

# Quiz Today

Please access the [Modules](#) page of AnimoSpace.

TOPIC

**Perception and Color**

ACCESS CODE

**7YT9FJYH1**

Time limit: **15 minutes**

Quiz closes at **6:20 PM**

# 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.

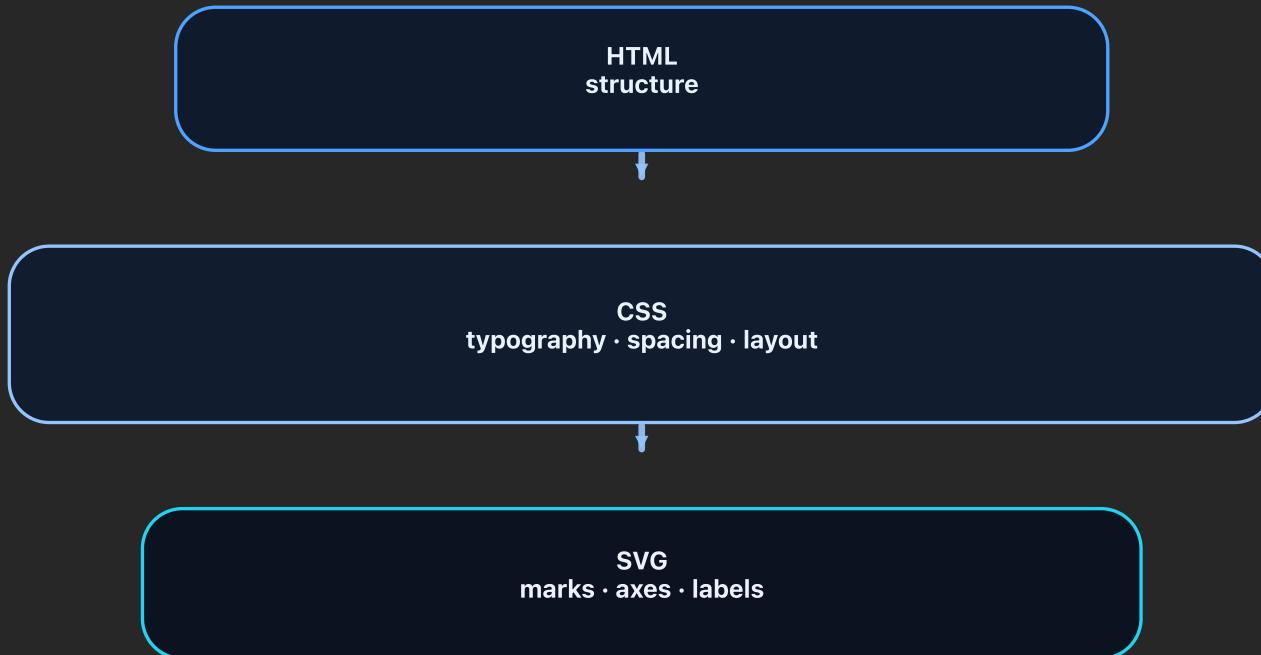
PART 1 · WEB OUTPUTS



# What the browser understands

HTML + CSS + SVG as the delivery layer for charts

# The Web Output Stack (for Charts)



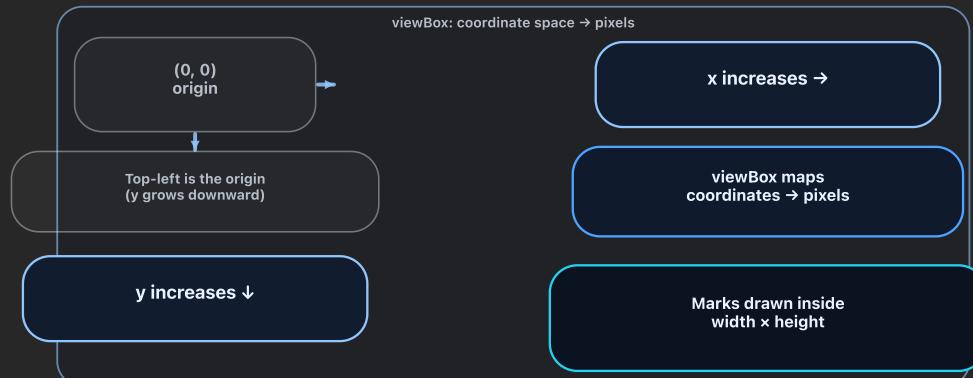
# SVG: The “Native” Format of Many Charts

## 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.

Typography  
(type scale)

Layout  
(responsive)

CSS

Spacing  
(margins, grid)

Contrast  
(readability)

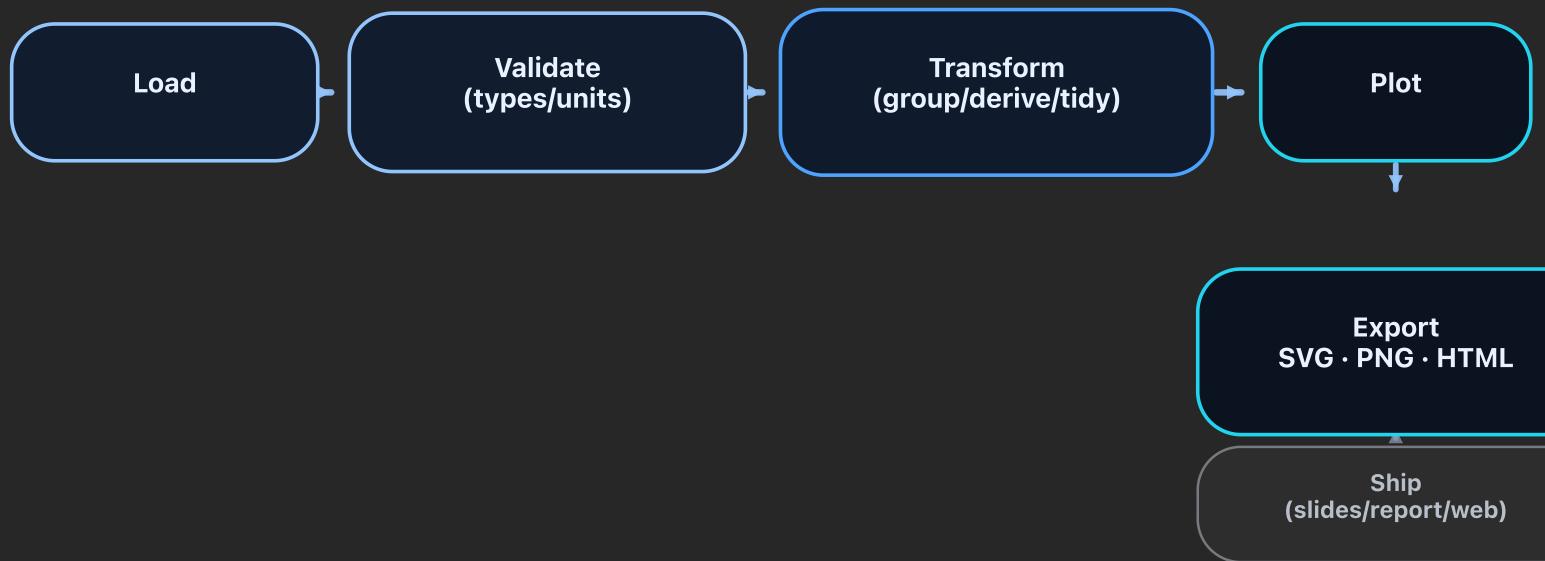
PART 2 · PYTHON WORKFLOW



# Notebook → chart → export

Reproducible visuals you can ship

# A Repeatable Python Visualization Pipeline



# Pandas as the “Chart Data Engine”

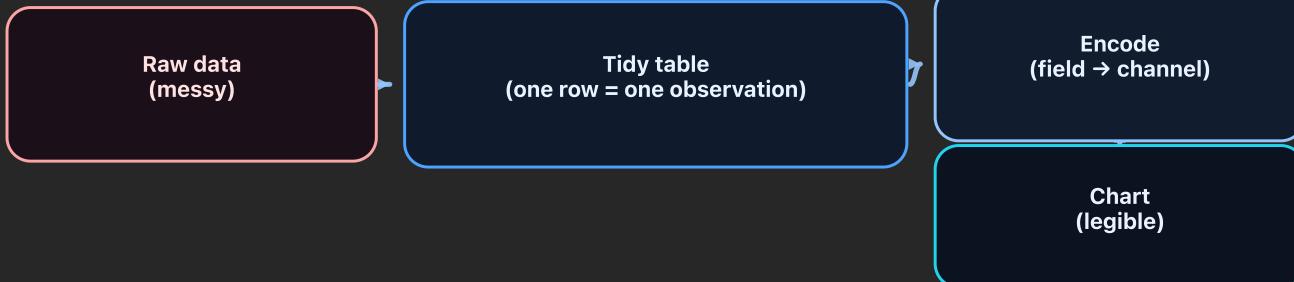
## COMMON TRANSFORMS

- 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.



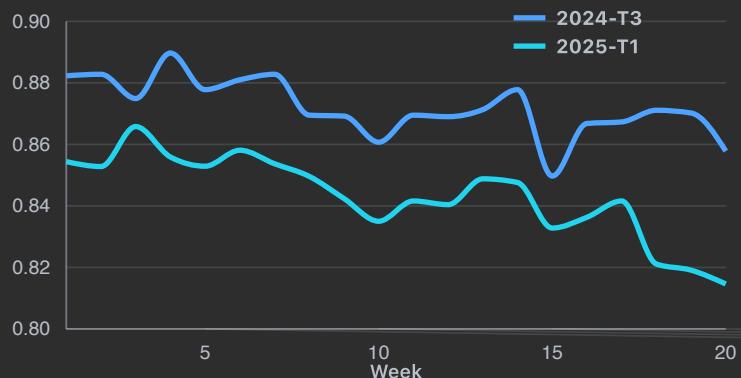
# Concrete D3 examples (class dataset)

## REAL DATA

These charts use the DATA101 class CSV (same dataset used in the guided activity).

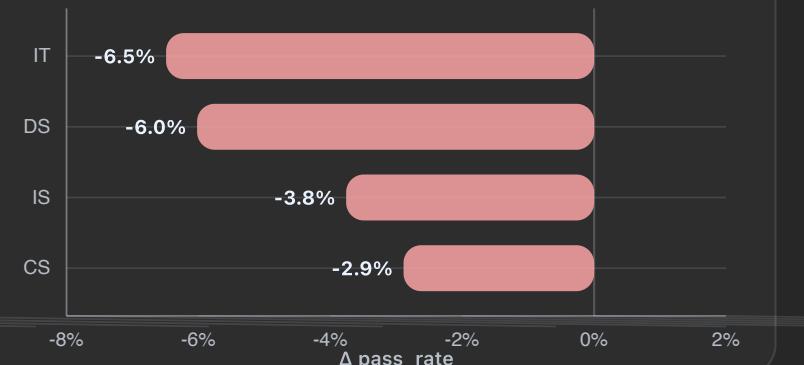
**Pass rate by week (CS)**

DATA101 class dataset



**Change in pass rate by program**

2025-T1 minus 2024-T3



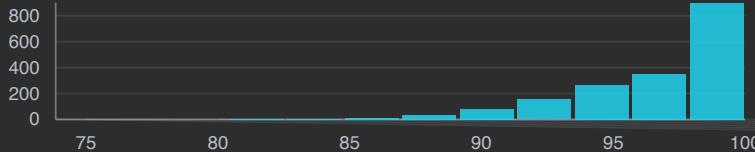
# Concrete D3 example: distributions (real data)

## REAL DISTRIBUTION

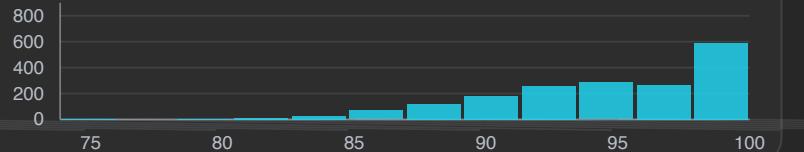
Avg score histograms per program (shared scales, same bins).

Avg score distribution by program (2025-T1)

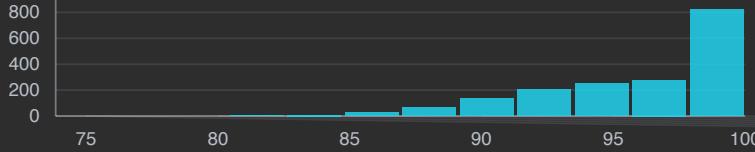
**CS**



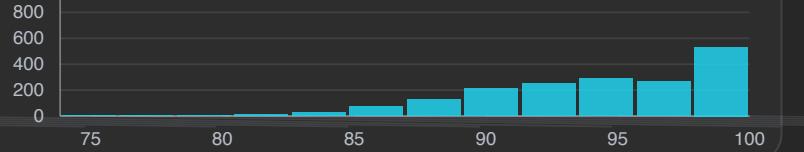
**DS**



**IS**



**IT**



# Demo A: Chart-ready tables (rates + deltas)

## GOAL

### Turn counts into comparable measures

- Counts → rate (pass\_rate)
- Rate over time → delta (change)
- One row = one observation (tidy)

## PROFESSIONAL HABIT

### Write the table before the chart

If the table is wrong, the chart will be wrong—no matter how polished it looks.

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

# Code Demo A — Run

```
import pandas as pd

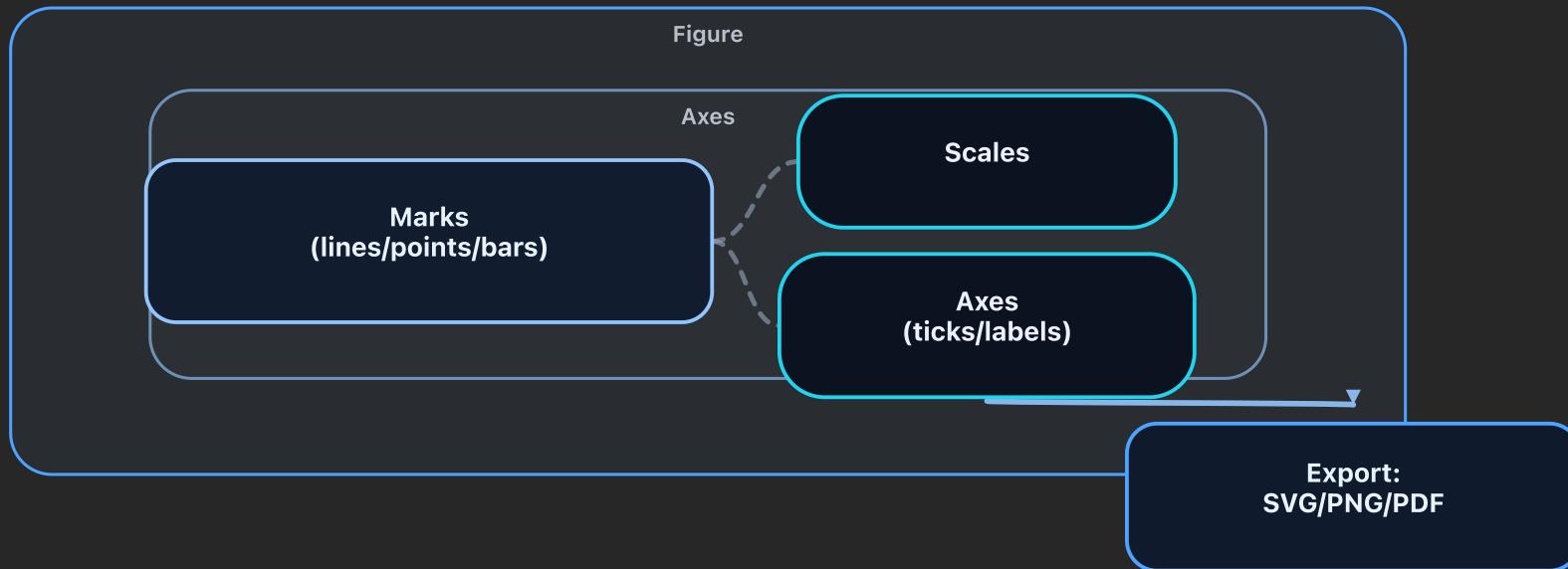
df = pd.DataFrame(
    {
        "program": ["A", "A", "B", "B", "C", "C"],
        "week": [1, 2, 1, 2, 1, 2],
        "n_pass": [70, 62, 90, 88, 40, 44],
        "n_students": [100, 100, 120, 120, 50, 50],
    }
)

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

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

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

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



# Demo B: Matplotlib → SVG export (web-ready)

## WHY SVG

### Crisp + editable + searchable

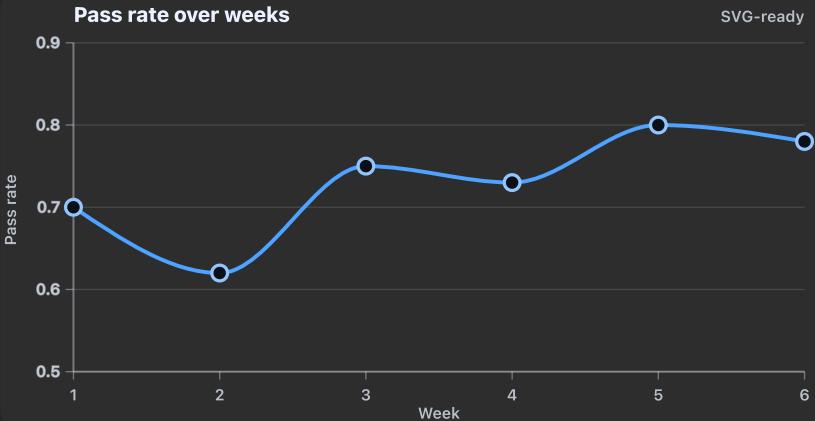
- Perfect for slides + PDF export
- Shapes editable in Figma/Illustrator
- Text stays selectable (accessibility + search)

## HABIT

**Export intentionally (don't screenshot)**

RENDERED RESULT (ILLUSTRATION)

Your SVG will scale crisply in slides.



# Code Demo B – Run

```
import io

import matplotlib.pyplot as plt
import numpy as np

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))
print("Starts with:", svg.splitlines()[0][:72] + "...")
```

# 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.

Distributions

Seaborn

Category  
comparisons

Small multiples  
(FacetGrid)

# Code Demo C: Faceted Histograms

RUNNABLE NEXT SLIDE

## Generate data + summarize spread

We'll compute stats and histogram bin counts (what a faceted histogram is actually drawing).

### WHAT TO LOOK FOR

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

# Code Demo C – Run

```
import numpy as np
import pandas as pd

rng = np.random.default_rng(7)
program = np.repeat(list("ABC"), 250)
score = np.concatenate(
    [
        rng.normal(76, 6, 250),
        rng.normal(76, 14, 250),
        rng.normal(70, 8, 250),
    ]
)
df = pd.DataFrame({"program": program, "score": score})

summary = df.groupby("program")["score"].agg(["mean", "std", "min", "max"]).round(2)
print("Summary stats (mean can be similar while spread differs):")
print(summary.to_string())

bins = np.linspace(df["score"].min(), df["score"].max(), 19)
counts = (
    df.assign(bin=pd.cut(df["score"], bins=bins, include_lowest=True))
    .groupby(["program", "bin"])
    .size()
```

# 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

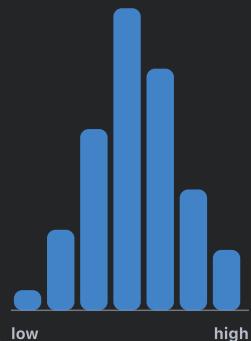
## WHY THIS WORKS

Small multiples outsource less work to the viewer: same axes, same units, clean comparison.

### Score distributions (small multiples)

Aligned panels reveal spread differences

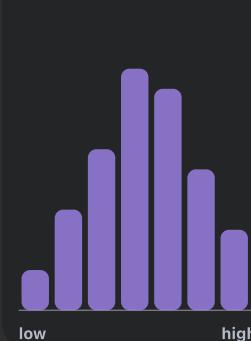
Program A



Program B



Program C



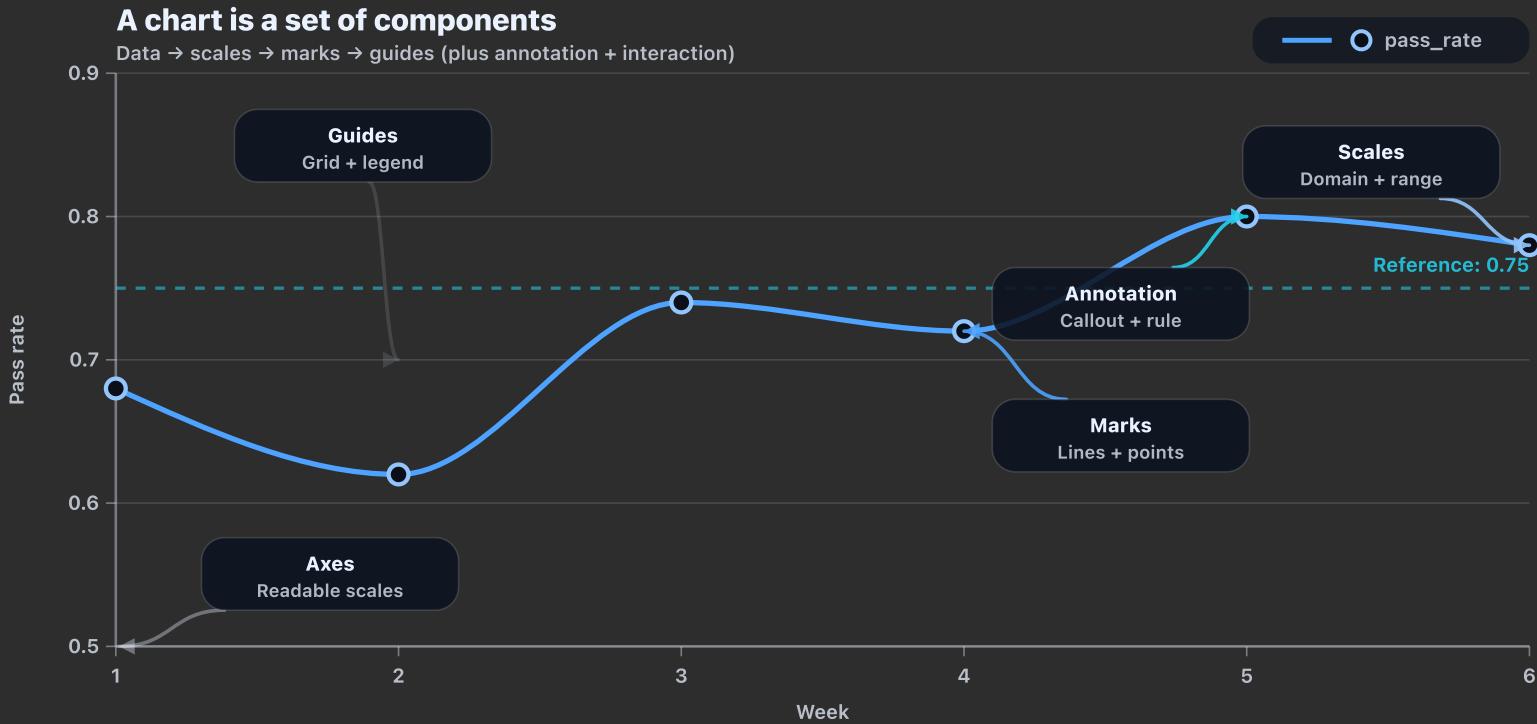
PART 3 · MODERN COMPONENTS



# 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

GRAMMAR-STYLE VISUALIZATION

## Field → channel mappings

A spec is a contract between data and marks. You can critique it before any pixels are rendered.

### WHAT TO WATCH

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

# Code Demo D — Run

```
import json

import pandas as pd

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", "scale": {"domain": [0.5, 0.9]}},
        "color": {"field": "term", "type": "nominal"},
        "tooltip": [{"field": "x"}, {"field": "y"}, {"field": "term"}],
    },
}

print("First 3 rows:")
```

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

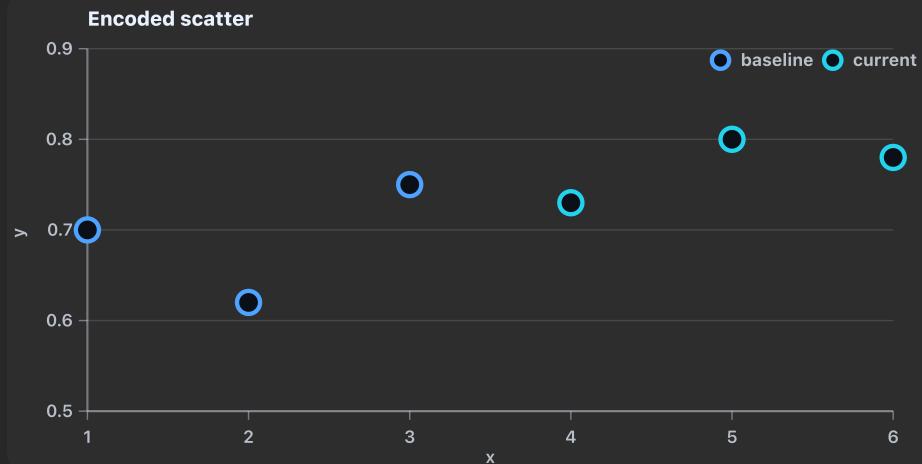
### DESIGN NOTE

#### Encoding is a contract

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

### RENDERED RESULT

Legend + axes carry the structure; tooltip adds detail without clutter.



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)

**Selection**  
(click a group)

**Brushing**  
(select a range)

**Filtering**  
(reduce space)

Compare a focus group  
vs others

Focus on a time window  
or region

Handle many categories  
or large data

# Demo E: Interactive HTML artifacts

WHEN TO USE HTML

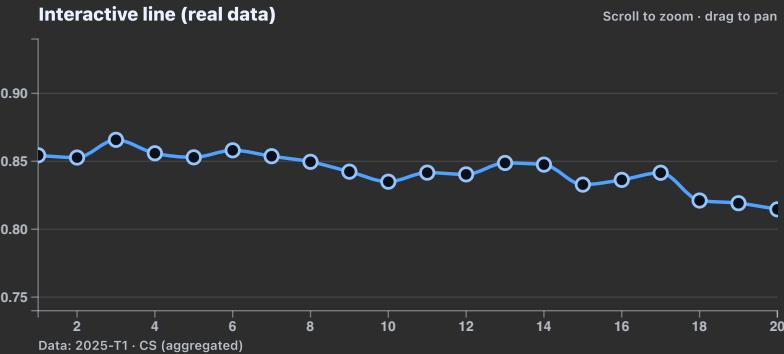
**Interactivity supports a task**

- Hover to confirm exact values
- Zoom/pan for dense time series
- Highlight/filter to compare groups

RENDERED RESULT (ILLUSTRATION)

Hover + zoom are built in.

Interactive line (real data)



DELIVERABLE

**A single HTML file you can share**

# Code Demo E – Run

```
import pandas as pd

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

width, height = 720, 320
pad_l, pad_r, pad_t, pad_b = 56, 18, 22, 44

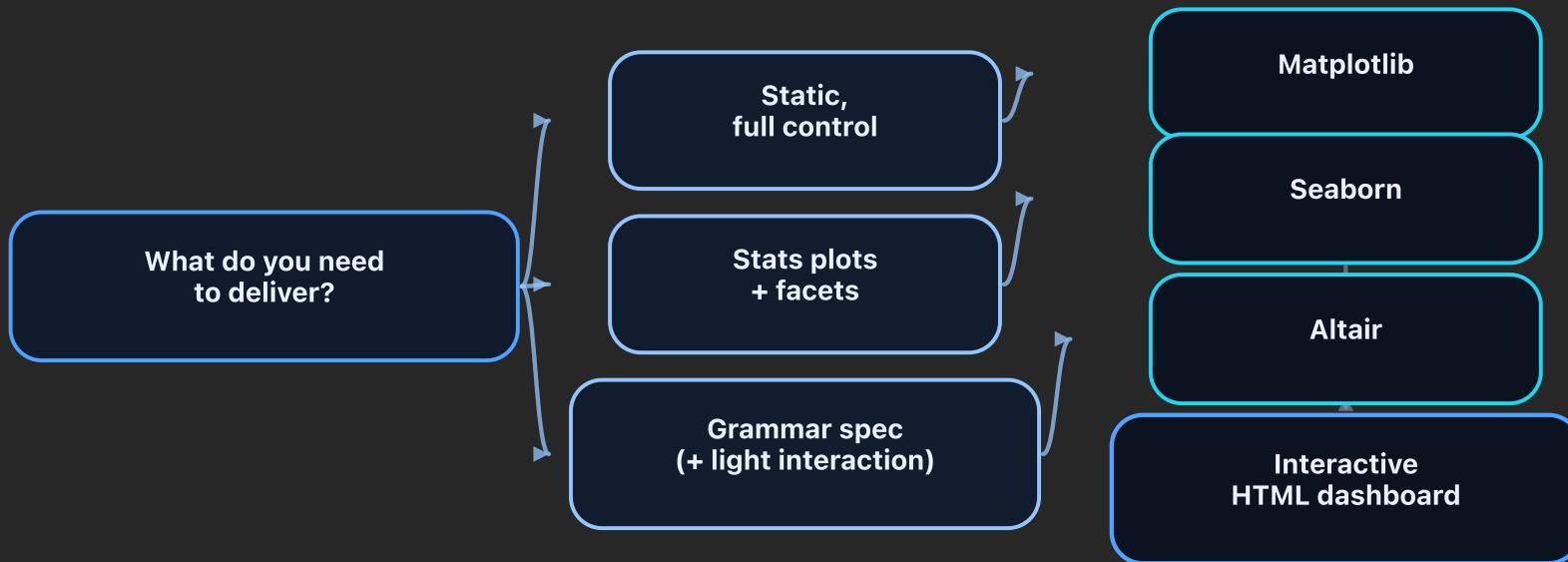
xmin, xmax = df["week"].min(), df["week"].max()
ymin, ymax = 0.5, 0.9

def sx(x: float) -> float:
    return pad_l + (x - xmin) / (xmax - xmin) * (width - pad_l - pad_r)

def sy(y: float) -> float:
    return pad_t + (1 - (y - ymin)) / (ymax - ymin)) * (height - pad_t - pad_b)

pts = [(sx(r.week), sy(r.pass_rate)) for r in df.itertuples(index=False)]
path = "M " + " L ".join(f"{x:.1f},{y:.1f}" for x, y in pts)
```

# 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

# Bonus: Animations (when they help)

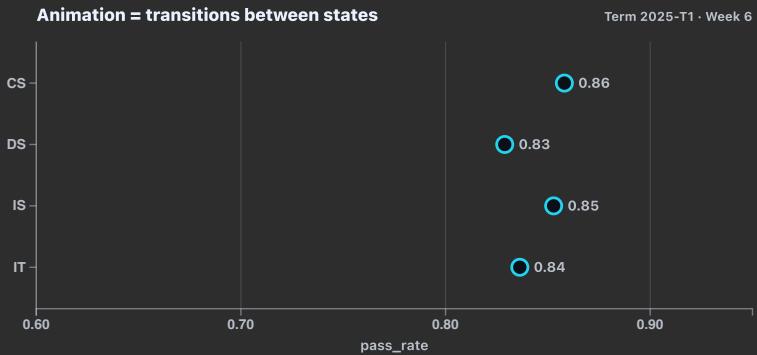
D3 DEMO (AUTO-ADVANCING)

Same data, same scale — watch the marks move.

## USE ANIMATION WHEN

- The task is to **see change over time**
- You keep a **fixed scale** (so frames are comparable)
- You also provide a **static alternative** for precise comparison

## Animation = transitions between states



## PYTHON OPTION

### Plotly animations

A single HTML file with frames + controls.

## 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.

1

**Pick a simple dataset**

10–200 rows.

2

**Include units + baseline**

If relevant.

3

**Justify your export choices**

Write 2–3 sentences.

PART 5 · PRACTICE

# 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

## WHAT YOU'RE BUILDING

### A tiny "style system"

- Figure size is intentional (export-ready)
- Typography is consistent
- Gridlines are subtle

## WHAT TO NOTICE

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

## RULE

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

# Live Python – Run

```
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)
fig.tight_layout()

buf = io.StringIO()
fig.savefig(buf, format="svg")
svg = buf.getvalue()
print("SVG length:", len(svg))
print("Starts with:", svg.splitlines()[0][:72] + "...")
```

# Color is a data encoding (not a theme)

MATCH MEANING

Type → palette

Categorical, ordered magnitude, baseline differences.

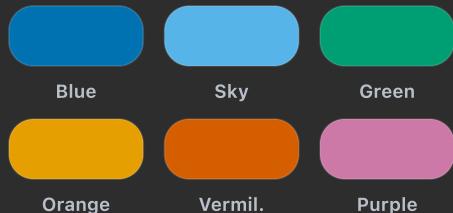
COMMON FAILURE

Pretty ≠ interpretable

If the scale is wrong, the chart is wrong.

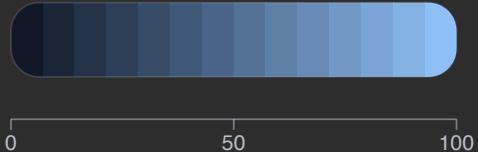
## Qualitative

Categories



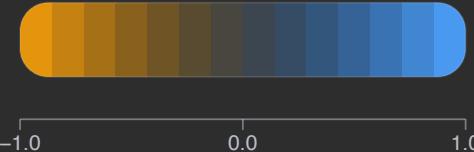
## Sequential

Low → High



## Diverging

Below ↔ Above



# Matplotlib colormaps: choose by semantics

## SEQUENTIAL

**Magnitude: low → high**

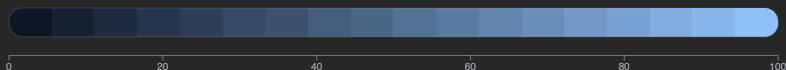
Use lightness ramps so order is visible.

## DIVERGING

**Difference: below ↔ above baseline**

Only when the midpoint is meaningful.

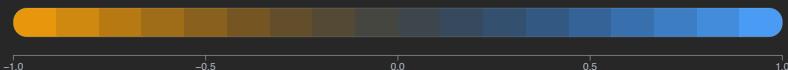
Lightness ramp (ordered magnitude)



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

## CONCEPT

### Colormaps are functions

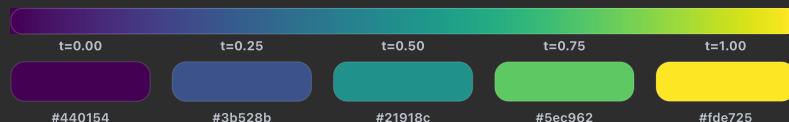
You can sample them, test them, and keep them consistent across many charts.

## WHY THIS MATTERS

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

### Colormap = function ( $t \rightarrow \text{color}$ )

Example: Viridis (sequential, order visible via lightness)



Tip: choose palettes by semantics (categorical vs sequential vs diverging), then sample consistently.

# Live Python – Run

```
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))
```

# Accessibility basics for color

DO

## Add redundancy

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

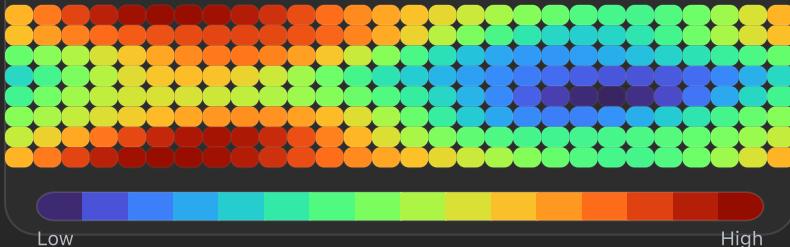
AVOID

## Red/green-only meaning

Many viewers cannot reliably distinguish it.

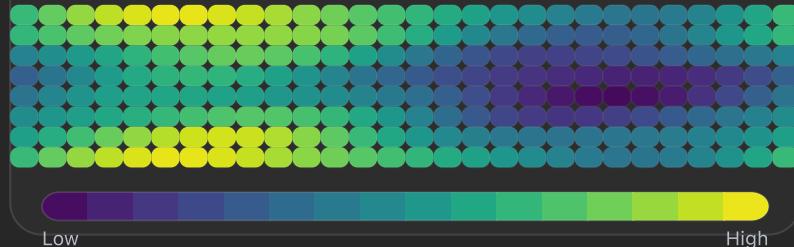
### Rainbow-like

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)

## PATTERN

### A “chart component” mindset

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

## HOW TO USE IT

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

# Live Python – Run

```
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", "pass_rate")))
```

# Layout matters (even in notebooks)

GOOD LAYOUT

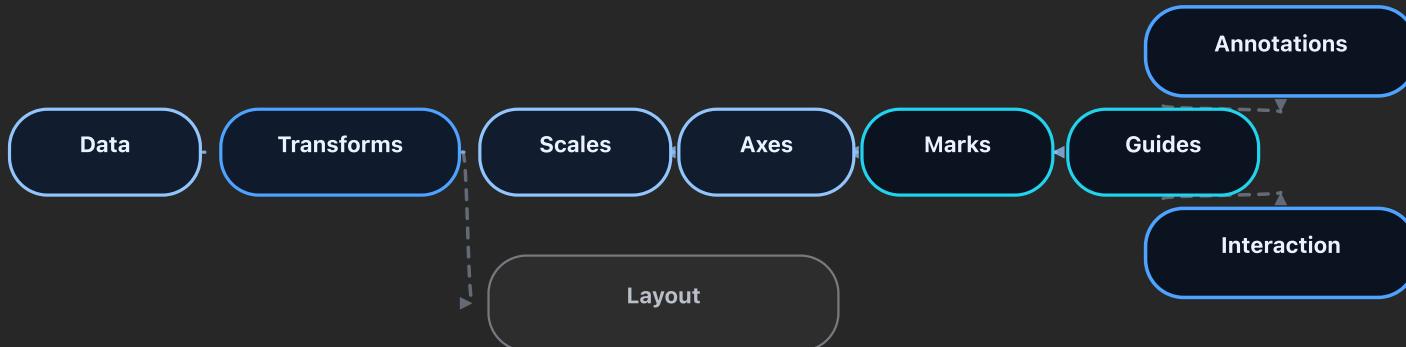
## Aligned comparisons

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

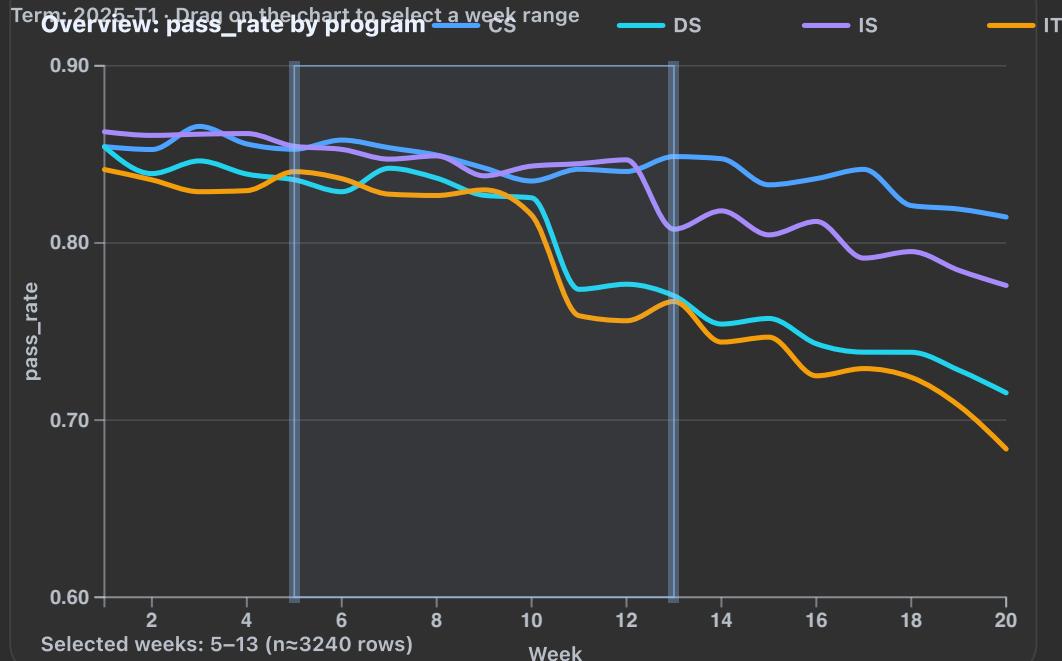
## WHY THIS PATTERN WORKS

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

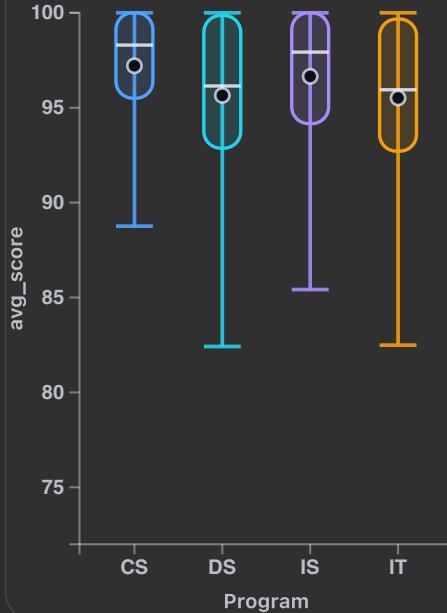


# Case study (interactive demo): brush → distribution

## Case study (real data): overview → brush → distribution



## Details: avg\_score distribution



# 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

**SVG/HTML are common “final forms”**

Export for the medium you ship to.

## PYTHON WORKFLOW

**Data → transforms → chart → export**

Re-run it tomorrow and get the same result.

## MODERN COMPONENTS

**Scales, marks, axes, guides, interaction**

Name the parts to design and debug faster.

## INTERACTIVITY

**Only “professional” when it supports a task**

Tooltips confirm; charts should still read.

## 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>