In [2]:
```
import numpy as np
```

1-D array

In [3]:
```
list1 =[12,34,56,56]
```

In [5]:
```
arr1d = np.array(list1)
```

In [6]:
```
type(arr1d)
```

Out[6]:
```
numpy.ndarray
```

In [7]:
```
arr1d
```

Out[7]:
```
array([12, 34, 56, 56])
```

In [8]:
```
list1.append(55)
```

In [9]:
```
list1
```

Out[9]:
```
[12, 34, 56, 56, 55]
```

In [11]:
```
arr1d +30  #here can possible to add the values.
```

Out[11]:
```
array([42, 64, 86, 86])
```

In [10]:
```
list +2 #concatanation can't done with the int .It can be possoble by using one more list.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-251d964fe16c> in <module>
----> 1 list +2

TypeError: unsupported operand type(s) for +: 'type' and 'int'
```

2-D array.

In [13]:
```
list2 = [[23,34,45,67,34],[12,78,75,90,33]]
array = np.array(list2)
print(array)
```

```
[[23 34 45 67 34]
 [12 78 75 90 33]]
```

In [15]:
```
type(array)
```

Out[15]:
```
numpy.ndarray
```

In [17]:
```
array.dtype
```

Out[17]:
```
dtype('int32')
```

In [18]:
```
array2d = np.array(list2,dtype=float)
print(array2d)
```

```
[[23. 34. 45. 67. 34.]
 [12. 78. 75. 90. 33.]]
```

In [19]:
```
array2d.astype(int)
```

Out[19]:
```
array([[23, 34, 45, 67, 34],
       [12, 78, 75, 90, 33]])
```

In [20]:
```
array2d.astype(str)
```

Out[20]:
```
array([['23.0', '34.0', '45.0', '67.0', '34.0'],
       ['12.0', '78.0', '75.0', '90.0', '33.0']], dtype='<U32')
```

In [21]:
```
array2d.tostring()  #it converted to the byte string
```

Out[21]:
```
b'\x00\x00\x00\x00\x00\x007@\x00\x00\x00\x00\x00\x00A@\x00\x00\x00\x00\x00\x80F@\x00\x00\x00\x00\x00\xc0P@\x00\x00\x00\x00\x00\x00A@\x00\x00\x00\x00\x00\x00(@\x00\x00\x00\x00\x00\x80S@\x00\x00\x00\x00\x00\xc0R@\x00\x00\x00\x00\x00\x80V@\x00\x00\x00\x00\x00\x80@@'
```

In [22]:
```
array2d.tobytes()
```

Out[22]:
```
b'\x00\x00\x00\x00\x00\x007@\x00\x00\x00\x00\x00\x00A@\x00\x00\x00\x00\x00\x80F@\x00\x00\x00\x00\x00\xc0P@\x00\x00\x00\x00\x00\x00A@\x00\x00\x00\x00\x00\x00(@\x00\x00\x00\x00\x00\x80S@\x00\x00\x00\x00\x00\xc0R@\x00\x00\x00\x00\x00\x80V@\x00\x00\x00\x00\x00\x80@@'
```

Dtype and Shape.

In [23]:
```
array2d.astype(float)
```

Out[23]:
```
array([[23., 34., 45., 67., 34.],
       [12., 78., 75., 90., 33.]])
```

In [24]:
```
array2d
```

Out[24]:
```
array([[23., 34., 45., 67., 34.],
       [12., 78., 75., 90., 33.]])
```

In [25]:
```
print('Shape :',array2d.shape)
```

```
Shape : (2, 5)
```

In [26]:
```
print('Size :',array2d.size)
```

```
Size : 10
```

In [27]:
```
array2d.dtype
```

Out[27]:
```
dtype('float64')
```

In [29]:
```
arr1d.ndim
```

Out[29]:
```
1
```

In [30]:
```
array2d.ndim
```

Out[30]:
```
2
```

In [31]:
```
array3 = array2d*array2d
```

In [32]:
```
print(array3)
```

```
[[ 529. 1156. 2025. 4489. 1156.]
 [ 144. 6084. 5625. 8100. 1089.]]
```

