COMP-SCI 5540 Principles of Big Data Management

University of Missouri-Kansas City

Department of Computer Science and Electrical Engineering

Project Report

GitHub URL: https://github.com/bharathkumarna/Principles-of-BigData

Team – 6

Abhiram Reddy Nalla

Bharath Kumar Natesan Arumugam

Sai Kumar Ponnamaneni

Sibi Chakravarthy Ramesh

Theme: (Wrestling)

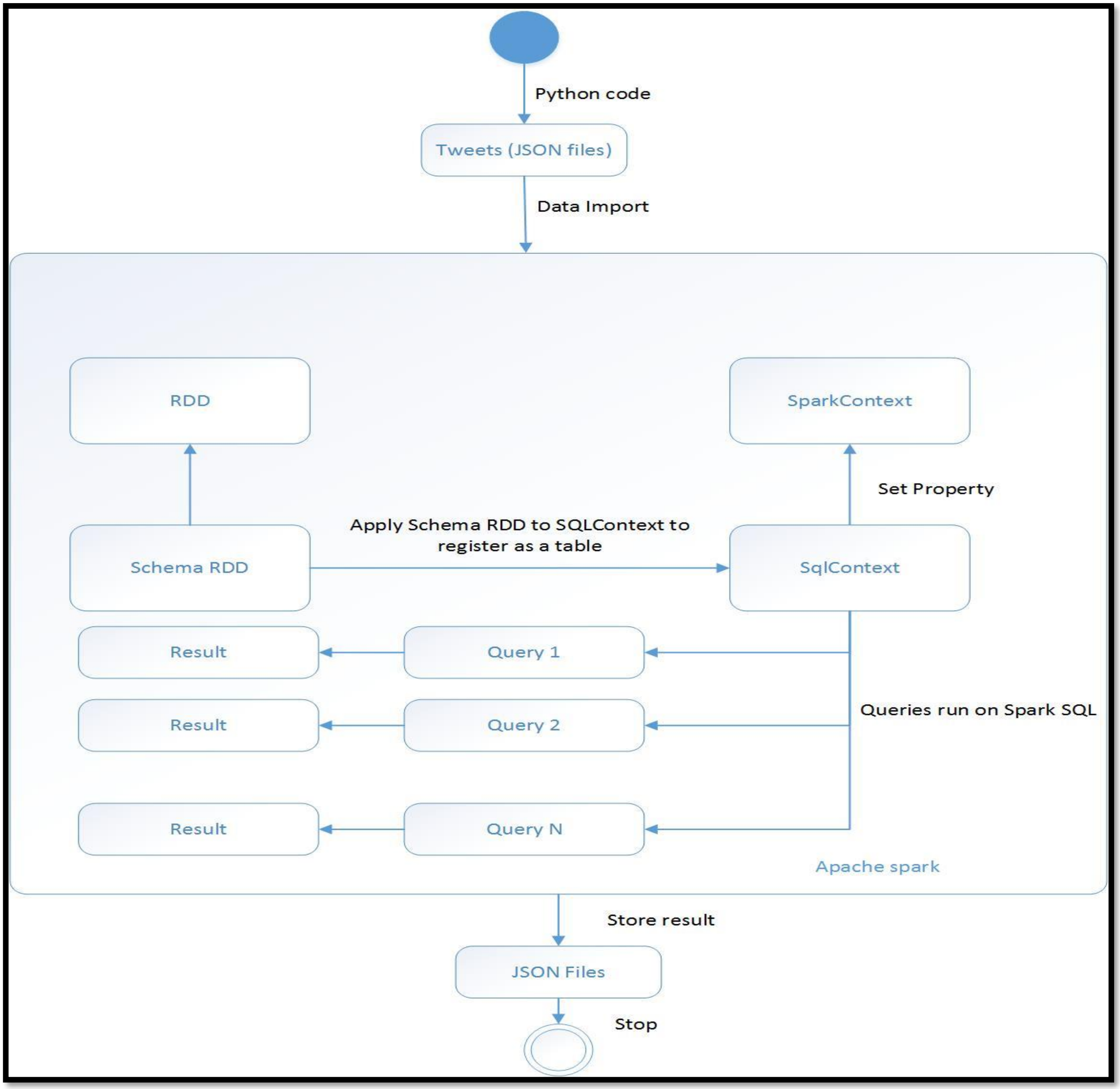
The Ultimate Fighting Championship (UFC) is the world's leading mixed martial arts (MMA) promoter and has held over 300 events to date. UFC is a combat sport abide by Unified Rules of Mixed Martial Arts where the outcomes are pre-determined and the matches are not choreographed. The UFC also connects with tens of millions of fans through its social media sites like Facebook, Instagram, and Twitter. The estimated tweets posted per hour (based on 1% sample) about #UFC is around 800.

World Wrestling Entertainment, Inc. (WWE) is an entertainment company that deals primarily in professional wrestling. WWE shows are purely entertainment based, featuring storyline-driven, scripted and choreographed matches. The estimated tweets posted per hour (based on 1% sample) about #WWE is around 750.

References:

1. Wikipedia
2. Hashtags.org Analytics

UML Diagram:


Python code
Tweets (JSON files)
Data Import
RDD
SparkContext
Set Property
Apply Schema RDD to SQLContext to register as a table
Schema RDD
SqlContext
Result
Query 1
Result
Query 2
Queries run on Spark SQL
Result
Query N
Apache spark
Store result
JSON Files
Stop

Design Steps:

1. Collect social media data (tweets) using any theme as filter and store it as JSON files.
2. A Spark Context is created to establish connection to Spark Cluster.
3. SQL Context class is created which represents an entry point into all functionality in Spark SQL.
4. Data Frames are created based on content of JSON file and register it to tables.
5. Run SQL queries programmatically using SQL function on registered tables.
6. Store the returned results as JSON file.

Libraries:

Spark Core contains the basic functionality of Spark and Spark SQL is Spark's package for working with Structured data.

1. org.apache.spark:spark-core_2.11:2.0.02
2. org.apache.spark:spark-sql_2.11:2.0.02

Signpost has been designed to work in conjunction with Apache HTTPComponents library for signing HTTP messages on the Scala platform in conformance with the OAuth Core 1.0 standard.

