HW4 - Exploring Social Networks

Brenden Lewis
Due: 10/25/2020 11:59PM

Q1

The code for Because I find it easier to work with dictionaries in regards to storing data for individuals, I read in each 'friend' entry from *HW4-friend-count.csv* as a dictionary containing their name and number of friends, and stored them in an list which was then sorted before any calculations were made. I utilized the *statistics* package to calculate the mean, median, and standard deviation of the friend counts (*statistics* is used for the same calculations for the other questions).

```python
import csv
import matplotlib
import matplotlib.pyplot as plt
import statistics
import math
import pandas as pd
from operator import itemgetter

friends=[]  # {'USER': 'name', 'FRIENDCOUNT': 'count'}
sorted_friends=[]
User = 'U'
mean = 1
standard = 1
median = 1

def process():
    with open("HW4-friend-count.csv",'r') as f:
        count = 1
        for line in csv.DictReader(f):
            new_dict = {'USER':"f"+str(count),'FRIENDCOUNT':int(line['
FRIENDCOUNT'])}
            friends.append(new_dict)
            count+=1

        friends.append({'USER':User, 'FRIENDCOUNT': len(friends)})

        global sorted_friends
        sorted_friends = sorted(friends, key=itemgetter('FRIENDCOUNT'))

        global mean,standard,median
        mean = calcMean()
        standard = calcStandDevi()
        median = calcMedian()

def calcMean():
    friend_counts = []
    for user in friends:
        friend_counts.append(int(user['FRIENDCOUNT']))

    result = statistics.mean(friend_counts)

    return math.trunc(result)

def calcStandDevi():
    friend_counts = []
    for user in friends:
        friend_counts.append(int(user['FRIENDCOUNT']))

    result = statistics.pstdev(friend_counts)
    return round(result,2)

def calcMedian():
    friend_counts = []
    for user in friends:
        friend_counts.append(int(user['FRIENDCOUNT']))

    friend_counts.sort()
    mid = statistics.median(friend_counts)

    return math.trunc(mid)

def drawGraph():
    x=[]
    y=[]
    ux=[]
    uy=[]
    for per in sorted_friends:
        x.append(per['USER'])
        y.append(per['FRIENDCOUNT'])
        if(per['USER']==User):
            ux.append(per['USER'])
            uy.append(per['FRIENDCOUNT'])

    plt.title('Friendship Paradox: User "'+User+'"')
    plt.xlabel(User+' vs. their friends')
    plt.ylabel('# of friends')

    plt.scatter(x,y, label=User+"'s friends")
    plt.tick_params(axis='x', which='both',bottom=False, labelbottom=False)
    plt.scatter(ux,uy,color='yellow', label=User)
    plt.legend(loc="upper left")

    plt.show()






def writeResults():
    print("Statistics for 'Friend' counts of "+User+" and their friends")
    print("Mean: "+str(mean))
    print("Median: "+str(median))
    print("Standard Deviation: "+str(standard))

    with open("FriendStatistics.txt","w") as f:
        f.write("Statistics for 'Friend' counts of "+User+" and their friends"+'\n')
        f.write("Mean: "+str(mean)+'\n')
        f.write("Median: "+str(median)+'\n')
        f.write("Standard Deviation: "+str(standard))

if __name__ == '__main__':
    process()
    writeResults()
    drawGraph()
```

Listing 1: Python code Q1

Listing 2 shows the output of the calculations for Q1. I'm a little skeptical of the result for standard deviation as it seems a bit too high; this could be due to outliers in the data, or an error in calculating the standard deviation (either wrong methods or some other factor I'm missing).

```
Statistics for 'Friend' counts of U and their friends
Mean: 538
Median: 395
Standard Deviation: 535.81
```

Listing 2: Calculated mean, median, and standard deviation for Q1

Figure 1 shows a scatter plot of the User's (U) friend counts compared to their friends' friend counts. The results of graph seem to show that the Friendship Paradox holds for this particular user. The User had about 98 friends in their list. Compared to the majority of their friends, the User has a lot less friends, with some friends having thousands more friends to their account.

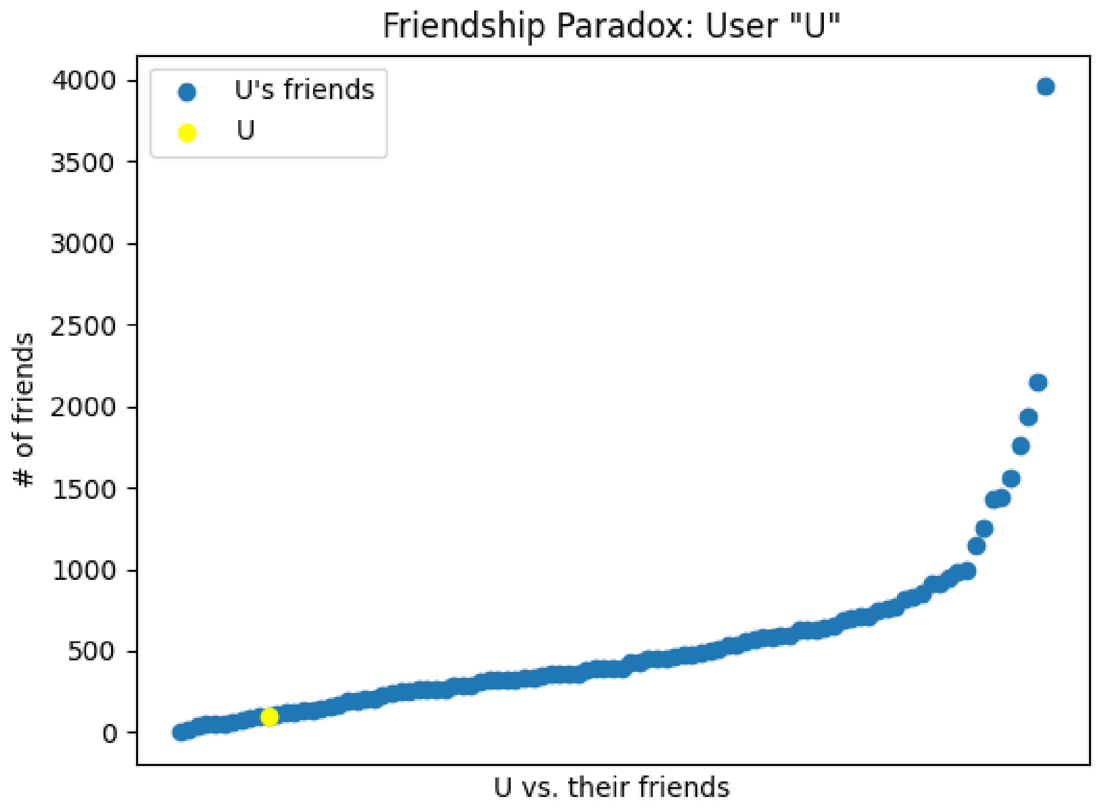

Figure 1: Graph showing the friend count of the user, and the friend counts of their friends

Q2

I opted to split this section into two programs: one to collect all the follower counts from Twitter (Listing 3), and one to handle data calculations and graphing (Listing 5). I don't have an active Twitter account, so I used our instructor's Twitter handle *'weiglemc'* as the user ID to collect data. The biggest challenge here was handling a RateLimitError that occured when the rate limit was exceeded pulling from Twitter. I got around this by adding the parameter *wait_rate_limit=True* to the *api* initialization. This caused the stream to halt when the rate limit was reached to give it time to refresh and prevent errors from attempting to exceed it.

A Cursor was used to move through the list of *weiglemc*'s followers. For each follower, a dictionary was created with their Twitter handle (ID) and their follower counts and stored in list which would then be sorted by follower counts after everyone was processed; follower counts was found using *'.followers_count'* for each user.

Of note: I ended up not using the getFollowerCount() method and forgot to remove it.

```
import matplotlib
import matplotlib.pyplot as plt
import statistics
import math
import pandas as pd
from operator import itemgetter
import tweepy
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream
import time

# Keys ommitted
consumer_key="***"
consumer_secret="***"
access_token="***"
access_secret="***"

# Handles authorization with Twitter
auth = OAuthHandler(consumer_key,consumer_secret)
auth.set_access_token(access_token,access_secret)
api = tweepy.API(auth, wait_on_rate_limit=True,
    wait_on_rate_limit_notify=True)
# api = tweepy.API(auth)

user_ID = 'weiglemc'
User=api.get_user(user_ID)

# STREAM_CAP = 433
STREAM_CAP = User.followers_count
```

```python
followers=[] # {'USER': ID, 'FOLLOWERCOUNT': count}
sortedFollowers=[]

def process():
    try:
        for follower in tweepy.Cursor(api.followers, user_ID).items(STREAM_CAP):
            follower_ID = follower.screen_name
            print("Processing "+follower_ID+"...")
            follower_count = follower.followers_count
            follower_dict = {'USER': follower_ID, 'FOLLOWERCOUNT': follower_count}
            followers.append(follower_dict)
            print("Finshed"+'\n')
    except tweepy.RateLimitError:
        print ("Rate Limit reached. Sleeping for 60s")
        time.sleep(60)

    followers.append({'USER': user_ID, 'FOLLOWERCOUNT': User.followers_count})

    global sortedFollowers
    sortedFollowers = sorted(followers, key=itemgetter('FOLLOWERCOUNT'))


def getFollowerCount(userName):
    c = tweepy.Cursor(api.followers, userName)

    count = 0
    for follower in c.items():
        count += 1

    return count

def writeOutput():
    for item in followers:
        print(str(item))

def writeToFile():
    with open("FollowerList.txt","w") as f:
        for item in sortedFollowers:
            f.write(str(item)+'\n')

if __name__=='__main__':
    process()
    writeOutput()
    writeToFile()
```

Listing 3: Python code Q2

Listing 4 below shows how the information for each user was stored in the output file:

```
{'USER': 'rachelheldevans', 'FOLLOWERCOUNT': 163355}
{'USER': 'KHayhoe', 'FOLLOWERCOUNT': 169495}
{'USER': 'mattcutts', 'FOLLOWERCOUNT': 529815}
```

Listing 4: Examples of stored dictionaries for followers

Listing 5 below is a slightly modified version of the code from Listing 1 to accomodate the data collected for Q2. The only major difference aside from file and graph nomenclature is how the information from the new file is read in. Listing 1 uses a method *csv.DictReader()* from the *csv* library to read in lines from csv files as dictionaries. Listing 5 (and also Listing 10 for Q3) uses *ast.literal_eval(line)* to convert each line from the file, as is, into a dictionary to be stored. The users were already sorted previously, so there was no need to sort again.

```
import csv
import matplotlib
import matplotlib.pyplot as plt
import statistics
import math
import pandas as pd
from operator import itemgetter
import ast

followers=[]  # {'USER': 'name', 'FRIENDCOUNT': 'count'}
User = 'weiglemc'
mean = 1
standard = 1
median = 1

def process():
    with open("FollowerList.txt",'r') as f:
        count = 1
        for line in f:
            new_dict = ast.literal_eval(line)
            followers.append(new_dict)
            count+=1

        global mean,standard,median
        mean = calcMean()
        standard = calcStandDevi()
        median = calcMedian()

def calcMean():
    friend_counts = []
```

```python
    for user in followers:
        friend_counts.append(int(user['FOLLOWERCOUNT']))

    result = statistics.mean(friend_counts)

    return math.trunc(result)

def calcStandDevi():
    friend_counts = []
    for user in followers:
        friend_counts.append(int(user['FOLLOWERCOUNT']))

    result = statistics.pstdev(friend_counts)
    return round(result,2)

def calcMedian():
    friend_counts = []
    for user in followers:
        friend_counts.append(int(user['FOLLOWERCOUNT']))

    friend_counts.sort()
    mid = statistics.median(friend_counts)

    return math.trunc(mid)

def drawGraph():
    x=[]
    y=[]
    ux=[]
    uy=[]
    for per in followers:
        x.append(per['USER'])
        y.append(per['FOLLOWERCOUNT'])
        if(per['USER']==User):
            ux.append(per['USER'])
            uy.append(per['FOLLOWERCOUNT'])

    plt.title('Friendship Paradox: User "'+User+'"')
    plt.xlabel(User+' vs. their followers')
    plt.ylabel('# of followers')

    plt.scatter(x,y, label=User+"'s followers")
    plt.tick_params(axis='x', which='both',bottom=False, labelbottom=
   False)
    plt.scatter(ux,uy,color='yellow', label=User)
    plt.legend(loc="upper left")

    plt.show()






def writeResults():
    print("Statistics for 'Follower' counts of "+User+" and their followers")
    print("Mean: "+str(mean))
    print("Median: "+str(median))
    print("Standard Deviation: "+str(standard))

    with open("FollowerStatisticsTrimmed.txt","w") as f:
        f.write("Statistics for 'Follower' counts of "+User+" and their followers"+'\n')
        f.write("Mean: "+str(mean)+'\n')
        f.write("Median: "+str(median)+'\n')
        f.write("Standard Deviation: "+str(standard))

if __name__ == '__main__':
    process()
    writeResults()
    drawGraph()
```

Listing 5: Python code for graphing follower counts

Listing 6 shows the output for calculations using data from follower counts. Listing 7 shows the output from the same data minus three outliers heavily skewing the data to the right. The errors related to standard deviation seem much more apparent in these listings compared to that in Q1, or once again the deviation is being affected by the larger follower counts skewing the data. It remains unclear.

```
Statistics for 'Follower' counts of weiglemc and their followers
Mean: 2915
Median: 195
Standard Deviation: 27839.99
```

Listing 6: Calculated mean, median, and standard deviation for Q2

```
Statistics for 'Follower' counts of weiglemc and their followers
Mean: 945
Median: 393
Standard Deviation: 2244.33
```

Listing 7: Calculated mean, median, and standard deviation for Q2, excluding a few outliers

Figure 2 shows the original data of the user *weiglemc*'s followers, and their followers' followers. To help provide a better visual curve, the three outliers were removed and the data was graphed

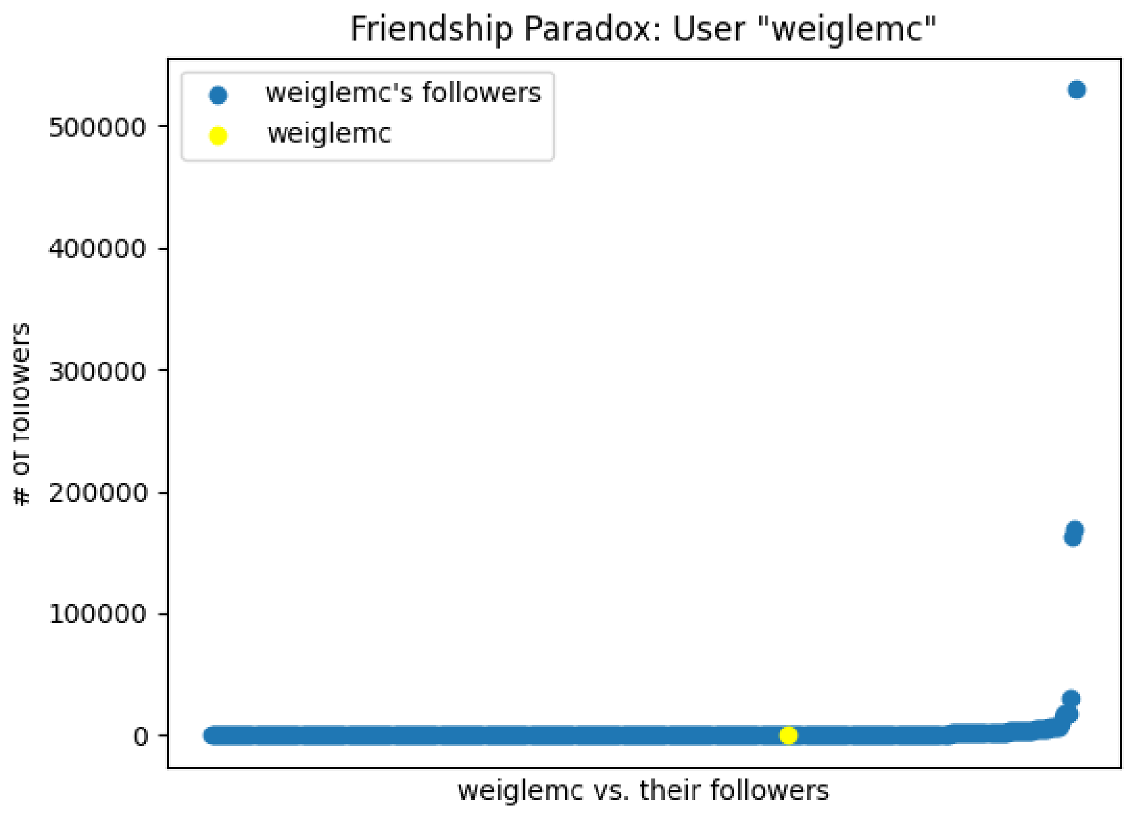

Figure 2: Graph showing the follower count of the user, and the follower counts of their followers

again in Figure 3. It is difficult to tell based on both graphs if the Friendship Paradox holds as *weiglemc* is just a little right of the middle of both curves; this would indicate that *weiglemc* has more followers than more than half of the users they follow.

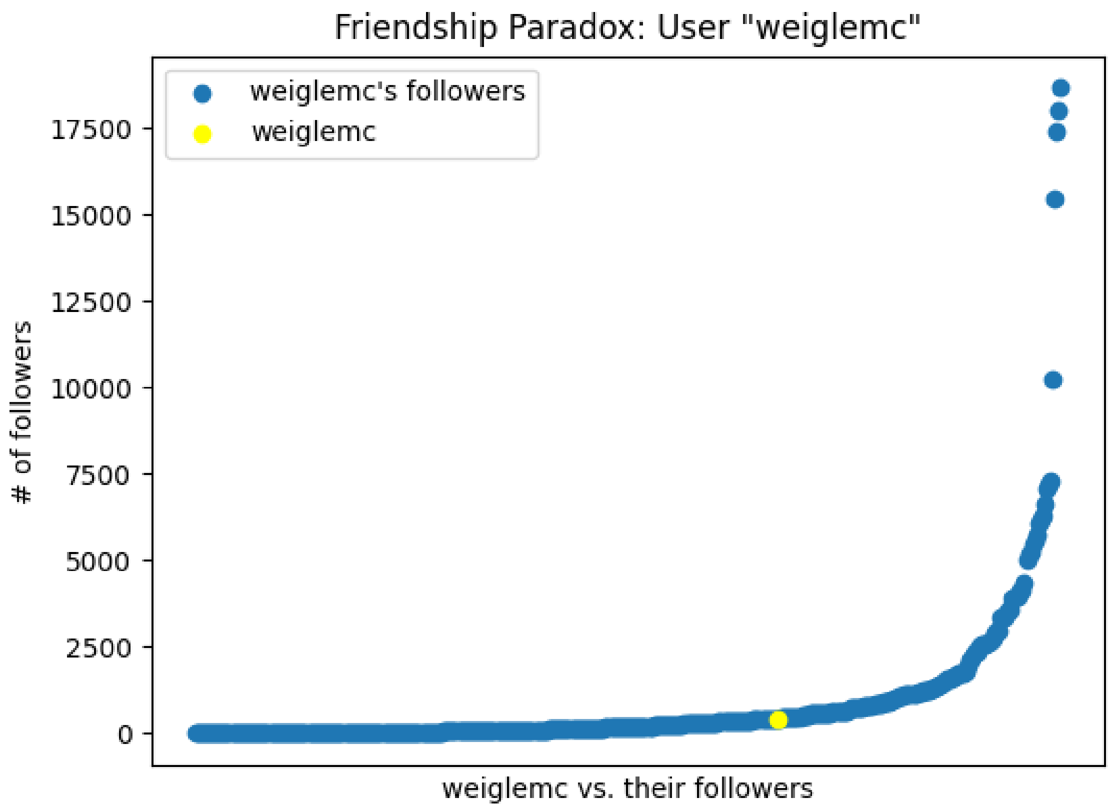

Figure 3: Graph showing the follower count of the user, and the follower counts of their followers, excluding outliers

Q3 (Extra)

The code used for Q3 (Listing 8 and 10) is very much the same to that used in Q2 with a few minor changes. In order to get *following* counts from a user instead of *followers*, I had to alter the Cursor to use *api.friends* and call *.friends_count* for each user to get the number of users they each follow. For some reason, the Twitter API uses Friend interchangeably with Follower when doing anything related to who a user follows. When collecting data, I had to double check a few Twitter accounts to ensure it was actually capturing the number of users they follow.

```python
import matplotlib
import matplotlib.pyplot as plt
import statistics
import math
import pandas as pd
from operator import itemgetter
import tweepy
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream
import time

# Keys ommitted
consumer_key="***"
consumer_secret="***"
access_token="***"
access_secret="***"

# Handles authorization with Twitter
auth = OAuthHandler(consumer_key,consumer_secret)
auth.set_access_token(access_token,access_secret)
api = tweepy.API(auth, wait_on_rate_limit=True,
   wait_on_rate_limit_notify=True)
# api = tweepy.API(auth)

user_ID = 'weiglemc'
User=api.get_user(user_ID)

# STREAM_CAP = 5
STREAM_CAP = User.friends_count
friend_list=[] # {'USER': ID, 'FRIENDCOUNT': count}
sortedFriends=[]

def process():
    try:
        for follower in tweepy.Cursor(api.friends, user_ID).items(
   STREAM_CAP):
            friend_ID = follower.screen_name
            print("Processing "+friend_ID+"...")
            friend_count = follower.friends_count
            friend_dict = {'USER': friend_ID, 'FRIENDCOUNT':
 friend_count}
            friend_list.append(friend_dict)
            print("Finshed"+'\n')
    except tweepy.RateLimitError:
        print ("Rate Limit reached. Sleeping for 60s")
        time.sleep(60)

    friend_list.append({'USER': user_ID, 'FRIENDCOUNT': STREAM_CAP})

    global sortedFriends
    sortedFriends = sorted(friend_list, key=itemgetter('FRIENDCOUNT'))


def getFollowerCount(userName):
    c = tweepy.Cursor(api.followers, userName)

    count = 0
    for follower in c.items():
        count += 1

    return count

def writeOutput():
    for item in sortedFriends:
        print(str(item))

def writeToFile():
    with open("FriendsList.txt","w") as f:
        for item in sortedFriends:
            f.write(str(item)+'\n')

if __name__=='__main__':
    process()
    writeOutput()
    writeToFile()
```

Listing 8: Python code Q3

Listing 9 shows how the data gathered was stored in the output file before any calculations or graphing is done.

```
{'USER': 'nakedpastor', 'FRIENDCOUNT': 17790}
{'USER': 'MarkWarner', 'FRIENDCOUNT': 25114}
{'USER': 'BarackObama', 'FRIENDCOUNT': 599538}
```

Listing 9: Examples of stored dictionaries for accounts followed

```python
import csv
import matplotlib
import matplotlib.pyplot as plt
import statistics
import math
import pandas as pd
from operator import itemgetter
import ast

followers=[]  # {'USER': 'name', 'FRIENDCOUNT': 'count'}
User = 'weiglemc'
mean = 1
standard = 1
median = 1

def process():
    with open("FriendsList.txt",'r') as f:
        count = 1
        for line in f:
            new_dict = ast.literal_eval(line)
            followers.append(new_dict)
            count+=1

        global mean,standard,median
        mean = calcMean()
        standard = calcStandDevi()
        median = calcMedian()

def calcMean():
    friend_counts = []
    for user in followers:
        friend_counts.append(int(user['FRIENDCOUNT']))

    result = statistics.mean(friend_counts)

    return math.trunc(result)

def calcStandDevi():
    friend_counts = []
    for user in followers:
        friend_counts.append(int(user['FRIENDCOUNT']))

    result = statistics.pstdev(friend_counts)
    return round(result,2)

def calcMedian():
    friend_counts = []
    for user in followers:
        friend_counts.append(int(user[’FRIENDCOUNT’]))

    friend_counts.sort()
    mid = statistics.median(friend_counts)

    return math.trunc(mid)

def drawGraph():
    x=[]
    y=[]
    ux=[]
    uy=[]
    for per in followers:
        x.append(per[’USER’])
        y.append(per[’FRIENDCOUNT’])
        if(per[’USER’]==User):
            ux.append(per[’USER’])
            uy.append(per[’FRIENDCOUNT’])

    plt.title(’Friendship Paradox: User "’+User+’"’)
    plt.xlabel(User+’ vs. who they follow’)
    plt.ylabel(’# following’)

    plt.scatter(x,y, label=User+"’s # following")
    plt.tick_params(axis=’x’, which=’both’,bottom=False, labelbottom=False)
    plt.scatter(ux,uy,color=’yellow’, label=User)
    plt.legend(loc="upper left")

    plt.show()






def writeResults():
    print("Statistics for ’Follower’ counts of "+User+" and their followers")
    print("Mean: "+str(mean))
    print("Median: "+str(median))
    print("Standard Deviation: "+str(standard))

    with open("FollowingStatistics.txt","w") as f:
        f.write("Statistics for ’Following’ counts of "+User+" and who they follow"+’\n’)
        f.write("Mean: "+str(mean)+'\n')
        f.write("Median: "+str(median)+'\n')
        f.write("Standard Deviation: "+str(standard))

if __name__ == '__main__':
    process()
    writeResults()
    drawGraph()
```

Listing 10: Python code for graphing following counts

Listing 11 shows the results of calculations of data collected for Q3. Listing 12 shows the same data exluding a single outlier that was *very* heavily skewing the data, the account of late President Barack Obama with nearly six-hundred thousand accounts followed.

```
Statistics for 'Following' counts of weiglemc and who they follow
Mean: 3238
Median: 394
Standard Deviation: 37048.7
```

Listing 11: Calculated mean, median, and standard deviation for Q3

```
Statistics for 'Following' counts of weiglemc and who they follow
Mean: 859
Median: 192
Standard Deviation: 2094.13
```

Listing 12: Calculated mean, median, and standard deviation for Q3, excluding an outlier

Figures 4 and 5 show scatter plots of the data for Q3, with Figure 5 excluding Obama's account to get a more accurate visual. In the case of *weiglemc*, just like in Q2, it is difficult to tell if the Friendship Paradox holds for counts of users followed as they are placed very near the center of the curves. User *weiglemc* veers slightly to the left of the median, meaning they are following less users than more than 50% of the users they follow.

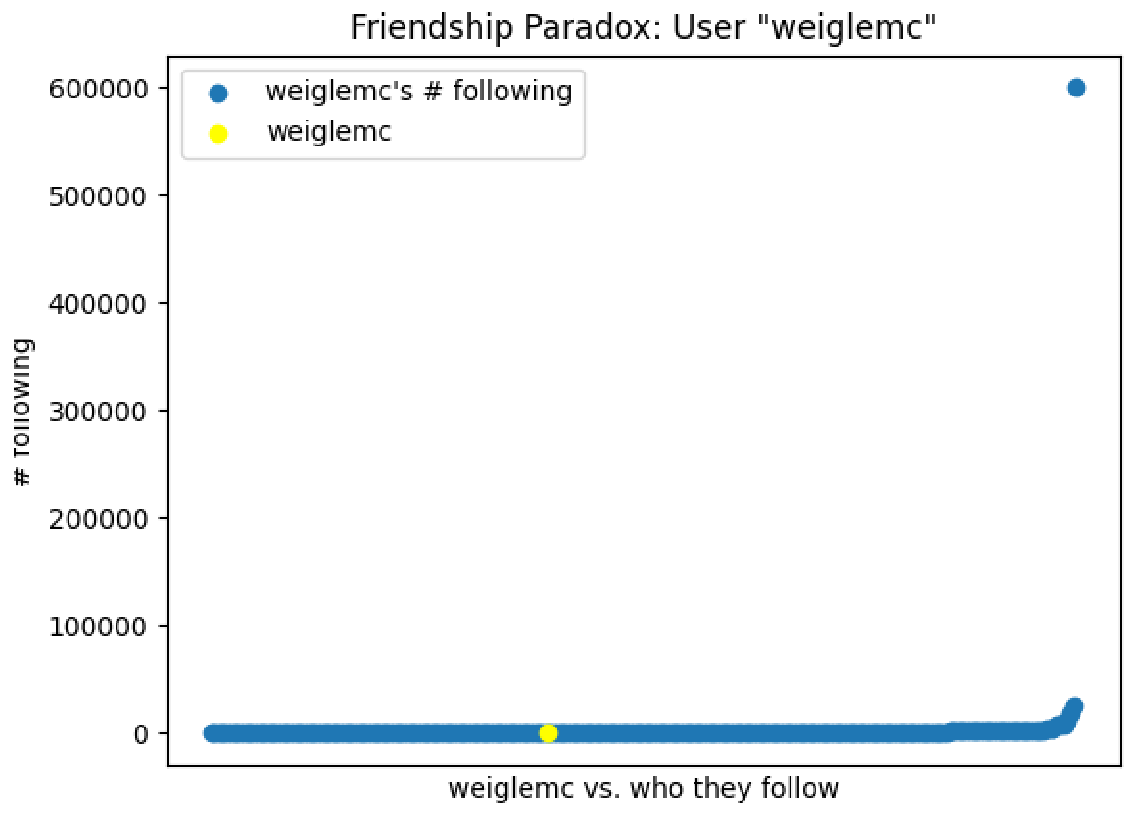

Figure 4: Graph showing the number of accounts the user follows, and the following counts of those they follow

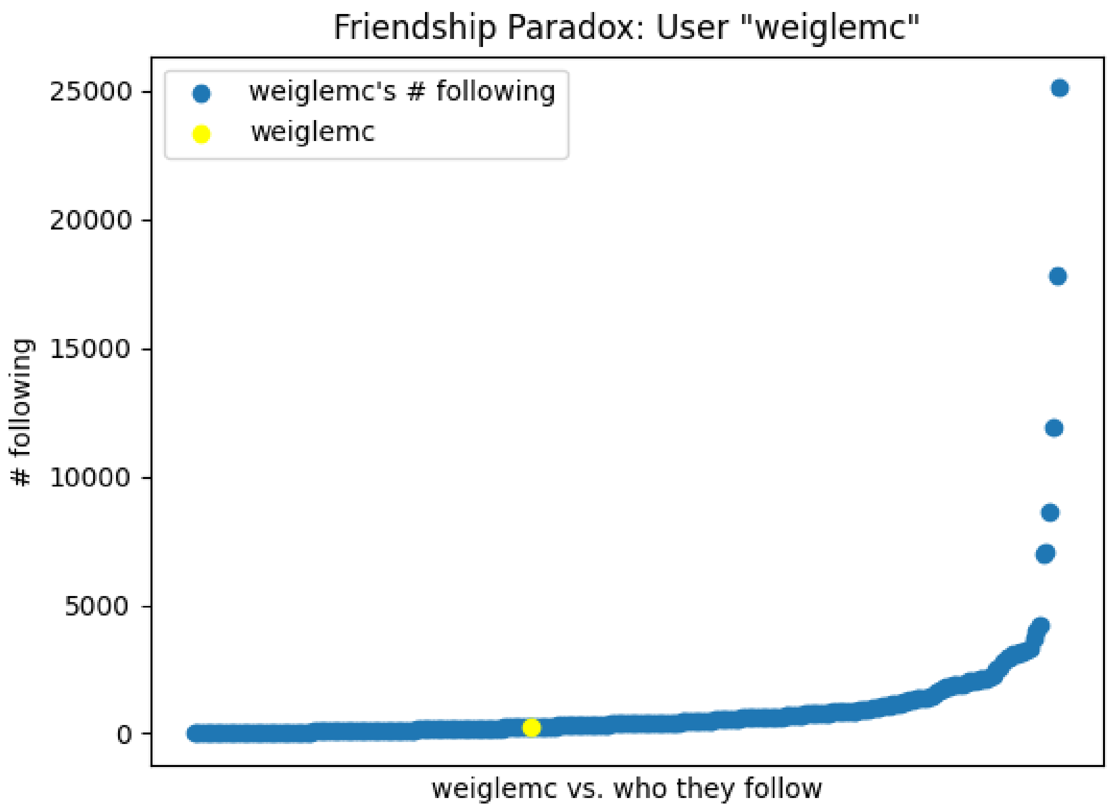

Figure 5: Graph showing the number of accounts the user follows, and the following counts of those they follow, excluding outliers