

Python Libraries

Numpy, Scipy & Matplotlib



November 27, 2018

REENU rATHI – A00258770

Submitted To – Brenda Reilly

Table of Contents

[Environment preparation: 1](#_Toc531078672)

[Pycharm installation: 1](#_Toc531078673)

[Issues & Errors Occurred During The Report Preparation: 2](#_Toc531078674)

[Libraries 4](#_Toc531078675)

[Numpy: 4](#_Toc531078676)

[Arrays in Numpy 5](#_Toc531078677)

[Array Indexing: 10](#_Toc531078678)

[Basic operations: 11](#_Toc531078679)

[Sorting of array: 14](#_Toc531078680)

[Scipy: 15](#_Toc531078681)

[File input/output: scipy.io 16](#_Toc531078682)

[Linear algebra operations: scipy.linalg 16](#_Toc531078683)

[Interpolation: scipy.interpolate 18](#_Toc531078684)

[Optimization and fit: scipy.optimize 20](#_Toc531078685)

[Statistics and random numbers: scipy.stats 22](#_Toc531078686)

[Image manipulation: scipy.ndimage 24](#_Toc531078687)

[Matplotlib: 26](#_Toc531078688)

[Basic plots in Matplotlib: 26](#_Toc531078689)

[Line plot: 27](#_Toc531078690)

[Plotting two or more lines on same plot: 28](#_Toc531078691)

[Bar plot: 29](#_Toc531078692)

[Histogram: 31](#_Toc531078693)

[Scatter Plot: 32](#_Toc531078694)

[Pie-chart: 34](#_Toc531078695)

[Plotting curves: 35](#_Toc531078696)

[References: 37](#_Toc531078697)

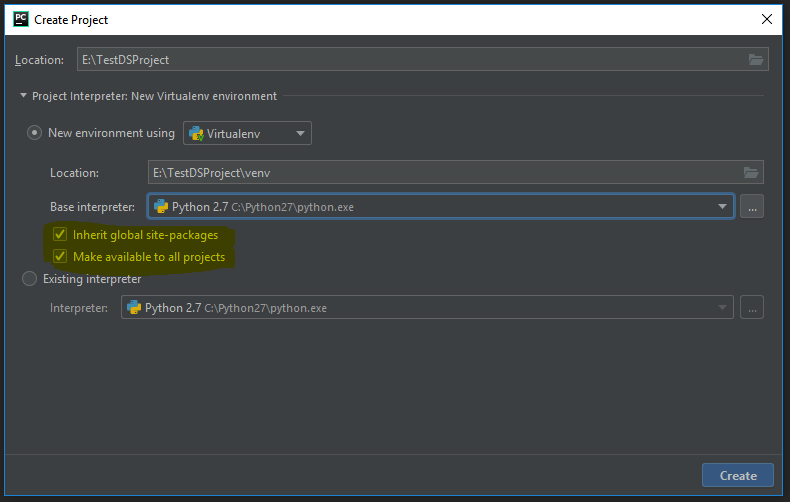
# Environment preparation:

## Pycharm installation:

Download the exe file from below link and follow the instruction on installation screen.

<https://www.jetbrains.com/pycharm/download/#section=windows>

Remember to tick yellow marked selection. It will allow to install eternal libraries as numpy, scipy and matplotlib.



**Adding numpy, scipy and matplotlib libraries to pycharm.**

Create project and follow the instructions given on below link.

<https://www.jetbrains.com/help/pycharm/installing-uninstalling-and-upgrading-packages.html>

## Issues & Errors Occurred During The Report Preparation:

1. Invalid syntax**.**

|  |
| --- |
| E:\NewDSProject\venv\Scripts\python.exe E:/NewDSProject/main.py  File "E:/NewDSProject/main.py", line 2  import numpy\_data-reader  ^  SyntaxError: invalid syntax  Process finished with exit code 1 |

The error came due to name of file numpy\_data-reader.py. After renaming numpy\_data-reader.py to numpy\_data\_reader.py, error resolved.

**Finding:**

We cannot use ‘-’ (hyphen) in any filename or module name.

2: Print complete output on screen by ***np.set\_printoptions(threshold=np.nan)***

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.set_printoptions.html>

3: Deprecated library function of scipy for image read

C:\Python27\python.exe C:/Users/pc/DataScienceProjects/main.py

|  |
| --- |
| Traceback (most recent call last):  File "C:/Users/pc/DataScienceProjects/main.py", line 14, in <module>  main()  File "C:/Users/pc/DataScienceProjects/main.py", line 11, in main  file\_io()  File "C:\Users\pc\DataScienceProjects\scipy\_usage.py", line 20, in file\_io  misc.imread('acron.png')  AttributeError: 'module' object has no attribute 'imread'  Process finished with exit code 1 |

**Solution**:

Either use ‘imageio’ library or matplotlib.pyplot imread function

3: The determinant of a matrix can be calculated if matrix is a square matrix.

|  |
| --- |
| Traceback (most recent call last):  File "C:/Users/pc/DataScienceProjects/main.py", line 15, in <module>  main()  File "C:/Users/pc/DataScienceProjects/main.py", line 12, in main  linear\_algebra\_operation()  File "C:\Users\pc\DataScienceProjects\scipy\_usage.py", line 38, in linear\_algebra\_operation  linalg.det(np.ones((3, 4)))  File "C:\Python27\lib\site-packages\scipy\linalg\basic.py", line 1032, in det  raise ValueError('expected square matrix')  ValueError: expected square matrix |

4: inverse of a singular matrix cannot be calculated as its determinant is always zero.

|  |
| --- |
| Traceback (most recent call last):  File "C:/Users/pc/DataScienceProjects/main.py", line 15, in <module>  main()  File "C:/Users/pc/DataScienceProjects/main.py", line 12, in main  linear\_algebra\_operation()  File "C:\Users\pc\DataScienceProjects\scipy\_usage.py", line 50, in linear\_algebra\_operation  linalg.inv(arr)  File "C:\Python27\lib\site-packages\scipy\linalg\basic.py", line 975, in inv  raise LinAlgError("singular matrix")  numpy.linalg.linalg.LinAlgError: singular matrix |

# Libraries

## Numpy:

Numpy provides high-performance multidimensional array object and various functions to operate over it.

Numpy is an array processing library. It provides multi-dimensional array object, and functionalities and algorithms for operating over these arrays.

Numpy can be used as multi-dimensional container of generic data.

### Arrays in Numpy

NumPy’s major functionality is the multidimensional array.

* It is similar to table of elements, usually numbers, all the elements are of same type. All are indexed using a tuple of positive integers.
* In NumPy, dimensions for data are called *axes*. The number of axes in data are called *rank*.
* NumPy’s array class is **ndarray**. It is also known as **array**.

**Example:**

|  |
| --- |
| [[ 2, 4 ,6, 8],  [ 1, 3, 5, 7]]  Here, rank = 2  as it is 2 dimensions or has 2 axes as row and column  first dimension or axis or row has length = 2, second dimension or axis or column has length = 4  overall, shape of (2, 4) |

**Code:**

|  |
| --- |
| # Python program to demonstrate basic array characteristics  import numpy as np  # Creating array object  arr = np.array( [[2, 4 ,6, 8], [1, 3, 5, 7]] )  # Printing type of arr object  print("Array is of type: ", type(arr))  # Printing count of array dimensions (axes)  print("No. of dimensions: ", arr.ndim)  # Printing shape of array  print("Shape of array: ", arr.shape)  # Printing size (total number of elements) of array  print("Size of array: ", arr.size)  # Printing type of elements in array  print("Array stores elements of type: ", arr.dtype) |

**Output:**

|  |
| --- |
| Array is of type:  No. of dimensions: 2  Shape of array: (2, 4)  Size of array: 8  Array stores elements of type: int64 |

#### **Array creation:**

There are many ways or methods to create arrays in NumPy.

For ex., an array can be created from a Python list or tuple using the array function. The type of the resulting array will be deduced from the type of the elements in the sequences (list or tuple).

The elements or values of an array are often unknown, but size is known. Here, NumPy provides many functions to create arrays with initial values content. This reduces the requirement of growing arrays which is an expensive operation.

**For example**: np.zeros, np.ones, np.full, np.empty, etc.

To create array of numbers, NumPy provides a function equivalent to range that returns arrays instead of lists.

**arange**: returns equally spaced values within the given range. The step size need to be specified.

**linspace**: It returns equally spaced values within a given range. Number of elements in sequence are returned.

**Reshaping array**: The method *reshape* is used to reshape an array. For ex., an array with shape (s1, s2, s3, …, sN). It can be reshaped and convert into another array with shape (ss1, ss2, ss3, …, ssM). The only condition is:

s1 x s2 x s3 … x sN = ss1 x ss2 x ss3 … x ssM . (i.e the original size will remain unchanged.)

**Flatten array**: *flatten* method can be used to get a copy of array converted into one dimension. This function accepts ordered argument. Default value is ‘C’ (for row-major order). Use ‘F’ for column major order.

**Note**: The Type of elements of an array can be explicitly defined at the time of array creation.

|  |
| --- |
| # Python program to demonstrate array creation techniques  import numpy as np  # Creating array from list with type float  a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')  print ("Array created using passed list:\n", a)  # Creating array from tuple  b = np.array((1 , 3, 2))  print ("\nArray created using passed tuple:\n", b)  # Creating a 3X4 array with all zeros  c = np.zeros((3, 4))  print ("\nAn array initialized with all zeros:\n", c)  # Create a constant value array of complex type  d = np.full((3, 3), 6, dtype = 'complex')  print ("\nAn array initialized with all 6s. Array type is complex:\n", d)  # Create an array with random values  e = np.random.random((2, 2))  print ("\nA random array:\n", e)  # Create a sequence of integers  # from 0 to 30 with steps of 5  f = np.arange(0, 30, 5)  print ("\nA sequential array with steps of 5:\n", f)  # Create a sequence of 10 values in range 0 to 5  g = np.linspace(0, 5, 10)  print ("\nA sequential array with 10 values between 0 and 5:\n", g)  # Reshaping 3X4 array to 2X2X3 array  arr = np.array([[1, 2, 3, 4], [5, 2, 4, 2], [1, 2, 0, 1]])  newarr = arr.reshape(2, 2, 3)  print ("\nOriginal array:\n", arr)  print ("Reshaped array:\n", newarr)  # Flatten array  arr = np.array([[1, 2, 3], [4, 5, 6]])  flarr = arr.flatten()  print ("\nOriginal array:\n", arr)  print ("Fattened array:\n", flarr) |

**Output:**

|  |
| --- |
| Array created using passed list:  [ [1. 2. 4.]  [ 5. 8. 7.]]  Array created using passed tuple:  [1 3 2]  An array initialized with all zeros:  [[ 0. 0. 0. 0.]  [ 0. 0. 0. 0.]  [ 0. 0. 0. 0.]]  An array initialized with all 6s. Array type is complex:  [[ 6.+0.j 6.+0.j 6.+0.j]  [ 6.+0.j 6.+0.j 6.+0.j]  [ 6.+0.j 6.+0.j 6.+0.j]]  A random array:  [[ 0.46829566 0.67079389]  [ 0.09079849 0.95410464]]  A sequential array with steps of 5:  [ 0 5 10 15 20 25]  A sequential array with 10 values between 0 and 5:  [ 0. 0.55555556 1.11111111 1.66666667 2.22222222 2.77777778 3.33333333 3.88888889 4.44444444 5. ]  Original array:  [[1 2 3 4]  [5 2 4 2]  [1 2 0 1]]  Reshaped array:  [[[1 2 3]  [4 5 2]]  [[4 2 1]  [2 0 1]]]  Original array:  [[1 2 3]  [4 5 6]]  Fattened array:  [1 2 3 4 5 6] |

### Array Indexing:

The use of Array indexing is very important in analysis and manipulation of the array object. NumPy library has many methods for array indexing.

**Slicing**: Similar to lists in python, NumPy arrays can also be sliced. As arrays can be multidimensional, so it required here to specify a slice for each dimension of the array.

**Integer array indexing**: In this functionality, lists are used for indexing of each dimension. One to one mapping of elements used to construct a new array.

**Boolean array indexing**: In this functionality, we pick elements from an array that satisfy some condition.

**Code:**

|  |
| --- |
| # Python program to demonstrate indexing in numpy  import numpy as np  # An exemplar array  arr = np.array([[-1, 2, 0, 4], [4, -0.5, 6, 0], [2.6, 0, 7, 8], [3, -7, 4, 2.0]])  # Slicing array  temp = arr[:2, ::2]  print ("Array with first 2 rows and alternate columns(0 and 2):\n", temp)  # Integer array indexing example  temp = arr[[0, 1, 2, 3], [3, 2, 1, 0]]  print ("\nElements at indices (0, 3), (1, 2), (2, 1), (3, 0):\n", temp)  # boolean array indexing example  cond = arr > 0 # cond is a boolean array  temp = arr[cond]  print ("\nElements greater than 0:\n", temp) |

**Output:**

|  |
| --- |
| Array with first 2 rows and alternate columns(0 and 2):  [[-1. 0.]  [ 4. 6.]]  Elements at indices (0, 3), (1, 2), (2, 1),(3, 0):  [ 4. 6. 0. 3.]  Elements greater than 0:  [ 2. 4. 4. 6. 2.6 7. 8. 3. 4. 2. ] |

### Basic operations:

**Operation on single array**: Use of overloaded arithmetic operators for element-wise operation on array to create a new array. In case of +=, -=, \*= operators, the existing array has been modified.

**Code:**

|  |
| --- |
| # Python program to demonstrate basic operations on single array  import numpy as np  a = np.array([1, 2, 5, 3])  # add 1 to every element  print ("Adding 1 to every element:", a+1)  # subtract 3 from each element  print ("Subtracting 3 from each element:", a-3)  # multiply each element by 10  print ("Multiplying each element by 10:", a\*10)  # square each element  print ("Squaring each element:", a\*\*2)  # modify existing array  a \*= 2  print ("Doubled each element of original array:", a)  # transpose of array  a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])  print ("\nOriginal array:\n", a)  print ("Transpose of array:\n", a.T) |

**Output:**

|  |
| --- |
| Adding 1 to every element: [2 3 6 4]  Subtracting 3 from each element: [-2 -1 2 0]  Multiplying each element by 10: [10 20 50 30]  Squaring each element: [ 1 4 25 9]  Doubled each element of original array: [ 2 4 10 6]  Original array:  [[1 2 3]  [3 4 5]  [9 6 0]]  Transpose of array:  [[1 3 9]  [2 4 6]  [3 5 0]] |

**Unary operators**: Unary operations are available as member functions of ndarray class as sum, min, max, etc. These functionalities can also be used as row-wise or column-wise using axis parameter.

**Code:**

|  |
| --- |
| # Python program to demonstrate binary operators in Numpy  import numpy as np  a = np.array([[1, 2],  [3, 4]])  b = np.array([[4, 3], [2, 1]])  # add arrays  print ("Array sum:\n", a + b)  # multiply arrays (element wise multiplication)  print ("Array multiplication:\n", a\*b)  # matrix multiplication  print ("Matrix multiplication:\n", a.dot(b)) |

**Output:**

|  |
| --- |
| Array sum:  [[5 5]  [5 5]]  Array multiplication:  [[4 6]  [6 4]]  Matrix multiplication:  [[ 8 5]  [20 13]] |

### Sorting of array:

np.sort method is used for sorting NumPy arrays.

**Code:**

|  |
| --- |
| # Python program to demonstrate sorting in numpy  import numpy as np  a = np.array([[1, 4, 2], [3, 4, 6], [0, -1, 5]])  # sorted array  print ("Array elements in sorted order:\n", np.sort(a, axis = None))  # sort array row-wise  print ("Row-wise sorted array:\n", np.sort(a, axis = 1))  # specify sort algorithm  print ("Column wise sort by applying merge-sort:\n", np.sort(a, axis = 0, kind = 'mergesort'))  # Example to show sorting of structured array  # set alias names for dtypes  dtypes = [('name', 'S10'), ('grad\_year', int), ('cgpa', float)]  # Values to be put in array  values = [(‘aaaaa’, 2009, 8.5), (‘bbbbb’, 2008, 8.7), (‘ccccc’, 2008, 7.9), (‘ddddd’, 2009, 9.0)]    # Creating array  arr = np.array(values, dtype = dtypes)  print ("\nArray sorted by names:\n", np.sort(arr, order = 'name'))    print ("Array sorted by grauation year and then cgpa:\n", np.sort(arr, order = ['grad\_year', 'cgpa'])) |

**Output:**

|  |
| --- |
| Array elements in sorted order:  [-1 0 1 2 3 4 4 5 6]  Row-wise sorted array:  [[ 1 2 4]  [ 3 4 6]  [-1 0 5]]  Column wise sort by applying merge-sort:  [[ 0 -1 2]  [ 1 4 5]  [ 3 4 6]]  Array sorted by names:  [(‘aaaaa’, 2009, 8.5) (‘bbbbb’, 2008, 8.7) (‘ccccc’, 2008, 7.9) (‘ddddd’, 2009, 9.0)]  Array sorted by grauation year and then cgpa:  [(‘ccccc’, 2008, 7.9) ('bbbbb', 2008, 8.7) ('aaaaa', 2009, 8.5) ('ddddd', 2009, 9.0)] |

## Scipy:

The scipy library contains many tool for common issues in scientific computing. It has many modules for different applications, such as optimization, image processing, interpolation, integration, statistics etc.

scipy is the core package for scientific functionalities in Python. It is built to work efficiently on numpy arrays.

**Covered topics:**

1. File input/output: scipy.io
2. Linear algebra operations: scipy.linalg
3. Interpolation: scipy.interpolate
4. Optimization and fit: scipy.optimize
5. Statistics and random numbers: scipy.stats
6. Image manipulation: scipy.ndimage

### File input/output: scipy.io

**Mat files:** Loading and saving mat files

|  |
| --- |
| from scipy import io as spio  a = np.ones(3)  spio.savemat('file.mat', {'a': a})  spio.loadmat('file.mat')['a'] |

**Image files:** Reading images files:

|  |
| --- |
| from scipy import misc  misc.imread('fname.png')  # Matplotlib also has a similar function  import matplotlib.pyplot as plt  plt.imread('fname.png') |

### Linear algebra operations: scipy.linalg

The scipy.linalg module provides standard linear algebra operations.

**The scipy.linalg.det()** function computes the determinant of a square matrix:

**Ex.**

|  |
| --- |
| >>> from scipy import linalg  >>> arr = np.array([[1, 2],[3, 4]])  >>> linalg.det(arr)  -2.0  >>> arr = np.array([[3, 2],[6, 4]])  >>> linalg.det(arr)  0.0  >>> linalg.det(np.ones((3, 4)))  Traceback (most recent call last):  File "<stdin>", line 1, in <module>  File "/usr/lib/python2.7/dist-packages/scipy/linalg/basic.py", line 442, in det  raise ValueError('expected square matrix')  ValueError: expected square matrix  >>> |

The **scipy.linalg.inv()** function computes the inverse of a square matrix:

**Ex.**

|  |
| --- |
| >>> arr = np.array([[1, 2],[3, 4]])  >>> iarr = linalg.inv(arr)  >>> iarr  array([[-2. , 1. ],  [ 1.5, -0.5]])  >>> np.allclose(np.dot(arr, iarr), np.eye(2))  True  >>> |

Now computing the inverse of a singular matrix will raise LinAlgError:

|  |
| --- |
| >>> arr = np.array([[3, 2], [6, 4]])  >>> linalg.inv(arr)  Traceback (most recent call last):  File "<stdin>", line 1, in <module>  File "/usr/lib/python2.7/dist-packages/scipy/linalg/basic.py", line 383, in inv  raise LinAlgError("singular matrix")  numpy.linalg.linalg.LinAlgError: singular matrix  >>> |

The original matrix can be re-composed by matrix multiplication of the outputs of singular-value decomposition with np.dot. Singular-value decomposition is widely used in statistics and signal processing.

**Ex.**

|  |
| --- |
| >>> arr = np.arange(9).reshape((3, 3)) + np.diag([1, 0, 1])  >>> uarr, spec, vharr = linalg.svd(arr)  >>> sarr = np.diag(spec)  >>> svd\_mat = uarr.dot(sarr).dot(vharr)  >>> np.allclose(svd\_mat, arr)  True  >>> |

### Interpolation: scipy.interpolate

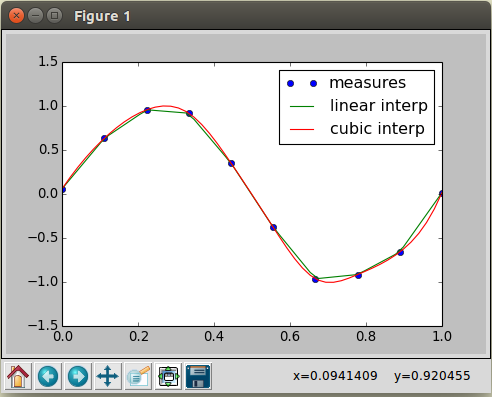
scipy.interpolate is used for deriving a function from data and evaluating points when no measure exists.

**Example.**

As using experimental data close to a sine function, scipy.interpolate.interp1d can build a linear interpolation function. The result, then, can be evaluated at the time of interest and a cubic interpolation can be selected by providing the optional argument.

|  |
| --- |
| >>> measured\_time = np.linspace(0, 1, 10)  >>> noise = (np.random.random(10)\*2 - 1) \* 1e-1  >>> measures = np.sin(2 \* np.pi \* measured\_time) + noise  >>> from scipy.interpolate import interp1d  >>> linear\_interp = interp1d(measured\_time, measures)  >>> interpolation\_time = np.linspace(0, 1, 50)  >>> linear\_results = linear\_interp(interpolation\_time)  >>> cubic\_interp = interp1d(measured\_time, measures, kind='cubic')  >>> cubic\_results = cubic\_interp(interpolation\_time)  >>>  >>> # Plot the data and the interpolation  >>> from matplotlib import pyplot as plt  >>> plt.figure(figsize=(6, 4))  <matplotlib.figure.Figure object at 0x7f0411997450>  >>> plt.plot(measured\_time, measures, 'o', ms=6, label='measures')  [<matplotlib.lines.Line2D object at 0x7f040346d050>]  >>> plt.plot(interpolation\_time, linear\_results, label='linear interp')  [<matplotlib.lines.Line2D object at 0x7f040346d290>]  >>> plt.plot(interpolation\_time, cubic\_results, label='cubic interp')  [<matplotlib.lines.Line2D object at 0x7f040346d790>]  >>> plt.legend()  <matplotlib.legend.Legend object at 0x7f040346d990>  >>> plt.show()  >>> |

**Result:**



### Optimization and fit: scipy.optimize:

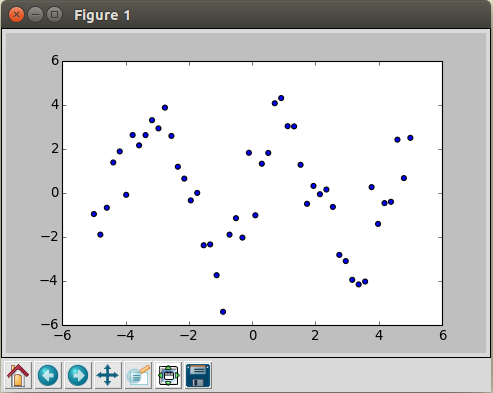
Optimization is the problem of finding a numerical solution to a minimization or equality.

The scipy.optimize module provides algorithms for function minimization, curve fitting and root finding.

**Generate some data**

|  |
| --- |
| import numpy as np  # Seed the random number generator for reproducibility  np.random.seed(0)  x\_data = np.linspace(-5, 5, num=50)  y\_data = 2.9 \* np.sin(1.5 \* x\_data) + np.random.normal(size=50)  # And plot it  import matplotlib.pyplot as plt  plt.figure(figsize=(6, 4))  plt.scatter(x\_data, y\_data)  plt.show() |

**Output:**



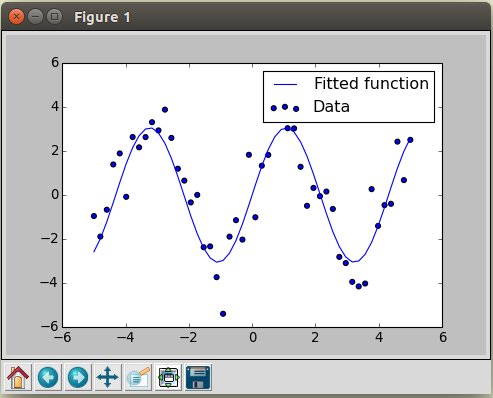
Now fit a simple sine function to the data

|  |
| --- |
| from scipy import optimize  def test\_func(x, a, b):  return a \* np.sin(b \* x)  params, params\_covariance = optimize.curve\_fit(test\_func, x\_data, y\_data, p0=[2, 2]) |

And plot the resulting curve on the data

|  |
| --- |
| plt.figure(figsize=(6, 4))  plt.scatter(x\_data, y\_data, label='Data')  plt.plot(x\_data, test\_func(x\_data, params[0], params[1]),  label='Fitted function')  plt.legend(loc='best')  plt.show() |

**Output:**

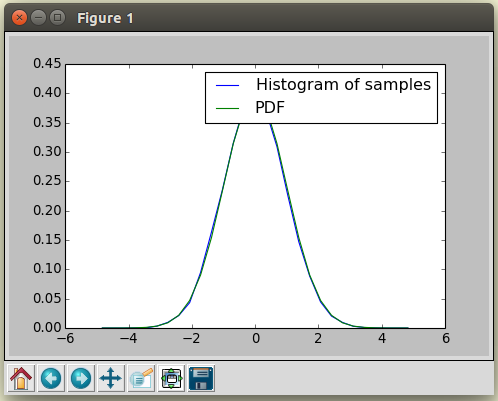


### Statistics and random numbers: scipy.stats

The module scipy.stats contains statistical tools and probabilistic descriptions of random processes.

**Distributions: histogram and probability density function**

|  |
| --- |
| >>> import numpy as np  >>>  >>> # Sample from a normal distribution using numpy's random number generator  ... samples = np.random.normal(size=10000)  >>>  >>> # Compute a histogram of the sample  ... bins = np.linspace(-5, 5, 30)  >>> histogram, bins = np.histogram(samples, bins=bins, normed=True)  >>>  >>> bin\_centers = 0.5\*(bins[1:] + bins[:-1])  >>>  >>> # Compute the PDF on the bin centers from scipy distribution object  ... from scipy import stats  >>> pdf = stats.norm.pdf(bin\_centers)  >>>  >>> from matplotlib import pyplot as plt  >>> plt.figure(figsize=(6, 4))  <matplotlib.figure.Figure object at 0x7f0400967b50>  >>> plt.plot(bin\_centers, histogram, label="Histogram of samples")  [<matplotlib.lines.Line2D object at 0x7f03ff239610>]  >>> plt.plot(bin\_centers, pdf, label="PDF")  [<matplotlib.lines.Line2D object at 0x7f0400967950>]  >>> plt.legend()  <matplotlib.legend.Legend object at 0x7f0403eaa710>  >>> plt.show()  >>> |



### Image manipulation: scipy.ndimage

scipy.ndimage provides functions and operations of n-dimensional arrays as images.

**Geometrical transformations on images**

|  |
| --- |
| >>> # Load some data  >>> from scipy import misc  >>> face = misc.face(gray=True)  >>>  >>> # Apply a variety of transformations  >>> from scipy import ndimage  >>> from matplotlib import pyplot as plt  >>> shifted\_face = ndimage.shift(face, (50, 50))  >>> shifted\_face2 = ndimage.shift(face, (50, 50), mode='nearest')  >>> rotated\_face = ndimage.rotate(face, 30)  >>> cropped\_face = face[50:-50, 50:-50]  >>> zoomed\_face = ndimage.zoom(face, 2)  >>> zoomed\_face.shape  (1536, 2048)  >>>  >>> plt.figure(figsize=(15, 3))  <matplotlib.figure.Figure object at 0x7f03fe7dccd0>  >>> plt.subplot(151)  <matplotlib.axes.AxesSubplot object at 0x7f0403eb3610>  >>> plt.imshow(shifted\_face, cmap=plt.cm.gray)  <matplotlib.image.AxesImage object at 0x7f0400d96ed0>  >>> plt.axis('off')  (-0.5, 1023.5, 767.5, -0.5)  >>>  >>> plt.subplot(152)  <matplotlib.axes.AxesSubplot object at 0x7f0403d4b6d0>  >>> plt.imshow(shifted\_face2, cmap=plt.cm.gray)  <matplotlib.image.AxesImage object at 0x7f03fe72f3d0>  >>> plt.axis('off')  (-0.5, 1023.5, 767.5, -0.5)  >>>  >>> plt.subplot(153)  <matplotlib.axes.AxesSubplot object at 0x7f03fe72f850>  >>> plt.imshow(rotated\_face, cmap=plt.cm.gray)  <matplotlib.image.AxesImage object at 0x7f03fe71d150>  >>> plt.axis('off')  (-0.5, 1270.5, 1176.5, -0.5)  >>>  >>> plt.subplot(154)  <matplotlib.axes.AxesSubplot object at 0x7f03fe71d5d0>  >>> plt.imshow(cropped\_face, cmap=plt.cm.gray)  <matplotlib.image.AxesImage object at 0x7f03fe69d690>  >>> plt.axis('off')  (-0.5, 923.5, 667.5, -0.5)  >>>  >>> plt.subplot(155)  <matplotlib.axes.AxesSubplot object at 0x7f03fe69db10>  >>> plt.imshow(zoomed\_face, cmap=plt.cm.gray)  <matplotlib.image.AxesImage object at 0x7f03fe603990>  >>> plt.axis('off')  (-0.5, 2047.5, 1535.5, -0.5)  >>>  >>> plt.subplots\_adjust(wspace=.05, left=.01, bottom=.01, right=.99, top=.99)  >>>  >>> plt.show()  >>> |

**Output:**



## Matplotlib:

V2.2.3 is LTS (long term support) version.

Matplotlib is a data visualization library built on NumPy arrays in python for 2D plots of arrays. The benefit of visualization is that it allows visual access to huge amounts of data visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

### Basic plots in Matplotlib:

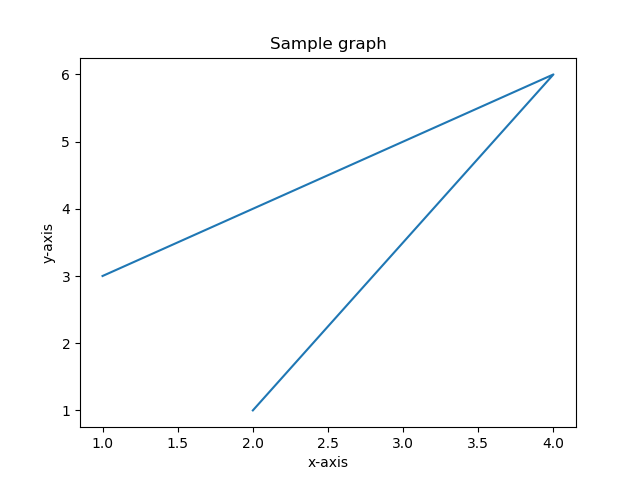
Matplotlib comes with many types of plots. Plots helps to understand trends, patterns, and to make correlations. Some of the sample plots are shown here.

### Line plot:

|  |
| --- |
| # importing the required module  import matplotlib.pyplot as plt  # x axis values  x = [1, 4, 2]  # corresponding y axis values  y = [3, 6, 1]  # plotting the points  plt.plot(x, y)  # naming the x axis  plt.xlabel('x-axis')  # naming the y axis  plt.ylabel('y-axis')  # giving a title to graph  plt.title('Sample graph')  # function to show the plot  plt.show() |

**Code explanation:**

Define the x-axis and corresponding y-axis values as lists. Plot them on canvas using .plot() function. Give a name to x-axis and y-axis using .xlabel() and .ylabel() functions. Give a title to your plot using. title() function. Finally, to view plot, use .show() function.



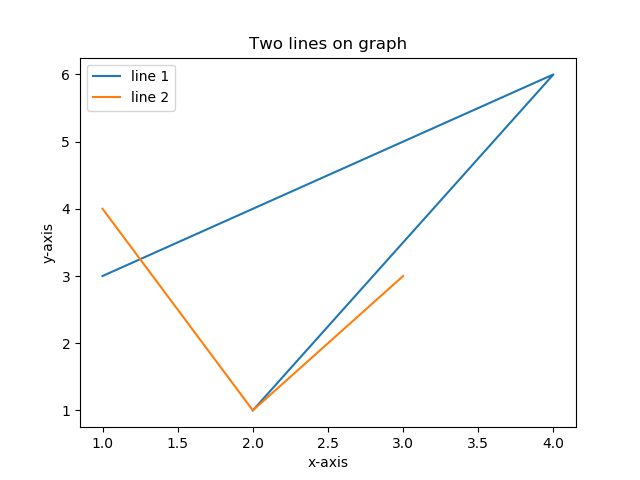
### Plotting two or more lines on same plot:

|  |
| --- |
| # importing the required module  import matplotlib.pyplot as plt  # line 2 points  x2 = [1, 2, 3]  y2 = [4, 1, 3]  # plotting the line 2 points  plt.plot(x2, y2, label="line 2")  # naming the x axis  plt.xlabel('x-axis')  # naming the y axis  plt.ylabel('y-axis')  # giving a title to graph  plt.title('Two lines on graph')  # show a legend on the plot  plt.legend()  # function to show the plot  plt.show() |

**Code explanation:**

Here, differentiate between line by giving them a name(label) which is passed as an argument of .plot() function.

The small rectangular box giving information about type of line and its color is called legend. We can add a legend to our plot using .legend() function.

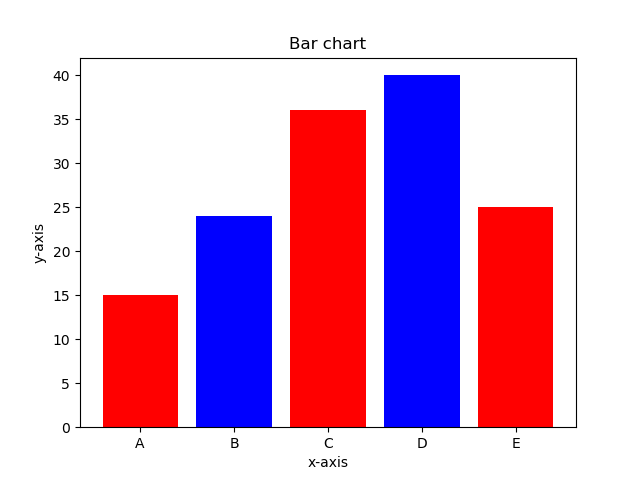


### Bar plot:

|  |
| --- |
| # importing the required module  import matplotlib.pyplot as plt  # x-coordinates of left sides of bars  left = [1, 2, 3, 4, 5]  # heights of bars  height = [15, 24, 36, 40, 25]  # labels for bars  tick\_label = ['A', 'B', 'C', 'D', 'E']  # plotting a bar chart  plt.bar(left, height, tick\_label=tick\_label, width=0.8, color=['red', 'blue'])  # naming the x-axis  plt.xlabel('x-axis')  # naming the y-axis  plt.ylabel('y-axis')  # plot title  plt.title('Bar chart')  # function to show the plot  plt.show() |

**Code explanation:**

Here, used plt.bar() function to plot a bar chart. X-coordinates of left side of bars are passed along with heights of bars.



### Histogram:

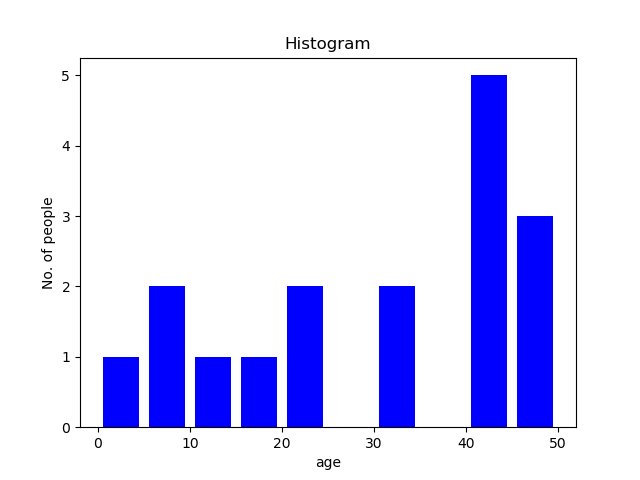
|  |
| --- |
| # importing the required module  import matplotlib.pyplot as plt  # frequencies  ages = [2, 5, 70, 40, 30, 45, 50, 45, 43, 40, 44, 60, 7, 13, 57, 18, 90, 77, 32, 21, 20, 40]  # setting the ranges and no. of intervals  range = (0, 50)  bins = 10  # plotting a histogram  plt.hist(ages, bins, range, color='blue', histtype='bar', rwidth=0.8)  # x-axis label  plt.xlabel('age')  # frequency label  plt.ylabel('No. of people')  # plot title  plt.title('Histogram')  # function to show the plot  plt.show() |

**Code explanation:**

Here, used plt.hist() function to plot a histogram. Frequencies are passed as the ages list.

Range could be set by defining a tuple containing min and max value. These range could be negative to positive.

Next step is to “bin” the range of values—that is, divide the entire range of values into a series of intervals—and then count how many values fall into each interval. Here we have defined bins = 10. So, there are a total of 50/10 = 5 intervals.



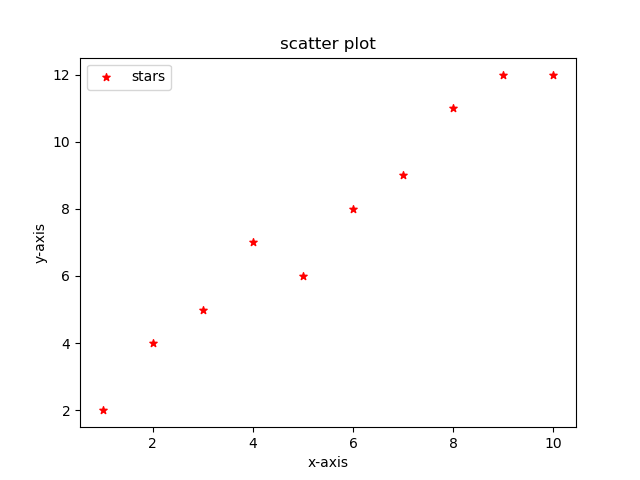
### Scatter Plot:

|  |
| --- |
| # importing the required module  import matplotlib.pyplot as plt  # x-axis values  x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  # y-axis values  y = [2, 4, 5, 7, 6, 8, 9, 11, 12, 12]  # plotting points as a scatter plot  plt.scatter(x, y, label="stars", color="red", marker="\*", s=30)  # x-axis label  plt.xlabel('x-axis')  # frequency label  plt.ylabel('y-axis')  # plot title  plt.title('scatter plot')  # showing legend  plt.legend()  # function to show the plot  plt.show() |

**Code explanation:**

Here, used plt.scatter() function to plot a scatter plot. Like a line, defined x and corresponding y - axis values as well.

Marker argument is used to set the character to use as marker. Its size can be defined using s parameter.



### Pie-chart:

|  |
| --- |
| # importing the required module  import matplotlib.pyplot as plt  # defining labels  vehicles = ['taxi', 'bike', 'cars', 'train']  # portion covered by each label  slices = [3, 7, 8, 6]  # color for each label  colors = ['r', 'y', 'g', 'b']  # plotting the pie chart  plt.pie(slices, labels=activities, colors=colors,  startangle=90, shadow=True, explode=(0, 0, 0.1, 0),  radius=1.2, autopct='%1.1f%%')  # plotting legend  plt.legend()  # showing the plot  plt.show() |

**Code explanation:**

Here, plotted a pie chart by using plt.pie() method.

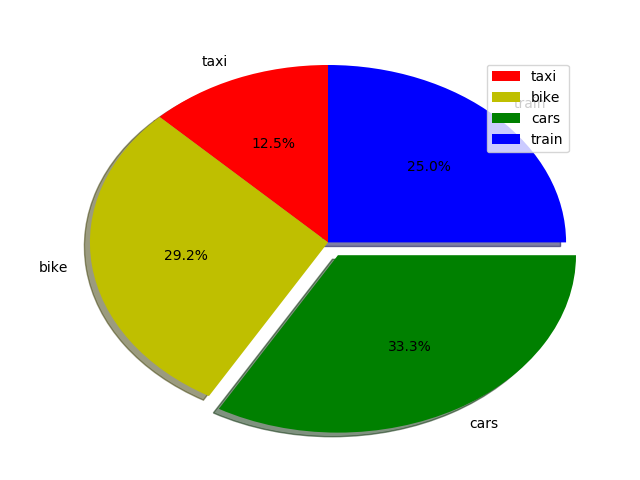
First of all, defined the labels using a list called vehicles. Then, portion of each label can be defined using another list called slices. Color for each label is defined using a list called colors.

shadow = True will show a shadow beneath each label in pie-chart.

startangle rotates the start of the pie chart by given degrees counterclockwise from the x-axis.

explode is used to set the fraction of radius with which we offset each wedge.

autopct is used to format the value of each label. Here, we have set it to show the percentage value only upto 1 decimal place.



### Plotting curves:

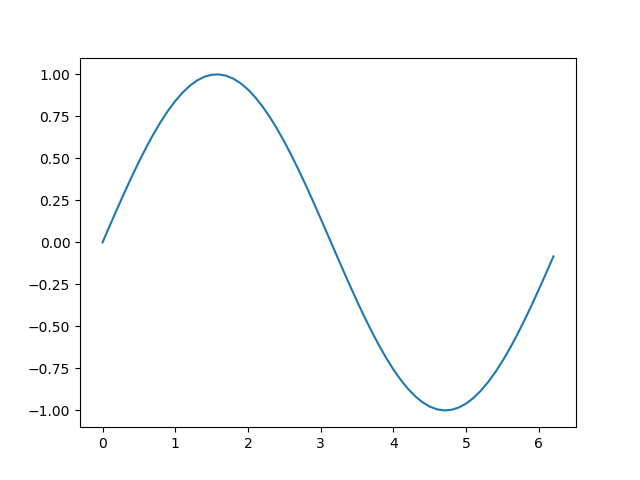
|  |
| --- |
| # importing the required module  import matplotlib.pyplot as plt  import numpy as np  # setting the x - coordinates  x = np.arange(0, 2 \* (np.pi), 0.1)  # setting the corresponding y - coordinates  y = np.sin(x)  # potting the points  plt.plot(x, y)  # function to show the plot  plt.show() |

**Code explanation:**

To set the x - axis values, used np.arange() method in which first two arguments are for range and third one for step-wise increment. The result is a numpy array.

To get corresponding y-axis values, used predefined np.sin() method on the numpy array.

Finally, plot the points by passing x and y arrays to the plt.plot() function.



# References:

* <https://matplotlib.org/2.2.3/index.html>
* <https://matplotlib.org/2.2.3/tutorials/index.html>
* <https://www.geeksforgeeks.org/python-introduction-matplotlib/>
* <http://www.numpy.org/>
* <https://docs.scipy.org/doc/numpy/reference/routines.html>
* <https://docs.scipy.org/doc/numpy/user/quickstart.html>
* <https://docs.scipy.org/doc/numpy/user/>
* <https://www.geeksforgeeks.org/python-numpy/>
* <https://docs.scipy.org/doc/scipy/reference/generated/scipy.misc.face.html#scipy.misc.face>
* <https://www.scipy-lectures.org/intro/scipy.html>
* <https://docs.scipy.org/doc/scipy-1.1.0/reference/tutorial/index.html>
* <https://scipy-cookbook.readthedocs.io/>