Some typos may exist in this document. If you find any typos, please let me know.

1. Array

```
import numpy as np
import pandas as pd

print(np.array([1,2,3,4]))
print(np.array(pd.Series([1,2,3,4])))
```

```
[1 2 3 4]
[1 2 3 4]
```

2. Arange

It creates an array in a range with a specified increment.

```
np.arange(2, 10, 2)
```

```
array([2, 4, 6, 8])
```

```
np.arange(0, 20, 5)
```

```
array([ 0,  5, 10, 15])
```

The first two arguments are lower and upper bounds (upper is exclusive). The third argument is the step size.

3. Linspace

It creates an array in a specified range with equidistant elements.

```
np.linspace(0, 20, 10)
```

```
array([ 0.        ,  2.22222222,  4.44444444,  6.66666667,  8.88888889,
       11.11111111, 13.33333333, 15.55555556, 17.77777778, 20.        ])
```

```
np.linspace(0, 1, 5)
```

```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

The first two arguments determine the lower and upper bounds. Unlike the arange function, upper bound is inclusive. The third arguments specify how many equidistant elements we want in that range.

4. Unique

It returns the number of unique elements in an array. We can also see how many times each element occur in the array using the return_counts parameter.

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

```
np.unique(a)
```

```
array([1, 2, 3])
```

```
np.unique(a, return_counts=True)
```

```
(array([1, 2, 3]), array([3, 2, 1]))
```

5. Argmax and argmin

They return the indices of maximum and minimum values along an axis.

```
A = np.random.randint(10, size=(4,3))
A
```

```
array([[9, 6, 2],
       [9, 5, 1],
       [3, 7, 8],
       [2, 3, 6]])
```

Argmax with axis=1 will return the indices of the maximum values in each row. Argmin with axis=0 will return the indices of the minimum values in each column.

```
np.argmax(A, axis=1)
```

```
array([0, 0, 2, 2])
```

```
np.argmin(A, axis=0)
```

```
array([3, 3, 1])
```

6. Random.random

It creates an array with random floats between 0 and 1. Same operation can be done with the random_sample function as well.

```
np.random.random(size=4)
```

```
array([0.31011635, 0.90369178, 0.38834297, 0.79433153])
```

```
np.random.random_sample(size=4)
```

```
array([0.42461156, 0.32158476, 0.29690321, 0.44240139])
```

7. Random.randint

It creates an array of integers in any shape.

```
np.random.randint(0,5, size=5)
```

```
array([1, 1, 1, 3, 0])
```

```
np.random.randint(10, size=(3,4))
```

```
array([[9, 6, 8, 6],
       [3, 2, 4, 7],
       [8, 6, 9, 4]])
```

The first two arguments determine the bounds. If only one bound is passed, it is considered as the upper bound and the lower bound is taken as 0.

8. Random.randn

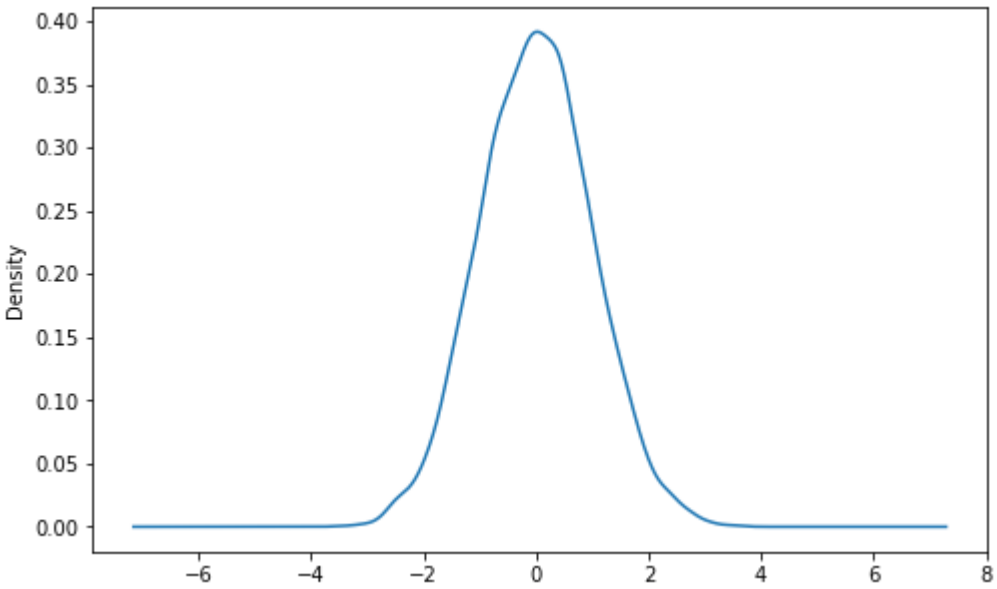
It returns a sample or samples from the standard normal distribution (i.e. Zero mean and unit variance).

```
a = np.random.randn(10000)
print(a.mean())
print(a.var())
```

```
0.0010318116218097015
0.9958376127146814
```

Let's also plot the values to observe the standard normal distribution.

```
pd.Series(a).plot(kind='kde', figsize=(8,5))
```



9. Random.shuffle

It modifies the sequence of an array by shuffling its elements.

```
a = np.array([0,1,2,3,4,5,6,7,8,9])
np.random.shuffle(a)
a
```

```
array([4, 8, 0, 5, 1, 6, 3, 2, 7, 9])
```

10. Reshape

As the name suggests, it changes the shape of an array. The overall size must be maintained. For instance, an array with a shape of 3x4 can be converted to an array of shape 2x6 (size is 12).

```
A = np.random.randint(10, size=(3,4))
A
```

```
array([[7, 2, 8, 0],
       [2, 5, 8, 5],
       [6, 3, 9, 5]])
```

```
A.reshape(2, 6)
```

```
array([[7, 2, 8, 0, 2, 5],
       [8, 5, 6, 3, 9, 5]])
```

You can also specify the size in one dimension and pass -1 for the other dimension. Numpy will infer the shape.

```
A.reshape(-1, 4)
```

```
array([[7, 2, 8, 0],
       [2, 5, 8, 5],
       [6, 3, 9, 5]])
```

Reshape is also used to increase the dimension of an array which is a common practice when working machine learning or deep learning models.

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

print(f'The array a has {a.ndim} dimension')
print(a.reshape(-1,1))
print(f'The reshaped array a has {a.reshape(-1,1).ndim} dimensions')
```

```
The array a has 1 dimension
[[1]
 [2]
 [3]
 [4]]
The reshaped array a has 2 dimensions
```

11. Expand_dims

It expands the dimension of an array.

```
a = np.array([1,2,3,4])
a.ndim
```

```
1
```

```
np.expand_dims(a, axis=0).ndim
```

```
2
```

The axis parameter allows to choose through which axis the expansion is done. Expand_dims with axis=1 is equivalent to reshape(-1,1).

```
np.expand_dims(a, axis=1)
```

```
array([[1],
       [2],
       [3],
       [4]])
```

```
np.expand_dims(a, axis=0)
```

```
array([[1, 2, 3, 4]])
```

12. Count_nonzero

It returns the count of non-zero elements in an array which may come in handy when working with arrays with high sparsity.

```
a = np.random.randint(0,5, size=100)
np.count_nonzero(a)
```

```
74
```

13. Argwhere

It returns the indices of nonzero elements in an array.

```
A = np.random.randint(3, size=(2,3))
A
```

```
array([[1, 0, 1],
       [2, 1, 0]])
```

```
np.argwhere(A)
```

```
array([[0, 0],
       [0, 2],
       [1, 0],
       [1, 1]])
```

For instance, the second column in the first row is zero so its index ([0,1]) is not returned by the argwhere function.

14. Zeros, Ones, Full

These are actually three separate functions but what they do is very similar. They create arrays with zeros, ones, or a specific value.

```
np.zeros(3)
```

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

```
np.ones((3,3), dtype='int32')
```

```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]], dtype=int32)
```

```
np.full((2,3), fill_value=4.2)
```

```
array([[4.2, 4.2, 4.2],
       [4.2, 4.2, 4.2]])
```

The default data type is float but can be changed to integers using the type parameter.

15. Eye and Identity

Both eye and identity create identity matrix with a specified dimension. **Identity matrix,** denoted as **I,** is a square matrix that have 1's on the diagonal and 0's at all other positions.

What makes an identity matrix special is that it does not change a matrix when multiplied. In this sense, it is similar to number 1 in real numbers.

```
np.eye(4)
```

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

```
np.identity(3)
```

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

The **inverse** of a matrix is the matrix that gives the identity matrix when multiplied with the original matrix. Not every matrix has an inverse. If matrix A has an inverse, then it is called **invertible** or **non-singular.**

16. Ravel

Ravel returns a flattened array. If you are familiar with convolutional neural netwoks (CNN), pooled feature maps are flattened before feeding to fully connected layer.

```
A = np.random.randint(5, size=(2,3))
A
```

```
array([[4, 2, 3],
       [4, 0, 4]])
```

```
np.ravel(A)
```

```
array([4, 2, 3, 4, 0, 4])
```

Second row is concatenated at the end of first row. Ravel function also allows column-wise concatenation using **order** parameter.

```
np.ravel(A, order='F')
```

```
array([4, 4, 2, 0, 3, 4])
```

17. Hsplit and Vsplit

They split arrays vertically (vsplit) or horizontally (hsplit).

```
A = np.random.randint(5, size=(4,4))
A
```

```
array([[3, 3, 1, 1],
       [3, 1, 0, 4],
       [2, 4, 1, 0],
       [4, 3, 0, 4]])
```

A is an array with a shape of 4x4. Splitting A horizontally will result in two arrays with 4x2.

```
np.hsplit(A, 2)[0]
```

```
array([[3, 3],
       [3, 1],
       [2, 4],
       [4, 3]])
```

```
np.hsplit(A, 2)[1]
```

```
array([[1, 1],
       [0, 4],
       [1, 0],
       [0, 4]])
```

If A is vertically split, the resulting arrays will have a shape of 2x4.

```
np.vsplit(A, 2)[0]
```

```
array([[3, 3, 1, 1],
       [3, 1, 0, 4]])
```

```
np.vsplit(A, 2)[1]
```

```
array([[2, 4, 1, 0],
       [4, 3, 0, 4]])
```

18. Hstack and Vstack

They stack arrays horizontally (column-wise) and vertically (rows on top of each other)

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

```
np.vstack((a,b))
```

```
array([[1, 1, 1, 2],
       [2, 3, 4, 5]])
```

```
np.hstack((a,b))
```

```
array([1, 1, 1, 2, 2, 3, 4, 5])
```

19. Transpose

It transposes an array. In case of 2-dimensional arrays (i.e. matrix), transposing means switching rows and columns.

```
A = np.random.randint(5, size=(3,4))
A
```

```
array([[3, 3, 3, 0],
       [1, 1, 3, 0],
       [4, 0, 3, 2]])
```

```
A.transpose()
```

```
array([[3, 1, 4],
       [3, 1, 0],
       [3, 3, 3],
       [0, 0, 2]])
```

20. Abs and Absolute

Both abs and absolute return the absolute values of elements in an array.

```
A = np.random.randint(-5,5, size=(2,3))
A
```

```
array([[ 2, -1,  4],
       [-3,  1,  0]])
```

```
np.abs(A)
```

```
array([[2, 1, 4],
       [3, 1, 0]])
```

21. Round and Around

Both of them round up the floats to a specified number of decimal points.

```
a = np.random.random(5)
a
```

```
array([0.24445311, 0.78788622, 0.90558656, 0.84804425, 0.31584555])
```

```
np.around(a, 2)
```

```
array([0.24, 0.79, 0.91, 0.85, 0.32])
```

```
np.round(a, 2)
```

```
array([0.24, 0.79, 0.91, 0.85, 0.32])
```

We have covered only a part of the operations that we can do with NumPy. However, these are the operations that you are likely to use in a typical data analysis and manipulation process.

Thank you for reading. Please let me know if you have any feedback.