Assignment 2

Miranda Smith

Python Program that extracts 1000 unique links from Twitter

I created multiple programs to tackle each step of this problem.

The file gatherURIs.py is used to create a twitter stream and collect links.

```
#http://socialmedia-class.org/twittertutorial.html

# Import the necessary methods from "twitter" library
from twitter import Twitter, OAuth, TwitterHTTPError, TwitterStream
import urllib.request
from urllib.parse import urlparse

file = open('urlList', 'a') #a = append

# Variables that contains the user credentials to access Twitter API
ACCESS_TOKEN = '824794440409706496-CGUoqhcZpfk4lOcblNVG1OuubwaCqZp'
ACCESS_SECRET = 'Mgd7wctkwI3nNUL6RHD4Cqjg9HVngccRrynoWfEsTUzlJ'
CONSUMER_KEY = '72haUMMgwOXNNGXXz4G0N7B9m'
CONSUMER_SECRET = 'zSx800UtH7OUTkVUMw2uORB39o1aP59uZgmXMWo8AZpoFkNgJQ'

oauth = OAuth(ACCESS_TOKEN, ACCESS_SECRET, CONSUMER_KEY, CONSUMER_SECRET)

# Initiate the connection to Twitter Streaming API
twitter_stream = TwitterStream(auth=oauth)

# Get a sample of the public data following through Twitter
iterator = twitter_stream.statuses.sample()


tweet_count = 9000
for tweet in iterator:
    if 'entities' in tweet:
        entities = tweet['entities']
        for url in entities['urls']:
            try:
                file.write(url['expanded_url'])
                file.write("\n")
                tweet_count = tweet_count - 1
                print(tweet_count)
            except:
                pass
            if tweet_count <= 0:
                break
        if tweet_count <=0:
            break
file.close()
```

I decided against the use of tweepy and went with twitter instead when I couldn't get the example code from the tutorial to authorize properly. I also got 9000 links to account for duplicates and others that would need to be thrown out. For each tweet I extracted the expanded url and wrote it into a file. Most of the code came from a tutorial on http://socialmedia-class.org/twittertutorial.html to show how to create a Twitter stream with the Twitter library.

The file sortURIs.py was to get rid of links that circled back to Twitter and to go through all the redirects for each link. This technique was modeled after the advice from Hussam

Hallak.

```
#!/usr/bin/env python3
import urllib.request
from urllib.parse import urlparse

unique_url_list = []
counter = 3000
with open('urlList', 'r') as rfile:
        with open('sortedUrlList', 'w') as wfile:
                for url in rfile:
                        req = urllib.request.Request(url)
                        try:
                                res = urllib.request.urlopen(req, timeout = 10)
                                actual_url = res.geturl()
                                parsed_uri = urlparse(actual_url)
                                domain = '{uri.scheme}://{uri.netloc}/'.format(uri=parsed_uri)
                                if(domain != 'https://twitter.com/' and domain != 'https://t.co/'):
                                        if counter >= 0:
                                                print(counter)
                                                counter = counter - 1
                                                wfile.write(actual_url)
                                                wfile.write("\n")
                                        else:
                                                break
                        except urllib.error.URLError as e:
                                print(e.reason)
                                continue
                        except:
                                continue
                        if counter < 0:
                                break
```

I couldn't get the extended mode to work, which I think was meant for the REST mode of the Twitter API. The expanded URL, while it was the only option similar to extended mode still gave links back to the tweet itself. Therefore, I decided to just throw out all of those links entirely, hence why I decided to collect so many links. I opened each url to follow all the redirects and keep the last url that gave a successful 200 code. I wrote the links that were left to a new file.

The next program, uniqueLines.py was sourced from http://www.wellho.net/mouth/3662_Finding-all-the-unique-lines-in-a-file-using-Python-or-Perl.html .

```
#from http://www.wellho.net/mouth/3662_Finding-a

file = open('uniqueUrls_1000', 'w')

def unique(source):
        sofar = {}
        for val in open(source):
                if not sofar.get(val):
                        yield val.strip()
                        sofar[val] = 1


for lyne in unique("sortedUrlList"):
        file.write(lyne)
        file.write("\n")
```

This program went through the previously created file and removed all of the duplicate links, writing them to a new file containing my list or unique URLs. I just took the first 1000 and deleted the rest. Putting a counter around the for loop did not work as expected so I went the old fashioned way and manually deleted the extra.

Download TimeMaps for each URI

Timemaps.py takes all of the urls in uniqueUrls_1000 and gets the amount of timemaps.

```
#!/usr/bin/env python3
#get the momentoes for all links in file
import urllib.request

urlFile = open('uniqueUrls_1000', 'r')
timeFile = open('timeMapCount_uniqueUrls_1000', 'w')

mementoAgg = 'http://memgator.cs.odu.edu/timemap/link/'

for dirtyline in urlFile:
    line = dirtyline.strip()
    #line = 'http://www.cs.odu.edu/'
    url = mementoAgg + line
    try:
        req = urllib.request.Request(url)
        res = urllib.request.urlopen(req)
        timemap = res.read().decode(res.info().get_param('charset') or 'utf-8')
        timeFile.write(str(timemap.count("rel=\"memento\""))+ "\n")
    except urllib.error.URLError as e:
        if e.reason == 'Not Found':
            timeFile.write("0\n")

urlFile.close()
timeFile.close()
```

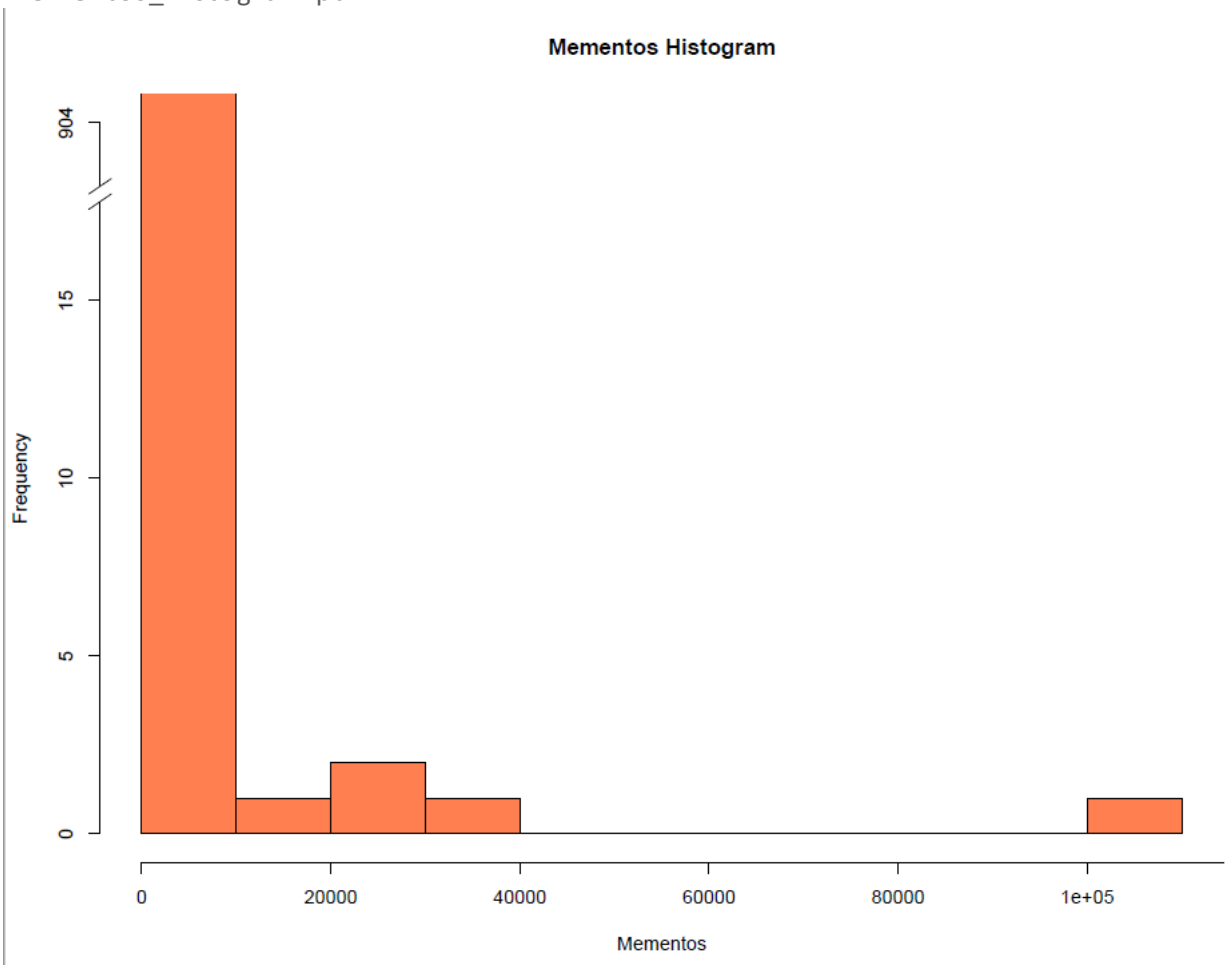
The program uses the memgator.cs.odu.edu/timemap/link website to pass each url to it and get a list of links describing the timemaps. The program the saves how many timemaps are there into a new file timeMapCount_uniqueUrls_1000. If there are no timemaps, the program just returns a not found error and it is written as a 0 into the file.

The histogram for the amount of urls to have a certain number of timemaps is written in histogram.R

```
library(plotrix)
mementos <- read.table("C:/Users/bitto/Documents/GitHub/cs532-s17/assignments/a2-solution/timeMapCount_uniqueUrls_1000")
counts <- table(mementos)

marks<- pretty(mementos$V1)
ygap <-ifelse(counts >20, counts-884, counts)
yat <- pretty(ygap)
ylab <-ifelse(yat > 19, yat+884, yat)
hist(mementos$V1, ylim=c(0,20), breaks = 15, axes=FALSE, main="Mementos Histogram", xlab="Mementos", col="coral")
axis(1,at=marks,labels=marks)
axis(2,at=yat,labels=ylab)
axis.break(2,18,style="slash")
```

The graph produced
Mementos_Histogram.pdf



Estimate Age Based on Carbon Date Tool

I used the online carbon date tool to collect the estimated creation date of each URI.

```
#!/usr/bin/env python3
import urllib.request
#from urllib.parse import urlparse
import time
import json

urlFile = open("uniqueUrls_1000", 'r')
ageFile = open("CarbonDate_uniqueUrls_1000" , 'w')

CarbonDateTool = "http://cd.cs.odu.edu/cd?url="

for url in urlFile:
    url = url.strip()
    finalurl = CarbonDateTool + url

    #make 10 attempts at connecting before quiting
    try:
        req = urllib.request.Request(finalurl)
        res = urllib.request.urlopen(req)
    except urllib.error.URLError as e:
        print(e.reason)
        if e.reason == "Service Unavailable":
            time.sleep(3)
            try:
                req = urllib.request.Request(finalurl)
                res = urllib.request.urlopen(req)
            except:
                #print ("Skipping, url failed with Service Unavailable:")
                #print(url)
                ageFile.write("\n")
                continue
        if e.reason == "Gateway Time-out":
            #print ("Skipping, url failed with gateway timeout:")
            #print(url)
            ageFile.write("\n")
            continue

    carbon_data = res.read().decode(res.info().get_param('charset') or 'utf-8')
    json_data = json.loads(carbon_data)
    if json_data["Estimated Creation Date"] == "":
        ageFile.write("NA")
        ageFile.write("\n")
    else:
        ageFile.write(json_data["Estimated Creation Date"])
        ageFile.write("\n")


urlFile.close()
ageFile.close()
```

I could not get docker to work on my computer to do it locally, so I made do with what I could and used the online tool at weird hours to hopefully reduce the load on it and get the data I needed. I put those dates in a new file, CarbonDate_uniqueUrls_1000.

I couldn't get R to convert my dates properly in time so I did not create a graph plotting the number of mementos and its age.

total URIs:	1000
no mementos:	904
no date estimate:	206