FIT1043 Introduction to Data Science
Assignment 3

Loh Jing Wei
30856183

Section A	Answers
A1	Original file size is 110MB. Decompressed file size is 343MB. **Code** `ls -lh FB_Dataset.gz` *ls* command with *'-lh'* option display the file size in readable form. `gunzip FB_Dataset.gz` *gunzip* command is used to unzip/decompress the file. `ls -lh FB_Dataset` **Output** (base) Shers-MacBook-Pro:Downloads ljw$ ls -lh FB_Dataset.gz -rw-r--r--@ 1 ljw staff 110M May 21 13:47 FB_Dataset.gz (base) Shers-MacBook-Pro:Downloads ljw$ gunzip FB_Dataset.gz (base) Shers-MacBook-Pro:Downloads ljw$ ls -lh FB_Dataset -rw-r--r-- 1 ljw staff 343M May 21 13:47 FB_Dataset
A2	The delimiter used is "," (comma) and there are 533940 rows. **Code** `head -1 FB_Dataset | less` This line of code displays the header row (with the column names) *head* command with *'-n'* option display first n number of lines (By default, head command display first 10 lines). In this case, we would like to see the header, which is the first line, hence n used is 1. *less* command shows file content without loading the whole file. `wc -l FB_Dataset` This line of code finds the total number of rows. *wc* command with *'-l'* option returns the number of lines, where each line represents a row. **Output** 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 (END) (base) Shers-MacBook-Pro:Downloads ljw$ wc -l FB_Dataset 533940 FB_Dataset

A3	There are 21 columns in total. **Code** `head -1 FB_Dataset | tr ',' '\n'` The *Pipe* operator '|' redirect output of one program as input of another program. *Head '-1'* command is used to extract the first line of the file which consist of the column name. The output (first line) is pipe to tr command. The *tr* command is used to replace characters. In this case we are replacing the delimiter ',' with empty lines. As a result, each line from the output represents one column name. (This is so that we can read the column names more easily) `head -1 FB_Dataset | tr ',' '\n' | wc -l` The *wc* command returns the number of lines, word count and character counts. Since we are using it with option *'-l'*, the command will output total number of lines. This output will represent the number of columns (since each line represent 1 column name). **Output** `(base) Shers-MacBook-Pro:Downloads ljw$ 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` `(base) Shers-MacBook-Pro:Downloads ljw$ head -1 FB_Dataset | tr ',' '\n' | wc -l` `21`
A4	Assuming that "unique pages" refers to number of entries, this dataset has 533939 entries. **Code** `wc -l FB_Dataset` The *wc* command with *'-l'* option returns the total number of lines. Each line represents an entry except the first line (first line consist of column names), hence we will subtract one from the output (533940 – 1 = 533939). **Output** `(base) Shers-MacBook-Pro:Downloads ljw$ wc -l FB_Dataset` `533940 FB_Dataset`

A5	The date ranges from 1/1/12 to 7/11/16. **Code** `cat FB_Dataset | awk -F ',' '{print $21}' | head` `cat FB_Dataset | awk -F ',' '{print $21}' \| tail` The *Pipe* operator '\|' redirect output of one program as input of another program. The *cat* command prints the FB_Dataset file. The output is redirected to awk command. The *awk* command process the file one line at a time. *'-F'* option is used to input field separator (delimiter), here it specifies that the delimiter used is comma. The *print* command output the entries in 21st column, which is the "posted_at" column consisting the dates. The output is then redirected to head / tail command. The *head* and *tail* command will output the first 10 row and last 10 row, which can be used to find the date range by taking the first and last row. **Output** First 10 rows `(base) Shers-MacBook-Pro:Downloads ljw$ cat FB_Dataset | awk -F ',' '{print $21}' \| head` `posted_at` `1/1/12 0:30` `1/1/12 1:08` `1/1/12 2:00` `1/1/12 2:35` `1/1/12 3:36` `1/1/12 4:13` `1/1/12 6:00` `1/1/12 14:00` `1/1/12 17:00` Last 10 rows `(base) Shers-MacBook-Pro:Downloads ljw$ cat FB_Dataset | awk -F ',' '{print $21}' | tail` `7/11/16 20:01` `7/11/16 20:10` `7/11/16 20:30` `7/11/16 21:00` `7/11/16 21:30` `7/11/16 22:00` `7/11/16 22:30` `7/11/16 23:00` `7/11/16 23:30` `7/11/16 23:45`

A6	The first mention of “Malaysia Airline” is on 8/4/14. The media source is ABC News. **Code** `cat FB_Dataset | awk -F ',' '{print $1,$5,$21}' \| grep -w "Malaysia Airlines" \| head -1` The *Pipe* operator ‘\|’ redirect output of one program as input of another program. The *cat* command prints the FB_Dataset file. The output is redirected to awk command. The *awk* command process the file one line at a time. *‘-F’* option is used to input field separator (delimiter), here it specifies that the delimiter used is comma. The print command output entries in the 1st, 5th and 21st column (“page_name”, “message” and “posted_at”), this output is then redirected to grep command. The *grep* command search for pattern (string of characters) in the file. The *‘-w’* option search for whole-word match of “Malaysia Airlines”, this can avoid searching “Malaysia Airlines” as a substring of another word. The output of the grep command is then redirected to head command. The *head* command with *‘-1’* option will output the “page_name”, “message” and “posted_at” which has the first mention of “Malaysian airline”. **Output** `(base) Shers-MacBook-Pro:Downloads ljw$ cat FB_Dataset | awk -F ',' '{print $1,$5,$21}' | grep -w "Malaysia Airlines" | head -1` `abc-news DEVELOPING: Malaysia Airlines spokesperson: Flight carrying 239 people from Kuala Lumpur to Beijing has gone missing  contact lost: http://abcn.ws/NHHeLT 8/3/14 0:47`
A7	There is 153 mentions of “Cat” in message column. **Code** `cat FB_Dataset | awk -F ',' '{print $5}' \| grep -w "Cat" \| wc -l` The *Pipe* operator ‘\|’ redirect output of one program as input of another program. The *cat* command prints the FB_Dataset file. The output is redirected to awk command. The *awk* command process the file one line at a time. *‘-F’* option is used to input field separator (delimiter), here we specify that the delimiter used is comma. The print command output the 5th column (“message”); this output is then redirected to grep command. The *grep* command with *‘-w’* option search for whole-word match of “Cat”. The output is rows which contain at least one mention of “Cat”. (I assume that message with multiple “Cat” text is considered as one mention). The output is redirected to wc command. The *wc* command with *‘-l’* option will output the total number of lines (rows), hence the output correspond to number of times “Cat” mentioned in message column. **Output** `(base) Shers-MacBook-Pro:Downloads ljw$ cat FB_Dataset | awk -F ',' '{print $5}' | grep -w "Cat" | wc -l` `153`

A8	There is 303 mentions of "Dog" in message column. If we assume that the popularity of Dog and Cat is correlated to the number of times "Dog and "Cat" is mentions in message, we can conclude that Dog is more popular than Cat because the number of times it is mention is higher than Cat (303 mentions of Dog vs 153 mentions of Cat) **Code** `cat FB_Dataset | awk -F ',' '{print $5}' \| grep -w "Dog" \| wc -l` The *Pipe* operator '\|' redirect output of one program as input of another program. The *cat* command prints the FB_Dataset file. The output is redirected to awk command. The *awk* command process the file one line at a time. *'-F'* option is used to input field separator (delimiter), here we specify that the delimiter used is comma. The print command output the 5th column ("message"); this output is then redirected to grep command. The *grep* command with *'-w'* option search for whole-word match of "Dog". The output is rows which contain at least one mention of "Dog". The output is redirected to wc command. The *wc* command with *'-l'* option will output the total number of lines (rows), hence the output correspond to number of times "Dog" mentioned in message column. **Output** `(base) Shers-MacBook-Pro:Downloads ljw$ cat FB_Dataset | awk -F ',' '{print $5}' | grep -w "Dog" | wc -l` `303`
A9	**Code** `cat FB_Dataset | awk -F ',' '{print $2,$10}' | head -1 > cat.txt` The *Pipe* operation '|' redirect output of one program as input of another program. The *awk* command process the file one line at a time. *'-F'* option is used to input field separator (delimiter), here we specify that the delimiter used is comma. It prints the 2nd and 10th column which correspond to the post_id and likes_count column. *head -1* is used to extract the first row for 2nd and 10th column, which is the header post_id and likes_count. It is then used to save data to cat.txt file. `cat FB_Dataset | awk -v FS=',' -v OFS=',' '$10>=1000 {print $2, $5, $10}' | grep -w -i "Cat" | awk -F ',' '{print $1, $3}' | sort -nk2 >> cat.txt` The *Pipe* operation '|' redirect output of one program as input of another program. The *awk* with *'-v'* option allows variable to be assign before operation starts, we assign both *FS* (field separator) and *OFS* (output field separator) as comma. This allows us to use comma as delimiter for next awk command (We could also use default delimiter which is empty spaces, however that could potentially remove the leading spaces in some post_id, so I would try to avoid this). The *$10>=1000* code select rows that have likes_count greater than or equal to 1000. The *print* command will output 2nd , 5th and 10th column (post_id, message and likes_count).

	The *grep* command with *'-w'* option search for whole-word match of "Cat", the *'-I'* option will ignore the case for "Cat" search (hence cat, cAt, cAT and etc will be match too). The second *awk* command with *'-F'* option specify the delimiter as comma. The *print* command selects the 1st and 3rd column which correspond to post_id and likes_count. The *sort* command with *'-n'* option will sort numerically. We also select the second column to be sorted (likes_count) using *'-k'* option. Lastly the *>>* command appends output into the existing cat.txt file **<u>Output</u>** First 5 rows with header `post_id likes_count` `131459315949_10152991226590950 1014` `10606591490_10153384001006491 1023` `18468761129_10152157468736130 1023` `5550296508_10155028327981509 1027` `_22228735667216_10153152501457217722 1042` Last 5 rows `5281959998_10150552562134999 76591` `_22228735667216_10152045698542217722 88868` `15704546335_10153169270411336 100029` `18468761129_10152164706781130 206313` `86680728811_10154249018303812 258713`

A10	Since “Cat” has higher love_count as well as lower angry_count than “Dog”, we can conclude that cat invoke more positive feelings among people. **Code** Total number of love_count for “Cats” `cat FB_Dataset | awk -v FS=',' -v OFS=',' '{print $5,$13}' | grep -w "Cat" | awk -F',' '{sum+=$2;}END{print sum;}'` Total number of angry_count for “Cat” `cat FB_Dataset | awk -v FS=',' -v OFS=',' '{print $5,$18}' | grep -w "Cat" | awk -F',' '{sum+=$2;}END{print sum;}'` Total number of love_count for “Dog” `cat FB_Dataset | awk -v FS=',' -v OFS=',' '{print $5,$13}' | grep -w "Dog" | awk -F',' '{sum+=$2;}END{print sum;}'` Total number of angry_count for “Dog” `cat FB_Dataset | awk -v FS=',' -v OFS=',' '{print $5,$18}' | grep -w "Dog" | awk -F',' '{sum+=$2;}END{print sum;}'` The *Pipe* operation ‘|’ redirect output of one program as input of another program. The *awk* with *‘-v’* option allows variable to be assign before operation starts, we assign both *FS* (field separator) and *OFS* (output field separator) as comma. This allows us to use comma as delimiter for next awk command. The *print* command will output 5th column (message) as well as 13 or 18th column (love_count or angry_count) The *grep* command with *‘-w’* option search for whole-word match of “Cat” or “Dog”. The second *awk* command with *‘-F’* option specify the delimiter as comma. • *{sum+=$2;}* will return the sum of all values in 2nd column (love_count or angry_count). • *END{print sum;}* will output the sum after every lines are read. Hence the output will be the sum of love_count or angry_count **Output** `(base) Shers-MacBook-Pro:Downloads ljw$ cat FB_Dataset | awk -v FS=',' -v OFS=',' '{print $5,$13}' | grep -w "Cat" | awk -F',' '{sum+=$2;}END{print sum;}'` `46944` `(base) Shers-MacBook-Pro:Downloads ljw$ cat FB_Dataset | awk -v FS=',' -v OFS=',' '{print $5,$18}' | grep -w "Cat" | awk -F',' '{sum+=$2;}END{print sum;}'` `126` `(base) Shers-MacBook-Pro:Downloads ljw$ cat FB_Dataset | awk -v FS=',' -v OFS=',' '{print $5,$13}' | grep -w "Dog" | awk -F',' '{sum+=$2;}END{print sum;}'` `32953` `(base) Shers-MacBook-Pro:Downloads ljw$ cat FB_Dataset | awk -v FS=',' -v OFS=',' '{print $5,$18}' | grep -w "Dog" | awk -F',' '{sum+=$2;}END{print sum;}'` `5996`

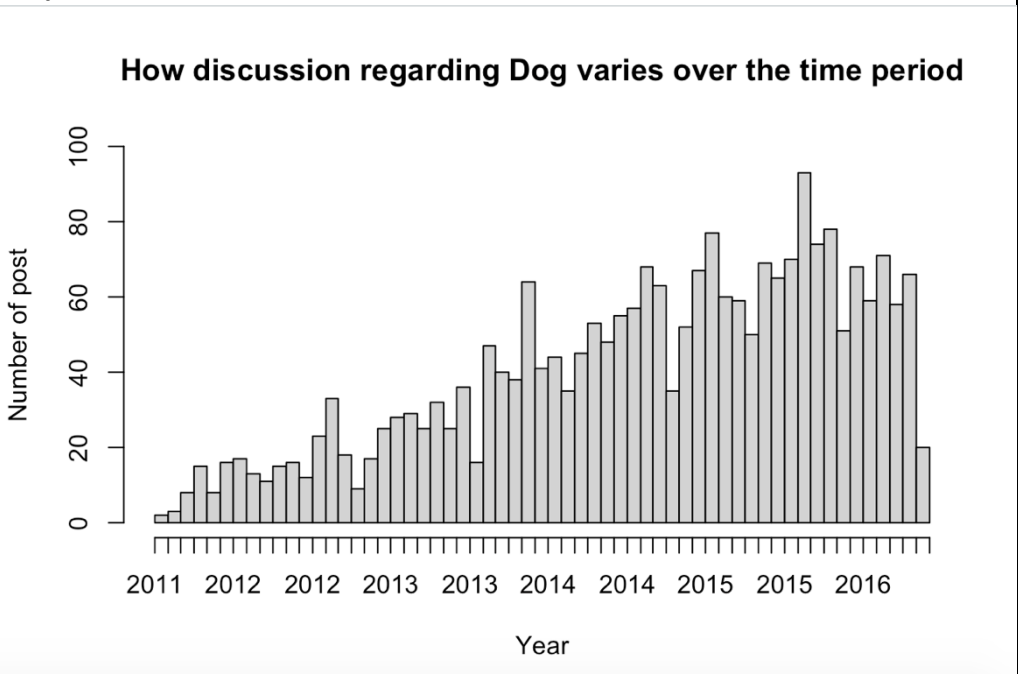
Section B	Answers
B1	**Bash codes** `cat FB_Dataset | awk -v FS=',' -v OFS=',' '{print $5,$21}' | grep -w -i "Dog" | awk -F',' '{print $2}' > RDog.txt` The *awk* with *'-v'* option allows variable to be assign before operation starts, we assign both *FS* (field separator) and *OFS* (output field separator) as comma. This allows us to use comma as delimiter for next awk command. The *print* command will output 5th (message) and 21st column (posted_at). The *grep* command with *'-w'* and '-i' option search for whole-word match of "Cat" or "Dog" while ignoring the case. The second *awk* command with *'-F'* option specify the delimiter as comma. And the print command selects the posted_at column, which provides the date and time we are interested in. We then save data to RDog.txt file using > operator. **R code** `setwd("/Users/ljw/Downloads")` `df <- read.table('RDog.txt', sep = "\n", header = FALSE)` `colnames(df) <- c("date_time")` `df$date_time <- strptime(df$date_time, "%d/%m/%y %H:%M")` First, we read the table from RDog.txt file, let header = FALSE which imply that there is no header from RDog.txt. We changed the column name to "date_time". Since the data from RDog.txt is string, we will need to convert it into date-time object. We can do this by using the strptime() function. The arguments we passed into the strptime() function is • df$date_time: the object to be converted (in this case the string). • "%d/%m/%y %H:%M": the format for the date and time. In our case, the input string provides the date as date/month/year and time as hour:minute.

```
hist(df$date_time, breaks="months", ylim = c(0,100), format =
"%Y", plot = TRUE, freq = TRUE, main = "How discussion
regarding Dog varies over the time period", xlab = "Year",
ylab = "Number of post")
```

After converting string into date and time (object of **POSIXlt** class), we can use the hist() function to create a histogram.
The histogram will show the date_time on x-axis and frequency of posts about dogs on y-axis. The arguments we passed into the hist() function is

- df$date_time
- breaks="months": Since histogram group data into bins of equal width, this parameter set the width to be a month long. As a result, y-axis will show the frequency of post in a month.
- ylim = c(0,100): Set the limit of y-axis from 0 to 100.
- format = "%Y": For the x-axis, so that it will shows only the year.
- plot = TRUE: So that a histogram is plotted.
- freq = TRUE: the histogram will represent the frequencies.

Output



B2

Bash code

```
cat FB_Dataset | awk -v FS=',' -v OFS=',' '{print $1,$8,$11}'
| grep -w "abc-news" | awk -F',' '{print $2,$3}' > Rengage.txt
```

This line of code will extracts post_type and comments_count for abc-news then save into Rengage.txt file.

R code

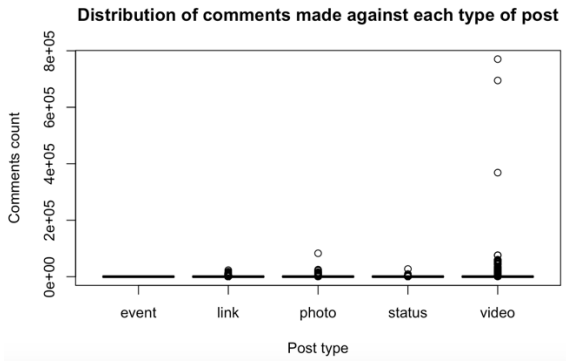
B2(i)

```
df2 <- read.table('REngage.txt', header = FALSE)

colnames(df2) <- c("post_type","comments_count")

boxplot(df2$comments_count~df2$post_type, xlab = "Post type",
ylab = "Comments count", main = "Distribution of comments made
against each type of post")
```

The boxplot() function will draw multiple boxplot side-by-side in same graph. Each boxplot represent a post_type. y-axis shows the statistic of comments_count of each post_type.

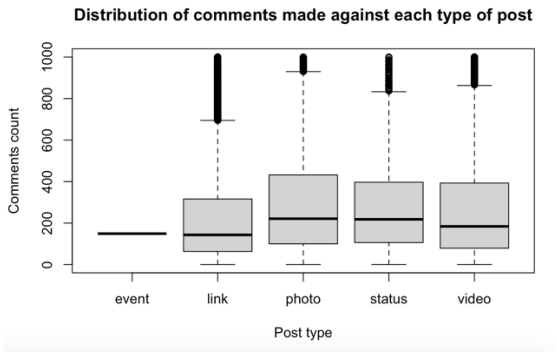


From the output graph, it is difficult to tell which is the most engaging post type since the outliers are too far apart from the boxplot itself.

B2(ii)

```
library(dplyr)
fltr <- filter(df2, comments_count <= 1000)
boxplot(fltr$comments_count~fltr$post_type, xlab = "Post
type", ylab = "Comments count", main = "Distribution of
comments made against each type of post")
```

By using the dplyr library's filter() function, we can filter out comments_count which are greater than 1000, thus removing the outliers affecting our boxplot readability.



Here, we can see that photos has the highest median for comments_count. Hence the most engaging post_type is probably photo.

B2(iii)

```
grp <- group_by(fltr, post_type)
summarise(grp, median = median(comments_count))
```

Using the group_by() function, we can group the data according to their post_type. Then, using summarise() function, we can find the median value of comments_count for each post_type.

event	149
link	143
photo	221
status	218
video	184

From the result, we can see that the post_type photo has the highest median value of comment_count. In fact, this is the same conclusion we drawn from B2(ii).