Assignment #4

Question 1: Uber wants to improve their cab dispatching service. One useful analysis is to find the center of busy area and peak hours of cab requests. From Blackboard, download the datasets containing six months (April to September in 2014) of Uber cab pickup data. For each month:

- Apply K-means clustering method to help Uber establish 8 optimal pickup locations
- Find out the peak hours of cab requests for each pickup location

```
// Import packages
import org.apache.spark.sql.types._
import org.apache.spark.storage.StorageLevel
import scala.io.Source
import scala.collection.mutable.HashMap
import java.io.File
import org.apache.spark.sql.Row
import org.apache.spark.sql.types._
import scala.collection.mutable.ListBuffer
import org.apache.spark.util.IntParam
import org.apache.spark.util.StatCounter
import org.apache.spark.rdd.RDD
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.sql.SQLContext
import org.apache.spark.rdd._
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors

val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext.implicits._
import sqlContext._

// Create schema for data input
val schema = StructType(Array(
        StructField("dateTime", StringType, false),
        StructField("lat", FloatType, false),
        StructField("long", FloatType, false),
        StructField("base", StringType, false)))

// Import csv files
val df1 = spark.read.option("header", "true").schema(schema).csv("uber-raw-data-
    apr14.csv")
val df2 = spark.read.option("header", "true").schema(schema).csv("uber-raw-data-
    may14.csv")
val df3 = spark.read.option("header", "true").schema(schema).csv("uber-raw-data-
    jun14.csv")
val df4 = spark.read.option("header", "true").schema(schema).csv("uber-raw-data-
    jul14.csv")
val df5 = spark.read.option("header", "true").schema(schema).csv("uber-raw-data-
    aug14.csv")
val df6 = spark.read.option("header", "true").schema(schema).csv("uber-raw-data-
    sep14.csv")
```

```
// Create arrays of dataframes, months and feature column
val dfs = Array(df1, df2, df3, df4, df5, df6)
val months = Array("April", "May", "June", "July", "August", "September")
val featureCols = Array("lat", "long")

// Iterate over all 6 months
for(i <- 0 to 5){

        // Perform K-means clustering to find 8 optimal locations for a particular
            month
        val rows = new VectorAssembler().setInputCols(featureCols).setOutputCol("
            features").transform(dfs(i))
        val k = new org.apache.spark.ml.clustering.KMeans().setK(8).setFeaturesCol("
            features").setPredictionCol("location")
        val model = k.fit(rows)

        // Print the month
        val m = months(i)
        println(s"Month: $m")

        // Put all locations into a sequence
        var cc: Seq[(Float, Float)] = Seq()
        for(e <- model.clusterCenters){
                cc = cc :+ ((e(0).toFloat), e(1).toFloat)
        }

        // Create a dataframe for the cluster centroids
        val s = Seq("lat", "long")
        var ccDF = cc.toDF(s: _*)

        // Round the coordinates to 4 digits for consistency
        var ccRounded = ccDF.select(round($"lat", 4).alias("lat"), round($"long", 4)
            .alias("long"))
        ccRounded.createOrReplaceTempView("ccView")
        var dfRounded = dfs(i).select(round($"lat", 4).alias("lat"), round($"long",
            4).alias("long"), $"dateTime")
        dfRounded.createOrReplaceTempView("dfView")

        // Get the hour from the date column
        val ts = to_timestamp($"dateTime", "MM/dd/yyyy HH:mm:ss").alias("time")
        val dfTime = dfRounded.select($"lat", $"long", ts)
        dfTime.createOrReplaceTempView("dfTimeView")
        val dfHour = spark.sql("Select lat, long, HOUR(time) as hr from dfTimeView")
        dfHour.createOrReplaceTempView("dfView")

        // Find all records that occur at a centroid coordinate
        val tog = spark.sql("Select ccView.lat as lat, ccView.long as long, dfView.
            hr as hr from ccView INNER JOIN dfView on dfView.lat = ccView.lat")
        tog.createOrReplaceTempView("joined")

        // List all coordinates and the hour it was requested
        val distinct = spark.sql("Select DISTINCT lat, long, hr from joined group by
             lat, long, hr order by lat, long, hr")
```

```
        // Turn the dataframe into an RDD
        val d: RDD[Row] = distinct.rdd

        // Sort the rdd by coordinates
        val dKV = d.keyBy(line => (line(0), line(1))).mapValues(line => line(2)).
            groupByKey()

        // Print each coordinate and hours
        println("Below is the Latitude, Longitude and list of Peak Hours")
        dKV.foreach(println)
        println("\n\n")
}
```

Question 2: Write a Spark program to find out the live popular tweets using Spark Streaming.

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.twitter._
import org.apache.spark.SparkConf
import org.apache.log4j.{Level, Logger}
object HWQ2 {
  def main(args: Array[String]) {

    Logger.getLogger("org").setLevel(Level.ERROR)
    var consumerKey = "PttEZFyQIhrxfqmIBfZ9x5fKc"
    var consumerSecret = "RyCgiLKqP8kQjcwxJ8h9EQFzLGm3dL5n2eCTN9YpQ2RRYG3cd7"
    var accessToken = "931749427211128832-UJP8jUVAEieK0fP9mmHn5yuD4DiGi8M"
    var accessTokenSecret = "TAuHxwpL6FghEom8IDUtQTQPUeHik6nxhjmhzvyGlfGUk"

  if (args.length > 3) {
    val Array(consumerKey, consumerSecret, accessToken, accessTokenSecret) = args.
        take(4)
    }

    System.setProperty("twitter4j.oauth.consumerKey", consumerKey)
    System.setProperty("twitter4j.oauth.consumerSecret", consumerSecret)
    System.setProperty("twitter4j.oauth.accessToken", accessToken)
    System.setProperty("twitter4j.oauth.accessTokenSecret", accessTokenSecret)

    val sparkConf = new SparkConf().setAppName("Twitter").setMaster("local[2]")
    val ssc = new StreamingContext(sparkConf, Seconds(3))
    val stream = TwitterUtils.createStream(ssc, None)

    val statuses = stream.map(status => status.getText())
    statuses.print()

    ssc.start()
    ssc.awaitTermination()
  }
}
```

Question 3: You are given two datasets:

Student: This dataset contains name and roll number of students in a class.
Results: This dataset contains roll number and result (Fail or Pass) of student.

Write a Pig script to analyze the given datasets and print the student names who have successfully cleared the exam. Use only PIG commands.

```
studentList = load 'student' USING PigStorage('\t') AS (name: chararray, ID: int);
resultList = load 'results' USING PigStorage('\t') AS (ID: int, grade: chararray);
joinedList = join studentList by $1, resultList by $0;
studentGrades = foreach joinedList generate name, grade;
stupendousStudents = filter studentGrades by grade matches 'pass';
dump stupendousStudents;
```