VISUALIZATION

Visualization is a method of representing data visually. Data visualization involves the graphical or pictorial representation of information, which helps in understanding complex data more easily. It enables users to quickly identify patterns, trends, and outliers within large datasets.

In Python, there are several popular libraries used for data visualization, including Matplotlib, Seaborn, and Plotly.

Let us explore Plotly and Seaborn in detail:

PLOTLY

Plotly, developed by Plotly Inc., is a graphing library that enables the creation of **interactive, publication-quality visualizations** for the web. Plotly can be used in Python, R, JavaScript, and MATLAB. In Python, it provides two primary interfaces:

- plotly.graph_objects — low-level API with full customization.
- plotly.express — high-level API for rapid and concise plotting.

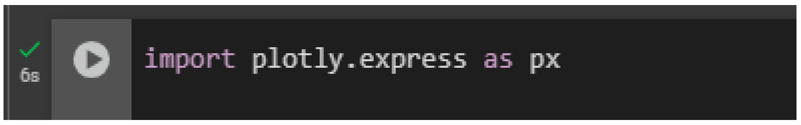
It is especially useful for building web dashboards and interactive visual analytics.

1. **Figure**
   The main object that holds the data (data) and layout (layout) for the plot. Created using go.Figure() or returned by Plotly Express.
2. **Data (Traces)**
   A trace represents a single series of data (e.g., go.Scatter, go.Bar, go.Pie). Multiple traces can be added to a figure.
3. **Layout**
   Defines the styling and structure of the figure, such as titles, axes, legend, margin, background, etc.
4. **Interaction**
   Includes hover tooltips, zoom/pan, and dynamic legends, enabled by default.

Plotly Express

plotly.express is similar in spirit to matplotlib.pyplot. It simplifies plotting by automatically inferring axes, legends, and colors based on pandas DataFrames.

We typically import Plotly as:

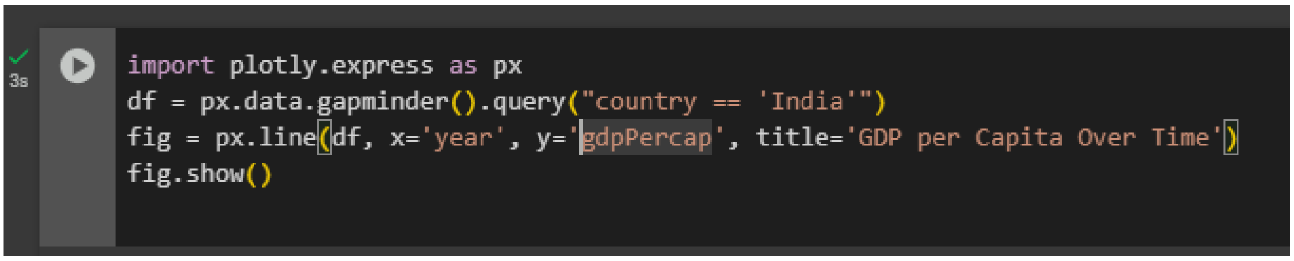


```
import plotly.express as px
```

LINE CHART

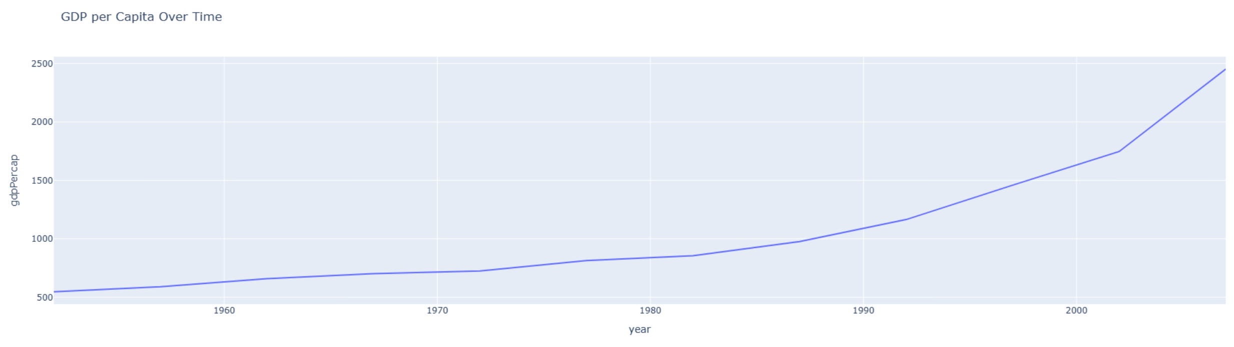
Used to display trends over time or continuous data. Helpful in time series and trend analysis.

Code snippet:



```
import plotly.express as px
df = px.data.gapminder().query("country == 'India'")
fig = px.line(df, x='year', y='gdpPercap', title='GDP per Capita Over Time')
fig.show()
```

Output:



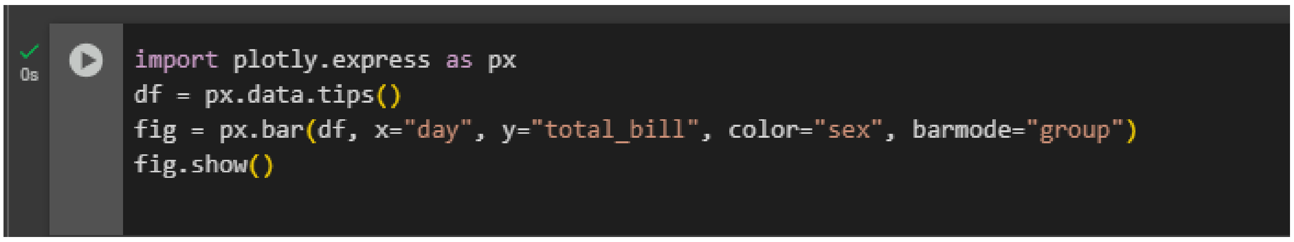
Description:

- x: time variable (years)
- y: continuous variable (GDP per capita)
- Interactive tooltip and zoom enabled.

BARGRAPH

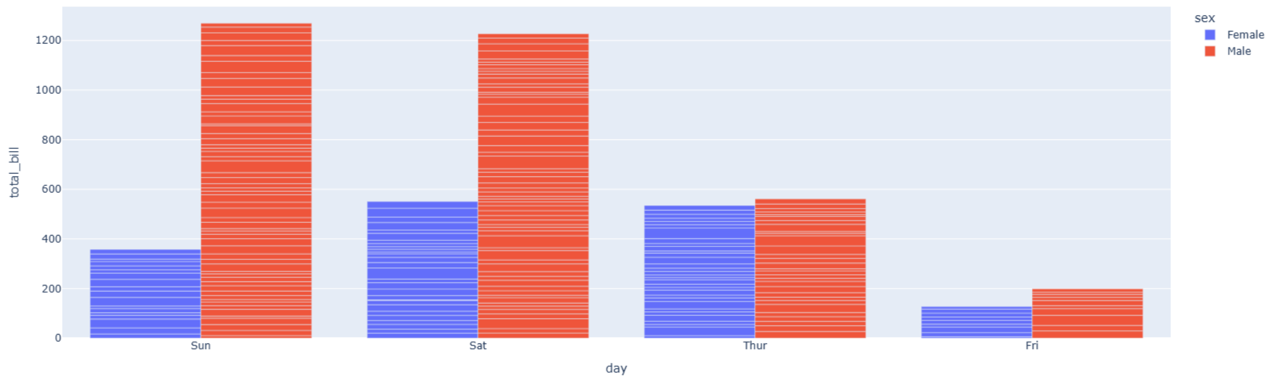
Used for comparing values across categories. Can be grouped or stacked.

Code snippet:



```
import plotly.express as px
df = px.data.tips()
fig = px.bar(df, x="day", y="total_bill", color="sex", barmode="group")
fig.show()
```

Output:



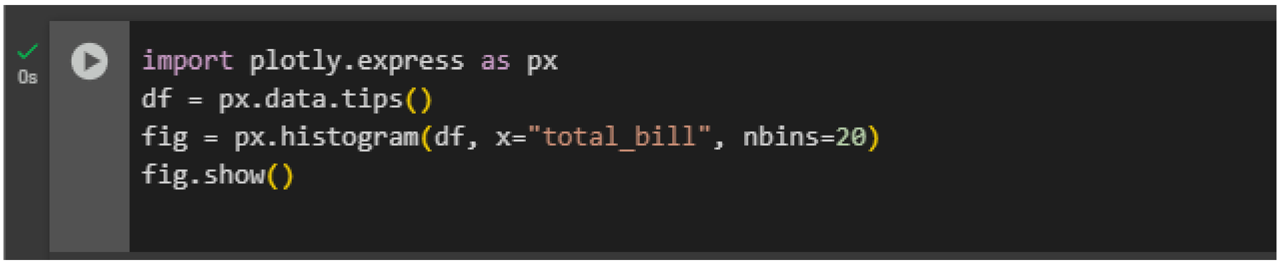
Description:

- barmode="group" places bars side-by-side.
- Total bill amounts are compared across days and sex.

HISTOGRAM

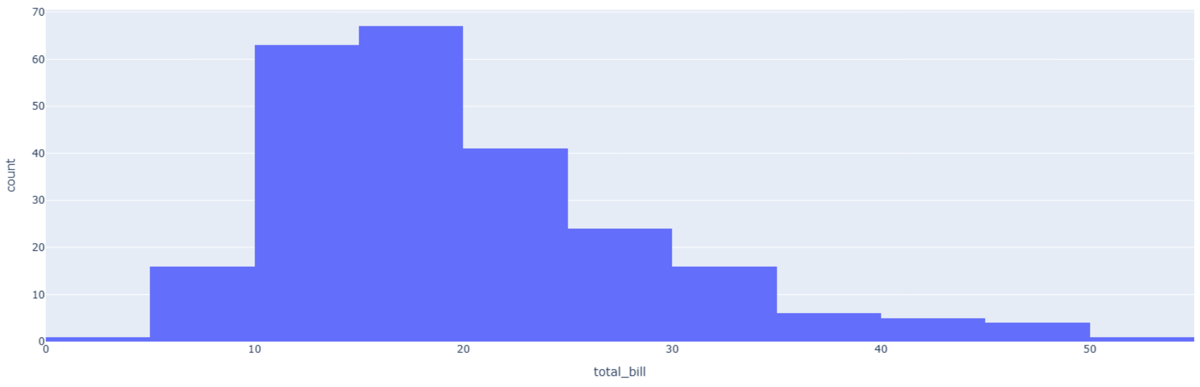
Displays distribution of a single variable by grouping into bins.

Code snippet:



```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", nbins=20)
fig.show()
```

Output:



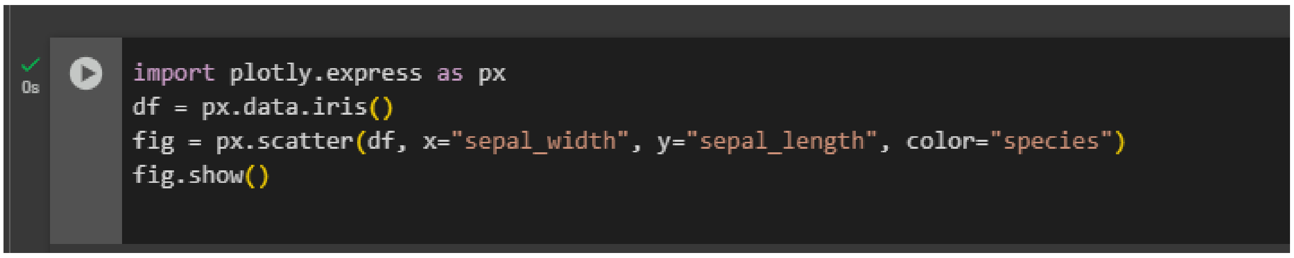
Description:

- x: variable to distribute.
- nbins: number of bins for grouping

SCATTER PLOT

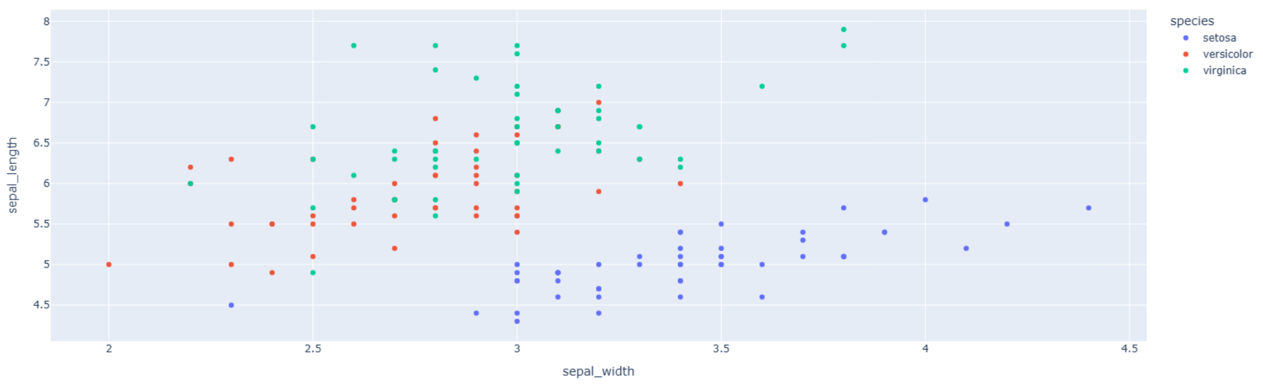
Used to examine relationships between two variables. Helpful in spotting clusters, trends, or correlations.

Code snippet:



```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species")
fig.show()
```

Output:



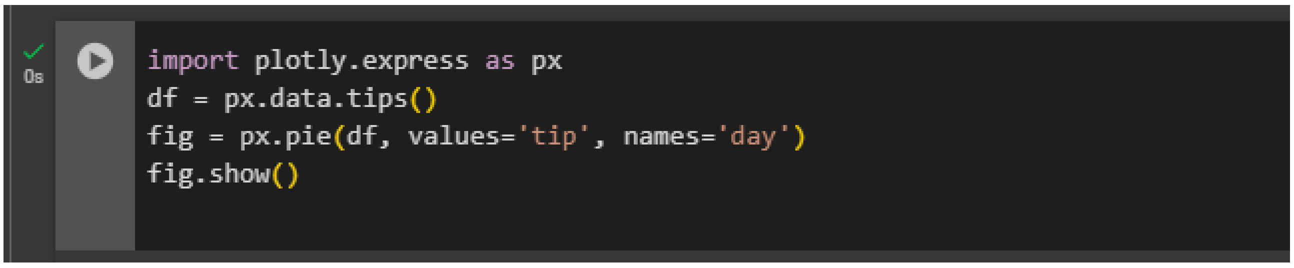
Description:

- Shows sepal width vs. length.
- Colors denote species for clustering.

PIECHART

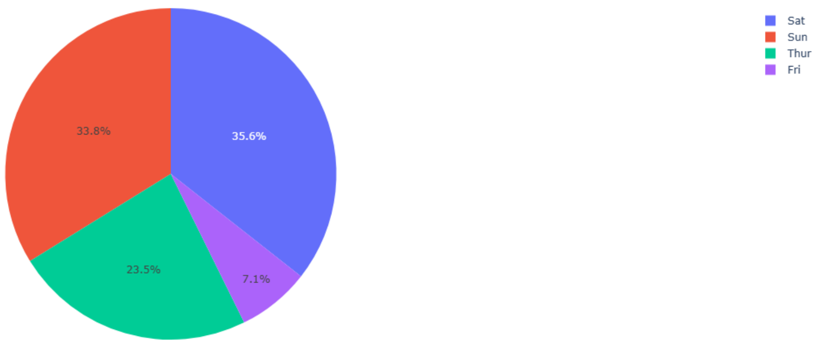
Displays part-to-whole relationships. Common for percentages and proportions.

Code snippet:



```
import plotly.express as px
df = px.data.tips()
fig = px.pie(df, values='tip', names='day')
fig.show()
```

Output:



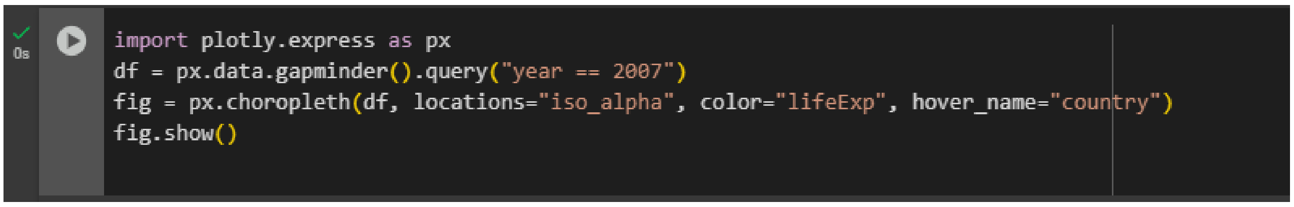
Description:

- Slices show total tips contributed by each day.
- Useful for overall category share.

CHOROPLETH MAP

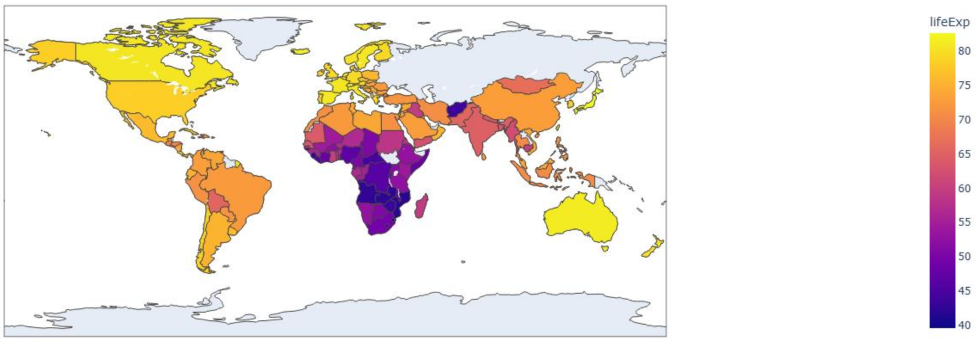
Used for geographical data visualization by coloring regions based on values.

Code snippet:



```python
import plotly.express as px
df = px.data.gapminder().query("year == 2007")
fig = px.choropleth(df, locations="iso_alpha", color="lifeExp", hover_name="country")
fig.show()
```

Output:



Description:

- World map showing life expectancy by country in 2007.
- locations: country ISO codes.

SEABORN

Seaborn is a Python data visualization library based on **Matplotlib**. It was developed to provide a high-level interface for drawing **statistical graphics**. It is closely integrated with **Pandas**, which makes it ideal for data analysis and exploration tasks.

Seaborn handles many **common data visualization needs** with less code and better aesthetics compared to Matplotlib.

A Seaborn plot typically consists of:

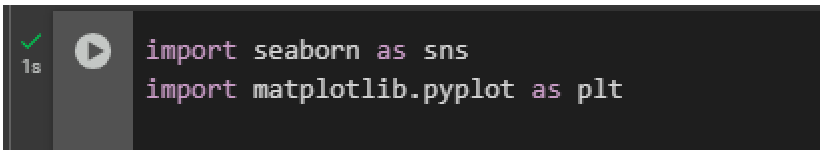
1. **Figure** – Same as in Matplotlib, it's the container of plots.
2. **Axes** – The individual plot elements where data is drawn.
3. **Data-aware functions** – Unlike Matplotlib, Seaborn directly accepts DataFrames and variable names.
4. **Themes/Color Palettes** – Built-in style settings for better visuals.

Here are some sample codes for some of the graphs.

SEABORN STRUCTURE & STYLE FEATURES

- Built on top of Matplotlib, inherits its backend.
- Uses sns.set_theme() to apply custom themes globally.
- Works seamlessly with **Pandas DataFrames**.
- Supports **aggregation and statistical estimation** (like confidence intervals).
- Useful for **Exploratory Data Analysis (EDA)**.

We import Seaborn typically as:



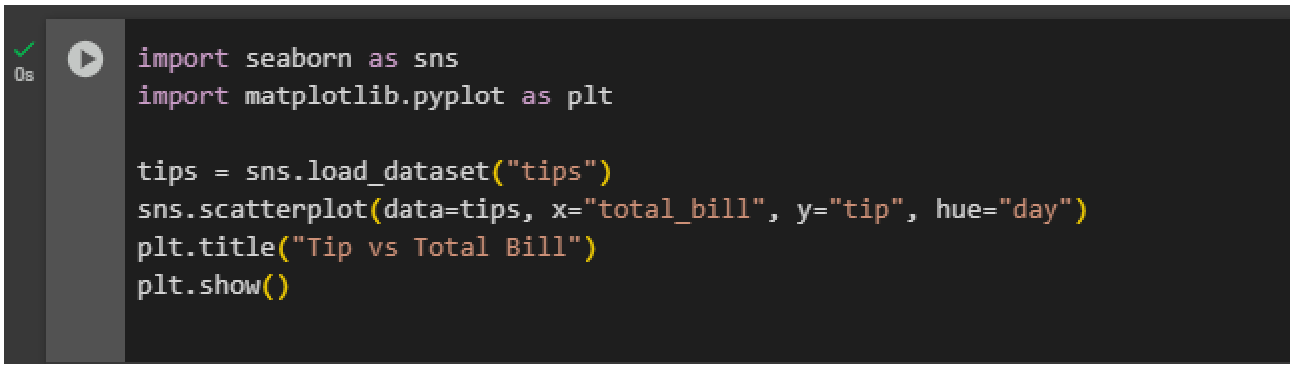
```
import seaborn as sns
import matplotlib.pyplot as plt
```

SCATTER PLOT

Scatter plots visualize relationships between two continuous variables.

Function: sns.scatterplot()

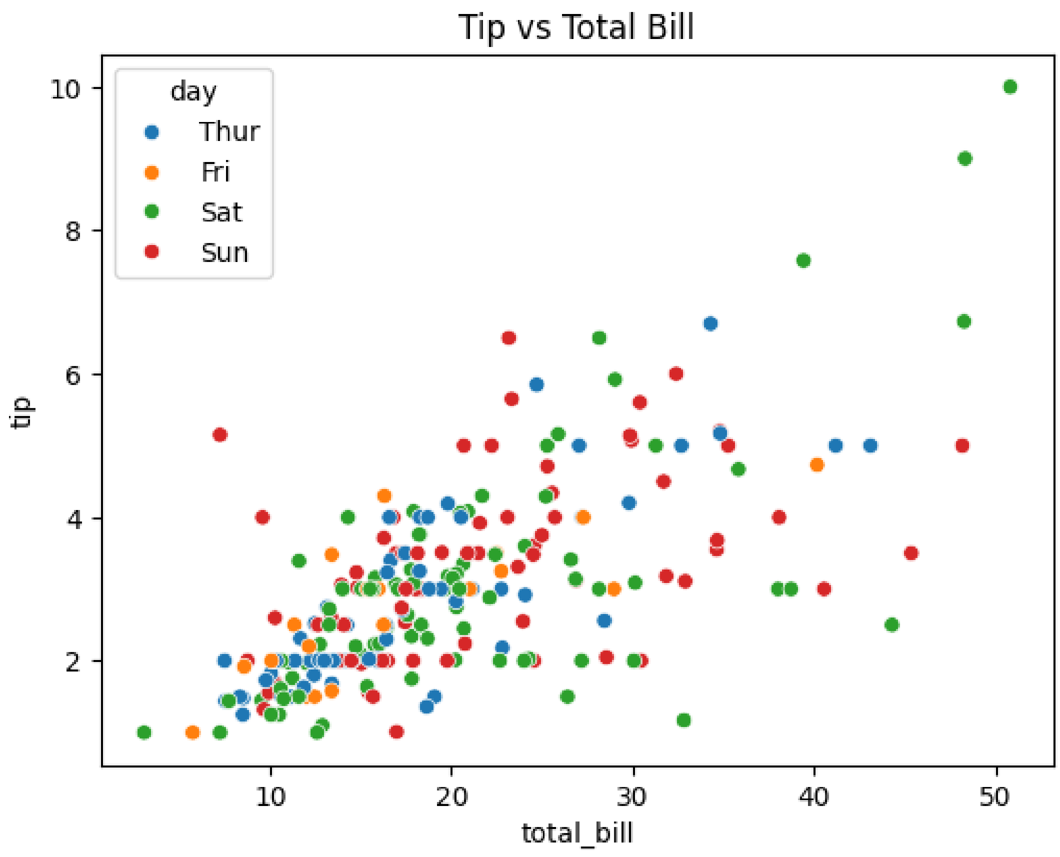
Code snippet:



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

tips = sns.load_dataset("tips")
sns.scatterplot(data=tips, x="total_bill", y="tip", hue="day")
plt.title("Tip vs Total Bill")
plt.show()
```

Output:



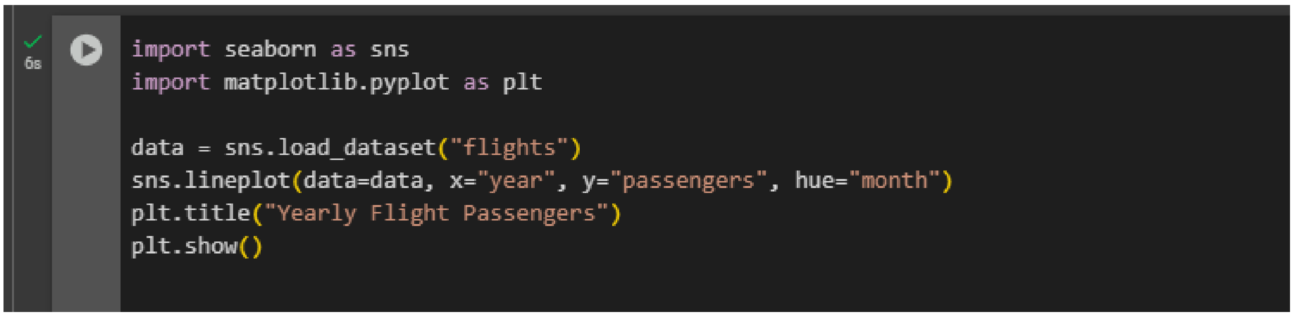
Description:

- hue adds color separation by category (like "day").
- Great for showing correlation or trend patterns.

LINE CHART

Line plots are used to show the trend of a variable over time or any sequential data.

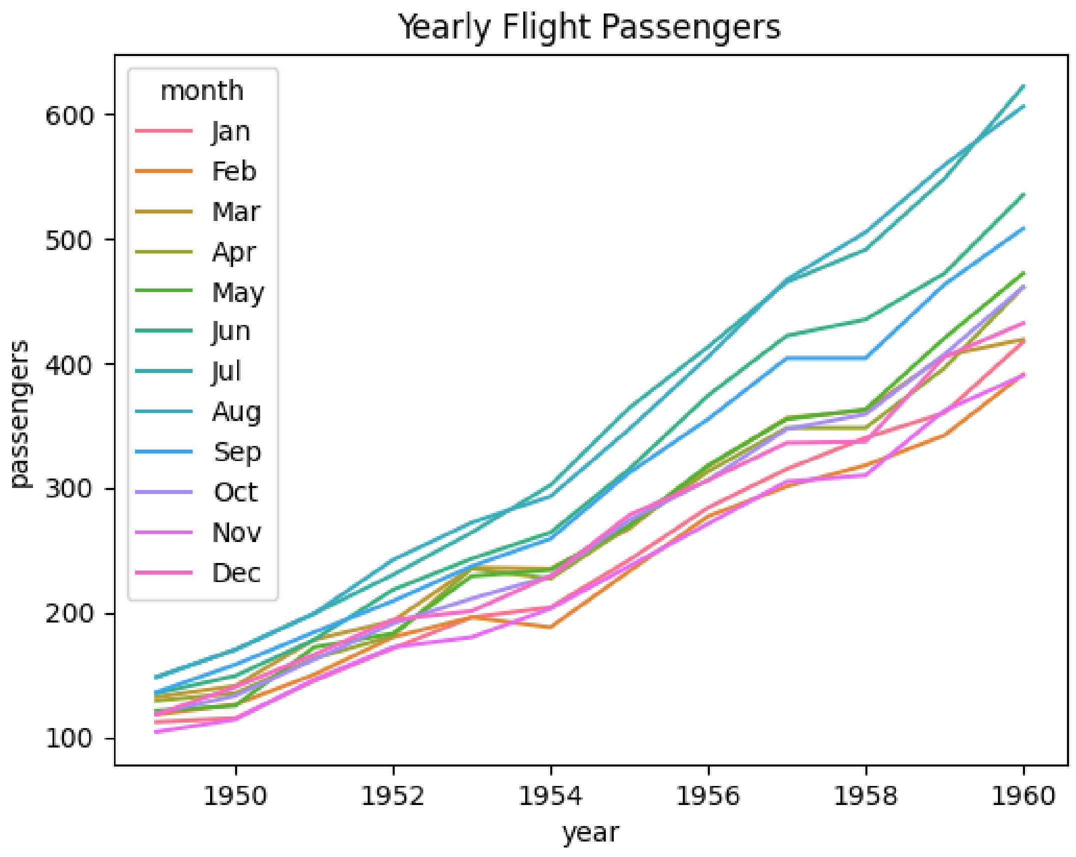
Code snippet:



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

data = sns.load_dataset("flights")
sns.lineplot(data=data, x="year", y="passengers", hue="month")
plt.title("Yearly Flight Passengers")
plt.show()
```

Output:



Description:

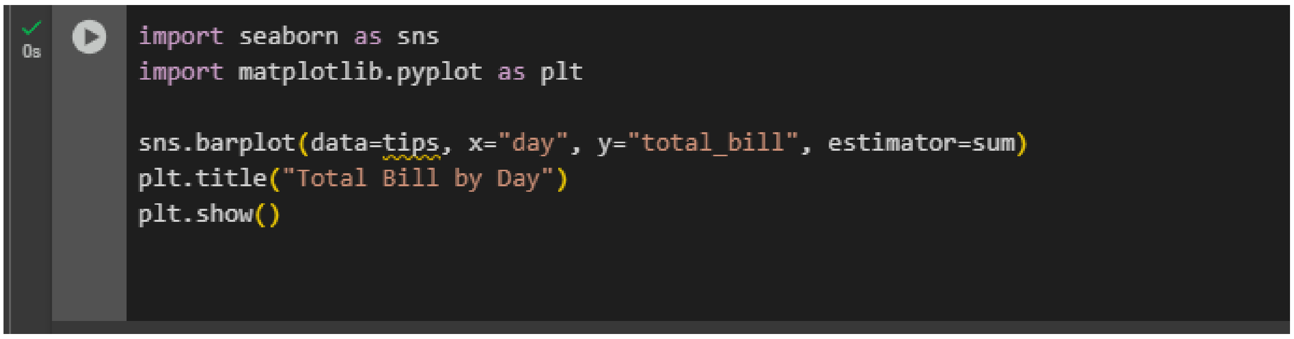
- lineplot() draws a line joining points across x and y.
- hue is used to split lines by category (e.g., month).
- Useful for time-series data visualization.

BAR PLOT

Bar plots compare quantities across categories.

Function: `sns.barplot()`

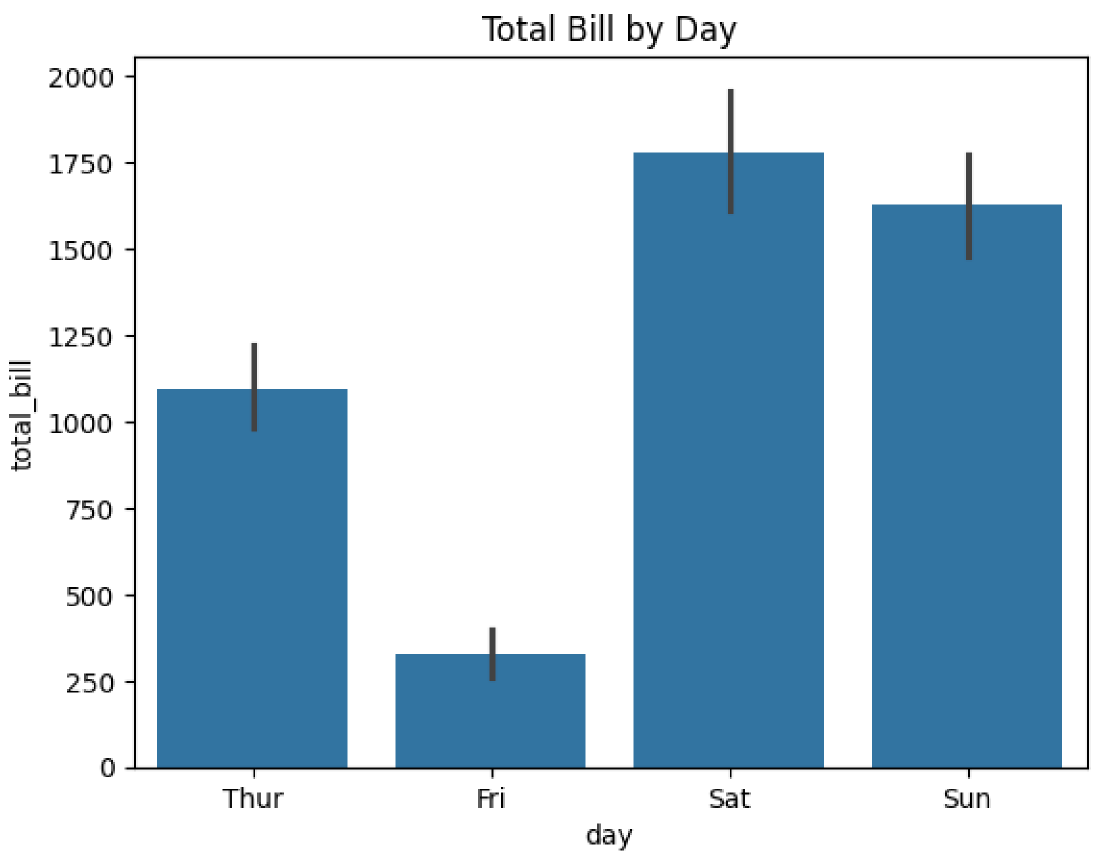
Code snippet:



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

sns.barplot(data=tips, x="day", y="total_bill", estimator=sum)
plt.title("Total Bill by Day")
plt.show()
```

Output:



Description:

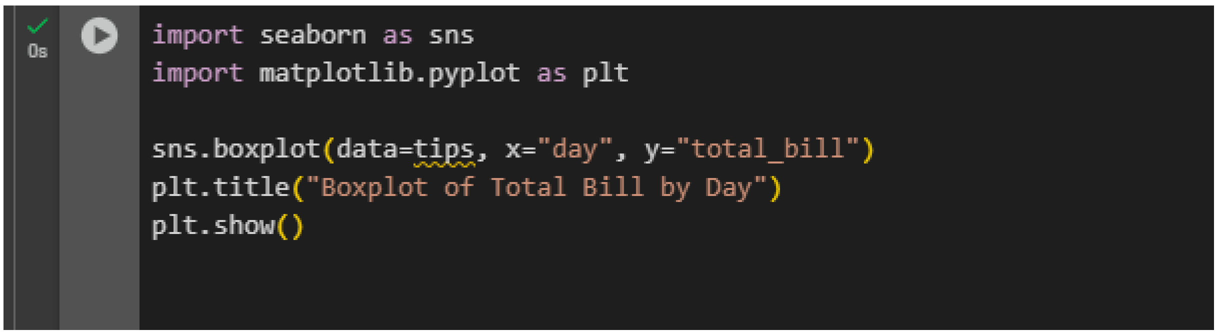
- estimator=sum aggregates the y-values for each category.
- By default, shows mean and confidence intervals.

BOX PLOT

Shows the **distribution** of quantitative data and identifies **outliers**.

Function: sns.boxplot()

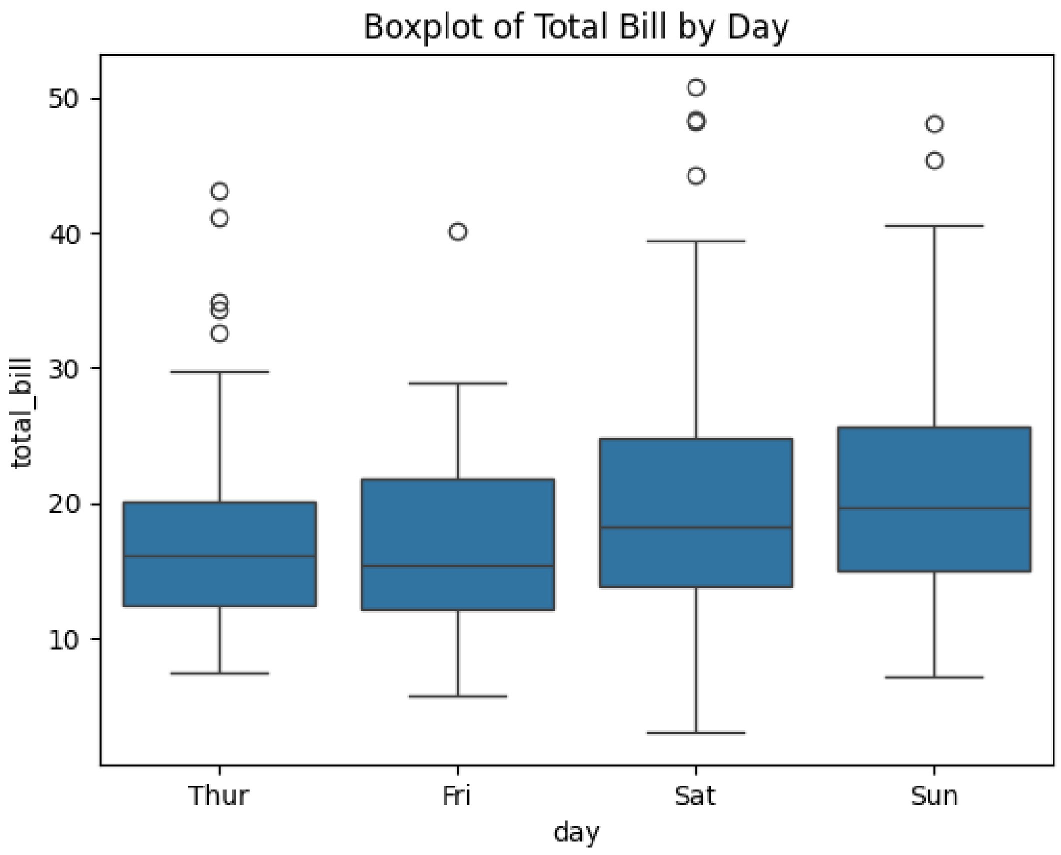
Code snippet:



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

sns.boxplot(data=tips, x="day", y="total_bill")
plt.title("Boxplot of Total Bill by Day")
plt.show()
```

Output:



Description:

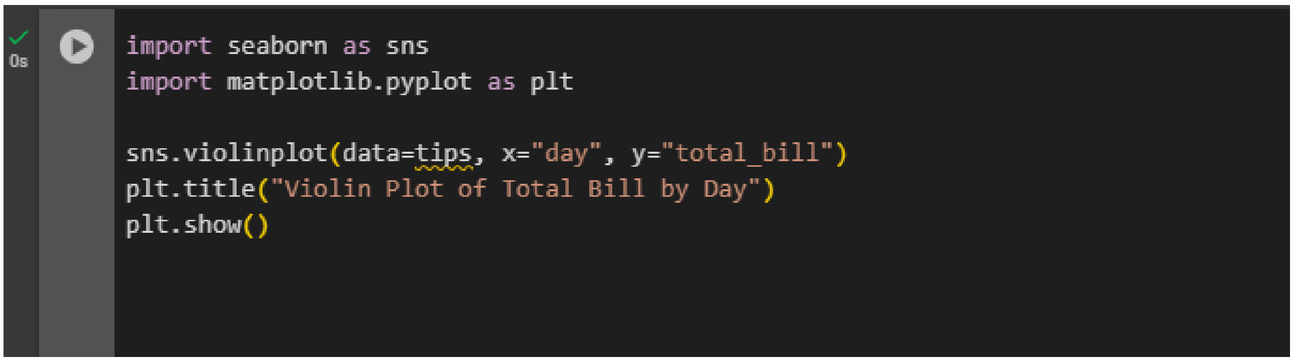
- Displays median, quartiles, and possible outliers.
- Useful for statistical comparison between groups.

VIOLIN PLOT

Combination of box plot and KDE (Kernel Density Estimation).

Function: `sns.violinplot()`

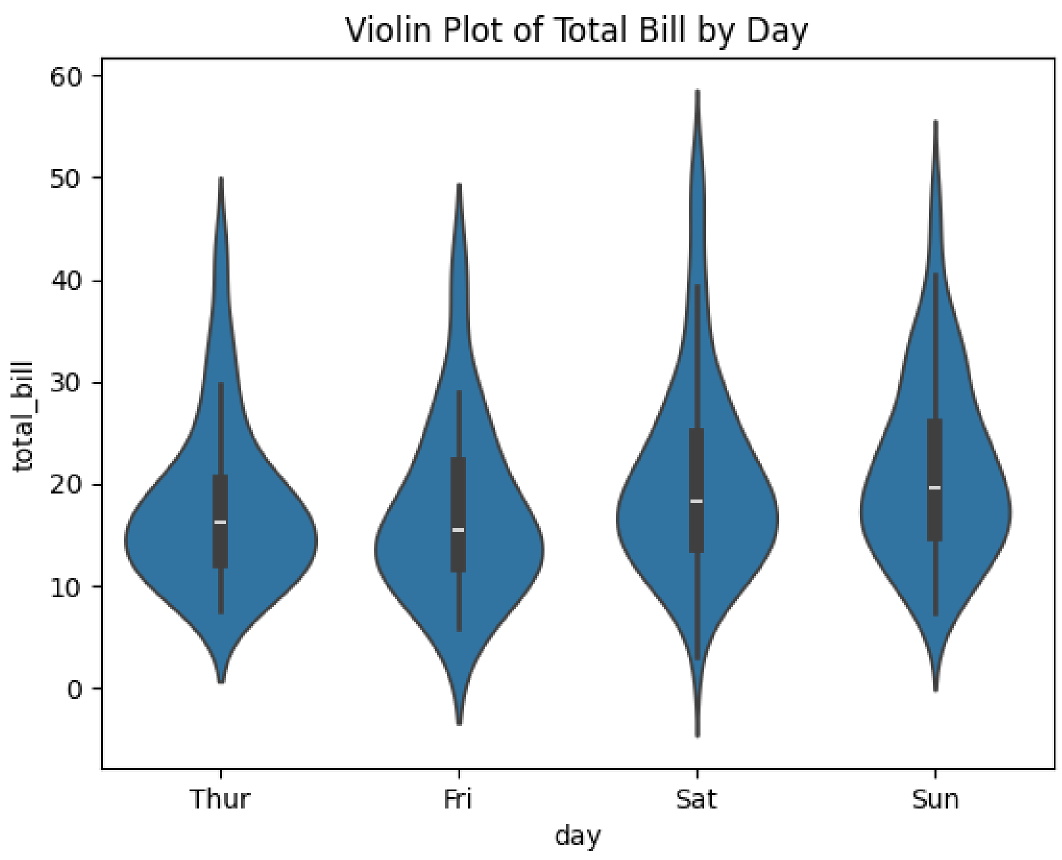
Code snippet:



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

sns.violinplot(data=tips, x="day", y="total_bill")
plt.title("Violin Plot of Total Bill by Day")
plt.show()
```

Output:



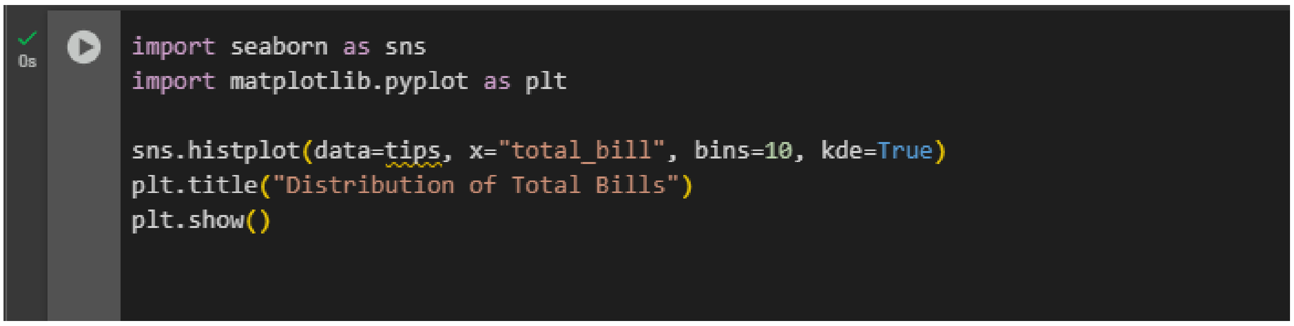
Description:

- Shows distribution and probability density.
- Good for comparing distribution shapes.

HISTOGRAM

Displays distribution of a single variable.

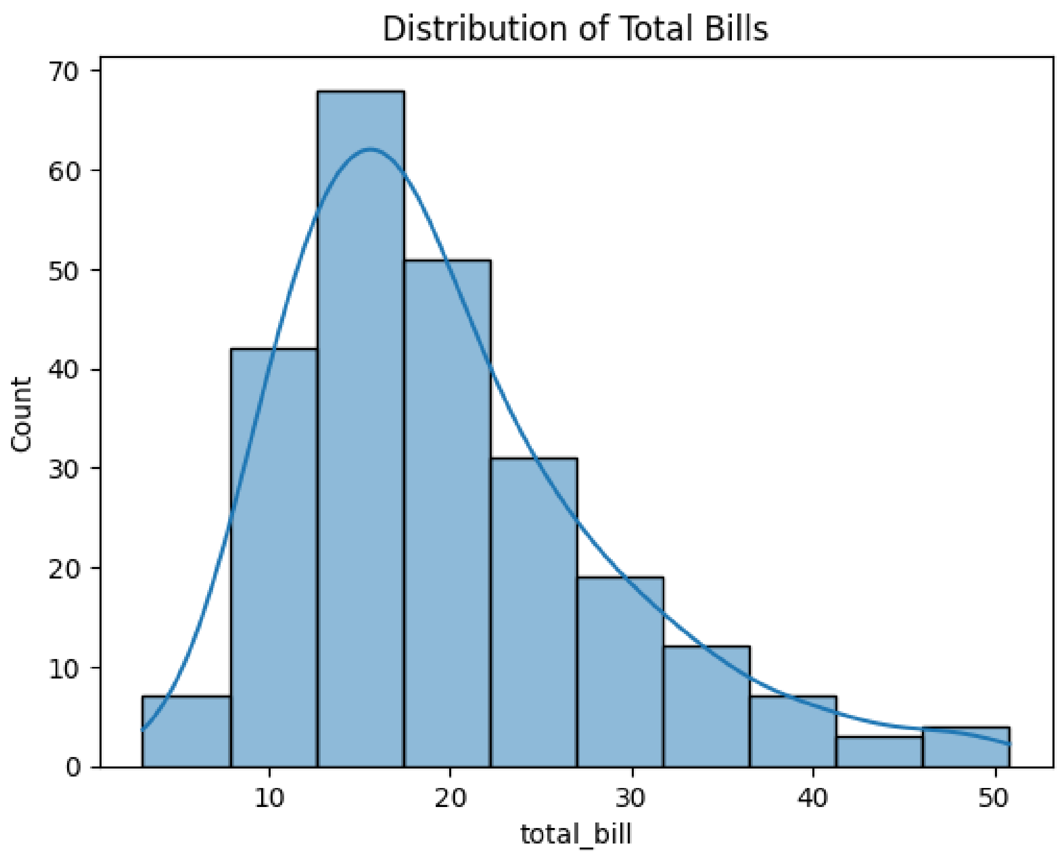
Function: `sns.histplot()`



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

sns.histplot(data=tips, x="total_bill", bins=10, kde=True)
plt.title("Distribution of Total Bills")
plt.show()
```

Output:



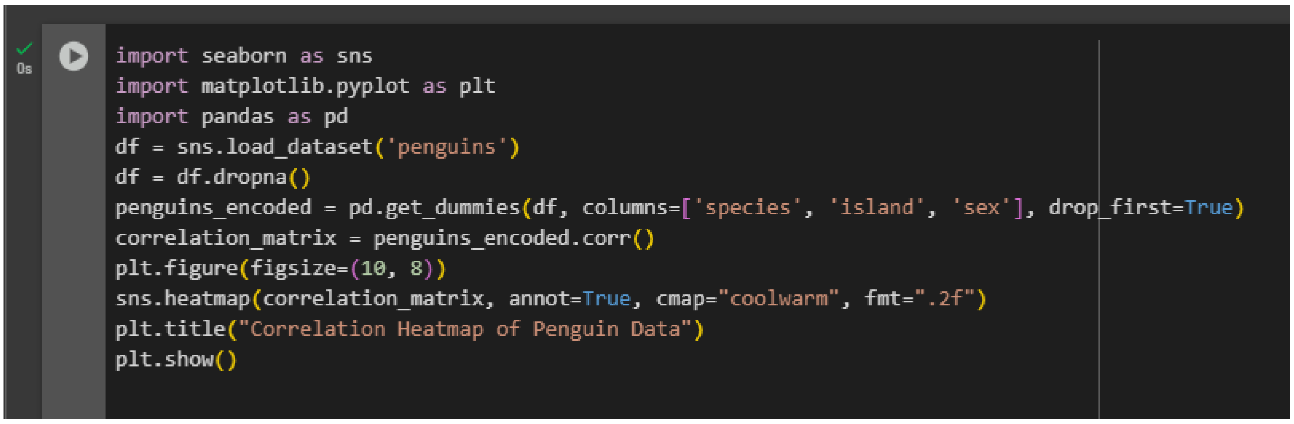
Description:

- kde=True overlays a density curve.
- Helps in understanding data spread and skewness.

CORRELATION HEATMAP

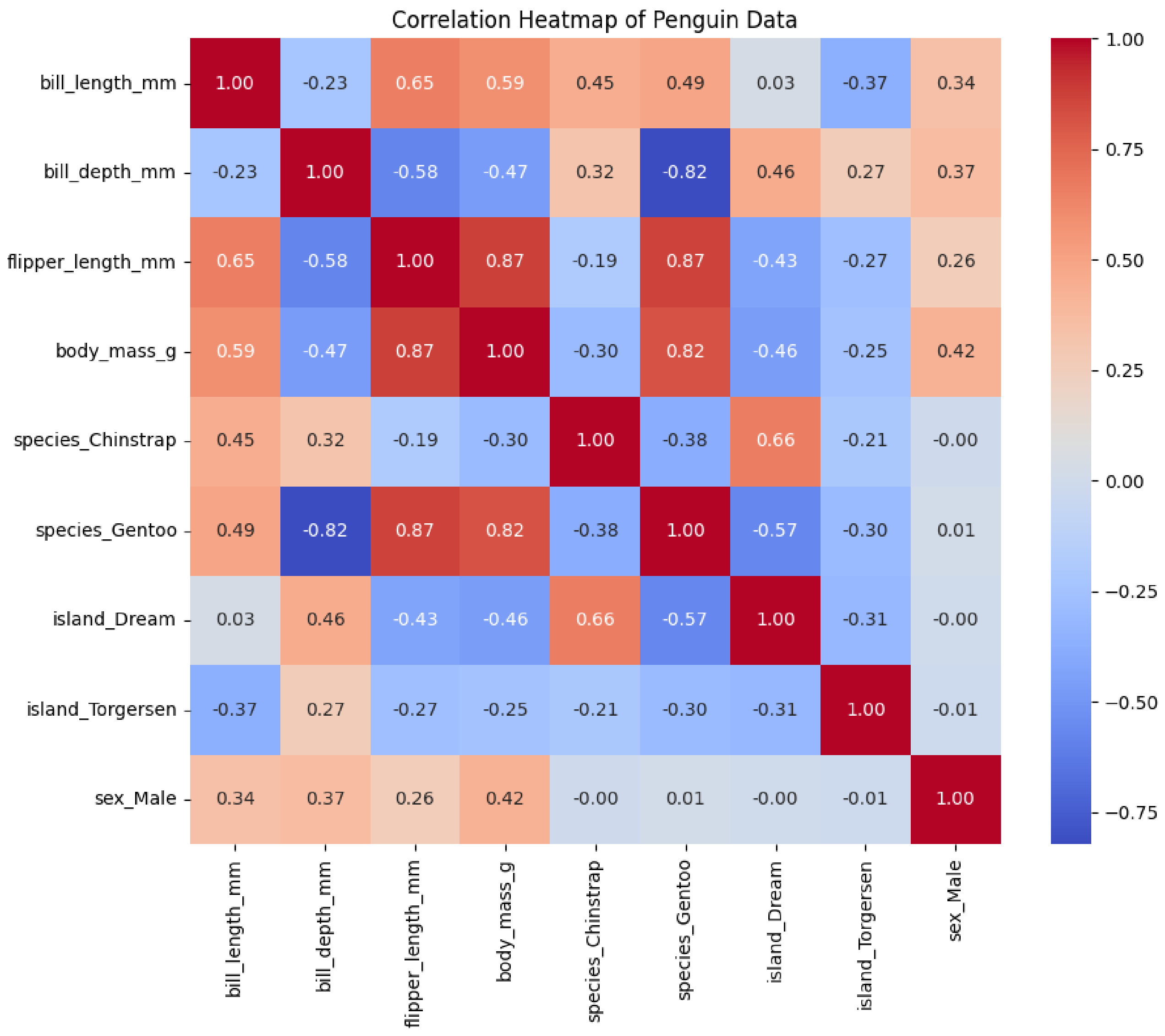
Shows how strongly variables are related.

Code snippet:



```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
df = sns.load_dataset('penguins')
df = df.dropna()
penguins_encoded = pd.get_dummies(df, columns=['species', 'island', 'sex'], drop_first=True)
correlation_matrix = penguins_encoded.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap of Penguin Data")
plt.show()
```

Output:



Description:

- annot=True shows numerical values.
- Color gradients represent correlation strength.
- Often used to check multicollinearity.

:

COMPARISON OF PLOTLY AND SEABORN

Plotly:

- **Interactive Visualization Library:**
  Plotly is a modern, open-source graphing library designed for interactive and web-based visualizations. It supports JavaScript-based interactivity through D3.js and works in both Jupyter notebooks and standalone web apps.
- **Web-Ready Interactivity:**
  Plotly produces interactive charts by default (zoom, pan, tooltip, hover, etc.), making it suitable for dashboards, web apps, and data exploration in browser environments.
- **Two Interfaces:**
  oplotly.express (High-level, concise syntax)
  oplotly.graph_objects (Low-level, full customization)
- **Customizability:**
  Plotly supports dynamic customization with layout updates, annotations, shapes, and trace manipulations. It’s suitable for both quick plotting and finely controlled custom visuals.
- **Integration:**
  Plotly integrates well with Pandas, NumPy, and Dash (Plotly's dashboarding framework), making it ideal for building full-fledged interactive data applications.

Advantages of Plotly:

- Fully interactive plots with minimal code.
- Great for web integration and dynamic dashboards.
- Modern, responsive visuals.
- Extensive support for complex chart types (e.g., choropleths, 3D plots, sunbursts).
- Can export to HTML for embedding in web pages.

Seaborn:

- **Statistical Visualization Library:**
  Seaborn is built on top of Matplotlib, offering a high-level API for statistical plots with beautiful default styles and themes.
- **Simplified Syntax for Complex Plots:**
  Seaborn reduces the boilerplate needed to produce informative graphics. It provides default statistical estimators, confidence intervals, and color mappings with just a few lines of code.
- **Statistical Focus:**
  Seaborn shines in statistical data visualization. It provides specialized plots like violin plots, pair plots, and categorical plots, which are ideal for exploratory data analysis (EDA).

- **Pandas Integration:**
  Seaborn is tightly integrated with Pandas, allowing users to pass DataFrames directly and refer to columns by name—no need for manual slicing or aggregation.

Advantages of Seaborn:

- Cleaner syntax for statistical plots.
- Built-in themes and color palettes for attractive visuals.
- Default support for aggregations (e.g., mean, CI).
- Perfect for EDA and quick visual comparisons.
- Automatically handles subplots with FacetGrid and PairGrid.

CONCLUSION:

- Choose Plotly when you need:
  - Interactive and web-ready visualizations.
  - Dynamic dashboards or HTML exports.
  - Advanced chart types like 3D plots, map visualizations, or animated figures.
- Choose Seaborn when you need:
  - Clean and beautiful statistical plots quickly.
  - High-level data exploration integrated with Pandas.
  - Customizable plots without focusing on interactivity.

In practice, many developers and data scientists use both libraries leveraging Seaborn for fast EDA and Plotly for presenting insights interactively.