Seaborn

- A Python library built on top of Matplotlib for creating visually appealing statistical graphics.
- Best for creating heatmaps, violin plots, pair plots, and categorical plots (like bar, box, and strip plots).
- It simplifies complex visualization tasks with high-level functions

Importing Seaborn and Dataset

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load an example dataset
tips = sns.load_dataset("tips")
print(tips.head())
```

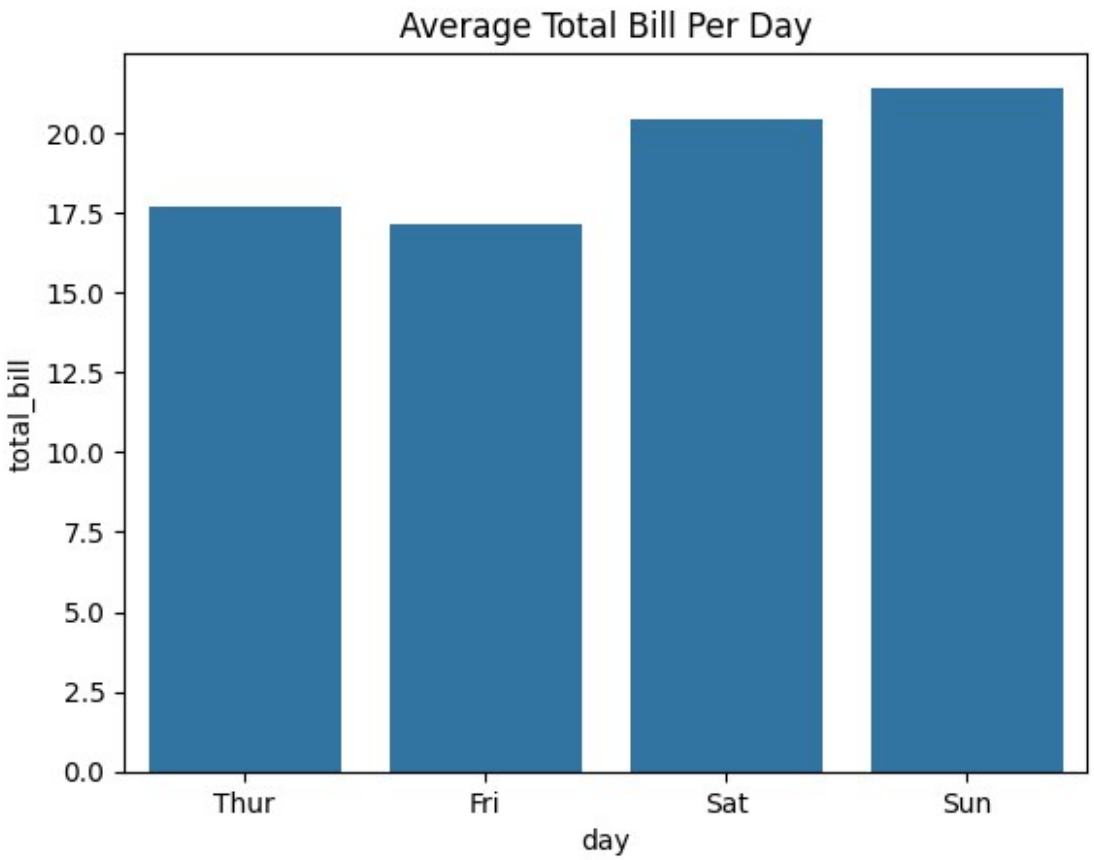
```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female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

Bar Plot

What: A bar plot is used to show the distribution of categorical data with rectangular bars where the length represents the value.

Why: Use it when you want to compare quantities across categories (e.g., total bill across different days).

```
# Bar plot: Average total bill per day
sns.barplot(x="day", y="total_bill", data=tips, errorbar=None)
plt.title("Average Total Bill Per Day")
plt.show()
```

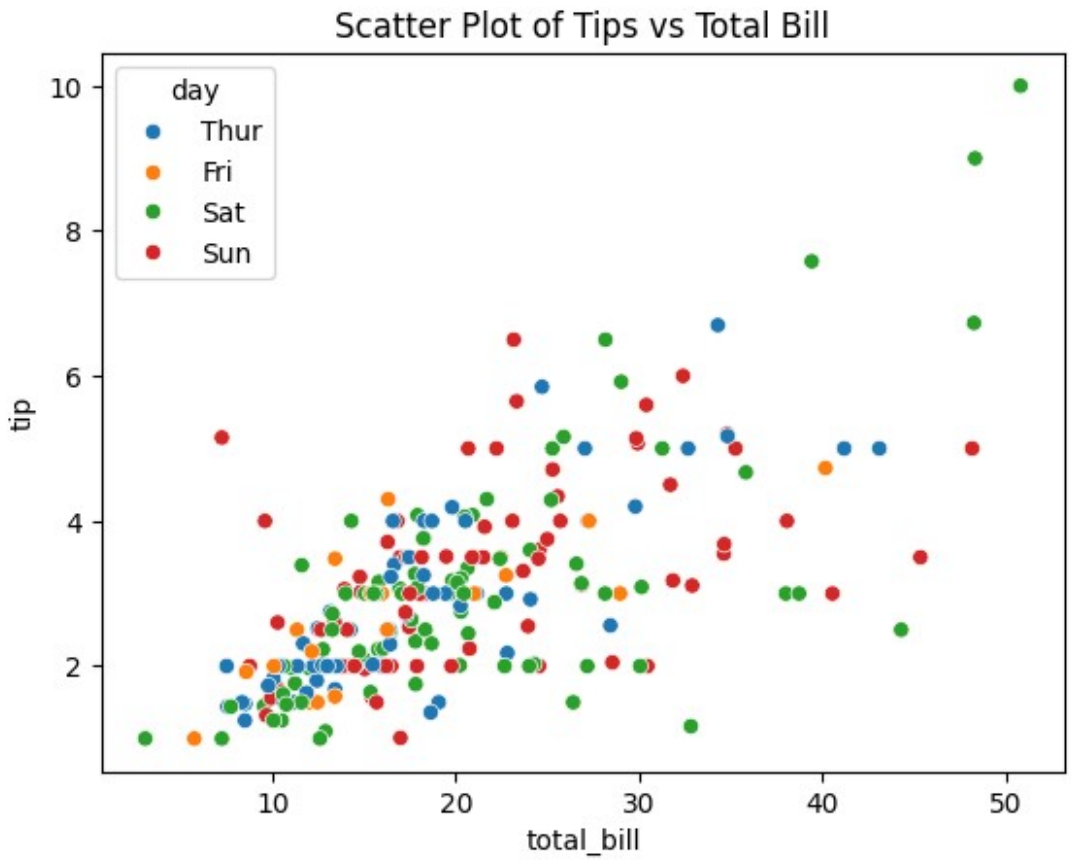


Scatter Plot

What: A scatter plot shows the relationship between two continuous variables, with each point representing a pair of values.

Why: Use it to identify correlations, trends, or clusters in data.

```
# Scatter plot: Relationship between total bill and tip
sns.scatterplot(x="total_bill", y="tip", hue="day", data=tips)
plt.title("Scatter Plot of Tips vs Total Bill")
plt.show()
```

