A Comprehensive Guide to Data Visualization Using Seaborn for Complete Beginners

helloml.org/a-comprehensive-guide-to-data-visualization-using-seaborn-for-complete-beginners

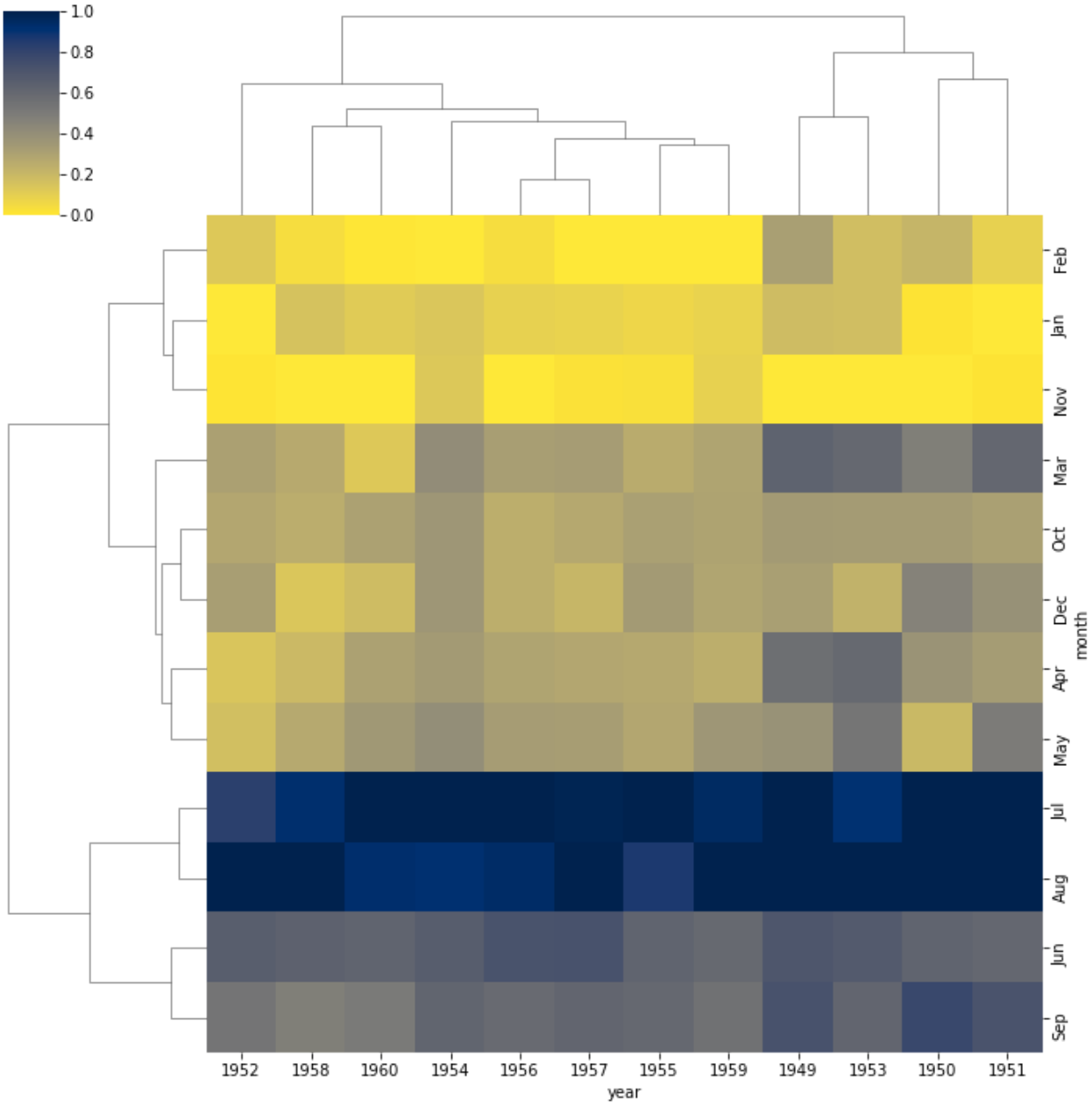
September 8, 2021

This article is a continuation of the 'Comprehensive guide to data visualization with Python for complete beginners' series, with the first article being a comprehensive introduction to data visualization using Matplotlib (link here).

An introduction to data visualization was already given in the previous article, so let us straight away dive into Seaborn!

Brief Introduction to Seaborn

Seaborn is a data visualization library in Python. It provides high-level commands for statistical plotting and implements Matplotlib under the hood.



An example of plotting advanced plots using Seaborn – A **matrix plot (clustermap)** to display the given data.

Besides Matplotlib, Seaborn is one of the most powerful visualization tools in Python. The reason why it is so popular (and recommended for beginners as well as for people who want quick data visualization) is that it has beautiful default themes and has a simpler syntax than matplotlib. One can create complex plots such as heatmaps with just two lines of code.

Seaborn is more powerfully equipped to handle complicated charts (like violin plots, heatmaps, regression plots, etc.) at times than Matplotlib. However, since Seaborn is simply a high-level interface over Matplotlib, it provides lesser customization options.

Next, we will be looking at creating **4** types of plots using Seaborn (**Categorical Plots**, **Distribution Plots**, **Linear model/Regression Plots**, and **Matrix Plots**).

Installing Seaborn and Getting Started with Distribution Plots

Assuming you're working on a Jupyter notebook (Anaconda installation of Python), installing Seaborn on your PC is very simple. All you need to do is type in the following command in the Anaconda command prompt:

conda install seaborn
Or if you are using Python without anaconda, you can also install it using pip as shown below.

pip install seaborn
The next step is to import the required libraries.

```
import numpy as np
import pandas as pd
import seaborn as sns
%matplotlib inline
```

For this article, we will use a dataset that comes in-built with Seaborn, the '**tips**' dataset for our visualization purposes. The dataset has **7** columns and **244** non-null entries consistent for all the columns. We will load the dataset and check the head of the data frame:

```
tips = sns.load_dataset('tips')
tips.head()
```

