Northeastern University
College of Professional Studies

PREDICTIVE ANALYTICS

SEC 01

ALY 6020, FALL 2022

<u>WEEK 2</u>: <u>MODULE 2 MIDWEEK PROJECT 2</u>

SUBMITTED BY: Akash Raj, Ananya Sharma, Vaibhav Arora, Vaibhav Jain

CRN: 70916

SUBMITTED TO: Dr. Mary Donhoffner

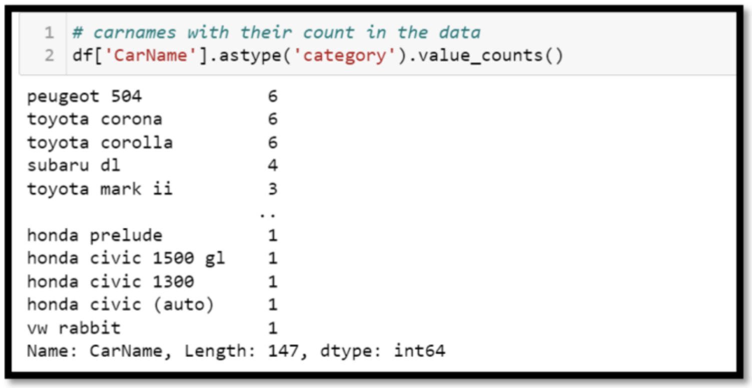
DATE: October 03, 2022

INTRODUCTION

The car price data set is being explored to understand the concepts of Regression Model. The data set has 205 rows and 26 columns. After checking the dimensions of the data set, we want the data set to be prepared for a Linear Regression Model.

The *info()* function gives us the non-Null values and the various data types of the columns. After this the *describe ()* function summarizes the statistical analysis of the dataset. It reveals that the min price of the cars starts from $5K and the max range goes to $45K approximately. Price of the car is our target variable and columns like car length, wheelbase are the attributes that can help to decide the price of the vehicle and strengthen the concept of Linear Regression.

Furthermore, we try to remove any null values by using the is null () function and trim our data set of any duplicate values by using the drop duplicates () function.



```
# carnames with their count in the data
df['CarName'].astype('category').value_counts()
```

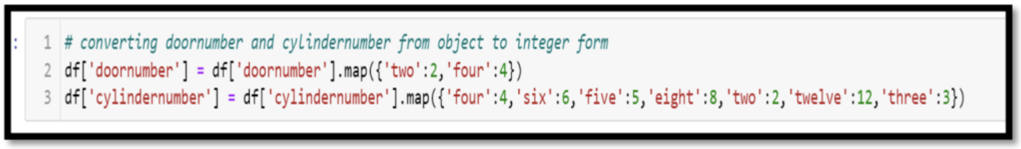
```
peugeot 504            6
toyota corona          6
toyota corolla         6
subaru dl              4
toyota mark ii         3
                      ..
honda prelude          1
honda civic 1500 gl    1
honda civic 1300       1
honda civic (auto)     1
vw rabbit              1
Name: CarName, Length: 147, dtype: int64
```

Fig1: Car Name and Value Counts ()

The Value_Counts() function helps to understand the distribution of the car as per their name. Similarly, when the function is applied to the Brand of the car it tells which car is more popular choice amongst the car buyers.

As we explore the count of the variables, we understand how the data is distributed. For example, the number of cars that have four doors are almost 115 versus two door cars that are 90 in number.

Here we notice that the number four and two are of type object and run codes to convert them into integer data type. Likewise we follow suit for the cylinder column.



```
# converting doornumber and cylindernumber from object to integer form
df['doornumber'] = df['doornumber'].map({'two':2,'four':4})
df['cylindernumber'] = df['cylindernumber'].map({'four':4,'six':6,'five':5,'eight':8,'two':2,'twelve':12,'three':3})
```

Fig 2: Categorical to Integer form

EXPLORATORY DATA ANALYSIS

Before we start EDA, there are certain conditions for Linear Regression that we must be met with:

1) The predictor and the indicator variables must portray a linear relationship
2) The dependent variable must be numeric and continuous
3) Multi-collinearity must be avoided amongst the variables.
4) The independent variable must not exhibit high variance values along the regression line.

Upon building the model further onwards we keep a check on these assumptions to check the veracity of the points.

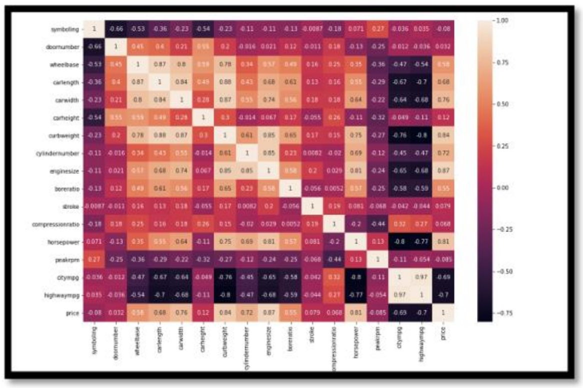


Fig 3: Correlation Matrix

Correlation helps to understand the +ve and the -ve correlation amongst the variables. The figures seem to be quite independent and therefore affirm that the data set abides by the concept of no multicollinearity.

Additionally, we drop a few rows like stroke, compression ratio and peak rm since they show very correlation and won't help further in our analysis of linear regression model. The figure above shows a lot of variables to consider while studying and confuses us to a certain extent. We remove a few variables from here so that we have only the relevant predictor variables for analysis.

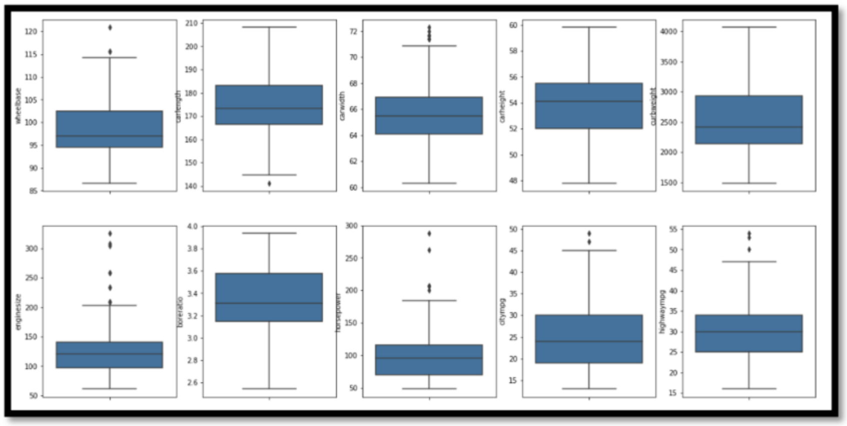


Fig 4: Box Plot

We use the sns seaborn library and matplotlib library to plot these Boxplots graphs. The above figure shows that the engine size and the car width show some outliers and must be cleaned.

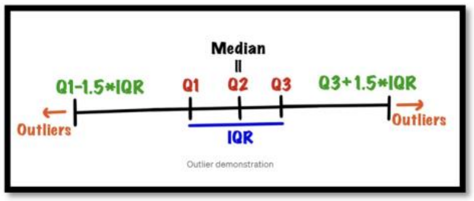


Fig 5: Interquartile method of removing Outliers, Source: (Dino, 2022)

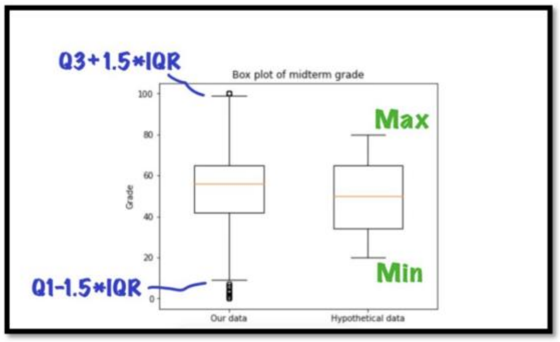


Fig 6: Interquartile Range

No better words than pictures trying to communicate as to how to remove the outliers and the figures above explain exactly how the upper 75th Percentile is subtracted from the lower 25th Percentile to remove the outliers and give a clean data for study.

PREDICTIVE MODELLING

In this section if you will create linear regression model and follow the steps given below:

1. Train dash test data split
2. Future scaling / normalization
3. Building linear regression model
4. Check for linearity and residual normality
5. Evaluate multivariate normality and QQ plot
6. Evaluate the model and interpret results

Part 1: Training and Testing Split

Now that we have cleaned the data and conducted exploratory data analysis, the first step to creating a linear model is to split the data set into training and testing sets. The data set is randomly divided into 80/20 ratio, where 80 is the training set in 20% of the data is the test set.

Part 2: Feature normalization

Most effective machine learning models are when features are scaled to normal form. Here we will use a standard scalar from sklearn library's preprocessing module. Using the standardization method, we convert all the variable's values into a range of -1 to 1. This is done to remove any bias from the variable values.

Part 3: Building a Linear Regression

To build a linear regression model, we import the linear regression class from the linear model module of sklearn. First, we fit the model on the training data set. The shape of the training data set is 155 records of 45 variables. Now we run the model on the test data set, and we get r-squared score (coefficient of determination)of 0.7819.

Now we check the model for linearity and residual normality that is residual error.

In the first graph, we see that the predicted and actual values are directly proportional to each other. This shows strong linearity. We can also notice an outlier in the graph. For residual error, the majority must be centered around the zero value. We can see that the dashed value is slightly off the mark and that is attributed to an outlier which we can notice on the left-hand side of the x-axis.

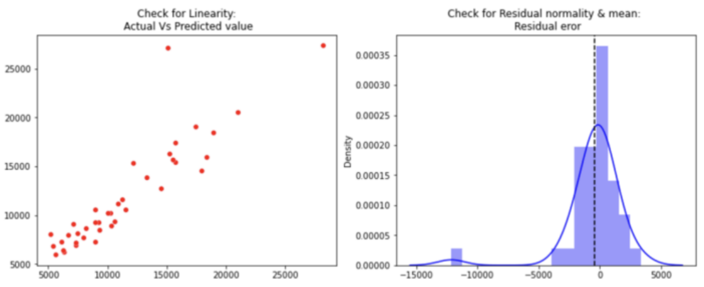


Fig 7: a) Linearity b) Residual Error

A normal QQ plot is used to depict the distribution of residuals. Usually, the residuals should be on the straight line of the QQ plot, and the points lying outside of the straight line are considered outliers. In this case, all the points are centered around the linear line except for one outlier.

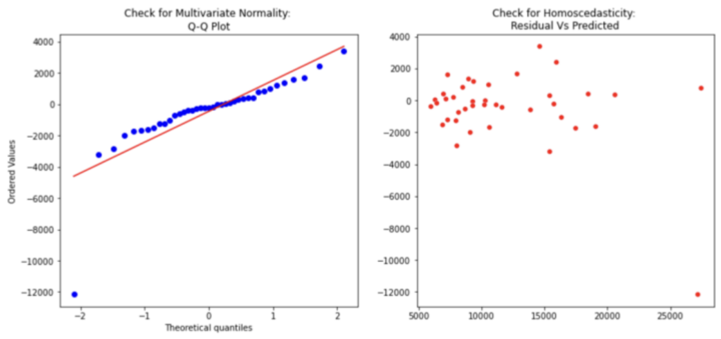


Fig 8: a) Q-Q Plot b) Homoscedasticity

Homoscedasticity refers to having the same scatter and a linear relationship between the values of X and y where X and y represent the actual and predicted values. Homeless candidacy is centered around the excess and the points are ideally placed near the line of the axis.

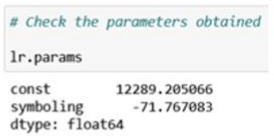
PREDICTION AND MODEL EVALUATION

Model evaluation involves investigating a machine learning model's performance and strengths and weaknesses using various evaluation metrics. Model evaluation is necessary to assess the efficacy of a model during the early stages of research, and it also plays a role in model monitoring. We are using Ordinary Least Square method to carry out our model evaluation.

Linear Regression (Ordinary Least Square) Statsmodels API

The linear regression class from Sklearn does not provide attributes for obtaining the model summary, variable coefficients, and VIF (variable inflation factor) for improving the model. Therefore, to obtain the model summary and coefficients, we need to use OLS class from the statsmodels API. The *OLS()* function creates an OLS object, which is then fit on the training data. A constant also need to be added before training the model. Once the model is trained, the summary statistics and variable coefficients can be obtained.

First, we create a model with just one variable to get an intuitive sense of the variable, and then keep on adding more variables. The linear regression was trained with the variable ‘symbolling’.



```
# Check the parameters obtained

lr.params

const        12289.205066
symboling      -71.767083
dtype: float64
```

Fig 9: Linear Regression parameters

The coefficient of the variable was -71, and the intercept was 12,289. The variable symbolling had a negative effect on price. As symbolling increases, the value of price goes down. Please refer below screenshot

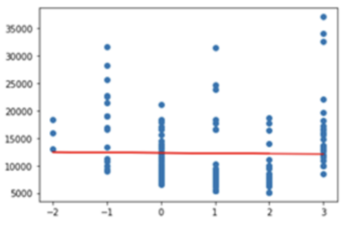
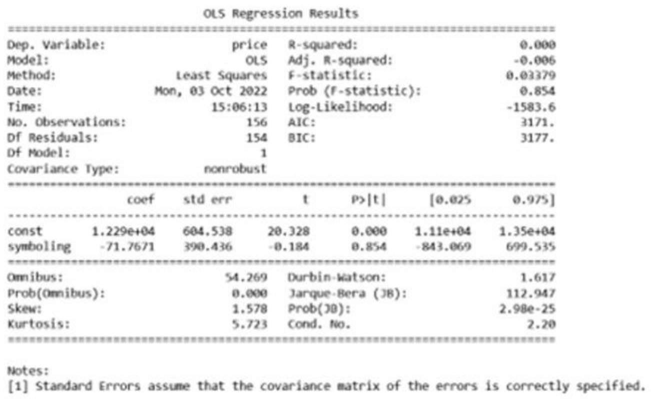


Fig 10: Viz using Fitted regression line

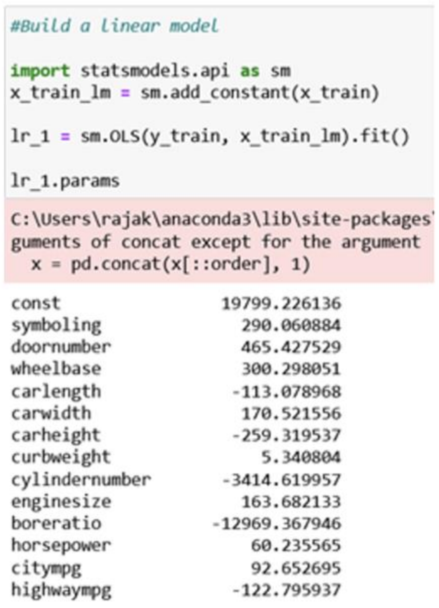
The following is the summary of the model created with just one variable. Refer Fig 11



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                 -0.006
Method:                 Least Squares   F-statistic:                   0.03379
Date:                Mon, 03 Oct 2022   Prob (F-statistic):              0.854
Time:                        15:06:13   Log-Likelihood:                -1583.6
No. Observations:                 156   AIC:                             3171.
Df Residuals:                     154   BIC:                             3177.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1.229e+04    604.538     20.328      0.000    1.11e+04    1.35e+04
symboling    -71.7671    390.436     -0.184      0.854    -843.069     699.535
==============================================================================
Omnibus:                       54.269   Durbin-Watson:                   1.617
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              112.947
Skew:                           1.578   Prob(JB):                     2.98e-25
Kurtosis:                       5.723   Cond. No.                         2.20
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Fig 11: Linear regression Summary

Since, the model was created with just one variable, the model results are extremely poor with 0% R-squared. We will now add all the variables to model. The following code snippet and output indicates a model with all the variables, and the coefficients of all the variables.



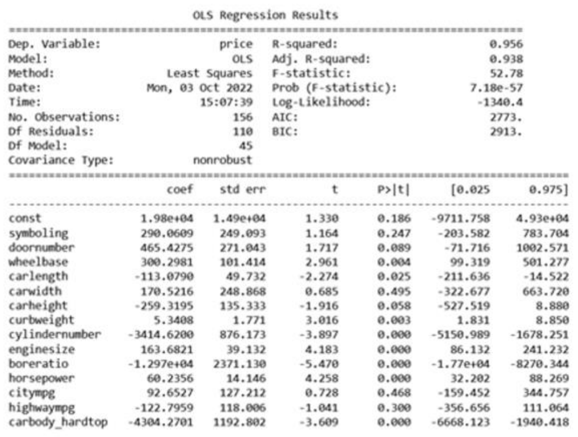
```
#Build a linear model

import statsmodels.api as sm
x_train_lm = sm.add_constant(x_train)

lr_1 = sm.OLS(y_train, x_train_lm).fit()

lr_1.params
```

```
C:\Users\rajak\anaconda3\lib\site-packages'
guments of concat except for the argument
  x = pd.concat(x[::order], 1)
```

```
const              19799.226136
symboling            290.060884
doornumber           465.427529
wheelbase            300.298051
carlength           -113.078968
carwidth             170.521556
carheight           -259.319537
curbweight             5.340804
cylindernumber     -3414.619957
enginesize           163.682133
boreratio         -12969.367946
horsepower            60.235565
citympg               92.652695
highwaympg          -122.795937
```

Fig 12: Linear Regression with all variables

Once, the model has been trained on all the training data and variables, the model summary can be obtained.

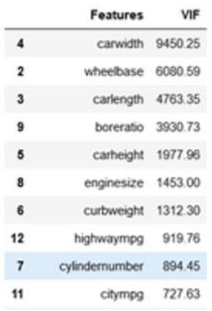


```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.956
Model:                            OLS   Adj. R-squared:                  0.938
Method:                 Least Squares   F-statistic:                     52.78
Date:                Mon, 03 Oct 2022   Prob (F-statistic):           7.18e-57
Time:                        15:07:39   Log-Likelihood:                -1340.4
No. Observations:                 156   AIC:                             2773.
Df Residuals:                     110   BIC:                             2913.
Df Model:                          45
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const               1.98e+04   1.49e+04      1.330      0.186   -9711.758    4.93e+04
symboling           290.0609    249.093      1.164      0.247    -203.582     783.704
doornumber          465.4275    271.043      1.717      0.089     -71.716    1002.571
wheelbase           300.2981    101.414      2.961      0.004      99.319     501.277
carlength          -113.0790     49.732     -2.274      0.025    -211.636     -14.522
carwidth            170.5216    248.868      0.685      0.495    -322.677     663.720
carheight          -259.3195    135.333     -1.916      0.058    -527.519       8.880
curbweight            5.3408      1.771      3.016      0.003       1.831       8.850
cylindernumber    -3414.6200    876.173     -3.897      0.000   -5150.989   -1678.251
enginesize          163.6821     39.132      4.183      0.000      86.132     241.232
boreratio         -1.297e+04   2371.130     -5.470      0.000   -1.77e+04   -8270.344
horsepower           60.2356     14.146      4.258      0.000      32.202      88.269
citympg              92.6527    127.212      0.728      0.468    -159.452     344.757
highwaympg         -122.7959    118.006     -1.041      0.300    -356.656     111.064
carbody_hardtop   -4304.2701   1192.802     -3.609      0.000   -6668.123   -1940.418
```

Fig 13: Linear Regression Summary with all Variables

The model has an **R-squared of 95.6%** and an **adjusted R-squared of 93.8%.** However, we need to obtain the VIF of the variables to check for multicollinearity.

The variance inflation factor can also be obtained from the statsmodels API using a function variance_inflation_factor(). Any variable with VIF greater than 10 should be removed from the model, as they are colinear with other variables.

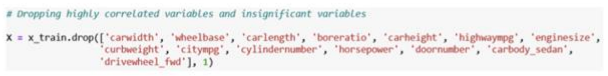


	Features	VIF
4	carwidth	9450.25
2	wheelbase	6080.59
3	carlength	4763.35
9	boreratio	3930.73
5	carheight	1977.96
8	enginesize	1453.00
6	curbweight	1312.30
12	highwaympg	919.76
7	cylindernumber	894.45
11	citympg	727.63

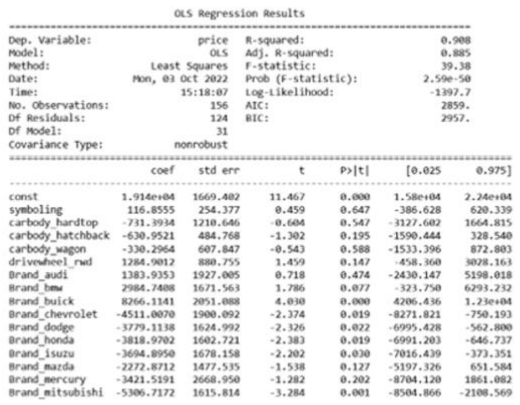
Fig 14: Variables VIF

Many variables like ‘carwidth’, ‘wheelbase’, ‘carlength’, ‘boreratio’ etc. are highly colinear with other variables. Therefore, these variables need to be removed from the model.

The final model is created after iteratively removing all the multicollinear variables.



```
# Dropping highly correlated variables and insignificant variables

X = x_train.drop(['carwidth', 'wheelbase', 'carlength', 'boreratio', 'carheight', 'highwaympg', 'enginesize',
                  'curbweight', 'citympg', 'cylindernumber', 'horsepower', 'doornumber', 'carbody_sedan',
                  'drivewheel_fwd'], 1)
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.908
Model:                            OLS   Adj. R-squared:                  0.885
Method:                 Least Squares   F-statistic:                     39.38
Date:                Mon, 03 Oct 2022   Prob (F-statistic):           2.59e-50
Time:                        15:18:07   Log-Likelihood:                -1397.7
No. Observations:                 156   AIC:                             2859.
Df Residuals:                     124   BIC:                             2957.
Df Model:                          31
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const              1.914e+04   1669.402     11.467      0.000    1.58e+04    2.24e+04
symboling           116.8555    254.377      0.459      0.647    -386.628     620.339
carbody_hardtop    -731.3934   1210.646     -0.604      0.547   -3127.602    1664.815
carbody_hatchback  -630.9521    484.768     -1.302      0.195   -1590.444     328.540
carbody_wagon      -330.2964    607.847     -0.543      0.588   -1533.396     872.803
drivewheel_rwd     1284.9012    880.755      1.459      0.147    -458.360    3028.163
Brand_audi         1383.9353   1927.005      0.718      0.474   -2430.147    5198.018
Brand_bmw          2984.7408   1671.563      1.786      0.077    -323.750    6293.232
Brand_buick        8266.1141   2051.088      4.030      0.000    4206.436    1.23e+04
Brand_chevrolet   -4511.0070   1900.092     -2.374      0.019   -8271.821    -750.193
Brand_dodge       -3779.1138   1624.992     -2.326      0.022   -6995.428    -562.800
Brand_honda       -3818.9702   1602.721     -2.383      0.019   -6991.203    -646.737
Brand_isuzu       -3694.8950   1678.158     -2.202      0.030   -7016.439    -373.351
Brand_mazda       -2272.8712   1477.535     -1.538      0.127   -5197.326     651.584
Brand_mercury     -3421.5191   2668.950     -1.282      0.202   -8704.120    1861.082
Brand_mitsubishi  -5306.7172   1615.814     -3.284      0.001   -8504.866   -2108.569
```

Fig 15: Summary after removing Multicollinear Variables

The final model has an R-Squared of 90.8% and an adjusted R-squared of 88.5%. The AIC value of the model is 2,859.

<u>Questions</u>

1. What are the three most significant variables?

Ans: We performed feature engineering to extract the brand name of the cars, which were then created into dummy variables. Some of the brand variables came out to be significant. The top significant variables are ‘hptype’, ‘curbtype’, ‘brand_buick’, and ‘brand_porsche’.

2. Of those three, which had the greatest positive influence on car prices?

Ans: Out of the significant variables, the brand variables had a positive influence on car prices, indicating that car prices increase when the cars are of these brands (Buick and Porsche). These also make sense as the cars of these brands are luxury cars.

3. How accurate was the model?

Ans: The final model has an R-Squared of 90.8% and an adjusted R-squared of 88.5%. The AIC value of the model is 2,859.

REFERENCES

[1.] Mahmood, M. S. (2020, December 26). Practical Implementation of Outlier Detection. Retrieved October 2, 2022, from *https://towardsdatascience.com/practicalimplementation-of-outlier-detection-in-python*

[2.] Dino, L. (2022, June 14). *Outlier detection using IQR method and Box plot in Python*. Medium. Retrieved October 2, 2022, from https://towardsdev.com/outlier-detection-using-iqr-method-and-box-plot-in-python-82e1e15232bdVaibhavV

[3.] Darekar, A. (2022, January 7). *Linear Regression A-Z (Using Car Price Prediction dataset)*. Medium. Retrieved October 2, 2022, from https://medium.com/analytics-vidhya/linear-regression-a-z-using-car-price-prediction-dataset-e32b50c7a561

APPENDIX

Code:
Importing the libraries

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns


df = pd.read_csv('CarPrice_Assignment.csv')


df.info()


df.describe()


df.isnull().sum()


df.shape
```

Data Cleaning

```
# dropping duplicate rows if any
df.drop_duplicates()


# carnames with their count in the data
df['CarName'].astype('category').value_counts()


# As we can see that CarName column contains the car company name and the model name together.
# So we can extract the name of Car Company from CarName column and put it in some other column.


def split(x):
   return x.split(' ')[0]

df['Brand'] = df['CarName'].apply(split)
df.head()


# count of car brands in the data
df['Brand'].astype('category').value_counts()


# We can see that there are some mistakes in the data.There are some repetitions in company names
# because of the writing mistake as mentioned below:
# toyota : toyouta
# Nissan : nissan
# porsche : porcsche
# mazda : maxda
# volkswagen : vw, vokswagen


# Converting all the Brand names to lowercase alphabets
df['Brand'] = df['Brand'].apply(lambda y: y.lower())

# corecting the wrongly marked data in the dataset
df.loc[(df['Brand'] == 'vw') | (df['Brand'] == 'vokswagen'),'Brand'] = 'volkswagen'
df.loc[(df['Brand'] == 'toyouta'),'Brand'] = 'toyota'
df.loc[(df['Brand'] == 'maxda'),'Brand'] = 'mazda'
df.loc[(df['Brand'] == 'porcshce'),'Brand'] = 'porsche'


# data in brand column after cleaning
df['Brand'].astype('category').value_counts()


# count of column cylinder number
df['cylindernumber'].astype('category').value_counts()


# count of column doornumber
df['doornumber'].astype('category').value_counts()


# converting doornumber and cylindernumber from object to integer form
df['doornumber'] = df['doornumber'].map({'two':2,'four':4})
df['cylindernumber'] = df['cylindernumber'].map({'four':4,'six':6,'five':5,'eight':8,'two':2,'twelve':12,'three':3})


# after conversion
df['cylindernumber'].astype('category').value_counts()


# after conversion
df['doornumber'].astype('category').value_counts()


df['enginetype'].astype('category').value_counts()


# count of column fuelsystem
df['fuelsystem'].astype('category').value_counts()


# count of column fueltype
df['fueltype'].astype('category').value_counts()


# count of column aspiration
df['aspiration'].astype('category').value_counts()


# count of column carbody
df['carbody'].astype('category').value_counts()
```

```
# count of column drivewheel
df['drivewheel'].astype('category').value_counts()


# count of column enginelocation
df['enginelocation'].astype('category').value_counts()


# count of column symboling
df['symboling'].astype('category').value_counts()


# Now we need to drop the columns which are not suitable for predictions and training data
# car_ID is an identity number given to every car and it does not have any affect on price.
# CarName needed to be dropped as we have already extracted the company name (Brand column) from the
column.
# Columns like enginetype, fuelsystem, enginelocation, aspiration, and fueltype are highly imbalanced classes
#or attributes. So we need to drop them as well.


# dropping all the required columns
df.drop(columns=['CarName','car_ID','enginetype','fuelsystem','enginelocation','aspiration','fueltype'],axis=1,in
place=True)
```

Exploratory Data Analysis

```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),annot=True)
plt.show()


# Dropping columns again according to their correlation
# Columns stroke, compressionratio, and peakrpm needs to be dropped because they don't have much impact
#or a good correlation with any other variables.


# dropping columns with low correlation with target column
df.drop(columns=['stroke','compressionratio','peakrpm'],axis=1,inplace=True)


# Removing outliers from the data because it can affect the model during prediction and also affects the
#LinearRegression line while making the model


# plotting boxplot to identify outliers
plt.figure(figsize=(20,10))
plt.subplot(2,5,1)
sns.boxplot(y='wheelbase',data=df)
plt.subplot(2,5,2)
sns.boxplot(y='carlength',data=df)
plt.subplot(2,5,3)
sns.boxplot(y='carwidth',data=df)
plt.subplot(2,5,4)
sns.boxplot(y='carheight',data=df)
plt.subplot(2,5,5)
sns.boxplot(y='curbweight',data=df)
```

```
plt.subplot(2,5,6)
sns.boxplot(y='enginesize',data=df)
plt.subplot(2,5,7)
sns.boxplot(y='boreratio',data=df)
plt.subplot(2,5,8)
sns.boxplot(y='horsepower',data=df)
plt.subplot(2,5,9)
sns.boxplot(y='citympg',data=df)
plt.subplot(2,5,10)
sns.boxplot(y='highwaympg',data=df)
plt.show()


# From the above graphs, we can see that the column enginesize have a good amount of outliers which
#needs to be cleaned or removed for the further predictive analysis


# 25th and 75th percentile of the values in enginesize column
Q1_es = df.enginesize.quantile(0.25)
Q3_es = df.enginesize.quantile(0.75)

# onter quartile range of the values
IQR_es = Q3_es - Q1_es

# upper limit : The values greater than this are outliers
upper_es = Q3_es + 1.5*IQR_es

# removing rows with outliers from dataframe
df = df[(df.enginesize < upper_es)]


# As enginesize, horsepower, and curbweight are highly correlated with price, we can categorize them into
three parts.
# curbweight : light, moderate and, heavy
# enginesize : small, medium, and large
# horsepower : low, medium, and high


# quantiles for all the columns
Q1_hp = df.horsepower.quantile(0.25)
Q3_hp = df.horsepower.quantile(0.75)
Q1_cw = df.curbweight.quantile(0.25)
Q3_cw = df.curbweight.quantile(0.75)

# function definitions
def curb_convert(x):
    if x <= Q1_cw:
        return 'light'
    elif x < Q3_cw and x > Q1_cw:
        return 'moderate'
    else:
        return 'heavy'

def es_convert(x):
    if x <= Q1_es:
        return 'small'
    elif x < Q3_es and x > Q1_es:
        return 'medium'
    else:
        return 'large'

def hp_convert(x):
    if x <= Q1_hp:
        return 'low'
    elif x < Q3_hp and x > Q1_hp:
        return 'medium'
    else:
        return 'high'

# calling the functions
df['curbtype'] = df['curbweight'].apply(curb_convert)
df['engsize'] = df['enginesize'].apply(es_convert)
df['hptype'] = df['horsepower'].apply(hp_convert)


# all categorical columns or the columns with object dtype
categorical = df.select_dtypes(include=['object'])
categorical.head()


# Creating categorical columns vs price box plots to understand data


plt.figure(figsize=(20,8))
plt.subplot(2,4,1)
sns.boxplot(x='symboling',y='price',data=df)
plt.subplot(2,4,2)
sns.boxplot(x='cylindernumber',y='price',data=df)
plt.subplot(2,4,3)
sns.boxplot(x='doornumber',y='price',data=df)
plt.subplot(2,4,4)
sns.boxplot(x='carbody',y='price',data=df)
plt.subplot(2,4,5)
sns.boxplot(x='drivewheel',y='price',data=df)
plt.subplot(2,4,6)
sns.boxplot(x='curbtype',y='price',data=df)
plt.subplot(2,4,7)
sns.boxplot(x='engsize',y='price',data=df)
plt.subplot(2,4,8)
sns.boxplot(x='hptype',y='price',data=df)


plt.figure(figsize=(20,6))
sns.boxplot(x='Brand',y='price',data=df)


# Creating dummies of all the categorical columns left in the dataset
# converting to dummies
dummies = pd.get_dummies(categorical,drop_first=True)
dummies.head()
```

```
# concat the dummies dataframe to the main dataframe
df = pd.concat([df,dummies],axis=1)

# dropping all the categorical columns as the dummies are already created
df.drop(columns=categorical.columns,axis=1,inplace=True)
```

Modeling

```
y = df['price']
x = df.drop('price',1)


y.head()


# splitting the main data into train and test data
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y,
                                        test_size= 0.2,
                                        shuffle= True, #shuffle the data to avoid bias
                                        random_state= 0)
x_train= np.asarray(x_train)
y_train= np.asarray(y_train)

x_test= np.asarray(x_test)
y_test= np.asarray(y_test)


# Scaling
# Now we need to scale the variables for better interpretability.
# As all the final independent variables in our dataset are numeric, we'll scale all the variables
# Let's scale all these columns using StandardScaler. You can use any other scaling method as well; it is
totally up
# to you.


from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test= sc.transform(x_test)
# from sklearn.preprocessing import StandardScaler
# # scaler object
# scaler = StandardScaler()
# x_train[x_train.columns] = scaler.fit_transform(x_train)


# Building the first Linear Regression model with all the features
# Equation of linear regression
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(x_train,y_train)


lm.score(x_train,y_train)
```

```
x_train.shape


y_train.shape


y_pred = lm.predict(x_test)


from sklearn.metrics import r2_score
r2_score(y_test, y_pred)


# Check for Linearity
f = plt.figure(figsize=(14,5))
ax = f.add_subplot(121)
sns.scatterplot(y_test,y_pred,ax=ax,color='r')
ax.set_title('Check for Linearity:\n Actual Vs Predicted value')

# Check for Residual normality & mean
ax = f.add_subplot(122)
sns.distplot((y_test - y_pred),ax=ax,color='b')
ax.axvline((y_test - y_pred).mean(),color='k',linestyle='--')
ax.set_title('Check for Residual normality & mean: \n Residual eror');


# Check for Multivariate Normality
# Quantile-Quantile plot
f,ax = plt.subplots(1,2,figsize=(14,6))
import scipy as sp
_,(_,_,r)= sp.stats.probplot((y_test - y_pred),fit=True,plot=ax[0])
ax[0].set_title('Check for Multivariate Normality: \nQ-Q Plot')

#Check for Homoscedasticity
sns.scatterplot(y = (y_test - y_pred), x= y_pred, ax = ax[1],color='r')
ax[1].set_title('Check for Homoscedasticity: \nResidual Vs Predicted');
```

Linear Regression from statsmodel

```
# splitting the main data into train and test data
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y,
                                    test_size= 0.2,
                                    shuffle= True, #shuffle the data to avoid bias
                                    random_state= 0)


x_train.head()


import statsmodels.api as sm

# Add a constant
```

```
x_train_lm = sm.add_constant(x_train[['symboling']])

# Create a first fitted model
lr = sm.OLS(y_train, x_train_lm).fit()


# Check the parameters obtained

lr.params


# Let's visualise the data with a scatter plot and the fitted regression line
plt.scatter(x_train_lm.iloc[:, 1], y_train)
plt.plot(x_train_lm.iloc[:, 1], 12289.205066 - 71.767083*x_train_lm.iloc[:, 1], 'r')
plt.show()


# Print a summary of the linear regression model obtained
print(lr.summary())
```

Adding all the variables to the model

```
#Build a linear model

import statsmodels.api as sm
x_train_lm = sm.add_constant(x_train)

lr_1 = sm.OLS(y_train, x_train_lm).fit()

lr_1.params


print(lr_1.summary())
```

Checking VIF

```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor


# Create a dataframe that will contain the names of all the feature variables and their respective VIFs
vif = pd.DataFrame()
vif['Features'] = x_train.columns
vif['VIF'] = [variance_inflation_factor(x_train.values, i) for i in range(x_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Dropping the variable and updating the model

```
# Dropping highly correlated variables and insignificant variables
```

```
X = x_train.drop(['carwidth', 'wheelbase', 'carlength', 'boreratio', 'carheight', 'highwaympg', 'enginesize',
                'curbweight', 'citympg', 'cylindernumber', 'horsepower', 'doornumber', 'carbody_sedan',
                'drivewheel_fwd'], 1)


# Build a third fitted model
x_train_lm = sm.add_constant(X)

lr_2 = sm.OLS(y_train, x_train_lm).fit()


# Print the summary of the model
print(lr_2.summary())


# Calculate the VIFs again for the new model

vif = pd.DataFrame()
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```