Week 3 Lab (Intro to Matplotlib) COSC 3337 Dr. Rizk Intro to Matplotlib Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is used along with NumPy to provide an environment that is an effective open source alternative to MatLab. Thus, this may feel very familiar to you if you've ever used MatLab. Please note that Matplotlib is not the only tool for creating data visualization in Python. Other common libraries include seaborn, plotly, and even pandas has some built in functions for quick plotting. This lab will mainly focus on Matplotlib, but I highly recommend checking out seaborn as well, since it can make some type of plots much easier to create. The good news is that seaborn is built on top of Matplotlib, so it will be very easy to pick up after going through this lab. However, you're free to use any of the plotting tools mentioned above throughout the course. Just as there were many ways to solve a problem in pandas, there are many ways to visualize our data. Let's begin by importing matplotlib and learning how to create some beautiful graphs. If for some reason you don't have matplotlib installed, you will first have to go to your terminal (or Anaconda Prompt if on Windows) and enter the following: conda install matplotlib Make sure you've already installed Anaconda import numpy as np import pandas as pd import matplotlib.pyplot as plt The cell below is optional. It just makes our figure sizes 15 x 5 and adds a darkgrid background by default. Feel free to experiment with these values if you'd like, but you'll have to restart the kernel and run all to see any changes. import seaborn as sns from matplotlib import rcParams rcParams['figure.figsize'] = 15, 5 sns.set style('darkgrid') **Plotting Numpy Arrays** Let's begin by creating some numpy arrays and plotting them. We can create a simple x, y graph using plt.plot(x, y). We can then display this by calling **plt.show()**. In [3]: x = np.linspace(0, 20)y = np.sin(x)plt.plot(x, y) plt.show() 1.00 0.75 0.50 0.25 0.00 -0.25 -0.50-0.75-1.000.0 2.5 5.0 7.5 12.5 15.0 17.5 20.0 10.0 It's would also be nice to add titles and axis labels to our graphs. We can accomplish this using plt.title() and plt.x/ylabel(). plt.plot(x, y) In [4]: plt.title('My Graph Title') plt.xlabel('My x-axis label') plt.ylabel('My y-axis label') plt.show() My Graph Title 1.00 0.75 0.50 0.25 ŝ -0.25 -0.50-0.75-1.000.0 2.5 5.0 7.5 12.5 15.0 20.0 My x-axis label Note: Make sure to add any titles/labels and such before calling plt.show(). Otherwise they will not show up. We'll sometimes like to plot more than one function together, which can be done by calling plt.plot() a second time in the same cell. However, make sure to add an additional *label* parameter when doing this to give each plot/function a unique label name. That way we can then call *plt.legend()*, which will automatically add a legend for us and let us know which plot is which. plt.plot(x, np.sin(x), label='y=sin(x)')plt.plot(x, np.cos(x), label='y=cos(x)')plt.title('Sine and Cosine Functions ') plt.xlabel("x-axis") plt.ylabel("y-axis") # Assigns a color to each plot along with it's corresponding label name to display in a legend. plt.legend() plt.show() Sine and Cosine Functions 1.00 0.75 0.50 0.25 0.00 -0.25-0.50-0.75y=sin(x) -1.00 y=cos(x) 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0 **Subplots** What if we instead wanted to plot sin(x) and cos(x) as separate plots, but side by side? This is where **subplots(nrows, ncols)** comes in handy. subplots(nrows, ncols) will return to us a figure object and a set of axes. To better understand subplots, think of a figure object as a large canvas where we can place a set of axes on. These set of axes are where we will plot our graphs on. Previously, we only had one single axis in our figure, and we were plotting 1 or many functions on that single axis. However, figures can have many axes, which we'll demonstrate using subplots (nrows, ncols). For example, in the code below we tell subplots (nrows, ncols) that we would like a figure object with 1 row and 2 columns (essentially an array with 2 elements, where each element is an axis that we can plot on). Since axes can be thought of as an array holding the figure axes, we can index it to specify what we'd like to plot in each one. # one figure object with 1x2=2 axes fig, axes = plt.subplots(nrows=1, ncols=2) # plotting sin(x) in axes[0]. left image, or 1st element/axis in the axes array. axes[0].plot(x, np.sin(x))# setting the title of axes[0] to be y=sin(x) axes[0].set title('y=sin(x)') # plotting cos(x) in axes[1]. right image, or 2nd element/axis in the axes array. axes[1].plot(x, np.cos(x))# setting the title of axes[1] to be y=cos(x)axes[1].set_title('y=cos(x)') plt.show() y=sin(x) y=cos(x) 1.00 1.00 0.75 0.75 0.50 0.50 0.25 0.25 0.00 0.00 -0.25-0.25-0.50-0.50-0.75-1.00 -1.000.0 2.5 5.0 Try to see if you can now make a figure object with 2 rows and 2 columns, and plot something on all 4 axes. This will work exactly the same as above, except that we now index our axes as a 2d array. x = np.linspace(1, 21)fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 8)) axes[0][0].plot(x, x**2)axes[0][1].plot(x, x)axes[1][0].plot(x, np.log(x))axes[1][1].plot(x, x**5)plt.show() 20.0 400 17.5 15.0 300 12.5 200 10.0 7.5 100 5.0 2.5 0 0.0 0.0 20.0 0.0 20.0 5.0 10.0 12.5 15.0 17.5 5.0 10.0 12.5 15.0 17.5 3.0 4.0 3.5 2.0 2.5 1.5 1.5 1.0 0.5 0.5 0.0 5.0 20.0 0.0 5.0 **Custom Axes Range** Something else we'd sometimes like to do is change the x and y axis range. We can configure the ranges of the axes using the set_ylim and set_xlim, or axis('tight') for automatically getting "tightly fitted" axes ranges: In [8]: x = np.linspace(0, 5, 11)fig, axes = plt.subplots(nrows=1, ncols=3) # we can also plot multiple graphs using plt.plot(x1, y1, x2, y2). axes[0].plot(x, x**2, x, x**4)axes[0].set_title('default axis range') axes[1].plot(x, x**2, x, x**4)# tight happens to be the same here, but this will come in handy when you want to automatically align axes i # fitted manner. axes[1].axis('tight') axes[1].set_title('tight axis range') # notice the zoom in that happened. axes[2].plot(x, x**2, x, x**4)axes[2].set_ylim([0, 40]) axes[2].set_xlim([2, 3]) axes[2].set title("custom axes range"); plt.show() default axis range tight axis range custom axes range 600 600 35 500 500 30 400 400 25 20 300 200 200 10 100 100 0 2.8 Final Note: The plt.plot() function supports additional arguments that can be used to specify a wide variety of line color and style alternatives. Here are just a few examples, but please refer to the official documentation if you'd like to learn more. x = np.linspace(0, 5, 11)plt.plot(x, x+1, color="red", linewidth=0.25) plt.plot(x, x+2, color="red", linewidth=0.50) plt.plot(x, x+3, color="red", linewidth=1.00) plt.plot(x, x+4, color="red", linewidth=2.00) # possible linestype options '-', '-', '-', ':', 'steps' plt.plot(x, x+5, color="green", lw=3, linestyle='-') plt.plot(x, x+6, color="green", lw=3, ls='-.') plt.plot(x, x+7, color="green", lw=3, ls=':') # custom dash (not something we'll do often) line, = plt.plot(x, x+8, color="black", lw=1.50) line.set dashes([5, 10, 15, 10]) # format: line length, space length, ... # possible marker symbols: marker = '+', 'o', '*', 's', ',', '.', '1', '2', '3', '4', ... plt.plot(x, x+ 9, color="blue", lw=3, ls='-', marker='+') plt.plot(x, x+10, color="blue", lw=3, ls='--', marker='o') plt.plot(x, x+11, color="blue", lw=3, ls='-', marker='s') plt.plot(x, x+12, color="blue", lw=3, ls='--', marker='1') # marker size and color plt.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2) plt.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4) plt.plot(x, x+15, color="purple", lw=1, ls='-', marker='o', markersize=8, markerfacecolor="red") plt.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8, markerfacecolor="yellow", markeredgewidth=3, markeredgecolor="green"); 20.0 17.5 12.5 10.0 7.5 5.0 0.0 **Plotting Pandas Columns** Great job! Let's now see how we can plot pandas columns (which essentially behave like arrays), which is much more common and what you'll be doing when exploring your datasets. To do this, let's revisit that pokemon dataset. Plese refer back to the pandas lab if you don't recall the pokemon dataset or need help accessing it. pokemon df = pd.read csv(filepath or buffer='Pokemon.csv') pokemon df.head() Type 1 Type 2 Total HP Attack Defense Sp. Atk Sp. Def Speed Generation Legendary 0 1 Bulbasaur Grass Poison 318 45 49 49 65 65 45 False **1** 2 Ivysaur Poison 405 60 62 63 80 80 60 False Grass **2** 3 Venusaur 525 83 100 100 80 False Grass Poison VenusaurMega Venusaur Grass Poison 625 80 100 123 122 120 80 False 4 4 Charmander 309 39 52 43 60 50 65 Fire NaN False matplotlib has a wide variety of plot types. We've only seen simple line plots above, but let's see how we can create a scatter plot of pokemon HP and Attack. We can acomplish this using **plt.scatter(x, y)**, where x and y can be arrays (as we saw earlier), or pandas columns. plt.scatter(x=pokemon df['HP'], y=pokemon df['Attack']) plt.xlabel('HP') plt.ylabel('Attack') plt.title('HP vs. Attack') plt.show() HP vs. Attack 175 150 100 50 100 200 0 Optional plot: If you've installed seaborn and imported it, we can easily expand on the scatterplot above by plotting each point by a specific category (Generation in this case). To do this, we simply specify the category we'd like to color by in an additional hue parameter. This can also be achieved in matplotlib, but you have to create a dictionary and map each unique value to a color, which can be a bit tedious. sns.scatterplot(x=pokemon_df['HP'], y=pokemon_df['Attack'], hue='Generation', data=pokemon_df) 175 150 125 100 75 50 25 0 50 100 200 We can visualize the distribution of any pandas column using **plt.hist(x)**, where x is any pandas column. plt.hist(x=pokemon_df['Attack']) plt.show() 175 150 100 75 50 25 0 50 75 100 Let's see a few plots that require us to do a little more work to create. I say a little more work because it may require us to gather multiple information in some cases. Recall what .value_counts() from the pandas section gave us. Calling value_counts() on pokemon_df['Type 1'] will list out each unique Type 1 pokemon type and how many pokemon of that type are in our dataset. However, this information looks rather boring. Let's see how we can turn this info into a bar plot. pokemon_df['Type 1'].value_counts() In [14]: Out[14]: Water 112 Normal Grass 70 Bug 69 Psychic 57 Fire 52 Electric 44 Rock 44 32 Ghost Ground 32 32 Dragon Dark 31 28 Poison Fighting 27 Steel 27 2.4 Ice 17 Fairy 4 Flying Name: Type 1, dtype: int64 Calling .index off of .value_counts() will return just the index names. Calling .values off of .value_counts() will return just the corresponding values. What we'll do is save this information into variables and then use these variables as our x and y in plt.bar(x, y) to create our bar plot. Notice how x is the index labels, and y is the corresponding value counts in the plot below. # collect index names (left side of .value counts()) type 1 names = pokemon df['Type 1'].value counts().index # collect corresponding value counts (right side of .value counts()) type 1 values = pokemon df['Type 1'].value counts().values # plt.bar(x, y)plt.bar(type 1 names, type 1 values) plt.title('Type 1 Pokemon value counts') plt.show() Type 1 Pokemon value counts 100 80 60 20 0 We can also take the top 5 types above (Water, Normal, Grass, Bug, Psychic) and represent this information in a pie chart using plt.pie(sizes, colors). Sizes is the size for each wegde in the pie chart. For sizes we'll pass only the first 5 type_1_values. We'll also pass a colors array to give each wedge a unique color. Shadow=True is optional (just adds a slight shadow to our pie chart for visual purposes), and startangle=90 is the angle by which the start of the pie is rotated, counterclockwise from the x-axis. Note that plt.pie(sizes, colors) returns two things. patches, which is a sequence of matplotlib.patches.Wedge instances, and texts, a list of the label Text instances. It's important that we save these into variables so that we can pass this information to plt.legend(patches, names). We'll pass patches so that the legend knows which patch/wedge corresponds to which color, and the first 5 type_1_names as the corresponding name to each patch/wedge. # colors we'll use for each patch/wedge colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral','brown'] patches, texts = plt.pie(type_1_values[:5], colors=colors, shadow=True, startangle=90) plt.legend(patches, type_1_names[:5]) # will center our pie chart, but not necessary plt.axis('equal') plt.tight layout() plt.show() Grass Bug Psychic Let's collect the first 4 pokemon names in our DataFrame along with their corresponding HP, Attack, and defense. x = pokemon_df['Name'][:4] $y1 = pokemon_df['HP'][:4]$ y2 = pokemon_df['Attack'][:4] y3 = pokemon_df['Defense'][:4] We can compare these 4 pokemon using a stackplot. plt.stackplot(x, y1, y2, y3) works as follows in our case: x is the first 4 pokemon in our dataset, y1 is their corresponding HPs, y2 their corresponding Attacks, and y3 their corresponding defense values. Notice how we're stacking each attribute on top of each of the 4 pokemon. labels = ["HP ", "Attack", "Defense"] plt.stackplot(x, y1, y2, y3) plt.legend(loc='upper left', labels=labels) # rotates the xlabels to appear vertical rather than horizontal by default. plt.xticks(rotation=90) plt.show() 300 Attack 250 200 150 100 50 0 VenusaurMega Venusaur We can also use our knowledge of subplots to represent this information in 3 separate subplots if we'd like. Recall that x is the first 4 pokemon names in our dataset (Bulbasaur, Ivysaur, Venusaur, VenusaurMega Venusaur). So for each of these 3 subplots we're plotting the names along the x axis, and their corresponding HP or Attack or Defense value as the y value. # Create 3 subplots sharing y axis fig, axes = plt.subplots(nrows=3, ncols=1, sharey=True) # 'ko-, r.-, and : are just line styles and optional.' # notice how you can also set multiple things together in one single command rather than # creating separate .set_title, .set_ylabel, .set_xlabel, etc. commands like we were doing earlier. axes[0].plot(x, y1, 'ko-')axes[0].set(title='3 subplots', ylabel='HP') axes[1].plot(x, y2, 'r.-')axes[1].set(xlabel='Pokemon', ylabel='Attack') axes[2].plot(x, y3, ':') axes[2].set(xlabel='Pokemon', ylabel='Defense') plt.show() 3 subplots 100 노 50 VenusaurMega Venusaur Venusaur lvysaur 100 50 Venusaur VenusaurMega Venusau 100 Defense 50 Bulbasau Venusaur VenusaurMega Venusaur Pokemon We can also make line plots of pandas columns, which we've seen in the earlier lab sections when we were passing arrays. These type of plots will come in handy if you decide to work with stock data for your project. plt.plot(pokemon df['Attack'], label='Attack') plt.plot(pokemon df['HP'], label='HP') plt.legend() plt.show() Attack 250 200 150 100 50 0 One last and very important plot that we'll talk about is the heatmap. If we call .corr() on a DataFrame we'll get back a correlation

matrix.

Defense

plt.show()

노

Attack

Defense

Sp. Atk

Def

Sp.

ΗP

simplicity and easy use).

plots you can think of.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

HP

1.000000

0.422386

Attack

0.422386

1.000000

0.239622 0.438687 1.000000 0.223549

Attack

Defense

0.239622

0.438687

Sp. Def 0.378718 0.263990 0.510747 0.506121 1.000000 0.259133

Speed 0.175952 0.381240 0.015227 0.473018 0.259133 1.000000

We can then pass this information into sns.heatmap() to visualize it.

Note: Make sure you don't have categorical variables in your DataFrame or you'll get an error.

pokemon df[['HP', 'Attack', 'Defense','Sp. Atk', 'Sp. Def', 'Speed']].corr()

Sp. Atk

0.362380

0.396362

Defense

Sp. Atk

recommend at least using Seaborn for heatmaps. You'll usually see people mixing a combination of matplotlib and Seaborn since it's a really powerful combo. Since Seaborn is built on matplotlib, a lot of plots can be made with just one or 2 lines of code in Seaborn and then you can just use matplotlib to tweek certain things to your liking that are hidden in Seaborn (hidden for

Congrats! Wou now know enough to begin exploring and visualizing data on your own. As always, feel free to continue exploring this dataset or repeat some of the steps above on a new dataset as practice. Also refer back to the documentation if we didn't get to cover a graph you need for your project this semester. Most visualization tools and libraries support just about every statistical

Note: You can also use matplotlib for this, but it's much more intuitive in Seaborn and just one line of code, so I'd really

Sp. Def

0.378718

0.263990

0.510747

sns.heatmap(pokemon df[['HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']].corr())

Speed

0.175952

0.381240

0.015227

- 0.8

0.6

0.2

Speed

Sp. Def