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:

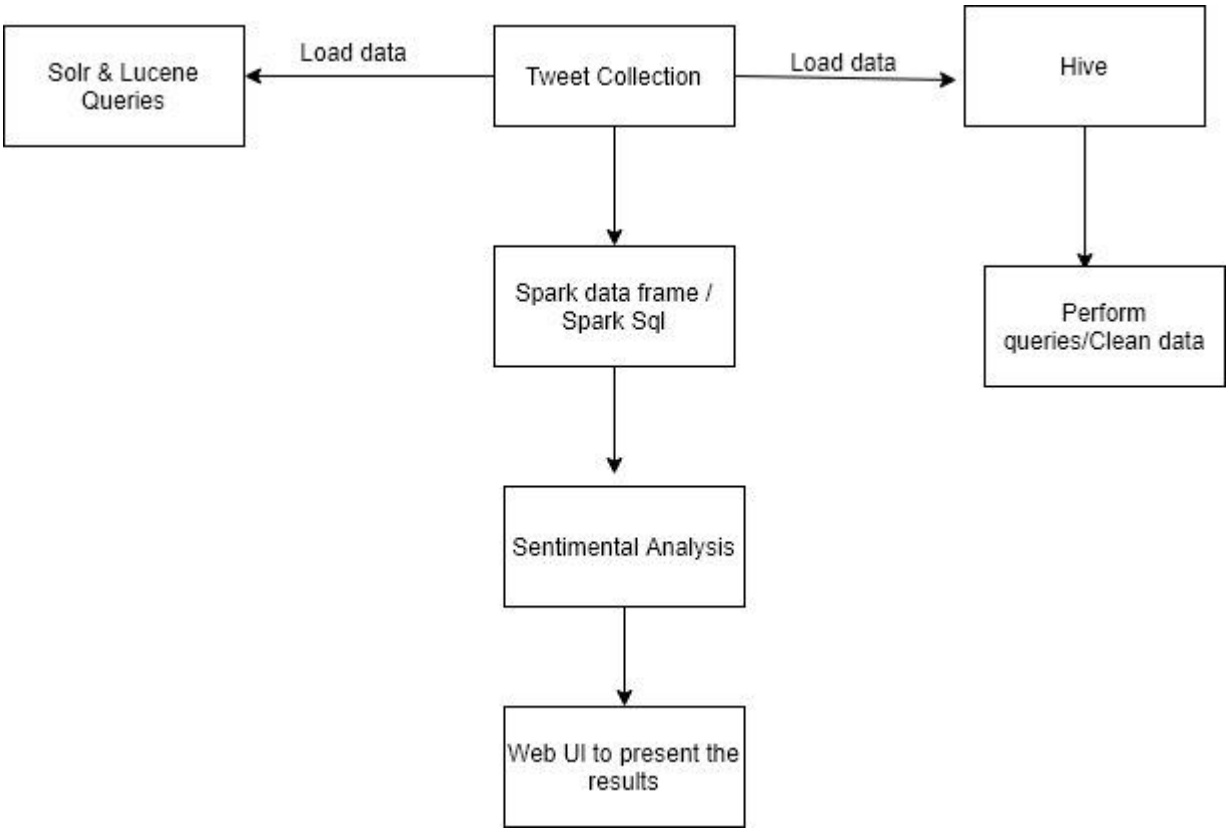


Fig: Architecture / Workflow Diagram

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

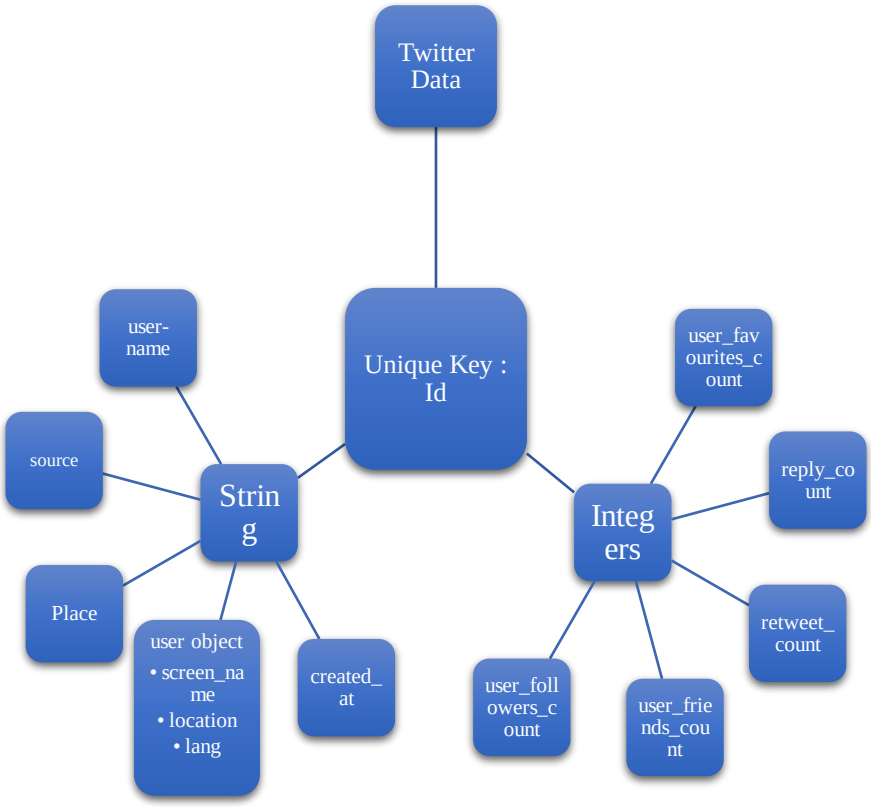


Figure: Design of Features/ Attributes

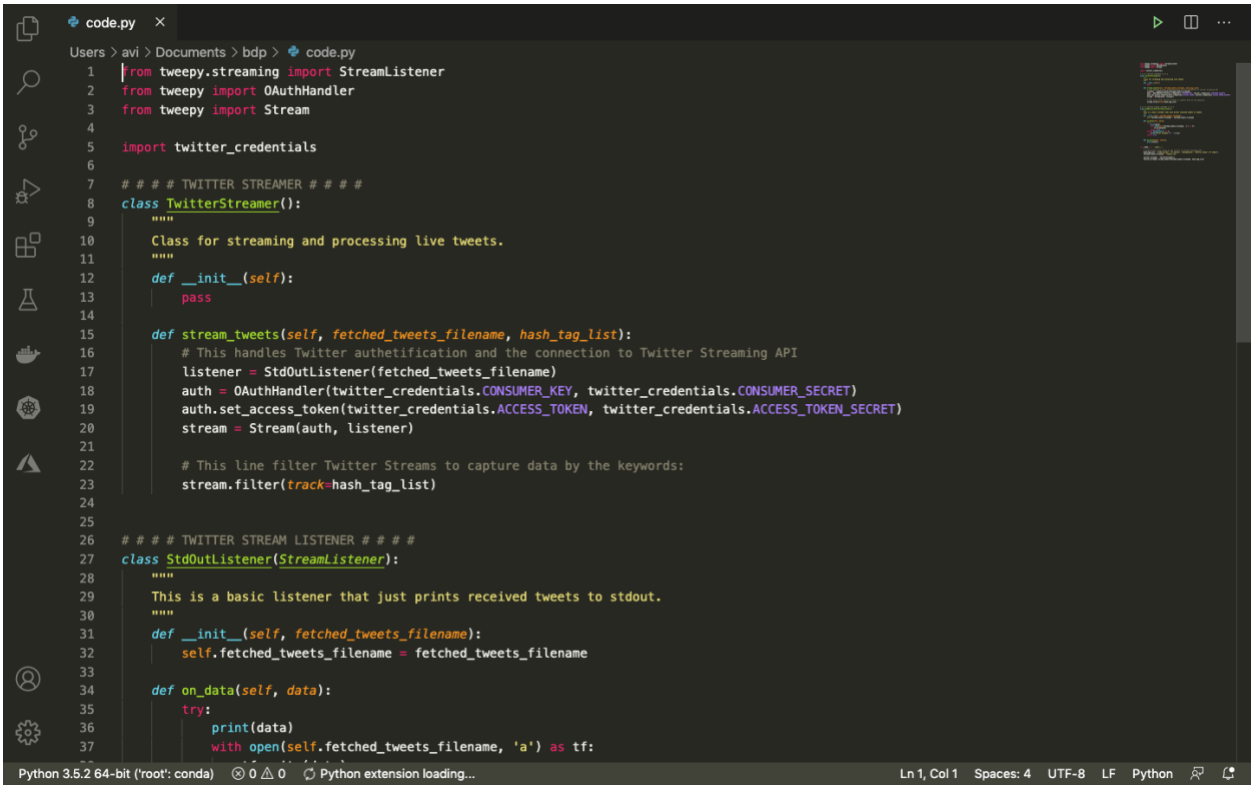
Analysis:

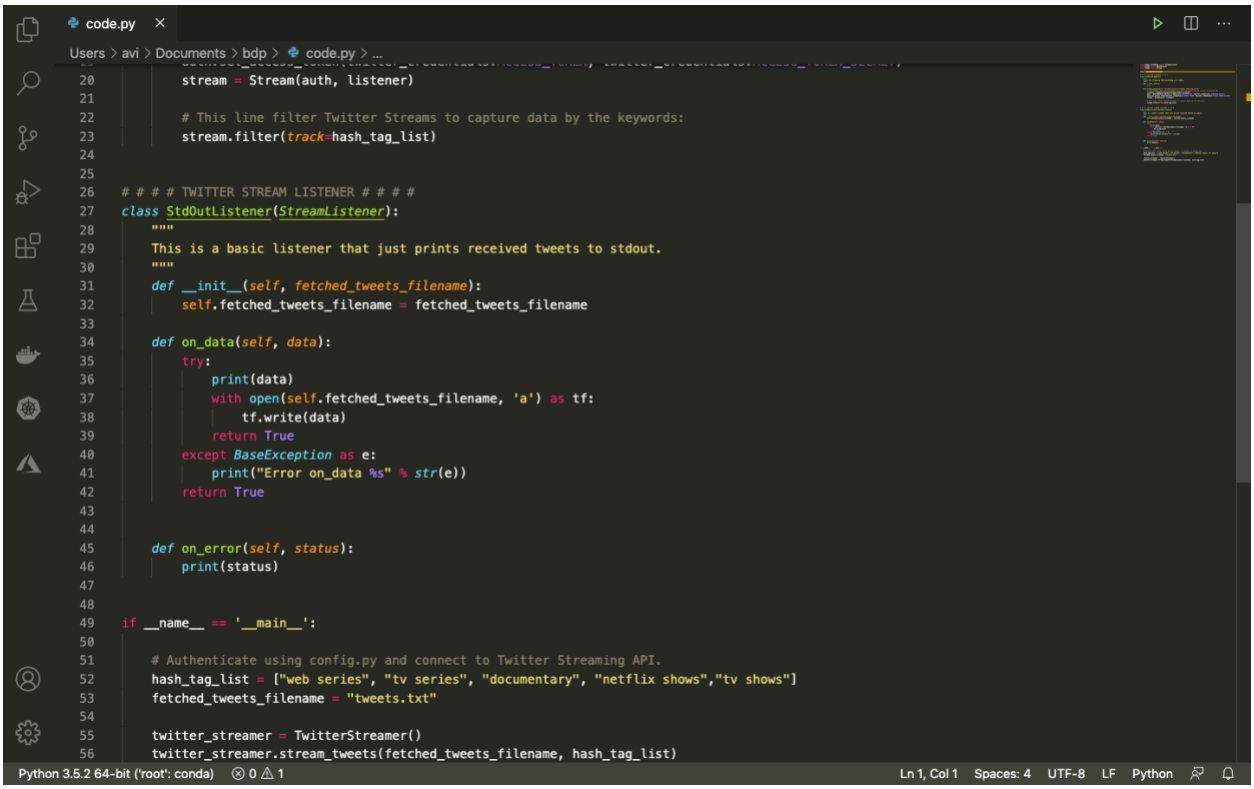
We have now first created one twitter developer account where next we got the tokens and credentials from API from where we took the twitter streaming and tweepy API in the python and stored tweets into the db. Extracted around 2.5 GB of dataset. Saved the data file in json format. Preprocess the data 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:





