Assignment 2: Predicting Taxi Trip Duration using Regression Trees
Mining Massive Datasets

Aniketh Suresh, G01172069

A. Data Exploration and Feature Extraction

The dataset initially revealed **1458644** records. There were no duplicate records ("id"). The distribution for vendor count is given in Figure 1.

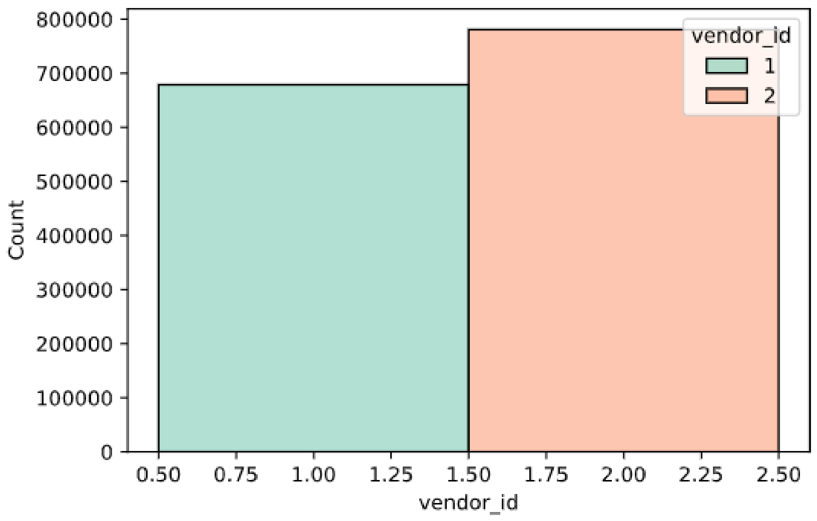


Figure 1: vendor_id (x-axis represents vendor: 1 and 2)

Plotting the "passenger_count" revealed a range of 0 <= passenger_count <= 9. A distribution for the passenger_counts is given in Figure 2.

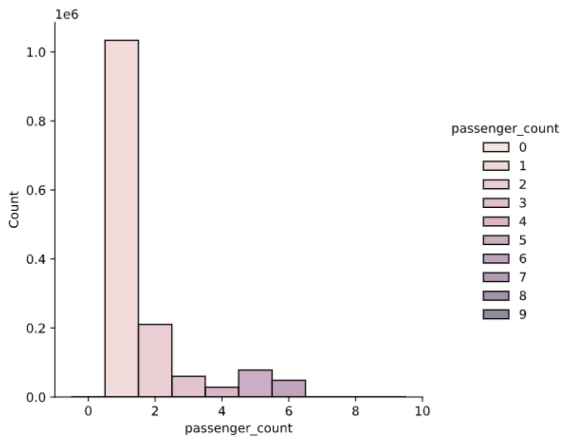


Figure 2: passenger_counts (since passenger counts of 1 account for most of the trips, the other values seems much smaller)

While a trip of 9 seems unlikely, it is most certain that passenger_count of 0 would not be possible. Therefore, all trips with “passenger_count” = 0, were dropped. This accounted for 60 trips.

It would not make sense for the “store_and_fwd_flags” to be of added value to the features, hence the column was dropped.

Since the trip_duration can be directly calculated through the difference between the “pickup_datetime” and “dropoff_datetime”, and since I intend to prevent this from becoming a trivial prediction problem, I decided to drop “dropoff_datetime”. Before doing so, I checked to see that *seconds (dropoff_datetime - pickup_datetime)* == *trip_duration*. However, for 5 trips, this was not the case. In these trips, the trip_duration exceeded “normal” trip times, with some being more than a few days in length. These trips were dropped.

The “pickup_datetime” values were converted to “month”, “day”, “hour” and “weekday”. A graph of the “vendor_id” with the “month” can be seen in Figure 3.

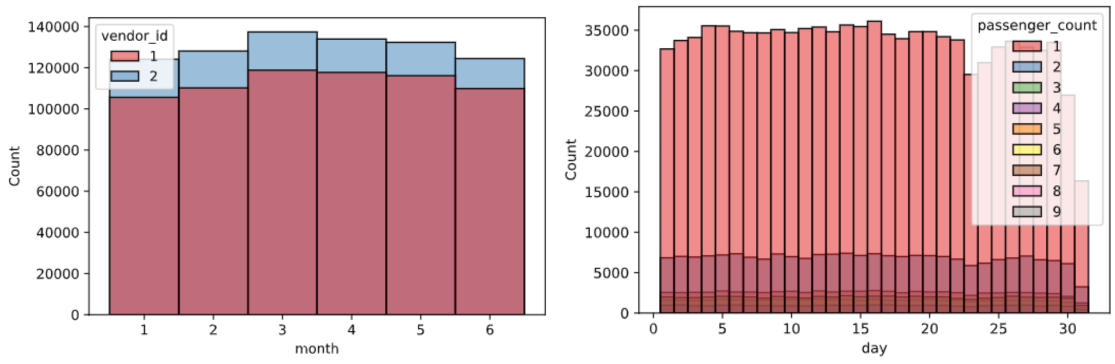


Figure 3: (left) A histogram of “vendor_id” with “month”. For every month, vendor_id 2 had more trips compared to vendor_id 1. (right) Histogram of the day of the month with the passenger_count

“pickup_longitude”, “pickup_latitude”, “dropoff_latitiude” and “dropoff_longitude” were used to calculate the distance between the pickup and dropoff location. The distance metric used here is the GEODESIC DISTANCE, which is the shortest distance between two points on a surface (the surface here being the curvature of the earth). This could also be related to the distance metric “as the crow flies”. This made up part of the feature called “distance”. Other distance metrics could be tested here (e.g., Manhattan distance). As a result, I now have 9 attributes:

```
root
|-- id: string (nullable = true)
|-- vendor_id: integer (nullable = true)
|-- passenger_count: integer (nullable = true)
|-- month: integer (nullable = true)
|-- day: integer (nullable = true)
|-- hour: integer (nullable = true)
|-- weekday: integer (nullable = true)
|-- distance: double (nullable = true)
|-- trip_duration: double (nullable = true)
```

B. Modelling and Algorithms

1. Regression Tree Model

A decision tree (regressor) was trained, validated (10 folds) and tested using 80% for training and 20% for testing. While you won't see the specific code for validation, and feature selection (since there are only a few parameters with which it can be tuned), I manually performed the validation, which resulted in the best model with depth set to 3. With larger values of depth, the RMSE (Root mean squared error)/MAE (mean absolute error) either overfits or produces worse results. Selecting depth of 3 also resulted in only 8 leaves, which allowed for easier testing/debugging.
The resulting tree is shown below:

```
DecisionTreeRegressionModel: uid=DecisionTreeRegressor_826b1c042824, depth=3,
numNodes=15, numFeatures=7
 If (feature 6 <= 2.7002733639393384)
  If (feature 6 <= 1.3071311166631614)
   If (feature 6 <= 0.825910208978972)
    Predict: 481.0347939172201
   Else (feature 6 > 0.825910208978972)
    Predict: 704.5021037177617
  Else (feature 6 > 1.3071311166631614)
   If (feature 6 <= 1.8821559421975278)
    Predict: 894.6080785101466
   Else (feature 6 > 1.8821559421975278)
    Predict: 1121.6659710628826
 Else (feature 6 > 2.7002733639393384)
  If (feature 6 <= 9.536082191310886)
   If (feature 6 <= 5.077256060808084)
    Predict: 1422.6793786966068
   Else (feature 6 > 5.077256060808084)
    Predict: 1959.3655415697908
  Else (feature 6 > 9.536082191310886)
   If (feature 4 <= 19.5)
    Predict: 3028.8669666598485
   Else (feature 4 > 19.5)
    Predict: 2309.267728834739
```

We can see from the tree above, that feature 6 (distance) was considered 12/14 times as the splitting test. The only other feature that was considered was feature 4 (hour) and was considered 2/14 times, which is interesting. Therefore, if you really need to create a smaller dataset to achieve the same result, we can drop all the columns apart from “distance” and “hour”.

2. Enhanced Regression Tree Model

An enhanced regression tree was constructed to predict the trip_duration. Similar to the regression tree model, the enhanced regression tree was also trained, validated and tested. Manual testing was not possible since there were a few parameters to tune. The parameters that were tuned were, epsilon, maxIter, regParam and elasticNetParam. Therefore, we can see that regularization was used in this tree model. 80% of the data was used as the training set with cross validation (10 folds) and the remaining was used as the test set.

The results in RMSE and MAE between the two models are shown below:

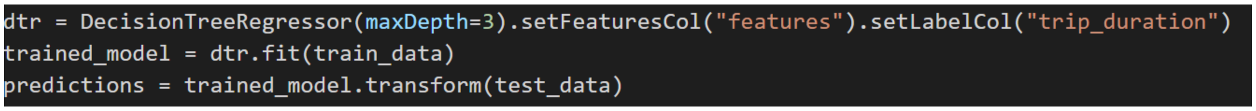
	RMSE	**MAE**
Regression Tree	3049.2010	421.2432
Enhanced Regression Tree	3136.940	427.5465

C. Conclusion

The Regression tree performed quite decently with the data given. Without any data exploration and feature extraction, the decision tree performed with an RMSE of about ~6000, which reduced to ~3000 after the case (not shown in the code. Was computed earlier). The Enhanced Decision Tree performed better than the original tree in some cases; this could be due to the randomness added due to the random split of the data. In some cases, even the original tree performed better.

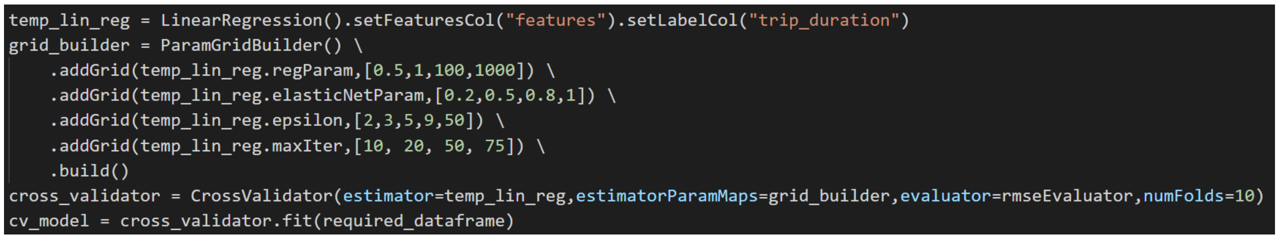
D. Pseudo-code (Examples)

The figure below depicts the training of a decision tree regressor.



```
dtr = DecisionTreeRegressor(maxDepth=3).setFeaturesCol("features").setLabelCol("trip_duration")
trained_model = dtr.fit(train_data)
predictions = trained_model.transform(test_data)
```

The figure below depicts the training and validation of a linear regression tree. Since I had 8 leaves, I ended up with 8 linear regression models.



```
temp_lin_reg = LinearRegression().setFeaturesCol("features").setLabelCol("trip_duration")
grid_builder = ParamGridBuilder() \
    .addGrid(temp_lin_reg.regParam,[0.5,1,100,1000]) \
    .addGrid(temp_lin_reg.elasticNetParam,[0.2,0.5,0.8,1]) \
    .addGrid(temp_lin_reg.epsilon,[2,3,5,9,50]) \
    .addGrid(temp_lin_reg.maxIter,[10, 20, 50, 75]) \
    .build()
cross_validator = CrossValidator(estimator=temp_lin_reg,estimatorParamMaps=grid_builder,evaluator=rmseEvaluator,numFolds=10)
cv_model = cross_validator.fit(required_dataframe)
```