Programming in Python for Data Science Coursework

- **Module Code:** CS3PP19
- **Assignment report Title:** CS3PP19 - Coursework
- **Student Numbers:** 27001654 27020363
- **Date:** 28/11/2021
- **Actual hrs spent for the assignment:** 35
- **Assignment evaluation:**
    1. Helpful to learn how to draw data from API's
    2. Really helpful to make me try and find inovative ways for visualization
    3. The network part has a lot to do with the users we choose, so i believe it would be a good idea for the lecturers to give us some options instead of us choosing which ever we want

Things to note

- For the second task, we used a library named **WordCloud**. We have spoken with Mr Miguel Sanchez Razo, and he allowed us to do this.
- For the Task 2 and Task 3, we used data that we saved when we first implemented Task 1. However we ran again Task 1 before sumbitting so there might be some differences between the data from Task 1 and the other two

Task 1 - Data Gathering and Pre-processing

API Connection

For the API Connection we will need to connect to the twitter API and using tweepy draw basic data for each user and their 300 latest tweets.

Importing libraries and setting keys

In [1]:
```
import tweepy
API_KEY = ''
API_SECRET_KEY = ''
ACCESS_TOKEN = ''
ACCESS_TOKEN_SECRET = ''
```

Establishing connection with Twitter

In [2]:
```
auth = tweepy.OAuthHandler(API_KEY, API_SECRET_KEY)
auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
api = tweepy.API(auth)
```

Getting the users data

In [3]:
```
# Because the API can only give us the first 200 and we need 300 we need to b
# will return 200 and one that will return 100 bellow the last one
# We use 101 as the count because the max_id we be included which afterwards
# Lastly we combine the 200 and the 100 timelines to get 300 tweets
# We could have used cursors but everyone has done that so lets do something
billGatesUser = api.get_user(screen_name='BillGates')
billGatesTimeline200 = api.user_timeline(screen_name='BillGates', count = 200
billGatesTimeline100 = api.user_timeline(screen_name='BillGates', count = 101
billGatesTimeline = billGatesTimeline200 + billGatesTimeline100

elonMuskUser = api.get_user(screen_name='elonmusk')
elonMuskTimeline200 = api.user_timeline(screen_name='elonmusk', count = 200,t
elonMuskTimeline100 = api.user_timeline(screen_name='elonmusk', count = 101,t
elonMuskTimeline = elonMuskTimeline200 + elonMuskTimeline100


ellenDeGeneresUser = api.get_user(screen_name='TheEllenShow')
ellenDeGeneresTimeline200 = api.user_timeline(screen_name='TheEllenShow', cou
ellenDeGeneresTimeline100 = api.user_timeline(screen_name='TheEllenShow', cou
ellenDeGeneresTimeline = ellenDeGeneresTimeline200 + ellenDeGeneresTimeline10
```

Data Pre-processing

For the data pre-processing, we aim to process the data that we draw from Twitter and parse them into data frames. The goal is to create one data frame with basic data from all the users together and three data frames with the timeline data of its user. The reason for that is that with the single data frame with the basic data, we can analyze and compare the users easier. With the timeline data frames, we can do more in-depth analyses for each user separately.

Necessary imports

In [4]:
```
import pandas as pd
import json
```

Definition of a function for the timeline processing

In [5]:
```
def setUpTimelineList(timeline):
    listOfTweets = []
    for each_json_tweet in timeline:
        listOfTweets.append(each_json_tweet._json)
    return listOfTweets
```

Users pre-processing

In [6]:
```
# Bill Gates
billGatesUserFrame = pd.json_normalize(billGatesUser._json)
billGatesTimelineFrame = pd.DataFrame(setUpTimelineList(billGatesTimeline))

#Elon Musk
elonMuskUserFrame = pd.json_normalize(elonMuskUser._json)
elonMuskTimelineFrame = pd.DataFrame(setUpTimelineList(elonMuskTimeline))

#Ellen DeGenerous
ellenDeGeneresUserFrame = pd.json_normalize(ellenDeGeneresUser._json)
ellenDeGeneresTimelineFrame = pd.DataFrame(setUpTimelineList(ellenDeGeneresTi

#Adding all the users into one dataframe
frames = [billGatesUserFrame, elonMuskUserFrame, ellenDeGeneresUserFrame]
usersFrame = pd.concat(frames)
```

Data Cleansing

In order to get the data ready for processing, we need to do a few things first.

We are going to keep all the columns that we think will be useful. That does not necessarily mean that we will use all the columns that we keep, but after a meeting with the team members, we agreed that these particular columns have valuable data that we might use. We will create two new columns for the user-mentioned and hashtags and change the NaN's values. Finally, we are going to transform the dates into pandas format dates.

Definition a function for field selection

In [7]:
```
def cleanUpColumns(columnsToKeep, dataframe):
    for column in dataframe:
        if column not in columnsToKeep:
            dataframe = dataframe.drop([column], axis = 1)
    return dataframe
```

Removing all the unnecessary data from the usersFrame

In [8]:
```
#Just to get a bit clever we will just write the ones we want since there are
importantColumns = ['id', 'name', 'screen_name', 'description', 'followers_co
                    'favourites_count', 'statuses_count', 'status.id', 'statu
usersFrame = cleanUpColumns(importantColumns, usersFrame)
```

Removing all the unnecessary data from the users timelines frames

In [9]:
```
#Following the same process as above we write only the onces we need
importantColumns = ['created_at', 'id', 'full_text','truncated', 'entities',
                    'in_reply_to_screen_name', 'is_quote_status','retweet_cou
billGatesTimelineFrame = cleanUpColumns(importantColumns, billGatesTimelineFr
elonMuskTimelineFrame = cleanUpColumns(importantColumns, elonMuskTimelineFram
ellenDeGeneresTimelineFrame = cleanUpColumns(importantColumns, ellenDeGeneres
```

Creating columns for hashtags and users mentioned from entities

In [10]:
```
# The bellow functions are used to process the hashtags and the users mention
def processingHashtags(entity):
    if entity['hashtags']:
        hashtags = []
        for hashtag in entity['hashtags']:
            hashtags.append(hashtag['text'])
        return hashtags
    return 'N/A'

def processingUserMentioned(entity):
    if entity['user_mentions']:
        users = []
        for user in entity['user_mentions']:
            users.append(user['screen_name'])
        return users
    return 'N/A'
```

In [11]:
```
#The bellow function extracts the important data from the entities column and
def processEntities(dataframe):
    dataframe['hashtags'] = dataframe['entities'].apply(processingHashtags)
    dataframe['user_mentions'] = dataframe['entities'].apply(processingUserMe
    dataframe = dataframe.drop(['entities'], axis=1)
    return dataframe
```

In [12]:
```
billGatesTimelineFrame = processEntities(billGatesTimelineFrame)
elonMuskTimelineFrame = processEntities(elonMuskTimelineFrame)
ellenDeGeneresTimelineFrame = processEntities(ellenDeGeneresTimelineFrame)
```

Converting all the NaN's to N/A

In [13]:
```
billGatesTimelineFrame = billGatesTimelineFrame.fillna('N/A')
elonMuskTimelineFrame = elonMuskTimelineFrame.fillna('N/A')
ellenDeGeneresTimelineFrame = ellenDeGeneresTimelineFrame.fillna('N/A')
usersFrame = usersFrame.fillna('N/A')
```

Fixing the dates

In [14]:
```
billGatesTimelineFrame['created_at'] = pd.to_datetime(billGatesTimelineFrame[
elonMuskTimelineFrame['created_at'] = pd.to_datetime(elonMuskTimelineFrame['c
ellenDeGeneresTimelineFrame['created_at'] = pd.to_datetime(ellenDeGeneresTime
usersFrame['created_at'] = pd.to_datetime(usersFrame['created_at'])
```

Final Result

In [15]:
```
billGatesTimelineFrame.head()
```

Out[15]:

	created_at	id	full_text	truncated	in_reply_to_status_id	in_rep

	created_at	id	full_text	truncated	in_reply_to_status_id	in_rep
0	2021-11-27 18:17:18+00:00	1464659614448771073	There's lots of speculation about what Shakesp...	False	N/A	
1	2021-11-26 17:40:49+00:00	1464288045541199874	When I was a kid, I was obsessed with science ...	False	N/A	
2	2021-11-24 15:21:05+00:00	1463528104614236167	I read a lot of great books this year. These w...	False	N/A	
3	2021-11-23 16:44:40+00:00	1463186751619436546	I really enjoyed reading these five books this...	False	N/A	
4	2021-11-22 00:45:36+00:00	1462583003272474628	It's hard to believe that it's	False	N/A	

In [16]:
```
elonMuskTimelineFrame.head()
```

Out[16]:

	created_at	id	full_text	truncated	in_reply_to_status_id
0	2021-11-27 21:25:25+00:00	1464706955130269702	@lexfridman Nice	False	1464701614149849088.(
1	2021-11-27 08:13:51+00:00	1464507751262875651	@WholeMarsBlog The most ironic outcome is the ...	False	1464491385277411328.(
2	2021-11-27 08:12:09+00:00	1464507322114342912	@EvaFoxU @teslaownersSV Yeah	False	1464507108892762112.(
3	2021-11-27 08:09:31+00:00	1464506658520899592	@teslaownersSV Hypothetically, if they did mak...	False	1464359265594855424.(
4	2021-11-27 02:38:29+00:00	1464423352295165954	@mdad8200 @existentialcoms Exactly	False	995219057900126208.(

In [17]:
```
ellenDeGeneresTimelineFrame.head()
```

Out[17]:

	created_at	id	full_text	truncated	in_reply_to_status_id	in_re
0	2021-11-27 20:40:54+00:00	1464695750215241728	You need inspiration on what to do with your	False	N/A	

	created_at	id	full_text	truncated	in_reply_to_status_id	in_re
			T...			
1	2021-11-27 02:47:33+00:00	1464425631899394051	Hailey Bieber has broken her foot not once, bu...	False	N/A	
2	2021-11-27 00:17:44+00:00	1464387932840681480	It's #BlackFriday over at The Ellen Shop! You ...	False	N/A	
3	2021-11-26 17:27:01+00:00	1464284571508346883	Is splitting the bill on the first date a red ...	False	N/A	
4	2021-11-26 01:43:05+00:00	1464047023448104965	Happy Thanksgiving from my family to	False	N/A	

In [18]:
```
usersFrame
```

Out[18]:

	id	name	screen_name	description	followers_count	friends_count	listed_cou
0	50393960	Bill Gates	BillGates	Sharing things I'm learning through my foundat...	56227139	341	1215
0	44196397	Elon Musk	elonmusk		64922207	104	817
0	15846407	Ellen DeGeneres	TheEllenShow	Comedian, talk show host and ice road trucker....	77735203	26513	968

Leasons Learned

Through task 1, we learned how to draw data from an API and the importance of data cleansing. It seemed nearly impossible to analyze the data without dropping all the useless columns and modifying the data in the correct format.

Task 2 – Exploratory Data Analysis

In [19]:
```
billGatesTimelineFrame = pd.read_csv('billGatesTimelineFrame.csv').fillna('N/
elonMuskTimelineFrame = pd.read_csv('elonMuskTimelineFrame.csv').fillna('N/A'
ellenDeGeneresTimelineFrame = pd.read_csv('ellenDeGeneresTimelineFrame.csv').
usersFrame = pd.read_csv('usersFrame.csv').fillna('N/A')
###IMPORTANT
usersFrame = usersFrame.drop(['Unnamed: 0'], axis=1)
```

Importing libraries

In [118…
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import math
from datetime import datetime
from wordcloud import WordCloud
```

Comparison between accounts

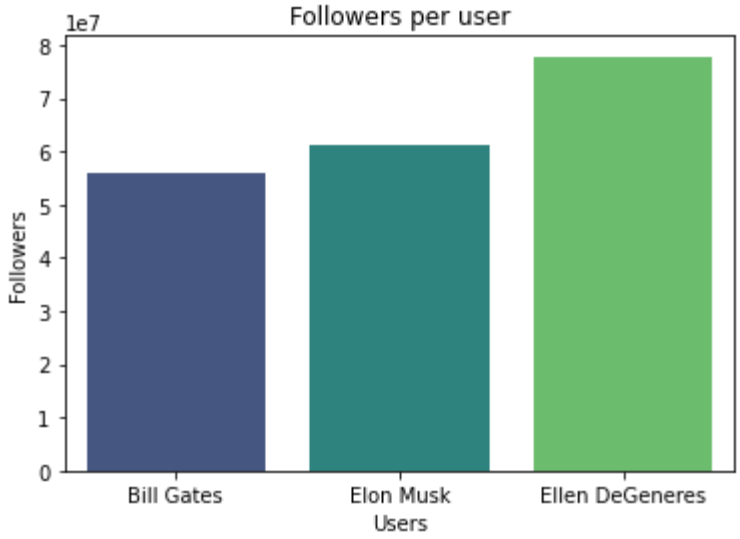
Followers and attemp to interpret the reason

In the following section we aim to see who has the most followers and try to interpert why. We will do that by investigating the tweets per user and if this has any correlation with the followers and followers per following ratio.

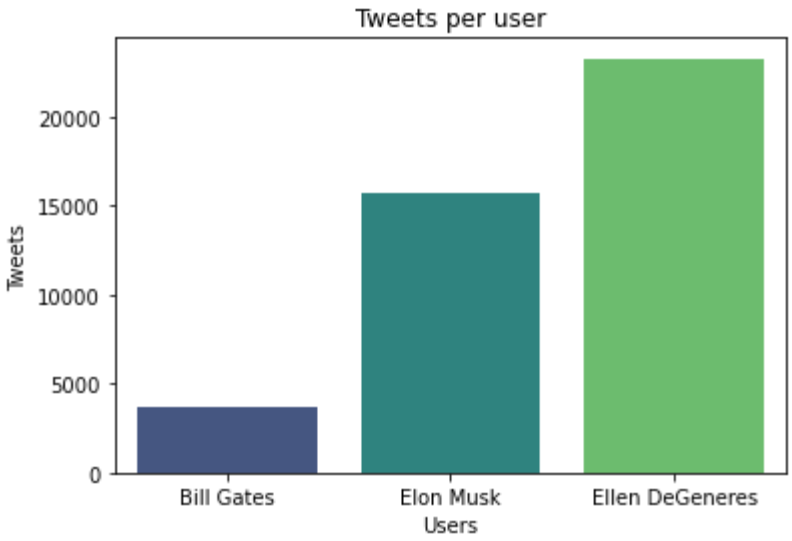
In [21]:
```
billGatesTimelineFrame = pd.read_csv('billGatesTimelineFrame.csv').fillna('N/
elonMuskTimelineFrame = pd.read_csv('elonMuskTimelineFrame.csv').fillna('N/A'
ellenDeGeneresTimelineFrame = pd.read_csv('ellenDeGeneresTimelineFrame.csv').
usersFrame = pd.read_csv('usersFrame.csv').fillna('N/A')
usersFrame = usersFrame.drop(['Unnamed: 0'], axis=1)
billGatesTimelineFrame = billGatesTimelineFrame.drop(['Unnamed: 0'], axis=1)
elonMuskTimelineFrame = elonMuskTimelineFrame.drop(['Unnamed: 0'], axis=1)
ellenDeGeneresTimelineFrame = ellenDeGeneresTimelineFrame.drop(['Unnamed: 0']

billGatesTimelineFrame = billGatesTimelineFrame.rename(columns={"text": "full
elonMuskTimelineFrame = elonMuskTimelineFrame.rename(columns={"text": "full_t
ellenDeGeneresTimelineFrame = ellenDeGeneresTimelineFrame.rename(columns={"te
```

In [22]:
```
ax = sns.barplot(x='name', y='followers_count', data=usersFrame,palette='viri
ax.set(xlabel='Users', ylabel='Followers', title='Followers per user')
plt.show()
```



In [23]:
```
ax = sns.barplot(x='name', y='statuses_count', data=usersFrame,palette='virid
ax.set(xlabel='Users', ylabel='Tweets', title='Tweets per user')
plt.show()
```



As we can see on the above bar plot, Ellen has the most followers following by Ellon, and last is Bill. We can also see that the same rank exists with the number of tweets perusers. The explanation for this could be that the more tweets a user has, the more followers the user will have. Let's see how correlated the values are.

In [24]:
```
usersFrame.corr()['followers_count']
```

Out[24]:
```
id                -0.997632
followers_count    1.000000
friends_count      0.970089
listed_count      -0.319700
favourites_count   0.130364
statuses_count     0.913517
status.id         -0.872444
Name: followers_count, dtype: float64
```

In the above table, we can see that there is a very strong correlation between the user friends and the user followers, which is to be expected. Also, as expected from the bar plots above, there is a strong correlation between the number of tweets and the number of followers. One reason for this could be that the more tweets you have, the more likely the possibility that someone will discover you.

What is surprising in the table above is that there is a minimal correlation between the user's likes and the tweeter followers. It would be expected that due to social conventions if someone likes your tweet regularly, you should follow him. One way to explain this, however, is that the user would like tweets regularly from particular accounts while the other users will like tweets from fun too, but not as regular.

It is important to note that the correlation could easily be wrong with a sample of three users.

In [25]:
```
usersFrame['followers per following'] = usersFrame['followers_count']/usersFr
usersFrame[['name', 'followers per following']]
```

Out[25]:

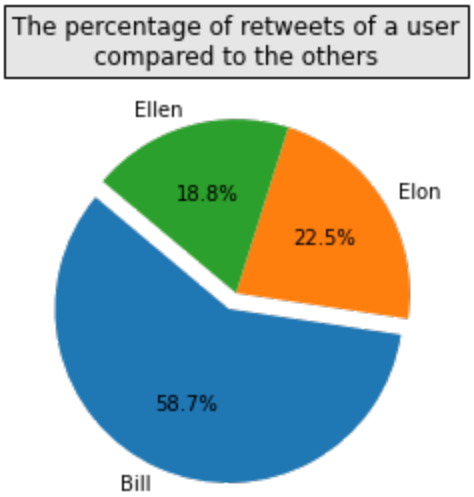
	name	followers per following
0	Bill Gates	169577.060790
1	Elon Musk	577349.037736
2	Ellen DeGeneres	2931.894537

Finally, as we can see, all of the users have huge followers per following ratio, as can be expected from influencers, with Elon having the biggest one and Ellen the smallest. This could also be the reason that Ellen has more followers, a lot of her followers are just following her back. However, we do not have enough information to prove that.

Original tweets vs retweets

In this section, we will calculate the original tweets and the retweets of the users and we will try to show who from the influencers has the most original content and who uses more retweets

In [26]:
```
# Fun fact retweets have zero likes because users cant like them. So if we ta
# It could be argued that if an original tweet has zero likes the method will
billRe = len(billGatesTimelineFrame.loc[billGatesTimelineFrame['favorite_coun
elonRe = len(elonMuskTimelineFrame.loc[elonMuskTimelineFrame['favorite_count'
ellenRe = len(ellenDeGeneresTimelineFrame.loc[ellenDeGeneresTimelineFrame['fa
# Calculate the original tweets
billOr = len(billGatesTimelineFrame) - billRe
elonOr = len(elonMuskTimelineFrame) - elonRe
ellenOr = len(ellenDeGeneresTimelineFrame) - ellenRe
data = [billRe, elonRe, ellenRe]
labels=['Bill', 'Elon', 'Ellen']
#Explode the first slice
explode = (0.1, 0, 0)
plt.pie(data, labels=labels, explode=explode,
autopct='%1.1f%%', startangle=140)
plt.title('The percentage of retweets of a user\n' +'compared to the others',
plt.show()
```



In the above pie chart we can see the percentage of the retweets owned by each user if we have add the retweets from the three users together. From all the retweets from the three users Bill has the most with 58.7%

In [27]:
```
# Function for creating percentage in a nice presentable way
def createPer(number):
    return str('{0:.3g}'.format(number/3)) + '%'
```

In [28]:
```
data = {'Bill':[createPer(billRe), createPer(billOr)], 'Elon':[createPer(elon
usersTweetsPercentages = pd.DataFrame(data, index=['Retweets Percentage', 'Or
usersTweetsPercentages
```

Out[28]:

	Bill	Elon	Ellen
Retweets Percentage	15.7%	6%	5%
Original Tweets Percentage	84.3%	94%	95%

By looking at the pie chart and also at the table, we can see that Bill has more retweets than

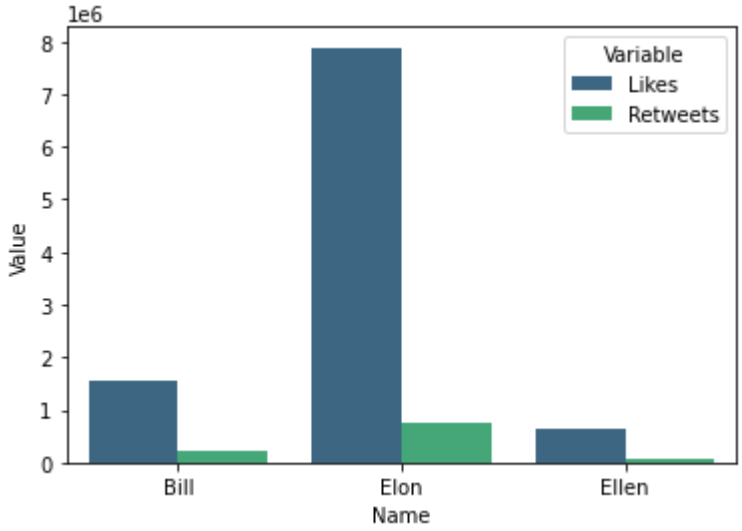
Elon and Ellen combined. However, all three users have a relatively small retweet percentage, with Bill having the higher at 15.7% of his three hundred last tweets. All the users thought mainly use original tweets.

Who has the most likes and retweets

Here we will try to find out who is the more likeable and has more influences. To do that, we will compare the likes and retweets the users had.

In [29]:
```
billLikes = billGatesTimelineFrame['favorite_count'].sum()
elonLikes = elonMuskTimelineFrame['favorite_count'].sum()
ellenLikes = ellenDeGeneresTimelineFrame['favorite_count'].sum()
billRe = billGatesTimelineFrame['retweet_count'].sum()
elonRe = elonMuskTimelineFrame['retweet_count'].sum()
ellenRe = ellenDeGeneresTimelineFrame['retweet_count'].sum()
d = {'Name':['Bill', 'Elon', 'Ellen'],
     'Likes' : [billLikes, elonLikes, ellenLikes],
     'Retweets':[billRe, elonRe, ellenRe]}
tweetFrame = pd.DataFrame(d)
tidy = tweetFrame.melt(id_vars='Name').rename(columns=str.title)
sns.barplot(x='Name', y='Value',hue='Variable', data=tidy,palette='viridis')
```

Out[29]:
```
<AxesSubplot:xlabel='Name', ylabel='Value'>
```



By looking at the above graph, we can see that Elon has way more likes than the other two. Also quite surprising is that Elen, even if she has the most followers, has fewer likes and retweets from the others.

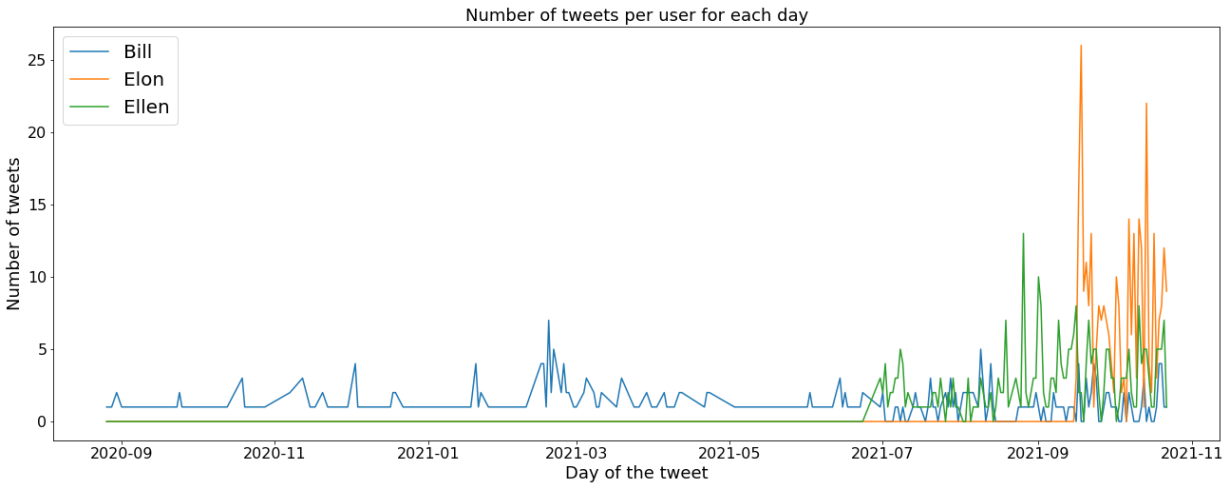
Number of tweets per user through time

In the following section we will do some time based analyses. We will try, using the volume of tweets per day, to find out any significant event that happened on our influencers lives.

In [30]:
```
# A function that will create a column that will contain the number of tweet
def createTweetsPerDay(dataFrame):
    newFrame = dataFrame[['created_at', 'full_text']].copy()
    newFrame['created_at'] = pd.to_datetime(newFrame['created_at'])#SHOULD BE
    for i, date in enumerate (newFrame['created_at']):
        newFrame['created_at'][i] = newFrame['created_at'][i].date()
    newFrame.rename(columns={'created_at': 'Day of the tweet', 'full_text': '
    return newFrame
```

In [31]:
```
names = ['Bill', 'Elon', 'Ellen']
bill = createTweetsPerDay(billGatesTimelineFrame)
elon = createTweetsPerDay(elonMuskTimelineFrame)
ellen = createTweetsPerDay(ellenDeGeneresTimelineFrame)
tweetsPerDay = pd.concat([bill.groupby(['Day of the tweet']).count(),
                          elon.groupby(['Day of the tweet']).count(),
                          ellen.groupby(['Day of the tweet']).count()], axis=
tweetsPerDay.plot(figsize=(22,8), fontsize=16)
plt.xlabel('Day of the tweet', fontsize=18)
plt.ylabel('Number of tweets', fontsize=18)
plt.title('Number of tweets per user for each day', fontsize=18)
plt.legend(names,fontsize=20)
```

Out[31]:
```
<matplotlib.legend.Legend at 0x1b769ef7fa0>
```



From the graph above, we can clearly see that Bill gates tweets less often than the others. While Elon tweets the most with a high volume of tweets each day, even passing 20 tweets a day sometimes. Bill's and Ellen's volume is relatively consistent with a few spikes. On the other hand, Elon has two huge spikes. Let's see if we can find out why.

In [32]:
```
elonCount = elon.groupby(['Day of the tweet']).count()
elonCount.loc[elonCount['Number Of Tweets'] > 20]
```

Out[32]:

Day of the tweet	Number Of Tweets
2021-09-18	26
2021-10-14	22

By doing some research, we can find out that on 2021-09-18, it was the launch of Inspiration4, a big project from SpaceX, a company that Elon is CEO of. On 2021-10-14, Elon announced that they were in talks with airlines about installing Starlink. Both things will explain the high volume of tweets these days.

Analyzing each user's tweets

Tweets with more likes for every user

Here we will find out the tweet with the most likes and try to explain why it has the most likes.

In [33]:
```
## Bill gates tweet with the most likes - the divorce
billGatesTimelineFrame.sort_values('retweet_count',ascending=False).head(1)
```

Out[33]:

	created_at	id	full_text	truncated	in_reply_to_status_id	in_rep
136	Mon May 03 20:30:19 +0000 2021	1389316412259270657	https://t.co/padmHSgWGc	False	N/A	

In [34]:
```
## Elon Musk tweet with the most likes
elonMuskTimelineFrame.sort_values('favorite_count',ascending=False).head(1)
```

Out[34]:

	created_at	id	full_text	truncated	in_reply_to_status_id	in_reply
14	Thu Oct 21 02:41:14 +0000 2021	1451015695106560000	https://t.co/pCO0wNNZtz	False	N/A	

In [35]:
```
## Ellen tweet with the most likes - the anniversary with her wife
ellenDeGeneresTimelineFrame.sort_values('favorite_count',ascending=False).hea
```

Out[35]:

	created_at	id	full_text	truncated	in_reply_to_status_id	in_reply_
231	Mon Aug 16 15:14:53 +0000 2021	1427287755969622016	Happy anniversary, Portia. I'm the luckiest gi...	False	N/A	

Note: We could easily use the same method to find out the tweets with the most retweet, just by changing 'favorite_count' to 'retweet_count'

Analysis on users mentioned

In this section we will find out what users our influencers mention most times in their tweets,

and how often they mention someone. With this we aim to find the topics that the users mostly tweet about, since the users that they mention will have something to do with the

In [36]:
```
# Function that will return a series with the users mentioned and the number
def getUsersMentioned(timelineFrame):
    users = {}
    for listOfUsers in timelineFrame['user_mentions']:
        if listOfUsers != 'N/A':
            for user in listOfUsers:
                if user in users:
                    users[user] +=1
                else:
                    users[user] = 1
    return pd.Series(users)
```

In [37]:
```
def getUsersMentionedPercentage(timelineFrame):
    counter = 0
    for listOfUsers in timelineFrame['user_mentions']:
        if listOfUsers != 'N/A':
            counter +=1
    return counter / 300 * 100
```

In [38]:
```
bill = getUsersMentioned(billGatesTimelineFrame)
elon = getUsersMentioned(elonMuskTimelineFrame)
ellen = getUsersMentioned(ellenDeGeneresTimelineFrame)
```

In [39]:
```
bill.sort_values(ascending=False).head(10)
```

Out[39]:
```
'    326
a    170
e    132
o    131
[    125
]    125
r    112
n    111
t     87
i     83
dtype: int64
```

In [40]:
```
getUsersMentionedPercentage(billGatesTimelineFrame)
```

Out[40]:
```
41.66666666666667
```

In [41]:
```
ellen.sort_values(ascending=False).head(10)
```

Out[41]:
```
'    384
e    213
a    186
i    159
[    145
```

```
]    145
n    141
l    141
r    121
o    111
```

In [42]:
```
getUsersMentionedPercentage(ellenDeGeneresTimelineFrame)
```

Out[42]:
```
48.333333333333336
```

In [43]:
```
elon.sort_values(ascending=False).head(10)
```

Out[43]:
```
'    942
e    475
a    473
i    290
r    288
s    285
o    284
t    283
[    268
]    268
dtype: int64
```

In [44]:
```
getUsersMentionedPercentage(elonMuskTimelineFrame)
```

Out[44]:
```
89.33333333333333
```

We can see that both Elon and Bill mainly mention their companies/foundations, while Ellen mostly mentions the people that appear in her show. All the users usually mention others in their tweets. Bill mentions has the lowest user mentioned percentage, with half of his tweets having a user mentioned, and Elon has the most with only 10% of his tweets not having a user mentioned.

Topics that its user talks about

As a continuation of the above section, in this section, we will find out the words that our influencers use most often in their tweets. This will shed more light and explain better which topics they usually talk about.

In [116…

```
#Function that will generate a wordcloud using a wordlist, and shows the most
def generateWordCloud(dataFrame):
    file = open('wordlist.txt')
    wordFounds = {}
    for text in dataFrame['full_text']:
        for line in file:
            for word in line.split():
                if word in text:
                    if word in wordFounds:
                        wordFounds[word] +=1
                    else:
                        wordFounds[word] = 1
        file.seek(0)
    file.close()
    wordcloud = WordCloud(collocations = False, background_color = 'white', w
    plt.figure(figsize=(10, 8))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off");
```

In [123…

```
generateWordCloud(billGatesTimelineFrame)
```

From the wordcloud generated by Bill Gates tweets we can see that he mostly tweets about topics that have to do with the climate, the world, time and health. This is to be expected as it is known that Bill gates has been focusing mostly on improving the world in the last few years.

In [124…

```
generateWordCloud(elonMuskTimelineFrame)
```

As to be expected Elon most used word is tesla. Also the word clean is shown quite a lot which could be because Elon talks a lot about clean energy. Moreover we can see the words orbit ship and rock which could refere to his other company SpaceX.

In [122…

