

LAB2: DATA AGGREGATION, BIG DATA ANALYSIS AND VISUALIZATION

- Nikhil Srihari, 50291966, nikhilsr

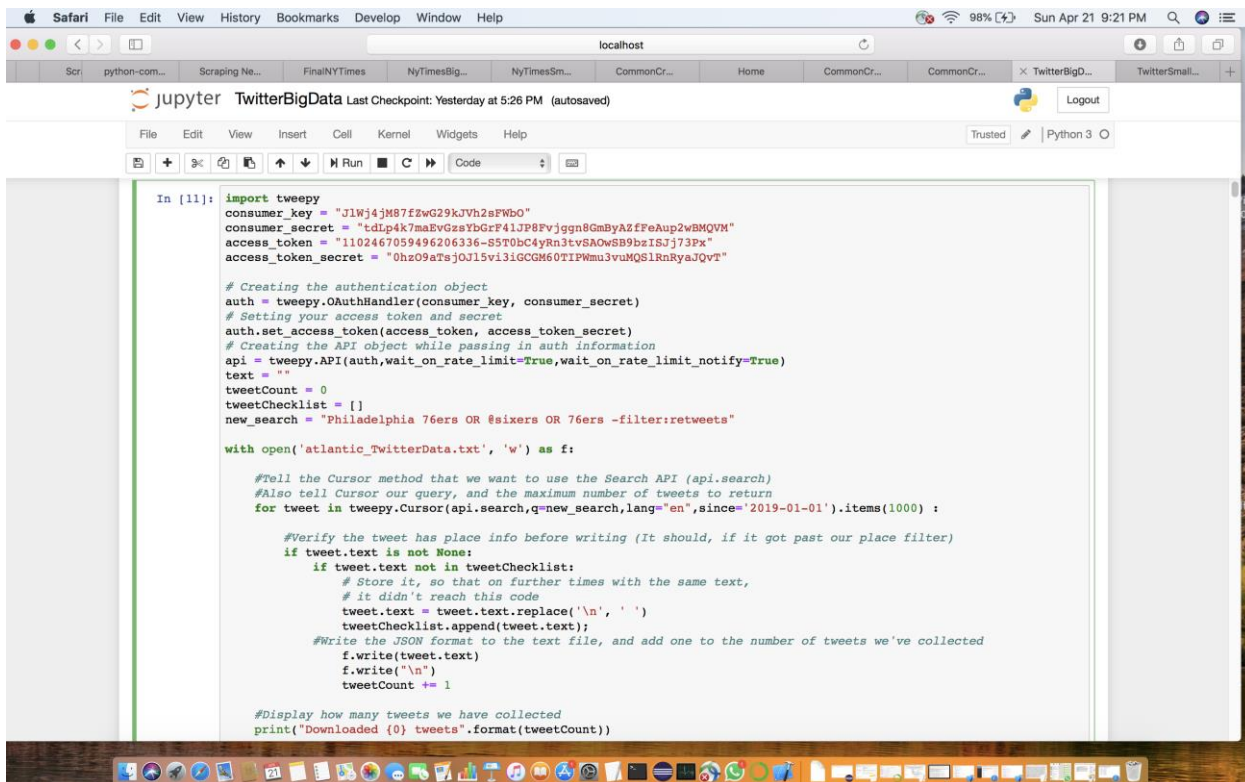
Part 1 :

Collect Twitter, NyTimes and Common Crawl .

We have selected NBA as the main topic . The subtopics being :

- i. Atlantic(Boston Celtics, Brooklyn Nets, New York Knicks, Philadelphia 76ers, Toronto Raptors),
- ii. Central(Cleveland Cavaliers, Detroit Pistons, Indiana Pacers, Milwaukee Bucks, Chicago Bulls)
- iii. Southeast(Atlanta Hawks, Charlotte Hornets, Miami Heat, Orlando Magic, Washington Wizards),
- iv. Southwest(Houston Rockets, Dallas Mavericks, Memphis Grizzlies, New Orleans Pelicans)
- v. Northwest(Denver Nuggets, Minnesota Timberwolves, Oklahoma City Thunder, Portland Trail Blazers, Utah Jazz),
- vi. Pacific(Golden State Warriors, Los Angeles Lakers, Phoenix Suns, Sacramento Kings).

We have chose Python as the language for data collection. Approximately we have collected 23000 tweets. We have used Tweepy for tweet collection. I have placed all the tweets in the textfile. Below is the code snippet used for tweet collection:



```
In [11]: import tweepy
consumer_key = "J1Wj4jM87fZwG29kVh2sFWb0"
consumer_secret = "tdLp4k7maEvGzaYbGrF41JP8Fvjggn8GmByAZfFeAup2wBMQVM"
access_token = "1102467059496206336-S5T0bC4yRn3tvSAOWSB9bzISJj73Px"
access_token_secret = "0hzO9aTsJ0J15vi31GCGH60TIPWmu3vuMQS1RnRyaJQvt"

# Creating the authentication object
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
# Setting your access token and secret
auth.set_access_token(access_token, access_token_secret)
# Creating the API object while passing in auth information
api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)
text = ""
tweetCount = 0
tweetChecklist = []
new_search = "Philadelphia 76ers OR @sixers OR 76ers -filter:retweets"

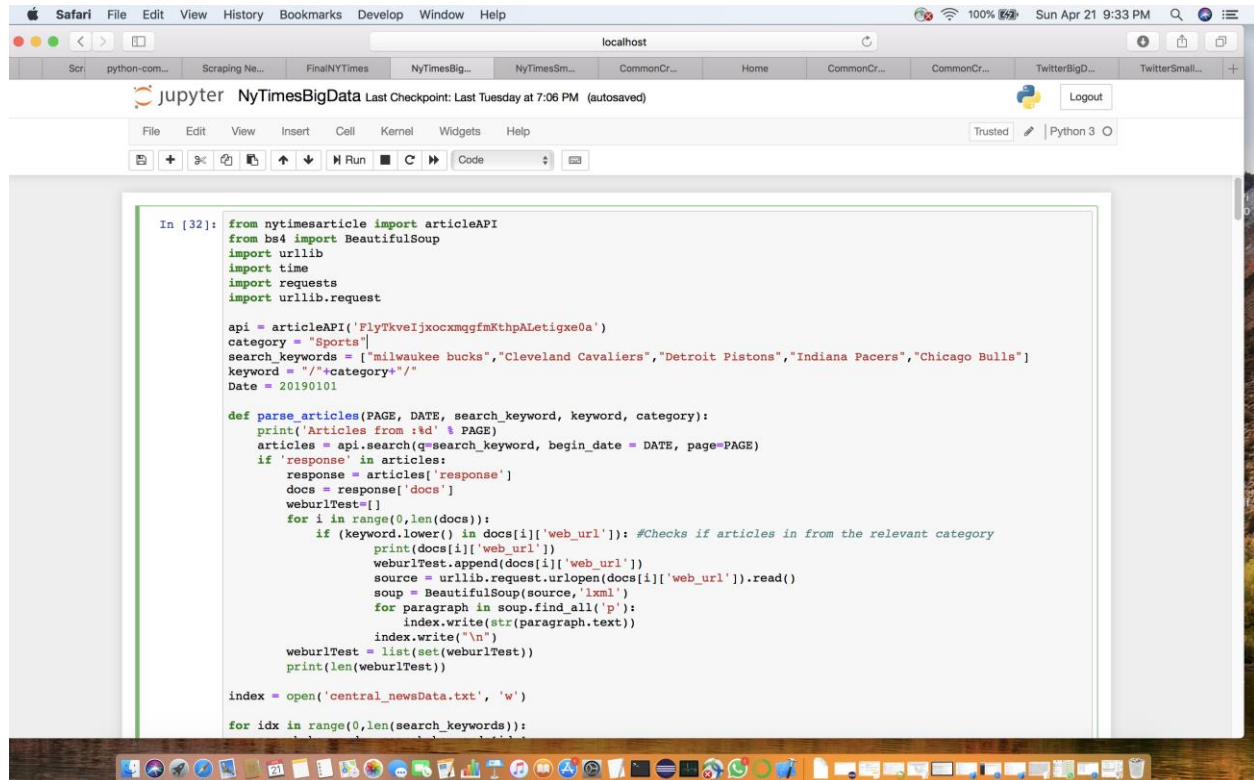
with open('atlantic_TwitterData.txt', 'w') as f:

    #Tell the Cursor method that we want to use the Search API (api.search)
    #Also tell Cursor our query, and the maximum number of tweets to return
    for tweet in tweepy.Cursor(api.search, q=new_search, lang="en", since='2019-01-01').items(1000) :

        #Verify the tweet has place info before writing (It should, if it got past our place filter)
        if tweet.text is not None:
            if tweet.text not in tweetChecklist:
                # Store it, so that on further times with the same text,
                # it didn't reach this code
                tweet.text = tweet.text.replace('\n', ' ')
                tweetChecklist.append(tweet.text);
                #Write the JSON format to the text file, and add one to the number of tweets we've collected
                f.write(tweet.text)
                f.write("\n")
                tweetCount += 1

    #Display how many tweets we have collected
    print("Downloaded {0} tweets".format(tweetCount))
```

For Nytimes, I created an account in NYtimes as developer and created an API key(FlyTkveljxocxmqqgmKthpALetigxe0a). Using the above team names as keywords, we have searched for articles which has those keywords in them. From the response received from the articles, we have fetched the URL's. I have collected approximately 600 articles(100 + in each subtopic). Using BeautifulSoup,I have scraped the URL's and fetched the contents and stored



```
In [32]: from nytimesarticle import articleAPI
from bs4 import BeautifulSoup
import urllib
import time
import requests
import urllib.request

api = articleAPI('FlyTkveljxocxmqqgmKthpALetigxe0a')
category = "Sports"
search_keywords = ["milwaukee bucks", "Cleveland Cavaliers", "Detroit Pistons", "Indiana Pacers", "Chicago Bulls"]
keyword = "/" + category + "/"
Date = 20190101

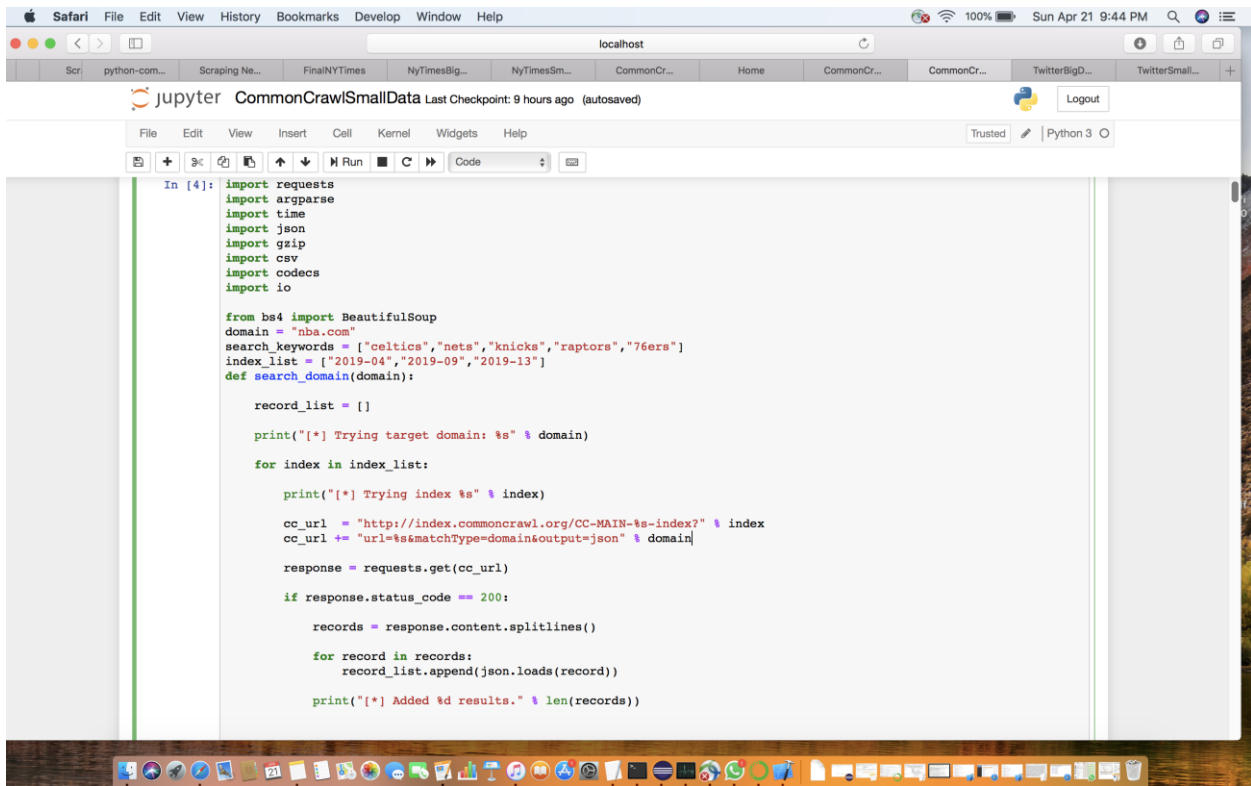
def parse_articles(PAGE, DATE, search_keyword, keyword, category):
    print('Articles from :%d' % PAGE)
    articles = api.search(q=search_keyword, begin_date = DATE, page=PAGE)
    if 'response' in articles:
        response = articles['response']
        docs = response['docs']
        weblurlTest=[]
        for i in range(0,len(docs)):
            if (keyword.lower() in docs[i]['web_url']): #Checks if articles in from the relevant category
                print(docs[i]['web_url'])
                weblurlTest.append(docs[i]['web_url'])
                source = urllib.request.urlopen(docs[i]['web_url']).read()
                soup = BeautifulSoup(source, 'lxml')
                for paragraph in soup.find_all('p'):
                    index.write(str(paragraph.text))
                index.write("\n")
        weblurlTest = list(set(weblurlTest))
        print(len(weblurlTest))

index = open('central_newsData.txt', 'w')

for idx in range(0,len(search_keywords)):
```

them in textile. Below is the code snippet, I have used for achieving this:

For common Crawl, I have fetched data from index.commoncrawl.org based on the indexes 2019-04,2019-09,2019-13. Of the results obtained, we are downloading the gzipped file. From the Gzipped file, we are fetching the html contents of the pages. I have approximately collected 600 articles.The results I have saved in the textfile. Below is the code snippet for achieving this.



Part 2:

As part of this task, I setup the Hadoop Environemnt on my system. I, first installed docker and then downloaded the hadoop image, as per the instructions given.

Once Hadoop Virtual Environemnt was setup, I ran the same program given : Finding the word occurence for pg345.txt file using the given mapper.py and reduce.py files, arranged in a certain folder structure.

Below are the Hadoop commands that I ran to accomplish this task:

```

docker run --hostname=quickstart.cloudera --privileged=true -t -i -v C:\Users\nikhi\Documents\UB\dockerMR\folder\part2:/src --publish-all=true -p 8888 cloudera/quickstart /usr/bin/docker-quickstart
hadoop fs -mkdir /user/nikhilsrc;hadoop fs -mkdir /user/nikhilsrc/MR;hadoop fs -mkdir /user/nikhilsrc/MR/input;
cd src/data;hadoop fs -put *.txt /user/nikhilsrc/MR/input;
hadoop fs -rm /user/nikhilsrc/MR/output;hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.7.0.jar -file /src/mapper.py -mapper /src/mapper.py -file /src/reducer.py -reducer /src/reducer.py -input /user/nikhilsrc/MR/input/*.txt -output /user/nikhilsrc/MR/output;hadoop fs -cat /user/nikhilsrc/MR/output/part*;hadoop fs -get /user/nikhilsrc/MR/output/ /src/

