PHY1112 Lab 6

Matplotlib – Turning Data Into Beauty

February 13th, 2024

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Part | 1 | 2 | 3 | Total |
| Points | 5 | 5 | 15 | 25 |
| Score |  |  |  |  |

Objectives

1. Introduce Matplotlib and the framework it uses to help in figure creation.
2. Visualize previously generated data using Matplotlib.
3. Explore statistics of a large data set and use plotting to better understand them.

Part 1: An Introduction to Matplotlib – Creating your first plot

1. (5 points) Using the code from lecture as a starting point:
   1. Plot and for in the range on the same graph.

A graph of a function

Description automatically generated

* 1. In a second figure, plot for in the range . Describe how you might want to make this graph look better (just describe in words

Using a wider data range, for both the x and y axes. Using more precise steps for both the x and y axes.

A graph of a function

Description automatically generated

Don’t forget to use the numpy trigonometric functions, and don’t forget to add a title and axis labels.

Include a snapshot of the resulting graphs in this document.

Part 2: f-strings

1. (5 points) Use f-strings to print out the following quantities in the specified format. Include a snapshot of your output in this document.
   1. math.pi as a float with 6 digits after the decimal
   2. 100000.3 as a float in scientific notation with 1 digit after the decimal
   3. 733 in upper-case hexadecimal format with width 4 and zero padded on left, and with “0x” at the beginning.
   4. 4242 as a character
   5. 72.6942 as a float with no digits after the decimal

Part 3: Using Plots and Statistics to Better Understand Data

1. (5 points) Plot the high and low daily temperatures in Ottawa over the entire year of 2022, similarly to the example plot below. Use the weather data file from Lab 5.

“Day of the Year” can be [1,365], inclusive. Make sure the x-axis label and tick marks are correct, and that you include a legend to label your data.

A graph showing the temperature of the day

Description automatically generated

Note there are discontinuities in the graph due to missing data (NaN values)

**Example:** We plot the high and low temperatures in Ottawa for September 2023. The figure below is much more readable than looking at an array of values. For instance, we see visually that the beginning of the month had a heat wave.

A graph with red and blue lines

Description automatically generated

Assuming the data exists in the lists “high\_temperatures” and “low\_temperatures”, here is what the rest of the code that generated the figure above looks like:

A screen shot of a computer code

Description automatically generated

1. (5 points) Using the data for the month of July 2022 in your data file (days 181 to 211, inclusive):
2. Plot the high and low temperatures for this month only, similar to what you did for question 3 for the whole year.
3. Calculate the average high and average low for the month and report your values here.
4. Add to your plot two horizonal dashed lines that indicate these averages, similar to the example plot below. Again, make sure the x-axis label and tick marks are correct and that you include a legend to label your data. Take a snapshot of your figure and include it here.

A graph with red and blue lines

Description automatically generated

Example:

A graph with red and blue lines

Description automatically generated

1. (5 points) Again using the data for the month of July 2022 in your data file (days 181 to 211)
2. Calculate standard deviation of the high and low temperature data. Don’t forget to use the proper estimator, which is the square root of the sample variance with the factor in the denominator as discussed in lecture,

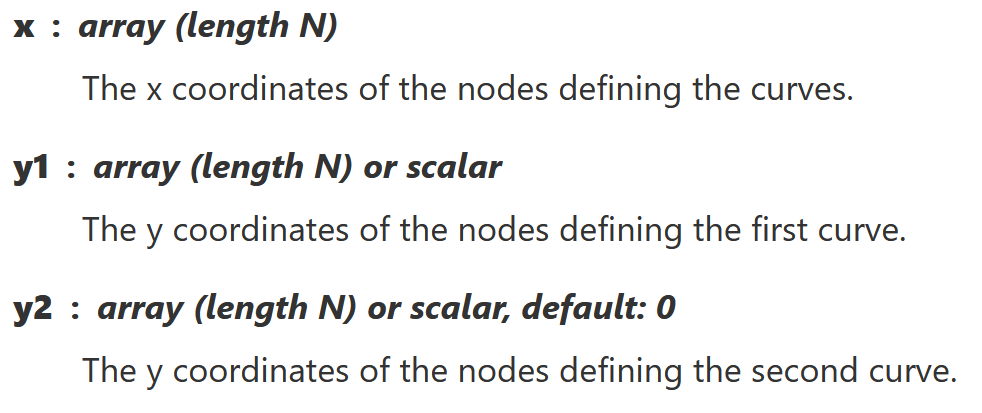
Recall that is the sample mean (*i.e.*, the average, that you already calculated in question 4 for high and low temperatures). Report the values that you obtained.

1. Add to your plot in question 4 two shaded areas that indicate the standard deviations, like in the example plot below. The bounds are placed at one standard deviation above and below the corresponding mean. Take a snapshot of your figure and include it here.

A graph showing the temperature of the year

Description automatically generated

**HINT**: For the shaded areas, use plt.fill\_between()[[1]](#footnote-2), where you will need to specify the following arguments:



Since the bounds of the shaded areas are horizontal lines, you only need two points to specify the x array (a list containing the first and last x values). Further, y1 and y2 will be scalars.

You can also specify colours of the shaded areas. For the pink and blue colours in the example plot below, we used the keyword arguments



and



A graph showing the average temperature of the month

Description automatically generated

**Remember to submit this document filled out, along with your python file on Brightspace!**

**Code**

'''

Filename:       lab6.py

Author:         Patrick Geraghty

Date Created:   2024-02-13

Date Modified:  2024-02-13

Description:    This file contains the functions for the lab 6 exercises and one additional function. The functions include sin\_cosine\_plot, tan\_plot, f\_strings, load\_data, year\_weather\_graph, and july\_weather\_graph.

'''

import matplotlib.pyplot as plt

import numpy as np

import math as m

*def* sin\_cosine\_plot():

    '''

    () -> None

    This function creates a plot of the sine and cosine functions on the same graph.

    Precondition: None

    '''

    # Specify the x-axis values

    x = np.linspace(0, 2\*np.pi)

    # Create the figure.

    plt.figure(1)

    # Create the graphs for sin and cosine. Differentiate with color.

    plt.plot(x, np.cos(x), *color* = 'b', *label* = 'cos(x)')

    plt.plot(x, np.sin(x), *color* = 'g', *label* = 'sin(x)')

    # Draw the x-axis and y-axis at y=0 and x=0, respectively. The color is red, the line style is dashed, and the line width is 2.

    plt.hlines(0, 0, 2\*np.pi, *colors* = 'r', *linestyles* = 'dashed', *linewidths* = 2)

    # Label the graph

    plt.title('y = cos(x) & y = sin(x)')

    plt.xlabel('x (radians)')

    plt.ylabel('y')

    # Set the x-axis limits

    plt.xlim(0, 2\*np.pi)

    # Set the x-axis tick values and labels

    xtick\_values = [0, np.pi/2, np.pi, 3\*np.pi/2, 2\*np.pi]

    xtick\_labels = ['0', *r*'$\frac{\pi}{2}$', *r*'$\pi$', *r*'$\frac{3\pi}{2}$', *r*'$2\pi$']

    plt.xticks(xtick\_values, xtick\_labels)

    # Add a legend to identify the graphs

    plt.legend()

    # Add a grid

    plt.grid()

    plt.show()

*def* tan\_plot():

    '''

    () -> None

    This function creates a plot of the tangent function.

    Precondition: None

    '''

    # define the x-axis values and the y-axis values

    x = np.linspace(-np.pi/2, np.pi/2, 1000)

    y = np.tan(x)

    # Create the figure

    plt.figure(2)

    # Create the graph for tan(x)

    plt.plot(x, y, *color* = 'r', *label* = 'tan(x)')

    # Draw the x-axis and y-axis at y=0 and x=0, respectively. The color is black, the line style is solid, and the line width is 2.

    plt.hlines(0, -np.pi, np.pi, *colors* = 'k', *linestyles* = '-', *linewidths* = 2)

    # Draw the vertical asymptotes at x = -pi/2 and x = pi/2. The color is black, the line style is dashed, and the line width is 2.

    plt.vlines(-np.pi/2, -5, 5, *colors* = 'k', *linestyles* = '--', *linewidths* = 2)

    plt.vlines(np.pi/2, -5, 5, *colors* = 'k', *linestyles* = '--', *linewidths* = 2)

    # Label the graph

    plt.title('y = tan(x)')

    plt.xlabel('x (radians)')

    plt.ylabel('y')

    # Set the x-axis limits and y-axis limits

    plt.xlim(-np.pi, np.pi)

    plt.ylim(-5, 5)

    # Set the x-axis tick values and labels

    xtick\_values = [-np.pi/2, 0, np.pi/2]

    xtick\_labels = [*r*'$-\frac{\pi}{2}$', '0', *r*'$\frac{\pi}{2}$']

    plt.xticks(xtick\_values, xtick\_labels)

    # Add a legend to identify the graph

    plt.legend()

    # Add a grid

    plt.grid()

    plt.show()

*def* f\_strings():

    '''

    () -> None

    This function demonstrates the use of f-strings.

    Precondition: None

    '''

    # Print the following values using f-strings

    print(*f*'{m.pi*:.6f*}')        # math.pi to 6 decimal places

    print(*f*'{10000.3*:.1e*}')     # 10000.3 in scientific notation to 1 decimal place

    print(*f*'0x{733*:04X*}')       # 733 in hexadecimal with 4 digits

    print(*f*'{chr(4242)}')       # Unicode character 4242

    print(*f*'{72.6942*:.0f*}')     # 72.6942 to 0 decimal places

# define a function to load the data

*def* load\_data():

    '''

    () -> np.array

    This function returns a numpy array of the data in the 'weather\_data\_lab6.csv'.

    Precondition: filename is a string

    '''

    # load the data from the file using np.genfromtxt. Define necessary columns, skip the header, identify the separator, and define the data type as float

    return np.genfromtxt('weather\_data\_lab5.csv', *usecols*=(9,11), *skip\_header*=1, *delimiter*=',', *dtype*=*float*)

# define a function to calculate the weather statistics

*def* year\_weather\_graph():

    '''

    () -> None

    Creates a graph of the weather data.

    Precondition: None

    '''

    # load the data

    data = load\_data()

    # create the figure

    plt.figure(3)

    # define highs and lows

    highs = data[:,0]

    lows = data[:,1]

    # plot the highs and lows

    plt.plot(highs, *label* = 'Highs', *color* = 'r')

    plt.plot(lows, *label* = 'Lows', *color* = 'b')

    # define the x-limit

    plt.xlim(1, 365)

    # label the graph

    plt.title('High and Low Temperatures')

    plt.xlabel('Day of the Year')

    plt.ylabel('Temperature (Celcius)')

    # add a legend to identify the graphs

    plt.legend()

    # add a grid

    plt.grid()

    # show the graph

    plt.show()

*def* july\_weather\_graph():

    '''

    () -> None

    Creates a graph of the weather data for July.

    Precondition: None

    '''

    # load the data

    data = load\_data()

    # create the figure

    plt.figure(4)

    # define the days in July

    days = np.arange(181, 212)

    highs = data[180:211,0]

    lows = data[180:211,1]

    # plot the highs and lows

    plt.plot(days, highs, *label* = 'Highs', *color* = 'r')

    plt.plot(days, lows, *label* = 'Lows', *color* = 'b')

    # define average highs and lows

    avg\_high = np.mean(highs)

    avg\_low = np.mean(lows)

    # represent the average highs and lows with a horizontal line

    plt.hlines(avg\_high, 181, 211, *colors* = 'r', *linestyles* = '--', *label* = *f*'Average High ({avg\_high*:.2f*})')

    plt.hlines(avg\_low, 181, 211, *colors* = 'b', *linestyles* = '--', *label* = *f*'Average Low ({avg\_low*:.2f*})')

    # label the graph

    plt.title('High and Low Temperatures in July')

    plt.xlabel('Day of the Year')

    plt.ylabel('Temperature (Celcius)')

    # identify the x-limit

    plt.xlim(181, 211)

    # change x-axis ticks

    xtick\_labels = np.arange(1, 32, 2)

    xtick\_values = np.arange(181, 212, 2)

    plt.xticks(xtick\_values, xtick\_labels)

    # define the standard deviations of the highs and lows

    high\_std = np.std(highs)

    low\_std = np.std(lows)

    # fill the area between the upper and lower standard deviations

    plt.fill\_between(days, avg\_high + high\_std, avg\_high - high\_std, *color* = 'r', *alpha* = 0.2)

    plt.fill\_between(days, avg\_low + low\_std, avg\_low - low\_std, *color* = 'b', *alpha* = 0.2)

    # add a legend to identify the graphs

    plt.legend(*fontsize* = 'small')

    # add a grid

    plt.grid()

    # show the graph

    plt.show()

1. <https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.fill_between.html> [↑](#footnote-ref-2)