CS2006 Python Practical 2

Report

Introduction

This practical specified the development of a project to analyse large data sets using Python and Jupyter notebooks in order to explore the use of the Python ecosystem's libraries for data analysis and visualisation.

The level of completeness with the respect to the requested functionality is as follows (all of the basic, easy, medium, and medium/hard requirements have been implemented, as well as one hard requirements):

Basic Specification:

Refine the dataset

Perform the descriptive analysis of the dataset

Building plots/visualisations

Additional Requirements:

Easy

Analyse applications used to send tweets

Extend the descriptive analysis, for example, by calculating the average number of times each user being retweeted and the average number of time each user being replied

Medium

Analyse patterns of user activity over the period covered by the dataset

Medium to Hard

Analyse interactions between users by constructing, visualising and analysing the graph with vertices corresponding to users and edges corresponding to their connections by means of retweets, replies and mentions. Determine some interesting properties of this graph and produce some visualisations.

Hard*

*This requirement was implemented on Mac OS and ran successfully (as shown in the pdf output file) however when run on the lab machines the widgets used to allow interaction do not display correctly

Implement interactive visualisations

Known Problems

- Unfortunately our hard requirement to display interactive visualisations does not display the widgets that are used to allow users to enter input correctly on the lab machines. However output of this requirement is available in the output pdf file.
- For some requirements the analysis is based on a subset of the total data set and may inconsistent if the data were analysed as a whole. For example in the medium requirement to analyse patterns of user activity over the period covered by the data set, a world map has been displayed with data points of all of the users locations at the time of tweeting plotted on the map. The data points can be selected which then displays the user's twitter handle as well as the time they tweeted. This was done to establish a pattern of user activity by mapping the country of origin for each user. However only just over four hundred user's had valid geo-location data available and so the data may be unrepresentative of the more the 70,000 tweets.

  Nevertheless, the data indicates that a large proportion of the users are located in the western world (more specifically Europe) and so graphs have been produced for users of British English, French and German to visualise their connections as part of the medium to hard networkx extension.

Problems Solved

Initially there was some difficulty coordinating the installation of libraries between the user profiles of our group members as new features would be developed with different packages. This problem was solved through the use of a requirements file which is run using "**pip3 install -r requirements**" and contains the names of all of the libraries/packages required to run our code.

Level of Reproducibility and Reusability of Analysis

Reproducing and reusing the data that we have used for our analysis is very easy due to the fact that our code is contained only within the Jupyter Notebook. To run individual stages of the notebook in sequence **shift + Enter** can be used to execute each cell or alternatively the whole notebook can be run by clicking the drop down **kernel** and then selecting the **restart and run all** option.

The notebook could potentially be run with other twitter data sets if a csv of the same format was provided in the data/ folder, but we have not attempted this.

Summary of Provenance

160001362 - Refine the dataset, Perform the descriptive analysis of the dataset, Analyse patterns of user activity over the period covered by the dataset,

160021429 - Refine the dataset, Perform the descriptive analysis of the dataset, Analyse interactions between users by constructing, visualising and analysing the graph with vertices corresponding to users and edges corresponding to their connections by means of retweets, replies and mentions. Determine some interesting properties of this graph and produce some visualisations.,

160016245 - Building plots/visualisations, Analyse applications used to send tweets, Extend the descriptive analysis, for example, by calculating the average number of times each user being retweeted and the average number of time each user being replied, Implement interactive visualisations

0. Import the Required Libraries

To install modules, you need to use pip3.

As we mentioned in the README.txt file, we made the text file called req.txt to install all python modules by using single line.

Usage: pip3 install -r req.txt

In [1]:
```
import pandas as pd
```

In [2]:
```
import numpy as np
```

In [3]:
```
import matplotlib.pyplot as plt
```

In [4]:
```
import json
```

In [5]:
```
import networkx as nx
```

In [6]:
```
from wordcloud import WordCloud as wc
```

In [7]:
```
from collections import Counter
```

In [8]:
```
import folium
```

In [9]:

```
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
from IPython.display import display
```

In [10]:

```
df=pd.read_csv("../data/CometLanding.csv")
```

Check the Initial Length of the CSV File

In [11]:

```
len(df)
```

Out[11]:

```
77319
```

Check the Data Type of Each Column

In [12]:

```
df.dtypes
```

Out[12]:

```
id_str                        float64
from_user                      object
text                           object
created_at                     object
time                           object
geo_coordinates                object
user_lang                      object
in_reply_to_user_id_str       float64
in_reply_to_screen_name        object
from_user_id_str              float64
in_reply_to_status_id_str     float64
source                         object
profile_image_url              object
user_followers_count          float64
user_friends_count            float64
status_url                     object
entities_str                   object
dtype: object
```

1. Refine the Dataset

1) Check Data File for Duplicated Tweets and Remove Duplicates

In [13]:

```
df.drop_duplicates(['id_str'], inplace = True)
```

The raw data contained some duplicate tweets, which were removed, and the number of total remaining unique tweets is displayed below:

In [14]:

```
len(df)
```

Out[14]:

77268

2) Remove Text That is Null

In [15]:

```
df = df[df['text'].notnull()] #find all rows that the text column is not null an
d save that data frame.
```

Moreover, we delted the null texts, as the normal tweets should contain texts in it.

The number of total not empty tweets is displayed below:

In [16]:

```
len(df)
```

Out[16]:

77267

2. Perform the Descriptive Analysis of the Dataset

In this section, we calculated the total number of tweets, retweets, and replies. The average number of tweets, retweets, and replies are also calculated. Moreover, we identified the most popular hashtag at the end of this section.

1.1) Calculate the Total Number of Tweets

In [17]:

```
tweets = (df[(df['in_reply_to_user_id_str'].notnull() == False) #Filters out rep
lies
                  & (df['text'].str.startswith('RT', na=False) == False)]) #Fil
ters out retweets
numTweets = len(tweets)
```

In [18]:

```
print("Total number of tweets (excluding replies and retweets): "
      + str(numTweets)) #Filters out retweets
```

```
Total number of tweets (excluding replies and retweets): 15582
```

1.2) Calculate the Total Number of Retweets

In [19]:

```
dfRT = df[df.text.str.startswith('RT', na=False)] #find all texts that the text
 starts with "RT"
```

In [20]:

```
numRT = len(dfRT)
```

In [21]:

```
print("Total number of retweets: " + str(numRT))
```

```
Total number of retweets: 59998
```

1.3) Calculate the Total Number of Replies

In [22]:

```
dfReplies = df[df['in_reply_to_user_id_str'].notnull()] #find all rows that the
 in_reply_to_user_id_str is not null
```

In [23]:

```
numReplies = len(dfReplies)
```

In [24]:

```
print("Number of replies: " + str(numReplies))
```

```
Number of replies: 1723
```

2) Calculate the Total Number of Different Users Tweeting in this DataSet

In [25]:

```
numUsers = len(df['from_user'].unique()) #count the number of all unique "from_u
ser"
```

In [26]:

```
print("The number of users: " + str(numUsers))
```

```
The number of users: 50195
```

3.1) Calculate the Average Number of Tweets by a User

In [27]:

```
print("Average number of tweets: ", str(numTweets / numUsers))
```

```
Average number of tweets:  0.31042932563004283
```

3.2) Calculate the Average Number of Retweets by a User

In [28]:

```
print("Average number of retweets: ", str(numRT / numUsers))
```

```
Average number of retweets:  1.195298336487698
```

3.3) Calculate the Average Number of Replies Sent by a User

In [29]:

```
print("Average number of replies: ", str(numReplies / numUsers))
```

```
Average number of replies:  0.03432612810040841
```

By comparing the average number of tweets, replies and retweets per user it is clear that the average user over the period of this dataset consumed more twitter content than they produced. This is observation is explained later in the section analysing how users in the data set are connected.

4) Identify the Most Popular Hashtags

In [30]:

```
hashtags_list = df.text.str.findall(r'#.*?(?=\s|$)') #Gets all text starting wit
h a hash
hashtags = hashtags_list.tolist() #Converts data to list format
hashtags = [item for sublist in hashtags for item in sublist] #Flattens list of
 lists to list.
count = Counter(hashtags) #Gets the counts for each hashtag
most_popular = ((count.most_common())[0:25]) #Gets the list in order of most pop
ular
for hashtag in most_popular: print(hashtag[0], hashtag[1])
```

```
#CometLanding 53441
#cometlanding 10486
#67P 5872
#Rosetta 4978
#Philae 2613
#CometLanding: 1745
#Cometlanding 1004
#CometLanding. 981
#WishKoSaPasko 970
#HappyBirthdaySandaraPark 965
#67P/CG 701
#esa 676
#rosetta 675
#SEP 623
#philae 588
#PoseToiPhilae 535
#Comâ€¦ 514
#CometLanding, 462
#space 443
#comâ€¦ 422
#cometlanding, 413
#CometLâ€¦ 408
#67P. 406
#rosettamission 404
#ESA 396
```

From the above most popular hashtags it is clear that the hashtags feature prominently terms relating to the comet landing. This is clear not just with the hashtags containing the phrase but also with terms such as Rosetta (referencing the European space agency probe), Philae (the Rosetta lander module), as well as 67P (the name of the comet being landed on).

