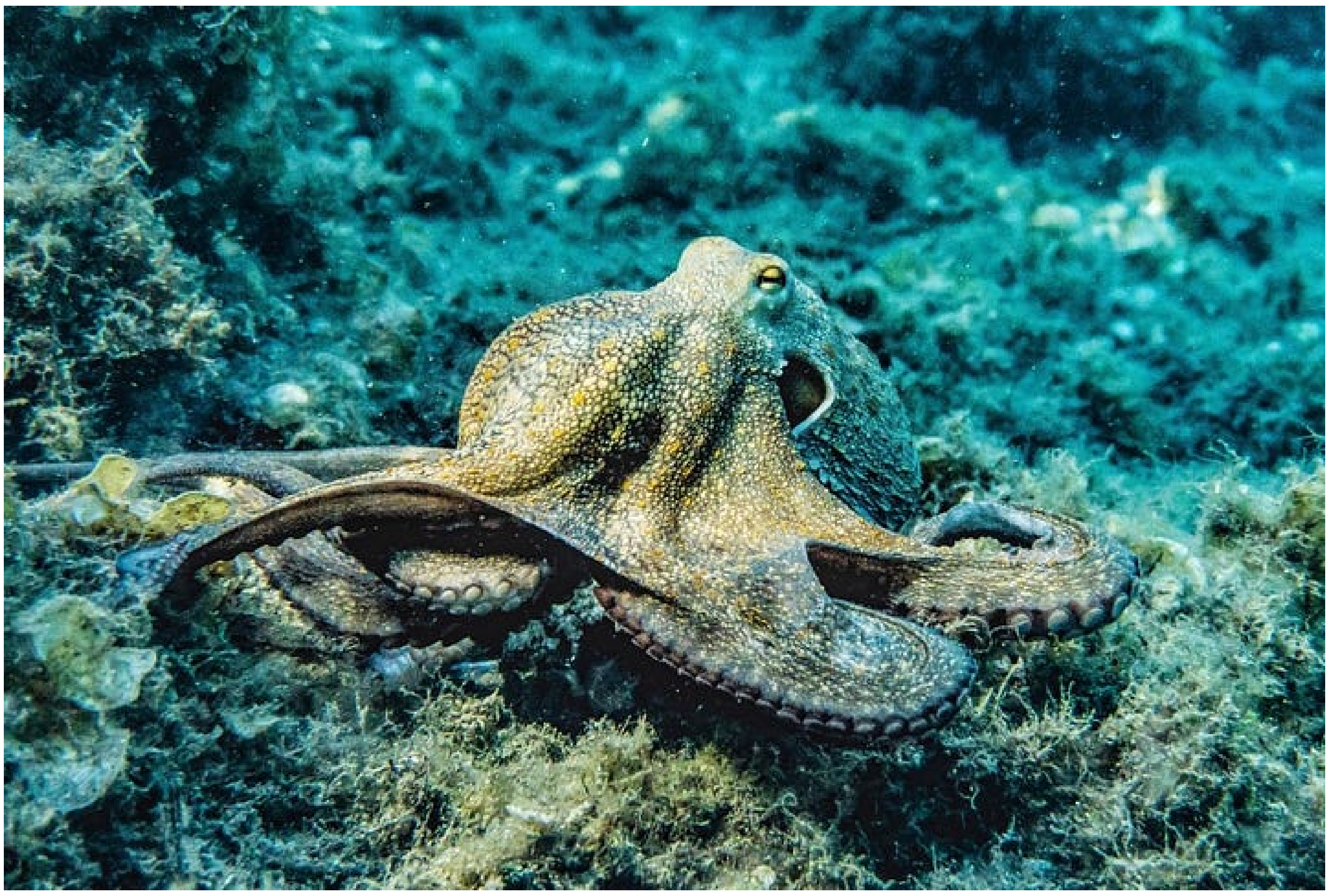
🐼 Master Data Visualization with Seaborn: A Guide to Creating Over 30 Statistical Charts for

Open in app ↗ Sign up Sign In

Search Medium

15 min read · Feb 9

Listen Share



This tutorial aims to build graphs to support the data science process. Visualizations can be used during exploratory analysis, before or after data processing, to construct statistical graphs for dataset analysis, identify variable relationships, or assess data distribution.

While Matplotlib can be used for this purpose, Seaborn is a more efficient and user-friendly library for creating statistical graphs. Hence, having the ability to create visualizations using any tool is crucial.

Visit Jupyter Notebook to explore the concepts of Data Visualization with Seaborn. Note: Key functions, outputs, and terms are bolded for ease of understanding.

Seaborn — Statistical Data Visualization

We'll craft stunning statistical graphs, tweak formatting for optimal presentation, and pre-process data for accurate plotting.

Check out the Seaborn gallery for a collection of impressive statistical chart examples and their accompanying code snippets: https://seaborn.pydata.org/"

Install Seaborn

Execute the command on your operating system — it's just like opening the terminal or command prompt in Windows and typing "pip install seaborn.

```
!pip install seaborn
```

An exclamation keeps us working directly on the Jupyter Notebook.

Loading Packages

```
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import numpy as np
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

```
sns.__version__
'0.11.1'
```

Check Seaborn datasets

Experience the full potential of Seaborn with its built-in datasets.

Simply call `get_dataset_names()` to obtain a comprehensive list of all available datasets for you to experiment with.

```
# Imported datasets with Seaborn
sns.get_dataset_names()
```

```
['anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'iris',
 'mpg',
 'planets',
 'tips',
 'titanic']
```

Load dataset

Access one of the datasets from the comprehensive list. Let's dive into the world-renowned iris dataset for this demonstration.

```
# Loading dataset
iris = sns.load_dataset("iris")
```

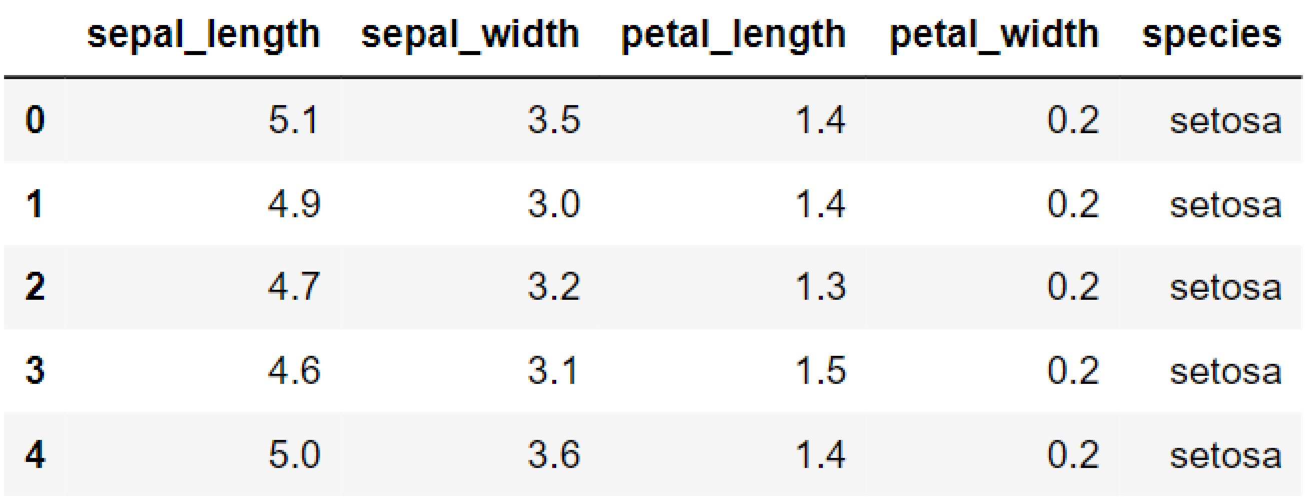
Check type

```
type(iris)
```

```
pandas.core.frame.DataFrame
```

Check first lines

```
iris.head()
```



	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Statistical summary

The "statistical summary function" in the Seaborn library is a method that provides a quick summary of the distribution of a dataset.

This summary typically includes measures such as the mean, median, mode, and quartiles of the data, and can help you to understand the overall pattern and spread of the data.

By visualizing the summary statistics, you can identify trends, outliers, and other important features in your data, which can inform further analysis and decision-making.

```
iris.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

The describe function can be applied to a variety of plot types in Seaborn, including histograms, box plots, violin plots, and others, to help you to quickly gain insights into your data.

Dataset columns

```
iris.columns
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'], dtype='object')
```

Seaborn Statistical Charts

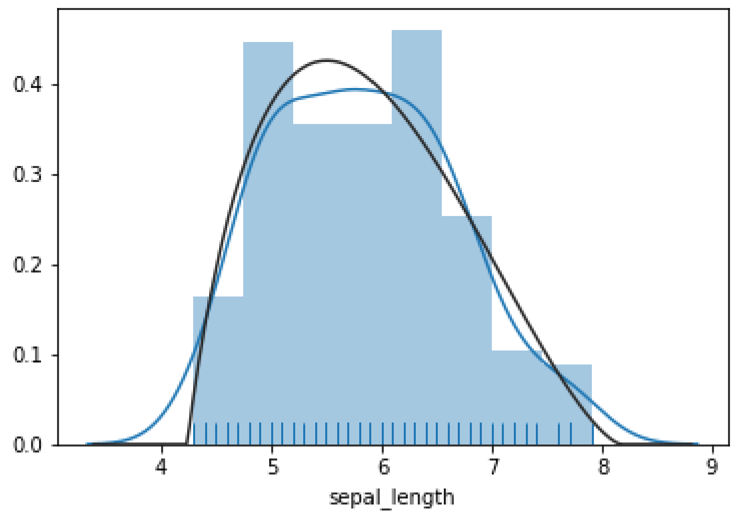
Distplot

The Seaborn distplot, also known as a univariate distribution plot, is used to visualize the distribution of a single variable. To create the plot, you simply call the `distplot` function, passing in the name of the variable you wish to visualize, such as 'sepal_length' from the iris dataset.

Additionally, you can specify options such as enabling the rug plot and adjusting the fit of the data to your preferences.

