In [3]:

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

In [4]:

```
df = pd.DataFrame({'A':[1,2,np.nan],'B':[5,np.nan,np.nan],'C':[1,2,3]})
df['States'] = "CA NV AZ".split()
df.set_index('States',inplace=True)
df
```

Out[4]:

	A	B	C
States			
CA	1.0	5.0	1
NV	2.0	NaN	2
AZ	NaN	NaN	3

- We can fill the null values with something else as shown below
- fillna(value="......") is used to fill the null values in the dataframe

In [7]:

```
print("\n Filling values with a default value\n",'-'*35,sep='')
print(df.fillna(value="FILL VALUE"))
```

```
 Filling values with a default value
-----------------------------------
                 A           B  C
States
CA             1.0         5.0  1
NV             2.0  FILL VALUE  2
AZ      FILL VALUE  FILL VALUE  3
```

In [10]:

```
print("\nFilling the values with computed values\n",'-'*40,sep='')
print(df.fillna(value=df['A'].mean()))
```

```
Filling the values with computed values
----------------------------------------
          A    B  C
States
CA      1.0  5.0  1
NV      2.0  1.5  2
AZ      1.5  1.5  3
```

In [12]:

```
data = {'Company':["GOOG","GOOG","MSFT","MSFT","FB","FB"],
        'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
        'Sales':[200,120,340,124,243,350]}
df = pd.DataFrame(data)
df
```

Out[12]:

	Company	Person	Sales
0	GOOG	Sam	200
1	GOOG	Charlie	120
2	MSFT	Amy	340
3	MSFT	Vanessa	124
4	FB	Carl	243
5	FB	Sarah	350

- **groupby('condition') is nothing but grouping the values based on some condition**

In [14]:

```
byComp = df.groupby('Company')
print("Gropuping by company name and listing mean sales\n",'-'*55,sep='')
print(byComp.mean())
```

```
Gropuping by company name and listing mean sales
-------------------------------------------------------
```

```
         Sales
Company
FB       296.5
GOOG     160.0
MSFT     232.0
```

In [15]:

```
df.groupby('Company').mean()
```

Out[15]:

	Sales
Company	
FB	296.5
GOOG	160.0
MSFT	232.0

In [17]:

```
print("\nAll in one line of command (Stats for 'FB')\n",'-'*65,sep='')
print(pd.DataFrame(df.groupby('Company').describe().loc['FB']).transpose())
```

```
All in one line of command (Stats for 'FB')
-----------------------------------------------------------------
   Sales
   count   mean        std    min     25%    50%     75%    max
FB   2.0  296.5  75.660426  243.0  269.75  296.5  323.25  350.0
```

In [22]:

```
df1=df.groupby('Company').describe()
df1
```

Out[22]:

	Sales							
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
Company								
FB	2.0	296.5	75.660426	243.0	269.75	296.5	323.25	350.0
GOOG	2.0	160.0	56.568542	120.0	140.00	160.0	180.00	200.0
MSFT	2.0	232.0	152.735065	124.0	178.00	232.0	286.00	340.0

In [29]:

```
df1.iloc[1:3,1:3]
```

Out[29]:

	Sales	
	mean	**std**
Company		
GOOG	160.0	56.568542
MSFT	232.0	152.735065

In [30]:

```
print(df.groupby('Company').describe().loc[['GOOG','MSFT']])
```

```
        Sales
        count   mean         std    min    25%    50%    75%    max
Company
GOOG      2.0  160.0   56.568542  120.0  140.0  160.0  180.0  200.0
MSFT      2.0  232.0  152.735065  124.0  178.0  232.0  286.0  340.0
```

In [31]:

```
#Merging two dataframes
#Creating a dataFrame
df2 = pd.DataFrame({'A':['A0','A1','A2','A3'],
                   'B':['B0','B1','B2','B3'],
                   'C':['C0','C1','C2','C3'],
                    'D':['D0','D1','D2','D3']
                       },index=[0,1,2,3])
```

In [32]:

```
df2
```

Out[32]:

	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 [35]:

In [36]:

```
df3
```

Out[36]:

	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 [37]:

```
df4 = pd.DataFrame({'A':['A8','A9','A10','A11'],
                    'B':['B8','B9','B10','B11'],
                    'C':['C8','C9','C10','C11'],
                    'D':['D8','D9','D10','D11']},
                     index=[8,9,10,11])
```

In [38]:

```
df4
```

Out[38]:

	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 [40]:

```
#concatenation -----> Vertically
df_cat1= pd.concat([df2,df3,df4],axis=0)
print("\nAfter Concatenation along row\n",'-'*30,sep='')
print(df_cat1)
df_cat1.loc[2]
```

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

Out[40]:

```
A    A2
B    B2
C    C2
D    D2
Name: 2, dtype: object
```

In [42]:

```
#concatenating row wise

df6 = pd.DataFrame({'A':['A0','A1','A2','A3'],
                    'B':['B0','B1','B2','B3'],
                    'C':['C0','C1','C2','C3'],
                     'D':['D0','D1','D2','D3']
                        },index=[0,1,2,3])
```

In [43]:

```
df7= pd.DataFrame({'A':['A4','A5','A6','A7'],
                  'B':['B4','B5','B6','B7'],
                  'C':['C4','C5','C6','C7'],
                  'D':['D4','D5','D6','D7']},
                 index=[0,1,2,3])
```

In [44]:

```
df8 = pd.DataFrame({'A':['A8','A9','A10','A11'],
                    'B':['B8','B9','B10','B11'],
                    'C':['C8','C9','C10','C11'],
                    'D':['D8','D9','D10','D11']},
                    index=[0,1,2,3])
```

In [45]:

```
df_cat2 = pd.concat([df6,df7,df8],axis=1)
```

In [46]:

```
df_cat2
```

Out[46]:

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

In [60]:

```
#merge by a common key
left = pd.DataFrame({'Key':['k0','k1','k2','k3'],
                    'A':['A0','A1','A2','A3'],
                    'B':['B0','B1','B2','B3']})
right = pd.DataFrame({'Key':['k0','k1','k2','k3'],
                      'C':['C0','C1','C2','C3'],
                      'D':['D0','D1','D2','D3']})
```

In [61]:

```
left
```

Out[61]:

	Key	A	B
0	k0	A0	B0
1	k1	A1	B1
2	k2	A2	B2
3	k3	A3	B3

In [62]:

```
rightmerg2 = pd.merge()
```

Out[62]:

	Key	C	D
0	k0	C0	D0
1	k1	C1	D1
2	k2	C2	D2
3	k3	C3	D3

In [63]:

```
#how is that that tellls how we are merging
merge1 = pd.merge(left,right,how = 'inner', on='Key')
merge1
```

Out[63]:

	Key	A	B	C	D
0	k0	A0	B0	C0	D0
1	k1	A1	B1	C1	D1
2	k2	A2	B2	C2	D2
3	k3	A3	B3	C3	D3

In [69]:
```
merge2 = pd.merge(left,right,how='left',on ='Key')
```

In [70]:
```
merge2
```

Out[70]:

	Key	A	B	C	D
0	k0	A0	B0	C0	D0
1	k1	A1	B1	C1	D1
2	k2	A2	B2	C2	D2
3	k3	A3	B3	C3	D3

In [109…
```
df1 = pd.DataFrame({'key1':['k0','k8','k2','k3'],
                   'A':['A0','A1','A2','A3'],
                   'B':['B0','B1','B2','B3']})
df2 = pd.DataFrame({'key2':['k0','k1','k2','k3'],
                     'C':['C0','C1','C2','C3'],
                     'D':['D0','D1','D2','D3']})
```

In [110…
```
pd.merge(df1,df2,how='inner',left_on='key1',right_on='key2')
#This will match the left table on key1 and right table on key2 and then which ever keys are common will return t
```

Out[110…

	key1	A	B	key2	C	D
0	k0	A0	B0	k0	C0	D0
1	k2	A2	B2	k2	C2	D2
2	k3	A3	B3	k3	C3	D3

In [111…
```
#If you want ot merge on multiple keys then we put on and mention the keys in the list
# if you want all data from left table then we mention how as left
```

In [112…
```
df3 = pd.DataFrame({'key1':['k0','k8','k2','k3'],
                   'A':['A0','A1','A2','A3'],
                    'C':['A0','A1','A2','A3'],
                   'B':['B0','B1','B2','B3']})
df4 = pd.DataFrame({'key2':['k0','k1','k2','k3'],
                     'C':['C0','C1','C2','C3'],
                     'D':['D0','D1','D2','D3']})
```

In [113…
```
pd.merge(df3,df4,how='inner',left_on='key1',right_on='key2')
#When we have two columns with same name in different dataframes then when we merge then column on left table is
#shown as ColumnName_x and column on right table is shown as ColumnName_y
```

Out[113…

	key1	A	C_x	B	key2	C_y	D
0	k0	A0	A0	B0	k0	C0	D0
1	k2	A2	A2	B2	k2	C2	D2
2	k3	A3	A3	B3	k3	C3	D3

In [116…
```
#If we try to merge two tables based on left then if the right table doesnt have the key present in left then
# apart from the key the values of right table is NaN in merging table
# you can event slect multiple keys for the merge operation
merge1= pd.merge(df1,df2,how='outer',on= [['key1','key2']])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/tmp/ipykernel_546029/2237716902.py in <module>
      2 # apart from the key the values of right table is NaN in merging table
      3 # you can event slect multiple keys for the merge operation
----> 4 merge1= pd.merge(df1,df2,how='inner',on= [['key1','key2']])

~/.local/lib/python3.8/site-packages/pandas/core/reshape/merge.py in merge(left, right, how, on, left_on, right_o
n, left_index, right_index, sort, suffixes, copy, indicator, validate)
    105     validate: str | None = None,
    106 ) -> DataFrame:
```

```
--> 107     op = _MergeOperation(
    108         left,
    109         right,

~/.local/lib/python3.8/site-packages/pandas/core/reshape/merge.py in __init__(self, left, right, how, on, left_on
, right_on, axis, left_index, right_index, sort, suffixes, copy, indicator, validate)
    698             self.right_join_keys,
    699             self.join_names,
--> 700         ) = self._get_merge_keys()
    701 
    702         # validate the merge keys dtypes. We may need to coerce

~/.local/lib/python3.8/site-packages/pandas/core/reshape/merge.py in _get_merge_keys(self)
   1090                     if not is_rkey(rk):
   1091                         if rk is not None:
-> 1092                             right_keys.append(right._get_label_or_level_values(rk))
   1093                         else:
   1094                             # work-around for merge_asof(right_index=True)

~/.local/lib/python3.8/site-packages/pandas/core/generic.py in _get_label_or_level_values(self, key, axis)
   1777             values = self.axes[axis].get_level_values(key)._values
   1778         else:
-> 1779             raise KeyError(key)
   1780 
   1781         # Check for duplicates

KeyError: ['key1', 'key2']
```

Join Opeations

- Join is almost similar to merge operation

In [85]:
```
left = pd.DataFrame({'A':['A0','A1','A2'],
                    'B':['B0','B1','B2']},
                   index=['k0','k1','k2'])
right = pd.DataFrame({'C':['C0','C2','C3'],
                     'D':['D0','D2','D3']},
                    index=['k0','k2','k3'])
```

In [86]:
```
left
```

Out[86]:

	A	B
k0	A0	B0
k1	A1	B1
k2	A2	B2

In [87]:
```
right
```

Out[87]:

	C	D
k0	C0	D0
k2	C2	D2
k3	C3	D3

In [97]:
```
left.join(right)
```

Out[97]:

	A	B	C	D
k0	A0	B0	C0	D0
k1	A1	B1	NaN	NaN
k2	A2	B2	C2	D2

Task

In [107…

```
df1 = pd.DataFrame({"k1" :["A","B","C"]})
df2 = pd.DataFrame({"k2" :["B","C","D"]})
df3 = pd.DataFrame({"k3" :["A","B","D"]})


df4 = pd.concat([df1,df2,df3],axis=1)
print(df4)
print("\n")


#dfeafult is axis 0
df5 = pd.concat([df1,df2,df3])
print(df5)
```

```
  k1 k2 k3
0  A  B  A
1  B  C  B
2  C  D  D


    k1   k2   k3
0    A  NaN  NaN
1    B  NaN  NaN
2    C  NaN  NaN
0  NaN    B  NaN
1  NaN    C  NaN
2  NaN    D  NaN
0  NaN  NaN    A
1  NaN  NaN    B
2  NaN  NaN    D
```

In [117…

```
df = pd.DataFrame({'col1': [1,2,3,4,5,6,7,8,9,10],
                  'col2' : [444,555,666,444,333,222,666,777,666,555],
                  'col3': 'aaa bb c dd eee fff gg h iii j'.split()})
df
```

Out[117…

	col1	col2	col3
0	1	444	aaa
1	2	555	bb
2	3	666	c
3	4	444	dd
4	5	333	eee
5	6	222	fff
6	7	666	gg
7	8	777	h
8	9	666	iii
9	10	555	j

In [118…

```
# How do we manipulate one single column and store it back
df['col10'] = df['col1']*10
df
```

Out[118…

	col1	col2	col3	col10
0	1	444	aaa	10
1	2	555	bb	20
2	3	666	c	30
3	4	444	dd	40
4	5	333	eee	50
5	6	222	fff	60
6	7	666	gg	70
7	8	777	h	80
8	9	666	iii	90
9	10	555	j	100

In [119…

```
In [120…
```

```
In [ ]:
```

```
In [122…
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js