databricks Rating Prediction

1.EDA Analysis

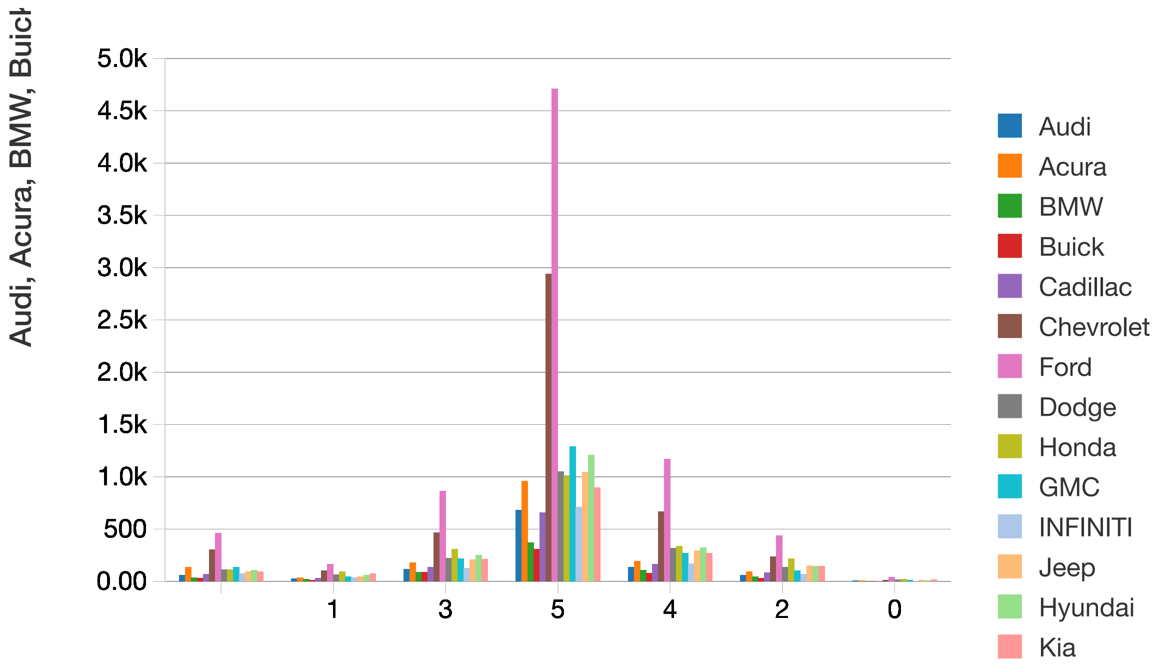
1. Inspecting inactive drivers who has not provided a drive

```
rides = spark.read.parquet("/mnt/cis442f-data/duocar/clean/rides/")
drivers = spark.read.parquet("/mnt/cis442f-data/duocar/clean/drivers/")
drivers.select('id').subtract(rides.select('driver_id')).count()
# There are 103 inactive drivers.
```

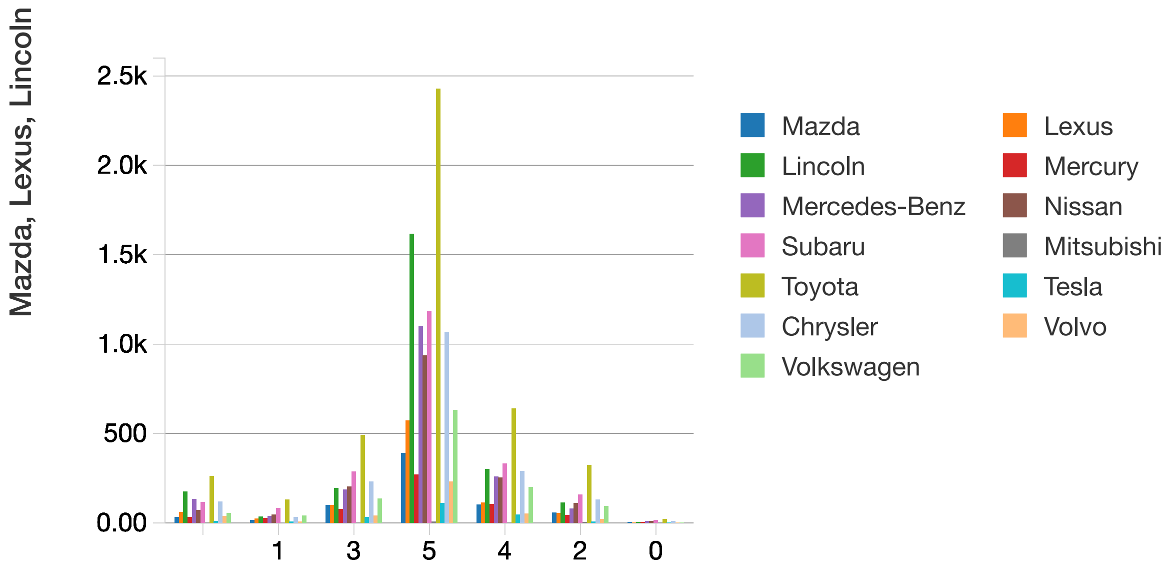
```
Out[8]: 103
```

2. Plot the ride rating in terms of vehicles

```
reviews = spark.read.parquet("/mnt/cis442f-data/duocar/clean/ride_reviews/")
joined =
rides.select('driver_id','star_rating').join(drivers.select('id','vehicle_make'
),rides.driver_id == drivers.id)
eda = joined.groupby('star_rating').pivot('vehicle_make').count()
display(eda)
```



```
display(eda)
```



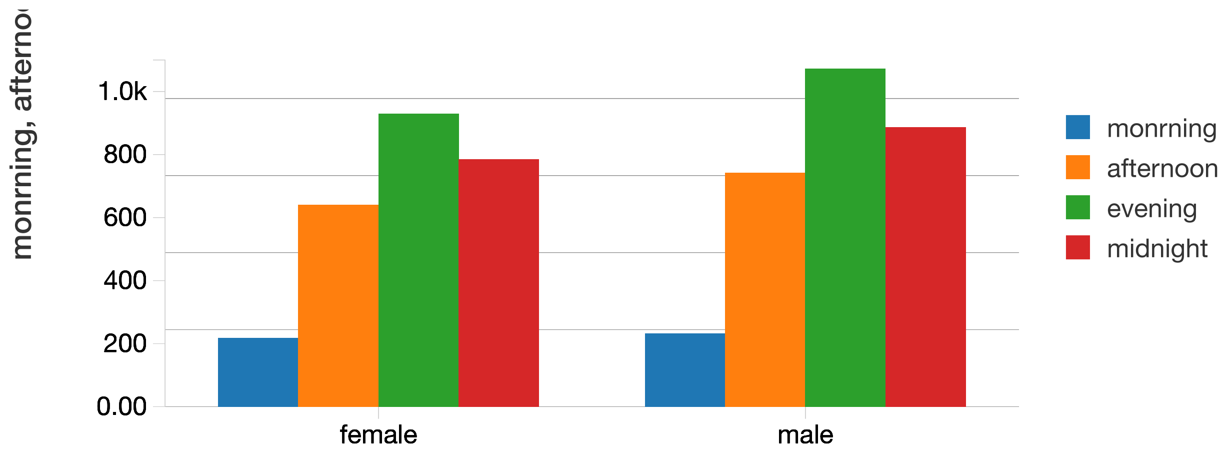
From the barplot and line chart, star rating is not dependent on vehicle make. The bar plot and line chart both indicates that 5 stars is the most frequently given rating. And the distribution of rating is similar regardless of vehicle make. Therefore, we can conclude that star rating is not dependent on vehicle make.

3. Investigate student users

```
# create new columns of the time information (date_time in a day, day of a
week, month)
riders = spark.read.parquet("/mnt/cis442f-data/duocar/clean/riders/")
joined2 = rides.join(riders,rides.rider_id ==
riders.id).select('rider_id','student','sex','date_time')

from pyspark.sql.functions import dayofweek,month,hour,when
joined2 = joined2.select('rider_id','student','sex',\

'date_time',dayofweek('date_time').alias('dayofweek'),month('date_time').alias(
'month'))\
          .withColumn('date_time', when((hour(joined2.date_time) > 5) &
(hour(joined2.date_time) < 13),'monrning')\
                  .when((hour(joined2.date_time) > 12) &
(hour(joined2.date_time) < 19),'afternoon').\
                   when((hour(joined2.date_time) > 18) &
(hour(joined2.date_time) < 24),'evening').otherwise('midnight'))
joined2.show(5)
```

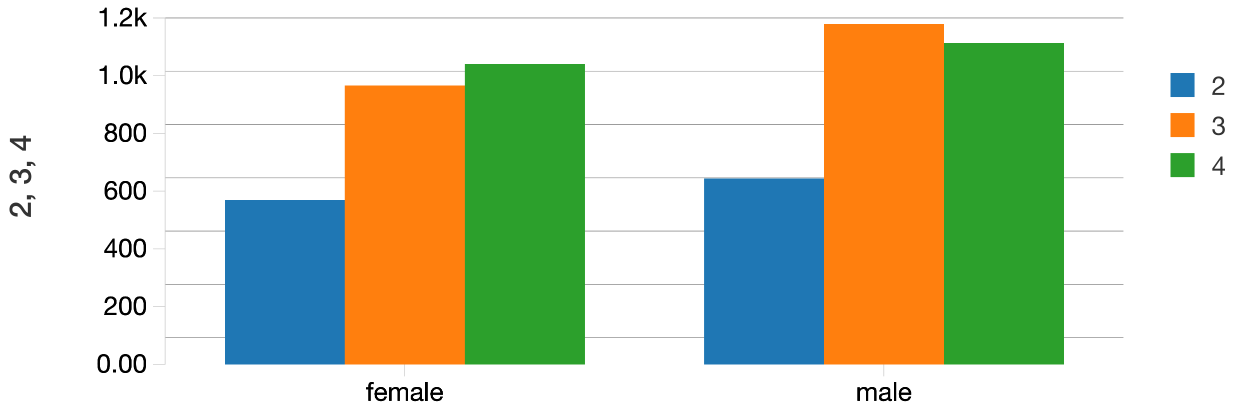
```
+------------+-------+------+---------+---------+-----+
|    rider_id|student|   sex|date_time|dayofweek|month|
+------------+-------+------+---------+---------+-----+
|220200000084|  false|female| monrning|        4|    2|
|220200000462|  false|  male| monrning|        4|    2|
|220200000489|  false|  male| monrning|        4|    2|
|220200000057|  false|  male| monrning|        4|    2|
|220200000012|   true|  null| monrning|        4|    2|
+------------+-------+------+---------+---------+-----+
only showing top 5 rows
```

```
# 1. time period during a day
datetime = joined2.dropna(subset = ['sex']).filter(joined2.student ==
True).groupby('sex').pivot('date_time').count()
display(datetime)
```



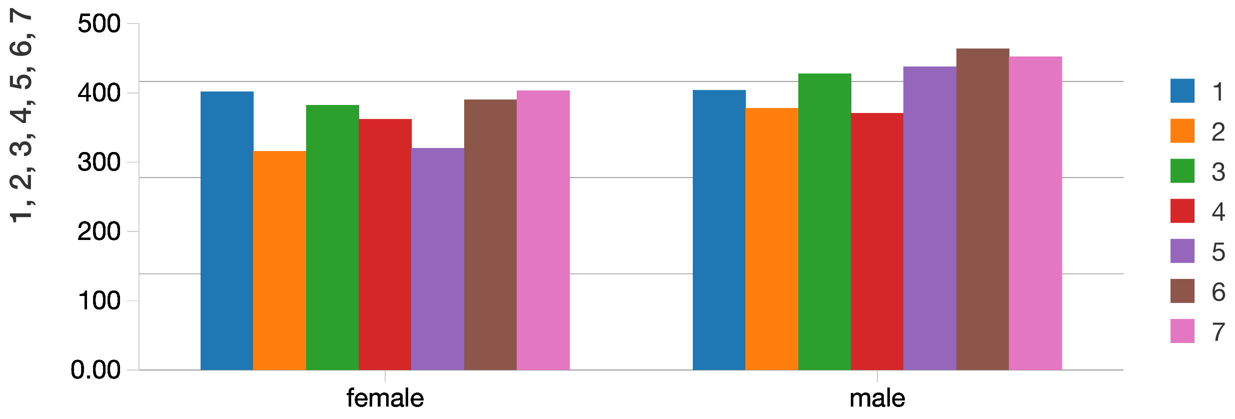
The bar chart indicates that student prefer taking a ride during evening and midnight (7pm to 5am the next day). Student seldom take a ride in the morning. Also, the distribution has no sex difference.

```
# 2.Month
month = joined2.dropna(subset = ['sex']).filter(joined2.student ==
True).groupby('sex').pivot('month').count()
display(month)
```



The bar chart indicates that females prefer taking a ride in April to February and March, while males prefer March. Both genders tend to take less rides in February.

```
# 3. day of week
dayofweek = joined2.dropna(subset = ['sex']).filter(joined2.student ==
True).groupby('sex').pivot('dayofweek').count()
display(dayofweek)
```



The bar chart shows that males tend to take more rides than females. Males prefer to take a ride during weekends and Fridays. And females take the most rides in Mondays and Sundays.

4. Explore the distance data

```
rides.where(rides.distance.isNull()).select('cancelled','distance').distinct().
show()
```

```
+---------+--------+
|cancelled|distance|
+---------+--------+
|     true|    null|
+---------+--------+
```

All rides records that has nulls in distance is due to cancelled trips.

2. Feature Selection and Engineering

```
records = spark.read.parquet("/mnt/cis442f-data/duocar/joined_all")
y = spark.read.format('csv').options(header='true',
inferSchema='true').load('/FileStore/tables/y.csv')
```

```
# Remove cancelled rides
from pyspark.ml.feature import SQLTransformer
processed = SQLTransformer(statement="SELECT * FROM __THIS__ WHERE cancelled ==
0").transform(records)
processed.count()
```

```
Out[17]: 45841
```

Basic NLP analysis of the reviews to select the most relevant features

```
rides = spark.read.parquet("/mnt/cis442f-data/duocar/clean/rides/")
ride_join_review = reviews.join(rides, reviews.ride_id == rides.id,
"left_outer")
reviews_with_ratings = ride_join_review.select("ride_id", "review",
"star_rating")
# Tokenize the reviews
from pyspark.ml.feature import RegexTokenizer
regexTokenizer = RegexTokenizer(inputCol="review", outputCol="words",
pattern="\\W")
regexTokenized = regexTokenizer.transform(reviews_with_ratings)
```

```
# Exam the sample of token
for item in regexTokenized.select('review','words').head(5):
       print(item[0],item[1])
```

```
Dale is extremely cordial. ['dale', 'is', 'extremely', 'cordial']
Very junky car. ['very', 'junky', 'car']
most awful stench of all time! throw away your air freshener! ['most', 'awful',
'stench', 'of', 'all', 'time', 'throw', 'away', 'your', 'air', 'freshener']
No trouble of note. ['no', 'trouble', 'of', 'note']
The driver drove so well!! ['the', 'driver', 'drove', 'so', 'well']
```

```
# Remove common words
from pyspark.ml.feature import StopWordsRemover
remover = StopWordsRemover(inputCol="words", outputCol="words_removed")
removed = remover.transform(regexTokenized)

# Compute term frequency
from pyspark.ml.feature import CountVectorizer
vectorizer = CountVectorizer(inputCol="words", outputCol="words_vectorized",
vocabSize=10)
vectorizer_model = vectorizer.fit(removed)
vectorized = vectorizer_model.transform(removed)
vectorized.select("words_removed", "words_vectorized").head(5)

# list the vocabulary and corresponding index
list(enumerate(vectorizer_model.vocabulary))
```

```
Out[20]:
[(0, 'the'),
 (1, 'ride'),
 (2, 'driver'),
 (3, 'to'),
 (4, 'was'),
 (5, 'air'),
 (6, 'i'),
 (7, 'freshener'),
 (8, 'and'),
 (9, 'car')]
```

```
from pyspark.sql.functions import array_contains, count, mean
for word in vectorizer_model.vocabulary:
  print ("**** word = %s ****\n" % word)
  vectorized \
    .select(array_contains("words_removed", word).alias("contains_word"),
"star_rating") \
    .groupBy("contains_word") \
    .agg(count("star_rating"), mean("star_rating")) \
    .show()
```

```
**** word = the ****

+-------------+------------------+-----------------+
|contains_word|count(star_rating)| avg(star_rating)|
+-------------+------------------+-----------------+
|        false|              1822|2.944017563117453|
+-------------+------------------+-----------------+
**** word = ride ****

+-------------+------------------+------------------+
|contains_word|count(star_rating)|  avg(star_rating)|
+-------------+------------------+------------------+
|         true|               631|3.0839936608557843|
|        false|              1191| 2.869857262804366|
+-------------+------------------+------------------+

**** word = driver ****

+-------------+------------------+------------------+
```

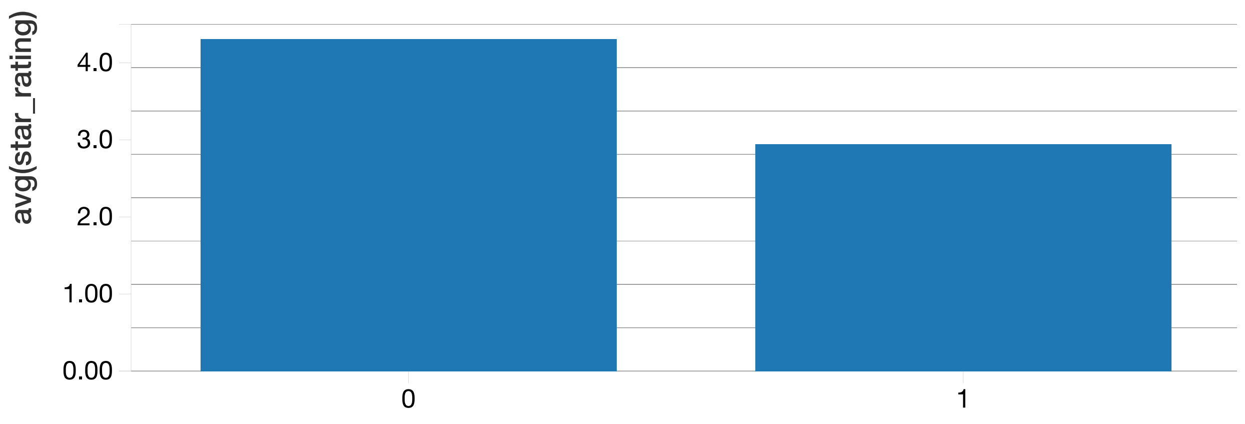
It shows that the vehicle, rider and service (such as refresher and air conditioner, but this part of information is not available, so I will focus on the first two parts) are riders concerns about.

```
def explore(df, feature, label):
  from pyspark.sql.functions import count, mean, stddev
  aggregated = df \
    .groupBy(feature) \
    .agg(count(label), mean(label), stddev(label)).orderBy(feature)
  return aggregated

records.withColumn("reviewed",
records.review.isNotNull()).groupBy("reviewed").count().show()
```

```
+--------+-----+
|reviewed|count|
+--------+-----+
|    true| 1822|
|   false|46953|
+--------+-----+
# Add new column of whether the ride was reviewed (numeric column because MLlib
requires numeric features)
from pyspark.sql.functions import col
engineered1 = processed.withColumn("reviewed",
col("review").isNotNull().cast("int"))

# apply the function defined above to explore the impact of whether reviewed on
rating
aggregated = explore(engineered1, "reviewed", "star_rating")
aggregated.show()
aggregated.createOrReplaceTempView("agg_view")
```

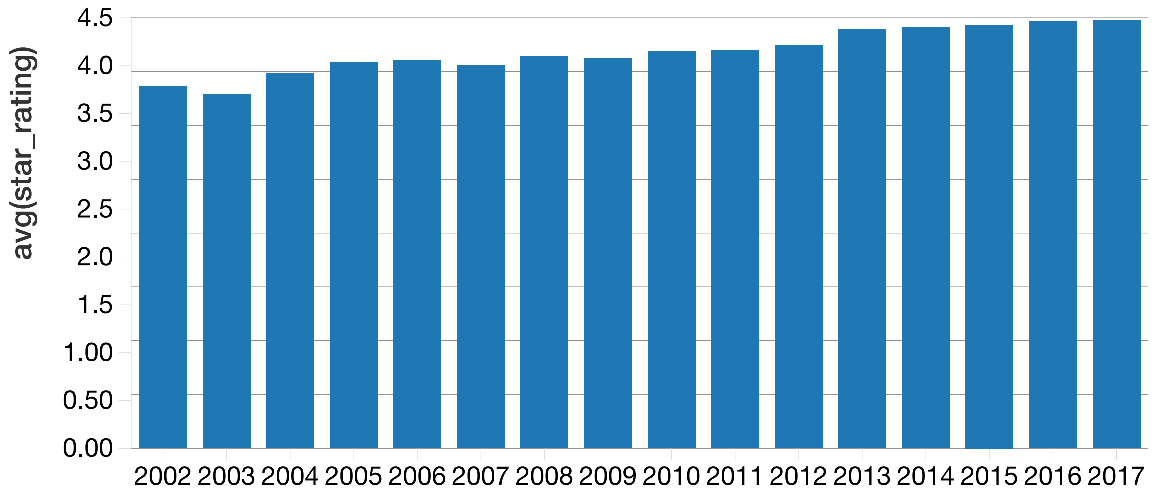
```
+--------+------------------+-----------------+-----------------------+
|reviewed|count(star_rating)| avg(star_rating)|stddev_samp(star_rating)|
+--------+------------------+-----------------+-----------------------+
|       0|             44019|4.307889774869943|      1.081740043694224|
|       1|              1822|2.944017563117453|     1.5765806972412695|
+--------+------------------+-----------------+-----------------------+
```

```
%sql
SELECT * FROM agg_view
```



The bar plot shows that riders tend to rate lower if a review is given. So this feature will be put into model.

```
# vehicle year
aggregated = explore(engineered1, "vehicle_year", "star_rating")
aggregated.show()
display(aggregated)
```



```
# create a new feature : dirver age ('age')
from pyspark.sql.functions import to_date, current_date, months_between, floor,
to_timestamp

engineered1 = engineered1.withColumn("age",
floor(months_between(current_date(), "driver_birth_date") / 12))
```

```
# Transform the categorical feature into a vector of doubles
# vehicle_color
from pyspark.ml.feature import StringIndexer
indexer = StringIndexer(inputCol="vehicle_color", outputCol="vehicle_color_ix")
indexer_model = indexer.fit(engineered1)
indexed = indexer_model.transform(engineered1)
from pyspark.ml.feature import OneHotEncoder
encoder = OneHotEncoder(inputCol="vehicle_color_ix",
outputCol="vehicle_color_cd")
encoded = encoder.transform(indexed)

# vehicle brand: vehicle_make
indexer2 = StringIndexer(inputCol="vehicle_make", outputCol="vehicle_make_ix")
indexer2_model = indexer2.fit(encoded)
indexed2 = indexer2_model.transform(encoded)
encoder2 = OneHotEncoder(inputCol="vehicle_make_ix",
outputCol="vehicle_make_cd")
encoded2 = encoder2.transform(indexed2)
```

```
# Features: 6
# numeric features: 'vehicle_year', driver age ('age'),'driver_median_income',
# categorical features: whether reviewed ('reviewed'), vehicle color
('vehicle_color_cd'), vehicle brand ('vehicle_make_cd')
```

```
# Select the features (and label)
selected = encoded2.select("reviewed", "age",
"driver_median_income","vehicle_year", "vehicle_color_cd",
"vehicle_make_cd","star_rating")
features = ["reviewed", "age", "driver_median_income","vehicle_year",
"vehicle_color_cd", "vehicle_make_cd"]
```

```
# Assemble the features into a single column of vectors
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=features, outputCol="features")
assembled = assembler.transform(selected)

for item in assembled.head(5):
    print (item)
```

```
Row(reviewed=0, age=33, driver_median_income=28520, vehicle_year=2015, vehicle_
color_cd=SparseVector(9, {1: 1.0}), vehicle_make_cd=SparseVector(26, {6: 1.0}),
star_rating=5, features=SparseVector(39, {1: 33.0, 2: 28520.0, 3: 2015.0, 5: 1.
0, 19: 1.0}))
```

```
Row(reviewed=0, age=51, driver_median_income=44444, vehicle_year=2015, vehicle_
color_cd=SparseVector(9, {0: 1.0}), vehicle_make_cd=SparseVector(26, {14: 1.
0}), star_rating=5, features=SparseVector(39, {1: 51.0, 2: 44444.0, 3: 2015.0,
4: 1.0, 27: 1.0}))
Row(reviewed=0, age=33, driver_median_income=28520, vehicle_year=2015, vehicle_
color_cd=SparseVector(9, {1: 1.0}), vehicle_make_cd=SparseVector(26, {6: 1.0}),
star_rating=5, features=SparseVector(39, {1: 33.0, 2: 28520.0, 3: 2015.0, 5: 1.
0, 19: 1.0}))
Row(reviewed=0, age=26, driver_median_income=21935, vehicle_year=2016, vehicle_
color_cd=SparseVector(9, {6: 1.0}), vehicle_make_cd=SparseVector(26, {23: 1.
0}), star_rating=5, features=SparseVector(39, {1: 26.0, 2: 21935.0, 3: 2016.0,
10: 1.0, 36: 1.0}))
Row(reviewed=0, age=26, driver_median_income=21935, vehicle_year=2016, vehicle_
color_cd=SparseVector(9, {6: 1.0}), vehicle_make_cd=SparseVector(26, {23: 1.
0}), star_rating=5, features=SparseVector(39, {1: 26.0, 2: 21935.0, 3: 2016.0,
10: 1.0, 36: 1.0}))
```

3. Modeling & Tuning Hyperparameters

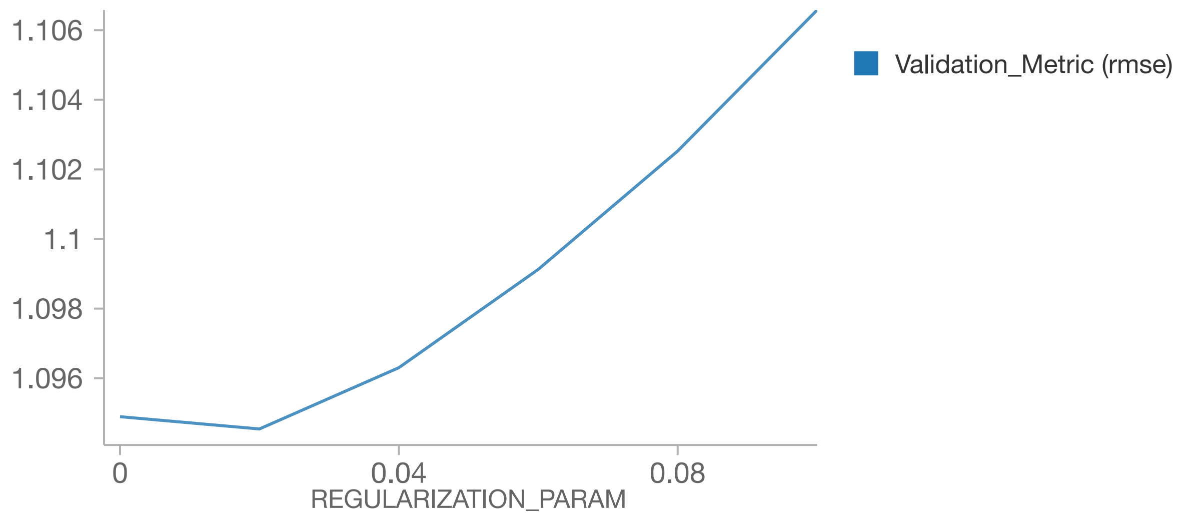
```
# split train, test datasets
(train, test) = assembled.randomSplit([0.8, 0.2], 12345)
```

```
# estimator
from pyspark.ml.regression import LinearRegression
lr = LinearRegression(featuresCol="features", labelCol="star_rating",
elasticNetParam=1.0)

# Set regularization parameters
from pyspark.ml.tuning import ParamGridBuilder
regParamList = [0.0, 0.02, 0.04, 0.06, 0.08, 0.1]
grid = ParamGridBuilder().addGrid(lr.regParam, regParamList).build()
```

Holdout Cross-Validation

```
# evaluator
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(predictionCol="prediction",
labelCol="star_rating", metricName="rmse")

# specify holdout cross-validation
from pyspark.ml.tuning import TrainValidationSplit
validator = TrainValidationSplit(estimator=lr, estimatorParamMaps=grid,
evaluator=evaluator, trainRatio=0.75, seed=54321)
cv_model = validator.fit(train)
cv_model.validationMetrics
to_plot = zip(regParamList,cv_model.validationMetrics) # zip the two lists
together
to_plot_df = spark.createDataFrame(to_plot, schema=["Regularization_Param",
"Validation_Metric (rmse)"])
```

```
display(to_plot_df)
```



The best performance was with the regularization parameter at 0.02. The range of 0-0.2 worth

```
# the final selected after hold-out cross validation
cv_model.bestModel
```

```
Out[34]: LinearRegression_513a99334311
```

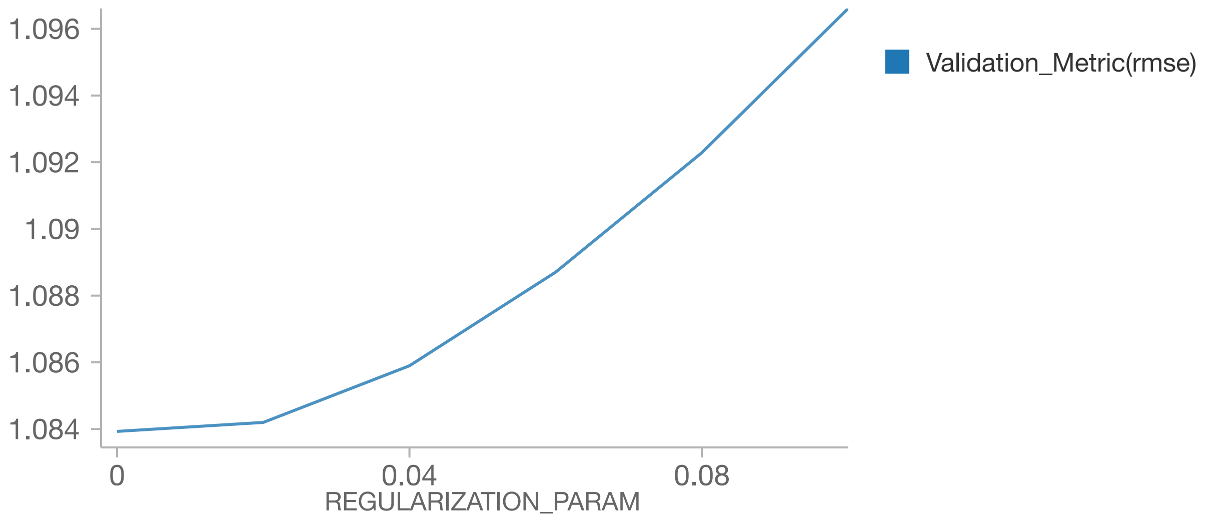
```
# Query model and model performance
print ("R-Squared: " + str(cv_model.bestModel.summary.r2))
print ("RMSE: " + str(cv_model.bestModel.summary.rootMeanSquaredError))
print ("Intercept: " + str(cv_model.bestModel.intercept))
print ("Coefficients: " + str(cv_model.bestModel.coefficients))
```

```
R-Squared: 0.09407924169267223
RMSE: 1.08404717362454
Intercept: -71.90237928431685
Coefficients: [-1.21920494996,0.0,0.0,0.0378759322913,0.0536217897841,0.0,0.0,
0.0,0.0,0.0,-0.401028538931,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
```

KFold Cross-Validation

```
# kfold cross validation
from pyspark.ml.tuning import CrossValidator
kfold_validator = CrossValidator(estimator=lr, estimatorParamMaps=grid,
evaluator=evaluator, numFolds=3, seed=54321)
kfold_model = kfold_validator.fit(train)

to_plot2 = zip(regParamList,kfold_model.avgMetrics)
to_plot_df2 = spark.createDataFrame(to_plot2, schema=
["Regularization_Param","Validation_Metric(rmse)"])
```

```
display(to_plot_df2)
```



```
# Query model and model performance
print ("R-Squared: " + str(kfold_model.bestModel.summary.r2))
print ("RMSE: " + str(kfold_model.bestModel.summary.rootMeanSquaredError))
print ("Intercept: " + str(kfold_model.bestModel.intercept))
print ("Coefficients: " + str(kfold_model.bestModel.coefficients))
```

```
R-Squared: 0.09607776524622447
RMSE: 1.0828507720804852
Intercept: -79.50948563409835
Coefficients: [-1.31289526381,0.000469241983867,-1.01738120035e-07,0.0416838240
305,0.121037822626,0.0475347447431,0.0805855540242,0.0517031075095,0.0508670265
528,0.0998551417725,-0.428269832117,0.0769728390834,0.0561764408168,-0.11451831
9463,-0.0916873485388,-0.110493440747,-0.111846701437,-0.175153358253,-0.127002
121635,-0.113241829759,-0.0971955671948,-0.146980197248,-0.140955346114,-0.1479
58632323,-0.0958962604623,-0.198755269448,-0.201953613522,-0.117716852587,-0.10
0472380078,-0.125570572321,-0.122787145928,-0.0783664383497,-0.0882144532323,-
0.0209449815484,-0.160219721351,-0.12380160688,-0.188814060627,-0.172859559228,
-0.10780640988]
```

4. Evaluate the model performance on the test set

```
test_with_predictions = kfold_model.bestModel.transform(test)
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(predictionCol="prediction",
labelCol="star_rating", metricName="r2")
evaluator.evaluate(test_with_predictions)
evaluator.setMetricName("rmse").evaluate(test_with_predictions)
```

```
Out[39]: 1.0821377606198583
```

However, the purpose is to improve user experience and manage drivers. Therefore, compared with predict rating simply, predicting whether the ride is "good" or "not satisfactory" is more inspiring.

- Turn it to classification problem [will use Random Forest]
- Label star_rating into two categories (above or below 4 stars)

```
# change the star_rating into categorical (choose 4 as threshold)
from pyspark.ml.feature import Binarizer
converted = encoded2.withColumn("star_rating",
col("star_rating").cast("double"))
binarizer = Binarizer(inputCol="star_rating", outputCol="high_rating",
threshold = 4)
labeled = binarizer.transform(converted)
labeled.crosstab("star_rating", "high_rating").show()
```

```
+-----------------------+----+-----+
|star_rating_high_rating| 0.0|  1.0|
+-----------------------+----+-----+
|                    0.0| 256|    0|
|                    5.0|   0|28385|
|                    1.0|1286|    0|
|                    2.0|3155|    0|
|                    3.0|5564|    0|
|                    4.0|7195|    0|
+-----------------------+----+-----+
```

```
# Select the features (and label)
selected2 = labeled.select("reviewed", "age",
"driver_median_income","vehicle_year", "vehicle_color_cd",
"vehicle_make_cd","high_rating")
features2 = ["reviewed", "age", "driver_median_income","vehicle_year",
"vehicle_color_cd", "vehicle_make_cd"]
```

```
from pyspark.ml.feature import VectorAssembler
assembler2 = VectorAssembler(inputCols=features2, outputCol="features")
assembled2 = assembler2.transform(selected2)
```

```
(train, test) = assembled2.randomSplit([0.8, 0.2], 12345)
```

```
# train random forest on training set
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(featuresCol = 'features', labelCol =
'high_rating',numTrees = 100)

# kfold cross-validation
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator2 = BinaryClassificationEvaluator(rawPredictionCol="prediction",
labelCol="high_rating", metricName="areaUnderROC")
kfold_validator2 = CrossValidator(estimator=rf, estimatorParamMaps=grid,
evaluator=evaluator2, numFolds=3, seed=54321)
kfold_model2 = kfold_validator2.fit(train)
```

```
rf_model = kfold_model2.bestModel
rf_predictions = rf_model.transform(test)
```

Evaluation - area under ROC

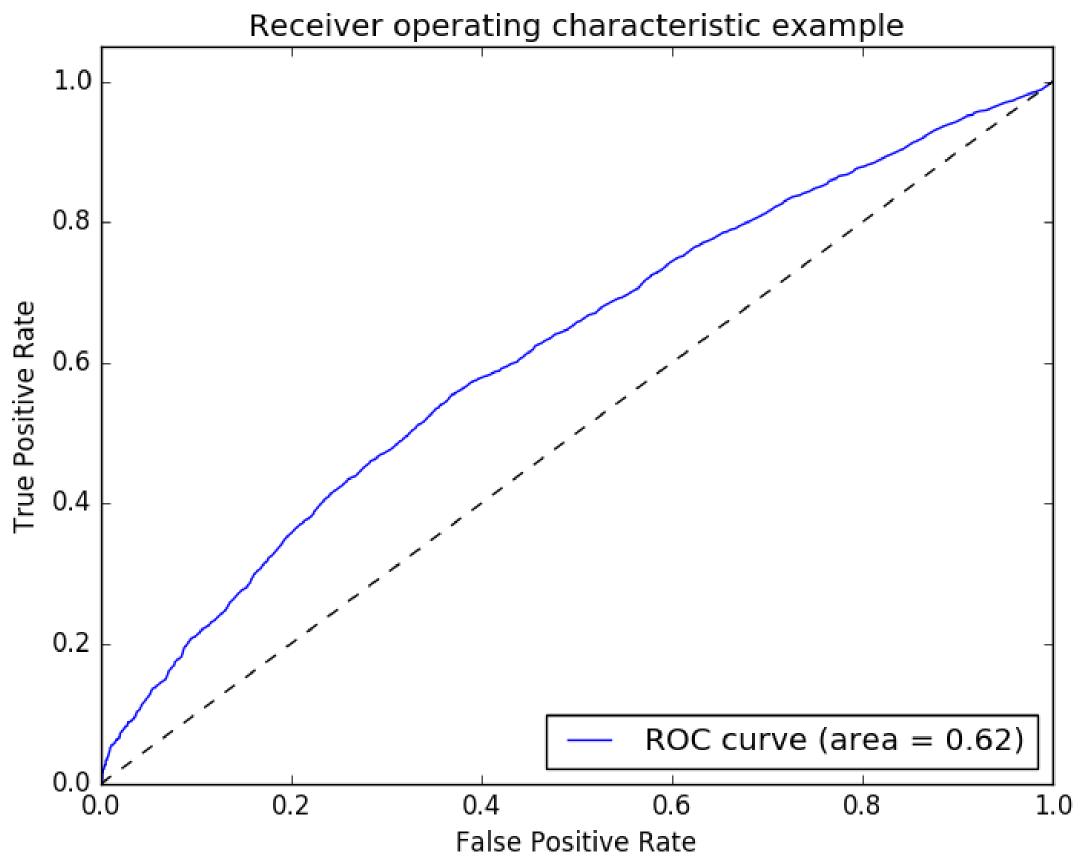
```
# area under ROC
from pyspark.mllib.evaluation import BinaryClassificationMetrics as metric
results = rf_predictions.select(['probability', 'high_rating'])

## prepare score-label set
results_collect = results.collect()
results_list = [(float(i[0][0]), 1.0-float(i[1])) for i in results_collect]
scoreAndLabels = sc.parallelize(results_list)

metrics = metric(scoreAndLabels)
print("The ROC score is (@numTrees=100): ", metrics.areaUnderROC)
```

```
The ROC score is (@numTrees=100):  0.6174601067545009
```

```
from sklearn.metrics import roc_curve, auc
from matplotlib import pyplot as plt

fpr = dict()
tpr = dict()
roc_auc = dict()

y_test = [i[1] for i in results_list]
y_score = [i[0] for i in results_list]

fpr, tpr, _ = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)

fig, ax = plt.subplots()
ax.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
ax.plot([0, 1], [0, 1], 'k--')
ax.set_xlim([0.0, 1.0])
ax.set_ylim([0.0, 1.05])
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.set_title('Receiver operating characteristic example')
ax.legend(loc="lower right")
```

```
Out[75]: <matplotlib.legend.Legend at 0x7f0f58045d30>
```

```
display(fig)
```



Evaluation - Accuracy, Precision, Recall, F-Score

```
tp = rf_predictions.filter((rf_predictions.high_rating == 1)&
(rf_predictions.prediction == 1)).count()
tn = rf_predictions.filter((rf_predictions.high_rating == 0)&
(rf_predictions.prediction == 0)).count()
fp = rf_predictions.filter((rf_predictions.high_rating == 0)&
(rf_predictions.prediction == 1)).count()
fn = rf_predictions.filter((rf_predictions.high_rating == 1)&
(rf_predictions.prediction == 0)).count()
```

```
Accuracy is 0.636275565123789
Recall is 0.977496970746062
Precision is 0.6347796762589928
F score is 0.7697130784433994
```