The distplot provides a comprehensive visual representation of the distribution of your data, including the central tendency, spread, skewness, and any potential outliers, allowing you to gain a deeper understanding of data."

```
sns.distplot(iris.sepal_length, rug = True, fit = stats.gausshyper);
```

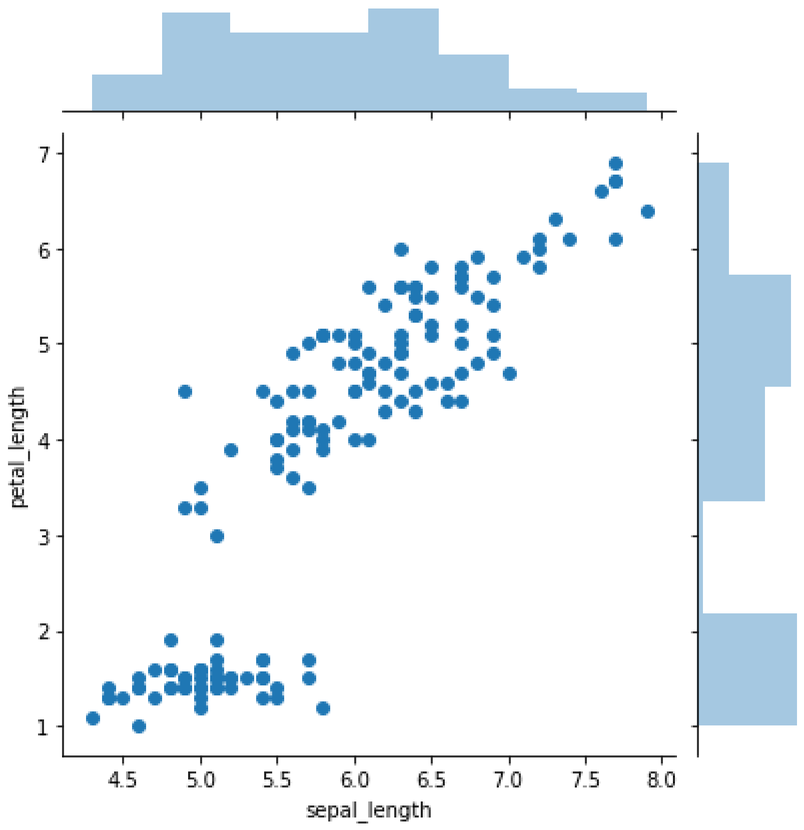


Jointplot

The Seaborn jointplot is a powerful tool for visualizing bivariate distributions. It combines a scatterplot with histograms of the variables on each axis, providing a comprehensive view of the relationship between two variables.

This single plot can reveal important information about the distribution of your data, including the presence of positive or negative relationships, the distribution of each variable, and the frequency of data points.

```
# Scatterplot - Bivariate Distribution
sns.jointplot(x = "sepal_length", y = "petal_length", data = iris);
```



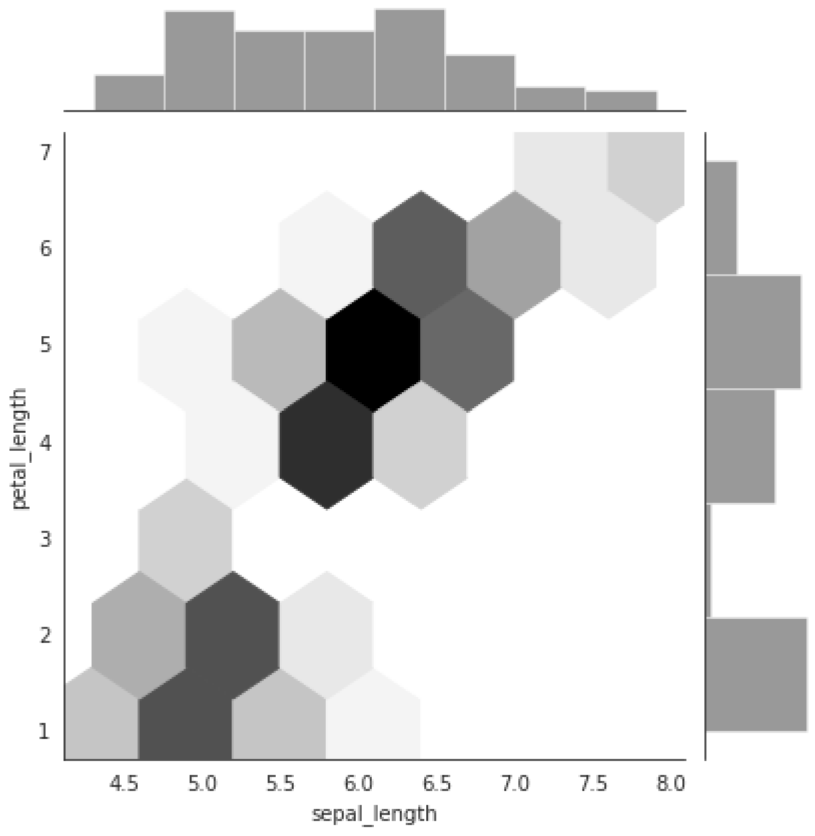
With just one simple command, the jointplot provides an efficient and effective way to gain insights into the relationships between variables in your data.

Hex Jointplot

The Seaborn jointplot hex is a variation of the jointplot, which presents the bivariate distribution in a different format.

Instead of using dots to represent individual data points, the hex chart uses hexagons to group and display the density of data points. This can provide a clearer representation of the relationship between two variables, particularly when there is a high density of data points.

```
# Useful graph when working with large datasets
with sns.axes_style("white"):
     sns.jointplot(x = "sepal_length",
                   y = "petal_length",
                   data = iris,
                   kind = "hex",
                   color = "k");
```



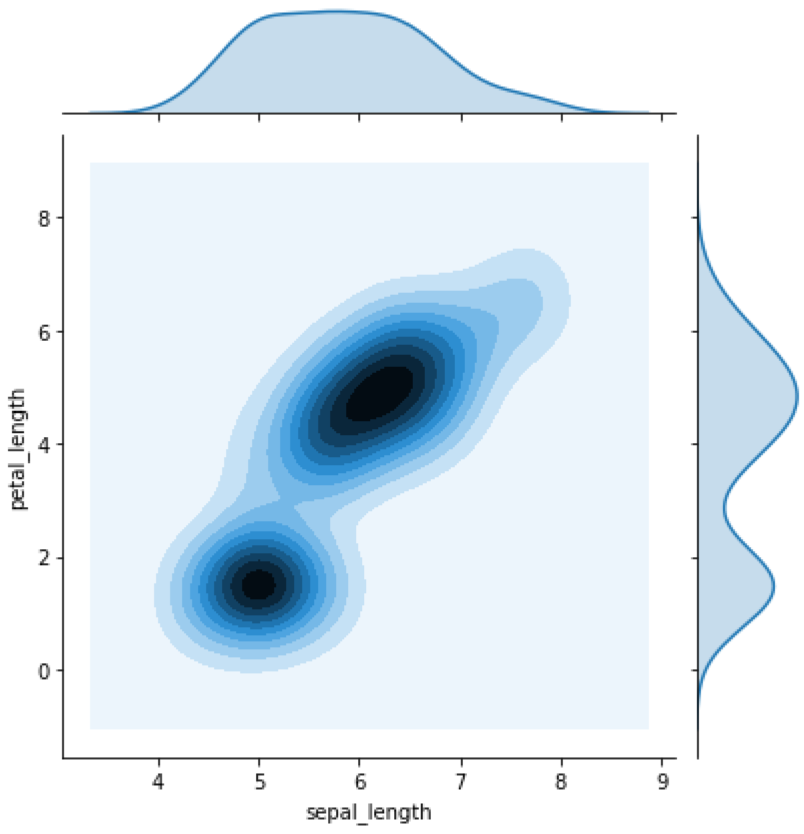
Additionally, the layout of the plot can be easily customized to fit your specific needs, allowing for greater control over the appearance and presentation of your data.”

Density Jointplot

The Seaborn density plot provides an alternative representation of the bivariate distribution by changing the display of data points to a kernel density estimate (KDE).

Instead of individual data points, the plot shows the estimated density of the data. This can provide a smoother representation of the distribution of the data and reveal underlying patterns that might not be immediately apparent in a scatterplot.

```
# Bivariate Distribution
sns.jointplot(x = "sepal_length",
              y = "petal_length",
              data = iris,
              kind = "kde");
```



Additionally, the histograms on the axes are also transformed into density plots, providing a more comprehensive view of the distribution of each variable.

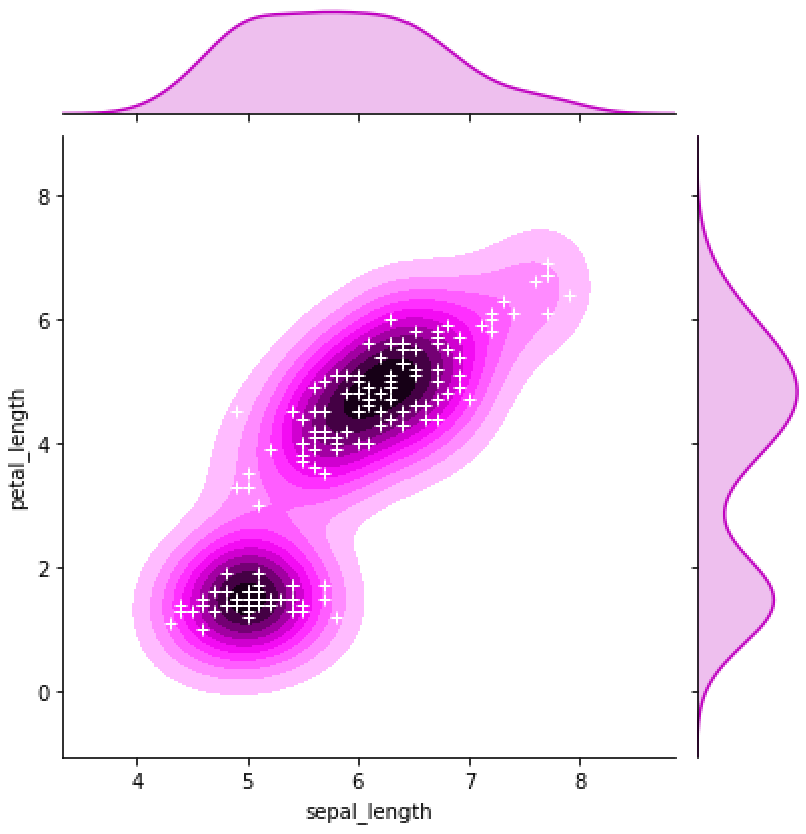
Customizing Density Jointplot

To showcase the versatility and capability of Seaborn, here we have an example of customizing the parameters of the previous density jointplot.

