Predictive Analytics

ALY 6020, CRN 80405

Professor Vladimir Shapiro

Module 2: Midweek Project - Car Price Prediction

Submitted By - Richa Umesh Rambhia

Implementation of Linear Regression - Car Price Analysis

Table of Contents

Introduction

Linear Regression Algorithm

Machine Learning algorithms are classified into *supervised and unsupervised* learning, where supervised learning algorithms are further classified into **Classification & Regression** based problems. Classification problems deal with *categorical data* in order to **classify the classes** for the data points, whereas Regression problems are **prediction** based models that are *continuous* in nature and predict the output variable depending on the features of the data. [2]

Simple Linear Regression Model Linear regression models are simple methods that are used for the predictive analysis that show the linear relationship between the independent variable of the dataset and the target variable.

Multiple Linear Regression Model Simple linear regression models are used when there is only 1 independent variable to predict the target variable. However, multiple linear regression model is used when there are multiple independent variables in order to predict a single dependent variable. [2]

Car Price Prediction

The car dataset contains multiple features for the price prediction such as *car name, car body, engine name, fuel type, car width, car height, engine location,* and many more than help in predicting the target variable which is the price column in this dataset. *The data dictionary shown below in Figure 1. helps to understand each of the parameters of the dataset.*

In the following project, the **price of the car is predicted** based on the different parameters or features using the **Linear Regression algorithm**, and since there are *multiple independent variables* in order to predict a single dependent variable, **multiple linear regression model** is implemented.

Analysis

Given a dataset and a data dictionary, you will perform a linear regression analysis that predicts the price of cars.

Installing required packages

In [2]:
```
!pip install pandas_profiling
!pip install featurewiz
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-w
heels/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Collecting pandas_profiling
  Downloading pandas_profiling-3.6.6-py2.py3-none-any.whl (324 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 324.4/324.4 kB 7.9 MB/s eta 0:00:00
Collecting ydata-profiling
  Downloading ydata_profiling-4.1.2-py2.py3-none-any.whl (345 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 345.9/345.9 kB 7.6 MB/s eta 0:00:00
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.9/dist-packages (from ydata-
profiling->pandas_profiling) (3.1.2)
Collecting imagehash==4.3.1
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 296.5/296.5 kB 15.3 MB/s eta 0:00:00
Collecting visions[type_image_path]==0.7.5
  Downloading visions-0.7.5-py3-none-any.whl (102 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 102.7/102.7 kB 5.4 MB/s eta 0:00:00
Requirement already satisfied: requests<2.29,>=2.24.0 in /usr/local/lib/python3.9/dist-packages (from yda
ta-profiling->pandas_profiling) (2.27.1)
Collecting typeguard<2.14,>=2.13.2
```

Importing libraries

In [4]:
```
import pandas as pd
import numpy as np
import pandas_profiling
import ydata_profiling
import matplotlib.pyplot as plt
import seaborn as sns
from featurewiz import featurewiz
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from sklearn import metrics
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
Imported version = 0.1.55.
from featurewiz import FeatureWiz
wiz = FeatureWiz(verbose=1)
X_train_selected = wiz.fit_transform(X_train, y_train)
X_test_selected = wiz.transform(X_test)
wiz.features  ### provides a list of selected features ###
```

Loading the dataset

In [5]:
```
car_data = pd.read_csv("CarPrice.csv")
car_data
```

Out[5]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	13
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	13
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	15
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	10
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	13
...	...	...	...	...	...	...	...	...	...	...	...	..
200	201	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	front	109.1	...	14
201	202	-1	volvo 144ea	gas	turbo	four	sedan	rwd	front	109.1	...	14
202	203	-1	volvo 244dl	gas	std	four	sedan	rwd	front	109.1	...	17
203	204	-1	volvo 246	diesel	turbo	four	sedan	rwd	front	109.1	...	14
204	205	-1	volvo 264gl	gas	turbo	four	sedan	rwd	front	109.1	...	14

205 rows × 26 columns

Table 1. Car Price Data

Data Dictionary

(To understand each of the parameters of the dataset)

1	**Car_ID**	Unique id of each observation (Interger)
2	**Symboling**	Its assigned insurance risk rating, A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.(Categorical)
3	**carCompany**	Name of car company (Categorical)
4	**fueltype**	Car fuel type i.e gas or diesel (Categorical)
5	**aspiration**	Aspiration used in a car (Categorical)
6	**doornumber**	Number of doors in a car (Categorical)
7	**carbody**	body of car (Categorical)
8	**drivewheel**	type of drive wheel (Categorical)
9	**enginelocation**	Location of car engine (Categorical)
10	**wheelbase**	Weelbase of car (Numeric)
11	**carlength**	Length of car (Numeric)
12	**carwidth**	Width of car (Numeric)
13	**carheight**	height of car (Numeric)
14	**curbweight**	The weight of a car without occupants or baggage. (Numeric)
15	**enginetype**	Type of engine. (Categorical)
16	**cylindernumber**	cylinder placed in the car (Categorical)
17	**enginesize**	Size of car (Numeric)
18	**fuelsystem**	Fuel system of car (Categorical)
19	**boreratio**	Boreratio of car (Numeric)
20	**stroke**	Stroke or volume inside the engine (Numeric)
21	**compressionratio**	compression ratio of car (Numeric)
22	**horsepower**	Horsepower (Numeric)
23	**peakrpm**	car peak rpm (Numeric)
24	**citympg**	Mileage in city (Numeric)
25	**highwaympg**	Mileage on highway (Numeric)
26	**price(Dependent variable)**	Price of car (Numeric)

Figure 1. Data Dictionary for Car Price Analysis [1]

Exploratory Data Analysis

EDA is performed on the data in order to analyze various parameters and features of the dataset and to understand the *structure* of the dataset such that various *trends and patterns* between the variables is known. Exploratory Data Analysis helps in understanding the *relationship between the various independent and dependent variables* of the dataset that would further be useful in building the model such as description analysis and statistical analysis.

Descriptive Analysis

In [4]:

```
# displaying number of rows and columns
print("Total number of Rows and Columns:", car_data.shape)

print("\n-------------------------------------------------------")

# displaying field values/column names
print("\nColumn Names:\n")
car_data.columns
```

```
Total number of Rows and Columns: (205, 26)

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

Column Names:
```

Out[4]:

```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

In [6]:
```
# displaying data types
print("Data types:\n")
car_data.dtypes
```

```
Data types:
```

Out[6]:
```
car_ID                int64
symboling             int64
CarName              object
fueltype             object
aspiration           object
doornumber           object
carbody              object
drivewheel           object
enginelocation       object
wheelbase           float64
carlength           float64
carwidth            float64
carheight           float64
curbweight            int64
enginetype           object
cylindernumber       object
enginesize            int64
fuelsystem           object
boreratio           float64
stroke              float64
compressionratio    float64
horsepower            int64
peakrpm               int64
citympg               int64
highwaympg            int64
price               float64
dtype: object
```

From the *descriptive analysis*, it is observed that there are total **205 rows of data** and **26 field values** and the data type for each of the field value is displayed in order to understand what data type values are present in the dataset.

Here, there are different types of data points that are present in the dataset which are **numerical data type** having *'int' and 'float'* values and remaining field values are of **object type**, which needs to be updated to **category** type later in the preprocessing stage.

Statistical Analysis

In [7]:
```
# dataset info
print("Dataset Info:\n")
car_data.info()
```

```
Dataset 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
```

Table 2. Information about the dataset

In [8]:
```
# describing the dataset
print("Describing the dataset:\n")
round(car_data.describe(),1)
```

```
Describing the dataset:
```

Out[8]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	hors
count	205.0	205.0	205.0	205.0	205.0	205.0	205.0	205.0	205.0	205.0	205.0	
mean	103.0	0.8	98.8	174.0	65.9	53.7	2555.6	126.9	3.3	3.3	10.1	
std	59.3	1.2	6.0	12.3	2.1	2.4	520.7	41.6	0.3	0.3	4.0	
min	1.0	-2.0	86.6	141.1	60.3	47.8	1488.0	61.0	2.5	2.1	7.0	
25%	52.0	0.0	94.5	166.3	64.1	52.0	2145.0	97.0	3.2	3.1	8.6	
50%	103.0	1.0	97.0	173.2	65.5	54.1	2414.0	120.0	3.3	3.3	9.0	
75%	154.0	2.0	102.4	183.1	66.9	55.5	2935.0	141.0	3.6	3.4	9.4	
max	205.0	3.0	120.9	208.1	72.3	59.8	4066.0	326.0	3.9	4.2	23.0	

Table 3. Dataset Description

Statistical Analysis helps in understanding about each of the numerical field type based on the **total count values, minimum value, maximum value, standard deviation**, etc. which gives an overall analysis of the field data about the various rows of data present in the dataset.

For example, as observed in the car price dataset, we see that there are multiple field values having the *minimum, maximum values* along with the *total count of values* which is **205** and *standard deviation* of the column values. It can be observed that the maximum value of *enginesize* is **326** whereas the maximum value of *price* is **45400**.

Thus, similarly, other parameters of the dataset can be analyzed based on their statistical values.

Data Profiling

In [6]:
```
car_data_report = car_data.profile_report(title='Car Price Analysis Report', explorative = True)
car_data_report
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Out[6]:

In [7]:
```
# Saving the profile report
car_data_report.to_file(output_file="Car Price Analysis Report.html")
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

The data profiling report generated for the dataset helps in understanding various parameters such as the data type of the field values, the missing and duplicate values present in the dataset, the correlation between each of the field value, and the analysis of each of the field value on a individual basis based on correlation plot, histogram, and interaction graphs.

From the profiling report, it is observed that there are **16 numerical variable type and 10 categorical data type** of field values present in the dataset of which the numerical data type have **integer and float values**. Also, there are **no missing values or duplicate values** present in the dataset, and the missing values visualization or plot also helps in understanding that there are no missing values present in the dataset, and for each field value a separate visualization is displayed in order to specifically analyze a particular field value.

Data Cleaning

1. Checking for null values in each column of the dataset, i.e., missing or bad values
2. Checking for data types & correcting the data type for the variables
3. Renaming the field values of the dataset
4. Checking for outliers in the dataset

   a. *Boxplot*

   b. *Distribution Plot*

1. Checking for null values in each column of the dataset, i.e., missing or bad values

In [12]:
```
for x in range(25):
    print("%-45s %10d" % (car_data.columns.values[x], car_data.iloc[:,x].isna().sum()))
```

```
car_ID                                                 0
symboling                                              0
CarName                                                0
fueltype                                               0
aspiration                                             0
doornumber                                             0
carbody                                                0
drivewheel                                             0
enginelocation                                         0
wheelbase                                              0
carlength                                              0
carwidth                                               0
carheight                                              0
curbweight                                             0
enginetype                                             0
cylindernumber                                         0
enginesize                                             0
fuelsystem                                             0
boreratio                                              0
stroke                                                 0
compressionratio                                       0
horsepower                                             0
peakrpm                                                0
citympg                                                0
highwaympg                                             0
```

Table 4. Missing Values Count

The code above shows that there are **no missing values** present in the dataset. The **isna()** function is used in order to display and check the 'Null' or 'NA' values that are present in each of the field values of the dataset.

In [13]:
```
# displaying the starting rows of the dataset

print("Displaying the first 10 rows of data")
car_data.head()
```

```
Displaying the first 10 rows of data
```

Out[13]:

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

5 rows × 26 columns

Table 5. First 10 rows of the dataset

In [14]:
```
# displaying the end rows of the dataset

print("Displaying the last 10 rows of data")
car_data.tail()
```

```
Displaying the last 10 rows of data
```

Out[14]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fu
200	201	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	front	109.1	...	141	
201	202	-1	volvo 144ea	gas	turbo	four	sedan	rwd	front	109.1	...	141	
202	203	-1	volvo 244dl	gas	std	four	sedan	rwd	front	109.1	...	173	
203	204	-1	volvo 246	diesel	turbo	four	sedan	rwd	front	109.1	...	145	
204	205	-1	volvo 264gl	gas	turbo	four	sedan	rwd	front	109.1	...	141	

5 rows × 26 columns

Table 6. Last 10 rows of the dataset

2. Checking for data types & correcting the data type for the variables

In [6]:
```
# correcting the data types for the variables of the dataset which are of object type to string/category type

car_data = car_data.astype({'CarName': 'category', 'fueltype': 'category', 'aspiration': 'category', 'doornum

print("Data Type conversion successful.")
```

```
Data Type conversion successful.
```

In [17]:
```
# checking for the correct data type of the variable
print("Data types:\n")
car_data.dtypes
```

```
Data types:
```

Out[17]:
```
car_ID                 int64
symboling              int64
CarName             category
fueltype            category
aspiration          category
doornumber          category
carbody             category
drivewheel          category
enginelocation      category
wheelbase            float64
carlength            float64
carwidth             float64
carheight            float64
curbweight             int64
enginetype          category
cylindernumber      category
enginesize             int64
fuelsystem          category
boreratio            float64
stroke               float64
compressionratio     float64
horsepower             int64
peakrpm                int64
citympg                int64
highwaympg             int64
price                float64
dtype: object
```

Here, the data type having object type data is converted to an appropriate data type, i.e., category data type, whereas the field values with integer and float data type are kept the same as in the dataset which represents their correct data type.

3. Renaming the field values of the dataset

In [22]:
```
car_data.columns
```

Out[22]:
```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

In [7]:
```
column_names = {i: i.capitalize() for i in car_data.columns}
car_data = car_data.rename(columns=column_names)

print("Renaming successful.")
```

```
Renaming successful.
```

In [85]:
```
# column names after renaming

car_data.columns
```

Out[85]:
```
Index(['Car_id', 'Symboling', 'Carname', 'Fueltype', 'Aspiration',
       'Doornumber', 'Carbody', 'Drivewheel', 'Enginelocation', 'Wheelbase',
       'Carlength', 'Carwidth', 'Carheight', 'Curbweight', 'Enginetype',
       'Cylindernumber', 'Enginesize', 'Fuelsystem', 'Boreratio', 'Stroke',
       'Compressionratio', 'Horsepower', 'Peakrpm', 'Citympg', 'Highwaympg',
       'Price'],
      dtype='object')
```

Since the column names are not correctly named, the above code shows how the columns have been renamed. Here, we have **captialized** the column names in order to be easily utilized further in the code.

4. Checking for outliers in the dataset

a. Boxplot

The below code creates **boxplots** for the various field values of the car dataset in order to check for outliers present in the dataset. Here, the boxplots are implemented for the variables **Carwidth**, **Carheight**, **Enginesize**, and **Price**, as shown in the below figures. The outliers that are present in the dataset will not be removed as each of the data point is important for analysis and model building.

In [31]:
```
# creating boxplot for 'Carwidth' and 'Carheight' variable

fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].boxplot(car_data['Carwidth'])
axs[1].boxplot(car_data['Carheight'])
axs[0].set_title('Boxplot for Carwidth')
axs[1].set_title('Boxplot for Carheight')
axs[0].set_ylabel('Data Values')
axs[1].set_ylabel('Data Values')

plt.show()
```

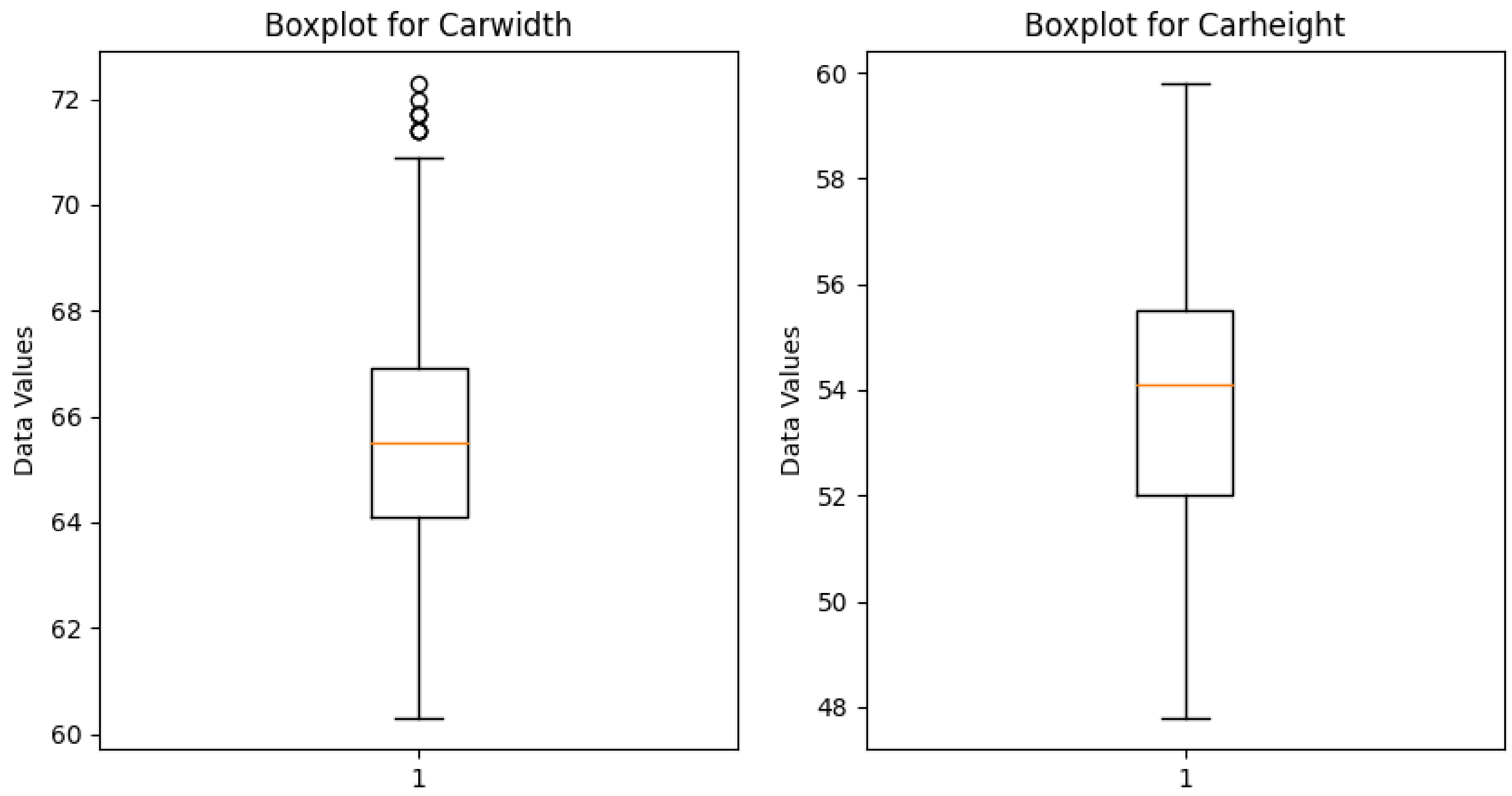

Figure 2. Boxplot for Carwidth and Carheight

In [32]:
```
# creating boxplot for 'Enginesize' and 'Price' variable

fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].boxplot(car_data['Enginesize'])
axs[1].boxplot(car_data['Price'])
axs[0].set_title('Boxplot for Enginesize')
axs[1].set_title('Boxplot for Price')
axs[0].set_ylabel('Data Values')
axs[1].set_ylabel('Data Values')

plt.show()
```

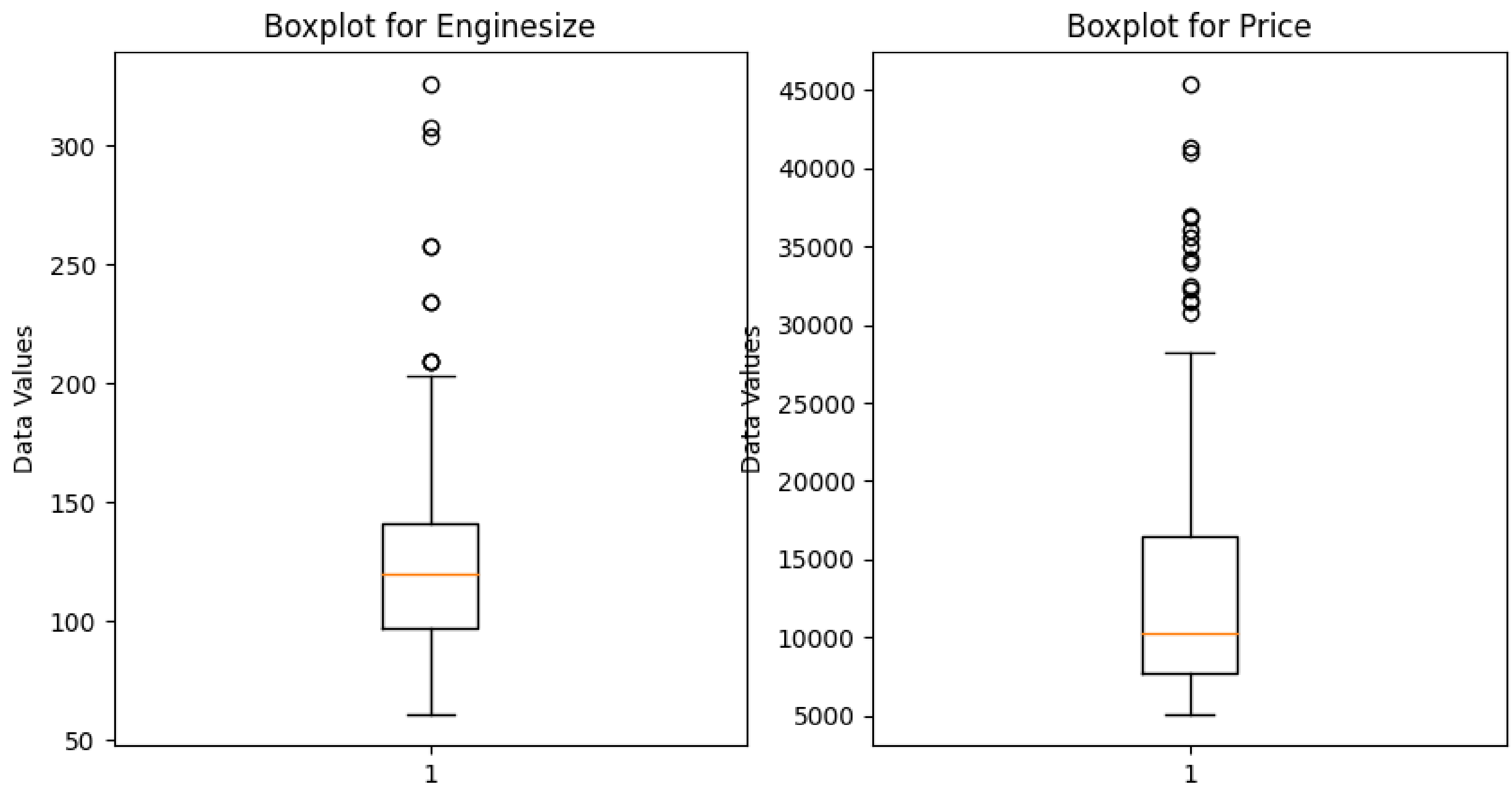

Figure 3. Boxplot for Enginesize and Price

b. Distribution Plot

The distribution plot for the various parameters of the dataset values gives an overview of the outliers that are present and the distribution of the data points across present in the dataset.

The plot below for both *Carwidth* and *Carheight* shows that the data is **normally distributed** across the data points, meaning that the data points are evenly distributed around the mean value. However, the plot for *Enginesize* and *Price* indicate that the data is **left skewed**, i.e., the data is concentrated towards a certain range of values and is not equally distributed.

In [33]:
```
# distribution plot for the carwidth & carheight

plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(car_data['Carwidth'])
plt.subplot(1,2,2)
sns.distplot(car_data['Carheight'])
plt.show()
```

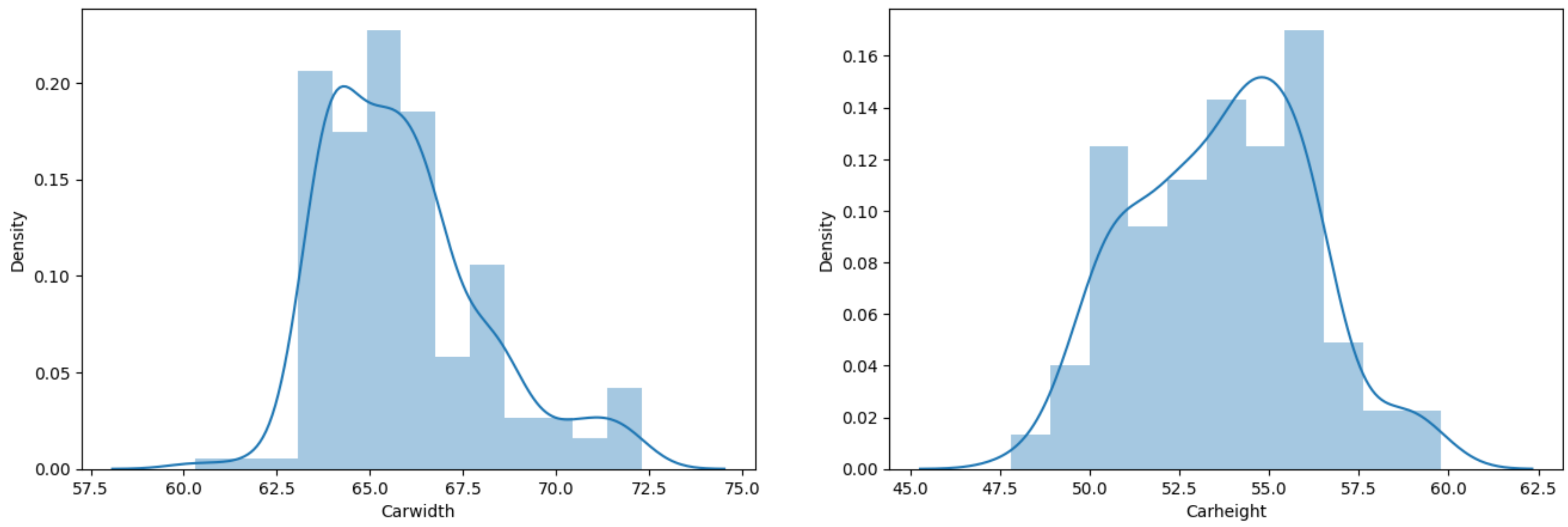

Figure 4. Distribution Plot for Carwidth and Carheight

In [34]:
```python
# distribution plot for the enginesize & price

plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(car_data['Enginesize'])
plt.subplot(1,2,2)
sns.distplot(car_data['Price'])
plt.show()
```

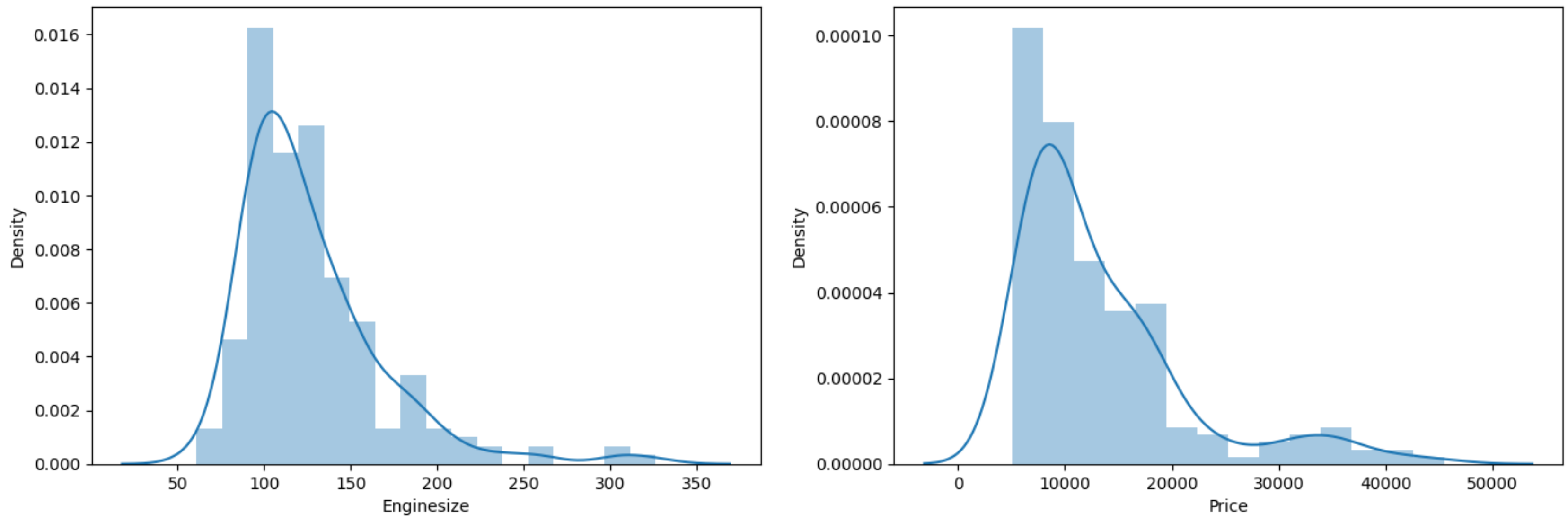

Figure 5. Distribution Plot for Enginesize and Price

Pre-Modeling Steps

1. Feature Selection & Extraction
2. Label Encoding
3. Correlation Plot
4. Defining the features for model training
5. Spliting the dataset into train & test set

1. Feature Selection and Extraction

In [35]:
```python
# Feature Extraction

target = 'Price'

features, train = featurewiz(car_data, target, corr_limit=0.7, verbose=2, sep=",",
header=0,test_data="", feature_engg="", category_encoders="")
```

```
############################################################################################
############       F A S T   F E A T U R E  E N G G    A N D    S E L E C T I O N ! ########
# Be judicious with featurewiz. Don't use it to create too many un-interpretable features! #
############################################################################################
featurewiz has selected 0.7 as the correlation limit. Change this limit to fit your needs...
Skipping feature engineering since no feature_engg input...
Skipping category encoding since no category encoders specified in input...
#### Single_Label Regression problem ####
    Loaded train data. Shape = (205, 26)
#### Single_Label Regression problem ####
No test data filename given...
#######################################################################################
######################## C L A S S I F Y I N G  V A R I A B L E S  ####################
#######################################################################################
        2 variable(s) to be removed since ID or low-information variables
        variables removed = ['Car_id', 'Carname']
train data shape before dropping 1 columns = (205, 26)
        train data shape after dropping columns = (205, 25)
GPU active on this device
```

The above code generated a feature selection & extraction report using the '**featurewiz**' function that helped in understanding which features are to be taken into consideration for the prediction of the car price where the target variable is the '**Price**' column.

The features selected by the featurewiz function are as shown below.

In [36]:
```
print("The extracted features are:")
features
```

```
The extracted features are:
```

Out[36]:
```
['Enginelocation',
 'Curbweight',
 'Fuelsystem',
 'Drivewheel',
 'Peakrpm',
 'Cylindernumber',
 'Enginetype',
 'Aspiration']
```

2. Label Encoding

As the features selected for prediction of the car prices are categorical data, it is important that these features are converted into **binary** or **numerical** type data such that the model is able to predict based on the independent variables.

Label Encoding is a method which helps to **convert the categorical variables** into **numerical values**, thus helping to transform the data point into a format where the algorithm is able to process the data for classification. *LabelEncoder()* function is used to encode the categorical type data to numerical type, where new columns of data are created for the categorical field value in the dataset which will be used in the training of the model.

In [8]:
```
labelencoder = LabelEncoder()

car_data['Carname_Label'] = labelencoder.fit_transform(car_data["Carname"])
car_data['Fueltype_Label'] = labelencoder.fit_transform(car_data["Fueltype"])
car_data['Aspiration_Label'] = labelencoder.fit_transform(car_data["Aspiration"])
car_data['Doornumber_Label'] = labelencoder.fit_transform(car_data["Doornumber"])
car_data['Carbody_Label'] = labelencoder.fit_transform(car_data["Carbody"])
car_data['Drivewheel_Label'] = labelencoder.fit_transform(car_data["Drivewheel"])
car_data['Enginelocation_Label'] = labelencoder.fit_transform(car_data["Enginelocation"])
car_data['Enginetype_Label'] = labelencoder.fit_transform(car_data["Enginetype"])
car_data['Cylindernumber_Label'] = labelencoder.fit_transform(car_data["Cylindernumber"])
car_data['Fuelsystem_Label'] = labelencoder.fit_transform(car_data["Fuelsystem"])

print("Label Encoding Successful.")
```

```
Label Encoding Successful.
```

In [87]:
```
car_data
```

Out[87]:

	Car_id	Symboling	Carname	Fueltype	Aspiration	Doornumber	Carbody	Drivewheel	Enginelocation	Wheelbase	...	Carnam
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	
...	...	...	...	...	...	...	...	...	...	...	...	
200	201	-1	volvo 145e (sw)	gas	std	four	sedan	rwd	front	109.1	...	
201	202	-1	volvo 144ea	gas	turbo	four	sedan	rwd	front	109.1	...	
202	203	-1	volvo 244dl	gas	std	four	sedan	rwd	front	109.1	...	
203	204	-1	volvo 246	diesel	turbo	four	sedan	rwd	front	109.1	...	
204	205	-1	volvo 264gl	gas	turbo	four	sedan	rwd	front	109.1	...	

205 rows × 36 columns

Table 7. Dataframe after Label Encoding

Field values after Label Encoding

In [88]:
```
car_data.columns
```

Out[88]:
```
Index(['Car_id', 'Symboling', 'Carname', 'Fueltype', 'Aspiration',
       'Doornumber', 'Carbody', 'Drivewheel', 'Enginelocation', 'Wheelbase',
       'Carlength', 'Carwidth', 'Carheight', 'Curbweight', 'Enginetype',
       'Cylindernumber', 'Enginesize', 'Fuelsystem', 'Boreratio', 'Stroke',
       'Compressionratio', 'Horsepower', 'Peakrpm', 'Citympg', 'Highwaympg',
       'Price', 'Carname_Label', 'Fueltype_Label', 'Aspiration_Label',
       'Doornumber_Label', 'Carbody_Label', 'Drivewheel_Label',
       'Enginelocation_Label', 'Enginetype_Label', 'Cylindernumber_Label',
       'Fuelsystem_Label'],
      dtype='object')
```

3. Corelation Plot

A **correlation plot** or matrix is a *visual representation of the variables* present in the dataset which helps in understanding the *relationship* between the different variables and how highly the variables are corelated to each other.

The values of the correlation plot range from **-1 to 1**, where -1 indicates a **negative correlation** between the variables, 0 indicates **no correlation**, and 1 indicates a **positive correlation**.

The variables that have positive correlation are said to be highly correlated to each and hence either of the two variables must be removed for the model building as it may lead to **multicollinearity** where the efficiency of the model may reduce.

In [9]:
```
# creating a new dataframe for defining only required features for correlation plot

new_car_data = pd.DataFrame()

new_car_data['Carname_Label'] = car_data['Carname_Label']
new_car_data['Fueltype_Label'] = car_data['Fueltype_Label']
new_car_data['Aspiration_Label'] = car_data['Aspiration_Label']
new_car_data['Carbody_Label'] = car_data['Carbody_Label']
new_car_data['Enginelocation_Label'] = car_data['Enginelocation_Label']
new_car_data['Wheelbase'] = car_data['Wheelbase']
new_car_data['Carlength'] = car_data['Carlength']
new_car_data['Carwidth'] = car_data['Carwidth']
new_car_data['Carheight'] = car_data['Carheight']
new_car_data['Curbweight'] = car_data['Curbweight']
new_car_data['Enginetype_Label'] = car_data['Enginetype_Label']
new_car_data['Cylindernumber_Label'] = car_data['Cylindernumber_Label']
new_car_data['Enginesize'] = car_data['Enginesize']
new_car_data['Fuelsystem_Label'] = car_data['Fuelsystem_Label']
new_car_data['Peakrpm'] = car_data['Peakrpm']
new_car_data['Price'] = car_data['Price']

print("New Dataframe Created!")
```

```
New Dataframe Created!
```

```
In [142]: # plotting correlation matrix

plt.figure(figsize = (12,5))
ax = plt.subplot()
sns.heatmap(new_car_data.corr(),annot=True, fmt='.1f', ax=ax, cmap="YlGnBu")
ax.set_title('Correlation Plot');
```

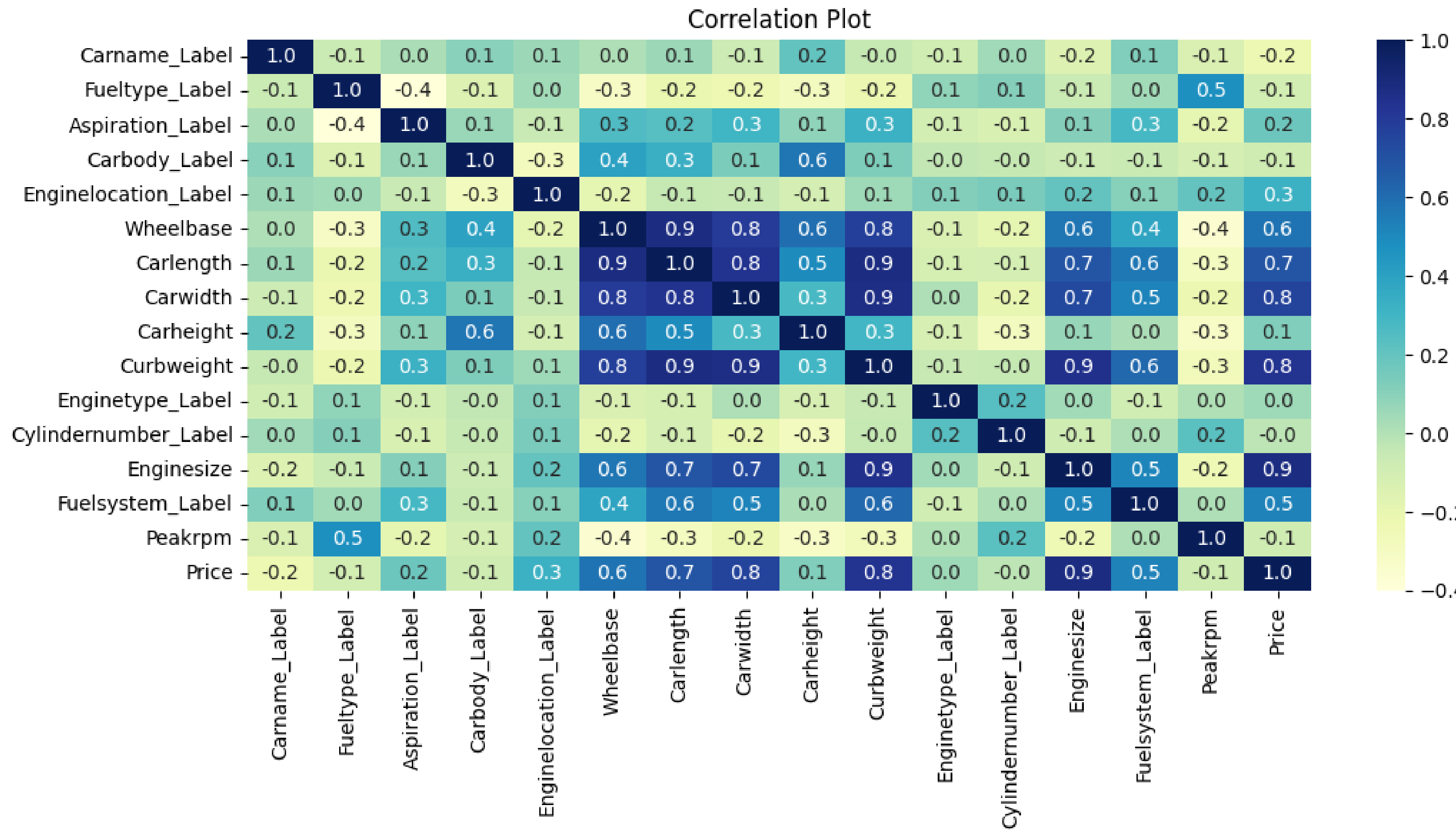

Figure 6. Correlation Plot

Looking for corelations between independent & dependent variables

---

As observed in the correlation matrix above, we see that there are many variables or features that are *highly correlated* to each other and hence we need to analyze the features that are strongly correlated such that these features are excluded from the training of the model in order to avoid **multicollinearity** and *improve the efficiency of the model*. The following features are highly correlated with the other features in the dataset and can be excluded from model building.

Correlation among the variables:

1. **Wheelbase** is highly correlated with *Carlength, Carwidth, Carheight, and Curbweight*, hence it is excluded
2. **Fueltype** is highly correlated with *Peakrpm*, hence Fueltype is excluded
3. **Carbody** is highly correlated with *Carheight*, hence Carheight is excluded
4. **Carlength**, **Carwidth**, **Carheight** and **Curbweight** are all highly correlated to each. Thus, Carlegth & Carheight are excluded
5. **Fuelsystem** is correlated to *Carwidth, Carlength, Carheight, and Curbweight*, hence we exclude the Fuelsystem, as it is more strongly correlated to *Curbweight with correlation value of 0.6*

Understanding the top features selected by Correlation Matrix

---

```
In [129]: corr_result = new_car_data.corr()
correlation_price = corr_result['Price'].sort_values(ascending=False)
topfeatures = correlation_price[1:4]
print("The top features selected by correlation matrix for Price:")
print(topfeatures)
```

```
The top features selected by correlation matrix for Price:
Enginesize     0.874145
Curbweight     0.835305
Carwidth       0.759325
Name: Price, dtype: float64
```

Lasso Regression to select the most important features for model training [4]

In [131]:
```
A = new_car_data.drop(['Price'], axis=1)
B = new_car_data['Price']
lasso_result = Lasso(alpha=0.1)
lasso_result.fit(A, B)
coef = pd.Series(lasso_result.coef_, index=A.columns)
features_lasso = coef.abs().sort_values(ascending=False).head(3).index
print("The top three features selected by Lasso regression:")
print(features_lasso)
```

```
The top three features selected by Lasso regression:
Index(['Enginelocation_Label', 'Carbody_Label', 'Carwidth'], dtype='object')
```

Features selected for model building

The features that are selected for the model building based on the Feature Selection & Extraction, Correlation Plot, and Lasso Regression are as follows:

Feature Selection & Extraction	Correlation Matrix	Lasso Regression
Enginelocation	Enginesize	Enginelocation
Curbweight	Carwidth	Carbody
Drivewheel	Curbweight	Carwidth
Peakrpm		
Cylindernumber		
Enginetype		
Aspiration		

Table 8. Features selected for model building

4. Defining the features for model training

The model is trained & built on the below mentioned features that is selected from the analysis of the Feature selection and extraction report, Correlation matrix, and Lasso regression.

In [10]:
```
X = new_car_data[['Enginesize', 'Curbweight', 'Carwidth', 'Enginelocation_Label', 'Carbody_Label', 'Carname_L
y = new_car_data['Price']
```

5. Splitting the dataset into train & test set

The dataset is split into training and testing data with a random split of **80%** train set and **20%** for test data.

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 42)
```

Model Building

Building the **Linear Regression** model to predict the price of the car based on the features selected for training.

Fitting the LR model

In [12]:
```
regressor_model = LinearRegression()
regressor_model.fit(X_train, y_train)
```

Out[12]:
```
▾ LinearRegression
LinearRegression()
```

Displaying the coefficients & intercepts after fitting the model

As observed from the below code, the coefficient values of the variables are either positive or negative, which indicates that the variables with **positive** value have a **positive relationship with the target variable** whereas values having a **negative sign** indicate that there is a **negative relationship between the independent variable and the target variable.** This helps in understanding which feature contributes in the prediction of the target variable, which in this case is this '**Price**'.

In [15]:
```python
coefficients = pd.DataFrame(regressor_model.coef_, X.columns, columns=['Coefficient'])
print(coefficients)
print('Intercept: ', regressor_model.intercept_)
```

```
                      Coefficient
Enginesize              64.743563
Curbweight               5.336752
Carwidth               903.647061
Enginelocation_Label  14991.392511
Carbody_Label          -620.898334
Carname_Label           -21.140461
Peakrpm                  1.273583
Enginetype_Label       128.998293
Intercept:  -72133.7827829365
```

Table 9. Coefficients & Intercepts

Summary Report of the Linear Regression model [6]

The summary report of the LR model displays various results and scores of the machine learning model that helps in understanding if the trained model is efficient or not, which in this case is the Linear Regression model for prediction of car price.

1. *R-squared:* It is the coefficient of determination that basically indicates the proportion of variation in the dependent variables. The **higher R-squared** value indicates a good fit model, and as it is observed from the summary report below, the R-squared value is **0.963**, which indicates it is a **good fit model** for the dataset.
2. *Adj. R-squared:* This is same as the R-squared value with a difference that while performing **multiple linear regression model** we consider the Adj. R-squared value as the addition of unnecessary variables to the model need a penalty and thus the R-squared value is adjusted for multiple features, else for a single independent variable, R-squared and Adj. R-squared value are the same. For the car price prediction model, the Adj. R-squared value is **0.961**.
3. *AIC & BIC values:* These values are used for the model robustness and the aim here is to minimize the AIC and BIC values in order to get an efficient model.
4. *Durbin-Watson:* This measure provides statistics for *autocorrelation in the residual*, which means that if the residual values are autocorrelated then the model will be biased which should not be the case, meaning that no value should be depending upon the other value. The ideal value for the Durbin-Watson test is from **0 to 4**, and here the Durbin-Watson test value comes to be equal to **1.930**, which is under the ideal score range. [6]
5. *coef & P>|t|:* The p-value in the summary report helps to understand which independent variables are significantly important for consideration and which are not. They are used to determine the significance of the predictor variables in the model.

   a. If **p-value is less** than the significance values of 0.05, it is considered to be **statistically significant**

   b. If the **p-value is greater** than the significance values of 0.05, it is **not considered statistically significant** which indicates that it is not contributing to the prediction of the target variables

---

- From the summary report below, it is observed that **'Peakrpm'** and **'Enginetype'** have *higher p-values* and hence it is *not statistically significant* indicating that these features are not contributing to the prediction of the car price.
- Considering the statistically significant variables, there is a *positive coefficient* obtained for **Enginesize, Curbweight, and Enginelocation** where the **highest coefficient** value is for *Enginelocation* and thus it implies that this feature will be contribute to the price of the car. The recommendation to the company is that the **Enginelocation** feature is the important parameter when it comes to *determining the price of car*

In [136]:
```
model = sm.OLS(y_train, X_train).fit()
print(model.summary())
```

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                  Price   R-squared (uncentered):                   0.963
Model:                            OLS   Adj. R-squared (uncentered):              0.961
Method:                 Least Squares   F-statistic:                              512.8
Date:                Thu, 20 Apr 2023   Prob (F-statistic):                   7.31e-108
Time:                        21:43:16   Log-Likelihood:                         -1541.9
No. Observations:                 164   AIC:                                      3100.
Df Residuals:                     156   BIC:                                      3125.
Df Model:                           8
Covariance Type:            nonrobust
========================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
Enginesize             66.9261     12.490      5.358      0.000      42.255      91.597
Curbweight              9.0689      1.028      8.821      0.000       7.038      11.100
Carwidth             -295.7895     63.348     -4.669      0.000    -420.920    -170.658
Enginelocation_Label 1.316e+04   2040.380      6.449      0.000    9127.676    1.72e+04
Carbody_Label        -679.6152    306.213     -2.219      0.028   -1284.473     -74.757
Carname_Label         -25.2596      6.115     -4.131      0.000     -37.338     -13.181
Peakrpm                 0.7892      0.528      1.495      0.137      -0.253       1.832
Enginetype_Label      176.8688    221.891      0.797      0.427    -261.431     615.168
==============================================================================
Omnibus:                        6.829   Durbin-Watson:                   1.930
Prob(Omnibus):                  0.033   Jarque-Bera (JB):                8.332
Skew:                           0.281   Prob(JB):                       0.0155
Kurtosis:                       3.951   Cond. No.                     5.01e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 5.01e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [16]:
```
y_pred = regressor_model.predict(X_test)
```

Evaluating the performance of the model [1]

The various metrics for evaluating the performance of the Linear Regression model are *Mean Absolute Error, Mean Squared Error, Root Mean Squared Error, and R-squared value.*

1. **MAE:** Measures the average absolute difference between the actual values and the predicted values
2. **MSE:** Measures the average of the squared differences between the actual values and the predicted values
3. **RMSE:** Square root of the MSE value
4. **R-squared:** Measures proportion of variance in the dependent variable

In [17]:
```
print("Model Evaluation of Linear Regression.")
print('Mean Absolute Error:', round(metrics.mean_absolute_error(y_test, y_pred),1))
print('Mean Squared Error:', round(metrics.mean_squared_error(y_test, y_pred),1))
print('Root Mean Squared Error:', round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)),1))
print("R-Squared value:", round(metrics.r2_score(y_test, y_pred),2))
```

```
Model Evaluation of Linear Regression.
Mean Absolute Error: 2409.8
Mean Squared Error: 12903101.6
Root Mean Squared Error: 3592.1
R-Squared value: 0.84
```

Residual Plot

In addition to the R-squared value, the residual plot helps to analyze and ensure that the data points are randomly distributed and have a constant variance.

In [138]:
```
plt.scatter(y_pred, y_test - y_pred)
plt.xlabel('Predicted')
plt.ylabel('Residual')
plt.axhline(y=0, color='k', linestyle='--')
```

Out[138]:
```
<matplotlib.lines.Line2D at 0x7f21d005c6a0>
```

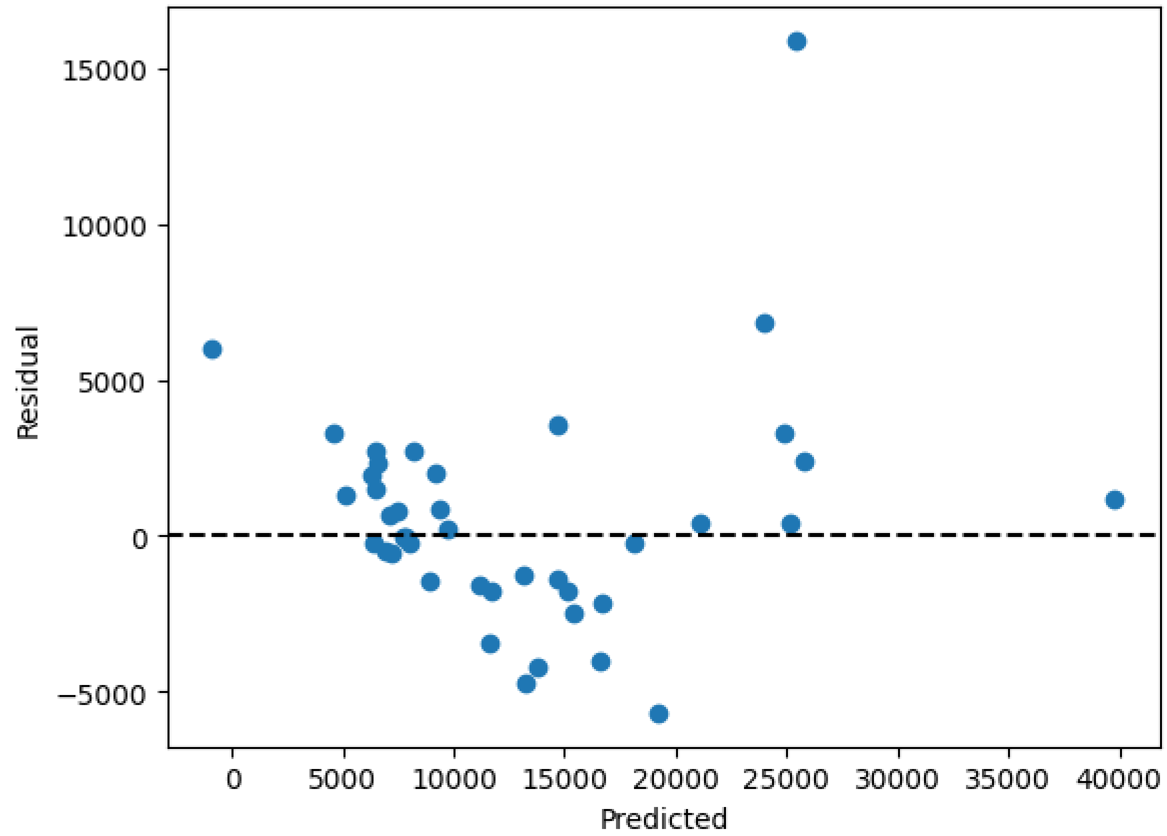

Figure 7. Residual Plot

In [18]:
```
# accuracy of the model on training and testing set

print('Accuracy of Linear Regressor model on training set: {:.2f}'.format(regressor_model.score(X_train, y_tr
print('Accuracy of Linear Regressor model on test set:     {:.2f}'.format(regressor_model.score(X_test, y_tes

result_LR = regressor_model.score(X_test, y_test)
result_LR = round(result_LR,4)
print("Overall Accuracy of the model is ",result_LR)
```

```
Accuracy of Linear Regressor model on training set: 0.89
Accuracy of Linear Regressor model on test set:     0.84
Overall Accuracy of the model is  0.8366
```

Overall Accuracy of the model is 83.66%

Accuracy on Training Data	Accuracy on Testing Data
89%	84%

The accuracy on training and testing data is nearly closeby, which indicates that the model efficiency is good on both training and testing sets of data. Hence, there is **no issue of overfitting of the model.**

Results

Respond to the following questions in your submission:

1. What were the three most significant variables?
2. Of those three, which had the greatest positive influence on car prices?
3. How accurate was the model?

Q1. What were the three most significant variables?

A1.

In order to determine the three most significant variables for the car price prediction, the **coefficients of the linear regression model** is checked upon in order to understand the **highest absolute coefficient value** which indicates that the independent variable is the *most significant variable* in determining or predicting the target variable.

The below code helps in determining the most significant variables based on the highest coefficient value. Here, **Enginelocation** has the highest coefficient value, followed by **Carwidth** and **Enginetype** and thus are the most significant variables in the prediction of the target variable. These variables contribute the most while predicting the price of the car.

In [19]:
```
coefficients_result = coefficients.sort_values(by='Coefficient', ascending=False)
print('The three most significant variables are:\n')
print(coefficients_result.head(3))
```

```
The three most significant variables are:

                      Coefficient
Enginelocation_Label  14991.392511
Carwidth                903.647061
Enginetype_Label        128.998293
```

Q2. Of those three, which had the greatest positive influence on car prices?

A2.

To check the greatest positive influence on car prices, the sign of the coefficient value can be looked upon, where a *positive sign* indicates a *positive relationship on the target variable*, i.e., the car price, whereas a *negative sign* would indicate a *negative relationship* with the target variable.

Thus, the **greatest positive influence on car prices** is the **Enginelocation**, indicating that the car prices varies with respect to the Enginelocation and hence needs to be taken into consideration while analyzing the prices of car.

In [20]:
```
influence_variable = coefficients_result.index[0]
print('The variable with the greatest positive influence on car prices is:', influence_variable)
```

```
The variable with the greatest positive influence on car prices is: Enginelocation_Label
```

Q3. How accurate was the model?

A3.

- The accuracy of the model obtained for the **train data is 89%** whereas the accuracy for the **test data is 84%**. This indicates that the model is accurately able to predict the price of the car 89% on the train data and 84% on test data, and since the accuracy of both the train and test dataset is almost nearby, we can conclude that the **model is not overfitted** and hence can be used to predict the car prices.
- Apart from the accuracy metric, if we consider the *MAE, MSE, RMSE, and R-squared values* of the model, it indicates that the model is efficient in order to predict the car price, as a **high R-squared value** which is close to 1 indicates that the model is accurate in predicting the car prices. Here, the R-squared value obtained for the regression model is **0.84**, which is an overall good score for the model.
- Also, the **mean absolute error (MAE) of 2409.8** and **root mean squared error (RMSE) of 3592.1** indicates that the predictions of the model for the car prices are relatively accurate.

Conclusion

- **Linear Regression models**

  Linear regression models are the simplest method for modeling the relationship between the dependent variable and one or more independent variables which can be used to make predictions that will identify the most significant variable that contributes to the prediction of the target variable. These models assume a linear relationship between the independent and dependent variables and can be used to solve regression based problems.

- **Car Price Prediction using LR model**

  The car price prediction is implemented using linear regression model which helps in understanding which parameters or features are contributing in the prediction of the car price. There are various *numerical and categorical* parameters present in the dataset that helps in analyzing the various features that could be useful in predicting the car price.

- **Model Performance & Accuracy**

  The model performance and accuracy is determined with the help of various evaluation metrics such as R-squared value, MSE, MAE, etc. which help in understanding if the model is accurate enough in order to predict the car prices based on the features selected for training. As observed above, we see that the R-squared value obtained is 0.84 which is close to 1 indicating that the model is accurate in predicting the price of the car. The accuracy for the train and test dataset is 89% and 84% respectively indicating that the model is not overfitted and can be used to predict the car price based on the features that are selected.

- **Recommendations**

  Based on the accuracy and coefficient values, we conclude that the model is not overfitted and hence can be used by companies in order to predict the price of the car as the accuracy of the train and test dataset is almost closeby which is 89% and 84% respectively.

  The coefficient values along with the p-values in the summary report for the model indicate that the Enginelocation parameter has a positive coefficient value which means that the feature has a positive relationship with the car price and thus is contributing strongly in the prediction.

- **Future Scope**

  Despite the Linear Regression model is accurately able to predict the car price based on the features that are selected, the accuracy of the model can still be improved based on the new features selected along with eliminating the issue of multicollinearity.

References

[1] UCI Machine Learning Repository: Automobile Data Set. (n.d.). https://archive.ics.uci.edu/ml/datasets/Automobile (https://archive.ics.uci.edu/ml/datasets/Automobile)

[2] Mali, K. (2022). Everything you need to Know about Linear Regression! Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/ (https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/)

[3] Luna, Z. (2022, January 6). Feature Selection in Machine Learning: Correlation Matrix | Univariate Testing | RFECV. Medium. https://medium.com/geekculture/feature-selection-in-machine-learning-correlation-matrix-univariate-testing-rfecv-1186168fac12 (https://medium.com/geekculture/feature-selection-in-machine-learning-correlation-matrix-univariate-testing-rfecv-1186168fac12)

[4] Great Learning Team. (2023, January 12). What is LASSO Regression Definition, Examples and Techniques. Great Learning Blog: Free Resources What Matters to Shape Your Career! https://www.mygreatlearning.com/blog/understanding-of-lasso-regression/#:~:text=Lasso%20regression%20is%20a%20regularization,i.e.%20models%20with%20fewer%20parameters (https://www.mygreatlearning.com/blog/understanding-of-lasso-regression/#:~:text=Lasso%20regression%20is%20a%20regularization,i.e.%20models%20with%20fewer%20parameters)).

[5] Mawardi, D. (2018, May 29). Linear Regression in Python - Towards Data Science. Medium. https://towardsdatascience.com/linear-regression-in-python-9a1f5f000606 (https://towardsdatascience.com/linear-regression-in-python-9a1f5f000606)

[6] Mahmood, M. S. (2022, April 24). Simple Explanation of Statsmodel Linear Regression Model Summary. Medium. https://towardsdatascience.com/simple-explanation-of-statsmodel-linear-regression-model-summary-35961919868b (https://towardsdatascience.com/simple-explanation-of-statsmodel-linear-regression-model-summary-35961919868b)