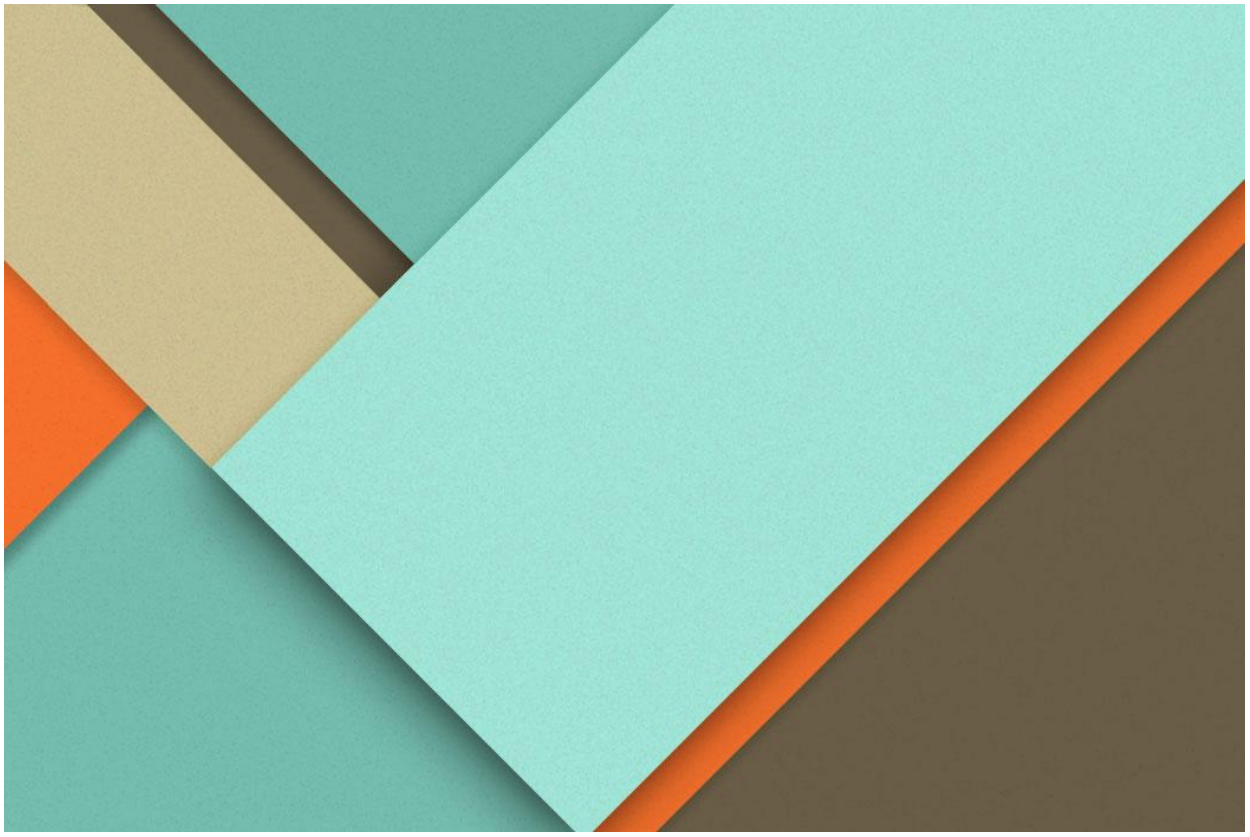


Principles of Big Data Management

Project Report

Arun George Zachariah

University of Missouri – Kansas City

Table of Contents

Goal

The main goal of the project is design, develop and execute a web application that would visualize interesting analytic queries executed on tweets for five famous brands namely Adidas, Nike, Puma, Skechers and Reebok.

Introduction

Twitter is an online news and social networking site where people communicate in short messages called tweets and is one of the most popular social media platforms in the world. This simple, but effective sharing channel has:

- Insights from 83% of the world's leaders.
- 330 million monthly users.
- 3 billion account holders.

The fast-paced nature of this platform means that it's a great way for brands to start building a stronger online presence and learning how to track the right metrics could help any brand make insightful decisions about their future marketing campaign.

Through this project, we decide to do the exact same analysis for five major apparel brands namely Adidas, Nike, Puma, Skechers and Reebok.

Design Architecture

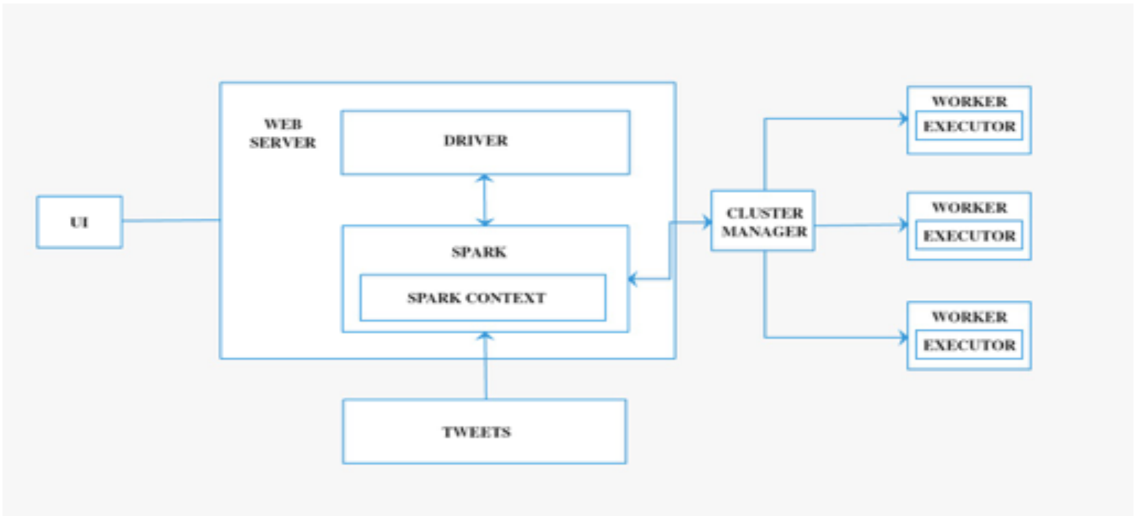


Fig1 : Architecture Diagram.

Tools and Libraries Used

Programming Languages: Java, Scala, JavaScript

Cluster-computing framework: Spark

Web Services: RESTEasy

Web Server: Jetty

Build Tool: Maven

Testing Framework: jUnit

Javascript Libraries: CanvasJS, Google Charts

Containerization Platform: Docker

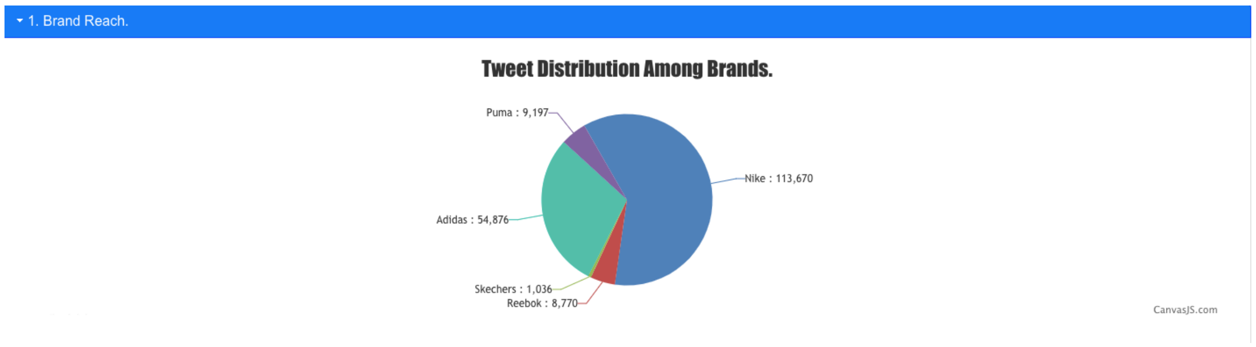
Sentiment Analysis: text-processing.com

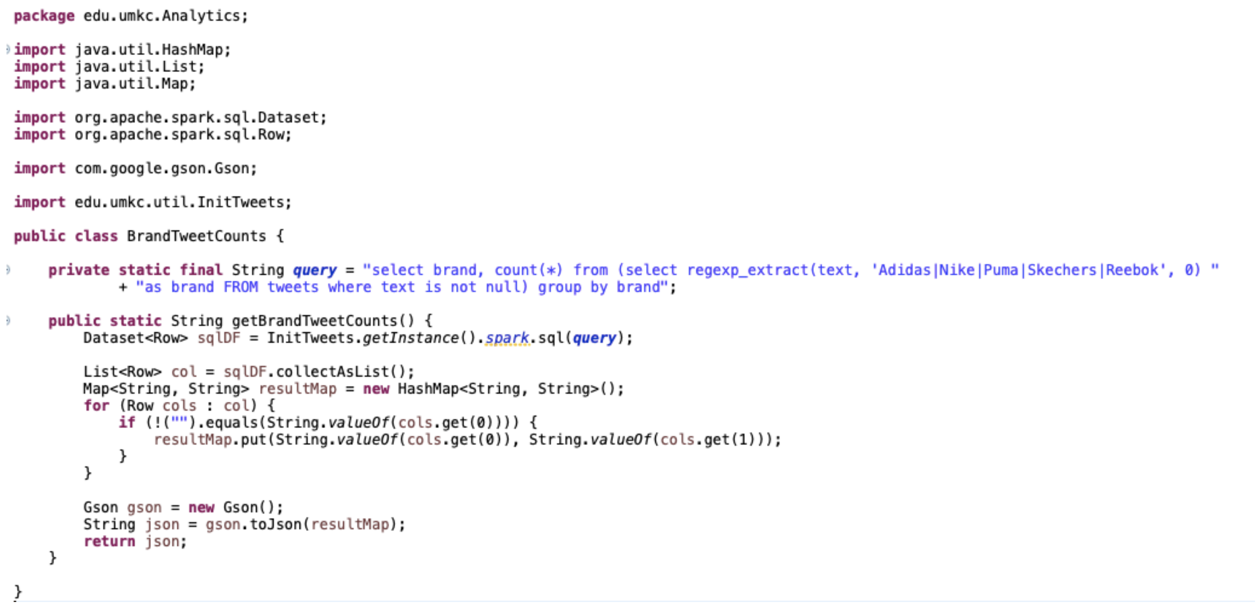
Queries

I. Brand Reach

```
select brand, count(*) from (select regexp_extract(text,
'Adidas|Nike|Puma|Skechers|Reebok', 0) as brand FROM tweets
where text is not null) group by brand
```

With the above query, we try to understand the reach of a particular brand among Twitterati's, by analyzing the tweet text for brand name keywords and aggregating this count for a particular brand.



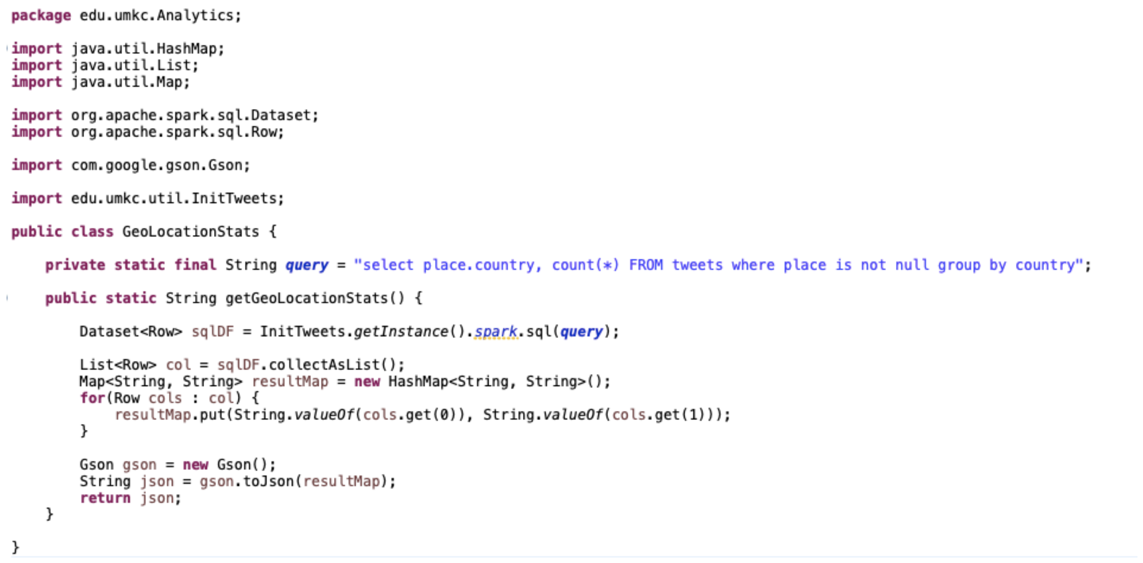


```java
package edu.umkc.Analytics;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;

import com.google.gson.Gson;

import edu.umkc.util.InitTweets;

public class BrandTweetCounts {

    private static final String query = "select brand, count(*) from (select regexp_extract(text, 'Adidas|Nike|Puma|Skechers|Reebok', 0) "
            + "as brand FROM tweets where text is not null) group by brand";

    public static String getBrandTweetCounts() {
        Dataset<Row> sqlDF = InitTweets.getInstance().spark.sql(query);

        List<Row> col = sqlDF.collectAsList();
        Map<String, String> resultMap = new HashMap<String, String>();
        for (Row cols : col) {
            if (!("").equals(String.valueOf(cols.get(0)))) {
                resultMap.put(String.valueOf(cols.get(0)), String.valueOf(cols.get(1)));
            }
        }

        Gson gson = new Gson();
        String json = gson.toJson(resultMap);
        return json;
    }

}
```

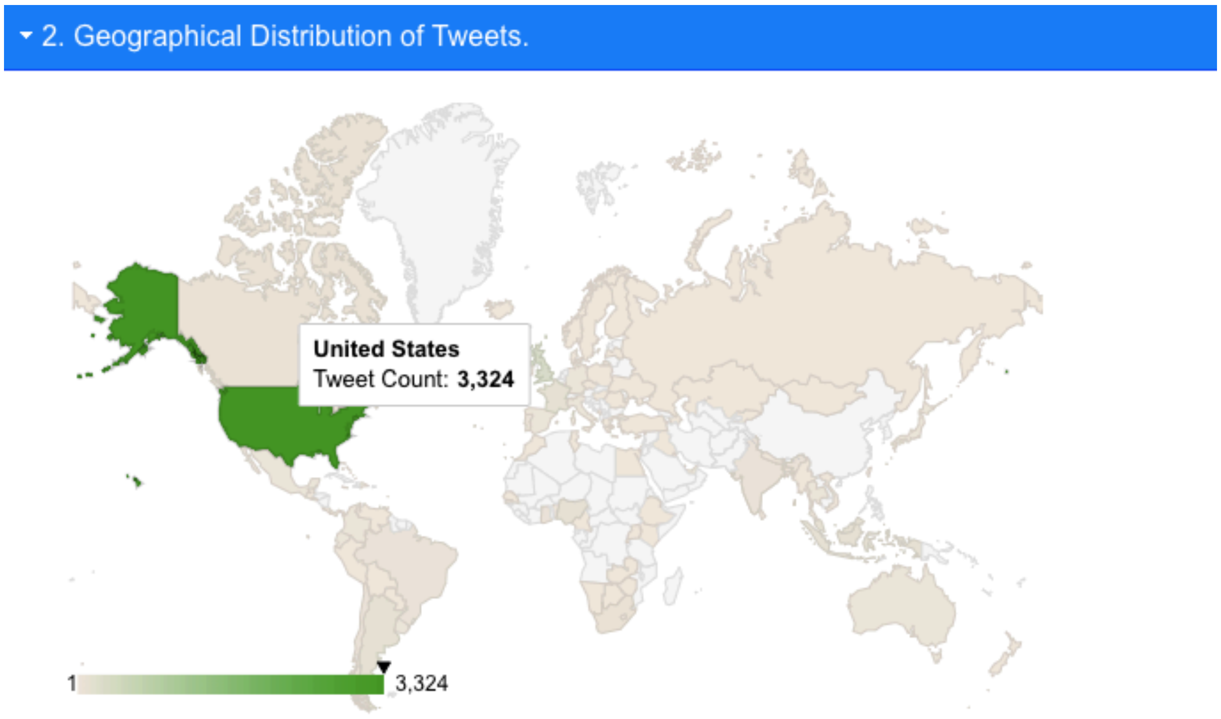
II. Geographical Distribution of Tweets

```
select place.country, count(*) FROM tweets where place is
not null group by country
```

With the above query, we try to visualize the Geographical Distribution of the origin of tweets, by grouping the tweets based on the country of origin.



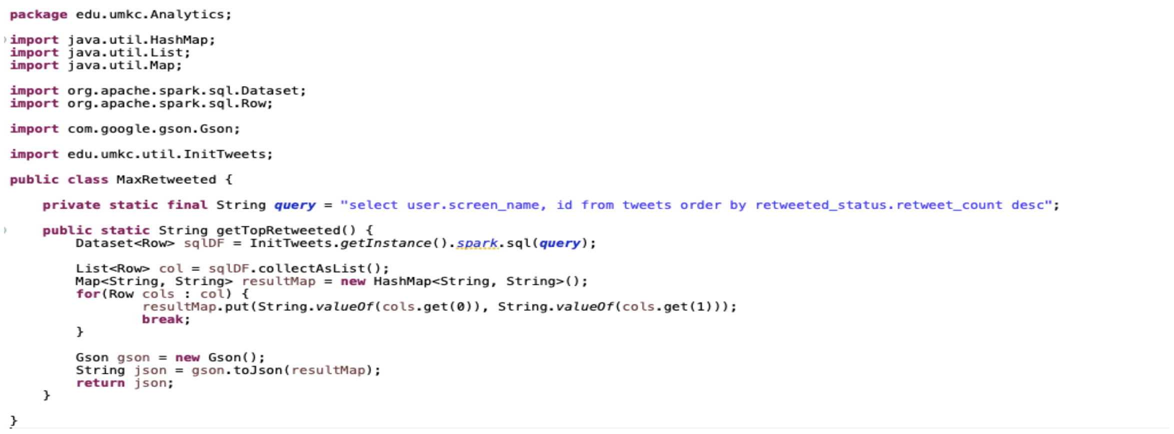
```java
package edu.umkc.Analytics;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;

import com.google.gson.Gson;

import edu.umkc.util.InitTweets;

public class GeoLocationStats {

    private static final String query = "select place.country, count(*) FROM tweets where place is not null group by country";

    public static String getGeoLocationStats() {

        Dataset<Row> sqlDF = InitTweets.getInstance().spark.sql(query);

        List<Row> col = sqlDF.collectAsList();
        Map<String, String> resultMap = new HashMap<String, String>();
        for(Row cols : col) {
            resultMap.put(String.valueOf(cols.get(0)), String.valueOf(cols.get(1)));
        }

        Gson gson = new Gson();
        String json = gson.toJson(resultMap);
        return json;
    }

}
```



III. Most Retweeted Tweet

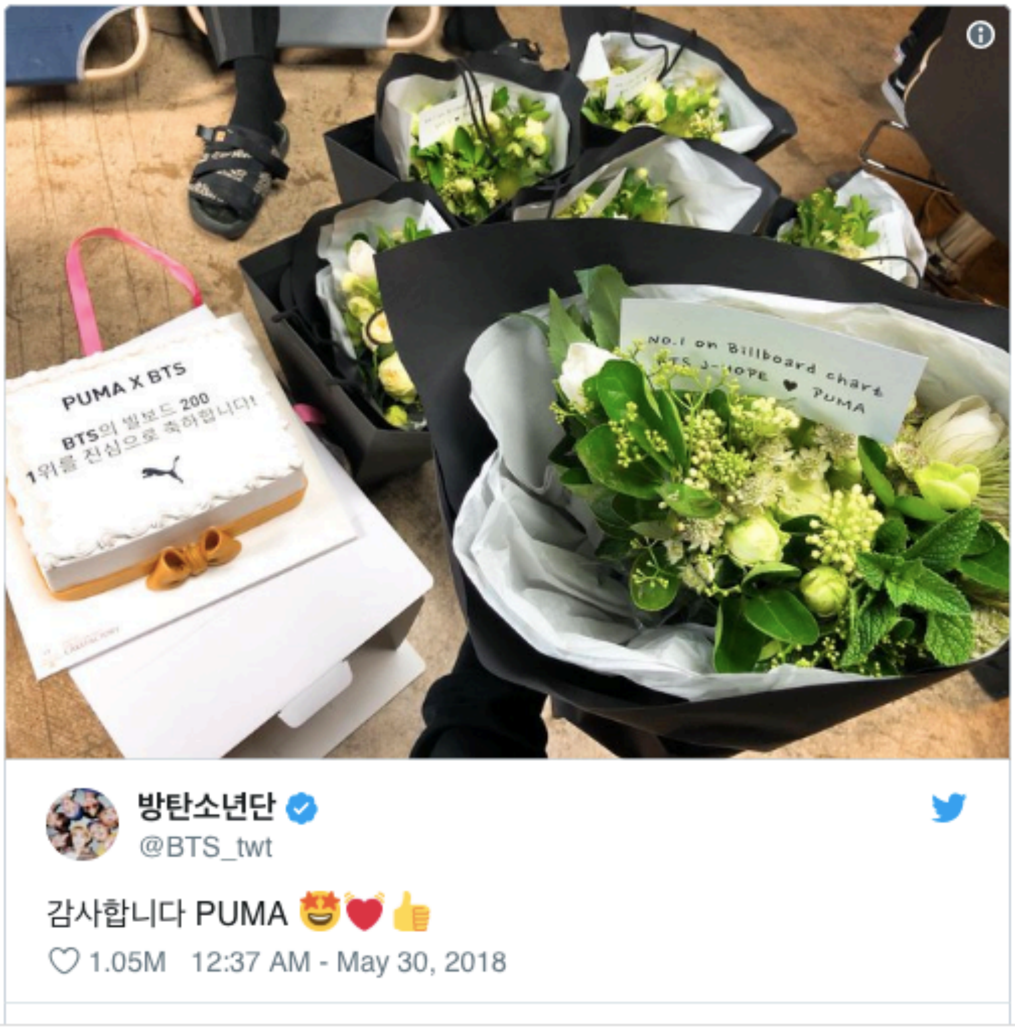
```
select user.screen_name, id from tweets order by
retweeted_status.retweet_count desc
```

With the above query, we fetch the username and the id of the most retweeted tweet. The username and id fetched are used to construct the URL for the tweet that is then displayed in the user interface.



```
package edu.umkc.Analytics;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;

import com.google.gson.Gson;

import edu.umkc.util.InitTweets;

public class MaxRetweeted {

    private static final String query = "select user.screen_name, id from tweets order by retweeted_status.retweet_count desc";

    public static String getTopRetweeted() {
        Dataset<Row> sqlDF = InitTweets.getInstance().spark.sql(query);

        List<Row> col = sqlDF.collectAsList();
        Map<String, String> resultMap = new HashMap<String, String>();
        for(Row cols : col) {
                resultMap.put(String.valueOf(cols.get(0)), String.valueOf(cols.get(1)));
                break;
        }

        Gson gson = new Gson();
        String json = gson.toJson(resultMap);
        return json;
    }

}
```

▾ 3. Most Retweeted Tweet.



방탄소년단
@BTS_twt

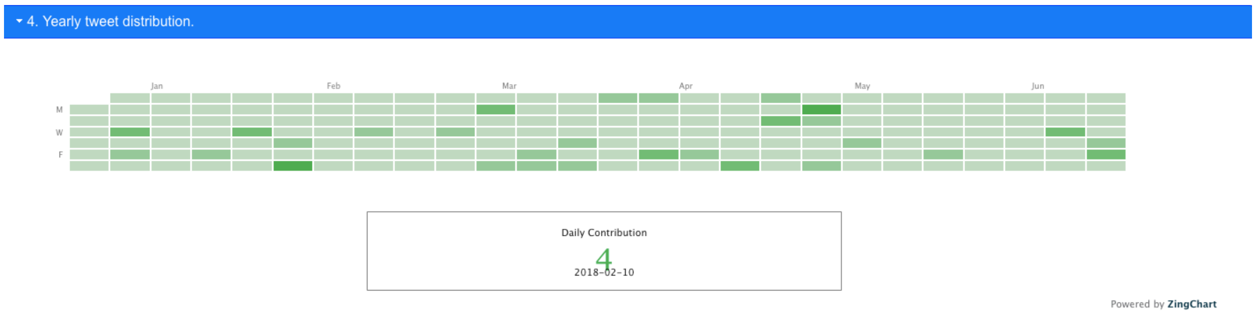
감사합니다 PUMA 🤩❤️👍

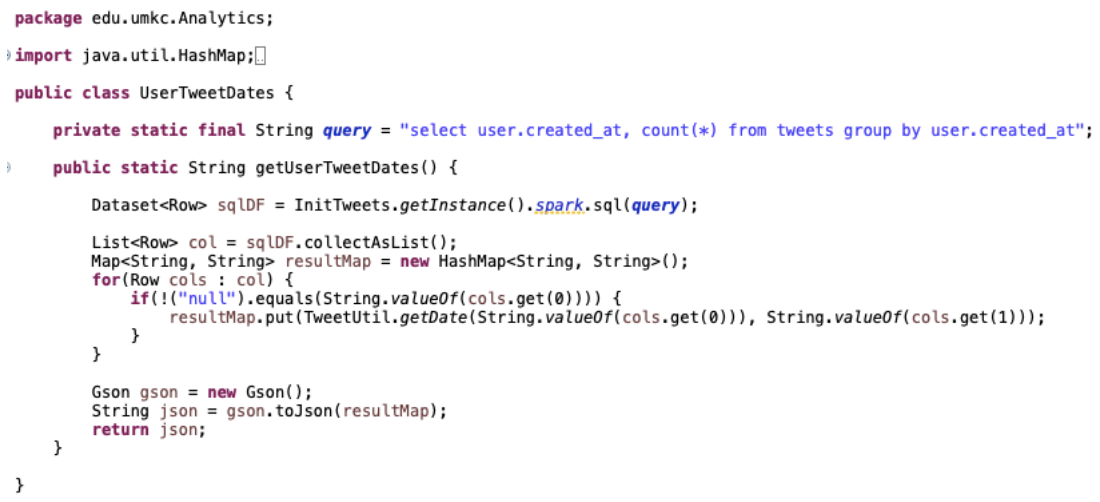
♡ 1.05M 12:37 AM - May 30, 2018

IV. Yearly Tweet Distribution

```
select user.created_at, count(*) from tweets group by
user.created_at
```

With the above query, we try to analyze the number of tweets generated over time. This is then visualized using a calendar, similar to how Github visualises the user commits over time.



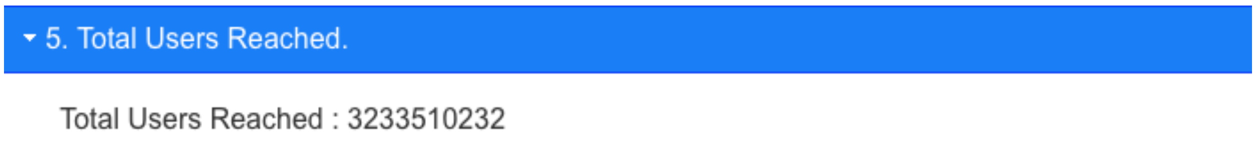


```
package edu.umkc.Analytics;

import java.util.HashMap;

public class UserTweetDates {

    private static final String query = "select user.created_at, count(*) from tweets group by user.created_at";

    public static String getUserTweetDates() {

        Dataset<Row> sqlDF = InitTweets.getInstance().spark.sql(query);

        List<Row> col = sqlDF.collectAsList();
        Map<String, String> resultMap = new HashMap<String, String>();
        for(Row cols : col) {
            if(!("null").equals(String.valueOf(cols.get(0)))) {
                resultMap.put(TweetUtil.getDate(String.valueOf(cols.get(0))), String.valueOf(cols.get(1)));
            }
        }

        Gson gson = new Gson();
        String json = gson.toJson(resultMap);
        return json;
    }

}
```

V. Total Users Reached

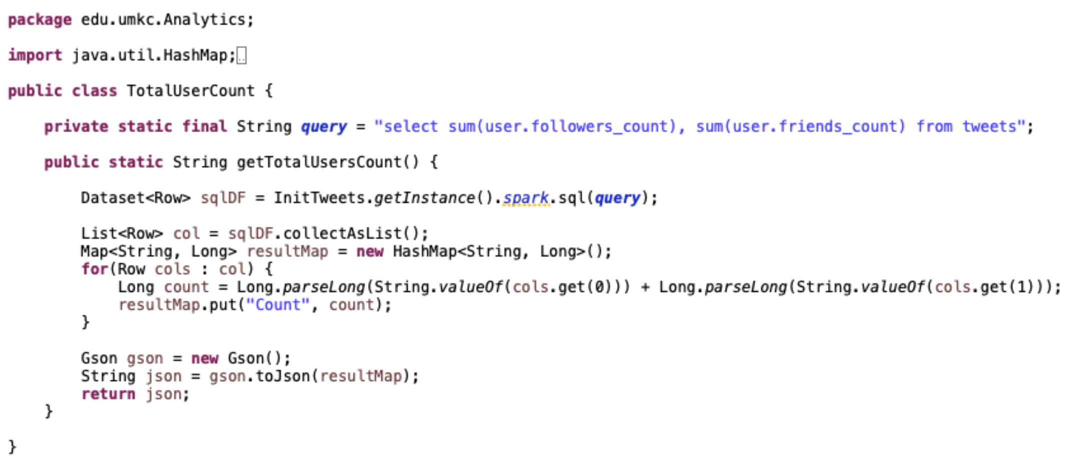
```
select sum(user.followers_count), sum(user.friends_count)
from tweets
```

With the above query, we try to find the total reach of the top 10 tweets, buy finding the sum of the followers and friends count.



▾ 5. Total Users Reached.

Total Users Reached : 3233510232

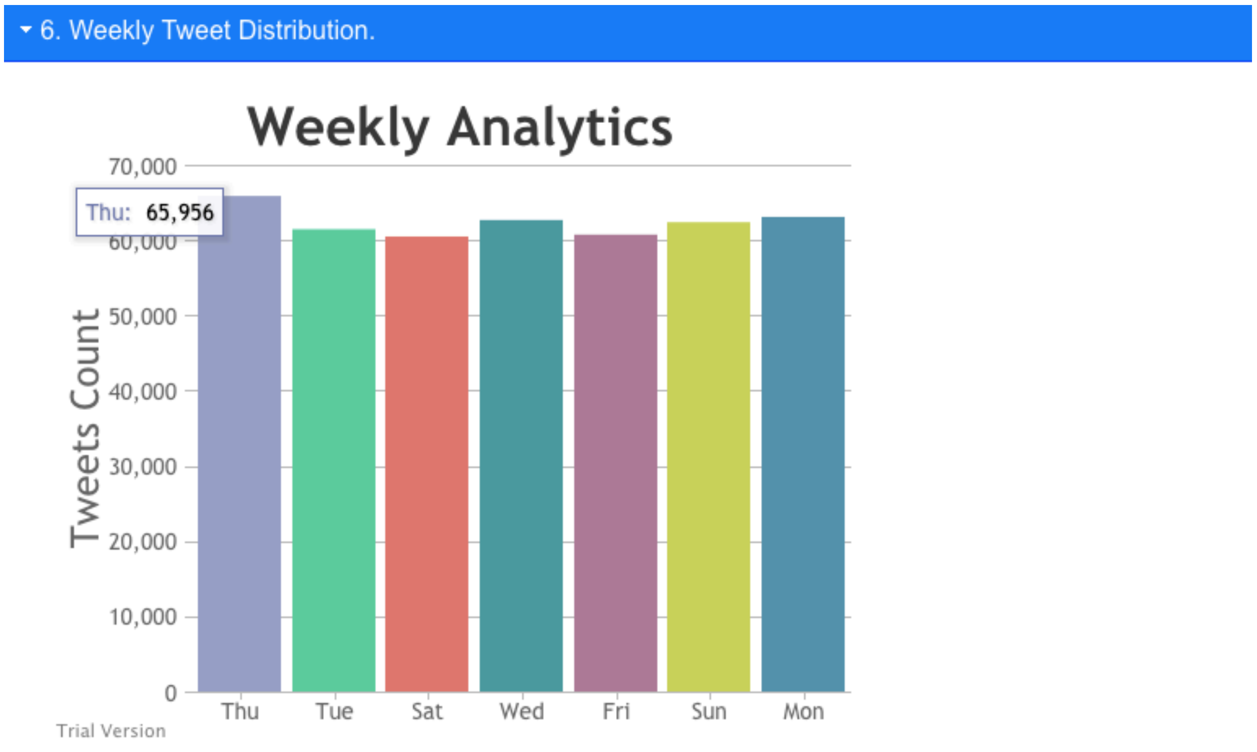


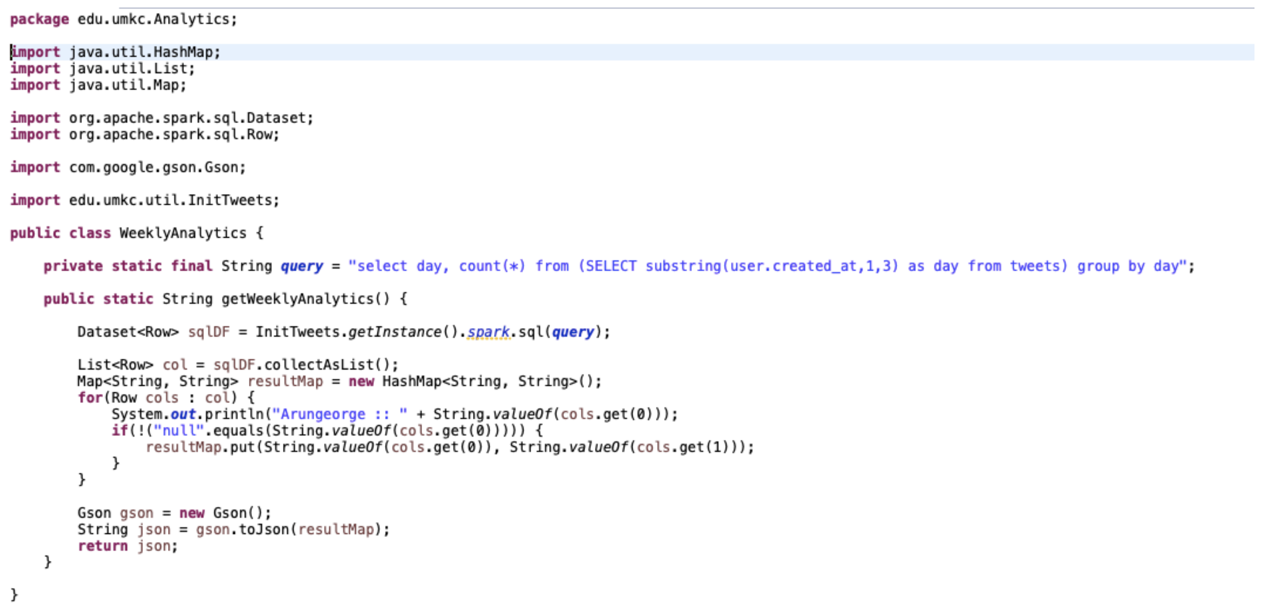
```
package edu.umkc.Analytics;

import java.util.HashMap;

public class TotalUserCount {

    private static final String query = "select sum(user.followers_count), sum(user.friends_count) from tweets";

    public static String getTotalUsersCount() {

        Dataset<Row> sqlDF = InitTweets.getInstance().spark.sql(query);

        List<Row> col = sqlDF.collectAsList();
        Map<String, Long> resultMap = new HashMap<String, Long>();
        for(Row cols : col) {
            Long count = Long.parseLong(String.valueOf(cols.get(0))) + Long.parseLong(String.valueOf(cols.get(1)));
            resultMap.put("Count", count);
        }

        Gson gson = new Gson();
        String json = gson.toJson(resultMap);
        return json;
    }
}
```

VI. Weekly Tweet Distribution

```
select day, count(*) from (select
substring(user.created_at,1,3) as day from tweets) group by
day
```

With the above query, we try to analyze the daily distribution of tweets that have originated for the various brands.



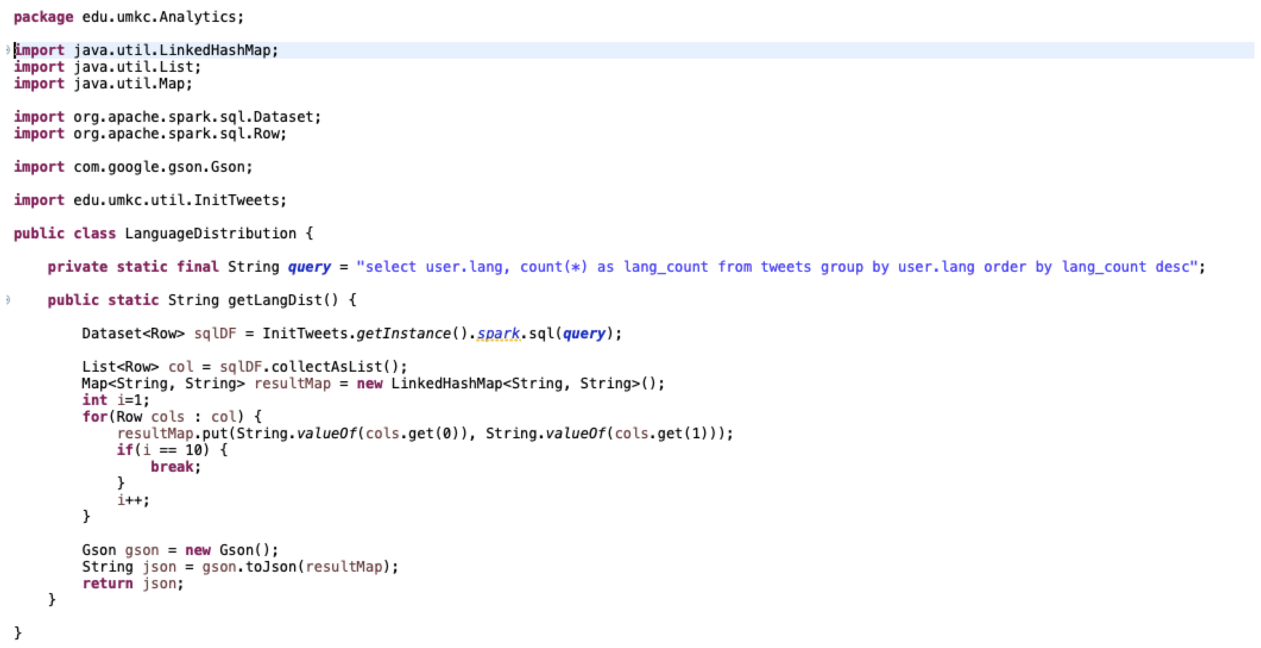


```java
package edu.umkc.Analytics;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;

import com.google.gson.Gson;

import edu.umkc.util.InitTweets;

public class WeeklyAnalytics {

    private static final String query = "select day, count(*) from (SELECT substring(user.created_at,1,3) as day from tweets) group by day";

    public static String getWeeklyAnalytics() {

        Dataset<Row> sqlDF = InitTweets.getInstance().spark.sql(query);

        List<Row> col = sqlDF.collectAsList();
        Map<String, String> resultMap = new HashMap<String, String>();
        for(Row cols : col) {
            System.out.println("Arungeorge :: " + String.valueOf(cols.get(0)));
            if(!("null".equals(String.valueOf(cols.get(0))))) {
                resultMap.put(String.valueOf(cols.get(0)), String.valueOf(cols.get(1)));
            }
        }

        Gson gson = new Gson();
        String json = gson.toJson(resultMap);
        return json;
    }

}
```

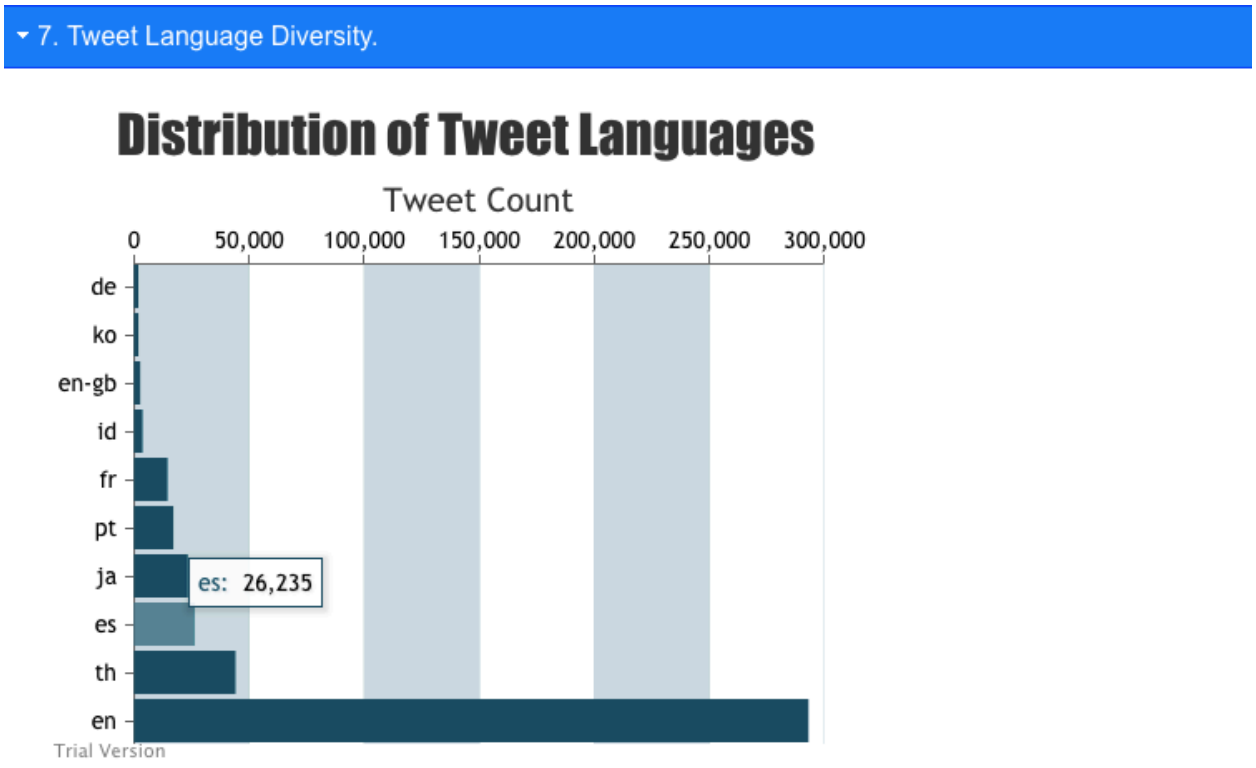
VII. Tweet Language Diversity

```
select user.lang, count(*) as lang_count from tweets group
by user.lang order by lang_count desc
```

With the above query, we try to analyze the language diversity of the tweets by grouping the tweets based on their language and getting the count of the number of tweets for each language.



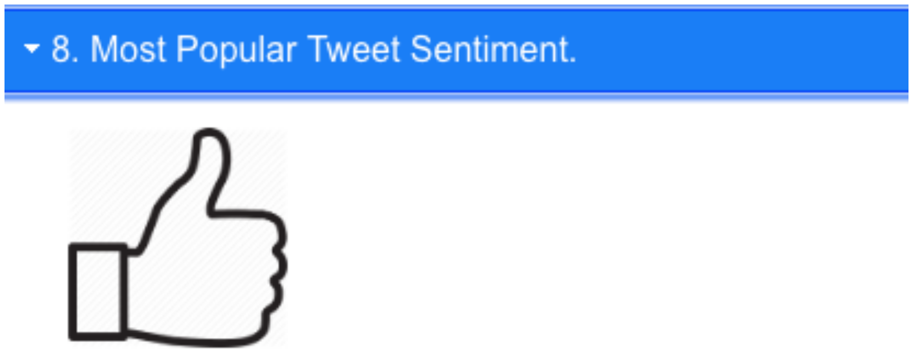
```java
package edu.umkc.Analytics;

import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;

import com.google.gson.Gson;

import edu.umkc.util.InitTweets;

public class LanguageDistribution {

    private static final String query = "select user.lang, count(*) as lang_count from tweets group by user.lang order by lang_count desc";

    public static String getLangDist() {

        Dataset<Row> sqlDF = InitTweets.getInstance().spark.sql(query);

        List<Row> col = sqlDF.collectAsList();
        Map<String, String> resultMap = new LinkedHashMap<String, String>();
        int i=1;
        for(Row cols : col) {
            resultMap.put(String.valueOf(cols.get(0)), String.valueOf(cols.get(1)));
            if(i == 10) {
                break;
            }
            i++;
        }

        Gson gson = new Gson();
        String json = gson.toJson(resultMap);
        return json;
    }

}
```

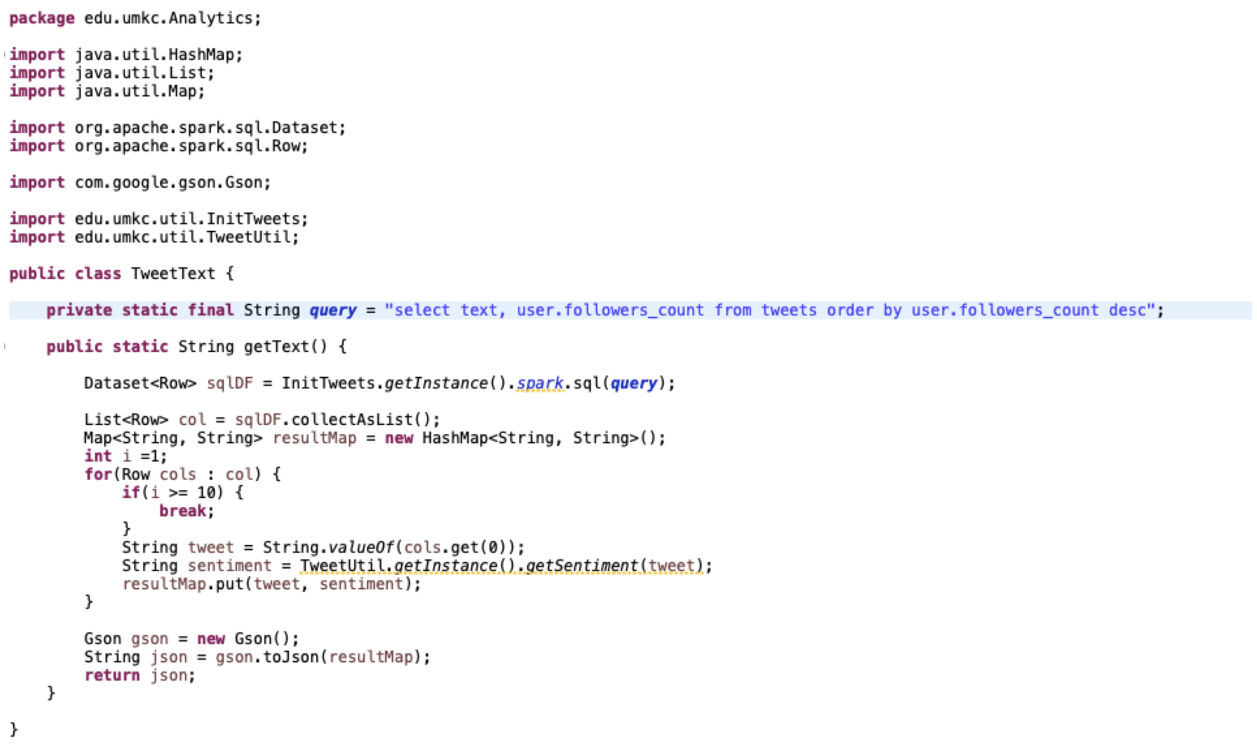


VIII. Most Popular Tweet Sentiment

```
select text, user.followers_count from tweets order by
user.followers_count desc
```

With the above query, we try to get the tweet of the most followed Twitterati's and do a simple sentiment analysis using the sentiment API provided by text-processing.com. The output is either positive or negative response, when is then visualized in the user interface.



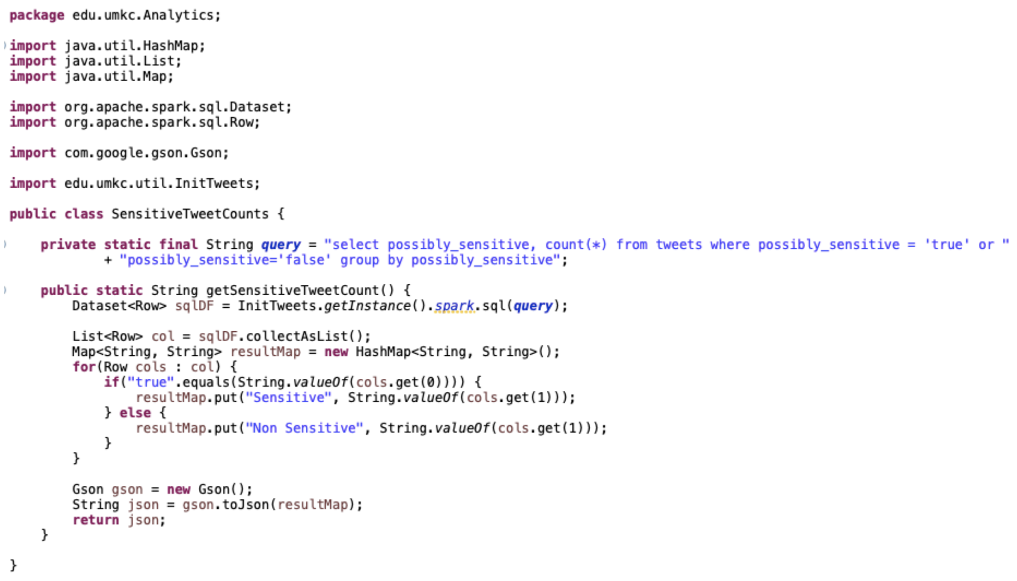


```java
package edu.umkc.Analytics;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;

import com.google.gson.Gson;

import edu.umkc.util.InitTweets;
import edu.umkc.util.TweetUtil;

public class TweetText {

    private static final String query = "select text, user.followers_count from tweets order by user.followers_count desc";

    public static String getText() {

        Dataset<Row> sqlDF = InitTweets.getInstance().spark.sql(query);

        List<Row> col = sqlDF.collectAsList();
        Map<String, String> resultMap = new HashMap<String, String>();
        int i =1;
        for(Row cols : col) {
            if(i >= 10) {
                break;
            }
            String tweet = String.valueOf(cols.get(0));
            String sentiment = TweetUtil.getInstance().getSentiment(tweet);
            resultMap.put(tweet, sentiment);
        }

        Gson gson = new Gson();
        String json = gson.toJson(resultMap);
        return json;
    }

}
```

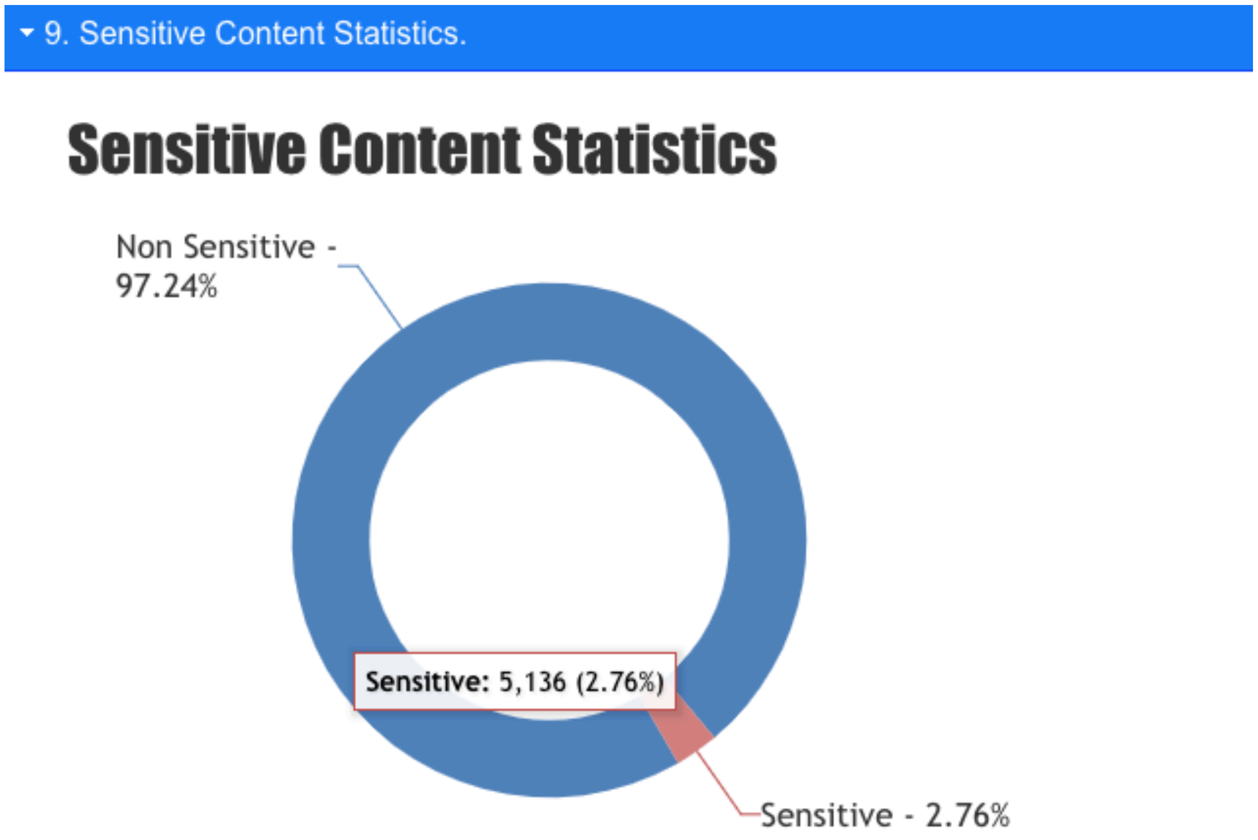
IX. Sensitive Content Statistics

```
select possibly_sensitive, count(*) from tweets where
possibly_sensitive = 'true' or possibly_sensitive='false'
group by possibly_sensitive
```

With the above query, we try to find out the count of tweets that had a sensitive content using the Boolean field possibly_sensitive.



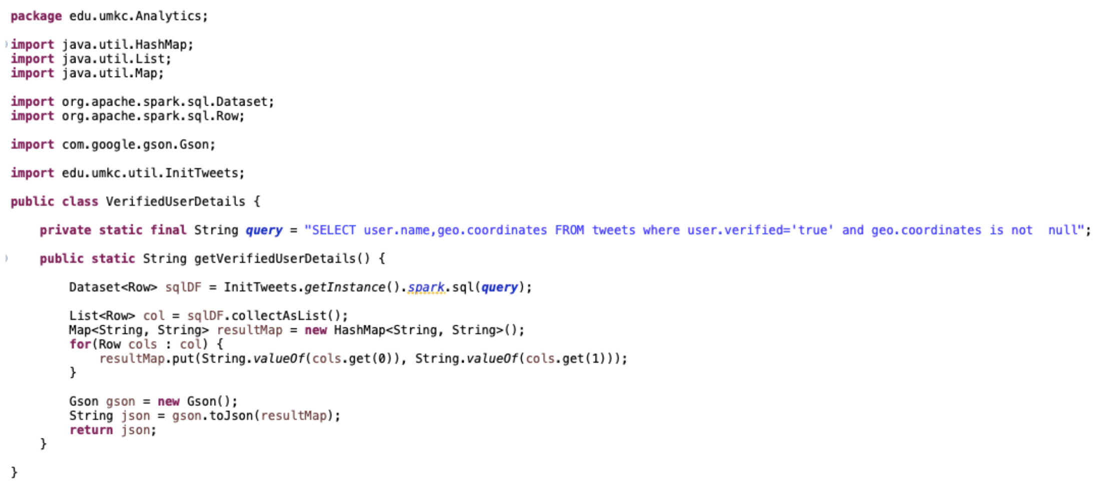
```java
package edu.umkc.Analytics;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;

import com.google.gson.Gson;

import edu.umkc.util.InitTweets;

public class SensitiveTweetCounts {

    private static final String query = "select possibly_sensitive, count(*) from tweets where possibly_sensitive = 'true' or "
            + "possibly_sensitive='false' group by possibly_sensitive";

    public static String getSensitiveTweetCount() {
        Dataset<Row> sqlDF = InitTweets.getInstance().spark.sql(query);

        List<Row> col = sqlDF.collectAsList();
        Map<String, String> resultMap = new HashMap<String, String>();
        for(Row cols : col) {
            if("true".equals(String.valueOf(cols.get(0)))) {
                resultMap.put("Sensitive", String.valueOf(cols.get(1)));
            } else {
                resultMap.put("Non Sensitive", String.valueOf(cols.get(1)));
            }
        }

        Gson gson = new Gson();
        String json = gson.toJson(resultMap);
        return json;
    }

}
```



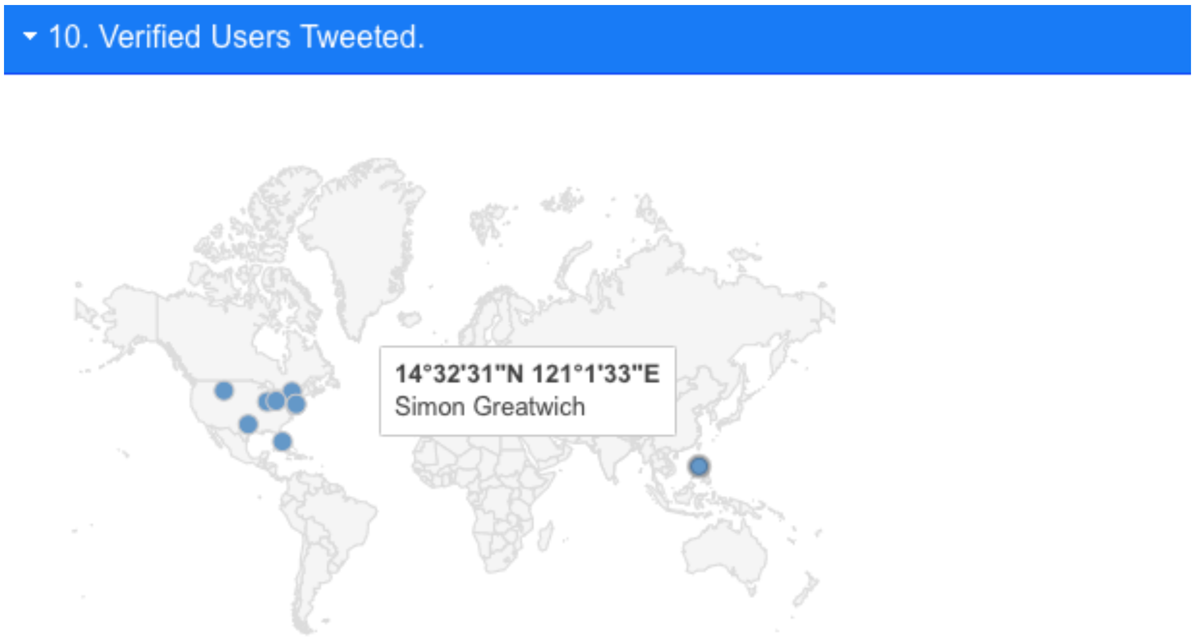
X. Verified Users Tweeted

```
select user.name,geo.coordinates from tweets where
user.verified='true' and geo.coordinates is not  null
```

With the above query, we try to analyze the verified users that have tweeted and the location.



```
package edu.umkc.Analytics;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;

import com.google.gson.Gson;

import edu.umkc.util.InitTweets;

public class VerifiedUserDetails {

    private static final String query = "SELECT user.name,geo.coordinates FROM tweets where user.verified='true' and geo.coordinates is not  null";

    public static String getVerifiedUserDetails() {

        Dataset<Row> sqlDF = InitTweets.getInstance().spark.sql(query);

        List<Row> col = sqlDF.collectAsList();
        Map<String, String> resultMap = new HashMap<String, String>();
        for(Row cols : col) {
            resultMap.put(String.valueOf(cols.get(0)), String.valueOf(cols.get(1)));
        }

        Gson gson = new Gson();
        String json = gson.toJson(resultMap);
        return json;
    }

}
```



Testing

Dataset is a new interface added in Spark 1.6 that provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.). The Dataset API is available in Scala and Java. As the application is built with Java, Junit was used as the testing tool. A subset of data was loaded to spark and unit test cases were written for the queries being implemented. The execution of these test cases were integrated with maven and were triggered at the time of execution. If any of the test cases failed, the build would fail, preventing further deployment.

Learning Outcome

- Understanding the importance of processing huge volumes of data in Spark.
- Understanding the architecture of Spark and the various modules built over Spark.
- Execution of various queries to retrieve insights.
- Explored various visualization tools such as google charts and CanvasJS.

Reference

- https://www.tutorialspoint.com
- https://stackoverflow.com/
- http://text-processing.com/api/sentiment/
- http://www.jesse-anderson.com/2016/04/unit-testing-spark-with-java/
- https://canvasjs.com/javascript-charts/
- https://developers.google.com/chart/