3. Exploring Matplotlib & Seaborn

1. Types of Analysis (by Number of Variables)

- When we explore data visually, we can look at one, two, or many columns together.
- Based on how many variables we study at once, there are 3 types of analysis:
  - **Univariate Analysis:**
    - One variable → understand distribution
  - **Bivariate Analysis**
    - Two variables → understand relationship
  - **Multivariate Analysis**
    - 3 or more variables → understand patterns & interactions

1.1. Univariate Analysis (One Variable)

- **Used when you are observing the behavior of a single variable / column / feature at a time.**

Goal:

- Understand its distribution
- Find its center (mean/median)
- See its spread (range, variance)
- Detect outliers or unusual values

Common Plots:

- **Histogram** — shows how values are spread
- **Boxplot** — shows median, quartiles, and outliers
- **Bar Chart** — for frequency counts of categories
- **Count Plot** — for counting categorical values
- **Pie Chart** — for showing proportion of categories

Example 1 — Age column (numeric):

- Suppose you have a dataset of 1,000 customers.
- **You want to see:**
  - What is the average age of customers?
  - Are most customers young, middle-aged, or older?
  - Are there any extremely old or very young outliers?

Plots to use:

- **Histogram** — shows how ages are spread
- **Boxplot** — shows median, quartiles, and outliers

Example 2 — Product Category column (categorical):

- Suppose you have an Online Retail dataset.
- You want to see:
  - How many products belong to each Category (like Electronics, Clothing, Toys)
  - Which category has the highest count of products

Plots to use:

- **Bar Chart / Count Plot** — show counts for each category
- **Pie Chart** — show proportions of each category

1.2. Bivariate Analysis (Two Variables)

- **Used when you are studying the relationship between two variables / columns / features at a time.**

Goal:

- Identify patterns, trends, or relationships between two variables
- Understand if they are positively/negatively correlated
- Compare how one variable changes with respect to another

Common Plots

- **Scatter Plot** — shows how two numeric variables move together
- **Line Plot** — shows how a numeric variable changes over time
- **Grouped Bar Chart** — compare numbers between categories
- **Side-by-side Boxplots** — compare distributions across categories
- **Heatmap (2D)** — shows relationship between numeric variables (correlations)

Example 1 — Age vs Salary (numeric vs numeric)

- Suppose you have a dataset of employees.
- **You want to see:**
  - Does salary increase as age increases?
  - Are there any outliers (very high salary for young age)?
  - Is there a clear trend or cluster in the data?

Plots to use:

- **Scatter Plot** — to see relationship and trend
- **Line Plot** — if data is time-based (like age vs salary growth by year)

Example 2 — Product Category vs Sales (categorical vs numeric)

- Suppose you have an Online Retail dataset.
- **You want to see:**
  - Which product category generates more sales on average
  - How sales are spread across categories

Plots to use:

- **Grouped Bar Chart** — compare average sales of each category
- **Side-by-side Boxplots** — show distribution of sales per category

1.3. Multivariate Analysis (Three or More Variables)

- **Used when you are studying the relationship between three or more variables / columns / features at the same time.**

Goal:

- Understand complex patterns and interactions between multiple variables
- Detect clusters, correlations, or combined effects
- Explore how multiple features together influence a target

Common Plots

- **Pairplot** — Scatterplots for all variable combinations (numeric only)
- **Heatmap** — shows correlations between many numeric variables
- **FacetGrid / Subplots** — create multiple smaller plots for subgroups
- **3D Scatter Plot** — visualize three numeric variables together

Example 1 — Age vs Salary vs Education (3 numeric/categorical mix)

- Suppose you have an employee dataset.
- **You want to see:**
  - How age and education level together affect salary
  - If higher education gives higher salary at any age
  - If there are clusters of employees by age and salary

Plots to use:

- **3D Scatter Plot** — plot Age (x), Salary (y), Experience (z)
- **Bubble Plot** — Age vs Salary, with bubble size showing Experience
- **FacetGrid** — separate plots for each Education level showing Salary vs Age

Example 2 — Correlations among multiple numeric columns

- Suppose you have a Sales dataset with columns like Sales, Profit, Discount, Quantity, Rating.
- **You want to see:**
  - Are Sales and Profit strongly related?
  - Does high Discount reduce Profit?
  - How are all variables connected?

Plots to use:

- **Heatmap** — show correlation matrix among all numeric columns
- **Pairplot** — see scatterplots for all pairs at once

2. Introduction to Visualization Tools

- When analyzing data, we need libraries to help us visualize it quickly.
- In Python, two of the most popular libraries are Matplotlib and Seaborn.

2.1. What is Data Visualization?

- Data Visualization is the process of converting raw data into visual form (charts, graphs).

It helps to:

- See patterns and trends
- Detect outliers
- Communicate results clearly to others

2.2. Why Use Matplotlib and Seaborn?

- **Matplotlib is the foundation** — it gives you full control over figures, axes, colors, and styling.
- **Seaborn is built on top of Matplotlib** — it makes plotting simpler and prettier with less code.

Difference Between Matplotlib and Seaborn

- **Feature --- Matplotlib --- Seaborn**
- Purpose --- Low-level library for full control --- High-level library for quick & attractive plots
- Code Required --- More code (manual styling) --- Less code (sensible defaults)
- Styling --- Plain by default --- Beautiful themes by default
- Customization --- Very flexible, full control --- Limited compared to Matplotlib
- Use Case --- When you want fine control --- When you want quick insights

2.3. Installation, Import and Version Check

A) Install using pip

- If Matplotlib and Seaborn not not already installed in your environment, use the following command:

```
# pip install matplotlib seaborn
```

If you're using Jupyter Notebook or Google Colab, just run:

```
# !pip install matplotlib seaborn
```

B) Import and Version Check

- Use the following code to check installed versions:

```
# Import Libraries
import matplotlib
import seaborn as sns

print("Matplotlib version:", matplotlib.__version__)
print("Seaborn version:", sns.__version__)
```

```
Matplotlib version: 3.10.0
Seaborn version: 0.13.2
```

3. Matplotlib — Core Concepts

3.1. Figure and Axes Objects

- In Matplotlib, every plot is made inside a Figure.
- Inside a Figure, you create one or more Axes (plot areas).

Key Terms

- **Figure** -----> The entire window or page that holds everything (like a canvas)
- **Axes** -----> The actual plot area inside the figure (where x/y axes and data appear)
- **Axis** -----> The x-axis or y-axis inside an Axes (controls ticks, labels, limits)

Think of:

- Figure = Big paper
- Axes = One chart drawn on the paper
- Axis = The x and y lines on the chart

Two Ways to Plot

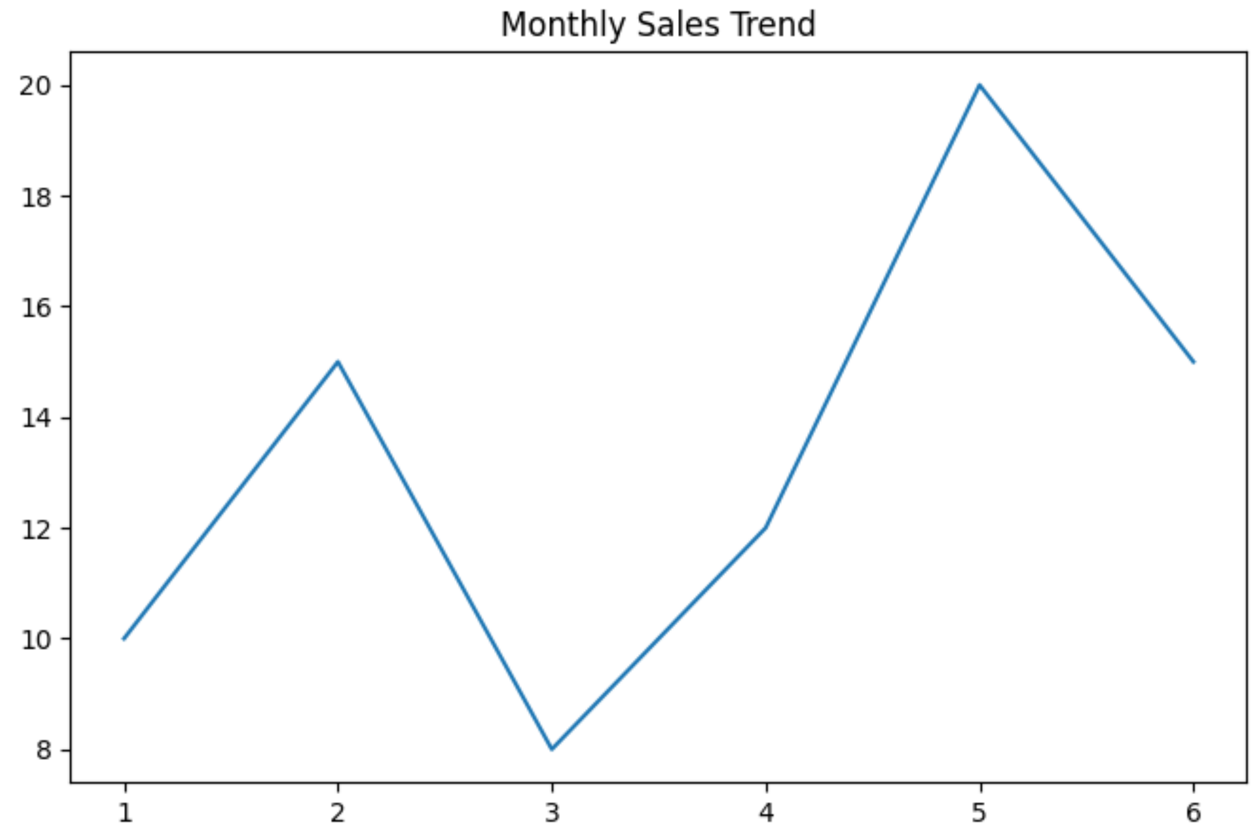
```
import pandas as pd
import matplotlib.pyplot as plt

# 1) Create sample dataset
data = {
    'Month': [1, 2, 3, 4, 5, 6],
    'Sales': [10, 15, 8, 12, 20, 15]
}

df = pd.DataFrame(data)

df
```

```
   Month  Sales
0      1     10
1      2     15
2      3      8
3      4     12
4      5     20
5      6     15
```

Next steps: Generate code with df New interactive sheet

1) State-based (Simple) way using plt

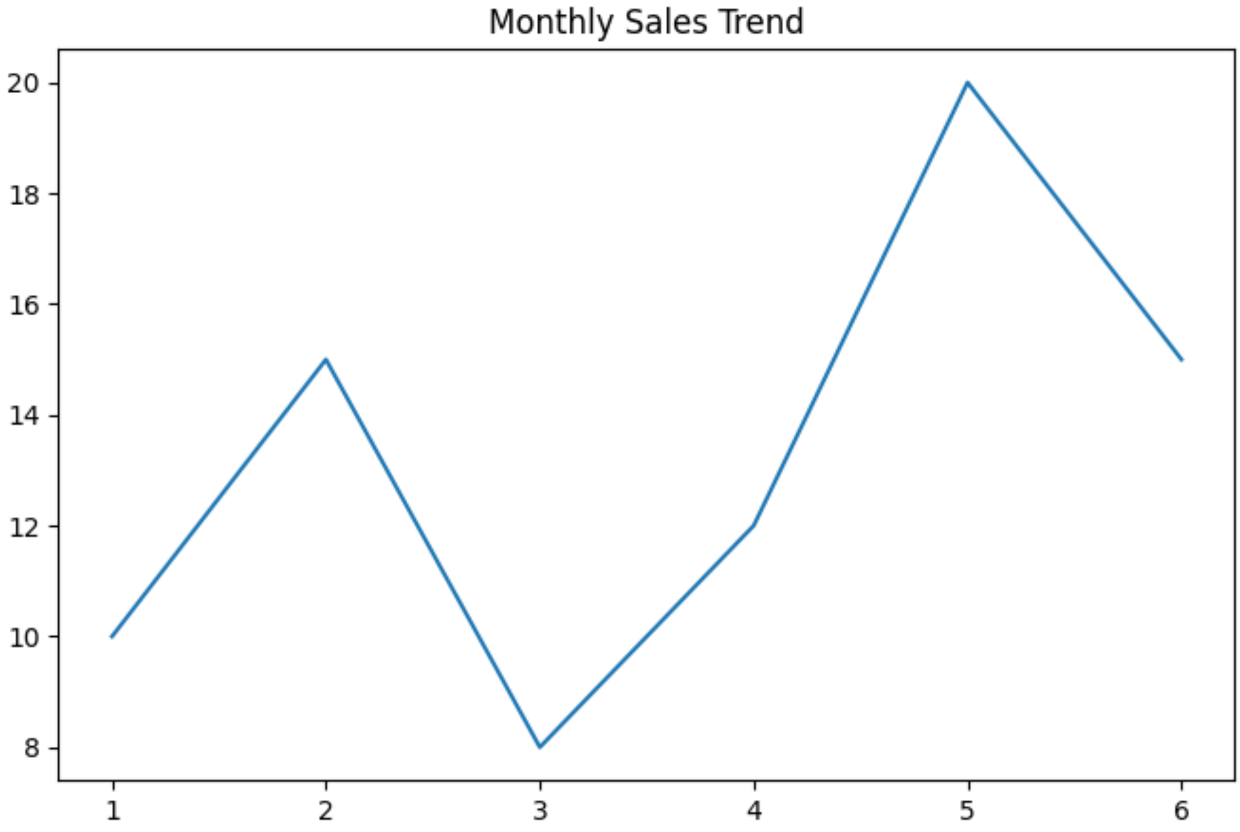
```
# width = 8 inches, height = 6 inches
plt.figure(figsize=(8, 5))

# Plot using DataFrame columns
plt.plot(df['Month'], df['Sales'])

plt.title("Monthly Sales Trend")
plt.show()

# plt automatically creates a Figure and one Axes behind the scenes.
```



2) Object-oriented way using fig, ax

- plt.subplots() returns:
  - fig ⟶ the Figure object
  - ax ⟶ the Axes object

```
# 2) Plot using fig, ax style
myfig, myax = plt.subplots(figsize=(8, 5))

myax.plot(df['Month'], df['Sales'])

myax.set_title("Monthly Sales Trend")

plt.show()
```



rcParams in Matplotlib

- rcParams is a dictionary-like object that stores all default plot settings.
- You can read or change these settings to control how all plots look.

```
Syntax:

plt.rcParams['setting_name'] = value
```

Common Global Settings

- **Setting Name** ----- Purpose ----- Example
- **'figure.figsize'** ----- Default size of figure (width, height in inches) ----- (8, 5)
- **'figure.dpi'** ----- Resolution (dots per inch) ----- 100
- **'font.size'** ----- Default text size (title, labels, ticks) ----- 14
- **'axes.titlesize'** ----- Title text size (axes title) ----- 16
- **'axes.labelsize'** ----- Axis label text size ----- 14
- **'xtick.labelsize'** ----- X-axis tick label size ----- 12
- **'ytick.labelsize'** ----- Y-axis tick label size ----- 12
- **'axes.edgecolor'** ----- Border color around plot ----- 'black'
- **'axes.facecolor'** ----- Background color of axes area -----'white'

```
Ex:
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (8, 5)
plt.rcParams['figure.dpi'] = 100

# Font sizes
plt.rcParams['font.size'] = 14          # all text
plt.rcParams['axes.titlesize'] = 16     # axes title
plt.rcParams['axes.labelsize'] = 14     # x/y labels
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
```

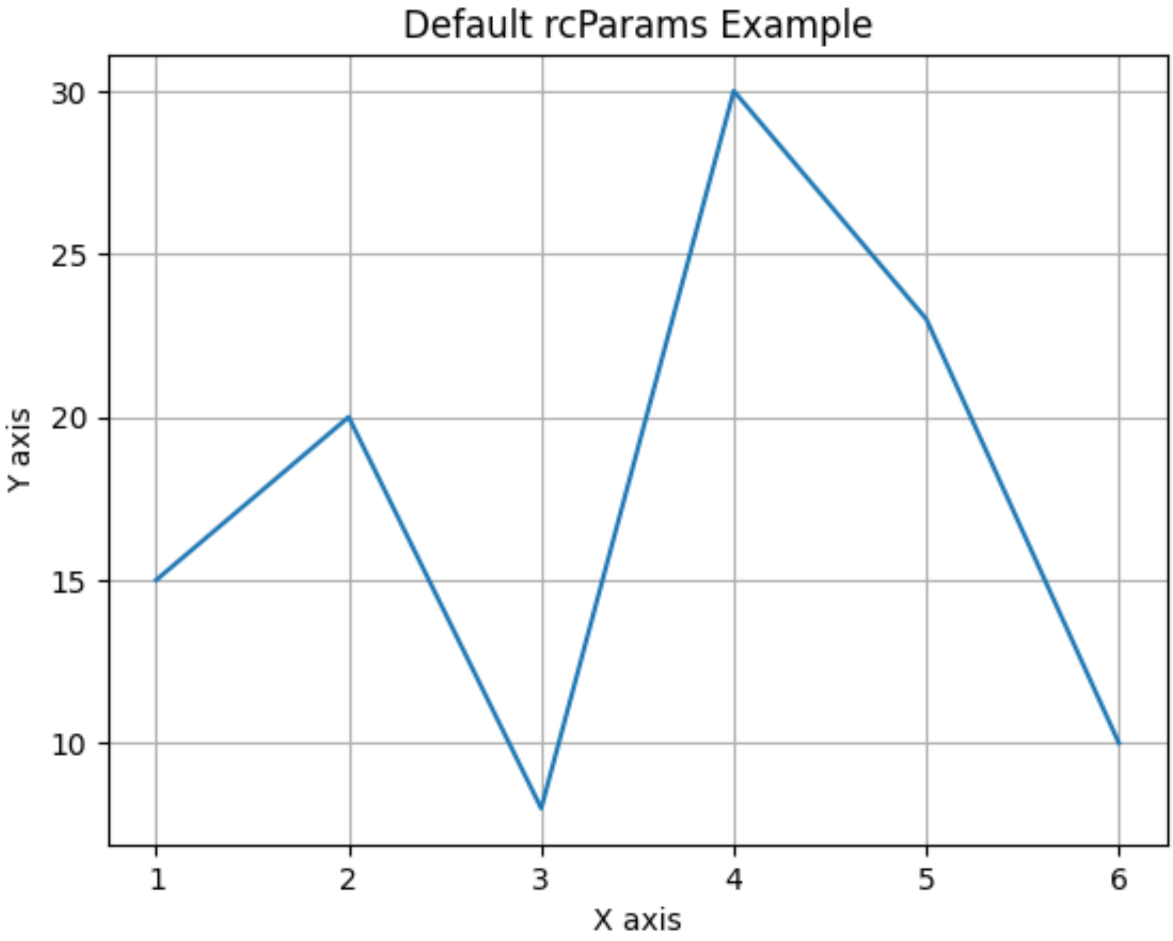
How to Reset to Defaults

plt.rcdefaults() # resets rcParams to default values

Example - to Check default rcParams

```
import matplotlib.pyplot as plt
import numpy as np

# Show some of the most common rcParams defaults
print("=== rcParams Defaults ===")
print("Figure size        :", plt.rcParams['figure.figsize'])
print("DPI                :", plt.rcParams['figure.dpi'])
print("Font size           :", plt.rcParams['font.size'])
print("Title size          :", plt.rcParams['axes.titlesize'])
print("Label size          :", plt.rcParams['axes.labelsize'])
print("Line width           :", plt.rcParams['lines.linewidth'])
print("Line color           :", plt.rcParams['lines.color'])
print("Marker size          :", plt.rcParams['lines.markersize'])
print("Grid enabled         :", plt.rcParams['axes.grid'])

# Create a simple lineplot using default settings
x = [1,2,3,4,5,6]
y = [15,20,8,30,23,10]

plt.plot(x, y)
plt.title("Default rcParams Example")
plt.xlabel("X axis")
plt.ylabel("Y axis")
plt.grid(True)

plt.show()
```

```
=== rcParams Defaults ===
Figure size        : [6.4, 4.8]
DPI                : 100.0
Font size           : 10.0
Title size          : large
Label size          : medium
Line width           : 1.5
Line color           : C0
Marker size          : 6.0
Grid enabled         : False
```

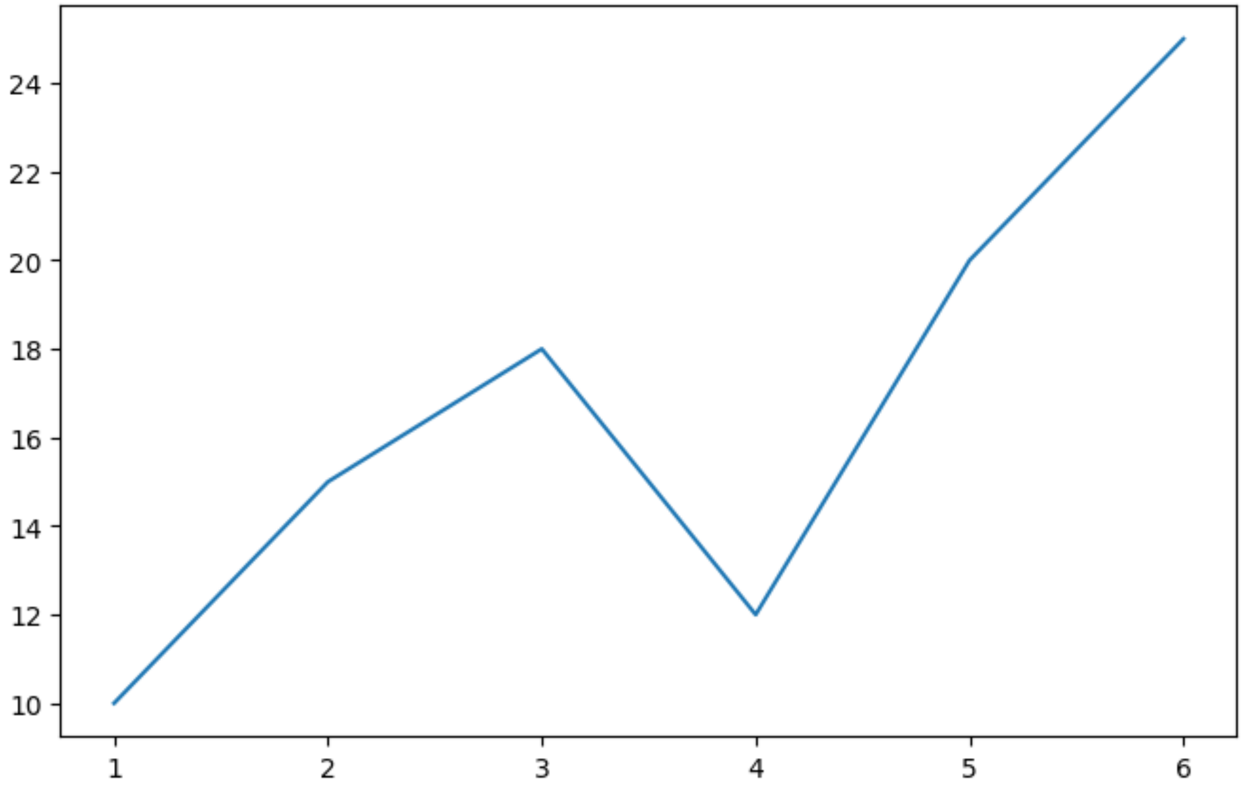


3.2. Creating Simple Line Plots (plt.plot())

- It shows a line connecting data points in order, useful for showing trends or changes.

Example 1 — Line Plot

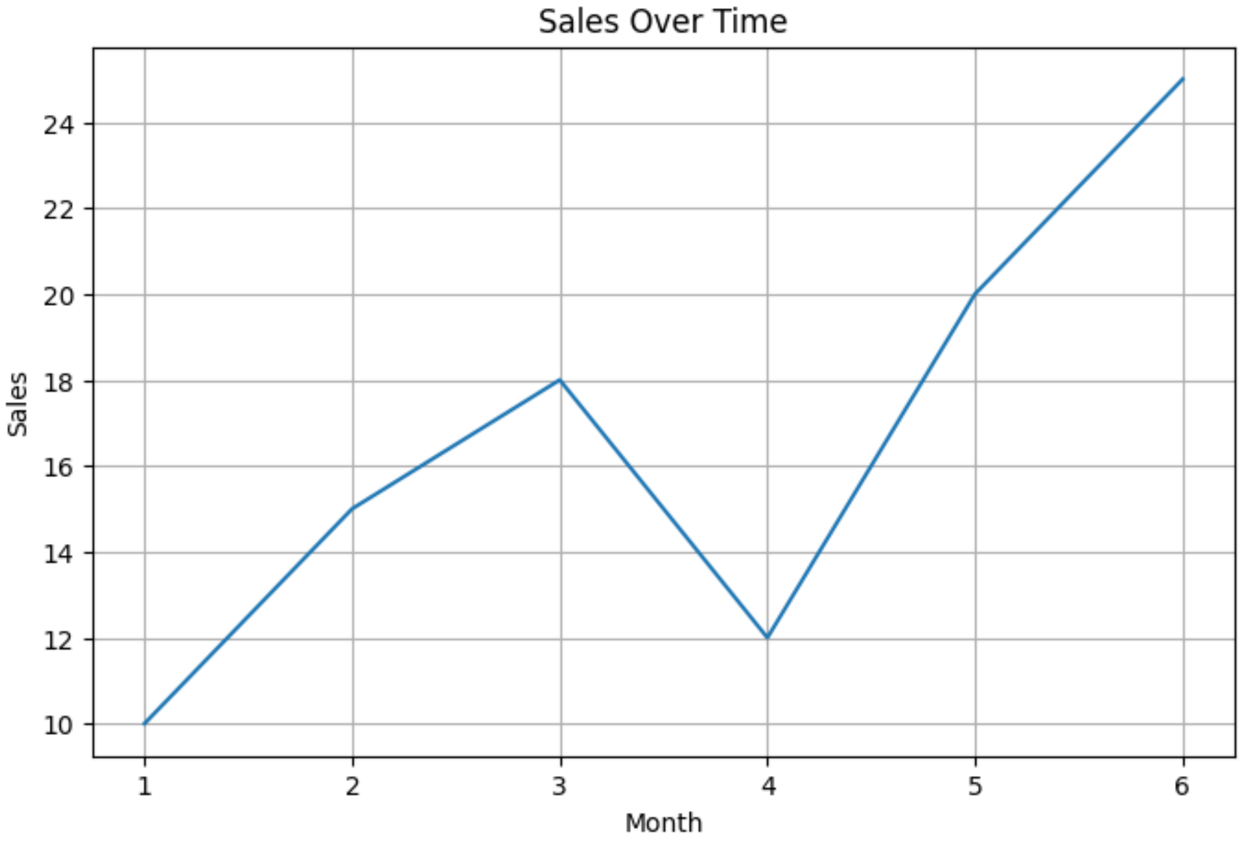
```python
import pandas as pd
import matplotlib.pyplot as plt

# Create sample dataset
data = {
    'Month': [1, 2, 3, 4, 5, 6],
    'Sales': [10, 15, 18, 12, 20,25]
}

df = pd.DataFrame(data)

# Set the Figsize
plt.figure(figsize=(8, 5))

# Create the plot
plt.plot(df['Month'], df['Sales'])

# Show the plot
plt.show()
```



3.3. Adding Labels, Titles, Legends, and Grids

Example 2 — Adding Labels, Title, Grid

```
# Set the Figsize
plt.figure(figsize=(8, 5))

# Create the plot
plt.plot(df['Month'], df['Sales'])

plt.title("Sales Over Time")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.grid(True)

# Show the plot
plt.show()
```



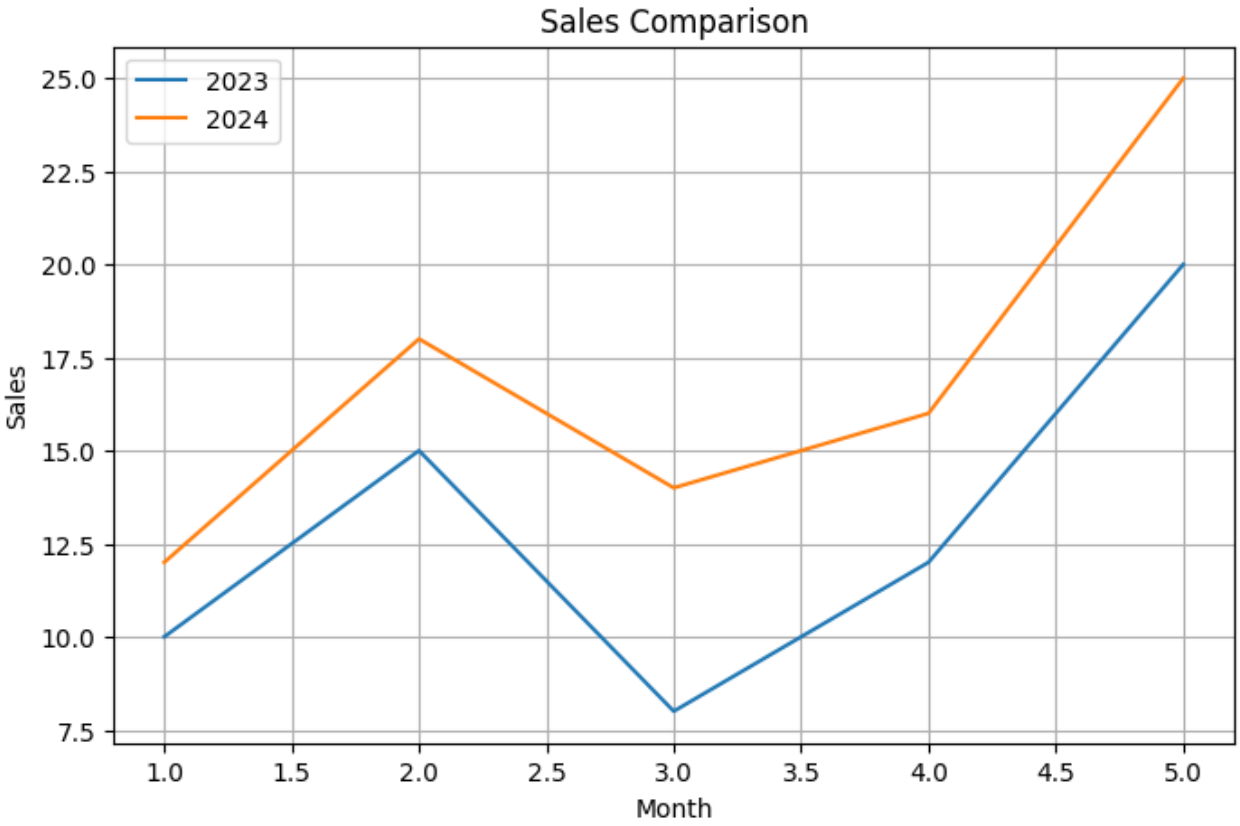
Example 3 — Multiple Lines in One Plot

```
# 1) Create sample dataset
data = {
    'Month': [1, 2, 3, 4, 5],
    'Sales_2023': [10, 15, 8, 12, 20],
    'Sales_2024': [12, 18, 14, 16, 25]
}

df = pd.DataFrame(data)
df
```

	Month	Sales_2023	Sales_2024
0	1	10	12
1	2	15	18
2	3	8	14
3	4	12	16
4	5	20	25

Next steps: Generate code with df New interactive sheet

```
# Set the Figsize
plt.figure(figsize=(8, 5))

plt.plot(df['Month'], df['Sales_2023'], label='2023')
plt.plot(df['Month'], df['Sales_2024'], label='2024')

plt.title("Sales Comparison")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.legend()
plt.grid(True)

plt.show()
```



3.4. Styling (color, marker, linestyle, linewidth)

- You can apply styles to make your line plots more clear, readable, and visually appealing.
- This helps differentiate multiple lines on the same axes.

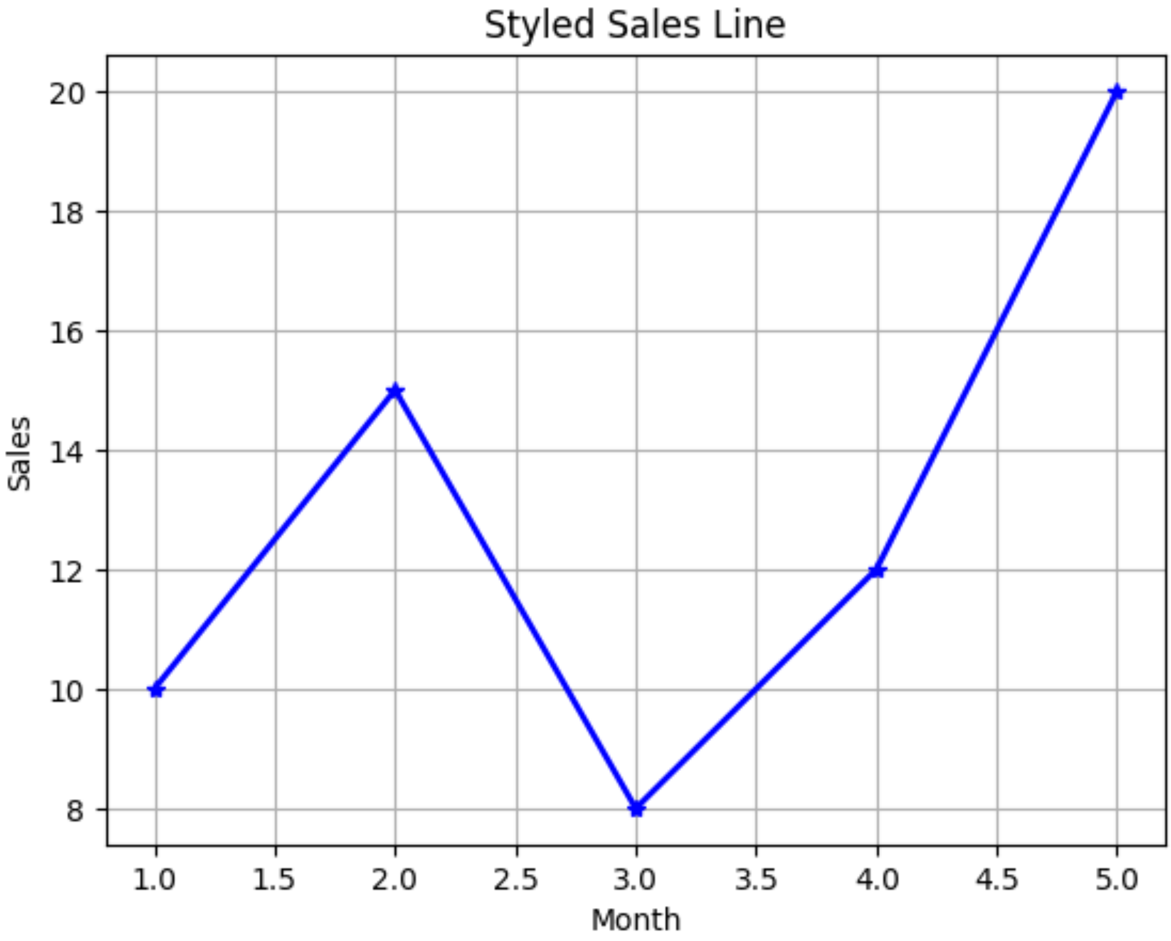
Common Styling Options

- Option ----- Parameter -----> Example ----- Description
- Color ----- color or c -----> 'red', 'blue', 'g' ----- Sets the line color
- Marker ----- marker -----> 'o', 's', '^', '*' ----- Adds symbols on each data point
- Line Style ----- linestyle or ls -----> '-', '--', ':', '-. ' ----- Controls line pattern
- Line Width ----- linewidth or lw -----> 2, 3 ----- Controls thickness of line
- You can also combine them into a style string:
- **Ex:**
    - 'r--o' → red dashed line with circle markers

Example 4 — Styling One Line

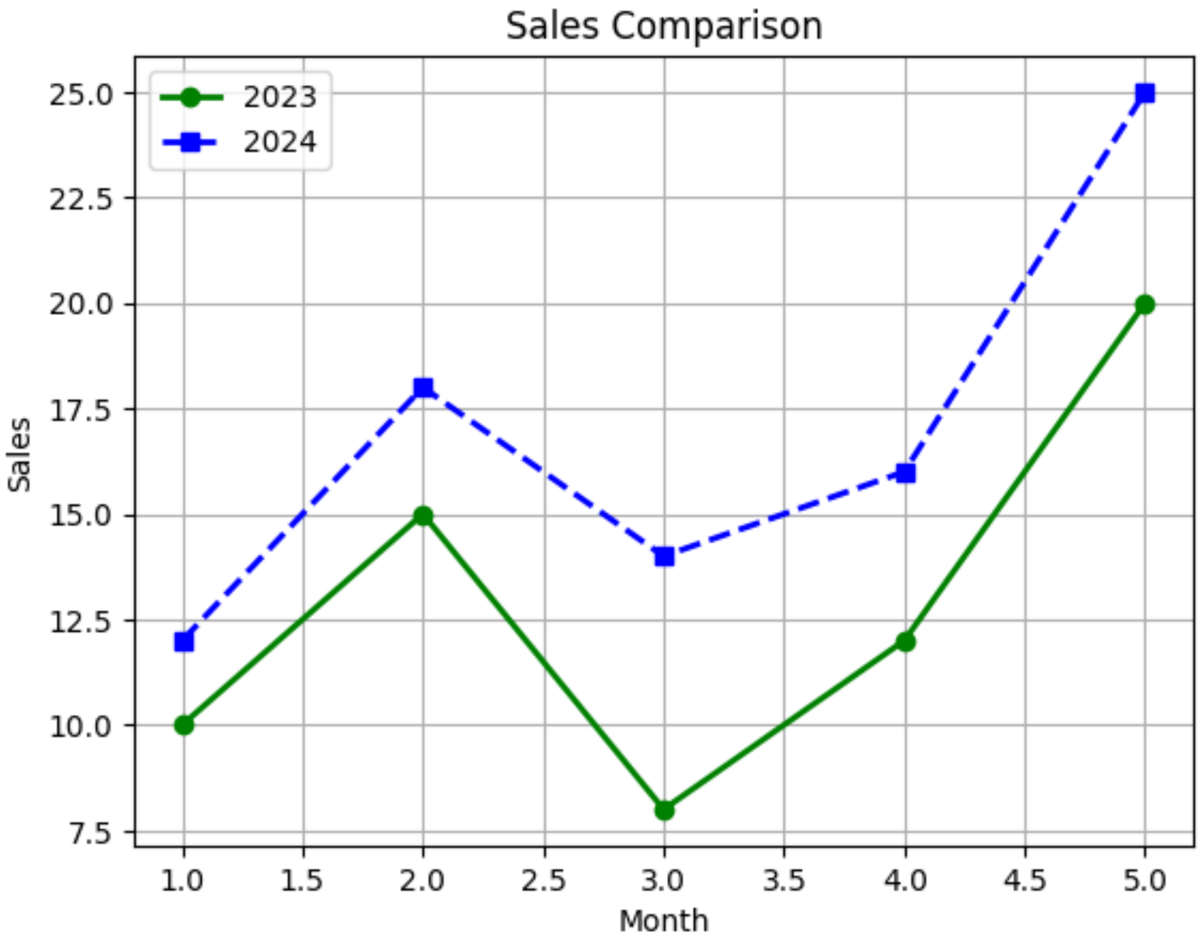
```
import pandas as pd
import matplotlib.pyplot as plt

# Dataset
data = {
    'Month': [1, 2, 3, 4, 5],
    'Sales': [10, 15, 8, 12, 20]
}
df = pd.DataFrame(data)

# Styled line
# Red dashed line, circular markers, thicker line

plt.plot(df['Month'], df['Sales'],
         color='blue', marker='*', linestyle='-', linewidth=2)

plt.title("Styled Sales Line")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.grid(True)
```

```
    plt.show()
```



Example 5 — Multiple Styled Lines

```
# Dataset
data = {
    'Month': [1, 2, 3, 4, 5],
    'Sales_2023': [10, 15, 8, 12, 20],
    'Sales_2024': [12, 18, 14, 16, 25]
}
df = pd.DataFrame(data)

# Two styled lines
# Green solid line with circles vs Blue dashed line with squares
plt.plot(df['Month'], df['Sales_2023'], 'g-o', linewidth=2, label='2023
plt.plot(df['Month'], df['Sales_2024'], 'b--s', linewidth=2, label='202

plt.title("Sales Comparison")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.legend()
plt.grid(True)

plt.show()
```



3.5. Subplots and Multiple Axes (plt.subplots())

- To show multiple plots in a single figure side by side or in grid layout.
- Each individual plot lives inside its own Axes inside the same Figure.

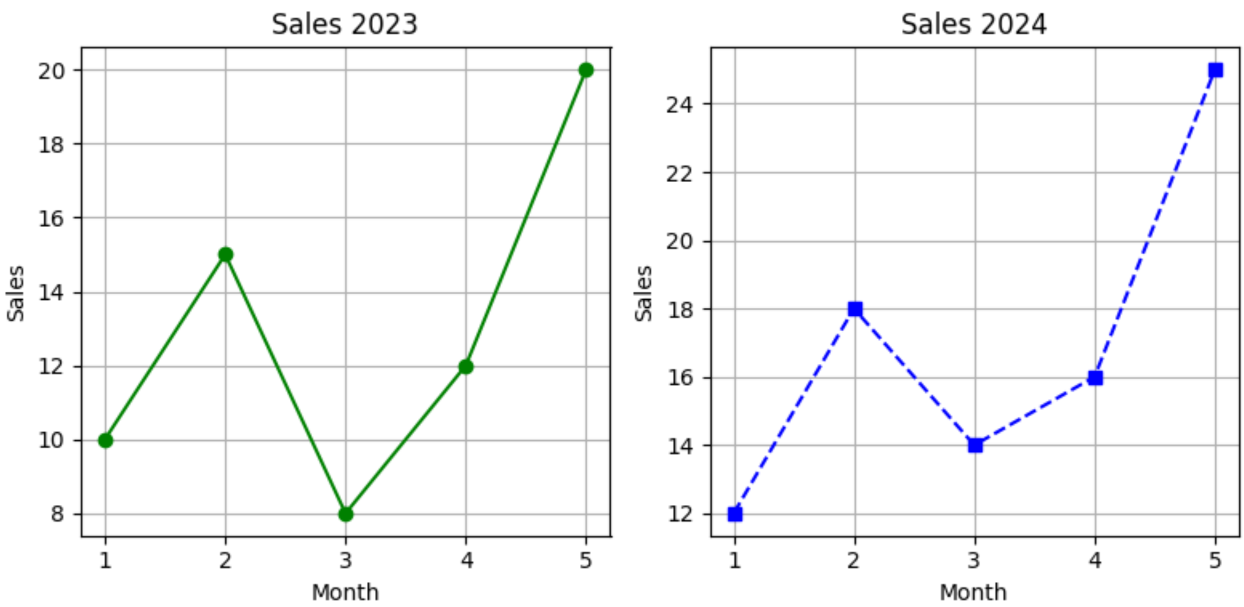
Key Concepts

- plt.subplots(rows, cols) → creates a grid of Axes
- Returns:
  - fig → the full figure
  - ax → one Axes (if 1 plot) or a NumPy array of Axes (if multiple)
- You can plot on each Axes separately using ax[i].plot(...)

Example 6 — Two Plots Side by Side

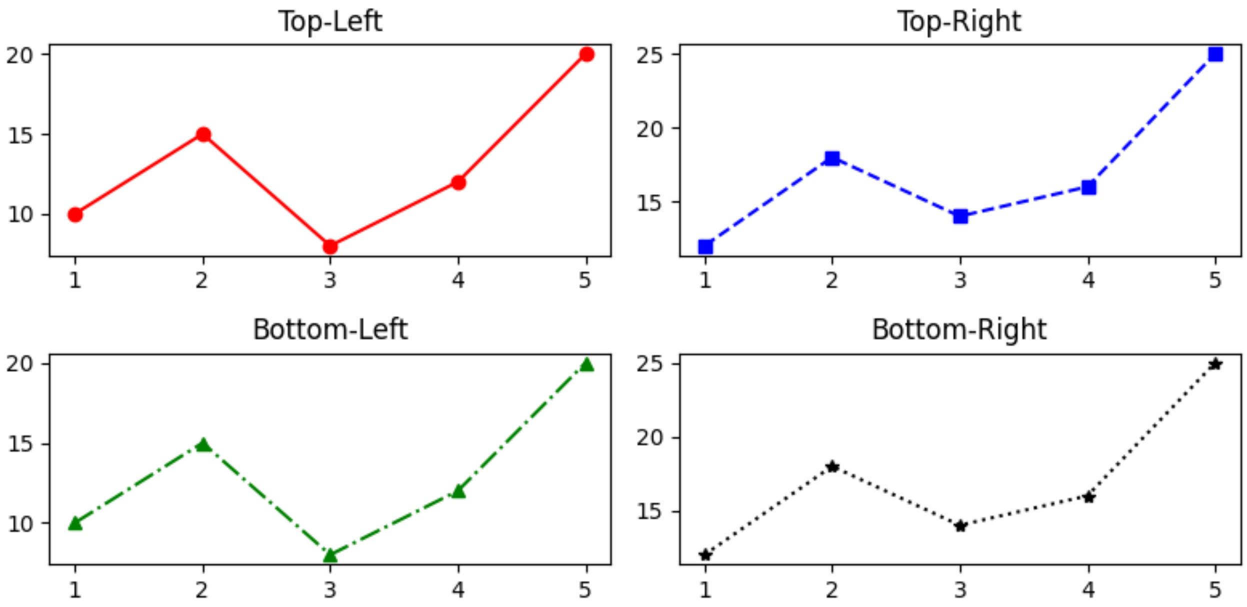
```
import pandas as pd
import matplotlib.pyplot as plt

# Sample dataset
data = {
    'Month': [1, 2, 3, 4, 5],
    'Sales_2023': [10, 15, 8, 12, 20],
    'Sales_2024': [12, 18, 14, 16, 25]
}
df = pd.DataFrame(data)

# Create 1 row, 2 columns of subplots
fig, ax = plt.subplots(1, 2, figsize=(8, 4))

# Left subplot
ax[0].plot(df['Month'], df['Sales_2023'], 'g-o', label='2023')
ax[0].set_title("Sales 2023")
ax[0].set_xlabel("Month")
ax[0].set_ylabel("Sales")
ax[0].grid(True)

# Right subplot
ax[1].plot(df['Month'], df['Sales_2024'], 'b--s', label='2024')
ax[1].set_title("Sales 2024")
ax[1].set_xlabel("Month")
ax[1].set_ylabel("Sales")
ax[1].grid(True)
```

```
plt.tight_layout()    # fixes spacing
plt.show()
```



Example 7 — 2x2 Grid of Subplots

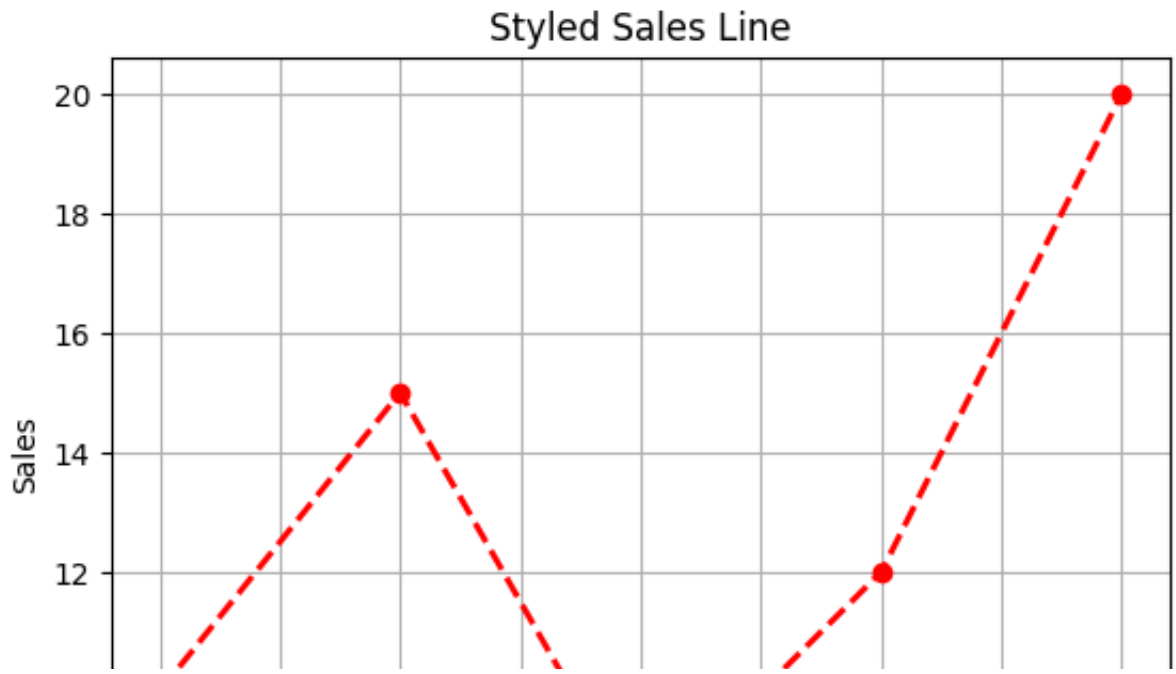
```
# 2 rows × 2 columns
fig, ax = plt.subplots(2, 2, figsize=(8, 4))

# Access using [row][col]
ax[0][0].plot(df['Month'], df['Sales_2023'], 'r-o')
ax[0][0].set_title("Top-Left")

ax[0][1].plot(df['Month'], df['Sales_2024'], 'b--s')
ax[0][1].set_title("Top-Right")

ax[1][0].plot(df['Month'], df['Sales_2023'], 'g-.^')
ax[1][0].set_title("Bottom-Left")

ax[1][1].plot(df['Month'], df['Sales_2024'], 'k:*')
ax[1][1].set_title("Bottom-Right")

plt.tight_layout()

plt.show()
```

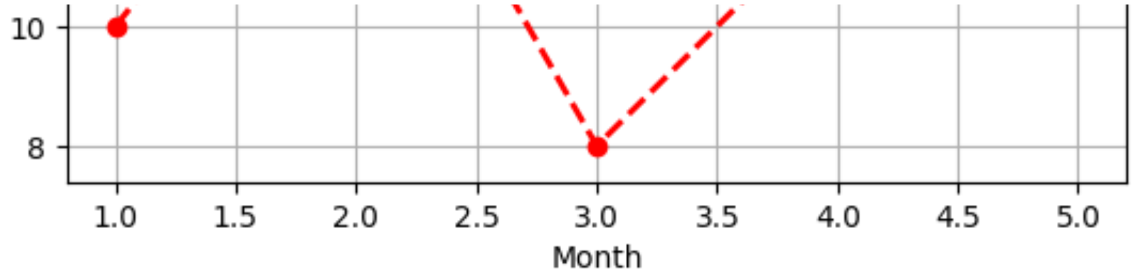


3.6. Saving Figures (plt.savefig())

Example 8 — Save Plots

```
import pandas as pd
import matplotlib.pyplot as plt

# Dataset
data = {
    'Month': [1, 2, 3, 4, 5],
    'Sales': [10, 15, 8, 12, 20]
}
df = pd.DataFrame(data)

# Styled line
# Red dashed line, circular markers, thicker line

plt.plot(df['Month'], df['Sales'],
         color='red', marker='o', linestyle='--', linewidth=2)

plt.title("Styled Sales Line")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.grid(True)

# Save to file
plt.savefig("sales_plot.png")

plt.show()
```





4. Seaborn — Core Concepts

4.1. Figure and Axes Objects in Seaborn

- Seaborn is built on top of Matplotlib.
- Any Seaborn plot is actually drawn inside a Matplotlib Axes object.
- So we can still use:
  - plt.figure() or
  - fig, ax = plt.subplots()
- to control the figure size and layout.

Two Ways to Plot

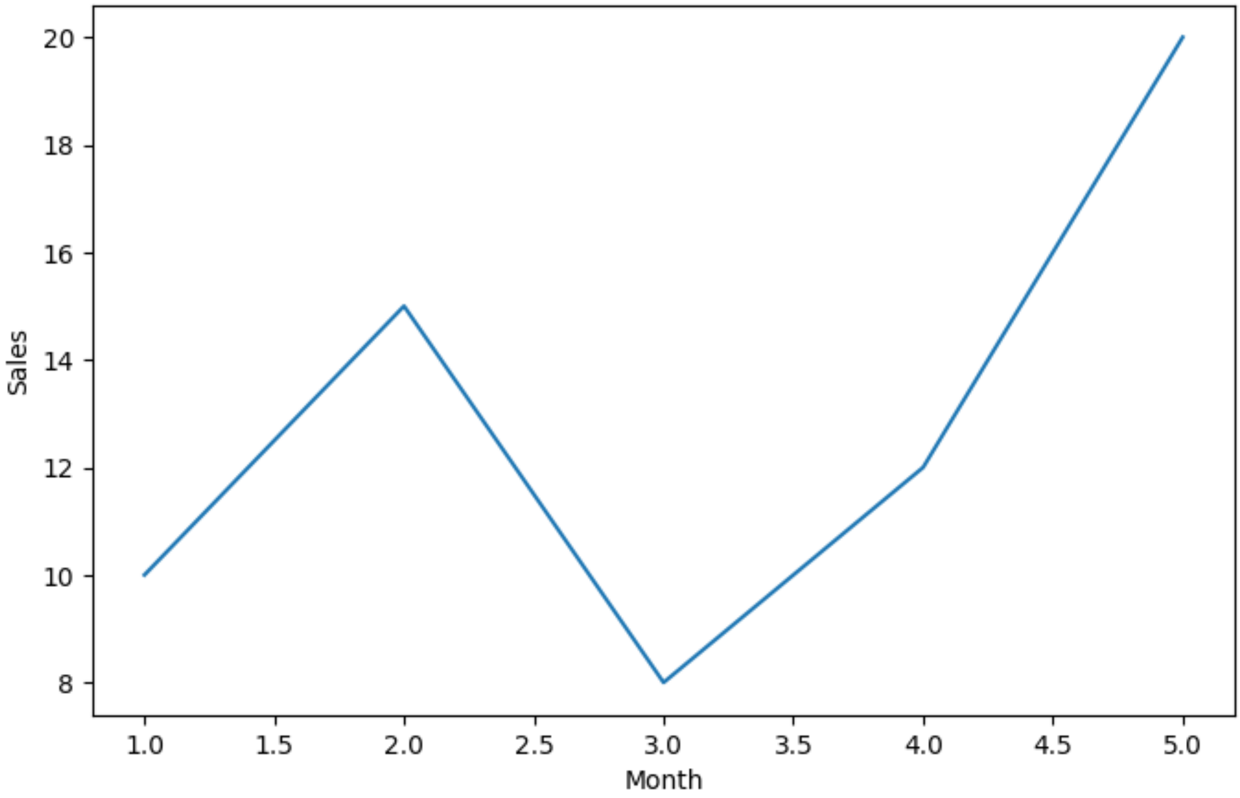
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Sample dataset
data = {
    'Month': [1, 2, 3, 4, 5],
    'Sales': [10, 15, 8, 12, 20]
}

df = pd.DataFrame(data)
df
```

```
   Month  Sales
0      1     10
1      2     15
2      3      8
3      4     12
4      5     20
```

Next steps: Generate code with df New interactive sheet

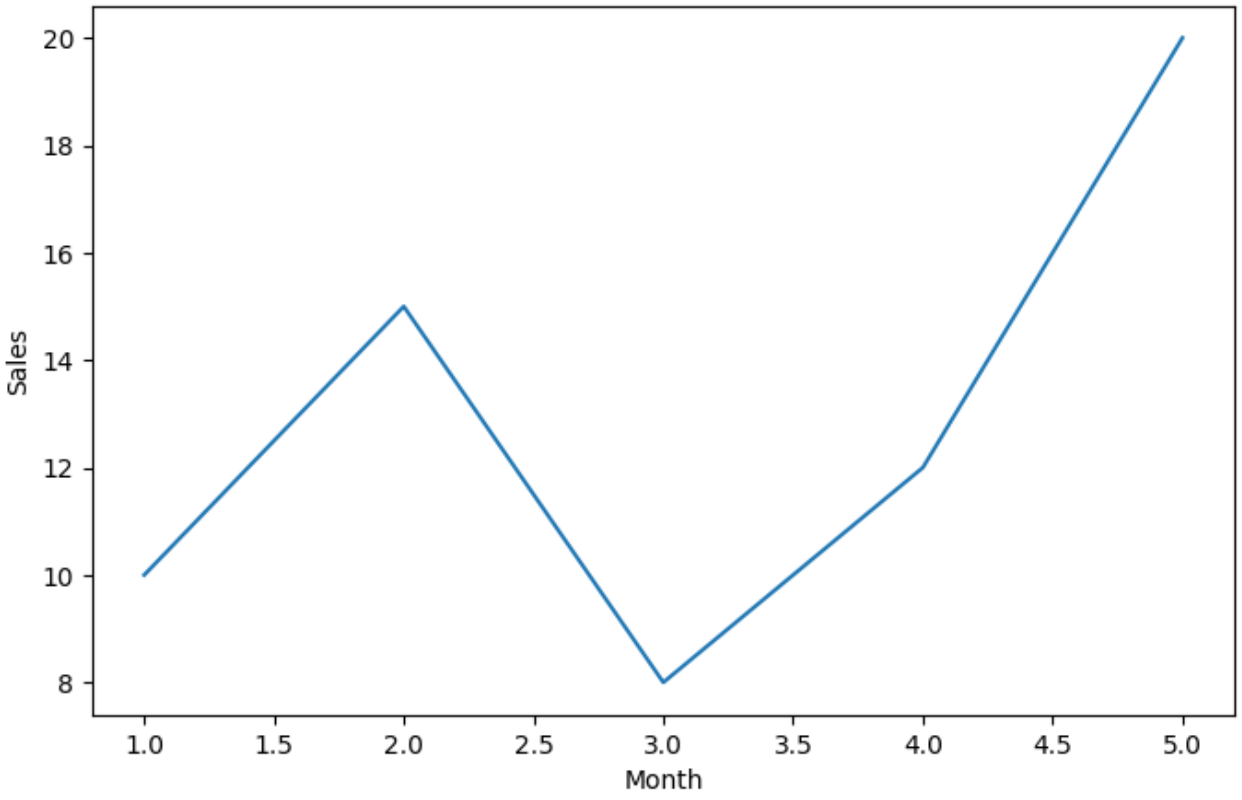
1) State-based (Simple) way using plt

```
# Plot using plt style

plt.figure(figsize=(8, 5))
sns.lineplot(data=df, x='Month', y='Sales')
plt.show()
```



2) Object-oriented way using fig, ax

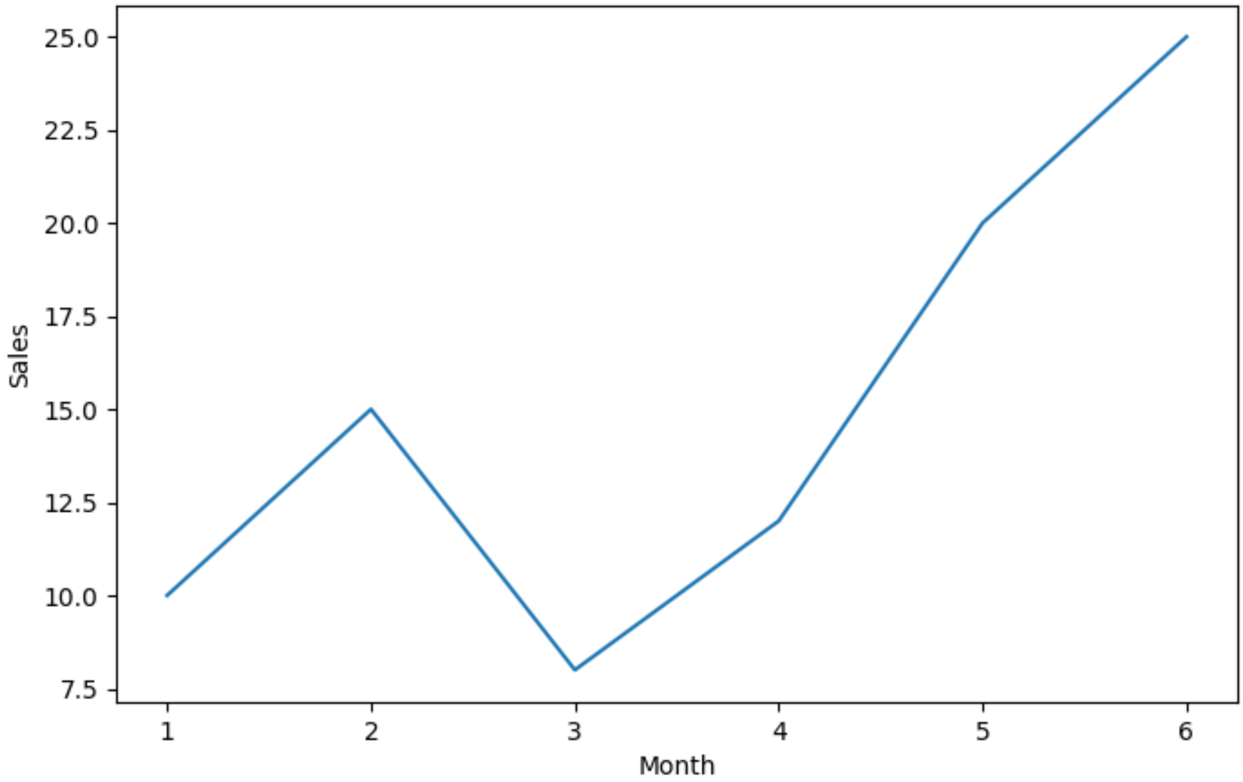
```
# Plot using fig, ax style

myfig, myax = plt.subplots(figsize=(8, 5))

sns.lineplot(data=df, x='Month', y='Sales', ax=myax)

plt.show()
```



4.2. Creating Simple Line Plots (sns.lineplot())

Example 1 — Line Plot

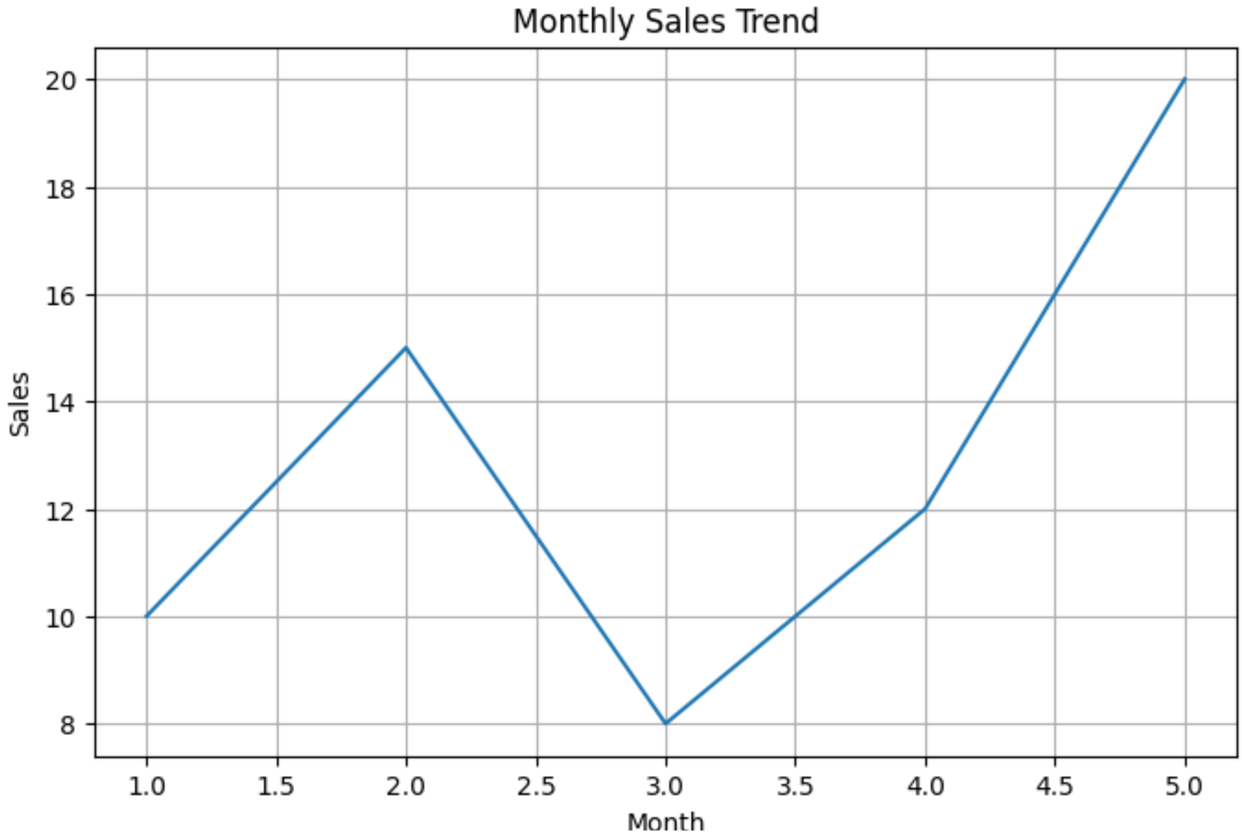
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Sample dataset
data = {
    'Month': [1, 2, 3, 4, 5,6],
    'Sales': [10, 15, 8, 12, 20,25]
}
df = pd.DataFrame(data)

# Plot using fig, ax style

fig, ax = plt.subplots(figsize=(8, 5))
sns.lineplot(data=df, x='Month', y='Sales', ax=ax)
plt.show()
```



4.3. Adding Labels, Titles, Legends, and Grids

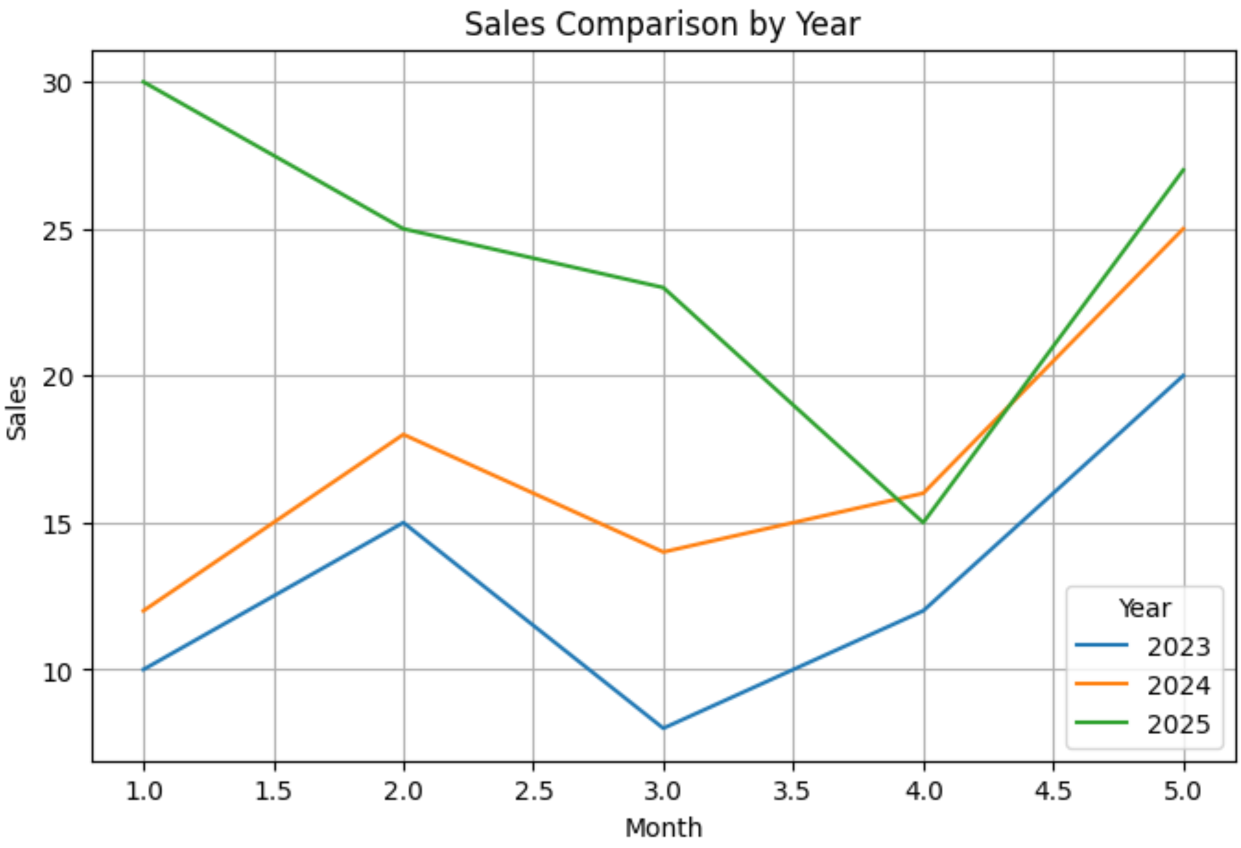
Example 2 — Single Line with Labels, Title, Grid

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

# Dataset
data = {
    'Month': [1, 2, 3, 4, 5],
    'Sales': [10, 15, 8, 12, 20]
}
df = pd.DataFrame(data)

fig, ax = plt.subplots(figsize=(8, 5))

# Seaborn line plot
ax = sns.lineplot(data=df, x='Month', y='Sales',ax=ax)

# Add labels, title, grid
ax.set_title("Monthly Sales Trend")
ax.set_xlabel("Month")
ax.set_ylabel("Sales")
ax.grid(True)

plt.show()
```



Example 3 — Two Lines (using hue) + Legend

```
# Dataset
data = {
    'Month': [1, 2, 3, 4, 5]*3,
    'Year': ['2023']*5 + ['2024']*5 + ['2025']*5,
    'Sales': [10, 15, 8, 12, 20, 12, 18, 14, 16, 25, 30, 25, 23, 15, 27
}
df = pd.DataFrame(data)
# print(df)

fig, ax = plt.subplots(figsize=(8, 5))

ax = sns.lineplot(data=df, x='Month', y='Sales', ax=ax , hue='Year')

# Add labels, title, grid, legend
ax.set_title("Sales Comparison by Year")
ax.legend(title="Year")
ax.grid(True)

plt.show()
```



4.4. Styling (palette, hue, linewidth, markers)

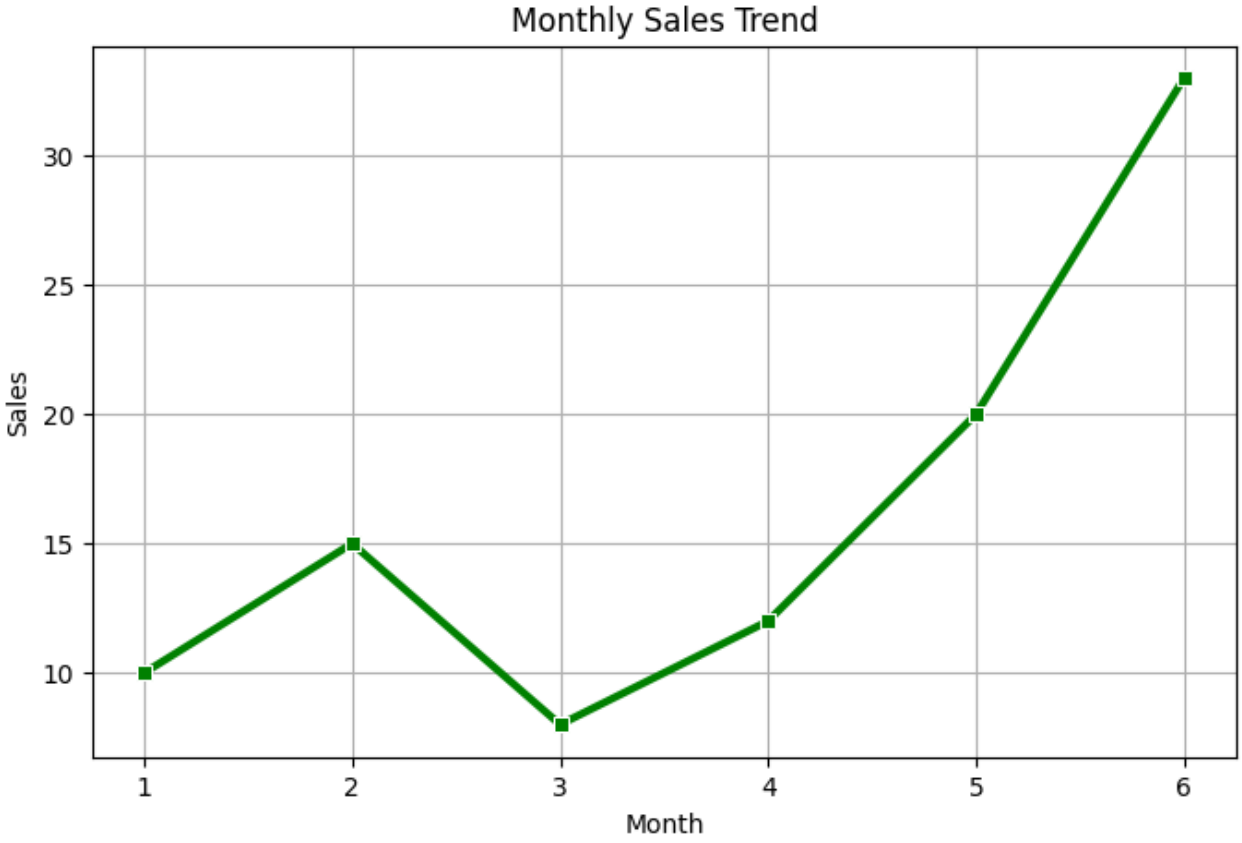
- You can apply styles to make your line plots more clear, readable, and visually appealing.
- This helps differentiate multiple lines on the same axes.

Common Styling Parameters

- **Parameter** ----- Purpose ----- Example
- **hue** ----- Separate lines by category ----- hue='Year'
- **palette** ----- Set color palette ----- 'deep', 'pastel', 'dark', 'colorblind'
- **linewidth** ----- Control line thickness ----- linewidth=3
- **markers** ----- Add markers to each point ----- markers=True
- **style** ----- Different line styles for categories ----- style='Year'

Example 4 — Styling One Line

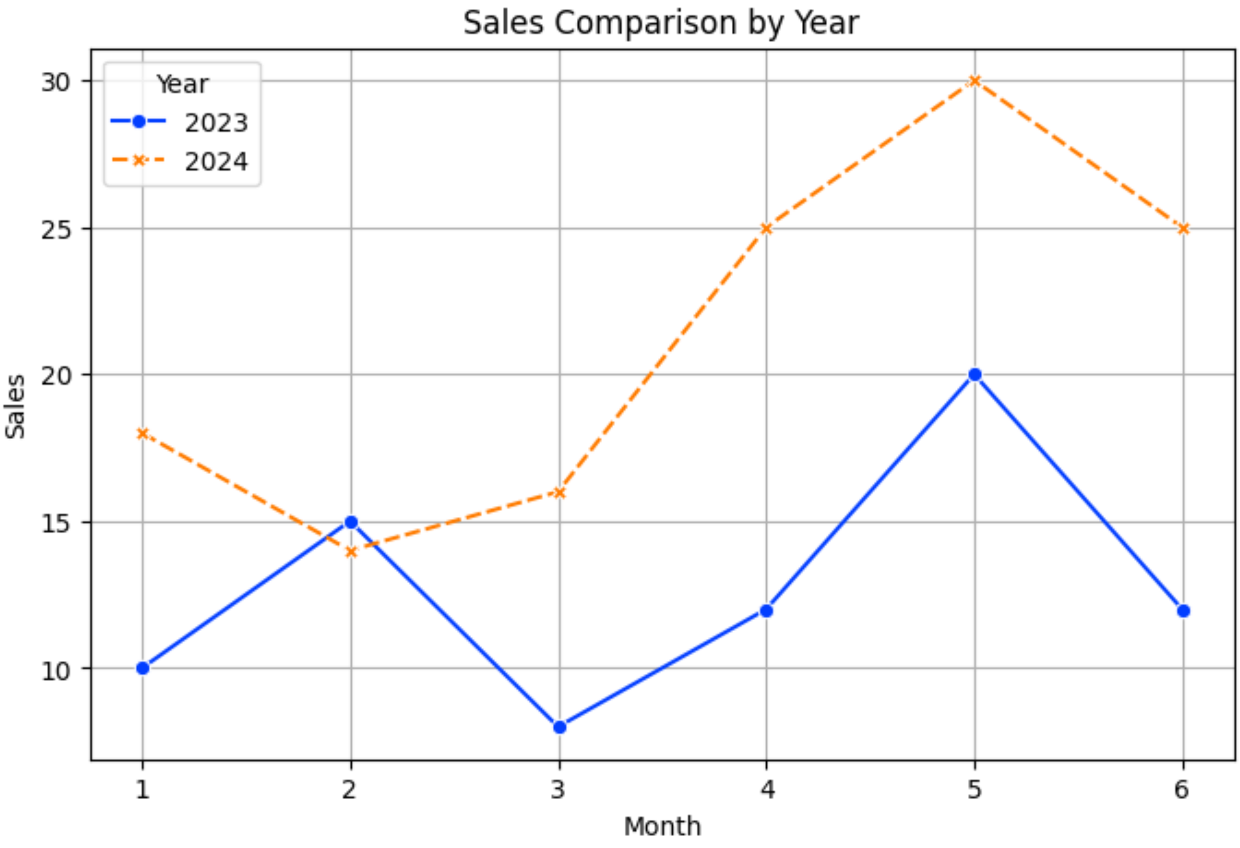
```
mydf = pd.DataFrame({
    'Month': [1, 2, 3, 4, 5, 6],
    'Sales': [10, 15, 8, 12, 20, 33]
})
plt.figure(figsize=(8,5))
sns.lineplot(data=mydf, x='Month', y='Sales', color='green',  marker='s

plt.title("Monthly Sales Trend")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.grid(True)
plt.show()
```



Example 5 — Multiple Styled Lines

- Qualitative (categorical): deep, muted, pastel, bright, dark, colorblind
- Continuous (numeric): viridis, magma, plasma, coolwarm, cividis
    - **sns.set_palette("deep")** # set globally

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

# Dataset
data = {
    'Month': [1, 2, 3, 4, 5, 6 ]*2,
    'Year': ['2023']*6 + ['2024']*6,
    'Sales': [10, 15, 8, 12, 20, 12, 18, 14, 16, 25, 30,25]
}
df = pd.DataFrame(data)

plt.figure(figsize=(8,5))

sns.lineplot(data=df, x='Month', y='Sales', hue='Year',
    style='Year', markers=True,  linewidth=1.5,  palette='bright' )

plt.title("Sales Comparison by Year")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.grid(True)
plt.show()
```



4.5. Subplots and Multiple Axes

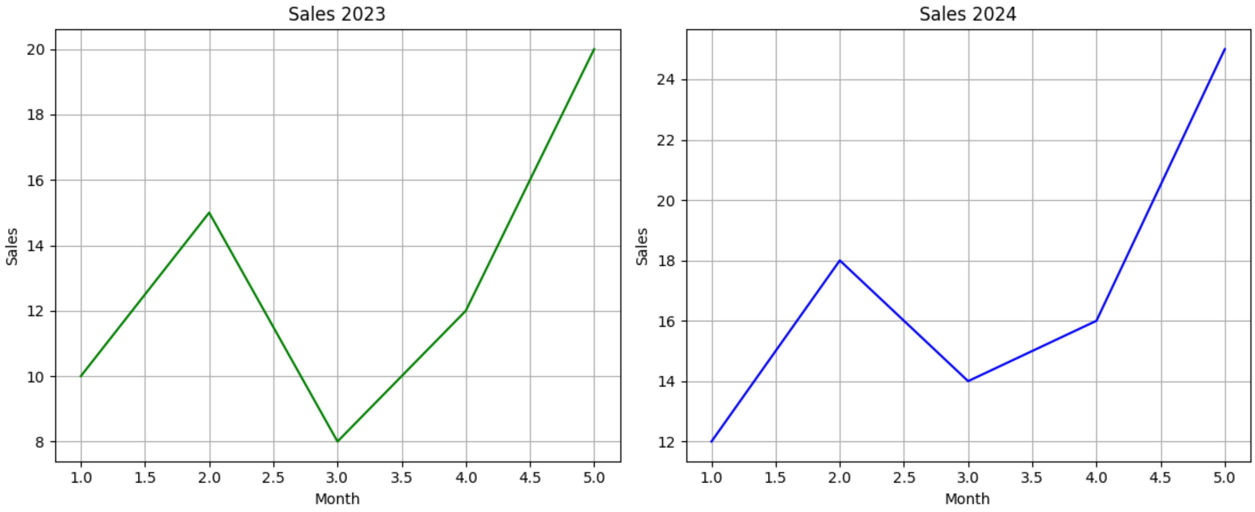
- To show multiple plots in a single figure side by side or in grid layout.
- Each individual plot lives inside its own Axes inside the same Figure.

Key Concepts

- plt.subplots(rows, cols) → creates a grid of Axes
- **Returns:**
    - fig → the full figure
    - ax → one Axes (if 1 plot) or a NumPy array of Axes (if multiple)
- You can plot on each Axes separately using ax[i].plot(...)

Example 6 — Two Plots Side by Side

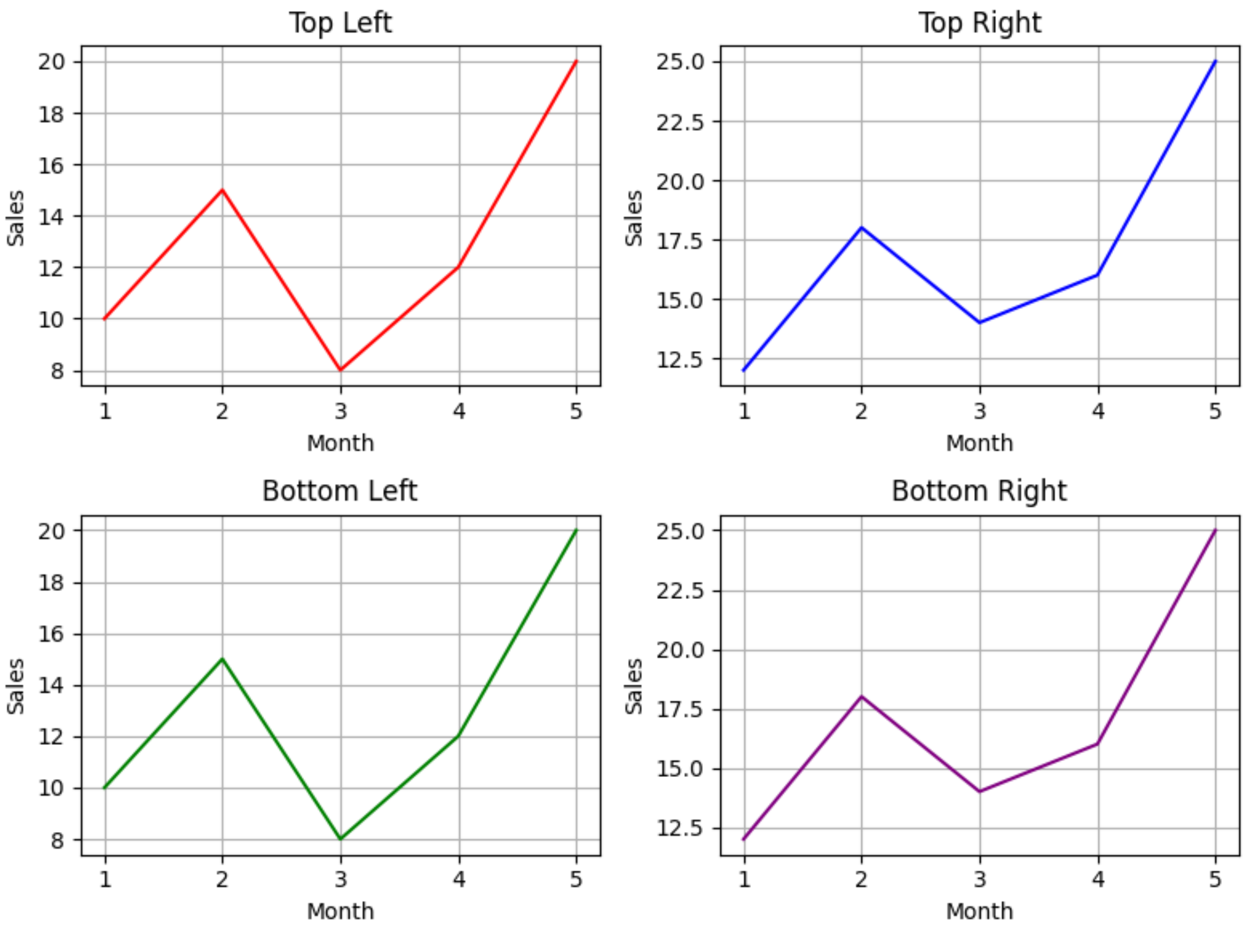
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Dataset
data = {
    'Month': [1, 2, 3, 4, 5]*2,
    'Year': ['2023']*5 + ['2024']*5,
    'Sales': [10, 15, 8, 12, 20, 12, 18, 14, 16, 25]
}
df = pd.DataFrame(data)

# Create 1 row, 2 columns of subplots
fig, ax = plt.subplots(1, 2, figsize=(12, 5))

# Left subplot: 2023 data
sns.lineplot(data=df[df['Year']=='2023'], x='Month', y='Sales', color='
ax[0].set_title("Sales 2023")
ax[0].grid(True)

# Right subplot: 2024 data
sns.lineplot(data=df[df['Year']=='2024'], x='Month', y='Sales', color='
ax[1].set_title("Sales 2024")
ax[1].grid(True)

plt.tight_layout()
plt.show()
```



Example 7 — 2x2 Grid of Subplots

```
fig, ax = plt.subplots(2, 2, figsize=(8, 6))

sns.lineplot(data=df[df['Year']=='2023'], x='Month', y='Sales', color='
ax[0][0].set_title("Top Left")
ax[0][0].grid(True)

sns.lineplot(data=df[df['Year']=='2024'], x='Month', y='Sales', color='
ax[0][1].set_title("Top Right")
ax[0][1].grid(True)

sns.lineplot(data=df[df['Year']=='2023'], x='Month', y='Sales', color='
ax[1][0].set_title("Bottom Left")
ax[1][0].grid(True)

sns.lineplot(data=df[df['Year']=='2024'], x='Month', y='Sales', color='
ax[1][1].set_title("Bottom Right")
ax[1][1].grid(True)

plt.tight_layout()
```

```
    plt.show()
```



4.6. Saving Figures (plt.savefig())

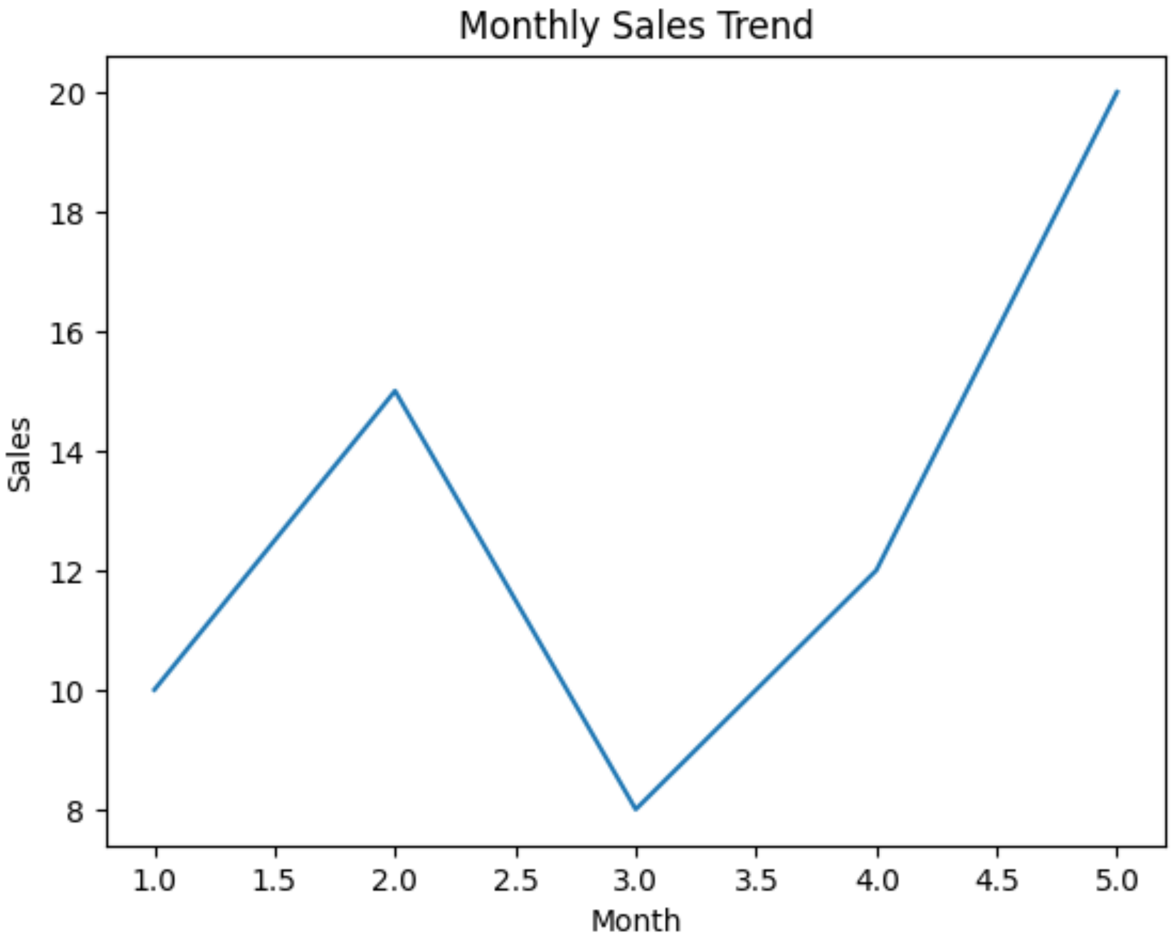
- To export your plots as image files (PNG, JPG, PDF, etc.)
- useful for reports, presentations, and documentation.

Key Points

- **plt.savefig("filename")** saves the current figure
- Can save in many formats: .png, .jpg, .pdf, .svg, etc.
- Works for both state-based (plt) and OO (fig) plots

Example 8 —Saving Figures

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

# Dataset
data = {
    'Month': [1, 2, 3, 4, 5],
    'Sales': [10, 15, 8, 12, 20]
}
df = pd.DataFrame(data)

# Create plot
sns.lineplot(data=df, x='Month', y='Sales')
plt.title("Monthly Sales Trend")

# Save to file
plt.savefig("mysales_plot.png")

plt.show()
```



5. Exploring Scatter Plot

A) Understanding Scatter Plot

- Scatter plot shows the relationship between two numeric columns.
- Each dot represents one row (one observation) from the dataset.
- If the dots form a pattern (line or curve), it suggests there is a clear relationship between the two variables.
- If the **dots do not form a pattern** (line or curve), it suggests there is a clear **NO** relationship between the two variables.

It helps us:

- Detect positive or negative relationships
- Judge the strength and direction of correlation (strong/weak, positive/negative)
- Spot clusters or natural groupings
- Identify outliers (unusual points far from the rest)

Key Points:

- Used for Bivariate Analysis
- Only for Numerical Data,
- Works only for Numerical vs Numerical data
- Not suitable for categorical data
- Order of data points doesn’t matter

B) Scatter Plot — Example

1) Create the Dataset

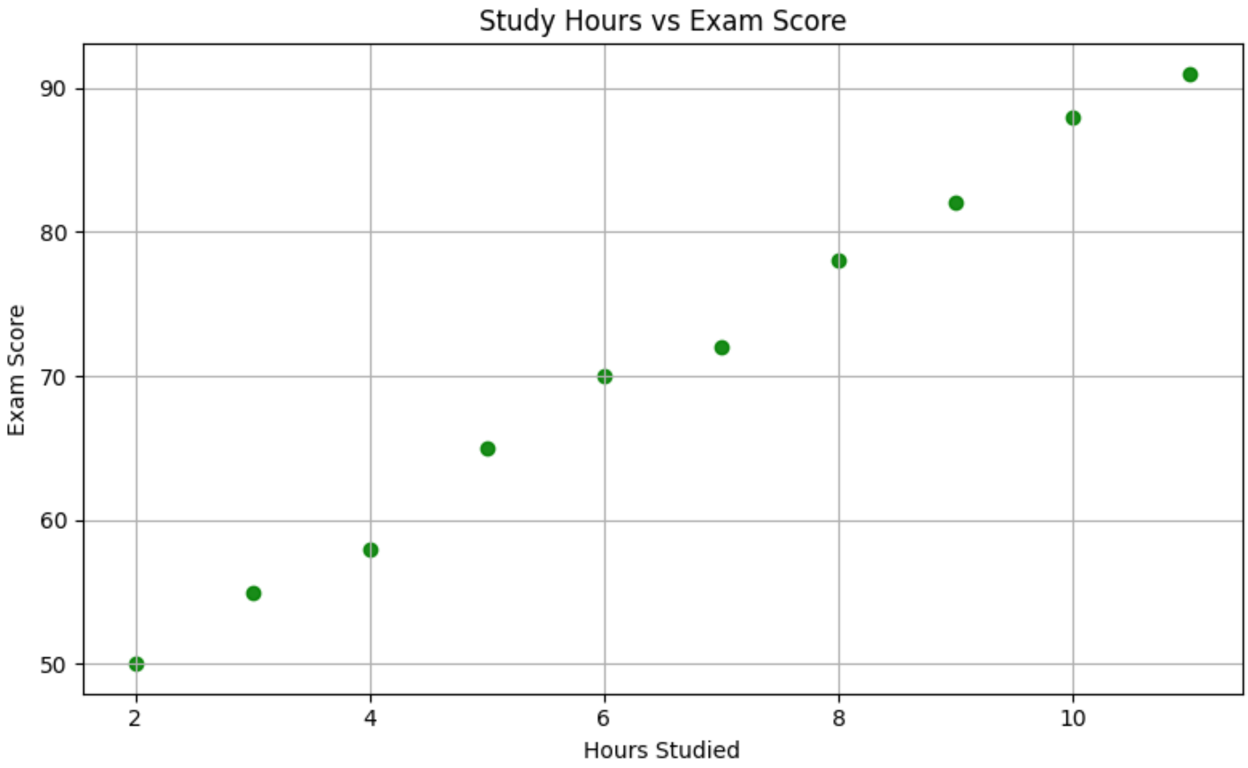
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = {
    'Hours_Studied': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    'Exam_Score':    [50, 55, 58, 65, 70, 72, 78, 82, 88, 91]
}

df = pd.DataFrame(data)

df
```

```
   Hours_Studied  Exam_Score
0              2          50
1              3          55
2              4          58
3              5          65
4              6          70
5              7          72
6              8          78
7              9          82
8             10          88
9             11          91
```

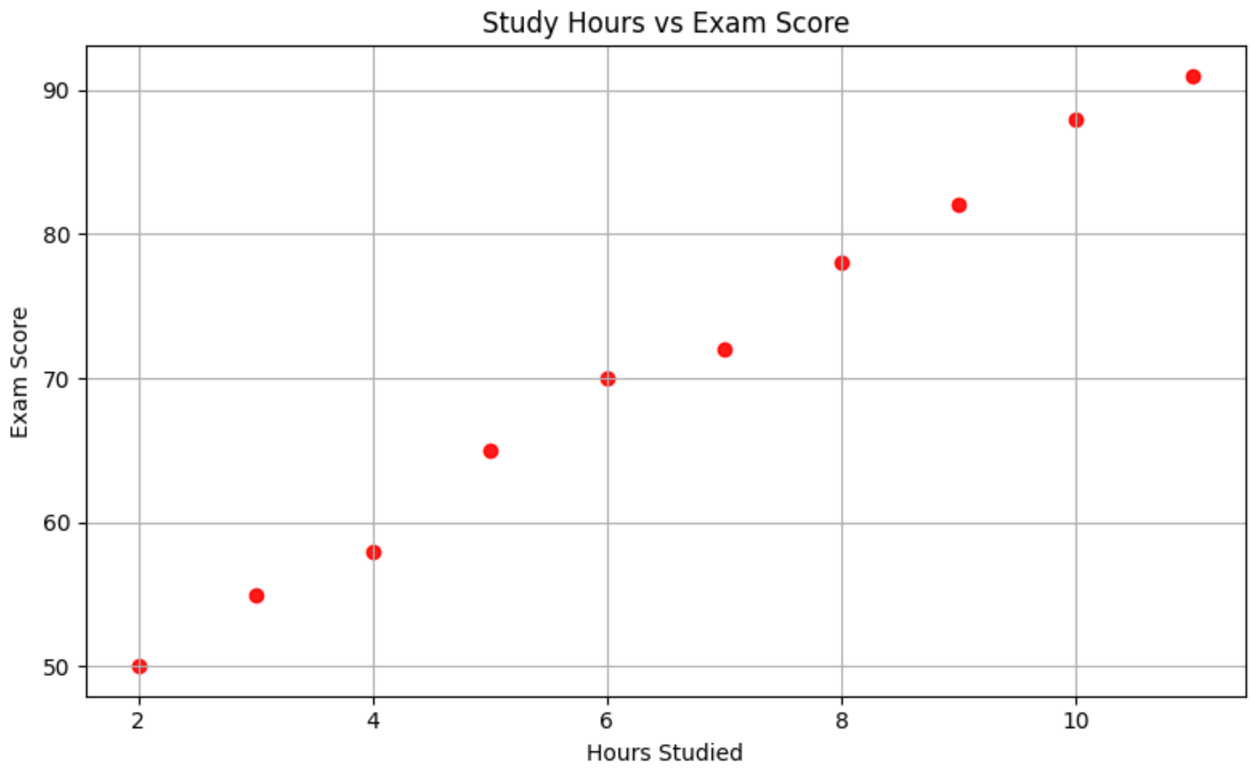
Next steps: Generate code with df New interactive sheet

3) Scatter Plot using Matplotlib

```
# pyplot style
plt.figure(figsize=(8, 5))

plt.scatter(df['Hours_Studied'], df['Exam_Score'], alpha=0.9, color='gr

plt.title("Study Hours vs Exam Score")
plt.xlabel("Hours Studied");
plt.ylabel("Exam Score")
plt.grid(True);

plt.tight_layout();
plt.show()
```

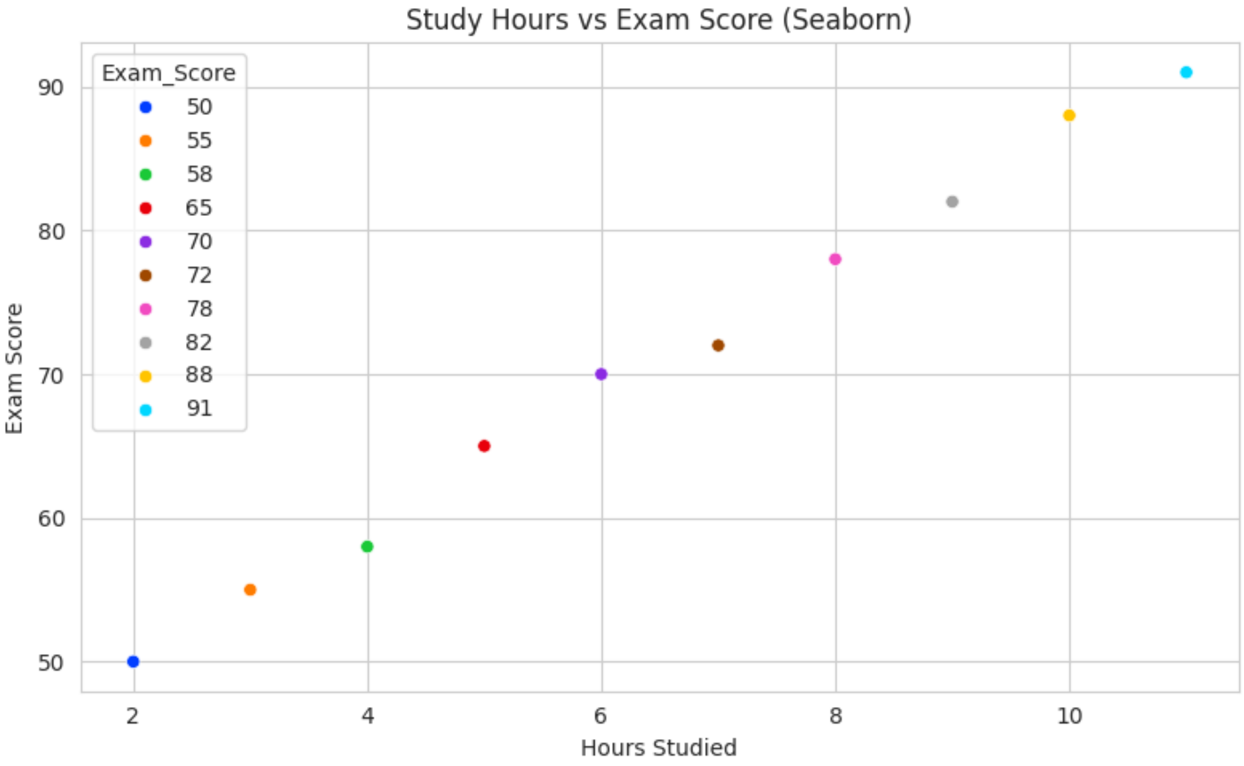


```
# OO style (recommended)
fig, ax = plt.subplots(figsize=(8, 5))

ax.scatter(df['Hours_Studied'], df['Exam_Score'], alpha=0.9,color='red'
ax.set_title("Study Hours vs Exam Score")
ax.set_xlabel("Hours Studied")
ax.set_ylabel("Exam Score")
ax.grid(True)

plt.tight_layout()
plt.show()
```



4) Scatter Plot using Seaborn

```
# Theme background
sns.set_style("whitegrid")

# Set global color palette
sns.set_palette("deep")

fig, ax = plt.subplots(figsize=(8, 5))
sns.scatterplot( data=df,  x='Hours_Studied', y='Exam_Score',
                hue='Exam_Score',  palette='bright',
                sizes=(50, 200), ax=ax
)

ax.set_title("Study Hours vs Exam Score (Seaborn)")
ax.set_xlabel("Hours Studied")
ax.set_ylabel("Exam Score")

plt.tight_layout()
plt.show()
```



C) Exploring Correlation

- Correlation is a statistical term that shows how strongly two numeric variables are related to each other and in which direction they move together.

Correlation tells us:

- Direction → Do they move in the same or opposite direction?
- Strength → How closely the points follow a straight line

Measured by:

- Correlation Coefficient (r) with the Range: -1 to +1
  - +1 = Perfect positive
  - -1 = Perfect negative
  - 0 = No relationship

Visualized using:

- Scatter Plot (most common tool)

Types of Correlation

1. Positive Correlation

- When one variable increases, the other also increases
- Both move in the same direction
- Scatter plot shows upward trend (bottom-left → top-right)

Examples

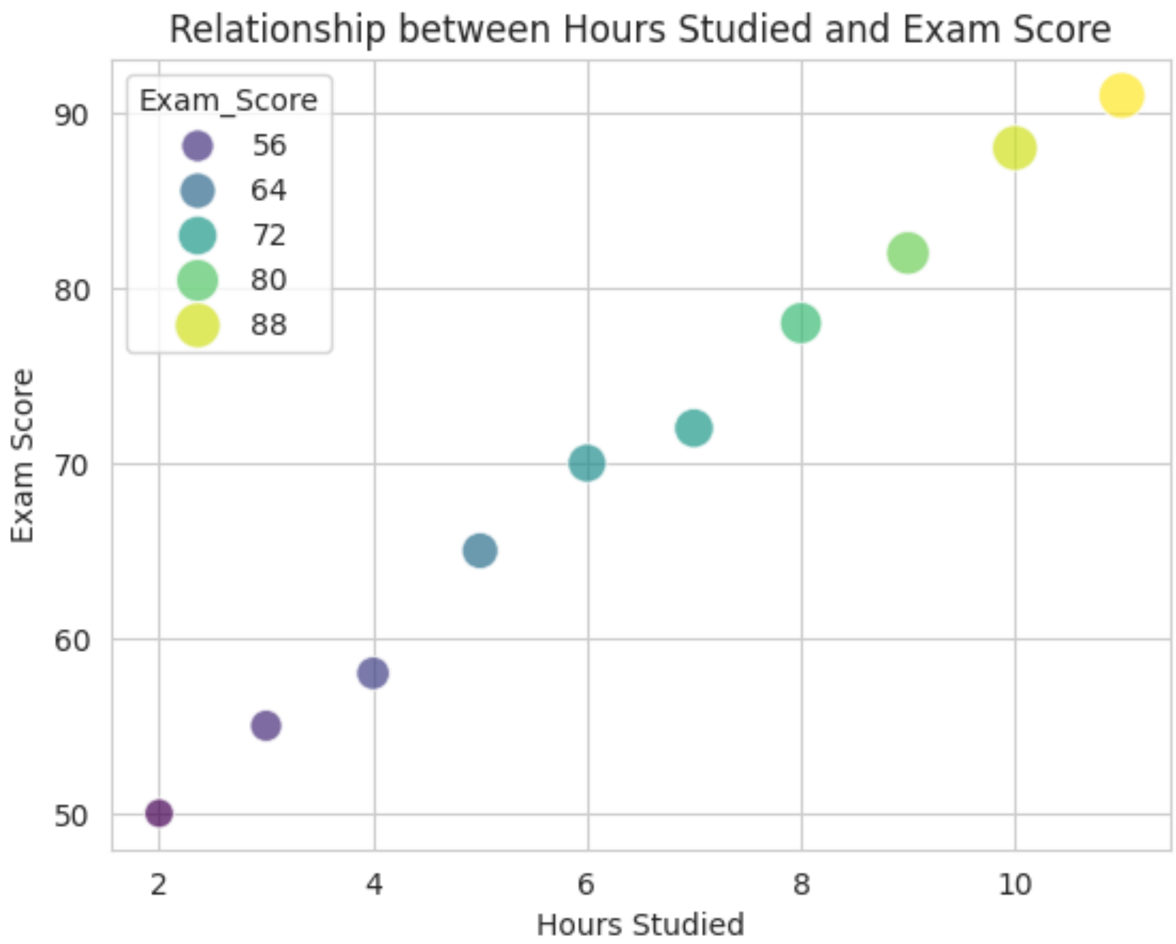
- More Hours Studied → Higher Exam Score
- More Experience → Higher Salary
- **Correlation value (r): Close to +1**

```
# Positive Correlation
data1 = {
    'Hours_Studied': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    'Exam_Score':    [50, 55, 58, 65, 70, 72, 78, 82, 88, 91]
}

df = pd.DataFrame(data1)

df.head()
```

```
   Hours_Studied  Exam_Score
0              2          50
1              3          55
2              4          58
3              5          65
4              6          70
```

Next steps: Generate code with df New interactive sheet

```
sns.scatterplot(
    data=df,
    x='Hours_Studied',
    y='Exam_Score',
    hue='Exam_Score',      # color depends on score
    size='Exam_Score',     # point size depends on score
    palette='viridis',
    sizes=(100, 250),
    alpha=0.7
)

plt.title("Relationship between Hours Studied and Exam Score")
plt.xlabel("Hours Studied")
plt.ylabel("Exam Score")

plt.show()
```



2. Negative Correlation

- When one variable increases, the other decreases
- They move in opposite directions
- Scatter plot shows downward trend (top-left → bottom-right)

Examples

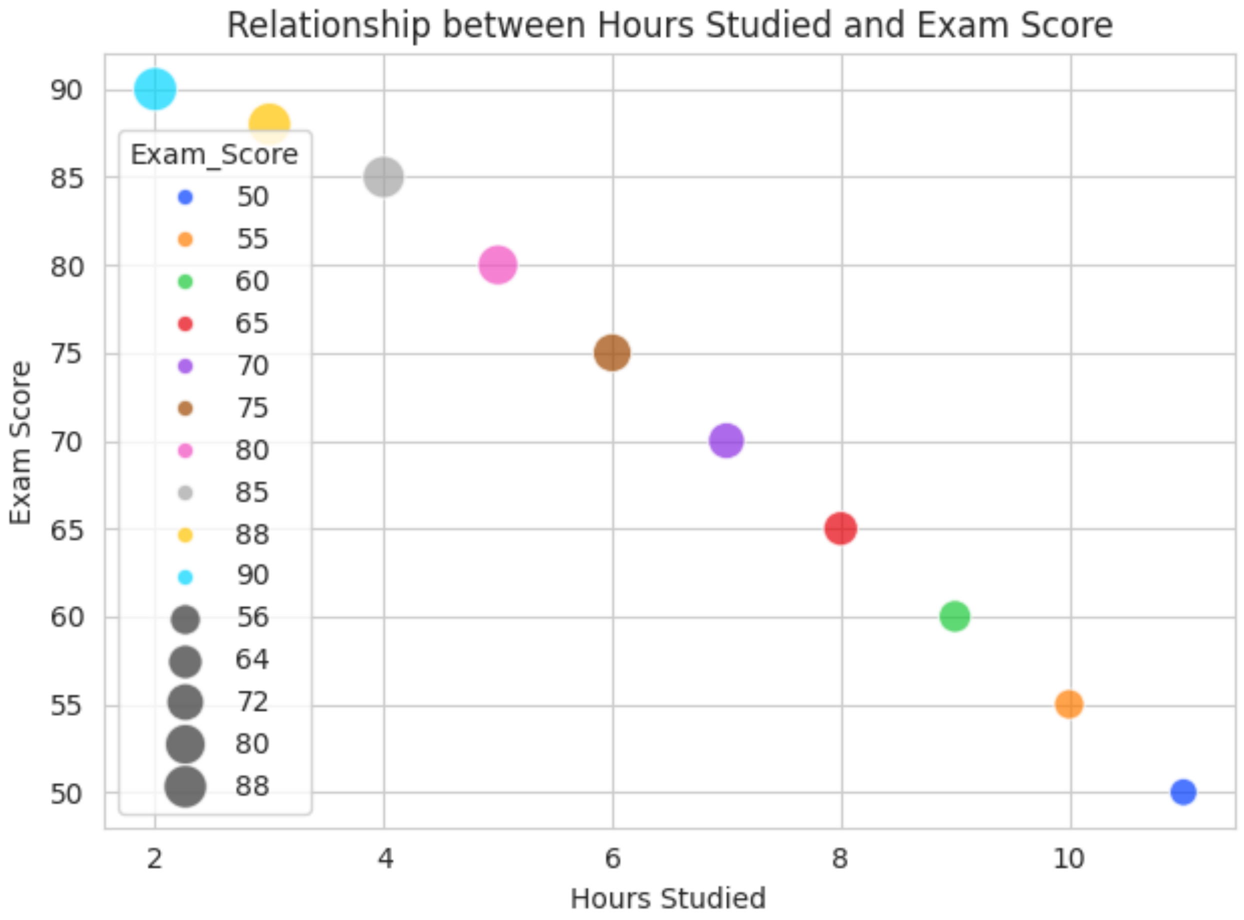
- More Hours Studied → Lower Exam Score (stress, burnout)
- More Distance → Less Fuel Efficiency
- **Correlation value (r): Close to –1**

```
# Negative Correlation
data2 = {
    'Hours_Studied': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    'Exam_Score':    [90, 88, 85, 80, 75, 70, 65, 60, 55, 50]
}

df = pd.DataFrame(data2)
df.head()
```

	Hours_Studied	Exam_Score
0	2	90
1	3	88
2	4	85
3	5	80
4	6	75

Next steps: Generate code with df New interactive sheet

```
sns.scatterplot(
    data=df,
    x='Hours_Studied',
    y='Exam_Score',
    hue='Exam_Score',      # color depends on score
    size='Exam_Score',     # point size depends on score
    palette='bright',
    sizes=(100, 250),
    alpha=0.7
)

plt.title("Relationship between Hours Studied and Exam Score")
plt.xlabel("Hours Studied")
plt.ylabel("Exam Score")

plt.tight_layout()

plt.show()
```



3. No Correlation

- Variables do not affect each other
- No clear pattern or trend on the scatter plot
- Points are scattered randomly

Examples

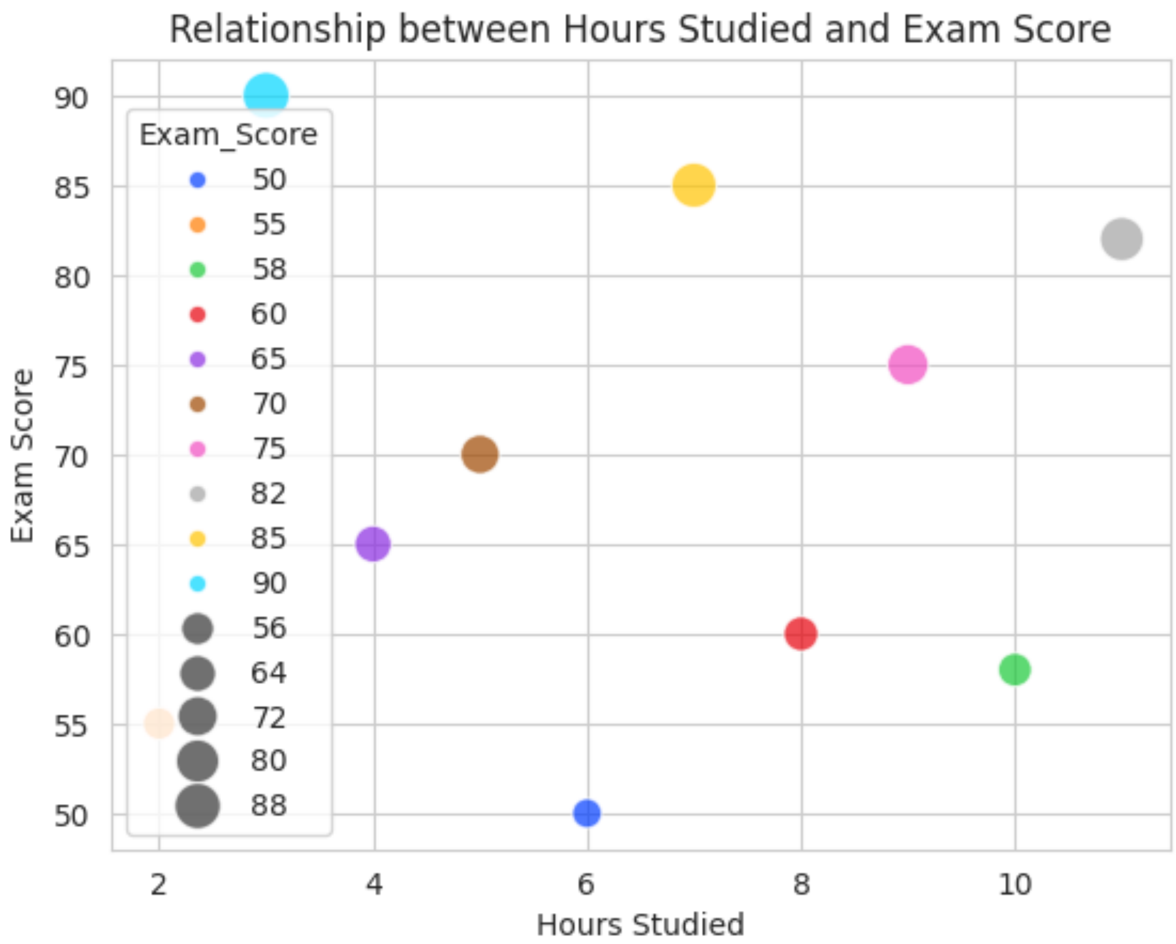
- Shoe Size vs Exam Score
- Number of Pets vs Salary
- **Correlation value (r): Around 0**

```
# No Correlation
data3 = {
    'Hours_Studied': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    'Exam_Score':    [55, 90, 65, 70, 50, 85, 60, 75, 58, 82]
}


df = pd.DataFrame(data3)

df.head()
```

	Hours_Studied	Exam_Score
0	2	55
1	3	90
2	4	65
3	5	70
4	6	50

Next steps: Generate code with df New interactive sheet

```
    sns.scatterplot(
        data=df,
        x='Hours_Studied',
        y='Exam_Score',
        hue='Exam_Score',      # color depends on score
        size='Exam_Score',     # point size depends on score
        palette='bright',
        sizes=(100, 250),
        alpha=0.7
    )

    plt.title("Relationship between Hours Studied and Exam Score")
    plt.xlabel("Hours Studied")
    plt.ylabel("Exam Score")

    plt.show()
```



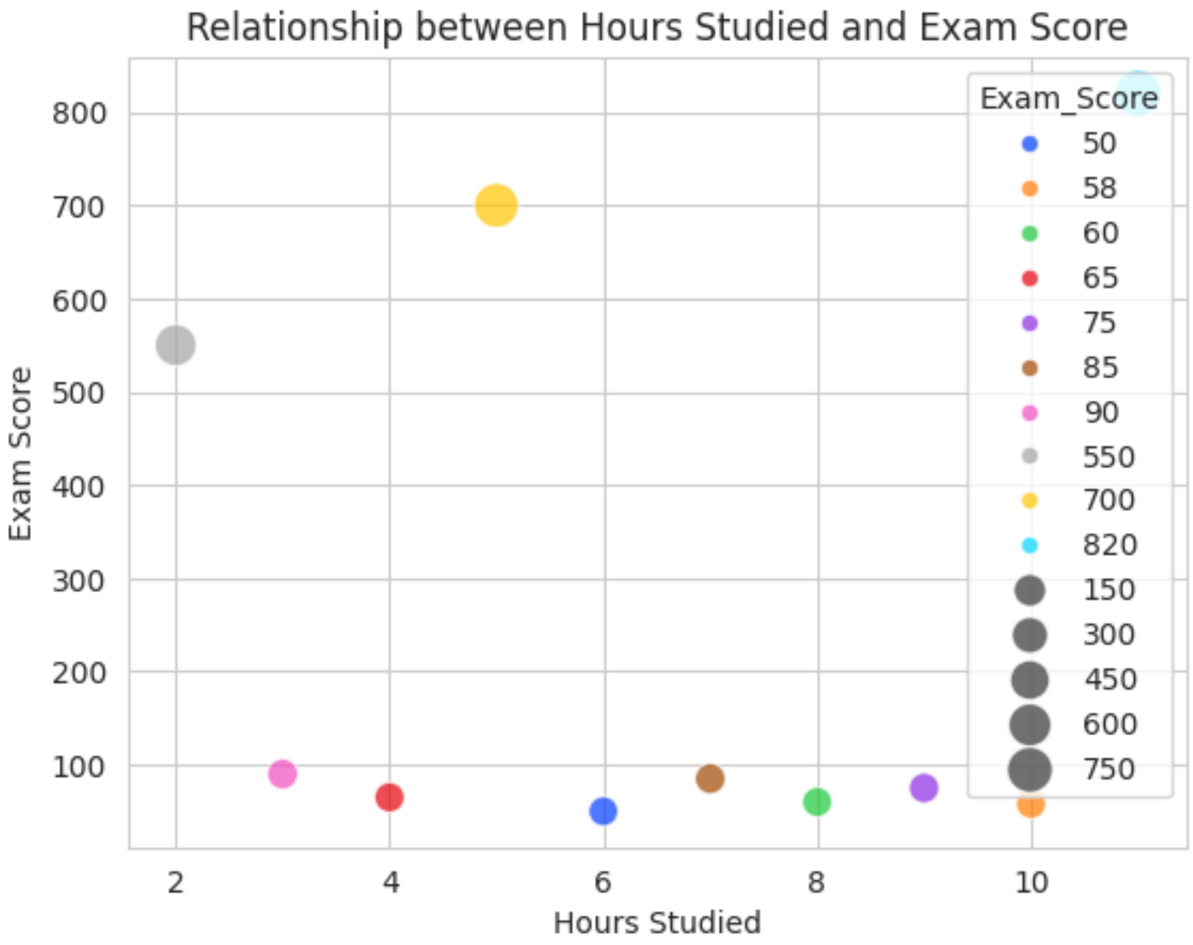
Detecting Outliers

```
# No Correlation
data4 = {
    'Hours_Studied': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    'Exam_Score':    [550, 90, 65, 700, 50, 85, 60, 75, 58, 820]
}


df = pd.DataFrame(data4)

df.head()
```

	Hours_Studied	Exam_Score
0	2	550
1	3	90
2	4	65
3	5	700
4	6	50

Next steps: Generate code with df New interactive sheet

```
    sns.scatterplot(
        data=df,
        x='Hours_Studied',
        y='Exam_Score',
        hue='Exam_Score',      # color depends on score
        size='Exam_Score',     # point size depends on score
        palette='bright',
        sizes=(100, 250),
        alpha=0.7
    )

    plt.title("Relationship between Hours Studied and Exam Score")
    plt.xlabel("Hours Studied")
    plt.ylabel("Exam Score")

    plt.show()
```



D) Real-Time Use-Cases of Scatter Plots

1) Correlation Analysis

Goal:

- Check if two continuous variables are related.

Examples:

- Hours studied vs Exam scores (Education)
- Advertising spend vs Sales (Marketing)
- Temperature vs Ice cream sales (Retail)

Why scatter plot?

- Quickly shows positive, negative, or no correlation patterns.

2) Detecting Clusters or Groups

Goal:

- See if data naturally forms groups.

Examples:

- Customer Age vs Spending → see low/mid/high spend segments
- Income vs Loan Amount → cluster different borrower profiles
- Product Price vs Quantity sold → cluster popular vs premium products

Why scatter plot?

- Helps decide if clustering algorithms (like KMeans) are meaningful.

3) Identifying Outliers / Anomalies

Goal:

- Find unusual points far away from others.

Examples:

- Quality vs Defect Rate in Manufacturing → find defective batches
- Transaction amount vs Frequency → spot fraudulent transactions
- Website visits vs Purchases → find bot-like behavior

4) Comparing Two Metrics Across Categories

Goal:

- Visualize how two metrics vary across groups.

Examples:

- Sales vs Profit colored by Region
- Height vs Weight separated by Gender
- Engine Size vs Mileage grouped by Fuel Type

5) Evaluating Model Predictions (Regression)

Goal:

- Compare actual vs predicted values.

Examples:

- True House Prices vs Predicted Prices
- Actual vs Predicted Stock Prices

6) Performance / Trend Diagnostics

Goal:

- Understand system or process behaviour

Examples:

- CPU usage vs Response time (DevOps monitoring)
- Load vs Latency (System Design/Performance)
- Employees’ experience vs productivity score (HR analytics)

6. Exploring Line Plots