Data Visualization

Agenda

1. Color Vision Deficiency (CVD)
2. Python Matplotlib
3. Matplotlib Example

Color Vision Deficiency (CVD)

Color Vision Deficiency (CVD)

- Persons with **CVD**:
  - colors most people see as different will look the same for them

- **Colorblindness** is not the most accurate term
  - instead, use **CVD**

CVD Studies

- **Red-green CVD**
  - About **8% of men**
    - 6% of men have deuteranomaly (green-weak) & deuteranopia (green-blind)
    - 2% of men have protanomaly (mild) & protanopia (severe)
  - About **0.5% of women**

- **Blue-yellow CVD**
  - About **5% of all CVD cases**

- **CVD** doesn't mean: person can't see color
  - unless in very rare cases (1 in 33,000)

CVD Commonly Referred to as

- red weak
- green weak
- red-green colorblindness

Data-viz Rule

Don’t use red & green together

CVD-friendly Designing Tips

1. Red and green together can be problematic.

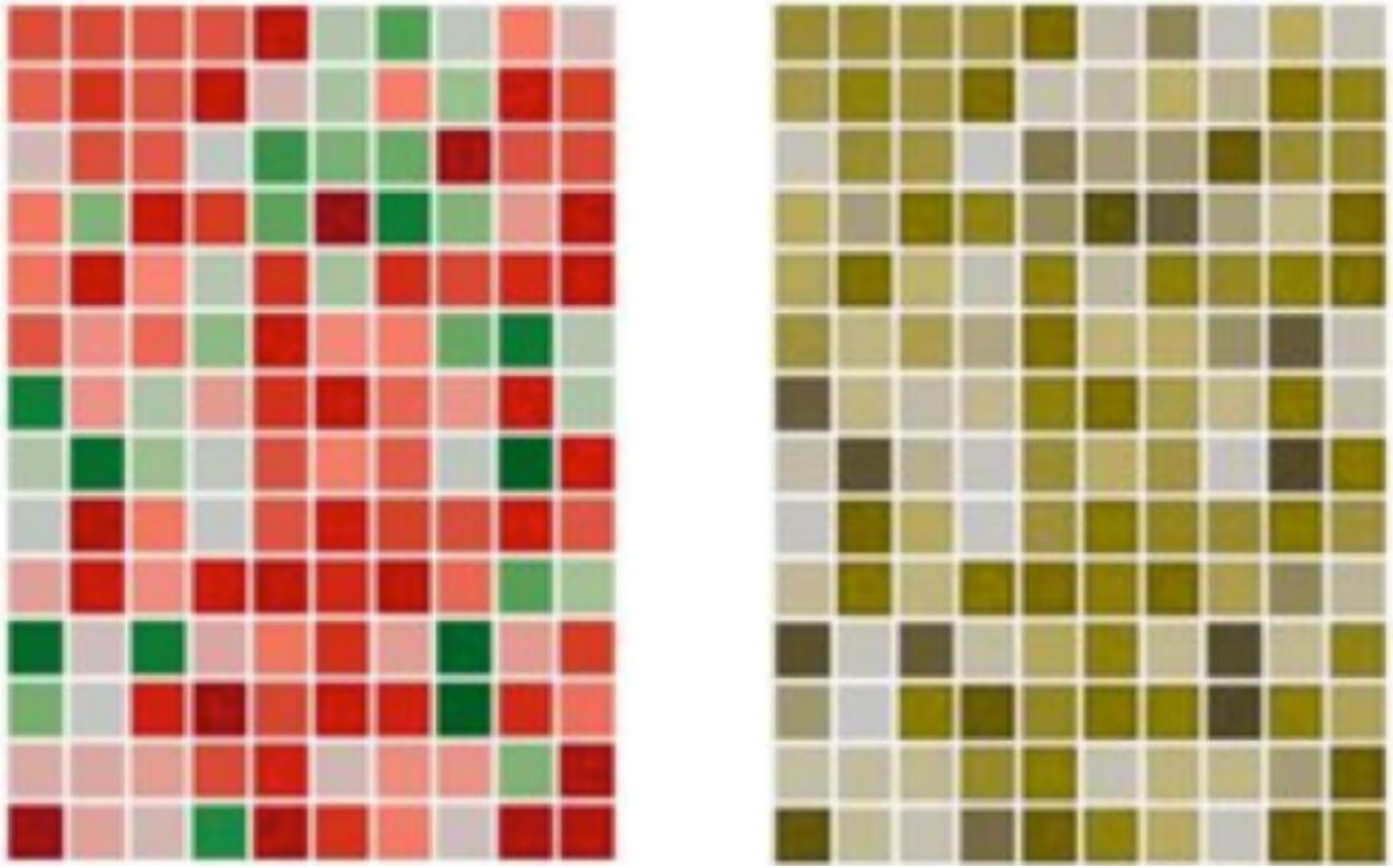
2. Be aware that it’s not just red and green.

3. Use a CVD-friendly palette when appropriate.

4. If you must use red and green together, you can:
    a. leverage light vs. dark
    b. stand each color (red and green) alone
    c. offer alternate methods of distinguishing data
    d. use a checkbox (or similar GUI) to switch the color palette to a CVD-friendly palette

Tip 1) CVD Simulation Example

- a good number vs. a bad number in a table
- one line vs. another line in the same line chart
- color is needed to tell a good square from a bad square
  - we can see how difficult this would be in the chart



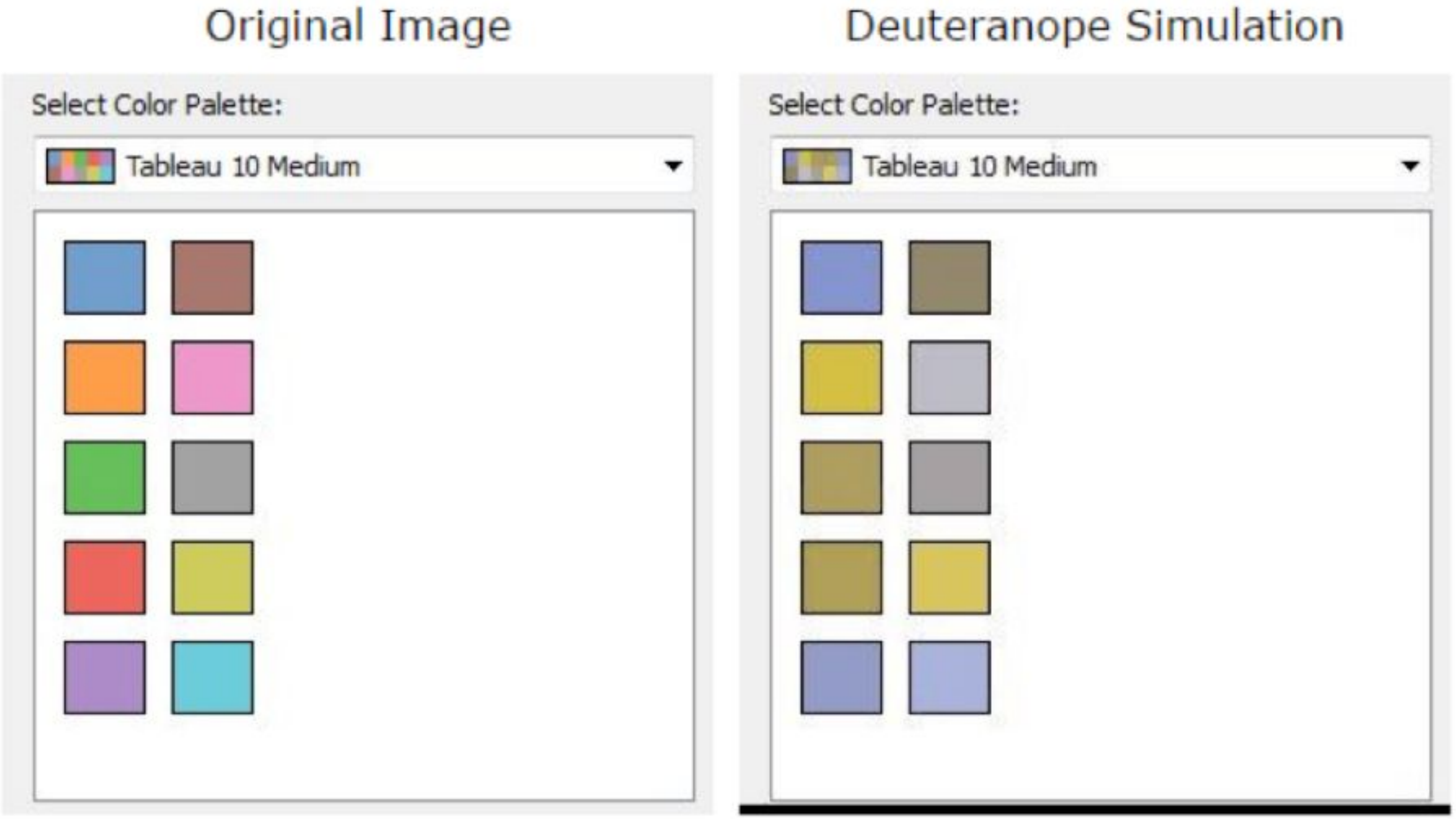
Tip 2) More Complex Than Red vs. Green

- For someone with **strong CVD**
  - **red** & **green** & **orange** all can appear **brown**
  - Maybe more accurate to say: Don't use **red** & **green** & **brown** & **orange** together
    - In the RGB model: orange is RGB(255,165,0) & brown is RGB(150, 75, 0)

- Also, when mixing colors, they can be problematic.
  - **Example**: using **blue** & **purple** together
    - In the RGB model, **purple** is RGB(160,32,240)
    - If someone has issues with **red**, they may have issues with **purple** (appear **blue**)

- Also, **pink** & **gray** or **gray** & **brown** can be problematic.

Tip 2) Deuteranope Simulation



Original Image

Select Color Palette:

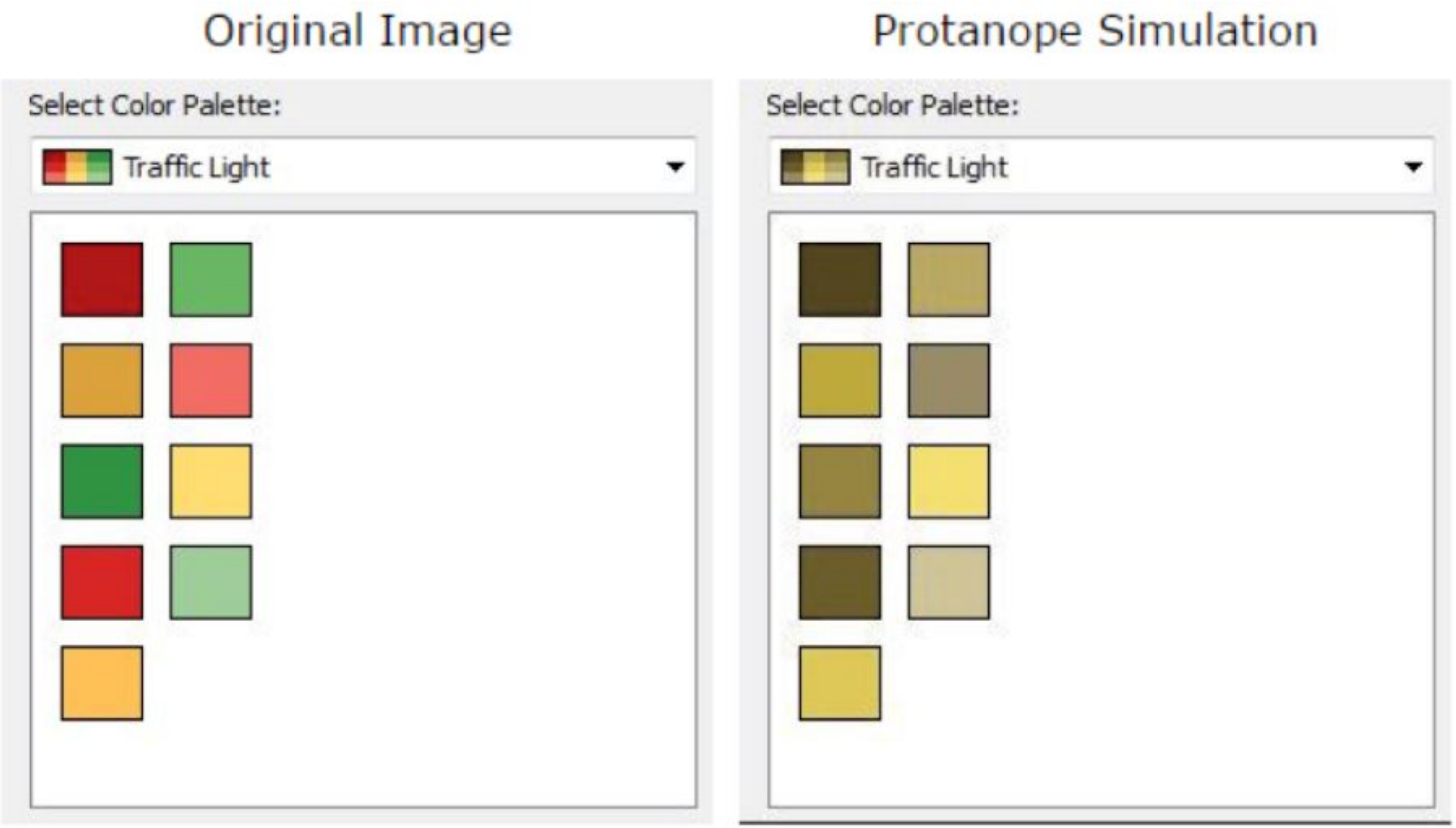
Tableau 10 Medium

Deuteranope Simulation

Select Color Palette:

Tableau 10 Medium

Tip 2) Protanope Simulation



Original Image

Select Color Palette: Traffic Light

Protanope Simulation

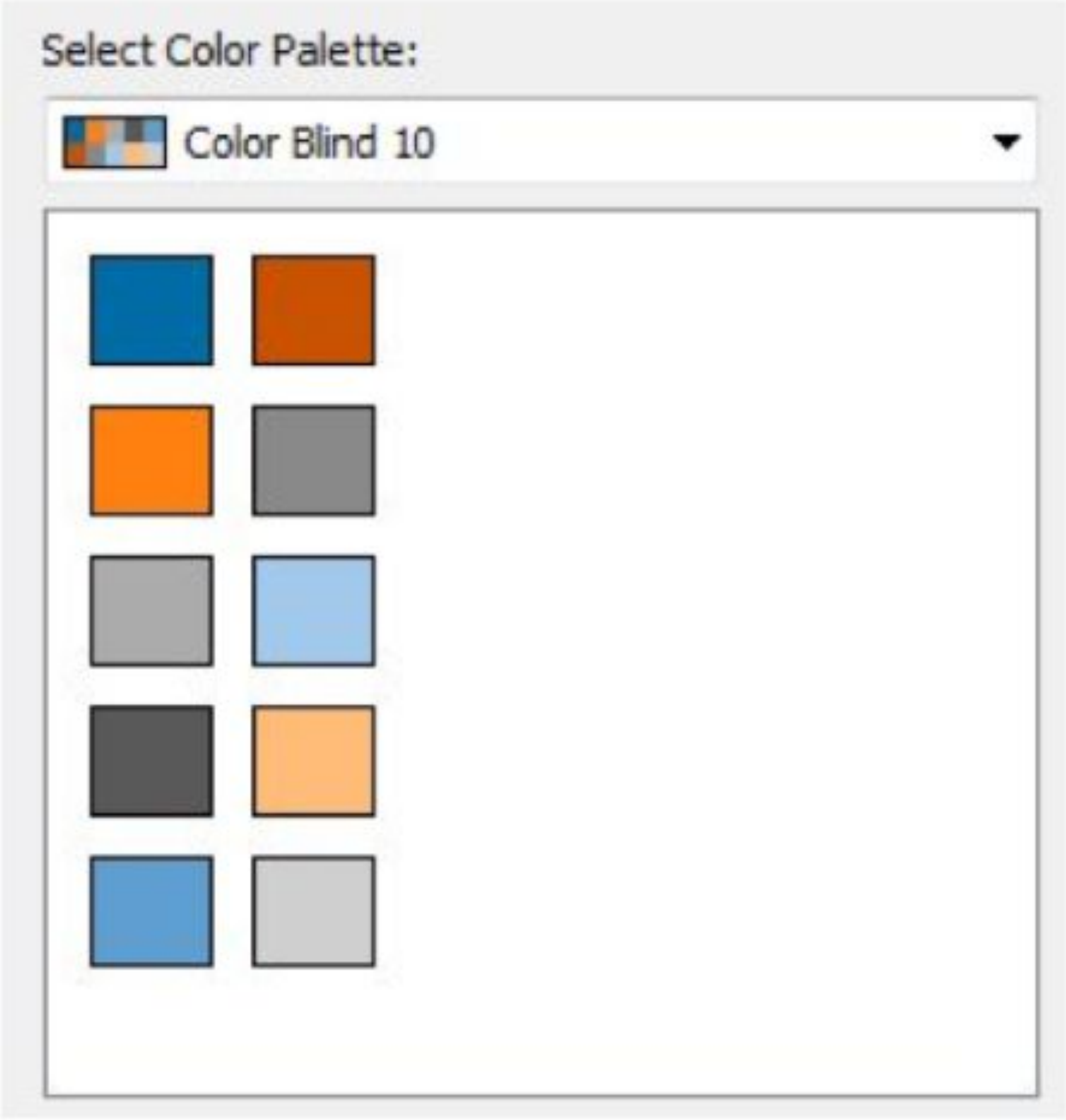
Select Color Palette: Traffic Light

Tip 3) CVD-friendly Palette

- One color combined with another color is generally fine
  - when one of them is not usually associated with CVD
- For the most common conditions of CVD
  - **blue** would generally look **blue**
  - **Examples:**
    - **blue/orange** is a common CVD-friendly palette
    - **blue/red** or **blue/brown** would also work

Tip 3) Deuteranope-friendly Palette

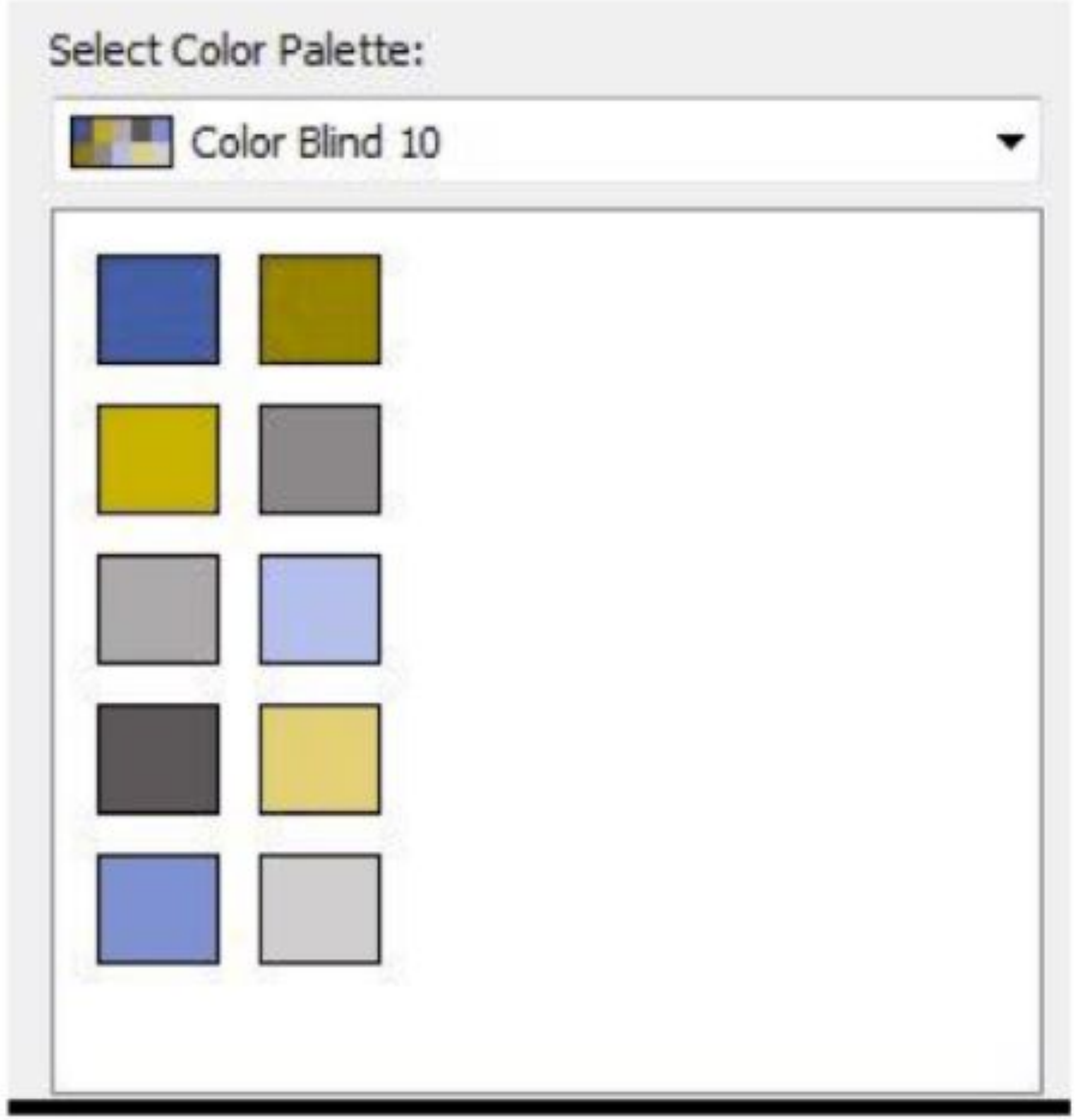
Original Image



Select Color Palette:

Color Blind 10

Deuteranope Simulation

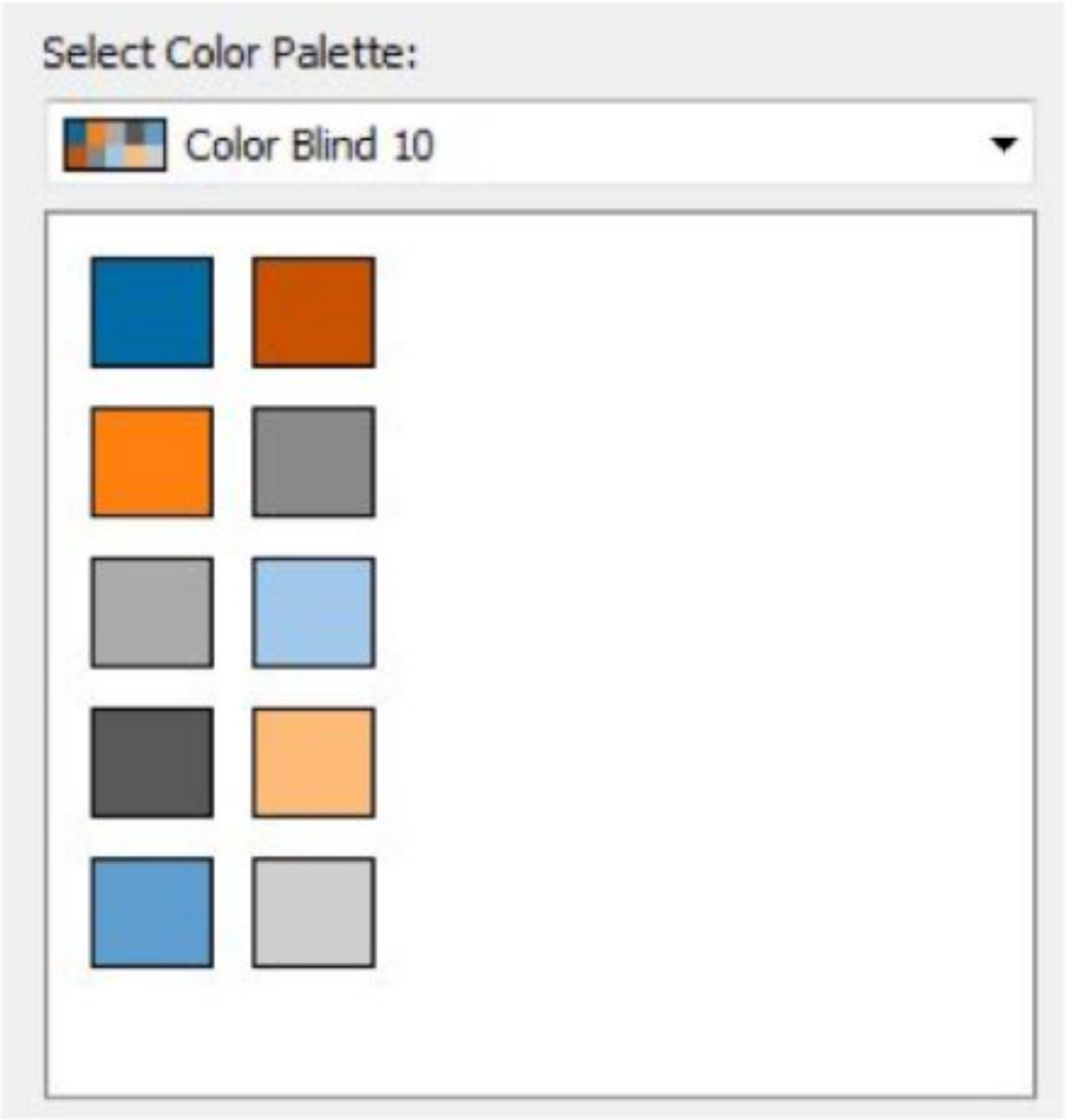


Select Color Palette:

Color Blind 10

Tip 3) Protanope-friendly Palette

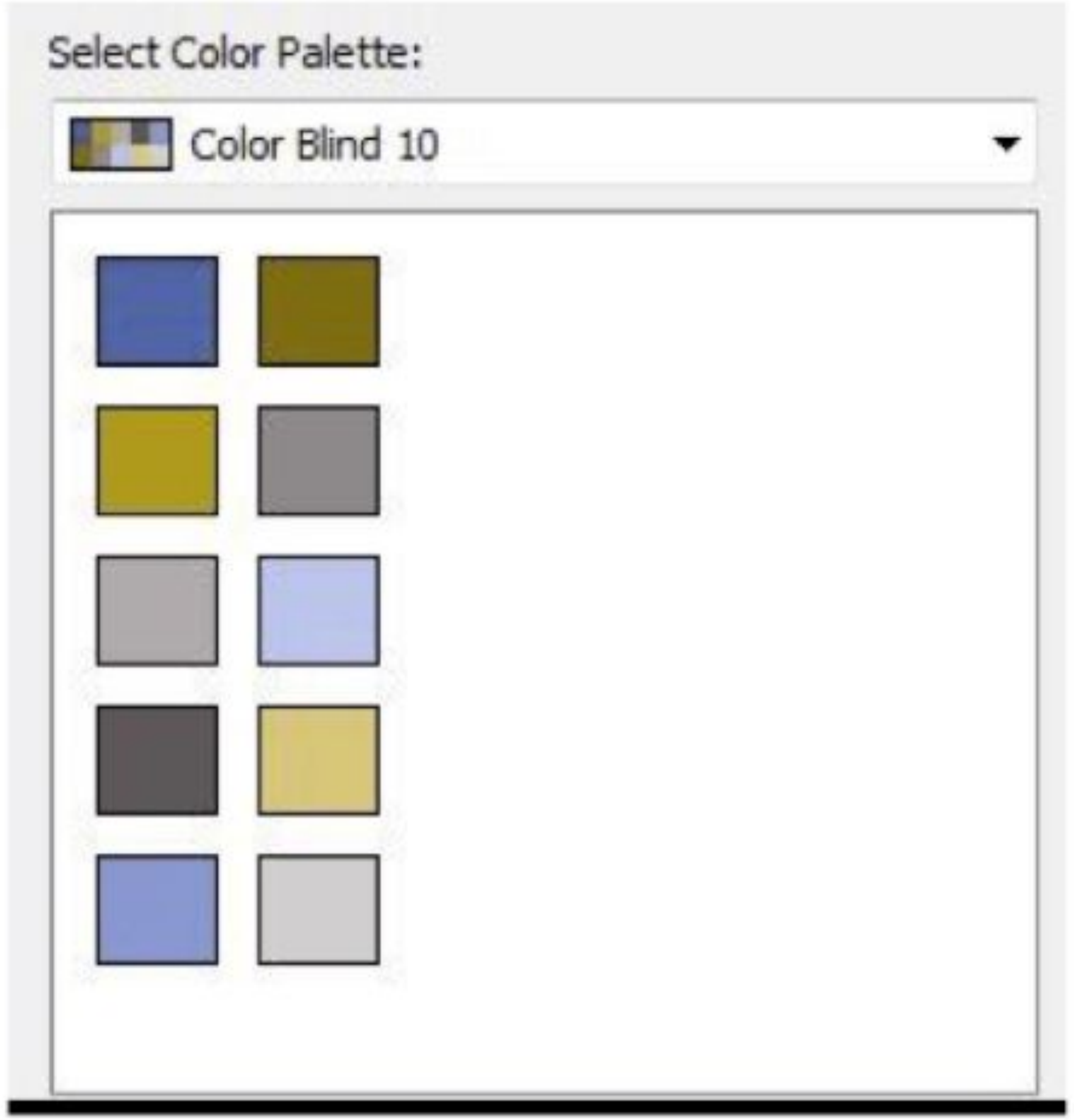
Original Image



Select Color Palette:

Color Blind 10

Protanope Simulation



Select Color Palette:

Color Blind 10

Tip 4-a) Leverage Light vs. Dark

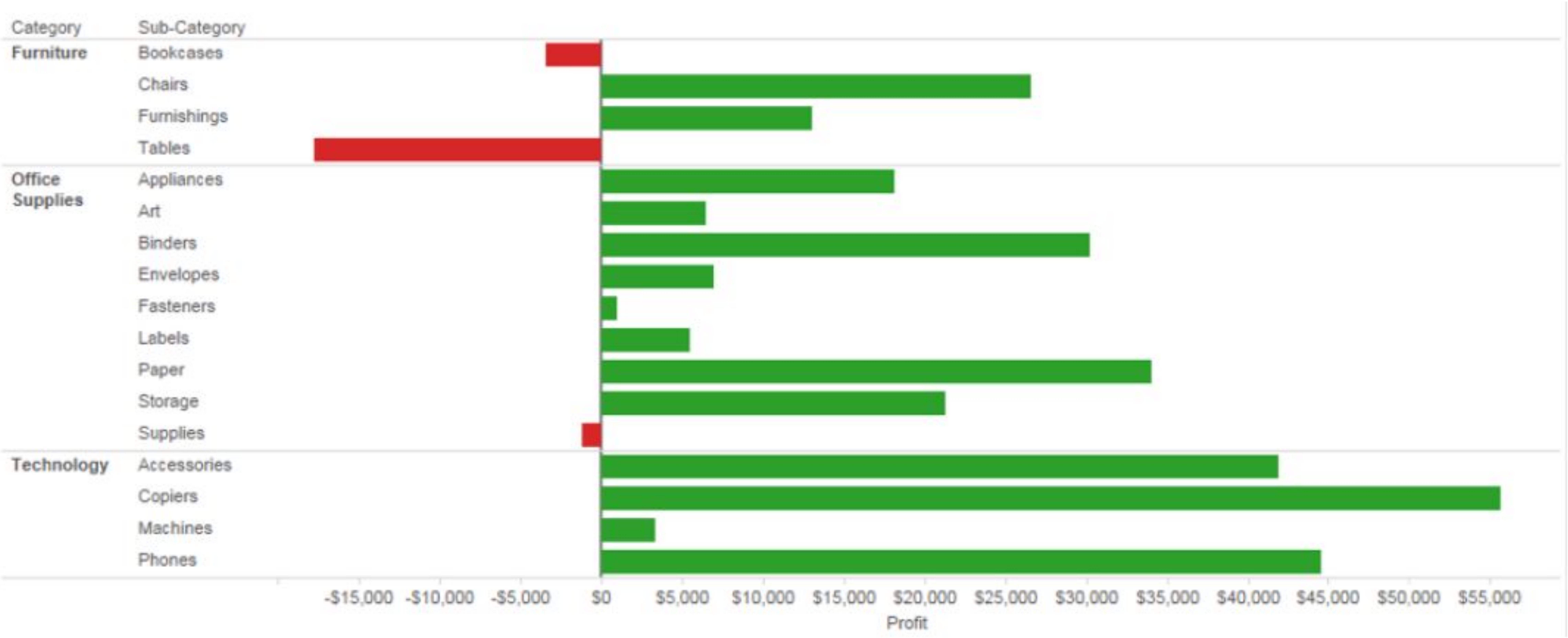
- The problem with **CVD** is **red vs. green** and not **light vs. dark**.

- Almost anyone can tell the difference between:
  - **very light color** and **very dark color**

- To use red and green together, we can use:
  - **light green**
  - **medium yellow**
  - **very dark red**

- Someone who has strong **CVD:**
  - would see as a **sequential color scheme**
  - would at least be able to distinguish based on **light vs. dark**

Tip 4-b) Stand Each Color Alone

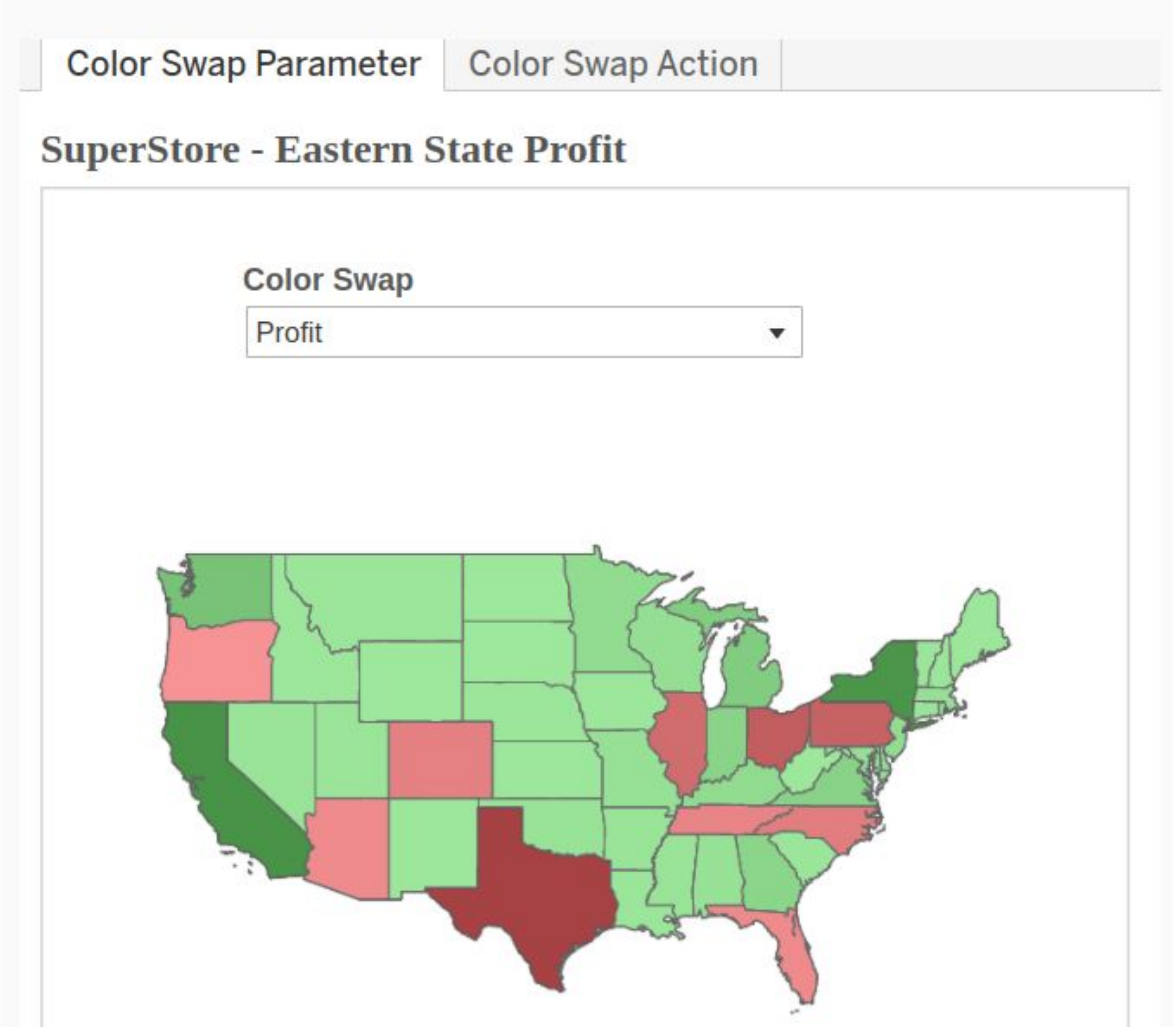


Tip 4-c) Alternate Distinguishing Methods

- Add indicators to allow to see that something is bad (red) vs. good (green), such as:
    - icons
    - directional arrows
    - labels
    - annotations
    - other indicators

Tip 4-d) Use a UI element to Switch Color Palette

- Use a checkbox (or similar GUI) to switch the color palette to **CVD-friendly Palette**



SuperStore - Eastern State Profit

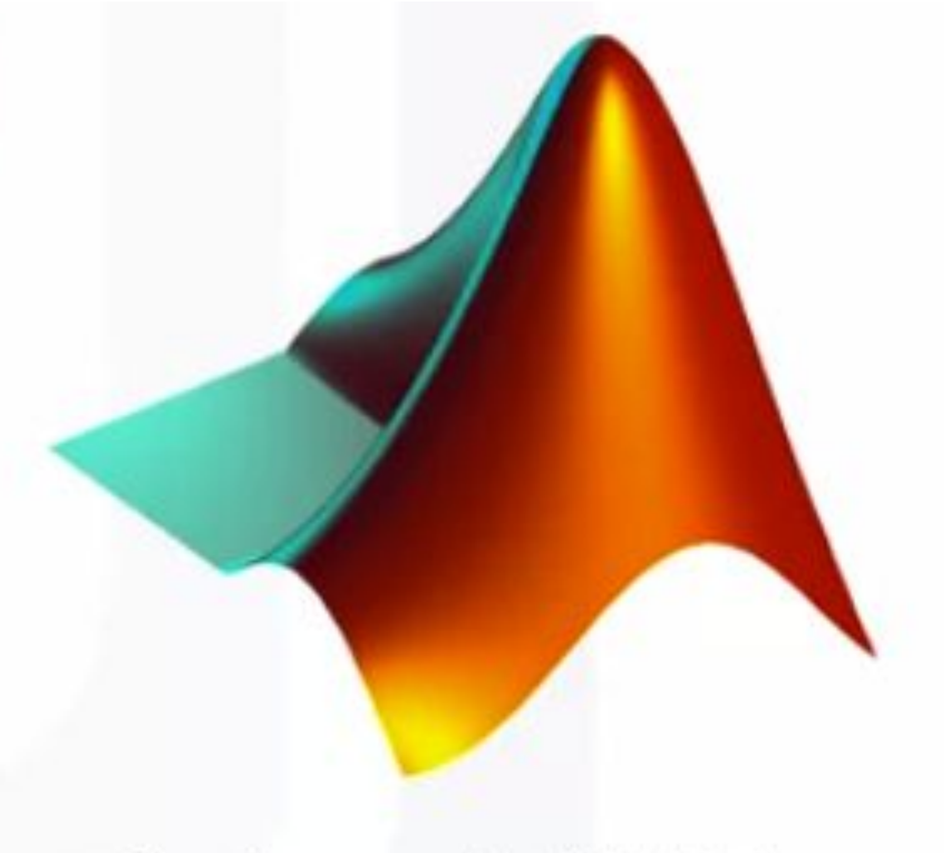
Color Swap: Profit

Python
Matplotlib

John Hunter (Matplotlib Creator)

- Neurobiologist
- Part of a team analyzing **Electrocorticography Signals (ECoG)**
  - **Electrocorticography** is the process of recording electrical activity in the brain

- The team
  - used a proprietary software (**MATLAB** based version) for analysis
  - had only one license and were taking turns in using it



- **John** replace the proprietary software with **Matplotlib**

Python Matplotlib

- MatLab-style Plotting Library
- Created in 2002
- Most popular data visualization library in Python
- Well supported in different environments
  - Python scripts & iPython shell & web app servers & **Jupyter Notebook**
- Originally developed as an **ECoG** visualization tool

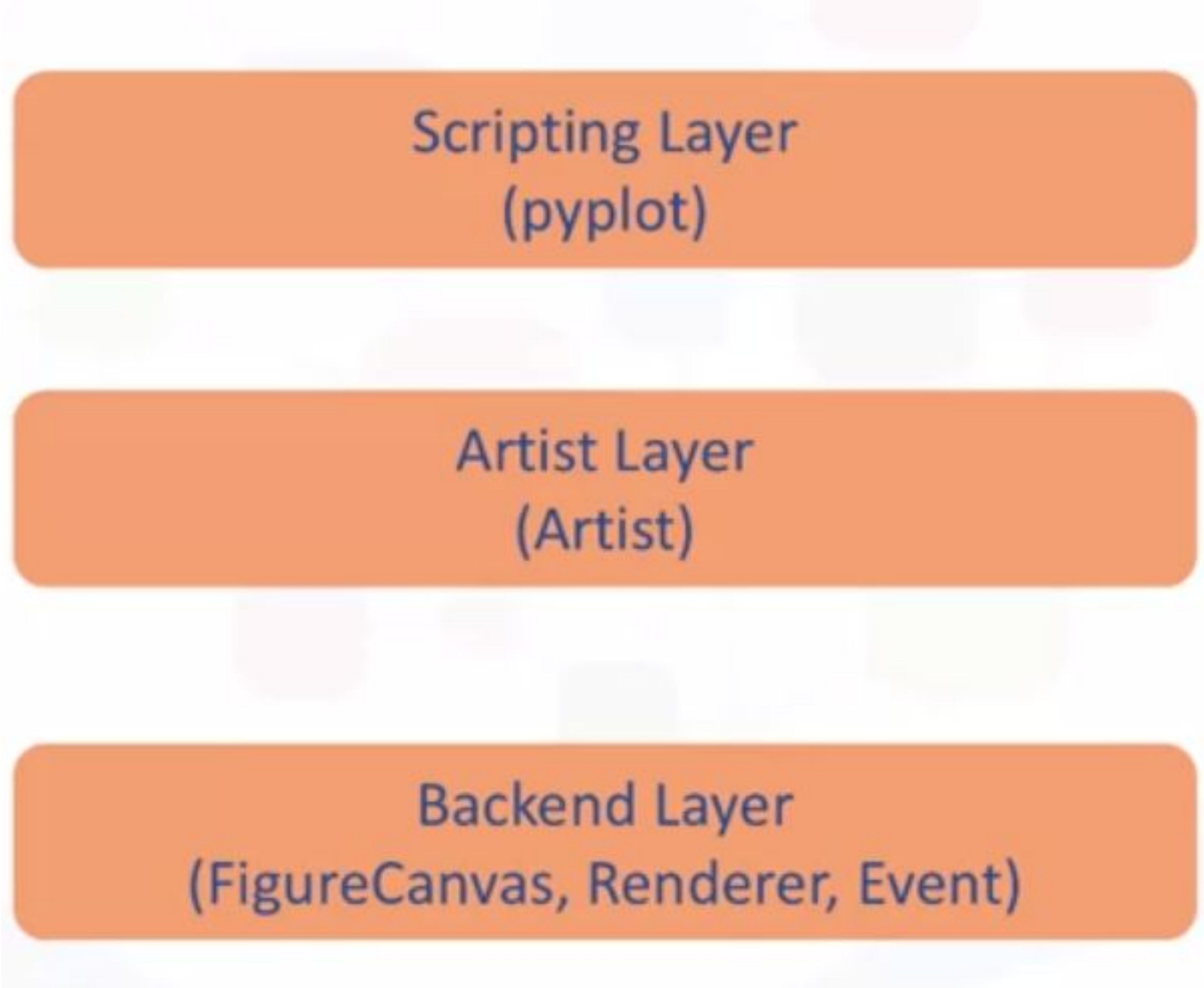
Jupyter Notebook

- open source web app
- allows to create & share documents that contain code and text
- spun off from **iPython** in **2014**

- **Jupyter name** is a reference to three programming languages:
  - **Julia**
  - **Python**
  - **R**
- **Jupyter logo**
  - homage to **Galileo**'s discovery of the **moons of Jupiter**
  - documented in **notebooks** attributed to **Galileo**

Matplotlib Architecture

1. **Back-end Layer**
2. **Artist Layer**
    - appropriate programming paradigm for
        - web app server
        - UI app
        - script to be shared with others
3. **Scripting Layer** (idea from MATLAB)
    - appropriate layer for everyday purposes
    - lighter interface to simplify common tasks
    - for a quick and easy generation of plots



Scripting Layer
(pyplot)

Artist Layer
(Artist)

Backend Layer
(FigureCanvas, Renderer, Event)

Matplotlib Architecture: 1) Back-end Layer

has built-in classes, such as:

1. **FigureCanvas**
   - defines and encompasses the area into which the figure is drawn

2. **Renderer**
   - knows how to draw on the **FigureCanvas**

3. **Event**
   - handles user inputs such as keyboard strokes and mouse clicks

https://github.dev/matplotlib/matplotlib/blob/main/lib/matplotlib/backend_bases.py
https://github.dev/matplotlib/matplotlib/blob/main/lib/matplotlib/backends/backend_agg.py

Matplotlib Architecture: 2) Artist Layer

- Composed of one main object (the **Artist**)

- **Artist**
  - knows how to use the **Renderer** to draw (put ink) on the **FigureCanvas**

- Everything you see on a **Matplotlib figure** is an **Artist instance**
  - **Example**: title, lines, tick labels, images, …
  - all of them correspond to an individual **Artist instance**

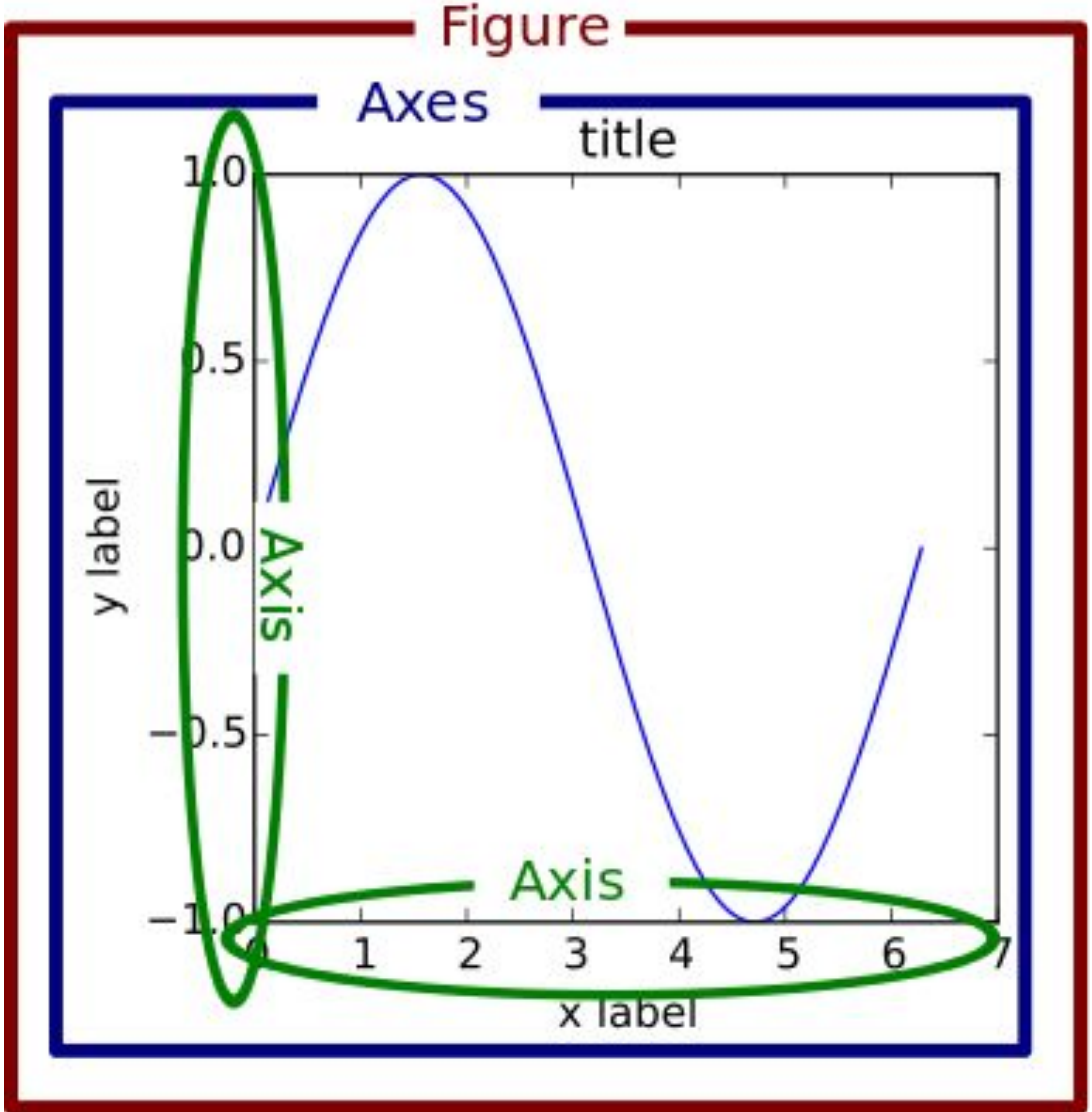
Matplotlib Architecture: 2) Artist Layer Types

1. **Primitive Artist**: as Line, Rectangle, Circle, Text

2. **Composite Artist:** may contain other **Artists**
   - **Example 1: Figure Artist**
     - top-level Matplotlib object
     - contains and manages all of the elements in a given graphic
     - https://github.dev/matplotlib/matplotlib/blob/main/lib/matplotlib/figure.py
   - **Example 2: Axes Artist**
     - most important Composite Artist
     - where most of the Matplotlib API plotting methods are defined
       - including methods to create and manipulate ticks, axis lines, grid, background
   - **Other Examples: Tick Artist**

Axes

- The plotting area
- including all axis
- Don’t mean plural of **Axis**



Matplotlib Architecture: 3) Scripting Layer

- Developed for scientists who are not professional programmers

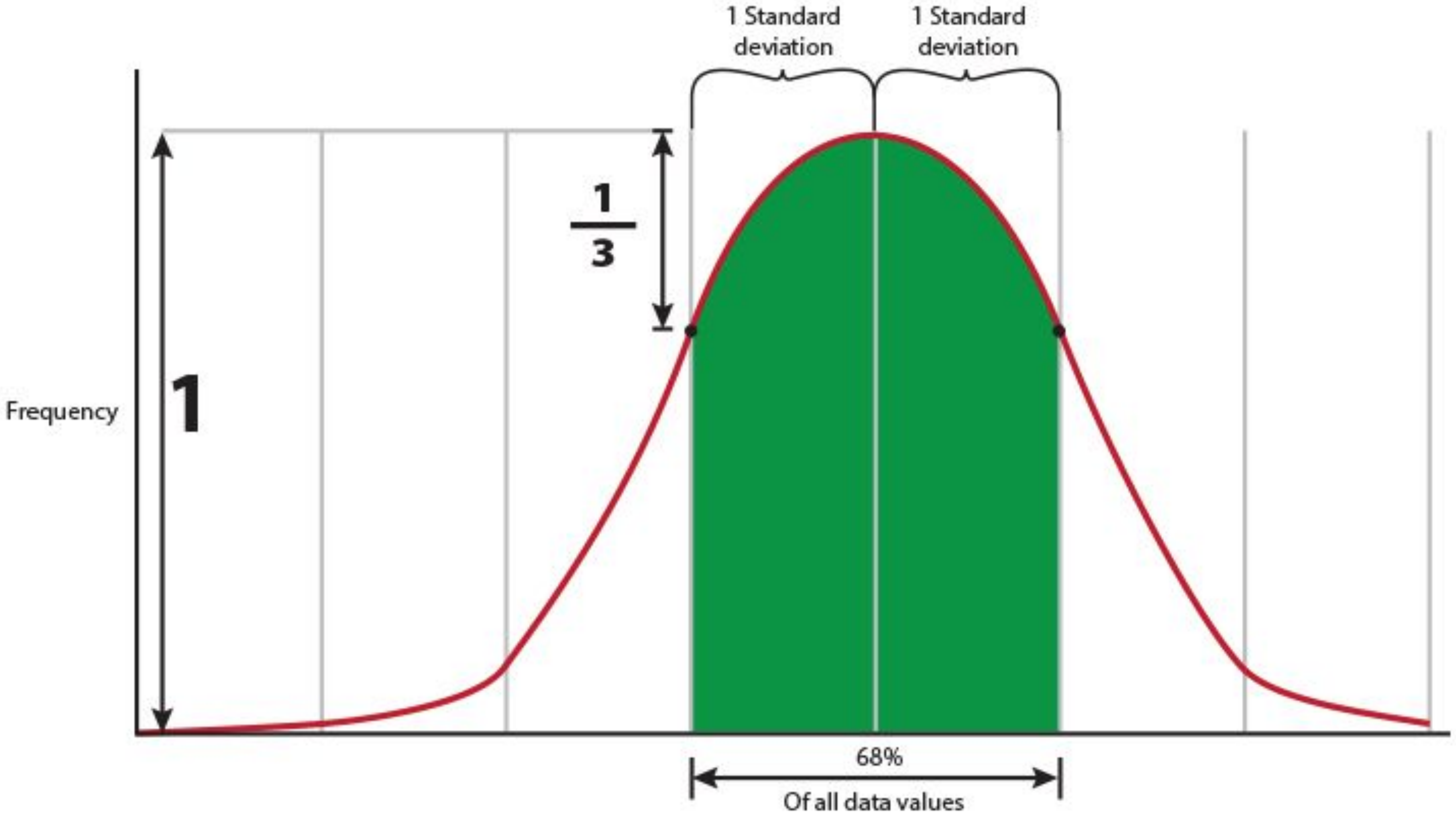
- Essentially the **Matplotlib.pyplot** that automates:
  - defining **FigureCanvas**
  - defining **Figure Artist**
  - connecting **FigureCanvas** and **Figure Artist**
  - https://github.dev/matplotlib/matplotlib/blob/main/lib/matplotlib/pyplot.py

- Comparing with **Layer 2 (Artist Layer)** which is:
  - heavy and for developers
  - not for individuals who want to perform quick **Exploratory Analysis** of some data

Matplotlib Example

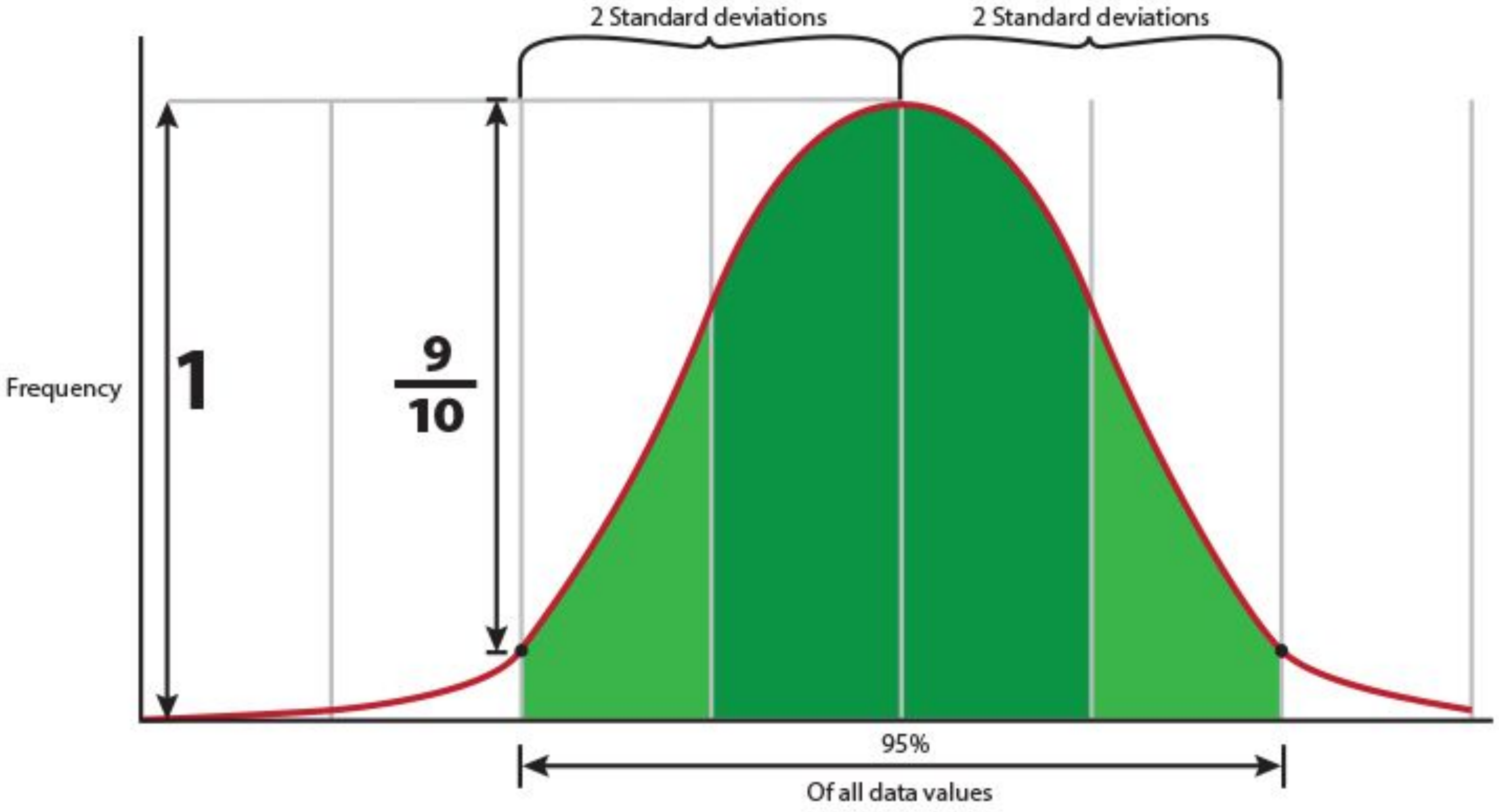
Normal Distributions (1 Standard Deviation)

68% of data is **< 1** standard deviation away from the mean



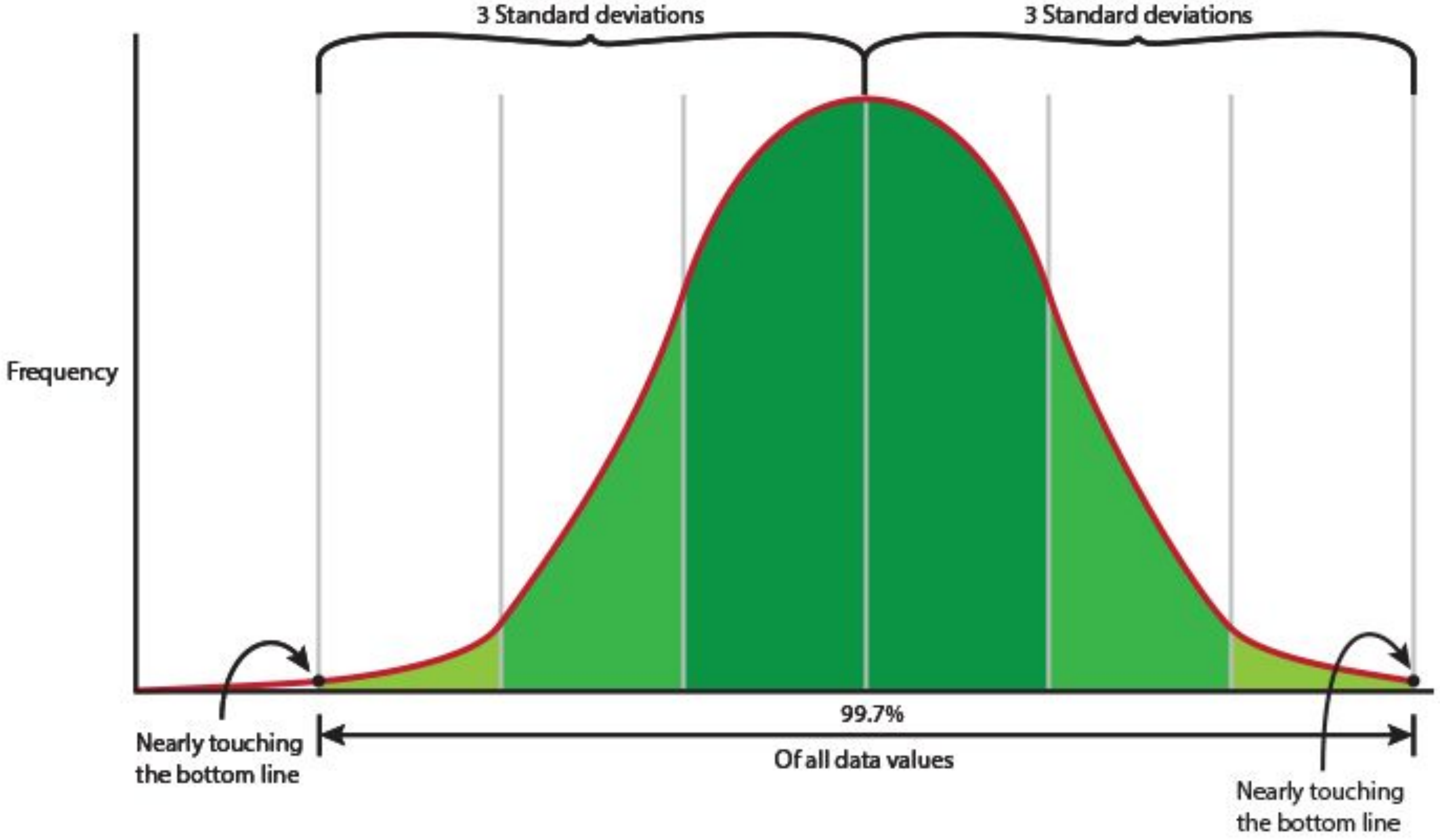
Normal Distributions (2 Standard Deviation)

95% of data is **< 2** standard deviations away from the mean

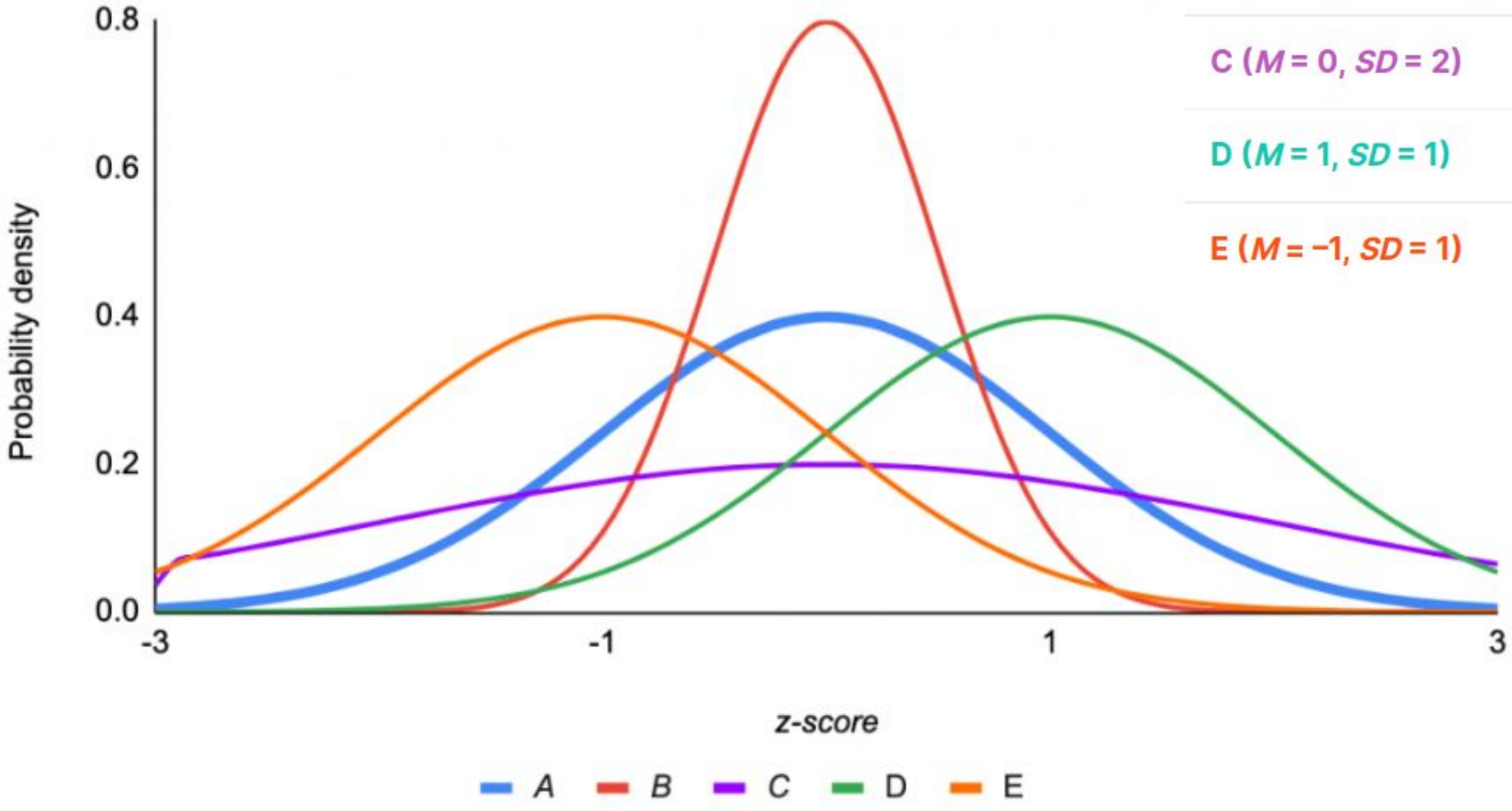


Normal Distributions (3 Standard Deviation)

99.7% of data is **< 3** standard deviations away from the mean



Standard Normal Distribution



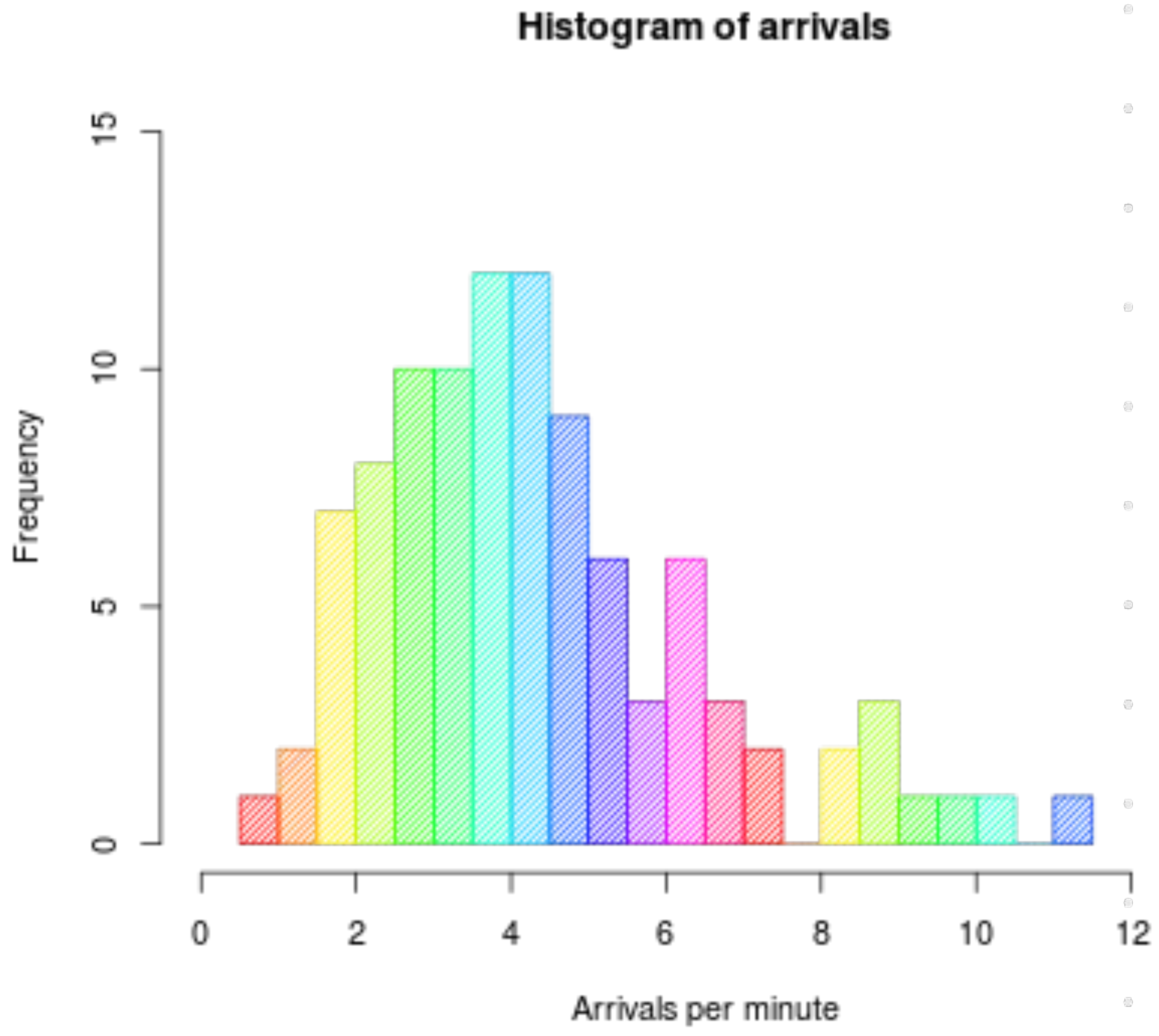
A (*M* = 0, *SD* = 1)	Standard normal distribution
B (*M* = 0, *SD* = 0.5)	Squeezed, because *SD* < 1
C (*M* = 0, *SD* = 2)	Stretched, because *SD* > 1
D (*M* = 1, *SD* = 1)	Shifted right, because *M > 0*
E (*M* = −1, *SD* = 1)	Shifted left, because *M* < 0

Histogram

- approximate representation of numerical data distribution

- **Construct Histogram**
  - bin (bucket) the range of values
  - count how many values fall into each interval

- **The Bins** are usually specified as
  - consecutive
  - non-overlapping intervals



Histogram vs Column Chart

- **Histogram**
  - used for **continuous data**, where the bins represent ranges of data

- **Column Chart**
  - plot of **categorical variables**

- **Recommendation**
  - **Column Chart** has gaps between the rectangles to clarify the distinction
  - **Histogram** rectangles touch each other to indicate: original variable is **continuous**

Scripting Layer Example (1/3)

localhost:8888/notebooks/dv-0.ipynb
jupyter
dv-0 Last Checkpoint: 21 hours ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Run
Code

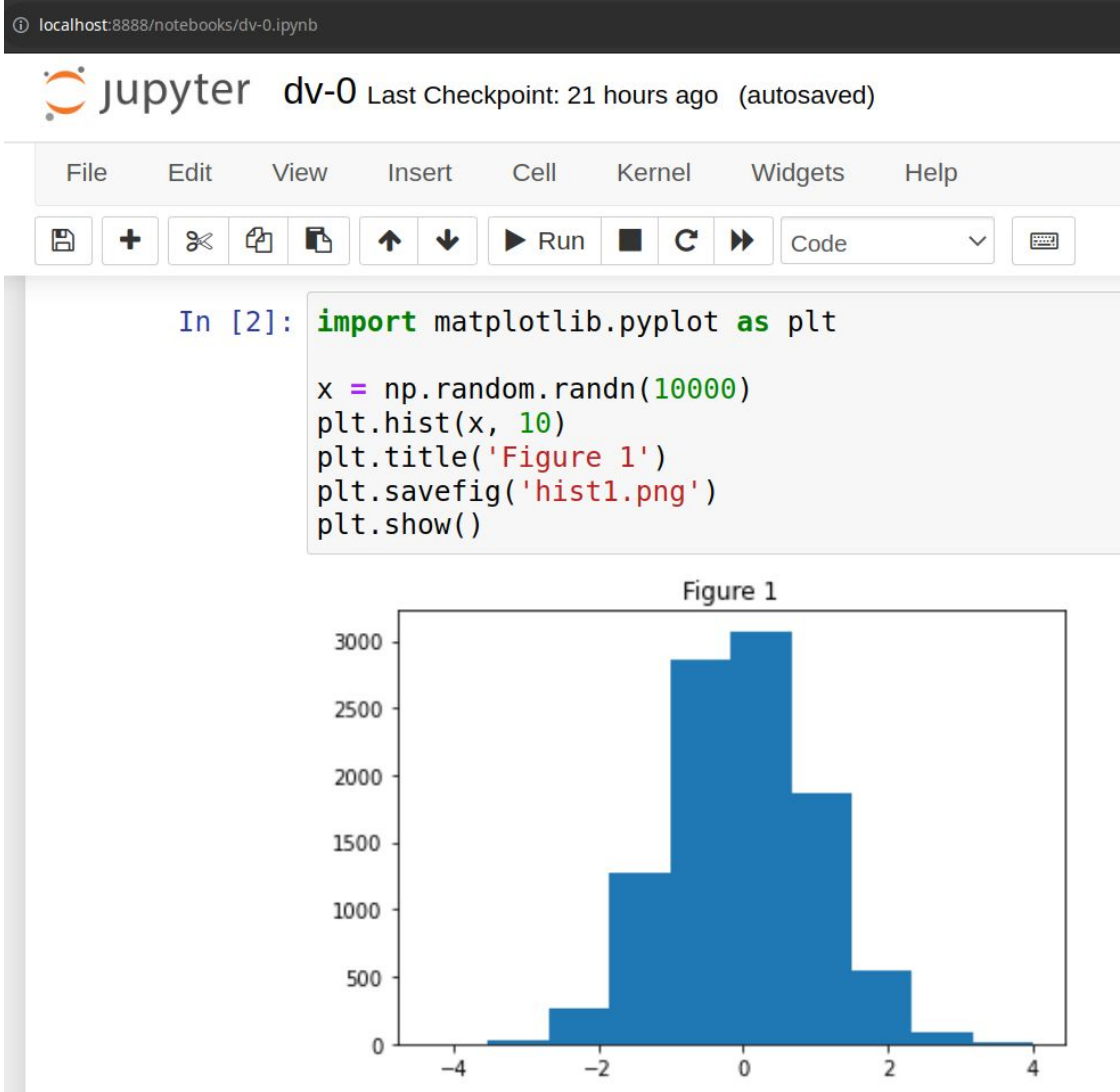
```
In [1]: import numpy as np

        np.random.randn(10) #Return 10 samples from the Standard Normal Distribution

Out[1]: array([-0.64393041,  0.0329367 , -0.16840147,  0.88846809,  0.76751103,
                0.18852699, -1.30213432,  0.58043701,  1.80149475, -0.18262329])
```

Scripting Layer Example (2/3)

- `import` **`pyplot`**
  - ○ from the **matplotlib library**

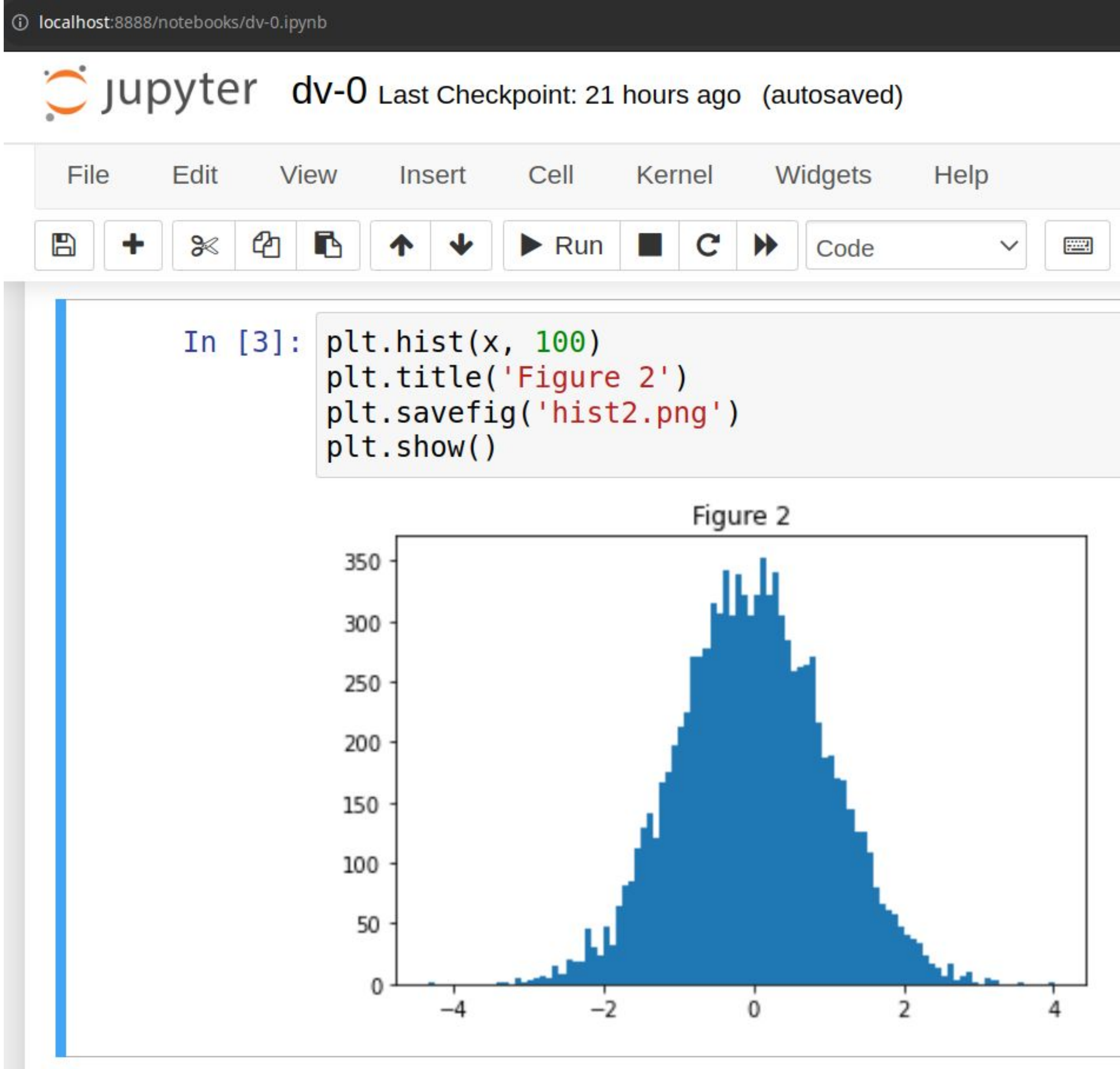


```
import matplotlib.pyplot as plt

x = np.random.randn(10000)
plt.hist(x, 10)
plt.title('Figure 1')
plt.savefig('hist1.png')
plt.show()
```

Scripting Layer Example (3/3)

- all methods
  - creating
    - histogram
    - other Artist objects
  - manipulating them
- are part of **pyplot**



```
plt.hist(x, 100)
plt.title('Figure 2')
plt.savefig('hist2.png')
plt.show()
```

Artist Layer Example 1

localhost:8888/notebooks/dv-1.ipynb
jupyter dv-1 Last Checkpoint: 07/10/2022 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
In [1]:

```
import numpy as np
x = np.random.randn(10000)

from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure
fig = Figure()                  #Creating Figure Artist
canvas = FigureCanvas(fig)      #Attach Figure Artist to FigureCanvas

ax = fig.add_subplot(111)       #Creating Axes Artist
ax.hist(x, 100)                 #Call the method hist of Axes Artist to generate the histogram
ax.set_title('Figure 3')
fig.savefig('hist3.png')
```

Artist Layer Example 1 - Notes

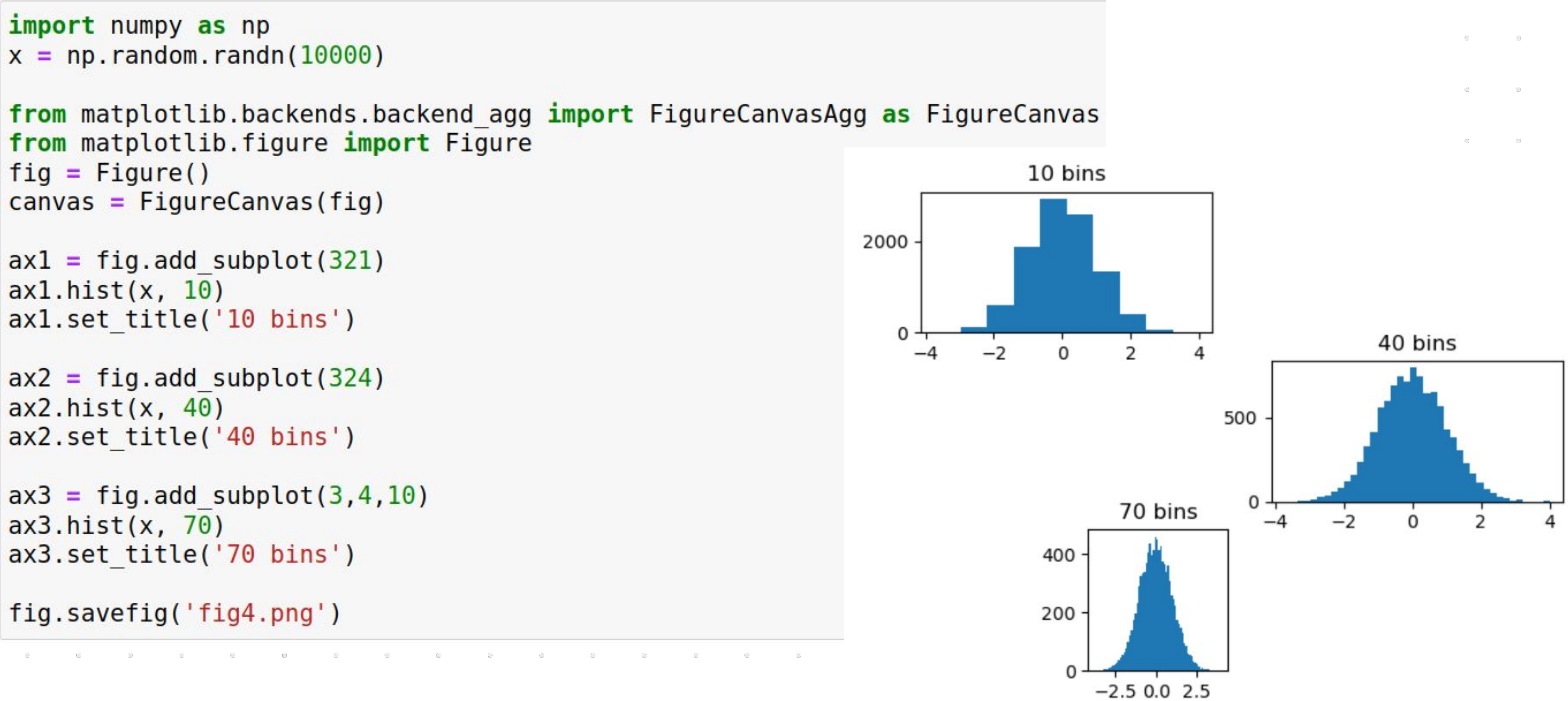
- Use **Artist Layer** to generate histogram of 10000 random numbers

- **Anti Grain Geometry (AGG)**
  - a high-performance library that produces attractive images

- use **111** (from MATLAB convention)
  - creates a grid with 1 row and 1 column
  - uses the first cell in that grid for the location of the new **Axes Artist**

- **hist method**
  - creates a sequence of **Rectangle Artists**

Artist Layer Example 2



```python
import numpy as np
x = np.random.randn(10000)

from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure
fig = Figure()
canvas = FigureCanvas(fig)

ax1 = fig.add_subplot(321)
ax1.hist(x, 10)
ax1.set_title('10 bins')

ax2 = fig.add_subplot(324)
ax2.hist(x, 40)
ax2.set_title('40 bins')

ax3 = fig.add_subplot(3,4,10)
ax3.hist(x, 70)
ax3.set_title('70 bins')

fig.savefig('fig4.png')
```

Questions

Links

https://github.com/fcai-b/dv

References

1. https://www.tableau.com/about/blog/examining-data-viz-rules-dont-use-red-green-together
2. https://www.coursera.org/learn/python-for-data-visualization