Complete Matplotlib Guide

Data Visualization with Python

BY NAVEEN DHAWAN

Matplotlib

🎨 What is Matplotlib?

- **Matplotlib** is a popular Python library for **data visualisation**.
- It helps you create different types of **charts, plots, and graphs** to understand data better.
- Module used:

```
import matplotlib.pyplot as plt
```

⚡ Features of Matplotlib

- Wide variety of plots (line, bar, scatter, histogram, etc.)
- Highly customizable (colors, labels, styles, etc.)
- Works well with **NumPy and Pandas**
- Can export to images (PNG, JPG) and PDFs
- Interactive plots with **Jupyter Notebook**

📝 Basic Example

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]
plt.plot(x, y) # line plot
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Simple Line Chart")
plt.show()
```

📊 Types of Charts in Matplotlib

Below is a **list of important charts/graphs,** their definition, and when to use them:

1) Line Chart

```
plt.plot(x, y)
```

Definition: Shows data as points connected by straight lines.
Use Case: Best for showing trends over time (e.g., stock price, sales growth).

2) Bar Chart

```
plt.bar(x, y)
```

Definition: Uses rectangular bars to compare values.
Use Case: Comparing categories (e.g., sales in different regions).

3) Horizontal Bar Chart

```
plt.barh(x, y)
```

Definition: Similar to bar chart but horizontal.
Use Case: When category names are long, or comparison looks better horizontally.

4) Histogram

```
plt.hist(data, bins=10)
```

Definition: Groups data into ranges (bins) and shows frequency.
Use Case: Understanding distribution of data (e.g., marks of students, age distribution).

5) Scatter Plot

```
plt.scatter(x, y)
```

Definition: Displays individual data points.
Use Case: To find a relationship/correlation between two variables (e.g., hours studied vs. marks scored).

6) Pie Chart

```
plt.pie(values, labels=categories, autopct='%1.1f%%')
```

Definition: Circular chart divided into slices.
Use Case: Showing proportion/percentage (e.g., market share of companies).

7) Stacked Bar Chart

```
plt.bar(x, y1, label="A")
plt.bar(x, y2, bottom=y1, label="B")
```

Definition: Bars stacked on top of each other.
Use Case: Comparing multiple categories within each group (e.g., sales by product type across different regions).

8) Area Chart

```
plt.fill_between(x, y)
```

Definition: Line chart with the area below filled.
Use Case: Emphasising the magnitude of change over time (e.g., cumulative sales, population growth).

9) Box Plot

```
plt.boxplot(data)
```

Definition: Shows data distribution using quartiles and outliers.
Use Case: Understanding data spread and identifying outliers (e.g., salary distribution, test scores).

10) Heatmap

```
plt.imshow(data, cmap='hot')
```

Definition: Uses colors to represent data values in a matrix.
Use Case: Showing correlation between variables or patterns in 2D data (e.g., correlation matrix, time vs. day analysis).

11) Stack Plot

```
plt.stackplot(x, y1, y2, y3, labels=['A', 'B', 'C'])
```

Definition: Shows multiple datasets stacked on top of each other as filled areas.
Use Case: Displaying cumulative data or parts of a whole over time (e.g., budget allocation over years, revenue by product categories).

12) Multiple Line Chart

```
plt.plot(x, y1, label='Line 1')
plt.plot(x, y2, label='Line 2')
plt.plot(x, y3, label='Line 3')
plt.legend()
```

Definition: Multiple lines plotted on the same chart for comparison.
Use Case: Comparing trends of different categories over time (e.g., sales performance of multiple products, temperature in different cities).

13) Bubble Chart

```
plt.scatter(x, y, s=sizes, alpha=0.5)
```

Definition: A Scatter plot where the size of each point represents a third variable.
Use Case: Showing relationships between three variables simultaneously (e.g., company revenue vs profit with bubble size representing number of employees).

📋 Quick Reference Summary

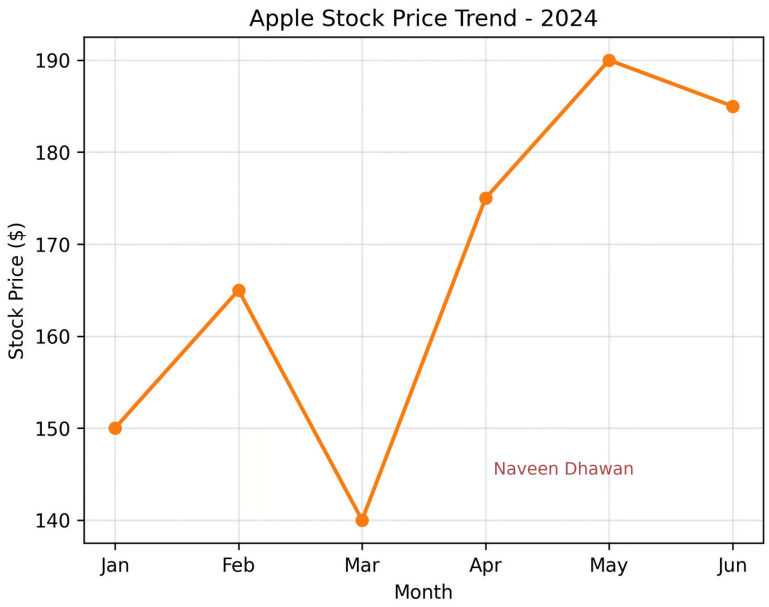
ChartType	BestUsed For
Line Chart	Trends over time
Bar Chart	Comparing categories
Horizontal Bar Chart	Long category names or horizontal comparison
Histogram	Data distribution and frequency
Scatter Plot	Correlation between two variables
Pie Chart	Proportions and percentages
Stacked Bar Chart	Multiple categories within groups
Area Chart	Magnitude of change over time
Box Plot	Data spread and outliers
Heatmap	Correlation matrix and 2D patterns
Stack Plot	Cumulative data over time
Multiple Line Chart	Comparing multiple trends
Bubble Chart	Three variables simultaneously

💻 Matplotlib Examples - Real-Life Use Cases

1) Line Chart - Stock Price Analysis

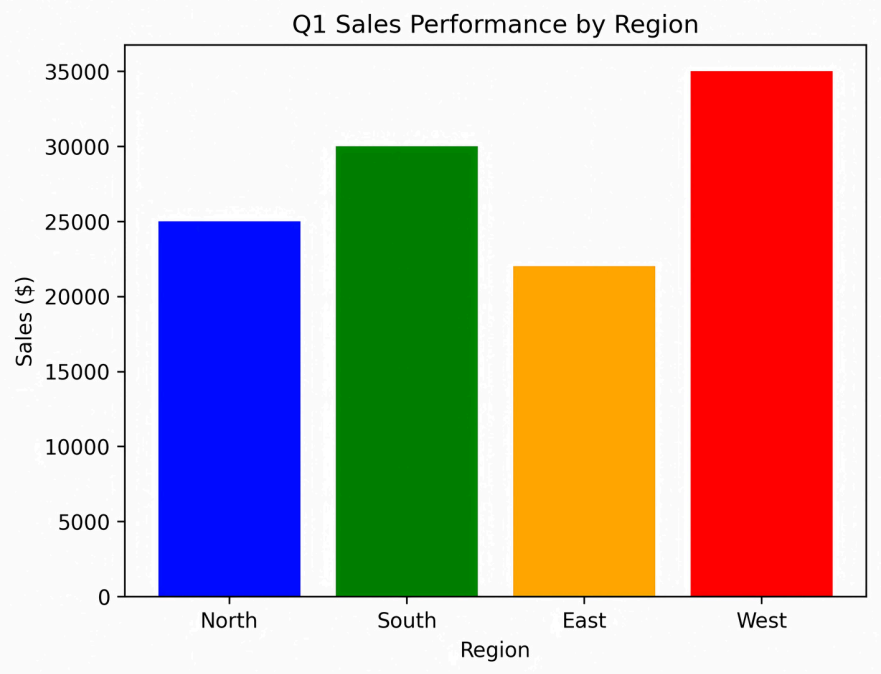
```
import matplotlib.pyplot as plt
import pandas as pd

# Stock price data
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
stock_price = [150, 165, 140, 175, 190, 185]

plt.plot(months, stock_price, marker='o', linewidth=2, color="C1")
```

```
plt.text(4.2, 145, "Naveen Dhawan",
         fontsize=9, color="darkred", ha="right", alpha=0.7)
plt.xlabel("Month")
plt.ylabel("Stock Price ($)")
plt.title("Apple Stock Price Trend - 2024")
plt.grid(True, alpha=0.3)
plt.show()
```



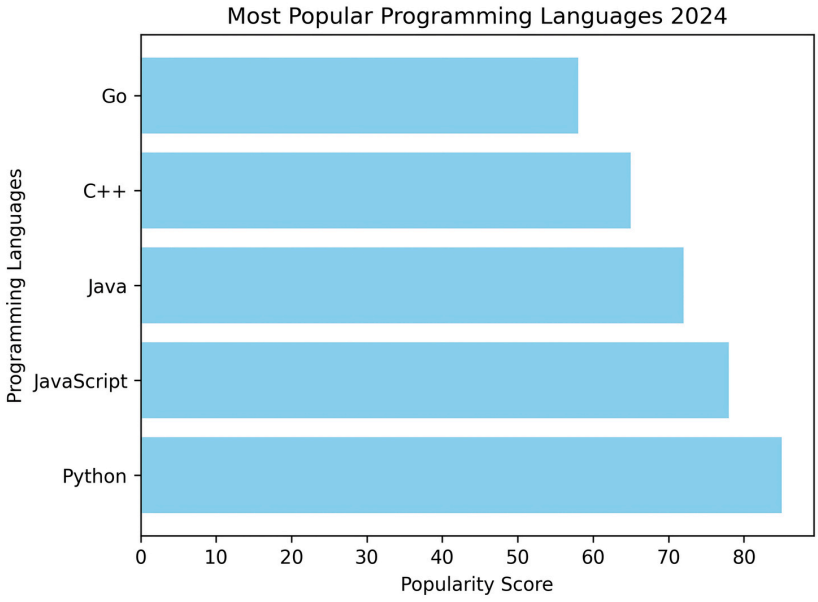
2) Bar Chart - Sales Performance by Region

```
regions = ['North', 'South', 'East', 'West']
sales = [25000, 30000, 22000, 35000]

plt.bar(regions, sales, color=['blue', 'green', 'orange', 'red'])
plt.xlabel("Region")
plt.ylabel("Sales ($)")
plt.title("Q1 Sales Performance by Region")
plt.show()
```



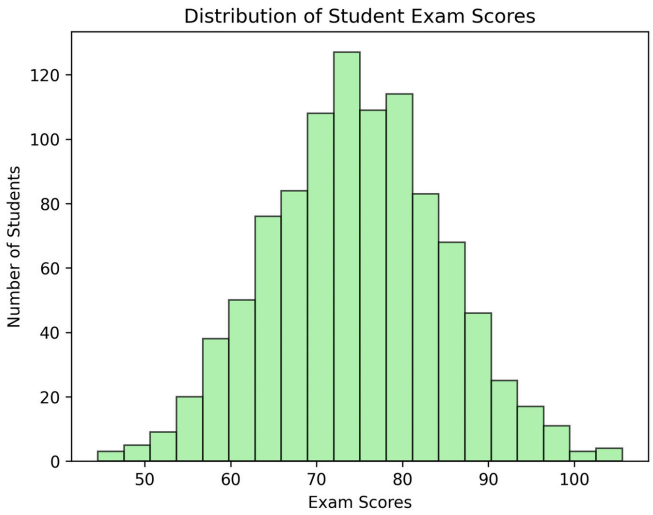
3) Horizontal Bar Chart - Top Programming Languages

```
languages = ['Python', 'JavaScript', 'Java', 'C++', 'Go']
popularity = [85, 78, 72, 65, 58]

plt.barh(languages, popularity, color='skyblue')
plt.xlabel("Popularity Score")
plt.ylabel("Programming Languages")
plt.title("Most Popular Programming Languages 2024")
plt.show()
```



4) Histogram - Student Exam Scores Distribution

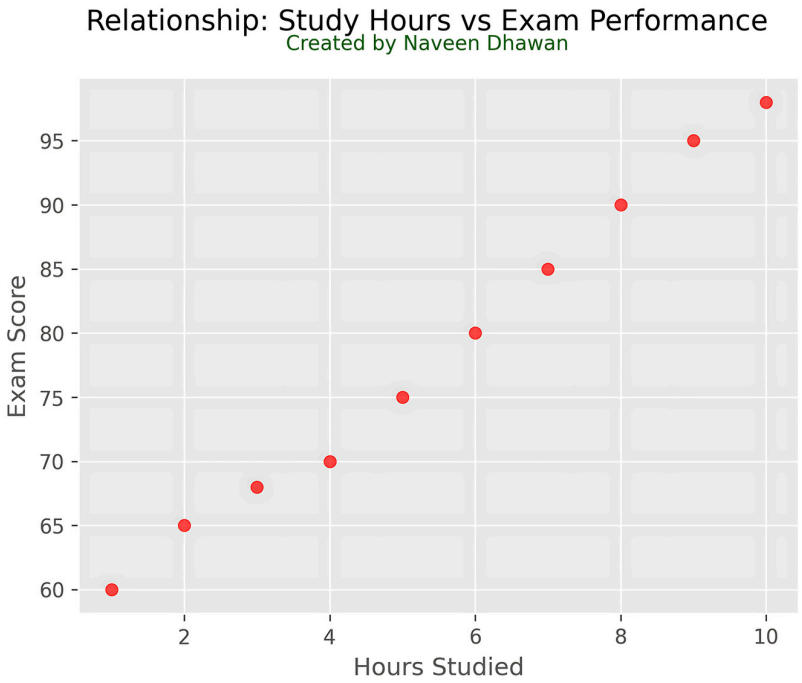
```
import numpy as np

# Exam scores data
scores = np.random.normal(75, 10, 1000) # Mean=75, Std=10

plt.hist(scores, bins=20, color='lightgreen', edgecolor='black', alpha=0.7)
plt.xlabel("Exam Scores")
plt.ylabel("Number of Students")
plt.title("Distribution of Student Exam Scores")
plt.show()
```



5) Scatter Plot - Hours Studied vs Exam Scores

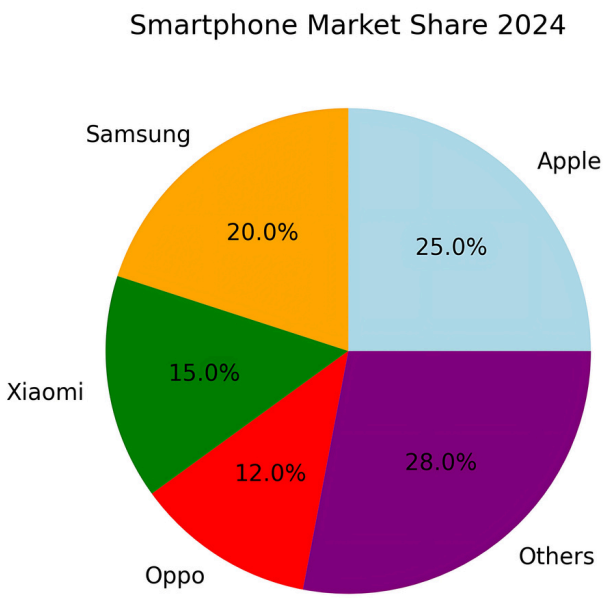
```
import matplotlib.pyplot as plt

hours_studied = [2, 4, 6, 8, 3, 5, 7, 9, 1, 10]
exam_scores = [65, 70, 80, 90, 68, 75, 85, 95, 60, 98]

fig, ax = plt.subplots()
ax.scatter(hours_studied, exam_scores, color='red', alpha=0.7)
ax.set_xlabel("Hours Studied")
ax.set_ylabel("Exam Score")

# Main title
fig.suptitle("Relationship: Study Hours vs Exam Performance", fontsize=14,)
```

```
# Subtitle with your name
fig.text(0.5, 0.92, "Created by Naveen Dhawan",
         ha="center", fontsize=10, color="darkgreen")
plt.show()
```



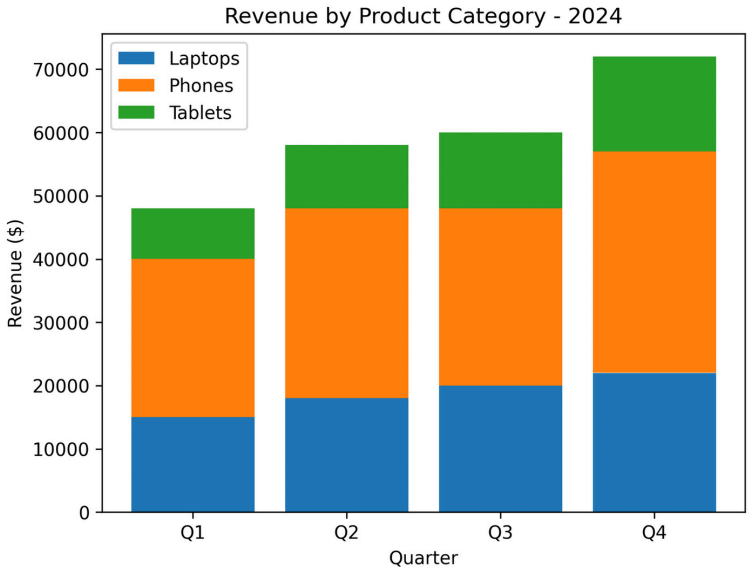
6) Pie Chart - Company Market Share

```
companies = ['Apple', 'Samsung', 'Xiaomi', 'Oppo', 'Others']
market_share = [25, 20, 15, 12, 28]

plt.pie(market_share, labels=companies, autopct='%1.1f%%',
        colors=['lightblue', 'orange', 'green', 'red', 'purple'])
plt.title("Smartphone Market Share 2024")
plt.show()
```



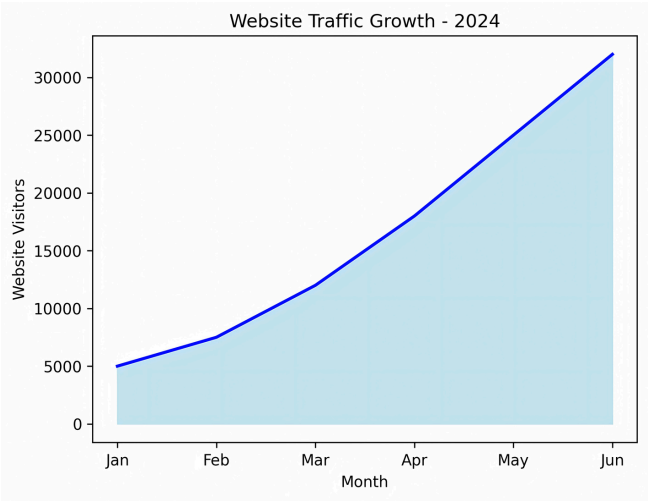
7) Stacked Bar Chart - Revenue by Product Category

```
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
laptops = [15000, 18000, 20000, 22000]
phones = [25000, 30000, 28000, 35000]
tablets = [8000, 10000, 12000, 15000]

plt.bar(quarters, laptops, label='Laptops')
plt.bar(quarters, phones, bottom=laptops, label='Phones')
plt.bar(quarters, tablets, bottom=np.array(laptops)+np.array(phones),
label='Tablets')

plt.xlabel("Quarter")
plt.ylabel("Revenue ($)")
plt.title("Revenue by Product Category - 2024")
plt.legend()
plt.show()
```



8) Area Chart - Website Traffic Growth

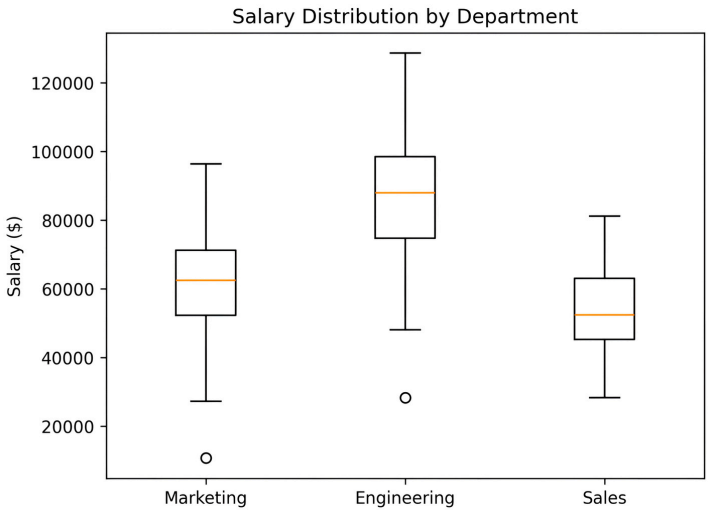
```
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
visitors = [5000, 7500, 12000, 18000, 25000, 32000]

plt.fill_between(months, visitors, color='lightblue', alpha=0.7)
plt.plot(months, visitors, color='blue', linewidth=2)
plt.xlabel("Month")
plt.ylabel("Website Visitors")
plt.title("Website Traffic Growth - 2024")
plt.show()
```



9) Box Plot - Salary Distribution by Department

```
# Sample salary data
marketing = np.random.normal(60000, 15000, 100)
```

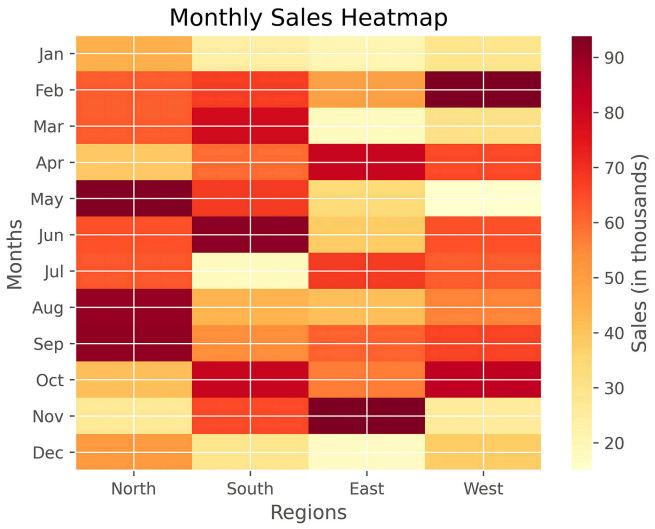
```
engineering = np.random.normal(85000, 20000, 100)
sales = np.random.normal(55000, 12000, 100)

data = [marketing, engineering, sales]
labels = ['Marketing', 'Engineering', 'Sales']

plt.boxplot(data, labels=labels)
plt.ylabel("Salary ($)")
plt.title("Salary Distribution by Department")
plt.show()
```



10) Heatmap - Monthly Sales Performance

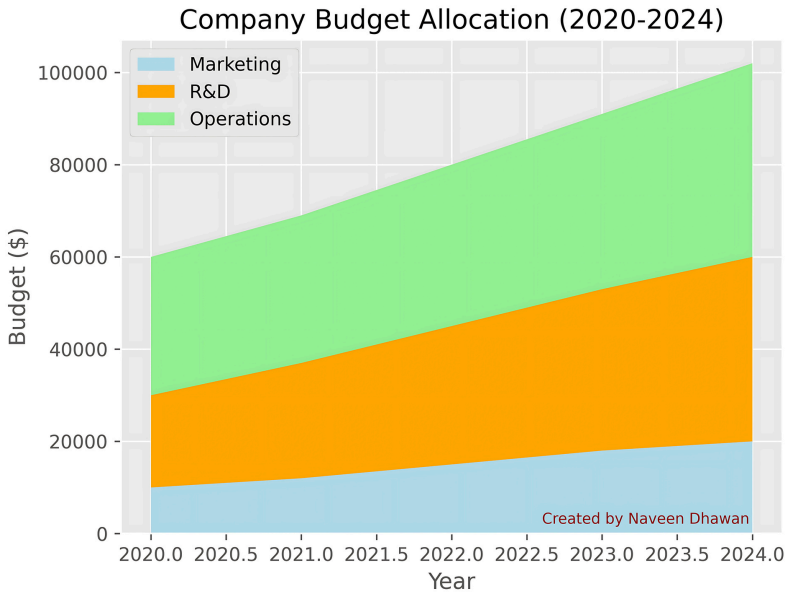
```
import numpy as np

# Monthly sales data (12 months x 4 regions)
sales_data = np.random.randint(10, 100, (12, 4))

plt.imshow(sales_data, cmap='YlOrRd', aspect='auto')
plt.colorbar(label='Sales (in thousands)')
plt.xlabel("Regions")
plt.ylabel("Months")
plt.title("Monthly Sales Heatmap")
plt.xticks(range(4), ['North', 'South', 'East', 'West'])
plt.yticks(range(12), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                       'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
```

```
plt.show()
```



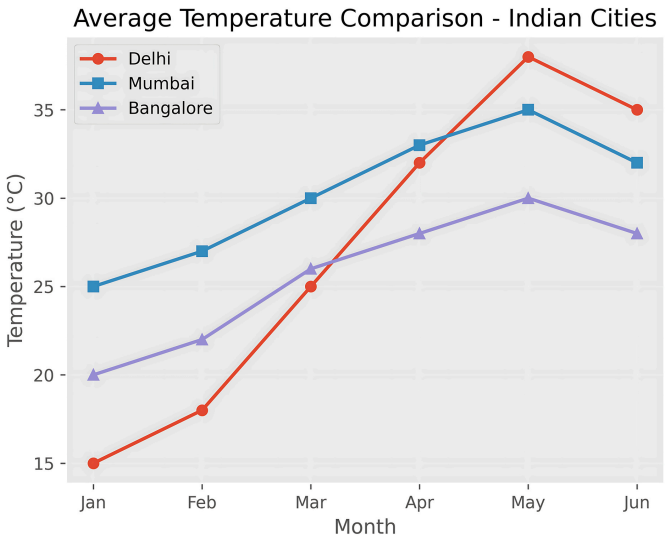
11) Stack Plot - Budget Allocation Over Time

```
import matplotlib.pyplot as plt

years = [2020, 2021, 2022, 2023, 2024]
marketing = [10000, 12000, 15000, 18000, 20000]
rd = [20000, 25000, 30000, 35000, 40000]
operations = [30000, 32000, 35000, 38000, 42000]

fig, ax = plt.subplots()

ax.stackplot(years, marketing, rd, operations,
             labels=['Marketing', 'R&D', 'Operations'],
             colors=['lightblue', 'orange', 'lightgreen'])
ax.set_xlabel("Year")
ax.set_ylabel("Budget ($)")
ax.set_title("Company Budget Allocation (2020-2024)")
ax.legend(loc='upper left')
ax.text(0.95, 0.02, "Created by Naveen Dhawan",
        transform=ax.transAxes, fontsize=8,
        color="darkred", ha="right", alpha=0.9)
plt.show()
```



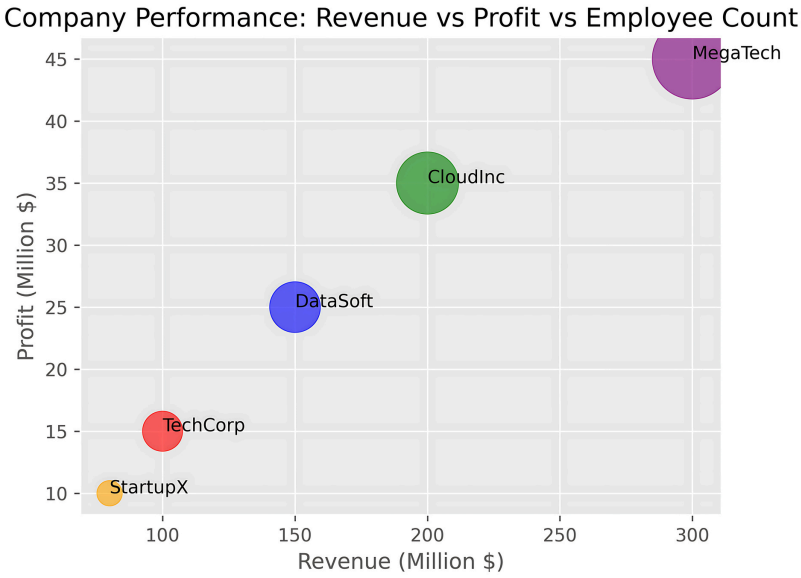
12) Multiple Line Chart - Temperature Comparison

```
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
delhi = [15, 18, 25, 32, 38, 35]
mumbai = [25, 27, 30, 33, 35, 32]
bangalore = [20, 22, 26, 28, 30, 28]

plt.plot(months, delhi, marker='o', label='Delhi', linewidth=2)
plt.plot(months, mumbai, marker='s', label='Mumbai', linewidth=2)
plt.plot(months, bangalore, marker='^', label='Bangalore', linewidth=2)

plt.xlabel("Month")
plt.ylabel("Temperature (°C)")
plt.title("Average Temperature Comparison - Indian Cities")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```



13) Bubble Chart - Company Performance Analysis

```
# Revenue (x), Profit (y), Number of Employees (bubble size)
companies = ['TechCorp', 'DataSoft', 'CloudInc', 'StartupX', 'MegaTech']
revenue = [100, 150, 200, 80, 300]
profit = [15, 25, 35, 10, 45]
employees = [500, 800, 1200, 200, 2000]

plt.scatter(revenue, profit, s=employees, alpha=0.6,
           c=['red', 'blue', 'green', 'orange', 'purple'])

for i, company in enumerate(companies):
    plt.annotate(company, (revenue[i], profit[i]))

plt.xlabel("Revenue (Million $)")
plt.ylabel("Profit (Million $)")
plt.title("Company Performance: Revenue vs Profit vs Employee Count")
plt.show()
```



Difference Between Matplotlib, Seaborn, and Plotly

Feature / Aspect	Matplotlib	Seaborn	Plotly
Type	Low-level plotting library	High-level statistical visualization	Interactive web-based visualization
Ease of Use	Moderate to Difficult (requires more code)	Easy (concise syntax)	Easy to Moderate (intuitive API)
Customization	Highly Customizable (fine-grained control)	Limited Customization (predefined styles)	Highly Customizable (extensive options)
Built-in Themes / Styles	Basic styles (need manual styling)	Beautiful built-in themes (whitegrid, darkgrid, etc.)	Modern templates (plotly, ggplot2, seaborn)
Interactivity	Static plots (no interactivity)	Static plots (no interactivity)	Fully Interactive (zoom, hover, filter)
Performance	Fast (efficient for large data)	Moderate (built on matplotlib)	Slower (web-based rendering)
Learning Curve	Steep (complex syntax)	Gentle (intuitive for beginners)	Moderate (easy to start, complex for advanced)
Best Use Cases	• Publication-quality plots • Scientific research	• Statistical analysis • Data exploration	• Interactive dashboards • Web applications

Installation	pip install matplotlib	pip install seaborn	pip install plotly
Import Statement	import matplotlib.pyplot as plt	import seaborn as sns	import plotly.express as px import plotly.graph_objects as go
Example Plot	plt.plot(x, y) plt.xlabel("X") plt.ylabel("Y") plt.title("Title") plt.show()	sns.lineplot(x=x, y=y) plt.title("Title") plt.show()	fig = px.line(df, x='x', y='y') fig.show()
Output Format	Static images (PNG, PDF, SVG)	Static images (PNG, PDF, SVG)	Interactive HTML (also PNG, PDF)
Data Integration	Works with arrays, lists Pandas integration	Excellent Pandas integration Built for DataFrames	Native DataFrame support JSON, CSV integration
Documentation	Extensive but complex	Clear and beginner-friendly	Excellent with examples
Community & Support	Large, mature community	Growing, active community	Active, modern community
Deployment	Any Python environment	Any Python environment	Web apps, Jupyter, Dash
File Size	Lightweight	Medium (depends on matplotlib)	Larger (includes JS libraries)
Pros	• Maximum control • Widely adopted • Publication ready • Stable and mature	• Beautiful defaults • Statistical functions • Easy syntax • Great for EDA	• Interactive features • Modern appearance • Web deployment • Real-time updates
Cons	• Verbose code • Steep learning curve • No interactivity • Manual styling needed	• Limited customization • Depends on matplotlib • Not interactive • Less control	• Larger file sizes • Performance issues • Complex for simple plots • Web dependency

✨ Summary

- **Matplotlib** → Use when you need **fine-grained control** over every element of a chart and for creating **static, publication-ready reports**. (**maximum control)**

- **Seaborn** → Use when you want **beautiful, statistical plots quickly**, with built-in themes and functions that simplify complex visualisations. (**statistical clarity)**
- **Plotly** → Use when you need **interactive visualisations**, **dashboards**, and **presentations** that allow users to explore the data dynamically. (**interactivity & storytelling)**

📌 Matplotlib Intermediate Notes

Ticks & Labels

- **Definition:** Control the placement and style of ticks & axis labels.
- **Why Used:** To make plots more readable, especially for time series, logarithmic scales, or scientific data.
- **Real-world Example:** Finance dashboards — customising date ticks (e.g., show
- quarters, not every day).

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import MultipleLocator, ScalarFormatter

 x = np.linspace(0, 10, 100)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
ax.xaxis.set_major_locator(MultipleLocator(2))
ax.xaxis.set_minor_locator(MultipleLocator(0.5))
ax.xaxis.set_major_formatter(ScalarFormatter())
plt.show()
```

Figure, Axes & Spines

- **Definition:** A Figure is the entire canvas, Axes are the plotting areas, Spines are the borders.
- **Why Used:** Control layout, remove unnecessary borders, or set background colours for styling.

- **Real-world Example:** Business dashboards often remove extra spines to create a “minimalist” look.

```
fig, ax = plt.subplots()
ax.plot(x, y)
ax.spines['top'].set_color('none') # remove top border
ax.spines['right'].set_color('none')
ax.set_facecolor("#f5f5f5")
plt.show()
```

Lines & Markers

- **Definition:** Lines connect data points; markers highlight individual points.
- **Why Used:** Distinguish multiple datasets or emphasise specific values.
- **Real-world Example:** Tracking daily sales where markers show weekends.

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

X = np.linspace(0, 10*np.pi, 1000)
Y = np.sin(X)
fig, ax = plt.subplots()
ax.plot(X, Y, 'C1o-', markevery=50, mfc='white')

# Main title
fig.suptitle("Sine Wave with Markers", fontsize=14, fontweight="bold")
# Subtitle with your name
fig.text(0.5, 0.92, "Created by Naveen Dhawan", ha="center", fontsize=10,
         color="darkgreen")
plt.show()
```

Scales & Projections

- **Definition:** Transform axes to linear, log, symlog, polar, etc.
- **Why Used:** Log scale reveals multiplicative growth; polar for angular data.
- **Real-world Example:** Epidemiology (COVID spread plotted on log scale).

```
fig, ax = plt.subplots()
ax.set_xscale('log')
ax.plot([1, 10, 100, 1000], [1, 2, 3, 4], marker="o")
plt.show()
```

Text & Ornaments

- **Definition:** Add annotations, filled areas, and labels to enrich plots.
- **Why Used:** To highlight key regions or add context.
- **Real-world Example:** Highlight profit zones vs. loss zones in stock market charts.

```
x = np.linspace(0, 10, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
ax.fill_between(x, 0, y, where=(y>0), alpha=0.3)
ax.text(2, 0.5, "Positive Area", fontsize=12)
plt.show()
```

Legend

- **Definition:** Explains what each colour/line/marker represents.
- **Why Used:** Essential for multi-series plots.
- **Real-world Example:** Energy consumption dashboard showing solar vs wind vs grid.

```
fig, ax = plt.subplots()
ax.plot(x, np.sin(x), "C0", label="Sine")
ax.plot(x, np.cos(x), "C1", label="Cosine")
ax.legend(loc="upper right")
plt.show()
```

Annotation

- **Definition:** Point out specific data points with arrows or labels.
- **Why Used:** To highlight anomalies, peaks, or thresholds.
- **Real-world Example:** Marking stock price peaks on a chart.

```
fig, ax = plt.subplots()
ax.plot(x, y)
ax.annotate("Peak", xy=(np.pi/2, 1), xytext=(4, 1.2),
            arrowprops=dict(arrowstyle="→", color="red"))
plt.show()
```

Colors

- **Definition:** Matplotlib supports categorical (`C0, C1`) and colourmap-based colours.
- **Why Used:** Colours encode categories and intensities.
- **Real-world Example:** Comparing multiple ad campaigns (each colour = a campaign).

```
colors = ["C0","C1","C2","C3"]
for i, c in enumerate(colors):
    plt.plot(x, np.sin(x+i), c=c, label=f"shift {i}")
plt.legend()
plt.show()
```

Size & DPI

- **Definition:** Control figure size and resolution for print/web.
- **Why Used:** Publications need high DPI (≥300), while dashboards need an optimised size.
- **Real-world Example:** Submitting research charts to academic journals.

```
fig = plt.figure(figsize=(3.15, 3.15), dpi=300)
plt.plot(x, y)
plt.savefig("figure_highres.png", dpi=300, bbox_inches="tight")
```

🎭 Advanced Matplotlib — 10 Core Topics

1) Object-Oriented (OO) API vs pyplot

- **Definition:** OO API:explicitFigureandAxesobjects(fig, ax = plt.subplots()), not the stateful plt interface.
- **Why:** Safer & composable for production, multi-axis figures, and functions that return/accept axes.
- **How it works:** Create a Figure and one or more Axes. Call ax.plot, ax.set_title, etc. Pass ax into helper functions.

```
# oo_api_example.py
import numpy as np, matplotlib.pyplot as plt
def plot_series(ax, x, y, label=None):
    ax.plot(x, y, linewidth=1.5, label=label)
    ax.set_xlabel('x'); ax.set_ylabel('y')
    if label: ax.legend()

x = np.linspace(0, 10, 200); y = np.sin(x)
fig, ax = plt.subplots(figsize=(6,3))
plot_series(ax, x, y, label='sin(x)')
fig.savefig('oo_api_example.png', dpi=150, bbox_inche
s='tight')
plt.show()
```

Tip: Always return fig, ax from plot builder functions for reusability.

2) Figures, Axes & Artists — anatomy & lifecycle

Definition:

- **Figure:** canvas.
- **Axes:** plotting area (x/y).
- **Artist:** any drawable element (Line2D, Text, Patch, Image, Collection).
- **Why:** Inspecting/updating artists is essential for performance (animations, streaming) and debugging.

```
# artist_lifecycle.py
import numpy as np, matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
```

```
x = np.linspace(0, 2*np.pi, 200)
fig, ax = plt.subplots()
 line, = ax.plot(x, np.sin(x), label='line')
# Line2D artist
txt = ax.text(0.5, 0.9, 'phase=0', transform=ax.transAxes)
rect = Rectangle((1.0, -1.0), 0.5, 1.8, alpha=0.15) ax.add_patch(rect)

# Update existing artists (efficient)
line.set_ydata(np.sin(x + 0.7))
txt.set_text('phase=0.7')
rect.remove()
fig.canvas.draw()
plt.show()
```

Tip: Use `ax.get_children()` to inspect artists; use `set_*` methods instead of re-plotting.

3) Layouts & Subplots (subplots, GridSpec, subplot_mosaic, inset axes)

Definition:

Tools to arrange multiple axes precisely.

Why:

Publication/dashboards require aligned panels, shared axes and unified colorbars.

How: Use plt.subplots for simple grids, GridSpec/subplot_mosaic for complex layouts, and inset_axes for zoom boxes.

```
# layouts_demo.py
import numpy as np, matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
from mpl_toolkits.axes_grid1.inset_locator import inset_axes

fig = plt.figure(constrained_layout=True, figsize=(9,4)) gs =
GridSpec(2, 3, figure=fig)
ax_main = fig.add_subplot(gs[:, :2])
ax_right = fig.add_subplot(gs[0, 2])
ax_bottom = fig.add_subplot(gs[1, 2])
```

```
x = np.linspace(0, 10, 200)
ax_main.plot(x, np.sin(x)); ax_main.set_title('Main')
ax_right.plot(x, np.cos(x)); ax_right.set_title('Right')
ax_bottom.plot(x, np.sin(x)*0.2); ax_bottom.set_title('Bottom')

# inset
axins = inset_axes(ax_main, width="30%", height="30%", loc='upper right')
axins.plot(x[:40], np.sin(x[:40])); axins.set_xticks([]); axins.set_yticks([])

 plt.show()
```

Tip: Prefer constrained_layout=True for automatic spacing, but test complex arrangements manually.

4) Lines, Markers, and Styles

Definition: Styling of line plots: colour, linewidth, linestyle, marker, zorder, alpha.

Why: Visual clarity, emphasis, and publication-style aesthetics.

How: Use kwargs in plot() and Line2D properties; plt.style.use() for theme.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 100)
# plt.style.use('seaborn-darkgrid') # Commented out the unavailable style
 plt.style.use('ggplot') # Using a default style available in Matplotlib
plt.plot(x, np.sin(x), label='sin', lw=2, marker='o', markevery=10)
plt.plot(x, np.cos(x), label='cos', lw=1.2, linestyle='--', alpha=0.9)
plt.legend(); plt.title('Lines & Markers'); plt.show()
```

Tip: Use `markevery` to avoid clutter when markers overplot dense lines.

5) Colours & Colourmaps

Definition:

Colour specifications (named colours, hex) and colourmaps for scalar-to-color mapping.

Why: Perceptually-correct colourmaps (e.g.,viridis) avoid misleading interpretation and help colorblind readers.

How: Use cmap and normalisation (Normalise, LogNorm, BoundaryNorm).

```
# colormaps_demo.py
import numpy as np, matplotlib.pyplot as plt
from matplotlib.colors import Normalize
data = np.random.randn(100,100)
plt.imshow(data, cmap='viridis', norm=Normalize(vmin=-2, vmax=2), origin
='lower')
plt.colorbar(label='value')
plt.title('Viridis with Normalize'); plt.show()
```

Tip: Avoid jet. Use cividis/viridis/plasma for accessibility.

6) Ticks, Tick Locators & Formatters

Definition: Control placement and formatting of major/minor ticks (including dates).

Why: Clean axis ticks improve readability and correctly convey temporal scales.

How: Use MaxNLocator, MultipleLocator, FuncFormatter, and matplotlib.dates locators/formatters.

```
# ticks_dates.py
import matplotlib.pyplot as plt, matplotlib.dates as mdates
import pandas as pd, numpy as np
dates = pd.date_range('2024-01-01', periods=12, freq= 'M')
vals = np.random.randn(12).cumsum()
fig, ax = plt.subplots()
ax.plot(dates, vals, marker='o')
ax.xaxis.set_major_locator(mdates.MonthLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
plt.xticks(rotation=30);
plt.show()
```

Tip: Use minor ticks (ax.xaxis.set_minor_locator) for gridlines and finer guides.

7) Axis Scales & Transformations

Definition: Log, symlog, probability scales; transforms between coordinate systems.

Why: Proper scaling reveals multiplicative relationships and wide-range data.

How: ax.set_yscale('log'), use ax.transData/ax.transAxes for annotation coordinates, Affine2D for custom transforms.

```
# axis_scales.py
import numpy as np, matplotlib.pyplot as plt
x = np.logspace(0, 3, 100)
y = x**2
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_xscale('log'); ax.set_yscale('log')
ax.set_title('Log-Log scale'); plt.show()
```

Tip: Use symlog for data with negative and positive values around zero.

8) Images, imshow, and colourmap tuning

Definition: Displaying 2D arrays (images, rasters) with control over interpolation, extent, origin, aspect ratio and normalisation.

Why: Analysts show heatmaps, correlation matrices, and spatial rasters.

How: ax.imshow(arr, cmap=..., origin='lower', extent=[..], interpolation='nearest').

```
# imshow_demo.py
import numpy as np, matplotlib.pyplot as plt
arr = np.outer(np.sin(np.linspace(0,3*np.pi,200)),
np.cos(np.linspace(0,2*np.pi,300)))
plt.imshow(arr, aspect='auto', cmap='cividis', origin='lower', interpolation
='bilinear')
plt.colorbar(label='value'); plt.title('imshow example'); plt.show()
```

Tip: Use extent to map pixel indices to real-world coordinates.

9) 2D Vector Fields (quiver, streamplot, barbs)

Definition: Visualizing vector fields with arrows (quiver), streamlines (streamplot) or meteorological barbs.
Why: Common for flow fields in engineering, weather, and physics analyses.
How: Provide U, V vector components on a grid and call ax.quiver or ax.streamplot.

```
# vector_fields.py
import numpy as np, matplotlib.pyplot as plt
Y, X = np.mgrid[-3:3:40j, -3:3:40j]
```

```
U = -Y; V = X # rotational field
fig, ax = plt.subplots()
ax.streamplot(X, Y, U, V, density=1, color=np.hypot(U, V), cmap='viridis')
ax.set_title('Streamplot (rotational field)'); plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
Y, X = np.mgrid[-3:3:40j, -3:3:40j]
U = -Y
V = X

fig, ax = plt.subplots()
ax.streamplot(X, Y, U, V, density=1)

fig.text(0.5, 0.5, "Naveen Dhawan", fontsize=40, color="gray",
ha="center", va="center", alpha=0.15, rotation=30)

plt.show()
```

Tip: Color by speed np.hypot(U,V) to encode magnitude.

10) 3D plotting (mplot3d)

Definition: 3D line, scatter, surface plots via mpl_toolkits.mplot3d.
Why: Useful for exploratory 3D views; for publication-grade 3D consider specialised libs (Mayavi, PyVista, Plotly).
How: Create Axes3D with projection='3d' and call plot_surface/scatter.

```
# mplot3d_demo.py
import numpy as np, matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
x = np.linspace(-3,3,80); y = x.copy()
X, Y = np.meshgrid(x,y)
Z = np.sin(np.sqrt(X**2+Y**2))
fig = plt.figure(figsize=(6,4))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='coolwarm', rstride=1, cstride=1, linewidth=0,
```

```
antialiased=True)
ax.set_title('3D surface'); plt.show()
```

Tip: Avoid complex 3D for publication; provide multiple 2D projections instead if clarity matters.

📌 Summary – Complete Matplotlib Guide

- **Matplotlib Fundamentals** → Learn to create essential charts such as line, bar, scatter, pie, and histograms.
- **Customisation** → Control every element of a chart: colours, markers, labels, ticks, scales, and annotations.
- **Practical Examples** → Apply visualisation techniques to real-world data like stock prices, sales, salaries, and heatmaps.
- **Intermediate & Advanced Tools** → Work with subplots, layouts, colour maps, vector fields, and 3D plots.
- **Library Comparison** → Understand when to use Matplotlib, Seaborn, or Plotly based on project needs.
- **Best Practices** → Focus on clarity, readability, accessibility, and effective storytelling with visuals.

📘 Learning Files

💻 Jupyter Notebook (.ipynb)

Matplotlib - Google Drive

https://drive.google.com/drive/folders/1fjrB15uU8yA6udunJ9-lE_6D5vInTQXD?usp=drive_link

Note: These files include all examples, plots, and explanations (with practical examples).