

Data Visualization Basics in Python

Disclaimer: These notes and examples are an adaptation of the references listed at the end. They are compiled to fit the scope of this specific course.

Introduction

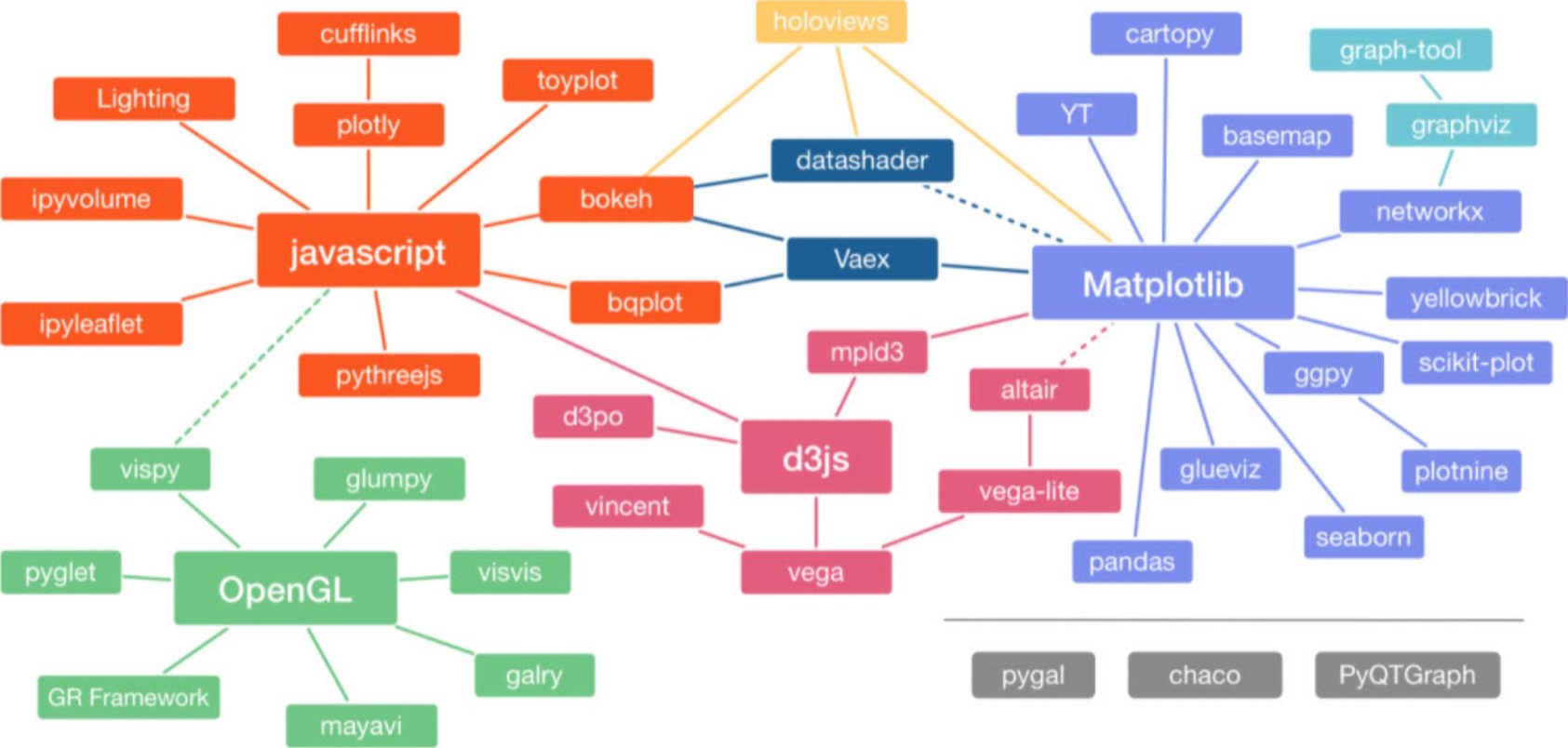
Python offers a variety of tools and packages for visualization. The following is a list of popular packages available in python for visualization

* matplotlib
* seaborn  pandas
* altair
* bokeh
* plot ly

Understanding which package to use, how to use it, and what graph to use for visualizing your data can be challenging. In this introductory handout, we will learn about matplotlib , seaborn , and pandas , three of the most commonly used visualization packages in python.

Python visualization landscape

The https://pyviz.org/index.html website provides an accurate image of python's visualization tools. This platform helps users decide on the best python data visualization tools and their purposes with links, overviews, comparisons, and examples. The most important part of this landscape is noticing the libraries' inner dependencies. The following graphics is taken from https://pyviz.org/overviews/index.html.



As you can see, many python visualization libraries heavily depend on matplotlib 

matplotlib

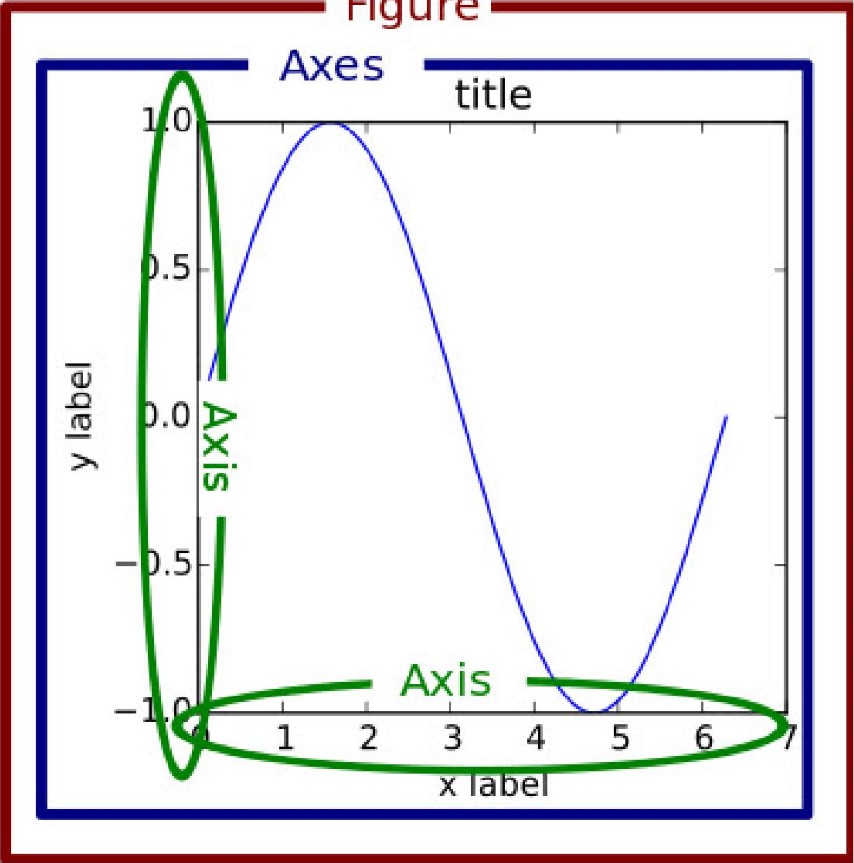
matplotlib is the grandfather of python visualization tools! As it is clear from the above landscape, many of python's visualization tools most notably seaborn are just a user-friendly skin (wrapper) over matplotlib . matplotlib is very powerful and complicated. Even though you can accomplish most of your visualization tasks with matplotlib , many users try to avoid using matplotlib due to its complexity. However, due to different packages' dependency on matplotlib , we will learn the basics before switching to seaborn and visualization using pandas 

plotting with matplotlib

matplotlib (https://matplotlib.org/) is python's most widely used visualization library. The Example section of this library is available at https://matplotlib.org/stable/gallery/index.htmllts shows a variety of graphs you can plot using matplotlib . Jupyter notebook and python source code for each of these examples are available for download in each example web page.

For anaconda users, matplotlib is already available. However, try pip install matplotlib if you need to install it. Let's understand the anatomy of a plot. Key components of a plot are Figure and Axes .

* Figure : A Figure is the top-level container for everything plotted. You can consider Figure to be a piece of paper or a canvas you can draw and write on.
* Axes : Contained within a Figure is one or more Axes . An Axes is a plotting surface. An Axes is where we plot our data, and add all the associated labels/ticks. Each Axes has one X-Axis and one Y-Axis (and possibly one Z-Axis ). The following shows the anatomy of a plot.



Figure

Note 1: Axes is NOT a plural of Axis ; they are entirely separate concepts. An Axes is the plotting surface and may contain X-Axis , YAxis , or Z-Axis 

Note 2: Please do not confuse the concept of axis in numpy , pandas , and Axes in matplotlib as they are very different concepts. In numpy and pandas , the setting of axis = or axis = 1 is used to dictate calculations for columns or rows; whereas, in matplotlib Axes is a plotting area and Axis is for X, Y, or Z axis.

Plotting in matplotlib is done using its pyplot module. matplotlib suggests importing pyplot using import matplotlib.pyplot as plt

* Older versions of Jupyter notebook users: you need to include the line %matplotlib inline in your notebook. This will direct matplotlib to display plots in the notebook.
* If you are using other IDES such as pycharm or spyder , you need to add plt.show() instead.

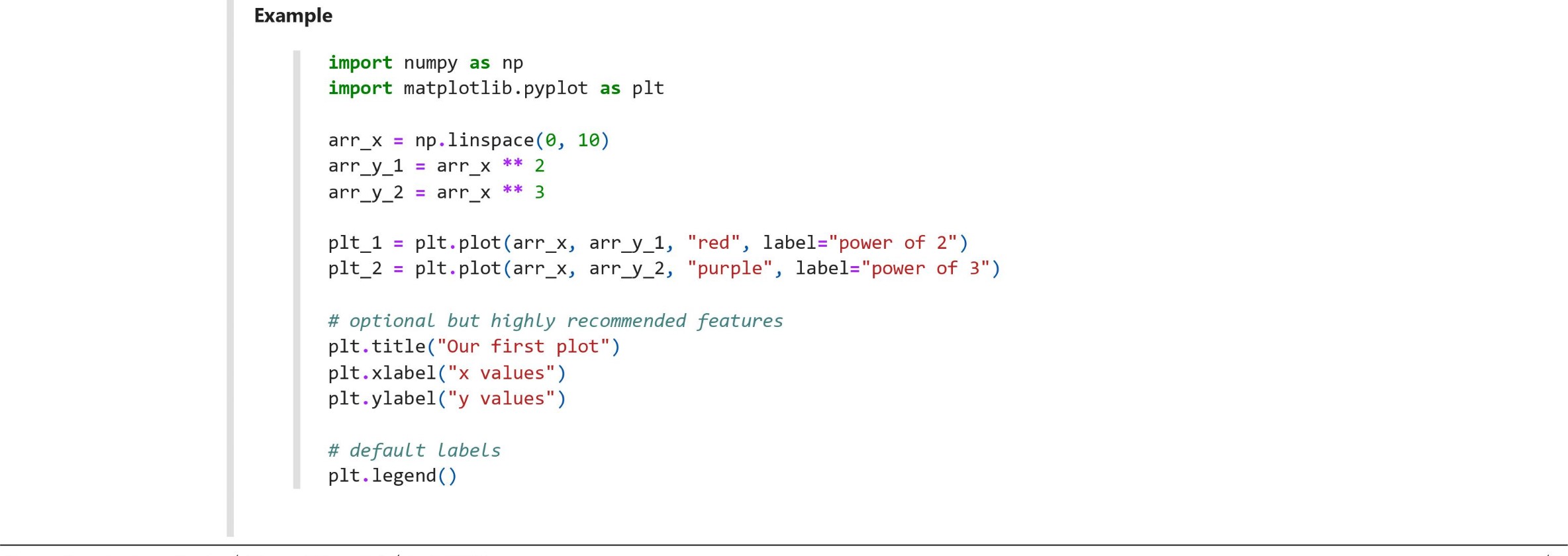
There are two approaches for creating plots in matplotlib as

