

# Python Visualization Basics

From notebooks → web-ready charts: SVG/HTML outputs, interactive graphics, and modern chart components.

**Marc Reyes**

Professional Lecturer · [marc.reyes@dlsu.edu.ph](mailto:marc.reyes@dlsu.edu.ph)  
DATA101 — De La Salle University

# Today's Plan

01 · WEB OUTPUTS

## **HTML, CSS, SVG**

What the browser actually renders.

02 · PYTHON WORKFLOW

## **Notebook → chart → export**

Reproducible visuals you can ship.

03 · MODERN COMPONENTS

## **Scales, marks, guides**

A reusable mental model.

04 · INTERACTIVITY

## **Selections + tooltips**

When interactivity is worth it.

# Learning Outcomes

WEB

**Explain SVG vs PNG vs HTML**

And choose the right export format.

PYTHON

**Build a reproducible chart workflow**

Data → transforms → plot → export.

DESIGN

**Name modern chart components**

Scales, axes, marks, guides, interactions.

PRACTICE

**Write code that produces legible outputs**

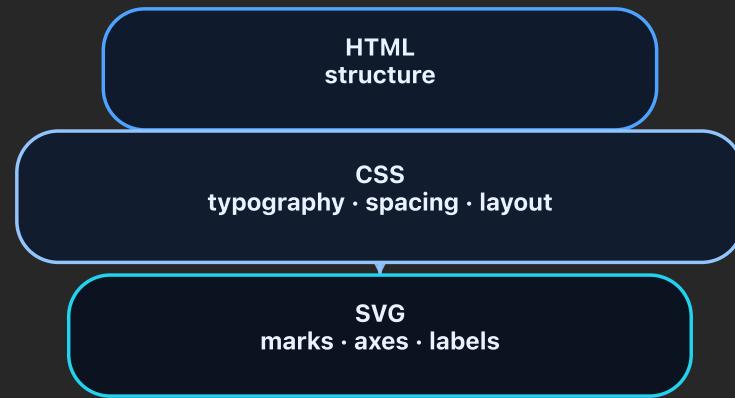
Readable on slides, PDFs, and web pages.

PART 1 · WEB OUTPUTS

# What the browser understands

HTML + CSS + SVG as the delivery layer for charts

# The Web Output Stack (for Charts)



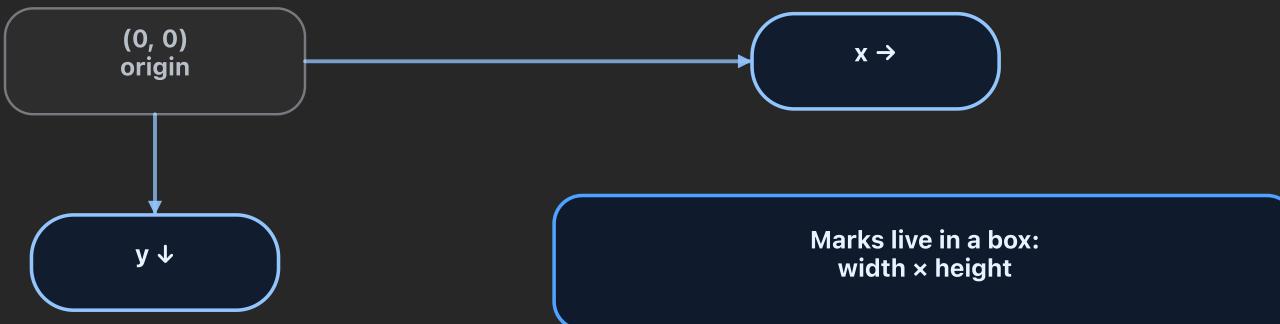
# SVG: The “Native” Format of Many Charts

## WHY SVG

- Scales crisply (great for slides and PDFs)
- Text remains selectable and searchable
- Shapes are editable (Illustrator/Figma)

## WHEN NOT SVG

- Huge point clouds (file size)
- Complex maps with many paths
- Photographic backgrounds



# CSS: The Hidden Part of “Professional”

## CSS CONTROLS

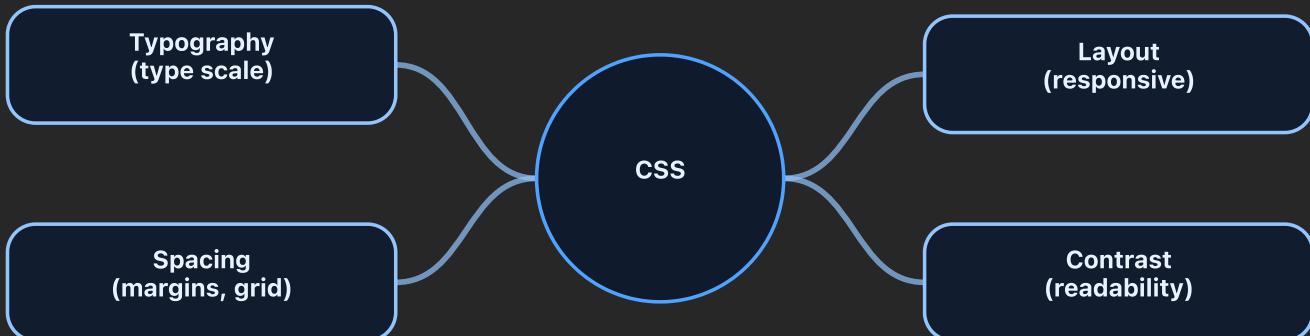
### Hierarchy + spacing + readability

Font sizes, line heights, margins, contrast.