	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

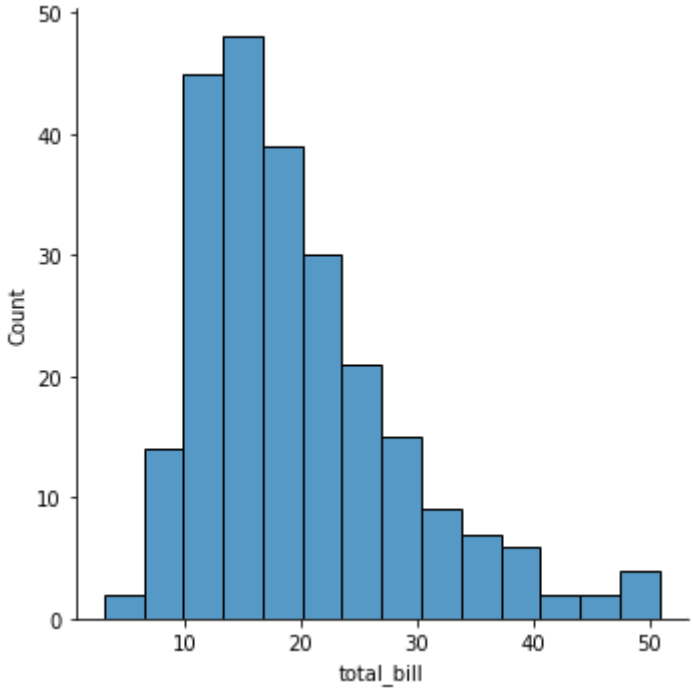
Next, we will plot basic distribution plots using Seaborn, i.e. **Displot**, **Jointplot**, and **Pairplot**.

Displot (Histogram and KDE plots)

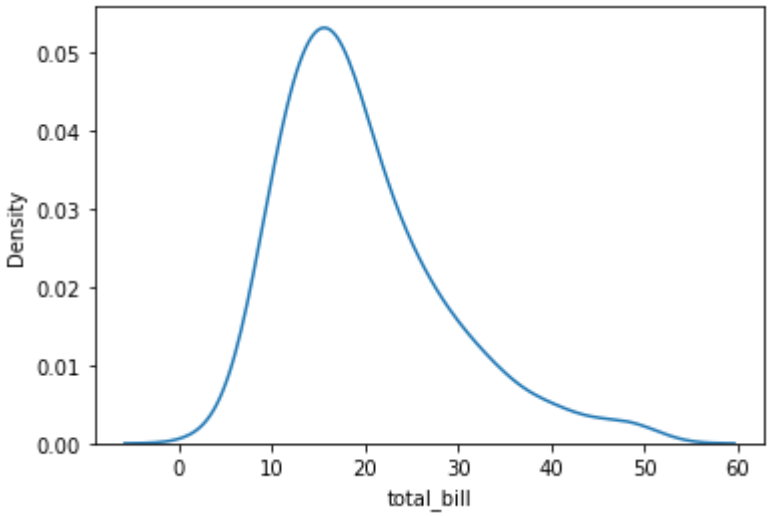
```
sns.displot(tips['total_bill'])
```

Displot allows us to plot the distribution of the column (univariate data). It also allows us to plot bivariate data. We are simply passing in the '**total_bill**' column of the tips dataset and it plots the distribution plot (histogram) for that particular column. By default, displot plots a histogram. However, we can also plot the kernel density estimate (kde) or the empirical cumulative distribution function (ecdf) plots by providing the arguments to the function as '**kind = kde**' and '**kind = ecdf**' respectively for kdeplots and ecdfplots as shown below.

```
sns.displot(tips['total_bill'],kind='kde')
```



Output

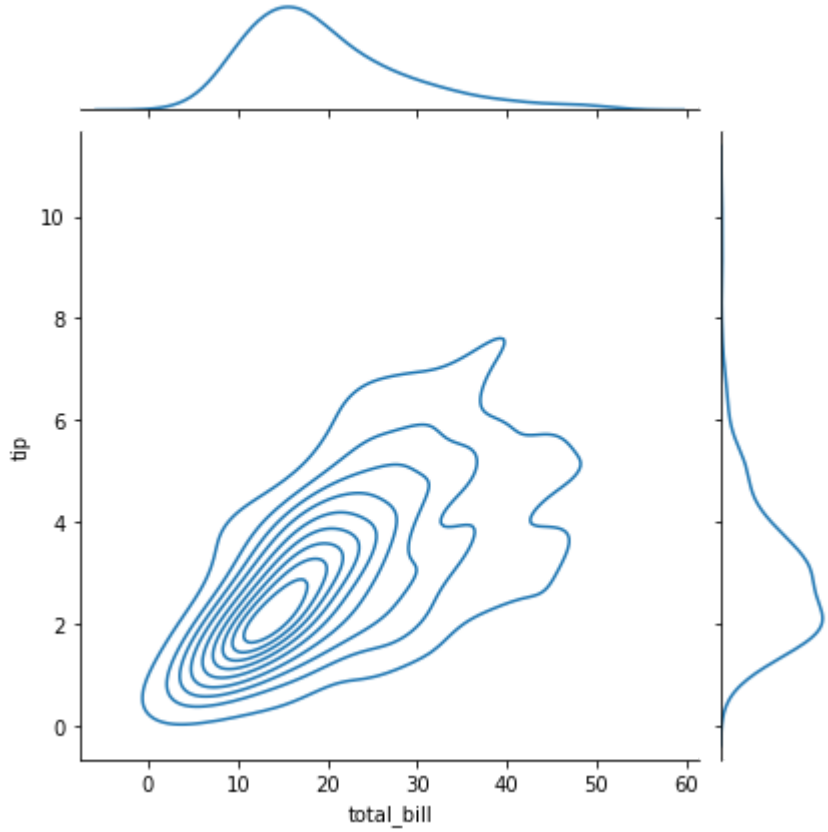


KDE plot

We can also directly plot histograms and kde plots by using the functions **sns.histplot()** and **sns.kdeplot()** respectively. All we need to do is provide the variables which we want to plot.

Jointplot

sns.jointplot(x='total_bill',y='tip',data=tips,kind='kde')



Output

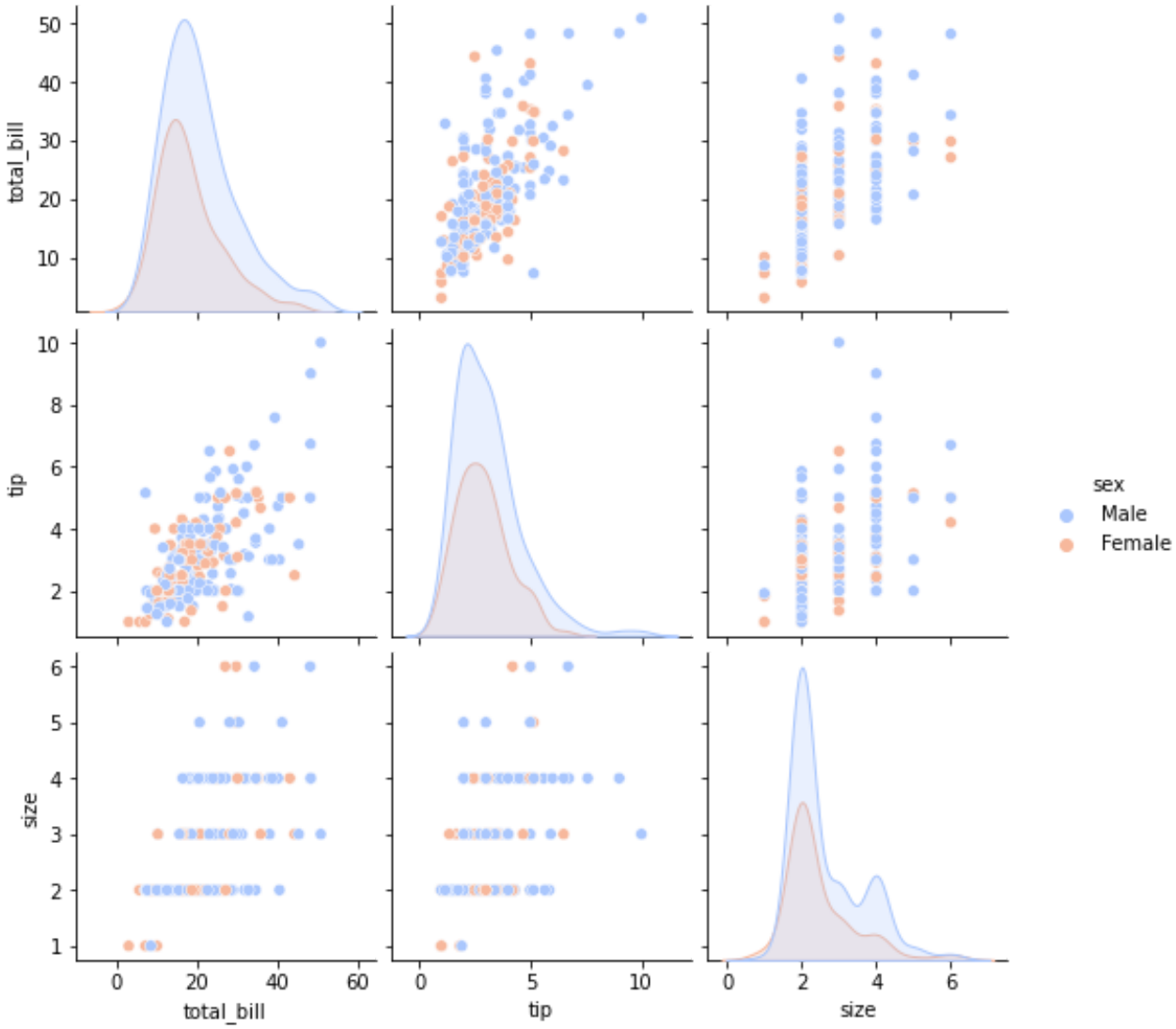
Here, instead of providing arguments like **tips['total_bill']**, we are specifically mentioning which variable we want on the x column, y column and we are only mentioning the name of the column. However, for the function to understand which data frame to look into to find our columns, we are also providing the additional argument of '**data = tips**'.

A joint plot allows us to draw multiple bivariate plots with univariate marginal distributions. The '**kind**' attribute can have one of the several options: "**scatter**"/"**kde**"/"**hist**"/"**hex**"/"**reg**"/"**resid**". For our example, we are using the **kde** plot.

Pairplot

Pairplot allows us to plot **pairwise relationships** in a dataset, i.e. we plot pairwise relationships (jointplot) between any 2 columns, for all the columns in the dataset. Further, the '**hue**' argument allows us to map plot aspects to different colors (variables). So here, we can differentiate data based on **male** and **female** genders. The '**palette**' argument is one of the plot customization options provided by Seaborn. For our plot, we are using the inbuilt '**coolwarm**' color scheme.

```
sns.pairplot(tips,hue='sex',palette='coolwarm')
```



Output

Categorical Plots using Seaborn

Categorical plots are the ones in which we plot a categorical column vs a numerical column or another categorical column.

If we look at the tips dataset:

tips.head()

	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

‘**total_bill**‘, ‘**tip**‘, ‘**size**‘ are numerical columns because the data stored in these columns are numbers that represent either a measurement or have numerical meaning.

‘**sex**‘, ‘**smoker**‘, ‘**day**‘, ‘**time**‘ are categorical columns because they represent types of data that may be divided into groups. For eg: the ‘sex’ variable differentiates our data point into one of the 2 fixed categories which this variable can take: ‘Male’ or ‘Female’. Similarly, ‘smoker’ categorizes a data point into being either a real-life smoker (Represented by ‘Yes’) or not (Represented by ‘No’). Usually, categorical variables take a value out of a fixed set of numerical values/character strings.

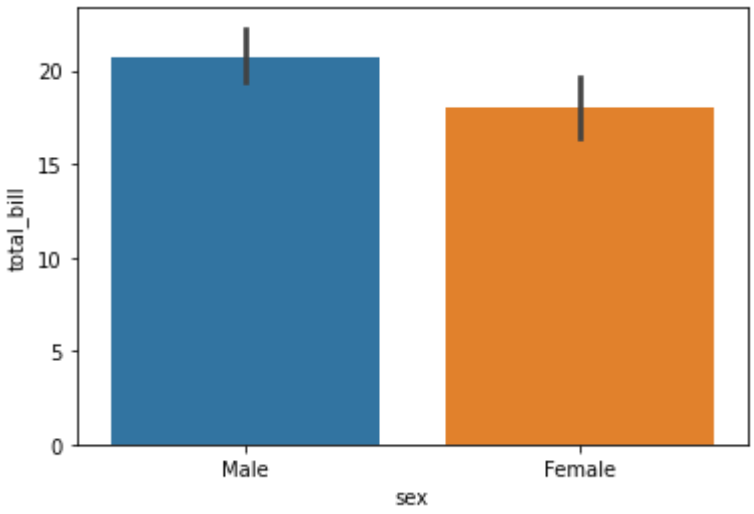
Next, we will look at plotting out some standard categorical plots using Seaborn: **Bar plot**, **Box plot**, **Violin plot**, **Strip plot**, and **Category plot** – We will continue using the in-built tips dataset. Also, we had imported NumPy earlier – It will come in handy here.

Barplot

A barplot, in Seaborn, displays/plots out the ‘**estimator**‘ for categorical columns. The ‘estimator’ is provided as an argument for the barplot function. Now, what is an estimator?

An ‘estimator’ is an estimate of central tendency. By default, the estimator value to barplot is numpy.mean, i.e. Mean. We can also input other measures of central tendencies such as the standard deviation (numpy.std).

```
sns.barplot(x='sex',y='total_bill',data=tips,estimator=np.mean)
```



Output

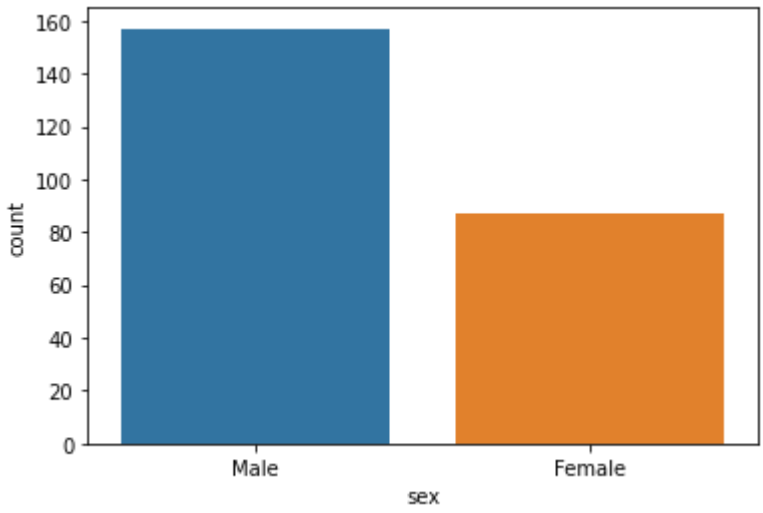
Barplot represents an approximation of central tendency for a numeric variable with the height of each rectangle (Bars) and provides some indication of the uncertainty around that estimate (Using the black marks on top of the bars). Thus, in short, a bar plot shows only the estimator values pertaining to a numerical column differentiated by a categorical variable.

For example, in our plot, we are providing the arguments x=’sex’, y=’total_bill’, ‘data=tips’, ‘estimator=np.mean’ (Which is the default estimator anyways so this argument was unnecessary here). Thus, we are showing the mean of the ‘total_bill’

column for ‘Male’ and ‘Female’ categories (In our case, gender). The mean of the total bill for males is approximately **20.74** while the mean for the females is approx. **18.05** which is shown on the plot with the degree of uncertainty (The error bars).

Countplot

sns.countplot(x='sex', data=tips)



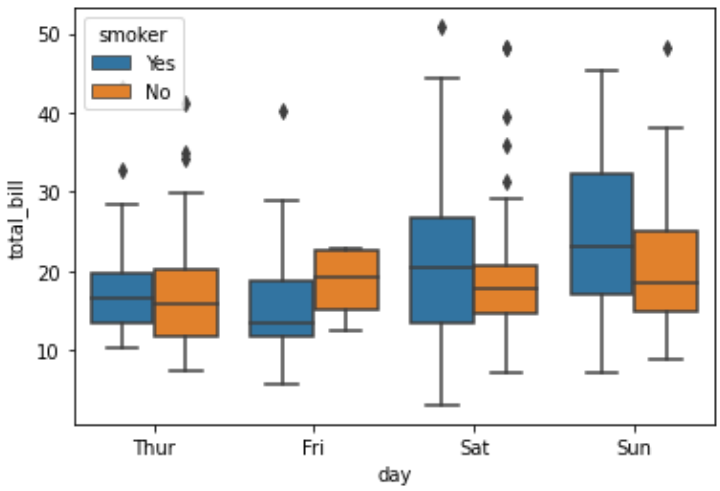
Output

A countplot is essentially the same as the previously discussed barplot, except that the estimator here is fixed and represents the count values for the column. Thus, in our example, we are plotting the number of data points which are male and females in our dataset. As represented by the plot, it comes out to be **157** for males and **87** for females.

Boxplot

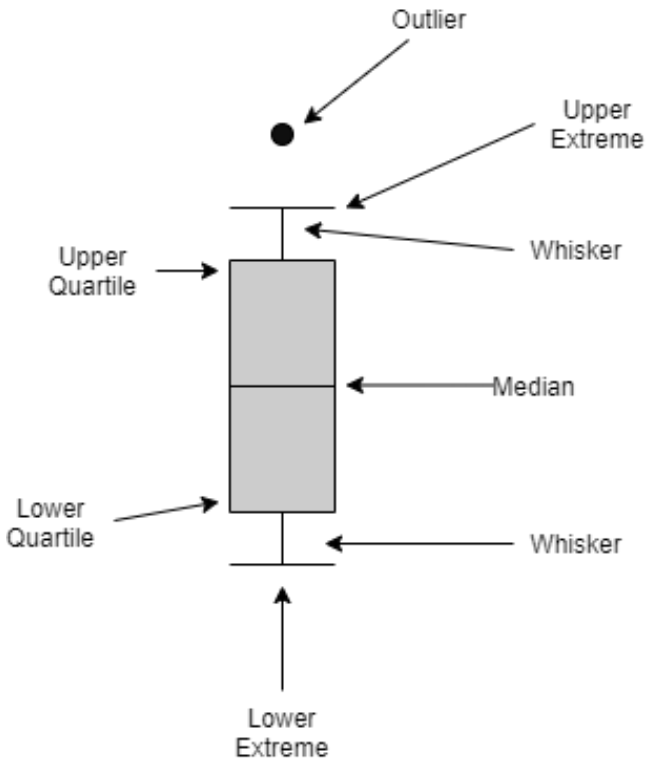
sns.boxplot(x='day',y='total_bill',data=tips,hue='smoker')

A boxplot represents a Box plot or a Box & Whisker plot. In our example, we are plotting the box plot for the total bill for all the days in our dataset. Note that data differentiation is done by the ‘hue’ argument, which in this case is ‘Smoker’ so two plots are shown for each day, with one for smokers and another for non-smokers. Since this tutorial is more about plotting the various types of statistical plots using Seaborn rather than explaining types of plots, we will cover Box plot in brief. This will also follow for the advanced plots that follow after this.



