

Big Data Programming

Final Report

Title:

Twitter Web series Data Visualization with Spark ETL

Team Members:

- Avinash Ganguri
- Akhil Teja Kanugolu
- Geetanjali Makineni
- Bhashitha Siddareddy

Github link: <https://github.com/avinashganguri/Analysis-on-web-series>

Introduction:

Twitter Data Analysis with Spark ETL and visualizing is our title. Here we are taking the Twitter's Web series data. The main objective of our project is doing the ETL process using the Spark's Batch Processing and then Spark Integration using Web UI. The main source of our data is twitter and then collecting the data with Spark Batch Process. We can perform our transactions on the set of RDD's and later we load our data in our Hive which is similarly equal to the ETL basic process. This is because we are living in a world where data handling and data using plays one most important role in making the decisions for most of the industries

Background:

1. We first collect the tweets from the Twitter api.
2. Later, we import the data that is collected into the hive from the HDFS.
3. Next we will export this data from there into RDBMS by usage of sqoop.
4. We later do the sentimental analysis on the tweets that are collected.

5. Now, we are using spark sql for writing required queries later by visualizing (examples like bar graphs and pie charts and some other type of graphs) the obtained results by using panda or Tableau.

Goals and Objectives

Motivation:

The motivation behind doing this analysis is we are living in a world where data handling and data using plays one most important role in making the decisions for most of the industries like Banking, Financial, Telecom and Health and IT sector serving ones. The main factors would be for getting the insights would be like managing its sheer volumes of data and its insights. Using the Apache Spark is one of the best amazing kind frameworks which will be handling big data and its real time performance of these analysis.

Objectives:

The main objective of our project is doing the ETL process using the Spark's Batch Processing and then Spark Integration using Web UI. The main source of our data is twitter and then collecting the data with Spark Batch Process. We can perform our transactions on the set of RDD's and later we load our data in our Hive which is similarly equal to the ETL basic process.

Features:

The main feature of the project is to collect the Real timed tweets from the twitter steaming API, also by performing the ETL which means we preprocess the data and extract the necessary data and then we load this data in our HIVE. Later, we feed the data into our HDFS and then we implement SQL and Hive queries. Sqoop is used for transferring data between SQL and HDFS.

Significance:

Analyzing of sentimental analysis is done using one tool which is already existing named as ML tool which is TextBlob for prediction of sentiment on the tweets and Later we are using spark for writing the queries by visualizing with Panda.

Model Design:

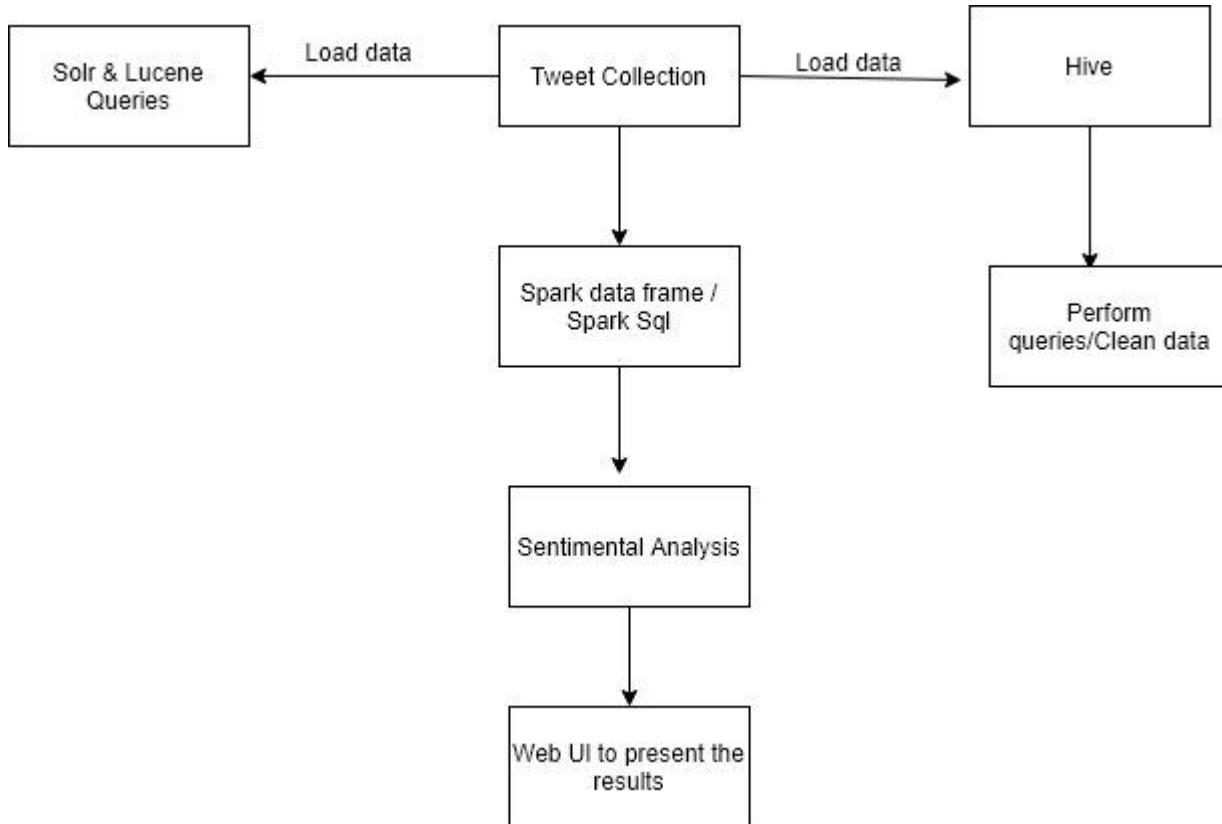


Fig: Architecture / Workflow Diagram

We have now first created one twitter developer account where next we got the tokens and credentials from API from where we took the twitter streaming and tweepy API in the python and stored tweets into the db. Extracted around 2.5 GB of dataset. Saved the data file in json format. Preprocess the data and create the database in spark framework with Scala programming. Will be creating a data frame and use the Sql Context to execute the SQL queries. In our next phases we are implementing it on Solrs and Lucene on our data and we also creating spark data frames on the file and creating different actions on data frames. We finally do the sentiment analysis on our data.

Dataset:

We collect the data from Twitter with API with a developer account credentials. Here we took the keywords as Netflix, TV shows, etc., These tweets are in JSON file format. We have collected nearly like 2.5GB size. It has an information that shows like web series, tv series, documentary, Netflix shows, tv shows. We are collecting here using an API which is in batch format which is nearly downloading like around 5kb for a second where we filter the keywords like Netflix shows, TV shows etc.,

Feature	Description
id	Unique id of user
created at	Tweet created time stamp in UTC
text	UTF-8 text Tweet data
source	Type of device used to post the tweet
Name	Name of the user
Place	Geo location of user at time of tweet posted
Screen_name	Profile or Screen name of the user
Lang	Language opted by user
user location	Location of the user
user_followers_count	Count of followers of user
user_friends_count	Count of friends of user
user_favourites_count	Count of people favorited the tweet
reply_count	Count of replies to the tweet
retweet_count	Count of retweets of the tweet

Table: Description of Attributes of Data

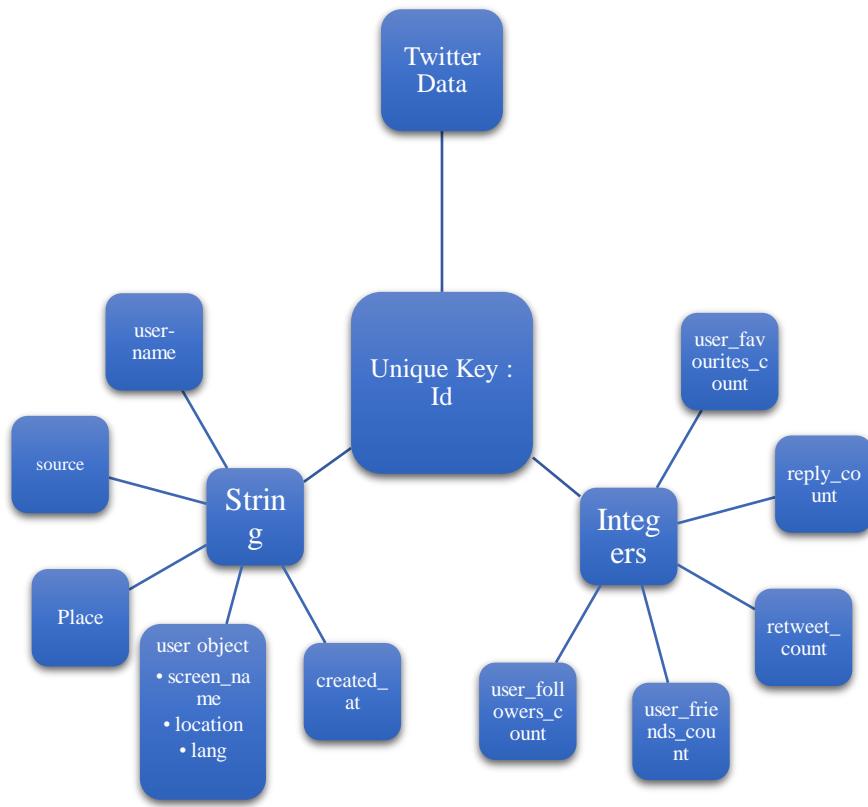


Figure: Design of Features/ Attributes

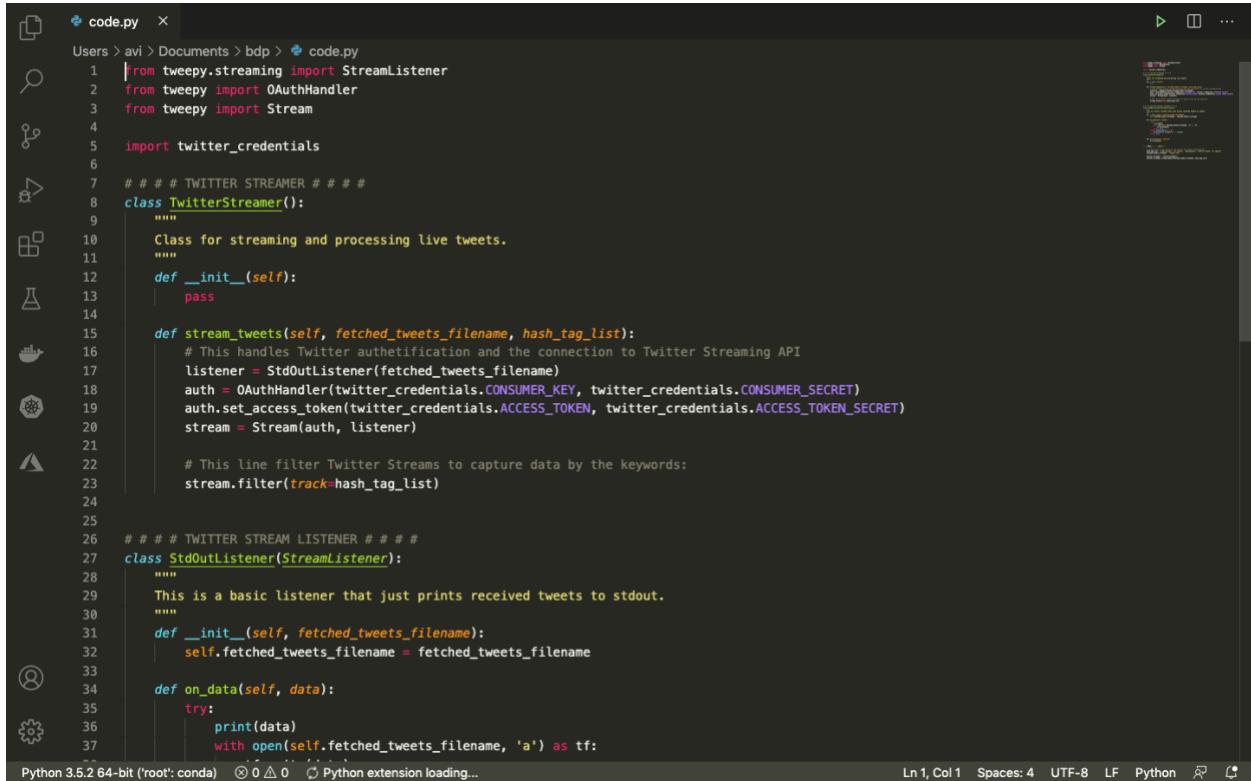
Analysis:

We have now first created one twitter developer account where next we got the tokens and credentials from API from where we took the twitter streaming and tweepy API in the python and stored tweets into the db. Extracted around 2.5 GB of dataset. Saved the data file in json format. Preprocess the data and create the database in spark framework with scala programming. Will be creating a data frame and use the Sql Context to execute the SQL queries.