## CHART IMPLICATION

### Your chart lives inside a layout

So it must be responsive and legible.



PART 2 · PYTHON WORKFLOW

# Notebook → chart → export

Reproducible visuals you can ship

# A Repeatable Python Visualization Pipeline



# Pandas as the “Chart Data Engine”

## COMMON TRANSFORMS

- groupby + aggregate
- derive rates and deltas
- melt / tidy reshape
- sort for ranking

## WHY IT MATTERS

### Most chart bugs are data bugs

Wrong denominator, wrong unit, wrong grain.

Raw data  
(messy)

Tidy table  
(one row = one observation)

Encode  
(field → channel)

Chart  
(legible)

# Code Demo A: Make a Chart-Ready Table

OUTPUT YOU SHOULD SEE

A tidy table with a derived rate + delta.

## RUNNABLE PYTHON

Copy into a notebook and run.

```
import pandas as pd

df = pd.DataFrame(
    {
        "program": ["A", "A", "B", "B", "C", "C"],
        "week": [1, 2, 1, 2, 1, 2],
        "n_pass": [70, 62, 90, 88, 40, 44],
        "n_students": [100, 100, 120, 120, 50, 50],
    }
)

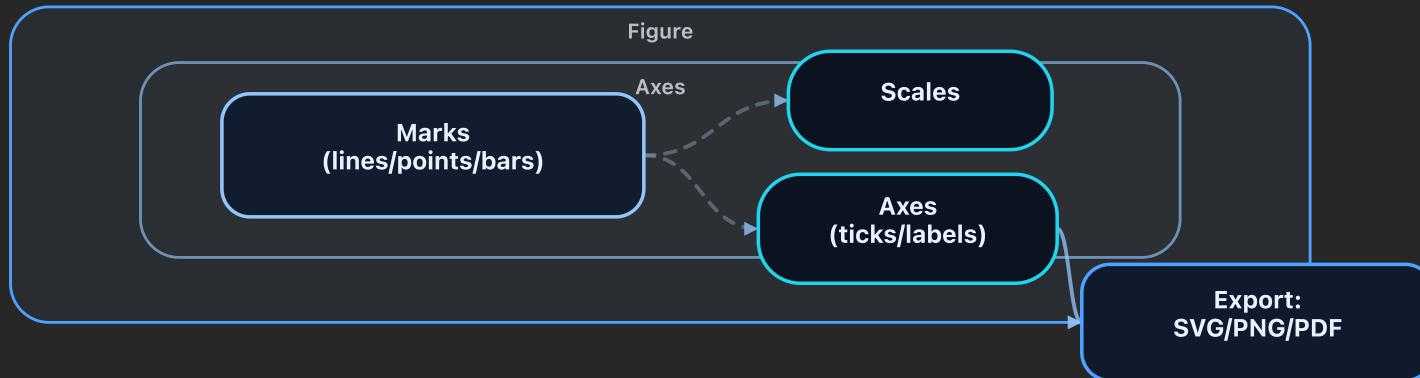
df["pass_rate"] = df["n_pass"] / df["n_students"]

weekly = (
    df.sort_values(["program", "week"])
    .assign(delta=lambda d: d.groupby("program")["pass_rate"]
)
    .groupby(["program", "week"])
    .mean()
    .reset_index()
)

print(weekly.to_string(index=False))
```

program	week	n_pass	n_students	pass_rate	delta
A	1	70	100	0.7	—
A	2	62	100	0.62	-0.08
B	1	90	120	0.75	—
B	2	88	120	0.733	-0.017
C	1	40	50	0.8	—
C	2	44	50	0.88	0.08

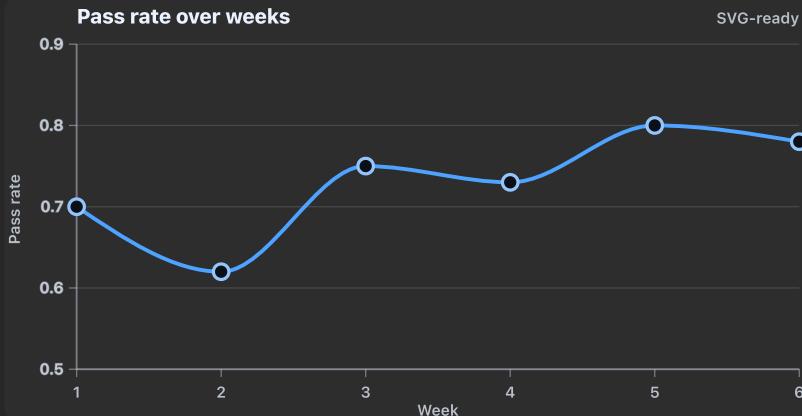
# Matplotlib: The “Artist” Model (Mental Map)



# Code Demo B: Matplotlib → SVG Export

RENDERED RESULT (ILLUSTRATION)

Your SVG will scale crisply in slides.



## RUNNABLE PYTHON

Produces a plot and exports SVG text (web-ready).

```
import io
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(1, 7)
y = np.array([0.70, 0.62, 0.75, 0.73, 0.80, 0.78])
fig, ax = plt.subplots(figsize=(7.2, 3.2))
ax.plot(x, y, marker="o", linewidth=2)
ax.set_title("Pass rate over weeks")
ax.set_xlabel("Week")
ax.set_ylabel("Pass rate")
ax.set_ylim(0.5, 0.9)
ax.grid(True, alpha=0.25)
fig.tight_layout()

buf = io.StringIO()
fig.savefig(buf, format="svg")
svg = buf.getvalue()
print("SVG chars:", len(svg))
```

# Seaborn: Statistics Defaults + Cleaner Styles

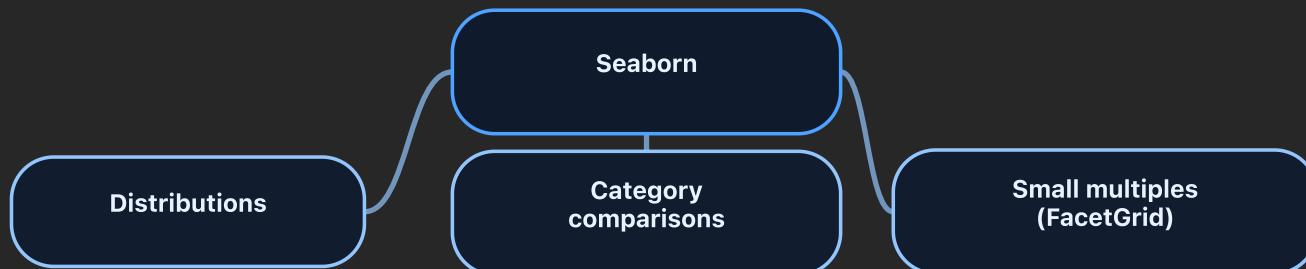
## WHAT IT ADDS

- Statistical plots (distributions, categories)
- Reasonable default aesthetics
- Easy small multiples ( FacetGrid )

## COMMON TRAP

**Pretty defaults ≠ correct story**

Always check units, bins, and baselines.



# Code Demo C: Faceted Histograms

RUNNABLE PYTHON

Shows why distribution beats averages.

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

rng=np.random.default_rng(7)
program=np.repeat(list("ABC"), 250)
score=np.r_[rng.normal(76, 6, 250), rng.normal(76, 14, 250),
df=pd.DataFrame({"program": program, "score": score})

sns.displot(
    df, x="score", col="program", bins=18,
    facet_kws=dict(sharesx=True, sharey=True),
    height=3, aspect=1.1
)
print(df.groupby("program")["score"].agg(["mean", "std"]).rou
plt.show()
```

ModuleNotFoundError: No module named 'numpy'

Tip: This may because of this package is not a [Pyodide](#) builtin package.

## WHAT TO LOOK FOR

- Panels share axes (fair comparison)
- Shape shows variance (not just the mean)
- Bins + scale are design decisions

# Code Demo C (Output): Small Multiples Reveal Spread

## INTERPRETATION

### Same mean, different risk

- Wider distributions imply more extreme outcomes
- Aligned panels make differences obvious
- Label bins and units when you publish

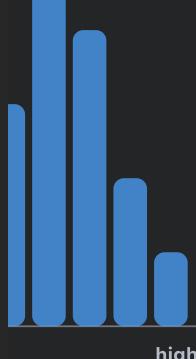
## RENDERED RESULT (ILLUSTRATION)

This is what your FacetGrid histogram should resemble.

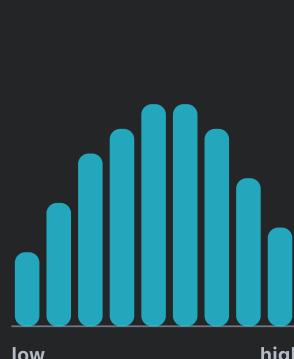
### Distributions (small multiples)

Programs reveal spread differences

A



Program B



Program C

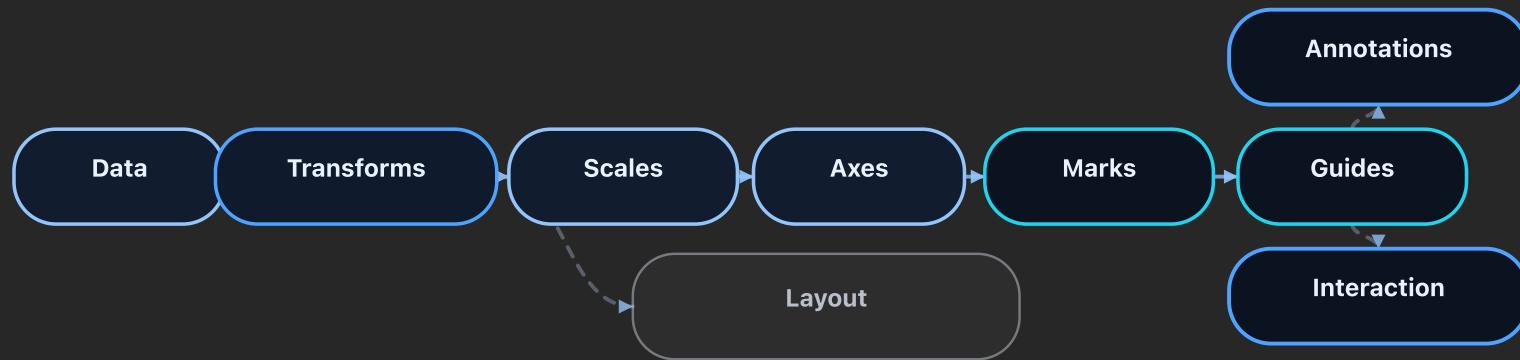


PART 3 · MODERN COMPONENTS

# Think in chart parts

Build charts like reusable UI components

# The Modern Chart Component Checklist



# “D3 Concepts” in a Python World

CORE IDEA

**Data  $\leftrightarrow$  marks mapping**

Bind rows to marks; encode columns to channels.

PYTHON ANALOGY

**DataFrame  $\rightarrow$  spec  $\rightarrow$  renderer**

Altair/Plotly generate web renderers.

DataFrame  
(rows)

Spec  
(field  $\rightarrow$  channel)

Renderer  
(Vega/JS)

Output  
SVG/HTML

# Code Demo D: Altair Encodings

## RUNNABLE PYTHON

Data → encoding → interactive tooltip.

```
import pandas as pd
import json

df=pd.DataFrame({
    "x": [1,2,3,4,5,6],
    "y": [0.70,0.62,0.75,0.73,0.80,0.78],
    "term": ["baseline"]*3 + ["current"]*3
})

spec={"mark":{"type":"circle","size":110},
      "encoding":{"x":{"field":"x","type":"quantitative"},
                  "y":{"field":"y","type":"quantitative","s
                      "color":{"field":"term","type":"nominal"}
                      "tooltip":[{"field":"x"}, {"field":"y"}, {""
print("First 3 rows:")
print(df.head(3).to_string(index=False))
print("\nEncoding spec (Vega-Lite style, excerpt):")
```

## WHAT TO WATCH

- Domains are explicit (comparison-friendly)
- Color encodes category, not magnitude
- Tooltip adds detail without clutter

# Code Demo D (Output): Encoded Scatter + Tooltip

## DESIGN NOTE

### Encoding is a contract

- Quantitative → position/scale
- Categorical → hue/grouping
- Set domains when comparisons matter

## RENDERED RESULT (ILLUSTRATION)

Your Altair chart should match this structure.

### Encoded scatter



PART 4 · INTERACTIVITY

# Interactivity with purpose

Tooltips, selection, filtering, and readable dashboards

# Tooltips Are “Details on Demand”

GOOD TOOLTIP

**Confirms values**

Units, exact numbers, IDs for traceability.

BAD TOOLTIP

**Replaces the chart**

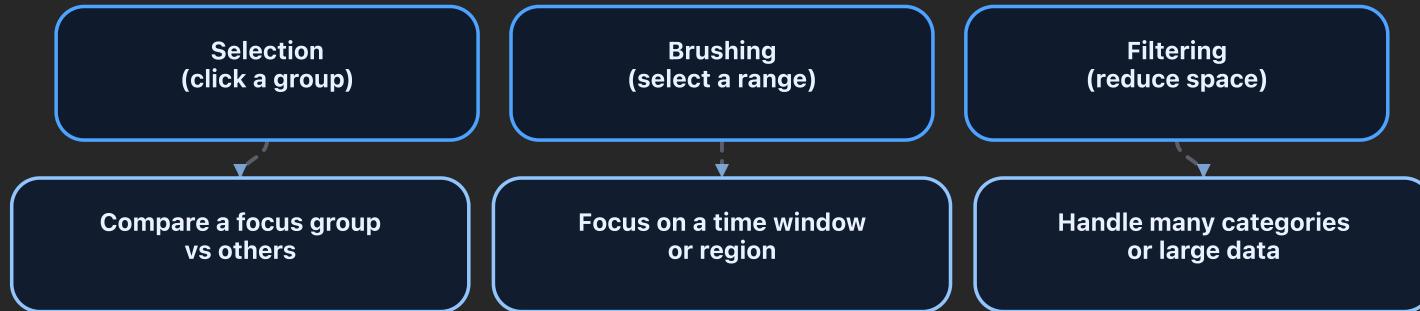
If you need to hover everything, redesign.

Chart

Tooltip

Rule:  
Confirm, don't decode

# Selections, Brushing, and Filtering (Task-Driven)



# Code Demo E: Plotly for Interactive HTML

RENDERED RESULT (ILLUSTRATION)

Hover + zoom are built in.

## RUNNABLE PYTHON

Exports an interactive chart to HTML.

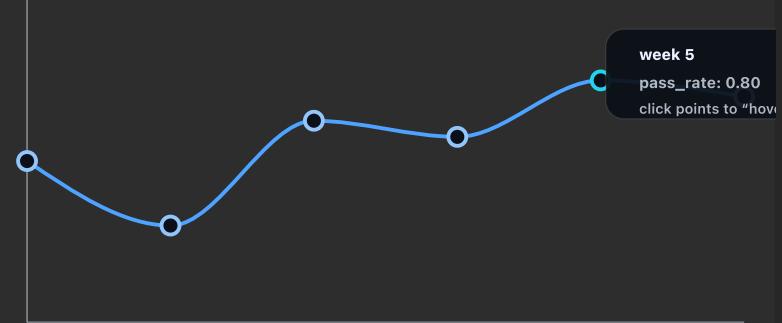
```
import pandas as pd
import plotly.express as px

df = pd.DataFrame(
    {
        "week": [1, 2, 3, 4, 5, 6],
        "pass_rate": [0.70, 0.62, 0.75, 0.73, 0.80, 0.78],
    }
)

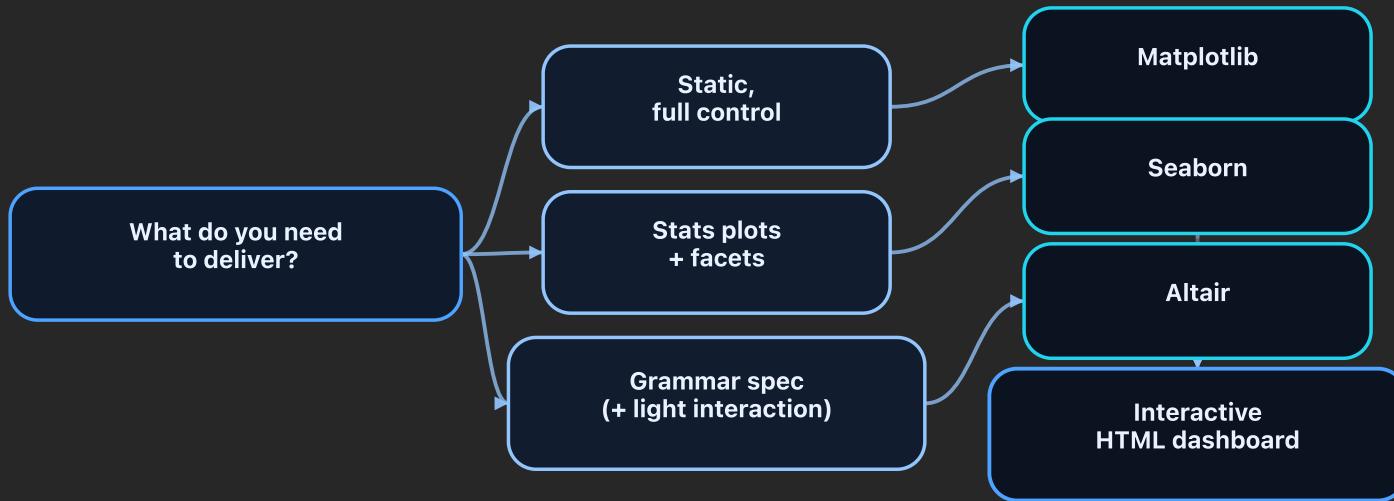
fig = px.line(df, x="week", y="pass_rate", markers=True, ti
fig.update_yaxes(range=[0.5, 0.9])
fig.write_html("pass_rate.html", include_plotlyjs="cdn")
fig
```

## Interactive line (HTML)

tooltip + zoom



# Choosing the Right Tool (A Practical Heuristic)



# Export Formats: What You Hand In (and Why)

SVG

**Slides + print**

Crisp, editable, searchable.

PNG

**Fixed images**

Good for dense marks; stable layout.

HTML

**Interactive**

Tooltips, zoom, filters.

RULE

**Choose format based on \*use case\*, not preference.**

# Micro-Checklist: “Looks Professional” in Practice

## TYPOGRAPHY

- Readable title (claim or task)
- Axis labels with units
- Limited tick density

## GRAPHICS

- Aligned scales for comparisons
- Legend only if necessary
- One clear emphasis (not rainbow)

## DATA

- Rates vs counts handled
- Missingness shown explicitly
- Aggregation explained

## EXPORT

- SVG/PNG/HTML matches use case
- Consistent sizing across figures
- Works on dark backgrounds

## Practice (In Class): Make One Chart, Three Exports

TASK

**Create one chart and export it as SVG, PNG, and HTML.**

Then explain which format you would submit for: slides, a PDF report, and an interactive critique.

- Pick a simple dataset (10–200 rows)
- Include units and a baseline if relevant
- Write 2–3 sentences justifying your export choices

layout: section

# Python-first visualization craft

Make charts that ship: readable, accessible, reusable.

# Styling is a constraint, not decoration

PROFESSIONAL HABIT

## Use one theme across charts

Typography, sizes, gridlines, colors.

WHY IT MATTERS

## Consistency builds trust

Viewers stop re-learning your chart style each slide.

DESIGN RULE

## Make the data loud. Keep the scaffolding quiet.

# Live Python: A consistent Matplotlib style

## RUNNABLE PYTHON

Sets a small style system and draws a chart.

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

plt.rcParams.update({"figure.dpi": 120, "font.size": 12})

x = np.arange(1, 9)
y = np.array([72, 71, 74, 76, 75, 78, 80, 79])

fig, ax = plt.subplots(figsize=(7.2, 2.8))
ax.plot(x, y, marker="o", linewidth=2)
ax.set_title("Consistent style: readable by default")
ax.set_xlabel("Week")
ax.set_ylabel("Score")
ax.grid(True, alpha=0.25)

buf = io.StringIO()
fig.savefig(buf, format="svg")
print("SVG length:", len(buf.getvalue()))

```

## WHAT TO NOTICE

- Font sizes are intentional
- Gridlines are subtle
- Labels are complete (units when needed)

# Color is a data encoding (not a theme)

MATCH MEANING

**Type → palette**

Categorical, ordered magnitude, baseline differences.

COMMON FAILURE

**Pretty ≠ interpretable**

If the scale is wrong, the chart is wrong.

# Matplotlib colormaps: choose by semantics

## SEQUENTIAL

**Magnitude: low → high**

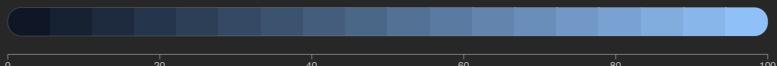
Use lightness ramps so order is visible.

## DIVERGING

**Difference: below ↔ above baseline**

Only when the midpoint is meaningful.

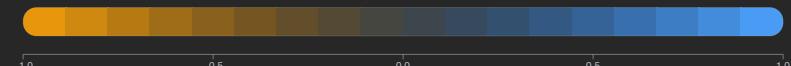
Lightness ramp (ordered magnitude)



Example: same values as a heatmap (higher = lighter)



Centered midpoint (baseline = 0)



Example: negative vs positive differences around the baseline



# Live Python: sampling colors from a colormap

## WHY THIS MATTERS

Colormaps are functions. You can sample them, test them, and keep them consistent across plots.

### RUNNABLE PYTHON

Prints a few RGBA samples from a Matplotlib colormap.

```
import matplotlib.cm as cm

cmap = cm.get_cmap("viridis")
samples = [cmap(i / 4) for i in range(5)]

for i, rgba in enumerate(samples):
    print(i, tuple(round(x, 3) for x in rgba))
```

ModuleNotFoundError: No module named 'matplotlib'

Tip: This may because of this package is not a Pyodide builtin package.

You may need to install it by adding the package name to the `python.installs` array in your headmatter.

# Accessibility basics for color

DO

## Add redundancy

Labels, position, shape—don't rely on color alone.

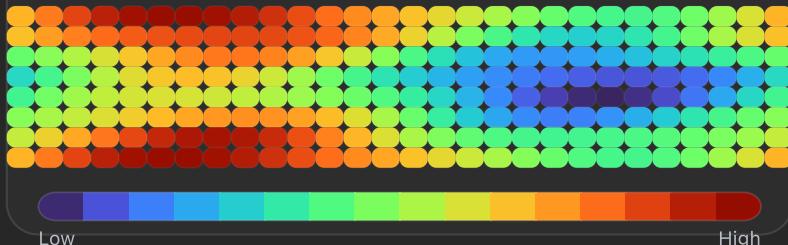
AVOID

## Red/green-only meaning

Many viewers cannot reliably distinguish it.

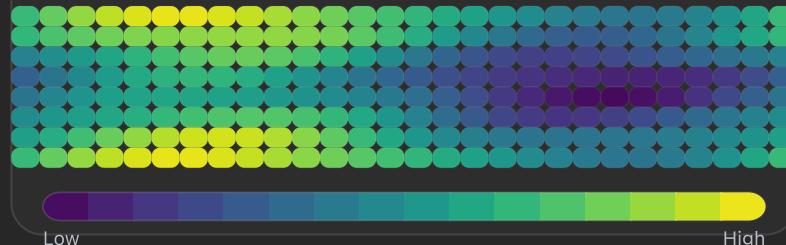
### Rainbow-like

Creates false boundaries



### Perceptual

Order visible in lightness



# Chart components as reusable code

INPUT

**DataFrame**

Types + units + grain.

PROCESS

**Transform + encode**

Compute chart-ready  
columns.

OUTPUT

**Figure object**

Exportable + consistent.

If your chart can't be wrapped as a function, it won't scale to a project.

# Live Python: a reusable chart function (template)

## HOW TO USE IT

Turn repeated chart decisions into parameters (titles, fields, scales, annotations).

### RUNNABLE PYTHON

A minimal “chart component” pattern.

```
from dataclasses import dataclass

@dataclass
class ChartSpec:
    title: str
    x: str
    y: str

def describe_chart(spec: ChartSpec) -> str:
    return f"{spec.title} | x={spec.x} | y={spec.y}"

print(describe_chart(ChartSpec("Pass rate trend", "week", "p
Pass rate trend | x=week | y=pass_rate
```

# Layout matters (even in notebooks)

GOOD LAYOUT

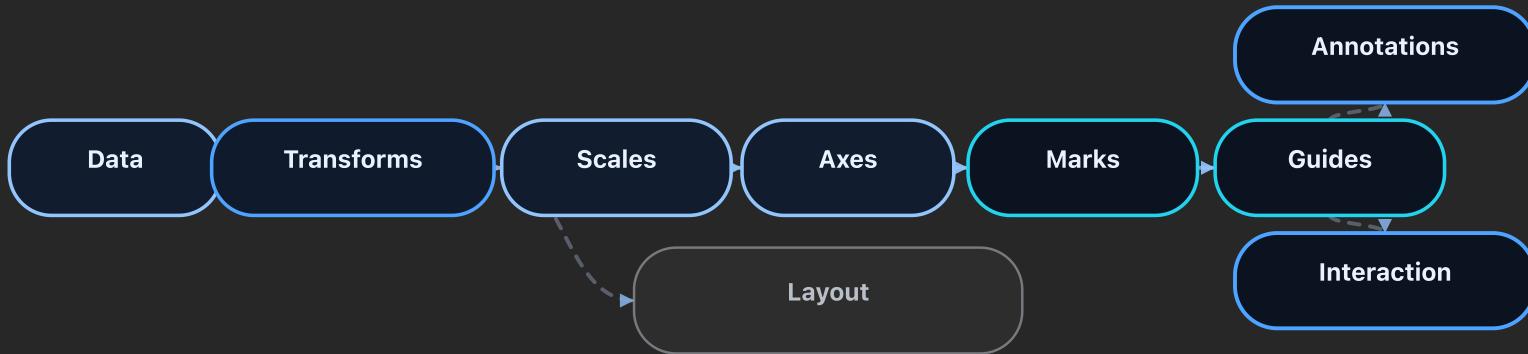
## Aligned comparisons

Shared scales; predictable reading order.

BAD LAYOUT

## Legend hunting

Too many colors; misaligned axes; crowded labels.



# Performance: reduce complexity before you draw

WHEN DATA IS LARGE

**Aggregate or bin**

Histograms, hexbin, summaries.

WHEN DATA IS DENSE

**Sample or faceting**

Downsample; split into panels.

**If you draw every point, you are encoding latency.**

# Publishing checklist (before you export)

## TEXT

- Readable title + labels
- Units included
- Consistent font sizes

## SCALES

- Baselines correct
- Domains chosen intentionally
- Comparisons are aligned

## COLOR

- Meaning matches palette
- Contrast is sufficient
- No color-only decoding

## EXPORT

- SVG for vector (slides)
- PNG for raster (photos)
- HTML for interaction

## Mini exercise (in-class)

### PROMPT

**Turn one messy table into a chart-ready table.**

Then choose a single chart and justify it in 4 sentences.

### DELIVERABLE

A tidy table + one exported figure (SVG or PNG).

### JUSTIFICATION

Task → transform → encoding → why it's readable.

# Export formats: choose the right artifact

SVG

## Slides + print

- Crisp at any zoom
- Searchable/selectable text
- Editable in Figma/Illustrator

Use when marks + labels must stay sharp.

PNG

## Screens + photos

- Reliable everywhere
- Good for raster layers (maps)
- Predictable file size

Use when you have imagery or heavy density.

HTML

## Interaction

- Tooltips + selections
- Responsive layouts
- Shareable dashboards

Use when interaction supports a task.

### RULE OF THUMB

If it needs to be read, prefer `SVG`. If it needs to be explored, prefer `HTML`. If it's an image, prefer `PNG`.

# Web-ready exports: what “done” looks like

## SHIP WITH CONTEXT

- A one-sentence caption (what + why)
- Units + time window + data source
- A note for missing data / caveats

A chart without context is a decoration.

## SHIP WITH STRUCTURE

- Consistent title sizing
- Aligned margins across figures
- Filenames: `topic\_metric\_scope\_date.svg`

The “professional” look is mostly layout discipline.

## EXPORT

Save at intended size (don’t “resize later”).

## CHECK

Open the file and verify labels/units are intact.

## EMBED

Place it into the final layout (slides, PDF, web page).

# Accessibility basics (for charts you publish)

## LEGIBILITY

- Contrast passes “squint test”
- Text is large enough at 100% zoom
- Direct labels when possible

## ROBUSTNESS

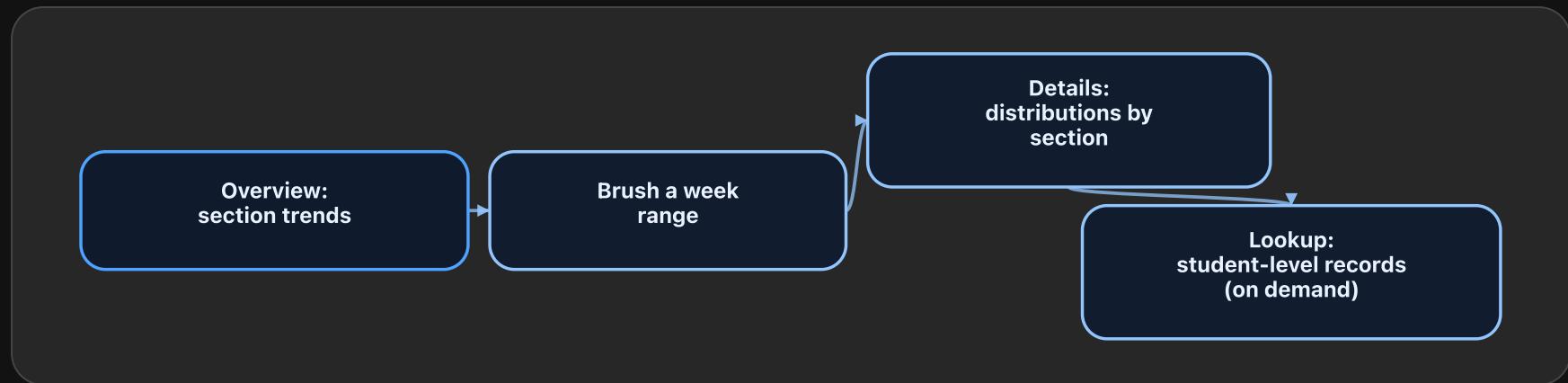
- No color-only meaning
- Patterns/markers for redundancy
- Clear “no data” encoding

## PROFESSIONAL HABIT

**Assume your chart will be viewed in bad conditions.**

Low brightness, projector washout, grayscale print, or viewers with color-vision differences.

## Case study: from overview to actionable detail



### WHY THIS PATTERN WORKS

Start broad (see trends), then narrow (choose a range), then inspect (compare distributions), then lookup only when necessary.

# Common export bugs (and quick fixes)

BUG

## Tiny text in the final file

It looked fine in the notebook, then became unreadable.

FIX

Set figure size + font sizes explicitly before export.

BUG

## Cropping / clipped labels

Axis labels or legends get cut off.

FIX

Use `tight\_layout()` / constrained layout and verify the file.

BUG

## Misleading scales after export

Domains/baselines changed across charts.

FIX

Lock domains for comparisons; label units; avoid implicit defaults.

RULE

Always open the exported file and proofread it like a report.

## Exit ticket (2 minutes)

1

**What is your dataset's "grain" after your transform?**

Example: one row = program × week.

2

**Which channel is doing the "hard work"?**

Position? Length? Lightness? (Name it.)

3

**What did you export, and why that format?**

SVG vs PNG vs HTML.

4

**One improvement you would make next iteration**

Labeling, domain, transform, or layout.

## Key Takeaways

- Web outputs matter: SVG/HTML are common “final forms”
- Python workflow: data → transforms → chart → export
- Modern charts are components: scales, marks, axes, guides, interaction
- Interactivity is only “professional” when it supports a task

## References (Recommended)

### Matplotlib documentation

Figure/Axes model, export formats, styling

<https://matplotlib.org/stable/>

### Altair documentation

Grammar of graphics + interactive selections

<https://altair-viz.github.io/>

### Plotly documentation

Interactive charts + HTML export

<https://plotly.com/python/>

### Wickham (2014)

Tidy data as a foundation for chart-ready tables

<https://doi.org/10.18637/jss.v059.i10>