**Exploratory Data Analysis**

Exploratory Data Analysis, or EDA, is an important step in any Data Analysis or Data Science project. EDA is the process of investigating the dataset to discover patterns, and anomalies (outliers), and form hypotheses based on our understanding of the dataset. EDA involves generating summary statistics for numerical data in the dataset and creating various graphical representations to understand the data better. This is usually the understanding the data sets by summarizing their main characteristics often plotting them visually. we will understand EDA with the help of an example datasets

Since I am a huge fan of cars, I got a very beautiful data-set of cars from Kaggle I have inserted the dataset for the reference. The data-set can be downloaded from [**here**](https://www.kaggle.com/CooperUnion/cardataset)**.**

To give a piece of brief information about the data set this data contains more of 10, 000 rows and more than 10 columns which contains features of the car such as Engine Fuel Type, Engine HP, Transmission Type, highway MPG, city MPG and many more. So, we will explore the data and make it ready for modelling.

**Importing libraries:**

We will be importing the libraries that require for performing EDA. These include NumPy, Pandas, Matplotlib, and Seaborn.

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)

## **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.csv")

//To display top 5 items of the dataset

print(df.head(5))

Make Model Year Engine Fuel Type Engine HP \

0 BMW 1 Series M 2011 premium unleaded (required) 335.0

1 BMW 1 Series 2011 premium unleaded (required) 300.0

2 BMW 1 Series 2011 premium unleaded (required) 300.0

3 BMW 1 Series 2011 premium unleaded (required) 230.0

4 BMW 1 Series 2011 premium unleaded (required) 230.0

Engine Cylinders Transmission Type Driven\_Wheels Number of Doors \

0 6.0 MANUAL rear wheel drive 2.0

1 6.0 MANUAL rear wheel drive 2.0

2 6.0 MANUAL rear wheel drive 2.0

3 6.0 MANUAL rear wheel drive 2.0

4 6.0 MANUAL rear wheel drive 2.0

Market Category Vehicle Size Vehicle Style \

0 Factory Tuner,Luxury,High-Performance Compact Coupe

1 Luxury,Performance Compact Convertible

2 Luxury,High-Performance Compact Coupe

3 Luxury,Performance Compact Coupe

4 Luxury Compact Convertible

highway MPG city mpg Popularity MSRP

0 26 19 3916 46135

1 28 19 3916 40650

2 28 20 3916 36350

3 28 18 3916 29450

4 28 18 3916 34500

//To display last 5 items of the dataset

Print(df.tail(5))

Make Model Year Engine Fuel Type Engine HP \

11909 Acura ZDX 2012 premium unleaded (required) 300.0

11910 Acura ZDX 2012 premium unleaded (required) 300.0

11911 Acura ZDX 2012 premium unleaded (required) 300.0

11912 Acura ZDX 2013 premium unleaded (recommended) 300.0

11913 Lincoln Zephyr 2006 regular unleaded 221.0

Engine Cylinders Transmission Type Driven\_Wheels Number of Doors \

11909 6.0 AUTOMATIC all wheel drive 4.0

11910 6.0 AUTOMATIC all wheel drive 4.0

11911 6.0 AUTOMATIC all wheel drive 4.0

11912 6.0 AUTOMATIC all wheel drive 4.0

11913 6.0 AUTOMATIC front wheel drive 4.0

Market Category Vehicle Size Vehicle Style highway MPG \

11909 Crossover,Hatchback,Luxury Midsize 4dr Hatchback 23

11910 Crossover,Hatchback,Luxury Midsize 4dr Hatchback 23

11911 Crossover,Hatchback,Luxury Midsize 4dr Hatchback 23

11912 Crossover,Hatchback,Luxury Midsize 4dr Hatchback 23

11913 Luxury Midsize Sedan 26

city mpg Popularity MSRP

11909 16 204 46120

11910 16 204 56670

11911 16 204 50620

11912 16 204 50920

11913 17 61 28995

## **Checking the types of data**

Here we check for the datatypes because sometimes there might be some attributes which required to be specified in specific datatype to avoid misconception.

We can **change the column type** with the astype method. Let's apply this method to the MSRP feature to convert it into int64:

df["MSRP "] = df["MSRP "].astype("int64")

In the above dataset we have an attribute named MSRP or the price of the car which might be stored as a string, if in that case, we have to convert that string to the integer data only then we can plot the data via a graph. Here, in this case, the data is already in integer format so nothing to worry.

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

## **Dropping irrelevant columns**

This step is certainly needed in every EDA because sometimes there would be many columns that we never use in such cases dropping is the only solution. In this case, the columns such as Engine Fuel Type, Market Category, Vehicle style, Popularity, Number of doors, Vehicle Size doesn't make any sense to me so I just dropped for this instance.

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 |

Lets print out column names using columns

print(df.columns)

Index(['Make', 'Model', 'Year', 'Engine HP', 'Engine Cylinders',

'Transmission Type', 'Driven\_Wheels', 'highway MPG', 'city mpg',

'MSRP'],

dtype='object')

## **Renaming the columns**

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

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 |

## **Dropping the duplicate rows**

This is often a handy thing to do because a huge data set as in this case contains more than 10, 000 rows often have some duplicate data which might be disturbing, so here I remove all the duplicate value from the data-set. For example prior to removing I had 11914 rows of data but after removing the duplicates 10925 data meaning that I had 989 of duplicate data.

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

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

df = df.drop\_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

## **Dropping the missing or null values**

This is mostly similar to the previous step but in here all the missing values are detected and are dropped later. Now, this is not a good approach to do so, because many people just replace the missing values with the mean or the average of that column, but in this case, I just dropped that missing values. This is because there is nearly 100 missing value compared to 10, 000 values this is a small number and this is negligible so I just dropped those 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.

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

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

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

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

**value\_counts()** function returns object containing counts of unique values.

df['Price'].value\_counts()

2000 599

29995 18

25995 16

20995 15

27995 15

...

49795 1

51525 1

47740 1

25199 1

2027 1

Name: Price, Length: 6014, dtype: int64

**Normalizing** the values of attirbutes

df["Price"].value\_counts(normalize=True)

2000 0.055325

29995 0.001663

25995 0.001478

20995 0.001385

27995 0.001385

...

49795 0.000092

51525 0.000092

47740 0.000092

25199 0.000092

2027 0.000092

Name: Price, Length: 6014, dtype: float64

**Sorting values in descending order**

df.sort\_values(by="Make", ascending=False).head()

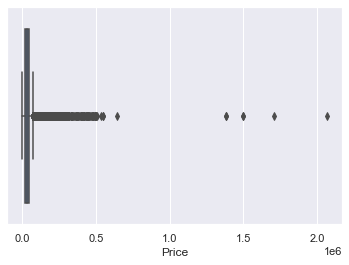
|  | **Make** | **Model** | **Year** | **HP** | **Cylinders** | **Transmission** | **Drive Mode** | **MPG-H** | **MPG-C** | **Price** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **8949** | Volvo | S90 | 2017 | 250.0 | 4.0 | AUTOMATIC | front wheel drive | 34 | 23 | 46950 |
| **11133** | Volvo | V60 | 2016 | 240.0 | 4.0 | AUTOMATIC | front wheel drive | 29 | 25 | 41750 |
| **11639** | Volvo | XC90 | 2017 | 316.0 | 4.0 | AUTOMATIC | all wheel drive | 25 | 20 | 55600 |
| **11640** | Volvo | XC | 2002 | 197.0 | 5.0 | AUTOMATIC | all wheel drive | 23 | 17 | 36500 |
| **11138** | Volvo | V60 | 2017 | 302.0 | 4.0 | AUTOMATIC | all wheel drive | 31 | 23 | 48950 |

## **Detecting Outliers**

An outlier is a point or set of points that are different from other points. 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 resulting in a less accurate model. Hence it's a good idea to remove them. The outlier detection and removing that I am going to perform is called IQR score technique. Often outliers can be seen with visualizations using a box plot. Shown below are the box plot of MSRP, Cylinders, Horsepower and EngineSize. Herein all the plots, you can find some points are outside the box they are none other than outliers

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

<AxesSubplot:xlabel='Price'>



## **Plot different features against one another (scatter), against frequency (histogram)**

### Histogram

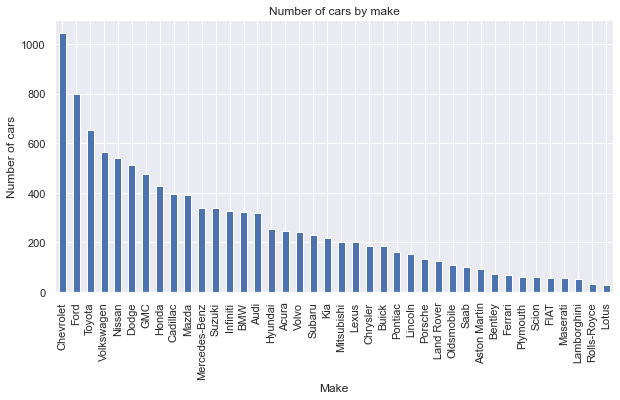
Histogram refers to the frequency of occurrence of variables in an interval. In this case, there are mainly 10 different types of car manufacturing companies, but it is often important to know who has the most number of cars. To do this histogram is one of the trivial solutions which lets us know the total number of car manufactured by a different company.

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');



### Heat Maps

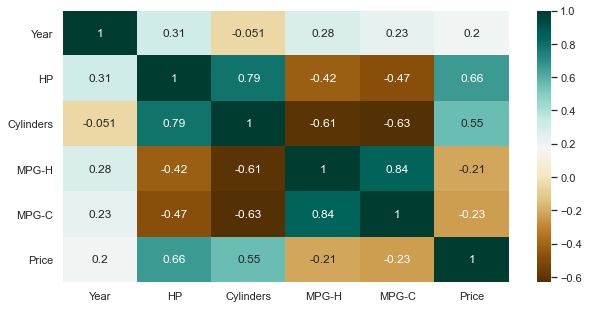
Heat Maps is a type of plot which is necessary when we need to find the dependent variables. One of the best way to find the relationship between the features can be done using heat maps. In the below heat map we know that the price feature depends mainly on the Engine Size, Horsepower, and Cylinders.

plt.figure(figsize=(10,5))

c= df.corr()

sns.heatmap(c,cmap="BrBG",annot=True)

<AxesSubplot:>



### Scatterplot

We generally use scatter plots to find the correlation between two variables. Here the scatter plots are plotted between Horsepower and Price and we can see the plot below. With the plot given below, we can easily draw a trend line. These features provide a good scattering of points.

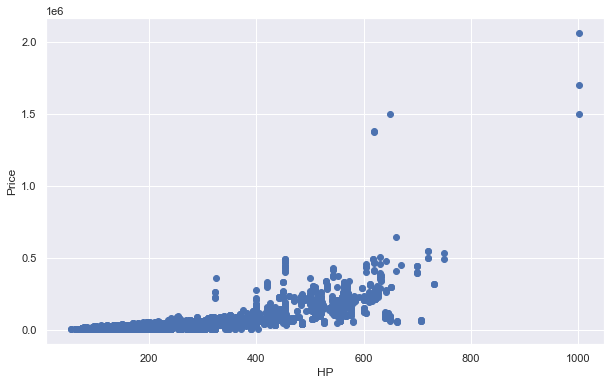
fig, ax = plt.subplots(figsize=(10,6))

ax.scatter(df['HP'], df['Price'])

ax.set\_xlabel('HP')

ax.set\_ylabel('Price')

plt.show()



References:

<https://www.kaggle.com/ekami66/detailed-exploratory-data-analysis-with-python>

<https://www.kaggle.com/imoore/intro-to-exploratory-data-analysis-eda-in-python>