Implementation:

Tweets collected:

- First, we got a developer access from the Twitter. After that we have created one app to use those API's.
- The code for which twitter streaming and tweepy for collected tweets is below:



The screenshot shows a code editor window with a dark theme. The file being edited is named 'code.py'. The code implements a Twitter Streamer and a StdOutListener. The Streamer class handles authentication and connects to the Twitter Streaming API to filter tweets by keywords. The StdOutListener prints received tweets to standard output. The code uses Python's built-in print function and the tweepy library's Stream class. The code editor interface includes a sidebar with icons for file operations like copy, paste, and search, and a status bar at the bottom indicating the Python version (3.5.2), the root environment ('root:conda'), and the current file ('code.py').

```
code.py  x
Users > avi > Documents > bdp > code.py
1  from tweepy.streaming import StreamListener
2  from tweepy import OAuthHandler
3  from tweepy import Stream
4
5  import twitter_credentials
6
7  # # # TWITTER STREAMER # # #
8  class TwitterStreamer():
9      """
10         Class for streaming and processing live tweets.
11     """
12     def __init__(self):
13         pass
14
15     def stream_tweets(self, fetched_tweets_filename, hash_tag_list):
16         # This handles Twitter authentication and the connection to Twitter Streaming API
17         listener = StdOutListener(fetched_tweets_filename)
18         auth = OAuthHandler(twitter_credentials.CONSUMER_KEY, twitter_credentials.CONSUMER_SECRET)
19         auth.set_access_token(twitter_credentials.ACCESS_TOKEN, twitter_credentials.ACCESS_TOKEN_SECRET)
20         stream = Stream(auth, listener)
21
22         # This line filter Twitter Streams to capture data by the keywords:
23         stream.filter(track=hash_tag_list)
24
25
26  # # # TWITTER STREAM LISTENER # # #
27  class StdOutListener(StreamListener):
28      """
29          This is a basic listener that just prints received tweets to stdout.
30      """
31      def __init__(self, fetched_tweets_filename):
32          self.fetched_tweets_filename = fetched_tweets_filename
33
34      def on_data(self, data):
35          try:
36              print(data)
37              with open(self.fetched_tweets_filename, 'a') as tf:
```

Python 3.5.2 64-bit ('root':conda) ⑧ 0 △ 0 ⚡ Python extension loading... Ln 1, Col 1 Spaces: 4 UTF-8 LF Python ⚡ ⚡

```

20     stream = Stream(auth, listener)
21
22     # This line filter Twitter Streams to capture data by the keywords:
23     stream.filter(track=hash_tag_list)
24
25
26     # # # TWITTER STREAM LISTENER # # #
27     class StdOutListener(StreamListener):
28         """
29             This is a basic listener that just prints received tweets to stdout.
30         """
31
32         def __init__(self, fetched_tweets_filename):
33             self.fetched_tweets_filename = fetched_tweets_filename
34
35         def on_data(self, data):
36             try:
37                 print(data)
38                 with open(self.fetched_tweets_filename, 'a') as tf:
39                     tf.write(data)
40                     return True
41             except BaseException as e:
42                 print("Error on_data %s" % str(e))
43             return True
44
45         def on_error(self, status):
46             print(status)
47
48
49     if __name__ == '__main__':
50
51         # Authenticate using config.py and connect to Twitter Streaming API.
52         hash_tag_list = ["web series", "tv series", "documentary", "netflix shows", "tv shows"]
53         fetched_tweets_filename = "tweets.txt"
54
55         twitter_streamer = TwitterStreamer()
56         twitter_streamer.stream_tweets(fetched_tweets_filename, hash_tag_list)

```

Python 3.5.2 64-bit ('root':conda) ⊗ 0 △1 Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 🔍 🔍

Spark using scala:

Starting the Spark-shell,

```

spark session available as `spark`.
Welcome to


$$\begin{array}{c} \diagup \diagdown \\ \diagdown \diagup \end{array} \begin{array}{c} \diagup \diagdown \\ \diagdown \diagup \end{array}$$
 version 2.2.0

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_232)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc);
warning: there was one deprecation warning; re-run with -deprecation for details
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@129348e8
|
scala> import sqlContext.implicits._
import sqlContext.implicits._

scala> val tweetstable = sqlContext.read.json("/project/data.json");

```

Query 1:

Number of Tweets that are from different countries on web series.

```
scala> val q1 = sqlContext.sql("SELECT place.country,count(*) AS count FROM tweetDatatable GROUP BY place.country ORDER BY count DESC limit 10");
q1: org.apache.spark.sql.DataFrame = [country: string, count: bigint]

scala> q1.show():
+-----+-----+
|      country| count|
+-----+-----+
|      null|628413|
| United States| 1798|
| United Kingdom| 228|
| Republic of the P...| 199|
|        India| 121|
| South Africa| 110|
|     Nigeria| 84|
|      Canada| 70|
|    Malaysia| 61|
|   Australia| 55|
+-----+-----+
```

Query 2:

We are here fetching the count of tweets based on the co-ordinates of web series.

We got the information based on the longitude and latitude.

```
scala> val q2=sqlContext.sql("select count(*) as count,coordinates.coordinates[0] as long, coordinates.coordinates[1] as lat from tweetDatatable where coordinates is not null group by coordinates order by count desc limit 10");
q2: org.apache.spark.sql.DataFrame = [count: bigint, long: double ... 1 more field]

scala> q2.show():
+-----+-----+
|count|      long|      lat|
+-----+-----+
|  3| -122.419| 37.7793|
|  3| -99.2161568| 19.5412983|
|  2|  -73.771| 40.7636|
|  2| -57.9410622| -34.8941991|
|  2|  -95.3694| 29.7602|
| 2|151.22666692|-33.89321579|
| 2|  -118.1938| 33.7692|
| 2|  -3.05| 53.55|
| 2| -74.212552| 11.24228945|
| 2|  -74.0794| 4.5997|
+-----+-----+
```

Query 3:

To fetch the different languages in which the tweets are made about web series.

We got the count of number of tweets about the web series in separate languages.

```

scala> val q3=sqlContext.sql("select count(*) as count,lang as language from tweetDatatable where lang is not null group by lang");
q3: org.apache.spark.sql.DataFrame = [count: bigint, language: string]

scala> q3.show();
+-----+-----+
| count|language|
+-----+-----+
|277255|    en|
|   28|     vi|
|    5|     ne|
|   12|     sl|
|   17|     ro|
| 4446|    und|
|   18|     ur|
|   11|     lv|
|  926|     pl|
| 1947|     pt|
| 3501|     tl|
| 3163|     in|
| 2510|     ko|
|   40|     uk|
|   51|     cs|
|  844|     tr|
|  715|     de|
|   15|     is|
|    1|     sd|
| 5477|     es|
+-----+-----+
only showing top 20 rows

```

Query 4:

Fetching the users account names with a greater number of tweets on web series.

```

scala> val q4=sqlContext.sql("SELECT count(*) as count, user.name from tweetDatatable where user.name is not null group by user.name order by count desc limit 10");
q4: org.apache.spark.sql.DataFrame = [count: bigint, name: string]

scala> q4.show();
+-----+-----+
|count| name|
+-----+-----+
| 703|  .|
| 357|  _|
| 265|  ☀|
| 261|  W|
| 202|  ✨|
| 174|  🌟|
| 148|  -|
| 148|  J|
| 148| Alex|
| 136|Chris|
+-----+-----+

```

Query 5:

Fetching the users that have more followers who are tweeting based on web series.

```

scala> val q5b = sqlContext.sql("SELECT user.name, max(user.followers_count) as followers_count, user.lang FROM tweetDatatable WHERE text like '%series%' group by user.name,user.lang order by followers_count desc limit 15");
q5b: org.apache.spark.sql.DataFrame = [name: string, followers_count: bigint ... 1 more field]

scala> q5b.show();
+-----+-----+
|      name|followers_count|lang|
+-----+-----+
|  El Universal|      5222178|null|
| Daily Express|       807418|null|
|  El Periódico|       639298|null|
|  Eurogamer|       371051|null|
| Datos 365 🇪🇸|      253316|null|
| GrandesMedios.com|      178922|null|
| Norman Buffong|       168711|null|
|  El Ilustrativo 🇪🇸|      115106|null|
| Evening Standard|       105733|null|
| La Publicación 🇪🇸|      101659|null|
| Postmediático 🇪🇸|       96189|null|
| Don Síndóptico 🇪🇸|      88010|null|
| Difusión 365 🇪🇸|       82541|null|
|      nah|        74567|null|
| La Cronología 🇪🇸|       69748|null|
+-----+-----+

```

Query 6:

Fetching the number of tweets based on different web series.

```
scala> val q6=sqlContext.sql("select count(*) as count,q.text from (select case when text like '%prison break%' then 'prison break' when text like '%black mirror%' then 'black mirror' when text like '%crown%' then 'crown' when text like '%arrow%' then 'arrow' WHEN text like '%flash%' THEN 'flash' WHEN text like '%Game of Thrones%' THEN 'Game of Thrones' WHEN text like '%breaking bad%' THEN 'breaking bad' else 'different series' end as text from tweetDatatable)q group by q.text");
q6: org.apache.spark.sql.DataFrame = [count: bigint, text: string]
```

```
scala> q6.show();
+-----+
| count|      text|
+-----+
|     4| black mirror|
|     1| prison break|
| 631535|different series|
|     7| breaking bad|
|    78|         arrow|
|    39|         crown|
|     1| Game of Thrones|
|    61|         flash|
+-----+
```

Query 7:

Fetching the non-verified users who are having more number of followers.

```
scala> val q7 = sqlContext.sql("SELECT user.verified,user.screen_name,max(user.followers_count) as followers_count FROM tweetDatatable WHERE user.verified = false GROUP BY user.vi
erified,user.screen_name ORDER BY followers_count DESC LIMIT 15");
q7: org.apache.spark.sql.DataFrame = [verified: boolean, screen_name: string ... 1 more field]
```

```
scala> q7.show();
+-----+
|verified| screen_name|followers_count|
+-----+
|  false| ThelifeDiaries|      3449579|
|  false| jugsjugsjugs|      3097534|
|  false| FactSoup|      2981871|
|  false| jascurtissmith|      2514660|
|  false| Fact|      2213698|
|  false| AngTanongKoSayo|      1719366|
|  false| QuoteBeauties|      1653735|
|  false| ScientificIdeas|      1469749|
|  false| cachaito235|      1424677|
|  false| GabbarSingh|      1329213|
|  false| TheInteresting|      1288611|
|  false| PoemPorns|      1286668|
|  false| smolAnimalsvid|      897755|
|  false| gspot1177|      895129|
|  false| UrbanEnglisch|      824574|
+-----+
```

Query 8:

Users with a greater number of retweets for their tweets on web series.

```
scala> val q8 = sqlContext.sql("SELECT user.screen_name, text,retweeted_status.retweet_count FROM tweetDatatable ORDER BY retweeted_status.retweet_count DESC LIMIT 20");
q8: org.apache.spark.sql.DataFrame = [screen_name: string, text: string ... 1 more field]

scala> q8.show();
+-----+-----+
| screen_name | text | retweet_count |
+-----+-----+
| SoudantMarc|RT @bricheesey: ... | 388878|
| nickelassy|RT @tanjirolove: ... | 223728|
| SenatorBaddest|RT @tanjirolove: ... | 223727|
| CatPanzer|RT @humorandanima... | 189611|
| seburop0824|RT @humorandanima... | 189610|
| Koerin72|RT @humorandanima... | 189609|
| lovechangchen|RT @humorandanima... | 189609|
| shay_osowski|RT @humorandanima... | 189609|
| Caffier_Lilian|RT @humorandanima... | 189606|
| FredAlert69|RT @humorandanima... | 189605|
| 8686jeep0|RT @humorandanima... | 189605|
| peacectruhope|RT @humorandanima... | 189604|
| wavinaApr|RT @humorandanima... | 189604|
| RoulaSanty|RT @bellatheboss... | 186545|
| mariahfairfield|RT @bellatheboss... | 186543|
| anand3zzz|RT @bellatheboss... | 186540|
| winglesssangel|RT @bellatheboss... | 186535|
| lukiezra|RT @bellatheboss... | 186532|
| sawyfab1|RT @queentrashcan... | 179741|
| saianachasey|RT @Clearly: whe... | 124027|
+-----+-----+
```

Query 9:

Number of Tweets that are on basis of different locations in the United States on web series.

```
scala> val q9=sqlContext.sql("SELECT user.location,count(text) as count FROM tweetDatatable WHERE place.country='United States' AND user.location is not null GROUP BY user.location ORDER BY count DESC LIMIT 15");
q9: org.apache.spark.sql.DataFrame = [location: string, count: bigint]

scala> q9.show();
+-----+-----+
| location | count |
+-----+-----+
| Los Angeles, CA | 65|
| California, USA | 35|
| Houston, TX | 27|
| Chicago, IL | 19|
| Dallas, TX | 14|
| Las Vegas, NV | 12|
| Texas, USA | 12|
| San Antonio, TX | 12|
| San Francisco, CA | 11|
| Atlanta, GA | 11|
| San Diego, CA | 10|
| Florida, USA | 10|
| Portland, OR | 9|
| Brooklyn, NY | 9|
| New York, NY | 9|
+-----+-----+
```

Query 10:

Fetching the top tweeted text and checking if someone re tweeted it on web series.

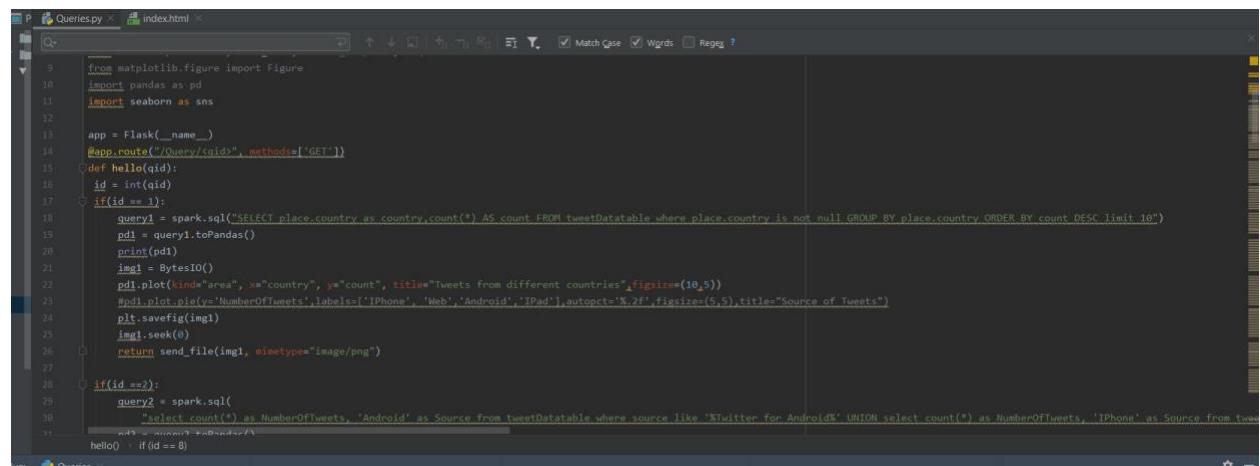
```
scala> val q10 = sqlContext.sql("SELECT user.name ,retweeted_status.text AS Retweet_Text,retweeted_status.retweet_count AS Retweet_Count FROM tweetDatatable WHERE retweeted_status.retweet_count IS NOT NULL ORDER BY retweeted_status.retweet_count DESC limit 10");
q10: org.apache.spark.sql.DataFrame = [name: string, Retweet_Text: string ... 1 more field]

scala> q10.show();
+-----+-----+
|      name|  Retweet_Text|Retweet_Count|
+-----+-----+
| soudant marc|My boyfriend lite...|      388878|
| ələçinə əzizlər evə...|my friends hyping...|      223728|
| Bobby McNasty|my friends hyping...|      223727|
| ねこパン|nala, an autism s...|      189611|
| ねこパン|nala, an autism s...|      189610|
| アドラー|nala, an autism s...|      189609|
| R0|nala, an autism s...|      189609|
| Shay Osowski|nala, an autism s...|      189609|
| Happy|nala, an autism s...|      189606|
| all.k|nala, an autism s...|      189605|
+-----+-----+
```

Spark using python: (Considered python programming as the execution time is much faster compared to scala dataframes – Visualization is done based on the python programming with Matplotlib)

Query 1:

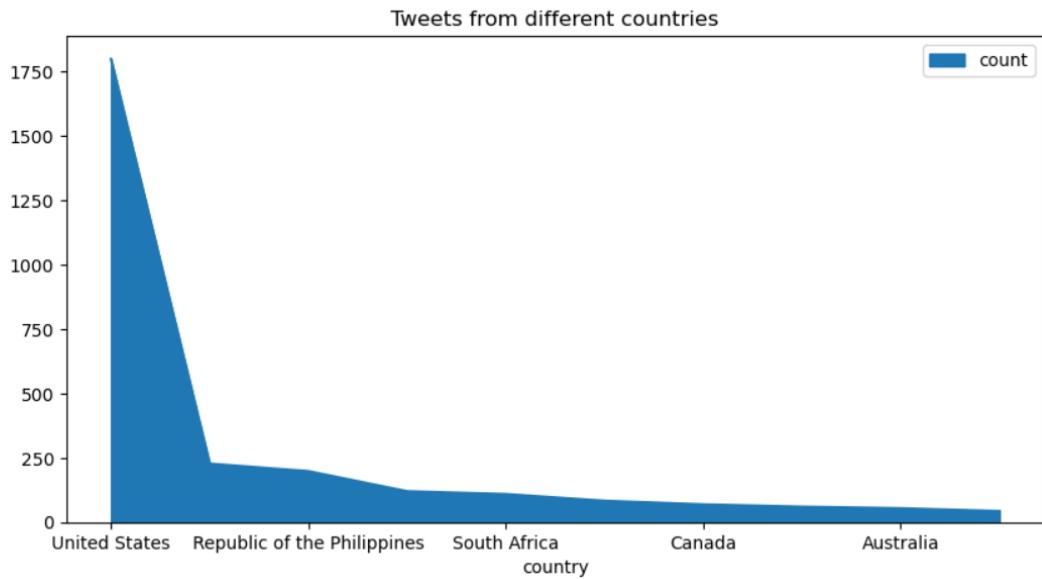
Counted the number of tweets based on the place and arranged in descending order.



```
from matplotlib.figure import Figure
import pandas as pd
import seaborn as sns

app = Flask(__name__)
@app.route('/query/', methods=['GET'])
def hello(qid):
    id = int(qid)
    if(id == 1):
        query1 = spark.sql("SELECT place.country as country, count(*) AS count FROM tweetDatatable WHERE place.country is not null GROUP BY place.country ORDER BY count DESC limit 10")
        query1 = query1.toPandas()
        pd1 = BytesIO()
        print(pd1)
        img1 = BytesIO()
        pd1.plot(kind="area", x="country", y="count", title="Tweets from different countries", figsize=(10,5))
        #pd1.plot.pie(y='NumberOfTweets', labels=['iPhone', 'Web', 'Android', 'IPad'], autopct='%2f', figsize=(5,5), title="Source of Tweets")
        plt.savefig(img1)
        img1.seek(0)
        return send_file(img1, mimetype="image/png")

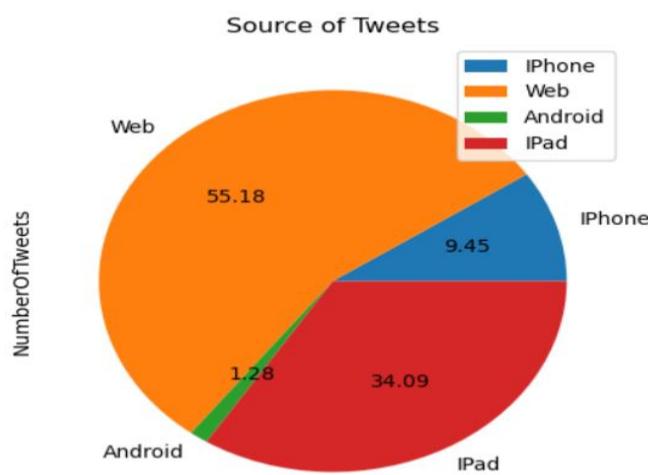
    if(id == 2):
        query2 = spark.sql(
            "select count(*) as NumberOfTweets, 'Android' as Source from tweetDatatable where source like '%Twitter for Android%' UNION select count(*) as NumberOfTweets, 'iPhone' as Source from tweetDatatable where source like '%Twitter for iPhone%'"
        )
        hello() - if(id == 0)
```



Query2:

Counted the no.of tweets based on the sources of data by performing the Union operation.

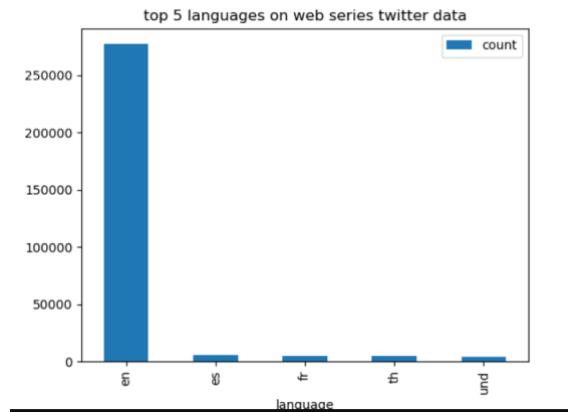
```
if(id ==2):
    query2 = spark.sql(
        "select count(*) as NumberOfTweets, 'Android' as Source from tweetDatatable where source like '%Twitter for Android%' UNION select count(*) as NumberOfTweets, 'iPhone' as Source from tweetDatatable where source like '%Twitter for iPhone%'")
    pd2 = query2.toPandas()
    img1 = BytesIO()
    pd2.plot.pie(y='NumberOfTweets', labels=['iPhone', 'Web', 'Android', 'IPad'], autopct='%.2f', figsize=(5, 5),
                  title="Source of Tweets")
    plt.savefig(img1)
    img1.seek(0)
    return send_file(img1, mimetype="image/png")
```



Query3:

Counted the tweets by grouping into languages and order in descending order with pulling top 5.

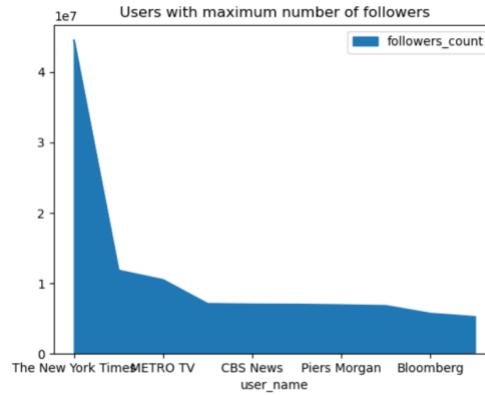
```
if(id == 3):
    query3 = spark.sql("select count(*) as count,lang as language from tweetDatatable where lang is not null group by lang order by count desc limit 5")
    print(query3)
    pd3 = query3.toPandas()
    pd3.plot(kind="bar", x="language", y="count", title="top 5 languages on web series twitter data")
    img3 = BytesIO()
    plt.savefig(img3)
    img3.seek(0)
    return send_file(img3, mimetype="image/png")
```



Query4:

Pulled the top 10 user_name by followers_count by dropping the duplicates

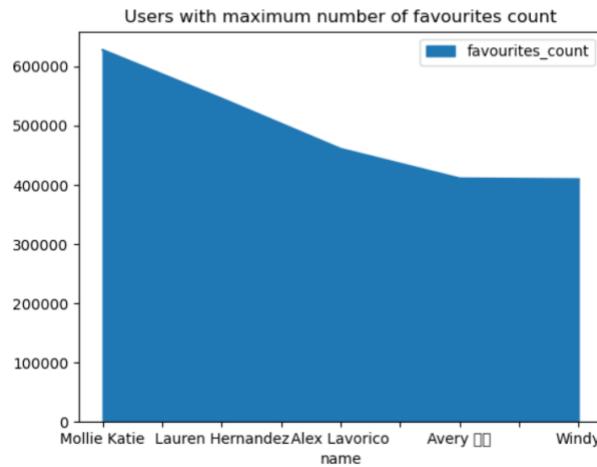
```
if(id == 4):
    query4 = spark.sql(
        "select user.name as user_name,user.followers_count as followers_count from tweetDatatable order by user.followers_count desc limit 10").dropDuplicates()
    pd4 = query4.toPandas()
    pd4.plot.area(x = "language",y="tweets",data=pd4)
    pd4.plot.area(x='user_name',y='followers_count',title="Users with maximum number of followers")
    img4 = BytesIO()
    plt.savefig(img4)
    img4.seek(0)
    return send_file(img4, mimetype="image/png")
```



Query5:

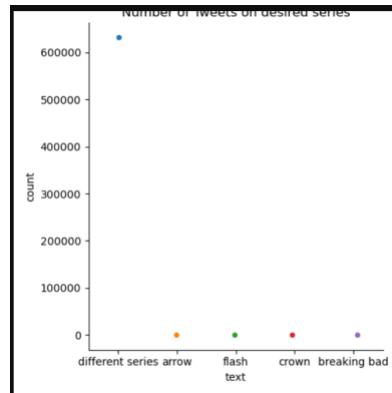
Based on top5 maximum favourites_count of tweets by user_name

```
if id == 5:
    query5 = spark.sql(
        "SELECT user.name as name, max(user.favourites_count) as favourites_count FROM tweetDatatable WHERE text like '%series%' group by user.name order by favourites_count desc limit 5")
    pd5 = query5.toPandas()
    pd5.plot.area(x="name", y="favourites_count", title="Users with maximum number of favourites count")
    img5 = BytesIO()
    plt.savefig(img5)
    img5.seek(0)
    return send_file(img5, mimetype="image/png")
```



Query6:

Pulling the count of tweets based on the desired series from the text attribute.



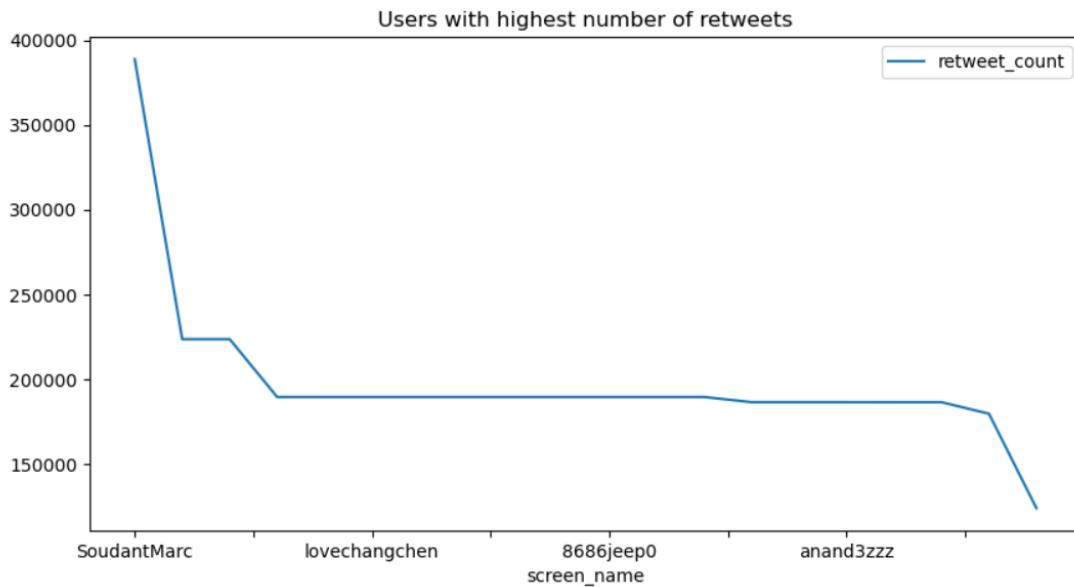
Query7:

Ordered the count of tweets of top 20 for the screen name.

```

if(id == 8):
    query8 = spark.sql(
        "SELECT user.screen_name AS screen_name, text,retweeted_status.retweet_count AS retweet_count FROM tweetDatatable ORDER BY retweeted_status.retweet_count DESC LIMIT 20")
    # query8.createOrReplaceTempView("df1")
    # subQuery8 =spark.sql(
    #     "SELECT name,followers_count FROM (SELECT *, MAX(followers_count) OVER (PARTITION BY name) AS maxB FROM df1) M WHERE followers_count = maxB")
    pd8 = query8.toPandas()
    pd8.plot(x="screen_name", y="retweet_count", title='Users with highest number of retweets', figsize=(10,5))
    img8 = BytesIO()
    plt.savefig(img8)
    img8.seek(0)
    return send_file(img8, mimetype="image/png")

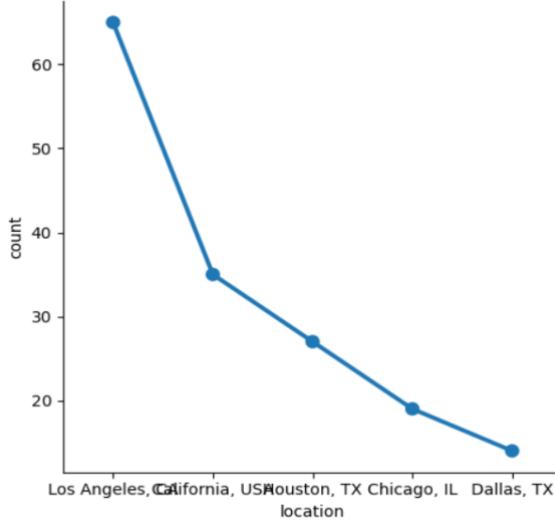
```



Query8:

Collected the top5 places in United states.

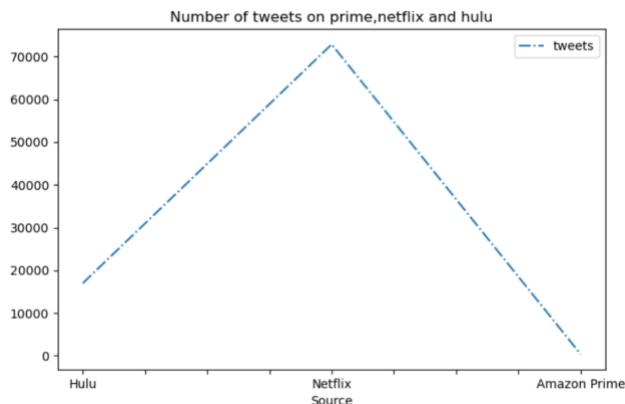
```
if(id == 9):
    query9 = spark.sql(
        "SELECT user_location as location, count(text) as count FROM tweetDatatable WHERE place.country='United States' AND user_location is not null GROUP_BY user_location ORDER_BY count DESC LIMIT 5")
    pd9 = query9.toPandas()
    #pd9.plot.pie(y="count", labels=pd9.location, autopct='%2f', figsize=(5,5), title="Source of Tweets")
    sns.catplot(x="location", y="count", kind="point", data=pd9, title="Top 5 places with highest number of tweets")
    img9 = BytesIO()
    plt.savefig(img9)
    img9.seek(0)
    return send_file(img9, mimetype="image/png")
```



Query9:

Count of tweets for Netflix, Hulu, prime

```
query10 = spark.sql(
    "select count(*) as tweets, 'Hulu' as Source from tweetDatatable where text like '%hulu%' or text like '%Hulu%' UNION select count(*) as tweets, 'Netflix' as Source from tweetDatatable
    pd10 = query10.toPandas()
    pd10.plot(linestyle='--', title="Number of tweets on prime,netflix and hulu", x='Source', y='tweets', figsize=(8,5))
    #pd3.plot(kind="bar", x="Party", y="tweets", title="Republicans and Democratic tweets with 100% positivity")
    img10 = BytesIO()
    plt.savefig(img10)
    img10.seek(0)
    return send_file(img10, mimetype="image/png")
```



Solr:

Started solr with configuration 6.6 and JDK 8.

Created the core name with samplecore

```
C:\Windows\System32\cmd.exe
ERROR: Solr at https://localhost:8983/solr did not come online within 30 seconds!

C:\solr\solr-6.6.2\bin>solr start
Waiting up to 30 to see Solr running on port 8983
Started Solr server on port 8983. Happy searching!

C:\solr\solr-6.6.2\bin>solr create -c samplecore

ERROR:
Core 'samplecore' already exists!
Checked core existence using Core API command:
https://localhost:8983/solr/admin/cores?action=STATUS&core=samplecore

C:\solr\solr-6.6.2\bin>
```

Converted the JSON file to CSV using python which makes loading data to solr easy.

```
1  import ...
2  reload(sys)
3  sys.setdefaultencoding('utf8')
4
5  tweets = []
6  for line in open('teams.json', 'r'):
7      try:
8          tweets.append(json.loads(line))
9      except:
10         pass
11
12     # tweets.append(json.loads(line))
13
14     fileName = "solr_data.csv"
15     fields = ['created_at', 'id', 'text', 'source', 'user_name', 'user_location', 'user_followers_count',
16               'user_friends_count', 'user_favourites_count', 'user_statuses_count', 'quote_count', 'reply_count',
17               'retweet_count', 'favorite_count']
18
19     with open(fileName, 'wb') as newFile:
20         csvwriter = csv.writer(newFile)
21
22         # writing the fields
23         csvwriter.writerow(fields)
24         for tweet in tweets:
25             rows = [tweet['created_at'], tweet['id'], tweet['text'], tweet['source'], tweet['user']['name'],
26                     tweet['user']['location'], tweet['user']['followers_count'], tweet['user']['friends_count'],
27                     tweet['user']['favourites_count'], tweet['user']['statuses_count'], tweet['quote_count'],
28                     tweet['reply_count'], tweet['retweet_count'], tweet['favorite_count']]
29             print row
30             csvwriter.writerow(row)
31
32     with open(fileName, 'wb') as ne...
33
34
35
```

Later edited the schema.xml with the attributes of csv file.

```
<field name="_text_" type="text_general" multiValued="true" indexed="true" stored="false"/>
<field name="_version_" type="long" indexed="false" stored="false"/>
<field name="created_at" type="strings"/>
<field name="favorite_count" type="tlongs"/>
<field name="id" type="string" multiValued="false" indexed="true" required="true" stored="true"/>
<field name="quote_count" type="tlongs"/>
<field name="reply_count" type="tlongs"/>
<field name="retweet_count" type="tlongs"/>
<field name="source" type="strings"/>
<field name="text" type="strings"/>
<field name="user_favourites_count" type="tlongs"/>
<field name="user_followers_count" type="tlongs"/>
<field name="user_friends_count" type="tlongs"/>
<field name="user_location" type="strings"/>
<field name="user_name" type="strings"/>
<field name="user_statuses_count" type="tlongs"/>
<dynamicField name="*_txt_en_split_tight" type="text_en_splitting_tight" indexed="true" stored="true"/>
<dynamicField name="* descendant path" type="descendant path" indexed="true" stored="true"/>
```

Overview after loading the csv data to solr.

Solr Admin

localhost:8983/solr/#/samplecore

Statistics

Last Modified: about an hour ago
Num Docs: 168680
Max Doc: 330773
Heap Memory Usage: -1
Deleted Docs: 162093
Version: 526
Segment Count: 30

Optimized: optimize now
Current:

Instance

CWD: C:\solr\solr-6.6.2\server
Instance: C:\solr\solr-6.6.2\server\solr\samplecore
Data: C:\solr\solr-6.6.2\server\solr\samplecore\data
Index: C:\solr\solr-6.6.2\server\solr\samplecore\data\index
Impl: org.apache.solr.core.NRTCachingDirectoryFactory

Replication (Master)

	Version	Gen	Size
Master (Searching)	1607279769776	7	203.38 MB
Master (Replicable)	1607279769776	7	-

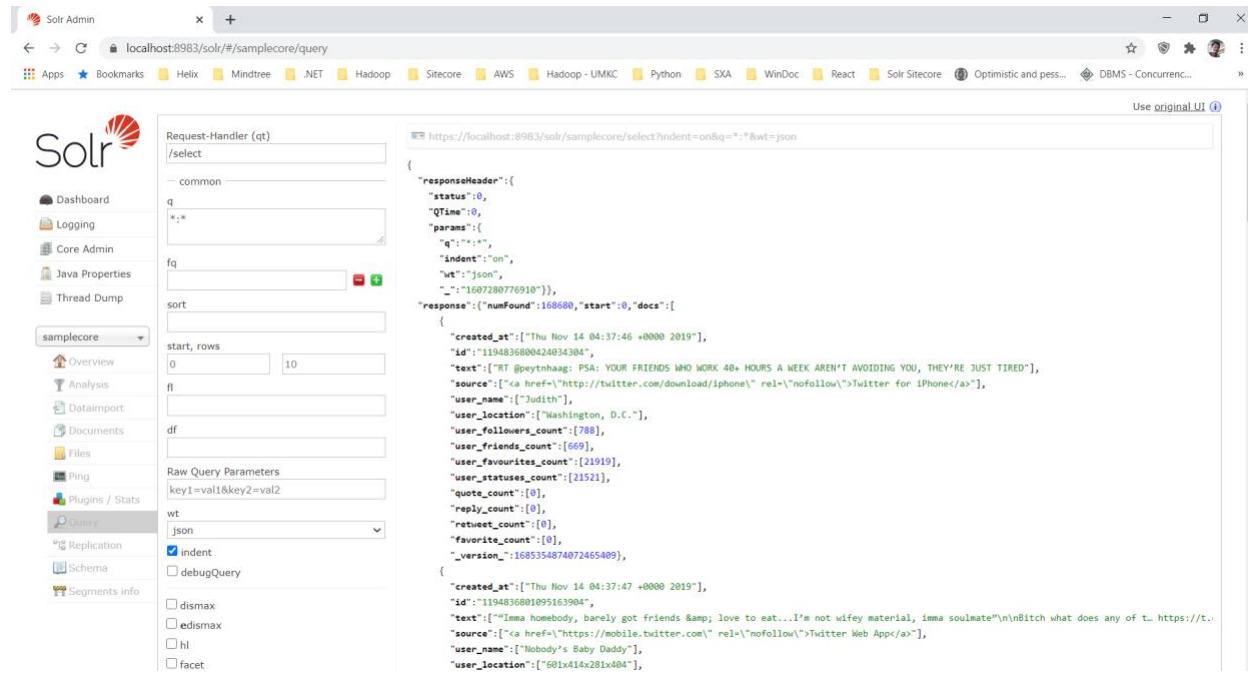
Healthcheck

Ping request handler is not configured with a healthcheck file.

Documentation Issue Tracker IRC Channel Community forum Solr Query Syntax

Query1:

Pulled the 10 records of data in json form.

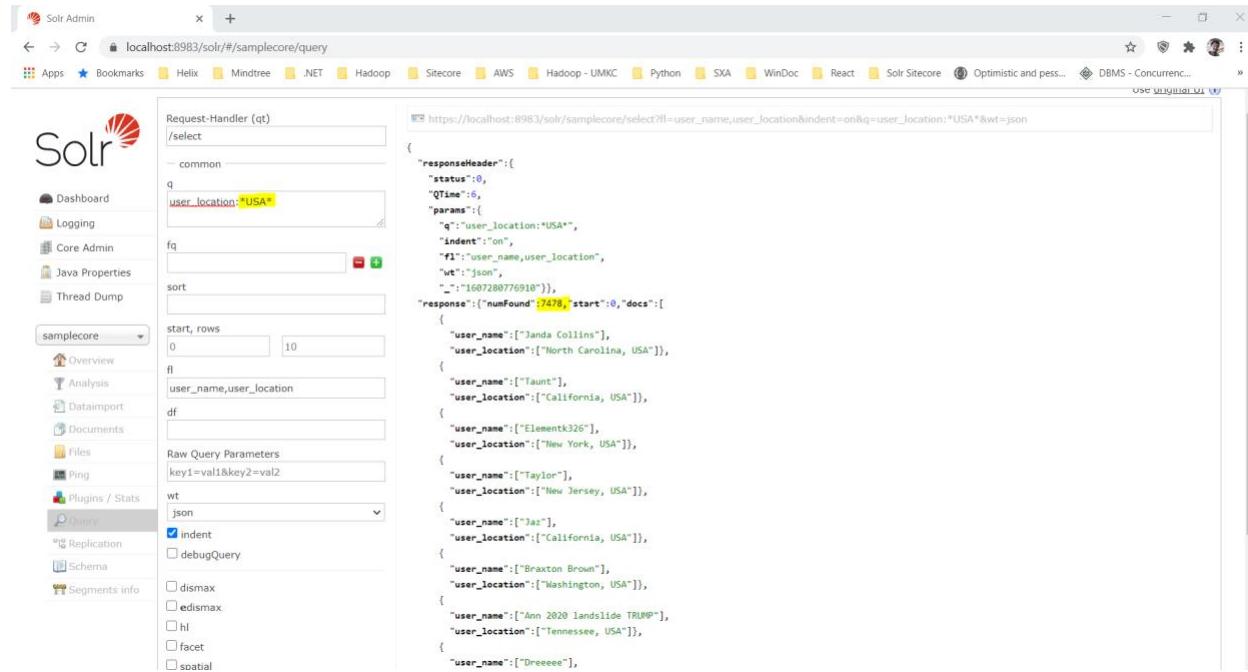


The screenshot shows the Solr Admin interface with the URL `localhost:8983/solr/#/samplecore/query`. The Request-Handler (qt) dropdown is set to /select. The query parameters include `q:*`, `fq`, `sort`, `start,rows:0,10`, and `wt:json`. The `indent` checkbox is checked. The response header shows a status of 0 and QTime of 0. The response body contains 168680 results, starting with two documents:

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 0,
    "params": {
      "q": "*",
      "indent": "on",
      "wt": "json",
      "_": "1607280776910"
    },
    "response": {"numFound": 168680, "start": 0, "docs": [
      {
        "created_at": ["Thu Nov 14 04:37:46 +0000 2019"],
        "id": "1104836800424034304",
        "text": ["RT @peytnhaag: PSA: YOUR FRIENDS WHO WORK 40+ HOURS A WEEK AREN'T AVOIDING YOU, THEY'RE JUST TIRED"],
        "source": ["<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for iPhone</a>"],
        "user_name": ["Judith"],
        "user_location": ["Washington, D.C."],
        "user_followers_count": [788],
        "user_friends_count": [669],
        "user_favourites_count": [21919],
        "user_statuses_count": [21521],
        "quote_count": [0],
        "reply_count": [0],
        "retweet_count": [0],
        "favorite_count": [0],
        "_version_": 1685354874072465489
      },
      {
        "created_at": ["Thu Nov 14 04:37:47 +0000 2019"],
        "id": "1104836801095163984",
        "text": ["Imma homebody, barely got friends & love to eat...I'm not wifey material, imma soulmate\n\nBitch what does any of t.. https://t.co/..."],
        "source": ["<a href=\"https://mobile.twitter.com\" rel=\"nofollow\">Twitter Web App</a>"],
        "user_name": ["Nobody's Baby Daddy"],
        "user_location": ["601x414x281x404"]
      }
    ]}}
}
```

Query2:

Pulled the data with user_location has USA (Regex)



The screenshot shows the Solr Admin interface with the URL `localhost:8983/solr/#/samplecore/query`. The Request-Handler (qt) dropdown is set to /select. The query parameters include `q:user_location:.*USA.*`, `fq`, `sort`, `start,rows:0,10`, and `wt:json`. The `indent` checkbox is checked. The response header shows a status of 0 and QTime of 6. The response body contains 74378 results, starting with several documents:

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 6,
    "params": {
      "q": "user_location:.*USA.*",
      "indent": "on",
      "fl": "user_name,user_location",
      "wt": "json",
      "_": "1607280776910"
    },
    "response": {"numFound": 74378, "start": 0, "docs": [
      {
        "user_name": ["Janda Collins"],
        "user_location": ["North Carolina, USA"]
      },
      {
        "user_name": ["Taunt"],
        "user_location": ["California, USA"]
      },
      {
        "user_name": ["Element326"],
        "user_location": ["New York, USA"]
      },
      {
        "user_name": ["Taylor"],
        "user_location": ["New Jersey, USA"]
      },
      {
        "user_name": ["Ja"],
        "user_location": ["California, USA"]
      },
      {
        "user_name": ["Braxton Brown"],
        "user_location": ["Washington, USA"]
      },
      {
        "user_name": ["Ann 2020 landslide TRUMP!"],
        "user_location": ["Tennessee, USA"]
      },
      {
        "user_name": ["Dreeeee"]
      }
    ]}}
}
```

Query3:

Pulled the data has Netflix with in text (Regex)

The screenshot shows the Solr Admin interface with the 'samplecore' core selected. In the 'Request-Handler (qt)' section, the query is set to '/select' and the 'q' field contains 'text:netflix*'. The results page displays a JSON response with two tweets from users @karinadacpano and @emilly. Both tweets mention 'netflix n chill? hulu n hang?? no!!! DISNEY PLUS N THRUST'.

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 5,
    "params": {
      "q": "text:netflix*",
      "indent": "on",
      "wt": "json",
      "_": "1607280776910"
    }
  },
  "response": {
    "numFound": 8589,
    "start": 0,
    "docs": [
      {
        "created_at": ["Thu Nov 14 04:37:53 +0000 2019"],
        "id": "1194836826394988544",
        "text": ["RT @karinadacpano: netflix n chill? hulu n hang?? no!!! DISNEY PLUS N THRUST"],
        "source": ["<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for iPhone</a>"],
        "user_name": ["alyssa"],
        "user_followers_count": [262],
        "user_friends_count": [259],
        "user_favourites_count": [19721],
        "user_statuses_count": [4035],
        "quote_count": [0],
        "reply_count": [0],
        "retweet_count": [0],
        "favorite_count": [0],
        "_version_": "1685354874130137094"
      },
      {
        "created_at": ["Thu Nov 14 04:37:53 +0000 2019"],
        "id": "1194836828051738624",
        "text": ["RT @karinadacpano: netflix n chill? hulu n hang?? no!!! DISNEY PLUS N THRUST"],
        "source": ["<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for iPhone</a>"],
        "user_name": ["emilly"],
        "user_location": ["getting pulled over somewhere"],
        "user_followers_count": [282]
      }
    ]
  }
}
```

Query4:

Using fuzzy search of user_favourites greater than 7000 and text having hulu.

The screenshot shows the Solr Admin interface with the 'samplecore' core selected. In the 'Request-Handler (qt)' section, the query is set to '/select' and the 'q' field contains 'user_favourites_count:[7000 TO *] AND text:hulu'. The results page displays a JSON response with three tweets from users @karinadacpano, @Drew, and @Zob. All three tweets mention 'hulu' and have user_favourites_count greater than 7000.

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 165,
    "params": {
      "q": "user_favourites_count:[7000 TO *] AND text:hulu",
      "indent": "on",
      "fl": "user_name,text,user_followers_count,user_friends_count,user_favourites_count",
      "wt": "json",
      "_": "1607280776910"
    }
  },
  "response": {
    "numFound": 2329,
    "start": 0,
    "docs": [
      {
        "text": ["RT @karinadacpano: netflix n chill? hulu n hang?? no!!! DISNEY PLUS N THRUST"],
        "user_name": ["alyssa"],
        "user_followers_count": [262],
        "user_friends_count": [259],
        "user_favourites_count": [19721],
        {
          "text": ["RT @karinadacpano: netflix n chill? hulu n hang?? no!!! DISNEY PLUS N THRUST"],
          "user_name": ["emilly"],
          "user_followers_count": [282],
          "user_friends_count": [146],
          "user_favourites_count": [29717]
        },
        {
          "text": ["RT @karinadacpano: netflix n chill? hulu n hang?? no!!! DISNEY PLUS N THRUST"],
          "user_name": ["Drew"],
          "user_followers_count": [329],
          "user_friends_count": [315],
          "user_favourites_count": [7219]
        },
        {
          "text": ["RT @karinadacpano: netflix n chill? hulu n hang?? no!!! DISNEY PLUS N THRUST"],
          "user_name": ["Zob"],
          "user_followers_count": [475],
          "user_friends_count": [315],
          "user_favourites_count": [29717]
        }
      }
    ]
  }
}
```

Query5:

Collected the response with having text:"prime" and prime:"love"

The screenshot shows the Solr Admin interface with the 'samplecore' core selected. The 'Query' tab is active. In the 'Request-Handler (qt)' dropdown, '/select' is chosen. The 'q' field contains the query 'text:"prime" OR text:"love"'. The 'wt' field is set to 'json'. The results pane displays the JSON response for two tweets. The first tweet is from '@will_prime' with the text 'Drunk me telling all my friends how much I love them'. The second tweet is from '@bombon' with the text 'Drunk me telling all my friends how much I love them'.

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 31,
    "params": {
      "q": "text:\"prime\" OR text:\"love\"",
      "indent": "on",
      "wt": "json",
      "_": "1607283778170"
    }
  },
  "response": {
    "numFound": 6006,
    "start": 0,
    "docs": [
      {
        "created_at": ["Thu Nov 14 04:37:53 +0000 2019"],
        "id": "1194836829113061576",
        "text": ["@will_prime: Drunk me telling all my friends how much I love them https://t.co/5xEkRGNFHm"],
        "source": [<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>],
        "user_name": ["marce"],
        "user_location": ["hnx"],
        "user_followers_count": [742],
        "user_friends_count": [712],
        "user_favourites_count": [64923],
        "user_statuses_count": [39988],
        "quote_count": [0],
        "reply_count": [0],
        "retweet_count": [0],
        "favorite_count": [0],
        "_version_": 1685354874137477120
      },
      {
        "created_at": ["Thu Nov 14 04:53:30 +0000 2019"],
        "id": "1194840760155777408",
        "text": ["RT @will_prime: Drunk me telling all my friends how much I love them https://t.co/5xEkRGNFHm"],
        "source": [<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>],
        "user_name": ["bombon ❤️"],
        "user_location": ["houston tx"]
      }
    ]
  }
}
```

Query 6:

Proximity query: Pulled the data having user_name with prince with proximity 3.

The screenshot shows the Solr Admin interface with the 'samplecore' core selected. The 'Query' tab is active. In the 'Request-Handler (qt)' dropdown, '/select' is chosen. The 'q' field contains the query 'user_name:"prince"~3'. The 'wt' field is set to 'json'. The results pane displays the JSON response for two tweets. The first tweet is from '@Nighly_night' with the text 'Nighty night ❤️'. The second tweet is from '@Espritdupetitprince' with the text 'Esprit du petit prince ❤️'.

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 39,
    "params": {
      "q": "user_name:\"prince\"~3",
      "indent": "on",
      "wt": "json",
      "_": "1607283778170"
    }
  },
  "response": {
    "numFound": 155,
    "start": 0,
    "docs": [
      {
        "created_at": ["Thu Nov 14 04:38:09 +0000 2019"],
        "id": "1194836894691057664",
        "text": ["@Nighly_night ❤️"],
        "source": [<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>],
        "user_name": ["@Espritdupetitprince ❤️"],
        "user_location": ["Malesse, près de Rennes"],
        "user_followers_count": [1215],
        "user_friends_count": [4411],
        "user_favourites_count": [4560],
        "user_statuses_count": [3220],
        "quote_count": [0],
        "reply_count": [0],
        "retweet_count": [0],
        "favorite_count": [0],
        "_version_": 1685354874229751814
      },
      {
        "created_at": ["Thu Nov 14 04:38:18 +0000 2019"],
        "id": "1194836934650028033",
        "text": ["RT @netflix: This is the pep talk everyone deserves to hear from a best friend. https://t.co/sBUZK6uTyy"],
        "source": [<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>],
        "user_name": ["@princess_"]
      }
    ]
  }
}
```

Query 7:

Based on created_at attribute to pull the data before 2019.

The screenshot shows the Solr Admin interface with the URL `localhost:8983/solr/#/samplecore/query`. The left sidebar shows the navigation tree with the 'samplecore' core selected. The main area is titled 'Request-Handler (qt)' and contains a form for a 'select' query. The 'q' field contains the query `created_at:(* TO 2019)`. The 'wt' field is set to 'json'. The 'indent' checkbox is checked. The results pane displays the JSON response, which includes the count of 168660 documents and two sample document snippets. The first snippet is from a user named 'Judith' with ID 1194836800424834304, and the second is from a user named 'Nobody's Baby Daddy' with ID 1194836801095163984.

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 30,
    "params": {
      "q": "created_at:(* TO 2019)",
      "indent": "on",
      "wt": "json",
      "_": "1607283778170"
    },
    "response": {"numFound": 168660, "start": 0, "docs": [
      {
        "created_at": ["Thu Nov 14 04:37:46 +0000 2019"],
        "id": "1194836800424834304",
        "text": ["RT @seytnhaag: PSA: YOUR FRIENDS WHO WORK 40+ HOURS A WEEK AREN'T AVOIDING YOU, THEY'RE JUST TIRED"],
        "source": ["<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for iPhone</a>"],
        "user_name": ["Judith"],
        "user_location": ["Washington, D.C."],
        "user_followers_count": [788],
        "user_friends_count": [669],
        "user_favourites_count": [21919],
        "user_statuses_count": [21521],
        "quote_count": [0],
        "reply_count": [0],
        "retweet_count": [0],
        "favorite_count": [0],
        "_version_": 1685354874072465489
      },
      {
        "created_at": ["Thu Nov 14 04:37:47 +0000 2019"],
        "id": "1194836801095163984",
        "text": ["Imma homebody, barely got friends & love to eat...I'm not wifey material, imma soulmate\nBitch what does any of t.. https://t.co/edfjwv"]
      }
    ]}}
}
```

Hive

Query 1:

Viewing the number of tweets based on popular streaming service Netflix.

```
apache-hive-1.2.2-bin — java -Xmx256m -Dhadoop.log.dir=/Users/avi/hadoop-2.8.1/logs -Dhadoo...
Time taken: 2.443 seconds
hive> select count(*) from tweets where text like "%Netflix%" or text like "%netflix%"
";
Query ID = avi_20201206231035_784f7655-2dc7-4abe-9ef6-97f4fe1d2ea6
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2020-12-06 23:10:38,915 Stage-1 map = 0%,  reduce = 0%
2020-12-06 23:10:41,938 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1218719562_0001
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 397010424 HDFS Write: 198505212 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
99394
Time taken: 6.004 seconds, Fetched: 1 row(s)
hive> Display all 475 possibilities? (y or n)
hive>
```

Query 2:

Viewing the number of tweets based on popular streaming service Hulu.

```
apache-hive-1.2.2-bin -- java -Xmx256m -Dhadoop.log.dir=/Users/avi/hadoop-2.8.1/logs -Dhadoo...
99394
Time taken: 6.004 seconds, Fetched: 1 row(s)
hive> Display all 475 possibilities? (y or n)
[hive> select count(*) from tweets where text like "%Hulu%" or text like "%hulu%";      ]
Query ID = avi_20201206231140_a3a2bd0b-9a8a-402c-ae40-8553ec1b0bf2
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2020-12-06 23:11:42,559 Stage-1 map = 0%,  reduce = 0%
2020-12-06 23:11:44,575 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local230233791_0002
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 794020848 HDFS Write: 198505212 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
32422
Time taken: 3.62 seconds, Fetched: 1 row(s)
hive> █
```

Query 3:

Details of tweets regarding web series that have a greater favourites count.

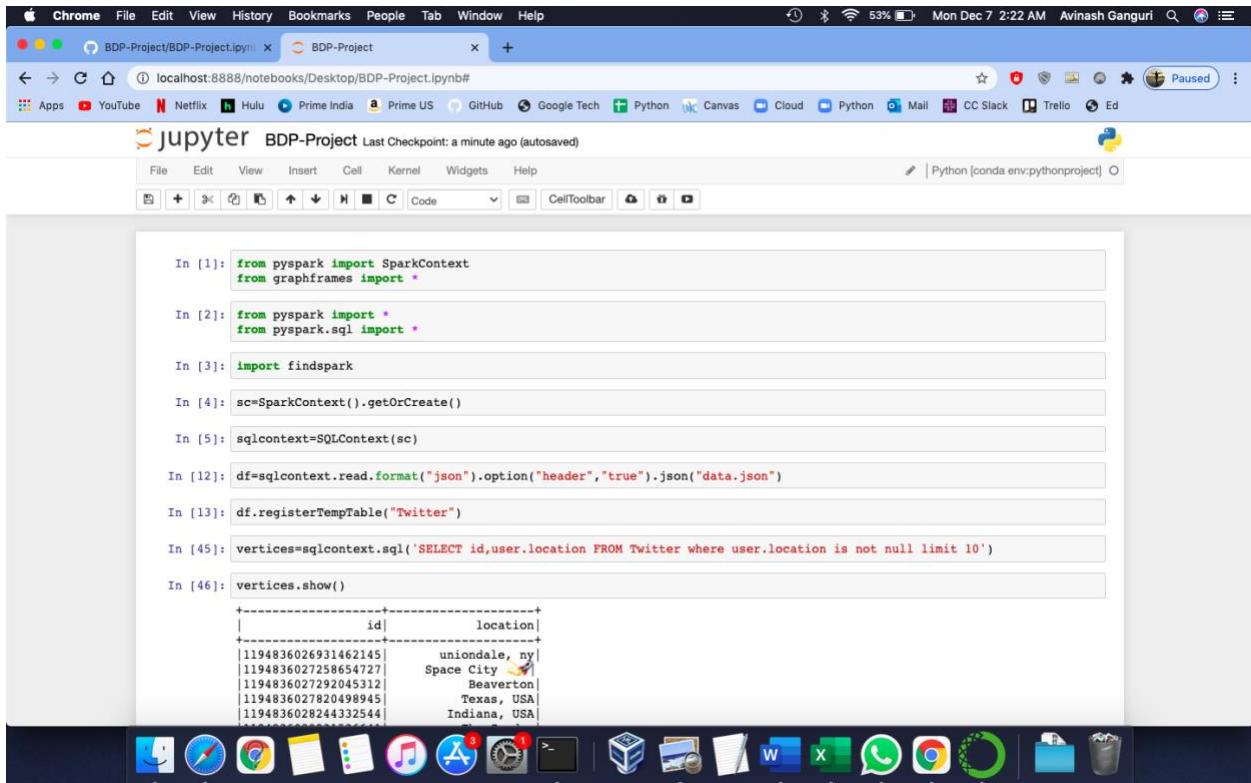
```
apache-hive-1.2.2-bin — java -Xmx256m -Dhadoop.log.dir=/Users/avi/hadoop-2.8.1/logs -Dhadoop...  
hive> select text,user_location,user_favourites_count as c from tweets order by c desc  
c limit 10;  
Query ID = avi_20201206231330_bd961448-4791-4fdf-87a9-9d3560de905c  
Total jobs = 1  
Launching Job 1 out of 1  
Number of reduce tasks determined at compile time: 1  
In order to change the average load for a reducer (in bytes):  
  set hive.exec.reducers.bytes.per.reducer=<number>  
In order to limit the maximum number of reducers:  
  set hive.exec.reducers.max=<number>  
In order to set a constant number of reducers:  
  set mapreduce.job.reduces=<number>  
Job running in-process (local Hadoop)  
2020-12-06 23:13:32,343 Stage-1 map = 0%,  reduce = 0%  
2020-12-06 23:13:34,358 Stage-1 map = 100%,  reduce = 0%  
2020-12-06 23:13:35,369 Stage-1 map = 100%,  reduce = 100%  
Ended Job = job_local1784895621_0004  
MapReduce Jobs Launched:  
Stage-Stage-1:  HDFS Read: 1389536484 HDFS Write: 397010424 SUCCESS  
Total MapReduce CPU Time Spent: 0 msec  
OK  
"Suhasini ji is taking body of her son to graveyard alone      KRK      5228063  
"As @disneyplus launches      "<a href=""https://www.echobox.com"" rel=""nofollow"">Echobox Social</a>"      3614296  
"Nickelodeon and Netflix have inked a new      Variety 2334130  
"Netflix      "<a href=""https://zapier.com/"" rel=""nofollow"">Zapier.com</a>"      1  
699961  
mother "<a href=""http://twitter.com/download/android"" rel=""nofollow"">Twitter for  
Android</a>"      1691484  
RightOnTV      48898      1482327  
"「さよならテレビ」、Netflix      "<a href=""https://mobile.twitter.com"" rel=""nofollow"">Twitter Web App</a>"      1451108  
Jeff Strong      3768      1421162
```

Query 4:

Details of users and their tweets about web series that has more number of followers count.

```
apache-hive-1.2.2-bin — java -Xmx256m -Dhadoop.log.dir=/Users/avi/hadoop-2.8.1/logs -Dhadoop.log.file=hadoop.log -...
hive> select user_name,text,user_followers_count from tweets order by user_followers_count desc limit 10;
Query ID = avi_20201206232724_0250e6e6-defc-4b1d-8a6a-a0eaf7a88852
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2020-12-06 23:27:25,995 Stage-1 map = 0%,  reduce = 0%
2020-12-06 23:27:28,010 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1355563580_0005
MapReduce Jobs Launched:
Stage-Stage-1: HDFS Read: 1588041696 HDFS Write: 397010424 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
<a href="http://twitter.com/download/iphone" rel=""nofollow"">Twitter for iPhone</a>      "RT @seew
hatsnext: Netflix and Nickelodeon have formed a multi-year deal to produce original animated feature f
ilms and television series    742617000027
<a href="http://twitter.com/download/android" rel=""nofollow"">Twitter for Android</a>      "RT @CKir
ubi: It is never easy my friends. To make it big    254732703661
The New York Times      Who lives in a pineapple under the sea? Spongebob Squarepants – who has a spi
noff headed to Netflix as the company... https://t.co/40j28qDUat 44503415
<a href="http://twitter.com/download/android" rel=""nofollow"">Twitter for Android</a>      "RT @peyt
nhaag: PSA: YOUR FRIENDS WHO WORK 40+ HOURS A WEEK AREN'T AVOIDING YOU    14112018
La Patilla      "Friends" negocia una reunión especial por su 25 aniversario https://t.co/i99AqrKVPT
                                . 7087663
barkha dutt      Can't wait to have you on @WeTheWomenAsia stage @Lisaraniray - friends in Mumbai block
your date and your seat for... https://t.co/ICTmXB7Gos 6984257
ELLE Magazine (US)      "Millie Bobby Brown hears your #StrangerThings fan theories and she's here for
Eleven as season 4's villain. ""I kind... https://t.co/fwD4TySgqz" 6795046
Bloomberg      Disney's market value is now double Netflix https://t.co/QIB032SbdI https://t.co/KLh8l
Yfnk2 5694810
El Universal      #Video Netflix y Nickelodeon informaron que llegaron a un acuerdo de producción de lar
gometrajes animados y series... https://t.co/axPb2lq8V0 5222178
El Universal      Netflix y Nickelodeon informaron que llegaron a un acuerdo de producción de largometra
jes animados y series de tele... https://t.co/DrratIzkf5 5222132
Time taken: 3.468 seconds, Fetched: 10 row(s)
hive>
```

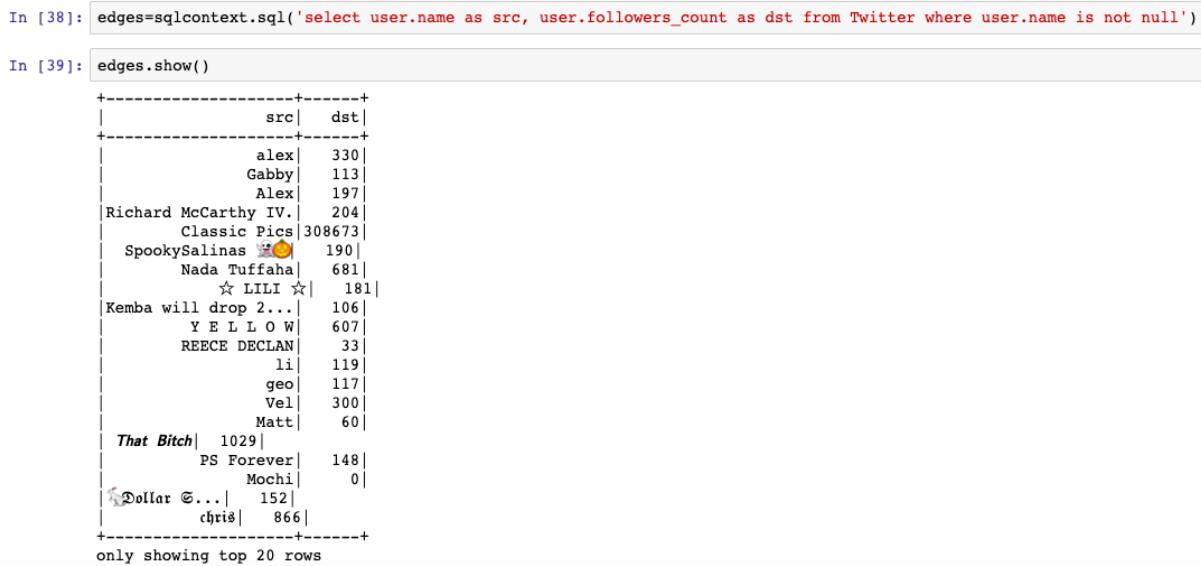
GraphFrames:



The screenshot shows a Jupyter Notebook interface within a Chrome browser. The notebook has a single cell containing Scala code to read a JSON file into a DataFrame and then register it as a temporary table named 'Twitter'. The code then runs a SQL query to select vertices from the table where user.location is not null, limiting the results to 10 rows. The output of this cell is a table showing 10 rows of data, each with an id and a location.

```
In [1]: from pyspark import SparkContext  
from graphframes import *  
  
In [2]: from pyspark import *  
from pyspark.sql import *  
  
In [3]: import findspark  
  
In [4]: sc=SparkContext().getOrCreate()  
  
In [5]: sqlcontext=SQLContext(sc)  
  
In [12]: df=sqlcontext.read.format("json").option("header","true").json("data.json")  
  
In [13]: df.registerTempTable("Twitter")  
  
In [45]: vertices=sqlcontext.sql('SELECT id,user.location FROM Twitter where user.location is not null limit 10')  
  
In [46]: vertices.show()  
+-----+-----+  
| id | location |  
+-----+-----+  
| 1194836026931462145 | uniondale, ny |  
| 1194836027258654727 | Space City 🌎 |  
| 1194836027292045312 | Beaverton |  
| 1194836027820498945 | Texas, USA |  
| 1194836028244332544 | Indiana, USA |  
+-----+-----+
```

Displaying the edges of the graph created,



The screenshot shows a Jupyter Notebook interface within a Chrome browser. The notebook has two cells. The first cell contains a SQL query to select user.name as src and user.followers_count as dst from the Twitter temporary table where user.name is not null. The second cell shows the result of running this query, which is a table of edges. The edges represent connections between users, where the src column is the user's name and the dst column is their follower count.

```
In [38]: edges=sqlcontext.sql('select user.name as src, user.followers_count as dst from Twitter where user.name is not null')  
  
In [39]: edges.show()  
+-----+-----+  
| src | dst |  
+-----+-----+  
| alex | 330 |  
| Gabby | 113 |  
| Alex | 197 |  
| Richard McCarthy IV. | 204 |  
| Classic Pics | 308673 |  
| SpookySalinas 🎃 | 190 |  
| Nada Tuffaha | 681 |  
| ☆ LILI ☆ | 181 |  
| Kemba will drop 2... | 106 |  
| Y E L L O W | 607 |  
| REECE DECLAN | 33 |  
| li | 119 |  
| geo | 117 |  
| Vel | 300 |  
| Matt | 60 |  
| That Bitch | 1029 |  
| PS Forever | 148 |  
| Mochi | 0 |  
| Dollar ⚡... | 152 |  
| chris | 866 |  
+-----+-----+  
only showing top 20 rows
```

Displaying the vertices of the graph created,

```
In [45]: vertices=sqlcontext.sql('SELECT id,user.location FROM Twitter where user.location is not null limit 10')
```

```
In [46]: vertices.show()
```

id	location
1194836026931462145	uniondale, ny
1194836027258654727	Space City 
1194836027292045312	Beaverton
1194836027820498945	Texas, USA
1194836028244332544	Indiana, USA
1194836028231536641	The Garden
1194836028407726080	California, USA
1194836028508581888	Ramsgate, South A...
119483602884830038	Azeroth
1194836029557088256	New York, NY

The inDegree of the graph is,

```
In [50]: graph.inDegrees.show()
```

id	inDegree
29	760
26	857
474	162
2250	14
964	71
1677	17
2509	13
1950	19
3506	6
3091	8
3764	5
1806	16
2040	14
2453	5
2927	3
229723	1
2214	8
2529	8
5409	2
1697	27

only showing top 20 rows

The outDegree of the graph is,

```
In [51]: graph.outDegrees.show()
```

	id	outDegree
Michelle	3	
donna cook	1	
t.	1	
Cassie	1	
Nell	3	
KJH	1	
oscar	10	
t.	9	
	34	
Edvin Escober	1	
Nazia	1	
phlower child	1	
Pomf & Thud Actu's	11	
dri	2	
Briana	1	
ruben	3	
Sha-leer-uh	1	
K	63	
simo	1	
malissa	1	

only showing top 20 rows

Triangle count for the graph,

```
In [52]: graph.triangleCount().show()
```

count	id	location
0	1194836026931462145	uniondale, ny
0	1194836027292045312	Beaverton
0	1194836029557088256	New York, NY
0	1194836028407726080	California, USA
0	1194836028244332544	Indiana, USA
0	1194836028508581888	Ramsgate, South A...
0	1194836027820498945	Texas, USA
0	1194836028231536641	The Garden
0	1194836028848300038	Azeroth
0	1194836027258654727	Space City 

```
In [39]: graph.vertices.write.parquet("vertices")
graph.edges.write.parquet("edges")
```

The vertices and edges are saved into different folders in the local system using the above command.

Docker:

Created the index.html where we linked all the queries we performed in spark.

Later we hosted on docker.

The screenshot shows the PyCharm IDE interface with the code editor open to the index.html file. The code is a simple HTML page with a background image and a list of links. The PyCharm interface includes toolbars, a sidebar with project and file navigation, and a bottom status bar showing the file path and version information.

```
<!DOCTYPE html>
<html>
<head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-Vkoo8xACGsO3+Hhxv8TQPAxKtkTu6ug5TOeNVgBiFewPGFN9MuHOF23Q9Ifjh" crossorigin="anonymous">
<link href="https://fonts.googleapis.com/css?family=Roboto&display=swap" rel="stylesheet">
</head>
<style>
body{
    font-family: 'Roboto', sans-serif;
}
img{
    max-width:100%;
    height:auto;
    display:none;
}
ul li{
    list-style-type:none;
    line-height:2.5em;
    cursor:pointer;
}
</style>
<body>
```

The screenshot shows the PyCharm IDE interface with the code editor open to the index.html file. The code has been modified to include ten links, each corresponding to a different query. The PyCharm interface includes toolbars, a sidebar with project and file navigation, and a bottom status bar showing the file path and version information.

```
<body>
<div class="container-fluid">
<div class="text-center"><h1 class="mx-auto mt-2">Tweet File Analysis</h1></div>
<div class="row">
<div class="col-12 col-md-6">
<ul class="list-group mt-3">
- >Tweets from different countries</li>
- >Source of Tweets</li>
- >Top 5 languages on web series twitter data</li>
- >Users with maximum number of followers</li>
- >Users with maximum number of favourites count</li>
- >Number of Tweets on desired series</li>
- >Users with highest number of retweets</li>
- >Top 5 places with highest number of tweets</li>
- >Number of tweets on prime, netflix and hulu</li>

</div>
<div class="col-12 col-md-6">











```

After creating the queries using spark, we deployed spark visualization into docker.



```

MINGW64/c/Program Files/Docker Toolbox/dockerfiles

Docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at https://docs.docker.com

Start interactive shell

jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ cd dockerfiles

jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker build -t jmaroju/bd-project .
Sending build context to Docker daemon 920.3MB
jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker build -t jmaroju/bdproject .
Sending build context to Docker daemon 2.293GB
Step 1/3 : FROM debian:stretch
stretch: Pulling from library/debian
7910f5b7d602: Pull complete
Digest: sha256:459a91bd90268d8f6f3dc6f2a84b9d0f3d1934828140d8ea2a5344a7c79063
Status: Downloaded newer image for debian:stretch
--> b33a5313558a: Pull complete
Step 2/3 : RUN apt-get update && apt-get install -y locales && dpkg-reconfigure -f noninteractive locales && locale-gen C.UTF-8 && /usr/sbin/update-locale LANG=C.UTF-8 && echo "en_US.UTF-8 UTF-8" > /etc/
locales.gen && locale-gen && apt-get clean && rm -rf /var/lib/apt/lists/*
--> Running in c3aa86c4dcbe
Get:1 http://security.debian.org/debian-security stretch/updates InRelease [53.0 kB]
Ign:2 http://deb.debian.org/debian stretch InRelease
Get:3 http://deb.debian.org/debian stretch-updates InRelease [93.6 kB]
Get:4 http://deb.debian.org/debian stretch Release [118 kB]
Get:5 http://deb.debian.org/debian stretch Release.gpg [2410 B]
Get:6 http://security.debian.org/debian-security stretch/updates/main amd64 Packages [632 kB]
Get:7 http://deb.debian.org/debian stretch-updates/main amd64 Packages [2596 B]
Get:8 http://deb.debian.org/debian stretch/main amd64 Packages [7080 kB]
Fetched 7982 kB in 45 (1894 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:


```

Push the latest image on to docker.

```

MINGW64/c/Program Files/Docker Toolbox/dockerfiles
--> 9a17a899c5ab
Step 3/3 : EXPOSE 5000
--> Running in f847a43ec730
Removing intermediate container f847a43ec730
--> 9a17a899c5ab
Successfully built dabe38dc58a
Successfully tagged jmaroju/bdproject:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensitive files and directories.

jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker push jmaroju/bdproject:bdptag
The push refers to repository [docker.io/jmaroju/bdpproject]
tag does not exist: jmaroju/bdpproject:bdptag

jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker push jmaroju/bdproject:tagname
The push refers to repository [docker.io/jmaroju/bdpproject]
tag does not exist: jmaroju/bdpproject:tagname

jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker commit jmaroju/bdproject:bdptag
Error response from daemon: No such container: jmaroju/bdpproject:bdptag

jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker push jmaroju/bdproject:latest
The push refers to repository [docker.io/jmaroju/bdpproject]
3d0b2e8a1533: Pushed
d02a5ea0b27: Pushed
51efdf2d4104: Pushed
d0f0333a0000: Pushed
58e5804ec704: Pushed
6556d138702b: Pushed
c8b8998e2e79: Pushed
abdf983d5b17: Pushed
16b4b1c74b5d: Pushed
83bc95313d01: Pushed
3b396ef0ab99: Pushed
0a7479930806: Pushed
23601bheff2: Pushed
c9614164b119: Pushed
3d27debdff00: Pushed
b9f053d3d19ff: Pushed
78d483cbdb37: Pushed
4bb8b800504b: Pushed
26711a4bf3b6: Mounted from library/debian
latest: digest: sha256:58f757ae2la6ef52444343570aa5e6ec863b82252568856bc708ed8f4f2e192 size: 4318

jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker push jmaroju/bdpproject:tagname

```

Using ps commands checking the containers. After that run the image on the 5000 port. And created the container.

```
jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker run -d -p 5000:5000 jmaroju/bdpproject
8597d86af368cb634a2f0f2e429c8fb6affa0d2c3a0ce34e394a94e81acb5c4

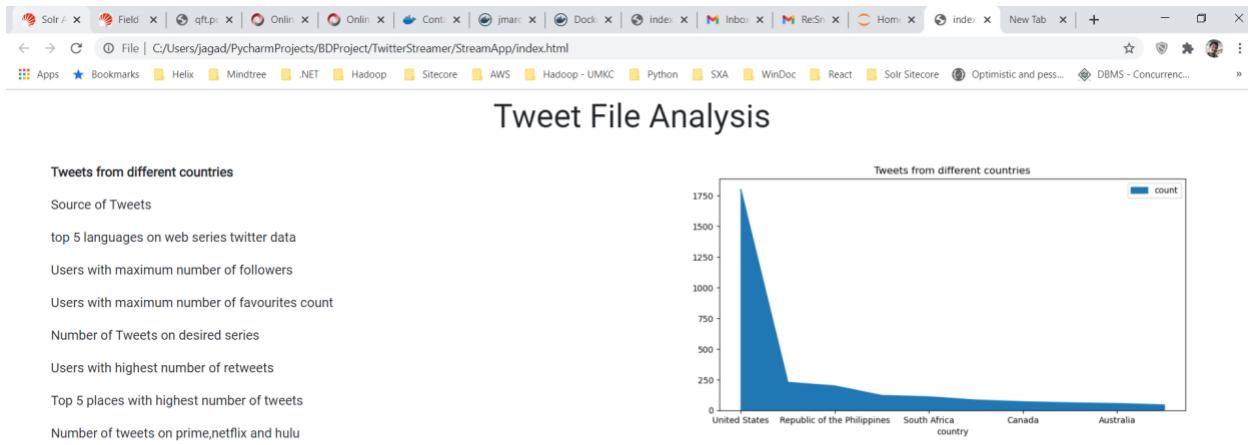
jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
8597d86af368      jmaroju/bdpproject   "/bin/sh -c 'python ..."   7 seconds ago    Up 5 seconds
jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker commit 8597d86af368 jmaroju/bdpproject:latest
sha256:a6aa1d75d9eb091c16b5f62ee1eedc64ebc0b7dbb73891921a67952743dae781

jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$
```

Bdpproject docker created with tag name latest.

The screenshot shows a Docker Hub repository page for the user `jmaroju` and the repository `bdpproject`. The page has a blue header with navigation links for Explore, Repositories, Organizations, Get Help, and a user dropdown. Below the header, there's a search bar and a message indicating 0 private repositories. The main content area has tabs for General, Tags, Builds, Timeline, Collaborators, Webhooks, and Settings. Under the General tab, it shows the repository name, a push button, and the last push time (43 minutes ago). It also lists one tag, `latest`, with details like OS (OS), Pulled, and Pushed times (both 43 minutes ago). A 'Docker commands' section includes a button to push a new tag. The 'Tags and Scans' section indicates vulnerability scanning is disabled. The 'Recent builds' section is empty.

Hosted the index template on the docker.



Conclusion:

Done the ETL process using the Spark's Batch Processing and then Spark Integration using Web UI. Performed our transactions on the set of RDD's and later we load our data in Hive which is similarly equal to the ETL basic process. This is because we are living in a world where data handling and data using plays one most important role in making the decisions for most of the industries. And at the present scenario where one has leisure time to binge watch series, where this web extraction and data analysis can be helpful.

Future work:

- Can use better Machine Learning Algorithm's such as clustering, sentimental analysis
- Try to stream the live data instead of collecting and parsing previous data from Twitter
- Analyze more data sources such as Facebook, Reddit and not only depending on Twitter

- Migrating to cloud fully without containerizing.

Project Management

Done the ETL operations on the dataset and used spark streaming with scala by creating Dataframes and RDD's with both scala and python programming (For initial increment - scala, for final report – python) it is observed that execution is much better in python compared to scala. Collected the dataset from twitter using tweepy and saved in json format. The tweet dataset is then converted into csv with data preprocessing to use and load it in solr. Applied algorithmic searches such as fuzzy and proximity in solr. And also loaded the data into hive table for query counts using different attributes/features. Used graphframes to find vertices, edges and indegree, outdegree, triangle count.

Finally created an index.html page for hosting it into the Docker container.

Team Responsibilities for Implementation,

- Twitter data streaming (Avinash, Geetanjali)
- Data processing (Akhil, Bhashitha)
- Spark streaming using scala (Bhashitha, Avinash)
- Spark streaming using python (Akhil, Geetanjali)
- Conversion of json data to csv (Geetanjali, Bhashitha)
- Solr execution (Avinash, Akhil)
- Hive implementation (Akhil, Geetanjali)
- Graphframes(Bhashitha, Avinash)
- Index web page (Geetanjali,Avinash)
- Dockerization (Akhil,Bhashitha)

Contributions: (workload was equally divided as a pairs for each task)

- Avinash Ganguri (25%)
- Akhil Teja Kanugolu (25%)
- Geetanjali Makineni (25%)
- Bhashitha Siddareddy (25%)

Issues:

- Jdk version compatibility between 15 and 8 for solr
- As json data is large we faced difficult in loading the data to hive and solr
- Converting json to csv is quite hectic because of the text file
- Faced difficulties with container in Docker.

STORY TELLING:

	WHO	WHAT	WHEN	WHERE	WHY
CHAPTER 1	Helps the advertiser's for pushing the advertisement and series makers for making quick analysis.	Pulling the data helps them in predicting the TRP which in result helps the series makers and advertisers.	As a result of COVID-19 most of the people settled back in their homes lacking entertainment.	All over the world.	Most of the people believe in IMDB rating rather than genre concept.
CHAPTER 2	The dataset is about the web series data where we sampled the tv shows, web series, Netflix series. The information disclosed about the shows watching around the world. Helps the advertiser's for pushing the advertisement and series makers for making quick analysis.	The events like which show is being watched more and where it is being watched and which kind of people are watching are seen. It helps advertisers in pulling the data helps them in predicting the TRP which in result helps the series makers and advertisers.	The events take place on how people react to the series they tweet. As a result of COVID-19 most of the people settled back in their homes lacking entertainment.	All over the world. As everyone in the world has an interest in web series and this data is being used everywhere. Thus, we are working on this data taking this into consideration.	Most of the people believe in IMDB rating rather than genre concept. That is the reason behind doing this web series data collection.
CHAPTER 3 (Scientist and AI)	People who watch web series which make them feel comfortable.	Collected the data using python as json file which we visualized by various platforms. Finally, hosted in docker.	Data of 2019 in all over the world collected with various hash tags related to web series of Netflix, prime, Hulu.	After getting the data. Later, Visualized the data using spark, solr, hive. Related to PB and BDP.	Helps to understand the data by visualizing the data stream pulled from twitter.
CHAPTER 4 (Users)	The set of people who are having free time	Helps to Visualize the Data or impact of series on the	Based on the Data available on the hosted index,	Deployed to docker and can	Make's aware of types of OTT

	and interested to watch the series can access the info.	current society based on various features/attributes.	user can get the contrast between the OTT platforms and the web series and their pulse among.	host on EC2 instance.	platforms impact on the society.
CHAPTER 5 (Society)	Users who are mostly spends their time on watching online series. Types of OTT platforms are sampled. No data is sampled.	This data which makes aware of the current situation about the OTT platforms, which mostly doesn't have social or cultural impacts but only have idea about the OTT platforms. As the data grabbed from the twitter streaming will have security reasons.	If the tweets pulled based on the hashtags which web series related to region, caste, cultural can create positive or negative impact on the social or cultural but mostly it creates the negative impact which may breach the security.	Impact will be on the users who mostly rely on web series. If the Data collected concentrated on the particular set of webseries can cause the evaluative bias.	At current pandemic situation it will be consequential to people who are misled by the various paid platforms which give the review of the series.

References:

1. developer.twitter.com/en/docs/tweets/data-dictionary/overview/intro-to-tweet-json
2. [www.info.com/Twitter Data Visualization](http://www.info.com/Twitter%20Data%20Visualization)
3. spark.apache.org/docs/latest/
4. <https://www.youtube.com/watch?v=lWzcZZEQYMk>
5. https://docs.docker.com/docker-hub/publish/customer_faq/