SECTION 1: Stages Achieved

This project is on twitter sentiment analysis and entity recognition which is part of natural language processing (NLP) and covers several stages namely; data collection, data pre-processing, model building, model training, model testing, model evaluation and visualization. All the five stages were completed successfully. The algorithm used is Naïve Bayes and support vector machine.

SECTION 2: Bugs and Weaknesses

One limitation is the fact that the tweets collected on London marathon which is a sports event and therefore most sentiments would be positive. The network graph generated did not come out clear as expected.

SECTION 3: Contributions

This project was undertaken individually as I did not find someone to partner with.

SECTION 4: Documentation

Stage 1: Dataset Selection

The objective of this project was to collect tweets on the recent London Marathon 2019. The London marathon is a major sports event and this year it attracted over 42000 runners and generated over 1 billion pounds in revenue. In this project, we sought to analyse the tweets to gain insight on the most influential twitters in this event. Secondly, a sentiment analysis would help us understand how marathon fans feel about the event. The dataset was collected using Twitter-API streaming. An additional dataset was downloaded from twitter corpus samples to be used as labelled training data. In the process of the analysis, information was gathered on London marathon followers and their friends.

Stage 2: Data- Pre-processing

The main activity in data pre-processing was to clean tweets and remove unnecessary words for easy analysis and visualization. The other reason is to prepare the data in a form that could be fed to machine learning algorithms. The cleaning was done using various methods such as; bag of words and tokenization.

Stage 3: Model Building

The models used were Naïve Bayes and support vector machine. These two models were chosen because they are known to perform well in text classification. The social network graph was implemented using to Networkx.

Part 1: Sentiment Analysis Using Naïve Bayes and Support Vector Machine

Naïve Bayes and Support Vector Machine are very popular NLP machine learning algorithms used to classify texts. They are based on supervised learning. In this part, we train the two algorithms on labelled data comprising of 10000 texts and labels. This dataset is cleaned and split into training and testing data, trained, and tested on training dataset before being applied on real tweet data.

Part 2: Social Network Graph-Entity Analysis

In the second part, we generate London marathon event tweet entities on followers and friends. The tweets are downloaded using twitterAPI. The data is used to create a social network graph to visualize the relationships and determine influential nodes.

Stage 4: Model validation and Testing

The Naïve Bayes model is tested using actual tweets from London marathon event and validated using accuracy , precision and confusion matrix.

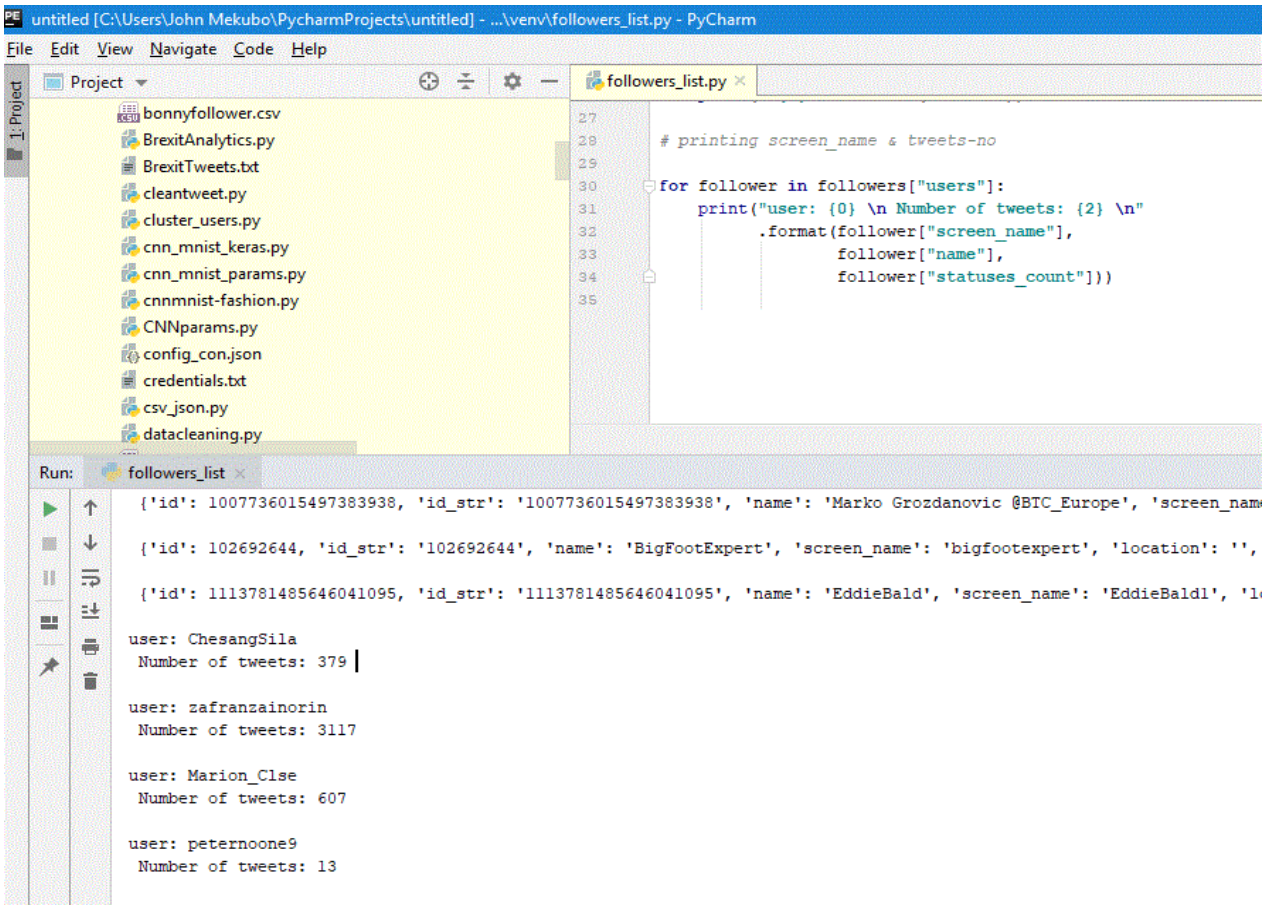
Stage 5: Visualization

Basic statistics on London marathon event. The following results are displayed in the command prompt after running followers.py file from users/londonmarathon folders as seen below. Remember the 15000 followers are not all as this is the max limit set by twitter API.

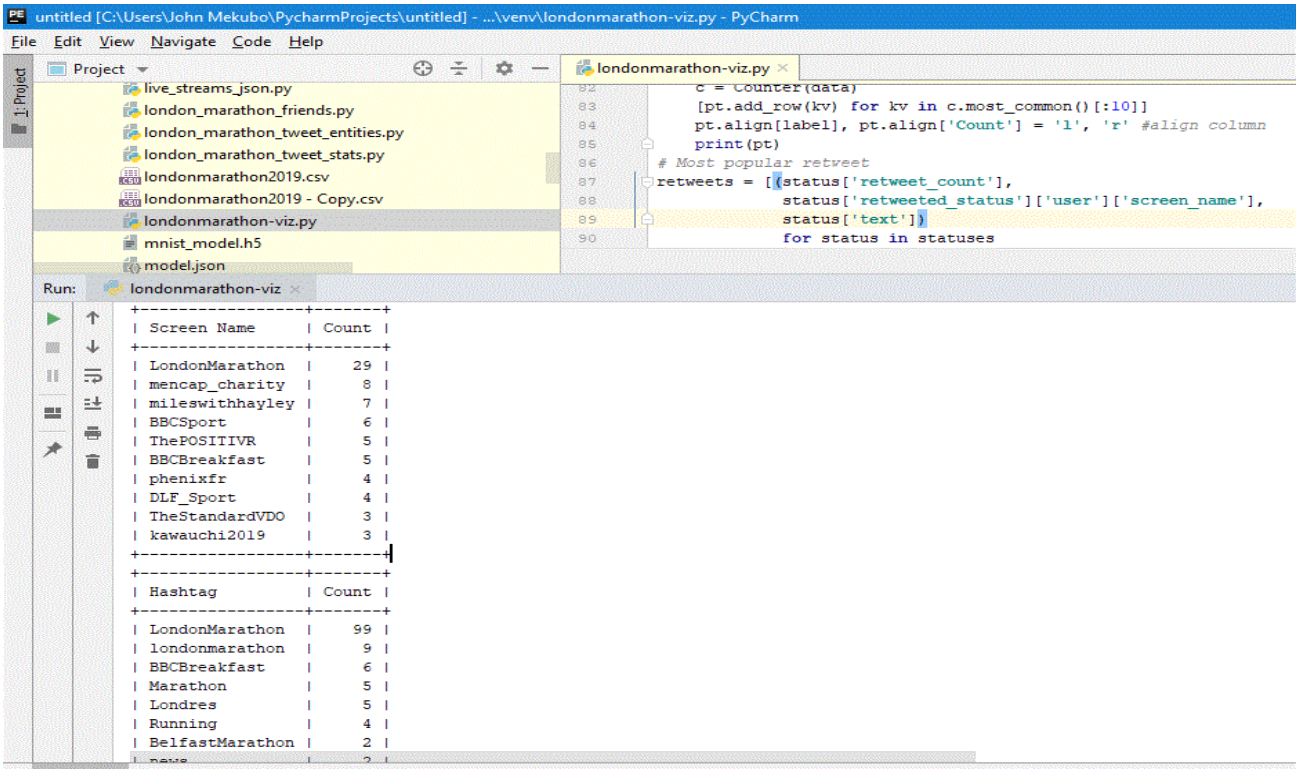
```
C:\Users\John Mekubo\PycharmProjects\untitled\venv>python followers.py londonmarathon
More results available. sleeping for 60 seconds to avoid rate limit
More results available. sleeping for 60 seconds to avoid rate limit
More results available. sleeping for 60 seconds to avoid rate limit

C:\Users\John Mekubo\PycharmProjects\untitled\venv>python followers_stats.py londonmarathon
 londonmarathon has 15000 followers
 londonmarathon has 1093 friends
 londonmarathon has 24 mutual friends
 1069 friends are not following londonmarathon back
 14976 followers are not followed back by londonmarathon
```

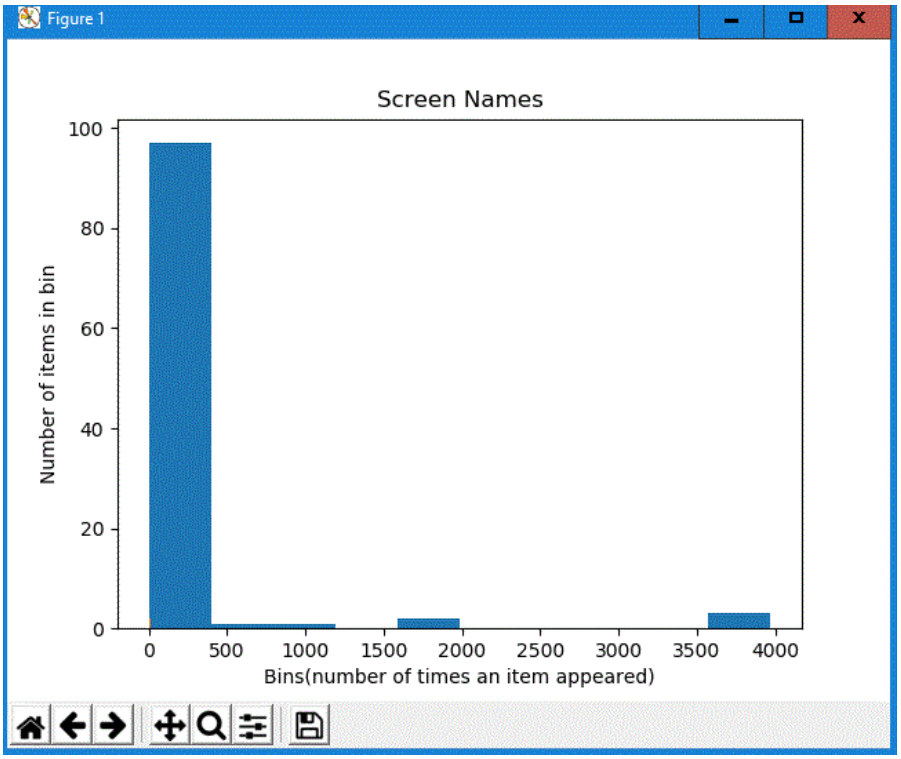
Displays some londonmarathon follower details



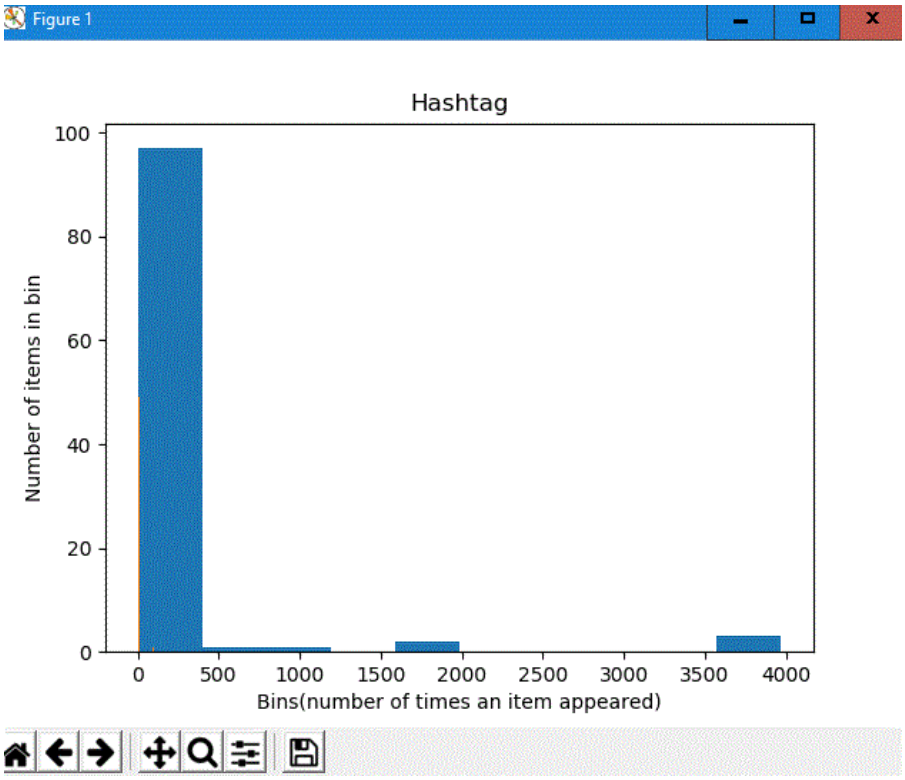
Displays some follower details such as number of tweets screen names



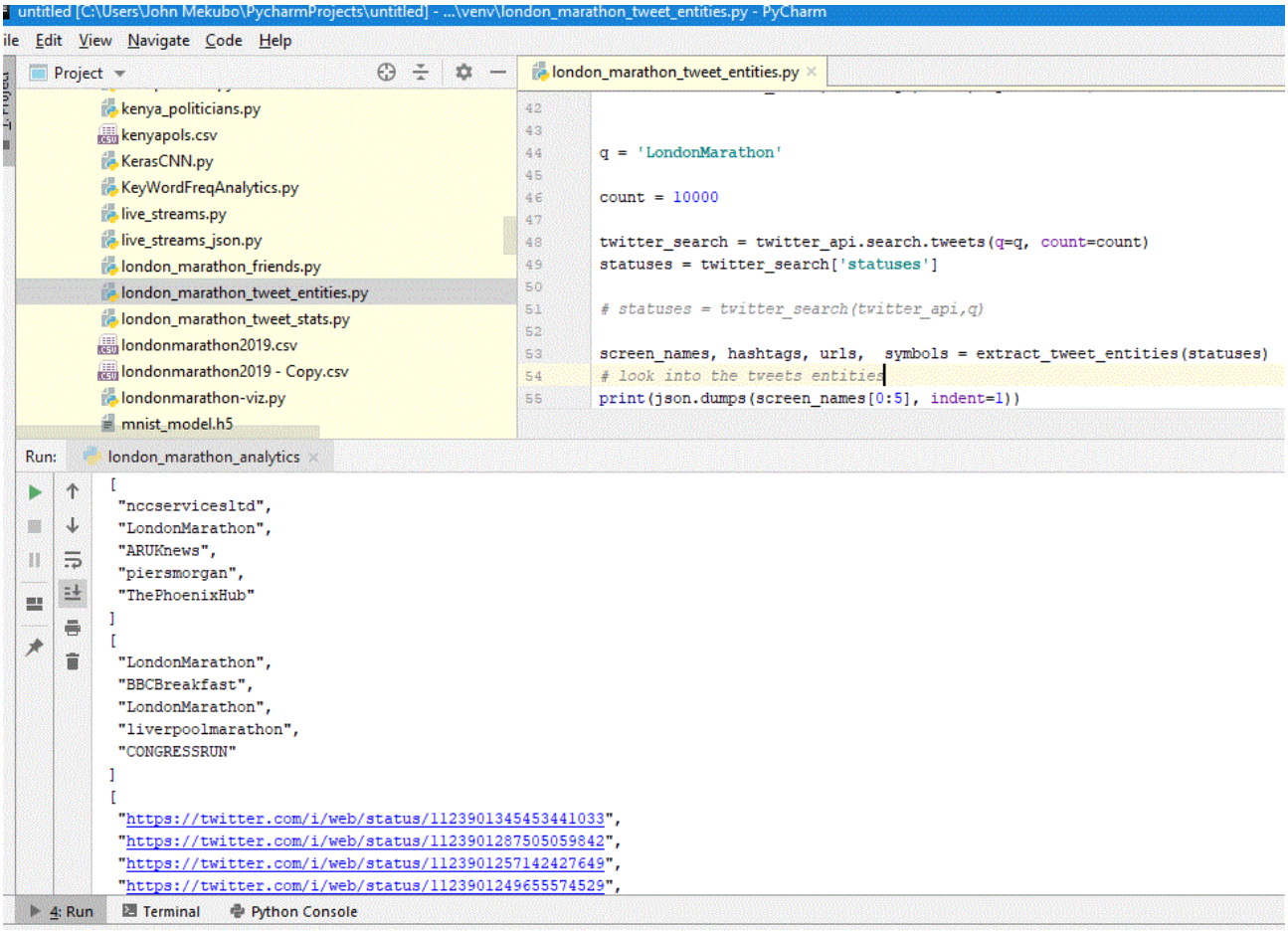
Above statistics show rank of hashtags and screennames appearing in tweets



Screen names histogram -Londonmarathon is Highest-29



hashtags histogram- Londonmarathon is Highest-99



Top 5 screen names, Hashtags and Urls

WordCloud- Displays most popular words in big size and bold

Section 5: Annotated Screenshots

Part 1 Text Classification

load data from labelled samples to be used for training

```
# conection variables
#Variables that contains the user credentials to access Twitter API
consumer_key = "cpXjUUvGMNouRBpt5opcCGOEZ"                              #API Key
consumer_secret = "zkrm2pjZPFa3etVR0xL4Q7EZpsQtoZO77zMF5yytMYeImXmoMX"   #API Secret
access_token = "1095637498510934016-BlferFMA6g4JkhawTeddsuLWxgytwy"      #Access Token
access_token_secret = "xE1wfrjRG3u1Qzu7lNqDeRVVaY7TfaTQDh3H97sbYdN5m"   #Token Secret
```

```
# authenticate

auth = tweepy.OAuthHandler(consumer_key=consumer_key, consumer_secret=consumer_secret)
auth.set_access_token(access_token,access_token_secret)
api =tweepy.API(auth)
```

```
#          **********LOAD AND PRE-PROCESS & TRAIN LABELLED TRAINNING CORPUS**************
```

```
# load corpus labeled data
ML_Corpus = pd.read_csv("data/corpus.csv",encoding='latin-1')
list(ML_Corpus.columns.values)
```

```
['text', 'label']
```

Pre-processing

Clean data by removing unnecessary words and features. We also perform stemming and Lemmatizing.

```
# data pre-processing: takes abit of time in my computer
# removing unwanted text
# Remove blank rows if any.
ML_Corpus['text'].dropna(inplace=True)
# convert text to lower case.
ML_Corpus['text'] = [entry.lower() for entry in ML_Corpus['text']]
# Tokenization
ML_Corpus['text']= [word_tokenize(entry) for entry in ML_Corpus['text']]
# Remove Stop words, Non-Numeric and perfom Word Stemming/Lemmenting.
# WordNetLemmatizer
tag_map = defaultdict(lambda : wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV
for index,entry in enumerate(ML_Corpus['text']):
    # Declaring Empty List to store the words that follow the rules for this step
    Final_words = []
    # Initializing WordNetLemmatizer()
    word_Lemmatized = WordNetLemmatizer()
    # pos_tag function below will provide the 'tag' i.e if the word is Noun(N) or Verb(V) or something else.
    for word, tag in pos_tag(entry):
        # Below condition is to check for Stop words and consider only alphabets
        if word not in stopwords.words('english') and word.isalpha():
            word_Final = word_Lemmatized.lemmatize(word,tag_map[tag[0]])
            Final_words.append(word_Final)
    # The final processed set of words for each iteration will be stored in 'text_final'
    ML_Corpus.loc[index,'text_final'] = str(Final_words)
```

```
# split data into rain and test, we are creating train set 80% and test set 20%
Train_X, Test_X, Train_Y, Test_Y = model_selection.train_test_split(ML_Corpus['text_final'],ML_Corpus['label'],test_size=0.2)
```

Transformation

transform the text from text to numeric data.

```
# transform data by encording from text to numerics that can be fed to machine learning algorithims
Encoder = LabelEncoder()
Train_Y = Encoder.fit_transform(Train_Y)
Test_Y = Encoder.fit_transform(Test_Y)
```

```
# word vectorize-convert text into vector matrix using Term Frequency – Inverse Document(TFID)
Tfidf_vect = TfidfVectorizer(max_features=10000)
Tfidf_vect.fit(ML_Corpus['text_final'])
Train_X_Tfidf = Tfidf_vect.transform(Train_X)
Test_X_Tfidf = Tfidf_vect.transform(Test_X)
```

```
# display vocabulary of labeled data
print(Tfidf_vect.vocabulary_)
```

```
{'stun': 4271, 'even': 1529, 'sound': 4116, 'track': 4556, 'beautiful': 387, 'paint': 3160, 'mind': 2839, 'well': 4865, 'woul
d': 4951, 'recomend': 3598, 'people': 3225, 'hate': 2054, 'video': 4763, 'game': 1855, 'music': 2925, 'play': 3298, 'cross':
1032, 'ever': 1533, 'best': 427, 'back': 330, 'away': 320, 'crude': 1035, 'take': 4371, 'fresh': 1812, 'step': 4210, 'guita
r': 2004, 'soulful': 4113, 'orchestra': 3101, 'impress': 2245, 'anyone': 203, 'care': 645, 'listen': 2611, 'soundtrack': 411
8, 'anything': 204, 'read': 3564, 'lot': 2655, 'review': 3721, 'say': 3840, 'figure': 1694, 'write': 4957, 'disagree': 1238,
'bit': 449, 'ultimate': 4633, 'masterpiece': 2756, 'timeless': 4506, 'year': 4976, 'beauty': 389, 'simply': 4011, 'refuse': 3
621, 'price': 3407, 'tag': 4370, 'pretty': 3402, 'must': 2931, 'go': 1927, 'buy': 604, 'cd': 676, 'much': 2916, 'money': 287
8, 'one': 3079, 'feel': 1671, 'worth': 4947, 'every': 1535, 'penny': 3224, 'amaze': 152, 'favorite': 1660, 'time': 4505, 'han
d': 2024, 'intense': 2337, 'sadness': 3807, 'prisoner': 3421, 'fate': 1655, 'mean': 2774, 'hope': 2153, 'distant': 1276, 'pro
```

We display the transformed data

```
# display vectorized data
print(Train_X_Tfidf)
```

```
  (0, 4976)     0.07110807215937723
  (0, 4936)     0.06389803853698107
  (0, 4783)     0.1628666078790877
  (0, 4709)     0.1954589799101139
  (0, 4281)     0.17010439938343686
  (0, 4172)     0.1437799944008138
  (0, 3843)     0.14442215592128319
  (0, 3793)     0.10069236026868997
  (0, 3685)     0.14647235541647163
  (0, 3667)     0.5432780222055527
  (0, 3451)     0.1628666078790877
  (0, 3442)     0.6118376364390653
  (0, 3434)     0.15338713136145987
  (0, 3079)     0.04776564298020434
```

We finally train the Naïve Bayers and Support Vector Machine algorithms.

```
# time now to apply Naives Bayers algorithim
# fit the training dataset on the NB classifier
Naive = naive_bayes.MultinomialNB()
Naive.fit(Train_X_Tfidf,Train_Y)
# predict the labels on validation dataset
predictions_NB = Naive.predict(Test_X_Tfidf)
# Use accuracy_score function to get the accuracy
print("Naive Bayes Accuracy Score is : ",accuracy_score(predictions_NB, Test_Y)*100)
```

```
Naive Bayes Accuracy Score is :  83.7
```

```
# apply support vector machine as well
# fit the training dataset on the classifier
SVM = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
SVM.fit(Train_X_Tfidf,Train_Y)
# predict the labels on validation dataset
predictions_SVM = SVM.predict(Test_X_Tfidf)
# Use accuracy_score function to get the accuracy
print("SVM Accuracy Score is : ",accuracy_score(predictions_SVM, Test_Y)*100)
```

```
SVM Accuracy Score is :  85.1
```

Load tweet file

Loading file downloaded from twitter API stream and look at the data

```
]: #          **********LOAD AND PRE-PROCESS SAMPLED LONDON MARATHON  STREAMED TWEETS***************
```

```
]: # load london marathon Tweets, I have used streamed sammple because the full list collected was slowing down the process.

tweets = api.user_timeline('LondonMarathon', count=200, tweet_mode='extended')
for t in tweets:
    print(t.full_text)
    print()
```

```
@Janderzzz Congratulations Jessica! 👊👏

@BethTonerRN Hey Beth, Thanks for your message. Just so we can respond adequately, in terms of this article, what is that w
e've done that has frustrated you? So you're aware we don't set the rules on what is a World Record and what isn't. Thanks, V
MLM

@AngelDavis2412 100% acceptable! 🙌

Congratulations to the Duke and Duchess of Sussex, Prince Harry and Meghan Markle on the arrival of their first child this mo
rning.

Prince Harry has been Patron of the London Marathon Charitable Trust since 2012 and we're sending our love! ❤️

#LondonMarathon #RoyalBaby https://t.co/SCdCsKCOMk

RT @INEOS159: The man who has run the fastest marathon of all time thinks he can go even faster. After Breaking2 and Berlin,
@EliudKipchoge…

RT @EliudKipchoge: Two years ago in Monza the world got to 26 seconds from breaking the last milestone in athletics. This fal
```

We put tweets in a list and visualize.

```
: # put tweets in list

def list_tweets(user_id, count, prt=False):
    tweets = api.user_timeline(
        "@" + user_id, count=count, tweet_mode='extended')
    tw = []
    for t in tweets:
        tw.append(t.full_text)
        if prt:
            print(t.full_text)
            print()
    return tw
```

```
: # viasualize tweets in list

user_id = 'LondonMarathon'
count=200
Tweet_RT = list_tweets(user_id, count)
print(Tweet_RT)
```

```
['@Janderzzz Congratulations Jessica! 👊👏', "@BethTonerRN Hey Beth, Thanks for your message. Just so we can respond adequate
ly, in terms of this article, what is that we've done that has frustrated you? So you're aware we don't set the rules on wha
t is a World Record and what isn't. Thanks, VMLM", '@AngelDavis2412 100% acceptable! 🙌', "Congratulations to the Duke and Du
chess of Sussex, Prince Harry and Meghan Markle on the arrival of their first child this morning. \n\nPrince Harry has been P
atron of the London Marathon Charitable Trust since 2012 and we're sending our love! ❤️\n\n#LondonMarathon #RoyalBaby http
s://t.co/SCdCsKCOMk", 'RT @INEOS159: The man who has run the fastest marathon of all time thinks he can go even faster. Afte
r Breaking2 and Berlin, @EliudKipchoge…', 'RT @EliudKipchoge: Two years ago in Monza the world got to 26 seconds from breakin
g the last milestone in athletics. This fall I want to br…', 'The last great barrier in modern athletics, and the greatest ma
rathon runner of all time.\n \nFresh from winning the Virgin Money London Marathon, @EliudKipchoge will attempt to break the
```

Analyse tweets by wordcloud

```
# analyse tweets by word cloud to Understand common words used in the tweets
def word_cloud(wd_list):
    stopwords = set('STOPWORDS')
    all_words = ' '.join([text for text in wd_list])
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        width=1600,
        height=800,
        random_state=21,
        colormap='jet',
        max_words=50,
        max_font_size=200).generate(all_words)
    plt.figure(figsize=(12, 10))
    plt.axis('off')
    plt.imshow(wordcloud, interpolation="bilinear");
```

Display wordcloud

Convert list to dataframe and visualize

```
# convert list to dataframe and visualise
df_marathon = pd.DataFrame(Tweet_RT)
df_marathon.head(10)
```

	0
0	@Janderzzz Congratulations Jessica! 👊👏
1	@BethTonerRN Hey Beth, Thanks for your message...
2	@AngelDavis2412 100% acceptable! 🙌
3	Congratulations to the Duke and Duchess of Sus...
4	RT @INEOS159: The man who has run the fastest ...
5	RT @EliudKipchoge: Two years ago in Monza the ...
6	The last great barrier in modern athletics, an...
7	How good were the crowds at Cutty Sark this ye...
8	One week ago today! 🙌👏\n\nSum up what finishin...
9	This moment...😍\n\n#LondonMarathon #ThanksaBil...

```
df_marathon.shape
```

```
(200, 1)
```

```
df_marathon.columns = ['Text']
```

```
df_marathon.columns
```

```
Index(['Text'], dtype='object')
```

Perform cleaning and display the results

```
# helper function to clean tweets
def processTweet(tweet):
    #Convert to lower case
    if type(tweet) is str:
        tweet = tweet.lower()
    #Convert www.* or https?://* to URL
    tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',str(tweet))
    #Convert @username to AT_USER
    tweet = re.sub('@[^\s]+','AT_USER',tweet)
    #Remove additional white spaces
    tweet = re.sub('[\s]+', ' ', tweet)
    #Replace #word with word
    tweet = re.sub(r'#([^\s]+)', r'\1', tweet)
    #trim
    tweet = tweet.strip('\'"')
    return tweet

# clean dataframe's text column
df_marathon['Text'] = df_marathon['Text'].apply(processTweet)
# preview some cleaned tweets
df_marathon['Text'].head(10)
```

```
0                AT_USER congratulations jessica! 👊👏
1    AT_USER hey beth, thanks for your message. jus...
2                        AT_USER 100% acceptable! 🙌
3    congratulations to the duke and duchess of sus...
4    rt AT_USER the man who has run the fastest mar...
5    rt AT_USER two years ago in monza the world go...
6    the last great barrier in modern athletics, an...
7    how good were the crowds at cutty sark this ye...
8    one week ago today! 🙌👏 sum up what finishing t...
9    this moment...😍 londonmarathon thanksabillion URL
Name: Text, dtype: object
```

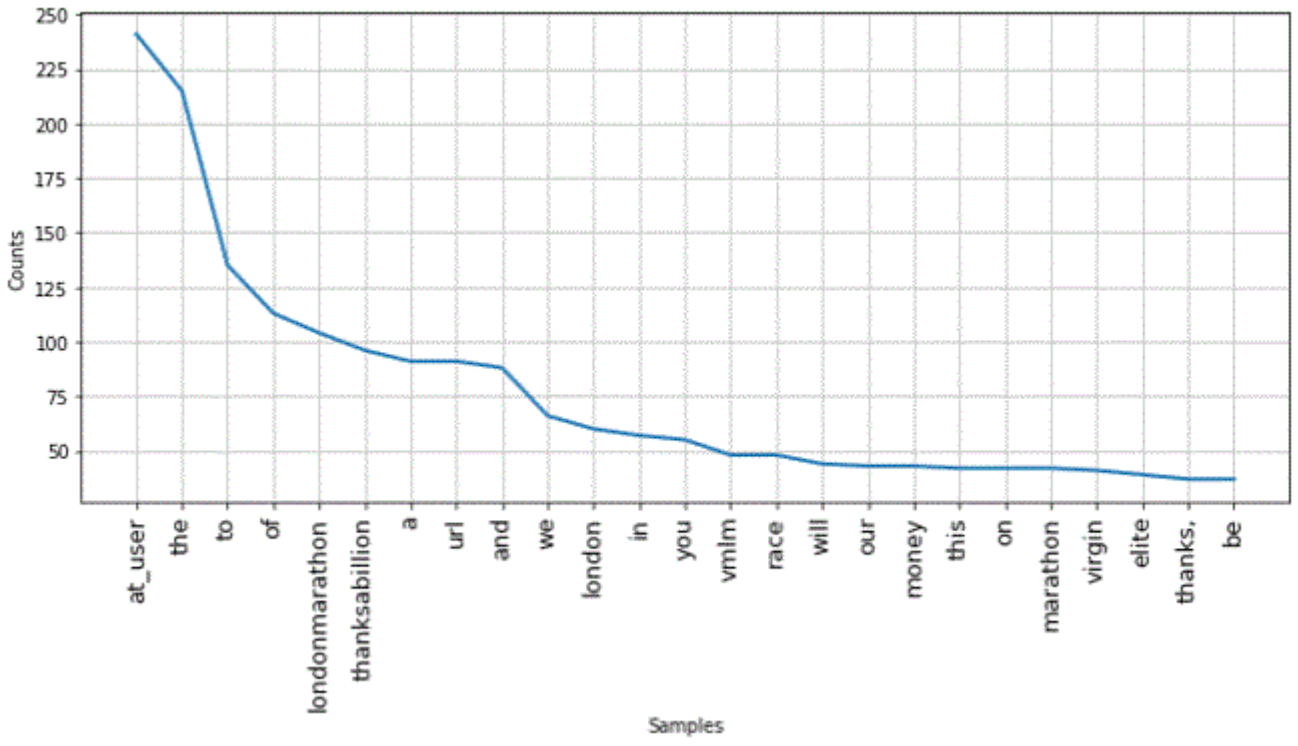
Word Frequency

Save file and create bag of words. Display word frequency and rank in graphs

```
# save tweets in file
df_marathon.to_csv('marathon/sampletweets.csv')
```

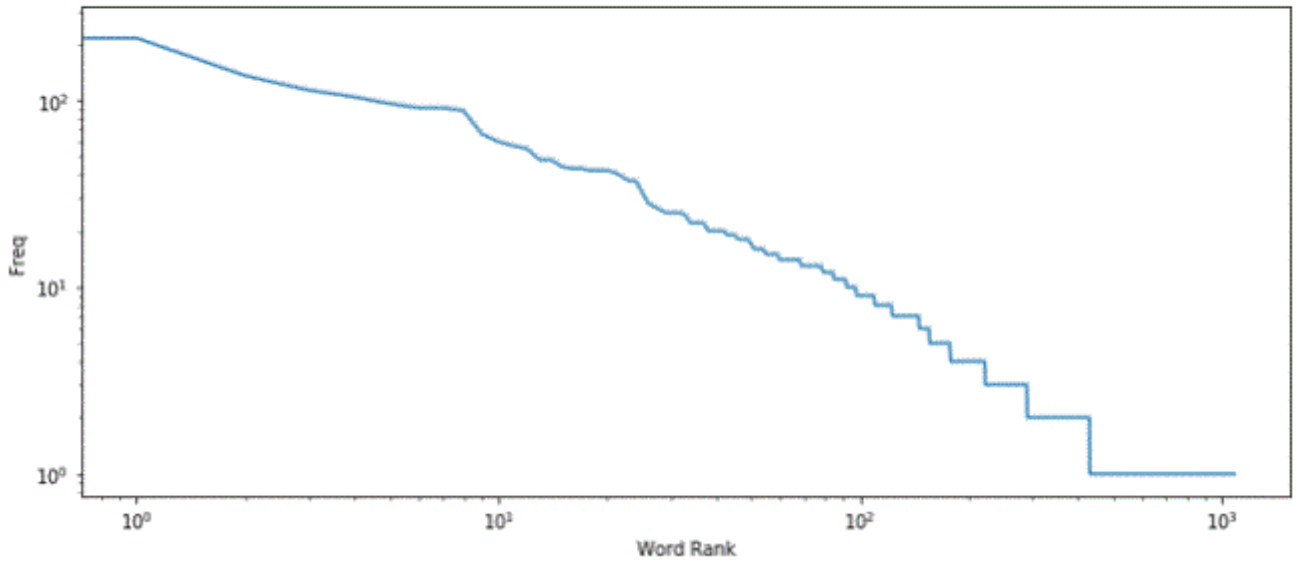
```
# most common words in twitter dataset
all_words = []
for line in list(df_marathon['Text']):
    words = line.split()
    for word in words:
        all_words.append(word.lower())
# plot word frequency distribution of first few words
plt.figure(figsize=(12,5))
plt.xticks(fontsize=13, rotation=90)
fd = nltk.FreqDist(all_words)
fd.plot(25,cumulative=False)
# log-log of all words
word_counts = sorted(Counter(all_words).values(), reverse=True)
plt.figure(figsize=(12,5))
plt.loglog(word_counts, linestyle='-', linewidth=1.5)
plt.ylabel("Freq")
plt.xlabel("Word Rank")
```

Word Frequency



WordRank

```
Text(0.5, 0, 'Word Rank')
```



Tokenize text file and view results

```
# tokenize helper function
def text_process(raw_text):

    # Check  punctuation
    nopunc = [char for char in list(raw_text) if char not in string.punctuation]
    # Join the characters again to form the string.
    nopunc = ''.join(nopunc)

    # remove any stopwords
    return [word for word in nopunc.lower().split() if word.lower() not in stopwords.words('english')]

def remove_words(word_list):
    remove = ['he','and','...','“','”','’','…','and’']
    return [w for w in word_list if w not in remove]

# tokenize message column and create a column for tokens
df_marathon = df_marathon.copy()
df_marathon['tokens'] = df_marathon['Text'].apply(text_process) # step 1
df_marathon['tokenized_tweet'] = df_marathon['tokens'].apply(remove_words) # step 2
df_marathon.head(10)
```

	Text	tokens	tokenized_tweet
0	AT_USER congratulations jessica! 👊👊	[atuser, congratulations, jessica, 👊👊]	[atuser, congratulations, jessica, 👊👊]
1	AT_USER hey beth, thanks for your message. jus...	[atuser, hey, beth, thanks, message, respond, ...	[atuser, hey, beth, thanks, message, respond, ...
2	AT_USER 100% acceptable! 🙌	[atuser, 100, acceptable, 🙌]	[atuser, 100, acceptable, 🙌]
3	congratulations to the duke and duchess of sus...	[congratulations, duke, duchess, sussex, princ...	[congratulations, duke, duchess, sussex, princ...
4	rt AT_USER the man who has run the fastest mar...	[rt, atuser, man, run, fastest, marathon, time...	[rt, atuser, man, run, fastest, marathon, time...
5	rt AT_USER two years ago in monza the world go...	[rt, atuser, two, years, ago, monza, world, go...	[rt, atuser, two, years, ago, monza, world, go...
6	the last great barrier in modern athletics, an...	[last, great, barrier, modern, athletics, grea...	[last, great, barrier, modern, athletics, grea...

Vectorize text

```
# vectorize
bow_transformer = CountVectorizer(analyzer=text_process).fit(df_marathon['tokenized_tweet'])
# print total number of vocab words
print(len(bow_transformer.vocabulary_))

# example of vectorized text
sample_tweet = df_marathon['tokenized_tweet'][16]
print(sample_tweet)
print('\n')
# vector representation
bow_sample = bow_transformer.transform([sample_tweet])
print(bow_sample)
print('\n')
```

```
198
['atuser', 'give', 'us', 'follow', 'well', 'send', 'dm', 'sophie', 'thanks', 'vmlm']


  (0, 42)       1
```

Transform text using TFIDF

```
7]: # transform the entire DataFrame of messages
    SM = bow_transformer.transform(df_marathon['Text'])
    # check out the bag-of-words counts for the entire corpus as a large sparse matrix
    print('Shape of Sparse Matrix: ', SM.shape)
    print('Amount of Non-Zero occurences: ', SM.nnz)
```

```
Shape of Sparse Matrix:  (200, 198)
Amount of Non-Zero occurences:  0
```

```
8]: SM.shape
```

```
8]: (200, 198)
```

```
9]: SM.data[0:5]
```

```
9]: array([], dtype=int64)
```

```
0]: #convert values obtained using the bag of words model into TFIDF values
    tfidfconverter = TfidfTransformer()
    SM = tfidfconverter.fit_transform(SM)

    SM.toarray()
```

```
0]: array([[0., 0., 0., ..., 0., 0., 0.],
           [0., 0., 0., ..., 0., 0., 0.],
           [0., 0., 0., ..., 0., 0., 0.],
           ...,
           [0., 0., 0., ..., 0., 0., 0.],
           [0., 0., 0., ..., 0., 0., 0.],
           [0., 0., 0., ..., 0., 0., 0.]])
```

Part 2: Social Network Graph

In [1]:
```
# code based on coursework tutorials and labs. Network graph code based on reasearch from networkx refrence manual.
#Release 2.4rc1.dev20190425110222.


# import required libraries
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler
import sys
import json
import pandas as pd
import re
import numpy as np
from datetime import datetime
import pandas as pd
import networkx as nx
import csv
from operator import itemgetter
import community
import matplotlib.pyplot as plt
from pandas import read_csv
```

In [2]:
```
# read from  file
tweetfile = 'marathon/londonmarathon2019.csv'
```

Prepare nodes and edges in pandas dataframe

:
```
# prepare nodes and edges from tweets to be used for network graph

fh = open(tweetfile, 'r')

userdata = pd.DataFrame(columns=('Id','Label','user_created_at','profile_image','followers_count','friends_count' ))
edges = pd.DataFrame(columns=('Source','Target','Strength'))

for line in fh:
    try:
        tweet = json.loads(line)
    except:
        continue
    if 'retweeted_status' not in tweet:
        continue

    userdata = userdata.append(pd.DataFrame([[tweet['user']['id_str'],
                                tweet['user']['screen_name'],
                                tweet['user']['created_at'],
                                tweet['user']['profile_image_url_https'],
                                tweet['user']['followers_count'],
                                tweet['user']['friends_count']]], columns=('Id','Label','user_created_at','profile_image','follo
    userdata = userdata.append(pd.DataFrame([[tweet['retweeted_status']['user']['id_str'],
                                tweet['retweeted_status']['user']['screen_name'],
                                tweet['retweeted_status']['user']['created_at'],
                                tweet['retweeted_status']['user']['profile_image_url_https'],
                                tweet['retweeted_status']['user']['followers_count'],
                                tweet['retweeted_status']['user']['friends_count']]], columns=('Id','Label','user_created_at','pr
    edges = edges.append(pd.DataFrame([[tweet['user']['id_str'],
                                tweet['retweeted_status']['user']['id_str'],
                                str(datetime.strptime(tweet['created_at'], '%a %b %d %H:%M:%S +0000 %Y'))]]
                                , columns=('Source','Target','Strength')), ignore_index=True)
```

```
# Network level is the number of times in total each of the tweeters responded to or mentioned the other.
# level 1 means all tweeters mentioned or replied to another at least once will be displayed.
# level 5 means only those who have mentioned or responded to a particular tweeter at least 5 times will be displayed,


strengthLevel = 3

edges2 = edges.groupby(['Source','Target'])['Strength'].count()
edges2 = edges2.reset_index()
edges2 = edges2[edges2['Strength'] >= strengthLevel]
```

```
# Export nodes from the edges and add node attributes for both Sources and Targets.
userdata = userdata.sort_values(['Id','followers_count'], ascending=[True, False])
userdata = userdata.drop_duplicates(['Id'], keep='first')

ids = edges2['Source'].append(edges2['Target']).to_frame()
ids.columns = ['Id']
ids = ids.drop_duplicates()

nodes = pd.merge(ids, userdata, on='Id', how='left')
```

```
nodes.columns = ['Id', 'Label', 'Date', 'Image', 'followers_count', 'friends_count']
edges2.columns = ['From','To','Strength']
```

Display top 10 nodes

[7]:
```
# Print nodes to check
nodes.head(10)
```

[7]:

	Id	Label	Date	Image	followers_count	friends_count
0	1024477392079859712	pineapplerissa	Wed Aug 01 02:10:18 +0000 2018	https://pbs.twimg.com/profile_images/111861983...	46	90
1	1044353084497944576	marathorunners	Mon Sep 24 22:29:13 +0000 2018	https://pbs.twimg.com/profile_images/108318932...	1534	3811
2	1110803839	Charly117tb	Tue Jan 22 06:46:40 +0000 2013	https://pbs.twimg.com/profile_images/106724275...	1202	2704
3	1262611021	RunningLife_	Tue Mar 12 18:35:51 +0000 2013	https://pbs.twimg.com/profile_images/108525993...	12633	2540
4	187079466	CharlesHenaine	Sun Sep 05 05:40:59 +0000 2010	https://pbs.twimg.com/profile_images/111655360...	700	1466
5	2888999729	Emi1yL	Sun Nov 23 09:50:30 +0000 2014	https://pbs.twimg.com/profile_images/111707942...	215	229
6	2889289549	ElegiCorrer	Tue Nov 04 02:20:46 +0000 2014	https://pbs.twimg.com/profile_images/106064879...	8909	1460
7	2943350389	arryanea	Fri Dec 26 00:53:26 +0000 2014	https://pbs.twimg.com/profile_images/103971337...	238	97
8	626213638	EugnieLe	Wed Jul 04 06:14:04 +0000 2012	https://pbs.twimg.com/profile_images/104213299...	2601	465
9	718239356083904513	Real_Infinity95	Fri Apr 08 00:49:41 +0000 2016	https://pbs.twimg.com/profile_images/110674107...	2003	2158

Display top 10 edges

:
```
# Print edges to check
edges2.head(5)
```

:

	From	To	Strength
102	1024477392079859712	126370504	9
185	1044353084497944576	54598980	4
488	1110803839	54598980	3
705	1262611021	54598980	7
1237	187079466	1262611021	3

Save nodes and edges to file in active directory

```
# Export nodes and edges to csv files
nodes.to_csv('marathon/Twitter_SNA/nodes.csv', encoding='utf-8', index=False)
edges2.to_csv('marathon/Twitter_SNA/edges.csv', encoding='utf-8', index=False)
```

Load and display number of nodes and edges

```
# open nodes
with open('marathon/Twitter_SNA/nodes.csv', 'r')as nodescsv:
    nonreader = csv.reader(nodescsv)
    nodes2 = [n for n in nonreader][1:] # get list of node names
node_names = [n[0] for n in nodes2]
# open edges
with open('marathon/Twitter_SNA/edges.csv', 'r') as edgecsv:
    edgereader = csv.reader(edgecsv)
    edges3 = [tuple(e) for e in edgereader][1:] # retrieve list of edges
```

```
# check nodes
print(len(node_names))
```

```
22
```

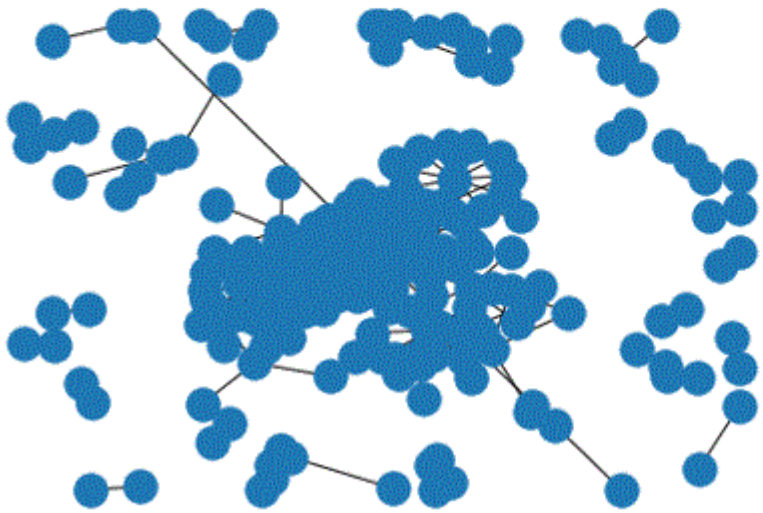
```
# check edgesS
print(len(edges3))
```

```
18
```

Draw graph.

```
# Add edges and edge attributes
my_graph = nx.Graph()
edgelist = pd.read_csv('marathon/Twitter_SNA/edges.csv')
for i, elrow in edgelist.iterrows():
    my_graph.add_edge(elrow[0], elrow[1], attr_dict=elrow[2:].to_dict())

    nx.draw(my_graph)
```



Get degree of centrality. This simply the number of connections a node has. This can signify popularity of the node. He highlighted node is more connected compared to the others.

```
# degree of centrality( finding most important/influential nodes)
nx.degree_centrality(my_graph)
```

```
{1024477392079859712: 0.047619047619047616,
 126370504: 0.047619047619047616,
 1044353084497944576: 0.047619047619047616,
 54598980: 0.6190476190476191,
 1110803839: 0.047619047619047616,
 1262611021: 0.09523809523809523,
 187079466: 0.09523809523809523,
 2888999729: 0.047619047619047616,
 2889289549: 0.047619047619047616,
 2943350389: 0.047619047619047616,
 626213638: 0.047619047619047616,
 718239356083904513: 0.047619047619047616,
 2097571: 0.047619047619047616,
 732561025921351681: 0.047619047619047616,
 38142380: 0.047619047619047616,
 761921737940406272: 0.047619047619047616,
 782284724152729601: 0.047619047619047616,
 906350459132227584: 0.047619047619047616,
 912652469154598912: 0.047619047619047616,
 25979455: 0.047619047619047616,
 967354083626639360: 0.047619047619047616,
 968571970026704896: 0.047619047619047616}
```

Eigenvector centrality is how a node is connected to more influential nodes. In our case 6 nodes with 4.79 score are more influential.

```
# Eigenvector Centrality(node importance increases as connecttion to other important nodes increase)
nx.eigenvector_centrality(my_graph)
```

```
{1024477392079859712: 4.790979861168125e-09,
 126370504: 4.790979861168125e-09,
 1044353084497944576: 0.18735039008682494,
 54598980: 0.6944361732081195,
 1110803839: 0.18735039008682494,
 1262611021: 0.2565702106807438,
 187079466: 0.2565702106807438,
 2888999729: 0.18735039008682494,
 2889289549: 0.18735039008682494,
 2943350389: 0.18735039008682494,
 626213638: 0.18735039008682494,
 718239356083904513: 4.790979861168125e-09,
 2097571: 4.790979861168125e-09,
 732561025921351681: 4.790979861168125e-09,
 38142380: 4.790979861168125e-09,
 761921737940406272: 0.18735039008682494,
 782284724152729601: 0.18735039008682494,
 906350459132227584: 0.18735039008682494,
 912652469154598912: 4.790979861168125e-09,
 25979455: 4.790979861168125e-09,
 967354083626639360: 0.18735039008682494,
 968571970026704896: 0.18735039008682494}
```

Betweenness centrality signifies the twitter node that is within the shortest path in the communication network. In this case highlighted node has the highest betweenness centrality.

```
In [20]: # betweeness ventrality(how many times anode appears in the shortest path, hence very influential in a group)
         nx.betweenness_centrality(my_graph)
```

```
Out[20]: {1024477392079859712: 0.0,
          126370504: 0.0,
          1044353084497944576: 0.0,
          54598980: 0.3666666666666667,
          1110803839: 0.0,
          1262611021: 0.0,
          187079466: 0.0,
          2888999729: 0.0,
          2889289549: 0.0,
          2943350389: 0.0,
          626213638: 0.0,
          718239356083904513: 0.0,
          2097571: 0.0,
          732561025921351681: 0.0,
          38142380: 0.0,
          761921737940406272: 0.0,
          782284724152729601: 0.0,
          906350459132227584: 0.0,
          912652469154598912: 0.0,
          25979455: 0.0,
          967354083626639360: 0.0,
          968571970026704896: 0.0}
```

References

Sandipan D. (2017) “Data science central” *Some Social Network Analysis with Python.* Data Science Central[Online] Available at: https://www.datasciencecentral.com/profiles/blogs/ (Accessed: 05/05/2019).

Bonzanini M (2016) ***Mastering social media mining with python,*** acquire and analyse data from all corners of the social web with Python. Packt Publishing

Russell M (2013) ***Mining the social web,*** Data mining facebook, twitter, linkedin, google, github and more. Second Edition O’Reilly Media Inc.

Sandipan D. (2017) “Machine Learning Mastery” *How to Prepare Text Data for Machine Learning with Scikit-learn.* Machine Learning Mastery [Online] Available at: https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/ (Accessed: 06/05/2019).