School of Computing and Engineering

CS-5540: Principles of Big Data Management

Project: Twitter Data Analysis

(Phase II)

By

Sindhu Reddy Golconda – 16209698

Meghana Chitta – 16208686

Priyadarsini Nidadavolu - 16212029

Table of Contents

1) Introduction

Big Data is about massive large-enormous amounts of Data which is difficult to process using traditional methods of analysis. An example of Big Data would be Petabytes or Exabyte's of data comprising of billions and trillions of records of data from millions of users from many different sources. In our project we have done the twitter data analysis on **Domestic violence** using Big Data tools.

2) Tools and Languages

2.1) Spark SQL: Spark SQL is used to query structured data inside Spark programs. As the tweets are collected in JSON and the Data frames created by the spark SQL helps in accessing any type of data, we have chosen spark SQL. We have used Scala shell to execute the queries by creating a SQL context.

2.2) Tools and Languages used:

Front-End: HTML, CSS, D3.js.
Back- End: Apache spark, Scala.

3) Tweets Collection:

- We have used Python for the collection of tweets.
- We collected tweets about Domestic violence using tweepy.
- We have streamed the twitter data using Python and stored them as a comma separated tweets.

3.1) Code

Code that is used for collecting Tweets in Python

Tweets.py

```
from __future__ import absolute_import, print_function

from tweepy import OAuthHandler
from tweepy import Stream
from tweepy.streaming import StreamListener

consumer_key="hkce7PkPYfBvHjrLGh8Um4FD1"
consumer_secret="PGix2mcFegGb0pPh2HT7UJNkzmQlfT8MNEcqDKwXP99gPmWVUZ"
access_token="3535791433-5Xm2uZXXODJNkjc2BO3gOGuQDVEuOWccDOfKfAx"
access_token_secret="LharOF5IVt00dkA5DFCq1IME2xvBsyE9gPPuX3MujCXnm"

class StdOutListener(StreamListener):

    def on_data(self, data):
                print(data)
                saveFile = open('TweetsDV.json','a')
                saveFile.write(data)
                saveFile.write('\n')
                saveFile.close()
                        return True

    def on_error(self, status):
        print(status)

if __name__ == '__main__':
    x = StdOutListener()
    auth = OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_token_secret)
```

```
stream = Stream(auth, x)
stream.filter(track=['DomesticViolence'])
```

4) Storing of the Tweets and Data Retrieval:

We have collected the real time twitter tweets using twitter API in JSON format. We have stored the tweets in local disk and have executed the commands to generate a table and to run sql queries.

//Loading the tweets:

```
val path = "/home/sindhu/Desktop/Bigdataproject/TweetsDV.json"
```

//Create Table:

```
val tweets = sqlContext.jsonFile(path)
```

//**Schema of the Table:**

```
tweets.printSchema()
```

//To register the table temp memory:

```
tweets.registerTempTable("tweets")
```

5) Queries and Visualization :

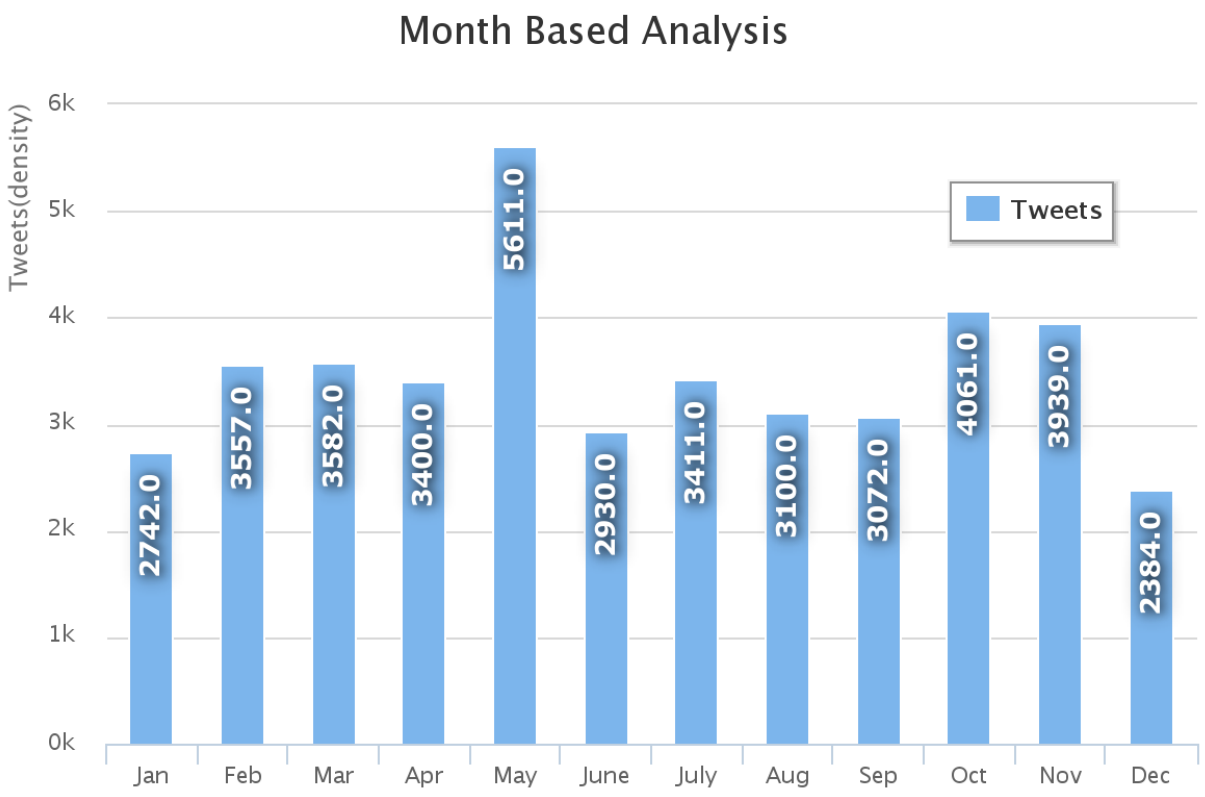
We had written eight queries for analyzing the twitter data and represented them graphically.

5.1 Query 1:

The query is written to analyze in which month of the year the highest response about Domestic violence is received on twitter. The query returns the month in which highest number of tweets are received along with the tweets count.

val Query1 = sqlContext.sql("select substring(user.created_at, 5,3), count(*) from tweets where user.created_at is not null group by substring(user.created_at, 5,3)")

Visualization:

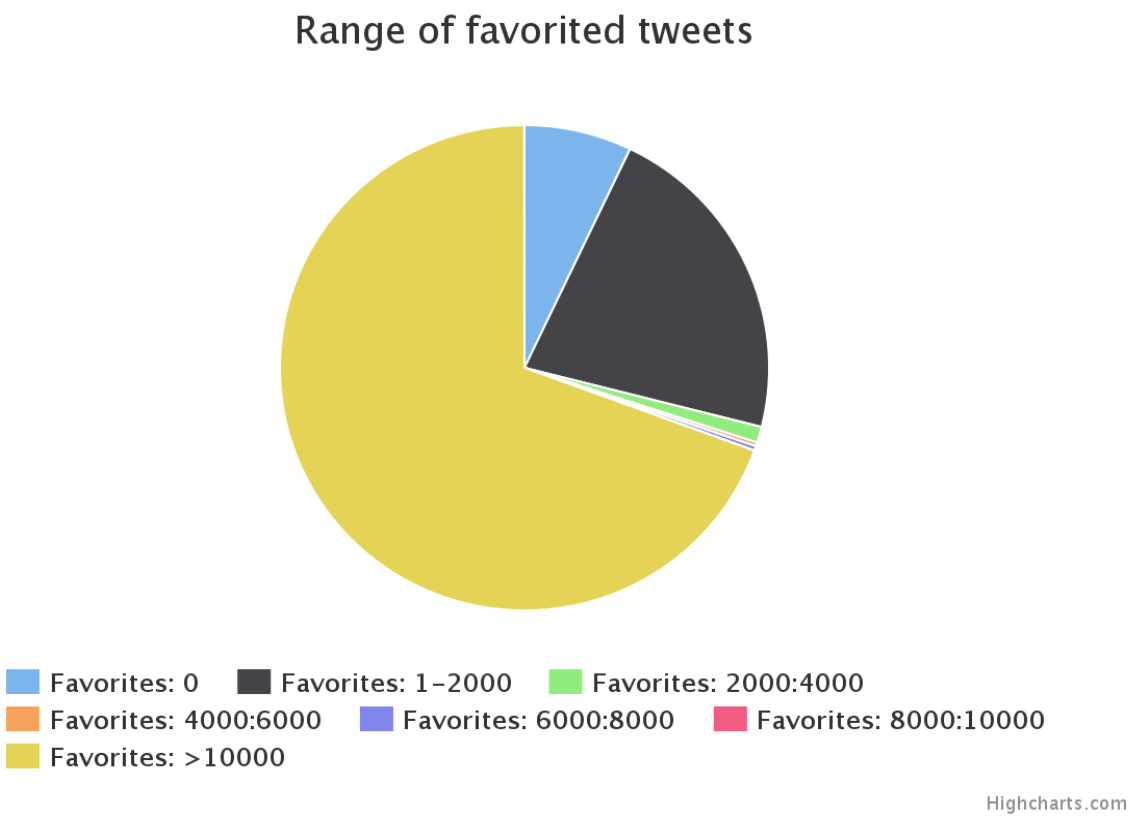


5.2 Query 2:

The query is to analyze the range of Favorite_count on the most famous tweets. We have queried for the retweeted status to know the range of favorite_count.

val Query2 = sqlContext.sql("select t.range as Favorites_range, count(*) as count from (select case when retweeted_status.favorite_count = 0 then ' 0 'when retweeted_status.favorite_count between 1 and 1000 then '1-1000' when retweeted_status.favorite_count between 1001 and 2000 then '1001-2000' when retweeted_status.favorite_count between 2001 and 3000 then '2001-3000' when retweeted_status.favorite_count between 3001 and 4000 then '3001-4000' when retweeted_status.favorite_count between 4001 and 5000 then '4001-5000' when retweeted_status.favorite_count between 5001 and 6000 then '5001-6000' when retweeted_status.favorite_count between 6001 and 7000 then '7001-8000' when retweeted_status.favorite_count between 8001 and 9000 then '8001-9000' when retweeted_status.favorite_count between 9001 and 10000 then '9001-10000' else '>10000' end as range from tweets) t group by t.range")

Visualization:

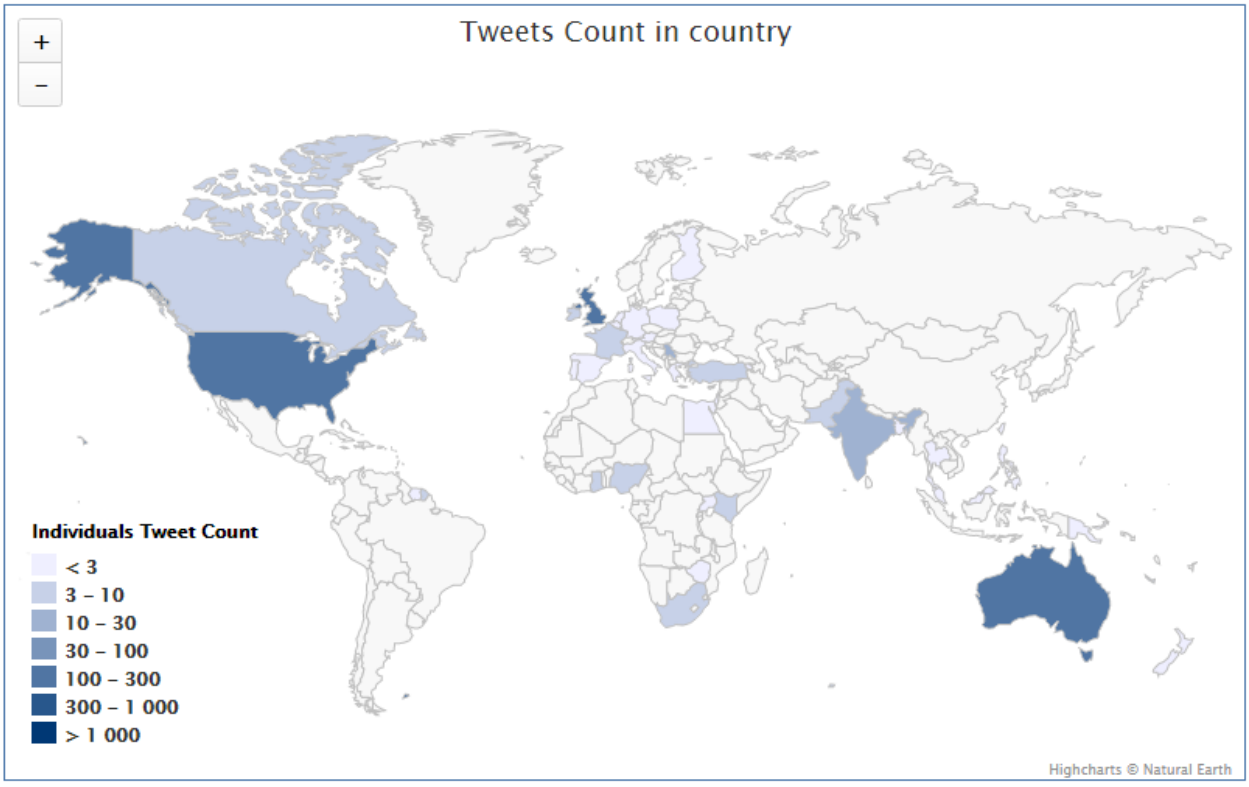


5.3 Query 3:

The query is to analyze geographically from which location the highest response on Domestic violence is tweeted by the users in twitter. The query returns the latitude and longitude data of the locations along with the count of the highest tweets recorded from that location and the count is displayed in descending order.

val Query3 = sqlContext.sql("SELECT place.country_code, place.country, count(*) as count FROM tweets WHERE place.country IS NOT NULL GROUP BY place.country, place.country_code ORDER BY place.country_code")

Visualization:



5.4 Query 4:

The query is to analyze the different hashtags related to the Domestic violence globally and to know the list of top tweeted hashtags for domestic violence.

val Query4 = sqlContext.sql("SELECT lower(entities.hashtags.text[0]), count(*) as count FROM tweets WHERE entities.hashtags.text[0] IS NOT NULL group by lower(entities.hashtags.text[0]) order by count DESC")

Visualization:

5.5 Query 5:

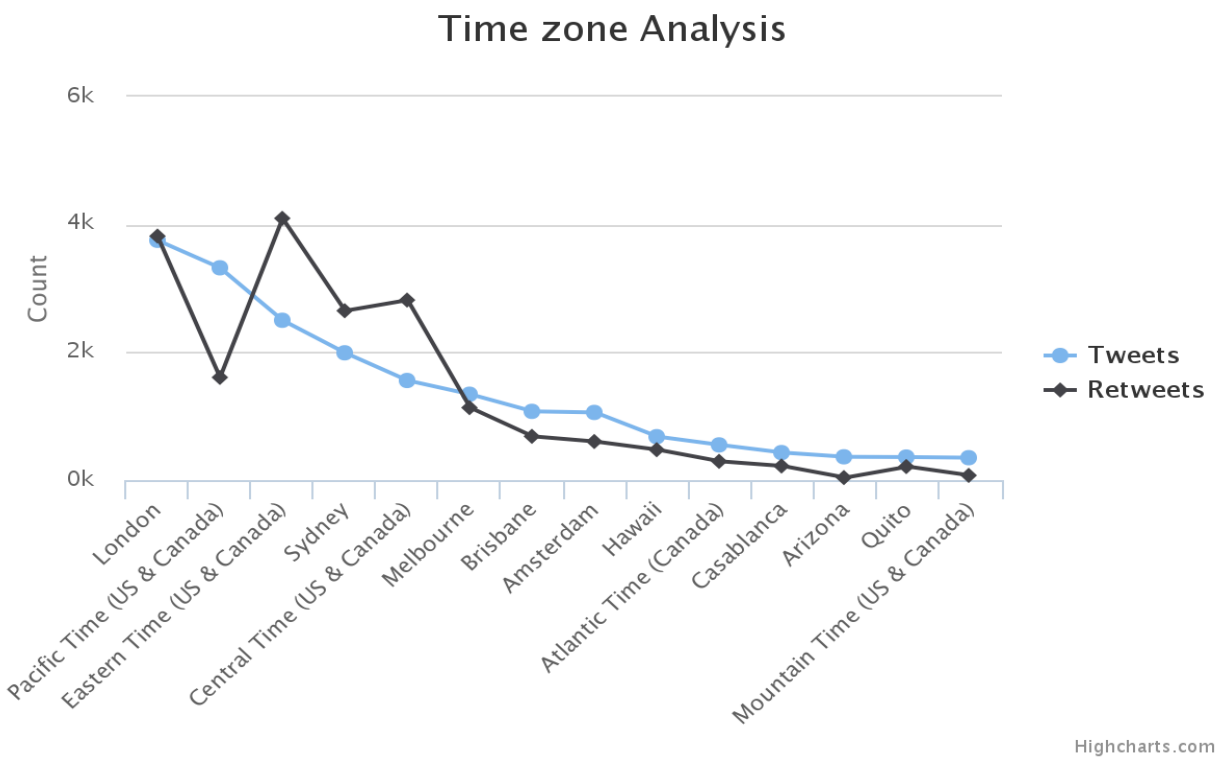
The query is to analyze the response over twitter on Domestic violence based on the Time zone. We have queried to obtain the total number of tweets and retweets that are obtained from the same time zone in order to analyze the user's positive responses on Domestic violence from a particular time zone.

val x1 = sqlContext.sql("select user.time_zone as time_zone, count(*) as Tweet_count from tweets where user.time_zone is not null group by user.time_zone order by Tweet_count desc")

val x2 = sqlContext.sql("select retweeted_status.user.time_zone as time_zone, count(*) as Retweet_count from tweets where retweeted_status.user.time_zone is not null group by retweeted_status.user.time_zone order by Retweet_count desc")

val Query5 = sqlContext.sql("select x1.time_zone, x1.Tweet_count, x2.Retweet_count from x1 inner join x2 on x1.time_zone = x2.time_zone order by x1.Tweet_count desc")

Visualization:

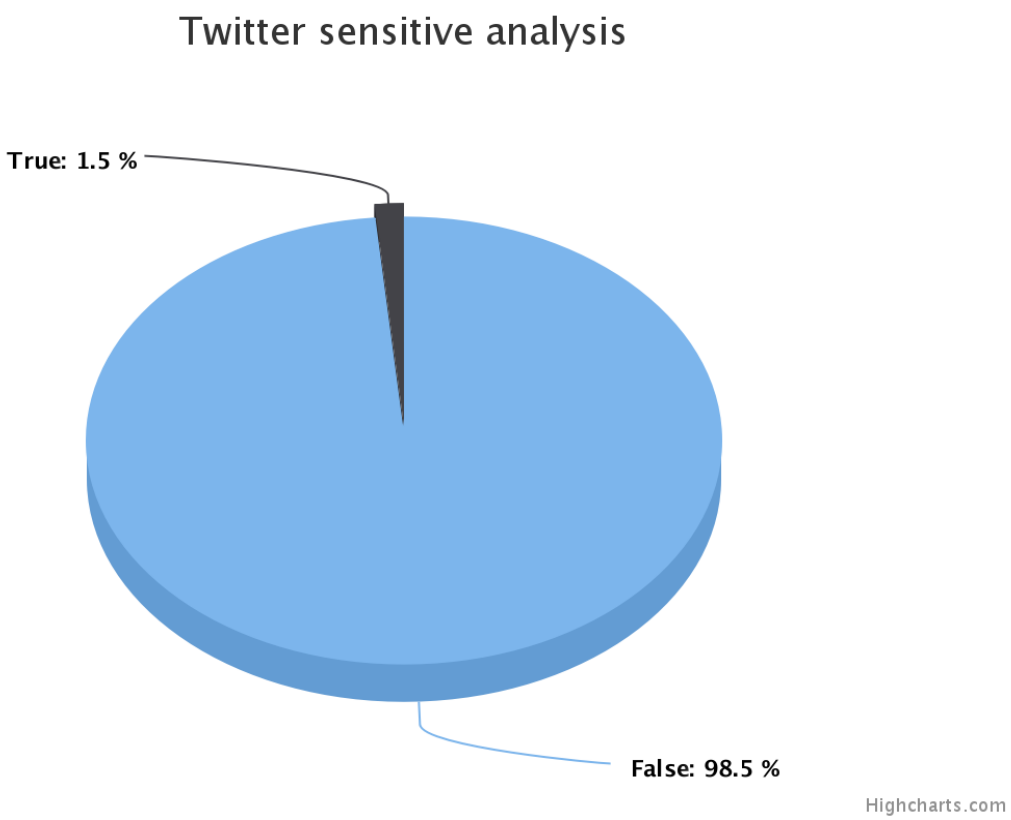


5.6 Query 6:

The query is to analyze how many twitter feels is possibly sensitive on domestic violence and the tweets on the same language. If the possibly sensitive is true Twitter takes all the privileges to remove the tweets on that topic from Twitter. We can analyze if the topic the users tweeting on is a sensitive one or not.

val Query6 = sqlContext.sql ("select possibly_sensitive, count(*) from tweets where lang = 'en' and possibly_sensitive is not null group by possibly_sensitive")

Visualization:

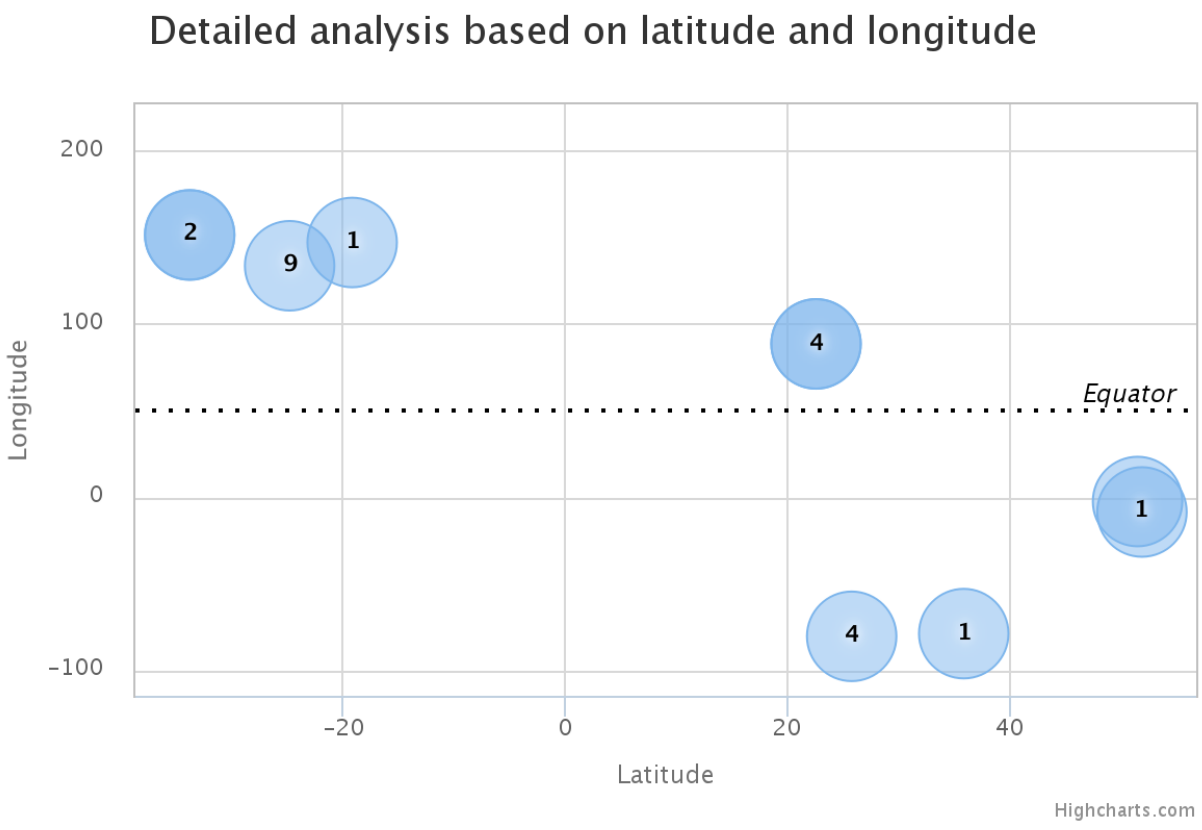


5.7 Query 7:

The query is to analyze from which geographical locations the overall tweets are recorded. It is to analyze the latitude and longitude information globally about the people's responses on Domestic violence.

val Query7 = sqlContext.sql("select geo.coordinates[0], geo.coordinates[1], count(*) as count from tweets where geo is not null group by geo.coordinates order by count desc limit 10")

Visualization:

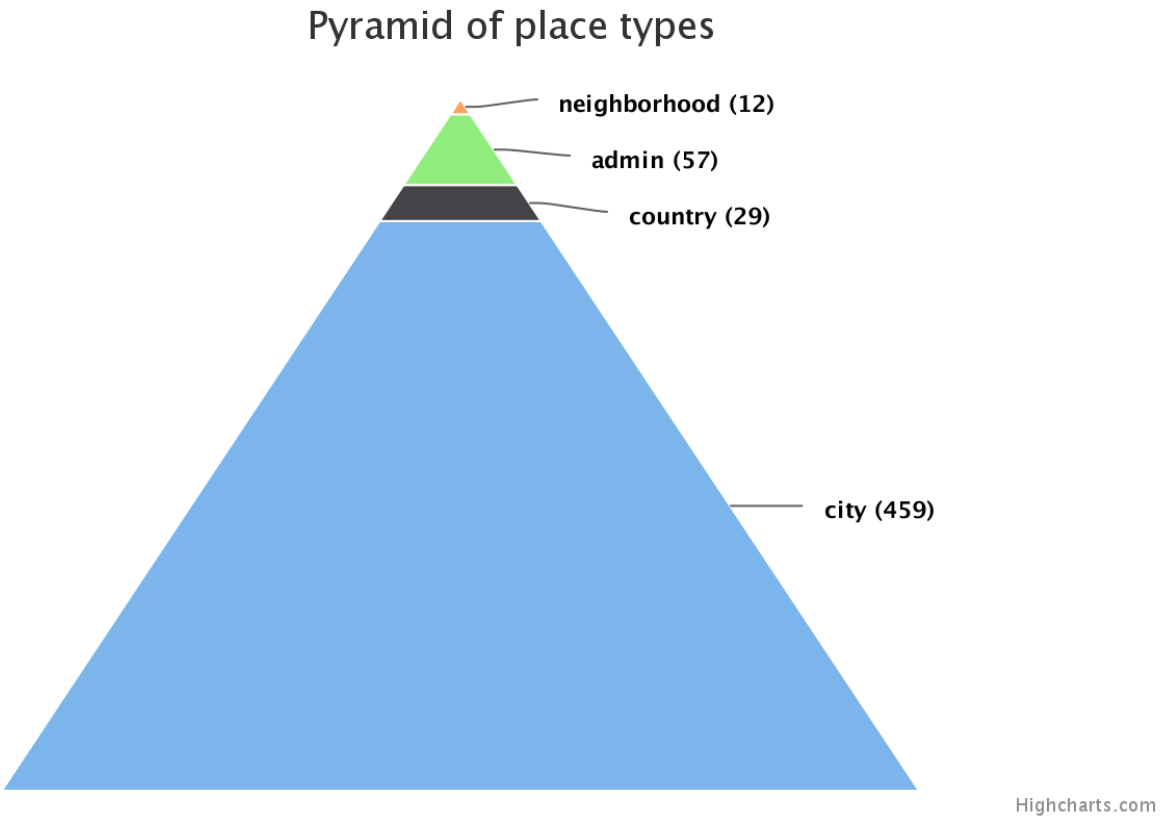


5.8 Query 8:

The query is to analyze how the tweets are obtained at country level, city level and neighborhood level. It is to know the distribution of overall tweets across the globe. We can know the statistics about the number of tweets from a country, city and neighborhood.

Val Query8 = sqlContext.sql ("SELECT place.place_type, count(*) from tweets where place.place_type is not null group by place.place_type")

Visualization:



6) Testing :

We have done the testing manually. We have collected the real time tweets and have run our queries on the sample data to check if the output matched with the expected results. Later we have tested the visualization for the sample data over queries. The output is matched with the expected output.