```

Here is the snapshot of the hadoop console while it was executing these commands:

```

@quickstart/src/data
9/04/22 02:25:45 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8032
9/04/22 02:25:46 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8032
9/04/22 02:25:46 INFO mapred.FileInputFormat: Total input paths to process : 1
9/04/22 02:25:46 INFO mapreduce.JobSubmitter: number of splits:2
9/04/22 02:25:46 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1555899829642_0001
9/04/22 02:25:47 INFO impl.YarnClientImpl: Submitted application application_1555899829642_0001
9/04/22 02:25:47 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1555899829642_0001/
9/04/22 02:25:47 INFO mapreduce.Job: Running job: job_1555899829642_0001
9/04/22 02:25:53 INFO mapreduce.Job: Job job_1555899829642_0001 running in uber mode : false
9/04/22 02:25:53 INFO mapreduce.Job: map 0% reduce 0%
9/04/22 02:25:58 INFO mapreduce.Job: map 50% reduce 0%
9/04/22 02:25:59 INFO mapreduce.Job: map 100% reduce 0%
9/04/22 02:26:03 INFO mapreduce.Job: map 100% reduce 100%
9/04/22 02:26:03 INFO mapreduce.Job: Job job_1555899829642_0001 completed successfully
9/04/22 02:26:03 INFO mapreduce.Job: Counters: 49
  File System Counters
    FILE: Number of bytes read=1515580
    FILE: Number of bytes written=3382033
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=871511
    HDFS: Number of bytes written=198227
    HDFS: Number of read operations=9
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  Job Counters
    Launched map tasks=2
    Launched reduce tasks=1
    Data-local map tasks=2
    Total time spent by all maps in occupied slots (ms)=5387
    Total time spent by all reduces in occupied slots (ms)=2706
    Total time spent by all map tasks (ms)=5387
    Total time spent by all reduce tasks (ms)=2706
    Total vcore-seconds taken by all map tasks=5387
    Total vcore-seconds taken by all reduce tasks=2706
    Total megabyte-seconds taken by all map tasks=5516288
    Total megabyte-seconds taken by all reduce tasks=2770944
  Map-Reduce Framework
    Map input records=15973
    Map output records=164424
    Map output bytes=1186726
    Map output materialized bytes=1515586
    Input split bytes=232
    Combine input records=0
    Combine output records=0
    Reduce input groups=19025
    Reduce shuffle bytes=1515586
    Reduce input records=164424
    Reduce output records=19025
    Spilled Records=328848
    Shuffled Maps =2
    Failed Shuffles=0
    Merged Map outputs=2
    GC time elapsed (ms)=97
    CPU time spent (ms)=3640
    Physical memory (bytes) snapshot=751030272
    Virtual memory (bytes) snapshot=4101820416
    Total committed heap usage (bytes)=754450432
  Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
  File Input Format Counters
    Bytes Read=871279
  File Output Format Counters
    Bytes Written=198227
9/04/22 02:26:03 INFO streaming.StreamJob: Output directory: /user/nikhilsr/MR/output
root@quickstart data]#

```

At the end of this process, the output file Part0001.txt was created with the word counts. Here is a snippet from that file:

@quickstart/src/data

```
yield; 1
yielded 3
yielded, 1
yielding 1
yields 2
yoke, 1
you 1047
you! 4
you!" 5
you!' 1
you!) 1
you'd 2
you'll 2
you're 2
you've 4
you, 98
you," 2
you,' 1
you--I, 1
you--account 1
you--and 2
you--as 2
you--how 1
you--oh, 1
you--with 1
you--yes, 1
you. 38
you." 13
you: 3
you; 5
you? 8
you?" 3
you_, 1
young 45
young, 3
young--here 1
young--like 1
young. 2
young; 1
younger 3
younger, 1
younger; 1
your 281
yours 3
yours!" 2
yours, 5
yours. 3
yours." 1
yours; 1
yourself 8
yourself, 4
yourself," 1
yourself. 6
yourself?" 2
yourself?' 1
yourselves 2
youth 5
youthful 2
zeal; 1
zealous 1
zoöphagous 3
zoöphagous, 1
zoöphagy!" 1
{pg} 8
{pg}184 1
£1 1
£10 1
æt. 1
ætāt 1
```

Part 3:

Task 1:

We started off with small dataset and did the word count. We optimized the word count code using associative arrays and it is giving a better word count.

Code Snippets are below:



```
mapper_wordcount.py x reducer_wordcount.py x
#!/usr/bin/env python
# coding: utf-8

# In[ ]:

#!/usr/bin/env python
"""mapper.py"""

import sys

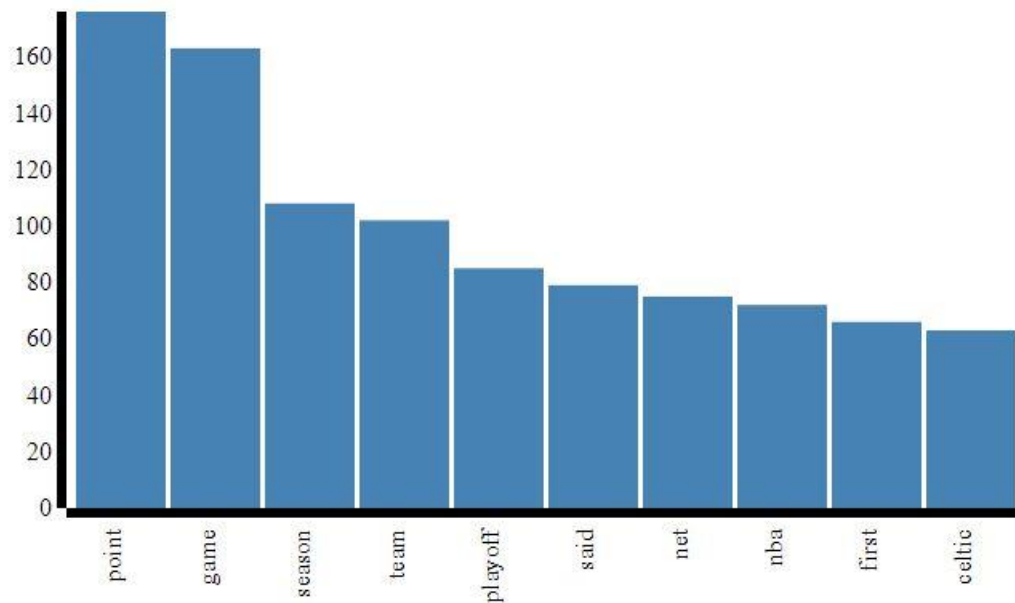
countDict = {}

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        if word in countDict:
            countDict[word]=countDict[word]+1
        else:
            countDict[word]=1
for countKey in countDict:
    print '%s\t%s' % (countKey, countDict[countKey])
```

```
mapper_wordcount.py x reducer_wordcount.py x
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[ ]:
5
6
7 #!/usr/bin/env python
8 """reducer.py"""
9
10 from operator import itemgetter
11 import sys
12
13 countDict = {}
14 current_word = None
15 current_count = 0
16 word = None
17
18 for line in sys.stdin:
19     line = line.strip()
20     word, count = line.split('\t', 1)
21     try:
22         count = int(count)
23     except ValueError:
24         continue
25     if current_word == word:
26         current_count += count
27     else:
28         if current_word:
29             countDict[current_word] = current_count
30             current_count = count
31             current_word = word
32 if current_word == word:
33     countDict[current_word] = current_count
34 countDict = sorted(countDict.items(), key=lambda kv: kv[1], reverse=True)
35 for i in countDict:
36     print '%s\t%s' % (i[0], i[1])
```

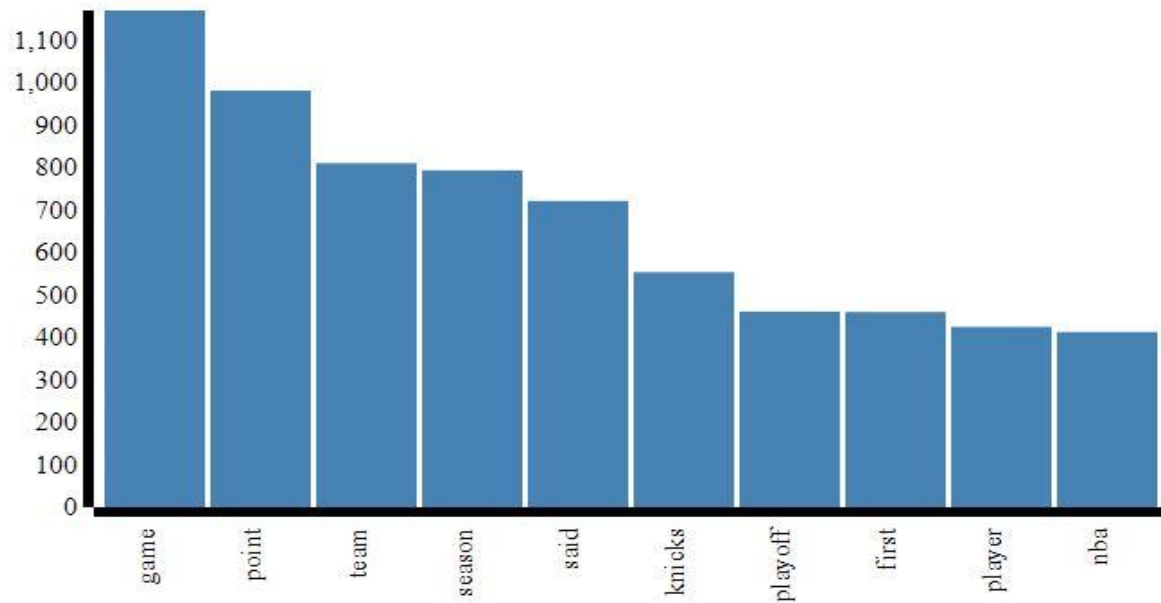

Task 2:

For the visualization part, we have drawn a bar graph for the top 10 words, obtained from Task 1.



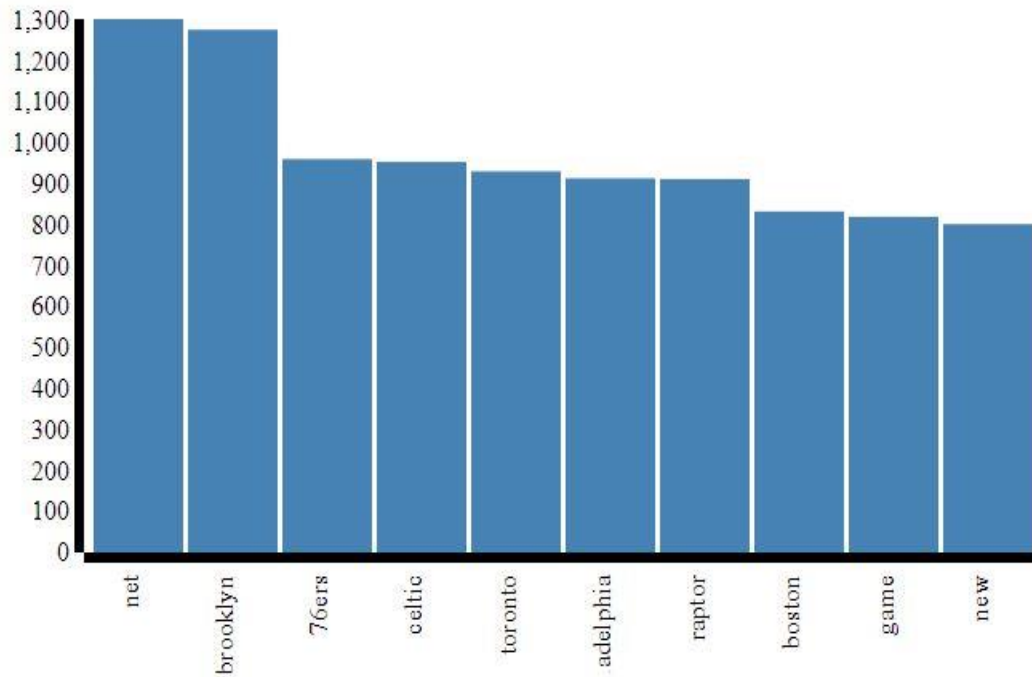
Task 3:

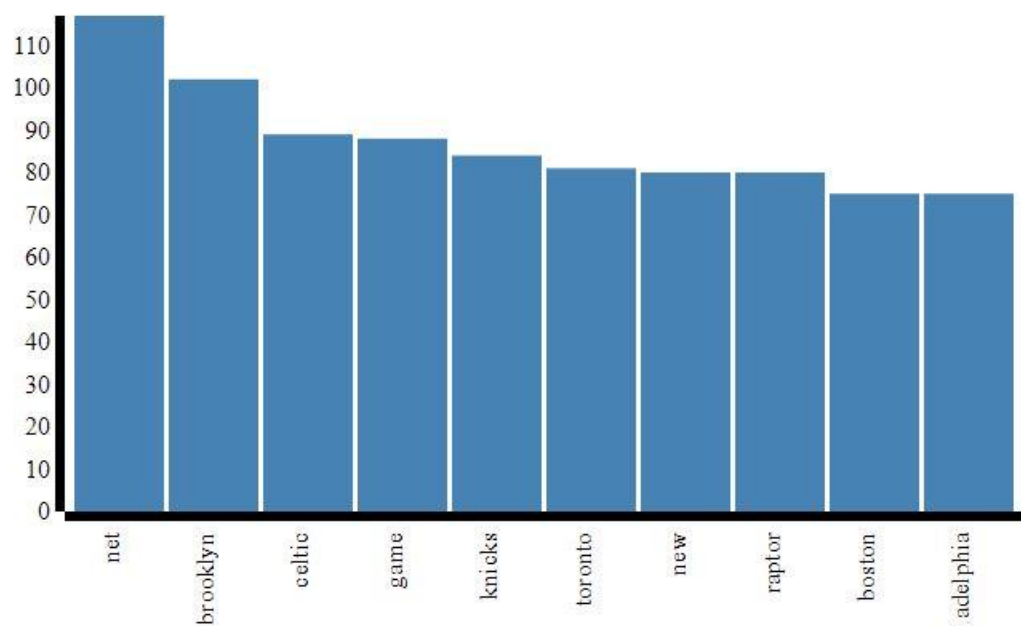
We created bar graphs for the corresponding large data.

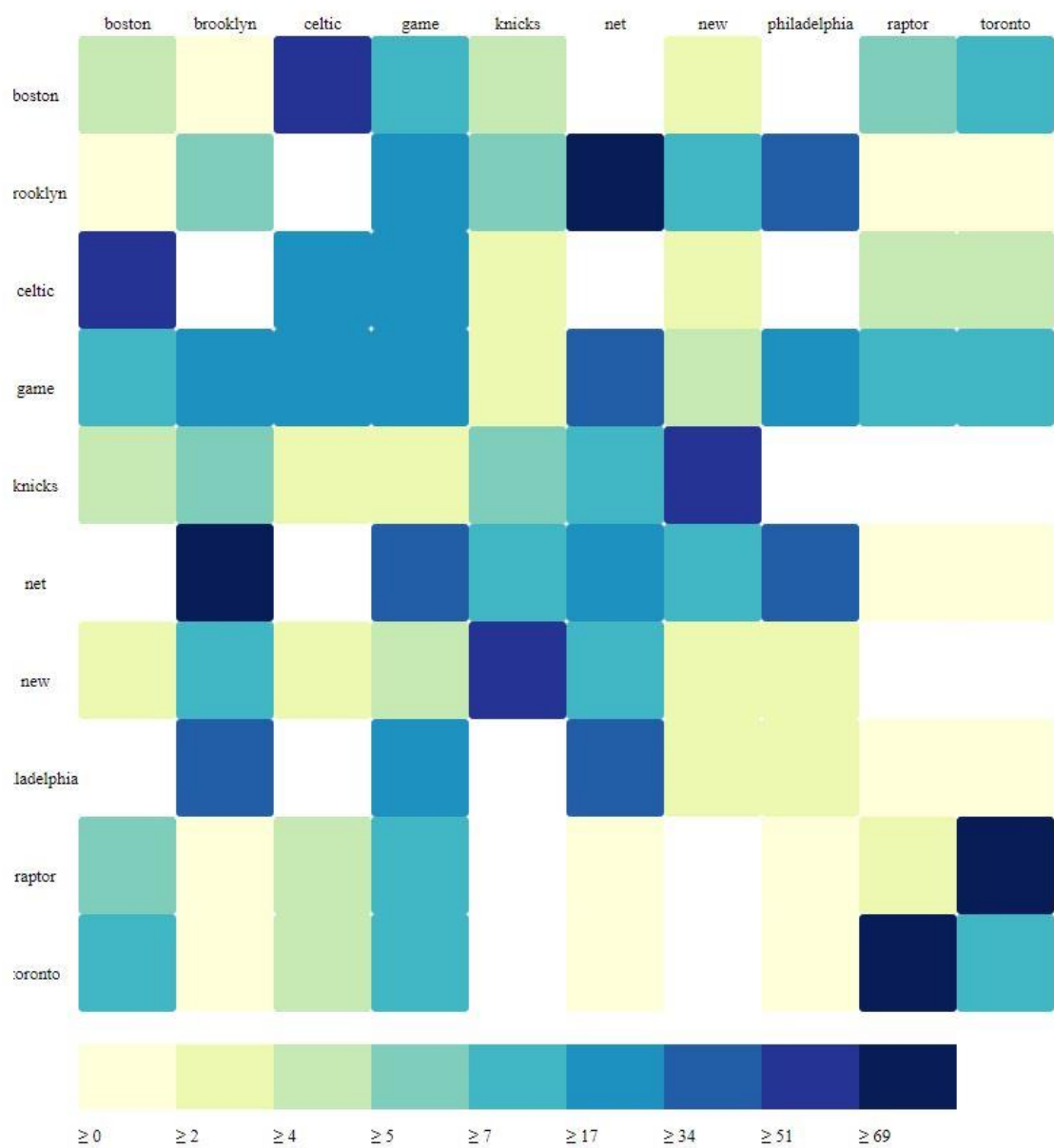


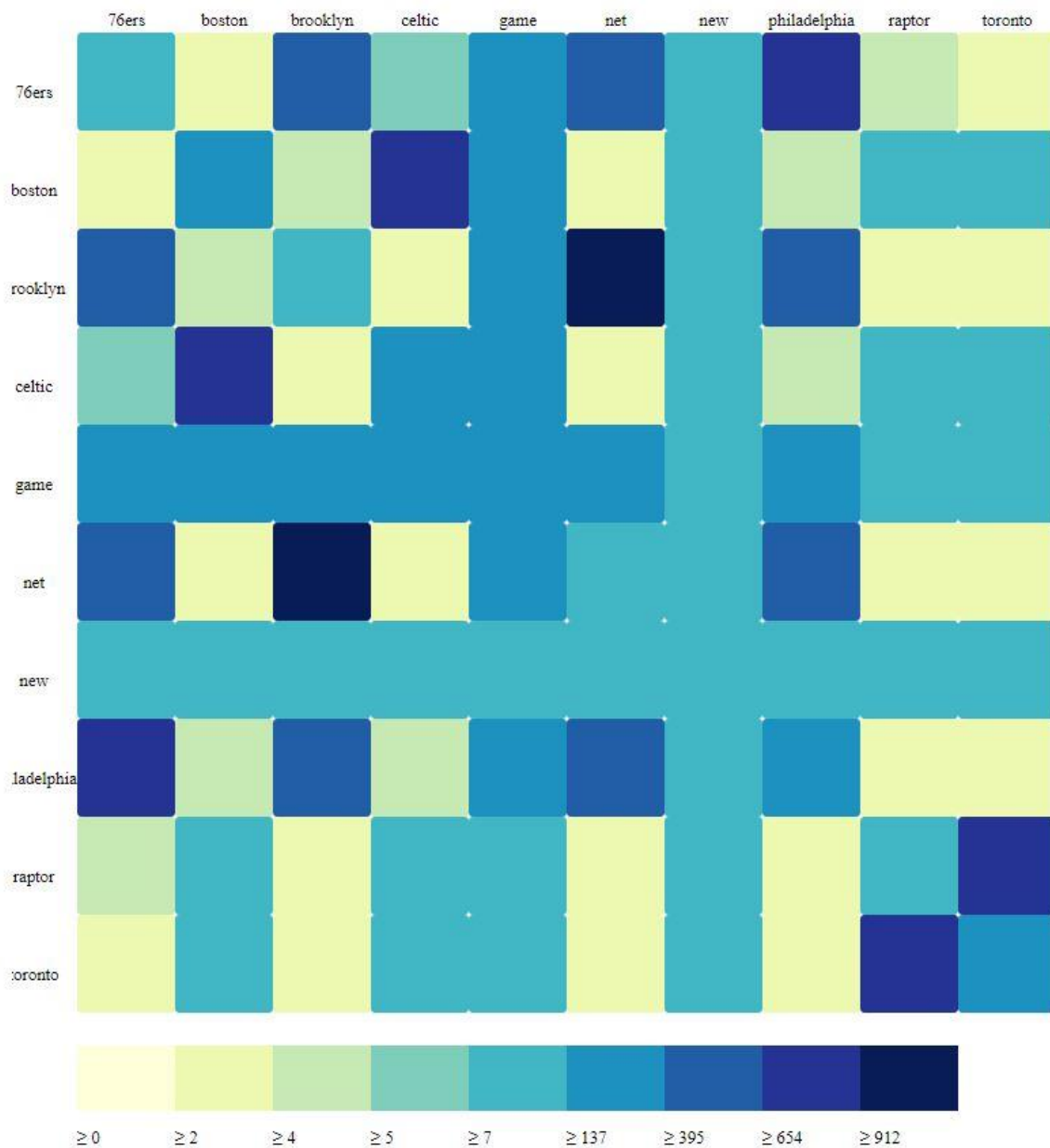
Task 4:

We built an interactive website using d3.js. We have provided 2 dropdown: One is the subtopic and the other being Data Source: Twitter, NyData and common crawl. For word count, we are plotting bar graphs. For word co-occurrence we are plotting heat map. Attached are the plots for subtopic – Atlantic division, data source – Twitter, data set size – Large and Small:









For building the website, we have to read from the text files. To achieve this, start chrome by enabling the cross origin request. Below is the code snippet for achieving this:

```
C:\Program Files (x86)\Google\Chrome\Application>start chrome.exe --allow-file-access-from-files
```

Task 5:

We took the top 10 words and we found the cooccurring words in the context. We repeated this for all the datasets (small and large) we have. Co-occurrence is a 10*10 square symmetric

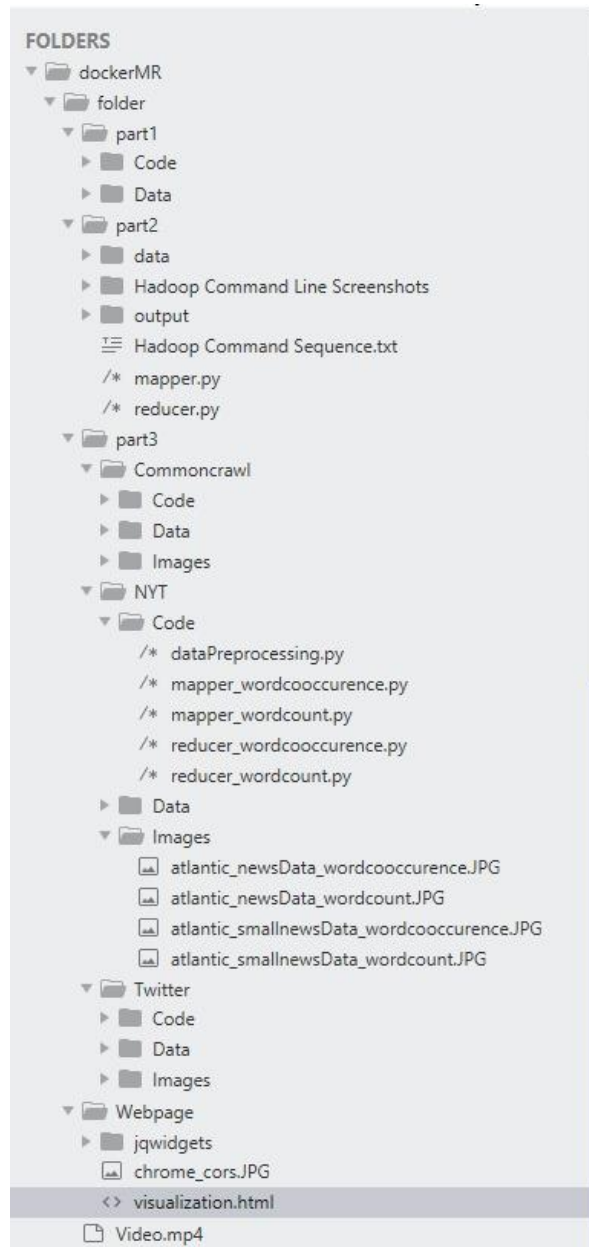
matrix. We passed the word count as the parameter for the mapper reducer and then we picked the top 10 words.

```
mapper_wordcooccurrence.py x reducer_wordcooccurrence.py x
```

```
#!/usr/bin/env python  
# coding: utf-8  
  
# In[ ]:  
  
#!/usr/bin/env python  
"""mapper.py"""  
  
import sys  
  
topWords = []  
i=1  
readConfigFile = open("southwest_scrawlData_wordcount.txt", "r")  
for readline in readConfigFile:  
    words = readline.split('\t', 1)  
    topWords.append(words[0])  
    i+=1  
    if(i==11):  
        break  
  
for line in sys.stdin:  
    line = line.strip()  
    words = line.split()  
    for iw, w in enumerate(words):  
        if w in topWords:  
            for iu, u in enumerate(words):  
                if (iw!=iu):  
                    if u in topWords:  
                        print '%s\t%s' % ("("+w+", "+u+")", 1)
```

```
mapper_wordcooccurrence.py x reducer_wordcooccurrence.py x
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[ ]:
5
6
7 #!/usr/bin/env python
8 """reducer.py"""
9
10 from operator import itemgetter
11 import sys
12
13 current_word = None
14 current_count = 0
15 word = None
16
17 for line in sys.stdin:
18     line = line.strip()
19     word, count = line.split('\t', 1)
20     try:
21         count = int(count)
22     except ValueError:
23         continue
24     if current_word == word:
25         current_count += count
26     else:
27         if current_word:
28             print '%s\t%s' % (current_word, current_count)
29             current_count = count
30             current_word = word
31 if current_word == word:
32     print '%s\t%s' % (current_word, current_count)
33
34
```

Task 6: Below is the directory structure we have used to save the files.



Task 7: We have attached the video of the demo where both of us have explained the steps we have used to achieve the result.

Conclusion:

In Conclusion, we visualized the word count and word co-occurrences for different topics from different sources. We also found that for a given topic, there were similar patterns occurring in word counts and word co-occurrences across the 3 different data sources. There is not a 100% match, but a lot of similarities can be seen. Especially between the NY Times data and Common Crawler data – understandably as they are similar sources by nature. This can be viewed clearly from the visualization on our website.

Reference:

1. www.d3js.org
2. <http://www.dealingdata.net/2016/07/23/PoGo-Series-Tweepy/>
3. <https://pythonprogramming.net/introduction-scraping-parsing-beautiful-soup-tutorial/>
4. <https://codeburst.io/web-scraping-101-with-python-beautiful-soup-bb617be1f486>
5. <https://www.bellingcat.com/resources/2015/08/13/using-python-to-mine-common-crawl/>
6. <https://dlab.berkeley.edu/blog/scraping-new-york-times-articles-python-tutorial>
7. <http://docs.tweepy.org/en/v3.5.0/>
8. <https://github.com/akshay993/Data-Analytics-Using-Apache-Spark/tree/master/Part2/code/dataCollection>