3. Build plots/visualisations

3.1) The structure of the dataset

In [31]:

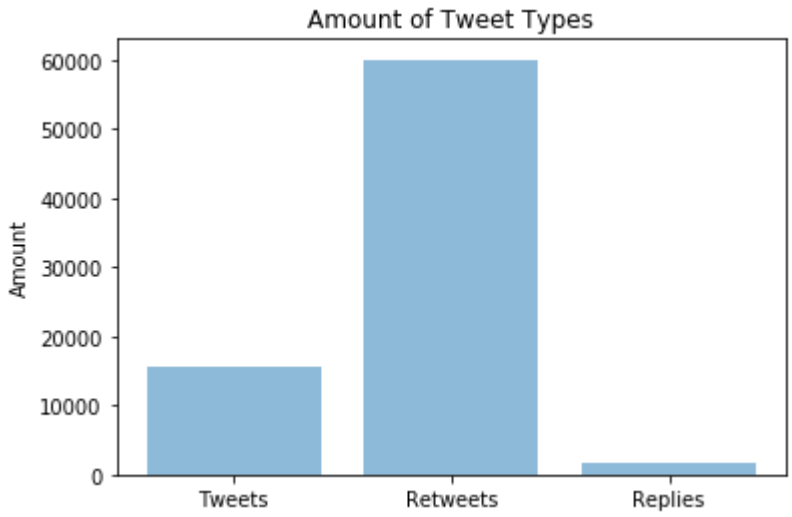
```
structure_names = ('Tweets', 'Retweets', 'Replies') #list for columns
```

In [32]:

```
structure_data = [numTweets, numRT, numReplies] #list for amount of tweets
```

In [33]:

```
pos = np.arange(len(structure_names))

plt.bar(pos, structure_data, align='center', alpha=0.5) #plot a bar chart using
 two lists
plt.xticks(pos, structure_names)
plt.ylabel('Amount')
plt.title('Amount of Tweet Types')
plt.show()
```



Most of the tweets are actually retweets. Unique tweets do not dominate the data set. The data set has an incredibly huge difference between retweets and replies. This is surprising because an estimate of 27% of total tweets on twitter are retweets and 24% are replies (January 2014)[1].

[1] - https://mislove.org/publications/Profiles-ICWSM.pdf (https://mislove.org/publications/Profiles-ICWSM.pdf) - The Tweets They are a-Changin': Evolution of Twitter Users and Behavior.

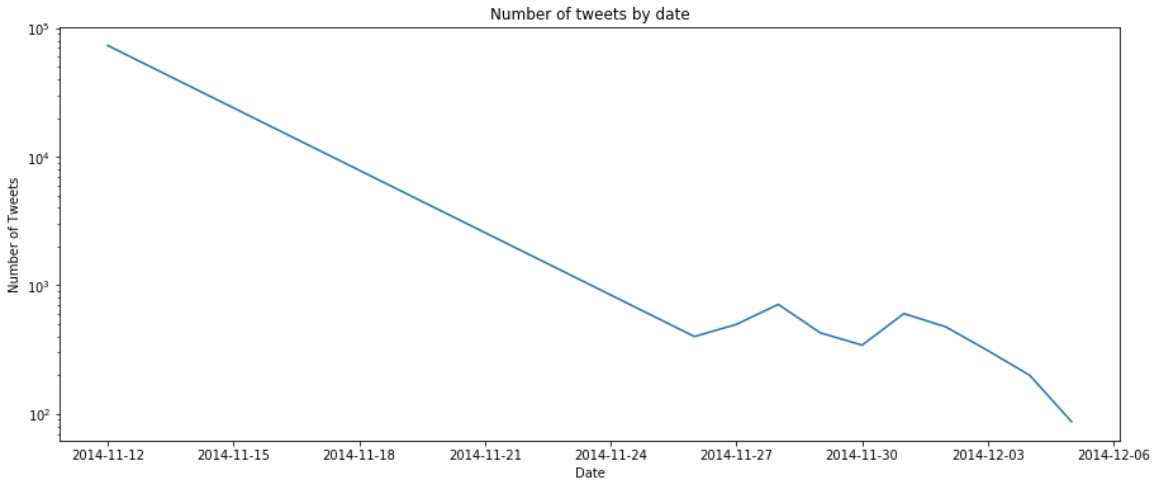
3.2) The timeline of the tweets activity

In [34]:

```
df["date_time"] = [str(x) for x in df["time"]] # Convert all entries in the time
 column of the dataframe to strings
df["date_time"] = [x.split(" ")[0] for x in df["time"]] # Get just the date part
 of the string
df["date_time"] = pd.to_datetime(df["date_time"], dayfirst=True) # convert the c
olumn to datetime
```

In [35]:

```
plt.rcParams["figure.figsize"] = [15, 6] # Make the graph wider to fit in the da
tes

values = pd.DataFrame(df.date_time.value_counts().reset_index()) # Count the num
ber of occurrances
values.columns = ["date", "amount"] # label the columns accordingly
values = values.sort_values(by="date")
plt.plot(values.date, values.amount) # Get just the date part of the string
plt.yscale("log") # Scale logarithmically - 12th November has the majority of th
e tweets
plt.title("Number of tweets by date")

plt.ylabel('Number of Tweets')
plt.xlabel('Date')
plt.show()
```



This graph shows perfectly how "hype" works on social media. The comet landing happened on 12th November 2014. The amount of tweets about this on the landing day hugely dominates any other day. In fact, this graph is scaled logarithmically just so it would be possible to see the difference between other days better.

3.3) Word cloud for hashtags

In [36]:

```
filteredCount = count

for hashtag in filteredCount.most_common():
    if ("cometlanding" in hashtag[0].lower()):
        del filteredCount[hashtag[0]]

hashtagDictionary = dict(filteredCount) #Change the type of the count from Count
er to dictionary.
#Generate the word cloud by processing the dictionary of hashtags (key = hashta
g, value = count).
wordcloud = wc(max_font_size=80).generate_from_frequencies(hashtagDictionary)
plt.figure(dpi=300)
plt.imshow(wordcloud, interpolation="bilinear") #set the settings of the word cl
oud
plt.axis("off")
plt.show() #display the word cloud
```

This word cloud contains all hashtags in the given dataset aside from variants of CometLanding.

Analysis:

Philae, 67P and Rosetta appear to be the most used hashtags in this word cloud. According to the space.com, Rosetta was a space probe built by the European Space Agency launched on 2 March 2004. Along with Philae, its lander module, Rosetta performed a detailed study of comet 67P/Churyumov–Gerasimenko (67P). Thus, we could assume that the name of the comet in these tweets is the 67P/Churyumov–Gerasimenko (67P).

url of the Space.com page : https://www.space.com/34254-rosetta-crash-lands-on-comet-mission-ends.html (https://www.space.com/34254-rosetta-crash-lands-on-comet-mission-ends.html)

Extensions

Easy 1 - Analyse applications used to send tweets

In [37]:

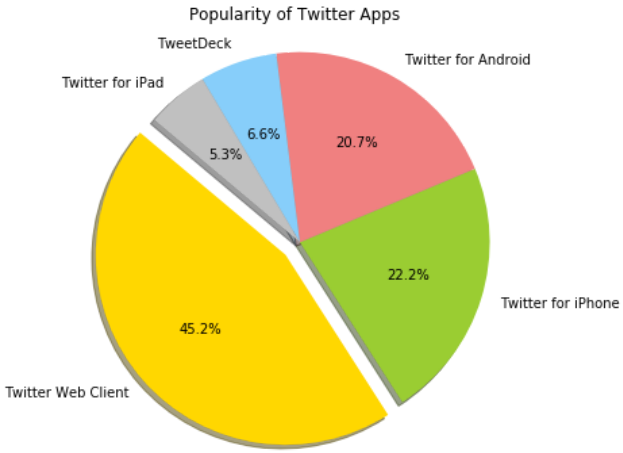
```
app_list = df.source.str.findall(r'<a[^>]*>(.*)</a>') # Find the device names us
ing a regex to extract the values between <a></a> tags #https://stackoverflow.co
m/questions/36336228/python-regex-extract-text-within-html-tags
app_list = app_list.tolist() # Convert list of individual applications to a big
 list
i = 0
# Convert the list of lists into a list
for item in app_list:
    if isinstance(item, list):
        app_list[i] = item[0]
    i+=1

# Get the total number of occurrences in the list
apps = pd.Series(app_list).value_counts().reset_index()
print("Total number of unique applications: " + str(len(apps)))
apps.columns = ["Applications", "Number"] # Set the appropriate columns

#Print nicely
for i in range(0, 10):
    print(str(i+1) + ". " + apps["Applications"][i] + " - " + str(apps["Number"]
[i]))
```

```
Total number of unique applications: 482
1. Twitter Web Client - 27925
2. Twitter for iPhone - 13743
3. Twitter for Android - 12770
4. TweetDeck - 4075
5. Twitter for iPad - 3282
6. dlvr.it - 1671
7. Twitter for Websites - 1462
8. Tweetbot for iÎŸS - 1055
9. Twitter for Windows Phone - 932
10. Tweet Old Post - 925
```

The most surprising result is that there are 482 unique applications used to send tweets in this data set. It makes sense that the most popular applications are the official ones.

In [38]:

```
popular_apps = apps.iloc[0:5] #Get 5 most popular
popular_apps.columns = ["Applications", "Number"] #Appropriate column names

colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'silver'] #Colors
 for the chart
explode = (0.1, 0, 0, 0, 0) #First section should be separated

#Make the pie chart and show percentages
plt.pie(popular_apps["Number"], explode=explode, labels=popular_apps["Applicatio
ns"], colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.title("Popularity of Twitter Apps")
plt.show()
```



Easy 2 - More descriptive analysis

Average time each user was retweeted and replied

In [39]:

```
cd = df[["text", "from_user"]] # Data frame with two columns
pd.options.mode.chained_assignment = None
cd.drop_duplicates(["text"], inplace = True) #Duplicate tweets don't count
cd.drop_duplicates(["from_user"], inplace = True) #Count the number of users
cd = cd["text"][np.logical_not (cd["text"].str.startswith('RT', na=False))] #Do
n't count retweets of retweets, since it should be attributed to the original us
er
num_unique_users = len(cd);
print("Number of users who made a unique tweet:", str(num_unique_users))
```

```
Number of users who made a unique tweet: 11603
```

In [40]:

```
numRTPerUser = numRT / num_unique_users
print("Avg. number of times each user was retweeted:", str(numRTPerUser))
```

```
Avg. number of times each user was retweeted: 5.170904076531931
```

In [41]:

```
numUserReplies = len(df[df['in_reply_to_user_id_str'].notnull()])
print("Number of times a user was replied: ", numUserReplies)
```

```
Number of times a user was replied:  1723
```

In [42]:

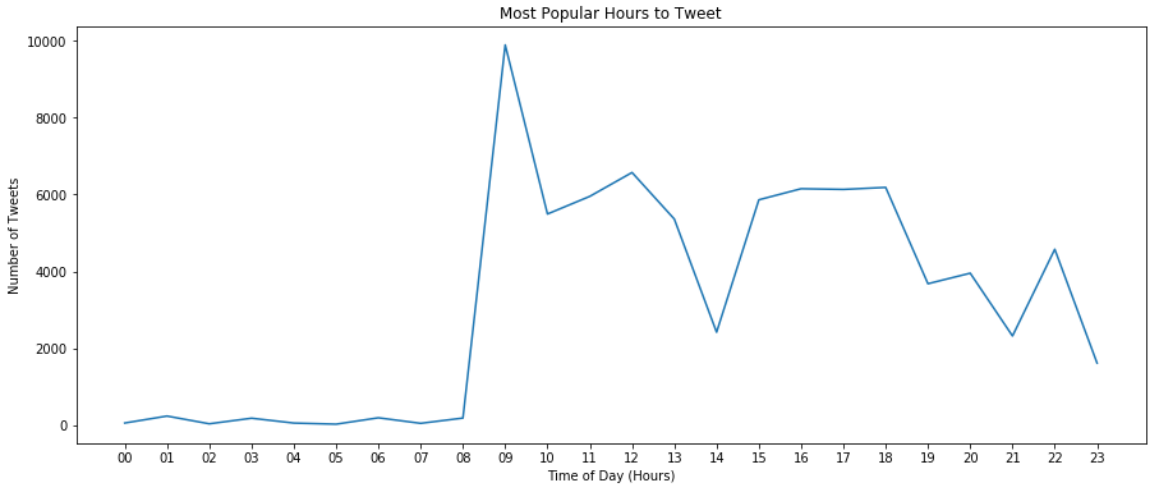
```
numRepliesPerUser = numUserReplies / numUsers
print("Avg. number of times each user was replied:", str(numRepliesPerUser))
```

```
Avg. number of times each user was replied: 0.03432612810040841
```

Popular hours

In [43]:

```
df["hour_time"] = [str(x) for x in df["time"]] # Convert the time column into st
rings
df["hour_time"] = [x.split(" ")[1].split(":")[0] for x in df["hour_time"]] # Get
 just the hour portion of the time
df["hour_time"] = pd.Series(item for item in df["hour_time"]) # Make it into a s
eries

a = Counter(df["hour_time"]) # Convert the series into a counter that shows how
 many times each hour occurs
hours = [keys for keys in a] # Make a list of hours
values = [a[str(hour)] for hour in hours] # Make a list of occurrance values
pt = {"hours":hours, "values":values} # Make a dictionary
pt = pd.DataFrame(pt) # Convert the dictionary into a dataframe
pt = pt.sort_values(by="hours") # Sort by hours from 00 to 23
pt = pt.iloc[0:24] # Get the first 24 instances, avoids nan value
plt.ylabel("Number of Tweets")
plt.xlabel("Time of Day (Hours)")
plt.title('Most Popular Hours to Tweet')
plt.plot(pt["hours"], pt["values"])
plt.show()
```



Even though it contains information about all days, it can be deducted when the important landing activities happened on November 12th since that day contains the majority of the tweets in the data set. It is interesting to see how fast paced twitter is. There was a huge surge of tweets between 9-10 AM GMT and it died off pretty quickly. Interestingly enough, the landing happened at around 4 PM GMT[1], but seems like some major announcements were done before that.

There is an interesting drop at around 2 PM GMT. Perhaps it is an unpopular time to tweet in the day. Likely, though, the time was a bit of a dead zone after the big announcement of the upcoming landing and a period of wait between the landing actually happened.

[1] - https://www.theguardian.com/science/2014/nov/12/why-is-the-rosetta-comet-landing-so-exciting (https://www.theguardian.com/science/2014/nov/12/why-is-the-rosetta-comet-landing-so-exciting)

Medium - Analyse patterns of user activity over the period covered by the dataset

Text analysis:

-Tweets/Retweets/Replies: From the previous data on tweets, retweets, and replies it is clear that a minority of users during this period produced actual tweets (and replies) where as the majority of users merely shared their posts in the form of retweets, perhaps indicating that influential users (such as those involved in the landing) were posting highly sharable content about the event.

-Timeline of Tweets: The timeline of tweets graph also shows that there was a spike in the number of tweets around the beginning of the time period covered by the data set and that the number of tweets continued to fall as time progressed. This correlates with the date of the Philae landing (12/11/2014) at the beginning of the data set as well as the most popular hashtag '#CometLanding'. Presumably the slow decline in tweets during the period of this data set correlates with a fading interest in the event as time went on after the landing.

-Popularity of Twitter Apps: As demonstrated by the pie chart plotting the popularity of different twitter platforms, the Twitter web client came out far ahead of any other platform with a 45.2% share. This makes sense as the Twitter Web Client is viewable on all major desktop operating systems such as Windows, Mac OS, and Linux via various web browsers. Perhaps somewhat noteworthy is the fact that Twitter for iOS platforms (Twitter for iPhone and Twitter for iPad) make up the second largest share with a combined share of 27.5%. This may at first come as a surprise due to Android having a much larger worldwide share of the operating system market with 74.24% compared to the iOS share of 20.83% (March 2018)[1]. However when this data is adjusted by region (for the United States in this case), iOS has a 56.26% market share compared to Android's 43.24% (March 2018)[2]. As 72.3% of Twitter's users are based in United States (April 2018)[3] it explains the discrepency between the worldwide market share of different mobile platforms and those used by Twitter's users.

