untitled-1

May 31, 2023

1 1-D Array

[13]:
```
import numpy as np
a=np.array([2,2,2])
print(a)

print("size of array is",a.size)

print("daimention of array is",(a.ndim))
```

```
[2 2 2]
size of array is 3
daimention of array is 1
```

[14]:
```
import numpy as np
a=np.zeros(2)
print(a)

print("daimention of array is",(a.ndim))
```

```
[0. 0.]
daimention of array is 1
```

[15]:
```
import numpy as np
b=np.ones(2)
print(b)

print("daimention of array is",(b.ndim))
```

```
[1. 1.]
daimention of array is 1
```

[16]:
```
import numpy as np
c=np.empty(2)
print(c)

print("daimention of array is",(c.ndim))
```

```
[1. 1.]
daimention of array is 1
```

[17]:
```
import numpy as np
food=np.array(["biryani", "korma","palao","chaae"])
print(food)

price=np.array([100,250,150,60])
print(price)

print(type(food))
print(type(price))

print("daimention of array is",(food.ndim))
print("daimention of array is",(price.ndim))
```

```
['biryani' 'korma' 'palao' 'chaae']
[100 250 150  60]
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
daimention of array is 1
daimention of array is 1
```

[18]:
```
print(len(food))
print(len(price))
```

```
4
4
```

[19]:
```
print(food[3])
price[3]
```

```
chaae
```

[19]:
```
60
```

[20]:
```
print(price[0:2])
print(food[0:4])
```

```
[100 250]
['biryani' 'korma' 'palao' 'chaae']
```

[21]:
```
print(price.mean())
print(price.max())
```

```
140.0
250
```

[22]:
```
Range=np.arange(10)
print(Range)

Range=np.arange(0,12)
print(Range)


Range=np.arange(5,51,5)
print(Range)

print("daimention of array is",(Range.ndim))
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[ 5 10 15 20 25 30 35 40 45 50]
daimention of array is 1
```

[23]:
```
LineSpace=np.linspace(1,100,15)
print(LineSpace)

print("  ")

LineSpace=np.linspace(1,100,3)
print(LineSpace)

print("daimention of array is",(LineSpace.ndim))
```

```
[  1.           8.07142857  15.14285714  22.21428571  29.28571429
  36.35714286  43.42857143  50.5         57.57142857  64.64285714
  71.71428571  78.78571429  85.85714286  92.92857143 100.        ]

[  1.   50.5 100. ]
daimention of array is 1
```

2 Array Function

[24]:
```
list1=np.array([20,18,16,14,2,4,6,8,10,12.2])
print(list1)

print("  ")
list1.sort()

print(list1)

print("daimention of array is",(list1.ndim))
```

```
[20.  18.  16.  14.   2.   4.   6.   8.  10.  12.2]
```

```
[ 2.   4.   6.   8.  10.  12.2 14.  16.  18.  20. ]
daimention of array is 1
```

[25]:
```
list2=np.array([10,20,30,40,50])

np.concatenate((list1,list2))
```

[25]:
```
array([ 2. ,  4. ,  6. ,  8. , 10. , 12.2, 14. , 16. , 18. , 20. , 10. ,
       20. , 30. , 40. , 50. ])
```

[26]:
```
complete_list=np.concatenate((list1,list2))
print(complete_list)
```

```
[ 2.   4.   6.   8.  10.  12.2 14.  16.  18.  20.  10.  20.  30.  40.
 50. ]
```

[27]:
```
complete_list.sort()
print(complete_list)
```

```
[ 2.   4.   6.   8.  10.  10.  12.2 14.  16.  18.  20.  20.  30.  40.
 50. ]
```

3 2-D Array

[28]:
```
a=np.array([[1,2],[5,4]])
print(a)

print("shape of array",a.shape)
a.size
```

```
[[1 2]
 [5 4]]
shape of array (2, 2)
```

[28]:
```
4
```

[29]:
```
b=np.array([[6,7],[10,9]])
print(b)

print("daimention of array is",(b.ndim))
```

```
[[ 6  7]
 [10  9]]
daimention of array is 2
```

[30]:
```
concatenating=np.concatenate((a,b), axis=1)
print(concatenating)
```

```
[[ 1  2  6  7]
 [ 5  4 10  9]]
```

[31]:
```
concatenating=np.concatenate((a,b,), axis=0)
print(concatenating)
```

```
[[ 1  2]
 [ 5  4]
 [ 6  7]
 [10  9]]
```

4 3-D Array

[32]:
```
three_d_array_1=np.
  ↪array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]],[[13,14,15],[16,16,18]]])
print(three_d_array_1)

print("Size of array is",three_d_array_1.size)

print("Shape of array is",three_d_array_1.shape)

print("daimention of array is",(three_d_array_1.ndim))
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[13 14 15]
  [16 16 18]]]
Size of array is 18
Shape of array is (3, 2, 3)
daimention of array is 3
```

[33]:
```
three_d_array_2=np.
  ↪array([[[19,20,21],[22,23,24]],[[25,26,27],[28,29,30]],[[31,32,33],[34,35,36]]])
print(three_d_array_2)
```

```
[[[19 20 21]
  [22 23 24]]

 [[25 26 27]
  [28 29 30]]
```

```
 [[31 32 33]
  [34 35 36]]]
```

[34]:
```
joined=np.concatenate((three_d_array_1,three_d_array_2))
print(joined)

joined.size
joined.shape
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[13 14 15]
  [16 16 18]]

 [[19 20 21]
  [22 23 24]]

 [[25 26 27]
  [28 29 30]]

 [[31 32 33]
  [34 35 36]]]
```

[34]:
```
(6, 2, 3)
```

[35]:
```
Range_1=np.arange(1,19)
print(Range_1)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18]
```

[36]:
```
reshape1=Range_1.reshape(3,2,3)
print(reshape1)
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[13 14 15]
  [16 17 18]]]
```

[37]:
```
a=np.arange(9)
print(a)
```

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

[38]:
```
# rowise array
b=a[np.newaxis, :]
print(b)
b.shape
```

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

[38]:
```
(1, 9)
```

[39]:
```
#columnwise
c=a[:, np.newaxis]
print(c)
print(c.shape)
c.ndim
```

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

[39]:
```
2
```

[40]:
```
# multiplication of array

b*6
```

[40]:
```
array([[ 0,  6, 12, 18, 24, 30, 36, 42, 48]])
```

[41]:
```
b+5
```

[41]:
```
array([[ 5,  6,  7,  8,  9, 10, 11, 12, 13]])
```

[42]:
```
b/2
```

[42]:
```
array([[0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. ]])
```

[43]:
```
b-4
```

[43]:
```
array([[-4, -3, -2, -1,  0,  1,  2,  3,  4]])
```

[44]:
```
b.sum()
```

[44]:
```
36
```

[45]:
```
b.mean()
```

[45]:
```
4.0
```

5 USING NUMPY Accessing Array Element

[46]:
```
import numpy as np
num=np.arange(10)
print(num)

print(num[1])

#Adding two number by giving their index number in 1-D array
print(num[-1]+num[-2])
```

```
[0 1 2 3 4 5 6 7 8 9]
1
17
```

[47]:
```
# Accessing element in 2-D Array

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

print("  ")
#Now we access 1 element of 1 row which is 2
print(x[0,1])

print("  ")
#Now we access 1 element of 2 row which is 9
print(x[1,1])

print("  ")
#Now we add 1 element of 1 row which is 2 and 1 element of 2 row which is 9  =␣
 ↪11
print(x[0,1]+x[1,1])
```

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

2
```

```
9

11
```

[48]:
```
# Accessing element in 3-D Array

y=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[9,8,7]],[[6,5,4],[3,2,1]]])
print(y)
```

```
[[[1 2 3]
  [4 5 6]]

 [[7 8 9]
  [9 8 7]]

 [[6 5 4]
  [3 2 1]]]
```

[49]:
```
#Now we Access 1 element of 1st 2-d Array in 1st Row Which is 2
print(y[0,0,1])

print("  ")
#Now we Access 1 element of 1st 2-d Array in 2nd Row Which is 5
print(y[0,1,1])

print("  ")
#Now we Access 1 element of 2nd 2-d Array in 1st Row Which is 8
print(y[1,0,1])

print("  ")
#Now we Access 1 element of 2nd 2-d Array in 2nd Row Which is 8
print(y[1,1,1])

print("  ")
#Now we Access 1 element of 3rd 2-d Array in 1st Row Which is 5
print(y[2,0,1])


print("  ")
#Now we Access 1 element of 2nd 2-d Array in 1st Row Which is 8
print(y[2,1,1])


print("  ")
#Adding 1st 2-D Array of 1st line 1st Number + 2nd 2-D Array of 1st line 1st␣
 ↪Number + 3rd 2-D Array of 1st line 1st Number=1+7+6=14
```

```
print(y[0,0,0]+y[1,0,0]+y[2,0,0])
```

```
2

5

8

8

5

2

14
```

6 NUMPY Array Slicing

[50]:
```
import numpy as np

# here we have take range from 0 to 9 means 10 number.
num=np.arange(10)
print(num)

#we can access number through positive and negative indexing
print(num[1]) # which is 2
print(num[0:3]) # will print 0,1,2 the last one is exclusive
print(num[-1]) # which is last one =9

# here we are giving condintion that print the number with gape of 3
print(num[1::3]) # will print 1,4,7


# here we are giving condintion that print the number with gape of 2
print(num[1::2]) # will print 1,3,5,7,9

print(num[1:3]+num[3:5]) # here the addition will be perform like this [1+3 :␣
↪2+4]=[4 : 6]
print(np.concatenate([num[1:3],num[3:5]])) # here we concatenating O/P will be␣
↪[1,2,3,4]
```

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

```
[4 6]
[1 2 3 4]
```

[51]:
```
# Slicing in 2-D Array

x=np.arange(10) # here we have give a range from 0-9 means 10 numbers
x=x.reshape(2,5) # here we are converting the above range into 2_D array
print(x) # print 2-D array

# here we will access 2 to 4 number from 1st line = [2,3,4]
print("  ")
print(x[0,2:5])

# here we will access 1 from 1st line
print("  ")
print(x[0:1, 1]) # here from 1st line 1 will print =[1]

#here is another way
print("  ")
print(x[0,1:2]) # O/P will be = [1]

# here we will access number from both line which are on the same position
print("  ")
print(x[0:2, 1]) # here from 1st line 1 and from 2nd line 6 =[1,6]

# here we will access 5 to 7 number from 1st line = [5,6,7]
print("  ")
print(x[1, 0:3])

# now we will access number with gape
print("  ")
print(x[0, 0:5:2])


print(x[0,1:2])
```

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

[2 3 4]

[1]

[1]

[1 6]

[5 6 7]
```

```
[0 2 4]
[1]
```

[52]:
```
# Slicing in 3-D Array

y=np.arange(24)
y=y.reshape(3,2,4)
print(y)
```

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

 [[ 8  9 10 11]
  [12 13 14 15]]

 [[16 17 18 19]
  [20 21 22 23]]]
```

[53]:
```
#Now we Access 0:2 element of 1st 2-d Array in 1st Row Which is [0,1,2]
print(y[0,0, 0:3])

print("  ")
#Now we Access 12 and 15 of 2nd 2-d Array in 2nd Row Which is [12,15]
print(y[1,1, 0::3])
```

```
[0 1 2]

[12 15]
```

7 NUMPY Data Type

[54]:
```
# data type in python: (1)string  (2) integer (3)float (4)boolean (5)complex

#data type in numpy:
# i for integer
# b for boolean
# c for complex float
# f for float
# u for unsigned integer
# m for timedelta
# M for datetime
# O for object
# S for string
# U for unicode string
# V for memory alocation
```

[55]:
```
import numpy as np

# lets take example it will give O/P numarically if there are 4 numbers it will␣
 ↪show int32. 8 bit for each

num=np.array([1,2,3,4])
print(num.dtype)
```

```
int32
```

[56]:
```
# it will show U and with number od maximum word contain, ex# here fahadKhan␣
 ↪contain 9 letter it will show U9
string=np.array(["fahadKhan","ijaz","baba"])
print(string.dtype)
```

```
<U9
```

[57]:
```
# creating array with defined datatype:

num=np.array([1,2,3,4], dtype='S')
print(num.dtype)
print(num)
num=np.array([1,2,3,4], dtype='c')
print(num)
print(num.dtype)
```

```
|S1
[b'1' b'2' b'3' b'4']
[b'1' b'2' b'3' b'4']
|S1
```

[58]:
```
# creating array with defined datatype with defined bytes:
num=np.array([1,2,3,4], dtype='i2') # here it will 2*8 (each has 8 bytes)=int16
print(num)
print(num.dtype)
```

```
[1 2 3 4]
int16
```

[59]:
```
num=np.array([1,2,3,4], dtype='i8') # here it will 8*8=int64
print(num.dtype)
```

```
int64
```

[60]:
```
num=np.array(['1','2','3','4'], dtype='i')
print(num)
print(num.dtype)
```

```
[1 2 3 4]
int32
```

[61]:
```
num=np.array([1.1,2.2,3.3])
print(num)
print(num.dtype)
```

```
[1.1 2.2 3.3]
float64
```

[62]:
```
num=np.array([1.1,2.2,3.3], dtype='i') # 1st method
print(num)
print(num.dtype)
```

```
[1 2 3]
int32
```

[63]:
```
num=np.array([1.1,2.2,3.3])
num1=num.astype('i') # 2nd method we can also write int without comma
print(num1)
print(num1.dtype)

num=np.array([1.1,2.2,3.3])
num1=num.astype(int) # 2nd method we can also write int without comma
print(num1)
print(num1.dtype)
```

```
[1 2 3]
int32
[1 2 3]
int32
```

[64]:
```
# we can also conver them into boolean
num=np.array([1.1,2.2,3.3])
num1=num.astype(bool) # 1st method
print(num1)
print(num1.dtype)
```

```
[ True  True  True]
bool
```

[65]:
```
# we can also conver them into boolean
num=np.array([1.1,2.2,3.3])
num1=num>0 #2nd method
print(num1)
print(num1.dtype)
```

```
[ True  True  True]
```

```
bool
```

8 NUMPY Array Copy and View

[66]:
```
import numpy as np

num=np.arange(5)
num1=num.copy() # when we copy array we can change values it will just change
 ↪in copy rhe orignal one remain same.

print(num)
print("  ")
num1[0]=342
print(num1)
```

```
[0 1 2 3 4]

[342   1   2   3   4]
```

[67]:
```
# .view() is used to view the array if any changes occure the both will show
 ↪same result
num=np.arange(5)
num1=num.view()
print(num1)
print("  ")
num[0]=22
print(num)
```

```
[0 1 2 3 4]

[22  1  2  3  4]
```

[68]:
```
# .view() is used to view the array if any changes occure the both will show
 ↪same result
num=np.arange(5)
num1=num.view()
num[0]=22
print(num1)
print("  ")
print(num)
```

```
[22  1  2  3  4]

[22  1  2  3  4]
```

9 NumPy Array Shape

[69]:
```
import numpy as np
# just write a variable_Name.shape

num=np.array([1,2,3,4], ndmin=32)
print(num.shape)
print(num)
```

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

10 NumPy Array Iterating

[70]:
```
import numpy as np
#Iteration in One D Array

one_d=np.arange(5)
print(one_d)

for i in one_d:
    print(i)
```

```
[0 1 2 3 4]
0
1
2
3
4
```

[71]:
```
import numpy as np

two_d = np.arange(8)
two_d=two_d.reshape(2, 4)

print(two_d)
print(' ')

print(two_d.ndim)

print(' ')

for i in two_d:
    print(i)

print(' ')
```

```
for i in two_d:  # we use nested loop for two d array.
    for j in i:
        print(j)
```

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

2

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

0
1
2
3
4
5
6
7
```

[72]:
```
three_d=np.arange(12)
three_d=three_d.reshape(3,2,2)

print(three_d)

# print('  ')

# for i in three_d:
#     print(i)

# print('  ')

# for i in three_d:
#     for j in i:
#         print(j)

# print('  ')

for i in three_d:
        for j in i:
        for k in j:
            print(k)
```

```
  Cell In[72], line 21
    for k in j:
```

```
    ^
IndentationError: expected an indented block after 'for' statement on line 20
```

[ ]:
```
# above method were soo complex so we use numpay NDITER for iteration
# we can use it in all types of array means 1-d ,2-d , 3-d
three_d=np.arange(12)
three_d=three_d.reshape(3,2,2)
print(three_d)

for i in np.nditer(three_d):
    print(i)
```

```
[[[ 0  1]
  [ 2  3]]

 [[ 4  5]
  [ 6  7]]

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

[ ]:
```
# here we are passingout valuse by 1 in 3-d there is little bit difference in␣
  ↪2-d
three_d=np.arange(12)
three_d=three_d.reshape(3,2,2)
print(three_d)

for i in np.nditer(three_d[:, :,::2]):
    print(i)
```

```
[[[ 0  1]
  [ 2  3]]

 [[ 4  5]
```

```
  [ 6  7]]

 [[ 8  9]
  [10 11]]]
0
2
4
6
8
10
```

[ ]:
```
# passingout value in 2-d array
two_d=np.arange(12)
two_d=two_d.reshape(2,6)
print(two_d)

for i in np.nditer(three_d[:,::2]):
    print(i)
```

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

[ ]:
```
# passingout value in 1-d array
one_d=np.arange(12)
print(one_d)

for i in np.nditer(one_d[::2]):
    print(i)
```

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

11 Numpy Array Joining Concatinate

[ ]:
```
import numpy as np

a = np.array([1,2,3,4])
b = np.array([4,3,2,1])
c=np.concatenate((a,b))
print(c)

d=np.array([[1,2,3],[1,2,3]])
a=np.array([[11,22,33],[44,55,66]])
c=np.concatenate((d,a), axis=1)
print(c)
```

```
[1 2 3 4 4 3 2 1]
[[ 1  2  3 11 22 33]
 [ 1  2  3 44 55 66]]
```

[ ]:
```
d=np.array([[1,2,3],[1,2,3]])
a=np.array([[11,22,33],[44,55,66]])
c=np.stack((d,a), axis=1)
print(c)
```

```
[[[ 1  2  3]
  [11 22 33]]

 [[ 1  2  3]
  [44 55 66]]]
```

[ ]:
```
a = np.array([1,2,3,4])
b = np.array([4,3,2,1])
c=np.stack((a,b), axis=1)
print(c)
```

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

[ ]:
```
# stacking along rows
a = np.array([1,2,3,4])
b = np.array([4,3,2,1])
c=np.hstack((a,b))
print(c)
```

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

[ ]:
```
# stacking along column
a = np.array([1,2,3,4])
b = np.array([4,3,2,1])
c=np.vstack((a,b))
print(c)
```

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

[ ]:
```
# stacking along height(depth)
a = np.array([1,2,3,4])
b = np.array([4,3,2,1])
c=np.dstack((a,b))
print(c)
```

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

12 NumPy Array Spliting

[ ]:
```
import numpy as np

spliting=np.arange(6)
result=np.array_split(spliting, 4)
print(result)
```

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

[ ]:
```
# getting splitted array with index in 1-d array
spliting=np.arange(6)
result=np.array_split(spliting, 4)

print(result[0])

print(result[2])
```

```
[0 1]
[4]
```

[ ]:
```
spliting=np.arange(6)
spliting=spliting.reshape(2,3)
print(spliting.ndim)
result=np.array_split(spliting, 3)
print(result)
```

```
2
[array([[0, 1, 2]]), array([[3, 4, 5]]), array([], shape=(0, 3), dtype=int32)]
```

[ ]:
```
# getting splitted array with index in 2-d Array

spliting=np.arange(6)
spliting=spliting.reshape(2,3)
result=np.array_split(spliting, 3, axis=1)
print(result)
```

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

13 NumPy Searching Array

[ ]:
```
import numpy as np
# here we use np.where() and will get index of those valuse which are␣
 ↪satisfaing the condition

x= np.array([1,2,3,4,5,6,1,2,3,1])
x=np.where(x ==1) # will return the index of all 1 means [0,6,9]
print(x)
```

```
(array([0, 6, 9], dtype=int64),)
```

[ ]:
```
x= np.array([1,2,3,4,5,6,1,2,3,1])
x=np.where(x%2==0) # will return the index of all even number means [1,3,5,7]
print(x)
```

```
(array([1, 3, 5, 7], dtype=int64),)
```

[ ]:
```
#we can search index by passing value and using searchsorted
x=np.array([1,3,2,3,4,2,5])
x=np.searchsorted(x, 3)
print(x)
```

```
1
```

[ ]:
```
#we can search index by passing value as well as side and using searchsorted
x=np.array([2,3,4,2,5])
x=np.searchsorted(x, 3, side='left')
print(x)
```

```
1
```

[ ]:
```
# we can all so find the index for (values we want to input in array)
#it will start indexing from minimum values.
#take blow example we are findind index fro [0,5,9] in array it is starting␣
 ↪from 1 so the indexing will be start from 0 the output will [0,2,3]
a=np.array([1,2,7])
a=np.searchsorted(a, [0,5,9])
print(a)
```

```
[0 2 3]
```

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

```
(array([0, 0, 1], dtype=int64), array([1, 3, 3], dtype=int64))
```

14 NumPy Array Sorting()

[ ]:
```
import numpy as np
number=np.array([5,2,8,2,5,3])
print(np.sort(number))
```

```
[2 2 3 5 5 8]
```

[ ]:
```
# sorting in alphabate
alphabate=np.array(["fahad","adnan","owais","banana"])
print(np.sort(alphabate))
```

```
['adnan' 'banana' 'fahad' 'owais']
```

[ ]:
```
# sorting in boolean it will work like 0,1
boolean=np.array([True,False,True])
print(np.sort(boolean))
```

```
[False  True  True]
```

[ ]:
```
# sorting will work same as in 2-D and 3-D array
```

15 Numpy Filtter Array

[73]:
```
import numpy as np
a=np.array([2,34,5,43,22])
b=[True,False,True,True,False]
c=a[b]
print(c)
```

```
[ 2  5 43]
```

[ ]:
```
# Filtering in 2-D Array
a=np.array([[1,3,5,7],[23,44,22,33]])
b=[[True, False,True,False],[True, False,True,False]]
c=a[b]
print(c)
```

```
[ 1  5 23 22]
```

[3]:
```
# another way of filtering through condition
import numpy as np

a = np.array([34, 2, 4, 6, 88, 55, 32, 12, 12, 55, 75, 33, 6, 24])

# Boolean indexing
b = a[a > 20]
c = a[a < 20]
print(np.sort(b))
print(np.sort(c))
```

```
[24 32 33 34 55 55 75 88]
[ 2  4  6  6 12 12]
```

[9]:
```
# another way of filtering through condition
import numpy as np

a = np.array([34, 2, 4,55, 32, 12, 13])

# Boolean indexing
b = a[a%2==1]
c = a[a%2==0]
print(np.sort(b),"odd")
print(np.sort(c),"even")
```

```
[13 55] odd
[ 2  4 12 32 34] even
```

[12]:
```
# another way of filtering through condition
import numpy as np

a = np.array([34, 2, 4,55, 32, 12, 13])

# Boolean indexing
b = a%2==1

c=a[b]
print(c)
```

```
print(b)
print(a)
```

```
[55 13]
[False False False  True False False  True]
[34  2  4 55 32 12 13]
```

16 NumPy Random Numbers

[1]:
```
from numpy import random

#it will print numbers between 1 to 100 randomly
#if we dont pass any value than it will takes values between 0 to 1

fahad=random.randint(100)
print(fahad)
fahad=random.randint(100)
print(fahad)
fahad=random.randint(100)
print(fahad)
fahad=random.randint(100)
print(fahad)
fahad=random.randint(100)
print(fahad)
fahad=random.randint(100)
print(fahad)
fahad=random.randint(100)
print(fahad)
```

```
38
16
11
62
29
80
57
```

[2]:
```
number= random.rand()
print(number)
```

```
0.19497463671065873
```

[32]:
```
# creating array with random values

array=random.randint(3.0, size=3)
print(array)
```

```
array=random.randint(2.5, size=3)
print(array)

array=random.randint(2.0, size=3)
print(array)

array=random.randint(1.5, size=3)
print(array)

array=random.randint(1.0, size=3)
print(array)
```

```
[0 0 2]
[1 1 0]
[1 0 1]
[0 0 0]
[0 0 0]
```

[66]:
```
# creating array with random values we should use () in size
# here we are creating 1-D Array

array=random.randint(30, size=(3))
print(array)

array=random.randint(25, size=3)
print(array)

array=random.randint(20, size=3)
print(array)

array=random.randint(15, size=3)
print(array)

array=random.randint(10, size=(3))
print(array)
```

```
[18 19 23]
[16  6  0]
[ 5  1 19]
[3 7 2]
[4 4 5]
```

[67]:
```
# here we are creating 1-D array

n=random.randint(50, size=(3))
print(n)
```

```
# Float value here we are creating 1-D array
print("  ")

n=random.rand(2)
print(n)
```

```
[40 25 44]

[0.76599893 0.3637848 ]
```

[63]:
```
# here we are creating 2-D array we just pass two argument in size one for row
 ↪and second for column

n=random.randint(50, size=(2,4))
print(n)

# Float value here we are creating 2-D array we just pass two argument in size
 ↪one for row and second for column
print("  ")

n=random.rand(2,4)
print(n)
```

```
[[13 47 16  0]
 [47 33  2 26]]

[[0.63255263 0.72707466 0.63557417 0.18073896]
 [0.38151201 0.0522074  0.80974942 0.55513179]]
```

[65]:
```
# here we are creating 3-D array we just pass two argument in size one for row
 ↪and second for column

n=random.randint(50, size=(2,4))
print(n)

# Float value here we are creating 3-D array we just pass two argument in size
 ↪one for row and second for column
print("  ")

n=random.rand(3,2,3)
print(n)
```

```
[[28 24 43 19]
 [27 26  8 48]]

[[[0.82028368 0.89163982 0.1143159 ]
  [0.85392205 0.18006386 0.97598263]]
```

```
[[0.66587293 0.36512709 0.5354366 ]
 [0.62874342 0.30332988 0.41263776]]

[[0.75684843 0.00426023 0.17920462]
 [0.73824522 0.54374128 0.12789296]]]
```

[68]:
```
# we can give vales and ask to random module to choose from it, 1-D array

n=random.choice([2,4,5,6,3,7,2,8], size=(3))
print(n)
```

```
[2 5 3]
```

[72]:
```
# we can give vales and ask to random module to choose from it, 2-D array

n=random.choice([0,1,5,0,1], size=(2,4))
print(n)
```

```
[[1 1 5 0]
 [5 0 0 5]]
```

[74]:
```
# we can give vales and ask to random module to choose from it, 3-D array

n=random.choice([2,"#",5,6,3,7,2,8,11,22,33], size=(3,2,4))
print(n)
```

```
[[['22' '11' '7' '5']
  ['3' '2' '11' '11']]

 [['8' '#' '22' '2']
  ['2' '8' '3' '8']]

 [['2' '5' '#' '5']
  ['#' '33' '22' '#']]]
```

17 NumPy Random data Distribution

[10]:
```
from numpy import random
# data distribution is a list of all possible value and how often each value␣
 ↪occure
# such lists are important when workin with statistics and data science

# Random Distribution: Probability Function

# We will set value for each number

```

```
fahad=random.choice([4,5,3,2], p=[0.5,0.2,0.2,0.1], size=(10))
print(fahad)
```

```
[4 2 3 4 4 5 5 4 4 4]
```

[12]:
```
fahad=random.choice([1,2,3,4], p=[0.5,0.1,0.2,0.2], size=(50))
print(fahad)
```

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

[23]:
```
from numpy import random

fahad = random.choice([1, 2, 3, 4], p=[0.5, 0.1, 0.2, 0.2], size=(1000))
count_1 = (fahad == 1).sum()
percentage_1 = count_1 / len(fahad) * 100

print(f"Percentage of 1s: {percentage_1}%")
```

```
Percentage of 1s: 46.300000000000004%
```

[29]:
```
# using random module for probability function we are creating 2-D array

two_d_array=random.choice([9,5,2,7,4,3], p=[0.1,0.1,0.1,0.1,0.6,0.0],␣
 ↪size=(3,8))
print(two_d_array)

print(" ")

two_d_array=random.choice([5,2,7,4,3], p=[0.1,0.1,0.1,0.1,0.6], size=(2,8))
print(two_d_array)
```

```
[[4 9 7 9 9 9 4 4]
 [2 4 4 4 9 5 9 7]
 [4 4 4 4 4 4 4 7]]

[[3 3 3 3 5 3 2 2]
 [3 4 3 3 5 3 5 3]]
```

18 NumPy random Permutation and Shuffling

[33]:
```
# Permutation refers to an arrangement of elements like [3,2,1] is permutation␣
 ↪of [1,2,3] and vice versa.
# The NumPy Random module provide 2 Methods: Shuffle() and Permutation.
#Now we will randomly shuffle elements for the below array:
```

```
# shuffle works on orignal array but permutation done not

from numpy import random
import numpy as np

num=np.array([2,3,5,75,3,2])
random.shuffle(num)
print(num)
```

```
[75  3  2  5  2  3]
```

[36]:
```
num=np.array([2,3,5,75,3,2])
print(random.shuffle(num))
```

```
None
```

[38]:
```
# now we are taking example od permutation
# it will never change any thing in the array, it works on copy
num=np.array([2,3,5,75,3,2])
random.permutation(num)
print(num)
```

```
[ 2  3  5 75  3  2]
```

[44]:
```
num=np.array([2,3,5,75,3,2])
print(random.permutation(num1))

print("  ")
print(num) # the orignal one is unchanged
```

```
[ 5  3 75  2  2  3]

[ 2  3  5 75  3  2]
```

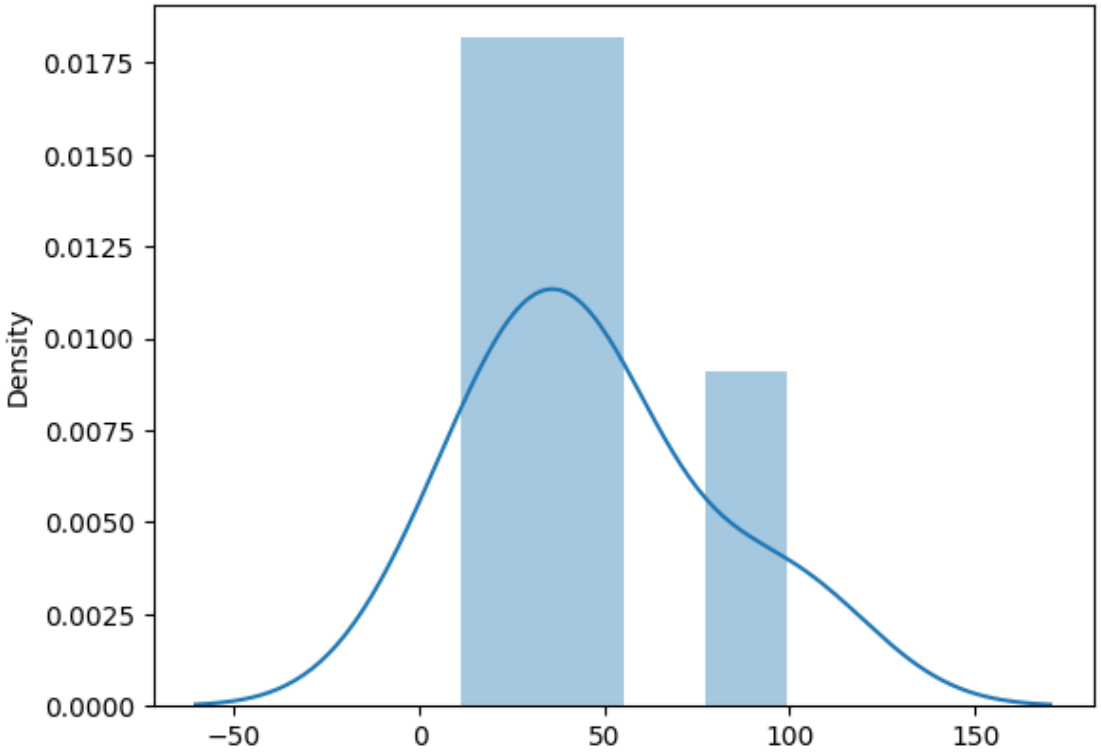
19 NumPy Seaborn [Visualize Distribution with seaborn]

[ ]:
```
# matplotlib (pyplot) => seaborn
# Seaborn is a library that uses matplotlib underneath to ploy graph i.r pyplot
# Displot => distribution plot( curve plot- histogram)
```

[12]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot((11,31,99,41,51))
plt.show()
```

```
C:\Users\fahad\AppData\Local\Temp\ipykernel_9000\3929550764.py:3: UserWarning:
```

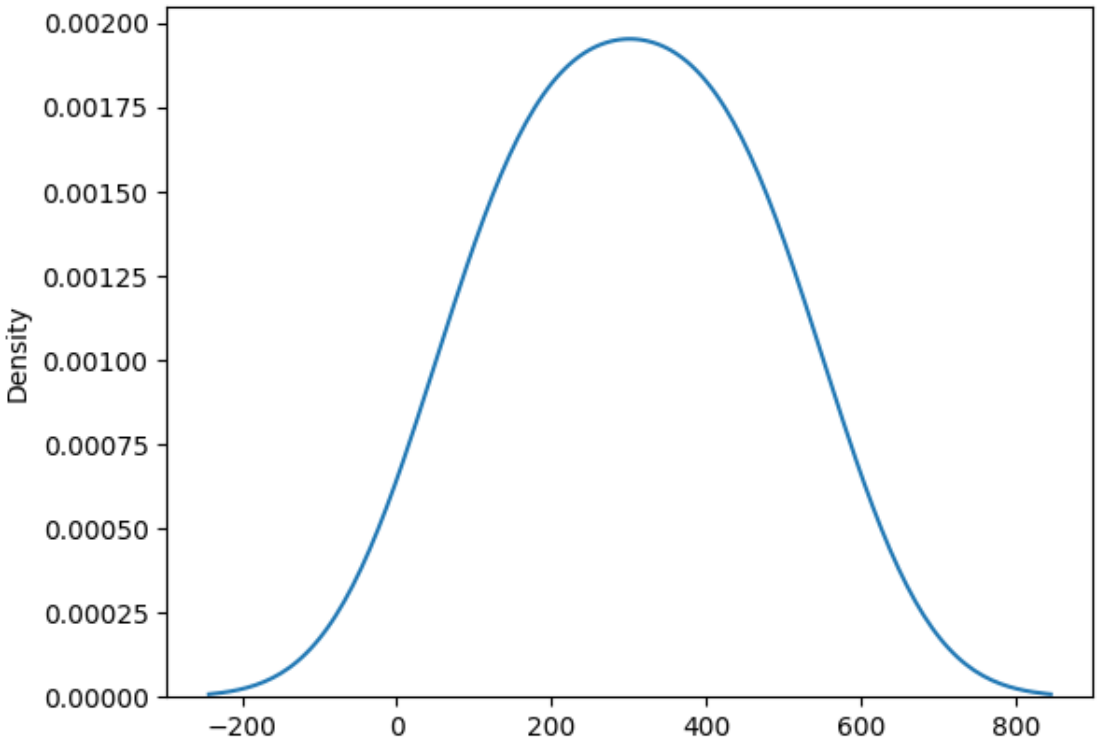
```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot((11,31,99,41,51))
```



[13]:
```
# plotting a distplot without the histrogram

import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot((101,201,301,401,501), hist=False)
plt.show()
```

```
C:\Users\fahad\AppData\Local\Temp\ipykernel_9000\290874189.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density
```

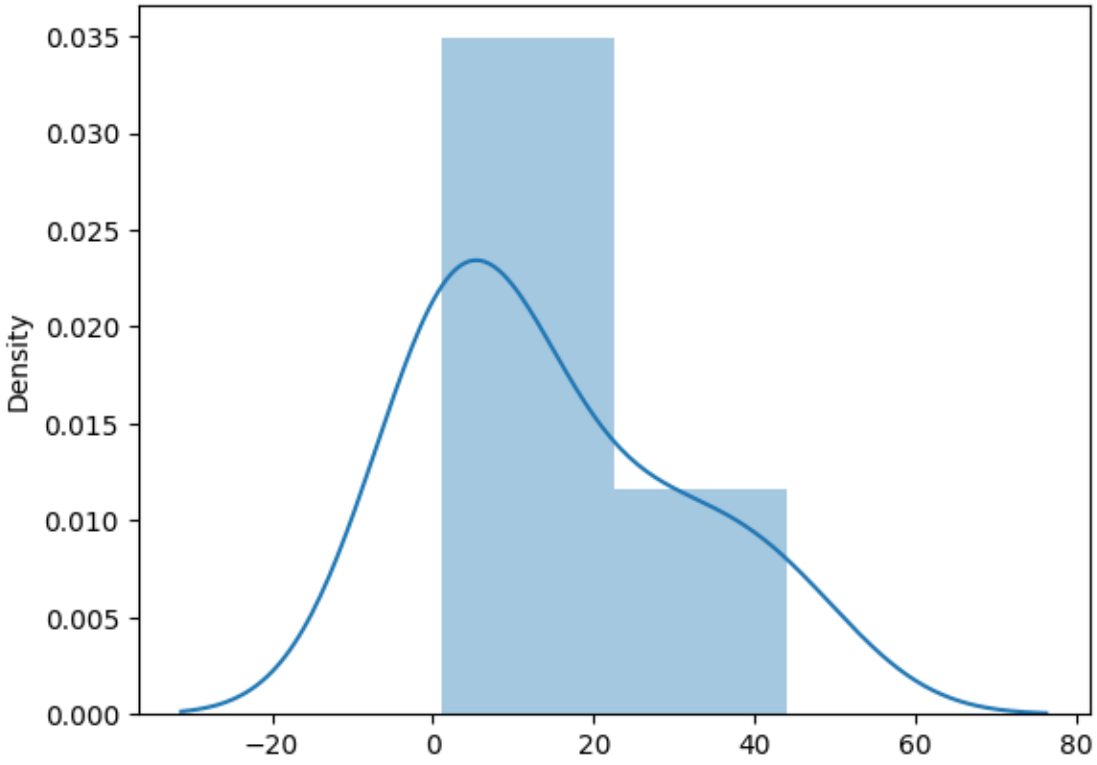
```
plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot((101,201,301,401,501), hist=False)
```



[15]:
```
import numpy as np

x=np.array([[1,2,3,4],[11,22,33,44]])
sns.distplot(x)
```

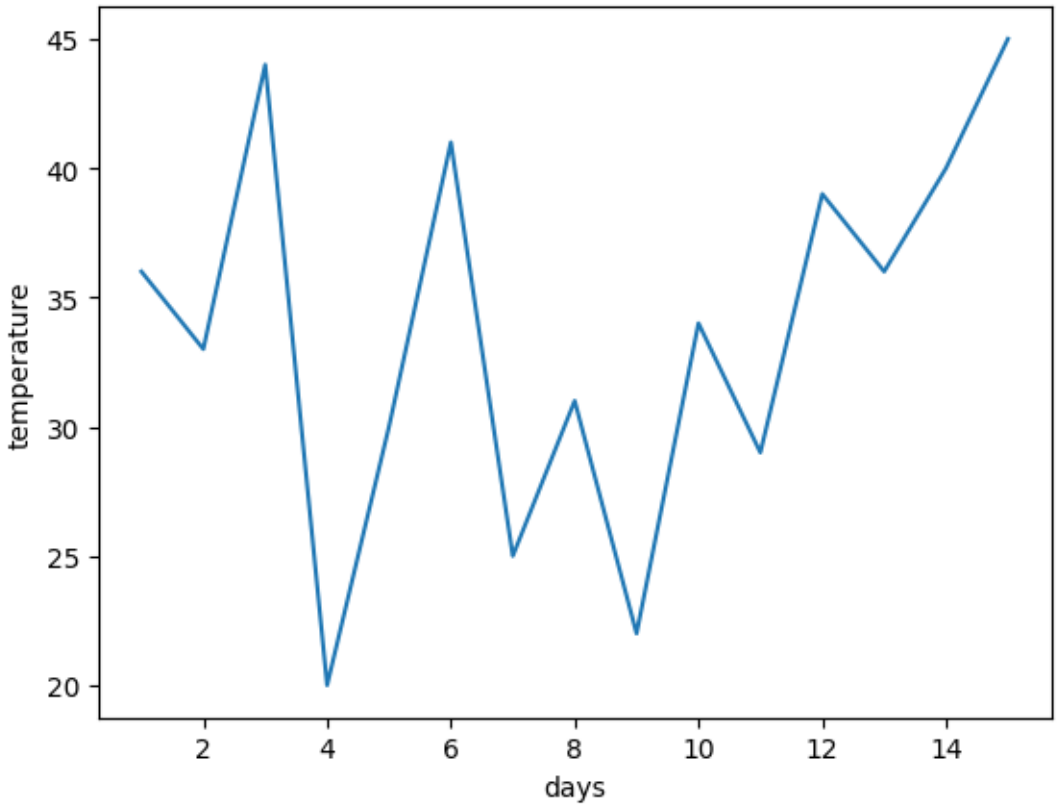
```
C:\Users\fahad\AppData\Local\Temp\ipykernel_9000\1047138909.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(x)
```

[15]: <Axes: ylabel='Density'>



[4]:
```
# now creating line plot using seaborn pandas and matplotlib.pyplot

import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

days=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
temperature=[36,33,44,20,30,41,25,31,22,34,29,39,36,40,45]

data_frame=pd.DataFrame({"days":days, "temperature":temperature})
sns.lineplot(x="days", y="temperature", data=data_frame)
```

[4]: <Axes: xlabel='days', ylabel='temperature'>


45
40
35
30
25
20
temperature
2
4
6
8
10
12
14
days

[ ]: