Matplotlib Summary – Unit 5

**HW notes / ?s:**

update my Matplotlib to version 2.2 or newer…how do I check?

For all plots, you have 3 df’s: rural, suburban, & urban

Scatter plots 1.9-1.12

plt.scatter( arguments) done 3 x stacked on top of each other

3 plots in 1

See slacked ex of setting sizes of legend markers

Pie charts

Do it in notebook

For last pie chart, use city data dataframe vs. the merged df

drop\_duplicates is key

for not repeating the same city twice

lgnd = plt.legend(loc="lower left", scatterpoints=1, fontsize=10)

lgnd.legendHandles[0].\_sizes = [30]

lgnd.legendHandles[1].\_sizes = [30]

(http://Matplotlib.org/), one of the most popular Python plotting libraries in use today.

* `%matplotlib notebook` is used in a number of activities.

It not only makes a plot interactive, it also allows it to be updated after the initial plot. If students encounter weirdness during the activities, check to see that they use this line before importing the plotting libraries.

\* Ideally, students should update their Matplotlib to version 2.2 or newer. A known bug with earlier version resizes plots after exporting an image with the `savefig` method.

PyPlot – a module which we can use to create simple charts quickly

\* The NumPy library is oftentimes used alongside PyPlot.

This package contains plenty of built-in methods which allow for simple scientific computing.

visualizations of data are valuable for far more than aesthetics.

Trends and "human" insights buried within complex data sets are often clearest when the data is visualized in some way.

1.1 Basic Graphs

# Import Numpy for calculations and matplotlib for charting

import numpy as np

import matplotlib.pyplot as plt

# Create our x\_axis list

x\_axis = np.arange(0, 6, 0.1)

# `np.arange(start, end, step)` creates a list of numbers from `start` to `end`, where each number in the list is `step` away from the next ones.

# Creates a list based on the sin of our x\_axis values

sin = np.sin(x\_axis)

# Creates a list based on the cos of our x\_axis values

cos = np.cos(x\_axis)

# Plot both of these lines/curves so that they will appear on our final chart

plt.plot(x\_axis, sin)

plt.plot(x\_axis, cos)

plt.show()

# 2ND PLOT BELOW...

same start as above...

# Creates a list from 0 to 5 with each step being 0.1 higher than the last

x\_axis = np.arange(0, 5, 0.1)

x\_axis

# Creates an exponential series of values which we can then chart

e\_x = [np.exp(x) for x in x\_axis]

e\_x

# Create a graph based upon the two lists we have created

plt.plot(x\_axis, e\_x)

# Show the graph that we have created

plt.show()

# Give our graph axis labels

plt.xlabel("Time With MatPlotLib")

plt.ylabel("How Cool MatPlotLib Seems")

# Have to plot our chart once again as it doesn't stick after being shown

plt.plot(x\_axis, e\_x)

plt.show()

\* See the [MatPlotLib Documentation]

(<https://matplotlib.org/2.0.2/index.html>)

for more information regarding the PyPlot library

A large part of the process of developing plots with the library is reading [examples] (<http://Matplotlib.org/examples/index.html> )

\* Also look into the [NumPy Documentation]

(<https://docs.scipy.org/doc/numpy-dev/reference/>)

for more information on the NumPy library

ABOVE DIDN’T WORK…HERE MAYBE? <https://docs.scipy.org/doc/>

\*\*\* 1.2 NJ Temp

# Dependencies

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

# Set x axis to numerical value for month

x\_axis\_data = np.arange(1,13,1)

x\_axis\_data

# Average weather temp

points = [39, 42, 51, 62, 72, 82, 86, 84, 77, 65, 55, 44]

# Plot the line

plt.plot(x\_axis\_data, points)

plt.show()

# Convert to Celsius C = (F-32) \* 0.56

points\_C = [(x-32) \* 0.56 for x in points]

points\_C

# Plot using Celsius

plt.plot(x\_axis\_data, points\_C)

plt.show()

# Plot both on the same chart

plt.plot(x\_axis\_data, points)

plt.plot(x\_axis\_data, points\_C)

plt.show()

1.3 Configuring Line Plots

%matplotlib notebook

REMOVE THIS LINE WHEN WORKING IN NOTEBOOK (OLD)

TO SHOW THE RESULT OF EACH CELL

%matplotlib notebook

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

# Set x axis and variables

x\_axis = np.arange(0, 10, 0.1)

sin = np.sin(x\_axis)

cos = np.cos(x\_axis)

# Draw a horizontal line with 0.25 transparency

plt.hlines(0, 0, 10, alpha=0.25)

# `plt.hlines()` is used to draw a horizontal line.

# This method takes in three parameters: the Y value across which the line will be drawn, the X value where the line will start, and the X value where the line will end.

# The transparency of the horizontal line can also be set using the `alpha=` keyword and passing a number between 0 and 1. This setting is possible with most MatplotLib plotting functions.

# Assign plots to tuples that stores result of plot

# Each point on the sine chart is marked by a blue circle

sine\_handle, = plt.plot(x\_axis, sin, marker ='o', color='blue', label="Sine")

# Each point on the cosine chart is marked by a red triangle

cosine\_handle, = plt.plot(x\_axis, cos, marker='^', color='red', label="Cosine")

# Adds a legend and sets its location to the lower right

plt.legend(loc="lower right")

# Saves an image of our chart so that we can view it in a folder

plt.savefig("../Images/lineConfig.png")

plt.show()

1.4 Legendary Temperature

# Include this line to make plots interactive

%matplotlib notebook

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

# Set x axis to numerical value for month

x\_axis = np.arange(1,13,1)

x\_axis

# Avearge weather temp

points\_F = [39, 42, 51, 62, 72, 82, 86, 84, 77, 65, 55, 44]

# Convert to Celsius C = (F-32) \* 0.56

points\_C = [(x-32) \* 0.56 for x in points\_F]

points\_C

# Create a handle for each plot

fahrenheit, = plt.plot(x\_axis, points\_F, marker="+",color="blue", linewidth=1, label="Fahreneit")

celcius, = plt.plot(x\_axis, points\_C, marker="s", color="Red", linewidth=1, label="Celcius")

* NOTE how both fahrenheit and celsius are followed by commas to set them as tuples. This is crucial since plt.legend() expects to be handed tuples within its handles parameter and would return an error otherwise.

# Set our legend to where the chart thinks is best

plt.legend(handles=[fahrenheit, celcius], loc="best")

# Create labels for the X and Y axis

plt.xlabel("Months")

plt.ylabel("Degrees")

# Save and display the chart

plt.savefig("../Images/avg\_temp.png")

plt.show()

LEGEND & LABELS WERE SEPARATE AND LAST STEP DID NOT WORK…?

look through the [MatPlotLib Documentation]

(<https://matplotlib.org/2.0.2/index.html>)

to see what additional formatting could be added to the chart.

1.5 Aesthetics

%matplotlib notebook

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

# Generate the x values from 0 to 10 using a step of 0.1

x\_axis = np.arange(0, 10, 0.1)

sin = np.sin(x\_axis)

cos = np.cos(x\_axis)

# Add a semi-transparent horizontal line at y = 0

plt.hlines(0, 0, 10, alpha=0.25)

# Use dots or other markers for your plots, and change their colors

plt.plot(x\_axis, sin, linewidth=0, marker="o", color="blue")

plt.plot(x\_axis, cos, linewidth=0, marker="^", color="red")

# Add labels to the x and y axes

plt.title("Juxtaposed Sine and Cosine Curves")

plt.xlabel("Input (Sampled Real Numbers from 0 to 10)")

plt.ylabel("Value of Sine (blue) and Cosine (red)")

# Set your x and y limits

plt.xlim(0, 10)

plt.ylim(-1, 1)

# Set a grid on the plot

plt.grid()

# Save the plot and display it

plt.savefig("../Images/sin\_cos\_with\_markers.png")

plt.show()

ONLY SEEING GRID & HORIZ. LINE…& LAST STEP NOT WORKING…?

# TO 1ST LINE, AND THEN I SEE ALL PARTS SEPARATE…?

**Sankey diagrams** are a specific type of [flow diagram](https://en.wikipedia.org/wiki/Flow_diagram), in which the width of the arrows is shown proportionally to the flow quantity.

From Wikipedia

1.6 Roller Coaster

AXIS LABELS & TITLE DON’T WORK…?

%matplotlib notebook

# Import Dependencies

import matplotlib.pyplot as plt

import numpy as np

# Create the X and Y axis lists

time = np.arange(0,130,10)

speed\_chain = [9, 8, 90, 85, 80, 70, 70, 65, 55, 60, 70, 65, 50]

speed\_launch = [75, 70, 60, 65, 60, 45, 55, 50, 40, 40, 35, 35, 30]

# Plot the charts and apply some styling

danger\_drop, = plt.plot(time, speed\_chain, color="red", label="Danger Drop")

railgun, = plt.plot(time, speed\_launch, color="blue", label="RailGun")

# Add labels to X and Y axes :: Add title

plt.title("Coaster Speed Over Time")

plt.xlabel("Coaster Runtime")

plt.ylabel("Speed (MPH)")

# Set the limits for the X and Y axes

plt.xlim(0,120)

plt.ylim(5,95)

# Create a legend for the chart

plt.legend(handles=[danger\_drop, railgun], loc="best")

# Add in a grid for the chart

plt.grid()

plt.show()

Matplotlib provides a simple interface for producing common charts including line plots, line charts; bar charts; pie charts; and scatter plots.

* **bar charts** are good for comparing different entities to one another.
* **pie charts** are good for displaying parts of a whole — in particular, to what extent different constituents of a whole contribute to that whole.
* **scatter plots** are good for displaying where pts fall w respect to 2 different factors.

Choose the right plot for a given data set, or the graphic may be less readable or may even make the data misleading.

1.7 Bar Charts

necessary to provide the heights of each bar within an array.

x-axis will also be an array whose length must equal that of the list of heights.

Instead of using plt.plot() bar charts are drawn using plt.bar().

The align parameter for plt.bar() is center to center.

%matplotlib notebook

import matplotlib.pyplot as plt

import numpy as np

# Create an array that contains the number of users each language has

users = [13000, 26000, 52000, 30000, 9000]

x\_axis = np.arange(len(users))

# Tell matplotlib that we will be making a bar chart

# Users is our y axis and x\_axis is, of course, our x axis

# We apply align="edge" to ensure our bars line up with our tick marks

plt.bar(x\_axis, users, color='r', alpha=0.5, align="center")

# Tell matplotlib where we would like to place each of our x axis headers

tick\_locations = [value for value in x\_axis]

plt.xticks(tick\_locations, ["Java", "C++", "Python", "Ruby", "Clojure"])

# Sets the x limits of the current chart

plt.xlim(-0.75, len(x\_axis)-0.25)

# Sets the y limits of the current chart

plt.ylim(0, max(users)+5000)

# Give our chart some labels and a tile

plt.title("Popularity of Programming Languages")

plt.xlabel("Programming Language")

plt.ylabel("Number of People Using Programming Languages")

\*\*\* 1.8 Py Bars

%matplotlib notebook

import matplotlib.pyplot as plt

import numpy as np

cities = ["New Orleans", "Milwaukee", "Omaha", "Pittsburgh", "Toledo"]

bars\_in\_cities = [8.6, 8.5, 8.3, 7.9, 7.2]

x\_axis = np.arange(len(bars\_in\_cities))

# Create a bar chart based upon the above data

plt.bar(x\_axis, bars\_in\_cities, color="b", align="center")

# Create the ticks for our bar chart's x axis

tick\_locations = [value for value in x\_axis]

plt.xticks(tick\_locations, cities)

plt.bar(x\_axis, bars\_in\_cities, color="r", align="center")

# Set the limits of the x axis

plt.xlim(-0.75, len(x\_axis)-0.25)

# Set the limits of the y axis

plt.ylim(0, max(bars\_in\_cities)+0.4)

# Give the chart a title, x label, and y label

plt.title("Density of Bars in Cities")

plt.xlabel("Cities")

plt.ylabel("Bars Per 10,000 Households")

# Save an image of the chart and print it to the screen

plt.savefig("../Images/BarDensity.png")

plt.show()

1.9 Pie Charts

**### Key Activities**

\* [Scatter Plots](1/Activities/12-Stu\_ScatterPy)

\* [Pandas Plotting](2/Activities/03-Stu\_BattlingKings)

\* [Plotting Groups](2/Activities/05-Stu\_BikeTrippin)

\* [Plotting Multi-lines](2/Activities/07-Ins\_PandasMultiLine)

\* [Mean, Median, Mode](3/Activities/01-Ins\_Mean\_Median\_Mode)

\* [Variance](3/Activities/02-Ins\_Variance\_and\_Z\_Score)

\* [Quartiles](3/Activities/04-Stu\_Quartiles\_and\_Outliers)

\* [Standard Error](3/Activities/06-Stu\_Standard\_Error)

**## Objectives**

\* Understand Matplotlib's pyplot interface.

\* Be able to create line; bar; scatter; and pie charts.

\* Be familiar with basic plot configuration options, such as `xlim` and `ylim`.

\* Feel comfortable creating plots using the `DataFrame.plot()` method.

\* Understand the advantages and disadvantages of creating charts using the `DataFrame.plot()` method.

\* Be able to work through a complex data set using Pandas and then chart some visualizations based upon the cleaned DataFrame.

\* Be able to define **\*\*mean\*\***, **\*\*median\*\***, and **\*\*mode\*\***, and choose which one is most appropriate to describe a given data set.

\* Be able to explain the meaning of variance and standard deviation.

\* Be able to describe standard error and the difference between a sample and a population.

\* Be able to add error bars to their plots.

\* Be able to fit lines to their data.

**### Helpful Links**

\* [Numpy](http://www.numpy.org/)

\* [Matplotlib Gallery](https://matplotlib.org/gallery.html)

\* [Pandas Plotting](https://pandas.pydata.org/pandas-docs/stable/visualization.html)

\* [Alternatives to Pie Charts](http://www.storytellingwithdata.com/blog/2014/06/alternatives-to-pies)