FIT1043 Assignment 3
Diong Chen Xi 32722656
28th September 2021

Part A:

1. The following line of code is used to decompress the zip file:
   "gunzip FIT1043_Dataset.gz"

   Then the next line of code is used to show the size of the decompressed text file.
   "ls -lh FIT1043_Dataset"
   This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ ls -lh FIT1043_Dataset
-rwx------+ 1 User User 193M Sep 28 11:36 FIT1043_Dataset
```

   We can see that the file size is 193 Megabytes.

2. The following line of code is used to show the first five lines of data in the file.
   "head -5 FIT1043_Dataset"

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ head -5 FIT1043_Dataset
0,1467810672,Mon Apr 06 22:19:49 PDT 2009,NO_QUERY,scotthamilton,is upset that he can't u
0,1467810917,Mon Apr 06 22:19:53 PDT 2009,NO_QUERY,mattycus,@Kenichan I dived many times
0,1467811184,Mon Apr 06 22:19:57 PDT 2009,NO_QUERY,ElleCTF,my whole body feels itchy and
0,1467811193,Mon Apr 06 22:19:57 PDT 2009,NO_QUERY,Karoli,@nationwideclass no it's not be
0,1467811372,Mon Apr 06 22:20:00 PDT 2009,NO_QUERY,joy_wolf,@Kwesidei not the whole crew
```

   There seems to be a lot of commas, which suggests the delimiter may be a "," character, therefore I pass the command to *"less"*.
   "head -5 FIT1043_Dataset | less"

   Then I typed "/," in the "less" interface, which highlights all the commas present in the shown data, and indeed the comma character is the delimiter used to separate the columns in the file.

```
0,1467810672,Mon Apr 06 22:19:49 PDT 2009,NO_QUERY,scotthamilton,is upset that he can't u
0,1467810917,Mon Apr 06 22:19:53 PDT 2009,NO_QUERY,mattycus,@Kenichan I dived many times
0,1467811184,Mon Apr 06 22:19:57 PDT 2009,NO_QUERY,ElleCTF,my whole body feels itchy and
0,1467811193,Mon Apr 06 22:19:57 PDT 2009,NO_QUERY,Karoli,@nationwideclass no it's not be
0,1467811372,Mon Apr 06 22:20:00 PDT 2009,NO_QUERY,joy_wolf,@Kwesidei not the whole crew
```

3. The following line of code is used to find the number of lines in the file.
   "wc -l FIT1043_Dataset"
   This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ wc -l FIT1043_Dataset
1471793 FIT1043_Dataset
```

   We can see that there are 1471793 rows in the file.

4. The following line of code is used to calculate the number of delimiters in the first line in the file.
   "head -n1 FIT1043_Dataset | grep -o "," | wc -l"
   This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ head -n1 FIT1043_Dataset | grep -o "," | wc -l
5
```

   As there are 5 delimiters, we know that there will be 6 columns in the row. Since the data is formatted in the same way, we can deduce that there will be 6 columns for each row in the file.

5. Since the user IDs are in the fifth column of the file, I can get the data by using *"awk -F ',' '{print $5}' FIT1043_Dataset".* Then, by using the "*sort"* pipeline, I can sort similar ids together, paired with the *"uniq"* pipeline, I can get the list of non-repetitive (i.e. unique) user IDs in that column. With the *"wc -l"* pipeline I can get the number of lines in the list, which corresponds to the number of unique users.

   The following line of code is used to calculate the number of occurrences of unique user IDs.
   "awk -F ',' '{print $5}' FIT1043_Dataset | sort | uniq | wc -l"
   This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{print $5}' FIT1043_Dataset | sort | uniq | wc -l
626684
```

   We can see that there are 626684 users.

6. Assuming that the data is ordered by date in chronological order, we know that the first line of data indicates the earliest instance of data recorded, while the last line of data indicates the last instance of data recorded. Therefore, by getting the data of the first and last line of data in the file we can know the date range of Twitter posts in this file. We can get the column of dates in the third column of the file.

   However, the data is sorted by sentiment, then by date, which means that the last instance of a positive post might be on a later date than the last instance of a negative post (and vice versa for the earliest instance)

   Therefore I used the following lines of code to find the dates of the first and last instances of the positive, neutral, and negative tweets.

   First instance of negative tweets:
   "awk -F ',' 'if ($1 == 0) {print $3}}' FIT1043_Dataset | head -1"
   This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 0) {print $3}}' FIT1043_Dataset | head -1
Mon Apr 06 22:19:49 PDT 2009
```

   Last instance of negative tweets:
   "awk -F ',' 'if ($1 == 0) {print $3}}' FIT1043_Dataset | tail -1"
   This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 0) {print $3}}' FIT1043_Dataset | tail -1
Thu Jun 25 10:28:31 PDT 2009
```

   First instance of neutral tweets:
   "awk -F ',' 'if ($1 == 2) {print $3}}' FIT1043_Dataset | head -1"
   This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 2) {print $3}}' FIT1043_Dataset | head -1
```

   Last instance of neutral tweets:
   "awk -F ',' 'if ($1 == 2) {print $3}}' FIT1043_Dataset | tail -1"
   This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 2) {print $3}}' FIT1043_Dataset | tail -1
```

First instance of positive tweets:
"awk -F ',' 'if ($1 == 4) {print $3}}' FIT1043_Dataset | head -1"
This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 4) {print $3}}' FIT1043_Dataset | head -1
Mon Apr 06 22:22:45 PDT 2009
```

Last instance of positive tweets:
"awk -F ',' 'if ($1 == 4) {print $3}}' FIT1043_Dataset | tail -1"
This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 4) {print $3}}' FIT1043_Dataset | tail -1
Tue Jun 16 08:40:50 PDT 2009
```

From here we can know that the date range for the Twitter posts in this file is from April 6th 2009 to June 25th 2009.

7. The mention of a person with the name "Ian" that is not a retweet or a reply-to means that the instance of the data should not contain a "@" symbol while containing the keyword "Ian". Now, we want the date and time of the first instance of the mention "Ian".
This can be done by the following line of code:
"awk -F ',' '{print $3 $6}' FIT1043_Dataset | grep -v "@" | grep "Ian" | head -1"
This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{print $3 $6}' FIT1043_Dataset | grep -v "@" | grep "Ian" | head -1
grep: (standard input): binary file matches
Tue Apr 07 03:43:28 PDT 2009Today I wear Ian's hoodie. I'm tired my allergies are acting up and it's like musical all over again.
```

Then by using the tweet, we can find the user which mentioned it by the following line of code:
"awk -F ',' '{print $5 $6}' FIT1043_Dataset | grep "Today I wear Ian's hoodie. I'm tired my allergies are acting up and it's like musical all over again." | head -1"
This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{print $5 $6}' FIT1043_Dataset | grep "Today I wear Ian's hoodie. I'm tired my allergies are acting up and it's like music
annaOrangeToday I wear Ian's hoodie. I'm tired my allergies are acting up and it's like musical all over again.
```

Therefore, we know that the user annaOrange mentioned the name "Ian" at 03:43:28 PDT on 7th April 2009 with the tweet "Today I wear Ian's hoodie. I'm tired my allergies are acting up and it's like musical all over again.".

8. To find how many tweets have the word "Australia" in the message column of the file, we need to find the line count that has the word "Australia". The message column is the sixth column in the file. To find the number of times that the word "Australia" appears while ignoring the case, the following line of code is used:
"awk -F ',' '{print $6}' FIT1043_Dataset | grep -i "Australia" | wc -l"
This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{print $6}' FIT1043_Dataset | grep -i "Australia" | wc -l
grep: (standard input): binary file matches
1653
```

We can see that there are 1653 tweets of the word "Australia".

9. To find how many times the word "Australia" appeared in the message column, we need to find the word count instead of the line count. Therefore, the following line of code is used:
"awk -F ',' '{print $6}' FIT1043_Dataset | grep -i "Australia" | wc -w"
This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{print $6}' FIT1043_Dataset | grep -i "Australia" | wc -w
grep: (standard input): binary file matches
27423
```

We can see that there are 27423 times the word “Australia” appeared in the message column of the file.

10. If we only want the word “Australia” and not other combinations such as “Australian”, we need to add the -o operation to the grep function. Thus, the following line of code is used:
“awk -F ',' '{print $6}' FIT1043_Dataset | grep -o "Australia" -i | wc -w”
This is the resulting output:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{print $6}' FIT1043_Dataset | grep -o "Australia" -i | wc -w
1691
```

We can see that there are 1691 appearances of the word “Australia” when other combinations are not selected.

11. To determine which country is more popular, the number of tweets the country name appeared in is a good indicator. *“\<Australia”* indicates getting the word that starts with Australia while *“Australia\>”* indicates getting the word that ends with Australia, therefore we can get the exact word we need. We need the following lines of code for Australia and India respectively:

“awk -F ',' '{print $6}' FIT1043_Dataset | grep "\<Australia\>" | wc -l”
The resulting output is:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{print $6}' FIT1043_Dataset | grep "\<Australia\>" | wc -l
grep: (standard input): binary file matches
870
```

“awk -F ',' '{print $6}' FIT1043_Dataset | grep -o "\<India\>" | wc -l”

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{print $6}' FIT1043_Dataset | grep "\<India\>" | wc -l
grep: (standard input): binary file matches
382
```

We can see that Australia has more appearances in tweets than India, thus we can deduce that Australia is more frequently talked about amongst the people in the world and is more popular than India.

12. To determine how many unique users that mentioned the country “India”, I selected all the tweets using *“grep”* and *“\<” “\>”* of the file, then pass it into the user name column (fifth column) to check for unique occurrences. Then I added the *“sort”* pipeline to put repetitive usernames together, and the *“uniq”* pipeline to filter out repeating users. The following line of code is used to do all in one line:
“grep “\<India\>” FIT1043_Dataset | awk -F ‘,’ ‘{print $5}’ | sort | uniq | wc -l”
The resulting output is:

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ grep "\<India\>" FIT1043_Dataset | awk -F ',' '{print $5}' | sort | uniq | wc -l
grep: FIT1043_Dataset: binary file matches
358
```

We can see that the number of unique users that mentioned the country “India” is 358.

13. As the first column indicates the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive), I can determine the number of positive, neutral, and negative tweets by using an *“if”* condition on the first column. After that, I filter out the tweets only containing the word “Australia” and “India” separately. The following lines of code are used and their respective output is attached right below them:
“awk -F ‘,’ ‘{if ($1 == 0) {print $6} }’ FIT1043_Dataset | grep “\<Australia\>” -i | wc -l”

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 0) { print $6 } }' FIT1043_Dataset | grep "\<Australia\>" -i | wc -l
grep: (standard input): binary file matches
612
```

"awk -F ',' '{if ($1 == 2) {print $6} }' FIT1043_Dataset | grep "\<Australia\>" -i | wc -l"

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 2) { print $6 } }' FIT1043_Dataset | grep "\<Australia\>" -i | wc -l
0
```

"awk -F ',' '{if ($1 == 4) {print $6} }' FIT1043_Dataset | grep "\<Australia\>" -i | wc -l"

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 4) { print $6 } }' FIT1043_Dataset | grep "\<Australia\>" -i | wc -l
671
```

"awk -F ',' '{if ($1 == 0) {print $6} }' FIT1043_Dataset | grep "\<India\>" -i | wc -l"

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 0) { print $6 } }' FIT1043_Dataset | grep "\<India\>" -i | wc -l
grep: (standard input): binary file matches
289
```

"awk -F ',' '{if ($1 == 2) {print $6} }' FIT1043_Dataset | grep "\<India\>" -i | wc -l"

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 2) { print $6 } }' FIT1043_Dataset | grep "\<India\>" -i | wc -l
0
```

"awk -F ',' '{if ($1 == 4) {print $6} }' FIT1043_Dataset | grep "\<India\>" -i | wc -l"

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{if ($1 == 4) { print $6 } }' FIT1043_Dataset | grep "\<India\>" -i | wc -l
210
```

Then, I need to save these values along with the type of tweet they indicate into the files *sentiment-australia.csv* and *sentiment-india.csv* respectively.
The format is :

```
Negative, 99
Neutral, 99
Positive, 99
```

Therefore, I use the *"echo"* command to print "Negative", then pass the *"awk -F ',' '{if ($1 == 0) {print $6} }' FIT1043_Dataset | grep "\<Australia\>" -i | wc -l"* into backticks to get the value after execution. After that, I save the two values into a newly created file named sentiment-australia.csv using the ">" operator.

This is the code that I used:
"echo "Negative", `awk -F ',' '{if ($1 == 0) { print $6 } }' FIT1043_Dataset | grep "\<Australia\>" -I | wc -l > sentiment-australia.csv"

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ echo "Negative", `awk -F ',' '{if ($1 == 0) { print $6 } }' FIT1043_Dataset | grep "\<Australia\>" -i | wc -l
` > sentiment-australia.csv
grep: (standard input): binary file matches
```

After the file has been created, I used the same method to print "Neutral" and "Positive", and used "2" to filter for neutral tweets and "4" to filter for positive tweets. After that, I used the ">>" operator to assign the new values into the already created file.
The following lines of code are used:
"echo "Neutral", `awk -F ',' '{if ($1 == 2) { print $6 } }' FIT1043_Dataset | grep "\<Australia\>" -I | wc -l >>sentiment-australia.csv"
"echo "Positive", `awk -F ',' '{if ($1 == 4) { print $6 } }' FIT1043_Dataset | grep "\<Australia\>" -I | wc -l >> sentiment-australia.csv"

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ echo "Neutral", `awk -F ',' '{if ($1 == 2) { print $6 } }' FIT1043_Dataset | grep "\<Australia\>" -i | wc -l`
 >> sentiment-australia.csv

User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ echo "Positive", `awk -F ',' '{if ($1 == 4) { print $6 } }' FIT1043_Dataset | grep "\<Australia\>" -i | wc -l`
 >> sentiment-australia.csv
```

The same is done for India using *"grep "\<India\>""* and saved into the file *sentiment-india.csv*.
The following lines of code are used:
"echo "Negative", `awk -F ',' '{if ($1 == 0) { print $6 } }' FIT1043_Dataset | grep "\<India\>" -I | wc -l > sentiment-india.csv"

“echo “Neutral”, `awk -F ‘,’ ‘{if ($1 == 2) { print $6 } }’ FIT1043_Dataset | grep “\<India\>” -I | wc -l > sentiment-india.csv”
“echo “Positive”, `awk -F ‘,’ ‘{if ($1 == 4) { print $6 } }’ FIT1043_Dataset | grep “\<India\>” -I | wc -l > sentiment-india.csv”

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ echo "Negative", `awk -F ',' '{if ($1 == 0) { print $6 } }' FIT1043_Dataset | grep "\<India\>" -i | wc -l` >
sentiment-india.csv
grep: (standard input): binary file matches

User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ echo "Neutral", `awk -F ',' '{if ($1 == 2) { print $6 } }' FIT1043_Dataset | grep "\<India\>" -i | wc -l` >>
sentiment-india.csv

User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ echo "Positive", `awk -F ',' '{if ($1 == 4) { print $6 } }' FIT1043_Dataset | grep "\<India\>" -i | wc -l` >>
 sentiment-india.csv
```

Then, I used the *“cat”* command to show the values in both files and compared them:
“cat sentiment-australia.csv”
“cat sentiment-india.csv”

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ cat sentiment-australia.csv
Negative, 612
Neutral, 0
Positive, 671

User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ cat sentiment-india.csv
Negative, 289
Neutral, 0
Positive, 210
```

As Australia has more positive tweets than negative tweets and India has more negative tweets than positive tweets, we can conclude that Australia has a higher positive-to-neutral tweet count ratio than India, thus it has more positive tweets related to it.

Part B:

1. i. First, I extracted the timestamps of the tweets referring to "Australia" into a file named *timestamps_aus.txt*.
   This is done by using the *"grep"* command to filter out tweets with "Australia" and the *"-i"* function is passed in to ignore the case. After that, I used the *"cut -d ' '"* command to remove the delimiter, and selected the timestamps only using *"-f 1-6"* for the **day of the week, month, day, time, time zone, year**. Then, I saved it into the file *timestamps_aus.txt* using the *">"* command.
   The following code is used:
   "awk -F ',' '{print $3,$6}' FIT1043_Dataset | grep "\<Australia\>" -I | cut -d ' ' -f 1-4,6 > timestamps_aus.csv"

```
User@DESKTOP-Q262TE5 /cygdrive/c/Users/User/Downloads
$ awk -F ',' '{print $3,$6}' FIT1043_Dataset | grep "\<Australia\>" -i | cut -d
' ' -f 1-4,6 > timestamps_aus.csv
grep: (standard input): binary file matches
```

   As the PDT time-zone is not recognized by R, I omitted it from the csv file to prevent errors on task ii.

   ii. In R, first I read the file and store it in a variable *df* with the following code:
   "df <- read.csv("timestamps_aus.csv", header=FALSE)"

   Then, I determine what format to parse into the *strptime()* function by observing the values of my data:
   "head(df)"
   This is the output:

```
                        V1
Mon Apr 06 23:49:29 2009
Mon Apr 06 23:55:14 2009
Tue Apr 07 01:16:43 2009
Tue Apr 07 03:06:17 2009
Tue Apr 07 03:21:47 2009
Tue Apr 07 03:58:25 2009
```

   I can determine that the string value has the format of (abbreviated_weekday abbreviated_month day_of_month hours:minutes:seconds year_with_century)
   Which are (%a %b %d %H:%M:%S %Y) respectively.

   After that, I converted the text values into timestamps and stored the result in another variable *timestamps* using the code:
   "timestamps <- strptime(df$V1, "%a %b %d %H:%M:%S %Y")"
   The result is shown as follows:

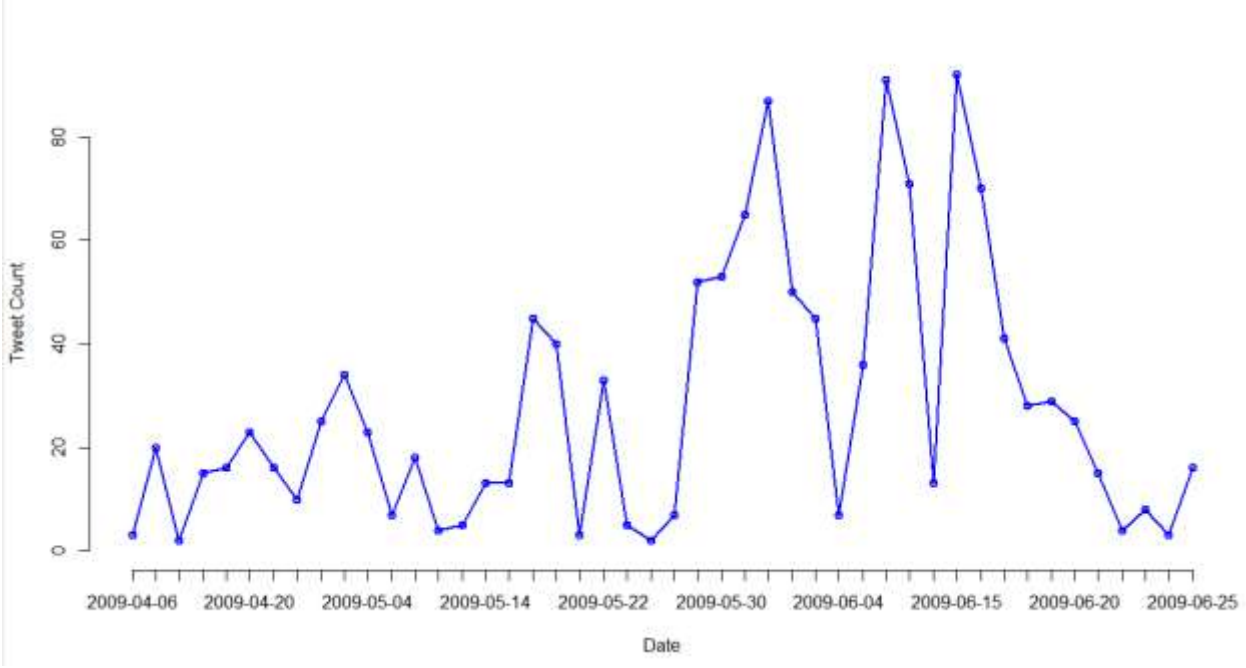
```
> head(df)
[1] "2009-04-06 23:49:29 +08" "2009-04-06 23:55:14 +08" "2009-04-07 01:16:43 +08"
[4] "2009-04-07 03:06:17 +08" "2009-04-07 03:21:47 +08" "2009-04-07 03:58:25 +08"
```

   I have successfully converted the string values into timestamps.

   iii. Using the data I have, I saved the timestamps as a date only format into the variable *dateonly.* Then, I count the number of occurrences of unique dates by using the *"table"* function, and saved that into a variable *tweetcount*.
   The code is as follows:
   dateonly <- format(timestamps, format="%Y%-%m-%d")
   tweetcount = table(dateonly)

   After that, I plot a simple graph using the *tweetcount* variable.
   The following code is used:
   plot(tweetcount, type="o",col ="blue", xlab="Date", ylab="Tweet Count")

The result is as below:



2. a) First, since the original file is too big to be fully stored in a csv file, I sampled a random 5% of the *FIT1043_Dataset* file and stored in in a *tweets.csv* file.
The following code is used:
"cat FIT1043_Dataset | awk 'BEGIN {srand()} !/^$/ { if (rand() <= .05) print $0}' > tweets.csv"

Then, I read the file in R, and stored the first column (polarity of the tweet) in the *polarity* variable and the last column (the tweet) in the *tweet* variable.
The following lines of code are used:
"dataset <- read.csv("tweets.csv",header=FALSE)"
"polarity <- dataset[1]"
"tweet <- dataset[6]"

For the Naïve Bayes algorithm to run effectively, the test data needs to be transformed into a volatile corpus (text that is efficient for text mining). The "tm" package in R is needed to do so.
The following code is used to install and import the package:
"install.packages("tm")"
"library(tm)"

Then, the data of the tweets is passed in to the VCorpus(VectorSource()) function, which converts the tweets into a volatile corpus, then it is stored in a variable *tweet_corpus*.
"tweet_corpus <- VCorpus(VectorSource(dataset$V6))"

Now, the tweets are standardized (in this case I convert it all to lower case letters) as the words "LIKE", "Like" and "like" should be treated as the same word. This is done by using the "tm_map" function in R, and the "content_transformer" function to transform the tweets in that way.

After that, the tweets have the numbers and punctuation removed as they do not determine the polarity of the tweet. Some words that appear often but do not contribute to the polarity of the tweet are also removed (such as "to", "and", "but" and "or").
"tweet_corpus_clean <- tm_map(tweet_corpus_clean, removeNumbers)"
"tweet_corpus_clean <- tm_map(tweet_corpus_clean, removeWords, stopwords())"
"tweet_corpus_clean <- tm_map(tweet_corpus_clean, removePunctuation)"

Next, I strip the suffix from words like “sleeping” to “sleep”, so that the same words with different suffix are converted to the base word. This can be performed using the “tm” package with help from the “SnowballC” package.
The following code is used to install and import the package:
“install.packages(“SnowballC”)”
“library(SnowballC)”

Then, the suffixes are stripped and the whitespaces are removed.
“tweet_corpus_clean <- tm_map(tweet_corpus_clean, stemDocument)”
“tweet_corpus_clean <- tm_map(tweet_corpus_clean, stripWhitespace)”

Then, I converted the corpus back into a dataframe to pass into the Naïve Bayes Algorithm training model later on.
“tweet_df <- data.frame(text=sapply(tweet_corpus_clean, as.character), stringsAsFactors = FALSE)”

b) Next, I split the training and testing datasets into 70% for training and 30% for testing. This is done with the help of the “caret” package, which maintains the same percentage of event rate in both training and testing datasets.
The following code is used to install and import the package:
“install.packages(“caret”)”
“library(caret)”

A seed is set to ensure the same results of randoming.
“*set.seed(123)”*

Then the following code is used to split the datasets into training and testing parts:
“train_index <- createDataPartition(dataset$V1, p=.7,list=FALSE, times=1)”
“X_train <- tweet_df[train_index,]”
“X_test <- tweet_df[-train_index,]”
“Y_train <- polarity[train_index,]”
“Y_test <- polarity[-train_index,]”

c) The “e1071” package is needed to implement the Naïve Bayes algorithm to train the model.
The following code is used to install and import the package:
“install.packages(“e1071”)”
“library(e1071)”

A Naïve Bayes algorithm model is trained with the code:
“NBmodel <- naiveBayes(X_train, Y_train)”

d) The model testing is conducted and the prediction values are stored in a variable *Y_pred*. The following code is used:
“Y_pred <- predict(NBmodel, X_test)”

e) Then, I created a confusion matrix using the predictions and test data.
"cm <- table(Y_test, Y_pred)"
"confusionMatrix(cm)"

This is the following output:
Confusion Matrix and Statistics

```
     Y_pred
Y_test     0     4
    0     0 10983
    4     0 11159

          Accuracy : 0.504
            95% CI : (0.4974, 0.5106)
   No Information Rate : 1
   P-Value [Acc > NIR] : 1

             Kappa : 0

 Mcnemar's Test P-Value : <2e-16

         Sensitivity :    NA
         Specificity : 0.504
      Pos Pred Value :    NA
      Neg Pred Value :    NA
          Prevalence : 0.000
      Detection Rate : 0.000
  Detection Prevalence : 0.496
     Balanced Accuracy :    NA

      'Positive' Class : 0
```

We can see that the model is not built well as it only returns positive for all the tweets. It might be because of it only being passed through a Naïve Bayes Algorithm of using certain words to determine the polarity of a tweet. I think that the model does not know which words determine the positivity and negativity of a tweet, thus it returns all as positive.

This concludes the end of my FIT1043 Assignment 3.

Sources

solution of Task B 1ii referenced from - r - Using strptime on NA values - Stack Overflow

solution of Task B 2 referenced from Naive Bayes Machine Learning (rstudio-pubs-static.s3.amazonaws.com), Naive Bayes Classifier in R Programming - GeeksforGeeks

converting Vcorpus back to dataframe referenced from text mining - Unable to convert a Corpus to Data Frame in R - Stack Overflow