1. Import the Data set

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import MinMaxScaler # For Normalization

from sklearn.model\_selection import train\_test\_split # For Splitting the data into train data and test data

from sklearn.linear\_model import LinearRegression # For Creation of Linear Regression Model

from sklearn.ensemble import RandomForestRegressor # For Creation of Random Forest Regressor Model

from sklearn.metrics import r2\_score

from catboost import CatBoostRegressor # For Creation of CatBoost Regressor Model

amjdata = pd.read\_csv("D:/Users/OtaiAA0B/Desktop/Bootcamp/Project/AmjadDt.csv")

amjdata.head()

|  | **model** | **year** | **price** | **transmission** | **mileage** | **fuelType** | **tax** | **mpg** | **engineSize** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | A1 | 2017 | 12500 | Manual | 15735 | Petrol | 150 | 55.4 | 1.4 |
| 1 | A6 | 2016 | 16500 | Automatic | 36203 | Diesel | 20 | 64.2 | 2.0 |
| 2 | A1 | 2016 | 11000 | Manual | 29946 | Petrol | 30 | 55.4 | 1.4 |
| 3 | A4 | 2017 | 16800 | Automatic | 25952 | Diesel | 145 | 67.3 | 2.0 |
| 4 | A3 | 2019 | 17300 | Manual | 1998 | Petrol | 145 | 49.6 | 1.0 |

amjdata.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10668 entries, 0 to 10667

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 model 10668 non-null object

1 year 10668 non-null int64

2 price 10668 non-null int64

3 transmission 10668 non-null object

4 mileage 10668 non-null int64

5 fuelType 10668 non-null object

6 tax 10668 non-null int64

7 mpg 10668 non-null float64

8 engineSize 10668 non-null float64

dtypes: float64(2), int64(4), object(3)

memory usage: 750.2+ KB

amjdata\_clean = amjdata.copy(deep = True)

amjdata.shape

(10668, 9)

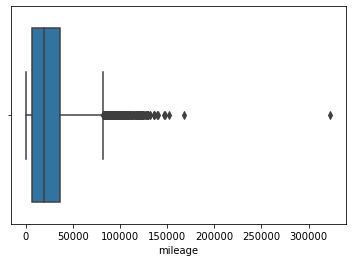
amjdata.isna().sum()

model 0 year 0 price 0 transmission 0 mileage 0 fuelType 0 tax 0 mpg 0 engineSize 0 dtype: int64

As shown above the data set is cleaned with no missing values.

1. Outlier

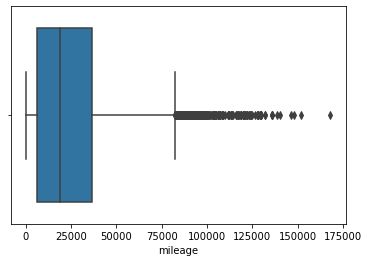
boxa = sns.boxplot(x = 'mileage', data = amjdata\_clean)



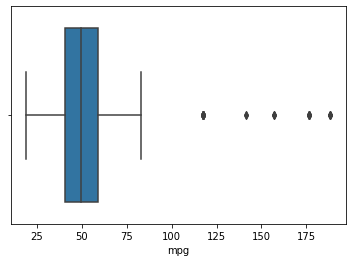
As shown above there is a car above the 300,000 miles which considered an outlier. By remove it the range of data will be from 0 to approxmitly 175000miles.

amjdata\_clean = amjdata\_clean[amjdata\_clean['mileage'] < 175000]

boxa = sns.boxplot(x = 'mileage', data = amjdata\_clean)



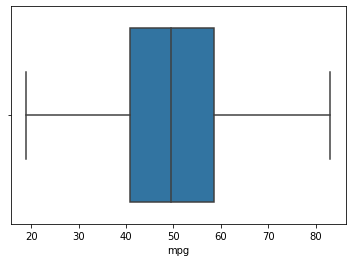
boxa = sns.boxplot(x = 'mpg', data = amjdata)



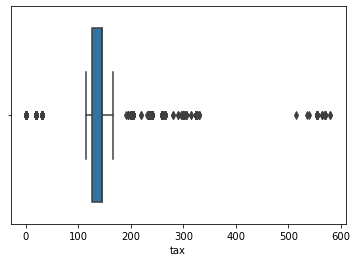
As we can see there are outliers ranging from 120 to 190 mileage. We removed those and make the range from 0 to 110.

amjdata\_clean = amjdata\_clean[amjdata\_clean['mpg'] < 110]

boxa = sns.boxplot(x = 'mpg', data = amjdata\_clean)

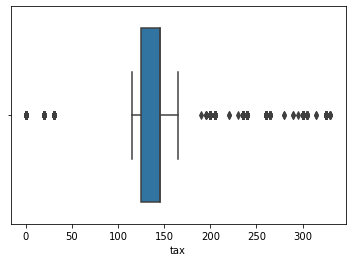


boxa = sns.boxplot(x = 'tax', data = amjdata\_clean)



It is obvious there is outliers in the range from 500 to 600. So, we remove the outliers.

boxa = sns.boxplot(x = 'tax', data = amjdata\_clean)



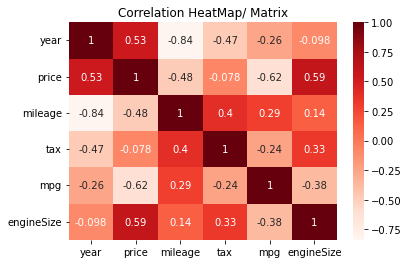
1. Exploratory Data Analysis:

After we removed all the outlier, the data is ready for Exploratory Data Analysis.

sns.heatmap(amjdata\_clean.corr(), cmap ="Reds", annot = True)

plt.title("Correlation HeatMap/ Matrix")

plt.show()



From the Correlation Matrix we get the following information:

* There is a negative correlation between mileage and price. Which means that the car that has more mileage the price is lesser because the car is used more.
* There is a negative correlation between price and mpg. Which means the less mpg the higher price.
* There is positive correlation between engine size and price. It means that having a higher engine size will cost more.
* There is a little positive correlation between price and tax.

fig, axes = plt.subplots(figsize = (12,10), nrows = 2, ncols = 3)

sns.histplot(amjdata\_clean["year"], ax = axes[0,0])

sns.histplot(amjdata\_clean["mileage"], ax = axes[0,1])

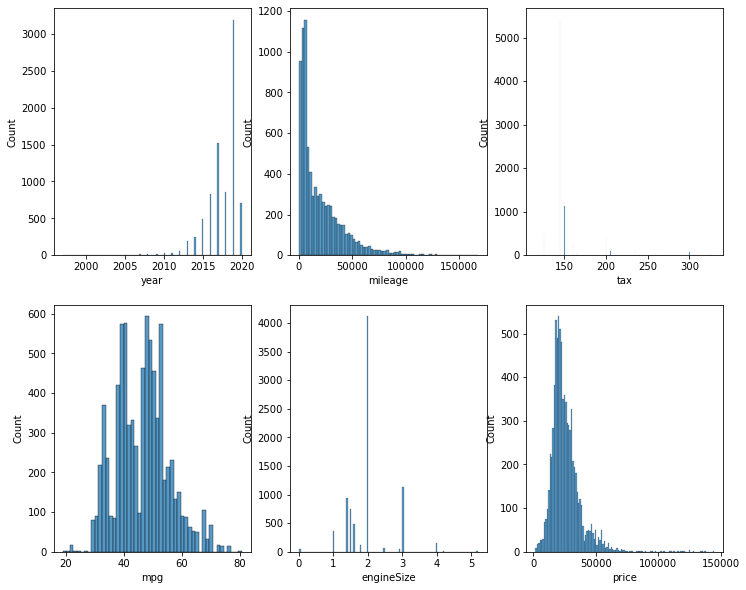
sns.histplot(amjdata\_clean["tax"], ax = axes[0,2])

sns.histplot(amjdata\_clean["mpg"], ax = axes[1,0])

sns.histplot(amjdata\_clean["engineSize"], ax = axes[1,1])

sns.histplot(amjdata\_clean["price"], ax = axes[1,2])

plt.show()

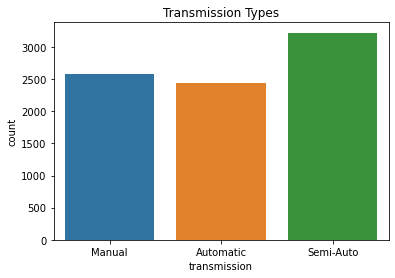


The year column is right-skewed which means that most of the cars are between 2015 to 2020. The mileage is left-skewed which show that most of the cars are driven for more than 5000 miles. The engineSize shows that the most engine size is between 1.5 to 2.

sns.countplot(x = "transmission", data = amjdata\_clean)

plt.title("Transmission Types")

plt.show()



As we can see above, there are 2500+ cars which are Manual and less than 2500 cars are Automatic and 3000+ are Semi-Auto transmission.

print(amjdata\_clean["model"].value\_counts())

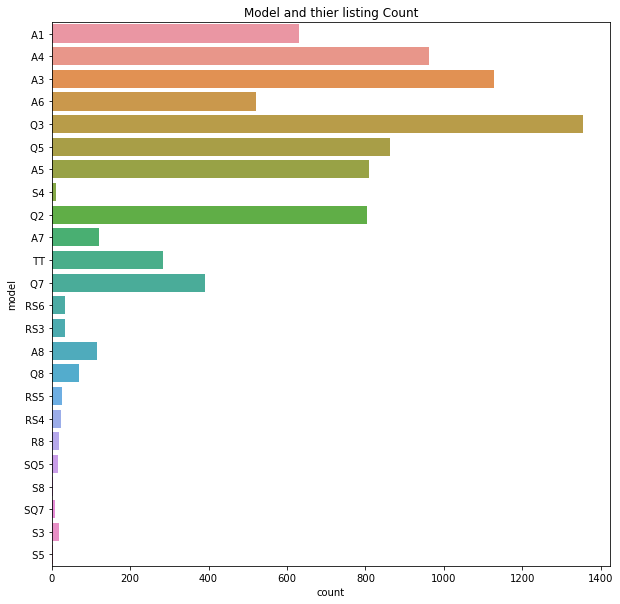
plt.figure(figsize = (10,10))

sns.countplot(y = amjdata\_clean["model"])

plt.title("Model and thier listing Count")

plt.show()

A7 120 A8 116 Q8 69 RS6 34 RS3 33 RS5 26 RS4 23 R8 18 S3 18 SQ5 16 S4 11 SQ7 8 S8 3 S5 3 Name: model, dtype: int64



It is clear that A3's is more than the other models.

plt.subplots(figsize = (18,18))

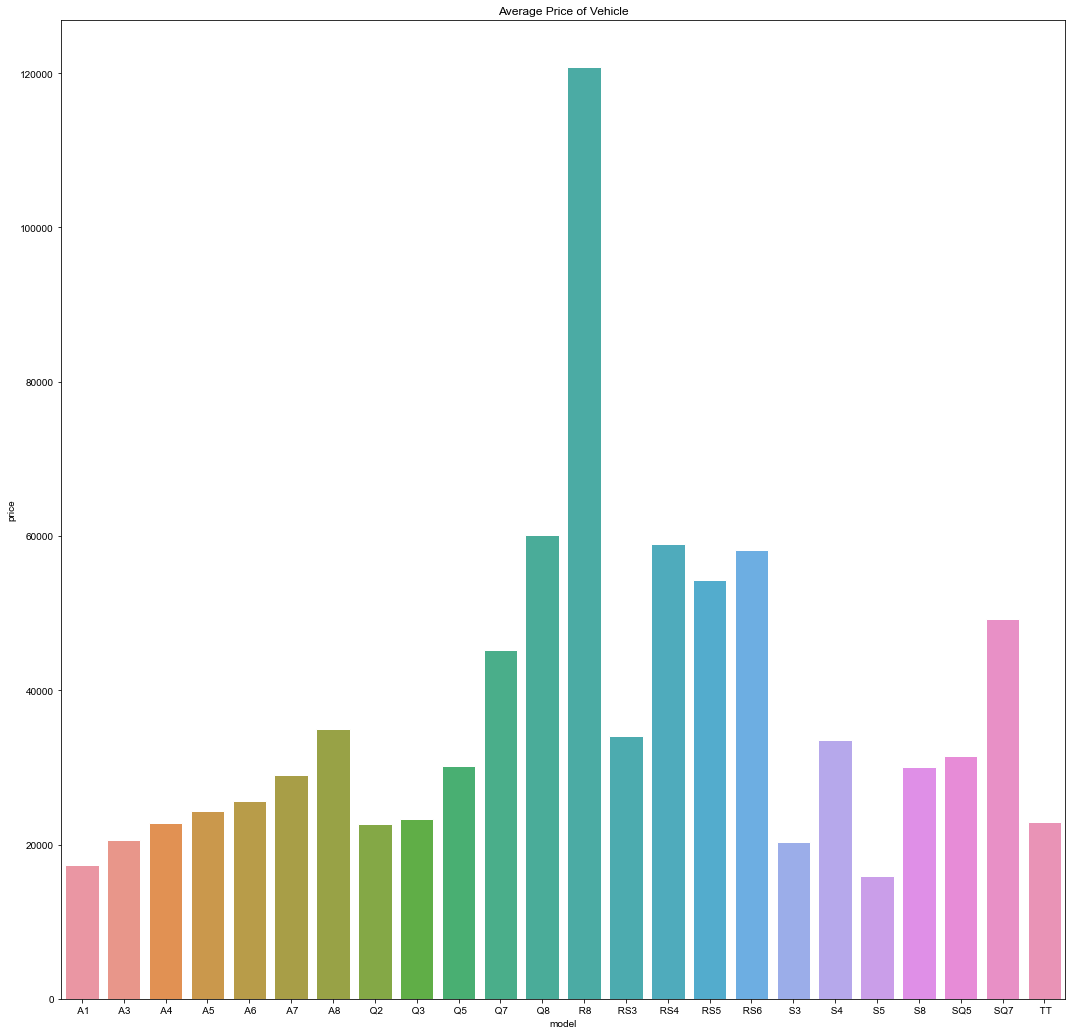
priceByModel = amjdata\_clean.groupby("model")['price'].mean().reset\_index()

plt.title("Average Price of Vehicle")

sns.set()

sns.barplot(x = 'model', y = 'price', data = priceByModel)

plt.show()



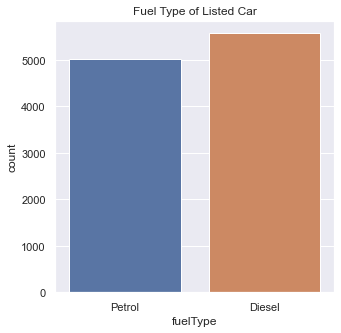
The previous chart shows that the price of R8 is higher than the others models.

plt.figure(figsize = (5,5))

sns.countplot(x = "fuelType", data = amjdata\_clean)

plt.title("Fuel Type of Listed Car")

plt.show()

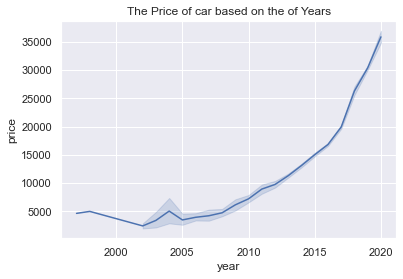


It shows that the cars with diesel fuel type more than the cars with Petrol fuel type.

sns.lineplot(x = "year", y = "price", data = amjdata\_clean)

plt.title("The Price of car based on the of Years")

plt.show()

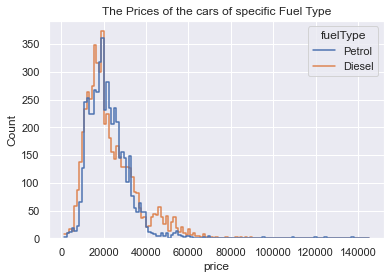


The lineplot shows that the new cars having higher prices due to less destination the car has travelled. It shows that there are some deviations or differences in prices of the car but the price failed to keep rising.

sns.histplot(data = amjdata\_clean, x = 'price', hue = 'fuelType', fill = False, element = 'step')

plt.title("The Prices of the cars of specific Fuel Type")

plt.show()



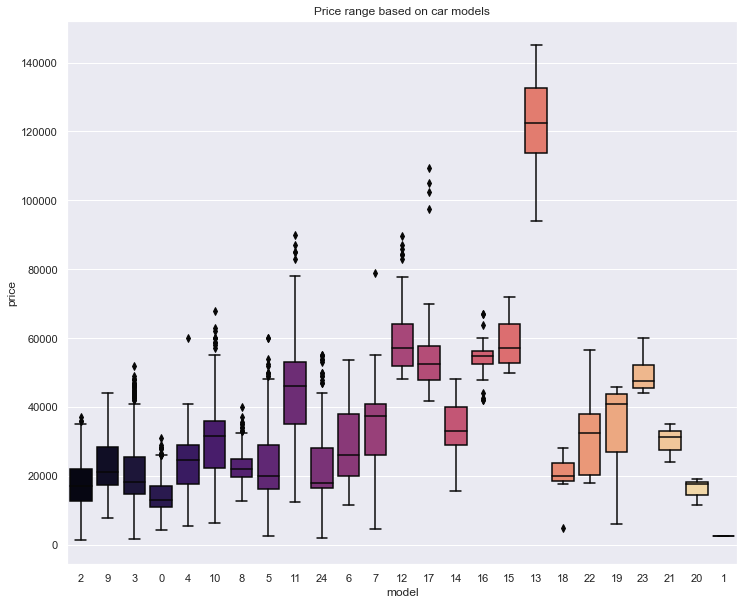
As obvious there is no different in the price of the car which is used diesel fuel type or the car which used petrol fuel type.

plt.figure(figsize = (12,10))

sns.boxplot(data = amjdata\_clean, x = 'model', y = 'price', order = amjdata\_clean['model'].value\_counts().index, palette = 'magma')

plt.title("Price range based on car models")

plt.show()



1. Used the Label Encoding Technique:

encoder = LabelEncoder()

amjdata\_clean['model'] = encoder.fit\_transform(amjdata\_clean['model'])

model\_mapping = {index : label for index, label in enumerate(encoder.classes\_)}

model\_mapping

{0: ' A1', 1: ' A2', 2: ' A3', 3: ' A4', 4: ' A5', 5: ' A6', 6: ' A7', 7: ' A8', 8: ' Q2', 9: ' Q3', 10: ' Q5', 11: ' Q7', 12: ' Q8', 13: ' R8', 14: ' RS3', 15: ' RS4', 16: ' RS5', 17: ' RS6', 18: ' S3', 19: ' S4', 20: ' S5', 21: ' S8', 22: ' SQ5', 23: ' SQ7', 24: ' TT'}

amjdata\_clean['fuelType'] = encoder.fit\_transform(amjdata\_clean['fuelType'])

fuelType\_mapping = {index : label for index, label in enumerate(encoder.classes\_)}

fuelType\_mapping

{0: 'Diesel', 1: 'Petrol'}

amjdata\_clean['transmission'] = encoder.fit\_transform(amjdata\_clean['transmission'])

transmission\_mapping = {index : label for index, label in enumerate(encoder.classes\_)}

transmission\_mapping

{0: 'Automatic', 1: 'Manual', 2: 'Semi-Auto'}

amjdata\_clean.head()

|  | **model** | **year** | **price** | **transmission** | **mileage** | **fuelType** | **tax** | **mpg** | **engineSize** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 2017 | 12500 | 1 | 15735 | 1 | 150 | 55.4 | 1.4 |
| 1 | 5 | 2016 | 16500 | 0 | 36203 | 0 | 20 | 64.2 | 2.0 |
| 2 | 0 | 2016 | 11000 | 1 | 29946 | 1 | 30 | 55.4 | 1.4 |
| 3 | 3 | 2017 | 16800 | 0 | 25952 | 0 | 145 | 67.3 | 2.0 |
| 4 | 2 | 2019 | 17300 | 1 | 1998 | 1 | 145 | 49.6 | 1.0 |

1. Train and Test Data:

x = amjdata\_clean.drop('price', axis = 1)

y = amjdata\_clean['price']

scaler = MinMaxScaler(copy = True, feature\_range = (0,1))

X = scaler.fit\_transform(x)

X[:10]

array([[0. , 0.86956522, 0.5 , 0.09364584, 1. , 0.45454545, 0.56853583, 0.26923077], [0.20833333, 0.82608696, 0. , 0.21546757, 0. , 0.06060606, 0.70560748, 0.38461538], [0. , 0.82608696, 0.5 , 0.17822707, 1. , 0.09090909, 0.56853583, 0.26923077], [0.125 , 0.86956522, 0. , 0.15445553, 0. , 0.43939394, 0.75389408, 0.38461538], [0.08333333, 0.95652174, 0.5 , 0.01188577, 1. , 0.43939394, 0.47819315, 0.19230769], [0. , 0.82608696, 0. , 0.19199957, 1. , 0.09090909, 0.62305296, 0.26923077], [0.20833333, 0.82608696, 0. , 0.45702195, 0. , 0.09090909, 0.66199377, 0.38461538], [0.125 , 0.82608696, 0.5 , 0.44748119, 0. , 0.06060606, 0.80529595, 0.38461538], [0.08333333, 0.7826087 , 0.5 , 0.2744441 , 1. , 0.06060606, 0.64174455, 0.26923077], [0. , 0.82608696, 0.5 , 0.13361823, 1. , 0.09090909, 0.56853583, 0.26923077]])

x\_train, x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size = 0.35, random\_state=0)

Here we split the data into two different parts for model creation: 65% Train and 35% Test.

print("Shape of the x\_train: ", x\_train.shape)

print("Shape of the x\_test: ", x\_test.shape)

print("Shape of y\_train:", y\_train.shape)

print("Shape of the y\_test: ", y\_test.shape)

Shape of the x\_train: (6886, 8) Shape of the x\_test: (3708, 8) Shape of y\_train: (6886,) Shape of the y\_test: (3708,)

1. Models:

* LinearRegression:

LinearRegressionModel = LinearRegression(fit\_intercept = True, normalize = True, copy\_X = True, n\_jobs = -1)

LinearRegressionModel.fit(x\_train, y\_train)

print('Linear Regression Train Score is : ' , LinearRegressionModel.score(x\_train, y\_train))

print('Linear Regression Test Score is : ' , LinearRegressionModel.score(x\_test, y\_test))

print('----------------------------------------------------')

y\_pred = LinearRegressionModel.predict(x\_test)

print('Predicted Value for Linear Regression is : ' , y\_pred[:10])

Linear Regression Train Score is : 0.8247222369421101 Linear Regression Test Score is : 0.802362767689407 ---------------------------------------------------- Predicted Value for Linear Regression is : [45350.5364866 35484.21220177 6641.76169241 15247.46129149 41272.85147646 44191.43873359 35123.80302147 14267.41599704 12960.46970305 19888.50087271]

pricePredicted = pd.DataFrame({'Actual Price': y\_test, 'Predicted Price': y\_pred})

pricePredicted = pricePredicted.reset\_index()

pricePredicted.head(10)

|  | **index** | **Actual Price** | **Predicted Price** |
| --- | --- | --- | --- |
| 0 | 4836 | 51990 | 45350.536487 |
| 1 | 5146 | 29990 | 35484.212202 |
| 2 | 8824 | 10500 | 6641.761692 |
| 3 | 10256 | 17600 | 15247.461291 |
| 4 | 3363 | 39500 | 41272.851476 |
| 5 | 8650 | 45950 | 44191.438734 |
| 6 | 4786 | 37990 | 35123.803021 |
| 7 | 8140 | 16599 | 14267.415997 |
| 8 | 8379 | 9999 | 12960.469703 |
| 9 | 8084 | 19700 | 19888.500873 |

There is a significant difference between the actual price of cars and the predicted price by Linear Regression model which has 80.23% accuracy.

* Random Forest:

RandomForestRegressorModel = RandomForestRegressor(n\_estimators=100,max\_depth=11, random\_state=33)

RandomForestRegressorModel.fit(x\_train, y\_train)

print('Random Forest  Train Score is : ' , RandomForestRegressorModel.score(x\_train, y\_train))

print('Random Forest  Test Score is : ' , RandomForestRegressorModel.score(x\_test, y\_test))

print('Random Forest  No. of features are : ' , RandomForestRegressorModel.n\_features\_)

print('----------------------------------------------------')

y\_pred = RandomForestRegressorModel.predict(x\_test)

print('Predicted Value for Random Forest  is : ' , y\_pred[:10])

Random Forest Train Score is : 0.9787520650217197 Random Forest Test Score is : 0.9556898105633875 Random Forest No. of features are : 8 ---------------------------------------------------- Predicted Value for Random Forest is : [53397.98768334 33352.0846768 10720.49012831 20566.74741075 39725.08112047 45049.42153305 36261.98396967 17302.14177341 11711.98301038 18800.79135818]

pricePredicted = pd.DataFrame({'Actual Price': y\_test, 'Predicted Price': y\_pred})

pricePredicted = pricePredicted.reset\_index()

pricePredicted.head(10)

|  | **index** | **Actual Price** | **Predicted Price** |
| --- | --- | --- | --- |
| 0 | 4836 | 51990 | 53397.987683 |
| 1 | 5146 | 29990 | 33352.084677 |
| 2 | 8824 | 10500 | 10720.490128 |
| 3 | 10256 | 17600 | 20566.747411 |
| 4 | 3363 | 39500 | 39725.081120 |
| 5 | 8650 | 45950 | 45049.421533 |
| 6 | 4786 | 37990 | 36261.983970 |
| 7 | 8140 | 16599 | 17302.141773 |
| 8 | 8379 | 9999 | 11711.983010 |
| 9 | 8084 | 19700 | 18800.791358 |

We can see that there a little difference between the actual price and the predicted price that predicted by Random Forest Model which is 95.56% accurate.

* CatBoost:

catModel = CatBoostRegressor(verbose = 0, random\_state = 35)

catModel.fit(x\_train, y\_train)

y\_pred = catModel.predict(x\_test)

r2 = r2\_score(y\_pred, y\_test)

print(f'CatBoost Regressor Model by Yandex r2 score : {r2:0.5f}')

CatBoost Regressor Model by Yandex r2 score : 0.96138

pricePredicted = pd.DataFrame({'Actual Price': y\_test, 'Predicted Price': y\_pred})

pricePredicted = pricePredicted.reset\_index()

pricePredicted.head(10)

|  | **index** | **Actual Price** | **Predicted Price** |
| --- | --- | --- | --- |
| 0 | 2049 | 14998 | 13352.436977 |
| 1 | 5609 | 21950 | 24429.887177 |
| 2 | 7638 | 28990 | 27923.722830 |
| 3 | 1603 | 25489 | 26536.447255 |
| 4 | 5953 | 30950 | 31780.507314 |
| 5 | 6928 | 30299 | 28639.214744 |
| 6 | 1712 | 14498 | 16067.636326 |
| 7 | 7642 | 32000 | 31334.839723 |
| 8 | 1180 | 19991 | 20292.454572 |
| 9 | 5867 | 29990 | 30992.428627 |

It is clear that there very little difference between the actual price and the predicted price of the cars, which because we have 96% accuracy of CatBoost Regressor Model.