Output

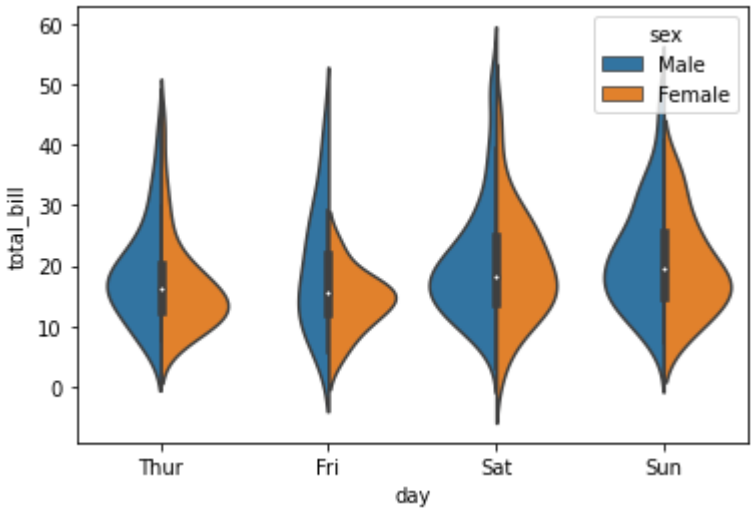
A boxplot displays the five-number summary of a set of data. The five-number summary is the minimum, first quartile, median, third quartile, and maximum. The minimum and maximum points are represented by the dash in the horizontal axis at the edge of each plot. The lines extending parallel (inline) from the boxes which reach up to the extremes (upper and lower) are known as the “whiskers”, which indicate variability outside the upper and lower quartiles. For some plots, dots follow the maximum dash. Those points are the outliers. The outliers are in the same line as the whiskers. For more clarity, refer to the below figure:



Explanation of Box-Plot

Violinplot

sns.violinplot(x='day',y='total_bill',data=tips,hue='sex',split=True)



Output

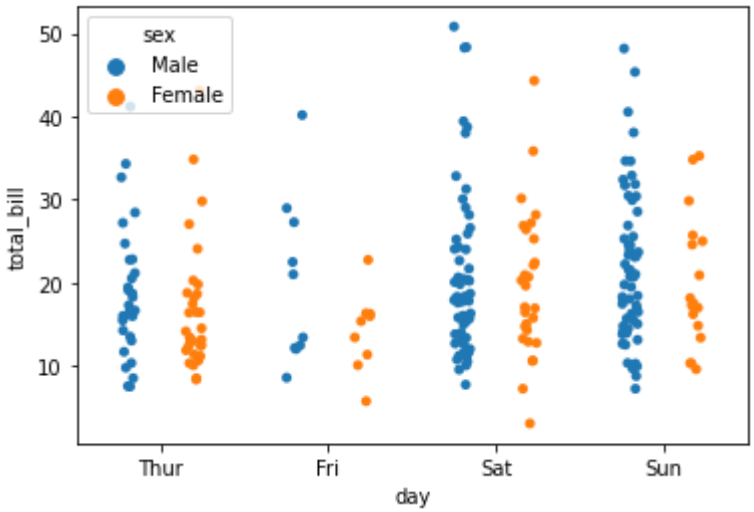
Violin plots are similar to box plots. However, they also show the probability density of the data at different values (The rotated kernel density plot on each side which results in the violin shape of the edges). A violin plot includes all the data that is in a box plot and also more information; While a box plot only shows the five-number summary of a set of data; The violin plot shows the full distribution of the data. Hence a violin plot is more informative than a box & whisker plot.

Here each plot (Violin) is further separated based on hue, i.e. sex which is either 'Male' or 'Female'. The '**split**' argument of the violinplot is a boolean attribute. When using hue nesting with a variable that takes two levels, setting split to True will draw half of a violin for each level. This can make it easier to directly compare the distributions.

We highly recommend that you play around with these functions to understand it completely.

Stripplot

sns.stripplot(x='day',y='total_bill',data=tips,jitter=True,hue='sex',dodge=True)



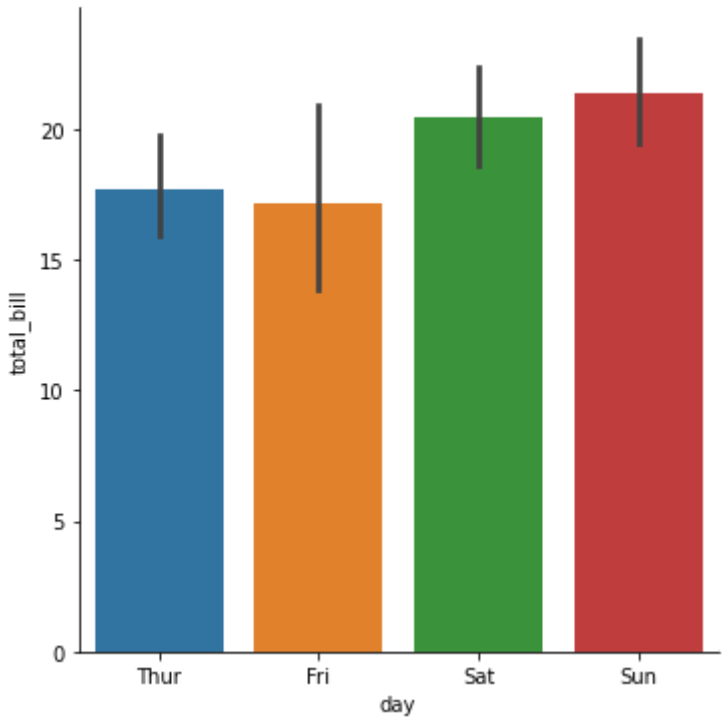
Output

The stripplot draws a scatterplot, where one variable is categorical in nature. Each point on the stripplot represents a data point, as with scatterplots. In our case, we are further separating the total bill for all days by the '**sex**' categorical variable; Which leads to the separation of **males**' and **females**' total bills on different days in our dataset. A strip plot is a good complement to a box or violin plot in cases where we want to show all the data points (observations) along with some representation of the underlying distribution.

Catplot

sns.catplot(x='day',y='total_bill',data=tips,kind='bar')
The category plot (Also known as **factor plot** plotted using the function **seaborn.factorplot()**)is the most general form of a categorical plot. As shown in our example, it takes a '**kind**' parameter to adjust the plot type. In our case, we are plotting the barplot with the default estimator (mean) for all days in our dataset. Thus, it will show

the median total bill for all days. As is predictable, the total bill on weekends is higher than the two weekdays.



