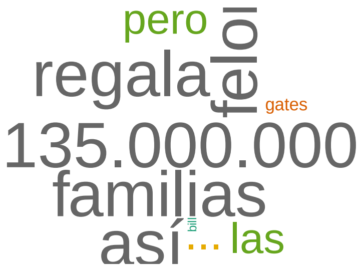
```
##
##   1   2   3   4   5   6   7   8
## 971  31  26 104   2 166  38  93
```

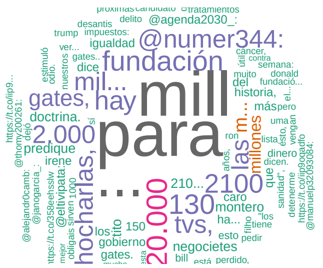
```r
n = 8
```

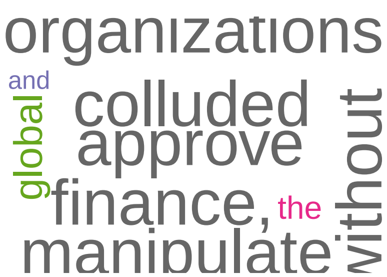
```r
# loop through each cluster, and create a word cloud for each
for (i in 1:n) {
  cluster.number = i # nominate each cluster, and explore it

  # find position of tweets in the manhattan clustering
  clusterTweetsId = which(manhatK$cluster == cluster.number)

  # extract tweets vectors for cluster
  clusterTweets = tweet.matrix[clusterTweetsId,]

  # combine the tweets into a mean tweet
  clusterTermWeight = colMeans(clusterTweets)

  # generate a word cloud to visualise the words in a cluster
  wordcloud(
    words = names(clusterTermWeight),
    freq = clusterTermWeight,
    min.freq = 3,
    max.words = 100,
    random.order = FALSE,
    rot.per = 0.35,
    colors = brewer.pal(8, "Dark2")
  )
}
```

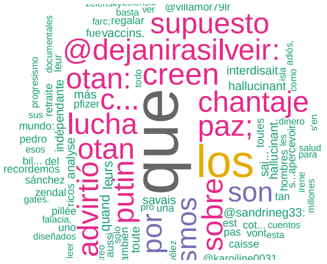


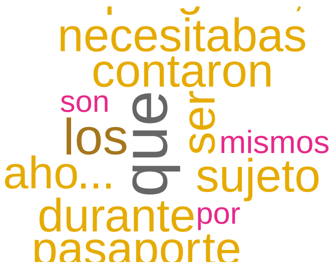


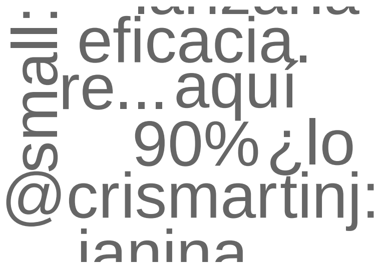












Interpretation

From the word clouds above, it can be seen that 8 plots, each representing a different cluster are provided. The number of elements present within a cluster directly represent the proportion of tweets, and their allocations as obsevered by the word cloud.

Looking across all plots and in accounting for the randomisation of words and their distribution across clusters, the words present within the centre of the plots and large in size are the most prominent in feature. This has shown to be consistent across majority of the previous clusterings' analysis, where words like "gates", "para", "mill" and "que" to name a few, remain as the most frequent terms across most documents.

By taking the minimum frequency value of 3, we can see that words such as "conversation", "leadership", “global” and “data” are colour coded, which highlights Bill Gates personality and traits when it comes to his striving ambitions for his company.

Building Networks

Question 11

Other Twitter users can be mentioned in the text of the tweet. A tweet can contain another account's Twitter username, preceded by the "@" symbol.

Build a **mention** network from the tweets you downloaded and plot a graph to present to AnalysisInMordor trainees.

In order to solve this question you should find all Twitter users mentioned in the text of the tweet and their screen name. Then get the screen names of the tweets that included in the mentions. Create an edge list of "who mentioned whom" and plot the graph.

Note that you should inspect entities object to investigate mentions.

You may need to investigate the *Twitter Data Dictionary* to find which variable contains this information and how to access it.

After building the graph, list the top 8 central users using degree centrality.

In the original data set we collected 1,431 tweets. Some of these are replies to tweets that have been made about Bill Gates, while others are mentioning one or more other users in their own tweets. All of the users who reply to tweets from others, and the users who are mentioned in the collected tweets, are included in the mention network below.

Therefore, we shall construct two edge lists and bind them in one final edge list at the end for visualisation.

Firstly, to construct the 1st edge list, we need to find the screen names of the people who made the collect tweets.

```
users <- users_data(tweets)
head(users$screen_name)
```

```
## [1] "stevoB59"        "nuijten_rob"     "edpoulter"       "Agustinbsas84"
## [5] "trWyA0ol8set22x" "anthmusic"
```

Next, the 1st edge list is constructed.

```
# column bind the above users to create the 1st edge list
el1 = cbind(tweets$in_reply_to_screen_name,users$screen_name)
colnames(el1) = c("mentioner","mentioned")

# since not every tweet is a reply to others, we should remove rows
  # with NA so they are not included in the network
el1 = na.omit(el1)
head(el1)
```

```
##      mentioner         mentioned
## [1,] "MarionKoopmans"  "nuijten_rob"
## [2,] "AllBiteNoBark88" "edpoulter"
## [3,] "KevinDa48962975" "anthmusic"
## [4,] "Afelia"          "ArminKarah"
## [5,] "AlessandraOdri"  "Simone172019"
## [6,] "Monica_Garcia_G" "Enrique_Fosar"
```

Now, each tweet also mentioned further users. Note that multiple users may be mentioned, and their names are stored in Twitter objects which are called tweet entities, which are as below:

```
# tweet 1 did not mention any user
tweets$entities[[1]]$user_mentions$screen_name
```

```
## [1] NA
```

```
# tweet 2 mentioned 3 users
tweets$entities[[2]]$user_mentions$screen_name
```

```
## [1] "MarionKoopmans" "WybrenvanHaga"  "Rob_Roos"
```

Following are the steps that were taken to create the 2nd edge list with all the mentions within the collected tweets:

```
# collect the above mentions in a list
get_mentioned_screennames = function () {
  mentions = list()
  for(i in 1:length(tweets$entities)) {
    mentions[[i]] = tweets$entities[[i]]$user_mentions$screen_name
  }
  # mentions = mentions[!is.na(mentions)]
  # remove NA values (i.e. tweets with no mentions) from the vector
  # mentions = unique(mentions) # return unique mentions only
  return(mentions)
}

mention_list = get_mentioned_screennames()
head(mention_list,3)
```

```
## [[1]]
## [1] NA
##
## [[2]]
## [1] "MarionKoopmans" "WybrenvanHaga"  "Rob_Roos"
##
## [[3]]
## [1] "AllBiteNoBark88"
```

```
a = mention_list
b = users$screen_name
d = c()

# the following loop repeats the tweeter's name for the number of people who they are
  # mentioning and places this in vector d, which is merely a transitional step
for (i in 1:length(a)){
  f = rep(b[i], length(a[[i]]))
  d = append(d, f)
}
```

```
# now we unwind the mention list and bind it with the
  # previous vector to create the 2nd edge list
k = c(unlist(a))
el2 = cbind(d, k)

# and remove NA which does not contribute to the final edge list
el2 = na.omit(el2)
head(el2)
```

```
##      d                 k
## [1,] "nuijten_rob"     "MarionKoopmans"
## [2,] "nuijten_rob"     "WybrenvanHaga"
## [3,] "nuijten_rob"     "Rob_Roos"
## [4,] "edpoulter"       "AllBiteNoBark88"
## [5,] "Agustinbsas84"   "AbogadoDijo"
## [6,] "trWyA0ol8set22x" "woofmelb"
```

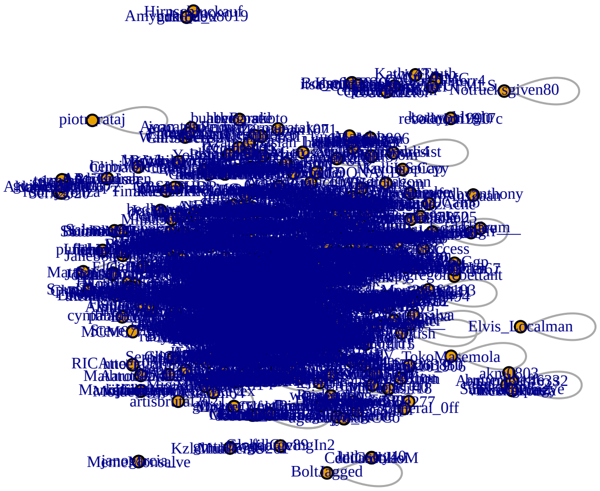
Now that we have the 2nd edge list, do a row bind with the 1st one to collect the final edge list:

```
el = rbind(el1,el2)
head(el)
```

```
##      mentioner         mentioned
## [1,] "MarionKoopmans"  "nuijten_rob"
## [2,] "AllBiteNoBark88" "edpoulter"
## [3,] "KevinDa48962975" "anthmusic"
## [4,] "Afelia"          "ArminKarah"
## [5,] "AlessandraOdri"  "Simone172019"
## [6,] "Monica_Garcia_G" "Enrique_Fosar"
```

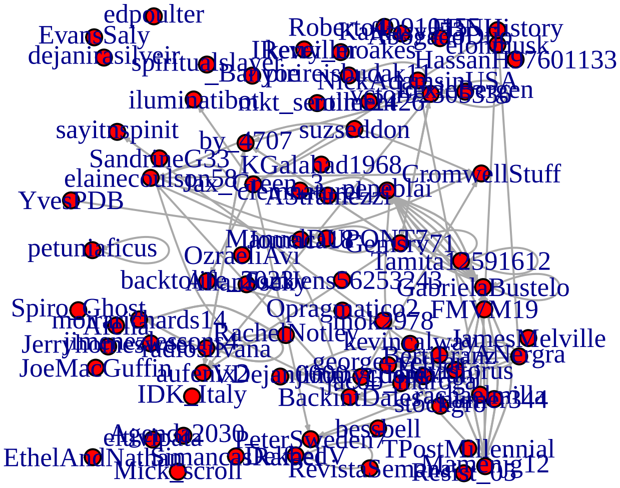
Use the final edge list to visualise the mention network.

```r
g = graph.edgelist(el)
plot(g,layout=layout.fruchterman.reingold,vertex.size = 5,
     edge.arrow.size=0.1, vertex.label.cex=0.5)
```



Note that the original graph is dense and therefore reduced multiple times to obtain the final mention network:

```
g4 = induced_subgraph(g, which(degree(g, mode = "all") > 5))
plot(g4, layout = layout.random, vertex.size = 7,
     edge.arrow.size=0.15, vertex.color="red",vertex.label.cex=0.8)
```



Now we calculate the degree centrality scores of the graph to find the top 8 users who tweeted the most about Bill Gates.

```
# calculate the top 8 users
centralIDs = order(degree(g),decreasing = TRUE)[1:8]
V(g)[centralIDs]
```

```
## + 8/1762 vertices, named, from 5008876:
## [1] dejanirasilveir crismartinj     numer344        AllanSseky
## [5] Agenda2030_     backtolife_2023 VDejan0000      bessbell
```

```
degree(g)[centralIDs]
```

```
## dejanirasilveir     crismartinj        numer344      AllanSseky     Agenda2030_
##             109              95              77              70              50
## backtolife_2023      VDejan0000        bessbell
##              33              32              27
```

The top 8 users are, in decreasing order of degree centrality scores: **dejanirasilveir, crismartinj, numer344, AllanSseky, Agenda2030_, backtolife_2023, VDejan0000, bessbell**.

Question 12

Interpret your results (the graph and the centrality).

The graph contains useful information regarding the public figure that we have chosen, Bill Gates. The reduced graph shows a number of edges but also multiple isolated vertices. This means that while we have a network of people having live conversations over Twitter indicated by the edges on the graph, several others are also tweeting actively by mentioning offline users. This is shown through the isolated vertices; one of such user is **@numer344** who has the 3rd highest degree centrality scores.

In the 1,762 users in the mention network, the score of each tweeter in the mention network also vary greatly at multiple folds: the user with the highest score, 109, is dejanirasilveir; meanwhile, the 8th top user bessbell has a score of 27, almost 5 times lower than dejanirasilveir. This could mean that there is a particularly close-knitted community to this user that is discussing about Bill Gates at the time that the tweets were collected. With high degree centrality scores, the top 8 central users (or nodes) have great influence on the topic of interest and it is perceived that these users may be close connections to Bill in his network.

However, interestingly at the time the tweets are collected, Bill Gates only has a degree centrality score of 4. An interpretation of this could be that he was not part of the conversations that were then taking place.

Time Series

Question 13

Download the last 550 posts of the Twitter handle you selected.

Draw a plot of the dates and the number of posts the user posted. Then test whether there is a linear relationship between the date and the number of posts.

For this question we have downloaded a separate data set of tweets from only Bill Gates, our chosen public figure, and saved it in a variable named "comp3020.dataset_timeline."

This is done in the below code chunk but the output is not printed. Instead, the **saved data file is loaded for this time series question and will be used** for the analysis.

```r
# record your authentication keys to create token as an environment variable
app = ""
key = ""
secret = ""
access_token = ""
access_secret = ""

# perform OAuth authentication
twitter_token = create_token(app, key, secret, access_token, access_secret,
                             set_renv=FALSE)

# retrieve 1,600 raw tweets and save object
tweets = get_timeline("BillGates", n = 1600, token = twitter_token,
                      include_rts = FALSE)

save(tweets, file="comp3020.dataset_timeline.RData")
```

```r
load("comp3020.dataset_timeline.RData")

# in the above timeline, we extract the latest 550 posts
tweets550 = tweets[1:550,]
```

Now since we are only using the 1st column of the above data set to construct both the x-axis and y-axis, there are several tasks to be done:

- Collect all the dates when Bill Gates made his latest 550 posts for the x-axis
- Count the number of tweets for the y-axis

Below are the first few rows of the series that we are extracting from:

```r
head(tweets550$created_at)
```

```
## [1] "2022-09-30 14:40:09 AEST" "2022-09-30 02:57:54 AEST"
## [3] "2022-09-29 07:46:50 AEST" "2022-09-23 07:56:22 AEST"
## [5] "2022-09-23 07:11:23 AEST" "2022-09-22 10:26:42 AEST"
```

```r
# obtain dates from the 1st column when the tweets are made
dates = sort(format(tweets550$created_at,'%Y-%m-%d'), decreasing=FALSE)
head(dates)
```

```
## [1] "2021-02-18" "2021-02-18" "2021-02-19" "2021-02-19" "2021-02-20"
## [6] "2021-02-20"
```

```
length(unique(dates))
```

```
## [1] 319
```

In this tweet collection Bill's first tweet was made on 2021-02-18. The last date was when we collected the tweets, 2022-09-30. This is a span of 590 days, even though Bill actually made his tweets on only 319 days out of 590.

Following is the code that we have used to wrangle the correct counts of tweets, by collecting values for the x-axis and y-axis in a single data frame.

```
# c is a temporary data frame that summarises all the tweet counts by dates
# note that the 550 collected tweets are unevenly spread across the collected time frame
  # so c would contain tweet counts for unique dates only
c = as.data.frame(dates) %>% group_by(dates) %>% summarise(counts = n())
c$dates <- as.Date(c$dates)

head(c) # is a 319-row data frame
```

```
## # A tibble: 6 x 2
##   dates      counts
##   <date>      <int>
## 1 2021-02-18      2
## 2 2021-02-19      2
## 3 2021-02-20      3
## 4 2021-02-21      3
## 5 2021-02-24      1
## 6 2021-02-25      5
```

```
# s is a temporary empty data frame that contains every
  # single date from 2021-02-18 and 2022-09-30
x.axis = c(seq(as.Date("2021-02-18"), as.Date("2022-09-30"), by="days"))
s = cbind.data.frame(
  dates = x.axis,
  counts = rep(0,length(x.axis))
)

head(s) # is a 590-row data frame
```

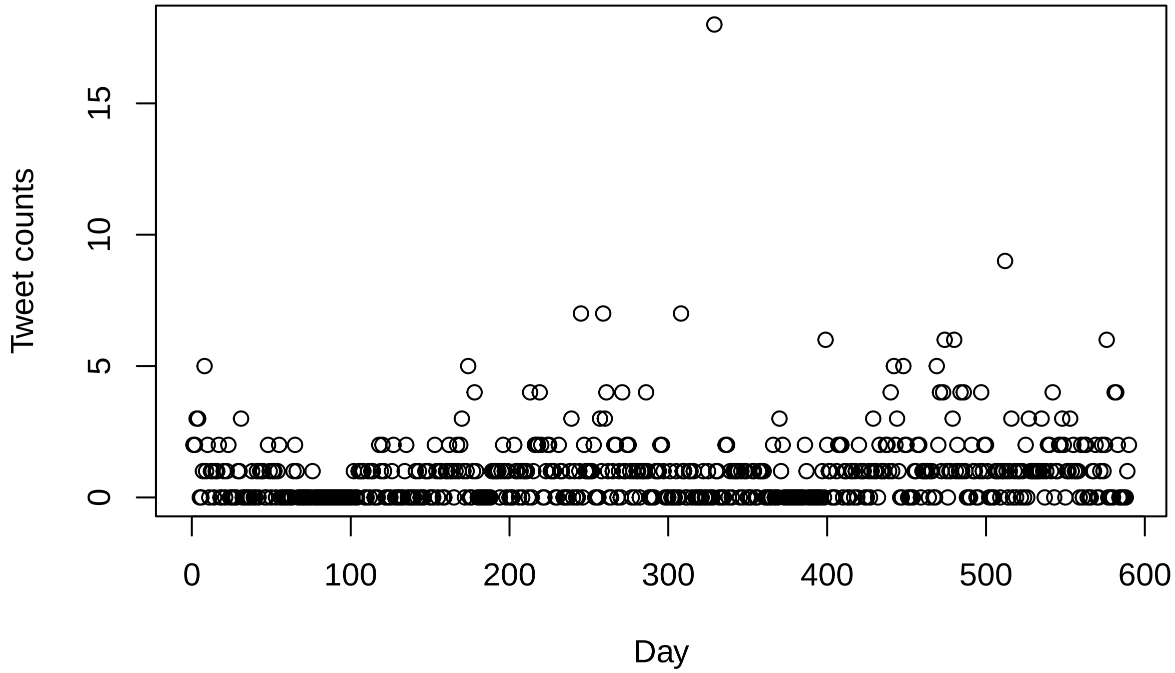
```
##        dates counts
## 1 2021-02-18      0
## 2 2021-02-19      0
## 3 2021-02-20      0
## 4 2021-02-21      0
## 5 2021-02-22      0
## 6 2021-02-23      0
```

```
# do a full-join between s and c to obtain the plotting data set, df
# note that previously, c$dates has been transformed into the corresponding
  # format to s$dates so merge() should work without errors
df <- merge(s, c, all = TRUE, by = "dates")
```

```
# drop unnecessary 2nd column and replace NA's in the 3rd column with 0's
df <- df[,-2]
df[is.na(df)] <- 0

# rename accordingly
colnames(df) = c("dates","counts")

# create a new column "day", for plotting
df <- df %>% mutate(day = row_number())

head(df)
```

```
##         dates counts day
## 1 2021-02-18      2   1
## 2 2021-02-19      2   2
## 3 2021-02-20      3   3
## 4 2021-02-21      3   4
## 5 2021-02-22      0   5
## 6 2021-02-23      0   6
```

```
# time-series plot of tweet counts
plot(df$day, df$counts,
     xlab = "Day", ylab = "Tweet counts")
```



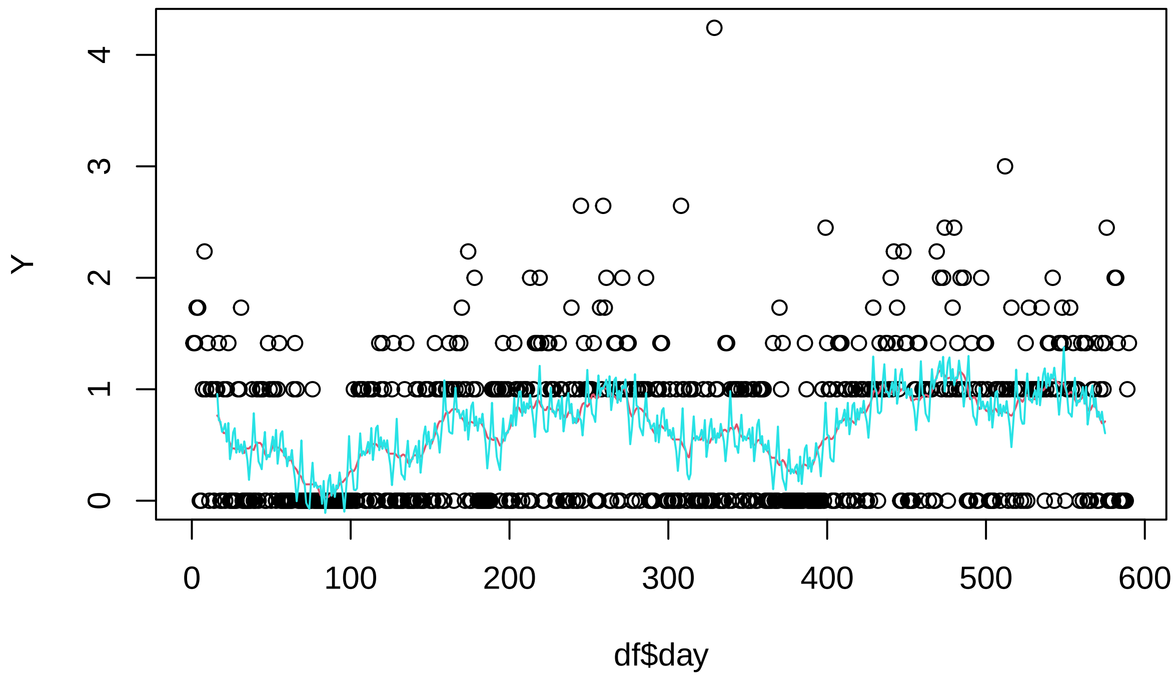
Some extra analysis...

Despite the question not stating anything about performing trend and seasonal analysis, we have done so for the purpose of our analysis. This is not necessarily needed to answer the question.

However, our question was: given we need to plot by dates, will moving averages and trend matter?

```
# initially, this looks like the data is clustered at the bottom of the plot
# we shall transform the y-values to make it easier to observe
# the following three functions are used to calculate the trend line and seasonal line

windowWeights = function(m) {
  if( m%%2 ) {
    ## m is odd
    w = rep(1,m)/m
  } else {
    ## m is even
    ## compute w for an even sized window
    w = c(0.5,rep(1,m-1),0.5)/m
  }
  return(w)
}

moving.average = function(x, m) {
  ## compute window weights
  w = windowWeights(m)
  j = floor(m/2)
  offsets = (-j):j
  n = length(x)
  res = rep(NA, n)
  ## slide window and apply window weights to obtain averages
  for(i in (j+1):(n-j)) {
    res[i] = sum(w*x[i+offsets])
  }
  return(res)
}

Y = sqrt(df$counts)
trend = moving.average(Y, 30)

centred = sqrt(df$counts) - trend
zcentred = centred[!is.na(centred)]
seasonal = function(x ,m) {
  ## extend the data so it is a multiple of m long
  tmp = c(rep(NA, m - length(x)%%m),x)
  ## convert to a matrix
  mat = matrix(tmp, nrow=m)
  ## Calculate the row means to get the seasonal component (excluding missing entries)
  seas = rowMeans(mat, na.rm=TRUE)
  seas = seas - mean(seas)
  return(seas)
}

season = seasonal(zcentred,30)
```

```
### Final plot with time-series data, and the trend/seasonal lines
plot(df$day, Y)
lines(df$day,trend,col=2)
lines(df$day[!is.na(centred)],trend[!is.na(centred)] +
        rep_len(season,length.out=sum(!is.na(centred))),col=5)
```



Now we shall do a hypothesis test to see whether there is a linear relationship between the date and the number of posts.

- H_0 **(null hypothesis** = there is no linear relationship between the date and the number of posts
- H_1 **(alternative hypothesis** = there is a linear relationship between the date and the number of posts

```
m = lm(df$counts~df$day)
summary(m)
```

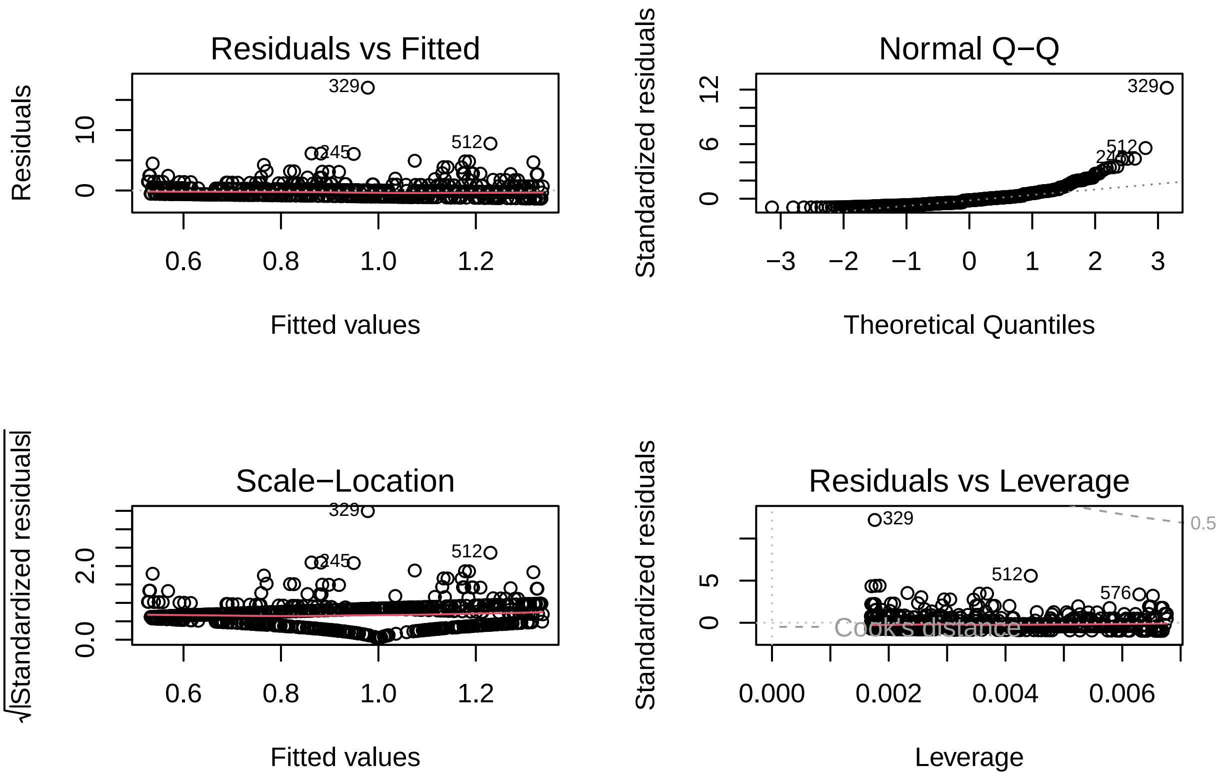
```
##
## Call:
## lm(formula = df$counts ~ df$day)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3346 -0.8307 -0.2438  0.3062 17.0217
##
## Coefficients:
```

```
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.5256482  0.1151407   4.565 6.08e-06 ***
## df$day      0.0013758  0.0003376   4.075 5.22e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.397 on 588 degrees of freedom
## Multiple R-squared:  0.02747,    Adjusted R-squared:  0.02582
## F-statistic: 16.61 on 1 and 588 DF,  p-value: 5.224e-05
```

The p-value is less 0.05 so we may reject the null hypothesis. So we conclude there is evidence of a linear relationship between the date and the latest 550 posts that Bill Gates has made.

Note, a 4x4 plot of the model can be found below:

```
par(mfrow=c(2,2))
plot(m)
```

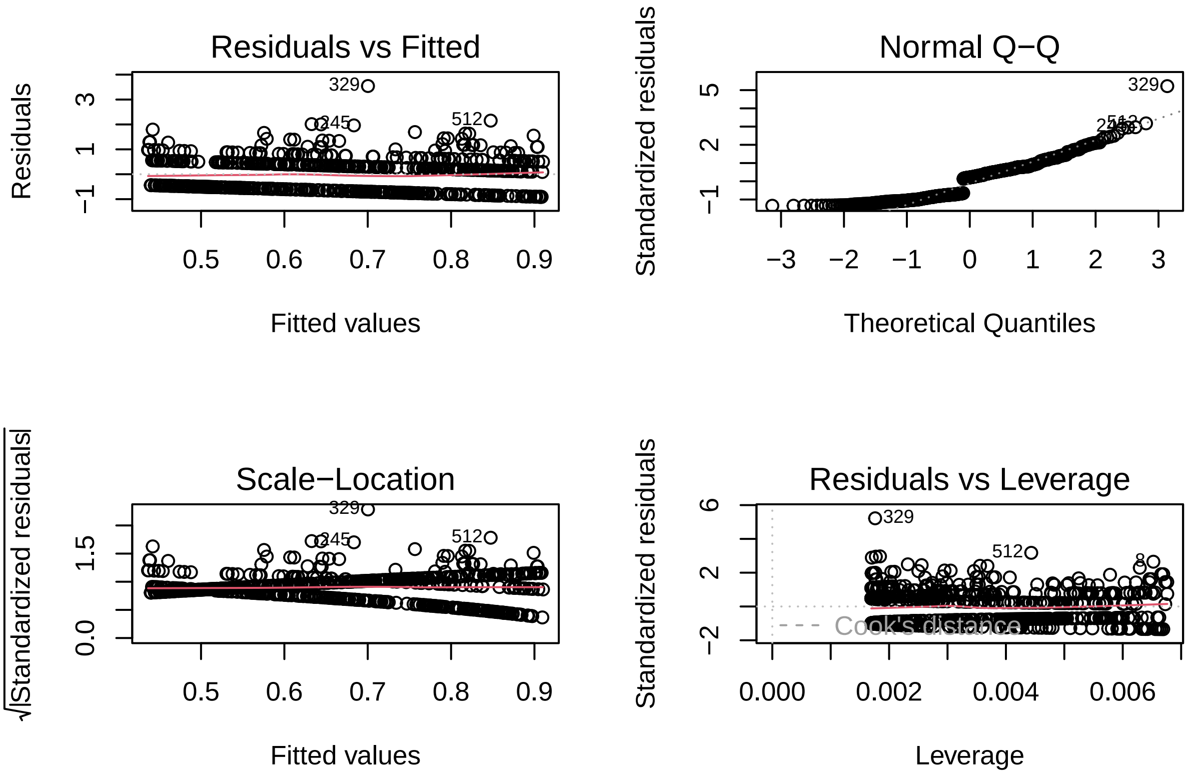


By plotting, we can see that the data is not normally distributed. Let's try a different model with transformed y-values:

```
m1 = lm(sqrt(df$counts)~df$day)
summary(m1)
```

```
##
## Call:
## lm(formula = sqrt(df$counts) ~ df$day)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9087 -0.6140  0.1444  0.4660  3.5423
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.4356263  0.0560121   7.777 3.34e-14 ***
## df$day      0.0008045  0.0001642   4.899 1.25e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6794 on 588 degrees of freedom
## Multiple R-squared:  0.03921,    Adjusted R-squared:  0.03758
## F-statistic:    24 on 1 and 588 DF,  p-value: 1.248e-06
```

```
par(mfrow=c(2,2))
plot(m1)
```



Now the data is more normally distributed.

Interpret your results.

From observation of the time-series plot, it is seen that Bill is not a frequent tweeter since his most recent 550 posts date back up to 590 days ago. On average, this is less than 1 tweet per day that Bill is making.

Both models `m` and `m1` explain less than 5% of the variations of the data. Therefore, in terms of prediction, these might not be the best models to be used.