* functional approach
* Object-oriented approach

Rule of thumb: Almost all matplotlib plot types (linear, bar, histogram, etc.) accept arrays ( python arrays, numpy arrays, pandas Series , or pandas DataFrame ) for values of x and y 

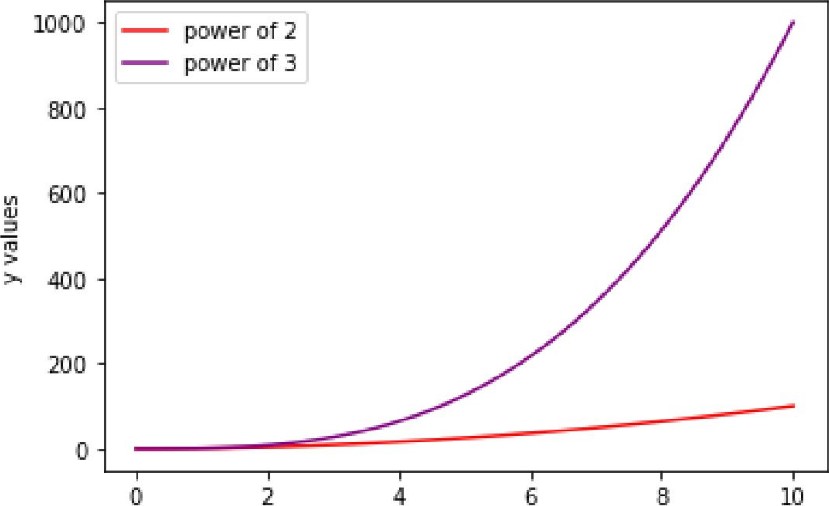
### Use matplotlib functional approach for plotting

Here is an example:



yields a plot with two lines plotted. Legend will also be shown; the default plot legend is the provided plot labels.

Our first plot



2

6

8

10

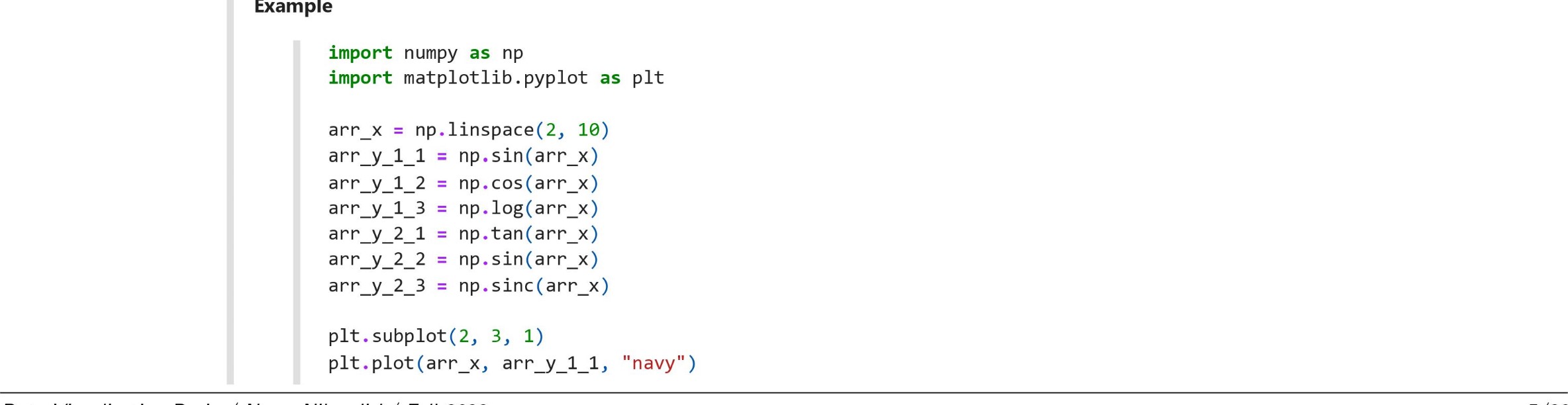
x values

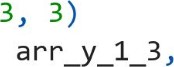
matplotlib default is plotting just one plot on the canvas. We can create multi-plots on the same figure (canvas) using . subplot method. This method has three arguments:

* nrows : number of rows in the Figure
* ncols : number of columns in the Figure  plot\_number : refers to a specific plot in the Figure

The following snippet will create six subplots, plot different functions in each subplot.

I'll



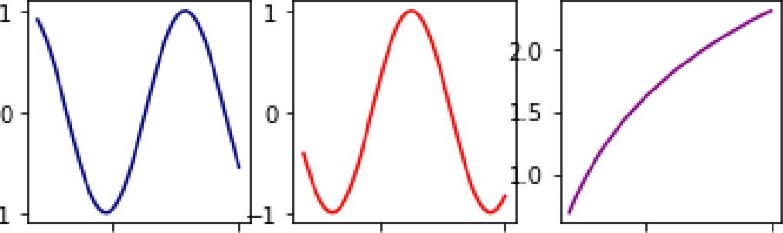
plt. subplot (2, plt. plot (arr\_x, arr—Y 1 2\_ , "red")

plt. subplot (2, plt. plot (arr\_x,"purple")

plt. subplot (2, plt. plot (arr\_x, arr—Y\_ 2\_1 "black")

plt. subplot (2, plt.plot (arr\_x, arr\_y 2 2, ' green " )

plt.subp10t(2, 3, 6) plt.plot (arr\_x, arr\_y 2 3, " orange " ) yields six plots in one Figure 

-1

5 10 5 10 5 10

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | 1  1 |  |  |  |  |
|

-20

—40

.1

5 10 5 10 5 10

Note: If your plots are not showing, please add the magic function %matplot inline to your cell. So the top part of your cell will look like

import numpy as np import matplotlib. pyplot as plt %matplotlib inline

### Use matplotlib object oriented approach for plotting

This is the most common way of creating plots. The main idea is to create a Figure object, then make one or more Axes in that Figure object, and call the necessary methods off the Figure and Axes to customize the contents of a plot.

### Use plt.figure()

Here is an example

Example

import numpy as np import matplotlib. pyplot as plt

# create data arr\_x = np.1inspace(2, 10) arr\_y\_l = np.sinc(arr\_x)

#create a figure. This is similar to a canvas surface fig\_l = PIt . figure()

# create an empty plot. We call it axes axes\_l =  0.2, 0.4, 0.5])

# plot in axes\_l axes\_l .  arr\_y\_l, " red " )

# set axes\_l settings axes\_l . set( xlim=(l, 11),

## 0.15),

xlabel="value of x", ylabel="value of y" , title:" axes title" ,

creates a figure called fig\_l ; adds an Axes called axes\_l at = 0.1 , = 0.2 , width = 0.4 , and height =

0.5 ; plots a red curve, and sets specific settings for the plot

axes title

# 0.15

0.10

0.05

o

 0.00

3 > -0.05

-0.10

-0.15

2 4 6 8 10 value of x

Explanation:

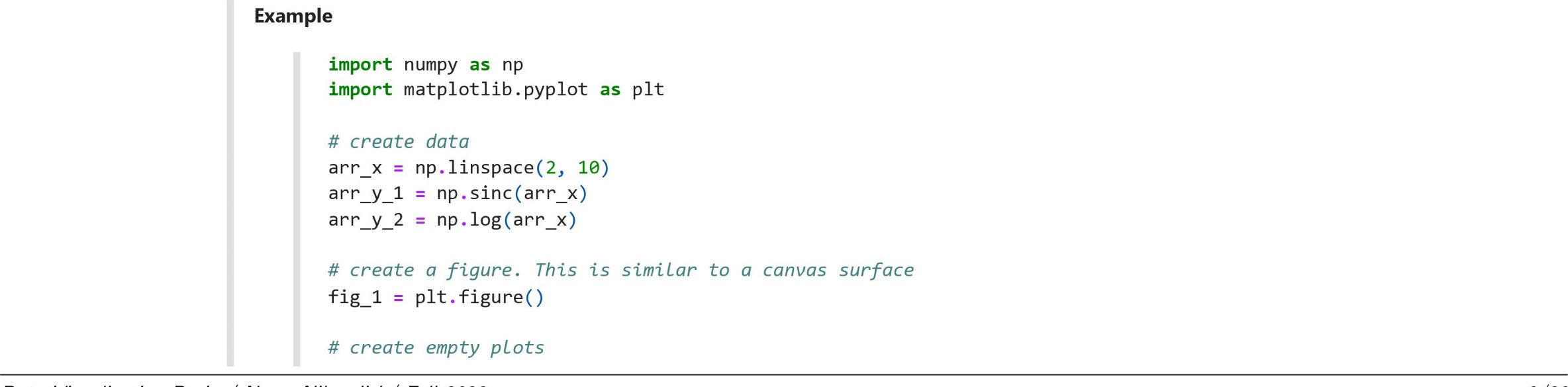
* fig\_l = plt.figure() creates a Figure object and calls it fig\_l . Figure coordinates go 0 to 1 , where (0, e ) is the lower left corner and (1, 1) is the upper right corner. A coordinate of .05 is hence slightly outside the figure.

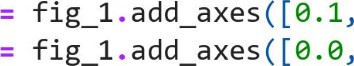
|  |  |  |  |
| --- | --- | --- | --- |
| Figure (0.5,1)   |  |  |  | | --- | --- | --- | | Axes |  | Axe s |   (o,o.5)  (0,0) (1,0) |
|  |

(1,1)

* axes\_l = fig\_1.add\_axes( [0.1, 0.2, 0.4, 0.5]) creates an Axes object. The syntax of add\_axes([x\_e, y\_Ø, w, h]) creates an Axes ; (x\_0, y\_0) is the lower left point of the new axes in figure coordinates, w is its width and h is its height. So the axes are positioned in absolute coordinates on the canvas.
* axes\_l.  arr\_y\_l, "red") plots a red curve based on the given data points.
* axes\_l. set() method is used to set axes\_l settings.

Adding one more Axes: In the following snippet, we will add one more Axes to the same canvas ( Figure object) and plot two different functions. This time we will add more settings for the Figure , and finally, we will save it.



axes 1 axes 2

# plot in axes 1 axes 1.plot (arr\_x, arr-y\_l,

# plot in axes 2 axes 2.plot(arr\_x, arr-Y\_2,

# set axes\_l settings axes\_l . set( xlim=(l, 11),

• 0.15, 0.15), xlabel="value of x 1" , ylabel="value of y\_1" , title="axes 1 title",

# set axes 2 settings axes \_2. set( xlim=(l, 11), ylim=(0.5, 3), xlabel="value of x 2" , of y\_2" , title="axes 2 title",

# figure settings

0.0, 0.4, 0.25])

0.5, 0.4])

" red " )

'purple")

fig\_l.suptitle("figure title" , size=20, x=e.2, y=l.l)

in this snippet, we will add two Axes to the Figure object; customize their settings; and finally, add a figure title with a font size of 20 at coordinates of (0.2, 1.1) relative to the lower left corner of the canvas

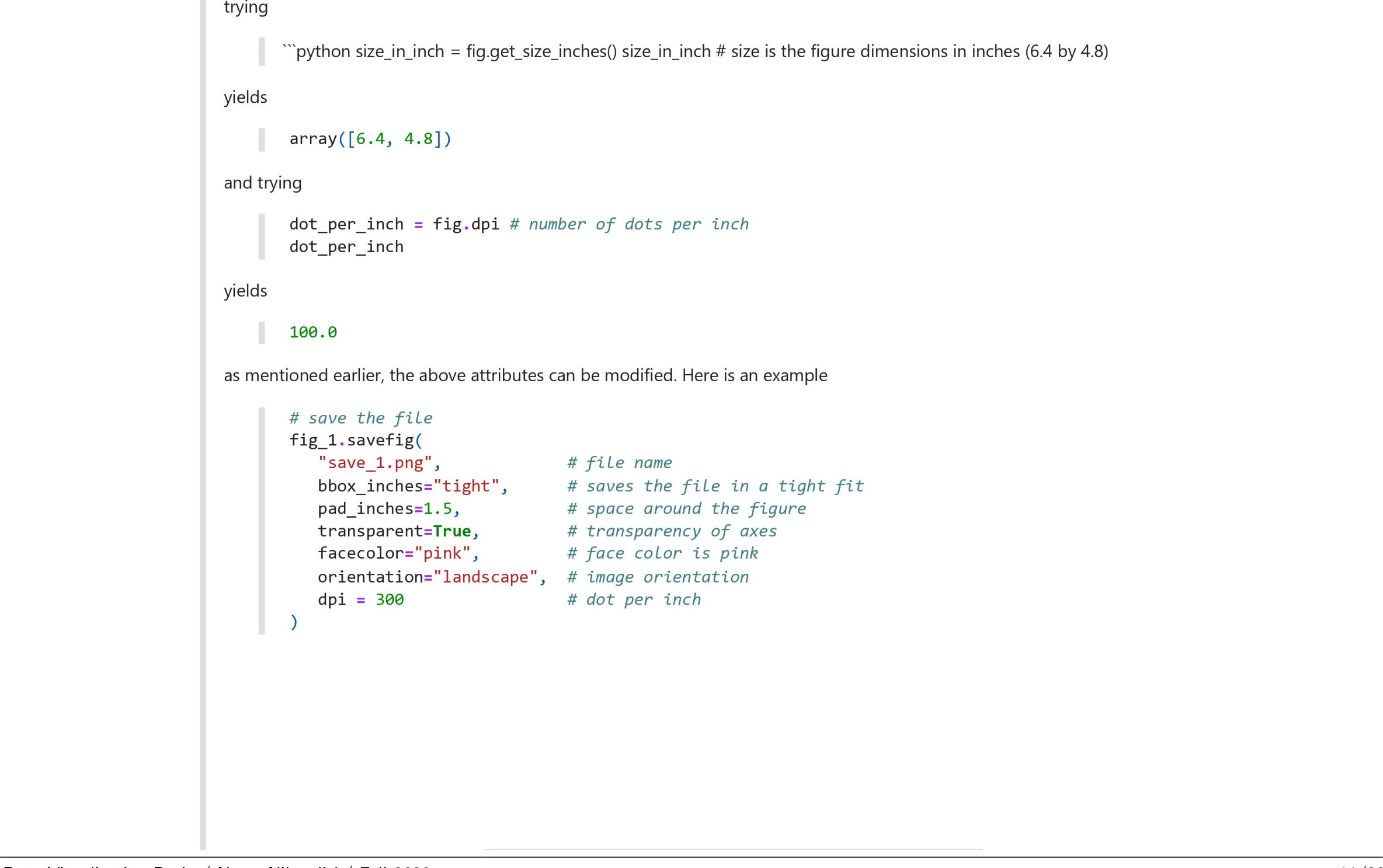
The default size of a canvas ( Figure ) in matplotlib is (6.4, 4.8) inches (6.4 inches in width and 4.8 inches in height) which is not the actual size of it on the screen but would be the exact size if you saved the Figure to a file (with a dpi of 100 pixels per inch); we can confirm that by calling fig\_l . get\_size\_inches() . We can further customize that by calling fig\_l . set\_size\_inches (12, 5) . Additionally, you can save an image using savefig() method, as well. The following will illustrate these points

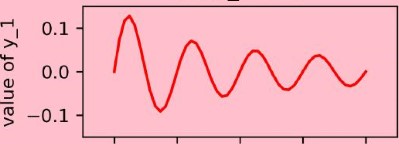
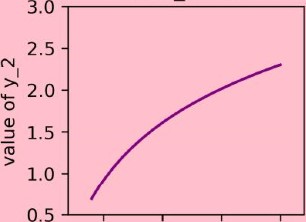


yields

 <Figure size with 0 Axes>

indicating that, by default, fig has 6.4 >< 100 640 pixels across the x-axis and 4.8 >< 100 — 480 pixels across the y-axis.



figure title

axes 2 title

2.5 5.0 7.5 10.0 value of x 2 axes 1 title

2 4 6 8 10 value of x I

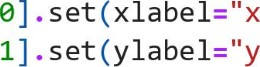
• dpi stands for dot per inch. This value dictates the figure resolution in dots per inch; a higher number means higher resolution ( and bigger image file size). For most applications, dpi in the range of 200 to 400 suffices.

Use plt. subplots(nrows, ncols)

We can create multiple plots in the object-oriented approach using the . subplots() method and NOT . subplot() . The . subplots() method takes two arguments, nrows , for the number of rows in the Figure and ncols , for the number of columns in the Figure .



arr\_x = np.1inspace(2, 10) arr = np.cos(arr\_x) arr — np.tan(arr\_x) fig, axes - plt.subp10ts(nrows=2, nc01s=3)

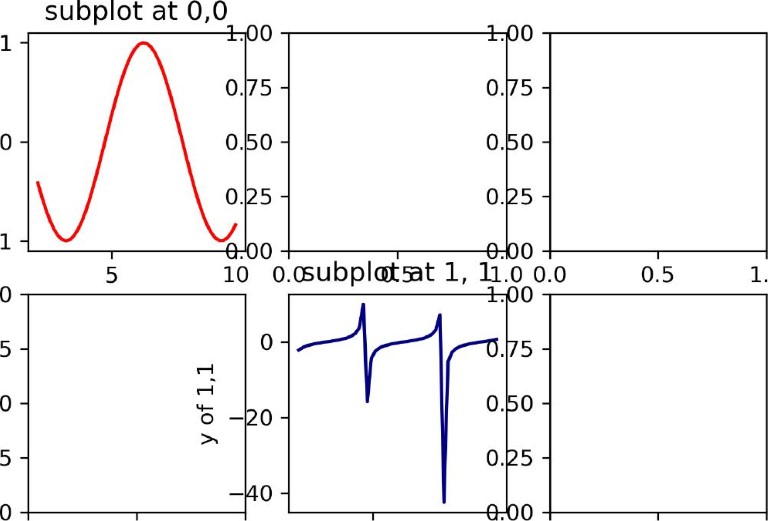


"y

axes [0, 0] . arr\_y\_l, "red") axes [1, 1] . arr\_y\_2, "navy")

axes [0, of e, 0", title:" subplot at e, 0") axes [1, 1] of 1, 1", title:" subplot at 1, 1")

fig. suptit1e("This is super title" , size--20, x=e.5, y-1.05)



This

is

super

title

0.00

0.0

0.5

1.0

5

10

0.0

0.5

1.0

—1

1.00

0.75

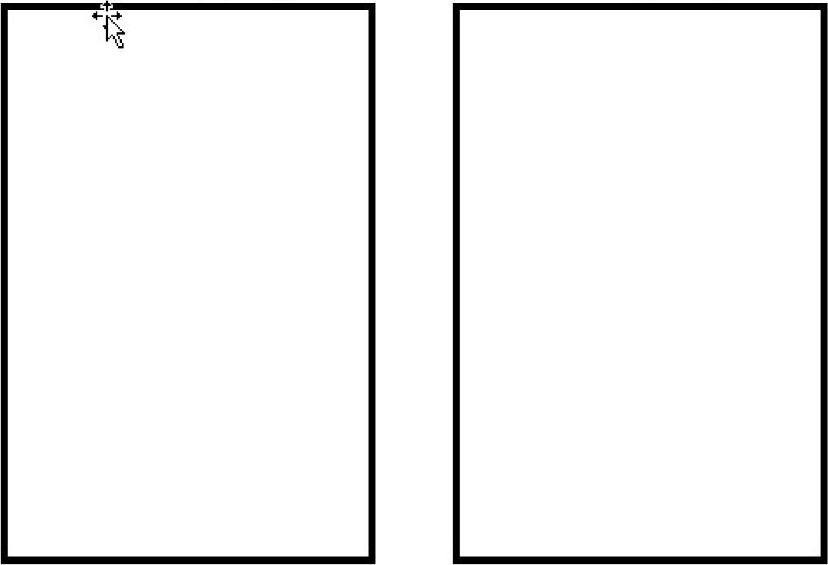
0.50

0.25

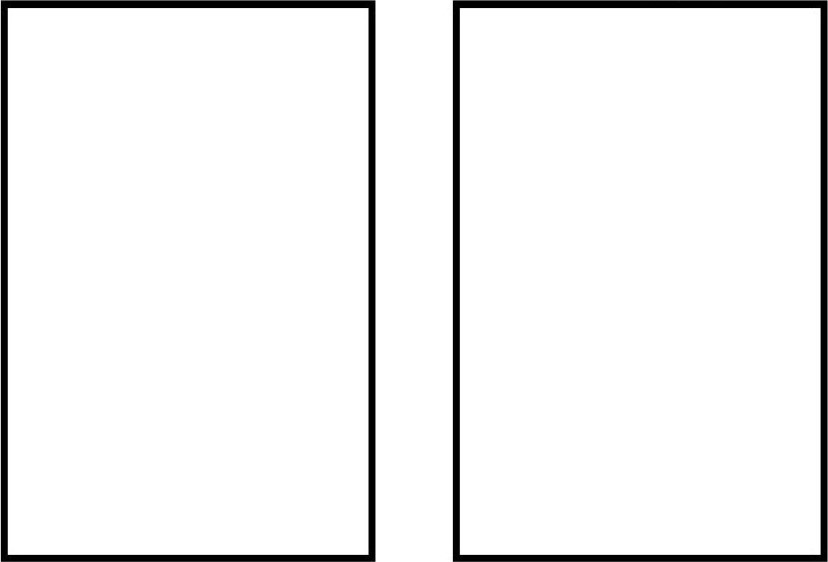
1.0

Explanation: The . subplots(nrows, ncols) function returns a two-item tuple object containing a Figure and one or more Axes objects (here 6 Axes ). The second item in the tuple of . subplots(nrows, ncols) is a numpy array containing all the Axes . You can check that by calling axes . We can use 0-based indexing to access each Axes and customize the plot and its features. The following shows the location of each Axes when using the . subplots() method:

Axes[O, 0] Axes[O, 1]



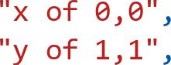
Axes[l, 01 Axes[l, 1]



Setting up the spacing between subplots

Subplots in Matplotlib are multiple plots within the same figure . As you can see, there is an overlapping issue in the subplots we have created. The spacing attribute determines how far apart the subplots are in the figure. We can handle that issue using the . tight\_layout() method. Subplots will be spaced out further away by adding plt.tight\_layout() . The following snippet will generate the same figure and subplots ; this time with better spacing between subplots 

Example



"y

import numpy as np import matplotlib. pyplot as plt

arr\_x = np.1inspace(2, 10) arr\_y\_l = np.cos(arr\_x) arr\_y\_2 = np.tan(arr\_x)

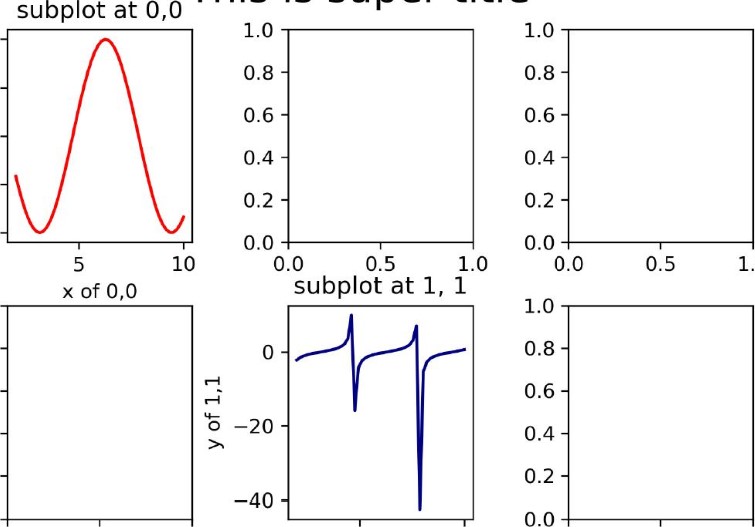
fig, axes = plt.subp10ts(nrows=2, nc01s=3)

axes [0, e] .  arr\_y\_l, " red" ) axes [1, 1] . plot(arr\_x, arr-y\_2, 'navy " ) plt. tight\_layout ( ) # control spacing

axes [0, title="subplot at e, 0") axes [1, 1]title:" subplot at 1, 1")

fig. is super title" , size=20, x=e.5, y-1.05)

0.5 1.0



1.0

0.5

0.0

-0.5

-1.0

1.0

0.8

0.6

0.4

0.2

0.0

This

is

super

title

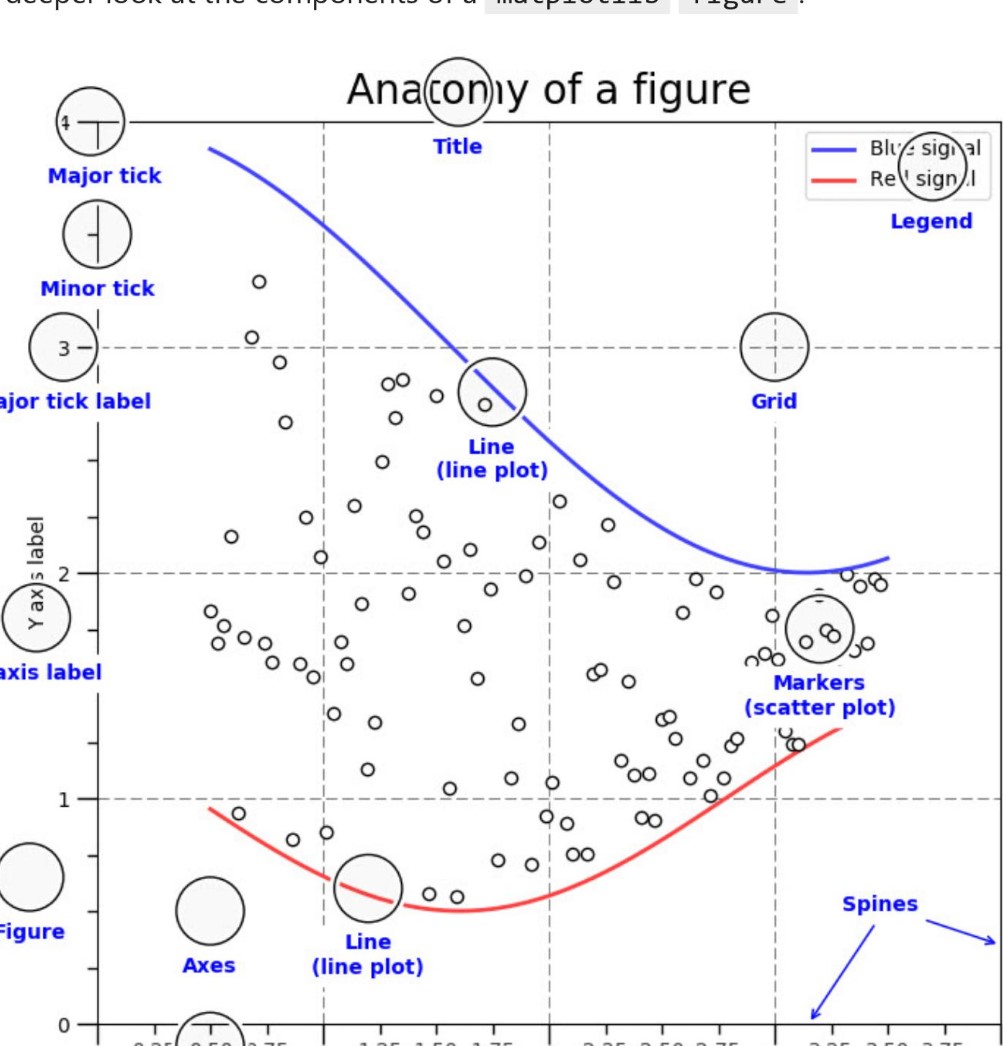
0.0 0.5 1.0 5 10 0.0 0.5 1.0

Note: PI t. tight\_layout() has an optional argument of pad with a default value of 1.08 . By increasing that number, the spacing between subplots will increase.

Side note: plt.tight\_layout() has optional arguments of h\_pad and w\_pad as well. I will leave exploring these options to you!

Fine-tuning matplotlib figure

One of the best features of matplotlib is its capability to customize every possible component of a plot. The following image is taken from the



Major

Y

axis

0.25

0.50

3.75

1.25

1.50

2.25

2.50

2.75

3.25

3.50

3.75

X

ax

s

label

Minor

tick

label

Made

with

http://matplotlib.org

matplotlib

website

and

shows

a

deeper

look

at

the

components

of

a

matplotlib

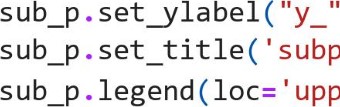
figure

X axis label

In this section, we will learn about most important customization we might need fora figure . While this is an attractive feature, with this level of flexibility comes complexity, frustration, and confusion. For further figure customization, please consult with matplotlib 3489-page-long documentation at https://matplotlib.org/stable/Matplotlib.pdf.

Let's look at one self-explanatory example:

|  |  |  |  |
| --- | --- | --- | --- |
| axes [0, | 0] . plot(arr\_x, np.sin(arr x), "red" , linewidth = 0.3, marker - o' | | markersize |
| label - | r') |  |  |
| axes [0, | np.cos(arr x), "black", label - | linestyle |  |
| axes [1, | 0] . plot(arr\_x, np.tan(arr\_x) "navy", label = 'n' ) |  |  |
| axes [1, | " purple" , linewidth | = 2.5 label - | 'p' ) |

Example

import numpy as np import matplotlib. pyplot as plt

arr\_x = np.1inspace(2, 10, 100)

# set figure settings at initiation fig, axes - plt.subp10ts(nrows=2, nc01s=2, figsize 3), dpi = 200) plt. tight\_layout (pad

#pLotting

# common notation for changing settings of all subplots all at once for i, sub\_p in enumerate(fig.axes):

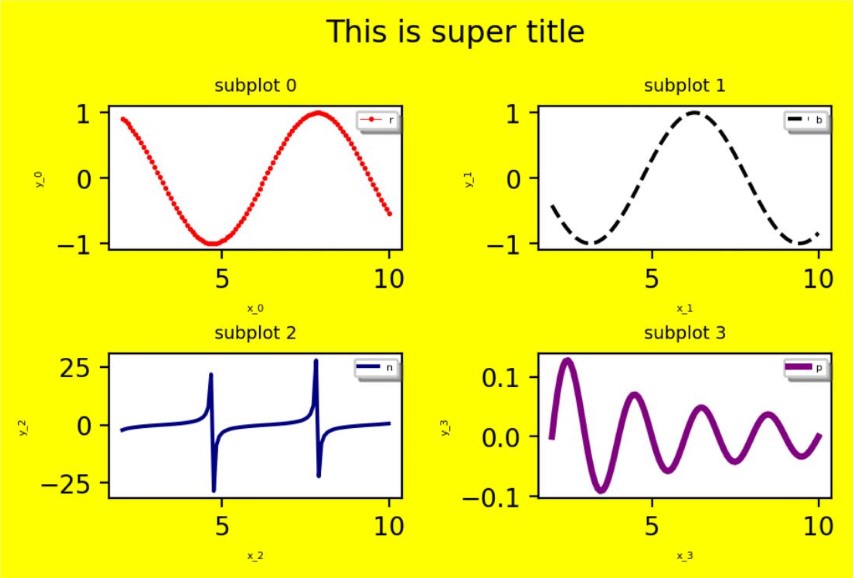
sub\_p.set\_xlabel("x\_" + str(i), fontsize = 4)

#### + str(i), fontsize = 4)

'subplot '+ str(i), fontsize = 7)

' upper right' , shadow = True, fontsize = 4) fig. suptit1e("This is super title" , size=12, x=0.5, y=1.05)

fig. savefig( "save\_me. png" , bbox  facecolor = 'yellow' )



Note 1: linestyle and marker arguments are used to control plot appearance. A whole list of what is possible can be found at https://matplotlib.org/stable/api/markers\_api.html and https://matplotlib.org/2.0.1/api/lines\_api.html.

Note 2: You can generate high quality images from matplotlib figure . You can use .savefig() method to save a figure in a variety of formats such as jpg , jpeg , png , svg , pdf , etc.

Optional: One can confirm existence of the above generated image file by running the following snippet

Optional

import matplotlib. image as mpimg plt. imshow(mpimg.imread( ' save\_me. png' ) )

This will read the image file save\_me .png and will display it.

