Seaborn in Python

What is Seaborn?

Have you ever used the ggplot2 library in R? It's one of the best visualization packages in any tool or language. Seaborn gives the same overall feel.
It gives us the capability to create amplified data visuals. This helps us understand the data by displaying it in a visual context to unearth any hidden correlations between variables or trends that might not be obvious initially. Seaborn has a high-level interface as compared to the low level of Matplotlib.

Why should you use Seaborn versus matplotlib?

Seaborn makes our charts and plots look engaging and enables some of the common data visualization needs (like mapping color to a variable or using faceting). Basically, it makes the data visualization and exploration easy to conquer.
There are essentially a couple of (big) limitations in matplotlib that Seaborn fixes:

1. Seaborn comes with a large number of high-level interfaces and customized themes that matplotlib lacks as it's not easy to figure out the settings that make plots attractive
2. Matplotlib functions don't work well with dataframes, whereas seaborn does

Setting up the Environment

The seaborn library has four mandatory dependencies you need to have:

- NumPy (>= 1.9.3)
- SciPy (>= 0.14.0)
- matplotlib (>= 1.4.3)
- Pandas (>= 0.15.2)

To install Seaborn and use it effectively, first, we need to install the aforementioned dependencies. Once this step is done, we are all set to install Seaborn and enjoy its mesmerizing plots. To install Seaborn, you can use the following line of code-
To install the latest release of seaborn, you can use pip:
pip install seaborn

You can also use conda to install the latest version of seaborn:
conda install seaborn
To import the dependencies and seaborn itself in your code, you can use the following code-
import numpy as np

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from scipy import stats
```

That's it! We are all set to explore seaborn in detail.

Datasets Used for Data Visualization

We'll be working primarily with two datasets:

- Pokemon
- A waiter's tip

I've picked these two because they contain a multitude of variables so we have plenty of options to play around with. Both these datasets also mimic real-world scenarios so you'll get an idea of how data visualization and exploration work in the industry.

Data Visualization using Seaborn

Let's get started! I have divided this implementation section into two categories:

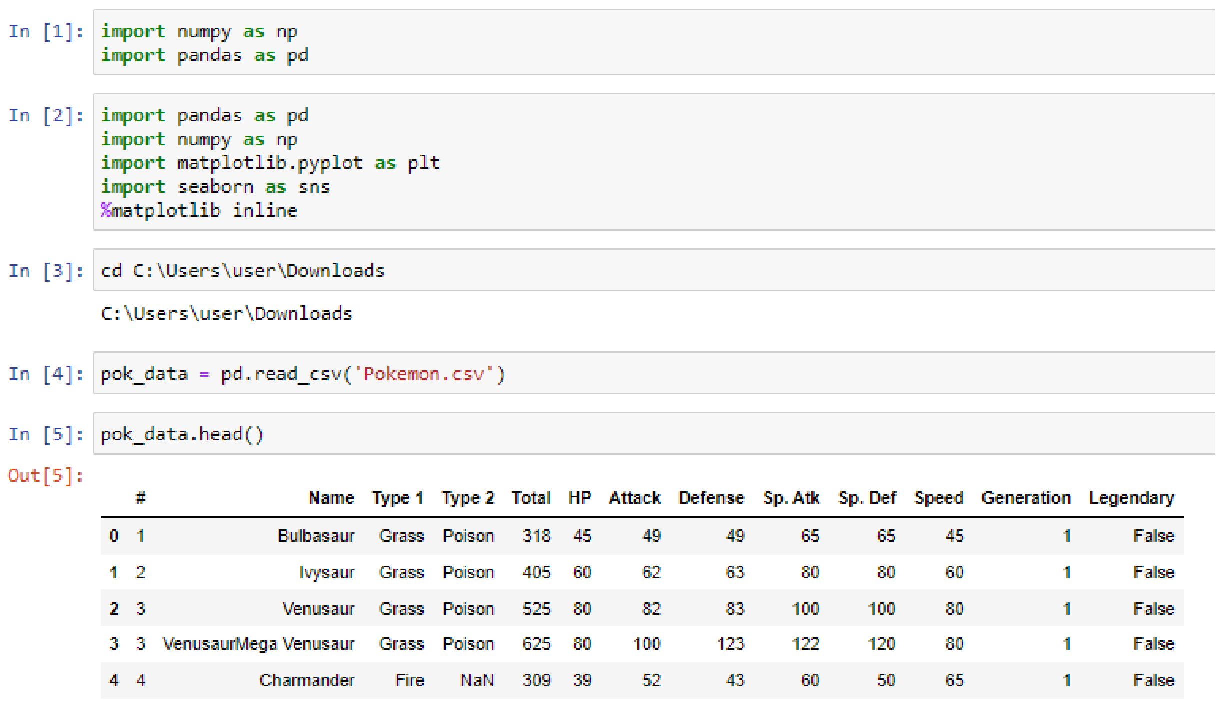
- Visualizing statistical relationships
- Plotting categorical data

We'll look at multiple examples of each category and how to plot it using seaborn.

Visualizing statistical relationships

A statistical relationship denotes a process of understanding relationships between different variables in a dataset and how that relationship affects or depends on other variables.
Here, we'll be using seaborn to generate the below plots:

- Scatter plot
- SNS.relplot
- Hue plot



```
In [1]: import numpy as np
        import pandas as pd
```

```
In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
```

```
In [3]: cd C:\Users\user\Downloads

        C:\Users\user\Downloads
```

```
In [4]: pok_data = pd.read_csv('Pokemon.csv')
```

```
In [5]: pok_data.head()
```

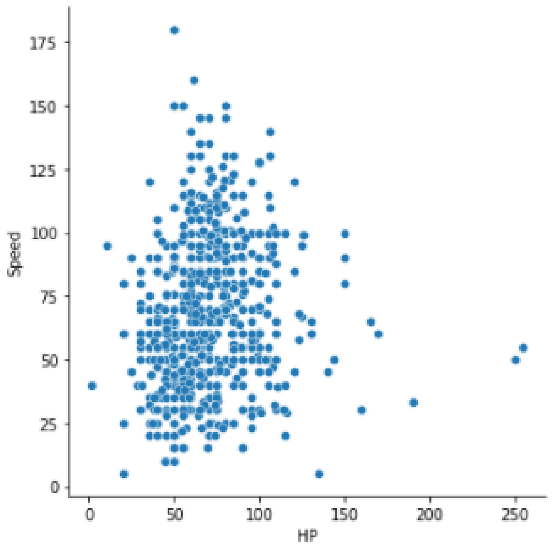
Out[5]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

Scatterplot using Seaborn

A scatterplot is perhaps the most common example of visualizing relationships between two variables. Each point shows an observation in the dataset and these observations are represented by dot-like structures. The plot shows the joint distribution of two variables using a cloud of points.To draw the scatter plot, we'll be using the relplot() function of the seaborn library. It is a figure-level role for visualizing statistical relationships. By default, using a relplot produces a scatter plot:

```
In [7]: sns.relplot(x="HP", y="Speed", data = pok_data)
Out[7]: <seaborn.axisgrid.FacetGrid at 0x1f67d104ee0>
```

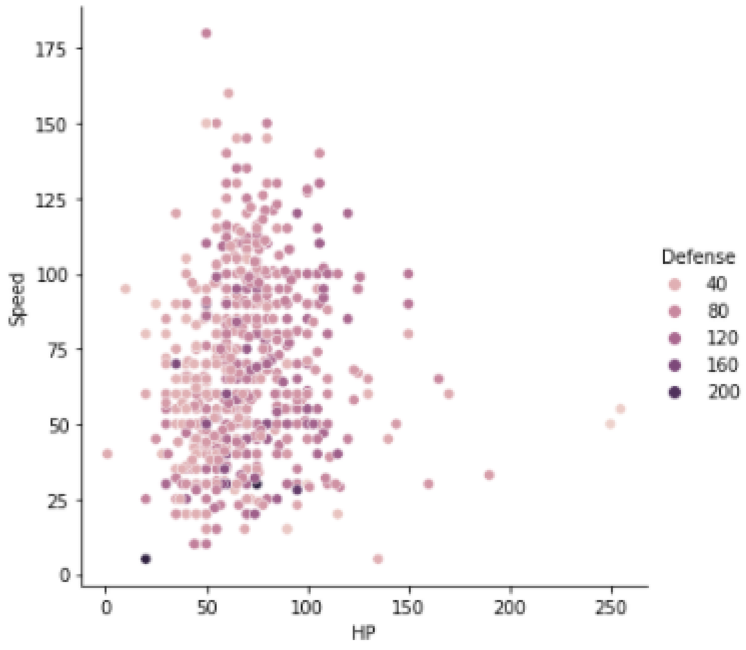


SNS.relplot using Seaborn

SNS.relplot is the relplot function from SNS class, which is a seaborn class that we imported above with other dependencies.
The parameters – x, y, and data – represent the variables on X-axis, Y-axis and the data we are using to plot respectively. Here, we've found a relationship between the views and upvotes.

Next, if we want to see the tag associated with the data, we can use the below code:

```
In [9]: sns.relplot(x="HP", y="Speed", hue = "Defense",data = pok_data)

Out[9]: <seaborn.axisgrid.FacetGrid at 0x1f672c36940>
```



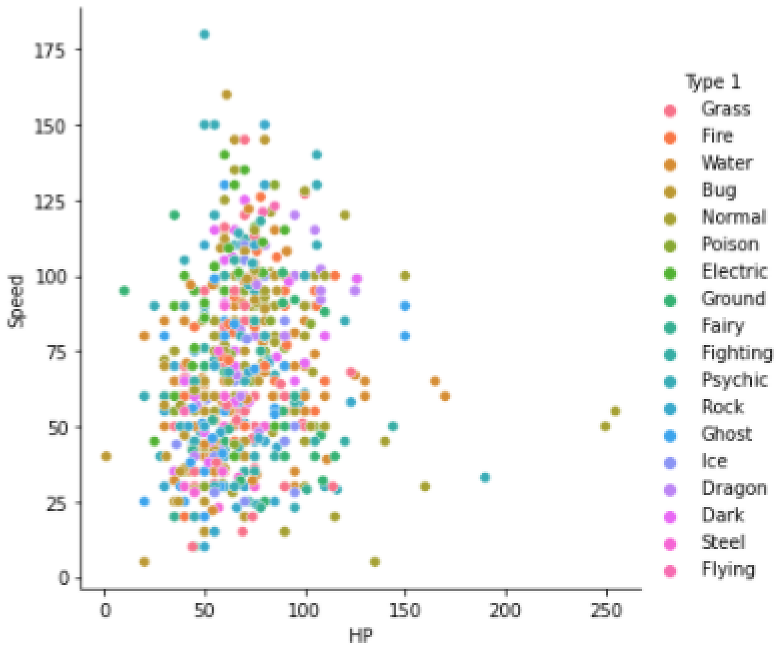
Hue Plot

We can add another dimension in our plot with the help of hue as it gives color to the points and each color has some meaning attached to it.

In the above plot, the hue semantic is categorical. That's why it has a different color palette. If the hue semantic is numeric, then the coloring becomes sequential.

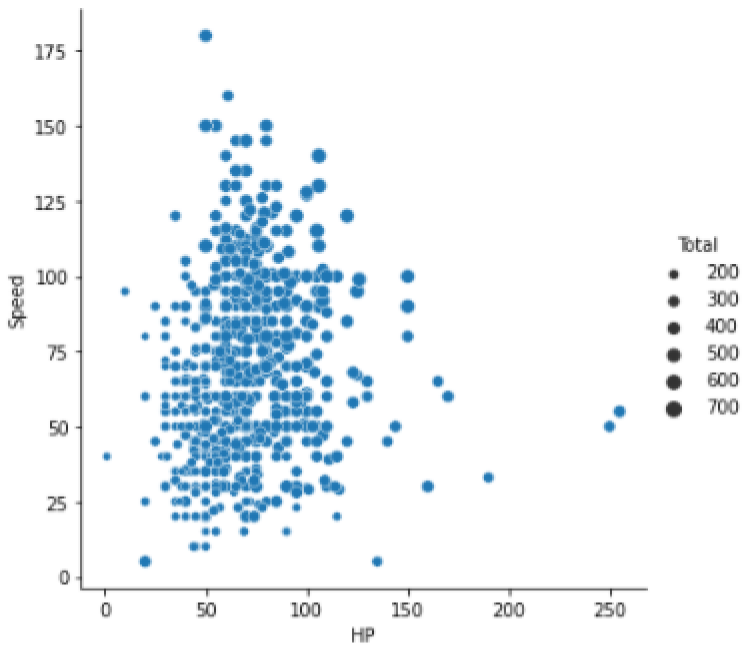
```
In [11]: sns.relplot(x="HP", y="Speed", hue = "Type 1",data = pok_data)
```

```
Out[11]: <seaborn.axisgrid.FacetGrid at 0x1f672c49670>
```



We can also change the size of each point:

```
In [15]: sns.relplot(x="HP", y="Speed", size = "Total", data = pok_data)

Out[15]: <seaborn.axisgrid.FacetGrid at 0x1f606279d60>
```



We can also change the size manually by using other parameter sizes as sizes = (15, 200).

Plotting Categorical Data

- Jitter
- Hue
- Boxplot
- Violin Plot
- Pointplot

In the above section, we saw how we can use different visual representations to show the relationship between multiple variables. We drew the plots between two numeric variables. In this section, we'll see the relationship between two variables of which one would be categorical (divided into different groups).
We'll be using the catplot() function of the seaborn library to draw the plots of categorical data.
Let's dive in.

Jitter Plot

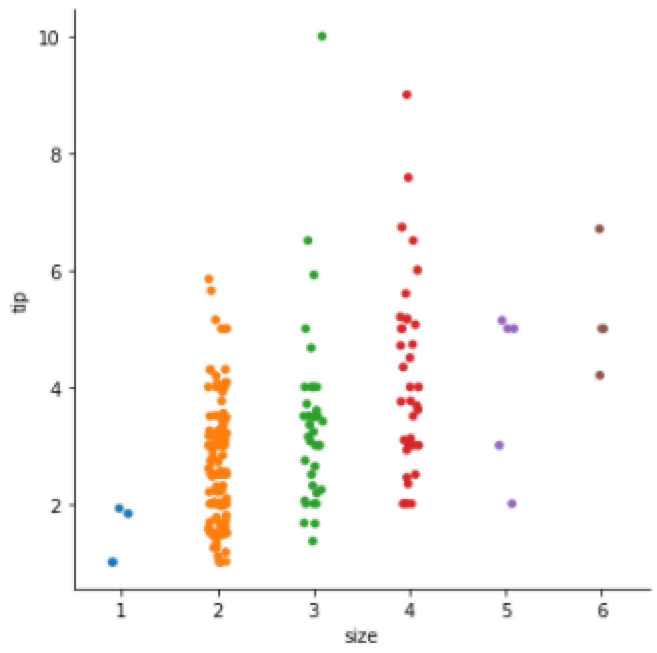
For the jitter plot we'll be using the dataset tips, let's import the dataset now.

```
In [16]: df2 = pd.read_csv(r"tips.csv")
         df2.head()
Out[16]:
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

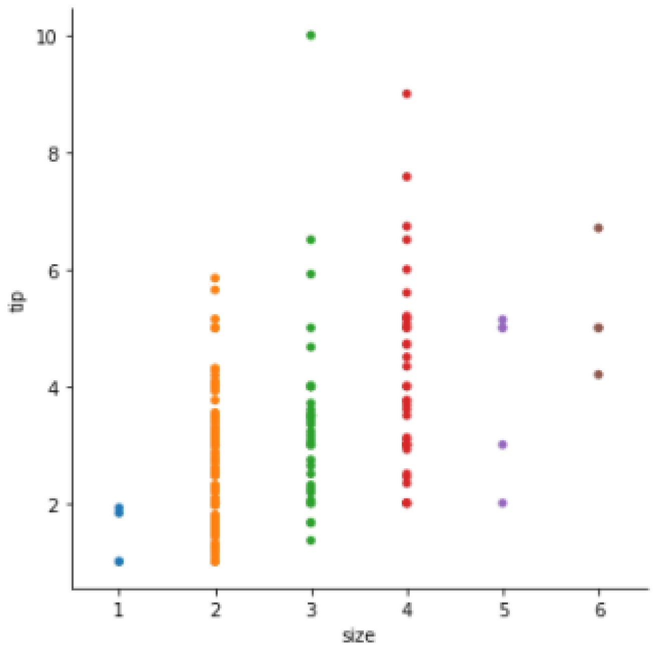
Now, we'll see the plot between the column **size** and **tip** by using the catplot() function.

```
In [18]: sns.catplot(x="size", y="tip", data=df2)
Out[18]: <seaborn.axisgrid.FacetGrid at 0x1f607773700>
```



Since we can see that the plot is scattered, so to handle that, we can set the jitter to false. Jitter is the deviation from the true value. So, we'll set the jitter to false by using another parameter.

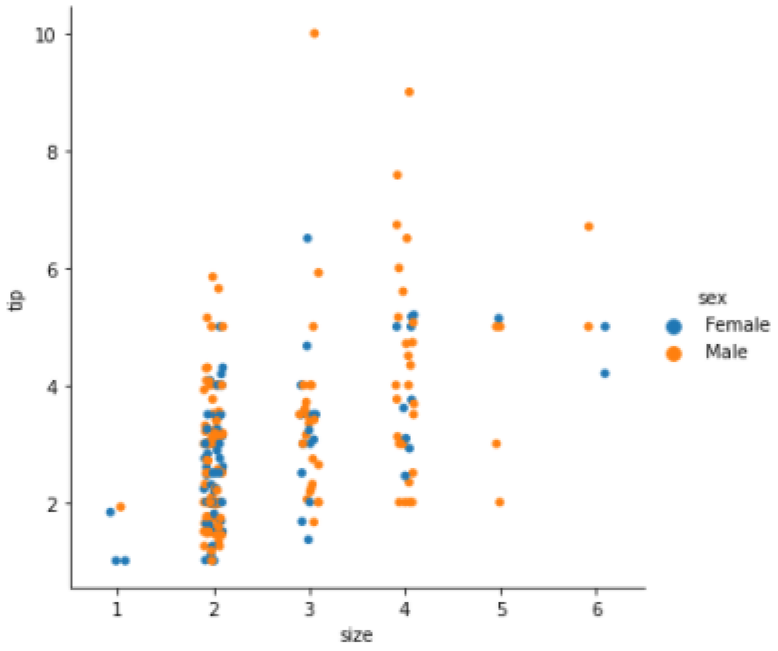
```
In [19]: sns.catplot(x="size", y="tip", jitter = False,  data=df2)
Out[19]: <seaborn.axisgrid.FacetGrid at 0x1f6077da9a0>
```



Hue Plot

Next, if we want to introduce another variable or another dimension in our plot, we can use the hue parameter just like we used in the above section. Let’s say we want to see the gender distribution in the plot of education and avg_training_score, to do that, we can use the following code

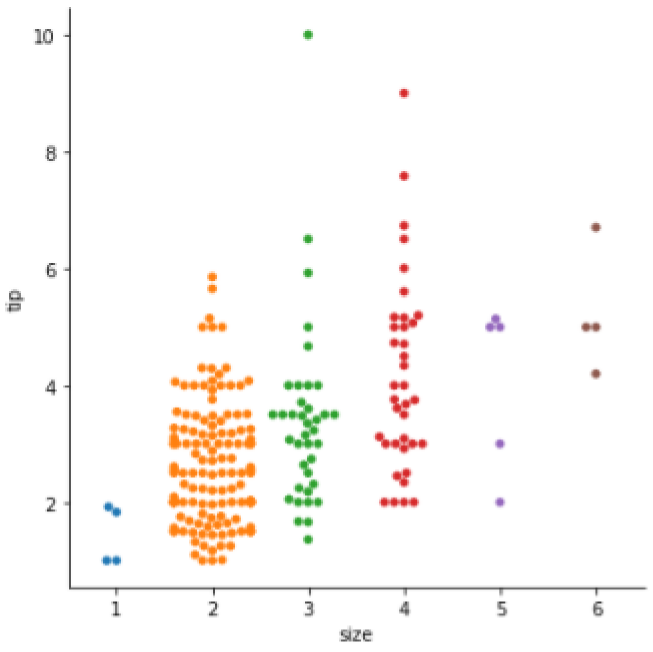
```
In [20]: sns.catplot(x="size", y="tip", hue = "sex", data=df2)
Out[20]: <seaborn.axisgrid.FacetGrid at 0x1f607838b20>
```



In the above plots, we can see that the points are overlapping each other, to eliminate this situation, we can set kind = “swarm”, swarm uses an algorithm that prevents the points from overlapping and adjusts the points along the categorical axis. Let’s see how it looks like-

```
In [21]: sns.catplot(x="size", y="tip", kind = "swarm", data=df2)
```

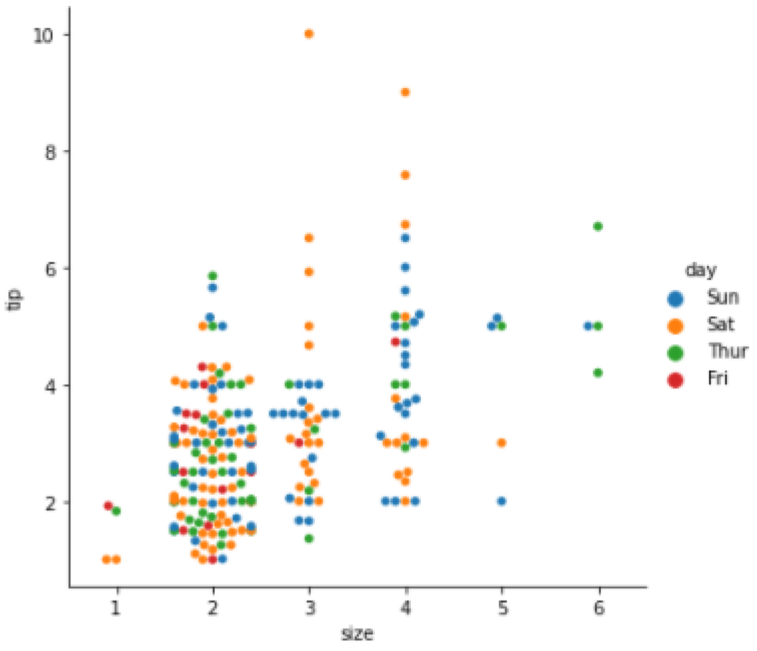
```
Out[21]: <seaborn.axisgrid.FacetGrid at 0x1f60778ad30>
```



Pretty amazing, right? What if we want to see the swarmed version of the plot as well as a third dimension? Let’s see how it goes if we introduce day as a new variable

```
In [22]: sns.catplot(x="size", y="tip", hue = "day", kind = "swarm", data=df2)
```

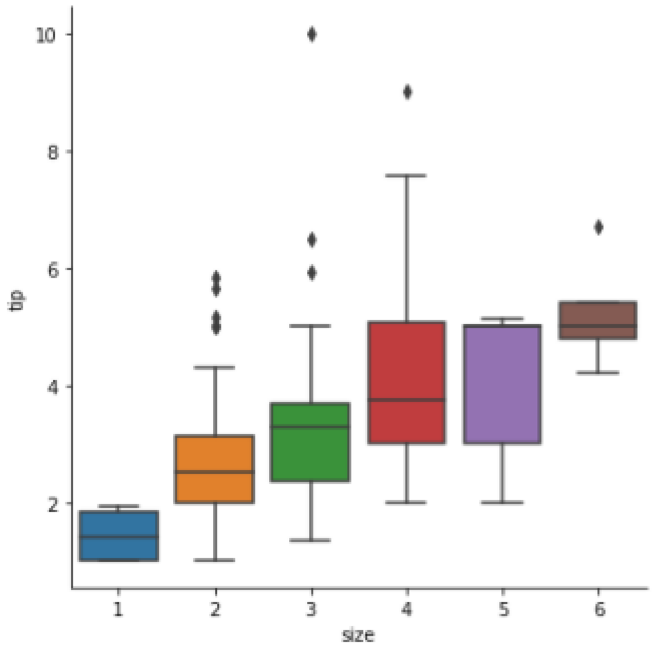
```
Out[22]: <seaborn.axisgrid.FacetGrid at 0x1f607816040>
```



Boxplot using seaborn

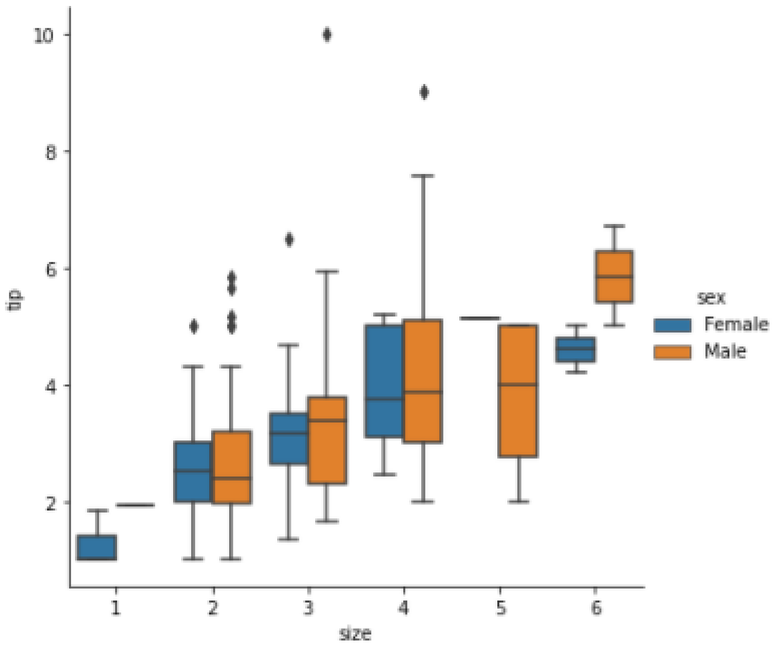
Another kind of plot we can draw is a **boxplot** which shows three quartile values of the distribution along with the end values. Each value in the boxplot corresponds to actual observation in the data. Let’s draw the boxplot now-

```
In [23]: sns.catplot(x="size", y="tip", kind = "box", data=df2)

Out[23]: <seaborn.axisgrid.FacetGrid at 0x1f607790280>
```



When we use hue semantic with boxplot, it is leveled along the categorical axis so they don’t overlap. The boxplot with hue would look like-

```
In [24]: sns.catplot(x="size", y="tip", hue = "sex", kind = "box", data=df2)

Out[24]: <seaborn.axisgrid.FacetGrid at 0x1f60794f4c0>
```

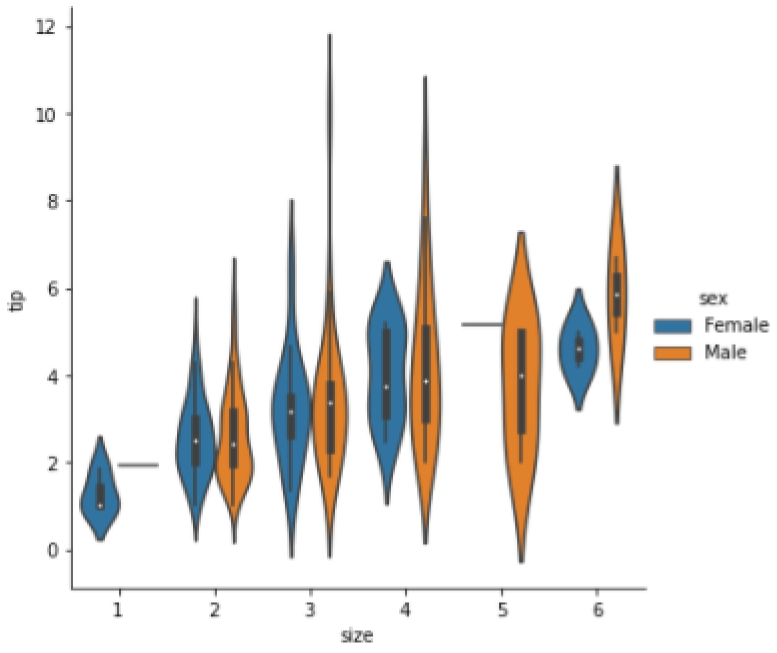


Violin Plot using seaborn

We can also represent the above variables differently by using violin plots. Let's try it out

```
In [25]: sns.catplot(x="size", y="tip", hue = "sex", kind = "violin", data=df2)
```

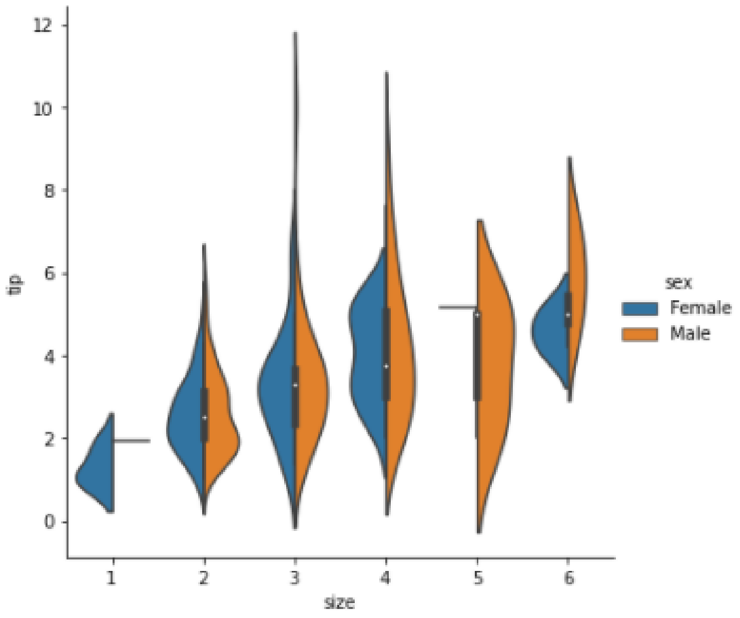
```
Out[25]: <seaborn.axisgrid.FacetGrid at 0x1f608a318e0>
```



The violin plots combine the boxplot and kernel density estimation procedure to provide richer description of the distribution of values. The quartile values are displayed inside the violin. We can also split the violin when the hue semantic parameter has only two levels, which could also be helpful in saving space on the plot. Let's look at the violin plot with a split of levels

.

```
In [26]: sns.catplot(x="size", y="tip", hue = "sex", kind = "violin", split = True, data=df2)

Out[26]: <seaborn.axisgrid.FacetGrid at 0x1f60897ae80>
```

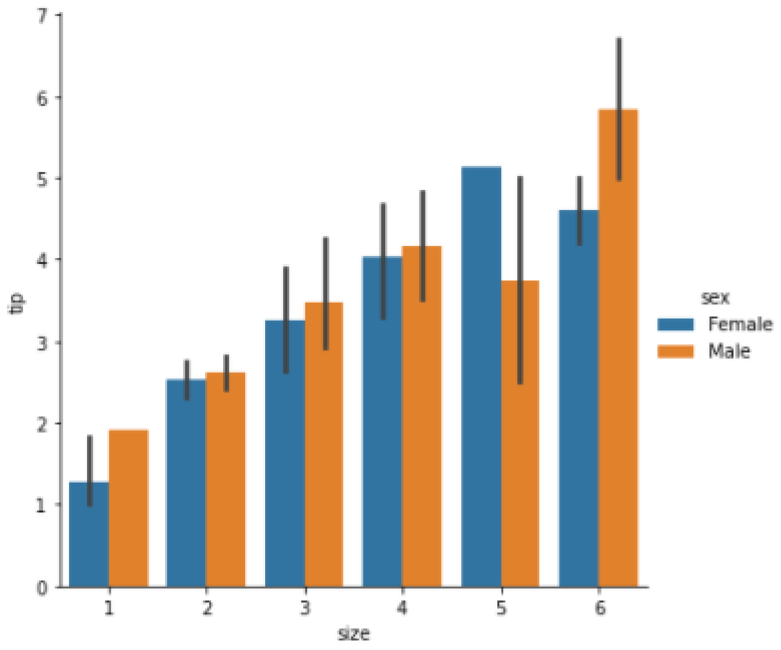


These amazing plots are the reason why I started using seaborn. It gives you a lot of options to display the data. Another coming in the line is boxplot.

Boxplot using seaborn

Boxplot operates on the full dataset and obtains the mean value by default. Let's face it now.

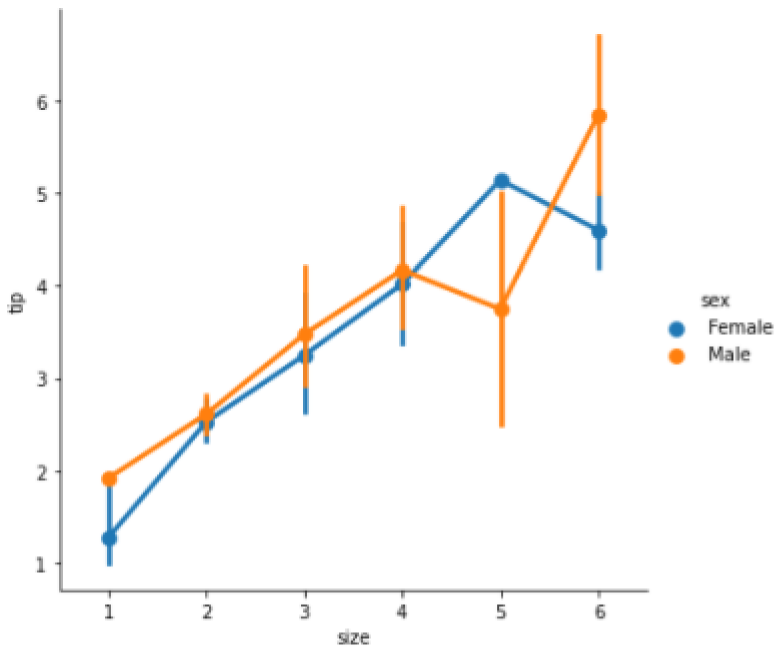
```
In [27]: sns.catplot(x="size", y="tip", hue = "sex", kind = "bar", data=df2)

Out[27]: <seaborn.axisgrid.FacetGrid at 0x1f607753280>
```



Pointplot using seaborn

Another type of plot coming in is pointplot, and this plot points out the estimated value and confidence interval. Pointplot connects data from the same hue category. This helps in identifying how the relationship is changing in a particular hue category. You can check out how a pointplot displays the information below.

```
In [28]: sns.catplot(x="size", y="tip", hue = "sex", kind = "point", data=df2)
Out[28]: <seaborn.axisgrid.FacetGrid at 0x1f608a0d490>
```



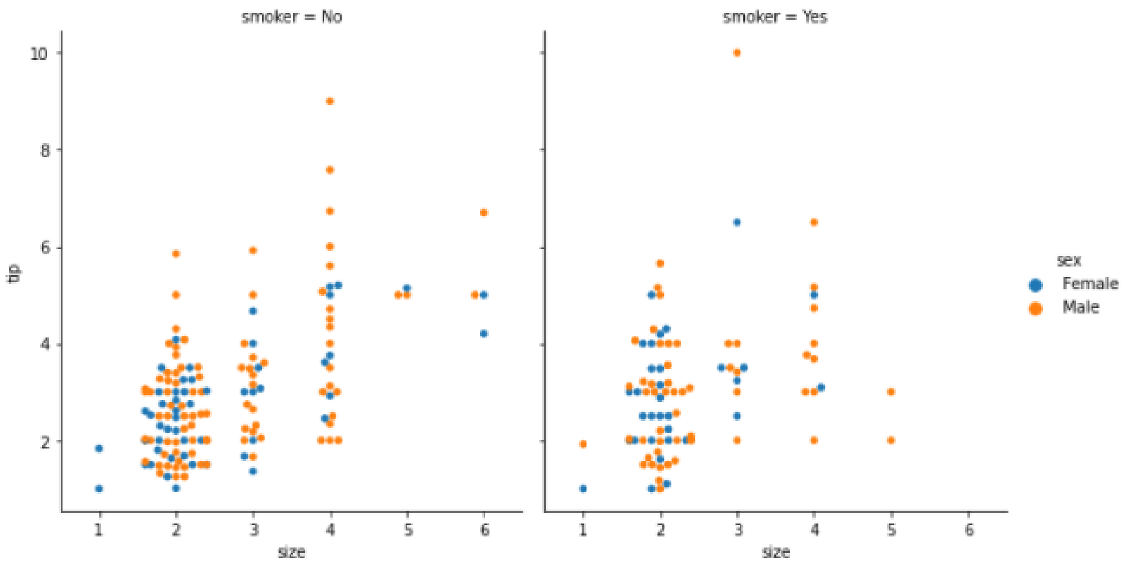
As it is clear from the above plot, the one whose score is high has more confidence in getting a promotion.

This is not the end, seaborn is a huge library with a lot of plotting functions for different purposes. One such purpose is to introduce multiple dimensions. We can visualize higher dimension relationships as well. Let’s check it out using a swarm plot.

Swarm plot using seaborn

```
In [30]: sns.catplot(x="size", y="tip", hue="sex",col="smoker", aspect=.9,kind="swarm", data=df2)
```

```
Out[30]: <seaborn.axisgrid.FacetGrid at 0x1f60ad15ee0>
```



It becomes so easy to visualize the insights when we combine multiple concepts into one. Here swarm plot is promoted attribute as hue semantic and gender attribute as a faceting variable.

Visualizing the Distribution of a Dataset

Whenever we are dealing with a dataset, we want to know how the data or the variables are being distributed. Distribution of data could tell us a lot about the nature of the data, so let's dive into it.

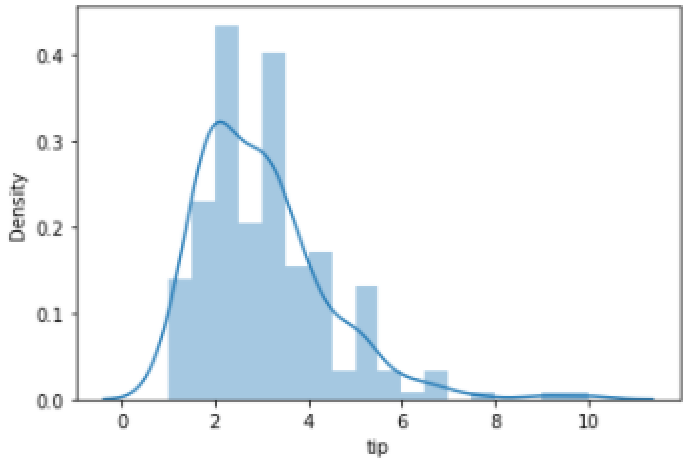
Plotting Univariate Distributions

- Histogram

One of the most common plots you'll come across while examining the distribution of a variable is distplot. By default, the distplot() function draws a histogram and fits a Kernel Density Estimate. Let's check out how age is distributed across the data.

```
In [31]: sns.distplot(df2.tip)
```

```
Out[31]: <AxesSubplot:xlabel='tip', ylabel='Density'>
```



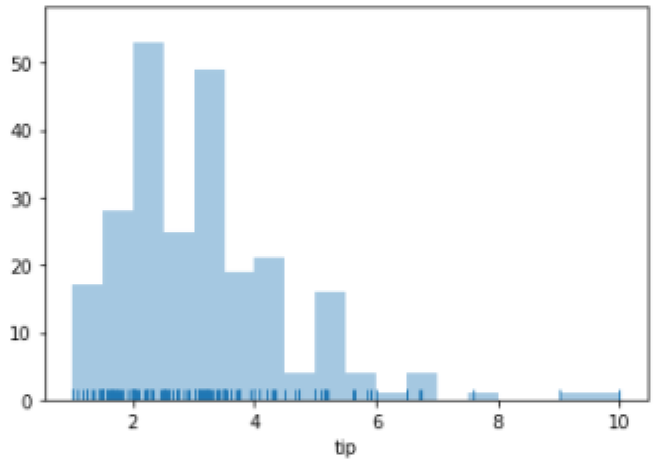
This clearly shows that the majority of people tip between 2 to 3 dollars.

Histogram using Seaborn

Another kind of plot that we use for univariate distribution is a histogram.
A histogram represents the distribution of data in the form of bins and uses bars to show the number of observations falling under each bin. We can also add a rugplot in it instead of using KDE (Kernel Density Estimate), which means at every observation, it will draw a small vertical stick.

```
In [32]: sns.distplot(df2.tip, kde=False, rug = True)
```

```
Out[32]: <AxesSubplot:xlabel='tip'>
```

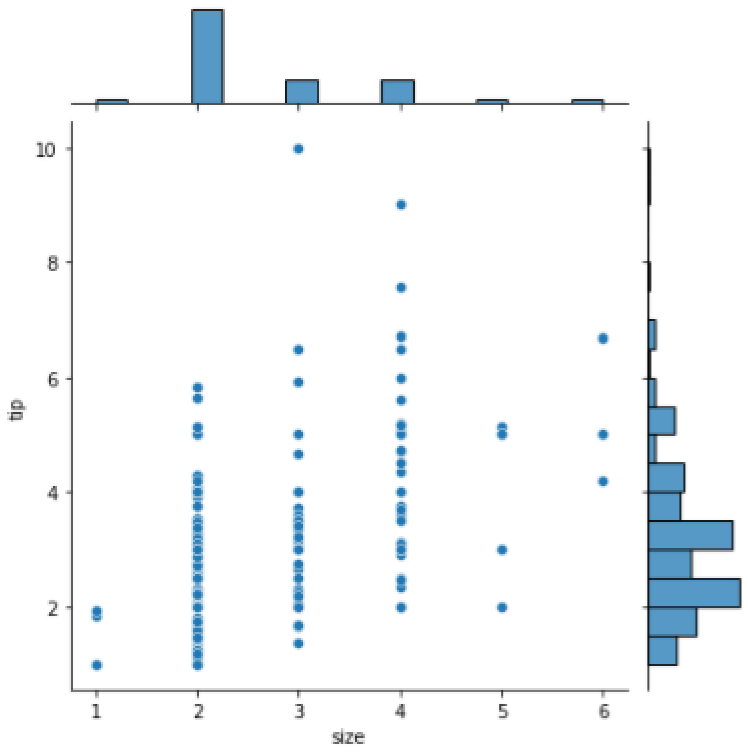


Plotting Bivariate Distributions

- Hexplot
- KDE plot
- Boxen plot
- Ridge plot (Joyplot)

Apart from visualizing the distribution of a single variable, we can see how two independent variables are distributed with respect to each other. Bivariate means joint, so to visualize it, we use the jointplot() function of seaborn library. By default, jointplot draws a scatter plot. Let's check out the bivariate distribution between tip and size.

```
In [33]: sns.jointplot(x="size", y="tip", data=df2);
```

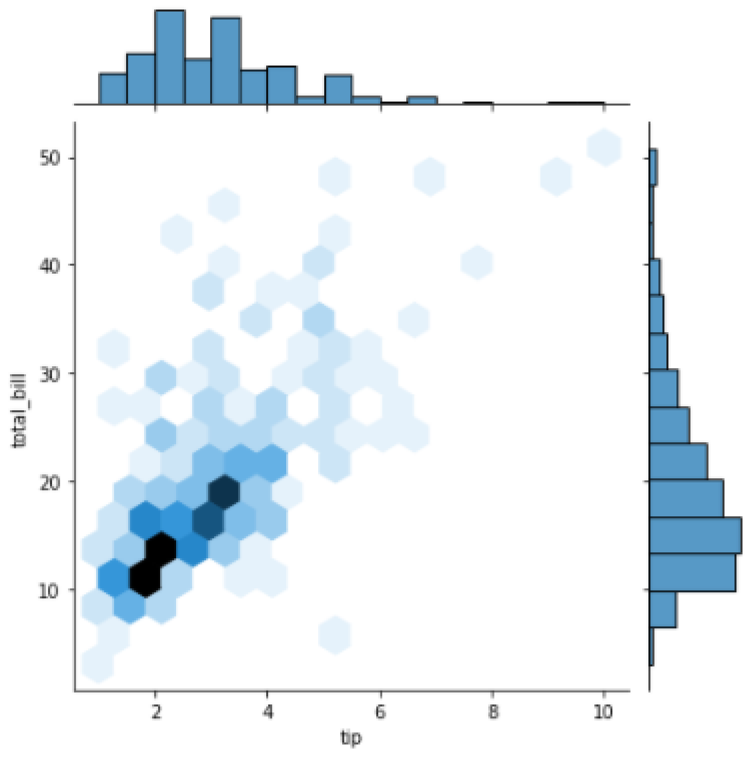


There are multiple ways to visualize bivariate distribution. Let's look at a couple of more.

Hexplot using Seaborn

Hexplot is a bivariate analog of histogram as it shows the number of observations that fall within hexagonal bins. This is a plot which works with a large dataset very easily. To draw a hexplot, we'll set a kind attribute to hex. Let's check it out now.

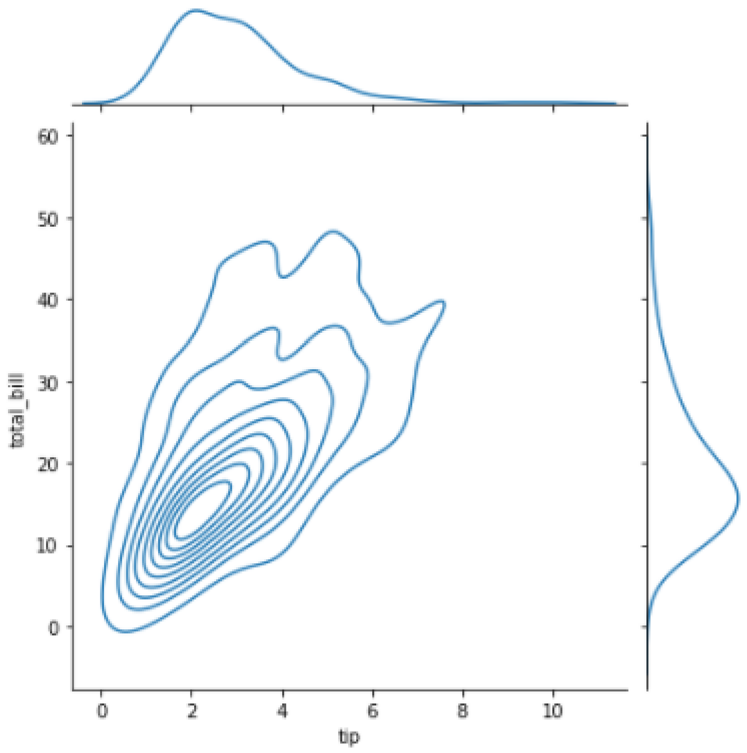
```
In [35]: sns.jointplot(x=df2.tip, y=df2.total_bill, kind="hex", data = df2)

Out[35]: <seaborn.axisgrid.JointGrid at 0x1f60c34b760>
```



KDE Plot using Seaborn

That’s not the end of this, next comes the KDE plot. It’s another very awesome method to visualize the bivariate distribution. Let’s see how the above observations could also be achieved by using jointplot() function and setting the attribute kind to KDE.

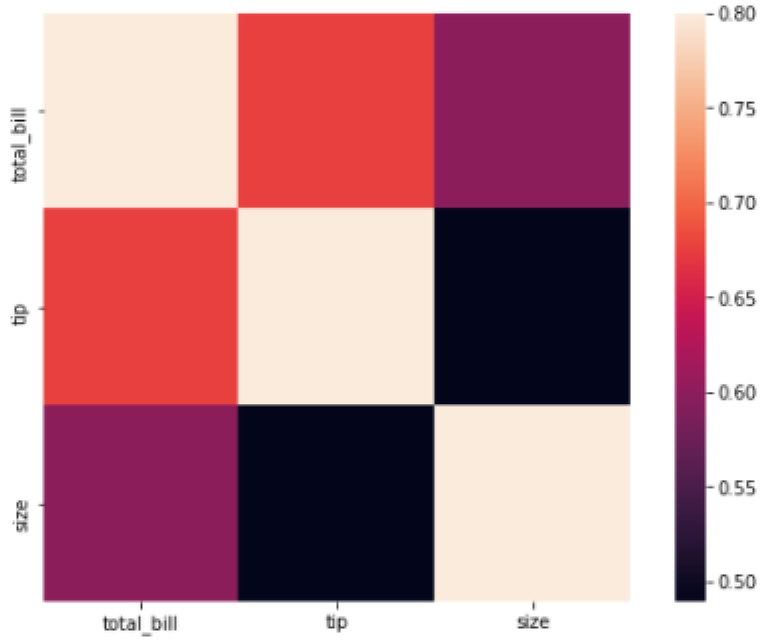
```
In [36]: sns.jointplot(x="tip", y="total_bill", data=df2, kind="kde");
```



Heatmaps using Seaborn

Now let’s talk about my absolute favorite plot, the heatmap. Heatmaps are graphical representations in which each variable is represented as a color.
Let’s go ahead and generate one:

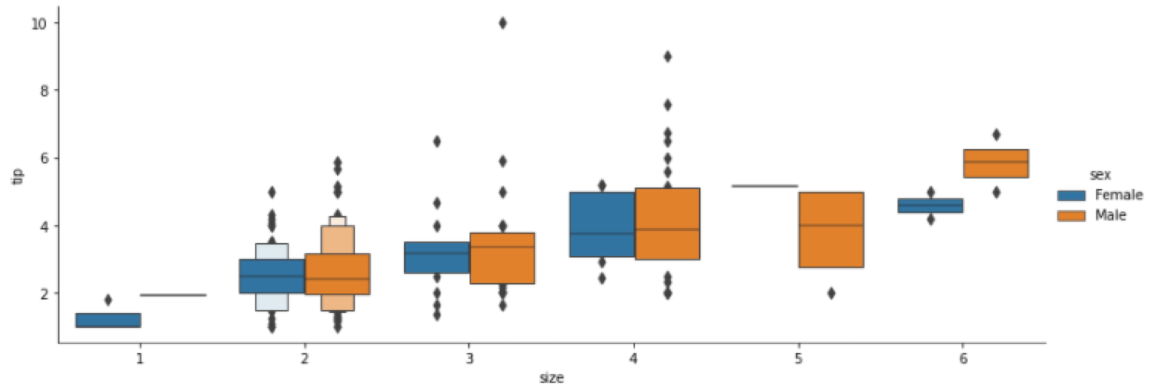
```
In [37]: corrmat = df2.corr()
         f, ax = plt.subplots(figsize=(9, 6))
         sns.heatmap(corrmat, vmax=.8, square=True)

Out[37]: <AxesSubplot:>
```



Boxen Plot using Seaborn

Another plot that we can use to show the bivariate distribution is the boxen plot. Boxen plots were originally named letter value plots as it shows a large number of values of a variable, also known as quantiles. These quantiles are also defined as letter values. By plotting a large number of quantiles, it provides more insights about the shape of the distribution. These are similar to box plots, let’s see how they could be used.

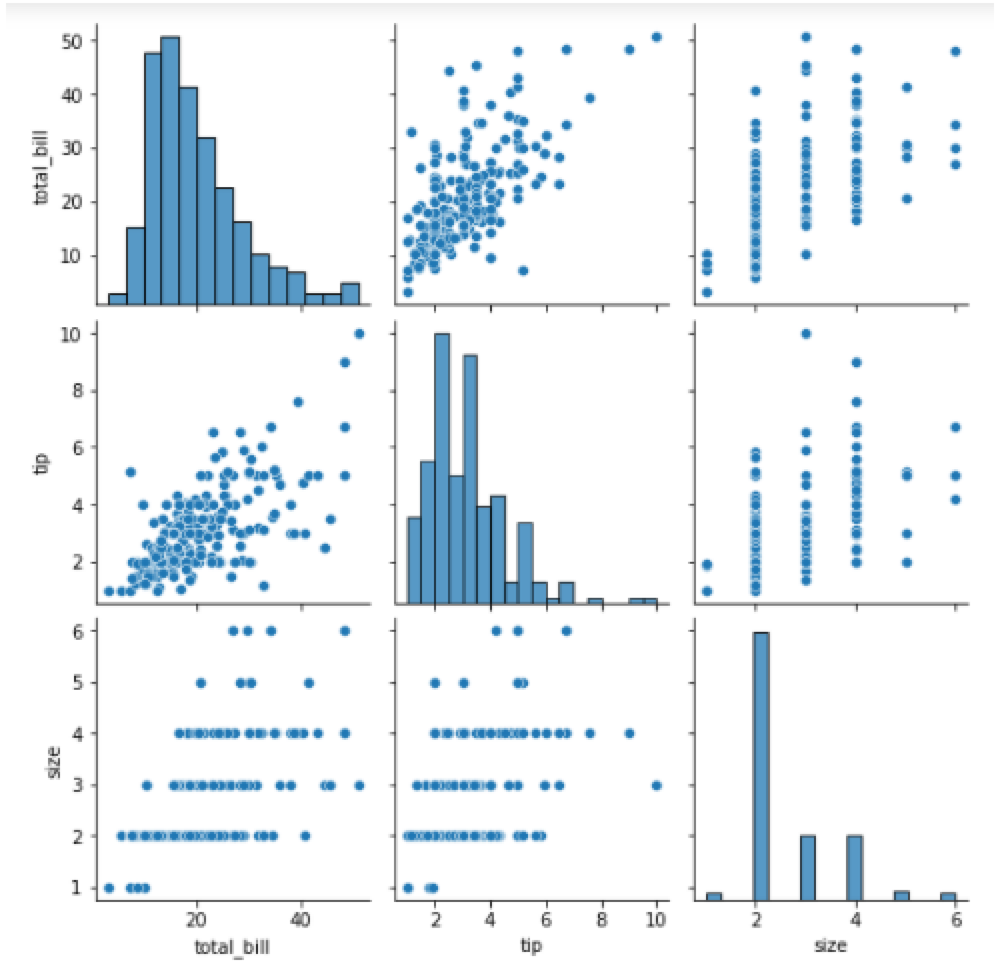
```
In [41]: sns.catplot(x="size", y="tip", data=df2, kind="boxen",height=4, aspect=2.7, hue = "sex")
Out[41]: <seaborn.axisgrid.FacetGrid at 0x1f60748ddf0>
```



Visualizing Pairwise Relationships in a Dataset

We can also plot multiple bivariate distributions in a dataset by using the pairplot() function of the seaborn library. This shows the relationship between each column of the database. It also draws the univariate distribution plot of each variable on the diagonal axis. Let’s see how it looks.

```
In [44]: sns.pairplot(df2)
Out[44]: <seaborn.axisgrid.PairGrid at 0x1f60e1d9e50>
```



We’ve covered a lot of plots here. We saw how the seaborn library can be so effective when it comes to visualizing and exploring data (especially large datasets). We also discussed how we can plot different functions of the seaborn library for different kinds of data.

Why only talk about knowledge when you can work it? Join our program to get trained and work on live industry projects and get certified by start-ups. Use your skills on projects and increase your practical knowledge.