In [1]:
```
print("Hello")
```

```
Hello
```

h1

h2

h3

h4

h5

h6

> Python is a Programming language

NumPy - "Numerical Python"

> **NumPy is a Python's library which is use to perform different mathematical operations on arrays.**
>
> **Array is denoted with "[ ]". It is ordered (index position is fixed), mutable (items can be modified) and homogenous (similar datatypes) collection of items.**

In [ ]:
```
#to install numpy
!pip install numpy
```

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

In [3]:
```
#to check the version of numpy
np.__version__
```

Out[3]: '1.26.4'

In [4]:
```
#Creating 1D-Array
a = np.array([1,3,5,7,9])
a
```

Out[4]: array([1, 3, 5, 7, 9])

In [6]:
```
type(a)  #ndarray: n-dimensional array
```

Out[6]: numpy.ndarray

In [7]:
```
print(a)
```

```
[1 3 5 7 9]
```

In [8]:
```
a
```

Out[8]:
```
array([1, 3, 5, 7, 9])
```

In [9]:
```
a = np.array([20,'tree',3.5,True])
a
```

Out[9]:
```
array(['20', 'tree', '3.5', 'True'], dtype='<U32')
```

In [12]:
```
a = np.array(['cat','tree','dog','aeroplane'])
a
```

Out[12]:
```
array(['cat', 'tree', 'dog', 'aeroplane'], dtype='<U9')
```

In [14]:
```
a = np.array(['cat','tree','dog','aeroplane',11,25])
a
```

Out[14]:
```
array(['cat', 'tree', 'dog', 'aeroplane', '11', '25'], dtype='<U11')
```

In [16]:
```
a = np.array(['cat','tree','dog','aeroplane',11,25],dtype='<U3')
a
```

Out[16]:
```
array(['cat', 'tre', 'dog', 'aer', '11', '25'], dtype='<U3')
```

In [18]:
```
a = np.array([20,3.5,55,True])
a
```

Out[18]:
```
array([20. ,  3.5, 55. ,  1. ])
```

In [19]:
```
#Tuple to Array
t = np.array((10,20,30,50))
t
```

Out[19]:
```
array([10, 20, 30, 50])
```

In [21]:
```
#Set to Array
s = np.array({'1',1,1,1,8,8,8,4,4,4})
s
```

Out[21]:
```
array({8, 1, 4, '1'}, dtype=object)
```

In [22]:
```
#Dict to Array
d = np.array({'a':1,'b':2,'c':3})
d
```

Out[22]:
```
array({'a': 1, 'b': 2, 'c': 3}, dtype=object)
```

In [26]:
```
#Creating 2D-Array
a = np.array([[1,3,5],[2,4,6]])
a
```

Out[26]:
```
array([[1, 3, 5],
       [2, 4, 6]])
```

In [1]:
```
import numpy as np
```

In [2]:
```
b = np.array([[1,2,3,4],[5,6,7,8],[2,9,6,4]])
b
```

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

In [3]:
```
#Array Attributes
b.shape
```

Out[3]:
```
(3, 4)
```

In [4]:
```
b.size
```

Out[4]:
```
12
```

In [5]:
```
b.ndim
```

Out[5]:
```
2
```

In [9]:
```
a = np.array([[[[1]]]])
a.ndim
```

Out[9]:
```
4
```

In [15]:
```
a = np.array([1],ndmin=32)
a
```

Out[15]:
```
array([[[[[[[[[[[[[[[[[[[[[[[[[[[[[1]]]]]]]]]]]]]]]]]]]]]]]]]]]])
```

In [17]:
```
#arange
np.arange(1,11,2)
```

Out[17]:
```
array([1, 3, 5, 7, 9])
```

In [18]:
```
np.arange(10,0,-1)
```

Out[18]:
```
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1])
```

In [19]:
```
#zeros
np.zeros(5)
```

Out[19]:
```
array([0., 0., 0., 0., 0.])
```

In [20]:
```
np.zeros((5,5))
```

Out[20]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [26]:
```
np.zeros((5,5),dtype=int)
```

Out[26]:
```
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

In [27]:
```
np.zeros((5,5)).astype(int)
```

Out[27]:
```
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

In [28]:
```
#ones
np.ones(5)
```

Out[28]:
```
array([1., 1., 1., 1., 1.])
```

In [29]:
```
np.ones((10,10))
```

Out[29]:
```
array([[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., 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., 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., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])
```

In [30]:
```
#full
np.full((5,5),7)
```

Out[30]:
```
array([[7, 7, 7, 7, 7],
       [7, 7, 7, 7, 7],
       [7, 7, 7, 7, 7],
       [7, 7, 7, 7, 7],
       [7, 7, 7, 7, 7]])
```

In [34]:
```
np.full((5,5),'😎')
```

Out[34]:
```
array([['😎', '😎', '😎', '😎', '😎'],
       ['😎', '😎', '😎', '😎', '😎'],
       ['😎', '😎', '😎', '😎', '😎'],
       ['😎', '😎', '😎', '😎', '😎'],
       ['😎', '😎', '😎', '😎', '😎']], dtype='<U1')
```

In [36]:
```
#eye - identity matrix
np.eye(5,dtype=int)
```

Out[36]:
```
array([[1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 0, 0, 1]])
```

In [42]:
```
#linspace - equally distributed numbers
```

```
In [42]: np.linspace(1,100,10)
```

```
Out[42]: array([  1.,  12.,  23.,  34.,  45.,  56.,  67.,  78.,  89., 100.])
```

```
In [46]: #Random Numbers
         np.random.rand()
```

```
Out[46]: 0.27407129441584843
```

```
In [47]: np.random.rand(5)
```

```
Out[47]: array([0.95848468, 0.19720434, 0.73807056, 0.3338385 , 0.12369492])
```

```
In [56]: np.random.rand(3,4)
```

```
Out[56]: array([[0.31949836, 0.67683876, 0.12388084, 0.28956098],
                [0.93445387, 0.80830122, 0.72544563, 0.85732649],
                [0.45541539, 0.08735575, 0.46621272, 0.35844503]])
```

```
In [55]: np.random.rand(3,4,5)
```

```
Out[55]: array([[[0.07231359, 0.27191426, 0.61430379, 0.32534918, 0.61321208],
                 [0.79844916, 0.86239188, 0.69009476, 0.37564406, 0.83382342],
                 [0.66932946, 0.77227066, 0.25995376, 0.00641763, 0.79802755],
                 [0.03369997, 0.03718426, 0.09915853, 0.31818728, 0.02382151]],

                [[0.01378429, 0.21435225, 0.45849899, 0.51250009, 0.83856398],
                 [0.4967119 , 0.39344791, 0.96716169, 0.45219818, 0.01581729],
                 [0.19062064, 0.46496714, 0.65660473, 0.07976462, 0.22047322],
                 [0.14570161, 0.63568156, 0.11421053, 0.99731248, 0.9516465 ]],

                [[0.54282226, 0.22336666, 0.01434185, 0.71813706, 0.91848183],
                 [0.68852184, 0.52861984, 0.21821017, 0.75972758, 0.53886095],
                 [0.76335987, 0.76651199, 0.68787407, 0.0530047 , 0.34032595],
                 [0.07588831, 0.39715505, 0.81934211, 0.94236777, 0.05882749]]])
```

```
In [52]: np.random.rand(3,4) * 100
```

```
Out[52]: array([[55.00935941, 28.05883497, 54.70434069, 58.96287503],
                [64.86370477,  4.50310407, 16.49139616, 17.78303052],
                [27.14684849, 18.13012135, 44.26375108, 33.5723771 ]])
```

```
In [59]: np.random.randint(1,20)
```

```
Out[59]: 10
```

```
In [60]: np.random.randint(1,20,(4,5))
```

```
Out[60]: array([[ 8,  7, 14, 17,  9],
                [ 6,  6, 19, 15,  5],
                [12,  5, 14,  3, 11],
                [17,  6,  9, 11,  4]])
```

```
In [63]: np.random.rand(1,20).astype(str)
```