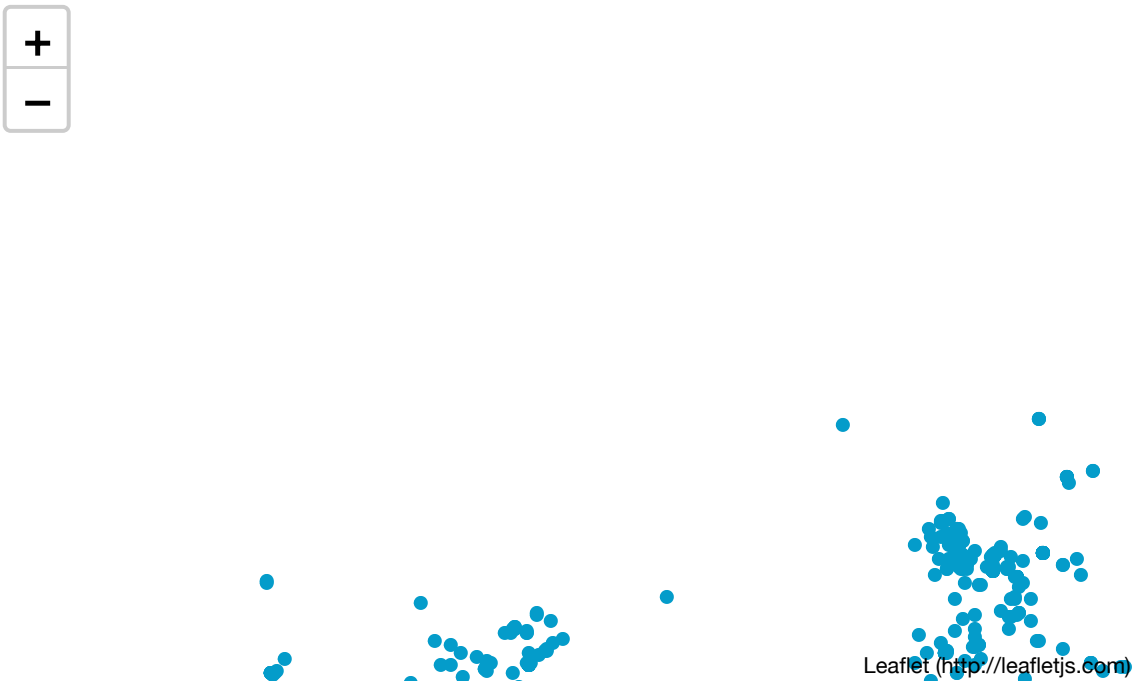
Sources:

[1] - http://gs.statcounter.com/os-market-share/mobile/worldwide (http://gs.statcounter.com/os-market-share/mobile/worldwide) - world mobile OS market share. [2] - http://gs.statcounter.com/os-market-share/mobile/united-states-of-america (http://gs.statcounter.com/os-market-share/mobile/united-states-of-america) - US mobile OS market share [3] - https://www.statista.com/statistics/242606/number-of-active-twitter-users-in-selected-countries/ (https://www.statista.com/statistics/242606/number-of-active-twitter-users-in-selected-countries/) - active twitter users by country.

In [44]:

```
#Filters out NULL geo coordinates
coordinate_df = df[(df.geo_coordinates.isnull() == False)]
#Coordinates that are 0,0 are filtered out.
coordinate_df = coordinate_df[(coordinate_df.geo_coordinates == "loc: 0,0") == F
alse]
popups = []

#The user name values and tweet time values associated with the locations are ex
tracted
for (x, y) in zip(coordinate_df['from_user'], coordinate_df['time']):
    popups.append(x + " - " + y)

#The coordinates are converted to a list of tuples (and "loc:" is removed)
coordinates = []
coordinates = coordinate_df.geo_coordinates.tolist() #Assigns coordinates to lis
t
coordinates = [x.replace('loc: ', '') for x in coordinates] #Eliminates 'loc:' t
ag
coordinates = [tuple(float(y) for y in x.split(',')) for x in coordinates] #Conv
erts string coords to tuple

#Coordinates are added to map
map = folium.Map(location=[50, 0], zoom_start=2)
for point in range(0, len(coordinates)):
    folium.CircleMarker(coordinates[point], color='#049CCA', popup=popups[point
],
                        fill_color='#049CCA', fill=True, fill_opacity=10, radius=2).
add_to(map)

map
```

Out[44]:



From the above plotmap we can observe that the majority of the users in the data set (with geo location data enabled) are concentrated in western nations such the United Kingdom, France, Germany and the United States. The time and date when each tweet was posted can also be observed if the data point is clicked on and this can be used to give an idea of the stage of the data set that the tweet pertains to.

Medium to Hard - Using networkx library to visualise the data

1) Analysis of User Connections through Replies

In [45]:

```
replyGraph = nx.Graph() # generate the initial graph instance

# Add all replies as edges and users as nodes
replyGraph = nx.from_pandas_edgelist(dfReplies, source='from_user', target='in_r
eply_to_screen_name')
```

In [46]:

```
searchDepth = 3 # set the default depth of the search as 3
```

The infomation of the reply graph is displayed below:

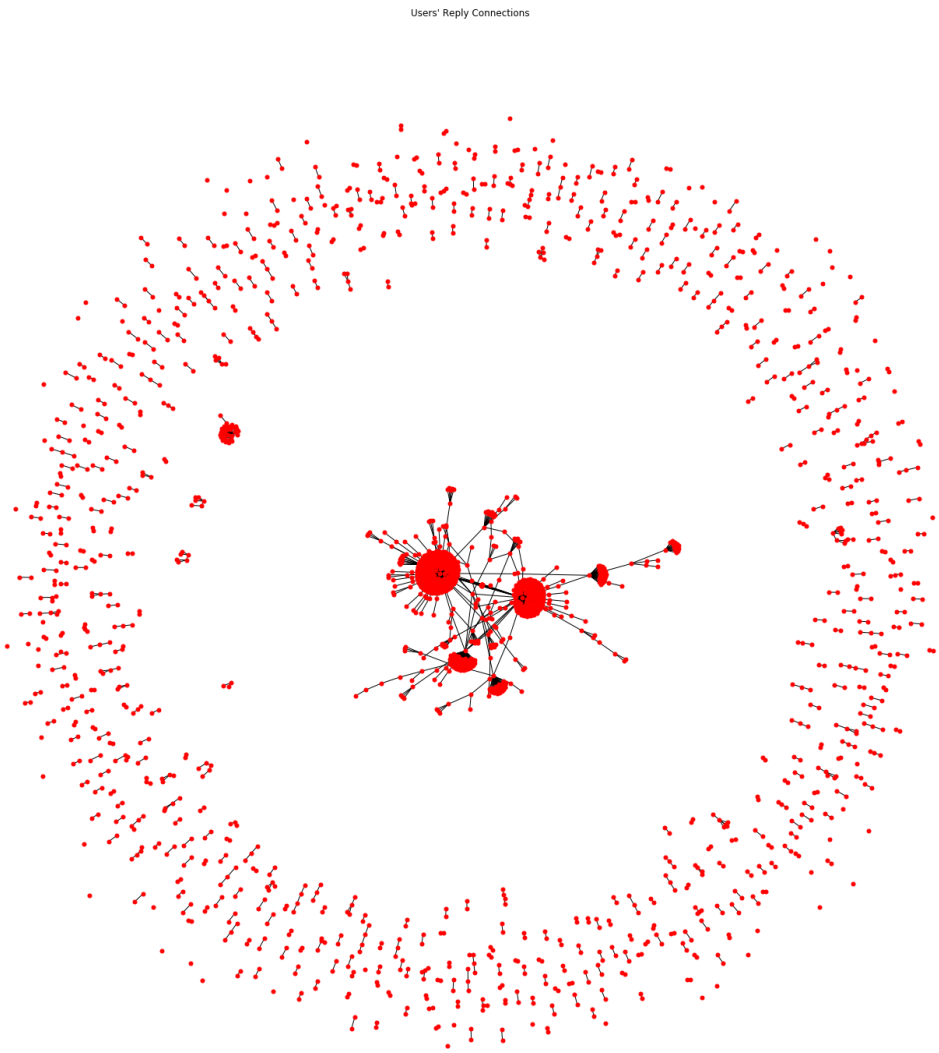
In [47]:

```
print(nx.info(replyGraph))
```

```
Name:
Type: Graph
Number of nodes: 2121
Number of edges: 1633
Average degree:   1.5398
```

In [48]:

```
# Draw Graph
pos = nx.spring_layout(replyGraph)
plt.figure(figsize=(20, 20))
nx.draw(replyGraph, pos,node_size=25, node_color='r')
plt.title("Users' Reply Connections")
plt.show()
```



The graph above is a reply graph. You could see that there is an isolated cluster on the center of the graph. It's because the tweets corresponding to those nodes in the center of the graph are all connected. There are some special users, that replied to many users, or vice versa. Similarly, the nodes, which are only connected with a few nodes, are located at the border.

To know why popular users correspond to the nodes in the big cluster on the center of the graph, I looked up the csv file and read all data in the 'in_reply_to_screen_name" column. I think it is because that the most of the people who replied to the "ESA_Rosetta" also replied to the "Philae2014". The other popular users in the center cluster are researchers, who participated in the 67P project.

Thus, it is possible to say that this graph is helpful as it helps us to filter the users who did not reply to official users ("ESA_Rosetta", "Philae2014", and researchers).

2) Analysis of User Connections through Retweets

In [49]:

```
retweetGraph = nx.Graph()
searchDepth = 2
```

In [50]:

```
# Make the new column called 'RT_name' in the dfRT.
dfRT['RT_name'] = dfRT['text'].str.replace("@","")
dfRT['RT_name'] = dfRT['RT_name'].str.split(" ").str[1]
dfRT['RT_name'] = dfRT['RT_name'].str.replace(":","")
```

In [51]:

```
retweetGraph = nx.from_pandas_edgelist(dfRT, source='from_user', target='RT_nam
e')
```

In [52]:

```
print(str(retweetGraph.number_of_nodes()))
print(str(retweetGraph.number_of_edges()))
```

```
44522
53357
```

As you could see, the number of nodes and edges of the retweet graph are too large. Because of the large number of nodes and edges, it takes more than an hour to process this graph, which is not good. Thus, we will generate several retweet graphs, which are grouped by the user language.

In [53]:

```
dfRT_UK = dfRT[(dfRT['user_lang'] == 'en-gb')] #UK English
dfRT_GB = dfRT[(dfRT['user_lang'] == 'en-GB')] #Another UK English

ukRTs = [dfRT_UK, dfRT_GB]
dfRT_UK = pd.concat(ukRTs) #concate dfRT_UK and dfRT_GB

dfRT_FR = dfRT[(dfRT['user_lang'] == 'fr')] #French
dfRT_GE = dfRT[(dfRT['user_lang'] == 'de')] #German
```

In [54]:

```
retweetGraph_UK = nx.from_pandas_edgelist(dfRT_UK, source='from_user', target='R
T_name')
```

In [55]:

```
retweetGraph_FR = nx.from_pandas_edgelist(dfRT_FR, source='from_user', target='R
T_name')
```

In [56]:

```
retweetGraph_GE = nx.from_pandas_edgelist(dfRT_GE, source='from_user', target='R
T_name')
```

I made 3 graphs for English (GB), French, and German. It is because that most tweets are uploaded by people who live in the Europe continent. As you could see on the map in the medium extension section, UK, France, and Germany are the top 3 countries that uploaded most tweets about the comet landing, among all countries in the Europe continent. Thus, we made 3 retweet graphs, which are: retweetGraph_UK (United Kingdom), retweetGraph_FR (France), and retweetGraph_GE (Germany).

Retweet graph for all users who use the language code 'en-gb' or 'en-GB' in the tweet (English - UK)

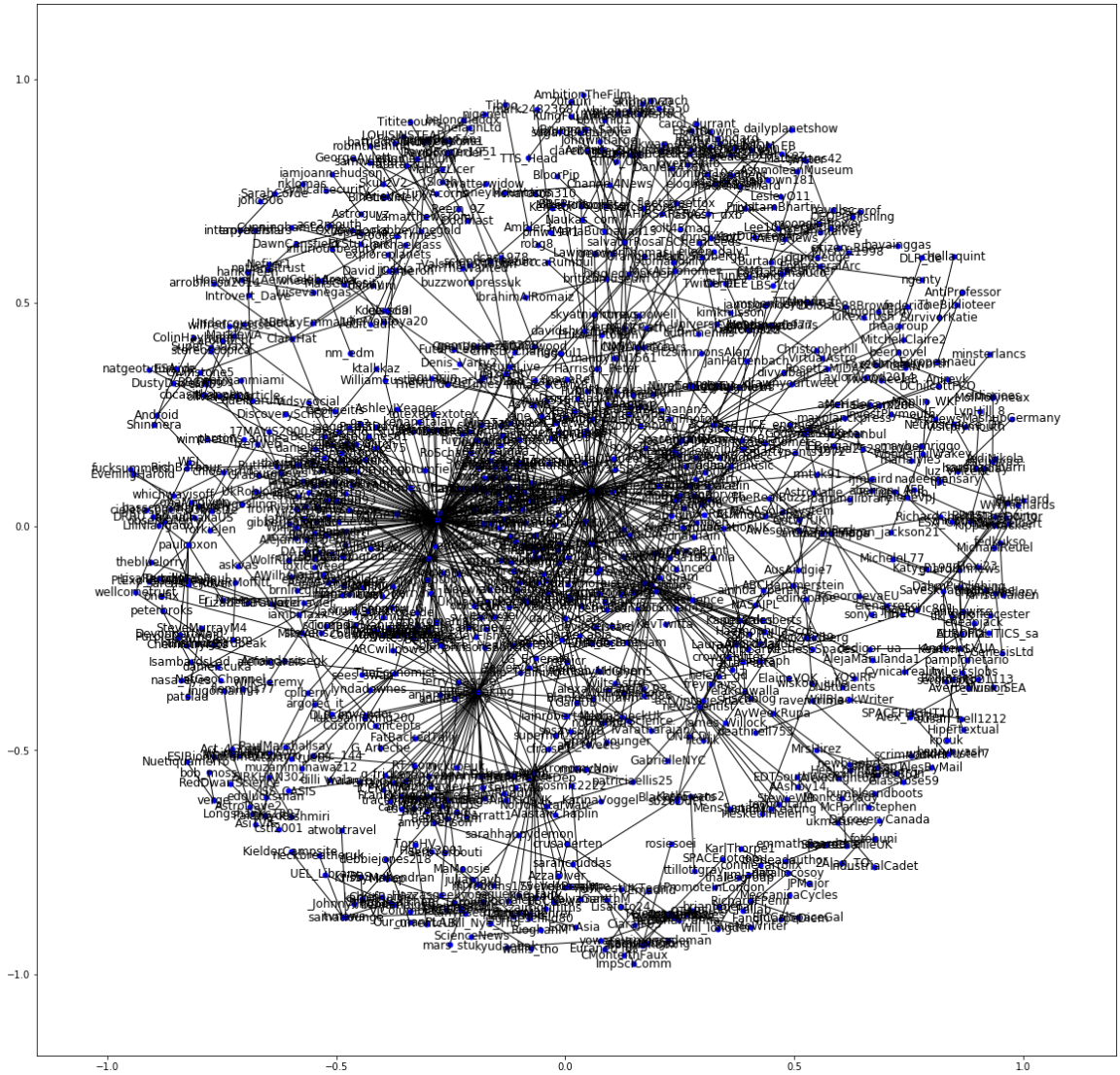
In [57]:

```
print(nx.info(retweetGraph_UK))
```

```
Name:
Type: Graph
Number of nodes: 984
Number of edges: 908
Average degree:   1.8455
```

In [58]:

```
pos = nx.spring_layout(retweetGraph_UK, k=0.15)
plt.figure(figsize=(20, 20))
nx.draw_networkx(retweetGraph_UK, pos,node_size=25, node_color='b')
plt.show()
```



If you see the graph above, there are several dense areas in the graph. If you see the graph carefully, there is a blue node on the middle of the each dense area, which is corresponding to the users that many other users retweeted.

Retweet graph for all users who use the language code 'fr' in the tweet (French)

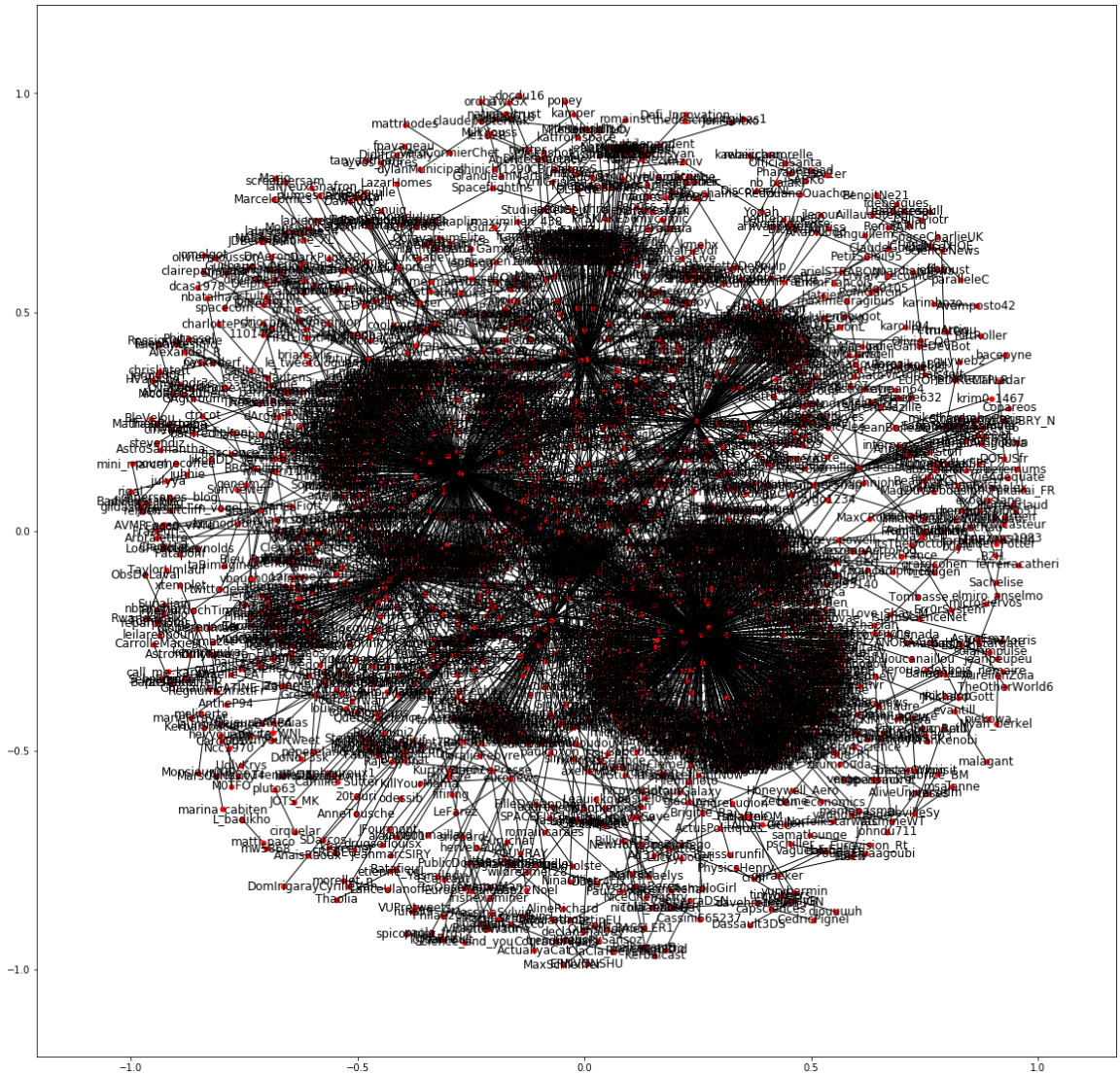
In [59]:

```
print(nx.info(retweetGraph_FR))
```

```
Name:
Type: Graph
Number of nodes: 2303
Number of edges: 2434
Average degree:   2.1138
```

In [60]:

```
pos = nx.spring_layout(retweetGraph_FR, k=0.1) #k is the optimal distance betwee
n nodes.
plt.figure(figsize=(20, 20))
nx.draw_networkx(retweetGraph_FR, pos,node_size=25, node_color='r')
plt.show()
```



If you see the section 2 (Perform the Descriptive Analysis of the Dataset), the average number of retweets sent by user is 1.195, which is less than average degree of this graph. Thus, it is possible to say that The users included in this graph raised the average number of retweets.

Retweet graph for all users who use the language code 'de' in the tweet (German)

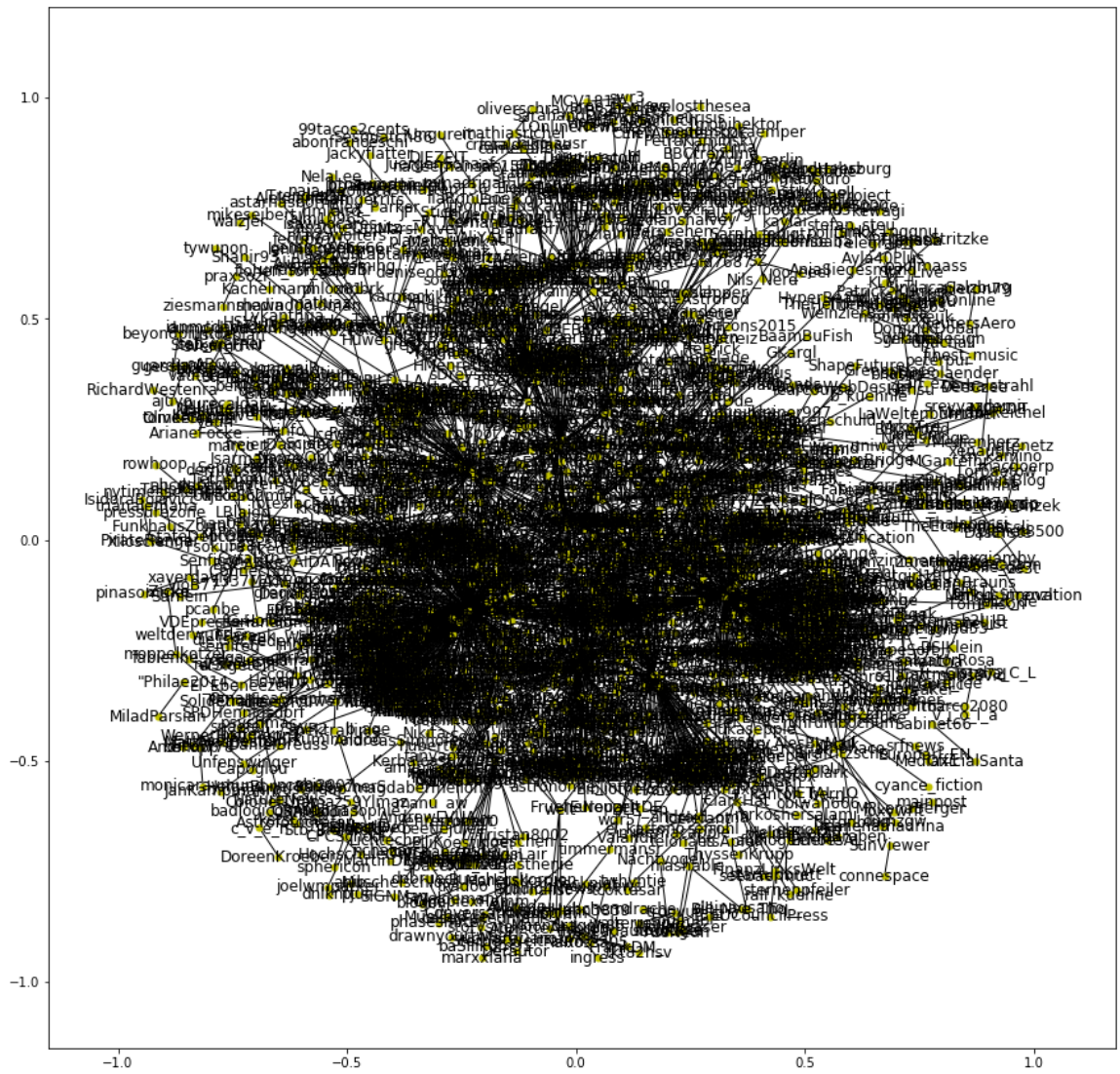
In [61]:

```
print(nx.info(retweetGraph_GE))
```

```
Name:
Type: Graph
Number of nodes: 1726
Number of edges: 1895
Average degree:   2.1958
```

In [62]:

```
pos = nx.spring_layout(retweetGraph_GE, k=0.1) #k is the optimal distance betwee
n nodes.
plt.figure(figsize=(15, 15))
nx.draw_networkx(retweetGraph_GE,pos,node_size=25, node_color='y')
plt.show()
```



Similarly, the average degree of this graph is 2.1958, which is bigger than the 1.195 (the average number of retweets seent by user).

After plotting all graphs, we found some interesting thing that the number of the nodes and edges in the retweetGraph_UK is much less than we expected.

Here are 2 possible scenarios:

1. Some british users' langauage code is 'en' rather than 'en-gb' or 'en-GB'.
2. Some people who use other language (not English) stay in the UK, and uploaded tweets. (International students, Migrants, etc)

To ensure the assumption that we made, I counted the number of users for all user languages.

In [63]:

```
#Gets different languages from the data
languages = df.groupby('user_lang')
```

In [64]:

```
languages.size() # Gets the user language codes, and the number of users for all
 languages.
```

Out[64]:

```
user_lang
ar            428
bg              1
ca            309
cs             42
da             89
de           2916
el             29
en          52316
en-AU           1
en-GB          23
en-gb        1972
es           7540
es-MX           2
eu             62
fa              2
fi            108
fil            10
fr           3313
gl             36
he              2
hi              2
hu             41
id             66
it           2664
ja           1514
ko             98
msa             1
nb              1
nl            838
no             36
pl            157
pt            508
pt-PT           1
ro              8
ru            794
sv            126
th             57
tr            761
uk             43
ur              1
vi              1
xx-lc          24
zh-CN           6
zh-Hans         6
zh-cn         285
zh-tw          27
dtype: int64
```

As you could see above, the number of users, who use language code 'en' in twitter, is 52316, which is the biggest number. Thus, we could assume that most British users are included in those 52316 users.

3) Analysis of User Connections through Mentions

In [65]:

```
# Build the Relationship graph
mentionGraph = nx.Graph()
searchDepth = 1
```

In [66]:

```
# Change the given string to the JSON. Use this function in the getUserMentions
 function.
def formatJSON(string):
    dataform = str(string).strip("'<>() ").replace('\'', '\"')
    return dataform
```

In [67]:

```
# To get the user mentions from the given tweet.
def getUserMentions (tweet):
    entities_str = formatJSON(str(tweet['entities_str']))
    try:
        entities = json.loads(entities_str)
        return entities['user_mentions']
    except ValueError:
        return []
```

In [68]:

```
#Add all mention connections as edges and users as nodes
for index, tweet in df.iterrows():
    userMentions = getUserMentions(tweet)
    for userMention in userMentions:
        mentionGraph.add_edge(tweet['from_user'], userMention['screen_name'])
```

In [69]:

```
print(nx.info(mentionGraph))
```

```
Name:
Type: Graph
Number of nodes: 48614
Number of edges: 96573
Average degree:   3.9731
```

As you could see above, the number of nodes and edges in the mention graph is too large, which will take more than an hour to process it.

Thus, we will do the same thing that we did with the retweet graph (group by the user language).

In [70]:

```
mentionGraph_UK = nx.Graph()
mentionGraph_FR = nx.Graph()
mentionGraph_GE = nx.Graph()

#Make the 3 mention graphs that are group by the user language
for index, tweet in df.iterrows():
    userMentions = getUserMentions(tweet)
    for userMention in userMentions:
        if tweet['user_lang'] == 'en-gb': #check if the language code of this tw
eet is 'en-gb'
            mentionGraph_UK.add_edge(tweet['from_user'], userMention['screen_nam
e'])
        elif tweet['user_lang'] == 'en-GB': #check if the language code of this
 tweet is 'en-GB'
            mentionGraph_UK.add_edge(tweet['from_user'], userMention['screen_nam
e'])
        elif tweet['user_lang'] == 'fr': #check if the language code of this twe
et is 'fr'
            mentionGraph_FR.add_edge(tweet['from_user'], userMention['screen_nam
e'])
        elif tweet['user_lang'] == 'de': #check if the language code of this twe
et is 'de'
            mentionGraph_GE.add_edge(tweet['from_user'], userMention['screen_nam
e'])
```

In [71]:

```
print(nx.info(mentionGraph_UK)) #print out the informaion about the mentionGraph
_UK.
```

```
Name:
Type: Graph
Number of nodes: 1167
Number of edges: 1674
Average degree:   2.8689
```

In [72]:

```
print(nx.info(mentionGraph_FR)) #print out the informaion about the mentionGraph
_FR.
```

```
Name:
Type: Graph
Number of nodes: 2534
Number of edges: 4396
Average degree:   3.4696
```

In [73]:

```
print(nx.info(mentionGraph_GE)) #print out the informaion about the mentionGraph
_GE.
```

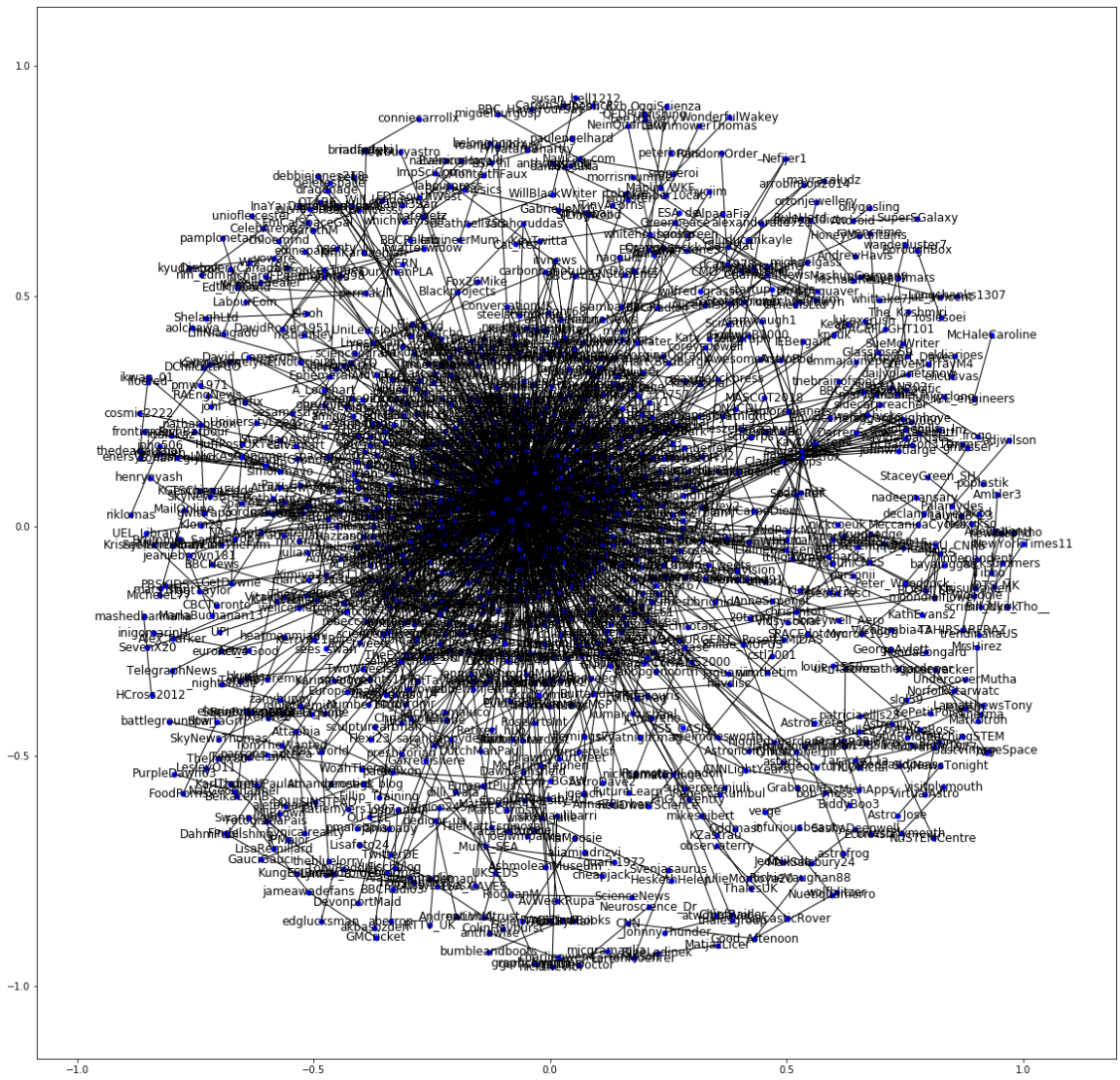
```
Name:
Type: Graph
Number of nodes: 2003
Number of edges: 3391
Average degree:   3.3859
```

Now the number of nodes and edges looks much better than before.

Generate the mention graph for language code 'en-gb' and 'en-GB' (English - UK)

In [74]:

```
pos = nx.spring_layout(mentionGraph_UK, k=0.15) #k is the optimal distance betwe
en nodes.
plt.figure(figsize=(20, 20))
nx.draw_networkx(mentionGraph_UK, pos,node_size=25, node_color='b')
plt.show()
```

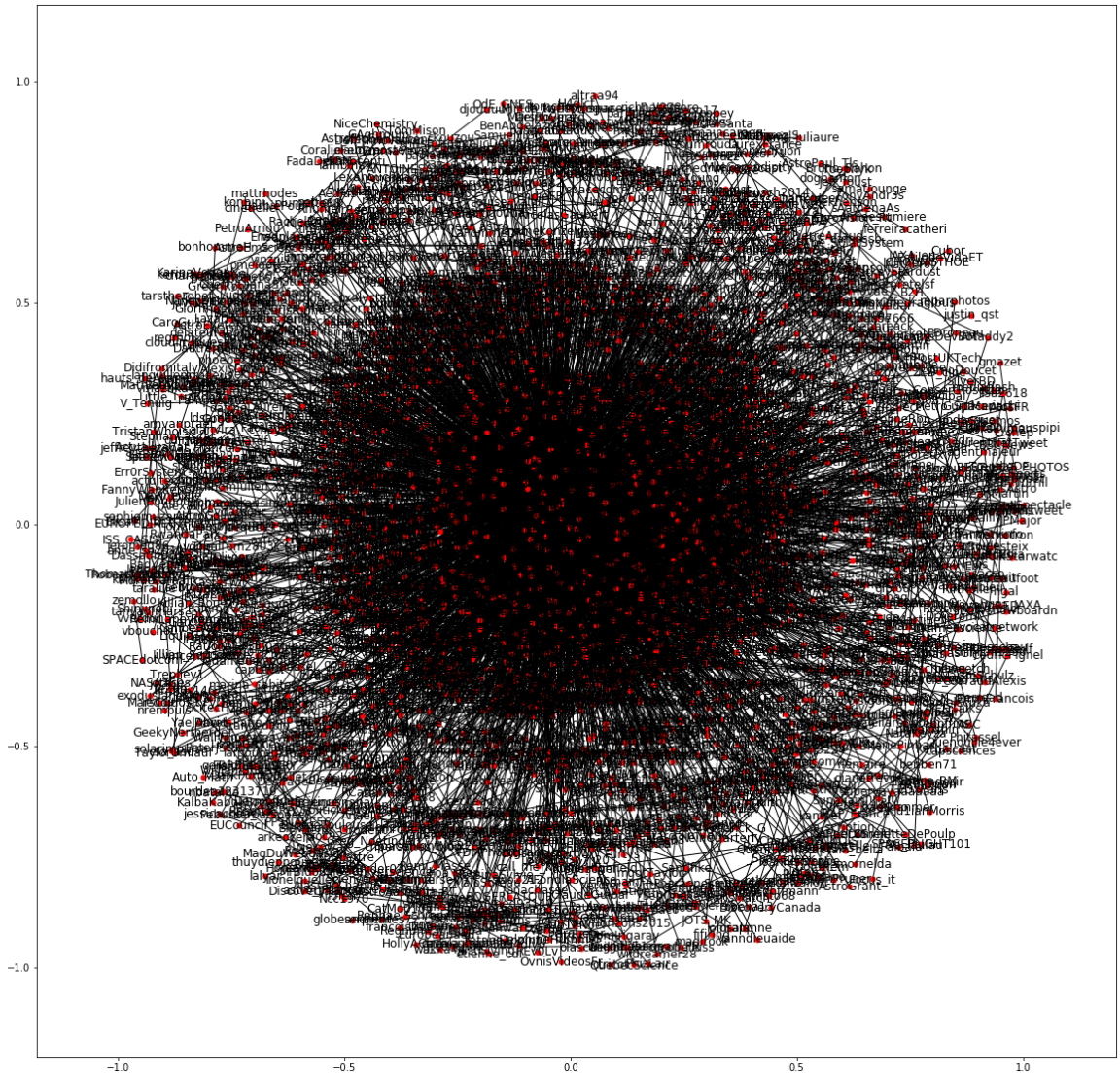


The graph above has a quite big dense area, which means that there are some popular users that are mentioned by the a number of users, and vice versa.

Generate the mention graph for language code 'fr' (French)

In [75]:

```
pos = nx.spring_layout(mentionGraph_FR, k=0.2) #k is the optimal distance betwee
n nodes.
plt.figure(figsize=(20, 20))
nx.draw_networkx(mentionGraph_FR, pos,node_size=25, node_color='r')
plt.show()
```

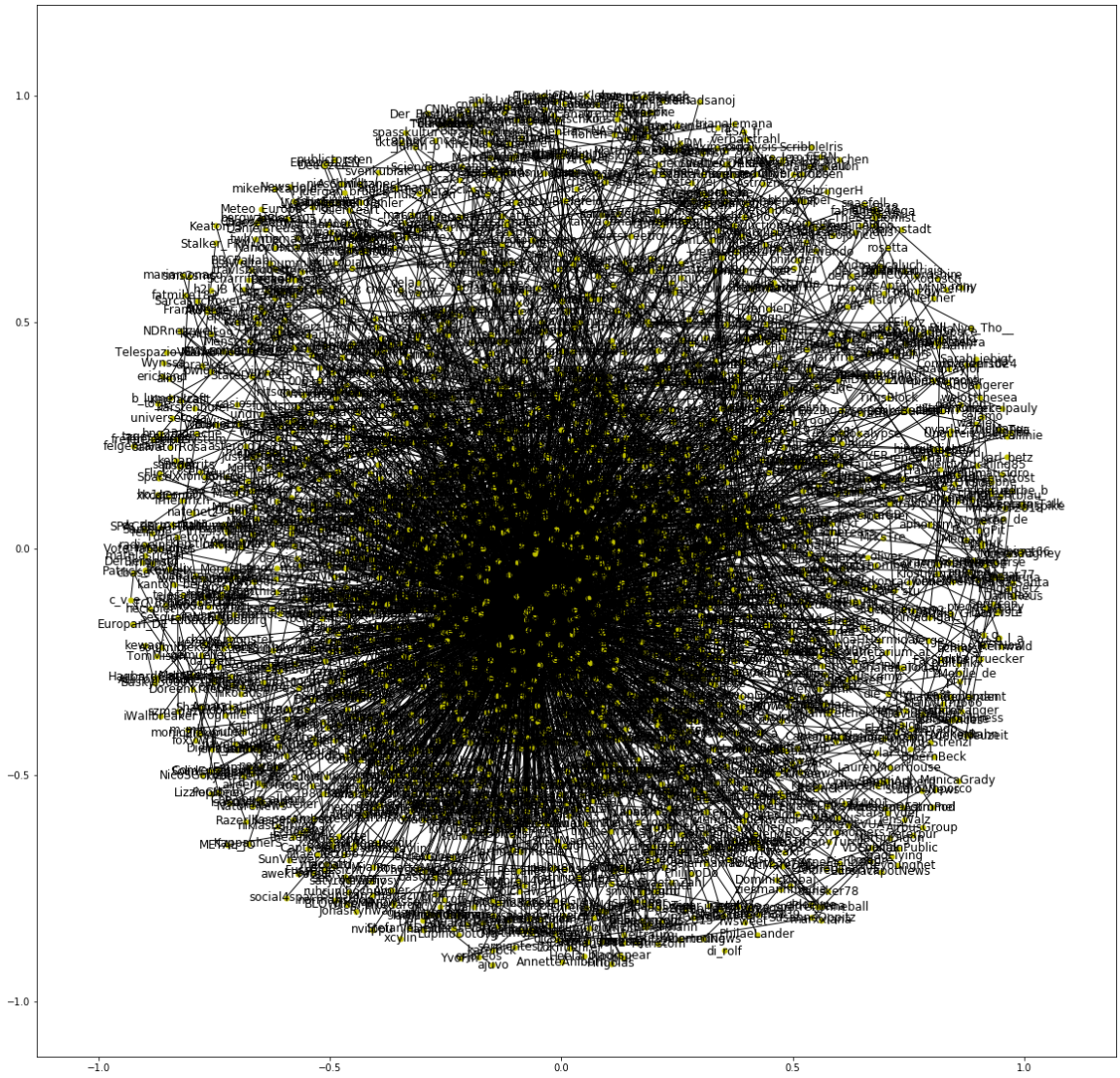


The mentionGraph_FR graph has a large dense area than the mentionGraph_UK. I assume that it is because there are less users and mentions in this graph than the mentionGraph_FR graph.

Generate the mention graph for language code 'de' (German)

In [76]:

```
pos = nx.spring_layout(mentionGraph_GE, k=0.2) #k is the optimal distance betwee
n nodes.
plt.figure(figsize=(20, 20))
nx.draw_networkx(mentionGraph_GE, pos,node_size=25, node_color='y')
plt.show()
```



The mentionGraph_GE graph has a smaller dense area than the mentionGraph_FR. I assume that it is because there are less number of users and mentions in this graph.

Hard - Interactive visualisations

An interactive pie chart that shows the top n most popular twitter apps for the data set.

In [77]:

```
# Function for updating the plot
def f(n=5):
    popular_apps = apps.iloc[0:n] #Get n most popular
    popular_apps.columns = ["Applications", "Number"] # Label the columns approp
riately

    # Make a pie chart out of the data
    plt.pie(popular_apps["Number"], labels=popular_apps["Applications"],
            autopct='%1.1f%%', shadow=True, startangle=140)

    plt.axis('equal')
    plt.title("Popularity of Twitter Apps")
    plt.show()

# call the function f every time the user interacts with the graph
interact(f,n=(1,15))
```

Out[77]:

```
<function __main__.f>
```

This shows us that merely 3 apps dominate most of the market. People use a huge variety of apps, though they are not very popular.

An interactive graph to show the amount of retweets, replies and tweets based on weekday.

In [78]:

```
tweet_types = ["Retweets", "Replies", "Tweets"] # Lists for 3 different types of
 tweets
tweet_amount = [None] * 3 # List to contain the amount of tweets for the differe
nt types
# Function for updating the plot
def f(rt=True, day="Monday"):
    plt.title("Tweets on " + day) # Update the title based on what the user chos
e
    plt.ylabel('Amount')
    day = day[0:3] # Get the first three letters of the day, that is how the day
s are used in the data
    tweet_amount[0] = len(dfRT[dfRT["created_at"].str.startswith(day, na=False
)]) # Get the number of retweets on a day
    tweet_amount[1] = len(dfReplies[dfReplies["created_at"].str.startswith(day,
na=False)]) # Get the number of replies on a day
    tweet_amount[2] = len(tweets[tweets["created_at"].str.startswith(day, na=Fal
se)]) # Get the number of tweets on a day

    # Since retweets dominate the data, the user may wish to hide the retweets c
olumn
    if (not rt):
        pos = np.arange(len(tweet_types[1:3]))
        plt.bar(pos, tweet_amount[1:3], align="center")
        plt.xticks(pos, tweet_types)
    else:
        pos = np.arange(len(tweet_types))
        plt.bar(pos, tweet_amount, align="center")
        plt.xticks(pos, tweet_types)

    plt.show()

# Call the function f and plot the graph according to the variables the user cho
se by interacting with the graph
interact(f,day=["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturda
y", "Sunday"],
         rt=widgets.Checkbox(description="Show retweets", value=True))
```

Out[78]:

```
<function __main__.f>
```

We can safely assume that the comet landing happened on a Wednesday. It seems that on Wednesday the difference between retweets and tweets is the smallest. On other days it is even bigger. Seems that the landing encouraged unique tweets a bit more. People are more likely to share "old news" by retweeting rather than by giving their unique thoughts.