FIT1043 Introduction to Data Science Assignment 3

Michelle Adeline
31989101

Introduction

The purpose of this report is to use BASH Shell to investigate, and manipulate large amounts of data obtained from Facebook posts originating from top mainstream media sources between the years of 2012 to 2016, and to write the results of our shell operations to a text or csv file from which we can import and use to plot various graphs in R.

The report has the following outline:

1. Introduction
2. BASH Shell
   Part A: Investigating Facebook Data
3. R
   Part B: Graphing Data in R
4. Conclusion
5. References

In this report, all code is indented to separate it from the text.

Note that I am using a Windows machine therefore I am using the Cygwin terminal for part 2.

Part A: Investigating Facebook Data using shell commands

Copying Dataset Into the Working Directory

Before we can begin to operate on the data, we must first copy it from its current location (for me this is in the Downloads folder) to the working directory.

We can do this using the `cp` command, which is used to copy files, groups of files or directories from one location to another. Here we are only copying one file, FB_Dataset.gz, which has the path, however Cygwin only recognizes this location if we use forward slashes (/) instead of backslashes (\). Therefore, the path we feed into cp must be: C:/Downloads/Lenovo/FB_Dataset.gz.

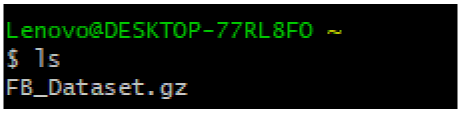
Note that this path will be different for your computer.

To specify the destination as the working directory, we can simply insert a period (.)

```
cp C:/Users/Lenovo/Downloads/FB_Dataset.gz .
```

We can check that the file has now been copied into the working directory by using the `ls` command which displays all the files and directories in our working directory.

```
ls
```



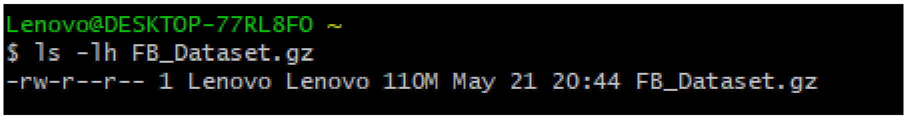
```
Lenovo@DESKTOP-77RL8FO ~
$ ls
FB_Dataset.gz
```

Now that we are certain that the dataset is in our working directory, we can now refer to this file simply as FB_Dataset.gz, and with this we can begin investigation.

Question 1

In order to find the file size of the original dataset, we can use the ls command along with the -lh option on the file FB_Dataset.gz to display the details of the dataset, which includes its file size (and since we specified h, which stands for "human-readable", the size will be represented in megabytes).

```
ls -lh FB_Dataset.gz
```



```
Lenovo@DESKTOP-77RL8FO ~
$ ls -lh FB_Dataset.gz
-rw-r--r-- 1 Lenovo Lenovo 110M May 21 20:44 FB_Dataset.gz
```

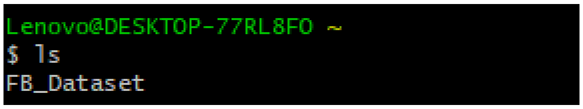
Here we can see that after "Lenovo", it outputs the size of FB_Dataset.gz, therefore the original file size is 110 megabytes.

Notice that the file has the extension .gz, which denotes that it is a compressed file generated through gzip compression. We can thus decompress the file using the `gunzip` command on FB_Dataset.gz, which then replaces the compressed file with the uncompressed file.

```
gunzip FB_Dataset.gz
```

We can now use the `ls` command once more to see the result of the decompression.

```
ls
```

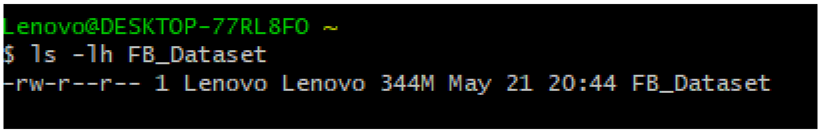


```
Lenovo@DESKTOP-77RL8FO ~
$ ls
FB_Dataset
```

Notice that the .gz extension is no longer present, the file has been successfully decompressed!

In order to find the size of the now-uncompressed file, we can again use the `ls` command with the -lh option on FB_Dataset, to display the details of this file.

```
ls -lh FB_Dataset
```



```
Lenovo@DESKTOP-77RL8FO ~
$ ls -lh FB_Dataset
-rw-r--r-- 1 Lenovo Lenovo 344M May 21 20:44 FB_Dataset
```

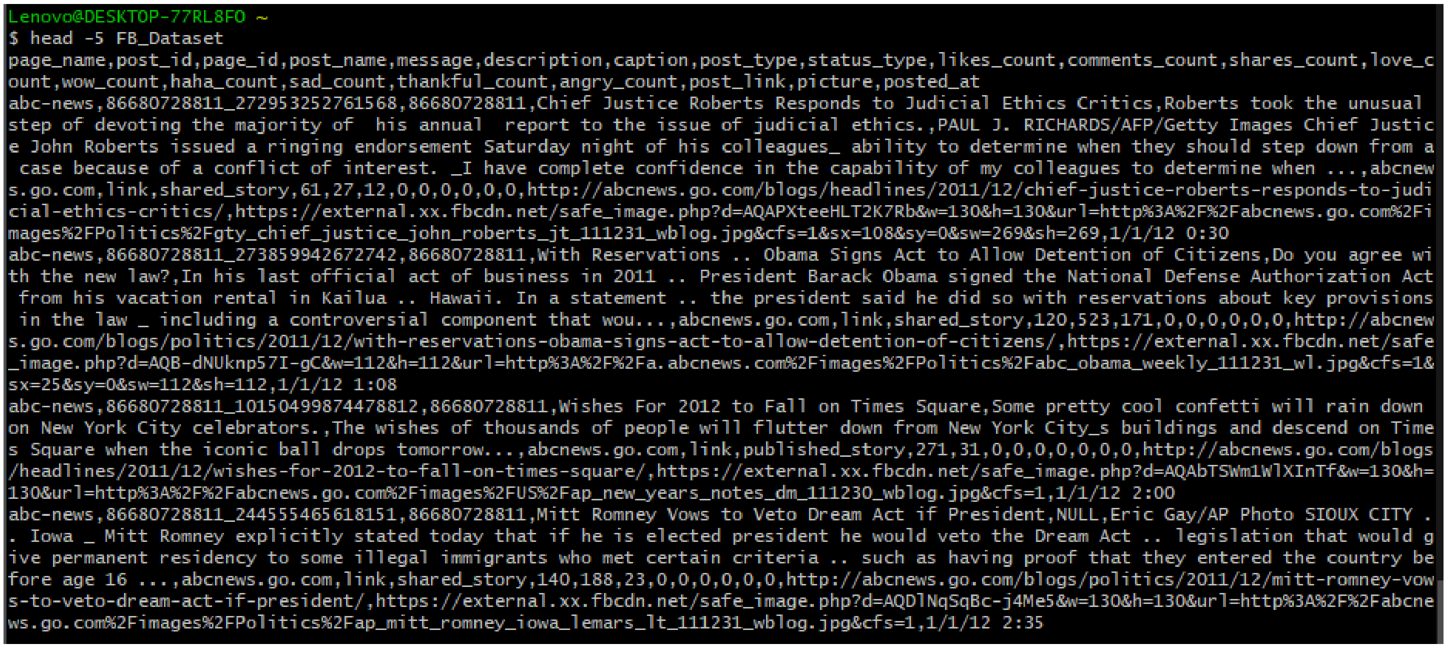
Next to “Lenovo”, we can see that the decompressed file, FB_Dataset has a file size of 344 megabytes.

Comparing this size with the original file size, we can calculate the compression rate (uncompressed / compressed * 100) to be 344/110 * 100 which comes out to be approximately 313%.

Question 2

In order to figure out the delimiter used in the file, we can begin by looking at its contents. Let’s print the first 5 lines of the file by using the head command on FB_Dataset and specifying the number of lines we want with -5.

```
head -5 FB_Dataset
```

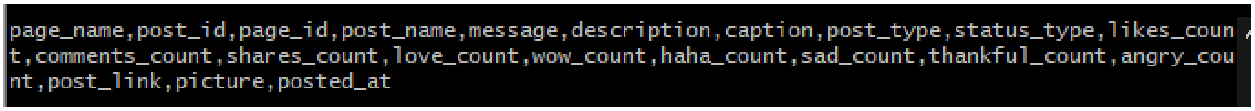


```
Lenovo@DESKTOP-77RL8FO ~
$ head -5 FB_Dataset
page_name,post_id,page_id,post_name,message,description,caption,post_type,status_type,likes_count,comments_count,shares_count,love_c
ount,wow_count,haha_count,sad_count,thankful_count,angry_count,post_link,picture,posted_at
abc-news,86680728811_272953252761568,86680728811,Chief Justice Roberts Responds to Judicial Ethics Critics,Roberts took the unusual
step of devoting the majority of  his annual  report to the issue of judicial ethics.,PAUL J. RICHARDS/AFP/Getty Images Chief Justic
e John Roberts issued a ringing endorsement Saturday night of his colleagues_ ability to determine when they should step down from a
 case because of a conflict of interest. _I have complete confidence in the capability of my colleagues to determine when ...,abcnew
s.go.com,link,shared_story,61,27,12,0,0,0,0,0,0,http://abcnews.go.com/blogs/headlines/2011/12/chief-justice-roberts-responds-to-judi
cial-ethics-critics/,https://external.xx.fbcdn.net/safe_image.php?d=AQAPXteeHLT2K7Rb&w=130&h=130&url=http%3A%2F%2Fabcnews.go.com%2Fi
mages%2FPolitics%2Fgty_chief_justice_john_roberts_jt_111231_wblog.jpg&cfs=1&sx=108&sy=0&sw=269&sh=269,1/1/12 0:30
abc-news,86680728811_273859942672742,86680728811,With Reservations .. Obama Signs Act to Allow Detention of Citizens,Do you agree wi
th the new law?,In his last official act of business in 2011 .. President Barack Obama signed the National Defense Authorization Act
 from his vacation rental in Kailua .. Hawaii. In a statement .. the president said he did so with reservations about key provisions
 in the law _ including a controversial component that wou...,abcnews.go.com,link,shared_story,120,523,171,0,0,0,0,0,0,http://abcnew
s.go.com/blogs/politics/2011/12/with-reservations-obama-signs-act-to-allow-detention-of-citizens/,https://external.xx.fbcdn.net/safe
_image.php?d=AQB-dNUknp57I-gC&w=112&h=112&url=http%3A%2F%2Fa.abcnews.com%2Fimages%2FPolitics%2Fabc_obama_weekly_111231_wl.jpg&cfs=1&
sx=25&sy=0&sw=112&sh=112,1/1/12 1:08
abc-news,86680728811_10150499874478812,86680728811,Wishes For 2012 to Fall on Times Square,Some pretty cool confetti will rain down
on New York City celebrators.,The wishes of thousands of people will flutter down from New York City_s buildings and descend on Time
s Square when the iconic ball drops tomorrow...,abcnews.go.com,link,published_story,271,31,0,0,0,0,0,0,0,http://abcnews.go.com/blogs
/headlines/2011/12/wishes-for-2012-to-fall-on-times-square/,https://external.xx.fbcdn.net/safe_image.php?d=AQAbTSWm1WlXInTf&w=130&h=
130&url=http%3A%2F%2Fabcnews.go.com%2Fimages%2FUS%2Fap_new_years_notes_dm_111230_wblog.jpg&cfs=1,1/1/12 2:00
abc-news,86680728811_244555465618151,86680728811,Mitt Romney Vows to Veto Dream Act if President,NULL,Eric Gay/AP Photo SIOUX CITY .
. Iowa _ Mitt Romney explicitly stated today that if he is elected president he would veto the Dream Act .. legislation that would g
ive permanent residency to some illegal immigrants who met certain criteria .. such as having proof that they entered the country be
fore age 16 ...,abcnews.go.com,link,shared_story,140,188,23,0,0,0,0,0,0,http://abcnews.go.com/blogs/politics/2011/12/mitt-romney-vow
s-to-veto-dream-act-if-president/,https://external.xx.fbcdn.net/safe_image.php?d=AQDlNqSqBc-j4Me5&w=130&h=130&url=http%3A%2F%2Fabcne
ws.go.com%2Fimages%2FPolitics%2Fap_mitt_romney_iowa_lemars_lt_111231_wblog.jpg&cfs=1,1/1/12 2:35
```

This is not entirely too helpful in figuring out which delimiter is used as there is simply too many characters and the whole thing is a bit of a mess. However, something very important to notice here is that the first record (first two lines in this case as it wraps to the second line) seems to contain header names.

Let's take a closer look by only printing out the first line of the file using the head command on FB_Dataset, specifying with the -1 option, except instead of directly printing out this line, we'll pipe it into the less command, using | which means that we are passing the output from the head command to be the input of the less command that simply allows us to view the input one screen at a time.

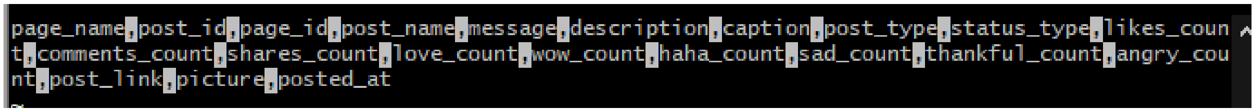
```
head -1 FB_Dataset | less
```



```
page_name,post_id,page_id,post_name,message,description,caption,post_type,status_type,likes_coun
t,comments_count,shares_count,love_count,wow_count,haha_count,sad_count,thankful_count,angry_cou
nt,post_link,picture,posted_at
```

With this it is quite clear that the first line of the file contains the column headers. We can also see that the header names seem to be separated by commas (,). To make it easier to locate the commas, we can type "/," and press Enter, to search for all the commas in the first line.

```
/,
```

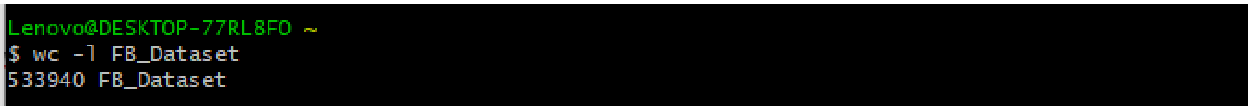


```
page_name,post_id,page_id,post_name,message,description,caption,post_type,status_type,likes_coun
t,comments_count,shares_count,love_count,wow_count,haha_count,sad_count,thankful_count,angry_cou
nt,post_link,picture,posted_at
```

Now we know that the file uses commas (,) as delimiters to separate the columns in the dataset.

In order to find out the number of rows in the dataset we can simply count the number of lines in the file. We can do this using the wc command, specifying that we want to count the number of lines with the option -l, on FB_Dataset.

```
wc -l FB_Dataset
```



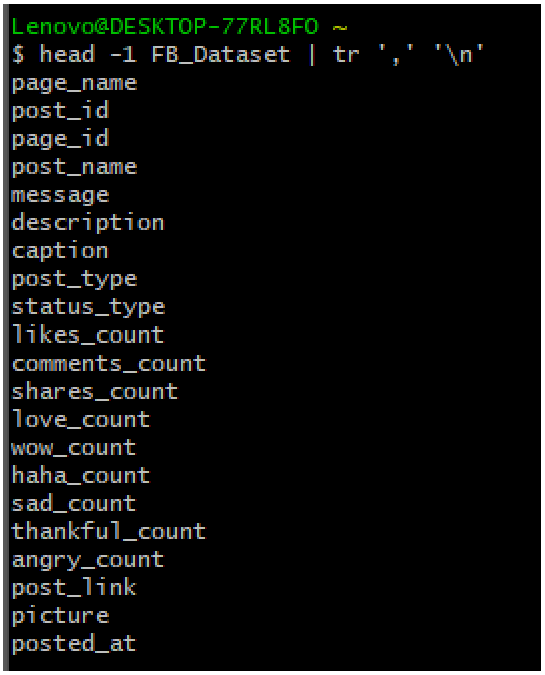
```
Lenovo@DESKTOP-77RL8FO ~
$ wc -l FB_Dataset
533940 FB_Dataset
```

We can see that there are 533,940 lines and thus 533,940 rows (including the header row) in the file, FB_Dataset.

Question 3

We can directly print out the columns by printing the first line of the file using the head command and the -1 option on FB_Dataset however the output is not very easy to read, therefore, we can pipe the output from this head command to the tr command to replace the comma characters (,) in the line with newline characters (\n) so that instead of being concatenated and separated by commas, the column headers are printed on separate lines.

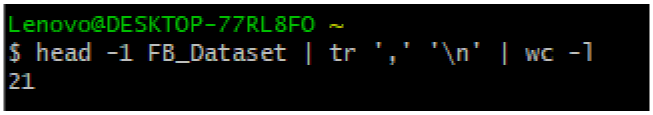
```
head -1 FB_Dataset | tr ',' '\n'
```



```
Lenovo@DESKTOP-77RL8FO ~
$ head -1 FB_Dataset | tr ',' '\n'
page_name
post_id
page_id
post_name
message
description
caption
post_type
status_type
likes_count
comments_count
shares_count
love_count
wow_count
haha_count
sad_count
thankful_count
angry_count
post_link
picture
posted_at
```

To count the number of columns we can pipe this output to the wc command, specifying that we want to count the number of lines using -l, since we have now made it so that the the column names are printed on separate lines.

```
head -1 FB_Dataset | tr ',' '\n' | wc -l
```

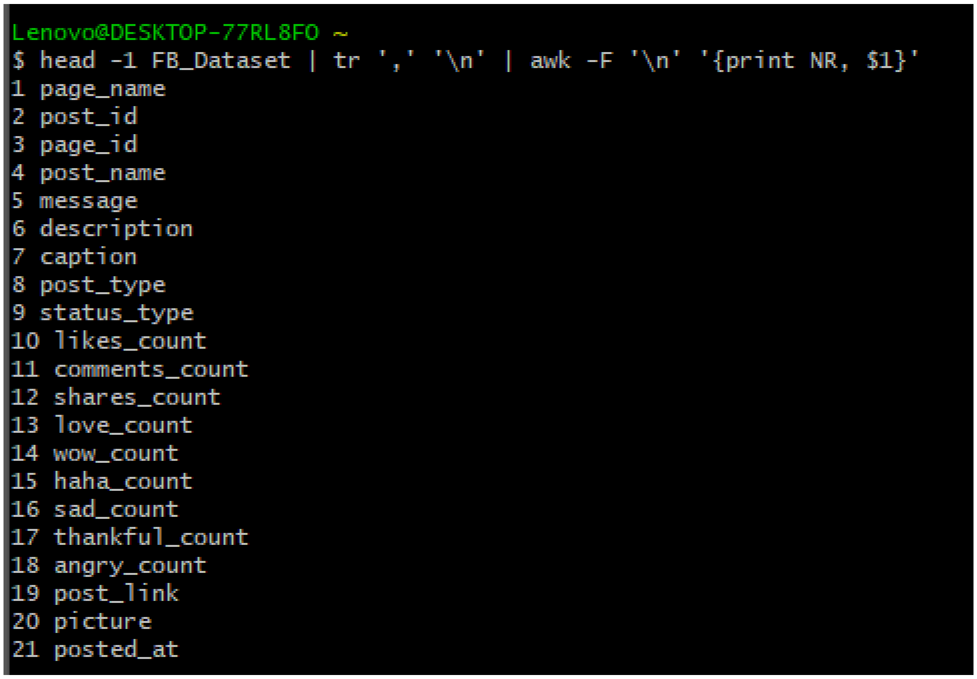


```
Lenovo@DESKTOP-77RL8FO ~
$ head -1 FB_Dataset | tr ',' '\n' | wc -l
21
```

So we can see that there are 21 columns in the file, FB_Dataset.

To aid in answering the following questions, I will also print the column/field number of each column name beside it for future reference. This can be done by passing the output from the tr command to the awk command and here we have to specify the field delimiters, which is the newline character (\n) after the translation and print the record/line number along with the column name.

```
head -1 FB_Dataset | tr ',' '\n' | awk -F '\n' '{print NR, $1}'
```



```
Lenovo@DESKTOP-77RL8FO ~
$ head -1 FB_Dataset | tr ',' '\n' | awk -F '\n' '{print NR, $1}'
1 page_name
2 post_id
3 page_id
4 post_name
5 message
6 description
7 caption
8 post_type
9 status_type
10 likes_count
11 comments_count
12 shares_count
13 love_count
14 wow_count
15 haha_count
16 sad_count
17 thankful_count
18 angry_count
19 post_link
20 picture
21 posted_at
```

This allows me to easily map the human readable column names with its associated field number which we will use in the code from this point onwards.

Question 4

To count the number of unique pages, we can count the number of unique entries in the field page_id, as the term id is typically used as a key which means that it is usually a unique identifier.

This can be done by using the awk command on FB_Dataset to print only the values in the third column ({print $3}) (which we know to be the page_id column from Question 3), skipping the first line (NR > 1) as we do not want to count the column header as a unique page. We should also specify that the file is delimited by commas (,) with -F ',' so that it knows where to separate the fields.

The output from the awk command has to be sorted before passing it to the uniq command, as this command outputs the unique entries in a column by removing duplicate lines of data. This is problematic if the duplicate lines are not adjacent to each other, i.e. unsorted.

For example, given the file with the following three lines:

One
One
Two

The uniq command will match the first line with the second line and since they are duplicates, it removes one of them and compares with the next line which is 'Two'. Both lines are distinct therefore both are kept, and the result of the uniq command will be:

One
Two

Whereas, given the file with the following three lines:

One
Two
One

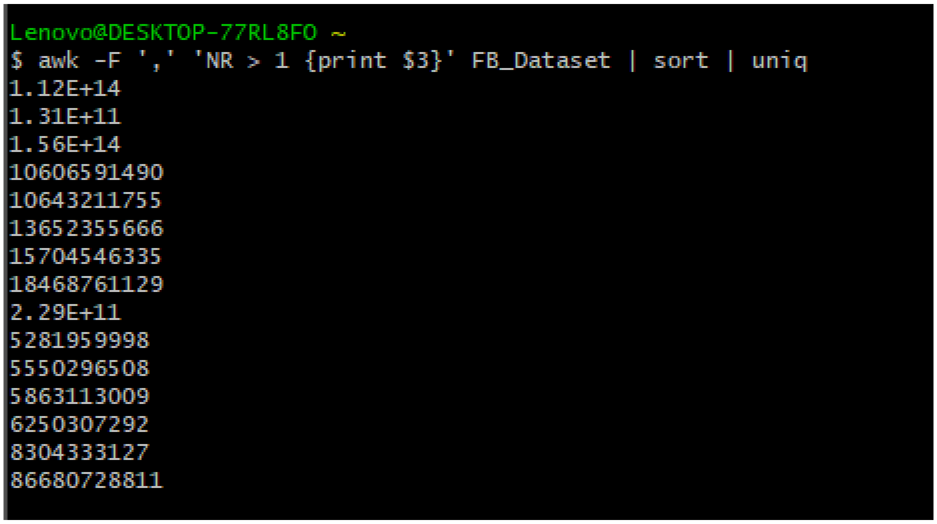
The uniq command will match the first line with the second line and since they are not duplicates, both lines are kept, now it compares 'Two' with 'One' and again, both lines are distinct, therefore both are kept and the result of the uniq command will be:

One
Two
One

Which is plainly wrong as 'One' is repeated twice in this so-called unique list.

Now we can see why we must pre-sort the page_ids to ensure that the uniq command works properly. Here we can see the various unique page ids.

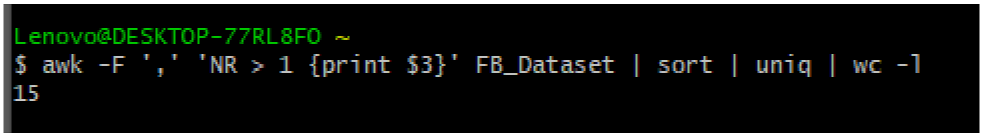
```
awk -F ',' 'NR > 1 {print $3}' FB_Dataset | sort | uniq
```



```
Lenovo@DESKTOP-77RL8FO ~
$ awk -F ',' 'NR > 1 {print $3}' FB_Dataset | sort | uniq
1.12E+14
1.31E+11
1.56E+14
10606591490
10643211755
13652355666
15704546335
18468761129
2.29E+11
5281959998
5550296508
5863113009
6250307292
8304333127
86680728811
```

To count the number of unique pages, we can pipe this output to the wc command with -l to count the number of lines as the page ids are printed on separate lines.

```
awk -F ',' 'NR > 1 {print $3}' FB_Dataset | sort | uniq | wc -l
```



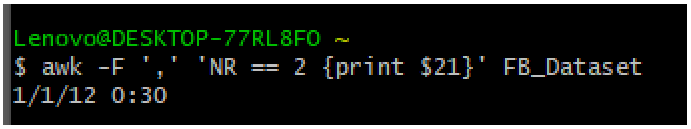
```
Lenovo@DESKTOP-77RL8FO ~
$ awk -F ',' 'NR > 1 {print $3}' FB_Dataset | sort | uniq | wc -l
15
```

Question 5

To find the range of dates for the Facebook posts in the dataset, we must investigate the posted_at column, which is presumably the date at which the post was created/published. Assuming that the data is ordered by date posted in chronological order, to get this range we can print the first entry in the posted_at column to find the earliest Facebook post in the file, and print the last entry in the posted_at column.

We can do this by using the `awk` command on FB_Dataset. `-F ','` Specifies that the delimiter used in the file is a comma (,) and `NR == 2` makes sure that it only selects the second record/line (not the first record as the first line consists of column headers). `{print $21}` tells the command to print the entry in the 21st field, which we know from Question 3 to be the posted_at column.

```
awk -F ',' 'NR == 2 {print $21}' FB_Dataset
```

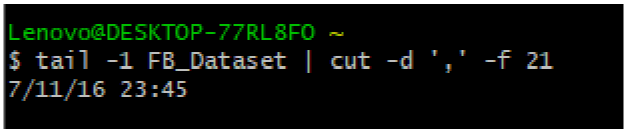


```
Lenovo@DESKTOP-77RL8FO ~
$ awk -F ',' 'NR == 2 {print $21}' FB_Dataset
1/1/12 0:30
```

To find the most recent Facebook post in the dataset, we can begin by using the `tail` command on FB_Dataset, which outputs the lines from the end of the file, with the -1 option telling it to print only the last line of the file.

The output from this command is then piped into the cut command using the | character. The cut command extracts sections from each line of a file, here `-d ','` tells it that the input is being delimited by commas (,), and `-f 21` makes sure the command only outputs the entry from the 21st field (which is the posted_at column).

```
tail -1 FB_Dataset | cut -d ',' -f 21
```



```
Lenovo@DESKTOP-77RL8FO ~
$ tail -1 FB_Dataset | cut -d ',' -f 21
7/11/16 23:45
```

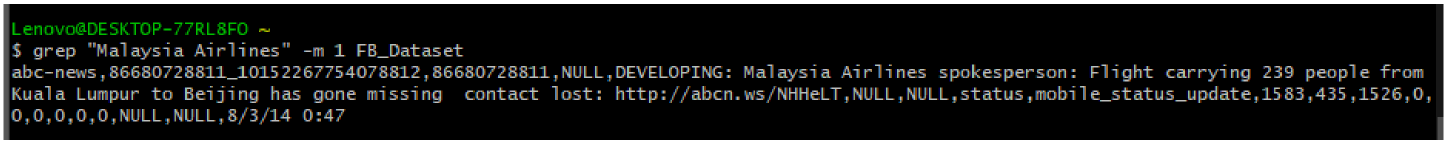
Now that we know when the earliest and latest Facebook post was posted, we know that the date range for Facebook posts in the file, FB_Dataset is from 1st of January 2012 00:30 (1/1/12 0:30) to 7th of November 2016 23:45 (7/11/16 23:45).

Question 6

To look for the first instance of the string "Malaysia Airlines", we can use the grep command on FB_Dataset, which is used to search for text in the file. `"Malaysia Airlines"` specifies the string that we want the command to look for in the file. `-m 1` tells the command to stop after finding the first match.

```
grep "Malaysia Airlines" -m 1 FB_Dataset
```

This outputs



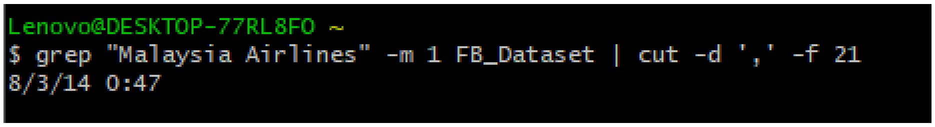
```
Lenovo@DESKTOP-77RL8FO ~
$ grep "Malaysia Airlines" -m 1 FB_Dataset
abc-news,86680728811_10152267754078812,86680728811,NULL,DEVELOPING: Malaysia Airlines spokesperson: Flight carrying 239 people from
Kuala Lumpur to Beijing has gone missing  contact lost: http://abcn.ws/NHHeLT,NULL,NULL,status,mobile_status_update,1583,435,1526,0,
0,0,0,0,0,NULL,NULL,8/3/14 0:47
```

Which is the first entry to mention "Malaysia Airlines".

To extract the date from this entry, we can pass this output to a cut command using the pipe (|) character. The cut command extracts sections from each line of a file, here `-d ','` tells it that the input is being delimited by commas (,), and `-f 21` makes sure the command only outputs the entry from the 21st field (which is the posted_at column).

```
grep "Malaysia Airlines" -m 1 FB_Dataset | cut -d ',' -f 21
```

This outputs



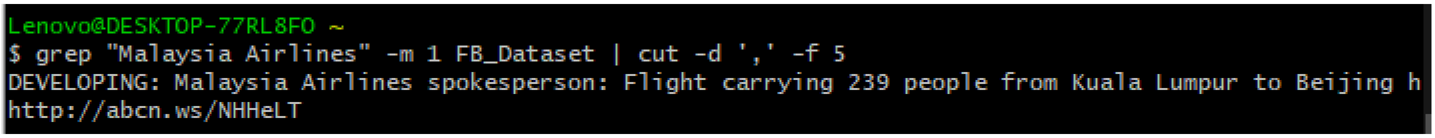
```
Lenovo@DESKTOP-77RL8FO ~
$ grep "Malaysia Airlines" -m 1 FB_Dataset | cut -d ',' -f 21
8/3/14 0:47
```

Therefore, the first mention of “Malaysia Airlines” in the file, FB_Dataset is on the **8th of March 2014 00:47 (8/3/14 0:47).**

To extract the message from the post, we can pass the output of the grep command to a cut command using the pipe (|) character. The cut command extracts sections from each line of a file, here `-d ','` tells it that the input is being delimited by commas (,), and `-f 5` makes sure the command only outputs the entry from the 5th field (which is the message column).

```
grep "Malaysia Airlines" -m 1 FB_Dataset | cut -d ',' -f 5
```

This outputs



```
Lenovo@DESKTOP-77RL8FO ~
$ grep "Malaysia Airlines" -m 1 FB_Dataset | cut -d ',' -f 5
DEVELOPING: Malaysia Airlines spokesperson: Flight carrying 239 people from Kuala Lumpur to Beijing h
http://abcn.ws/NHHeLT
```

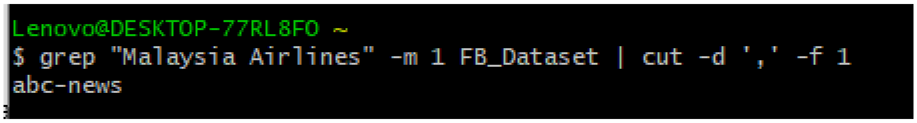
Therefore, the message of the post that first mentioned “Malaysia Airlines” in the file, FB_Dataset is:

DEVELOPING: Malaysia Airlines spokesperson: Flight carrying 239 people from Kuala Lumpur to Beijing hhttp://abcn.ws/NHHeLT

From analyzing the dataset, we can notice that the media source of a post is actually stored in the page_name column. To extract the media source from the post, we can pass the output of the grep command to a cut command using the pipe (|) character. The cut command extracts sections from each line of a file, here `-d ','` tells it that the input is being delimited by commas (,), and `-f 1` makes sure the command only outputs the entry from the 1st field (which is the page_name column).

```
grep "Malaysia Airlines" -m 1 FB_Dataset | cut -d ',' -f 1
```

This outputs



```
Lenovo@DESKTOP-77RL8FO ~
$ grep "Malaysia Airlines" -m 1 FB_Dataset | cut -d ',' -f 1
abc-news
```

Therefore, the media source of the post that first mentioned “Malaysia Airlines” in the file, FB_Dataset is **abc-news**.

Question 7

We can find out how many times “Donald Trump” was mentioned in the message column of the file by first isolating the message column before searching for the string so that we do not accidentally find matches in other columns such as the post_name or description columns.

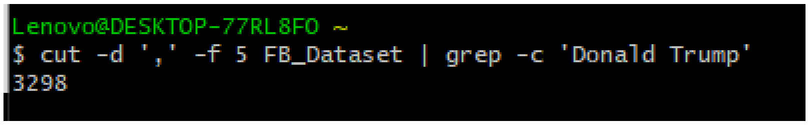
Note: For this purpose, if there are multiple mentions of “Donald Trump” in the message of a post, I will only count it as 1 occurrence because I think that it is more insightful, and in general more useful,l to know the number of unique posts mentioning “Donald Trump” which is a better indicator of, say, how famous or influential he is, compared to the absolute count of the string “Donald Trump”.

We can isolate the message column by using the cut command on FB_Dataset. This command extracts sections from each line of a file, here `-d ','` tells it that the input is being delimited by commas (,), and `-f 5` makes sure the command only outputs the entry from the 5th field (which is the message column).

To count the number of times “Donald Trump” is mentioned in the message column, we can pass the output from the cut command to a grep command using the pipe (|) character. The grep command is used to search for text in the file. `'Donald Trump'` specifies the string that we want the command to look for in the file. Here grep, by default, takes into account upper/lower case when matching with the target string. The -c option suppresses the command’s normal output, which is to print out all the lines that contain the target string, and instead the command prints the number of lines where a match was found.

```
cut -d ',' -f 5 FB_Dataset | grep -c 'Donald Trump'
```

This outputs



```
Lenovo@DESKTOP-77RL8FO ~
$ cut -d ',' -f 5 FB_Dataset | grep -c 'Donald Trump'
3298
```

Therefore, “Donald Trump” was mentioned 3298 times in the message column of the FB_Dataset file.

Question 8

We can find out how many times “Barack Obama” was mentioned in the message column of the file in much the same way that we did for Question 7 - by first isolating the message column before searching for the string so that we do not accidentally find matches in other columns such as the post_name or description columns.

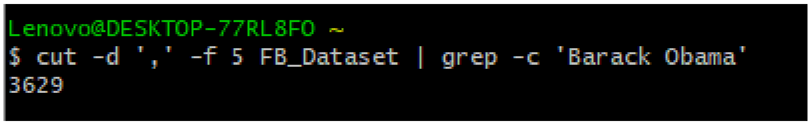
Note: Similar to the previous question, if there are multiple mentions of Barack Obama” in the message of a post, I will only count it as 1 occurrence because I think that it is more insightful to know the number of unique posts mentioning “Barack Obama” which is a better indicator of, say, how famous or influential he is, compared to the absolute count of the string “Barack Obama”.

We can isolate the message column by using the cut command on FB_Dataset. This command extracts sections from each line of a file, here `-d ','` tells it that the input is being delimited by commas (,), and `-f 5` makes sure the command only outputs the entry from the 5th field (which is the message column).

To count the number of times “Barack Obama” is mentioned in the message column, we can pass the output from the cut command to a grep command using the pipe (|) character. The grep command is used to search for text in the file. `'Barack Obama'`specifies the string that we want the command to look for in the file. Here grep, by default, takes into account upper/lower case when matching with the target string. The -c option suppresses the command’s normal output, which is to print out all the lines that contain the target string, and instead the command prints the number of lines where a match was found.

```
cut -d ',' -f 5 FB_Dataset | grep -c 'Barack Obama'
```

This outputs



```
Lenovo@DESKTOP-77RL8FO ~
$ cut -d ',' -f 5 FB_Dataset | grep -c 'Barack Obama'
3629
```

To find out who is more popular on Facebook – Barack Obama or Donald Trump, we must first define what we mean by “popular”. Here, I will interpret popular as something or someone that many people like or support. Therefore, to determine who is more popular we can compare the average number of likes of posts mentioning Barack Obama to those mentioning Donald Trump.

We can obtain these values by isolating the message and likes_count column, searching for either “Barack Obama” (if we’re finding the average number of likes for posts mentioning Obama) or “Donald Trump” (if we’re finding the average number of likes for posts mentioning Trump) and totaling up the entries in the likes_count column of the output and dividing it by the total number of lines/matches found.

Starting with Barack Obama, we must first use the cut command on FB_Dataset. This command extracts sections from each line of a file, here `-d ','` tells it that the input is being delimited by commas (,), and `-f`

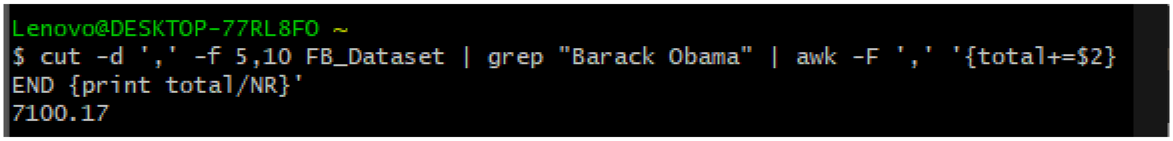
`5,10` makes sure the command only outputs the entries from the 5th and 10th field (which is the message, and likes_count column respectively).

The output from this command is passed to the grep command using the pipe (|) character. The grep command is used to search for text in the file. `'Barack Obama'` specifies the string that we want the command to look for in the file and outputs all of the lines that contain a match.

This output is then passed to the awk command using the pipe (|) character. The awk command is typically used for pattern scanning and to perform some code if a match is found. Here, `-F ','` specifies that the input uses commas (,) as delimiters. `{total+=$2}` creates a variable total and adds each value in the second column (which is now the likes_count column due to the cut command), and `END {print total/NR}` tells it that once the end of the file has been reached, print total divided by the number of records/lines, NR.

```
      cut -d ',' -f 5,10 FB_Dataset | grep "Barack Obama" | awk -F ','
'{total+=$2} END {print total/NR}'
```

This outputs



```
Lenovo@DESKTOP-77RL8FO ~
$ cut -d ',' -f 5,10 FB_Dataset | grep "Barack Obama" | awk -F ',' '{total+=$2}
END {print total/NR}'
7100.17
```

We can do the same procedure to obtain the average number of likes of posts with messages containing "Donald Trump"

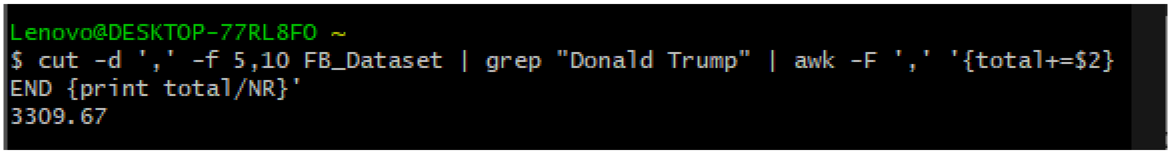
First we can use the cut command on FB_Dataset. This command extracts sections from each line of a file, here `-d ','` tells it that the input is being delimited by commas (,), and `-f 5,10` makes sure the command only outputs the entries from the 5th and 10th field (which is the message, and likes_count column respectively).

The output from this command is passed to the grep command using the pipe (|) character. The grep command is used to search for text in the file. `'Donald Trump'` specifies the string that we want the command to look for in the file and outputs all of the lines that contain a match.

This output is then passed to the awk command using the pipe (|) character. The awk command is typically used for pattern scanning and to perform some code if a match is found. Here, `-F ','` specifies that the input uses commas (,) as delimiters. `{total+=$2}` creates a variable total and adds each value in the second column (which is now the likes_count column due to the cut command), and `END {print total/NR}` tells it that once the end of the file has been reached, print total divided by the number of records/lines, NR.

```
      cut -d ',' -f 5,10 FB_Dataset | grep "Donald Trump" | awk -F ','
'{total+=$2} END {print total/NR}'
```

This outputs



```
Lenovo@DESKTOP-77RL8FO ~
$ cut -d ',' -f 5,10 FB_Dataset | grep "Donald Trump" | awk -F ',' '{total+=$2}
END {print total/NR}'
3309.67
```

Comparing the average number of likes of posts mentioning Barack Obama to those mentioning Donald Trump, we can see that Obama seems to be vastly more popular than Trump, with an average of 7100 likes for every post mentioning Obama compared to half of that, 3310 for posts mentioning Trump.

Question 9

In order to tackle this task, we must first extract the column headers – post_id and likes_count and write it into the file “trump.txt” because they will be filtered out when we eventually search for records that contain the string “Trump”.

We can do this very efficiently by using the awk command on FB_Dataset. This command is typically used for pattern scanning and to perform some code if a match is found. Here, `-F ','` specifies that the input uses commas (,) as delimiters, `NR == 1` tells it to only execute the print if it is the first line, therefore it extracts the first record containing column headers from the rest of the file. From this, it only outputs the 2nd and 10th fields, which are the post_id and likes_count fields respectively. The “,” in the middle ensures that the column headers are still separated by a comma.

The output from this command is saved to a file called ‘trump.txt’. > denotes writing to a file, which overwrites the current contents of the file with the provided output.

```
awk -F ',' 'NR == 1 {print $2","$10}' FB_Dataset > trump.txt
```

Now that we have put the column headers in place, it’s time to fill the file with content. First we can extract the columns required for this task by using the cut command on FB_Dataset. The cut command extracts sections from each line of a file, here -d ',' tells it that the file is being delimited by commas (,), and `-f 2,5,10` makes sure the command only outputs entries from the 2nd, 5th, and 10th field (which are the post_id, message, and likes_count columns respectively).

The output from this command is then passed to the grep command, using the pipe (|) character so that it is able to search for the word ‘Trump’. Here, the -i option specifies that the command should not care about upper/lower case when matching words to the target string, ‘Trump’.

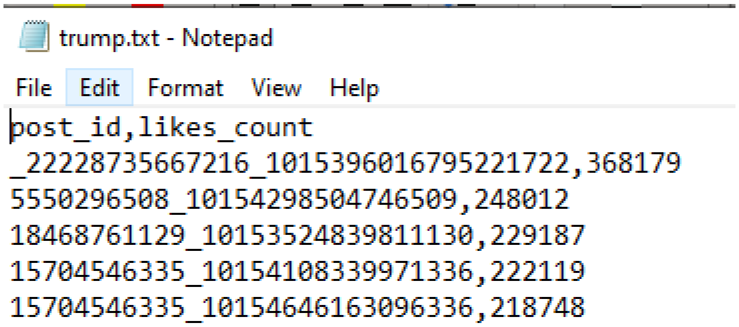
The output, which now consists of posts with messages mentioning Trump, is passed to the awk command, using the pipe (|) character, to be filtered. -F ',' tells it that the input is being delimited by commas (,), `$3 >= 1000` is the condition for the following print statement to be executed, it specifies that if and only if the value in the third column (which is the likes_count column) is more than or equal to 1000, then `{print $1”,”$3}`, which prints the values in the post_id and likes_count columns. The “,” in the middle ensures that these values are still delimited by commas(,). In effect this simply filters out posts with a like count of less than 1000.

This filtered output is now passed to the sort command, using the pipe (|) character, to sort the rows in descending order based on their number of likes. The -t ',' specifies that the field separator is a comma (,), -nrk 2 can be split into -n, which tells it that it will be performing numeric sorting, -r, which sorts the values in descending order, and -k 2, which sorts the values in the 2nd column (likes_count column). This results in the posts being sorted from highest to lowest like count.

This sorted output is finally appended into the 'trump.txt' file. Note that here we are using >> not >, as writing it to the file would overwrite the current contents of the file (removing our column headers), whereas appending it simply adds the records to the end of the file.

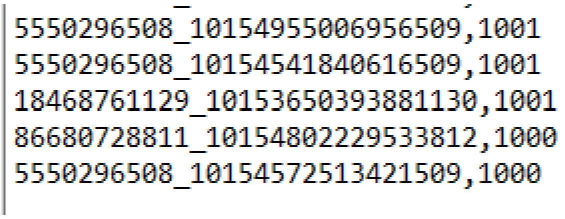
```
     cut -d ',' -f 2,5,10 FB_Dataset | grep -i 'Trump' | awk -F',' '$3 >= 1000
{print $1","$3}' | sort -t ',' -nrk 2 >> trump.txt
```

The column headers and the first 5 rows of 'trump.txt'



```
post_id,likes_count
_22228735667216_10153960167952217 22,368179
5550296508_10154298504746509,248012
18468761129_10153524839811130,229187
15704546335_10154108339971336,222119
15704546335_10154646163096336,218748
```

The last 5 rows of 'trump.txt'



```
5550296508_10154955006956509,1001
5550296508_10154541840616509,1001
18468761129_10153650393881130,1001
86680728811_10154802229533812,1000
5550296508_10154572513421509,1000
```

Question 10

In order to obtain the total number of love_count and angry_count for Donald Trump and the total for Barack Obama, we need to search for mentions of “Donald Trump” and “Barack Obama” separately, and total up the values in the love_count columns and the angry_count columns.

For this question, I will search for “Donald Trump” and “Barack Obama” throughout all the fields, this means that unlike previous questions, posts that do not mention, say, “Donald Trump” in the actual message but in the post_name, description, etc. will **not** be filtered out. This is done to get as much information as possible because there may be posts that only refer to them as “Trump” or “the President” in the actual message but **does** mention, “Donald Trump” in full within the name or description of the post. Such entries should not be overlooked.

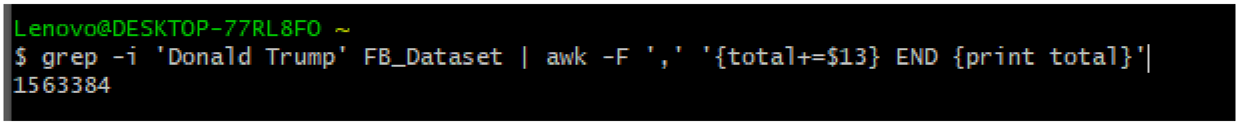
In addition, I will also be ignoring case when searching for “Donald Trump” and “Barack Obama” to account for the possibility of capitalization errors, regardless of whether their names were wrongly capitalized, readers would still understand who the post is talking about and express their feelings just as well. Thus, these entries are as valid as those with proper capitalization.

We will begin by obtaining the total number of love_count values for Donald Trump by using the grep command on FB_Dataset which searches all the rows and columns of the file for the target string. Here, `-i 'Donald Trump'` specifies that the command should be looking for matches with the string “Donald Trump” while ignoring upper/lower case. The result are those entries that mention Donald Trump somewhere in the post.

This output is then passed to the awk command, through the pipe (|) character, to total up the values in the love_count column. -F ‘,’ tells it that the input is being delimited by commas (,), `{total+=$13}` creates a variable total which aggregates the values in the 13th column (love_count column) as it traverses through each row. `END {print total}` gives it a condition so that only when it reaches the end of the file, print the aggregated total. This outputs the sum of values in the love_count column.

```
	grep -i 'Donald Trump' FB_Dataset | awk -F ',' '{total+=$13} END {print total}'
```

This outputs



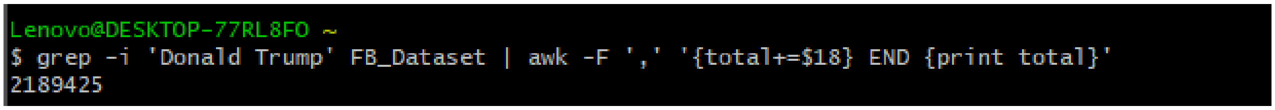
```
Lenovo@DESKTOP-77RL8FO ~
$ grep -i 'Donald Trump' FB_Dataset | awk -F ',' '{total+=$13} END {print total}'
1563384
```

We will move on to obtaining the total number of angry_count values for Donald Trump in almost the exact same way. We start by using the grep command on FB_Dataset which searches all the rows and columns of the file for the target string. Here, `-i 'Donald Trump'` specifies that the command should be looking for matches with the string “Donald Trump” while ignoring upper/lower case. The result are those entries that mention Donald Trump somewhere in the post.

This output is then passed to the awk command, through the pipe (|) character, to total up the values in the love_count column. -F ',' tells it that the input is being delimited by commas (,). Here is where it differs from the previous code, `{total+=$18}` creates a variable total which aggregates the values in the 18th column (angry_count column) as it traverses through each row. `END {print total}` gives it a condition so that only when it reaches the end of the file, print the aggregated total. This outputs the sum of values in the angry_count column.

```
    grep -i 'Donald Trump' FB_Dataset | awk -F ',' '{total+=$18} END {print
total}'
```

This outputs



```
Lenovo@DESKTOP-77RL8FO ~
$ grep -i 'Donald Trump' FB_Dataset | awk -F ',' '{total+=$18} END {print total}'
2189425
```

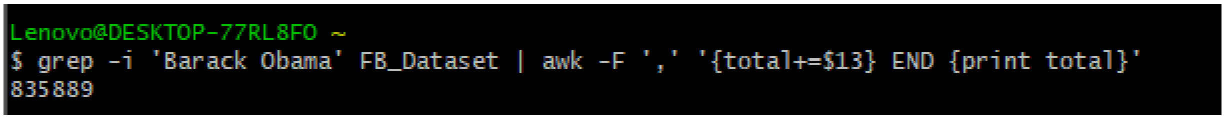
Now that we have calculated both sums for Trump, we can move on to Obama.

We will begin by obtaining the total number of love_count values for Barack Obama by using the grep command on FB_Dataset which searches all the rows and columns of the file for the target string. Here, `-i 'Barack Obama'` specifies that the command should be looking for matches with the string "Barack Obama" while ignoring upper/lower case. The result are those entries that mention Barack Obama somewhere in the post.

This output is then passed to the awk command, through the pipe (|) character, to total up the values in the love_count column. -F ',' tells it that the input is being delimited by commas (,), `{total+=$13}` creates a variable total which aggregates the values in the 13th column (love_count column) as it traverses through each row. `END {print total}` gives it a condition so that only when it reaches the end of the file, print the aggregated total. This outputs the sum of values in the love_count column.

```
    grep -i 'Barack Obama' FB_Dataset | awk -F ',' '{total+=$13} END {print
total}'
```

this outputs



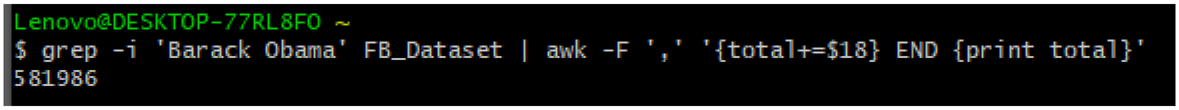
```
Lenovo@DESKTOP-77RL8FO ~
$ grep -i 'Barack Obama' FB_Dataset | awk -F ',' '{total+=$13} END {print total}'
835889
```

We will move on to obtaining the total number of angry_count values for Barack Obama in almost the exact same way. We start by using the grep command on FB_Dataset which searches all the rows and columns of the file for the target string. Here, `-i 'Barack Obama'` specifies that the command should be looking for matches with the string “Barack Obama” while ignoring upper/lower case. The result are those entries that mention Barack Obama somewhere in the post.

This output is then passed to the awk command, through the pipe (|) character, to total up the values in the love_count column. -F ‘,’ tells it that the input is being delimited by commas (,). Here is where it differs from the previous code, `{total+=$18}` creates a variable total which aggregates the values in the 18th column (angry_count column) as it traverses through each row. `END {print total}` gives it a condition so that only when it reaches the end of the file, print the aggregated total. This outputs the sum of values in the angry_count column.

```
    grep -i 'Barack Obama' FB_Dataset | awk -F ',' '{total+=$18} END {print
total}'
```

this outputs



```
Lenovo@DESKTOP-77RL8FO ~
$ grep -i 'Barack Obama' FB_Dataset | awk -F ',' '{total+=$18} END {print total}'
581986
```

In summary, we have found the following:

Donald Trump		Barack Obama	
love_count	angry_count	love_count	angry_count
1,563,384	2,189,425	835,889	581,986

Here I will interpret “positive feeling among people” with a bit of nuance. The people who are giving these reactions (love or angry) on Facebook are more likely to be ardent supporters or haters of the politician, as these reactions are quite extreme. Someone who vaguely supports Obama will be more likely to react with a like or not react at all rather than a love reaction. Therefore, if I interpret positive feeling as those who completely support Barack Obama or Donald Trump **while also including the much larger body of people who vaguely support one side or the other**, I think that Barack Obama would overall rouse more positive feeling for the general public compared to Donald Trump.

This is because Donald Trump is a much more divisive figure compared to Barack Obama, which is reflected in the data found as his total love_count and total angry_count both greatly exceeds that of Barack Obama’s. This means that many more people either love Trump or hate him compared to Obama where there are much more people who simply have a close-to-neutral opinion towards him, i.e. neither love or hate him. Notice that Trump’s angry_count is 1.4 times higher than his love_count, thus it follows that the average person is more likely to have a slightly negative than a positive feeling towards him. To further support this point, there is a huge group of American citizens who do not follow politics at all (Gao, 2014). These people are much more likely to form an impression of Trump or Obama based on what they have heard, typically from the media. These people are more likely to have a bad impression of Trump as he often gets himself into scandals which are heavily publicized and talked about.

On the other hand, it is completely the opposite for Barack Obama, his love_count is 1.4 times higher than his angry_count, and thus it is more probable that the average citizen have a slightly positive opinion towards him. This is supported by the fact that Obama rarely got himself into scandals, and was praised as a good orator (Henderson, 2017). In the data itself, we have also seen that posts mentioning Barack Obama get twice as many likes as those that mention Donald Trump.

Since there are much more people in the grey area between loving hating him, and it is more likely for the average person to have a positive feeling towards him, I think that while there may be more ardent supporters of Trump who completely love him, there is a much larger group of people who feel positive about Obama.

Part B: Graphing the Data in R

Question 1

To graph changes in the amount of discussion regarding Barack Obama over time, we will first need to use the BASH shell in order to extract the timestamps of posts referring to "Barack Obama". For this task, I will be searching for "Barack Obama" in all of the fields so that we do not miss any posts that only mention his name in the title or description.

We don't need to write the column names from FB_Dataset into the csv file as it will be changed later on and it is much easier to add/edit headings with R.

We can use the grep command on FB_Dataset to look for the target string "Barack Obama", here it will disregard upper/lower case when matching words. The output are all of the posts in the file that mention Obama.

This output is passed onto the awk command, through the pipe (|) character, in order to extract the timestamps from these posts. Here -F ',' specifies that the input uses commas (,) as delimiters, and `'{print $21}'` which prints the values in the 21st column (posted_at column) which contains the date and time information of when the post was published.

The output, i.e. the timestamps are then written to a csv file 'obama_mentions_timestamps.csv' with the > character.

```
    grep -i "Barack Obama" FB_Dataset | awk -F',' '{print $21}' >
obama_mentions_timestamps.csv
```

Now that we have extracted the timestamps of posts that mention "Barack Obama" into a csv file, we can move to R in order to begin graphing the histogram.

Firstly, in R, we should set the working directory to be the same as that of our terminal. We can do this by locating the path of the working directory, for me this is C:\cygwin64\home\Lenovo. R expects two forward slashes instead of one, so the path becomes C:\\cygwin64\\home\\Lenovo.

This is passed as the parameter of the setwd() function which changes the current working directory to the one provided in the argument of the function.

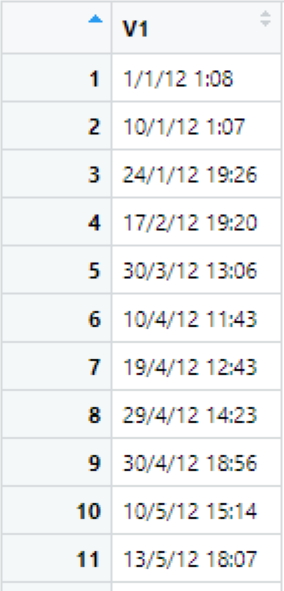
```
setwd("C:\\cygwin64\\home\\Lenovo")
```

Now we should read the csv file that we have just created into a data frame, which is what R mainly uses to work with and manipulate data. This can be done using the read.csv() method and passing in the filename to read from, “obama_mentions_timestamps.csv”, as its first argument.

Here, I have set header to be FALSE as we did not write the column header into our csv file and we want to prevent the function from using the first row as the header.

```
obama_mentions_timestamps <- read.csv("obama_mentions_timestamps.csv", header = FALSE)
```

table:

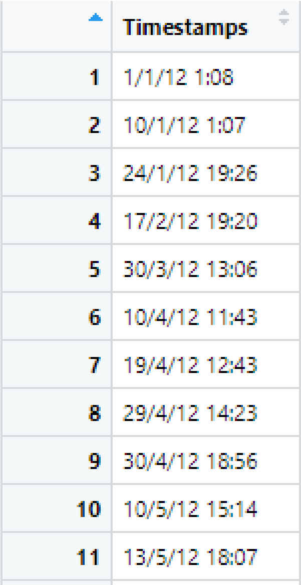


	V1
1	1/1/12 1:08
2	10/1/12 1:07
3	24/1/12 19:26
4	17/2/12 19:20
5	30/3/12 13:06
6	10/4/12 11:43
7	19/4/12 12:43
8	29/4/12 14:23
9	30/4/12 18:56
10	10/5/12 15:14
11	13/5/12 18:07

We can now give an appropriate column header, “Timestamps” for the data frame

```
colnames(obama_mentions_timestamps) <- "Timestamps"
```

table:

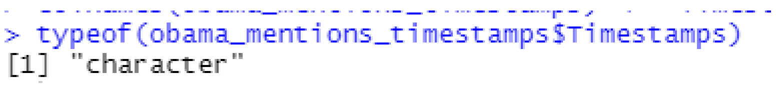


	Timestamps
1	1/1/12 1:08
2	10/1/12 1:07
3	24/1/12 19:26
4	17/2/12 19:20
5	30/3/12 13:06
6	10/4/12 11:43
7	19/4/12 12:43
8	29/4/12 14:23
9	30/4/12 18:56
10	10/5/12 15:14
11	13/5/12 18:07

In order to properly plot the histogram, we have to make sure that the values in the Timestamps column can actually be recognized as timestamps by R. To check the type of these values we can use the typeof() function

```
typeof(obama_mentions_timestamps$Timestamps)
```

output:



```
> typeof(obama_mentions_timestamps$Timestamps)
[1] "character"
```

They are characters! This means that they cannot be recognized as timestamps by R, we will have to first convert them using the strptime() function which parses the given data and converts it to objects of classes that represent calendar dates and times.

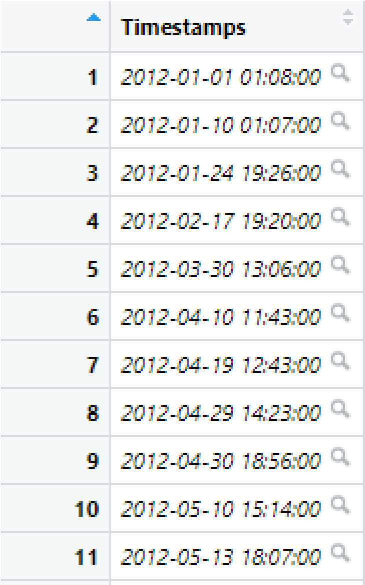
We can convert these values by first selecting them using the [table]$[column] notation and replacing each element with the converted result.

In order for strptime() to be able to convert these values, we have to provide their format.

Looking at the list of dates in our table, we can see that the format of the dates are day/month/year hour:minute, this can be represented by the string “%d/%m/%y %H:%M”, where %d is the day as a decimal number, %m is the month as a decimal number, %y represents the year without the century part, i.e. represents the year with 2 numbers such as 16, which refers to 2016, and 12 which likely refers to 2012. %H is the hours represented as a decimal number and %M is the minutes represented as a decimal number.

```
        obama_mentions_timestamps$Timestamps <- strptime(obama_mentions_timestamps$Timestamps,
format="%d/%m/%y %H:%M")
```

table:



	Timestamps
1	2012-01-01 01:08:00
2	2012-01-10 01:07:00
3	2012-01-24 19:26:00
4	2012-02-17 19:20:00
5	2012-03-30 13:06:00
6	2012-04-10 11:43:00
7	2012-04-19 12:43:00
8	2012-04-29 14:23:00
9	2012-04-30 18:56:00
10	2012-05-10 15:14:00
11	2012-05-13 18:07:00

Subtask i

Now that we have successfully converted the contents of the Timestamps column to something that R can recognize as timestamps, we can plot the histogram.

-The first argument passed into is the vector of values with which the histogram is plotted, in this case this is the Timestamps column of the obama_mentions_timestamps data frame

- The string passed to main is what is set as the title of the histogram

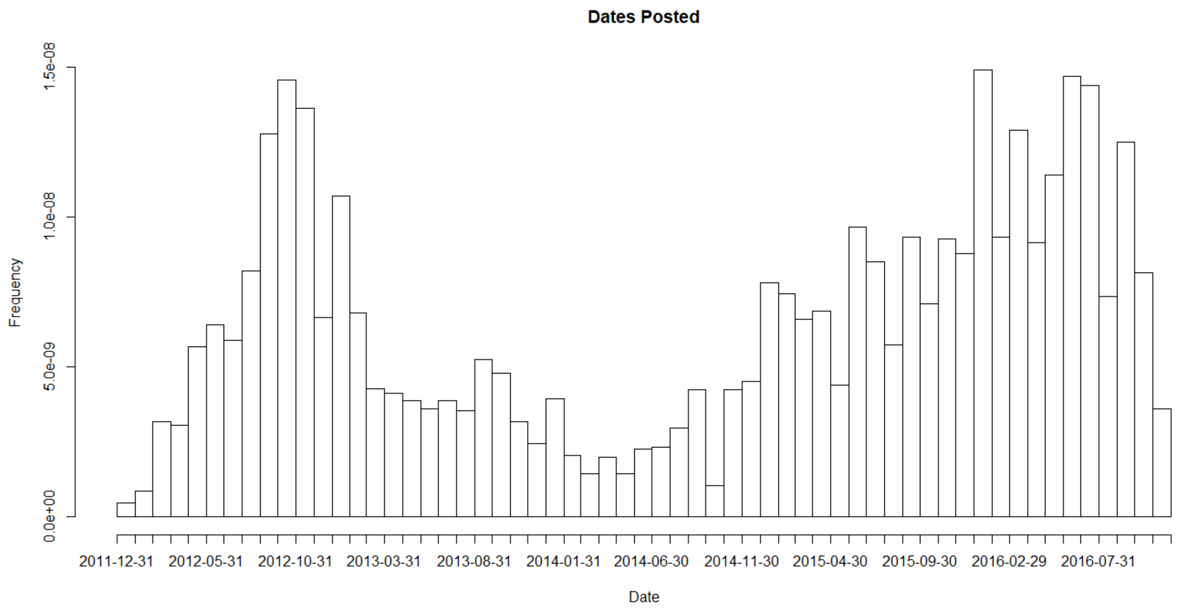
- the x axis is labeled with the text “Date”

- the y axis is labeled with the text “Frequency”

- the breaks argument determines the breakpoints between histogram cells, in this case “months” breaks up the data into months and plots the frequency for each month

```
    hist(obama_mentions_timestamps$Timestamps, main="Dates Posted", xlab="Date", ylab="Frequency", breaks="months")
```

this code outputs



Subtask ii

The shape of the histogram is quite unusual as it fluctuates up then down then up again. There are two peaks in the graph, one at the end of 2012 and another one during 2016. The graph starts off with very low frequency at the beginning of 2012, then it slowly rises until it reaches its first peak near the end of 2012. Afterwards, the graph starts decreasing quite rapidly and around 2013 to 2014, the frequency of posts mentioning Obama is quite steady and low. From this point, the graph gradually increases until it reaches its second peak which occurs in 2016.

The reason why there are so many posts mentioning Obama during 2012 and 2016 and not so much during other times is due to the presidential elections held during those years. The exact date of the 2012 elections is November 6th 2012 (Munro, 2012), which is exactly when the first peak of the graph occurred. At that time, Barack Obama was running to be re-elected as President of the United States so there would have been a lot of talks and discussion surrounding him and about him, as well as unending media attention. This translated to a high frequency of posts mentioning his name during that time.

It makes sense that the frequency of posts abated after the election concluded and people settles down from the election frenzy which meant that the media attention waned off and so did interest.

It is very intriguing that the graph peaks once more during 2016, sure it was an election year but Obama was not even in the running this time around, so why was he mentioned so much during this year? The reason for this is because of Donald Trump, who **was** in the running to become the President of the United States in 2016 (Beckwith, 2016). At the time, Trump was incredibly divisive and controversial, which we have concluded from another question above, his provocateur nature resulted in many Democrats disapproving of him and they were constantly comparing his somewhat erratic, vulgar behavior with the previous President, Obama's usual calm and cogent behavior. In addition, Obama has openly spoken about his distaste of Trump, calling him 'Un-American' which further spurred the media and public attention (Cassidy, 2016).

Unsurprisingly, as the 2016 elections went by the number of posts mentioning Barack Obama similarly decreased as America settled down with their new President and interest began to wore off.

Question 2

We need to extract the relevant columns (page_name, post_type, comments_count columns), then search for and filter for posts created by “abc-news” and store the post’s type as well as the number of comments the post received to a text file, ‘abc_news_post_details.txt’

First, we can use the cut command on FB_Dataset to isolate the columns that we require, -d ',' tells it that the input is being delimited by commas (,), and -f 1,8,11 makes sure the command only outputs the entry from the 1st, 8th, and 11th field (which are the page_name, post_type, comments_count columns).

The output is then passed to the grep command, through the pipe (|) character, to filter posts created by “abc-news”, the -i tells it to ignore upper/lower case when matching words.

The filtered result is then passed to another cut command, through the pipe (|) character to remove the page_name column. The -d ',' tells it that the input is being delimited by commas (,), and -f 2,3 makes sure the command only outputs the entry from the 2nd, and 3rd fields (which are the post_type, and comments_count columns).

This data is then written to the file ‘abc_news_post_details.txt’ using the > character.

```
      cut -d ',' -f 1,8,11 FB_Dataset | grep -i "abc-news" | cut -d ',' -f 2,3 >
abc_news_post_details.csv
```

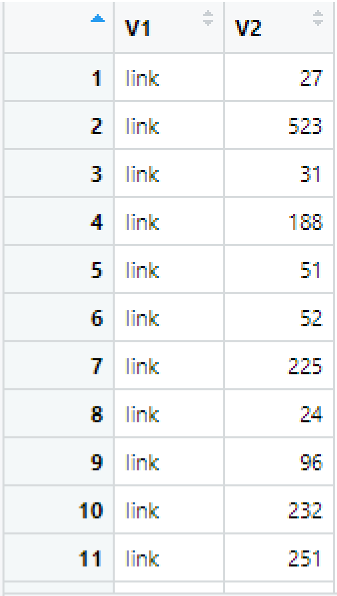
Now that we have created the csv file, we can move to R to graph our boxplot.

To begin, we must read the csv file into a data frame using the read.csv() function, passing in the filename to read from as the first argument. Data frames typically do not allow for repetition in the row names, and just using read.csv() will result in an error due to this characteristic. Therefore, the row.names = NULL argument generates numbers to be used as row names instead of using the post_type column, which allows us to read the csv file without issues.

Here, I have set header to be FALSE as we did not write the column header into our csv file and we want to prevent the function from using the first row as the header.

```
      abc_news_post_details = read.csv("abc_news_post_details.csv", row.names = NULL, header=FALSE)
```

table:

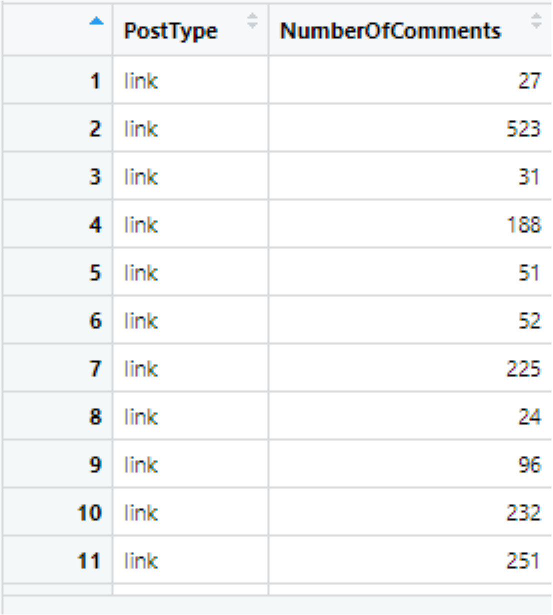


	V1	V2
1	link	27
2	link	523
3	link	31
4	link	188
5	link	51
6	link	52
7	link	225
8	link	24
9	link	96
10	link	232
11	link	251

We can now rename the column headers of the data frame with more appropriate names – PostType and NumberOfComments.

```
colnames(abc_news_post_details) <- c("PostType", "NumberOfComments")
```

table:



	PostType	NumberOfComments
1	link	27
2	link	523
3	link	31
4	link	188
5	link	51
6	link	52
7	link	225
8	link	24
9	link	96
10	link	232
11	link	251

Now we can finally plot our boxplot using the boxplot() function,

- the first argument is the formula NumberOfComments~PostType where a separate boxplot is created for every type of post, based on the number of comments received by posts of that type.

- the data argument specifies the data frame which is providing the data. In this case, it is abc_news_post_details.

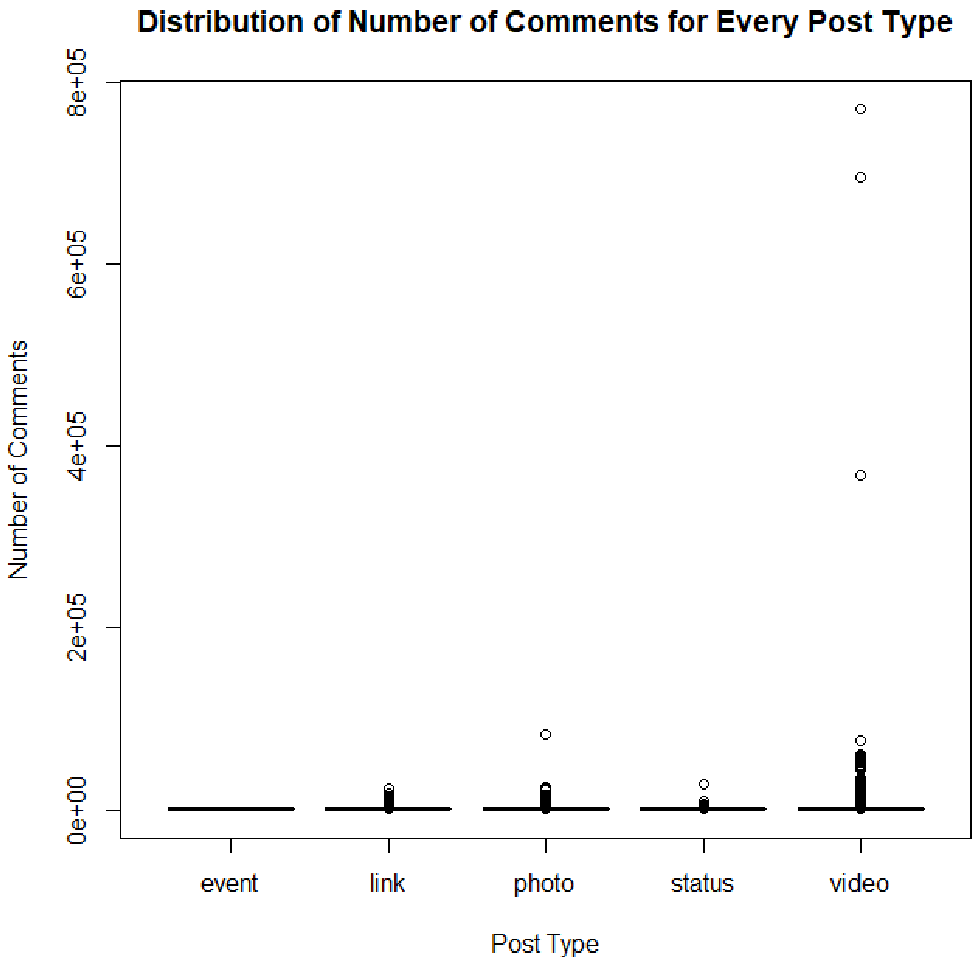
- the main argument sets the title of the boxplot as Distribution of Number of Comments for Every Post Type

- xlab argument sets the label of the x-axis as "Post Type"

- ylab argument sets the label of the y-axis as "Number of Comments"

```
        boxplot(NumberOfComments~PostType, data=abc_news_post_details, main="Distribution of Number of Comments for Every Post Type", xlab="Post Type", ylab="Number of Comments")
```

this code outputs



From this plot we can see that there are quite a lot of outliers for most of the post types (link, photo, status, video), except for event, for which there is not a single outlier visible. We can observe that some of the outliers of the video posts result in a massive number of comments, and when compared to other post types, they have three outliers that have much more comments than even the photo, status, and link posts with the highest number of comments. From this we can infer that videos have a much larger potential to go viral, i.e. receive an abnormally high number of comments compared to other post types. After video, photo posts have the second largest chance to go viral. These make sense because video and photo tend to be more exciting and engaging, for example a video has a variety of audiovisual elements which has a larger chance to capture the attention of the viewer, compared to a status post which is just static.

Observing this plot leads me to believe that video posts are the most engaging post types, not only because some of their outliers get an absurdly high number of comments but also if you look closely at the cluster of outliers at the bottom of every boxplot, notice that the cluster of outliers for video is much more spread in the upward direction (higher frequency), meaning that on average the outliers of the video posts receive much more comments compared to those of other post types. This indicates that the video posts are the most engaging post types.

Subtask ii

In the previous boxplot, we can see that the outliers heavily affected the scale of plot, so much so that we could only see the outliers for every post type. To improve the readability of the boxplots, we can filter out some of these outliers, more specifically posts that have more than 1000 comments.

We can do this by applying the condition <= 1000 on every element of the NumberOfComments column of the abc_news_post_details data frame. This results in a data frame of boolean elements, where it is True, that means that the corresponding element in the abc_news_post_details data frame is <= 1000, and otherwise it is > 1000. This boolean data frame is used to select the rows of the data frame to remove entries with > 1000 comments. This resulting data frame is assigned to filtered_posts.

```
filtered_posts <- abc_news_post_details[abc_news_post_details$NumberOfComments <= 1000,]
```

Now we can try and plot our boxplot again but this time with the filtered_posts data frame using the boxplot() function,

- the first argument is the formula NumberOfComments~PostType where a separate boxplot is created for every value in PostType.

- the data argument specifies the data frame which is providing the data. In this case, it is filtered_posts.

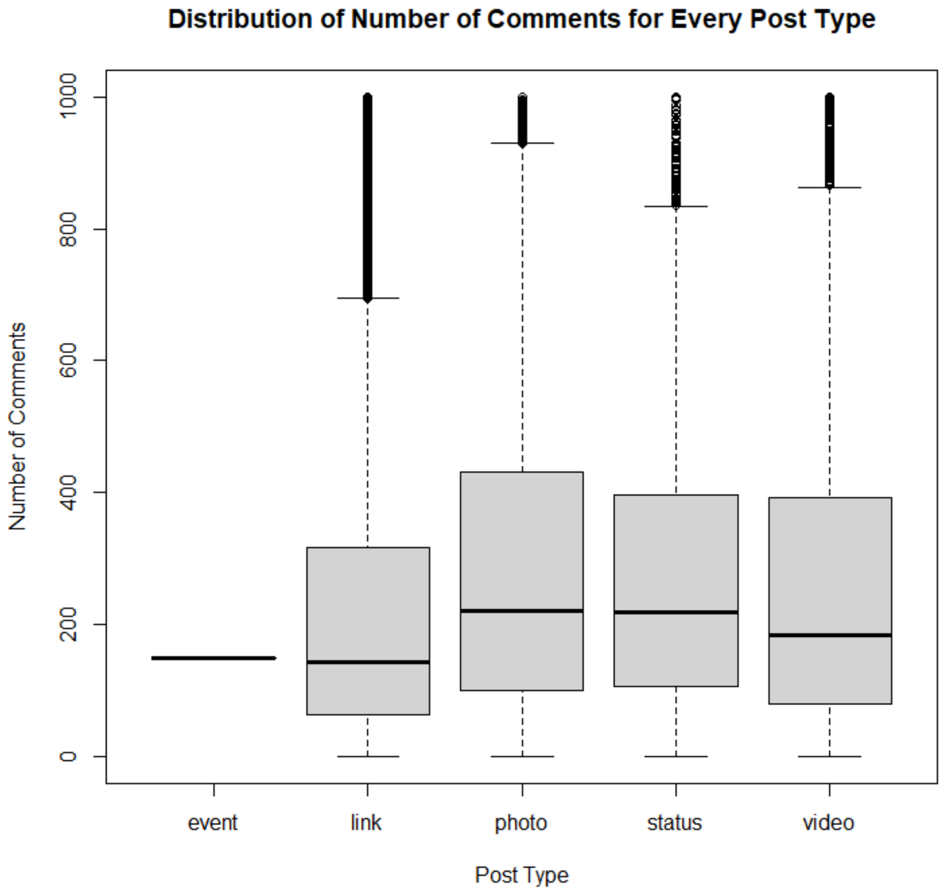
- the main argument sets the title of the boxplot as Distribution of Number of Comments for Every Post Type

- xlab argument sets the label of the x-axis as “Post Type”

- ylab argument sets the label of the y-axis as “Number of Comments”

```
boxplot(NumberOfComments~PostType, data=filtered_posts, main="Distribution of Number of Comments for Every Post Type", xlab="Post Type", ylab="Number of Comments")
```

this code outputs



With this we can finally see the more details of the boxplots, for example the median, 1st, 3rd quartiles, as well as the lower and upper bound for the different post types. This graph is much more readable than before!

Subtask iii

On average, it seems that photo and status posts have been most effective in terms of engagement We can see that the median line (the black line in the middle of the boxplot) for photo and status posts are almost at the same level, thus it is difficult to see which one of them has the higher median. To resolve this issue, we can label the actual value of the median on each of the boxplots.

In order to do this, we must first compute the median number of comments for each post type by using the aggregate() function and specifying the summary statistic that we want, in this case it is median.

```
medians <- aggregate(NumberOfComments ~  PostType, filtered_posts, median)
```

Now we can once more plot the boxplots, using boxplot(), where:

- the first argument is the formula NumberOfComments~PostType where a separate boxplot is created for every value in PostType.

- the data argument specifies the data frame which is providing the data. In this case, it is filtered_posts.

- the main argument sets the title of the boxplot as Distribution of Number of Comments for Every Post Type

- xlab argument sets the label of the x-axis as “Post Type”

- ylab argument sets the label of the y-axis as “Number of Comments”

```
boxplot(NumberOfComments ~ PostType, filtered_posts, main="Distribution of Number of Comments for Every Post Type", xlab="Post Type", ylab="Number of Comments")
```

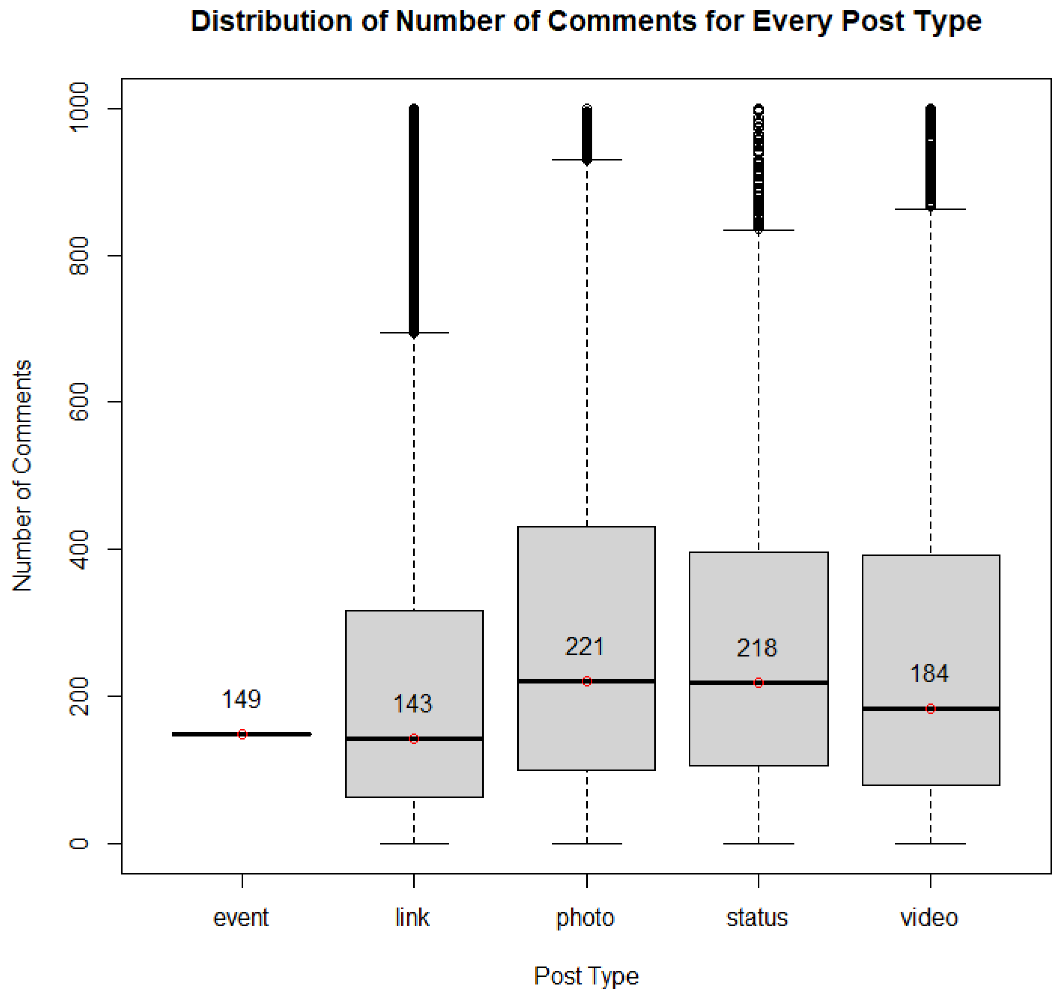
provides data labels for the median of each of the 5 post types and place it with 50px offset from the median line so that it is more readable.

```
text(1:5, medians$NumberOfComments + 50, labels = medians$NumberOfComments)
```

Plots 5 red points on the center of the median line so viewers are able to get a better intuitive understanding of what the text is meant to be labelling.

```
points(1:5, medians$NumberOfComments, col = "red")
```

the output



Therefore, from the boxplot we can see that for abc-news, photo posts have the highest median number of comments, 221 among the post types.

Conclusion

In this report, I have read, analyzed, and manipulated the provided dataset of Facebook posts using shell commands, and I obtained valuable insights from graphing this data in R, especially when comparing the differences in posts and how people react to posts that mention polarizing political leaders such as Donald Trump and Barack Obama.

I gained experience working with BASH shell to investigate large files, the importance, and the convenience of pipelining, and I have learned a large variety of commands in order to perform various operations such as filtering, searching, sorting, aggregating the data and writing the results into a file, which are integral as Data Scientists. Additionally, in the second part of the report, I have practiced using R in order to create graphs such as histograms and boxplots.

References

Beckwith, D. (2016). United States Presidential Election of 2016. Retrieved from https://www.britannica.com/topic/United-States-presidential-election-of-2016

Cassidy, J. (2016). Obama’s Powerful Message: Donald Trump Is Un-American. Retrieved from https://www.newyorker.com/news/john-cassidy/obamas-powerful-message-donald-trump-is-un-american

Gao, G. (2014). 1-in-10 Americans don’t give a hoot about politics. Retrieved from https://www.pewresearch.org/fact-tank/2014/07/07/1-in-10-americans-dont-give-a-hoot-about-politics/

Henderson, C. (2017). Where does Obama rank in American oratory?. Retrieved from https://www.washingtonpost.com/opinions/where-does-obama-rank-in-american-oratory/2017/01/13/e29d3128-d374-11e6-945a-76f69a399dd5_story.html

Munro, A. (2012). United States Presidential Election of 2012. Retrieved from https://www.britannica.com/event/United-States-Presidential-Election-of-2012