CS 751: Assignment 1

Bharath Kongara

Spring 2014

Contents

1 Question 1

Write a program that extracts 10000 tweets with links from Twitter. Reference - http://thomassileo.com/blog/2013/01/25/usi twitter-rest-api-v1-dot-1-with-python/ Other resources are available Note that only Twitter API 1.1 is currently available; version 1 code will no longer work.

- Save the tweets URIs, and the mapping to the link(s) each tweet contains
- For each `t.co` link use "curl I L" to record the HTTP headers all the way to a terminal HTTP status (i.e. chase down all the redirects)
- How many unique final URIs? How many duplicate URIs?
- Build a histogram of how many redirects (every URI will have at least 1) `http://en.wikipedia.org/wiki/Histogram` Build a histogram of HTTP status codes encountered (youll have at least 20000: 10000 301s, and 10000+ more)

1.1 Solution

The following steps were taken to extract tweets from twitter:

- Consumer key, consumer secret key, OAUTH token, OAUTH secret token are collected by registering an application in Twitter.
- Tweepy package is used to get the URI's from tweets with a specific keyword.
- 'Tweet Id', 'Tweet URI', 'Tweet Created Date' are stored in file as JSON format for further usage.
- Requests package is used to get all redirect headers, redirect count and final URI's.
- Unique URI's, duplicate URI's are counted using set function in python.

1.2 Code Listing

Here is the python code that is used to collect Thousand URL's from twitter.

```
import tweepy
import json
import time
import sys
import re

# Authentication Keys to Connect to Twitter API
CONSUMER_KEY = "qhJBwYCG9nYR7d5a5ZNfJD3Wy"

CONSUMER_SECRET = "gWAh87ScdyhUE5weUNc7PvLsc2Lax2yLm6sGKOjIbxELnfcKBS"

OAUTH_TOKEN = "2801896164-B9QLY4qwRBDKzi0LH1e1utgywvPrYdDOHJHnG0i"

OAUTH_TOKEN_SECRET = "ATnP7537AG1Q6tNgvwnekprYz5vpYgj7jykxRGYEgo8nX"
tweetFile = open('tweetLink', 'w')                                  #Open a file to write the
    output
auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)           #Get OAuth handler from
    tweepy
auth.set_access_token(OAUTH_TOKEN, OAUTH_TOKEN_SECRET)
url_list = set()
api = tweepy.API(auth)                                              #Getting Tweepy API
search_results = tweepy.Cursor(api.search,q="http:").items()

while True:                                                         # Infinite loop through tweets
    try:
        tweet = search_results.next()
        item= tweet._json
        eachitem={}
        created_date= item['user']['created_at']
        tweet_id= item['id_str']
        eachitem['id'] = tweet_id
        eachitem['createdDate'] = created_date
        for link in item['entities']['urls']:
            url_list.add(link['url'])
            eachitem['link']=link['url']
            tweetFile.write(json.dumps(eachitem) + '\n')
            print link['url']
        if len(url_list) == 10000:                                  # Break at 10000
            break
    except tweepy.TweepError:
        time.sleep(60 * 15)
        continue
    except StopIteration:
        break
```

Listing 1: Python program for extracting 10000 links for a given keyword

histogramData.py

```
import requests
import json

dicts={}
redirect={}
histogramdata = open('histogramdata', 'w')
histogramLinkData = open('histogramLinkdata', 'w')
histogramStatusData =  open('histogramStatusdata', 'w')
for i in range(1,11):
    filepath = "statusoutputs"
    filepath+=`i`
    fh=open(filepath,'r')
    for line in fh:
        eachJson=json.loads(line)
        for status in eachJson['statuscodes']:
            histogramStatusData.write(str(status)+'\n')
            if status in dicts:
                dicts[status]+=1
            else:
                dicts[status] =1
            statuslen = len(eachJson['statuscodes'])-1
            histogramLinkData.write(str(statuslen) + '\t'+str(eachJson['finalurl'])+'\n')
            if statuslen in redirect:
                redirect[statuslen]+=1
            else:
                redirect[statuslen]=1
finalJson={}
finalJson['statusfinaldata']=dicts
finalJson['redirectdata']=redirect
histogramdata.write(json.dumps(finalJson) + '\n')
```

Listing 2: Python program to find count of unique,duplicate final URI's and

Uniqueuris.py

```
import requests
import json


histogramdata = open('histogramdata', 'w')
histogramLinkData = open('histogramLinkdata', 'w')
histogramStatusData =  open('histogramStatusdata', 'w')
url_list = set()
for i in range(1,11):
    filepath = "statusoutputs"
    filepath+=`i`
    fh=open(filepath,'r')
    for line in fh:
        eachJson=json.loads(line)
        for status in eachJson['finalurl']:
            url_list.add(status)
print len(url_list)
```

Listing 3: Python program to find count of unique,duplicate final URI's and

RedirectHistogram.R

```
1 d=read.csv("histogramLinkdata.txt",stringsAsFactors=F,header=FALSE,sep="\t")
2 data=d[,1]
3 newData=data[which(data<10)]
4 png("redirect-histogramNew2.png")
5 hist(newData,main="Redirects Histogram",freq=T,xlab="Redirects",ylab="Frequency",ylim=c
      (0,20000),xlim=c(0,10))
6 dev.off()
```

Listing 4: R program for creating histogram to represent redirect frequency of URI

StatusCodesHistogram.R

```
1 d=read.csv("StatusCodes.txt",stringsAsFactors=F,header=FALSE,sep="\t")
2 data=d[,1]
3 newData=data[which(data<10)]
4 png("StatusCodes-histogramNew2.png")
5 hist(newData,main="StatusCodes Histogram",freq=T,xlab="Status Codes",ylab="Frequency")
6 dev.off()
```

Listing 5: R program for creating histogram to represent status codes frequency of URI

1.3 Results


```
{"link": "http://t.co/x4gG3aSiws", "id": "565391360061345792", "createdDate": "Thu May 01
    18:33:51 +0000 2014"}
{"link": "http://t.co/k9GANn2mAU", "id": "565391360057159682", "createdDate": "Thu Dec 31
    15:58:00 +0000 2009"}
{"link": "http://t.co/yLUmCXBH2R", "id": "565391360052953088", "createdDate": "Thu Jan 09
    08:59:45 +0000 2014"}
{"link": "http://t.co/ulzWwBUrCJ", "id": "565391360048783360", "createdDate": "Fri May 10
    10:54:00 +0000 2013"}
{"link": "http://t.co/ulzWwBUrCJ", "id": "565391360044593153", "createdDate": "Tue Dec 30
    02:08:42 +0000 2014"}
{"link": "http://t.co/ulzWwBUrCJ", "id": "565391360044584960", "createdDate": "Sun Jul 07
    03:09:11 +0000 2013"}
{"link": "http://t.co/usIvf3FPeu", "id": "565391360040370176", "createdDate": "Fri Feb 11
    19:01:57 +0000 2011"}
{"link": "http://t.co/TAHBVZMeek", "id": "565391360036200449", "createdDate": "Sat Feb 18
    17:21:14 +0000 2012"}
{"link": "http://t.co/qGgEQbjTHa", "id": "565391360036196353", "createdDate": "Wed Jul 30
    07:19:40 +0000 2014"}
{"link": "http://t.co/ulzWwBUrCJ", "id": "565391360036196352", "createdDate": "Fri Oct 04
    13:35:08 +0000 2013"}
{"link": "http://t.co/eEtpUfkgSM", "id": "565391360036192257", "createdDate": "Thu Oct 16
    15:34:19 +0000 2014"}
{"link": "http://t.co/5mWvEMxhDC", "id": "565391360032014336", "createdDate": "Fri Aug 21
    20:46:27 +0000 2009"}
{"link": "http://t.co/GkyPnGbWxV", "id": "565391360027807745", "createdDate": "Thu Oct 23
    18:14:24 +0000 2008"}
{"link": "http://t.co/5SBpnDdRX6", "id": "565391360023609345", "createdDate": "Wed Dec 25
    13:21:48 +0000 2013"}
{"link": "http://t.co/wBLsbgUZMR", "id": "565391360023605249", "createdDate": "Sat Nov 02
    16:14:56 +0000 2013"}
{"link": "http://t.co/lf7BXQWGyV", "id": "565391357762863104", "createdDate": "Sat Aug 17
    04:37:27 +0000 2013"}
{"link": "http://t.co/VNI4jA6F2z", "id": "565391357758668802", "createdDate": "Thu Jul 26
    06:01:13 +0000 2012"}
{"link": "http://t.co/lpsyYlPu8k", "id": "565391357750292480", "createdDate": "Mon Oct 08
    06:04:33 +0000 2012"}
{"link": "http://t.co/mxW8DcIF5h", "id": "565391357733515264", "createdDate": "Sun Apr 20
    14:10:32 +0000 2014"}
{"link": "http://t.co/1aYjTRcoGr", "id": "565391357720944640", "createdDate": "Mon Nov 10
    23:55:39 +0000 2014"}
{"link": "http://t.co/AFpHxKG9sh", "id": "565391357712539649", "createdDate": "Fri May 23
    16:22:31 +0000 2014"}
{"link": "http://t.co/XlLxfMHidn", "id": "565391357708345345", "createdDate": "Wed May 21
    12:45:19 +0000 2014"}
{"link": "http://t.co/gUfDM0pRGU", "id": "565391357704146944", "createdDate": "Tue Aug 05
    05:08:04 +0000 2014"}
{"link": "http://t.co/qLP4wl0caW", "id": "565391357699969025", "createdDate": "Fri Mar 01
    13:57:44 +0000 2013"}
{"link": "http://t.co/7J8v6ieNvS", "id": "565391357699964928", "createdDate": "Wed Mar 03
    15:24:57 +0000 2010"}
{"link": "http://t.co/FW5Ldr8pGz", "id": "565391357695782912", "createdDate": "Tue Sep 17
    23:14:40 +0000 2013"}
{"link": "http://t.co/0LKZuZ8dQS", "id": "565391357695766529", "createdDate": "Fri Dec 30
    09:06:36 +0000 2011"}
{"link": "http://t.co/vqNnUyX85Z", "id": "565391357687373825", "createdDate": "Tue Jul 02
    09:17:11 +0000 2013"}
{"link": "http://t.co/WD9Pl3Z93y", "id": "565391357683200002", "createdDate": "Sun Apr 07
    04:12:16 +0000 2013"}
{"link": "http://t.co/852b8oL7Ie", "id": "565391357678997505", "createdDate": "Wed Nov 14
    05:07:41 +0000 2012"}
{"link": "http://t.co/TRF8sBRCVl", "id": "565391354831073280", "createdDate": "Mon Jun 06
    13:43:06 +0000 2011"}
{"link": "http://t.co/rIN59kOXEt", "id": "565391354831065089", "createdDate": "Sat Sep 20
    07:17:30 +0000 2014"}
{"link": "http://t.co/CCqPTv34xD", "id": "565391354814275585", "createdDate": "Fri Oct 24
    13:14:35 +0000 2014"}
```