```
generateWordCloud(ellenDeGeneresTimelineFrame)
```

Elen most used word is birthday, and by looking at her tweets it is quite clear that she wishes happy birthday to people very often. Another words that she uses often are excited and love and by doing some reaserch we can see that she uses them to promote her show.

Analyses combining the users together

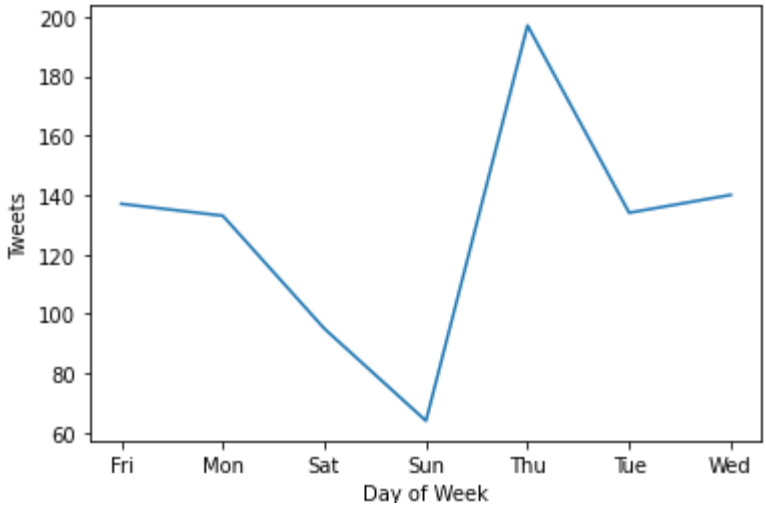
In this last section, even though it is split into three sub-sections, we will analyze the day that the users tweet more often. We will first discover the day that the tweet the most, then the month, and finally, we will combine both and try to find the time in the week that the tweet a lot. With this, we aim to see if there is any particular reason that the users tweet more at specific times.

In [49]:
```
allTweets = pd.concat([billGatesTimelineFrame, elonMuskTimelineFrame, ellenDe
```

Research on the most common day that the users tweet

In [50]:
```
allTweets['created_at'] = pd.to_datetime(allTweets['created_at'])#SHOULD BE R
allTweets['Day of Week'] = allTweets['created_at'].apply(lambda time: time.da
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
allTweets['Day of Week'] = allTweets['Day of Week'].map(dmap)
allTweets['Hour'] = allTweets['created_at'].apply(lambda time: time.hour)
byDay = allTweets.groupby('Day of Week').count()
byDay['id'].plot(ylabel ='Tweets')
```

Out[50]:
```
<AxesSubplot:xlabel='Day of Week', ylabel='Tweets'>
```

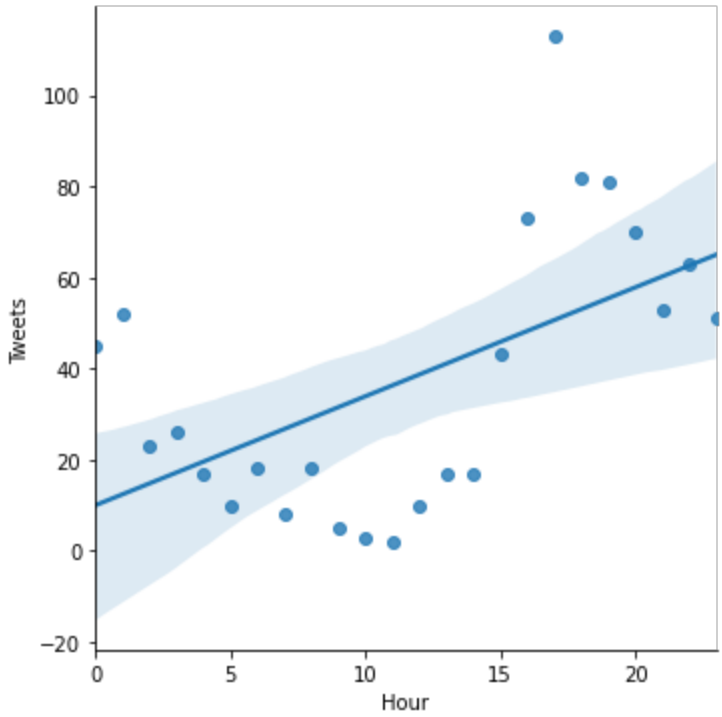


Unfortunately, pandas place the x-axis in alphabetical order, so the days are not in the correct row, but that does not really affect the point of the analysis. It is clear that the users mostly tweet on Thursday and least tweet on Sunday.

Research on the most common time that the users tweet

In [51]:
```
byHour = allTweets.groupby('Hour').count()
fig = sns.lmplot(x='Hour',y='id',data=byHour.reset_index(), )
plt.ylabel('Tweets')
```

Out[51]:
```
Text(1.6749999999999972, 0.5, 'Tweets')
```

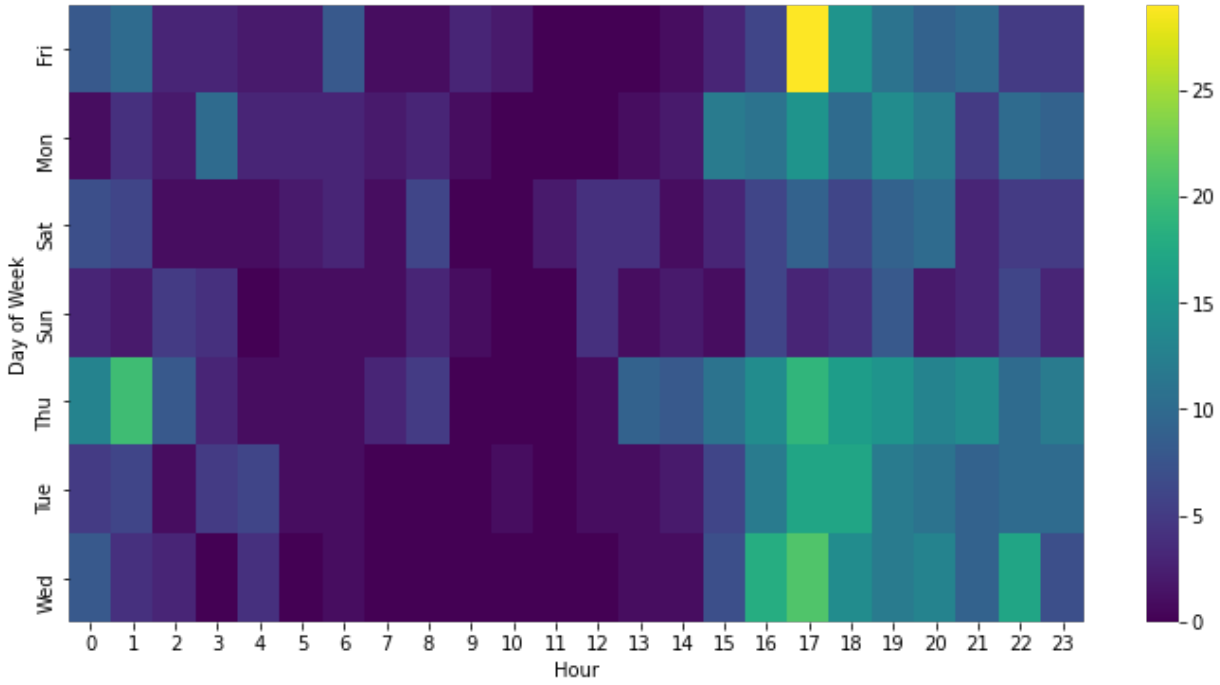


In the above graph we can see that the users tweet mostly on the afternoon, and they prefer to tweet letter on the day as the positive slope of the line indicate

Combining the last two sections

In [52]:
```
dayTweets = allTweets.groupby(['Day of Week','Hour']).count()['id'].unstack()
plt.figure(figsize=(12,6))
sns.heatmap(dayTweets,cmap='viridis')
```

Out[52]: <AxesSubplot:xlabel='Hour', ylabel='Day of Week'>



The above graphs show that the users mostly tweet on Thursday, Wednesday, and Friday, and later on the day, mainly in the evening. Although the daily plot clearly shows that most tweets are on Thursday, we can see from the heatmap that they tweet more on Fridays at five in the afternoon. Looking at the hourly plot and the heatmap, we can see that users tweet more at five in the afternoon.

Lessons Learned

On this task, we had to get really creative. Not only did we increase our knowledge of statistics, but we also had to find ways to creative and useful depict the data we had and the results we got. It was exciting to analyze the data of the influencers that we have been following and to see how data can show important details about them, for example, how we found the date of the launch of insipiration4.

Task 3 – Network analysis

Notes

As 'area of influence' we define

1. The Influencer's network of engaged users(users that retweet)
2. The geographic area of the users, belonging to the Influencer's network So the analysis

[Stage 1] Data acquisition

Due to the vast amounts of data needed to be taken for the analysis of part 3, we keeped the network-specific data here, to avoid confusion with the explanatory data analysis above

Initialization of nessesary libraries and tools

Import of necessary libraries and definition of necessary keys for Twitter API

In [53]:
```
#Core Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx


import math #math.log() used in visualisations
import tweepy #Twitter API Wrapper library
import time #Sleep functionality to keep up with twitter's Quotas
import random #Contribute with time library to keep the twitter quotas under
```

In [54]:
```
API_KEY = ''
API_SECRET_KEY = ''
ACCESS_TOKEN = ''
ACCESS_TOKEN_SECRET = ''
BEARER_TOKEN = ''
```

Initialization of tweepy library

In [55]:
```
auth = tweepy.OAuthHandler(API_KEY, API_SECRET_KEY)
auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
api = tweepy.API(auth) #Client to be used for twitter's V1 API calls
client=tweepy.Client(BEARER_TOKEN,API_KEY,API_SECRET_KEY,ACCESS_TOKEN,ACCESS_
```

The First operation to be done, is to retrieve for our choosen influencers their unique ID's

In [56]:
```
billId=str(api.get_user(screen_name='BillGates')._json['id'])
elonId=str(api.get_user(screen_name='elonmusk')._json['id'])
ellenId=str(api.get_user(screen_name='TheEllenShow')._json['id'])
[billId,elonId,ellenId] #Their retrieved ID's
```

Out[56]:
```
['50393960', '44196397', '15846407']
```

Tweets retrieval

In this section, we will retrieve the maximum allowed number of tweets, 3100 per user, because the maximum allowed retrieved tweets per request are 100, we are forced to use pagination From the following data retrieval, we are interested only in the network of each influencer, therefore we drop all of the columns but the tweet_id for the given user_id

In [57]:
```
userTweetMapFrame = pd.DataFrame(columns=['user_id','tweet_id']) #The DataFra
max_results=100 #Maximum allowed number of tweets per request = 100

#Other variables
userIds=[billId,elonId,ellenId]
paginatorSelector=0
```

In [58]:
```
#We will retrieve 100*31 = 3100 tweets for every influencer of choice(limit=3
billPaginator = tweepy.Paginator(client.get_users_tweets, id=billId,max_resul
elonPaginator = tweepy.Paginator(client.get_users_tweets, id=elonId,max_resul
ellenPaginator = tweepy.Paginator(client.get_users_tweets, id=ellenId,max_res
paginators = [billPaginator,elonPaginator,ellenPaginator]
```

In [59]:

```
#The sole purpose of this function is to enclose the following, time-intensiv
#minutes to complete, in order to be able to run this analysis top-to-bottom
#the code can be run at any time by uncommenting the call below(indicated wit
#of this computation can be retrieved at any time by reading the file tweetMa

def getAllAvailableTweets():
    userTweetMapFrame = pd.DataFrame(columns=['user_id','tweet_id'])
    for batchOfTweets in billPaginator:
        try:
            series = map(lambda x:x.id,batchOfTweets.data)
            ids=[userIds[0]]*len(batchOfTweets.data)
            frame = pd.DataFrame({'user_id':ids,'tweet_id':series})
            print(frame)
            userTweetMapFrame=userTweetMapFrame.append(frame)
            userTweetMapFrame.to_csv("tweetMapFull.csv",index=False)
            time.sleep(2)
        except:
            continue

    for batchOfTweets in elonPaginator:
        try:
            series = map(lambda x:x.id,batchOfTweets.data)
            ids=[userIds[1]]*len(batchOfTweets.data)
            frame = pd.DataFrame({'user_id':ids,'tweet_id':series})
            print(frame)
            userTweetMapFrame=userTweetMapFrame.append(frame)
            userTweetMapFrame.to_csv("tweetMapFull.csv",index=False)
            time.sleep(2)
        except:
            continue

    for batchOfTweets in ellenPaginator:
        try:
            series = map(lambda x:x.id,batchOfTweets.data)
            ids=[userIds[2]]*len(batchOfTweets.data)
            frame = pd.DataFrame({'user_id':ids,'tweet_id':series})
            print(frame)
            userTweetMapFrame=userTweetMapFrame.append(frame)
            userTweetMapFrame.to_csv("tweetMapFull.csv",index=False)
            time.sleep(2)
        except:
            continue

#getAllAvailableTweets() #*********
#Checkpoint 1
tweetMapFull = pd.read_csv("tweetMapFull.csv",dtype = {'user_id':str,'tweet_i
```

In [60]:

```
tweetMapFull #Example of the retrieved data
```

Out[60]:

	user_id	tweet_id
0	50393960	1464659614448771073
1	50393960	1464288045541199874
2	50393960	1463528104614236167

	user_id	tweet_id
3	50393960	1463186751619436546
4	50393960	1462583003272474628
...	...	...
6494	15846407	1182026314636619776
6495	15846407	1182017738627108865
6496	15846407	1182016640537354240
6497	15846407	1182000506132021249
6498	15846407	1181992707696128002

6k retweets seems a suspicious number, given that we have 3 influencers and have taken 3100 tweets from each of them, the number should be closer to 9k, lets have a look at how many tweets we have per user

In [61]:
```
tweetMapFull.groupby(by='user_id').size()
```

Out[61]:
```
user_id
15846407    3099
44196397     300
50393960    3100
dtype: int64
```

In [62]:
```
[billId,elonId,ellenId] #Their retrieved ID's
```

Out[62]:
```
['50393960', '44196397', '15846407']
```

Elon has only 300 tweets, bill and ellen has thousands, to not bias our analysis we keep 300 tweets on each account

In [63]:
```
billTweets = tweetMapFull.loc[tweetMapFull['user_id']==billId][0:300]
elonTweets = tweetMapFull.loc[tweetMapFull['user_id']==elonId][0:300]
ellenTweets = tweetMapFull.loc[tweetMapFull['user_id']==ellenId][0:300]
```

In [64]:
```
#Checkpoint 2
finalExport = billTweets.append(elonTweets).append(ellenTweets)
finalExport.to_csv("tweetMap.csv",index=False)
finalExport
```

Out[64]:

	user_id	tweet_id
0	50393960	1464659614448771073
1	50393960	1464288045541199874
2	50393960	1463528104614236167
3	50393960	1463186751619436546
4	50393960	1462583003272474628

	user_id	tweet_id
...	...	...
3695	15846407	1433245716202209283
3696	15846407	1433242268459094020
3697	15846407	1433239858735026179
3698	15846407	1433234633320136705
3699	15846407	1433188250940702721

Retweets and user locations retrieval

In the following code, we will retrieve as many retweeter accounts as possible for those 900 tweets(300 per influencer). We will also request from Twitter via the metadata, the location field of the particular user who performed the retweet. The Location field will be used for the geolocation area of influence analysis later on

In [65]:
```
influencer_tweets=pd.read_csv("tweetMap.csv",dtype = {'user_id':str,'tweet_id
```

In [66]:
```
tweets_retweeters = pd.DataFrame(columns=['tweet_id','retweeter_id','retweete
iterator=iter(influencer_tweets['tweet_id'].tolist())  #Iterator declared her
#be fault-tolerant (if the function below stops, the state of the iterator re
```

In [67]:

```
#The sole purpose of this function is to enclose the following, time-intensiv
#hours to complete, in order to be able to run this analysis top-to-bottom in
#the code can be run at any time by uncommenting the call below(indicated wit
#of this computation can be retrieved at any time by reading the file retweet

def getAllAvailableReTweets():
    tweets_retweeters= pd.DataFrame(columns=['tweet_id','retweeter_id','retwe
    while(True):
        every_influencer_tweet = next(iterator)
        try:
            time.sleep(15) #Time delay to avoid quota
            retweeters = list(client.get_retweeters(every_influencer_tweet,us
            instances=len(retweeters)
            ids = (map(lambda x:x.id,retweeters))
            locations = (map(lambda x:x.location,retweeters))
            tweets_retweeters_current = pd.DataFrame({
                'tweet_id':[every_influencer_tweet]*instances,
                'retweeter_id':ids,
                'retweeter_location':locations})
            tweets_retweeters=tweets_retweeters.append(tweets_retweeters_curr
            tweets_retweeters.to_csv("retweetMap.csv",index=False)
            print(tweets_retweeters_current)
        except Exception as e:
            print("Ignoring error, probably quota limit, continue"+str(e))
            time.sleep(10) #Further time delay to avoid quota limit


#getAllAvailableReTweets()*********
#Checkpoint 3
retweets = pd.read_csv("retweetMap.csv",dtype = {'tweet_id':str,'retweeter_id
retweets
```

Out[67]:

	Unnamed: 0	tweet_id	retweeter_id	retweeter_location
0	0	1461380998743015425	861629679886880768	Dallas, TX
1	1	1461380998743015425	1034572631025098754	El Mundo
2	2	1461380998743015425	1425487033309581314	Bristol
3	3	1461380998743015425	1409329636954148869	Surrey, British Columbia
4	4	1461380998743015425	10221	philly
...	...	...	...	...
35781	39414	1415752955152666627	1412392837916864514	bead.99@icloud.com
35782	39415	1415752955152666627	2399137929	tulungagung
35783	39416	1415752955152666627	113410319	Brasil
35784	39417	1415752955152666627	751724810640785408	Karachi, Pakistan
35785	39418	1415752955152666627	200503070	Sydney, New South Wales

35786 rows × 4 columns

Convert vague locations into real countries

In [68]:
```
#Loading from Checkpoint 3
tweets =pd.read_csv("tweetMap.csv",dtype = {'user_id':str,'tweet_id':str})
retweets = pd.read_csv("retweetMap.csv",dtype = {'tweet_id':str,'retweeter_id
```

In [69]:
```
#Eliminate entries with unknown location
retweets['retweeter_location']=retweets['retweeter_location'].map(lambda x:st
retweets=retweets.loc[retweets['retweeter_location']!='nan']
retweets
```

Out[69]:

	Unnamed: 0	tweet_id	retweeter_id	retweeter_location
0	0	1461380998743015425	861629679886880768	Dallas, TX
1	1	1461380998743015425	1034572631025098754	El Mundo
2	2	1461380998743015425	1425487033309581314	Bristol
3	3	1461380998743015425	1409329636954148869	Surrey, British Columbia
4	4	1461380998743015425	10221	philly
...	...	...	...	...
35781	39414	1415752955152666627	1412392837916864514	bead.99@icloud.com
35782	39415	1415752955152666627	2399137929	tulungagung
35783	39416	1415752955152666627	113410319	Brasil
35784	39417	1415752955152666627	751724810640785408	Karachi, Pakistan
35785	39418	1415752955152666627	200503070	Sydney, New South Wales

35786 rows × 4 columns

Any attempt to analyse the area of influence for the given account will be in vain, if we don't have standardized Data points for locations, the Location field of Twitter seems to be an non-validated field that accepts invalid locations

In [70]:
```
retweets.loc[retweets['retweeter_id']=='1253442333334548482']
```

Out[70]:

	Unnamed: 0	tweet_id	retweeter_id	retweeter_location
6013	6013	1380231432006631425	1253442333334548482	Saving the world
6815	6815	1367519920393654274	1253442333334548482	Saving the world

We need a process of converting these unreliable data points into standard countries if possible and discarding the rest, The following is the algorithm to perform such a task

To construct such an algorithm, we will perform string matching on the data points, trying to extract names of cities, areas or states, finding the country that contains the city/state in

question. Given the fact of multiple cities around the world that share similar names, we will assign on each finding a weight proportional to the population of the country in question.

In [71]:
```
#Csv Containing ~150k datapoints, of towns and their states/countries
#The data, originally as a .sql file, taken from the following Github project
#https://github.com/dr5hn/countries-states-cities-database
#And exported to .csv using a custom script and a JOIN statement
world = pd.read_csv("places_of_the_world.csv")
world
```

Out[71]:

	#	country_name	state_name	city_name
0	1	Afghanistan	Ghazni	Ghazni
1	2	Afghanistan	Badghis	Ghormach
2	3	Afghanistan	Badghis	Qala i Naw
3	4	Afghanistan	Bamyan	Bāmyān
4	5	Afghanistan	Bamyan	Panjāb
...	...	...	...	...
147961	147962	Zimbabwe	Masvingo Province	Mashava
147962	147963	Zimbabwe	Masvingo Province	Masvingo
147963	147964	Zimbabwe	Masvingo Province	Masvingo District
147964	147965	Zimbabwe	Masvingo Province	Mwenezi District
147965	147966	Zimbabwe	Masvingo Province	Zvishavane

147966 rows × 4 columns

In [72]:
```
#Saved Html page, containing the population of each country in question
#Original URL : https://www.worldometers.info/world-population/population-by-
populations = pd.read_html('populations/populations.html')[0]
populations=populations[['Country (or dependency)','Population (2020)']]
populations
```

Out[72]:

	Country (or dependency)	Population (2020)
0	China	1439323776
1	India	1380004385
2	United States	331002651
3	Indonesia	273523615
4	Pakistan	220892340
...	...	...
230	Montserrat	4992
231	Falkland Islands	3480
232	Niue	1626

	Country (or dependency)	Population (2020)
233	Tokelau	1357
234	Holy See	801

In [73]:

```
def getCountryWeight(country:str):
    """
    This simple function calculates and returns the weight of each country by
    aforementioned countries table
    """
    try:
        populationOfCountry = populations.loc[populations['Country (or depend
        return int(populationOfCountry['Population (2020)'])
    except:
        return 0
```

In [74]:

```
def translateToCountry(vagueLocation,world):
    """
    This simple function is the core of our translation. Splits the vague loc
    tokens (given some criteria, see below) and searches for occurrences of t
    places_of_the_world.csv. If multiple occurrences are found, it assigns a
    the biggest weight, or 'None' if there is no hit. The code is quite slow
    performed using some sort of a hashmap for better results, but given the
    a faster solution, but a faster solution is possible
    """

    #Step one, split location into tokens
    if("," in vagueLocation):
        #Some locations is like 'Chinatown, New York' , containing comma
        elements=list(map(lambda x:x.strip().lower(),vagueLocation.split(","))
    elif("_" in vagueLocation):
        #Some other locations, containing underscore
        elements=list(map(lambda x:x.strip().lower(),vagueLocation.split("_")
    else:
        #If none of the aforementioned exist, then split by empty character
        #If the location does not contain any of the following (' ',',','_')
        #A single-token-list will be produced (i.e 'Canada' = ['Canada'])
        elements=list(filter(lambda x:len(x)>1,vagueLocation.split(" ")))
        elements=list(map(lambda x:x.lower(),elements))

    #If Empty, short-circut
    if(len(elements)==0):
        return 'None'


    countries=[] #List to be populated with pairs of [country_found,weight_of
    for index,row in world.iterrows():

        #if Country name or State name exists, the short_circut and return th
        if(row['country_name'].lower() in elements or row['state_name'].lower
            return row['country_name']
        #If city name found, add the country name that belongs to to the coun
        elif(row['city_name'].lower() in elements):
            countries.append((row['country_name'],getCountryWeight(row['count

    #sort the countries based on the weight, the biggest is not on top
    countries.sort(reverse=True,key=lambda x:x[1])

    #If not hits found, return none, otherwise return the country with the bi
    if(len(countries)==0):
        return 'None'
    return(countries[0][0])

def passAndPrint(vague):
    """
    Simple function that used for logging during the computation
    """
    ans=translateToCountry(vague,world)
    print("vague "+str(vague)+" = true "+str(ans))
    return ans
```

In [75]:
```
#The sole purpose of this function is to enclose the following, time-intensiv
#hours to complete, in order to be able to run this analysis top-to-bottom in
#the code can be run at any time by uncommenting the call below(indicated wit
#of this computation can be retrieved at any time by reading the file retweet

def convertVagueLocationsToCountries():
    """
    This function
    """
    retweets['retweeter_location']=retweets['retweeter_location'].map(lambda
    retweets.to_csv("retweetsMapRealLocations.csv",index=False)


#convertVagueLocationsToCountries()*********

#Checkpoint 4
retweetsMapRealLocations=pd.read_csv("retweetsMapRealLocations.csv",dtype = {
retweetsMapRealLocations
```

Out[75]:

	Unnamed: 0	tweet_id	retweeter_id	retweeter_location
0	0	1461380998743015425	861629679886880768	United States
1	1	1461380998743015425	1034572631025098754	None
2	2	1461380998743015425	1425487033309581314	United States
3	3	1461380998743015425	1409329636954148869	Canada
4	4	1461380998743015425	10221	None
...	...	...	...	...
35781	39414	1415752955152666627	1412392837916864514	None
35782	39415	1415752955152666627	2399137929	Indonesia
35783	39416	1415752955152666627	113410319	None
35784	39417	1415752955152666627	751724810640785408	Pakistan
35785	39418	1415752955152666627	200503070	Australia

35786 rows × 4 columns

In [76]:
```
#Dropping all the datapoints that the previous algorithm failed to recognise
retweetsMapRealLocationsDiscardNones=retweetsMapRealLocations.loc[retweetsMap
retweetsMapRealLocationsDiscardNones.to_csv("retweetsMapRealLocationsDiscardN
retweetsMapRealLocationsDiscardNones
```

Out[76]:

	Unnamed: 0	tweet_id	retweeter_id	retweeter_location
0	0	1461380998743015425	861629679886880768	United States
2	2	1461380998743015425	1425487033309581314	United States
3	3	1461380998743015425	1409329636954148869	Canada
5	5	1461380998743015425	1053386113199603712	Brazil

	Unnamed: 0	tweet_id	retweeter_id	retweeter_location
6	6	1461380998743015425	38105260	Australia
...	...	...	...	...
35778	39411	1415752955152666627	984581120271331329	Australia
35779	39412	1415752955152666627	172579237	India
35782	39415	1415752955152666627	2399137929	Indonesia
35784	39417	1415752955152666627	751724810640785408	Pakistan
35785	39418	1415752955152666627	200503070	Australia

Final Merging of the data

We will now merge our acquired data into a unified DataFrame

In [77]:
```
"""
Merge tweetMap with retweetsMapRealLocations
plot
"""
combined=tweets.merge(retweetsMapRealLocationsDiscardNones,on="tweet_id")
bill = combined.loc[combined['user_id']==billId]
elon = combined.loc[combined['user_id']==elonId]
ellen= combined.loc[combined['user_id']==ellenId]
countries = combined['retweeter_location'].unique()

combined
```

Out[77]:

	user_id	tweet_id	Unnamed: 0	retweeter_id	retweeter_location
0	50393960	1461380998743015425	0	861629679886880768	United States
1	50393960	1461380998743015425	2	1425487033309581314	United States
2	50393960	1461380998743015425	3	1409329636954148869	Canada
3	50393960	1461380998743015425	5	1053386113199603712	Brazi
4	50393960	1461380998743015425	6	38105260	Australia
...	...	...	...	...	..
21922	15846407	1433188250940702721	34889	1260292359239036928	Taiwar
21923	15846407	1433188250940702721	34890	1226492046460014593	Canada
21924	15846407	1433188250940702721	34891	1337887775891337218	United States
21925	15846407	1433188250940702721	34892	820879792660762624	United States
21926	15846407	1433188250940702721	34893	39264553	United States

21927 rows × 5 columns

[Stage 2] Visualisation of the Aquired Data

After acquiring all the necessary data, the time is come to start producing visualisations and getting an insight on the networks generated

Account's Influence network analysis

In this section we will use our data to investigate the topology of the influencer's retweeters networks, getting some useful insights on how ideas are spread on the Twitter community

In [78]:
```
#Calculation of the weight of each influencer. Each influencer's weight is pr
#on the given account. If x has retweeted 16 bill gates tweets their weight w
weights=retweetsMapRealLocationsDiscardNones.groupby(by=["retweeter_id"]).siz
weights['weight']=weights[0]
del weights[0]
weights
```

Out[78]:

	retweeter_id	weight
0	1000190726326685698	1
1	1000218707455553537	1
2	1000245113518309376	1
3	1000323282070056960	1
4	1000368846790770688	1
...	...	...
13039	999514335423561728	4
13040	99959293	1
13041	999647495423639552	1
13042	999731945625006080	1
13043	999765192824238086	29

13044 rows × 2 columns

In [79]:
```
#Creation of the edge list
RetweeterDensityEdgeList = pd.DataFrame({'user_id':combined['user_id'],'retwe
RetweeterDensityEdgeList=RetweeterDensityEdgeList.merge(weights,on='retweeter
#Example of the edge list with weights(10 records)
RetweeterDensityEdgeList[1:10]
```

Out[79]:

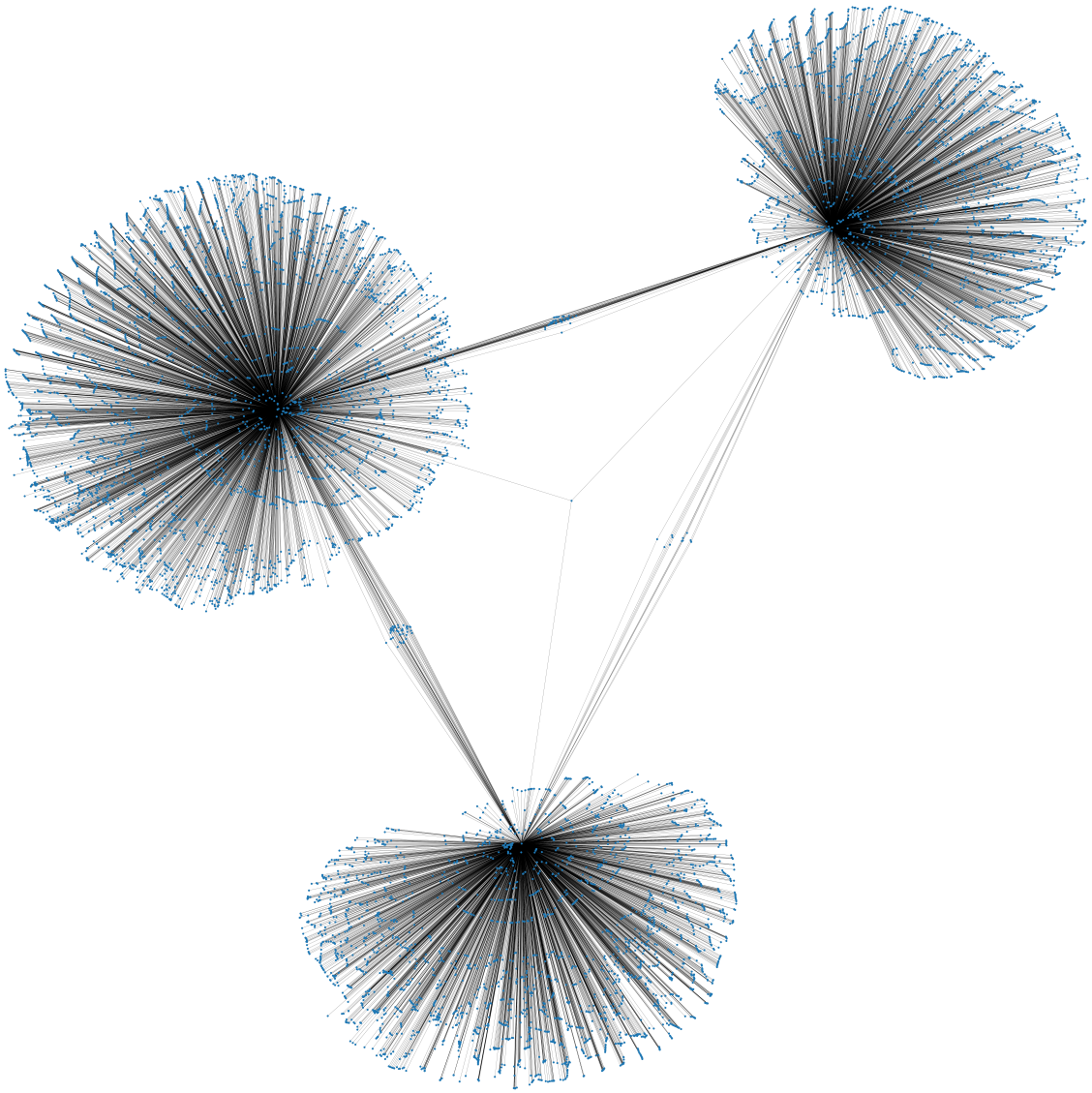
	user_id	retweeter_id	weight
1	50393960	861629679886880768	8
2	50393960	861629679886880768	8
3	50393960	861629679886880768	8
4	50393960	861629679886880768	8
5	50393960	861629679886880768	8
6	50393960	861629679886880768	8

	user_id	retweeter_id	weight
7	50393960	861629679886880768	8
8	50393960	1425487033309581314	2

In [80]:
```
def drawRetweeterDensity(edge_list):
    """
    This function accepts the edge_list of the influence network and plots it
    Returns a list of statistics for the produced graph
    """
    areaOfInfluenceGraph= nx.Graph()
    pos = nx.spring_layout(areaOfInfluenceGraph)
    for index, row in edge_list.iterrows():
        areaOfInfluenceGraph.add_edges_from([ (row['user_id'],row['retweeter_
    pos = nx.spring_layout(areaOfInfluenceGraph)
    plt.figure(3,figsize=(150,150))
    nx.draw(areaOfInfluenceGraph,pos,with_labels=False)
    plt.show()
    return [
        nx.degree_histogram(areaOfInfluenceGraph),
        nx.clustering(areaOfInfluenceGraph),
        nx.betweenness_centrality(areaOfInfluenceGraph),
        nx.degree_assortativity_coefficient(areaOfInfluenceGraph)]
```

In [85]:
```
def drawDegreeDistributionHistogram(degree_hist):
    plt.bar(range(0,len(degree_hist)),degree_hist)
    plt.xlabel("Degree")
    plt.ylabel("Frequency")
    plt.show()
def drawDegreeDistributionHistogramFiltered(degree_hist):
    k,v = zip(*degree_hist)
    plt.bar(k,v)
    plt.xlabel("Degree")
    plt.ylabel("Frequency")
    plt.show()
```

In [81]:
```
stats = drawRetweeterDensity(RetweeterDensityEdgeList)
```



Network Statistics

In this section, we will investigate the topology and the statistics of the aforementioned network

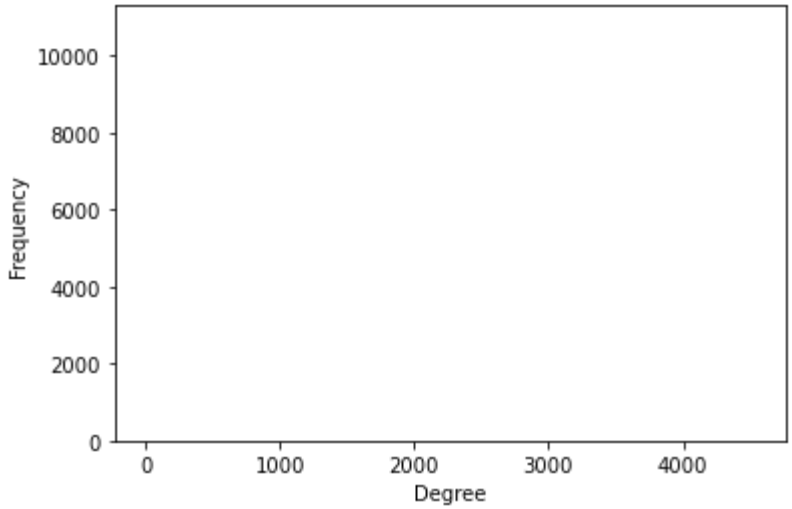
The produced network has a very interesting topology indeed, it shows how information flows between communities of different people. There are clusters of people sharing common ideas but, with the help of a few retweeters that retweet content from more than one account, the area of influence is significantly increased. Evidence suggests that in most real-world networks, as well as social networks, nodes tend to create tightly knit groups characterised by a relatively high density of ties(Holland and Leinhardt, 1971,Watts and Strogatz, 1998), something that we can confirm by our networks topology. Let's investigate the topology with the help of statistics to extract insightful knowledge.

Degree Distribution

In [83]:
```
#Aquire the degree distibution
degree_distribution = stats[0]
```

Due to the nature of the network in question, the majority of the nodes have 0 connectivity, this shadows any meaningful information from the degree distribution histogram, as seen below

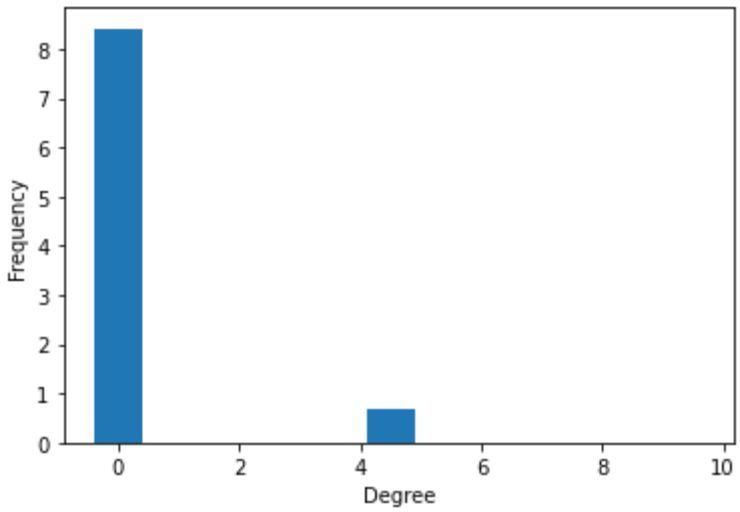
In [86]:
```
drawDegreeDistributionHistogram(degree_distribution)
```



We need a way to expose the nature of the network distribution histogram, for this, we will perform the following operations

1. Omit the degrees occurrences when degree=0
2. plot the logarithm of the frequency and the degree

In [87]:
```
zipped=list(zip(degree_distribution,range(0,len(degree_distribution))))
zipped_filtered = list(filter(lambda x:x[0]!=0,zipped))
zipped_filtered_logged = list(map(lambda x:(math.log(x[0]),math.log(x[1])),zi
drawDegreeDistributionHistogramFiltered(zipped_filtered_logged)
```



In [88]:
```
zipped_filtered
```

Out[88]:
```
[(10760, 1), (90, 2), (1, 3), (1, 3190), (1, 3215), (1, 4538)]
```

We can now reveal the true nature of the histogram, we see that the majority of the nodes (>99%) have connectivity=1 (only interacted with the influencer in question), a few nodes that interacted with 2 influencers and only one(!!) interacted with all of 3 influencers in question. Finally, the 3 remaining nodes are the influencer nodes themselves.

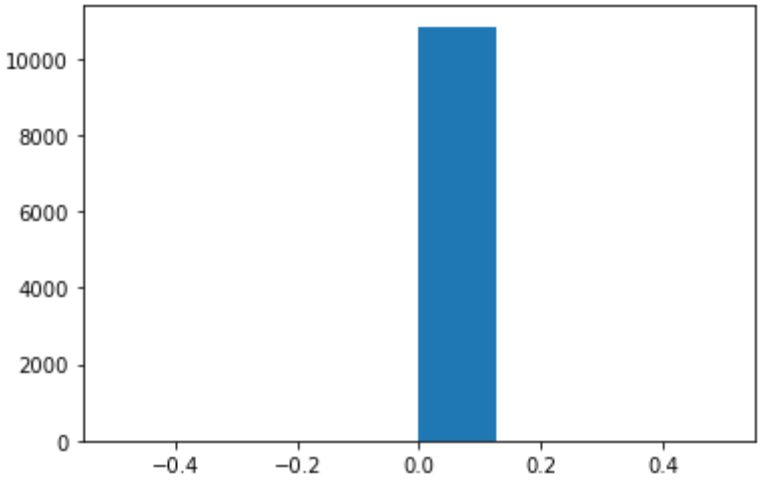
Cluster Coefficient

In [89]:
```
cluster_coeff = stats[1] #Get Clustering coeficcient statistic
```

The cluster coefficient of this network is zero, as every node has zero clustering coefficient, that's because of the topology of the network itself, as there is no clusters formed (3 star topologies loosely connected)

In [90]:
```
plt.hist(cluster_coeff.values(),bins=8)
```

Out[90]:
```
(array([    0.,     0.,     0.,     0., 10854.,     0.,     0.,     0.]),
 array([-0.5  , -0.375, -0.25 , -0.125,  0.   ,  0.125,  0.25 ,  0.375,
         0.5  ]),
 <BarContainer object of 8 artists>)
```



In [91]:
```
#There is not a single node with cluster coefficient other than 0
series_cluster_coeff = pd.Series(cluster_coeff.values()).reset_index()
series_cluster_coeff.loc[series_cluster_coeff[0]!=0]
```

Out[91]:

	index	0

Betweenness Centrality

Given the topology of the network, and combining the information taken from the degree distribution, we can infer that

1. There are 90 nodes that retweet from more than 1 account
2. There is 1 node retweeting from all 3 accounts selected
3. Therefore, there are 91(common) + 3(3 influencer nodes) nodes with a centrality of more than 0

Let's have a look and verify our findings

In [92]:
```
betweenness_centrality=pd.Series(stats[2]).reset_index()
important_nodes = betweenness_centrality.loc[betweenness_centrality[0]>0.0]
print(important_nodes)
print(len(important_nodes) == 91 + 3)
```

