# Introduction to Data Visualization with Seaborn

Here we will learn how to create various kinds of plots using one of Python's most efficient libraries built especially for data visualization.

Data visualization which helps us to present our analysis from any data which we analyze is primarily performed using Matplotlib which is a very strong and comprehensive library for performing such tasks. But one of the points where Matplotlib suffers is the length of the code which can increase to significant extent while building small representations.

This is where Seaborn comes as our saviour. Seaborn is utilised for plotting of some of the most pleasing data visualization representations.

## Introduction to Seaborn

Around the globe, Seaborn is known for its ability to make statistical graphs in Python. Matplotlib is the language which acts as the basic building block for Seaborn along with Pandas.

Salient features which make it so loveable are as follows:

1. Seaborn is used to visualize univariate or bivariate distributions and even for comparing them.
2. It simplifies the representations of complex datasets.
3. Seaborn provides us with the control over matplotlib's figure styling through various in-built themes which it possess.
4. Through seaborn, we can choose amongst variety of color palettes for making our plots much more conclusive to the viewer.
5. Using seaborn, we have the facility of representation of categorical values.

Overall the usage of dataframes and arrays makes seaborn much more efficient with whole datasets and it produces plots which consist of lots of information. One thing we must remember is that Seaborn is not something which can act as a substitute for Matplotlib but it can definitely act as a complement for Matplotlib and improve upon the lacking points of it, thus presenting a holistic view.

## Seaborn Tutorial Contents

In this tutorial, we'll look at some of the most essential plots which are built using Seaborn library. All the way through this tutorial, we'll try to learn the concepts using examples and its explanation.

## The steps for covering this tutorial:

1. What is Data Visualization
2. Why Data Visualization
3. Installing Seaborn
4. Importing libraries and dataset
5. Different types of plots/graphs which are build using Seaborn
   - Bar Plot
   - Box Plot
   - Scatter Plot
   - Violin Plot
   - Swarm Plot
   - Overlaying Swarm Plot with Violin plot
   - Joint Kernel Density Plot
   - Joint Plot
   - Heatmap
   - Clustermap

## Step 1: What is Data Visualization?

**Data Visualization** can be defined as a process of extracting essential information from raw/processed data and then representing it pictorially for better understanding and analysis of the facts/figures. Data Visualization is an amalgamation of two fields i.e. Science and Art, this means we are applying our scientific and art skills in the making of any kind of visualizations.

To be more precise, data visualization is a strategy of depicting the quantitative knowledge obtained through various data wrangling processes in graphical manner.

The main objective of data visualization is to decide the most optimal way of presenting a data set with the data visualization practices in mind and all these tasks become much more difficult when we have to deal with large datasets.

## Step 2: Why Data Visualization is required?

In recent times, the amount of data which is generated is gigantic and we want it to be of some significance. According to reports, the data which revolves around the internet every second is more than were stored in the entire internet in past 20 years.

This is because as people are using internet for building connections not only amongst themselves but also for connecting machines the data will grow exponentially.

The human mind is not competent enough to deal with such data and thus we need some kind of tool which can help us to understand profoundly what all such data is trying to convey. For these purposes only data visualization is used since it not only reduces the size of large datasets but is able to represent the useful information very easily in the form of graphs which also aids in delivering the content to the viewer.

There are numerous plots which are used in Data Visualization such as Histograms, Pie Chart, Box Plot, Word Cloud, Scatter plot etc. there is a long list of such graphs and most of them we'll see with examples very soon in this tutorial.

## Step 3: Installing Seaborn

As we will be working with Seaborn, an in-built library of python. We are required to install it on our system on which we are working. Along with Seaborn, as I have worked with Jupyter Notebook for this tutorial I would recommend to use it as well.

You can go for following installations if you wish:

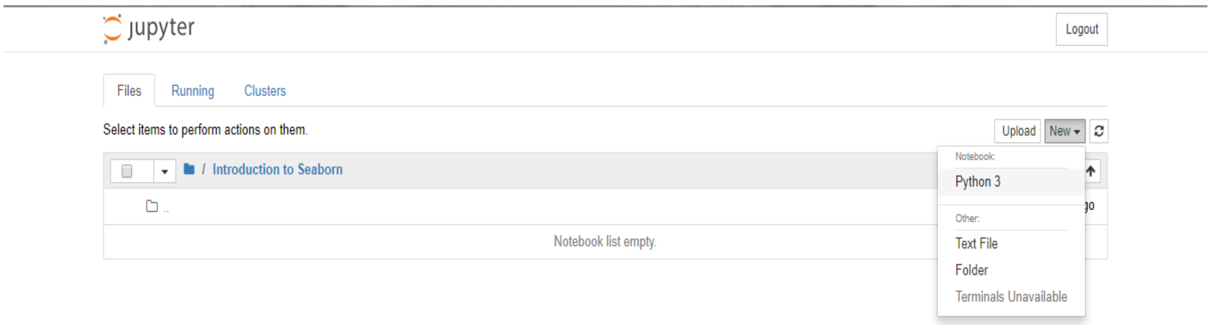
1. Python 2.7/ Python 3(Preferred)
2. Python Libraries for Data Visualization
   - Pandas
   - Matplotlib
   - Seaborn
3. Jupyter Notebook.

You can download and install Anaconda Distribution as it comes along with the packages which are required. You only have to follow the steps on that page.

If you have python and you want to install seaborn in your system, you can execute the following and then it will help you to install the latest version of Seaborn in your system.

```
sudo pip install seaborn
```

If you are opting to work upon Jupyter notebook. Then, once you have installed Anaconda, open Jupyter Notebook (either through command line or navigator app) and create a new notebook.



## Step 4: Importing Libraries and dataset.

Initially we will import Pandas which will be used for handling relational datasets.

```
#Pandas for managing datasets
import pandas as pd
```

After this we have to import Matplotlib which will aid in making changes to the plots which we will be created using Seaborn.

```
#Matplotlib for additional customization
import matplotlib.pyplot as plt
%matplotlib inline
```

**NOTE:** While using Jupyter Notebook, we write %matplotlib inline for displaying the plots in the notebook.

Finally we will import the library which is the base of this tutorial i.e. Seaborn.

```
#Seaborn as our primary library for plotting and styling
import seaborn as sns
```

Now we are all set to import the dataset which we will be using for Visualization purposes.

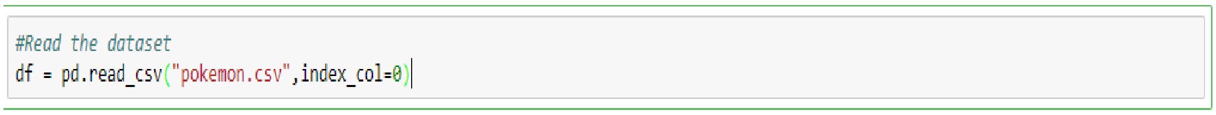
**NOTE:** We have used aliases for the imported libraries such as pd for pandas, plt for Matplotlib and sns for Seaborn. This is actually done for our convenience, so that we can invoke these libraries and its methods using these aliases and we will not be required to write the full name of the library.

The dataset for this tutorial is a very interesting one i.e. Pokémon dataset. You can download it for free from here.

Pokemon.csv

After having downloaded the dataset, remember to keep the dataset (csv file) in the same folder where your python file/ Jupyter notebook is present, as there will no issues of providing the location of the dataset.

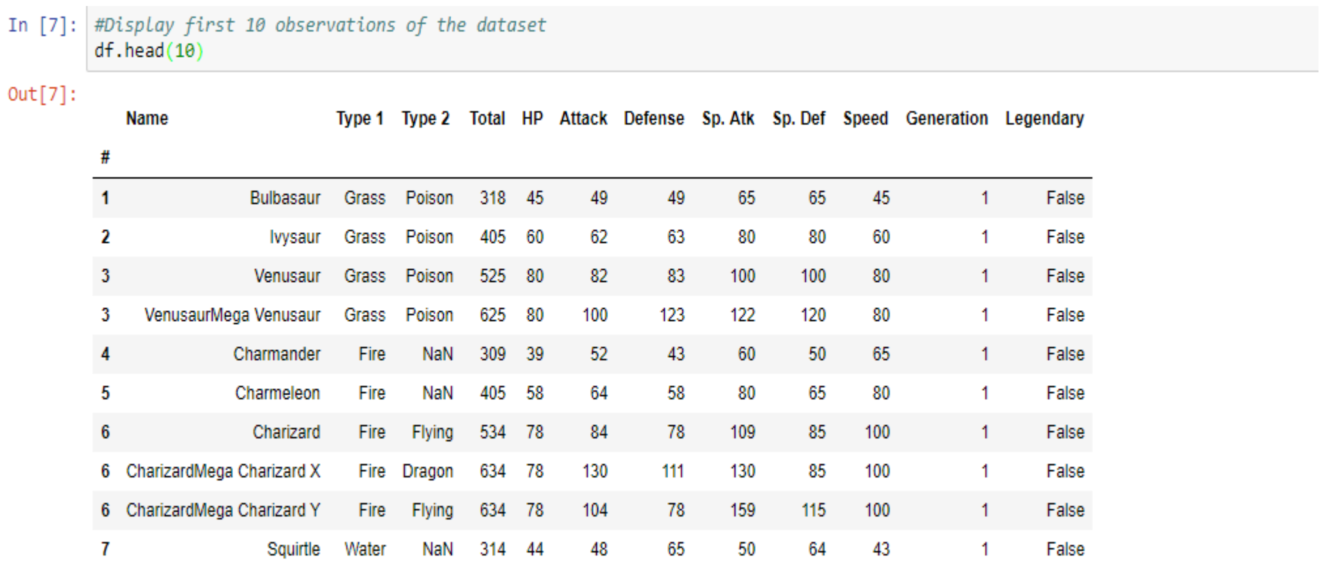
Now to import the dataset we have to execute the following code.



```
#Read the dataset
df = pd.read_csv("pokemon.csv",index_col=0)
```

**NOTE:** Here the index_col=0 argument tells that we will be using the first column of the dataset as the ID column.

Once we have imported the dataset, we can view the values of the dataset and see how the dataset looks like by the head () function of df (dataframe). We can pass different values to the head () function for viewing more or less values. The default value is 5.

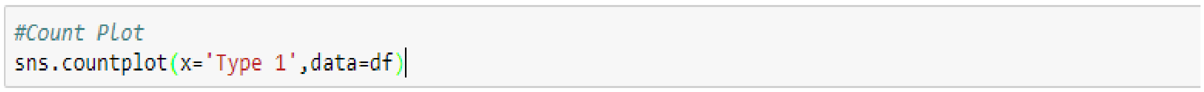


```
In [7]: #Display first 10 observations of the dataset
        df.head(10)
Out[7]:
```

| | Name | Type 1 | Type 2 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#** | | | | | | | | | | | | |
| **1** | Bulbasaur | Grass | Poison | 318 | 45 | 49 | 49 | 65 | 65 | 45 | 1 | False |
| **2** | Ivysaur | Grass | Poison | 405 | 60 | 62 | 63 | 80 | 80 | 60 | 1 | False |
| **3** | Venusaur | Grass | Poison | 525 | 80 | 82 | 83 | 100 | 100 | 80 | 1 | False |
| **3** | VenusaurMega Venusaur | Grass | Poison | 625 | 80 | 100 | 123 | 122 | 120 | 80 | 1 | False |
| **4** | Charmander | Fire | NaN | 309 | 39 | 52 | 43 | 60 | 50 | 65 | 1 | False |
| **5** | Charmeleon | Fire | NaN | 405 | 58 | 64 | 58 | 80 | 65 | 80 | 1 | False |
| **6** | Charizard | Fire | Flying | 534 | 78 | 84 | 78 | 109 | 85 | 100 | 1 | False |
| **6** | CharizardMega Charizard X | Fire | Dragon | 634 | 78 | 130 | 111 | 130 | 85 | 100 | 1 | False |
| **6** | CharizardMega Charizard Y | Fire | Flying | 634 | 78 | 104 | 78 | 159 | 115 | 100 | 1 | False |
| **7** | Squirtle | Water | NaN | 314 | 44 | 48 | 65 | 50 | 64 | 43 | 1 | False |

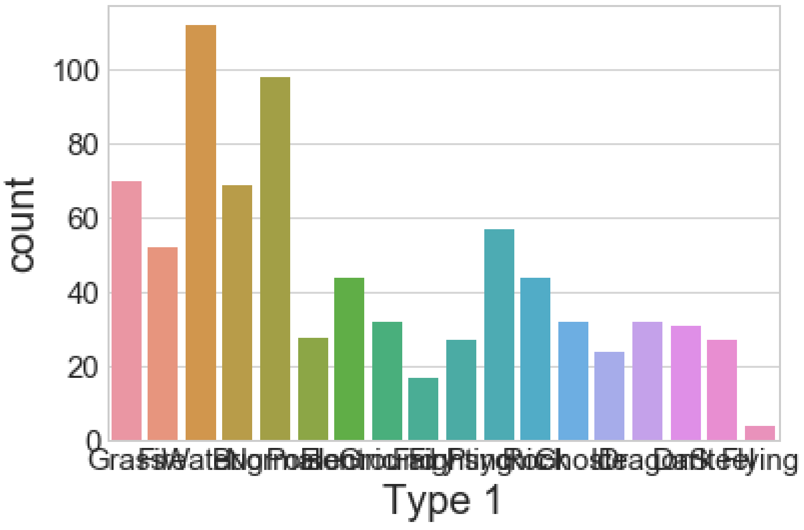
**<u>Step 5</u>:** Let's start plotting various graphs and visualizations.

## Bar Plot

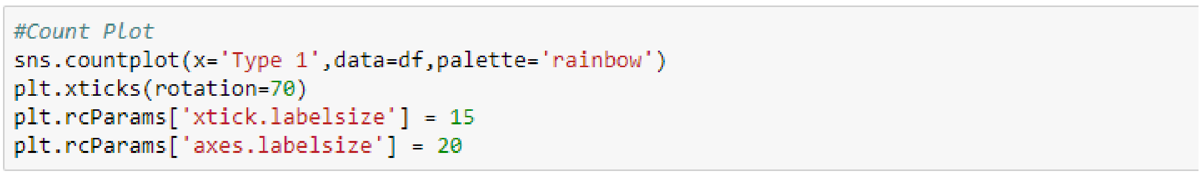


```
#Count Plot
sns.countplot(x='Type 1',data=df)
```

We use the in-built function of seaborn i.e. countplot() for  plotting the bar graph where we have provided the 'Type 1' as the value for x-axis and 'df' as the value for data.



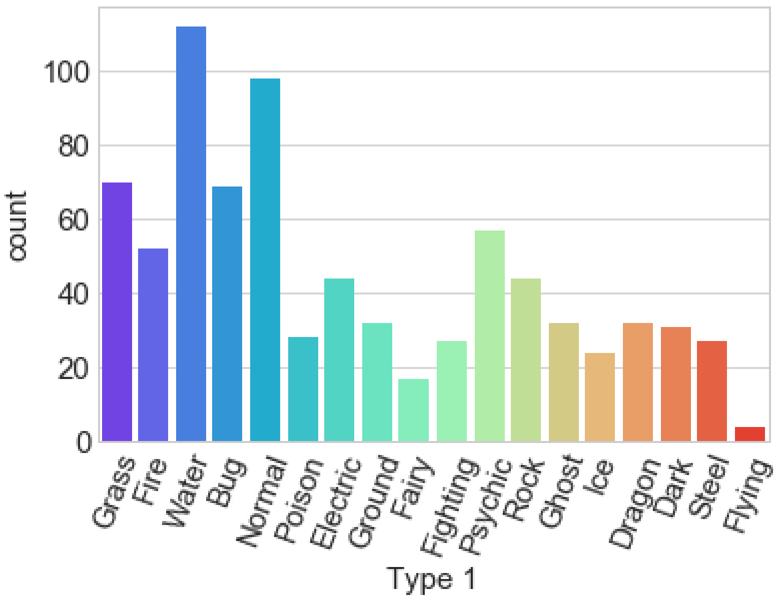
This is the output we obtain when we execute the above code, where x-axis as Type 1 and y-axis is labelled as count. But you must have noticed that the x-axis values are not visible due to the lack of space. For getting rid of this problem we can try the following.



```
#Count Plot
sns.countplot(x='Type 1',data=df,palette='rainbow')
plt.xticks(rotation=70)
plt.rcParams['xtick.labelsize'] = 15
plt.rcParams['axes.labelsize'] = 20
```

Here we have made some additions to the previous code so that we can get a clean output of the bar plot as compared to previous one. In this, you can see we have used matplotlib's 'xticks' method in which we have set the value of 'rotation' as 70 which will tilt the x-axis values by 70 degrees making it clearly visible. Moreover, we have passed another argument in countplot i.e. palette and its value as 'rainbow' this will present the color of bars in rainbow colours.

Along with this, we have used 'rcParams' method of matplotlib to increase the size of x-axis values and labels of x-axis.



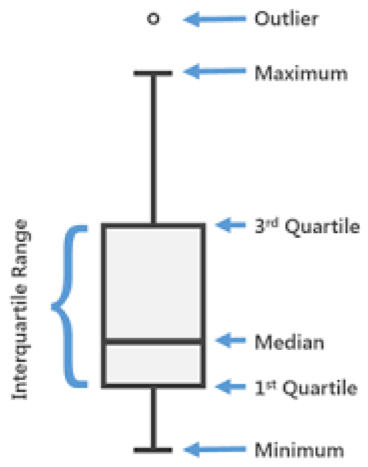
We can clearly see the values of x-axis are presented neatly and the labels of both the axes are also visible clearly.

### Uses:

A Bar Plot is used to represent comparison between categories of data. It can be represented either in vertical/horizontal manner.

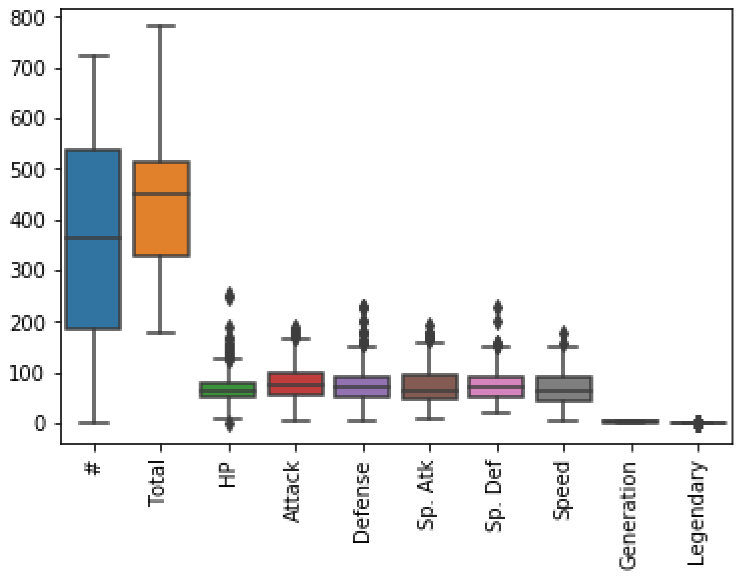
## Box Plot

A box plot is the visual representation statistical five number summary of a given data set i.e. Minimum, First Quartile, Median, Third Quartile and Maximum.



*Box Plot representation*

```
#Boxplot
sns.boxplot(data=df)
plt.xticks(rotation=90)
```

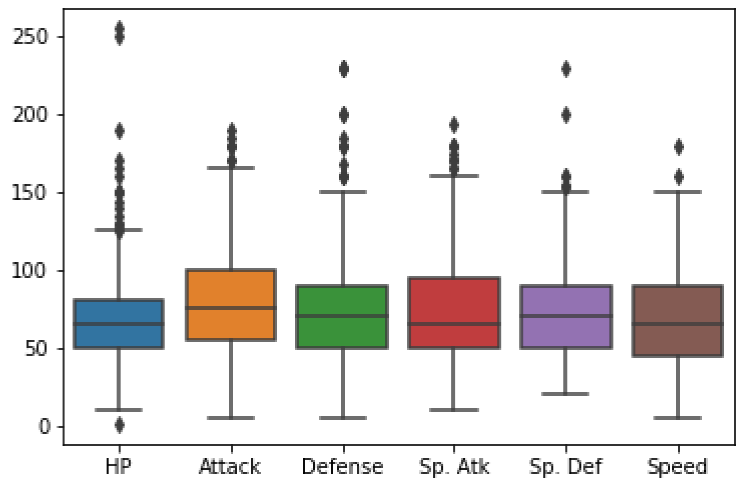


For plotting the boxplot we have used the boxplot() function of seaborn, but we can see that for some values the result obtained is insignificant, thus we will have to remove all those columns which are redundant like 'Total' as we have the individual stats and the one's which are not combat i.e. 'Legendary ' and 'Generation'.

**NOTE:** The black dots which are visible are the outliers in the dataset.

```
df = df.drop(['#','Total','Legendary','Generation'],axis=1)
sns.boxplot(data=df)
```

Now we have removed the ‘#’, ‘Total’, ‘Legendary’ and ‘Generation’ column from the dataset by using the drop() method of df and also specified the axis for identifying the x-axis where they are located.



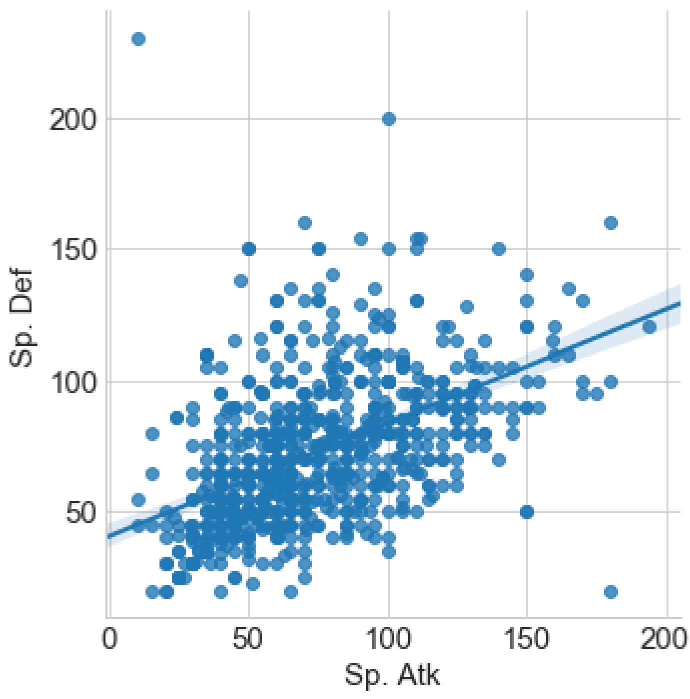
## Uses:

It is used in exploratory data analysis. Through this, we can represent the shape of the distribution of data through it.

## Scatter Plot

```
sns.lmplot(x="Sp. Atk",y="Sp. Def", data=df)
```

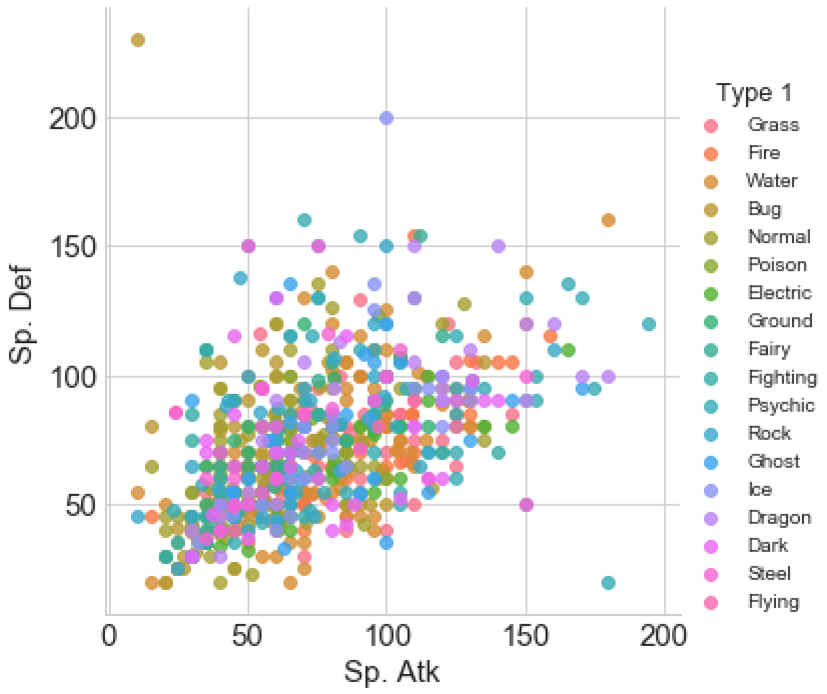
Here we have used the lmplot () function of seaborn for creating the scatter plot where we have provided values of x-axis and y-axis as ‘Sp. Atk’ and ‘Sp. Def’ respectively and provided the ‘data’ parameter with value ‘df’



It is evident that we were looking to plot scatter plot but we have also obtained the regression line. This is because Seaborn has no specific scatter plot function, reason why we see a diagonal line. The function ‘lmplot ()’ is used to fit and plot the regression line.

```
sns.lmplot(x="Sp. Atk",y="Sp. Def", data=df,hue='Type 1',fit_reg=False)
```

To solve the regression line issue, we can use ‘fit_reg’ argument and set it as ‘False’. Moreover, we can use the ‘hue’ parameter which will help us to present the points on the graph with much more clarity i.e. we represent the third dimension of information using ‘Type 1’ as the value for ‘hue’.



We can clearly view different generations of Pokémon and their special attack and special defense values through this scatter plot.

### Uses:

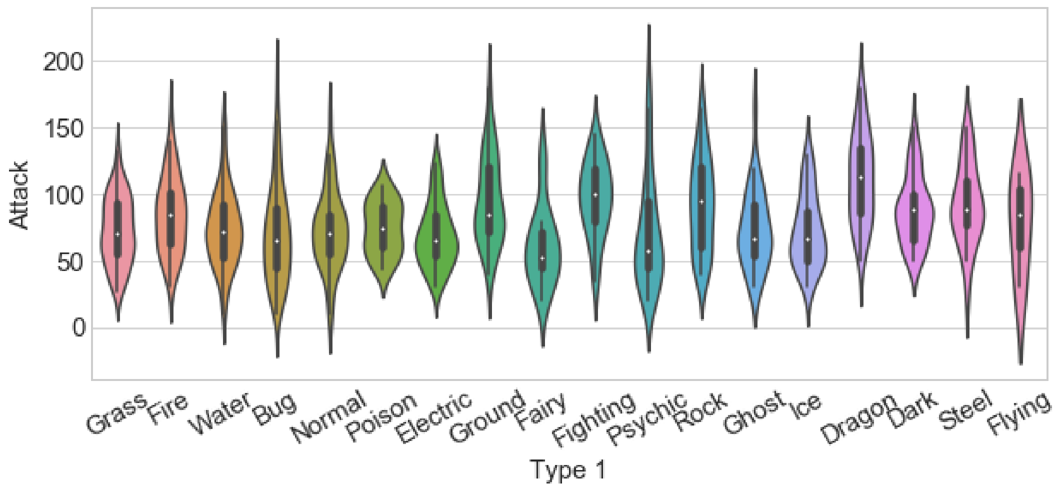
Scatter plot helps us to find potential relationships between values and they can help in detecting outliers in the datasets. They can simplify the large datasets representation easily.

## Violin Plot

```
#Setting the theme
sns.set_style('whitegrid')
plt.figure(figsize=(10,4))

#Violin Plot
vio_plot = sns.violinplot(x='Type 1', y='Attack',data=df)
plt.xticks(rotation=30)
plt.rcParams['xtick.labelsize'] = 15
plt.rcParams["axes.labelsize"] = 15
```

Violin plot is built using seaborn's violinplot () function. Before this, we can set the background style of the violin plot by the 'set_style' method which has been given the value as ‘whitegrid’, there are other values as well like ‘dark’, ‘white’, ‘darkgrid’ and ‘ticks’.

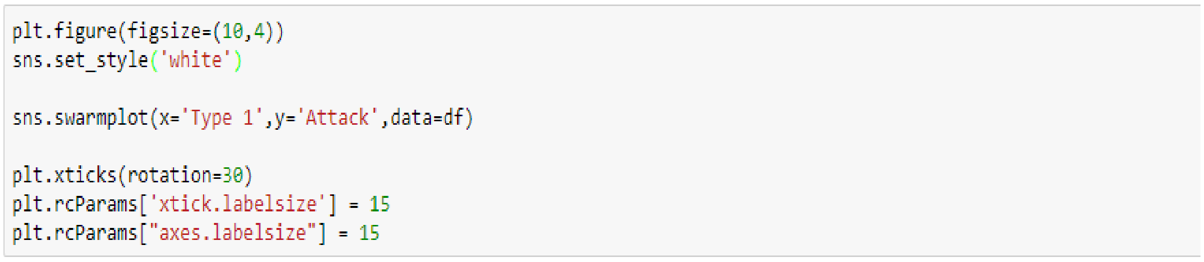
Then, we can pass the x-axis and y-axis values and then customizing the plot using matplotlib methods.



## Uses:

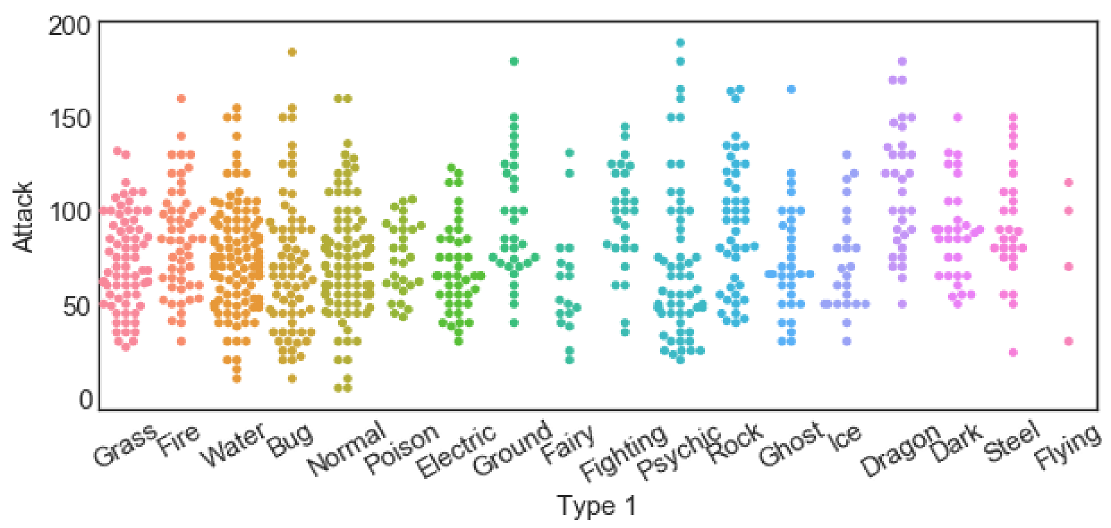
They help in representing fantastically huge amount of information effectively in small amount of space.

## Swarm Plot



```
plt.figure(figsize=(10,4))
sns.set_style('white')

sns.swarmplot(x='Type 1',y='Attack',data=df)

plt.xticks(rotation=30)
plt.rcParams['xtick.labelsize'] = 15
plt.rcParams["axes.labelsize"] = 15
```

Now for plotting swarm plot, we use the ‘swarmplot ()’ function of seaborn with ‘Type 1’ and ‘Attack’ as the values. Initially, we have set the size of our swarm plot using ‘figure ()’ function.



**Uses:**

It gives a better representation of distribution of values. It can be built on its own but is also a good complement to a box or violin plot.

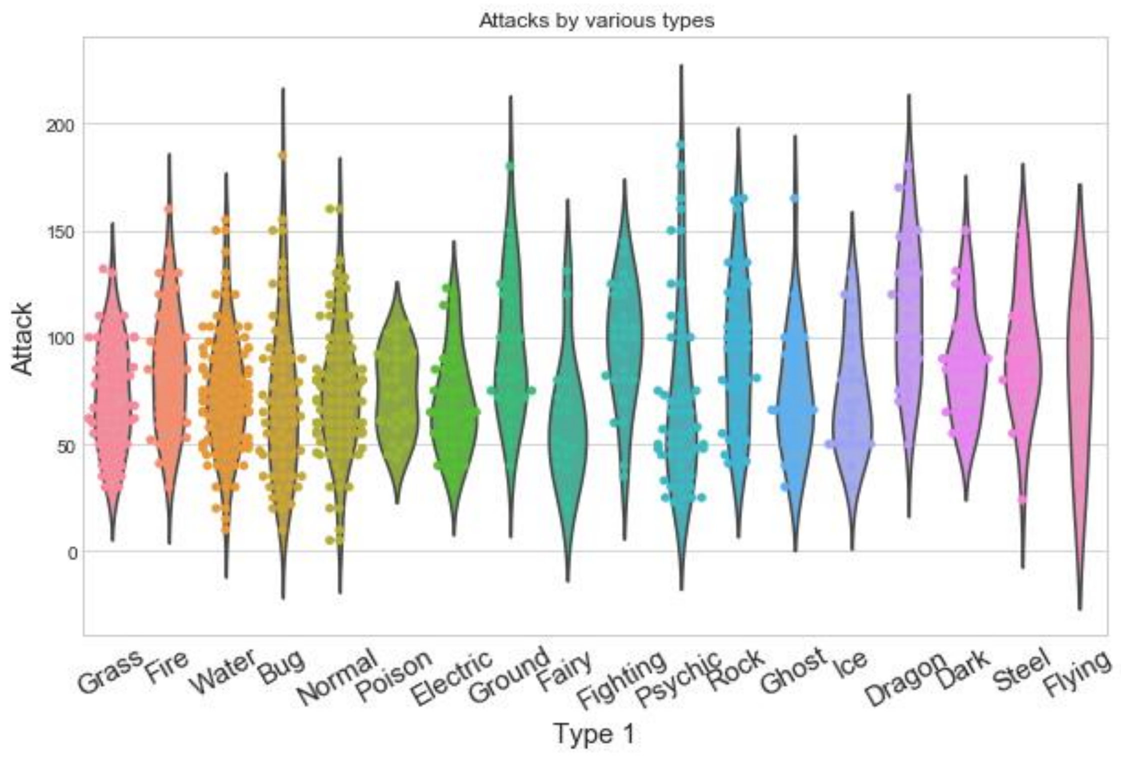
## Overlaying swarm and violin plots

Through overlaying of swarm and violin plots on similar values can help us to analyse in a much more efficient manner.

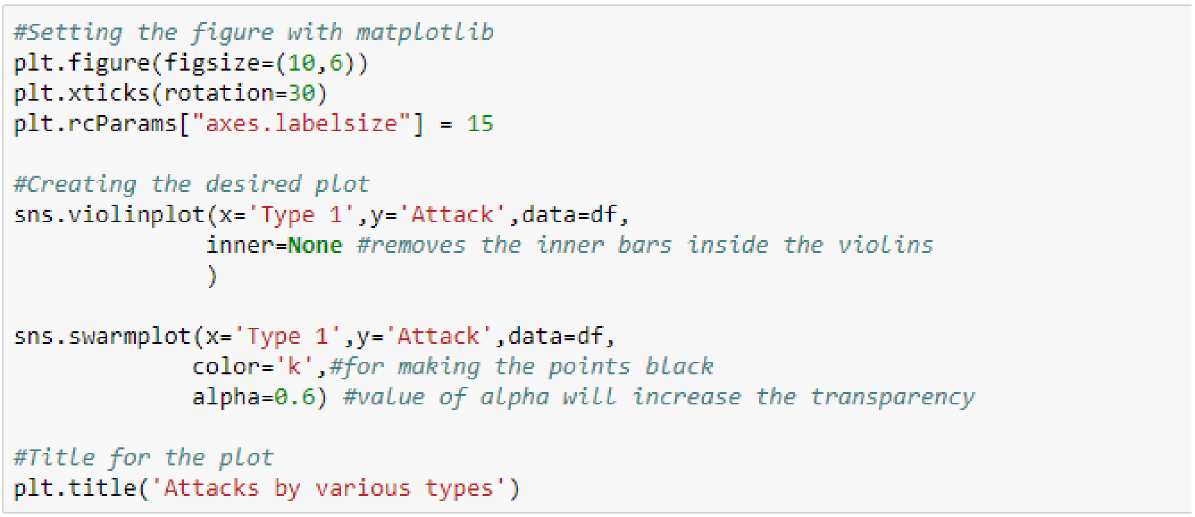
```
#Setting the figure with matplotlib
plt.figure(figsize=(10,6))
plt.xticks(rotation=30)
plt.rcParams["axes.labelsize"] = 15

#Creating the desired plot
sns.violinplot(x='Type 1',y='Attack',data=df,
               inner=None #removes the inner bars inside the violins
               )

sns.swarmplot(x='Type 1',y='Attack',data=df)

#Title for the plot
plt.title('Attacks by various types')
```

First to make this plot, we will make the figure a bit larger by using the figure () function of matplotlib. After this, we have used violinplot () function for plotting the violin plot using the 'Type 1' and 'Attack' values and data is given 'df' as value. We have also used 'inner' argument with 'None' as the value for removing the inner bars which are inside the violin.

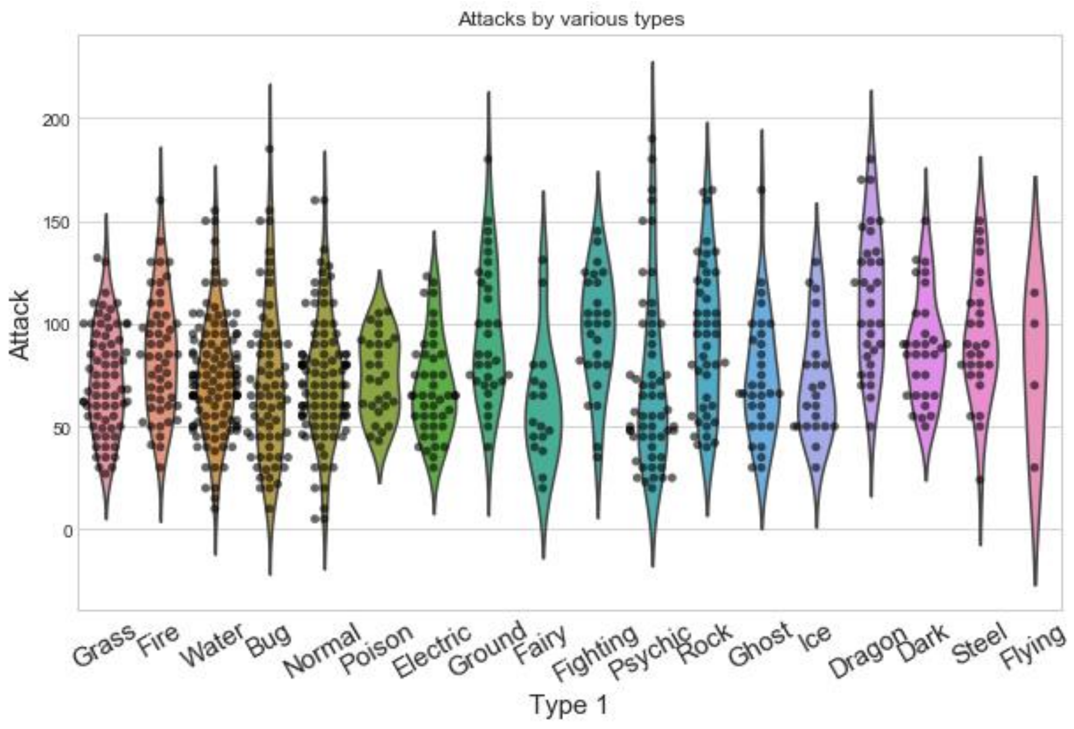
Next we will plot the swarm plot using swarmplot () function with the same values for x-axis and y-axis.  We have also provided the title of the plot using title () function of matplotlib.



Here in this plot, we can see the swarm plot over violin plot. But the points of swarm plot is not clearly visible, so we will try to remove this using our following code:


```
#Setting the figure with matplotlib
plt.figure(figsize=(10,6))
plt.xticks(rotation=30)
plt.rcParams["axes.labelsize"] = 15

#Creating the desired plot
sns.violinplot(x='Type 1',y='Attack',data=df,
              inner=None #removes the inner bars inside the violins
              )

sns.swarmplot(x='Type 1',y='Attack',data=df,
             color='k',#for making the points black
             alpha=0.6) #value of alpha will increase the transparency

#Title for the plot
plt.title('Attacks by various types')
```

In the above code, to make the swarm plot points visible we will be using two arguments i.e. ‘color’ and ‘alpha’. We have specified the color value as ‘k’ which is represents black and 'alpha' is used to increase/decrease the transparency which is why we have given the value as '0.6'



Now we can clearly see the points of swarm plot over the violin plots. Therefore, this is our final plot where the swarm plot is over the violin plot.

**Uses:**

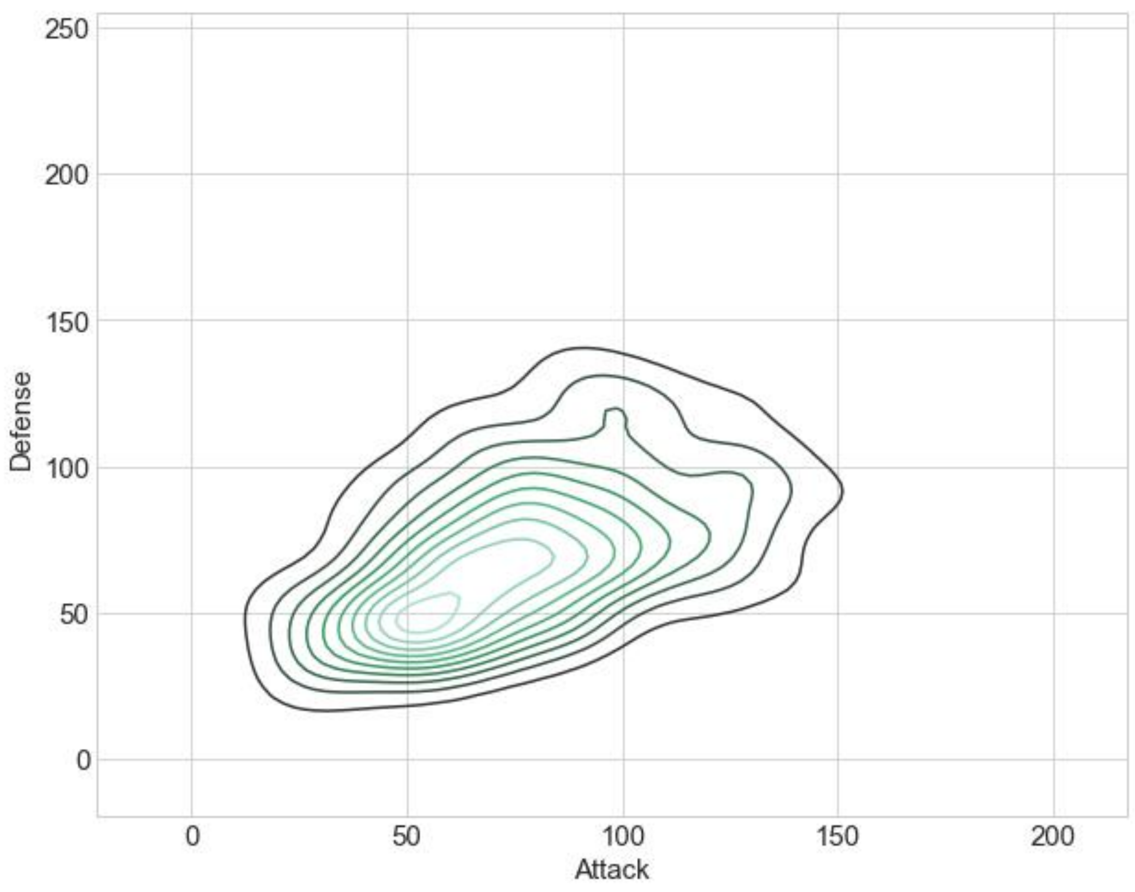
It helps in visualizing the distribution of data and the probability density.

## Joint Kernel Density Plot

In this we get to know how much the data is scattered for the two columns which are under consideration.

```
sns.set_style('whitegrid')

plt.figure(figsize=(10,8))
plt.rcParams['xtick.labelsize'] = 15
plt.rcParams['ytick.labelsize'] = 15

sns.kdeplot(df.Attack,df.Defense)
```

For plotting the joint kernel density plot, we proceed with the styling which is done through seaborn and matplotlib. After that, we will use the kdeplot () function of seaborn. Here we can see that the arguments to the kdeplot () is passed differently as compared to other plotting functions. Here, we use 'dataframe (df)' to call the values for both the axes.



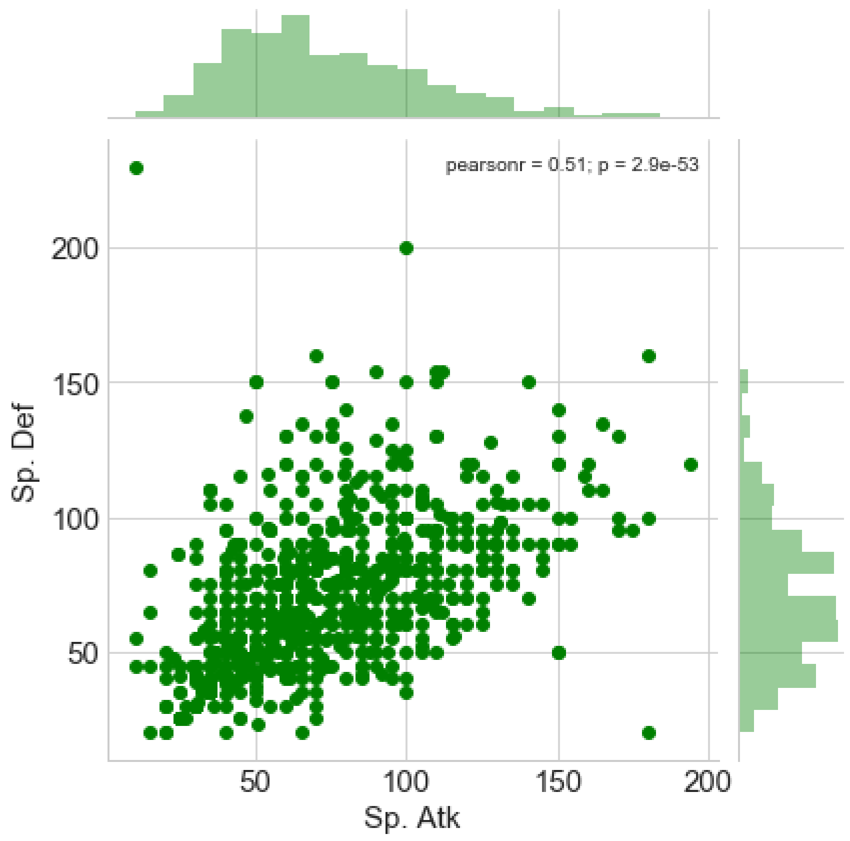
**Uses**:

It helps in finding the probability density function of any dataset. This can help in smoothing the data around values of PDF.

## Joint Plot

```
sns.jointplot(x="Sp. Atk",y='Sp. Def',data=df,kind='scatter',color='g')
```

Joint plot is build using the jointplot () function of seaborn where we provide the values of x-axis and y-axis along with this we give the argument 'kind' for specifying the plot which we are creating jointly, here we have given the value as 'scatter' and we have even specified the 'color' value as 'g' i.e. green. There are other values for 'kind' which are 'reg', 'resid', 'hex' etc.



**Uses:**

Through joint plot, we get the liberty to use two plots for representation of the same data which helps in better analysis.

## Heatmap

For plotting Heatmap we will be using a different dataset i.e. 'flights.csv' which is an in-built dataset in Seaborn library and we will be load this dataset using seaborn itself.

- Loading the dataset using Seaborn

```
flights = sns.load_dataset('flights')
```

Here we have used the load_dataset function to load the 'flights' dataset for Visualization.

- Displaying the flights dataset.

```
flights.head()
```

| | year | month | passengers |
|---|---|---|---|
| **0** | 1949 | January | 112 |
| **1** | 1949 | February | 118 |
| **2** | 1949 | March | 132 |
| **3** | 1949 | April | 129 |
| **4** | 1949 | May | 121 |

- Creating pivot table.

```
flights = flights.pivot('month','year','passengers')
```

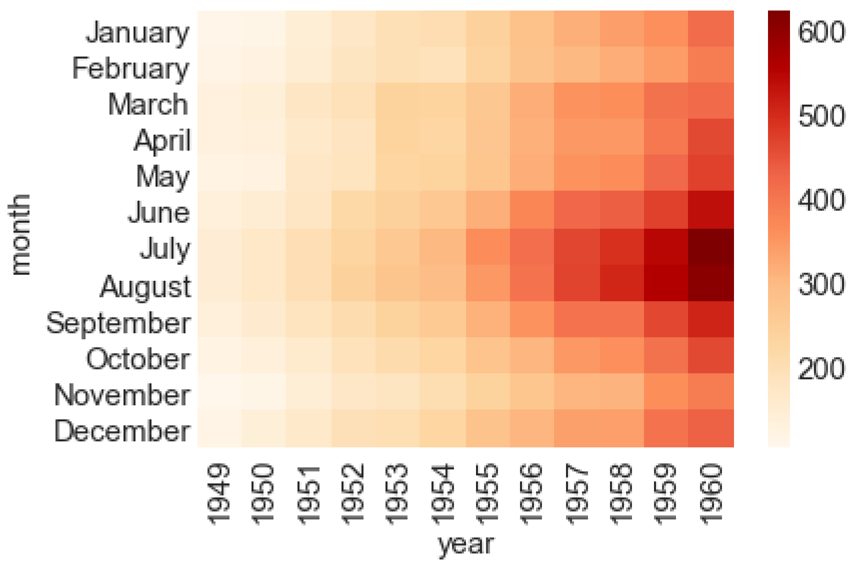
To plot the Heatmap, we will be required to draw correlation between the columns which is done through the pivot () function where we have passed month and year as x-axis and y-axis values respectively and passengers for range.

- Plotting the Heatmap

```
sns.heatmap(flights,cmap="OrRd")
```

Now to plot the Heatmap, we use the heatmap () function of Seaborn where we have passed the dataset flights as one argument and color of the Heatmap as ‘OrRd’ i.e. Orange and Red.

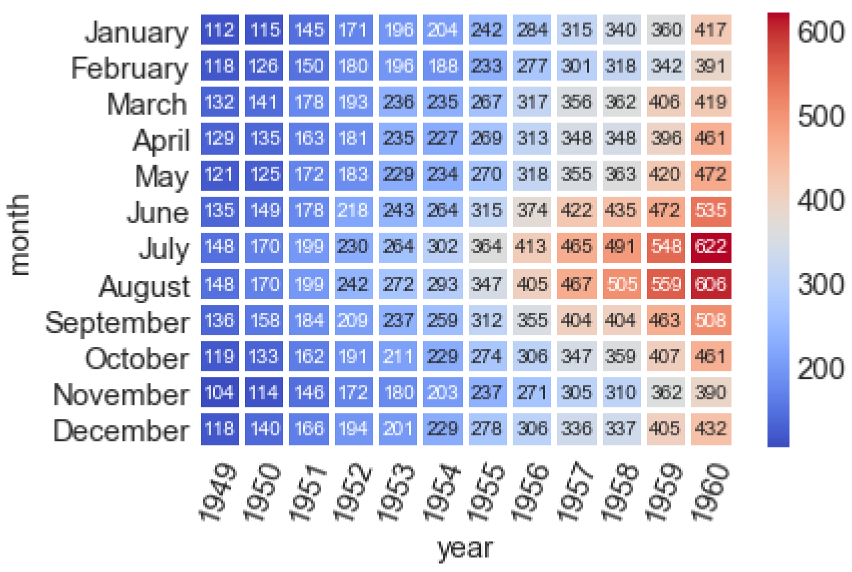
**NOTE:** There are many cmap values which can be passed for trying out.



In heatmap we can see as the value of range gets higher the intensity of color increases and for lesser values the color is lighter in shade.

```
sns.heatmap(flights,cmap="coolwarm",annot=True,fmt='d',linewidths=2)
plt.xticks(rotation=70)
```

To make the heatmap more informative we can add some more arguments to the heatmap () function. The values displayed in each cell is represented by 'annot' argument which is set as 'True' and for displaying those values in decimal format we use the 'fmt' argument. Lastly, to make it more neat we will draw borders between the lines using 'linewidths' argument, we can specify different values for it as per our requirements.



**Uses:**

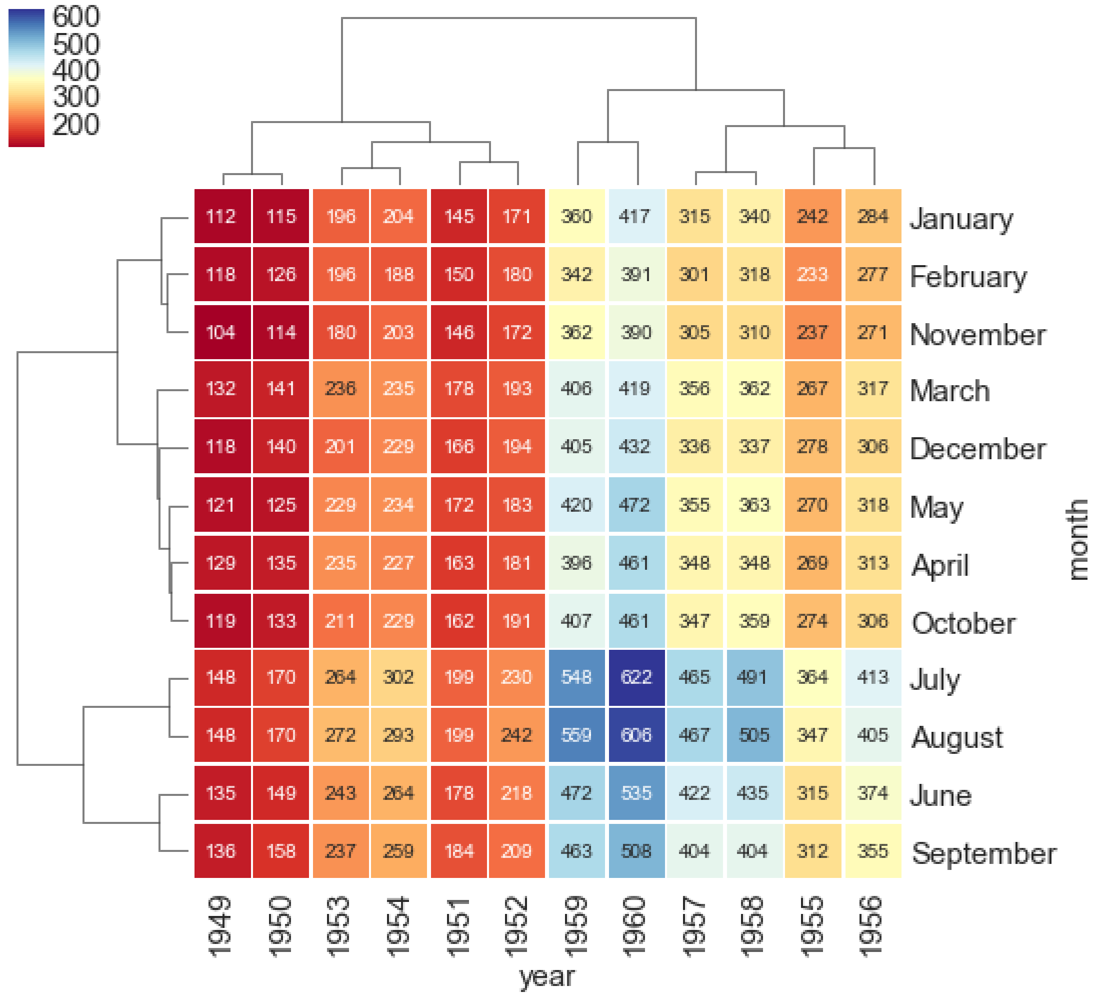
Since it is 2-dimensional figure, it helps in visualizing complex data. We can even use it to represent large datasets.

## Clustermap

```
plt.figure(figsize=(6,4))
sns.clustermap(flights,cmap='RdYlBu',linewidths=0.5,figsize=(8,8),annot=True,fmt='d')
```

So we'll end this tutorial with another graph i.e. Clustermap which is built using the clustermap () function of seaborn. Clustermap is very similar to heatmap, the only difference is that the similar values are combined together and represented in clustermap, which is evident from its name.

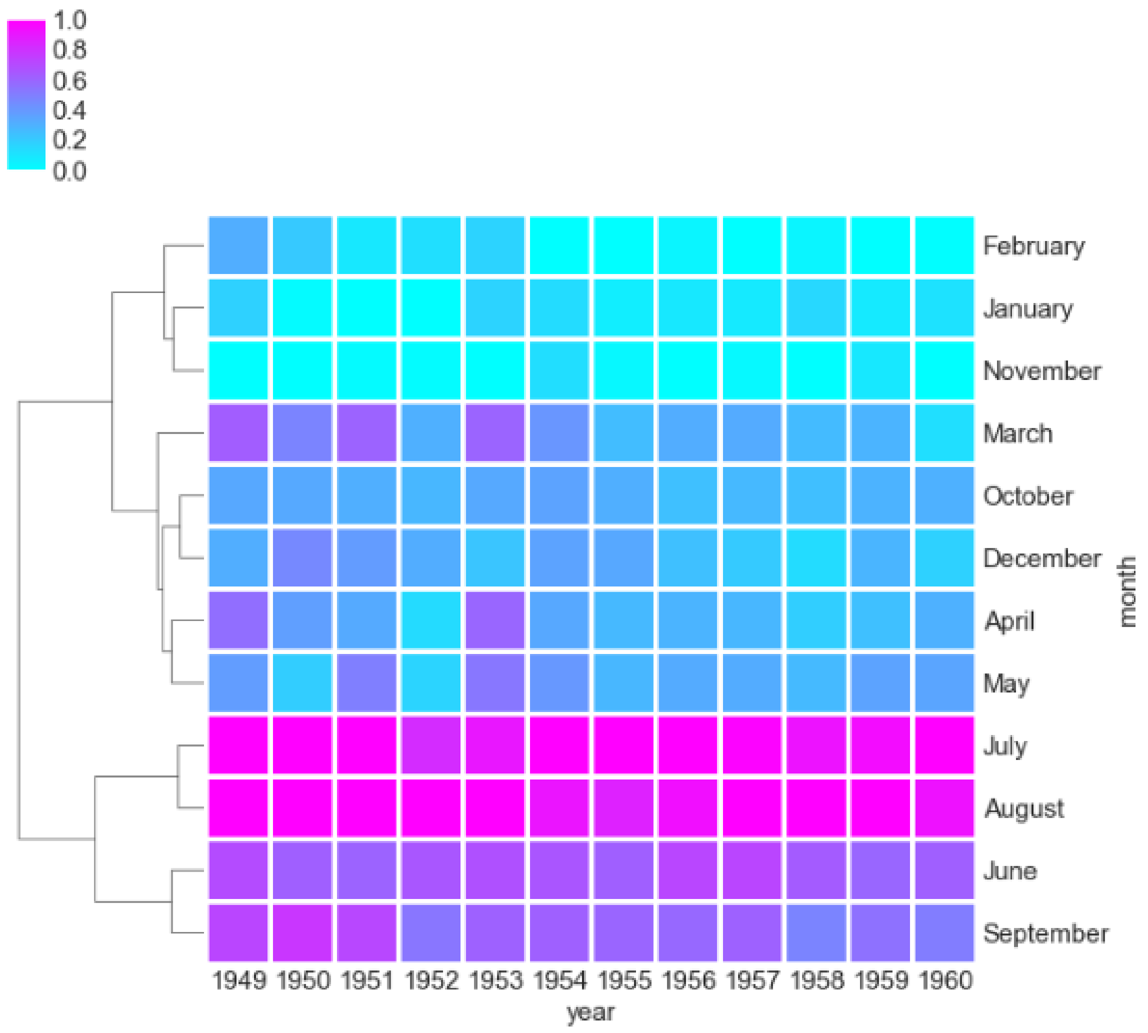
Here again, we have use matplotlib for designing the figure. The other arguments are similar to the previous heatmap example.



Here you can see the months are not in the known order, this is because the clustermap has combined the months on the basis of similar values i.e. clusters. Similarly, the years are clustered on the same criteria.

```
sns.clustermap(flights,col_cluster=False,cmap='cool',linewidths=2,standard_scale =1)
```

Even we can control the clustering in clustermap, this can be done by using 'col_cluster'/'row_cluster' arguments which are given values as true/false. In the above example we have set col_cluster as 'False' resulting in no clustering for columns. Secondly, we have used standard_scale for standardising our dataset either by columns or rows, when we give value as '0' the data across rows will be standardised and if we pass '1' the column data will be standardised.



Here at the top-left corner we can see the range has been changed (200 – 600 in previous example) and has been made from 0.0 to 1.0, so this how the values are standardised.

## Uses:

With the ability to standardise data, we can represent datasets through cluster map which have columns with huge differences making it easier for analysis.

So finally we have reached the end of this Introduction to Data Visualization with Seaborn, I hope you would have liked it and most importantly learned from it.

By Palash Sharma.