Phase II- Design and Implementation of Twitter Analysis using Spark

THE PANAMA PAPERS
Politicians, Criminals and the Rogue Industry That Hides Their Cash

THE PANAMA PAPERS

By

Rahul Ramini

Manoj Kumar Ravilla

Sree Ravi Kishore Lellapalli

Github link of source code: https://github.com/rahulramini/PBPhase2

Introduction:

The panama papers are leaked set of millions of confidential documents that provide information about the offshore companies listed by the Panamanian corporate service provider Mossack Fonseca, including the identities of shareholders and directors of the companies.

The papers identified five then-heads of state or government leaders from Argentina, Iceland, Saudi Arabia, Ukraine and the United Arab Emirates as well as government officials, close relatives, and close associates of various heads of government of more than forty other countries including Brazil, China, France, India, Malaysia, Mexico, Pakistan, Peru, Romania, Russia, South Africa, Spain, Syria, and the United Kingdom. The UK was singled out by the media as being "at the heart of [a] super-rich tax-avoidance network.

Here in this project we have analyzed tweets related to 'THE PANAMA PAPERS' to draw some insights from the data.

Project Phases:

We have done our projects in three phases.

- Data Collection
- Data Analysis
- Data Visualization

Data Collection:

In this phase we have collected the tweets in json format for our analysis. We have used tweepy API in python to download the tweets. We have used OAuth to authorize the python code to connect to twitter. We have gathered all the new tweets related to our keywords using StreamListener().

We have collected 100K tweets in a json file for our analysis.

Data Analysis:

We have analyzed the tweets collected in Apache Spark.

We have made use of the SqlContext available in the Spark. First, we have read the collected tweets into a SqlContext Value. Next, we have stored it as a temporary table and ran sql queries on the table to analyze the tweets.

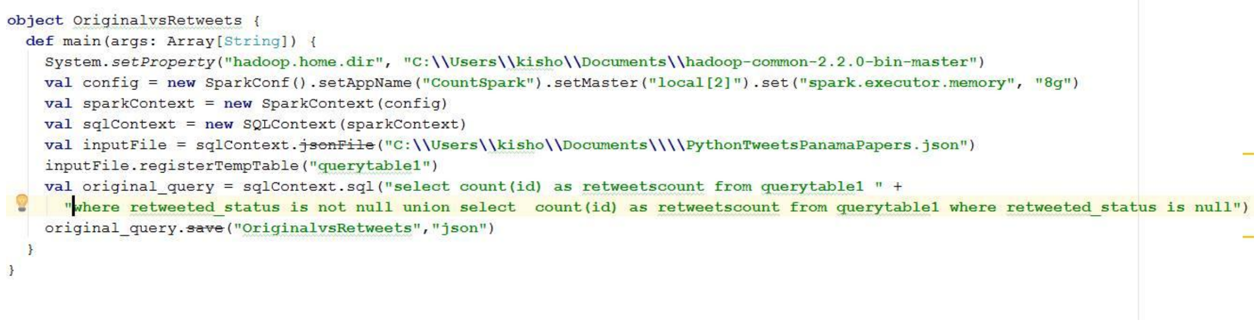
Analysis Queries:

We have analyzed our tweets using the following sql queries:

1st Query: (% of retweets vs original tweets)

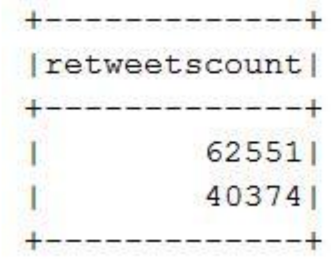
In twitter, one can retweet the previous tweet.

Interestingly, we found that more than 60 % of the tweets are retweets not the original ones. We have counted the number of retweets by counting tweets where retweeted_status is not null.



```
object OriginalvsRetweets {
  def main(args: Array[String]) {
    System.setProperty("hadoop.home.dir", "C:\\Users\\kisho\\Documents\\hadoop-common-2.2.0-bin-master")
    val config = new SparkConf().setAppName("CountSpark").setMaster("local[2]").set("spark.executor.memory", "8g")
    val sparkContext = new SparkContext(config)
    val sqlContext = new SQLContext(sparkContext)
    val inputFile = sqlContext.jsonFile("C:\\Users\\kisho\\Documents\\\\PythonTweetsPanamaPapers.json")
    inputFile.registerTempTable("querytable1")
    val original_query = sqlContext.sql("select count(id) as retweetscount from querytable1 " +
      "where retweeted_status is not null union select  count(id) as retweetscount from querytable1 where retweeted_status is null")
    original_query.save("OriginalvsRetweets","json")
  }
}
```

Output:

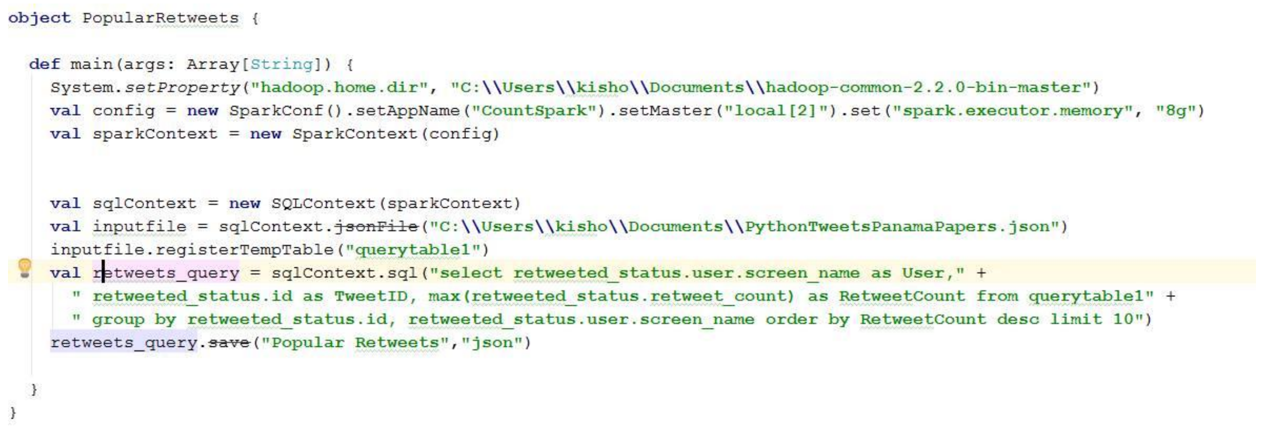


```
+-------------+
|retweetscount|
+-------------+
|        62551|
|        40374|
+-------------+
```

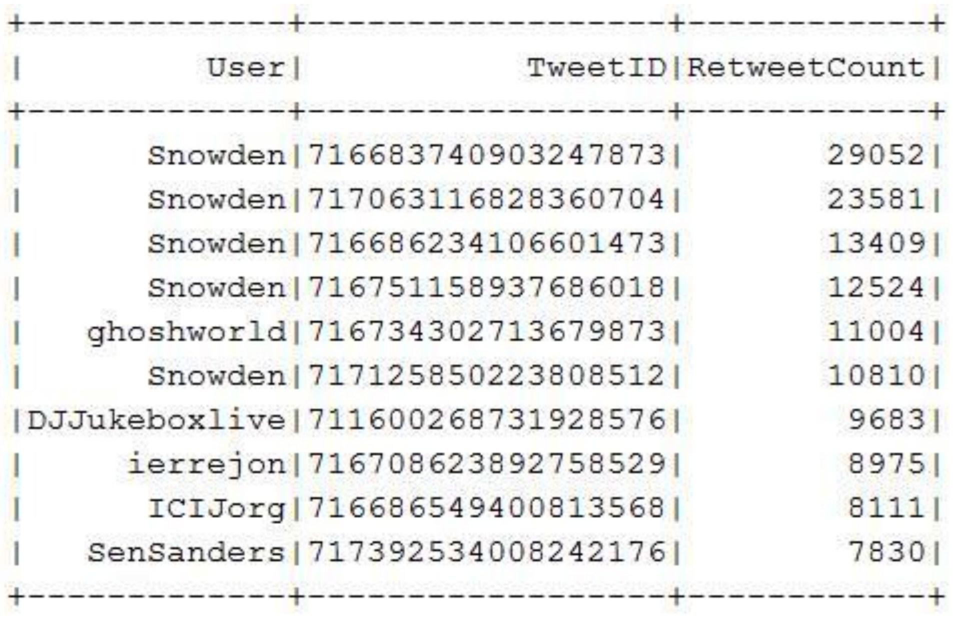
2nd Query: (No. of retweets of a particular user tweet)

Analyzing the popularity based on count of retweets.

No. of times a tweet by a particular user is retweeted.



```
object PopularRetweets {

  def main(args: Array[String]) {
    System.setProperty("hadoop.home.dir", "C:\\Users\\kisho\\Documents\\hadoop-common-2.2.0-bin-master")
    val config = new SparkConf().setAppName("CountSpark").setMaster("local[2]").set("spark.executor.memory", "8g")
    val sparkContext = new SparkContext(config)


    val sqlContext = new SQLContext(sparkContext)
    val inputfile = sqlContext.~~jsonFile~~("C:\\Users\\kisho\\Documents\\PythonTweetsPanamaPapers.json")
    inputfile.registerTempTable("querytable1")
    val retweets_query = sqlContext.sql("select retweeted_status.user.screen_name as User," +
      " retweeted_status.id as TweetID, max(retweeted_status.retweet_count) as RetweetCount from querytable1" +
      " group by retweeted_status.id, retweeted_status.user.screen_name order by RetweetCount desc limit 10")
    retweets_query.~~save~~("Popular Retweets","json")

  }
}
```

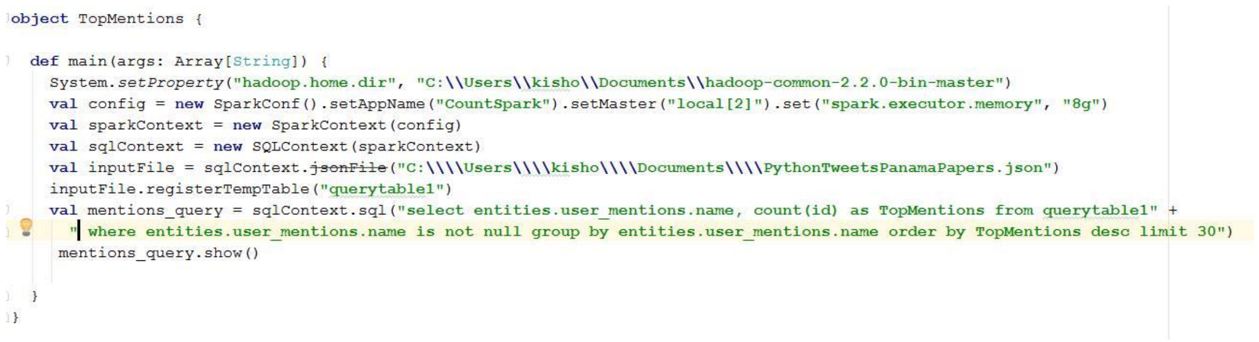


```
+------------+------------------+------------+
|        User|           TweetID|RetweetCount|
+------------+------------------+------------+
|     Snowden|716683740903247873|       29052|
|     Snowden|717063116828360704|       23581|
|     Snowden|716686234106601473|       13409|
|     Snowden|716751158937686018|       12524|
| ghoshworld|716734302713679873|       11004|
|     Snowden|717125850223808512|       10810|
|DJJukeboxlive|711600268731928576|        9683|
|    ierrejon|716708623892758529|        8975|
|    ICIJorg|716686549400813568|        8111|
|  SenSanders|717392534008242176|        7830|
+------------+------------------+------------+
```

3rd Query: (User Mentions)

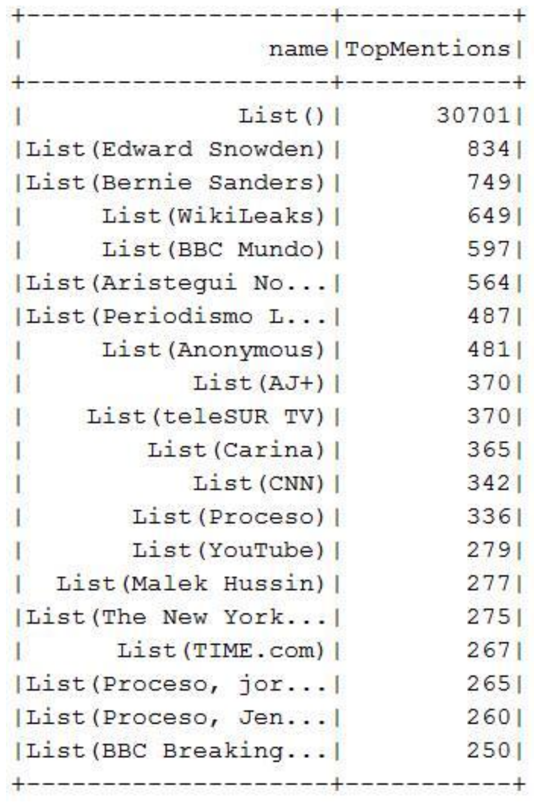
Through tweet a user not only expresses his/her opinions but also mentions other users and express their opinions.

In tweet, entities.user_mentions. name contains user mentioned screen name. In tweet’s text field user mention starts with ‘@’.



```
object TopMentions {

  def main(args: Array[String]) {
    System.setProperty("hadoop.home.dir", "C:\\Users\\kisho\\Documents\\hadoop-common-2.2.0-bin-master")
    val config = new SparkConf().setAppName("CountSpark").setMaster("local[2]").set("spark.executor.memory", "8g")
    val sparkContext = new SparkContext(config)
    val sqlContext = new SQLContext(sparkContext)
    val inputFile = sqlContext.~~jsonFile~~("C:\\\\Users\\\\kisho\\\\Documents\\\\PythonTweetsPanamaPapers.json")
    inputFile.registerTempTable("querytable1")
    val mentions_query = sqlContext.sql("select entities.user_mentions.name, count(id) as TopMentions from querytable1" +
      " where entities.user_mentions.name is not null group by entities.user_mentions.name order by TopMentions desc limit 30")
     mentions_query.show()

  }
}
```

Output:



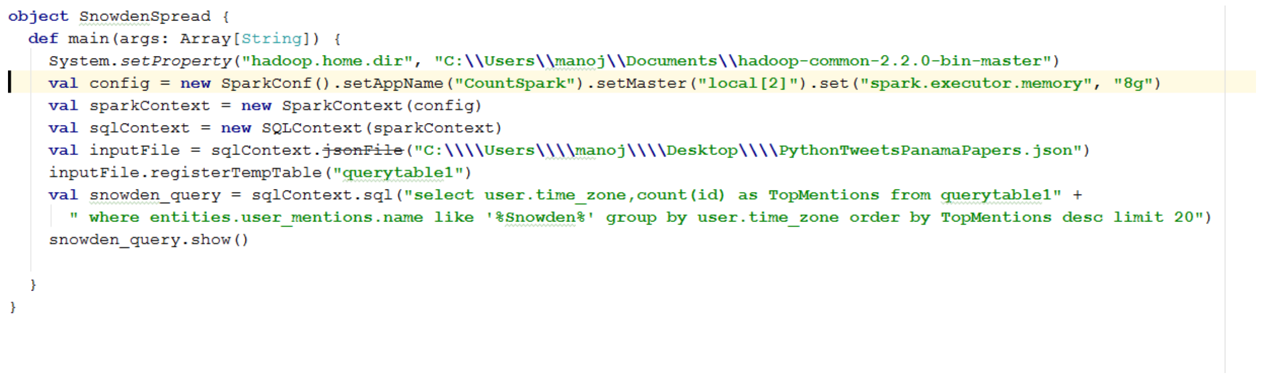
```
+--------------------+-----------+
|                name|TopMentions|
+--------------------+-----------+
|              List()|      30701|
|List(Edward Snowden)|        834|
|List(Bernie Sanders)|        749|
|      List(WikiLeaks)|        649|
|      List(BBC Mundo)|        597|
|List(Aristegui No...|        564|
|List(Periodismo L...|        487|
|     List(Anonymous)|        481|
|           List(AJ+)|        370|
|    List(teleSUR TV)|        370|
|        List(Carina)|        365|
|           List(CNN)|        342|
|       List(Proceso)|        336|
|       List(YouTube)|        279|
|  List(Malek Hussin)|        277|
|List(The New York...|        275|
|      List(TIME.com)|        267|
|List(Proceso, jor...|        265|
|List(Proceso, Jen...|        260|
|List(BBC Breaking...|        250|
+--------------------+-----------+
```

4th **Query:** (Location spread)

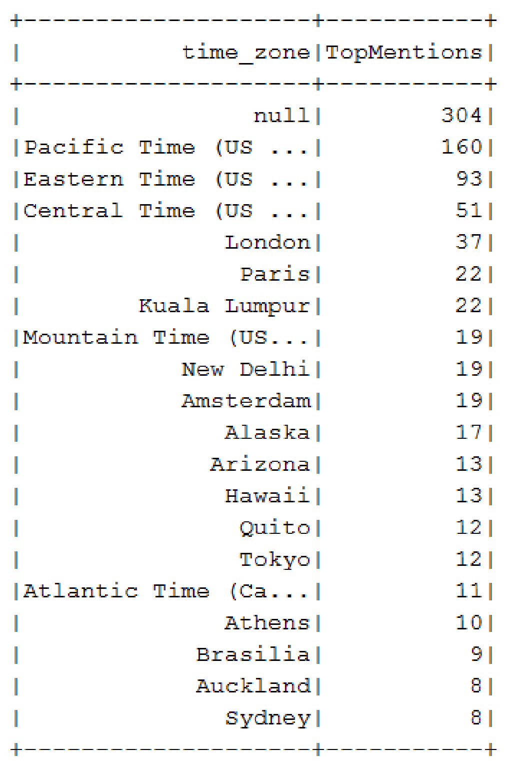
In our analysis, we thought that the user mentions will contain ‘snowden’ more based on the outputs of Queries 2 & 3.

So we ran a query to analyze the no. of times Snowden is mentioned in

For this we have used like operator in where condition.



```
object SnowdenSpread {
  def main(args: Array[String]) {
    System.setProperty("hadoop.home.dir", "C:\\Users\\manoj\\Documents\\hadoop-common-2.2.0-bin-master")
    val config = new SparkConf().setAppName("CountSpark").setMaster("local[2]").set("spark.executor.memory", "8g")
    val sparkContext = new SparkContext(config)
    val sqlContext = new SQLContext(sparkContext)
    val inputFile = sqlContext.~~jsonFile~~("C:\\\\Users\\\\manoj\\\\Desktop\\\\PythonTweetsPanamaPapers.json")
    inputFile.registerTempTable("querytable1")
    val snowden_query = sqlContext.sql("select user.time_zone,count(id) as TopMentions from querytable1" +
      " where entities.user_mentions.name like '%Snowden%' group by user.time_zone order by TopMentions desc limit 20")
    snowden_query.show()

  }
}
```

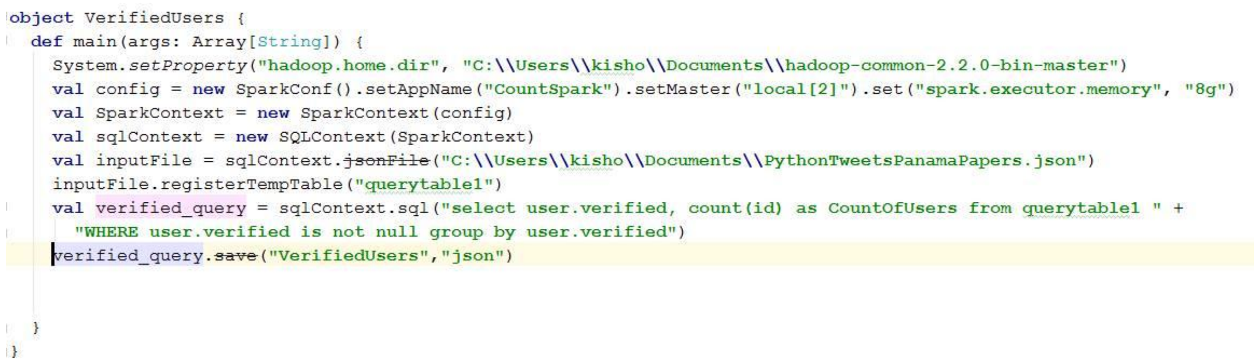


```
+--------------------+-----------+
|           time_zone|TopMentions|
+--------------------+-----------+
|                null|        304|
|Pacific Time (US ...|        160|
|Eastern Time (US ...|         93|
|Central Time (US ...|         51|
|              London|         37|
|               Paris|         22|
|        Kuala Lumpur|         22|
|Mountain Time (US...|         19|
|           New Delhi|         19|
|           Amsterdam|         19|
|              Alaska|         17|
|             Arizona|         13|
|              Hawaii|         13|
|               Quito|         12|
|               Tokyo|         12|
|Atlantic Time (Ca...|         11|
|              Athens|         10|
|            Brasilia|          9|
|            Auckland|          8|
|              Sydney|          8|
+--------------------+-----------+
```

5th Query: (tweets from verified users)

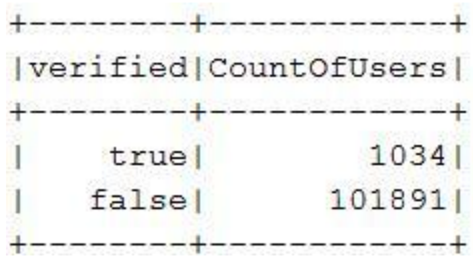
Recently released ‘Panama Papers’ revealed how politicians, banks, criminals and sports celebrities have taken advantage of the secrecy provided by tax havens and, in some cases, broken the law. Many big personalities might have reacted on twitter on this issue. So this query is to just see out of our 100K tweets how many of them are from verified accounts and how many are not.

We have used the ‘user.verified’ flag to analyze this. We have calculated the number of tweets from both verified and non verified users.



```
object VerifiedUsers {
  def main(args: Array[String]) {
    System.setProperty("hadoop.home.dir", "C:\\Users\\kisho\\Documents\\hadoop-common-2.2.0-bin-master")
    val config = new SparkConf().setAppName("CountSpark").setMaster("local[2]").set("spark.executor.memory", "8g")
    val SparkContext = new SparkContext(config)
    val sqlContext = new SQLContext(SparkContext)
    val inputFile = sqlContext.~~jsonFile~~("C:\\Users\\kisho\\Documents\\PythonTweetsPanamaPapers.json")
    inputFile.registerTempTable("querytable1")
    val verified_query = sqlContext.sql("select user.verified, count(id) as CountOfUsers from querytable1 " +
      "WHERE user.verified is not null group by user.verified")
    verified_query.~~save~~("VerifiedUsers","json")


  }
}
```

Output:



```
+--------+------------+
|verified|CountOfUsers|
+--------+------------+
|    true|        1034|
|   false|      101891|
+--------+------------+
```

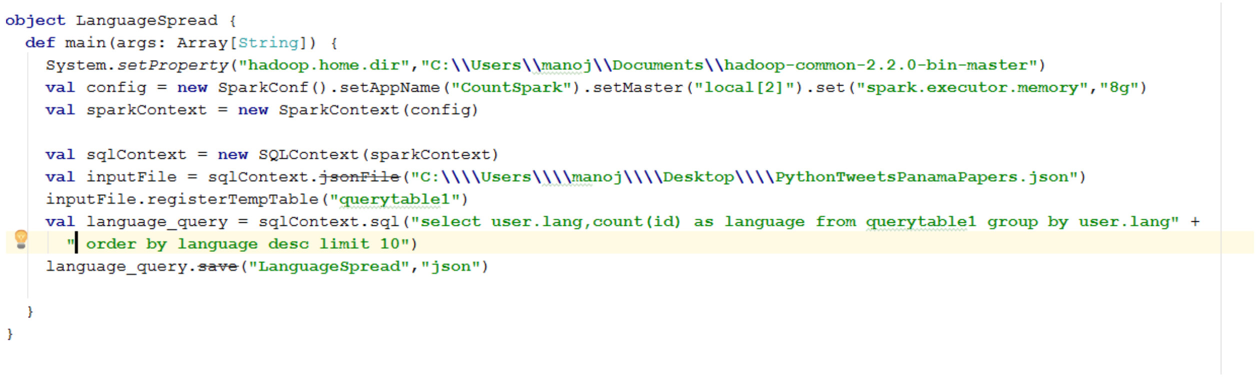
6th Query: (language spread)

We assume that 100K tweets are contributed from all over the globe.

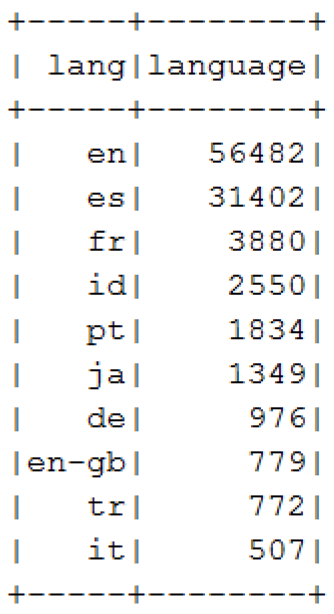
To check how diverse the tweets based on languages, we have ran a query to count the no. of tweets grouped by user.time_zone

From that we got no. of tweets written in each language.

The only limitation from this query is most of the tweets has language as null because twitter was unable to recognize or may be due to any other reason. We have considered only the tweets where language is not null.



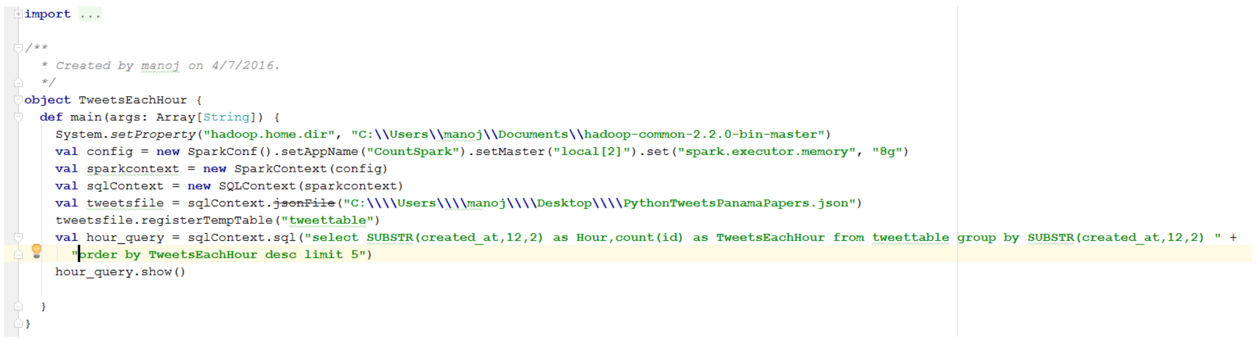
```scala
object LanguageSpread {
  def main(args: Array[String]) {
    System.setProperty("hadoop.home.dir","C:\\Users\\manoj\\Documents\\hadoop-common-2.2.0-bin-master")
    val config = new SparkConf().setAppName("CountSpark").setMaster("local[2]").set("spark.executor.memory","8g")
    val sparkContext = new SparkContext(config)

    val sqlContext = new SQLContext(sparkContext)
    val inputFile = sqlContext.jsonFile("C:\\\\Users\\\\manoj\\\\Desktop\\\\PythonTweetsPanamaPapers.json")
    inputFile.registerTempTable("querytable1")
    val language_query = sqlContext.sql("select user.lang,count(id) as language from querytable1 group by user.lang" +
      " order by language desc limit 10")
    language_query.save("LanguageSpread","json")

  }
}
```



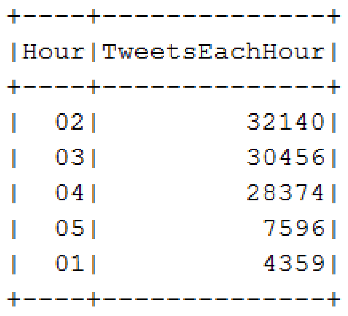
```
+-----+--------+
| lang|language|
+-----+--------+
|   en|   56482|
|   es|   31402|
|   fr|    3880|
|   id|    2550|
|   pt|    1834|
|   ja|    1349|
|   de|     976|
|en-gb|     779|
|   tr|     772|
|   it|     507|
+-----+--------+
```

7th **Query** (tweets each hour)

No. of tweets each hour which provides on what hour most no. of tweets are tweeted.



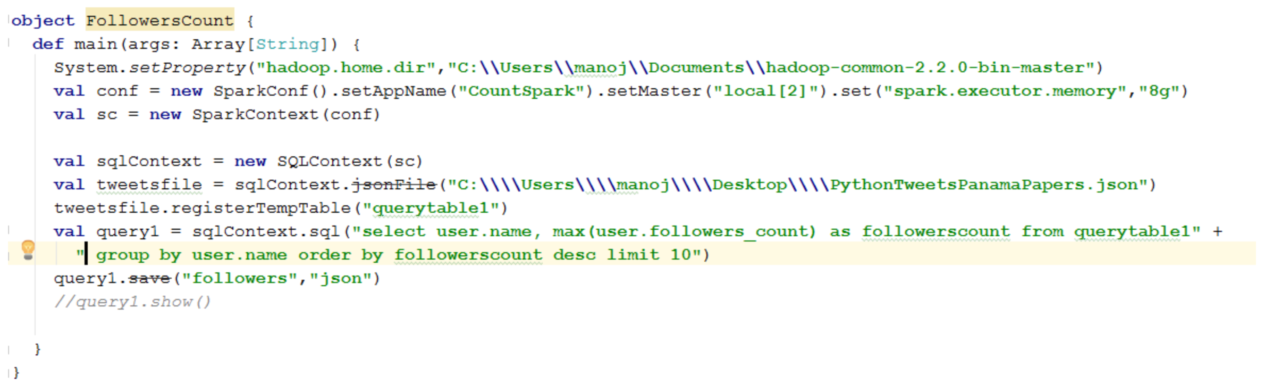
```scala
import ...

/**
  * Created by manoj on 4/7/2016.
  */
object TweetsEachHour {
  def main(args: Array[String]) {
    System.setProperty("hadoop.home.dir", "C:\\Users\\manoj\\Documents\\hadoop-common-2.2.0-bin-master")
    val config = new SparkConf().setAppName("CountSpark").setMaster("local[2]").set("spark.executor.memory", "8g")
    val sparkcontext = new SparkContext(config)
    val sqlContext = new SQLContext(sparkcontext)
    val tweetsfile = sqlContext.~~jsonFile~~("C:\\\\Users\\\\manoj\\\\Desktop\\\\PythonTweetsPanamaPapers.json")
    tweetsfile.registerTempTable("tweettable")
    val hour_query = sqlContext.sql("select SUBSTR(created_at,12,2) as Hour,count(id) as TweetsEachHour from tweettable group by SUBSTR(created_at,12,2) " +
      "order by TweetsEachHour desc limit 5")
    hour_query.show()

  }
}
```



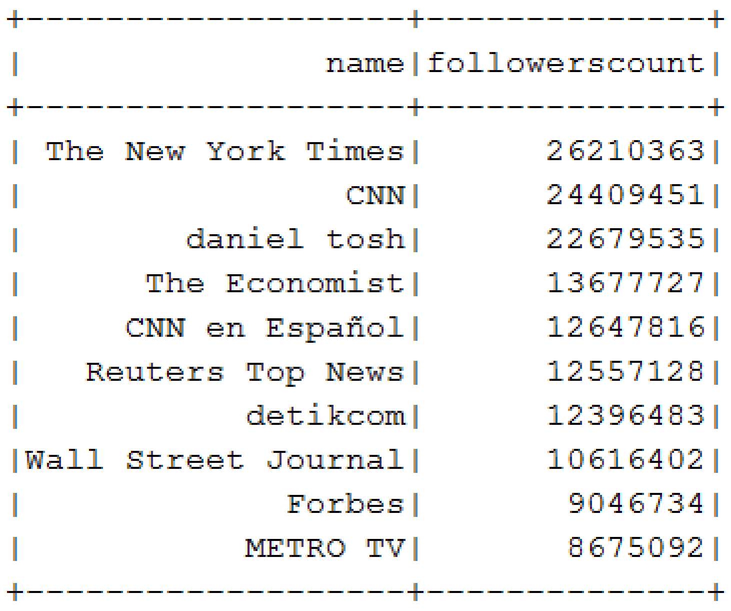
```
+----+--------------+
|Hour|TweetsEachHour|
+----+--------------+
|  02|         32140|
|  03|         30456|
|  04|         28374|
|  05|          7596|
|  01|          4359|
+----+--------------+
```

8th **Query** (Followers count)

From the above query we have analyzed the highest count of followers for the users.



```
object FollowersCount {
  def main(args: Array[String]) {
    System.setProperty("hadoop.home.dir","C:\\Users\\manoj\\Documents\\hadoop-common-2.2.0-bin-master")
    val conf = new SparkConf().setAppName("CountSpark").setMaster("local[2]").set("spark.executor.memory","8g")
    val sc = new SparkContext(conf)

    val sqlContext = new SQLContext(sc)
    val tweetsfile = sqlContext.~~jsonFile~~("C:\\\\Users\\\\manoj\\\\Desktop\\\\PythonTweetsPanamaPapers.json")
    tweetsfile.registerTempTable("querytable1")
    val query1 = sqlContext.sql("select user.name, max(user.followers_count) as followerscount from querytable1" +
      " group by user.name order by followerscount desc limit 10")
    query1.~~save~~("followers","json")
    //query1.show()

  }
}
```



```
+-------------------+--------------+
|               name|followerscount|
+-------------------+--------------+
| The New York Times|      26210363|
|                CNN|      24409451|
|        daniel tosh|      22679535|
|       The Economist|      13677727|
|      CNN en Español|      12647816|
|   Reuters Top News|      12557128|
|           detikcom|      12396483|
|Wall Street Journal|      10616402|
|             Forbes|       9046734|
|           METRO TV|       8675092|
+-------------------+--------------+
```

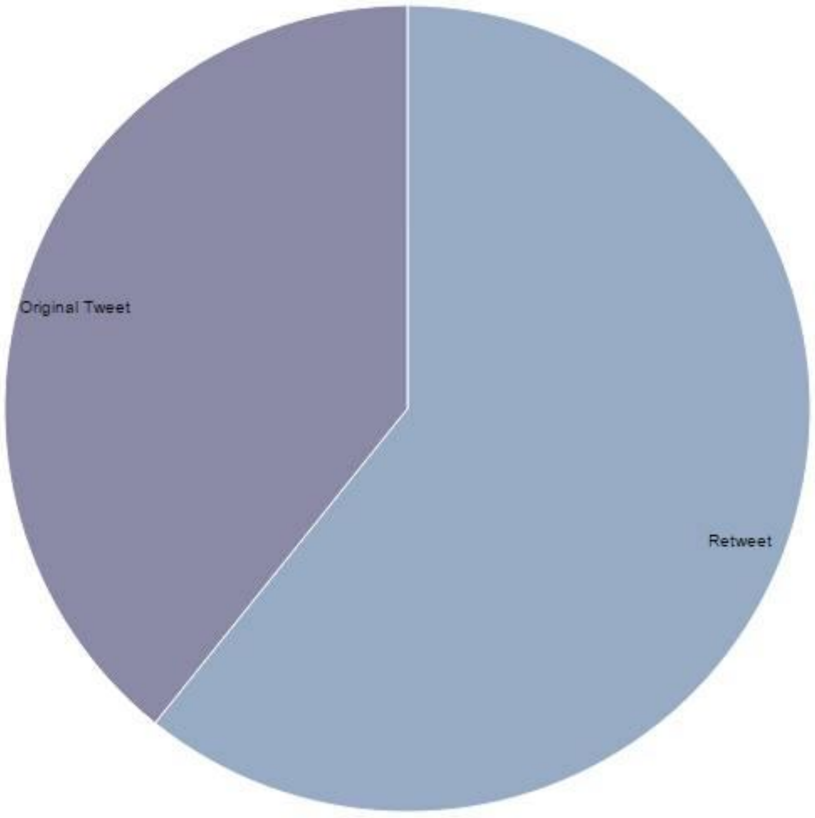
Data Visualization:

In this phase, we have visualized the data in the form of graphs to draw some insights from the analyzed json data.

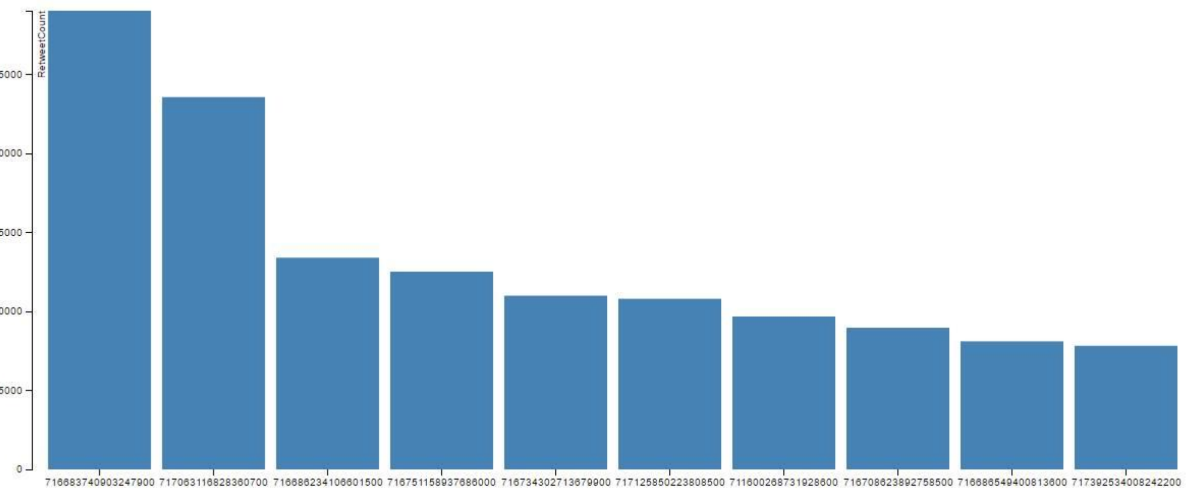
For this, we have made use of the High Charts API available in java.

We have created a UI Page where the data can be visualized to draw some insights.After looking at the UI page we can draw some conclusions on our 70K tweets very quickly.

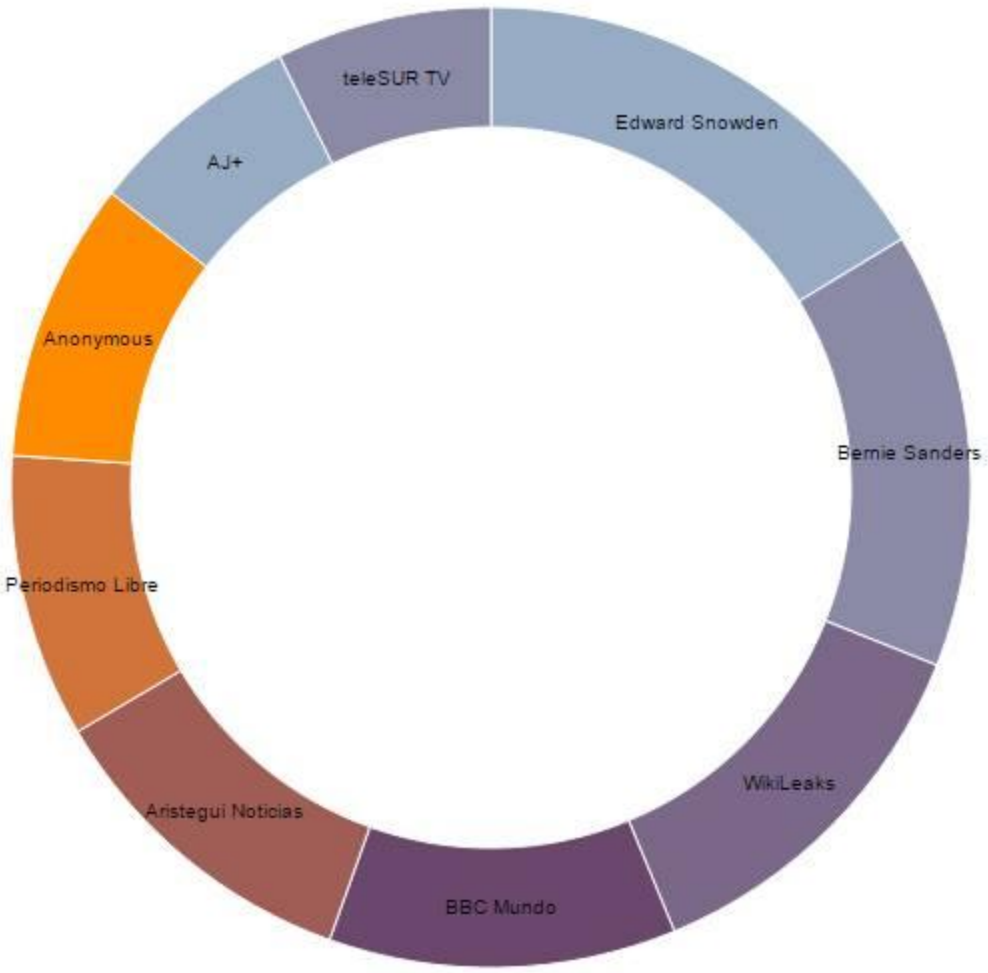
Visualization 1:(% of retweets vs original tweets)



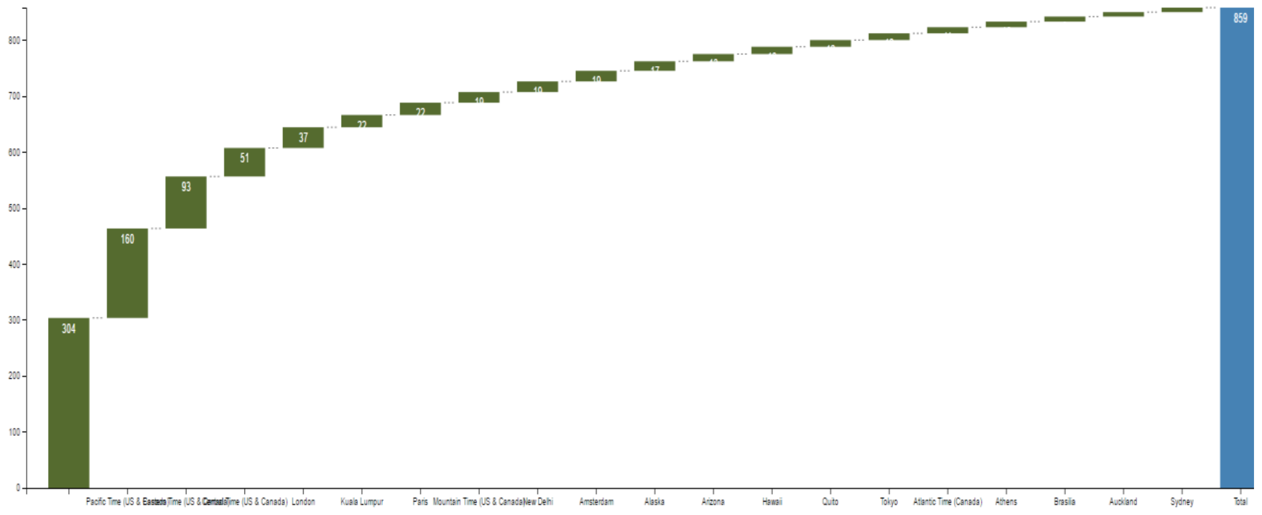
Visualization 2: (No .of retweets made by particular user tweet)



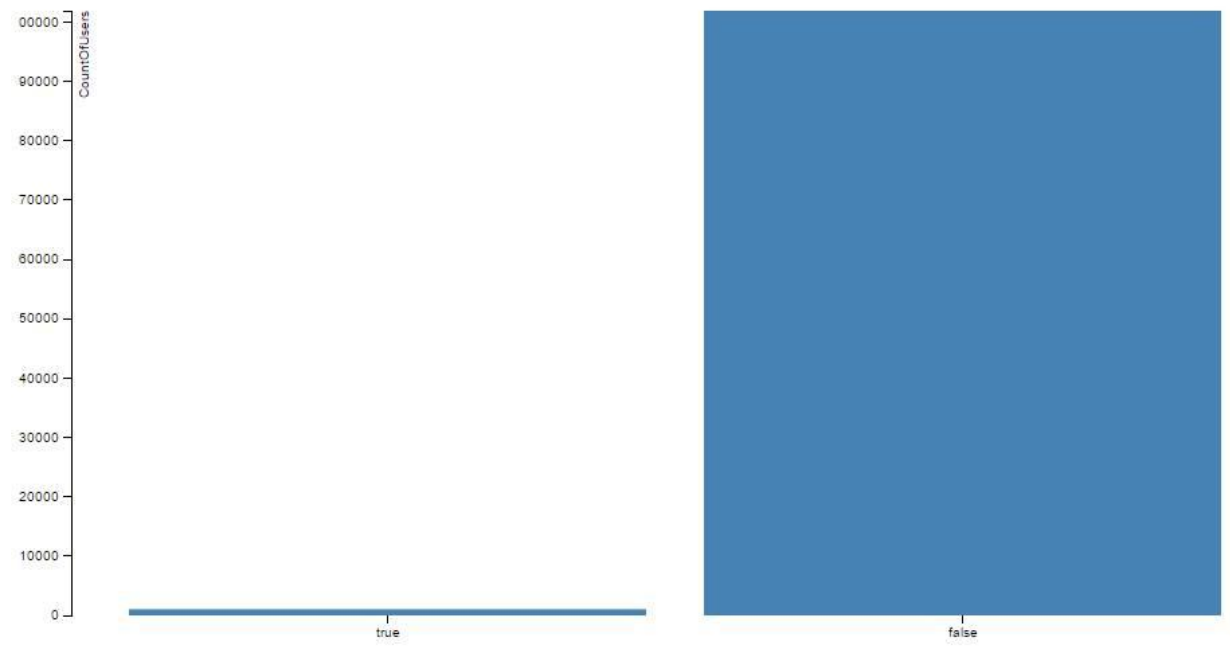
Visualization 3:Top Mentions



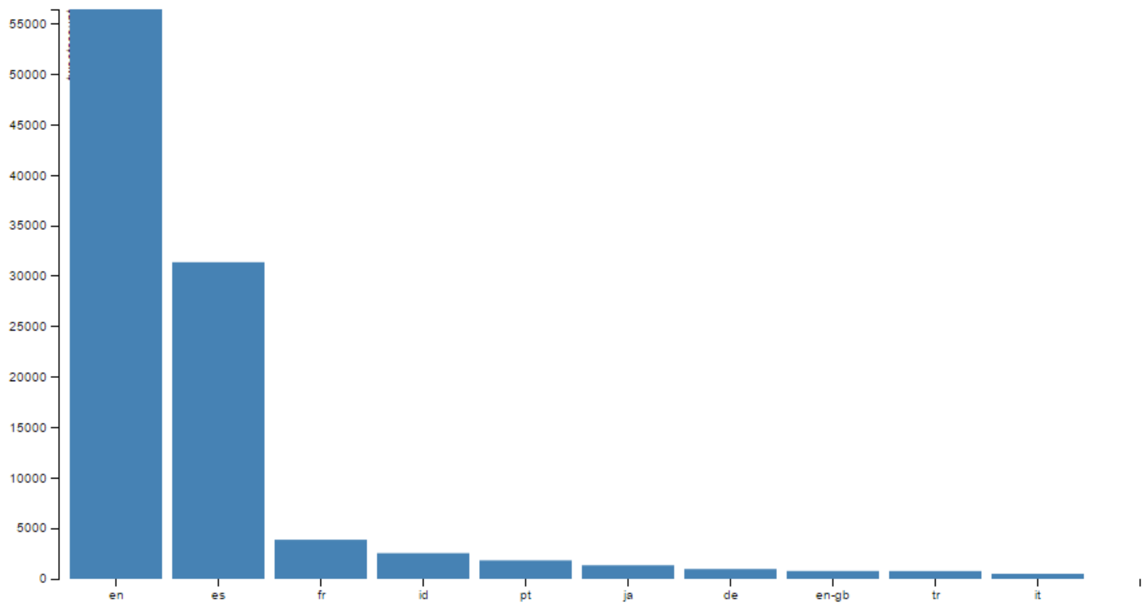
Visualization 4: (no. of times a user is mentioned based on time zone)



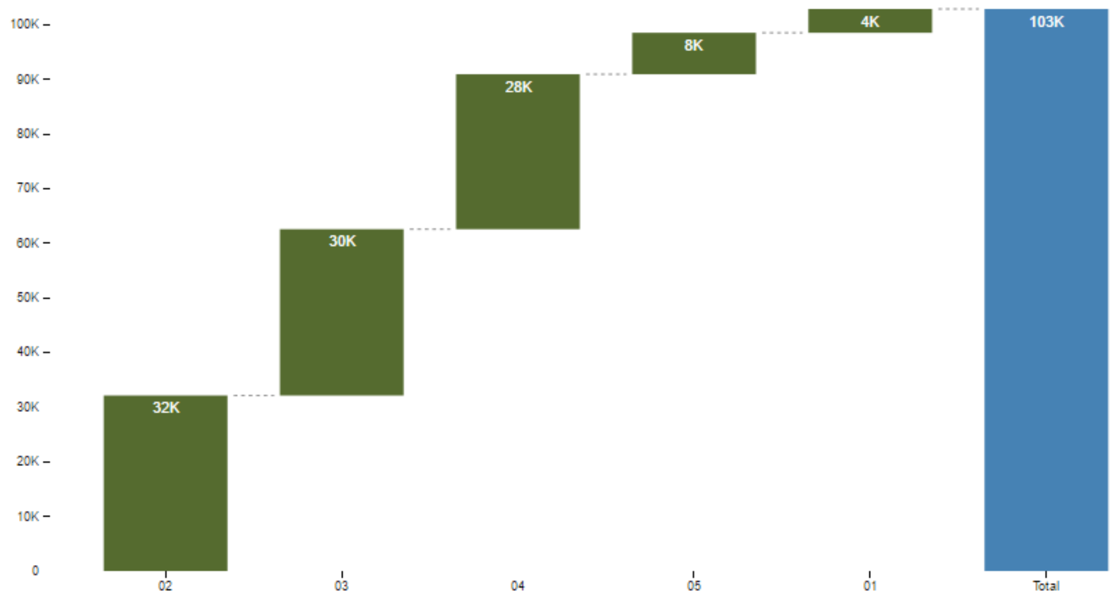
Visualization 5: (verified user vs non verified user)



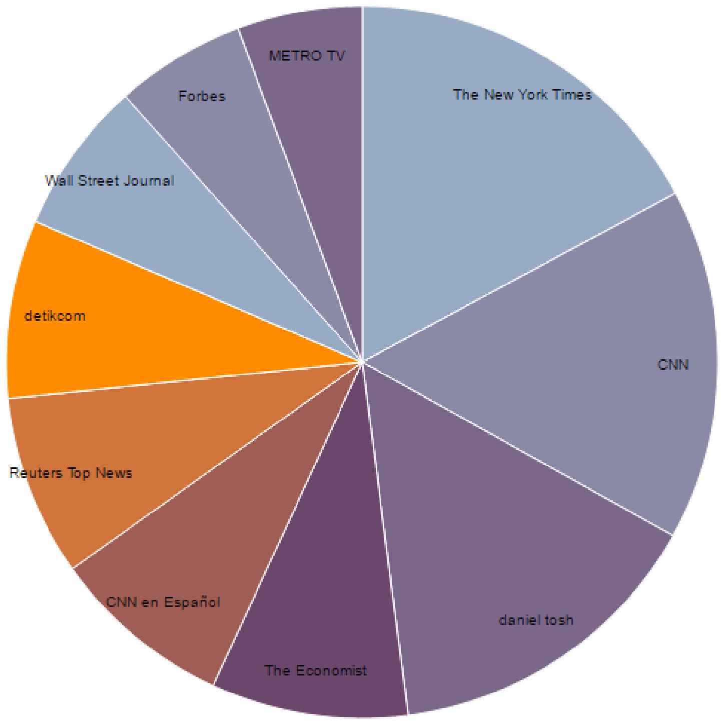
Visualization 6:



Visualization 7:



Visualization 8:



References:

- https://d3js.org/
- https://developers.google.com/chart/
- http://marcobonzanini.com/2015/03/02/mining-twitter-data-with-python-part-1/