Social Web Assignment1

Group 7

2023-10-11

Group member information:

Group 7

- Group 7 Leader: Long Nguyen
- Team member: 4

Names - Student ID - Contribution (%)

- Long Nguyen (Peter Nguyen) - - 25%
- Quang Dong Nguyen - 20744696 - 25%
- Veronica Pham (Van Pham) - - 25%
- Yelina Nguyen (Yen Nguyen) - - 25%

Declaration

By including this statement, we the authors of this work, verify that:

- We hold a copy of this assignment that we can produce if the original is lost or damaged.
- We hereby certify that no part of this assignment/product has been copied form any other student's work or from any other source except where due acknowledgement is made in the assignment.
- No part of this assignment/product has been written/produced for us by another person except where such collaboration has been authorised by the subject lecturer/ tutor concerned.
- We are aware that this work may be reproduced and submitted to plagiarism detection software programs to detect possible plagiarism **(which may retain a copy on its database for future plagiarism checking)**
- We hereby certify that we have read and understood what the School of Computing and Mathematics defines as minor and substantial breaches of misconduct as outlined in the learning guide for this unit.

Firstly, we import libraries that will be used in the assignment.

```
library(RedditExtractoR) #Use to extract Reddit threads and more reddit related stuff
library(tm) # text mining library
library(wordcloud) #word cloud library
library(igraph)
```

Question 1

Part1: Using Reddit API, identify the relevant thread URL's for your chosen topic. Focus on either weekly or monthly timeframe.

Using library `RedditExtractor`, we extract relevant threads that contain our chosen topic, tesla, set the focus time as monthly, and store it as `tesla_thread.RDS` file (Due to the constant altercation of information in the social web, we will try to obtain information once and store it in local computer, this will prevent changing in data information).

```
#tesla_thread_url <- find_thread_urls(keywords = 'tesla',
#                                     sort_by = 'comments',
#                                     period = 'month')
#saveRDS(tesla_thread_url, 'tesla_threads.RDS')
tesla_threads <- readRDS('tesla_threads.RDS') #read the RDS file and store value in tesla_thread
tesla_threads2 <- readRDS('tesla_gaming.RDS')
```

Part 2: Find the top 3 threads with the highest number of comments

We select the top 3 threads with the highest number of comments from the collection of threads in the previous section.

```
#Part 2:
highest_comments <- order(tesla_threads$comments,
                          decreasing = TRUE)[1:3] #select the top 3 threads with highest comments
thread_highest_comments <- tesla_threads[highest_comments,]
```

Therefore, the top 3 threads with the highest counts of comments are stored in `thread_highest_comments`.

Part 3: Retrieve and display the main post from each of these 3 threads. Generate a word cloud for the comments and replies within each of these threads, resulting in a word cloud for each thread.

Due to the same reason as in part 1, we will save the contents collected into RDS files so that we could recall the data from the local computer in anytime.

```
#Part 3:
#thread_contents1 <- get_thread_content(thread_highest_comments$url[1]) #type = list
#thread_contents2 <- get_thread_content(thread_highest_comments$url[2])
#thread_contents3 <- get_thread_content(thread_highest_comments$url[3])
#saveRDS(thread_contents1, 'thread_contents1.RDS')
#saveRDS(thread_contents2, 'thread_contents2.RDS')
#saveRDS(thread_contents3, 'thread_contents3.RDS')
```

Now we recall the data information through the command `readRDS`

```
thread_contents1 <- readRDS('thread_contents1.RDS')
thread_contents2 <- readRDS('thread_contents2.RDS')
thread_contents3 <- readRDS('thread_contents3.RDS')
```

Thread 1 Word Cloud:

Create a term document matrix for thread 1:

First, from library `tm`, we use `Corpus` to create a collection of texts comments. some of the document will be dropped along the way when converting texts into the standard ASCII format.

```
corpus <- Corpus(VectorSource(thread_contents1$comments$comment))
corpus <- tm_map(corpus,
                 function(x) iconv(x, to = 'ASCII'))
```

```
## Warning in tm_map.SimpleCorpus(corpus, function(x) iconv(x, to = "ASCII")):
## transformation drops documents
```

```
#convert it to the standard ASCII, it will drop document
```

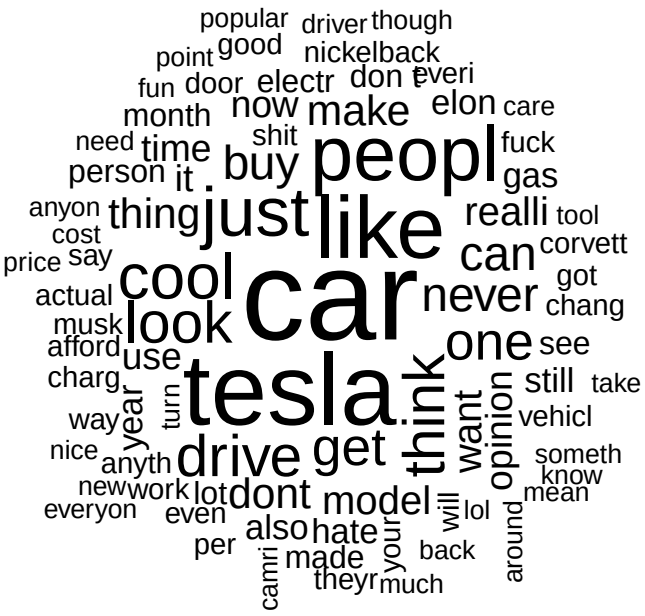
We then move on to create a term document matrix

```
#Term document Matrix for thread 1
tdm <- TermDocumentMatrix(corpus,
                          control = list(removePunctuation = TRUE,
                                         stopwords = TRUE,
                                         removeNumbers = TRUE,
                                         tolower = TRUE,
                                         stemming = TRUE))

tdm <- tdm[,-which(colSums(as.matrix(tdm)) == 0)] #remove empty columns
M <- as.matrix(tdm)
cat('The total number of terms appeared: ', nrow(M),
    '\nThe total number of doccuments available: ', ncol(M))
```

```
## The total number of terms appeared:  2035
## The total number of doccuments available:  491
```

Now, from the result above, we create a word cloud.

```
freqs <- rowSums(M)

wordcloud(names(freqs),
          freqs,
          random.order = FALSE,
          max.words = 85)
```



Thread 1 Word Cloud (using weighted TF-IDF):

Create a weighted TF - IDF term document matrix for thread #1.

```
set.seed(1)
#Create a weighted TF-IDF Word Cloud for thread #1
tdm <- TermDocumentMatrix(corpus,
                          control = list(removePunctuation = TRUE,
                                         stopwords = TRUE,
                                         removeNumbers = TRUE,
                                         tolower = TRUE,
                                         stemming = TRUE))
tdmw <- weightTfIdf(tdm) #weighted term document
```

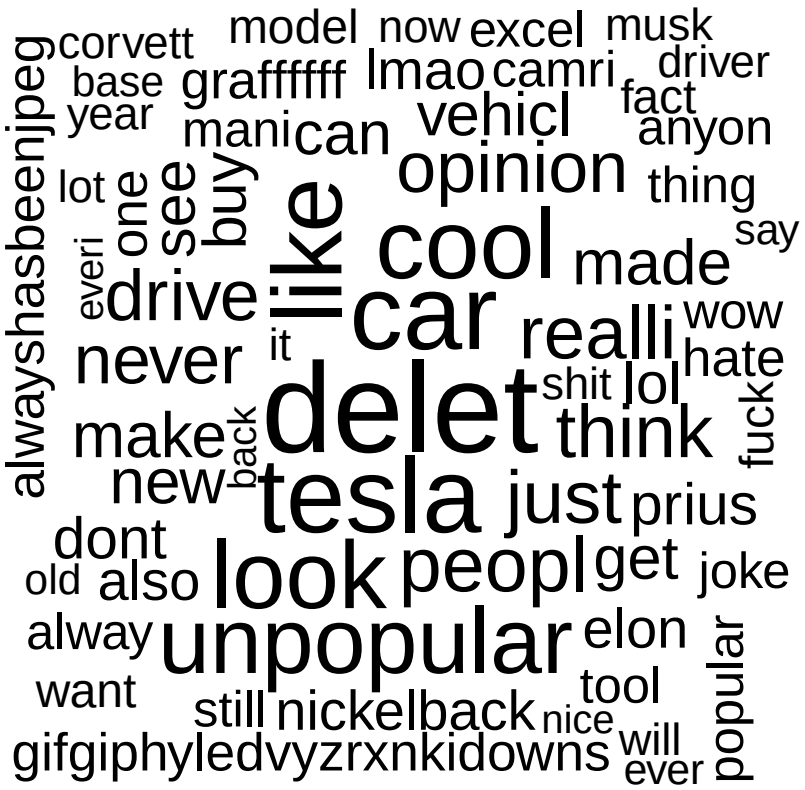
```
## Warning in weightTfIdf(tdm): empty document(s): 12 132 219 416
```

```
T <- as.matrix(tdmw) #weighted term document matrix
```

Then we create a word cloud using the weighted Term Document Matrix.

```
freqsw <- rowSums(T)

##Remove words that have 'NA'.
freqsw <- freqsw[!is.na(freqsw)]
```

```
wordcloud(names(freqsw),
          freqsw,
          random.order = FALSE,
          max.words = 85) #wordcloud creation
```



Thread 2 Word Cloud:

We follow the same steps as when creating a word cloud for thread 1.

- The code section below is for creating a Term Document Matrix for thread #2.

```
#create a term-document matrix for thread #2
corpus <- Corpus(VectorSource(thread_contents2$comments$comment))
corpus <- tm_map(corpus, function(x) iconv(x, to = 'ASCII')) #convert it to the standard ASCII, it will
```

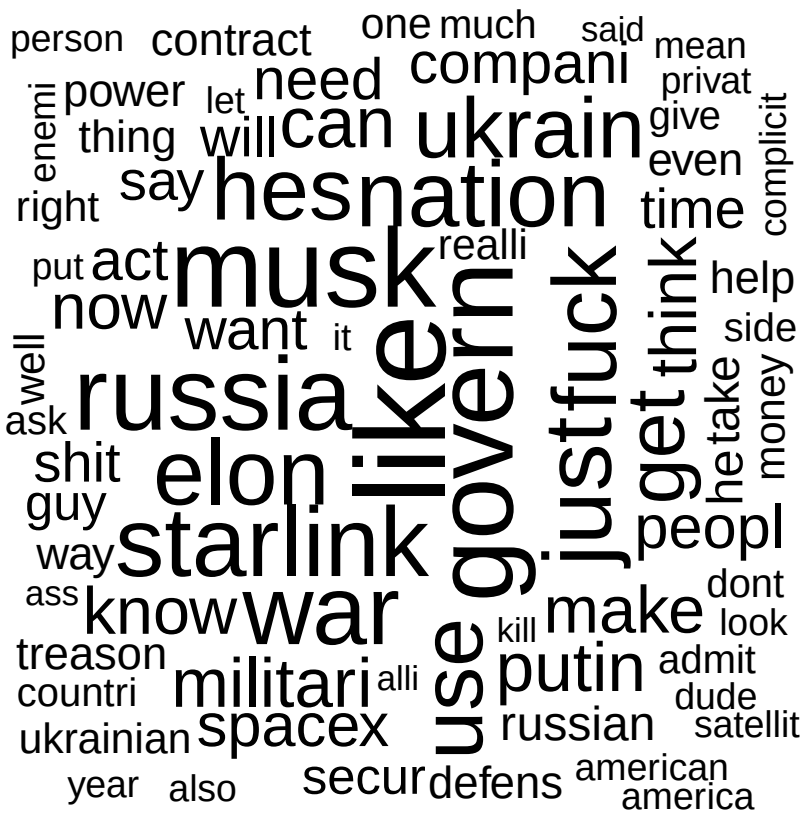
```
## Warning in tm_map.SimpleCorpus(corpus, function(x) iconv(x, to = "ASCII")):
## transformation drops documents
```

```
set.seed(1)
#tdm
tdm <- TermDocumentMatrix(corpus,
                          control = list(removePunctuation = TRUE,
                                         stopwords = TRUE,
```

```
                                        removeNumbers = TRUE,
                                        tolower = TRUE,
                                        stemming = TRUE))

empties <- which(colSums(as.matrix(tdm)) == 0)
tdm <- tdm[,-empties] #remove empty columns
M <- as.matrix(tdm)
```

Then we create a word cloud:

```
freqs <- rowSums(M)

#Create a word cloud for thread #2
wordcloud(names(freqs),
          freqs,
          random.order = FALSE,
          max.words = 85)
```



Thread 2 Word Cloud (using weighted TF-IDF):

```
set.seed(1)
#Create a TF-IDF Word Cloud for thread #2
tdm <- TermDocumentMatrix(corpus,
```

```
                          control = list(removePunctuation = TRUE,
                                         stopwords = TRUE,
                                         removeNumbers = TRUE,
                                         tolower = TRUE,
                                         stemming = TRUE))
tdmw <- weightTfIdf(tdm)
```

```
## Warning in weightTfIdf(tdm): empty document(s): 72 159 193
```

```
T <- as.matrix(tdmw)
```

Then we create a word cloud for thread #2 using the weighted Term Document Matrix:

```
freqsw <- rowSums(T)


##Remove words that have 'NA'.
freqsw <- freqsw[!is.na(freqsw)]
wordcloud(names(freqsw),
          freqsw,
          random.order = FALSE,
          max.words = 85)
```

Thread 3 Word Cloud:

Again, we follow the steps as described in previous section for thread #1 and #2.

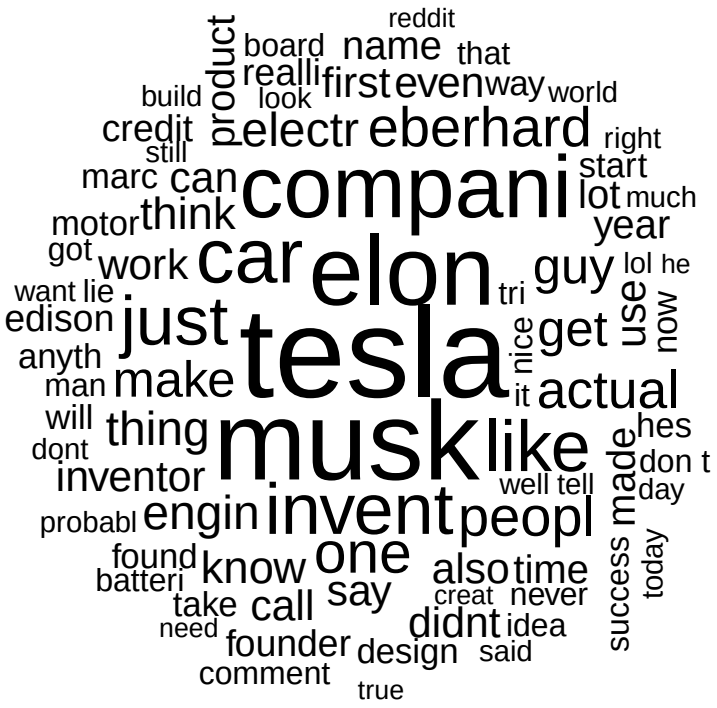
```
#create a term-document matrix for thread #3
corpus <- Corpus(VectorSource(thread_contents3$comments$comment))
corpus <- tm_map(corpus, function(x) iconv(x, to = 'ASCII')) #convert it to the standard ASCII, it will
```

```
## Warning in tm_map.SimpleCorpus(corpus, function(x) iconv(x, to = "ASCII")):
## transformation drops documents
```

```
set.seed(1)
#tdm
tdm <- TermDocumentMatrix(corpus,
                          control = list(removePunctuation = TRUE,
                                         stopwords = TRUE,
                                         removeNumbers = TRUE,
                                         tolower = TRUE,
                                         stemming = TRUE))


empties <- which(colSums(as.matrix(tdm)) == 0)
tdm <- tdm[,-empties] #remove empty columns
M <- as.matrix(tdm)
```

Then again, we create a word cloud for thread #3

```
freqs <- rowSums(M)

#Create a word cloud for thread #3
wordcloud(names(freqs),
          freqs,
          random.order = FALSE,
          max.words = 85)
```



Thread 3 Word Cloud (using weighted TF-IDF):

```
#Create a TF-IDF Word Cloud for thread #3
tdm <- TermDocumentMatrix(corpus,
                          control = list(removePunctuation = TRUE,
                                         stopwords = TRUE,
                                         removeNumbers = TRUE,
                                         tolower = TRUE,
                                         stemming = TRUE))
tdmw <- weightTfIdf(tdm)
```

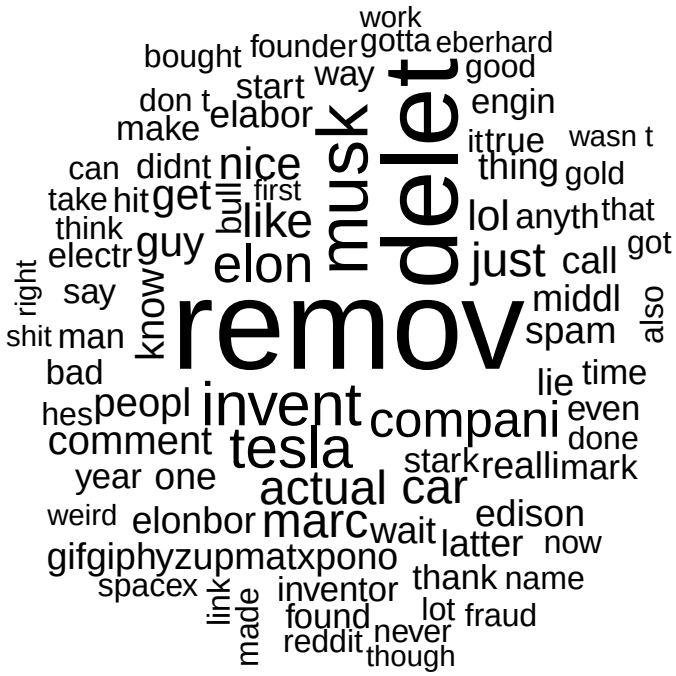
```
## Warning in weightTfIdf(tdm): empty document(s): 249 256 451
```

```
T <- as.matrix(tdmw)
```

Again, we plot the word cloud using the weight Term Document Matrix

```
freqsw <- rowSums(T)


##Remove words that have 'NA'.
freqsw <- freqsw[!is.na(freqsw)]
wordcloud(names(freqsw),
```

```
        freqsw,
        random.order = FALSE,
        max.words = 85) #word cloud
```



Part 4: Comment on your word clouds. Explain the key discussions and topics being addressed in each of the three thread

Question 2:

Part 1: Combine all the threads you collected in question 1 and create a column to label them with their thread number

We use `rbind` to row binding comments from thread #1, 2 and 3.

```
full_thread_content <- rbind(thread_contents1$comments,
                             thread_contents2$comments,
                             thread_contents3$comments)
```

Part 2: For example, lable the first thread comments as '1', label the second thread comments as '2', label the third thread comments as '3'

For this data frame, we set the label for the first thread comments as '1', '2' for the second thread comments, and '3' for the third thread comments.

```
full_thread_content$thread_number[full_thread_content$url ==
                                  thread_contents1$threads$url] <- 1
full_thread_content$thread_number[full_thread_content$url ==
                                  thread_contents2$threads$url] <- 2
full_thread_content$thread_number[full_thread_content$url ==
                                  thread_contents3$threads$url] <- 3
full_thread_content$thread_number <- as.numeric(full_thread_content$thread_number)
```

Checking the structure again

```
mat.full_thread_content <- data.matrix(full_thread_content[,c(-1)]) #remove the url
#column as we already have the column 'thread_number' that represents the thread number
str(mat.full_thread_content)
```

```
##  num [1:1481, 1:10] 266 943 725 189 923 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:10] "author" "date" "timestamp" "score" ...
```

Part 3: Next apply K-means clustering to cluster the combined threads

K-means using Euclidean Distance on the original thread

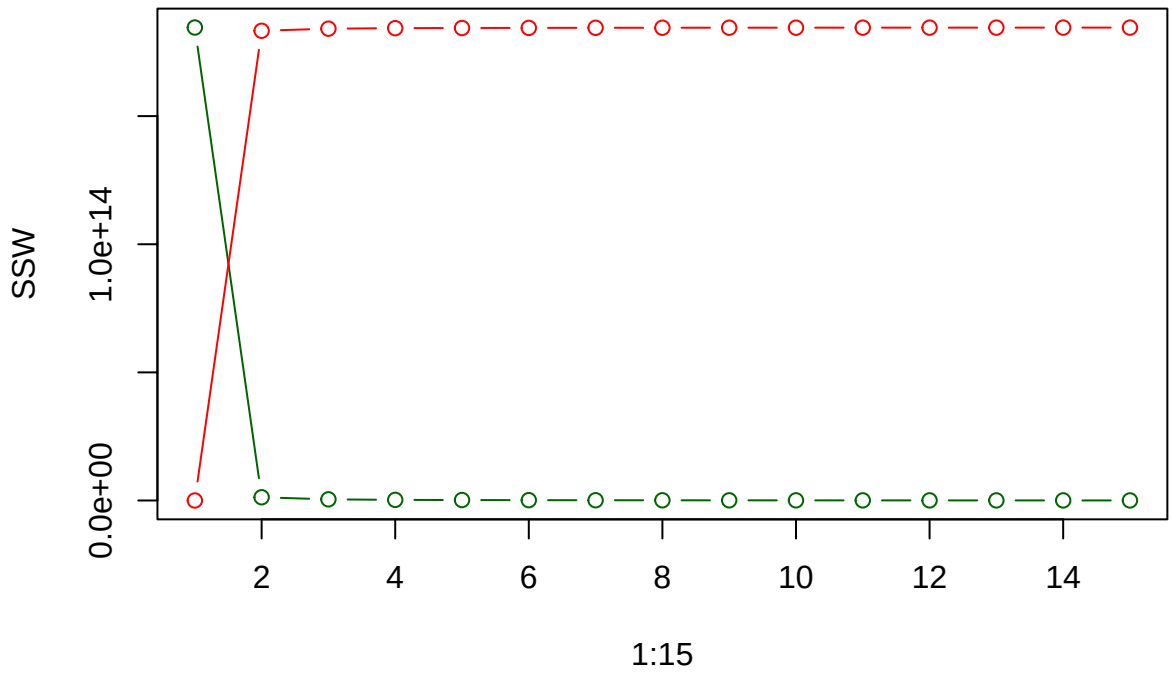
- Since the data `mat.full_thread_content` exists in multi-dimensions (having at least 10 attributes), and there exists a distance between all pairs of observations in the p -dimensions. We can construct a low dimensional 2D space to represent the distance between all pairs of observations.
- Furthermore, using K-means clustering is to mean to use Euclidean space, meaning it uses the Euclidean distance. Hence we will need to convert the distance into Euclidean distance.

```
##Creating a distance matrix:
D <- dist(mat.full_thread_content, method = 'euclidean') #distance matrix on
#matrix full_thread_content using Euclidean distance.
```

```r
#perform Multi-dimensional scaling to 2D Space:
mds.full_thread_content <- cmdscale(D) #performs MDS given the distance D
#and projects it on 2D space.
```

We then plot the values of Sum of squares withins and betweens after performing k-mean clustering with different numbers of clusters on the matrix, in order to visualise how many clusters we need for the k-mean algorithm.

```r
SSW <- rep(0, 15)
SSB <- rep(0, 15)
for (a in 1:15) {
  K <- kmeans(mds.full_thread_content, a, nstart = 20)
  SSW[a] <- K$tot.withinss
  SSB[a] <- K$betweenss
}
#plot results
plot(1:15, SSW, type = 'b', col = 'dark green')
lines(1:15, SSB, type = 'b', col = 'red')
```



By assessing the value of sum of square within and between clusters above, it says the best number of clusters we should have is two. Because:

- The Within Sum of Square (SSW) dropped significantly from when the number of cluster increased from 1 to 2, meaning the distance between points within and the cluster centre is small enough to form its own cluster.

- And the Between Sum of Square (SSB) skyrocketed when increasing the number of cluster from 1 to 2, meaning the distance between cluster centres is large enough to correctly identify the clusters.

From the conclusion above, we can start performing k-mean clustering with the number of cluster of 2.

```
#perform K means
set.seed(1)
K.original <- kmeans(x = mds.full_thread_content, 2 , nstart = 10)
cat("Within Sum of Square: ", K.original$withinss,
    "\nBetween Sum of Square: ",K.original$betweenss)
```

```
## Within Sum of Square:  97824413584 1.165843e+12
## Between Sum of Square:  1.833026e+14
```

K-means using Term Document Matrix on comments (with distance as Cosine Distance):

In the section above, we have used the original thread to compute K-means clustering. However, in this section, we will perform some text transformations using library `tm` to obtain a Term Document Matrix. We then use Term Document Matrix to apply K-means on it to find the best clustering.

```
red.corpus <- Corpus(VectorSource(full_thread_content$comment))
corpus = tm_map(red.corpus, function(x) iconv(x, to='ASCII')) # remove #special characters

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

#Warning transformation drops documents
```

We can inspect the final document to see how texts are transformed:

```
inspect(corpus[1:5]) #check on the first five documents
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 5
##
## [1] honest wtf nickelback ever anyon look graph
## [3] everi time make laugh            ahh gday ahhh gdayuyah
## [5] wth joey head
```

- As we can see above, words have been transformed, texts become more transparent and readable.

After that, we formulate a Term Document Matrix from the corpus obtained above and assign a label to each document to distinguish which thread each of the document belongs to.

```
#Create a term document matrix
red.tdm <- TermDocumentMatrix(corpus)
red.tdm <- t(as.matrix(red.tdm))
red.tdm <- cbind(red.tdm,full_thread_content$thread_number) #labels each doc with
#thread number

red.tdm_nolabel <- red.tdm[,-ncol(red.tdm)] #create another variable to store
#the tdm matrix without the labels. This will be used for K-means clutering.

#remove empty
empties <- which(rowSums(abs(red.tdm_nolabel)) == 0)
red.mat <- red.tdm[-empties,]
red.tdm_nolabel <- red.tdm_nolabel[-empties,] #we also remove empty row for
#this variable.
```

With the variable `red.tdm_nolabel`, we could perform K-means clustering to assign cluster to each document .

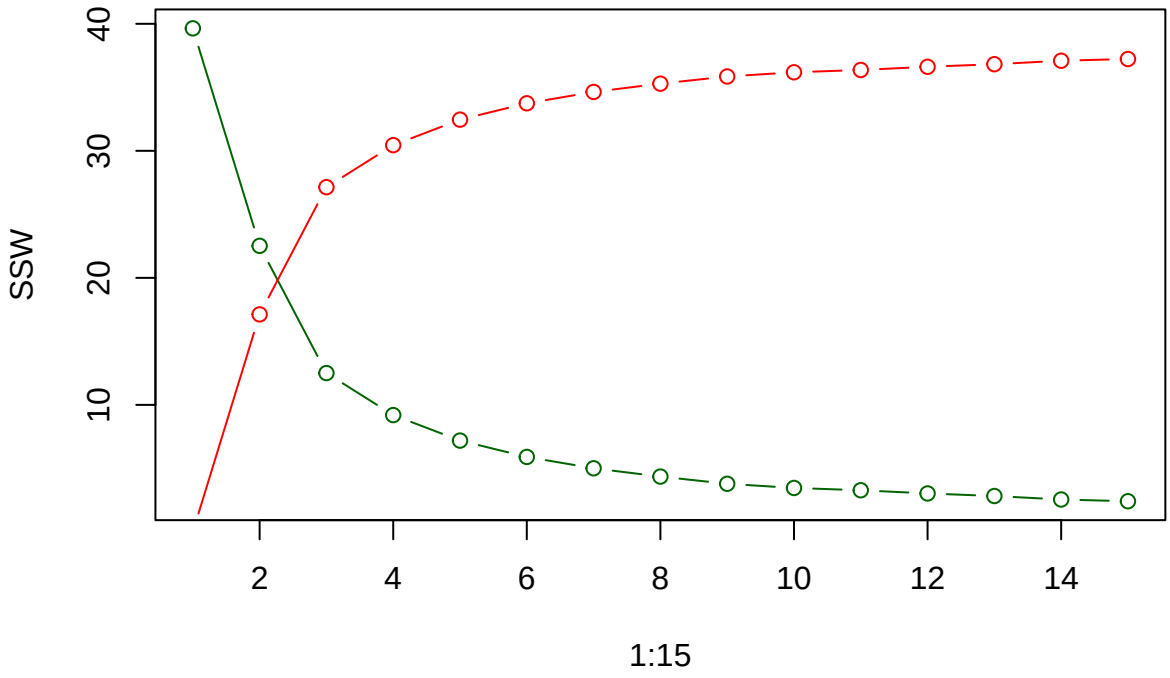
```
#normalise to unit length
norm.red.tdm_nolabel <- diag(1/sqrt(rowSums(red.tdm_nolabel^2))) %*% red.tdm_nolabel

#create distance matrix
D.tdm <- dist(norm.red.tdm_nolabel, method = 'euclidean')^2/2
red.mds <- cmdscale(D.tdm)
```

- We also need to check the Within Sum of Square and the Between Sum of Square for the kmean algorithm

```
SSW <- rep(0, 15)
SSB <- rep(0, 15)
for (a in 1:15) {
  K.red <- kmeans(red.mds, a, nstart = 20)
  SSW[a] <- K.red$tot.withinss
  SSB[a] <- K.red$betweenss
}
```

```
## Warning: did not converge in 10 iterations
```

```
#plot results
plot(1:15, SSW, type = 'b', col = 'dark green')
lines(1:15, SSB, type = 'b', col = 'red')
```



By assessing the value of sum of square within and between clusters above, We know the best number of clusters we should get is 3. Because:

- The Within Sum of Square (SSW) dropped significantly from when the number of cluster increased from 1 to 2 and from 2 to 3, meaning the distance between points within and the cluster centre is small enough to form its own cluster.
- And the Between Sum of Square (SSB) significantly increased when increasing the number of cluster from 1 to 2 and as well from 2 to 3, meaning the distance between cluster centre is large enough to correctly identify the clusters.
- Also at three 3 the Within Sum of Square seems to converge (meaning it becomes more stabilised)

Hence the best number of clusters is 3, we will perform K-means with 3 clusters

```
set.seed(1)
red.kmeans <- kmeans(red.mds, 3, nstart = 20)
```

K-means using TF-IDF on comments (with distance as Cosine Distance):

Similar to the method above, we obtain the Term Document Matrix, and perform TF-IDF on each document to get the weighted TF-IDF.

```
#Create a term document matrix
red.tdmw <- TermDocumentMatrix(corpus)
red.tdmw <- weightTfIdf(red.tdmw)
```

```
## Warning in weightTfIdf(red.tdmw): empty document(s): 12 132 219 416 567 654 688
## 1236 1243 1438
```

```
red.tdmw <- t(as.matrix(red.tdmw))

red.tdmw <- cbind(red.tdmw,full_thread_content$thread_number) #labels each doc
red.tdmw_nolabel <- red.tdmw[,-ncol(red.tdmw)] #create another variable to store
#the tdm matrix without the labels. This will be used for K-means clutering.

#remove empty
empties <- which(rowSums(abs(red.tdmw_nolabel)) == 0)
red.matw <- red.tdmw[-empties,]
red.tdmw_nolabel <- red.tdmw_nolabel[-empties,] #we also remove empty row for
#this variable.
```

After that, we use normalise the matrix `red.tdmw_nolabel` to unit length, which will be used to calculate the cosine distance:

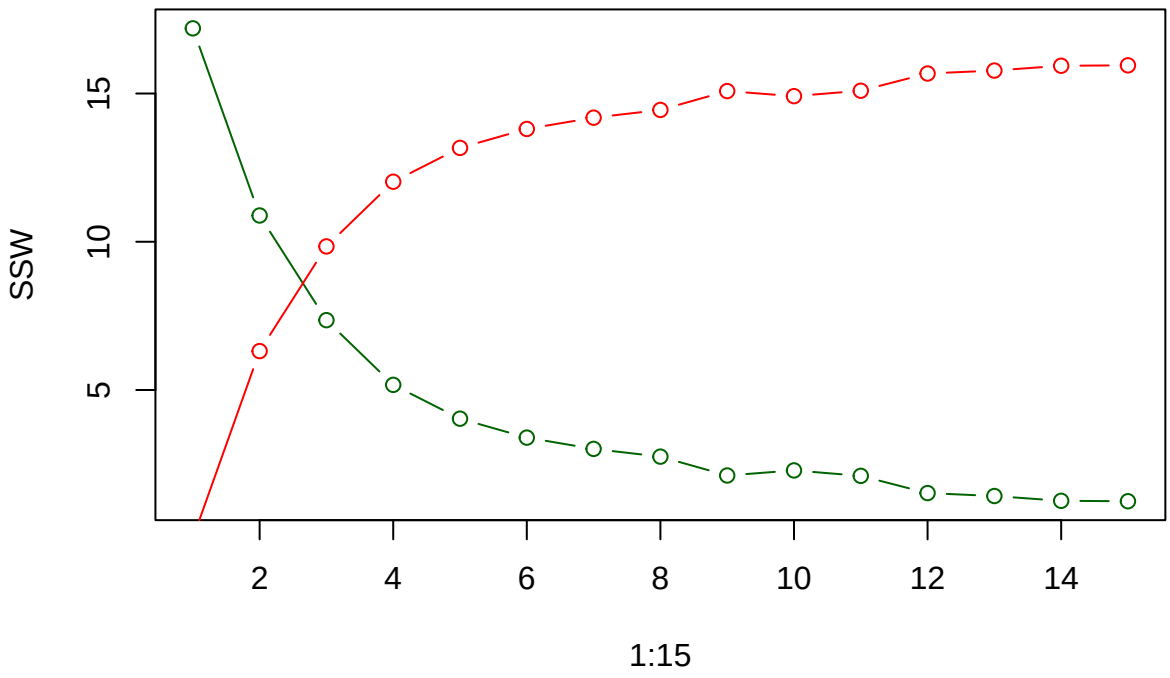
```
#normalise to unit length
norm.red.tdmw_nolabel <-
  diag(1/sqrt(rowSums(red.tdmw_nolabel^2))) %*% red.tdmw_nolabel

#create a cosine distance matrix
D.weighted <- dist(norm.red.tdmw_nolabel, method = 'euclidean')^2/2
red.mds.weighted <- cmdscale(D.weighted)
```

Again, We need to check for the Within Sum of Square and Between Sum of Square for different number of clusters from 1 to at least 15, to see which value is the best for K-means clustering when using Cosine Distance on this dataset.

```
SSW <- rep(0, 15)
SSB <- rep(0, 15)
for (a in 1:15) {
  K.red.weighted <- kmeans(red.mds.weighted, a, nstart = 20)
  SSW[a] <- K.red.weighted$tot.withinss
  SSB[a] <- K.red.weighted$betweenss
}
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

```
#plot results
plot(1:15, SSW, type = 'b', col = 'dark green')
lines(1:15, SSB, type = 'b', col = 'red')
```



- From the graph, we can see that the Within Sum of Square gradually decreased.
- From 1 to 3, the value of SSW seems, to drop heavily but becomes softer after 3.

Hence, by using the elbow method, It would be appropriate to pick 3 as the best number of clusters for this K-means clustering.

```
set.seed(1)
red.kmeans.weighted <- kmeans(red.mds.weighted, 3, nstart = 20)
```

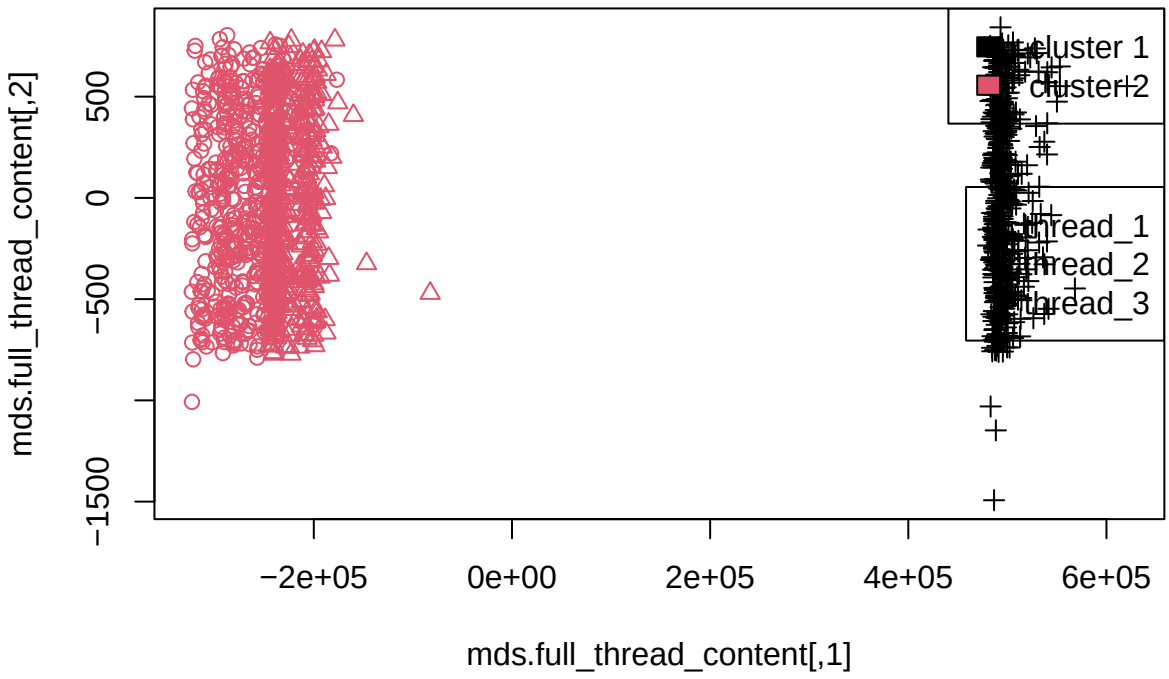
Part 4: Visualise the results of your clustering in two-dimensional vector space. Ensure that your visualisation includes both the clusters and the original labels of the documents

(Continued from Question 2 Part 3 - Kmeans clustering on the original thread) - Visualising K-means clustering with the original thread:

From the `K` data obtained above, we visualise the clustering in two-dimensional vector space.

```
#visualise the clusters

plot(mds.full_thread_content,
     col = K.original$cluster,
     pch = full_thread_content$thread_number)
```

```
legend(x = 'topright',
       legend = c('cluster 1', 'cluster 2'),
       fill = c(1,2),
       border = 'black')
legend(x = 'right',
       legend = c('thread_1','thread_2','thread_3'),
       pch = c(1,2,3))
```



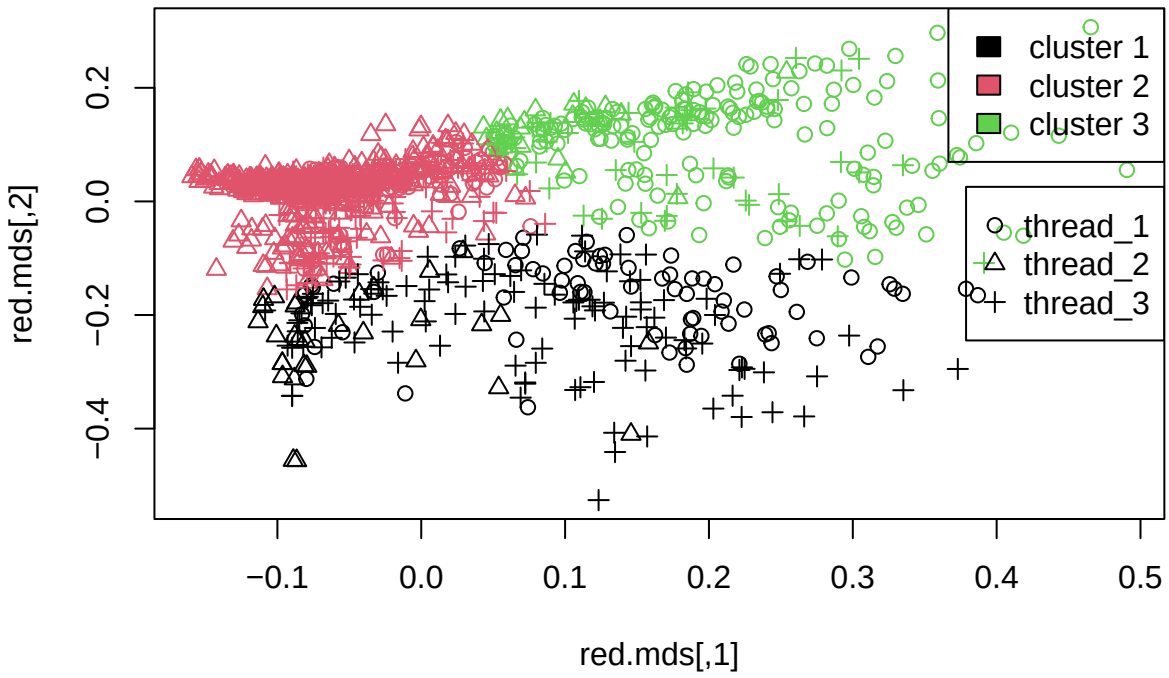
From the visualisation of the clustering above, it can be seen that:

- Thread 1 and 2 are perfectly clustered into one cluster (cluster 2), when the second cluster (cluster 1) is independently separated from the first cluster.
- In the first cluster (cluster 2), there are 2 threads comments being combined together, thread 1 and 2 (meaning thread comments with the label '1' and '2'); whereas in the second cluster (cluster 1), there exists only thread 3 (thread comments with the label '3').

(Continued from Question 2 - Part 3 - Kmeans Clustering using TDM matrix with Cosine Distance) - Visualising K-means using Term Document Matrix (with Cosine Distance):

```
plot(red.mds,
     col = red.kmeans$cluster,
     pch = red.mat[,ncol(red.mat)])
legend(x = 'topright',
```

```
       legend = c('cluster 1', 'cluster 2','cluster 3'),
       fill = c(1,2,3),
       border = 'black')
legend(x = 'right',
       legend = c('thread_1','thread_2','thread_3'),
       pch = c(1,2,3))
```



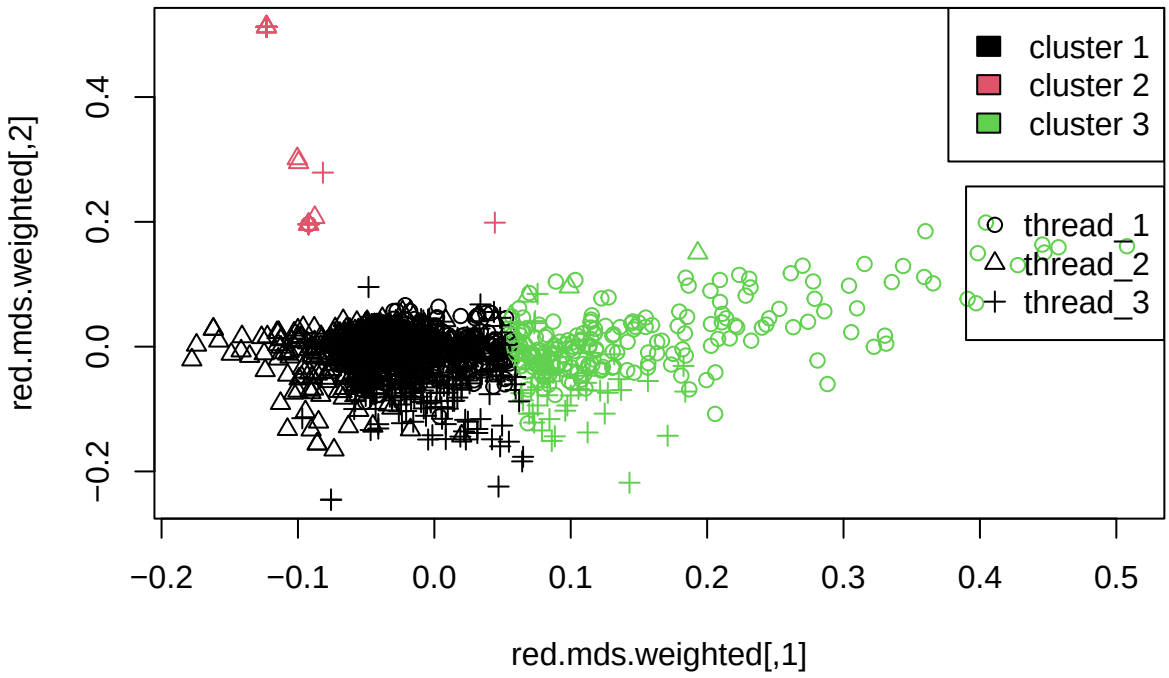
From the plot above, it can be seen that:

- Using Cosine Distance on Term Document Data can help captures most of the data and making it clearer to observe.
- There are three 3 clusters and 3 threads numbers, each cluster contains a mixture of threads. There is no distinguishable line between 3 threads.

(Continued from Question 2 - Part 3 - Kmeans Clustering using TF-IDF TDM matrix with Cosine Distance) - Visualising K-means using weighted TF-IDF TDM Matrix (with Cosine Distance):

```
plot(red.mds.weighted,
     col = red.kmeans.weighted$cluster,
     pch = red.matw[,ncol(red.matw)])
legend(x = 'topright',
```

```
       legend = c('cluster 1', 'cluster 2','cluster 3'),
       fill = c(1,2,3),
       border = 'black')
legend(x = 'right',
       legend = c('thread_1','thread_2','thread_3'),
       pch = c(1,2,3))
```



From the visualisation of the K-Means clustering above, we observe that:

- Most of the data is captured in the plot, and can be seen to cluster around the original centred point (0,0)
- There is no distinguishable line between each cluster and the threads numbers.

Part 5: Comment on your findings. Assess the performance of K-means clustering in terms of correctly identifying clusters. Did it effectively identify the clusters?

In terms of minimising the Within Sum of Square, and maximising the Between Sum of Square, the clustering model with k = 2 above has done effectively in clustering the data into 2 distinctive clusters while maintaining a small SSW and large SSB value.

```
#confusion matrix
cm <- table(full_thread_content$thread_number, K.original$cluster)
cm
```

```
##
##       1   2
##   1   0 495
##   2   0 492
##   3 494   0
```

```
cat("Accuracy Score: ", (cm[3,1] + cm[2,2])/sum(cm))
```

```
## Accuracy Score:  0.6657664
```

- From the confusion matrix table above, we can see that cluster 2 has effectively clustered the data labelled as '1' and '2' into one cluster, while cluster 1 includes only the data labelled as '3'.

(Continued from Question 2 - Part 4 - K-mean clustering on the original thread) Assessing the performance of K-means clustering using k = 3:

However, since we have already known that we have 3 distinguished threads that are labelled as '1', '2', '3', we only need to assess on whether the performance of K-means clustering where K number of clusters equals to 3 is performing better than where K number of clusters equals to 2, in terms of correctly identifying clusters.

```
set.seed(1)
K <- kmeans(x = mds.full_thread_content, 3 , nstart = 10) #K-means

#confusion matrix
cm0 <- table(full_thread_content$thread_number, K$cluster)
cm0
```

```
##
##       1   2   3
##   1   0 156 339
##   2   0 492   0
##   3 494   0   0
```

```
cat("Accuracy Score: " ,(cm0[1,3] + cm0[2,2] + cm0[3,1])/sum(cm0))
```

```
## Accuracy Score:  0.8946658
```

- Based on the graph of SSW and SSB in the previous section, the SSW value of `K` is slightly smaller for when K number of clusters = 3 than K = 2, and the SSB value of `K` for when K = 3 is also closer to the maximised SSB value that could be obtained.
- However, in terms of correctly identifying clusters, using K-means clustering with K number of clusters = 3 is better than K = 2. Based on the confusion matrix table shown in the above section. There are about 156 misclassified threads labelled as '1', stored within Cluster 2, which also contains apprximately 492 threads labelled as '2'. Meanwhile the confusion matrix in the previous K-means (K = 2), shows there are 495 thread #1 is clustered into cluster 2 with thread #2.
- The Accuracy score for k-means model when K = 2 is approximately 66%. Meanwhile, the accuracy score for k-means model when K = 3 is 89%.

Therefore, Based on the performance of the two K-means clustering methods used above, the K-means model with K number of clusters equals to 3 has done more effective than the second model where the number of clusters equals to 2, in terms of identifying the correct clusters for each data points.

(Continued from Question 2 Part 4 - K-means Clustering on TDM with Cosine Distance) Assess the performance of K-means Clustering on TDM matrix with Cosine Distance:

From the k-means obtained above `red.kmeans` we can check how many threads have been assigned correctly to each cluster:

```
cm1 <- table(red.mat[,ncol(red.mat)], red.kmeans$cluster)
cm1
```

```
##
##       1   2   3
##   1  78 235 178
##   2  31 433  25
##   3 118 325  48
```

```
cat("Accuracy Score: ", ((cm1[3,1] + cm1[2,2] + cm1[1,3])/sum(cm1)))
```

```
## Accuracy Score:  0.4955812
```

- From the table, it can be seen that there are 122 threads #1 have been correctly identified as being in cluster 1 and 437 threads #2 as being in cluster 2 and about 177 threads #3 as in cluster 3.
- Beside the estimation of data above, the overall accuracy score of the performance is also quite low. The model returns an accuracy score of 49%.

This overall suggests that transforming the comments of each thread into Term Document Matrix and then using K-means Clustering with Cosine Distance, is not a very effective approach to classify threads' true labels.

(Continued from Question 2 Part 4 - K-means Clustering on TF-IDF weighted TDM with Cosine Distance)

```
cm2 <- table(red.matw[,ncol(red.matw)],red.kmeans.weighted$cluster)
cm2
```

```
##
##       1   2   3
##   1 288   4 199
##   2 473   8   8
##   3 428  17  46
```

```
cat("Accuracy Score: ", (cm2[3,1] + cm2[2,2] + cm2[1,3])/sum(cm2))
```

```
## Accuracy Score:  0.4316791
```

- From the table, it can be seen that there are 430 threads #1 have been correctly identified as being in cluster 1 and 8 threads #2 as being in cluster 2 and about 200 threads #3 as in cluster 3.
- Beside the estimation of data above, the overall accuracy score of the performance is also quite low. The model returns an accuracy score of 43%, which is also lower than the previous K-mean model's accuracy score.

Therefore, this overall suggests that transforming the comments of each thread into Term Document Matrix with TF-IDF weight and then using K-means Clustering with Cosine Distance, is also not a very effective approach to classify threads' true labels.

Question 3:

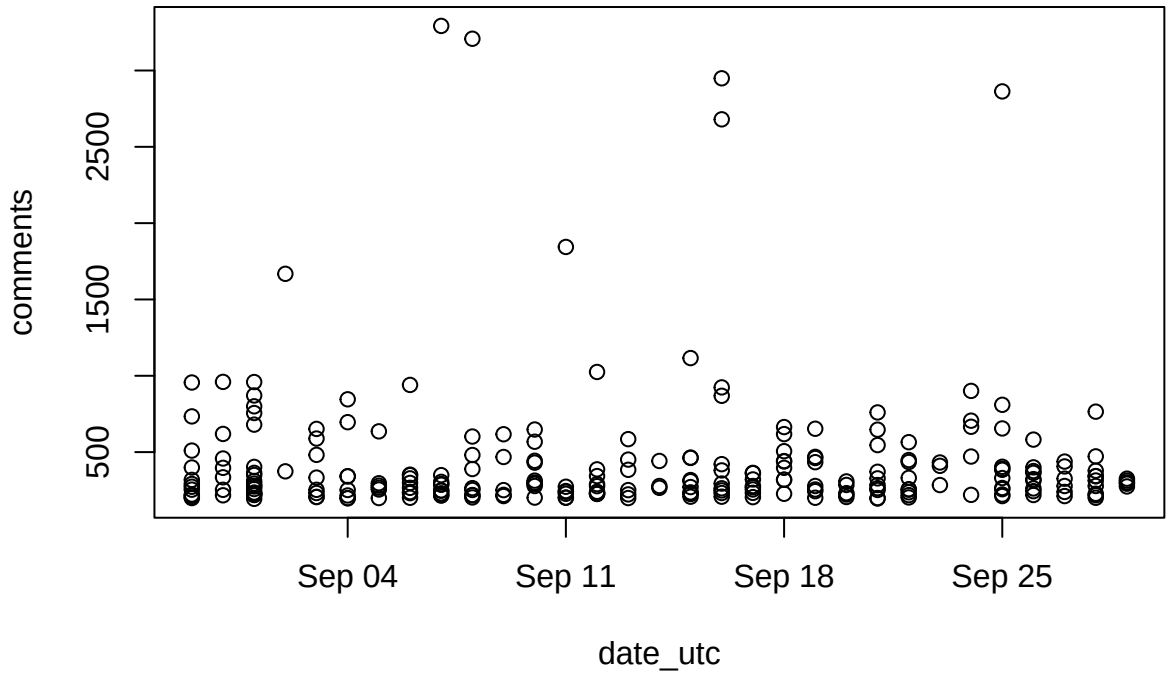
Part 1: Use all thread URLs on Reddit that you identified in question 1.

```
full_thread_content <- tesla_threads
full_thread_content$date_utc <- as.Date(full_thread_content$date_utc)
sapply(full_thread_content, class)
```

```
##    date_utc   timestamp       title        text   subreddit    comments
##      "Date"   "numeric" "character" "character" "character"   "numeric"
##         url
## "character"
```

Let's see the distribution of the numbers of comments of each thread over a period of time.

```
plot(comments ~ date_utc, data = full_thread_content)
```



Part 2: Test if there exists a linear relationship between the number of comments in threads and their corresponding dates.

Note: you may need to convert dates to a date format

$H_0 : \beta = 0$. There is no evidence of a linear relationship between the number of comments and their corresponding dates.

$H_a : \beta \neq 0$: There is evidence of a linear relationship between the number of comments and their corresponding dates.

Assuming the based line p-value to reject the Hypothesis starts at 0.05.

```
lin_reg <- lm(comments ~ date_utc, data = full_thread_content)
summary(lin_reg)
```

```
##
## Call:
## lm(formula = comments ~ date_utc, data = full_thread_content)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -248.85 -191.17 -132.93   13.96 2855.86
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 22353.475  58945.811   0.379    0.705
## date_utc       -1.118      3.005  -0.372    0.710
##
## Residual standard error: 432.1 on 248 degrees of freedom
## Multiple R-squared:  0.0005575,  Adjusted R-squared:  -0.003472
## F-statistic: 0.1383 on 1 and 248 DF,  p-value: 0.7103
```

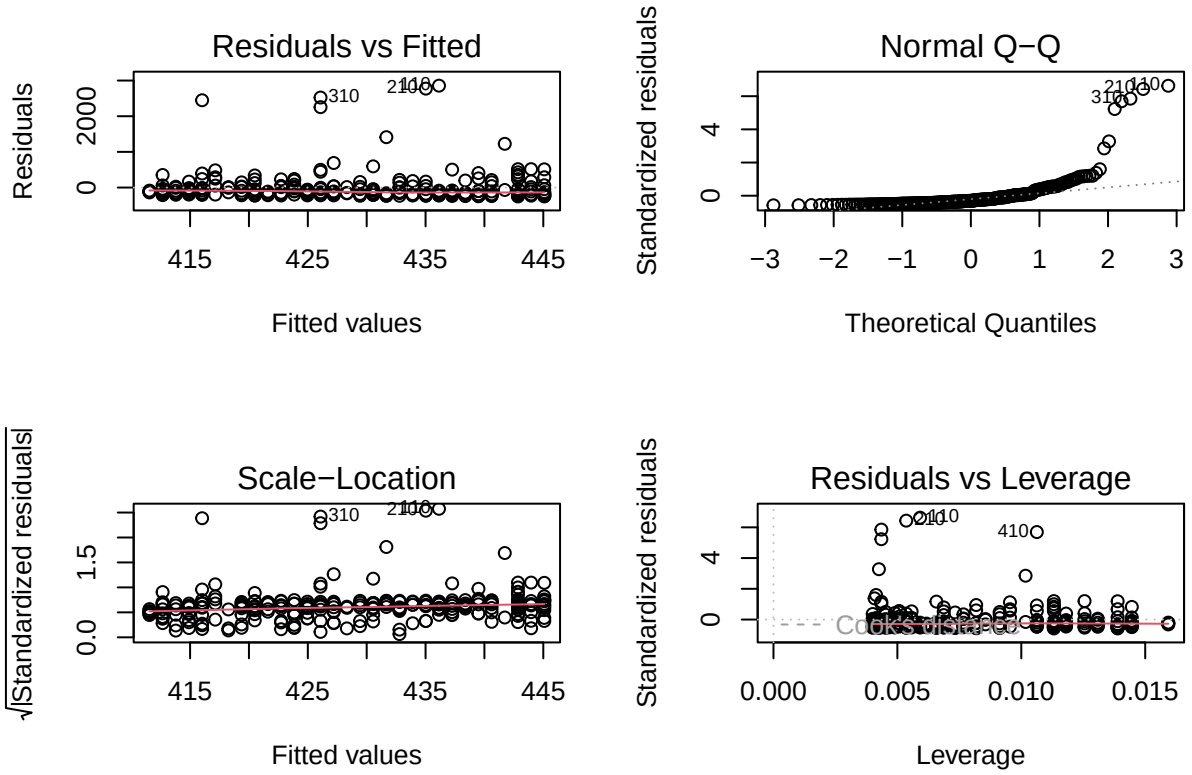
Based on the summarised information about the linear regression model above:

- The intercept value beween `date_utc` and `comments` is estimated around 22353.475 and and the gradient is about -1.118.
- The overall p-value for the gradient of the model is large ($0.710 > 0.05$). and its t-value is small, meaning the population gradient is closed to be zero.

Hence, we do not reject the null hypothesis. There is not enough evidence of a linear relationship between the corresponding date and the number of comments.

Extra part:

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



Plot 1(Residual vs Fitted): showing that many of the points deviate far away from the fitted line, meaning the residuals between the actual points and the fitted line are large.

Plot 2(Q-Q plot): showing that the data does not follow the normal distribution as many of the points at the start and at the end does not lay on the expected line of normal distribution.

Question 4:

Part 1: Retrieve the content of the top thread you identified in question 1. Test whether the number of comments on a thread is equally likely on each day.

Before that, we can check the number of comments on each day

```
thread1_comments <- thread_contents1$comments
thread1_comments$date <- as.Date(thread1_comments$date)
thread1_mat <- table(thread1_comments$date)
thread1_mat
```

```
##
## 2023-09-07 2023-09-08
##        390        105
```

- On 7/9/2023, there is a total of 390 comments and
- On 8/9/2023, there are 105 comments.
- Together, we have a total of 425 comments on thread #1

In this case, To test if the number of comments on a thread is equally likely on each day, we use chi-square test for proportion.

- We also know that the number of comments on each day is independent of day, meaning the number of comments on one day do not affect the number of comments of another day.
- The data is large enough to compute chi-square test (there are more than 5 observations).

From there, we can form our hypothesis:

H_0: The number of comments on each day is equally likely.

H_a: The number of comment on each day is not equally likely.

```
chisq.test(thread1_mat, p = c(1/2, 1/2), B = 2000)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  thread1_mat
## X-squared = 164.09, df = 1, p-value < 2.2e-16
```

```
#we set the probabilities to equal to 1/2 and 1/2 since we expect the number
#of comments to be equal on each day.
#B = 2000, we sample it 2000 times
```

According to the result from the chi-square test above, we have the p-value is smaller than 0.05. Hence, we reject the null-hypothesis, because there is evidence to support that the number of comments is not equally likely on each day.

Question 5:

Part 1: Using Mastodon, identify users who are related to your chosen topic

```
library(rtoot)
```

```
## Warning: package 'rtoot' was built under R version 4.2.3
```

```
library(igraph)
#vignette('rtoot') #show a specific package instruction

#auth_setup('mastodon.social', 'user')
#a <- get_timeline_hashtag(hashtag = 'tesla', instance = 'mastodon.social', local = TRUE)
#saveRDS(a, 'Mastodon_timeline.RDS')
a <- readRDS('Mastodon_timeline.RDS')
```

Part 2: Identify the top 5 most active users, based on the highest number of statuses they have posted.

```
#five most active users based on the highest number of statuses they have posted
tesla_active_users <- do.call(rbind, a$account)
tesla_active_users <-
  unique(tesla_active_users[(order(tesla_active_users$statuses_count,
                                   decreasing = TRUE)),])[c(1,4,6,7,8),]
# we have to use unique to eliminate posts that share the same users.
```

Reason to why I were not choosing the top 5 users in order (e.g. 1 to 5):

- Due to the lack of accessibility to some users' private information, some users do not show the names of their followers or users' followings. Hence I have chosen to extract the top 5 most active users from the top 10 most active users (having highest statuses counts), that are willing to share their information of followers and who they are following with. In this case, I have chosen top 1, top 4, 6, 7 and top 8 to be my top 5 most active users.
- For why I did not choose top 2, 3, 5 to be a part of my top 5 is because there were not enough information about their following users or not having information at all. Therefore, I seek for an alternative path to select my top 5 active users.

Part 3: Download 50 followers and 50 friends (there are the users that your user follows) of these 5 top users.

Note 1: You might encounter limitations in downloading friends.

Note 2: If you can only download fewer than 50 followers or friends for certain users that's acceptable.

We retrieve the information for the top 5 users' friends/following and save it as RDS files and can load it to a stored variable later on. It is because the content of data tends to be fluctuated over a period of time (in this case, it is daily), giving imprecise records of data.

```
#create a list of friends
#n <- length(tesla_active_users$id)
#user_friends <- list(NA)
#for (i in 1:n) {
#  friends <- get_account_following(tesla_active_users$id[i], limit = 50)
#  user_friends[[i]] <- friends$username
#}
#saveRDS(user_friends, 'user_friends.RDS')
```

As similar to the previous step, we retrieve the information for the top 5 users' followers.

```
#Create a list of followers
#n <- length(tesla_active_users$id)
#users_followers <- list(NA)
#for (i in 1:n) {
#  followers <- get_account_followers(tesla_active_users$id[i], limit = 50)
#  users_followers[[i]] <- followers$username
#}
#saveRDS(users_followers, 'users_followers.RDS')
```

Part 4: Create a graph and visualise the relationships among these users

We then load the `user_friends.RDS` file, and make a table that have two columns, first column data represents the starting edges and the second column data represents the ending edges.

```
user_friends <- readRDS('user_friends.RDS')

user_friends_graph <- list(NA)
for (i in 1:length(user_friends)) {
  name_rep <- rep(tesla_active_users$username[i], length(user_friends[[i]]))
  user_friends_graph[[i]] <- cbind(name_rep, user_friends[[i]])
}

user_friends_graph <- do.call(rbind, user_friends_graph) #row bind the dataset
head(user_friends_graph, 3)
```

```
##      name_rep
## [1,] "Verfassungklage" "thoralf"
## [2,] "Verfassungklage" "fabianschaar"
## [3,] "Verfassungklage" "sils"
```

```
dim(user_friends_graph) #check the dimension #as expected, because we will
```

```
## [1] 400   2
```

```
#visualise it as a network. So it needs to have initial edges and ending edge
#meaning 2 columns. The first column stores the initial edges and the second stores
#the final edges.
```

Similarly, we do the same to information of the top 5 users' followers

```
users_followers <- readRDS('users_followers.RDS')

user_followers_graph <- list(NA)
for (i in 1:length(users_followers)) {
  name_rep <- rep(tesla_active_users$username[i], length(users_followers[[i]]))
  user_followers_graph[[i]] <- cbind(name_rep, users_followers[[i]])
}

user_followers_graph <- do.call(rbind, user_followers_graph)
user_followers_graph <- user_followers_graph[,c(2,1)]
head(user_followers_graph, 3)
```

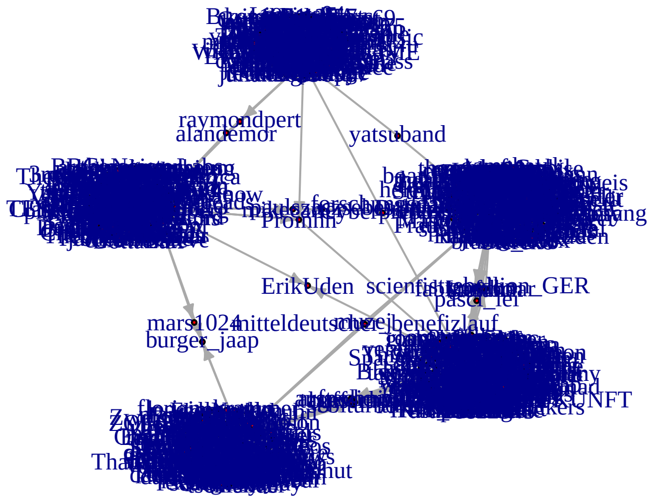
```
##                             name_rep
## [1,] "andreasva"            "Verfassungklage"
## [2,] "Mathias"              "Verfassungklage"
## [3,] "theanxiousresistance" "Verfassungklage"
```

```
dim(user_followers_graph) # dimension is 2 #hence, it is corrected.
```

```
## [1] 400   2
```

And now we can plot the social network graph.

```
#combine friends and followers into a big relationship table
friend_followers <- rbind(user_friends_graph, user_followers_graph)


#plot these relationships
set.seed(1)
g1 <- graph_from_edgelist(friend_followers, directed = TRUE)
plot(g1,
     layout = layout.fruchterman.reingold,
     vertex.size = 2,
     edge.arrow.size = 0.4,
     vertex.color = 'red',
     vertex.label.cex = 0.75)
```


raymondpert
alandemor
yatsuband
ErikUden
mars1024
burger_jaap

Question 6:

Part 1: Find the most central users in your graph using all centrality measures you learned in this subject. Comment on your findings

Centrality measure (Degree Centrality):

```
sort(degree(g1, mode = 'out'), decreasing = TRUE)[1:20]
```

```
##             Verfassungklage                  pallenberg
##                          80                          80
##                richardknott                   DharmaDog
##                          80                          80
##                  macmaniacs              londonerabroad
##                          80                           3
##                     Proinhh                  mikecarter
##                           2                           2
##                 raymondpert                       muzej
##                           2                           2
##                   yatsuband mitteldeutscher_benefizlauf
##                           2                           2
##                      gordon              paulszaszsebes
##                           2                           2
##                 Cyberpreppy                   alandemor
##                           2                           2
##                  Coolmccool                Podsafepilot
##                           2                           2
##                  sorcerer86                fabianschaar
##                           2                           1
```

```
sort(degree(g1, mode = 'in'), decreasing = TRUE)[1:20]
```

```
## Verfassungklage      pallenberg    richardknott       DharmaDog      macmaniacs
##              80              80              80              80              80
##  londonerabroad       pasci_lei           Lambo        ErikUden     aufpolieren
##               3               2               2               2               2
##       bufferapp     arstechnica     tagesschau         Proinhh     burger_jaap
##               2               2               2               2               2
##         thoralf    fabianschaar            sils         danarel        WildMics
##               1               1               1               1               1
```

Centrality measures (Closeness Centrality):

```
sort(closeness(g1), decreasing = TRUE)[1:20]
```

```
##     macmaniacs    burger_jaap         layer8 the_white_wolf       aurelien
##    0.012500000    0.006289308    0.006289308    0.006289308    0.006289308
##     FrankB1274        Saidfaz            jws           mich    toensberger
```

```
##    0.006289308    0.006289308    0.006289308    0.006289308    0.006289308
##    andrekaiser    nickycolman   DerKlimablog  torstengrisel        corsair
##    0.006289308    0.006289308    0.006289308    0.006289308    0.006289308
##       magzag71  Herr_Davidson       nerdweib     Breznsoiza      sebiturbo
##    0.006289308    0.006289308    0.006289308    0.006289308    0.006211180
```

Centrality measure (Betweenness Centrality):

```
sort(betweenness(g1), decreasing = TRUE)[1:10]
```

```
##      pallenberg    richardknott Verfassungklage      macmaniacs       DharmaDog
##       46605.971       37968.295       36429.314       29908.533       29655.886
##         Proinhh       sebiturbo      mikecarter     raymondpert    fabianschaar
##       22944.655       13132.238        9148.167        6849.498        6011.990
```