City of Chicago

TAXI FAIR PREDICTION

STUDENT B

12/11/2018

Project Overview:

Why we are doing this project ?

Many taxi companies are struggling to stay in the business due to effective pricing models of ride share companies like Uber & Lyft

Goal: Use City of Chicago Taxi trip dataset to build a more competitive pricing model (fare) based on trip_seconds & trip_miles.

ROI: Better profit to Taxi companies & Drivers

Dataset Overview:

Key Insights:

- 1-year data is distributed in 12 files ,one each for every month
- More than 1 million records in each file
- Have 20 key attributes to explain the data.

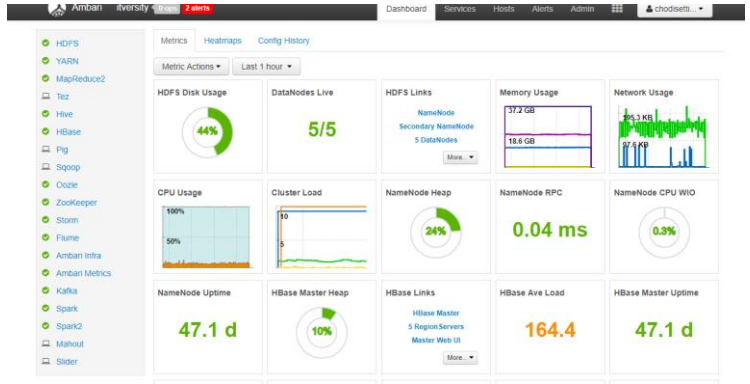
Challenges:

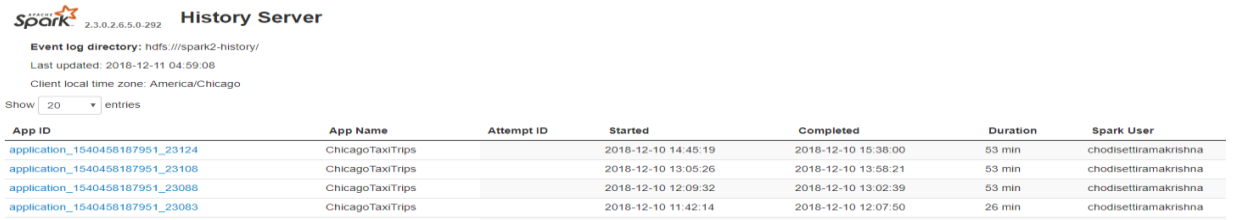
- Loading time due to the data volume
- Data Cleaning
- Data analysis due to the data volume.
- Training time.

taxi_id
trip_start_timestamp
trip_end_timestamp
trip_seconds
trip_miles
pickup_census_tract
dropoff_census_tract
pickup_community_area
dropoff_community_area
fare
tips
tolls
extras
trip_total
payment_type
company
pickup_latitude
pickup_longitude
dropoff_latitude
dropoff_longitude

Infrastructure

- ITVersity Lab Big data systems
- Spark Job History Server UI
- Putty to submit spark jobs
- Winscp to deploy code
- Ambari File view to upload dataset files to hadoop cluster.
- Achieved faster run times to train models
- Run time – Around 30 mins to train all models





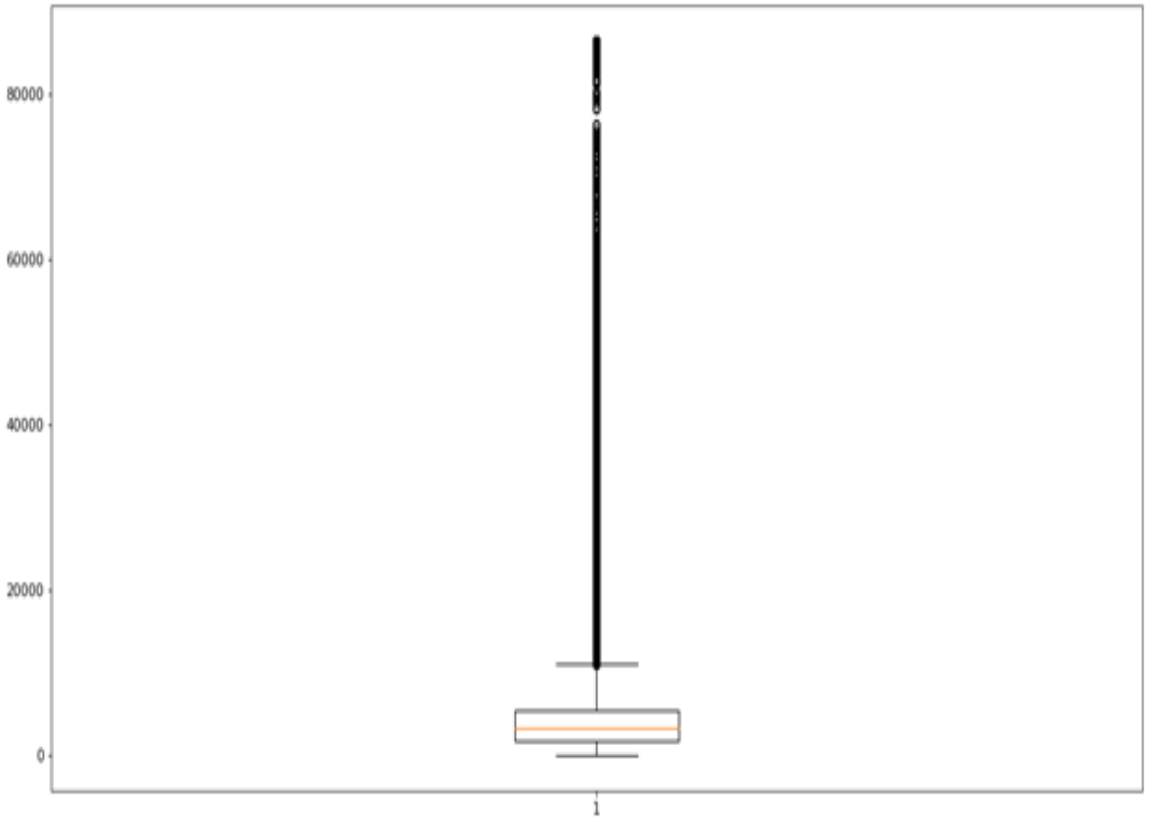
.

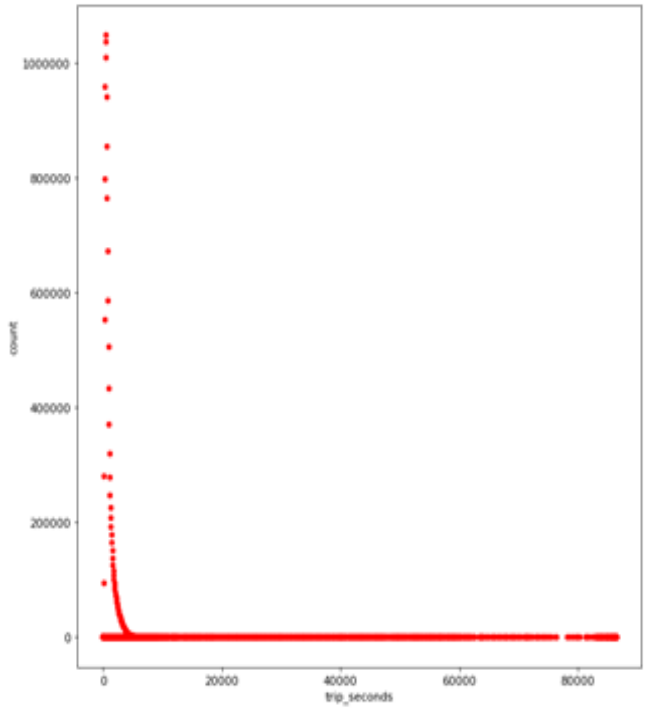
spark-submit --master yarn --deploy-mode client --conf spark.ui.port=50865 --conf spark.dynamicAllocation.enabled=false --num-executors 8 --executor-memory 1536M --executor-cores 2 FinalProject2.py

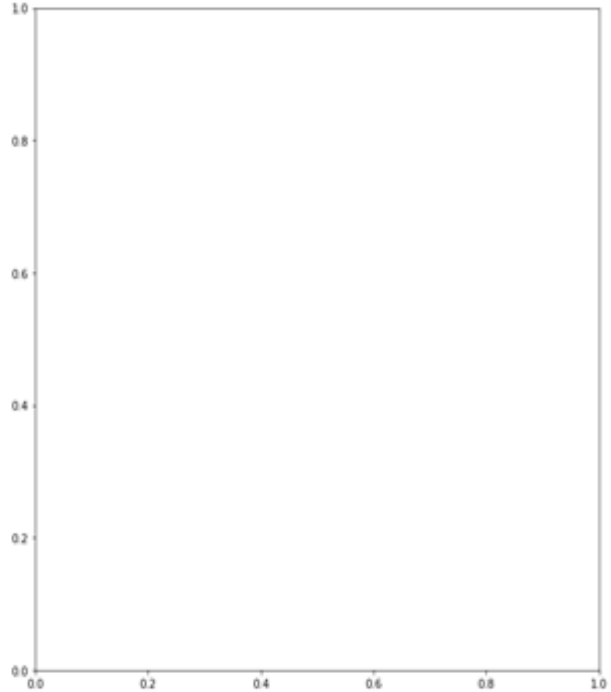
Data Analysis:

- Dropped null values and zero values to avoid bias in the results.
- Outlier analysis on trip_seconds, trip_miles using box plot and scatter plot.
- Identified trip_seconds are less than trip_miles.
- Considered trip miles range 0 to 2000
- Considered trip_seconds range 0 to 60000 seconds

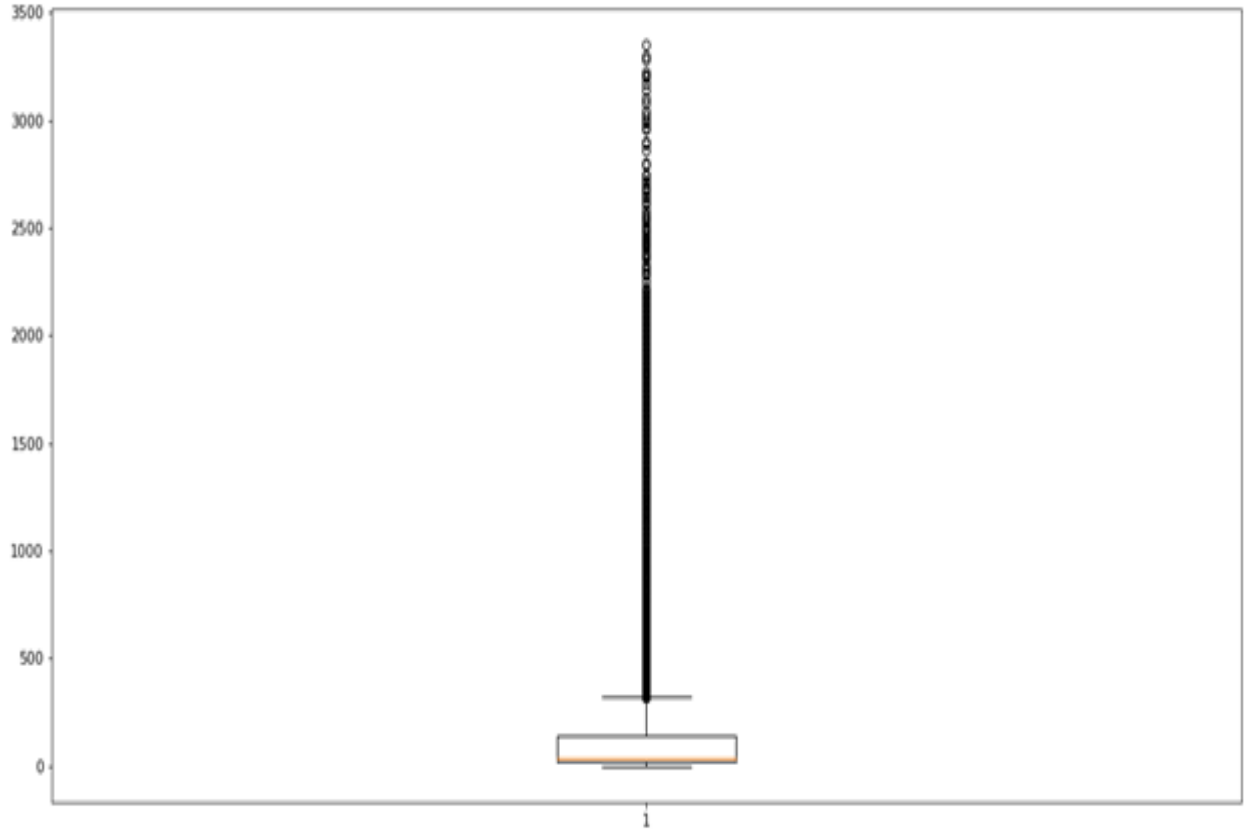
Data analysis- trip_seconds

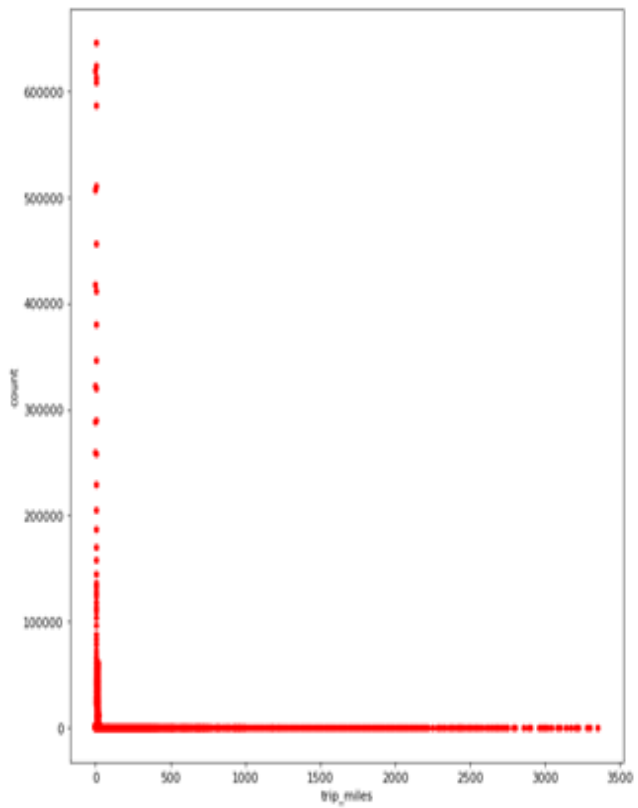


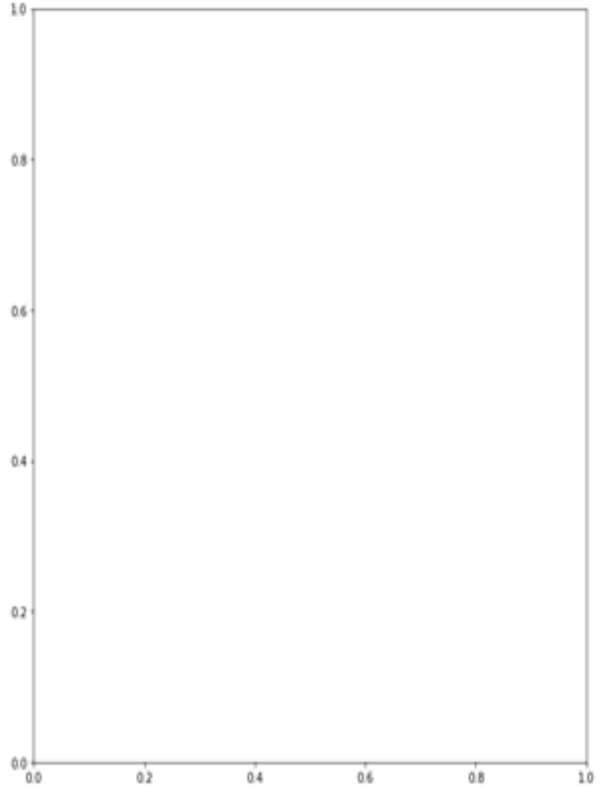




Data analysis- trip_miles







Model Analysis with different run conditions

- Run 1 - Filtered Zeros , null values
- Run 2- Filtered Zeros , null values, trip_miles < 2000,trip_seconds < 60000
- Run 3- Filtered Zeros , null values, trip_seconds >trip_miles, trip_miles < 2000,trip_seconds < 60000

Model 1 :Elastic net

- Mix of Lasso & Ridge regression
- Objective is to find the best regularization parameters to reduce the SSE of Elastic Net

$$\text{minimize}\left\{SSE + \lambda_1 \sum_{j=1}^{p} \beta_j^2 + \lambda_2 \sum_{j=1}^{p} |\beta_j|\right\}$$

- When lambda 1 and 2=0 then we get least square parameter estimate.
- When lambda 1 >0 and lambda =0 then it is Lasso Regression.
- When lambda =0 and lambda>0 then it is Ridge regression.

RMSE		
RUN 1	RUN 2	RUN 3
19.3432	19.1767	19.1767

```
#*******************************************************************************
##Mean square calculation -Linear regression
#*******************************************************************************

lm = LinearRegression(featuresCol = 'features', labelCol='label',
                      maxIter=10, regParam=1.0, elasticNetParam=0.5)

lm_model = lm.fit(train)
print("Coefficients: " + str(lm_model.coefficients))
print("Intercept: " + str(lm_model.intercept))
trainingSummary = lm_model.summary
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)

lm_predictions = lm_model.transform(test)
lm_predictions.select("prediction","label","features").show(10)

lr_evaluator = RegressionEvaluator(predictionCol="prediction", \
                 labelCol="label",metricName="r2")
print("R Squared (R2) on test data = %g" % lr_evaluator.evaluate(lm_predictions))
test_result = lm_model.evaluate(test)

print("Root Mean Squared Error (RMSE) on test data = %g" % test_result.rootMeanSquaredError)
lm_predictions = lm_model.transform(test)
lm_predictions.select("prediction","label","features").show(5)
```

MODEL 2: SIMPLE TREE MODEL

- In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.
- It works for both categorical and continuous input and output variables.

```
#*******************************************************************************
###Decision tree
#*******************************************************************************
print("Decision tree Started")

dt_taxiTrips_Tree = DecisionTreeRegressor(featuresCol = 'features', labelCol = 'label', maxDepth = 3)
decision_Model = dt_taxiTrips_Tree.fit(train)
predictions = decision_Model.transform(test)
predictions.show()

dt_evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = dt_evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

RMSE		
RUN 1	RUN 2	RUN 3
17.8708	17.9874	17.9874

MODEL 3: RANDOM FOREST

Random forests or **random decision forests** are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees to add text

```
#*************************************************************************************
####RandomForest
#*************************************************************************************

Taxi_rf = RandomForestRegressor(featuresCol = 'features', labelCol = 'label')
randomforestModel = Taxi_rf.fit(train)
predictions = randomforestModel.transform(test)
predictions.show()

dt_evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = dt_evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
predictions.show(5)
```

RMSE		
RUN 1	RUN 2	RUN 3
17.6018	17.7166	16.3496

OUTCOMES FROM THREE MODELS

- Out of three models ,Random forest model comes with low RMSE compared to others.
- RMSE was lower for Random forest even for the three runs

K-MEAN CLUSTERING

- Assemblers & transformation
- Feature Scaling
- Building the Model

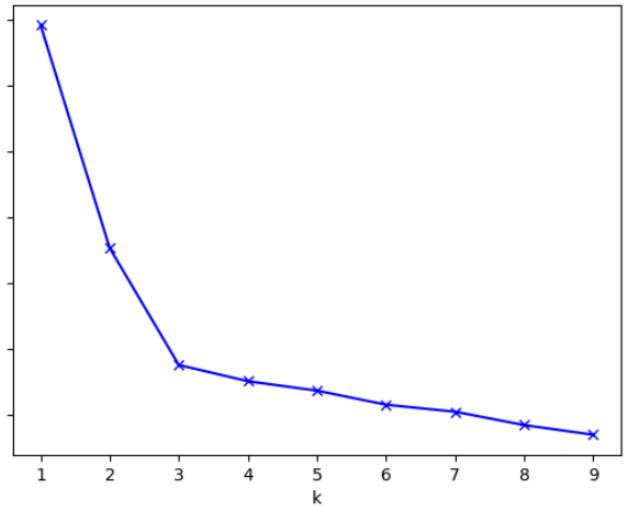
```
#*******************************************************************************
#       K-means
#*******************************************************************************
print("K- means started")
assembler = VectorAssembler(inputCols=['trip_seconds','trip_miles','fare'], outputCol='features')
assembled_data = assembler.transform(df_taxiTrips)
assembled_data.show(5)
scaler = StandardScaler(inputCol='features', outputCol='scaledFeatures')

scaler_model = scaler.fit(assembled_data)
scaled_data = scaler_model.transform(assembled_data)
scaled_data.printSchema()
scaled_data.show(5)

kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(scaled_data)
predictions = model.transform(scaled_data)

predictions.groupBy('prediction').count().show()
```

CLUSTER OPTIMIZATION-SILHOUETTE METHOD

- Provides a measure of how close each point in one cluster is to points in the neighboring clusters.
- This metric (silhouette width) ranges from **-1** to **1** for each observation in your data

Close to 1	Matched to assigned cluster
Close to 0	Borderline match between two clusters
Close to -1	May be assigned to wrong cluster

Silhouette with squared euclidean distance		
RUN 1	RUN 2	RUN 3
0.999256868	0.822596	0.832390661




```
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)

evaluator = ClusteringEvaluator()
cost = np.zeros(20)
silhouette = np.zeros(20)
for k in range(2,20):

    kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("features")
    model = kmeans.fit(scaled_data.sample(False,0.1, seed=42))
    cost[k] = model.computeCost(scaled_data)
    predictions = model.transform(scaled_data)
    silhouette[k] = evaluator.evaluate(predictions)
```

```
import matplotlib.pyplot as plt
%matplotlib inline
fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(range(2,20),cost[2:20])
ax.set_xlabel('k')
ax.set_ylabel('cost')
```

```
import matplotlib.pyplot as plt
%matplotlib inline
fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(range(2,20),silhouette[2:20])
ax.set_xlabel('k')
ax.set_ylabel('silhouette')
```


THANK
YOU