Commonly used plots in matplotlib

A quick look at the matplotlib gallery available at https://matplotlib.org/stable/gallery/index.html# reveals a wide variety of plots that can be generated using matplotlib . However, a small set of relatively limited plot types is used in most practical situations. The rest of this handout will provide quick examples of each of these plot types

Histogram

* Application: Displays the shape of the data. A histogram is the standard way to show a (statistical) distribution of a numerical value.
* Common syntax: Use .hist() method with most common syntax of ax. hist(x, bins = , range - cumulative - 

|  |  |  |
| --- | --- | --- |
| Parameter | Optional? | Description |
| x | No | array or sequence of arrays |
| bins | Yes | Most common format: integer number.  It defines the number of equal-width bins in the range |
| range | Yes | A tuple. The lower and upper range of the bins.  Lower and upper outliers are ignored.  If not provided, range is (x.min(), x.max()) |
| cumulative | Yes | Default: False  If set True, it returns cumulative histogram of the data |

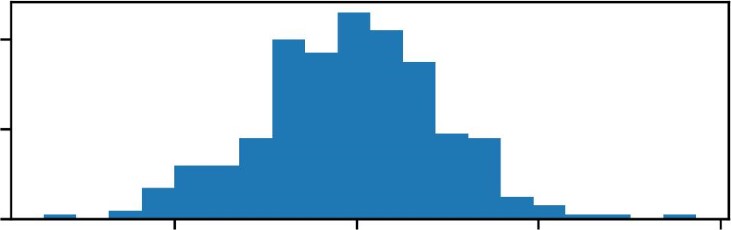
Histogram Example

import numpy as np import matplotlib. pyplot as plt

# Generate data data 1 = np. random. normal (40, 5, size = 300) data 2 = np. random. random(size -\_ 4000)

fig, axs = plt.subplots(nrows = 2, ncols = 1, figsize = (4, 3), dP1 • - - 150) plt. tight\_layout ( )

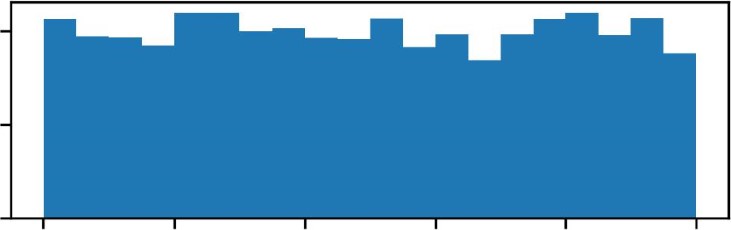
axs [0] . hist(data\_l, bins=20) axs[l] .hist(data\_2, bins=20)

40

20

# 0

30 40 50 60

200

100

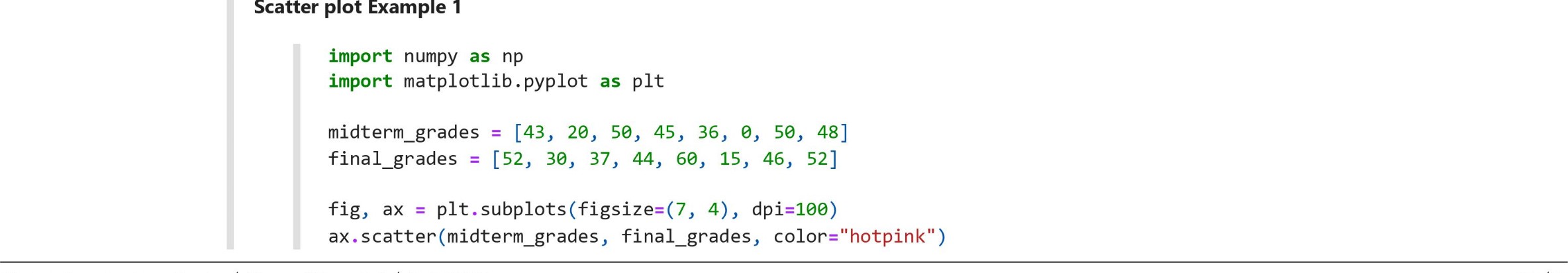
0

0.0 0.2 0.4 0.6 0.8 1.0

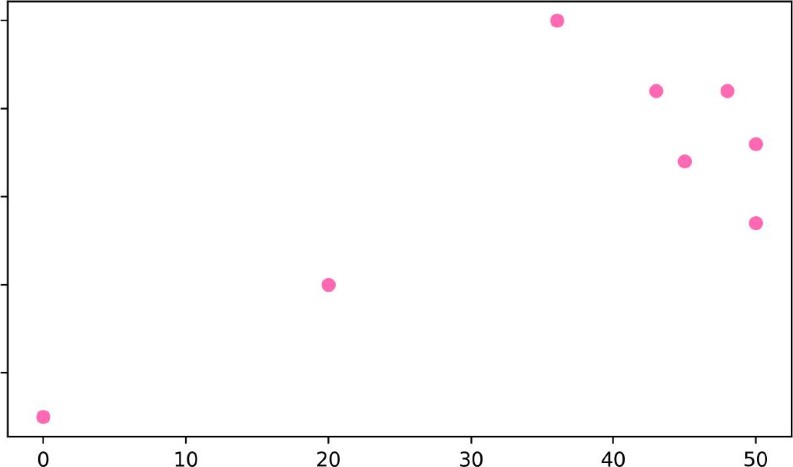
Scatter plots

* Application: Visualizes how two numerical values are related or affected by each other; this plot helps us understand the relationships between variables.
* Common syntax: Use . scatter() method with most commonly used syntax of ax. scatter(x, y, color -

|  |  |  |
| --- | --- | --- |
| Parameter | Optional? | Description |
| x | No | sequence of scalars. The data x coordinates |
|  | No | sequence of scalars. The data y coordinates |
| color | Yes | Optional array or list of colors or color |



60



10

20

30

50

50

40

30

20

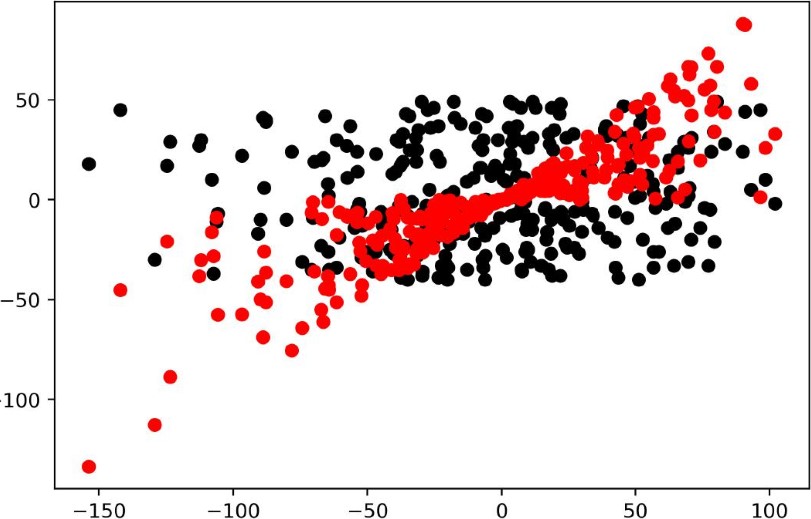
Scatter plot Example 2

import numpy as np import matplotlib. pyplot as plt

# Generate data data 1 = np. random. normal (1, 50, size = 300) temp\_l - np. random. random(size = 300) temp\_2 - np.  size = 300) data 2 = data\_l \* temp\_l

fig, ax = plt.subplots() plt. tight\_layout ( )

ax. scatter(data 1, temp\_2, color - 'black' ) ax. scatter(data 1, data\_2, color - ' red' )



-100

-150

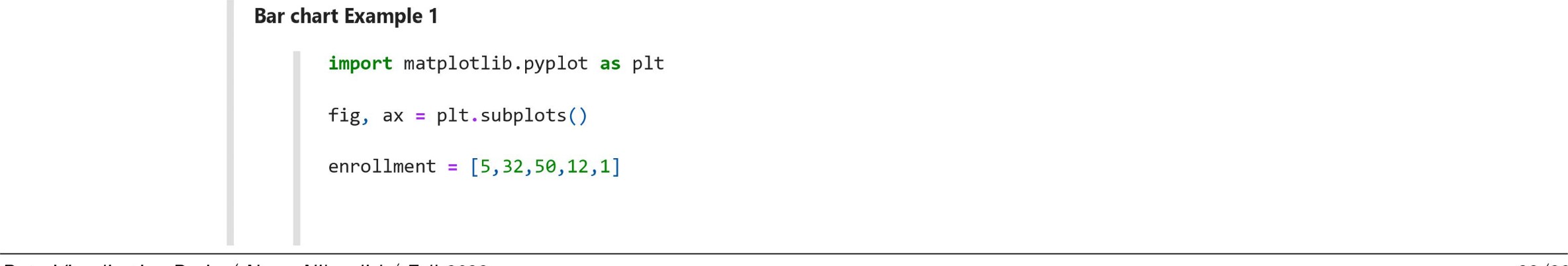
-100

-50

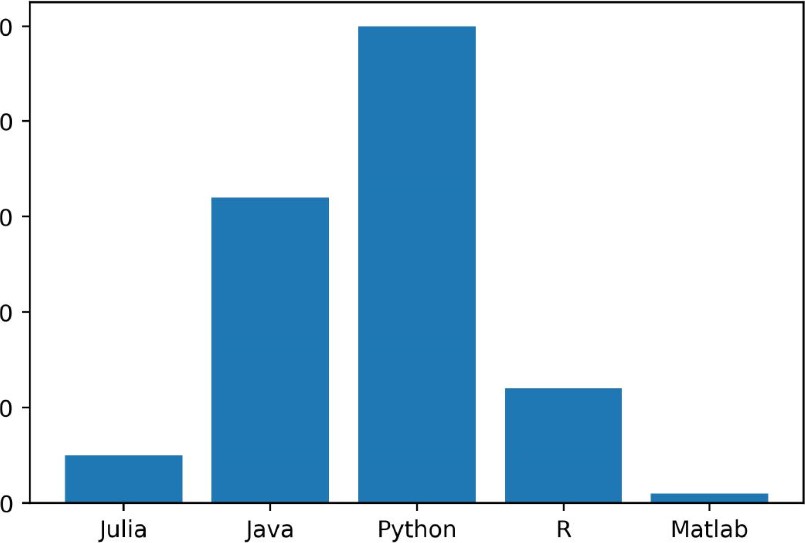
Bar chart

* Application: Presents categorical data with vertical or horizontal bars with heights or lengths proportional to the values they represent.
* Common syntax: Use . bar() method with most common syntax of ax. bar (x, height)

|  |  |  |
| --- | --- | --- |
| Parameter | Optional? | Description |
| x | No | sequence of scalars representing the x coordinates of the bars. |
| bins | No | scalar or sequence of scalars representing the height(s) of the bars. |
| width | Yes | The width of the bars. Default = 0.8 scalar or array-like, optional. |
| bottom | Yes | The y coordinate(s) of the bars bases. default is 0 Used mostly for building stacked bar chart |



languages = [ 'Julia' , 'Java' , 'Python' , 'R', ' Mat lab ' ] ax. bar( languages , enrollment)

50

40

30

20

#### 10

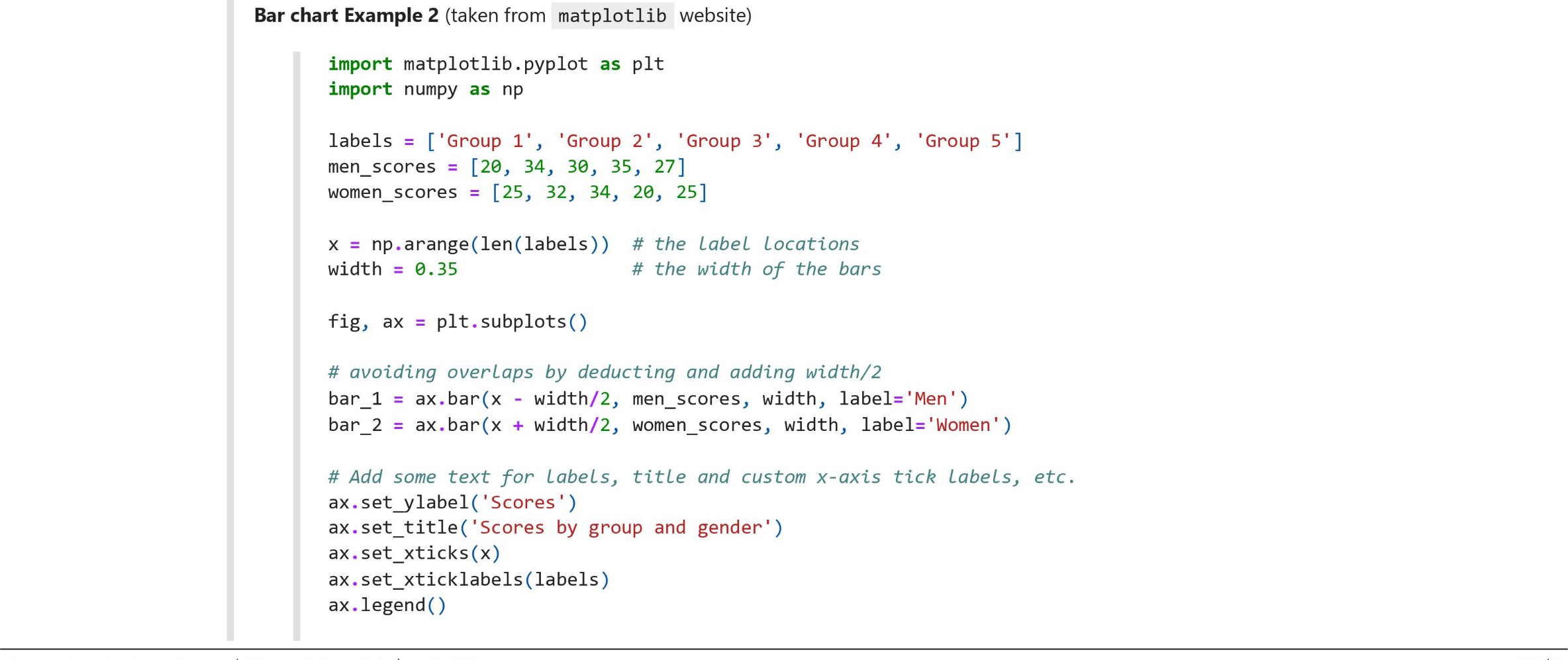
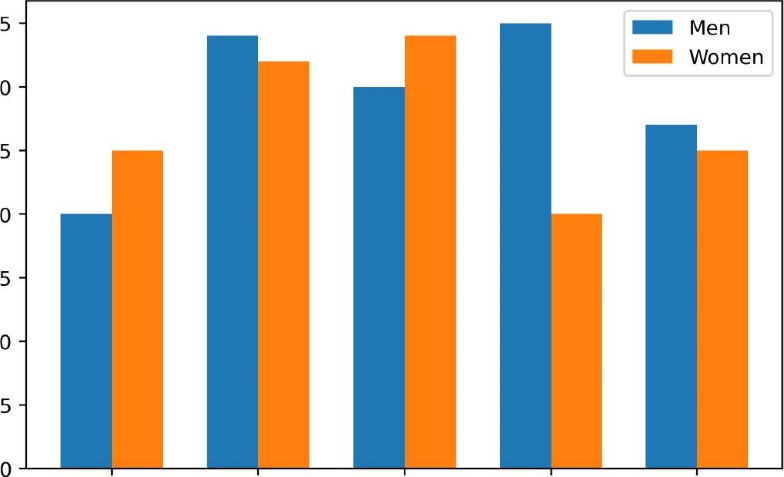


 fig. tight\_layout ( )

Scores by group and gender

35 30

25

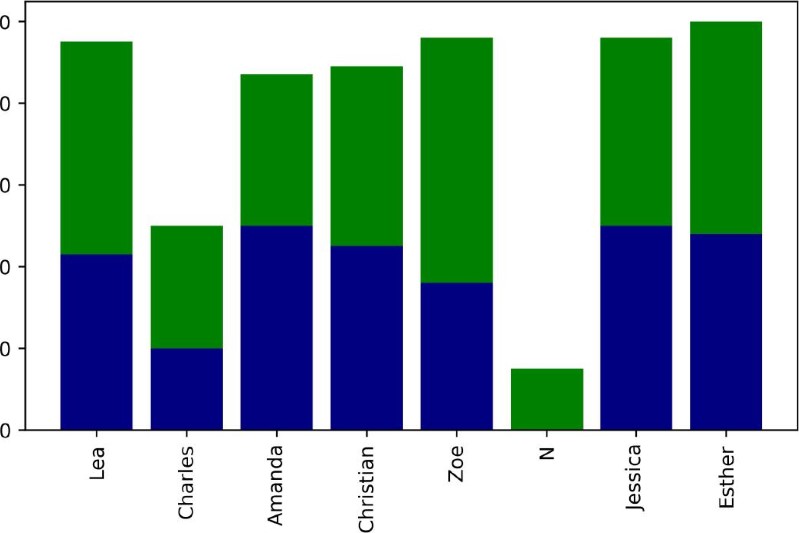
8 20

15

#### 10

Group I Group 2 Group 3 Group 4 Group 5



100

80

60

40

20

Note: Use . barh() for horizontal bar chart.

seaborn

Matplotlib is python's most important visualization library because many other visualization libraries in python depend on matplotlib Though matplotlib offers high customization, it involves a lot of coding, so working with matplotlib could be very time-consuming.

seaborn library provides simple, quick (low code), and attractive visualizations, especially for data analysis. One of the most critical aspects of seaborn is that it focuses just on the key elements of the charts. seaborn integrates well with pandas 

seaborn is built over matplotlib and simplifies many operations. We will use aspects of matplotlib in visualizing with seaborn because some of the seaborn functions return matplotlib objects.

seaborn documentation suggests importing it as sns  import seaborn as sns

Because of inner dependency of seaborn to matplotlib.pyplot it is highly recommended that whenever you use seaborn , you should have import matplotlib.pyplot as plt

as well.

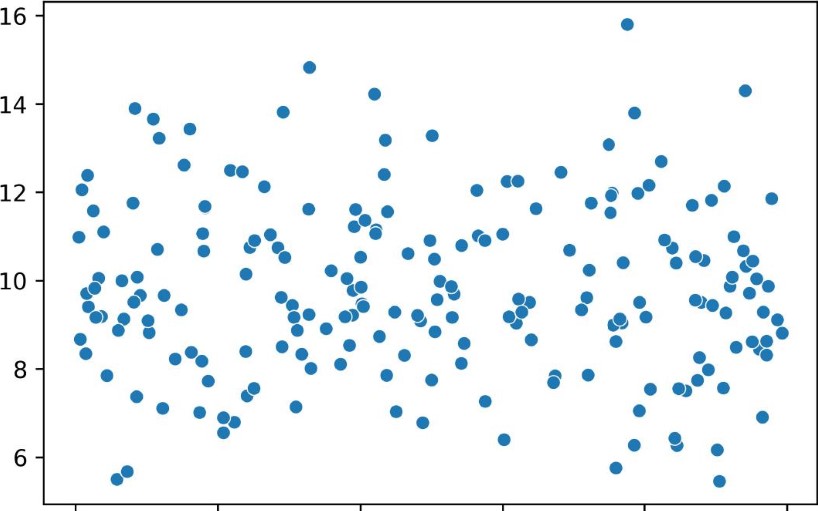
Let's start with a simple scatter plot.

Example

import numpy as np import matplotlib. pyplot as plt import seaborn as sns

arr 1 arr 2 = np.random.norma1(10c=10, scale=2, size-200)

axes -- sns.scatterplot(x = arr\_l, y = arr\_2)



0.0 0.2 0.4 0.6 0.8 1.0

Note about passing data to seaborn : One of the many differences between seaborn and matplotlib is that in using seaborn it is highly recommended to explicitly determine values of x and y 

Passing pandas objects to seaborn : Often, we pass columns of a pandas DataFrame as values of x and y . This could be done in at least two different ways:

* (x = df["col\_l"], y = df["c01\_2"] ) where col\_l and col\_2 are valid columns in DataFrame df.
* The syntax of (data = df x ' col\_l ' , Y = col\_2' ) can work equivalently well. In the remainder of this handout, we will use the clean\_search\_history.csv data set.

### sns . barplot

A bar plot is used for visualizing categorical columns vs. numerical columns. Lets create a barplot of data\_science popularity for each category value (here e and 1 ) of categorical column.

Example

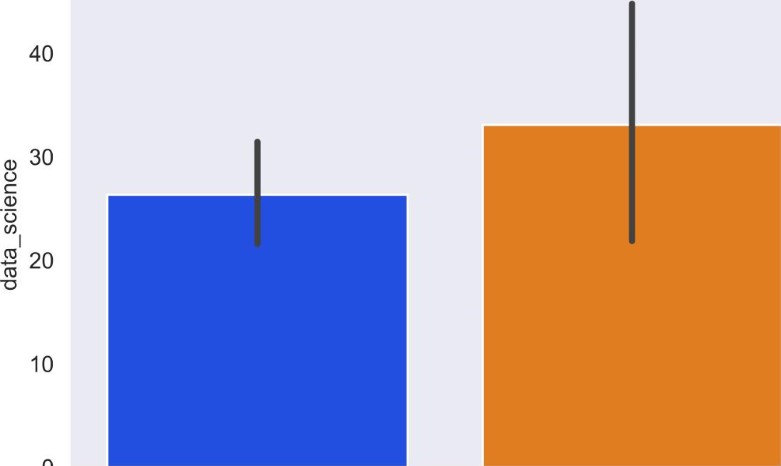
import numpy as np import pandas as pd import matplotlib. pyplot as plt import seaborn as sns

df = pd.

sns . " dark" )

axes = sns.barplot(data=df, x="categorical", Y="data science", palette: "bright" )

axes . set (xlabel= " Country" , xtick1abe1s=["USA", "Canada"] )



USA Canada

Country

This will set the height of every bar equal to the average values in the data\_science column for each value of the categorical variable (0 and 1 here). barplot has an optional argument estimator that can control the height of each bar. Try estimator = sum , estimator = len , or estimator = median 

### sns . boxplot

A boxplot is a visual representation of five-point summary statistics of a given data set. A five-point summary includes minimum , first quartile (25th percentile) , median (second quartile or 50th percentile) , third quartile (75th percentile) , and maximum . A boxplot shows distributions for categories. According to seaborn documentation:

A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be "outliers" using a method that is a function of the interquartile (IQ) range.

Technical note: A dataset's Inter-quartile ( IQ ) is defined as IQ = Q3-Q1 where Q3 is the 3rd quartile, and QI is the 1st quartile. According to the Inter-Quartile method, any observation outside [QI-3\*IQ, Q3+3\*IQ] is considered an outlier 

Here is an example:

Example

import numpy as np import pandas as pd import matplotlib. pyplot as plt import seaborn as sns

df = pd.

sns . set\_style( "dark" )

axes -- sns.boxplot(data=df, x="categorical", y="data science", palette="bright")

axes . set (xlabel= " Country" , xtick1abe1s=["USA", "Canada"] )

8

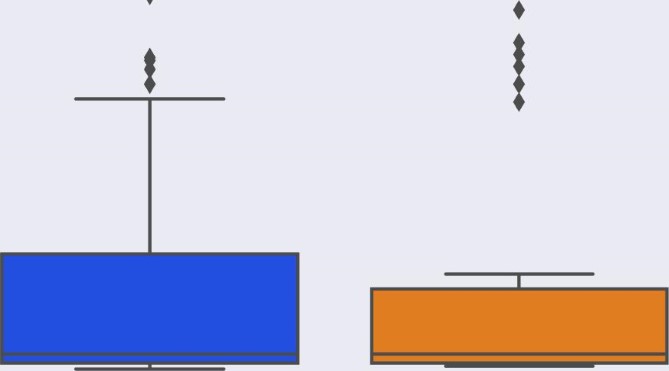




• boxplot has an optional argument of

sns . violinplot

140

120

100

80

60

40

### 20

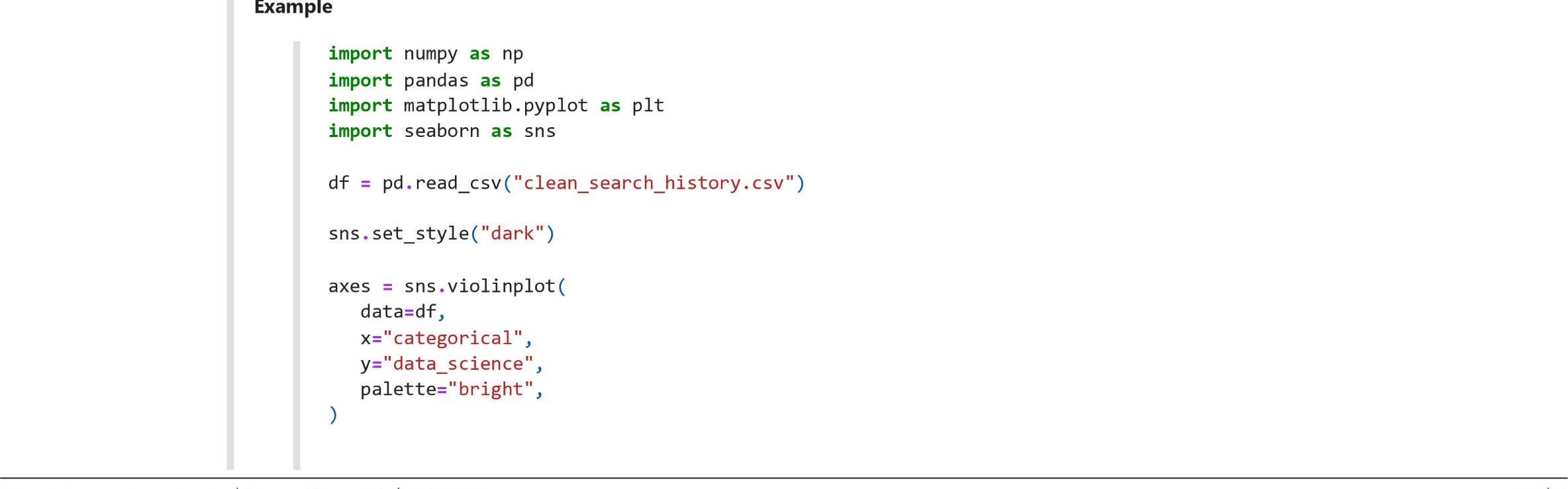
USA Canada

Country

orient . Try orient = 'h'

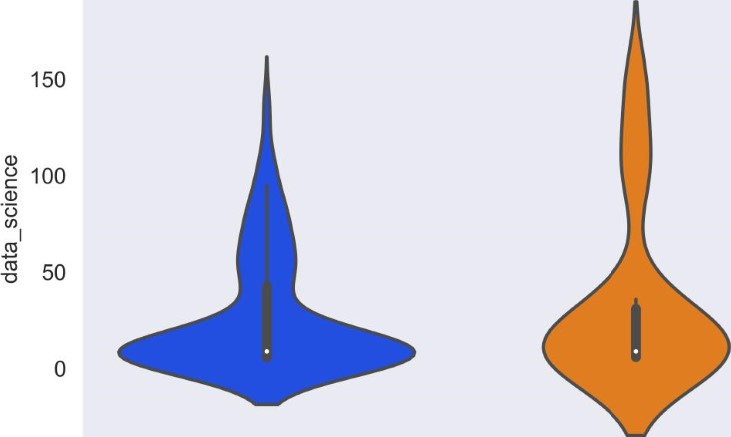
A violinplot combines a boxplot and a distplot . A boxplot or a violinplot are created for categorical-continuous variables

(if the x-axis is categorical and the y-axis is continuous ). Here is an example



axes . set xtick1abe1s=["USA", "Canada"] )

### 200



USA Canada

Country

sns . histplot

Use sns.histplot to visualize the distribution of one or more variables. Here are a few examples

Example

import numpy as np import pandas as pd import matplotlib. pyplot as plt import seaborn as sns

df = pd.

sns . set\_style( "dark" )

axes -- sns.histplot(data=df, , color:" navy", bins=15, palette:" bright") axes . set (ylabel= " Freq " , title="Histogram")

Histogram

120

100

80

u- 60

40

### 20

0

* 20 40 60 80 100 120 140

data science

* One acceptable rule of thumb for the number of bins is number of rows  Instead of bins , one can pass the binwidth value.
* A histogram with horizontal bars can be achieved simply by determining the value of y instead of x .
* a kernel density estimate can be added by passing kde = True to the sns.histplot() method. This provides (smoothed) information about the shape of the distribution. More about kernel density estimate at https://en.wikipedia.org/wiki/Kernel\_density\_estimation  hue parameter can be used for mapping values to a categorical variable. Try hue = 'categorical
* If a stacked histogram is desired, use multiple - ' stack'
* If neither x nor y is assigned, a histogram is drawn for each numeric column in data 
* When both x and y are assigned, a bi-variate histogram is computed and shown as a heatmap 

sns . distplot

A closely related plot is sns . distplot to visualize the distribution of one or more variables.

sns . jointplot

This method draws a plot of two variables with bi-variate (scatter plot) and uni-variate (histogram) graphs. A Joint-Plot takes two variables and creates Histogram and Scatter plot together.

A scatter plot is used for the visual inspection of correlation, and the histogram is used for inspecting the approximate shape of data distribution.

Example

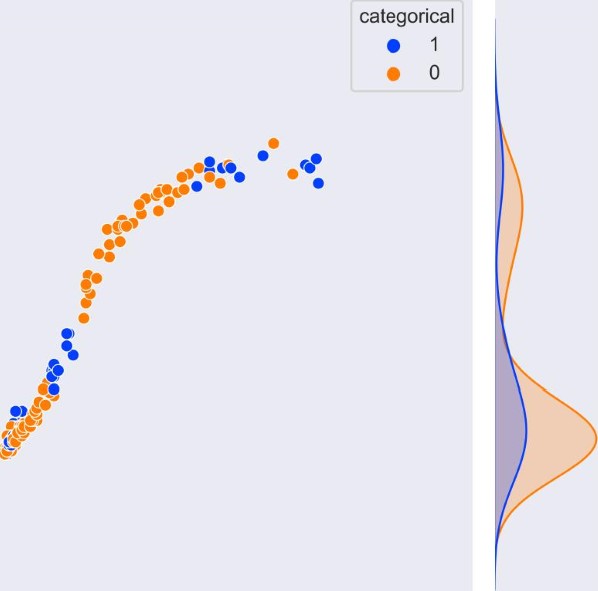
import numpy as np import pandas as pd import matplotlib. pyplot as plt import seaborn as sns

df = pd. read\_csv("clean\_search\_history.csv")

sns . set\_style( "dark" )

axes -- sns.jointplot( data=df, x="data science", y: machine\_learning" , hue=" categorical " , # categorical variable for mapping color of plot elements hue\_order=[l, 0], # order of plotting for categorical Levels of the hue color="navy' , palette:" bright" ,





150

125

100

.E

75

c

50

25

-25

-50  50 100 150 200 data science

Note: jointplot() has an optional argument of kind with a default value of scatter . Try: kind = reg , kind = resid , kind = kde and kind = hist . How do you think you can use kind—resid ?

### sns . pairplot

A pairplot visualizes pairwise relationships among numerical values in a data set.

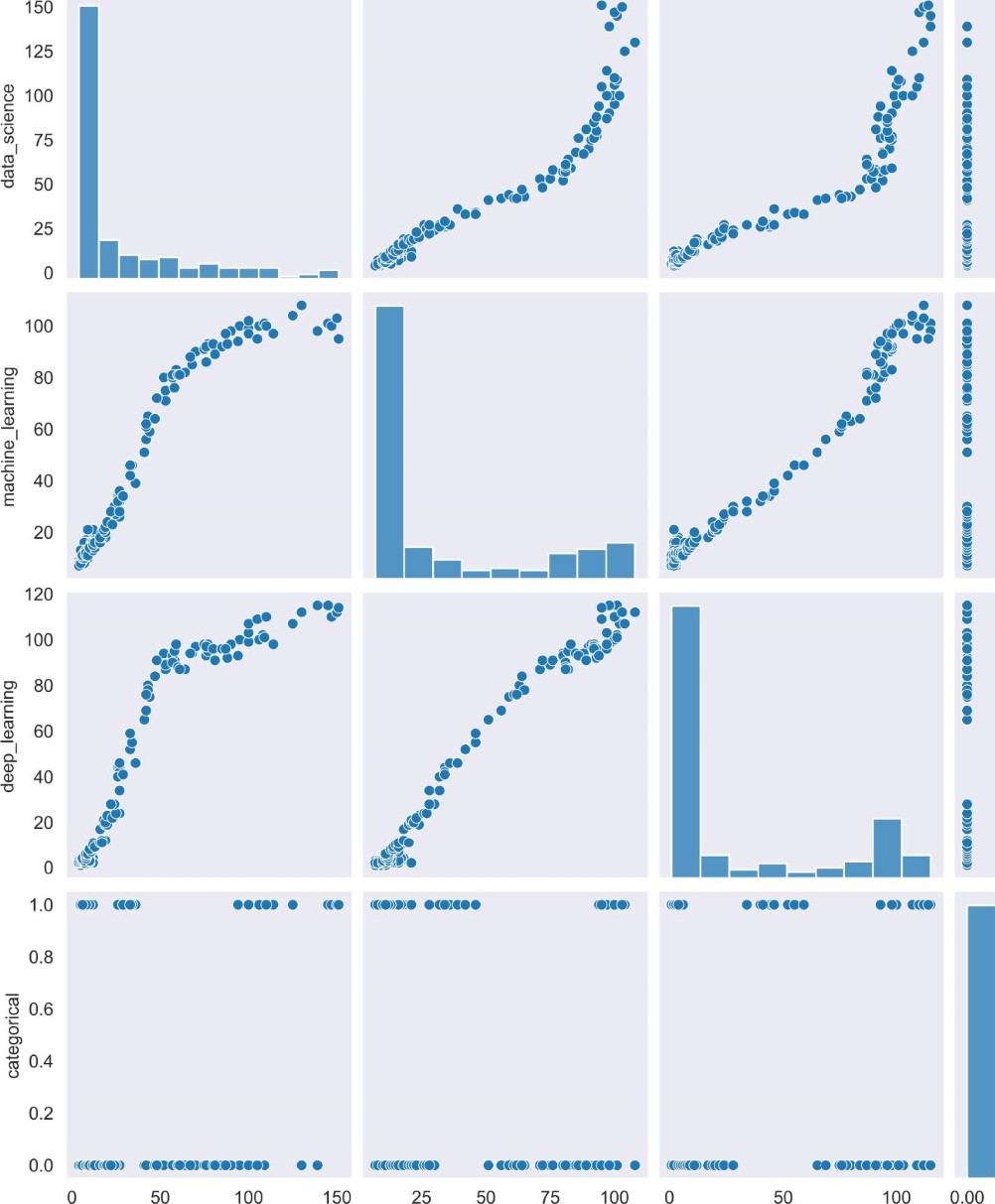
Ill Example

import numpy as np import pandas as pd import matplotlib. pyplot as plt import seaborn as sns

df = pd. read\_csv("clean\_search\_history.csv")

sns . set\_style( "dark" )

axes -- sns.pairplot(data = df)

0.00 0.25 0.50 0.75 1.00 data science machine\_leaming deep\_learning categorical

* The diagonal part shows the distplot or histogram with kernel density estimation. The upper and lower part of the Pairplot shows the scatterplot 
* The hue parameter can be used to color the plot using categorical columns.
* The kind parameter is similar to jointplot 

Try axes = sns.pairplot(data=df, kind="reg", hue="categorical")

sns . Implot

You are familiar with Implot from the linear regression part of the course. Implot is a plot that fits the ordinary least square regression line to

|  |  |
| --- | --- |
| the data set showing as | scatterplot (s). |

Example

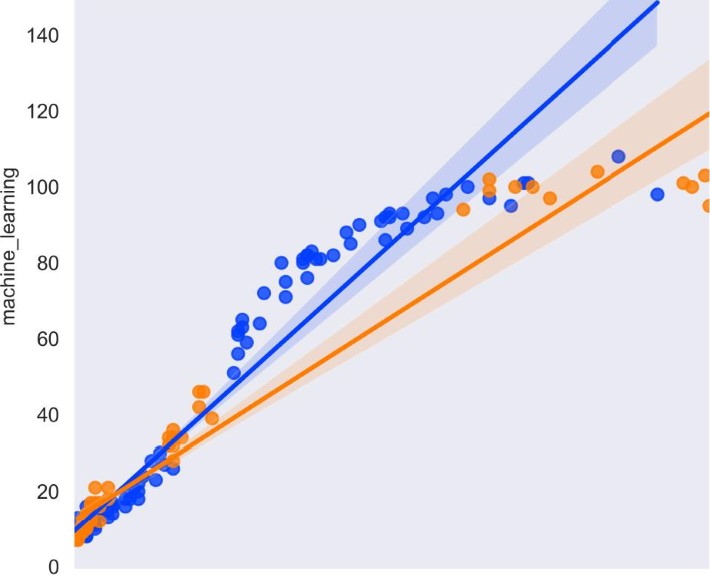
import numpy as np import pandas as pd import matplotlib. pyplot as plt import seaborn as sns

df = pd. read\_csv("clean\_search\_history.csv")

sns . set\_style( "dark" )

axes = sns.lmplot(data = df, x — 'data science'machine\_learning hue — ' categorical ' , palette — 'bright' )

160

 categorical

20 40 60 80 100 120 140

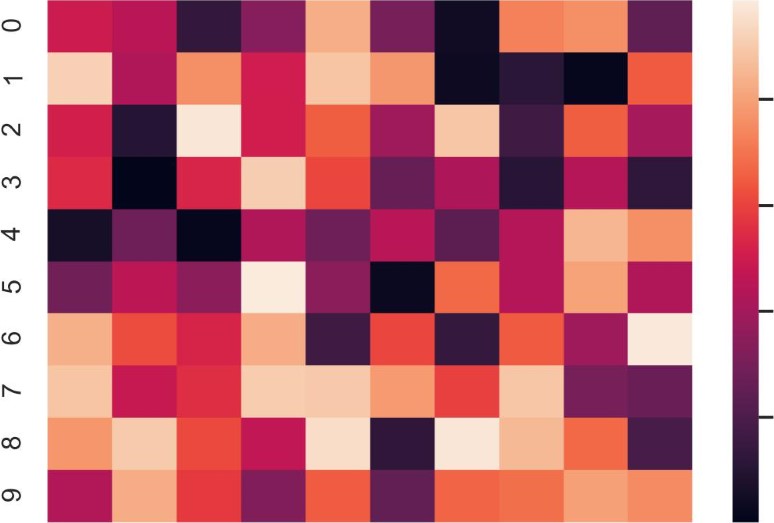
data science

sns . heatmap

heatmap One of the most popular plots provided by seaborn . Let's look at several cool heatmaps taken from the seaborn website.

Uniform heatmap example

import numpy as np import seaborn as sns sns . set\_theme( ) uniform data - np.random. random(size = (10, 10)) ax = sns . heatmap (uniform\_data )

0.8

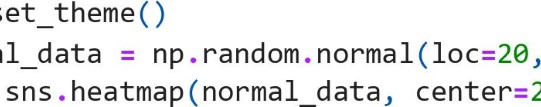
0.6

0.4

0.2

o 1 2 3 4 5 6 7 8 9

Normal heatmap example



normal

ax

=

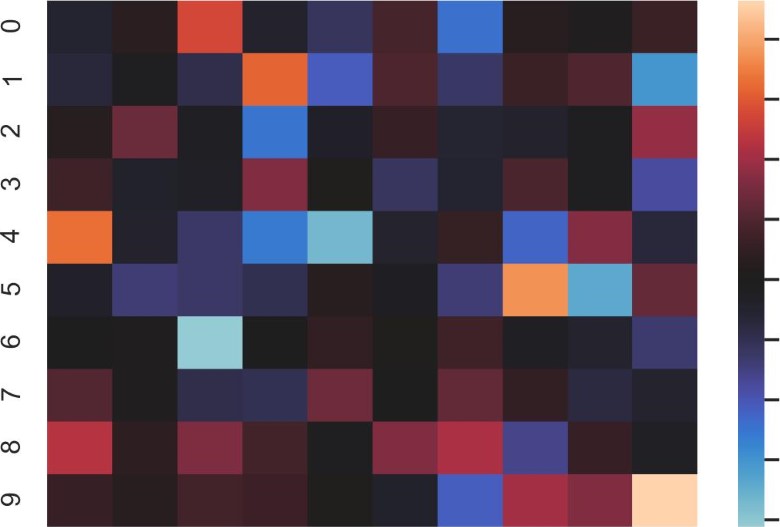
center=20)

import numpy as np import seaborn as sns

sns .

#### scale=3, size-(10, 10) )

Ill

28

26

24

22

20

18

16

14

##### 12 o 1 2 3 4 5 6 7 8 9

Besides making cool-looking plots(!), in data analytics, it is widespread to use heatmap to show all the correlations between variables in a dataset. Here is an example

heatmap for visualizing correlation example

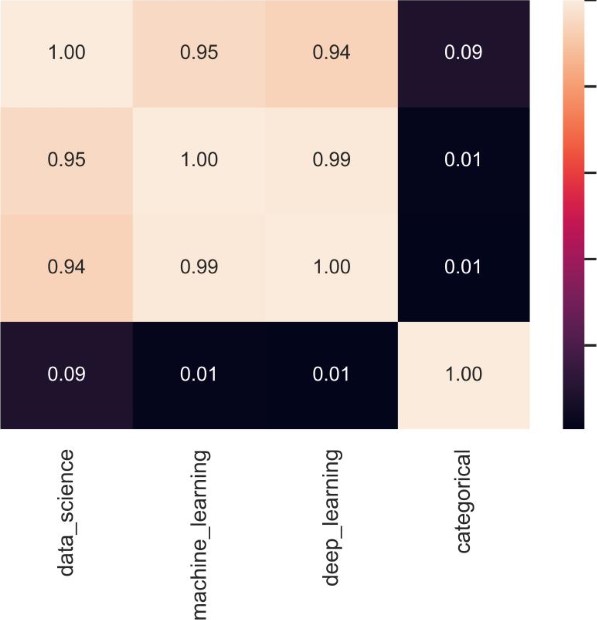
import numpy as np import pandas as pd import matplotlib. pyplot as plt import seaborn as sns

df = pd.

sns . set\_style( "dark" )

axes = sns.heatmap(df.corr(), annot=True,

- 1.0

data science

0.8

machine\_learning0.6

0.4 deep\_learning

0.2 categorical

### saving a sns plot

Similar to matplotlib , savefig() method could be used to save a seaborn plot. Here is the most common syntax used for savefig

PIt. savefig( "file\_name.extension", dpi facecolor = edgecolor = , bbox\_inches= 'tight ' , pad\_inches = 0.5)

References

This document is an adaption from the following references:

1. Python online documentation; available at https://docs.python.org/3/
2. Exploratory Data Analysis with Python; Suresh Kumar Mukhiya and Usman Ahmed, Packt publications, 2020
3. Introduction to programming in Python; available at https://introcs.cs.princeton.edu/Python/home
4. Introduction to Computation and Programming Using Python: With Application to Understanding Data, John Guttag, The MIT Press, 2nd edition, 2016
5. Introduction to Python for Computer Science and Data Science, Paul J. Deitel, Harvey Deitel, Pearson, 1st edition, 2019
6. Data Mining for Business Analytics, Galit Shmueli, Peter C. Bruce, Peter Gedeck, Nitin R. Patel, Wiley, 2020
7. Interactive Data Visualization with Python; Abha Belorkar, Sharath Chandra Guntuka, Shubhangi Hora, and Anshu Kumar. Packt publications, 2020
8. Simulation Modeling with Python; Giuseppe Ciaburro, Packt publication, 2020
9. https://betterprogramming.pub/ 10- https://www.learnpython.org/ 11- https://seaborn.pydata.org/

12- https://pyviz.org/index.html 13- https://numpy.org/

1. https://pandas.pydata.org/
2. Pandas Cookbook, Matt Harrison and Theodore Petrou, Packt publications, 2020