What is Exploratory Data Analysis?

Exploratory Data Analysis or (EDA) is understanding the data set by summarizing its main characteristics and often plotting them visually. This step is very important especially when we arrive at modelling the data to apply Machine learning. Plotting in EDA consists of Histograms, Box plot, Scatter plots and many more.

Through the process of EDA, we can also refine the problem statement or definition of our problem.

How to perform Exploratory Data Analysis?

This is one such question that everyone is keen on knowing the answer. Well, the answer is it depends on the data set that you are working with.

There is no one common method of performing EDA. Here we will perform some of the more common methods and plots that would be used in the EDA process.

What data are we exploring today?

We are going to look at a data set on cars called "cardata.csv".

The data contains more than 10, 000 rows and more than 10 columns which have features of the car such as Engine Fuel Type, Engine Size, HP, Transmission Type, highway MPG, city MPG and many more.

1. Importing the required libraries for EDA

Below are the libraries that we will use to perform EDA (Exploratory data analysis)

```
# Importing required libraries.
import pandas as pd
import numpy as np
import seaborn as sns #visualisation
import matplotlib.pyplot as plt #visualisation
%matplotlib inline
sns.set(color_codes=True)
```

2. Loading the data into the data frame.

Loading the data into the pandas data frame is certainly one of the most important steps in EDA, as we can see that the value from the data set is comma-separated. So all we have to do is to just read the CSV into a data frame and pandas data frame does the job for us.

```
df = pd.read_csv("data/cardata.csv")
```

 # To display the top 5 rows
df.head(5)

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularity	MSRP
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe	26	19	3916	46135
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible	28	19	3916	40650
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe	28	20	3916	36350
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe	28	18	3916	29450
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	28	18	3916	34500

Displaying the top 5 rows.

 # To display the bottom 5 rows
df.tail(5)

Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularity	MSRP
Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	46120
Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	56670
Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	50620
Acura	ZDX	2013	premium unleaded (recommended)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover,Hatchback,Luxury	Midsize	4dr Hatchback	23	16	204	50920
Lincoln	Zephyr	2006	regular unleaded	221.0	6.0	AUTOMATIC	front wheel drive	4.0	Luxury	Midsize	Sedan	26	17	61	28995

Displaying the last 5 rows.

3. Checking the types of data

Here we check for the datatypes because sometimes features (variables) be stored as a string or an object.

In that case, we would have to convert any strings to integer data so that we could plot the data via a graph. In this case, the data is already in integer format so there’s nothing to worry about.

 # Checking the data type
df.dtypes

```
Make                  object
Model                 object
Year                   int64
Engine Fuel Type      object
Engine HP            float64
Engine Cylinders     float64
Transmission Type     object
Driven_Wheels         object
Number of Doors      float64
Market Category       object
Vehicle Size          object
Vehicle Style         object
highway MPG            int64
city mpg               int64
Popularity             int64
MSRP                   int64
dtype: object
```

Checking the type of data.

4. Dropping irrelevant columns

This step removes any features that we don't need. In this case, the columns such as Engine Fuel Type, Market Category, Vehicle style, Popularity, Number of doors, Vehicle Size are irrelevant and so I remove them

 # Dropping irrelevant columns
df = df.drop(['Engine Fuel Type', 'Market Category', 'Vehicle Style', 'Popularity', 'Number of Doors', 'Vehicle Size'], axis=1)
df.head(5)

	Make	Model	Year	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	highway MPG	city mpg	MSRP
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

Dropping irrelevant columns.

5. Renaming the columns

In this instance, some of the column names are a it confusing, so I just tweaked their column names. This is a good approach it improves the readability of the data set.

 # Renaming the column names
df = df.rename(columns={"Engine HP": "HP", "Engine Cylinders": "Cylinders", "Transmission Type": "Transmission", "Driven_Wheels": "Drive Mode","highway MPG": "MPG-H", "city mpg": "MPG-C", "MSRP": "Price" })
df.head(5)

	Make	Model	Year	HP	Cylinders	Transmission	Drive Mode	MPG-H	MPG-C	Price
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

Renaming the column name.

6. Dropping the duplicate rows

Duplicate observations can usually cause confusion in a data analysis. In this case I am going to remove them. Prior to removing I had 11914 rows of data but after removing the duplicates I have 10925 rows meaning that I had 989 rows of duplicate data.

 # Total number of rows and columns
df.shape

(11914, 10)

 # Rows containing duplicate data
duplicate_rows_df = df[df.duplicated()]
print(“number of duplicate rows: “, duplicate_rows_df.shape)

number of duplicate rows: (989, 10)

Now let us remove the duplicate data.

 # Used to count the number of rows before removing the data
df.count()

Make 11914
Model 11914
Year 11914
HP 11845
Cylinders 11884
Transmission 11914
Drive Mode 11914
MPG-H 11914
MPG-C 11914
Price 11914
dtype: int64

So, as seen above there are 11914 rows and we are removing 989 rows of duplicate data.

 # Dropping the duplicates
df = df.drop_duplicates()
df.head(5)

	Make	Model	Year	HP	Cylinders	Transmission	Drive Mode	MPG-H	MPG-C	Price
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

```
# Counting the number of rows after removing duplicates.
df.count()
```

```
Make            10925
Model           10925
Year            10925
HP              10856
Cylinders       10895
Transmission    10925
Drive Mode      10925
MPG-H           10925
MPG-C           10925
Price           10925
dtype: int64
```

7. Dropping the missing or null values.

This is very similar to the previous step but here all the missing values are detected and are then dropped.

Now, this is a controversial step, many people just replace the missing values with the mean or the average of that column, but in this case, I am just dropping the rows with missing values. As there are approximately 100 missing values compared to over 10, 000 values in the whole dataset, this is negligible.

```
# Finding the null values.
print(df.isnull().sum())
```

```
Make             0
Model            0
Year             0
HP              69
Cylinders       30
Transmission     0
Drive Mode       0
MPG-H            0
MPG-C            0
Price            0
dtype: int64
```

This is the reason in the above step while counting both Cylinders and Horsepower (HP) had 10856 and 10895 over 10925 rows.

 # Dropping the missing values.
df = df.dropna()
df.count()

Make 10827
Model 10827
Year 10827
HP 10827
Cylinders 10827
Transmission 10827
Drive Mode 10827
MPG-H 10827
MPG-C 10827
Price 10827
dtype: int64

Now we have removed all the rows which contain the Null or N/A values (Cylinders and Horsepower (HP)).

 # After dropping the values
print(df.isnull().sum())

Make 0
Model 0
Year 0
HP 0
Cylinders 0
Transmission 0
Drive Mode 0
MPG-H 0
MPG-C 0
Price 0
dtype: int64

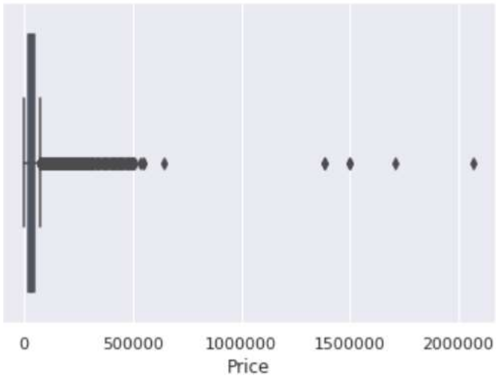
8. Detecting Outliers

An outlier is a datapoint or set of datapoints that are vastly different from other data points in your dataset. Sometimes they can be very high or very low.

It’s often a good idea to detect and remove the outliers because outliers are one of the primary reasons for models becoming less accurate. We are going to use the IQR (Interquartile Range) Scoring technique to detect and remove any outliers. Often outliers can be seen with visualizations such as a box plot. Shown below are the box plots of MSRP, Cylinders, Horsepower and EngineSize. In all the plots, you can find some points are that outside the “box”. These are our outliers.

sns.boxplot(x=df['Price'])

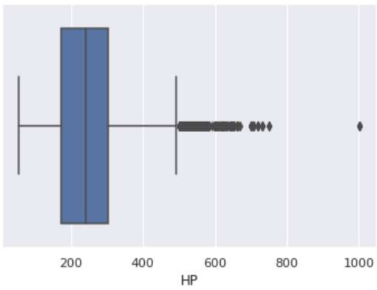
<matplotlib.axes._subplots.AxesSubplot at 0x7f69f68edc18>



Box plot of Price

sns.boxplot(x=df['HP'])

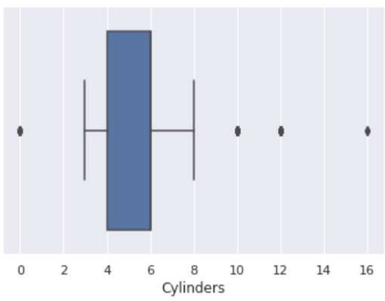
<matplotlib.axes._subplots.AxesSubplot at 0x7f69f3d68240>



Box Plot of HP

sns.boxplot(x=df['Cylinders'])

<matplotlib.axes._subplots.AxesSubplot at 0x7f69f3d2d400>



Box Plot of Cylinders

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 — Q1
print(IQR)
```

Year 9.0
HP 130.0
Cylinders 2.0
MPG-H 8.0
MPG-C 6.0
Price 21327.5
dtype: float64

Don't worry about the above values because it's not important to know each and every one of them. The program will use these values to remove our outliers. It's just important to know how to use this technique.

```
df = df[~((df < (Q1–1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
df.shape
```

(9191, 10)

As seen above there were around 1600 outliers' rows removed.

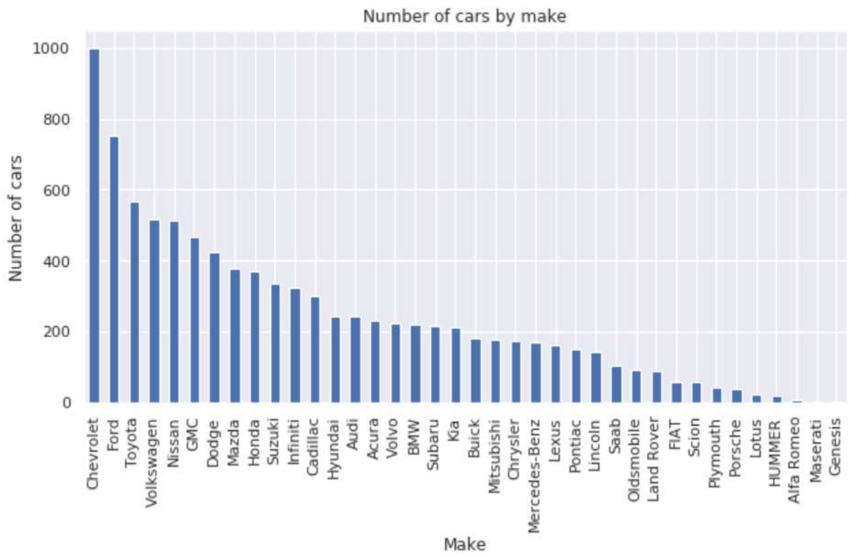
NOTE. This technique will remove MOST of the outliers but usually not all, a couple will remain, but they will have become negligible in the overall analysis.

9. Plot different features against one another (scatter), against frequency (histogram)

Histogram

Histograms refer to the frequency of occurrence of variables in an interval. In this case, there are mainly 10 different types of car manufacturing companies, and we want to know who has the greatest number of cars. A histogram will quickly lets us know the total number of cars manufactured by each company.

```
# Plotting a Histogram
df.Make.value_counts().nlargest(40).plot(kind='bar', figsize=(10,5))
plt.title("Number of cars by make")
plt.ylabel('Number of cars')
plt.xlabel('Make');
```



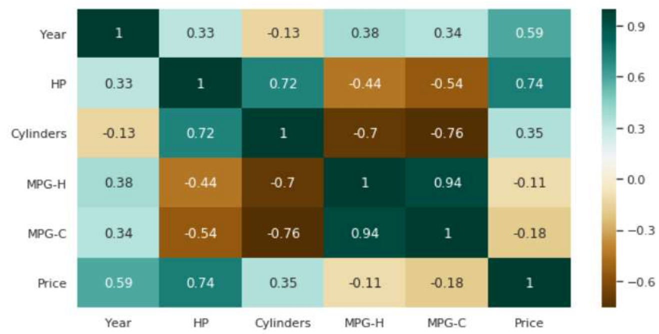
Histogram

Heat Maps

Heat Maps are a type of plot which is allow us to find the dependent variables or the relationship between the features. For Example, In the heat map below, we can see that the price feature depends mainly on the Engine Size, Horsepower, and Cylinders.

 # Finding the relations between the variables.
plt.figure(figsize=(20,10))
c= df.corr()
sns.heatmap(c,cmap="BrBG",annot=True)
c

	Year	HP	Cylinders	MPG-H	MPG-C	Price
Year	1.000000	0.326726	-0.133920	0.378479	0.338145	0.592983
HP	0.326726	1.000000	0.715237	-0.443807	-0.544551	0.739042
Cylinders	-0.133920	0.715237	1.000000	-0.703856	-0.755540	0.354013
MPG-H	0.378479	-0.443807	-0.703856	1.000000	0.939141	-0.106320
MPG-C	0.338145	-0.544551	-0.755540	0.939141	1.000000	-0.180515
Price	0.592983	0.739042	0.354013	-0.106320	-0.180515	1.000000



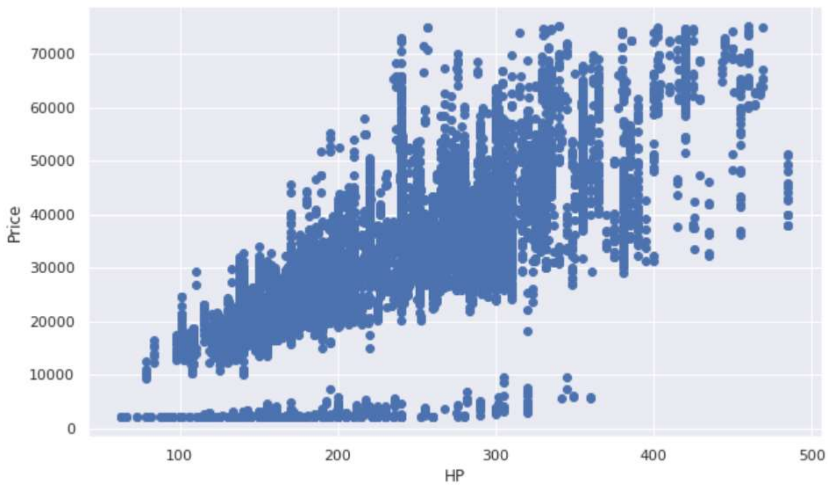
Heat Maps.

Scatterplot

We generally use scatter plots to find the correlation between two variables. Horsepower and Price have been plotted and we can easily draw a trend line using this visualization suggesting that there is a correlation between these 2 features.

 # Plotting a scatter plot

```
fig, ax = plt.subplots(figsize=(10,6))
ax.scatter(df['HP'], df['Price'])
ax.set_xlabel('HP')
ax.set_ylabel('Price')
plt.show()
```



Scatter Plot