US ELECTION TWEETS ANALYSIS

Introduction:

Big data is an evolving term that describes any voluminous amount of structured, semi-structured and unstructured data that has the potential to be mined for information. But it's not the amount of data that's important. It's what organizations do with the data that matters. Big data is all about 3Vs: the extreme volume of data, the wide variety of types of data and the velocity at which the data must be must processed. Companies are trying to perform analytics on big data and get some valuable output which gives an edge over their competitors. Big data can generate significant financial value across sectors. We can achieve improved revenues at lower costs using Big Data analytics. Accuracy in big data may lead to more confident decision making.

In this project we have analysed twitter data based on US elections criteria. We have drawn different conclusions and visualized the results.

Technology and Tools Used:

1. Environment: Ubuntu
2. Tools: IntelliJ, Webstorm, Spark 1.6.1
3. Java REST API for backend servies
4. Angular js for front end
5. Plotly.js, Amcharts, Charts.js, Google Maps for visualization

Implementation:

1. Collected tweets from Twitter Streaming API using Java Scribe library with filters as 'US Elections & Contestants name'**.** This collected Json data is stored to a Json file.
2. Using Java created a class which uses spark library that runs selected queries.
3. The output is modelled into Json format.
4. Java Servlet is created which invokes SparkJava class which returns output data in Json format.
5. Created web interface with angular js which uses this servlets to get data and visualizes the output in interesting charts using different charting js libraries.
6. Web interface has dropdown list of queries on selecting query, web servlet invokes java program which executes the query using spark library and returns Json data for visualization.

Github URL: https://github.com/nagakrishna/PB-Project.git

Tweet File: https://www.dropbox.com/s/2mlqiqmx4t48hn8/concat.txt?dl=0

Blumeix URL: http://naga.mybluemix.net/

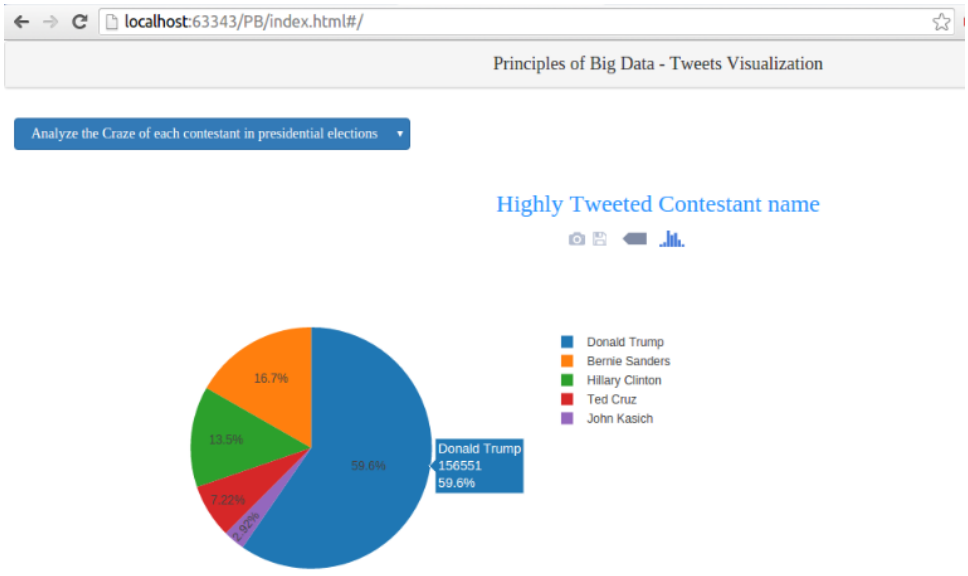
References:
Tweet collection:
http://www.jeffkuchta.com/Tutorial/2013/02/21/java--twitter-public-streams-with-support-from-heroku-spark--oauth/

Java programming:
https://spark.apache.org/docs/0.9.1/java-programming-guide.html

Queries:

Query1:
SELECT count(*) total, sum(case when lower(text) like '%trump%' or text like '%donald%' then 1 else 0 end) Trump, sum(case when lower(text) like '%sanders%' or lower(text) like '%bernie%'then 1 else 0 end) Sanders, sum(case when lower(text) like '%hillary%' or text like '%clinton%' then 1 else 0 end) Clinton, sum(case when lower(text) like '%ted%' or text like '%cruz%' then 1 else 0 end) Cruz, sum(case when lower(text) like '%john%' or text like '%kasich%' then 1 else 0 end) John from tweets

Description:
This query finds tweets which has particular contestant in US presidential elections word in it, find the count of each contestant. Based on the count each contestant craze is determined.

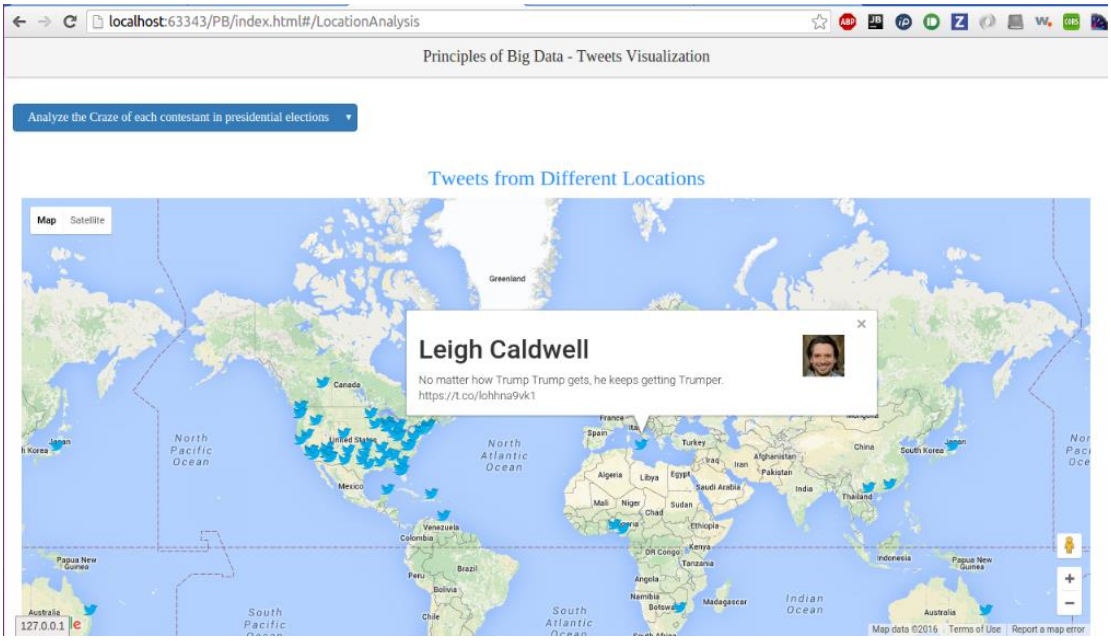


Query2:

SELECT user.name, text, geo, user.profile_image_url from tweets where geo IS NOT NULL

Description:

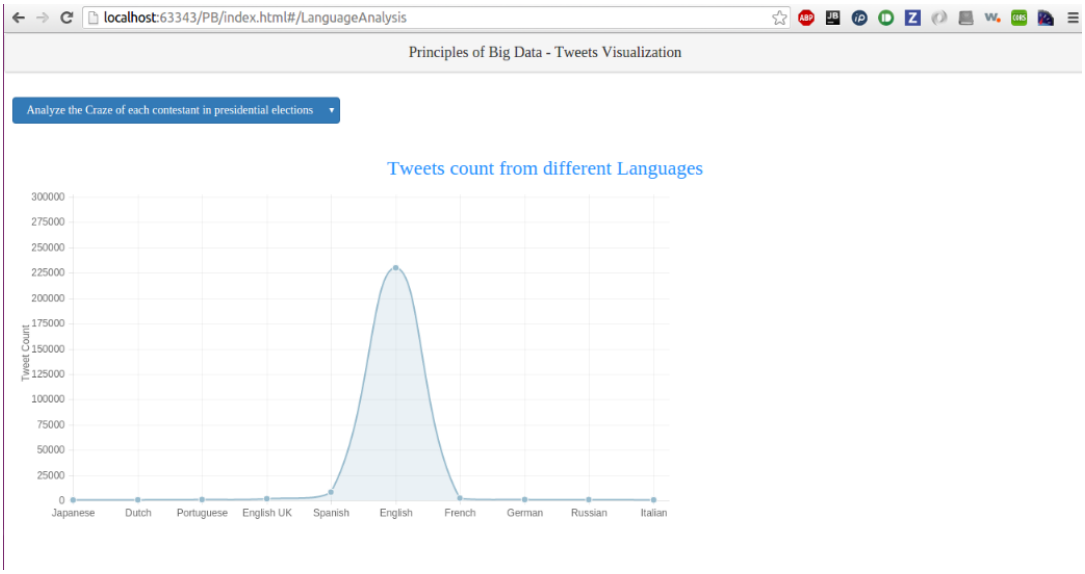
This query collects tweeted user name, user location coordinates, tweeted text and is profile image url. This data is used to display on world map using coordinates.



Query 3:

SELECT user.lang, count(*) as lang_user_count from tweets WHERE user.lang IS NOT NULL GROUP BY user.lang ORDER BY lang_user_count DESC LIMIT 10

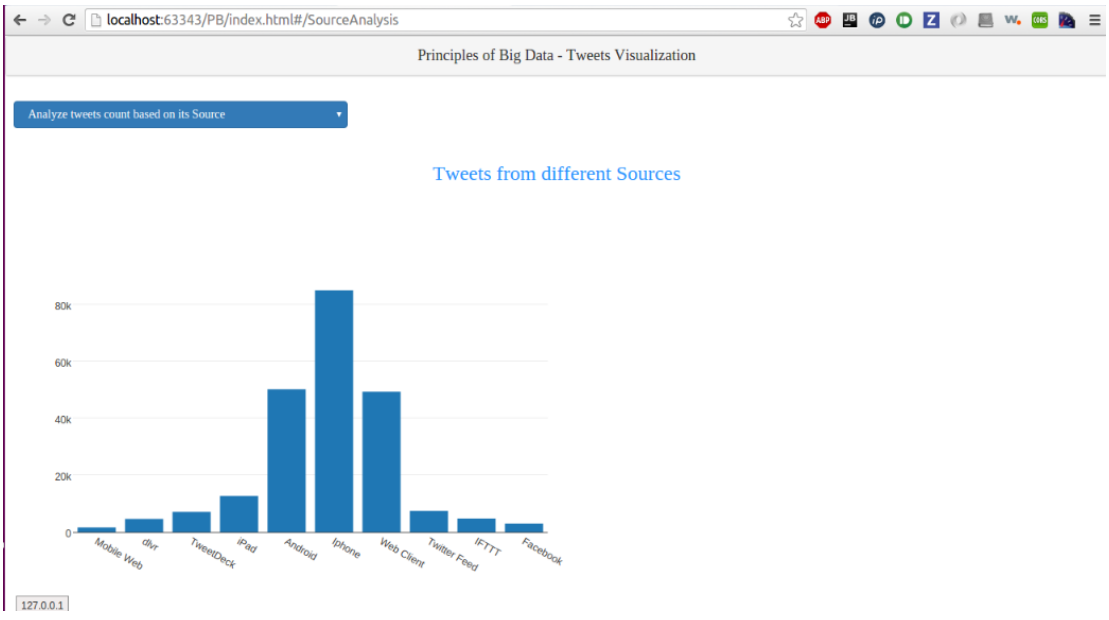
Description: This query collects the language information of each tweet tweeted along with the total count in each language.



Query 4:

SELECT source, count(source) as c from tweets group by source order by c desc limit 10

Description: This query collects information about the source of tweet from where it generated.

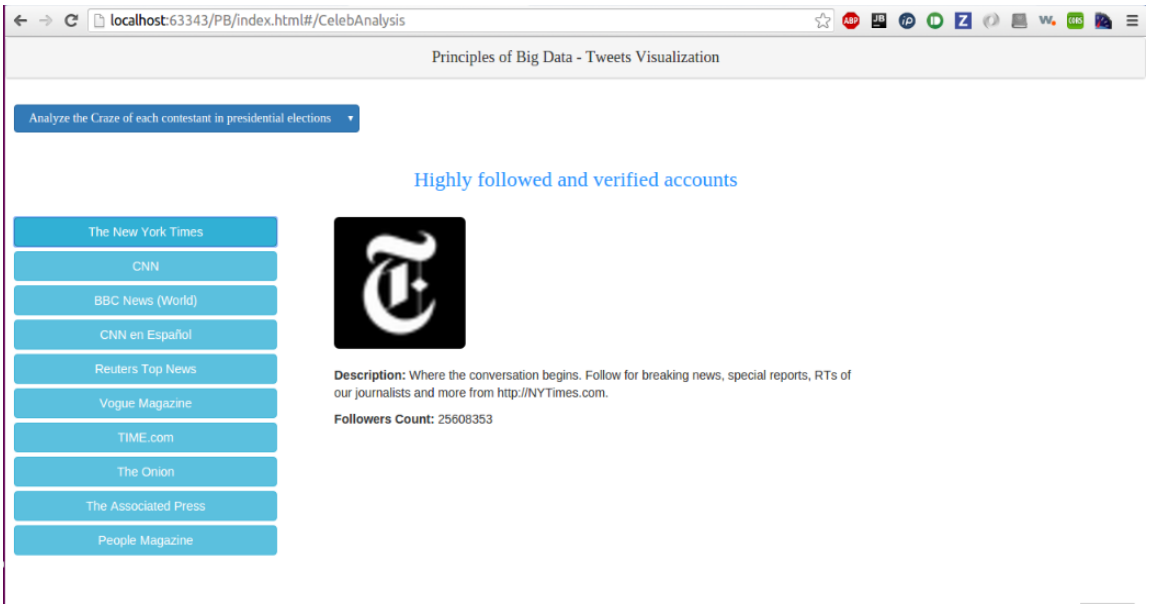


Query 5:

SELECT distinct user.name,user.followers_count as c, user.profile_image_url, user.description FROM tweets where user.verified = true ORDER BY c desc limit 25

Description: This query collects information about all verified accounts along with the

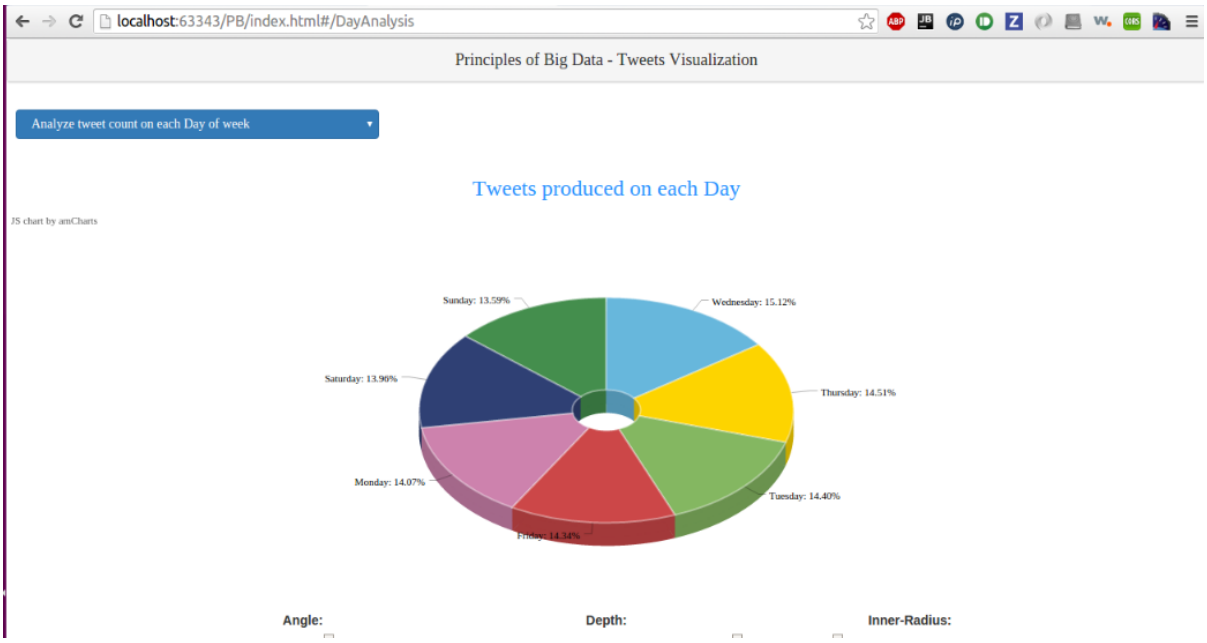
description of account, name, followers count and profile image url**.**



Query 5:

SELECT substr(user.created_at, 1, 3) as day, count(*) as count from tweets where user.created_at is not null group by substr(user.created_at, 1, 3) order by count DESC

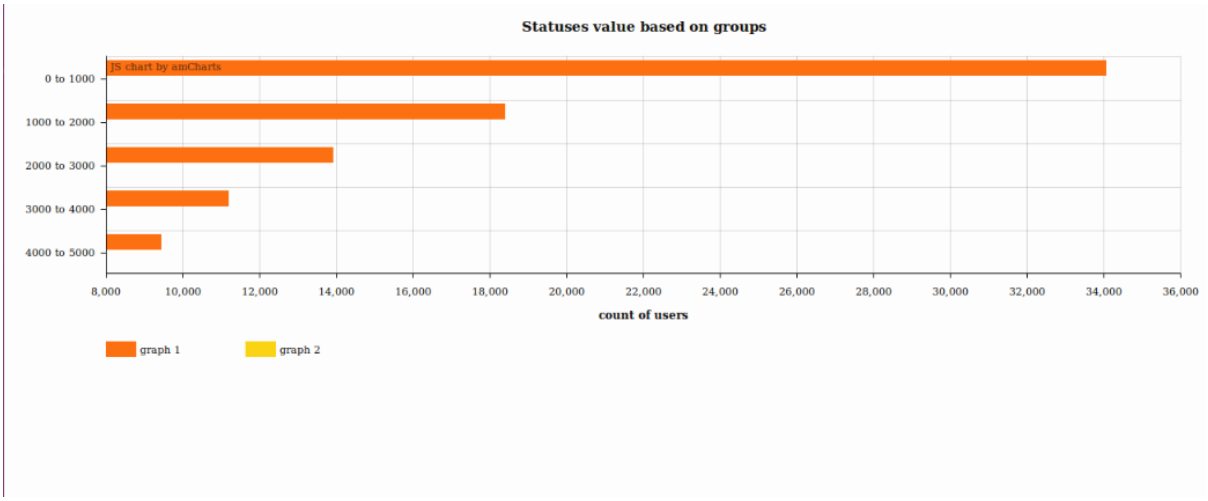
Description: This query collects information about on which day is the tweet created



Query 7:

```
val userName = sqlContext.sql("SELECT count(user.statuses_count) from tweets WHERE user.statuses_count IS NOT NULL and user.statuses_count between 0 and 1000")
val userName = sqlContext.sql("SELECT count(user.statuses_count) from tweets WHERE user.statuses_count IS NOT NULL and user.statuses_count between 1000 and 2000")
val userName = sqlContext.sql("SELECT count(user.statuses_count) from tweets WHERE user.statuses_count IS NOT NULL and user.statuses_count between 2000 and 3000")
val userName = sqlContext.sql("SELECT count(user.statuses_count) from tweets WHERE user.statuses_count IS NOT NULL and user.statuses_count between 3000 and 4000")
val userName = sqlContext.sql("SELECT count(user.statuses_count) from tweets WHERE user.statuses_count IS NOT NULL and user.statuses_count between 4000 and 5000")
```

Description:

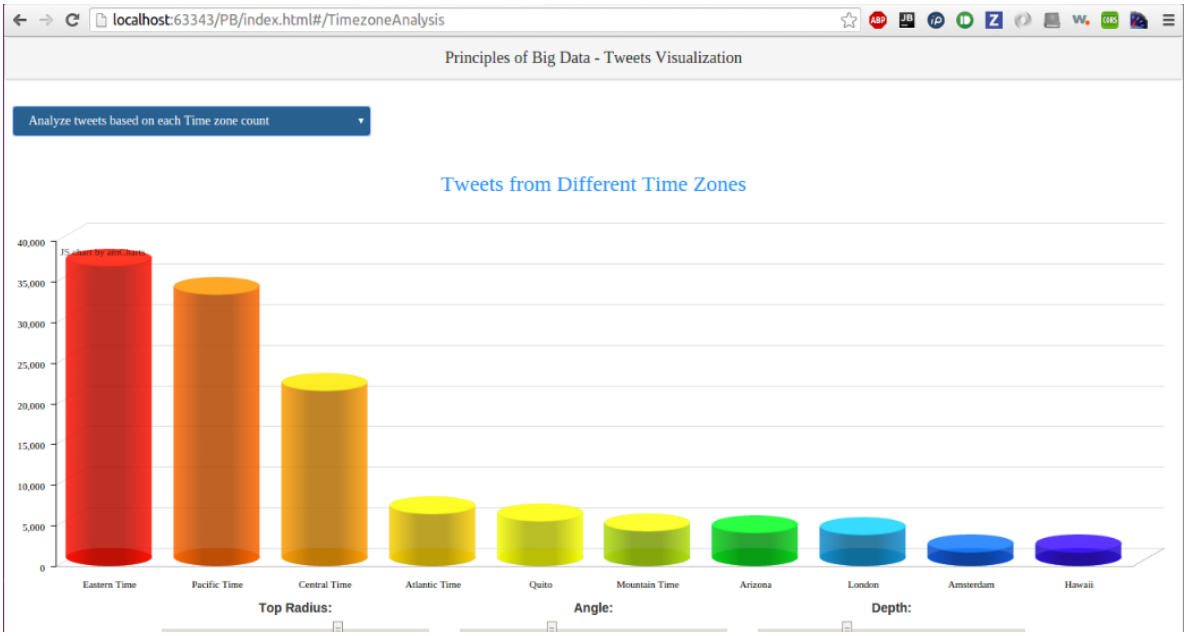


Query 8:

```
SELECT user.time_zone, count(*) AS count from tweets where user.time_zone is not null GROUP BY user.time_zone ORDER BY count DESC limit 10
```

Desccription: This query collects information about time zone from where each tweet is generated and outputs the count of each time zone

.



Query 9:

```
val userName = sqlContext.sql("SELECT avg(user.friends_count) from tweets WHERE user.friends_count IS NOT NULL and user.friends_count between 0 and 1000")
val userName = sqlContext.sql("SELECT avg(user.friends_count) from tweets WHERE user.friends_count IS NOT NULL and user.friends_count between 1000 and 2000")
val userName = sqlContext.sql("SELECT avg(user.friends_count) from tweets WHERE user.friends_count IS NOT NULL and user.friends_count between 2000 and 3000")
val userName = sqlContext.sql("SELECT avg(user.friends_count) from tweets WHERE user.friends_count IS NOT NULL and user.friends_count between 3000 and 4000")
val userName = sqlContext.sql("SELECT avg(user.friends_count) from tweets WHERE user.friends_count IS NOT NULL and user.friends_count between 4000 and 5000")
```