In [33]:
```
array2d[0]
```

Out[33]:
```
array([23., 34., 45., 67., 34.])
```

In [34]:
```
array2d[1]
```

Out[34]:
```
array([12., 78., 75., 90., 33.])
```

In [35]:
```
arr1d[1]
```

Out[35]:
```
34
```

In [36]:
```
array2d[0][0]   #[row][col]
```

Out[36]:
```
23.0
```

Bool array

In [39]:
```
boolarray= array2d>30
boolarray
```

Out[39]:
```
array([[False,  True,  True,  True,  True],
       [False,  True,  True,  True,  True]])
```

In [44]:
```
array2d[boolarray] #return the true vallues.
```

Out[44]:
```
array([34., 45., 67., 34., 78., 75., 90., 33.])
```

In [46]:
```
array2d[:,:-1 ]
```

Out[46]:
```
array([[23., 34., 45., 67.],
       [12., 78., 75., 90.]])
```

In [47]:
```
array2d
```

Out[47]:
```
array([[23., 34., 45., 67., 34.],
       [12., 78., 75., 90., 33.]])
```

In [48]:
```
array2d[::-1,::-1]
```

Out[48]:
```
array([[33., 90., 75., 78., 12.],
       [34., 67., 45., 34., 23.]])
```

np.nan and np.inf

In [50]:
```
np.nan  #not a number
```

Out[50]:
```
nan
```

In [51]:
```
np.inf #infinite can't define.
```

Out[51]:
```
inf
```

In [53]:
```
array2d
```

Out[53]:
```
array([[23., 34., 45., 67., 34.],
       [12., 78., 75., 90., 33.]])
```

In [54]:
```
array2d[0][0]=np.nan
array2d[0][1]=np.inf
array2d
```

Out[54]:
```
array([[nan, inf, 45., 67., 34.],
       [12., 78., 75., 90., 33.]])
```

In [55]:
```
np.isnan(array2d)
```

Out[55]:
```
array([[ True, False, False, False, False],
       [False, False, False, False, False]])
```

In [56]:
```
np.isinf(array2d)
```

Out[56]:
```
array([[False,  True, False, False, False],
       [False, False, False, False, False]])
```

In [57]:
```
missing_flag = np.isnan(array2d)|np.isinf(array2d)
missing_flag
```

Out[57]:
```
array([[ True,  True, False, False, False],
       [False, False, False, False, False]])
```

In [58]:
```
#replace the inf and nan with 0.
```

In [59]:
```
array2d[missing_flag]
```

Out[59]:
```
array([nan, inf])
```

In [60]:
```
array2d[missing_flag] =0
```

In [61]:
```
array2d
```

Out[61]:
```
array([[ 0.,  0., 45., 67., 34.],
       [12., 78., 75., 90., 33.]])
```

Stastical operation.

In [62]:
```
array2d.mean()
```

Out[62]:
```
43.4
```

In [64]:
```
array2d.max()
```

Out[64]:
```
90.0
```

In [65]:
```
array2d.min()
```

Out[65]:
```
0.0
```

In [66]:
```
array2d.std()
```

Out[66]:
```
31.426740206391116
```

In [67]:
```
array2d.var() #variance
```

Out[67]:
```
987.64
```

In [68]:
```
array2d.squeeze()
```

Out[68]:
```
array([[ 0.,  0., 45., 67., 34.],
       [12., 78., 75., 90., 33.]])
```

In [69]:
```
array2d.cumsum()
```

Out[69]:
```
array([  0.,   0.,  45., 112., 146., 158., 236., 311., 401., 434.])
```

In [70]:
```
array2d.sum()
```

Out[70]:
```
434.0
```

In [73]:
```
arr = array2d[0:,:3]
```

In [74]:
```
arr
```

Out[74]:
```
array([[ 0.,  0., 45.],
       [12., 78., 75.]])
```

In [76]:
```
array2d[0:,2:]
```

Out[76]:
```
array([[45., 67., 34.],
       [75., 90., 33.]])
```

Reshape()

In [77]:
```
array2d.reshape(5,2)
```

Out[77]:
```
array([[ 0.,  0.],
       [45., 67.],
       [34., 12.],
       [78., 75.],
       [90., 33.]])
```

In [81]:
```
aa= array2d.flatten()
print(aa)  #it is just copy not will afftedt on the parrent array.
```

```
[ 0.  0. 45. 67. 34. 12. 78. 75. 90. 33.]
```

In [82]:
```
bb=array2d.ravel()
print(bb)#if we cange the anything inside the ravel then it will change automatically of array2d element.
#It is actualy the reference to the parent array and it is memory efficient.
```

```
[ 0.  0. 45. 67. 34. 12. 78. 75. 90. 33.]
```

In [83]:
```
#for an example:
bb[0]=1  #if we change the 0 index position value of the reval then it will affect on the main array.
```

In [84]:
```
array2d  #if we see the oth index position taken the value i.e.1.
```

Out[84]:
```
array([[ 1.,  0., 45., 67., 34.],
       [12., 78., 75., 90., 33.]])
```

Sequences,repetitions and random numbers.

In [85]:
```
np.arange(1,10,dtype ='int')
```

Out[85]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [87]:
```
np.arange(1,10,2)
```

Out[87]:
```
array([1, 3, 5, 7, 9])
```

In [90]:
```
np.linspace(1,100,100)
```

Out[90]:
```
array([  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,  11.,
        12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,  22.,
        23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,  33.,
        34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,  44.,
        45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,  55.,
        56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,  66.,
        67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,  77.,
        78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,  88.,
        89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,  99.,
       100.])
```

In [92]:
```
np.zeros([2,2])
```

Out[92]:
```
array([[0., 0.],
       [0., 0.]])
```

In [93]:
```
np.ones([3,3])
```

Out[93]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [94]:
```
a =[1,2,3]
```

In [95]:
```
print(a)
```

```
[1, 2, 3]
```

In [96]:
```
np.tile(a,3)  #it is repeated the same list element three times.
```

Out[96]:
```
array([1, 2, 3, 1, 2, 3, 1, 2, 3])
```

In [97]:
```
np.repeat(a,3) #it is repeatating the one by one element of the list.
```

Out[97]:
```
array([1, 1, 1, 2, 2, 2, 3, 3, 3])
```

In [98]:
```
np.repeat(array2d,3)
```

Out[98]:
```
array([ 1.,  1.,  1.,  0.,  0.,  0., 45., 45., 45., 67., 67., 67., 34.,
       34., 34., 12., 12., 12., 78., 78., 78., 75., 75., 75., 90., 90.,
       90., 33., 33., 33.])
```

Random

In [99]:
```
np.random.rand(3)  #it will gives us the dimensions.
```

Out[99]:
```
array([0.27919725, 0.24025436, 0.73724932])
```

In [100…
```
np.random.rand(3,3)
```

Out[100…
```
array([[0.06745237, 0.2375528 , 0.56718996],
       [0.6521476 , 0.64476953, 0.90109756],
       [0.91959317, 0.86181678, 0.19145437]])
```

In [101…
```
np.random.randn(3,3)  #it will give us the uniformaly distributed random numbers
```

Out[101…
```
array([[ 0.37653252,  0.97454075, -0.82976096],
       [-1.42364852, -0.12908292,  0.14716561],
       [ 0.24563172, -2.3261353 ,  0.32553691]])
```

In [105…
```
np.random.randint(0,10,size=[3,3]) #when we want the only int type of data randomly.
```

Out[105…
```
array([[0, 9, 2],
       [6, 4, 7],
       [1, 8, 7]])
```

In [107…
```
np.random.randint(0,10,size=9).reshape(3,3)
```

Out[107…
```
array([[4, 7, 8],
       [8, 6, 3],
       [6, 9, 1]])
```

By the some we want to reproduce the random numbers that time use the seed function.

In [108…
```
np.random.seed(0)
np.random.randint(1,10,[3,3])
```

Out[108…
```
array([[6, 1, 4],
       [4, 8, 4],
       [6, 3, 5]])
```

In [109…
```
np.random.seed(0)
np.random.randint(1,10,[3,3])  #it maintain the numbers if we cange the seed 1 then it will change the numbers.
```

Out[109…
```
array([[6, 1, 4],
       [4, 8, 4],
       [6, 3, 5]])
```

In [111…
```
np.random.seed(1)
np.random.randint(1,10,[3,3])#now it changed due to change in the seed.
```

Out[111…
```
array([[6, 9, 6],
       [1, 1, 2],
       [8, 7, 3]])
```

In [112…
```
np.unique(array2d) #return the unique numbers.
```

Out[112…
```
array([ 0.,  1., 12., 33., 34., 45., 67., 75., 78., 90.])
```

In [113…
```
#if we want to known the count that time
unique,counts = np.unique(array2d,return_counts=True)
```

In [114…
```
unique
```

Out[114…
```
array([ 0.,  1., 12., 33., 34., 45., 67., 75., 78., 90.])
```

In [115…
```
counts
```

Out[115…
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Numpy 2nd Part.

In [116…
```
arr = np.array([4,4,556,763,76,662,98,88,13])
arr
```

Out[116…
```
array([  4,   4, 556, 763,  76, 662,  98,  88,  13])
```

In [118…
```
index_grt20 = np.where(arr>20)
```

In [119…
```
index_grt20  #it will return the index postion.
```

Out[119…
```
(array([2, 3, 4, 5, 6, 7], dtype=int32),)
```

In [120…
```
arr[index_grt10]#it will return the values.
```

Out[120…
```
array([556, 763,  76, 662,  98,  88])
```

In [121…
```
arr>20
```

Out[121…
```
array([False, False,  True,  True,  True,  True,  True,  True, False])
```

In [122…
```
np.where(arr>20,'gt20','lr20')  #those who are greater than 20 represented by 'gt20' if not then it represented
```

Out[122…
```
array(['lr20', 'lr20', 'gt20', 'gt20', 'gt20', 'gt20', 'gt20', 'gt20',
       'lr20'], dtype='<U4')
```

In [125…
```
arr.max()  #it will give us the maximum value directly.
```

Out[125…
```
763
```

In [126…
```
arr.argmax()  #it wiil gives us the index position.
```

Out[126…
```
3
```

In [127…
```
arr[arr.argmax()]
```

Out[127…
```
763
```

In [128…
```
arr.argmin() #similarly
```

Out[128…
```
0
```

In [129…
```
arr[arr.argmin()]
```

Out[129…
```
4
```

Import the text file

In [137…
```
data=np.genfromtxt('C:/Users/Microsoft/Desktop/pandas/ml-data-files/data.csv',delimiter=',',filling_values=-10(
```

to turn off the scientific notation.

In [140…
```
np.set_printoptions(suppress=True)
data[:3]
```

Out[140…
```
array([[-1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ,
        -1000.        , -1000.        , -1000.        , -1000.        ],
       [   -0.08649912,     0.11848083,    -0.02075784,     0.09546704,
            0.19051735,     0.20827844,     0.24120596,     0.08614283,
            1.02866149,     0.0301287 ,    -1.26693287,     0.01189133,
            0.11862914,    -0.17136208,    -0.23013495,    -0.27257676,
            0.04454237,     0.03295845,    -0.23626836,    -0.03098087,
            0.06569398,     0.21798717,    -0.03557262,    -0.02004006,
           -0.00885055,    -0.05107759,     0.11138666,     0.22285188,
            0.79323519,     0.20553354,    -0.16306627,     0.00715929,
            0.18194654,     0.0075955 ,     0.17692452,     0.08970178,
           -0.10477582,    -0.32320886,     0.08770954,    -0.2203856 ,
           -0.3962091 ,     0.02746607,    -0.2846461 ,     0.33036949,
            0.04428755,     0.32360739,     0.02030764,     0.03891198,
            0.15667064,    -0.17894335,     0.32751315,    -0.13908795,
           -0.40527157,    -0.11110723,     0.01519251,     0.02680688,
           -0.15658962,     0.09902957,     0.12809877,     0.27401199,
            0.17147569,     0.17197563,     0.2797261 ,    -0.00172266,
           -0.63018256,    -0.32264054,    -0.04576092,    -0.0018049 ,
           -0.0693687 ,    -0.33688083,    -0.100414  ,     0.0335996 ,
           -0.03659241,    -0.00822361,    -0.03846767,    -0.02498686,
            0.16894196,    -0.01077181,     0.31421319,     0.08287599,
            0.02331666,     0.26946019,    -0.06324149,     0.16463117,
           -0.00357734,    -0.24800819,    -0.06971136,     0.02624214,
           -0.14532503,     0.03751659,    -0.04334343,     0.06524113,
           -0.21944129,     0.1459193 ,     0.34284507,    -0.45479804,
            0.12371363,     0.3442846 ,    -0.17057948,     0.24305382,
           -0.00579616,     0.03216057,    -0.05788332,    -0.05616903,
           -0.17603609,    -0.00600777,    -0.10910645,    -0.04136181,
            0.05458794,     0.18859449,    -0.09477459,    -0.0237288 ],
       [   -0.1437924 ,     0.1182484 ,    -0.02400218,     0.09671633,
            0.19570523,    -0.03705217,     0.24795677,    -0.04595389,
            1.39729293,    -0.44840637,    -1.62960426,    -0.19998388,
            0.12007333,     0.05245593,    -0.24465164,    -1.13535693,
            0.0456243 ,     0.03279524,    -0.68317376,    -0.03113068,
            0.06610222,     0.23946477,     0.01852673,    -0.20424112,
           -0.01065773,    -0.39238298,     0.34180609,     0.66715807,
            0.84803739,    -0.01928862,    -0.16987185,    -0.08126863,
            0.18271371,     0.10968772,     0.21476339,     0.49337554,
           -0.79857915,    -0.64896082,     0.09240467,    -0.49329588,
           -0.42076355,     0.03688696,    -1.53280197,     0.17305754,
            0.04523265,     1.81151236,     0.01735107,     0.240997  ,
           -0.07644407,    -0.21673869,     0.34806859,     0.06547261,
           -0.46783674,    -0.11106306,     0.02013296,     0.55537771,
           -0.08484928,     0.75684713,     0.65364832,     0.3060715 ,
            0.18868794,     0.17243317,     0.17483266,     0.42223615,
           -1.3705853 ,    -0.35520042,    -0.00659092,    -0.00446057,
           -0.07243068,    -0.83085279,    -0.37426292,     0.0357125 ,
           -0.00580077,    -0.01723243,    -0.15465974,    -0.02530271,
            0.17914621,    -0.17077639,     0.32295581,     1.15755851,
            0.2079751 ,     1.62159888,    -0.75507665,     0.1727053 ,
           -0.00503595,    -0.29751919,    -0.07115484,     0.02450749,
           -0.45486147,    -0.25158945,    -0.04895936,    -0.00358193,
           -0.23041033,     0.14912678,     0.55103326,    -0.47090653,
            0.15330302,     0.76206904,    -0.17705423,     1.00162803,
           -0.00714129,     0.08217163,    -0.09785957,     0.03646329,
           -0.17978888,    -0.00421818,    -0.33920279,    -0.18261106,
            0.06451501,     0.18853393,    -0.12230375,    -0.02893746]])
```

In [ ]:
```

```

In [145…
```
np.stack(array2d)
```

Out[145…
```
array([[ 1.,  0., 45., 67., 34.],
       [12., 78., 75., 90., 33.]])
```

In [146…
```
np.hstack(array2d)
```

Out[146…
```
array([ 1.,  0., 45., 67., 34., 12., 78., 75., 90., 33.])
```

In [148…
```
np.vstack(array2d)
```

Out[148…
```
array([[ 1.,  0., 45., 67., 34.],
       [12., 78., 75., 90., 33.]])
```

In [149…
```
np.column_stack(array2d)
```

Out[149…
```
array([[ 1., 12.],
       [ 0., 78.],
       [45., 75.],
       [67., 90.],
       [34., 33.]])
```

In [150…
```
np.row_stack(array2d)
```

Out[150…
```
array([[ 1.,  0., 45., 67., 34.],
       [12., 78., 75., 90., 33.]])
```

In [ ]:
```

```

In [153…
```
d =np.datetime64('2019-06-02 23:10:23')
```

In [154…
```
d
```

Out[154…
```
numpy.datetime64('2019-06-02T23:10:23')
```

In [155…
```
d+1000 #it will helps to add the seconds.
```

Out[155…
```
numpy.datetime64('2019-06-02T23:27:03')
```

In [156…
```
oneday = np.timedelta64(1,'D')
```

In [157…
```
oneday
```

Out[157…
```
numpy.timedelta64(1,'D')
```

In [158…
```
oneday+1#It is helping to adding the day only.
```

Out[158…
```
numpy.timedelta64(2,'D')
```

In [159…
```
oneminute = np.timedelta64(1,'m')
oneminute
```

Out[159…
```
numpy.timedelta64(1,'m')
```

In [160…
```
oneminute+1#it is helping to add the one minute
```

Out[160…
```
numpy.timedelta64(2,'m')
```

In [162…
```
dates= np.arange(np.datetime64('2019-06-02'),np.datetime64('2020-06-02'),2)
dates
```

Out[162…
```
array(['2019-06-02', '2019-06-04', '2019-06-06', '2019-06-08',
       '2019-06-10', '2019-06-12', '2019-06-14', '2019-06-16',
       '2019-06-18', '2019-06-20', '2019-06-22', '2019-06-24',
       '2019-06-26', '2019-06-28', '2019-06-30', '2019-07-02',
       '2019-07-04', '2019-07-06', '2019-07-08', '2019-07-10',
       '2019-07-12', '2019-07-14', '2019-07-16', '2019-07-18',
       '2019-07-20', '2019-07-22', '2019-07-24', '2019-07-26',
       '2019-07-28', '2019-07-30', '2019-08-01', '2019-08-03',
       '2019-08-05', '2019-08-07', '2019-08-09', '2019-08-11',
       '2019-08-13', '2019-08-15', '2019-08-17', '2019-08-19',
       '2019-08-21', '2019-08-23', '2019-08-25', '2019-08-27',
       '2019-08-29', '2019-08-31', '2019-09-02', '2019-09-04',
       '2019-09-06', '2019-09-08', '2019-09-10', '2019-09-12',
       '2019-09-14', '2019-09-16', '2019-09-18', '2019-09-20',
       '2019-09-22', '2019-09-24', '2019-09-26', '2019-09-28',
       '2019-09-30', '2019-10-02', '2019-10-04', '2019-10-06',
       '2019-10-08', '2019-10-10', '2019-10-12', '2019-10-14',
       '2019-10-16', '2019-10-18', '2019-10-20', '2019-10-22',
       '2019-10-24', '2019-10-26', '2019-10-28', '2019-10-30',
       '2019-11-01', '2019-11-03', '2019-11-05', '2019-11-07',
       '2019-11-09', '2019-11-11', '2019-11-13', '2019-11-15',
       '2019-11-17', '2019-11-19', '2019-11-21', '2019-11-23',
       '2019-11-25', '2019-11-27', '2019-11-29', '2019-12-01',
       '2019-12-03', '2019-12-05', '2019-12-07', '2019-12-09',
       '2019-12-11', '2019-12-13', '2019-12-15', '2019-12-17',
       '2019-12-19', '2019-12-21', '2019-12-23', '2019-12-25',
       '2019-12-27', '2019-12-29', '2019-12-31', '2020-01-02',
       '2020-01-04', '2020-01-06', '2020-01-08', '2020-01-10',
       '2020-01-12', '2020-01-14', '2020-01-16', '2020-01-18',
       '2020-01-20', '2020-01-22', '2020-01-24', '2020-01-26',
       '2020-01-28', '2020-01-30', '2020-02-01', '2020-02-03',
       '2020-02-05', '2020-02-07', '2020-02-09', '2020-02-11',
       '2020-02-13', '2020-02-15', '2020-02-17', '2020-02-19',
       '2020-02-21', '2020-02-23', '2020-02-25', '2020-02-27',
       '2020-02-29', '2020-03-02', '2020-03-04', '2020-03-06',
       '2020-03-08', '2020-03-10', '2020-03-12', '2020-03-14',
       '2020-03-16', '2020-03-18', '2020-03-20', '2020-03-22',
       '2020-03-24', '2020-03-26', '2020-03-28', '2020-03-30',
       '2020-04-01', '2020-04-03', '2020-04-05', '2020-04-07',
       '2020-04-09', '2020-04-11', '2020-04-13', '2020-04-15',
       '2020-04-17', '2020-04-19', '2020-04-21', '2020-04-23',
       '2020-04-25', '2020-04-27', '2020-04-29', '2020-05-01',
       '2020-05-03', '2020-05-05', '2020-05-07', '2020-05-09',
       '2020-05-11', '2020-05-13', '2020-05-15', '2020-05-17',
       '2020-05-19', '2020-05-21', '2020-05-23', '2020-05-25',
       '2020-05-27', '2020-05-29', '2020-05-31'], dtype='datetime64[D]')
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```