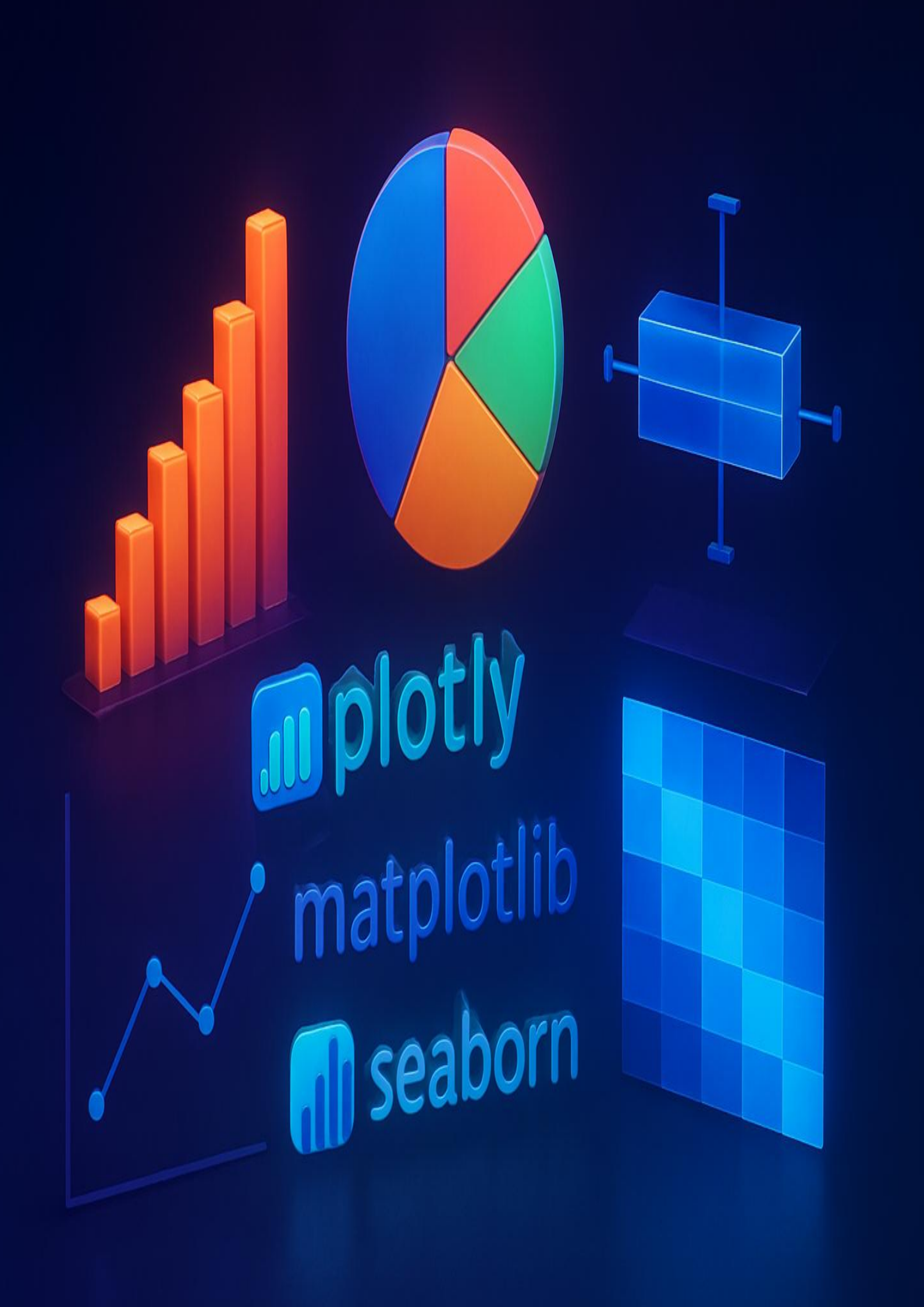

plotly
matplotlib
seaborn

Table of Contents

Introduction

Purpose

The purpose of this document is to provide a complete exploration of data visualization in Python using three of the most widely adopted libraries: **Matplotlib**, **Seaborn**, and **Plotly**. These tools enable developers, data scientists, and analysts to effectively communicate insights through a wide range of static, interactive, and statistical plots.

Scope

The document covers **all major types of visualizations,** including line plots, histograms, bar charts, scatter plots, pie charts, boxplots, violin plots, heatmaps, and 3D plots. For each visualization type, we provide:

- A clear **description** of its purpose.
- The **parameters** available and how to use them effectively.
- Practical **examples** with annotated code.
- Guidance on **when to use** each visualization for maximum clarity.

Procedure

The document walks through visualizations incrementally, starting from basic plots in **Matplotlib**, moving into statistical visualization with **Seaborn**, and ending with interactive and advanced visualizations using **Plotly**. Each section is accompanied by **sample outputs**, allowing learners to directly compare different libraries for the same visualization type.

Problem Brief

Data without visualization can be overwhelming and difficult to interpret. Choosing the right visualization is crucial for accurately conveying patterns, distributions, relationships, and insights. This document addresses the problem by not only demonstrating **how** to create visualizations but also explaining **when** and **why** to use specific diagrams.

Functions

Matplotlib

- **Matplotlib is the foundation library for visualization in Python. It provides low-level control and flexibility.**
- **Import matplotlib.pyplot as plt**

Function	Description	Example
plt.plot()	Line plots	plt.plot(x, y, color="red", linestyle="--", marker="o")
plt.scatter()	Scatter plots	plt.scatter(x, y, marker="x", s=100, c="blue")
plt.bar()	Bar charts	plt.bar(categories, values, color="green")
plt.hist()	Histograms	plt.hist(data, bins=10, density=True, alpha=0.7)
plt.pie()	Pie charts	plt.pie(sizes, labels=labels, autopct="%1.1f%%")
plt.boxplot()	Box plots	plt.boxplot(data, vert=False, patch_artist=True)
plt.subplot()	Create a single subplot (m×n grid)	plt.subplot(2, 2, 1)
plt.subplots()	Create a grid of subplots (returns figure + axes)	fig, ax = plt.subplots(2, 2, figsize=(10,6))
plt.title(), plt.xlabel(), plt.ylabel()	Add titles and axis labels	plt.title("My Plot", fontsize=14, color="blue")
plt.legend()	Adds a legend	plt.legend(loc="upper right")
plt.savefig()	Save figure	plt.savefig("plot.png", dpi=300, bbox_inches="tight")
plt.show()	Displays figure	plt.show()
plt.xticks()	Set x-axis ticks/labels	plt.xticks(rotation=45, fontsize=10)
plt.tight_layout()	Adjusts subplot spacing automatically	plt.tight_layout(pad=2.0)

Function	Description	Example
plt.figure()	Creates a new figure	plt.figure(figsize=(8,6), dpi=100, facecolor="white")
plt.style.use()	sets the plotting style for all following plots. (Colors, backgrounds, line width and markers, font, grid lines)	Plt.style.use(“ggplot”)
plt.style.available	Shows all available styles for plt.style.use()	print(plt.style.available)
plt.annotate()	Used to **add text annotations with optional arrows** to highlight specific points in your Matplotlib plot	plt.annotate(text, xy, xytext=None, xycoords='data', textcoords=None, arrowprops=None, fontsize=None, color=None)

Detailed Subplot Use

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

x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

plt.figure(figsize=(8,6))

# First subplot (2 rows, 1 column, 1st plot)
plt.subplot(2, 1, 1)
plt.plot(x, y1, 'r-', label="sin(x)")
plt.title("Sine Function")
plt.legend()

# Second subplot (2 rows, 1 column, 2nd plot)
plt.subplot(2, 1, 2)
plt.plot(x, y2, 'b--', label="cos(x)")
plt.title("Cosine Function")
plt.legend()

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

Detailed Subplots Use

```
fig, ax = plt.subplots(2, 2, figsize=(10, 6))

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

ax[0,0].plot(x, np.sin(x), color="red")
ax[0,0].set_title("sin(x)")

ax[0,1].plot(x, np.cos(x), color="blue")
ax[0,1].set_title("cos(x)")

ax[1,0].scatter(x, np.sin(x), color="green", s=10)
ax[1,0].set_title("Scatter sin(x)")

ax[1,1].bar([1,2,3], [3,5,7], color="purple")
ax[1,1].set_title("Bar Chart")

fig.suptitle("Subplots Example", fontsize=16, color="darkblue")
plt.tight_layout(rect=[0,0,1,0.95])   # prevent overlap with suptitle
plt.show()
```

Make_Subplots Use

```
fig = make_subplots(
   rows=2, cols=2,
   subplot_titles=("Line", "Scatter", "Bar", "Histogram")
)

# Line
fig.add_trace(go.Scatter(x=[1,2,3], y=[3,1,6], mode="lines"), row=1, col=1)

# Scatter
fig.add_trace(go.Scatter(x=[1,2,3], y=[2,5,3], mode="markers"), row=1, col=2)

# Bar
fig.add_trace(go.Bar(x=["A","B","C"], y=[4,7,2]), row=2, col=1)

# Histogram
fig.add_trace(go.Histogram(x=[1,2,2,3,3,3,4,4,4,4]), row=2, col=2)

fig.update_layout(height=600, width=800, title="2x2 Subplots")
fig.show()

fig = make_subplots(rows=2, cols=1, shared_xaxes=True)

fig.add_trace(go.Scatter(x=[1,2,3], y=[3,1,6], mode="lines", name="Line"), row=1, col=1)
fig.add_trace(go.Bar(x=[1,2,3], y=[2,5,3], name="Bar"), row=2, col=1)

fig.update_layout(title="Shared X-axis Subplots")
fig.show()
```

Seaborn

- **Seaborn is built on top of Matplotlib — it makes statistical visualizations easier with better defaults.**
- **Import seaborn as sns**

Function	Description	Example
sns.histplot()	Histogram with options for KDE	sns.histplot(data, bins=20, kde=True)
sns.barplot()	Bar plot with statistical aggregation	sns.barplot(x="cat", y="val", data=df)
sns.countplot()	Counts unique categories	sns.countplot(x="cat", data=df)
sns.boxplot()	Box plot	sns.boxplot(x="cat", y="val", data=df)
sns.violinplot()	Combines boxplot + KDE	sns.violinplot(x="cat", y="val", data=df)
sns.scatterplot()	Scatter plot	sns.scatterplot(x="x", y="y", data=df)
sns.lineplot()	Line plot	sns.lineplot(x="time", y="val", data=df)
sns.heatmap()	Heatmap	sns.heatmap(matrix, annot=True, cmap="coolwarm")
sns.pairplot()	Pairwise scatter plots for all features	sns.pairplot(df)
sns.jointplot()	Scatter + histogram/marginal	sns.jointplot(x="x", y="y", data=df, kind="hex")
sns.lmplot()	Scatter + regression line	sns.lmplot(x="x", y="y", data=df, hue="cat")
sns.kdeplot()	Kernel Density Estimate (smooth distribution curve)	sns.kdeplot(data, shade=True, bw_adjust=0.5)
sns.stripplot()	Plots individual data points (with jitter option)	sns.stripplot(x="cat", y="val", data=df, jitter=True)
sns.pointplot()	Shows mean + confidence interval as points	sns.pointplot(x="cat", y="val", data=df, capsize=0.1)

Plotly Express

- **Plotly Express makes interactive charts with concise syntax.**
- **Import plotly.express as px**
- **Fig = px. …()**

Function	Description	Example
px.line()	Line plot	px.line(df, x="time", y="val")
px.scatter()	Scatter plot	px.scatter(df, x="x", y="y", color="cat")
px.bar()	Bar plot	px.bar(df, x="cat", y="val")
px.histogram()	Histogram	px.histogram(df, x="col", nbins=20)
px.box()	Box plot	px.box(df, x="cat", y="val")
px.violin()	Violin plot	px.violin(df, x="cat", y="val")
px.pie()	Pie chart	px.pie(df, values="val", names="cat")
px.area()	Area chart	px.area(df, x="time", y="val")
px.density_heatmap()	2D heatmap	px.density_heatmap(df, x="x", y="y")
px.treemap()	Treemap	px.treemap(df, path=["region", "country"], values="pop")
px.sunburst()	Sunburst chart	px.sunburst(df, path=["region", "country"], values="pop")
fig.update_layout()	Updates layout of diagram	fig.update_layout(title="Plotly Scatter Plot")
fig.write_image()	Saves image	fig.write_image("histogram_plotly.png")
fig.write_html()	Saves image as html to be dynamic	fig.write_html(“fig3.html”)
fig.update_xaxes (rangeslider_visible=True)	Adds a **slider below the x-axis** so you can zoom in/out across a date range. (Used in Time Series plots)	fig.update_xaxes(rangeslider_visible=True)

Histogram

What is a Histogram?

A **histogram** is a graphical representation of the **distribution of a dataset**. It groups continuous or discrete data into **bins** (intervals) and counts how many values fall into each bin. The result is a series of bars, where:

- The **x-axis** = data intervals (bins).
- The **y-axis** = frequency (count of data points in each bin).

It's one of the most fundamental tools for **exploratory data analysis (EDA)** because it shows the **shape of the distribution** (normal, skewed, uniform, etc.).

When to Use a Histogram

✓ When you want to **understand the distribution** of a dataset.
✓ When you want to **detect outliers, skewness, or modality** (uni-modal, bi-modal).
✓ When comparing distributions across multiple groups (e.g., test scores of two classes).
✗ Not ideal for categorical data — use **bar charts** instead.

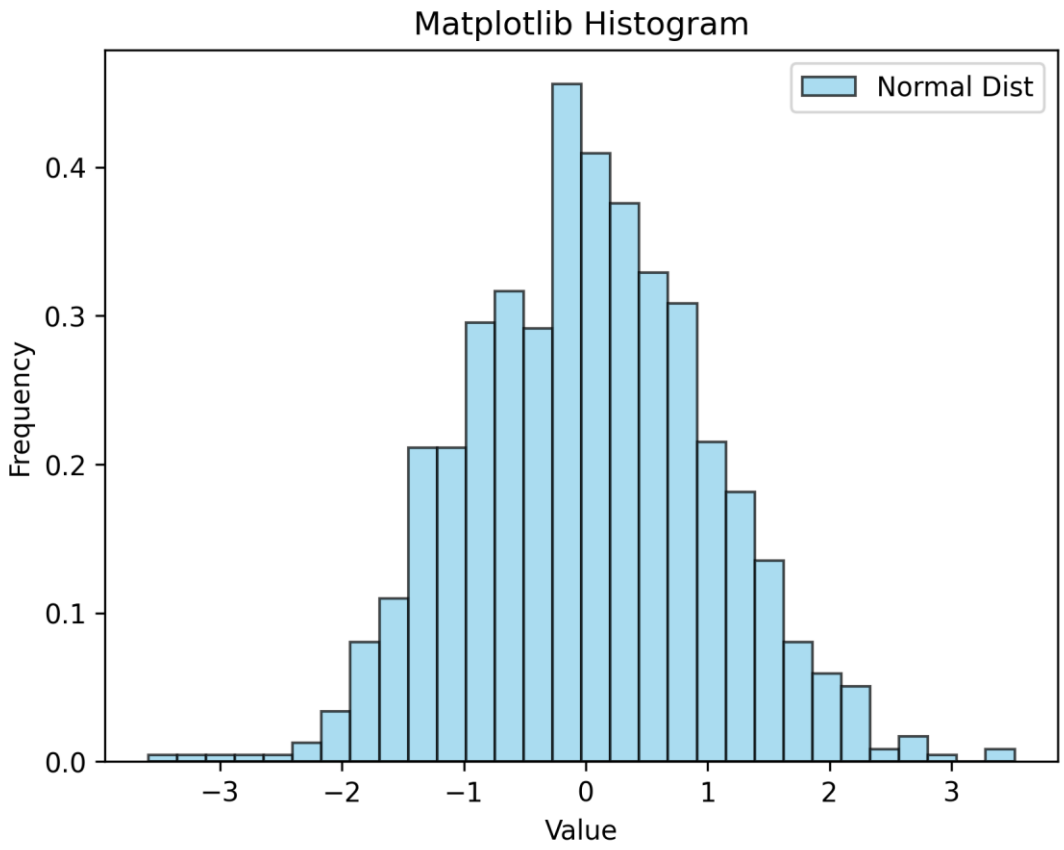
Examples across Libraries

Matplotlib plt.hist() Parameters

Parameter	Description	Example
x	Data to plot (array/list)	plt.hist(data)
bins	Number of bins or bin edges	plt.hist(data, bins=20)
range	Tuple (min, max) → limit values considered	plt.hist(data, bins=20, range=(0,100))
density	If True, normalizes to probability density	plt.hist(data, density=True)
weights	Apply weights to data points	plt.hist(data, weights=w)
cumulative	If True, shows cumulative histogram	plt.hist(data, cumulative=True)
bottom	Shifts bars vertically	plt.hist(data, bottom=2)
histtype	Type: 'bar', 'barstacked', 'step', 'stepfilled'	plt.hist(data, histtype="step")
align	'left', 'mid', 'right'	plt.hist(data, align="mid")
orientation	'vertical' or 'horizontal'	plt.hist(data, orientation="horizontal")
rwidth	Relative bar width (0–1)	plt.hist(data, rwidth=0.8)
color	Bar color	plt.hist(data, color="skyblue")
label	Label for legend	plt.hist(data, label="Sample")
stacked	If True, stacks multiple datasets	plt.hist([data1, data2], stacked=True)
alpha	Transparency (0–1)	plt.hist(data, alpha=0.5)

Code Example

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

data = np.random.randn(1000)

plt.hist(data, bins=30, color="skyblue", edgecolor="black",
        alpha=0.7, density=True, histtype="stepfilled",
        orientation="vertical", label="Normal Dist")
plt.title("Matplotlib Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

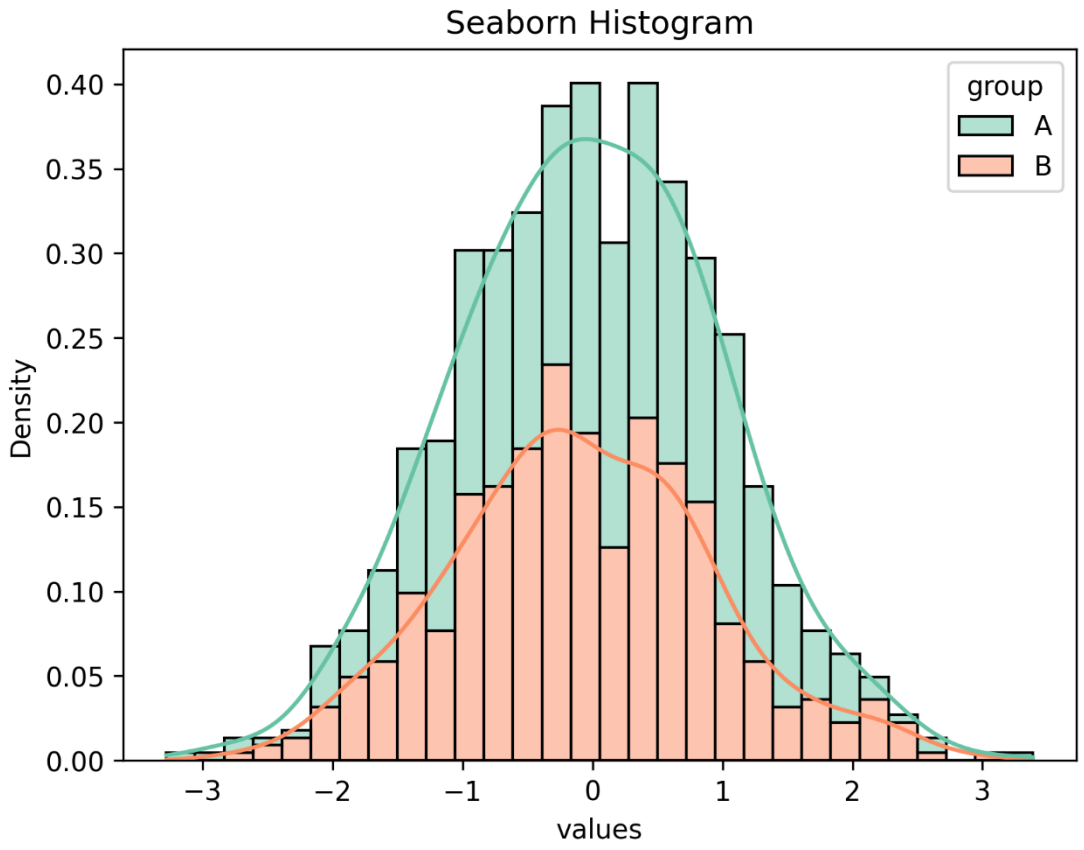


Seaborn sns.histplot() Parameters

Parameter	Description	Example
data	Dataset	sns.histplot(data, bins=30)
x / y	Variable(s) to plot	sns.histplot(data=df, x="col")
bins	Number of bins	sns.histplot(data, bins=40)
binwidth	Width of each bin	sns.histplot(data, binwidth=5)
binrange	Range of bins (min, max)	sns.histplot(data, binrange=(0,100))
stat	'count', 'frequency', 'density', 'probability'	sns.histplot(data, stat="density")
cumulative	Cumulative histogram	sns.histplot(data, cumulative=True)
element	'bars', 'step', 'poly'	sns.histplot(data, element="step")
fill	Fill bars (True/False)	sns.histplot(data, fill=False)
multiple	'layer', 'stack', 'dodge'	sns.histplot(data=df, x="col", hue="cat", multiple="stack")
hue	Color by category	sns.histplot(data=df, x="val", hue="cat")
palette	Color palette	sns.histplot(data, palette="Set2")
kde	Add kernel density line	sns.histplot(data, kde=True)
common_norm	Normalize across groups (default True)	sns.histplot(data=df, x="val", hue="cat", common_norm=False)

Code Example

```
import seaborn as sns
import numpy as np
import pandas as pd

df = pd.DataFrame({"values": np.random.randn(1000),
                   "group": np.random.choice(["A", "B"], 1000)})

sns.histplot(data=df, x="values", hue="group", bins=30,
             multiple="stack", kde=True, stat="density",
             element="bars", palette="Set2")
plt.title("Seaborn Histogram")
plt.show()
```

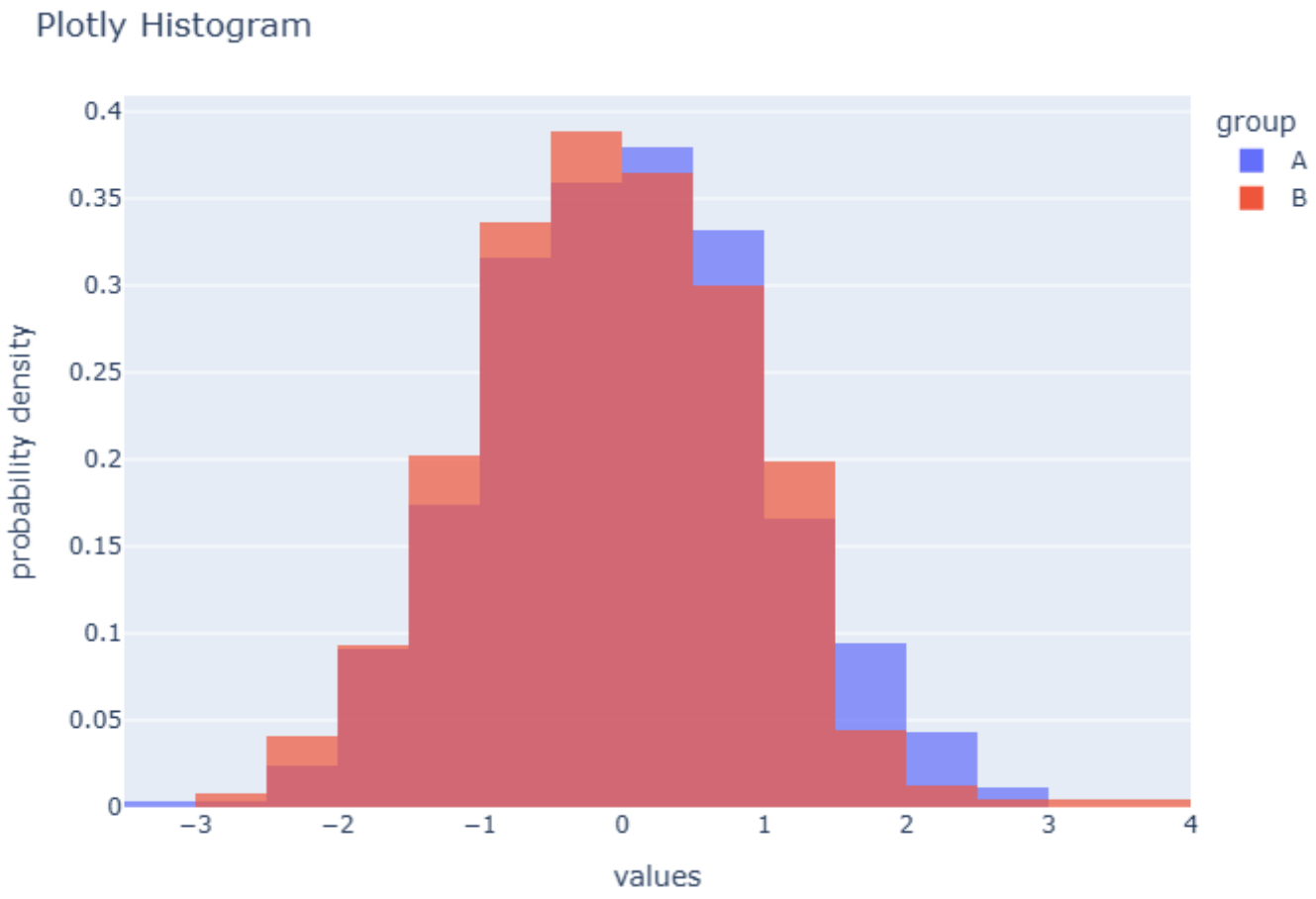


Plotly px.histogram() Parameters

Parameter	Description	Example
data_frame	Data source (DataFrame/dict/array)	px.histogram(df, x="col")
x / y	Variable(s) to plot	px.histogram(df, x="values", color="group")
color	Group by category	px.histogram(df, x="values", color="group")
nbins	Number of bins	px.histogram(df, x="values", nbins=30)
histnorm	'percent', 'probability', 'density', or '' (count)	px.histogram(df, x="values", histnorm="percent")
barmode	'overlay', 'stack', 'relative', 'group'	px.histogram(df, x="values", color="group", barmode="stack")
opacity	Transparency (0–1)	px.histogram(df, x="values", opacity=0.6)
facet_row / facet_col	Small multiples (facets)	px.histogram(df, x="values", facet_col="group")
category_orders	Order of categorical variables	px.histogram(df, x="values", color="group", category_orders={"group":["B","A"]})

Code Example

```
import plotly.express as px
import pandas as pd
import numpy as np

df = pd.DataFrame({"values": np.random.randn(1000),
                   "group": np.random.choice(["A", "B"], 1000)})

fig = px.histogram(df, x="values", color="group", nbins=30,
                   barmode="overlay", histnorm="probability density",
                   opacity=0.7)
fig.update_layout(title="Plotly Histogram")
fig.show()
```



Bar Chart

What is a bar chart?

A **bar chart** (or bar plot) visualizes **categorical** data by using rectangular bars whose lengths (or heights) represent the value (count, mean, sum, etc.) associated with each category. Bars can be vertical or horizontal. You can show single-category values, compare groups (grouped bars), stack values (stacked bars), or display normalized proportions (percent / relative bars).

Key variations

- **Single (simple) bar chart** — one bar per category (e.g., sales per product).
- **Grouped (clustered) bar chart** — side-by-side bars per category to compare subgroups (e.g., sales by product and region).
- **Stacked bar chart** — bars are stacked to show parts of a whole (e.g., contribution of subcategories to total).
- **Horizontal bar chart** (barh) — useful when category labels are long or many.
- **Normalized/relative bars** — show percentages instead of raw counts/values.

When to use a bar chart

Use a bar chart when you want to:

- Compare **values across discrete categories** (e.g., months, product types, countries).
- Visualize **aggregated statistics** (counts, sums, means) per category.
- Compare **subgroups** within categories (grouped bar charts).
- Show **composition** of categories (stacked bars) when the focus is on part-to-whole relationships.
- Use a **horizontal** layout when category names are long or there are many categories.

Do **not** use bar charts when you want to:

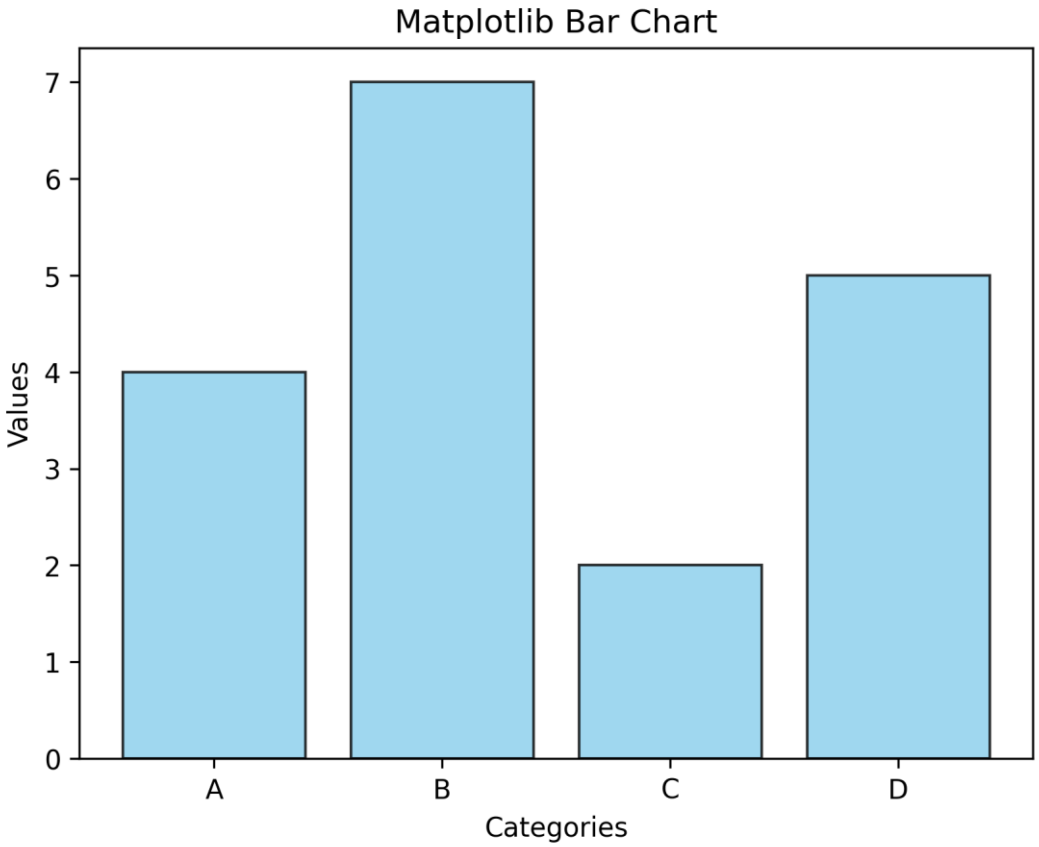
- Show distribution of continuous data (use **histogram**, **kde**, etc.).
- Show relationships between two continuous variables (use **scatter** / **line**).

Matplotlib plt.bar()

Parameter	Description	Example
x	Positions of bars (list/array)	plt.bar([1,2,3], [4,5,6])
height	Heights of bars (values)	plt.bar(x, height)
width	Width of bars (default=0.8)	plt.bar(x, height, width=0.5)
bottom	Y-position of bars (useful for stacking)	plt.bar(x, height, bottom=2)
align	'center' or 'edge'	plt.bar(x, height, align="edge")
color	Bar colors	plt.bar(x, height, color="skyblue")
edgecolor	Border color of bars	plt.bar(x, height, edgecolor="black")
linewidth	Width of bar edges	plt.bar(x, height, linewidth=2)
tick_label	Custom labels for bars	plt.bar(x, height, tick_label=["A","B","C"])
label	Label for legend	plt.bar(x, height, label="Category A")
alpha	Transparency (0–1)	plt.bar(x, height, alpha=0.7)
orientation	Not directly supported (for horizontal use plt.barh())	plt.barh(x, height)

Code Example

```
import matplotlib.pyplot as plt

categories = ["A", "B", "C", "D"]
values = [4, 7, 2, 5]

plt.bar(categories, values, color="skyblue", edgecolor="black", alpha=0.8)
plt.title("Matplotlib Bar Chart")
plt.xlabel("Categories")
plt.ylabel("Values")
plt.show()
```

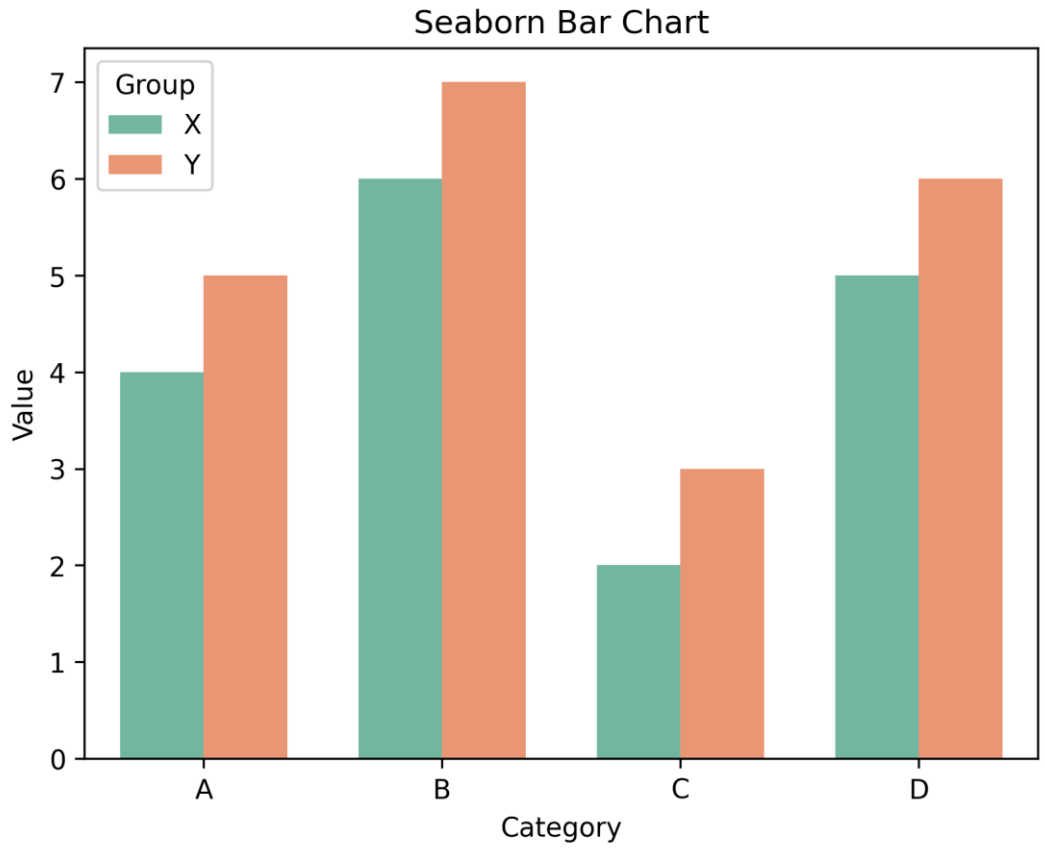


Seaborn sns.barplot()

Parameter	Description	Example
data	DataFrame input	sns.barplot(data=df, x="cat", y="val")
x / y	Variables for categories and values	sns.barplot(x="cat", y="val", data=df)
hue	Group by category	sns.barplot(x="cat", y="val", hue="group", data=df)
order	Custom category order	sns.barplot(x="cat", y="val", order=["C","A","B"], data=df)
estimator	Function to aggregate values (default = mean)	sns.barplot(x="cat", y="val", estimator=sum, data=df)
ci	Confidence interval size (95, 68, None)	sns.barplot(x="cat", y="val", ci=95, data=df)
n_boot	Number of bootstraps for CI	sns.barplot(x="cat", y="val", n_boot=100, data=df)
palette	Color palette	sns.barplot(x="cat", y="val", palette="Set2", data=df)
saturation	Bar color saturation (0–1)	sns.barplot(x="cat", y="val", saturation=0.75, data=df)
width	Width of bars (0–1)	sns.barplot(x="cat", y="val", width=0.7, data=df)
orient	'v' (vertical) or 'h' (horizontal)	sns.barplot(y="cat", x="val", orient="h", data=df)
capsize	Add caps to error bars	sns.barplot(x="cat", y="val", capsize=0.2, data=df)
errwidth	Width of error bars	sns.barplot(x="cat", y="val", errwidth=2, data=df)

Code Example

```
import seaborn as sns
import pandas as pd

df = pd.DataFrame({
   "Category": ["A","A","B","B","C","C","D","D"],
   "Value": [4,5,6,7,2,3,5,6],
   "Group": ["X","Y","X","Y","X","Y","X","Y"]
})

sns.barplot(x="Category", y="Value", hue="Group", data=df,
          palette="Set2", ci=None, width=0.7)
plt.title("Seaborn Bar Chart")
plt.show()
```

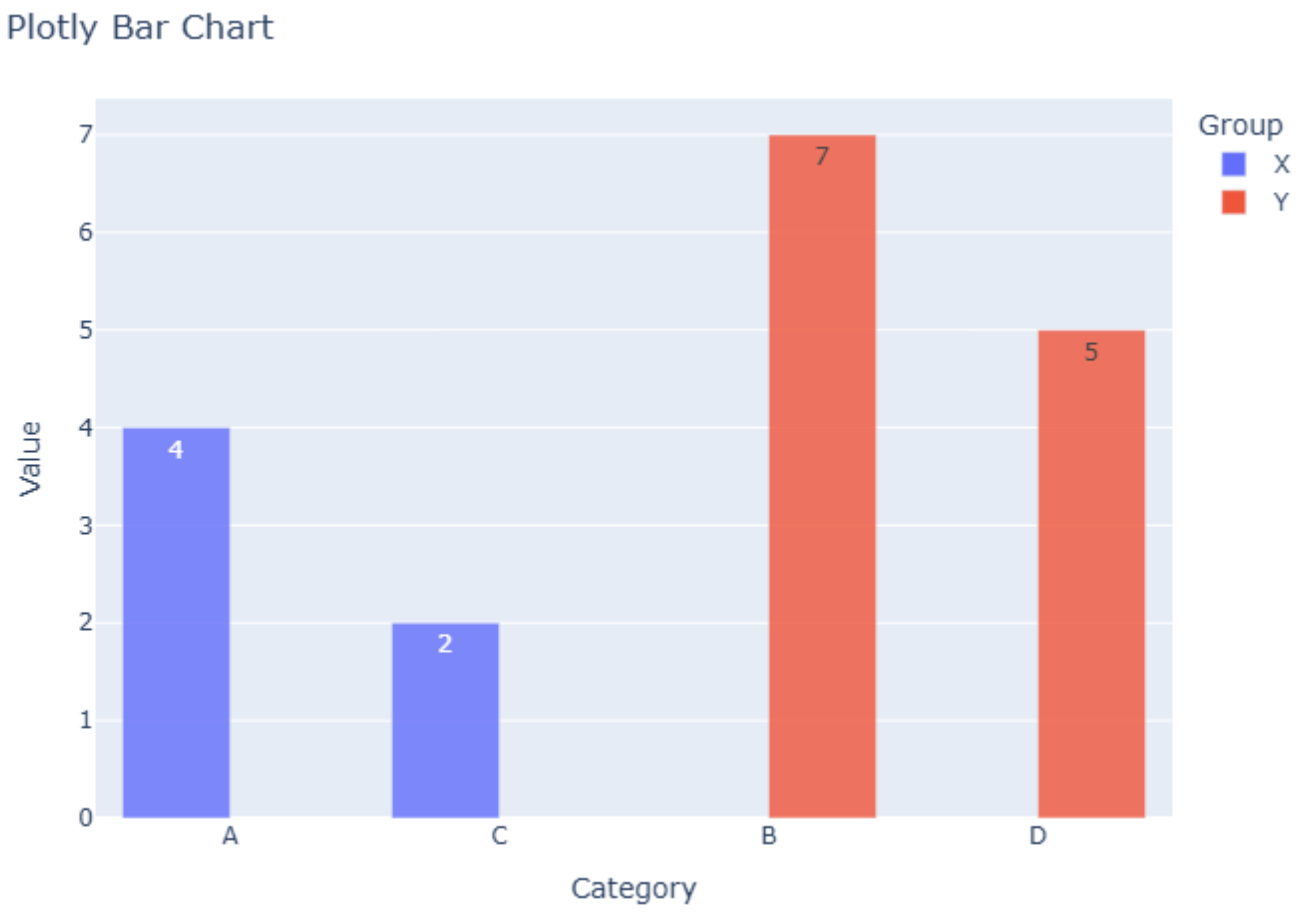


Plotly px.bar()

Parameter	Description	Example
data_frame	Data source	px.bar(df, x="Category", y="Value")
x / y	Variables for axes	px.bar(df, x="Category", y="Value")
color	Color by category	px.bar(df, x="Category", y="Value", color="Group")
barmode	'group', 'stack', 'relative', 'overlay'	px.bar(df, x="Category", y="Value", color="Group", barmode="stack")
orientation	'v' (vertical, default) or 'h'	px.bar(df, x="Category", y="Value", orientation="h")
text	Show labels on bars	px.bar(df, x="Category", y="Value", text="Value")
opacity	Transparency (0–1)	px.bar(df, x="Category", y="Value", opacity=0.7)
facet_row / facet_col	Facet charts (small multiples)	px.bar(df, x="Category", y="Value", facet_col="Group")
category_orders	Define category order	px.bar(df, x="Category", y="Value", category_orders={"Category":["D","C","B","A"]})
title	Add a chart title	px.bar(df, x="Category", y="Value", title="Bar Chart")

Code Example

```
import plotly.express as px
import pandas as pd

df = pd.DataFrame({
   "Category": ["A","B","C","D"],
   "Value": [4,7,2,5],
   "Group": ["X","Y","X","Y"]
})

fig = px.bar(df, x="Category", y="Value", color="Group",
          barmode="group", text="Value", opacity=0.8)
fig.update_layout(title="Plotly Bar Chart")
fig.show()
```



Pie Chart

What is a Pie Chart

A **pie chart** visualizes data as slices of a circle, where each slice represents the proportion of a category relative to the whole. The size (angle) of each slice corresponds to its value. Variations include **donut charts** (pie with a hole), and **nested pies** (concentric circles).

When to use a pie chart?

- To show **part-to-whole relationships**.
- When the number of categories is small (ideally fewer than 5–6).
- When exact comparisons between parts are not critical, but you want to give a quick impression of proportions.

When not to use a pie chart:

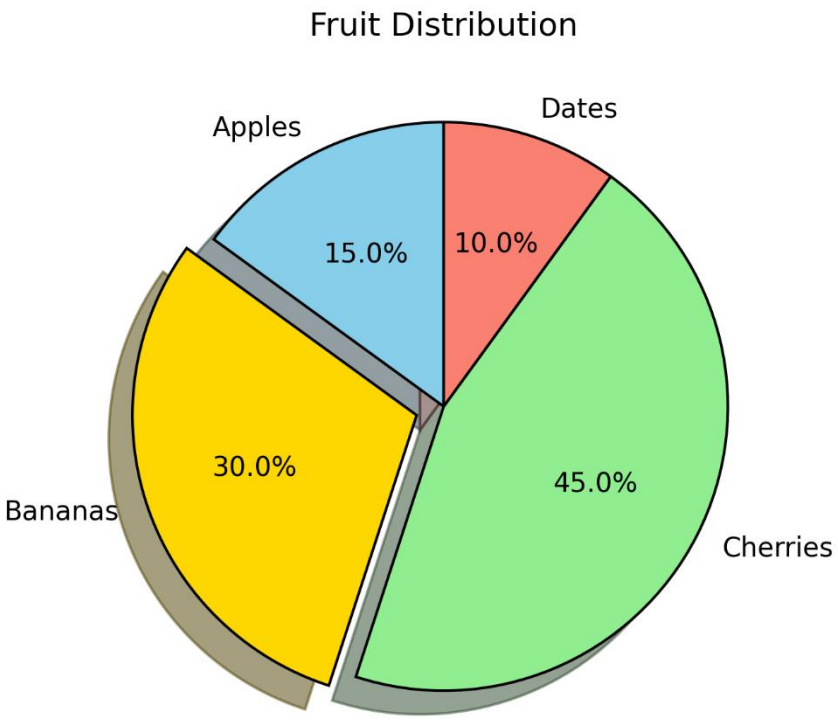
- When you have many categories (slices get too small).
- When precise comparison is needed (bar chart is better).
- When values can be negative (pie charts cannot represent this).

Matplotlib plt.pie()

Parameter	Description	Example Values
x	Data (array-like, values for slices)	[10, 20, 30]
labels	Category labels	["A", "B", "C"]
colors	Colors for slices	["red", "blue", "green"]
autopct	Show percentages on slices	"%1.1f%%", lambda p: f"{p:.2f}%"
explode	Offset slices from center	[0, 0.1, 0]
shadow	Draw shadow behind pie	True / False
startangle	Starting rotation angle (counterclockwise)	90
counterclock	Draw clockwise or counterclockwise	True / False
radius	Pie radius	1.0
wedgeprops	Properties for wedges	{"edgecolor":"black", "linewidth":2}
normalize	Normalize values to sum=1 (Matplotlib ≥ 3.4)	True / False

Code Example

```
import matplotlib.pyplot as plt

sizes = [15, 30, 45, 10]
labels = ["Apples", "Bananas", "Cherries", "Dates"]
colors = ["skyblue", "gold", "lightgreen", "salmon"]
explode = [0, 0.1, 0, 0]  # explode 2nd slice

plt.pie(
   sizes,
   labels=labels,
   colors=colors,
   autopct="%1.1f%%",
   shadow=True,
   explode=explode,
   startangle=90,
   wedgeprops={"edgecolor":"black"}
)
plt.title("Fruit Distribution")
plt.show()
```

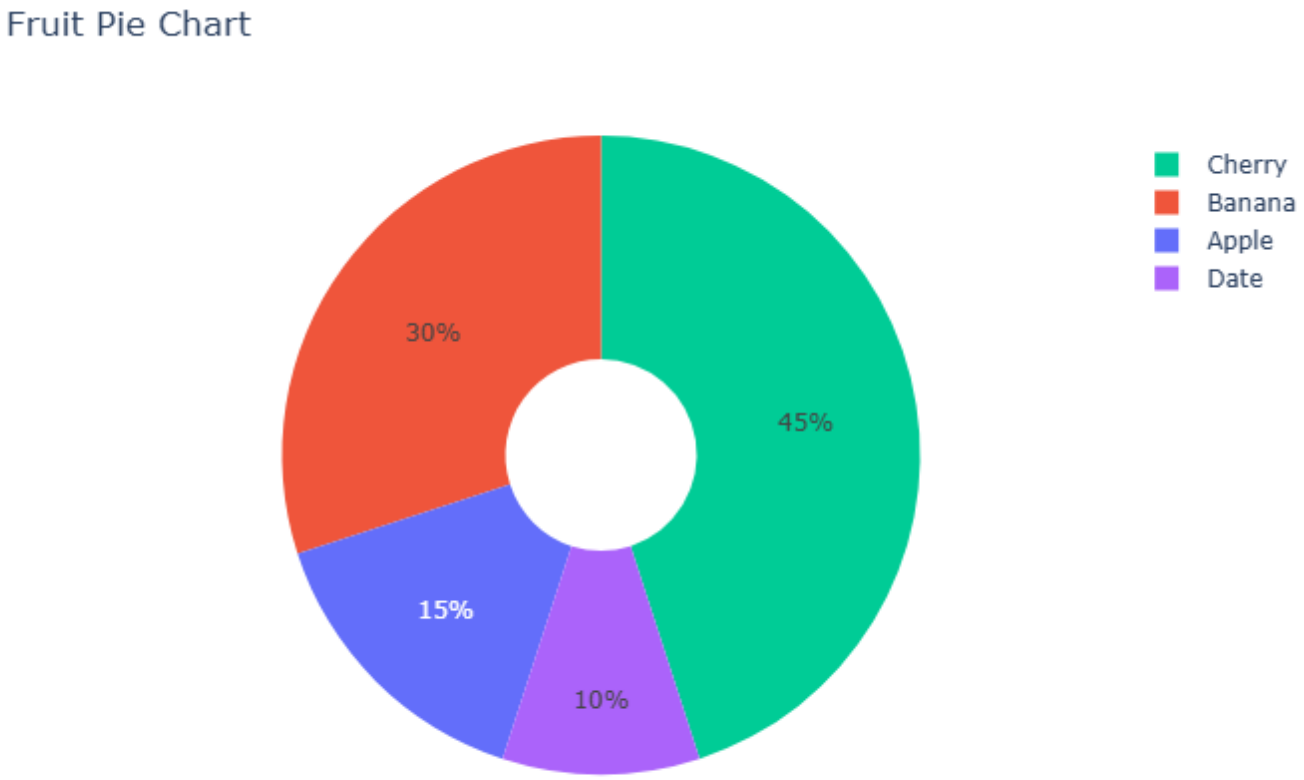


Plotly px.pie()

Parameter	Description	Example
data_frame	Pandas DataFrame	df
values	Column with slice sizes	"values"
names	Column with labels	"categories"
color	Category column for color mapping	"categories"
hole	Size of hole (0 = pie, >0 = donut)	0.4
title	Title of chart	"Pie Chart"
color_discrete_sequence	Custom colors	["red", "blue"]

Code Example

```
import plotly.express as px
import pandas as pd

df = pd.DataFrame({
   "Fruit": ["Apple", "Banana", "Cherry", "Date"],
   "Count": [15, 30, 45, 10]
})

fig = px.pie(
   df,
   names="Fruit",
   values="Count",
   color="Fruit",
   hole=0.3,  # donut chart
   title="Fruit Pie Chart"
)
fig.show()
```



Box Plot

What is a Box Plot?

A **box plot** summarizes the **distribution of a dataset** using five key statistics:

- **Minimum** (lowest point excluding outliers)
- **First quartile (Q1)** – 25th percentile
- **Median (Q2)** – 50th percentile
- **Third quartile (Q3)** – 75th percentile
- **Maximum** (highest point excluding outliers)

The box itself represents the **IQR (interquartile range)**, whiskers extend to min/max values, and points beyond whiskers are outliers.

When to use a box plot?

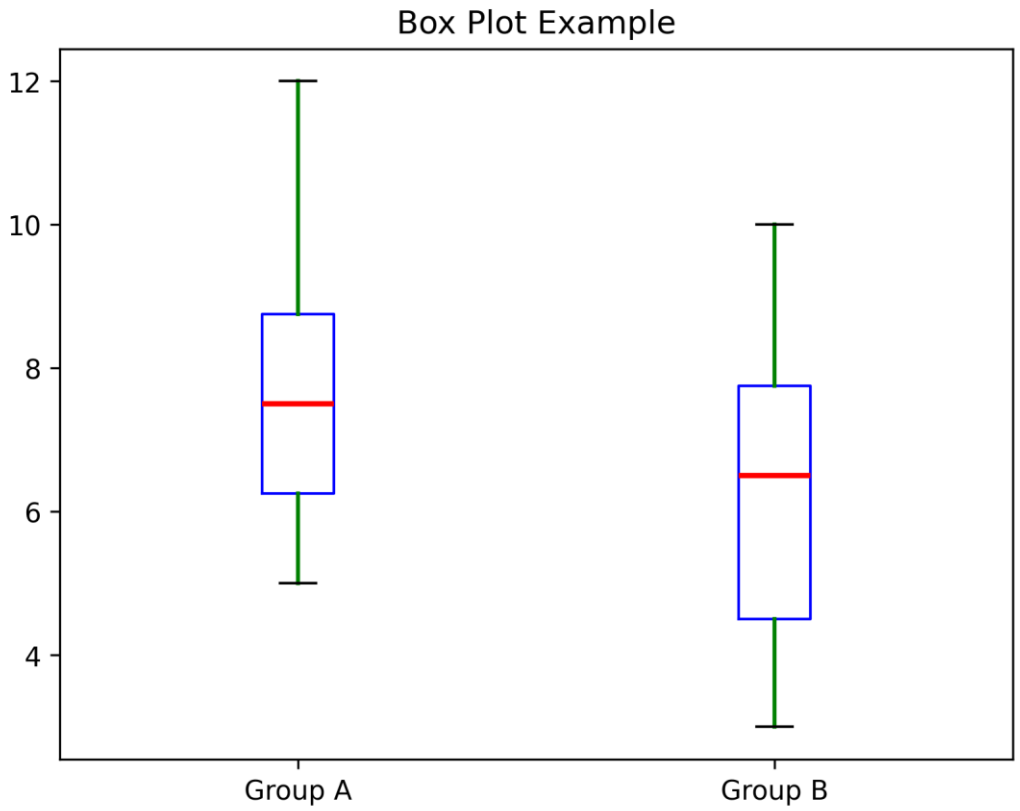
- To visualize **spread and skewness** of data.
- To compare **distributions across categories**.
- To identify **outliers**.
- To compare **central tendency** and **variability** between groups.

Matplotlib plt.boxplot()

Parameter	Description	Example Values
x	Data (array-like or list of arrays)	[7,8,5,6,9,12]
notch	Notched box to show CI around median	True / False
vert	Vertical (True) or horizontal (False)	True
patch_artist	Fill boxes with color	True
tick_labels	Category labels	["Group A", "Group B"]
showmeans	Show mean as point/line	True / False
boxprops, whiskerprops, capprops, medianprops	Styling options	{"color":"blue"}

Code Example

```
import matplotlib.pyplot as plt

data = [ [7, 8, 5, 6, 9, 12], [6, 7, 3, 4, 8, 10] ]

plt.boxplot(
   data,
   vert=True,
   tick_labels=["Group A", "Group B"],

   boxprops=dict(color="blue"),
   medianprops=dict(color="red", linewidth=2),
   whiskerprops=dict(color="green", linewidth=1.5)

)
plt.title("Box Plot Example")
plt.show()
```

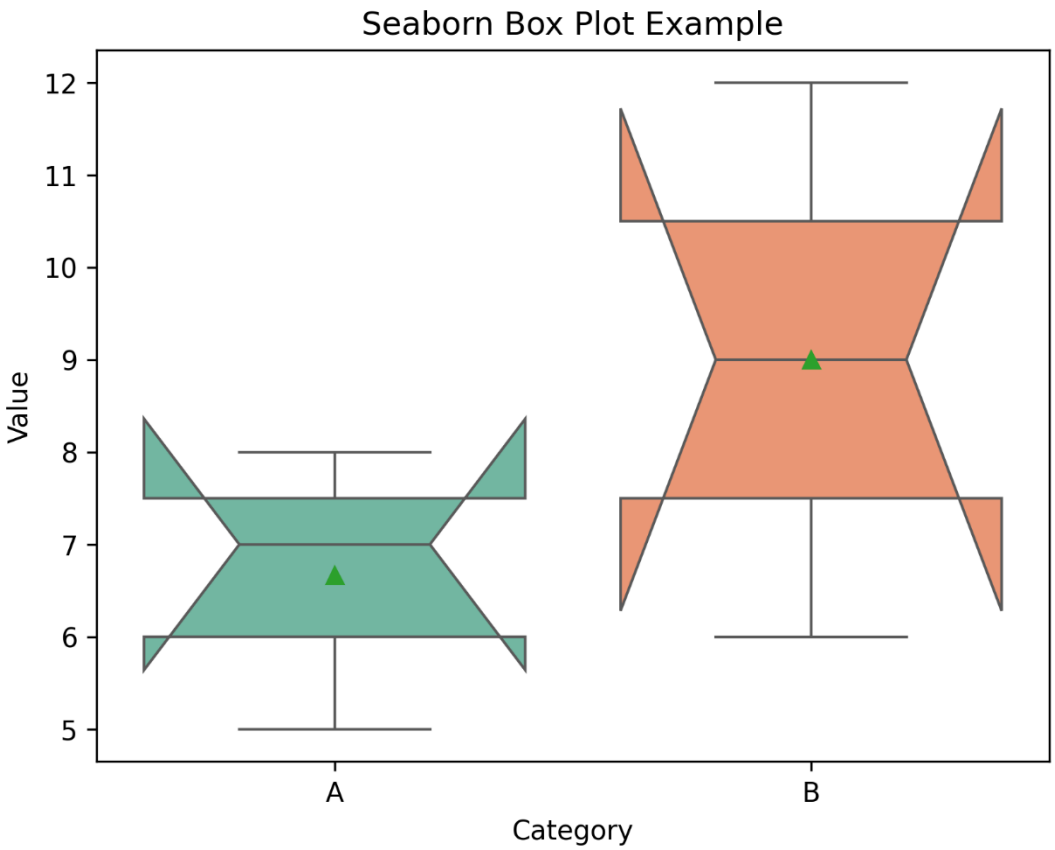


Seaborn sns.boxplot()

Parameter	Description	Example
x, y	Axis variables	x="category", y="value"
data	Pandas DataFrame	df
hue	Group by another category	hue="gender"
palette	Colors	"Set2"
orient	"v" (vertical) or "h" (horizontal)	"h"
width	Width of boxes	0.6
showmeans	Show mean value	True
notch	Notch around median (like matplotlib)	True

Code Example

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

df = pd.DataFrame({
   "Category": ["A","A","A","B","B","B"],
   "Value": [7,8,5,6,9,12]
})

sns.boxplot(
   x="Category", y="Value", data=df,
   palette="Set2", showmeans=True, notch=True
)
plt.title("Seaborn Box Plot Example")
plt.show()
```

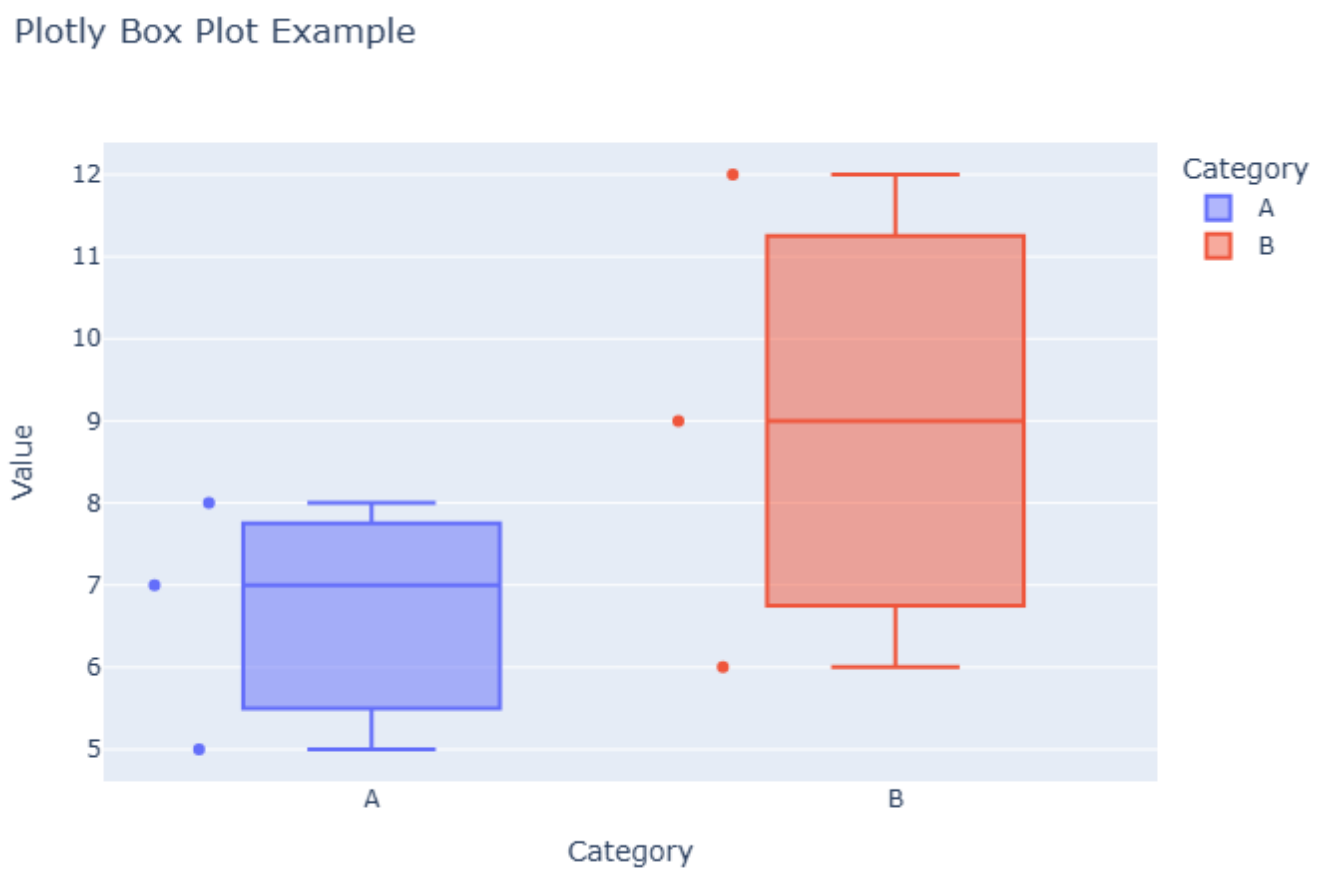


Plotly px.box()

Parameter	Description	Example
data_frame	Pandas DataFrame	df
x, y	Axis variables	"Category", "Value"
color	Group by another category	"Gender"
points	Show all points ("all", "outliers", "suspectedoutliers", False)	"all"
notched	Draw notch	True / False
title	Chart title	"Box Plot"

Code Example

```
import plotly.express as px
import pandas as pd

df = pd.DataFrame({
   "Category": ["A","A","A","B","B","B"],
   "Value": [7,8,5,6,9,12]
})

fig = px.box(
   df,
   x="Category", y="Value",
   color="Category",
   points="all",  # show individual data points
   title="Plotly Box Plot Example"
)
fig.show()
```



Violin Plot

What is a Violin Plot?

A **violin plot** is similar to a box plot but adds **distribution shape** using a mirrored KDE (density curve). It shows:

- The **quartiles** (like a box plot).
- The **median**.
- The **distribution/density** (fat where many values cluster, thin where rare).

It looks like a violin 🎻 (hence the name).

When to use a violin plot?

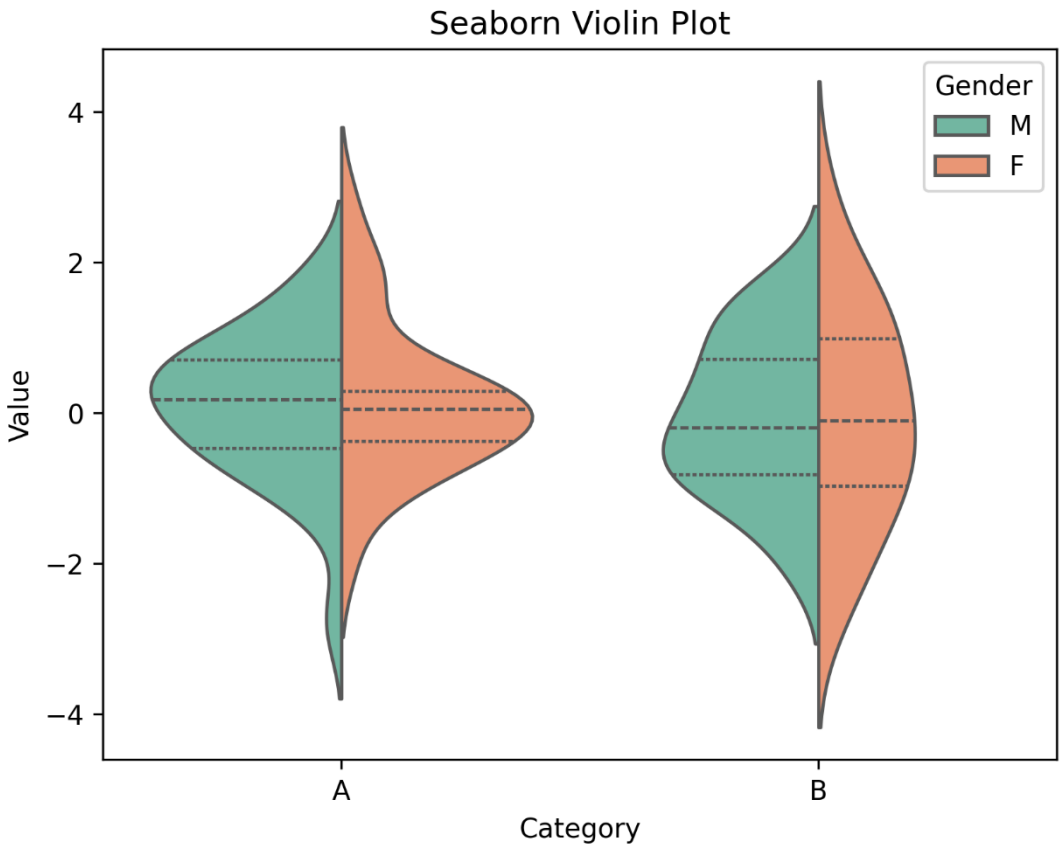
- When you want to show **both summary statistics** (median, quartiles) **and distribution**.
- When comparing **distributions across multiple categories**.
- Useful for **skewed data** or when outliers are frequent.

Seaborn sns.violinplot()

Parameter	Description	Example
x, y	Axis variables	x="Category", y="Value"
data	DataFrame	df
hue	Subcategory splitting	hue="Gender"
split	Split violins for hue	True/False
inner	Show box/point/stick inside violin	"box", "quartile", "point", "stick", None
bw	Bandwidth for KDE	0.2
scale	Scale violins by "area", "count", "width"	"area"
palette	Colors	"Set2"

Code Example

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

df = pd.DataFrame({
   "Category": ["A"]*50 + ["B"]*50,
   "Value": np.random.randn(100),
   "Gender": ["M"]*25 + ["F"]*25 + ["M"]*25 + ["F"]*25
})

sns.violinplot(
   x="Category", y="Value", data=df,
   hue="Gender", split=True,
   inner="quartile", palette="Set2"
)
plt.title("Seaborn Violin Plot")
plt.show()
```

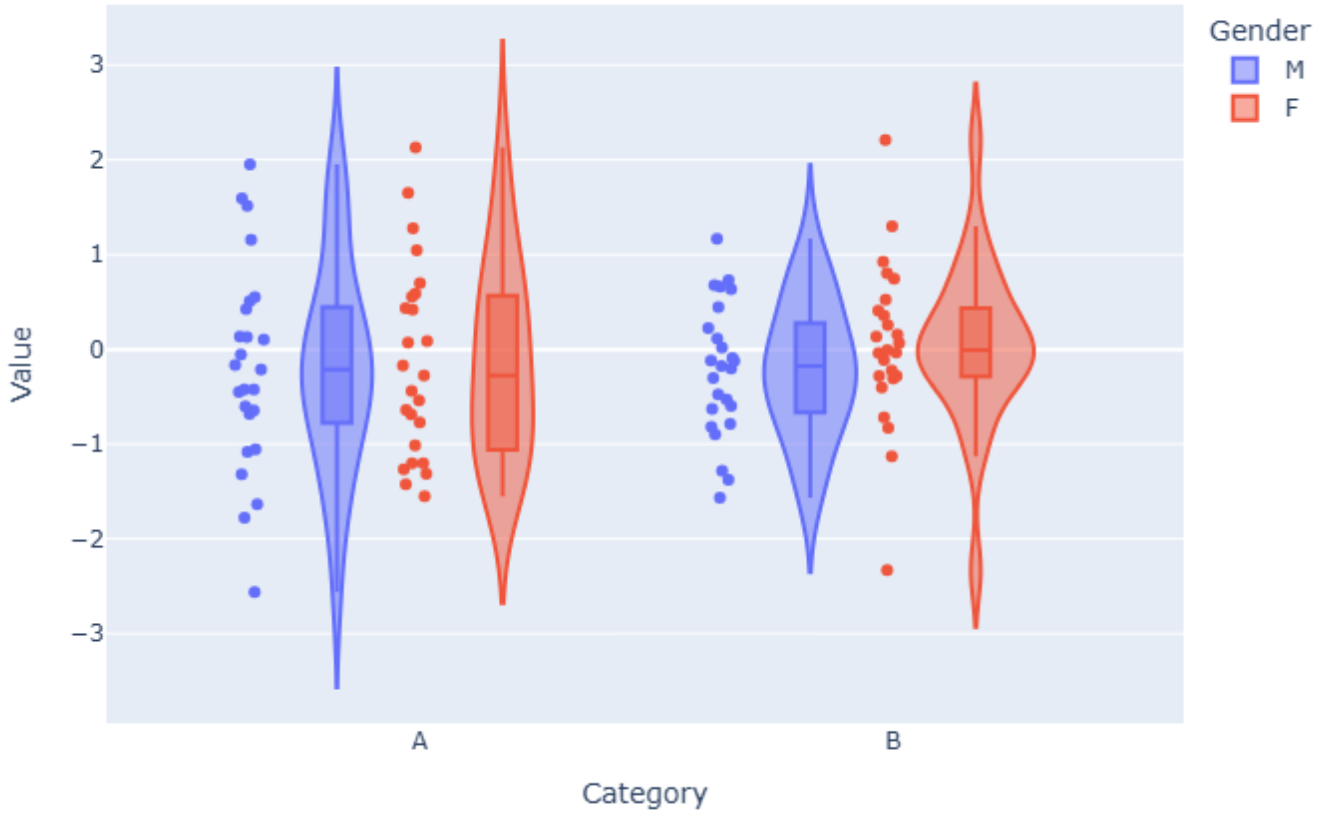


Plotly px.violin()

Parameter	Description	Example
data_frame	DataFrame	df
x, y	Axes	"Category", "Value"
color	Split violins by subgroup	"Gender"
box	Add a mini box plot inside violin	True
points	Show all points ("all", "outliers", False)	"all"
hover_data	Extra info on hover	["Age"]

Code Example

```
import plotly.express as px

fig = px.violin(
    df,
    x="Category", y="Value",
    color="Gender",
    box=True, points="all",
    hover_data=df.columns
)
fig.show()
```



Scatter Plot

What is a Scatter Plot?

A **scatter plot** is a chart that displays points on a 2D plane based on two variables (x, y). Each point represents an observation.

When to use a scatter plot?

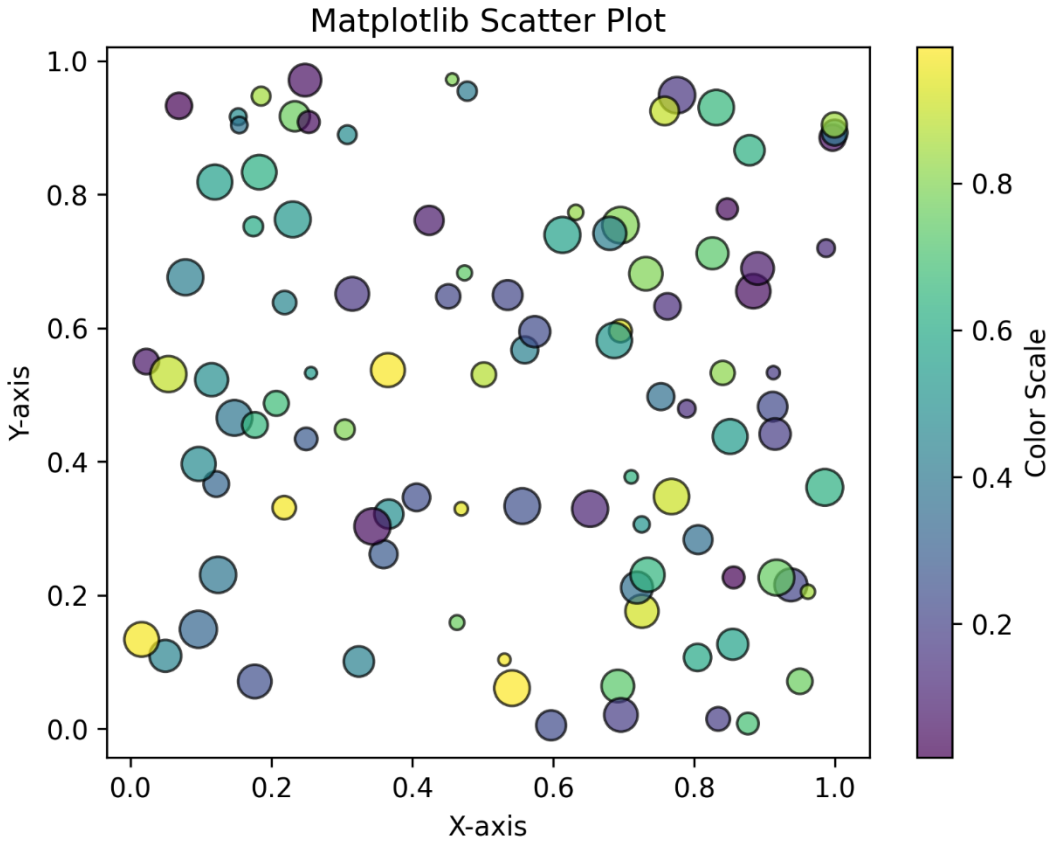
- To examine **relationships/correlations** between two continuous variables.
- To detect **clusters, trends, or outliers**.
- To check **linear/non-linear relationships** before modeling.

Matplotlib plt.scatter()

Parameter	Description	Example
x, y	Coordinates of points	plt.scatter(x, y)
c	Point colors (array or colormap)	c="red" / c=values
s	Point size	s=50
alpha	Transparency	alpha=0.6
marker	Shape of marker	"o", "x", "s"
cmap	Colormap if c is numeric	"viridis"

Code Example

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

x = np.random.rand(100)
y = np.random.rand(100)
sizes = np.random.randint(20, 200, 100)
colors = np.random.rand(100)

plt.scatter(x, y, s=sizes, c=colors, cmap="viridis", alpha=0.7, edgecolors="black")
plt.colorbar(label="Color Scale")
plt.title("Matplotlib Scatter Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

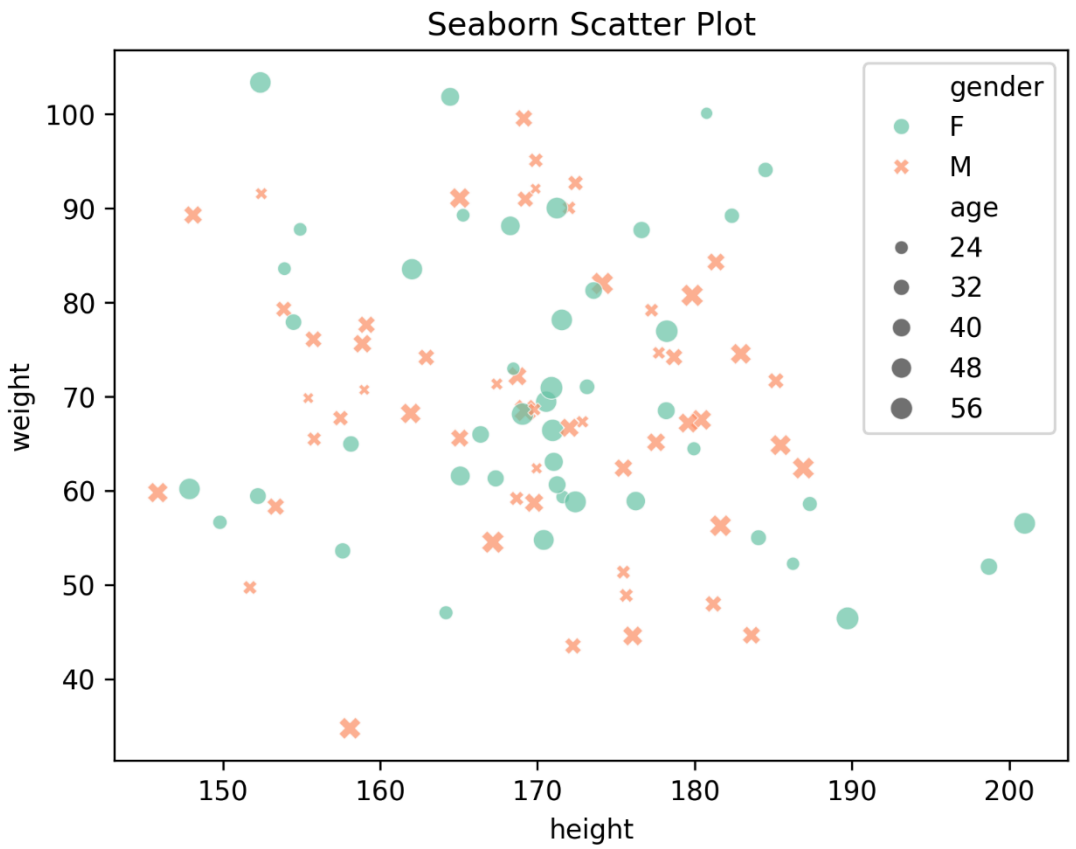


Seaborn sns.scatterplot()

Parameter	Description	Example
x, y	Variables	x="height", y="weight"
data	DataFrame	df
hue	Color grouping	hue="species"
style	Marker style grouping	style="gender"
size	Size grouping	size="age"
palette	Color palette	"Set2"
alpha	Transparency	0.6

Code Example

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

df = pd.DataFrame({
   "height": np.random.normal(170, 10, 100),
   "weight": np.random.normal(70, 15, 100),
   "gender": np.random.choice(["M", "F"], 100),
   "age": np.random.randint(18, 60, 100)
})

sns.scatterplot(
   x="height", y="weight", data=df,
   hue="gender", size="age", style="gender",
   palette="Set2", alpha=0.7
)
plt.title("Seaborn Scatter Plot")
plt.show()
```

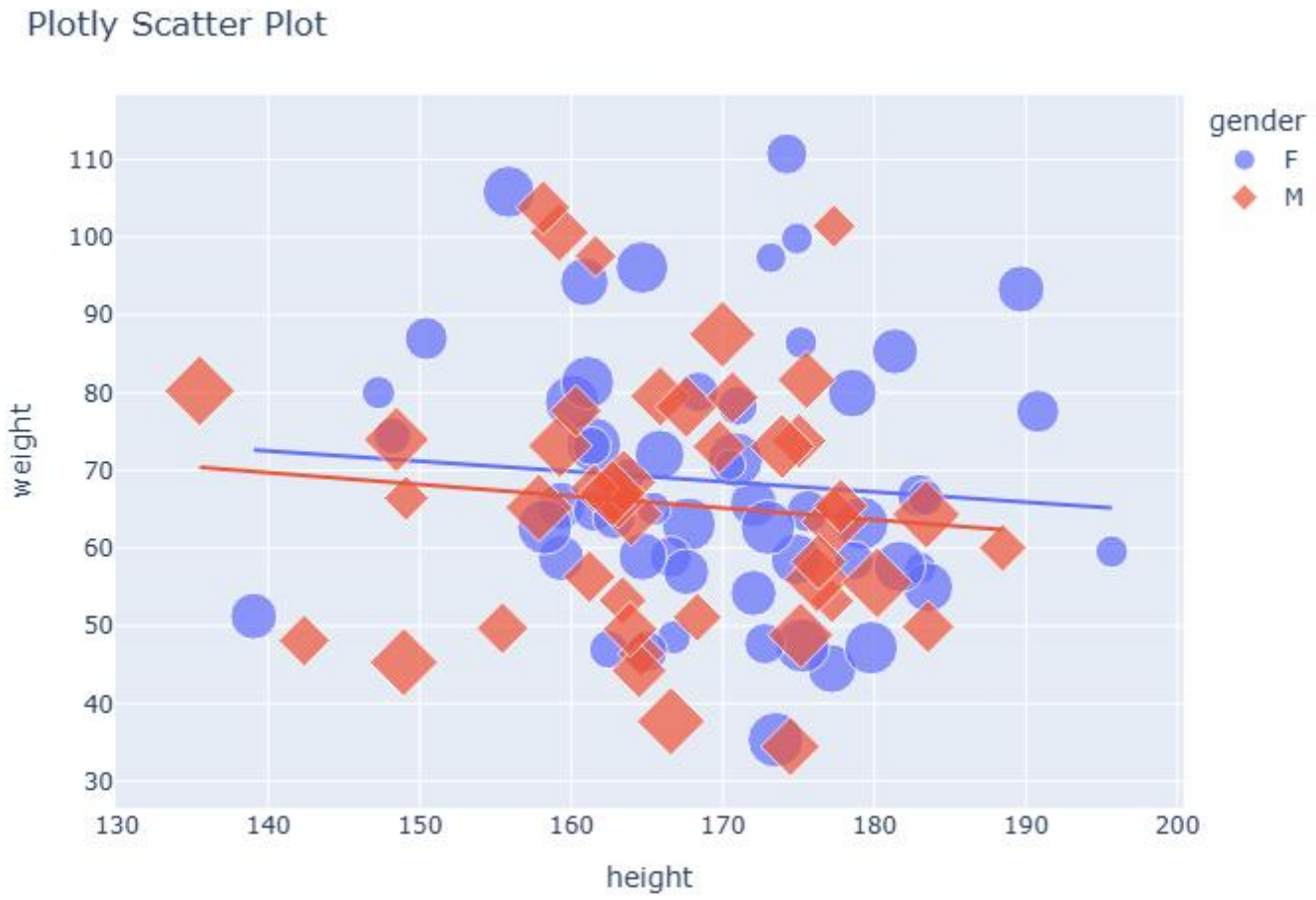


Plotly px.scatter()

Parameter	Description	Example
data_frame	DataFrame	df
x, y	Variables	"height", "weight"
color	Color grouping	"gender"
size	Size grouping	"age"
symbol	Marker symbol grouping	"gender"
hover_data	Extra info on hover	["age"]
trendline	Regression line	"ols"

Code Example

```
import plotly.express as px
import statsmodels.api as sm

fig = px.scatter(
    df,
    x="height", y="weight",
    color="gender", size="age",
    symbol="gender", hover_data=["age"],
    trendline="ols"
)
fig.update_layout(title="Plotly Scatter Plot")
fig.show()
```



Line Plot

What is a Line Plot?

A **line plot** shows data points connected by straight lines. It's often used to visualize **trends, patterns, or changes** over a continuous variable (like time).

When to use a line plot?

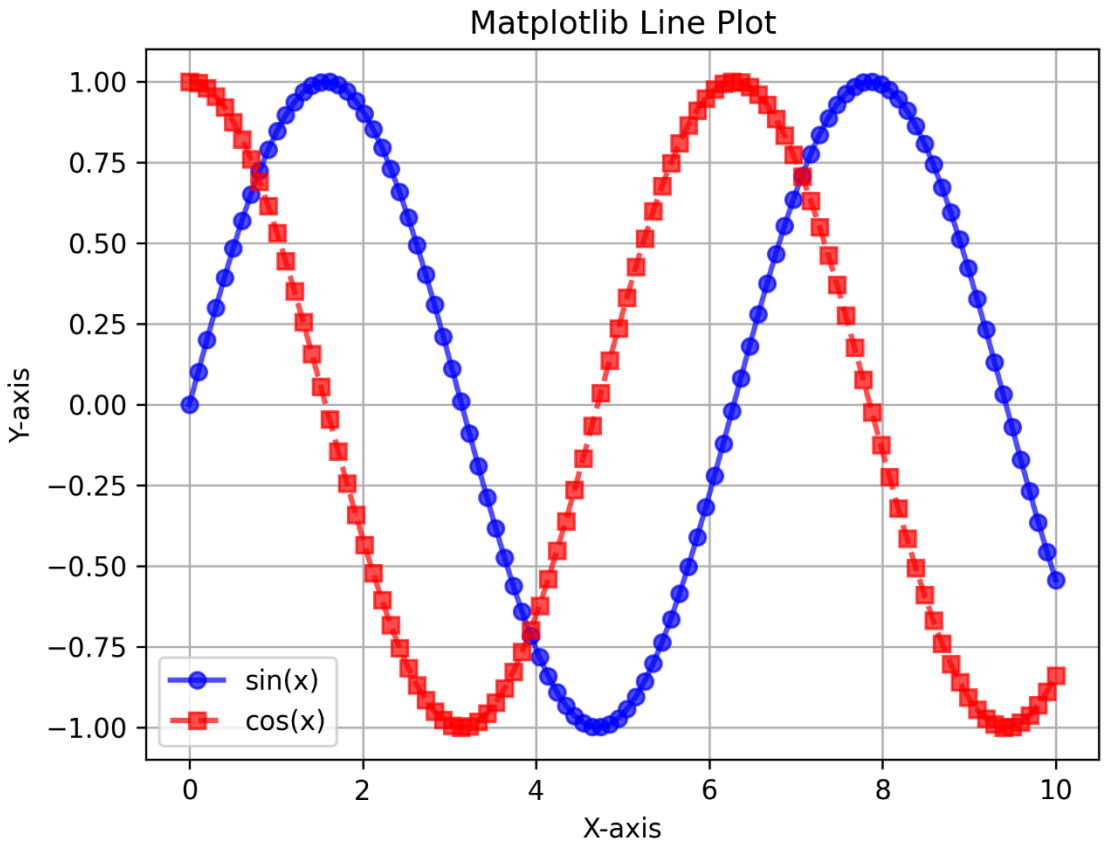
- To visualize **time series data** (stock prices, weather, etc.).
- To compare **multiple series** across the same x-axis.
- To show **continuous trends** instead of discrete points.

Matplotlib plt.plot()

Parameter	Description	Example
x, y	Coordinates	plt.plot(x, y)
color	Line color	"red", "blue"
linestyle (ls)	Line style	"-", "--", ":", "-."
linewidth (lw)	Line thickness	2
marker	Marker style	"o", "x", "s"
alpha	Transparency	0.7
label	Legend label	"Series 1"

Code Example

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

x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y1, color="blue", linestyle="-", linewidth=2, marker="o", alpha=0.7, label="sin(x)")
plt.plot(x, y2, color="red", linestyle="--", linewidth=2, marker="s", alpha=0.7, label="cos(x)")
plt.title("Matplotlib Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)
plt.show()
```

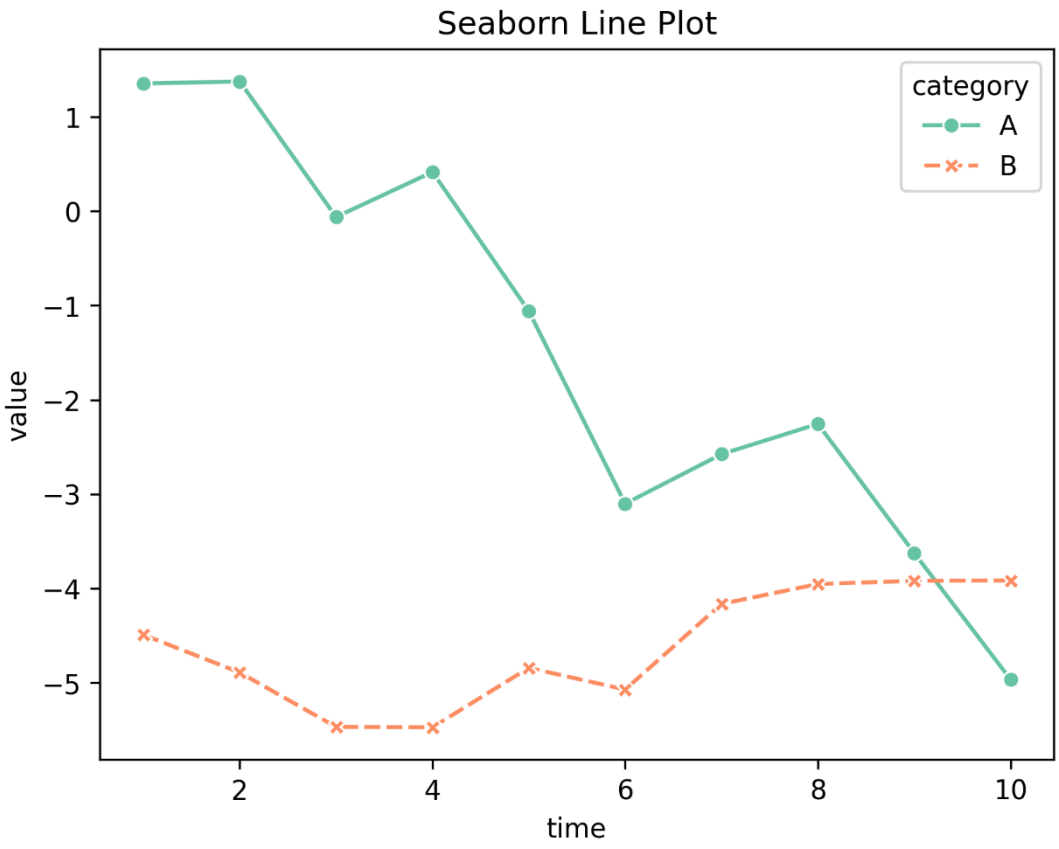


Seaborn sns.lineplot()

Parameter	Description	Example
x, y, data	Variables + DataFrame	sns.lineplot(x="time", y="val", data=df)
hue	Grouping by color	hue="category"
style	Different line styles	style="group"
size	Line thickness	size="importance"
markers	Show markers	True
errorbar	Confidence interval	95 (default)
palette	Color palette	"Set2"

Code Example

```
import seaborn as sns
import pandas as pd
import numpy as np

df = pd.DataFrame({
   "time": np.tile(np.arange(1, 11), 2),
   "value": np.random.randn(20).cumsum(),
   "category": ["A"]*10 + ["B"]*10
})

sns.lineplot(
   x="time", y="value", data=df,
   hue="category", style="category",
   markers=True, errorbar=None, palette="Set2"
)
plt.title("Seaborn Line Plot")
plt.show()
```

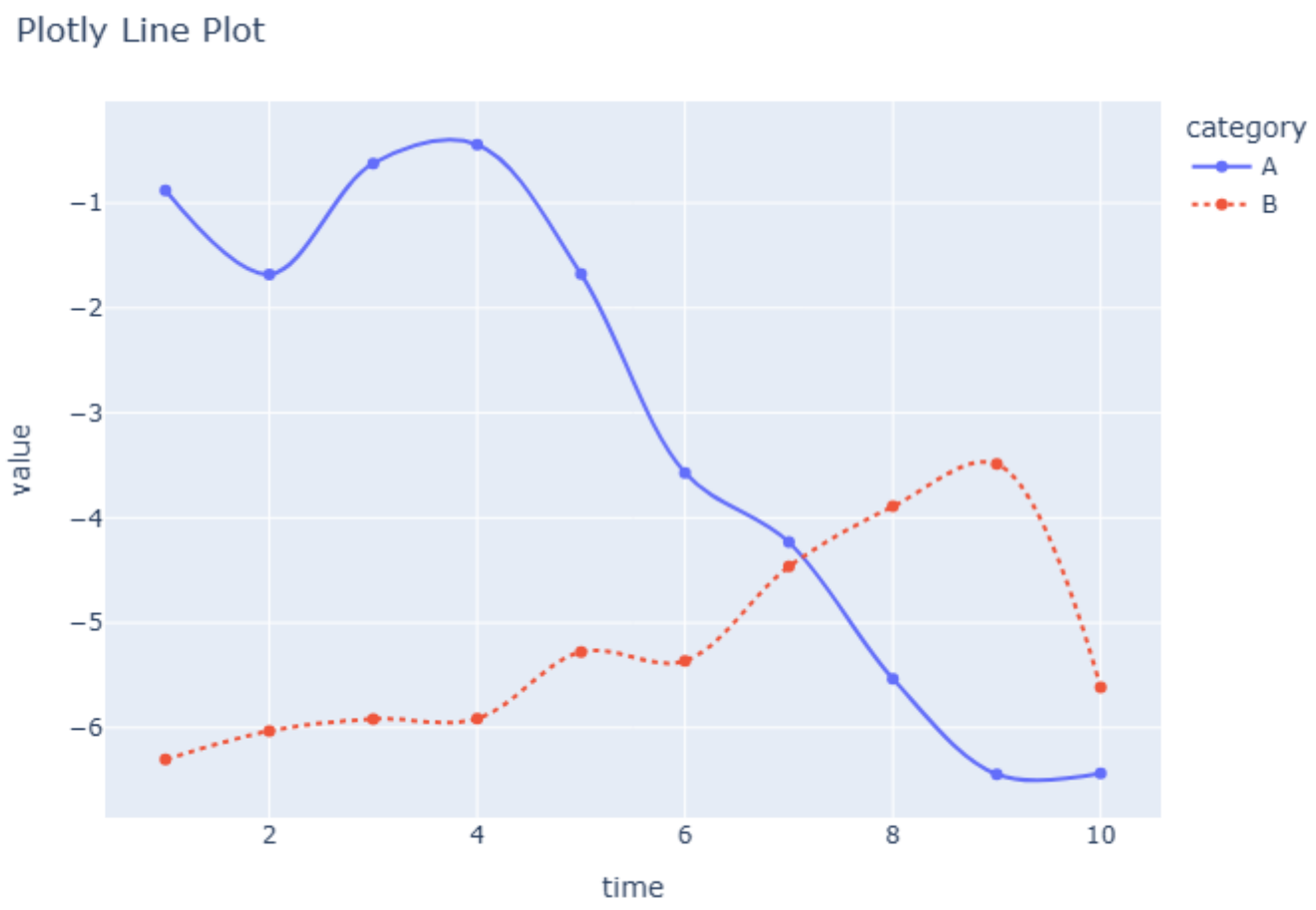


Plotly px.line()

Parameter	Description	Example
data_frame	Data source	df
x, y	Variables	"time", "value"
color	Grouping	"category"
line_dash	Line style	"category"
markers	Show markers	True
hover_data	Extra info on hover	["time"]
line_shape	Line interpolation	"linear", "spline", "hv", "vh", "hvh"

Code Example

```
import plotly.express as px

fig = px.line(
   df, x="time", y="value", color="category",
   line_dash="category", markers=True,
   hover_data=["time"], line_shape="spline"
)
fig.update_layout(title="Plotly Line Plot")
fig.show()
```



Heatmap

What is a Heatmap?

A **heatmap** is a 2D graphical representation where values in a matrix are represented as colors. Each cell's intensity corresponds to its value.

When to use it?

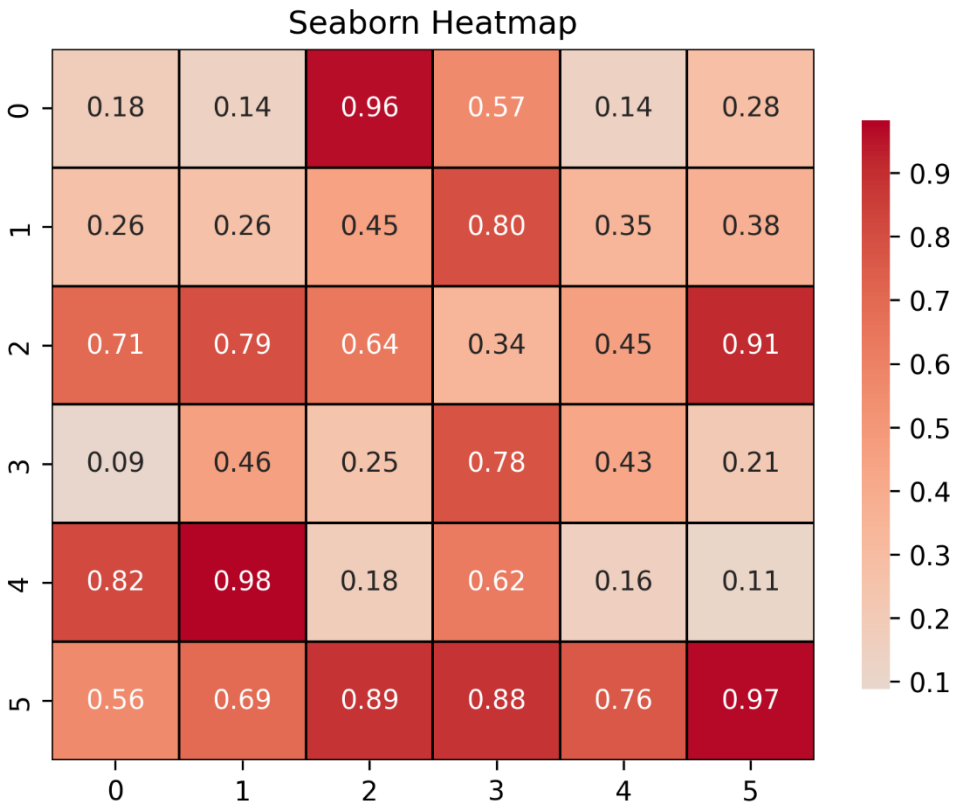
- To visualize **correlation matrices** (relationships between features).
- To represent **frequency/intensity** across 2D grids (e.g., pixels, geographic data).
- To **spot patterns, clusters, or anomalies** in datasets.
- To visualize the **correctness of predictions** with respect to actual values

Seaborn sns.heatmap()

Parameter	Description	Example
data	2D data/matrix	sns.heatmap(data)
annot	Show values inside cells	True
fmt	Format for annotations	".2f"
cmap	Colormap	"coolwarm", "viridis"
center	Center value	0
linewidths	Space between cells	0.5
linecolor	Color of lines	"white"
cbar	Show color bar	True/False
cbar_kws	Customize colorbar	{"shrink": 0.7}
mask	Hide certain cells	mask=np.triu(np.ones_like(data))

Code Example

```
import seaborn as sns
import numpy as np

data = np.random.rand(6, 6)

sns.heatmap(
   data, annot=True, fmt=".2f",
   cmap="coolwarm", center=0,
   linewidths=0.5, linecolor="black",
   cbar=True, cbar_kws={"shrink": 0.8}
)
plt.title("Seaborn Heatmap")
plt.show()
```

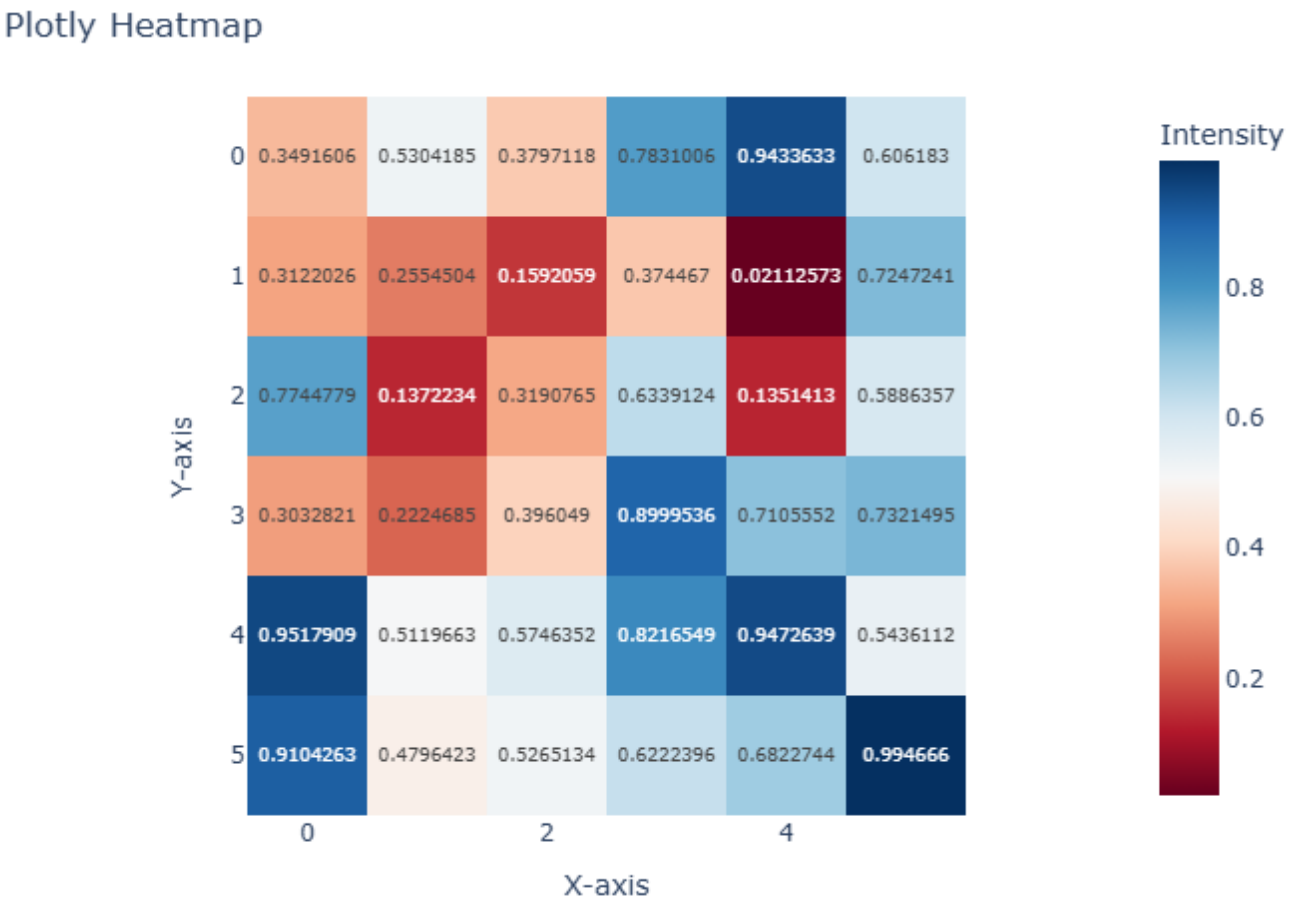


Plotly px.imshow() / go.Heatmap()

Function / Parameter	Description	Example
px.imshow()	High-level function for quick heatmaps (like Matplotlib's imshow) with optional labels and annotations.	px.imshow(data, color_continuous_scale="RdBu")
go.Heatmap()	Low-level, fully customizable heatmap object with more styling flexibility.	go.Heatmap(z=data, colorscale="Viridis")
z	The actual 2D data (matrix values) that define the color intensities.	go.Heatmap(z=[[1,2],[3,4]])
x, y	Labels for the columns (x) and rows (y). Useful for categorical data.	px.imshow(data, x=["Mon","Tue"], y=["A","B"])
colorscale	Controls the color scheme. Options include "Viridis", "Cividis", "Plasma", "Jet", "RdBu", "Greens", etc.	go.Heatmap(z=data, colorscale="Cividis")
zmin, zmax	Fixes the lower and upper bounds for the color scale (default = min and max of z). Useful for comparing multiple heatmaps.	go.Heatmap(z=data, zmin=0, zmax=1)
text_auto (only in px.imshow)	Automatically annotate each cell with its numeric value. Formatting can be adjusted.	px.imshow(data, text_auto=True)
hoverongaps	Whether gaps (NaN values) should appear in hover tooltips. Default = True.	go.Heatmap(z=[[1, None],[3, 4]], hoverongaps=False)
colorbar / coloraxis_colorbar	Customize the colorbar (title, ticks, size).	go.Heatmap(z=data, colorbar=dict(title="Intensity"))
reversescale	Reverses the order of the colormap.	go.Heatmap(z=data, colorscale="Viridis", reversescale=True)

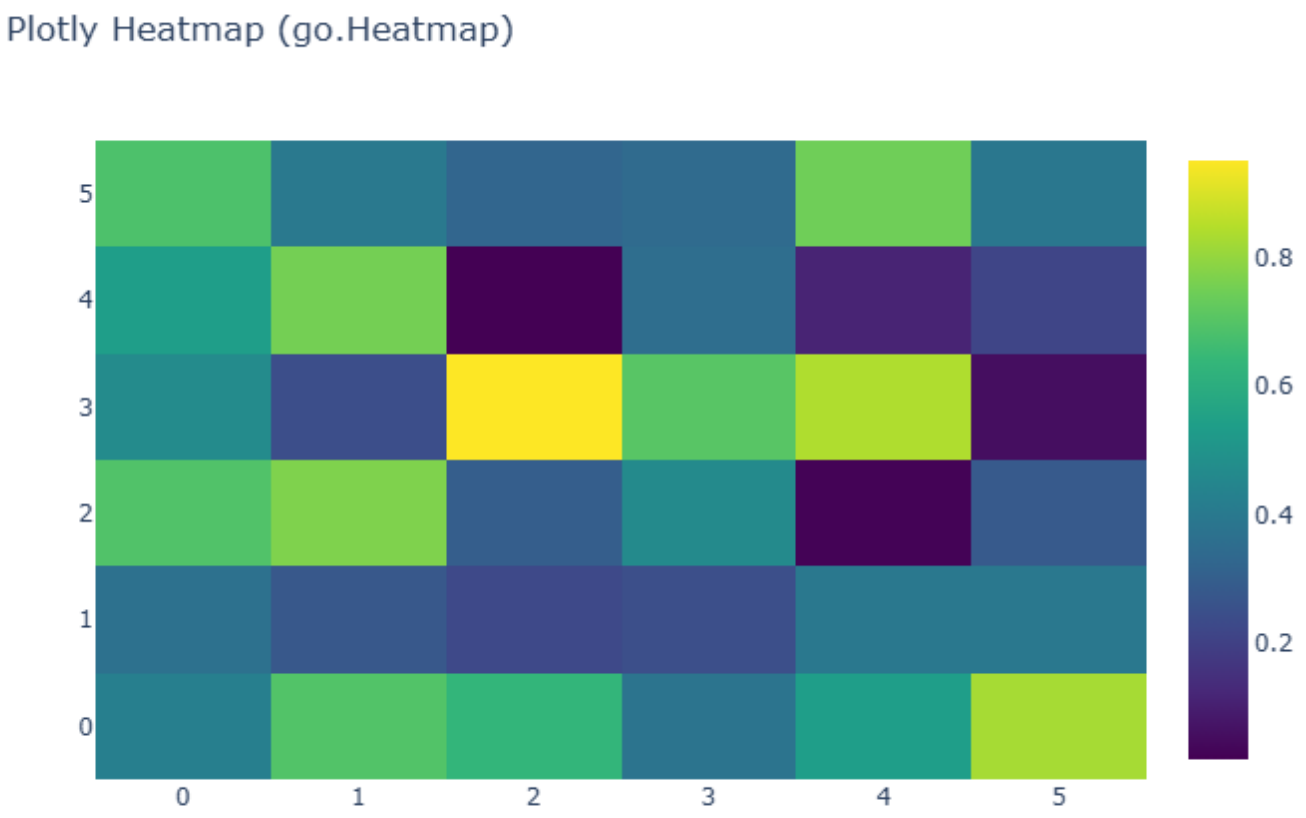
Code Example

```
import plotly.express as px
import numpy as np

data = np.random.rand(6, 6)

fig = px.imshow(
   data, text_auto=True, color_continuous_scale="RdBu",
   labels=dict(x="X-axis", y="Y-axis", color="Intensity")
)
fig.update_layout(title="Plotly Heatmap")
fig.show()
```



Code Example

```
import plotly.graph_objects as go

fig = go.Figure(data=go.Heatmap(
   z=data,
   colorscale="Viridis"
))
fig.update_layout(title="Plotly Heatmap (go.Heatmap)")
fig.show()
```



Pair plot

What is a Pair Plot?

A **pair plot** is a grid of plots that shows the pairwise relationships between variables in a dataset. It combines:

- **Scatterplots** (for variable pairs, numeric vs. numeric).
- **Histograms / KDE plots** (on the diagonal, showing distribution of each variable).

When to use it?

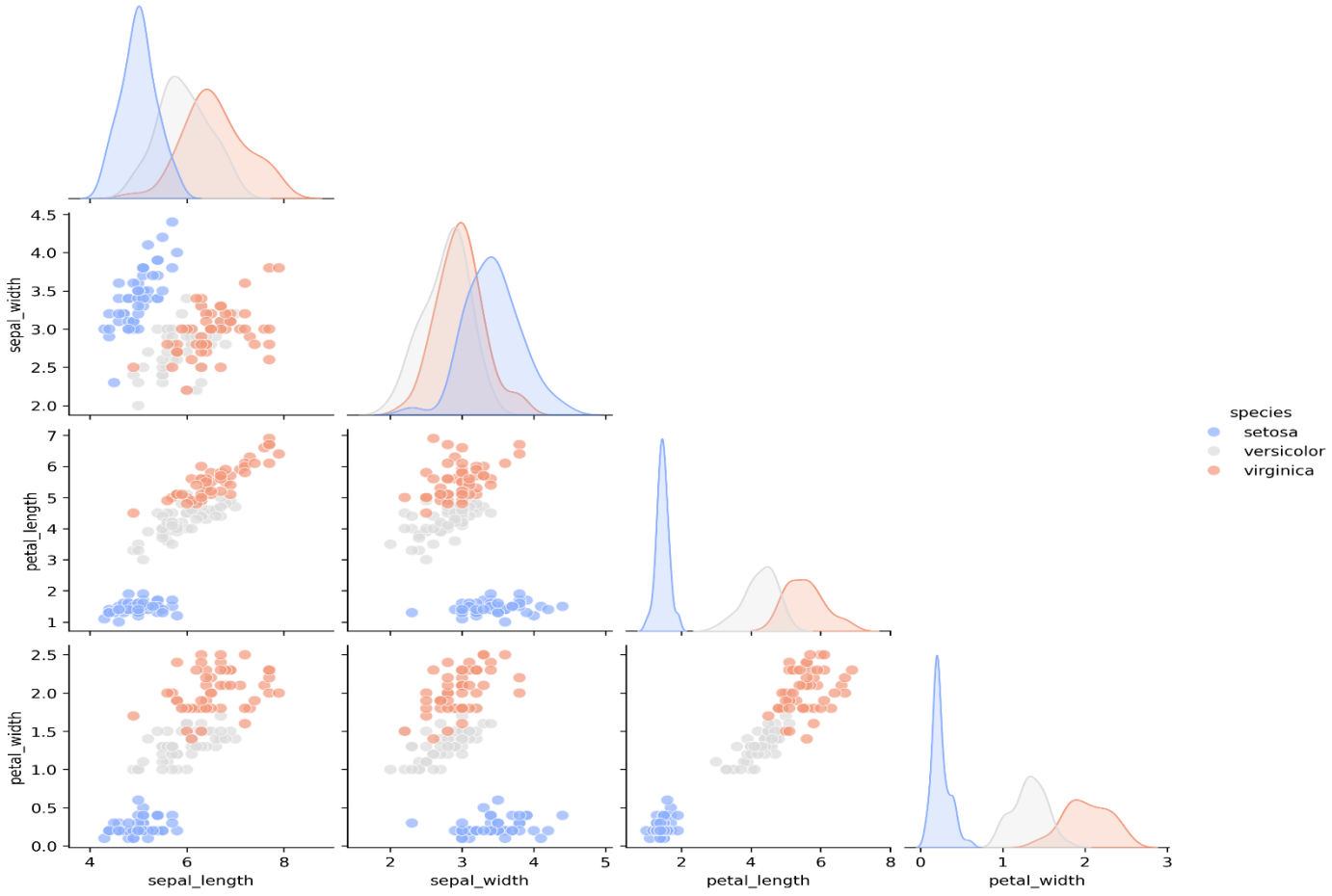
- **Exploratory Data Analysis (EDA)**: To quickly understand relationships and correlations.
- Detect **linear/non-linear relationships** between variables.
- Spot **outliers**.
- Compare **distributions** of variables.

Seaborn sns.pairplot()

Parameter	Description	Example
data	DataFrame input	sns.pairplot(df)
hue	Grouping variable to color-code points	sns.pairplot(df, hue="species")
palette	Color palette for groups	sns.pairplot(df, hue="species", palette="coolwarm")
kind	Plot type for off-diagonal ("scatter", "kde", "hist", "reg")	sns.pairplot(df, kind="reg")
diag_kind	Plot type on diagonal ("hist" or "kde")	sns.pairplot(df, diag_kind="kde")
markers	Marker style per category	sns.pairplot(df, hue="species", markers=["o","s","D"])
corner	Show only lower triangle (avoid duplicates)	sns.pairplot(df, corner=True)
plot_kws	Dict of keyword args for plots	sns.pairplot(df, plot_kws={"alpha":0.6})
diag_kws	Dict of keyword args for diagonal plots	sns.pairplot(df, diag_kind="hist", diag_kws={"bins":15})
height	Size of each subplot (inches)	sns.pairplot(df, height=2.5)

Code Example

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

# Load sample dataset
iris = load_dataset("iris")

# Basic pairplot
sns.pairplot(iris)
plt.show()

# Pairplot with grouping + KDE + corner mode
sns.pairplot(
    iris,
    hue="species",
    palette="coolwarm",
    kind="scatter",
    diag_kind="kde",
    corner=True,
    plot_kws={"alpha": 0.7, "s": 60},  # scatter options
    diag_kws={"fill": True},          # KDE options
    height=2.5
)
plt.show()
```



Pair Plot

What is a Joint Plot?

A **joint plot** is a **bivariate plot** that combines:

- A **scatter plot** (or hexbin/2D KDE) showing the relationship between two variables.
- **Marginal plots** (histograms/KDEs) along the **top and right axes** to show the distribution of each variable individually.

So, you get both the **relationship between two variables** *and* the **univariate distribution** of each variable, all in one figure.

When to Use It

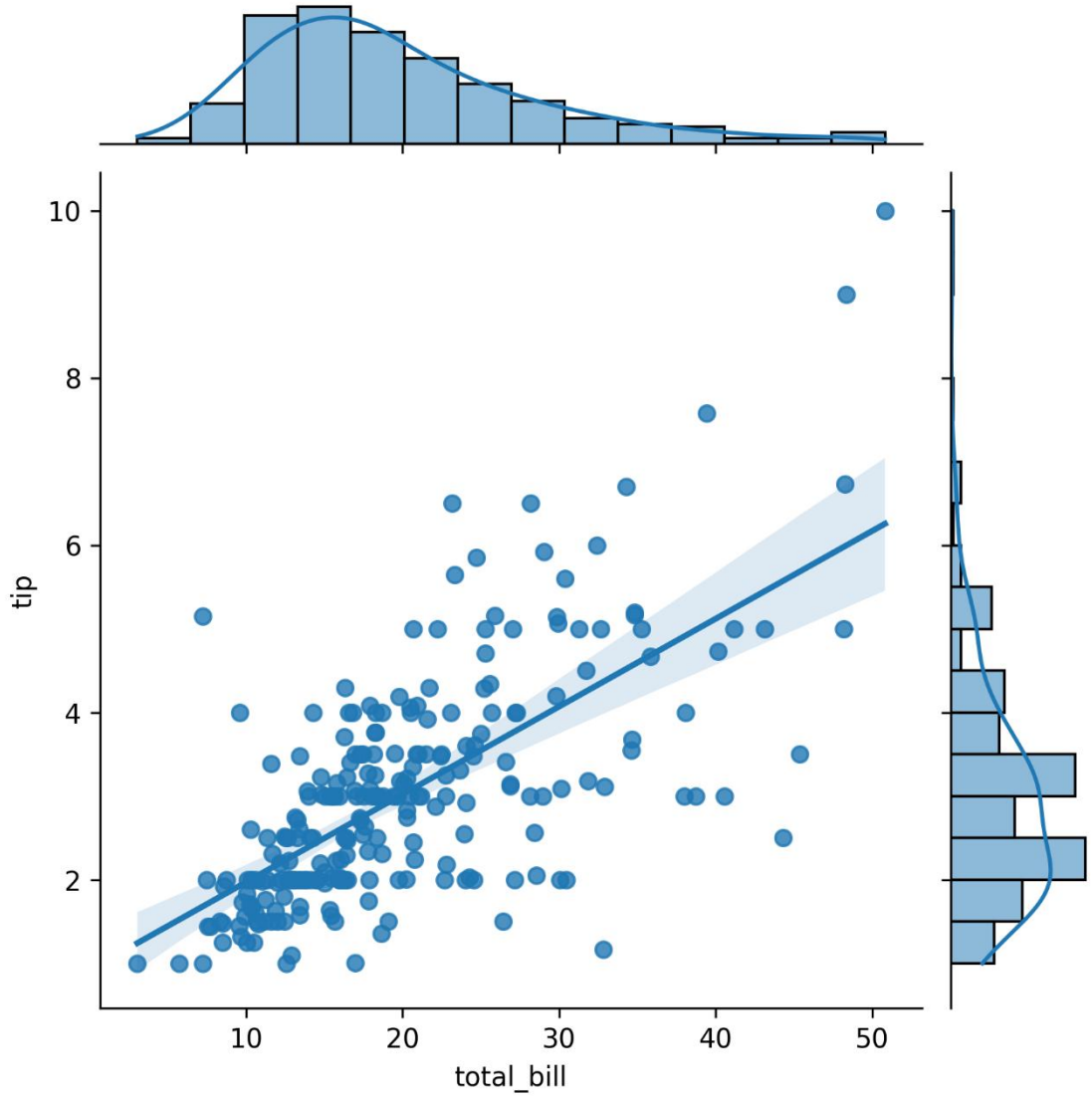
- When you want to study **two variables** in detail.
- When you want to see both:
    - How they relate to each other (scatter/hexbin/regression).
    - How they are distributed individually.
- When pairplot is "too much" (since pairplot compares all variables), but you want a **focused 2-variable analysis**.

Seaborn sns.jointplot()

Parameter	Description	Example
x, y	Column names (variables to plot)	x="total_bill", y="tip"
data	DataFrame source	data=tips
kind	Type of plot in the joint space: "scatter" (default), "kde", "hist", "hex", "reg"	kind="hex"
hue	Grouping variable (colors by category)	hue="Gender"
marginal_ticks	Show ticks on marginal axes (True/False)	marginal_ticks=True
height	Size of the figure (in inches)	height=6
ratio	Ratio of joint plot size to marginal plots	ratio=5
space	Space between joint and marginal axes	space=0.2
dropna	Ignore missing values	dropna=True

Code Example

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

tips = sns.load_dataset("tips")

# Scatter + marginal histograms
sns.jointplot(x="total_bill", y="tip", data=tips, kind="scatter")

# Regression line instead of raw scatter
sns.jointplot(x="total_bill", y="tip", data=tips, kind="reg")

plt.show()
```



LM Plot

What is an LM Plot?

An **LM Plot** is a **regression plot** that combines:

- A **scatter plot** showing data points.
- A **regression line** (linear or logistic) fitted through the data.
- Optionally, **confidence intervals** around the regression line.

It's essentially a specialized scatter plot that automatically adds regression modeling.

When to Use It

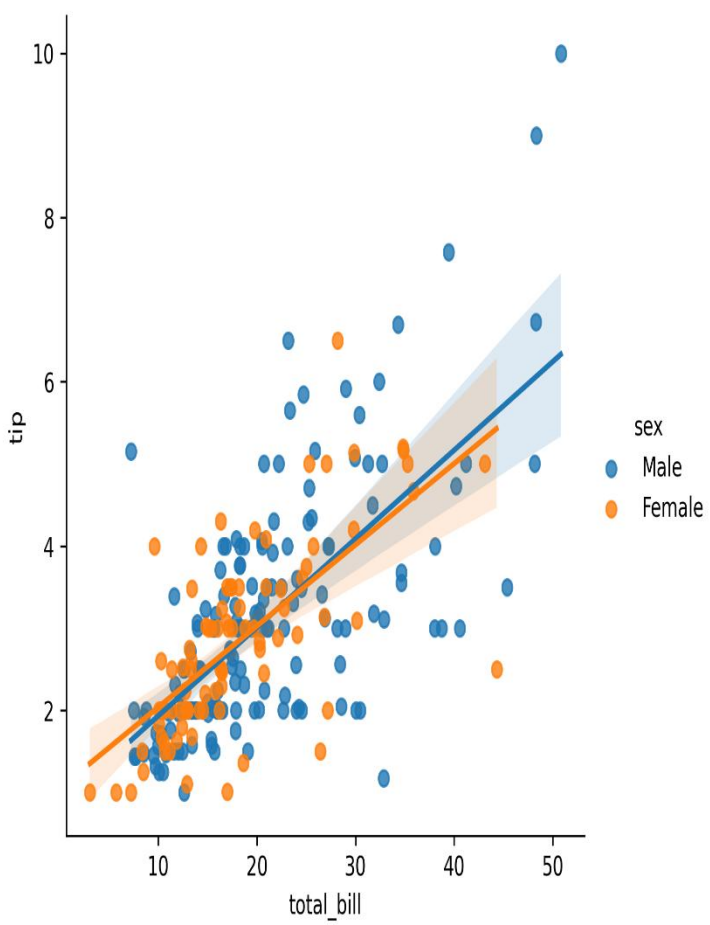
- To analyze the **relationship between two continuous variables**.
- When you want to check if the relationship is **linear**.
- To visualize regression results with confidence intervals.
- To **compare regression lines across categories** using hue, col, or row.

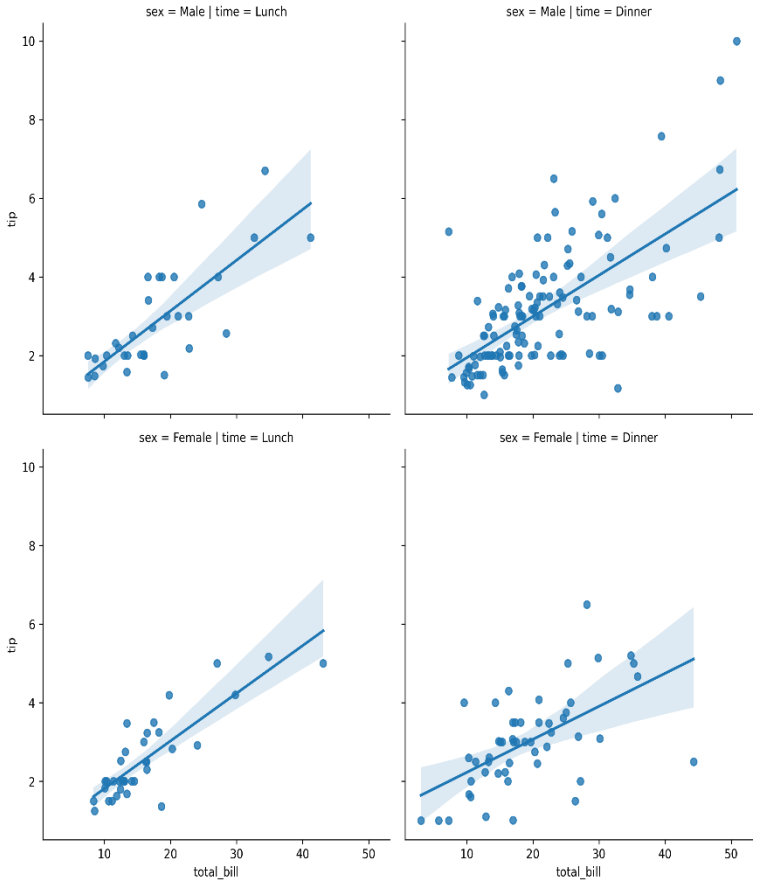
Seaborn sns.lmplot()

Parameter	Description	Example
x, y	Variables to plot	x="total_bill", y="tip"
data	DataFrame	data=tips
hue	Grouping variable (color)	hue="sex"
col, row	Create multiple plots by category	col="time", row="sex"
palette	Color palette	palette="Set1"
height	Plot size (inches)	height=5
aspect	Width/height ratio	aspect=1.2
order	Polynomial order of regression line	order=2 (quadratic)
ci	Size of confidence interval (in %)	ci=95
scatter	Show scatter points (True/False)	scatter=False
fit_reg	Fit regression line (True/False)	fit_reg=False
logistic	Logistic regression instead of linear	logistic=True
robust	Robust regression (less sensitive to outliers)	robust=True

Code Example

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

tips = sns.load_dataset("tips")

# Add hue to compare groups
sns.lmplot(x="total_bill", y="tip", data=tips, hue="sex")

# Multiple columns for comparison
sns.lmplot(x="total_bill", y="tip", data=tips, col="time", row="sex")

plt.show()
```





KDE Plot

What is a KDE Plot?

A **KDE (Kernel Density Estimation) plot** is a **smoothed version of a histogram**.
Instead of showing frequencies in discrete bins, it estimates the **probability density function** of a continuous variable.

Think of it as drawing a **smooth curve** that shows where data is concentrated.

When to Use It

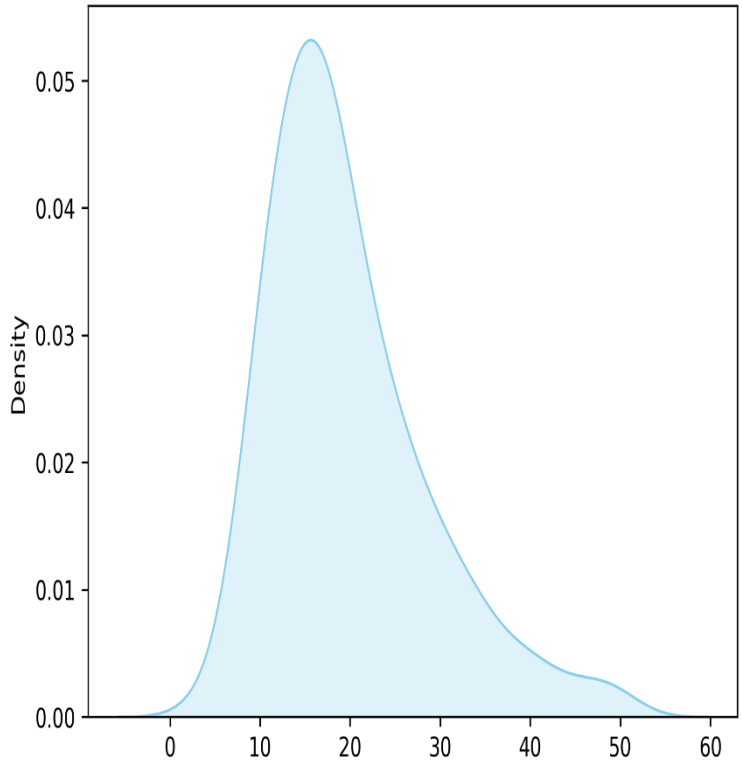
- When you want to see the **distribution of continuous data** more smoothly than a histogram.
- To compare **multiple distributions** across groups.
- To check for **skewness, modality (uni/bi-modal)**, and spread of the data.
- As an alternative (or complement) to histograms and boxplots.

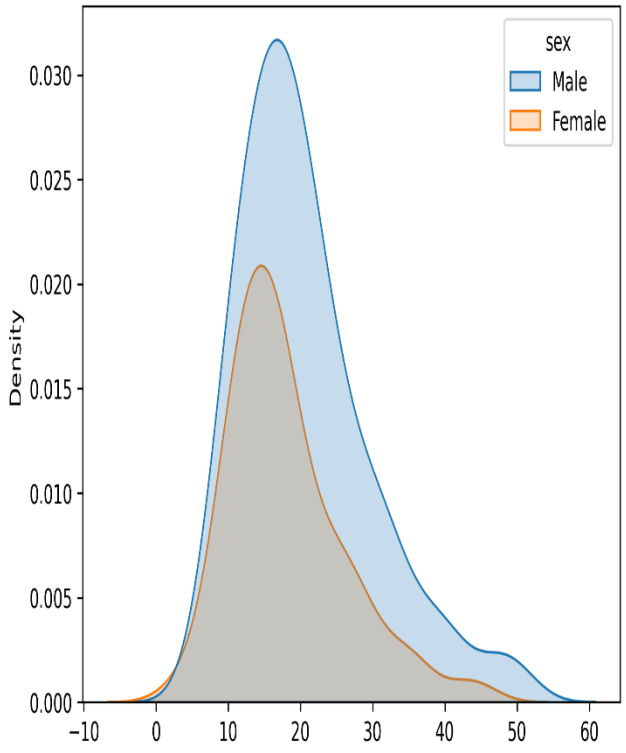
Seaborn sns.kdeplot()

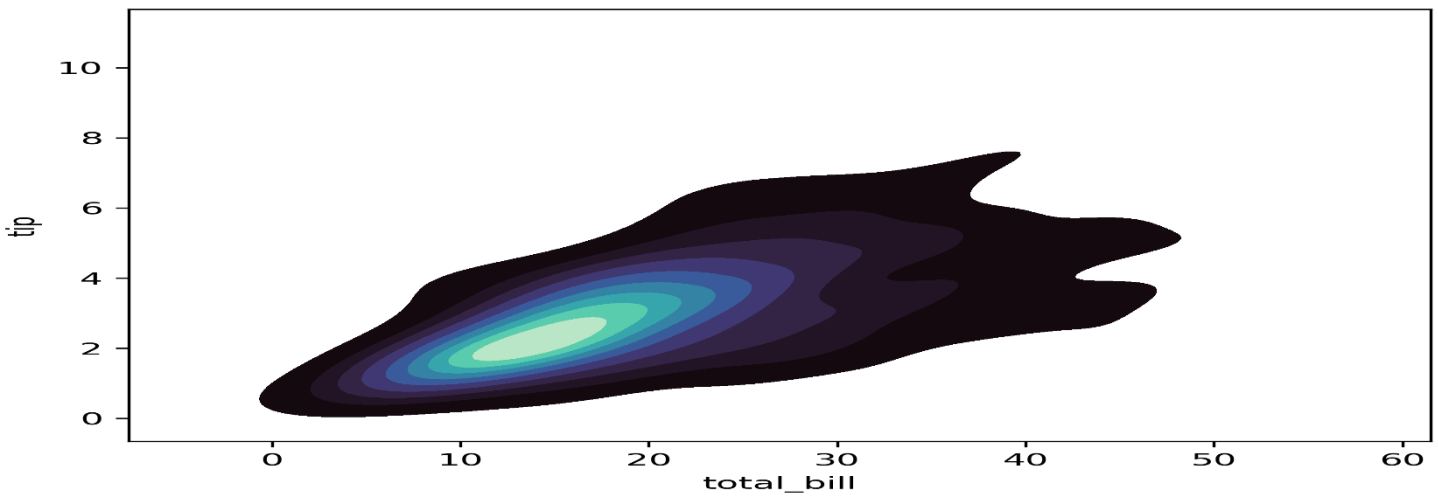
Parameter	Description	Example
x, y	Variables to plot (1D or 2D)	x="total_bill", y="tip"
data	DataFrame	data=tips
hue	Grouping variable (color)	hue="sex"
fill	Fill area under curve	fill=True
bw_adjust	Bandwidth adjustment (smoothness)	bw_adjust=0.5 (less smooth), bw_adjust=2 (more smooth)
common_norm	Normalize densities across groups	common_norm=False
cumulative	Cumulative density	cumulative=True
levels	Number of contour levels (for 2D)	levels=10
thresh	Minimum contour level for plotting	thresh=0.1
cmap	Colormap for 2D KDE	cmap="mako"

Code Example

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

tips = sns.load_dataset("tips")

# KDE with shading
sns.kdeplot(x="total_bill", data=tips, fill=True, color="skyblue")

# Multiple groups
sns.kdeplot(x="total_bill", data=tips, hue="sex", fill=True)

# Bivariate KDE (2D density)
sns.kdeplot(x="total_bill", y="tip", data=tips, fill=True, cmap="mako")

plt.show()
```







Strip Plot

What is a Strip Plot?

A **strip plot** is a **scatter plot for categorical data**.
It places **individual data points along a categorical axis**, often with some jitter to prevent overlap.
It's especially useful when you want to **see raw observations** instead of aggregated summaries (like bar plots).

When to Use It

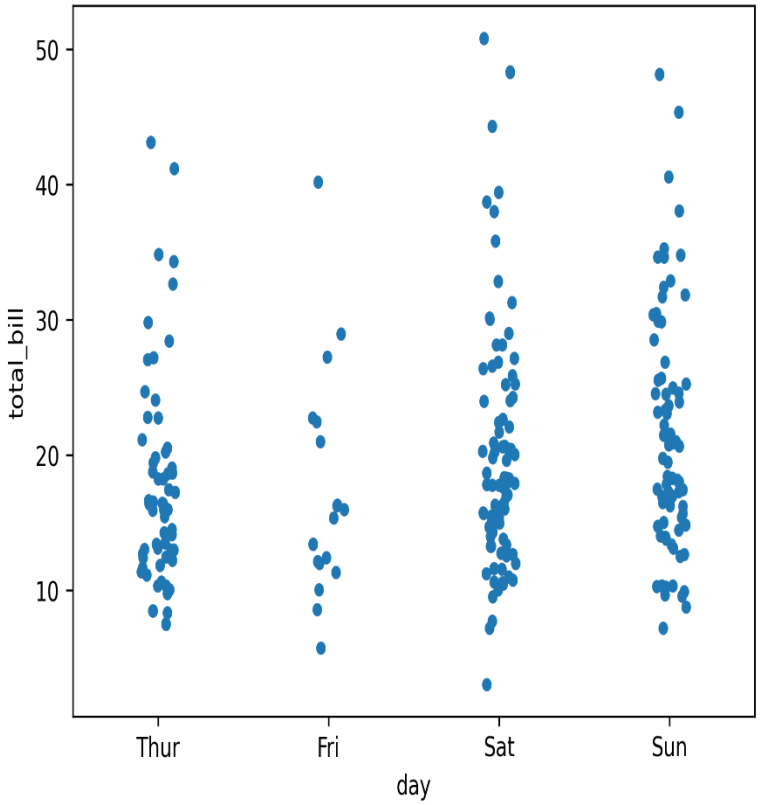
- When you want to **visualize all individual data points** within categories.
- To check for **distribution, spread, and outliers** in categorical data.
- Often used **with boxplots or violin plots** for more detail.

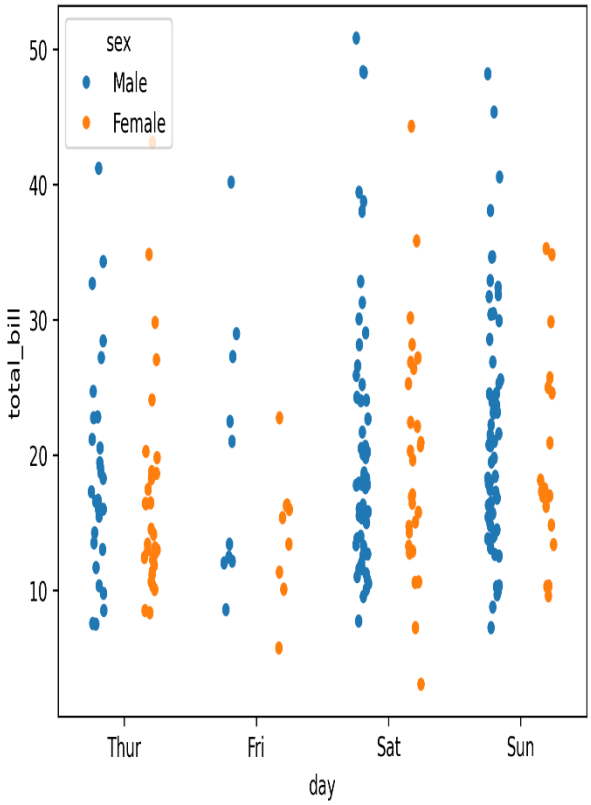
Seaborn sns.stripplot()

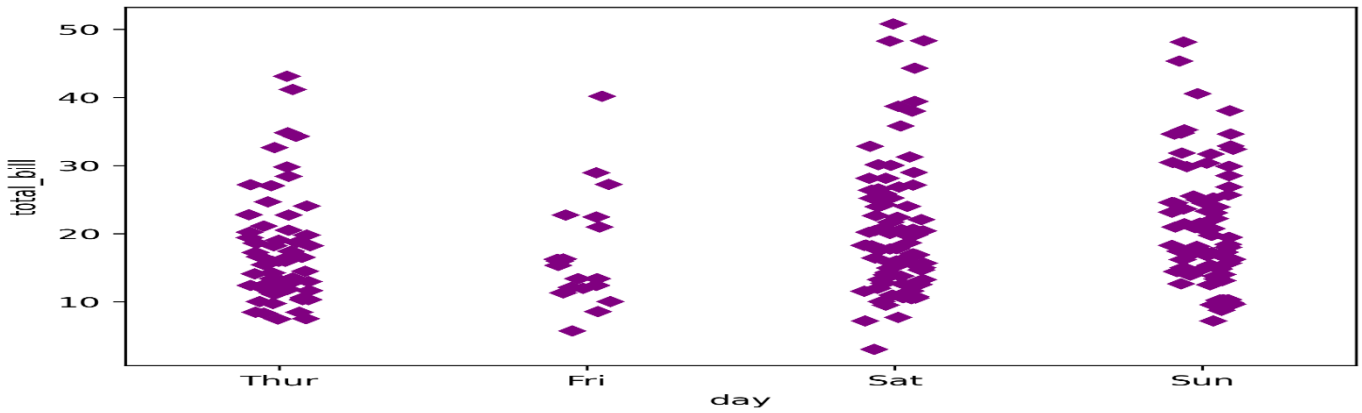
Parameter	Description	Example
x, y	Variables to plot (categorical on one axis)	x="day", y="total_bill"
data	DataFrame	data=tips
hue	Grouping variable	hue="sex"
jitter	Adds random noise to spread points (avoids overlap)	jitter=True
dodge	Separates hue groups side-by-side	dodge=True
size	Marker size	size=6
marker	Marker style ("o", "s", "D", "^", etc.)	marker="D"
alpha	Transparency	alpha=0.6
palette	Color palette	palette="Set2"
orient	Orientation ("v" = vertical, "h" = horizontal)	orient="h"

Code Example

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

tips = sns.load_dataset("tips")

# Add jitter (spread points out horizontally to avoid overlap)
sns.stripplot(x="day", y="total_bill", data=tips, jitter=True)

# Add hue for grouping
sns.stripplot(x="day", y="total_bill", data=tips, hue="sex", jitter=True, dodge=True)

# Change marker style and size
sns.stripplot(x="day", y="total_bill", data=tips, jitter=True, size=6, marker="D", color="purple")

plt.show()
```





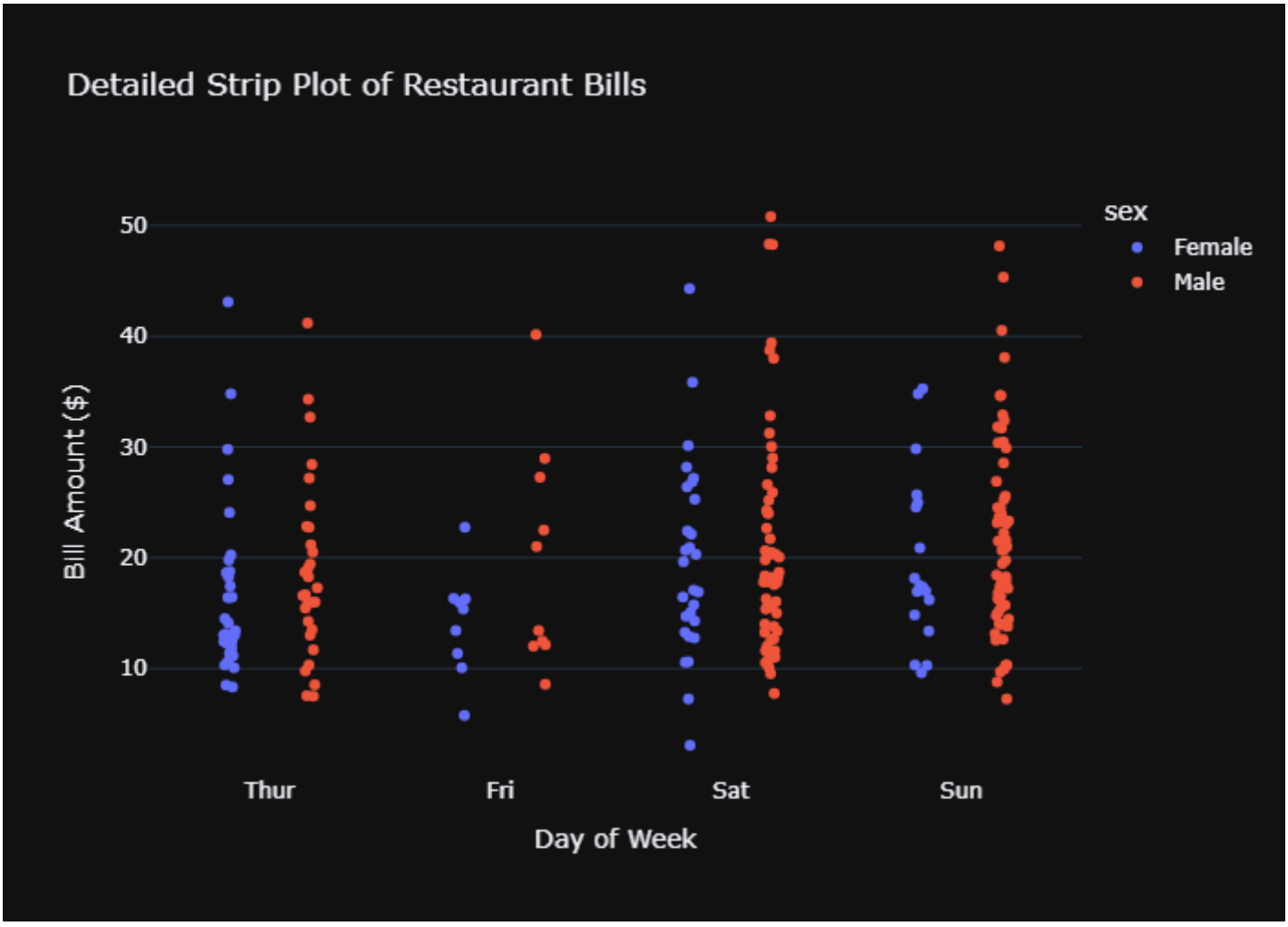


Plotly px.strip()

Parameter	Description	Example
data_frame	Input DataFrame	px.strip(tips, x="day", y="total_bill")
x, y	Variables for axes (categorical usually on x)	x="day", y="total_bill"
color	Coloring by categorical variable	color="sex"
facet_row, facet_col	Create multiple strip plots by categories	facet_col="time"
stripmode	"overlay" (default, points overlap) or "group" (side-by-side by category)	stripmode="group"
hover_data	Extra columns to show on hover	hover_data=["tip", "size"]
orientation	"v" (vertical, default) or "h" (horizontal)	orientation="h"
width, height	Figure size in pixels	width=700, height=500
labels	Rename axis labels / legend titles	labels={"total_bill":"Bill ($)"}
category_orders	Custom category order	category_orders={"day":["Thur","Fri","Sat","Sun"]}
title	Figure title	title="Restaurant Bills Strip Plot"
template	Style/theme ("plotly", "seaborn", "ggplot2", etc.)	template="seaborn"

Code Example

```
import plotly.express as px
tips = px.data.tips()

fig = px.strip(
    tips,
    x="day",
    y="total_bill",
    color="sex",
    stripmode="group",
    hover_data=["tip", "size"],
    orientation="v",
    category_orders={"day": ["Thur", "Fri", "Sat", "Sun"]},
    labels={"total_bill": "Bill Amount ($)", "day": "Day of Week"},
    title="Detailed Strip Plot of Restaurant Bills",
    template="plotly_dark",
    width=800,
    height=500
)
fig.show()
```



Point Plot

What is a Point Plot?

A **point plot** is a categorical plot that shows **mean values (or another estimator)** of a variable across categories, with **confidence intervals (error bars)** by default.
It is basically a **barplot alternative**, but instead of bars, it uses **points with lines** to show trends.

It's especially useful for showing:

- **Trends** across categories.
- **Comparisons** between groups (via hue).
- **Error ranges** (via confidence intervals).

When to Use It

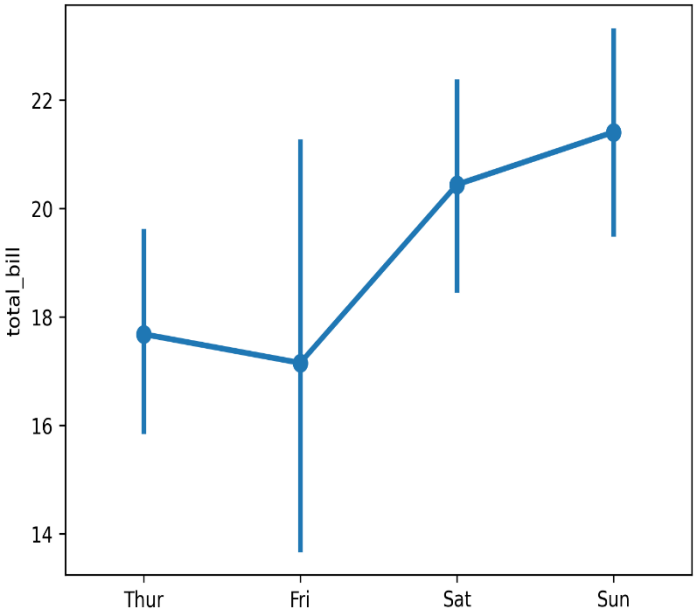
- When you want to emphasize **trends and comparisons**, rather than actual magnitudes (barplot is better for absolute size).
- When showing **statistical summaries** (mean ± confidence interval).
- When comparing **two categorical variables** and their interaction with a numeric variable.

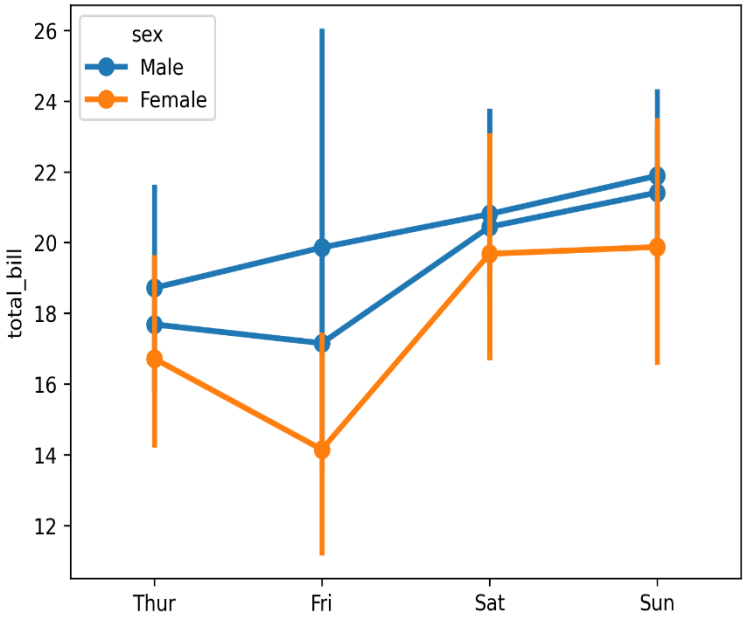
Seaborn sns.pointplot()

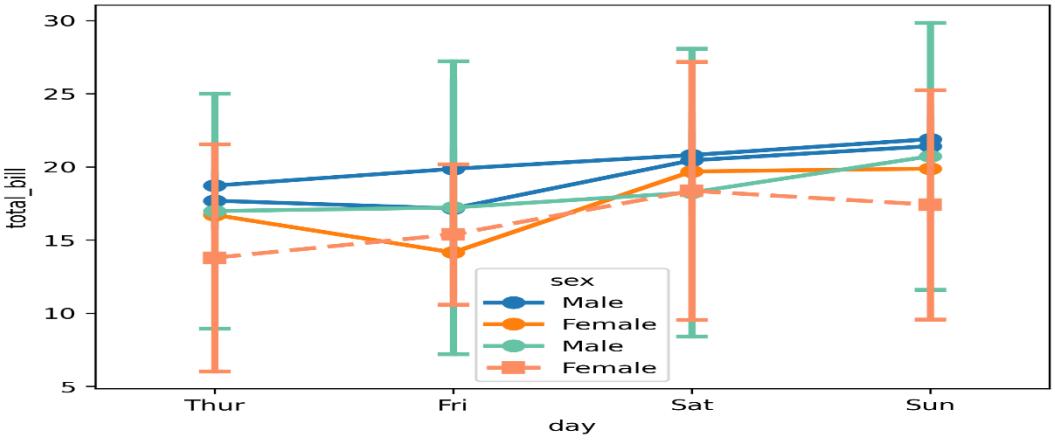
Parameter	Description	Example
x, y	Variables to plot	x="day", y="total_bill"
data	DataFrame	data=tips
hue	Subgrouping variable (different colors)	hue="sex"
order, hue_order	Custom category order	order=["Thur","Fri","Sat","Sun"]
estimator	Function for central tendency (default mean)	estimator=np.median
dodge	Separate hue groups side-by-side	dodge=True
markers	Marker style	markers=["o", "s"]
linestyles	Line style	linestyles=["-", "--"]
capsize	Size of CI bar caps	capsize=0.1
palette	Color palette	palette="Set2"

Code Example

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

tips = sns.load_dataset("tips")

# Basic pointplot
sns.pointplot(x="day", y="total_bill", data=tips)

# With hue and confidence intervals
sns.pointplot(x="day", y="total_bill", hue="sex", data=tips)

# Custom estimator, order, and styles
sns.pointplot(
    x="day", y="total_bill", hue="sex", data=tips,
    estimator=np.median,
    markers=["o", "s"], linestyles=["-", "--"], palette="Set2", capsize=0.1
)

plt.show()
```







Treemap

What is a Treemap?

A **treemap** is a visualization for **hierarchical data** using **nested rectangles**.

- Each **rectangle's size** represents a numeric value.
- Rectangles can be **grouped into categories (parents/children)**.
- Color can represent another variable (like category or value).

When to Use It

Best for:

- Showing **part-to-whole relationships** (like a pie chart, but better with many categories).
- Visualizing **hierarchical breakdowns** (e.g., continent → country → population).
- Comparing sizes of subcategories efficiently.

Avoid when:

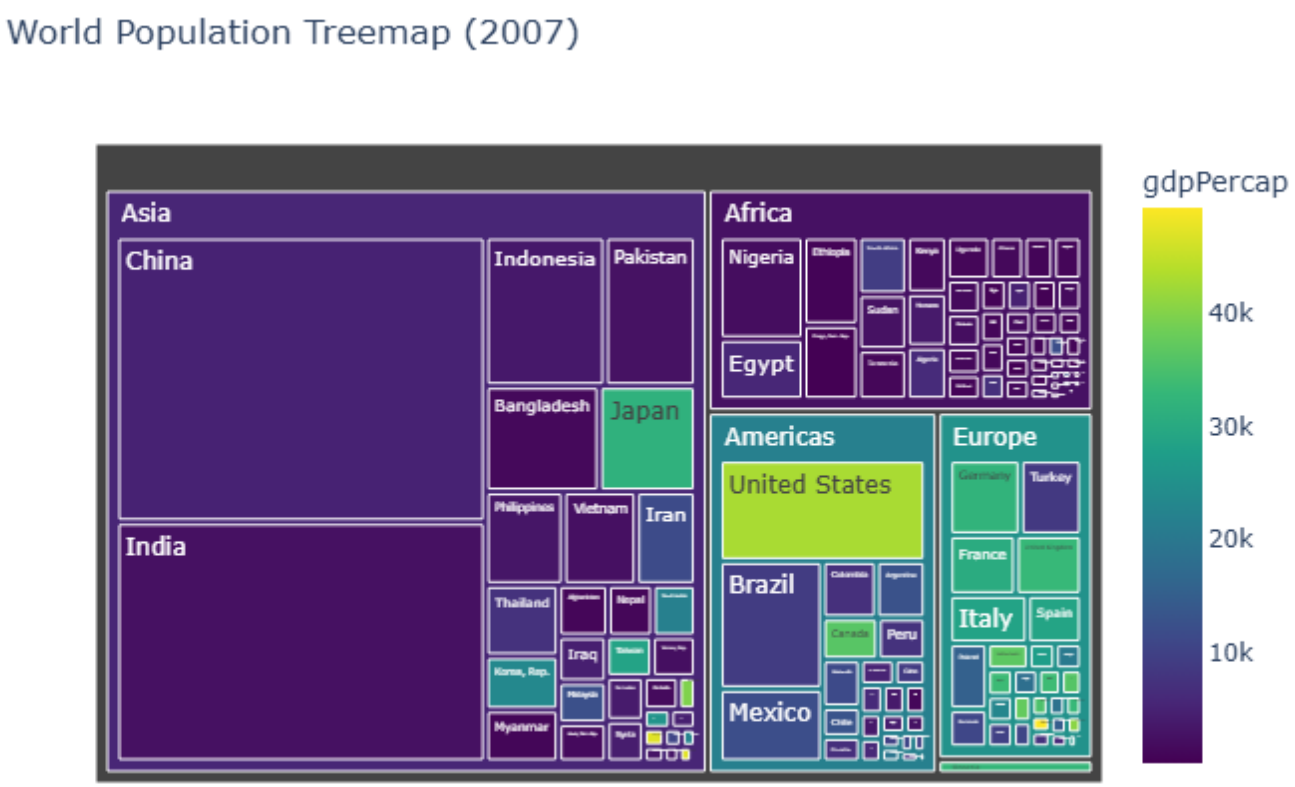
- Exact comparisons are needed (bar charts are better).
- Hierarchy depth is not important.

Plotly px.treemap()

Parameter	Description	Example
data_frame	Input DataFrame	px.treemap(df, path=["continent","country"], values="pop")
path	Defines hierarchy (list of categorical columns)	["continent", "country"]
values	Numeric column that defines rectangle size	"population"
color	Column for coloring rectangles	color="gdpPercap"
hover_data	Extra columns to show on hover	hover_data=["lifeExp"]
color_continuous_scale	Continuous color scale ("Viridis", "RdBu")	color_continuous_scale="Viridis"
color_discrete_sequence	Discrete color palette	color_discrete_sequence=px.colors.qualitative.Set3
maxdepth	Limit levels of hierarchy shown	maxdepth=2
branchvalues	"total" (default: parent = sum of children) or "remainder"	branchvalues="total"
title	Plot title	title="World Population Treemap"
width, height	Figure size	width=800, height=600

Code Example

```
import plotly.express as px

# Built-in dataset
df = px.data.gapminder().query("year == 2007")

fig = px.treemap(
    df,
    path=["continent", "country"],  # hierarchy
    values="pop",                # size
    color="gdpPercap",           # coloring
    hover_data=["lifeExp"],
    color_continuous_scale="Viridis",
    title="World Population Treemap (2007)"
)
fig.show()
```



Sunburst

What is a Sunburst?

A **sunburst chart** shows hierarchical data as **concentric rings**:

- The **center** = root node.
- Each **ring** = a level of the hierarchy.
- **Arc length** = proportional to a value (like population, sales, etc.).
- **Color** = another variable (e.g., category, GDP per capita).

It's essentially a **circular treemap**.

When to Use It

Best for:

- Exploring hierarchical data (like continent → country → city).
- Showing **proportion & breakdowns**.
- Making hierarchy depth clear visually.

Avoid when:

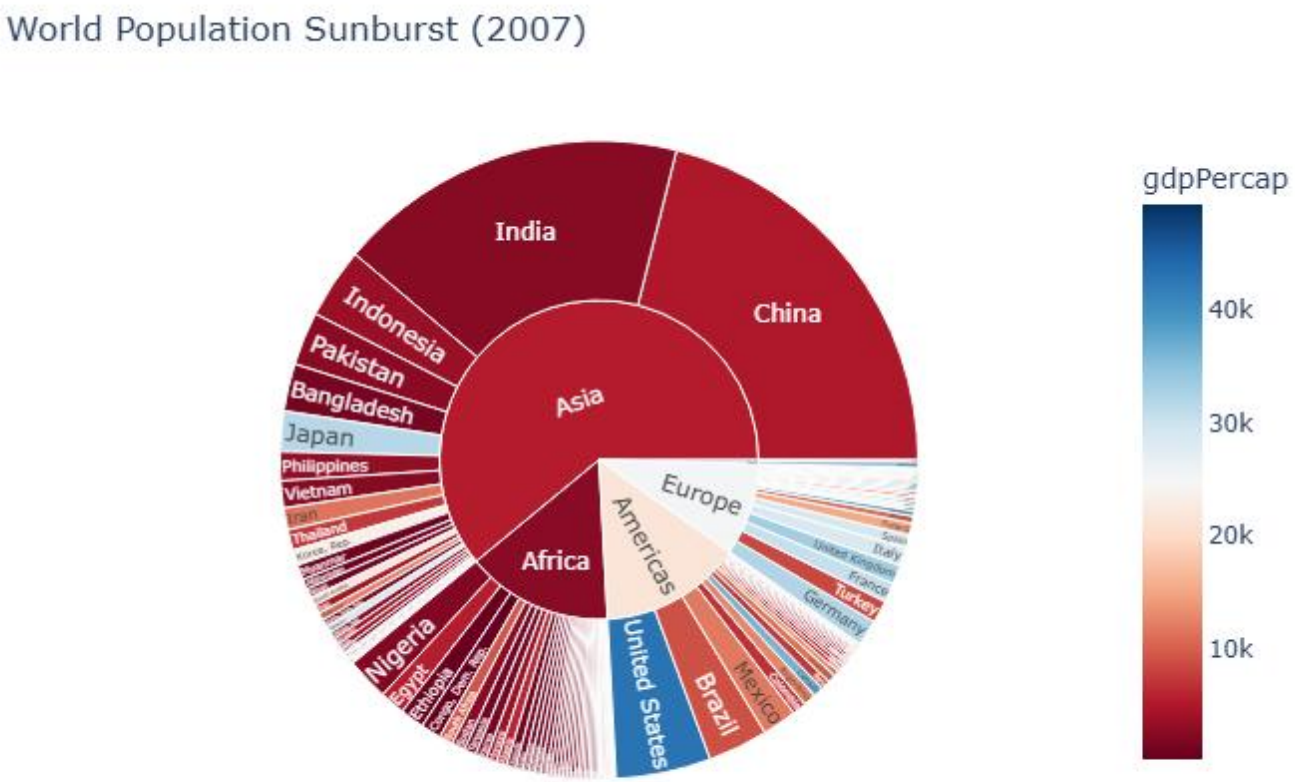
- You need **precise comparisons** (use bar/treemap instead).
- Many categories make it cluttered.

Plotly px.sunburst()

Parameter	Description	Example
data_frame	Input DataFrame	df = px.data.gapminder()
path	Defines hierarchy (list of columns)	["continent", "country"]
values	Numeric column for size of slices	"pop"
color	Column for coloring slices	color="gdpPercap"
hover_data	Extra info on hover	hover_data=["lifeExp"]
color_continuous_scale	Continuous color scale	"Viridis", "Cividis", "RdBu"
color_discrete_sequence	Discrete palette	px.colors.qualitative.Set3
maxdepth	Limit depth of hierarchy shown	maxdepth=2
branchvalues	"total" (parent = sum of children) or "remainder"	branchvalues="total"
title	Chart title	title="World Population Sunburst"
width, height	Figure size	width=800, height=600

Code Example

```
import plotly.express as px

# Load data
df = px.data.gapminder().query("year == 2007")

# Create sunburst
fig = px.sunburst(
    df,
    path=["continent", "country"],  # hierarchy
    values="pop",                # size of slices
    color="gdpPercap",           # coloring
    hover_data=["lifeExp"],
    color_continuous_scale="RdBu",
    title="World Population Sunburst (2007)"
)
fig.show()
```



Area Chart

What is an Area Chart?

An **area chart** is like a **line plot with the area below the line filled**.

- It shows **trends over time** or continuous variables.
- Useful to visualize **cumulative magnitude** or emphasize the "volume" of change.
- Can be **stacked** to show contributions of categories.

When to Use It

Best for:

- Showing how values **change over time**.
- Emphasizing the **magnitude under a curve**.
- Comparing contributions of multiple series (stacked area).

Avoid when:

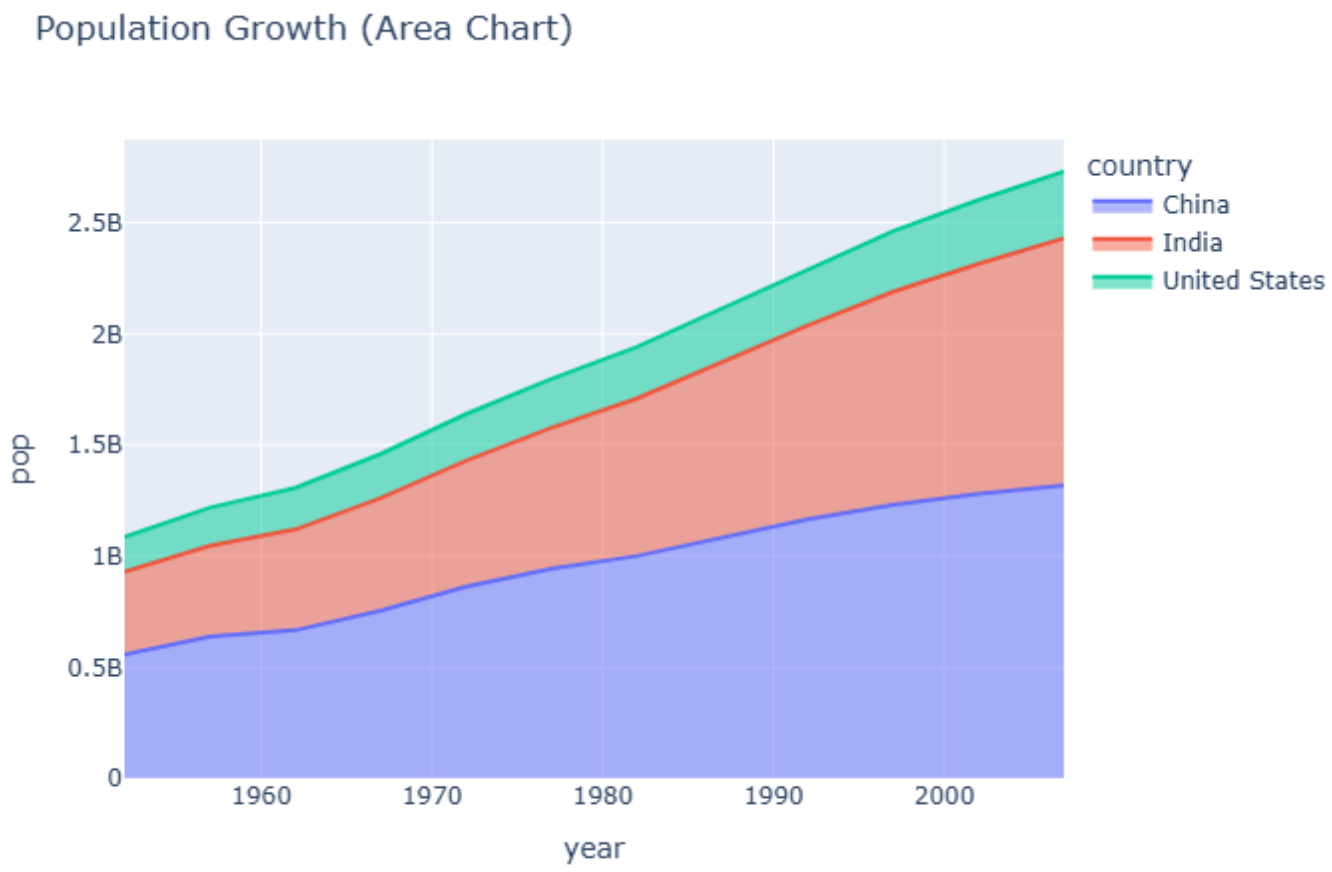
- You need precise value comparisons → use line/bar chart.
- Too many categories → becomes messy.

Plotly px.area()

Parameter	Description	Example
data_frame	Input DataFrame	px.data.gapminder()
x	X-axis column	"year"
y	Y-axis column	"pop"
color	Category grouping	"continent"
line_group	Groups for multiple lines	"country"
title	Chart title	"Population Growth"
labels	Axis labels	labels={"pop": "Population"}
groupnorm	"fraction" or "percent" for normalization	groupnorm="percent"
stackgroup	Stack series	"one"

Code Example

```
import plotly.express as px

df = px.data.gapminder().query("country in ['India','China','United States']")
fig = px.area(
    df,
    x="year", y="pop",
    color="country",
    line_group="country",
    title="Population Growth (Area Chart)"
)
fig.show()
```



Geospatial & Temporal Visualization

Choropleth (Region-based)

- Shaded regions → great for aggregated data at *country/state/district* level.
- Example: COVID-19 cases by state.

Scatter Geo (Point-based)

- Location-specific markers → great for *city, event, coordinate* level data.
- Example: earthquake epicenters, store locations.

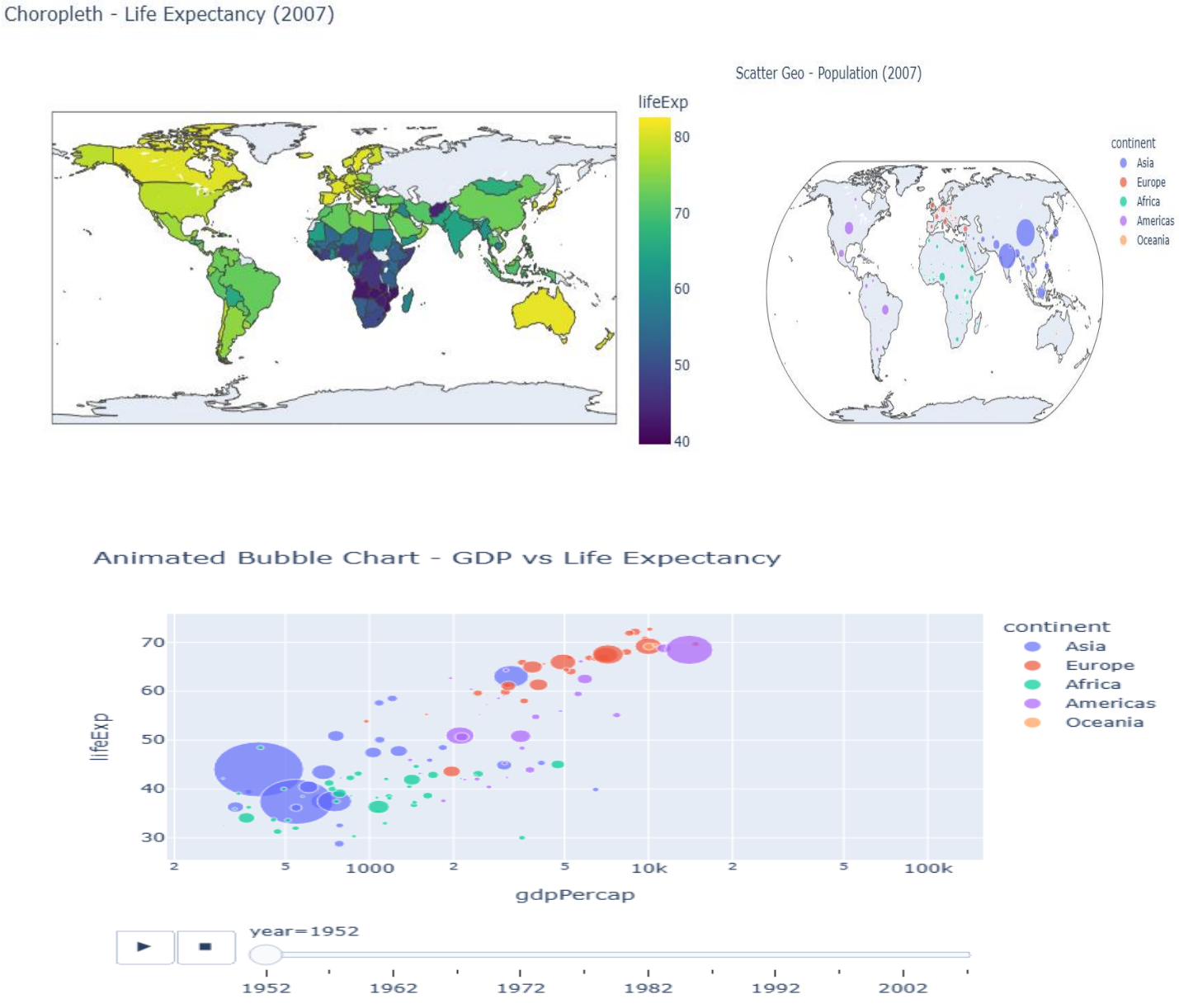
Animation Frame (Time-based)

- Adds a **temporal layer** to both choropleths and scatter maps.
- Example: population growth across years, CO_2 emissions trend.

Code Example

```
import plotly.express as px

df = px.data.gapminder()

# --- Choropleth ---
fig1 = px.choropleth(
   df.query("year==2007"),
   locations="iso_alpha",
   color="lifeExp",
   hover_name="country",
   color_continuous_scale="Viridis",
   title="Choropleth - Life Expectancy (2007)"
)
fig1.show()

# --- Scatter Geo ---
fig2 = px.scatter_geo(
   df.query("year==2007"),
   locations="iso_alpha",
   size="pop",
   color="continent",
```

```
    hover_name="country",
    projection="natural earth",
    title="Scatter Geo - Population (2007)"
)
fig2.show()

# --- Animated Scatter (spatio-temporal) ---
fig3 = px.scatter(
    df,
    x="gdpPercap", y="lifeExp",
    size="pop", color="continent",
    hover_name="country", hover_data={"year": True, "pop": ":,", "continent": False}
    log_x=True, size_max=60,
    animation_frame="year", animation_group="country",
    title="Animated Bubble Chart - GDP vs Life Expectancy"
)
fig3.write_html("fig3.html") # Save as HTML To be Dynamic
```



Conclusion

This document has provided a structured and practical overview of **data visualization in Python** using **Matplotlib**, **Seaborn**, and **Plotly**. From simple static charts to advanced interactive visualizations, we explored the breadth of options available to transform raw data into meaningful stories. By highlighting parameters, use cases, and visual examples, this guide equips learners and practitioners to confidently select the right visualization techniques for their analyses.

Ultimately, the ability to **visualize data effectively** is not just a technical skill but a powerful communication tool for decision-making, research, and innovation.

Feedback & Contribution

This is a living document. If you have feedback, suggestions, or additional insights that could improve it, I welcome your input. Sharing ideas helps keep this material accurate, relevant, and valuable for everyone.

Copyright & Usage

Acknowledgments

This document is authored and copyrighted by **[Youssef Amgad Elkhatib]**.
Special thanks to the contributors who provided valuable feedback and suggestions for improvements:

Name	Contribution