Learning pandas library

In [15]:
```
import pandas as pd
import numpy as np
labels=['a','b','c','d','e']
my_list=[100,200,300,400,500]
d={'p':101,'q':102,'r':103,'s':104,'t':105}
```

In [16]:
```
pd.Series(data=my_list,index=labels)
```

Out[16]:
```
a    100
b    200
c    300
d    400
e    500
dtype: int64
```

In [17]:
```
pd.Series(d)
```

Out[17]:
```
p    101
q    102
r    103
s    104
t    105
dtype: int64
```

In [18]:
```
pd.Series(my_list,d)
```

Out[18]:
```
p    100
q    200
r    300
s    400
t    500
dtype: int64
```

In [19]:
```
#Importance of indexing
ser1=pd.Series([10,20,30,40],index=['Mumbai','Chennnai','Banglore','kanpur'])
ser2=pd.Series([10,20,30,40],index=['Mumbai','rajasthan','Banglore','lucknow'])
ser1+ser2
```

Out[19]:
```
Banglore     60.0
Chennnai      NaN
Mumbai       20.0
kanpur        NaN
lucknow       NaN
rajasthan     NaN
dtype: float64
```

In [20]:
```
from numpy.random import randn
```

In [21]:
```
#Gnerates randomly 20 numbers from standard distribution or gaussian distribution
randn(5,4)
```

Out[21]:
```
array([[ 0.11423221, -0.07763501, -0.25202973, -0.38130643],
       [ 1.33893882,  0.92397238, -0.90754011,  0.7856374 ],
```

```
       [ 0.82459685,  0.18690026,  1.61745298, -0.13045838],
       [-2.86110154,  0.20353265, -0.96509894, -0.1115763 ],
       [-0.19658778, -0.51132292,  0.56869624, -2.02329701]])
```

In [22]:
```
#If we want to generate the same sequence only once
np.random.seed(101)
```

In [23]:
```
randn(5,4)
```

Out[23]:
```
array([[ 2.70684984,  0.62813271,  0.90796945,  0.50382575],
       [ 0.65111795, -0.31931804, -0.84807698,  0.60596535],
       [-2.01816824,  0.74012206,  0.52881349, -0.58900053],
       [ 0.18869531, -0.75887206, -0.93323722,  0.95505651],
       [ 0.19079432,  1.97875732,  2.60596728,  0.68350889]])
```

In [24]:
```
np.random.seed(101)
```

In [25]:
```
randn(5,4)
```

Out[25]:
```
array([[ 2.70684984,  0.62813271,  0.90796945,  0.50382575],
       [ 0.65111795, -0.31931804, -0.84807698,  0.60596535],
       [-2.01816824,  0.74012206,  0.52881349, -0.58900053],
       [ 0.18869531, -0.75887206, -0.93323722,  0.95505651],
       [ 0.19079432,  1.97875732,  2.60596728,  0.68350889]])
```

In [26]:
```
#Creating data frame of 5 rows and 4 columns
df=pd.DataFrame(np.random.randn(5,4),index='A B C D E'.split(),columns='W X Y Z'.spl
```

In [27]:
```
df
```

Out[27]:

	W	X	Y	Z
A	0.302665	1.693723	-1.706086	-1.159119
B	-0.134841	0.390528	0.166905	0.184502
C	0.807706	0.072960	0.638787	0.329646
D	-0.497104	-0.754070	-0.943406	0.484752
E	-0.116773	1.901755	0.238127	1.996652

In [28]:
```
df['W']
```

Out[28]:
```
A    0.302665
B   -0.134841
C    0.807706
D   -0.497104
E   -0.116773
Name: W, dtype: float64
```

In [29]:
```
#Specific access of W and x data frame elements
df[['W','X','Y','Z']]
```

Out[29]:

	W	X	Y	Z

	W	X	Y	Z
A	0.302665	1.693723	-1.706086	-1.159119
B	-0.134841	0.390528	0.166905	0.184502
C	0.807706	0.072960	0.638787	0.329646
D	-0.497104	-0.754070	-0.943406	0.484752
E	-0.116773	1.901755	0.238127	1.996652

In [30]:
```
#How to add the column in present data frame
df['Submittion']=df['X']+df['Y']
```

In [31]:
```
df
```

Out[31]:

	W	X	Y	Z	Submittion
A	0.302665	1.693723	-1.706086	-1.159119	-0.012363
B	-0.134841	0.390528	0.166905	0.184502	0.557432
C	0.807706	0.072960	0.638787	0.329646	0.711747
D	-0.497104	-0.754070	-0.943406	0.484752	-1.697476
E	-0.116773	1.901755	0.238127	1.996652	2.139882

In [32]:
```
#How to drop the column added permanently
df.drop('Submittion',axis=1,inplace=True)
```

In [33]:
```
df
```

Out[33]:

	W	X	Y	Z
A	0.302665	1.693723	-1.706086	-1.159119
B	-0.134841	0.390528	0.166905	0.184502
C	0.807706	0.072960	0.638787	0.329646
D	-0.497104	-0.754070	-0.943406	0.484752
E	-0.116773	1.901755	0.238127	1.996652

In [34]:
```
#How to access the row
df.loc['B']
```

Out[34]:
```
W   -0.134841
X    0.390528
Y    0.166905
Z    0.184502
Name: B, dtype: float64
```

In [35]:
```
#How to access the row with index value
df.iloc[2]
```

Out[35]:
```
W    0.807706
X    0.072960
Y    0.638787
Z    0.329646
Name: C, dtype: float64
```

In [36]:
```
#How to acess specific value from selected row and column
df.loc[['B'],['W']]
```

Out[36]:

	W
B	-0.134841

In [37]:
```
#How to acess MULTIPLE value from selected rowS and columns
df.loc[['B','C'],['W','X']]
```

Out[37]:

	W	X
B	-0.134841	0.390528
C	0.807706	0.072960

In [38]:
```
#How to add condition on present dataframe
df[df>0]
```

Out[38]:

	W	X	Y	Z
A	0.302665	1.693723	NaN	NaN
B	NaN	0.390528	0.166905	0.184502
C	0.807706	0.072960	0.638787	0.329646
D	NaN	NaN	NaN	0.484752
E	NaN	1.901755	0.238127	1.996652

In [39]:
```
#Gives you result where value is more than 0
df[df['W']>0]
```

Out[39]:

	W	X	Y	Z
A	0.302665	1.693723	-1.706086	-1.159119
C	0.807706	0.072960	0.638787	0.329646

In [40]:
```
#Some condition checks
(df['W']>0) & (df['X']>0)
df
```

Out[40]:

	W	X	Y	Z
A	0.302665	1.693723	-1.706086	-1.159119
B	-0.134841	0.390528	0.166905	0.184502
C	0.807706	0.072960	0.638787	0.329646

	W	X	Y	Z
D	-0.497104	-0.754070	-0.943406	0.484752
E	-0.116773	1.901755	0.238127	1.996652

In [41]:
```
newind='MU BA PU KA HY'.split()
```

In [42]:
```
newind
```

Out[42]:
```
['MU', 'BA', 'PU', 'KA', 'HY']
```

In [43]:
```
df['cities']=newind
```

In [44]:
```
df
```

Out[44]:

	W	X	Y	Z	cities
A	0.302665	1.693723	-1.706086	-1.159119	MU
B	-0.134841	0.390528	0.166905	0.184502	BA
C	0.807706	0.072960	0.638787	0.329646	PU
D	-0.497104	-0.754070	-0.943406	0.484752	KA
E	-0.116773	1.901755	0.238127	1.996652	HY

In [45]:
```
#How to change the index as per your own conveniance
df.set_index('cities',inplace=True)
```

In [46]:
```
df
```

Out[46]:

cities	W	X	Y	Z
MU	0.302665	1.693723	-1.706086	-1.159119
BA	-0.134841	0.390528	0.166905	0.184502
PU	0.807706	0.072960	0.638787	0.329646
KA	-0.497104	-0.754070	-0.943406	0.484752
HY	-0.116773	1.901755	0.238127	1.996652

In [47]:
```
df
```

Out[47]:

cities	W	X	Y	Z
MU	0.302665	1.693723	-1.706086	-1.159119
BA	-0.134841	0.390528	0.166905	0.184502

cities	W	X	Y	Z
PU	0.807706	0.072960	0.638787	0.329646
KA	-0.497104	-0.754070	-0.943406	0.484752
HY	-0.116773	1.901755	0.238127	1.996652

In [48]:
```
inside=[1,2,3,1,2,3]
outside=['G1','G1','G1','G2','G2','G2']
h1=list(zip(outside,inside))
```

In [49]:
```
h1
```

Out[49]:
```
[('G1', 1), ('G1', 2), ('G1', 3), ('G2', 1), ('G2', 2), ('G2', 3)]
```

In [50]:
```
hier_ind=pd.MultiIndex.from_tuples(h1)
```

In [51]:
```
hier_ind
```

Out[51]:
```
MultiIndex([('G1', 1),
            ('G1', 2),
            ('G1', 3),
            ('G2', 1),
            ('G2', 2),
            ('G2', 3)],
           )
```

In [52]:
```
df2=pd.DataFrame(np.random.randn(6,2),index=hier_ind,columns=['A','B'])
```

In [53]:
```
df2
```

Out[53]:

		A	B
G1	1	-0.993263	0.196800
	2	-1.136645	0.000366
	3	1.025984	-0.156598
G2	1	-0.031579	0.649826
	2	2.154846	-0.610259
	3	-0.755325	-0.346419

In [54]:
```
df2.loc['G1'].loc[2]
```

Out[54]:
```
A   -1.136645
B    0.000366
Name: 2, dtype: float64
```

In [55]:
```
df2.index.names=['Group','Number']
```

In [56]:
```
df2
```

Out[56]:

Group	Number	A	B
G1	1	-0.993263	0.196800
	2	-1.136645	0.000366
	3	1.025984	-0.156598
G2	1	-0.031579	0.649826
	2	2.154846	-0.610259
	3	-0.755325	-0.346419

In [57]:
```
#Cross section
df2.xs('G1')#EVERYTHING FROM G1
```

Out[57]:

Number	A	B
1	-0.993263	0.196800
2	-1.136645	0.000366
3	1.025984	-0.156598

In [58]:
```
df2.xs(['G1',2]) #Accessing G1 for 2 index value
```

```
C:\Users\dell\AppData\Local\Temp/ipykernel_864/2486483696.py:1: FutureWarning: Passi
ng lists as key for xs is deprecated and will be removed in a future version. Pass k
ey as a tuple instead.
  df2.xs(['G1',2]) #Accessing G1 for 2 index value
A   -1.136645
B    0.000366
Name: (G1, 2), dtype: float64
```

In [59]:
```
#Use of level to acess elements in multiindexing
df2.xs(2,level='Number')
```

Out[59]:

Group	A	B
G1	-1.136645	0.000366
G2	2.154846	-0.610259

Missing values

In [60]:
```
import numpy as np
```

In [61]:
```
df3=pd.DataFrame({'marks':[10,20,np.nan],'runs':[45,np.nan,np.nan],'salary':[5000,30
```

In [62]:
```
df3.dropna()
```

Out[62]:

	marks	runs	salary
0	10.0	45.0	5000

In [63]:
```
df3
```

Out[63]:

	marks	runs	salary
0	10.0	45.0	5000
1	20.0	NaN	3000
2	NaN	NaN	2000

In [64]:
```
df3.dropna(axis=1)
```

Out[64]:

	salary
0	5000
1	3000
2	2000

In [65]:
```
df3
```

Out[65]:

	marks	runs	salary
0	10.0	45.0	5000
1	20.0	NaN	3000
2	NaN	NaN	2000

In [66]:
```
df3.dropna(thresh=1)
```

Out[66]:

	marks	runs	salary
0	10.0	45.0	5000
1	20.0	NaN	3000
2	NaN	NaN	2000

In [67]:
```
df3.fillna(value=0)
```

Out[67]:

	marks	runs	salary
0	10.0	45.0	5000

	marks	runs	salary
1	20.0	0.0	3000
2	0.0	0.0	2000

In [68]:
```
df3['marks'].mean()
```

Out[68]:
```
15.0
```

In [69]:
```
df3['marks'].fillna(value=df3['marks'].mean())
```

Out[69]:
```
0    10.0
1    20.0
2    15.0
Name: marks, dtype: float64
```

Group by

In [70]:
```
df4=pd.DataFrame({'Company':['goog','goog','MSFT','MSFT','fb','fb'],'Person':['Sam',
```

In [71]:
```
df4
```

Out[71]:

	Company	Person	Sales
0	goog	Sam	200
1	goog	Charlie	120
2	MSFT	amy	340
3	MSFT	vaneesa	124
4	fb	carl	243
5	fb	sarah	350

In [73]:
```
bycomp=df4.groupby('Company')
```

In [74]:
```
bycomp
```

Out[74]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000025477624FA0>
```

In [77]:
```
bycomp
```

Out[77]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000025477624FA0>
```

In [79]:
```
#To find the minimum sale made by company in the present made data frame
bycomp.min()
```

Out[79]:

	Person	Sales

Company	Person	Sales

Company		
MSFT	amy	124
fb	carl	243
goog	Charlie	120

In [80]:
```
#To find the maximum sale made by company in the present made data frame
bycomp.max()
```

Out[80]:

Company	Person	Sales
MSFT	vaneesa	340
fb	sarah	350
goog	Sam	200

In [83]:
```
bycomp.describe().loc['goog']
```

Out[83]:
```
Person  count       2
        unique      2
        top       Sam
        freq        1
Sales   count       2
        unique      2
        top       200
        freq        1
Name: goog, dtype: object
```

Concatenation

In [102…
```
df5=pd.DataFrame({'A':['A0','A1','A2','A3'],'B':['B0','B1','B2','B3'],'C':['C0','C1'
df5
```

Out[102…

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [103…
```
df6=pd.DataFrame({'A':['A4','A5','A6','A7'],'B':['B4','B5','B6','B7'],'C':['C4','C5'
```

In [104…
```
df6
```

Out[104…

	A	B	C	D

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

In [105…
```
df7=pd.DataFrame({'A':['A8','A9','A10','A11'],'B':['B8','B9','B10','B11'],'C':['C8',
```

In [106…
```
df7
```

Out[106…

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

In [107…
```
pd.concat([df5,df6,df7],axis=1)
```

Out[107…

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	A4	B4	C4	D4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	A5	B5	C5	D5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	A6	B6	C6	D6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	A7	B7	C7	D7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A8	B8	C8	D8
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A9	B9	C9	D9
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A10	B10	C10	D10
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A11	B11	C11	D11

Merging

In [111…
```
left=pd.DataFrame({'Key':['K0','K1','K2','k3'],'A':['A0','A1','A3','A4'],'B':['B0','
left
```

Out[111…

	Key	A	B

	Key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A3	B2
3	k3	A4	B3

In [113…
```
right=pd.DataFrame({'Key':['K0','K1','K2','k3'],'C':['C0','C1','C3','C4'],'D':['D0',
```

In [114…
```
right
```

Out[114…

	Key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C3	D2
3	k3	C4	D3

In [115…
```
pd.merge(left,right,how='inner',on='Key')
```

Out[115…

	Key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1
2	K2	A3	B2	C3	D2
3	k3	A4	B3	C4	D3

In [120…
```
left1=pd.DataFrame({'Key1':['K0','K0','K1','K2'],'Key2':['K0','K1','K0','K1'],'A':['
```

In [121…
```
left1
```

Out[121…

	Key1	Key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A3	B2
3	K2	K1	A4	B3

In [122…
```
right1=pd.DataFrame({'Key1':['K0','K1','K1','K2'],'Key2':['K0','K0','K0','K0'],'A':[
```

In [123…
```
right1
```

Out[123…

	Key1	Key2	A	B

	Key1	Key2	A	B
0	K0	K0	A0	B0
1	K1	K0	A1	B1
2	K1	K0	A3	B2
3	K2	K0	A4	B3

In [126…
```
pd.merge(left1,right1,on=['Key1','Key2'])
```

Out[126…

	Key1	Key2	A_x	B_x	A_y	B_y
0	K0	K0	A0	B0	A0	B0
1	K1	K0	A3	B2	A1	B1
2	K1	K0	A3	B2	A3	B2

In [127…
```
pd.merge(left1,right1,how='outer',on=['Key1','Key2'])
```

Out[127…

	Key1	Key2	A_x	B_x	A_y	B_y
0	K0	K0	A0	B0	A0	B0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A3	B2	A1	B1
3	K1	K0	A3	B2	A3	B2
4	K2	K1	A4	B3	NaN	NaN
5	K2	K0	NaN	NaN	A4	B3

In [128…
```
pd.merge(left1,right1,how='left',on=['Key1','Key2'])
```

Out[128…

	Key1	Key2	A_x	B_x	A_y	B_y
0	K0	K0	A0	B0	A0	B0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A3	B2	A1	B1
3	K1	K0	A3	B2	A3	B2
4	K2	K1	A4	B3	NaN	NaN

In [129…
```
pd.merge(left1,right1,how='right',on=['Key1','Key2'])
```

Out[129…

	Key1	Key2	A_x	B_x	A_y	B_y
0	K0	K0	A0	B0	A0	B0
1	K1	K0	A3	B2	A1	B1
2	K1	K0	A3	B2	A3	B2

	Key1	Key2	A_x	B_x	A_y	B_y
3	K2	K0	NaN	NaN	A4	B3

Joining

In [132...
```
left2=pd.DataFrame({'A':['A0','A1','A3'],'B':['B0','B1','B2']},index=['K0','K1','K2'
```

In [133...
```
left2
```

Out[133...

	A	B
K0	A0	B0
K1	A1	B1
K2	A3	B2

In [134...
```
right2=pd.DataFrame({'C':['C0','C1','C3'],'D':['D0','D1','D2']},index=['K0','K2','K3
```

In [135...
```
right2
```

Out[135...

	C	D
K0	C0	D0
K2	C1	D1
K3	C3	D2

In [136...
```
left2.join(right2)
```

Out[136...

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A3	B2	C1	D1

In [137...
```
right2.join(left2)
```

Out[137...

	C	D	A	B
K0	C0	D0	A0	B0
K2	C1	D1	A3	B2
K3	C3	D2	NaN	NaN

In [139...
```
left2.join(right2,how='outer')
```

Out[139...

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A3	B2	C1	D1
K3	NaN	NaN	C3	D2

Operations

In [154...
```
df9=pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':['abc','def','ghi
```

In [155...
```
df9
```

Out[155...

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

In [156...
```
#Gives you the  unique column
df9['col2'].unique()
```

Out[156...
```
array([444, 555, 666], dtype=int64)
```

In [157...
```
#Gives you the count of number of unique value
df9['col2'].nunique()
```

Out[157...
```
3
```

In [158...
```
df9['col2'].value_counts()
```

Out[158...
```
444    2
555    1
666    1
Name: col2, dtype: int64
```

In [159...
```
df9['col1']>2
```

Out[159...
```
0    False
1    False
2     True
3     True
Name: col1, dtype: bool
```

In [160...
```
#Giving condition to dataframe
(df9['col2']>2) & (df9['col2']==444)
```

Out[160...
```
0     True
```

Out[160…
```
1    False
2    False
3     True
Name: col2, dtype: bool
```

In [161…
```
def find_cube(x):
    return x**3
```

In [162…
```
#Apply function on dataframe directly
df9['col2'].apply(find_cube)
```

Out[162…
```
0     87528384
1    170953875
2    295408296
3     87528384
Name: col2, dtype: int64
```

In [165…
```
#Finds the lenght of string present in dataframe
df9['col3'].apply(len)
```

Out[165…
```
0    3
1    3
2    3
3    3
Name: col3, dtype: int64
```

In [169…
```
df9.sort_values(by='col1')
```

Out[169…

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

In [ ]:
```
```