2017

Assignment 4

ROSS REELACHART

PROFESSOR MICHAEL L. NELSON

Question 1

Answering this question was made a lot simpler because the raw data was already provided to me (mln.graphml). Looking at it as an XML file, I wrote a python program (getFriendListing.py) that parsed the file, found the necessary values within it, and then printed those values to a CSV file (friendListing.csv). When viewed in Microsoft Excel, it cleanly listed out the findings in two columns: “Name” and “Friend Count.” The second row of the sheet simply lists the original owner (PROF MLN) and their friend count. Then all following rows display a friend of PROF MLN and their associated friend count.

```
#!.usr/local/bin/python2.7

import sys
from xml.dom.minidom import parseString

def getInfo(xml):

    dom = parseString(xml)
    countDict = {}

    for element in dom.getElementsByTagName("data"):

        if(element.attributes['key'].value == 'name'):
            name = element.childNodes[0].data

        if(element.attributes['key'].value == 'friend_count'):
            count = element.childNodes[0].data

            countDict[name] = count
            name = ''
            count = ''

    return countDict

def getCount(xml):

    dom = parseString(xml)
    return len(dom.getElementsByTagName("node"))

if __name__ == "__main__":

    friendDataFile = sys.argv[1]

    f = open(friendDataFile)
    xml = f.read()
    f.close()

    friendCount = getCount(xml)
    friendInfo = getInfo(xml)

    print("Name, Friend Count")
    print('PROF MLN, ' + str(friendCount))

    for friend in friendInfo:
        print(friend + ', ' + friendInfo[friend])
```

Figure 1 The code for getFriendListing.py

Now that I had a CSV file with all the necessary data neatly arranged, I needed to process it in R to make a nice graph and perform some basic statistical analysis. For this task, I wrote an R script (processFriendListing.r) that could be run from the command line, taking as arguments 1.) The name of CSV file, 2.) The name of the graph it would produce, 3.) The length of the list as an integer, and 4.) a short text string that would be used to indicate the position of PROF MLN on that same graph. This R script would also print to the command line the mean, median and standard deviation of the data.

```
rreelac@sirius:~/assignment 4$
rreelac@sirius:~/assignment 4$ Rscript processFriendListing.r friendListing.csv friendListingGraph.png 156 'PROF_MLN'
Mean:  357.735483870968
Median:  259
Std Dev:  370.704490379149
null device
          1
rreelac@sirius:~/assignment 4$
```

Figure 2 The statistical analysis of the data for Question 1.

This seemed to work just fine initially. However, the graph this script produced was not completely what I wanted. It lacked the color and position indicator of PROF MLN. At first, I moved on from this minor hiccup and completed the rest of the assignment. But I wanted that indicator and color. So I went back and redid the script to be used specifically in RStudio (processFriendListing2.r), and ended up with a much more satisfactory graph. This script re-writing would also be performed on the R script in question 2.

```
#!/usr/bin/Rscript

friendListing <- read_csv("C:/Users/Rossv2/Desktop/assignment 4/friendListing.csv")
outputGraph <- "friendListingGraph.pdf"
listLength <- 155
indicator <- "PROF_MLN"

sortedData <- sort(friendListing$'Friend Count')

meanText <- paste("Mean: ", mean(sortedData), collapse = "")

medianText <- paste("Median: ", median(sortedData), collapse = "")

sdText <- paste("Std Dev: ", sd(sortedData), collapse = "")

write(meanText, stdout())
write(medianText, stdout())
write(sdText, stdout())

pdf(outputGraph)

pos <- (sortedData == listLength)
cols <- c("white", "red")

barplot(sortedData, main="Friends of MLN's Friends on Facebook", xlab="Friends by increasing
number of friends", ylab="Number of friends", col=cols[pos + 1], ylim=c(0, max(sortedData) + 100))

text(x=match(c(listLength), sortedData) + 8, y=max(sortedData), labels=indicator, col='red')

arrows(x0=match(c(listLength), sortedData) + 8, y0=(max(sortedData) - 50), x1=match(c(listLength)
, sortedData) + 8, y1=175, length=0.1, lwd=3, col='red')

dev.off()
```

Figure 3 The updated R script that was used in RStudio, processFriendListing.r

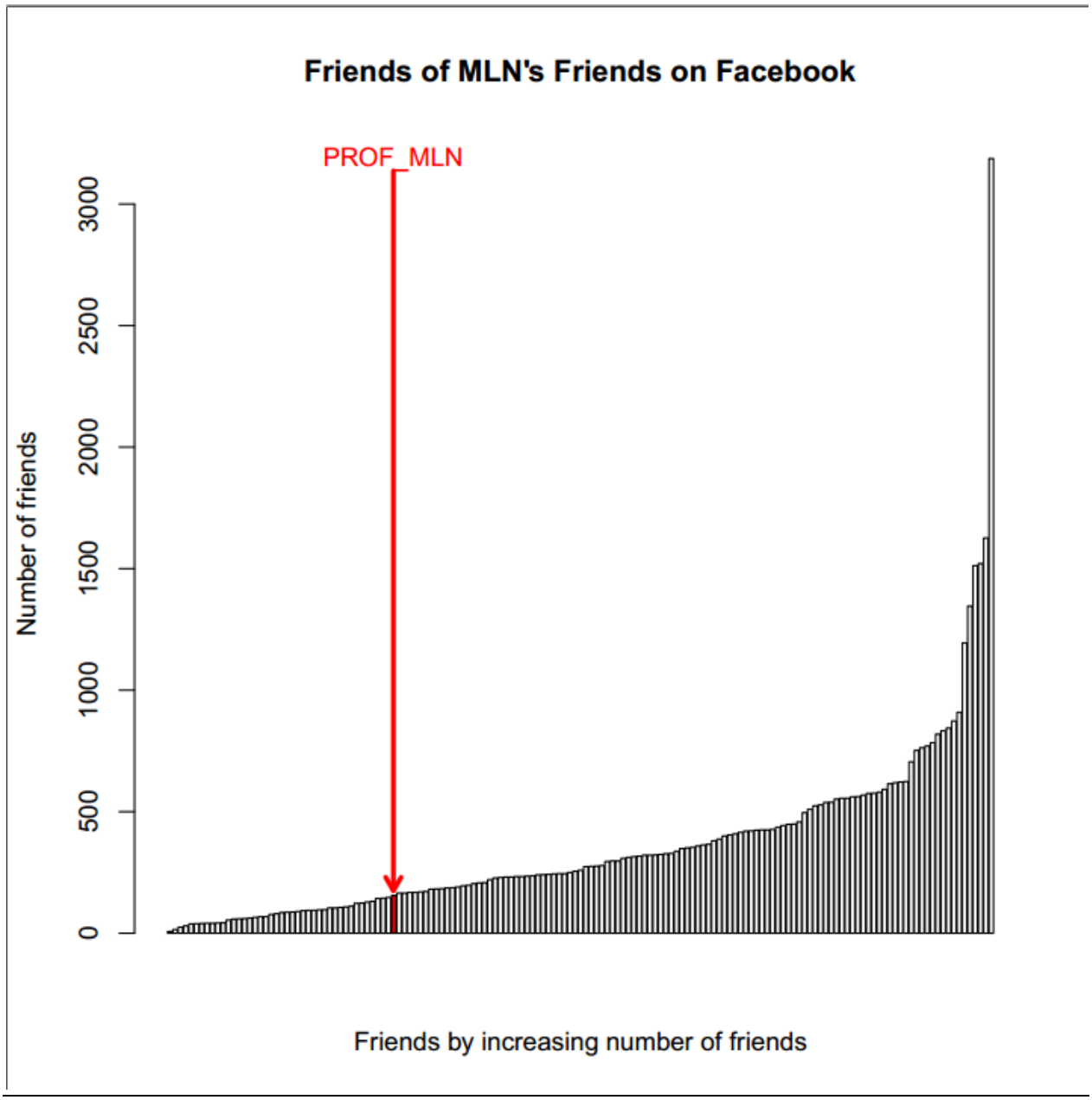

Figure 4 The final colored and labeled graph displaying PROF MLN's position relative to the number of friends his friends on Facebook have.

Judging by the fact the PROF MLN basically marks the end of the first quarter of all the data, it would seem that the "Friendship Paradox" holds true. The large majority of PROF MLN's friends have more friends than he does.

Question 2

The first portion of question proved to be more difficult initially. For this question, I decided to pull from the provided twitter account “phonedude_mln” because I believed it would create more consistent numbers.

I decided that pulling from Twitter would be made much easier by repurposing most of the code from assignment 2 (getTweets.py). The large majority of the resultant code for this assignment (getTwitterFollowers.py) is exactly the same as it was in assignment 2, specifically the code that queries and authenticates the Twitter API. Instead of pulling tweets, those functions were replaced by two new functions that printed the two columns to a CSV file (mlnTwitterFollowers.csv). These two columns arrange data similar to the listing in question 1. Writing these new functions were easier to write because of the experience of writing the original code for assignment 2.

```
def call_api_for_followers(oauth, count, screenName, cursor):
    url = "https://api.twitter.com/1.1/followers/list.json?screen_name=" + screenName + "&count=" + str(count) + "&cursor=" + cursor

    response = requests.get(url, auth=oauth)

    if 'errors' in response:
        raise TwitterError(json.dumps(response.json(), sort_keys=True,indent=4, separators=(',', ': ')))

    return response

def print_friend_counts(oauth, numberCalls, count, screenName):
    cursor="-1"

    print("Name, Friend Count")

    followers_count = 0

    for i in range(0, numberCalls):
        response = call_api_for_followers(oauth, count, screenName, cursor)

        for entry in response.json()['users']:
            ident = str(entry['screen_name'])
            followers = str(entry['followers_count'])
            print(ident + ', ' + followers)
            followers_count += 1

        print(screenName + ', ' + str(followers_count))

def usage():
    print("Usage: " + sys.argv[0] + "<apiCalls><screenName>")

if __name__ == "__main__":
    try:
        apiCalls = int(sys.argv[1])
        screenName = sys.argv[2]
    except IndexError as e:
        usage()
        sys.exit(1)

    if not OAUTH_TOKEN:
        token, secret = setup_oauth()
        print("OAUTH_TOKEN: " + token)
        print("OAUTH_TOKEN_SECRET: " + secret)
        print( )
    else:
        oauth = get_oauth()
        count = 200
```

Figure 5 The added code in getTwitterFollowers.py that differentiates it from getTweets.py. The rest of the code (not shown) is exactly the same as getTweets.py.

There are two columns: “Name” and “Friend Count.*” Again, the names of accounts phonedude_mln follows is listed, along with the number of accounts they follow. Phonedude_mln and the number of accounts he follows is actually listed last.

*It should’ve been called “Follower Count,” but I didn’t realize that minor distinction until I was completely done with the assignment. The code still works as intended despite the semantics.

Once more, this CSV file needed to be processed into a graph and for basic statistical analysis. The R script used to do so (processTwitterFollowers.r) is almost exactly the same as processFriendListing.r, and works almost exactly the same. And again, it was re-qritten slightly to work within RStudio to produce a more desirable graph (processTwitterFollower2.r). Also, a new line was added that removed duplicate data from my CSV file. Even after removing these duplicates, the statistical numbers remained the same and only the graph was slightly effected in terms of size.

```
rreelac@sirius:~/assignment 4$ Rscript processTwitterFollowers.r mlnTwitterFollowers.csv mlnTwitterFollowersGraph.pdf 401 'PROF_MLN'

Mean:  2588.30845771144
Median:  326
Std Dev:  17376.7468441927
null device
          1
```

Figure 6 The statistical data for Question 2

[Continued on next page.]

```
mlnTwitterFollowers <- read_csv("C:/Users/Rossv2/Desktop/assignment 4/mlnTwitterFollowers.csv")
dedupedTwitterFollowers <- unique(mlnTwitterFollowers[ ,1:2])
outputGraph <- "mlnTwitterFollowers10.pdf"
listLength <- 202
indicator <- "PROF_MLN"

sortedData <- sort(dedupedTwitterFollowers$`Friend Count`)

meanText <- paste("Mean: ", mean(sortedData), collapse = "")

medianText <- paste("Median: ", median(sortedData), collapse = "")

sdText <- paste("Std Dev: ", sd(sortedData), collapse = "")

write(meanText, stdout())
write(medianText, stdout())
write(sdText, stdout())

pdf(outputGraph)

pos <- (sortedData == listLength)
cols <- c("white", "red")

barplot(sortedData, main="Followers of MLN's Followers on Twitter", xlab="Followers by
increasing number of followers", ylab="Number of followers", col=cols[pos + 1], ylim=c(0, max(
sortedData) + 100))

text(x=match(c(listLength), sortedData) + 20, y=max(sortedData), labels=indicator, col='red')

arrows(x0=match(c(listLength), sortedData) + 20, y0=(max(sortedData)), x1=match(c(listLength),
sortedData) + 20, y1=300, length=0.1, lwd=3, col='red')

dev.off()
```

Figure 7 The updated code used to process the data in question 2, processTwitterFollower2.r. Note the second line, which removes duplicate data items.

[Continued on next page.]

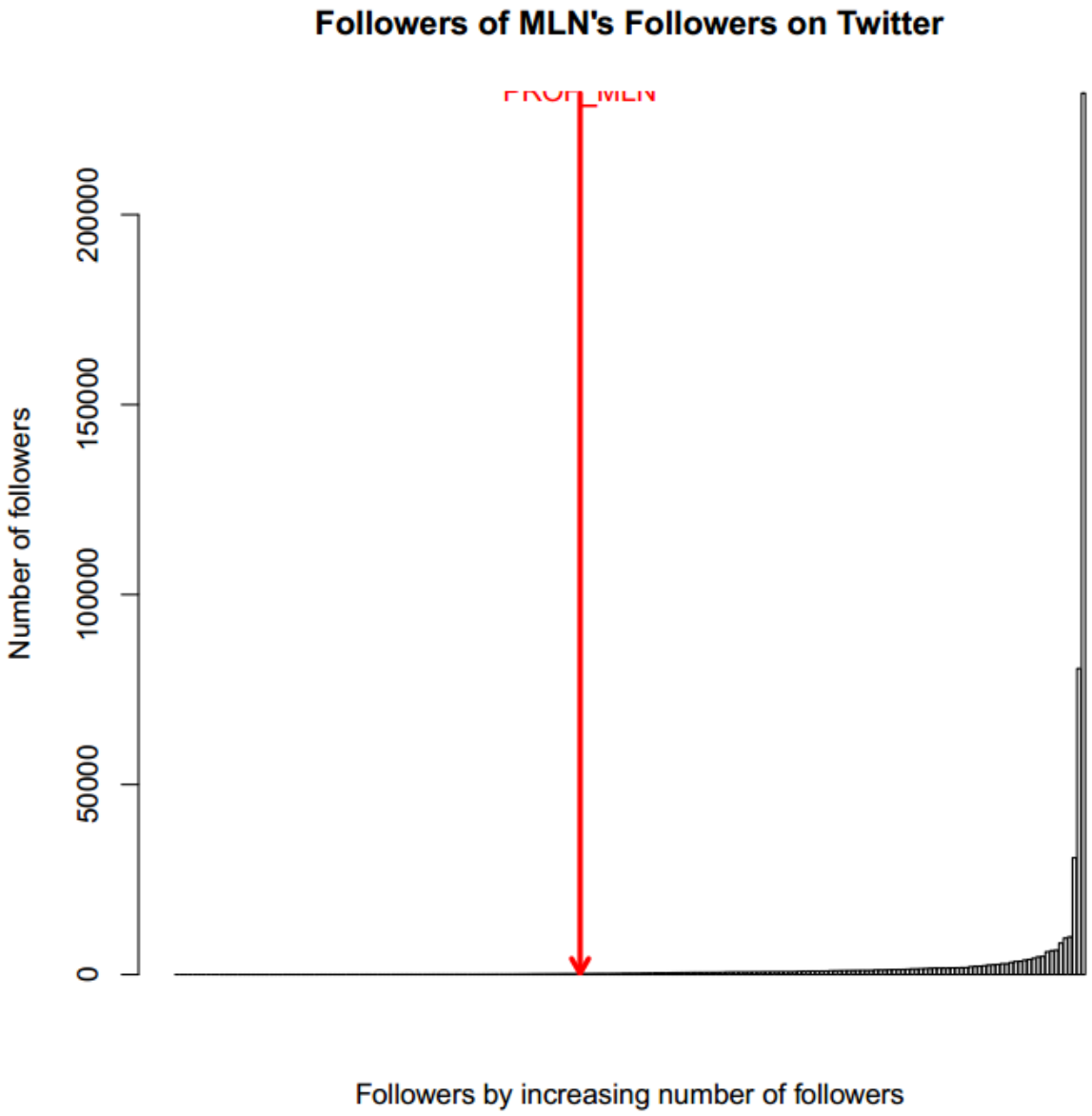

Figure 8 The final graph showing the position of phonedude_mln relative to the number of followers of his followers on Twitter.

It would also seem that the "Friendship Paradox" holds true for Twitter accounts too, though to a slightly lesser extent. Phonedude_mln sits closer to the middle of the graph, so there's a better chance that a follower chosen at random might have LESS followers than him.