With the ability to adjust various aspects of the plot, such as color, marker style, and plot appearance, Seaborn provides a flexible and robust platform for data visualization, allowing you to create plots that effectively communicate the insights contained in your data.

```
# Bivariate Distribution
g = sns.jointplot(x = "sepal_length",
                  y = "petal_length",
                  data = iris,
                  kind = "kde",
                  color = "m")
```

```
g.plot_joint(plt.scatter, c = "w", s = 30, linewidth=1, marker="+")
g.ax_joint.collections[0].set_alpha(0);
```

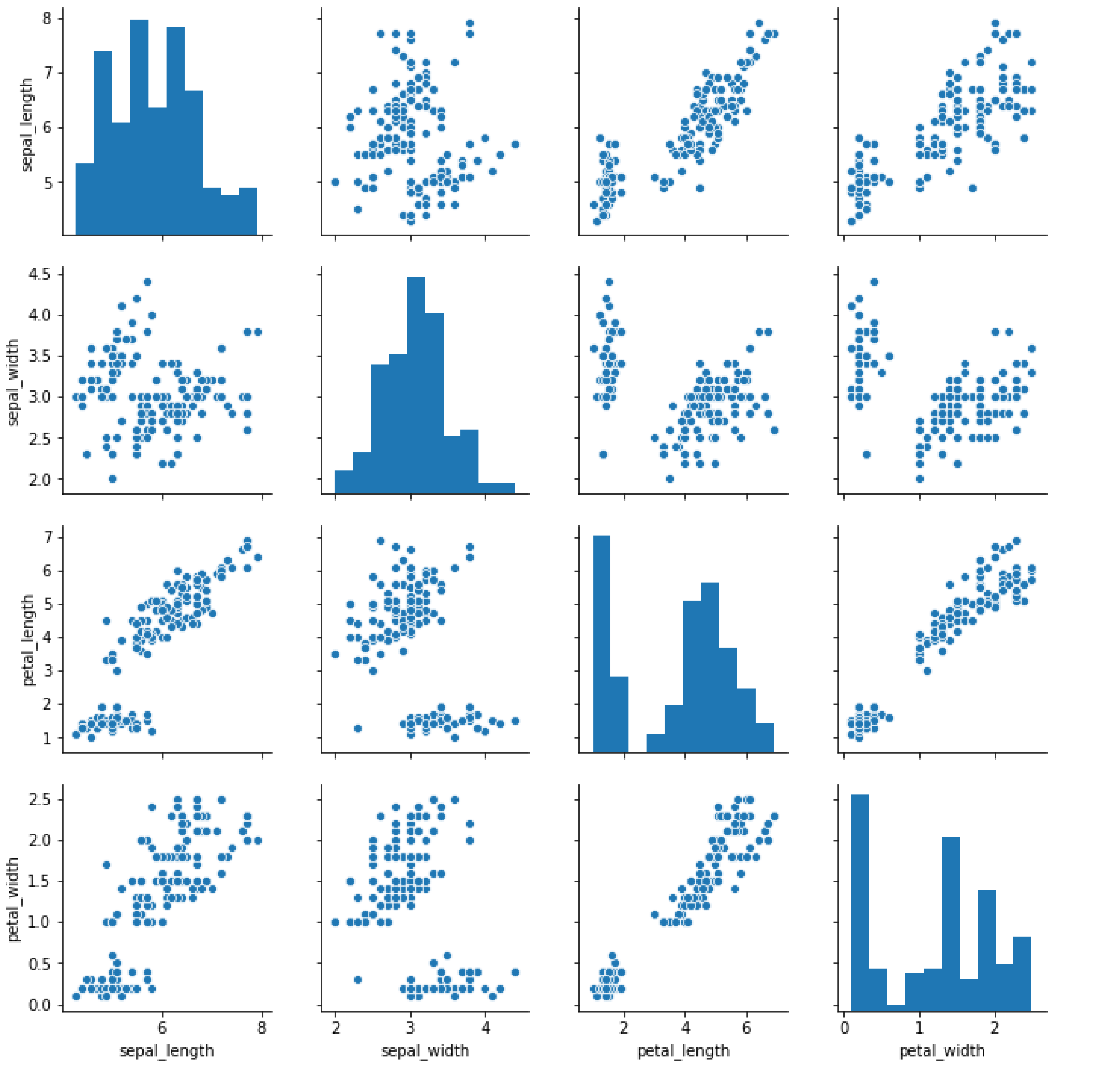


Pairplot — Handy graph for few variables

The Seaborn pairplot provides a comprehensive view of the relationships between all variables in a dataset.

By calling the pairplot function and passing in the dataset, Seaborn automatically creates scatterplots and histograms for all combinations of variables, displaying their relationships in a single plot.

```
# Bivariate Distribution
sns.pairplot(iris);
```



This plot is a powerful tool for gaining a general understanding of the distribution of variables in your data and can quickly reveal any significant relationships or patterns.

With minimal input from the user, the pairplot function can produce a highly informative plot that is valuable for exploratory data analysis.

Relationship Visualizations

Load dataset

Seaborn offers a vast array of options for visualizing data. To demonstrate this, we will be exploring another dataset, the 'tips' dataset, which provides a wealth of information just waiting to be visualized.

```
# Loading tips dataset
tips = sns.load_dataset("tips")
```

Check type

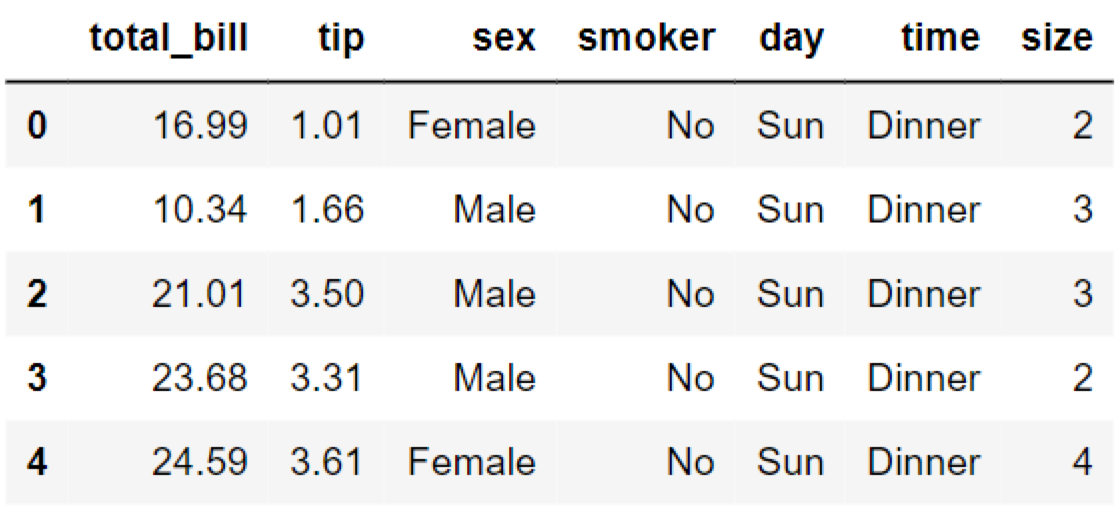
Checking the data type of your variables in Pandas is important for several reasons. First, it helps you to understand the structure of your data and identify any potential errors or inconsistencies.

For example, if you expect a column to contain numerical data but it is instead stored as text, this can cause problems when you try to perform numerical operations on that column.

```
type(tips)
pandas.core.frame.DataFrame
```

Check first lines

```
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

Statistical summary

```
tips.describe()
```

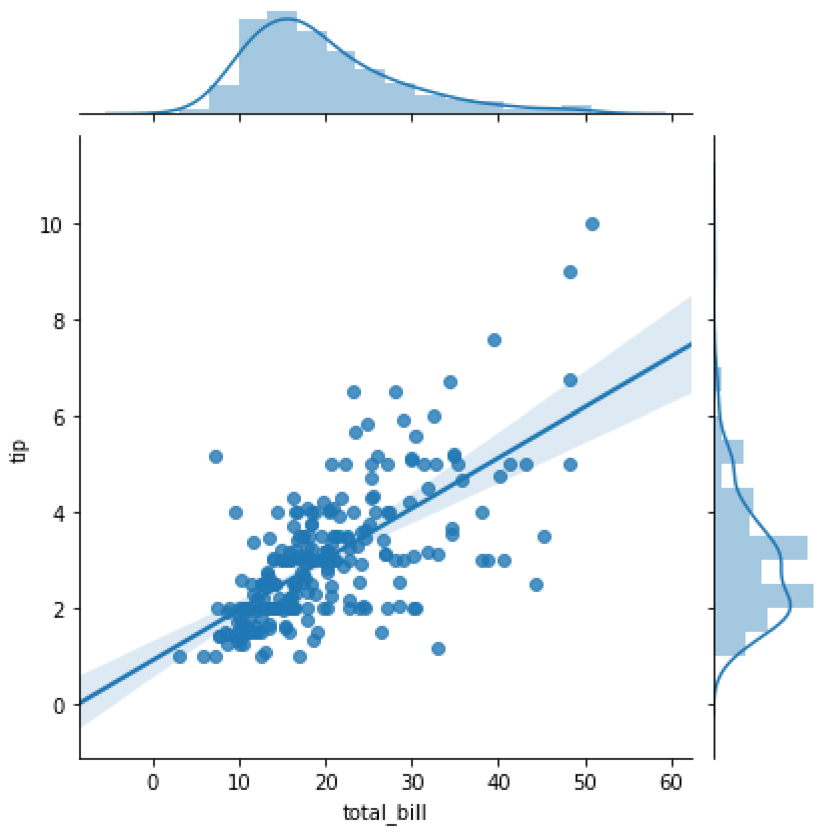
	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000
25%	13.347500	2.000000	2.000000
50%	17.795000	2.900000	2.000000
75%	24.127500	3.562500	3.000000
max	50.810000	10.000000	6.000000

Jointplot — Linear Regression

The joint plot with linear regression in Seaborn is a powerful tool for exploring the relationship between two variables.

By passing the “reg” argument to the “kind” parameter, the plot creates a scatterplot and fits a linear regression line to the data, showing the strength of the relationship between the two variables.

```
# Scatterplot with regression line - Bivariate Distribution
sns.jointplot(x = "total_bill",
              y = "tip",
              data = tips,
              kind = "reg");
```



In addition, the plot also includes histograms on the axes to show the distribution of each variable and a density plot to provide a visual representation of the density of the data points.

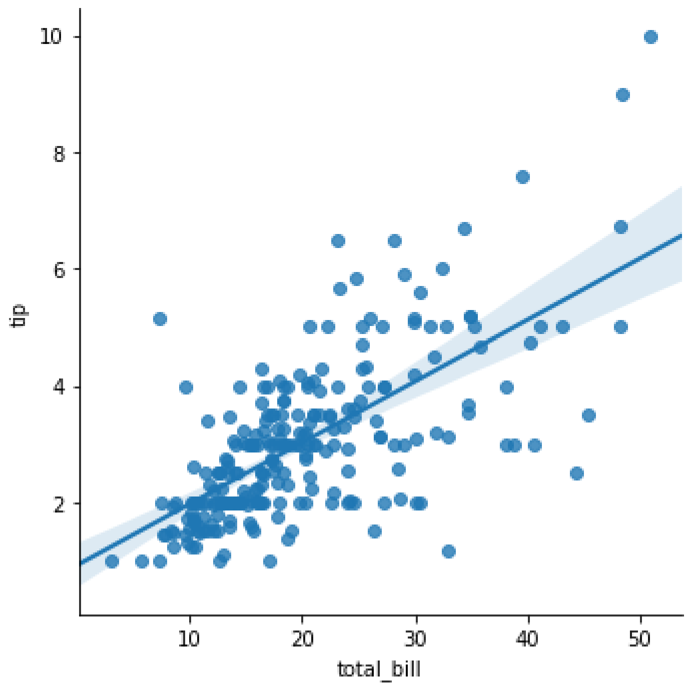
This combination of features makes the joint plot with linear regression an incredibly useful tool for understanding the relationships in your data and exploring potential trends or patterns.

Lmplot

The lmplot in the Seaborn library is a powerful tool for visualizing the relationship between two variables in a scatterplot with a fitted linear regression line.

Unlike the joint plot, the lmplot only shows the scatterplot and regression line, allowing for a clearer focus on the relationship between the variables.

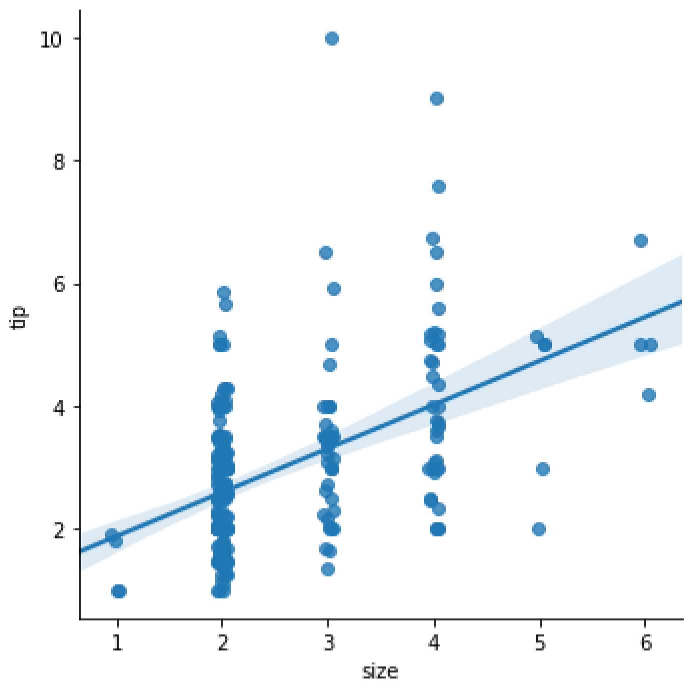
```
# Linear Regression (uses 95% confidence interval by default
sns.lmplot(x = "total_bill",
           y = "tip",
           data = tips);
```



The lmplot also offers a range of customization options, including the ability to control the appearance of the data points and the regression line.

This versatility makes the lmplot a useful tool for exploring and understanding the relationships between variables in your data, and for presenting your results in a clear and compelling way.

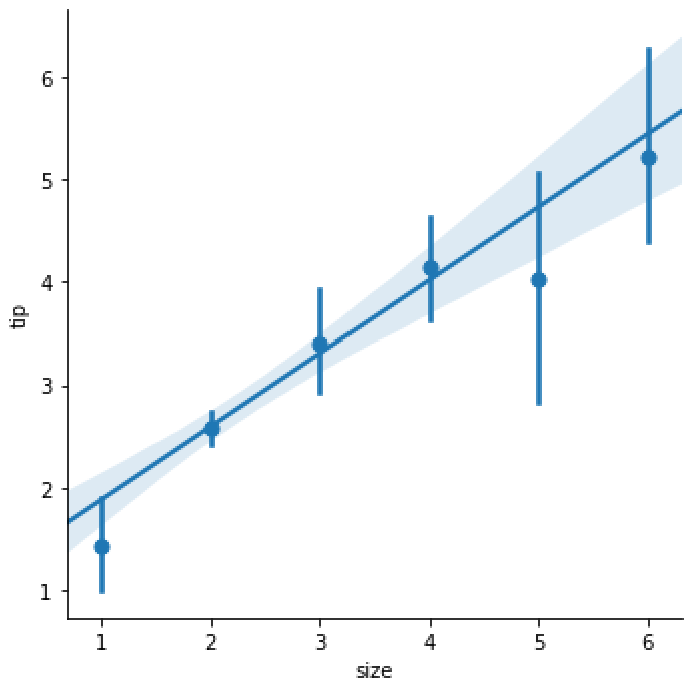
```
sns.lmplot(x = "size",
           y = "tip",
           data = tips,
           x_jitter = .05);
```



Limits

This plot not only displays a scatterplot with a fitted linear regression line, but it also adds upper and lower limits to each data point, providing a visual representation of the uncertainty in the data.

```
sns.lmplot(x = "size",
           y = "tip",
           data = tips,
           x_estimator = np.mean);
```



By visualizing the limits, you can better understand the relationships between variables and make informed decisions about the data and your analysis.

Load Dataset

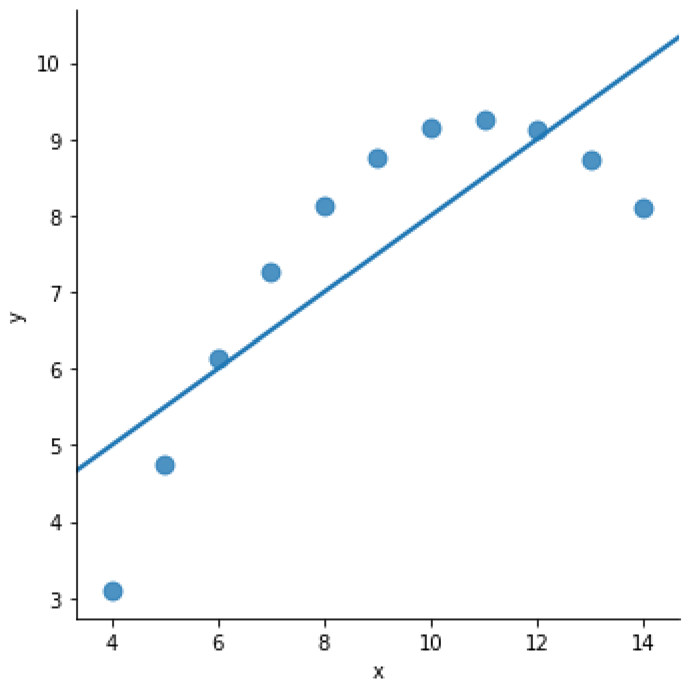
Let's load another dataset from Seaborn:

```
# Loading anscombe dataset
anscombe = sns.load_dataset("anscombe")
```

Query — filter + regression

We can filter the data from the dataset to plot it in a chart, creating a kind of query.

```
# Non-linear relationship
sns.lmplot(x = "x",
           y = "y",
           data = anscombe.query("dataset == 'II'"),
           ci = None,
           scatter_kws = {"s": 80});
```

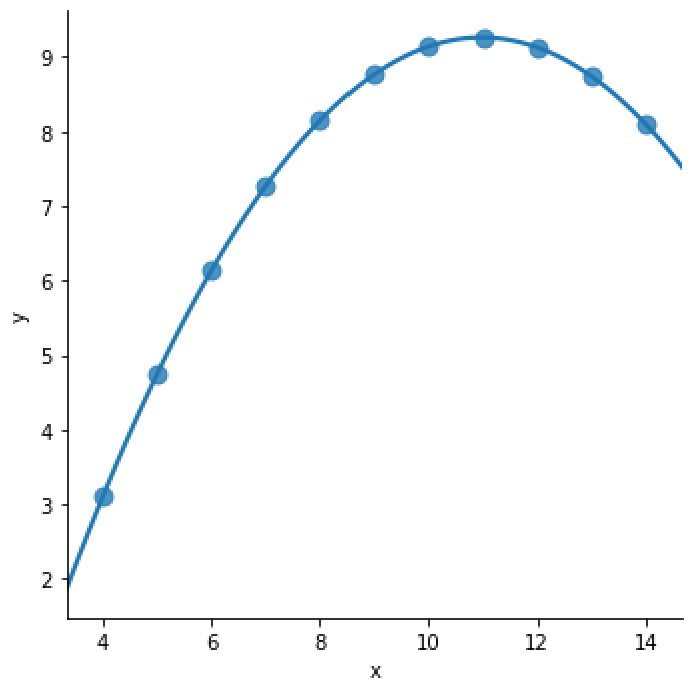


Parameter adjustment

The adjustment of parameters in seaborn allows you to customize the appearance of your charts, including the regression line.

By making tweaks to the parameters, you can fine-tune the visualization of your data, making it easier to understand the relationships between variables and identifying trends or patterns in the data.

```
# We can adjust the parameters to fit the curve
sns.lmplot(x = "x",
           y = "y",
           data = anscombe.query("dataset == 'II'"),
           order = 2,
           ci = None,
           scatter_kws = {"s": 80});
```



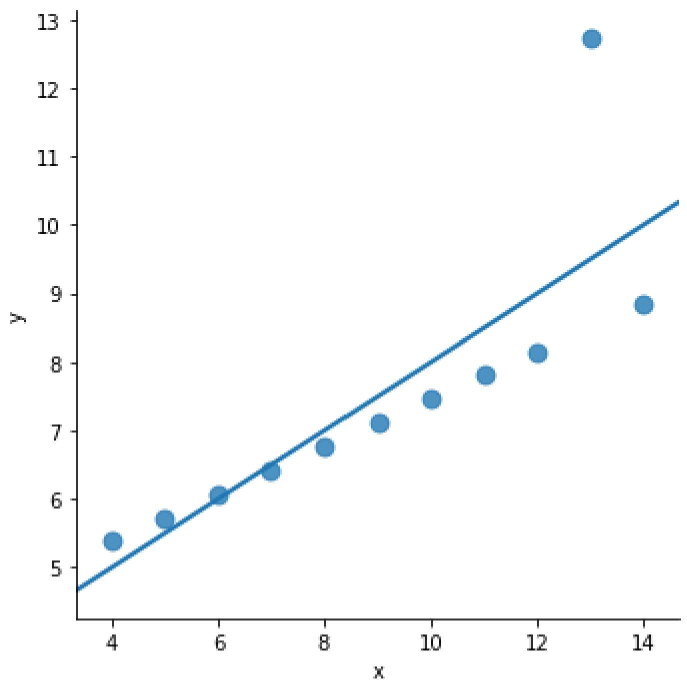
Whether you want to change the line color, marker style, or axis labels, the ability to adjust parameters gives you the flexibility to present your data in the way that best supports your analysis.

Detecting outliers

Detecting outliers is crucial as it helps in identifying data points that deviate significantly from the normal pattern of the data.

By identifying these outliers, one can gain a deeper understanding of the distribution of the data and make informed decisions based on the insights.

```
# Visualizing outliers
sns.lmplot(x = "x",
           y = "y",
           data = anscombe.query("dataset == 'III'"),
           ci = None,
           scatter_kws = {"s": 80});
```

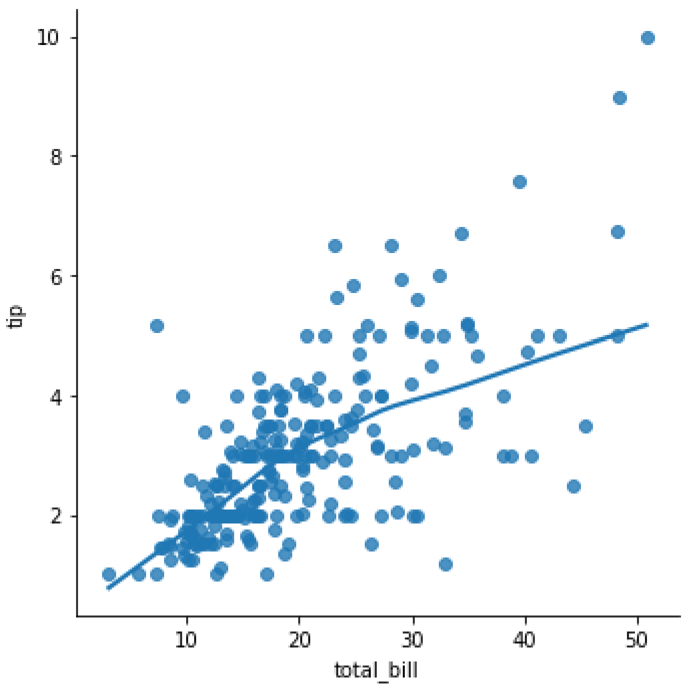


In Seaborn, the visualization of outliers is an essential step in exploring the dataset. This can be achieved through various techniques such as scatter plots, box plots, and others.

Nonlinear relationship

We can visualize a nonlinear relationship, where the change in one variable is not proportionally linked to the change in another variable.

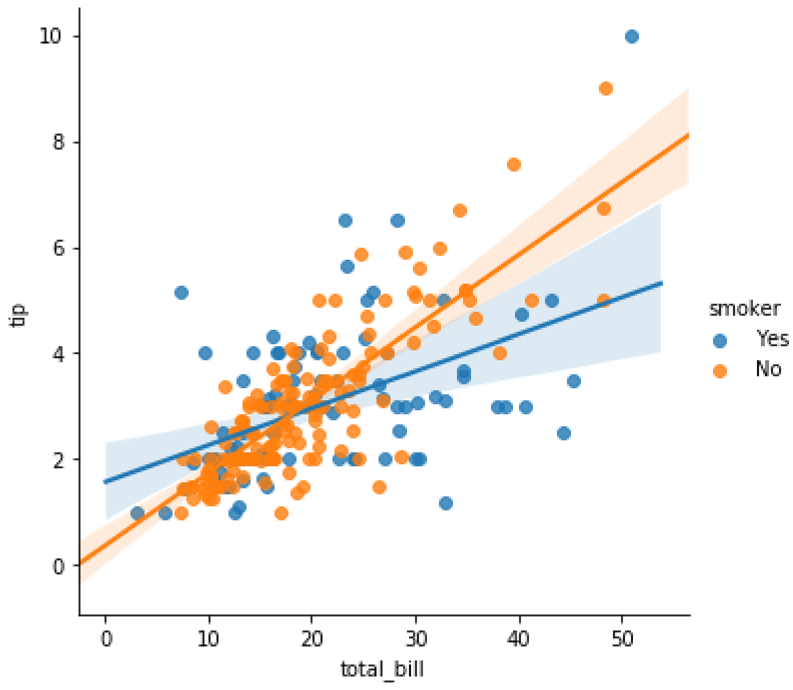
```
# Using lowess smoother for variable with non-linear relationships
sns.lmplot(x = "total_bill",
           y = "tip",
           data = tips,
           lowess = True);
```



Different pieces of information

Another lmplot displaying various pieces of information.

```
# Using more than 2 variables
sns.lmplot(x = "total_bill",
           y = "tip",
           hue = "smoker",
           data = tips);
```

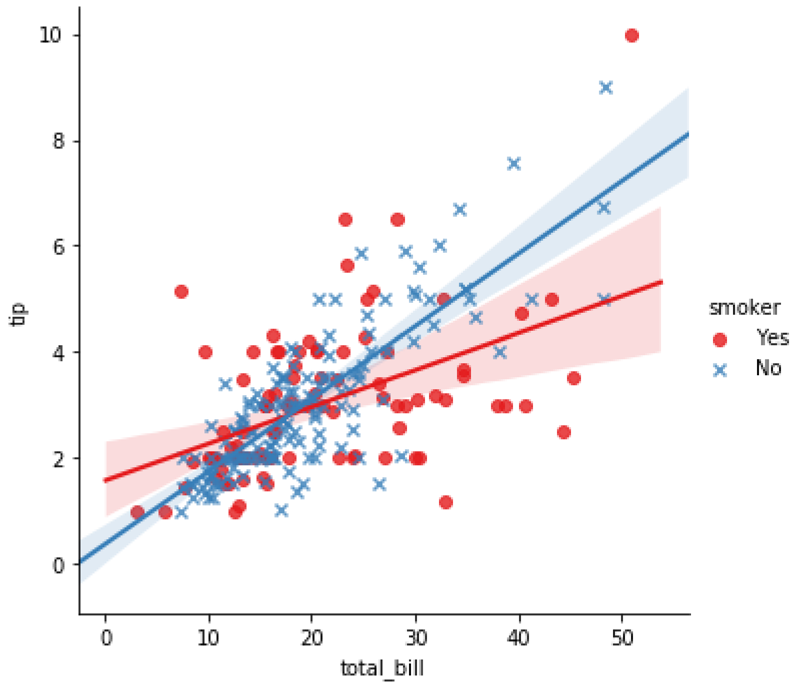


Customize chart

The importance of customizing charts in seaborn lies in the ability to visually highlight the differences between variables.

By adjusting the parameters, we can create graphs that clearly and effectively communicate the information we want to convey.

```
# Changing the chart setting
sns.lmplot(x = "total_bill",
           y = "tip",
           hue = "smoker",
           data = tips, markers = ["o", "x"],
           palette = "Set1");
```



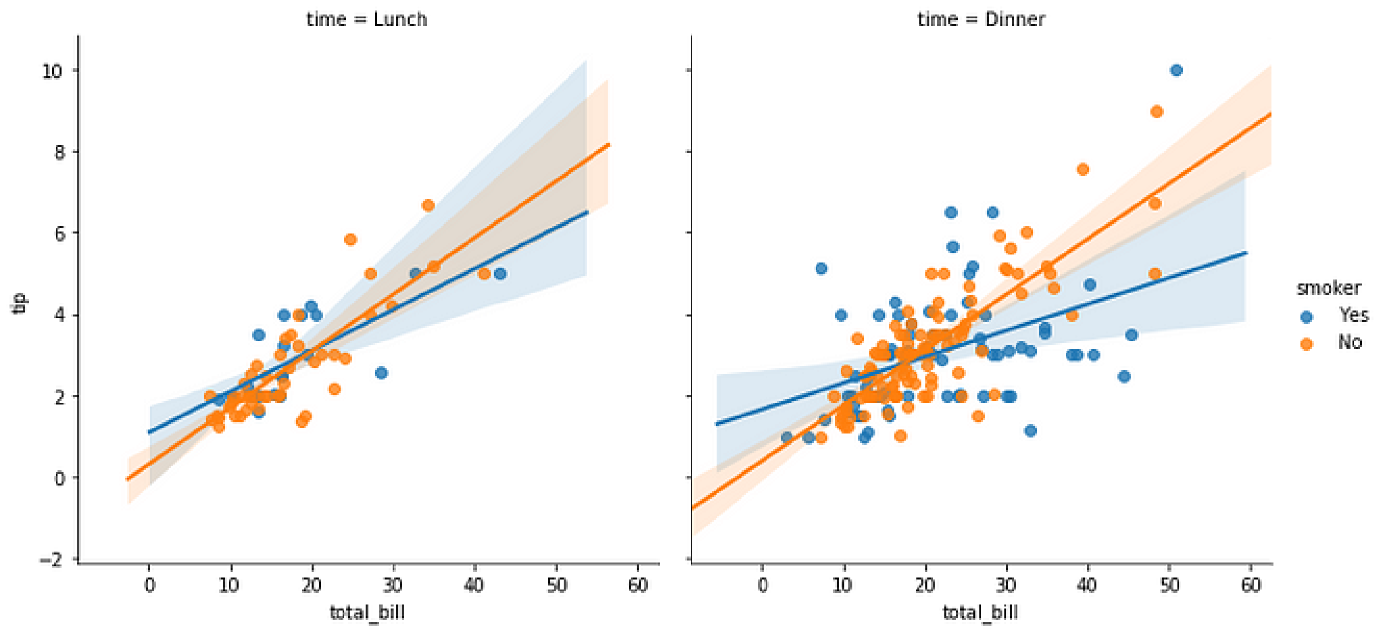
Customization is a key tool in the arsenal of data visualization and can greatly enhance the impact of a graph.

Split area

We have the option to split the plot area into multiple sections. The entire plot area is referred to as the plot area, and above we have a single chart in the plot area, while below we have two charts in the plot area.

For example, we will examine the “tip” variable on the y-axis. We use the same variable for two charts and make a change in the “col” parameter, dividing the charts into “total_bill” corresponding to lunchtime and dinner time.

