1. [Python](https://nbviewer.org/github/Tanu-N-Prabhu/Python/tree/master)

1. [Exploratory\_data\_Analysis.ipynb](https://nbviewer.org/github/Tanu-N-Prabhu/Python/tree/master/Exploratory_data_Analysis.ipynb)

**Exploratory data analysis in Python.**

**Let us understand how to explore the data in python.**



Image Credits: Morioh

**Introduction**

**What is Exploratory Data Analysis ?**

Exploratory Data Analysis or (EDA) is understanding the data sets by summarizing their main characteristics often plotting them visually. This step is very important especially when we arrive at modeling the data in order to apply Machine learning. Plotting in EDA consists of Histograms, Box plot, Scatter plot and many more. It often takes much time to explore the data. Through the process of EDA, we can ask to define the problem statement or definition on our data set which is very important.

**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. There is no one method or common methods in order to perform EDA, whereas in this tutorial you can understand some common methods and plots that would be used in the EDA process.

**What data are we exploring today ?**

Since I am a huge fan of cars, I got a very beautiful data-set of cars from Kaggle. 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 in this tutorial, we will explore the data and make it ready for modeling.

**1. Importing the required libraries for EDA**

Below are the libraries that are used in order to perform EDA (Exploratory data analysis) in this tutorial.

In [0]:

**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.

To get or load the dataset into the notebook, all I did was one trivial step. In Google Colab at the left-hand side of the notebook, you will find a > (greater than symbol). When you click that you will find a tab with three options, you just have to select Files. Then you can easily upload your file with the help of the Upload option. No need to mount to the google drive or use any specific libraries just upload the data set and your job is done. One thing to remember in this step is that uploaded files will get deleted when this runtime is recycled. This is how I got the data set into the notebook.

In [2]:

df = pd.read\_csv("data.csv")

*# To display the top 5 rows*

df.head(5)

Out[2]:

|  | **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 |

In [3]:

df.tail(5) *# To display the botton 5 rows*

Out[3]:

|  | **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** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **11909** | 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 |
| **11910** | 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 |
| **11911** | 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 |
| **11912** | 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 |
| **11913** | Lincoln | Zephyr | 2006 | regular unleaded | 221.0 | 6.0 | AUTOMATIC | front wheel drive | 4.0 | Luxury | Midsize | Sedan | 26 | 17 | 61 | 28995 |

**3. Checking the types of data**

Here we check for the datatypes because sometimes the MSRP or the price of the car would 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.

In [4]:

df.dtypes

Out[4]:

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

**4. 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.

In [5]:

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

df.head(5)

Out[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 |

**5. 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.

In [6]:

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)

Out[6]:

|  | **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 |

**6. 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.

In [7]:

df.shape

Out[7]:

(11914, 10)

In [8]:

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 because it's ok to remove them.

In [9]:

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

Out[9]:

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.

In [10]:

df = df.drop\_duplicates()

df.head(5)

Out[10]:

|  | **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 |

In [11]:

df.count()

Out[11]:

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 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.

In [12]:

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.

In [13]:

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

df.count()

Out[13]:

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)).

In [14]:

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

**8. Detecting Outliers**

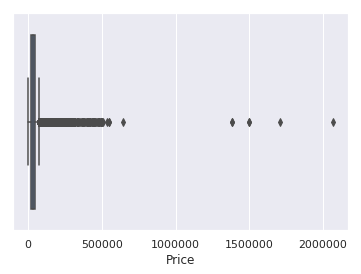
An outlier is a point or set of points that are different from other points. Sometimes they can be very high or very low. It's often a good idea to detect and remove the outliers. Because outliers are one of the primary reasons for resulting in a less accurate model. Hence it's a good idea to remove 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. The technique of finding and removing outlier that I am performing in this assignment is taken help of a tutorial from[towards data science](https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba).

In [15]:

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

Out[15]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f0d36a38be0>

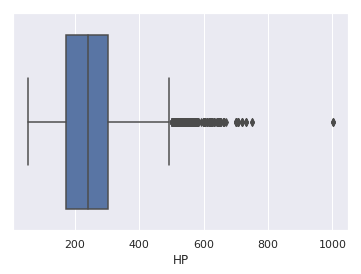


In [16]:

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

Out[16]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f0d369b3ba8>

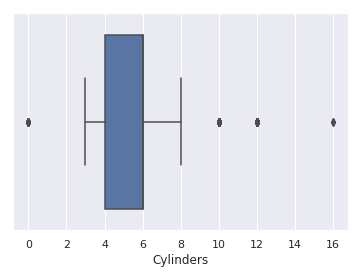


In [17]:

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

Out[17]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f0d3413ff28>



In [18]:

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 because it's just important to know how to use this technique in order to remove the outliers.

In [19]:

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

df.shape

Out[19]:

(9191, 10)

As seen above there were around 1600 rows were outliers. But you cannot completely remove the outliers because even after you use the above technique there maybe 1–2 outlier unremoved but that ok because there were more than 100 outliers. Something is better than nothing.

**9. 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.

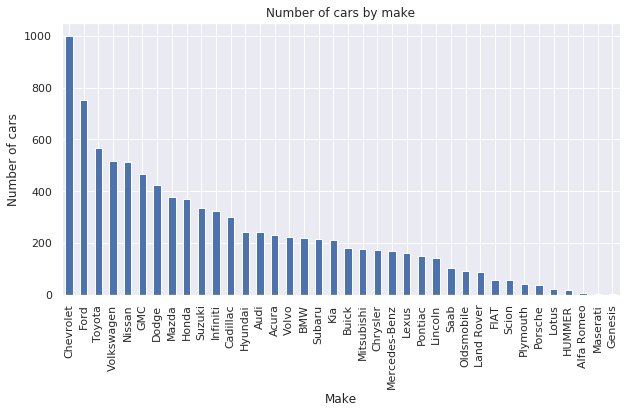
In [20]:

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.

In [21]:

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

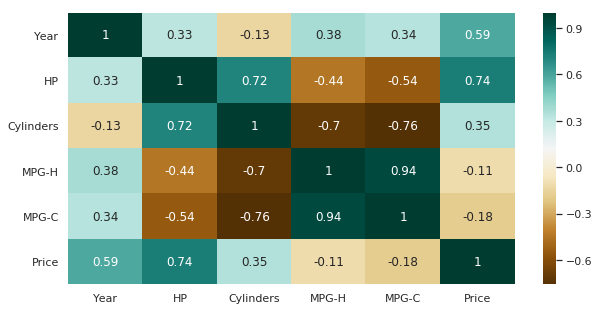
c= df.corr()

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

c

Out[21]:

|  | **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 |



**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.

In [22]:

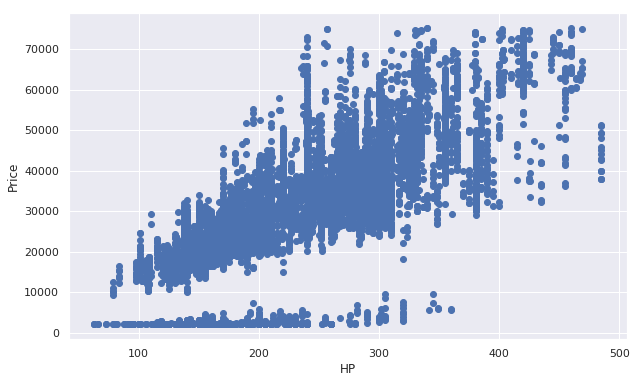
fig, ax = plt.subplots(figsize=(10,6))

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

ax.set\_xlabel('HP')

ax.set\_ylabel('Price')

plt.show()



**Hence the above are some of the steps involved in Exploratory data analysis, these are some general steps that you must follow in order to perform EDA. There are many more yet to come but for now, this is more than enough idea as to how to perform a good EDA given any data sets. Stay tuned for more updates.**

**Thank you.**

This website does not host notebooks, it only renders notebooks available on other websites.

Delivered by [Fastly](https://www.fastly.com/), Rendered by [OVHcloud](https://ovhcloud.com/)

nbviewer GitHub [repository](https://github.com/jupyter/nbviewer).

nbviewer version: [d25d3c3](https://github.com/jupyter/nbviewer/commit/d25d3c39c10d5f6b9891eba2503f30689c1aae28)

nbconvert version: [5.6.1](https://github.com/jupyter/nbconvert/releases/tag/5.6.1)

Rendered a minute ago