Out[63]:
```
array([['0.9508676168436844', '0.5224962631347068', '0.0980194959983528',
        '0.14773756499383062', '0.047326701250847036',
        '0.30755559681674116', '0.9768350597172487',
        '0.6786990996560694', '0.8887922071440164',
        '0.46095020120760577', '0.40757888636394357',
        '0.05590193376889152', '0.9620196047369051',
        '0.17129426079440946', '0.8595396107931239',
        '0.39301712469224603', '0.4688875383003581',
        '0.16510888121203782', '0.2022481902204376',
        '0.5575269531863064']], dtype='<U32')
```

Mathematical Operations

In [64]:
```
a = np.array([23,57,75,82,35])
b = np.array([4,6,8,9,7])
```

In [65]:
```
a + b
```

Out[65]:
```
array([27, 63, 83, 91, 42])
```

In [66]:
```
#add
np.add(a,b)
```

Out[66]:
```
array([27, 63, 83, 91, 42])
```

In [67]:
```
#subtract
np.subtract(a,b)
```

Out[67]:
```
array([19, 51, 67, 73, 28])
```

In [68]:
```
#multiply
np.multiply(a,b)
```

Out[68]:
```
array([ 92, 342, 600, 738, 245])
```

In [69]:
```
#divide
np.divide(a,b)
```

Out[69]:
```
array([5.75      , 9.5       , 9.375     , 9.11111111, 5.        ])
```

In [70]:
```
#floor_divide
np.floor_divide(a,b)
```

Out[70]:
```
array([5, 9, 9, 9, 5])
```

In [71]:
```
#mod - modulo
np.mod(a,b)
```

Out[71]:
```
array([3, 3, 3, 1, 0])
```

In [72]:
```
#divmod -
np.divmod(a,b)
```

Out[72]:
```
(array([5, 9, 9, 9, 5]), array([3, 3, 3, 1, 0]))
```

In [73]:
```
#Log
np.log(a)
```

Out[73]:
```
array([3.13549422, 4.04305127, 4.31748811, 4.40671925, 3.55534806])
```

In [74]:
```
np.log10(a)
```

Out[74]:
```
array([1.36172784, 1.75587486, 1.87506126, 1.91381385, 1.54406804])
```

In [76]:
```
#sqrt - Square Root
np.sqrt(a)
```

Out[76]:
```
array([4.79583152, 7.54983444, 8.66025404, 9.05538514, 5.91607978])
```

In [77]:
```
#cbrt - Cube Root
np.cbrt(27)
```

Out[77]:
```
3.0
```

In [78]:
```
np.cbrt(a)
```

Out[78]:
```
array([2.84386698, 3.84850113, 4.21716333, 4.34448149, 3.27106631])
```

In [82]:
```
#power
np.power(a,2)
```

Out[82]:
```
array([ 529, 3249, 5625, 6724, 1225], dtype=int32)
```

In [84]:
```
#Trigonometry
#sin
np.sin(90)  #in radians
```

Out[84]:
```
0.8939966636005579
```

In [85]:
```
np.sin(90 * (np.pi/180))  #in degrees
```

Out[85]:
```
1.0
```

In [86]:
```
np.sin(a)
```

Out[86]:
```
array([-0.8462204 ,  0.43616476, -0.38778164,  0.31322878, -0.42818267])
```

In [87]:
```
#cos
np.cos(a)
```

Out[87]:
```
array([-0.53283302,  0.89986683,  0.92175127,  0.9496777 , -0.90369221])
```

In [88]:
```
#tan
np.tan(a)
```

Out[88]:
```
array([ 1.58815308,  0.48469923, -0.42070095,  0.32982641,  0.47381472])
```

In [89]:
```
#cot
1/np.tan(a)
```

Out[89]:
```
array([ 0.62966222,  2.06313513, -2.37698536,  3.03189793,  2.11052962])
```

Statiscal Functions

In [90]:
```
a
```

Out[90]:
```
array([23, 57, 75, 82, 35])
```

In [91]:
```
#max
np.max(a)
```

Out[91]:
```
82
```

In [92]:
```
#min
np.min(a)
```

Out[92]:
```
23
```

In [93]:
```
#argmax
np.argmax(a)
```

Out[93]:
```
3
```

In [94]:
```
#argmin
np.argmin(a)
```

Out[94]:
```
0
```

In [95]:
```
#sort
np.sort(a) #ascending order
```

Out[95]:
```
array([23, 35, 57, 75, 82])
```

In [96]:
```
#argsort
np.argsort(a)
```

Out[96]:
```
array([0, 4, 1, 2, 3], dtype=int64)
```

In [97]:
```
a
```

Out[97]:
```
array([23, 57, 75, 82, 35])
```

In [98]:
```
#flip
np.flip(a)
```

Out[98]:
```
array([35, 82, 75, 57, 23])
```

In [101…
```
np.flip(np.sort(a)) #descending order
```

Out[101…
```
array([82, 75, 57, 35, 23])
```

In [102…
```
#mean - average
np.mean(a)
```

Out[102… 54.4

In [103…
```
#median
np.median(a)
```

Out[103… 57.0

In [104…
```
#std - Standard Deviation
np.std(a)
```

Out[104… 22.60619384151167

In [105…
```
#var - Variance
np.var(a)
```

Out[105… 511.0400000000001

In [106…
```
b
```

Out[106… array([4, 6, 8, 9, 7])

In [107…
```
#cumsum - Cumulative Summation
np.cumsum(b)
```

Out[107… array([4, 10, 18, 27, 34])

In [108…
```
#cumprod - Cumulative Product
np.cumprod(b)
```

Out[108… array([4, 24, 192, 1728, 12096])

In [109…
```
np.cumproduct(b)
```

Out[109… array([4, 24, 192, 1728, 12096])

In [112…
```
#Round Off Numbers
#round
np.round(25.6)
```

Out[112… 26.0

In [113…
```
#ceil
np.ceil(25.3)
```

Out[113… 26.0

In [114…
```
#trunc
np.trunc(25.7)
```

Out[114… 25.0

In [115…
```
#repeat
a = np.array([1,2,3])
np.repeat(a,4)
```

Out[115… array([1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3])

In [116…
```
#tile
a = np.array([1,2,3])
np.tile(a,4)
```

Out[116…
```
array([1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])
```

In [117…
```
#where
a = np.array([1,2,4,3,4,4,5,4,6,4])
np.where(a==4)
```

Out[117…
```
(array([2, 4, 5, 7, 9], dtype=int64),)
```

Vector Maths

In [118…
```
a = np.array([[1,3],[2,4]])
b = np.array([[5,7],[8,9]])

print(a)
print(b)
```

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

In [119…
```
#dot
np.dot(a,b)
```

Out[119…
```
array([[29, 34],
       [42, 50]])
```

In [122…
```
a@b  #dot product
```

Out[122…
```
array([[29, 34],
       [42, 50]])
```

In [120…
```
#cross
np.cross(a,b)
```

Out[120…
```
array([ -8, -14])
```

In [123…
```
#transpose
np.transpose(a)
```

Out[123…
```
array([[1, 2],
       [3, 4]])
```

In [124…
```
a.T
```

Out[124…
```
array([[1, 2],
       [3, 4]])
```

In [1]:
```
import numpy as np
```

In [2]:
```
#Indexing and Slicing in 1D Array
a = np.array([34,89,55,21,35])
a
```

Out[2]:
```
array([34, 89, 55, 21, 35])
```

In [3]:
```
#indexing
a[2]
```

Out[3]:
```
55
```

In [4]:
```
a[-3]
```

Out[4]:
```
55
```

In [5]:
```
#Slicing
a[0:5:2]
```

Out[5]:
```
array([34, 55, 35])
```

In [6]:
```
a[::2]
```

Out[6]:
```
array([34, 55, 35])
```

In [7]:
```
a[::-2]
```

Out[7]:
```
array([35, 55, 34])
```

In [8]:
```
#Indexing and Slicing in 2D Array
b = np.random.randint(1,20,(5,5))
b
```

Out[8]:
```
array([[ 3, 14, 12, 14,  5],
       [13, 19, 17, 18, 11],
       [10, 14,  5, 12, 18],
       [ 6, 10, 12, 15, 12],
       [19,  7,  5, 10,  7]])
```

In [9]:
```
#Indexing
#Syntax - var[row_index,col_index]
b[2,3]
```

Out[9]:
```
12
```

In [10]:
```
b[2][3]
```

Out[10]:
```
12
```

In [12]:
```
b[3]  #bydefault row
```

Out[12]:
```
array([ 6, 10, 12, 15, 12])
```

In [13]:
```
#Slicing
b[0:4:1,0:3:1]
```

Out[13]:
```
array([[ 3, 14, 12],
       [13, 19, 17],
       [10, 14,  5],
       [ 6, 10, 12]])
```

In [14]:
```
b[1:4:1,1:4:1]
```

Out[14]:
```
array([[19, 17, 18],
       [14,  5, 12],
       [10, 12, 15]])
```

In [17]:
```
#all rows and even columns (0,2,4)
b[0:5:1,0:5:2]
```

Out[17]:
```
array([[ 3, 12,  5],
       [13, 17, 11],
       [10,  5, 18],
       [ 6, 12, 12],
       [19,  5,  7]])
```

In [18]:
```
b[::,::2]
```

Out[18]:
```
array([[ 3, 12,  5],
       [13, 17, 11],
       [10,  5, 18],
       [ 6, 12, 12],
       [19,  5,  7]])
```

In [19]:
```
#odd rows and odd columns (1,3)
b[1:5:2,1:5:2]
```

Out[19]:
```
array([[19, 18],
       [10, 15]])
```

In [20]:
```
b[1::2,1::2]
```

Out[20]:
```
array([[19, 18],
       [10, 15]])
```

In [21]:
```
#Concatenation - joining matrix
a = np.random.randint(1,20,(2,2))
a
```

Out[21]:
```
array([[13, 15],
       [13,  3]])
```

In [22]:
```
b = np.random.randint(1,20,(2,4))
b
```

Out[22]:
```
array([[13,  9,  5,  3],
       [ 8, 11,  3,  6]])
```

In [23]:
```
c = np.random.randint(1,20,(4,4))
c
```

Out[23]:
```
array([[ 2, 11,  7,  7],
       [18, 18,  7,  1],
       [ 1,  4, 18, 17],
       [18,  4,  2,  3]])
```

In [28]:
```
#Column Wise (axis=1)
np.concatenate((a,b),axis=1)
```

Out[28]:
```
array([[13, 15, 13,  9,  5,  3],
       [13,  3,  8, 11,  3,  6]])
```

In [29]:
```
#Row Wise (axis=0) (by default)
np.concatenate((b,c),axis=0)
```

Out[29]:
```
array([[13,  9,  5,  3],
       [ 8, 11,  3,  6],
       [ 2, 11,  7,  7],
       [18, 18,  7,  1],
       [ 1,  4, 18, 17],
       [18,  4,  2,  3]])
```

In [36]:
```
#Stack - joining matrix
#hstack - Horizontal Stack
np.hstack((a,b))
```

Out[36]:
```
array([[13, 15, 13,  9,  5,  3],
       [13,  3,  8, 11,  3,  6]])
```

In [37]:
```
#vstack - Vertical Stack
np.vstack((b,c))
```

Out[37]:
```
array([[13,  9,  5,  3],
       [ 8, 11,  3,  6],
       [ 2, 11,  7,  7],
       [18, 18,  7,  1],
       [ 1,  4, 18, 17],
       [18,  4,  2,  3]])
```

In [40]:
```
#stack
a = np.array([1,2,3])
b = np.array([4,5,6])

np.stack((a,b),axis=1)
```

Out[40]:
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [41]:
```
np.stack((a,b),axis=0)
```

Out[41]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

Pandas 🐼

Pandas is a Python's library which is use to analyze and manipulate the the data.

It is also use to read and create datasets.

Pandas is of two types :-
--> Series (single column)
--> DataFrame (multiple columns)

In [ ]:
```
!pip install pandas
```

In [1]:
```
import pandas as pd
```

In [2]:
```
#Creating Series
s = pd.Series(['a','b','c','d','e'])
s
```

Out[2]:
```
0    a
1    b
2    c
3    d
4    e
dtype: object
```

In [3]:
```
type(s)
```

Out[3]:
```
pandas.core.series.Series
```

In [4]:
```
s = pd.Series(['a','b','c','d','e'],index=[10,20,20,40,50])
s
```

Out[4]:
```
10    a
20    b
20    c
40    d
50    e
dtype: object
```

In [5]:
```
#indexing
s[3]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File C:\anaconda\Lib\site-packages\pandas\core\indexes\base.py:3805, in Index.get
_loc(self, key)
   3804 try:
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:191, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:228, in pandas._libs.index.IndexEngine._get_loc_duplicates()

KeyError: 3

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[5], line 2
      1 #indexing
----> 2 s[3]

File C:\anaconda\Lib\site-packages\pandas\core\series.py:1121, in Series.__getite
m__(self, key)
   1118     return self._values[key]
   1120 elif key_is_scalar:
-> 1121     return self._get_value(key)
   1123 # Convert generator to list before going through hashable part
   1124 # (We will iterate through the generator there to check for slices)
   1125 if is_iterator(key):

File C:\anaconda\Lib\site-packages\pandas\core\series.py:1237, in Series._get_val
ue(self, label, takeable)
   1234     return self._values[label]
   1236 # Similar to Index.get_value, but we do not fall back to positional
-> 1237 loc = self.index.get_loc(label)
   1239 if is_integer(loc):
   1240     return self._values[loc]

File C:\anaconda\Lib\site-packages\pandas\core\indexes\base.py:3812, in Index.get
_loc(self, key)
   3807     if isinstance(casted_key, slice) or (
   3808         isinstance(casted_key, abc.Iterable)
   3809         and any(isinstance(x, slice) for x in casted_key)
   3810     ):
   3811         raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
   3813 except TypeError:
   3814     # If we have a listlike key, _check_indexing_error will raise
   3815     #  InvalidIndexError. Otherwise we fall through and re-raise
   3816     #  the TypeError.
   3817     self._check_indexing_error(key)

KeyError: 3
```

In [6]:
```
s[40]
```

Out[6]:
```
'd'
```

In [7]:
```
s[20]
```

Out[7]:
```
20    b
20    c
dtype: object
```

In [8]:
```
#slicing
s[20:50:20]
```

Out[8]:
```
Series([], dtype: object)
```

In [9]:
```
s[1::2]
```

Out[9]:
```
20    b
40    d
dtype: object
```

In [10]:
```
s[::-1]
```

Out[10]:
```
50    e
40    d
20    c
20    b
10    a
dtype: object
```

In [11]:
```
s = pd.Series(['a','b','c','d','e'],index=range(5,10))
s
```

Out[11]:
```
5    a
6    b
7    c
8    d
9    e
dtype: object
```

In [12]:
```
#Tuple to Series
t = pd.Series((100,200,300,400))
t
```

Out[12]:
```
0    100
1    200
2    300
3    400
dtype: int64
```

In [13]:
```
#indexing
t[2]
```

Out[13]:
```
300
```

In [14]:
```
t[-2]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
File C:\anaconda\Lib\site-packages\pandas\core\indexes\range.py:413, in RangeInde
x.get_loc(self, key)
    412 try:
--> 413     return self._range.index(new_key)
    414 except ValueError as err:

ValueError: -2 is not in range

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[14], line 1
----> 1 t[-2]

File C:\anaconda\Lib\site-packages\pandas\core\series.py:1121, in Series.__getite
m__(self, key)
   1118     return self._values[key]
   1120 elif key_is_scalar:
-> 1121     return self._get_value(key)
   1123 # Convert generator to list before going through hashable part
   1124 # (We will iterate through the generator there to check for slices)
   1125 if is_iterator(key):

File C:\anaconda\Lib\site-packages\pandas\core\series.py:1237, in Series._get_val
ue(self, label, takeable)
   1234     return self._values[label]
   1236 # Similar to Index.get_value, but we do not fall back to positional
-> 1237 loc = self.index.get_loc(label)
   1239 if is_integer(loc):
   1240     return self._values[loc]

File C:\anaconda\Lib\site-packages\pandas\core\indexes\range.py:415, in RangeInde
x.get_loc(self, key)
    413         return self._range.index(new_key)
    414     except ValueError as err:
--> 415         raise KeyError(key) from err
    416 if isinstance(key, Hashable):
    417     raise KeyError(key)

KeyError: -2
```

In [15]:
```
#slicing
t[::1]
```

Out[15]:
```
0    100
1    200
2    300
3    400
dtype: int64
```

In [16]:
```
#Set to Series
s = pd.Series({35,35,35,10,10,25})
s
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[16], line 2
      1 #Set to Series
----> 2 s = pd.Series({35,35,35,10,10,25})
      3 s

File C:\anaconda\Lib\site-packages\pandas\core\series.py:584, in Series.__init__
(self, data, index, dtype, name, copy, fastpath)
    582         data = data.copy()
    583 else:
--> 584     data = sanitize_array(data, index, dtype, copy)
    586     manager = _get_option("mode.data_manager", silent=True)
    587     if manager == "block":

File C:\anaconda\Lib\site-packages\pandas\core\construction.py:642, in sanitize_a
rray(data, index, dtype, copy, allow_2d)
    633     return sanitize_array(
    634         data,
    635         index=index,
   (...)
    638         allow_2d=allow_2d,
    639     )
    641 else:
--> 642     _sanitize_non_ordered(data)
    643     # materialize e.g. generators, convert e.g. tuples, abc.ValueView
    644     data = list(data)

File C:\anaconda\Lib\site-packages\pandas\core\construction.py:693, in _sanitize_
non_ordered(data)
    689 """
    690 Raise only for unordered sets, e.g., not for dict_keys
    691 """
    692 if isinstance(data, (set, frozenset)):
--> 693     raise TypeError(f"'{type(data).__name__}' type is unordered")

TypeError: 'set' type is unordered
```

In [17]:
```
#Dictionary to Series
d = pd.Series({'a':1,'b':2,'c':3,'d':4})
d
```

Out[17]:
```
a    1
b    2
c    3
d    4
dtype: int64
```

In [18]:
```
d1 = pd.Series({'a':1,'b':2,'c':3,'d':4},index=['e','a','g','b'])
d1
```

Out[18]:
```
e    NaN
a    1.0
g    NaN
b    2.0
dtype: float64
```

In [19]:
```
d
```

Out[19]:
```
a    1
b    2
c    3
d    4
dtype: int64
```

In [20]:
```
#indexing
d[1]
```

```
C:\Users\VICTUS\AppData\Local\Temp\ipykernel_19392\719679795.py:2: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  d[1]
```

Out[20]:
```
2
```

In [21]:
```
d['b']
```

Out[21]:
```
2
```

In [22]:
```
#Slicing
d[1:4:2]
```

Out[22]:
```
b    2
d    4
dtype: int64
```

In [23]:
```
d['b':'d':2]
```

Out[23]:
```
b    2
d    4
dtype: int64
```

In [25]:
```
import pandas as pd
```

In [52]:
```
#Creating DataFrame
d = {'Name':['Vicky','Thalla','Sunny','Chiku','Babu Rao'],
    'Roll':[111,222,333,444,555],
    'Skills':['CR','Cricket','Dettol','Chef','Chor']}
d
```

Out[52]:
```
{'Name': ['Vicky', 'Thalla', 'Sunny', 'Chiku', 'Babu Rao'],
 'Roll': [111, 222, 333, 444, 555],
 'Skills': ['CR', 'Cricket', 'Dettol', 'Chef', 'Chor']}
```

In [53]:
```
df = pd.DataFrame(d)
df
```

Out[53]:

	Name	Roll	Skills
0	Vicky	111	CR
1	Thalla	222	Cricket
2	Sunny	333	Dettol
3	Chiku	444	Chef
4	Babu Rao	555	Chor

In [54]:
```
type(df)
```

Out[54]:
```
pandas.core.frame.DataFrame
```

In [55]:
```
#Indexing
df.Name[2]
```

Out[55]:
```
'Sunny'
```

In [56]:
```
df.Skills[4]
```

Out[56]:
```
'Chor'
```

In [57]:
```
#Slicing
#iloc - used for slicing
df.iloc[::,::]
```

Out[57]:

	Name	Roll	Skills
0	Vicky	111	CR
1	Thalla	222	Cricket
2	Sunny	333	Dettol
3	Chiku	444	Chef
4	Babu Rao	555	Chor

In [58]:
```
df.iloc[3,::]
```

Out[58]:
```
Name      Chiku
Roll        444
Skills     Chef
Name: 3, dtype: object
```

In [59]:
```
df.iloc[::,1]
```

Out[59]:
```
0    111
1    222
2    333
3    444
4    555
Name: Roll, dtype: int64
```

In [60]:
```
df.iloc[::2,::]
```

Out[60]:

	Name	Roll	Skills
0	Vicky	111	CR
2	Sunny	333	Dettol
4	Babu Rao	555	Chor

In [61]:
```
df.iloc[1:5:2,::2]
```

Out[61]:

	Name	Skills
1	Thalla	Cricket
3	Chiku	Chef

In [62]:
```
df.iloc[::-1,::-1]
```

Out[62]:

	Skills	Roll	Name
4	Chor	555	Babu Rao
3	Chef	444	Chiku
2	Dettol	333	Sunny
1	Cricket	222	Thalla
0	CR	111	Vicky

In [63]:
```
#rename
df = df.rename(columns={'Name':'Stu_Name'})
df
```

Out[63]:

	Stu_Name	Roll	Skills
0	Vicky	111	CR
1	Thalla	222	Cricket
2	Sunny	333	Dettol
3	Chiku	444	Chef
4	Babu Rao	555	Chor

In [64]:
```
#head - it represents top rows (bydefault 5)
df.head(3)
```

Out[64]:

	Stu_Name	Roll	Skills
0	Vicky	111	CR
1	Thalla	222	Cricket
2	Sunny	333	Dettol

In [65]:
```
df.head()
```

Out[65]:

	Stu_Name	Roll	Skills
0	Vicky	111	CR
1	Thalla	222	Cricket
2	Sunny	333	Dettol
3	Chiku	444	Chef
4	Babu Rao	555	Chor

In [66]:
```
#tail - it represents bottom rows (bydefault 5)
df.tail(2)
```

Out[66]:

	Stu_Name	Roll	Skills
3	Chiku	444	Chef
4	Babu Rao	555	Chor

In [67]:
```
#sample
df.sample()
```

Out[67]:

	Stu_Name	Roll	Skills
2	Sunny	333	Dettol

In [68]:
```
df.sample(2)
```

Out[68]:

	Stu_Name	Roll	Skills
2	Sunny	333	Dettol
4	Babu Rao	555	Chor

In [69]:
```
#set_index
df.set_index("Roll",inplace=True)
df
```

Out[69]:

	Stu_Name	Skills
Roll		
111	Vicky	CR
222	Thalla	Cricket
333	Sunny	Dettol
444	Chiku	Chef
555	Babu Rao	Chor

In [70]:
```
df.Stu_Name[444]
```

Out[70]:
```
'Chiku'
```

In [73]:
```
import numpy as np
```

In [81]:
```
d = pd.DataFrame({'Days':[1,2,3,4,5],
                  'Places':['Sikkim',np.nan,'Puri','Pune','Ranchi'],
                  'Visitors':[5000,4859,5983,4000,5200]})
d
```

Out[81]:

	Days	Places	Visitors
0	1	Sikkim	5000
1	2	NaN	4859
2	3	Puri	5983
3	4	Pune	4000
4	5	Ranchi	5200

In [82]:
```
#to check missing values
#isnull
d.isnull()
```

Out[82]:

	Days	Places	Visitors
0	False	False	False
1	False	True	False
2	False	False	False
3	False	False	False
4	False	False	False

In [83]:
```
d.isnull().sum()
```

Out[83]:
```
Days        0
Places      1
Visitors    0
dtype: int64
```

In [85]:
```
#fillna
d.Places.fillna("Puri",inplace=True)
```

In [86]:
```
d
```

Out[86]:

	Days	Places	Visitors
0	1	Sikkim	5000
1	2	Puri	4859
2	3	Puri	5983
3	4	Pune	4000
4	5	Ranchi	5200

In [87]:
```
#to check datatypes
d.dtypes
```

Out[87]:
```
Days         int64
Places      object
Visitors     int64
dtype: object
```

In [90]:
```
#info
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Days      5 non-null      int64 
 1   Places    5 non-null      object
 2   Visitors  5 non-null      int64 
dtypes: int64(2), object(1)
memory usage: 252.0+ bytes
```

In [91]:
```
d.shape
```

Out[91]:
```
(5, 3)
```

In [92]:
```
d.ndim
```

Out[92]:
```
2
```

In [93]:
```
d.Places.unique()
```

Out[93]:
```
array(['Sikkim', 'Puri', 'Pune', 'Ranchi'], dtype=object)
```

In [95]:
```
d.Visitors.mean()
```

Out[95]:
```
5008.4
```

In [96]:
```
d.Visitors.max()
```

Out[96]:
```
5983
```

In [97]:
```
d.Visitors.min()
```

Out[97]:
```
4000
```

In [98]:
```
#describe
d.describe()
```

Out[98]:

	Days	Visitors
count	5.000000	5.000000
mean	3.000000	5008.400000
std	1.581139	711.656026
min	1.000000	4000.000000
25%	2.000000	4859.000000
50%	3.000000	5000.000000
75%	4.000000	5200.000000
max	5.000000	5983.000000

In [99]:
```
d.describe(include='all')
```

Out[99]:

	Days	Places	Visitors
count	5.000000	5	5.000000
unique	NaN	4	NaN
top	NaN	Puri	NaN
freq	NaN	2	NaN
mean	3.000000	NaN	5008.400000
std	1.581139	NaN	711.656026
min	1.000000	NaN	4000.000000
25%	2.000000	NaN	4859.000000
50%	3.000000	NaN	5000.000000
75%	4.000000	NaN	5200.000000
max	5.000000	NaN	5983.000000

In [100…
```
d.describe(include='O')
```

Out[100…

	Places
count	5
unique	4
top	Puri
freq	2

Iris Dataset

In [101…
```
#Importing Packages
import pandas as pd
import numpy as np
```

In [108…
```
#Reading a Dataset
iris = pd.read_csv(r"C:\Users\VICTUS\Downloads\iris\iris.data",header=None)
iris
```

Out[108…

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
…	…	…	…	…	…
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [109…
```
#Rename the columns
iris.columns = ['SL','SW','PL','PW','Flower Type']
iris
```

Out[109…

	SL	SW	PL	PW	Flower Type
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [110…
```
iris.head(10)
```

Out[110…

	SL	SW	PL	PW	Flower Type
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

In [111…
```
iris.head()
```

Out[111…

	SL	SW	PL	PW	Flower Type
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [112…
```
iris.tail()
```

Out[112…

	SL	SW	PL	PW	Flower Type
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

In [113…
```
iris.sample()
```

Out[113…

	SL	SW	PL	PW	Flower Type
137	6.4	3.1	5.5	1.8	Iris-virginica

In [114…
```
iris.sample(6)
```

Out[114…

	SL	SW	PL	PW	Flower Type
123	6.3	2.7	4.9	1.8	Iris-virginica
22	4.6	3.6	1.0	0.2	Iris-setosa
26	5.0	3.4	1.6	0.4	Iris-setosa
108	6.7	2.5	5.8	1.8	Iris-virginica
74	6.4	2.9	4.3	1.3	Iris-versicolor
49	5.0	3.3	1.4	0.2	Iris-setosa

In [116…
```
iris.isnull().sum()
```

Out[116…
```
SL             0
SW             0
PL             0
PW             0
Flower Type    0
dtype: int64
```

In [117…
```
iris.dtypes
```

Out[117…
```
SL             float64
SW             float64
PL             float64
PW             float64
Flower Type     object
dtype: object
```

In [119…
```
iris.SL.dtypes
```

Out[119…
```
dtype('float64')
```

In [120…
```
iris.Flower Type.dtype
```

```
  Cell In[120], line 1
    iris.Flower Type.dtype
                ^
SyntaxError: invalid syntax
```

In [121…
```
iris["Flower Type"].dtype
```

Out[121…
```
dtype('O')
```

In [122…
```
iris["SW"].dtypes
```

Out[122…
```
dtype('float64')
```

In [123…
```
#info
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   SL           150 non-null    float64
 1   SW           150 non-null    float64
 2   PL           150 non-null    float64
 3   PW           150 non-null    float64
 4   Flower Type  150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [124…
```
iris.SL.unique()
```

Out[124…
```
array([5.1, 4.9, 4.7, 4.6, 5. , 5.4, 4.4, 4.8, 4.3, 5.8, 5.7, 5.2, 5.5,
       4.5, 5.3, 7. , 6.4, 6.9, 6.5, 6.3, 6.6, 5.9, 6. , 6.1, 5.6, 6.7,
       6.2, 6.8, 7.1, 7.6, 7.3, 7.2, 7.7, 7.4, 7.9])
```

In [125…
```
iris['Flower Type'].unique()
```

Out[125…
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

In [126…
```
iris.columns
```

Out[126…
```
Index(['SL', 'SW', 'PL', 'PW', 'Flower Type'], dtype='object')
```

In [131…
```
for i in iris.columns:
    print(i,':','\n',iris[i].unique(),'\n')
```

```
SL :
 [5.1 4.9 4.7 4.6 5.  5.4 4.4 4.8 4.3 5.8 5.7 5.2 5.5 4.5 5.3 7.  6.4 6.9
 6.5 6.3 6.6 5.9 6.  6.1 5.6 6.7 6.2 6.8 7.1 7.6 7.3 7.2 7.7 7.4 7.9]

SW :
 [3.5 3.  3.2 3.1 3.6 3.9 3.4 2.9 3.7 4.  4.4 3.8 3.3 4.1 4.2 2.3 2.8 2.4
 2.7 2.  2.2 2.5 2.6]

PL :
 [1.4 1.3 1.5 1.7 1.6 1.1 1.2 1.  1.9 4.7 4.5 4.9 4.  4.6 3.3 3.9 3.5 4.2
 3.6 4.4 4.1 4.8 4.3 5.  3.8 3.7 5.1 3.  6.  5.9 5.6 5.8 6.6 6.3 6.1 5.3
 5.5 6.7 6.9 5.7 6.4 5.4 5.2]

PW :
 [0.2 0.4 0.3 0.1 0.5 0.6 1.4 1.5 1.3 1.6 1.  1.1 1.8 1.2 1.7 2.5 1.9 2.1
 2.2 2.  2.4 2.3]

Flower Type :
 ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

In [132…
```
iris.describe()
```

Out[132…

	SL	SW	PL	PW
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [133…
```
iris.describe(include='all')
```

Out[133...

	SL	SW	PL	PW	Flower Type
count	150.000000	150.000000	150.000000	150.000000	150
unique	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	Iris-setosa
freq	NaN	NaN	NaN	NaN	50
mean	5.843333	3.054000	3.758667	1.198667	NaN
std	0.828066	0.433594	1.764420	0.763161	NaN
min	4.300000	2.000000	1.000000	0.100000	NaN
25%	5.100000	2.800000	1.600000	0.300000	NaN
50%	5.800000	3.000000	4.350000	1.300000	NaN
75%	6.400000	3.300000	5.100000	1.800000	NaN
max	7.900000	4.400000	6.900000	2.500000	NaN

In [134...
```
iris.describe(include='O')
```

Out[134...

	Flower Type
count	150
unique	3
top	Iris-setosa
freq	50

In [135...
```
iris['Flower Type'].value_counts()
```

Out[135...
```
Flower Type
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

In [136...
```
iris.SL.value_counts()
```

Out[136…

```
SL
5.0    10
5.1     9
6.3     9
5.7     8
6.7     8
5.8     7
5.5     7
6.4     7
4.9     6
5.4     6
6.1     6
6.0     6
5.6     6
4.8     5
6.5     5
6.2     4
7.7     4
6.9     4
4.6     4
5.2     4
5.9     3
4.4     3
7.2     3
6.8     3
6.6     2
4.7     2
7.6     1
7.4     1
7.3     1
7.0     1
7.1     1
5.3     1
4.3     1
4.5     1
7.9     1
Name: count, dtype: int64
```

In [139…

```
#to drop a column
iris.drop('Flower Type',axis=1)
```

Out[139…

	SL	SW	PL	PW
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
…	…	…	…	…
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Auto MPG

In [140…

```
import numpy as np
import pandas as pd
```

In [144…

```
auto = pd.read_csv(r"C:\Users\VICTUS\Downloads\auto+mpg\auto-mpg.data",header=No
auto
```

```
C:\Users\VICTUS\AppData\Local\Temp\ipykernel_19392\1660018015.py:1: FutureWarnin
g: The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be remove
d in a future version. Use ``sep='\s+'`` instead
  auto = pd.read_csv(r"C:\Users\VICTUS\Downloads\auto+mpg\auto-mpg.data",header=N
one,delim_whitespace=True)
```

Out[144...

	0	1	2	3	4	5	6	7	8
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.00	2790.0	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52.00	2130.0	24.6	82	2	vw pickup
395	32.0	4	135.0	84.00	2295.0	11.6	82	1	dodge rampage
396	28.0	4	120.0	79.00	2625.0	18.6	82	1	ford ranger
397	31.0	4	119.0	82.00	2720.0	19.4	82	1	chevy s-10

398 rows × 9 columns

In [145...

```
auto = pd.read_csv(r"C:\Users\VICTUS\Downloads\auto+mpg\auto-mpg.data",header=No
auto
```

```
<>:1: SyntaxWarning: invalid escape sequence '\s'
<>:1: SyntaxWarning: invalid escape sequence '\s'
C:\Users\VICTUS\AppData\Local\Temp\ipykernel_19392\551619158.py:1: SyntaxWarning:
invalid escape sequence '\s'
  auto = pd.read_csv(r"C:\Users\VICTUS\Downloads\auto+mpg\auto-mpg.data",header=N
one,sep='\s+')
```

Out[145...

	0	1	2	3	4	5	6	7	8
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.00	2790.0	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52.00	2130.0	24.6	82	2	vw pickup
395	32.0	4	135.0	84.00	2295.0	11.6	82	1	dodge rampage
396	28.0	4	120.0	79.00	2625.0	18.6	82	1	ford ranger
397	31.0	4	119.0	82.00	2720.0	19.4	82	1	chevy s-10

398 rows × 9 columns

In [146…
```
auto.columns = ['mpg','cylinders','displacement','horsepower',
                'weight','acceleration','model year','origin',
                'car name']
auto.head()
```

Out[146…

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	che ch r
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	s
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	ply s
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	re
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	

In [147…
```
#check missing values
auto.isnull().sum()
```

Out[147…
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

In [148…
```
#check data types
auto.dtypes
```

Out[148…
```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight          float64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

In [149…
```
#check unique values
for i in auto.columns:
    print(i,':','\n',auto[i].unique(),'\n')
```

```
mpg :
 [18.  15.  16.  17.  14.  24.  22.  21.  27.  26.  25.  10.  11.   9.
 28.  19.  12.  13.  23.  30.  31.  35.  20.  29.  32.  33.  17.5 15.5
 14.5 22.5 24.5 18.5 29.5 26.5 16.5 31.5 36.  25.5 33.5 20.5 30.5 21.5
 43.1 36.1 32.8 39.4 19.9 19.4 20.2 19.2 25.1 20.6 20.8 18.6 18.1 17.7
 27.5 27.2 30.9 21.1 23.2 23.8 23.9 20.3 21.6 16.2 19.8 22.3 17.6 18.2
 16.9 31.9 34.1 35.7 27.4 25.4 34.2 34.5 31.8 37.3 28.4 28.8 26.8 41.5
 38.1 32.1 37.2 26.4 24.3 19.1 34.3 29.8 31.3 37.  32.2 46.6 27.9 40.8
 44.3 43.4 36.4 44.6 40.9 33.8 32.7 23.7 23.6 32.4 26.6 25.8 23.5 39.1
 39.  35.1 32.3 37.7 34.7 34.4 29.9 33.7 32.9 31.6 28.1 30.7 24.2 22.4
 34.  38.  44. ]

cylinders :
 [8 4 6 3 5]

displacement :
 [307.  350.  318.  304.  302.  429.  454.  440.  455.  390.  383.  340.
 400.  113.  198.  199.  200.   97.  110.  107.  104.  121.  360.  140.
  98.  232.  225.  250.  351.  258.  122.  116.   79.   88.   71.   72.
  91.   97.5  70.  120.   96.  108.  155.   68.  114.  156.   76.   83.
  90.  231.  262.  134.  119.  171.  115.  101.  305.   85.  130.  168.
 111.  260.  151.  146.   80.   78.  105.  131.  163.   89.  267.   86.
 183.  141.  173.  135.   81.  100.  145.  112.  181.  144. ]

horsepower :
 ['130.0' '165.0' '150.0' '140.0' '198.0' '220.0' '215.0' '225.0' '190.0'
 '170.0' '160.0' '95.00' '97.00' '85.00' '88.00' '46.00' '87.00' '90.00'
 '113.0' '200.0' '210.0' '193.0' '?' '100.0' '105.0' '175.0' '153.0'
 '180.0' '110.0' '72.00' '86.00' '70.00' '76.00' '65.00' '69.00' '60.00'
 '80.00' '54.00' '208.0' '155.0' '112.0' '92.00' '145.0' '137.0' '158.0'
 '167.0' '94.00' '107.0' '230.0' '49.00' '75.00' '91.00' '122.0' '67.00'
 '83.00' '78.00' '52.00' '61.00' '93.00' '148.0' '129.0' '96.00' '71.00'
 '98.00' '115.0' '53.00' '81.00' '79.00' '120.0' '152.0' '102.0' '108.0'
 '68.00' '58.00' '149.0' '89.00' '63.00' '48.00' '66.00' '139.0' '103.0'
 '125.0' '133.0' '138.0' '135.0' '142.0' '77.00' '62.00' '132.0' '84.00'
 '64.00' '74.00' '116.0' '82.00']

weight :
 [3504. 3693. 3436. 3433. 3449. 4341. 4354. 4312. 4425. 3850. 3563. 3609.
 3761. 3086. 2372. 2833. 2774. 2587. 2130. 1835. 2672. 2430. 2375. 2234.
 2648. 4615. 4376. 4382. 4732. 2264. 2228. 2046. 2634. 3439. 3329. 3302.
 3288. 4209. 4464. 4154. 4096. 4955. 4746. 5140. 2962. 2408. 3282. 3139.
 2220. 2123. 2074. 2065. 1773. 1613. 1834. 1955. 2278. 2126. 2254. 2226.
 4274. 4385. 4135. 4129. 3672. 4633. 4502. 4456. 4422. 2330. 3892. 4098.
 4294. 4077. 2933. 2511. 2979. 2189. 2395. 2288. 2506. 2164. 2100. 4100.
 3988. 4042. 3777. 4952. 4363. 4237. 4735. 4951. 3821. 3121. 3278. 2945.
 3021. 2904. 1950. 4997. 4906. 4654. 4499. 2789. 2279. 2401. 2379. 2124.
 2310. 2472. 2265. 4082. 4278. 1867. 2158. 2582. 2868. 3399. 2660. 2807.
 3664. 3102. 2875. 2901. 3336. 2451. 1836. 2542. 3781. 3632. 3613. 4141.
 4699. 4457. 4638. 4257. 2219. 1963. 2300. 1649. 2003. 2125. 2108. 2246.
 2489. 2391. 2000. 3264. 3459. 3432. 3158. 4668. 4440. 4498. 4657. 3907.
 3897. 3730. 3785. 3039. 3221. 3169. 2171. 2639. 2914. 2592. 2702. 2223.
 2545. 2984. 1937. 3211. 2694. 2957. 2671. 1795. 2464. 2572. 2255. 2202.
 4215. 4190. 3962. 3233. 3353. 3012. 3085. 2035. 3651. 3574. 3645. 3193.
 1825. 1990. 2155. 2565. 3150. 3940. 3270. 2930. 3820. 4380. 4055. 3870.
 3755. 2045. 1945. 3880. 4060. 4140. 4295. 3520. 3425. 3630. 3525. 4220.
 4165. 4325. 4335. 1940. 2740. 2755. 2051. 2075. 1985. 2190. 2815. 2600.
 2720. 1800. 2070. 3365. 3735. 3570. 3535. 3155. 2965. 3430. 3210. 3380.
 3070. 3620. 3410. 3445. 3205. 4080. 2560. 2230. 2515. 2745. 2855. 2405.
 2830. 3140. 2795. 2135. 3245. 2990. 2890. 3265. 3360. 3840. 3725. 3955.
```

```
 3830. 4360. 4054. 3605. 1925. 1975. 1915. 2670. 3530. 3900. 3190. 3420.
 2200. 2150. 2020. 2595. 2700. 2556. 2144. 1968. 2120. 2019. 2678. 2870.
 3003. 3381. 2188. 2711. 2434. 2110. 2800. 2085. 2335. 2950. 3250. 1850.
 2145. 1845. 2910. 2420. 2500. 2905. 2290. 2490. 2635. 2620. 2725. 2385.
 1755. 1875. 1760. 2050. 2215. 2380. 2320. 2210. 2350. 2615. 3230. 3160.
 2900. 3415. 3060. 3465. 2605. 2640. 2575. 2525. 2735. 2865. 3035. 1980.
 2025. 1970. 2160. 2205. 2245. 1965. 1995. 3015. 2585. 2835. 2665. 2370.
 2790. 2295. 2625.]

acceleration :
 [12.  11.5 11.  10.5 10.   9.   8.5  8.   9.5 15.  15.5 16.  14.5 20.5
 17.5 12.5 14.  13.5 18.5 19.  13.  19.5 18.  17.  23.5 16.5 21.  16.9
 14.9 17.7 15.3 13.9 12.8 15.4 17.6 22.2 22.1 14.2 17.4 16.2 17.8 12.2
 16.4 13.6 15.7 13.2 21.9 16.7 12.1 14.8 18.6 16.8 13.7 11.1 11.4 18.2
 15.8 15.9 14.1 21.5 14.4 19.4 19.2 17.2 18.7 15.1 13.4 11.2 14.7 16.6
 17.3 15.2 14.3 20.1 24.8 11.3 12.9 18.8 18.1 17.9 21.7 23.7 19.9 21.8
 13.8 12.6 16.1 20.7 18.3 20.4 19.6 17.1 15.6 24.6 11.6]

model year :
 [70 71 72 73 74 75 76 77 78 79 80 81 82]

origin :
 [1 3 2]

car name :
 ['chevrolet chevelle malibu' 'buick skylark 320' 'plymouth satellite'
 'amc rebel sst' 'ford torino' 'ford galaxie 500' 'chevrolet impala'
 'plymouth fury iii' 'pontiac catalina' 'amc ambassador dpl'
 'dodge challenger se' "plymouth 'cuda 340" 'chevrolet monte carlo'
 'buick estate wagon (sw)' 'toyota corona mark ii' 'plymouth duster'
 'amc hornet' 'ford maverick' 'datsun pl510'
 'volkswagen 1131 deluxe sedan' 'peugeot 504' 'audi 100 ls' 'saab 99e'
 'bmw 2002' 'amc gremlin' 'ford f250' 'chevy c20' 'dodge d200' 'hi 1200d'
 'chevrolet vega 2300' 'toyota corona' 'ford pinto'
 'plymouth satellite custom' 'ford torino 500' 'amc matador'
 'pontiac catalina brougham' 'dodge monaco (sw)'
 'ford country squire (sw)' 'pontiac safari (sw)'
 'amc hornet sportabout (sw)' 'chevrolet vega (sw)' 'pontiac firebird'
 'ford mustang' 'mercury capri 2000' 'opel 1900' 'peugeot 304' 'fiat 124b'
 'toyota corolla 1200' 'datsun 1200' 'volkswagen model 111'
 'plymouth cricket' 'toyota corona hardtop' 'dodge colt hardtop'
 'volkswagen type 3' 'chevrolet vega' 'ford pinto runabout'
 'amc ambassador sst' 'mercury marquis' 'buick lesabre custom'
 'oldsmobile delta 88 royale' 'chrysler newport royal' 'mazda rx2 coupe'
 'amc matador (sw)' 'chevrolet chevelle concours (sw)'
 'ford gran torino (sw)' 'plymouth satellite custom (sw)'
 'volvo 145e (sw)' 'volkswagen 411 (sw)' 'peugeot 504 (sw)'
 'renault 12 (sw)' 'ford pinto (sw)' 'datsun 510 (sw)'
 'toyouta corona mark ii (sw)' 'dodge colt (sw)'
 'toyota corolla 1600 (sw)' 'buick century 350' 'chevrolet malibu'
 'ford gran torino' 'dodge coronet custom' 'mercury marquis brougham'
 'chevrolet caprice classic' 'ford ltd' 'plymouth fury gran sedan'
 'chrysler new yorker brougham' 'buick electra 225 custom'
 'amc ambassador brougham' 'plymouth valiant' 'chevrolet nova custom'
 'volkswagen super beetle' 'ford country' 'plymouth custom suburb'
 'oldsmobile vista cruiser' 'toyota carina' 'datsun 610' 'maxda rx3'
 'mercury capri v6' 'fiat 124 sport coupe' 'chevrolet monte carlo s'
 'pontiac grand prix' 'fiat 128' 'opel manta' 'audi 100ls' 'volvo 144ea'
 'dodge dart custom' 'saab 99le' 'toyota mark ii' 'oldsmobile omega'
 'chevrolet nova' 'datsun b210' 'chevrolet chevelle malibu classic'
```

```
'plymouth satellite sebring' 'buick century luxus (sw)'
'dodge coronet custom (sw)' 'audi fox' 'volkswagen dasher' 'datsun 710'
'dodge colt' 'fiat 124 tc' 'honda civic' 'subaru' 'fiat x1.9'
'plymouth valiant custom' 'mercury monarch' 'chevrolet bel air'
'plymouth grand fury' 'buick century' 'chevroelt chevelle malibu'
'plymouth fury' 'buick skyhawk' 'chevrolet monza 2+2' 'ford mustang ii'
'toyota corolla' 'pontiac astro' 'volkswagen rabbit' 'amc pacer'
'volvo 244dl' 'honda civic cvcc' 'fiat 131' 'capri ii' 'renault 12tl'
'dodge coronet brougham' 'chevrolet chevette' 'chevrolet woody'
'vw rabbit' 'dodge aspen se' 'ford granada ghia' 'pontiac ventura sj'
'amc pacer d/l' 'datsun b-210' 'volvo 245' 'plymouth volare premier v8'
'mercedes-benz 280s' 'cadillac seville' 'chevy c10' 'ford f108'
'dodge d100' 'honda accord cvcc' 'buick opel isuzu deluxe'
'renault 5 gtl' 'plymouth arrow gs' 'datsun f-10 hatchback'
'oldsmobile cutlass supreme' 'dodge monaco brougham'
'mercury cougar brougham' 'chevrolet concours' 'buick skylark'
'plymouth volare custom' 'ford granada' 'pontiac grand prix lj'
'chevrolet monte carlo landau' 'chrysler cordoba' 'ford thunderbird'
'volkswagen rabbit custom' 'pontiac sunbird coupe'
'toyota corolla liftback' 'ford mustang ii 2+2' 'dodge colt m/m'
'subaru dl' 'datsun 810' 'bmw 320i' 'mazda rx-4'
'volkswagen rabbit custom diesel' 'ford fiesta' 'mazda glc deluxe'
'datsun b210 gx' 'oldsmobile cutlass salon brougham' 'dodge diplomat'
'mercury monarch ghia' 'pontiac phoenix lj' 'ford fairmont (auto)'
'ford fairmont (man)' 'plymouth volare' 'amc concord'
'buick century special' 'mercury zephyr' 'dodge aspen' 'amc concord d/l'
'buick regal sport coupe (turbo)' 'ford futura' 'dodge magnum xe'
'datsun 510' 'dodge omni' 'toyota celica gt liftback' 'plymouth sapporo'
'oldsmobile starfire sx' 'datsun 200-sx' 'audi 5000' 'volvo 264gl'
'saab 99gle' 'peugeot 604sl' 'volkswagen scirocco' 'honda accord lx'
'pontiac lemans v6' 'mercury zephyr 6' 'ford fairmont 4'
'amc concord dl 6' 'dodge aspen 6' 'ford ltd landau'
'mercury grand marquis' 'dodge st. regis' 'chevrolet malibu classic (sw)'
'chrysler lebaron town @ country (sw)' 'vw rabbit custom'
'maxda glc deluxe' 'dodge colt hatchback custom' 'amc spirit dl'
'mercedes benz 300d' 'cadillac eldorado' 'plymouth horizon'
'plymouth horizon tc3' 'datsun 210' 'fiat strada custom'
'buick skylark limited' 'chevrolet citation' 'oldsmobile omega brougham'
'pontiac phoenix' 'toyota corolla tercel' 'datsun 310' 'ford fairmont'
'audi 4000' 'toyota corona liftback' 'mazda 626' 'datsun 510 hatchback'
'mazda glc' 'vw rabbit c (diesel)' 'vw dasher (diesel)'
'audi 5000s (diesel)' 'mercedes-benz 240d' 'honda civic 1500 gl'
'renault lecar deluxe' 'vokswagen rabbit' 'datsun 280-zx' 'mazda rx-7 gs'
'triumph tr7 coupe' 'ford mustang cobra' 'honda accord'
'plymouth reliant' 'dodge aries wagon (sw)' 'toyota starlet'
'plymouth champ' 'honda civic 1300' 'datsun 210 mpg' 'toyota tercel'
'mazda glc 4' 'plymouth horizon 4' 'ford escort 4w' 'ford escort 2h'
'volkswagen jetta' 'renault 18i' 'honda prelude' 'datsun 200sx'
'peugeot 505s turbo diesel' 'volvo diesel' 'toyota cressida'
'datsun 810 maxima' 'oldsmobile cutlass ls' 'ford granada gl'
'chrysler lebaron salon' 'chevrolet cavalier' 'chevrolet cavalier wagon'
'chevrolet cavalier 2-door' 'pontiac j2000 se hatchback' 'dodge aries se'
'ford fairmont futura' 'amc concord dl' 'volkswagen rabbit l'
'mazda glc custom l' 'mazda glc custom' 'plymouth horizon miser'
'mercury lynx l' 'nissan stanza xe' 'honda civic (auto)' 'datsun 310 gx'
'buick century limited' 'oldsmobile cutlass ciera (diesel)'
'chrysler lebaron medallion' 'ford granada l' 'toyota celica gt'
'dodge charger 2.2' 'chevrolet camaro' 'ford mustang gl' 'vw pickup'
'dodge rampage' 'ford ranger' 'chevy s-10']
```

In [ ]: