Learning Practice 7 for the University of Tulsa's QM-7063 Data Mining Course

Classification and Regression Trees

Professor: Dr. Abdulrashid, Spring 2023

Noah L. Schrick - 1492657

```python
# Imports

%matplotlib inline

from pathlib import Path
import matplotlib.pylab as plt

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import export_text
from sklearn import tree
import scikitplot as skplt
from dmba import regressionSummary, classificationSummary
from dmba import plotDecisionTree
from sklearn.model_selection import GridSearchCV
from prettytable import PrettyTable
```

Problem 9.1

Problem Statement Competitive Auctions on eBay.com.

The file eBayAuctions.csv contains information on 1972 auctions that transacted on eBay.com during May–June 2004. The goal is to use these data to build a model that will classify auctions as competitive or non-competitive. A competitive auction is defined as an auction with at least two bids placed on the item auctioned. The data include variables that describe the item (auction category), the seller (his/her eBay rating), and the auction terms that the seller selected (auction duration, opening price, currency, day-of-week of auction close). In addition, we have the price at which the auction closed. The task is to predict whether or not the auction will be competitive.

Tasks

Data Preprocessing.

Convert variable Duration into a categorical variable. Split the data into training (60%) and validation (40%) datasets.

```
# Data pre-processing
auction_df = pd.read_csv('eBayAuctions.csv')

# Convert cols to categorical
auction_df['Duration'] = auction_df['Duration'].astype('category')
auction_df = pd.get_dummies(auction_df, prefix_sep='_',
drop_first=True)

# Spec outcome
X = auction_df.drop(columns=['Competitive?'])
y = auction_df['Competitive?']
# 60/40 split
train_X, valid_X, train_y, valid_y = train_test_split(X, y,
test_size=0.4, random_state=1)
```

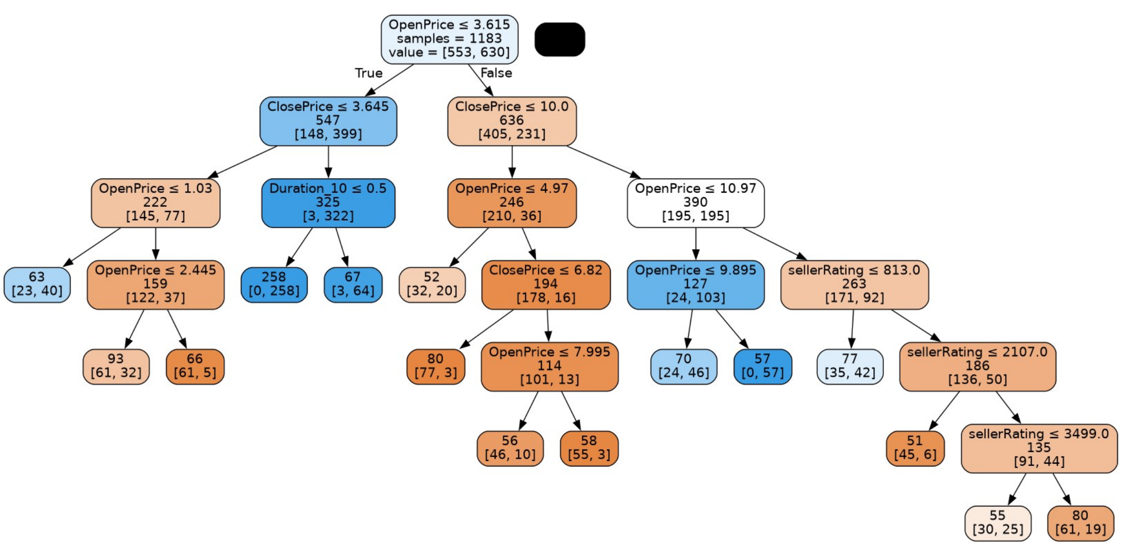
a.

Fit a classification tree using all predictors. To avoid overfitting, set the minimum number of records in a terminal node to 50 and the maximum tree depth to 7. Write down the results in terms of rules. (Note: If you had to slightly reduce the number of predictors due to software limitations, or for clarity of presentation, which would be a good variable to choose?)

```
# a
fullClassTree = DecisionTreeClassifier(random_state=1,
min_samples_leaf=50, max_depth=7)
fullClassTree.fit(train_X, train_y)
tree_rules = export_text(fullClassTree,
feature_names=list(train_X.columns))
print(tree_rules)

plotDecisionTree(fullClassTree, feature_names=train_X.columns)
```

```
|--- OpenPrice <= 3.62
|   |--- ClosePrice <= 3.64
|   |   |--- OpenPrice <= 1.03
|   |   |   |--- class: 1
|   |   |--- OpenPrice >  1.03
|   |   |   |--- OpenPrice <= 2.45
|   |   |   |   |--- class: 0
|   |   |   |--- OpenPrice >  2.45
|   |   |   |   |--- class: 0
|   |--- ClosePrice >  3.64
```

```
|   |   |--- Duration_10 <= 0.50
|   |   |   |--- class: 1
|   |   |--- Duration_10 >  0.50
|   |   |   |--- class: 1
|--- OpenPrice >  3.62
|   |--- ClosePrice <= 10.00
|   |   |--- OpenPrice <= 4.97
|   |   |   |--- class: 0
|   |   |--- OpenPrice >  4.97
|   |   |   |--- ClosePrice <= 6.82
|   |   |   |   |--- class: 0
|   |   |   |--- ClosePrice >  6.82
|   |   |   |   |--- OpenPrice <= 7.99
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- OpenPrice >  7.99
|   |   |   |   |   |--- class: 0
|   |--- ClosePrice >  10.00
|   |   |--- OpenPrice <= 10.97
|   |   |   |--- OpenPrice <= 9.89
|   |   |   |   |--- class: 1
|   |   |   |--- OpenPrice >  9.89
|   |   |   |   |--- class: 1
|   |   |--- OpenPrice >  10.97
|   |   |   |--- sellerRating <= 813.00
|   |   |   |   |--- class: 1
|   |   |   |--- sellerRating >  813.00
|   |   |   |   |--- sellerRating <= 2107.00
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- sellerRating >  2107.00
|   |   |   |   |   |--- sellerRating <= 3499.00
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- sellerRating >  3499.00
|   |   |   |   |   |   |--- class: 0
```



If you had to slightly reduce the number of predictors due to software limitations, or for clarity of presentation, which would be a good variable to choose?

Category and Currency. Both predictors have low importance, and are not used in branching for the Decision Tree.

b.

Is this model practical for predicting the outcome of a new auction?

```
# b
classificationSummary(valid_y, fullClassTree.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8162)

       Prediction
Actual   0   1
     0 305  48
     1  97 339
```

This model works well for the dataset provided, but is not practical. The primary issue is that this model uses closePrice to predict the outcome, and closePrice is not something known in advance. In addition, for this set of data, the tree is quick to build and use, and has a 81.62% accuracy. However, many of the rules appear overfitted for the data provided.

c.

Describe the interesting and uninteresting information that these rules provide. Of interest: The tree starts the split with OpenPrice, and is able to cleanly make a binary split.

Not of interest: If the OpenPrice > 3.62, the next split is based on ClosePrice. However, it "splits" into a "0" category, meaning ClosePrice does not apply much to the training data. From here, the rules appear overfitted, choosing various price points to split at.

d.

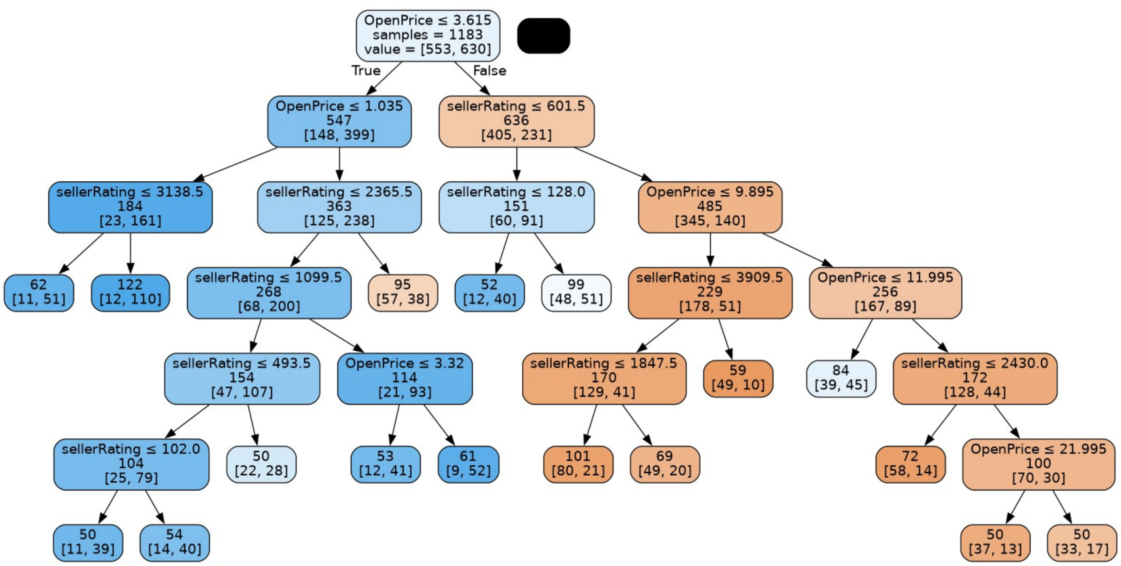
Fit another classification tree (using a tree with a minimum number of records per terminal node = 50 and maximum depth = 7), this time only with predictors that can be used for predicting the outcome of a new auction. Describe the resulting tree in terms of rules. Make sure to report the smallest set of rules required for classification.

```
# d
auction_df_2 = pd.read_csv('eBayAuctions.csv')

# Convert cols to categorical
auction_df_2['Duration'] = auction_df_2['Duration'].astype('category')
auction_df_2 = pd.get_dummies(auction_df_2, drop_first=True)

# Spec outcome
X_2 = auction_df_2.drop(list(auction_df_2.filter(regex = 'Category')),
axis = 1)
```

```python
X_2 = X_2.drop(list(X_2.filter(regex = 'currency')), axis = 1)
X_2 = X_2.drop(list(X_2.filter(regex = 'Competitive?')), axis = 1)
X_2 = X_2.drop(list(X_2.filter(regex = 'ClosePrice')), axis = 1)
y_2 = auction_df_2['Competitive?']

# 60/40 split
train_X_2, valid_X_2, train_y_2, valid_y_2 = train_test_split(X_2,
y_2, test_size=0.4, random_state=1)

fullClassTree_2 = DecisionTreeClassifier(random_state=1,
min_samples_leaf=50, max_depth=7)
fullClassTree_2.fit(train_X_2, train_y_2)
tree_rules_2 = export_text(fullClassTree_2,
feature_names=list(train_X_2.columns))
print(tree_rules_2)
plotDecisionTree(fullClassTree_2, feature_names=train_X_2.columns)
```

```
|--- OpenPrice <= 3.62
|   |--- OpenPrice <= 1.04
|   |   |--- sellerRating <= 3138.50
|   |   |   |--- class: 1
|   |   |--- sellerRating >  3138.50
|   |   |   |--- class: 1
|   |--- OpenPrice >  1.04
|   |   |--- sellerRating <= 2365.50
|   |   |   |--- sellerRating <= 1099.50
|   |   |   |   |--- sellerRating <= 493.50
|   |   |   |   |   |--- sellerRating <= 102.00
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- sellerRating >  102.00
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- sellerRating >  493.50
|   |   |   |   |   |--- class: 1
|   |   |   |--- sellerRating >  1099.50
|   |   |   |   |--- OpenPrice <= 3.32
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- OpenPrice >  3.32
|   |   |   |   |   |--- class: 1
|   |   |--- sellerRating >  2365.50
|   |   |   |--- class: 0
|--- OpenPrice >  3.62
|   |--- sellerRating <= 601.50
|   |   |--- sellerRating <= 128.00
|   |   |   |--- class: 1
|   |   |--- sellerRating >  128.00
|   |   |   |--- class: 1
|   |--- sellerRating >  601.50
|   |   |--- OpenPrice <= 9.89
|   |   |   |--- sellerRating <= 3909.50
|   |   |   |   |--- sellerRating <= 1847.50
```

```
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- sellerRating >  1847.50
|   |   |   |   |   |--- class: 0
|   |   |   |--- sellerRating >  3909.50
|   |   |   |   |--- class: 0
|   |   |--- OpenPrice >  9.89
|   |   |   |--- OpenPrice <= 11.99
|   |   |   |   |--- class: 1
|   |   |   |--- OpenPrice >  11.99
|   |   |   |   |--- sellerRating <= 2430.00
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- sellerRating >  2430.00
|   |   |   |   |   |--- OpenPrice <= 21.99
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- OpenPrice >  21.99
|   |   |   |   |   |   |--- class: 0
```



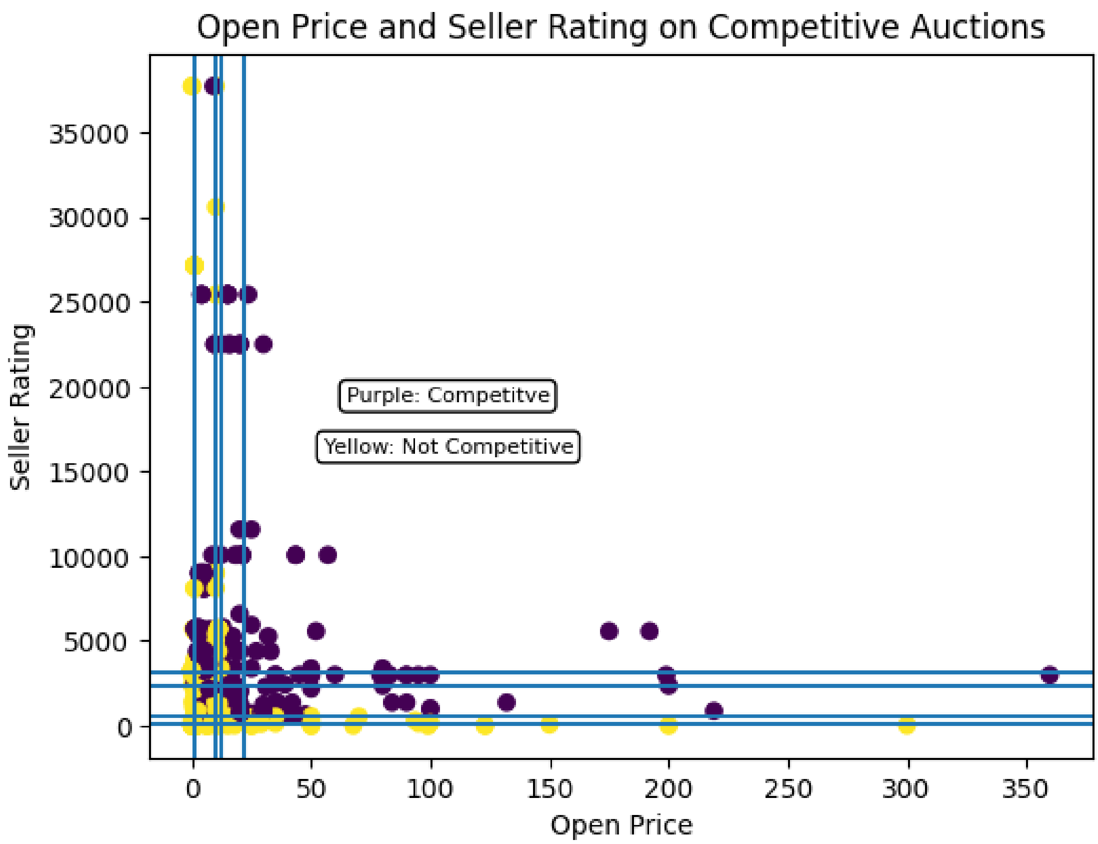
e.

Plot the resulting tree on a scatter plot: Use the two axes for the two best (quantitative) predictors. Each auction will appear as a point, with coordinates corresponding to its values on those two predictors. Use different colors or symbols to separate competitive and noncompetitive auctions. Draw lines (you can sketch these by hand or use Python) at the values that create splits.

```python
# e Plot the resulting tree on a scatter plot
preds = fullClassTree_2.predict(valid_X_2)
plt.scatter(valid_X_2['OpenPrice'], valid_X_2['sellerRating'],
c=preds)
plt.xlabel("Open Price")
plt.ylabel("Seller Rating")
```

```python
plt.axvline(x=1.3)
plt.axvline(x=9.895)
plt.axvline(x=11.995)
plt.axvline(x=21.995)

plt.axhline(y=601.5)
plt.axhline(y=128)
plt.axhline(y=3138.5)
plt.axhline(y=2365.5)

plt.annotate('Purple: Competitve', xy=(150, 20000),
             size=8, ha='right', va='top',
             bbox=dict(boxstyle='round', fc='w'))
plt.annotate('Yellow: Not Competitive', xy=(160, 17000),
             size=8, ha='right', va='top',
             bbox=dict(boxstyle='round', fc='w'))

plt.title("Open Price and Seller Rating on Competitive Auctions")
```

```
Text(0.5, 1.0, 'Open Price and Seller Rating on Competitive Auctions')
```



Does this splitting seem reasonable with respect to the meaning of the two predictors?

The splitting is overdone. There are multiple examples that show a split was overfitted with regard to the data. The splitting is correctly breaking groups apart, but is doing so more than it should be.

Does it seem to do a good job of separating the two classes?

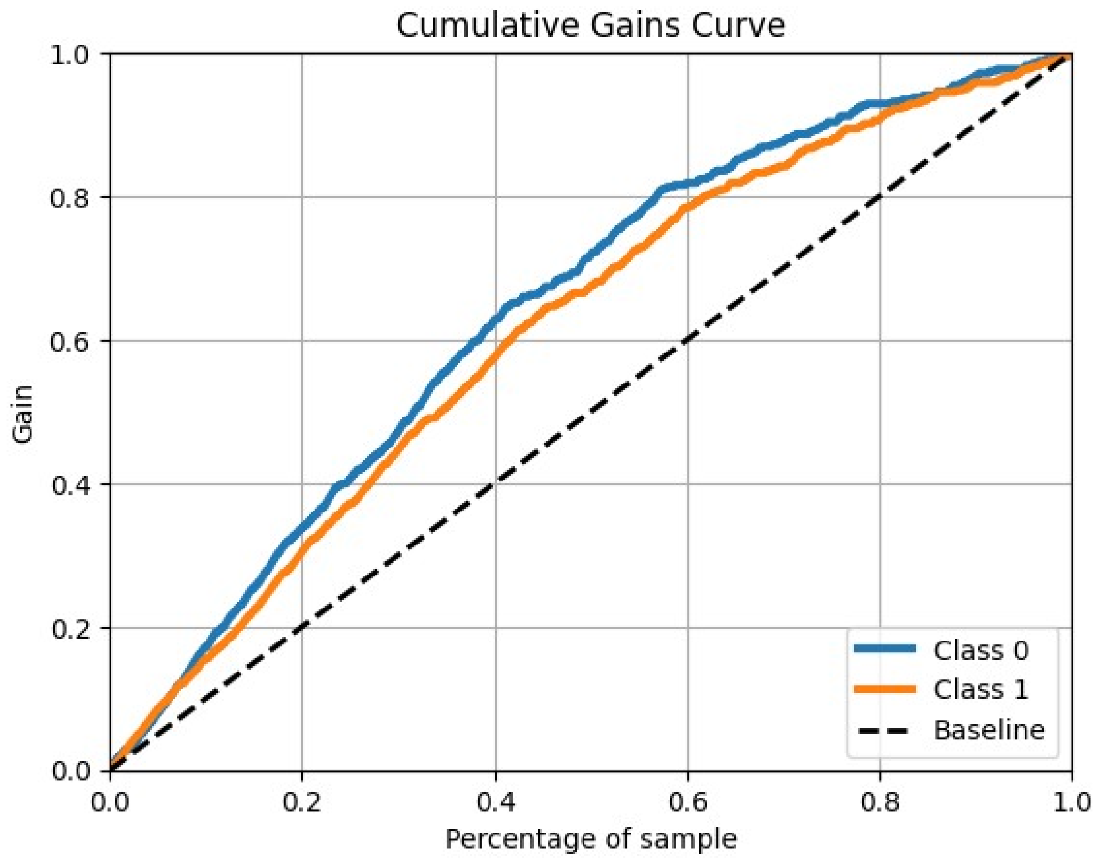
It does do a good job separating the two classes, but it does so excessively.

f.

Examine the lift chart and the confusion matrix for the tree.

```
# f lift chart and the confusion matrix
cls_sum = classificationSummary(valid_y_2,
fullClassTree_2.predict(valid_X_2))
preds = fullClassTree_2.predict_proba(valid_X_2)

skplt.metrics.plot_cumulative_gain(valid_y_2, preds)
plt.show()
```

```
Confusion Matrix (Accuracy 0.7148)

       Prediction
Actual   0   1
     0 222 131
     1  94 342
```



What can you say about the predictive performance of this model?

The confusion matrix indicates that the overall accuracy for the classification with a classification tree is 89.52%. This shows that the classification tree performs well for the data. The lift chart also indicates that this model performs well for the data.

g.

Based on this last tree, what can you conclude from these data about the chances of an auction obtaining at least two bids and its relationship to the auction settings set by the seller (duration, opening price, ending day, currency)? What would you recommend for a seller as the strategy that will most likely lead to a competitive auction?

For sellers with both a high rating and a low rating, the best strategy for a competitive auction is to have a lower OpenPrice. Starting with a higher OpenPrice, though possible for highly rated sellers, tends to result in an auction with one bid.

Problem 9.3 Predicting Prices of Used Cars (Regression Trees)

Problem Statement

The file ToyotaCorolla.csv contains the data on used cars (Toyota Corolla) on sale during late summer of 2004 in the Netherlands. It has 1436 records containing details on 38 attributes, including Price, Age, Kilometers, HP, and other specifications. The goal is to predict the price of a used Toyota Corolla based on its specifications. (The example in Section 9.7 is a subset of this dataset.)

Tasks

Data Preprocessing.

Split the data into training (60%), and validation (40%) datasets.

```
# Pre-processing
toyotaCorolla_df = pd.read_csv('ToyotaCorolla.csv')

toyotaCorolla_df = toyotaCorolla_df.rename(columns={'Age_08_04':
'Age', 'Quarterly_Tax': 'Tax'})

predictors = ['Age', 'KM', 'Fuel_Type', 'HP', 'Automatic', 'Doors',
'Tax', 'Mfr_Guarantee',
              'Guarantee_Period', 'Airco', 'Automatic_airco',
'CD_Player', 'Powered_Windows', 'Sport_Model', 'Tow_Bar']
outcome = 'Price'

X = pd.get_dummies(toyotaCorolla_df[predictors], drop_first=True)
y = toyotaCorolla_df[outcome]
train_X, valid_X, train_y, valid_y = train_test_split(X, y,
test_size=0.4, random_state=1)
```

a. Run a full-grown regression tree (RT) with outcome variable Price and predictors Age_08_04, KM, Fuel_Type (first convert to dummies), HP, Automatic, Doors, Quarterly_Tax, Mfr_Guarantee, Guarantee_Period, Airco, Automatic_airco, CD_Player, Powered_Windows, Sport_Model, and Tow_Bar. Set random_state=1.

```
# create a regressor object
regTree = DecisionTreeRegressor(random_state = 1)

# fit the regressor with X and Y data
regTree.fit(train_X, train_y)
```

```
DecisionTreeRegressor(random_state=1)
```

i. Which appear to be the three or four most important car specifications for predicting the car's price?

```
feat_importance =
regTree.tree_.compute_feature_importances(normalize=True)
```

```
X_cols = ['Age', 'KM', 'Fuel_Type', 'HP', 'Automatic', 'Doors', 'Tax',
'Mfr_Guarantee',
              'Guarantee_Period', 'Airco', 'Automatic_airco',
'CD_Player', 'Powered_Windows', 'Sport_Model', 'Tow_Bar']

x = PrettyTable(X_cols)
x.add_row(feat_importance[:-1])

print(x)

tree_rules_2 = export_text(regTree,
feature_names=list(train_X.columns))
# print(tree_rules_2)

print(X.columns)
```

```
+--------------------+--------------------+--------------------
+-----------------------+----------------------+----------------------
+----------------------+----------------------+----------------------
+----------------------+-----------------------+----------------------
+----------------------+-----------------------
+-----------------------+
|        Age         |         KM         |      Fuel_Type      |
HP          |      Automatic       |        Doors         |
Tax          |    Mfr_Guarantee     |   Guarantee_Period   |
Airco          |    Automatic_airco    |      CD_Player       |
Powered_Windows    |      Sport_Model       |        Tow_Bar          |
+--------------------+--------------------+--------------------
+-----------------------+----------------------+----------------------
+----------------------+----------------------+----------------------
+----------------------+-----------------------+----------------------
+----------------------+-----------------------
+-----------------------+
| 0.8448669121751072 | 0.0496011757060043 | 0.05378927753745229 |
0.0013335259152443414 | 0.004864085543376635 | 0.006769262083788287 |
0.003713714332944603 | 0.002384509049907461 | 0.004726710209098987 |
0.013357749456923985 | 0.0020879649883029905 | 0.00522121799417696 |
0.004458817438985015 | 0.0023446324158703778 | 1.0674008526073774e-05
|
+--------------------+--------------------+--------------------
+-----------------------+----------------------+----------------------
+----------------------+----------------------+----------------------
+----------------------+-----------------------+----------------------
+----------------------+-----------------------
+-----------------------+
Index(['Age', 'KM', 'HP', 'Automatic', 'Doors', 'Tax',
'Mfr_Guarantee',
       'Guarantee_Period', 'Airco', 'Automatic_airco', 'CD_Player',
       'Powered_Windows', 'Sport_Model', 'Tow_Bar',
```

```
'Fuel_Type_Diesel',
       'Fuel_Type_Petrol'],
      dtype='object')
```

Age, KM, Fuel Type, and HP are the most important four predictors.

ii.

Compare the prediction errors of the training and validation sets by examining their RMS error and by plotting the two boxplots. How does the predictive performance of the validation set compare to the training set? Why does this occur?

```
regressionSummary(train_y, regTree.predict(train_X))
regressionSummary(valid_y, regTree.predict(valid_X))


Regression statistics

                      Mean Error (ME) : 0.0000
       Root Mean Squared Error (RMSE) : 0.0000
            Mean Absolute Error (MAE) : 0.0000
          Mean Percentage Error (MPE) : 0.0000
Mean Absolute Percentage Error (MAPE) : 0.0000

Regression statistics

                      Mean Error (ME) : 76.6557
       Root Mean Squared Error (RMSE) : 1492.3365
            Mean Absolute Error (MAE) : 1152.4852
          Mean Percentage Error (MPE) : -0.3363
Mean Absolute Percentage Error (MAPE) : 11.3783
```

iii.

How might we achieve better validation predictive performance at the expense of training performance?

We could obtain better validation predictive performance by making the training set smaller. In this way, we can avoid overtraining and overfitting on the data.

iv. Create a smaller tree by using GridSearchCV() with cv = 5 to find a fine-tuned tree. Compared to the full-grown tree, what is the predictive performance on the validation set?

```
# user grid search to find optimized tree
param_grid = {
'max_depth': [5, 10, 15, 20, 25],
'min_impurity_decrease': [0, 0.001, 0.005, 0.01],
'min_samples_split': [10, 20, 30, 40, 50],
```

```
}

gridSearch = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=5,
n_jobs=-1)
gridSearch.fit(train_X, train_y)

print('Initial parameters: ', gridSearch.best_params_)

param_grid = {
'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
'min_impurity_decrease': [0, 0.001, 0.002, 0.003, 0.005, 0.006, 0.007,
0.008],
'min_samples_split': [14, 15, 16, 18, 20, ],
}

gridSearch = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=5,
n_jobs=-1)
gridSearch.fit(train_X, train_y)

print('Improved parameters: ', gridSearch.best_params_)
regTree = gridSearch.best_estimator_

regressionSummary(train_y, regTree.predict(train_X))
regressionSummary(valid_y, regTree.predict(valid_X))
```

```
Initial parameters:  {'max_depth': 10, 'min_impurity_decrease': 0.005,
'min_samples_split': 20}
Improved parameters:  {'max_depth': 6, 'min_impurity_decrease': 0.001,
'min_samples_split': 20}

Regression statistics

                      Mean Error (ME) : -0.0000
       Root Mean Squared Error (RMSE) : 1082.6992
            Mean Absolute Error (MAE) : 786.5953
          Mean Percentage Error (MPE) : -0.9986
Mean Absolute Percentage Error (MAPE) : 7.6224

Regression statistics

                      Mean Error (ME) : 24.8976
       Root Mean Squared Error (RMSE) : 1251.3861
            Mean Absolute Error (MAE) : 958.1684
          Mean Percentage Error (MPE) : -1.0544
Mean Absolute Percentage Error (MAPE) : 9.5594
```

The tree generated with GridSearchCV performs at a greater accuracy than the default regression tree. The Mean Error, RMSE, MAE, MPE, and MAPE are all lower than that of the default decision tree.

The tree itself is also of reduced size, and splits at different feature values.

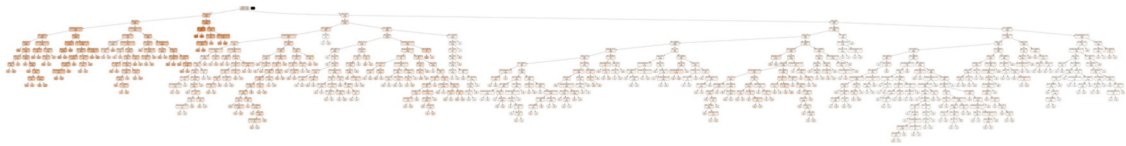
b.

Let us see the effect of turning the price variable into a categorical variable. First, create a new variable that categorizes price into 20 bins. Now repartition the data keeping Binned_Price instead of Price. Run a classification tree with the same set of input variables as in the RT, and with Binned_Price as the output variable. As in the less deep regression tree, create a smaller tree by using GridSearchCV() with cv = 5 to find a fine-tuned tree.

```
tmp_df = toyotaCorolla_df

toyota_b = toyotaCorolla_df
toyota_b['Price'] = pd.cut(tmp_df.Price, bins=20, labels=False,
include_lowest=True)

predictors = ['Age', 'KM', 'Fuel_Type', 'HP', 'Automatic', 'Doors',
'Tax', 'Mfr_Guarantee',
              'Guarantee_Period', 'Airco', 'Automatic_airco',
'CD_Player', 'Powered_Windows', 'Sport_Model', 'Tow_Bar']
outcome = 'Price'

X = pd.get_dummies(toyota_b[predictors], drop_first=True)
y = toyota_b[outcome]
train_X, valid_X, train_y, valid_y = train_test_split(X, y,
test_size=0.4, random_state=1)

# create a regressor object
regTree_2 = DecisionTreeRegressor(random_state = 1)

# fit the regressor with X and Y data
regTree_2.fit(train_X, train_y)

regressionSummary(train_y, regTree_2.predict(train_X))
regressionSummary(valid_y, regTree_2.predict(valid_X))

tree_rules = export_text(regTree_2,
feature_names=list(train_X.columns))
# print(tree_rules)

plotDecisionTree(regTree_2, feature_names=train_X.columns)
```

```
Regression statistics

               Mean Error (ME) : 0.0000
Root Mean Squared Error (RMSE) : 0.0000
     Mean Absolute Error (MAE) : 0.0000

Regression statistics
```

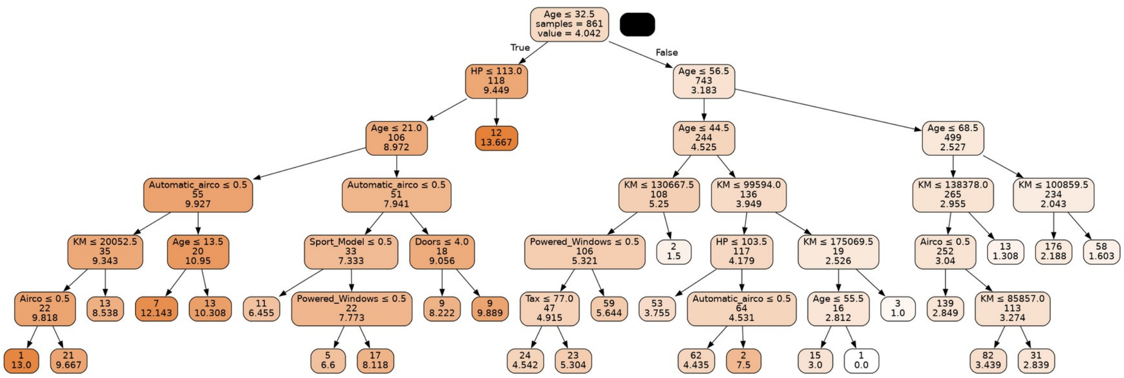
```
               Mean Error (ME) : 0.0626
Root Mean Squared Error (RMSE) : 1.0616
     Mean Absolute Error (MAE) : 0.7304
```



```
# user grid search to find optimized tree
param_grid = {
'max_depth': [5, 10, 15, 20, 25],
'min_impurity_decrease': [0, 0.001, 0.005, 0.01],
'min_samples_split': [10, 20, 30, 40, 50],
}

gridSearch = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=5,
n_jobs=-1)
gridSearch.fit(train_X, train_y)

print('Initial parameters: ', gridSearch.best_params_)

param_grid = {
'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
'min_impurity_decrease': [0, 0.001, 0.002, 0.003, 0.005, 0.006, 0.007,
0.008],
'min_samples_split': [14, 15, 16, 18, 20, ],
}

gridSearch = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=5,
n_jobs=-1)
gridSearch.fit(train_X, train_y)

print('Improved parameters: ', gridSearch.best_params_)
regTree = gridSearch.best_estimator_

regressionSummary(train_y, regTree.predict(train_X))
regressionSummary(valid_y, regTree.predict(valid_X))

tree_rules = export_text(regTree, feature_names=list(train_X.columns))
# print(tree_rules)

plotDecisionTree(regTree, feature_names=train_X.columns)
```

```
Initial parameters:  {'max_depth': 10, 'min_impurity_decrease': 0.01,
'min_samples_split': 20}
Improved parameters:  {'max_depth': 6, 'min_impurity_decrease': 0.007,
'min_samples_split': 16}
```

```
Regression statistics

                      Mean Error (ME) : -0.0000
Root Mean Squared Error (RMSE) : 0.7908
     Mean Absolute Error (MAE) : 0.6028

Regression statistics

                      Mean Error (ME) : 0.0206
Root Mean Squared Error (RMSE) : 0.8938
     Mean Absolute Error (MAE) : 0.6858
```



i. Compare the smaller tree generated by the CT with the smaller tree generated by RT. Are they different? (Look at structure, the top predictors, size of tree, etc.) Why?

```
# i
# RT
feat_importance =
regTree_2.tree_.compute_feature_importances(normalize=True)
x = PrettyTable(list(X.columns))
x.add_row(feat_importance)

print(x)

# GridSearchCV
feat_importance =
regTree.tree_.compute_feature_importances(normalize=True)
x = PrettyTable(list(X.columns))
x.add_row(feat_importance)

print(x)

+--------------------+--------------------+--------------------
+----------------------+----------------------+----------------------
+---------------------+-----------------------+----------------------
+---------------------+-----------------------+--------------------
+---------------------+----------------------
+----------------------+----------------------+
|         Age          |          KM          |          HP          |
```

```
Automatic       |       Doors         |         Tax          |
Mfr_Guarantee     |    Guarantee_Period   |        Airco         |
Automatic_airco    |       CD_Player       |   Powered_Windows   |
Sport_Model      |        Tow_Bar        |    Fuel_Type_Diesel   |
Fuel_Type_Petrol   |
+--------------------+---------------------+--------------------
+----------------------+----------------------+----------------------
+----------------------+-----------------------+---------------------
+----------------------+-----------------------+--------------------
+----------------------+----------------------
+----------------------+----------------------+
| 0.8324504437721553 | 0.06403826382530488 | 0.04564028022287883 |
0.0013681638642594816 | 0.007200288617493181 | 0.005658103141462496 |
0.004525492609570666 | 0.0026908925202734067 | 0.005357362376023264 |
0.014317289565787894 | 0.0016447160497330261 | 0.00531577694686054 |
0.004356466603013246 | 0.0024978869898324403 | 0.0011900190723459117 |
0.0017485538230054554 |
+--------------------+---------------------+--------------------
+----------------------+----------------------+----------------------
+----------------------+-----------------------+---------------------
+----------------------+-----------------------+--------------------
+----------------------+----------------------
+----------------------+----------------------+
+--------------------+---------------------+-------------------
+-----------+----------------------+--------------------
+--------------+------------------+-------------------
+--------------------+-----------+--------------------
+----------------------+---------+----------------
+------------------+
|        Age         |          KM         |         HP          |
Automatic |         Doors          |         Tax           |
Mfr_Guarantee | Guarantee_Period |        Airco         |
Automatic_airco   | CD_Player |   Powered_Windows    |
Sport_Model      | Tow_Bar | Fuel_Type_Diesel | Fuel_Type_Petrol |
+--------------------+---------------------+-------------------
+-----------+----------------------+--------------------
+--------------+------------------+-------------------
+--------------------+-----------+--------------------
+----------------------+---------+----------------
+------------------+
| 0.8969861338212077 | 0.02790192635705718 | 0.0458962281652297 |
0.0    | 0.0022489763462299377 | 0.001229142203460168 |      0.0
|       0.0        | 0.003937745613259301 | 0.01540363351498619 |
0.0    | 0.004103621121382595 | 0.0022925928571871216 |   0.0   |
0.0        |       0.0        |
+--------------------+---------------------+-------------------
+-----------+----------------------+--------------------
+--------------+------------------+-------------------
+--------------------+-----------+--------------------
```

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

Top Predictors

Full-tree: Age, KM, HP, Automatic
GridSearchCV Age, KM, HP, Automatic

Structure

Rather than being a tree that is heavily leaned, using bins leads to a tree that appears relatively balanced.

Size

Using bins leads to a tree that is significantly smaller. The plots shown above display the large visual difference in size between the two trees.

Explain why

Using bins reduces the number of variables. Since price is now sorted into a finite number of categorical bins rather than a continuous value, the number of variables can be greatly reduced to fit the outcome prediction.

ii. Predict the price, using the smaller RT and CT, of a used Toyota Corolla with the specifications listed in Table 9.10.

```
TABLE 9.10 SPECIFICATIONS FOR A PARTICULAR TOYOTA COROLLA
Variable            Value
Age_-08_-04         77
KM                  117,000
Fuel_Type           Petrol
HP                  110
Automatic           No
Doors               5
Quarterly_Tax       100
Mfg_Guarantee       No
Guarantee_Period    3
Airco               Yes
Automatic_airco     No
CD_Player           No
Powered_Windows     No
Sport_Model         No
Tow_Bar             Yes
```

```
# ii. Predict the price, using the smaller RT and CT, of a used Toyota
Corolla with the specifications listed in Table 9.10.
sample_car = pd.DataFrame(columns=X.columns)
print(len(X.columns))

sample_car.loc[0] = [77, 117000, 0, 110, 0, 5, 100, 0, 3, 1, 0, 0, 0,
0, 1, 0]
```

```
fullClassTree = DecisionTreeClassifier(random_state=1,
min_samples_leaf=50, max_depth=7)
fullClassTree.fit(train_X, train_y)

RT_pred = regTree.predict(sample_car)
CT_pred = fullClassTree.predict(sample_car)

print(RT_pred)
print(CT_pred)
```

```
16
[1.60344828]
[1]
```

iii Compare the predictions in terms of the predictors that were used, the magnitude of the difference between the two predictions, and the advantages and disadvantages of the two methods.

Regression Tree with bins: $7,125 Classification Tree with bins: $7,950

The predictions obtained from the two trees are 10.9453% different. This value is not largely significant, but it is not considered insignificant.

In this instance, the Regression Tree performed better for this set of data since it was better trained. Our regression model made use of GridSearchCV to find parameters that functioned well for this given set. This is opposed to the Classification Tree, which did not have any additional tuning.

Each tree functions in different ways, and the application and underlying data will determine which tree is best for which situation.