```
# Dividing the drawing area
sns.lmplot(x = "total_bill",
           y = "tip",
           hue = "smoker",
           col = "time",
           data = tips);
```

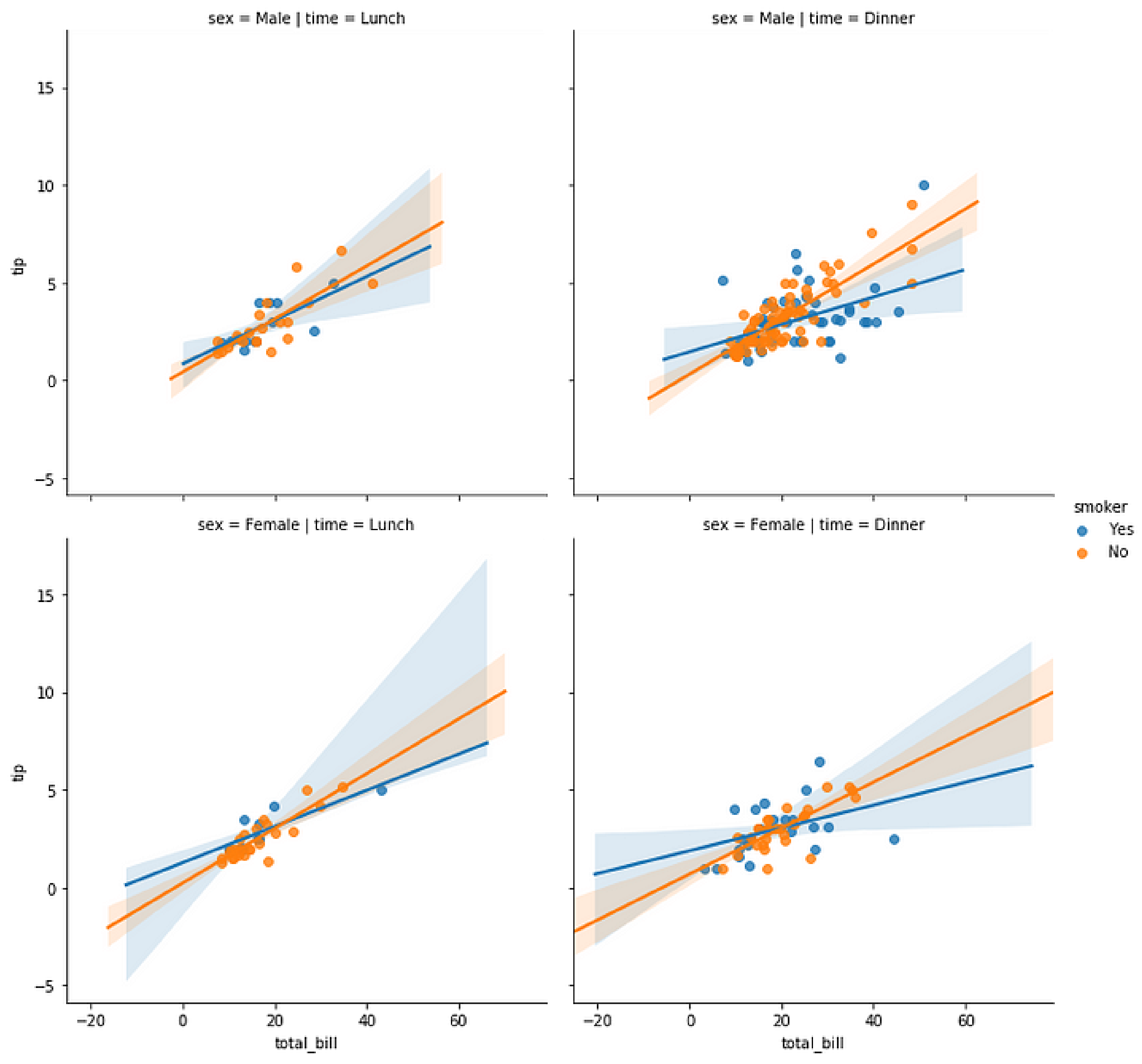


With these charts, we have the ability to make predictions based on the information displayed. By analyzing the relationship between the tip and total bill, we can estimate the expected tip amount during lunch or dinner time.

Divide areas with more variables

By dividing the area into multiple blocks, we can better understand the trends and patterns in the data and make more informed predictions. By incorporating multiple variables, we can build a more comprehensive understanding of the data and draw more meaningful insights from it

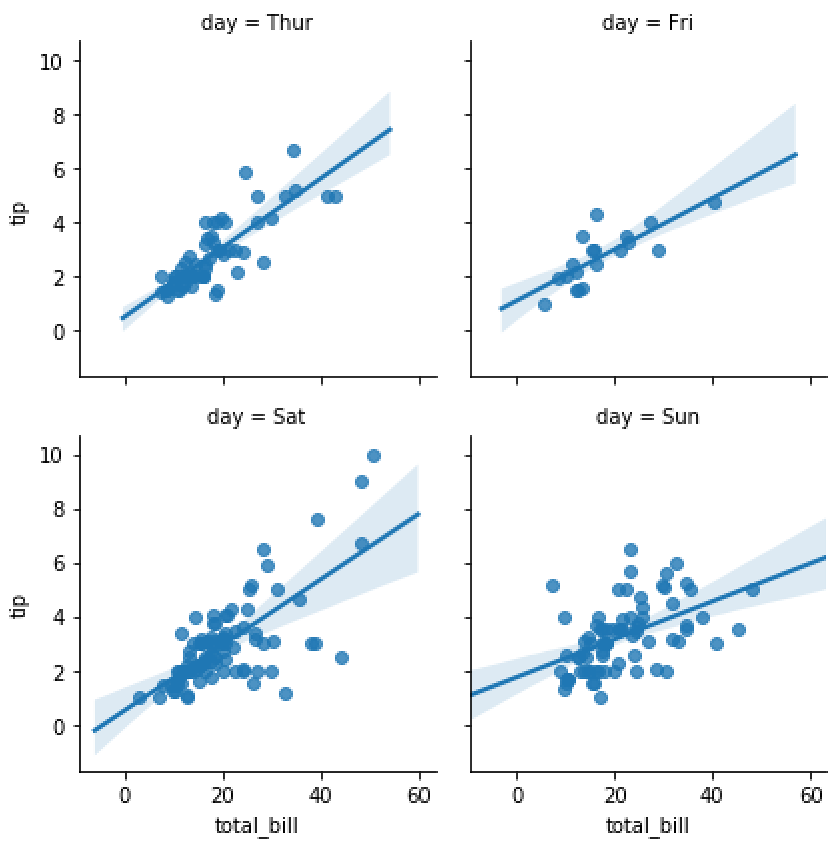
```
# Dividing the drawning area
sns.lmplot(x = "total_bill",
           y = "tip",
           hue = "smoker",
           col = "time",
           row = "sex",
           data = tips);
```



• Split area

We can further divide the plot area by changing the `col` parameter to `days`, thereby creating different blocks of data for each day of the week.

```
# Dividing the drawining area
sns.lmplot(x = "total_bill",
           y = "tip",
           col = "day",
           data = tips,
           col_wrap = 2,
           size = 3);
```

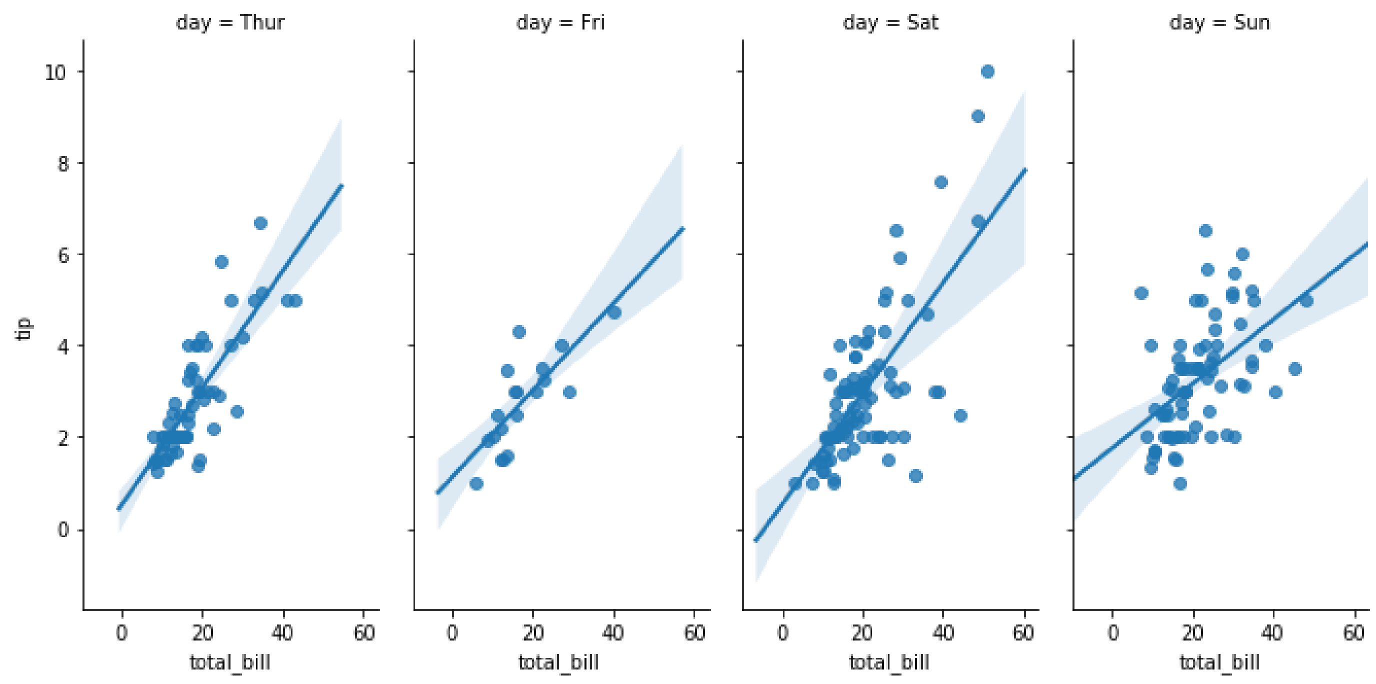


Split horizontal area

Instead of having multiple plots stacked vertically, we have now changed the orientation to have them side by side.

The days of the week are now all in one row, providing a horizontal visual representation with the change in parameters from col_wrap.

```
# Dividing the drawning area
sns.lmplot(x = "total_bill",
           y = "tip",
           col = "day",
           data = tips,
           aspect = .5);
```



Working with Categorical variables

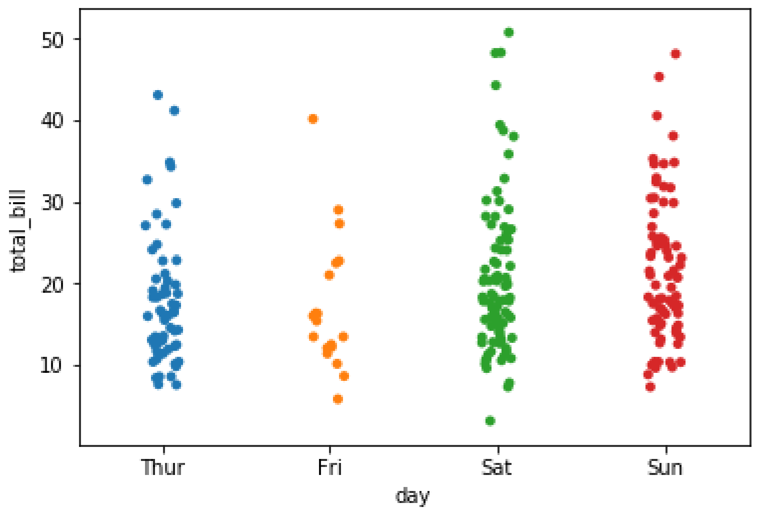
The representation of categorical variables, or variables that contain string values, is an important aspect of data visualization.

Until now, we have explored charts for numerical variables, but it's equally important to visually analyze the relationships between categorical data and other variables in our dataset.

Stripplot

Now we aim to identify the total bill per day of the week. Since the day of the week is a categorical variable, it must be represented differently.

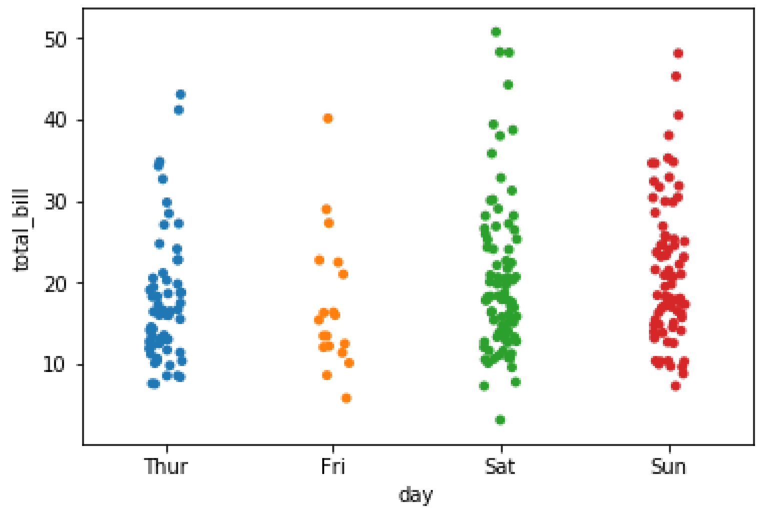
```
# stripplot
sns.stripplot(x = "day",
              y = "total_bill",
              data = tips);
```



Customize Stripplot

We can make slight customizations to the Stripplot in seaborn. With these modifications, we can create a chart that is more compact and concise than the previous one.

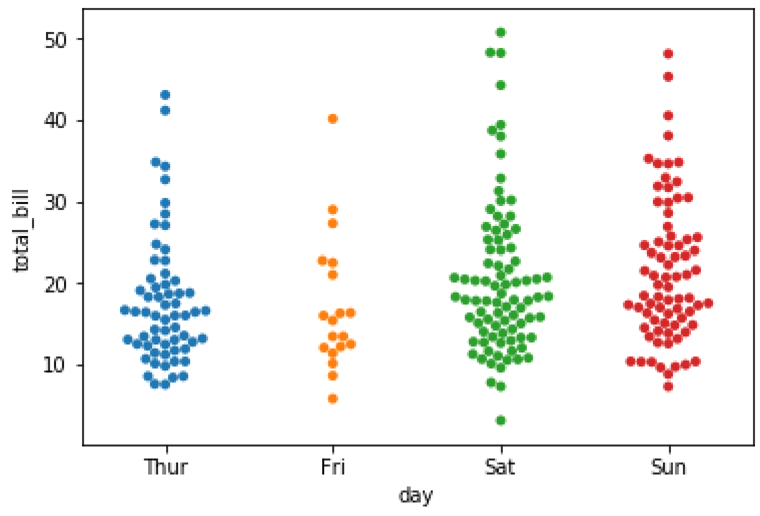
```
# stripplot
sns.stripplot(x = "day",
              y = "total_bill",
              data = tips,
              jitter = True);
```



Swarmplot

The Seaborn Swarmplot is similar to the previous chart, but it represents the data points in a way that avoids overlap.

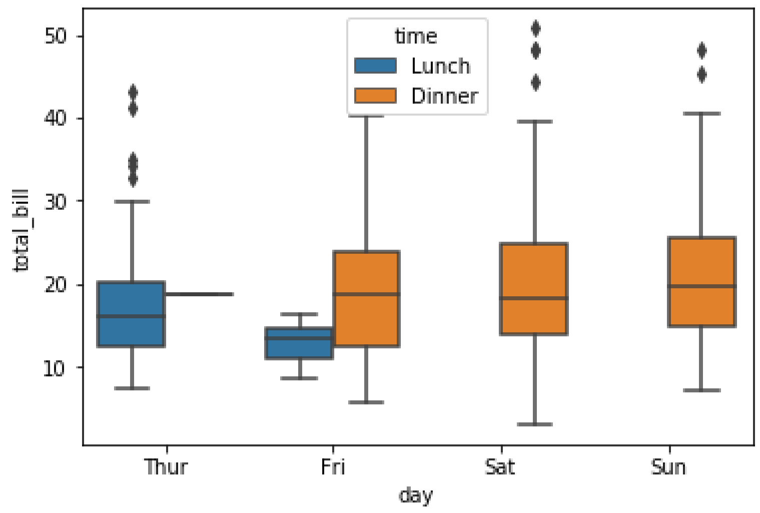
```
# swarmplot - Avoiding Categorical overlap points
sns.swarmplot(x = "day",
              y = "total_bill",
              data = tips);
```



Boxplot

The Seaborn boxplot is a popular chart in statistics when dealing with categorical variables. It displays outliers, or values that deviate from the typical pattern of the data representation.

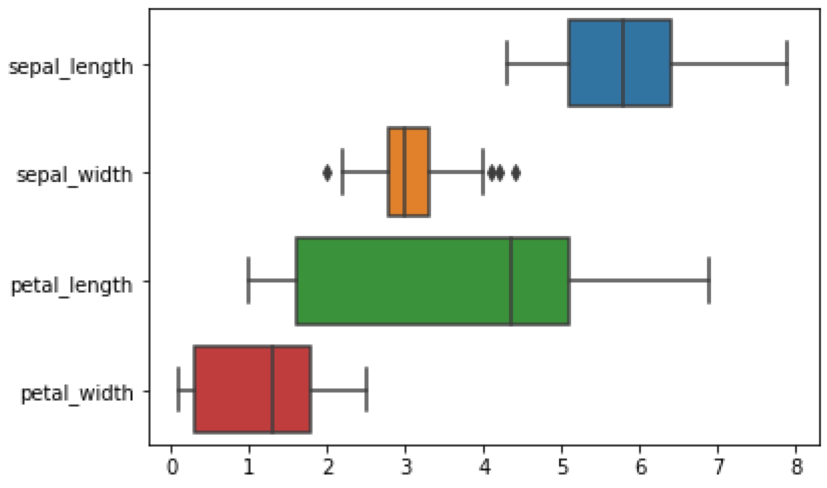
```
# boxplot
sns.boxplot(x = "day",
            y = "total_bill",
            hue = "time",
            data = tips);
```



Horizontal Boxplot

We can also adjust the orientation of the boxplot by changing it to a horizontal layout.

```
# boxplot
sns.boxplot(data = iris,
            orient = "h");
```

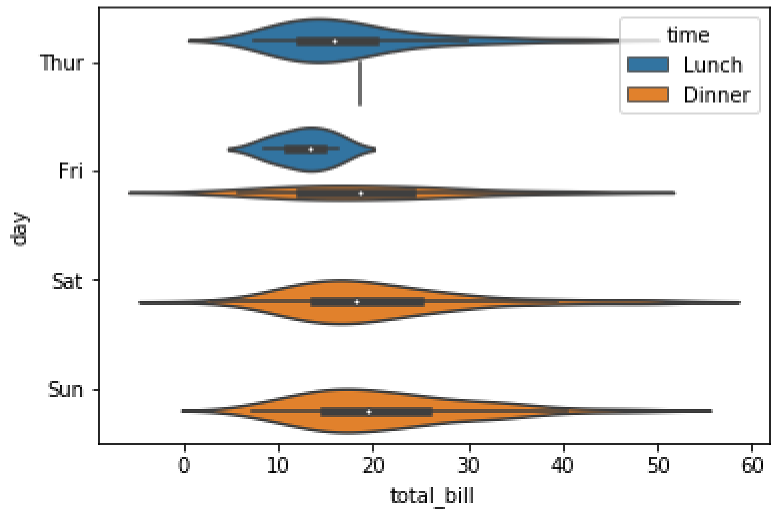


Violin plot

The violin plot is a useful chart in seaborn for visualizing the distribution of a variable.

It is a combination of a box plot and a kernel density plot, and it shows the density of the data along with the range and median values.

```
# violinplot
sns.violinplot(x = "total_bill",
               y = "day",
               hue = "time",
               data = tips);
```



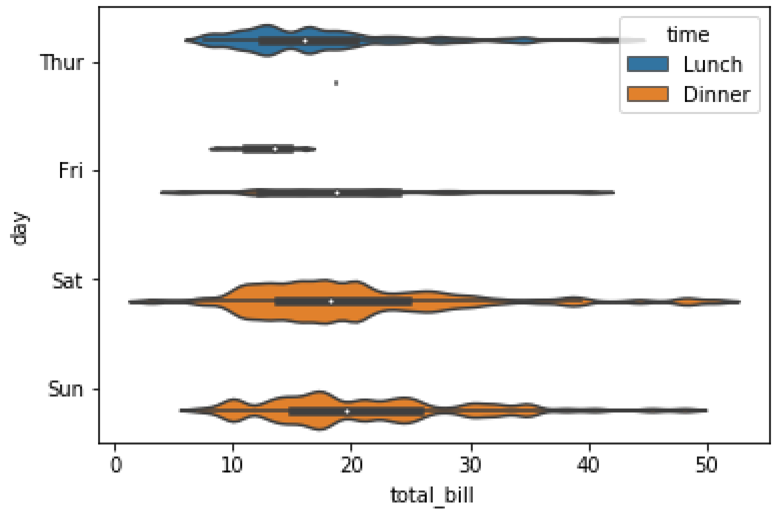
This type of plot is particularly useful for understanding the distribution of multi-modal data or for comparing the distributions of different groups or categories.

By using the violin plot, we can gain a deeper insight into the distribution of a variable and make more informed decisions based on our analysis.

Customizing Violin plot

We can make some customization to the violin plot to make the shape of the violins narrower.

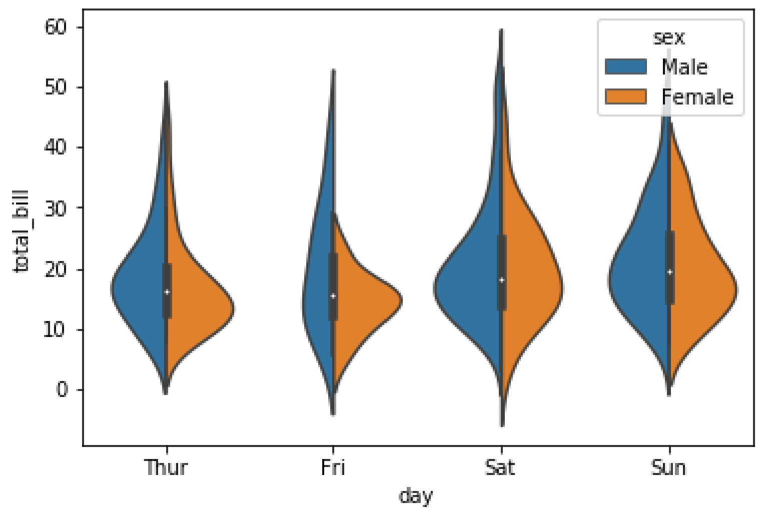
```
# violinplot
sns.violinplot(x = "total_bill",
               y = "day",
               hue = "time",
               data = tips,
               bw = .1,
               scale = "count",
               scale_hue = False);
```



Vertical plot

The Violin plot can be displayed in a vertical orientation.

```
# violinplot
 sns.violinplot(x = "day",
               y = "total_bill",
               hue = "sex",
               data = tips,
               split = True);
```

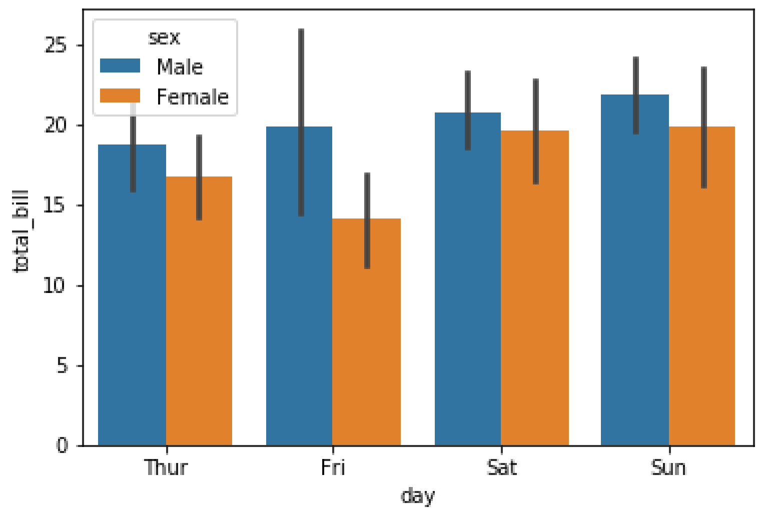


Barplot

Another chart commonly used for representing categorical variables is the bar plot.

Bar plots are a commonly used chart for categorical variables and provide a visual representation of data through the height of bars.

```
# barplot
sns.barplot(x = "day",
            y = "total_bill",
            hue = "sex",
            data = tips);
```

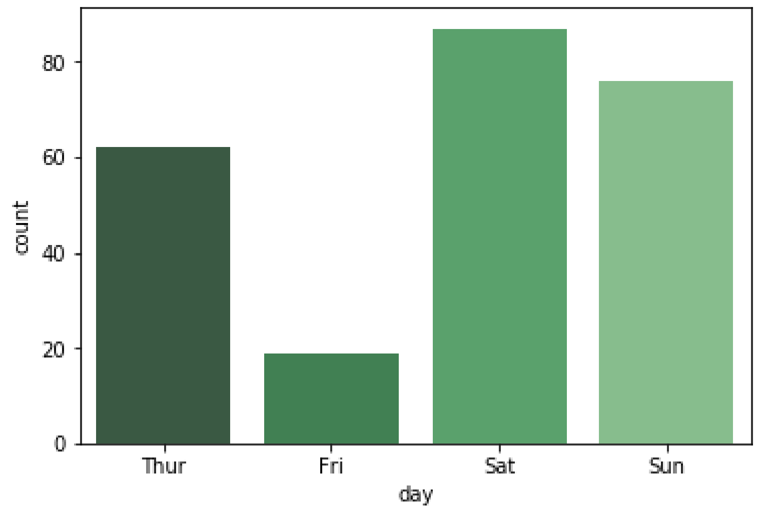


They help to quickly compare different categories and identify trends and patterns. Additionally, bar plots allow for easy customization and can be tailored to better fit the data being analyzed.

Countplot

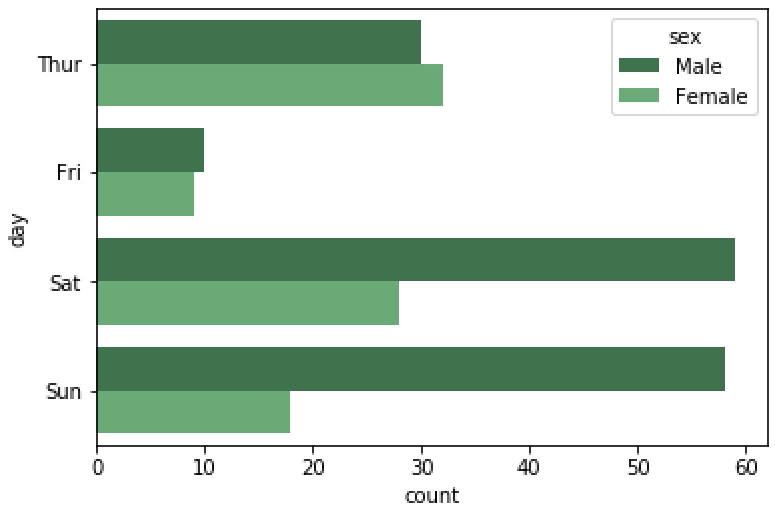
The countplot is a graph used to display the count of elements for each category in a categorical variable. In this case, it can be used to show the count of elements for each day of the week.

```
# countplot
sns.countplot(x = "day",
              data = tips,
              palette = "Greens_d");
```



We can make changes to the countplot to display the number of individuals per gender per day of the week in a horizontal orientation.

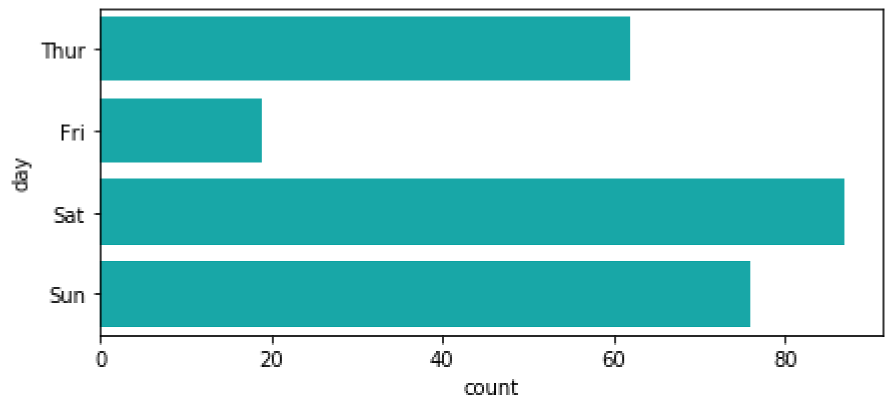
```
# countplot
sns.countplot(y = "day",
              hue = "sex",
              data = tips,
              palette = "Greens_d");
```



Continuous Countplot

Here we see an example of counting using continuous bars, which are divided into sections for each day of the week.

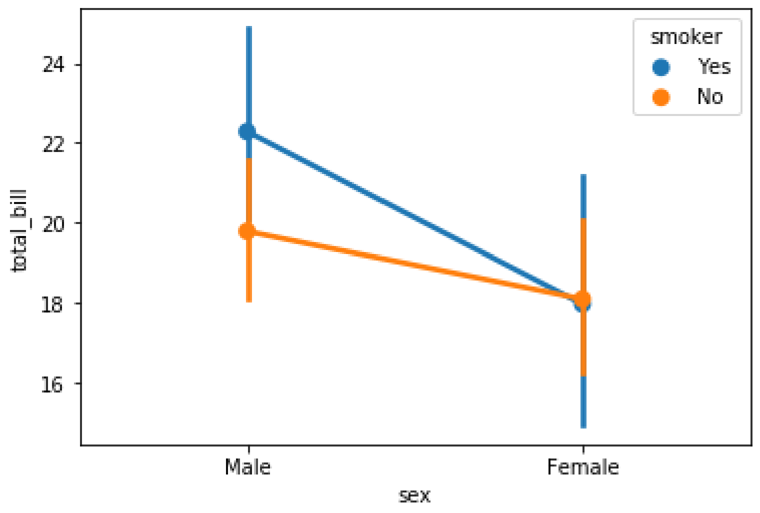
```
# countplot
f, ax = plt.subplots(figsize = (7, 3))
                    sns.countplot(y = "day",
                    data = tips,
                    color = "c");
```



Point plot

Point plot is another type of graph that can be used for representing categorical variables by showing the relationship between gender and the total count.

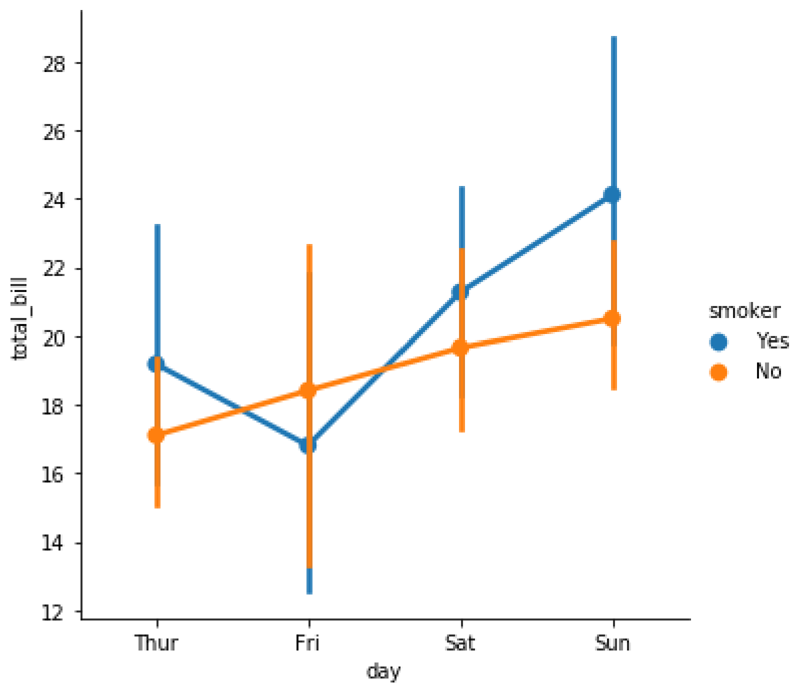
```
# pointplot
sns.pointplot(x = "sex",
             y = "total_bill",
             hue = "smoker",
             data = tips);
```



Factorplot

Factorplot is a variant of the point plot that features customization options to represent categorical data.

```
# factorplot
sns.factorplot(x = "day",
               y = "total_bill",
               hue = "smoker",
               data = tips);
```



In conclusion, Seaborn library has proven to be a valuable tool for data visualization in today's world. The variety of charts it offers, from basic ones such as bar plots and countplots, to more complex ones like violin plots and factorplots, has made it an essential tool for data analysis.

The ability to customize and tailor these charts to the specific needs of the data and user make Seaborn an ideal tool that should be in every data analyst's toolkit.

I hope you have found this useful. Thank you for reading. 🐼

Leonardo Anello
in/anello92

Data · Data Science · Statistics · Python · Data Visualization

Follow

Written by 🐼 panData