

Crisp python plots based on visualization theory

It's time to upgrade from Matplotlib to Plotly — here's how and why.



Nathan Lambert

Apr 11 · 8 min read ★

Practically speaking, this post is a large chunk of what we can all learn from the classic, *The Visual Display of Quantitative Information* — Edward Tufte. And I include how to implement it in Python.

Pillars of Displaying Information

Graphical Excellence

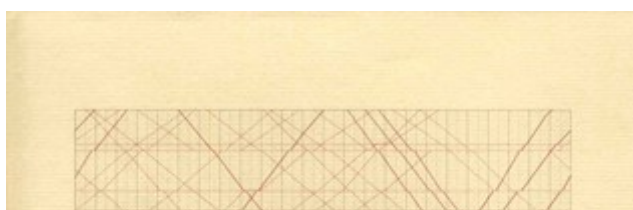
This is the **quality** of the presentation. By removing extra shapes, distracting colors, and inconsistent fonts, your data will be viewed more favorably. In my experience (and that of Edward Tufte), graphical excellence comes from the accumulated benefits of many small changes. Every piece of ink on your plot matters.

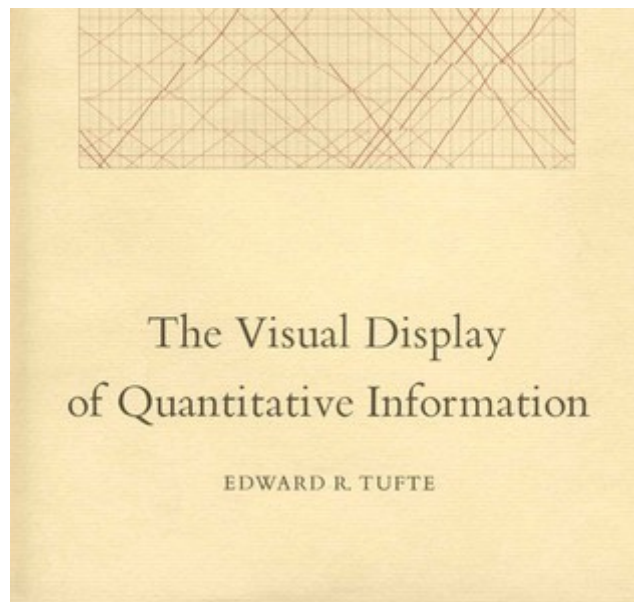
Graphical Integrity

This is the **truth** of the presentation. By making sure your axes are scaled correctly and reflect the trends, your data will be interpreted more honestly. In the long term, *the truth is always more valuable*. Everyone has seen one of the plots with a nonlinear y-axis to make hypotheses look stronger — vow to never do this.

This tutorial is the few lines of code you need to cleanly and truly show your data.

. . .





If you're passionate about your data, I would pick up a copy.

For all the data scientists and information nerds out there, *The Visual Display of Quantitative Information* is the ultimate coffee table book. There is a reason it is the undisputed king in books on displaying information, and it is quite enjoyable to peruse.

I've been using the methods outlined in it to create powerful plots in my research papers in robotics and machine learning. A couple excerpts below, kindly downsampled by Medium's lack of rendering pdf images (for now).

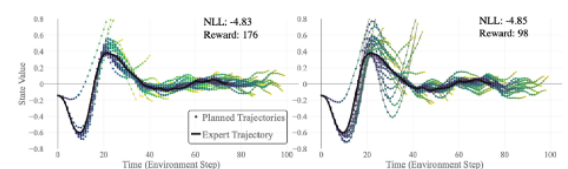
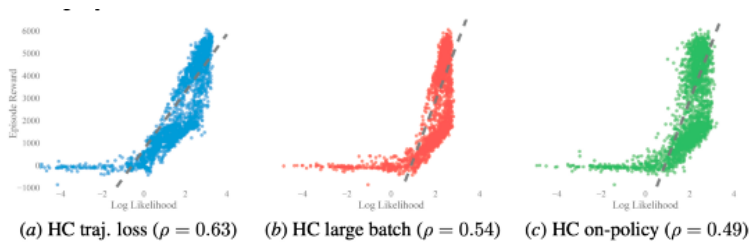


Figure 6: (left) Planned trajectories along the expert trajectory for the initial model and (right) the adversarially generated model trained to lower the reward. This is a snapshot of 1 trial, not averaged. It can be seen how the planned trajectories are qualitatively similar except for the peak at $t = 25$ where the control behavior deviates.

Left) Scatter plots that have an inherent sense of density. Right) Colored trajectories through the lines+markers plotly scatter object.

• • •

The goal of this post is threefold.

1. I want to make sure everyone understands some **critical mistakes** that they are making using default plotting functions.

2. **Introduce Plotly** to my audience, who works with data and explores articles for self-improvement in the area of artificial intelligence and data.
3. Cover the basics of **visualization theory**, and advanced techniques for drawing wandering eyes to your work in crowded digital domains.

The code included is excerpts from `plot.py` in my tutorial repo, which I will expand to include best practices for 3d plots, animation, and more.

natolambert/plotting-basics

Collection of lines of code for basics of clean plots in Plotly and Matplotlib — natolambert/plotting-basics

github.com

. . .

Tutorial begins here. The tutorial will work side-by-side with two plotting tools — Matplotlib and Plotly.



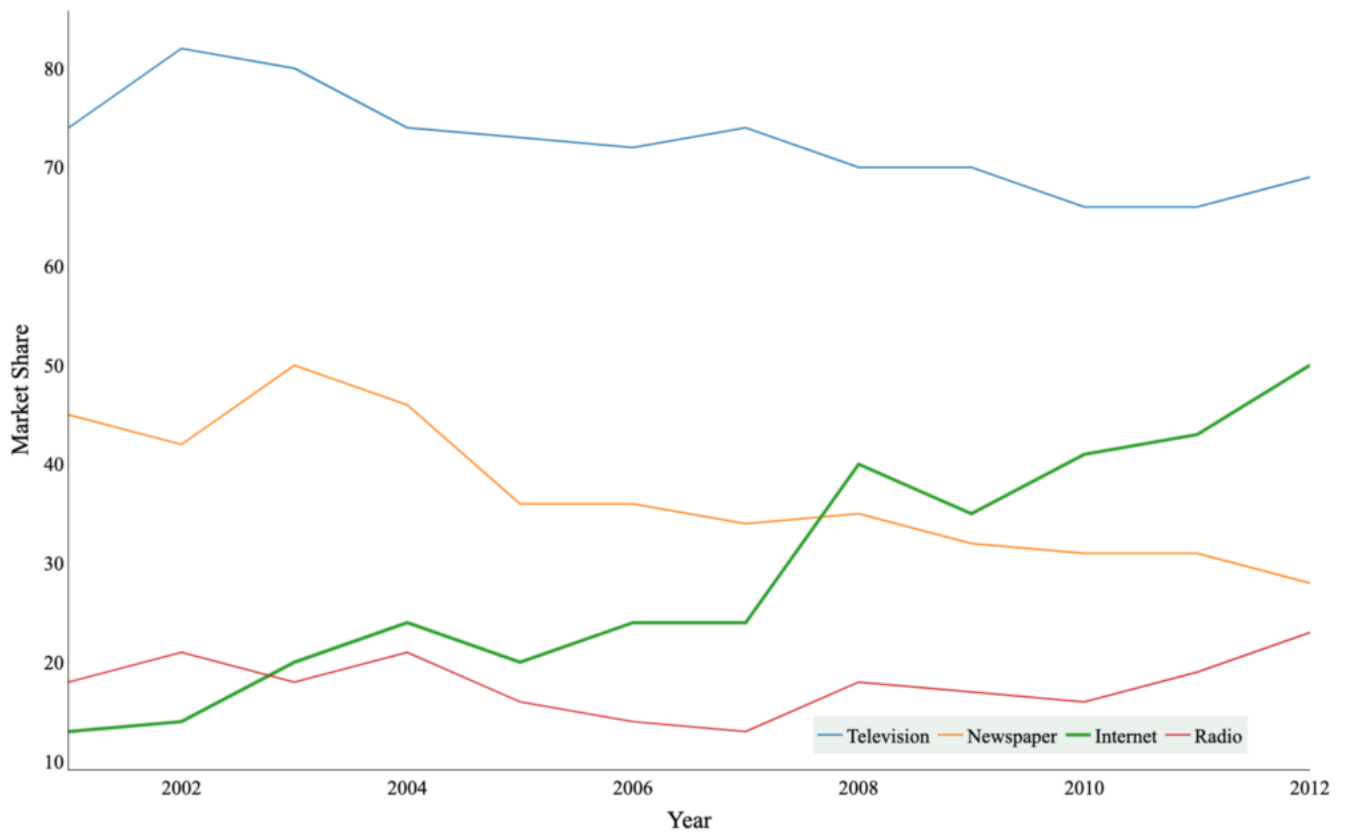
Left — Matplotlib logo, right — Plotly logo.

1. **Matplotlib**: The old plotting engine driving so much legacy experimentation code, the crutch of the engineer stuck in the past.

2. Plotly: The future of plotting in data science, analytics, and definitely my career.

Throughout, I make the case that plotly gives the user more tools to *maintain graphical excellence and integrity*.

0. Setup



Here's the plot we are going to build. Little clutter, simple data. A platform for more advanced representations.

I include the setup (imports and data-loading) at the end of this post, so people can reproduce it if desired. The barrier of dependencies is low, so we can easily get top tier plotting and safe saving mechanisms. *Fewer dependencies means more people will fork, copy, and build on your work.*

- Plotting tools for easy development: `matplotlib`, `matplotlib.pyplot`, `plotly`, `plotly.graph_objects`
- Math & data tools: `numpy`, `pandas`
- System tools: `os`, `sys`

Initialize Plots

The first step in creating a new data visualization can set the user up for failure. ***Always create an axis or a specific figure object.*** This allows complete control over where data is sent in how.

Plotly already takes a step ahead. With **subplots** the figure is indexed with each row and column rather than a list of axes that you have to keep track of in matplotlib (when `n=1`, the `plt.subplots` call works out).

```
##### Init plot / subplots #####
# mpl
fig_mpl, ax = plt.subplots()

# plotly
fig_plo = plotly.subplots.make_subplots(rows=1, cols=1)

##### add data #####
for i in range(0, 4):
    # mpl
    ax.plot(...)

    # plotly
    fig_plo.add_trace(go.Scatter(...))
```

1. Remove extra information

Gridlines should be gone

The ultimate source of clutter digitally and in print is grid lines. Even when rendering as a PDF, gridlines don't look great (zoom out and see what the gridlines look like), and they rarely help with understanding for a focused reader. Let the trends speak for themselves.

```
# mpl
ax.grid(False)

#plotly
fig.update_layout(xaxis_showgrid=False, yaxis_showgrid=False)
```

Extra box lines are useless

Use the white space. In any medium, the space is limited. Boxing in your data removes precious space from the page where you could present data. The right and top lines

should be removed, but sometimes the left and bottom lines are pretty.

```
# mpl
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.set_xlabel("Year")
ax.set_ylabel("Market Share (%)")

# plotly
fig_plo.update_xaxes(title_text="Year", linecolor='black',
                    row=1, col=1, zeroline=True,)
fig_plo.update_yaxes(title_text="Market Share", linecolor='black',
                    row=1, col=1, zeroline=True,)
```

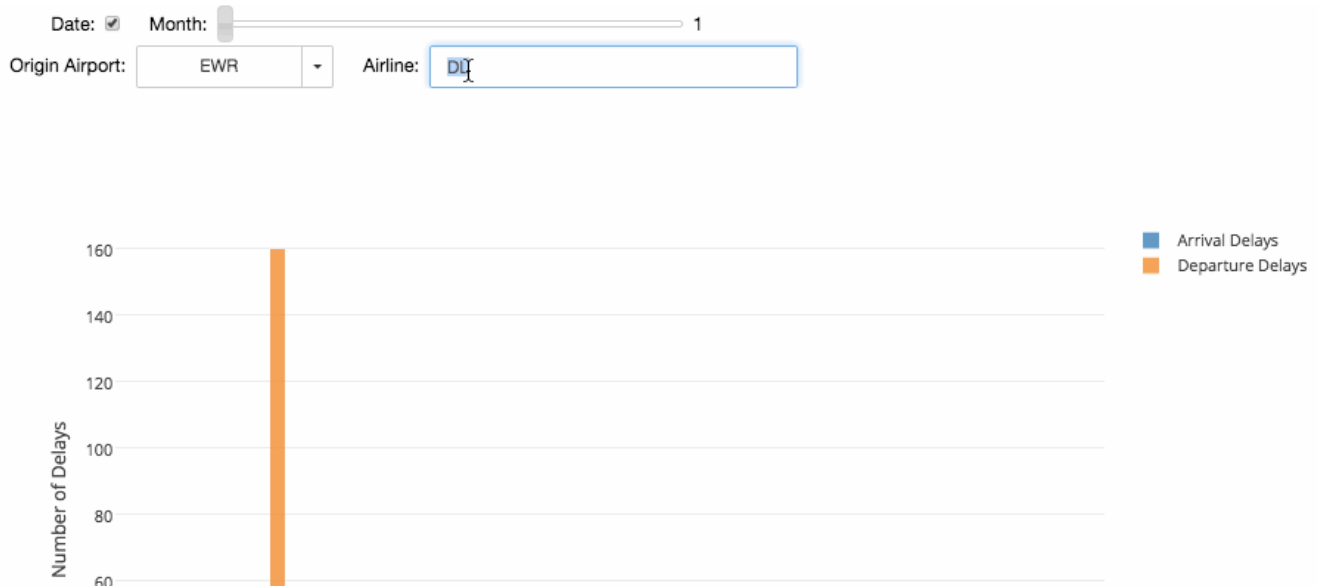
Control your legends

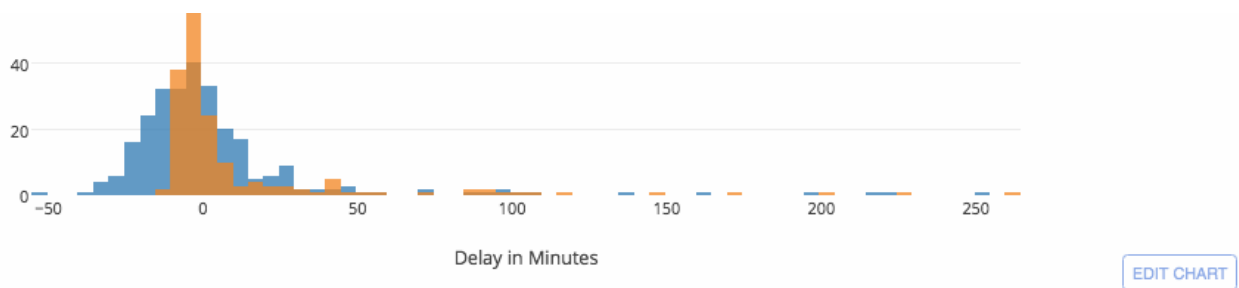
There'll always be quick readers bouncing around plots. Always have a legend answering their questions of what's what. Plotly has incredible legend tools like grouping, hidden items that are always visible, and interactive plots that show a subset of chosen legend entries.

Let your users see the data in full and see what they want with the **interactive plotly dashboard**.

```
# mpl
ax.legend()

# plotly
fig_plo.update_layout(showlegend=True,)
```





An example of the plotly dashboard — Source.

2. Consistent Fonts

Size matters & match your medium

Font is often neglected. Dropping a font with Arial font into your sophisticated report with Times font will always look out of place. Scale the font size to match the text (nearly) and always match the font.

```
# mpl
font = {'size': 24, 'family': 'serif', 'serif': ['Times']}
matplotlib.rc('font', **font)
matplotlib.rc('text', usetex=True)

# plotly
fig_plo.update_layout(font=dict(
    family="Times New Roman, Times, serif",
    size=24,
    color="black"
),
)
```

3. Printable

Remove backgrounds and use high resolutions

Always crank up the resolution on your plots. By default they're low dpi and low size, so projecting them on a screen is nearly useless. *Plotly lets you set the resolution for viewing and saving at once, so you can see exactly the plot you will save when you show, this is a win you'll understand when you try it.*

Matplotlib separates saving from viewing, which makes you generate different plots with `show()` than with `savefig()`, and leads to headaches.

```
# mpl
ax.set_facecolor((1, 1, 1))
# no resolution tool until saving :(

# plotly
fig_plo.update_layout(plot_bgcolor='white',)
fig_plo.update_layout(width=1000, height=1000,)
```

4. Work with screens in mind

Save as a PDF


The portable document format (PDF) saves your plot as a series of colored vector objects, so when it moves around a screen or to a new device, it is rendered again. *PDF will make you never have a pixelated plot in a presentation or manuscript again.*

```
# mpl
plt.savefig(os.getcwd()+"plop_mpl.pdf", dpi=300)
plt.show()

# plotly
fig_plo.write_image(os.getcwd()+"plot_plotly.pdf")
fig_plo.show()
```

Low contrast colors

#23aaff 

#ff6555 

#66c56c 

#f4b247 

Nice colors on a screen. Soft, distinct, uniquely designed by me.

A subtle point in plotting for screens is which colors to use. The colors should be 1) differentiable and 2) easy on the eye. This ends up being *washed out*, core colors. Check out the colormaps available below, but first I have a few tried and tested colors.

My colors **sky blue** #23aaff , **red apple** #ff6555 , **moss green** #66c56c , **mustard yellow** #f4b247 . Below are links to look at some colors from matplotlib and plotly.

Choosing Colormaps in Matplotlib - Matplotlib 3.1.0 documentation

Matplotlib has a number of built-in colormaps accessible via . There are also external libraries like [palettable] and...

matplotlib.org

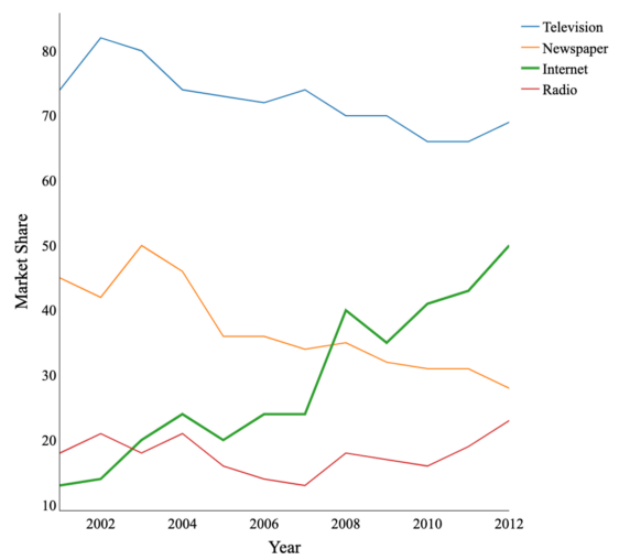
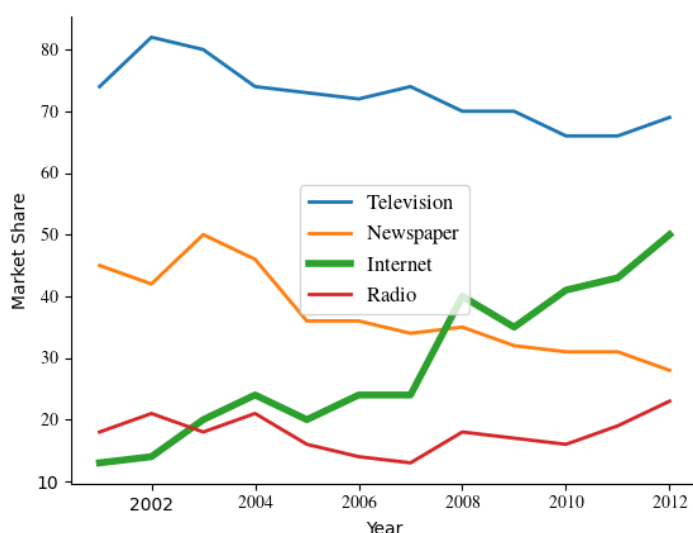
Discrete Colors

In the same way as the X or Y position of a mark in cartesian coordinates can be used to represent continuous values...

plotly.com

Outputs

Before scrolling further, look closely at these plots. See which one is more consistent and bug-free. Then you can decide which platform to use. **The bugs in one of them are from the source and unavoidable**, which is a non-tenable position for me.



Left) Matplotlib, Right) Plotly. The Matplotlib version auto-legend location is a total fail, some of the fonts don't get converted correctly, and is all in all less sharp.

I'm team plotly (right above), bug-free and sharp.

Improved Version in Plotly

Plotly's API has an easily accessible tool for almost every setting in plotting, and you can pass them in batches of one or two lines of code. Here is a collection of tweaks (preferences) I like in my plots. It lets you a) control the legend shape and location and b) remove white space around the plot. Give it a go, and look at the API — you'll find tons of tools.

```
# advanced
fig_plo.update_layout(legend_orientation="h",
                      legend=dict(x=.6, y=0.07,
                                bgcolor='rgba(205, 223, 212, .4)',
                                bordercolor="Black",
                                ),
                      margin=dict(r=10, l=10, b=10, t=10),
                      )
```

Python API reference for plotly - 4.6.0 documentation

This is the reference of plotly's API. Also see plotly's documentation website.

plotly.com



Take a step ahead with better visualization. Source — Author.

The less flashy details:

Imports

Here is how I like to structure my imports for a plotting script/module.

```
# file tools
import os
import sys

# plotting tools
import plotly
import plotly.graph_objects as go
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt

# Core
import numpy as np
```

Data

The data I used from this tutorial is from a plotly line plot example. You can check it out [here](#). An important thing to note is that plotly has nice color science — *muted colors are easier on the eyes when viewed digitally* (lower total color count r+g+b).

```
# From https://plotly.com/python/line-charts/
title = 'Main Source for News'
labels = ['Television', 'Newspaper', 'Internet', 'Radio']
colors = ['#1f77b4', # muted blue
          '#ff7f0e', # safety orange
          '#2ca02c', # cooked asparagus green
          '#d62728', # brick red
          ]

mode_size = [8, 8, 12, 8]
line_size = [2, 2, 4, 2]

x_data = np.vstack((np.arange(2001, 2014),)*4)

y_data = np.array([
    [74, 82, 80, 74, 73, 72, 74, 70, 70, 66, 66, 69],
    [45, 42, 50, 46, 36, 36, 34, 35, 32, 31, 31, 28],
    [13, 14, 20, 24, 20, 24, 24, 40, 35, 41, 43, 50],
    [18, 21, 18, 21, 16, 14, 13, 18, 17, 16, 19, 23],
])
```

To pretty plots!

Thanks to Anne Bonner.

[Computer Science](#)

[Python](#)

[Programming](#)

[Visualization](#)

[Data Science](#)

[About](#)

[Help](#)

[Legal](#)