

PRINCIPLES OF BIG DATA MANAGEMENT
PROJECT – III
SPRING 2017

Submitted by: TEAM 13

Sri Sai Anusha Gandu (sgr43)

Sushma Mitta (smgp6)

Sadanand Kallakuri (sk789)

Abhilash Reddy Gaddam (aggg6)

CONTENTS

Title	Pg. No.
1. Introduction	1
1.1 About Twitter	1
1.2 About the Project	1
2. Requirements	1
2.1 Languages	1
2.2 Software	1
3. Tasks	2
4. Query 1 (Using RDD)	2
5. Query 2	4
6. Query 3 (Using Trends File)	5
7. Extra Requirement	6
7.1 Pie Chart Representation of Query 1	7
7.2 Bar Graph Representation of Query 2	8
7.3 Line Chart Representation of Query 3	9
References	10

1. INTRODUCTION

1.1 About Twitter

Twitter is an online news and social networking service where users post and interact with messages, "tweets," restricted to 140 characters. Registered users can post tweets, but those who are unregistered can only read them. Twitter Inc. is based in San Francisco, California, United States, and has more than 25 offices around the world. Twitter was created in March 2006 by Jack Dorsey, Noah Glass, Biz Stone, and Evan Williams and launched in July, whereby the service rapidly gained worldwide popularity. As of 2016, Twitter had more than 319 million monthly active users.

1.2 About the Project

Here, we have collected the tweets using twitter API through tweepy using the keywords Python, JavaScript and Ruby in JSON (JavaScript Object Notation) format. The tweets then collected have been analyzed and different SQL queries are written to obtain the result.

2. REQUIREMENTS

2.1 Languages

1. Scala
2. SQL
3. Java
4. Javascript
5. HTML

2.2 Software

1. IntelliJ IDEA 3.4 (IDE)
2. JDK 1.8
3. Scala 2.12.1
4. Spark 2.1
5. Virtual Box (Cloudera)
6. WebStorm 2016.2.4

3. TASKS

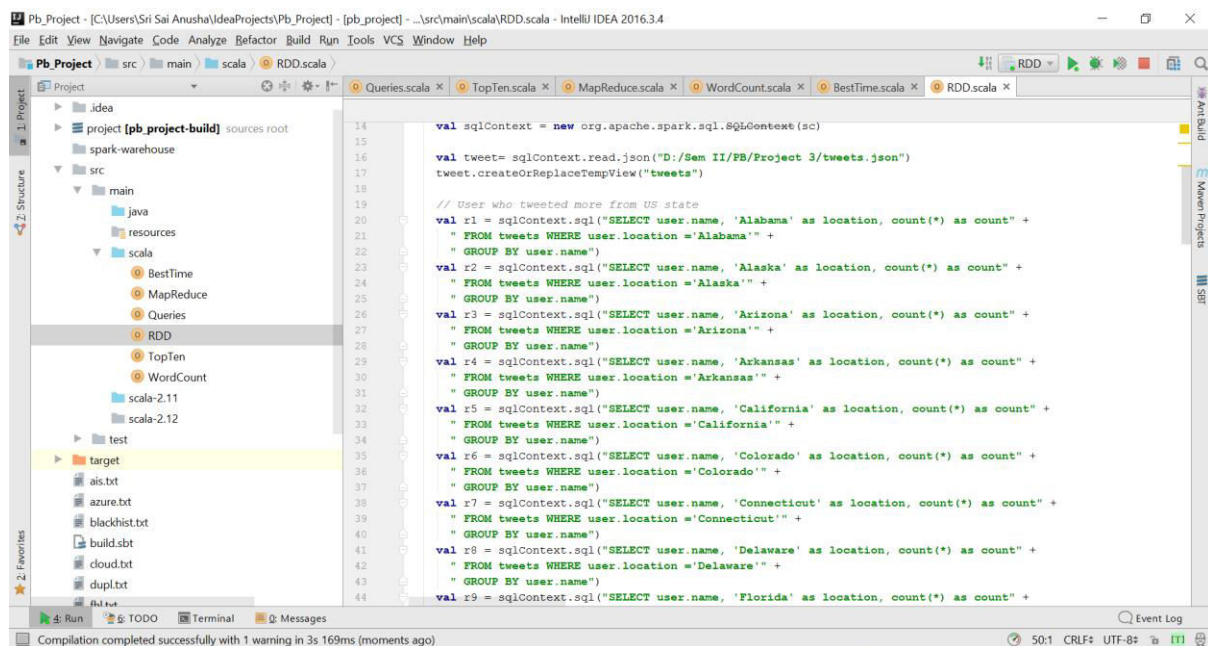
- Implement below three analytical tasks using a collection of tweets.
 1. One using RDD transformations and actions.
 2. The other two using Spark SQL and DFs.
 3. One of the query including the input file trends.txt

EXTRA REQUIREMENT

- To provide the visual implementation of the queries.

4. QUERY 1 (USING RDD)

The following RDD query extracts the users who have tweeted the most from US states.



```
14 val sqlContext = new org.apache.spark.sql.SQLContext(sc)
15
16 val tweet= sqlContext.read.json("D:/Sem II/PB/Project 3/tweets.json")
17 tweet.createOrReplaceTempView("tweets")
18
19 // User who tweeted more from US state
20 val r1 = sqlContext.sql("SELECT user.name, 'Alabama' as location, count(*) as count" +
21   " FROM tweets WHERE user.location = 'Alabama'" +
22   " GROUP BY user.name")
23 val r2 = sqlContext.sql("SELECT user.name, 'Alaska' as location, count(*) as count" +
24   " FROM tweets WHERE user.location = 'Alaska'" +
25   " GROUP BY user.name")
26 val r3 = sqlContext.sql("SELECT user.name, 'Arizona' as location, count(*) as count" +
27   " FROM tweets WHERE user.location = 'Arizona'" +
28   " GROUP BY user.name")
29 val r4 = sqlContext.sql("SELECT user.name, 'Arkansas' as location, count(*) as count" +
30   " FROM tweets WHERE user.location = 'Arkansas'" +
31   " GROUP BY user.name")
32 val r5 = sqlContext.sql("SELECT user.name, 'California' as location, count(*) as count" +
33   " FROM tweets WHERE user.location = 'California'" +
34   " GROUP BY user.name")
35 val r6 = sqlContext.sql("SELECT user.name, 'Colorado' as location, count(*) as count" +
36   " FROM tweets WHERE user.location = 'Colorado'" +
37   " GROUP BY user.name")
38 val r7 = sqlContext.sql("SELECT user.name, 'Connecticut' as location, count(*) as count" +
39   " FROM tweets WHERE user.location = 'Connecticut'" +
40   " GROUP BY user.name")
41 val r8 = sqlContext.sql("SELECT user.name, 'Delaware' as location, count(*) as count" +
42   " FROM tweets WHERE user.location = 'Delaware'" +
43   " GROUP BY user.name")
44 val r9 = sqlContext.sql("SELECT user.name, 'Florida' as location, count(*) as count" +
```

Compilation completed successfully with 1 warning in 3s 169ms (moments ago)

```

151 main(args: Array[String])
152 {
153   val r45 = sqlContext.sql("SELECT user.name, 'Vermont' as location, count(*) as count" +
154     " FROM tweets WHERE user.location = 'Vermont'" +
155     " GROUP BY user.name")
156   val r46 = sqlContext.sql("SELECT user.name, 'Virginia' as location, count(*) as count" +
157     " FROM tweets WHERE user.location = 'Virginia'" +
158     " GROUP BY user.name")
159   val r47 = sqlContext.sql("SELECT user.name, 'Washington' as location, count(*) as count" +
160     " FROM tweets WHERE user.location = 'Washington'" +
161     " GROUP BY user.name")
162   val r48 = sqlContext.sql("SELECT user.name, 'West Virginia' as location, count(*) as count" +
163     " FROM tweets WHERE user.location = 'West Virginia'" +
164     " GROUP BY user.name")
165   val r49 = sqlContext.sql("SELECT user.name, 'Wisconsin' as location, count(*) as count" +
166     " FROM tweets WHERE user.location = 'Wisconsin'" +
167     " GROUP BY user.name")
168   val r50 = sqlContext.sql("SELECT user.name, 'Wyoming' as location, count(*) as count" +
169     " FROM tweets WHERE user.location = 'Wyoming'" +
170     " GROUP BY user.name")
171   val group = r1.union(r2).union(r3).union(r4).union(r5).union(r6).union(r7).union(r8).union(r9).union(r10).union(r11)
172   group.createOrReplaceTempView("output")
173   val rdd = sqlContext.sql("SELECT * from output ORDER BY count DESC")
174   rdd.createOrReplaceTempView("result")
175   rdd.show()
176 }
177
178
179
180
181

```

Compilation completed successfully with 1 warning in 3s 169ms (moments ago)

The output thus obtained from the query is shown below.

```

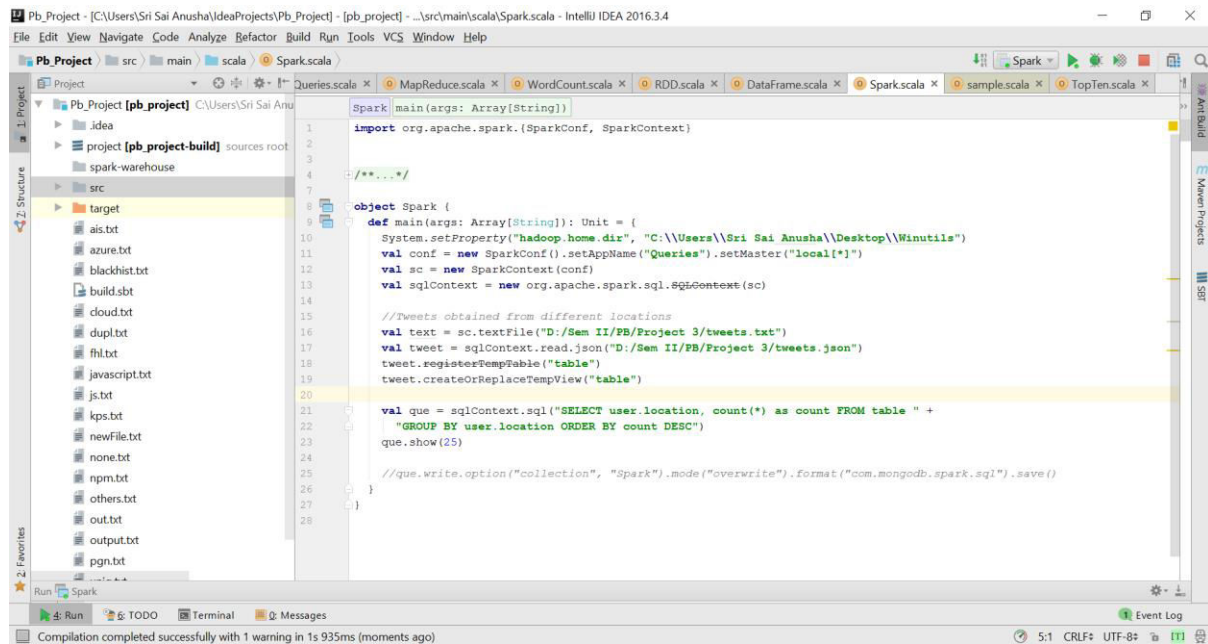
17/04/18 12:31:54 INFO DAGScheduler: Job 1 finished: show at RDD.scala:177, took 138.092354 s
17/04/18 12:31:54 INFO CodeGenerator: Code generated in 12.399403 ms
+-----+-----+-----+
| name | location | count |
+-----+-----+-----+
| Patrick Sanders | Arkansas | 21 |
| Ruby and Rails Log | New York | 15 |
| Ruby On Rails Jobs | New York | 10 |
| Brenda B Taylor | Texas | 9 |
| Torgeir Helgevoid | New York | 7 |
| ClearanceJobs.com NE | Nebraska | 5 |
| Uli Kunkel | New Jersey | 5 |
| AMagnificentMess | California | 4 |
| Alfred Hill | California | 3 |
| Rachel Hawkins | Alabama | 3 |
| Darin S. Lory | New York | 3 |
| Sherri Antonuccio | Oregon | 3 |
| MN Vikings Store | Minnesota | 3 |
| Loretta McComb | North Carolina | 3 |
| Cynthia Woolf | Colorado | 3 |
| Dot | Wisconsin | 3 |
| Moon Majick Studio | Arizona | 2 |
| maryricksen | Florida | 2 |
| randy | Kentucky | 2 |
| courtney | Georgia | 2 |
+-----+-----+-----+
only showing top 20 rows
17/04/18 12:31:54 INFO SparkContext: Invoking stop() from shutdown hook

```

Compilation completed successfully with 1 warning in 3s 169ms (3 minutes ago)

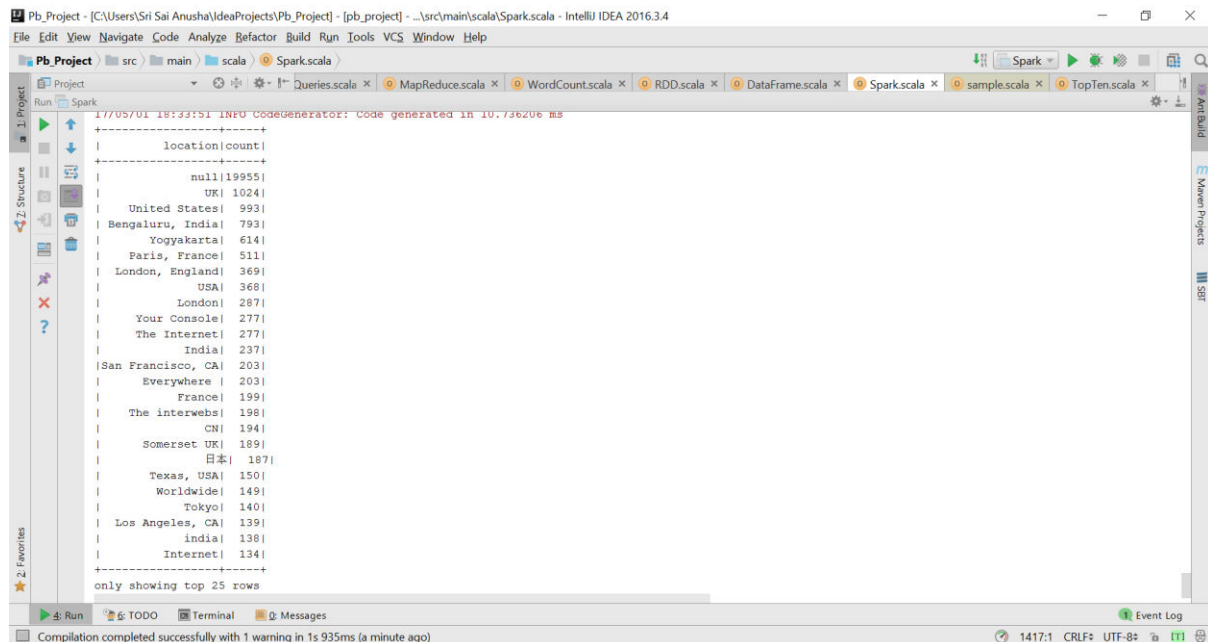
5. QUERY 2

This query extracts the tweet count obtained from different locations.



```
1 import org.apache.spark.{SparkConf, SparkContext}
2
3 /**...*/
4
5 object Spark {
6
7   def main(args: Array[String]): Unit = {
8     System.setProperty("hadoop.home.dir", "C:\\Users\\Sri Sai Anusha\\Desktop\\Winutils")
9     val conf = new SparkConf().setAppName("Queries").setMaster("local[*]")
10    val sc = new SparkContext(conf)
11    val sqlContext = new org.apache.spark.sql.SQLContext(sc)
12
13    //Tweets obtained from different locations
14    val text = sc.textFile("D:/Sem 11/PB/Project 3/tweets.txt")
15    val tweet = sqlContext.read.json("D:/Sem 11/PB/Project 3/tweets.json")
16    tweet.registerTempTable("table")
17    tweet.createOrReplaceTempView("table")
18
19    val que = sqlContext.sql("SELECT user.location, count(*) as count FROM table " +
20      "GROUP BY user.location ORDER BY count DESC")
21    que.show(25)
22
23    //que.write.option("collection", "Spark").mode("overwrite").format("com.mongodb.spark.sql").save()
24  }
25
26 }
27
28
```

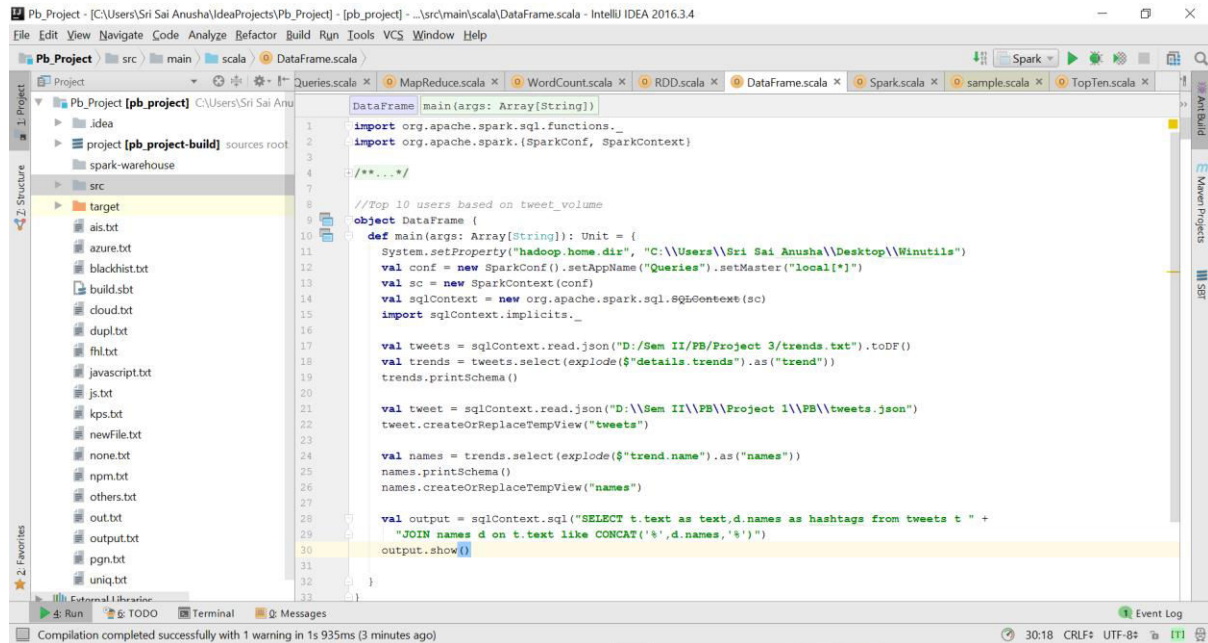
The following is the result obtained by the query.



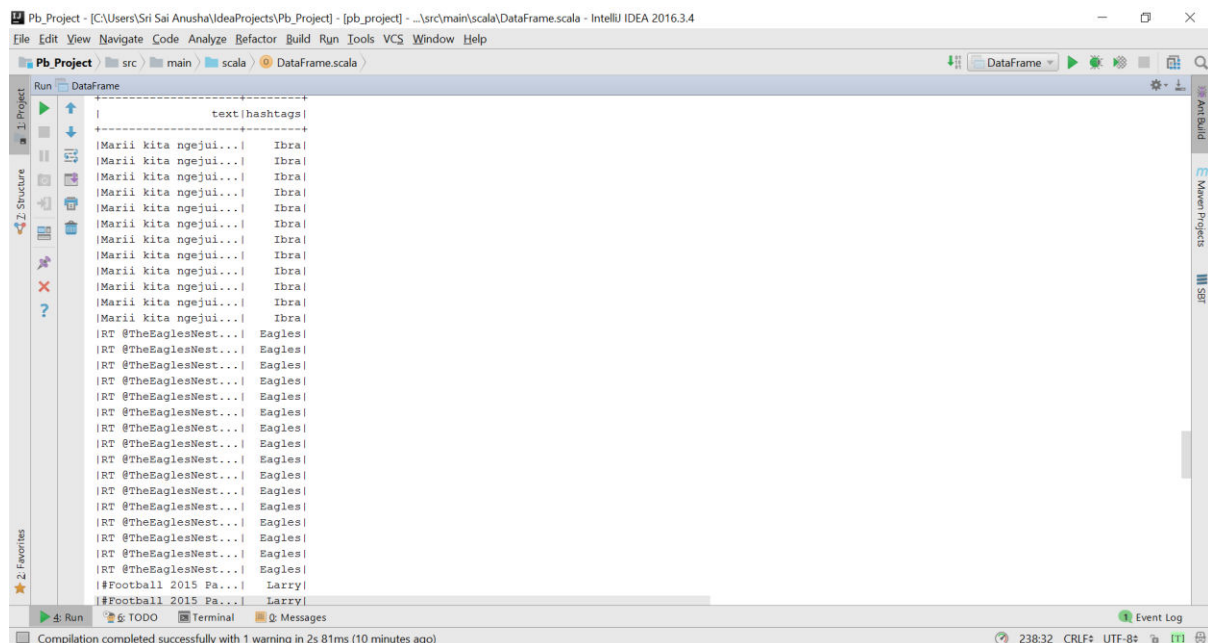
```
17/05/01 18:33:51 INFO CodeGenerator: Code generated in 10.736206 ms
+-----+
|location|count|
+-----+
|null|19955|
|UK|1024|
|United States|993|
|Bengaluru, India|793|
|Yogyakarta|614|
|Paris, France|511|
|London, England|369|
|USA|368|
|London|287|
|Your Console|277|
|The Internet|277|
|India|237|
|San Francisco, CA|203|
|Everywhere|203|
|France|199|
|The interwebs|198|
|CN|194|
|Somerset UK|189|
|日本|187|
|Texas, USA|150|
|Worldwide|149|
|Tokyo|140|
|Los Angeles, CA|139|
|india|138|
|Internet|134|
+-----+
only showing top 25 rows
```

6. QUERY 3 (USING TRENDS FILE)

This query searches through the names of the trends file and gives the output as the text and hashtags that are associated with the common names.



