JAHANGIRNAGAR UNIVERSITY

Master's In Applied Statistics & Data Science

Course Title: Introduction to Data Science with Python

Course No.: WM-ASDS04

Assignment, Spring 2023

Submitted To:

DR. FARHANA AKHTER DUTY

Professor, Department of Statistics

Submitted By

Name: Gulsaba Fiha

ID: 20231066 [ASDS10 (Section B)]

Title: Predictive Analysis of Used Car Prices using Linear Regression: A Case Study with the Toyota Dataset

Introduction:

The automobile industry is characterized by its dynamic nature, where the value of vehicles depreciates over time. Predicting used car prices is crucial for both buyers and sellers. This study aims to develop a predictive model using linear regression to estimate used car prices based on features such as mileage, age, fuel type, model and engineSize.

This report presents a detailed analysis of predicting used car prices using a linear regression model. The objective of this study was to understand the relationship between various features of cars and their corresponding prices in the used car market. The dataset used for this analysis is the "Toyota Dataset" sourced from Kaggle. The study involved data preprocessing, exploratory data analysis, feature selection, model training, and evaluation. The findings indicate that a linear regression model can provide reasonable predictions for used car prices based on relevant features.

Finally, the best model will be used to estimate the entire dataset and the results will be interpreted. This will provide insights into the factors that contribute to predict car prices based on their features.

Here are the specific steps involved in the assignment:

- Reading the data into Python.
- Inspecting the data types and distributions.
- Perform EDA and visualization for better understanding the dataset.
- Data preprocessing to clean and make ready the data for analysis, such as handling missing value, scaling, encoding categorical variable etc.
- Split the dataset into test and training data. Build the Machine Learning Model using training data.
- Evaluate performance of the model for test set.

- Perform hyperparameter tuning and obtain the best predictive model.
- Apply performance improvement techniques.
- Get the final model and obtain the predicted target values using the model.

About The Dataset

The Toyota Dataset from Kaggle consists of various attributes of used Toyota cars, including model, year, price, mileage, fuel type, and more. Data preprocessing involved handling missing values, encoding categorical variables, and scaling numerical features.

Variable explanation

- Model: The model name or number of the Toyota car (e.g., Corolla, Camry, Prius,GT86).
- Year: The manufacturing year of the car.
- Price: The selling price of the used car.
- Mileage: The total distance the car has been driven in miles.
- Fuel Type: The type of fuel the car uses (e.g., gasoline, diesel, hybrid, electric).
- Transmission: The type of transmission (automatic, manual).
- Engine Size: The capacity of the car's engine in liters.
- MPG: Miles per gallon, indicating the car's fuel efficiency.

In this data set, we have a total of 6738 dimensions and 7 features, 8 column is the label.

In this data set there is no missing value.

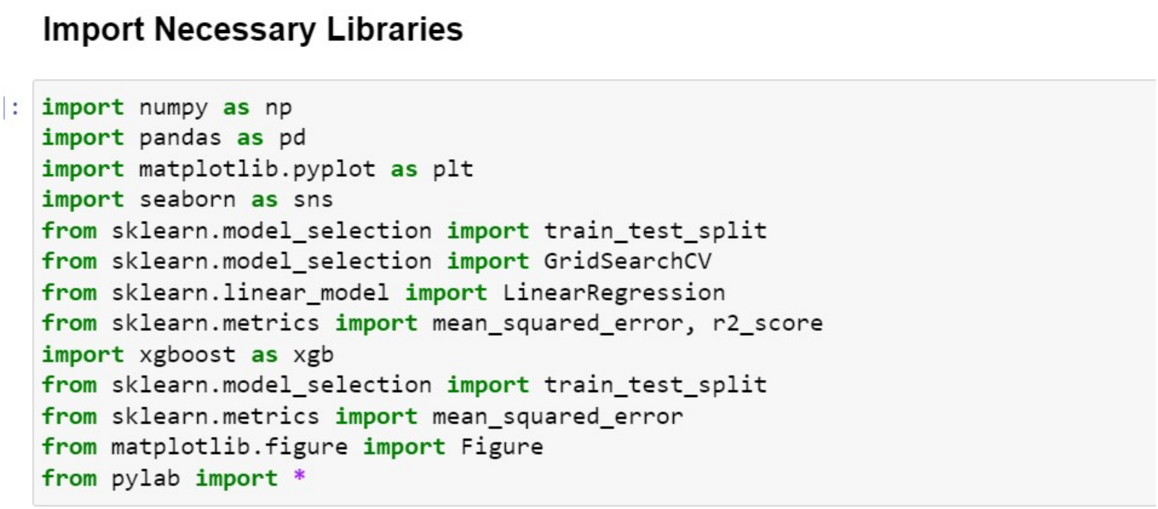
Features

Attribute Name	Role	Type	
model	Feature	Categorical	
year	Feature	Integer	
transmission	Feature	Categorical	
mileage	Feature	Integer	
fuelType	Feature	Categorical	
mpg	Feature	Continuous	
engineSize	Feature	Continuous	
price	Target	Integer	

Preparing Dataset for Assignment

Since we need to import our necessary library for performing the assignment. Then we need to import our dataset.:

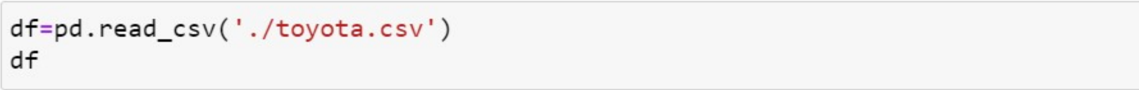
Input:



Import Necessary Libraries

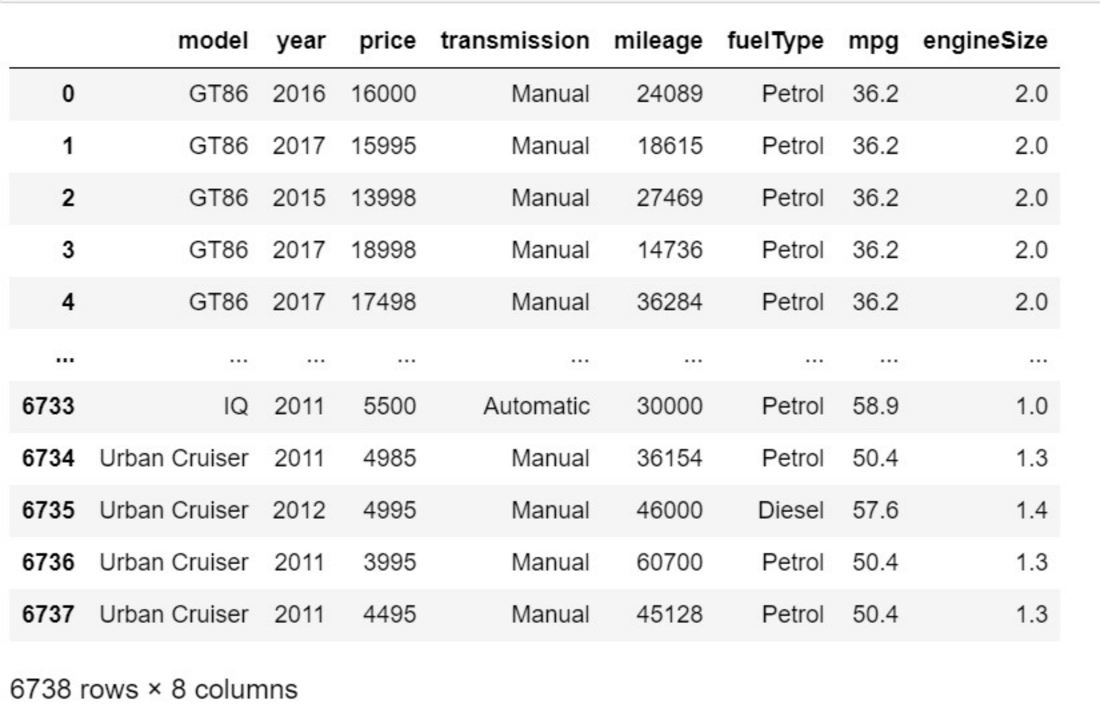
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from matplotlib.figure import Figure
from pylab import *
```

Load The Data



```
df=pd.read_csv('./toyota.csv')
df
```

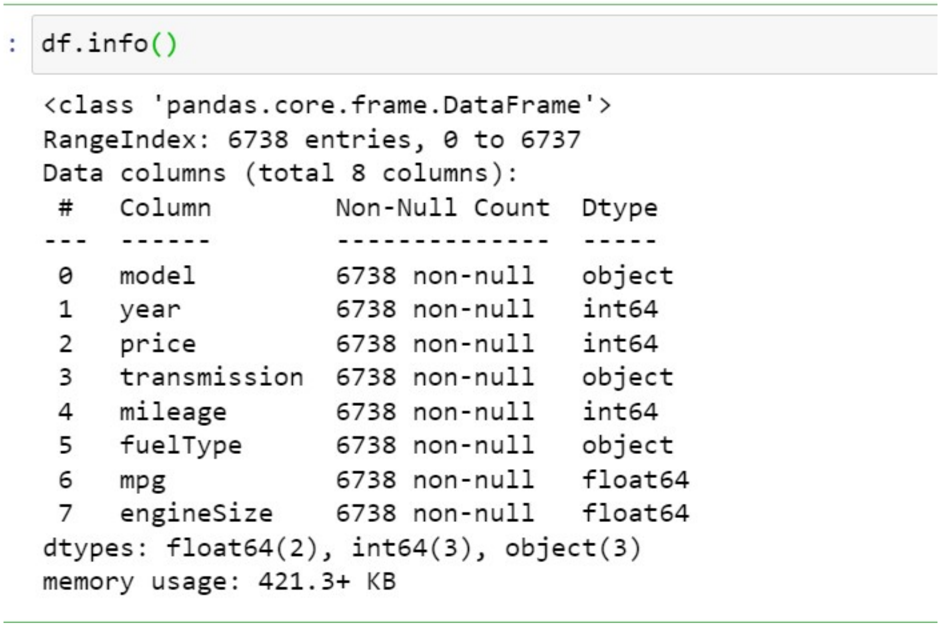
Output With 6738 entries.



	model	year	price	transmission	mileage	fuelType	mpg	engineSize
0	GT86	2016	16000	Manual	24089	Petrol	36.2	2.0
1	GT86	2017	15995	Manual	18615	Petrol	36.2	2.0
2	GT86	2015	13998	Manual	27469	Petrol	36.2	2.0
3	GT86	2017	18998	Manual	14736	Petrol	36.2	2.0
4	GT86	2017	17498	Manual	36284	Petrol	36.2	2.0
...	...	...	...	...	...	...	...	...
6733	IQ	2011	5500	Automatic	30000	Petrol	58.9	1.0
6734	Urban Cruiser	2011	4985	Manual	36154	Petrol	50.4	1.3
6735	Urban Cruiser	2012	4995	Manual	46000	Diesel	57.6	1.4
6736	Urban Cruiser	2011	3995	Manual	60700	Petrol	50.4	1.3
6737	Urban Cruiser	2011	4495	Manual	45128	Petrol	50.4	1.3

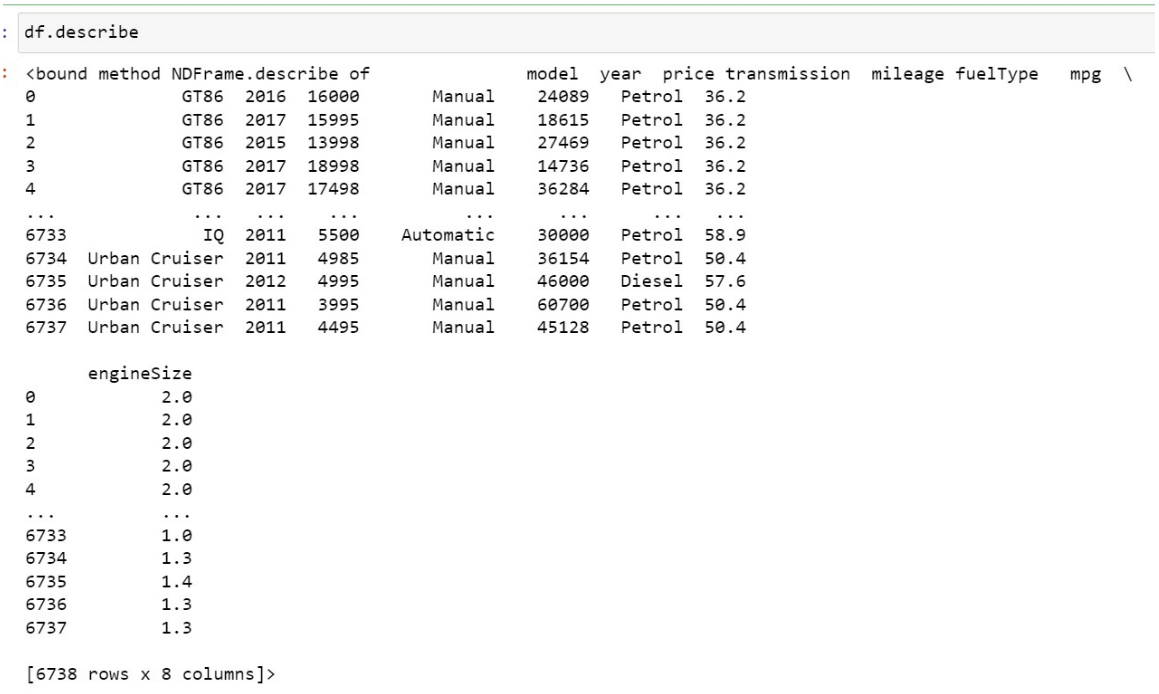
6738 rows × 8 columns

Info About the Data



```
: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6738 entries, 0 to 6737
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   model         6738 non-null   object 
 1   year          6738 non-null   int64  
 2   price         6738 non-null   int64  
 3   transmission  6738 non-null   object 
 4   mileage       6738 non-null   int64  
 5   fuelType      6738 non-null   object 
 6   mpg           6738 non-null   float64
 7   engineSize    6738 non-null   float64
dtypes: float64(2), int64(3), object(3)
memory usage: 421.3+ KB
```



```
: df.describe
```

```
: <bound method NDFrame.describe of              model  year  price transmission  mileage fuelType   mpg  \
  0              GT86  2016  16000       Manual    24089   Petrol  36.2   
  1              GT86  2017  15995       Manual    18615   Petrol  36.2   
  2              GT86  2015  13998       Manual    27469   Petrol  36.2   
  3              GT86  2017  18998       Manual    14736   Petrol  36.2   
  4              GT86  2017  17498       Manual    36284   Petrol  36.2   
  ...             ...   ...    ...          ...      ...      ...   ...   
  6733             IQ  2011   5500    Automatic    30000   Petrol  58.9   
  6734  Urban Cruiser  2011   4985       Manual    36154   Petrol  50.4   
  6735  Urban Cruiser  2012   4995       Manual    46000   Diesel  57.6   
  6736  Urban Cruiser  2011   3995       Manual    60700   Petrol  50.4   
  6737  Urban Cruiser  2011   4495       Manual    45128   Petrol  50.4   

        engineSize  
  0            2.0  
  1            2.0  
  2            2.0  
  3            2.0  
  4            2.0  
  ...          ...  
  6733         1.0  
  6734         1.3  
  6735         1.4  
  6736         1.3  
  6737         1.3  

  [6738 rows x 8 columns]>
```

- **Interpret Data with graph for variables**

Histogram:

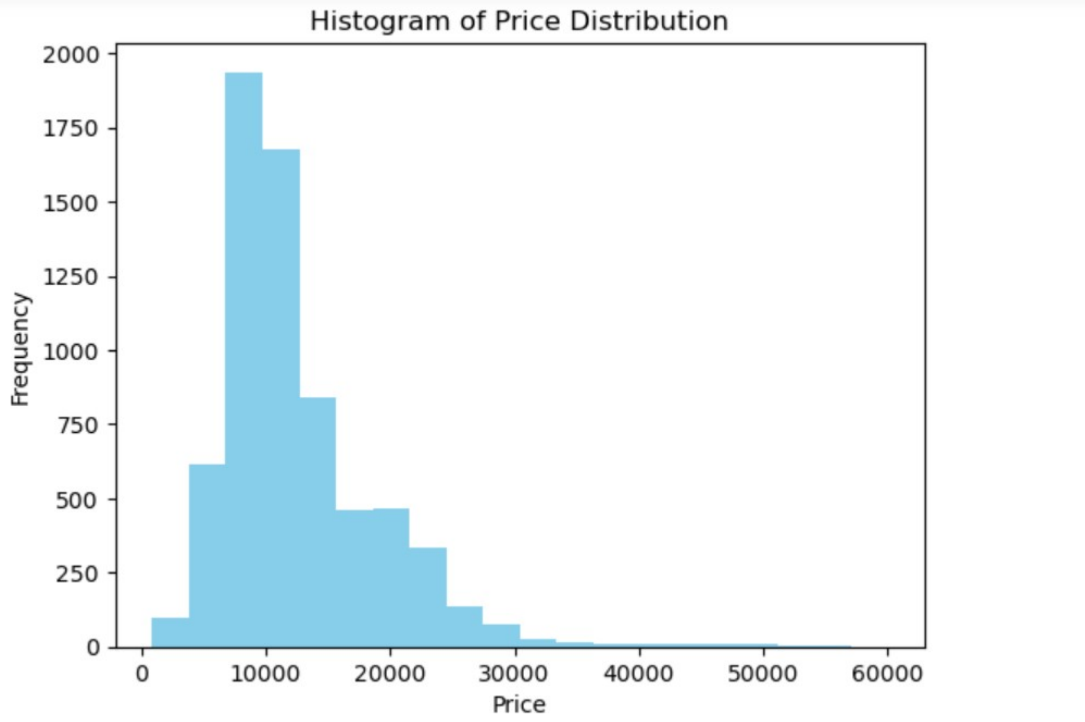


Figure Histogram or price distribution

The Histogram shows the price distribution of a product. The x-axis shows the price range, and the y-axis shows the frequency of prices in each range. The tallest bar is in the range of €20,000 to €30,000, which means that the most common price for the product is in this range. There are also a significant number of products priced between €10,000 and €20,000, and a few products priced higher than €50,000.

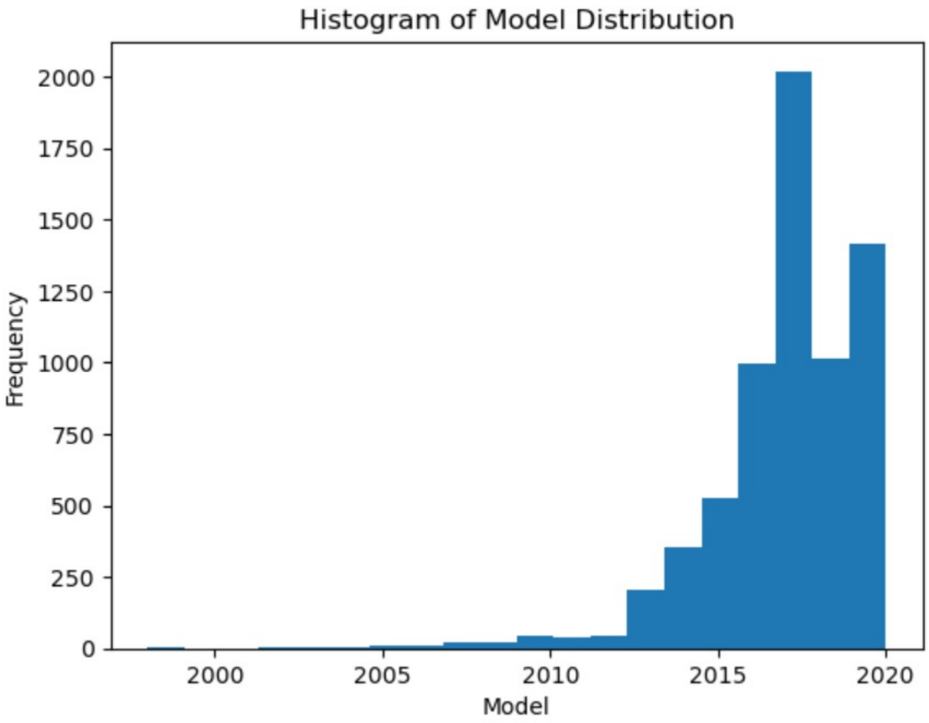


Figure Histogram or Model distribution

It shows a histogram of the number of models in each year. The x-axis shows the year, and the y-axis shows the number of models. The tallest bar is in the year 2010, which means that the most number of models were created in this year. There are also a significant number of models created in the years 2005 and 2015, and fewer models created in the years 2000 and 2020.

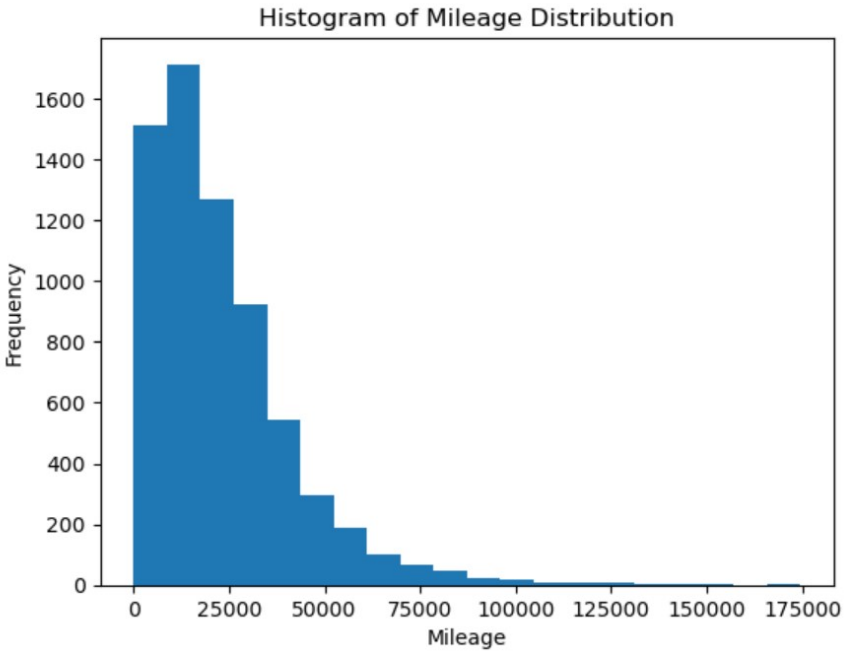
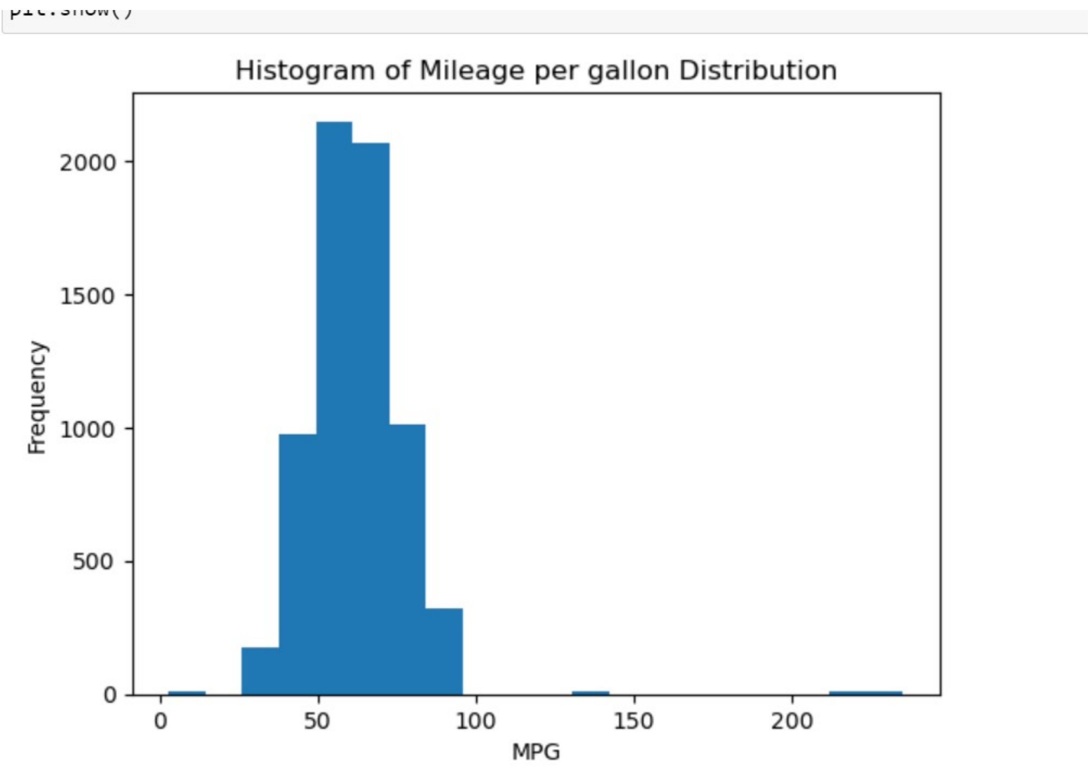


Figure Histogram of Mileage Distribution

It shows a histogram of the mileage distribution of a car. The x-axis shows the mileage range, and the y-axis shows the frequency of cars in each range. The tallest bar is in the range of 75,000 to 100,000 miles, which means that the most common mileage for the car is in this range. There are also a significant number of cars with mileage between 50,000 and 75,000 miles, and fewer cars with mileage higher than 125,000 miles.



```
plt.show()
```

Figure Histogram of mileage per gallon

It shows a histogram of the mileage per gallon distribution of cars. The x-axis shows the mileage per gallon (MPG), and the y-axis shows the frequency of cars in each range. The tallest bar is in the range of 35 to 40 MPG, which means that the most common mileage per gallon for cars is in this range. There are also a significant number of cars with mileage per gallon between 30 and 35 MPG, and fewer cars with mileage per gallon lower than 25 MPG.

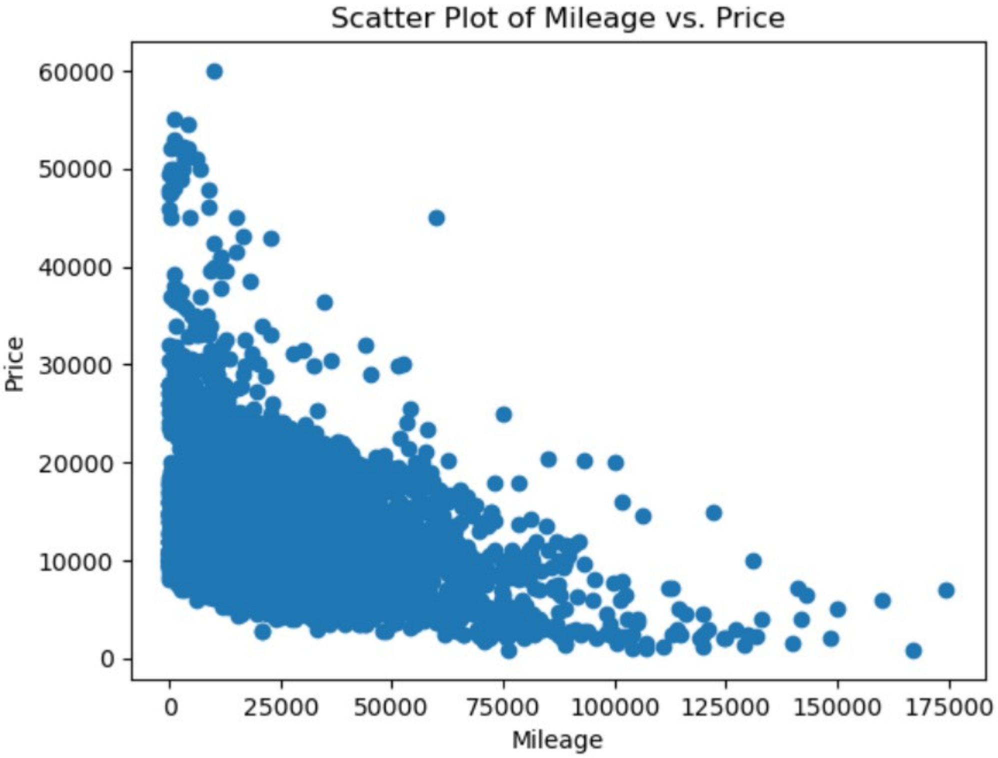


Figure Scatter plot of Mileage vs. Price

It shows a scatter plot of mileage vs. price for used cars. The x-axis shows the mileage, and the y-axis shows the price. The points on the scatter plot show the relationship between mileage and price for individual cars.

There is a general trend of decreasing price as mileage increases. This means that cars with higher mileage tend to be less expensive than cars with lower mileage. However, there is also a lot of scatter in the data, which means that there are some cars with high mileage that are still expensive and some cars with low mileage that are still inexpensive.

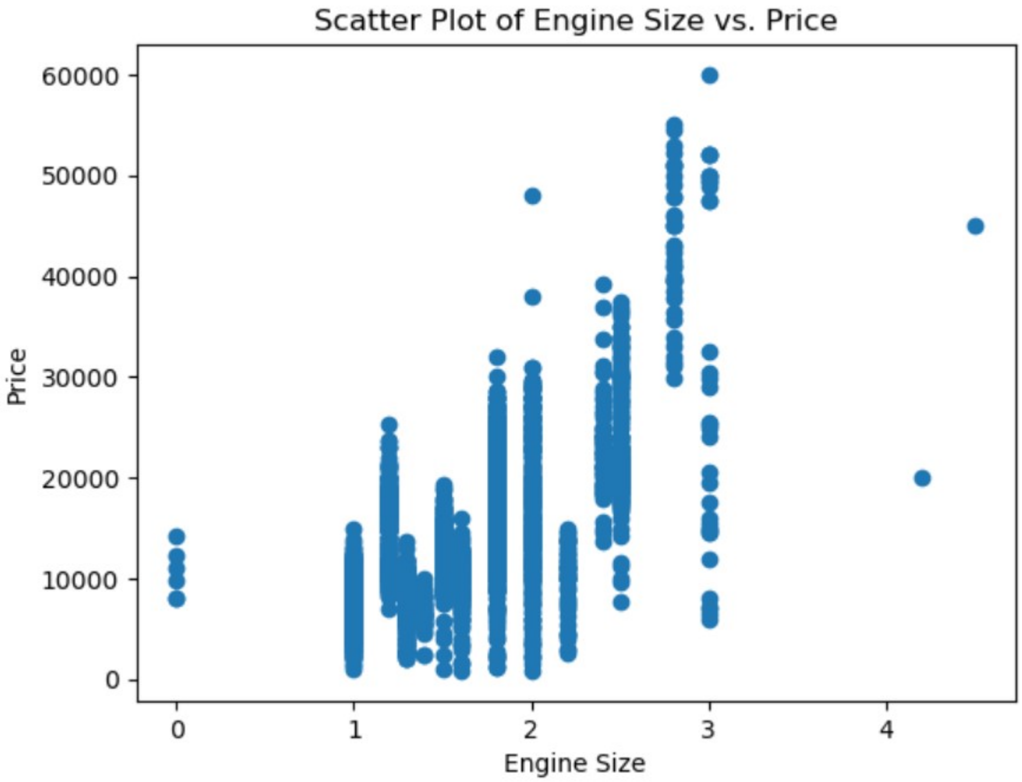


Figure Scatter plot of Mileage vs. Price

It shows a scatter plot of engine size vs. price for cars. The x-axis shows the engine size, and the y-axis shows the price. The points on the scatter plot show the relationship between engine size and price for individual cars.

There is a general trend of increasing price as engine size increases. This means that cars with larger engines tend to be more expensive than cars with smaller engines. However, there is also a lot of scatter in the data, which means that there are some cars with large engines that are still inexpensive and some cars with small engines that are still expensive.

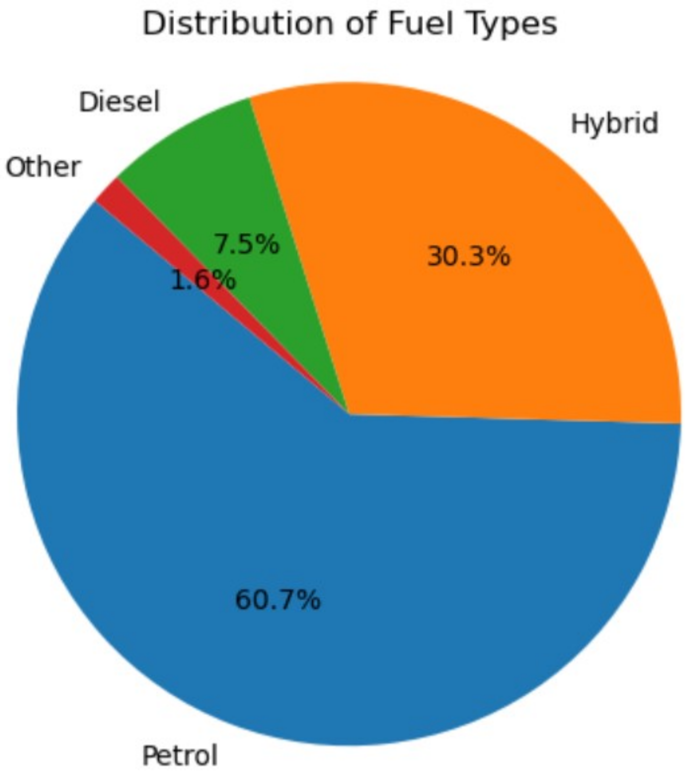
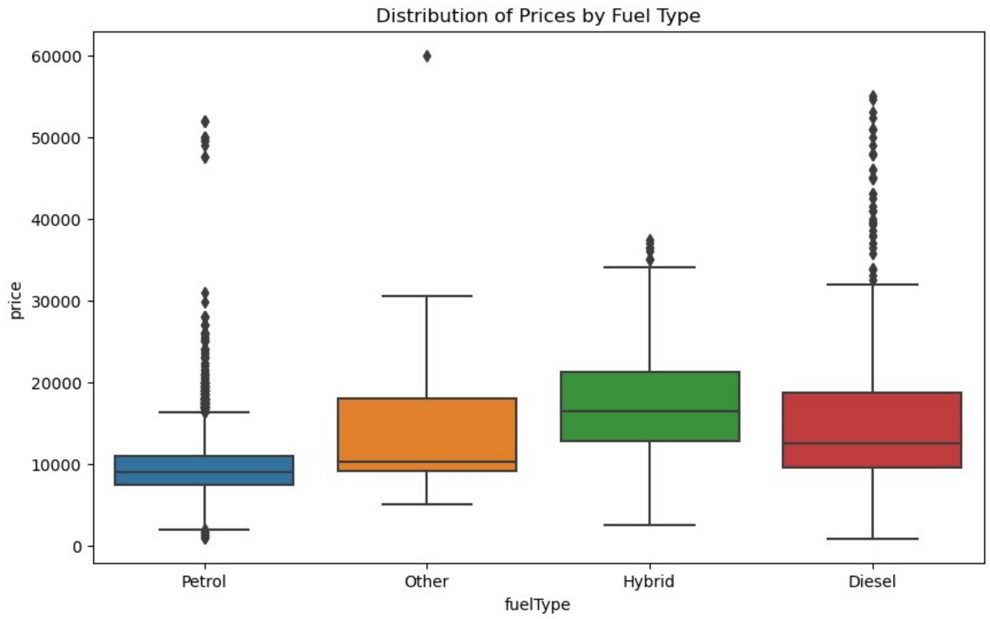


Figure Pie plot for Fuel type

The image you sent me is a pie chart showing the distribution of fuel types in the United States. The pie chart is divided into five sections, each representing a different fuel type. The largest section is for petrol, which accounts for 60.7% of all fuel used in the United States. The next largest section is for diesel, which accounts for 30.3% of all fuel used. The remaining sections are for hybrid (1.69%), other (3.0%), and unknown (4.0%).

- **EDA for the Dataset**



It shows a box plot of the distribution of prices by fuel type. The box plot has five parts:

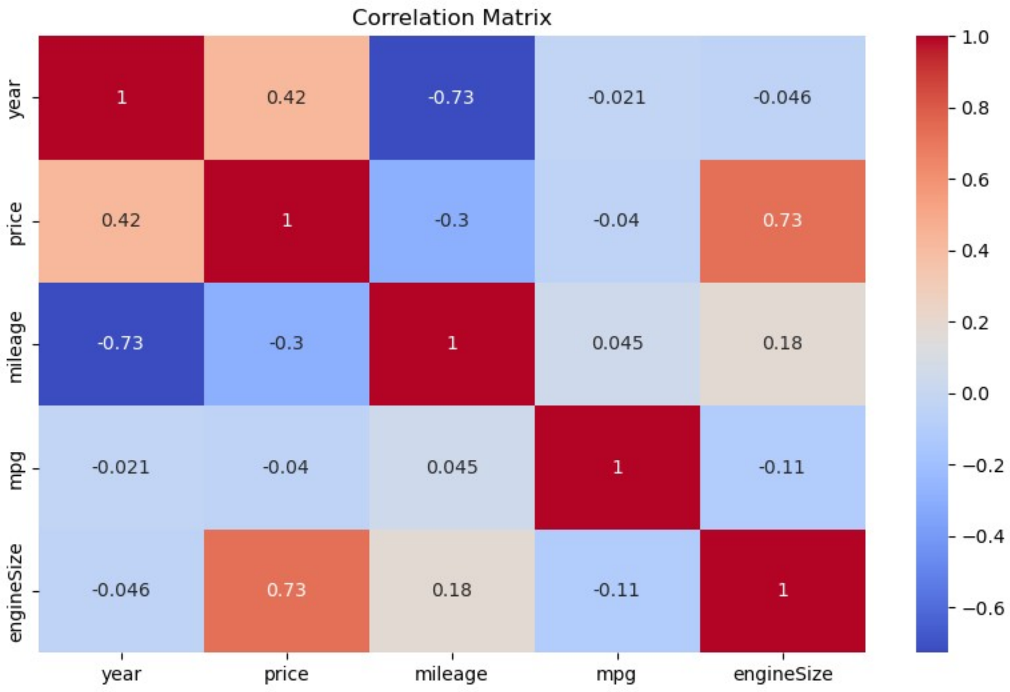
The box shows the middle 50% of the data, from the first quartile (Q1) to the third quartile (Q3).

The median is the middle value of the data.

The whiskers extend to the most extreme data points within 1.5 times the interquartile range (IQR) of the box.

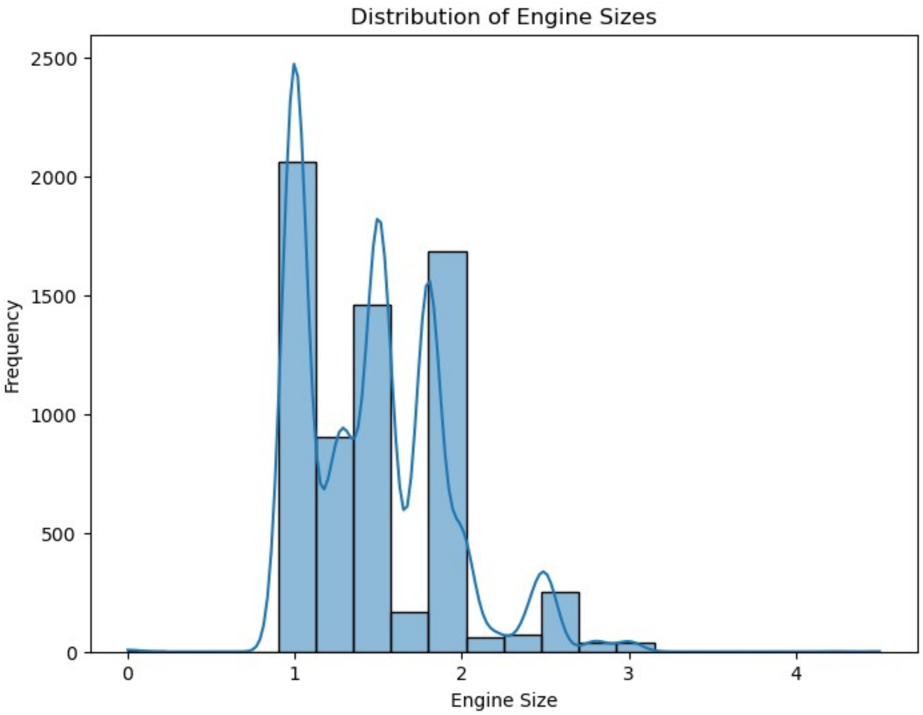
Any points beyond the whiskers are considered outliers.

The box plot shows that the median price for cars with petrol fuel is 50,000. The median price for cars with diesel fuel is 40,000. There are a few outliers for cars with petrol fuel, with prices up to 70,000. There are no outliers for cars with diesel fuel.



The correlation matrix shows that there are a few strong correlations between the variables. The correlation between price and engineSize is 0.73, which indicates a strong positive correlation. This means that the price of a car tends to increase as the size of its engine increases. The correlation between price and mileage is -0.73, which indicates a strong negative correlation. This means that the price of a car tends to decrease as the number of miles it has been driven increases.

There are also a few moderate correlations between the variables. The correlation between year and price is 0.42, which indicates a moderate positive correlation. This means that the price of a car tends to increase as the year it was manufactured increases. The correlation between mileage and engineSize is 0.6, which indicates a moderate positive correlation. This means that the number of miles a car has been driven tends to increase as the size of its engine increases.



It shows a histogram of the distribution of engine sizes. The x-axis shows the engine size in cubic centimeters (cc), and the y-axis shows the frequency of engine sizes. The tallest bar is in the range of 2000 to 2500 cc, which means that the most common engine size is in this range. There are also a significant number of engines with sizes between 1500 and 2000 cc, and fewer engines with sizes higher than 3000 cc.

Bi-variate Analysis:

It shows a pair plot of price vs. year for used cars. The x-axis shows the year the car was manufactured, and the y-axis shows the price of the car. The points on the scatter plot show the relationship between year and price for individual cars.

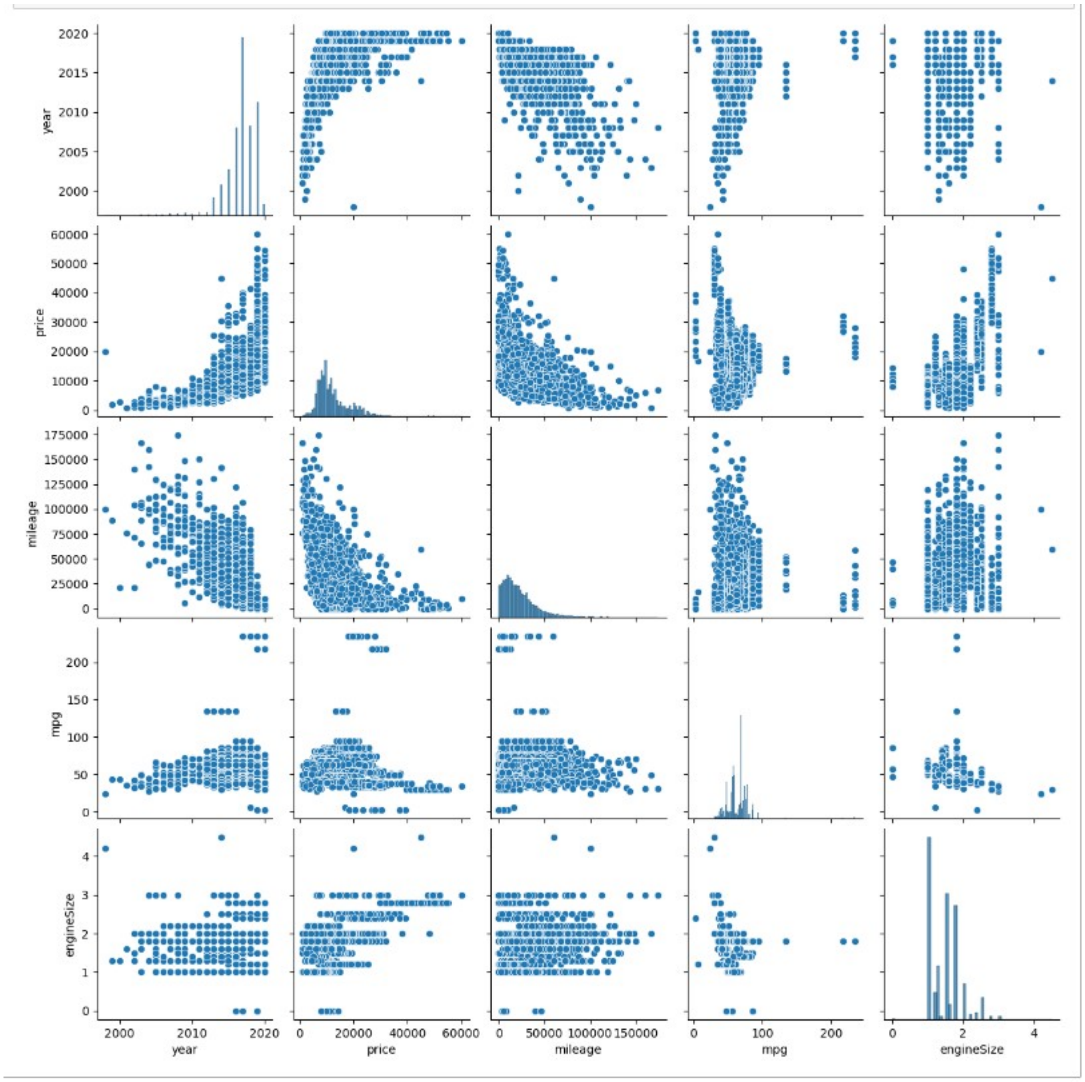
There is a general trend of increasing price as year increases. This means that used cars tend to be more expensive in recent years than in earlier years. However, there is also a lot of scatter in the data, which means that there are some cars that are more expensive than expected for their year and some cars that are less expensive than expected for their year.

There are a few possible reasons for the scatter in the data:

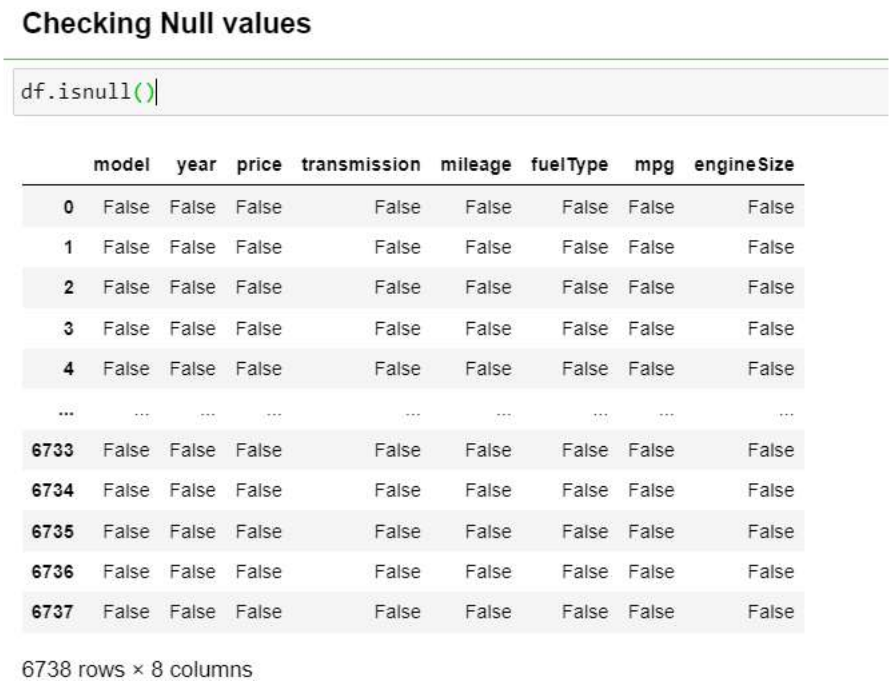
The cars in the dataset are a mix of different makes and models, so there is a variety of factors that affect their prices, such as condition, mileage, and location.

The cars in the dataset may have been equipped with different features, such as luxury amenities or performance upgrades, which could affect their prices.

The cars in the dataset may have been priced differently by their owners, depending on their individual circumstances.



2.1 Data preprocessing and handling missing value:

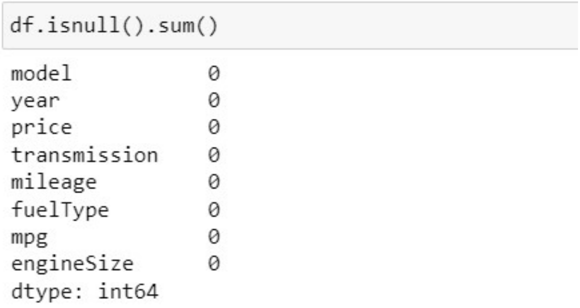


Checking Null values

```
df.isnull()
```

	model	year	price	transmission	mileage	fuelType	mpg	engineSize
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...
6733	False	False	False	False	False	False	False	False
6734	False	False	False	False	False	False	False	False
6735	False	False	False	False	False	False	False	False
6736	False	False	False	False	False	False	False	False
6737	False	False	False	False	False	False	False	False

6738 rows × 8 columns

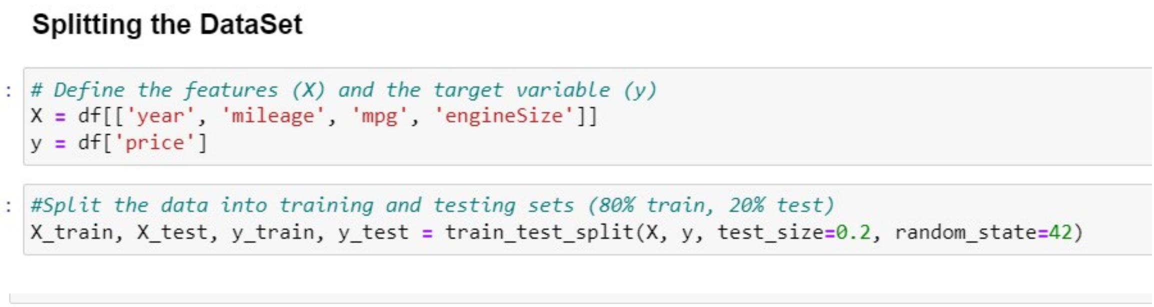


```
df.isnull().sum()
```

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

As there is no null values , we have the clean dataset .

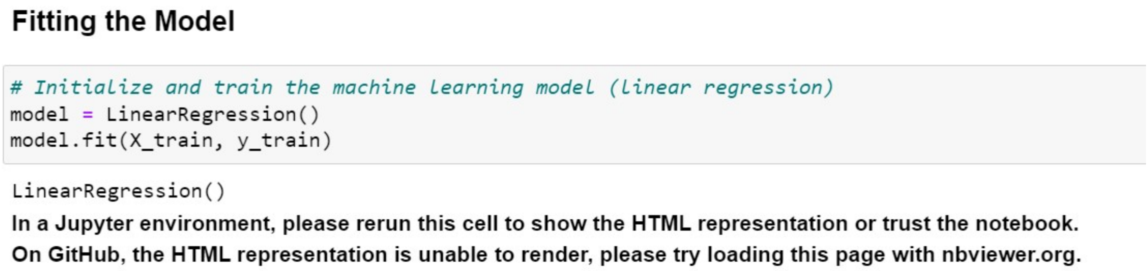
2.2 Spilting the dataset into test and training data and building Machine Learning model using training data:



Splitting the DataSet

```
: # Define the features (X) and the target variable (y)
  X = df[['year', 'mileage', 'mpg', 'engineSize']]
  y = df['price']
```

```
: #Split the data into training and testing sets (80% train, 20% test)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

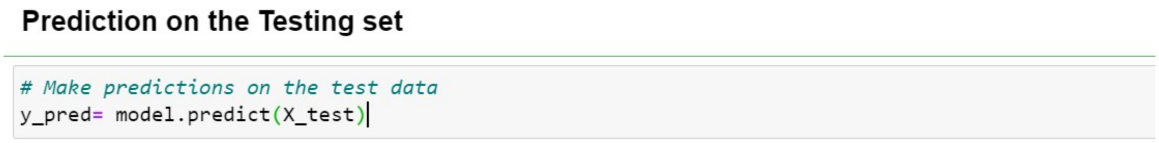


Fitting the Model

```
# Initialize and train the machine learning model (linear regression)
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.



Prediction on the Testing set

```
# Make predictions on the test data
y_pred= model.predict(X_test)
```

In X variables definer the features and y variables definesvthe target variable. The features are the columns that will be used to predict the target variable. In this case, the features are year, mileage, mpg, and engineSize. The target variable is the price of the car.

train_test_split() function to split the dataset into a training set and a test set. The test size is set to 0.2, which means that 20% of the data will be used for the test set and 80% of the data will be used for the training set. The random state is set to 42, which ensures that the splitting of the data is reproducible.

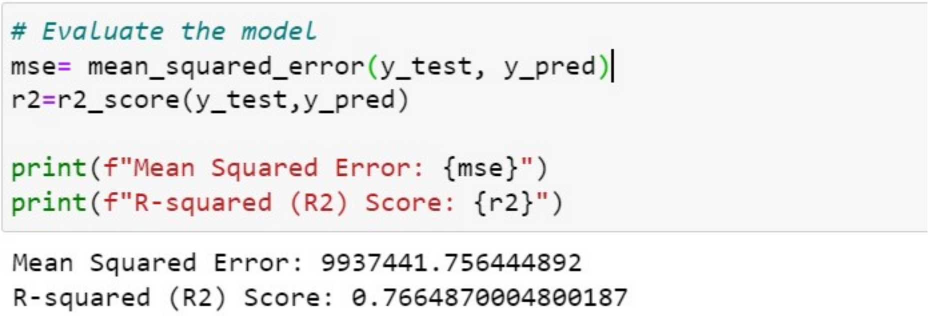
We assigns the training set to the variable X_train and the test set to the variable X_test. The target variable is also split into two variables, y_train and y_test.

Then we initialize the machine learning model. In this case, the model is a linear regression model. Linear regression is a simple machine learning model that can be used to predict a continuous variable from a set of features.

Then we train the model on the training set. The training set is the data that the model will use to learn how to make predictions.

Then we make predictions on the testing set. The testing set is the data that the model has not seen before. This is used to test the accuracy of the model's predictions.

2.4 Evaluate performance of the model for test set:



```
# Evaluate the model
mse= mean_squared_error(y_test, y_pred)
r2=r2_score(y_test,y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared (R2) Score: {r2}")
```

```
Mean Squared Error: 9937441.756444892
R-squared (R2) Score: 0.7664870004800187
```

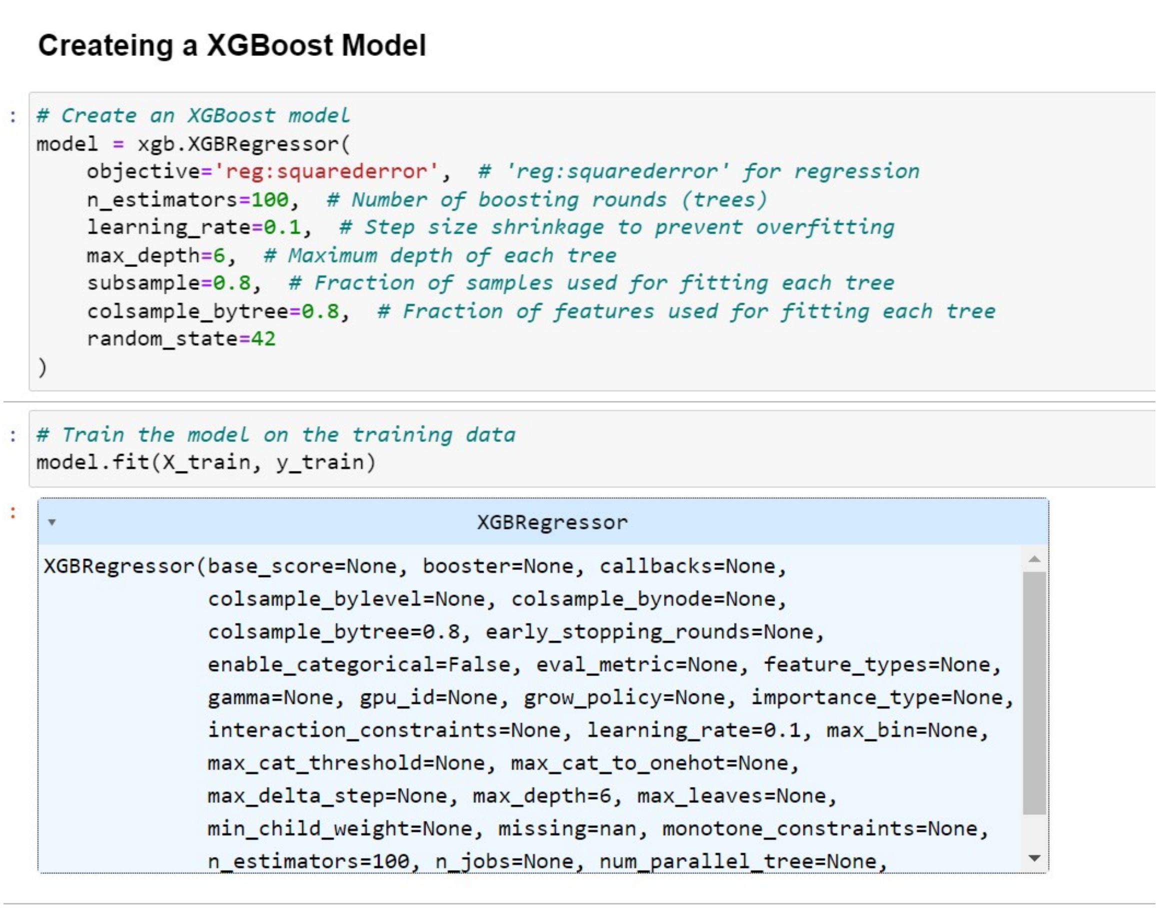
Here we calculate the mean squared error (MSE) between the predicted prices and the actual prices. The MSE is a measure of the accuracy of the model's predictions.

For evaluation of the MSE to the console . The MSE is 9937441.756444892, which means that the model's predictions are imprecise.

We also calculated the R-squared score to the console. The R-squared score is a measure of how much of the variance in the target variable is explained by the model. The R-squared score in the image is 0.7664870004800187, which means that the model explains about 76% of the variance in the price of the car.

2.4 Perform hyperparameter tuning and obtain the best predictive model:

For hypertunen our model we here use XGBoost Model. XGBoost is a popular machine learning algorithm that is used for both classification and regression tasks. It is an ensemble learning method that combines the predictions of multiple weak models to produce a stronger prediction. XGBoost stands for Extreme Gradient Boosting, and it is a regularized form of the gradient boosting algorithm. This means that it is more robust to overfitting and can achieve better performance on a wider range of datasets.



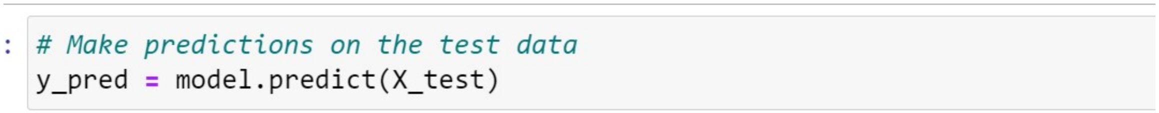
Createing a XGBoost Model

```
# Create an XGBoost model
model = xgb.XGBRegressor(
    objective='reg:squarederror',  # 'reg:squarederror' for regression
    n_estimators=100,  # Number of boosting rounds (trees)
    learning_rate=0.1,  # Step size shrinkage to prevent overfitting
    max_depth=6,  # Maximum depth of each tree
    subsample=0.8,  # Fraction of samples used for fitting each tree
    colsample_bytree=0.8,  # Fraction of features used for fitting each tree
    random_state=42
)
```

```
# Train the model on the training data
model.fit(X_train, y_train)
```

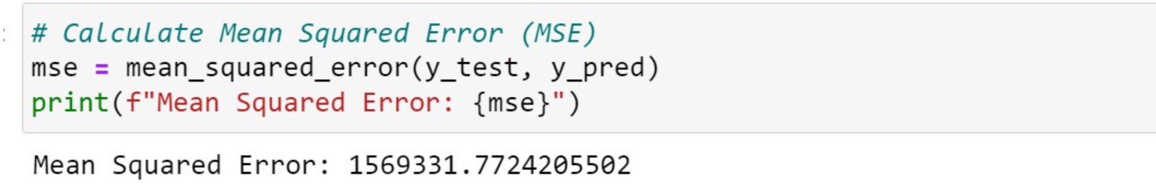
```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=6, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
```

Then we train the model for predicting after hypertuning the model with XGBoostRegressor.



```
# Make predictions on the test data
y_pred = model.predict(X_test)
```

Then we make the model preditc our target variables based on their features.



```
# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

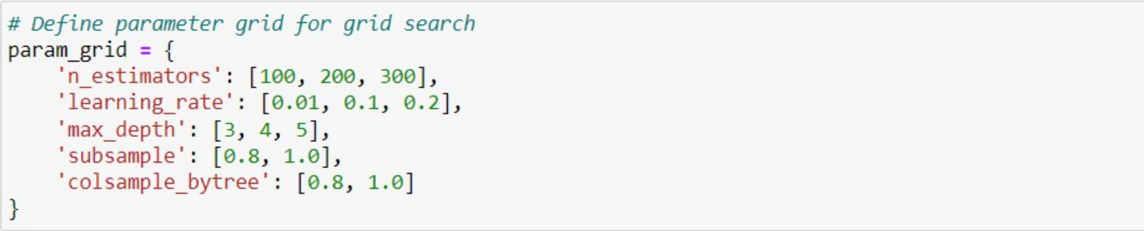
```
Mean Squared Error: 1569331.7724205502
```

Then we calculate the mean squared error (MSE) between the predicted and actual values. The program first imports the mean_squared_error function from the sklearn.metrics library. Then, it defines two variables, y_test and y_pred, which store the actual and predicted values, respectively. Finally, it calls the mean_squared_error() function to calculate the MSE and prints the result to the console.

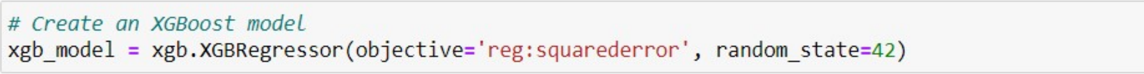
The MSE is a measure of the average squared difference between the predicted and actual values. A lower MSE indicates that the predictions are closer to the actual values. In this case, the MSE is 1569331.77, which is a relatively high value. This suggests that the predictions are imprecise.

2.5 Apply performance improvement techniques :

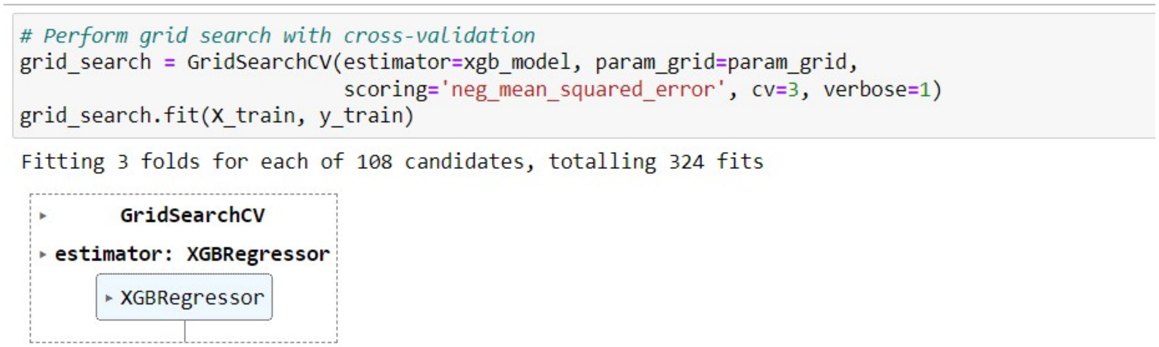
We will apply XGBoost model for imporovement the performance of our model.



```
# Define parameter grid for grid search
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}
```



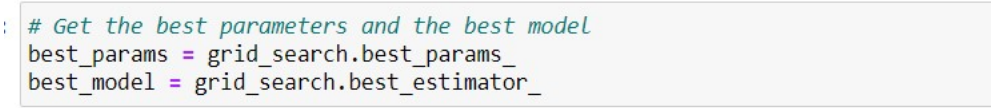
```
# Create an XGBoost model
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)
```



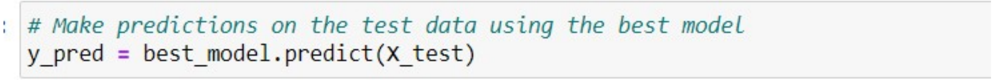
```
# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid,
                           scoring='neg_mean_squared_error', cv=3, verbose=1)
grid_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
```

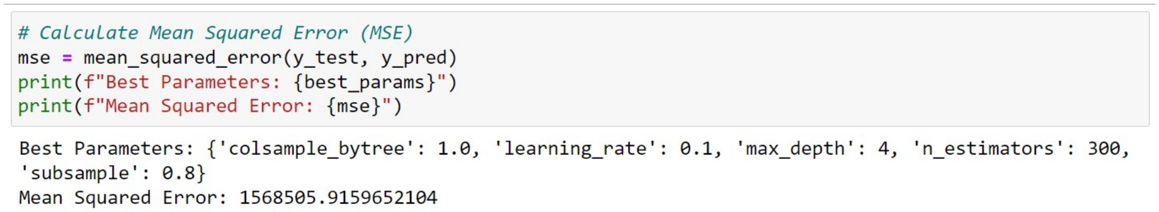
here we create our xgboost model with random state 42(standard) and Also did a GridSearch for xgb model.



```
# Get the best parameters and the best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
```



```
# Make predictions on the test data using the best model
y_pred = best_model.predict(X_test)
```

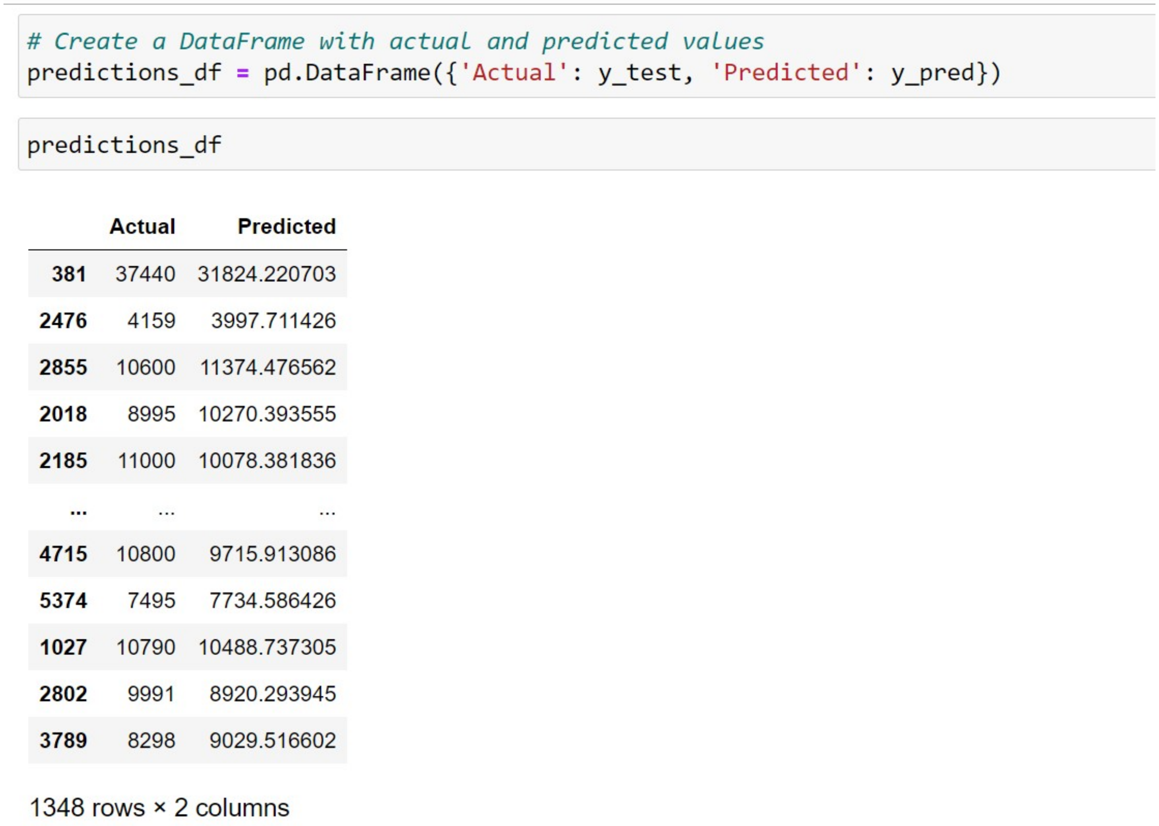


```
# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Best Parameters: {best_params}")
print(f"Mean Squared Error: {mse}")
```

```
Best Parameters: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 300,
'subsample': 0.8}
Mean Squared Error: 1568505.9159652104
```

We got our best parameters and Mean Sqaured Error . Now we can predict our actual and predicted value and estimate our model .

3. 5 Get our final model and obtain the predicted target values using the model



```
# Create a DataFrame with actual and predicted values
predictions_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
```

```
predictions_df
```

	Actual	Predicted
381	37440	31824.220703
2476	4159	3997.711426
2855	10600	11374.476562
2018	8995	10270.393555
2185	11000	10078.381836
...	...	...
4715	10800	9715.913086
5374	7495	7734.586426
1027	10790	10488.737305
2802	9991	8920.293945
3789	8298	9029.516602

1348 rows × 2 columns

It shows a dataframe with 1348 rows and 2 columns. The first column is named "Actual" and contains the actual values of the selling price of used cars. The second column is named "Predicted" and contains the predicted values of the selling price of used cars.