Python Plotly

Panchatcharam M
Associate Professor
Interdisciplinary Dual Degree in Data Science Committee
IIT Tirupati

- A Python library for interactive, publication-quality graphs
- Open source – built on top of plotly.js (JavaScript)
- Over 40 chart types – statistical, scientific, 3D, maps, finance
- Figures render as interactive HTML – hover, zoom, pan, select
- Works in Jupyter, scripts, and web apps

Plotly – Why

- Matplotlib images are static – Plotly figures respond to the mouse
- Hover reveals exact values – no more guessing from the axis
- Zoom into dense regions of large datasets
- One-line charts with Plotly Express
- No data files needed today – we type small data, or use built-ins

Install and Import

- Convention: px for quick charts, go for fine control

```
pip install plotly

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

Our Lab Data – Typed In, No Files

```
df = pd.DataFrame({
 "Name":   ["Aditya","Ajay","Amal","Badal","Chitra","Divya"],
 "Hostel": ["Ganga","Ganga","Yamuna","Yamuna","Ganga","Yamuna"],
 "Maths":  [85, 75, 95, 45, 88, 72],
 "Python": [65, 35, 85, 65, 92, 58],
 "Stats":  [82, 72, 52, 55, 79, 91],
})

months = ["Jan","Feb","Mar","Apr","May","Jun",
          "Jul","Aug","Sep","Oct","Nov","Dec"]
rain = [12,8,15,22,60,110,190,175,140,95,45,18]
```

Name	Hostel	Maths	Python	Stats
Aditya	Ganga	85	65	82
Ajay	Ganga	75	35	72
Amal	Yamuna	95	85	52
Badal	Yamuna	45	65	55
Chitra	Ganga	88	92	79
Divya	Yamuna	72	58	91

Type this once at the top of your notebook – every chart today uses df, months, rain

Built-in Datasets – Zero Typing

```
iris = px.data.iris()       # 150 flowers
tips = px.data.tips()       # 244 restaurant bills
gap  = px.data.gapminder() # countries by year

print(iris.head())
```

sepal_length	sepal_width	petal_length	petal_width	species	species_id
5.1	3.5	1.4	0.2	setosa	1
4.9	3	1.4	0.2	setosa	1
4.7	3.2	1.3	0.2	setosa	1
4.6	3.1	1.5	0.2	setosa	1
5	3.6	1.4	0.2	setosa	1

These ship with Plotly – installed means available, no download, no CSV

px.data – Syntax

Syntax:

```
px.data.iris()       # -> pandas DataFrame
px.data.tips()       # -> pandas DataFrame
```

- **iris:** 150 rows × 6 columns: sepal_length, sepal_width, petal_length, petal_width, species, species_id
- **tips:** 244 rows × 7 columns: total_bill, tip, sex, smoker, day, time, size
- **no arguments:** each function simply returns the ready-made table
- **regular DataFrames:** head(), describe(), groupby() – everything from the pandas session works

px.data.gapminder() – Syntax

Syntax:

```
px.data.gapminder(year=None, pretty_names=False)
```

- **returns:** 1704 rows = 142 countries × 12 years (1952–2007, every 5 years)
- **columns:** country, continent, year, lifeExp, pop, gdpPercap, iso_alpha, iso_num
- **year:** pass year=2007 to get a single year directly
- **filtering:** or take everything and filter: gap.query("year == 2007")
- **iso_alpha:** 3-letter country codes – exactly what px.choropleth locations expects

First Figure – Scatter

- Two lines: figure, show

```
fig = px.scatter(iris, x="sepal_width",
                 y="sepal_length")
fig.show()
```

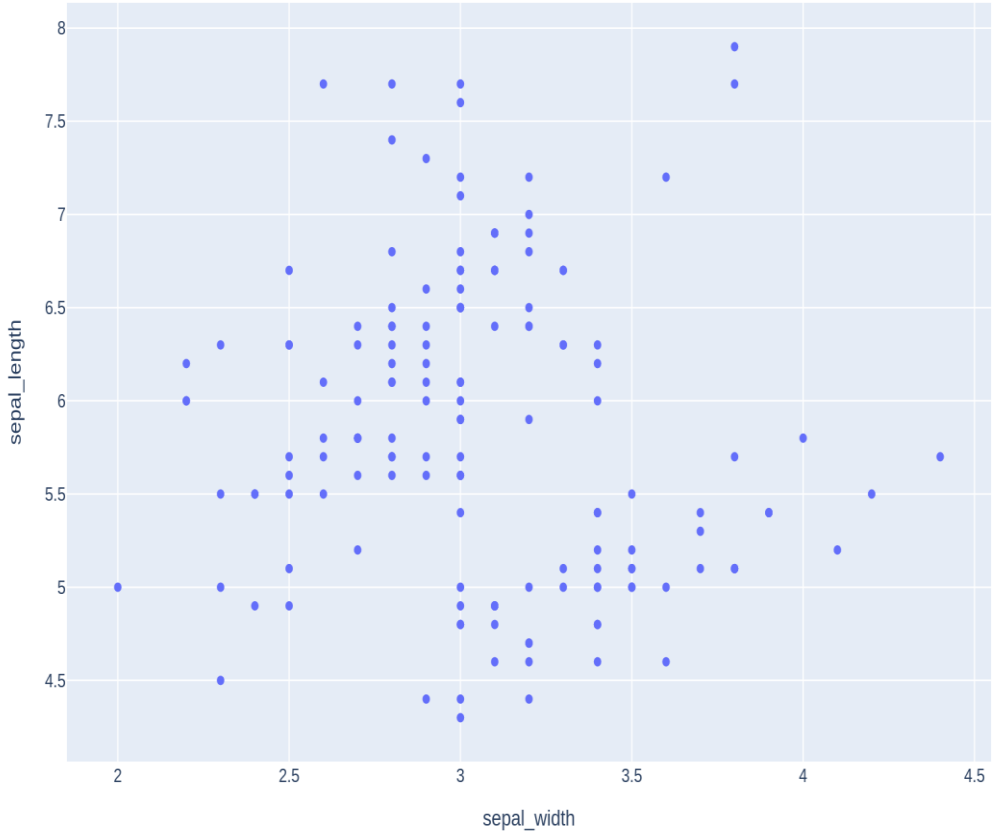


fig.show() opens the interactive figure in the notebook or browser

Scatter – Color, Size, Text

```
fig = px.scatter(df, x="Maths", y="Python",
                 color="Hostel", size="Stats",
                 hover_name="Name", text="Name")
fig.update_traces(textposition="top center")
fig.show()
```

One column per visual channel: color, size, text – the grammar-of-graphics idea

px.scatter – Syntax

Syntax:

```
px.scatter(data_frame=None, x=None, y=None, color=None,
           size=None, text=None, hover_name=None, hover_data=None)
```

- **data_frame:** the DataFrame; or omit it and pass lists/arrays straight to x and y
- **x, y:** column names (strings) – the horizontal and vertical positions
- **color:** a column; categories get a legend, numbers get a colour scale
- **size:** a numeric column – mapped to marker area (not radius)
- **text:** a column printed beside every marker
- **hover_name:** the bold title line of the tooltip
- **hover_data:** extra tooltip columns; a dict can format them, e.g. {"pop": ":,"}

px.scatter – More Arguments

Syntax:

```
px.scatter(..., log_x=False, facet_col=None, labels=None,
           symbol=None, trendline=None, title=None)
```

- **log_x, log_y:** True gives a logarithmic axis – essential for incomes, populations
- **facet_col, facet_row:** a column – one panel per category (small multiples)
- **labels:** a dict renaming anything on screen: {"gdpPercap": "GDP per capita"}
- **symbol:** a column – marker shape per category (works in black and white)
- **trendline:** "ols" fits one regression line per colour group
- **title:** the figure heading – every chart you show should have one

update_traces and update_layout – Syntax

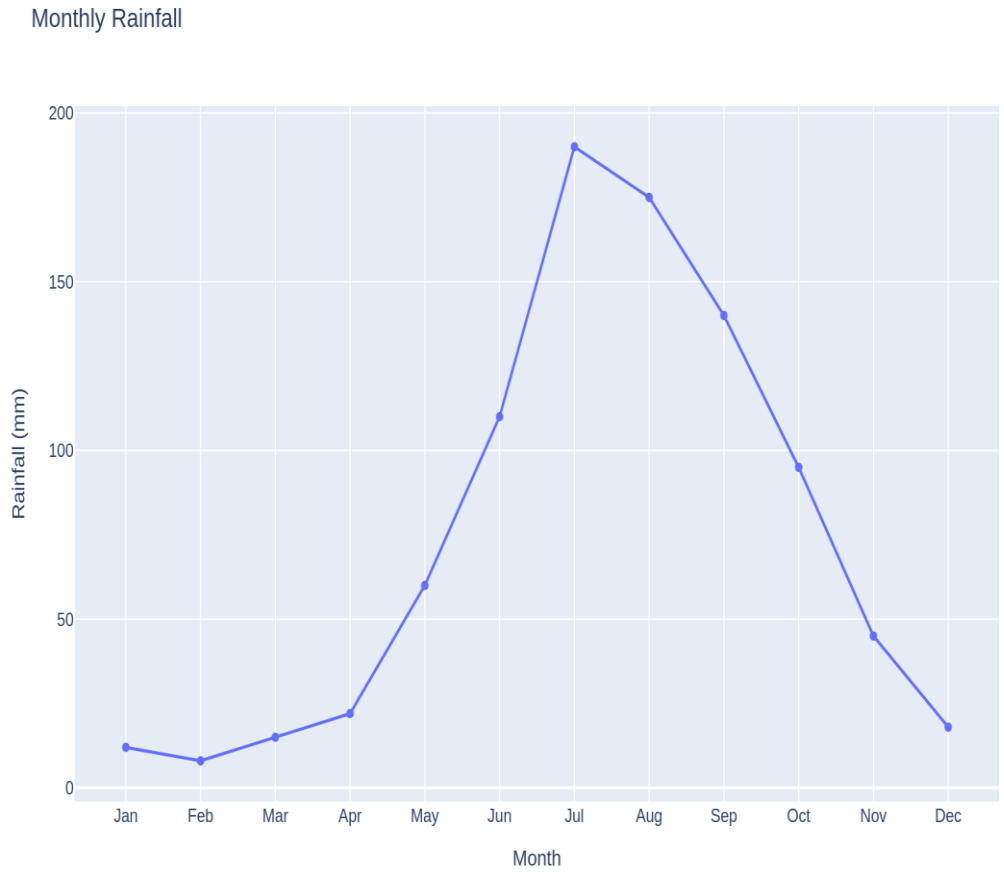
Syntax:

```
fig.update_traces(textposition=None, marker=None, mode=None, ...)
fig.update_layout(title=None, xaxis_title=None, template=None, ...)
```

- **update_traces:** changes properties of the drawn data – applied to every trace at once
- **textposition:** "top center", "bottom right", … where the text sits relative to the marker
- **marker:** a dict of marker properties: dict(size=14, line=dict(width=1))
- **update_layout:** changes everything outside the data: titles, axes, legend, theme
- **template:** the whole look in one word: "plotly_white", "plotly_dark", "seaborn"
- **chaining:** both return the figure – calls can be written one after another

Line Chart

```
fig = px.line(x=months, y=rain, markers=True,
              labels={"x": "Month",
                      "y": "Rainfall (mm)"},
              title="Monthly Rainfall")
fig.show()
```



Plain lists work directly – a DataFrame is not compulsory

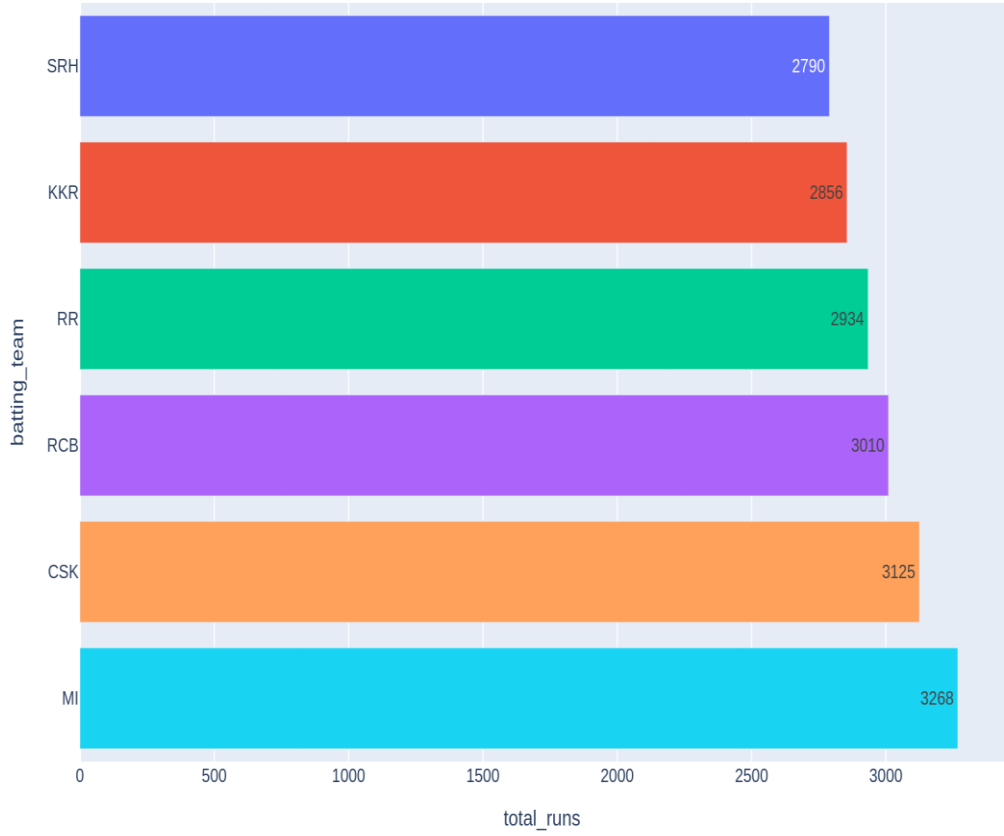
Syntax:

```
px.line(data_frame=None, x=None, y=None, color=None,
        markers=False, line_dash=None, labels=None, title=None)
```

- **x, y:** columns or plain lists – time (or any ordered variable) on x
- **markers:** True draws a point at every observation on top of the line
- **color:** a column – one line per category, with a legend
- **line_dash:** a column – dash pattern per category (readable when printed)
- **labels:** rename axes without touching the data: {"x": "Month"}
- **order matters:** px.line connects points in row order – sort by x first

Bar Chart

```
teams = ["CSK","MI","RCB","KKR","SRH","RR"]
runs  = [3125, 3268, 3010, 2856, 2790, 2934]

fig = px.bar(x=runs, y=teams, orientation="h",
             color=teams, text_auto=True)
fig.update_layout(showlegend=False)
fig.show()
```



orientation="h" – horizontal bars make long names readable

px.bar – Syntax

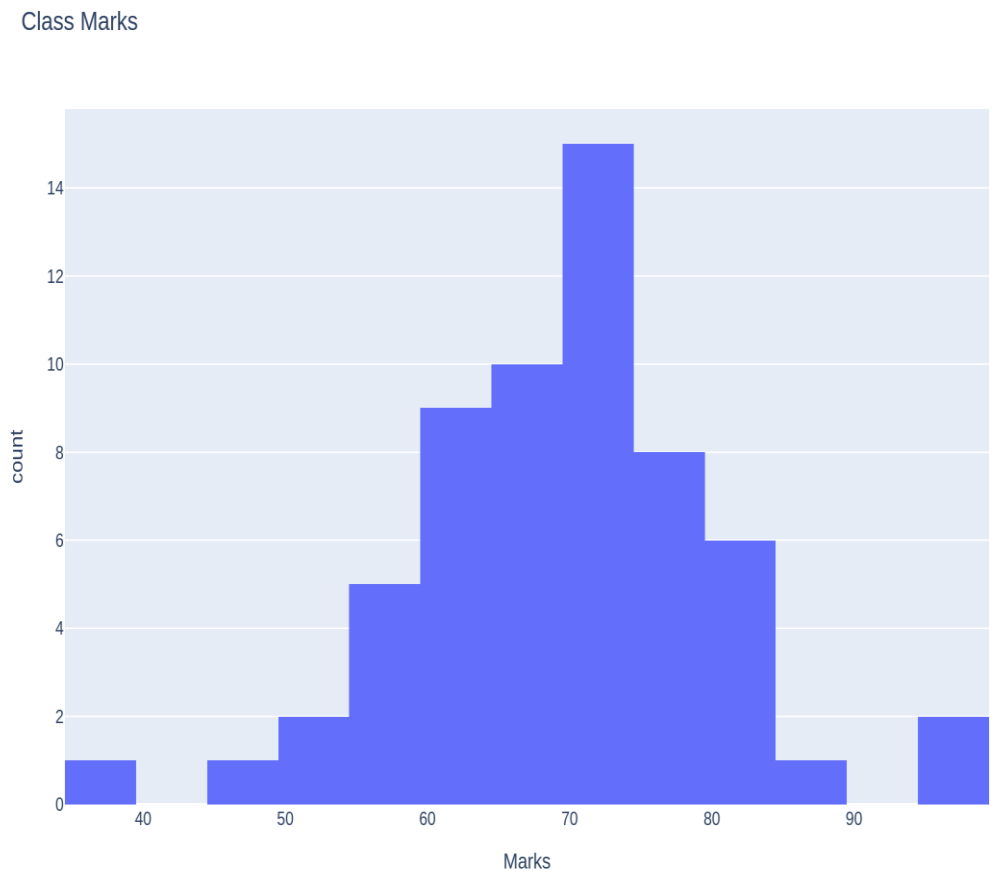
Syntax:

```
px.bar(data_frame=None, x=None, y=None, color=None,
       orientation="v", barmode="relative", text_auto=False)
```

- **x, y:** categories on one axis, values on the other
- **orientation:** "v" vertical (default) or "h" horizontal – for "h", put values on x
- **color:** a column – same category: stacked segments; different: coloured bars
- **barmode:** "relative" stacks, "group" places side by side, "overlay" overlaps
- **text_auto:** True prints the value on each bar; a format string like ".2s" rounds it
- **bars start at zero:** always – a truncated bar axis is a lie (L10!)

Histogram

```
rng = np.random.default_rng(1)
marks = rng.normal(70, 12, 60).round(0)

fig = px.histogram(x=marks, nbins=12,
                   labels={"x": "Marks"},
                   title="Class Marks")
fig.show()
```



One line of simulated data – change nbins and watch the story change

default_rng and rng.normal – Syntax

Syntax:

```
rng = np.random.default_rng(seed=None)
rng.normal(loc=0.0, scale=1.0, size=None)
```

- **default_rng(seed):** creates a random Generator; a fixed seed makes runs reproducible
- **loc:** the mean of the normal distribution – 70 for our marks
- **scale:** the standard deviation – 12 marks of spread
- **size:** how many numbers – 60; a tuple like (8, 6) gives a matrix
- **siblings:** rng.integers(1, 7, 10) – dice; rng.uniform(); rng.choice(list)
- **why seed?:** same seed ⇒ same data ⇒ your neighbour's chart matches yours

px.histogram – Syntax

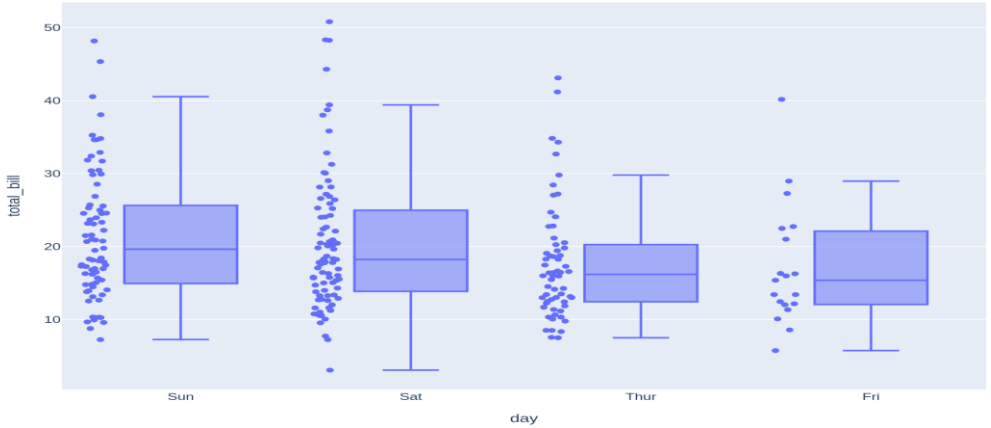
Syntax:

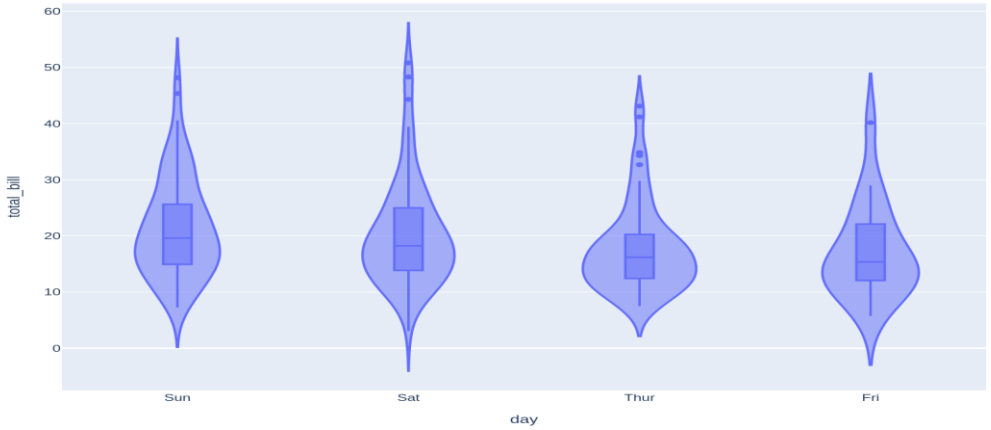
```
px.histogram(data_frame=None, x=None, nbins=None, color=None,
             marginal=None, barmode="relative", histnorm=None)
```

- **x:** the numeric column or list to bin; use y instead for horizontal bars
- **nbins:** a suggestion for the bin count – Plotly picks tidy edges near it
- **color:** a column – one distribution per group
- **marginal:** "box", "violin", or "rug" adds a summary panel on top
- **barmode:** "overlay" (with opacity) is best for comparing two groups
- **histnorm:** "percent" or "probability density" instead of raw counts

Box and Violin

```
tips = px.data.tips()

fig = px.box(tips, x="day", y="total_bill",
             points="all")
fig.show()

fig = px.violin(tips, x="day", y="total_bill",
                box=True)
fig.show()
```





points="all" overlays every observation – honest for small samples

px.box and px.violin – Syntax

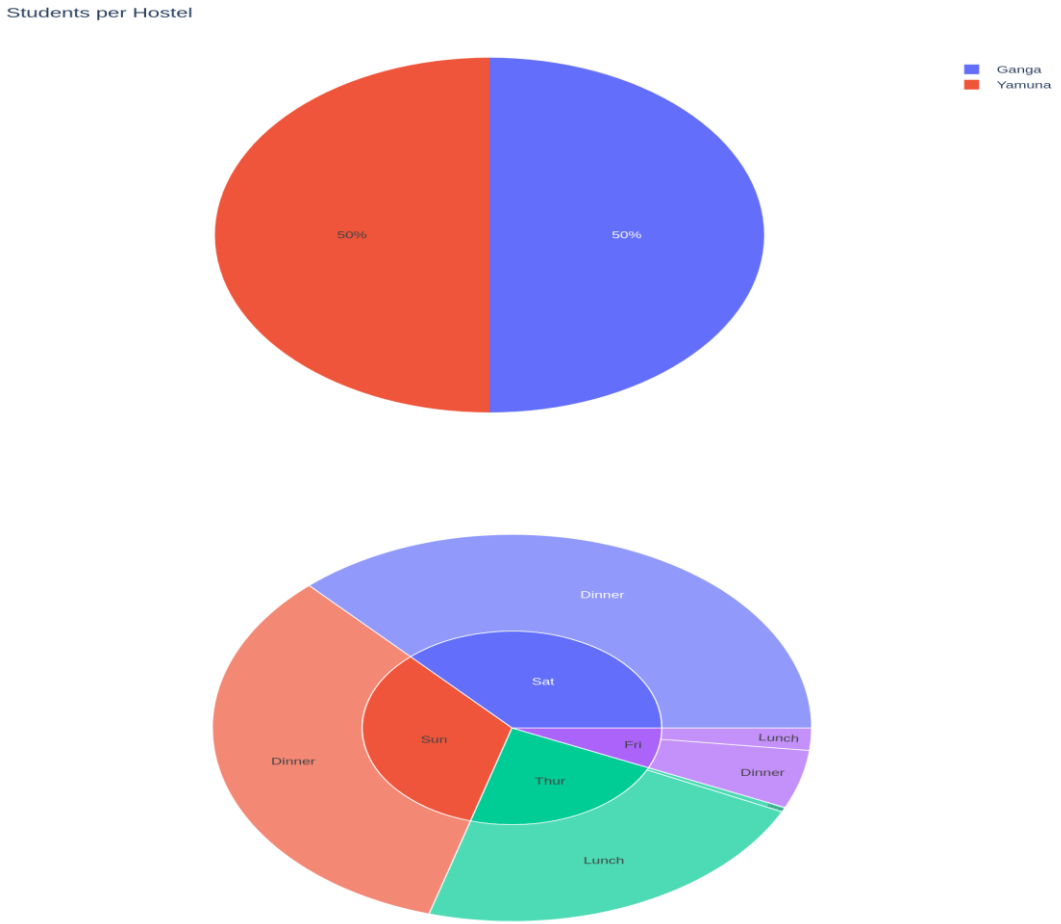
Syntax:

```
px.box(data_frame, x=None, y=None, color=None, points="outliers")
px.violin(data_frame, x=None, y=None, box=False, points=None)
```

- **x, y:** category on x, numeric on y – one box/violin per category
- **points:** "outliers" (default), "all" shows every observation, False hides them
- **color:** a second grouping – boxes appear side by side within each category
- **box=True:** (violin) draws a mini box plot inside the density shape
- **what the box is:** median, quartiles, Tukey whiskers – exactly the L6 definitions
- **violin adds:** the full distribution shape – catches bimodality a box hides

Pie and Sunburst

```
fig = px.pie(df, names="Hostel",
             title="Students per Hostel")
fig.show()

fig = px.sunburst(tips, path=["day", "time"],
                  values="total_bill")
fig.show()
```



px.pie counts the categories for you; sunburst = hierarchical pie

px.pie and px.sunburst – Syntax

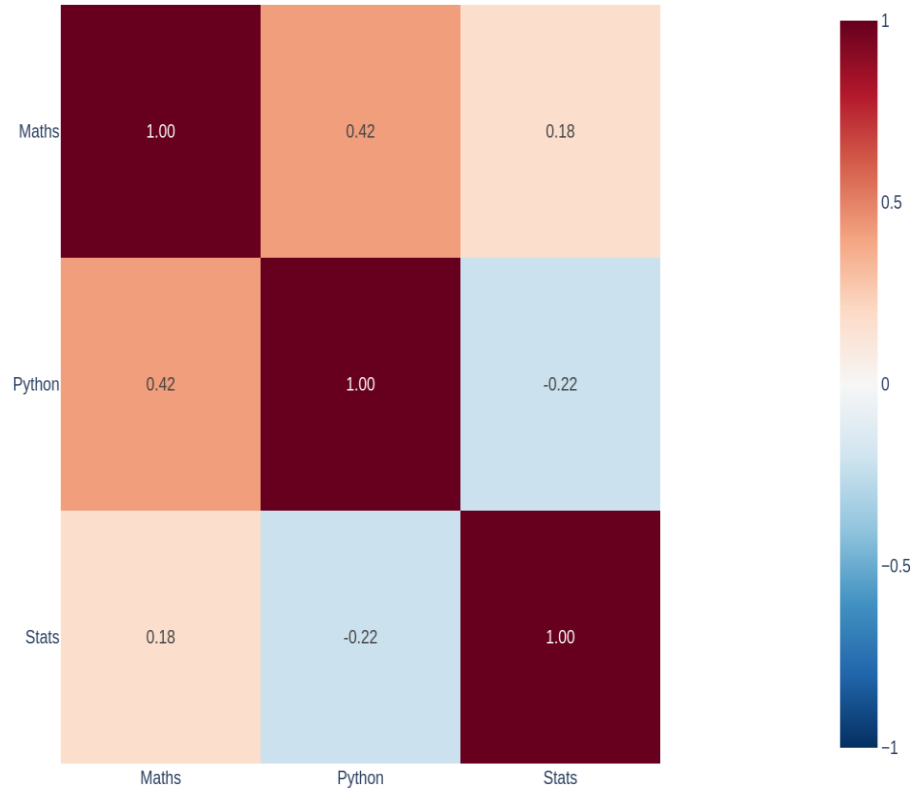
Syntax:

```
px.pie(data_frame, names=None, values=None, hole=0)
px.sunburst(data_frame, path=None, values=None, color=None)
```

- **names:** the category column; with no values, px.pie simply counts the rows
- **values:** a numeric column – slice sizes proportional to it
- **hole:** 0 to 1 – hole=0.4 turns the pie into a donut
- **path:** (sunburst) the hierarchy, outermost first: ["day", "time"]
- **interaction:** click a sunburst sector to zoom into that branch; click centre to return
- **honesty note:** angles are hard to read (L10) – few categories only, else use bars

Heatmap

```
corr = df[["Maths","Python","Stats"]].corr()

fig = px.imshow(corr, text_auto=".2f",
                color_continuous_scale="RdBu_r",
                zmin=-1, zmax=1)
fig.show()
```



Diverging colour scale centred at zero – the correct choice for correlations

px.imshow – Syntax

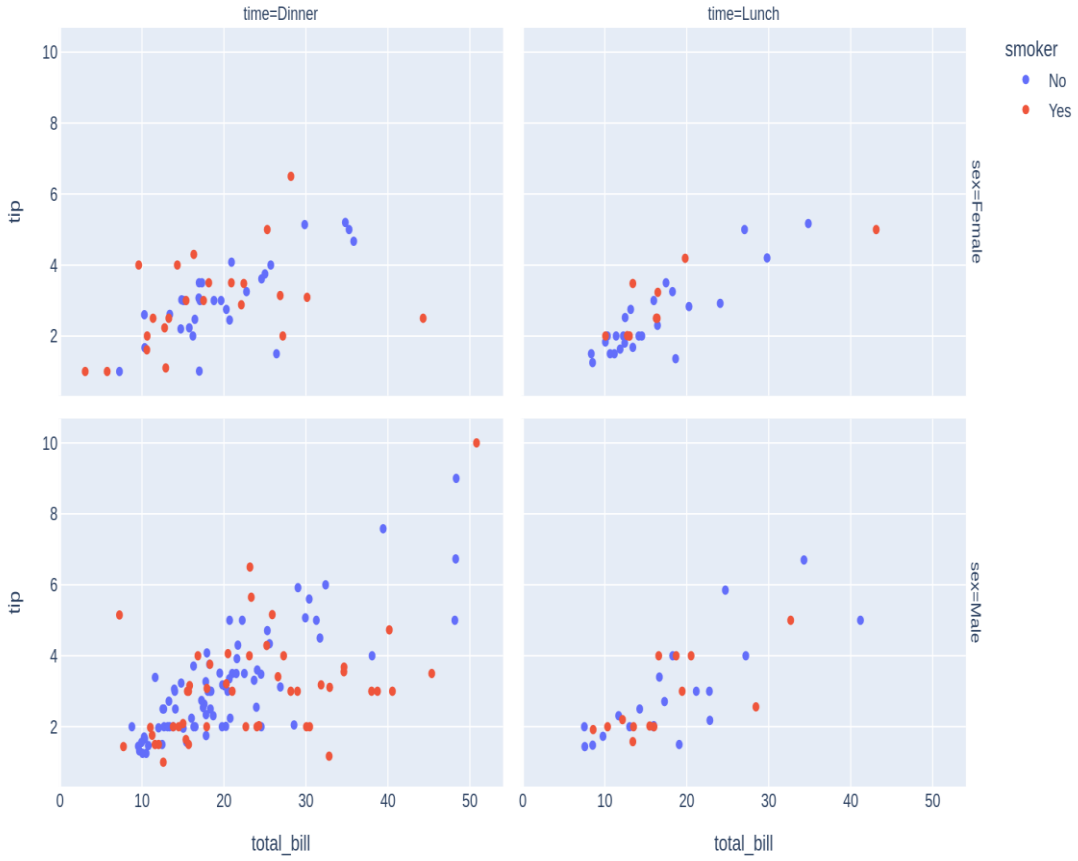
Syntax:

```
px.imshow(img, text_auto=False, color_continuous_scale=None,
          zmin=None, zmax=None, aspect=None)
```

- **img:** a 2-D array or DataFrame (heatmap) – or an actual image
- **text_auto:** True prints each value in its cell; ".2f" controls the decimals
- **color_continuous_scale:** "Blues", "Viridis", "RdBu_r" – append _r to reverse
- **zmin, zmax:** pin the scale ends – −1 and +1 so correlations are comparable
- **aspect:** "auto" stretches cells to fill the figure; default keeps them square
- **DataFrame input:** row and column labels become the axis labels automatically

Facets – Small Multiples

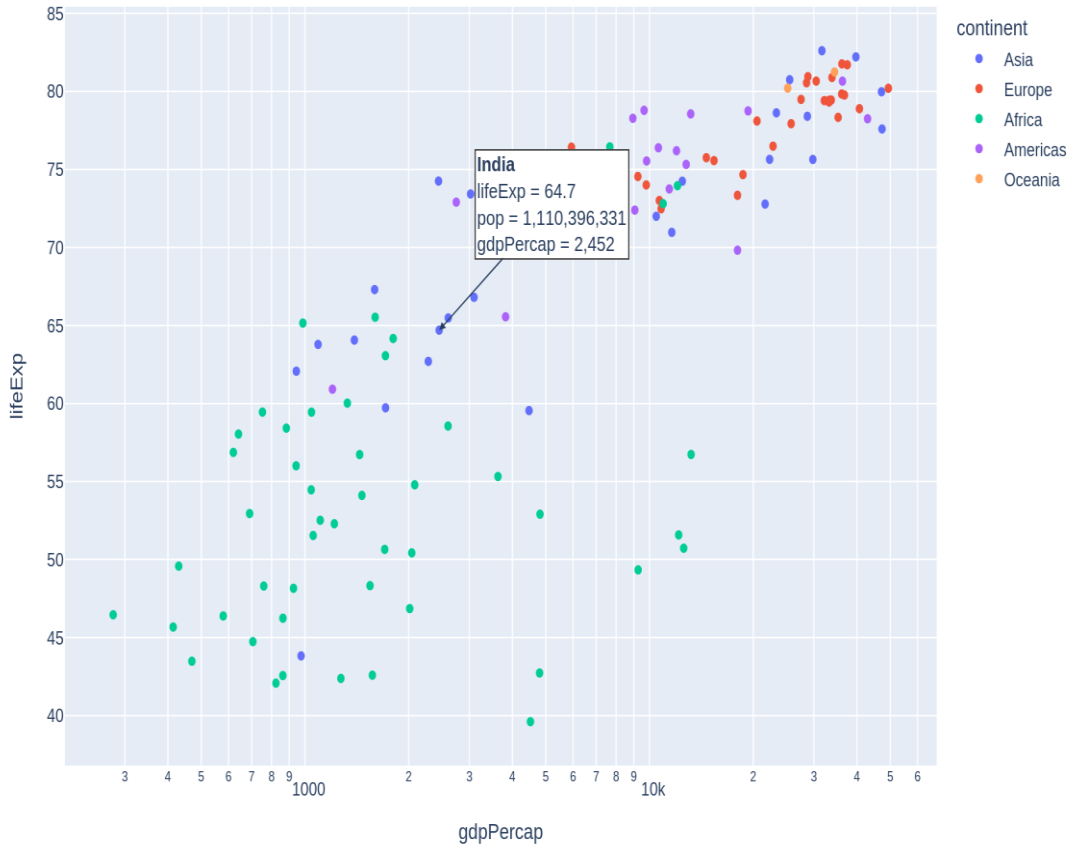
```
fig = px.scatter(tips, x="total_bill", y="tip",
                 color="smoker",
                 facet_col="time",
                 facet_row="sex")
fig.show()
```



One panel per category, shared scales – comparison without clutter

Hover – Detail on Demand

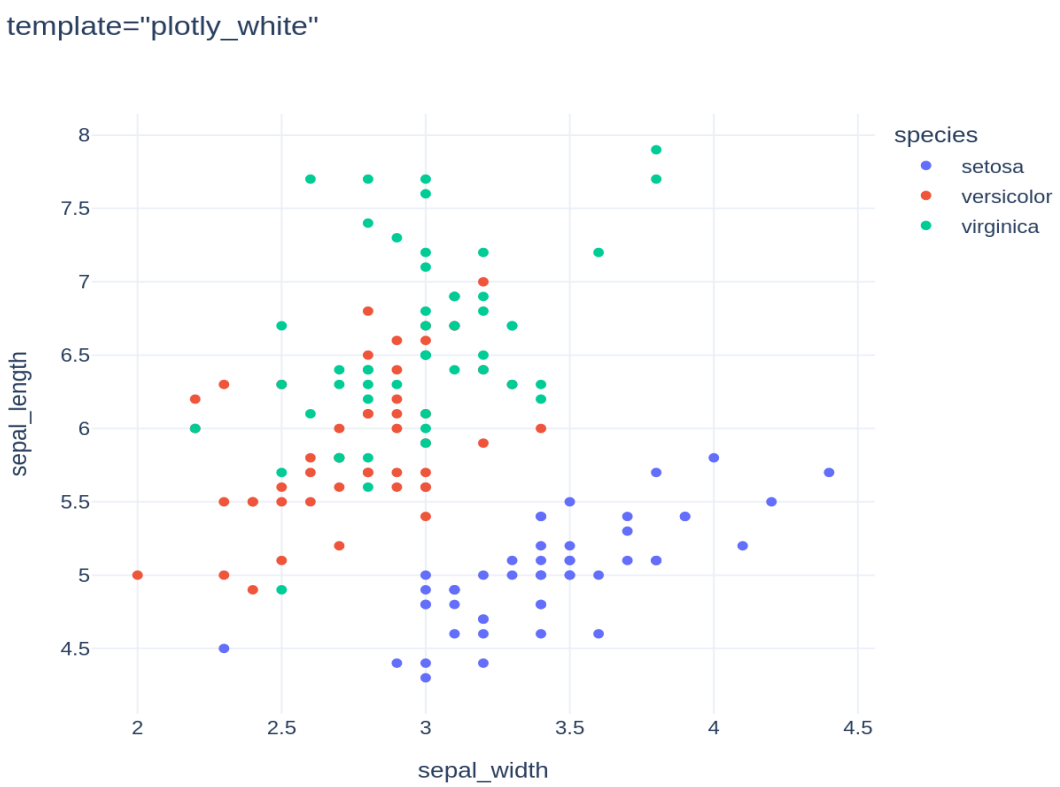
```
gap07 = px.data.gapminder().query("year == 2007")

fig = px.scatter(gap07, x="gdpPercap", y="lifeExp",
                 color="continent", log_x=True,
                 hover_name="country",
                 hover_data={"pop": ":,"})
fig.show()
```

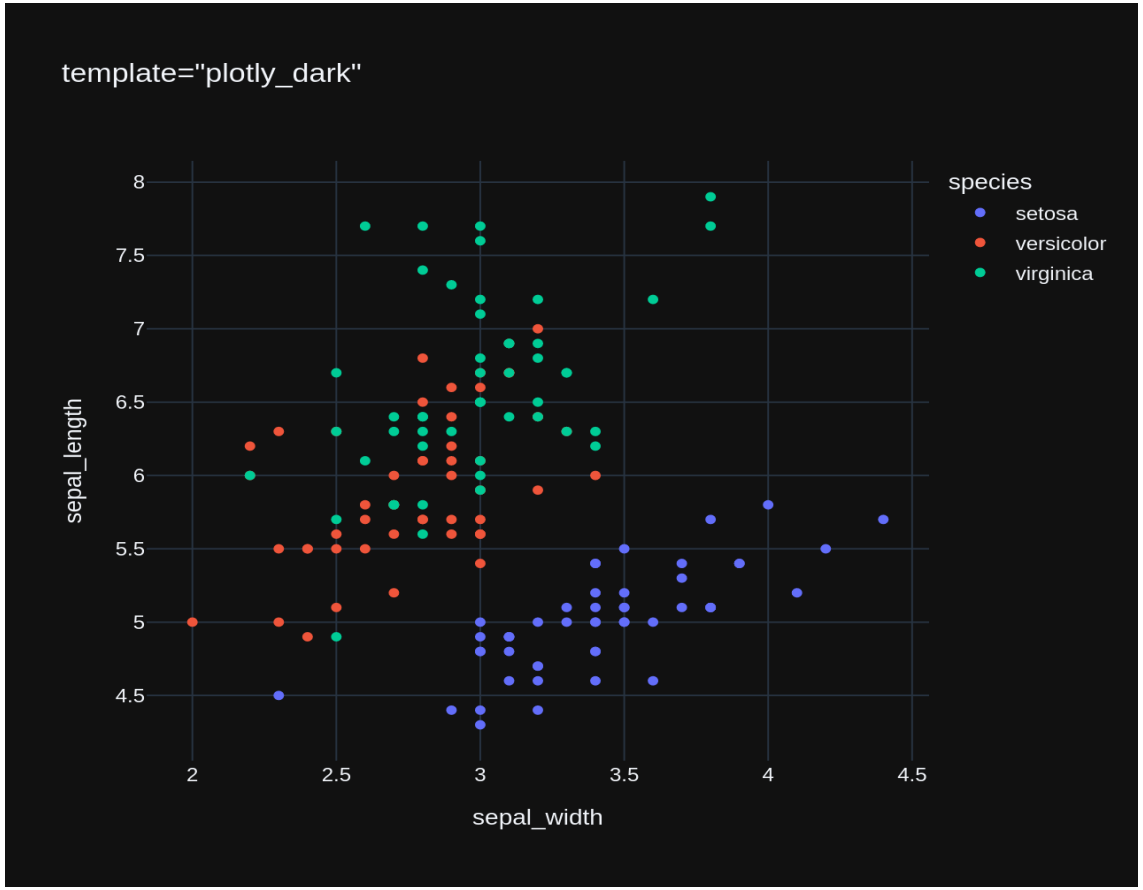


Move the mouse over any point – the tooltip shows that country's values

Themes (Templates)

- fig.update_layout(template="plotly_white") – one line restyles the whole figure
- Available: plotly, plotly_white, plotly_dark, simple_white, ggplot2, seaborn





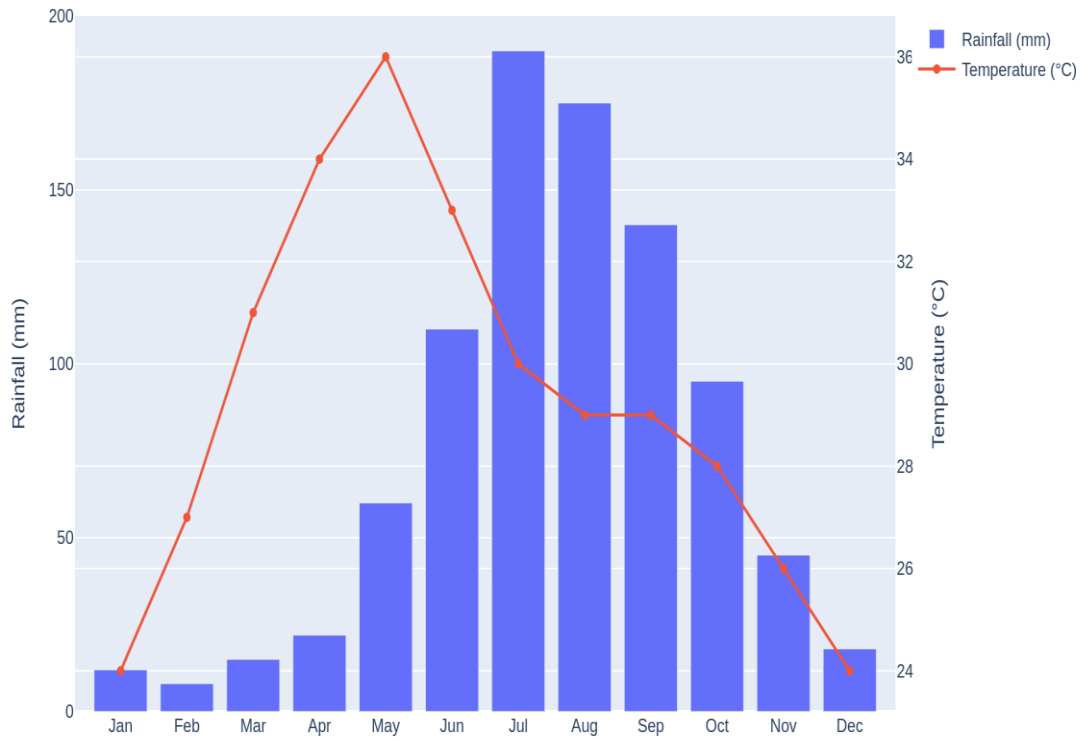
graph_objects – Two Traces

```
last = [10,11,18,30,60,95,160,150,120,80,40,15]

fig = go.Figure()
fig.add_trace(go.Scatter(x=months, y=rain,
    mode="lines+markers", name="This year"))
fig.add_trace(go.Scatter(x=months, y=last,
    mode="lines+markers", name="Last year"))
fig.update_layout(title="Rainfall: Two Years")
fig.show()
```

go builds the figure trace by trace – full control when px is not enough

Combining Chart Types

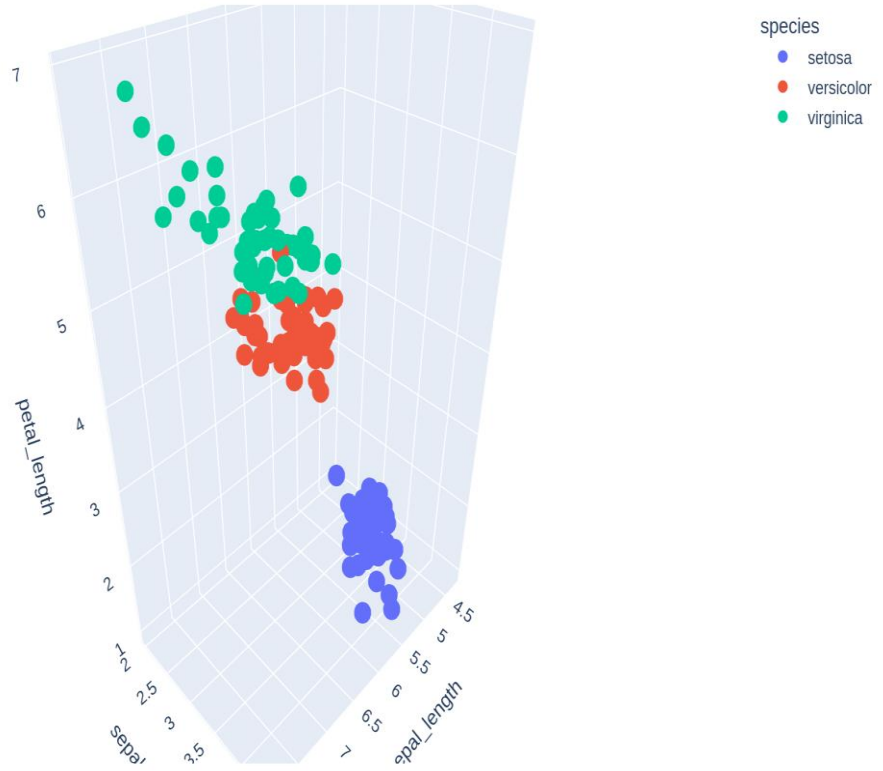
```
temp = [24,27,31,34,36,33,30,29,29,28,26,24]

fig = go.Figure()
fig.add_trace(go.Bar(x=months, y=rain,
    name="Rainfall (mm)"))
fig.add_trace(go.Scatter(x=months, y=temp,
    name="Temp (°C)", yaxis="y2"))
fig.update_layout(yaxis2=dict(
    overlaying="y", side="right"))
fig.show()
```



Bar + line with a secondary axis – use sparingly and label both axes

3D Scatter

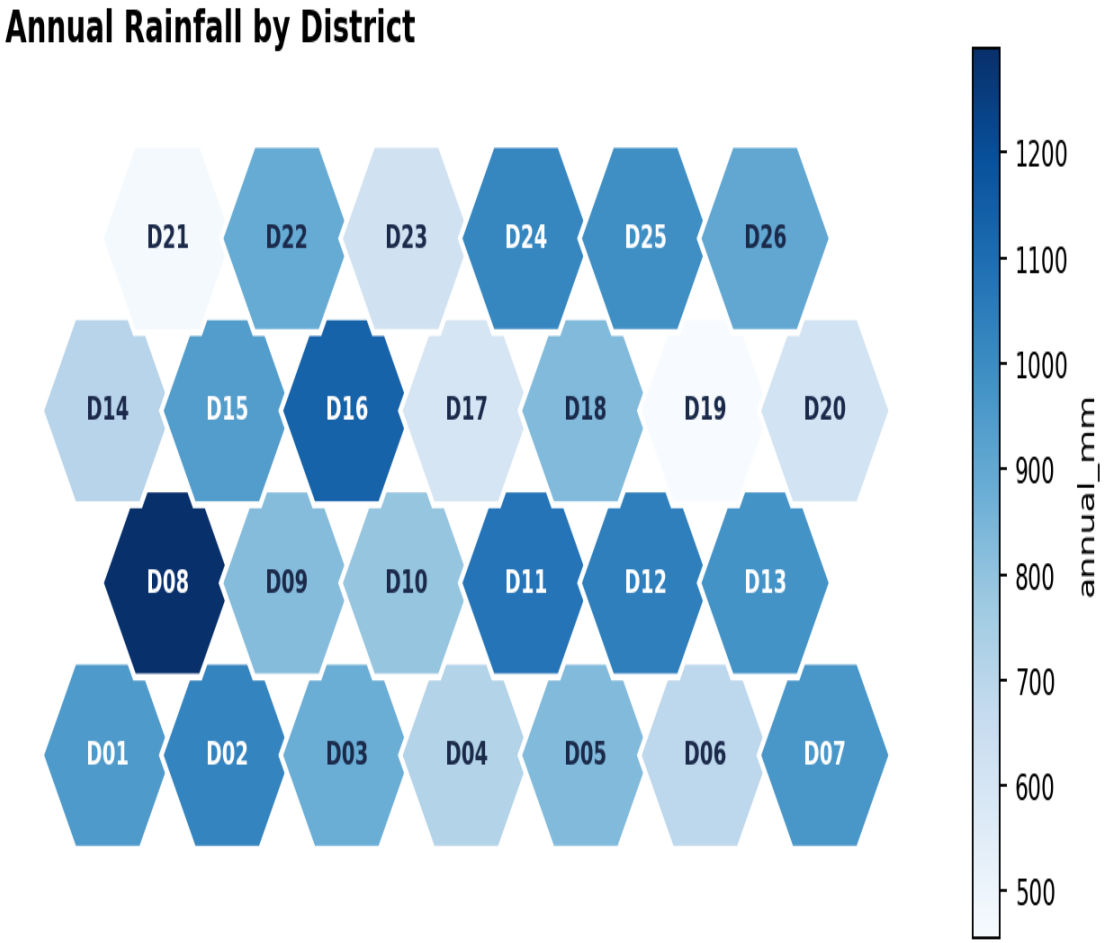
```
fig = px.scatter_3d(iris, x="sepal_length",
                    y="sepal_width",
                    z="petal_length",
                    color="species")
fig.show()
```



Rotate with the mouse – 3D is for exploration, rarely for reports

Maps – Choropleth (Demo)

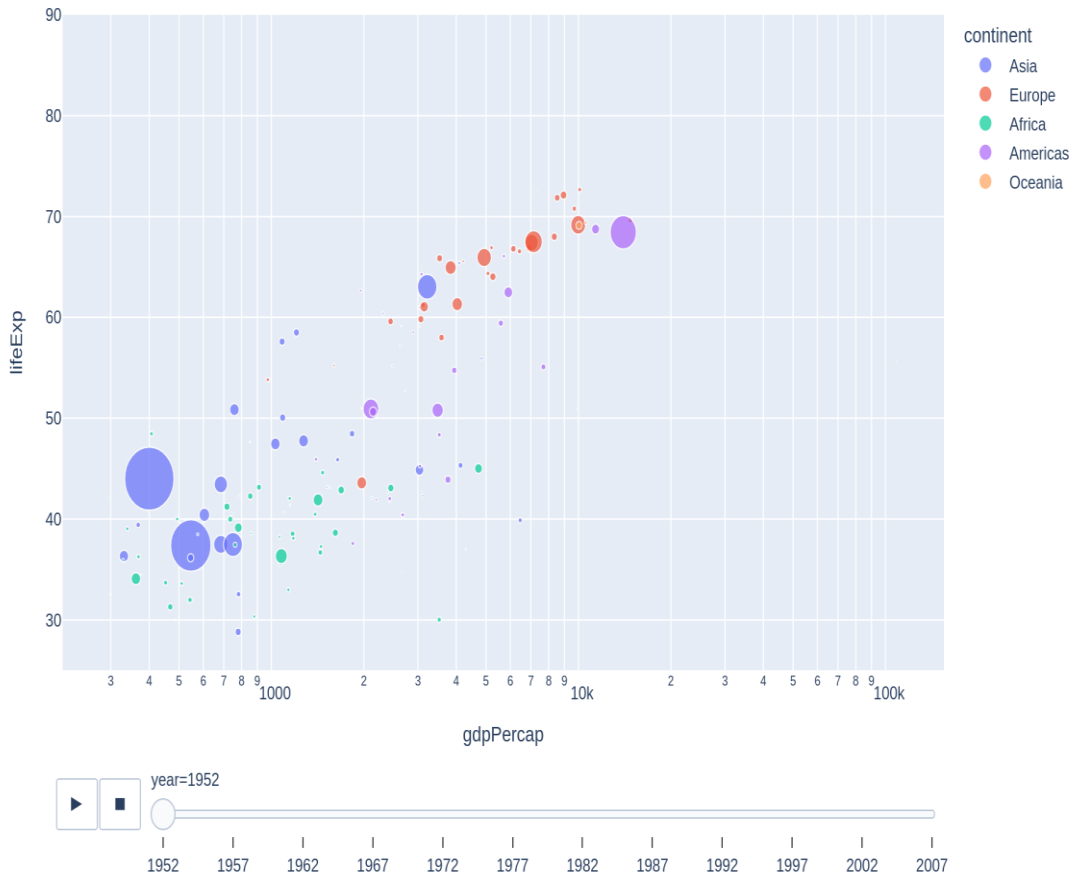
```
gap07 = px.data.gapminder().query("year == 2007")

fig = px.choropleth(gap07,
        locations="iso_alpha",
        color="lifeExp",
        hover_name="country",
        color_continuous_scale="Blues")
fig.show()
```



Your output: a world map coloured by life expectancy (illustration shown). Map rates, not counts.

Animation (Demo)

```
fig = px.scatter(px.data.gapminder(),
        x="gdpPercap", y="lifeExp",
        size="pop", color="continent",
        animation_frame="year",
        animation_group="country",
        log_x=True, range_y=[25, 90])
fig.show()
```



The famous Gapminder animation – press play, or drag the year slider

Saving Figures

```
fig.write_html("figure.html")    # interactive, share anywhere
fig.write_image("figure.png")    # static (pip install kaleido)
fig.write_image("figure.pdf")    # vector, for print
```

write_html keeps hover and zoom – send it to anyone with a browser

Plotly vs Matplotlib vs Seaborn

- Matplotlib – static, total control, the foundation everything builds on
- Seaborn – statistical graphics with beautiful defaults, still static
- Plotly – interactive, web-native, dashboard-ready
- Reports and papers: Matplotlib/Seaborn. Exploration: Plotly
- You now know all three – choose by audience, not by habit

Try It Now

- Scatter of Stats vs Maths from df – coloured by Hostel, sized by Python
- Bar chart of each student’s Python marks – sorted, values printed on bars
- Histogram of 100 simulated marks – try nbins = 5, 12, and 30
- Line of rain with markers – add last year as a second trace
- One tips chart of your choice – restyle it with template="plotly_dark"
- Save your best figure as HTML and open it in the browser