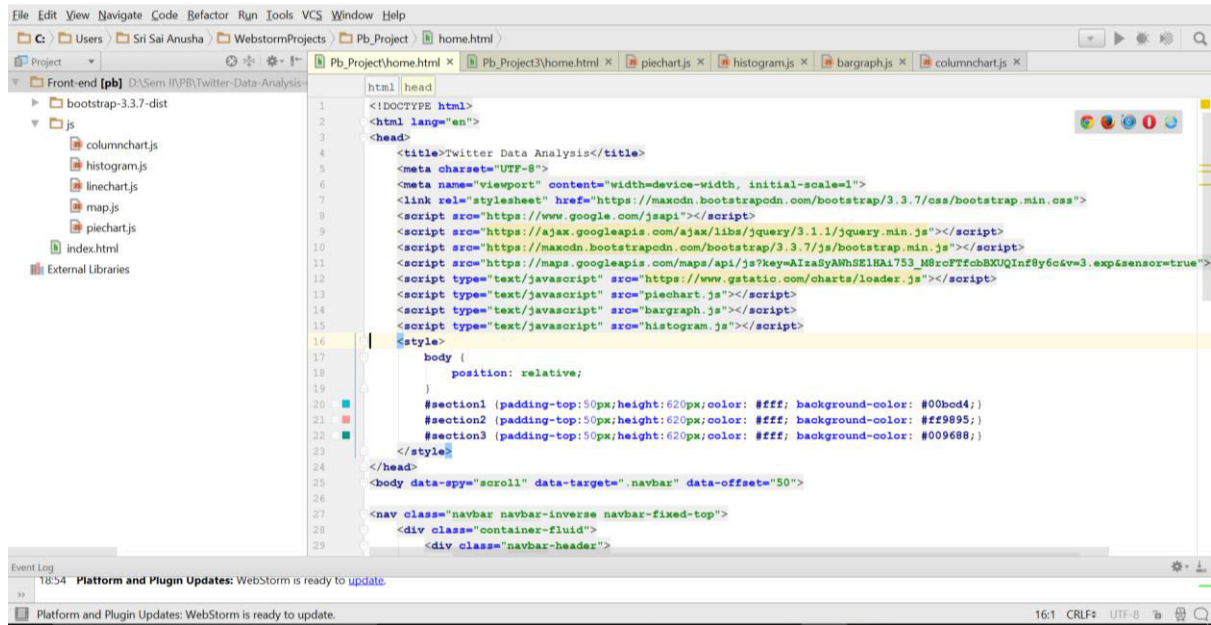
The output of the above query is shown below.



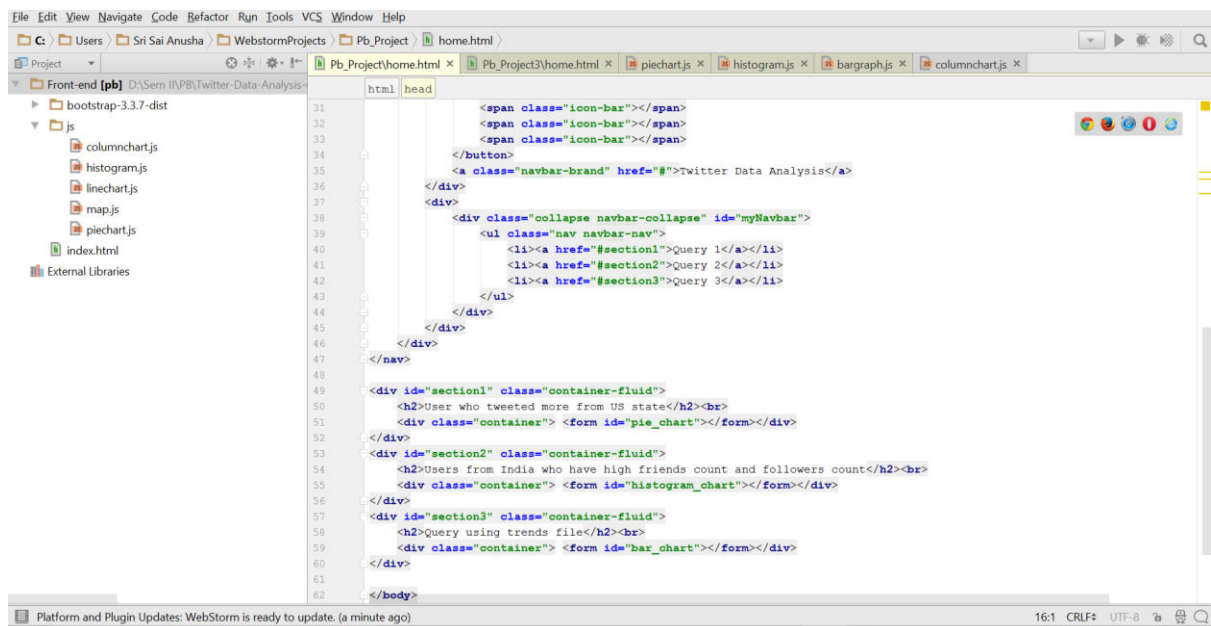
7. EXTRA REQUIREMENT

The visualization of the above queries is done for the dynamic representation of the queries.

The HTML code combining all the queries is shown below.



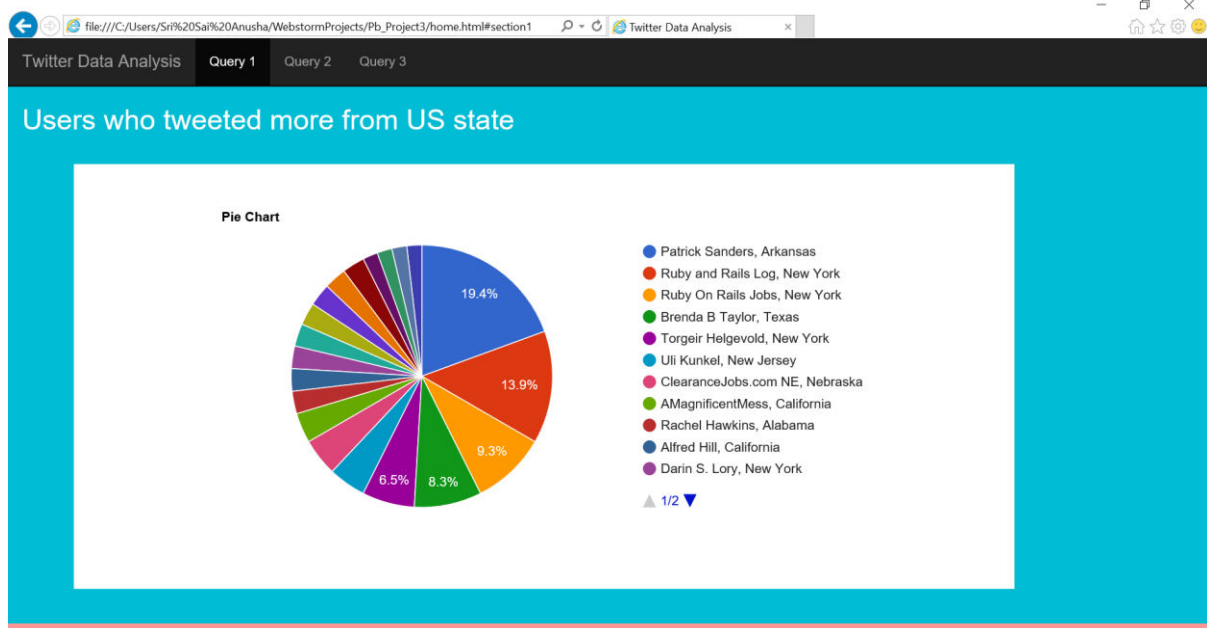
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <title>Twitter Data Analysis</title>
5 <meta charset="UTF-8">
6 <meta name="viewport" content="width=device-width, initial-scale=1">
7 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
8 <script src="https://www.google.com/jsapi"></script>
9 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
10 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
11 <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyAWhSE1l8A753_M0rc0TfcbBXUQlnf8y6c&v=3.exp&sensor=true">
12 <script type="text/javascript" src="https://www.getatic.com/charts/loader.js"></script>
13 <script type="text/javascript" src="piechart.js"></script>
14 <script type="text/javascript" src="bargraph.js"></script>
15 <script type="text/javascript" src="histogram.js"></script>
16 </head>
17 <body>
18 <div class="container-fluid">
19 <div class="navbar-header">
```



```
31 <div class="collapse navbar-collapse" id="myNavbar">
32 <ul class="nav navbar-nav">
33 <li><a href="#section1">Query 1</a></li>
34 <li><a href="#section2">Query 2</a></li>
35 <li><a href="#section3">Query 3</a></li>
36 </ul>
37 </div>
38 </div>
39 </div>
40 </div>
41 </div>
42 </div>
43 </div>
44 </div>
45 </div>
46 </div>
47 </div>
48 </div>
49 <div id="section1" class="container-fluid">
50 <h2>User who tweeted more from US state</h2><br>
51 <div class="container"> <form id="pie_chart"></form></div>
52 </div>
53 <div id="section2" class="container-fluid">
54 <h2>Users from India who have high friends count and followers count</h2><br>
55 <div class="container"> <form id="histogram_chart"></form></div>
56 </div>
57 <div id="section3" class="container-fluid">
58 <h2>Query using trends file</h2><br>
59 <div class="container"> <form id="bar_chart"></form></div>
60 </div>
61 </div>
62 </body>
```

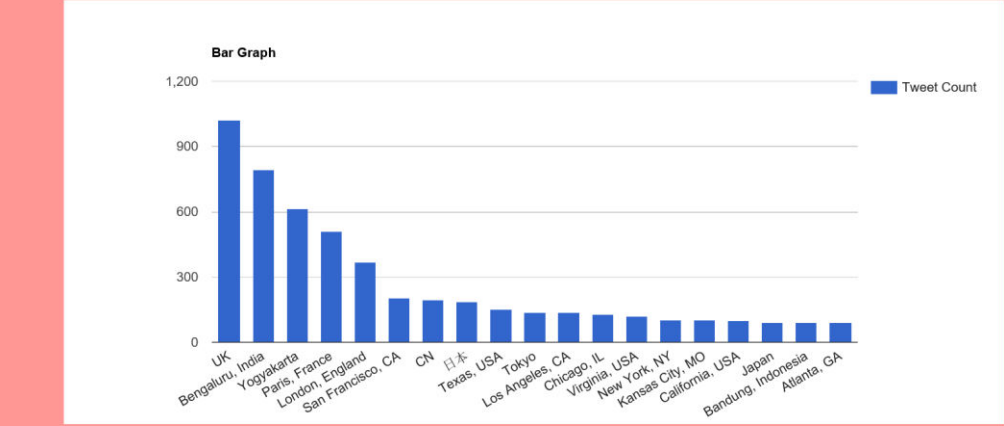

7.1 Pie Chart Representation of Query 1

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
C:\Users\Sri Sai Anusha\WebstormProjects\Pb_Project3\home.html x piechart.js x histogram.js x bargraph.js x columnchart.js x
Project
Front-end [pb] D:\Sem INPB\Twitter-Data-Analysis
  bootstrap-3.3.7-dist
  js
    columnchart.js
    histogram.js
    linechart.js
    map.js
    piechart.js
    index.html
  External Libraries
5 google.charts.load('current', {'packages':['corechart']});
6 google.charts.setOnLoadCallback(drawChart);
7
8 function drawChart() {
9   //API call
10  var data_file = "https://api.mlabb.com/api/1/databases/pb/collections/rdd?apiKey=m0infmMnGNwFFX2yinnjs2pxw00wtWYN";
11  var http_request = new XMLHttpRequest();
12
13  http_request.onreadystatechange = function() {
14
15    if (http_request.readyState == 4) {
16      var jsonObj = JSON.parse(http_request.responseText);
17      var data = new google.visualization.DataTable();
18      data.addColumn('string', 'Name and Location');
19      data.addColumn('number', 'Count');
20      for (var i=0; i<jsonObj.length; i++) {
21        data.addRow([
22          jsonObj[i].location, jsonObj[i].count
23        ]);
24      }
25
26      // Set chart options
27      var options = {
28        'title': 'Pie Chart',
29        'width': 1000,
30        'height': 450;
31      };
32
33      var chart = new google.visualization.PieChart(document.getElementById('pie_chart'));
34      chart.draw(data, options);
35    }
36
37    http_request.open("GET", data_file, true);
38    http_request.send();
39  }
40}
```

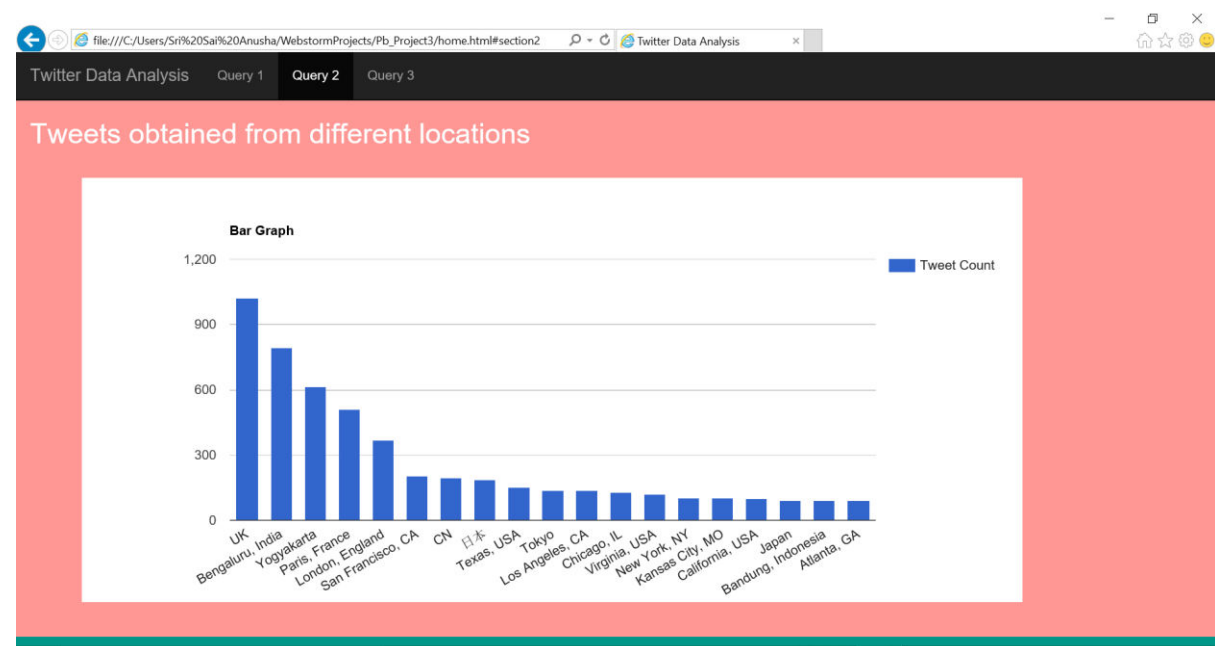


The dynamic representation of query 1 shows the users who tweeted the most from US state.

7.2 Bar Graph Representation of Query 2

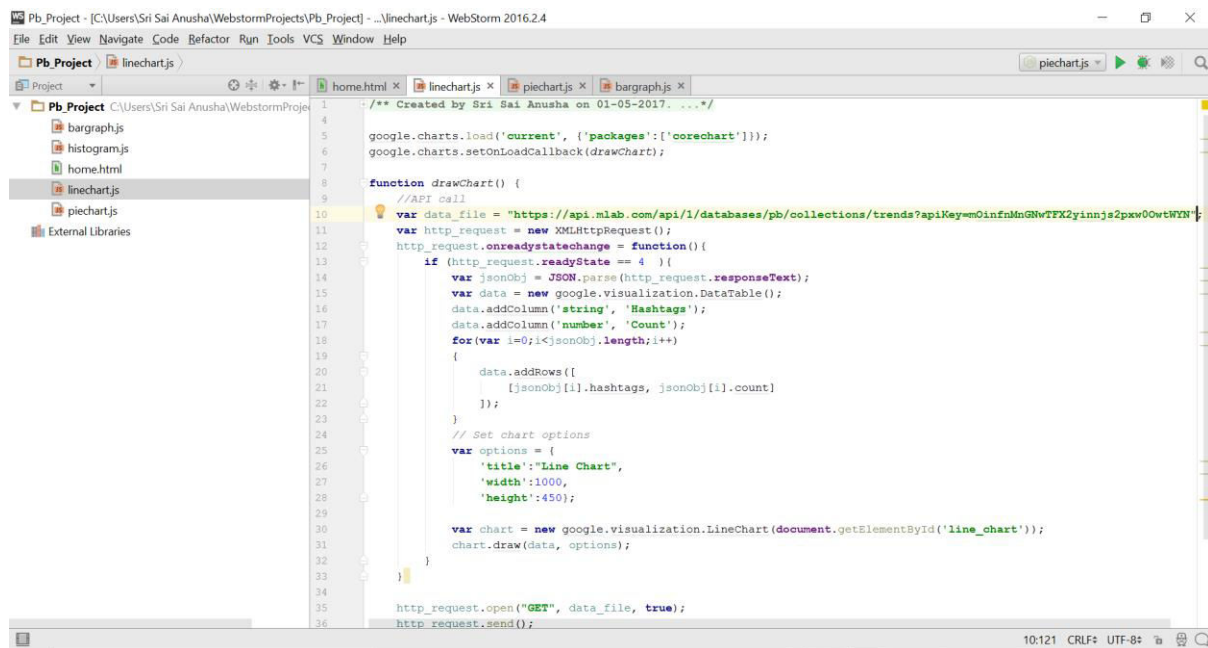


```
5 google.charts.load('current', {'packages':['corechart']});
6 google.charts.setOnLoadCallback(drawChart);
7
8 function drawChart() {
9     //API call
10    var data_file = "https://api.mlabb.com/api/1/databases/pb/collections/apark?apiKey=m0infmNGNwTFX2yinnjs2pxw00wtWYN";
11    var http_request = new XMLHttpRequest();
12
13    http_request.onreadystatechange = function() {
14
15        if (http_request.readyState == 4) {
16            var jsonObj = JSON.parse(http_request.responseText);
17            var data = new google.visualization.DataTable();
18            data.addColumn('string', 'Location');
19            data.addColumn('number', 'Tweet Count');
20            for (var i=0; i<jsonObj.length; i++) {
21                data.addRow([
22                    jsonObj[i].location, jsonObj[i].tweetcount
23                ]);
24            }
25            // Set chart options
26            var options = {
27                'title': 'Bar Graph',
28                'width': 1000,
29                'height': 450;
30            };
31
32            var chart = new google.visualization.ColumnChart(document.getElementById('bar_chart'));
33            chart.draw(data, options);
34        }
35    }
36    http_request.open("GET", data_file, true);
37    http_request.send();
38}
```

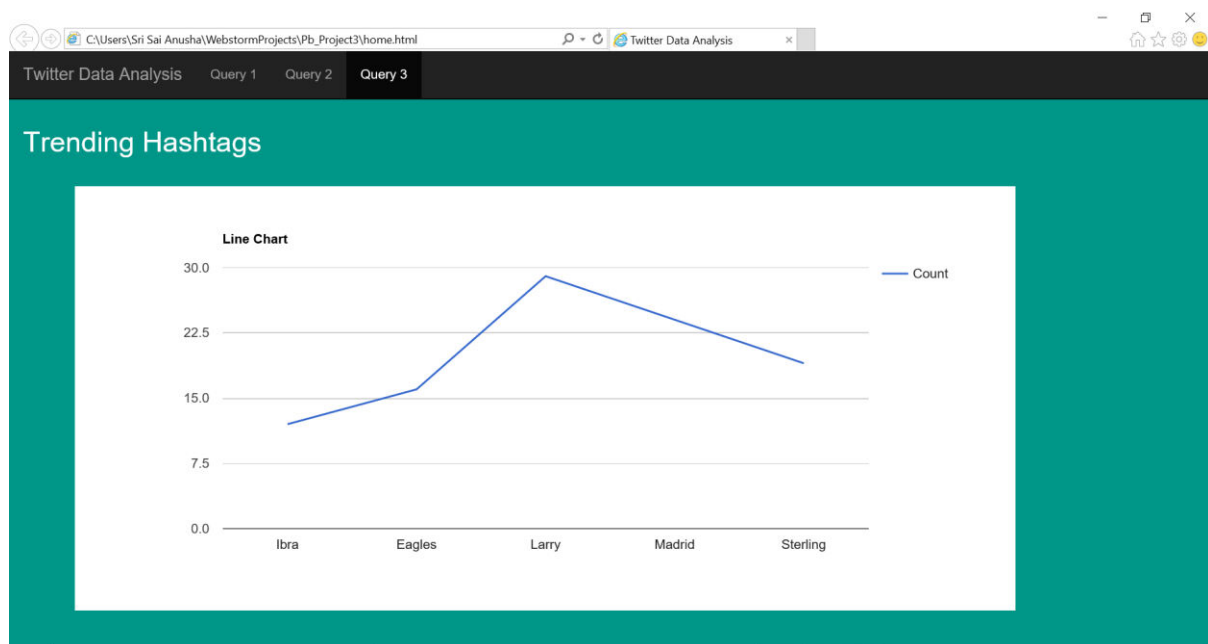


The dynamic representation of query 2 shows the tweets that have been obtained from different locations.

7.3 Line Chart Representation of Query 3



```
1  /** Created by Sri Sai Anusha on 01-05-2017. ... */
2
3  google.charts.load('current', {'packages':['corechart']});
4  google.charts.setOnLoadCallback(drawChart);
5
6  function drawChart() {
7      //API call
8      var data_file = "https://api.mlslab.com/api/1/databases/pb/collections/trends?apiKey=m0infmMnGmWTFX2yinnjs2pxw0OwtWYN";
9      var http_request = new XMLHttpRequest();
10     http_request.onreadystatechange = function() {
11         if (http_request.readyState == 4) {
12             var jsonObj = JSON.parse(http_request.responseText);
13             var data = new google.visualization.DataTable();
14             data.addColumn('string', 'Hashtags');
15             data.addColumn('number', 'Count');
16             for(var i=0;i<jsonObj.length;i++) {
17                 data.addRow([
18                     jsonObj[i].hashtags, jsonObj[i].count
19                 ]);
20             }
21             // Set chart options
22             var options = {
23                 'title': "Line Chart",
24                 'width': 1000,
25                 'height': 450;
26             };
27             var chart = new google.visualization.LineChart(document.getElementById('line_chart'));
28             chart.draw(data, options);
29         }
30     };
31     http_request.open("GET", data_file, true);
32     http_request.send();
33 }
```



The dynamic representation of query 3 shows trending hashtags that are common in the tweets file and trends file.

REFERENCES

- www.github.com
- <https://mlab.com/databases/pb>
- <https://developers.google.com/chart/interactive/docs/examples>
- <https://ikanow.com/how-to-visualize-json-documents-from-mongodb/>
- https://mvnrepository.com/artifact/org.mongodb.spark/mongo-spark-connector_2.11/0.1