Matplotlib

Matplotlib

- Visualizing data is crucial to quickly understanding trends and relationships in your dataset.
- Matplotlib is one of the most popular libraries for plotting with Python.

Matplotlib

- Matplotlib is known as the “Grandfather” of plotting and visualization libraries for Python.
- Many other visualization libraries are built directly off of Matplotlib (e.g. seaborn and pandas built-in visualization).

Matplotlib

- Matplotlib is heavily inspired by the plotting functions of the MatLab programming language.
- It allows for the creation of almost any plot type and heavy customization.

Matplotlib

- This ability to heavily customize a plot comes at a trade-off for beginners, since it can be confusing to learn the Matplolib syntax at first.
- This is mainly due to the fact that there are actually two separate approaches to creating plots, functional based methods and OOP based methods.

Matplotlib

- This Matplotlib section seeks to clear up any confusion by clearly separating out these two approaches.
    - Matplotlib Basics
        - Functional Method
    - Matplotlib Figures and Subplots
        - OOP Method

Matplotlib

- Topics Covered
    - Matplotlib Basics and Functions
    - Matplotlib Figures
    - Matplotlib Subplots
    - Matplotlib Styling
    - Exercise Questions and Solutions

Matplotlib

- Specialized plot types such as histograms won’t be covered with matplotlib, since we will later learn how to use seaborn to easily create statistical plots.
- It is important to learn matplotlib first however, since seaborn builds directly off of Matplotlib.

Matplotlib

- Throughout this section we will be referencing the excellent Matplotlib online documentation:
    - https://matplotlib.org/
    - As well as the gallery of example plots and codes (*very useful!*)
        - https://matplotlib.org/gallery.html

Matplotlib

- Two main goals with Matplotlib:
  - Be able to plot out a functional relationship:
    - y = 2x
  - Be able to plot out a relationship between raw data points:
    - x = [1,2,3,4]
    - y = [2,4,6,8]

Let’s get started!

Matplotlib Basics

PIERIAN DATA

Matplotlib

- The most basic way to use Matplotlib is through the function plot calls:
    - **plt.plot(x,y)**
- These function calls are simple to use, but don’t allow for very high degrees of control.

Matplotlib

- We recommend using these simple plt.plot() calls for quickly visualizing relationships and data.
- Later on we will explore the more robust OOP Matplotlib Figure API.

Matplotlib

- Note!
  - There are slight differences in displaying plots within a notebook versus running a python script.
  - If you are running .py scripts instead of .ipynb notebooks, you will need to add the plt.show() command discussed in this video.

Matplotlib Figure Object

PART ONE : UNDERSTANDING THE FIGURE

PIERIAN DATA

Matplotlib

- The more comprehensive Matplotlib OOP API makes use of a Figure object.
- We then add axes to this Figure object and then plot on those axes.
- This allows for very robust controls over the entire plot.

Matplotlib

- Let’s quickly visually build an understanding of the Figure object before coding it with Python...
- Note:
    - The Figure object we’re about to show is technically not visible until you add axes to it.

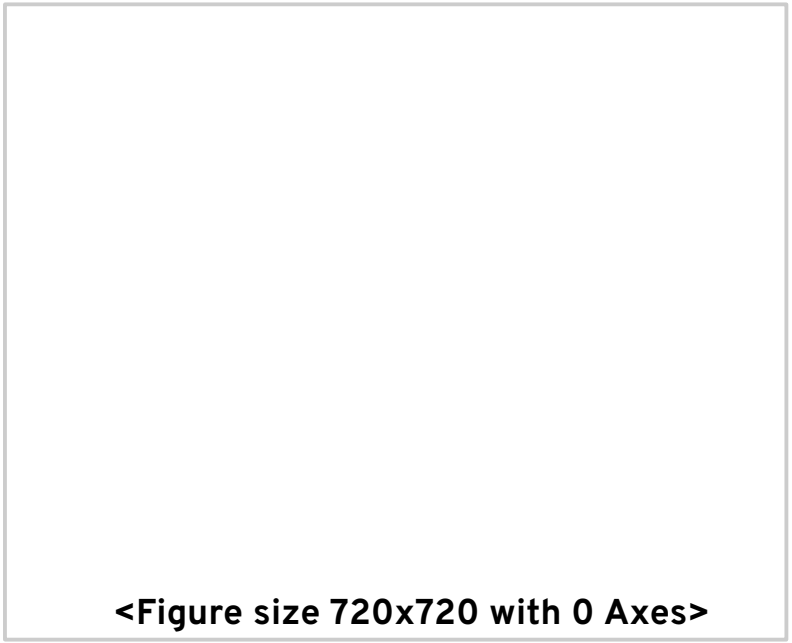
Matplotlib

- plt.figure()

PIERIAN DATA

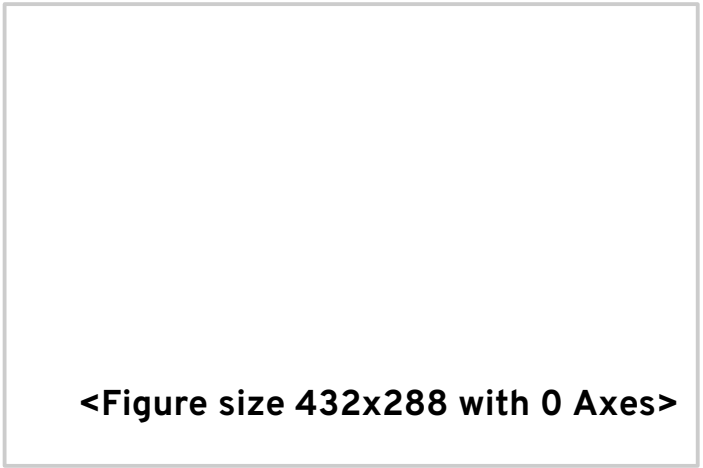
Matplotlib

- plt.figure(figsize=(10,10))



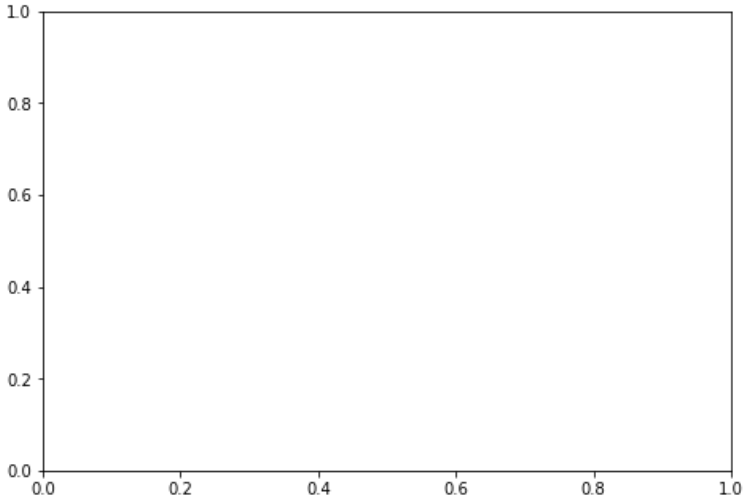
Matplotlib

- fig = plt.figure()
- Blank canvas, waiting for a set of axes for plotting.



Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0, 0, 1, 1])



Matplotlib

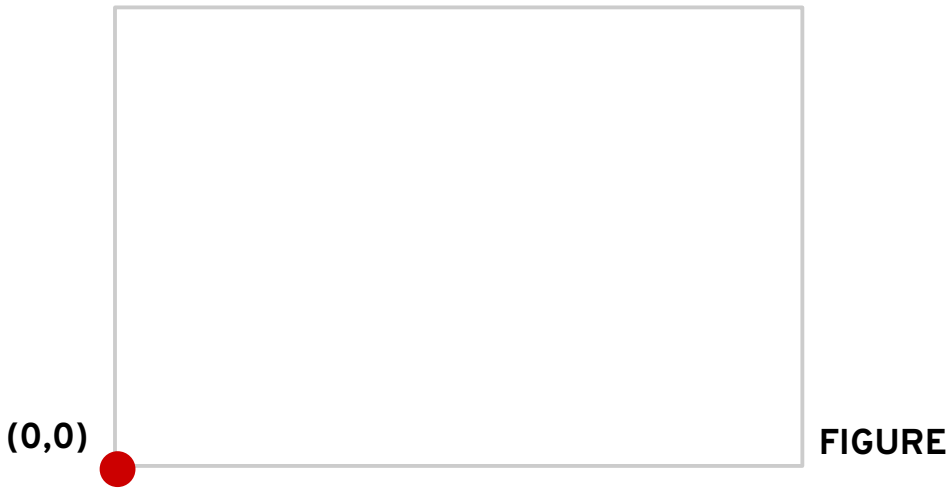
- fig = plt.figure()
  axes = fig.add_axes([0, 0, 1, 1])

FIGURE

Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0, 0, 1, 1])

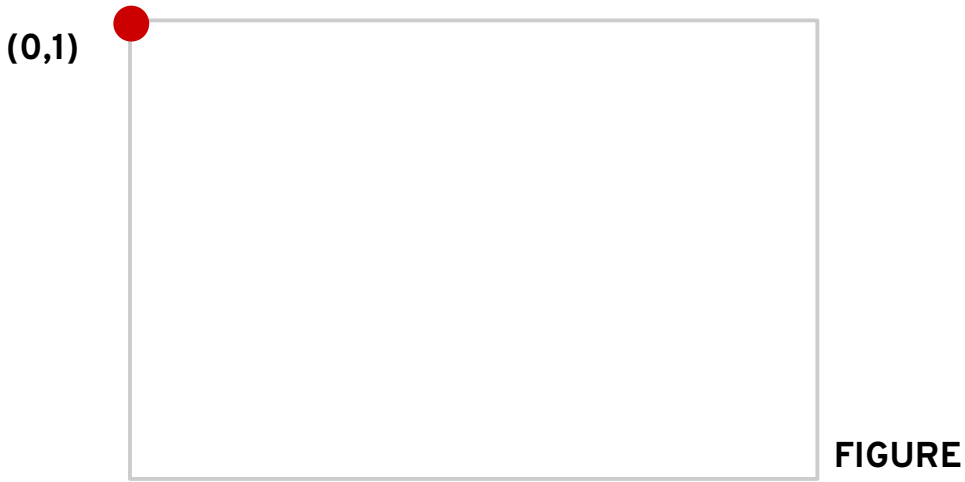
(x,y)
Lower Left Corner of Axes



Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0, 1, 1, 1])

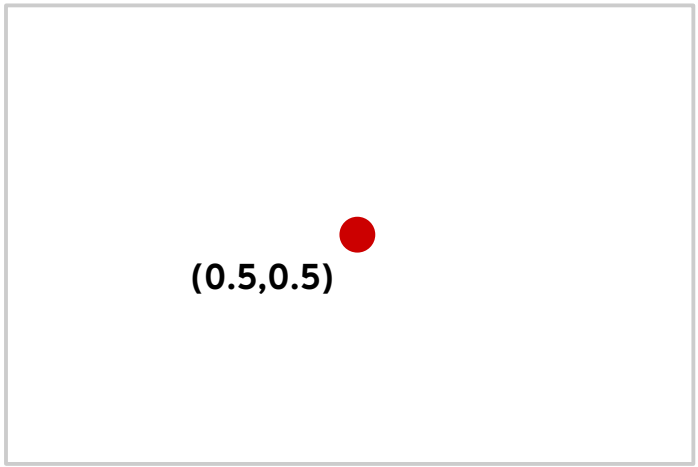
(x,y)
Lower Left Corner of Axes



Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0.5, 0.5, 1, 1])

(x,y)
Lower Left Corner of Axes



FIGURE

Matplotlib

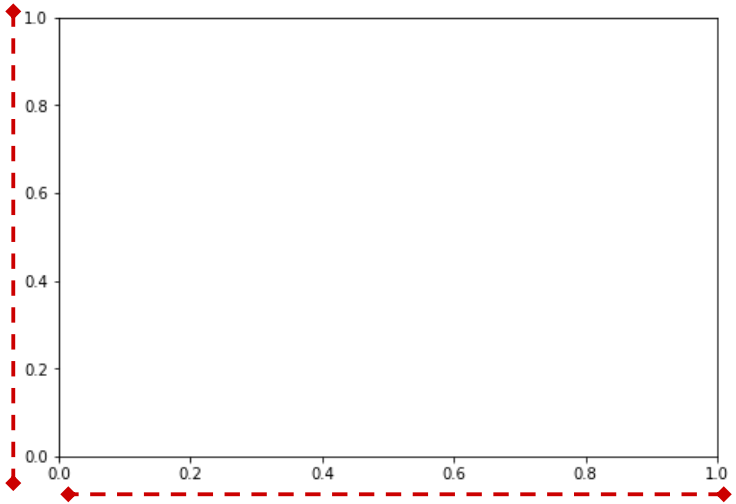
- fig = plt.figure()
  axes = fig.add_axes([0, 0, 1, 1])

(width,height)
of Axes

Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0, 0, 1, 1])

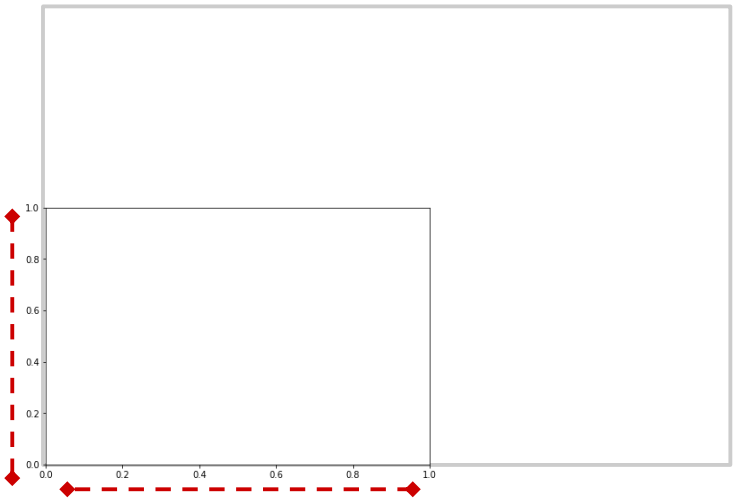
(width,height)
of Axes



Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0, 0, 0.5, 0.5])

(width,height)
of Axes



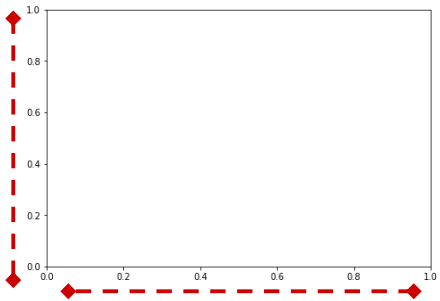
FIGURE

PIERIAN DATA

Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0, 0, 0.5, 0.5])

(width,height)
of Axes

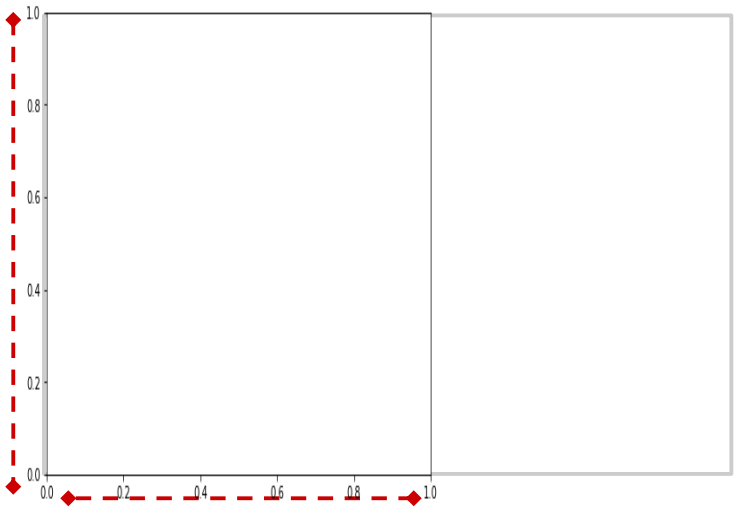


PIERIAN DATA

Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0, 0, 0.5, 1])

(width,height)
of Axes

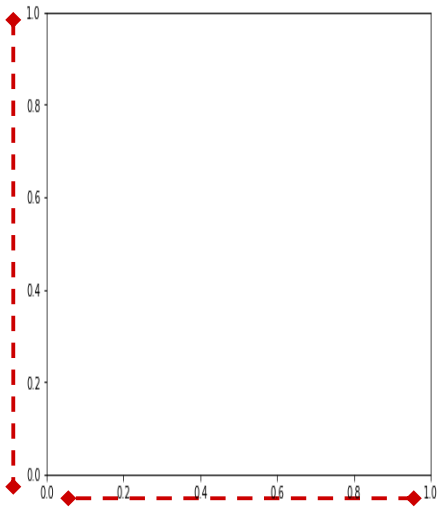


FIGURE

Matplotlib

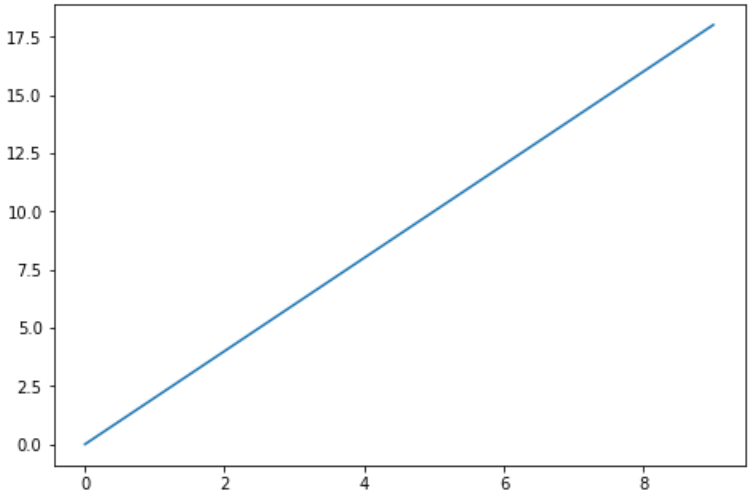
- fig = plt.figure()
  axes = fig.add_axes([0, 0, 0.5, 1])

(width,height)
of Axes



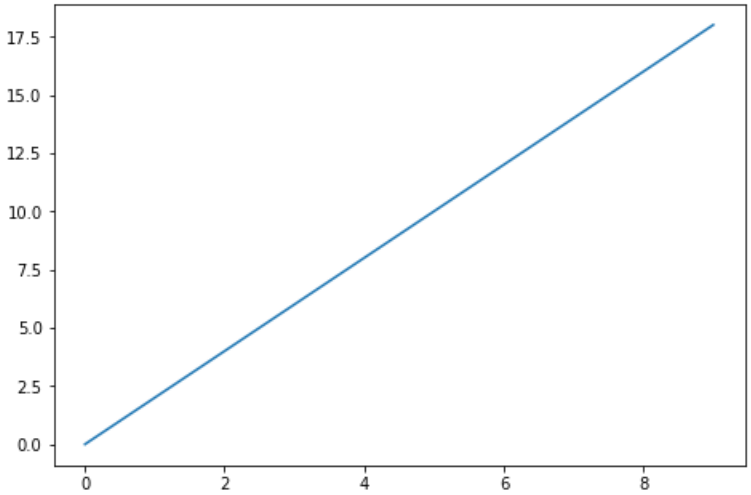
Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0, 0, 1, 1])
  axes.plot(x, y)



