Topic : Introduction to Jupyter Notebook and Colab Environment

Objective for this template:

1. To introduce participants components of the Jupyter/Colab Notebook environments.
2. To allow participants to gain basic understanding of the Python Programming Language
3. To provide participants opportunities to practice implementing basic data analysis activities that demonstrate the cabilities of Python and Jupyter notebook.

Permission granted to use template for educational purposes so long as this heading is not removed.

What is Jupyter Notebook?

- The ***Jupyter Notebook*** is an incredibly powerful tool for interactively
- A ***notebook*** integrates code, its output, as well as narrative text into a single document such that you can run code, display the output, and also add explanations, formulas, and charts in a single document.
- Using Notebooks is now a major part of the data science workflow at companies across the globe.
- If your goal is to work with data, using a Notebook will speed up your workflow and make it easier to communicate and share your results.
- Jupyter Notebooks are completely free, being part of the open source Project Jupyter.
- You can download the software on its own, or as part of the Anaconda data science toolkit.

Installation

- The easiest way to get started with using ***Jupyter Notebooks*** is by installing ***Anaconda***.
- ***Anaconda*** is the most widely used ***Python distribution for data science*** and comes pre-loaded with all the most popular libraries and tools.
- Some of the biggest Python libraries included in Anaconda include ***NumPy***, ***pandas***, and ***Matplotlib***
- The latest version of Anaconda for Python is 3.8. The steps for Installing Anaconda include the following:

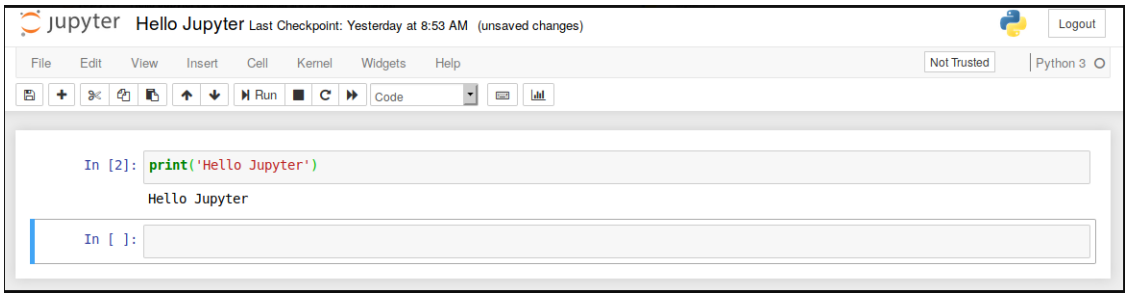
1. Visit Anaconda.com/downloads

2. Select Windows Installler
3. Download the .exe installer
4. Open and run the .exe installer
5. Open the Anaconda Prompt and run some Python code

Starting the Jupyter Notebook Server

After installing Anaconda, the next step is to start the Notebook Server application:
1. In the Windows Startup menu, find and execute the Anaconda Prompt terminal application
2. Inside the terminal application, go to a folder of your choice (e.g., in the Documents folder, create a subfolder called Notebooks or something else that is easy to remember).
3. Run the following command:
   jupyter notebook <enter>
   This will start up the Jupyter Notebook server and your default browser should start (or open a new tab) to the following URL: http://localhost:8888/tree

Your browser should now look like this:



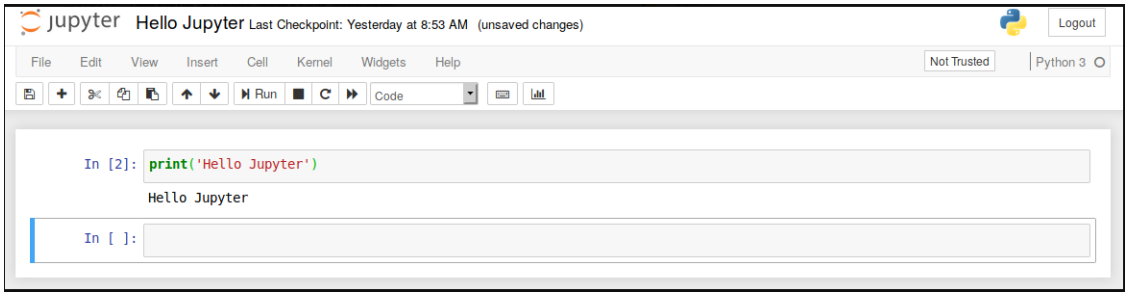
Note that you are just running the Notebook server and not actually running a Notebook.

Creating a Notebook

After starting the Notebook server, the next step is to create an actual Notebook document.

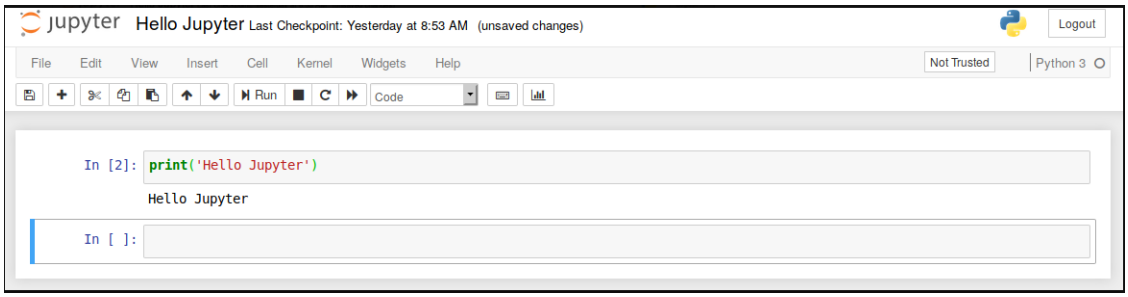
Click on the New button (upper right) and choose Python 3.

Your web page should now look like this:



Naming a Notebook

Notice that at the top of the page is the word Untitled. This is the default title for the page and the name of your Notebook. To change this title just move your mouse over the word Untitled and click on the text. In the in-browser dialog that appears, type the new name of your notebook.



Cell Types

Cells form the body of a notebook. In the screenshot of a new notebook in the section above, that box with the green outline is an empty cell. There are two basic cell types: Code and Markdown cells.

- A code cell contains code to be executed in the kernel. When the code is run, the notebook displays the output below the code cell that generated it.
- A Markdown cell contains text formatted using Markdown and displays its output in-place when the Markdown cell is run. The first cell in a new notebook is always a code cell.

Running Cells

A Notebook's cell defaults to using code whenever you first create one, and that cell uses the kernel that you chose when you started your Notebook.

In this case, you started yours with Python 3 as your kernel, so that means you can write Python code in your code cells. Since your initial Notebook has only one empty cell in it, the Notebook can't really do anything.

To verify that everything is working as it should, try typing the following code to that cell:

print('Hello Jupyter!')

Running a cell means that you will execute the cell's contents. To execute a cell, you can just select the cell and click the Run button that is in the row of buttons along the top. It's towards the middle. If you prefer using your keyboard, you can just press Shift+Enter.

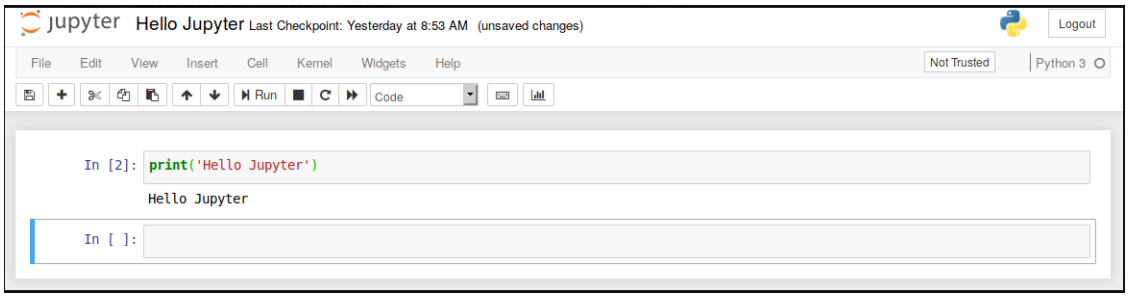
When a cell is executed, its output is displayed below the code cell and the label to the left of the code cell will have changed from In [ ] to In [1].

The output of a code cell also forms part of the document. The difference between code and Markdown cells is that code cells have that label on the left and Markdown cells do not.

The "In" part of the label is simply short for "Input," while the label number indicates when the cell was executed on the kernel — in this case the cell was executed first.

Running the cell again will change the label to In [2] because now the cell was the second to be run on the kernel.

When the sample code above is executed, the screen output should look like this:



Keyboard Shortcuts

When cells are running, their border turns blue, whereas it was green while being edited. In a Jupyter Notebook, there is always one "active" cell highlighted with a border whose color denotes its current mode:

Green outline — cell is in "edit mode" Blue outline — cell is in "command mode"

When a cell is in command mode, there are plenty of other commands that can be used. The best way to call these commands is with keyboard shortcuts

Keyboard shortcuts are a very popular aspect of the Jupyter environment because they facilitate a speedy cell-based workflow. Many of these are actions you can carry out on the active cell when it's in command mode.

Below is a list of some of Jupyter's keyboard shortcuts.

Toggle between edit and command mode with Esc and Enter, respectively. Once in command mode: Scroll up and down your cells with your Up and Down keys. Press A or B

to insert a new cell above or below the active cell. M will transform the active cell to a Markdown cell. Y will set the active cell to a code cell. D + D (D twice) will delete the active cell. Z will undo cell deletion. Hold Shift and press Up or Down to select multiple cells at once. With multiple cells selected, Shift + M will merge your selection. Ctrl + Shift + -, in edit mode, will split the active cell at the cursor. You can also click and Shift + Click in the margin to the left of your cells to select them.

Practice using these commands in your own notebook.

Kernels

Behind every notebook runs a kernel. When you run a code cell, that code is executed within the kernel. Any output is returned back to the cell to be displayed. The kernel's state persists over time and between cells — it pertains to the document as a whole and not individual cells.

For example, if you import libraries or declare variables in one cell, they will be available in another. To try and get a feel for this setup, type the following code in a code cell:

```
import numpy as np
def square(x):
    return x * x
```

Once the code cell for the code above is executed, then create another code cell and type the sample codes below. Notice that the variable np and function square can be refenced in this new cell evn though they were declared in another cell.

```
x = np.random.randint(1, 10)
y = square(x)
print('%d squared is %d' % (x, y))
```

This will work regardless of the order of the cells in the notebook. As long as a cell has been run, any variables declared or libraries imported inside them will always be available in other cells.

==========

Basic Data Analysis

After looking at the features of Jupyter Notebook, we can now apply these featues in practicing basic data analysis.

The goal is to apply basic data analysis commands to better explore the nature and characteristics of a dataset.

Take note that in data analysis, each person will develop his/her own preferences and style of analysis, but the general techniques demonstrated here will still be applicable. You can either setup your own analysis approach and create your own notebook or follow along with the steps provided and use it as a guide.

The external data used in this example is stored in the file car_features_data.csv.

The CSV file contains information about various features of automobiles including make, model, year, engine, and other properties that can be used to predict its price.

Step 1:

Import the libraries that are needed to perform basic Exploratory data analysis. We import pandas to work with our data, Matplotlib to plot charts, and Seaborn to make more complex.

The first line isn't a Python command, but uses something called a line magic to instruct Jupyter to capture Matplotlib plots and render them in the cell output.

```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="darkgrid")
```

Step 2:

Loading the data into the pandas data frame. Since the data set is comma-separated all we have to do is to read the CSV into a pandas data frame to allow us to manipulate the data inside Jupyter notebook.

```
df = pd.read_csv('E:/My Documents/Files (1st Sem
2021-2022)/Research/Training - DataScience and DL
workshop/datasets/car_features_data.csv')
```

Step 3:

Checking shape, sample data, data types, and other data characteristics

```
df.shape
```

```
(11914, 16)
```

```
df.head(10)
```

```
  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
5  BMW    1 Series  2012  premium unleaded (required)      230.0
6  BMW    1 Series  2012  premium unleaded (required)      300.0
7  BMW    1 Series  2012  premium unleaded (required)      300.0
8  BMW    1 Series  2012  premium unleaded (required)      230.0
9  BMW    1 Series  2013  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
5               6.0           MANUAL  rear wheel drive
2.0
6               6.0           MANUAL  rear wheel drive
2.0
7               6.0           MANUAL  rear wheel drive
2.0
8               6.0           MANUAL  rear wheel drive
2.0
9               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
5                     Luxury,Performance      Compact         Coupe
6                     Luxury,Performance      Compact   Convertible
7                Luxury,High-Performance      Compact         Coupe
8                                Luxury      Compact   Convertible
9                                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
```

```
5           28        18        3916  31200
6           26        17        3916  44100
7           28        20        3916  39300
8           28        18        3916  36900
9           27        18        3916  37200

df.tail()

          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
```

```
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
```

```
len(df)
```

```
11914
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10827 entries, 0 to 11913
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Make          10827 non-null  object 
 1   Model         10827 non-null  object 
 2   Year          10827 non-null  int64  
 3   HP            10827 non-null  float64
 4   Cylinders     10827 non-null  float64
 5   Transmission  10827 non-null  object 
 6   Drive Mode    10827 non-null  object 
 7   MPG-H         10827 non-null  int64  
 8   MPG-C         10827 non-null  int64  
 9   Price         10827 non-null  int64  
dtypes: float64(2), int64(4), object(4)
memory usage: 930.4+ KB
```

Step 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.

```
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  \
0  BMW  1 Series M  2011      335.0               6.0
MANUAL
1  BMW    1 Series  2011      300.0               6.0
MANUAL
2  BMW    1 Series  2011      300.0               6.0
MANUAL
3  BMW    1 Series  2011      230.0               6.0
MANUAL
4  BMW    1 Series  2011      230.0               6.0
MANUAL

      Driven_Wheels  highway MPG  city mpg   MSRP
0  rear wheel drive           26        19  46135
1  rear wheel drive           28        19  40650
2  rear wheel drive           28        20  36350
3  rear wheel drive           28        18  29450
4  rear wheel drive           28        18  34500
```

```
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  \
0  BMW  1 Series M  2011  335.0        6.0       MANUAL  rear wheel
drive
1  BMW    1 Series  2011  300.0        6.0       MANUAL  rear wheel
drive
2  BMW    1 Series  2011  300.0        6.0       MANUAL  rear wheel
drive
3  BMW    1 Series  2011  230.0        6.0       MANUAL  rear wheel
drive
4  BMW    1 Series  2011  230.0        6.0       MANUAL  rear wheel
```

```
drive

   MPG-H  MPG-C  Price
0     26     19  46135
1     28     19  40650
2     28     20  36350
3     28     18  29450
4     28     18  34500
```

```
duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows: ", duplicate_rows_df.shape)
```

```
number of duplicate rows:  (989, 10)
```

```
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
```

```
df = df.drop_duplicates()
print(df.head(5))
print(df.count())
```

```
  Make       Model  Year     HP  Cylinders Transmission        Drive
Mode  \
0  BMW  1 Series M  2011  335.0        6.0       MANUAL  rear wheel
drive
1  BMW    1 Series  2011  300.0        6.0       MANUAL  rear wheel
drive
2  BMW    1 Series  2011  300.0        6.0       MANUAL  rear wheel
drive
3  BMW    1 Series  2011  230.0        6.0       MANUAL  rear wheel
drive
4  BMW    1 Series  2011  230.0        6.0       MANUAL  rear wheel
drive

   MPG-H  MPG-C  Price
0     26     19  46135
1     28     19  40650
2     28     20  36350
3     28     18  29450
4     28     18  34500
```

```
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
```

```
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
```

```
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
```

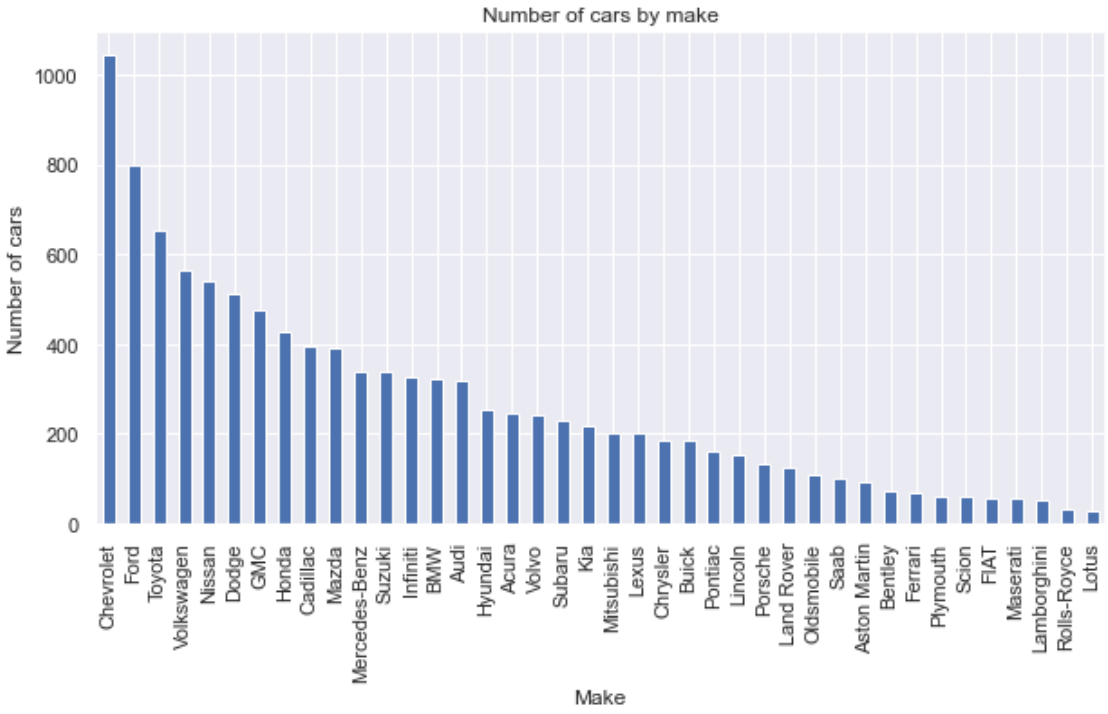
Step 5:

Plotting Features

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');
```

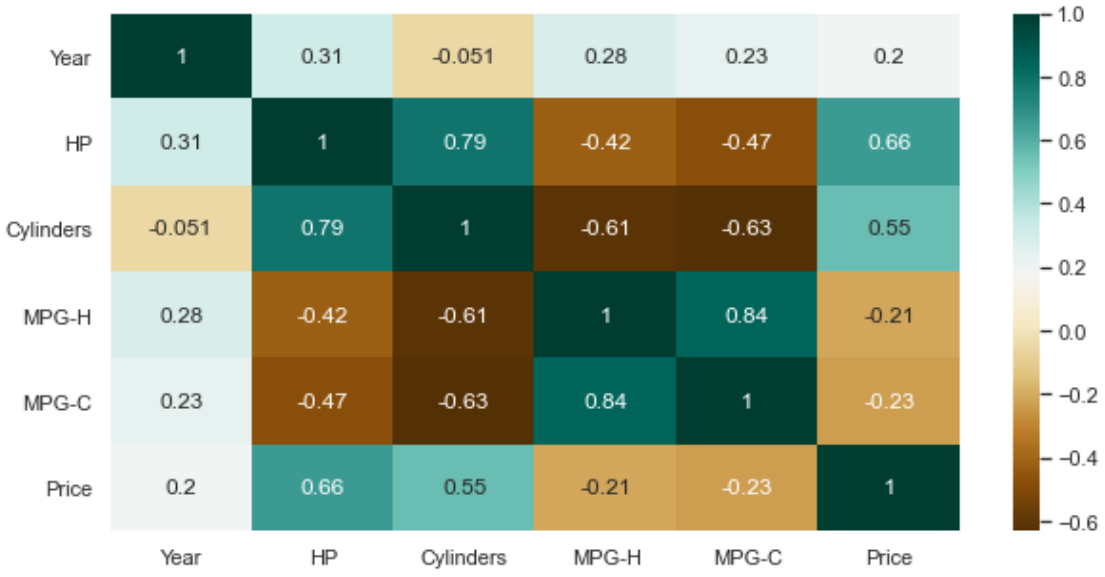


Heat Maps

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)
```

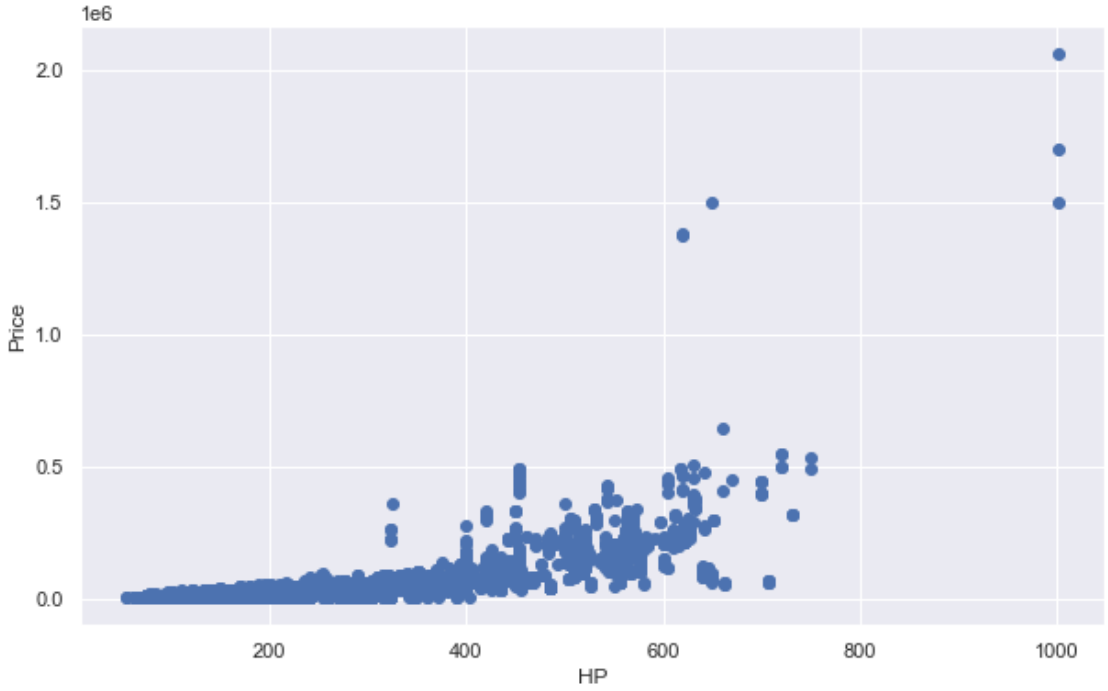
```
<AxesSubplot:>
```



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()
```


1e6
2.0
1.5
1.0
0.5
0.0
Price
200
400
600
800
1000
HP