



Fall, 2020

Python Programming Workshop

Workshop 5

Jupyter Notebook, Scientific Computing and Plotting

Kevin Kam Fung Yuen

Department of Computing

Hong Kong Polytechnic University

kevin.yuen@polyu.edu.hk



Class Schedule

19 Sep: Python Programming Environment and Basics

26 Sep: Control Flow and Lists

3 Oct: Lists, Tuples and Dictionaries

10 Oct: Functions, Modules, OO and Packages

17 Oct: Jupyter Notebook, Scientific Computing and Plotting

Lecture notes are available in

<https://github.com/kkfyuen/PythonWorkshop2020>



Outline

- Jupyter notebook
- Markdown language
- Install packages
- Numpy package
- Matplotlib package



Jupyter notebook

Make sure that your internet is connected.

pip: Package Installer for Python

```
Windows PowerShell
PS C:\> pip install jupyter
```

for more information,
> pip help
> pip list

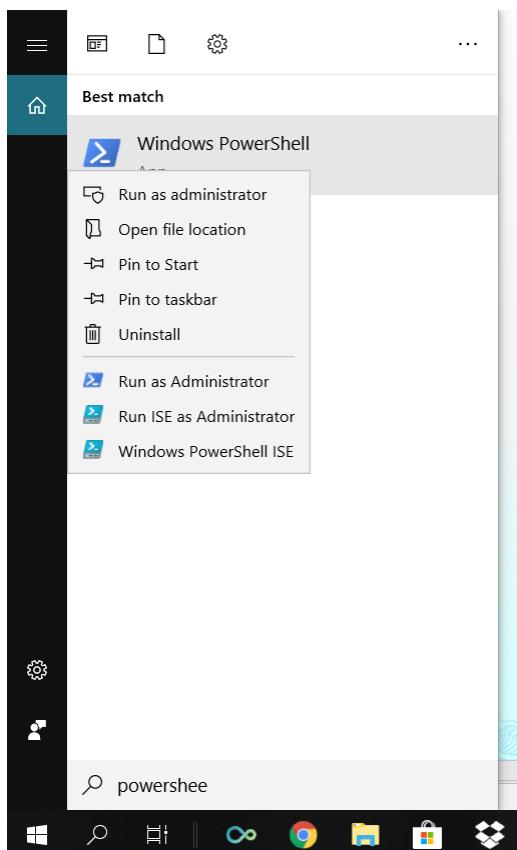
```
Could not install packages due to an EnvironmentError: [WinError 5] Access is denied: 'c:\\program
files\\python37\\Lib\\site-packages\\wcwidth'
Consider using the '--user' option or check the permissions.
```

So?

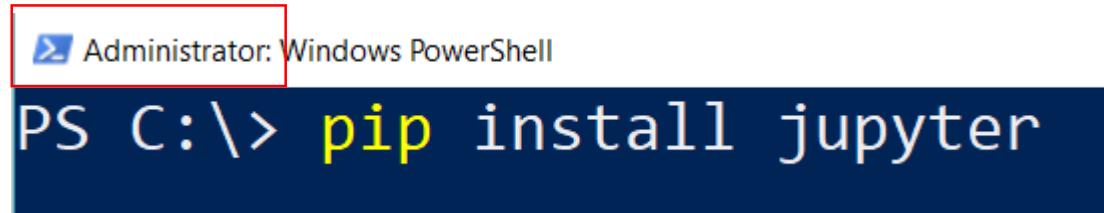
First time to Jupyter notebook

Step 1: right click

Windows PowerShell and
run as administrator



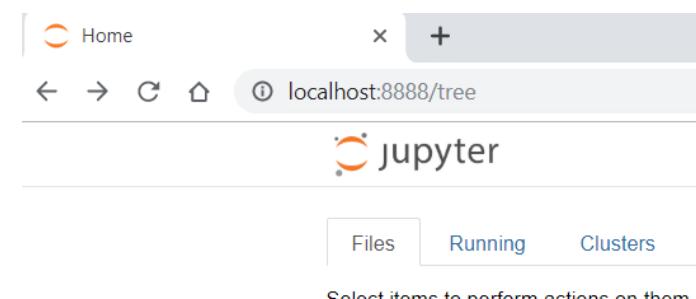
Step 2: pip install jupyter



```
Administrator: Windows PowerShell
PS C:\> pip install jupyter
```

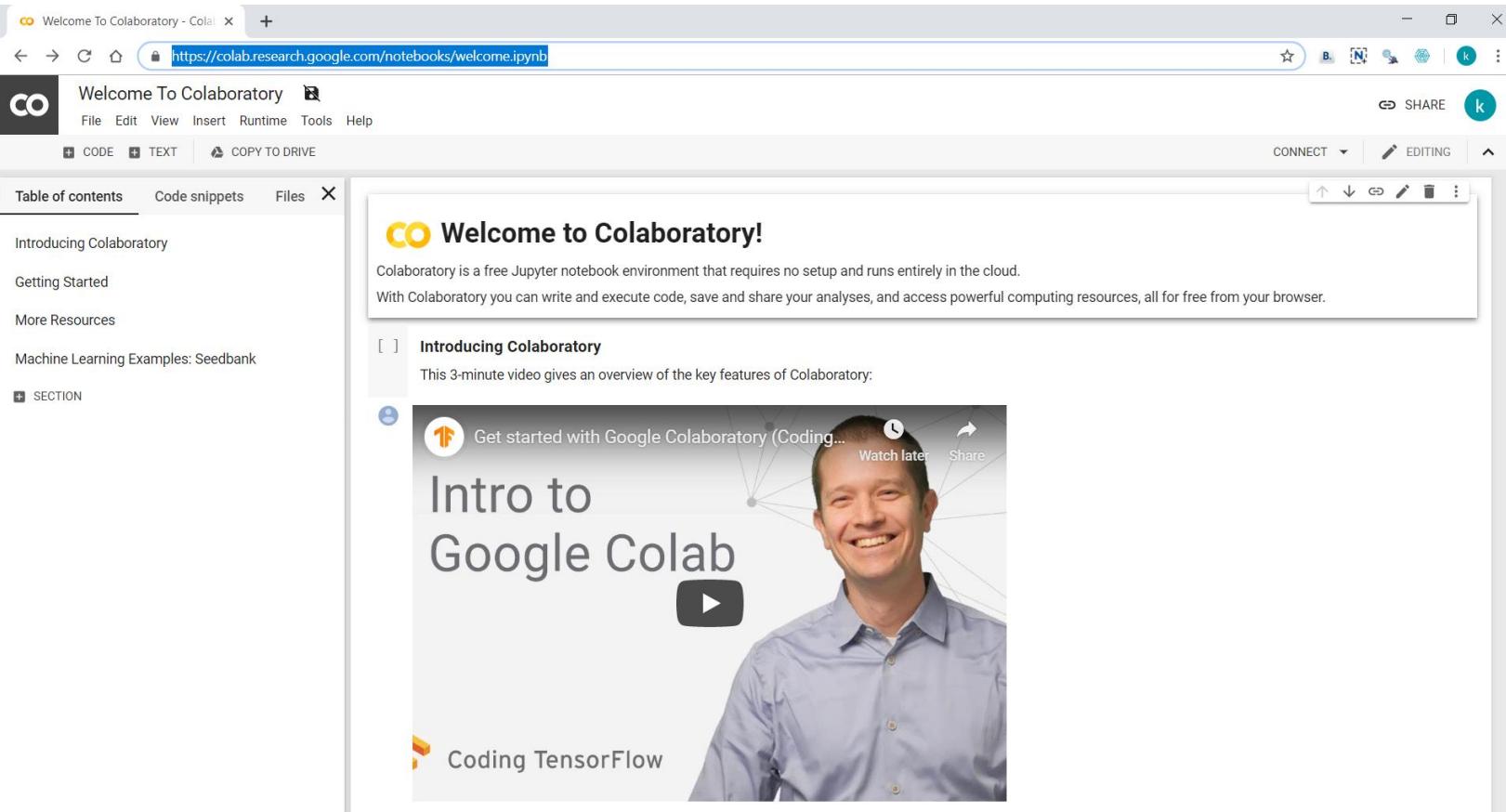
Make sure that your internet is connected.

Step 3: type **jupyter notebook** to start the web application.





Alternative: Google colab

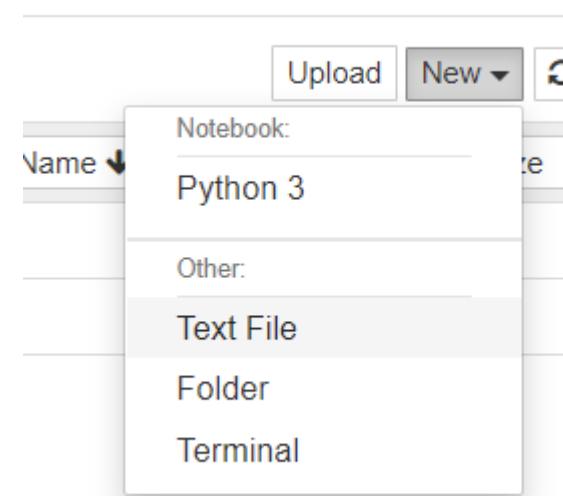


The screenshot shows the Google Colaboratory interface. At the top, there's a navigation bar with tabs for 'CODE' and 'TEXT'. Below it is a sidebar with a 'Table of contents' section containing links like 'Introducing Colaboratory', 'Getting Started', 'More Resources', and 'Machine Learning Examples: Seedbank'. The main content area features a large video player with the title 'Welcome to Colaboratory!' and a subtitle 'Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.' Below the video, there's another section titled 'Introducing Colaboratory' with a subtitle 'This 3-minute video gives an overview of the key features of Colaboratory:' followed by a video thumbnail for 'Get started with Google Colaboratory (Coding...)'.

<https://colab.research.google.com>

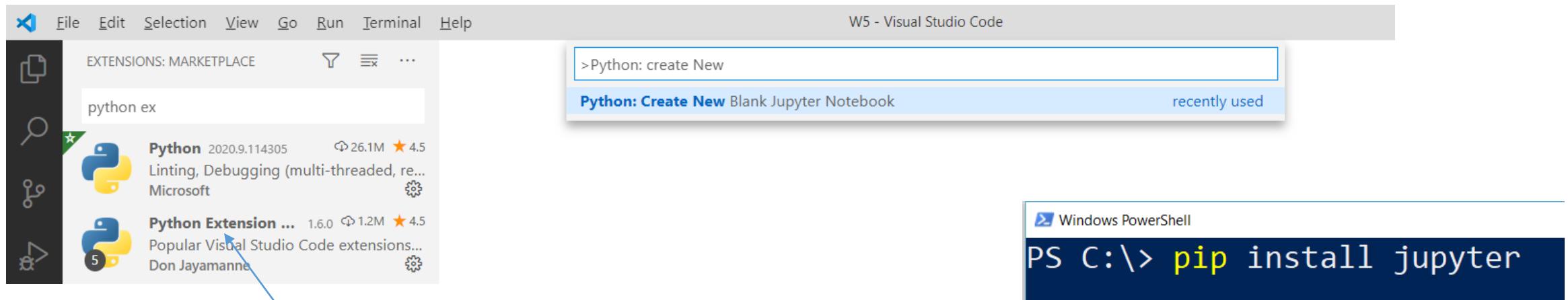


- Create a **Folder**, named PythonWorshop2020/W5
- Under PythonWorshop2020/W5, create a *Python 3* file with your preferred name.



Jupyter Notebooks in Visual Studio Code

View → Command Pallete (Crt +shift + p)



1. Make sure that Jupyter Package has been installed
2. Python Extension has been installed

<https://code.visualstudio.com/docs/python/jupyter-support>

1 # Markdown Introduction

1 ## Normalization function

2 There are different ways to normalize the data. one of the methods is shown as below.
3 $\hat{x}_{ij} = \frac{x_{ij} - \min\{x_{kj}\}}{\max\{x_{kj}\} - \min\{x_{kj}\}}, k \in \{1, 2, \dots, m\}$

```
1 def normFun(x):
2     min_x = min(x)
3     max_x = max(x)
4     result = []
5     for j in x:
6         xj = (j - min_x) / (max_x - min_x)
7         result.append(xj)
8
9     return result
```

```
1 x = [1, 2, 3, 4, 5, 6]
2 normFun(x)
```

Markdown
language

Source

Python
language

1 Markdown Introduction

1.1 Normalization function

There are different ways to normalize the data. one of the methods is shown as below.

$$\hat{x}_{ij} = \frac{x_{ij} - \min x_{kj}}{\max x_{kj} - \min x_{kj}}, k \in \{1, 2, \dots, m\}$$

```
In [1]: 1 def normFun(x):
2     min_x = min(x)
3     max_x = max(x)
4     result = []
5     for j in x:
6         xj = (j - min_x) / (max_x - min_x)
7         result.append(xj)
8
9     return result
```

```
In [2]: 1 x = [1, 2, 3, 4, 5, 6]
2 normFun(x)
```

Out[2]: [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]

Output



Markdown is a way to style text on the web. You control the display of the document; formatting words as bold or italic, adding images, and creating lists are just a few of the things we can do with Markdown. Mostly, Markdown is just regular text with a few non-alphabetic characters thrown in, like # or *.

HEADERS

```
# This is an <h1> tag
## This is an <h2> tag
##### This is an <h6> tag
```

EMPHASIS

```
*This text will be italic*
_This will also be italic_

**This text will be bold**
__This will also be bold__

*You **can** combine them*
```

BLOCKQUOTES

```
As Grace Hopper said:
> I've always been more interested
> in the future than in the past.
```

```
As Grace Hopper said:
I've always been more interested
in the future than in the past.
```

LISTS

Unordered

- * Item 1
- * Item 2
- * Item 2a
- * Item 2b

Ordered

1. Item 1
2. Item 2
3. Item 3
 - * Item 3a
 - * Item 3b

IMAGES

![GitHub Logo](/images/logo.png)

Format: ! [Alt Text] (url)

LINKS

<http://github.com> - automatic!

[GitHub](http://github.com)

BACKSLASH ESCAPES

Markdown allows you to use backslash escapes to generate literal characters which would otherwise have special meaning in Markdown's formatting syntax.

literal asterisks

literal asterisks

Markdown provides backslash escapes for the following characters:

\ backslash	() parentheses
' backtick	# hash mark
* asterisk	+ plus sign
_ underscore	- minus sign (hyphen)
{ curly braces	. dot
[] square brackets	! exclamation mark

<https://guides.github.com/pdfs/markdown-cheatsheet-online.pdf>



USERNAME @Mentions

Typing an `@` symbol, followed by a username, will notify that person to come and view the comment.

This is called an “@mention”, because you’re mentioning the individual. You can also @mention teams within an organization.

ISSUE REFERENCES

Any number that refers to an Issue or Pull Request will be automatically converted into a link.

```
#1
github-flavored-markdown#1
defunkt/github-flavored-markdown#1
```

EMOJI

To see a list of every image we support, check out www.emoji-cheat-sheet.com

```
GitHub supports emoji!
:+1: :sparkles: :camel: :tada:
:rocket: :metal: :octocat:
```

GitHub supports emoji!



FENCED CODE BLOCKS

Markdown converts text with four leading spaces into a code block; with GFM you can wrap your code with ````` to create a code block without the leading spaces. Add an optional language identifier and your code will get syntax highlighting.

```
```javascript
function test() {
 console.log("look ma', no spaces");
}
...
```

```

```
function test() {
  console.log("look ma', no spaces");
}
```

TASK LISTS

- [x] this is a complete item
- [] this is an incomplete item
- [x] @mentions, #refs, [links](), **formatting**, and tags supported
- [x] list syntax required (any unordered or ordered list supported)

- this is a complete item
- this is an incomplete item
- @mentions, #refs, links, formatting, and tags supported
- list syntax required (any unordered or ordered list supported)

TABLES

You can create tables by assembling a list of words and dividing them with hyphens `-` (for the first row), and then separating each column with a pipe `|`:

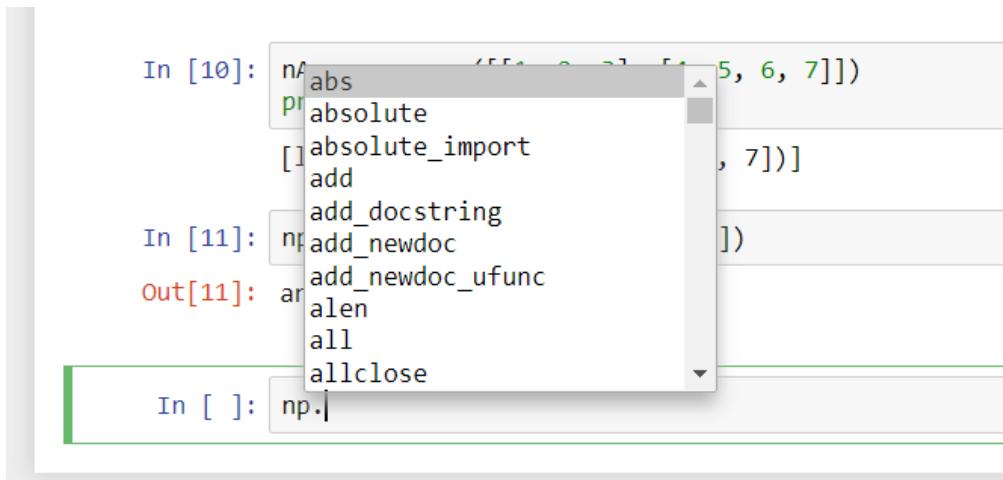
| First Header | Second Header |
|------------------|------------------|
| Content cell 1 | Content cell 2 |
| Content column 1 | Content column 2 |

| First Header | Second Header |
|------------------|------------------|
| Content cell 1 | Content cell 2 |
| Content column 1 | Content column 2 |

Auto-completion

Hit “Tab” when you use it.

If not work, try this first and then hit Tab.
%config IPCompleter.greedy=True



Or install bnextensions package for Jupyter to work without “Tab”

<https://jupyter-contrib-nbextensions.readthedocs.io/en/latest/install.html>



- Shut down Jupyter:
 - Click kernel -> Shutdown or
 - Press two times, Ctr + C, in PowerShell to shut down Jupyter.



Python For Data Science Cheat Sheet

Jupyter Notebook

Learn More Python for Data Science Interactively at www.DataCamp.com

Saving/Loading Notebooks

Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells.

Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

Command Mode:

The Jupyter Notebook interface in Command Mode.

Widgets

Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

1. Save and checkpoint
2. Interrupt kernel
3. Restart kernel
4. Display characteristics
5. Open command palette
6. Current kernel
7. Kernel status
8. Log out from notebook server

Asking For Help

Walk through a UI tour

DataCamp

Learn Python for Data Science Interactively

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Jupyter_Notebook_Cheat_Sheet.pdf

K.K.F. Yuen

14



1. NumPy Array and Vectorised Operations
2. Subsetting NumPy Array
3. NumPy Array Attributes



Python For Data Science Cheat Sheet

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com



NumPy

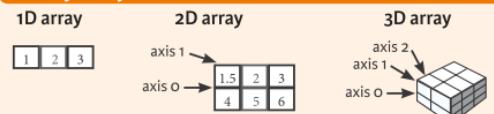
The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```



NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]),
      dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))
>>> np.ones((2,3,4),dtype=np.int16)
>>> d = np.arange(10,25,5)

>>> np.linspace(0,2,9)

>>> e = np.full((2,2),7)
>>> f = np.eye(2)
>>> np.random.random((2,2))
>>> np.empty((3,2))
```

Create an array of zeros
Create an array of ones
Create an array of evenly spaced values (step value)
Create an array of evenly spaced values (number of samples)
Create a constant array
Create a 2x2 identity matrix
Create an array with random values
Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savetxt('array.npy', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64
>>> np.float32
>>> np.complex
>>> np.bool_
>>> np.object_
>>> np.string_
>>> np.unicode_
```

Signed 64-bit integer types
Standard double-precision floating point
Complex numbers represented by 128 floats
Boolean type storing TRUE and FALSE values
Python object type
Fixed-length string type
Fixed-length unicode type

Inspecting Your Array

```
>>> a.shape
>>> len(a)
>>> b.ndim
>>> e.size
>>> b.dtype
>>> b.dtype.name
>>> b.astype(int)
```

Array dimensions
Length of array
Number of array dimensions
Number of array elements
Data type of array elements
Name of data type
Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b
array([-0.5,  0. ,  0. ],
      [-3. , -3. , -3. ])
>>> np.subtract(a,b)
>>> b + a
array([ 2.5,  4. ,  6.1,
      [ 5. ,  7. ,  9.1]])
>>> np.add(b,a)
>>> a / b
array([ 0.66666667,  1. ,
      [ 0.25,  0.4,  0.5 ],
      [ 1.5,  4. ,  9. ],
      [ 4. ,  10. ,  18. ]])
>>> np.divide(a,b)
>>> a * b
array([ 1.5,  4. ,  9. ],
      [ 4. ,  10. ,  18. ])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
array([[ 7.,  7.],
      [ 7.,  7.1],
```

Subtraction
Subtraction
Addition
Addition
Division
Multiplication
Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product

Comparison

```
>>> a == b
array([False, True, True],
      [False, False, False]), dtype=bool)
>>> a < 2
array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

Element-wise comparison
Element-wise comparison
Array-wise comparison

Aggregate Functions

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> a.corrcoef()
>>> np.std(b)
```

Array-wise sum
Array-wise minimum value
Maximum value of an array row
Cumulative sum of the elements
Mean
Median
Correlation coefficient
Standard deviation

Copying Arrays

```
>>> h = a.view()
>>> np.copy(a)
>>> h = a.copy()
```

Create a view of the array with the same data
Create a copy of the array
Create a deep copy of the array

Sorting Arrays

```
>>> a.sort()
>>> c.sort(axis=0)
```

Sort an array
Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

| | | |
|-------------------|--------------------------|--|
| Subsetting | | Select the element at the 2nd index |
| >>> a[2] | 3 | Select the element at row 1 column 2 (equivalent to b[1][2]) |
| >>> b[1,2] | 6.0 | Select items at index 0 and 1 |
| >>> b[0:2,1] | array([2., 5.]) | Select all items at row 0 (equivalent to b[0:1, :]) |
| >>> b[:,1] | array([[1.5, 2., 3.], | Same as [1,:,:] |
| >>> c[1,...] | array([[[3., 2., 1.], | Reversed array a |
| >>> a[:1] | [4., 5., 6.]]) | Select elements from a less than 2 |
| >>> a[:1:1] | array([3.]) | Select elements (1,0),(0,1),(1,2) and (0,0) |
| >>> a[::2] | array([1.]) | Select a subset of the matrix's rows and columns |

Array Manipulation

| | | |
|--|---|--|
| Transposing Array | | Permute array dimensions
Permute array dimensions |
| >>> i = np.transpose(b) | i.T | Flatten the array
Reshape, but don't change data |
| >>> b.ravel() | | Return a new array with shape (2,6) |
| >>> g.reshape(3,-2) | | Append items to an array |
| >>> np.append(h,g) | | Insert items in an array |
| >>> np.insert(a, 1, 5) | | Delete items from an array |
| >>> np.delete(a,[1]) | | Concatenate arrays |
| >>> np.concatenate((a,d),axis=0) | | Stack arrays vertically (row-wise) |
| >>> np.vstack((a,b)) | array([[1., 2., 3., 4., 5., 6.]]) | Stack arrays vertically (row-wise) |
| >>> np.r_[e,f] | array([7., 7., 1., 0.]) | Stack arrays horizontally (column-wise) |
| >>> np.hstack((e,f)) | array([7., 7., 1., 0.]) | Create stacked column-wise arrays |
| >>> np.column_stack((a,d)) | array([[1., 10., | Create stacked column-wise arrays |
| >>> np.column_stack((a,d)) | [2., 15., | |
| >>> np.c_[a,d] | [3., 20.])] | |
| >>> np.hsplit(a,3) | | Split the array horizontally at the 3rd index |
| >>> (array([1]),array([2]),array([3])) | | Split the array vertically at the 2nd index |
| >>> np.vsplit(c,2) | array([[1.5, 2., 3.], | |
| >>> np.vsplit(c,2) | [4., 5., 6.]]]) | |

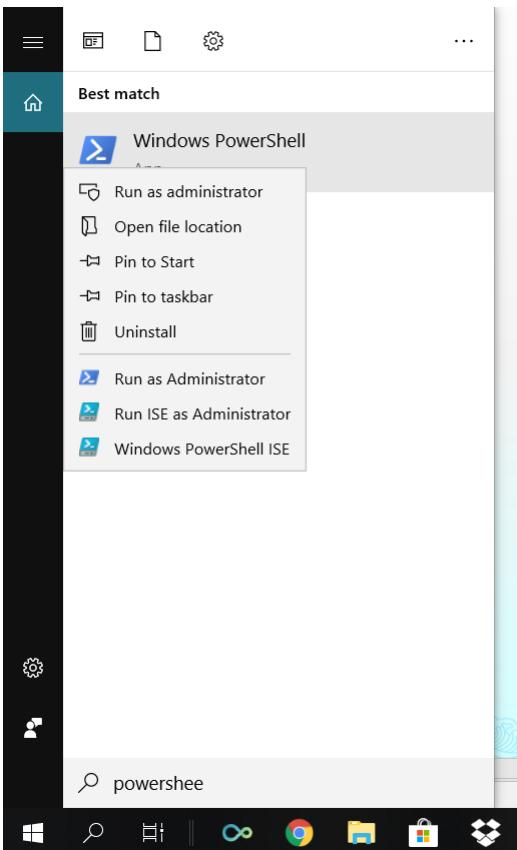
DataCamp

Learn Python for Data Science Interactively



First time to use numpy

Step 1: right click
Windows PowerShell and
run as administrator



Step 2: pip install numpy

```
Administrator: Windows PowerShell
PS C:\> pip install numpy
```

Step 3 (optional): where the package installed?

C:\Program Files\Python37\Lib\site-packages\numpy

List vs numpy array

In [1]:

```
import numpy as np
# create a list
a = [1, 2, 3]
print(a * 2)
```

```
[1, 2, 3, 1, 2, 3]
```

In [2]:

```
# convert a list to numpy array
na = np.array(a)
print(na * 2)
```

```
[2 4 6]
```

Discussion 1

Using Numpy's arrays can be 10 times faster than Python's lists. To enable the speed,

- Numpy arrays are fixed in size (but can change the values), unlike lists which can change in size.
- Numpy arrays must all be the same type whereas lists can hold any type. By restricting ndarrays in this way, it makes ndarrays both much more space efficient than lists, but also opens up a range of memory and computational optimizations.

About the size

```
In [ ]: 1 A1 = np.array([[1, 2, 3], [4, 5, 6, 7]])  
        2 print(A1)
```

```
In [ ]: 1 print(A1[0])
```

```
In [ ]: 1 print(A1 * 2)
```

```
In [ ]: 1 A2 = np.array([[1, 2, 3], [4, 5, 6]])  
        2 print(A2)
```

```
In [ ]: 1 print(A2[0, 1])
```

```
In [ ]: 1 print(A2 * 2)
```



▼ 4.2 About the type

```
In [ ]: 1 listType = ["a", True, 1, (1, 2, 3), ['a', 'b', 'c']]
```

```
In [ ]: 1 print(listType)
```

```
In [ ]: 1 arrayType = np.array( ["a", True, 1, (1, 2, 3), ['a', 'b', 'c']])
```

```
In [ ]: 1 print(type(arrayType))
```

```
In [ ]: 1 print(arrayType)
```

```
In [ ]: 1 arrayType * 2
```

```
In [ ]: 1 print(type(arrayType[0]), type(arrayType[1]), type(arrayType[2]), type(arrayType[3]))
```



Quiz 1

Which of the following is correct to create a 2D array?

np.array([[11,12,13],[21,22,23]])

np.array([11,12,13],[21,22,23])

np.array([11,12,13,21,22,23])

np.array[[11,12,13,21,22,23]]

```
In [36]: 1 np.array([[11,12,13],[21,22,23]])
```

```
Out[36]: array([[11, 12, 13],  
                 [21, 22, 23]])
```

```
In [37]: 1 np.array([11,12,13],[21,22,23])
```

```
-----  
TypeError                                         Traceback (most recent call last)  
<ipython-input-37-0584aa6c4d7a> in <module>  
----> 1 np.array([11,12,13],[21,22,23])
```

TypeError: data type not understood

```
In [38]: 1 np.array([11,12,13,21,22,23])
```

```
Out[38]: array([11, 12, 13, 21, 22, 23])
```

```
In [39]: 1 np.array[[11,12,13,21,22,23]]
```

```
-----  
TypeError                                         Traceback (most recent call last)  
<ipython-input-39-82183304a2f6> in <module>  
----> 1 np.array[[11,12,13,21,22,23]]
```

TypeError: 'builtin_function_or_method' object is not subscriptable

Quiz 2

Do the two NumPy arrays have the same shape?

```
first_array = np.array([3, 33, 333])
```

```
second_array = np.array([[3, 33, 333]])
```

```
In [ ]: 1 first_array = np.array([3, 33, 333])
```

```
In [ ]: 1 second_array = np.array([[3, 33, 333]])
```

```
In [ ]: 1 print(first_array)
```

```
In [ ]: 1 first_array.shape
```

```
In [ ]: 1 print(second_array)
```

```
In [ ]: 1 second_array.shape
```

Other ways to create NumPy arrays?
<https://www.numpy.org/devdocs/user/quickstart.html#array-creation>

Write code to create each of the following NumPy arrays.

```
array([[ 0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.]])
```

```
array([10, 15, 20, 25])
```

```
array([ 0. ,  0.25,  0.5 ,  0.75,  1. ,  1.25,  1.5 ,  1.75,  2. ])
```

Discussion 2

```
In [2]: 1 np.zeros( 3,4 )  
Out[2]: array([[0., 0., 0., 0.],  
               [0., 0., 0., 0.],  
               [0., 0., 0., 0.]])
```



```
In [3]: 1 np.arange( 10, 30, 5 )  
Out[3]: array([10, 15, 20, 25])
```



```
In [4]: 1 np.linspace( 0, 2, 9 )  
Out[4]: array([0. , 0.25, 0.5 , 0.75, 1. , 1.25, 1.5 , 1.75, 2. ])
```

Discussion 3

```
a = np.array( [20,30,40,50] )  
b = np.arange( 4 )
```

What does each of the following evaluate to?

a-b

b**2

a<35

```
In [5]: 1 a = np.array( [20,30,40,50] )  
         2 b = np.arange( 4 )
```

```
In [6]: 1 print(a)  
[20 30 40 50]
```

```
In [7]: 1 print(b)  
[0 1 2 3]
```

```
In [8]: 1 a - b  
out[8]: array([20, 29, 38, 47])
```

```
In [9]: 1 b ** 2  
out[9]: array([0, 1, 4, 9], dtype=int32)
```

```
In [10]: 1 a < 35  
out[10]: array([ True,  True, False, False])
```

Discussion 4

```
a = np.array([[0,0],[0,0]])  
b1 = np.array([1,1])  
b2 = 1
```

Do `print(a+b1)` and `print(a+b2)` result in the same matrix?

```
In [ ]: 1 a = np.array([[0,0],[0,0]])  
         2 b1 = np.array([1,1])  
         3 b2 = 1
```

```
In [ ]: 1 print(a, b1, b2)
```

```
In [ ]: 1 print(a + b1)
```

```
In [ ]: 1 print(a + b2)
```

Discussion 5

What's printed after executing the code below?

```
arrA = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])  
arrB = [0,1,0,2]  
print(arrA + arrB)
```

```
In [22]: 1 arrA = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])  
          2 arrB = [0,1,0,2]
```

```
In [23]: 1 print(arrA, " + ", arrB, " = ")  
          [[ 1  2  3  4]  
           [ 5  6  7  8]  
           [ 9 10 11 12]] + [0, 1, 0, 2] =
```

```
In [24]: 1 print( arrA + arrB)  
          [[ 1  3  3  6]  
           [ 5  7  7 10]  
           [ 9 11 11 14]]
```



Quiz 3

After the code below has been executed, what value does `a[i]` refer to?

```
a = np.arange(12)**2
i = np.array( [ 1,1,3,8,5 ] )
print(a[i])
```

Then execute the code below. What will happen?

`a[i] = 'OK'`

```
In [2]: 1 a = np.arange(12)**2
          2 i = np.array( [ 1,1,3,8,5 ] )
```

```
In [3]: 1 print(a)
          2 print(i)

[ 0   1   4   9  16  25  36  49  64  81 100 121]
[1 1 3 8 5]
```

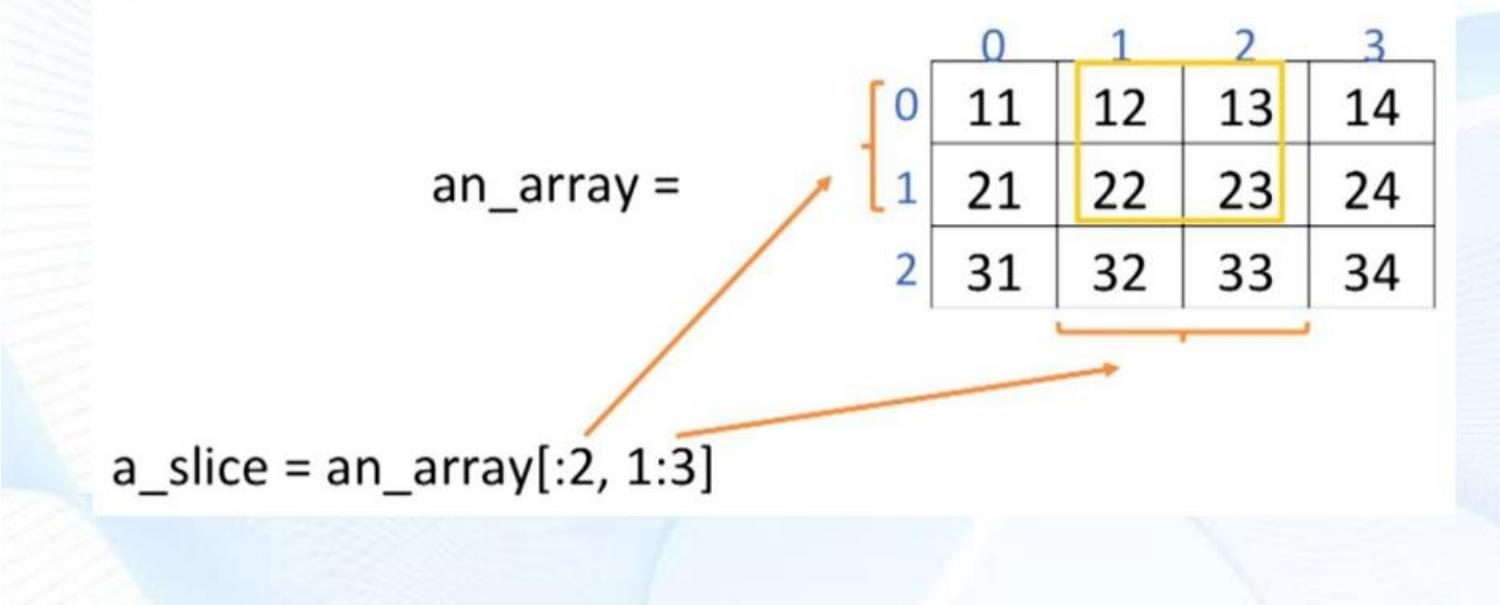
```
In [4]: 1 print(a[i])
          2
```

[1 1 9 64 25]

```
In [6]: 1 a[i] = 'OK'
```

```
-----  
ValueError                                     Traceback (most recent call last)  
<ipython-input-6-3cfea04ff6ba> in <module>  
----> 1 a[i] = 'OK'  
  
ValueError: invalid literal for int() with base 10: 'OK'
```

Subsetting NumPy 2D arrays



[Inclusive, exclusive]

Or

[index a, index b -1] in



Quiz 4

Given a 3-by-3 NumPy array called “arr”, how many rows will be returned by calling arr[:2,]?

- 2
- 3

IndexError: index out of bounds

- 1

```
In [7]: 1 arr = np.arange(9).reshape(3, 3)
```

```
In [8]: 1 print(arr)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

```
In [9]: 1 arr[:2,]
```

```
Out[9]: array([[0, 1, 2],
 [3, 4, 5]])
```

Quiz 5

```
an_array = np.array([[11,12,13,14], [21,22,23,24], [31,32,33,34]])
```

What does each of the following evaluate to?

an_array.shape

an_array[1, :]

an_array[1:2, :]

an_array[:, 1]

an_array[:, 1:2]

```
In [2]: 1 an_array = np.array([[11,12,13,14], [21,22,23,24], [31,32,33,34]])
2 print(an_array)
```

```
[[11 12 13 14]
 [21 22 23 24]
 [31 32 33 34]]
```

```
In [3]: 1 an_array.shape
```

```
Out[3]: (3, 4)
```

```
In [4]: 1 an_array[1, :]
```

```
Out[4]: array([21, 22, 23, 24])
```

```
In [5]: 1 an_array[1:2, :]
```

```
Out[5]: array([[21, 22, 23, 24]])
```

```
In [6]: 1 an_array[:, 1]
```

```
Out[6]: array([12, 22, 32])
```

```
In [7]: 1 an_array[:, 1:2]
```

```
Out[7]: array([[12],
 [22],
 [32]])
```



Quiz 6

```
an_array = np.array([[11,12,13,14], [21,22,23,24], [31,32,33,34]])
```

Write code to print

elements greater than 15

elements greater than 15 and less than 30

elements which are even

```
In [8]: 1 an_array % 2
```

```
Out[8]: array([[1, 0, 1, 0],
               [1, 0, 1, 0],
               [1, 0, 1, 0]], dtype=int32)
```

```
In [9]: 1 an_array % 2 == 0
```

```
Out[9]: array([[False,  True, False,  True],
               [False,  True, False,  True],
               [False,  True, False,  True]])
```

```
In [10]: 1 an_array[an_array % 2 == 0]
```

```
Out[10]: array([12, 14, 22, 24, 32, 34])
```

```
In [3]: 1 an_array = np.array([[11,12,13,14], [21,22,23,24], [31,32,33,34]])
2 print(an_array)
```

```
[[11 12 13 14]
 [21 22 23 24]
 [31 32 33 34]]
```

```
In [4]: 1 an_array > 15
```

```
Out[4]: array([[False, False, False, False],
               [ True,  True,  True,  True],
               [ True,  True,  True,  True]])
```

```
In [5]: 1 an_array[an_array > 15]
```

```
Out[5]: array([21, 22, 23, 24, 31, 32, 33, 34])
```

```
In [6]: 1 index = (30 > an_array) & (an_array > 15)
2 index
```

```
Out[6]: array([[False, False, False, False],
               [ True,  True,  True,  True],
               [False, False, False, False]])
```

```
In [7]: 1 an_array[index]
```

```
Out[7]: array([21, 22, 23, 24])
```

Discussion 6

```
an_array = np.array([[11,12,13,14], [21,22,23,24], [31,32,33,34]])
```

Write code to change the elements in an_array to

[[100011 12 13] [21 100022 23] [31 32 100033] [100041 42 43]]

```
In [2]: 1 an_array = np.array([[11,12,13,14], [21,22,23,24], [31,32,33,34]])
```

```
In [3]: 1 print(an_array)
```

```
[[11 12 13 14]
 [21 22 23 24]
 [31 32 33 34]]
```

```
In [4]: 1 an_array.resize(4,3)
```

```
In [5]: 1 print(an_array)
```

```
[[11 12 13]
 [14 21 22]
 [23 24 31]
 [32 33 34]]
```

```
In [12]: 1 an_array[1,] = [21, 10022, 23]
2 an_array[2,] = [31, 32, 100033]
3 an_array[3,] = [100041, 42, 43]
```

```
In [13]: 1 print(an_array)
```

```
[[ 10011      12      13]
 [      21     10022      23]
 [      31      32 100033]
 [100041      42      43]]
```



4. NumPy Functions for Statistics and Random Sampling

Discussion 7

Working with NumPy arrays' methods

<https://www.numpy.org/devdocs/reference/arrays.ndarray.html#array-methods>

Many operations are implemented as methods of the NumPy array objects.

```
b = np.arange(12).reshape(3,4)
b.sum(axis=0)
b.min(axis=1)
b.cumsum(axis=1)
```

In [16]:

| | |
|---|--------------------------------|
| 1 | b = np.arange(12).reshape(3,4) |
| 2 | print(b) |

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [17]:

| | |
|---|---------------|
| 1 | b.sum(axis=0) |
|---|---------------|

Out[17]: array([12, 15, 18, 21])

In [18]:

| | |
|---|---------------|
| 1 | b.min(axis=1) |
|---|---------------|

Out[18]: array([0, 4, 8])

In [19]:

| | |
|---|------------------|
| 1 | b.cumsum(axis=1) |
|---|------------------|

Out[19]: array([[0, 1, 3, 6],
 [4, 9, 15, 22],
 [8, 17, 27, 38]], dtype=int32)



5. Plotting with matplotlib

<https://matplotlib.org/gallery/index.html>

```
PS C:\> python -m pip install matplotlib
```

```
In [ ]: 1 # !pip install matplotlib
```

Source: <https://www.numpy.org/devdocs/user/quickstart.html#histograms>

```
In [3]: 1 import numpy as np
2 import matplotlib.pyplot as plt
```

```
In [4]: 1 # Build a vector of 10000 normal deviates with variance 0.5^2 and mean 2
2 mu, sigma = 2, 0.5
```

```
In [5]: 1 v = np.random.normal(mu,sigma,10000)
2 v
```

```
Out[5]: array([2.31192204, 1.92572497, 2.57443487, ..., 1.87154162, 2.21761945,
   1.8429948 ])
```

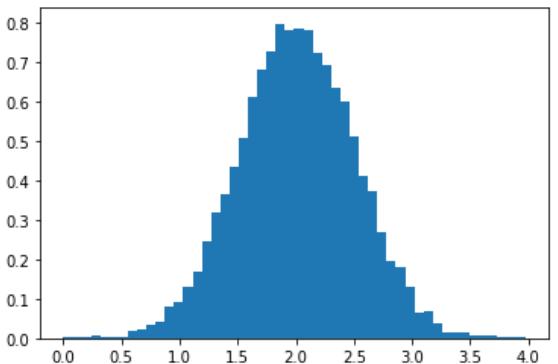
```
In [7]: 1 v.mean()
```

```
Out[7]: 2.010053755796068
```

```
In [8]: 1 v.std()
```

```
Out[8]: 0.49719348966733545
```

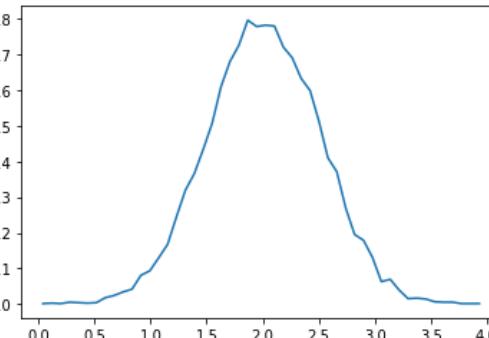
```
In [11]: 1 plt.hist(v, bins=50, density=1)
2 plt.show()
```



```
In [13]: 1 # Compute the histogram with numpy and then plot it
2 (n, bins) = np.histogram(v, bins=50, density=True) # NumPy version (no plot)
3 print(n)
4 print(bins)
```

```
[0.00125829 0.00251657 0.00125829 0.00503314 0.00377486 0.00251657
 0.00377486 0.017616 0.02390742 0.03397371 0.04152342 0.08053026
 0.09311312 0.12960339 0.16735196 0.24536565 0.31960449 0.36616105
 0.43410846 0.50708901 0.61026841 0.68073239 0.72603066 0.79649465
 0.77887865 0.78265351 0.78013694 0.72899752 0.69079867 0.63291755
 0.59894384 0.51212215 0.41020103 0.37119419 0.26927307 0.19503423
 0.17867652 0.13086168 0.06291427 0.0692057 0.03900685 0.01509942
 0.01635771 0.01384114 0.00629143 0.00503314 0.00503314 0.00125829
 0.00125829 0.00125829]
[-7.84441858e-04 7.86887854e-02 1.58162013e-01 2.37635240e-01
 3.17108467e-01 3.96581694e-01 4.76054921e-01 5.55528149e-01
 6.35001376e-01 7.14474603e-01 7.93947830e-01 8.73421057e-01
 9.52894285e-01 1.03236751e+00 1.11184074e+00 1.19131397e+00
 1.27078719e+00 1.35026042e+00 1.42973365e+00 1.50920688e+00
 1.58868010e+00 1.66815333e+00 1.74762656e+00 1.82709978e+00
 1.90657381e+00 1.98604624e+00 2.06551947e+00 2.14499269e+00
 2.22446592e+00 2.30393915e+00 2.38341237e+00 2.46288560e+00
 2.54235883e+00 2.62183206e+00 2.70130528e+00 2.78077851e+00
 2.86025174e+00 2.93972496e+00 3.01919819e+00 3.09867142e+00
 3.17814465e+00 3.25761787e+00 3.33709110e+00 3.41656433e+00
 3.49603756e+00 3.57551078e+00 3.65498401e+00 3.73445724e+00
 3.81393046e+00 3.89340369e+00 3.97287692e+00]
```

```
In [14]: 1 plt.plot(.5*(bins[1:]+bins[:-1]), n)
2 plt.show()
```





A lot of examples with source codes

matplotlib.org/gallery/index.html

The screenshot shows the Matplotlib gallery page. At the top, there's a large logo for 'matplotlib Version 3.1.1'. Below it, a navigation bar includes links for 'home | examples | tutorials | API | contents »' on the left and 'modules | index' on the right. A prominent orange button on the right says 'Fork me on GitHub'. On the left, a section titled 'Gallery' contains four examples: 'Stacked Bar Graph', 'Grouped bar chart with labels', 'Horizontal bar chart', and 'Broken Barh'. Below these are four more examples: 'Categorical Plotting', 'Time Series Plotting', 'Frequency Plotting', and '3D Plotting'. To the right of the examples is a sidebar with a 'Quick search' input field and a 'Go' button, followed by a 'Table of Contents' section and a detailed 'Gallery' navigation menu.

- Lines, bars and markers
- Images, contours and fields
- Subplots, axes and figures
- Statistics
- Pie and polar charts
- Text, labels and annotations
- Pyplot
- Color
- Shapes and collections
- Style sheets
- Axes Grid
- Axis Artist
- Showcase
- Animation
- Event handling
- Front Page
- Miscellaneous
- 3D plotting

Discussion 8

- How to set the min and max of x and y axes?

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.axis.html#matplotlib.pyplot.axis

- How to customize the color, marker and line style?

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot

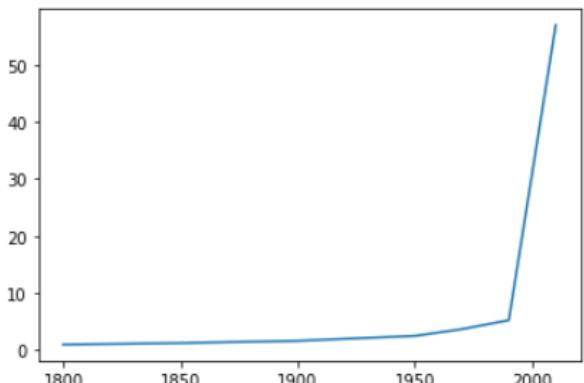
```

1 import matplotlib.pyplot as plt
2 x = [1800, 1850, 1900, 1950, 1970, 1990, 2010]
3 y = [1.0, 1.262, 1.650, 2.519, 3.692, 5.263, 56.972]

```

```
1 plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x1ef55ab8688>]
```

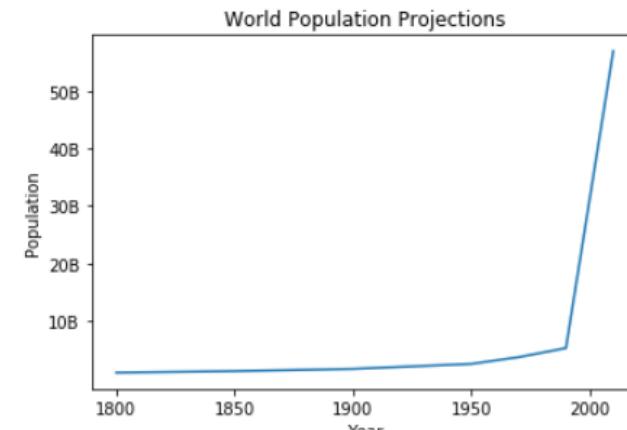


```

1 plt.plot(x, y)
2 plt.xlabel('Year')
3 plt.ylabel('Population')
4 plt.title("World Population Projections")
5 plt.yticks([10,20,30,40,50], ['10B','20B','30B','40B','50B'])
6 plt.show()

```

K.K.F. Yuen



matplotlib.pyplot.axis

`matplotlib.pyplot.axis(*args, **kwargs)`

[source]

Convenience method to get or set some axis properties.

Call signatures:

```
xmin, xmax, ymin, ymax = axis()  
xmin, xmax, ymin, ymax = axis([xmin, xmax, ymin, ymax])  
xmin, xmax, ymin, ymax = axis(option)  
xmin, xmax, ymin, ymax = axis(**kwargs)
```

Parameters:

xmin, xmax, ymin, ymax : float, optional

The axis limits to be set. Either none or all of the limits must be given. This can also be achieved using

```
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
```

option : bool or str

If a bool, turns axis lines and labels on or off. If a string, possible values are:

| Value | Description |
|----------|---|
| 'on' | Turn on axis lines and labels. Same as True. |
| 'off' | Turn off axis lines and labels. Same as False. |
| 'equal' | Set equal scaling (i.e., make circles circular) by changing axis limits. |
| 'scaled' | Set equal scaling (i.e., make circles circular) by changing dimensions of the plot box. |
| 'tight' | Set limits just large enough to show all data. |
| 'auto' | Automatic scaling (fill plot box with data). |
| 'normal' | Same as 'auto'; deprecated. |

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.axis.html#matplotlib.pyplot.axis



matplotlib.pyplot.plot

```
matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)
```

[sou

Plot y versus x as lines and/or markers.

Call signatures:

```
plot([x], y, [fmt], *, data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

The coordinates of the points or line nodes are given by x, y.

The optional parameter *fmt* is a convenient way for defining basic formatting like color, marker and linestyle. It's a shortcut string notation described in the *Notes* section below.

fmt = '[marker][line][color]'

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot

Line Styles

| character | description |
|-----------|---------------------|
| '-' | solid line style |
| '--' | dashed line style |
| '-..' | dash-dot line style |
| '::' | dotted line style |

`fmt = '[marker][line][color]'`
order for argument string is not important

Colors

The supported color abbreviations are the single letter codes

| character | color |
|-----------|---------|
| 'b' | blue |
| 'g' | green |
| 'r' | red |
| 'c' | cyan |
| 'm' | magenta |
| 'y' | yellow |
| 'k' | black |
| 'w' | white |

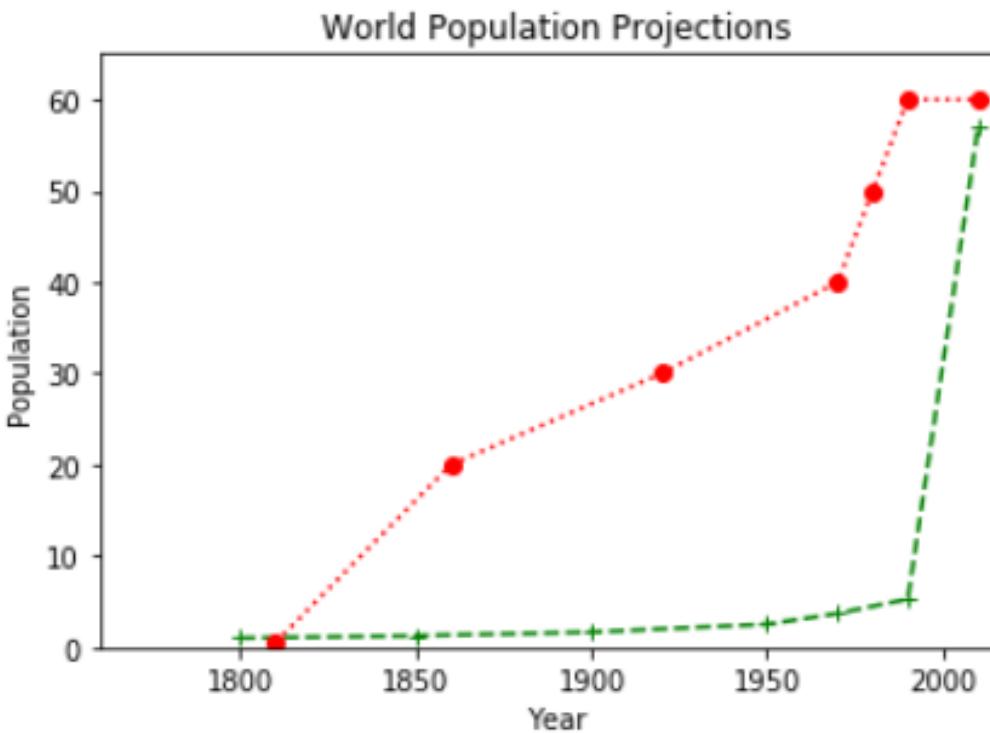


example

Markers

| character | description |
|-----------|-----------------------|
| '.' | point marker |
| ',' | pixel marker |
| 'o' | circle marker |
| 'v' | triangle_down marker |
| '^' | triangle_up marker |
| '<' | triangle_left marker |
| '>' | triangle_right marker |
| '1' | tri_down marker |
| '2' | tri_up marker |
| '3' | tri_left marker |
| '4' | tri_right marker |
| 's' | square marker |
| 'p' | pentagon marker |
| '*' | star marker |
| 'h' | hexagon1 marker |
| 'H' | hexagon2 marker |
| '+' | plus marker |
| 'x' | x marker |
| 'D' | diamond marker |
| 'd' | thin_diamond marker |
| ' ' | vline marker |
| '_' | hline marker |

```
1 x2 = [1810, 1860, 1920, 1970, 1980, 1990, 2010]
2 y2 = [0.5, 20, 30, 40, 50, 60, 60]
3 plt.plot(x, y, 'g+--', x2, y2, 'ro:')
4 plt.xlabel('Year')
5 plt.ylabel('Population')
6 plt.title("World Population Projections")
7 plt.axis([1760,2020,0,65])
8 plt.show()
```



Discussion 9

Box plot

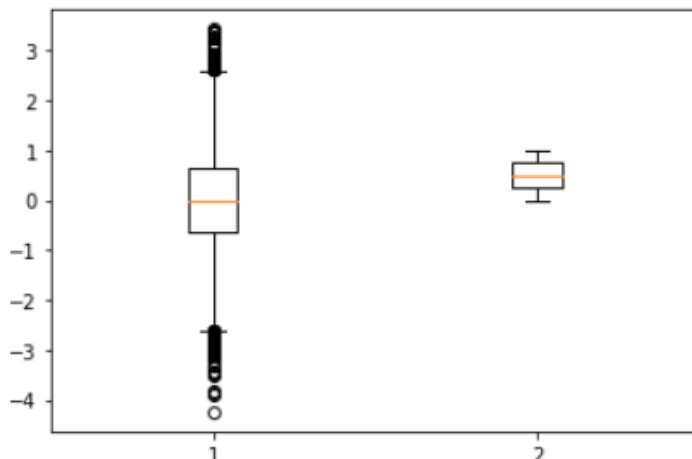
Understand the code below

```
normal_sample = np.random.normal(loc=0.0,scale=1.0,size=10000)
random_sample = np.random.random(size = 10000)
plt.figure()
plt.boxplot([normal_sample,random_sample])
```

Discuss the purposes of parameter *whis*, *showfliers*.

(https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html#matplotlib.pyplot.boxplot)

```
1 normal_sample = np.random.normal(loc=0.0,scale=1.0,size=10000)
2 random_sample = np.random.random(size = 10000)
3 plt.figure()
4 plt.boxplot([normal_sample,random_sample])
5 plt.show()
```



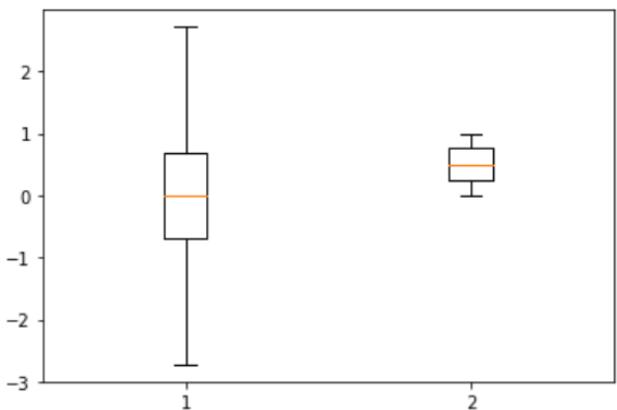
whis : float, sequence, or string (default = 1.5)

As a float, determines the reach of the whiskers to the beyond the first and third quartiles. In other words, where IQR is the interquartile range ($Q3 - Q1$), the upper whisker will extend to last datum less than $Q3 + whis * IQR$. Similarly, the lower whisker will extend to the first datum greater than $Q1 - whis * IQR$. Beyond the whiskers, data are considered outliers and are plotted as individual points. Set this to an unreasonably high value to force the whiskers to show the min and max values. Alternatively, set this to an ascending sequence of percentile (e.g., [5, 95]) to set the whiskers at specific percentiles of the data. Finally, `whis` can be the string '`range`' to force the whiskers to the min and max of the data.

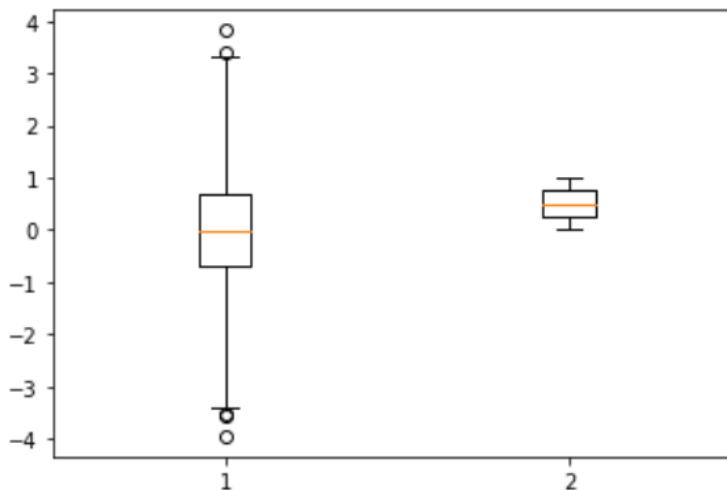
showfliers : bool, optional (True)

Show the outliers beyond the caps.

```
1 # Hide Outlier Points with showfliers=False
2 plt.figure()
3 plt.boxplot([normal_sample,random_sample], showfliers=False)
4 plt.show()
```



```
1 plt.figure()
2 plt.boxplot([normal_sample,random_sample], whis = 2)
3 plt.show()
```



Discussion 10

Scatter plot

(https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html#matplotlib.pyplot.scatter)

Understand the code below

x = np.arange(1,9)

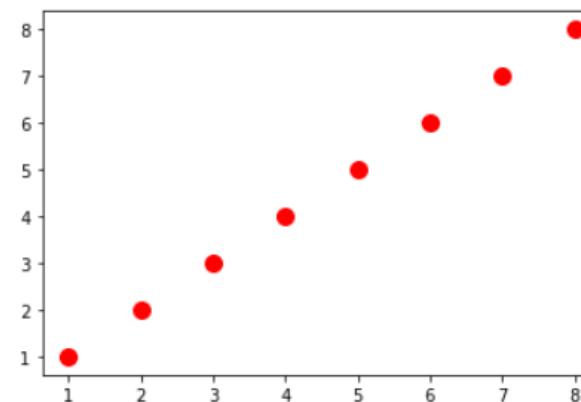
y = x

plt.figure()

plt.scatter(x,y,s=100,c='red')

```
1 x = np.arange(1,9)
2 y = x
3 plt.figure()
4 plt.scatter(x,y,s=100,c='red')
```

<matplotlib.collections.PathCollection at 0x1ef55e7c948>



Read the example at

https://matplotlib.org/gallery/shapes_and_collections/scatter.html#sphx-glr-gallery-shapes-and-collections-scatter-py and customize the color, size and transparency.



s : scalar or array_like, shape (n,), optional

The marker size in points**2. Default is `rcParams['lines.markersize'] ** 2.`

c : color, sequence, or sequence of color, optional

The marker color. Possible values:

- A single color format string.
- A sequence of color specifications of length n.
- A sequence of n numbers to be mapped to colors using `cmap` and `norm`.
- A 2-D array in which the rows are RGB or RGBA.

Note that `c` should not be a single numeric RGB or RGBA sequence because that is indistinguishable from an array of values to be colormapped. If you want to specify the same RGB or RGBA value for all points, use a 2-D array with a single row. Otherwise, value- matching will have precedence in case of a size matching with `x` and `y`.

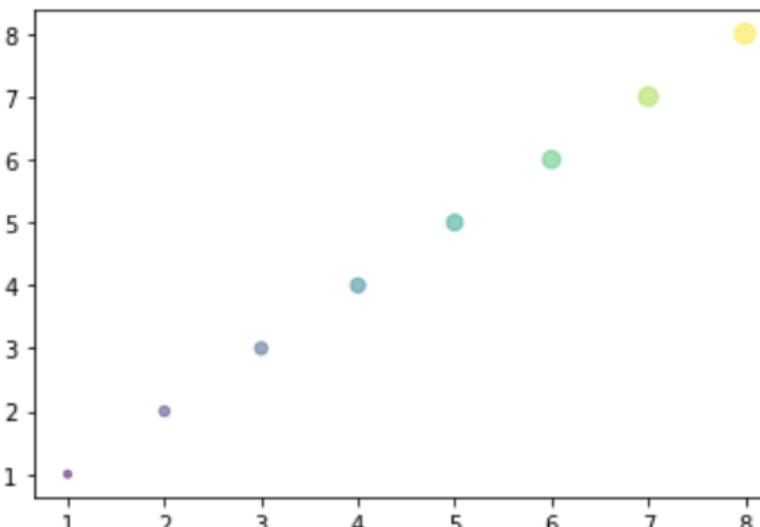
Defaults to None. In that case the marker color is determined by the value of `color`, `facecolor` or `facecolors`. In case those are not specified or None, the marker color is determined by the next color of the Axes' current "shape and fill" color cycle. This cycle defaults to `rcParams["axes.prop_cycle"]`.

alpha : scalar, optional, default: None

The alpha blending value, between 0 (transparent) and 1 (opaque).

```
1 #plt.figure()
2 plt.scatter(x,y,s=x*10,c=x*10,alpha=0.5)
```

<matplotlib.collections.PathCollection at 0x234b0570588>



Discussion 11

Bar chart

(https://matplotlib.org/api/_gen/matplotlib.pyplot.bar.html#matplotlib.pyplot.bar)

Understand the code below

```

plt.figure()
linear_data = np.arange(1,9)
quadratic_data = linear_data ** 2
xvals = np.arange(len(linear_data))
plt.bar(xvals, linear_data, width = 0.3)
    
```

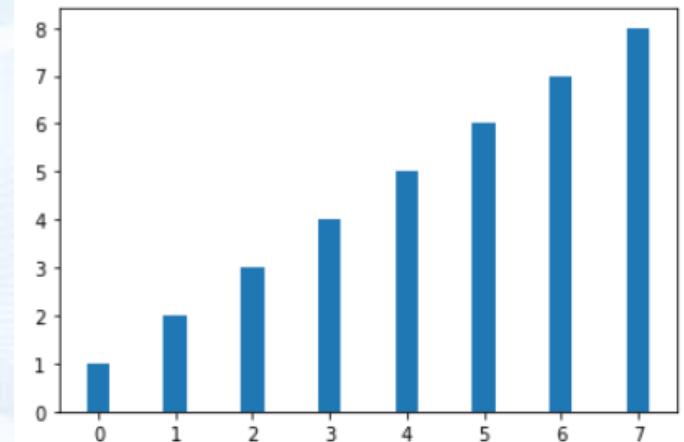
To plot the two arrays of data: linear_data and quadratic_data

- Create a side-by-side bar chart
- Read the example at
https://matplotlib.org/gallery/lines_bars_and_markers/bar_stacked.html#sphx-glr-gallery-lines-bars-and-markers-bar-stacked-py, and
 - create a stacked bar chart.
 - add legend

```

1 plt.figure()
2 linear_data = np.arange(1,9)
3 quadratic_data = linear_data ** 2
4 xvals = np.arange(len(linear_data))
5 plt.bar(xvals, linear_data, width = 0.3)
    
```

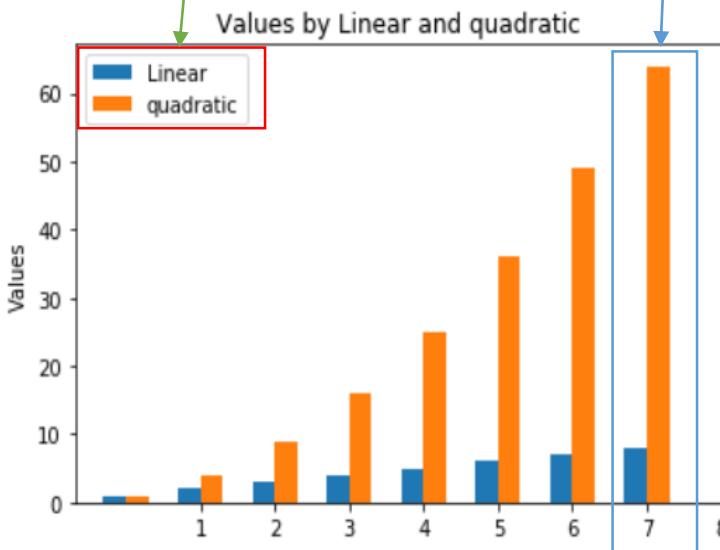
<BarContainer object of 8 artists>



```

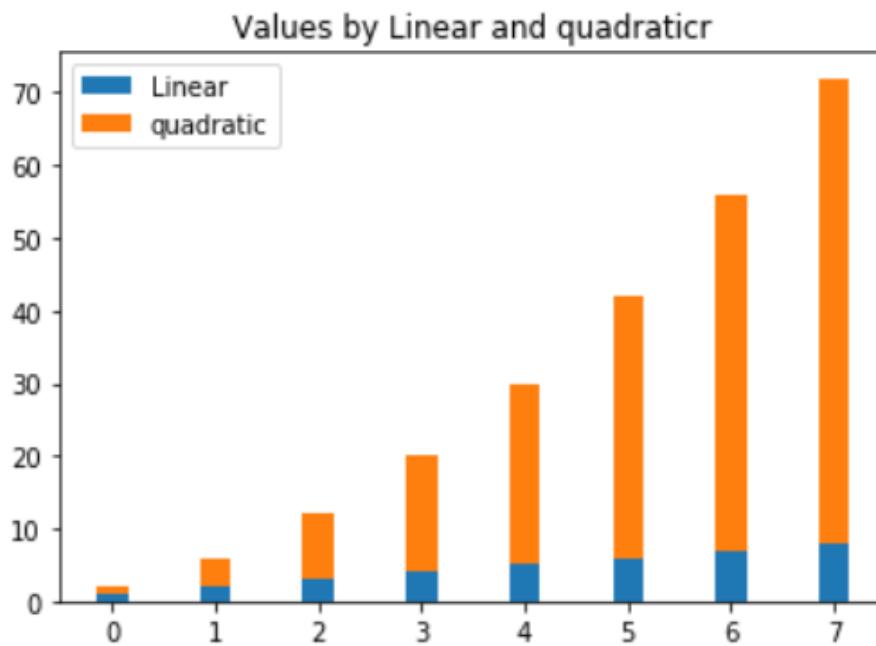
1 # data
2 linear_data = np.arange(1,9)
3 quadratic_data = linear_data ** 2
4 xvals = np.arange(len(linear_data))
5 # plot
6 fig, ax = plt.subplots()
7 # why 0.15 and 0.3?
8 plt.bar(xvals-0.15, linear_data, width = 0.3, label = 'Linear')
9 plt.bar(xvals+0.15, quadratic_data, width = 0.3, label = 'quadratic')
10 ax.set_ylabel('Values')
11 ax.set_title('Values by Linear and quadratic')
12 ax.set_xticks(x)
13 ax.legend()
14
  
```

<matplotlib.legend.Legend at 0x234b58eb148>



```

1 # data
2 linear_data = np.arange(1,9)
3 quadratic_data = linear_data ** 2
4 xvals = np.arange(len(linear_data))
5 # plot
6 plt.bar(xvals, linear_data, width = 0.3, label = 'Linear')
7 plt.bar(xvals, quadratic_data, width = 0.3, label = 'quadratic', bottom=linear_data)
8 plt.title('Values by Linear and quadraticr')
9 plt.legend()
10 plt.show()
11
  
```





References

- <https://www.numpy.org/devdocs/user/quickstart.html>
- <https://docs.scipy.org/doc/numpy/reference/>
- Numpy cheat sheet, <https://datacamp-community-prod.s3.amazonaws.com/e9f83f72-a81b-42c7-af44-4e35b48b20b7>
- Markdown, <https://github.com/adam-p/markdown-here/wiki/Markdown-Here-Cheatsheet>
- <https://guides.github.com/pdfs/markdown-cheatsheet-online.pdf>



Q&A

Thank you