Seaborn by Mrittika Megaraj

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In [2]:
```
# Use sns.get_dataset_names() see the available datasets in seaborn
# Let's use the restaurant tips data available in seaborn
tips = sns.load_dataset('tips')
```

In [3]:
```
tips.head()
```

Out[3]:

	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

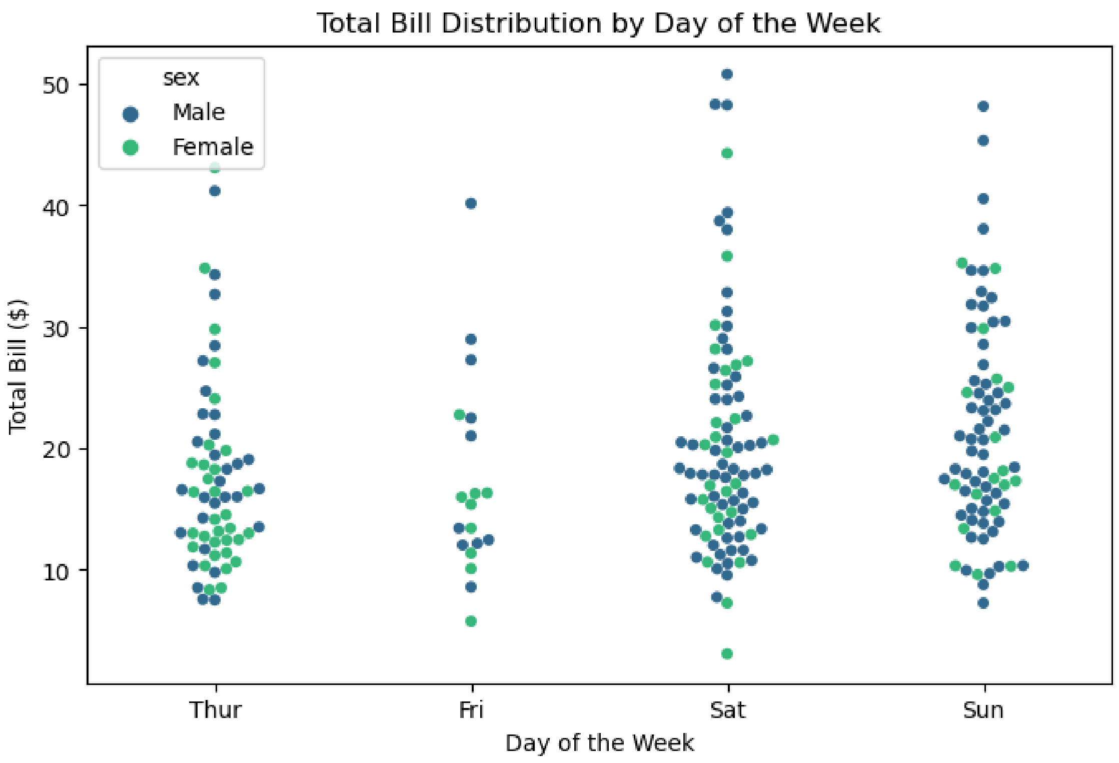
Swarm Plot

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. Now, As you can guess 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.

Use sns.swarmplot(x,y,data) to make the swarm plot and use palette to change the colors.

In [4]:
```
# Swarm Plot
plt.figure(figsize=(8, 5))
sns.swarmplot(x="day", y="total_bill", data=tips, hue="sex", palette="viric
plt.title("Total Bill Distribution by Day of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill ($)")
plt.show()
```

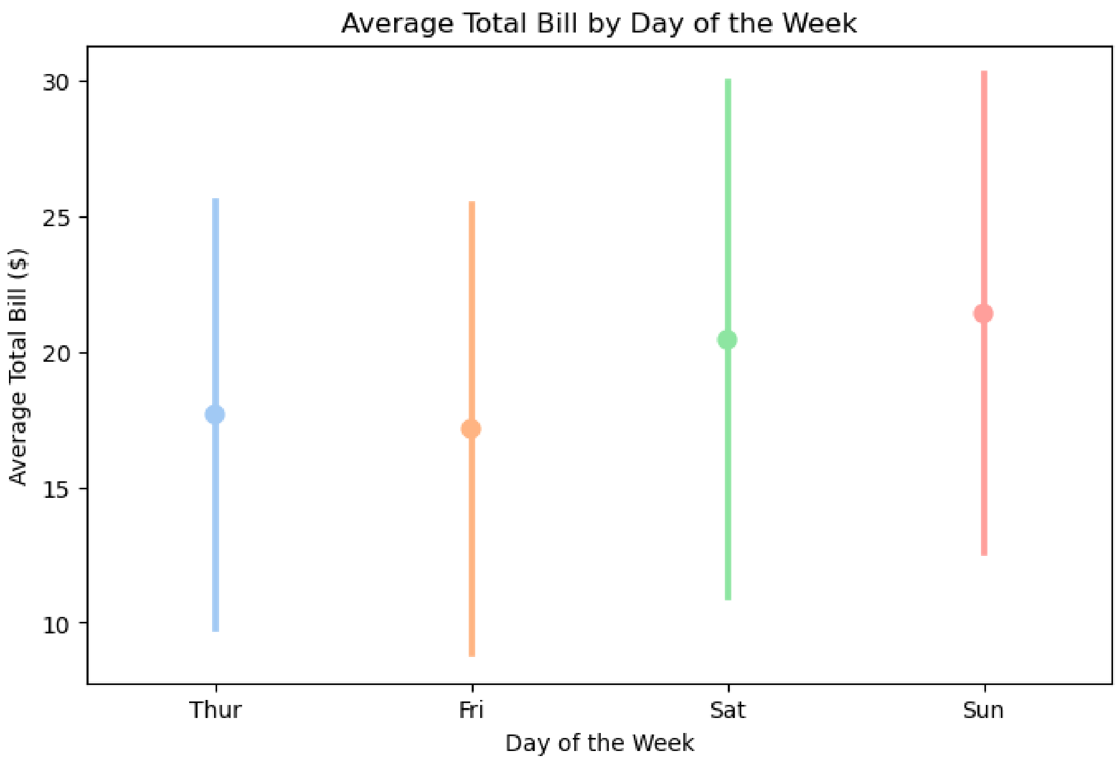


As you can see, They can help identify outliers or unusual data points within a category. so we can positively say that a swarm plot is useful in understanding the density of data points as they provide a clear representation of where data points are concentrated and where they are sparse.

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.

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



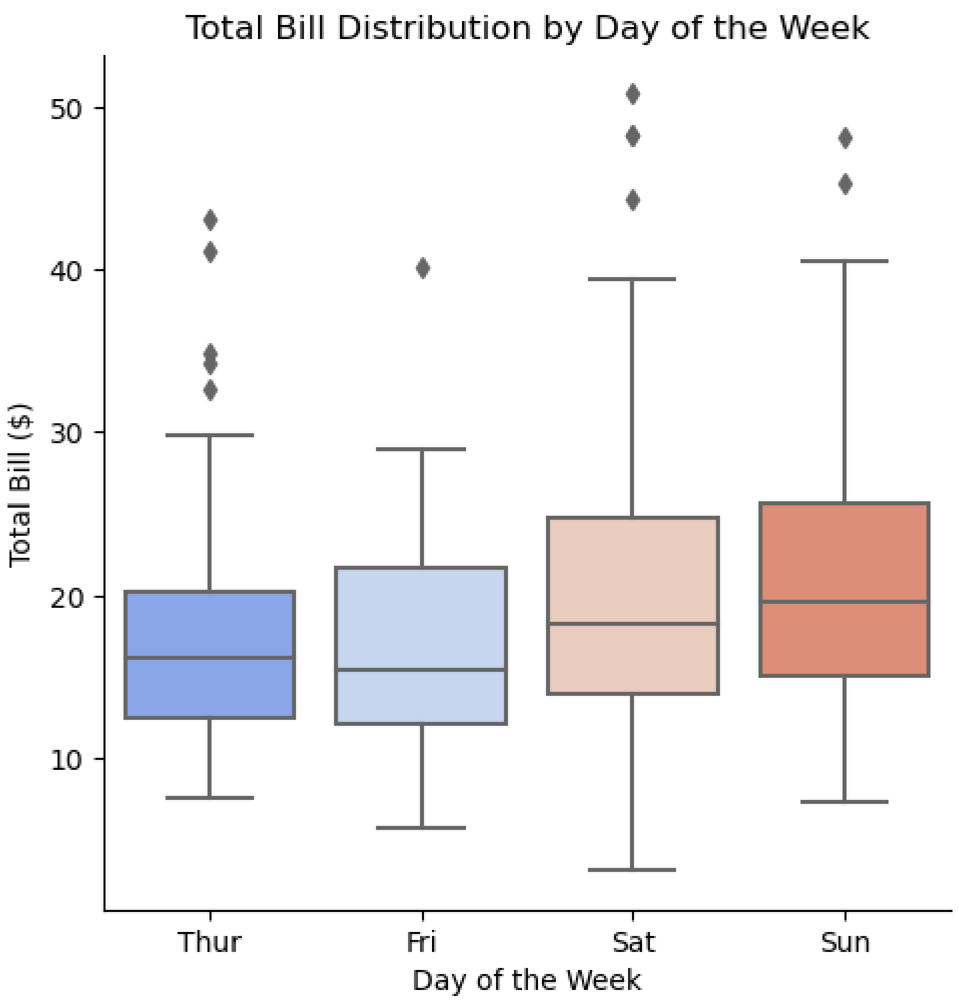
- This line or point represents the average or median value of the numerical variable for each category. So by using this, you can quickly see whether there are differences in the average or median values across categories.
- The same above use cases would also apply here, but just that we will observe the central tendency through this plot. This gives us idea of what is average, low, and high scores in each class.

Cat Plot

A Cat Plot, short for “Categorical Plot,” is a very powerful plotting function in Seaborn. It allows you to all the categorical plots we have seen above with just one parameter called kind in the command sns.catplot(x,y).You can usekind=bar, swarm , box, violin , count , point , etc.

```
In [6]: # Box Plot using cat plot
        plt.figure(figsize=(8, 5))
        sns.catplot(x="day", y="total_bill", data=tips, kind="box", palette="coolwa
        plt.title("Total Bill Distribution by Day of the Week")
        plt.xlabel("Day of the Week")
        plt.ylabel("Total Bill ($)")
        plt.show()
```

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



Univariate Plots

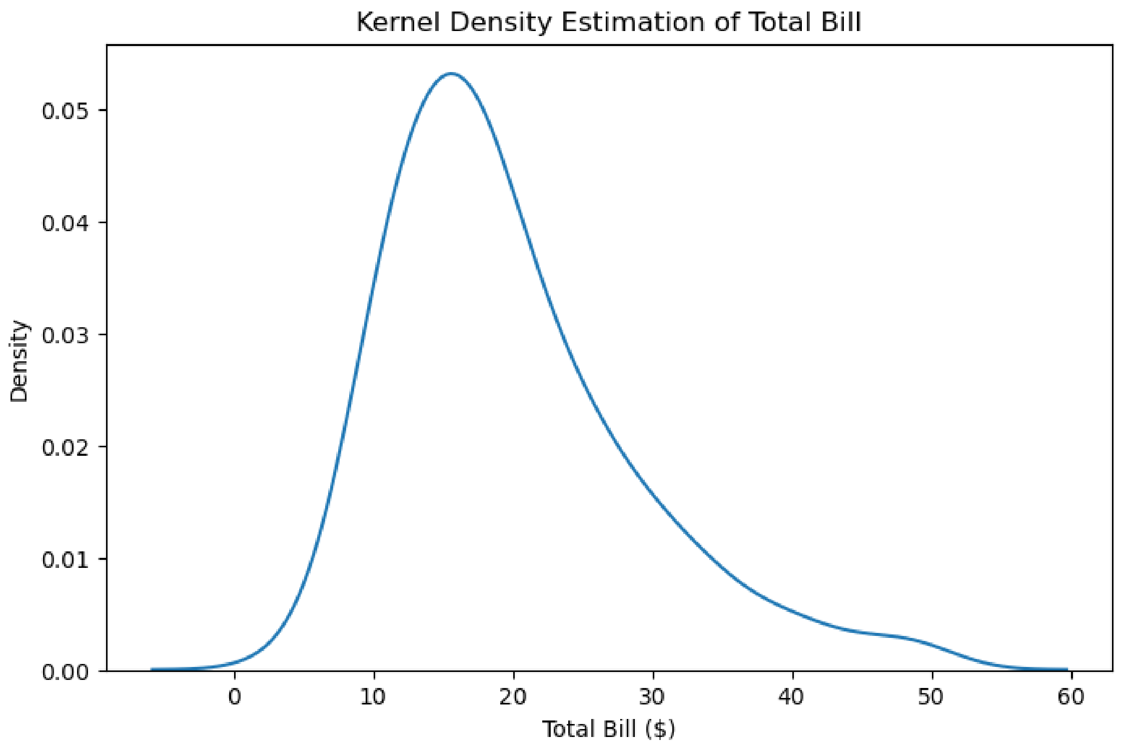
Univariate plots are used to visualize and analyze a single variable at a time. They are useful for understanding the distribution, and central tendency, identifying outliers, and checking for patterns or trends within a single variable.

KDE Plot

A KDE (Kernel Density Estimation) Plot is a type of data visualization that is used to estimate the probability density function of a continuous variable. So, we will have our continuous variable on the x-axis and the y-axis represents the estimated probability density area under the curve sums to 1, indicating that the data distribution is normalized.

Use sns.kdeplot(data=dataframe[column]) to create a KDE plot for a continuous variable

```
In [7]: # 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()
```



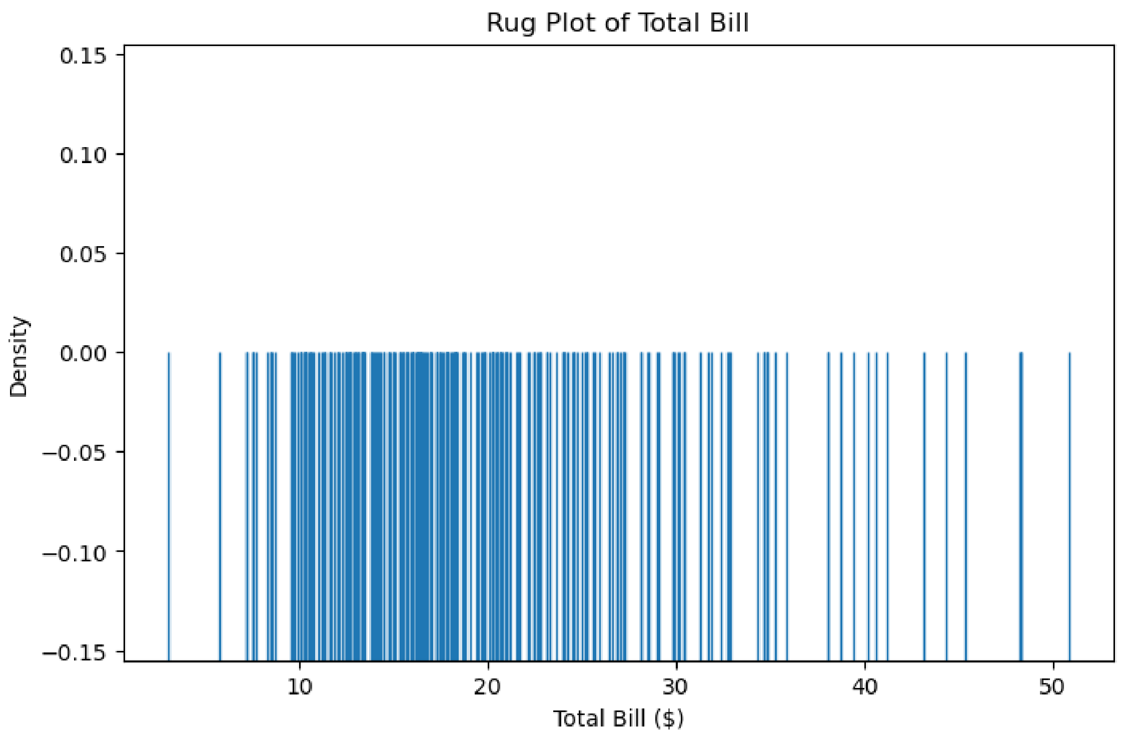
Hence, KDE Plots provide insights into the shape, peaks, modes, and spread of the data, and these are valuable for estimating the probability density of data points, which can be helpful in statistical analysis and hypothesis testing.

Rug Plot

A Rug Plot consists of vertical lines (or “ticks”) positioned along a single axis (usually the x-axis). Each tick represents the location of an individual data point, and the closeness of the ticks represents the density of the data points; typically denser areas have more ticks closely packed together.

Use sns.rugplot(data=dataframe[column]) to create a rug plot. You can customize the appearance of Rug Plots by adjusting parameters such as the height, color, and orientation of the ticks.

In [8]:
```python
# 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()
```

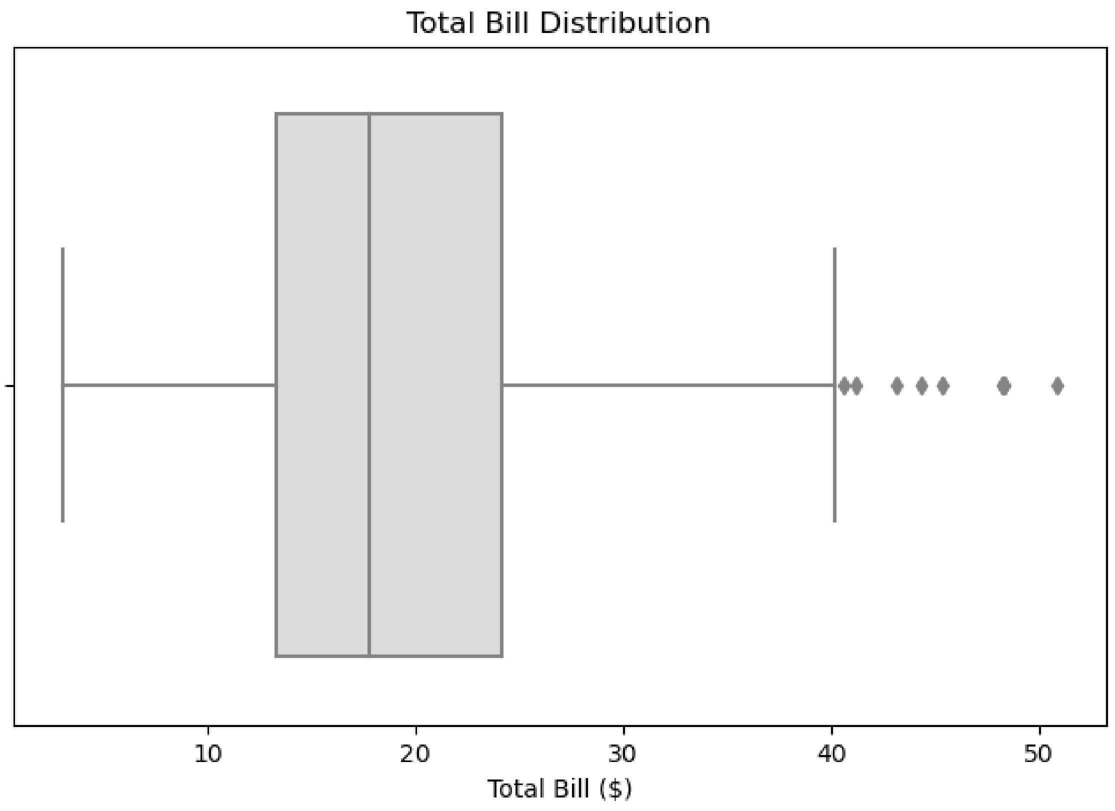


- From KDE, we got the insight that the Bill Amount peak is somewhere between 10–20, but through the Rug plot, we get an idea that the density is more at 15–20.
- Hence, This plot is particularly useful when you want to see the exact position of individual data points. Rug Plots are often combined with 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

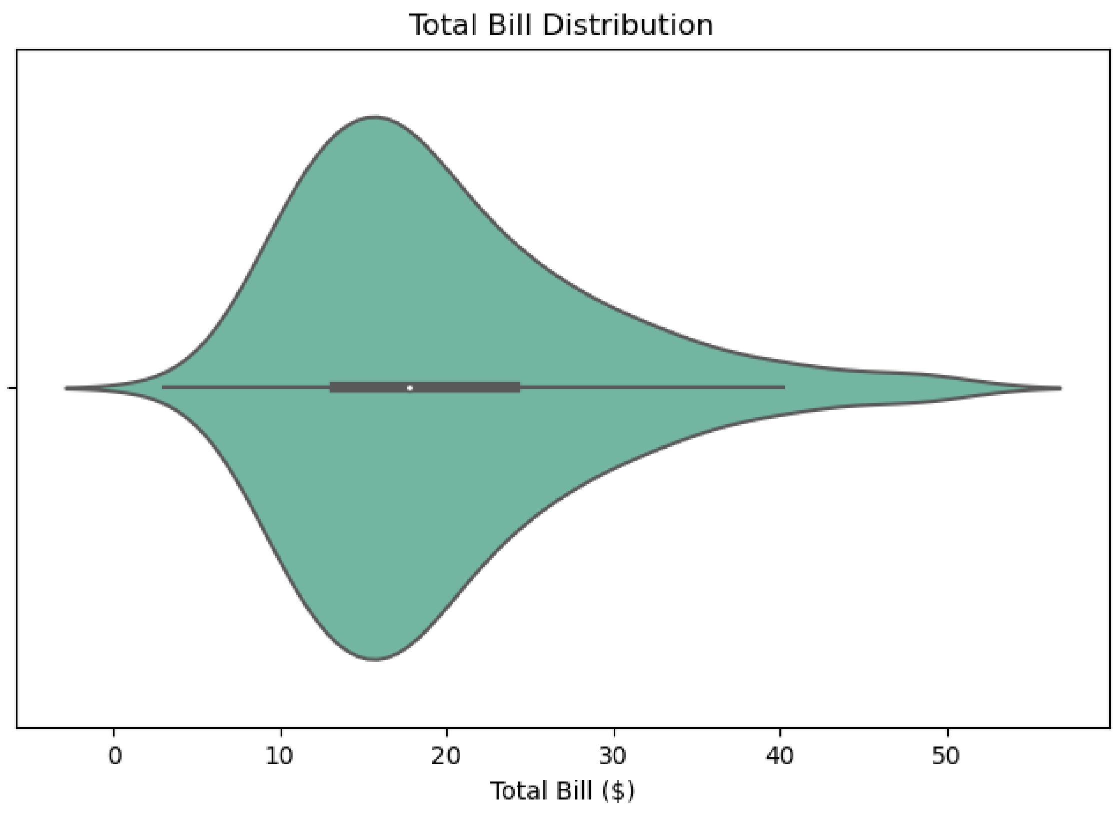
We have discussed these in the categorical plots section, everything is the same, just that you pass only one variable when you want to do univariate analysis.

In [9]:
```python
# 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()
```



Violin Plot

In [10]:
```
# 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()
```

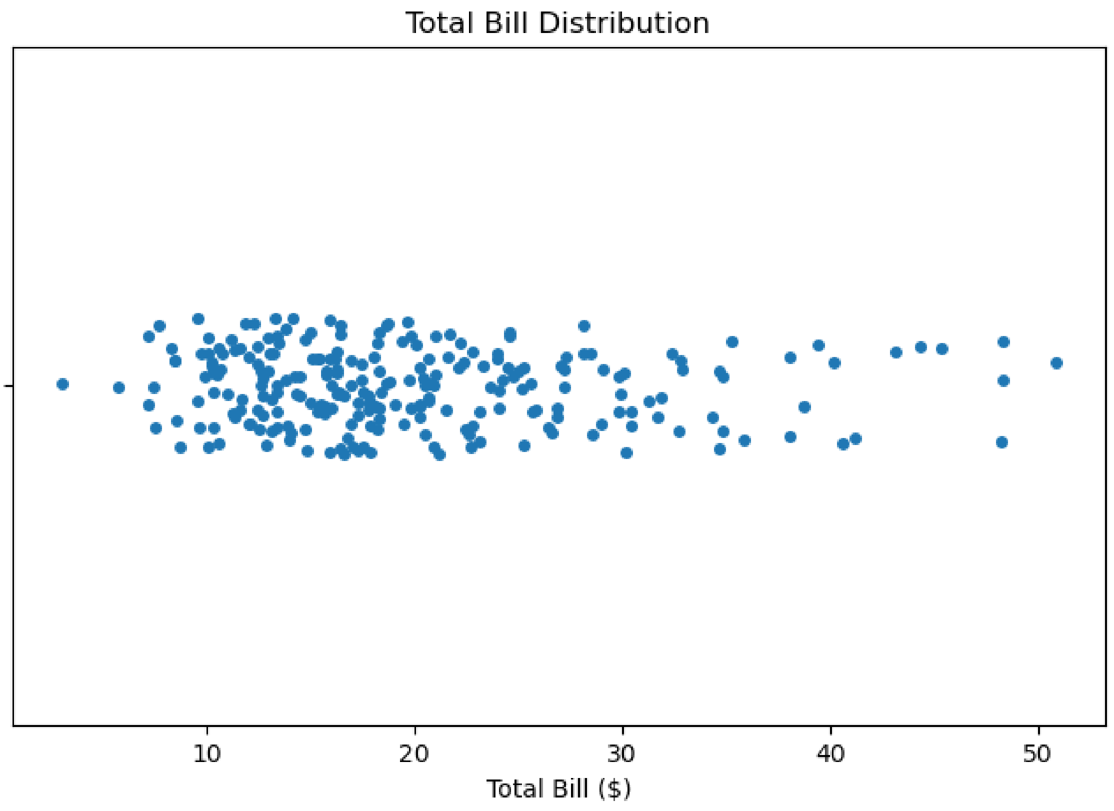


Strip Plot

A Strip Plot is a bit similar to a Swarm Plot and displays individual data points along a single axis. However, the data points will overlap in the strip plot, so we can say that Swarm Plots are particularly useful with smaller datasets and when you want to prevent overlap.

Use sns.stripplot(data=dataframe[column]) , and you can also use this for bivariate. Jittering (adding a small amount of random noise) can be applied to the data points to reduce the overlap and improve visibility.

In [11]:
```python
# 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()
```

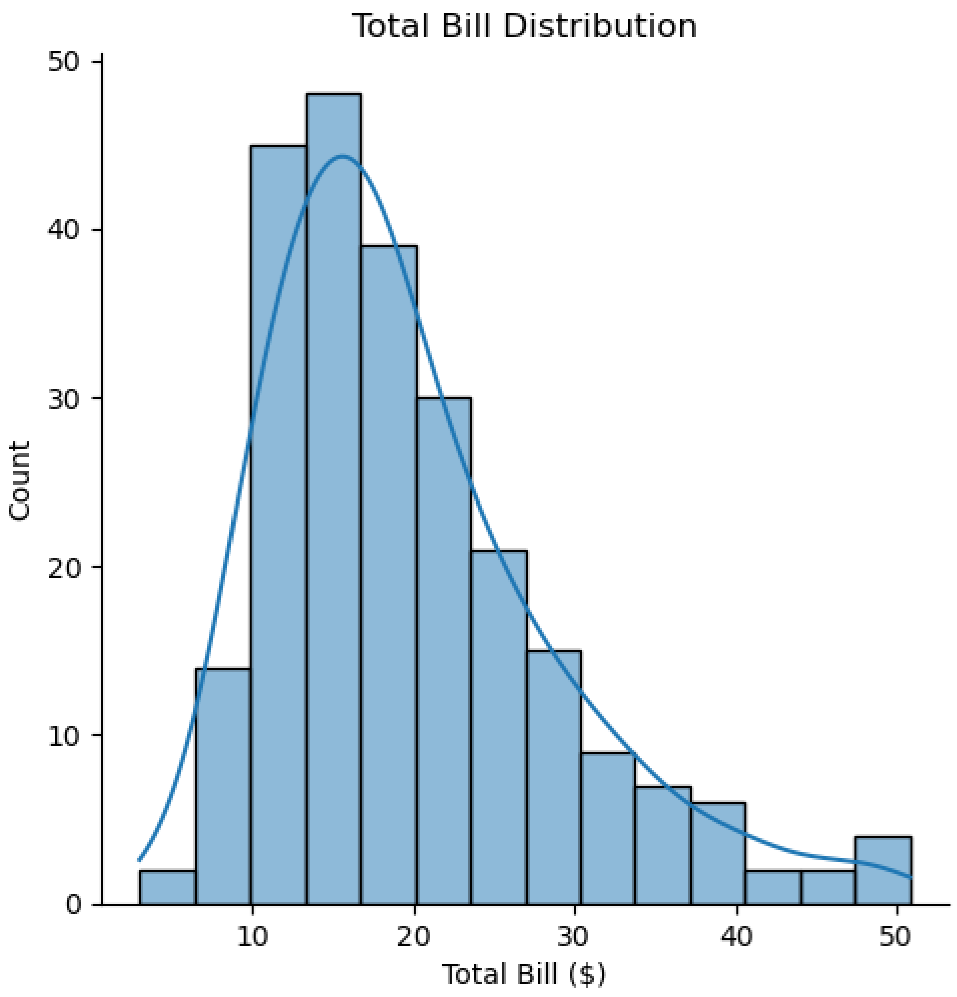


Dist Plot

A Dist Plot short for Distribution plot typically resembles a histogram, but additionally, it includes a smoothed curve, which is a KDE plot. It divides the range of the numerical variable into bins or intervals and displays the frequency or density of data points within each bin.

In [12]:
```python
# 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>
```



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.

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.

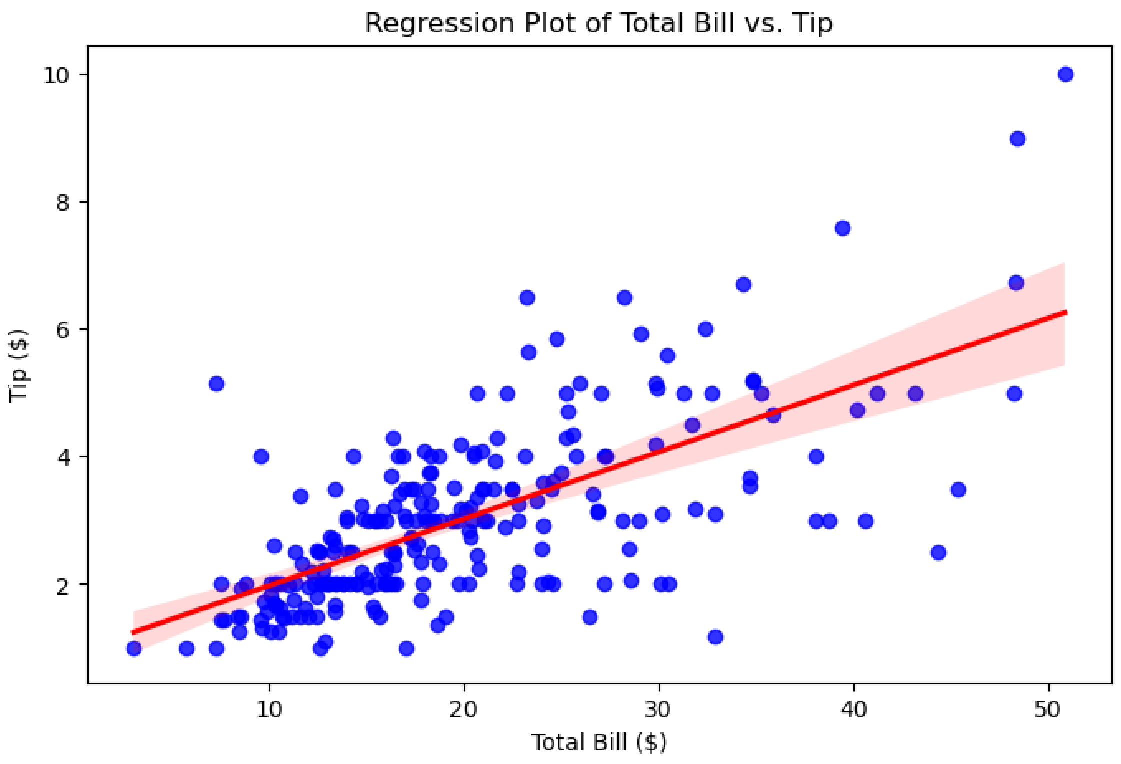
Regression Plot

A Regression Plot focuses on the relationship between two numerical variables: the independent variable (often on the x-axis) and the dependent variable (on the y-axis). There are individual data points are displayed as dots and the central element of a

Regression Plot is the regression line or curve, which represents the best-fitting mathematical model that describes the relationship between the variables.

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

In [13]:
```python
# Regression Plot
plt.figure(figsize=(8, 5))
sns.regplot(x="total_bill", y="tip", data=tips, scatter_kws={"color": "blue
plt.title("Regression Plot of Total Bill vs. Tip")
plt.xlabel("Total Bill ($)")
plt.ylabel("Tip ($)")
plt.savefig('regression_plot.png')
plt.show()
```



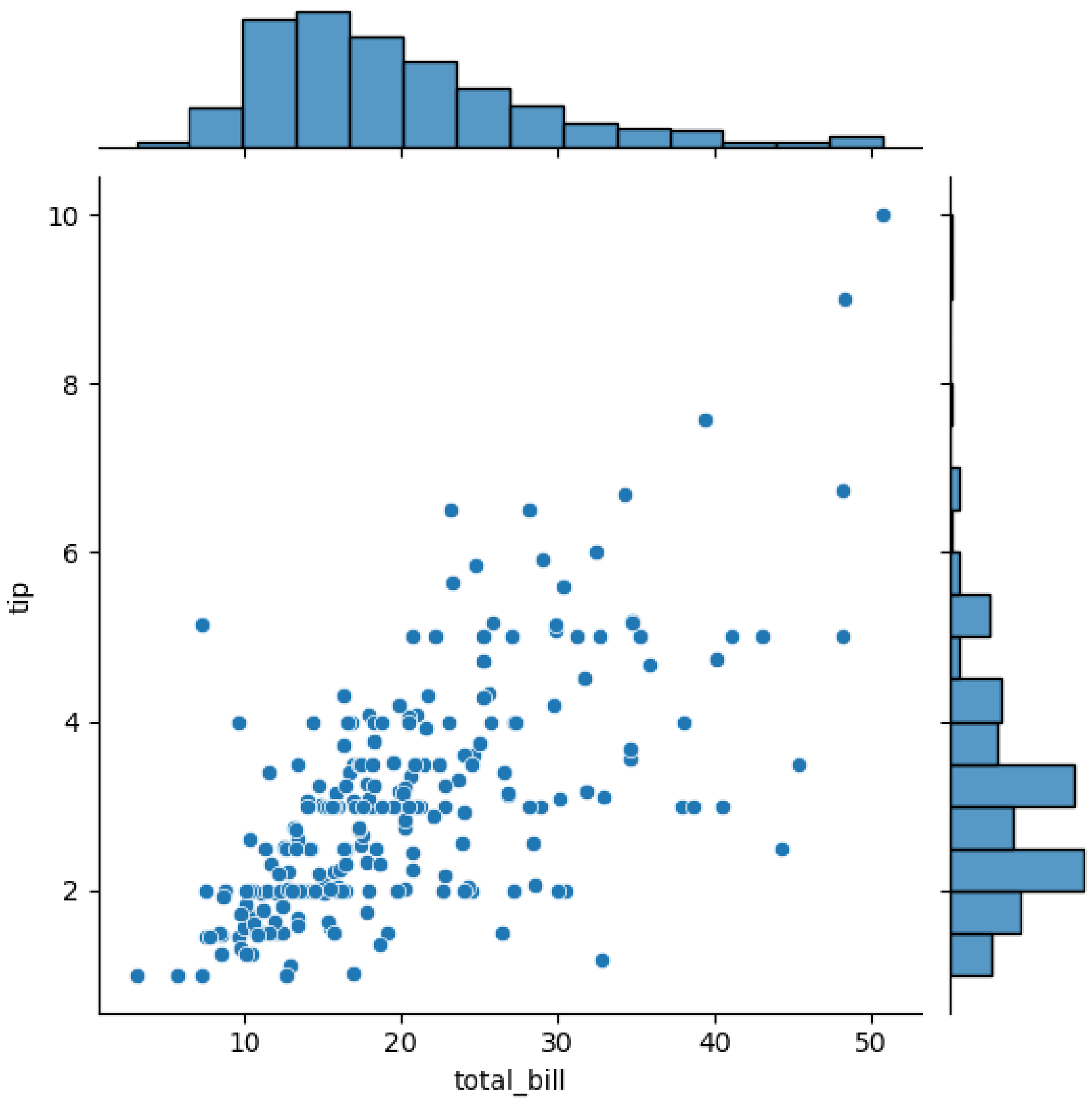
The regression line represents the best-fitting linear model for predicting tips based on total bill amounts. The scatter points show individual data points, and you can observe how they cluster around the regression line. This plot is useful for understanding the linear relationship between these two variables.

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.

```
In [14]: # Joint Plot
         sns.jointplot(x="total_bill", y="tip", data=tips, kind="scatter")
         plt.savefig('joint_plot.png')
         plt.show()
```



As we can see, this shows the relation between the two variables through a scatter plot, while the marginal histograms show the distribution of each variable separately.

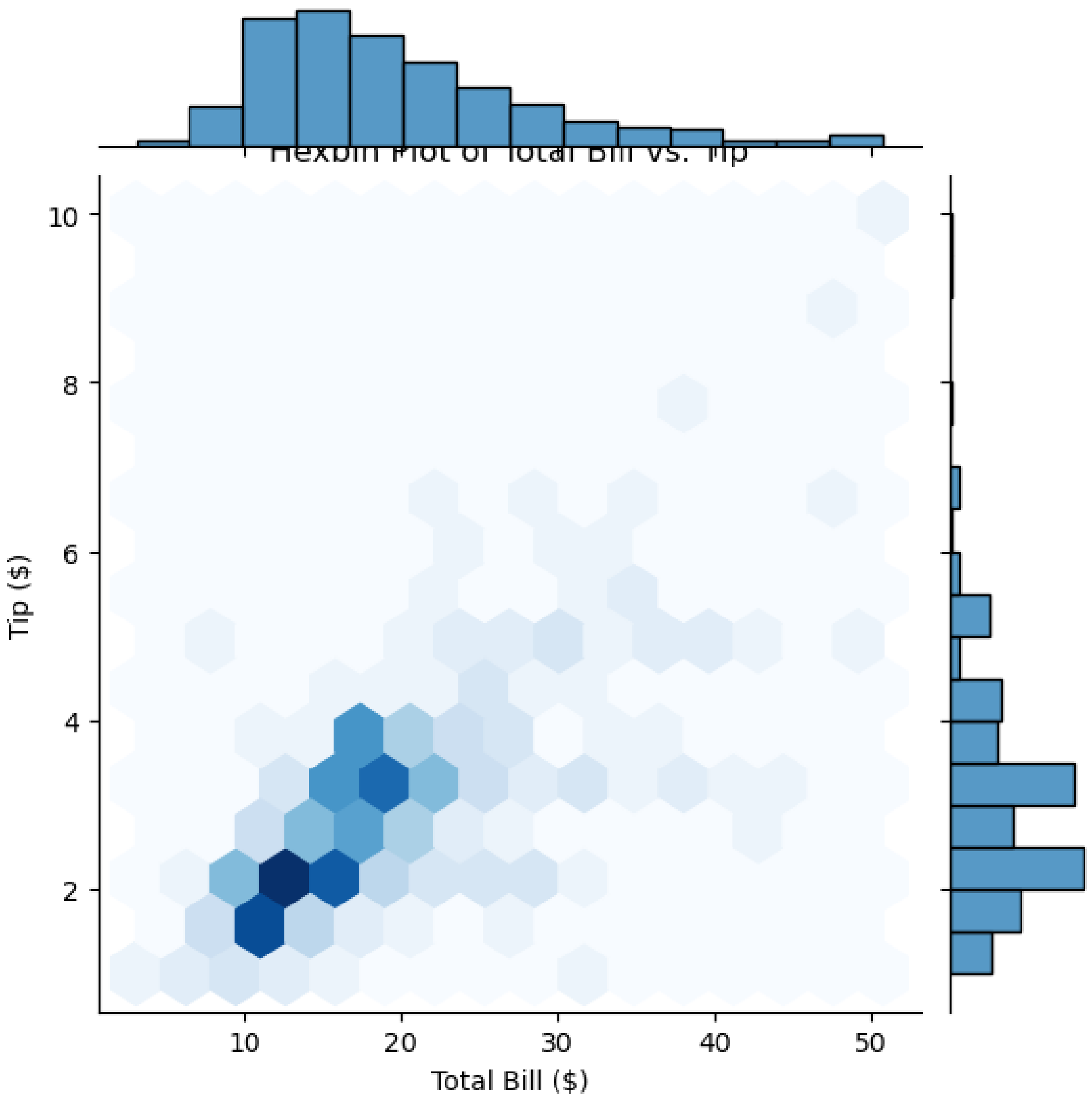
Hexbin Plot

A Hexbin plot, short for Hexagonal Binning plot, groups data points into hexagonal bins, allowing you to visualize data density and patterns more effectively. These are especially valuable with large datasets when scatter plots with individual points become too crowded and hard to interpret!

You can create a Hexbin plot, by using the kind parameter of joint plot. You can customize the color map, grid size, and other plot parameters to fine-tune the appearance of the Hexbin Plot.

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

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



Now, this confirms that 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.

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.

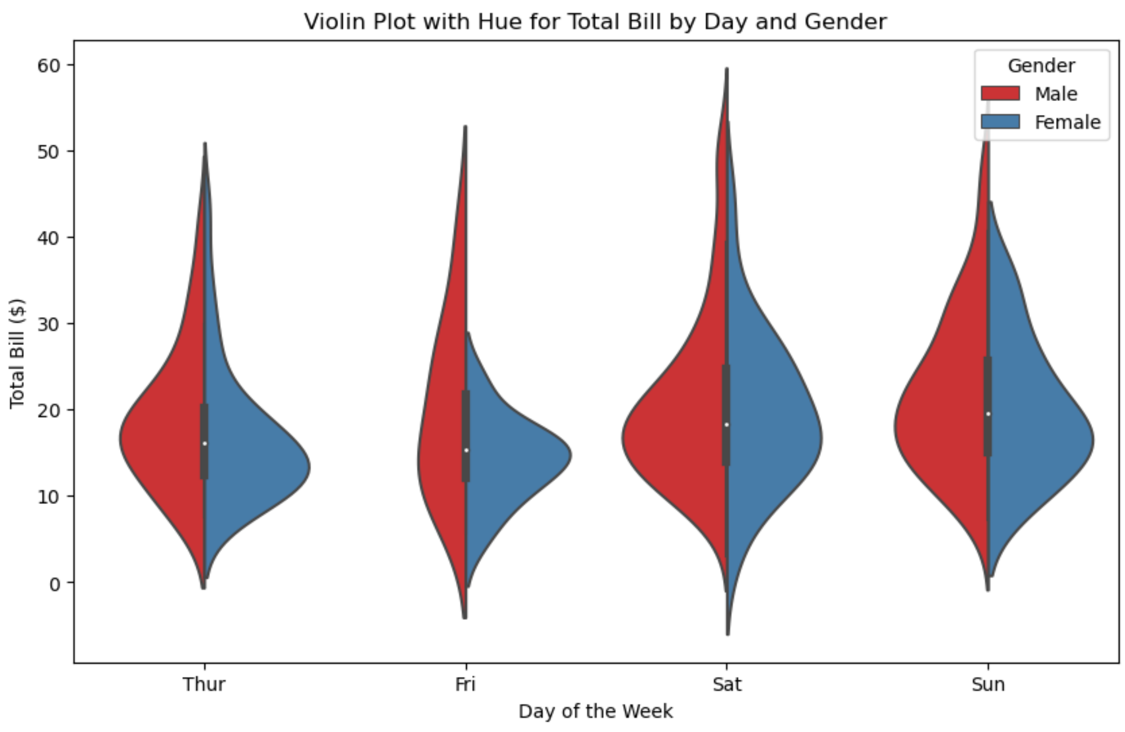
Using Parameters

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.

In [16]:
```
# Violin Plot with Hue
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()
```



- We 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.

Using hue and size parameters:

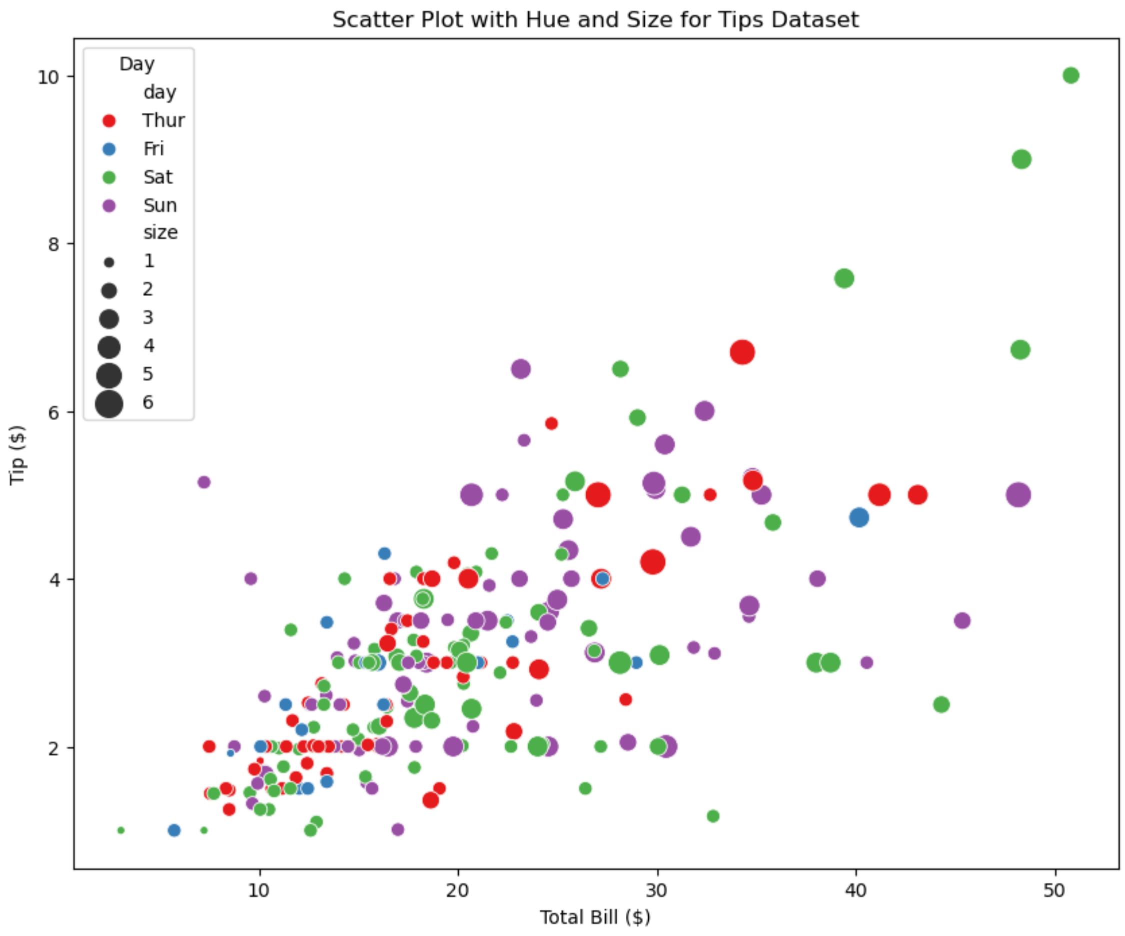
size is another parameter that can be used to add another numeric variable dimension.

```
In [17]: 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()
```



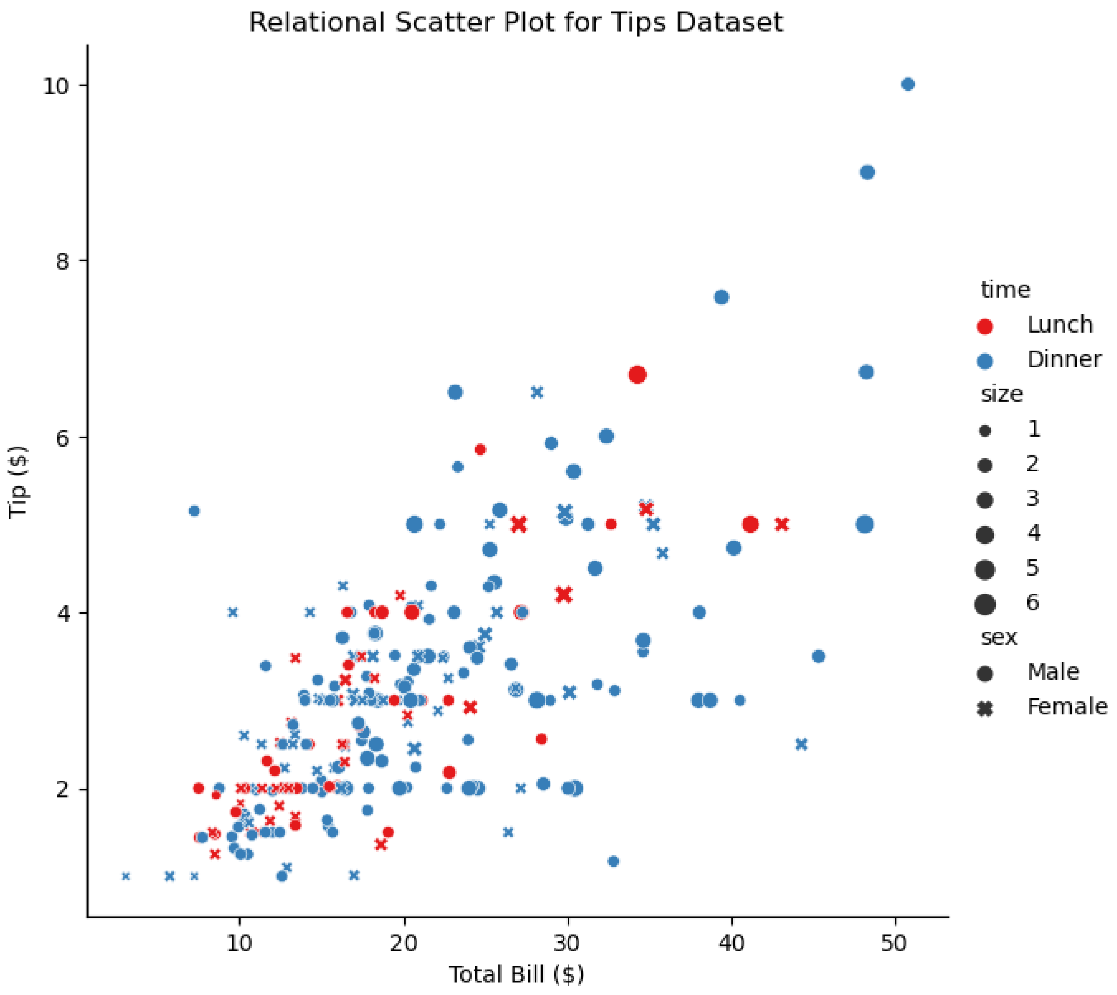
Each day is represented by a different color with the hue parameter adding a third dimension, and marker size based on the size column in the dataset to add a fourth dimension.

Relational Plot

A Relational Plot allows you to visualize the relationship between two numerical variables, along with additional categorical or numerical dimensions.

Use sns.relplot(x,y,data,hue,size,style). you can use the hue parameter to color data points by a categorical variable, the size parameter to vary marker size based on a numerical variable, and the style parameter to differentiate markers or lines by a categorical variable. The kind parameter allows you to specify the type of relational plot you want to create.

In [18]:
```
# Create a scatterplot using a Relational Plot (relplot)
sns.relplot(x="total_bill", y="tip", data=tips, hue="time", style="sex", si
plt.title("Relational Scatter Plot for Tips Dataset")
plt.xlabel("Total Bill ($)")
plt.ylabel("Tip ($)")
plt.savefig('relplot_with_mv.png')
plt.show()
```



The time column is used for coloring data points with the hue parameter, size column for varying size with the size parameter, and sex column for the marker parameter.

FacetGrid by using col and row parameters of relplot

You can also create a Relational Plot (relplot) using the col and row parameters to create a

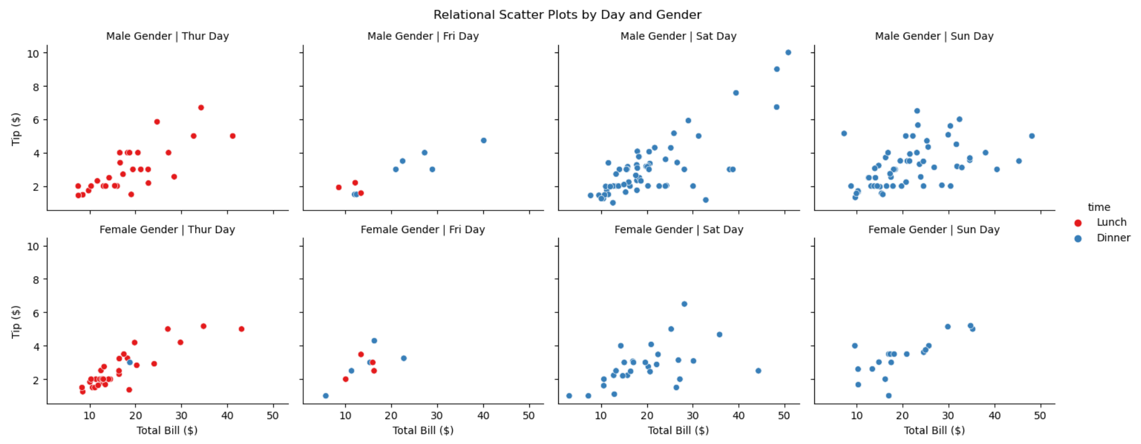
In [19]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# 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()
```

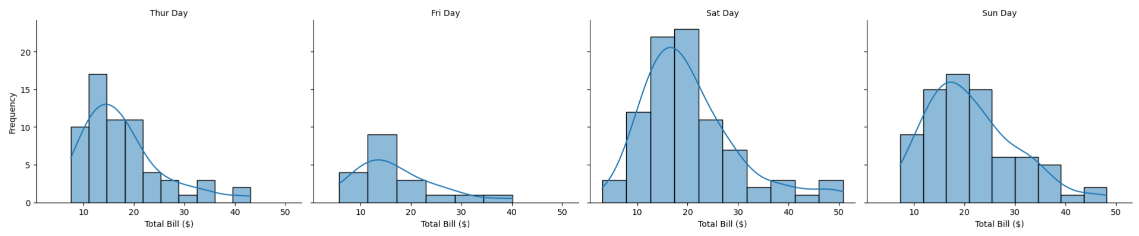


FacetGrid

A Facet Grid is a feature in Seaborn that allows you to create a grid of subplots, each representing a different subset of your 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 you need to map it to any plot of your choice.

In [20]:
```python
# 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()
```



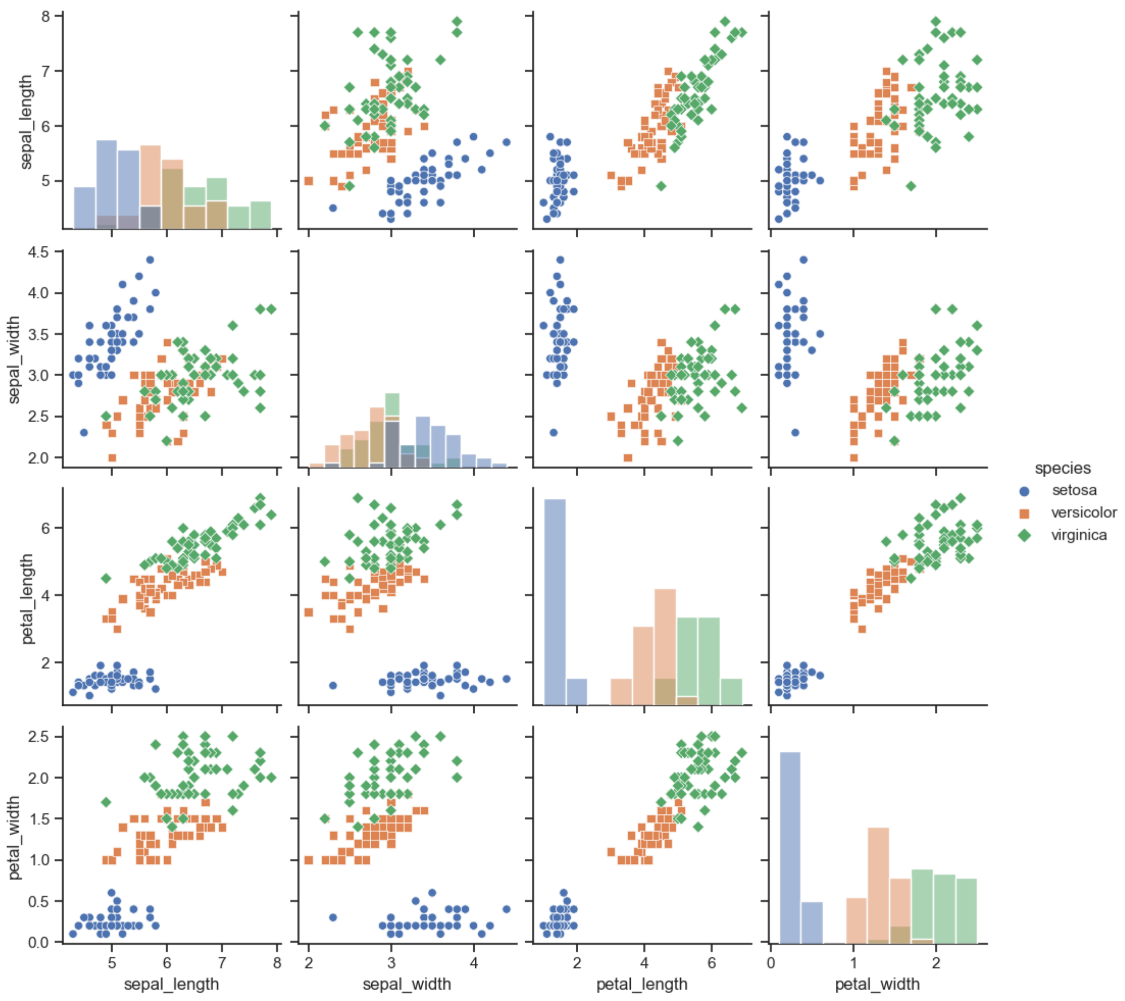
Pair Plot

A Pair plot provides a grid of scatterplots, and histograms, where each plot shows the relationship between two variables, which is why it is also called a Pairwise Plot or Scatterplot Matrix.

The diagonal cells typically display histograms or kernel density plots for individual variables, showing their distributions. The off-diagonal cells in the grid often display scatterplots, showing how two variables are related. Pair Plots are particularly useful for understanding patterns, correlations, and distributions across multiple dimensions in your data.

In [21]:
```
# 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()
```



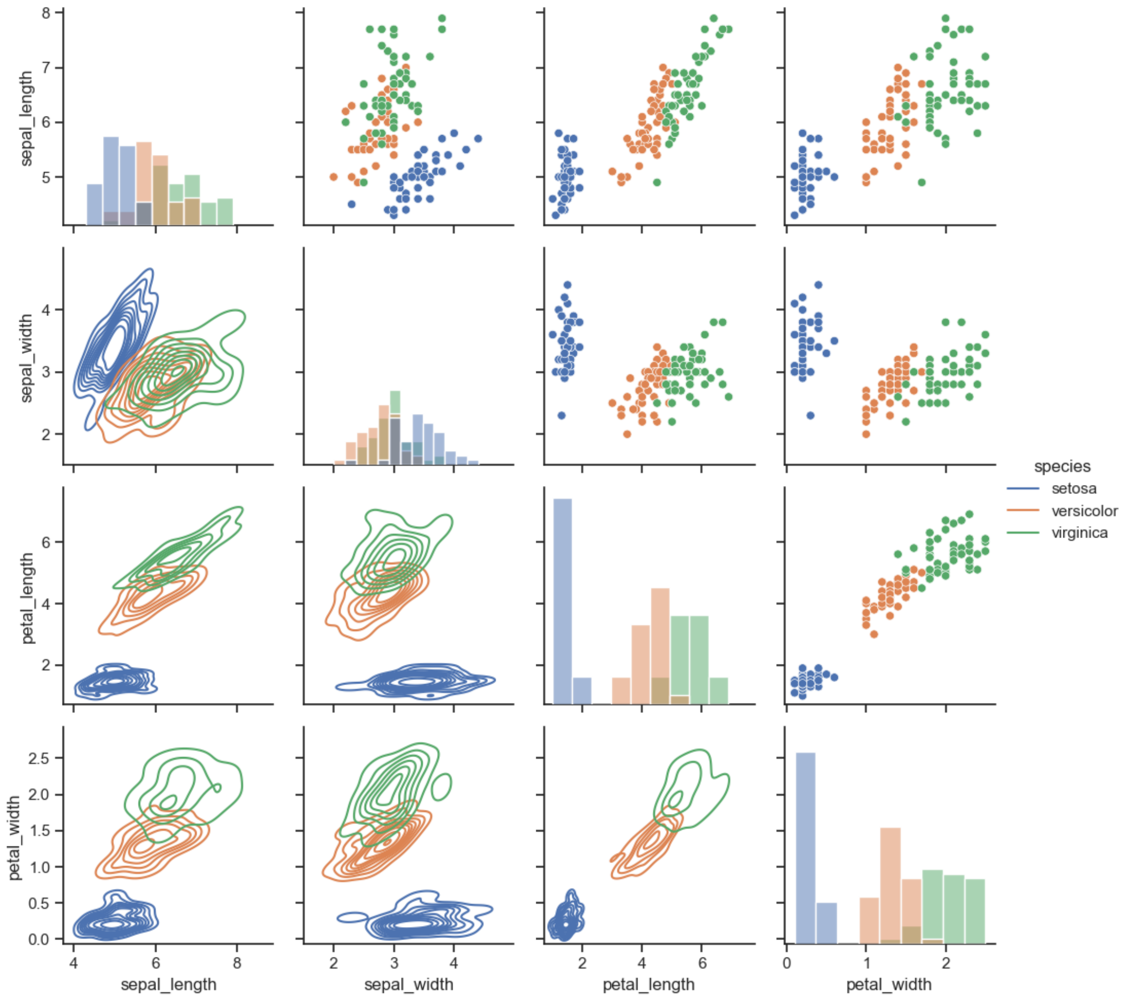
Pair Grid

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

In [22]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Load the "iris" dataset
iris = sns.load_dataset("iris")

# Create a Facet Grid of pairwise scatterplots
g = sns.PairGrid(iris, hue="species")
g.map_upper(sns.scatterplot)
g.map_diag(sns.histplot, kde_kws={"color": "k"})
g.map_lower(sns.kdeplot)
g.add_legend()
plt.savefig('pair_grid.png')
plt.show()
```



6. Matrix Plots

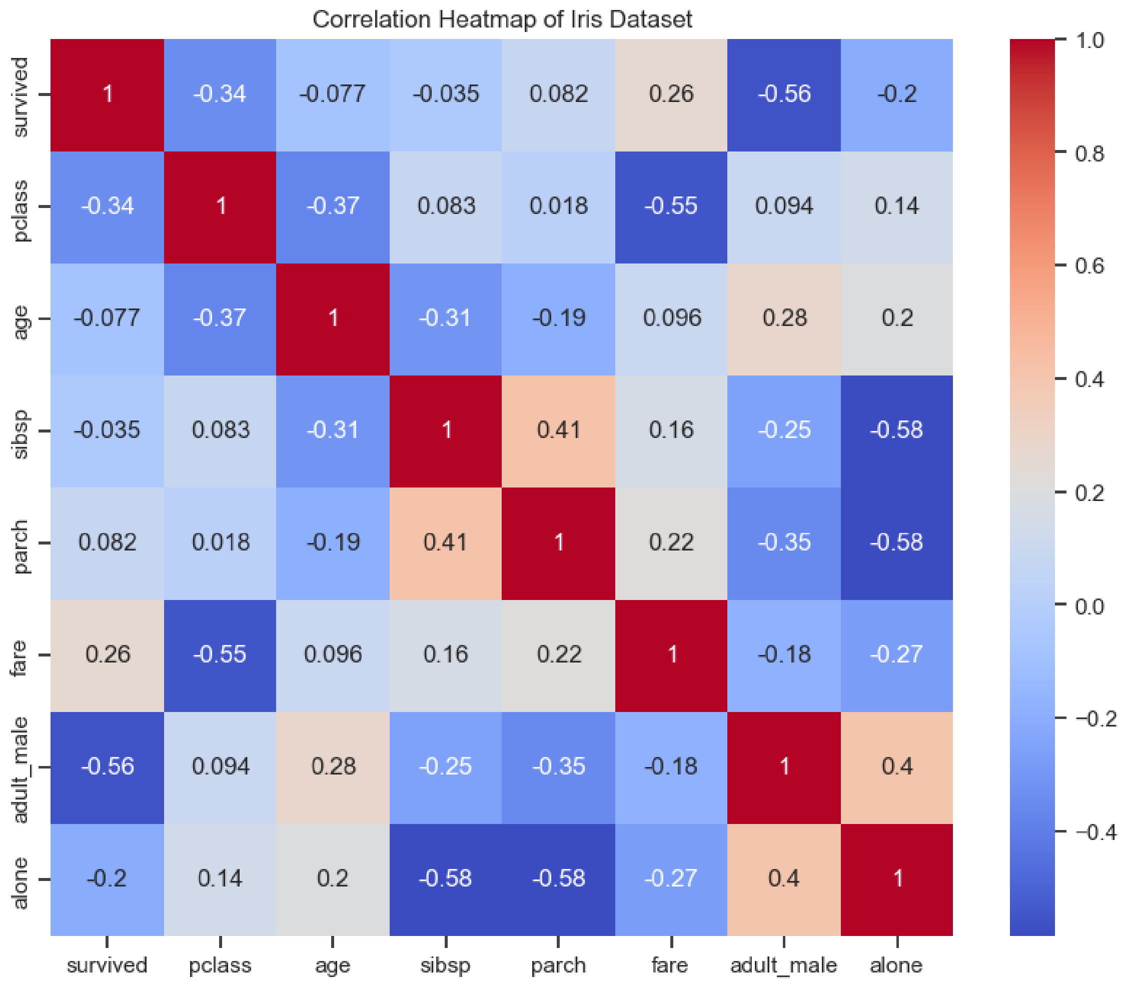
These plots visualize relationships within matrices or grids of data.

Heat Map

Use dataframe.corr() pandas data frame method to get the correlations, and then pass it to the sns.heatmap(corr_matrix) method to plot the heatmap. Let’s see an example.

```
In [23]: # Sample data as a correlation matrix
         correlation_matrix = sns.load_dataset("titanic").corr(numeric_only=True)

         plt.figure(figsize=(10, 8))
         # Create a heatmap of the correlation matrix
         sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
         plt.title("Correlation Heatmap of Iris Dataset")
         plt.savefig('corr_matrix.png')
         plt.show()
```



	survived	pclass	age	sibsp	parch	fare	adult_male	alone
survived	1	-0.34	-0.077	-0.035	0.082	0.26	-0.56	-0.2
pclass	-0.34	1	-0.37	0.083	0.018	-0.55	0.094	0.14
age	-0.077	-0.37	1	-0.31	-0.19	0.096	0.28	0.2
sibsp	-0.035	0.083	-0.31	1	0.41	0.16	-0.25	-0.58
parch	0.082	0.018	-0.19	0.41	1	0.22	-0.35	-0.58
fare	0.26	-0.55	0.096	0.16	0.22	1	-0.18	-0.27
adult_male	-0.56	0.094	0.28	-0.25	-0.35	-0.18	1	0.4
alone	-0.2	0.14	0.2	-0.58	-0.58	-0.27	0.4	1

The main point from the heatmap is that if you find variables that are highly correlated, it's better to combine those columns as they both contribute the same.

Cluster Map

Last but not least, The Cluster Map! Don't you think it will be easy to interpret if the like color squares are close to each other? And that's what exactly cluster map does. The primary purpose of a Cluster Map is to reveal clusters or groups of similar rows and columns.

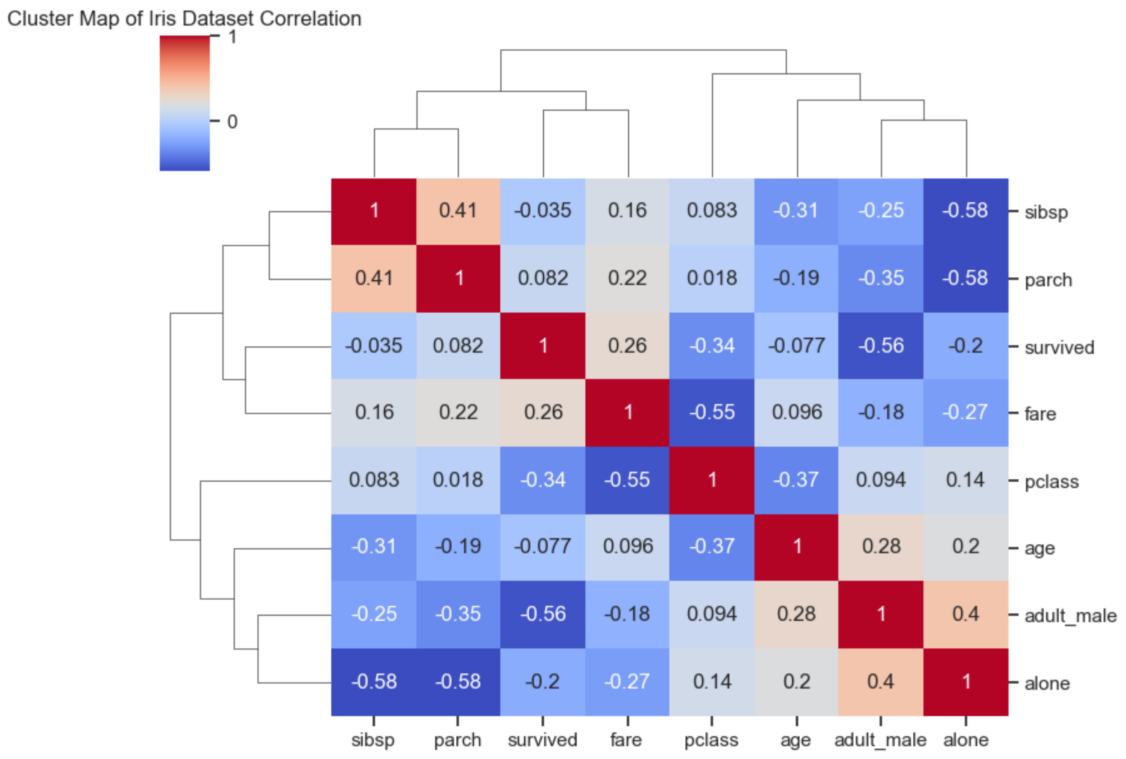
A Cluster Map is a type of heatmap in Seaborn that not only displays data as a grid of colored cells but also arranges rows and columns in a way that groups similar data together, it uses hierarchical clustering algorithms to reorder rows and columns in the heatmap. The side of the grouped heatmap, it includes dendrograms to make it a cluster map. Dendrograms are tree-like diagrams that show the hierarchical relationships between rows and columns.

Use sns.clustermap(data.corr()) to create a cluster map.

```
In [24]:
import seaborn as sns
import matplotlib.pyplot as plt

# Load a sample dataset
data = sns.load_dataset("titanic")

# Create a Cluster Map of the correlation matrix
sns.clustermap(data.corr(numeric_only=True), annot=True, cmap="coolwarm", 
plt.title("Cluster Map of Iris Dataset Correlation")
plt.savefig('cluster_map.png')
plt.show()
```



If you observe now, The rows are reordered based on how strongly they correlate with each other. Variables that are more strongly negatively correlated with each other are placed close together, creating clusters of related variables. Similarly, the Positively correlated ones. That’s what a cluster map does pointing clusters in a heatmap.