Output

Matrix Plots using Seaborn

Matrix plots plot data as color-encoded matrices and can also be used to indicate clusters within the data. We will explore Seaborn's heatmap and clustermap functionalities.

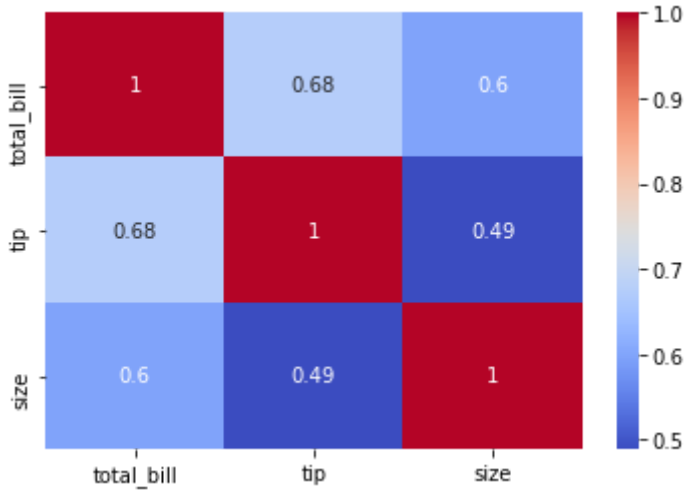
Heatmap

Our data should already be in a matrix form if we want to plot a proper heat map. The heatmap function basically just colors in the values for us. For example, continuing with our tips dataset, first, we will create a new variable to store correlation data (which is always in the form of a matrix):

tips.corr()

	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

sns.heatmap(tips.corr(),annot=True,cmap='coolwarm')
First, we provide the data (**tips.corr()**) as the argument. '**annot=True**' leads to the display of the actual values (Annotation) on top of each square. '**cmap=coolwarm**' sets the colormap for the heat map.



Output

The Flights Dataset and Further Exploration of Heatmap, Clustermap

Next, we will plot another heatmap with a new example (Using another built-in dataset provided by Seaborn).

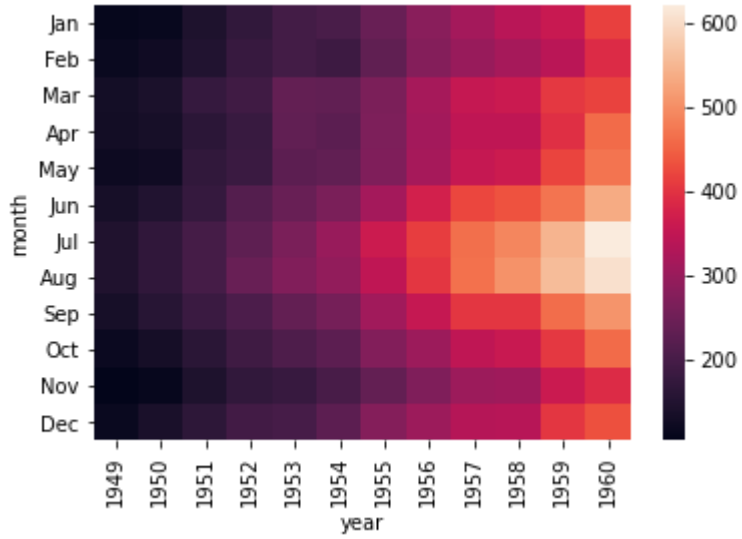
Load the **flights'** dataset and view the head for a better understanding of the data which we will plot.

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

Flights dataset contains data of the number of passengers for every month for **12** years time period, from **1949** to **1960**. Thus, it contains **12×12** matrix data if we transform the dataframe into a pivot table; And then we can plot our heat map:

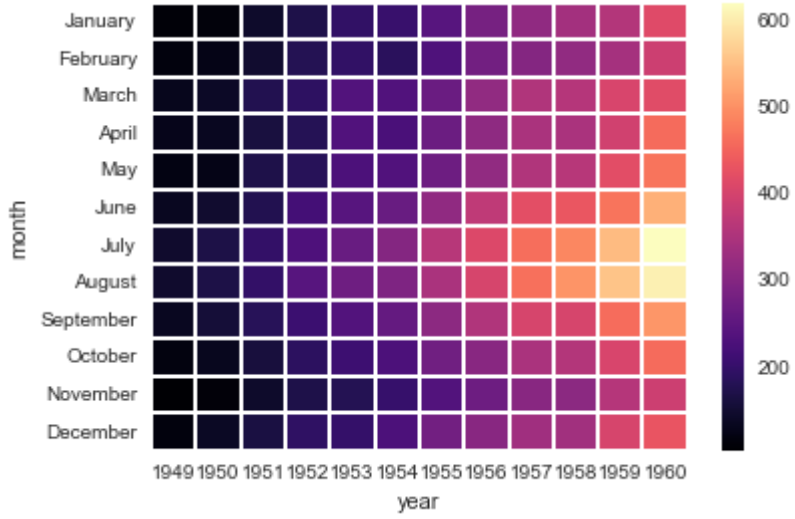
```
fp = flights.pivot_table(index='month',columns='year',values='passengers')
sns.heatmap(fp)
```



Output

To make the heatmap more visually appealing and clear, we can provide some additional arguments:

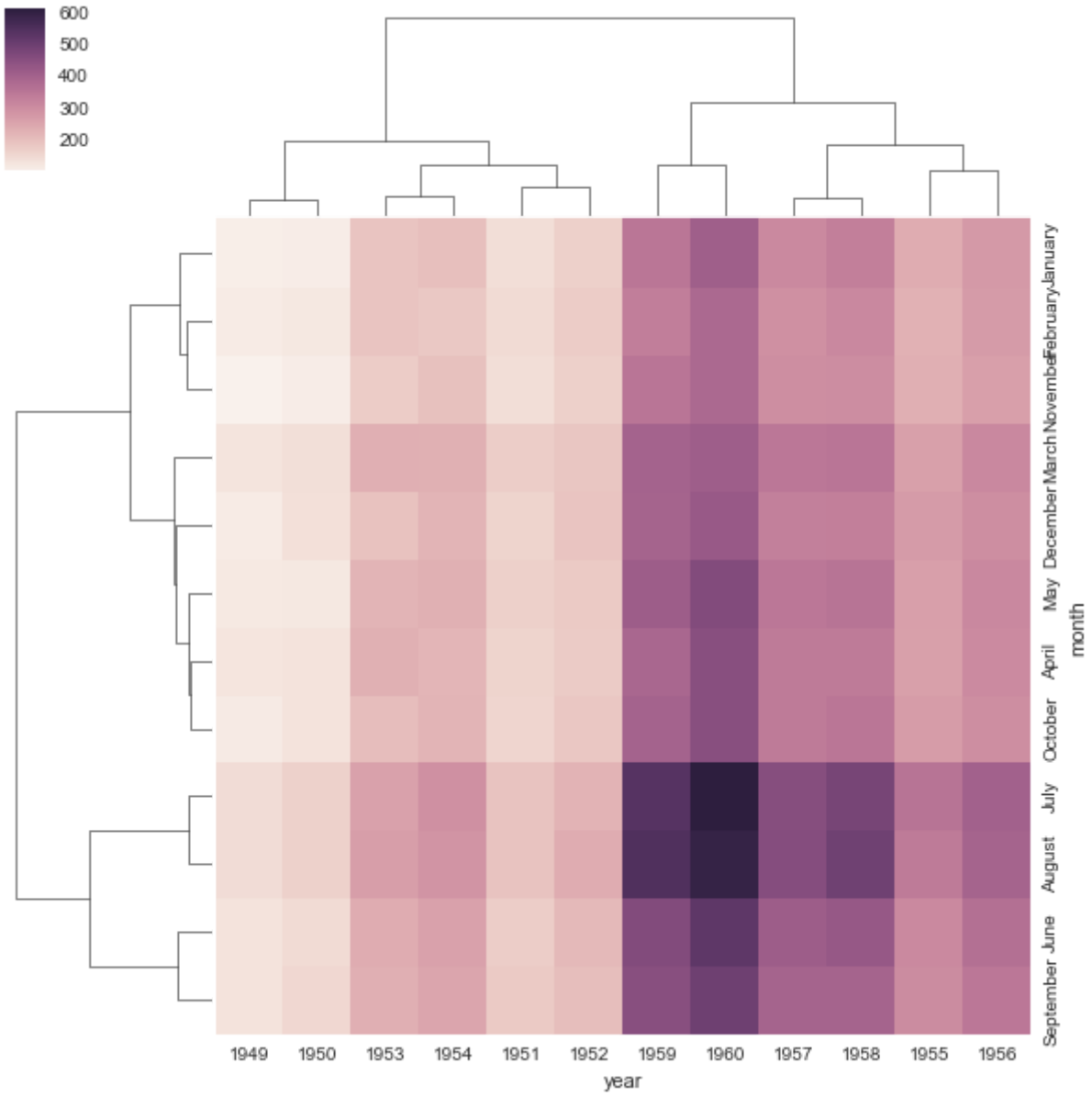
sns.heatmap(fp,cmap='magma',linecolor='white',linewidths=1)



Output

The clustermap uses hierarchal clustering to produce a clustered version of the heatmap. For example:

sns.clustermap(fp)



Output

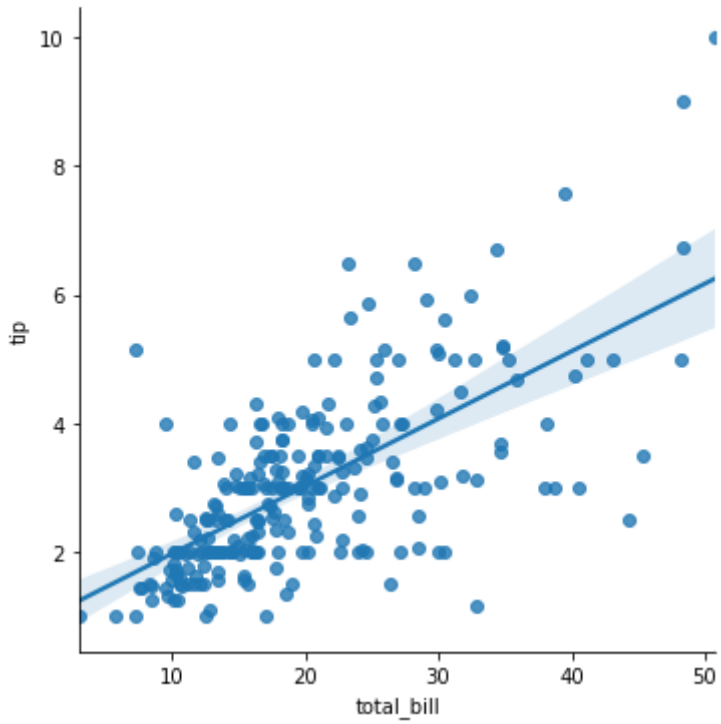
Now, the years and months are no longer in order, instead, they are grouped by similarity in value (passenger count). That means we can begin to make inferences from our clustermap, such as August and July being similar (makes sense, since they are both summer months in which we experience more travel hence a higher number of passengers) Also we can notice a general trend that as the years pass by, the number of passengers for any given month increases as compared to the previous years.

Regression Plots using Seaborn

Seaborn has many built-in capabilities for regression plots (relational plots), out of which we will cover the **Implot()** function. lmplot allows us to display linear models (regression line), but it also conveniently allows us to split up those plots based on features, as well as coloring the hue based on features.

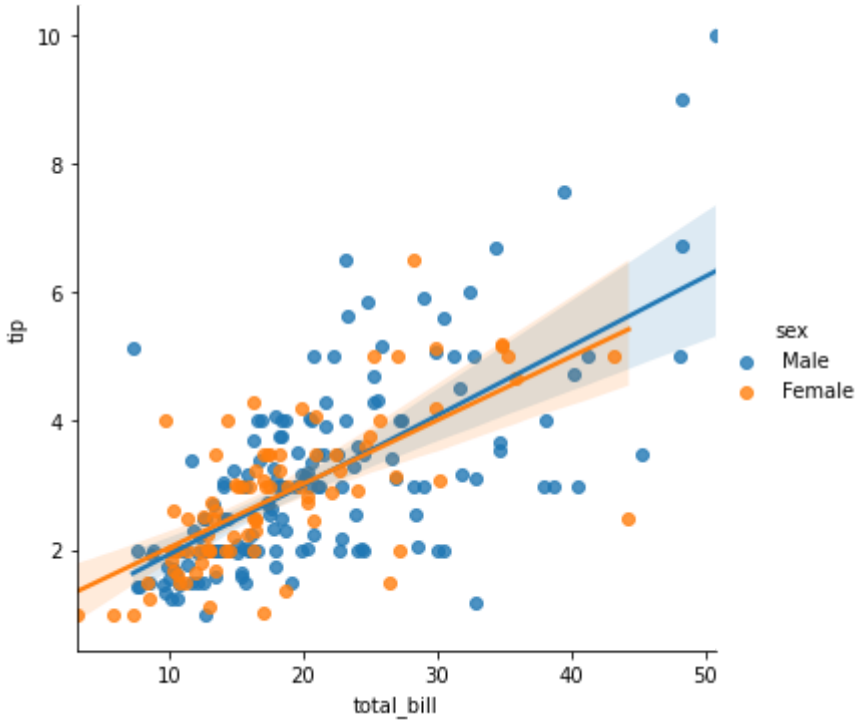
To explore lmplot, we will continue working on the tips dataset:

sns.lmplot(x='total_bill',y='tip',data=tips)



Output

sns.lmplot(x='total_bill',y='tip',data=tips,hue='sex')

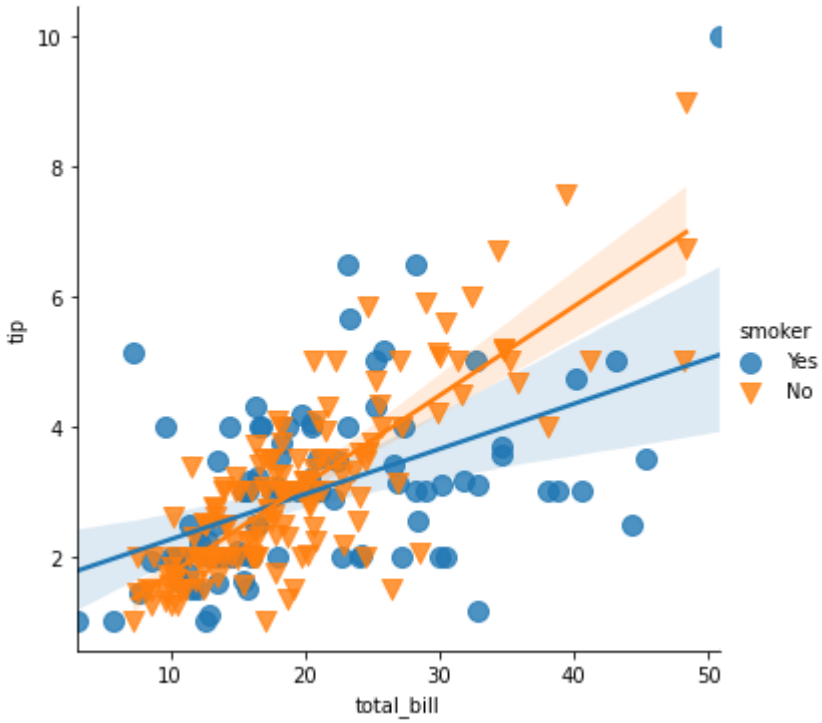


Output

Adding Markers and Using Grid

Underneath the hood, Seaborn is using Matplotlib. Hence we will use Matplotlib arguments to provide markers. There is another attribute, the '**scatter_kws**', which comes from **regplot** (lmplot is an implementation of regplot) which takes in a dictionary with parameter '**s**', which further defines the size of each scatter point.

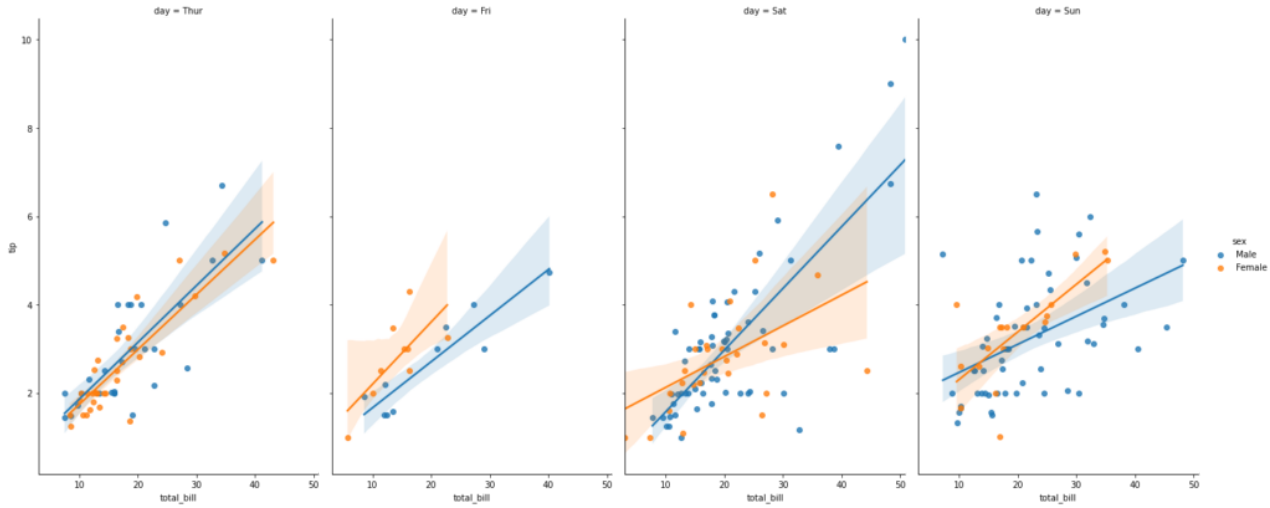
sns.lmplot(x='total_bill', y='tip',data=tips,hue='smoker',markers=['o','v'], scatter_kws={'s':100})



Output

We can add more variable separation through columns and rows with the use of a grid. To plot on a grid, we have to add in the '**col**' and/or '**row**' arguments.

sns.lmplot(x='total_bill', y='tip',data=tips,col='day',hue='sex',aspect=0.6,height=8)



Output

The '**aspect**' and the '**height**' are the aspect ratio and the size attributes of the Seaborn figures themselves. For our grid, we are plotting only one row and many columns which are separated by the days present in the dataset.

If you made it to the end, congrats! You have sufficient Seaborn and statistical plotting knowledge to plot most of the basic plots. Now you can also make better data-driven decisions on which kinds of plots to use for your visualizations. Further topics in Seaborn such as more on Facetgrids, Customization, etc. will be covered in a later article.

iamajit.pythonanywhere.com