Numpy

-np.array.dot

-np.array.ndim

-np.arange

-np.array.dtype

-np.array.astype(np.float32)

-np.array.astype(np.int32)

-np.array.shape

return tuple : if one-dimensional : return one single value within the bracket

-range(len(np.array))

For Loop

- np.ones((row, column), dtype = np.int)

-np.zeros((row,column))

-np.zeros((row,column)) + the specified number

-np.eye(number)

Identity matrix

-np.diag((val_r0c0, val_r1c1, val_r2c2))

The rest of the values are zeros

-np.arange(start_val, stop_val)

stop_val = stop – 1

default start_val = 0

-np.arange(start_val, stop_val, step)

-np.linspace(start_val, real_stop, num = val)

Used to create a particular number of evenly spaced values in a specified half-open interval

```
np.linspace(6., 15., num=10)
```

```
>>array([ 6.,  7.,  8.,  9., 10., 11., 12., 13., 14., 15.])
```

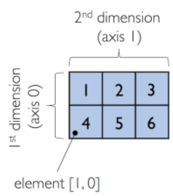
-np.array[start_slice: end_slice] # within the same row or the same column

-np.array[row, loc_at_that_row]

-1 means the last element

0 means the first row

Comma "," is used for slicing the arrays with more than one dimension




```
ary[:, 0] # entire first column

>>array([1, 4])

ary[:, :2] # first two columns

>>array([[1, 2],

       [4, 5]])
```

-np.add(np.array, value)

-np.add.reduce(np.array, axis = 0) [or] np.array.sum(axis = 0)

-np.add.reduce(np.array, axis = 1) [or] np.array.sum(axis = 1)

```
ary = np.array([[1, 2, 3],

                [4, 5, 6]]) # rolling over the 1st axis, axis 0

np.add.reduce(ary, axis=0)

>>array([5, 7, 9])

np.add.reduce(ary, axis=1) # row sums

>>array([ 6, 15])
```

-np.array.sum()

```
ary.sum()
```

```
>>21
```

-np.power(np.array, value)

-np.sqrt(np.array)

- np.mean (computes arithmetic mean or average)
- np.std (computes the standard deviation)
- np.var (computes variance)
- np.sort (sorts an array)
- np.argsort (returns indices that would sort an array)
- np.min (returns the minimum value of an array)
- np.max (returns the maximum value of an array)
- np.argmin (returns the index of the minimum value)
- np.argmax (returns the index of the maximum value)
- np.array_equal (checks if two arrays have the same shape and elements)

-np.array.copy() #changes in a copied array does not affect the original array
-np.array > value #return an array including True and False

In:

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

mask = ary > 2

mask
```

Out:

```
array([False, False,  True,  True,  True])
```

In:

```
ary[mask]
```

Out:

```
array([3, 4, 5])
```

In:

```
mask.sum()
```

Out:

```
3
```

return the total number of True values

- np.random.seed (define the range of the seed for randomization)
- np.random.rand (define the number of value from the range)
- np.random.RandomState(seed = range)
- np.random.RandomState.rand(number)
- np.random.default_rng(seed = range)
- np.random.default_rng.random(number)
- np.array.reshape(row, column) # reshaping arrays
- np.may_share_memory(np.array.reshape(row, column), np.array) # return true
- np.array.reshape(-1, column) # -1 for unspecified axis, NyPy will create a row in accordance with the column you have defined
- np.array.reshape(-1) # flattening, this will give you one dimensional array
- np.array.flatten() # this will create a copy of the array
- np.array.ravel() # this will create a view of the reshape
- np.concatenate((np.array, np.array)) # to combine two or more array objects
- np.concatenate((np.array, np.array), axis = 0) # stack along the first axis (here: rows); stacked vertically
- np.concatenate((np.array, np.array), axis = 1) # stack along the column; stack horizontally
- np.where(np.array > the numbers you want to compare) # return 0 if false, 1 if true

In:

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

np.where(ary > 2, 1, 0)
```

Out:

```
array([0, 0, 1, 1, 1])
```

Or we can apply the following ways instead of using np.where()

In:

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

mask = ary > 2

ary[mask] = 1

ary[~mask] = 0

ary
```

Out:

```
array([0, 0, 1, 1, 1])
```

A few examples related to bitwise operators

In:

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


ary[(ary > 3) | (ary < 2)]
```

Out:

```
array([1, 4, 5])
```

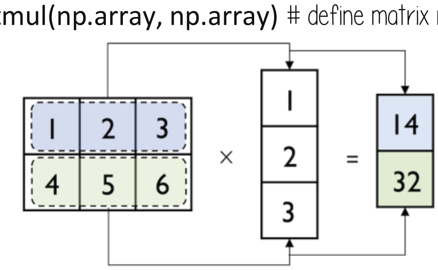
In:

```
ary[~((ary > 3) | (ary < 2))]
```

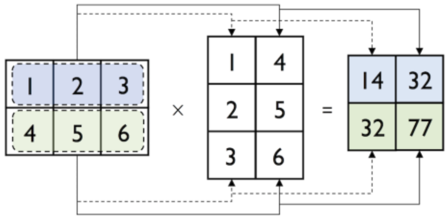
Out:

```
array([2, 3])
```

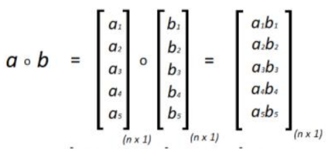
- np.matmul(np.array, np.array) # define matrix multiplication, dimensions problems are solved by broadcasting



- np.array.transpose() # swap the rows and the columns
- np.array.T # swap the rows and the columns; same as transpose()
- np.dot(np.array, np.array) # do matrix multiplication



- np.multiply(np.array, np.array)



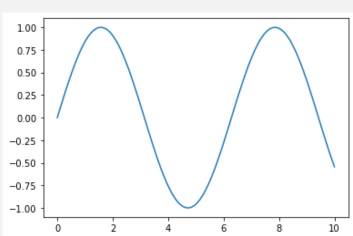
Element wise Product

MATPLOT

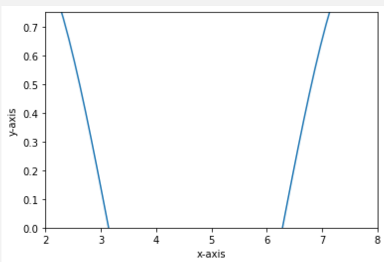
```
%matplotlib inline
import matplotlib.pyplot as plt
```

Plotting Functions and Lines

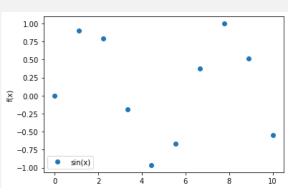
```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))


plt.show()
>>
```



```
x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))

plt.xlim([2, 8])

plt.ylim([0, 0.75])

plt.xlabel('x-axis')

plt.ylabel('y-axis')

plt.show()

>>
```




```
x = np.linspace(0, 10, 10)

plt.plot(x, np.sin(x), label=('sin(x)'), linestyle='', marker='o')

plt.ylabel('f(x)')

plt.xlabel('x')

plt.legend(loc='lower left')

plt.show()

>>
```

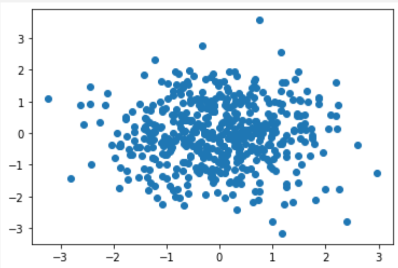


A legend is an area describing the elements of the graph. In the matplotlib library, there's a function called legend() which is used to Place a legend on the axes. The attribute Loc in legend() is used to specify the location of the legend.

plt.linestyle= 'solid' or 'dotted' or 'dashed' or 'dashdot' or ' '

Scatter Plots

```
rng = np.random.RandomState(123)

x = rng.normal(size=500)  #normal Gaussian distribution

y = rng.normal(size=500)

plt.scatter(x, y)

plt.show()

>>
```



$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

numpy.random.normal(*loc=0.0*, *scale=1.0*, *size=None***)**

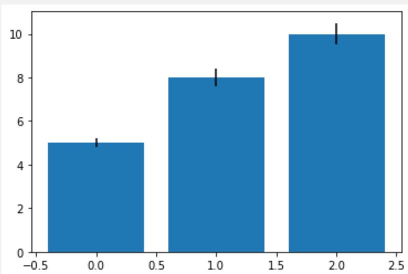
Parameters

loc*float or array_like of floats*
Mean ("centre") of the distribution.
scale*float or array_like of floats*
Standard deviation (spread or "width") of the distribution. Must be non-negative.
size*int or tuple of ints, optional*
Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. If size is `None` (default), a single value is returned if `loc` and `scale` are both scalars. Otherwise, `np.broadcast(loc, scale).size` samples are drawn.

Bar Plots

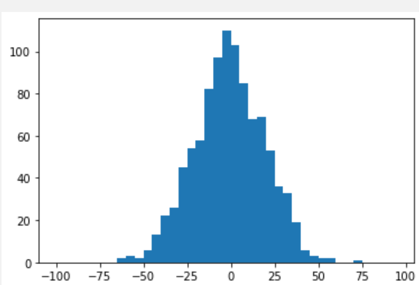
```
# input data

means = [5, 8, 10]

stddevs = [0.2, 0.4, 0.5]

bar_labels = ['bar 1', 'bar 2', 'bar 3']
```

```
# plot bars
x_pos = list(range(len(bar_labels)))
plt.bar(x_pos, means, yerr=stddevs)
plt.show()
>>
```

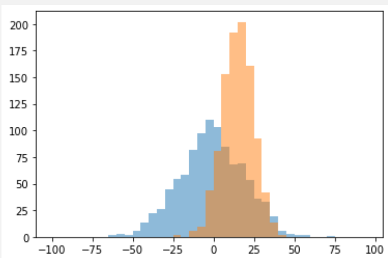


Histograms

```
rng = np.random.RandomState(123)
x = rng.normal(0, 20, 1000)
# fixed bin size
bins = np.arange(-100, 100, 5) # fixed bin size
plt.hist(x, bins=bins)
plt.show()
>>
```



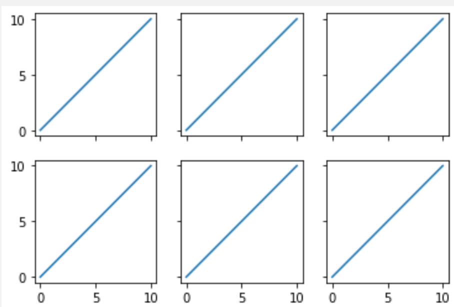
```
rng = np.random.RandomState(123)

x1 = rng.normal(0, 20, 1000)

x2 = rng.normal(15, 10, 1000)

# fixed bin size

bins = np.arange(-100, 100, 5) # fixed bin size

plt.hist(x1, bins=bins, alpha=0.5)

plt.hist(x2, bins=bins, alpha=0.5)

plt.show()

>>
```



pyplot. hist(x, alpha=n) with x as a data set and n as an integer between 0 and 1 specifying the transparency of each histogram. A lower value of n results in a more transparent histogram.

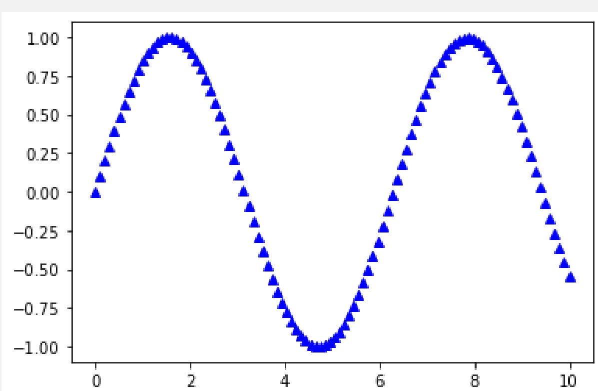
Subplots

```
x = range(11)

y = range(11)

fig, ax = plt.subplots(nrows=2, ncols=3,sharex=True, sharey=True)

for row in ax:

    for col in row:

        col.plot(x, y)

plt.show()

>>
```



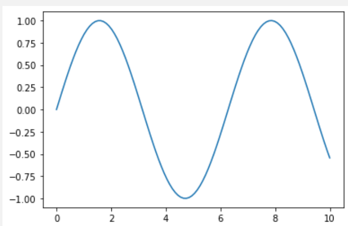
Colors and Makers

```
x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x),color='blue', marker='^',linestyle='')

plt.show()

>>
```



Saving Plots

```
x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))

plt.savefig('myplot.png', dpi=300)

plt.savefig('myplot.pdf')

plt.show()

>>
```



savefig(filename, dpi=None) to save a matplotlib. pyplot figure as an image named filename with a resolution of dpi in dots per inch. A high dpi value such as 200 or more indicates a high resolution image, while a low dpi value such as 10 indicates a lower resolution image.

Resources:

NumPy and Matplotlib reference material:

- The official NumPy documentation
- The official Matplotlib Gallery
- The official Matplotlib Tutorials

NumPy books, tutorials, and papers:

- Rougier, N.P., 2016. From Python to NumPy.
- Oliphant, T.E., 2015. A Guide to NumPy: 2nd Edition. USA: Travis Oliphant, independent publishing.
- Varoquaux, G., Gouillart, E., Vahtras, O., Haenel, V., Rougier, N.P., Gommers, R., Pedregosa, F., Jędrzejewski-Szmek, Z., Virtanen, P., Combelles, C. and Pinte, D., 2015. SciPy Lecture Notes.
- Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array Programming with NumPy. Nature 585, 357–362 (2020).