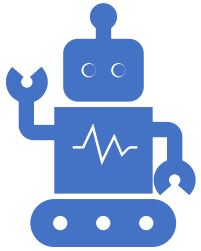
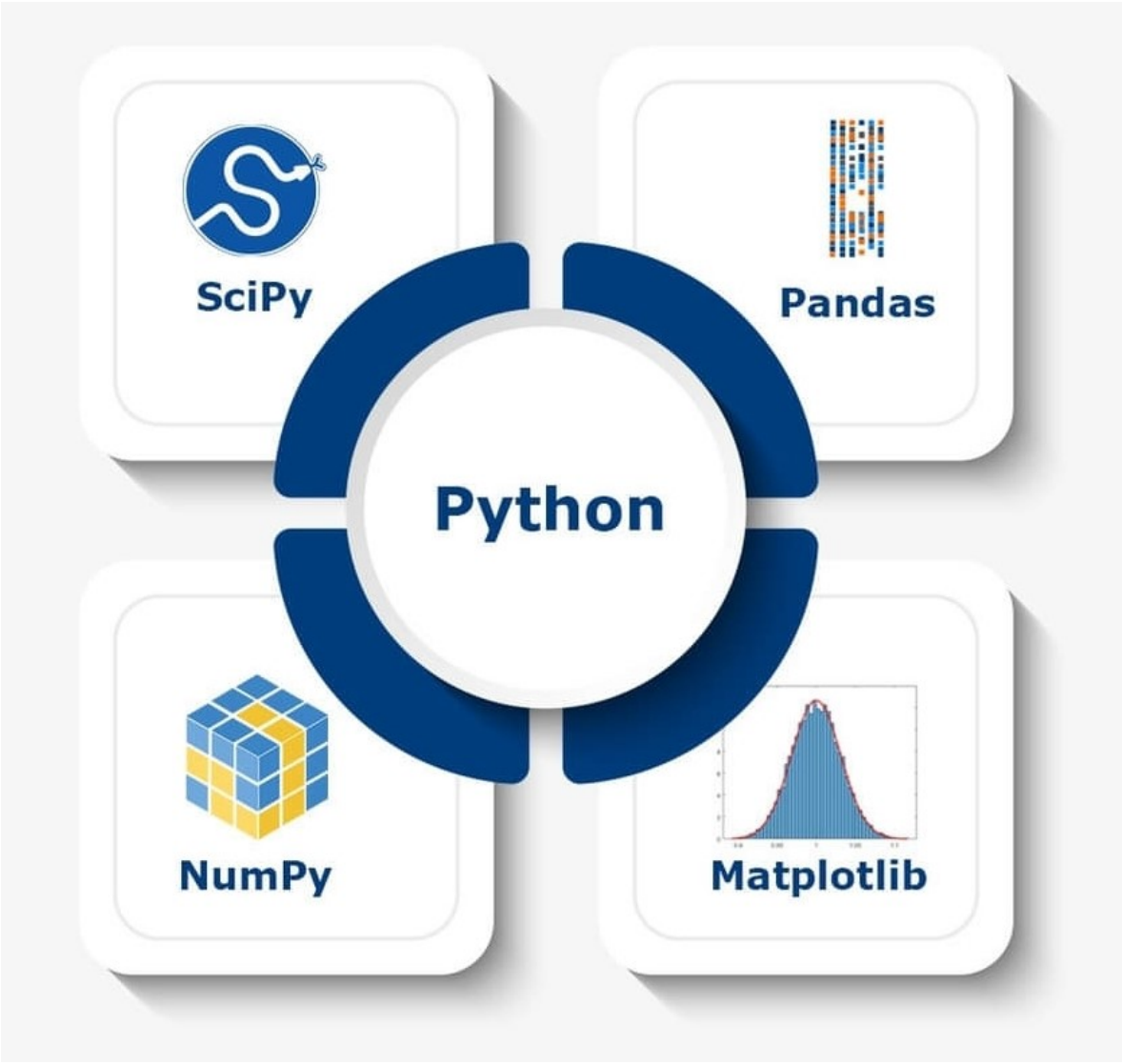


Python for Machine Learning II

Dr. Junya Michanan

Agenda

- NumPy
- SciPy
- Pandas
- Class Activity
- Homework



What is NumPy?

- NumPy is a Python library used for working with arrays.
- It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open-source project and you can use it freely.
- NumPy stands for Numerical Python.

The source code for NumPy is located at this github repository https://github.com/numpy/numpy

Why Use NumPy?

- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- The array object in NumPy is called ***ndarray***, it provides a lot of supporting functions that make working with ***ndarray*** very easy.
- Arrays are very frequently used in data science, where speed and resources are very important.

Why is NumPy Faster Than Lists?

- NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
- This behavior is called locality of reference in computer science.
- This is the main reason why NumPy is faster than lists.
- It is optimized to work with latest CPU architectures.

NumPy Getting Started

- Installation

```
C:\Users\Your Name>pip install numpy
```

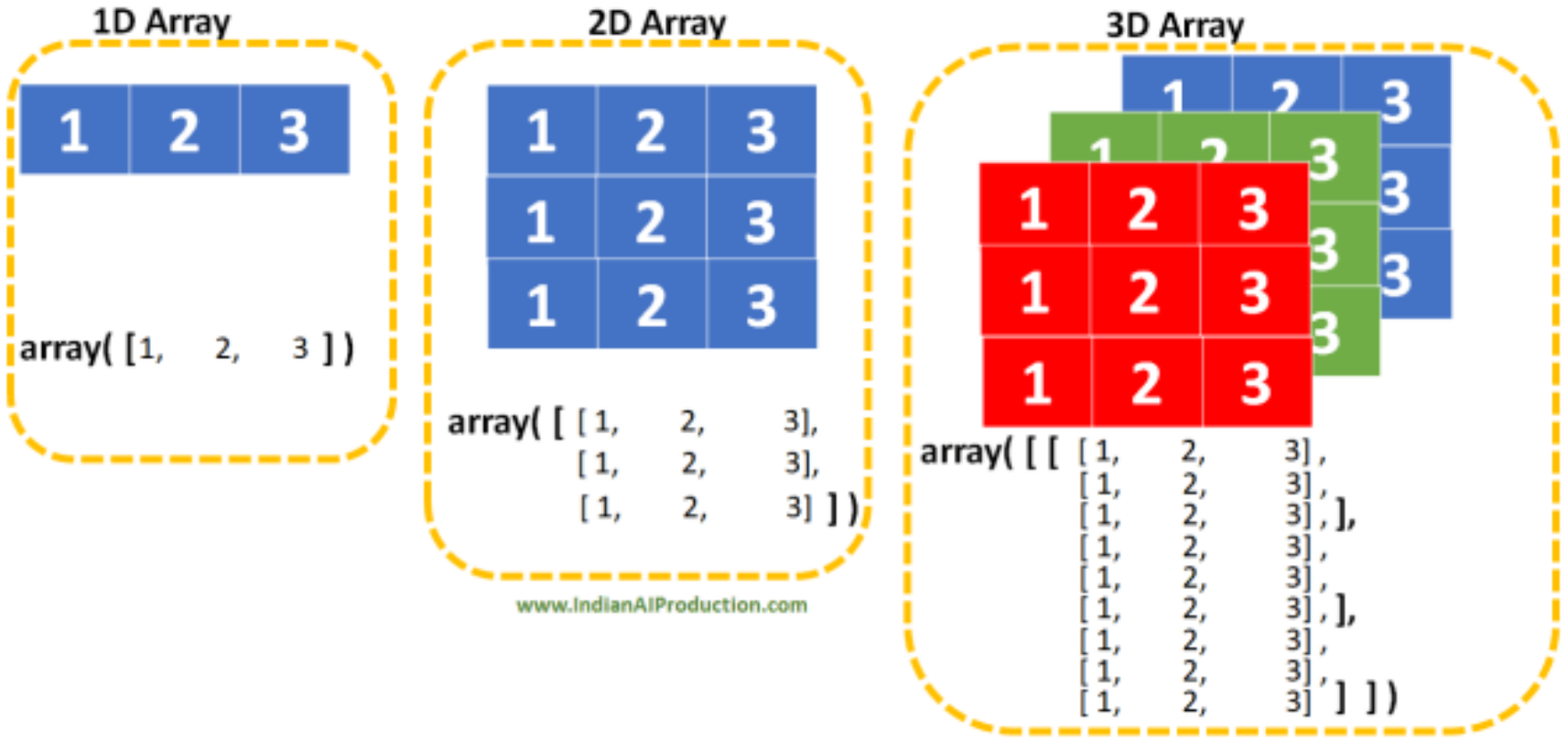
Create a NumPy ***ndarray*** Object

```
import numpy

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

Dimensions in Arrays

- A dimension in arrays is one level of array depth (nested arrays).
  - **nested array:** are arrays that have arrays as their elements.



Dimensions in Arrays

- A dimension in arrays is one level of array depth (nested arrays).
  - **nested array:** are arrays that have arrays as their elements.

0-D Arrays

```
import numpy as np

arr_0d = np.array(42)

print(arr_0d)
```

1-D Arrays

```
import numpy as np

arr_1d = np.array([1, 2, 3, 4, 5])

print(arr_1d )
```

2-D Arrays

```
import numpy as np

arr_2d = np.array([[1, 2, 3], [4, 5, 6]])

print(arr_2d)
```

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

3-D Arrays

```
import numpy as np

arr_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr_3d)
```

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

Check Number of Dimensions?

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

NumPy Array Indexing

- Access Array Elements

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])
print(arr[1])
print(arr[2] + arr[3])

arr2 = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st dim: ', arr2[0, 1])
print('5th element on 2nd dim: ', arr2[1, 4])

arr3 = np.array([[[1, 2, 3], [4, 5, 6]],
                 [[7, 8, 9], [10, 11, 12]]])

print(arr3[0, 1, 2])
```

Example Explained

`arr[0, 1, 2]` prints the value `6`.
And this is why:
The first number represents the first dimension, which contains two arrays:
[[1, 2, 3], [4, 5, 6]]
and:
[[7, 8, 9], [10, 11, 12]]
Since we selected `0`, we are left with the first array:
[[1, 2, 3], [4, 5, 6]]
The second number represents the second dimension, which also contains two arrays:
[1, 2, 3]
and:
[4, 5, 6]
Since we selected `1`, we are left with the second array:
[4, 5, 6]
The third number represents the third dimension, which contains three values:
4
5
6
Since we selected `2`, we end up with the third value:
6

Negative Indexing

- Use negative indexing to access an array from the end.

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

```
Last element from 2nd dim: 10
```

NumPy Array Slicing

- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: *[start:end].*
- We can also define the step, like this: *[start:end:step].*
  - If we don't pass start its considered 0
  - If we don't pass end its considered length of array in that dimension
  - If we don't pass step its considered 1
- **Example:**

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5]) #->1
print(arr[4:])  #->2
print(arr[-3:-1])   #->3
print(arr[1:5:2])   #->4
print(arr[::2])     #->5
```

1. Slice elements from index 1 to index 5 from the following array
2. Slice elements from the beginning to index 4 (not included):
3. Slice from the index 3 from the end to index 1 from the end:
4. Return every other element from index 1 to index 5:
5. Return every other element from the entire array:

Slicing 2-D Arrays

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])  #->1
print(arr[0:2, 2])  #->2
print(arr[0:2, 1:4]) #->3
```

1. From the second element, slice elements from index 1 to index 4 (not included)
2. From both elements, return index 2
3. From both elements, slice index 1 to index 4 (not included), this will return a 2-D array

NumPy Data Types

`strings` - used to represent text data, the text is given under quote marks. e.g. "ABCD"
`integer` - used to represent integer numbers. e.g. -1, -2, -3
`float` - used to represent real numbers. e.g. 1.2, 42.42
`boolean` - used to represent True or False.
`complex` - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

```
import numpy as np

arr1 = np.array([1, 2, 3, 4])
arr2 = np.array(['apple', 'banana', 'cherry'])

print(arr1.dtype)
print(arr2.dtype)

arr3 = np.array([1, 2, 3, 4], dtype='S')

print(arr3)
print(arr3.dtype)
```

NumPy Array Copy vs View

- The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.
- The copy owns the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.
- The view does not own the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)

x2 = arr.view()
arr[0] = 22
print(x2)
```

NumPy Array Shape

- The shape of an array is the number of elements in each dimension.
- NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)
```

```
(2, 4)
```

NumPy Array Reshaping

- Reshaping means changing the shape of an array.
- The shape of an array is the number of elements in each dimension.
- By reshaping we can add or remove dimensions or change number of elements in each dimension.

Reshape From 1-D to 2-D, to 3-D

```
import numpy as np

arr =
np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)
newarr2 = arr.reshape(2, 3, 2)

print(newarr)
print(newarr2)
```

NumPy Joining Array

- Joining means putting contents of two or more arrays in a single array.
- In SQL we join tables based on a key, whereas in NumPy we join arrays by axes.
- We pass a sequence of arrays that we want to join to the *concatenate()* function, along with the axis. If axis is not explicitly passed, it is taken as 0.

```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

Random Permutations of Elements

- A permutation refers to an arrangement of elements. e.g. [3, 2, 1] is a permutation of [1, 2, 3] and vice-versa.
- The NumPy Random module provides two methods for this: shuffle() and permutation().
- Shuffle means changing arrangement of elements in-place. i.e. in the array itself.

```
from numpy import random
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

random.shuffle(arr)

print(arr)
```

The `shuffle()` method makes changes to the original array.

```
from numpy import random
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(random.permutation(arr))
```

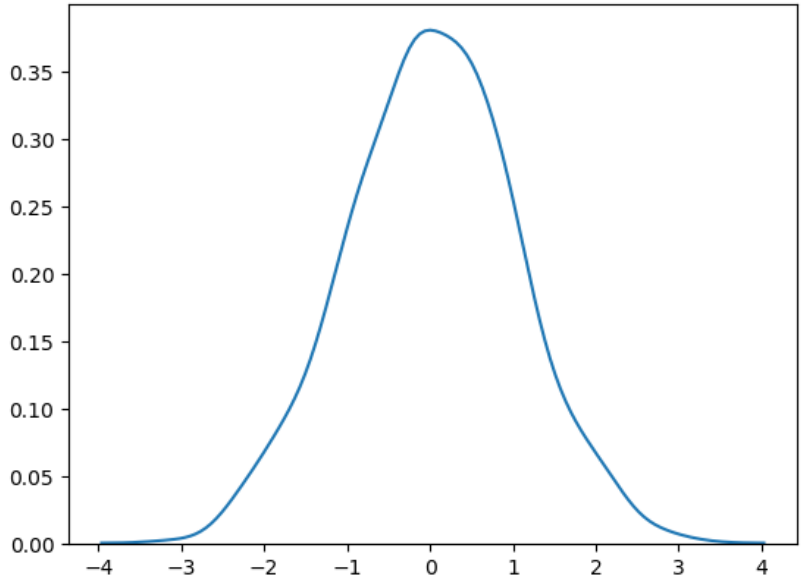
The `permutation()` method *returns* a re-arranged array (and leaves the original array un-changed).

Normal (Gaussian) Distribution

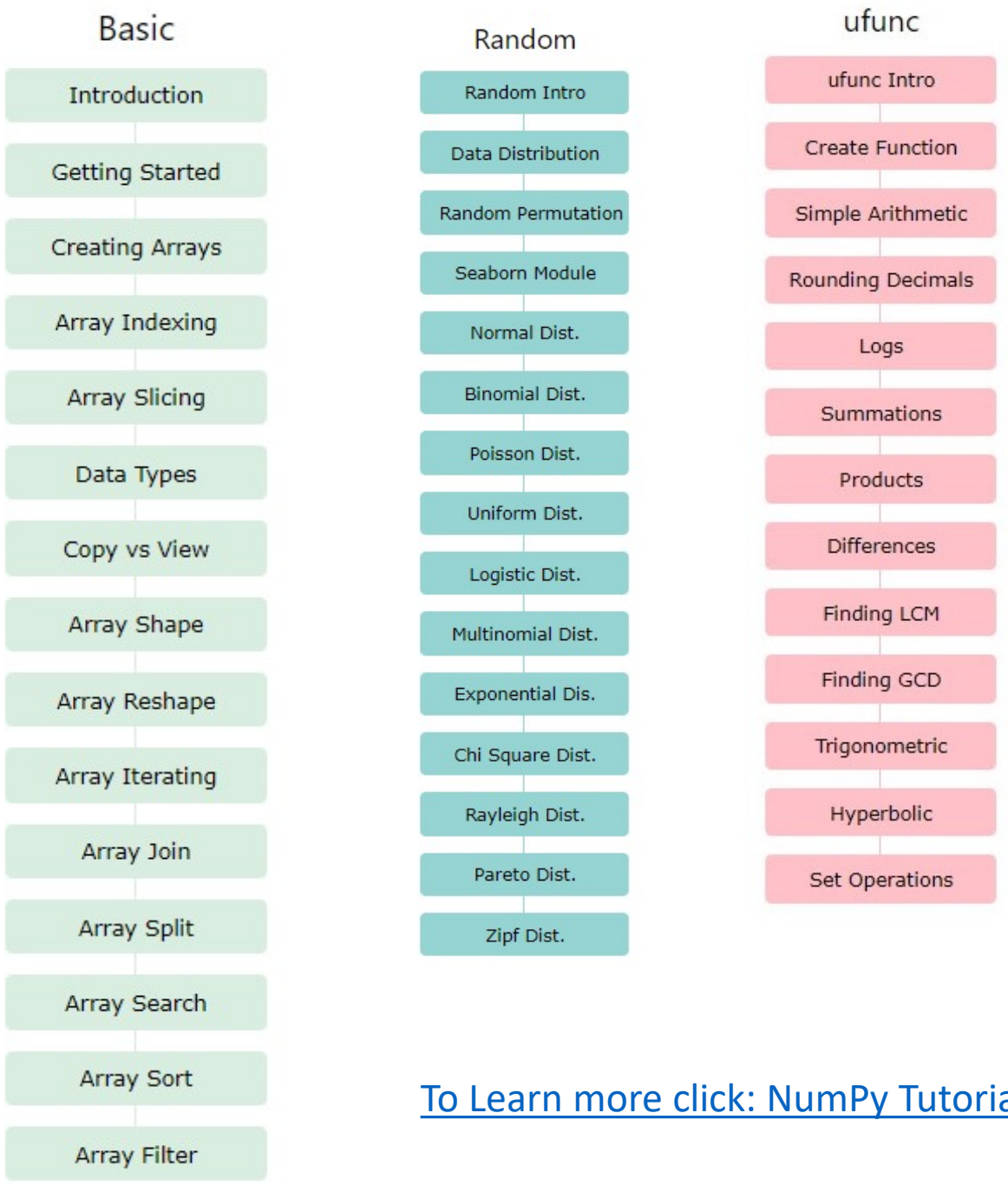
- The Normal Distribution is one of the most important distributions.
- It is also called the Gaussian Distribution after the German mathematician Carl Friedrich Gauss.
- It fits the probability distribution of many events, eg. IQ Scores, Heartbeat etc.
- Use the random.normal() method to get a Normal Data Distribution.
- It has three parameters:
  - loc - (Mean) where the peak of the bell exists.
  - scale - (Standard Deviation) how flat the graph distribution should be.
  - size - The shape of the returned array.

Normal (Gaussian) Distribution

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

x = random.normal(size=(2, 3))

print(x)

x2 = random.normal(loc=1, scale=2, size=(2, 3))

print(x2)

sns.distplot(random.normal(size=1000),
hist=False)

plt.show()
```



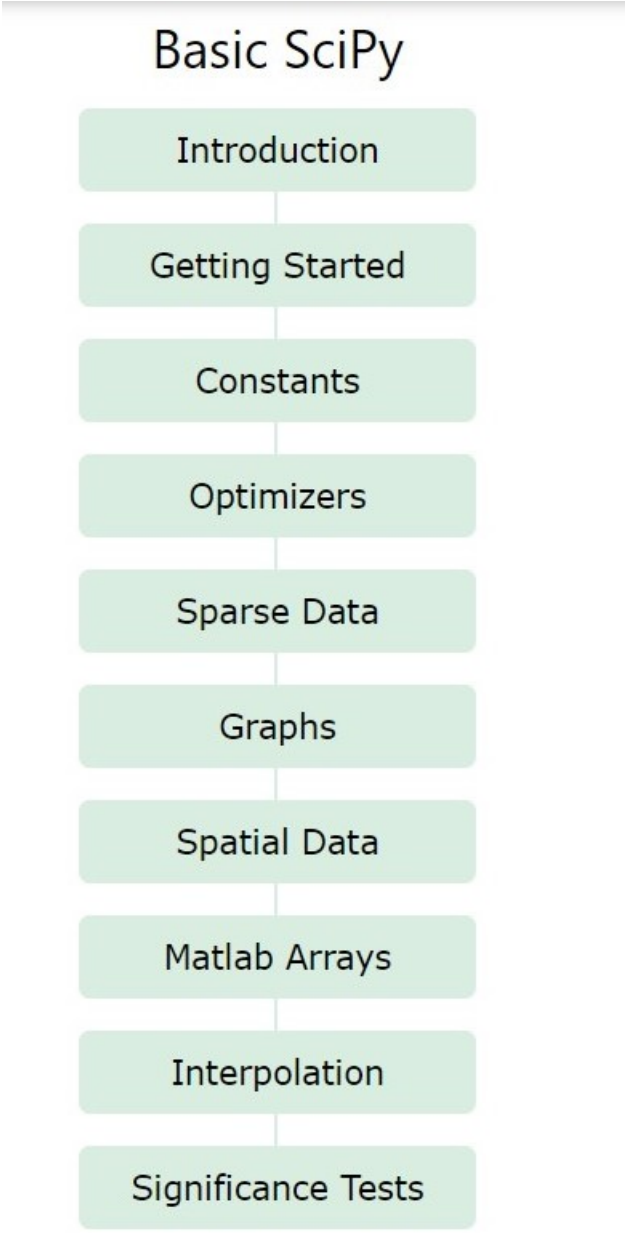
NumPy Features



[To Learn more click: NumPy Tutorial (w3schools.com)]

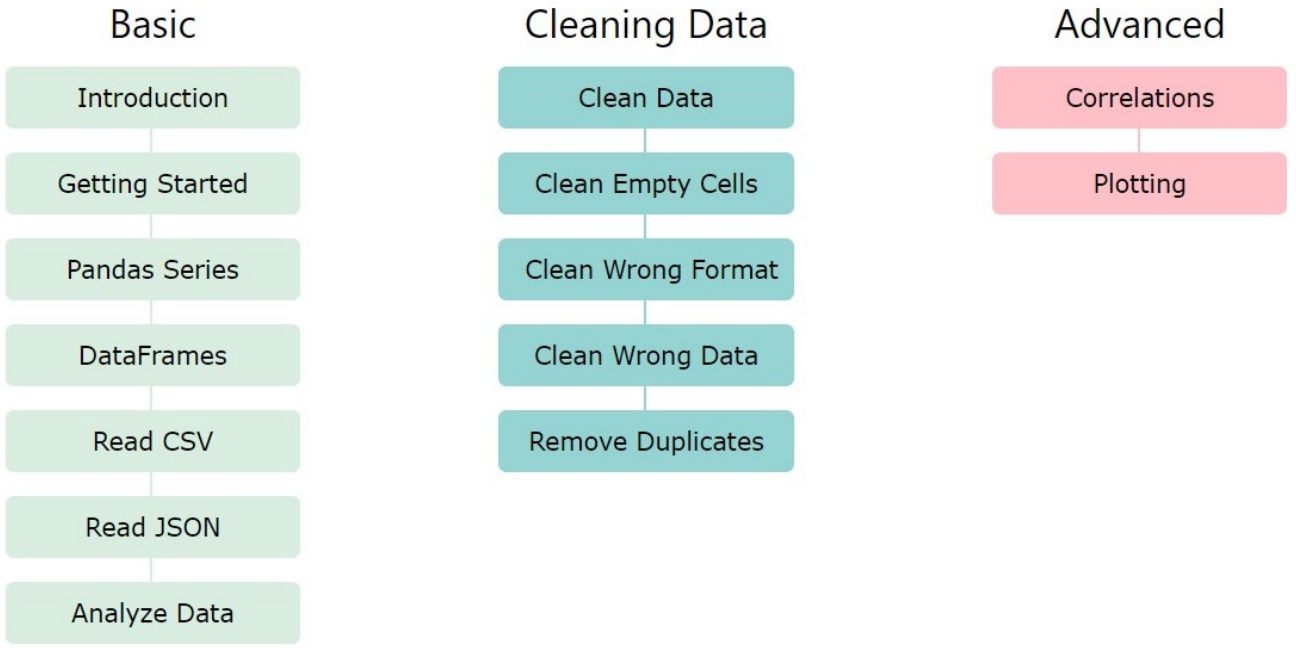
SciPy

- SciPy is a scientific computation library that uses NumPy underneath.
- SciPy stands for Scientific Python.
- It provides more utility functions for optimization, stats and signal processing.
- Like NumPy, SciPy is open source so we can use it freely.
- SciPy was created by NumPy's creator Travis Olliphant.



Pandas

- Pandas is a Python library.
- Pandas is used to analyze data.
- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.



SciPy &Pandas Installations

```
C:\Users\Your Name>pip install scipy
```

```
C:\Users\Your Name>pip install pandas
```

Class Activity - NumPy Learning (10 points)

- Gather all the codes into .py or .ipynb files
- Run the codes and make sure they execute successfully
- Capture the images of codes and running results
- Submit the files and screen captures to E-Learning class assignment

Exercises – 10 Points

- Finish Exercises in w3schools.com
  - Go to https://www.w3schools.com/python/numpy/exercise.asp
  - Copy all the codes and put in JupyterNotebook (.ipynb) or .py