```
https://github.com/ondreiya/capstone
```

https://github.com/ondreiya/capstone

```
from google.colab import drive
drive.mount("/content/drive")

%cd /content/drive/MyDrive/ST1
```

```
Mounted at /content/drive
/content/drive/MyDrive/ST1
```

```
!ls
```

```
car_price_prediction.csv  car_price_prediction.ipynb  DataforML.pkl  Final_XGB_Model.pkl
```

```
# Suppressing the warning messages
import warnings
warnings.filterwarnings('ignore')
```

This project is based on the Car Price Prediction Challenge data available from Kaggle repository (https://www.kaggle.com/datasets/deepcontractor/car-price-prediction-challenge?resource=download)

- It contains the details of 19,237 car prices.
- My project task is to create a machine learning model which can predict the average price of a car based on its characteristics.
- To solve this, I will approach the task, with a step-by-step approach to create a data anlysis and prediction model based on (machine learning/AI algorithms, regression algorithm for example) available from different Python packages, modules and classes.

Step 1: Reading the Data with Python

One of the most important steps in data analysis.

```
# Reading the dataset
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
car_price = pd.read_csv("/content/drive/MyDrive/ST1/car_price_prediction.csv", encoding="latin")
print("Shape before deleting duplicate values: ", car_price.shape)
# Removing duplicate rows if any
car_price=car_price.drop_duplicates()
print("Shape after deleting duplicate values: ", car_price.shape)
car_price['Mileage'] = car_price['Mileage'].str.replace(' km', '').astype(float)
car_price['Engine volume'] = car_price['Engine volume'].str.replace(' Turbo', '').astype(float)
car_price['Levy'] = car_price['Levy'].replace('-', np.nan).astype(float)

le = LabelEncoder()
car_price['Manufacturer'] = le.fit_transform(car_price['Manufacturer']).astype(float)
car_price['Model'] = le.fit_transform(car_price['Model']).astype(float)
car_price['Category'] = le.fit_transform(car_price['Category']).astype(float)
car_price['Leather interior'] = le.fit_transform(car_price['Leather interior']).astype(float)
car_price['Fuel type'] = le.fit_transform(car_price['Fuel type']).astype(float)
car_price['Gear box type'] = le.fit_transform(car_price['Gear box type']).astype(float)
car_price['Drive wheels'] = le.fit_transform(car_price['Drive wheels']).astype(float)
car_price['Doors'] = le.fit_transform(car_price['Doors']).astype(float)
car_price['Wheel'] = le.fit_transform(car_price['Wheel']).astype(float)
car_price['Color'] = le.fit_transform(car_price['Color']).astype(float)
# Printing sample data
# Start observing the Quantitative/Categorical/Qualitative variables
car_price.head(10)
```

```
Shape before deleting duplicate values:  (19237, 18)
Shape after deleting duplicate values:  (18924, 18)
```

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Eng vol
0	45654403	13328	1399.0	32.0	1242.0	2010	4.0	1.0	2.0	
1	44731507	16621	1018.0	8.0	658.0	2011	4.0	0.0	5.0	
2	45774419	8467	NaN	21.0	684.0	2006	3.0	0.0	5.0	
3	45769185	3607	862.0	16.0	661.0	2011	4.0	1.0	2.0	
4	45809263	11726	446.0	21.0	684.0	2014	3.0	1.0	5.0	
5	45802912	39493	891.0	23.0	1305.0	2016	4.0	1.0	1.0	
6	45656768	1803	761.0	58.0	1154.0	2010	3.0	1.0	2.0	
7	45816158	549	751.0	23.0	1334.0	2013	9.0	1.0	5.0	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	

Key Observations from Step 1 about Data Description

- This file contains 19,237 car details before deletion of duplication for car prices and 18,924 car details after deletion of duplication.
- There are 18 attributes and they are outlined below:

  1. ID
  2. Price
  3. Levy
  4. Manufacturer
  5. Model
  6. Prod. Year
  7. Category
  8. Leather Interior
  9. Fuel Type
  10. Engine Volume
  11. Mileage
  12. Cylinders
  13. Gear Box Type
  14. Drive Wheels
  15. Doors
  16. Wheel
  17. Color
  18. Airbags

Step 2: Problem Statement Definition

- Creating a prediction model to predict the price (Price) of a car.
- Target Variable: Price Predictors/Features: ID, Levy, Manufacturer, Model, Prod. Year, etc.

Step 3: Choosing the Appropriate ML/AI Algorithm for Data Analysis

- Based on the problem statement we need to create a supervised ML Regression model, as the target variable is continuous.

```
# Importing necessary libraries
import pandas as pd

# Reading the dataset
df = pd.read_csv("car_price_prediction.csv")

# Displaying the first few rows of the dataset.
print("First 5 rows of the dataset: ")
print(df.head())

# Displaying information about the dataset including the data types
print("\nDataset Information: ")
print(df.info())

# Identifying continuous target variable
print("\nContinuous Target Variable: ")
for column in df.columns:
  if df[column].dtype in ["int64", "float64"] and column != "ID":
    break
```

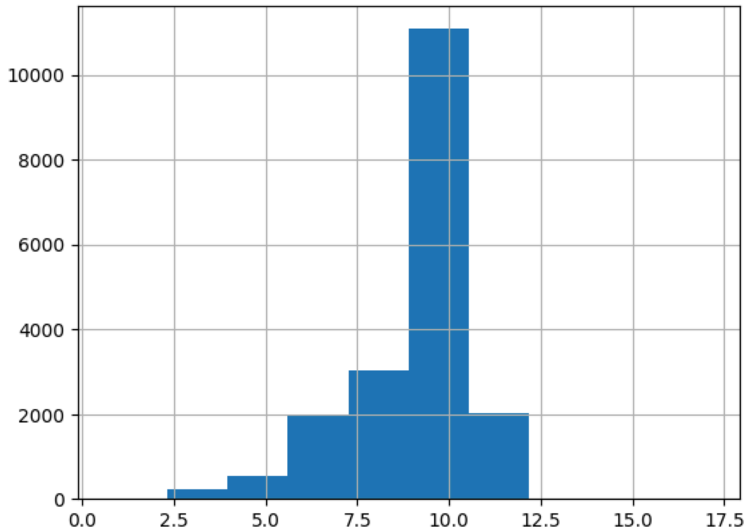
```
First 5 rows of the dataset: 
         ID  Price  Levy Manufacturer    Model  Prod. year   Category  \
0  45654403  13328  1399        LEXUS   RX 450        2010       Jeep   
1  44731507  16621  1018    CHEVROLET  Equinox        2011       Jeep   
2  45774419   8467     -        HONDA      FIT        2006  Hatchback   
3  45769185   3607   862         FORD   Escape        2011       Jeep   
4  45809263  11726   446        HONDA      FIT        2014  Hatchback   

  Leather interior Fuel type Engine volume    Mileage  Cylinders  \
0              Yes    Hybrid           3.5  186005 km        6.0   
1               No    Petrol             3  192000 km        6.0   
2               No    Petrol           1.3  200000 km        4.0   
3              Yes    Hybrid           2.5  168966 km        4.0   
4              Yes    Petrol           1.3   91901 km        4.0   

  Gear box type Drive wheels   Doors             Wheel   Color  Airbags  
0     Automatic          4x4  04-May        Left wheel  Silver       12  
1     Tiptronic          4x4  04-May        Left wheel   Black        8  
2      Variator        Front  04-May  Right-hand drive   Black        2  
3     Automatic          4x4  04-May        Left wheel   White        0  
4     Automatic        Front  04-May        Left wheel  Silver        4  

Dataset Information: 
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19237 entries, 0 to 19236
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                19237 non-null  int64  
 1   Price             19237 non-null  int64  
 2   Levy              19237 non-null  object 
 3   Manufacturer      19237 non-null  object 
 4   Model             19237 non-null  object 
 5   Prod. year        19237 non-null  int64  
 6   Category          19237 non-null  object 
 7   Leather interior  19237 non-null  object 
 8   Fuel type         19237 non-null  object 
 9   Engine volume     19237 non-null  object 
 10  Mileage           19237 non-null  object 
 11  Cylinders         19237 non-null  float64
 12  Gear box type     19237 non-null  object 
 13  Drive wheels      19237 non-null  object 
 14  Doors             19237 non-null  object 
 15  Wheel             19237 non-null  object 
 16  Color             19237 non-null  object 
 17  Airbags           19237 non-null  int64  
dtypes: float64(1), int64(4), object(13)
memory usage: 2.6+ MB
None

Continuous Target Variable: 
```

Step 4: Looking at the Class Distribution (Target Variable distribution to check if the data is balanced or skewed).

- If target variable's distribution is too skewed then the predictive modelling will lead to poor results.
- Ideally, the bell curve is desirable but a slightly positive or negative skew is also fine.
- When performing Regression Algorithm modelling and analysis, we need to make sure the histogram looks like a bell curve or a slightly skewed version of it. Otherwise, it will impact the Machine Learning algorithm's ability to learn all the scenarios from the data.

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
# Creating histogram as the target variable is continuous.
# This will help us understand the distribution of the Price values.
print("Price Before:\n" , car_price['Price'])
car_price['Price'] = np.log(car_price['Price'] + 1)

car_price["Price"].hist()
print("Price After:\n" , car_price['Price'])
```

```
Price Before:
 0        13328
1        16621
2         8467
3         3607
4        11726
         ...
19232     8467
19233    15681
19234    26108
19235     5331
19236      470
Name: Price, Length: 18924, dtype: int64
Price After:
 0         9.497697
1         9.718482
2         9.044050
3         8.190909
4         9.369649
           ...
19232     9.044050
19233     9.660269
19234    10.170035
19235     8.581482
19236     6.154858
Name: Price, Length: 18924, dtype: float64
```



Observations from Step 4

- The data distribution of the target variable is satisfactory to proceed further.
- There are sufficient number of rows for most values to learn from.

Step 5: Basic Exploraty Data Analysis

- This step is performed to gauge the overall data.
    - The volume of data and the types of columns present in the data.
- Initial assessment of the data should be done to identify which columns are Quantitative, Categorical or Qualitative.
- This step helps with starting the column/data rejection process.
- There are 4 commands which are used for basic data exploratory in analysis in Python.
- head(): helps to see a few sample rows of the data.
- info(): provides the summarised information of the data.
- describe(): provides the descriptive statistical details of the data. nunique(): helps us to identify if a column is categorical or continuous.

```
# Looking at sample rows in the data
car_price.head()
```

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	E v
0	45654403	9.497697	1399.0	32.0	1242.0	2010	4.0	1.0	2.0	
1	44731507	9.718482	1018.0	8.0	658.0	2011	4.0	0.0	5.0	
2	45774419	9.044050	NaN	21.0	684.0	2006	3.0	0.0	5.0	

```
# Looking at sample rows in the data
car_price.tail()
```

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fu ty
19232	45798355	9.044050	NaN	36.0	385.0	1999	1.0	1.0	
19233	45778856	9.660269	831.0	23.0	1334.0	2011	9.0	1.0	
19234	45804997	10.170035	836.0	23.0	1442.0	2010	4.0	1.0	

```
# Observing the summarised information of data.
# Data types, missing values based on the number of non-null values vs total rows, etc.
# Remove the variables from data which have too many missing values (missing values > 30%)
# Remove qualitative variables which cannot be used in Machine Learning
car_price.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 18924 entries, 0 to 19236
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                18924 non-null  int64
 1   Price             18924 non-null  float64
 2   Levy              13215 non-null  float64
 3   Manufacturer      18924 non-null  float64
 4   Model             18924 non-null  float64
 5   Prod. year        18924 non-null  int64
 6   Category          18924 non-null  float64
 7   Leather interior  18924 non-null  float64
 8   Fuel type         18924 non-null  float64
 9   Engine volume     18924 non-null  float64
 10  Mileage           18924 non-null  float64
 11  Cylinders         18924 non-null  float64
 12  Gear box type     18924 non-null  float64
 13  Drive wheels      18924 non-null  float64
 14  Doors             18924 non-null  float64
 15  Wheel             18924 non-null  float64
 16  Color             18924 non-null  float64
 17  Airbags           18924 non-null  int64
dtypes: float64(15), int64(3)
memory usage: 2.7 MB
```

```
# Looking at the descriptive statistics of the data.
car_price.describe(include="all")
```

	ID	Price	Levy	Manufacturer	Model	Prod. ye
count	1.892400e+04	18924.000000	13215.000000	18924.000000	18924.000000	18924.0000
mean	4.557538e+07	9.028981	906.299205	33.087349	862.224530	2010.9142
std	9.375468e+05	1.585140	463.296871	17.787356	410.990871	5.6657
min	2.074688e+07	0.693147	87.000000	0.000000	0.000000	1939.0000
25%	4.569501e+07	8.581482	640.000000	21.000000	537.000000	2009.0000
50%	4.577191e+07	9.485925	781.000000	32.000000	834.000000	2012.0000
75%	4.580174e+07	10.001703	1058.000000	54.000000	1226.000000	2015.0000
max	4.581665e+07	17.085365	11714.000000	64.000000	1589.000000	2020.0000

```
# Finding unique values for each column.
# To understand which column is categorical and which one is continuous -
# typically if the number of unique values are < 20, then the variable is likely
# to be categorical
car_price.nunique()
```

```
ID                  18924
Price                2315
Levy                  558
Manufacturer           65
Model                1590
Prod. year             54
Category               11
Leather interior        2
Fuel type               7
Engine volume          65
Mileage              7687
Cylinders              13
Gear box type           4
Drive wheels            3
Doors                   3
Wheel                   2
Color                  16
Airbags                17
dtype: int64
```

Observations from Step 5 - Basic Exploratory Data Analysis

Based on the basic exploration above, I can identify that:

- ID - Continuous. Selected.
- Price - Continuous. Selected. This is the target variable which is to be predicted by the proposed regression model.
- Levy - Continuous. Selected.
- Manufacturer - Continuous. Selected.
- Model - Continuous. Selected.
- Prod. year - Continuous. Selected.
- Category - Categorical. Selected.
- Leather interior - Categorical. Selected.
- Fuel type - Categorical. Selected.
- Engine volume - Continuous. Selected.
- Mileage - Continuous. Selected.
- Cylinders - Categorical. Selected.
- Gear box type - Categorical. Selected.
- Drive wheels - Categorical. Selected.
- Doors - Categorical. Selected.
- Wheel - Categorical. Selected.
- Color - Categorical. Selected.
- Airbags - Categorical. Selected.

Step 6: Removing Unwanted Columns

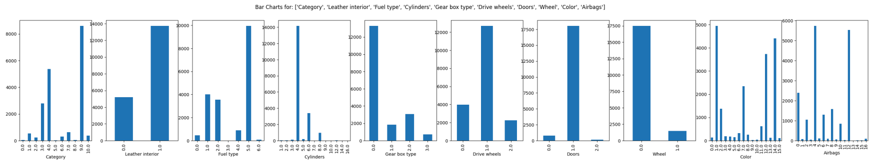
There are no qualitative columns in the data, hence, I will not be removing any column.

Step 7: Visual Exploratory Data Analysis

- Visualise distribution of all the Categorical Predictor variables in the data using bar plots.
- We can spot a categorical variable in the data by looking at the unique values in them.
- Typically a categorical variable contains less than 20 unique values AND there are repetition of values, which means the data can be grouped by those unique values.
- Based on the Basic Exploration Data Analysis in the previous step, we can see 10 Categorical Predictors in the data.
    - Category
    - Leather interior
    - Fuel type
    - Cylinders
    - Gear box type
    - Drive wheels
    - Doors
    - Wheel
    - Color
    - Airbags

- We will use bar charts to see how the data is distributed for these categorical columns.

```
# Plotting multiple bar charts at once for categorical variables.
# Since there is no default function which can plot bar charts for multiple
# columns at once
# We are defining our own function for the same

def PlotBarCharts(inpData, colsToPlot):
  %matplotlib inline

  import matplotlib.pyplot as plt

  # Generating multiple subplots
  fig, subPlot=plt.subplots(nrows=1, ncols=len(colsToPlot), figsize=(35,5))
  fig.suptitle("Bar Charts for: "+str(colsToPlot))

  for colName, plotNumber in zip(colsToPlot, range(len(colsToPlot))):
    inpData.groupby(colName).size().plot(kind="bar", ax=subPlot[plotNumber])


# Calling the function PlotBarCharts() we have created

PlotBarCharts(inpData=car_price,
              colsToPlot=["Category", "Leather interior", "Fuel type", "Cylinders", "Gear box type",
                          "Drive wheels", "Doors", "Wheel", "Color", "Airbags"])
```



Observations from Step 7 - Visual Exploratory Data Analysis

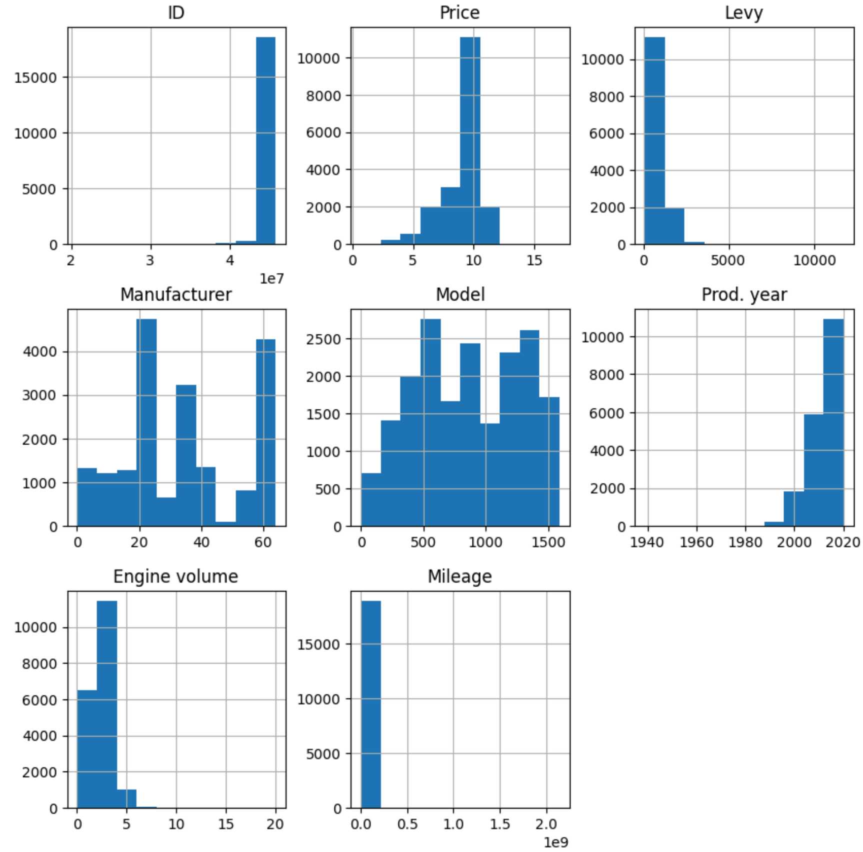
- Bar Charts have allowed for interpretations on the 10 data columns.
- The bar charts represent the frequencies of each category on the Y-axis and the category names on the X-axis.
- In the ideal bar chart, each category has comparable frequency. *Hence, there are enough rows for each category in the data for the ML/AI regression algortihm to learn.
- If there is a column which shows too skewed distribution where there is only one dominant bar and the other categories are present in very low numbers:
    - These columns may not be very helpful in machine learning model development.
- We can confirm this with the correlation analysis step coming up, and take a final call to select or reject the column/data attribute.
- In this dataset, it is worth noting that "Leather interior", "Cylinders", "Gear box type", "Drive wheels", "Doors" and "Wheel" are skewed as there is only one bar that dominates the others.
    - Such columns may not be correlated with the Target Variable as there is not enough information to learn.
- However, the selected Categorical Variables will be slected for further analysis.

Step 8: Now Visualise Distribution of all Continuous Predictor Variables in the Data using Histograms

- Based on the Basic Exploratory Data Analysis, there are 8 Continuous Predictor variables.
    - ID
    - Price
    - Levy
    - Manufacturer
    - Model
    - Prod. Year
    - Engine Volume
    - Mileage

```
# Plotting histograms of multiple columns together
car_price.hist(["ID", "Price", "Levy", "Manufacturer", "Model", "Prod. year",
                "Engine volume", "Mileage"], figsize=(10,10))
```

```
array([[<Axes: title={'center': 'ID'}>,
        <Axes: title={'center': 'Price'}>,
        <Axes: title={'center': 'Levy'}>],
       [<Axes: title={'center': 'Manufacturer'}>,
        <Axes: title={'center': 'Model'}>,
        <Axes: title={'center': 'Prod. year'}>],
       [<Axes: title={'center': 'Engine volume'}>,
        <Axes: title={'center': 'Mileage'}>, <Axes: >]], dtype=object)
```



Observations from Step 8:

- Each of the histograms above show the data disribution for a single Continuous Variable.
- The X-axis shows the range of values and Y-axis represents the number of values in that range.
  - For example, the "Prod. year" histogram has around 11,000 rows of data between years 1940-2020.
- The ideal outcome for a histogram is a bell curve or slightly skewed bell curve.
- If there is too much skewness, then outlier removal treatment should be done and the column should be re-examined. If that doesn't solve the problem then reject the column/data attribute.
- Selected Continuous Variables:
  - ID
  - Price
  - Levy
  - Manufacturer
  - Model
  - Prod. year
  - Engine volume
  - Mileage

Step 9: Outlier Analysis

- Outliers are extreme values in the data which are far away from most of the values.
- They can be seen as the tails in the histogram.
- Outlier must be treated one column/data attribute at a time.
- As the treatment will be slightly different for each column, why should the outliers be analysed?
  - Outliers bias the building of machine learning models.
  - As the algorithm tries to fit the extreme value, it goes away from the majority of the data.
- Outlined below are two options to treat outliers in the data.
  - Option 1: Delete the outlier Records. Only if there are a few rows that are lost.
  - Option 2: Impute the outlier values with a logical business value.
- Let's find out the most logical value to replace the outliers by looking at the histogram.

```
# Replacing outliers for "Levy"
# Finding nearest values to 5000 mark
car_price["Levy"][car_price["Levy"]<5000].sort_values(ascending=False)
```

```
19048    4860.0
17495    4741.0
9222     4736.0
1571     4508.0
8887     4283.0
          ...
7022       87.0
12917      87.0
2010       87.0
4814       87.0
7685       87.0
Name: Levy, Length: 13200, dtype: float64
```

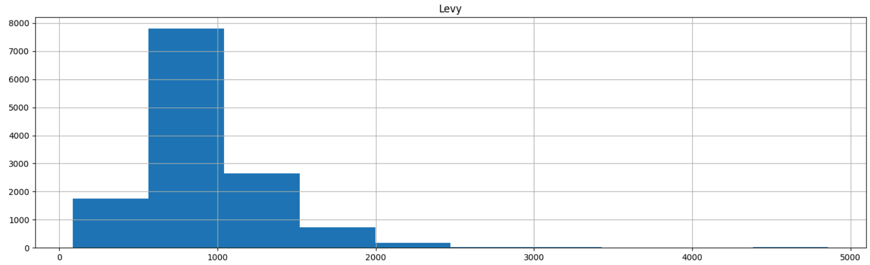
Observation: Above result shows the nearest logical value is 4860, hence, replacing any value above 5000 with it.

```
# Replacing outliers with nearest possible value
car_price["Levy"][car_price["Levy"]>5000] =4860
```

Step 10: Visualising Data Distribution after Outlier Removal

```
car_price.hist(["Levy"], figsize=(18,5))
```

```
array([[<Axes: title={'center': 'Levy'}>]], dtype=object)
```



Observation from Step 10:

- The distribution has improved after outlier treatment.
- It is now slightly more balanced.

Step 11: Missing Values Analysis

- Missing values are treated for each column separately.

- If a column has more than 30% data missing, then missing value treatment cannot be done.
- That column must be rejected because too much information is missing.
- Below are some options for treating missing values in data:
- Delete the missing value rows if there are only a few records.
- Impute the missing values with MEDIAN value for continuous variables.
- Impute the missing values with MODE value for categorical variables.
- Interpolate the values based on nearby values.
- Interpolates the values based on business logic.

```
# Finding how many missing values there are for each column
car_price.isnull().sum()
```

```
ID                   0
Price                0
Levy              5709
Manufacturer         0
Model                0
Prod. year           0
Category             0
Leather interior     0
Fuel type            0
Engine volume        0
Mileage              0
Cylinders            0
Gear box type        0
Drive wheels         0
Doors                0
Wheel                0
Color                0
Airbags              0
dtype: int64
```

Observations from Step 11: Missing Value Analysis

- "Levy" has data missing, therefore it needs to be removed.

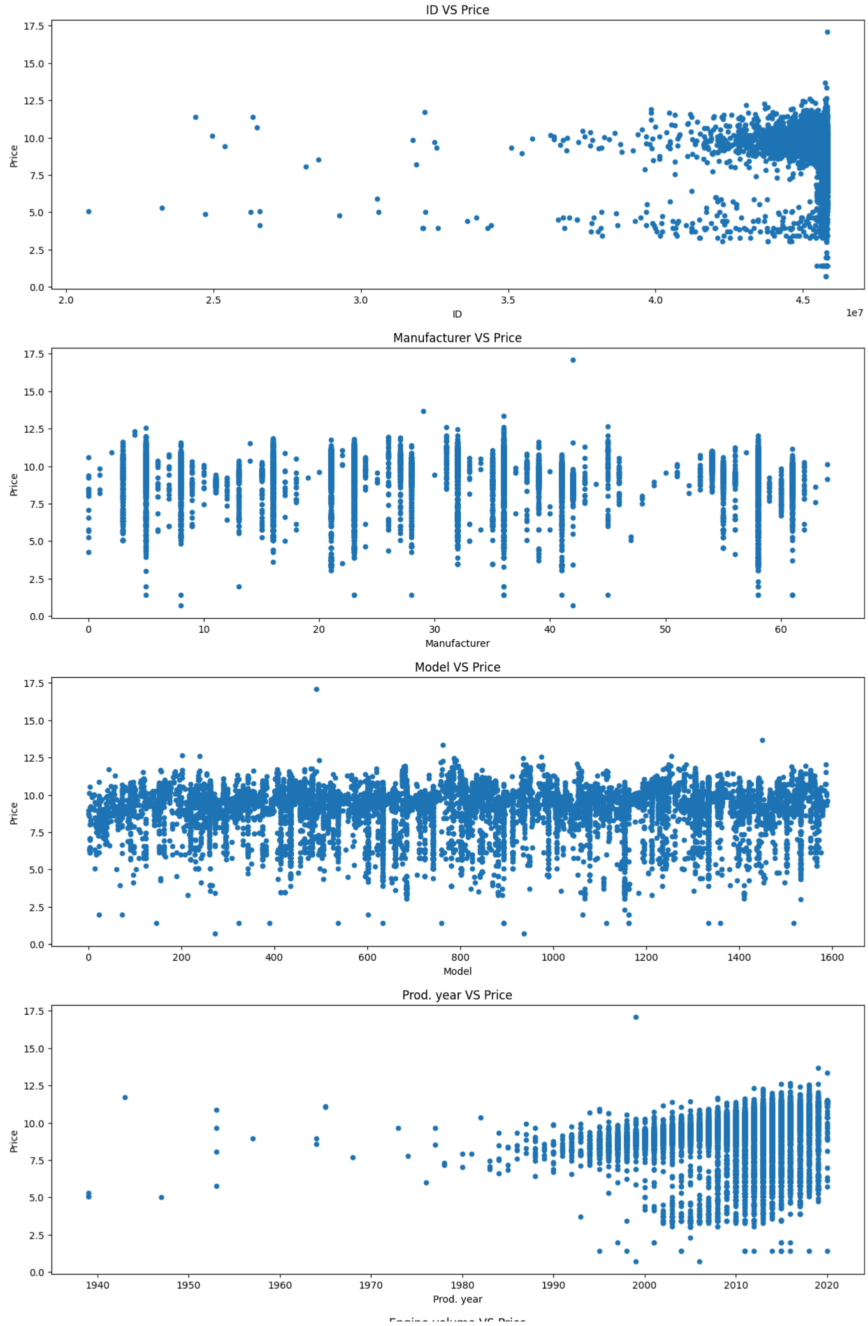
Step 12: Feature Selection (Attribute Selection)

- Now, we choose the best columns (features) which are correlated to the target variable.
- This can be done directly by measuring the correlation values or ANOVa analysis or chi-square tests.
- However, it is always helpful to visualise the relation between the target/class variable and each of the predictors (features) to get a better sense of data.
- Listed below are some of the techniques used for visualising relationship between two variables as well as measuring the strength statistically.
- Visual exploration of relationship between variables.
- Continuous vs Continuous -- Scatter Plot
- Categorical vs Continuous -- Box Plot
- Categorical vs Categorical -- Grouped Bar Plots
- Statistical measurement of relationship strength between variables.
- Continuous vs Continuous -- Correlation Matrix
- Categorical vs Continuous -- ANOVA Test
- Categorical vs Categorical -- Chi-square Test
- For this dataset, the target variable is continuous, hence, the following two scenarios will be used.
- Continuous Target Variable vs Continuous Predictor
- Continuous Target Variable vs Categorical Predictor

Relationship Exploration: Continuous vs Continuous - Scatter Plots

- When the Target variable is continuous and the predictor is also continuous, we can visualise the relationship between the two variables using scatter plot and measure the strength of relation using a metric called Pearson's Correlation Value.

```
ContinuousCols=["ID", "Manufacturer", "Model", "Prod. year",
                "Engine volume", "Mileage"]

# Plotting scatter chart for each predictor vs the target variable
for predictor in ContinuousCols:
  car_price.plot.scatter(x=predictor, y="Price", figsize=(15,5), title=predictor+" VS "+"Price")
```



Scatter Plots Interpretation

- We can see from the Manufacturer and Model scatter plot graphs that they have an increasing trend where when one value increases, so does the other.
- The Engine volume scatter plot graphs are skewed mostly to the left indicating that the lower the Levy and Engine volume, the higher the price will be.
- The more recent the Prod. year of a car, the more expensive the car will be as it is newer.
- Lastly, the lower the mileage, the more expensive the car.

Step 13: Statistical Feature Selection (Continuous vs Continuous) using Correlation Value

- Pearson's Correlation Co-efficient is a powerful metric for doing this.
- It can simply be calculated as the co-variance between two features x and y (numerator) divided by the product of their standard deviations (denominator).
- This value can be calculated only between two numeric columns.
- Correlation between [-1,0) means inversely proportional, the scatter plot will show a downward trend.
- Correlation between (0,1] means directly proportional, the scatter plot will show an upward trend.
- Correlation near {0} means no relationship, the scatter plot will show no clear trend.
- If the correlation value between two variables is > 0.5 in magnitude, it indicates a good relationship and the sign does not matter.
- We observe the correlation between the target variable and all other predictor variable(s) to check which columns/features/predictors are actually related to the target variable in question.

```
# Calculating correlation matrix
ContinuousCols=["ID", "Price", "Manufacturer", "Model", "Prod. year",
                  "Engine volume", "Mileage"]

# Creating the correlation matrix
CorrelationData=car_price[ContinuousCols].corr()
CorrelationData
```

	ID	Price	Manufacturer	Model	Prod. year	Engine volume	Milea
ID	1.000000	0.058353	-0.034396	-0.003355	0.072030	-0.013155	0.0042
Price	0.058353	1.000000	-0.073322	0.054874	0.139292	-0.021627	-0.0191
Manufacturer	-0.034396	-0.073322	1.000000	-0.017196	-0.051567	-0.041471	0.0125
Model	-0.003355	0.054874	-0.017196	1.000000	0.064736	0.027045	-0.0081
Prod. year	0.072030	0.139292	-0.051567	0.064736	1.000000	-0.032427	-0.0640
Engine	-0.013155	-0.021627	-0.041471	0.027045	-0.032427	1.000000	-0.0062

```
# Filtering only those columns where absolute correlation > 0.1 with
# target variable
# Reduce the 0.1 threshold if no variable is selected
CorrelationData["Price"][abs(CorrelationData["Price"]) > 0.1]
```

```
Price         1.000000
Prod. year    0.139292
Name: Price, dtype: float64
```

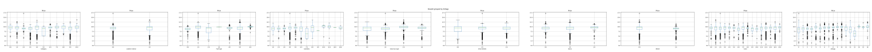
Observations from Step 13:

- Final selected Continuous column: "Prod. year"

Step 14: Relationship Exploration: Categorical vs Continuous - Box Plots

- When the target variable is continuous and the predictor variable is categorical, we analyse the relation using Box Plots.
- We measure the strength of the relation using the ANOVA Test.

```
# Box Plots for Continuous Target Variable "Price" and Categorical Predictors
CategoricalColsList=["Category", "Leather interior", "Fuel type", "Cylinders",
                     "Gear box type", "Drive wheels", "Doors", "Wheel", "Color", "Airbags"]

import matplotlib.pyplot as plt
fig, PlotCanvas=plt.subplots(nrows=1, ncols=len(CategoricalColsList),
                             figsize=(120,5))

# Creating box plots for each continuous predictor against the Target Variable
# "Price"
for PredictorCol, i in zip(CategoricalColsList, range(len(CategoricalColsList))):
  car_price.boxplot(column="Price", by=PredictorCol, figsize=(5,5), vert=True,
                   ax=PlotCanvas[i])
```



Observations from Step 14: Box Plots Interpretation

- These plots give an idea about data distribution of the Continuous Predictor on the Y-axis for each of the Categorical Predictors on the X-axis.
- If the distribution looks similar for each category, that means the Continuous variable has NO effect on the Target variable. Therefore, the variables are NOT correlated.
- If the distribution is different for each category, then these variables might be correlated to Price.
- For this data, all the Categorical predictors look like they correlate with the Target variable.

We can confirm this by looking at the results of the ANOVA test below.

Step 15: Statistical Feature Selection (Categorical vs Continuous) using ANOVA Test

- Analysis of Variance (ANOVA) is performed to check if there is any relationship between the given continuous and categorical variable.
- Assumption (H0) Null Hypothesis: There is no relation between the given variables. E.g.
- The average (mean) values of the numeric Target Variable is the same for all the groups in the Categorical Predictor Variable)

- ANOVA Test Result: Probability of H0 (Null Hypothesis being True)

```
# Defining a function to find the statistical relationship with all the categorical variables
def FunctionAnova(inpData, TargetVariable, CategoricalPredictorList):
    from scipy.stats import f_oneway

    # Creating an empty list of final selected predictors
    SelectedPredictors=[]

    print('##### ANOVA Results ##### \n')
    for predictor in CategoricalPredictorList:
        CategoryGroupLists=inpData.groupby(predictor)[TargetVariable].apply(list)
        AnovaResults = f_oneway(*CategoryGroupLists)

        # If the ANOVA P-Value is <0.05, that means we reject H0
        if (AnovaResults[1] < 0.05):
            print(predictor, 'is correlated with', TargetVariable, '| P-Value:', AnovaResults[1])
            SelectedPredictors.append(predictor)
        else:
            print(predictor, 'is NOT correlated with', TargetVariable, '| P-Value:', AnovaResults[1])

    return(SelectedPredictors)


# Calling the function to check which Categorical variables are correlated with
# Target
CategoricalList=["Category", "Leather interior", "Fuel type","Cylinders", "Gear box type",
                 "Drive wheels", "Doors", "Wheel", "Color", "Airbags"]

FunctionAnova(inpData=car_price,
              TargetVariable='Price',
              CategoricalPredictorList=CategoricalList)
```

```
##### ANOVA Results #####

Category is correlated with Price | P-Value: 1.568619676476271e-111
Leather interior is correlated with Price | P-Value: 0.0020743181243794173
Fuel type is correlated with Price | P-Value: 9.088848648088443e-259
Cylinders is correlated with Price | P-Value: 2.5635685565620376e-28
Gear box type is correlated with Price | P-Value: 4.150818363223648e-211
Drive wheels is correlated with Price | P-Value: 1.4043384460144667e-22
Doors is correlated with Price | P-Value: 0.0012120748385459942
Wheel is correlated with Price | P-Value: 3.5754350060801356e-31
Color is correlated with Price | P-Value: 6.409893629388122e-12
Airbags is correlated with Price | P-Value: 0.0
['Category',
 'Leather interior',
 'Fuel type',
 'Cylinders',
 'Gear box type',
 'Drive wheels',
 'Doors',
 'Wheel',
 'Color',
 'Airbags']
```

Observations from Step 15:

- The results from our ANOVA test confirm our visual analysis using the box plots above, however, all categorical variables EXCEPT Leather interior are correlated with our target variable.
- Final selected Categorical columns:
  - Category
  - Fuel type
  - Cylinders
  - Gear box type
  - Drive wheels
  - Doors
  - Wheel
  - Color
  - Airbags

Selecting Final Predictors/Features for building Machine Learning/AI Model

- Based on the extensive tests with Exploratory Data Analysis, the final features/predictors/columns for Machine Learning Model building can be selected as:
  - Prod. year
  - Category

- Fuel type
- Cylinders
- Gear box type
- Drive wheels
- Doors
- Wheel
- Color
- Airbags

```
SelectedColumns=["Prod. year", "Category", "Fuel type", "Cylinders", "Gear box type",
                 "Drive wheels", "Doors", "Wheel", "Color", "Airbags", "Price"]

# Selecting final columns
DataforML=car_price[SelectedColumns]
DataforML.head()
```

	Prod. year	Category	Fuel type	Cylinders	Gear box type	Drive wheels	Doors	Wheel	Color	Airbags	Pr
0	2010	4.0	2.0	6.0	0.0	0.0	1.0	0.0	12.0	12	9.497
1	2011	4.0	5.0	6.0	2.0	0.0	1.0	0.0	1.0	8	9.718
2	2006	3.0	5.0	4.0	3.0	1.0	1.0	1.0	1.0	2	9.044

```
# Saving this final data subset for reference during deployment
DataforML.to_pickle("DataforML.pkl")
```

Step 15: Data Pre-Processing for Machine Learning Model Building or Model Development

- List of steps that need to be performed on predictor variables before data can be used for Machine Learning.
- Converting each Ordinal Categorical column to numeric.
- Converting Binary Nomical Categorical columns to numeric using pd.get_dummies().
- Data Transformation (Optional): Standardisation/Normalisation/log/sqrt. Important if you are using distance based algorithms like KNN, or Neural Networks.
- Converting the Ordinal variable to numeric - in this data there is no Ordinal Categorical variable.
- Converting the Binary Nominal variable to numeric using 1/0 mapping: there is no binary nominal variable in string format in this data.

```
# Treating all the nominal variables at once using dummy variables
DataforML_Numeric=pd.get_dummies(DataforML)

# Adding Target Variable to the data
DataforML_Numeric["Price"]=car_price["Price"]

# Printing sample rows
DataforML_Numeric.head()
```

	Prod. year	Category	Fuel type	Cylinders	Gear box type	Drive wheels	Doors	Wheel	Color	Airbags	Pr
0	2010	4.0	2.0	6.0	0.0	0.0	1.0	0.0	12.0	12	9.497
1	2011	4.0	5.0	6.0	2.0	0.0	1.0	0.0	1.0	8	9.718
2	2006	3.0	5.0	4.0	3.0	1.0	1.0	1.0	1.0	2	9.044

Step 16: Machine Learning Model Development

- Splitting the data into a Training and Testing sample.
- The full data for creating the model (training data) won't be used.
- Some data is randomly selected and kept aside for checking how good the model is.
- This is known as Testing Data and the remaining data is called Training Data on which the model is built.
- Typically, 70% of data is used as Training Data and the remaining 30% is used as Testing Data.

```
# Printing all the column names for our reference
DataforML_Numeric.columns
```

```
Index(['Prod. year', 'Category', 'Fuel type', 'Cylinders', 'Gear box type',
       'Drive wheels', 'Doors', 'Wheel', 'Color', 'Airbags', 'Price'],
      dtype='object')
```

```
# Separate Target and Predictor Variables
TargetVariable="Price"
Predictors=["Prod. year", "Category", "Fuel type", "Cylinders", "Gear box type",
                 "Drive wheels", "Doors", "Wheel", "Color", "Airbags"]

x=DataforML_Numeric[Predictors].values
y=DataforML_Numeric[TargetVariable].values

# Split the data into training and testing set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
                                                    random_state=428)
```

Step 17: Standardisation/Normalisation of Data

- If the resultant accuracy of this transformation wants to be compared with the accuracy of raw data, this step can be chosen to not run (optional step).
- However, if using KNN or Neural Networks, this step becomes necessary.

```
# Standardisation of Data
from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Choose either standardisation or normalisation
# On this data, min, max, and normalisation produce better results.

# Choose between standardisation and MinMax normalisation
# PredictorScaler=StandardScaler()
PredictorScaler=MinMaxScaler()

# Storing the fit object for later reference
PredictorScalerFit=PredictorScaler.fit(x)

# Split the data into training and testing set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
                                                    random_state=42)
```

```
# Sanity check for sample data
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(13246, 10)
(13246,)
(5678, 10)
(5678,)
```

Step 18: Multiple Linear Regression Algorithm for ML/AI Model Building

```
# Multiple Linear Regression
from sklearn.linear_model import LinearRegression
RegModel = LinearRegression()

# Printing all the parameters of Linear Regression
print(RegModel)

# Creating the model on Training Data
LREG=RegModel.fit(x_train, y_train)
prediction=LREG.predict(x_test)

from sklearn import metrics
# Measuring goodness of fit in Training Data
print("R2 Value: ", metrics.r2_score(y_train, LREG.predict(x_train)))

###############################################################################

print("\n##### Model Validation and Accuracy Calculations #####")

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=x_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[("Predicted"+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults["ERR"]=100 * ((abs(
    TestingDataResults["Price"]-TestingDataResults
    ["PredictedPrice"]))/TestingDataResults["Price"])

MAPE=np.mean(TestingDataResults["ERR"])
MedianMAPE=np.median(TestingDataResults["ERR"])

Accuracy = 100 - MAPE
MedianAccuracy = 100 - MedianMAPE
print("Mean Accuracy on Test Data: ", Accuracy)
# Can be negative sometimes due to outlier
print("Median Accuracy on Test Data: ", MedianAccuracy)

# Defining a custom function to calculate accuracy
# Make sure there are no zeroes in the Target Variable if you are using MAPE
def Accuracy_Score(orig, pred):
  MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
  print("#"*70, "Accuracy: ", 100-MAPE)
  return(100-MAPE)

# Custom scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validtion function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross Validation on a given algorithm
# Passing full data x and y because the K-fold will split the data and
# automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, x, y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print("\nFinal Average Accuracy of the Model: ", round(Accuracy_Values.mean(),2))
```

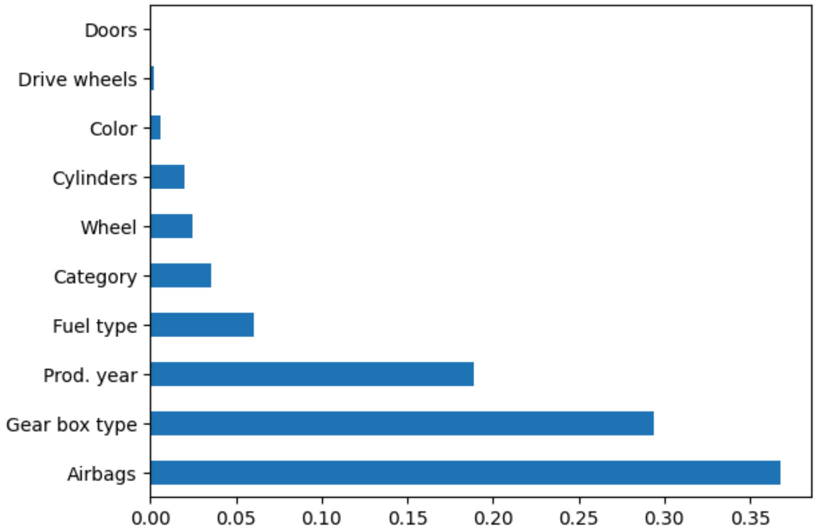
```
LinearRegression()
R2 Value:  0.12175834092153748

##### Model Validation and Accuracy Calculations #####
   Prod. year  Category  Fuel type  Cylinders  Gear box type  Drive wheels  \
0      2013.0       9.0        5.0        4.0            0.0           1.0
1      2013.0       9.0        5.0        4.0            0.0           1.0
2      2015.0       9.0        5.0        4.0            0.0           1.0
3      2001.0       3.0        5.0        4.0            0.0           1.0
4      1998.0       6.0        1.0        6.0            1.0           2.0

   Doors  Wheel  Color  Airbags     Price  PredictedPrice
0    1.0    0.0   14.0      4.0  9.359191             9.0
1    1.0    0.0   12.0     12.0  6.154858             8.0
2    1.0    0.0    1.0      4.0  9.911605             9.0
3    1.0    1.0    0.0      8.0  8.926385             7.0
4    1.0    0.0    2.0      2.0  9.842569             9.0
Mean Accuracy on Test Data:  84.2041541086751
Median Accuracy on Test Data:  90.80061034777975
###################################################################### Accuracy:  84.27148978044296
###################################################################### Accuracy:  85.28386790794927
###################################################################### Accuracy:  84.80644402948857
```

```
###################################################################### Accuracy:  84.1969423442071
###################################################################### Accuracy:  84.31824029863196
###################################################################### Accuracy:  85.30254377148091
###################################################################### Accuracy:  84.62251372460238
###################################################################### Accuracy:  84.05314163278501
###################################################################### Accuracy:  83.94039164795163
###################################################################### Accuracy:  85.33432937120605

Accuracy values for 10-fold Cross Validation:
 [84.27148978 85.28386791 84.80644403 84.19694234 84.3182403  85.30254377
 84.62251372 84.05314163 83.94039165 85.33432937]

Final Average Accuracy of the Model:  84.61
```

Decision Tree Regressor

```
# Decision Trees (Multiple if-else statements!)
from sklearn.tree import DecisionTreeRegressor
RegModel = DecisionTreeRegressor(max_depth=5,criterion='friedman_mse')
# Good Range of Max_depth = 2 to 20

# Printing all the parameters of Decision Tree
print(RegModel)

# Creating the model on Training Data
DT=RegModel.fit(x_train, y_train)
prediction=DT.predict(x_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, DT.predict(x_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(DT.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

###########################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=x_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults["ERR"]=100 * ((abs(
  TestingDataResults["Price"]-TestingDataResults['PredictedPrice']))/TestingDataResults["Price"])

MAPE=np.mean(TestingDataResults['ERR'])
MedianMAPE=np.median(TestingDataResults['ERR'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on Test Data:', Accuracy) # Can be negative sometimes due to outlier
print('Median Accuracy on Test Data:', MedianAccuracy)

# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, x , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
DecisionTreeRegressor(criterion='friedman_mse', max_depth=5)
R2 Value: 0.3790751516645221

##### Model Validation and Accuracy Calculations ##########
   Prod. year  Category  Fuel type  Cylinders  Gear box type  Drive wheels  \
0      2013.0       9.0        5.0        4.0            0.0           1.0
1      2013.0       9.0        5.0        4.0            0.0           1.0
2      2015.0       9.0        5.0        4.0            0.0           1.0
3      2001.0       3.0        5.0        4.0            0.0           1.0
4      1998.0       6.0        1.0        6.0            1.0           2.0

   Doors  Wheel  Color  Airbags     Price  PredictedPrice
0    1.0    0.0   14.0      4.0  9.359191            10.0
1    1.0    0.0   12.0     12.0  6.154858             8.0
2    1.0    0.0    1.0      4.0  9.911605            10.0
3    1.0    1.0    0.0      8.0  8.926385             7.0
4    1.0    0.0    2.0      2.0  9.842569             9.0
Mean Accuracy on Test Data: 87.48745169213821
Median Accuracy on Test Data: 94.03499077275747

Accuracy values for 10-fold Cross Validation:
 [87.77698883 88.82989368 88.05319367 87.54675747 87.70146806 88.35726781
 88.31529284 87.23417467 87.34987976 88.37155277]

Final Average Accuracy of the model: 87.95
```



Plotting/Visualising the Decision Tree

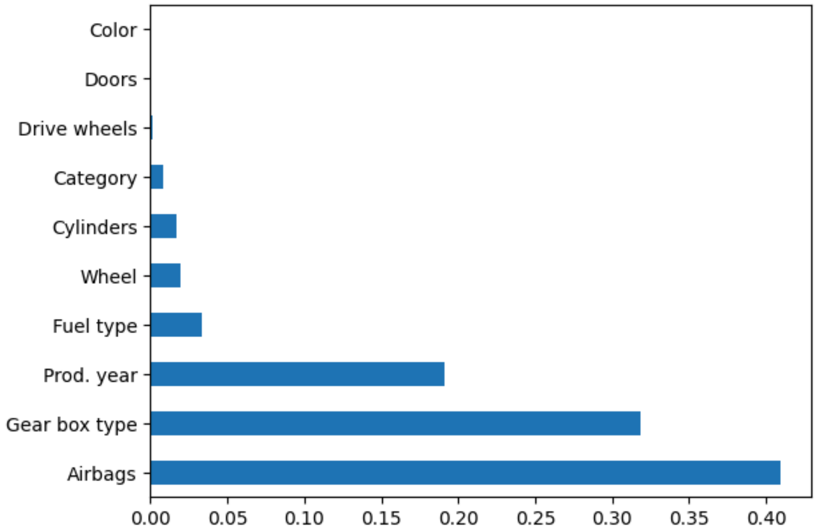
```
# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data for the 6th Decision Tree in Random Forest
dot_data = tree.export_graphviz(RegModel, out_file=None,
                                feature_names=Predictors,
                                class_names=TargetVariable)

# Printing the rules
print(dot_data)

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=2000, height=1500)

# Double click on the graph to zoom in
```

```
digraph Tree {
node [shape=box, fontname="helvetica"] ;
edge [fontname="helvetica"] ;
0 [label="Airbags <= 11.5\nfriedman_mse = 2.515\nsamples = 13246\nvalue = 9.032"]
1 [label="Airbags <= 3.5\nfriedman_mse = 1.953\nsamples = 9286\nvalue = 9.304"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label="Gear box type <= 0.5\nfriedman_mse = 2.847\nsamples = 2456\nvalue = 8.47
1 -> 2 ;
3 [label="Fuel type <= 3.0\nfriedman_mse = 3.408\nsamples = 1523\nvalue = 8.168"]
2 -> 3 ;
4 [label="Fuel type <= 1.5\nfriedman_mse = 3.204\nsamples = 651\nvalue = 7.627"] ;
3 -> 4 ;
5 [label="friedman_mse = 2.122\nsamples = 160\nvalue = 8.593"] ;
4 -> 5 ;
6 [label="friedman_mse = 3.153\nsamples = 491\nvalue = 7.312"] ;
4 -> 6 ;
7 [label="Color <= 12.5\nfriedman_mse = 3.178\nsamples = 872\nvalue = 8.572"] ;
3 -> 7 ;
8 [label="friedman_mse = 2.872\nsamples = 696\nvalue = 8.719"] ;
7 -> 8 ;
9 [label="friedman_mse = 3.97\nsamples = 176\nvalue = 7.994"] ;
7 -> 9 ;
10 [label="Prod. year <= 2004.5\nfriedman_mse = 1.539\nsamples = 933\nvalue = 8.96
2 -> 10 ;
11 [label="Fuel type <= 4.5\nfriedman_mse = 1.227\nsamples = 561\nvalue = 8.613"]
10 -> 11 ;
12 [label="friedman_mse = 1.025\nsamples = 264\nvalue = 8.911"] ;
11 -> 12 ;
13 [label="friedman_mse = 1.258\nsamples = 297\nvalue = 8.349"] ;
11 -> 13 ;
14 [label="Prod. year <= 2019.5\nfriedman_mse = 1.544\nsamples = 372\nvalue = 9.49
10 -> 14 ;
15 [label="friedman_mse = 1.335\nsamples = 370\nvalue = 9.525"] ;
14 -> 15 ;
16 [label="friedman_mse = 4.667\nsamples = 2\nvalue = 3.547"] ;
14 -> 16 ;
17 [label="Prod. year <= 2009.5\nfriedman_mse = 1.292\nsamples = 6830\nvalue = 9.6
1 -> 17 ;
18 [label="Wheel <= 0.5\nfriedman_mse = 1.967\nsamples = 1928\nvalue = 8.875"] ;
17 -> 18 ;
19 [label="Gear box type <= 1.5\nfriedman_mse = 1.177\nsamples = 1511\nvalue = 9.6
18 -> 19 ;
20 [label="friedman_mse = 1.298\nsamples = 1104\nvalue = 8.954"] ;
19 -> 20 ;
21 [label="friedman_mse = 0.671\nsamples = 407\nvalue = 9.445"] ;
19 -> 21 ;
22 [label="Category <= 3.5\nfriedman_mse = 4.082\nsamples = 417\nvalue = 8.109"] ;
18 -> 22 ;
23 [label="friedman_mse = 6.538\nsamples = 186\nvalue = 7.075"] ;
22 -> 23 ;
24 [label="friedman_mse = 0.553\nsamples = 231\nvalue = 8.941"] ;
22 -> 24 ;
25 [label="Prod. year <= 2015.5\nfriedman_mse = 0.735\nsamples = 4902\nvalue = 9.8
17 -> 25 ;
26 [label="Fuel type <= 1.5\nfriedman_mse = 0.684\nsamples = 3708\nvalue = 9.738"]
25 -> 26 ;
27 [label="friedman_mse = 0.191\nsamples = 1011\nvalue = 10.109"] ;
```

Random Forest Regressor

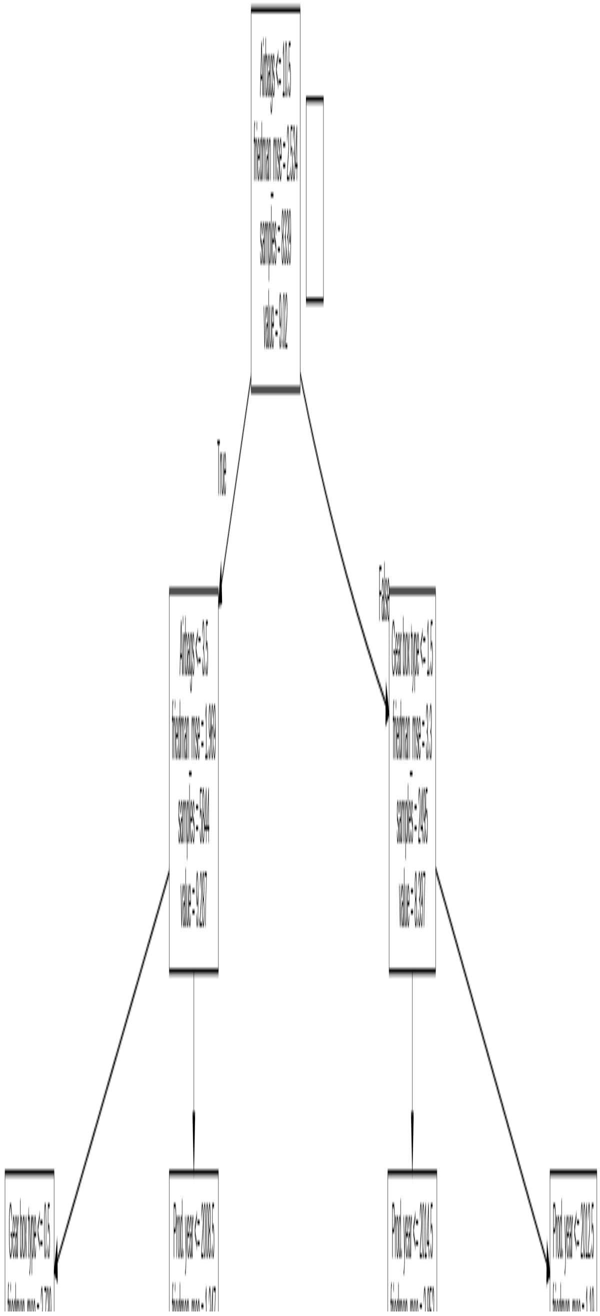
```
# Random Forest (Bagging of multiple Decision Trees)
from sklearn.ensemble import RandomForestRegressor
RegModel = RandomForestRegressor(max_depth=4, n_estimators=400,criterion='friedman_mse')
# Good range for max_depth: 2-10 and n_estimators: 100-1000

# Printing all the parameters of Random Forest
print(RegModel)

# Creating the model on Training Data
RF=RegModel.fit(x_train,y_train)
prediction=RF.predict(x_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, RF.predict(x_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(RF.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

###########################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=x_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults['ERR']=100 * ((abs(
  TestingDataResults['Price']-TestingDataResults['PredictedPrice']))/TestingDataResults['Price'])

MAPE=np.mean(TestingDataResults['ERR'])
MedianMAPE=np.median(TestingDataResults['ERR'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on Test Data:', Accuracy) # Can be negative sometimes due to outlier
print('Median Accuracy on Test Data:', MedianAccuracy)


# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, x , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
RandomForestRegressor(criterion='friedman_mse', max_depth=4, n_estimators=400)
R2 Value: 0.35258198335093216

##### Model Validation and Accuracy Calculations ##########
   Prod. year  Category  Fuel type  Cylinders  Gear box type  Drive wheels  \
0      2013.0       9.0        5.0        4.0            0.0           1.0
1      2013.0       9.0        5.0        4.0            0.0           1.0
2      2015.0       9.0        5.0        4.0            0.0           1.0
3      2001.0       3.0        5.0        4.0            0.0           1.0
4      1998.0       6.0        1.0        6.0            1.0           2.0

   Doors  Wheel  Color  Airbags     Price  PredictedPrice
0    1.0    0.0   14.0      4.0  9.359191            10.0
1    1.0    0.0   12.0     12.0  6.154858             8.0
2    1.0    0.0    1.0      4.0  9.911605            10.0
3    1.0    1.0    0.0      8.0  8.926385             8.0
4    1.0    0.0    2.0      2.0  9.842569             9.0
Mean Accuracy on Test Data: 87.04860711249134
Median Accuracy on Test Data: 93.55444315379066

Accuracy values for 10-fold Cross Validation:
 [87.34824773 88.43600811 87.75298863 87.16395641 87.24569844 87.98340488
 87.88021881 86.83277776 86.93348481 87.95448318]

Final Average Accuracy of the model: 87.55
```



Plotting One of the Decision Tree in Random Forest Regressor

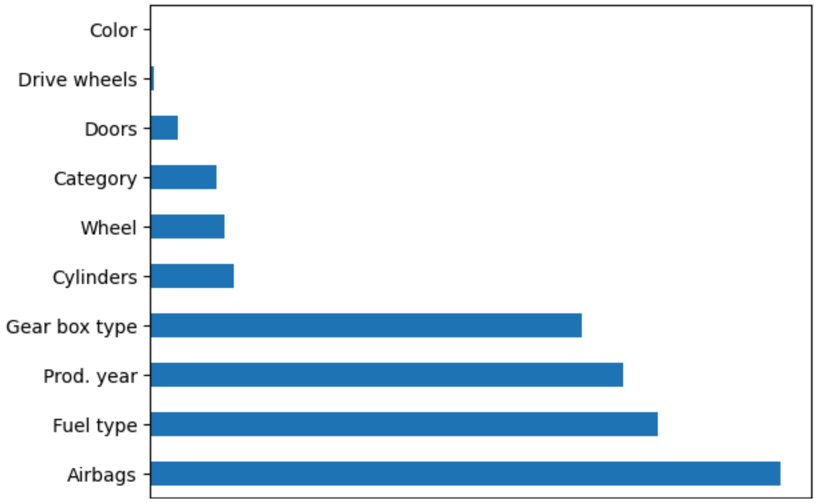
```
# Plotting a single Decision Tree from Random Forest
# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data for the 6th Decision Tree in Random Forest
dot_data = tree.export_graphviz(RegModel.estimators_[5], out_file=None, feature_names=Predictors,
                                class_names=TargetVariable)

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=2000, height=2000)
```



Step 19: AdaBoost Algorithm for ML/AI Model Building

```
# Adaboost (Boosting of Multiple Decision Trees)
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor

# Choosing Decision Tree with 6 level as the weak learner
DTR = DecisionTreeRegressor(max_depth=3)
RegModel = AdaBoostRegressor(n_estimators=100, base_estimator=DTR,
                             learning_rate=0.04)
# Printing all the parameters of Adaboost
print(RegModel)

# Creating the model on Training Data
AB=RegModel.fit(x_train, y_train)
prediction=AB.predict(x_test)

from sklearn import metrics
# Measuring goodness of fit in Training Data
print("R2 Value: ", metrics.r2_score(y_train, AB.predict(x_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(AB.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind="barh")

################################################################################
print("\n##### Model Validation and Accuracy Calculations #####")

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=x_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[("Predicted"+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults["ERR"]=100 * ((abs(TestingDataResults["Price"]-TestingDataResults["PredictedPrice"]))/TestingDataResults["Price"])

MAPE=np.mean(TestingDataResults["ERR"])
MedianMAPE=np.median(TestingDataResults["ERR"])

Accuracy = 100-MAPE
MedianAccuracy = 100-MedianMAPE
print("Mean Accuracy on Test Data: ", Accuracy)
# Can be negative sometimes due to outlier
print("Median Accuracy on Test Data: ", MedianAccuracy)

# Defining a custom function to calculate accuracy
# Make sure there are no zeroes in Target Variable if using MAPE
def Accuracy_Score(orig, pred):
  MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
  print("#"*70, "Accuracy: ", 100-MAPE)
  return(100-MAPE)

# Custom scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validtion function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross Validation on a given algorithm
# Passing full data x and y because the K-fold will split the data and
# automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, x, y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print("\nFinal Average Accuracy of the Model: ", round(Accuracy_Values.mean(),2))
```

```
AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=3),
                  learning_rate=0.04, n_estimators=100)
R2 Value:  0.19491322394559862

##### Model Validation and Accuracy Calculations #####
   Prod. year  Category  Fuel type  Cylinders  Gear box type  Drive wheels  \
0      2013.0       9.0        5.0        4.0            0.0           1.0
1      2013.0       9.0        5.0        4.0            0.0           1.0
2      2015.0       9.0        5.0        4.0            0.0           1.0
3      2001.0       3.0        5.0        4.0            0.0           1.0
4      1998.0       6.0        1.0        6.0            1.0           2.0

   Doors  Wheel  Color  Airbags     Price  PredictedPrice
0    1.0    0.0   14.0      4.0  9.359191             9.0
1    1.0    0.0   12.0     12.0  6.154858             8.0
2    1.0    0.0    1.0      4.0  9.911605             9.0
3    1.0    1.0    0.0      8.0  8.926385             8.0
4    1.0    0.0    2.0      2.0  9.842569             9.0
Mean Accuracy on Test Data:  85.52286994670716
Median Accuracy on Test Data:  90.6254806234404
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:

Accuracy values for 10-fold Cross Validation:
 [85.20555178 86.23595801 85.71058263 85.14495233 85.07186554 86.08292784
 85.45636198 84.85600358 85.75282149 85.90689126]

Final Average Accuracy of the Model:  85.54
```



XGBoost Regressor

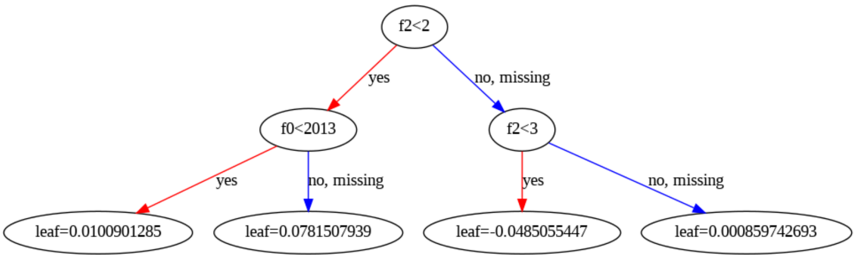
```
# Xtreme Gradient Boosting (XGBoost)
from xgboost import XGBRegressor
RegModel=XGBRegressor(max_depth=2, learning_rate=0.1, n_estimators=1000,
                      objective="reg:linear", booster="gbtree")

# Printing all the parameters of XGBoost
print(RegModel)

# Creating the model on Training Data
XGB=RegModel.fit(x_train, y_train)
prediction=XGB.predict(x_test)

from sklearn import metrics
# Measuring goodness of fit in Training Data
print("R2 Value: ", metrics.r2_score(y_train, XGB.predict(x_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(XGB.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind="barh")

###############################################################################
print("\n##### Model Validation and Accuracy Calculations #####")

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=x_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[("Predicted"+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults["ERR"]=100 * ((abs(TestingDataResults["Price"]-TestingDataResults["PredictedPrice"]))/TestingDataResults["Price"])

MAPE=np.mean(TestingDataResults["ERR"])
MedianMAPE=np.median(TestingDataResults["ERR"])

Accuracy = 100-MAPE
MedianAccuracy = 100-MedianMAPE
print("Mean Accuracy on Test Data: ", Accuracy)
# Can be negative sometimes due to outlier
print("Median Accuracy on Test Data: ", MedianAccuracy)

# Defining a custom function to calculate accuracy
# Make sure there are no zeroes in Target Variable if using MAPE
def Accuracy_Score(orig, pred):
  MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
  print("#"*70, "Accuracy: ", 100-MAPE)
  return(100-MAPE)

# Custom scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validtion function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross Validation on a given algorithm
# Passing full data x and y because the K-fold will split the data and
# automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, x, y, cv=10, scoring=custom_Scoring)
print("\nAccuracy Values for 10-Fold Cross Validation:\n", Accuracy_Values)
print("\nFinal Average Accuracy of the Model: ", round(Accuracy_Values.mean(),2))
```

```
XGBRegressor(base_score=None, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=2, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=1000, n_jobs=None,
             num_parallel_tree=None, objective='reg:linear', ...)
R2 Value:  0.42652269503769846

##### Model Validation and Accuracy Calculations #####
   Prod. year  Category  Fuel type  Cylinders  Gear box type  Drive wheels  \
0      2013.0       9.0        5.0        4.0            0.0           1.0
1      2013.0       9.0        5.0        4.0            0.0           1.0
2      2015.0       9.0        5.0        4.0            0.0           1.0
3      2001.0       3.0        5.0        4.0            0.0           1.0
4      1998.0       6.0        1.0        6.0            1.0           2.0

   Doors  Wheel  Color  Airbags     Price  PredictedPrice
0    1.0    0.0   14.0      4.0  9.359191            10.0
1    1.0    0.0   12.0     12.0  6.154858             8.0
2    1.0    0.0    1.0      4.0  9.911605            10.0
3    1.0    1.0    0.0      8.0  8.926385             8.0
4    1.0    0.0    2.0      2.0  9.842569            10.0
Mean Accuracy on Test Data:  87.88597745598847
Median Accuracy on Test Data:  94.19228683381888
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:
###################################################################### Accuracy:

Accuracy Values for 10-Fold Cross Validation:
 [88.12561858 89.24320232 88.4723761  87.95826491 88.11027351 88.9256622
 88.67314792 87.96336797 87.86094942 88.78676296]

Final Average Accuracy of the Model:  88.41
```

Plotting a Single Decision Tree out of XGBoost

```
# Plotting a Single Decision Tree out of XGBoost
from xgboost import plot_tree
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(20,8))
plot_tree(XGB, num_trees=10, ax=ax)
```

```
<Axes: >
```



K-Nearest Neighbour (KNN)

```
# K-Nearest Neighbour (KNN)
from sklearn.neighbors import KNeighborsRegressor
RegModel = KNeighborsRegressor(n_neighbors=3)

# Printing all the parameters of KNN
print(RegModel)

# Creating the model on Training Data
KNN=RegModel.fit(x_train, y_train)
prediction=KNN.predict(x_test)

from sklearn import metrics
# Measuring goodness of fit in Training Data
print("R2 Value: ", metrics.r2_score(y_train, KNN.predict(x_train)))

#Plotting the feature importance for Top 10 most important columns
# The variable importance chart is not available for KNN

###########################################################################
print("\n##### Model Validation and Accuracy Calculations #####")

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=x_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[("Predicted"+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults["ERR"]=100 * ((abs(TestingDataResults["Price"]-TestingDataResults["PredictedPrice"]))/TestingDataResults["Price"])

MAPE=np.mean(TestingDataResults["ERR"])
MedianMAPE=np.median(TestingDataResults["ERR"])

Accuracy = 100-MAPE
MedianAccuracy = 100-MedianMAPE
print("Mean Accuracy on Test Data: ", Accuracy)
# Can be negative sometimes due to outlier
print("Median Accuracy on Test Data: ", MedianAccuracy)

# Defining a custom function to calculate accuracy
# Make sure there are no zeroes in Target Variable if using MAPE
def Accuracy_Score(orig, pred):
  MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
  print("#"*70, "Accuracy: ", 100-MAPE)
  return(100-MAPE)

# Custom scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validtion function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross Validation on a given algorithm
# Passing full data x and y because the K-fold will split the data and
# automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, x, y, cv=10, scoring=custom_Scoring)
print("\nAccuracy Values for 10-Fold Cross Validation:\n", Accuracy_Values)
print("\nFinal Average Accuracy of the Model: ", round(Accuracy_Values.mean(),2))
```

```
KNeighborsRegressor(n_neighbors=3)
R2 Value:  0.6103834591579823

##### Model Validation and Accuracy Calculations #####
   Prod. year  Category  Fuel type  Cylinders  Gear box type  Drive wheels  \
0      2013.0       9.0        5.0        4.0            0.0           1.0
1      2013.0       9.0        5.0        4.0            0.0           1.0
2      2015.0       9.0        5.0        4.0            0.0           1.0
3      2001.0       3.0        5.0        4.0            0.0           1.0
4      1998.0       6.0        1.0        6.0            1.0           2.0

   Doors  Wheel  Color  Airbags     Price  PredictedPrice
0    1.0    0.0   14.0      4.0  9.359191            10.0
1    1.0    0.0   12.0     12.0  6.154858             8.0
2    1.0    0.0    1.0      4.0  9.911605            10.0
3    1.0    1.0    0.0      8.0  8.926385             9.0
4    1.0    0.0    2.0      2.0  9.842569             9.0
Mean Accuracy on Test Data:  88.60944178588957
Median Accuracy on Test Data:  94.92585950261103
###################################################################### Accuracy:  89.39421178229804
###################################################################### Accuracy:  90.03191268608248
###################################################################### Accuracy:  89.2659713734012
```

```
##################################################################### Accuracy:  89.20342157531974
```

Support Vector Machine (SVM) Regressor

```
##################################################################### Accuracy:  89.67785038718301
```

```
# Support Vector Machines (SVM)
from sklearn import svm
RegModel = svm.SVR(C=50, kernel="rbf", gamma=0.01)

# Printing all the parameters
print(RegModel)

# Creating the model on Training Data
SVM=RegModel.fit(x_train, y_train)
prediction=SVM.predict(x_test)

from sklearn import metrics
# Measuring goodness of fit in Training Data
print("R2 Value: ", metrics.r2_score(y_train, SVM.predict(x_train)))

# Plotting the feature importance for Top 10 most important columns
# The built-in attribute SVM.coef_ works only for linear kernel
%matplotlib inline
```