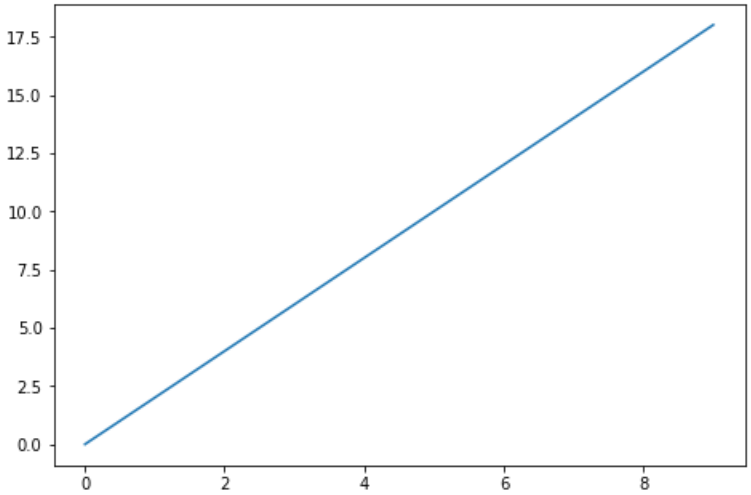
Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0, 0, 1, 1])
  axes.plot(x, y)



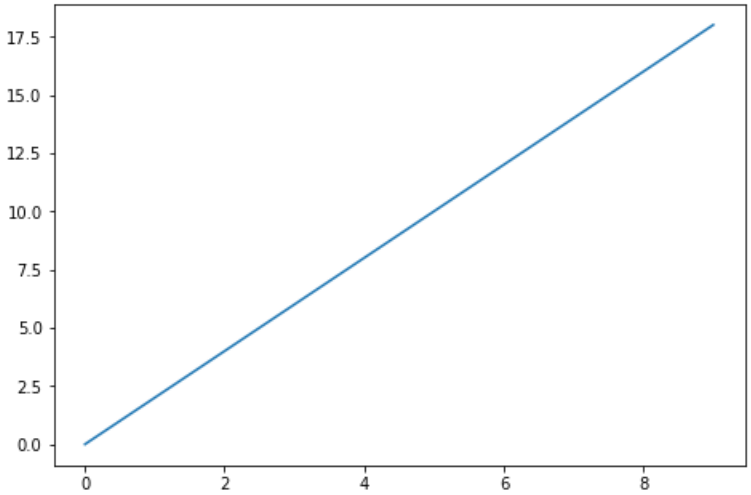
Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0, 0, 1, 1])
  axes.plot(x, y)



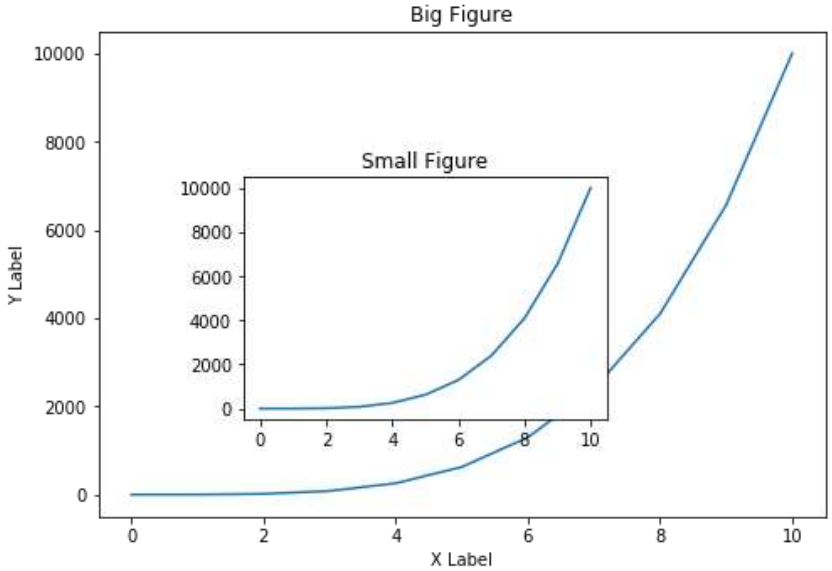
Matplotlib

- fig = plt.figure()
  axes = fig.add_axes([0, 0, 1, 1])
  axes.plot(x, y)



Matplotlib

- This methodology allows us to add in multiple axes as well as move and resize the axes.



Matplotlib

- In theory we could set axes side by side using plt.figure() calls, but typically it is easier to use plt.subplots() function calls for this.
- We'll explore multiple side by side plots in a future lecture, for now let's explore the Figure object methodology for Matplotlib!

Matplotlib Figure Object

PART TWO: IMPLEMENTING FIGURES AND AXES

PIERIAN DATA

Matplotlib Figure Object

PART THREE: FIGURE PARAMETERS

PIERIAN DATA

Matplotlib SubPlots

PIERIAN DATA

Matplotlib

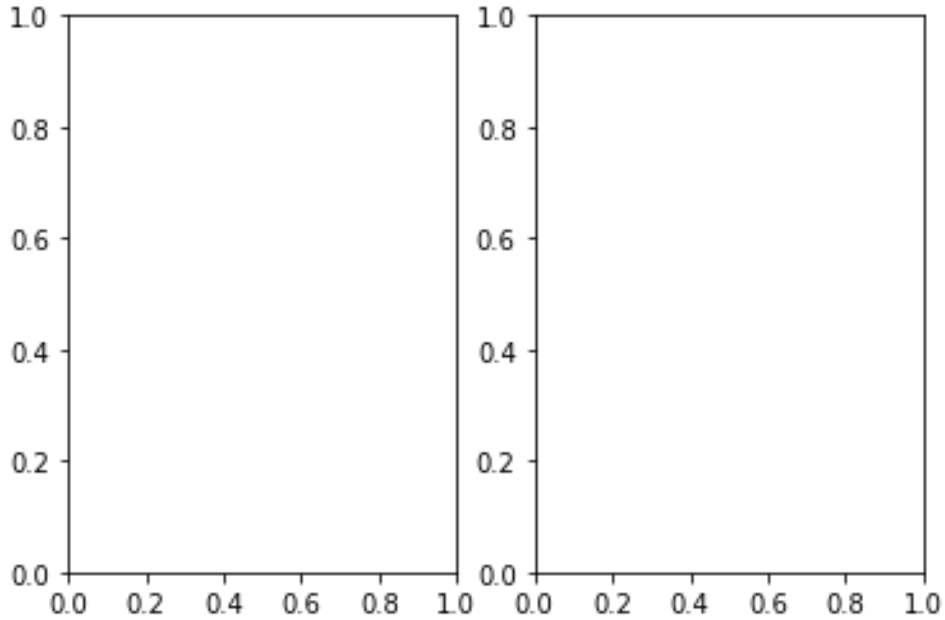
- In theory we could create a Figure object and then manually add and arrange sets of axes to line up multiple plots side by side.
- However, Matplotlib comes with a pre-configured function call **plt.subplots()** that automatically does this for us!

Matplotlib

- The plt.subplots() call allows us to easily create Figure and Axes objects in side by side formations.
- The plt.subplots() command returns a tuple containing the Figure canvas and then a numpy array holding the axes objects.

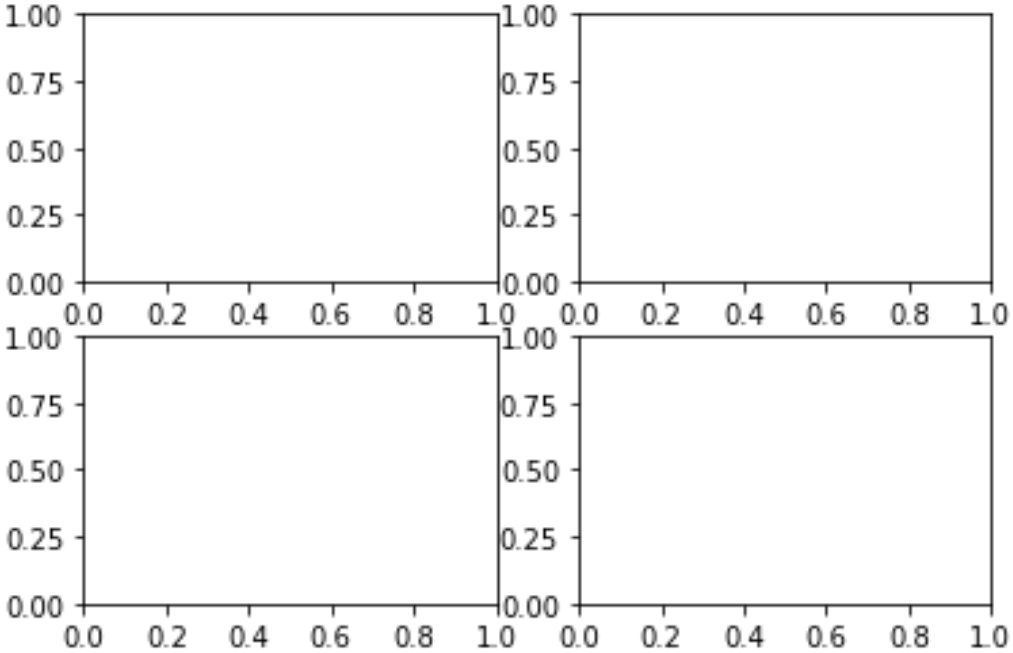
Matplotlib

- fig, axes = plt.subplots(nrows=1, ncols=2)



Matplotlib

- fig, axes = plt.subplots(nrows=2, ncols=2)



Matplotlib

- plt.subplots() returns a tuple which by common convention we label **(fig,axes):**
  - **fig**
    - This is the entire Figure canvas.
  - **axes**
    - This is a numpy array holding each of the axes according to position in the overall canvas.

Matplotlib

- Let’s explore how to use plt.subplots() to easily create and align multiple plots!

Matplotlib Styling

PART ONE: LEGENDS

PIERIAN DATA

Matplotlib

- Matplotlib offers very robust styling functions that allow us to edit colors, legends, line widths, markers, and much more!
- Note: Due to the wide amount of possible variations, we will be copying and pasting from the lecture notebook to save typing time in the video.

Matplotlib

- Main Styling Discussed:
    - Legends
    - Visual Styling
        - Colors
        - Editing Lines
            - Colors, Widths, Styles
        - Editing Markers
            - Colors, Size, Styles, Edges

Matplotlib

- Let’s begin with adding legends!

Matplotlib Styling

PART TWO: VISUAL STYLING

PIERIAN DATA

Additional Matplotlib Commands

Matplotlib

- Matplotlib is a huge library!
- We've added a notebook with some additional concepts you may want to explore on your own.
- We don't use these concepts in the course however, so feel free to skip this notebook for now.

Matplotlib

- An important note is that almost any Matplotlib question you can think of already has an answer in StackOverflow or an example in the Matplotlib gallery.
- Leverage these many examples to your advantage and do not waste energy and time into memorizing esoteric commands!

Matplotlib

- Section

Matplotlib Exercises Solutions

PIERIAN DATA