CS 412 – Car Price Prediction
Project Report - Group #1

Group Members:

Ulaş Eraslan - 25058

Zeynep Tandoğan - 25200

Duygu Tümer - 24842

İrem Gürak - 25442

Ömer Cem Tabar - 25031

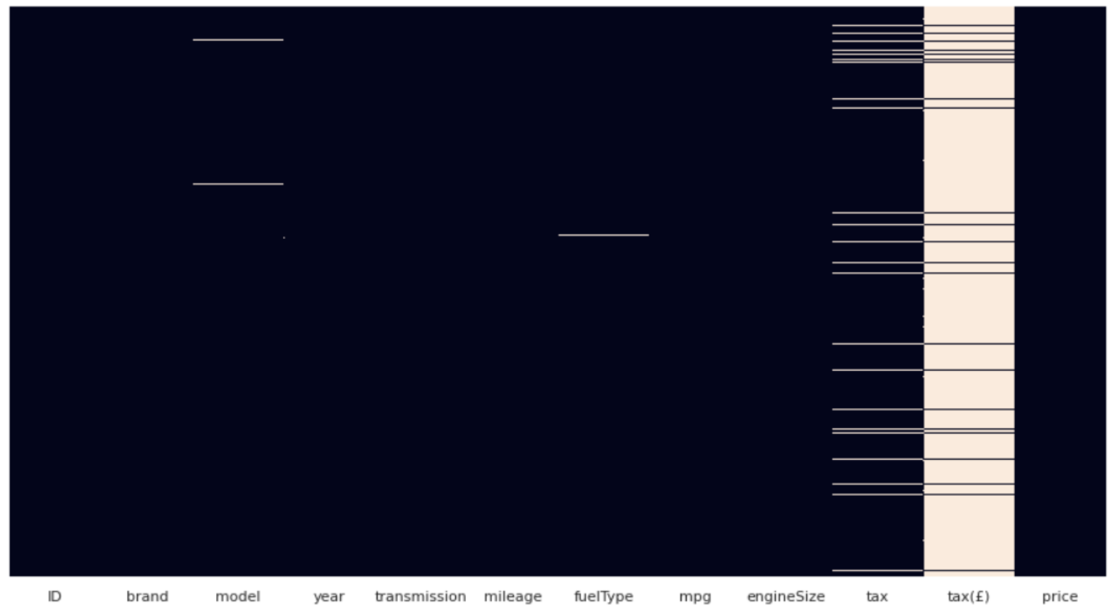
Sabancı University, 2022

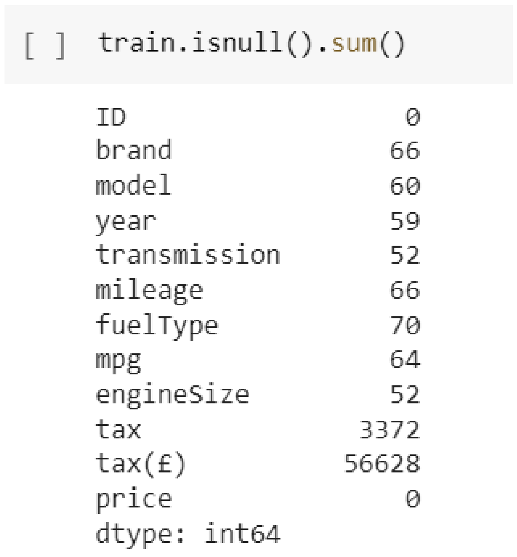
1) Exploratory Data Analysis and Data Preprocessing

In this project, it is desired to predict car prices in given data. For this, the given train.csv data was analyzed first. The columns of data were analyzed. Shape was examined. It was seen that there are 12 features and information of 60,000 cars. Then, the number of missing data in the information of each feature was analyzed.

```
[ ]  sns.set(rc = {'figure.figsize':(15,8)})
     sns.heatmap(train.isnull(),yticklabels=False,cbar=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f302c72afd0>
```





```
[ ]  train.isnull().sum()

     ID                 0
     brand             66
     model             60
     year              59
     transmission      52
     mileage           66
     fuelType          70
     mpg               64
     engineSize        52
     tax             3372
     tax(£)         56628
     price              0
     dtype: int64
```

<u>**Pre-processing Phase of the Given Data**</u>

Before the models are created and the price estimation phase is started, although the train data set that is given to us have appear as a whole, many data improvement steps have been applied on a column-by-column basis in order to compensate for the data losses under each column and to build our models on more solid foundations. If we need to mention the steps that is applied in order to organize and fill the missing data in detail within the train data set , we can list the preprocessing steps as follows:

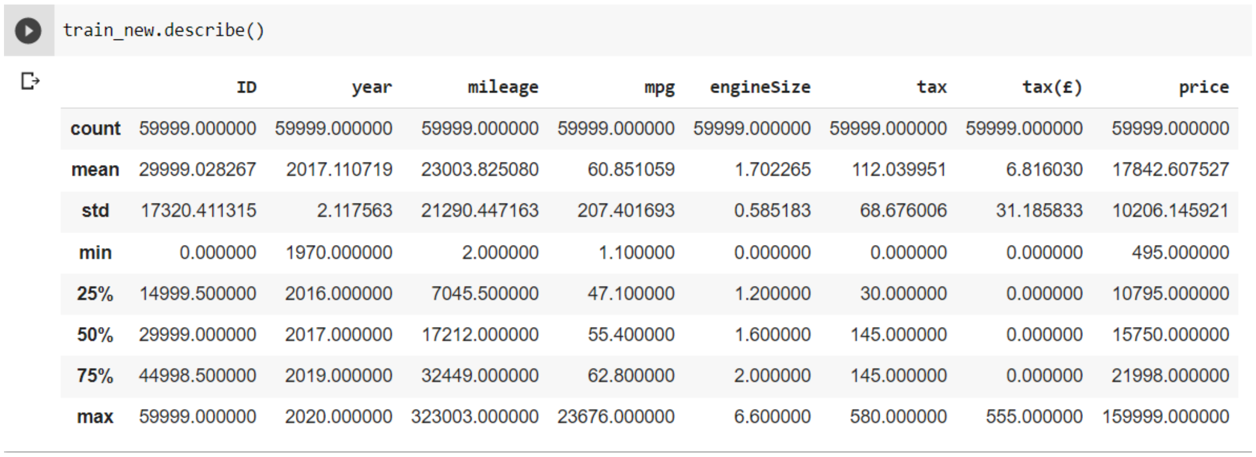
- The first thing we noticed on the data set was that there were many missing data entries in the **"brand"** and **"model"** columns. We have used our carefully created dictionaries in order to be able to fill the missing values in the model column with mapping from brand to model and missing values in the brand column with mapping from model to brand. In detail, the filling and arrangement of these two columns was as follows: For the missing values in the brand column, by using the matches between the brand and model column, a required dictionary was created and the brand corresponding to the model was written according to one-to-one correspondence within the dictionary. Then, the missing values in the model column were filled accordingly with the matches between the brand and model columns, again with the help of a dictionary that has been created, with additional attribute as occurrence count, and the most repetitive potential model corresponding to the brand was used to fill the gaps.

- Secondly, we continued with filling the missing values in the tax columns which are **"tax"** and **"tax(£)"**. Within the tax columns that are determined as dollar and pound wise we observed that if tax was taken on dollar basis, the pound part was blank and if the tax was based on pounds, the dollar part was left blank. Hence, missing values that exist in both columns are eliminated by filling the empty values with 0.
- In order to fill the missing values in the column containing the **"transmission"** information, occurrences about the transmission information that corresponds to the brand and model relationship were examined. The most occured transmission mechanism related to the specific brand and model was determined and the required transmission part was filled in accordance with brand-model-transmission relationship.
- For filling the missing values in the **"mileage"** column, correlation values are examined from the confusion matrix that again enables us to come up with a correlation based formula that reflects the negative correlation between mileage-price and mileage- year which results in an output that is rounded to fill the corresponding missing value of mileage.
- For filling the missing values in the **"mpg"** column, correlation values are examined from the confusion matrix that again enables us to come up with a correlation based formula that reflects the positive correlation between mpg and mileage, and negative

correlation between mpg and tax(£) which resulted in an output that is rounded to fill the corresponding missing value of mpg.

- For filling the missing values that exist in the **“year”** column, most occurred years that correspond to the brand and model relationship were examined. The most occured year related to the specific brand and model was determined and the required year part was filled in accordance with brand-model-year relationship.
- For filling the missing values that exist in the **“fuelType”** column, most occurred fuel types that correspond to the brand and model relationship were examined. The most occured fuel type related to the specific brand and model was determined and the required fuel type part was filled in accordance with brand-model-fuelType relationship.
- For filling the missing values that exist in the **“engineSize”** column, occurred engine sizes to the brand and model relationship were examined. From the occurred engine size, the average of the engine size related to the specific brand and model was calculated and the required engine size part was filled in accordance with brand-model-engineSize relationship.
- **You can access the Google Collaboratory link of the source code for Preprocessing part below.**

After dealing with null values, exploratory data analysis is performed, to have an idea about how certain features affect the target(price). Below, there are some figures acquired to see if there are any needs for feature engineering.



```
train_new.describe()
```

```
                 ID          year        mileage           mpg    engineSize           tax        tax(£)          price
count  59999.000000  59999.000000   59999.000000  59999.000000  59999.000000  59999.000000  59999.000000   59999.000000
mean   29999.028267   2017.110719   23003.825080     60.851059      1.702265    112.039951      6.816030   17842.607527
std    17320.411315      2.117563   21290.447163    207.401693      0.585183     68.676006     31.185833   10206.145921
min        0.000000   1970.000000       2.000000      1.100000      0.000000      0.000000      0.000000     495.000000
25%    14999.500000   2016.000000    7045.500000     47.100000      1.200000     30.000000      0.000000   10795.000000
50%    29999.000000   2017.000000   17212.000000     55.400000      1.600000    145.000000      0.000000   15750.000000
75%    44998.500000   2019.000000   32449.000000     62.800000      2.000000    145.000000      0.000000   21998.000000
max    59999.000000   2020.000000  323003.000000  23676.000000      6.600000    580.000000    555.000000  159999.000000
```

Figure 1:

In Figure 1, general insights are extracted from our newly generated data, and some of the information mentioned there is used while obtaining below graphs.

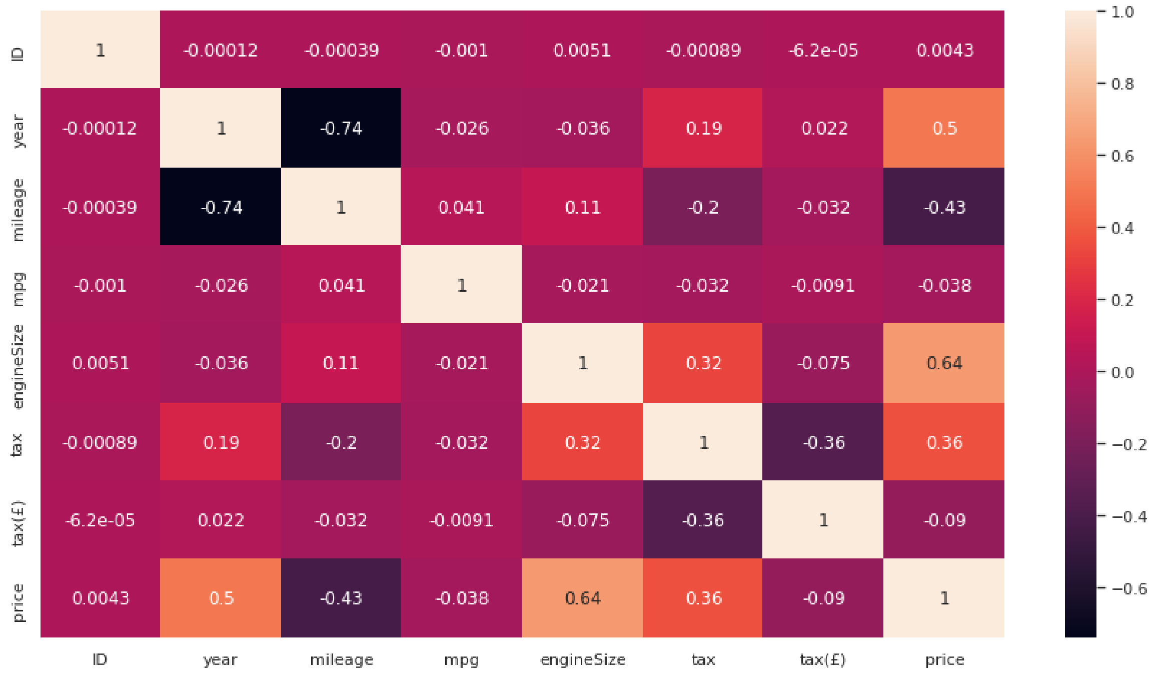


Figure 2: Confusion Matrix

In Figure 2, there is a confusion matrix. There are some features missing due to object types, encoding is done during model construction and after, a feature importance graph is obtained (see Figure 3)

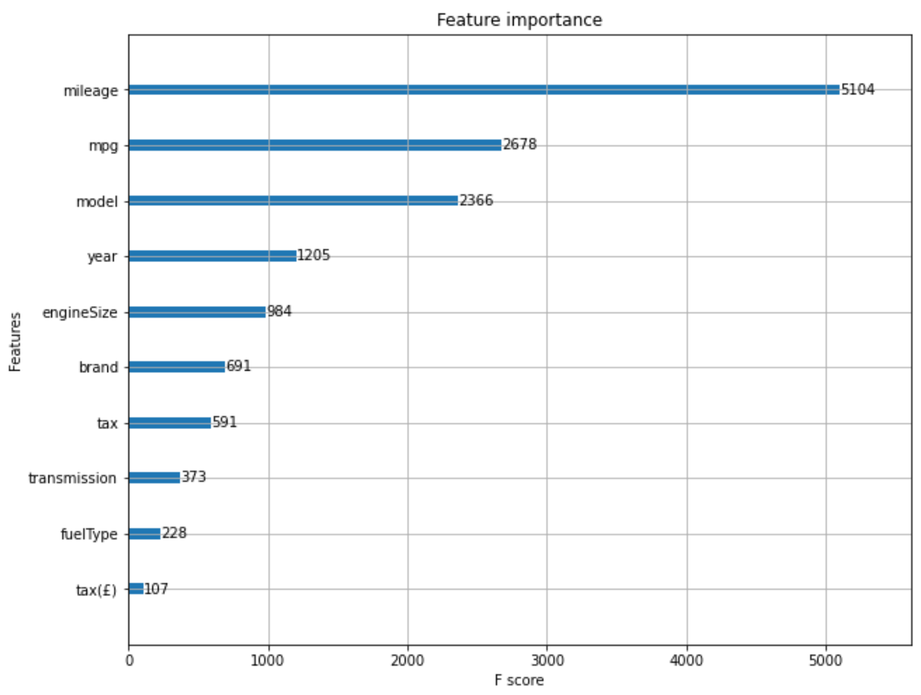
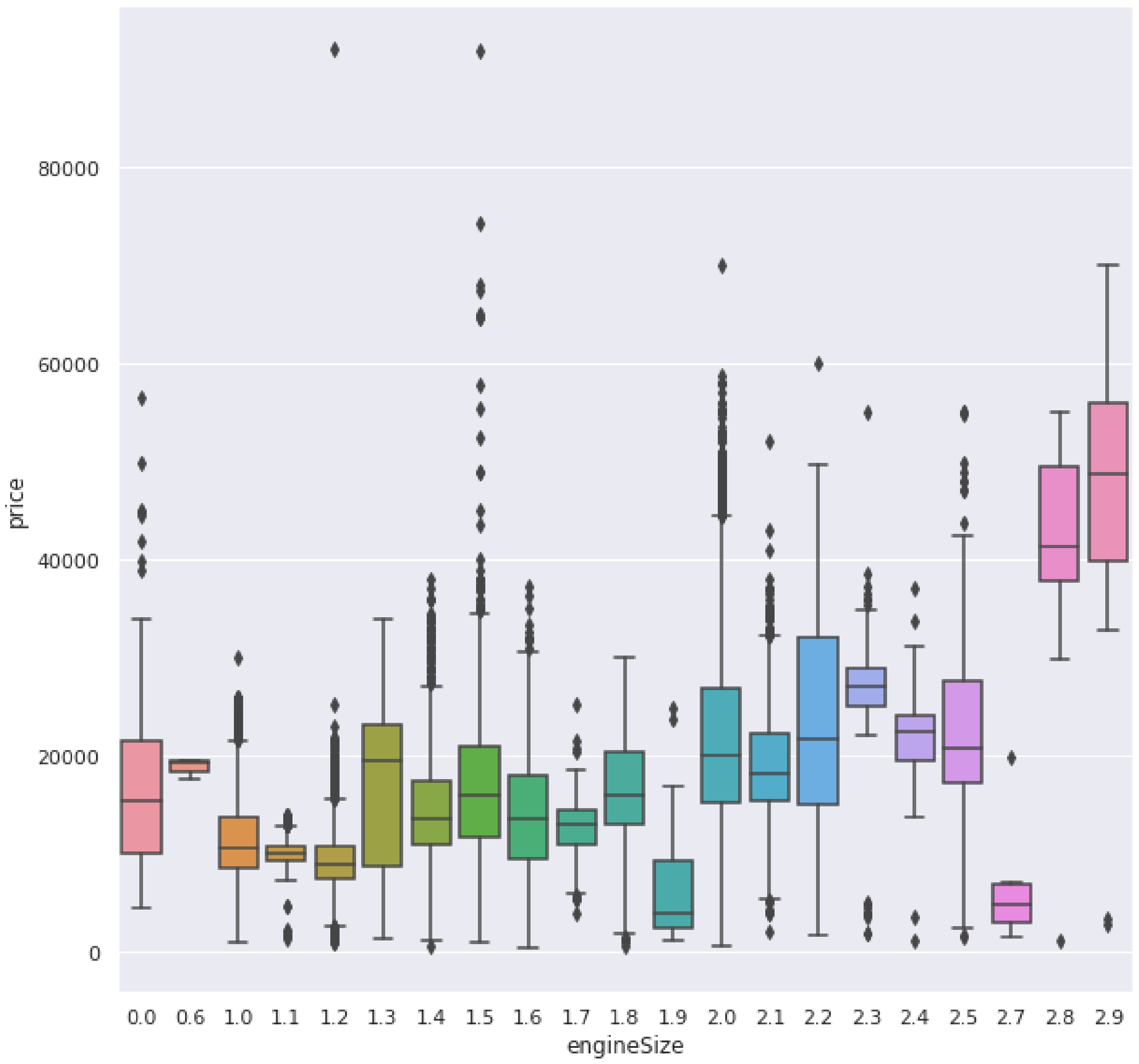


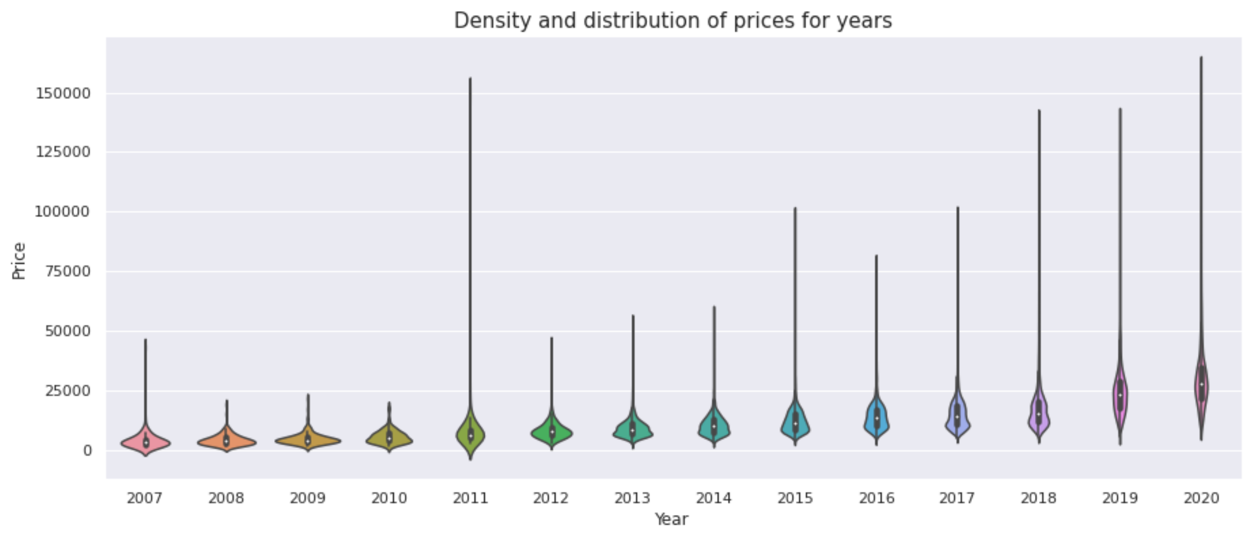
Figure 3: Feature Importance

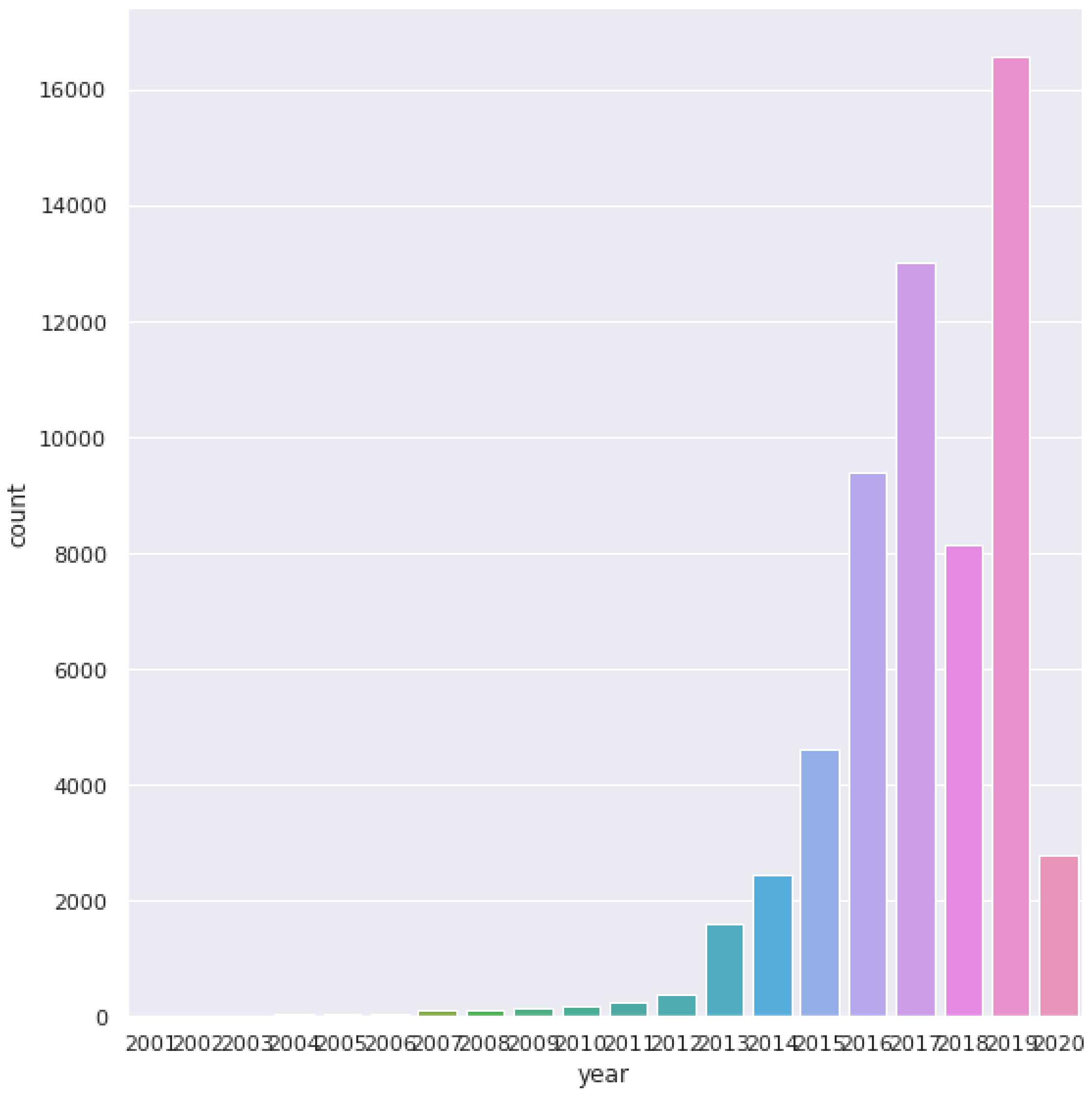
According to the confusion matrix and the feature importance graph, some features have been examined. First, since the confusion matrix showed a positive correlation between price and engine size (%64), I wanted to see the relationship between them in a graph. I used a box plot to see how the price range changes with different engine sizes. The engine size is restricted to below 3, since as it can be seen from Figure 1, mean value was closer to the minimum value and this meant that there were outliers, so it is restricted to see a cleaner graph. The graph can be seen in Figure 4.



As it can be seen from the above graph, there are no direct effects of engine size to price, but still it can be said that in general, as the engine size increases, the mean of the prices is also increased.

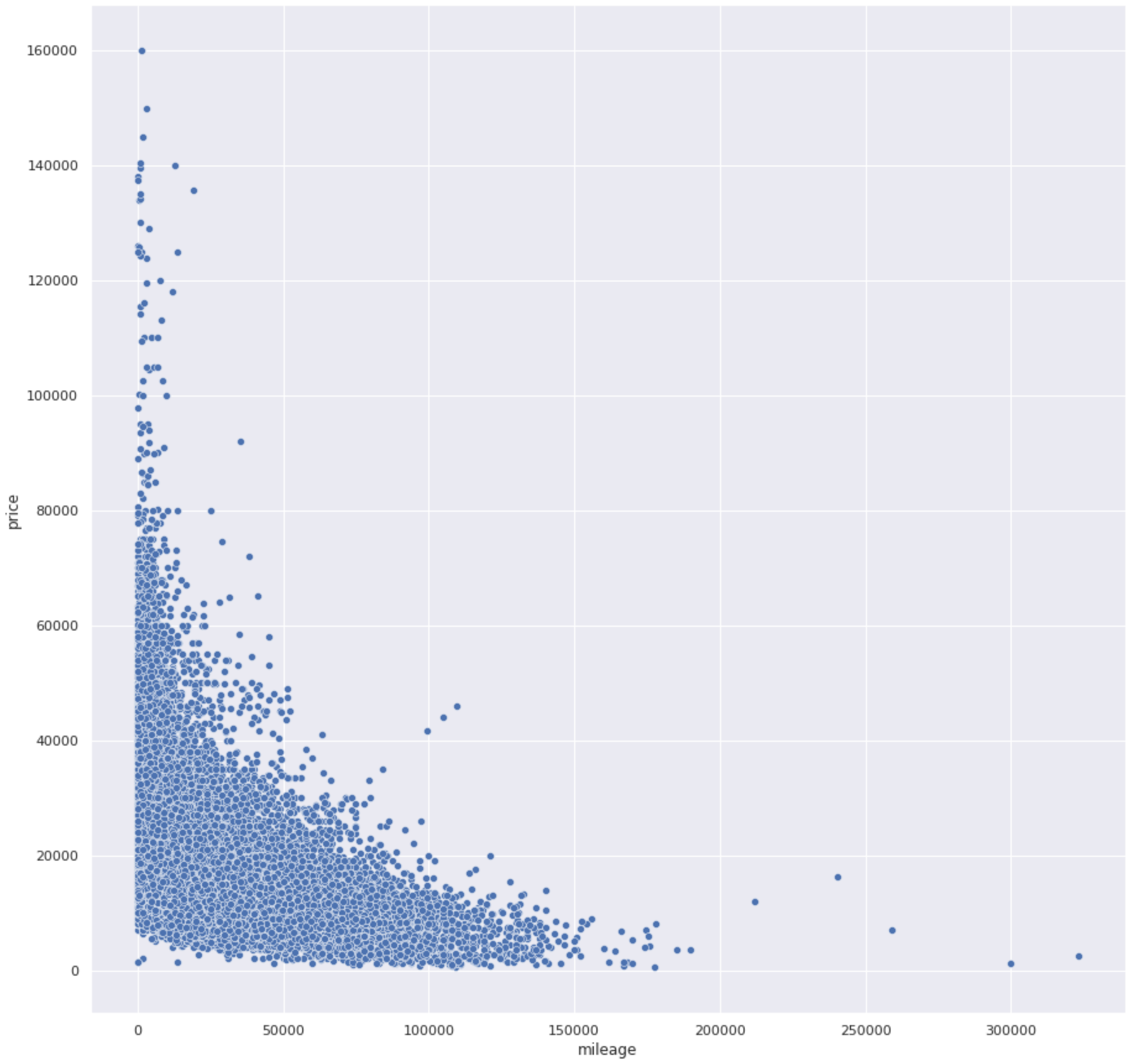
Second highest correlation was detected to be between year and price (%50). In Figure 5 and 6, this correlation is examined.


Density and distribution of prices for years
Price
150000
125000
100000
75000
50000
25000
0
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
Year


16000
14000
12000
10000
8000
6000
4000
2000
0
count
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
year

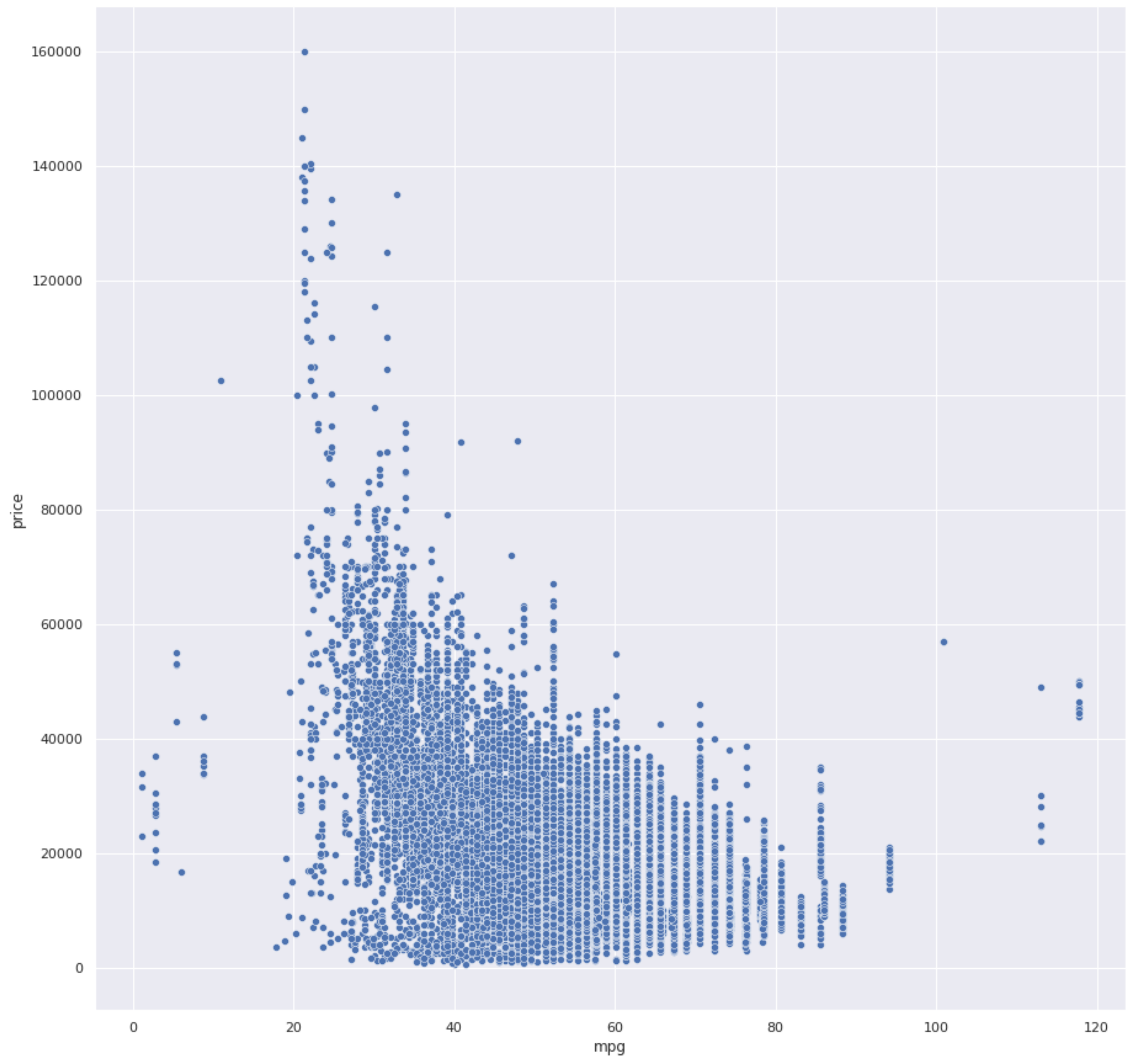
A violin graph is used to show the relationship between year and price and the year feature is restricted between 2007 and 2020, since in Figure 6, it can be seen that there are not many cars belonging to earlier years. From Figure 5, it can be understood that, except the year 2011, the range of the prices is increasing as the year of the car becomes closer to today's date.

After this correlation, the mileage feature is examined with a scatter plot, the output can be seen in Figure 7.

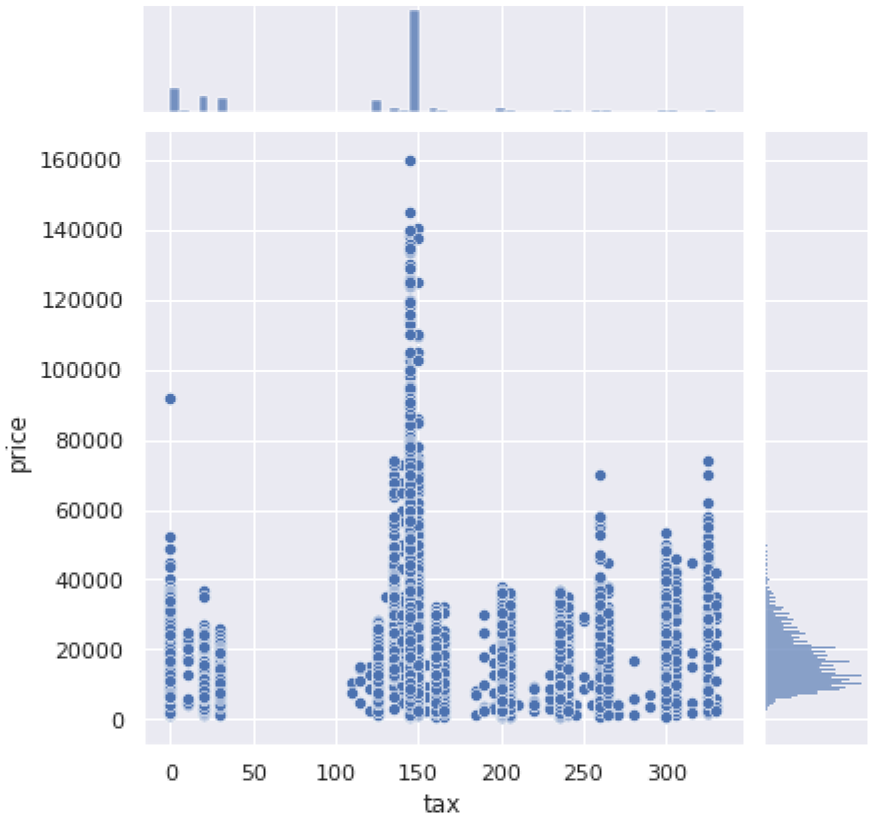


As the correlation matrix indicated, there is a slight negative correlation observed between the mileage and the price.

Afterwards, the mpg and tax features are taken into consideration, because the importance graph has shown the mpg as an important feature, and tax feature was the only remaining integer based column to be examined.

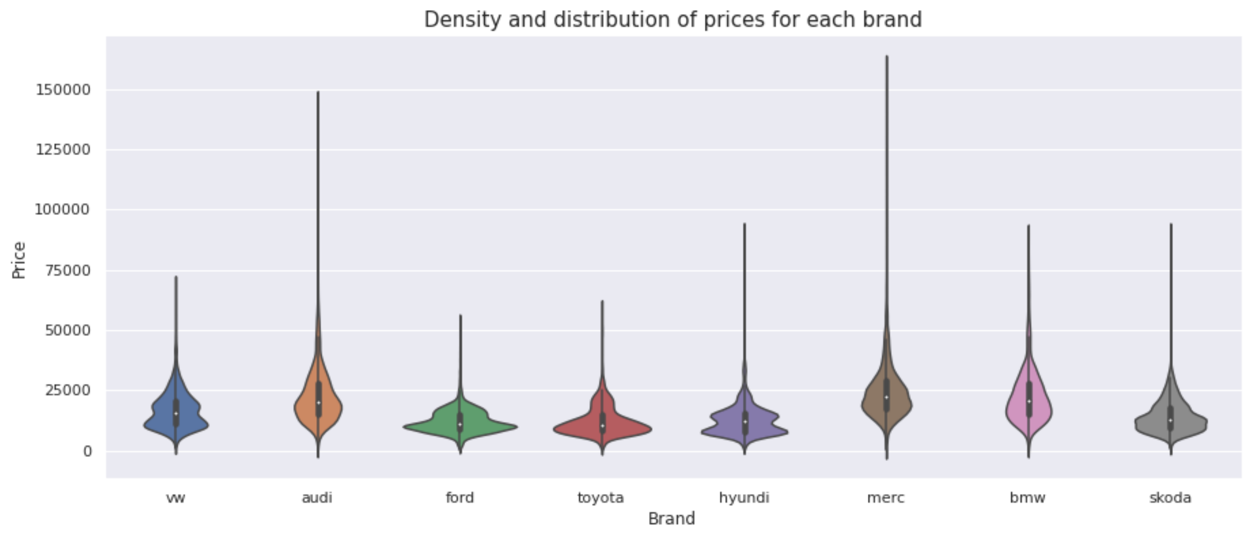


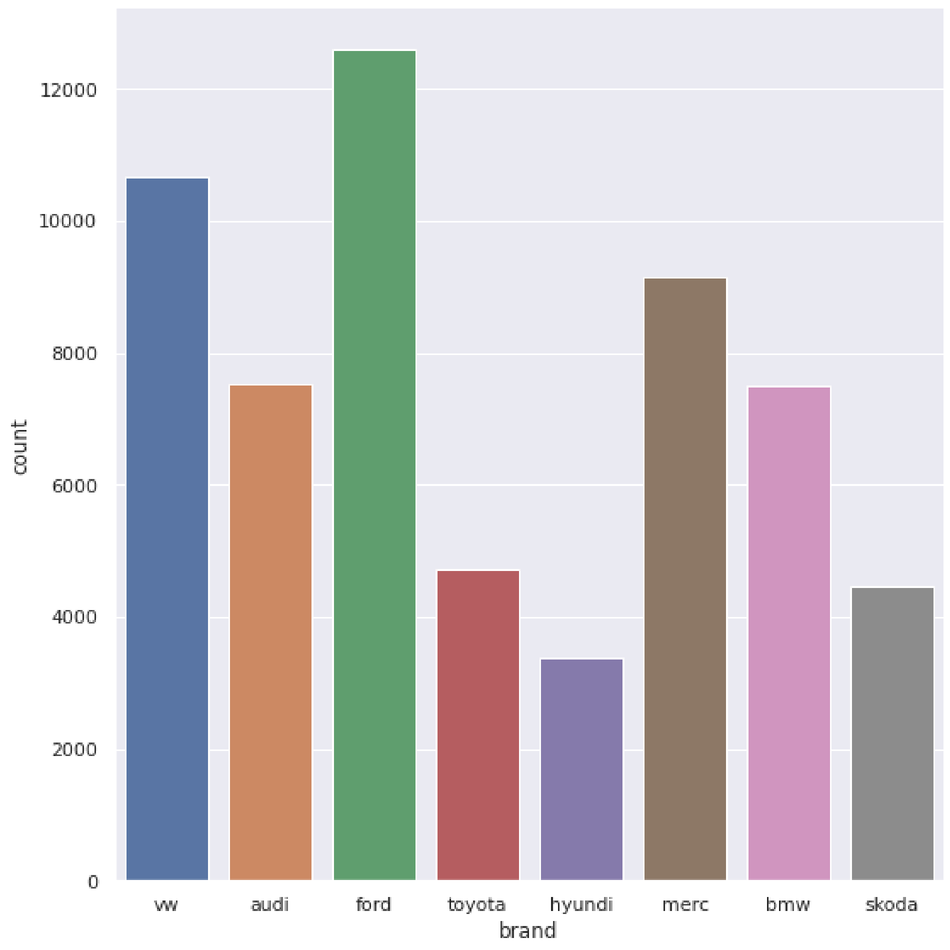
There was not any clear correlation observed first since there were outliers, so again looking at the mean value, mpg value is restricted. As it is seen in Figure 8,it can be said that the price and mpg had a slightly negative correlation.



Here, tax feature is examined but there could not be found any clear correlation against the target.

Lastly, brand feature is taken into account. The “model” feature was seen to be more important than the brand in the feature importance graph, but as mentioned in the preprocessing part, the model and the brand columns were correlated, so to avoid multiple columns in the graph (there were too many models), brand feature is examined. In Figure 10, the Mercedes brand is observed to have a bigger range of pricing compared to other brands, and the Ford brand is observed to have a smaller range of pricing. Further, in Ford, the prices were generally distributed around low prices and the count of the ford cars was the highest. And this showed that there was a lot cheap priced cars in the data.





Before starting to implement the models, it is required to convert the categorical attributes which are brand, model, transmission and fuelType to numeric values. This is a must, since the problem is a regression problem. There are many different encoding techniques, in this project, three of them are considered and implemented to see which is the best encoding technique for the car price prediction problem. These are;

- **Target Encoding:** It replaces a feature's categories with some number derived from the target. The most common practice is to compute the mean for each class group and put these values instead of the categorical attributes. Whatever coding is applied to the training set should also be applied to the test set. Since the encoding method is required to know the target value, this can not be applied to the test set. For this reason, it is not used in our project.
- **Label Encoding:** It assigns ordinal integers to different categorical levels of categorical variables. We have tried this encoding with the models that are implemented; however, when the errors received were compared, it was observed that one hot encoding was much more successful in terms of contributing to performance of the model. The problem of the label encoding is that since it gives different values in the same column, the model will misunderstand the data such that the one that has the higher value is more important than the others. This problem is solved in one hot encoding.
- **One Hot Encoding:** In this encoding, the categorical column is taken and each categorical value of that column is separated into columns. In other words, the categorical column is splitted into multiple columns. Since each column is filled with 1 and 0s, the model does not misinterpret the data. In our implementation, we used this encoding method for categorical attributes.

2) Model Implementations

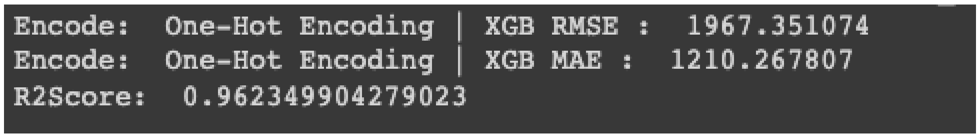
2.1 RFE with Regression

After filling some missing values on the training data, Random Forest (RFE) with regression is implemented which you can access the source code from the Google Collaboratory link below.

Even though the mean squared error was pretty good for training data sets, it turned out very bad for validation. Therefore, since we could not reach the desired accuracy, a different model from this model was tried.

2.2 XGBoost Regression and AutoML(AutoSklearn - TPOT)

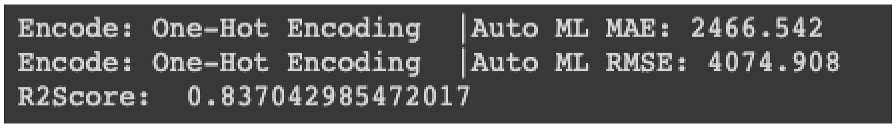
Following the trial of Random Forest model implementation, XGBoost(XGB) Regressor which stands for Extreme Gradient Boosting Regression was used. The reason to use this model was based on the high predictability of ensemble methods that consist of aggregation of different results at the same place. The XGB Regressor model has different hyperparameters in itself hence a good fine tuning may increase the accuracy and R^2 value of the model results. To realize this objective, a hyperparameter tuning based on the GridSearch method was applied on various parameters such as learning rate, maximum tree depth, number of estimators, etc. Hyperparameter tuning outcomes state that the best hyperparameters that give the highest accuracy (lowest mean-absolute error and root mean square error) are the following ones: n_estimators = 1300, learning_rate=0.50, max_depth=4, colsample_bytree = 1. Therefore, the XGBoost model was established with these parameters and trained on the given dataset which was splitted as train and validation data sets with 80% and 20% ratios respectively. Then, the model was evaluated with test data based on mean absolute error, root mean squared error and R^2 metrics. Model results are presented in the image below.



```
Encode:  One-Hot Encoding | XGB RMSE :  1967.351074
Encode:  One-Hot Encoding | XGB MAE :  1210.267807
R2Score:  0.962349904279023
```

Code Block 1: XGBoost Regressor Model Results

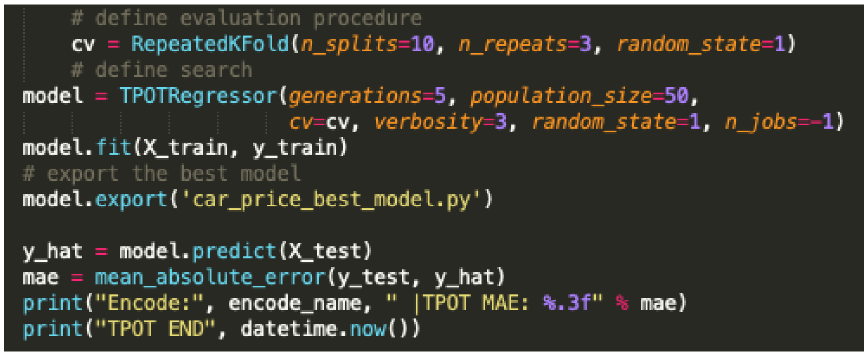
Even though the results are quite satisfactory, other methods to construct an alternative model were considered. In this regard, 2 different AutoML tools were used. The first tool was AutoSklearn which is an automated machine learning tool that generates the best model with best hyperparameters by considering the given dataset's features. The model results which were produced by the AutoSklearn model were fairly bad as compared to the XGBoost Regressor's results. Hence, this method was skipped.



```
Encode: One-Hot Encoding  |Auto ML MAE: 2466.542
Encode: One-Hot Encoding  |Auto ML RMSE: 4074.908
R2Score:  0.837042985472017
```

Code Block 2: AutoSklearn Model Results

The second AutoML tool that was used was TPOT which stands for Tree Based Pipeline Optimization Tool. This tool exhaustively evaluates different models based on the scoring method which is the root mean squared error and finds the most suitable model with best hyperparameters. The code block below represents the implementation of this tool.

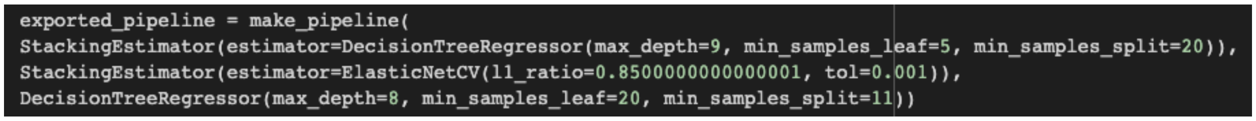


```
    # define evaluation procedure
    cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
    # define search
model = TPOTRegressor(generations=5, population_size=50,
                      cv=cv, verbosity=3, random_state=1, n_jobs=-1)
model.fit(X_train, y_train)
# export the best model
model.export('car_price_best_model.py')

y_hat = model.predict(X_test)
mae = mean_absolute_error(y_test, y_hat)
print("Encode:", encode_name, " |TPOT MAE: %.3f" % mae)
print("TPOT END", datetime.now())
```

Code Block 3: TPOT Model Implementation

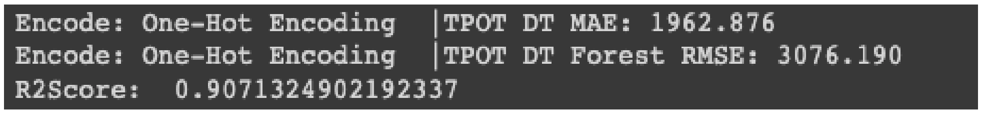
One of the downsides of this method is that it requires a huge amount of time to compute all the different models with various hyperparameter values. In our case, it takes around 5 hours to complete the process. The final recommendation of this tool was the following model:



```
exported_pipeline = make_pipeline(
StackingEstimator(estimator=DecisionTreeRegressor(max_depth=9, min_samples_leaf=5, min_samples_split=20)),
StackingEstimator(estimator=ElasticNetCV(l1_ratio=0.8500000000000001, tol=0.001)),
DecisionTreeRegressor(max_depth=8, min_samples_leaf=20, min_samples_split=11))
```

Code Block 4: TPOT Recommended Model 1

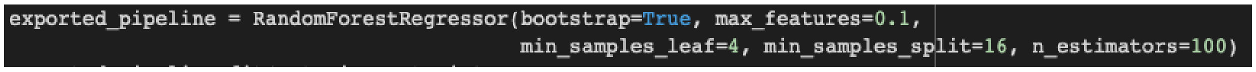
The recommended model is a stacked machine learning that consists of two Decision Tree Regressor with different hyperparameters and one ElasticNetCV model. The test results of this model presented below.



```
Encode: One-Hot Encoding   |TPOT DT MAE: 1962.876
Encode: One-Hot Encoding   |TPOT DT Forest RMSE: 3076.190
R2Score:  0.9071324902192337
```

Code Block 5: TPOT Recommended Model Results 1

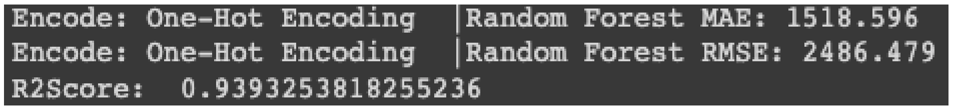
The test results are better as compared to the AutoSklearn model yet the TPOT model is still not a better alternative for the XGBoost model. Following this observation, another run has been taken from the TPOT. At the second run, another recommendation was given which was the following Random Forest Tree model.



```
exported_pipeline = RandomForestRegressor(bootstrap=True, max_features=0.1,
                                          min_samples_leaf=4, min_samples_split=16, n_estimators=100)
```

Code Block 6: TPOT Recommended Model 2

Next, the second model tested on the test data. Results are as follows.



```
Encode: One-Hot Encoding   |Random Forest MAE: 1518.596
Encode: One-Hot Encoding   |Random Forest RMSE: 2486.479
R2Score:  0.9393253818255236
```

Code Block 7: TPOT Recommended Model Results 2

As indicated above, the model results are better than the previously recommended TPOT model. However, it is still worse as compared to the XGBoost model.

Based on the results of all different models and methodologies, One-Hot Encoded dataset with XGBoost Regressor was used as the main model for solving the price prediction problem. Related implementations and test results can be found in the following section under the Main Model Implementation part.

Google Collaboratory Links for the Codes:

- Preprocessing Part Collab Link: https://colab.research.google.com/drive/1E3E6HQ2_DBH2qxfCh0gGa8Wuk9ypkODp?usp=sharing
- RFE with Regression Collab Link: https://colab.research.google.com/drive/1XFbFQP_iluLRetdzigWeyeAoUnKtn-3X?usp=sharing
- Main Model Implementation (XGBoost & AutoML(AutoSklearn - TPOT)) https://colab.research.google.com/drive/1X1QWcH4EfQAIAPCK-QgkzxBxSNgZfm0r?usp=sharing