### **Built-in Plotly Express Styling Arguments**

Many common styling options can be set directly in the px function call. Every Plotly Express function accepts the following arguments:

* title to set the figure title
* width and height to set the figure dimensions
* template to [set many styling parameters at once](https://plotly.com/python/templates/) (see below for more details)
* labels to override the default axis and legend labels behaviour, which is to use the data frame column name if available, and otherwise to use the label name itself like "x", "y", "color" etc. labels accepts a dict whose keys are the label to rename and whose values are the desired labels. These labels appear in axis labels, legend and color bar titles, and in hover labels.
* category\_orders to override the default category ordering behaviour
* hover\_data and hover\_name to control which attributes appear in the hover label and how they are formatted.
* [Various color-related attributes](https://plotly.com/python/colorscales/) such as color\_continuous\_scale, color\_range, color\_discrete\_sequence and/or color\_discrete\_map set the colors used in the figure. color\_discrete\_map accepts a dict whose keys are values mapped to color and whose values are the desired CSS colors.
* Increasing the plot’s size
* Despining the plot
* Enlarging axes’ labels
* Including a title and a subtitle
* Making secondary information more transparent
* Aligning all the elements of the plot
* Using natural colors

**Advanced Features:**

Custom buttons, sliders, dropdown menus, Range slider and selectors

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

<https://advancedplotting.github.io/>

<https://python.plainenglish.io/sankeying-with-plotly-90500b87d8cf>

<https://shiny.rstudio.com/py/api/reference/shiny.render.plot.html>

<https://docs.enthought.com/mayavi/mayavi/>

**Links for customization guide:**

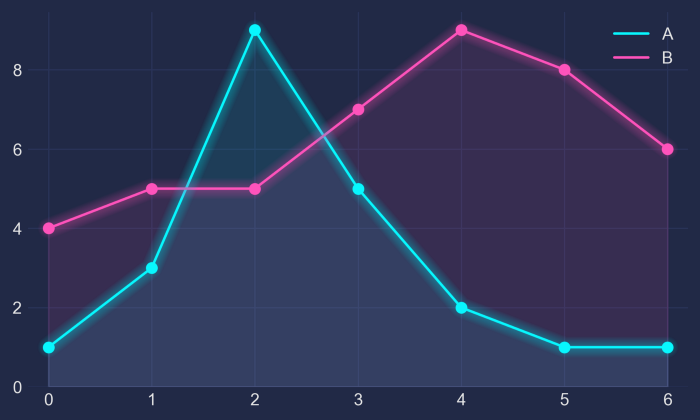
<https://plotly.com/python/reference/>

<https://towardsai.net/p/data-science/5-tricks-to-improve-bar-graphs-matplotlib>

<https://www.dataquest.io/blog/how-to-make-your-plots-appealing-in-python/>

# **Cyberpunk Style with Matplotlib**

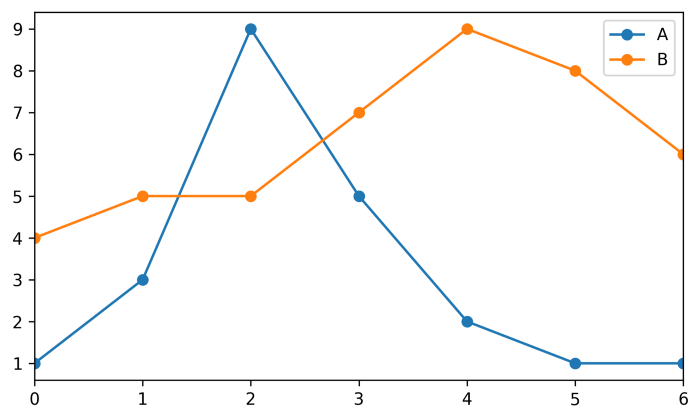
Futuristic neon glow for your next data visualization.

 *There’s now a Python package to conveniently apply this style, see* [*here*](https://github.com/dhaitz/mplcyberpunk)*. Install via*pip install mplcyberpunk

# **1 — The Basis**

Let’s make up some numbers, put them in a Pandas dataframe and plot them:

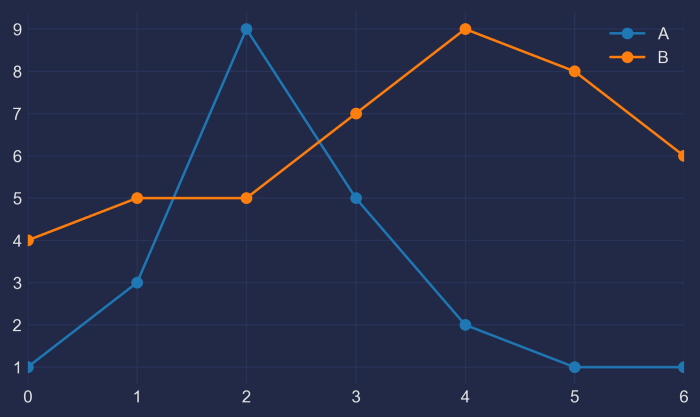
import pandas as pd  
import matplotlib.pyplot as pltdf = pd.DataFrame({'A': [1, 3, 9, 5, 2, 1, 1],  
 'B': [4, 5, 5, 7, 9, 8, 6]})df.plot(marker='o')  
plt.show()



# **2 — The Darkness**

Not bad, but somewhat ordinary. Let’s customize it by using Seaborn’s dark style, as well as changing background and font colors:

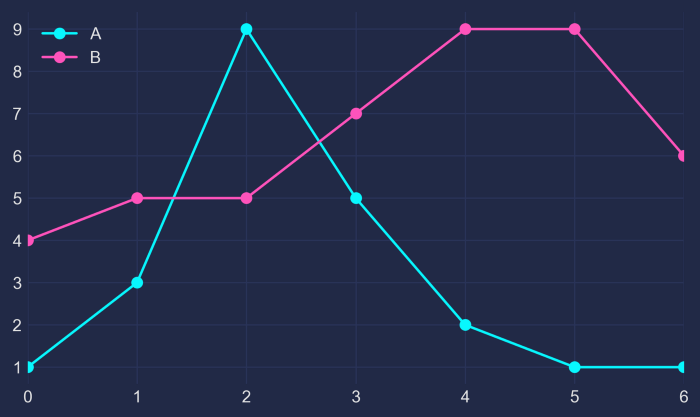
plt.style.use("seaborn-dark")for param in ['figure.facecolor', 'axes.facecolor', 'savefig.facecolor']:  
 plt.rcParams[param] = '#212946' # bluish dark greyfor param in ['text.color', 'axes.labelcolor', 'xtick.color', 'ytick.color']:  
 plt.rcParams[param] = '0.9' # very light greyax.grid(color='#2A3459') # bluish dark grey, but slightly lighter than background



# **3 — The Light**

It looks more interesting now, but we need our colors to shine more against the dark background:

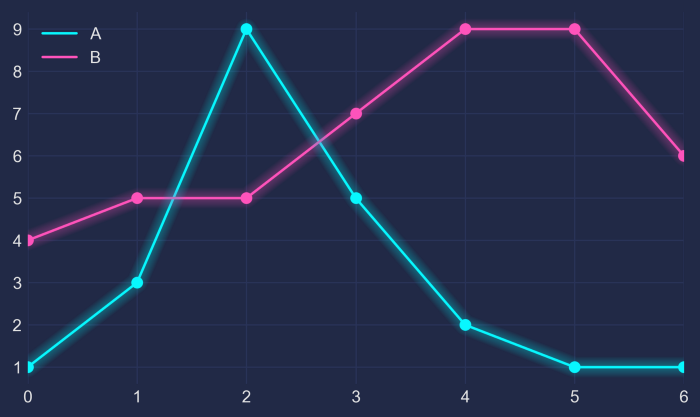
fig, ax = plt.subplots()  
colors = [  
 '#08F7FE', # teal/cyan  
 '#FE53BB', # pink  
 '#F5D300', # yellow  
 '#00ff41', # matrix green  
]  
df.plot(marker='o', ax=ax, color=colors)



# **4 — The Glow**

Now, how to get that neon look? To make it shine, we *redraw the lines multiple times*, with low alpha value and slighty increasing linewidth. The overlap creates the glow effect.

n\_lines = 10  
diff\_linewidth = 1.05  
alpha\_value = 0.03for n in range(1, n\_lines+1): df.plot(marker='o',  
 linewidth=2+(diff\_linewidth\*n),  
 alpha=alpha\_value,  
 legend=False,  
 ax=ax,  
 color=colors)



# **5 — The Finish**

For some more fine tuning, we color the area below the line (via ax.fill\_between) and adjust the axis limits.

Here’s the full code:

import pandas as pd  
import matplotlib.pyplot as plt  
plt.style.use("dark\_background")for param in ['text.color', 'axes.labelcolor', 'xtick.color', 'ytick.color']:  
 plt.rcParams[param] = '0.9' # very light greyfor param in ['figure.facecolor', 'axes.facecolor', 'savefig.facecolor']:  
 plt.rcParams[param] = '#212946' # bluish dark greycolors = [  
 '#08F7FE', # teal/cyan  
 '#FE53BB', # pink  
 '#F5D300', # yellow  
 '#00ff41', # matrix green  
]  
df = pd.DataFrame({'A': [1, 3, 9, 5, 2, 1, 1],  
 'B': [4, 5, 5, 7, 9, 8, 6]})fig, ax = plt.subplots()df.plot(marker='o', color=colors, ax=ax)# Redraw the data with low alpha and slighty increased linewidth:  
n\_shades = 10  
diff\_linewidth = 1.05  
alpha\_value = 0.3 / n\_shadesfor n in range(1, n\_shades+1): df.plot(marker='o',  
 linewidth=2+(diff\_linewidth\*n),  
 alpha=alpha\_value,  
 legend=False,  
 ax=ax,  
 color=colors)# Color the areas below the lines:  
for column, color in zip(df, colors):  
 ax.fill\_between(x=df.index,  
 y1=df[column].values,  
 y2=[0] \* len(df),  
 color=color,  
 alpha=0.1)ax.grid(color='#2A3459')ax.set\_xlim([ax.get\_xlim()[0] - 0.2, ax.get\_xlim()[1] + 0.2]) # to not have the markers cut off  
ax.set\_ylim(0)plt.show()