Listing 6: Sample Result for Above Program

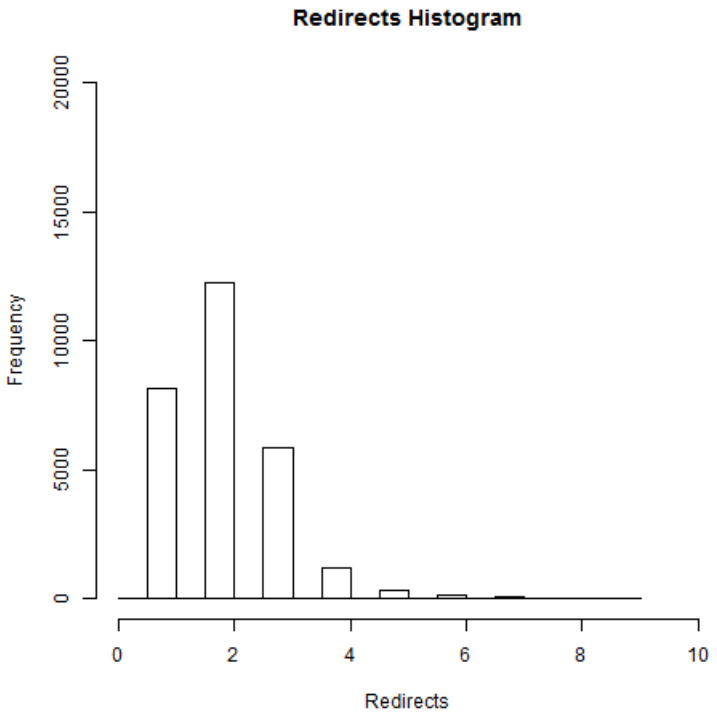

Figure 1: Histogram

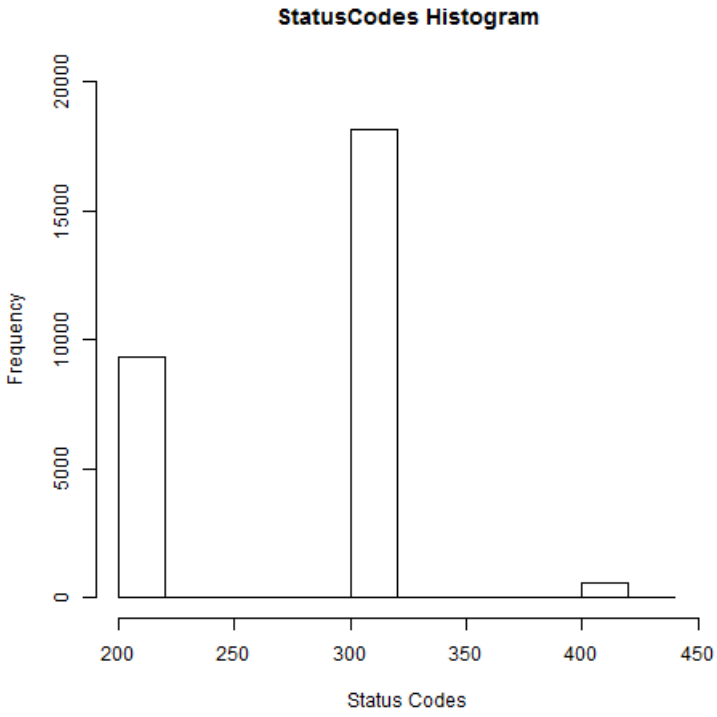

Figure 2: Histogram

2 Question 2

Write a Python program that:

- Uses "Carbon Date" to estimate the age of each links(s) in a tweet -See:"http://ws-dl.blogspot.com/2013/04/2013-04-19-carbon-dating-web.html"
- Create a histogram of (Age_{tweet} - Age_{link}) .Many (most?) deltas will be 0,but there should be many¿0
- For these, compute: median, mean, std dev, std err
- Use wget to download the text for all the links. Hold on to those, we'll come back to them later. -See : "`http://superuser.com/questions/55040/save-a-single-web-page-with-background-images-with-wget`" "`http://stackoverflow.com/questions/6348289/download-a-working-local-copy-of-a-webpage`"

2.1 Solution

- Carbon Date tool installed as per instructions provided by author.
- Few URI's were consuming lot of time to query all of its services. So, I have divided 10000 URI's to 10 files and ran 10 scripts on different linux servers.
- I have made few changes to local.py in the carbon tool to read 1000 URI's from a text file and write back the result into another text file in JSON format.
- To find an age of a link, I have written a python program 'calculateDays.py' which will take the date and return days.
- Mean, median, standard deviation and standard error are calculated using statistics python package.

2.2 Code Listing

local.py

```
1  from checkForModules import checkForModules
2  import json
3  from ordereddict import OrderedDict
4  import urlparse
5  import re
6  from getBitly import getBitlyCreationDate
7  from getArchives import getArchivesCreationDate
8  from getGoogle import getGoogleCreationDate
9  from getBacklinks import *
10 from getLowest import getLowest
11 from getLastModified import getLastModifiedDate
12 from getTopsyScrapper import getTopsyCreationDate
13 from htmlMessages import *
14 from pprint import pprint
15 from threading import Thread
16 import Queue
17 import datetime
18 import os,sys, traceback
19 def cd(url, backlinksFlag = False):
20
21     #print 'Getting Creation dates for: ' + url
22     #scheme missing?
23     parsedUrl = urlparse.urlparse(url)
24     if( len(parsedUrl.scheme)<1 ):
25         url = 'http://'+url
26     threads = []
27     outputArray =['','','','','','']
28     now0 = datetime.datetime.now()
29     lastmodifiedThread = Thread(target=getLastModifiedDate, args=(url, outputArray, 0))
30     bitlyThread = Thread(target=getBitlyCreationDate, args=(url, outputArray, 1))
31     googleThread = Thread(target=getGoogleCreationDate, args=(url, outputArray, 2))
32     archivesThread = Thread(target=getArchivesCreationDate, args=(url, outputArray, 3))
33     if( backlinksFlag ):
```

```
34          backlinkThread = Thread(target=getBacklinksFirstAppearanceDates, args=(url,
                outputArray, 4))
35      topsyThread = Thread(target=getTopsyCreationDate, args=(url, outputArray, 5))
36      # Add threads to thread list
37      threads.append(lastmodifiedThread)
38      threads.append(bitlyThread)
39      threads.append(googleThread)
40      threads.append(archivesThread)
41      if( backlinksFlag ):
42          threads.append(backlinkThread)
43      threads.append(topsyThread)
44      # Start new Threads
45      lastmodifiedThread.start()
46      bitlyThread.start()
47      googleThread.start()
48      archivesThread.start()
49      if( backlinksFlag ):
50          backlinkThread.start()
51      topsyThread.start()
52      # Wait for all threads to complete
53      for t in threads:
54          t.join()
55      # For threads
56      lastmodified = outputArray[0]
57      bitly = outputArray[1]
58      google = outputArray[2]
59      archives = outputArray[3]
60      if( backlinksFlag ):
61          backlink = outputArray[4]
62      else:
63          backlink = ''
64      topsy = outputArray[5]
65      #note that archives["Earliest"] = archives[0][1]
66      try:
67          lowest = getLowest([lastmodified, bitly, google, archives[0][1], backlink, topsy]) #
                for thread
68      except:
69         print sys.exc_type, sys.exc_value , sys.exc_traceback
70      result = []
71      result.append(("URI", url))
72      result.append(("Estimated Creation Date", lowest))
73      result.append(("Last Modified", lastmodified))
74      result.append(("Bitly.com", bitly))
75      result.append(("Topsy.com", topsy))
76      result.append(("Backlinks", backlink))
77      result.append(("Google.com", google))
78      result.append(("Archives", archives))
79      values = OrderedDict(result)
80      r = json.dumps(values, sort_keys=False, indent=2, separators=(',', ': '))
81      now1 = datetime.datetime.now() - now0
82      print 'runtime in seconds:  ' +  str(now1.seconds) + '\n' + r + '\n'
83      return lowest
84  i=1
85  fil = open("../statusoutputs1",'r+')
86  lowestModified = open('carbondateoutputs1', 'w')
87  for line in fil:
88      finaljson={}
89      eachJson=json.loads(line)
90      url=eachJson['finalurl']
91      finaljson['tweetcreatedDate'] = eachJson['createdDate']
92      finaljson['createdDate'] = cd(url)
93      lowestModified.write(json.dumps(finaljson) + '\n')
94      print i
95      i+=1
```

Listing 7: Python program for getting creation date for URI's

CalculateDays.py

```
1
2  import sys
3  import time
4  import datetime
5  import json
6  import statistics
7  import scipy import stats
8  def cd(cdate):
9      try:
10         ct = time.strptime(cdate, "%Y-%m-%dT%H:%M:%S")
11         cdt = datetime.datetime.fromtimestamp(time.mktime(ct))
12         now = datetime.datetime.now()
13         days = (now - cdt).days
14
15     except ValueError:
16         print ValueError
17     return str(days)
18 def cdtwitter(cdate):
19     try:
20         ts = time.strftime('%Y-%m-%d %H:%M:%S', time.strptime(cdate,'%a %b %d %H:%M:%S +0000
               %Y'))
21         ct = time.strptime(ts, "%Y-%m-%d %H:%M:%S")
22         cdt = datetime.datetime.fromtimestamp(time.mktime(ct))
23         now = datetime.datetime.now()
24         days = (now - cdt).days
25     except ValueError:
26         print ValueError
27     return str(days)
28 fh=open('carbondateoutputs1','r')
29 finalStat=open('finalstatistics','w')
30 finalStatus=open('finalstatisticstxt.txt','w')
31 finalData = set()
32 i=0
33 totalData={}
34 for line in fh:
35     eachJson=json.loads(line)
36     if len(eachJson['tweetcreatedDate'])!=0:
37         twitterAge = cdtwitter(eachJson['tweetcreatedDate'])
38     if eachJson['createdDate']!="":
39         linkage = cd(eachJson['createdDate'])
40     absValue= abs(int(twitterAge) - int(linkage))
41     finalData.add(absValue)
42     finalStatus.write(str(absValue)+ '\n')
43     sortedlist = sorted(finalData)
44 print statistics.mean(sortedlist)
45 print statistics.median(sortedlist)
46 totalData['data'] = sortedlist
47 totalData['mean'] =  statistics.mean(sortedlist)
48 totalData['median'] = statistics.median(sortedlist)
49 totalData['Std Dev'] = statistics.stdev(sortedlist)
50 totalData['Std Err'] = stats.sem(sortedlist
51 finalStat.write(json.dumps(totalData))
52 #print statistics.mode(sortedlist)
53 fh.close()
```

Listing 8: Python program for calculating the age of URI's

StatusCodesHistogram.py

```
1 d=read.csv("StatusCodes.txt",stringsAsFactors=F,header=FALSE,sep="\t")
2 data=d[,1]
3 newData=data[which(data<10)]
4 png("StatusCodes-histogramNew2.png")
5 hist(newData,main="StatusCodes Histogram",freq=T,xlab="Status Codes",ylab="Frequency")
6 dev.off()
```

Listing 9: R program for creating histogram of Tweet Age - Link Age

2.3 Results


```
{"tweetcreatedDate": "Sun Apr 27 03:09:07 +0000 2014", "createdDate": "2014-09-03T00:00:00"}
{"tweetcreatedDate": "Sun Jan 26 17:49:12 +0000 2014", "createdDate": ""}
{"tweetcreatedDate": "Thu May 24 19:32:59 +0000 2012", "createdDate": "2001-02-01T00:00:00"}
{"tweetcreatedDate": "Sat Feb 01 02:42:50 +0000 2014", "createdDate": "2015-02-06T16:40:55"}
{"tweetcreatedDate": "Sun Aug 15 02:39:59 +0000 2010", "createdDate": "2015-02-09T14:32:28"}
{"tweetcreatedDate": "Tue Jul 01 01:49:24 +0000 2014", "createdDate": "2001-02-01T00:00:00"}
{"tweetcreatedDate": "Tue May 26 12:07:03 +0000 2009", "createdDate": ""}
{"tweetcreatedDate": "Tue Sep 27 16:05:54 +0000 2011", "createdDate": "2012-02-09T00:00:00"}
{"tweetcreatedDate": "Fri Sep 03 20:04:55 +0000 2010", "createdDate": ""}
{"tweetcreatedDate": "Fri Oct 10 08:24:36 +0000 2014", "createdDate": "2011-05-12T00:00:00"}
{"tweetcreatedDate": "Mon Mar 16 19:49:45 +0000 2009", "createdDate": ""}
{"tweetcreatedDate": "Sat Dec 24 02:35:37 +0000 2011", "createdDate": "2015-02-10T17:02:24"}
{"tweetcreatedDate": "Sat Jun 05 10:41:46 +0000 2010", "createdDate": "2015-01-23T00:01:05"}
{"tweetcreatedDate": "Sun Apr 27 06:04:54 +0000 2014", "createdDate": "2014-09-03T00:00:00"}
{"tweetcreatedDate": "Thu Feb 13 21:42:04 +0000 2014", "createdDate": "2015-01-27T04:59:32"}
{"tweetcreatedDate": "Sat Apr 07 09:19:00 +0000 2012", "createdDate": "2007-10-31T00:00:00"}
{"tweetcreatedDate": "Mon Sep 30 09:41:16 +0000 2013", "createdDate": "2001-02-01T00:00:00"}
{"tweetcreatedDate": "Fri Oct 03 14:03:02 +0000 2014", "createdDate": "2001-02-01T00:00:00"}
{"tweetcreatedDate": "Thu Mar 08 01:45:24 +0000 2012", "createdDate": ""}
{"tweetcreatedDate": "Sun May 24 14:14:04 +0000 2009", "createdDate": ""}
{"tweetcreatedDate": "Fri Sep 05 18:47:56 +0000 2014", "createdDate": ""}
{"tweetcreatedDate": "Sun Jan 08 03:53:03 +0000 2012", "createdDate": "2015-02-09T20:42:18"}
{"tweetcreatedDate": "Thu Aug 06 20:04:43 +0000 2009", "createdDate": "2013-02-04T00:00:00"}
{"tweetcreatedDate": "Mon Feb 24 06:26:27 +0000 2014", "createdDate": "2015-02-09T20:37:39"}
{"tweetcreatedDate": "Sat Oct 08 15:20:24 +0000 2011", "createdDate": "2015-02-09T14:54:26"}
{"tweetcreatedDate": "Sun Apr 21 08:45:39 +0000 2013", "createdDate": "2015-02-09T20:51:42"}
{"tweetcreatedDate": "Fri Mar 01 18:46:45 +0000 2013", "createdDate": "2014-04-11T00:00:00"}
{"tweetcreatedDate": "Sun Feb 23 07:02:49 +0000 2014", "createdDate": "2015-02-09T20:37:39"}
{"tweetcreatedDate": "Sat Jun 27 07:29:25 +0000 2009", "createdDate": ""}
{"tweetcreatedDate": "Wed Apr 24 18:18:15 +0000 2013", "createdDate": "2015-02-09T20:31:56"}
{"tweetcreatedDate": "Fri Oct 21 10:55:27 +0000 2011", "createdDate": "2001-02-01T00:00:00"}
{"tweetcreatedDate": "Fri Nov 21 07:30:18 +0000 2014", "createdDate": "2001-02-01T00:00:00"}
{"tweetcreatedDate": "Thu Feb 14 12:04:21 +0000 2013", "createdDate": "2008-08-25T00:00:00"}
{"tweetcreatedDate": "Thu Aug 08 21:08:44 +0000 2013", "createdDate": "2014-09-16T14:14:14"}
{"tweetcreatedDate": "Thu Oct 09 07:43:44 +0000 2014", "createdDate": "2011-01-14T00:00:00"}
{"tweetcreatedDate": "Wed Feb 19 17:23:41 +0000 2014", "createdDate": "2015-02-09T20:45:36"}
{"tweetcreatedDate": "Tue Oct 28 11:35:36 +0000 2014", "createdDate": ""}
{"tweetcreatedDate": "Wed Apr 25 01:49:38 +0000 2012", "createdDate": "2001-02-01T00:00:00"}
{"tweetcreatedDate": "Fri Apr 25 23:39:39 +0000 2014", "createdDate": "2014-12-19T00:00:00"}
{"tweetcreatedDate": "Sun Feb 23 15:42:39 +0000 2014", "createdDate": "2015-02-09T20:37:39"}
{"tweetcreatedDate": "Mon Aug 11 08:31:36 +0000 2014", "createdDate": "2015-02-09T20:51:41"}
{"tweetcreatedDate": "Tue Jan 17 14:02:20 +0000 2012", "createdDate": "2015-02-08T05:46:47"}
{"tweetcreatedDate": "Tue Jan 17 14:02:20 +0000 2012", "createdDate": ""}
{"tweetcreatedDate": "Sun Feb 14 07:59:55 +0000 2010", "createdDate": "2015-02-09T20:51:41"}
{"tweetcreatedDate": "Sat Jul 27 19:19:47 +0000 2013", "createdDate": "2015-02-09T18:06:23"}
{"tweetcreatedDate": "Sun Apr 05 02:06:05 +0000 2009", "createdDate": ""}
{"tweetcreatedDate": "Fri May 16 08:03:42 +0000 2014", "createdDate": "2015-01-25T05:11:31"}
{"tweetcreatedDate": "Sat Apr 07 08:50:13 +0000 2012", "createdDate": "2007-10-31T00:00:00"}
{"tweetcreatedDate": "Fri Sep 20 12:26:54 +0000 2013", "createdDate": "2014-10-13T00:00:00"}
```


Listing 10: Sample Result for Above Program

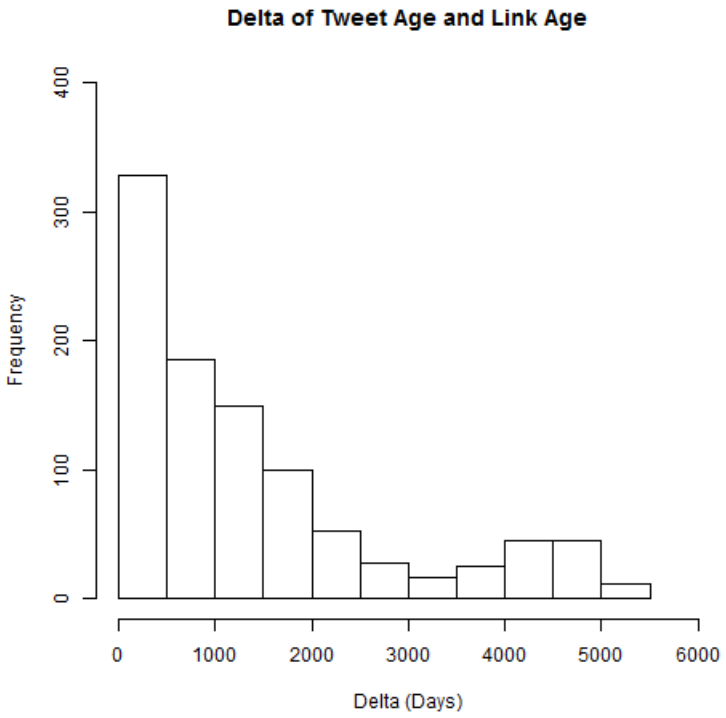

Figure 3: Histogram

Bibliography

[1] Basic date and time types in python. https://docs.python.org/2/library/datetime.html.

[2] Github for carbondate. https://github.com/HanySalahEldeen/CarbonDate.

[3] Producing simple graphs in r. http://www.harding.edu/fmccown/r/.

[4] Twitter search for python. https://github.com/ckoepp/TwitterSearch.

[5] Using twitter api keys. http://thomassileo.com/blog/2013/01/25/using-twitter-rest-api-v1-dot-1-with-python/.