Citi Bike NYC

By,

Sushmanth Sagala

Spark Project – UpX Academy

Project Information

- **Domain:** Travel
- **Technology use:** Spark, Spark SQL, Spark MLlib
- **Dataset:** https://s3.amazonaws.com/tripdata/index.html
- Citi Bike is the nation's largest bike share program, with 10,000 bikes and 600 stations across Manhattan, Brooklyn, Queens and Jersey City. Analyze this data to find the interesting patterns in the traffic.

Data Description

- Trip Duration (seconds)
- Start Time and Date
- Stop Time and Date
- Start Station Name
- End Station Name
- Station ID
- Station Lat/Long
- Bike ID
- User Type (Customer = 24-hour pass or 3-day pass user; Subscriber = Annual Member)
- Gender (Zero=unknown; 1=male; 2=female)
- Year of Birth

Business Questions

- **Spark Core / Spark SQL**
  - Which route Citi Bikers ride the most?
  - Find the biggest trip and its duration?
  - When do they ride?
  - How far do they go?
  - Which stations are most popular?
  - Which days of the week are most rides taken on?
- **Spark MLLIB**
  - Predicting the gender based on patterns in the trip

Data Loading

- Used com.databricks.spark library to load CSV data.
- val bike = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true").option("delimiter", ",").load("/user/cloudera/Project/CitiBikeNYC/* - Citi Bike trip data.csv")
- Parsed to rename the columns and register the Dataframe as a Temporary Table.
- val bikeDF = bike.toDF("trip_duration", "start_time", "stop_time", "start_station_id", "start_station_name", "start_station_latitude", "start_station_longitude", "end_station_id", "end_station_name", "end_station_latitude", "end_station_longitude", "bike_id", "user_type", "birth_year", "gender")
- bikeDF.registerTempTable("bike")

Business Questions Answered

- Q1. Which route Citi Bikers ride the most?
  - To determine the route Citi Bikers ride the most.
  - Grouped the Start and End station combination to find the aggregate count and fetching the route with highest rides taken on.
  - sqlContext.sql("SELECT start_station_id, start_station_name, end_station_id, end_station_name, COUNT(1) AS cnt FROM bike GROUP BY start_station_id, start_station_name, end_station_id, end_station_name ORDER BY cnt DESC LIMIT 1")

- Q2. Find the biggest trip and its duration?
  - To Determine the biggest trip, analyzed trip duration and distance of each trip.
  - Distance of each trip calculated based on distance between station location (Latitude and Longitude).
  - Defined an UDF function calcDist to calculate the distance between locations.

Business Questions Answered

- def calcDist(lat1: Double, lon1: Double,lat2: Double, lon2: Double): Int = {

  val AVERAGE_RADIUS_OF_EARTH_KM = 6371

  val latDistance = Math.toRadians(lat1 - lat2)

  val lngDistance = Math.toRadians(lon1 - lon2)

  val sinLat = Math.sin(latDistance / 2)

  val sinLng = Math.sin(lngDistance / 2)

  val a = sinLat * sinLat + (Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) * sinLng * sinLng)

  val c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a))
  (AVERAGE_RADIUS_OF_EARTH_KM * c).toInt }
- sqlContext.udf.register("calcDist", calcDist _)
- Ordered by the trip duration and distance to find the biggest trip.
- sqlContext.sql("SELECT start_station_id, start_station_name, end_station_id, end_station_name, trip_duration, calcDist(start_station_latitude, start_station_longitude, end_station_latitude, end_station_longitude) AS dist FROM bike ORDER BY trip_duration DESC, dist DESC LIMIT 1")

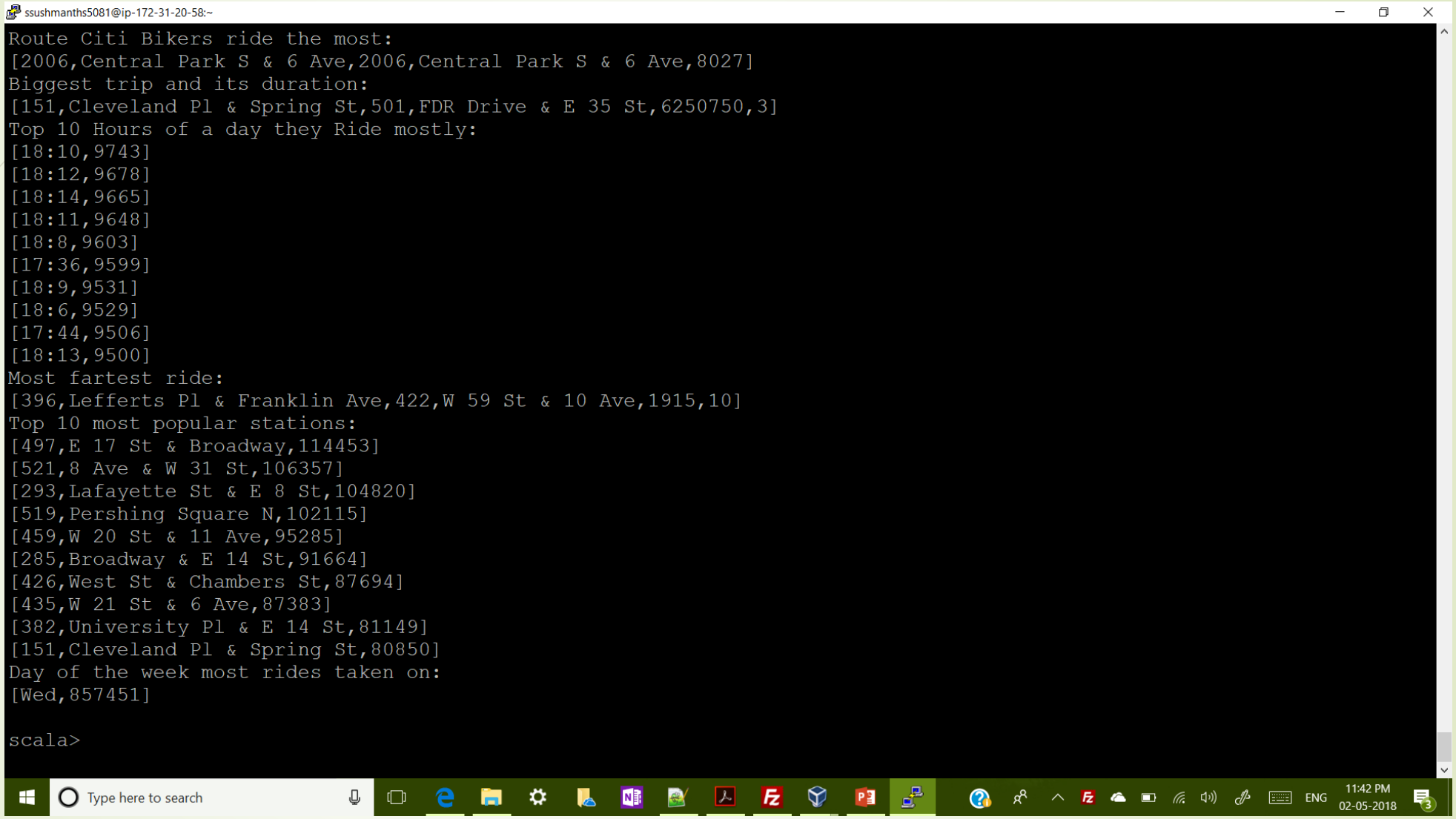
Business Questions Answered

- Q3. When do they ride?
  - To determine the top 10 most preferred time for rides.
  - Grouped the Start time based on Hours and Minutes to aggregate the number of rides taken at respective time and fetching the top 10 most preferred Start time by riders.
  - sqlContext.sql("SELECT date_format(start_time,'H:m'), COUNT(1) as cnt FROM bike GROUP BY date_format(start_time,'H:m') ORDER BY cnt DESC LIMIT 10")

- Q4. How far do they go?
  - To determine the farthest ride ever taken.
  - Calculated the distance between Start and End stations based on the UDF function calcDist created and fetched the ride with longest distance.
  - sqlContext.sql("SELECT start_station_id, start_station_name, end_station_id, end_station_name, trip_duration, calcDist(start_station_latitude, start_station_longitude, end_station_latitude, end_station_longitude) AS dist FROM bike ORDER BY dist DESC LIMIT 1")

Business Questions Answered

- Q5. Which stations are most popular?
  - To determine the top 10 most popular stations among the riders.
  - To get the count aggregations of each station, used union to get the overall station list irrespective of Start or End station and fetch the top 10 stations.
  - var popularStationsDF = sqlContext.sql("SELECT start_station_id as station_id, start_station_name as station_name, COUNT(1) as cnt FROM bike GROUP BY start_station_id, start_station_name UNION ALL SELECT end_station_id as station_id, end_station_name as station_name, COUNT(1) as cnt FROM bike GROUP BY end_station_id, end_station_name")
  - popularStationsDF.registerTempTable("popularStations")
  - sqlContext.sql("SELECT station_id as station_id,station_name, SUM(cnt) as total FROM popularStations GROUP BY station_id,station_name ORDER BY total DESC LIMIT 10")
- Q6. Which days of the week are most rides taken on?
  - To determine the day of the week when most of the rides are taken.
  - Grouping each day of week to aggregate the total rides and fetch the day with highest rides taken on.
  - sqlContext.sql("SELECT date_format(start_time,'E') AS Day, COUNT(1) AS cnt FROM bike GROUP BY date_format(start_time,'E') ORDER BY cnt DESC LIMIT 1")



Sample output screenshot

Q7. Predicting the gender based on patterns in the trip

- Building a model using Logistic Regression to predict the gender based on patterns in the trip.
- Converting the bikeDF Dataframe to RDD so as to apply the data to model.
  - val bikeRDD = bikeDF.rdd
- To define the Label and Features, extracted the features based on ride patterns.
  - val features = bikeRDD.map {

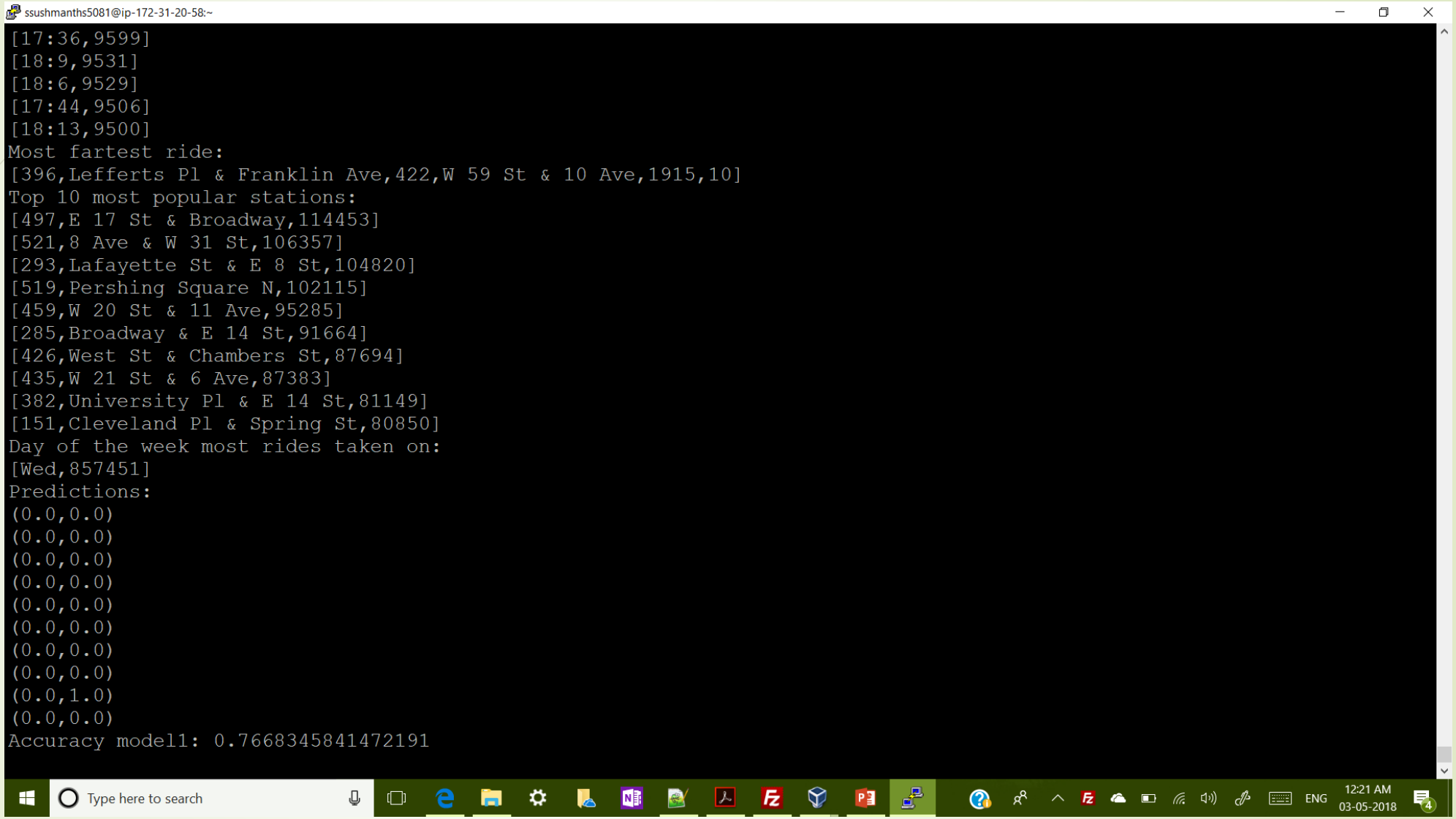
```
bike => val gender = if (bike.getInt(14) == 1) 0.0 else if (bike.getInt(14) == 2) 1.0 else  2.0
val trip_duration = bike.getInt(0).toDouble
val start_time = bike.getTimestamp(1).getTime.toDouble
val stop_time = bike.getTimestamp(2).getTime.toDouble
val start_station_id = bike.getInt(3).toDouble
val start_station_latitude = bike.getDouble(5)
val start_station_longitude = bike.getDouble(6)
val end_station_id = bike.getInt(7).toDouble
val end_station_latitude = bike.getDouble(9)
val end_station_longitude = bike.getDouble(10)
val user_type = if (bike.getString(12) == "Customer") 1.0 else 2.0
Array(gender, trip_duration, start_time, stop_time, start_station_id, start_station_latitude,
start_station_longitude, end_station_id, end_station_latitude, end_station_longitude, user_type)    }
```

Q7. Predicting the gender based on patterns in the trip

- Gender is used as label and other relevant columns are used as features to form the Labeled Vector.
  - val labeled = features.map { x => LabeledPoint(x(0), Vectors.dense(x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9), x(10)))}
- Splitting the data for Tests and Training respectively.
  - val training = labeled.filter(_.label != 2).randomSplit(Array(0.60, 0.40))(1)
    val test = labeled.filter(_.label != 2).randomSplit(Array(0.40, 0.60))(1)
- Running the training algorithm to build the predicting model. Cached training set is passed to the training algorithm.
- Number of classes is set to 2.
  - val model = new LogisticRegressionWithLBFGS().setNumClasses(2).run(training)

Q7. Predicting the gender based on patterns in the trip

- Computing the predictions on the Testing data.
  - val predictionAndLabels = test.map {
    case LabeledPoint(label, features) =>
    val prediction = model.predict(features)
    (prediction, label) }
- To determine the accuracy of the built model.
- Fetching the count of Wrong predictions found based on label and prediction values.
- With the help of total test count and wrong counts accuracy is determined.
  - val wrong = predictionAndLabels.filter {
    case (label, prediction) => label != prediction }
  - val wrong_count = wrong.count
  - val accuracy = 1 - (wrong_count.toDouble / test_count)



Sample output screenshot

Conclusion

- Citi Bike NYC data loaded and analysed successfully.
- CSV data loaded as Dataframe to answer the business questions using Spark SQL.
- Built an Mllib Logistic Regression model to predict the gender of each rider based on patterns in the trip.
- Code: https://github.com/ssushmanth/CitiBikeNYC-spark