```
                    index         0
0                50393960  0.656112
6     1455980057499869185  0.006489
7                44196397  0.496024
9               148405713  0.006489
14      701314473328414720  0.005485
...                    ...       ...
6403   1367909681952350208  0.014140
6627   1164496701832519680  0.014140
6889             834672494  0.014140
7121    867828035864723456  0.014140
7161   1060981731489132544  0.014140

[94 rows x 2 columns]
True
```

Ascosiativity

As expected, due to the nature of the network topology (>99% of nodes, connected with only one node) the network associativity expected to be extremely high

In [93]:
```
ascosiativity=pd.Series(stats[3]).reset_index()
ascosiativity
```

Out[93]:

	index	0
0	0	-0.942149

Geographic Influence

In this section we will use our data to extract intuition on the geographic area of influence In the first attempt, we will draw our graph between each influencer and the country, Each country distance from the account node will be proportional of the number of unique retweets coming from this country.

In [94]:
```
#Calculation of the edge list with weights for the geographic area of influen
countryInfluence1 = combined.groupby(by=['user_id','retweeter_location']).siz
countryInfluence1['weight'] = countryInfluence1[0]
del countryInfluence1[0]
```

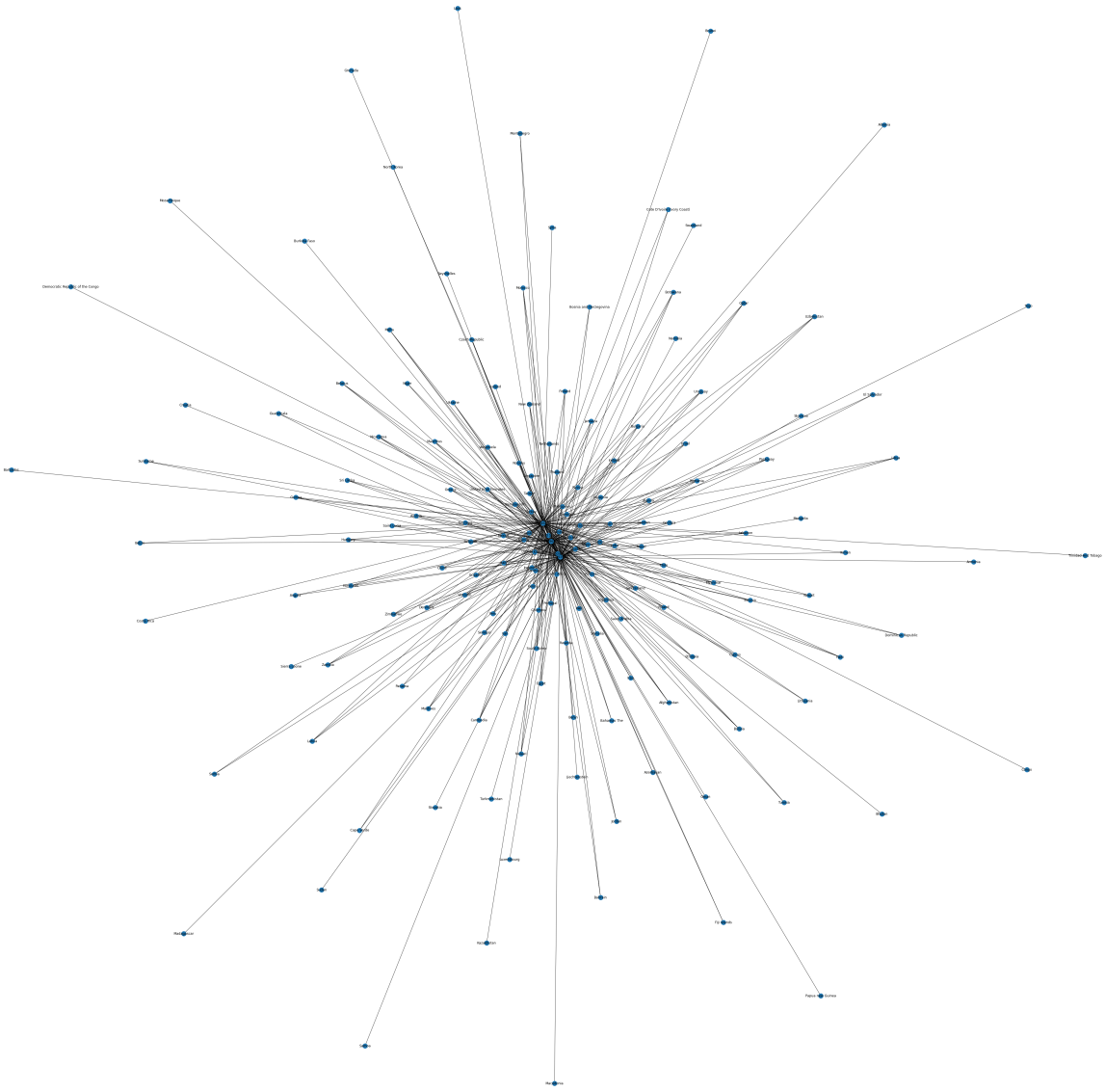
In [95]:
```
#Example of the edge list with weights(10 records)
countryInfluence1[1:10]
```

Out[95]:

	user_id	retweeter_location	weight
1	15846407	Albania	1
2	15846407	Algeria	1
3	15846407	Argentina	20
4	15846407	Australia	212
5	15846407	Austria	4
6	15846407	Bangladesh	4
7	15846407	Barbados	1
8	15846407	Belarus	2
9	15846407	Belgium	44

In [96]:
```
#Draw central account and retweeterIDs, the distance from center is how many
def drawAccountInfluence1(account,n):
    """
    This function accepts the edge_list of the Geographic Influence network a
    """
    areaOfInfluenceGraph= nx.Graph()
    pos = nx.spring_layout(areaOfInfluenceGraph)
    for index, row in account.iterrows():
        areaOfInfluenceGraph.add_edges_from([ (row['user_id'],row['retweeter_
        areaOfInfluenceGraph.add_edges_from([ (row['retweeter_location'],row[
    pos = nx.spring_layout(areaOfInfluenceGraph)
    plt.figure(3,figsize=(60,60))
    nx.draw(areaOfInfluenceGraph,pos,with_labels=True)
    plt.show()
```

In [97]:
```
drawAccountInfluence1(countryInfluence1,countryInfluence1.size)
```



Geographic Influence improved

The previous approach had a core issue, it didn't account for engagement. Engagement quantifies how much user interaction there is with your campaign (Source). There are two statistics that we can count on

1. Number of Unique retweets per account per country.
2. Number of Unique retweeters per account per country.

If we account only for the first parameter(as we did on the previous graph) then a country A with 100 retweets coming from 100 retweeters(accounts) will be equal to a country B with 100 retweets from 30 retweeters(accounts). Obviously, the country B is way more engaged

We define our weight function as...

$$f(u,r) = 1 - \frac{u}{r}$$

Where

1. u: Users engaged with this account in the given country
2. r: Retweets produced by those users for the given country

In [98]:
```
uniqueUsers = combined.groupby(by=['user_id','tweet_id','retweeter_location']
uniqueRetweets = combined.groupby(by=['user_id','retweeter_location']).size()
```

Calculation of the edge list for the improved geographic location graph

In [99]:
```
#Calculation for bill gates

billUsers = uniqueUsers.loc[uniqueUsers['user_id']==billId].groupby(by=['retw
billRetweets = uniqueRetweets.loc[uniqueRetweets['user_id']==billId].reset_in

del billRetweets['index']
billUsers['unique_users'] = billUsers[0]
del billUsers[0]
billRetweets['unique_tweets']=billRetweets[0]
del billRetweets[0]

mergedBill = billUsers.merge(billRetweets,on='retweeter_location')
mergedBill['score']=1-mergedBill['unique_users']/mergedBill['unique_tweets']

#Example, Score of United States
mergedBill.loc[mergedBill['retweeter_location']=='United States']
```

Out[99]:

	retweeter_location	unique_users	user_id	unique_tweets	score
130	United States	248	50393960	2208	0.887681

In [100…
```
#Calculation for elon musk

elonUsers = uniqueUsers.loc[uniqueUsers['user_id']==elonId].groupby(by=['retw
elonRetweets = uniqueRetweets.loc[uniqueRetweets['user_id']==elonId].reset_in


del elonRetweets['index']
elonUsers['unique_users'] = elonUsers[0]
del elonUsers[0]
elonRetweets['unique_tweets']=elonRetweets[0]
del elonRetweets[0]

mergedElon = elonUsers.merge(elonRetweets,on='retweeter_location')
mergedElon['score']=1-mergedElon['unique_users']/mergedElon['unique_tweets']

#Example, Score of United States
mergedElon.loc[mergedElon['retweeter_location']=='United States']
```

Out[100…

	retweeter_location	unique_users	user_id	unique_tweets	score
108	United States	164	44196397	2557	0.935862

In [101…

```
#Calculation for ellen DeGenerous

ellenUsers = uniqueUsers.loc[uniqueUsers['user_id']==ellenId].groupby(by=['re
ellenRetweets = uniqueRetweets.loc[uniqueRetweets['user_id']==ellenId].reset_

del ellenRetweets['index']
ellenUsers['unique_users'] = ellenUsers[0]
del ellenUsers[0]
ellenRetweets['unique_tweets']=ellenRetweets[0]
del ellenRetweets[0]

mergedEllen = ellenUsers.merge(ellenRetweets,on='retweeter_location')
mergedEllen['score']=1-mergedEllen['unique_users']/mergedEllen['unique_tweets

#Example, Score of United States
mergedEllen.loc[mergedEllen['retweeter_location']=='United States']
```

Out[101…

	retweeter_location	unique_users	user_id	unique_tweets	score
113	United States	253	15846407	3584	0.929408

In [102…

```
#Merged edge list with weights
merged=mergedBill.append(mergedElon).append(mergedEllen)
#Example of edge list with weights(10 records)
merged[1:10]
```

Out[102…

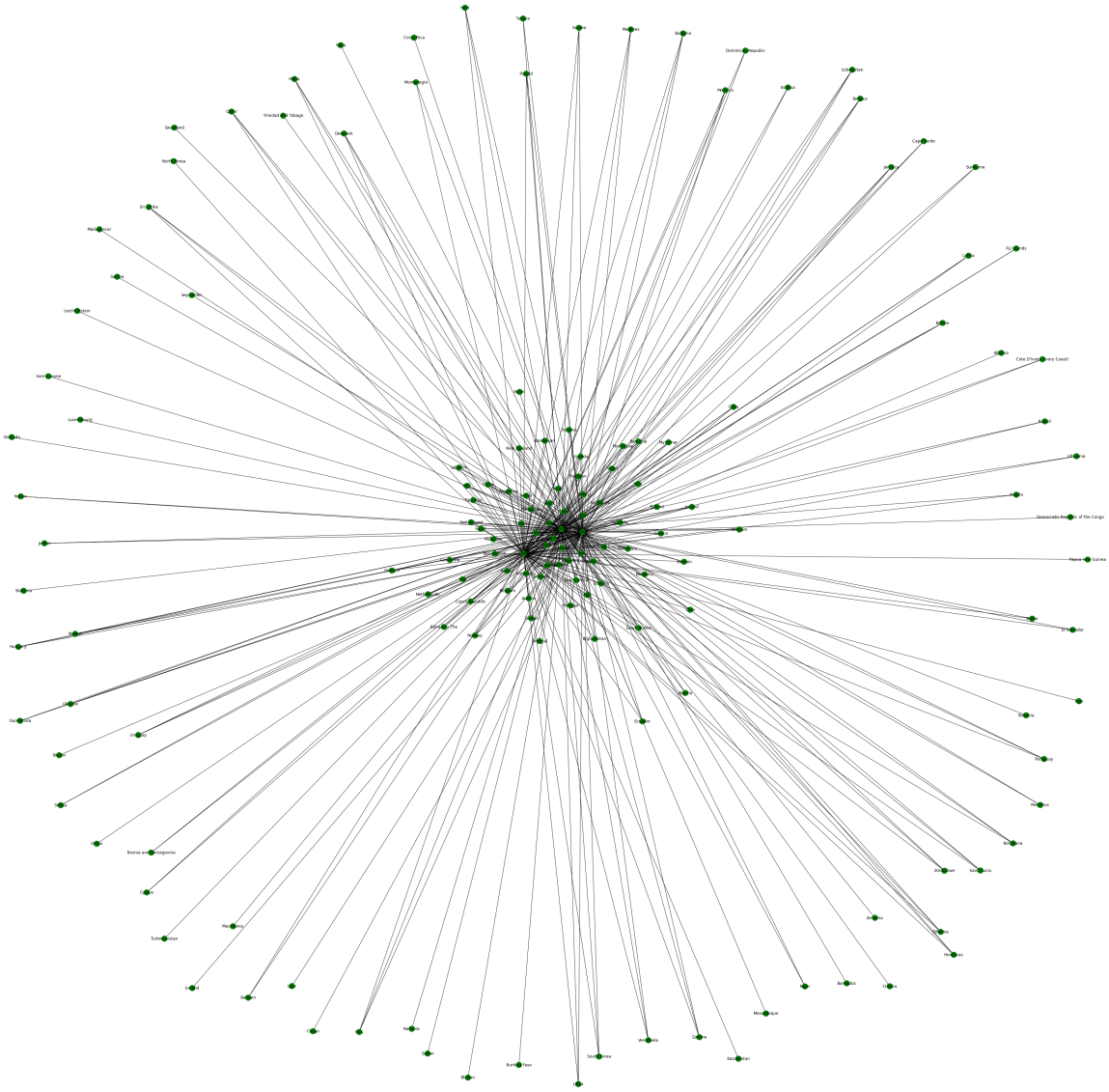
	retweeter_location	unique_users	user_id	unique_tweets	score
1	Algeria	44	50393960	45	0.022222
2	Angola	5	50393960	5	0.000000
3	Argentina	22	50393960	23	0.043478
4	Armenia	2	50393960	2	0.000000
5	Australia	159	50393960	242	0.342975
6	Austria	9	50393960	11	0.181818
7	Azerbaijan	5	50393960	6	0.166667
8	Bahamas The	2	50393960	2	0.000000
9	Bahrain	2	50393960	2	0.000000

In [103…

```
def drawEngagement(account,color_map=None):
    """
    This function accepts the edge_list of the improvement Geographic Influen
    """
    color_map= ["green"]*(len(account['user_id'].unique())+len(account['retwe
    areaOfInfluenceGraph= nx.Graph()
    pos = nx.spring_layout(areaOfInfluenceGraph)
    for index, row in account.iterrows():
        areaOfInfluenceGraph.add_edge(row['user_id'],row['retweeter_location'
        areaOfInfluenceGraph.add_edge(row['retweeter_location'],row['user_id'
    pos = nx.spring_layout(areaOfInfluenceGraph)
    plt.figure(3,figsize=(50,50))
    nx.draw(areaOfInfluenceGraph,pos,with_labels=True,node_color=color_map)
    plt.show()
    return [
        nx.degree_histogram(areaOfInfluenceGraph),
        nx.clustering(areaOfInfluenceGraph),
        nx.betweenness_centrality(areaOfInfluenceGraph),
        nx.degree_assortativity_coefficient(areaOfInfluenceGraph)]
```

In [104…

```
stats=drawEngagement(merged)
```



Network Statistics

The network produced, contains similar characteristics to the previous network, providing however an insight view on the influenced geographic areas. Let's have a quick look at the statistics of this network

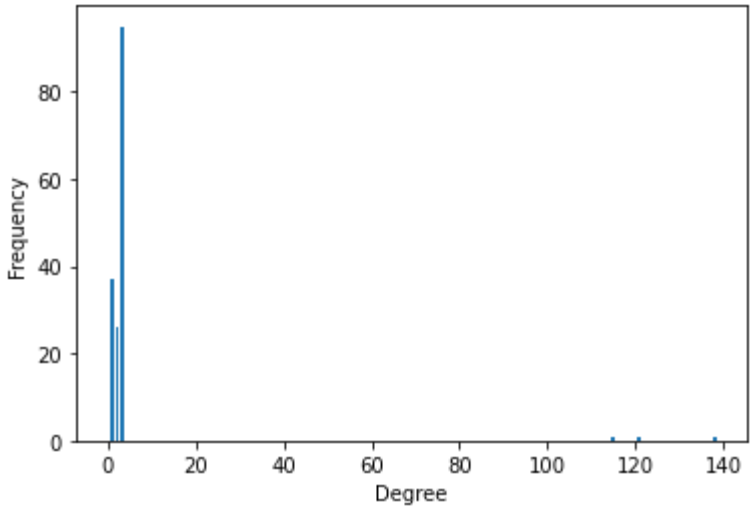
Degree Distribution

In [105…

```
#Aquire the degree distibution
degree_distribution = stats[0]
```

In [106…

```
drawDegreeDistributionHistogram(degree_distribution)
```



We can see that the majority of the countries have very low engagement scores, with very few having an extremely high score(closer to the centre), some of the countries are shared between the accounts and some others not

In [107…
```
high_engaged_countries=merged.loc[merged['score']>0.6]
low_engaged_countries=merged.loc[merged['score']<=0.6]
```

In [108…
```
high_engaged_countries
```

Out[108…

	retweeter_location	unique_users	user_id	unique_tweets	score
53	India	240	50393960	1778	0.865017
58	Israel	2	50393960	10	0.800000
130	United States	248	50393960	2208	0.887681
45	India	154	44196397	645	0.761240
108	United States	164	44196397	2557	0.935862
88	Rwanda	5	15846407	26	0.807692
113	United States	253	15846407	3584	0.929408

In [109…
```
low_engaged_countries
```

Out[109…

	retweeter_location	unique_users	user_id	unique_tweets	score
0	Afghanistan	9	50393960	10	0.100000
1	Algeria	44	50393960	45	0.022222
2	Angola	5	50393960	5	0.000000
3	Argentina	22	50393960	23	0.043478
4	Armenia	2	50393960	2	0.000000
...	...	...	...	...	...

	retweeter_location	unique_users	user_id	unique_tweets	score
116	Venezuela	3	15846407	3	0.000000
117	Vietnam	7	15846407	7	0.000000
118	Yemen	1	15846407	1	0.000000
119	Zambia	2	15846407	2	0.000000
120	Zimbabwe	1	15846407	1	0.000000

Betweenness Centrality

The Betweenness Centrality of this network is trivial, due to the topology, there are 3 central account nodes(influencer's nodes) that have very high betweenness centrality score, the rest of the nodes contain near zero score

In [110…

```
betweenness_centrality=pd.Series(stats[2]).reset_index()
important_nodes = betweenness_centrality.loc[betweenness_centrality[0]>0.1]
not_important_nodes = betweenness_centrality.loc[betweenness_centrality[0]<=0
important_nodes
```

Out[110…

	index	0
0	50393960	0.467303
139	44196397	0.242331
148	15846407	0.324563

In [111…

```
not_important_nodes
```

Out[111…

	index	0
1	Afghanistan	0.000570
2	Algeria	0.000307
3	Angola	0.000133
4	Argentina	0.000570
5	Armenia	0.000000
...	...	...
156	Laos	0.000000
157	Seychelles	0.000000
158	Swaziland	0.000000
159	Syria	0.000000
160	Trinidad And Tobago	0.000000

158 rows × 2 columns

Ascosiativity

As expected again, due to the nature of the network topology (>99% of nodes, connected with only one node) the network associativity expected to be extremely high

In [112…
```
ascosiativity=pd.Series(stats[3]).reset_index()
ascosiativity
```

Out[112…

	index	0
0	0	-0.987365

Cluster Coefficient

The cluster coefficient of this network is zero, as every node has zero clustering coefficient, that's because of the topology of the network itself, as there is no clusters formed (3 star topologies interconnected)

In [113…
```
cluster_coeff = stats[1] #Get Clustering coeficcient statistic
```

In [114…
```
#There is not a single node with cluster coefficient other than 0
series_cluster_coeff = pd.Series(cluster_coeff.values()).reset_index()
series_cluster_coeff.loc[series_cluster_coeff[0]!=0]
```

Out[114…

	index	0

Lessons Learned

Using networks, we furthered our understanding of how a social network is structured and how the ideas are transferred there. We also learned how to depict networks and how to calculate the required values.