Manchester Metropolitan University

NAME:

RICHARD ADEMOLA OGUNDELE

STUDENT CANDIDATE NUMBER:

22539744

COURSE:

MACHINE LEARNING CONCEPTS

INDIVIDUAL ASSESSMENT

JANUARY 2023

Overview

Packages/Tools Used

- Python
- Pandas
- Jupyter notebook
- NumPy
- Matplotlib
- Seaborn
- Scikit-Learn

In this assignment we have the opportunity to consolidate our learning in the Machine Learning Concepts unit by creating a working solution for a real-world, industry-based machine learning problem. It involves typical tasks such as data understanding, exploration, cleaning, and preparation; and machine learning model selection, tuning, fitting, evaluation, and analysis. More specifically, we will work with a Car Sale Adverts dataset provided by AutoTrader (AT), one of our industry partners. The dataset contains an anonymised collection of adverts with information on vehicles such as brand, type, colour, mileage, as well as the selling price. the main task is to produce a regression model for predicting the selling price given the characteristics of the cars in the historical data given. This type of task is similar to that addressed by AT's machine learning engineers and data scientists in the implementation of the price indicator feature in AT's website.

1. **DATA/DOMAIN UNDERSTANDING AND EXPLORATION**

1.1. Meaning and Type of Features; Analysis of Univariate Distributions

The features in the data are categorical and numerical data. Categorical data are data that can be converted into groups. Numerical data are data in number forms.

1. Mileage: provides information on the number of miles the vehicle has been travelled, which can be used to gauge the vehicle's wear and tear.
2. Reg code: This feature provides the registration code, which can be used to locate and identify a vehicle. It contains random alphabets.
3. Standard colour: is a property that describes the colour of the car. The column has a categorical data type.
4. Standard Make: is a property that gives information about the manufacturer of the car. It can be used to determine a vehicle's pricing or level of popularity.
5. Standard Model is a feature that provides information about a car's model, enables it to be distinguished from other models, and can be used to forecast a car's price or level of popularity.
6. Vehicle Condition: This describes the state of the vehicle whether it a new car or a used car. It is used to determine the likelihood of selling the car.
7. Year of registration: is a feature that provides the year the vehicle was first registered.
8. Price is a feature that informs about the car's selling price. It is the target variable and used to estimate the car's price based on other features.
9. Body type: is a property that describes the type of body an automobile has. It can be used to determine a car's size and shape and to determine its pricing or level of popularity.
10. Crossover car and van: This feature helps to identify particular car kinds by indicating if it a crossover car or crossover van.
11. Fuel type: is a characteristic that describes the fuel type used in the vehicle. It can be used to calculate the vehicle's fuel economy and environmental impact.

```
data_distribution(car['mileage']/1000)
```

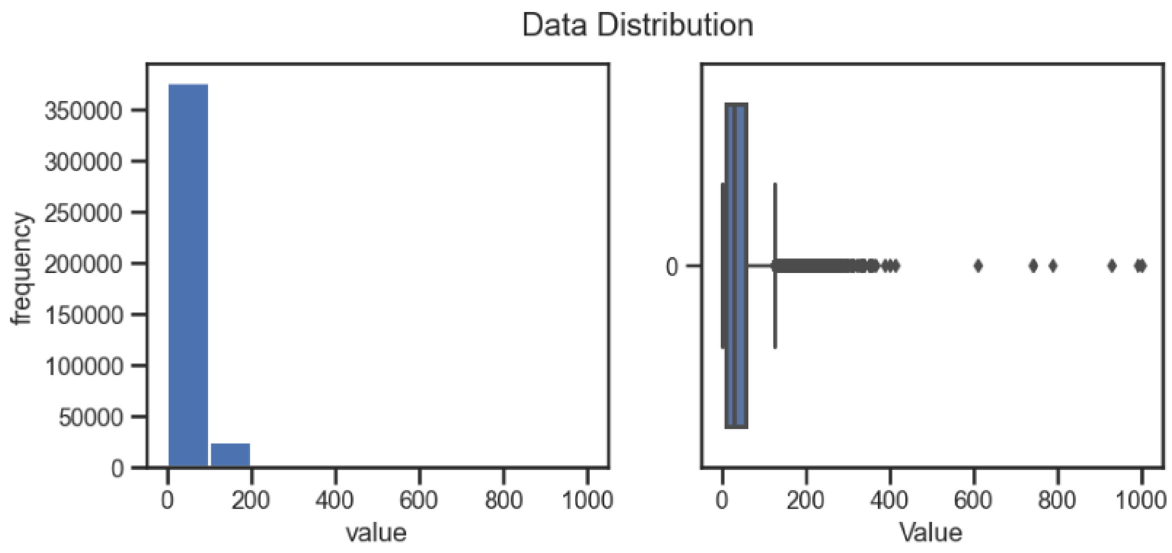


Figure 1: data distribution for the Mileage column showing the frequency on the left and boxplot on the right.

This gives a basic visualization of the mileage column and the boxplot gives us an insight that there are outliers in the columns that needs to be filtered out.

```
data_distribution(car['price']/10000)
```

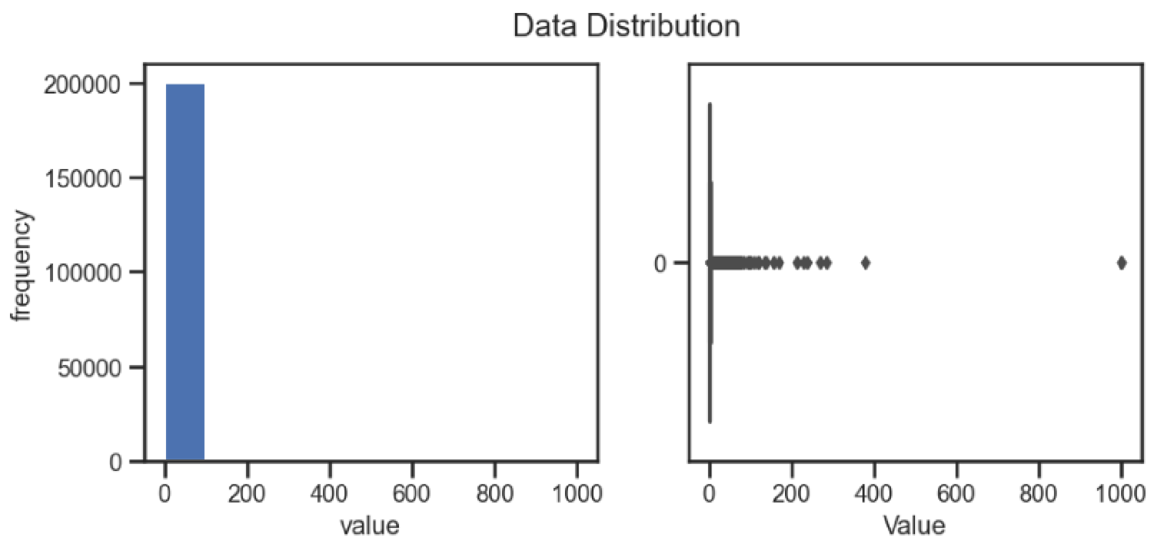


Figure 2: data distribution for the Price column showing the frequency on the left and boxplot on the right. Price

This gives a basic visualization of the price column and the boxplot gives us an insight that there are outliers in the columns that needs to be filtered out with a skewer far from the data.

1.2. Analysis of Predictive Power of Features

```
correlate = car.corr()
sns.heatmap(correlate, xticklabels=correlate.columns, yticklabels=correlate.columns, annot=True)
```

```
C:\Users\22539744\AppData\Local\Temp\ipykernel_18312\827384862.py:2: FutureWarning: The default value of numeric_only
ame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value
c_only to silence this warning.
  correlate = car.corr()
```

```
<AxesSubplot: >
```

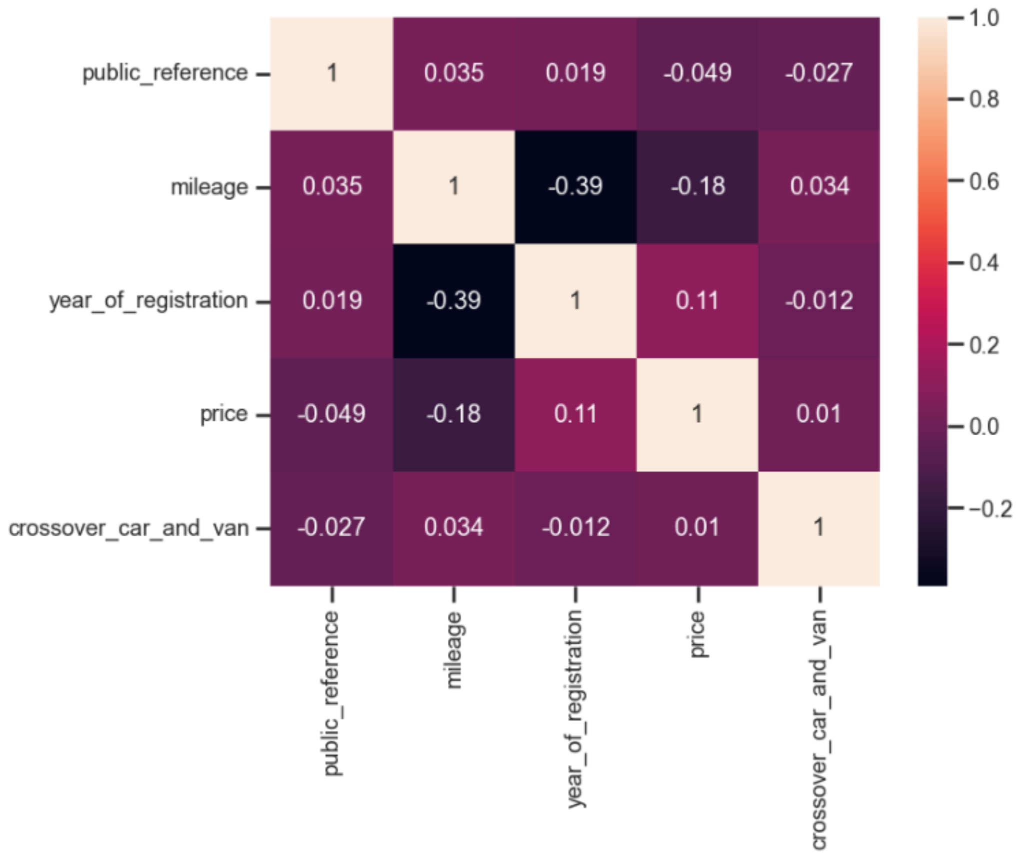
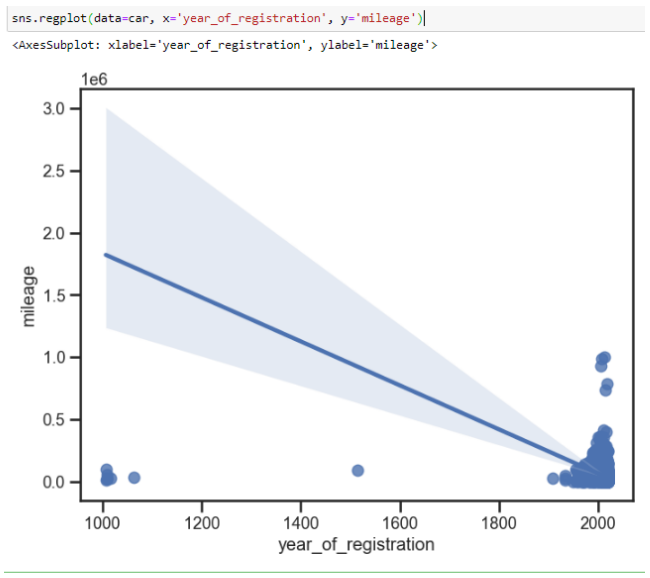


Figure 3: showing a correlation analysis to identify predictive power of features in the dataset.

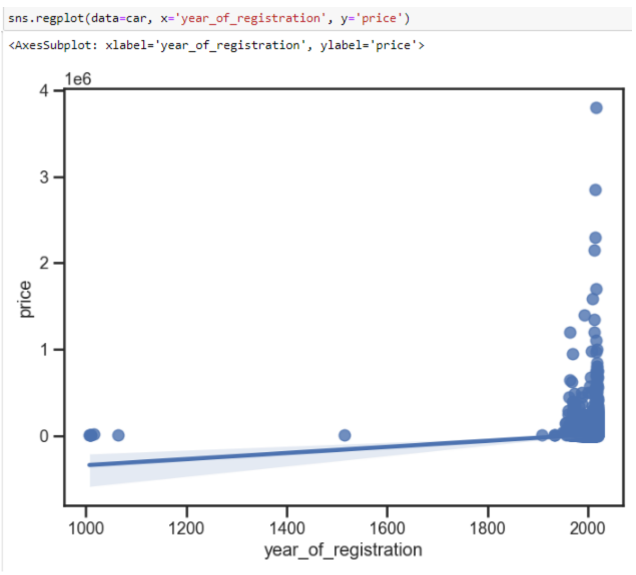
From the plot, it can be inferred that Mileage and year of registration has the highest negative correlation of -0.39 Year of registration has a small positive correlation with the price. Mileage and year of registration has an fair correlation with the price.



```
sns.regplot(data=car, x='year_of_registration', y='mileage')
```

```
<AxesSubplot: xlabel='year_of_registration', ylabel='mileage'>
```

Figure 4: Mileage vs year of registration



```
sns.regplot(data=car, x='year_of_registration', y='price')
```

```
<AxesSubplot: xlabel='year_of_registration', ylabel='price'>
```

Figure 5: Price vs year of registration

Figure 4 shows a negative correlation between mileage and year of registration with little outliers which indicates that the number of miles the car runs decreases as the year of registration increases.

Figure 5 shows a positive correlation between price and year of registration with outliers. Most of the cars have a higher price as the year of registration.

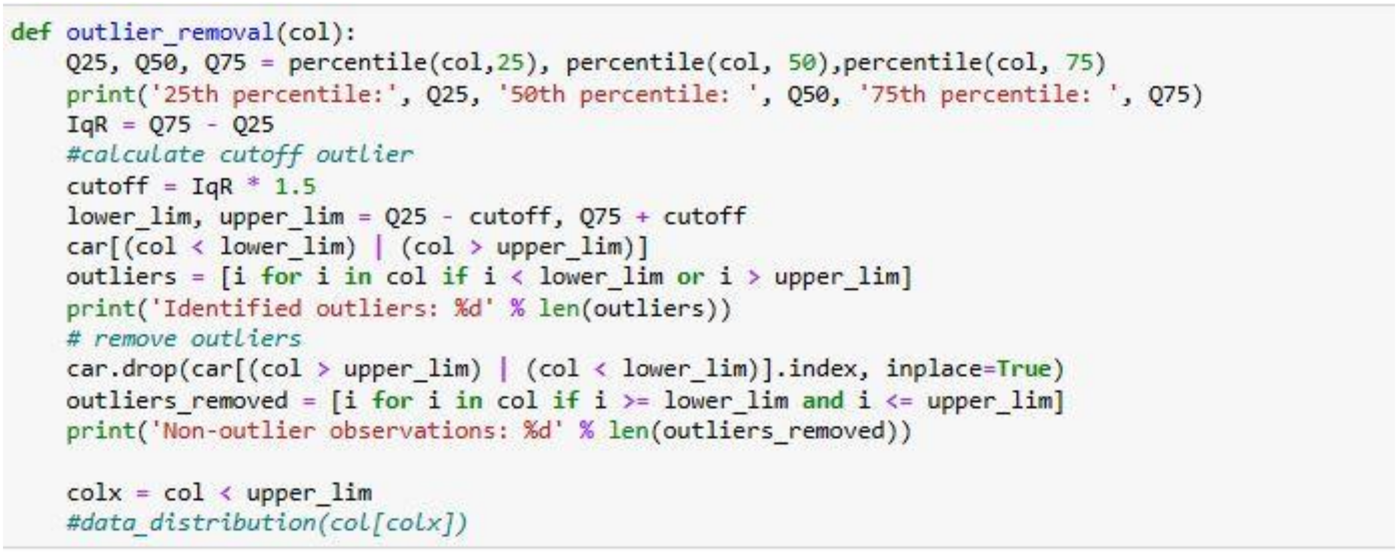
1.3. Data Processing for Data Exploration and Visualisation

The following steps were taken in processing and exploring the data.

- Deleted the rows with missing values because they ae not up to 10% of the data.
- Change categorical columns to numerical columns using label encoder, get dummies and mapping for columns which just two types of categorical data.
- Removed the commas, space and miles input in all the values of the mileage columns.

2. DATA PROCESSING FOR MACHINE LEARNING

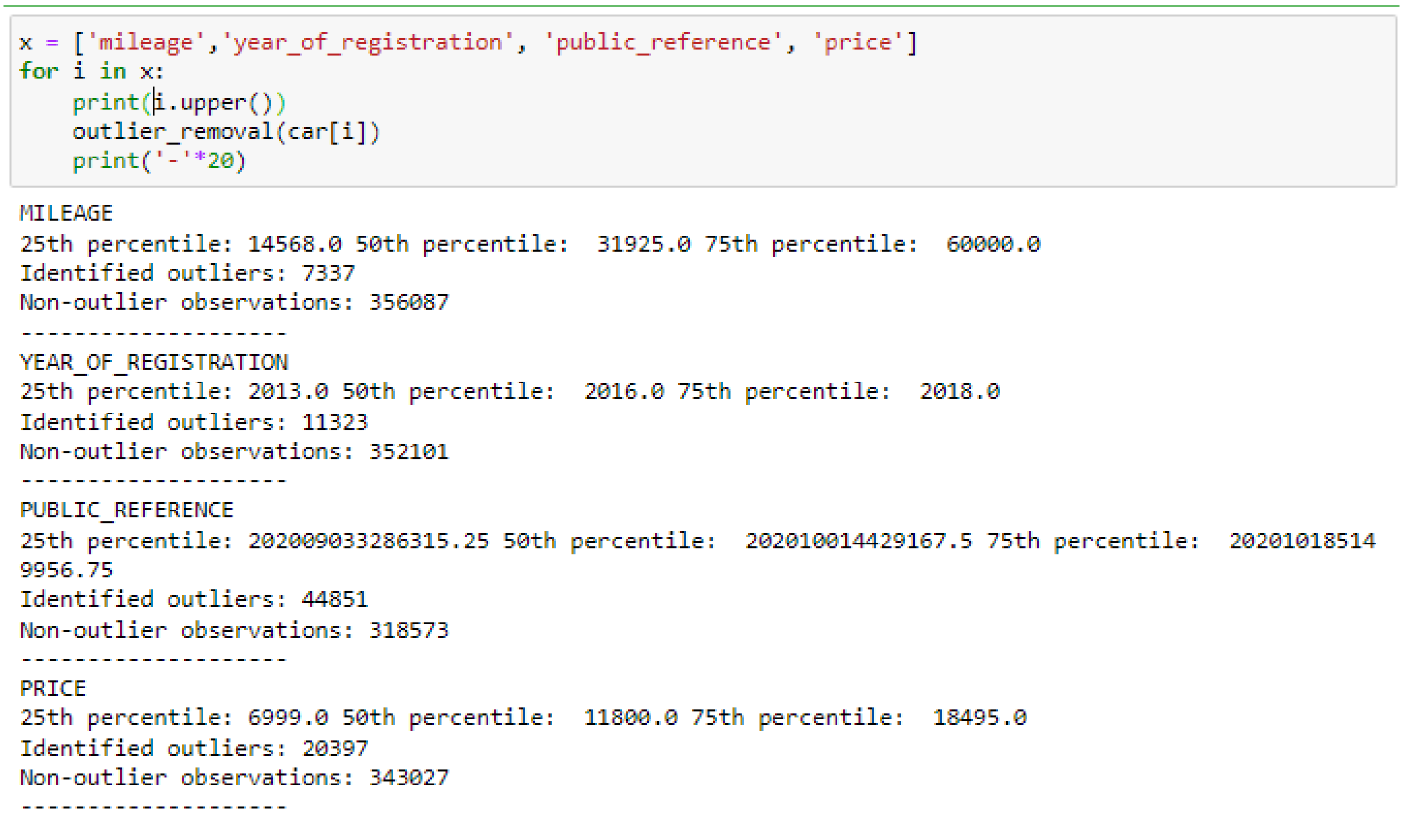
2.1. **Dealing with Missing Values, Outliers, and Noise**



```
def outlier_removal(col):
    Q25, Q50, Q75 = percentile(col,25), percentile(col, 50),percentile(col, 75)
    print('25th percentile:', Q25, '50th percentile: ', Q50, '75th percentile: ', Q75)
    IqR = Q75 - Q25
    #calculate cutoff outlier
    cutoff = IqR * 1.5
    lower_lim, upper_lim = Q25 - cutoff, Q75 + cutoff
    car[(col < lower_lim) | (col > upper_lim)]
    outliers = [i for i in col if i < lower_lim or i > upper_lim]
    print('Identified outliers: %d' % len(outliers))
    # remove outliers
    car.drop(car[(col > upper_lim) | (col < lower_lim)].index, inplace=True)
    outliers_removed = [i for i in col if i >= lower_lim and i <= upper_lim]
    print('Non-outlier observations: %d' % len(outliers_removed))

    colx = col < upper_lim
    #data_distribution(col[colx])
```

Figure 6: Outlier Removal of the datasets

From the visualizations in figures 1 and 2, it showed that there were outliers in the columns so it needed to be removed to avoid a low metrics score and make the algorithm efficient. Removing outliers was done using interquartile range which was first used to identify outliers by specifying a limit with a default k value of 1.5 and the outliers were dropped.



```
x = ['mileage','year_of_registration', 'public_reference', 'price']
for i in x:
    print(i.upper())
    outlier_removal(car[i])
    print('-'*20)
```

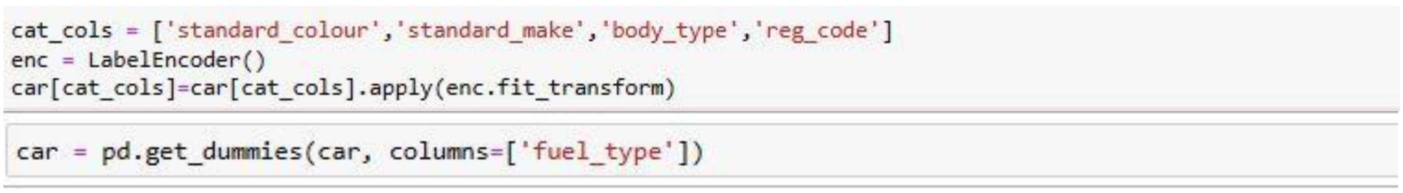
```
MILEAGE
25th percentile: 14568.0 50th percentile:  31925.0 75th percentile:  60000.0
Identified outliers: 7337
Non-outlier observations: 356087
--------------------
YEAR_OF_REGISTRATION
25th percentile: 2013.0 50th percentile:  2016.0 75th percentile:  2018.0
Identified outliers: 11323
Non-outlier observations: 352101
--------------------
PUBLIC_REFERENCE
25th percentile: 202009033286315.25 50th percentile:  202010014429167.5 75th percentile:  20201018514
9956.75
Identified outliers: 44851
Non-outlier observations: 318573
--------------------
PRICE
25th percentile: 6999.0 50th percentile:  11800.0 75th percentile:  18495.0
Identified outliers: 20397
Non-outlier observations: 343027
--------------------
```

Figure 7: Code showing outliers removed in the numerical columns

The list x stored the names of the columns and a for loop was ran to interate through all the columns and the function ‘outlier_removal’ was passed to each of them.

2.2. **Feature Engineering, Data Transformations, Feature Selection**

In this section, All columns with missing values were removed as that is the best way for me to reduce low metrics. We still have to deal with categorical variables in the data from columns: reg code, standard colour, standard make, standard model, fuel type, body type, vehicle condition and crossover car and van.

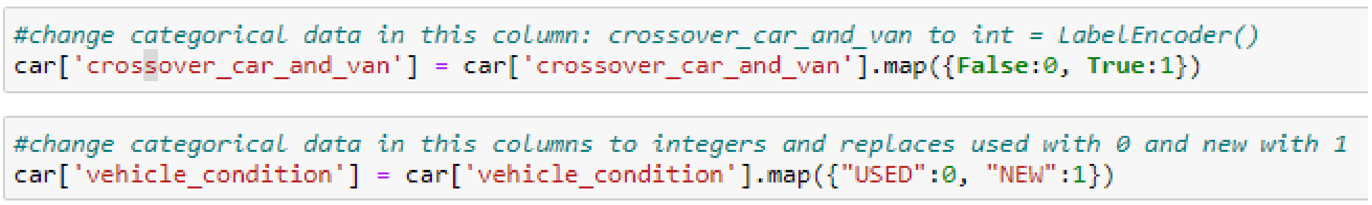


```
cat_cols = ['standard_colour','standard_make','body_type','reg_code']
enc = LabelEncoder()
car[cat_cols]=car[cat_cols].apply(enc.fit_transform)
```

```
car = pd.get_dummies(car, columns=['fuel_type'])
```

Figure 8: Data transformation

Label encoder was used to convert some categorical datas to numerical data and get dummies was used to transform the fuel type column into multiple columns.



```
#change categorical data in this column: crossover_car_and_van to int = LabelEncoder()
car['crossover_car_and_van'] = car['crossover_car_and_van'].map({False:0, True:1})
```

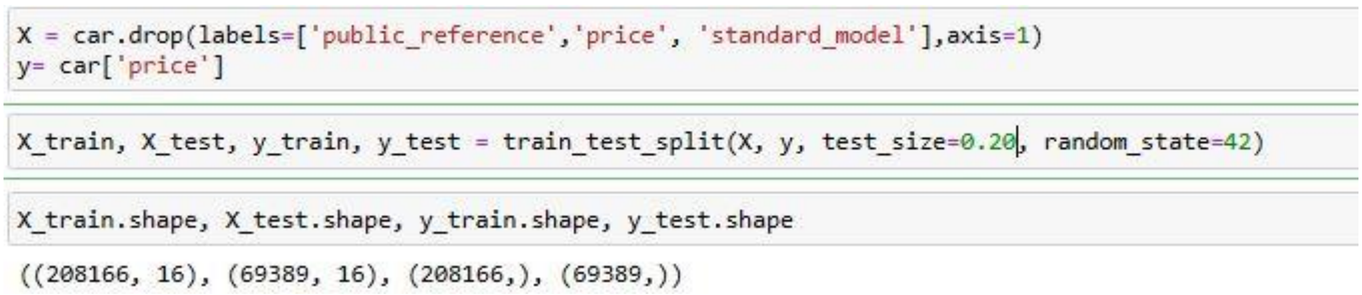
```
#change categorical data in this columns to integers and replaces used with 0 and new with 1
car['vehicle_condition'] = car['vehicle_condition'].map({"USED":0, "NEW":1})
```

Figure 9 : Feature Engineering

Categorical features such as vehicle conditions had just two types of categorical variable which is ‘used’ and ‘new’ same with crossover car and van that has a Boolean (True or False). The both columns were transformed into numerical features(0 and 1) since they had just two types of categorical data as seen in figure 9.

3. MODEL BUILDING

3.1. **Algorithm Selection, Model Instantiation and Configuration**



```
X = car.drop(labels=['public_reference','price', 'standard_model'],axis=1)
y= car['price']
```

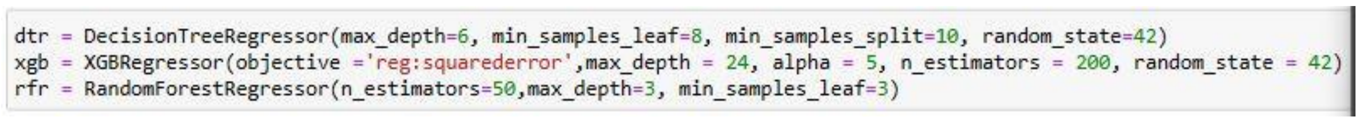
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((208166, 16), (69389, 16), (208166,), (69389,))
```

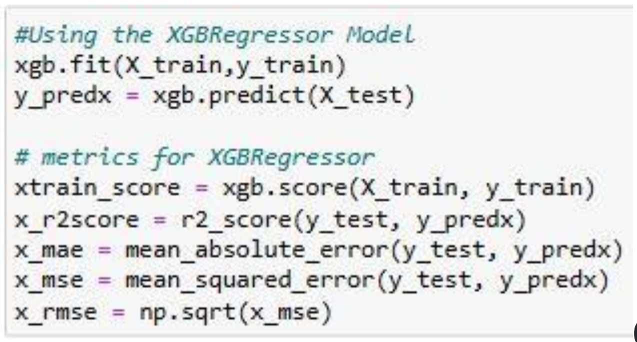
Figure 10: Data Configuration

The following columns were dropped because they are irrelevant ‘price’, ‘public_reference’, and ‘standard_model’ as shown in figure 10. The data were divided into train and test. 80% of the data was given to training and 20% was given to testing.



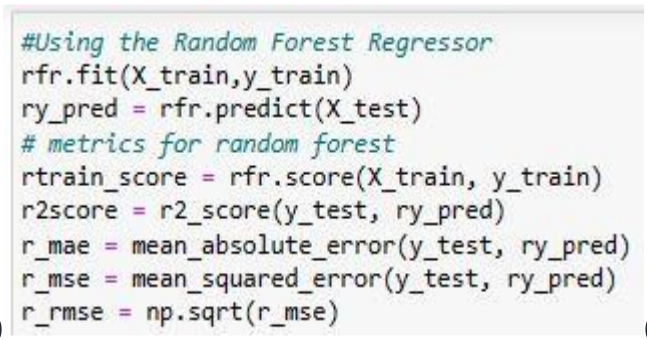
```
dtr = DecisionTreeRegressor(max_depth=6, min_samples_leaf=8, min_samples_split=10, random_state=42)
xgb = XGBRegressor(objective ='reg:squarederror',max_depth = 24, alpha = 5, n_estimators = 200, random_state = 42)
rfr = RandomForestRegressor(n_estimators=50,max_depth=3, min_samples_leaf=3)
```

Figure 11: Algorithm selection and model instantiation



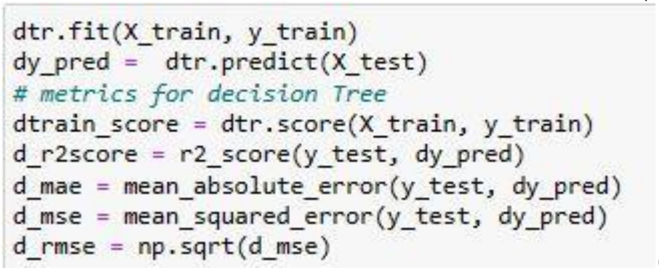
```
#Using the XGBRegressor Model
xgb.fit(X_train,y_train)
y_predx = xgb.predict(X_test)

# metrics for XGBRegressor
xtrain_score = xgb.score(X_train, y_train)
x_r2score = r2_score(y_test, y_predx)
x_mae = mean_absolute_error(y_test, y_predx)
x_mse = mean_squared_error(y_test, y_predx)
x_rmse = np.sqrt(x_mse)
```

(a)



```
#Using the Random Forest Regressor
rfr.fit(X_train,y_train)
ry_pred = rfr.predict(X_test)
# metrics for random forest
rtrain_score = rfr.score(X_train, y_train)
r2score = r2_score(y_test, ry_pred)
r_mae = mean_absolute_error(y_test, ry_pred)
r_mse = mean_squared_error(y_test, ry_pred)
r_rmse = np.sqrt(r_mse)
```

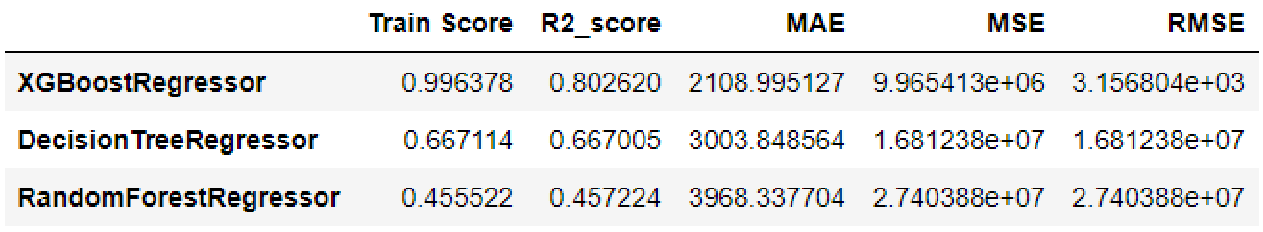
(c)



```
dtr.fit(X_train, y_train)
dy_pred =  dtr.predict(X_test)
# metrics for decision Tree
dtrain_score = dtr.score(X_train, y_train)
d_r2score = r2_score(y_test, dy_pred)
d_mae = mean_absolute_error(y_test, dy_pred)
d_mse = mean_squared_error(y_test, dy_pred)
d_rmse = np.sqrt(d_mse)
```

(c)

Figure 12: (a) XGB Regressor , (b) Random Forest Regressor and (c) Decision Tree Regressor, models were all built.

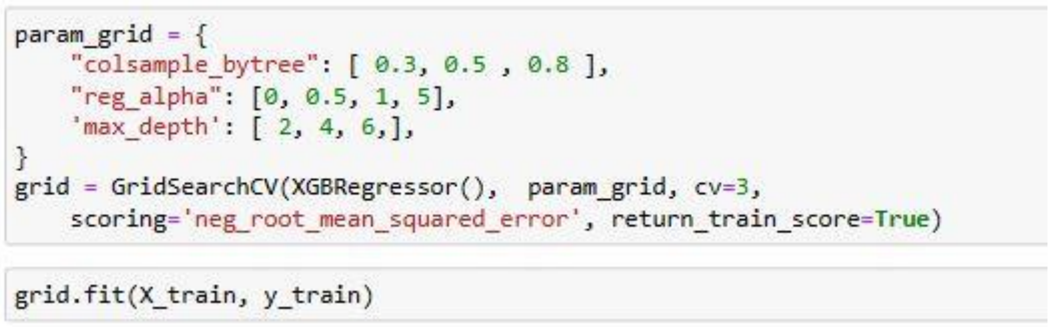


	Train Score	R2_score	MAE	MSE	RMSE
XGBoostRegressor	0.996378	0.802620	2108.995127	9.965413e+06	3.156804e+03
DecisionTreeRegressor	0.667114	0.667005	3003.848564	1.681238e+07	1.681238e+07
RandomForestRegressor	0.455522	0.457224	3968.337704	2.740388e+07	2.740388e+07

Figure 11: Metrics evaluation of the algorithms

The figure 11 shows the metric evaluation and from the table we can see that the XGBoost regressor has the best performance on the data followed by the Decision tree regressor. Random forest regressor performed poorly on the data.

3.2. **Grid Search, and Model Ranking and Selection**



```
param_grid = {
    "colsample_bytree": [ 0.3, 0.5 , 0.8 ],
    "reg_alpha": [0, 0.5, 1, 5],
    'max_depth': [ 2, 4, 6,],
}
grid = GridSearchCV(XGBRegressor(),  param_grid, cv=3,
    scoring='neg_root_mean_squared_error', return_train_score=True)
```

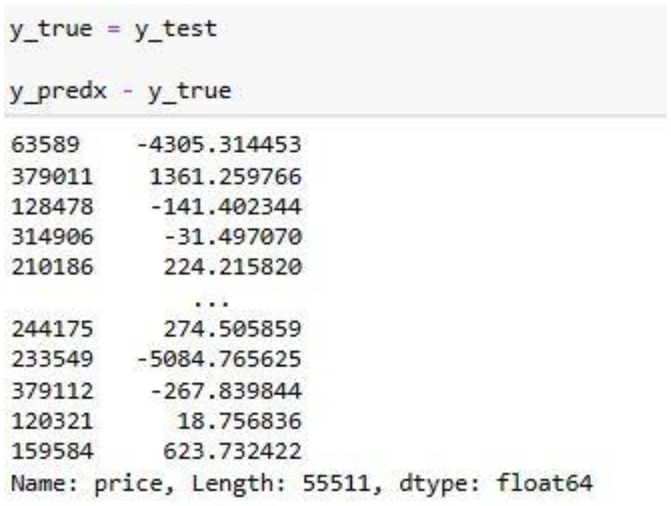
```
grid.fit(X_train, y_train)
```

Figure 12: Grid search

The XGBoostRegressor model has the best performance and it was hypertuned using GridSearchCV to improve the performance.

4. **MODEL EVALUATION AND ANALYSIS**

4.1**. Coarse-Grained Evaluation/Analysis (1-2) (e.g., with model scores)**

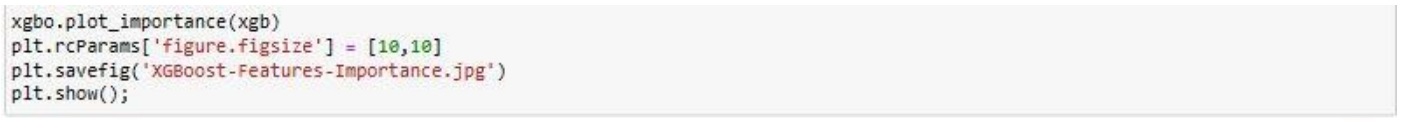


```
y_true = y_test

y_predx - y_true
```

```
63589    -4305.314453
379011    1361.259766
128478    -141.402344
314906     -31.497070
210186     224.215820
              ...
244175     274.505859
233549   -5084.765625
379112    -267.839844
120321      18.756836
159584     623.732422
Name: price, Length: 55511, dtype: float64
```

Figure 13 : analysis

4.2. **Feature Importance**



```
xgbo.plot_importance(xgb)
plt.rcParams['figure.figsize'] = [10,10]
plt.savefig('XGBoost-Features-Importance.jpg')
plt.show();
```

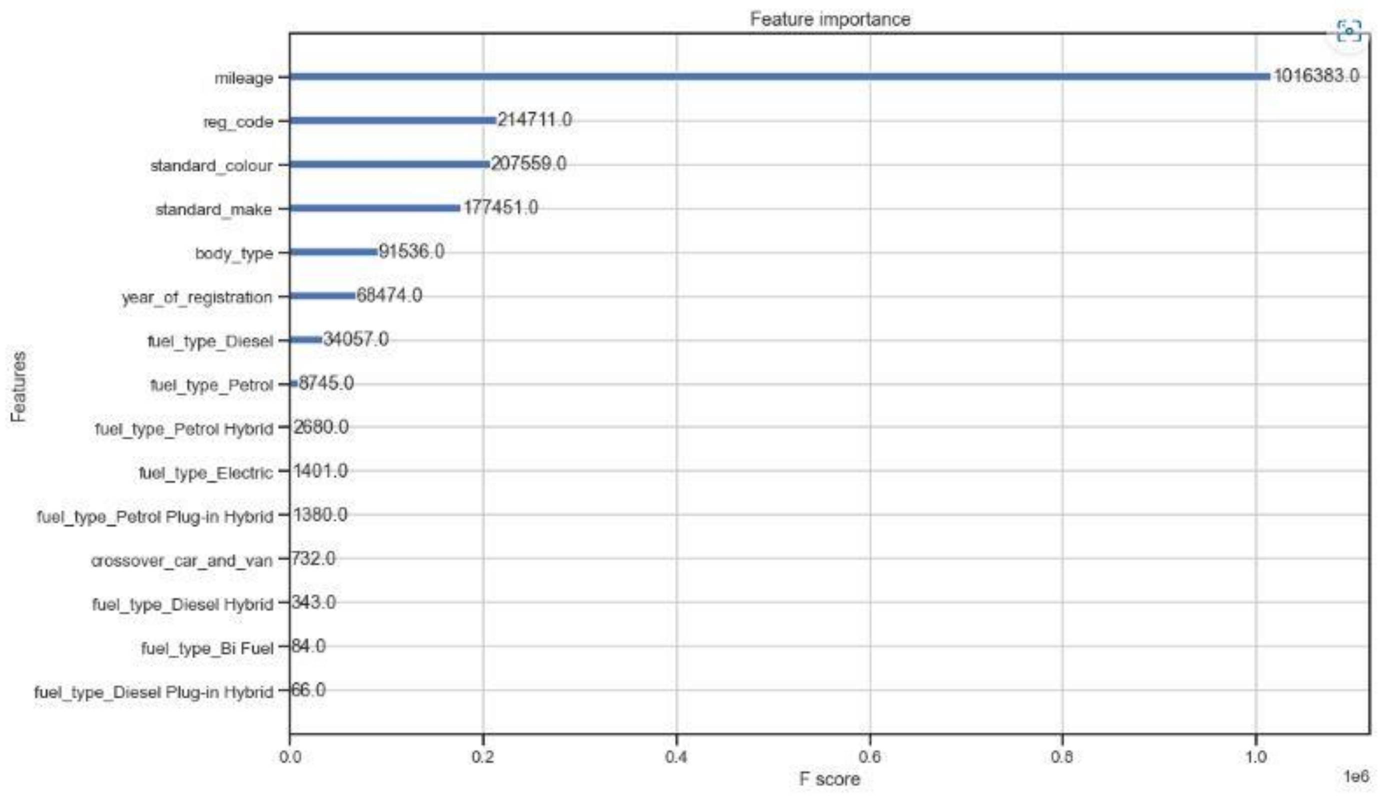


Figure 14: Feature Importance

The most important features are mileage, reg code, standard colour, standard make, body type and year of registration. This features are so important when it comes to knowing the price of a car.

4.3. **Fine-Grained Evaluation (1-2) (e.g., with instance-level errors)**