Spark using scala:

Starting the Spark-shell,

```
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   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 lal from tweetDatatable where coordinates is not null gro
up 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|        lal|
+-----+-----------+-----------+
|    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| -74.2125552|11.24220945|
|    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|
|    10|      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|    M|
|  202|    ✨|
|  174|    ♀|
|  148|    ~|
|  148|    J|
|  140| 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|         802410|null|
|       El Periódico|         639290|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 🇲🇽|         101650|null|
|   Postemático 🇲🇽|          96189|null|
| Don Sinóptico 🇲🇽|          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 mi
rror' 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|
|    70|          arrow|
|    39|          crown|
|     1|Game of Thrones|
|    61|          flash|
+------+---------------+
```

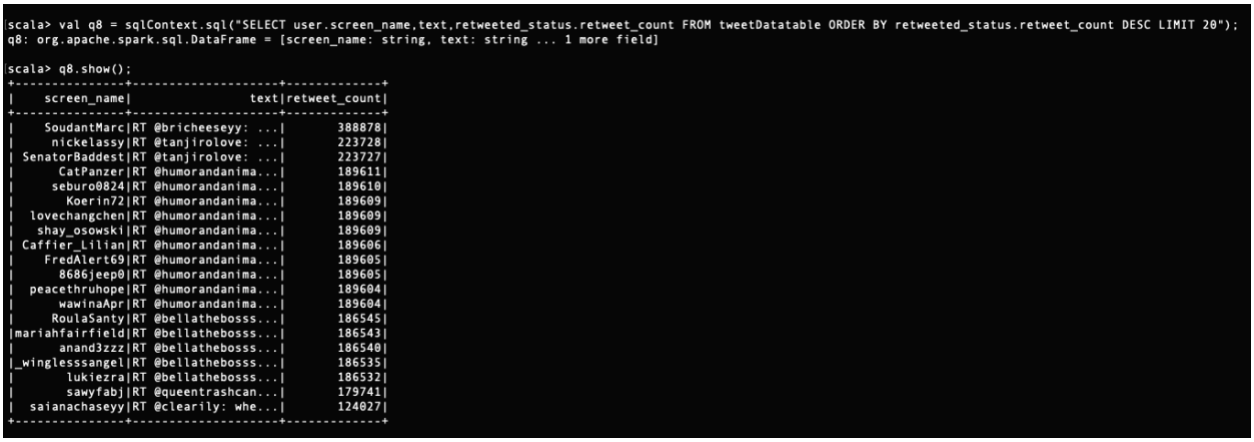
Query 7:

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.v
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|        1424077|
|   false|   GabbbarSingh|        1320213|
|   false| TheInteresting|        1288611|
|   false|      PoemPorns|        1286668|
|   false| smolAnimalsvid|         897755|
|   false|       gspot1177|         895129|
|   false|   UrbanEngIish|         824574|
+--------+---------------+---------------+
```

Query 8:

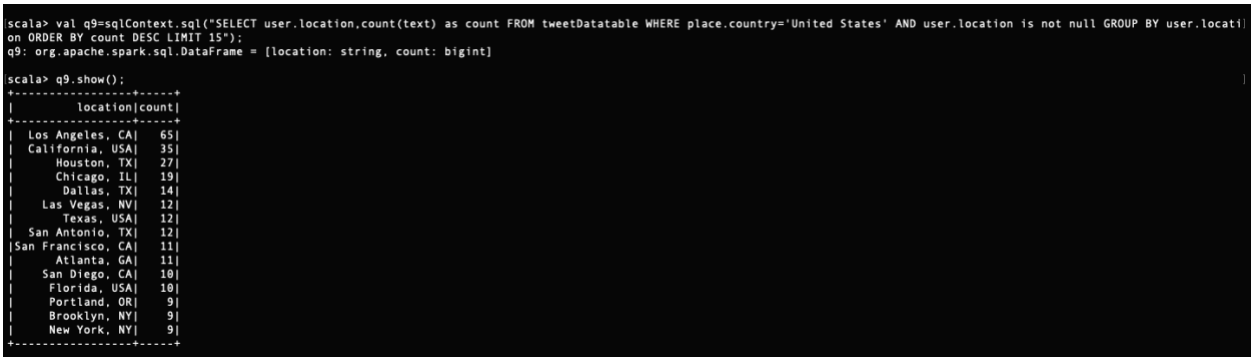
Users with a greater number of retweets for their tweets on web series.



```
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 @bricheeseyy: ...|       388878|
|     nickelassy|RT @tanjirolove: ...|       223728|
| SenatorBaddest|RT @tanjirolove: ...|       223727|
|      CatPanzer|RT @humorandanima...|       189611|
|     seburo0824|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|
|  peacethruhope|RT @humorandanima...|       189604|
|      wawinaApr|RT @humorandanima...|       189604|
|     RoulaSanty|RT @bellathebosss...|       186545|
|mariahfairfield|RT @bellathebosss...|       186543|
|      anand3zzz|RT @bellathebosss...|       186540|
|_winglesssangel|RT @bellathebosss...|       186535|
|       lukiezra|RT @bellathebosss...|       186532|
|       sawyfabj|RT @queentrashcan...|       179741|
|  saianachaseyy|RT @clearily: 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.locati
on 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:

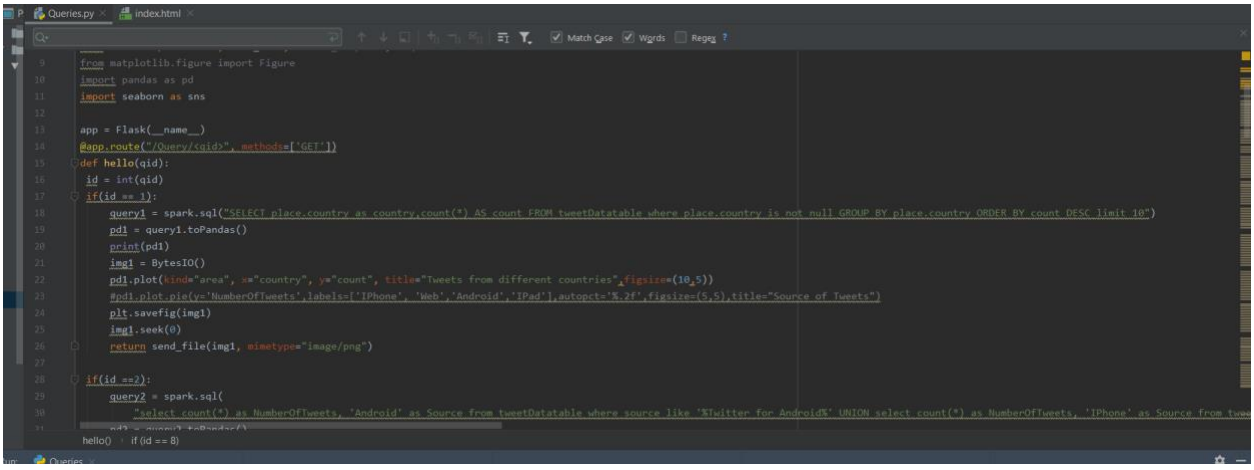
Fetching the top tweeted text and checking if someone re tweeted it on web series.

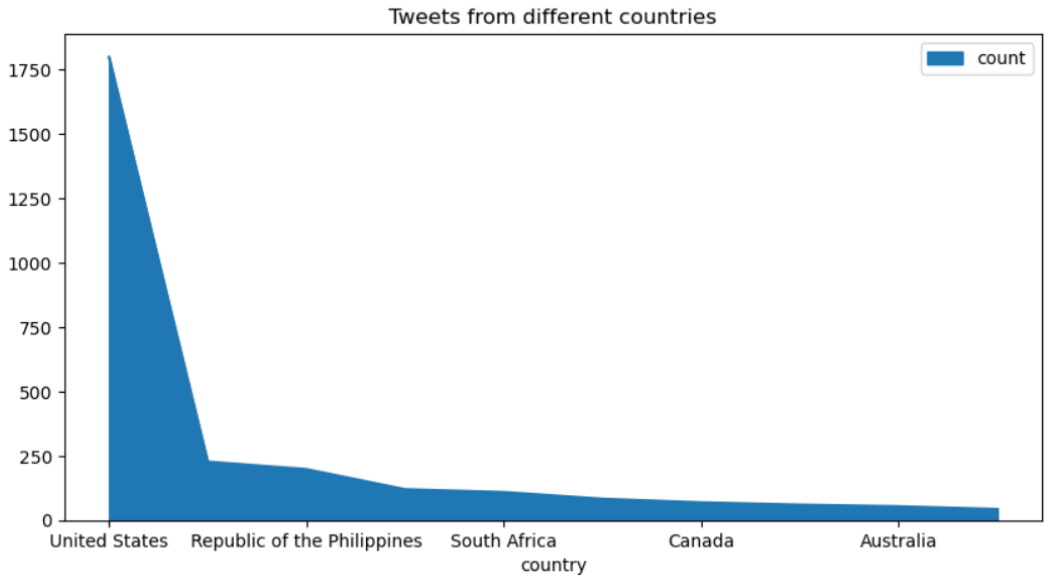
```
scala> val q10 = sqlContext.sql("SELECT user.name ,retweeted_status.text AS Retweet_Text,retweeted_status.retweet_count AS Retweet_Count FROM tweetDatatable WHERE retweeted_statu
s.retweet_count IS NOT NULL ORDER BY retweeted_status.retweet_count DESC limit 10");
q10: org.apache.spark.sql.DataFrame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

scala> q10.show();
+------------------+------------------+-------------+
|              name|      Retweet_Text|Retweet_Count|
+------------------+------------------+-------------+
|       soudant marc|My boyfriend lite...|       388878|
|ǝlɔɔıu ǝzıuǝp ɐxǝ...|my friends hyping...|       223728|
|   📌Bobby McNasty📌|my friends hyping...|       223727|
|              ねこパン|nala, an autism s...|       189611|
|                 研|nala, an autism s...|       189610|
|              アドラー|nala, an autism s...|       189609|
|                R®|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.

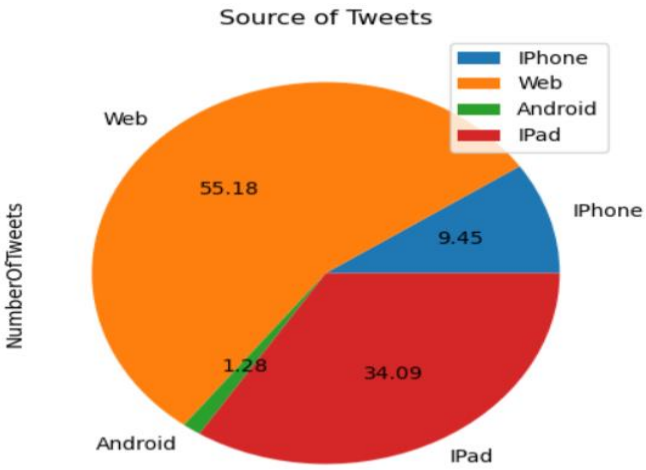




Query2:

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

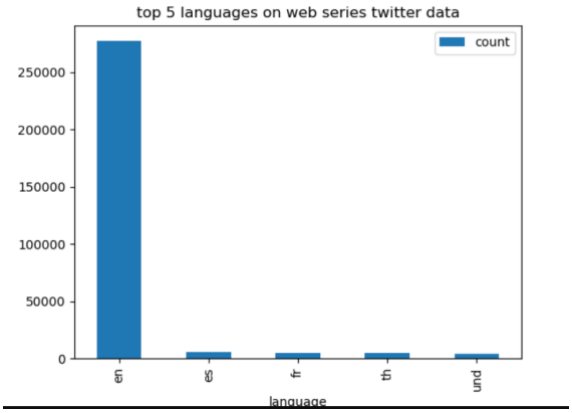
```
if(id ==2):
    query2 = spark.sql(
        "select count(*) as NumberOfTweets, 'Android' as Source from tweetDatatable where source like '%Twitter for Android%' UNION select count(*) as NumberOfTweets, 'IPhone' as Source from t
    pd2 = query2.toPandas()
    img1 = BytesIO()
    pd2.plot.pie(y='NumberOfTweets', 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.

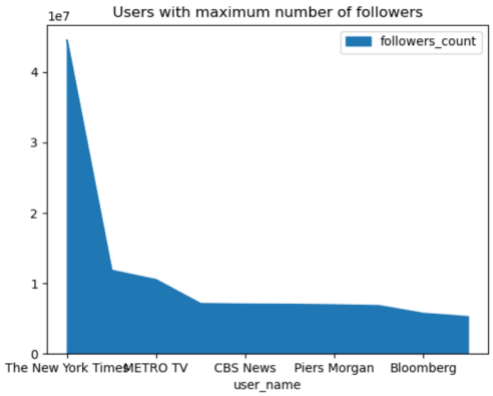
```
if(id == 3):
    query3 = spark.sql("select count(*) as count,lang as language from tweetDatatable where lang is not null group by lang order by count desc limit 5")
    print(query3)
    pd3 = query3.toPandas()
    pd3.plot(kind="bar", x="language", y="count", title="top 5 languages on web series 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

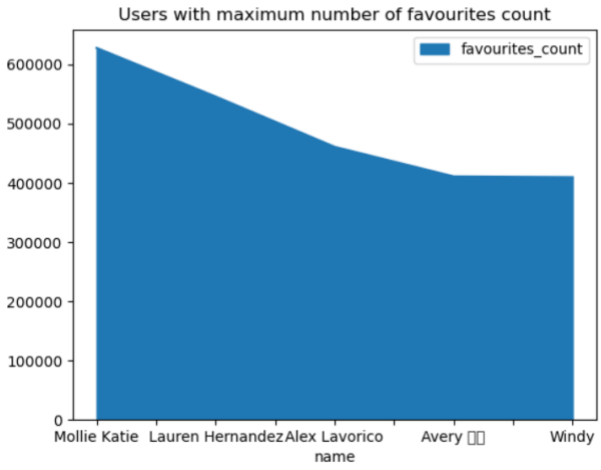
```
if(id == 4):
    query4 = spark.sql(
        "select user.name as user_name,user.followers_count as followers_count from tweetDatatable order by user.followers_count desc limit 10").dropDuplicates()
    pd4 = query4.toPandas()
    # pd.plot.area(x = "language",y="tweets",data=pd)
    pd4.plot.area(x='user_name',y='followers_count',title="Users 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

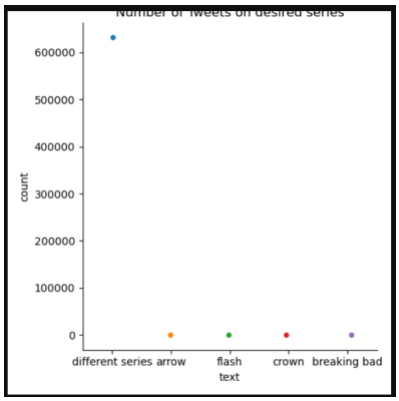
```
(id == 5):
 query5 = spark.sql(
     "SELECT user.name as name, max(user.favourites_count) as favourites_count FROM tweetDatatable WHERE text like '%series%' group by user.name order by favourites_count desc limit 5")
 pd5 = query5.toPandas()
 pd5.plot.area(x="name", y="favourites_count", title="Users 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.

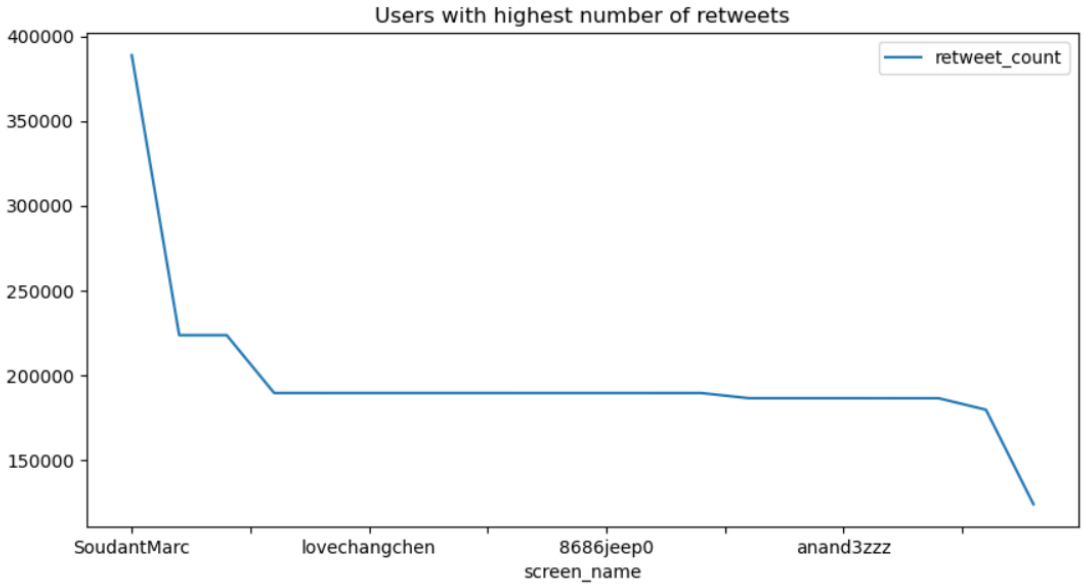
```
(id == 6):
 query6 = spark.sql(
     "select count(*) as count,q.text as text from (select case when text like '%prison break%' then 'prison break' when text like '%black mirror%' then 'black mirror' when text like '%crown%' t
 pd6 = query6.toPandas()
 print(pd6)
 # sns.catplot(x='candidate', y='polarity',data=pd6)
 # plt.title("")
 # pd6.plot(kind='line', x='candidate', y='polarity', color='red', ax=ax)
 sns.catplot(x='text', y='count', data=pd6).set(title="Number of Tweets on desired series")
 img6 = BytesIO()
 plt.savefig(img6)
 img6.seek(0)
 return send_file(img6, mimetype="image/png")
```



Query7:

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

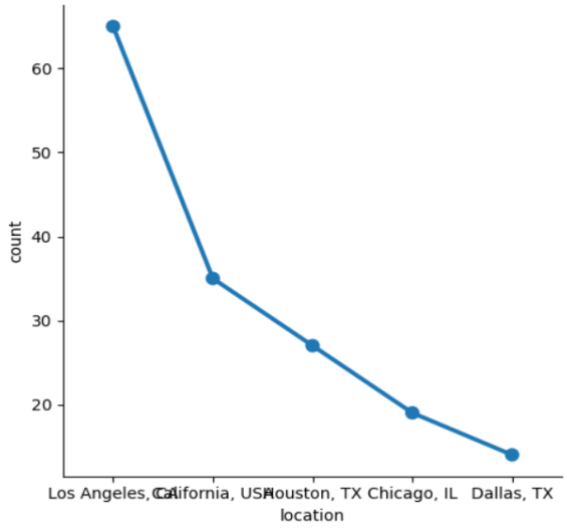
```
if(id == 8):
    query8 = spark.sql(
        "SELECT user.screen_name as screen_name,text,retweeted_status.retweet_count as retweet_count FROM tweetDatatable ORDER BY retweeted_status.retweet_count DESC LIMIT 20")
    # query8.createOrReplaceTempView("df1")
    # subQuery8 =spark.sql(
    #     "SELECT 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.

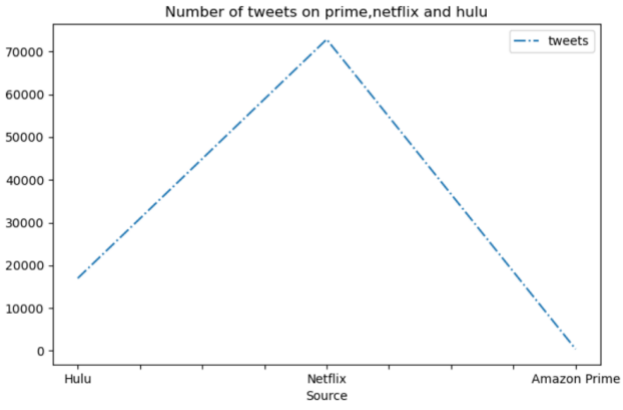
```
if(id == 9):
    query9 = spark.sql(
        "SELECT user.location as location,count(text) as count FROM tweetDatatable WHERE place.country='United States' AND user.location is not null GROUP BY user.location ORDER BY count DESC LI
    pd9 = query9.toPandas()
    #pd9.plot.pie(y='count',labels=pd9.location,autopct='%.2f',figsize=(5,5),title="Source 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

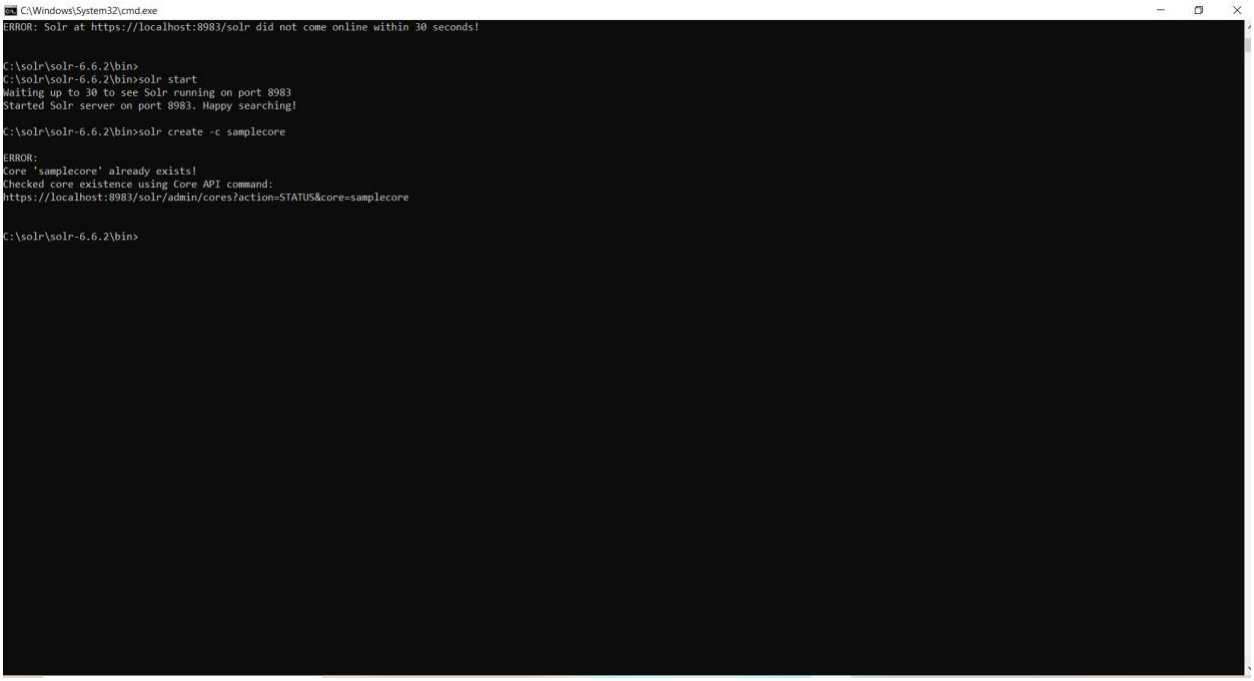
```
query10 = spark.sql(
    "select count(*) as tweets, 'Hulu' as Source from tweetDatatable where text like '%hulu%' or text like'%Hulu%' UNION select count(*) as tweets, 'Netflix' as Source from tweetDatatable
pd10 = query10.toPandas()
pd10.plot(linestyle='--',title="Number of tweets on prime,netflix 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")
```



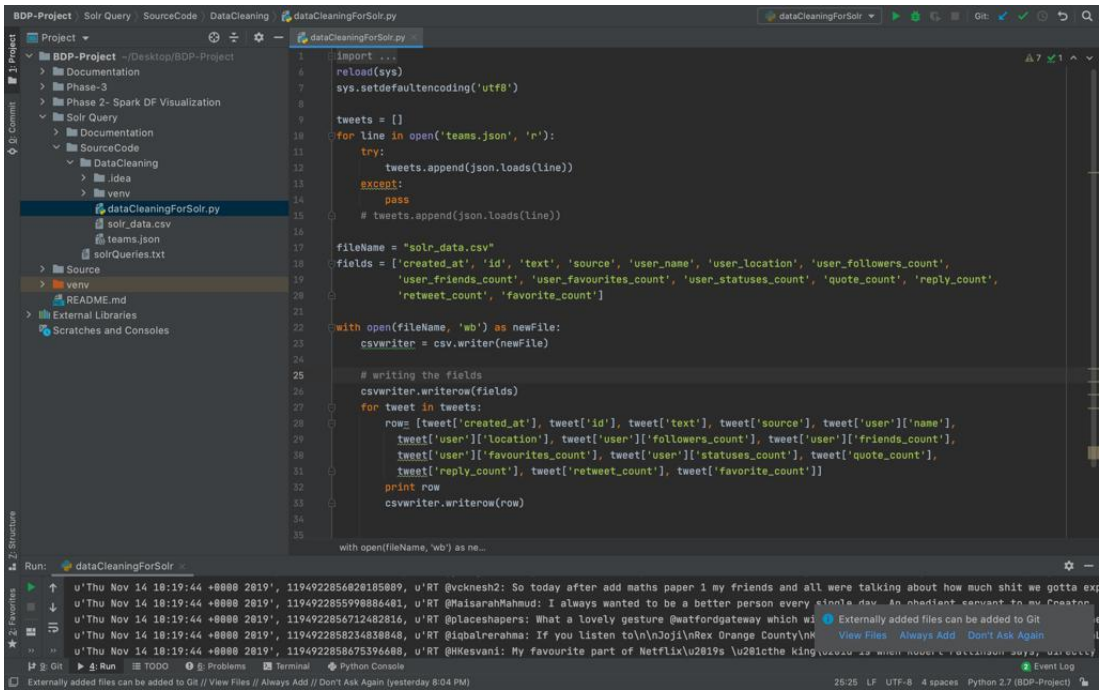
Solr:

Started solr with confiruation 6.6 and JDK 8.

Created the core name with samplecore



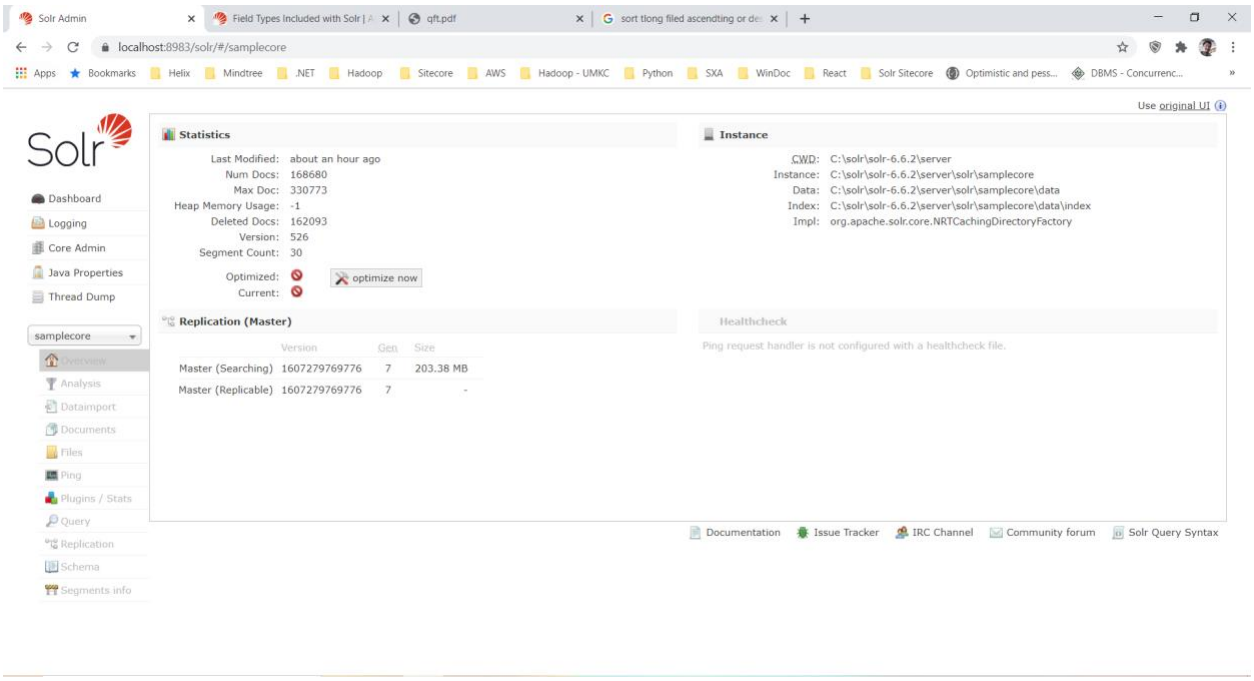
Converted the JSON file to CSV using python which makes loading data to solr easy.



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

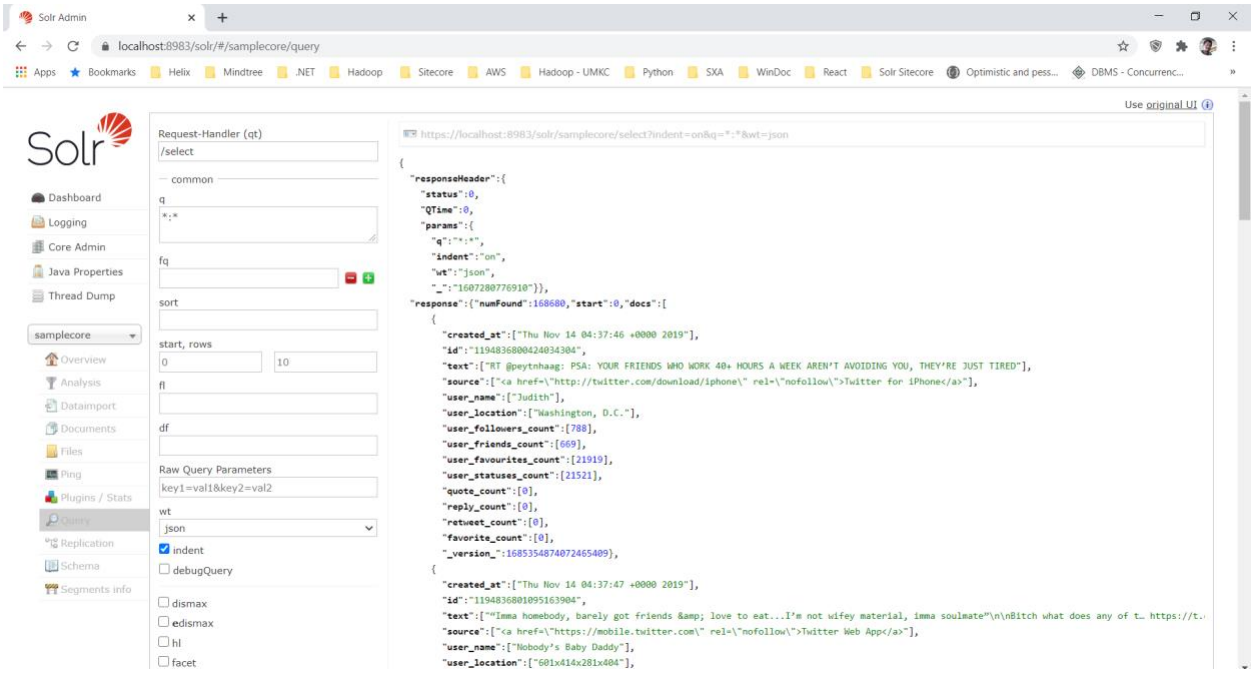
```
<field name="_text_" type="text_general" multiValued="true" indexed="true" stored="false"/>
<field name="_version_" type="long" indexed="false" stored="false"/>
<field name="created_at" type="strings"/>
<field name="favorite_count" type="tlongs"/>
<field name="id" type="string" multiValued="false" 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="* descendent path" type="descendent path" indexed="true" stored="true"/>
```

Overview after loading the csv data to solr.



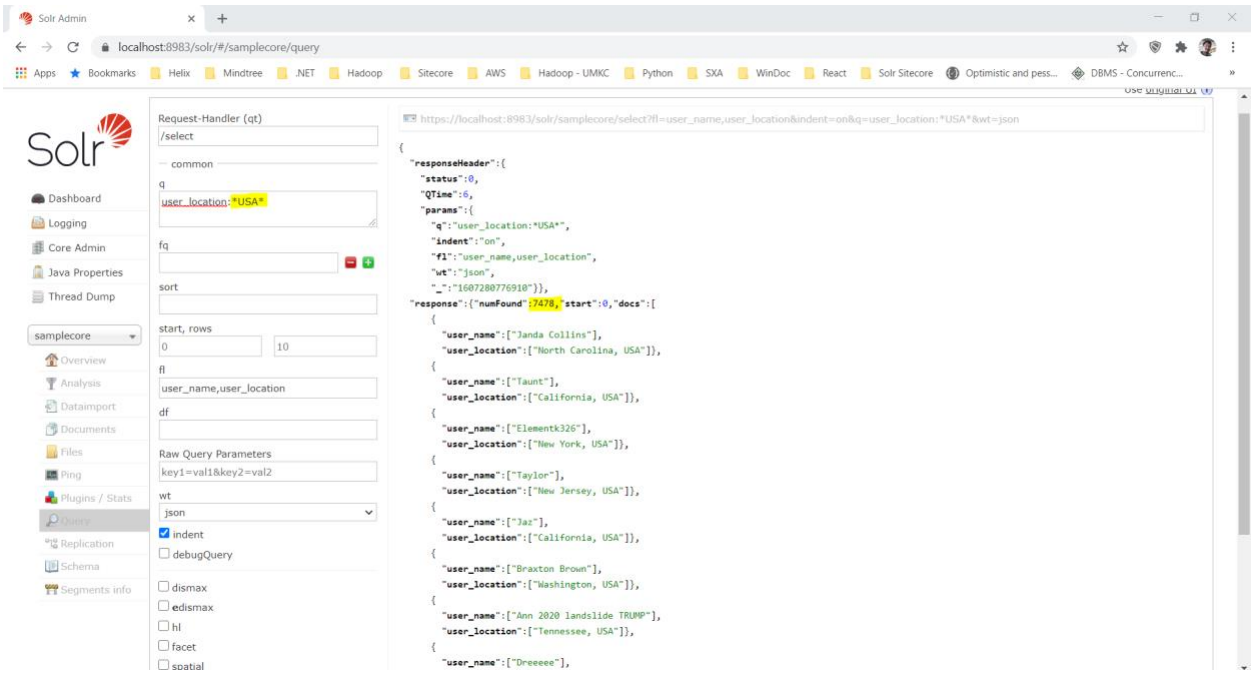
Query1:

Pulled the 10 records of data in json form.



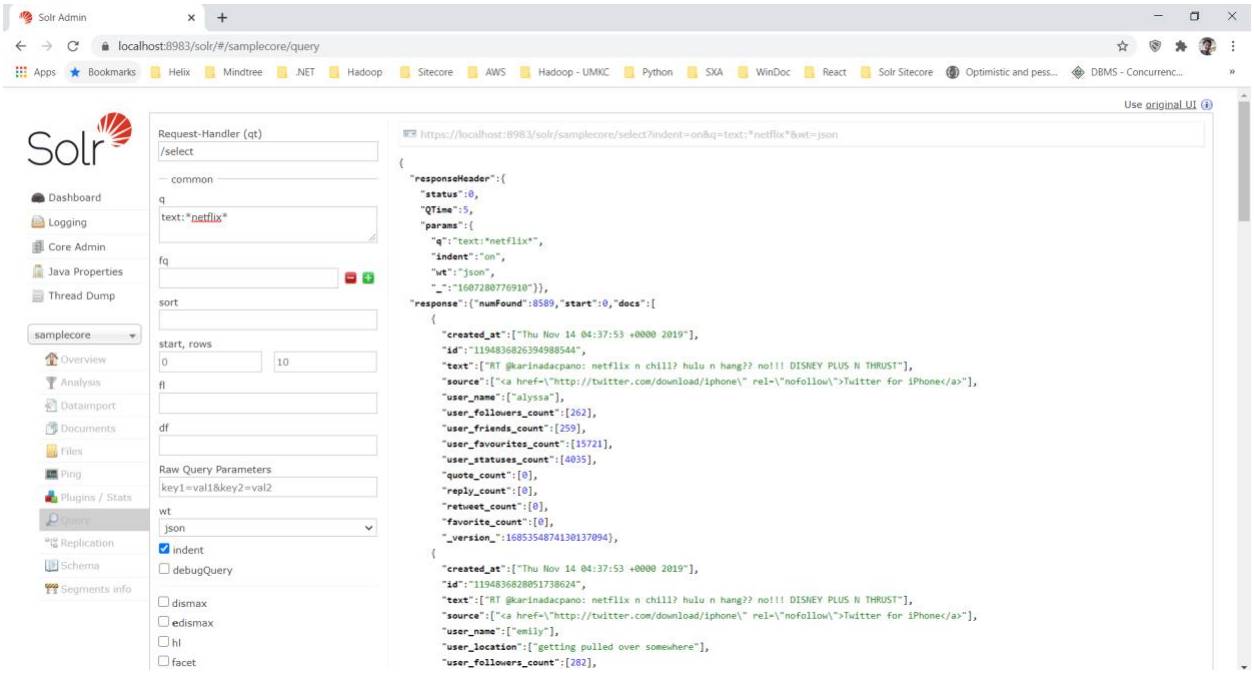
Query2:

Pulled the data with user_location has USA (Regex)



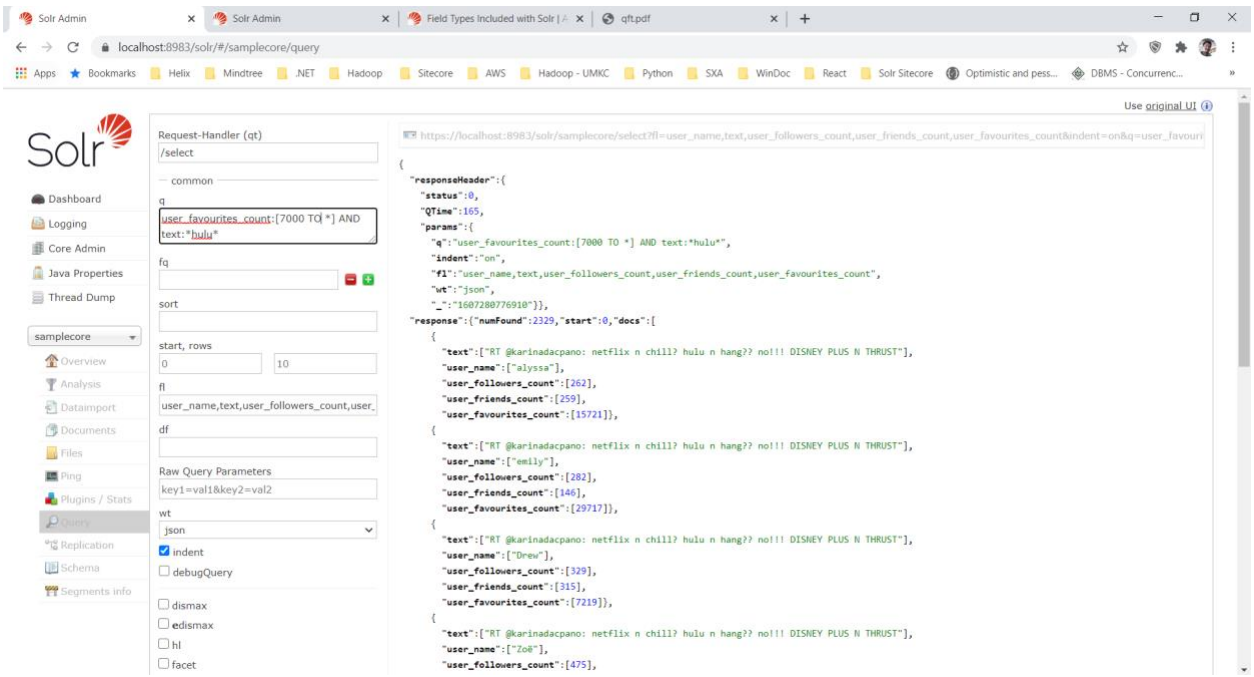
Query3:

Pulled the data has Netflix with in text (Regex)



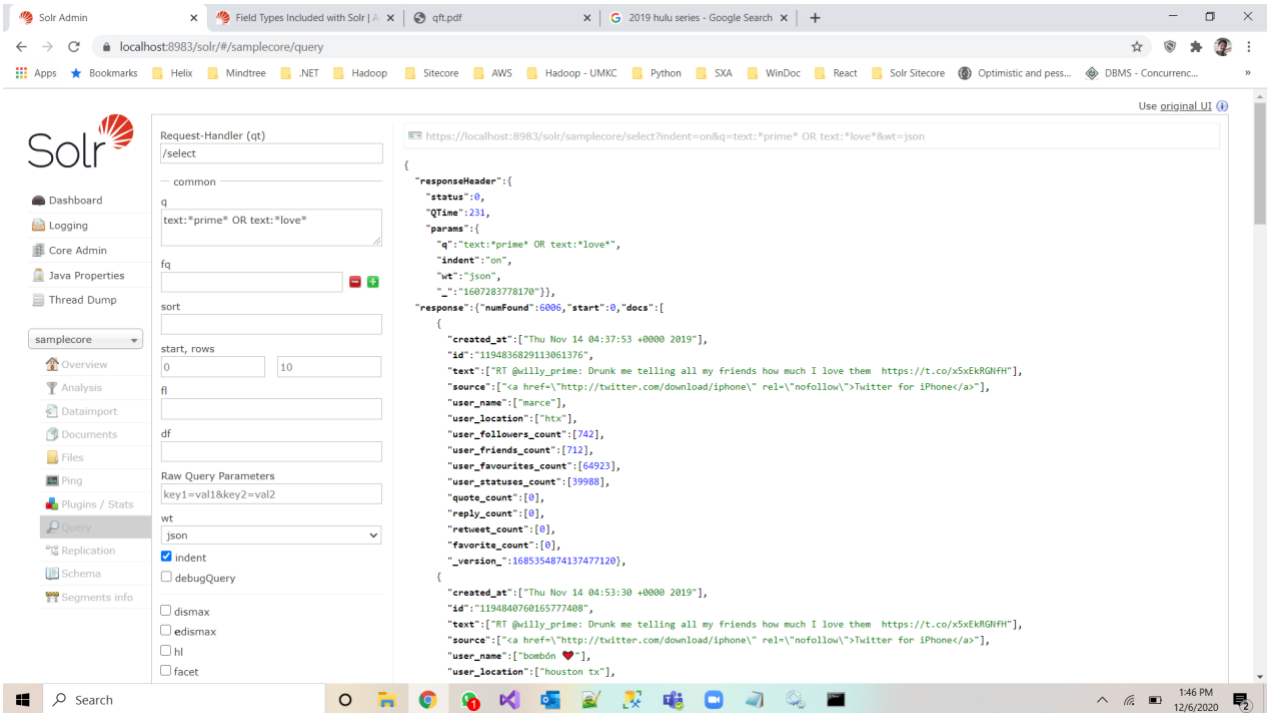
Query4:

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



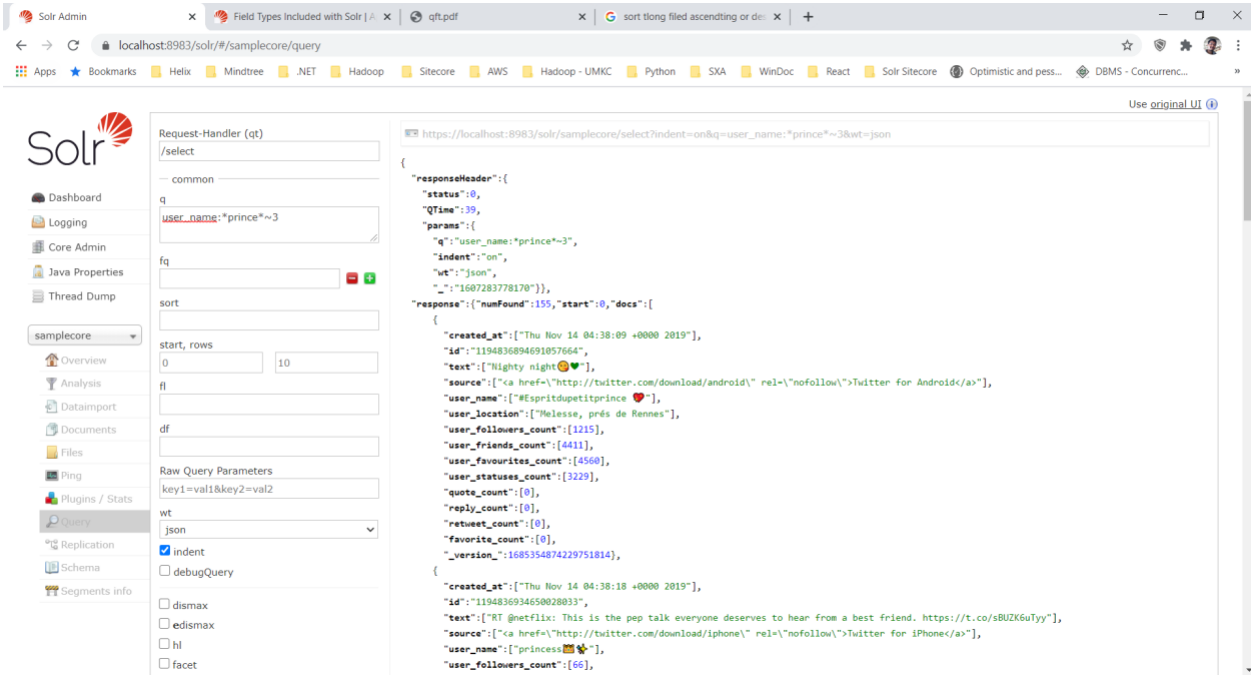
Query5:

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



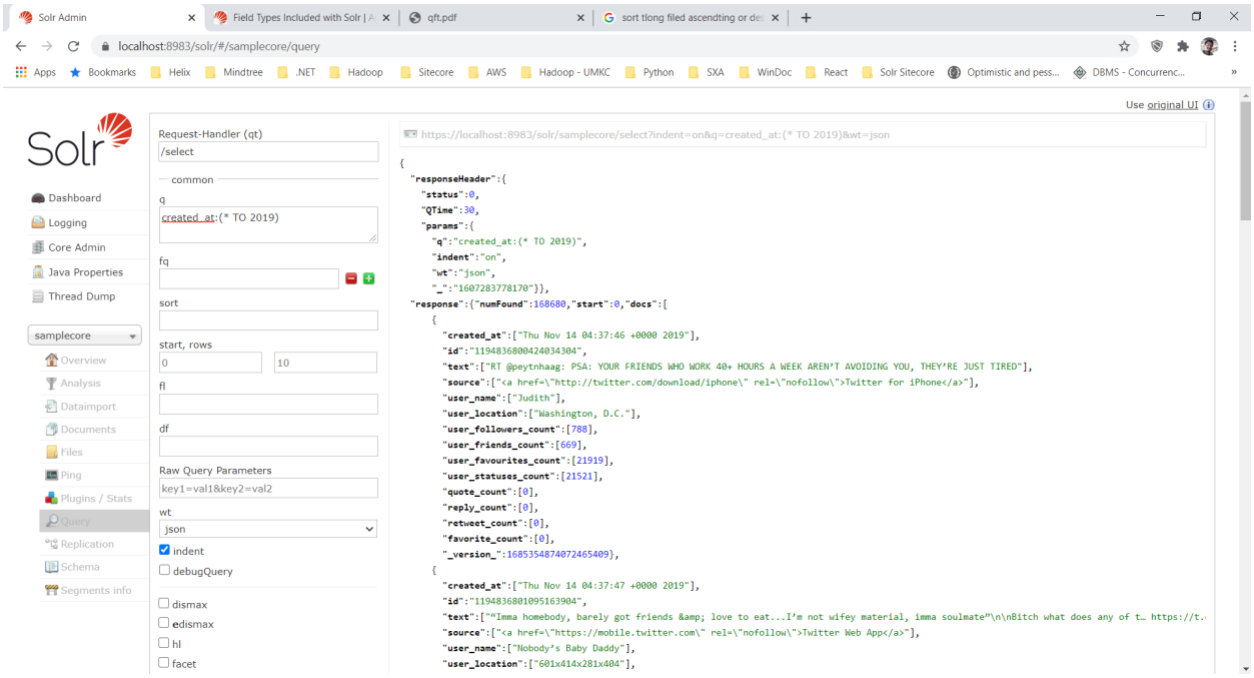
Query 6:

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



Query 7:

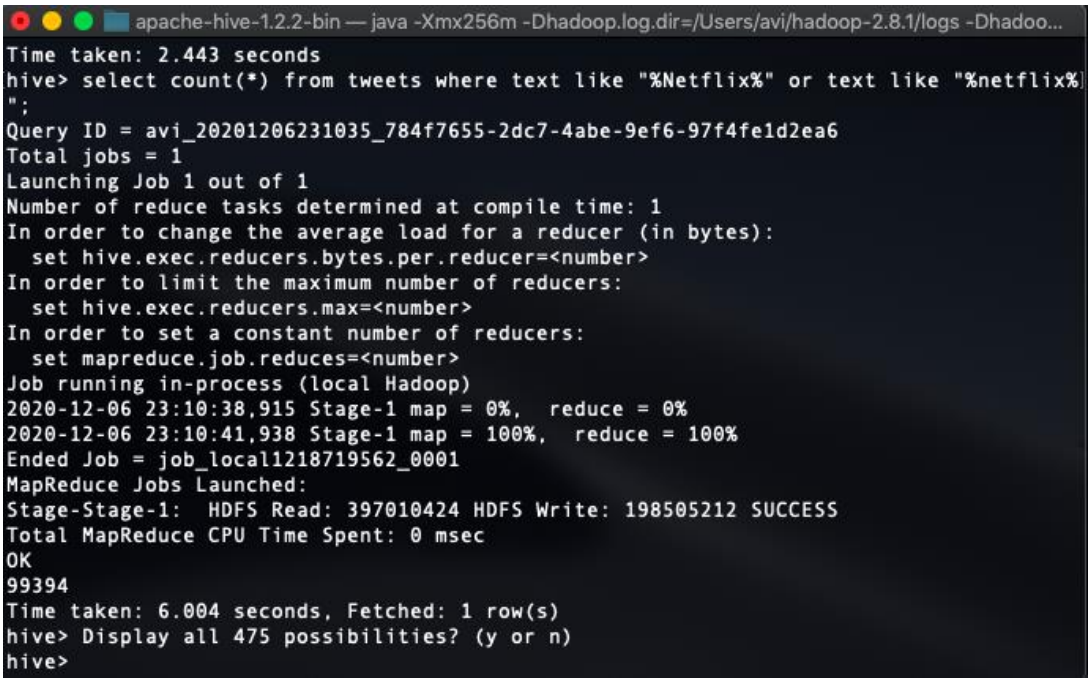
Based on created_at attribute to pull the data before 2019.



Hive

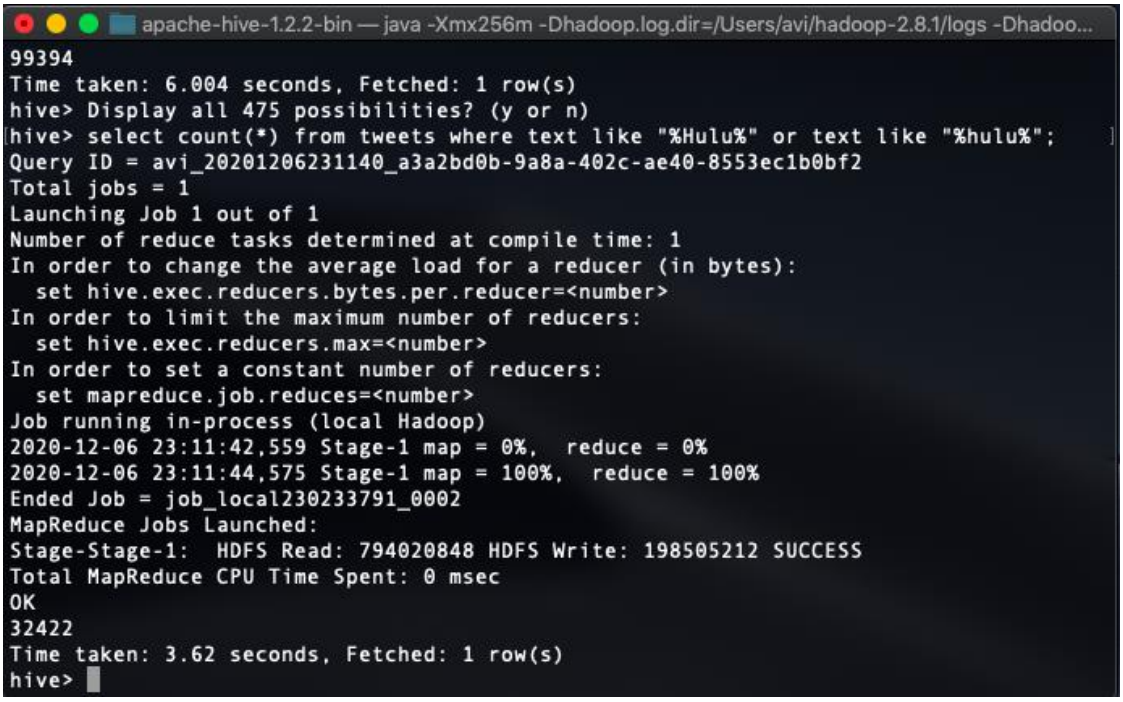
Query 1:

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



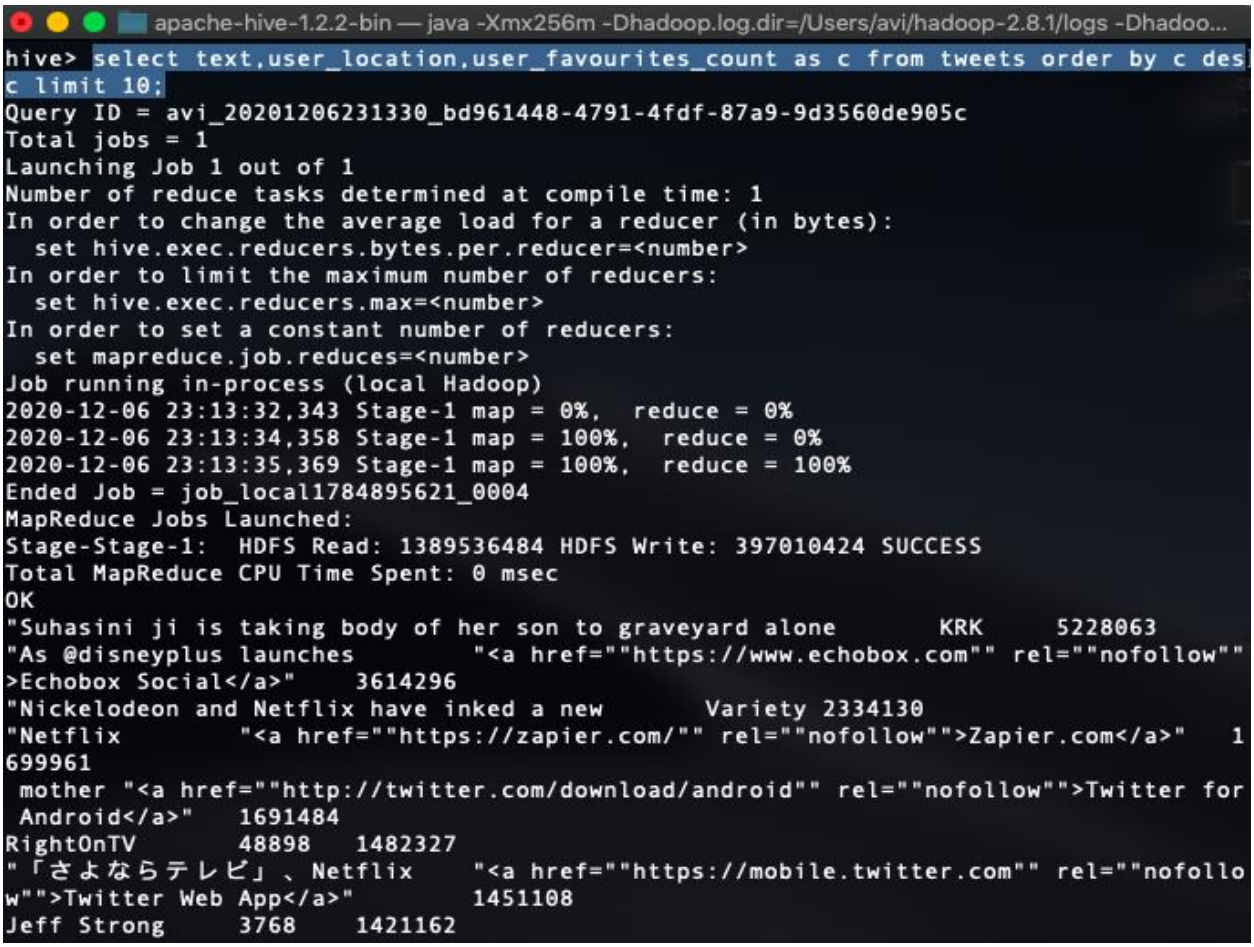
Query 2:

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



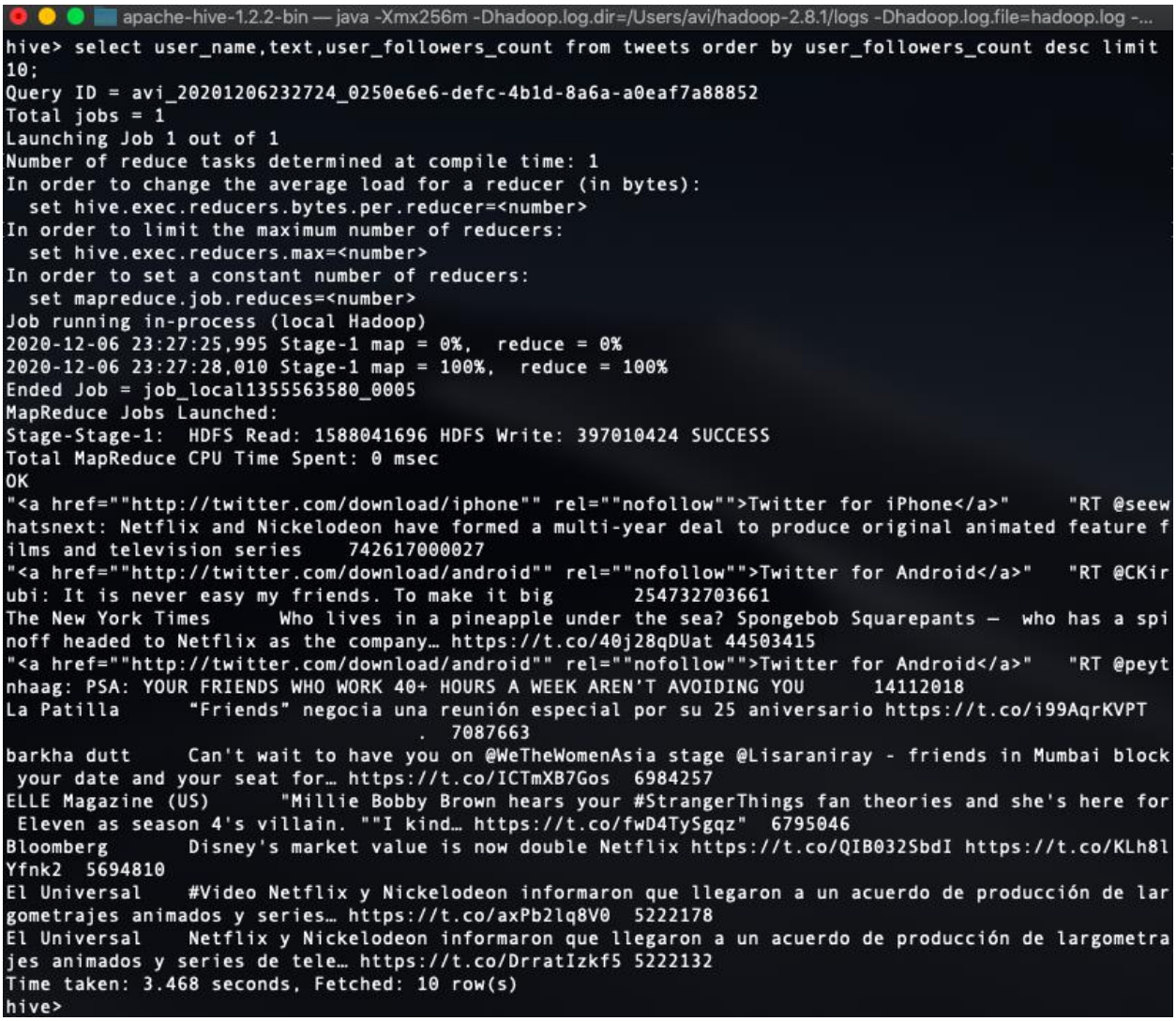
Query 3:

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

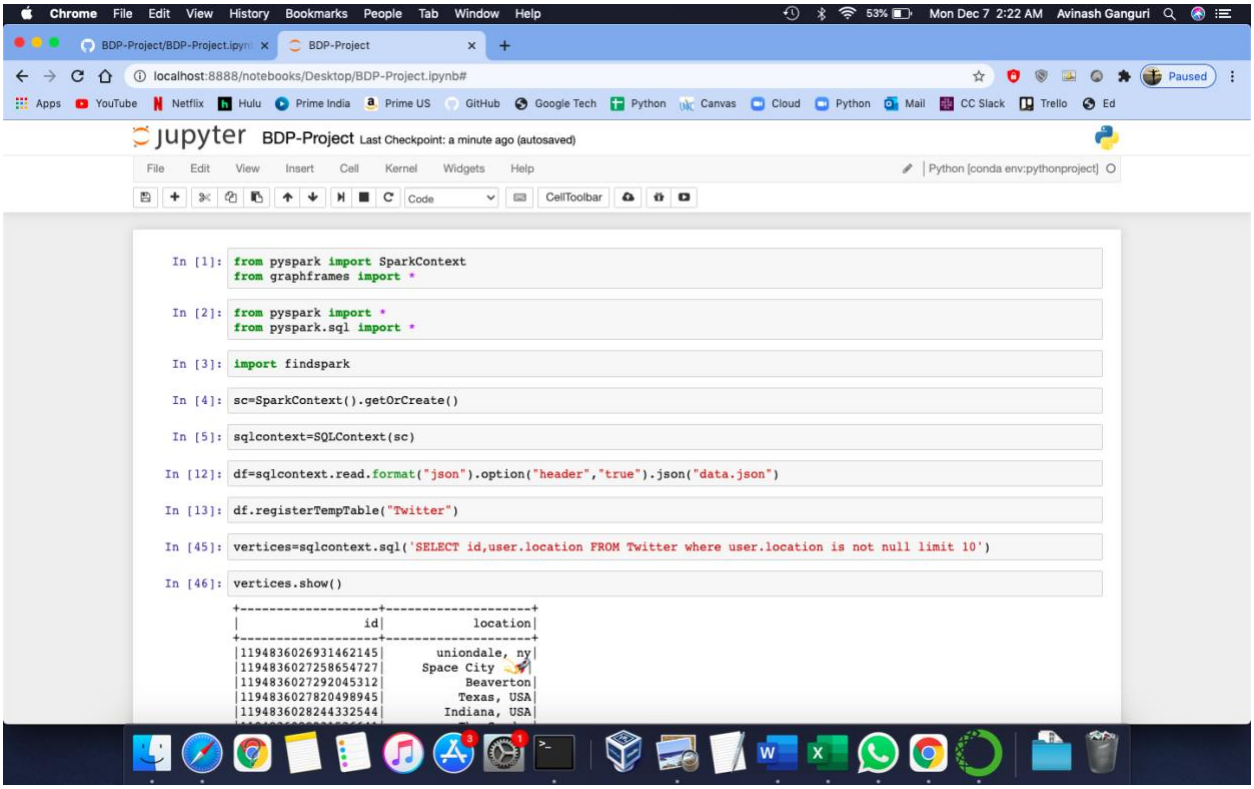


Query 4:

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
"Twitter for iPhone" "RT @seewhatsnext: Netflix and Nickelodeon have formed a multi-year deal to produce original animated feature films and television series 742617000027
"Twitter for Android" "RT @CKirubi: 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 spinoff headed to Netflix as the company… https://t.co/40j28qDUat 44503415
"Twitter for Android" "RT @peytnhaag: 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/KLh8lYfnk2 5694810
El Universal #Video Netflix y Nickelodeon informaron que llegaron a un acuerdo de producción de largometrajes animados y series… https://t.co/axPb2lq8V0 5222178
El Universal Netflix y Nickelodeon informaron que llegaron a un acuerdo de producción de largometrajes animados y series de tele… https://t.co/DrratIzkf5 5222132
Time taken: 3.468 seconds, Fetched: 10 row(s)
hive>

GraphFrames:



Displaying the edges of the graph created,

```
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 𝔊...|   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...|
|1194836028848300038|             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☆彡|        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 🚀|
+-----+------------------+--------------------+
```

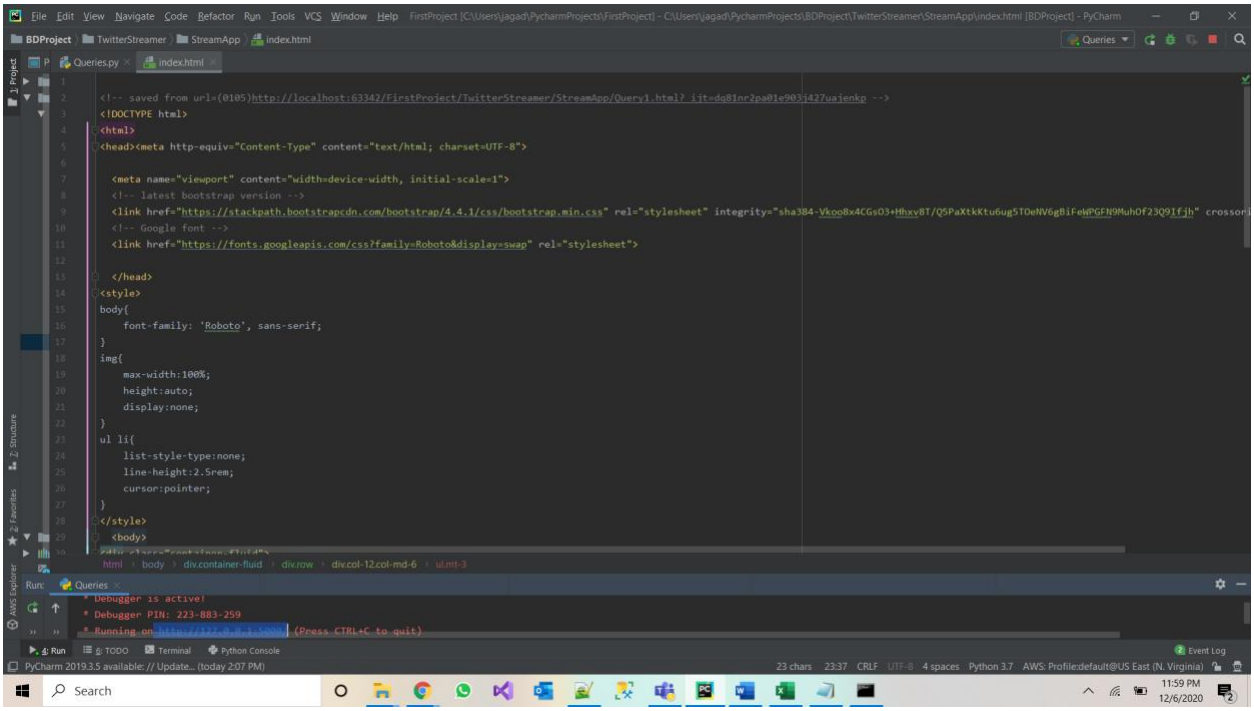
```
In [39]: graph.vertices.write.parquet("vertices")
         graph.edges.write.parquet("edges")
```

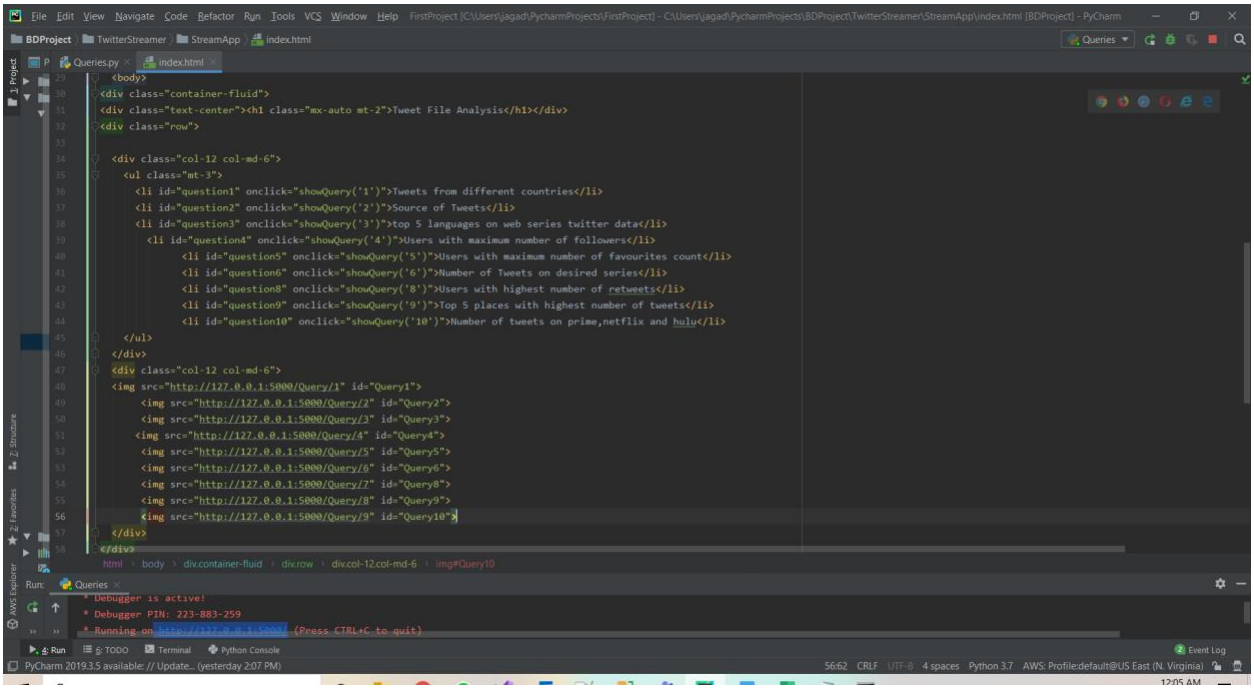
The vertices and edges are saved into different folders in the local system using the above command.

Docker:

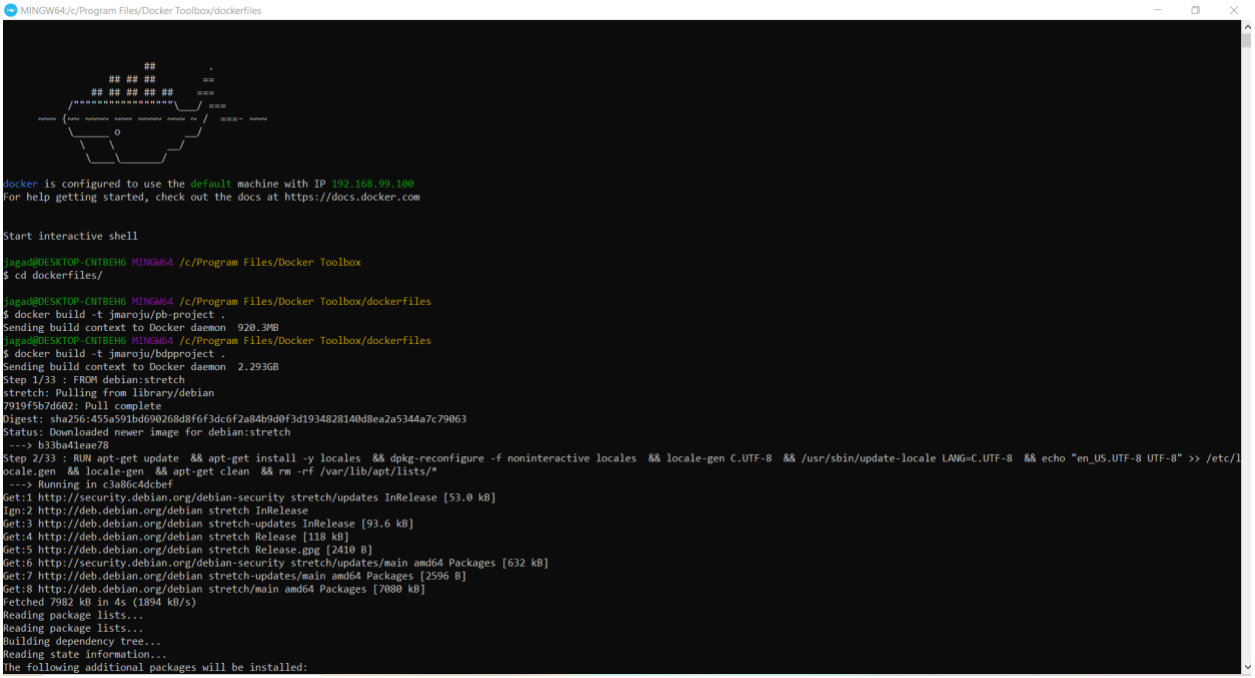
Created the index.html where we linked all the queries we performed in spark.

Later we hosted on docker.

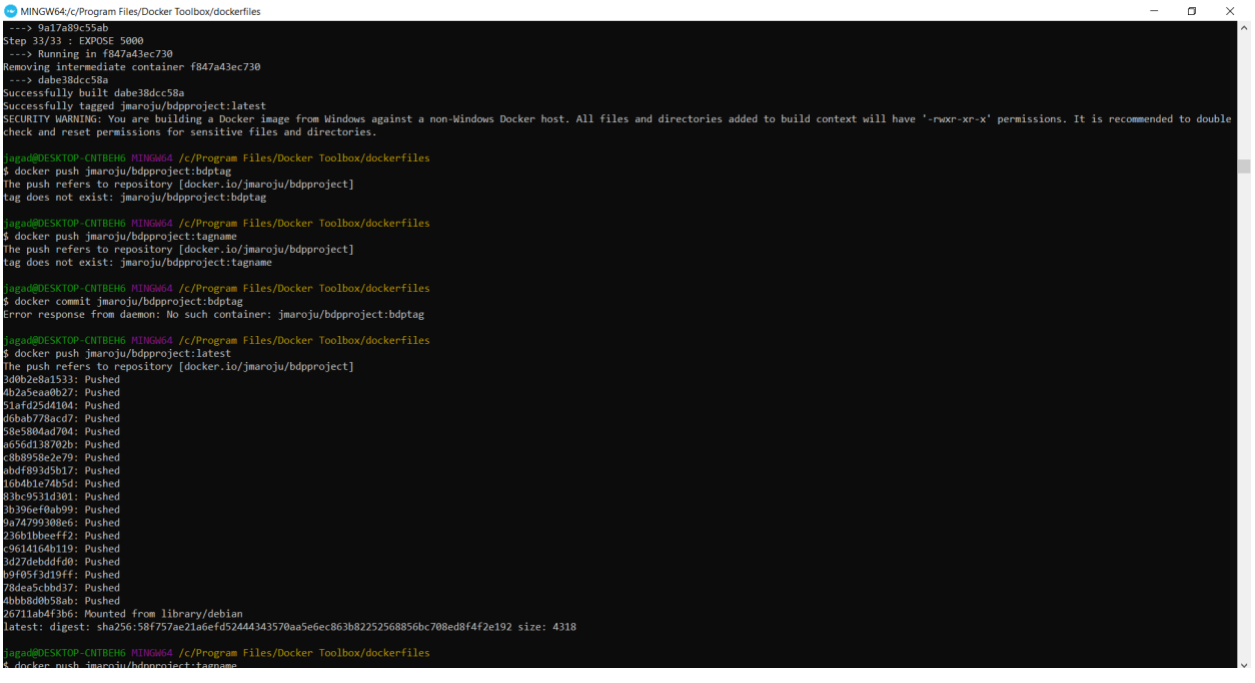




After creating the queries using spark, we deployed spark visualization into docker.



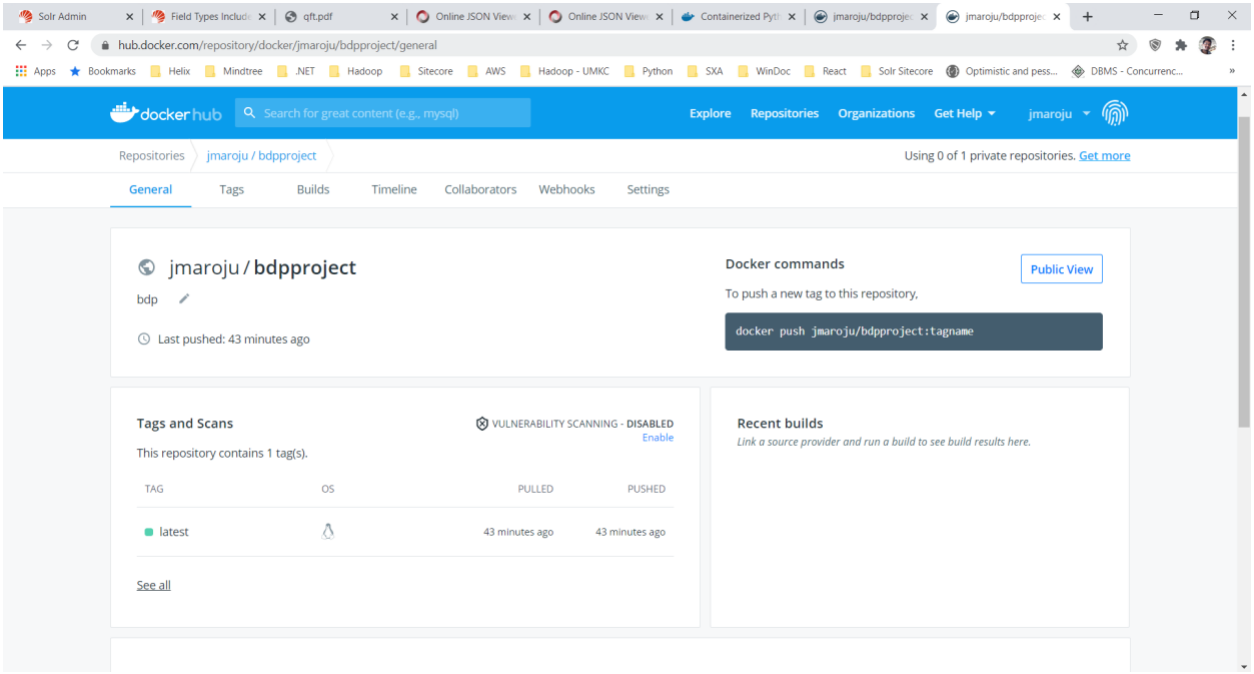
Push the latest image on to docker.



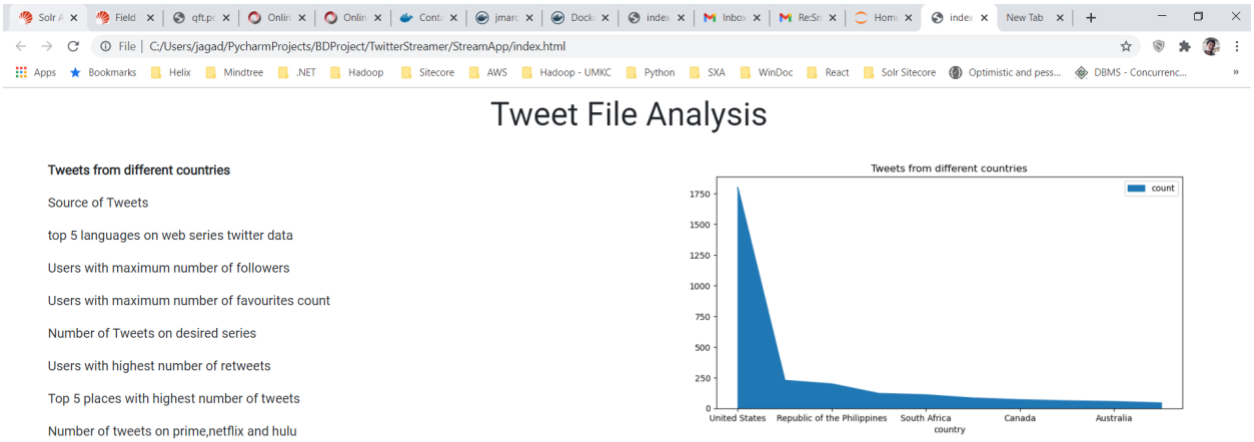
Using ps commands checking the containers. After that run the image on the 5000 port. And created the container.

```
jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker run -d -p 5000:5000 jmaroju/bdpproject
8597d86af368cb634a2f0f2e429c8fbf6affa0d2c3a0ce34e394a94e81acb5c4

jagad@DESKTOP-CNTBEH6 MINGW64 /c/Program Files/Docker Toolbox/dockerfiles
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
8597d86af368        jmaroju/bdpproject   "/bin/sh -c 'python …"   7 seconds ago       Up 5 seconds        0.0.0.0:5000->5000/tcp   loving_dewdney

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.



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)
- Convertion 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 compatability 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 filed
- 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 an 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 haves an interest on 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 platform, 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 understanding 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
3. spark.apache.org/docs/latest/
4. https://www.youtube.com/watch?v=lWzcZZEQYMk
5. https://docs.docker.com/docker-hub/publish/customer_faq/