VISUALIZATIONS: PANDAS & SEABORN

Introduction

Data visualization is the practice of transforming raw numbers and statistics into meaningful visual representations that communicate insights effectively. Python offers a robust ecosystem of libraries to support this, enabling everything from simple line charts to advanced interactive dashboards. Whether you are exploring a dataset for the first time, presenting insights to stakeholders, or developing a live data application, choosing the right visualization tool can significantly enhance clarity and impact.

This guide provides an overview of some of the most widely used Python visualization libraries, including **pandas** and **Seaborn**. It explains their strengths, key differences, and practical use cases. By the end, you will have a clear understanding of which library best suits your needs and how to begin creating compelling, data-driven visuals.

PANDAS

Pandas is a versatile Python library widely used for data manipulation, analysis, and visualization. It introduces two core data structures—**Series** (1D) and **DataFrame** (2D)—that make it easy to work with structured, tabular data.

While Pandas is primarily recognized for its powerful data cleaning, transformation, and analysis capabilities, it also provides built-in visualization methods. These allow users to quickly generate plots directly from Series or DataFrames, making Pandas especially valuable for **exploratory data analysis (EDA)**.

Key Features of Pandas

1. **Powerful Data Structures** – Provides **Series (1D)** and **DataFrame (2D)** for handling structured datasets.
2. **Flexible Data Manipulation** – Supports filtering, grouping, merging, joining, and reshaping data.
3. **Robust Handling of Missing Data** – Includes tools to detect, remove, or impute missing values.
4. **Seamless Data Import & Export** – Reads and writes multiple formats, including **CSV, Excel, JSON, and SQL databases**.
5. **Built-in Visualization** – Offers quick plotting directly from Series/DataFrames using .plot().
6. **Integration with Other Libraries** – Works efficiently with **NumPy, Matplotlib, and other data science libraries**.
7. **High Performance** – Optimized for large-scale datasets with efficient indexing and operations.

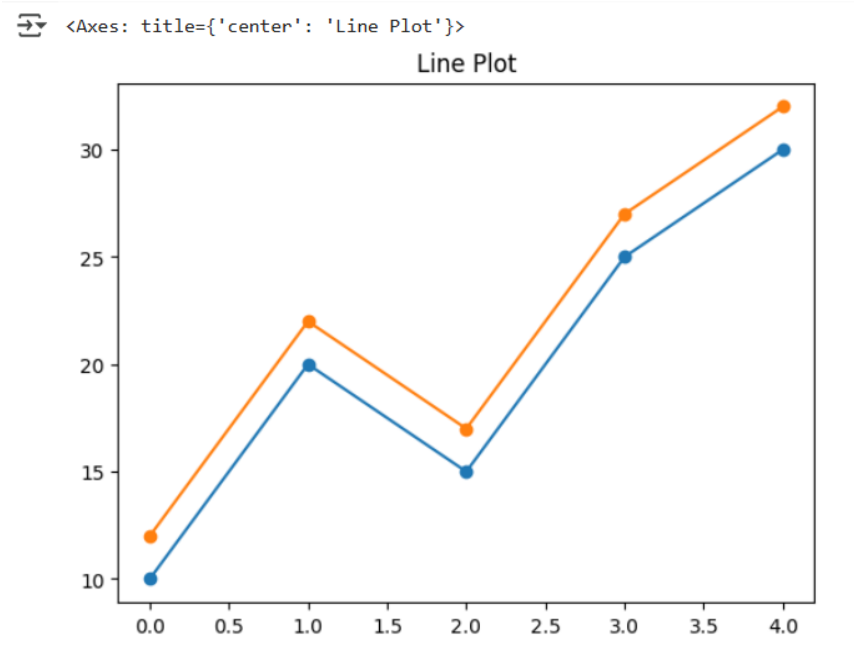
GRAPH TYPES:

1. Line Plot

Description: Plots data points connected by lines, showing trends over continuous data.
Use Case: Sales growth over months.

Sample Code:

```
import pandas as pd
data = pd.Series([10, 20, 15, 25, 30])
data2  = pd.Series([12,22,17,27,32])
data.plot(title="Line Plot", marker='o')
data2.plot(title="Line Plot", marker='o')
```

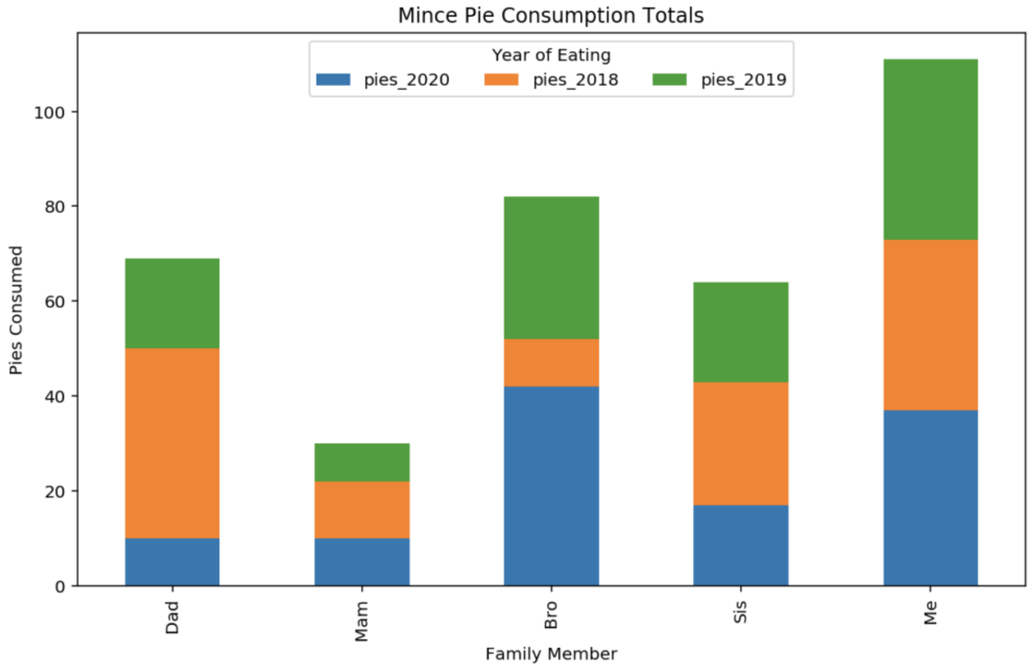


2. Bar Chart

Description: Displays data as rectangular bars for categorical comparison.
Use Case: Comparing product sales.

Sample Code:

```
df = pd.Series([5, 8, 12], index=['A', 'B', 'C'])
df.plot(kind='bar', title="Bar Chart", color='skyblue')
```

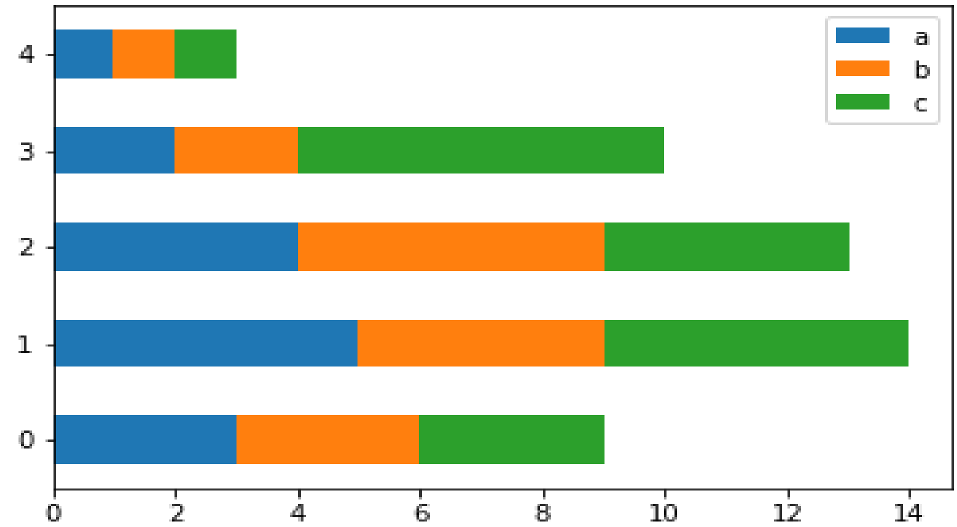


3. Horizontal Bar Chart

Description: Same as a bar chart but horizontal.
Use Case: Comparing values when category labels are long.

Sample Code:

df.plot(kind='barh', title="Horizontal Bar Chart", color='lightgreen')



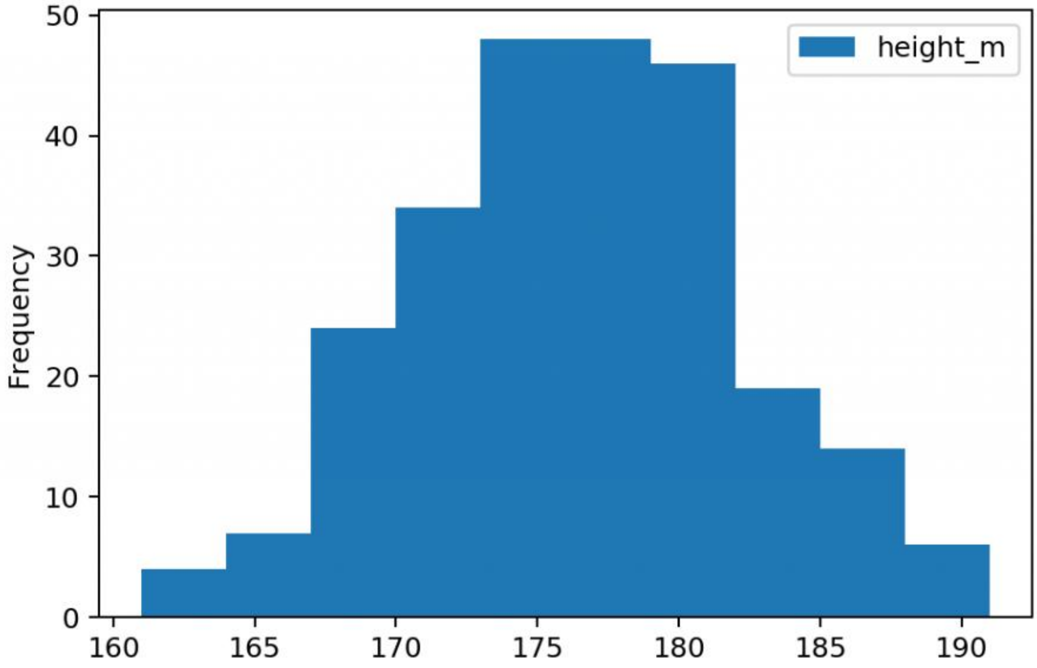
4. Histogram

Description: Groups numeric data into bins to show frequency distribution.
Use Case: Analyzing exam score distributions

Sample Code:

```
data = pd.Series([3, 5, 5, 6, 7, 8, 8, 9, 10])

data.plot(kind='hist', bins=5, title="Histogram", color='orange', edgecolor='black')
```

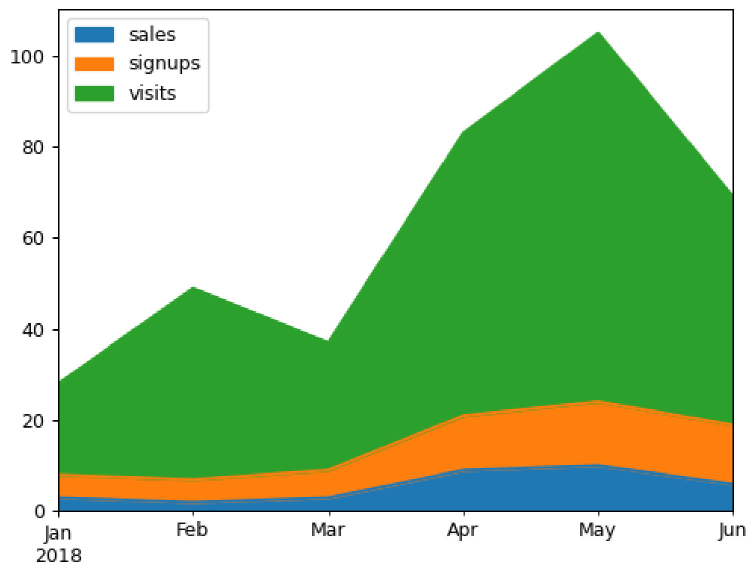


5. Area Plot

Description: Like a line plot but the area under the line is filled.
Use Case: Showing cumulative trends.

Sample Code:

```
df = pd.DataFrame({
    'A': [1, 3, 4],
    'B': [2, 4, 6]
})
df.plot(kind='area', alpha=0.5, title="Area Plot")
```

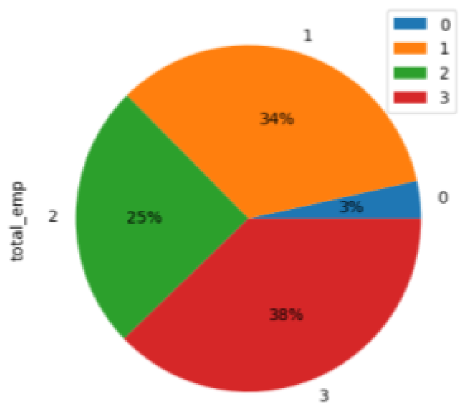


6. Pie Chart

Description: Shows proportions of a whole as slices of a circle.
Use Case: Visualizing budget distribution.

Sample Code:

```
df = pd.Series([30, 20, 50], index=['A', 'B', 'C'])
df.plot(kind='pie', autopct='%1.1f%%', title="Pie Chart")
```



SEABORN

Seaborn is a Python data visualization library built on top of Matplotlib, designed to provide a higher-level interface for creating attractive and informative graphics. Compared to Matplotlib, Seaborn offers advanced built-in features such as improved default styles, color palettes, and simplified syntax, making it easier to generate visually appealing plots with minimal code.

Seaborn supports a wide range of plot types that cater to different data visualization needs, including:

1 Relational Plots:

· scatterplot()

· lineplot()

· relplot()

2 Categorical Plots:

· barplot()

· countplot()

· boxplot()

· violinplot()

· swarmplot()

· pointplot()

· catplot()

3 Distribution Plots:

· histplot()

· kdeplot()

· rugplot()

· distplot()

4 Relational Plots:

· regplot()

· lmplot()

5 Matrix Plots:

· heatmap()

· clustermap()

Here are some sample codes for some of the graphs.

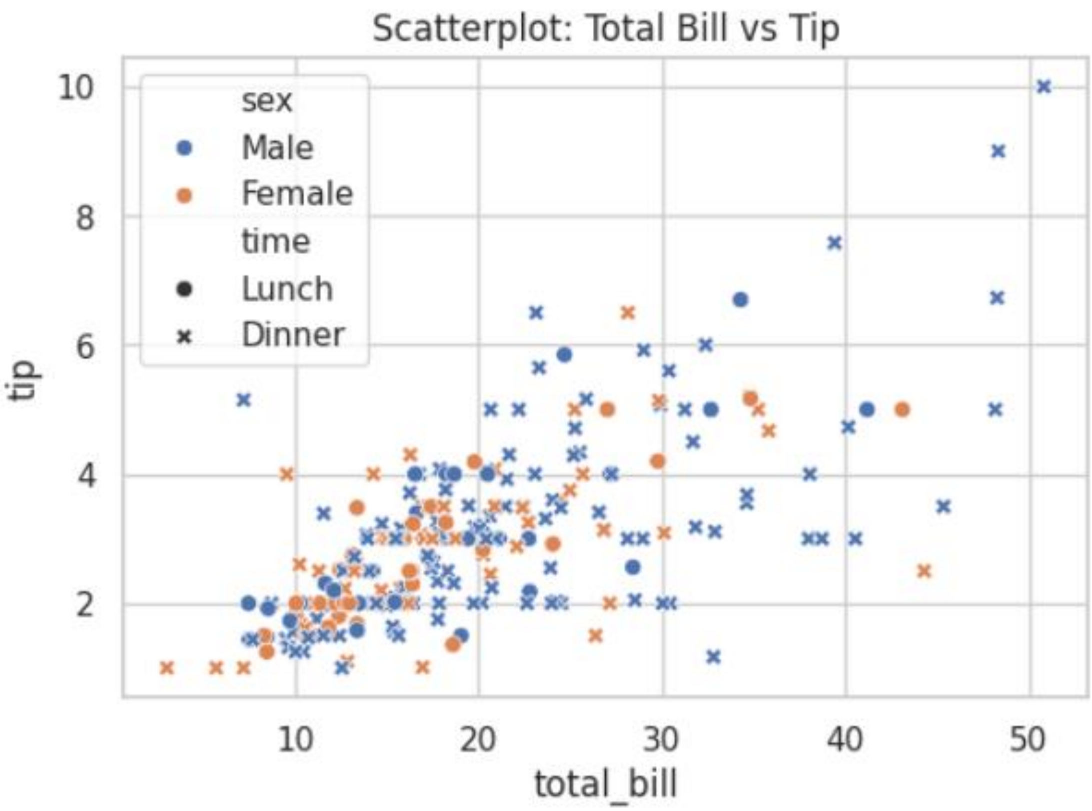
```
# Import required libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load sample datasets
tips = sns.load_dataset("tips")
flights = sns.load_dataset("flights")
iris = sns.load_dataset("iris")

# Set Seaborn style
sns.set(style="whitegrid")
```

1. Scatterplot

```
plt.figure(figsize=(6,4))
sns.scatterplot(x="total_bill", y="tip", data=tips, hue="sex", style="time")
plt.title("Scatterplot: Total Bill vs Tip")
plt.show()
```



Description:
scatterplot() plots total_bill on the x-axis and tip on the y-axis.

- The hue="sex" parameter assigns different colors to Male and Female categories.

- The style="time" parameter varies marker shapes based on Lunch/Dinner.
- Each point represents one customer.
- title() adds a title, xlabel() and ylabel() label the axes.
- show() displays the scatterplot.

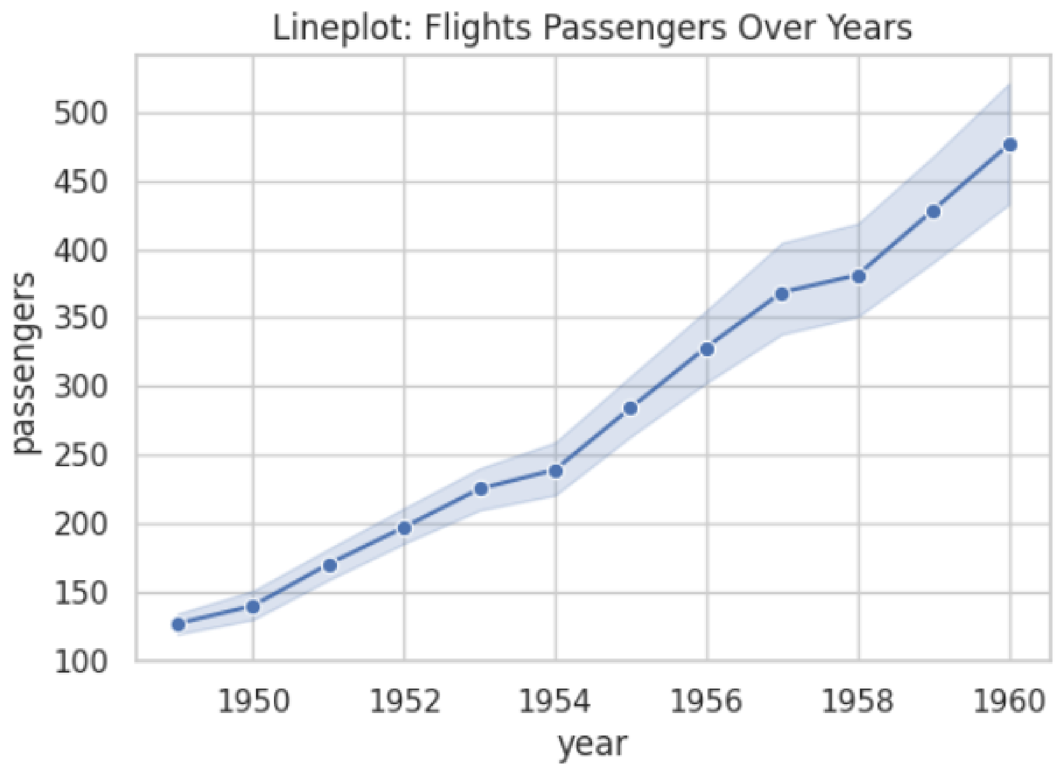
2. Lineplot

plt.figure(figsize=(6,4))

sns.lineplot(x="year", y="passengers", data=flights, marker="o")

plt.title("Lineplot: Flights Passengers Over Years")

plt.show()



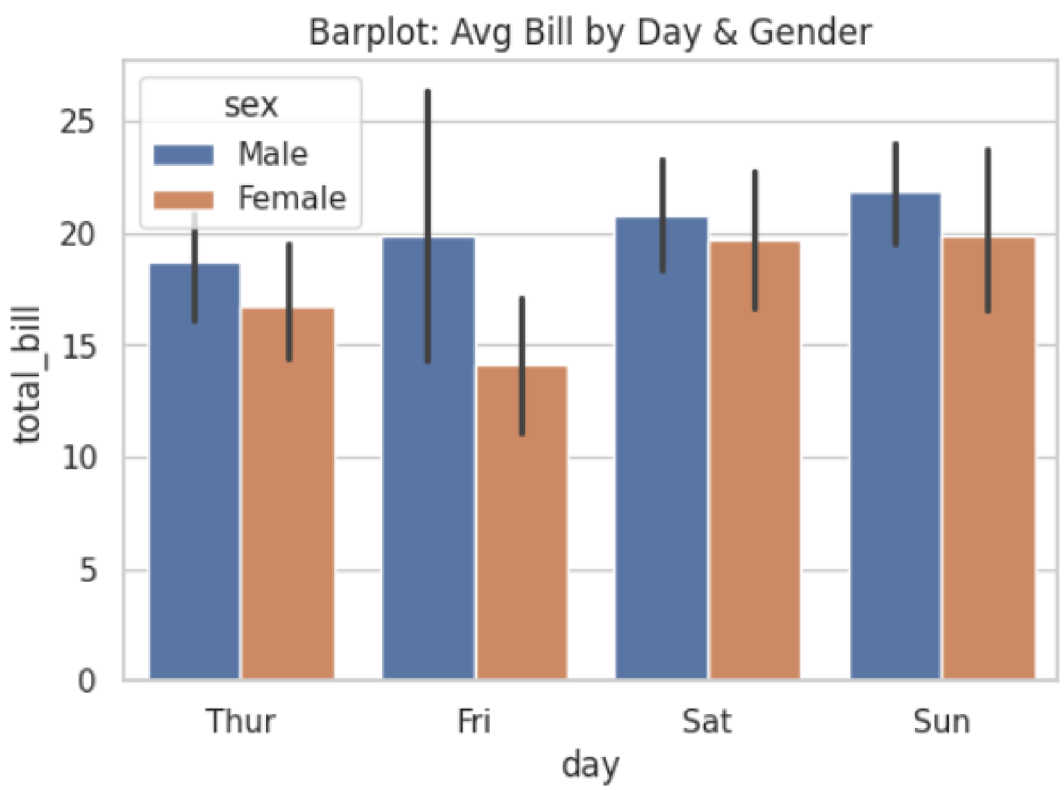
Description:

lineplot() plots year on the x-axis and passengers on the y-axis.

- The line connects points to show trends over time.
- The marker="o" adds circular markers for each data point.
- Useful for visualizing how the number of passengers changes across years.

3. Barplot

```
plt.figure(figsize=(6,4))
sns.barplot(x="day", y="total_bill", data=tips, hue="sex")
plt.title("Barplot: Avg Bill by Day & Gender")
plt.show()
```



Description:
barplot() displays the average total bill for each day of the week.

- The hue="sex" parameter splits each bar by Male/Female.
- By default, it shows the mean with confidence intervals.
- Useful for comparing group averages across categories.

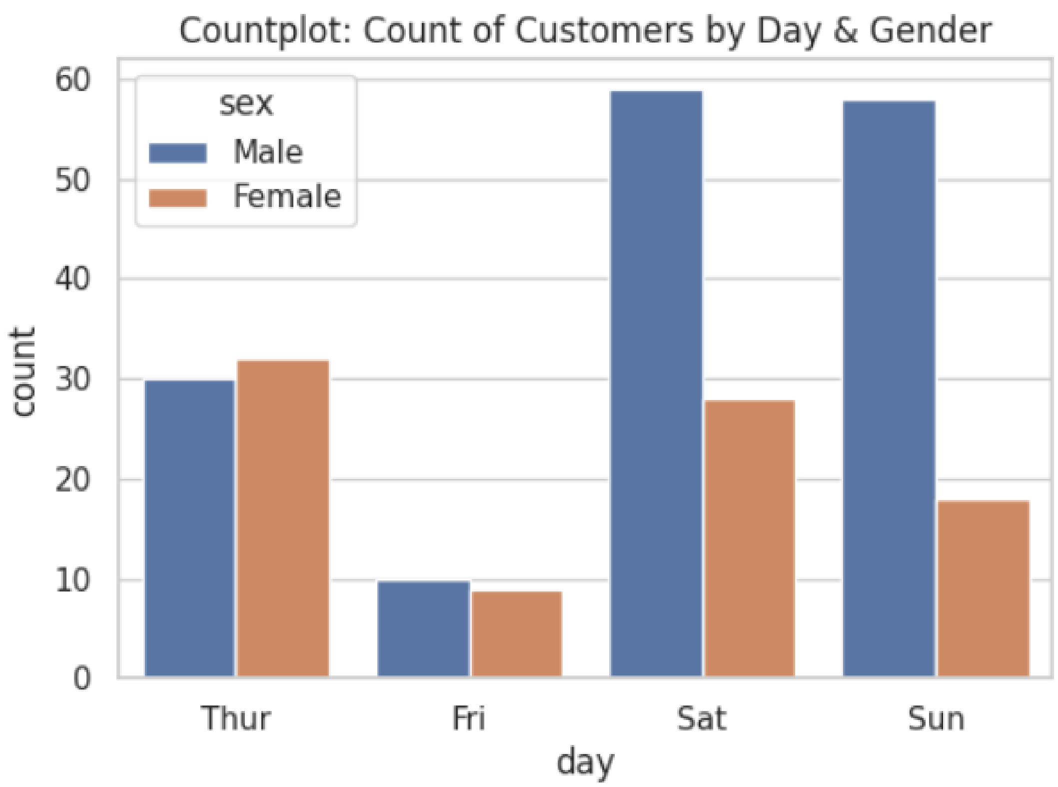
4. Countplot

plt.figure(figsize=(6,4))

sns.countplot(x="day", data=tips, hue="sex")

plt.title("Countplot: Count of Customers by Day & Gender")

plt.show()



Description:
countplot() shows the count of observations for each day of the week.

- The hue="sex" parameter splits counts into Male/Female categories.
- Bars represent frequency rather than averages**.**
- Helpful for seeing category distributions.

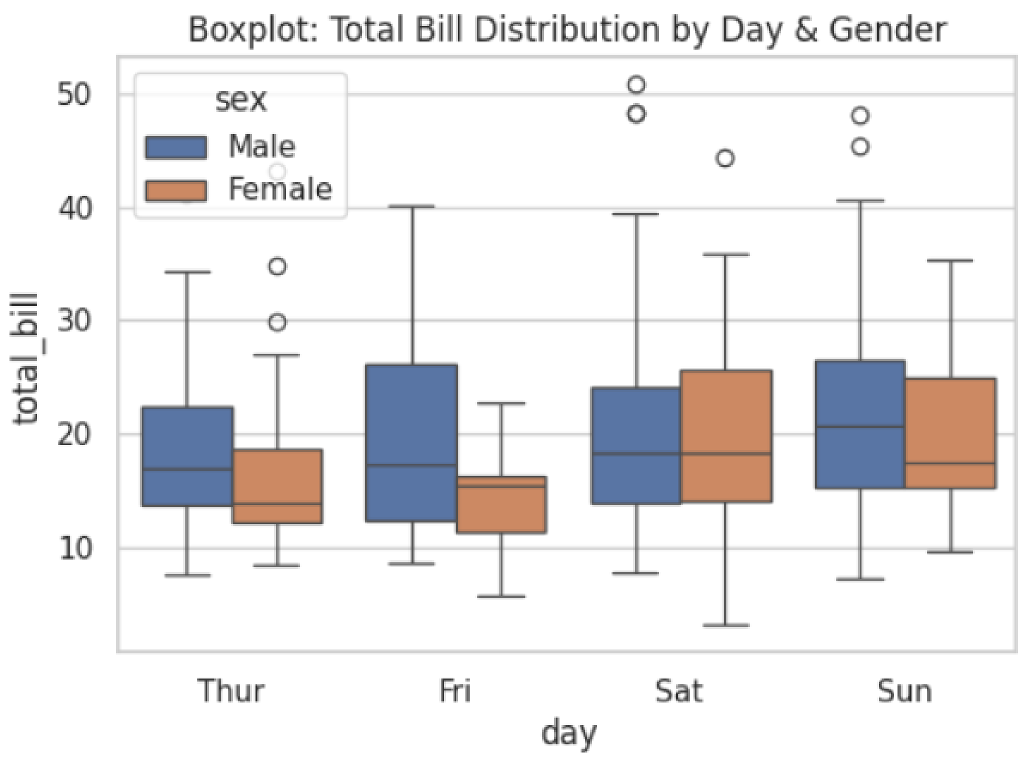
5. Boxplot

plt.figure(figsize=(6,4))

sns.boxplot(x="day", y="total_bill", data=tips, hue="sex")

plt.title("Boxplot: Total Bill Distribution by Day & Gender")

plt.show()



Description:

boxplot() displays the distribution of total bills for each day.

- The box shows the median, 25th percentile, and 75th percentile.
- Whiskers show variability outside the quartiles.
- Dots represent outliers (very high/low bills).
- The hue="sex" parameter allows Male/Female comparison.

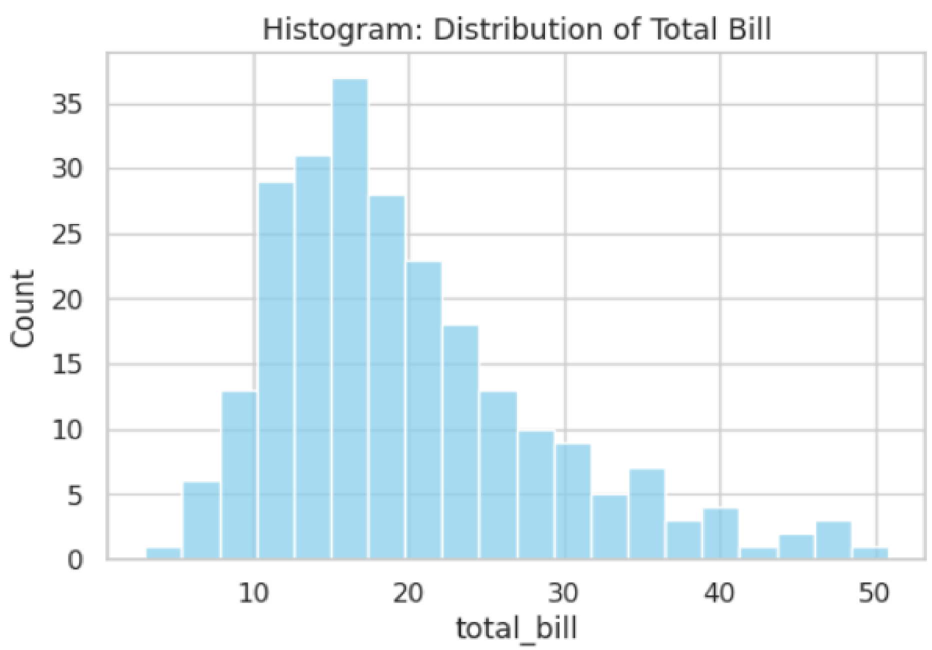
6. Histogram (Histplot)

plt.figure(figsize=(6,4))

sns.histplot(tips["total_bill"], bins=20, kde=False, color="skyblue")

plt.title("Histogram: Distribution of Total Bill")

plt.show()

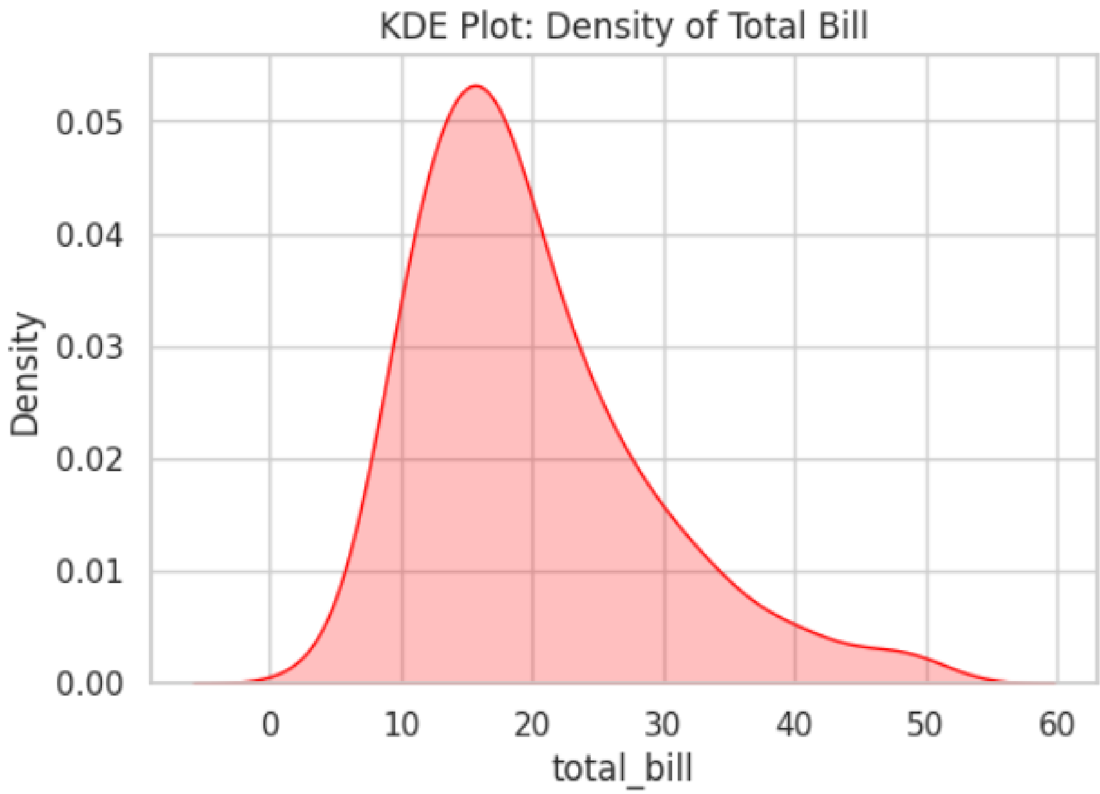


Description:
histplot() shows the frequency distribution of total bills.

- The bins=20 parameter controls the number of intervals.
- The kde=False hides the kernel density line (can be enabled if needed).
- The color parameter sets bar color.
- Useful for understanding distribution shape and spread.

 # 7. KDE Plot

```
plt.figure(figsize=(6,4))
sns.kdeplot(tips["total_bill"], shade=True, color="red")
plt.title("KDE Plot: Density of Total Bill")
plt.show()
```



Description:
kdeplot() displays the Kernel Density Estimation for total_bill.

- The shade=True parameter fills the area under the curve.
- The curve represents a smoothed version of the histogram.
- Useful for identifying peaks and overall distribution shape.

8. Heatmap

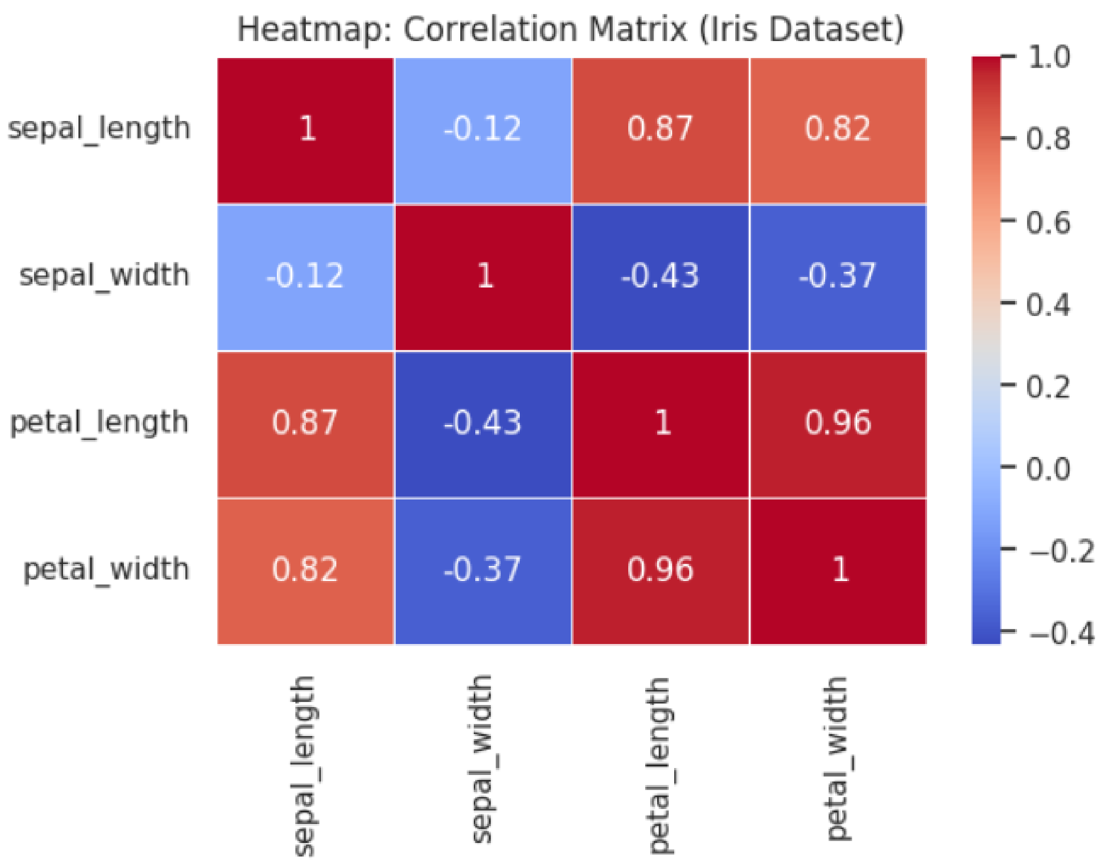
plt.figure(figsize=(6,4))

corr = iris.select_dtypes(include=["float64", "int64"]).corr() # Only numeric columns

sns.heatmap(corr, annot=True, cmap="coolwarm", linewidths=0.5)

plt.title("Heatmap: Correlation Matrix (Iris Dataset)")

plt.show()



Description:
heatmap() visualizes the correlation matrix of numeric columns in the Iris dataset.

- annot=True displays correlation values inside cells.
- cmap="coolwarm" applies a blue-to-red color gradient.
- linewidths=0.5 adds spacing between cells.
- Darker colors indicate stronger positive/negative correlations.

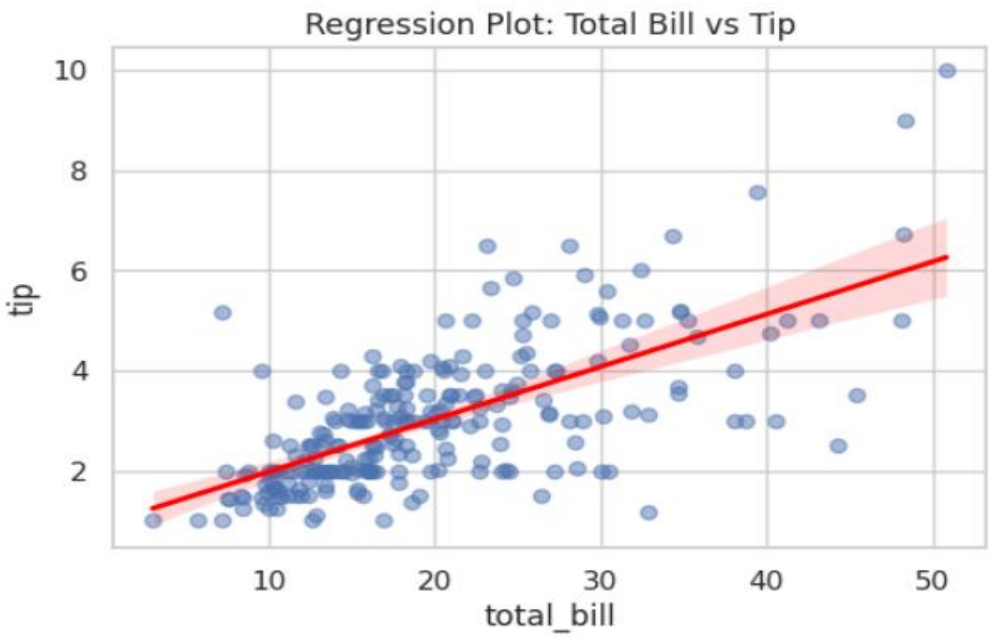
9. Regression Plot

plt.figure(figsize=(6,4))

sns.regplot(x="total_bill", y="tip", data=tips, scatter_kws={"alpha":0.5}, line_kws={"color":"red"})

plt.title("Regression Plot: Total Bill vs Tip")

plt.show()



Description:
regplot() shows the relationship between total bill and tip with a regression line.

- scatter_kws={"alpha":0.5} makes points semi-transparent.
- line_kws={"color":"red"} customizes the regression line color.
- Useful for analyzing linear trends and correlation strength between variables.

COMPARISON OF PANDAS AND SEABORN

□ **Purpose**

- **Pandas** → Mainly for data manipulation with some basic visualization.
- **Seaborn** → Mainly for data visualization with advanced statistical plots.

□ **Plotting Capability**

- **Pandas** → Limited to basic plots (.plot() for line, bar, histogram, scatter).
- **Seaborn** → Offers a wide variety (scatter, line, bar, box, KDE, heatmap, regression, etc.).

□ **Customization**

- **Pandas** → Customization needs extra work with Matplotlib.

- **Seaborn** → Provides beautiful defaults (colors, styles, palettes) with less effort.

 Ease of Use

- **Pandas** → Quick plots for fast exploratory checks.
- **Seaborn** → Better for presentation-ready visuals and detailed analysis.

 Integration

- **Pandas** → Works seamlessly with NumPy, Seaborn, Matplotlib.
- **Seaborn** → Built on Matplotlib, works smoothly with Pandas DataFrames.

CONCLUSION:

Both Pandas are essential tools in Python's data visualization ecosystem, each serving different needs. For **in-depth, highly customized visuals** → use **Seaborn**.For **fast, simple visualizations during analysis** → use **Pandas**.