3. oauth.signpost:signpost-commonshttp4:1.2.1.22
4. org.apache.directory.studio:org.apache.httpcomponents.httpclient:4.02
5. signpost-core-1.2.1.22
6. org.apache.directory.studio:org.apache.httpcomponents.httpcore:4.02

Tweepy – An easy-to-use Python library for accessing the Twitter API.

7. tweepy-3.5.0

APIs:

1. Twitter public REST APIs - GET followers/ids

   Resource URL: *https://api.twitter.com/1.1/followers/ids.json*

Returns a collection of user IDs for every user following the specified user.

Programming Languages:

1. Scala – to run Spark Programs.
2. Python – to run Tweets collection program.

Environment:

Runtime Information:

Name	***Value***
Java Version	1.8.0_101 (Oracle Corporation)
Scala Version	version 2.11.8

Spark Properties:

Name	***Value***
spark.sql.warehouse.dir	file:///c:/tmp/spark-warehouse
spark.scheduler.mode	FIFO
spark.master	local[2]
spark.executor.id	driver
spark.driver.port	55681
spark.driver.host	192.168.1.146
spark.app.name	CountSpark
spark.app.id	local-1478459427915

System Properties:

Name	***Value***
file.encoding	UTF-8
hadoop.home.dir	C:\hadoop-2.3.0\bin\tweet
idea.launcher.bin.path	C:\Program Files (x86)\JetBrains\IntelliJ IDEA Community Edition 2016.2.5\bin
os.arch	amd64
os.name	Windows 10
os.version	10.0

Queries:

Query 1:

Description:

Query to display the top 10 users who tweeted the most times.

Code:

val Query1 = sqlcontext.sql("select user.name,user.screen_name, count(user.followers_count) as tweetsCount from querytable1 group by user.screen_name,user.name order by tweetsCount desc limit 10")

Query 2:

Description:

Query to display the top 10 users with most Sensitive Tweet numbers.

Code:

val Query2 = sqlcontext.sql("select user.name,count(user.name) as no_of_sensitive_tweets from querytable1 where possibly_sensitive=true and user.lang='en' group by user.name order by no_of_sensitive_tweets desc limit 10")

Query 3:

Description:

Query to display the top hashtags used in my collected tweets in conjunction with data in the HashtagsTopics.txt file posted on Blackboard.

Code:

val Query3 = sqlcontext.sql("select querytable3.name,count(querytable1.text) as count from querytable1 join querytable3 on querytable1.text like concat ('%',querytable3.name,'%') group by querytable3.name order by count desc limit 10 ")

Query 4:

Description:

Query to display cities with most number of Twitter users.

Code:

```
val Query4 = sqlcontext.sql("select user.location,count(*) as no_of_users  from querytable2 where user.location <> 'null' and user.location like concat ('%', ',' ,'%') group by user.location order by no_of_users desc limit 10")
```

Query 5:

Description:

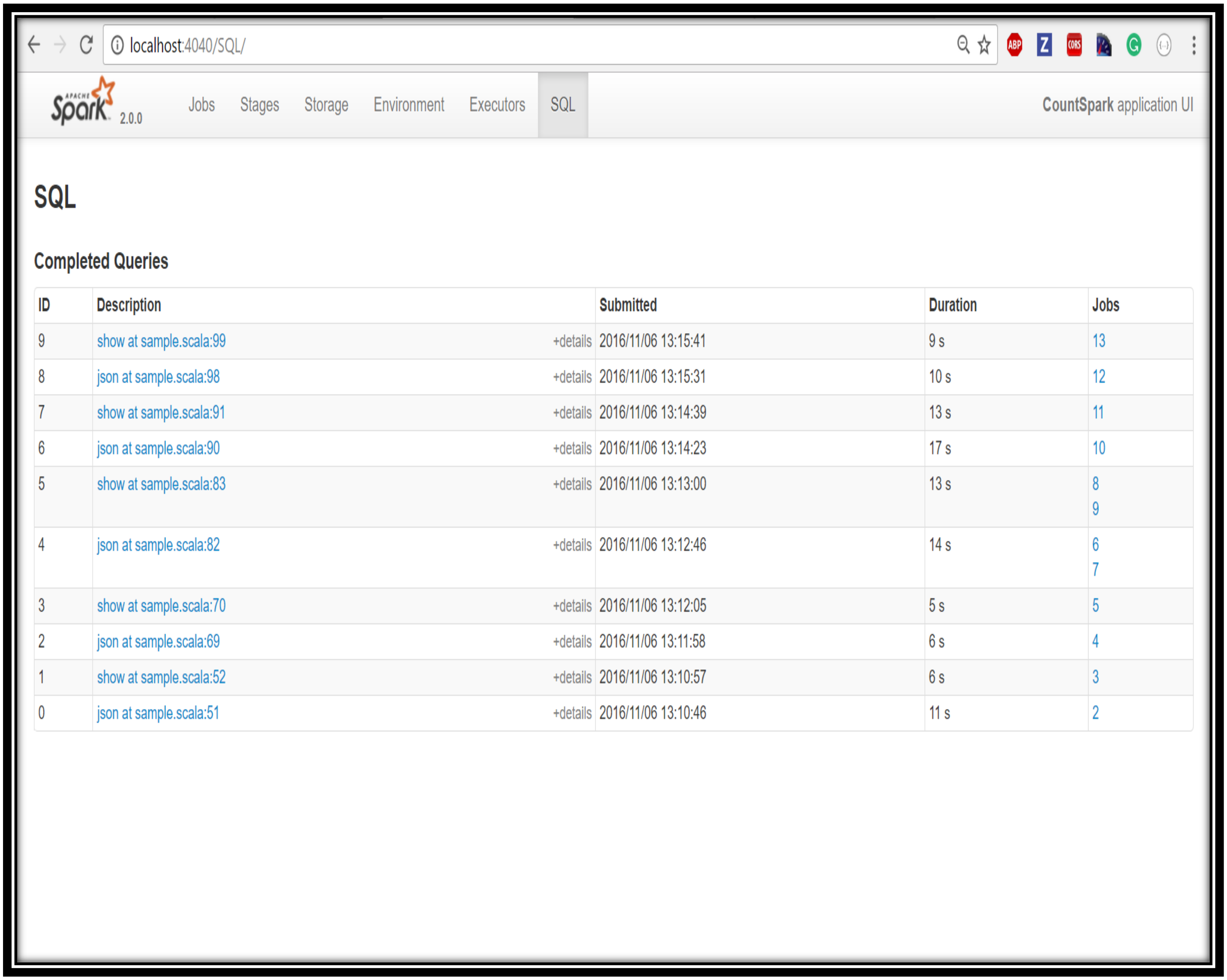
Query to display the most popular time zones.

Code:

```
val Query5 = sqlcontext.sql("SELECT user.time_zone,count(*) as no_of_tweets from querytable2 where user.time_zone <> 'null' group by user.time_zone order by no_of_tweets desc limit 10")
```

Runtime Measurements for Queries:

Query	***Runtime – JSON*** **(sec)**	***Runtime – Display*** **(sec)**	***Total*** **(sec)**
Query 1	11	6	17
Query 2	6	5	11
Query 3	14	13	27
Query 4	17	13	30
Query 5	10	9	19



SQL

Completed Queries

ID	Description		Submitted	Duration	Jobs
9	show at sample.scala:99	+details	2016/11/06 13:15:41	9 s	13
8	json at sample.scala:98	+details	2016/11/06 13:15:31	10 s	12
7	show at sample.scala:91	+details	2016/11/06 13:14:39	13 s	11
6	json at sample.scala:90	+details	2016/11/06 13:14:23	17 s	10
5	show at sample.scala:83	+details	2016/11/06 13:13:00	13 s	8 9
4	json at sample.scala:82	+details	2016/11/06 13:12:46	14 s	6 7
3	show at sample.scala:70	+details	2016/11/06 13:12:05	5 s	5
2	json at sample.scala:69	+details	2016/11/06 13:11:58	6 s	4
1	show at sample.scala:52	+details	2016/11/06 13:10:57	6 s	3
0	json at sample.scala:51	+details	2016/11/06 13:10:46	11 s	2

Code:

Collecting Tweets:

```
from tweepy.streaming import StreamListener

from tweepy import OAuthHandler

from tweepy import Stream


#Twitter Authentication

access_token = "1048610250-QQZ8D05FWBIon130QSgjg0XGDN0dw3lXXhP7KFt"

access_token_secret = "RRiMG6c7mIY61apEJWSwoxMMaSVN8tQwIcuK627ugp46r"

consumer_key = "RRAnQIWfiuDBpJm94OWgwmpEF"

consumer_secret = "uXj3hPKmkU931K8ye5FMZemBUky4UyEQxQCz2Ej5qyS4zp0Ddw"


class StdOutListener(StreamListener):


    def on_data(self, data):

        print(data)

        with open('fetched_tweet.json','a') as tf:

            tf.write(data)

        return True


if __name__ == '__main__':

    l = StdOutListener()

    auth = OAuthHandler(consumer_key, consumer_secret)

    auth.set_access_token(access_token, access_token_secret)

    stream = Stream(auth, l)


#Filter Tweets according to theme

    stream.filter(track=['UFC','WWE'])
```

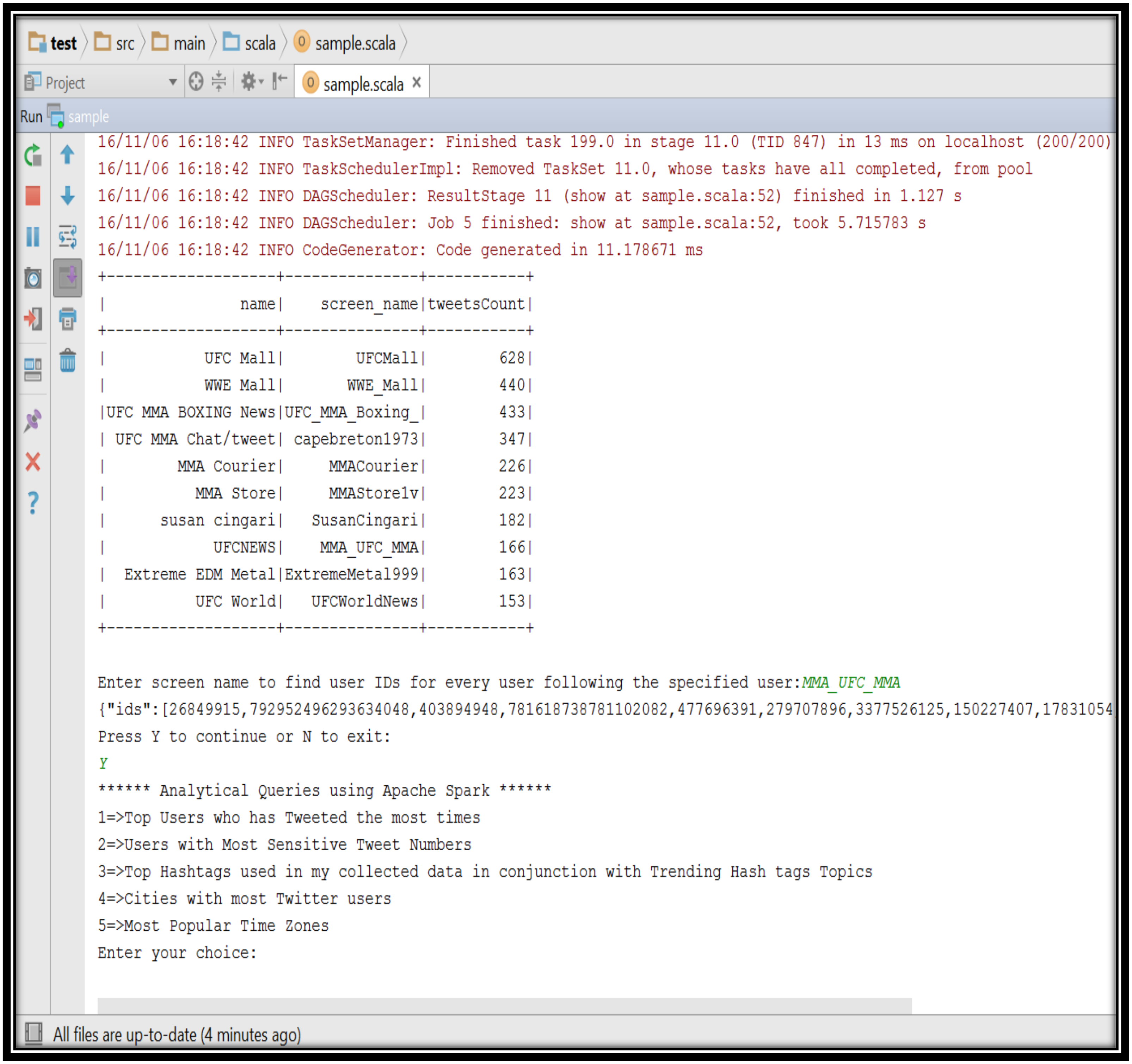
Spark SQL Program:

```
import oauth.signpost.commonshttp.CommonsHttpOAuthConsumer
import org.apache.commons.io.IOUtils
import org.apache.http.client.methods.HttpGet
import org.apache.http.impl.client.DefaultHttpClient
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext

object sample {
  //Twitter Authentication
  val AccessToken = "1048610250-
QQZ8D05FWBIon130QSgjg0XGDN0dw3lXXhP7KFt";
  val AccessSecret = "RRiMG6c7mIY61apEJWSwoxMMaSVN8tQwIcuK627ugp46r";
  val ConsumerKey = "RRAnQIWfiuDBpJm94OWgwmpEF";
  val ConsumerSecret =
"uXj3hPKmkU931K8ye5FMZemBUky4UyEQxQCz2Ej5qyS4zp0Ddw";

  def main(args: Array[String]) {

    System.setProperty("hadoop.home.dir","C:\\hadoop-
2.3.0\\bin\\tweet")
    val conf = new
SparkConf().setAppName("CountSpark").setMaster("local[2]").set("spark.
sql.warehouse.dir","file:///c:/tmp/spark-warehouse")
    val sc = new SparkContext(conf)
    val sqlcontext = new SQLContext(sc)
    import sqlcontext.implicits._
    //Spark DataFrames
    val tweetsfile =
sqlcontext.read.json("C:\\Users\\bn4n5\\workspace\\Pb-
ass\\mypackage\\fetched_tweet.json")
    tweetsfile.registerTempTable("querytable1")

    //Spark RDD's
    val string=sc.textFile("C:\\Users\\bn4n5\\workspace\\Pb-
ass\\mypackage\\fetched_tweet.json")
    sqlcontext.jsonRDD(string).registerTempTable("querytable2")

    var a='Y'
    while (a=='Y') {
    //Menu Option
    println("****** Analytical Queries using Apache Spark ******")
    println("1=>Top Users who has Tweeted the most times")
    println("2=>Users with Most Sensitive Tweet Numbers")
    println("3=>Top Hashtags used in my collected data in conjunction
with Trending Hash tags Topics")
    println("4=>Cities with most Twitter users")
    println("5=>Most Popular Time Zones")
    println("Enter your choice:")
    val choice=readInt()
      choice match {

        case 1 =>
          //Query 1 using Spark DataFrames
          val Query1 = sqlcontext.sql("select
user.name,user.screen_name, count(user.followers_count) as tweetsCount
from querytable1 group by user.screen_name,user.name order by
tweetsCount desc limit 10")
```

```
          Query1.write.json("C:\\Users\\bn4n5\\workspace\\Pb-
ass\\mypackage\\Query1")
          Query1.show()

          //Query 1 calling public API
          val name = readLine("Enter screen name to find user IDs for
every user following the specified user:")
          val consumer = new CommonsHttpOAuthConsumer(ConsumerKey,
ConsumerSecret)
          consumer.setTokenWithSecret(AccessToken, AccessSecret)
          val request = new
HttpGet("https://api.twitter.com/1.1/followers/ids.json?cursor=-
1&screen_name=" + name)
          consumer.sign(request)
          val client = new DefaultHttpClient()
          val response = client.execute(request)
          println(IOUtils.toString(response.getEntity().getContent()))
          println("Press Y to continue or N to exit:")
          a = readChar()

        case 2 =>
          //Query 2 using Spark DataFrames
          val Query2 = sqlcontext.sql("select
user.name,count(user.name) as no_of_sensitive_tweets from querytable1
where possibly_sensitive=true and user.lang='en' group by user.name
order by no_of_sensitive_tweets desc limit 10")
          Query2.write.json("C:\\Users\\bn4n5\\workspace\\Pb-
ass\\mypackage\\Query2")
          Query2.show()
          println("Press Y to continue or N to exit:")
          a = readChar()
        case 3 =>
          //Query 3 using Spark DataFrames
          //Query 3 uses data in the PopularHahtagsAndTopics.txt file
posted on Blackboard in conjunction with my collected data
          val text = sc.textFile("C:\\Users\\bn4n5\\workspace\\Pb-
ass\\mypackage\\PopularHahtagsAndTopics.txt").map(_.split("/n")).map(f
rt => Text(frt(0))).toDF()
          text.registerTempTable("querytable")
          val Query=sqlcontext.sql("select querytable.name from
querytable where querytable.name like '%#UFC%' or querytable.name like
'%#WWE%' or querytable.name like '%#MMA%' ")
          Query.registerTempTable("querytable3")
          val Query3 = sqlcontext.sql("select
querytable3.name,count(querytable1.text) as count from querytable1
join querytable3 on querytable1.text like concat
('%',querytable3.name,'%') group by querytable3.name order by count
desc limit 10 ")
          Query3.write.json("C:\\Users\\bn4n5\\workspace\\Pb-
ass\\mypackage\\Query3")
          Query3.show();
          println("Press Y to continue or N to exit:")
          a = readChar()

        case 4 =>
          //Query 4 using Spark RDD's
          val Query4 = sqlcontext.sql("select user.location,count(*)
as no_of_users  from querytable2 where user.location <> 'null' and
user.location like concat ('%',',','%') group by user.location order
by no_of_users desc limit 10")
          Query4.write.json("C:\\Users\\bn4n5\\workspace\\Pb-
ass\\mypackage\\Query4")
          Query4.show()
          println("Press Y to continue or N to exit:")
          a = readChar()

        case 5 =>
          //Query 5 using Spark RDD's
          val Query5 = sqlcontext.sql("SELECT user.time_zone,count(*)
as no_of_tweets from querytable2 where user.time_zone <> 'null' group
by user.time_zone order by no_of_tweets desc limit 10")
          Query5.write.json("C:\\Users\\bn4n5\\workspace\\Pb-
ass\\mypackage\\Query5")
          Query5.show()
          println("Press Y to continue or N to exit:")
          a = readChar()



      }
    }
  }
}
case class Text(name: String)
```

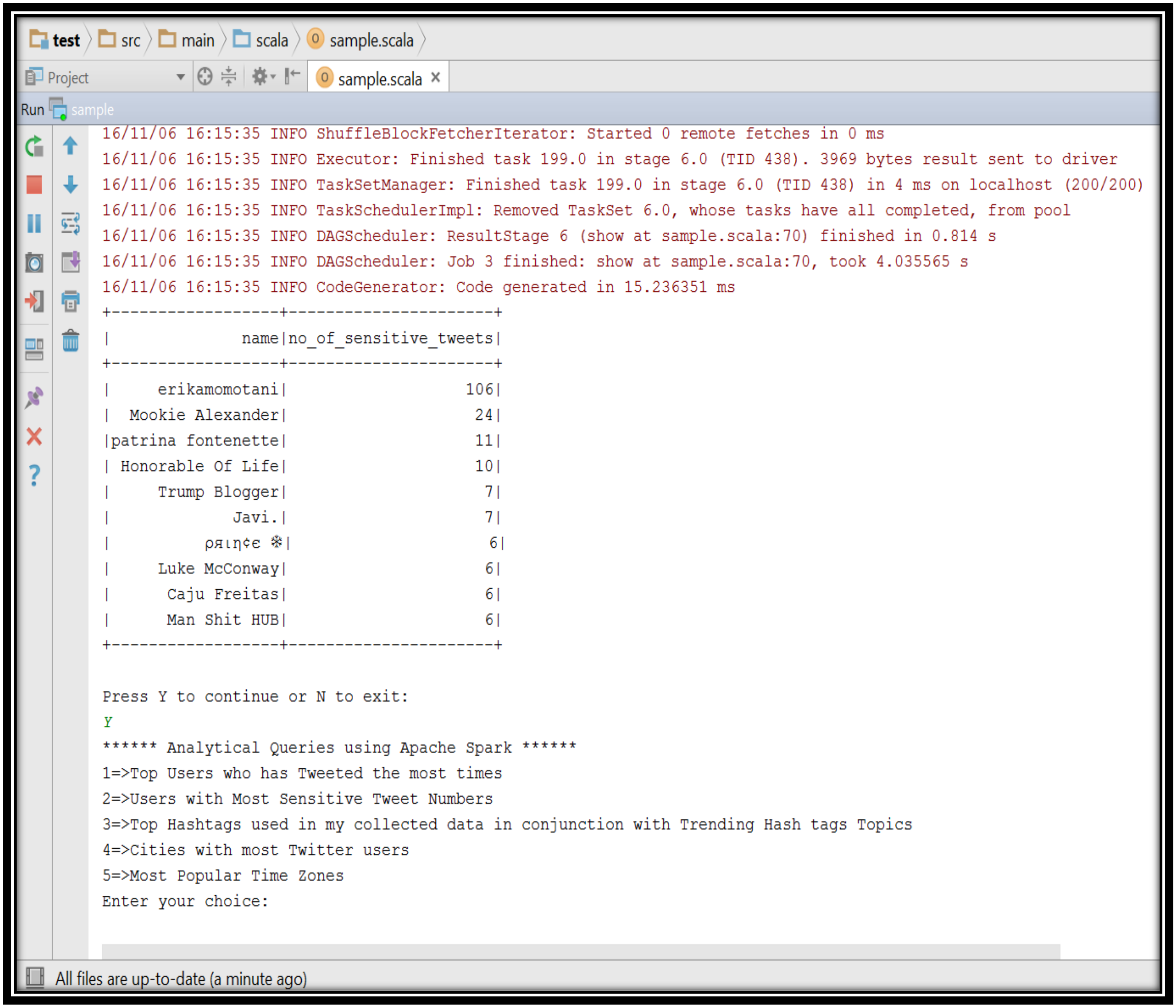
Output:

Query 1: **Top Users who has Tweeted the most times (Twitter API)**



```
16/11/06 16:18:42 INFO TaskSetManager: Finished task 199.0 in stage 11.0 (TID 847) in 13 ms on localhost (200/200)
16/11/06 16:18:42 INFO TaskSchedulerImpl: Removed TaskSet 11.0, whose tasks have all completed, from pool
16/11/06 16:18:42 INFO DAGScheduler: ResultStage 11 (show at sample.scala:52) finished in 1.127 s
16/11/06 16:18:42 INFO DAGScheduler: Job 5 finished: show at sample.scala:52, took 5.715783 s
16/11/06 16:18:42 INFO CodeGenerator: Code generated in 11.178671 ms
+-------------------+---------------+-----------+
|               name|    screen_name|tweetsCount|
+-------------------+---------------+-----------+
|           UFC Mall|        UFCMall|        628|
|           WWE Mall|       WWE_Mall|        440|
|UFC MMA BOXING News|UFC_MMA_Boxing_|        433|
| UFC MMA Chat/tweet| capebreton1973|        347|
|        MMA Courier|     MMACourier|        226|
|          MMA Store|     MMAStore1v|        223|
|      susan cingari|    SusanCingari|        182|
|            UFCNEWS|    MMA_UFC_MMA|        166|
|  Extreme EDM Metal|ExtremeMetal999|        163|
|          UFC World|   UFCWorldNews|        153|
+-------------------+---------------+-----------+

Enter screen name to find user IDs for every user following the specified user:MMA_UFC_MMA
{"ids":[26849915,792952496293634048,403894948,781618738781102082,477696391,279707896,3377526125,150227407,17831054
Press Y to continue or N to exit:
Y
****** Analytical Queries using Apache Spark ******
1=>Top Users who has Tweeted the most times
2=>Users with Most Sensitive Tweet Numbers
3=>Top Hashtags used in my collected data in conjunction with Trending Hash tags Topics
4=>Cities with most Twitter users
5=>Most Popular Time Zones
Enter your choice:
```

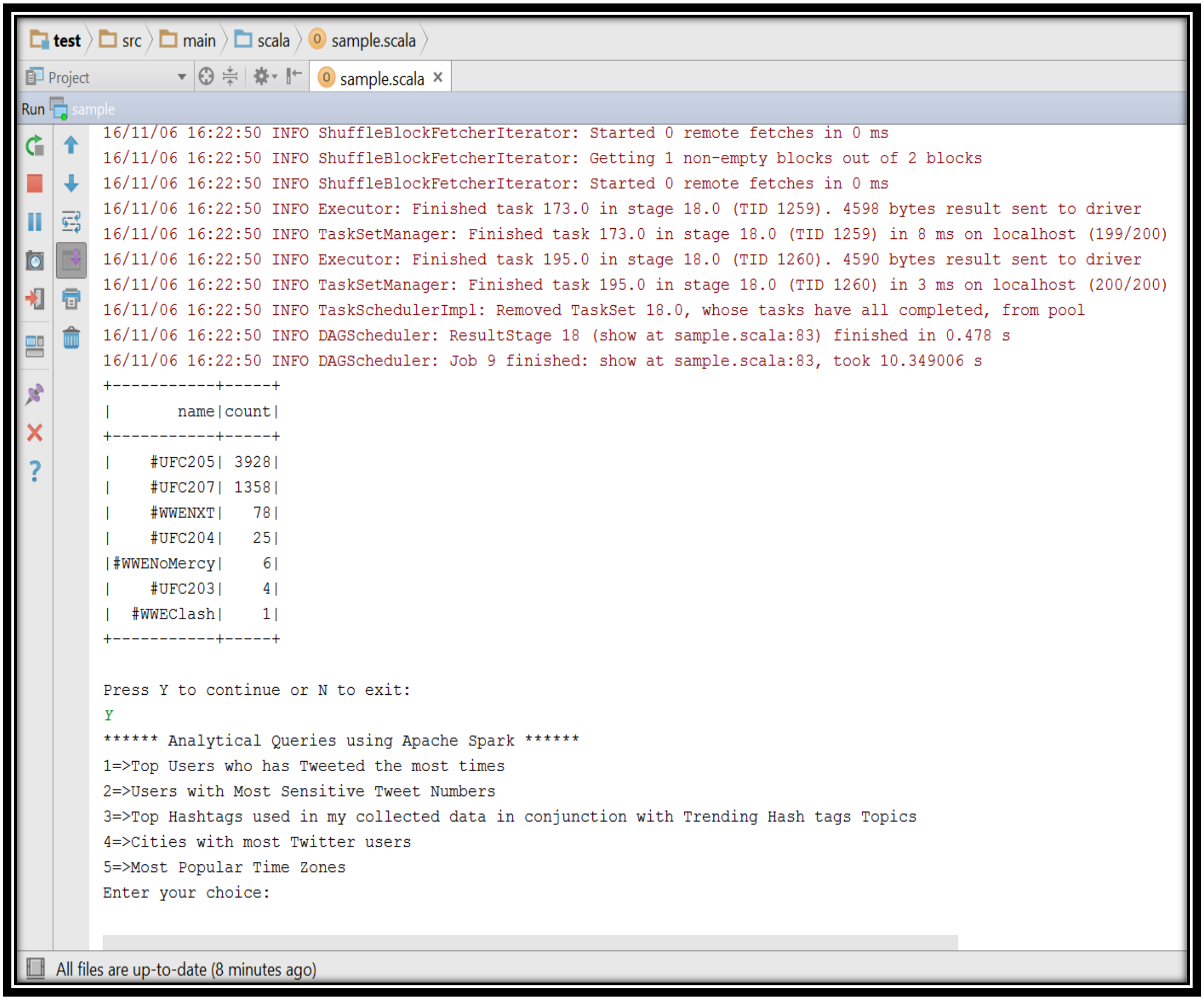
Query 2: **Users with Most Sensitive Tweet Numbers**



```
16/11/06 16:15:35 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
16/11/06 16:15:35 INFO Executor: Finished task 199.0 in stage 6.0 (TID 438). 3969 bytes result sent to driver
16/11/06 16:15:35 INFO TaskSetManager: Finished task 199.0 in stage 6.0 (TID 438) in 4 ms on localhost (200/200)
16/11/06 16:15:35 INFO TaskSchedulerImpl: Removed TaskSet 6.0, whose tasks have all completed, from pool
16/11/06 16:15:35 INFO DAGScheduler: ResultStage 6 (show at sample.scala:70) finished in 0.814 s
16/11/06 16:15:35 INFO DAGScheduler: Job 3 finished: show at sample.scala:70, took 4.035565 s
16/11/06 16:15:35 INFO CodeGenerator: Code generated in 15.236351 ms
+------------------+----------------------+
|              name|no_of_sensitive_tweets|
+------------------+----------------------+
|     erikamomotani|                   106|
|  Mookie Alexander|                    24|
|patrina fontenette|                    11|
| Honorable Of Life|                    10|
|      Trump Blogger|                     7|
|              Javi.|                     7|
|          ρяιη¢є ❄|                     6|
|      Luke McConway|                     6|
|       Caju Freitas|                     6|
|       Man Shit HUB|                     6|
+------------------+----------------------+

Press Y to continue or N to exit:
Y
****** Analytical Queries using Apache Spark ******
1=>Top Users who has Tweeted the most times
2=>Users with Most Sensitive Tweet Numbers
3=>Top Hashtags used in my collected data in conjunction with Trending Hash tags Topics
4=>Cities with most Twitter users
5=>Most Popular Time Zones
Enter your choice:
```

All files are up-to-date (a minute ago)

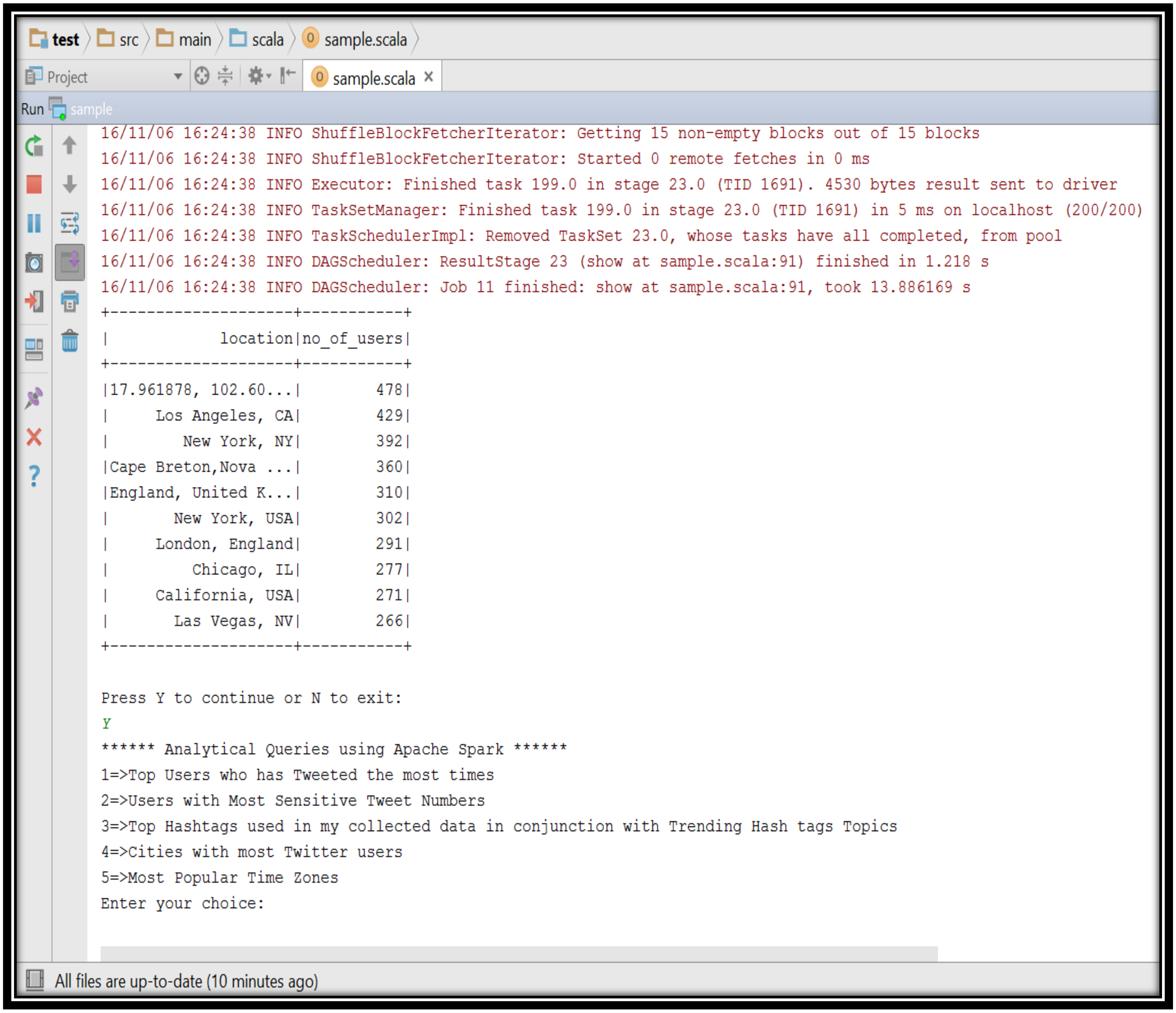
Query 3: **Top Hashtags used in my collected data in conjunction with Trending Hashtags Topics**



```
16/11/06 16:22:50 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
16/11/06 16:22:50 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 2 blocks
16/11/06 16:22:50 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
16/11/06 16:22:50 INFO Executor: Finished task 173.0 in stage 18.0 (TID 1259). 4598 bytes result sent to driver
16/11/06 16:22:50 INFO TaskSetManager: Finished task 173.0 in stage 18.0 (TID 1259) in 8 ms on localhost (199/200)
16/11/06 16:22:50 INFO Executor: Finished task 195.0 in stage 18.0 (TID 1260). 4590 bytes result sent to driver
16/11/06 16:22:50 INFO TaskSetManager: Finished task 195.0 in stage 18.0 (TID 1260) in 3 ms on localhost (200/200)
16/11/06 16:22:50 INFO TaskSchedulerImpl: Removed TaskSet 18.0, whose tasks have all completed, from pool
16/11/06 16:22:50 INFO DAGScheduler: ResultStage 18 (show at sample.scala:83) finished in 0.478 s
16/11/06 16:22:50 INFO DAGScheduler: Job 9 finished: show at sample.scala:83, took 10.349006 s
+-----------+-----+
|       name|count|
+-----------+-----+
|    #UFC205| 3928|
|    #UFC207| 1358|
|    #WWENXT|   78|
|    #UFC204|   25|
|#WWENoMercy|    6|
|    #UFC203|    4|
|  #WWEClash|    1|
+-----------+-----+

Press Y to continue or N to exit:
Y
****** Analytical Queries using Apache Spark ******
1=>Top Users who has Tweeted the most times
2=>Users with Most Sensitive Tweet Numbers
3=>Top Hashtags used in my collected data in conjunction with Trending Hash tags Topics
4=>Cities with most Twitter users
5=>Most Popular Time Zones
Enter your choice:
```

All files are up-to-date (8 minutes ago)

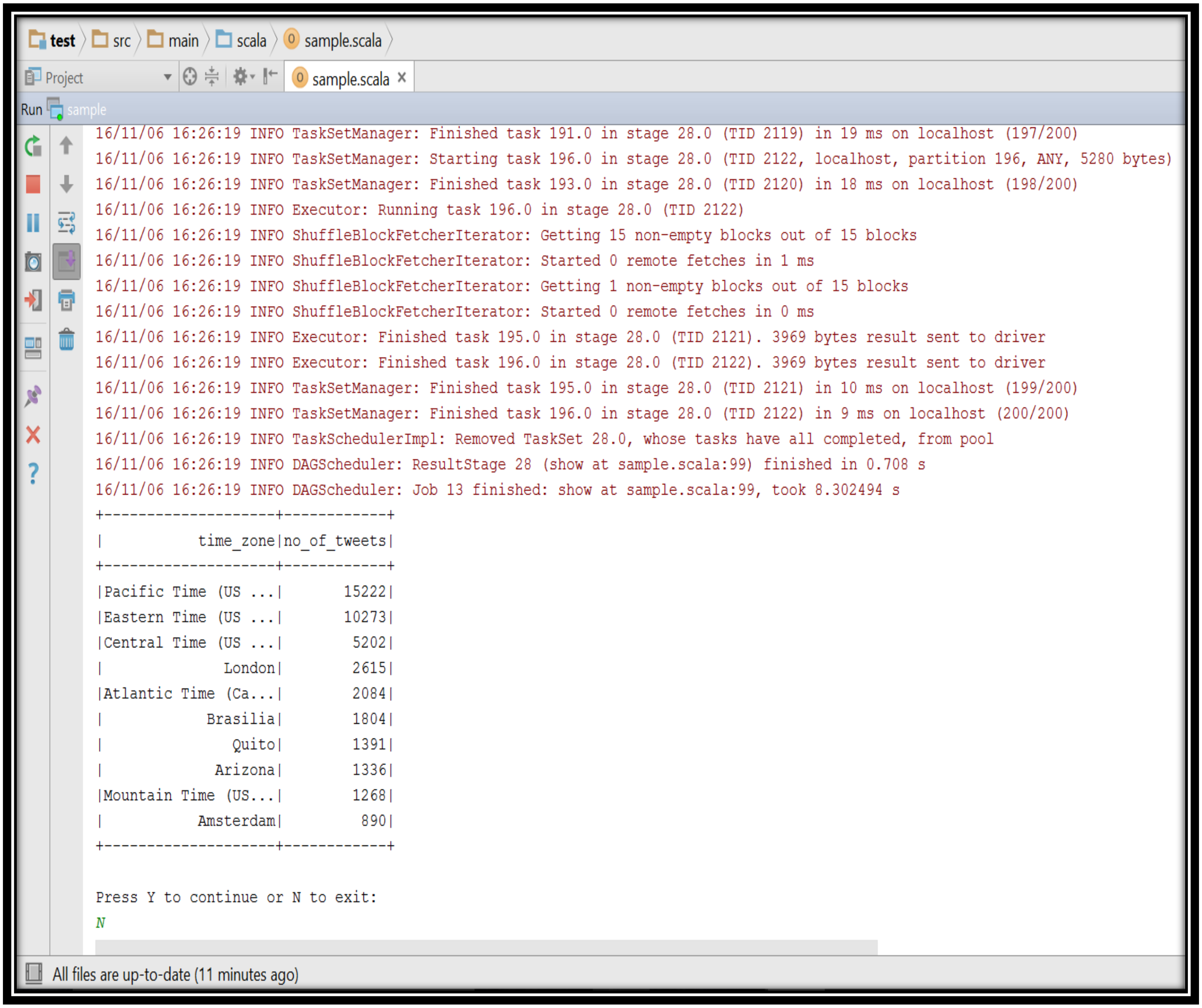
Query 4: **Cities with most Twitter users**



```
16/11/06 16:24:38 INFO ShuffleBlockFetcherIterator: Getting 15 non-empty blocks out of 15 blocks
16/11/06 16:24:38 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
16/11/06 16:24:38 INFO Executor: Finished task 199.0 in stage 23.0 (TID 1691). 4530 bytes result sent to driver
16/11/06 16:24:38 INFO TaskSetManager: Finished task 199.0 in stage 23.0 (TID 1691) in 5 ms on localhost (200/200)
16/11/06 16:24:38 INFO TaskSchedulerImpl: Removed TaskSet 23.0, whose tasks have all completed, from pool
16/11/06 16:24:38 INFO DAGScheduler: ResultStage 23 (show at sample.scala:91) finished in 1.218 s
16/11/06 16:24:38 INFO DAGScheduler: Job 11 finished: show at sample.scala:91, took 13.886169 s
+--------------------+-----------+
|            location|no_of_users|
+--------------------+-----------+
|17.961878, 102.60...|        478|
|     Los Angeles, CA|        429|
|        New York, NY|        392|
|Cape Breton,Nova ...|        360|
|England, United K...|        310|
|       New York, USA|        302|
|     London, England|        291|
|         Chicago, IL|        277|
|     California, USA|        271|
|       Las Vegas, NV|        266|
+--------------------+-----------+

Press Y to continue or N to exit:
Y
****** Analytical Queries using Apache Spark ******
1=>Top Users who has Tweeted the most times
2=>Users with Most Sensitive Tweet Numbers
3=>Top Hashtags used in my collected data in conjunction with Trending Hash tags Topics
4=>Cities with most Twitter users
5=>Most Popular Time Zones
Enter your choice:
```

All files are up-to-date (10 minutes ago)

Query 5: **Most Popular Time Zones**



```
16/11/06 16:26:19 INFO TaskSetManager: Finished task 191.0 in stage 28.0 (TID 2119) in 19 ms on localhost (197/200)
16/11/06 16:26:19 INFO TaskSetManager: Starting task 196.0 in stage 28.0 (TID 2122, localhost, partition 196, ANY, 5280 bytes)
16/11/06 16:26:19 INFO TaskSetManager: Finished task 193.0 in stage 28.0 (TID 2120) in 18 ms on localhost (198/200)
16/11/06 16:26:19 INFO Executor: Running task 196.0 in stage 28.0 (TID 2122)
16/11/06 16:26:19 INFO ShuffleBlockFetcherIterator: Getting 15 non-empty blocks out of 15 blocks
16/11/06 16:26:19 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 1 ms
16/11/06 16:26:19 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 15 blocks
16/11/06 16:26:19 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
16/11/06 16:26:19 INFO Executor: Finished task 195.0 in stage 28.0 (TID 2121). 3969 bytes result sent to driver
16/11/06 16:26:19 INFO Executor: Finished task 196.0 in stage 28.0 (TID 2122). 3969 bytes result sent to driver
16/11/06 16:26:19 INFO TaskSetManager: Finished task 195.0 in stage 28.0 (TID 2121) in 10 ms on localhost (199/200)
16/11/06 16:26:19 INFO TaskSetManager: Finished task 196.0 in stage 28.0 (TID 2122) in 9 ms on localhost (200/200)
16/11/06 16:26:19 INFO TaskSchedulerImpl: Removed TaskSet 28.0, whose tasks have all completed, from pool
16/11/06 16:26:19 INFO DAGScheduler: ResultStage 28 (show at sample.scala:99) finished in 0.708 s
16/11/06 16:26:19 INFO DAGScheduler: Job 13 finished: show at sample.scala:99, took 8.302494 s
+--------------------+------------+
|           time_zone|no_of_tweets|
+--------------------+------------+
|Pacific Time (US ...|       15222|
|Eastern Time (US ...|       10273|
|Central Time (US ...|        5202|
|              London|        2615|
|Atlantic Time (Ca...|        2084|
|            Brasilia|        1804|
|               Quito|        1391|
|             Arizona|        1336|
|Mountain Time (US...|        1268|
|           Amsterdam|         890|
+--------------------+------------+

Press Y to continue or N to exit:
N
```

All files are up-to-date (11 minutes ago)