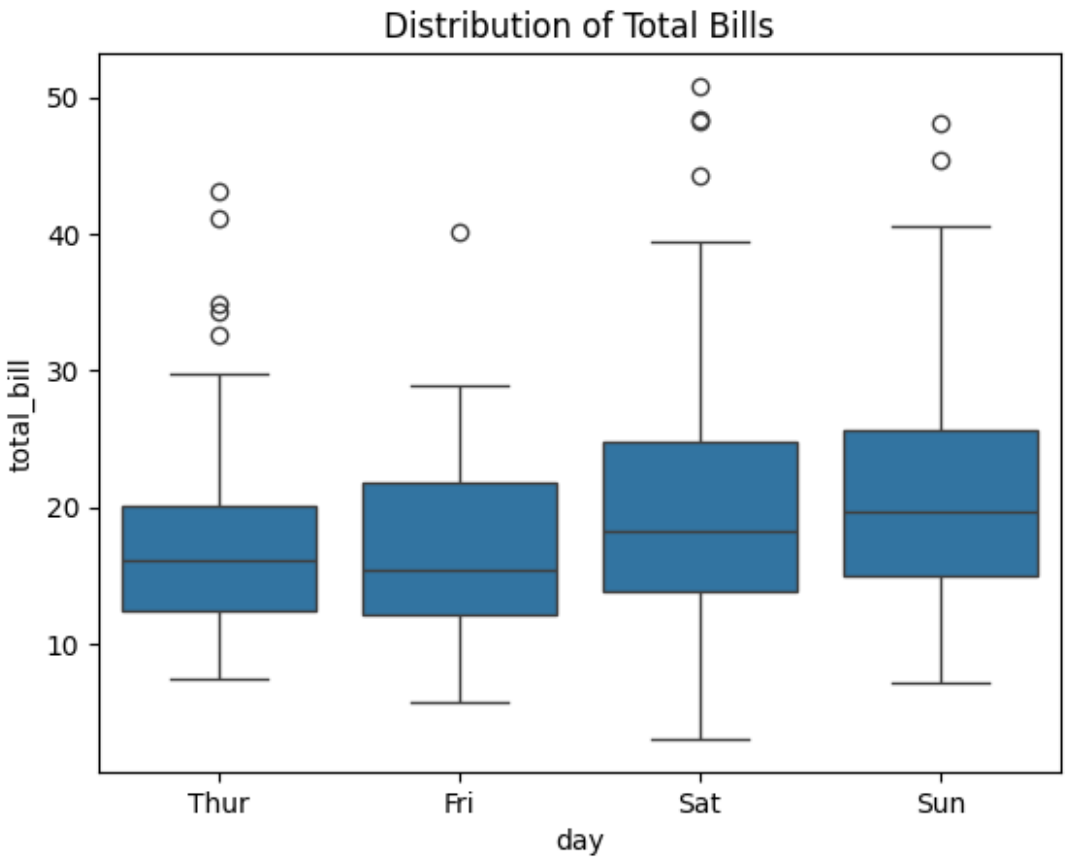


Box Plot

What: A box plot visualizes the distribution of data based on five summary statistics: minimum, first quartile (Q1), median, third quartile (Q3), and maximum.

Why: Use it to detect outliers, understand the spread, and compare distributions across categories.

```
# Box plot: Distribution of total bills per day
sns.boxplot(x="day", y="total_bill", data=tips)
plt.title("Distribution of Total Bills")
plt.show()
```

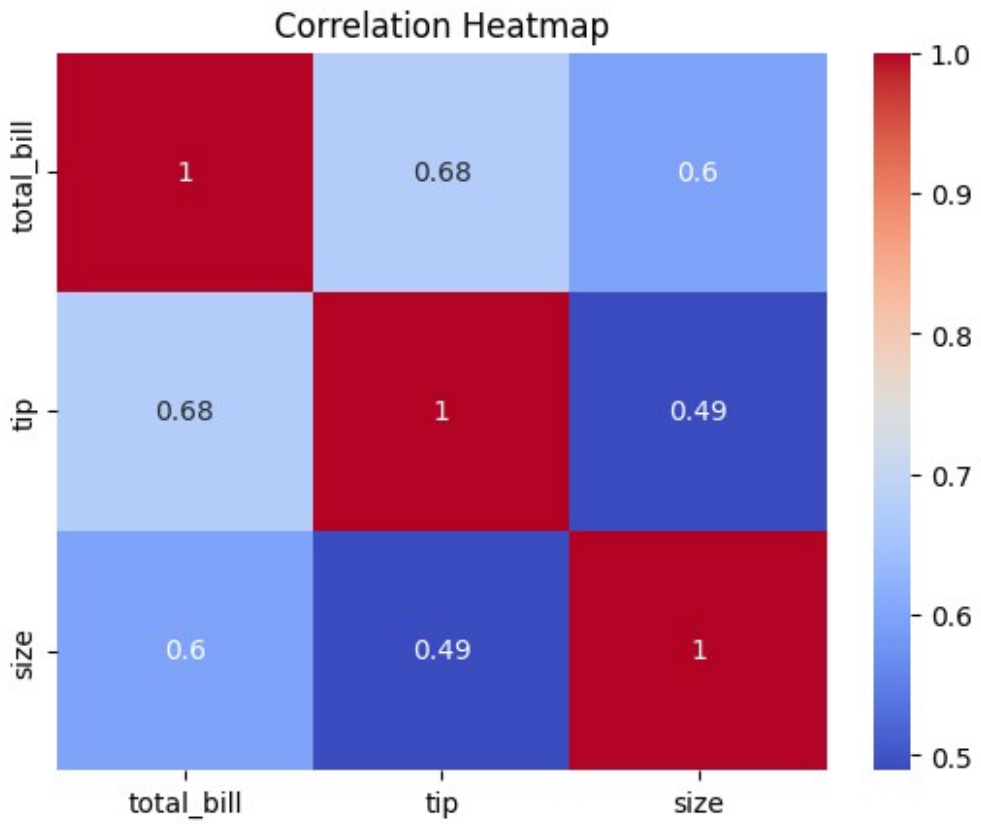


Heatmap

What: A heatmap displays data in matrix format, where values are represented by colors, useful for visualizing correlations or intensity.

Why: Use it to observe patterns, correlations, or relationships between numerical variables.

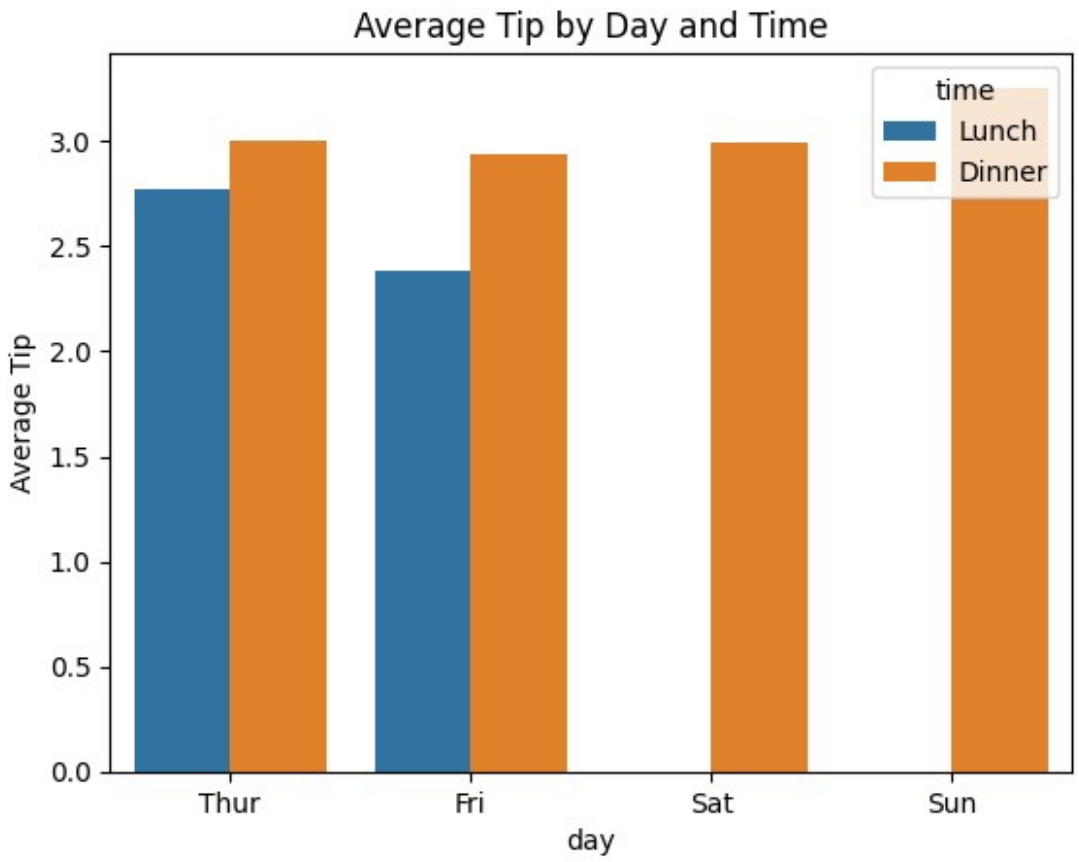
```
# Select numeric columns from the dataset
numeric_tips = tips.select_dtypes(include=["float64", "int64"])

# Compute the correlation matrix
correlation = numeric_tips.corr()

# Display the heatmap
sns.heatmap(correlation, annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```



Bar Plot: Average Tip by Day and Time

Bar plots are great for summarizing and comparing values.

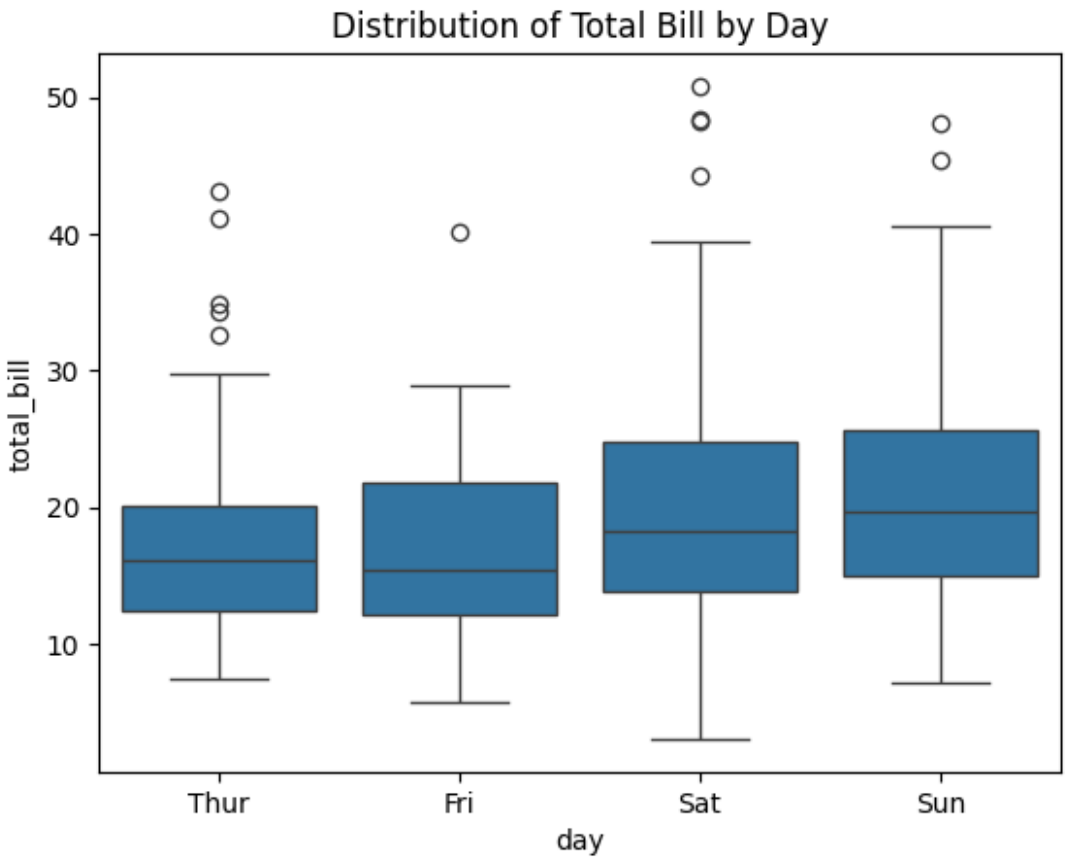
```
import seaborn as sns
import matplotlib.pyplot as plt

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

# Bar plot: Average tip by day and time
sns.barplot(x="day", y="tip", hue="time", data=tips, errorbar=None)
plt.title("Average Tip by Day and Time")
plt.ylabel("Average Tip")
plt.show()
```



Box Plot: Distribution of Total Bill by Day

Box plots help visualize the spread and outliers in the data.

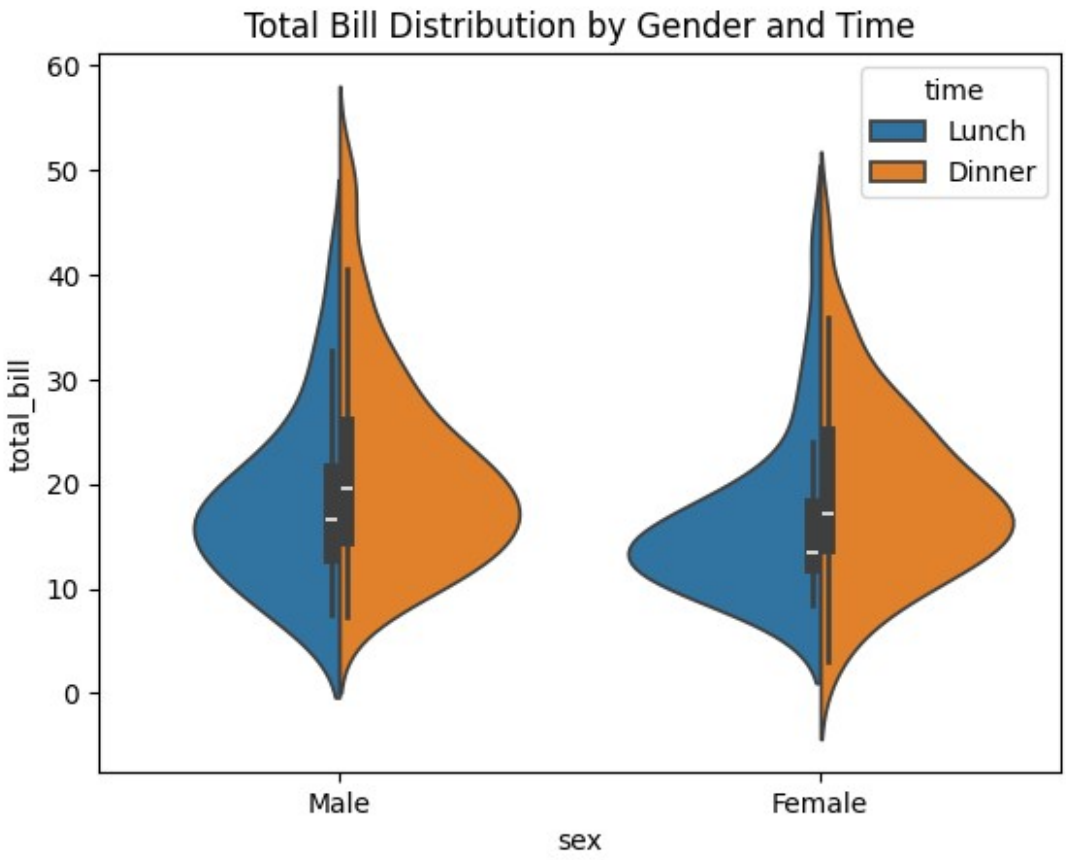
```
# Box plot: Total bill by day
sns.boxplot(x="day", y="total_bill", data=tips)
plt.title("Distribution of Total Bill by Day")
plt.show()
```



Violin Plot: Total Bill by Gender and Time

Violin plots show the distribution and density of the data.

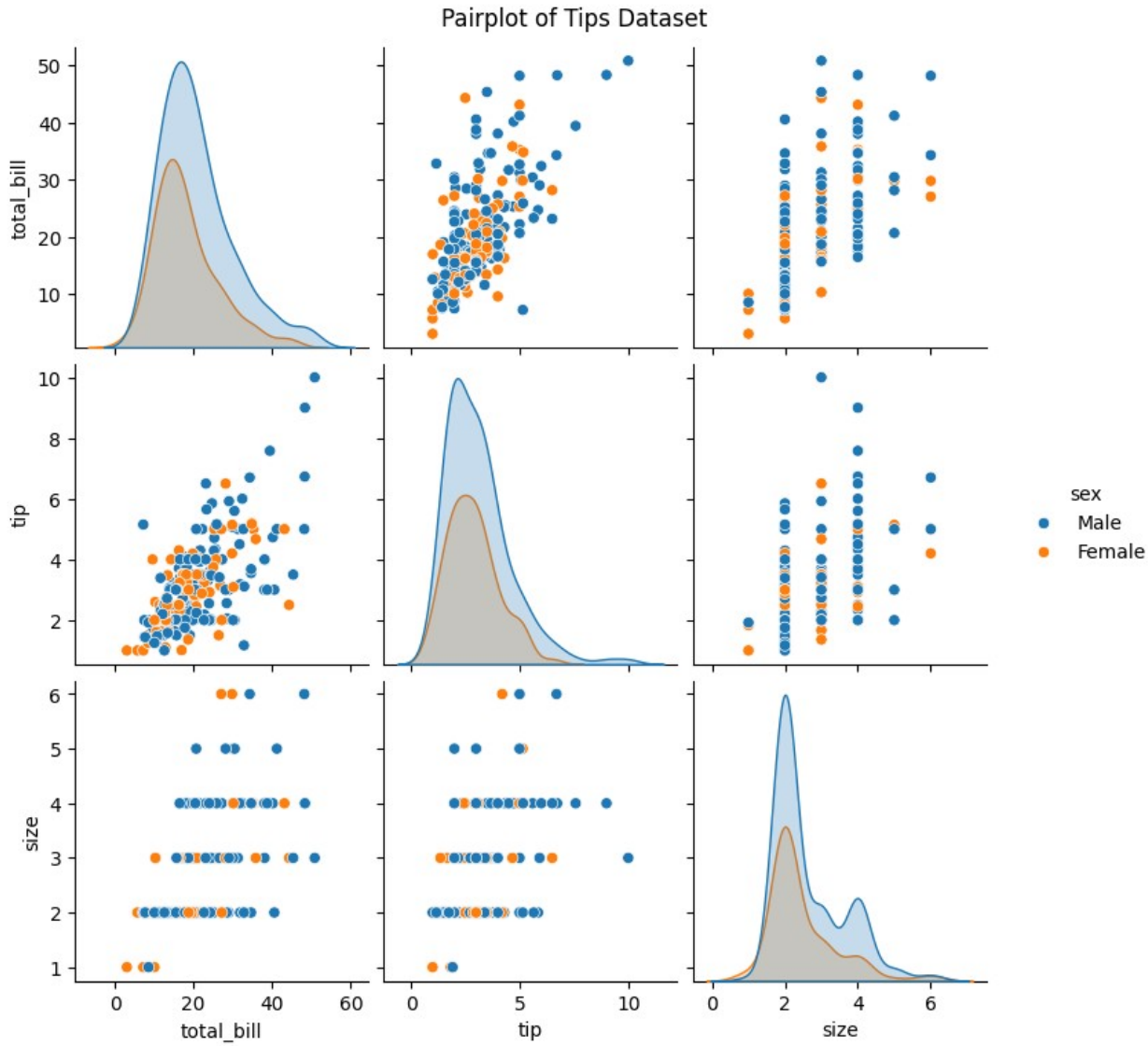
```
# Violin plot: Total bill by gender and time
sns.violinplot(x="sex", y="total_bill", hue="time", data=tips,
split=True)
plt.title("Total Bill Distribution by Gender and Time")
plt.show()
```



Pairplot: Relationships Between Variables

Pair plots are ideal for exploring relationships in datasets with numerical and categorical features.

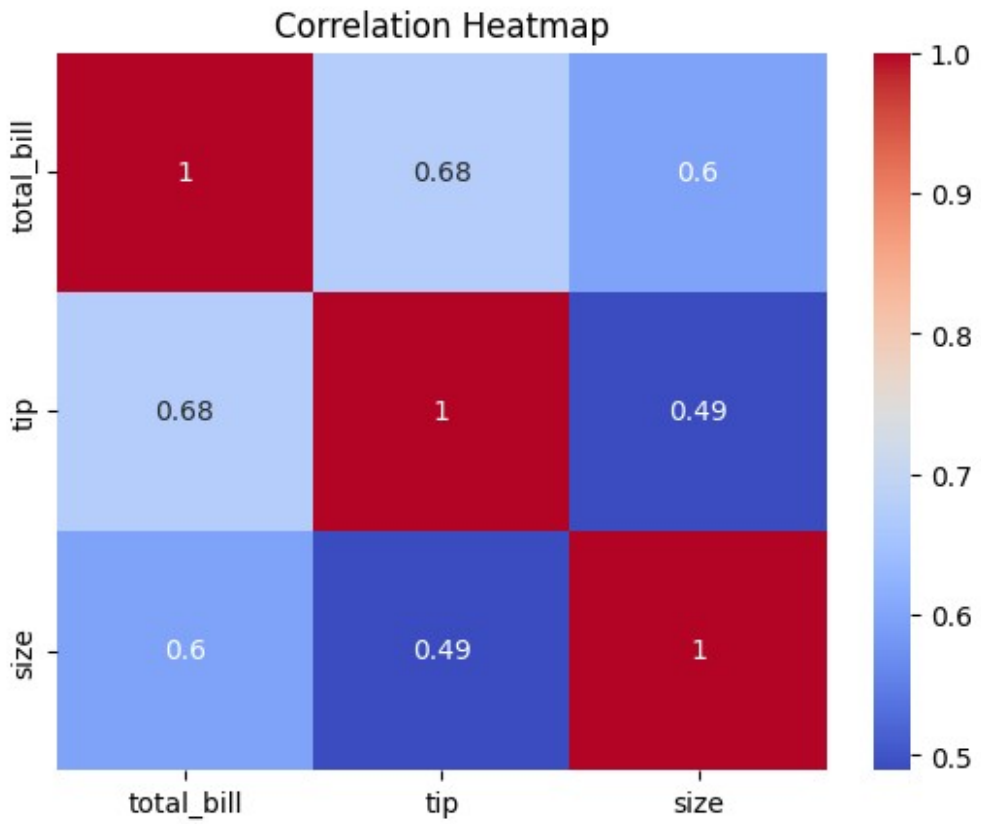
```
# Pairplot for numerical relationships in tips
sns.pairplot(tips, hue="sex")
plt.suptitle("Pairplot of Tips Dataset", y=1.02)
plt.show()
```



Heatmap: Correlation Between Numerical Variables

A heatmap can help identify patterns and strong correlations.

```
# Select numeric columns from the dataset
numeric_tips = tips.select_dtypes(include=["float64", "int64"])

# Compute the correlation matrix
correlation = numeric_tips.corr()

# Display the heatmap
sns.heatmap(correlation, annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```



Plotly

- A library for creating interactive visualizations.
- Allows for dynamic updates, hover functionality, zooming, and exporting graphs.
- Offers integration with Dash for creating interactive web apps. Best for line plots, scatter plots, 3D plots, and dashboards.

Importing Plotly and Dataset

```
import plotly.express as px

# Load example data
df = px.data.iris()
print(df.head(10))
```

```
   sepal_length  sepal_width  petal_length  petal_width species
species_id
0           5.1          3.5           1.4          0.2  setosa
1
1           4.9          3.0           1.4          0.2  setosa
1
2           4.7          3.2           1.3          0.2  setosa
1
```

```
3            4.6           3.1            1.5           0.2  setosa
1
4            5.0           3.6            1.4           0.2  setosa
1
5            5.4           3.9            1.7           0.4  setosa
1
6            4.6           3.4            1.4           0.3  setosa
1
7            5.0           3.4            1.5           0.2  setosa
1
8            4.4           2.9            1.4           0.2  setosa
1
9            4.9           3.1            1.5           0.1  setosa
1
```

Line Plot

```
# Line plot: Sepal length across samples
fig = px.line(df, x=df.index, y="sepal_length", title="Sepal Length
Trend")
fig.show()

# Line plot: Petal length across samples
fig = px.line(df, x=df.index, y="petal_length", title="Petal Length
Trend")
fig.show()
```

Scatter Plot

```
# Scatter plot: Sepal vs Petal Length
fig = px.scatter(df, x="sepal_length", y="petal_length",
color="species", title="Sepal vs Petal Length")
fig.show()
```

Bar Plot

```
# Bar plot: Average sepal width per species
fig = px.bar(df, x="species", y="sepal_width", title="Average Sepal
Width by Species")
fig.show()

# Bar plot: Average petal width per species
fig = px.bar(df, x="species", y="petal_width", title="Average Petal
Width by Species")
fig.show()
```

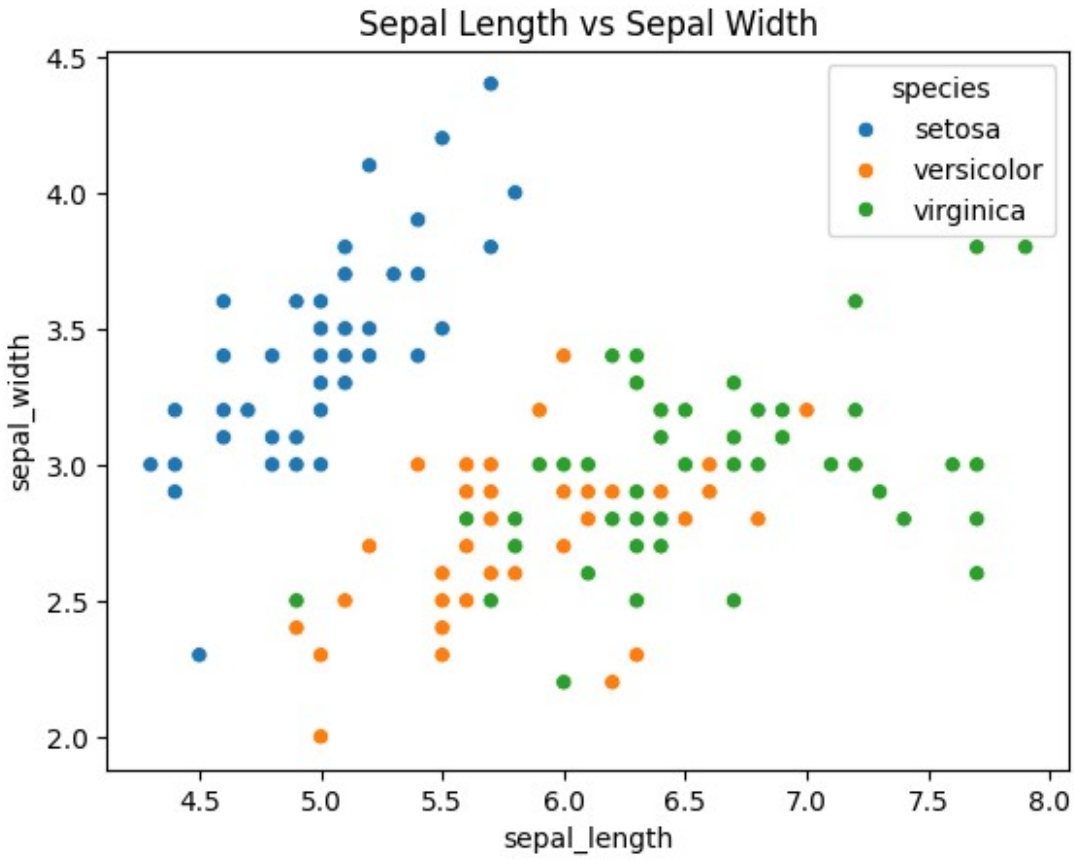
Pie Chart

```
# Pie chart: Species distribution
fig = px.pie(df, names="species", title="Species Distribution")
fig.show()
```

Scatter Plot: Sepal Length vs Sepal Width

Scatter plots show relationships between two numerical variables.

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

# Scatter plot: Sepal length vs sepal width
sns.scatterplot(x="sepal_length", y="sepal_width", hue="species",
data=iris)
plt.title("Sepal Length vs Sepal Width")
plt.show()
```

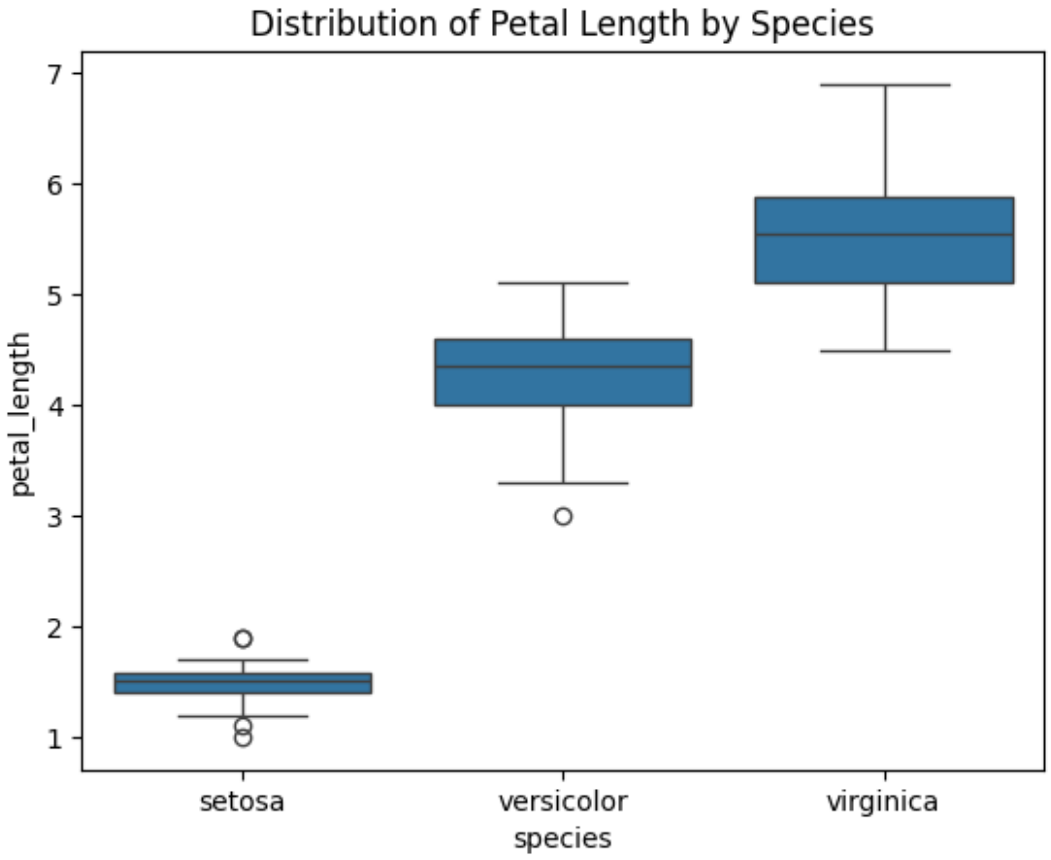


Box Plot: Distribution of Petal Length

Box plots help analyze distributions of numerical data.

```
# Box plot: Petal length by species
sns.boxplot(x="species", y="petal_length", data=iris)
```

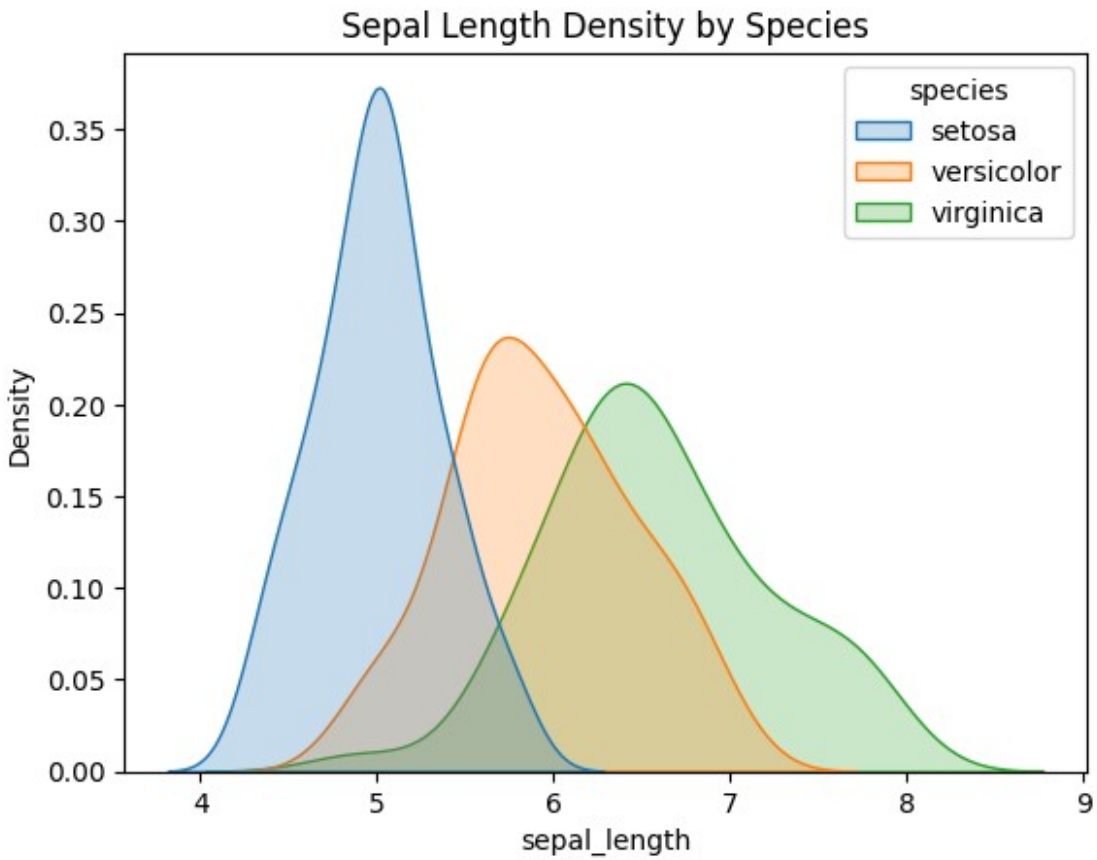
```
plt.title("Distribution of Petal Length by Species")
plt.show()
```



KDE Plot: Sepal Length Density

Kernel Density Estimation (KDE) plots visualize the probability density.

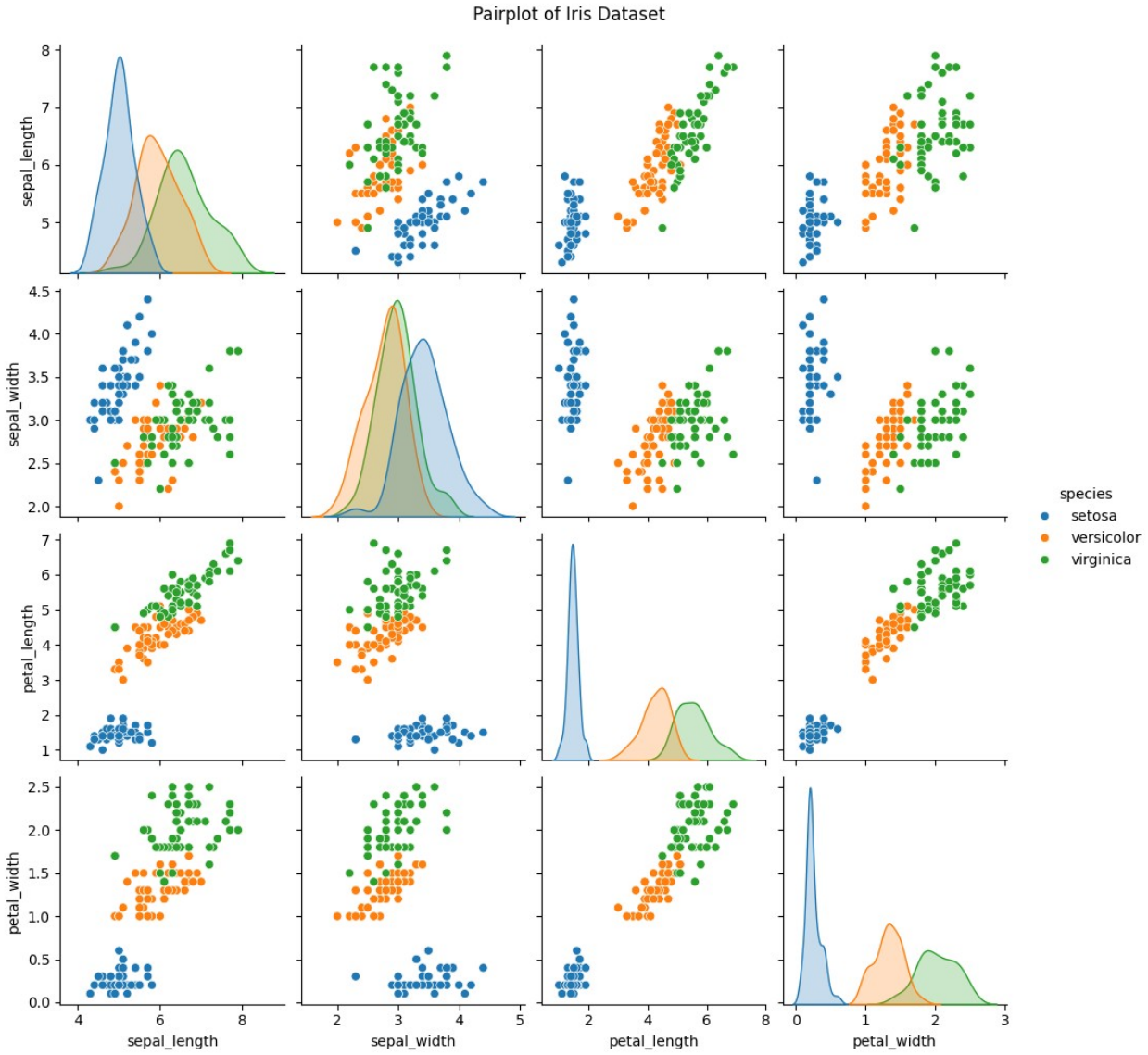
```
# KDE plot for Sepal length by species
sns.kdeplot(data=iris, x="sepal_length", hue="species", fill=True)
plt.title("Sepal Length Density by Species")
plt.show()
```



Pairplot: Relationships in Iris Dataset

Pair plots allow a quick overview of relationships among variables.

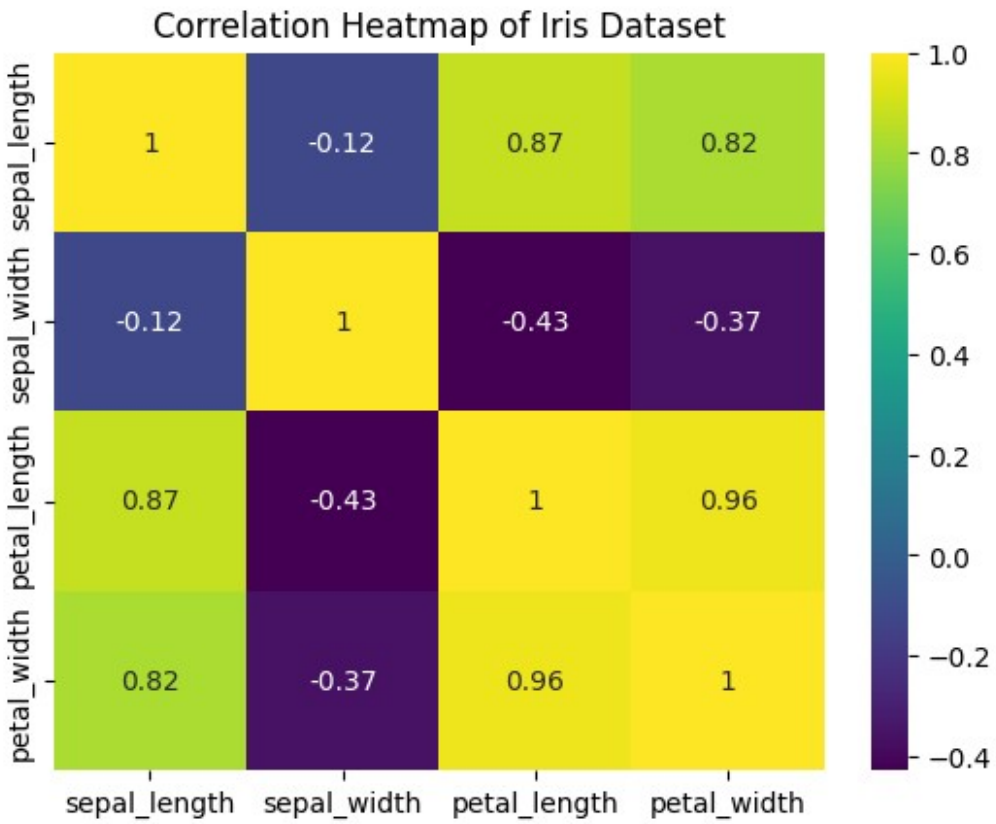
```
# Pairplot of iris dataset
sns.pairplot(iris, hue="species")
plt.suptitle("Pairplot of Iris Dataset", y=1.02)
plt.show()
```



Heatmap: Correlation Between Numerical Variables

This shows numerical relationships and their strength.

```
# Correlation matrix
correlation_iris = iris.drop("species", axis=1).corr()

# Heatmap
sns.heatmap(correlation_iris, annot=True, cmap="viridis")
plt.title("Correlation Heatmap of Iris Dataset")
plt.show()
```



Student Guidelines

- Use Comments: Add comments to explain your code.
- Experiment: Try changing the axes, colors, and other parameters to make the plots more informative.
- Combine Seaborn and Plotly: Use both libraries for the same dataset to compare their features.
- Submit Code and Outputs: Provide the Python script and screenshots of your outputs.

Seaborn Classwork

Dataset: penguins

The penguins dataset contains information about penguin species, their sizes, and island habitats.

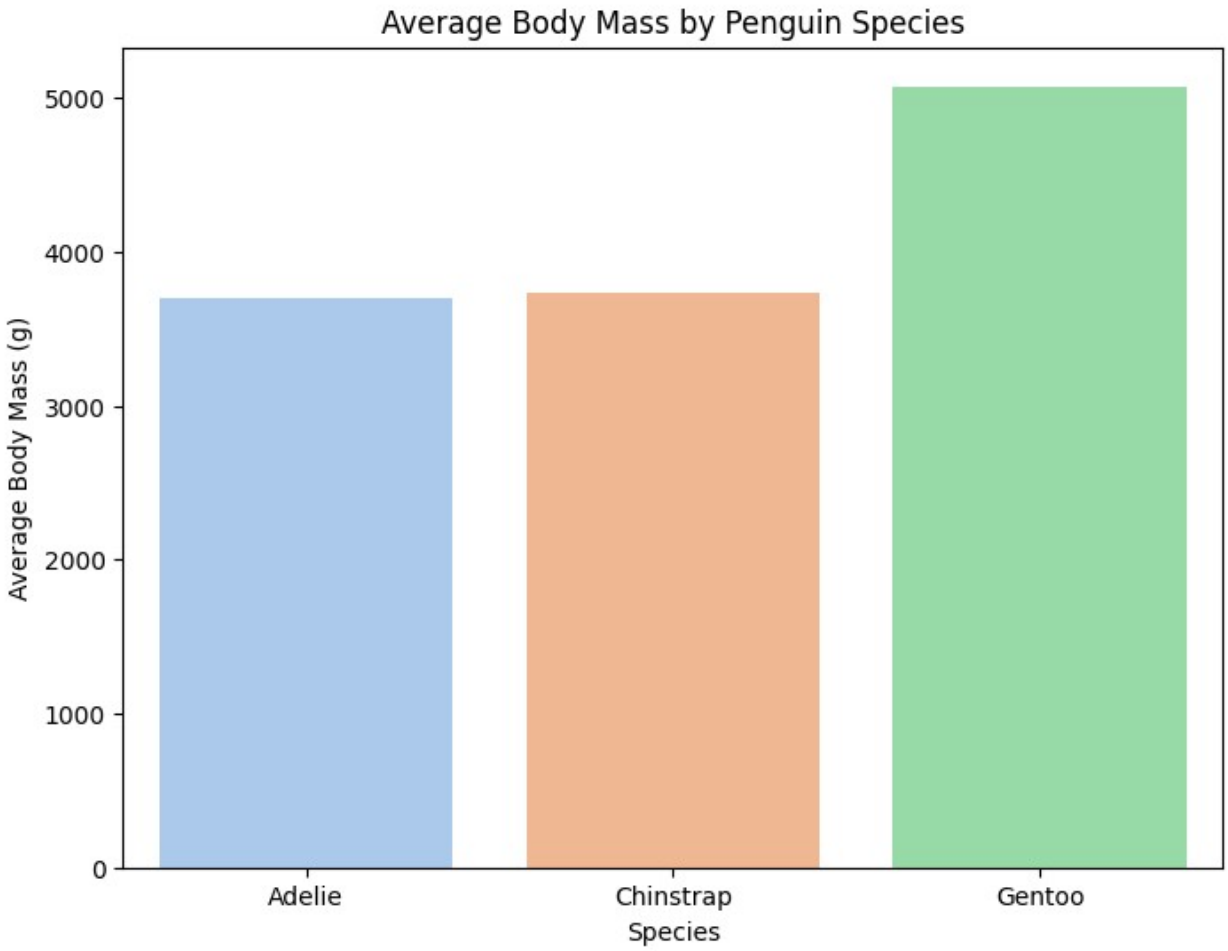
Tasks:

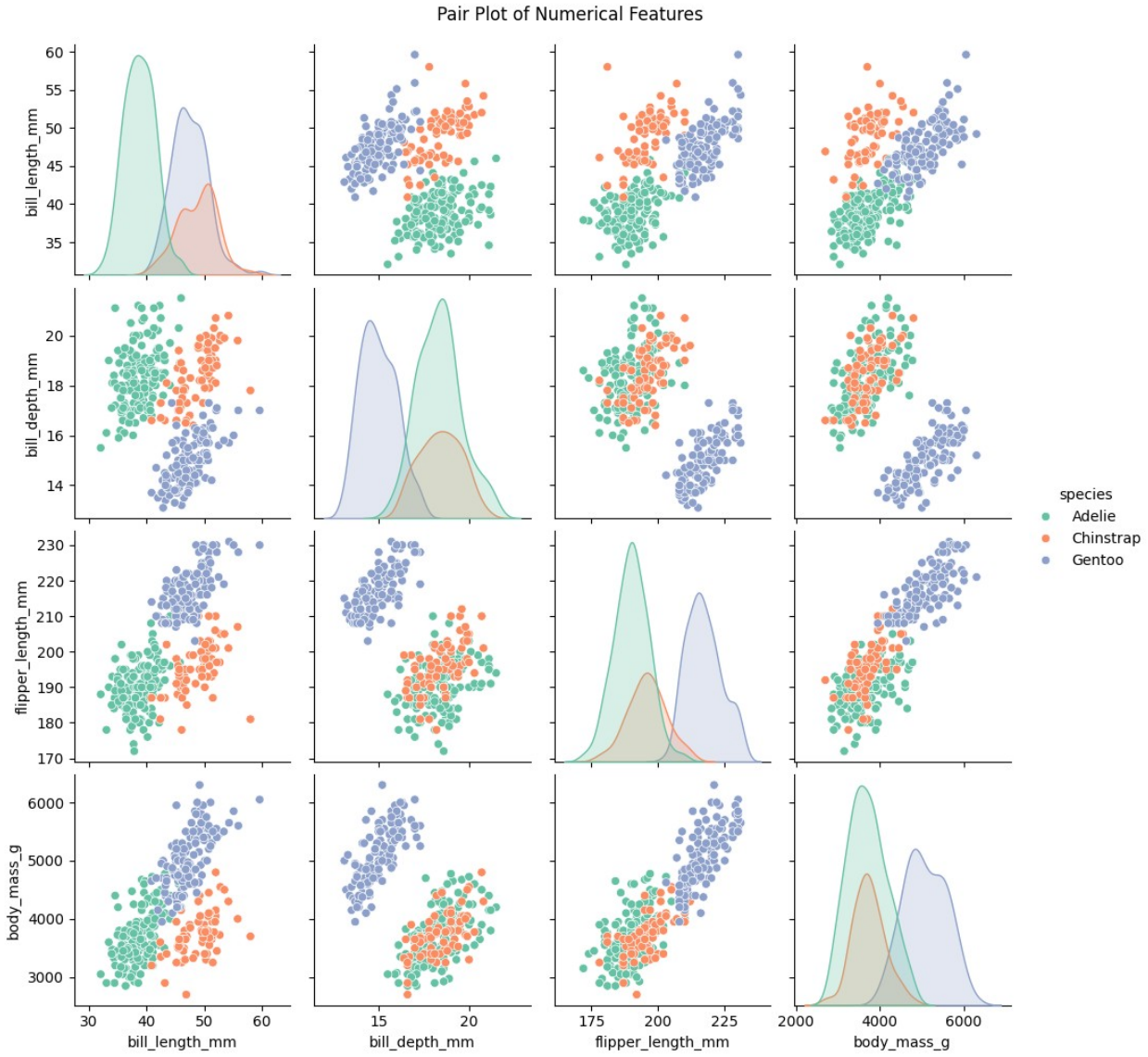
- Bar Plot: Create a bar plot showing the average body mass for each penguin species.
- Pairplot: Create a pair plot showing relationships between numerical features of the penguins dataset, categorized by species.

- Box Plot: Create a box plot showing the distribution of flipper length for each species.

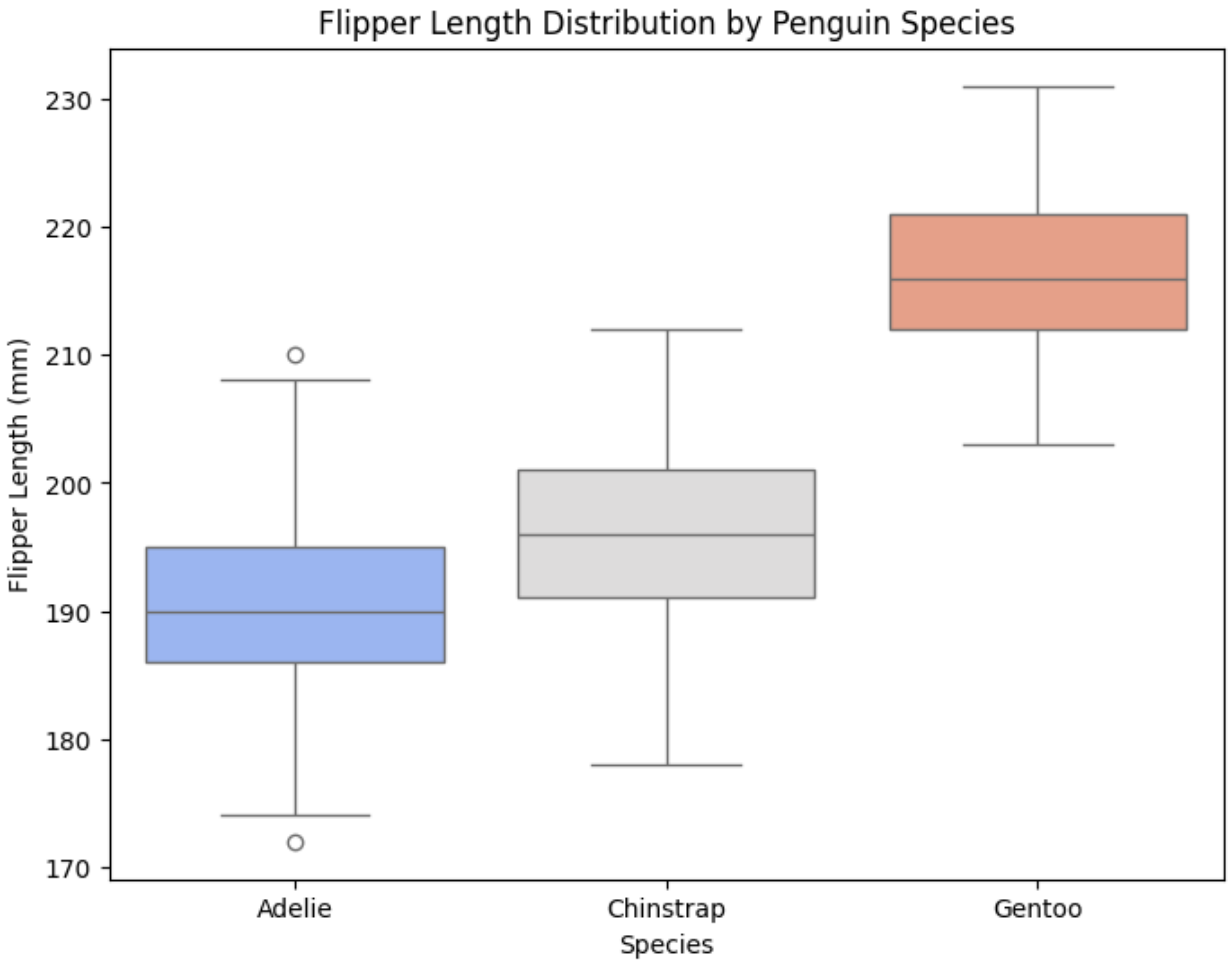
```
import seaborn as sns
import matplotlib.pyplot as plt

# Load penguins dataset
penguins = sns.load_dataset("penguins")

# Bar Plot: Average body mass for each penguin species
plt.figure(figsize=(8, 6))
sns.barplot(x="species", y="body_mass_g", data=penguins, ci=None,
palette="pastel")
plt.title("Average Body Mass by Penguin Species")
plt.ylabel("Average Body Mass (g)")
plt.xlabel("Species")
plt.show()

# Pair Plot: Relationships between numerical features, categorized by
species
sns.pairplot(penguins, hue="species", diag_kind="kde", palette="Set2")
plt.suptitle("Pair Plot of Numerical Features", y=1.02)
plt.show()

# Box Plot: Distribution of flipper length for each species
plt.figure(figsize=(8, 6))
sns.boxplot(x="species", y="flipper_length_mm", data=penguins,
palette="coolwarm")
plt.title("Flipper Length Distribution by Penguin Species")
plt.ylabel("Flipper Length (mm)")
plt.xlabel("Species")
plt.show()
```

```
<ipython-input-1-bbd09daf7fb0>:9: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same
effect.

  sns.barplot(x="species", y="body_mass_g", data=penguins, ci=None,
palette="pastel")
<ipython-input-1-bbd09daf7fb0>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x="species", y="body_mass_g", data=penguins, ci=None,
palette="pastel")
```


Average Body Mass by Penguin Species
Average Body Mass (g)
0
1000
2000
3000
4000
5000
Adelie
Chinstrap
Gentoo
Species




```
<ipython-input-1-bbd09daf7fb0>:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x="species", y="flipper_length_mm", data=penguins,
palette="coolwarm")
```



Plotly Classwork

Dataset: gapminder

The gapminder dataset contains information about countries, their GDP, life expectancy, and population over time.

Tasks:

- Scatter Plot: Create an interactive scatter plot showing the relationship between GDP per capita and life expectancy for the year 2007.
- Bar Plot: Create a bar chart showing the total population for each continent in 2007.
- 3D Plot: Create a 3D scatter plot to visualize GDP per capita, life expectancy, and population for the year 2007.
- Line Plot: Create a line plot showing the evolution of life expectancy over time for a selected country (e.g., India,UAE,Germany,China,Australia,Russia).

```
import plotly.express as px
import plotly.graph_objects as go
```

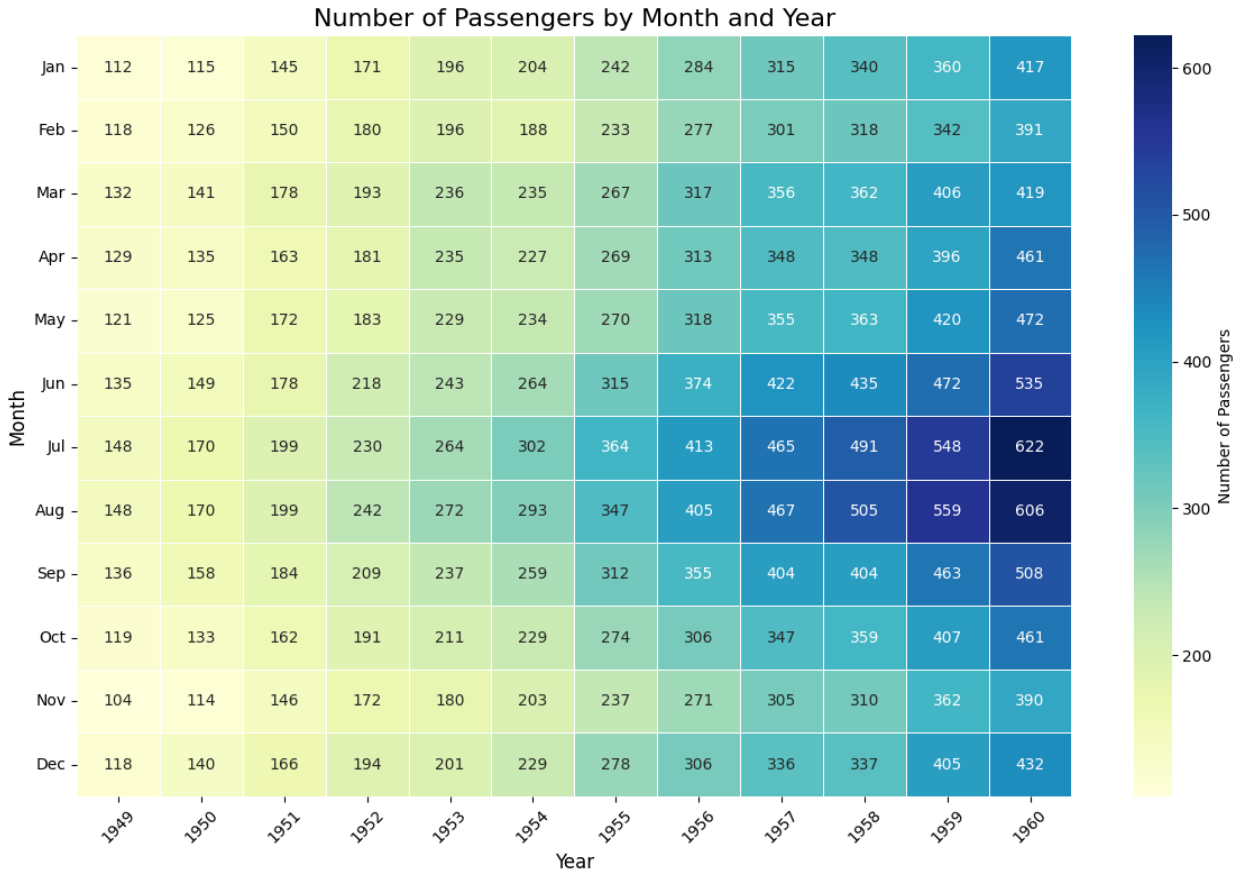
```
# Load gapminder dataset
gapminder = px.data.gapminder()

# Interactive Scatter Plot
scatter_plot = px.scatter(
    gapminder[gapminder['year'] == 2007],
    x="gdpPercap",
    y="lifeExp",
    size="pop",
    color="continent",
    hover_name="country",
    title="GDP per Capita vs Life Expectancy (2007)",
    labels={"gdpPercap": "GDP per Capita", "lifeExp": "Life
Expectancy"},
    log_x=True,
    size_max=60
)
scatter_plot.show()

# Bar Plot
bar_plot = px.bar(
    gapminder[gapminder['year'] == 2007],
    x="continent",
    y="pop",
    color="continent",
    title="Total Population by Continent (2007)",
    labels={"pop": "Population"},
    text_auto=True
)
bar_plot.show()

# 3D Scatter Plot
scatter_3d = px.scatter_3d(
    gapminder[gapminder['year'] == 2007],
    x="gdpPercap",
    y="lifeExp",
    z="pop",
    color="continent",
    size="pop",
    hover_name="country",
    title="3D Scatter Plot: GDP per Capita, Life Expectancy, and
Population (2007)",
    labels={"gdpPercap": "GDP per Capita", "lifeExp": "Life
Expectancy", "pop": "Population"},
    log_x=True,
    size_max=60
)
scatter_3d.show()
```

```
# Line Plot for Selected Countries
countries = ["India", "UAE", "Germany", "China", "Australia",
"Russia"]
line_plot = px.line(
    gapminder[gapminder["country"].isin(countries)],
    x="year",
    y="lifeExp",
    color="country",
    title="Evolution of Life Expectancy Over Time",
    labels={"lifeExp": "Life Expectancy", "year": "Year"},
    markers=True
)
line_plot.show()
```

Load the **flights** dataset from Seaborn. Create a heatmap to show the number of passengers for each month and year. Customize the heatmap with labels and a color bar.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load the flights dataset
flights = sns.load_dataset('flights')

# Pivot the dataset to create a matrix for the heatmap
flights_pivot = flights.pivot(index="month", columns="year",
values="passengers")

# Set the figure size
plt.figure(figsize=(12, 8))

# Create the heatmap
sns.heatmap(
    flights_pivot,
    annot=True,          # Show the numbers on the heatmap
    fmt="d",             # Format as integers
    cmap="YlGnBu",       # Color palette
    linewidths=0.5,      # Add lines between cells
    cbar_kws={'label': 'Number of Passengers'}  # Label for the color
bar
)

# Customize labels and title
plt.title("Number of Passengers by Month and Year", fontsize=16)
```

```
plt.xlabel("Year", fontsize=12)
plt.ylabel("Month", fontsize=12)
plt.xticks(rotation=45)  # Rotate the year labels for better
readability
plt.yticks(rotation=0)   # Keep month labels horizontal

# Display the plot
plt.tight_layout()
plt.show()
```



Load the **tips** dataset. First, create a Seaborn boxplot to show the distribution of total_bill across different days of the week. Then, create an interactive Plotly pie chart showing the percentage contribution of each day to the total number of records.

```
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

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

# Create a boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(x="day", y="total_bill", data=tips, palette="Set2")

# Customize the plot
plt.title("Distribution of Total Bill Across Different Days",
fontsize=16)
plt.xlabel("Day of the Week", fontsize=12)
plt.ylabel("Total Bill", fontsize=12)

# Display the plot
plt.tight_layout()
plt.show()


# Group the data by day and count the number of records
day_counts = tips['day'].value_counts().reset_index()
day_counts.columns = ['day', 'count']

# Create a pie chart
fig = px.pie(
    day_counts,
    names='day',
    values='count',
    title="Percentage Contribution of Each Day to Total Records",
    color_discrete_sequence=px.colors.sequential.RdBu
)

# Show the pie chart
fig.show()
```

```
<ipython-input-7-10942d9174b9>:10: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.
```