2. Exploratory Data Analysis

September 24, 2024

1 Introduction

All machine learning models or algorithms work on data.

Machine learning model results in inaccurate conclusions when:

- The data is incorrect
- The data contains abnormal values
- The data contains duplicate rows

The data has to be analysed and clean before it is given to the Machine Learning model. This is called Exploratory Data Analysis (EDA).

Advantages of Exploratory Data Analysis

1. Provides better understanding of data.
2. Possible to identify the relationships between various data elements (attributes).
3. Possible to recognnize patterns in the data
4. Possible to clean, modify, delete the data, thus making the data useful for the Machine Learning model.

Generally, the following tasks are done as part of EDA:

- Know the initial details of the data
- Modify or remove unwanted data
- Retrieve the required attributes/variables
- Plotting graphs

2 Dataset

A dataset is a rows of data that is being analysed and used by a Machine Learning model.

https://www.kaggle.com/ is a world renowned company to find various Datasets with coding examples on how to use these datasets.

3 Cars.csv

[266]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

3.1 Reading the data from the file

[267]:
```
cars_df = pd.read_csv("./datasets/cars.csv")
```

3.2 Knowing the Initial Details About the Data

3.2.1 Get the number of rows and columsn present in the given dataset

[268]:
```
cars_df.shape
```

[268]:
```
(429, 15)
```

3.2.2 Get the column names and their data types

[269]:
```
cars_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 429 entries, 0 to 428
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Make         429 non-null    object 
 1   Model        429 non-null    object 
 2   Type         429 non-null    object 
 3   Origin       429 non-null    object 
 4   DriveTrain   429 non-null    object 
 5   MSRP         429 non-null    object 
 6   Invoice      429 non-null    object 
 7   EngineSize   429 non-null    float64
 8   Cylinders    427 non-null    float64
 9   Horsepower   429 non-null    int64  
 10  MPG_City     429 non-null    int64  
 11  MPG_Highway  429 non-null    int64  
 12  Weight       429 non-null    int64  
 13  Wheelbase    429 non-null    int64  
 14  Length       429 non-null    int64  
dtypes: float64(2), int64(6), object(7)
memory usage: 50.4+ KB
```

There are 429 rows and 15 columns in the dataset.

The column Cylinders has 2 null values.

3.2.3 To display the first 5 rows of the dataset

[270]:
```
cars_df.head()
```

[270]:

	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	\
0	Acura	MDX	SUV	Asia	All	$36,945	$33,337	

```
1  Acura  RSX Type S 2dr  Sedan   Asia      Front  $23,820   $21,761
2  Acura         TSX 4dr  Sedan   Asia      Front  $26,990   $24,647
3  Acura          TL 4dr  Sedan   Asia      Front  $33,195   $30,299
4  Acura      3.5 RL 4dr  Sedan   Asia      Front  $43,755   $39,014

   EngineSize  Cylinders  Horsepower  MPG_City  MPG_Highway  Weight  \
0         3.5        6.0         265        17           23    4451
1         2.0        4.0         200        24           31    2778
2         2.4        4.0         200        22           29    3230
3         3.2        6.0         270        20           28    3575
4         3.5        6.0         225        18           24    3880

   Wheelbase  Length
0        106     189
1        101     172
2        105     183
3        108     186
4        115     197
```

3.2.4 To display the last 5 rows of the dataset

[271]:
```
cars_df.tail()
```

[271]:
```
      Make                    Model   Type  Origin DriveTrain      MSRP  \
424  Volvo  C70 HPT convertible 2dr  Sedan  Europe      Front  $42,565
425  Volvo                S80 T6 4dr  Sedan  Europe      Front  $45,210
426  Volvo                       V40  Wagon  Europe      Front  $26,135
427  Volvo                      XC70  Wagon  Europe        All  $35,145
428  Volvo                      XC70  Wagon  Europe        All  $35,145

      Invoice  EngineSize  Cylinders  Horsepower  MPG_City  MPG_Highway  \
424  $40,083         2.3        5.0         242        20           26
425  $42,573         2.9        6.0         268        19           26
426  $24,641         1.9        4.0         170        22           29
427  $33,112         2.5        5.0         208        20           27
428  $33,112         2.5        5.0         208        20           27

     Weight  Wheelbase  Length
424    3450        105     186
425    3653        110     190
426    2822        101     180
427    3823        109     186
428    3823        109     186
```

3.2.5 To get the name of the columns

[272]:
```
cars_df.columns
```

[272]:
```
Index(['Make', 'Model', 'Type', 'Origin', 'DriveTrain', 'MSRP', 'Invoice',
       'EngineSize', 'Cylinders', 'Horsepower', 'MPG_City', 'MPG_Highway',
       'Weight', 'Wheelbase', 'Length'],
      dtype='object')
```

3.2.6 Rename the columns

[273]:
```
cars_df.rename(columns={'MSRP': 'MRP', 'MPG_City': 'Mileage_City',␣
↪'MPG_Highway':'Mileage_Highway'}, inplace=True)
```

[274]:
```
cars_df.columns
```

[274]:
```
Index(['Make', 'Model', 'Type', 'Origin', 'DriveTrain', 'MRP', 'Invoice',
       'EngineSize', 'Cylinders', 'Horsepower', 'Mileage_City',
       'Mileage_Highway', 'Weight', 'Wheelbase', 'Length'],
      dtype='object')
```

3.2.7 Getting Statistical Information

[275]:
```
cars_df.describe()
```

[275]:

	EngineSize	Cylinders	Horsepower	Mileage_City	Mileage_Highway	\
count	429.000000	427.000000	429.000000	429.000000	429.000000	
mean	3.195105	5.805621	215.867133	20.060606	26.843823	
std	1.107810	1.557103	71.753072	5.232095	5.734495	
min	1.300000	3.000000	73.000000	10.000000	12.000000	
25%	2.400000	4.000000	165.000000	17.000000	24.000000	
50%	3.000000	6.000000	210.000000	19.000000	26.000000	
75%	3.900000	6.000000	255.000000	21.000000	29.000000	
max	8.300000	12.000000	500.000000	60.000000	66.000000	

	Weight	Wheelbase	Length
count	429.000000	429.000000	429.000000
mean	3578.524476	108.156177	186.361305
std	758.188346	8.302198	14.341219
min	1850.000000	89.000000	143.000000
25%	3105.000000	103.000000	178.000000
50%	3476.000000	107.000000	187.000000
75%	3977.000000	112.000000	194.000000
max	7190.000000	144.000000	238.000000

To know the relationship between the numerical columns

[276]:
```
cars_df.select_dtypes(exclude=object).corr()
```

[276]:

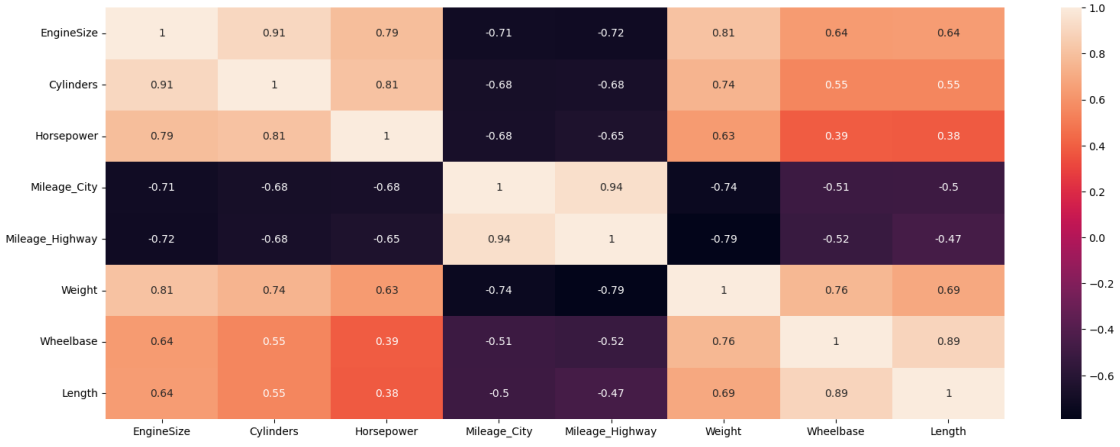
	EngineSize	Cylinders	Horsepower	Mileage_City	Mileage_Highway	Weight	Wheelbase	Length
EngineSize	1.000000	0.908058	0.787222	-0.709127	-0.717010	0.806922	0.636066	0.637191
Cylinders	0.908058	1.000000	0.810207	-0.684170	-0.675917	0.741498	0.546430	0.547645
Horsepower	0.787222	0.810207	1.000000	-0.676687	-0.647193	0.630627	0.387362	0.381555
Mileage_City	-0.709127	-0.684170	-0.676687	1.000000	0.941019	-0.737885	-0.507280	-0.501525
Mileage_Highway	-0.717010	-0.675917	-0.647193	0.941019	1.000000	-0.790872	-0.524647	-0.466093
Weight	0.806922	0.741498	0.630627	-0.737885	-0.790872	1.000000	0.760678	0.689917
Wheelbase	0.636066	0.546430	0.387362	-0.507280	-0.524647	0.760678	1.000000	0.889177
Length	0.637191	0.547645	0.381555	-0.501525	-0.466093	0.689917	0.889177	1.000000

[277]:

```
plt.figure(figsize=(19, 7))

sns.heatmap(cars_df.select_dtypes(exclude=object).corr(), annot=True)

plt.show()
```



3.3 Modify or remove unwanted Data

3.3.1 To drop a single column

[278]:
```
cars_df.columns
```

[278]:
```
Index(['Make', 'Model', 'Type', 'Origin', 'DriveTrain', 'MRP', 'Invoice',
       'EngineSize', 'Cylinders', 'Horsepower', 'Mileage_City',
       'Mileage_Highway', 'Weight', 'Wheelbase', 'Length'],
      dtype='object')
```

[279]:
```
cars_df.drop('Model', axis=1, inplace=True)
```

[280]:
```
cars_df.columns
```

[280]:
```
Index(['Make', 'Type', 'Origin', 'DriveTrain', 'MRP', 'Invoice', 'EngineSize',
       'Cylinders', 'Horsepower', 'Mileage_City', 'Mileage_Highway', 'Weight',
       'Wheelbase', 'Length'],
      dtype='object')
```

3.3.2 To drop multiple columns

[281]:
```
cars_df.drop(['MRP', 'Invoice'], axis=1, inplace=True)
```

[282]:
```
cars_df.columns
```

[282]:
```
Index(['Make', 'Type', 'Origin', 'DriveTrain', 'EngineSize', 'Cylinders',
       'Horsepower', 'Mileage_City', 'Mileage_Highway', 'Weight', 'Wheelbase',
       'Length'],
      dtype='object')
```

3.3.3 Check for duplicate rows

[283]:
```
cars_df.duplicated()
```

[283]:
```
0      False
1      False
2      False
3      False
4      False
       …
424    False
425    False
426    False
427    False
428     True
Length: 429, dtype: bool
```

[284]:
```
cars_df.duplicated().sum()
```

[284]:
```
22
```

3.3.4 Drop the duplicate rows

[285]:
```
cars_df.drop_duplicates(inplace=True)
```

[286]:
```
cars_df.duplicated().sum()
```

[286]:
```
0
```

3.3.5 Knowing the count of rows with null values

[287]:
```
cars_df.isnull().sum()
```

[287]:
```
Make              0
Type              0
Origin            0
DriveTrain        0
EngineSize        0
Cylinders         2
Horsepower        0
Mileage_City      0
Mileage_Highway   0
Weight            0
Wheelbase         0
Length            0
dtype: int64
```

[288]:
```
cars_df.isna().sum()
```

[288]:
```
Make              0
Type              0
Origin            0
DriveTrain        0
EngineSize        0
Cylinders         2
Horsepower        0
Mileage_City      0
Mileage_Highway   0
Weight            0
Wheelbase         0
Length            0
dtype: int64
```

3.3.6 Droping the rows with null values

[289]:
```
#cars_df.dropna(inplace=True)
```

[290]:
```
cars_df.shape
```

[290]:
```
(407, 12)
```

3.3.7 Fill the null value with mean/median value

[291]:
```
cars_df.isna().sum()
```

[291]:
```
Make              0
Type              0
Origin            0
DriveTrain        0
EngineSize        0
Cylinders         2
Horsepower        0
Mileage_City      0
Mileage_Highway   0
Weight            0
Wheelbase         0
Length            0
dtype: int64
```

[292]:
```
cars_df['Cylinders'].mean()
```

[292]:
```
5.817283950617284
```

[293]:
```
cars_df['Cylinders'].median()
```

[293]:
```
6.0
```

[294]:
```
cars_df['Cylinders'].fillna(cars_df['Cylinders'].mean(), inplace=True)
```

```
C:\Users\deepa\AppData\Local\Temp\ipykernel_21940\3381060714.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cars_df['Cylinders'].fillna(cars_df['Cylinders'].mean(), inplace=True)
```

[295]:
```
cars_df.isna().sum()
```

[295]:
```
Make              0
Type              0
Origin            0
DriveTrain        0
EngineSize        0
Cylinders         0
Horsepower        0
Mileage_City      0
Mileage_Highway   0
Weight            0
Wheelbase         0
Length            0
dtype: int64
```

3.3.8 Datatype Conversion

[296]:
```
cars_df.dtypes
```

[296]:
```
Make               object
Type               object
Origin             object
DriveTrain         object
EngineSize        float64
Cylinders         float64
Horsepower          int64
Mileage_City        int64
Mileage_Highway     int64
Weight              int64
Wheelbase           int64
Length              int64
dtype: object
```

[297]:
```
cars_df['Cylinders'] = cars_df['Cylinders'].astype(np.int64)
```

[298]:
```
cars_df.dtypes
```

[298]:
```
Make               object
Type               object
Origin             object
DriveTrain         object
EngineSize        float64
Cylinders           int64
Horsepower          int64
Mileage_City        int64
Mileage_Highway     int64
```

```
Weight                int64
Wheelbase             int64
Length                int64
dtype: object
```

3.3.9 Modifying the Data in a column

Retrieving the data from a column

[299]:
```
cars_df.head()
```

[299]:
```
    Make   Type Origin DriveTrain  EngineSize  Cylinders  Horsepower  \
0  Acura    SUV   Asia        All         3.5          6         265
1  Acura  Sedan   Asia      Front         2.0          4         200
2  Acura  Sedan   Asia      Front         2.4          4         200
3  Acura  Sedan   Asia      Front         3.2          6         270
4  Acura  Sedan   Asia      Front         3.5          6         225

   Mileage_City  Mileage_Highway  Weight  Wheelbase  Length
0            17               23    4451        106     189
1            24               31    2778        101     172
2            22               29    3230        105     183
3            20               28    3575        108     186
4            18               24    3880        115     197
```

[300]:
```
cars_df['Make']
```

[300]:
```
0      Acura
1      Acura
2      Acura
3      Acura
4      Acura
       …
423    Volvo
424    Volvo
425    Volvo
426    Volvo
427    Volvo
Name: Make, Length: 407, dtype: object
```

Retrieving the data from multiple columns

[301]:
```
cars_df[['Make', 'Type', 'DriveTrain']]
```

[301]:
```
      Make   Type DriveTrain
0    Acura    SUV        All
1    Acura  Sedan      Front
2    Acura  Sedan      Front
3    Acura  Sedan      Front
```

```
4    Acura  Sedan      Front
..     …    …        …
423  Volvo  Sedan      Front
424  Volvo  Sedan      Front
425  Volvo  Sedan      Front
426  Volvo  Wagon      Front
427  Volvo  Wagon        All

[407 rows x 3 columns]
```

Get the distinct values of a column

[302]:
```
cars_df['Origin'].unique()
```

[302]:
```
array(['Asia', 'Europe', 'USA'], dtype=object)
```

3.4 Retrieving the Data

[303]:
```
cars_df.head()
```

[303]:
```
    Make    Type Origin DriveTrain  EngineSize  Cylinders  Horsepower  \
0  Acura     SUV   Asia        All         3.5          6         265
1  Acura  Sedan   Asia      Front         2.0          4         200
2  Acura  Sedan   Asia      Front         2.4          4         200
3  Acura  Sedan   Asia      Front         3.2          6         270
4  Acura  Sedan   Asia      Front         3.5          6         225

   Mileage_City  Mileage_Highway  Weight  Wheelbase  Length
0            17               23    4451        106     189
1            24               31    2778        101     172
2            22               29    3230        105     183
3            20               28    3575        108     186
4            18               24    3880        115     197
```

[304]:
```
cars_df.tail()
```

[304]:
```
      Make    Type  Origin DriveTrain  EngineSize  Cylinders  Horsepower  \
423  Volvo  Sedan  Europe      Front         2.4          5         197
424  Volvo  Sedan  Europe      Front         2.3          5         242
425  Volvo  Sedan  Europe      Front         2.9          6         268
426  Volvo  Wagon  Europe      Front         1.9          4         170
427  Volvo  Wagon  Europe        All         2.5          5         208

     Mileage_City  Mileage_Highway  Weight  Wheelbase  Length
423            21               28    3450        105     186
424            20               26    3450        105     186
425            19               26    3653        110     190
426            22               29    2822        101     180
```

```
427            20                27     3823         109      186
```

Using *loc* method The *loc* method is used for label based indexing.

[305]:
```
cars_df.loc[0, 'Make']
```

[305]:
```
'Acura'
```

[306]:
```
cars_df.loc[0:2, 'Make': 'Type']
```

[306]:
```
    Make   Type
0  Acura    SUV
1  Acura  Sedan
2  Acura  Sedan
```

[307]:
```
cars_df.loc[[2, 5, 423], ['Make', 'Mileage_City']]
```

[307]:
```
      Make  Mileage_City
2    Acura            22
5    Acura            18
423  Volvo            21
```

Using *iloc* method

[308]:
```
cars_df.iloc[0, 0]
```

[308]:
```
'Acura'
```

[309]:
```
cars_df.iloc[0:2, 0:3]
```

[309]:
```
    Make   Type Origin
0  Acura    SUV   Asia
1  Acura  Sedan   Asia
```

[310]:
```
cars_df.iloc[[2, 5], [0, 1, 5]]
```

[310]:
```
    Make   Type  Cylinders
2  Acura  Sedan          4
5  Acura  Sedan          6
```

[311]:
```
cars_df.loc[cars_df['Origin'] == 'Asia', :]
```

[311]:
```
       Make    Type Origin DriveTrain  EngineSize  Cylinders  Horsepower  \
0     Acura     SUV   Asia        All         3.5          6         265
1     Acura   Sedan   Asia      Front         2.0          4         200
2     Acura   Sedan   Asia      Front         2.4          4         200
3     Acura   Sedan   Asia      Front         3.2          6         270
4     Acura   Sedan   Asia      Front         3.5          6         225
..      …     …    …        …          …         …          …
```

```
396  Toyota  Sports   Asia       Rear         1.8          4         138
397  Toyota   Truck   Asia       Rear         2.4          4         142
398  Toyota   Truck   Asia       Rear         3.4          6         190
399  Toyota   Truck   Asia        All         3.4          6         190
400  Toyota   Wagon   Asia      Front         1.8          4         130

     Mileage_City  Mileage_Highway  Weight  Wheelbase  Length
0              17               23    4451        106     189
1              24               31    2778        101     172
2              22               29    3230        105     183
3              20               28    3575        108     186
4              18               24    3880        115     197
..            …                …      …         …       …
396            26               32    2195         97     153
397            22               27    2750        103     191
398            16               18    3925        128     218
399            14               17    4435        128     218
400            29               36    2679        102     171

[152 rows x 12 columns]
```

[312]:
```
cars_df.loc[cars_df['Mileage_City'] > 50, :]
```

[312]:
```
       Make    Type Origin DriveTrain  EngineSize  Cylinders  Horsepower  \
150   Honda  Hybrid   Asia      Front         2.0          3          73
373  Toyota  Hybrid   Asia      Front         1.5          4         110

     Mileage_City  Mileage_Highway  Weight  Wheelbase  Length
150            60               66    1850         95     155
373            59               51    2890        106     175
```

Get the data based on the column dtypes.

[313]:
```
cars_df.dtypes
```

[313]:
```
Make               object
Type               object
Origin             object
DriveTrain         object
EngineSize        float64
Cylinders           int64
Horsepower          int64
Mileage_City        int64
Mileage_Highway     int64
Weight              int64
Wheelbase           int64
Length              int64
dtype: object
```

[314]:
```
cars_num_df = cars_df.select_dtypes(include=np.number)
```

[315]:
```
cars_num_df.dtypes
```

[315]:
```
EngineSize         float64
Cylinders            int64
Horsepower           int64
Mileage_City         int64
Mileage_Highway      int64
Weight               int64
Wheelbase            int64
Length               int64
dtype: object
```

[316]:
```
num_columns = cars_df.select_dtypes(include=np.number).columns

num_columns
```

[316]:
```
Index(['EngineSize', 'Cylinders', 'Horsepower', 'Mileage_City',
       'Mileage_Highway', 'Weight', 'Wheelbase', 'Length'],
      dtype='object')
```

[317]:
```
cars_cat_df = cars_df.select_dtypes(exclude=np.number)
```

[318]:
```
cars_cat_df.dtypes
```

[318]:
```
Make          object
Type          object
Origin        object
DriveTrain    object
dtype: object
```

3.5 Handling non-numeric Data

The dataset main contain numerical and/or categorical variables.

Most of the machine learning algorithms are designed to work on numerical data.

Types of encoding:

- n-1 dummy encoding
- One-hot encoding
- Label encoding
- Ordinal encoding
- Frequency encoding
- Target encoding

[319]:
```
cars_cat_df.head()
```

[319]:
	Make	Type	Origin	DriveTrain
0	Acura	SUV	Asia	All
1	Acura	Sedan	Asia	Front
2	Acura	Sedan	Asia	Front
3	Acura	Sedan	Asia	Front
4	Acura	Sedan	Asia	Front

3.5.1 One-hot encoding

For a categorical variable that can take k values, k dummy variables are created.

Each category is converted into one column with values '0' and '1', depending on the presence or absence of the category in the corresponding observatio.n

[320]:
```
np.sort(cars_cat_df['Type'].unique())
```

[320]:
```
array(['Hybrid', 'SUV', 'Sedan', 'Sports', 'Truck', 'Wagon'], dtype=object)
```

[321]:
```
from sklearn.preprocessing import OneHotEncoder
```

[322]:
```
encode = OneHotEncoder()
```

[323]:
```
encode.fit_transform(cars_cat_df[['Type']]).toarray()
```

[323]:
```
array([[0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       …,
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 1.]])
```

[324]:
```
cars_encode_df = pd.DataFrame(encode.fit_transform(cars_cat_df[['Type']]).
 ↪toarray(),
                        columns=np.sort(cars_cat_df['Type'].unique()))
```

[325]:
```
cars_encode_df
```

[325]:
	Hybrid	SUV	Sedan	Sports	Truck	Wagon
0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.0	0.0	1.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	0.0	0.0
4	0.0	0.0	1.0	0.0	0.0	0.0
..	…	…	…	…	…	…
402	0.0	0.0	1.0	0.0	0.0	0.0
403	0.0	0.0	1.0	0.0	0.0	0.0
404	0.0	0.0	1.0	0.0	0.0	0.0
405	0.0	0.0	0.0	0.0	0.0	1.0

```
406     0.0  0.0    0.0     0.0    0.0    1.0

[407 rows x 6 columns]
```

3.5.2 Label encoding

The LabelEncoder considers the levels in a categorical variable by alphabetical order for encodin.

LabelEncoder labels different levels of the categorical variable with values between 0 and n-1, where 'n' is number of distinct categorie.sg

[326]:
```
np.sort(cars_cat_df['DriveTrain'].unique())
```

[326]:
```
array(['All', 'Front', 'Rear'], dtype=object)
```

[327]:
```
from sklearn.preprocessing import LabelEncoder
```

[328]:
```
label_encoder = LabelEncoder()
```

[329]:
```
label_encoder.fit_transform(cars_cat_df['DriveTrain'])
```

[329]:
```
array([0, 1, 1, 1, 1, 1, 2, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 2, 2, 0, 0, 2, 0, 2, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 2, 2, 2, 2, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2,
       1, 2, 0, 2, 1, 1, 1, 1, 0, 2, 2, 2, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 0,
       1, 2, 2, 2, 0, 0, 1, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2,
       2, 2, 0, 1, 2, 2, 2, 2, 1, 1, 0, 1, 1, 1, 1, 2, 2, 2, 2, 0, 0, 0,
       2, 2, 2, 0, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2,
       0, 1, 1, 2, 1, 2, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 0, 0, 2, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 2, 2, 0, 2, 0, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1,
       0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 2, 2, 2, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0])
```

3.6 Data Normalization

A technique used to transform the data into a common scale. Since the features have various ranges, it becomes a necessary step in data preprocessing while using machine learning algorithms

[330]:
```
cars_num_df.describe()
```

[330]:
```
       EngineSize   Cylinders  Horsepower  Mileage_City  Mileage_Highway  \
count  407.000000  407.000000  407.000000    407.000000       407.000000
mean     3.207371    5.813268  217.206388     19.985258        26.697789
std      1.112400    1.547132   72.558430      5.271896         5.754807
min      1.300000    3.000000   73.000000     10.000000        12.000000
25%      2.400000    4.000000  165.000000     17.000000        24.000000
50%      3.000000    6.000000  210.000000     19.000000        26.000000
75%      3.850000    6.000000  259.000000     21.000000        29.000000
max      8.300000   12.000000  500.000000     60.000000        66.000000

             Weight  Wheelbase      Length
count   407.000000  407.00000  407.000000
mean   3594.788698  108.19656  186.253071
std     762.405497    8.36134   14.330110
min    1850.000000   89.00000  143.000000
25%    3119.000000  103.00000  178.000000
50%    3477.000000  107.00000  187.000000
75%    3982.000000  112.00000  194.000000
max    7190.000000  144.00000  238.000000
```

3.6.1 Standard Scaler

Standardization transforms the data such that the data has mean 0 and unit variance.

[331]:
```
from sklearn.preprocessing import StandardScaler
```

[332]:
```
scaler = StandardScaler()
```

[333]:
```
cars_num_df = pd.DataFrame(scaler.fit_transform(cars_num_df),
                           columns=cars_num_df.columns)
```

[334]:
```
cars_num_df.describe()
```

[334]:
```
         EngineSize     Cylinders    Horsepower  Mileage_City  \
count  4.070000e+02  4.070000e+02  4.070000e+02  4.070000e+02
mean   2.094966e-16  3.229740e-16  1.047483e-16  1.658515e-16
std    1.001231e+00  1.001231e+00  1.001231e+00  1.001231e+00
min   -1.716756e+00 -1.820614e+00 -1.989898e+00 -1.896386e+00
25%   -7.266854e-01 -1.173461e+00 -7.203938e-01 -5.669558e-01
50%   -1.866471e-01  1.208443e-01 -9.944065e-02 -1.871188e-01
75%    5.784071e-01  1.208443e-01  5.767083e-01  1.927183e-01
max    4.583691e+00  4.003762e+00  3.902257e+00  7.599541e+00

       Mileage_Highway        Weight     Wheelbase        Length
count     4.070000e+02  4.070000e+02  4.070000e+02  4.070000e+02
mean     -1.767628e-16 -9.601929e-17 -2.793288e-16 -1.047483e-16
std       1.001231e+00  1.001231e+00  1.001231e+00  1.001231e+00
min      -2.557145e+00 -2.291348e+00 -2.298697e+00 -3.022050e+00
```

```
25%      -4.693657e-01 -6.248306e-01 -6.222634e-01 -5.766340e-01
50%      -1.214024e-01 -1.546863e-01 -1.432824e-01  5.218718e-02
75%       4.005425e-01  5.085061e-01  4.554439e-01  5.412703e-01
max       6.837863e+00  4.721419e+00  4.287292e+00  3.615507e+00
```

3.6.2 MinMax Scaler

After normalization, all values will be between 0 and 1.

[335]:
```
from sklearn.preprocessing import MinMaxScaler
```

[336]:
```
scaler = MinMaxScaler()
```

[337]:
```
cars_num_df = pd.DataFrame(scaler.fit_transform(cars_num_df),
                           columns=cars_num_df.columns)
```

[338]:
```
cars_num_df.describe()
```

[338]:

	EngineSize	Cylinders	Horsepower	Mileage_City	Mileage_Highway	\
count	407.000000	407.000000	407.000000	407.000000	407.000000	
mean	0.272482	0.312585	0.337720	0.199705	0.272181	
std	0.158914	0.171904	0.169926	0.105438	0.106570	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.157143	0.111111	0.215457	0.140000	0.222222	
50%	0.242857	0.333333	0.320843	0.180000	0.259259	
75%	0.364286	0.333333	0.435597	0.220000	0.314815	
max	1.000000	1.000000	1.000000	1.000000	1.000000	

	Weight	Wheelbase	Length
count	407.000000	407.000000	407.000000
mean	0.326739	0.349028	0.455295
std	0.142773	0.152024	0.150843
min	0.000000	0.000000	0.000000
25%	0.237640	0.254545	0.368421
50%	0.304682	0.327273	0.463158
75%	0.399251	0.418182	0.536842
max	1.000000	1.000000	1.000000

3.7 Concatenating DataFrames

[339]:
```
cars_df = pd.concat([cars_encode_df, cars_num_df], axis=1)
```

[340]:
```
cars_df.head()
```

[340]:

	Hybrid	SUV	Sedan	Sports	Truck	Wagon	EngineSize	Cylinders	\
0	0.0	1.0	0.0	0.0	0.0	0.0	0.314286	0.333333	
1	0.0	0.0	1.0	0.0	0.0	0.0	0.100000	0.111111	
2	0.0	0.0	1.0	0.0	0.0	0.0	0.157143	0.111111	

```
3     0.0  0.0    1.0     0.0    0.0    0.0    0.271429   0.333333
4     0.0  0.0    1.0     0.0    0.0    0.0    0.314286   0.333333

   Horsepower  Mileage_City  Mileage_Highway    Weight  Wheelbase    Length
0    0.449649          0.14         0.203704  0.487079   0.309091  0.484211
1    0.297424          0.28         0.351852  0.173783   0.218182  0.305263
2    0.297424          0.24         0.314815  0.258427   0.290909  0.421053
3    0.461358          0.20         0.296296  0.323034   0.345455  0.452632
4    0.355972          0.16         0.222222  0.380150   0.472727  0.568421
```

[341]:
```
cars_df.shape
```

[341]:
```
(407, 14)
```

3.8 Outliers

During the data collection, there are chances of abnormal values creeping into the data. This abnormal value is called an outlier.

The outliers modify the final results or conclusions drawn from the data.

Outliers are observations that are distant or abnormal from other observations.

Outliers can be very low or very high values.

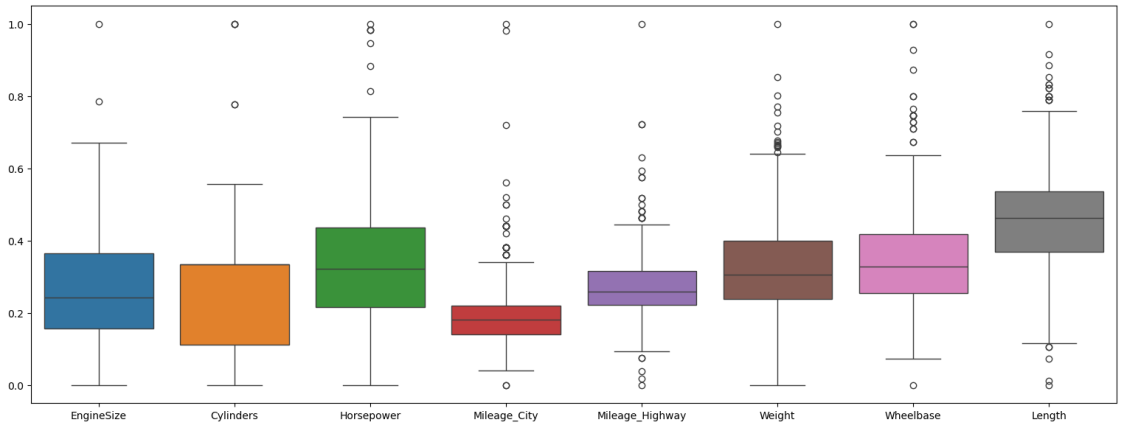
Causes of Outliers:

- Incorrect data entry
- Mis-reporting
- Sampling error
- Exceptional but true value

[342]:
```
plt.figure(figsize=(19, 7))

sns.boxplot(cars_df[num_columns])

plt.show()
```



3.8.1 Based on Inter Quartile Range (IQR)

[343]:
```
# obtain the first quartile
Q1 = cars_df.quantile(0.25)

# obtain the third quartile
Q3 = cars_df.quantile(0.75)

# obtain the IQR
IQR = Q3 - Q1

# print the IQR
print(IQR)
```

```
Hybrid             0.000000
SUV                0.000000
Sedan              1.000000
Sports             0.000000
Truck              0.000000
Wagon              0.000000
EngineSize         0.207143
Cylinders          0.222222
Horsepower         0.220141
Mileage_City       0.080000
Mileage_Highway    0.092593
Weight             0.161610
Wheelbase          0.163636
Length             0.168421
dtype: float64
```

[344]:
```
ul = Q3 + 1.5 * IQR
```

[345]:
```
ll = Q1 - 1.5 * IQR
```

[346]:
```
type(ll)
```

[346]:
```
pandas.core.series.Series
```

Any values above the upper limit or below the lower limit are considered as outliers.

When outliers are present in the dataset, either delete them or replace them with appropriate value.

When the dataset has huge number of rows, it is better to delete the outliers.

When the dataset had less number of rows, deleting the outliers will affect the result.

Deleting the outliers

[347]:
```
cars_df.shape
```

[347]: (407, 14)

[348]:
```
#cars_df = cars_df[~((cars_df < ll) |(cars_df > ul)).any(axis=1)]
```

Replacing the outlier by mean/median value

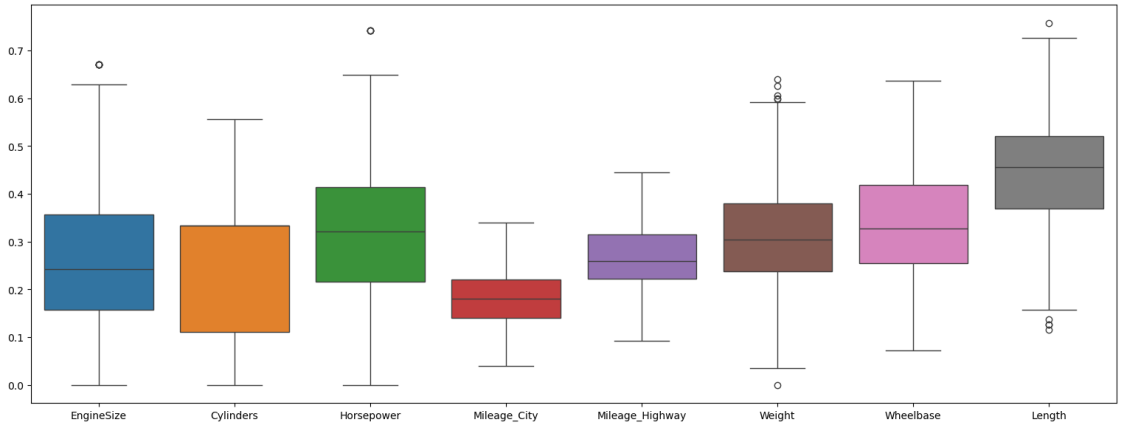
[349]:
```
cars_df['EngineSize'] = np.where(
    (cars_df['EngineSize'] < ll['EngineSize']) | (cars_df['EngineSize'] >␣
 ↪ul['EngineSize']),
    cars_df['EngineSize'].mean(),
    cars_df['EngineSize']
)
```

[350]:
```
for column in num_columns:
    cars_df[column] = np.where(
    (cars_df[column] < ll[column]) | (cars_df[column] > ul[column]),
    cars_df[column].mean(),
    cars_df[column]
    )
```

[351]:
```
plt.figure(figsize=(19, 7))

sns.boxplot(cars_df[num_columns])

plt.show()
```



[352]:
```
cars_df.shape
```

[352]: (407, 14)

[353]:
```
cars_df.head()
```

[353]:

	Hybrid	SUV	Sedan	Sports	Truck	Wagon	EngineSize	Cylinders	\
0	0.0	1.0	0.0	0.0	0.0	0.0	0.314286	0.333333	
1	0.0	0.0	1.0	0.0	0.0	0.0	0.100000	0.111111	
2	0.0	0.0	1.0	0.0	0.0	0.0	0.157143	0.111111	
3	0.0	0.0	1.0	0.0	0.0	0.0	0.271429	0.333333	
4	0.0	0.0	1.0	0.0	0.0	0.0	0.314286	0.333333	

	Horsepower	Mileage_City	Mileage_Highway	Weight	Wheelbase	Length
0	0.449649	0.14	0.203704	0.487079	0.309091	0.484211
1	0.297424	0.28	0.351852	0.173783	0.218182	0.305263
2	0.297424	0.24	0.314815	0.258427	0.290909	0.421053
3	0.461358	0.20	0.296296	0.323034	0.345455	0.452632
4	0.355972	0.16	0.222222	0.380150	0.472727	0.568421

[ ]: