```
# Importing Library
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```
#the restaurant tips data available in seaborn
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 steps: Generate code with `tips` View recommended plots

Swarm Plot

Swarm Plot typically has a categorical variable on the x-axis and a numerical variable on the y-axis, and it displays individual data points along each category.

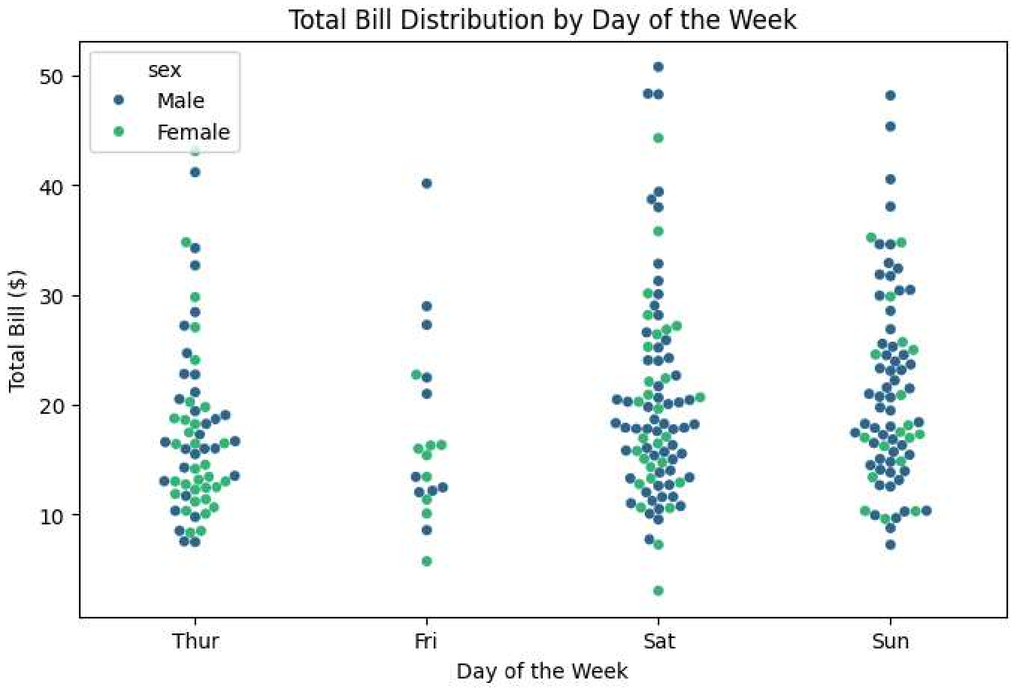
This gives us a visualization of the distribution and density of data points within categories.

One of the main features of a Swarm Plot is that it minimizes overlap between data points.

This means that each data point is plotted in such a way that it does not overlap with other points in the same category. This makes it easier to see the density of data. But it’s difficult to visualize with larger data, so it’s effective with relatively small datasets.

t a swarm plot is useful in understanding the density of data points as they provide a clear representation of where data points are concentrated and where they are sparse.

```
# Swarm Plot
plt.figure(figsize=(8, 5))
sns.swarmplot(x="day", y="total_bill", data=tips, hue="sex", palette="viridis")
plt.title("Total Bill Distribution by Day of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill ($)")
plt.show()
```



Point Plot

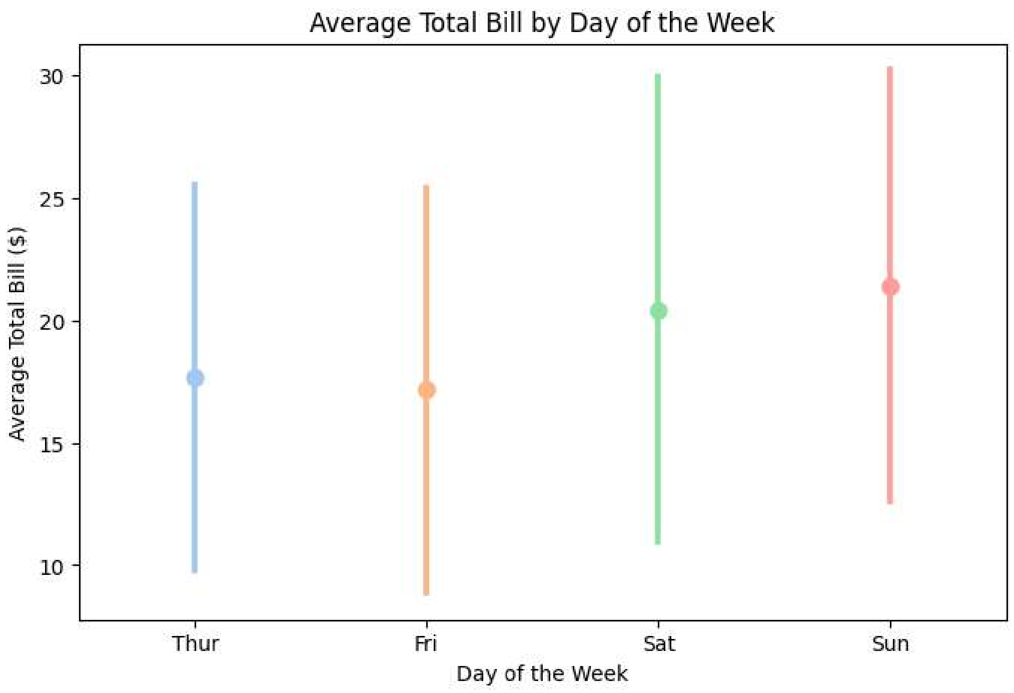
The Point Plot typically has a categorical variable on the x-axis and a numerical variable on the y-axis and can provide insights into the distribution and central tendency of data within each category, such as a line or a point.

```
# Point Plot
plt.figure(figsize=(8, 5))
sns.pointplot(x="day", y="total_bill", data=tips, errorbar='sd', palette='pastel')
plt.title("Average Total Bill by Day of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Average Total Bill ($)")
plt.show()
```

```
<ipython-input-7-68c5859c90c4>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14

  sns.pointplot(x="day", y="total_bill", data=tips, errorbar='sd', palette='pastel')
```



A Cat Plot

It is short for "Categorical Plot," is a highly versatile plotting function available in Seaborn. It consolidates various categorical plots into one by utilizing a single parameter called "kind" in the command `sns.catplot(x, y)`.

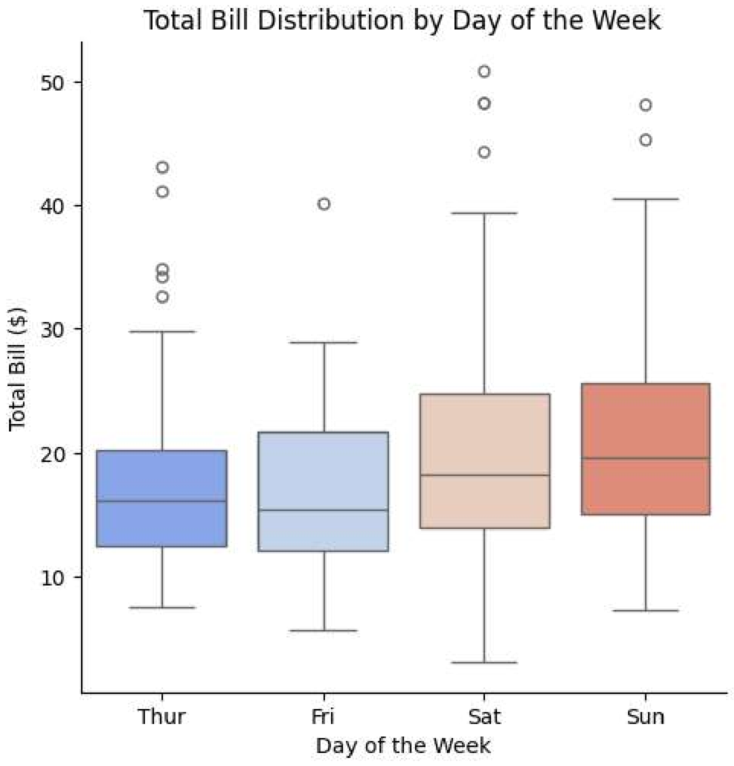
With this parameter, you can easily generate different types of categorical plots such as bar plots, swarm plots, box plots, violin plots, count plots, and point plots, among others.

```
# Box Plot using cat plot
plt.figure(figsize=(8, 5))
sns.catplot(x="day", y="total_bill", data=tips, kind="box", palette="coolwarm")
plt.title("Total Bill Distribution by Day of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill ($)")
plt.show()
```

```
<ipython-input-8-2bd8f3eff8fd>:3: FutureWarning: 

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14

  sns.catplot(x="day", y="total_bill", data=tips, kind="box", palette="coolwarm")
<Figure size 800x500 with 0 Axes>
```



Univariate Plots

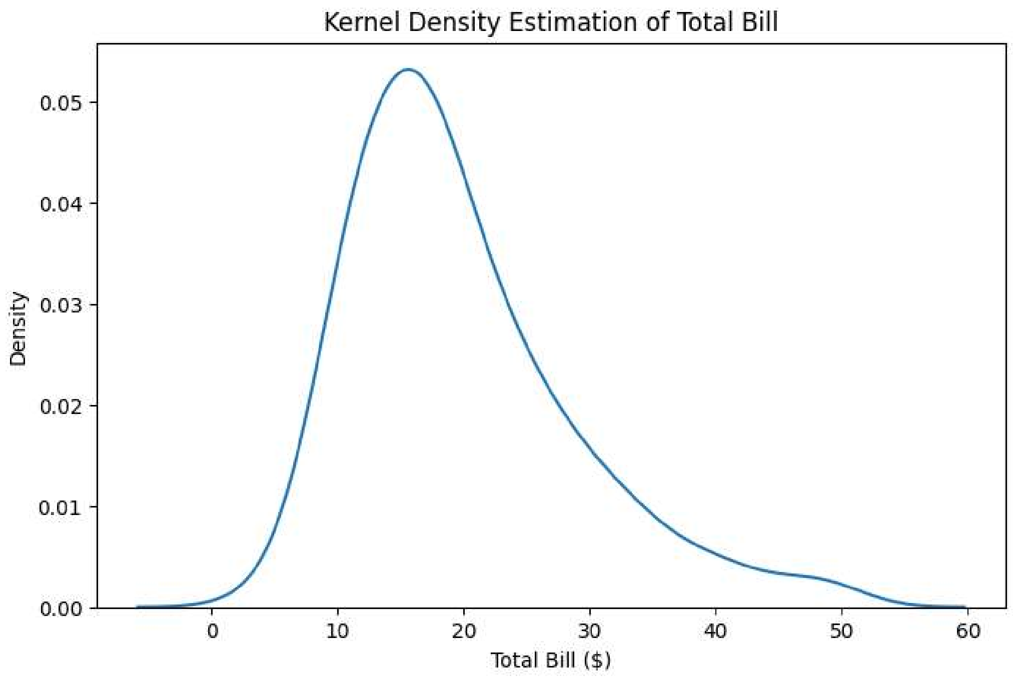
Univariate plots are used to visualize and analyze a single variable at a time. They are useful for understanding the distribution, and central tendency, identifying outliers, and checking for patterns or trends within a single variable.

KDE Plot

A KDE (Kernel Density Estimation) Plot is a visualization tool employed to estimate the probability density function of a continuous variable. In this plot, the x-axis typically represents the continuous variable, while the y-axis depicts the estimated probability density. The area under the curve of the plot sums to 1, signifying that the data distribution is normalized.

KDE Plots provide insights into the shape, peaks, modes, and spread of the data, and these are valuable for estimating the probability density of data points, which can be helpful in statistical analysis and hypothesis testing.

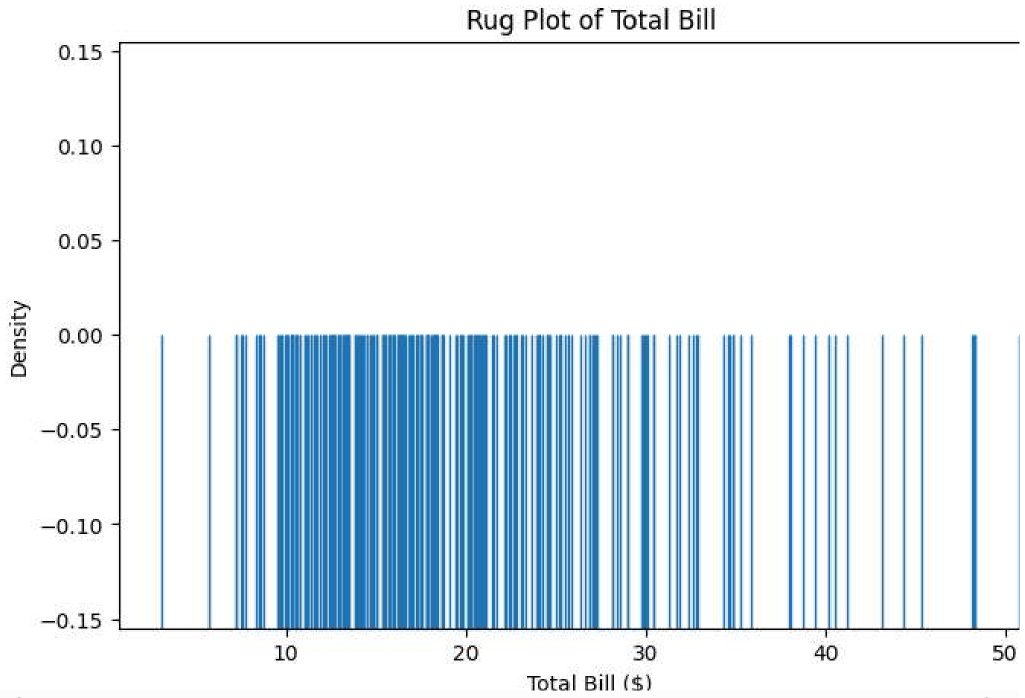
```
# KDE Plot
plt.figure(figsize=(8, 5))
sns.kdeplot(data=tips['total_bill'])
plt.title("Kernel Density Estimation of Total Bill")
plt.xlabel("Total Bill ($)")
plt.ylabel("Density")
plt.savefig('kde_plot.png')
plt.show()
```



Rug Plot

A Rug Plot is comprised of vertical lines (or "ticks") placed along a single axis, often the x-axis. Each tick symbolizes the position of an individual data point, and the proximity of the ticks indicates the density of the data points; areas with more ticks closely clustered together generally represent denser regions. To generate a Rug Plot, you can use `sns.rugplot(data=dataframe[column])`. Furthermore, you have the option to customize the appearance of Rug Plots by modifying parameters like the height, color, and orientation of the ticks.

```
#  Rug Plot
plt.figure(figsize=(8, 5))
sns.rugplot(data=tips['total_bill'], height=0.5)
plt.title("Rug Plot of Total Bill")
plt.xlabel("Total Bill ($)")
plt.ylabel("Density")
plt.savefig('rug_plot.png')
plt.show()
```



From KDE, we got the insight that the Bill Amount peak is somewhere between 10–20, but through the Rug plot, we get an idea that the density is more at 15–20.

Hence, This plot is particularly useful when you want to see the exact position of individual data points. Rug Plots are often combined with histograms, KDE plots, or box plots to provide additional context and detail. They can help highlight outliers or areas of high data density.

Box Plot

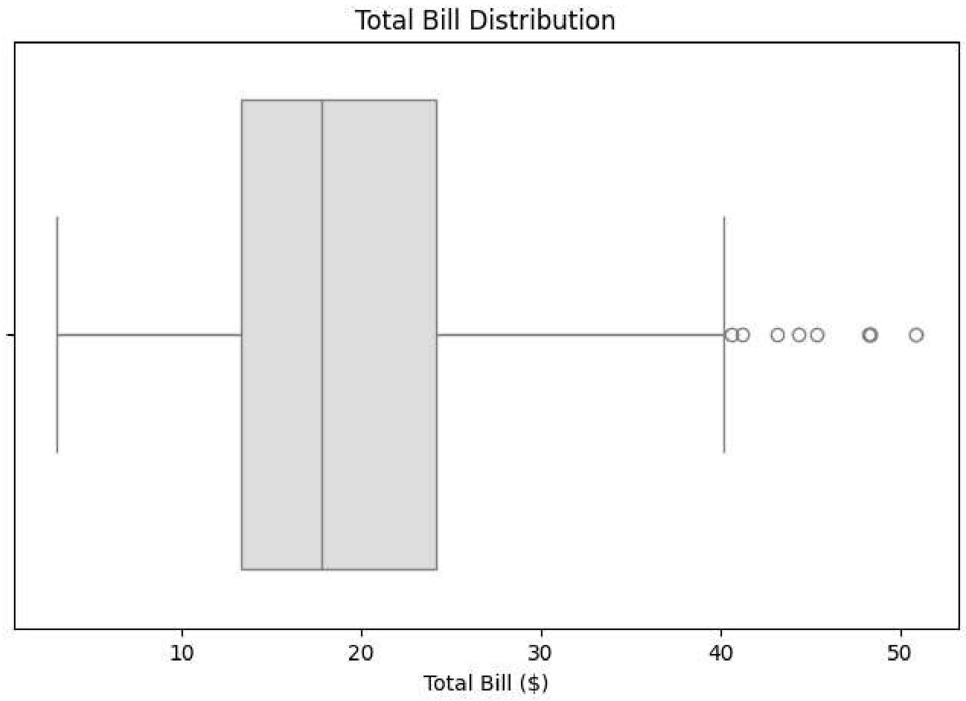
One can only pass one variable when you want to do univariate analysis.

```
# Box Plot for Univariate Visualization
plt.figure(figsize=(8, 5))
sns.boxplot(x=tips['total_bill'], palette="coolwarm")
plt.title("Total Bill Distribution")
plt.xlabel("Total Bill ($)")
plt.savefig('univariate_box_plot.png')
plt.show()
```

```
<ipython-input-11-6fba4010671e>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14

  sns.boxplot(x=tips['total_bill'], palette="coolwarm")
```



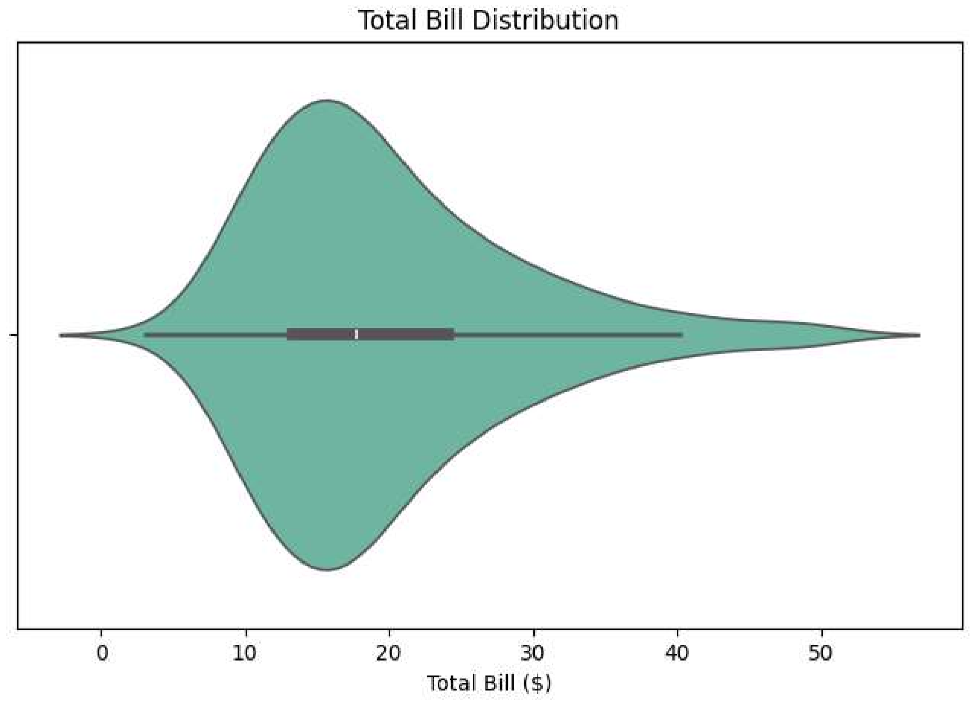
Violin Plot

```
# Violin Plot for Univariate Visualization
plt.figure(figsize=(8, 5))
sns.violinplot(x=tips['total_bill'], palette="Set2")
plt.title("Total Bill Distribution")
plt.xlabel("Total Bill ($)")
plt.show()
```

```
<ipython-input-12-1bbb36b6527d>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14

  sns.violinplot(x=tips['total_bill'], palette="Set2")
```

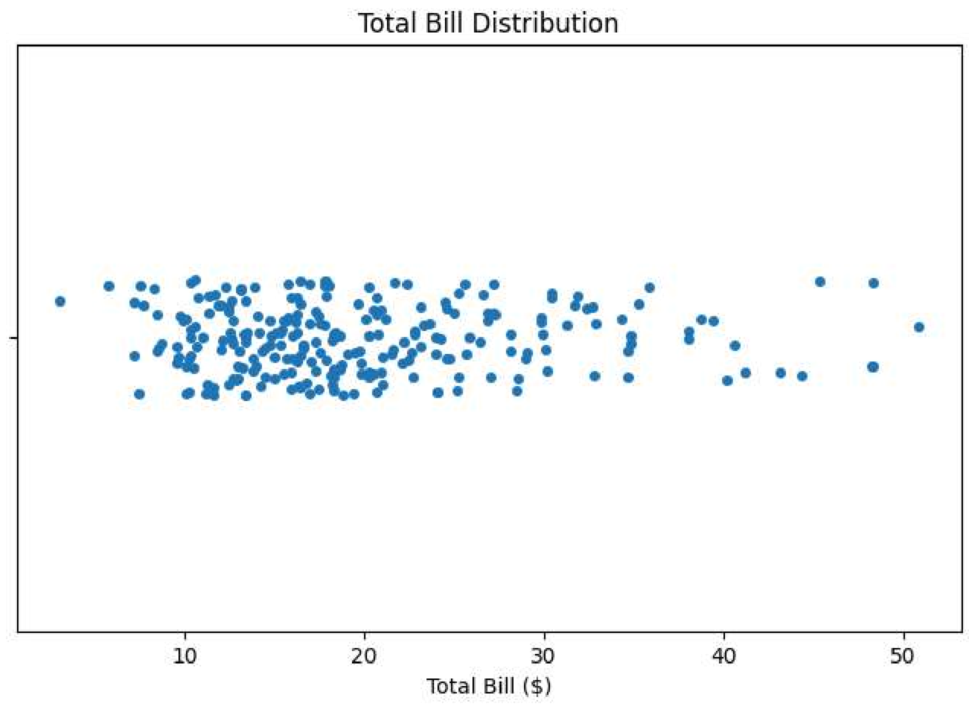


Strip Plot

A Strip Plot resembles a Swarm Plot as it exhibits individual data points along a single axis. Nonetheless, in a Strip Plot, these data points may overlap, making it more suitable for visualizing smaller datasets compared to Swarm Plots, which are better at avoiding overlap.

To create a Strip Plot, utilize `sns.stripplot(data=dataframe[column])`, which can also be used for bivariate data representation. Additionally, applying jittering, which involves introducing a small amount of random noise to the data points, can help diminish overlap and enhance clarity.

```
#  Strip Plot for Univariate Visualization
plt.figure(figsize=(8, 5))
sns.stripplot(x=tips['total_bill'], jitter=True)
plt.title("Total Bill Distribution")
plt.xlabel("Total Bill ($)")
plt.savefig('strip_plot.png')
plt.show()
```



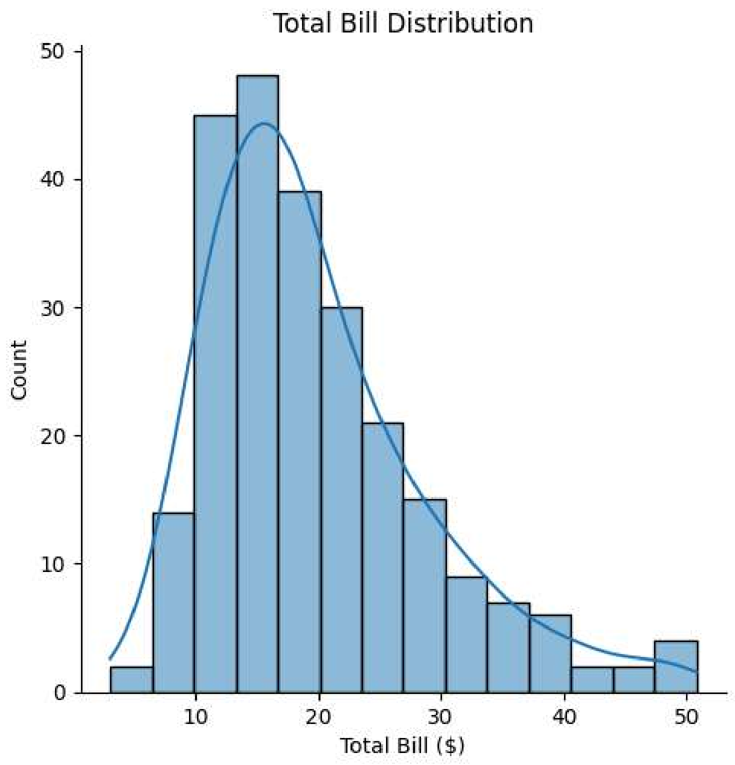
Dist Plot

A Dist Plot, abbreviated for Distribution plot, usually resembles a histogram. However, it also incorporates a smoothed curve known as a KDE plot. This plot segments the numerical variable's range into bins or intervals and illustrates the frequency or density of data points within each bin.

Optionally, a rug plot can be added along the x-axis, showing individual data points as small vertical lines. This provides insight into the density and distribution of individual data points.

```
# Strip Plot for Univariate Visualization
plt.figure(figsize=(8, 5))
sns.displot(data=tips['total_bill'], kde=True) # Using displot with kernel
plt.title("Total Bill Distribution")
plt.xlabel("Total Bill ($)")
plt.savefig('distribution_plot.png')
plt.show()
```

```
<Figure size 800x500 with 0 Axes>
```



Bi-Variate Plots

Bivariate plots involve the visualization and analysis of the relationship between two variables simultaneously. They are used to explore how two variables are related or correlated.

Regression Plot

A Regression Plot examines the correlation between two numerical variables: the independent variable (typically on the x-axis) and the dependent variable (on the y-axis).

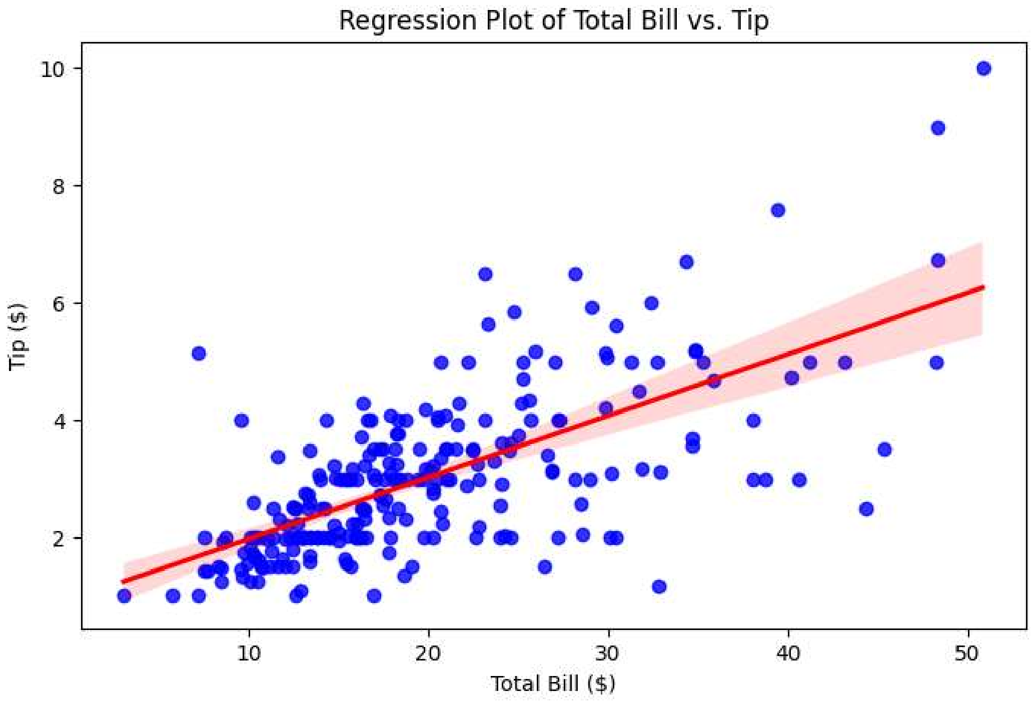
It showcases individual data points as dots, with the main feature being the regression line or curve. This line or curve represents the most accurate mathematical model depicting the relationship between the variables.

Use sns.regplot(x,y,data) to create a regression plot.

The regression line represents the best-fitting linear model for predicting tips based on total bill amounts.

The scatter points show individual data points, and one can observe how they cluster around the regression line. This plot is useful for understanding the linear relationship between these two variables.

```
# Regression Plot
plt.figure(figsize=(8, 5))
sns.regplot(x="total_bill", y="tip", data=tips, scatter_kws={"color": "blue"}, line_kws={"color": "red"})
plt.title("Regression Plot of Total Bill vs. Tip")
plt.xlabel("Total Bill ($)")
plt.ylabel("Tip ($)")
plt.savefig('regression_plot.png')
plt.show()
```



Joint Plot

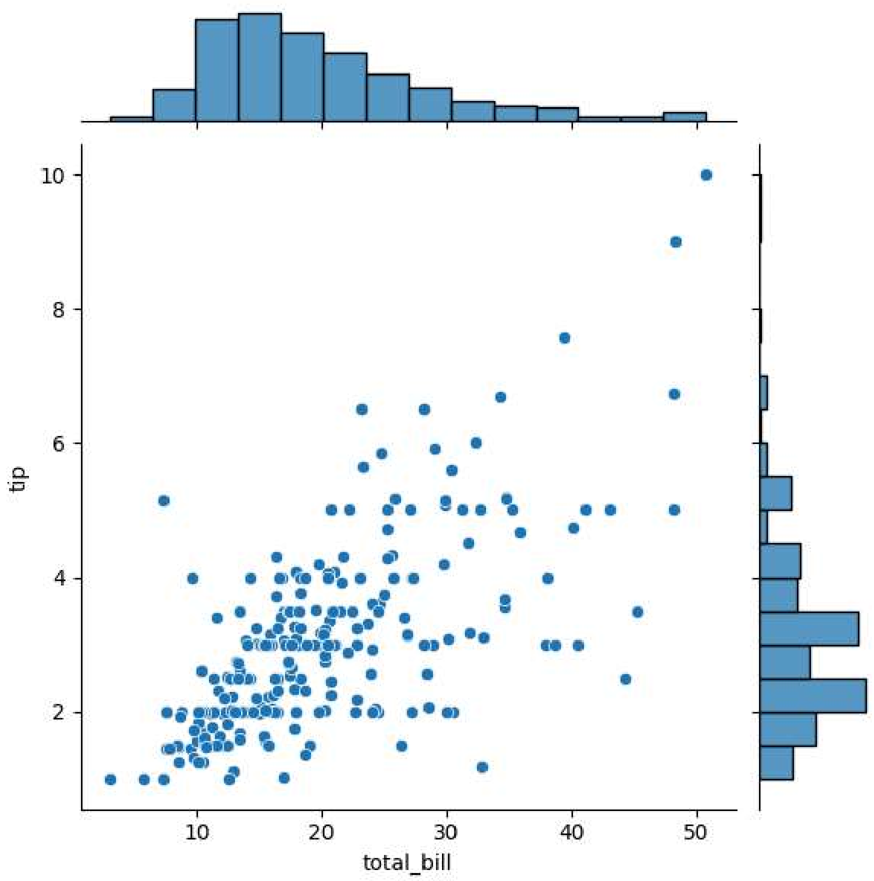
A joint plot combines scatter plots, histograms, and density plots to visualize the relationship between two numerical variables.

The central element of a Joint Plot is a Scatter Plot that displays the data points of the two variables against each other, along the x-axis and y-axis of the Scatter Plot, there are histograms or Kernel Density Estimation (KDE) plots for each individual variable.

These marginal plots show the distribution of each variable separately. Use sns.jointplot(x,y,data=dataframe,kind) , kind can be one of ['scatter', 'hist', 'hex', 'kde', 'reg', 'resid'] these.

This plot shows the relation between the two variables through a scatter plot, while the marginal histograms show the distribution of each variable separately.

```
# Joint Plot
sns.jointplot(x="total_bill", y="tip", data=tips, kind="scatter")
plt.savefig('joint_plot.png')
plt.show()
```



Hexbin Plot

A Hexbin plot, also known as a Hexagonal Binning plot, organizes data points into hexagonal bins, enhancing the visualization of data density and patterns.

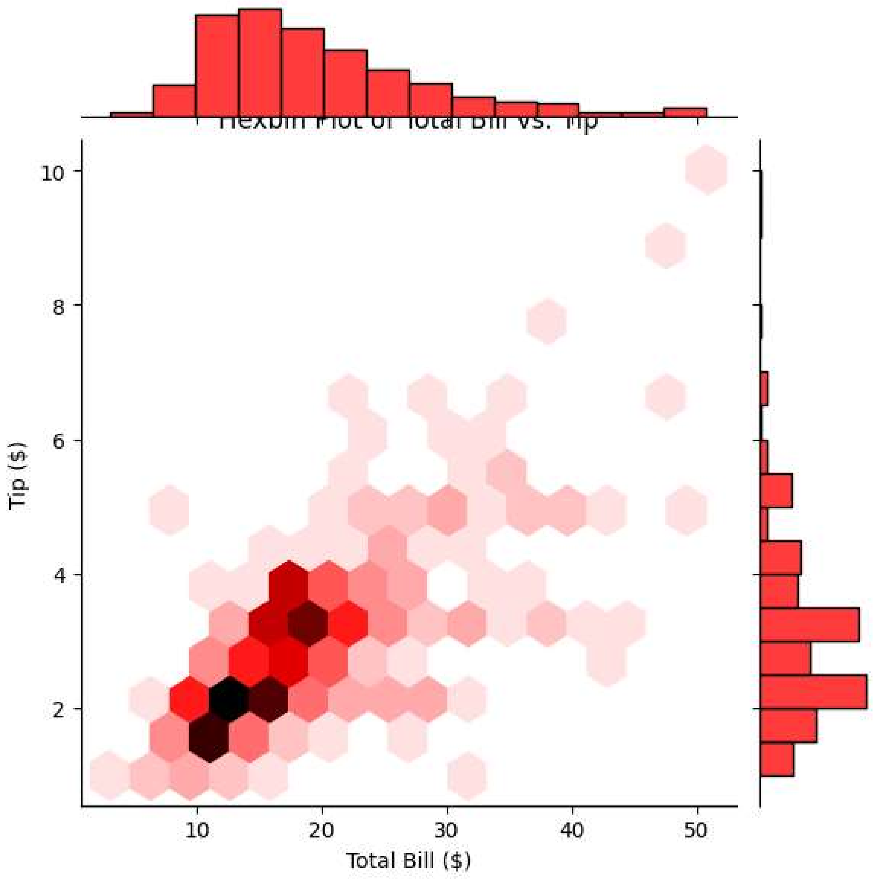
They prove particularly useful with extensive datasets, mitigating the clutter and confusion that can arise with scatter plots containing individual points.

To generate a Hexbin plot, you can utilize the kind parameter in the joint plot function. Additionally, you have the flexibility to customize various aspects of the plot, including the color map, grid size, and other parameters, to refine the Hexbin Plot's appearance.

the Hexbin plot gives much more clarity than the scatter plot for a large dataset. Each hexagon in the plot is color-coded to indicate the density of data points within that bin.

```
# Hexbin Plot
plt.figure(figsize=(8, 5))
sns.jointplot(x="total_bill", y="tip", kind='hex', data=tips, gridsize=15, color='red')
plt.title("Hexbin Plot of Total Bill vs. Tip", loc='center')
plt.xlabel("Total Bill ($)")
plt.ylabel("Tip ($)")
plt.savefig('hexbin_plot.png')
plt.show()
```

```
<Figure size 800x500 with 0 Axes>
```



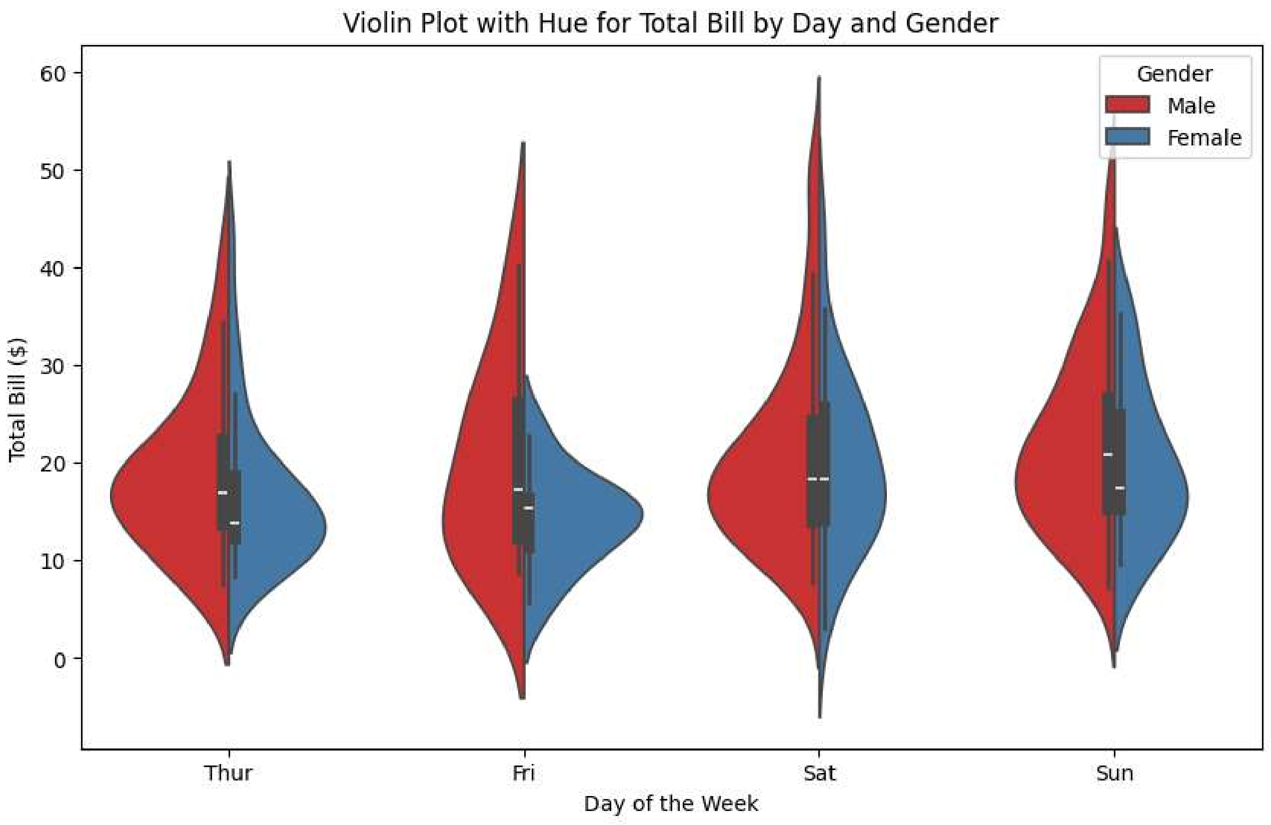
MultiVariate Plots

These are my favorite, these plots give us a lot of flexibility to explore the relationships and patterns among three or more variables simultaneously. That is, Multivariate plots extend the analysis to more than two variables, which will be often needed in the Data Analysis.

Using Hue Parameter

Using the hue parameter will add color to the plot based on the provided categorical variable, specifying a unique color for each of the categories. This parameter can be used almost all of the plots like .scatterplot() , .boxplot() , .violinplot() , .lineplot() , etc

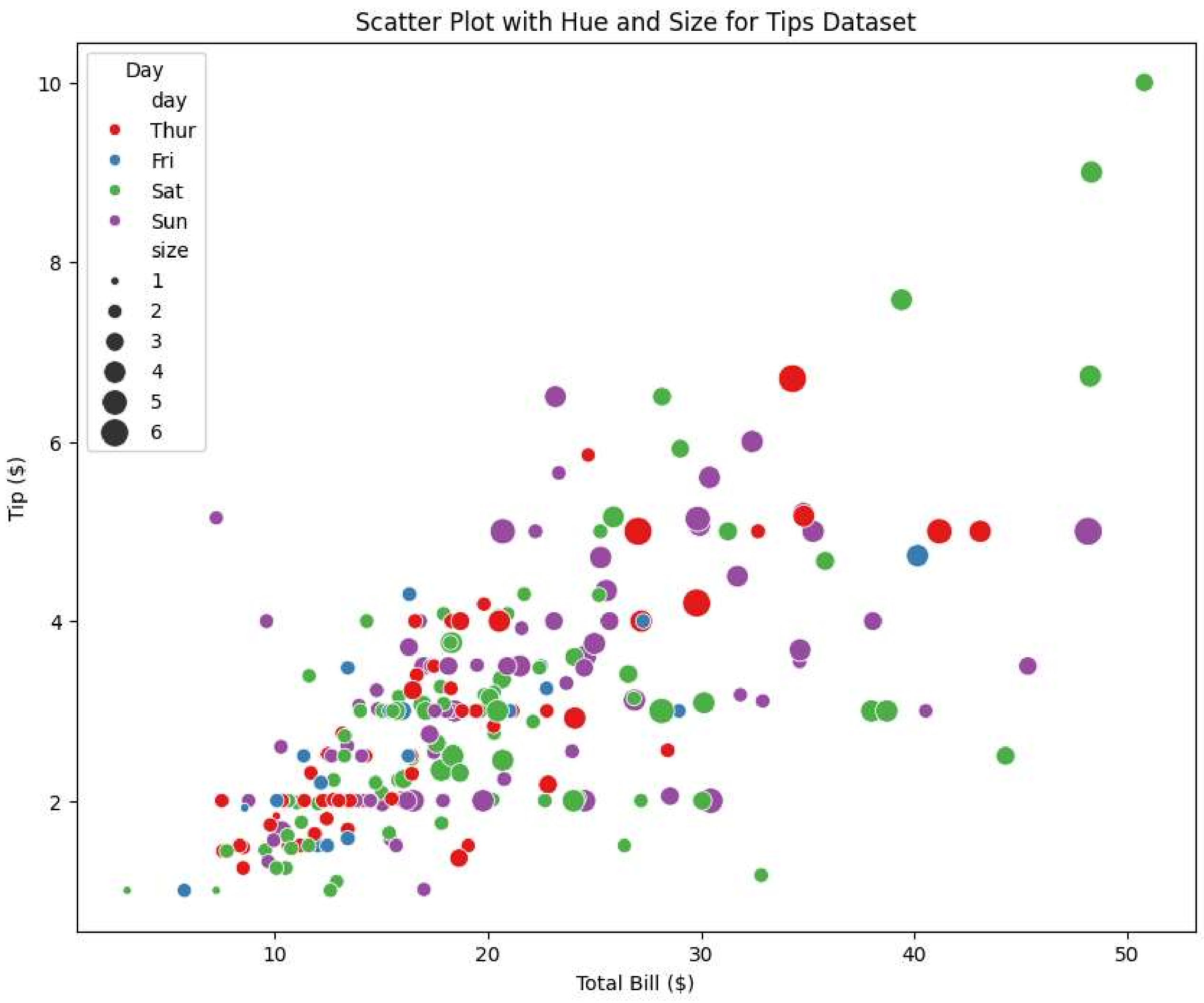
```
# Violin Plot with Hue
# use the sex column to color the violins based on gender, adding one dimension to
# the plot. Each gender is represented by a different color.
plt.figure(figsize=(10, 6))
sns.violinplot(
 x="day", # x-axis: Days of the week (categorical)
 y="total_bill", # y-axis: Total bill amount (numerical)
 data=tips,
 hue="sex", # Color by gender (categorical)
 palette="Set1", # Color palette
 split=True) # Split violins by hue categories
plt.title("Violin Plot with Hue for Total Bill by Day and Gender")
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill ($)")
plt.legend(title="Gender")
plt.show()
```



Using hue and size parameters

size is another parameter that can be used to add another numeric variable dimension.

```
"""Each day is represented by a different color with the hue parameter adding a third
dimension, and marker size based on the size column in the dataset to add a fourth
dimension.
"""
import seaborn as sns
import matplotlib.pyplot as plt
# Load the "tips" dataset
tips = sns.load_dataset("tips")
# Scatter Plot with Hue and Size
plt.figure(figsize=(10, 8))
sns.scatterplot(
 x="total_bill",
 y="tip",
 data=tips,
 hue="day", # Color by day (categorical)
 size="size", # Vary marker size by size column (numerical)
 sizes=(20, 200), # Define the size range for markers
 palette="Set1" # Color palette
)
plt.title("Scatter Plot with Hue and Size for Tips Dataset")
plt.xlabel("Total Bill ($)")
plt.ylabel("Tip ($)")
plt.legend(title="Day")
plt.savefig('scatterplot_with_hue_and_size.png')
plt.show()
```

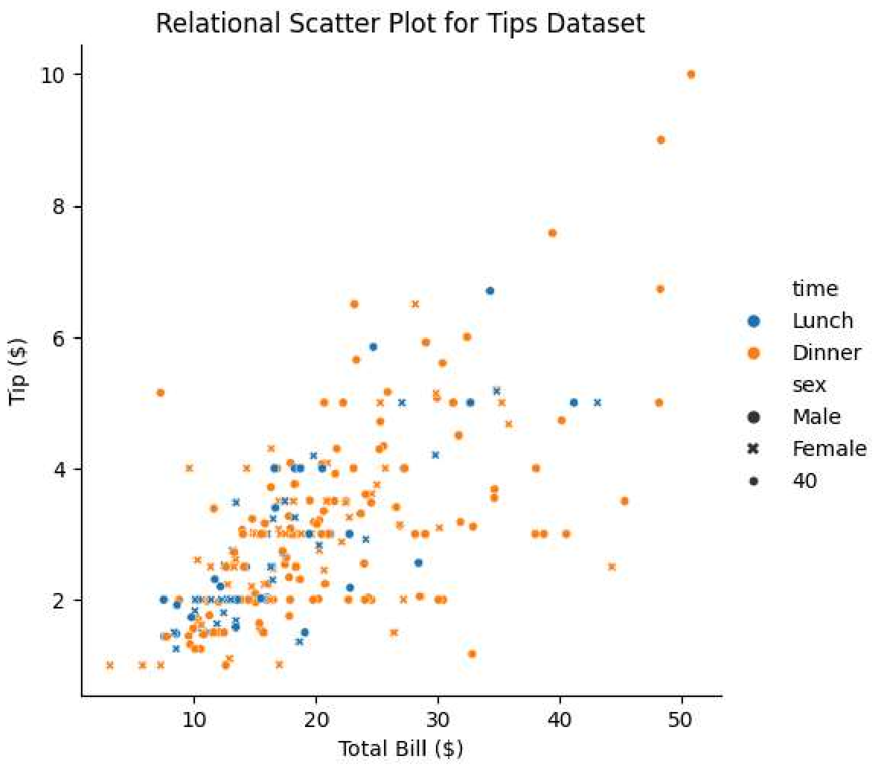


Relational Plot

A Relational Plot enables one to visualize the connection between two numerical variables, alongside other categorical or numerical dimensions. You can employ `sns.relplot(x, y, data, hue, size, style)` to create such plots.

The `hue` parameter allows coloring data points based on a categorical variable, `size` permits adjusting marker size according to a numerical variable, and `style` facilitates distinguishing markers or lines using a categorical variable. Furthermore, one can specify the type of relational plot one desire using the `kind` parameter.

```
# Create a scatterplot using a Relational Plot (relplot)
sns.relplot(x="total_bill", y="tip", data=tips, hue="time", style="sex", size=40)
plt.title("Relational Scatter Plot for Tips Dataset")
plt.xlabel("Total Bill ($)")
plt.ylabel("Tip ($)")
plt.savefig('relplot_with_mv.png')
plt.show()
```



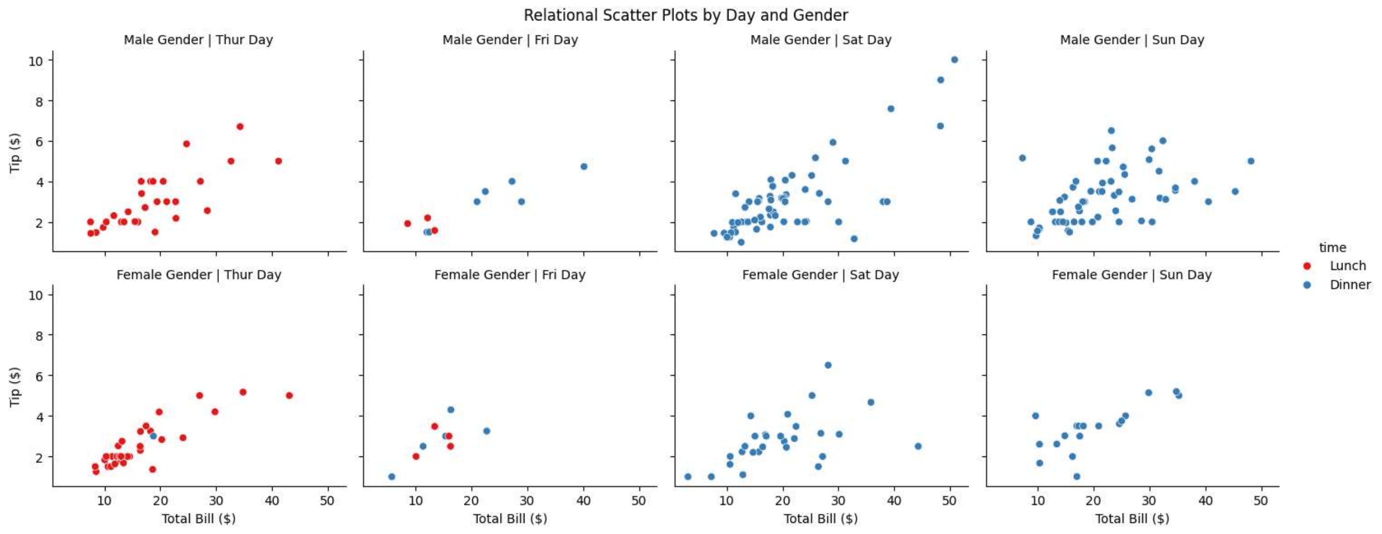
FacetGrid by using col and row parameters of Relplot

A Facet Grid is a feature in Seaborn that allows one to create a grid of subplots, each representing a different subset of his/her data.

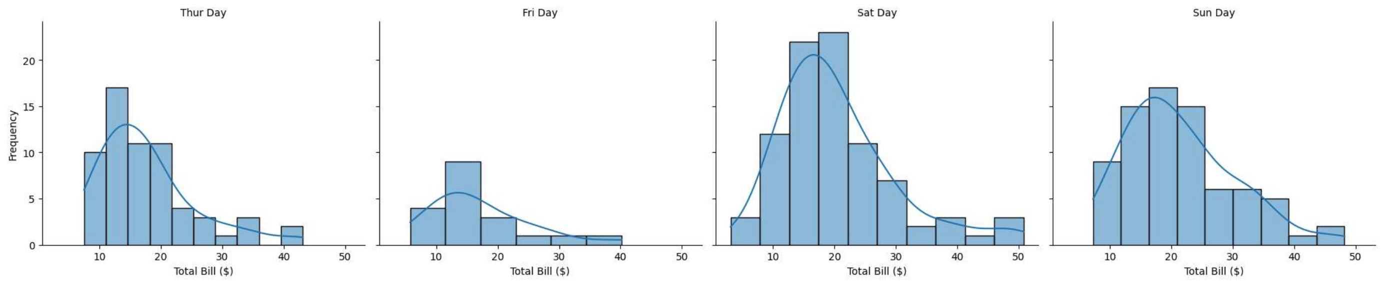
In this way, Facet Grids are used to compare patterns or relationships with multiple variables within different categories. Use sns.

FacetGrid(data,col,row) to create a facet grid, which returns the grid object. After creating the grid object one need to map it to any plot of his/her choice.

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load the "tips" dataset
tips = sns.load_dataset("tips")
# Create a facet grid using a Relational Plot (relplot)
g = sns.relplot(
 x="total_bill",
 y="tip",
 data=tips,
 hue="time",
 col="day", # Separate plots by day (columns)
 row="sex", # Separate plots by gender (rows)
 palette="Set1",
 height=3, # Height of each subplot
 aspect=1.2 # Aspect ratio of each subplot
)
# Set titles and labels for the facets
g.set_titles(col_template="{col_name} Day", row_template="{row_name} Gender")
g.set_axis_labels("Total Bill ($)", "Tip ($)")
plt.suptitle("Relational Scatter Plots by Day and Gender")
plt.subplots_adjust(top=0.9) # Adjust the title position
plt.savefig('relplot_col_row.png')
plt.show()
```



```
# Create a Facet Grid of histograms for different days
g = sns.FacetGrid(tips, col="day", height=4, aspect=1.2)
g.map(sns.histplot, "total_bill", kde=True)
g.set_axis_labels("Total Bill ($)", "Frequency")
g.set_titles(col_template="{col_name} Day")
plt.savefig('facet_grid_hist_plot.png')
plt.show()
```



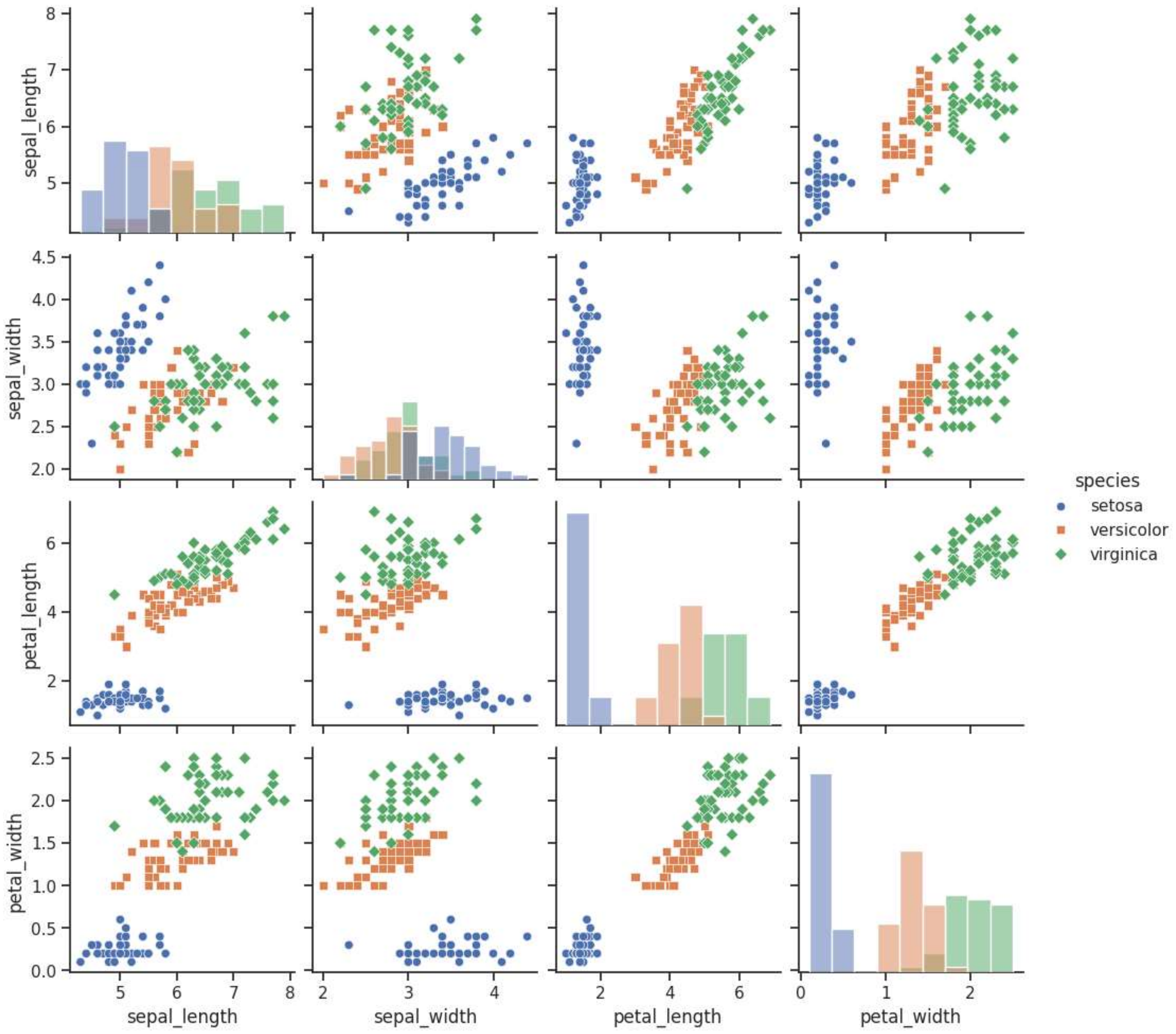
Pair Plot

A Pair plot offers a grid of scatterplots and histograms, with each plot illustrating the relationship between two variables.

This type of plot is also referred to as a Pairwise Plot or Scatterplot Matrix. The diagonal cells usually depict histograms or kernel density plots to visualize the distributions of individual variables. On the other hand, the off-diagonal cells in the grid typically show scatterplots, demonstrating the relationship between two variables.

Pair Plots are valuable for analyzing patterns, correlations, and distributions across multiple dimensions in your dataset.

```
# Load the "iris" dataset
iris = sns.load_dataset("iris")
# Pair Plot
sns.set(style="ticks")
sns.pairplot(iris,diag_kind='hist', hue="species", markers=["o", "s", "D"])
plt.savefig('pair_plot.png')
plt.show()
```



Pair Grid

By using a pair grid you can customize the lower, upper, and diagonal plots individually.