COMP-SCI 5540 Principles of Big Data Management
University of Missouri-Kansas City

Department of Computer Science and Electrical Engineering

Project Report Phase-2

Team - 6
Trinadha Raji Muppala
Pranoop Mutha
Vinay Jaibheem

GitHub URL: https://github.com/PranoopMutha/CS5540_PB_FlyingSquirrels_TwitterProject

Design Steps:

1. Collect social media data (tweets) using any theme as filter and store it as a JSON file.
   We achieved the above functionality in Python. The main part is getting consumer key, consumer secret, access token and access secret. Then we will be using the filter command to get the tweets.

2. A Spark Context is created to establish connection to Spark Cluster.
   We achieved this task by creating a Spark Context to establish a 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.

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

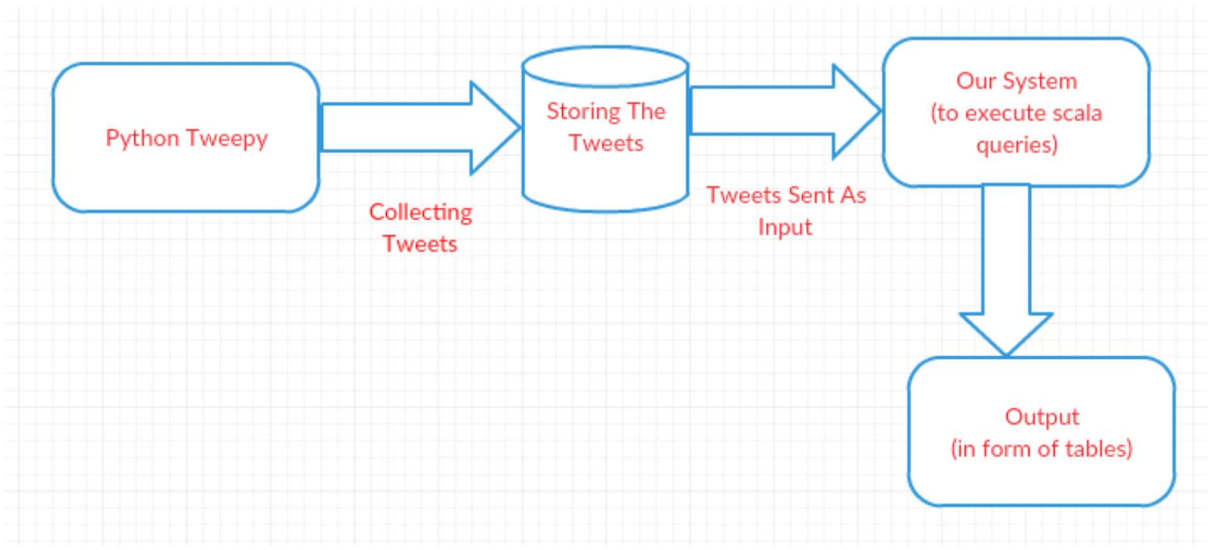
APIs:

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/Web Languages:

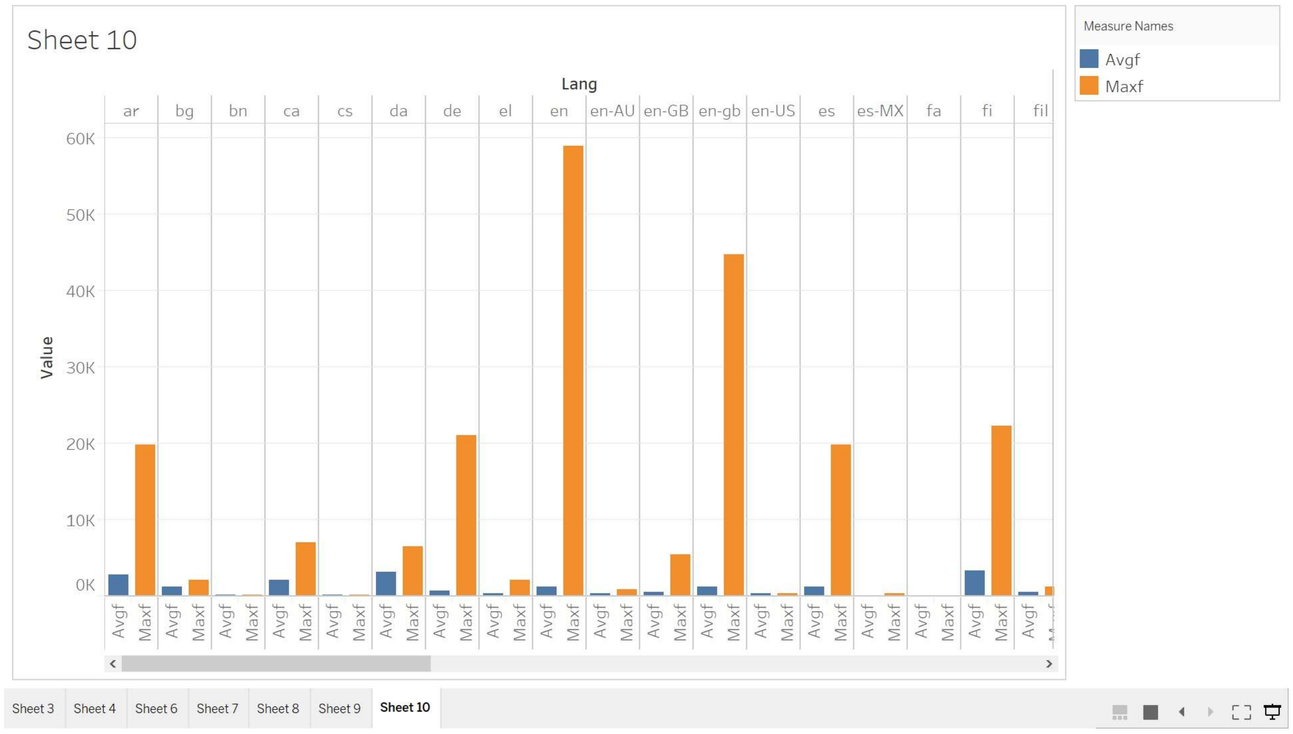
1. Scala – to run Spark Programs.
2. HTML5, CSS3 – to design user interface and front-end development.
3. JavaScript – to do API calls and visualize.

System Architecture:



Output:

Query 1:

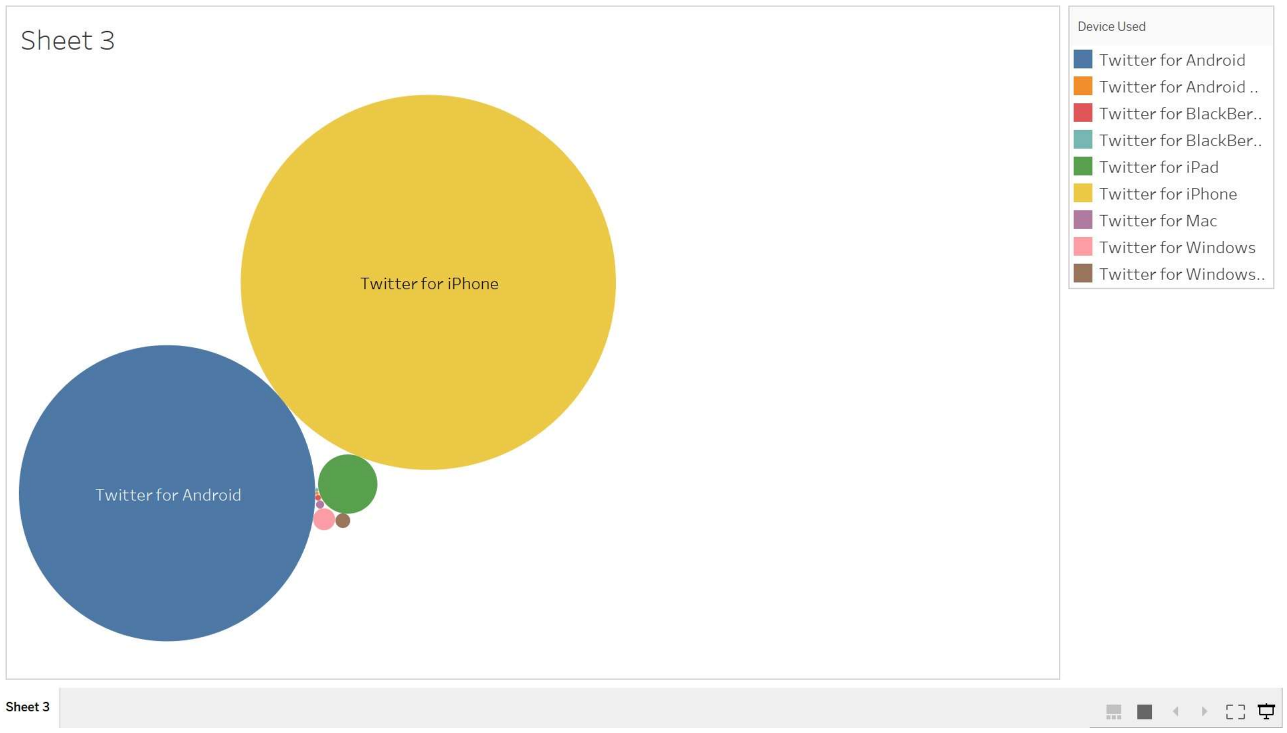


According to average and maximum friends count based on language.

Query Output:

lang	avgf	maxf
ar	2801.961	19831
bg	1144.333	2093
bn	147.5	148
ca	2034.714	7021
cs	137.8333	247
da	3219.2	6541
de	789.4528	21058
el	390.5667	2117
en	1187.787	59061

Query 2:



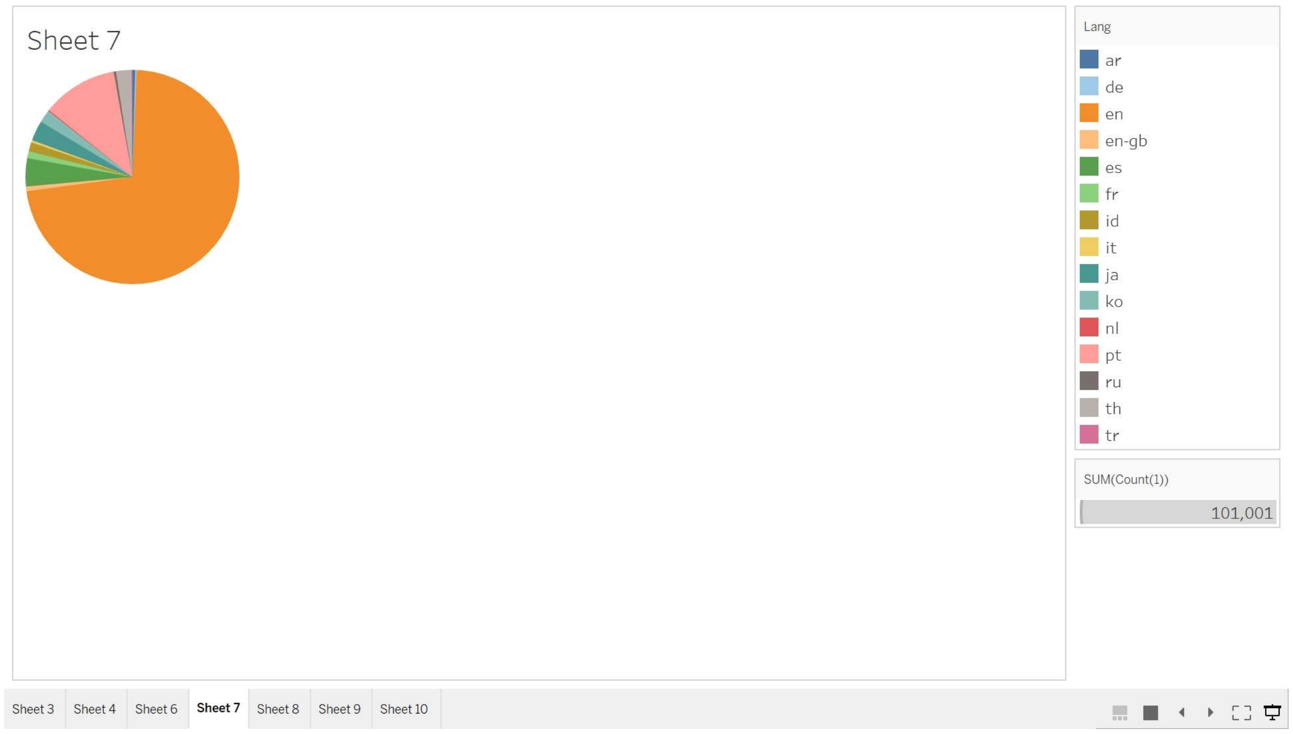
Devise which is used the most for using twitter.

Query Output:

device_used	count
Twitter for Android Tablets	3
Twitter for Windows Phone	57
Twitter for BlackBerry	9

Twitter for BlackBerryÂ®	3
Twitter for iPhone	36705
Twitter for iPad	919
Twitter for Android	22895
Twitter for Mac	17
Twitter for Windows	126

Query 3:

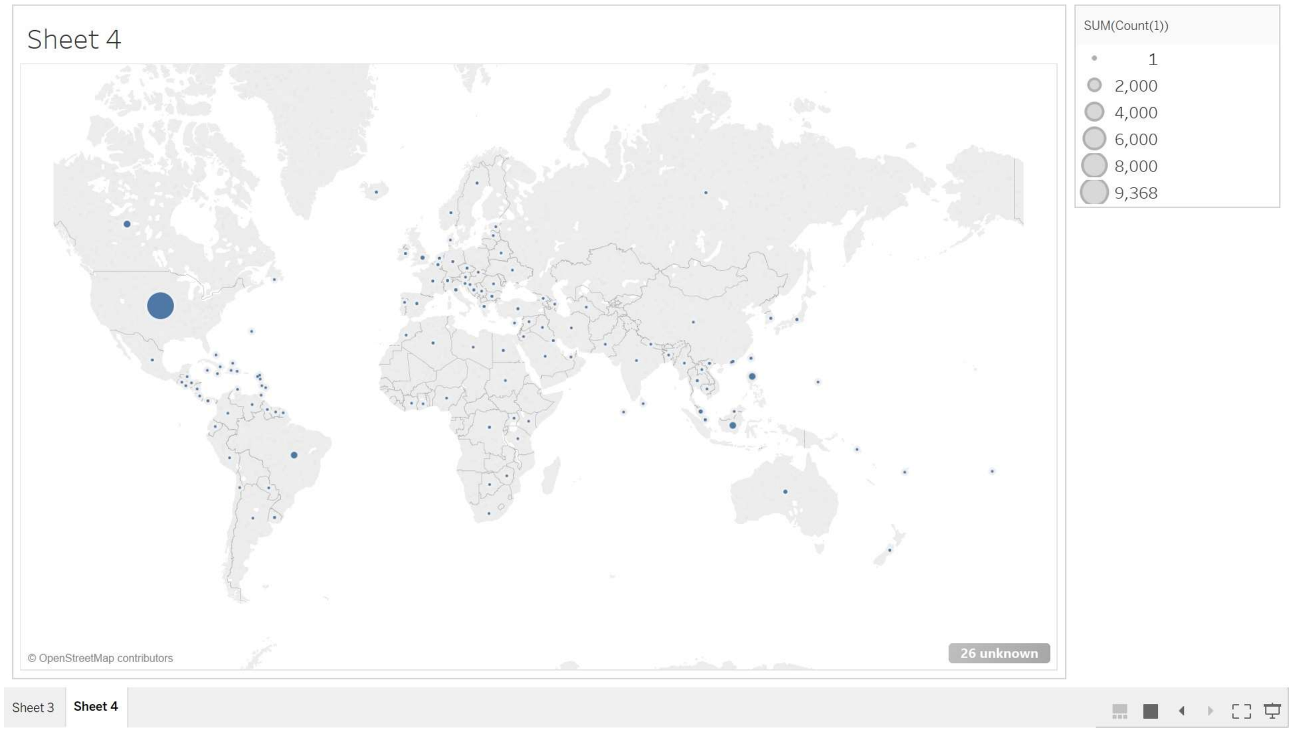


Top 10 languages used in tweets.

Query Output:

lang	count(1)
en	72962
zh-tw	103
vi	142
nb	1
ro	1
en-gb	728
ur	1
lv	1
pl	33

Query 4:

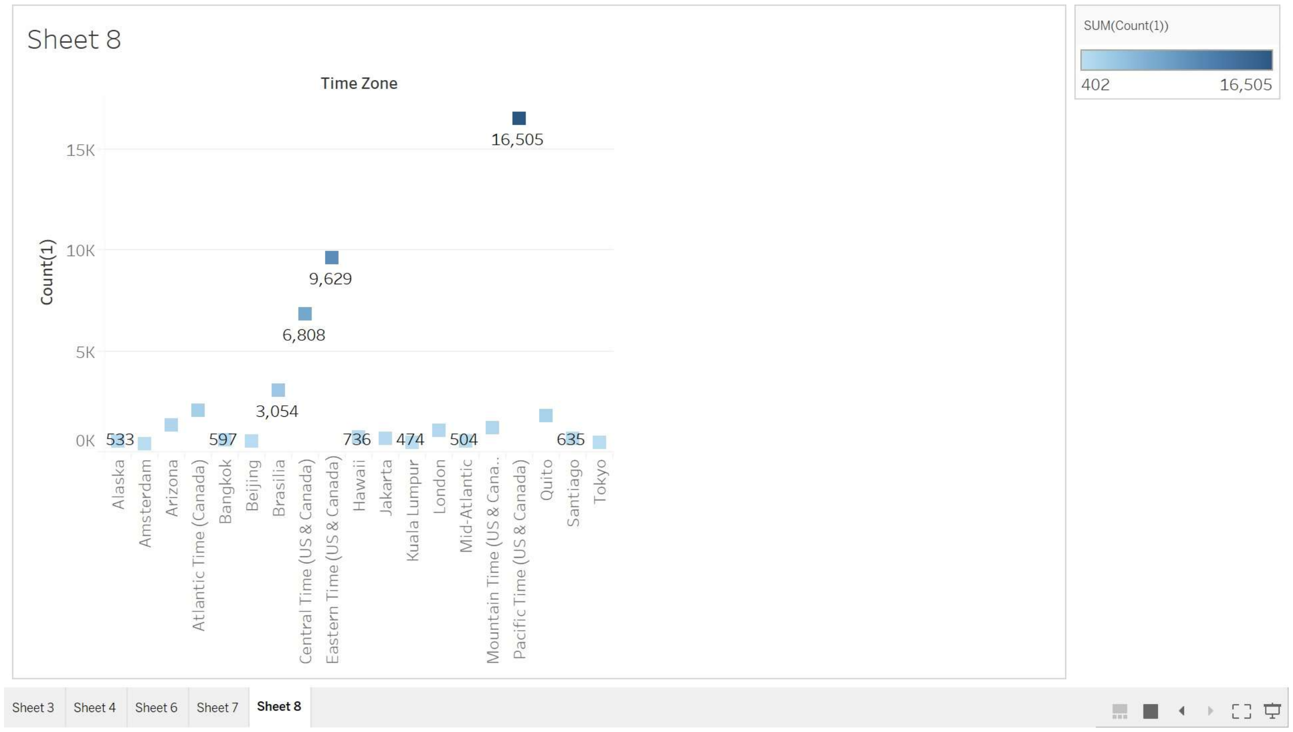


Cities from which most tweets posted.

Query Output:

country	count(1)
Russia	5
Paraguay	5
CanadÃ¡	2
Viá»‡t Nam	2
Islamic Republic of Iran	2
Sweden	3
Î•Î»Î»Î¬Ï,	6
The Netherlands	11
Schweiz	3
Republic of Korea	26
Etats-Unis	2
Guyana	1
Spanien	2
Norge	1

Query 5:



Most Popular Time Zones

Query Output:

time_zone	count(1)
Pacific Time (US & Canada)	16505
Eastern Time (US & Canada)	9629
Central Time (US & Canada)	6808
Brasilia	3054
Atlantic Time (Canada)	2085
Quito	1801
Arizona	1296
Mountain Time (US & Canada)	1204
London	1085
Hawaii	736
Jakarta	674
Santiago	635
Bangkok	597
Beijing	562

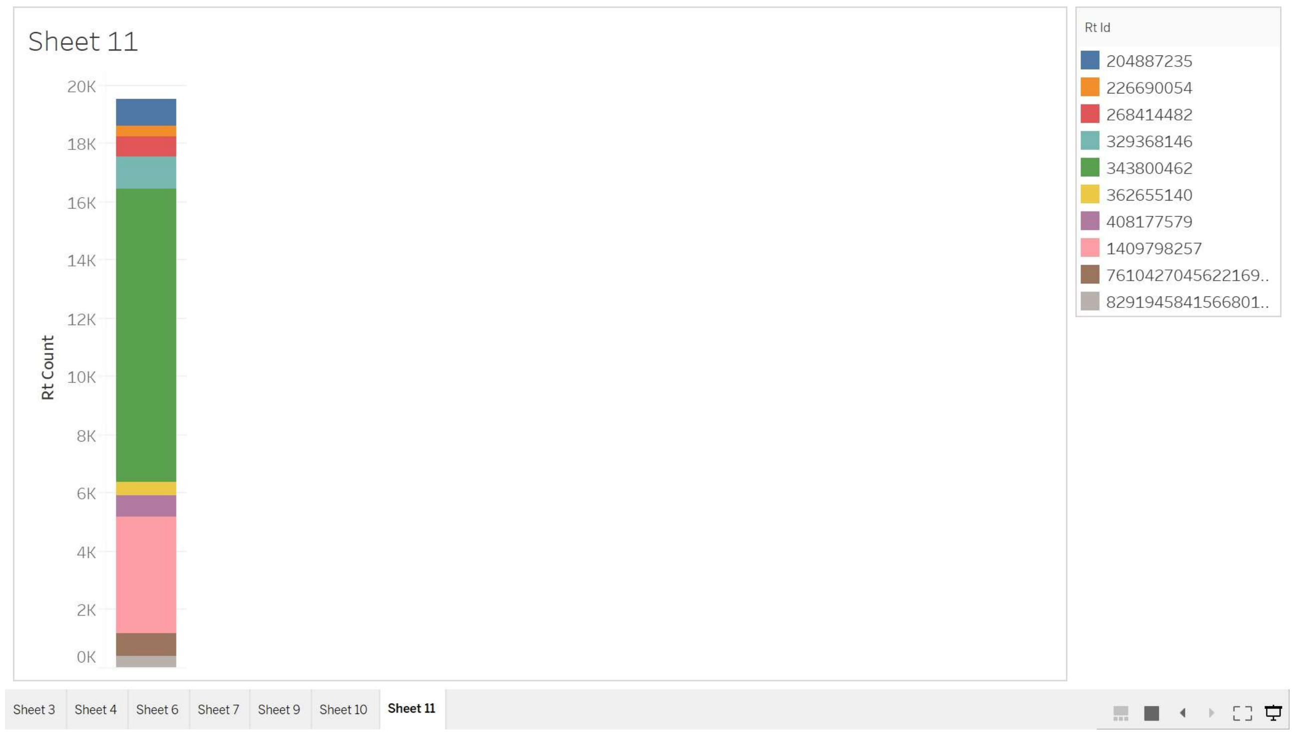
Query 6:

Trending Hashtags in twitter

Query Output:

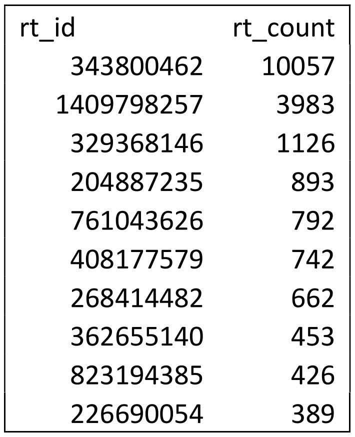
description	wordcount
life	47455
and	14494
|	13332
the	13304
of	11285
a	10754
to	8708
&	8235
I	8192
in	6831

Query 7:



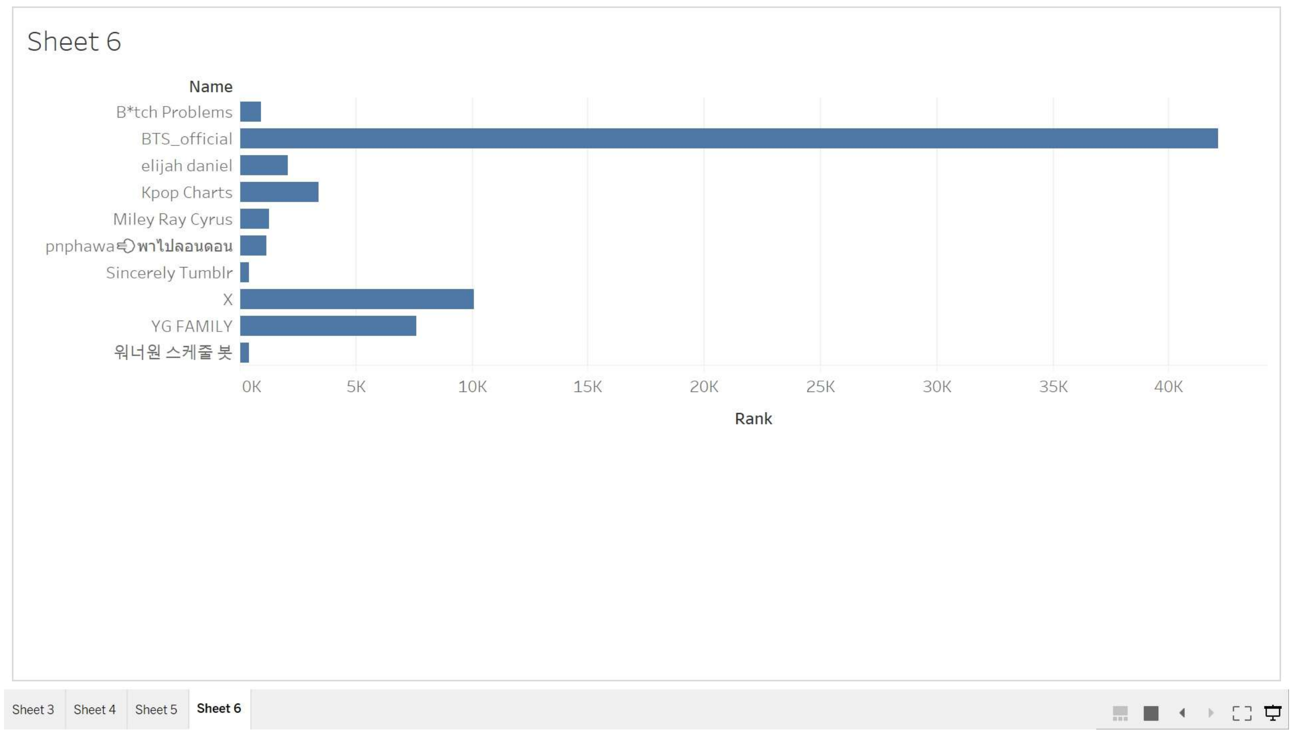
User who has retweeted most of the time

Query Output:



rt_id	rt_count
343800462	10057
1409798257	3983
329368146	1126
204887235	893
761043626	792
408177579	742
268414482	662
362655140	453
823194385	426
226690054	389

Query 8:

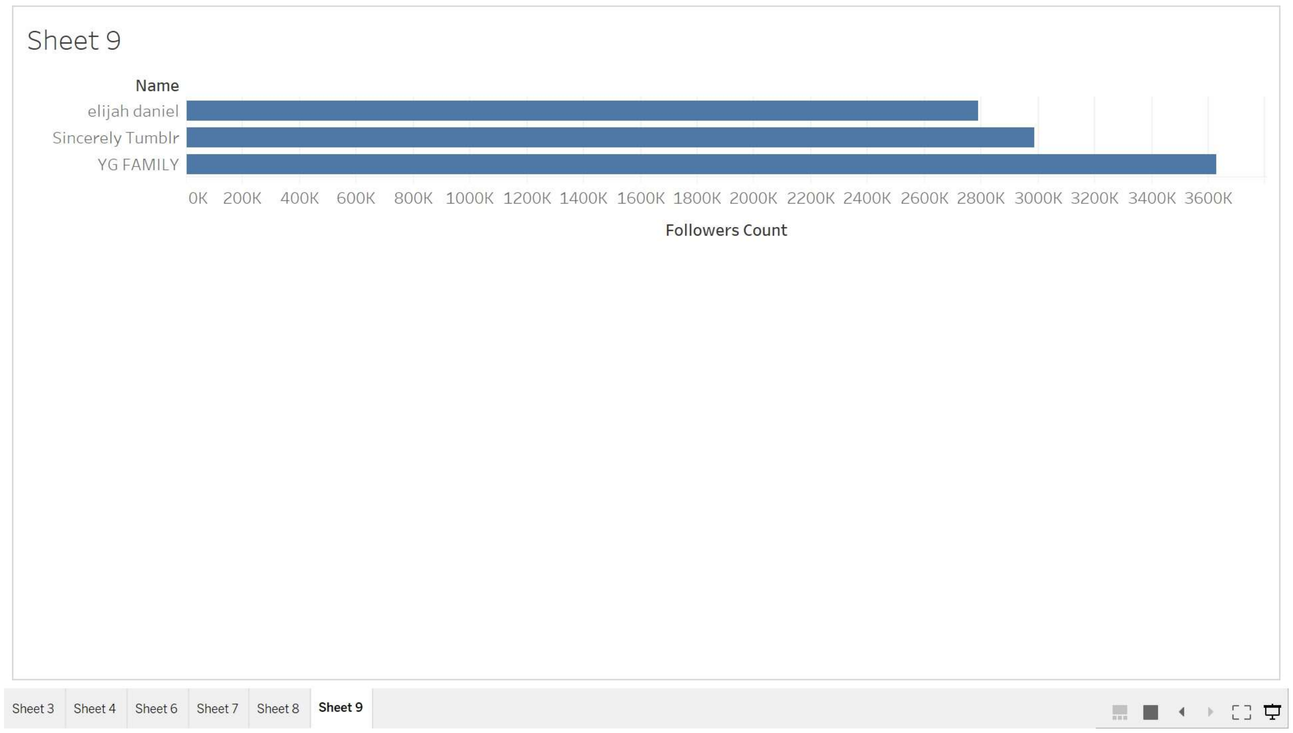


Twitter user with highest rank number

Query Output:

user_id	name	rt_time	rank
268414482	Miley Ray Cyrus	Fri Sep 29 12:58:44 +0000 2017	1
268414482	Miley Ray Cyrus	Fri Sep 29 12:58:44 +0000 2017	1
268414482	Miley Ray Cyrus	Fri Sep 29 12:58:44 +0000 2017	1
268414482	Miley Ray Cyrus	Fri Sep 29 12:58:44 +0000 2017	1
268414482	Miley Ray Cyrus	Fri Sep 29 12:58:44 +0000 2017	1

Query 9:

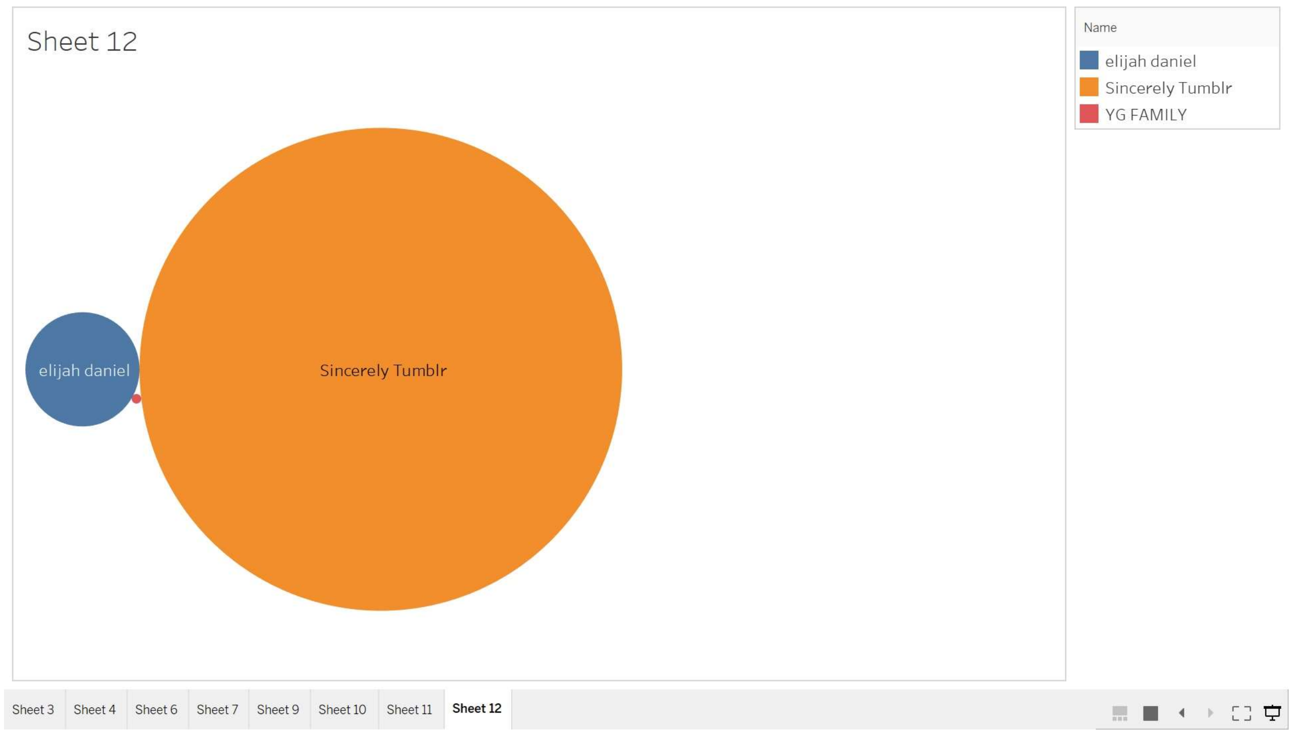


User with highest number of followers

Query Output:

id	name	location	followers_count	friends_count
226690054	Sincerely Tumblr		2987782	89406
362655140	elijah daniel	los angeles, ca	557572	1000
362655140	elijah daniel	los angeles, ca	557572	1000
362655140	elijah daniel	los angeles, ca	557500	1000
362655140	elijah daniel	los angeles, ca	557498	1000
408177579	YG FAMILY	Seoul, Korea	3625890	35
362655140	elijah daniel	los angeles, ca	557526	1000

Query 10:



User with highest number of friends count

Query Output:

id	name	location	followers_count	friends_count
226690054	Sincerely Tumblr		2987782	89406
362655140	elijah daniel	los angeles, ca	557572	1000
362655140	elijah daniel	los angeles, ca	557572	1000
362655140	elijah daniel	los angeles, ca	557500	1000
362655140	elijah daniel	los angeles, ca	557498	1000
408177579	YG FAMILY	Seoul, Korea	3625890	35
362655140	elijah daniel	los angeles, ca	557526	1000

Code:

```
import org.apache.spark.sql.{DataFrame, SparkSession}

object TwitterDataNew{

  def main(args: Array[String]): Unit = {
    val spark = SparkSession
      .builder()
      .appName("Spark SQL basic example")
        .config("spark.master","local")
      .getOrCreate()

    //val df = spark.read.json("/home/raji/twitter.json") //small file from local file system

    //val df = spark.read.json("hdfs://localhost:9000/data/tweets.json") //large file from hdfs

    df.createOrReplaceTempView("tweet_tbl")
    val user = df.select("user")
    user.createOrReplaceTempView("user_detl")

    var runProg='Y'
    while (runProg=='Y') {
      //Menu Option
      println("****** Analytical Queries using Apache Spark ******")
      println("1 => Top 10 retweeters , number of retweets ");
      println("2 => Top 10 retweeters details -- used join ")
      println("3 => Rank on retweet partition by id , rank on create date -- used window rank ")
      println("4 => Lag 3 on retweet partition by id , lag on create date -- used window lag ")
      println("5 =>Devices(iPhone,Andriod,Mac etc) used to Tweet")
      println("6 =>Tweets count from different TimeZone , country in seperate file, county_code in
seprate file ")
      println("7 =>Twees by lang , max(friendscount), avg(friendscount) , group by lang , order by
lang")
      println("8 => Get Language count grouping, and followers_count grouping ")
      println("9 => Get Hashtags wordcount  ")
      println("10 => Get Description wordcount  ")
      println("Enter your choice:")

      val choice=scala.io.StdIn.readInt()
      choice match {
        case 1 =>
          top10retweeters(spark)
          println("Press Y to continue or N to exit:")
         runProg = scala.io.StdIn.readChar()
```

```
case 2 =>
  top10rtUserDetl(spark)
  println("Press Y to continue or N to exit:")
  runProg = scala.io.StdIn.readChar()

 case 3 =>
   reTweetRankbyTime(spark)
   println("Press Y to continue or N to exit:")
   runProg = scala.io.StdIn.readChar()

case 4 =>
  reTweetLagTime(spark)
  println("Press Y to continue or N to exit:")
  runProg = scala.io.StdIn.readChar()

case 5 =>
  //1. Spark functions string,instr used
  getDeviceUsedDF(spark, df)

  println("Press Y to continue or N to exit:")

  runProg = scala.io.StdIn.readChar()



case 6 =>
  //2. group by
  getUserTimezone(spark)
  println("Press Y to continue or N to exit:")

  runProg = scala.io.StdIn.readChar()

case 7 =>
  //3 - lang, max(lang), avg(lang) group by lang, order by lang
  getlangAggr(spark)
  println("Press Y to continue or N to exit:")

  runProg = scala.io.StdIn.readChar()

case 8 =>
  getUserother(spark)
  println("Press Y to continue or N to exit:")

  runProg = scala.io.StdIn.readChar()

case 9 =>
  getHashtagWc(spark)
```

```
        println("Press Y to continue or N to exit:")
        runProg = scala.io.StdIn.readChar()

      case 10 =>
        getDescriptionWc(spark)
        println("Press Y to continue or N to exit:")
        runProg = scala.io.StdIn.readChar()
    }
  }
}

private def top10retweeters(spark: SparkSession): Unit = {

  import spark.implicits._

  val sr = spark.sql("select  retweeted_status.user.id rt_id , count(*) rt_count " +
    "from  tweet_tbl where retweeted_status.user.id is not null " +
    "group by rt_id order by rt_count desc").limit(10)

  sr.createOrReplaceTempView("top_100_rt")

  sr.coalesce(1) //single partition
    .write.format("com.databricks.spark.csv").mode("overwrite")
    .option("header", "true").save("/home/raji/proj2b/top_retweeters")
}

private def top10rtUserDetl(spark: SparkSession): Unit = {
   import spark.implicits._
   top10retweeters(spark)
   val sr = spark.sql("select user.id , user.name , user.location , user.followers_count ,
user.friends_count from " +
     " user_detl ur inner join top_100_rt rt on user.id = rt_id " )
    sr.coalesce(1) //single partition
      .write.format("com.databricks.spark.csv").mode("overwrite")
      .option("header", "true").save("/home/raji/proj2b/retweet_users_details")
}

 private def reTweetRankbyTime(spark: SparkSession): Unit = {

    import spark.implicits._
    import org.apache.spark.sql.expressions.Window

    top10retweeters(spark)
    val sr = spark.sql("select retweeted_status.user.id user_id,retweeted_status.user.name name,
retweeted_status.created_at rt_time , " +
      "dense_rank() over (partition by retweeted_status.user.id order by retweeted_status.created_at)
as rank from " +  " tweet_tbl tr inner join top_100_rt rt on retweeted_status.user.id = rt_id " )
```

```
      //val windowId = Window.partitionBy("user_id").orderBy("rt_time")
      //val wsr = sr.withColumn("rank", rank over windowId).show()
       sr.coalesce(1) //single partition
         .write.format("com.databricks.spark.csv").mode("overwrite")
         .option("header", "true").save("/home/raji/proj2b/rank")
    }

       private def reTweetLagTime(spark: SparkSession): Unit = {
          import spark.implicits._
          import org.apache.spark.sql.expressions.Window
          top10retweeters(spark)
      val sr = spark.sql("select retweeted_status.user.id user_id, retweeted_status.created_at rt_time ,
" +  "lag(retweeted_status.created_at,3) over (partition by retweeted_status.user.id order by
retweeted_status.created_at) as lag_time from " +  " tweet_tbl tr inner join top_100_rt rt on
retweeted_status.user.id = rt_id " )
           sr.coalesce(1) //single partition
             .write.format("com.databricks.spark.csv") .mode("overwrite")
             .option("header", "true").save("/home/raji/proj2b/lag")
       }

  private def getUserother(spark: SparkSession): Unit = {
    import spark.implicits._
    val sr = spark.sql("select  user.lang , count(*) from user_detl where user.lang is not null " +
      "group by user.lang")

    //sr.printSchema()
    //sr.show()
    sr.coalesce(1) //single partition
      .write.format("com.databricks.spark.csv").mode("overwrite")
      .option("header", "true").save("/home/raji/proj2b/lang")

    val fl = spark.sql("select  user.followers_count , count(*) from user_detl where
user.followers_count is not null " +
      "group by user.followers_count")
    //sr.printSchema()
    //sr.show()
    fl.coalesce(1) //single partition
      .write.format("com.databricks.spark.csv").mode("overwrite")
      .option("header", "true").save("/home/raji/proj2b/followers")
  }
  //group by
  private def getUserTimezone(spark: SparkSession): Unit = {

    import spark.implicits._
    val sr = spark.sql("select  user.time_zone , count(*) from user_detl where user.time_zone is not
null " +
      "group by user.time_zone")
    //sr.printSchema()
    //sr.show()
    sr.coalesce(1) //single partition
      .write.format("com.databricks.spark.csv").mode("overwrite")
      .option("header", "true").save("/home/raji/proj2b/usr_timezone")

    val src = spark.sql("select  place.country as country, count(*) from tweet_tbl where place.country
is not null " +
      "group by place.country")
    //sr.printSchema()
    //sr.show()
    src.coalesce(1) //single partition
      .write.format("com.databricks.spark.csv").mode("overwrite")
      .option("header", "true").save("/home/raji/proj2b/country")

    val srcd = spark.sql("select  place.country_code as country_code, count(*) from tweet_tbl where
place.country_code is not null " +
      "group by place.country_code")
    //sr.printSchema()
    //sr.show()
    srcd.coalesce(1) //single partition
      .write.format("com.databricks.spark.csv").mode("overwrite")
      .option("header", "true").save("/home/raji/proj2b/country_code")
  }

  private def getlangAggr(spark: SparkSession): Unit = {
    import spark.implicits._
    /*val sr = spark.sql("select  user.id, user.friends_count from user_detl where user.description is
not null " )
    //sr.printSchema()
    //sr.show()
    sr.coalesce(1) //single partition
      .write.format("com.databricks.spark.csv")
      .option("header", "true").save("/home/raji/proj2b/description")*/

    val sr = spark.sql("select user.lang, avg(user.friends_count) as avgf, max(user.friends_count) as
maxf " +  "from user_detl where user.lang is not null group by user.lang order by user.lang")
    sr.coalesce(1) //single partition
      .write.format("com.databricks.spark.csv").mode("overwrite")
      .option("header", "true").save("/home/raji/proj2b/aggr")
  }

  //substring,instr
  private def getDeviceUsedDF(spark: SparkSession, df : DataFrame): Unit = {
    import spark.implicits._

    println("getDeviceUsedDF")
    val sr = df.withColumn("source", 'source.cast("string")).select("source")
    sr.createOrReplaceTempView("source_table")
    //sr.show()

    //val sr1 = spark.sql("select substring(source.indexOf(\">\")+1 ,source.indexOf(\"</a>\") ) as v
from source_table")
    //val sr1 = spark.sql("select substring(source,  (instr(source, \">\" ) +1 ), (instr(source, \"</a\" ) -
10) ) as device_used " +
    //  " from source_table where source like '%Twitter for%'  ")

    val sr1 = spark.sql("select substring(source,  instr(source, \">\" ) +1 , instr(source, \"</a\" ) -
instr(source, \">\" ) - 1 ) as device_used " +
      " from source_table where source like '%Twitter for%'  ")
    //sr1.show()
    sr1.createOrReplaceTempView("device_table")

    val ts3 = spark.sql("SELECT device_used , count(*) as count from device_table group by
device_used")
    //ts3.show()
    ts3.coalesce(1) //single partition
      .write.format("com.databricks.spark.csv").mode("overwrite")
      .option("header", "true").save("/home/raji/proj2b/deviceused")
  }

  private def getHashtagWc(spark: SparkSession): Unit = {

    import spark.implicits._
    import org.apache.spark.sql.functions._

    val hs = spark.sql("select  cast(entities.hashtags.text as string) as text from tweet_tbl " )
    hs.createTempView("hashTags")
    val sr = spark.sql("select  substring(text,  2 , instr(text, \"]\" ) - 2 ) as hastag_text  from hashTags
")
    val hsd = sr.withColumn("hastag_text", explode(split($"hastag_text", "[,]")))
    hsd.createTempView("hashtags_tbl")
    val src = spark.sql("select  hastag_text , count(*) wordcount from hashtags_tbl where hastag_text
is not null " +
     "group by hastag_text order by wordcount desc")
    src.show()
    src.coalesce(1) //single partition
      .write.format("com.databricks.spark.csv").mode("overwrite")
      .option("header", "true").save("/home/raji/proj2b/hashtags_wc")
  }

  private def getDescriptionWc(spark: SparkSession): Unit = {
    import spark.implicits._
    import org.apache.spark.sql.functions._
    val hs = spark.sql("select  user.description   as description from user_detl " )
    val hsd = hs.withColumn("description", explode(split($"description", "[ ]")))
    hsd.createTempView("Desc_tbl")
    val src = spark.sql("select  description , count(*) wordcount from Desc_tbl where description is
not null " +  "group by description order by wordcount desc")
    src.show()
    src.coalesce(1) //single partition
      .write.format("com.databricks.spark.csv").mode("overwrite")
      .option("header", "true").save("/home/raji/proj2b/description")
  }
}
```