FIT1043 Introduction to Data Science

Assignment 3

FOO KAI YAN
33085625

13th October 2022

Introduction

My local machine is Windows OS. Therefore, all BASH line commands will be done using Cygwin64 Terminal. The 2 data sets used for the assignment is as mentioned as below:

- Twitter_Data_1.gz
- Dataset_TIST2015.tar

Processing Data with Cygwin64 Terminal

The following commands assist in navigation of the compressed folder obtained from Moodle:

```
$ pwd

$ cp /cygdrive/c/Users/Nicol\ Foo/Downloads/dataset_TIST2015.tar .

$ ls
```

```
Nicol Foo@KaiYanFoo ~
$ pwd
/home/Nicol Foo

Nicol Foo@KaiYanFoo ~
$ cp /cygdrive/c/Users/Nicol\ Foo/Downloads/dataset_TIST2015.tar .

Nicol Foo@KaiYanFoo ~
$ ls
dataset_TIST2015.tar
```

Task A

1. A1 - Decompression of file

   Originally, the compressed folder is inspected to have size of 2443296768 bytes. Inspection is done with the command as shown below:

   ```
   $ ls -l dataset_TIST2015.tar
   ```

   ```
   Nicol Foo@KaiYanFoo ~
   $ ls -l dataset_TIST2015.tar
   -rwx------ 1 Nicol Foo None 2443296768 Oct 14 11:01 dataset_TIST2015.tar
   ```

   ls used within the command means directory listing which means that the information of the file will be listed out when specified. ls -l together means to list the directories in a long form. With the use of ls -l, the exact file size will be shown in bytes.

   Decompression of the file is usually done with the following command:

   ```
   $ gunzip dataset_TIST2015.tar
   ```

   ```
   Nicol Foo@KaiYanFoo ~
   $ gunzip -d dataset_TIST2015.tar
   gzip: dataset_TIST2015.tar: unknown suffix -- ignored
   ```

But gunzip only works with files with .gz extension. Hence, the command above shows an error which means the command doesn't work on dataset_TIST2015.tar as dataset_TIST2015.tar does not have a .gz extension. The gz files will be either decompressed or compressed with using gunzip within the command. -d used within the command means to decompress the file.

This is the file size of dataset_TIST2015.tar when compressed:

```
$ ls -lh
```

```
Nicol Foo@KaiYanFoo ~
$ ls -lh
total 2.3G
-rwx------ 1 Nicol Foo None 2.3G Oct 14 11:01 dataset_TIST2015.tar
```

Because this is a tar file without any .gz extension, an alternative command will need to be used. When decompressed with the following command, the file size of dataset_TIST2015.tar will be shown as followed:

```
$ tar -xvf dataset_TIST2015.tar

$ ls -lh
```

```
Nicol Foo@KaiYanFoo ~
$ tar -xvf dataset_TIST2015.tar
dataset_TIST2015_Checkins_v2.txt
dataset_TIST2015_Cities.txt
dataset_TIST2015_POIs.txt
dataset_TIST2015_readme_v2.txt

Nicol Foo@KaiYanFoo ~
$ ls -lh
total 4.6G
-rwx------ 1 Nicol Foo None 2.3G Oct 16 14:23 dataset_TIST2015.tar
-rwxr-xr-x 1 Nicol Foo None 2.1G Oct  6 18:53 dataset_TIST2015_Checkins_v2.txt
-rwxr-xr-x 1 Nicol Foo None  25K Aug 12  2015 dataset_TIST2015_Cities.txt
-rwxr-xr-x 1 Nicol Foo None 222M Aug 12  2015 dataset_TIST2015_POIs.txt
-rwxr-xr-x 1 Nicol Foo None 2.0K Oct  6 18:59 dataset_TIST2015_readme_v2.txt
```

-lh used in the commands means to list the information of the file in a human readable form like Gigabyte (GB), Megabyte (MB) or Kilobyte (KB).

The size of dataset_TIST2015.tar when decompressed is 2.3GB and the individual file size of each file within dataset_TIST2015.tar are also shown above.

From the command above, we can see there is a total of 4 files in the tar file and each file size are as written below with the format (file name | file size):

- dataset_TIST2105_Checkins_v2.txt | 2.1GB
- dataset_TIST2105_Cities.txt | 25KB
- dataset_TIST2105_POIs.txt | 222MB
- dataset_TIST2105_readme_v2.txt | 2.0KB

2. A2 – Separation of columns in file named dataset_TIST2015_Checkins_v2.txt

First, the first 10 lines of dataset_TIST2015_Checkins_v2.txt will be outputted to get an understanding on how the data within the files looked like. The command to do this is show below:

```
$ head -10 dataset_TIST2015_Checkins_v2.txt
```

```
Nicol Foo@KaiYanFoo ~
$ head -10 dataset_TIST2015_Checkins_v2.txt
user_id venue_id        UTC_time        timezone_offset
50756   4f5e3a72e4b053fd6a4313f6        Tue Apr 03 18:00:06 +0000 2012  240
190571  4b4b87b5f964a5204a9f26e3        Tue Apr 03 18:00:07 +0000 2012  180
221021  4a85b1b3f964a520eefe1fe3        Tue Apr 03 18:00:08 +0000 2012  -240
66981   4b4606f2f964a520751426e3        Tue Apr 03 18:00:08 +0000 2012  -300
21010   4c2b4e8a9a559c74832f0de2        Tue Apr 03 18:00:09 +0000 2012  240
28761   4b4bade2f964a520cfa326e3        Tue Apr 03 18:00:09 +0000 2012  -240
39350   49bbd6c0f964a520f4531fe3        Tue Apr 03 18:00:09 +0000 2012  -240
1446    4e88cf4ed22d53877981fdab        Tue Apr 03 18:00:09 +0000 2012  -300
82296   4dfc825bc65b31579b2e7679        Tue Apr 03 18:00:11 +0000 2012  180
```

The following command shows the first line of dataset_TIST2015_Checkins_v2.txt which consist of the column headers:

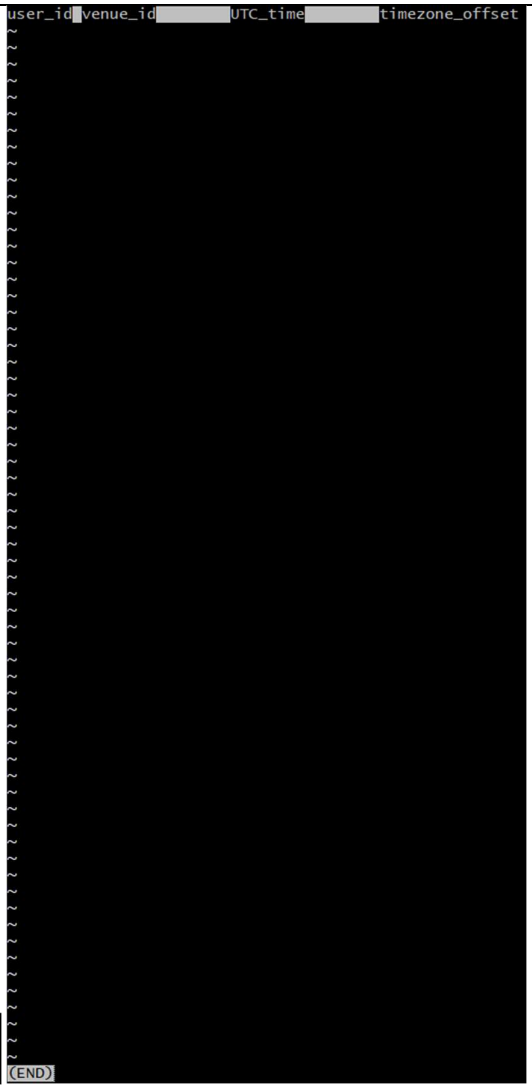
```
$ awk 'NR==1 {print; exit' dataset_TIST2015_Checkins_v2.txt
```

```
Nicol Foo@KaiYanFoo ~
$ awk 'NR==1 {print; exit}' dataset_TIST2015_Checkins_v2.txt
user_id venue_id        UTC_time        timezone_offset
```

We can see that the delimiter is obviously tab ‘ ’ but just to be sure, we will use the next command shown below to check if tab is really the actual delimiter.

```
$ head -1 dataset_TIST2015_Checkins.txt | less

/<tab>
```

```
Nicol Foo@KaiYanFoo ~
$ head -1 dataset_TIST2015_Checkins_v2.txt | less
```

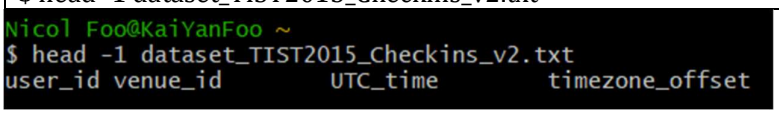


The delimiter used to separate each column is whitespace like tab ' '. This can be seen when the command /<tab> was entered after the command head -1 dataset_TIST2015_Checkins.txt | less where the delimiter, which is tab in this case, is highlighted in white. less is a command where it can be used to display text file contents one screen at a time when piped.

There is a total of 4 columns in the file named dataset_TIST2015_Checkins_v2.txt.

3. A3 – Column names for the file named dataset_TIST2015_Checkins_v2.txt
   To obtain the column named for file the named dataset_TIST2015_Checkins_v2.txt, the following commands were used:

```
$ head -1 dataset_TIST2015_Checkins_v2.txt
```



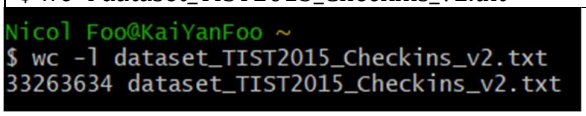
First line of a file that is said to contain columns will usually be the column headers where the names of each column is located at. Hence, to get the column names, only the first line of the file is required.

Column 1 name is given to be user_id. The names of each column are shown as below:

i. Column 2 = venue_id
ii. Column 3 = UTC_time
iii. Column 4 = timezone_offset

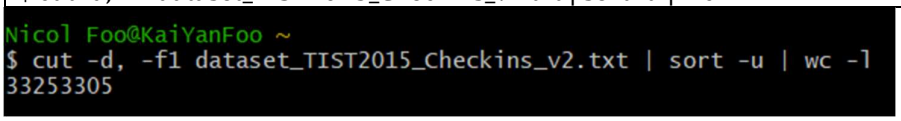
4. A4 – Number of Check-ins and users found in dataset_TIST2015_Checkins_v2.txt

```
$ wc -l dataset_TIST2015_Checkins_v2.txt
```



wc stands for word count. As the meaning suggested, wc is mostly used to do counting. -l here stands for total lines as since dataset_TIST2015_Checkins_v2.txt is a text file, 1 line is equivalent to 1 entry so to get the total number of check-ins, the total number of lines will need to be counted. But the total number of lines includes the column headers so the output from the command above will need to subtract 1 so the column header line is not included as a check-in record.

```
$ cut -d, -f1 dataset_TIST2015_Checkins_v2.txt | sort -u | wc -l
```



The command used above to get the total number of unique users can be explained by using cut to separate the first column, which is user_id, then sort it by -u which mean to get unique values and then using wc -l to count the number of lines left after sorting unique values.

Although the output gave 33253305 but the column header must be removed from the count so 1 is to be subtracted from the output given. Column header is not a user.

There is a total of 33263633 check-ins recorded within the file named dataset_TIST2015_Checkins_v2.txt.
There is a total of 33253304 users recorded within the file named dataset_TIST2015_Checkins_v2.txt.

5. A5 – Dates found within dataset_TIST2015_Checkins_v2.txt

The data recorded to dataset_TIST2015_Checkins_v2.txt was already sorted by date and time. If the file wasn't sorted numerically in accordance to date and time, we can just use the sort command with -n on the specific date column to sort the file.

Using the head and tail commands as shown below, the first 5 and last 5 line of the data recorded within dataset_TIST2015_Checkins_v2.txt will be printed out. -5 within the commands point to the number of rows or lines to be printed out.

```
$ head -5 dataset_TIST2015_Checkins_v2.txt

$ tail -5 dataset_TIST2015_Checkins_v2.txt
```

```
Nicol Foo@KaiYanFoo ~
$ head -5 dataset_TIST2015_Checkins_v2.txt
user_id venue_id        UTC_time        timezone_offset
50756   4f5e3a72e4b053fd6a4313f6        Tue Apr 03 18:00:06 +0000 2012  240
190571  4b4b87b5f964a5204a9f26e3        Tue Apr 03 18:00:07 +0000 2012  180
221021  4a85b1b3f964a520eefe1fe3        Tue Apr 03 18:00:08 +0000 2012  -240
66981   4b4606f2f964a520751426e3        Tue Apr 03 18:00:08 +0000 2012  -300

Nicol Foo@KaiYanFoo ~
$ tail -5 dataset_TIST2015_Checkins_v2.txt
16349   4c957755c8a1bfb7e89024f3        Mon Sep 16 23:24:11 +0000 2013  -240
256757  4c8bbb6d9ef0224bd2d6667b        Mon Sep 16 23:24:13 +0000 2013  -180
66425   513e82a5e4b0ed4f0f3bcf2d        Mon Sep 16 23:24:14 +0000 2013  -180
1830    4b447865f964a5204cf525e3        Mon Sep 16 23:24:14 +0000 2013  120
22704   50df4ee5e4b0c48b5a1c2968        Mon Sep 16 23:24:15 +0000 2013  180
```

First date found would be 3rd April 2012 (Tuesday).
Last date found would be 16th September 2013 (Monday).

6. A6 – Unique venue IDs found within dataset_TIST2015_POIs.txt

From the command below to get the first 2 rows of the file, dataset_TIST2015_POIs.txt, it is shown that dataset_TIST2015_POIs.txt does not contain a column header. Hence, the value obtained when counting unique venues IDs does not need to subtract 1 from it.

```
$ head -2 dataset_TIST2015_POIs.txt
```

```
Nicol Foo@KaiYanFoo ~
$ head -2 dataset_TIST2015_POIs.txt
3fd66200f964a52000e71ee3        40.733596       -74.003139      Jazz Club       US
3fd66200f964a52000e81ee3        40.758102       -73.975734      Gym     US
```

```
$ wc -l dataset_TIST2015_POIs.txt
```

```
Nicol Foo@KaiYanFoo ~
$ wc -l dataset_TIST2015_POIs.txt
3680126 dataset_TIST2015_POIs.txt
```

From the command above, we can now know that dataset_TIST2015_POIs.txt have a total of 3680126 recorded data. It is still yet unknown is any of the data recorded in dataset_TIST2015_POIs.txt are a duplicate or not.

```
$ cut -d, -f1 dataset_TIST2015_POIs.txt | sort -u | wc -l
```

```
Nicol Foo@KaiYanFoo ~
$ cut -d, -f1 dataset_TIST2015_POIs.txt | sort -u | wc -l
3680126
```

From the command above, we can see that none of the rows in dataset_TIST2015_POIs.txt are duplicated.

Total number of unique venue IDs found within dataset_TIST2015_POIs.txt is 3680126.

7. A7 – France's unique venue categories found within dataset_TIST2015_POIs.txt

```
$ head -5 dataset_TIST2015_POIs.txt
```

```
Nicol Foo@KaiYanFoo ~
$ head -5 dataset_TIST2015_POIs.txt
3fd66200f964a52000e71ee3        40.733596       -74.003139      Jazz Club       US
3fd66200f964a52000e81ee3        40.758102       -73.975734      Gym     US
3fd66200f964a52000ea1ee3        40.732456       -74.003755      Indian Restaurant       US
3fd66200f964a52000ec1ee3        42.345907       -71.087001      Indian Restaurant       US
3fd66200f964a52000ee1ee3        39.933178       -75.159262      Sandwich Place  US
```

From the command displayed above, we can see that the last column is the column where the country code is recorded at. The country code for France is FR. Hence, what is needed to do to get France data is just to filter the last column to get all the rows that have been recorded with FR.

With the command shown below, it can be explained in words that from the file dataset_TIST2015_POIs.txt, the 5th column is extracted and piped "FR" to grep. wc -l was used to count the number of lines or rows that have the word "FR" after piping "FR" to grep. Grep stands for global regular expression print which is used to search and locate a specific character or string within a file specified. Here, grep was used to find all occasions of "FR" being recorded within the file dataset_TIST2015_POIs.txt on the 5th column.

```
$ cut -f 4,5 dataset_TIST2015_POIs.txt | grep "FR" | sort -u | wc -l
```

```
Nicol Foo@KaiYanFoo ~
$ cut -f 4,5 dataset_TIST2015_POIs.txt | grep "FR" | sort -u | wc -l
384
```

The total number of France's unique venue categories found within dataset_TIST2015_POIs.txt is 19837.

8. A8 – European subset (POIeu.txt)
   a. Creation of European subset (POIeu.txt)

First, we will take a look at dataset_TIST2015_readme_v2.txt

```
$ head -30 dataset_TIST2015_readme_v2.txt
```

```
Nicol Foo@KaiYanFoo ~
$ head -30 dataset_TIST2015_readme_v2.txt
This dataset includes long-term (about 18 months from April 2012 to September 2013) global-scale check-in data collected from Foursquare.


- File dataset_TIST2015_Checkins.txt contains all check-ins with 4 columns, which are:
1. User ID (anonymized)
2. Venue ID (Foursquare)
3. UTC time
4. Timezone offset in minutes (The offset in minutes between when this check-sin occurred and the same time in UTC, i.e., UTC time + offset is the local time)

- File dataset_TIST2015_POIs.txt contains all venue data with 7 columns, which are:
1. Venue ID (Foursquare)
2. Latitude
3. Longitude
4. Venue category name (Foursquare)
5. Country code (ISO 3166-1 alpha-2 two-letter country codes)

- File dataset_TIST2015_Cities.txt contains all 415 cities data with 6 columns, which are:
Venue category ID (Foursquare)
1. City name
2. Latitude (of City center)
3. Longitude (of City center)
4. Country code (ISO 3166-1 alpha-2 two-letter country codes)
5. Country name
6. City type (e.g., national capital, provincial capital)


================================================================================================================================
Please cite our papers if you publish material based on this dataset.

================================================================================================================================
```

Looking further at dataset_TIST2015_POIs.txt, with the command below, we can see that there is 77 different country code recorded within dataset_TIST2015_POIs.txt, which means, there is 77 different countries within dataset_TIST2015_POIs.txt and our aim is to know which of the 77 countries are European countries.

```
$ cut -f5 dataset_TIST2015_POIs.txt | sort -u | wc -l
```

```
Nicol Foo@KaiYanFoo ~
$ cut -f5 dataset_TIST2015_POIs.txt | sort -u | wc -l
77
```

```
$ cut -f5 dataset_TIST2015_POIs.txt | sort | uniq -c
```

```
Nicol Foo@KaiYanFoo ~
$ cut -f5 dataset_TIST2015_POIs.txt | sort | uniq -c
  10610 AE
  19191 AR
   5636 AT
  31875 AU
   2362 AZ
  36826 BE
   2411 BG
   2987 BH
 370064 BR
   6693 BY
  35996 CA
   2930 CH
  94919 CL
  29791 CN
  23417 CO
  18222 CR
   6804 CY
   5707 CZ
  34713 DE
   2735 DK
   5324 DO
   6216 EC
   2170 EE
   6486 EG
  39187 ES
   5651 FI
  19837 FR
  54278 GB
   3574 GH
  18259 GR
   8681 HU
 379978 ID
   3968 IE
   4103 IL
  26775 IN
  34332 IT
   2599 JM
   3034 JO
 215237 JP
   5807 KE
  51148 KR
  26815 KW
   2535 KZ
   4922 LB
   4994 LK
   7924 LV
   2005 MA
   3190 MQ
 166617 MX
 268981 MY
  38536 NL
   3316 NZ
   1420 OM
   6461 PA
  22270 PE
  71810 PH
   3651 PL
   5570 PR
   8721 PT
  14157 PY
   2804 QA
   3858 RO
 227525 RU
  14747 SA
   6389 SE
  33892 SG
   3600 SV
 150576 TH
   3598 TN
 377302 TR
   1820 TT
  29276 UA
 501900 US
   2099 UY
   4265 VE
   4724 VN
   7323 ZA
```

The command above prints out all the 77 country codes and how many of the venues falls under each country code.

It is widely known that Europe have a lot of countries so now we have to identify and remove other countries that are not European countries within the list shown above. Using country code is a method to extract all European countries out but I have used longitude and latitude bounding range to do this.

With the use of the map below, I have determined the range for longitude and latitude.

```
$ awk -F '\t' '$2>=33 && $2<=73 && $3>=-27 && $3<=47' dataset_TIST2015_POIs.txt > POIeu.txt
```

```
Nicol Foo@KaiYanFoo ~
$ awk -F '\t' '$2>=33 && $2<=73 && $3>=-27 && $3<=47' dataset_TIST2015_POIs.txt > POIeu.txt
```

'\t' used above means the delimiter is tab which is essentially whitespace. > within the command above is used to save the extracted European countries to a text file named POIeu.txt. The extraction of European countries from dataset_TIST2015_POIs.txt is done using awk.

b. European country with most and least venues

```
$ cut -f5 POIeu.txt | sort | uniq -c
```

```
Nicol Foo@KaiYanFoo ~
$ cut -f5 POIeu.txt | sort | uniq -c
   5636 AT
  36826 BE
   2411 BG
   6693 BY
   2930 CH
   6804 CY
   5707 CZ
  34713 DE
   2735 DK
   2170 EE
  39187 ES
   5651 FI
  19837 FR
  54278 GB
  18259 GR
   8681 HU
   3968 IE
  34332 IT
   4922 LB
   7924 LV
   2005 MA
  38536 NL
   3651 PL
   8721 PT
   3858 RO
 162353 RU
   6389 SE
   3598 TN
 377302 TR
  29276 UA
```

The command above can be explained in words as cutting the 5th column of POIeu.txt and sort it by counting the unique country code

The European country with the most venues is TR which is a country code for Turkey.
The European country with the least venues is EE which is a country code for Estonia.

c. European country with most seafood restaurants

The 4th column of the file contains the venues' names so to find restaurants we only need to look at the 4th column. But first, with the command below, we can see the total amount of seafood restaurant in POIeu.txt.

```
$ grep -o 'Seafood Restaurant' POIeu.txt | sort | uniq -c
```

```
Nicol Foo@KaiYanFoo ~
$ grep -o 'Seafood Restaurant' POIeu.txt | sort | uniq -c
   2568 Seafood Restaurant
```

Now, to get the numbers of seafood restaurant for each European country recorded in POIeu.txt, we will have to extract column 4 and 5 then grab the rows that have the word 'Seafood Restaurant' then sort the

remaining rows before counting the unique values based on the 5th column. 4th column is basically the venue's name whilst 5th column is the country code used to identify the countries each venue belongs to.

```
$ cut -f4,5 POIeu.txt | grep 'Seafood Restaurant' | sort | uniq -c
```

```
Nicol Foo@KaiYanFoo ~
$ cut -f4,5 POIeu.txt | grep 'Seafood Restaurant' | sort | uniq -c
     16 Seafood Restaurant      AT
     63 Seafood Restaurant      BE
      6 Seafood Restaurant      BG
      2 Seafood Restaurant      BY
      2 Seafood Restaurant      CH
     25 Seafood Restaurant      CY
      6 Seafood Restaurant      CZ
     76 Seafood Restaurant      DE
      6 Seafood Restaurant      DK
      2 Seafood Restaurant      EE
    123 Seafood Restaurant      ES
      2 Seafood Restaurant      FI
     39 Seafood Restaurant      FR
    108 Seafood Restaurant      GB
    110 Seafood Restaurant      GR
      6 Seafood Restaurant      HU
      7 Seafood Restaurant      IE
    134 Seafood Restaurant      IT
     22 Seafood Restaurant      LB
      5 Seafood Restaurant      LV
     10 Seafood Restaurant      MA
     94 Seafood Restaurant      NL
      1 Seafood Restaurant      PL
     57 Seafood Restaurant      PT
      6 Seafood Restaurant      RO
     66 Seafood Restaurant      RU
     15 Seafood Restaurant      SE
     11 Seafood Restaurant      TN
   1522 Seafood Restaurant      TR
     26 Seafood Restaurant      UA
```

From the output above, we can easily identify the European country with the most seafood restaurant which is Turkey with the country code of 1522.

d. Common restaurant venues within European country

Following the thinking process for A8c, what was done in the command is that only column 4 will be extracted from POIeu.txt as the 4th column is essentially the venue's name. Then the word 'Restaurant' was grab using grep then the extracted rows will be sorted and counted accordingly to unique venue names that contains the word 'Restaurant'.

```
$ cut -f4 POIeu.txt | grep 'Restaurant' | sort | uniq -c
```

```
Nicol Foo@KaiYanFoo ~
$ cut -f4 POIeu.txt | grep 'Restaurant' | sort | uniq -c
    182 Afghan Restaurant
    325 African Restaurant
   1357 American Restaurant
    182 Arepa Restaurant
    319 Argentinian Restaurant
   2429 Asian Restaurant
     67 Australian Restaurant
    187 Brazilian Restaurant
     73 Cajun / Creole Restaurant
    137 Caribbean Restaurant
   2236 Chinese Restaurant
     96 Cuban Restaurant
     96 Dim Sum Restaurant
    131 Dumpling Restaurant
   1723 Eastern European Restaurant
     77 Ethiopian Restaurant
    593 Falafel Restaurant
   8750 Fast Food Restaurant
     23 Filipino Restaurant
   2914 French Restaurant
   1100 German Restaurant
     51 Gluten-free Restaurant
   1507 Greek Restaurant
   1374 Indian Restaurant
     67 Indonesian Restaurant
   7745 Italian Restaurant
   1783 Japanese Restaurant
    208 Korean Restaurant
     95 Latin American Restaurant
     53 Malaysian Restaurant
   2129 Mediterranean Restaurant
    767 Mexican Restaurant
   2825 Middle Eastern Restaurant
    128 Molecular Gastronomy Restaurant
     27 Mongolian Restaurant
    179 Moroccan Restaurant
     55 New American Restaurant
    131 Paella Restaurant
     37 Peruvian Restaurant
    422 Portuguese Restaurant
  15362 Restaurant
    326 Scandinavian Restaurant
   2568 Seafood Restaurant
     89 South American Restaurant
     58 Southern / Soul Food Restaurant
   1913 Spanish Restaurant
   2199 Sushi Restaurant
    137 Swiss Restaurant
   1457 Tapas Restaurant
    755 Thai Restaurant
  10104 Turkish Restaurant
    522 Vegetarian / Vegan Restaurant
    339 Vietnamese Restaurant
```

The most common restaurant venue is Europe is “Restaurant” with a total of 15362.

Processing Data with Cygwin64 Terminal

The following commands assist in navigation of the compressed folder obtained from Moodle:

```
$ pwd

$ cp /cygdrive/c/Users/Nicol\ Foo/Downloads/Twitter_Data_1.gz .

$ ls
```

```
Nicol Foo@KaiYanFoo ~
$ pwd
/home/Nicol Foo

Nicol Foo@KaiYanFoo ~
$ cp /cygdrive/c/Users/Nicol\ Foo/Downloads/Twitter_Data_1.gz .

Nicol Foo@KaiYanFoo ~
$ ls
Twitter_Data_1                    dataset_TIST2015_POIs.txt
Twitter_Data_1.gz                 dataset_TIST2015_readme_v2.txt
dataset_TIST2015.tar              european_countries.txt
dataset_TIST2015_Checkins_v2.txt  file.txt
dataset_TIST2015_Cities.txt
```

Now, we will compress Twitter_Data_1.gz using gunzip as Twitter_Data_1.gz have .gz extension.

```
$ gunzip Twitter_Data_1.gz
```

```
Nicol Foo@KaiYanFoo ~
$ gunzip Twitter_Data_1.gz
```

With the command ls, we can see that within Twitter_Data_1.gz, there is Twitter_Data_1. Now, we will view the first 3 lines of Twitter_Data_1 to get a view on how Twitter_Data_1 look like.

```
$ ls

$ head -3 Twitter_Data_1
```

```
Nicol Foo@KaiYanFoo ~
$ ls
Twitter_Data_1                    dataset_TIST2015_POIs.txt
Twitter_Data_1.gz                 dataset_TIST2015_readme_v2.txt
dataset_TIST2015.tar              european_countries.txt
dataset_TIST2015_Checkins_v2.txt  file.txt
dataset_TIST2015_Cities.txt

Nicol Foo@KaiYanFoo ~
$ head -3 Twitter_Data_1
433213478539513856      TRY_Sound       Tue Feb 11 12:18:36 +0000 2014  またたび食べると一時的に楽しくなるし、血行良くなるから頭痛も無くなるけど、覚めた後死ぬ。が食べる。うまい
433213478543716352      kengoushougun_  Tue Feb 11 12:18:36 +0000 2014  我に優しくない世界になりそうだな… #剣豪義輝bot
433213478535327744      TyphaineArmy    Tue Feb 11 12:18:36 +0000 2014  Pour rassurer les gens qui n'ont pas pu regarder le live,personne ne viole la fille.
```

Now we will check the delimiter and the number of columns with the command below:

```
$ head -1 Twitter_Data_1 | less

/<tab>
```

From the 2 commands above, we can see that Twitter_Data_1 have 4 columns and its delimiter is once again, whitespace like tab ‘ ’. We can also see the brief content of Twitter_Data_1 which goes by tweet_id, tweet_author, tweet_date_time and tweet_content. The names of each column given is given by myself as there is no column header provided within Twitter_Data_1.

Task B

1. B1 – Donald Trump tweets within the dataset

   We will look at the 4th column which is the tweet_content to check the number of lines whereby “Donald Trump” was being mentioned in the tweets recorded within Twitter_Data_1.

   ```
   $ cut -f4 Twitter_Data_1 | grep “Donald Trump” | wc -l
   ```

   ```
   Nicol Foo@KaiYanFoo ~
   $ cut -f4 Twitter_Data_1 | grep "Donald Trump" | wc -l
   109
   ```

   There is 109 lines whereby “Donald Trump” was mentioned within Twitter_Data_1.

   ```
   $ grep -o “Donald Trump” Twitter_Data_1 | wc -l
   ```

   ```
   Nicol Foo@KaiYanFoo ~
   $ grep -o "Donald Trump" Twitter_Data_1 | wc -l
   116
   ```

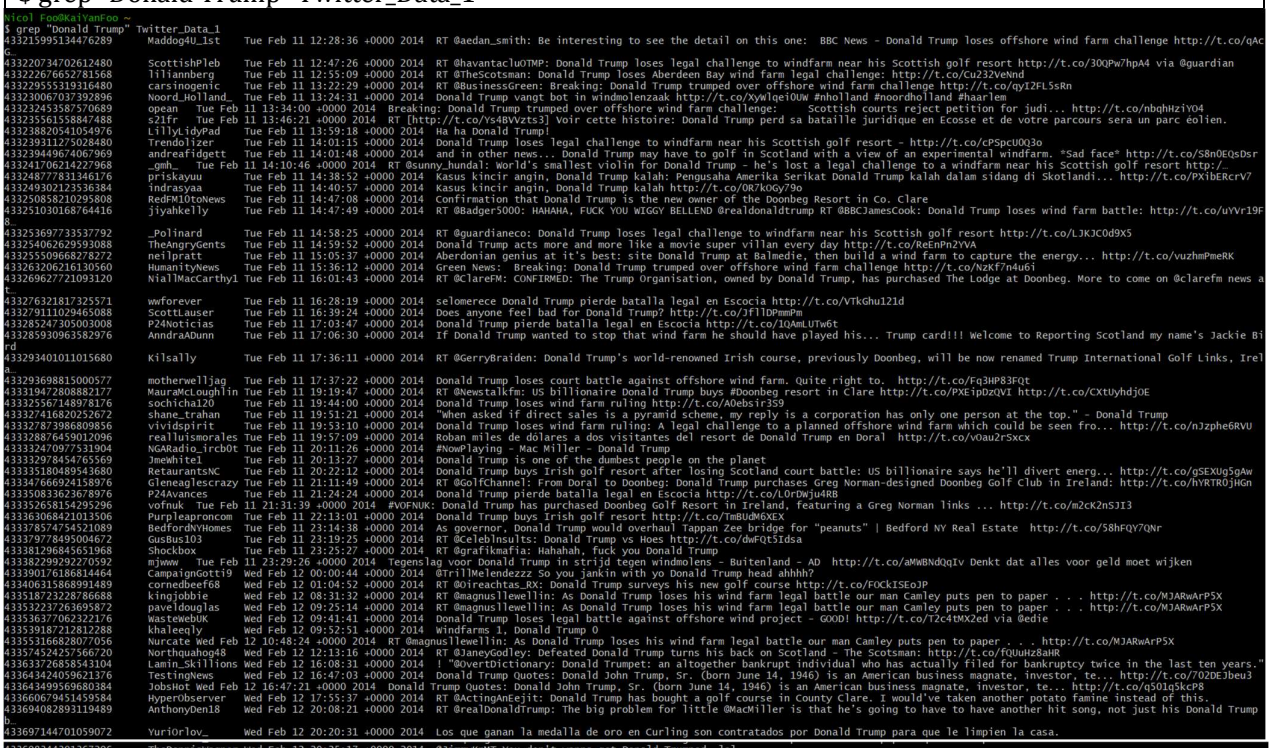
   -o flag used in grep is to match multiple occurrences of “Donald Trump” happening within the tweets.

"Donald Trump" has been mentioned 116 times in tweets recorded within Twitter_Data_1.

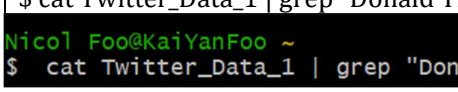
2. B2 – Donald Trump discussion over a period of time in Twitter

Extracting tweets that contain "Donald Trump" and printing them out with the command below:

```
$ grep "Donald Trump" Twitter_Data_1
```




```
$ cat Twitter_Data_1 | grep "Donald Trump" > donald1.txt
```



The command above extracts all the lines with their respective columns as long as these lines have the word "Donald Trump" but we only need to extract the time column from the above output. Hence, awk was

used after grep to help split the columns by the delimiter, tab whitespace, then get the 3rd column values to be extracted to a separate file with the name 'donald_time.txt'. Grep allows us to separate the lines containing the word "Donald Trump".

```
$ grep -i -w "Donald Trump" Twitter_Data_1 | awk -F'\t' '{print $3}' > donald_time.txt
```

```
Nicol Foo@KaiYanFoo ~
$ grep -i -w 'Donald Trump' Twitter_Data_1 | awk -F'\t' '{print $3}' > donald_time.txt
```

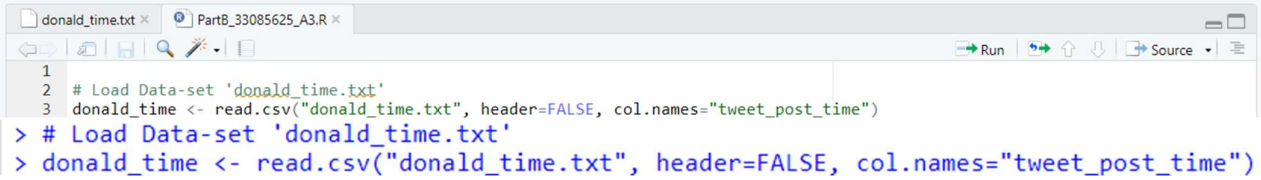
The command above can be explained in 2 parts:

- grep is used to grab and extract all lines with "Donald Trump"
- awk is used to extract all the timestamps from column 3 and > is used to save the extracted timestamps to donald_time.txt

With the file, we will now move on to RStudio Cloud to visualise donald.txt as a histogram. But first, what will be done is that the contents in donald_time.txt will be formatted. RStudio Cloud is a web version of RStudio.

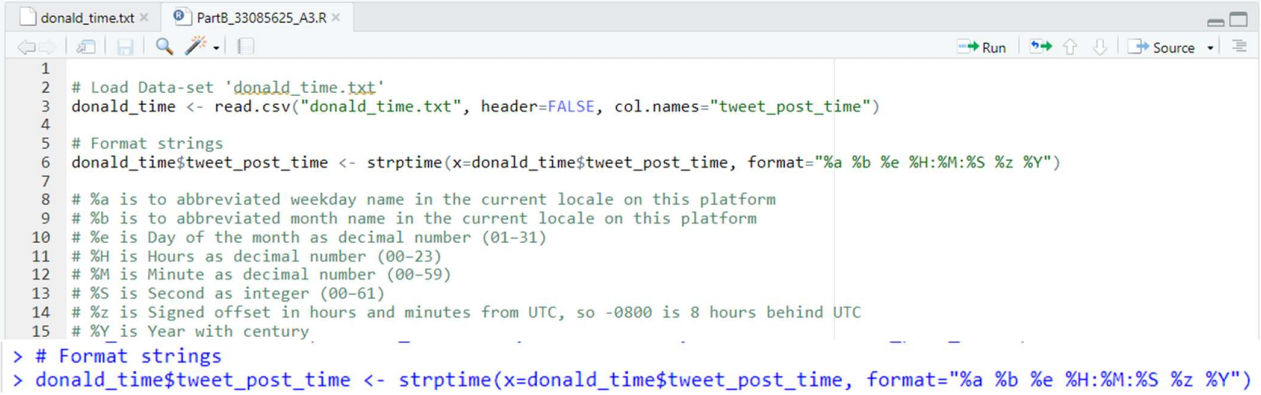
The command below is to read donald_time.txt to R as a CSV using read_csv() and naming the column as tweet_post_time with col.names.

```
> donald_time <- read.csv("donald_time.txt", header=FALSE, col.names="tweet_post_time")
```



The command below is to parse the particular date and time format within donald_time.txt.

```
> # Format strings
> donald_time$tweet_post_time <- strptime(x=donald_time$tweet_post_time,
format="%a %b %e %H:%M:%S %z %Y")
```



The format string "%a %b %e %H:%M:%S %z %Y" is chosen to correspond to the column itself:

- %a – Weekday names
- %b – Month names
- %e – Day of the month
- %H:%M:%S – Hour:Minute:Second
- %z – Signed offset in hours and minutes
- %Y – Year with century

The command below is to check the data within donald_time.txt is in R-recognizable type for timestamps.

```
> # Check output after formatting string
> str(donald_time)
```

```
17  # Check output after formatting string
18  str(donald_time)
19
> # Check output after formatting string
> str(donald_time)
'data.frame':   120 obs. of  1 variable:
 $ tweet_post_time: POSIXlt, format: "2014-02-11 12:28:36" "2014-02-11 12:47:26" "2014-02-11 12:55:09" "2014-02-11 13:22:29" ...
```

3. B3 – Histogram of Donald Trump discussion over a period of time in Twitter

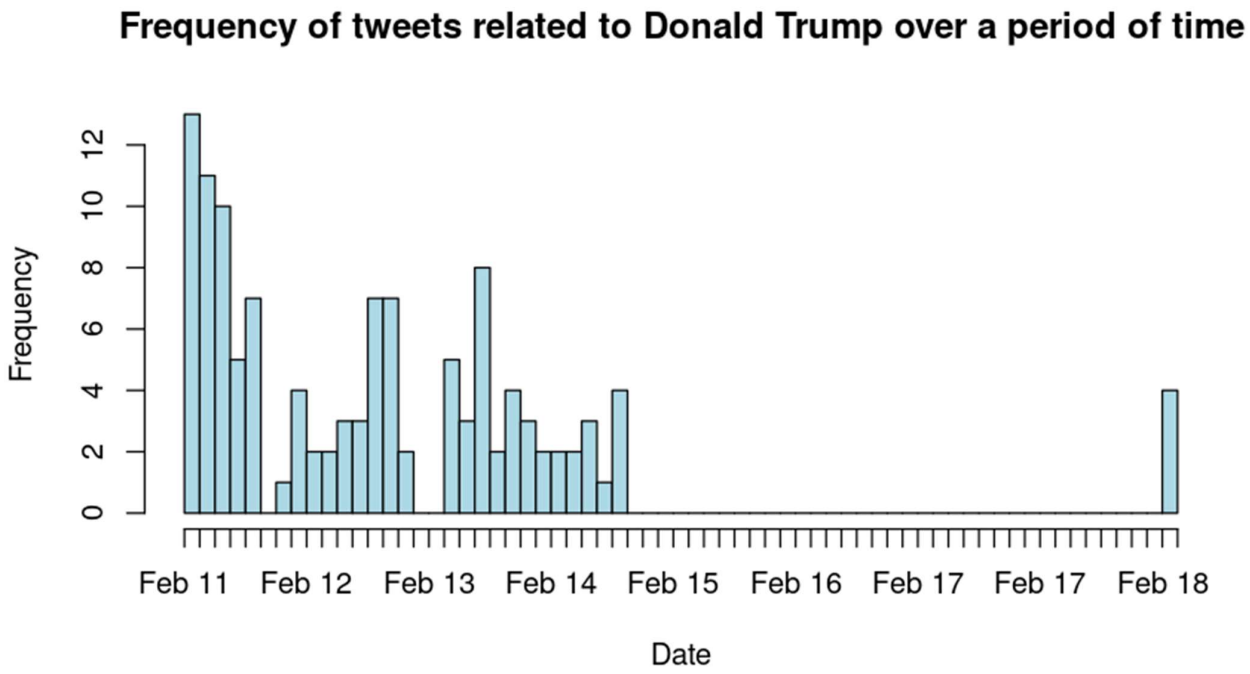
Using hist(), a histogram can be created in R. The command below can be explained as:

- donald_time$tweet_post_time is essentially the data to be plotted within the histogram.
- main = "Frequency of tweets related to Donald Trump over a period of time" is the title of the histogram
- xlab = "Date" is the x-axis label of the histogram
- col = "light blue" is to specify the colour for the columns of the histogram
- breaks = 47 is basically the bins of the histogram and here we have 47 bins
- freq = TRUE is to allow y-axis to be labelled as frequency which shows the frequency of a particular data

```
> # Histogram
> hist(donald_time$tweet_post_time, main = "Frequency of tweets related to Donald Trump over a period of time", xlab = "Date", col = "light blue", breaks = 47, freq = TRUE)
```

```
20  # Histogram
21  hist(donald_time$tweet_post_time, main = "Frequency of tweets related to Donald Trump over a period of time", xlab = "Date", col = "light blue", breaks = 47, freq = TRUE)
> # Histogram
> hist(donald_time$tweet_post_time, main = "Frequency of tweets related to Donald Trump over a period of time", xlab = "Date", col = "light blue", breaks = 47, freq = TRUE)
Warning message:
In breaks[-1L] + breaks[-nB] : NAs produced by integer overflow
```

The histogram output is displayed as below:



4. B4 – Histogram pattern

The histogram displayed above have a usual shape but it can be seen that the histogram is rightly-skewed and is not symmetric. This means that the median of donald_time.txt is greater than the mean of donald_time.txt. It also can be deducted that before 15th February, Donald Trump was quite a hot-topic around the start of February in the Twitter platform as something related to Donald Trump may have happened that caused a huge disturbance and discussion amongst the netizens. Before 15th February, the discussions surrounding Donald Trump in the Twitter platform peaked at 11th February and slowly declined as the day goes on but has risen up slowly throughout 12th February and declined on the start of 13th February. Donald Trump must have made a minor incident that aroused netizen to discuss online on

Twitter from the end of 13th February to the end of 14th February. It is unknown why at the start of 15th February that any tweets related to Donald Trump were nowhere to be found until 18th February. It may have been the news and discussion surrounding Donald Trump was suppressed by the person himself or netizens just have a loss of interest regarding incidents or actions of Donald Trump.

5. B5 – Histogram of the accounts in Twitter

```
$ cut -f2 Twitter_Data_1 | sort | uniq -c | sort > b5_author_tweets.txt
```

```
Nicol Foo@KaiYanFoo ~
$ cut -f2 Twitter_Data_1 | sort | uniq -c | sort > b5_author_tweets.txt
```

The command above can be explained in 5 parts:

- Separate the 2nd column from Twitter_Data_1 with using cut -f2
- Sort the 2nd column from Twitter_Data_1 with using sort
- Count all the unique values from the sorted 2nd column from Twitter_Data_1 with using uniq -c
- Sort again the unique values counted from the sorted 2nd column from Twitter_Data_1 in ascending order with using sort
- Extract the output to a file named with 'b5_author_tweets.txt'

The R code commands below can be explained as:

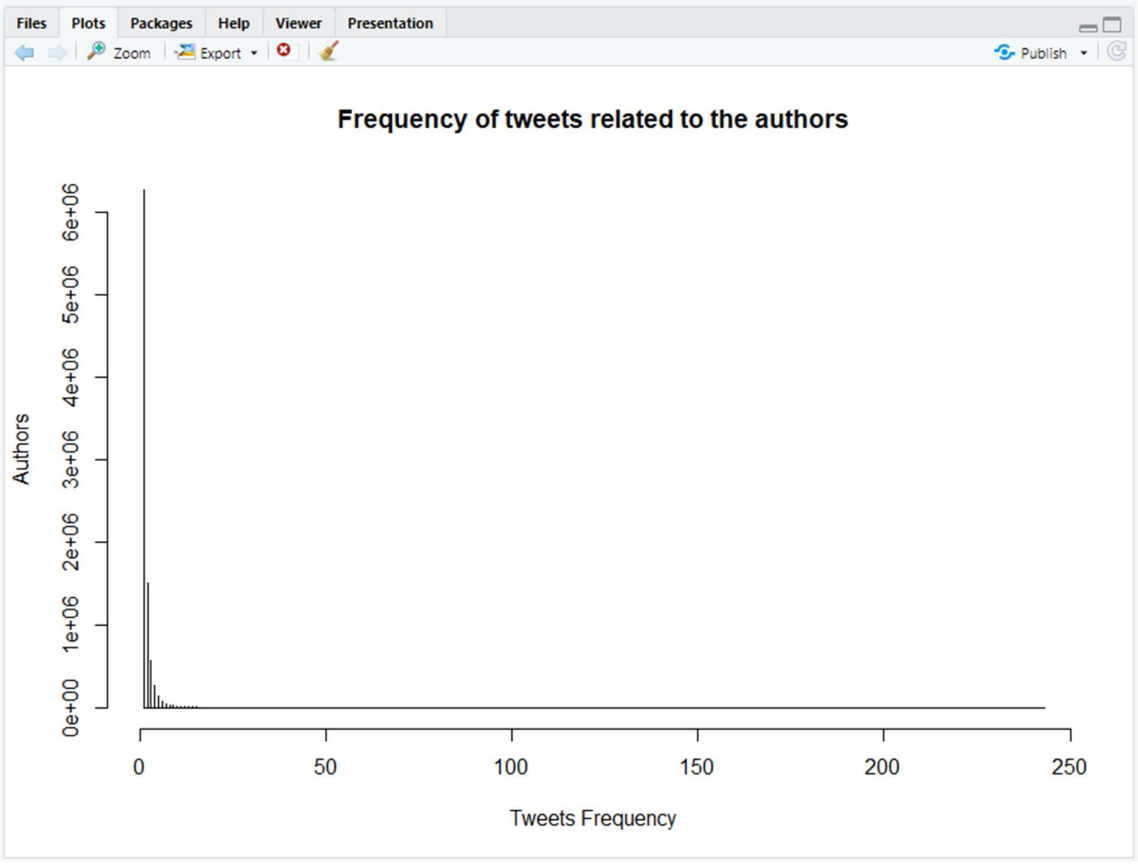
- > tweet_author <- read.csv("b5_author_tweets.txt", header=FALSE, sep="")
  - read.csv("b5_author_tweets.txt") → To read the text file, b5_author_tweets.txt, as a csv into RStudio under the variable tweet_author
  - header=FALSE → No header is required to be placed
  - sep = "" → Separate the data by whitespace
- > tweet_author
  - Show the data after the command above was executed
- > head(tweet_author)
  - Show the first few lines of the data after the command on read.csv() was executed
- > summary(tweet_author)
  - Give a basic summary on the statistical result of the data within b5_author_tweets.txt
- > hist(tweet_author$V1, main = "Frequency of tweets related to the authors", xlab = "Tweets Frequency", ylab = "Authors",col = "light blue", breaks = 999999)
  - Produce a histogram on the 1st column of the file which is shown as tweet_author$V1
  - main = "Frequency of tweets related to the authors" → Title of the histogram
  - xlab = "Tweets Frequency", ylab = "Authors" → Labels for x and y axis of histogram
  - col = "light blue" → colour of the bars of the histogram
  - breaks = 999999 → bins for the histogram, here it is 999999 bins

```
> # Part B5
> tweet_author <- read.csv("b5_author_tweets.txt", header=FALSE, sep="")
> tweet_author
> head(tweet_author)

> summary(tweet_author)

> # Histogram
> hist(tweet_author$V1, main = "Frequency of tweets related to the authors", xlab = "Tweets Frequency", ylab = "Authors",col = "light blue", breaks = 999999)
```

```
Console  Terminal  Background Jobs
R 4.2.1 · ~/
> # Part B5
> tweet_author <- read.csv("b5_author_tweets.txt", header=FALSE, sep="")
> tweet_author
    V1              V2
1    1
2    1 000000000003737
3    1   0000000000_24
4    1 0000000000yours
5    1     00000000Ksa
6    1  0000000Anonimo
7    1     0000000Suki
8    1       0000000_8
9    1        0000000g
10   1       0000000sh
11   1    0000000zeroo
12   1     00000010You
13   1      0000001524
14   1       000000430
15   1       000000567
16   1    000000Nourah
17   1    000000_Kyoru
18   1      000000dani
19   1        000000zz
20   1       000001080
21   1       000001125
22   1   0000025Mikako
```

```
Console  Terminal  Background Jobs
R 4.2.1 · ~/
500  1     001Flowerfp
 [ reached 'max' / getOption("max.print") -- omitted 8977404 rows ]
> head(tweet_author)
  V1              V2
1  1
2  1 000000000003737
3  1   0000000000_24
4  1 0000000000yours
5  1     00000000Ksa
6  1  0000000Anonimo
> summary(tweet_author)
       V1                V2
 Min.   :  1.000   Length:8977904
 1st Qu.:  1.000   Class :character
 Median :  1.000   Mode  :character
 Mean   :  1.681
 3rd Qu.:  2.000
 Max.   :243.000
> # Histogram
> hist(tweet_author$V1,
+      main = "Frequency of tweets related to the authors",
+      xlab = "Tweets Frequency",
+      ylab = "Authors",
+      col = "light blue",
+      breaks = 999999,
+      freq = TRUE)
```

Histogram:



The histogram above is the output from RStudio Desktop version. b5_author_tweets.txt is a large file and RStudio Cloud could not process such a huge file so RStudio Desktop version was used to complete Part B Question 5 instead.

The histogram above has small bar width even though the bins limit was set to the maximum which means that the file's data itself is too big to be processed. Hence, the display on the histogram is very small.

The x-axis of the histogram is number of tweets by authors while y-axis is the author frequency. The histogram is rightly-skewed and is not symmetric. We can see that from the histogram above, most Twitter authors usually post in Twitter around less than 25 times. The lesser the Tweets frequency, the higher the count of authors are.