In [3]:
```
# Pandas
# - panel data
# - core library for data manipulation and data analysis
#- single and multidimensional data sturcture for data manipulation
# single-dimensional --> series object
# multi-dimensional--> data frame
```

In [7]:
```
#------   pandas series object(one-dimensinonal)----------

import pandas as pd
a=pd.Series([1,2,3,4])
a
```

Out[7]:
```
0    1
1    2
2    3
3    4
dtype: int64
```

In [11]:
```
#--- changing index ---

b=pd.Series(([1,2,3,4]),index=['a','b','c','d'])
b
```

Out[11]:
```
a    1
b    2
c    3
d    4
dtype: int64
```

In [33]:
```
# series object from dictionary

c=pd.Series({'a':1,'b':2,'c':3,'d':4})
c
```

Out[33]:
```
a    1
b    2
c    3
d    4
dtype: int64
```

In [36]:
```
c1=pd.Series({'a':1,'b1':2,'c':3,'d1':4})
c+c1                    # opertsion between Series
```

Out[36]:
```
a     2.0
b     NaN
b1    NaN
c     6.0
d     NaN
d1    NaN
dtype: float64
```

In [17]:
```
 # index position change

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

Out[17]:
```
b    2
c    3
a    1
```

```
d    4
dtype: int64
```

In [28]:
```
# extracting a singel element
            #0 1 2 3 4 5
d=pd.Series([1,2,3,4,5,6])
d[2]
```

Out[28]:
```
3
```

In [34]:
```
d[:4]     #  sequence of element
```

Out[34]:
```
0    1
1    2
2    3
3    4
dtype: int64
```

In [36]:
```
d[-3:]     # extracting elements from back
```

Out[36]:
```
3    4
4    5
5    6
dtype: int64
```

In [52]:
```
# assign value to the elements(change value)
d[3]=10
d[4]='nikhil'
d
```

Out[52]:
```
0         1
1         2
2         3
3        10
4    nikhil
5         6
dtype: object
```

In [57]:
```
#   defining Series from numpy array and other Series

import numpy as np
arr=np.array([1,2,3,4])
e=pd.Series(arr)
e
```

Out[57]:
```
0    1
1    2
2    3
3    4
dtype: int32
```

In [65]:
```
arr[2]=-2      #change value
e
```

Out[65]:
```
0    1
1    2
2   -2
3    4
dtype: int32
```

In [80]:
```
# basic math operation on Series
#  Adding a scalar value to series elements
```

```
f=pd.Series([1,2,3,4,5])
f1=f+10   # (+,-,*,/)
f1
```

Out[80]:
```
0    11
1    12
2    13
3    14
4    15
dtype: int64
```

In [84]:
```
f+2  # (+,-,*,/)
```

Out[84]:
```
0    3
1    4
2    5
3    6
4    7
dtype: int64
```

In [85]:
```
np.log(f)   # log
```

Out[85]:
```
0    0.000000
1    0.693147
2    1.098612
3    1.386294
4    1.609438
dtype: float64
```

In [77]:
```
#adding two Series

g=pd.Series([5,6,7,8,9])
f1=f+g
f1
```

Out[77]:
```
0     6
1     8
2    10
3    12
4    14
dtype: int64
```

In [40]:
```
#  evaluting values
import pandas as pd
g=pd.Series([1,0,2,1,2,3],index=['white','white','blue','green','green','yellow'])
g
```

Out[40]:
```
white     1
white     0
blue      2
green     1
green     2
yellow    3
dtype: int64
```

In [41]:
```
g.unique()                  # unique value()
```

Out[41]:
```
array([1, 0, 2, 3], dtype=int64)
```

In [42]:
```
g.value_counts()          # return unique values and calculate occurences within a Seri
```

Out[42]:
```
2    2
1    2
3    1
0    1
dtype: int64
```

In [51]:
```
g.isin([0,3])   #  values are contained within the data structure
```

Out[51]:
```
white     False
white      True
blue      False
green     False
green     False
yellow     True
dtype: bool
```

In [22]:
```
#  NaN(Not a Number) values


import numpy as np
h=pd.Series([3,-6,np.NaN,14])
h
```

Out[22]:
```
0     3.0
1    -6.0
2     NaN
3    14.0
dtype: float64
```

In [24]:
```
 # isnull()
 # notnull()
#  - identify the indexes without a value

h.isnull()
```

Out[24]:
```
0    False
1    False
2     True
3    False
dtype: bool
```

In [26]:
```
h.notnull()
```

Out[26]:
```
0     True
1     True
2    False
3     True
dtype: bool
```

In [30]:
```
h[h.notnull()]
```

Out[30]:
```
0     3.0
1    -6.0
3    14.0
dtype: float64
```

In [32]:
```
h[h.isnull()]
```

Out[32]:
```
2   NaN
dtype: float64
```

In [68]:
```
#-------------Data frame()   2 dimensional- --------------------
```

```
import pandas as pd
a=pd.DataFrame({"Name":['nikhil','rahul','kk'],"marks":[80,75,78],"class":['TY','SY','F
a
```

Out[68]:

	Name	marks	class
0	nikhil	80	TY
1	rahul	75	SY
2	kk	78	FY

In [64]:
```
a=pd.DataFrame(a,columns=['Name','marks'])     # columns  show specific columns
a
```

Out[64]:

	Name	marks
0	nikhil	80
1	rahul	75
2	kk	78

In [17]:
```
import pandas as pd
import numpya as
b=pd.DataFrame(np.arange(16).reshape((4,4)),
               index=['red','blue','yellow','green'],
               columns=['ball','pen','pencil','paper'])
b
```

Out[17]:

	ball	pen	pencil	paper
red	0	1	2	3
blue	4	5	6	7
yellow	8	9	10	11
green	12	13	14	15

In [18]:
```
# selecting elements

b.columns
```

Out[18]:
```
Index(['ball', 'pen', 'pencil', 'paper'], dtype='object')
```

In [19]:
```
b.index
```

Out[19]:
```
Index(['red', 'blue', 'yellow', 'green'], dtype='object')
```

In [21]:
```
b.values
```

Out[21]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [23]:
```
# slicing

b[:3]
```

Out[23]:

	ball	pen	pencil	paper
red	0	1	2	3
blue	4	5	6	7
yellow	8	9	10	11

In [24]:
```
# convert column to row

b.transpose()
```

Out[24]:

	red	blue	yellow	green
ball	0	4	8	12
pen	1	5	9	13
pencil	2	6	10	14
paper	3	7	11	15

In [25]:
```
# adding new column

b['price']=[5,7,8,10]
b
```

Out[25]:

	ball	pen	pencil	paper	price
red	0	1	2	3	5
blue	4	5	6	7	7
yellow	8	9	10	11	8
green	12	13	14	15	10

In [26]:
```
b1=np.arange(1,8,2)                # set new value for price
b1
```

Out[26]:
```
array([1, 3, 5, 7])
```

In [27]:
```
b['price']=b1
b
```

Out[27]:

	ball	pen	pencil	paper	price
red	0	1	2	3	1
blue	4	5	6	7	3
yellow	8	9	10	11	5
green	12	13	14	15	7

In [59]:
```
# membership of a value

b.isin([3,'b'])     # value present or not
```

Out[59]:

	ball	pen	pencil	paper	price
red	False	False	False	True	False
blue	False	False	False	False	True
yellow	False	False	False	False	False
green	False	False	False	False	False

In [63]:
```
#delete column

del b['pen']
b
```

Out[63]:

	ball	pencil	price
red	0	2	1
blue	4	6	3
yellow	8	10	5
green	12	14	7

In [70]:
```
#filtering
#-   appply the filtering through the application of certain condition(<,>,>=,<=)

b[b<10]
# b[b>10]
# b[b<=10]
# b[b>=10]
```

Out[70]:

	ball	pencil	price
red	0.0	2.0	1
blue	4.0	6.0	3
yellow	8.0	NaN	5
green	NaN	NaN	7

In [12]:
```
# # The index objects
# - show index


c=pd.Series([5,0,3,8,4],index=['red','blue','yellow','white','green'])
c.index
```

Out[12]:
```
Index(['red', 'blue', 'yellow', 'white', 'green'], dtype='object')
```

In [27]:
```
#  methods on index
```

```
c.idxmax()     #----- for max value
```

Out[27]:
```
'white'
```

In [29]:
```
c.idxmin()         # for min value
```

Out[29]:
```
'blue'
```

In [34]:
```
# index with duplicate labels

c=pd.Series(range(7),index=['red','blue','yellow','white','green','red','blue'])
c
```

Out[34]:
```
red       0
blue      1
yellow    2
white     3
green     4
red       5
blue      6
dtype: int64
```

In [36]:
```
c['red']
```

Out[36]:
```
red    0
red    5
dtype: int64
```

In [37]:
```
# other functionalities on indexes
# 1) reindexing
# 2) dropping
# 3) alignment
```

In [52]:
```
# 1) reindexing
# to fill missing value(in order)

d=pd.Series([2,5,7,4],index=['one','two','three','four'])  # normal series
d
```

Out[52]:
```
one      2
two      5
three    7
four     4
dtype: int64
```

In [80]:
```
d.reindex(['three','four','five','eee'])
```

Out[80]:
```
three    7.0
four     4.0
five     NaN
eee      NaN
dtype: float64
```

In [61]:
```
# 2) dropping
# for delete

d.drop(['one','three'])
```

Out[61]:
```
two     5
four    4
```

```
dtype: int64
```

In [67]:
```
# 3) airthmetic and data alignment

d1=pd.Series([1,6,7,8,9],index=['one','brown','three','black','two'])
d1
```

Out[67]:
```
one      1
brown    6
three    7
black    8
two      9
dtype: int64
```

In [81]:
```
d+d1   # addition of d+d1 , means one(2)+one(1)=3
```

Out[81]:
```
black     NaN
brown     NaN
four      NaN
one       3.0
three    14.0
two      14.0
dtype: float64
```

In [86]:
```
# operation between data structure

###--- flexible arithmetic methods
# add()   sub()   div()  mul()

d.mul(d1)   # div()  sub()  add()
```

Out[86]:
```
black     NaN
brown     NaN
four      NaN
one       2.0
three    49.0
two      45.0
dtype: float64
```

In [104…
```
#operation between DataFrame  and   Series



import numpy as np
b=pd.DataFrame(np.arange(16).reshape((4,4)),
               index=['red','blue','yellow','green'],              # dataframe
               columns=['ball','pen','pencil','paper'])


c=pd.Series([5,0,3,8,4],index=['ball','blue','pen','pencil','paper'])            # seri


c-b              # +,*,/
```

Out[104…

	ball	blue	paper	pen	pencil
red	5	NaN	1	2	6
blue	1	NaN	-3	-2	2
yellow	-3	NaN	-7	-6	-2

	ball	blue	paper	pen	pencil
green	-7	NaN	-11	-10	-6

In [4]:
```
 # function apllication and mapping



b=pd.DataFrame(np.arange(16).reshape((4,4)),
               index=['red','blue','yellow','green'],
               columns=['ball','pen','pencil','paper'])
b
```

Out[4]:

	ball	pen	pencil	paper
red	0	1	2	3
blue	4	5	6	7
yellow	8	9	10	11
green	12	13	14	15

In [7]:
```
 #  sqrt()    ----- calculate square root

np.sqrt(b)
```

Out[7]:

	ball	pen	pencil	paper
red	0.000000	1.000000	1.414214	1.732051
blue	2.000000	2.236068	2.449490	2.645751
yellow	2.828427	3.000000	3.162278	3.316625
green	3.464102	3.605551	3.741657	3.872983

In [12]:
```
#   function by row or column--------------

    # define function

#lambda function:-block of code that can be passed as an argument to to a function call
# used when you need a function for a short period time
# anonymous function
# calculate the range covered by the elements in an array

b=lambda x:x.max()-x.min()
b
```

Out[12]:
```
<function __main__.<lambda>(x)>
```

In [46]:
```
def f(x):
    return x.max()-x.min()
f
```

Out[46]:
```
<function __main__.f(x)>
```

In [71]:
```
z1=pd.DataFrame(np.arange(49).reshape((7,7)),
```

```
                  index=['red','blue','yellow','green','black','purple','white'],
                  columns=['ball','pen','pencil','paper','book','textbook','notebook'])
z1
```

Out[71]:

	ball	pen	pencil	paper	book	textbook	notebook
red	0	1	2	3	4	5	6
blue	7	8	9	10	11	12	13
yellow	14	15	16	17	18	19	20
green	21	22	23	24	25	26	27
black	28	29	30	31	32	33	34
purple	35	36	37	38	39	40	41
white	42	43	44	45	46	47	48

In [72]:
```
# apply() -- apply the function just defined on the DataFrame


z1.apply(f,axis=0)
```

Out[72]:

	ball	pen	pencil	paper	book	textbook	notebook
min	0	1	2	3	4	5	6
max	42	43	44	45	46	47	48

In [74]:
```
  # DataFrame in-build function
    # 1) head()--> 1st five records
    # 2) tail()---> last 5 records
    # 3) shape()--> how much records presents
    # 4) describe()--> information about records

#1) head()
z1.head()
```

Out[74]:

	ball	pen	pencil	paper	book	textbook	notebook
red	0	1	2	3	4	5	6
blue	7	8	9	10	11	12	13
yellow	14	15	16	17	18	19	20
green	21	22	23	24	25	26	27
black	28	29	30	31	32	33	34

In [94]:
```
# 2) tail()
z1.tail()
```

Out[94]:
```
<bound method NDFrame.tail of         ball  pen  pencil  paper  book  textbook  notebook
red        0    1       2      3     4         5         6
blue       7    8       9     10    11        12        13
yellow    14   15      16     17    18        19        20
green     21   22      23     24    25        26        27
black     28   29      30     31    32        33        34
```

```
purple    35   36      37     38    39       40       41
white     42   43      44     45    46       47       48>
```

In [91]:
```
# 3) shape()

z1.shape
```

Out[91]:
```
(7, 7)
```

In [95]:
```
# 4) describe()
z1.describe()
```

Out[95]:

	ball	pen	pencil	paper	book	textbook	notebook
count	7.000000	7.000000	7.000000	7.000000	7.000000	7.000000	7.000000
mean	21.000000	22.000000	23.000000	24.000000	25.000000	26.000000	27.000000
std	15.121728	15.121728	15.121728	15.121728	15.121728	15.121728	15.121728
min	0.000000	1.000000	2.000000	3.000000	4.000000	5.000000	6.000000
25%	10.500000	11.500000	12.500000	13.500000	14.500000	15.500000	16.500000
50%	21.000000	22.000000	23.000000	24.000000	25.000000	26.000000	27.000000
75%	31.500000	32.500000	33.500000	34.500000	35.500000	36.500000	37.500000
max	42.000000	43.000000	44.000000	45.000000	46.000000	47.000000	48.000000

In [7]:
```
#   sorting and  Ranking
 # sort
    # ascending order
import pandas as pd
s=pd.Series([4,1,2,5,9],index=['a','b','c','d','e'])
s.sort_values()
```

Out[7]:
```
b    1
c    2
a    4
d    5
e    9
dtype: int64
```

In [13]:
```
# Ranking

# --  it will know which value is at which rank
s.rank()
```

Out[13]:
```
a    3.0
b    1.0
c    2.0
d    4.0
e    5.0
dtype: float64
```

In [7]:
```
# assigning a NaN value

#(Not a  Number)
import pandas as pd
import numpy as np
```

```
n=pd.Series([0,1,2,np.NaN],index=['red','black','white','blue'])
n
```

Out[7]:
```
red      0.0
black    1.0
white    2.0
blue     NaN
dtype: float64
```

In [37]:
```
n['black']=None
n
```

Out[37]:
```
red      0.0
black    NaN
white    2.0
blue     NaN
dtype: float64
```

In [39]:
```
# filtering out NaN values

n.dropna()     # eliminated all NaN values
```

Out[39]:
```
red      0.0
white    2.0
dtype: float64
```

In [41]:
```
n[n.notnull()]    # another way to drop Nan values
```

Out[41]:
```
red      0.0
white    2.0
dtype: float64
```

In [44]:
```
#filling in NaN Occurrences

n.fillna(3)     # fill NaN value
```

Out[44]:
```
red      0.0
black    3.0
white    2.0
blue     3.0
dtype: float64
```

In [8]:
```
n.fillna({'black':5,'blue':4})
```

Out[8]:
```
red      0.0
black    1.0
white    2.0
blue     4.0
dtype: float64
```

In [14]:
```
#  hierachical indexing and leveling

h=pd.Series(np.random.rand(9),
            index=[['white','white','white','blue','blue','blue','red','red','red'],
                   ['up','down','right','up','down','right','up','down','left']])
h
```

Out[14]:
```
white  up       0.928982
       down     0.504775
       right    0.060468
blue   up       0.036114
       down     0.828382
       right    0.413168
```

```
red    up       0.458447
       down     0.998822
       left     0.029433
dtype: float64
```

In [16]:
```
h.index
```

Out[16]:
```
MultiIndex([('white',    'up'),
            ('white',  'down'),
            ('white', 'right'),
            ( 'blue',    'up'),
            ( 'blue',  'down'),
            ( 'blue', 'right'),
            (  'red',    'up'),
            (  'red',  'down'),
            (  'red',  'left')],
           )
```

In [19]:
```
h['white']
```

Out[19]:
```
up       0.928982
down     0.504775
right    0.060468
dtype: float64
```

In [21]:
```
h[:,'up']
```

Out[21]:
```
white    0.928982
blue     0.036114
red      0.458447
dtype: float64
```

In [23]:
```
h['white','up']
```

Out[23]:
```
0.9289817809959987
```

In [24]:
```
# unstack()
#--converts the Series with hierachical index in a simple DataFrame,
# where the second set of indexes is converted into a new set of columns

    # convert series to dataframe
h.unstack()
```

Out[24]:

	down	left	right	up
blue	0.828382	NaN	0.413168	0.036114
red	0.998822	0.029433	NaN	0.458447
white	0.504775	NaN	0.060468	0.928982

In [31]:
```
h1=pd.DataFrame([1,2,3,4,5],index=['a','b','c','d','e'])
h1.stack()                          # convert dataframe to series
```

Out[31]:
```
a  0    1
b  0    2
c  0    3
d  0    4
e  0    5
dtype: int64
```

In [9]:
```
# index option and column option
import pandas as pd
import numpy as np
ic=pd.DataFrame(np.random.rand(16).reshape(4,4),
                index=[['white','white','red','red'],['up','down','up','down']],
               columns=[['pen','pen','paper','paper'],[1,2,1,2]])
ic
```

Out[9]:

		pen		paper	
		1	2	1	2
white	up	0.061587	0.832907	0.386859	0.020504
	down	0.680455	0.693956	0.825806	0.842088
red	up	0.604831	0.461955	0.946557	0.936826
	down	0.313891	0.813661	0.456050	0.527880

In [10]:
```
# Recordering and sorting levels

ic.columns.names=['objects','id']
ic.index.names=['colors','status']
ic
```

Out[10]:

objects		pen		paper	
id		1	2	1	2
colors	status				
white	up	0.061587	0.832907	0.386859	0.020504
	down	0.680455	0.693956	0.825806	0.842088
red	up	0.604831	0.461955	0.946557	0.936826
	down	0.313891	0.813661	0.456050	0.527880

In [11]:
```
#  return new object with two levels

ic.swaplevel('colors','status')
```

Out[11]:

objects		pen		paper	
id		1	2	1	2
status	colors				
up	white	0.061587	0.832907	0.386859	0.020504
down	white	0.680455	0.693956	0.825806	0.842088
up	red	0.604831	0.461955	0.946557	0.936826
down	red	0.313891	0.813661	0.456050	0.527880

In [25]:
```
ic.sort_values(by='colors')
```

Out[25]:

objects		pen		paper	

	objects	pen		paper	
	id	1	2	1	2
colors	status				
red	up	0.604831	0.461955	0.946557	0.936826
	down	0.313891	0.813661	0.456050	0.527880
white	up	0.061587	0.832907	0.386859	0.020504
	down	0.680455	0.693956	0.825806	0.842088

In [31]:
```
ic.sort_index()
```

Out[31]:

	objects	pen		paper	
	id	1	2	1	2
colors	status				
red	down	0.313891	0.813661	0.456050	0.527880
	up	0.604831	0.461955	0.946557	0.936826
white	down	0.680455	0.693956	0.825806	0.842088
	up	0.061587	0.832907	0.386859	0.020504

In [34]:
```
# summary Statistic by level


ic.sum(level='colors')
```

Out[34]:

objects	pen		paper	
id	1	2	1	2
colors				
white	0.742042	1.526863	1.212664	0.862592
red	0.918722	1.275616	1.402606	1.464707

In [39]:
```
ic.sum(level='id',axis=1)   # for column (axis=1)
```

Out[39]:

	id	1	2
colors	status		
white	up	0.448446	0.853412
	down	1.506261	1.536044
red	up	1.551388	1.398781
	down	0.769941	1.341541

In [ ]: