Lesson 1: Week 7 - Plotting with Matplotlib

Created: 2024-07-09T16:26:31.17487+10:00

The Matplotlib family

We will be looking at how to visualise data using Matplotlib, a powerful Python plotting library with which you can quickly generate plots of your data. It is inspired by Matlab, and many of its objects are similarly named. Hence, if you are familiar with Matlab then you should feel at home using Matplotlib.Matplotlib provides a module called pyplot as a convenient way to access the Matplotlib functionality, and it is actually with pyplot that we do the plotting.Plotting with pyplotTo use pyplot you must import it. Note that pyplot is part of the Matplotlib library, but since it is the only part of Matplotlib that we will be using we can use a more targeted import. It is standard to use the alias plt for the pyplot object:import matplotlib.pyplot as pltPlotting with pyplot via pandasWorking directly with pyplot can be a bit laborious. Pandas provides its own plot() function as a method of series and data frames, which automatically performs many of the common tasks involved in using pyplot.Because the plot() method is part of pandas, you do not need to import anything other than pandas to use it. It is standard to use the alias pd:import pandas as pdPlotting with pyplot via seabornAlthough pandas' plot() method simplifies the process of plotting with Matplotlib, there are some ways in which it is still limited. Seaborn is a library that is designed to further simplify the task of using pyplot. It is not part of pandas, but it is designed to work well with pandas. It is particularly good for working with categorical (i.e., non-numerical) data.To use seaborn you must import it. It is standard to use the alias sns:import seaborn as snsSeaborn has some very nice plotting styles. You can set a style by calling seaborn's set() function, and if you do not specify which style you would like then seaborn will just set its default style:import seaborn as sns sns.set()Keep in mind that calling set() will affect the style of all matplotlib plots, not just those you create using seaborn.The seaborn website has an excellent example gallery of plots, with the code that is used to produce them.

Working directly with pyplot

Although it is easier to do plotting with pandas and seaborn, we will go through some basic plotting directly with pyplot. This will give you a better understanding of what is going on behind the scenes when you plot with pandas and seaborn.An exampleSuppose you have the following (fictitious) data about quarterly unemployment rates for NSW and VIC, loaded into a Python dictionary:data = { 'NSW': {'Q1': 3.2, 'Q2': 3.4, 'Q3': 3.4, 'Q4': 3.6}, 'VIC': {'Q1': 3.5, 'Q2': 3.4, 'Q3': 3.0, 'Q4': 3.1}, }The following program uses pyplot to create line plots of these quarterly figures, one line for each state.import matplotlib.pyplot as plt data = { 'NSW': {'Q1': 3.2, 'Q2': 3.4, 'Q3': 3.4, 'Q4': 3.6}, 'VIC': {'Q1': 3.5, 'Q2': 3.4, 'Q3': 3.0, 'Q4': 3.1}, } # Create a new figure and call it 'fig' fig = plt.figure() # Add an axes to the figure and call it 'ax' ax = fig.add_subplot() # Add a line plot to ax # Use the quarters as the x-values and the NSW percentages as the y-values ax.plot(data['NSW'].keys(), data['NSW'].values()) # Add a line plot to ax # This time use the VIC percentages as the y-values ax.plot(data['VIC'].keys(), data['VIC'].values()) # Save the figure # This step is necessary for getting the plot to show here in Ed fig.savefig('plot.png')To get a plot to show here in Ed, you must save it using fig.savefig(). You can name the plot whatever you want.Adding some featuresIt would be better if we added a figure title, some line labels and a legend, and some axis labels:import matplotlib.pyplot as plt data = { 'NSW': {'Q1': 3.2, 'Q2': 3.4, 'Q3': 3.4, 'Q4': 3.6}, 'VIC': {'Q1': 3.5, 'Q2': 3.4, 'Q3': 3.0, 'Q4': 3.1}, } fig = plt.figure() # Add a figure title fig.suptitle('Unemployment Rates') ax = fig.add_subplot() # Specify labels this time ax.plot(data['NSW'].keys(), data['NSW'].values(), label='NSW') ax.plot(data['VIC'].keys(), data['VIC'].values(), label='VIC') # Show a legend ax.legend() # Specify axis labels ax.set_xlabel('Quarter') ax.set_ylabel('Unemployment (%)') fig.savefig('plot.png')Using multiple axesIn the figure above, both line plots were drawn on the same axes. You can draw them on separate axes instead, by adding two axes to the figure and specifying how they should be laid out.import matplotlib.pyplot as plt data = { 'NSW': {'Q1': 3.2, 'Q2': 3.4, 'Q3': 3.4, 'Q4': 3.6}, 'VIC': {'Q1': 3.5, 'Q2': 3.4, 'Q3': 3.0, 'Q4': 3.1}, } fig = plt.figure() fig.suptitle('Unemployment Rates') # Add an axes. It's the first axes of a 1 x 2 grid of axes. ax1 = fig.add_subplot(1, 2, 1) # No need for a label this time ax1.plot(data['NSW'].keys(), data['NSW'].values()) # Specify a title for the axes, and labels for the x- and y-axis ax1.set_title('NSW') ax1.set_xlabel('Quarter') ax1.set_ylabel('Unemployment (%)') # Add an axes. It's the second axes of a 1 x 2 grid of axes. ax2 = fig.add_subplot(1, 2, 2) # No need for a label this time ax2.plot(data['VIC'].keys(), data['VIC'].values()) # Specify a title for the axes, and labels for the x- and y-axis ax2.set_title('VIC') ax2.set_xlabel('Quarter') ax2.set_ylabel('Unemployment (%)') fig.savefig('plot.png')Some finishing touchesLet us add a few finishing touches:import matplotlib.pyplot as plt data = { 'NSW': {'Q1': 3.2, 'Q2': 3.4, 'Q3': 3.4, 'Q4': 3.6}, 'VIC': {'Q1': 3.5, 'Q2': 3.4, 'Q3': 3.0, 'Q4': 3.1}, } #

Set the size to be 10 inches wide by 8 inches tall fig = plt.figure(figsize=[10, 8]) fig.suptitle('Unemployment Rates') ax1 = fig.add_subplot(1, 2, 1) ax1.plot(data['NSW'].keys(), data['NSW'].values()) ax1.set_title('NSW') ax1.set_xlabel('Quarter') ax1.set_ylabel('Unemployment (%)') # Set the y-axis values to go from 3 to 4 ax1.set_ylim(3, 4) # Set the y-axis ticks to be 3.0, 3.1, 3.2, ..., 4.0 ax1.set_yticks([x/10 for x in range(30, 41)]) # Show gridlines on the axes ax1.grid() # Tell ax2 to share its y-axis with ax1 ax2 = fig.add_subplot(1, 2, 2, sharey=ax1) # Specify the colour of the line ax2.plot(data['VIC'].keys(), data['VIC'].values()…

More plots examples using pyplot

Example 1: Simple plotimport matplotlib.pyplot as plt x_numbers = [1, 6, 3] y_numbers = [2, 4, 6] plt.plot(x_numbers, y_numbers) plt.savefig('a.png')Example 2: Adding a markerimport matplotlib.pyplot as plt x_numbers = [1, 6, 3] y_numbers = [2, 4, 6] plt.plot(x_numbers, y_numbers, '+') plt.savefig('a.png')Example 3: Changing the markerimport matplotlib.pyplot as plt x_numbers = [1, 6, 3] y_numbers = [2, 4, 6] plt.plot(x_numbers, y_numbers, marker='*') plt.savefig('a.png')Example 4: Annual temperatures in NYCimport matplotlib.pyplot as plt nyc_temp = [53.9, 56.3, 56.4, 53.4, 54.5, 55.8, 56.8, 55.0, 55.3, 54.0, 56.7, 56.4, 57.3] plt.plot(nyc_temp, marker='o') plt.savefig('a.png')Example 5: Annual temperatures in NYC with yearsimport matplotlib.pyplot as plt nyc_temp = [53.9, 56.3, 56.4, 53.4, 54.5, 55.8, 56.8, 55.0, 55.3, 54.0, 56.7, 56.4, 57.3] years = range(2000, 2013) plt.plot(years, nyc_temp, marker='o') plt.savefig('a.png')Example 6: Comparing multiple datasetsimport matplotlib.pyplot as plt # New York City temperatures (farenheight) temp_2000 = [31.3, 37.3, 47.2, 51.0, 63.5, 71.3, 72.3, 72.7, 66.0, 57.0, 45.3, 31.1] temp_2006 = [40.9, 35.7, 43.1, 55.7, 63.1, 71.0, 77.9, 75.8, 66.6, 56.2, 51.9, 43.6] temp_2012 = [37.3, 40.9, 50.9, 54.8, 65.1, 71.0, 78.8, 76.7, 68.8, 58.0, 43.9, 41.5] months = range(1, 13) plt.plot(months, temp_2000, months, temp_2006, months, temp_2012) plt.savefig('a.png')Example 7: Multiple datasets with legendsimport matplotlib.pyplot as plt # New York City temperatures (farenheight) temp_2000 = [31.3, 37.3, 47.2, 51.0, 63.5, 71.3, 72.3, 72.7, 66.0, 57.0, 45.3, 31.1] temp_2006 = [40.9, 35.7, 43.1, 55.7, 63.1, 71.0, 77.9, 75.8, 66.6, 56.2, 51.9, 43.6] temp_2012 = [37.3, 40.9, 50.9, 54.8, 65.1, 71.0, 78.8, 76.7, 68.8, 58.0, 43.9, 41.5] months = range(1, 13) plt.plot(months, temp_2000, months, temp_2006, months, temp_2012) plt.legend([2000, 2006, 2012]) plt.savefig('a.png')Example 8: Adding a title and labelsimport matplotlib.pyplot as plt # New York City temperatures (farenheight) temp_2000 = [31.3, 37.3, 47.2, 51.0, 63.5, 71.3, 72.3, 72.7, 66.0, 57.0, 45.3, 31.1] temp_2006 = [40.9, 35.7, 43.1, 55.7, 63.1, 71.0, 77.9, 75.8, 66.6, 56.2, 51.9, 43.6] temp_2012 = [37.3, 40.9, 50.9, 54.8, 65.1, 71.0, 78.8, 76.7, 68.8, 58.0, 43.9, 41.5] months = range(1, 13) plt.plot(months, temp_2000, months, temp_2006, months, temp_2012) plt.legend([2000, 2006, 2012]) plt.title('Average monthly temperature in NYC') plt.xlabel('Month') plt.ylabel('Temperature (F)') plt.savefig('a.png')Example 9: Adjusting axes rangesimport matplotlib.pyplot as plt nyc_temp = [53.9, 56.3, 56.4, 53.4, 54.5, 55.8, 56.8, 55.0, 55.3, 54.0, 56.7, 56.4, 57.3] plt.plot(nyc_temp, marker='o') print(f'Original axes ranges {plt.axis()}') # Display existing axis values (auto-generated) newAxisRange = [0, 12, 53.4, 57.3] plt.axis(newAxisRange) print(f'updated axes ranges {plt.axis()}') plt.savefig('a.png')Example 10: Adjusting axes ranges version 2import matplotlib.pyplot as plt nyc_temp = [53.9, 56.3, 56.4, 53.4, 54.5, 55.8, 56.8, 55.0, 55.3, 54.0, 56.7, 56.4, 57.3] plt.plot(nyc_temp, marker='o') print(f'Original axes {plt.axis()}') # Display existing axis values (auto-generated) plt.axis(ymin = 50) # Set minimum of y-axis to zero plt.axis(ymax = 60) # Set maximum of y-axis print(f'updated axes ranges {plt.axis()}') plt.savefig('a.png')Example 11: Plot 1 revisitedimport matplotlib.pyplot as plt def create_graph(): x_numbers = [1, 6, 3] y_numbers = [2, 4, 6] plt.plot(x_numbers, y_numbers) plt.savefig('a.png') if __name__ == '__main__': create_graph()Example 12: Graphing functions - Gravityimport matplotlib.pyplot as plt # Draw the graph - takes the x and y sets of data as parameters def draw_graph(x, y): plt.plot(x, y, marker='o') plt.xlabel('Distance (metres)') plt.ylabel('Gravitational force (newton…