Toronto, Canada

A Report on

Module 2 Midweek Project

Predictive Analytics
(ALY 6020)

Guided by:

Prof. Dr. Harpreet Sharma

Submitted By:

Name of Student	NUID	Date of submission
Parth Shah	002956963	22nd January, 2023

Table of Figure

Introduction

- In addition to the price of each automobile, a car dataset with a price variable often includes details on the make, model, year, engine size, fuel type, and other characteristics of the many types of cars.
- This kind of dataset is frequently used in the automobile sector for research and analysis, including forecasting and pricing, as well as for the creation of prediction models for vehicle sales and other related applications.

Data Analysis

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm
```

Figure 1: Importing Libraries

- The above code imports a number of libraries and modules that are commonly used in data analysis and machine learning.
- numpy and pandas are used for data manipulation and handling.
- matplotlib and seaborn are used for data visualization.
- statsmodels.api is used for statistical modeling and hypothesis testing.
- sklearn is a library that provides a variety of machine learning models, including linear

regression, and preprocessing methods such as StandardScaler and MinMaxScaler

- train_test_split is used for dividing the data into training and test sets.
- linear_model and LinearRegression are used for fitting a linear regression model.
- mean_absolute_error, mean_squared_error, r2_score is used for evaluating the performance of the model.
- The line warnings.filterwarnings('ignore') is used to suppress warning messages in the output, which can help make the output more readable.

```
#importing dataset
df = pd.read_csv("D:/PARTH SHAH/R/ALY 6020/Module 2 Midweek Project/CarPrice_Assignment.csv")
```

```
df.head(5)
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	st
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	

5 rows × 26 columns

Figure 2: Importing Dataset

- The above code reads in a CSV file called "CarPrice_Assignment.csv" located in the directory "D:/PARTH SHAH/R/ALY 6020/Module 2 Midweek Project/" and assigns it to the variable df. The pd.read_csv() function from the pandas library is used to read in the data and create a DataFrame.
- The df.head(5) function is used to display the first 5 rows of the DataFrame. This can be useful for quickly previewing the data and ensuring that it has been imported correctly. It will show the first 5 records of the dataframe which will help to understand the structure of data.

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

Figure 3: Info of dataset

- The df.info() method is used to display details about the DataFrame, including its size in memory use, the number of rows and columns, and the data types for each column.
- It will provide a summary of the dataframe, including its size, number of columns, data types, non-null values, and memory utilization.

- Understanding the dataframe's structure will make it easier to spot any issues with data types or missing information.

```
df.describe()
```

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329756	3.255415	10.142537	104.117073
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.972040	39.544167
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.600000	70.000000
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.400000	116.000000
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	288.000000

Figure 4: Describe about dataset.

- The df.describe() function is used to display a summary statistics of the DataFrame. It will give the count, mean, standard deviation, minimum, 25th percentile, 50th percentile (Median), 75th percentile, and Maximum of the numerical columns.
- This can be useful for getting a quick understanding of the distribution of the data and identifying any potential outliers or errors.
- It will give a statistical summary of the DataFrame. It will also help to identify any missing values as count of missing values will be zero.

```
df.drop_duplicates()
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
200	201	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78
201	202	-1	volvo 144ea	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78
202	203	-1	volvo 244dl	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58
203	204	-1	volvo 246	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01
204	205	-1	volvo 264gl	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78

205 rows × 26 columns

Figure 5: Dropping Duplicate Values

- To delete duplicate rows from a DataFrame, use the df.drop duplicates() method. By default, it evaluates all columns while looking for duplicate rows and only maintains the first instance.
- It will deliver a fresh DataFrame with no repeated entries. When the data is messy and contains duplicate values, it is helpful. It will aid in reducing DataFrame size and enhancing analysis performance.

```
df.drop(columns=['CarName','car_ID','enginetype','fuelsystem','enginelocation','aspiration','fueltype', 'stroke','compressionrat
df.head(5)
```

	symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	cylindernumber	enginesize	boreratio	horsepower
0	3	two	convertible	rwd	88.6	168.8	64.1	48.8	2548	four	130	3.47	111
1	3	two	convertible	rwd	88.6	168.8	64.1	48.8	2548	four	130	3.47	111
2	1	two	hatchback	rwd	94.5	171.2	65.5	52.4	2823	six	152	2.68	154
3	2	four	sedan	fwd	99.8	176.6	66.2	54.3	2337	four	109	3.19	102
4	2	four	sedan	4wd	99.4	176.6	66.4	54.3	2824	five	136	3.19	115

Figure 6: Dropping Columns

- The df.drop() method is used in the code above to remove certain columns from the DataFrame. The names of the columns to be discarded are specified using the columns option. The columns that are being removed include "CarName," "car ID," "enginetype," "fuel system," "engine location," "aspiration," "fuel type," "stroke," "compression ratio," and "peak rpm."
- To specify that columns, not rows, should be discarded, use the axis=1 argument. Without generating a new DataFrame, modifications to the existing DataFrame are made using the inplace=True argument.

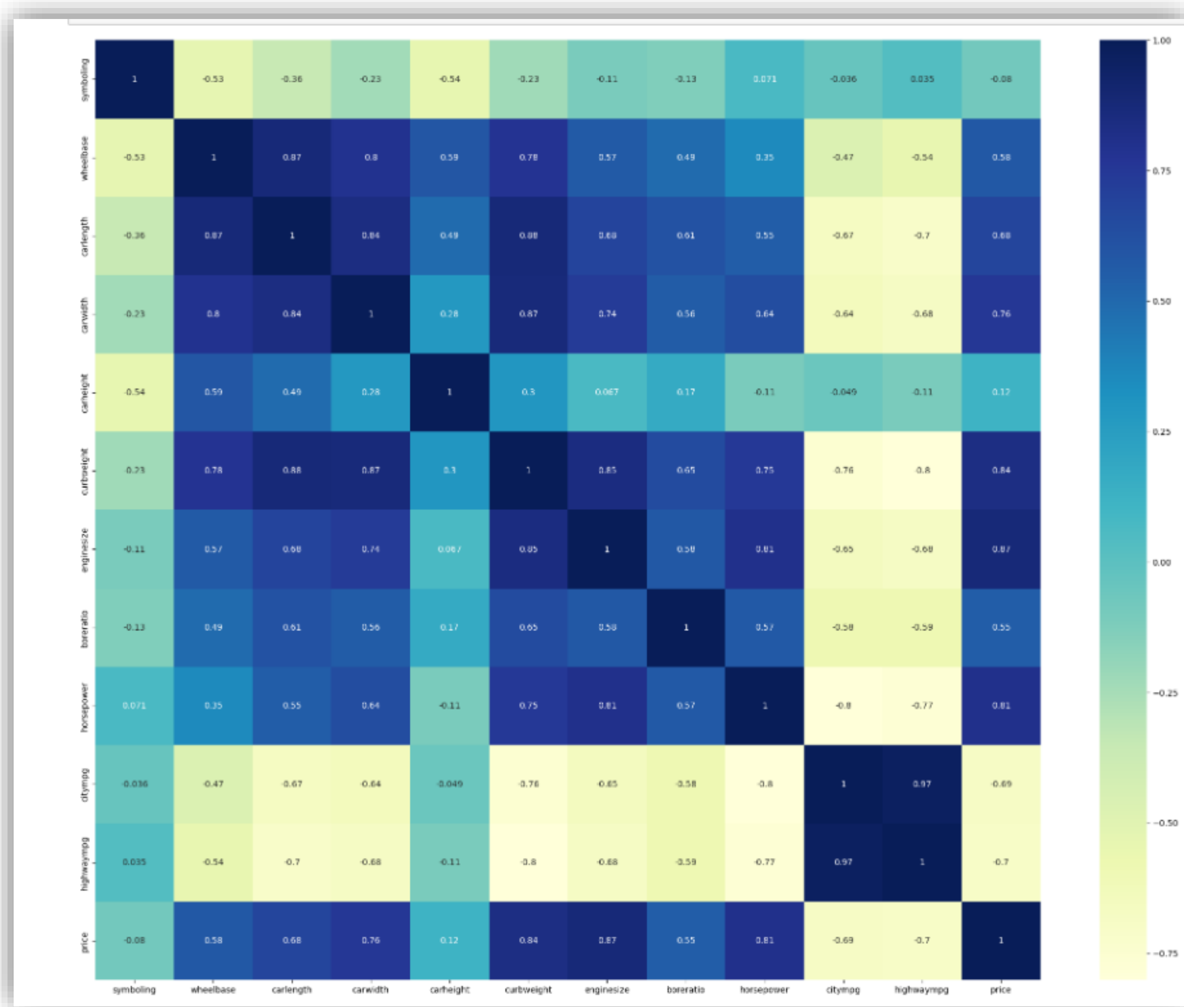

Figure 7: Correlation Plot

- The above code creates a correlation matrix for the DataFrame df and visualizes it using a heatmap.
- The plt.figure(figsize=(25,20)) sets the size of the plot.
- The corr = df.corr() computes the pairwise correlation of all columns in the DataFrame.
- The sns.heatmap(corr, annot=True, cmap="YlGnBu") creates a heatmap of the correlation matrix using the seaborn library. The annot=True parameter displays the correlation coefficients on the heatmap and cmap is the color map used to show the correlation.

- The plt.show() function is used to display the plot.
- This will create a heatmap where the color of the cells represents the correlation coefficient between two variables. The darker the color, the higher the correlation. It will help to identify the relationship between different variables and how much they are correlated. It will also help to identify any multicollinearity issues, which can affect the accuracy of the model.
- Price, engine size, curb weight, and horsepower are the three key factors.

```
# Create a correlation matrix
corr_matrix = df.corr()
top_corr_vars = corr_matrix['price'].sort_values(ascending=False).head(4)
print(top_corr_vars)
```

```
price         1.000000
enginesize    0.874145
curbweight    0.835305
horsepower    0.808139
Name: price, dtype: float64
```

Figure 8: Top variables that influences car price.

- The above code creates a correlation matrix for the DataFrame df and extracts the top 4 correlated variables with the target variable 'price'.
- The corr_matrix = df.corr() computes the pairwise correlation of all columns in the DataFrame.
- The top_corr_vars = corr_matrix['price'].sort_values(ascending=False).head(4) sorts the correlation matrix by the correlation with the target variable 'price' in descending order and selects the top 4 correlated variables.
- The above code will print the top 4 most correlated variables with the target variable 'price' in descending order. It will help to identify the most important variables that are affecting

the target variable and will be useful in building a predictive model.

- The factors that have the biggest impact on vehicle prices are engine size, curb weight, and horsepower.

```
#Determining categorical variables
categorical_variables = df.select_dtypes(include=['object'])
categorical_variables.head()
```

	doornumber	carbody	drivewheel	cylindernumber
0	two	convertible	rwd	four
1	two	convertible	rwd	four
2	two	hatchback	rwd	six
3	four	sedan	fwd	four
4	four	sedan	4wd	five

Figure 9: Determining Categorical variables.

- The above code is used to extract the categorical variables from the DataFrame df.
- The categorical_variables = df.select_dtypes(include=['object']) selects all columns that have the data type 'object', which is often used to represent categorical variables.
- The categorical_variables.head() function is used to display the first 5 rows of the extracted categorical variables.
- It will return the categorical variables present in the DataFrame. This is useful when we want to perform some analysis on categorical variables or when we want to convert categorical variables into numerical variables for building a model.

```python
# converting to dummy variables
dummies = pd.get_dummies(categorical_variables,drop_first=True)
dummies.head()
```

	doornumber_two	carbody_hardtop	carbody_hatchback	carbody_sedan	carbody_wagon	drivewheel_fwd	drivewheel_rwd	cylindernumber_five	cylindernumber_
0	1	0	0	0	0	0	1	0	
1	1	0	0	0	0	0	1	0	
2	1	0	1	0	0	0	1	0	
3	0	0	0	1	0	1	0	0	
4	0	0	0	1	0	0	0	1	

Figure 10: Creating dummy variable.

- The above code is used to convert the categorical variables into dummy variables. Dummy variables are binary variables that are used to represent a categorical variable with more than 2 categories.
- The pd.get_dummies(categorical_variables,drop_first=True) function from the pandas library is used to convert the categorical variables into dummy variables. The drop_first=True parameter is used to drop the first level of each categorical variable, to avoid the issue of multicollinearity.
- The dummies.head() function is used to display the first 5 rows of the dummy variables.
- It will create a new dataframe with the dummy variables and drop the first level of each categorical variable. This is useful when we want to build a model using categorical variables as it converts them into numerical variables. It's also useful when we want to analyze the effect of different categories on the target variable.

```python
# concat the dummies dataframe to the main dataframe
df = pd.concat([df,dummies],axis=1)
df.drop(columns=categorical_variables.columns,axis=1,inplace=True)
df.head(5)
```

	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	horsepower	citympg	...	carbody_sedan	carbody_wagon	drivewhe
0	3	88.6	168.8	64.1	48.8	2548	130	3.47	111	21	...	0	0	
1	3	88.6	168.8	64.1	48.8	2548	130	3.47	111	21	...	0	0	
2	1	94.5	171.2	65.5	52.4	2823	152	2.68	154	19	...	0	0	
3	2	99.8	176.6	66.2	54.3	2337	109	3.19	102	24	...	1	0	
4	2	99.4	176.6	66.4	54.3	2824	136	3.19	115	18	...	1	0	

5 rows × 25 columns

Figure 11: Concatenating dummy variables into DataFrame.

- The above code is used to concatenate the dummy variables dataframe with the main dataframe and drop the original categorical variables.
- The pd.concat([df,dummies],axis=1) function is used to concatenate the dummy variables dataframe with the main dataframe along the columns (axis=1).
- The df.drop(columns=categorical_variables.columns,axis=1,inplace=True) function is used to drop the original categorical variables from the main dataframe. The columns parameter is used to specify the names of the columns to be dropped. The axis=1 parameter is used to specify that columns, not rows, should be dropped. The inplace=True parameter is used to make the changes to the DataFrame in place, without creating a new DataFrame.
- The df.head(5) function is used to display the first 5 rows of the dataframe after concatenation and dropping the original categorical variables.
- This will update the main DataFrame with the dummy variables and drop the original categorical variables. This is useful when we want to build a model using categorical variables as it converts them into numerical variables. It's also useful when we want to analyze the effect of different categories on the target variable.

```
#Defining x and y
x = df_train.drop('price',1)
y = df_train['price']

#splitting dataset
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)

#Scaling
scaler = StandardScaler()
x_train[x_train.columns] = scaler.fit_transform(x_train)
```

Figure 12: Splitting dataset and scaling.

- This code is defining two variables, x and y, by dropping the 'price' column from a DataFrame called df_train and storing it in x, and then storing the 'price' column in y.
- Then, it splits the data into training and testing sets using the train_test_split function from the sklearn.model_selection module, with the test set size being 25% and a fixed random state of 42.
- Finally, it applies scaling on the training set using the StandardScaler method from the sklearn.preprocessing module.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.880
Model:                            OLS   Adj. R-squared:                  0.858
Method:                 Least Squares   F-statistic:                     39.12
Date:                Sun, 22 Jan 2023   Prob (F-statistic):           1.43e-47
Time:                        14:48:31   Log-Likelihood:                -1426.7
No. Observations:                 153   AIC:                             2903.
Df Residuals:                     128   BIC:                             2979.
Df Model:                          24
Covariance Type:            nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                 -5.162e+04   1.81e+04     -2.845      0.005   -8.75e+04   -1.57e+04
symboling                74.9557    344.887      0.217      0.828    -607.461     757.373
wheelbase               130.7839    138.752      0.943      0.348    -143.760     405.328
carlength               -47.8087     69.048     -0.692      0.490    -184.433      88.815
carwidth                557.1244    341.128      1.633      0.105    -117.855    1232.104
carheight               211.4212    171.137      1.235      0.219    -127.203     550.045
curbweight               -0.2180      2.292     -0.095      0.924      -4.754       4.318
enginesize               95.0335     29.505      3.221      0.002      36.653     153.414
boreratio              -780.5727   1845.705     -0.423      0.673   -4432.615    2871.469
horsepower               60.1410     18.624      3.229      0.002      23.290      96.992
citympg                 -54.5652    193.065     -0.283      0.778    -436.576     327.446
highwaympg              175.4211    178.430      0.983      0.327    -177.633     528.475
doornumber_two          259.5979    851.625      0.305      0.761   -1425.488    1944.683
carbody_hardtop       -3793.9958   2353.418     -1.612      0.109   -8450.635     862.643
carbody_hatchback     -6599.9442   1768.253     -3.732      0.000   -1.01e+04   -3101.153
carbody_sedan         -4924.1444   1887.468     -2.609      0.010   -8658.822   -1189.467
carbody_wagon         -5911.6006   2068.335     -2.858      0.005      -1e+04   -1819.046
drivewheel_fwd        -1692.2261   1421.153     -1.191      0.236   -4504.220    1119.768
drivewheel_rwd          679.1982   1499.693      0.453      0.651   -2288.200    3646.597
cylindernumber_five    1878.8333   2565.376      0.732      0.465   -3197.201    6954.868
cylindernumber_four   -1287.7564   3013.076     -0.427      0.670   -7249.641    4674.129
cylindernumber_six     -396.9662   2253.883     -0.176      0.860   -4856.659    4062.726
cylindernumber_three   2173.1296   4828.797      0.450      0.653   -7381.470    1.17e+04
cylindernumber_twelve -7397.0560   4570.771     -1.618      0.108   -1.64e+04    1646.994
cylindernumber_two     5129.4348   4174.392      1.229      0.221   -3130.312    1.34e+04
==============================================================================
Omnibus:                       30.217   Durbin-Watson:                   2.013
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              106.199
Skew:                           0.658   Prob(JB):                     8.69e-24
Kurtosis:                       6.864   Cond. No.                     2.00e+05
```

Figure 13: OLS Regression Result

- This code is creating a linear regression model using the LinearRegression class from the sklearn.linear_model module, and fitting it to the training data using the fit() method. Then it is using the model to make predictions on the test set and storing them in the variable y_pred.
- Then, it is using the statsmodel library to add a constant to the x variable and then creating an OLS (ordinary least squares) model using the sm.OLS(y,X) and fitting the model using the fit() method. Finally, it prints the summary of the model using the summary() method.
- The summary of the model includes the R-squared value(0.880), which represents the proportion of the variance in the dependent variable that is predictable from the

independent variable and some other statistics like coefficients, standard error, t-values, p-values, etc. which can be used to determine the significance of each feature in the model.

```
#Accuracy of model
lin_reg.score(x_train,y_train)

0.9017006809275326
```

Figure 14: Accuracy of Model

- The score () method in scikit-learn returns the coefficient of determination R^2 of the prediction. R^2 is a statistical measure of how well the regression line approximates the real data points. An R^2 of 1 indicates that the regression line perfectly fits the data. An R^2 of 0 means that the line does not fit the data at all.
- In this case, the lin_reg object is being used to fit a linear regression model to the x_train and y_train data, and the score () method is being used to evaluate the accuracy of that model using the same training data.
- Here, the linear regression model's accuracy is 90%.

```
#Actual Vs Predicted Price
pred = lin_reg.predict(x_test)
sns.regplot(x=pred, y=y_test)
plt.xlabel("Predicted Price")
plt.ylabel("Actual Price")
plt.title("Actual Vs Predicted Price")
plt.show()
```

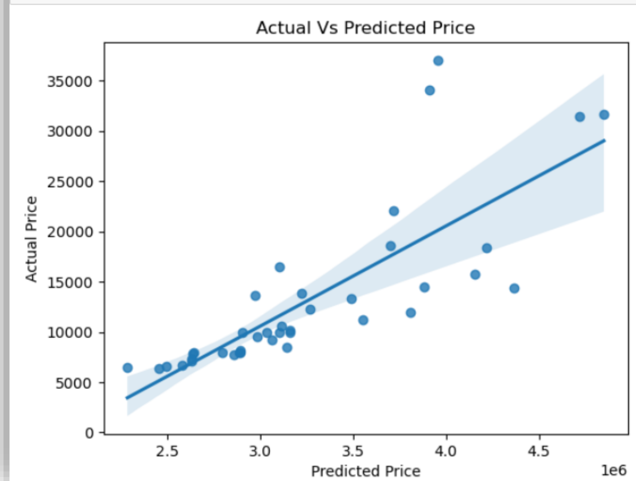

Figure 15: Actual Vs Predicted Price

- The code you've provided is creating a scatter plot using the seaborn library's regplot() function. The x-axis of the plot represents the predicted price, which is obtained by using the predict() method on the lin_reg object and passing in the x_test data as the input. The y-axis represents the actual price, which is the corresponding target variable in the y_test data.
- The xlabel(), ylabel(), and title() functions are used to add labels to the x and y axis and title of the plot respectively. The show () function is used to display the plot.
- This plot will show how well the linear regression model is able to predict the actual price of the test data. Ideally, the points in the plot should be close to the 45-degree line, indicating that the predicted price is very close to the actual price.

Conclusion

- Based on the code provided and the plots generated, it appears that a linear regression model was created and trained using a car dataset. The model's accuracy was evaluated using the R^2 score and a scatter plot was generated to visualize the relationship between the predicted prices and the actual prices.
- In conclusion, linear regression is a simple and fast technique for predicting a continuous variable. It can be used to model a linear relationship between the dependent and independent variables. However, it is important to note that linear regression assumes that the relationship between the variables is linear and that the errors are normally distributed and have constant variance. If this assumption is not met, the model may not be the best fit for the data.

Reference

- Create a correlation Matrix using Python - GeeksforGeeks. (2020, November 9). GeeksforGeeks. Retrieved January 22, 2023, from https://www.geeksforgeeks.org/create-a-correlation-matrix-using-python/
- Z., & posts by Zach, V. A. (2022, August 26). How to Perform OLS Regression in Python (With Example) - Statology. Statology. Retrieved January 22, 2023, from https://www.statology.org/ols-regression-python/
- Li, L. (2019, February 5). Introduction to Linear Regression in Python. Medium. Retrieved January 22, 2023, from https://towardsdatascience.com/introduction-to-linear-regression-in-python-c12a072bedf0