PANDAS

##TO READ THE CSV FILE

Import pandas as pd

pd.read_csv(‘file.csv’,sep=’#’,header=None,names=[‘a’,’b’,’c’],skiprows=[0,4,6,8],na_values=[‘value’])

where,

1)na_values=[] -> is used when we have to replace some value with NAN then we use this which value name we write inside list of na_values then value replace by NAN value.

2)Headers=None -> use when we don’t want headers

3)Names=[‘a’,’b’,..] -> use when we want to give our own column names that means our headers.

4)Skiprows -> when we want to skip some rows

**NOTE= If we never give parameter header=None then it will consider our headers as the first row and if we write 0 in the skiprows parameter then it will consider 0th row as our header row.

FUNCTIONS=

1)df.describe() -> gives us all mathematical things like count,mean,std,min,etc.

-By default describe function gives the insight of all the numerical columns which is in dataset it will not give the insights of categorical column.

-In describe function we got 25%,50% and 75% after sorting the data in ascending order it will do it automatically.

2)df.dtypes -> gives us dtypes of all the columns

3)dataframe.to_csv()

- Parameters of to_csv() function ->
    - index=False -> when we don’t want index while saving the data in the csv file
    - header=False -> use when we don’t want headers while storing the data in the csv file
    - sep=’#’ -> use this when you want to give any separator other than comma while storing the data in the csv file
    - columns=[‘column-1’,’column-2’] -> use when you want particular column to store the data in the csv file.

4)df.columns -> By using this you got all the column name of the dataframe

5)df[‘column-name’] -> By using this you can access the particular column

6)In case of Pandas there is 2 data types 1)Series 2)Data frame

- If there is only one record in terms of row or column then it is series
- Series is like a list but in pandas we tell it as series

- If you write like this df['player_id'] -> and if you check the type of this then it is series
- And if you want to convert series into dataframe so write it in list
  e.g= df[['player_id']] -> if you see the type of this it will give you dataframe.
- If you write like this df['column-1','column-2] it will give error because you don't write in the list if one column is there so at that time it is working but if you want more than one column so at that time you have to write in list e.g df[['column-1','column-2']]

READ EXCEL=

df=pd.read_excel('file-name.xlsx',sheet='name of sheet')

you can also use sheet parameter in this read_excel function while reading the excel file if you want to read particular sheet if you have more than one sheet in the excel and if you don't give the sheet name then it will take by default 0th sheet.

READ HTML

-df=pd.read_html('https://www.basketball-reference.com/players/d/derozde01.html')

-type(df) -> list

-len(df)

-We can store the particular table also in csv format like

df[0].to_csv('result.csv')

-> if we read the html page by using read_html page then it will extract all the tables in that html page and store it as a list

->e.g:-

https://www.basketball-reference.com/players/d/derozde01.html

this is the html page and in this there are 8 tables so if we want the first table then we access that like df[0] so it will give first table.

READ JSON DATA

```
s="""
{
  "name":"komal",
  "number":398439489,
  "comp":["deloitte","ineuron","amdox","wiley"]
}

"""
```

```
import json
import pandas as pd
data=json.loads(s)
pd.DataFrame(data,columns=['comp'])
```

I only want comp data from dataframe that’s why I write here columns=[‘comp’]

READ JSON DATA FROM API:

Import pandas as pd

Import requests

- res=requests.get(‘api url’)
- print(res)
- js=res.json()
- pd.DataFrame(js)
  or
- you can read the json data by using pd.read_json(‘api url’)

if suppose the json data is in list

so access it as js[0], js[1] and if suppose dictionary is there the nested dictionary so if you want to extract title of all the data

- for i in len(js):

  print(js[i][‘title’])
- And if you want to store the json data in the dataframe
- pd.DataFrame(js,columns=[‘url’,’title’])

The columns which you want to access from data set mention in this columns attribute.

If the data is like [{‘url’:’ ‘,’title’:’ ‘,user’:{‘login’:’ ‘,’contact’: }},{‘url’:’ ‘,’title’:’ ‘,user’:{‘login’:’ ‘,’contact’: }}]

So if you want to access users from the first data of js and I want to convert that dictionary into the dataframe then do like this

```
ls=[]
for i in range(len(json_data)):
  ls.append(json_data[i]['user'])
  print(ls)
pd.DataFrame(ls)
```

OR

```
import pandas as pd
data1=pd.read_json('https://api.github.com/repos/pandas-
dev/pandas/issues')
data=pd.DataFrame.from_records(data1['user'])
```

PICKLE FORMAT

Import pandas as pd

form_file = pd.read_csv('result.csv')

form_file.to_pickle("my_pickle")

- NOTE= We can store the data in the byte object by storing it in pickle format file.

  Pickle file means serialized data or it will store the data in binary format.

  In pandas this is the first format which stores the data in the binary format.

  No anyone read the pickle file directly bcz the data is stored in the binary format.

h5 FILE FORMAT :=

An H5 is one of the Hierarchical Data Formats (HDF) used to store large amount of data. It is used to store large amount of data in the form of multidimensional arrays. The format is primarily used to store scientific data that is well-organized for quick retrieval and analysis. H5 was introduced as a more enhanced file format to H4. It was originally developed by the National Centre for Supercomputing Applications, and is now supported by The HDF Group.

import pandas as pd

form_file=pd.read_csv("data.csv")

h5_data=pd.HDFStore('test.h5')

h5_data['obj1'] = form_file

h5_data['obj1']

- The data in the h5 file is serialized means it is in binary format so we cannot read that file directly and if we want to read that file then use pandas api to read the file or use the another language to read the h5 file.

- H5 file is store the data in the key value format and key is the object or instance.

E.g : h5_data['obj1'] = form_file

- Or We can perform many operations in the h5 file like put, select,etc.
- PUT OPERATION=

  E.g:=
  h5_data.put('obj2',form_file,format='table')

  it will add the form_file in the table format in obj2 in h5_data file
- SELECT OPERATION=

  E.g :=
  h5_data.select('obj2', where=['index >= 10 and index<= 15'])
  it will show the data whose index is greate than 10 and less than 15 bcz we are not able to see the h5 file directly so we can see the data by performing this type of operations on h5 file.

NOTE= Serde (Serialization Deserialization)

Serialization=object to bytecode

Deserialization=bytecode to object

IMP POINTS=

- If we zip any file so the size of that zip file is less than the original file bcz the way it is storing our data is optimize way it is not losing our data but the architect which they use to store the data is the optimize way. Pyspark is also used for storing large data in the optimize way.
  If we write the data in the h5 file or any binary file then it is serialization and reading data from that file so it is deserialization e.g here we write h5_data.select('obj2', where=['index >= 10 and index<= 15']) where query to take the particular data so it means it is deserialization.

DATA MANUPULATION IN PANDAS:

- df=pd.read_csv('result.csv')

- df.describe()
  This will give the description of all the numeric columns by default.

- If we want the description of the data which has object dtype then do this

  df1=df[df.dtypes[df.dtypes == 'object'].index]
  df1.describe()

  STEPS=
  1)First get all the data which has dtype = object from the dataframe and store it in one variable.

2)Then apply describe function in that then it will give the description of that data bcz all the data is of object dtype.

- df.dtypes gives us the series not the dataframe and the series is like a list and the column is the index and int64,object,float,etc this is the data

- url of titanic data

```
"https://raw.githubusercontent.com/datasciencedojo/datasets/ma
ster/titanic.csv"
```

- If we want the description of the data which has only float dtype then do this

  ```
  df2=df[df.dtypes[df.dtypes == 'float64'].index]
  ```

  Explaination=

  -List=[1,2,3,4]
  -List[3]
  -type(df.dtypes) -> Series/List
  -df.dtypes[index] -> index means df.dtypes == ‘float64’ if we want only the float dtype
  -if we want to access particular column from the data frame so we access like
  df[[‘name’,’age’]]
  as like that we write df[df.dtypes[df.dtypes == ‘float64’].index]
- df[‘name’] -> Series and Series is like a list so we apply all operations in the series which we apply in list

  df[[‘name’]] ->DataFrame

- df[‘Name’][5:16] -> It will give name from index 5 to 15 with dtype Series.
- Sorted the Series => sorted(df[‘Name’])
- df[[‘Name’]][5:16] -> It will give name from index 5 to 15 with dtype dataframe.
- df.columns -> if we want to get each column so do like this

  ```
  for i in df.columns:
          print(df[[i]])
  ```
- To add the new column in the dataframe
  df[‘New Column’] = “Kumar”

  To update the column
  df[‘column name’]=’updated value’

  To add the series in the new column
  df[‘New Column’] = df[‘column-name’]

- If we want to get particular column
  pd.Categorical(df[‘Age’])
  It will give distinct categories which is in column and some other info like length

- IF we want the unique values from the column
  df['Sex'].unique()
- If we want to check is there any null values in this column
  df['Sex'].isnull()
  If the value is null then it will give True and if the value is not null then it will give False.

- If you want to see the index of all null values which is available in the column
  E.g Suppose if I want to see the index of available null values in the Cabin column
  df['Cabin'][df['Cabin'].isnull() == True]

  If I want to see the index of not null values with values in the Cabin column
  Ser=df['Cabin'][df['Cabin'].isnull() == False]

  If we want to extract only the index of the not null values
  Ser.index

  If we want to get the data row wise OR It is a row wise filter
  df.iloc[Ser.index]
  df.iloc[[2,5,6,8]]

- QUESTION=
  1)Extract the name who has maximum fare

```
fare_index=df['Fare'][df['Fare'] == max(df['Fare'])]
for i in fare_index.index:
      print(df.iloc[i]['Name'])
```

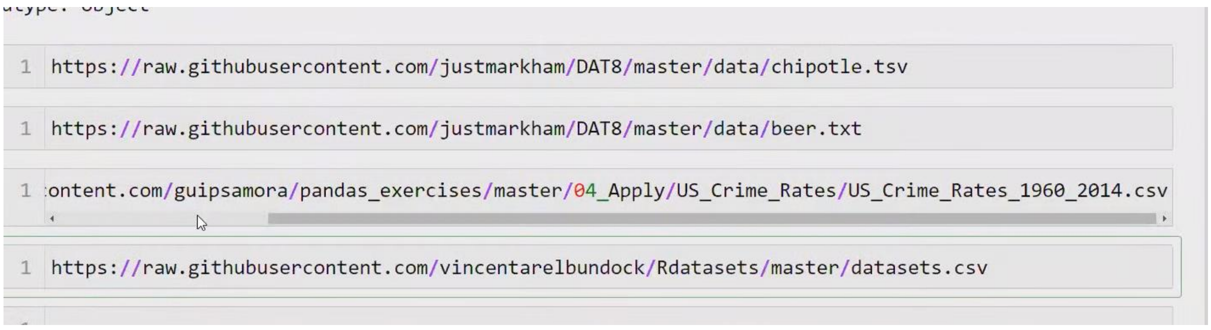
OR

```
df[df['Fare'] == max(df['Fare'])]['Name']
```

2)Create new column in which there is addition of passengerid and age

df['new_column'] = df['passenger-id'] + df['age']

PRACTISE



```
https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv
https://raw.githubusercontent.com/justmarkham/DAT8/master/data/beer.txt
ontent.com/guipsamora/pandas_exercises/master/04_Apply/US_Crime_Rates/US_Crime_Rates_1960_2014.csv
https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/datasets.csv
```

- DRIVE LINK FOR PRACTISE=
  https://drive.google.com/file/d/1etGP85wJWsgqbsh3gX5oxc8dPKLMA1eT/view

- We can change the dtype of the column=

  titanic_train.PassengerId.astype('object')

  -titanic_train = dataset
  -PassengerId =column name

- If we want to see the count of unique values which is in column

  titanic_dataset.Sex.unique()
  titanic_dataset.Sex.value_counts()

- If you want the description of all the columns including categorical and numeric columns
  titanic_dataset.describe(include='all')

- If we want to convert object column into categorical column then

  Embarked is the column name

  New_Embarked=pd.Categorical(df['Embarked'])
  df['New_Embarked']=New_Embarked

  Then if you see the dtypes then it will show category

- Category contains less space than object
- Float64 contains more space than int64
- If we want to give orders to the categories

- Pd.Categorical(["one","two","three",
  "one","two","three"],categories=["one","two","three"],ordered=True)

- Count values of categories
  E,g Pclass has 3 unique values or categories
  titanic_dataset.Pclass.value_counts()

  ```
  o/p= 3   96
       1   30
       2   30
  ```

- Machine learning will not work on categorical data at the end we need to convert all textual data into numeric form.

- Convert data into string format
  Char_cabin=titanic_data['Cabin'].astype(str)

- To check the null value
  - Titanic_data.isnull()

- To check the not null value
  - Titanic_data.notnull()

- To find the null values
  - Titanic_dataset.isnull().sum()

- To check the not null values
  - Titanic_dataset.notnull().sum()

- To drop the row
  - Titanic_dataset.drop([0])

- If we want to drop the column
  - Titanic_dataset.drop([‘Passenger-id’],axis=1)

  Where passenger-id is column name and axis=1 is target to the column

- If I do any operations like

  - Titanic_dataset.drop([‘Passenger-id’],axis=1) this so it will not drop passenger id column from the original dataset but if we want that it will delete it permanently from the original dataset then use inplace =True parameter.
  - Titanic_dataset.drop([‘Passenger-id’],axis=1,inplace=True)

- If we want to get all the null values
  - titanic_dataset[‘age’][titanic_dataset.age.isnull()]

- If we want to get the rows in which age value is nan
  - x=titanic_dataset[‘Age’][titanic_dataset.isnull()].index
    titanic_dataset.iloc[x,:]
    OR
  - np.where(titanic_data[‘fare’].isnull() == True)

- If we want the maximum fare(fare is the column)
  - max(titanic_data[‘fare’])

- If we want the maximum fare whole row
  - max_fare=np.where(titanic_data[‘fare’]==max(titanic_data[‘fare’])
    titanic_data.iloc[max_fare]

- Difference between loc and iloc
  - iloc is used for index and loc is used for index as well as value
  - E.g iloc= 1)titanic_data.iloc[0:100,4:9]
    2)titanic_data.iloc[4,:]

  - E.g loc = 1)titanic_data.loc[titanic_data.Name,titanic_data.columns]
    (Name column is the index here bcz we replace Name column with index)

2)titanic_data.loc[[0,1],['Passenger-id','Name']]
3)If we replace default index with Name column
   - titanic_data.index=titanic_data.Name
4)titanic_data.loc[['Brandon',],titanic_data.columns]

- If we want to add two columns
  - ➔ titanic_data['Name']=titanic_data['first Name']+titanic_data['last name']

- If we want to replace nan value from particular column
  - ➔ titanic_train['Age']=titanic_train[['Age']].fillna(min(titanic_train['Age']))

- If the index is categorical means suppose if the name column is the index column then we can access it by loc function
  - ➔ titanic_data.loc[titanic_data.index[0:5]]
  - ➔ titanic_data.loc[titanic_data.index]

- If the index is in integer
  - ➔ Titanic_data.iloc[0:5,]

- We can give limit to the columns also
  - ➔ titanic_data.iloc[0:5,titanic_data.columns[0:5]]
  - ➔ titanic_data.iloc[0:5,titanic_data.columns]
  - ➔ titanic_data.loc[titanic_data.index[0:5], titanic_data.columns[0:5]]

- if we want to replace the value
  - ➔ titanic_data.replace({1:"one"})
    it will replace 1 with one in dataset.

- Question=If there is one column in the dataset i.e name and I only want that in the name it will give us before the comma words.
  - ➔ titanic_dataset['Name'].apply(lambda x:x[:find(',')])
    and if you want the values
  - ➔ titanic_dataset['New Name']=titanic_dataset['Name'].apply(lambda x:x[:find(',')]).values

- If you want the data who has age greater than 21
  - ➔ titanic_dataset.iloc[np.where(titanic_dataset.Age >21)]

- The reset index() method allows you reset the index back to the default 0,1,2 etc indexes.
  - ➔ titanic_dataset.reset_index(inplace=True,drop=True)
    where, inplace=True use to do this changes in real dataset
    drop=True use to drop the existing index which we set by our own and reset it with index 0,1,2….

- By using shift underscore tab you can see the info about the function

- If we want to rename the columns
  - ➔ titanic_train.rename(columns={"A":"a", "B":"b"})

- titanic_train['Name'] ->type=series

- titanic_train[['Name']]   -> type=Dataframe
- titanic_train[['Name','Age']]   -> used to access the particular column from the database.

- If you want to see the rows in which the age is null

  - ➔ titanic_train_null_value=np.where([titanic_train['Age'].isnull() == True)
  - ➔ titanic_train.iloc[titanic_train_null]

  (It gives the index of rows where age is null)

  OR

  - ➔ titanic_train[titanic_train['Age'].isnull() == True]

- If we want to get the data from particular index
  titanic_train.iloc[5]

- If we want to get the multiple rows
  - ➔ titanic_train.iloc[[1,2,3,4,5]]

- Give the person name who has paid a high fair
  - ➔ titanic_train.iloc[np.where(titanic_train['fair'] == max(titanic_train['fair']))]['Name']
    OR
  - ➔ titanic_train[titanic_train['fair'] == titanic_train['fair'].max()]['Name']

- Give the name of the person who's age is 24
  titanic_train.iloc[np.where(titanic_train['Age'] == 24.0]]['Name']

  OR

  titanic_train[titanic_train['Age'] == 24.0]['Name']

- Give the name of the person who has min age
  - ➔ titanic_train[titanic_train['Age'] == titanic_train['Age'].min()]['Name']

  OR
  - ➔ titanic_train.iloc[np.where(titanic_train['Age'] == titanic_train['Age'].min())]['Name']

- Give the age of the person who's cabin is NAN
  - ➔ titanic_train.iloc[np.where(titanic_train['Cabin'].isnull() == True)][['Age','Cabin']]

- Give the name of the person who is male and who has survived survived means it is equal to one
  - ➔ titanic_train[(titanic_train.Sex == 'male') & (titanic_train.Survived == 1)]['Name']

- If we want to convert dict into dataframe and if we want to convert numpy array into dataframe both we do that

- d={‘key’ : “sudh”, ‘title’: “kumar”, ‘email’: sudh@gmail.com}

  pd.DataFrame(d,index=[‘key’, ‘title’, ‘email’])

  pd.Series(d)

- arr=np.random.randn(3,4)

  pd.DataFrame(arr)

- If we want to convert numpy array into series
  - Ser=pd.Series(np.array([1,2,3,4]),index=[‘a’,’b’,’sudh’,’kumar’])
  - Ser[‘sudh’]

  By using iloc and loc

  - Ser.iloc[‘sudh’] ---(error bcz iloc is used for integer index but only the default index not the string)
  - Ser.loc[‘sudh’]

- Note=loc always take integer index and name index also but iloc only take integer index and it should be the default index.

- We can create series with index
  - series=pd.Series(np.array([1,2,3,4,5]),index=[41,23,34,”sudh”,87])
  - series.loc[‘sudh’]
  - series.loc[41]

- data=pd.DataFrame(np.random.randint(3,6,(4,5)),index=[‘a’,’b’,’c’,’d’], columns=[‘x’,’y’,’z’,’w’,’o’])
  ----- (4 rows 5 columns and generate numbers between 3 to 6)

- If we want to 3rd and 4th row and ‘z’ and ‘w’ column
  - data.loc[[‘c’,’d’],[‘z’,’w’]]
  - data.iloc[2:,2:4]

- create new column with name new and store the addition of x and y value
  - data['new']=data.x+data.y

- If we want to drop the column
- # By default axis=0 it means row but we want to delete the column that’s why I write axis=1
- Inplace=True we use when we have to do the operation on real dataframe.
  - data.drop('x',axis=1,inplace=True)

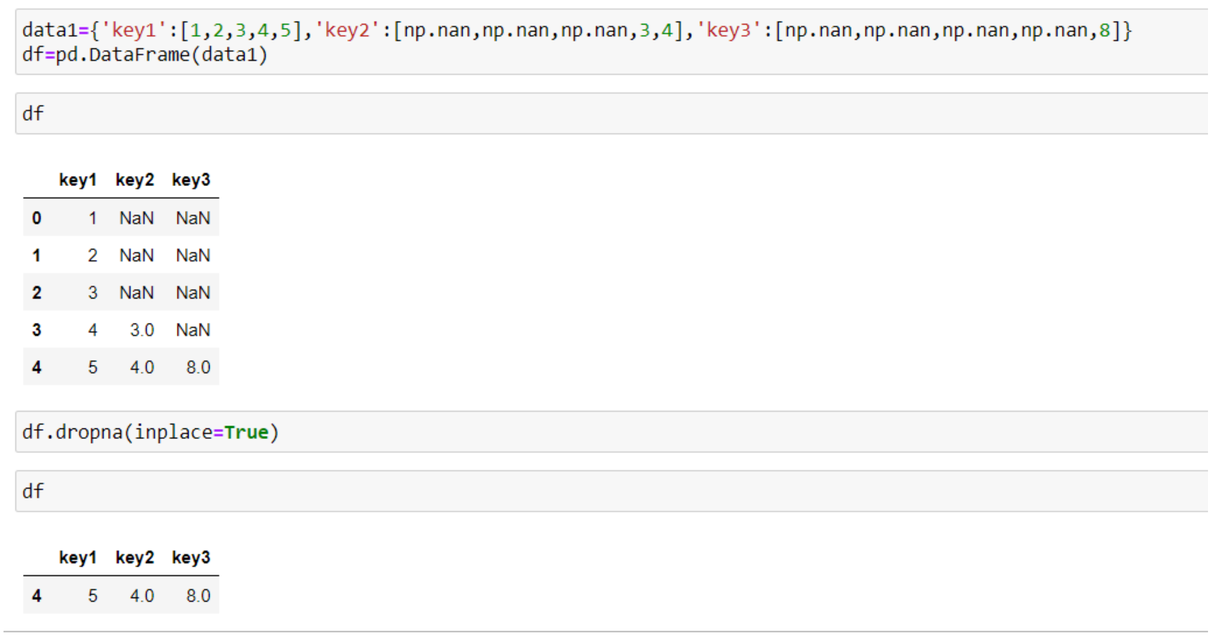
- If we want to extract negative values only from the dataframe
  - ➔ np.random.seed(23)
    df=pd.DataFrame(np.random.randn(4,5),index=['a','b','c','d'],columns=['x','y','z','o','w'])
  (randn is for negative values)
  - ➔ df.values[df.values<0]

- If we want the row name and column name of negative values from the dataframe.
  - ➔ c=list(df.values[df.values<0])

```
for i in df.index:
    for j in df.columns:
        if df.loc[i,j] in c:
            print("Row=",i)
            print("Column=",j)
            print("value=",df.loc[i,j])
```

##DROPNA function

- If we want to create the dataframe from dictionary and we use dropna function to remove all the rows where nan value is present and np.nan is considered as NaN value when we convert into DataFrame.
- By default dropna function takes axis=0 means row
- And if we want to drop the column where nan value is there so at that time use axis=1



```
data1={'key1':[1,2,3,4,5],'key2':[np.nan,np.nan,np.nan,3,4],'key3':[np.nan,np.nan,np.nan,np.nan,8]}
df=pd.DataFrame(data1)
```

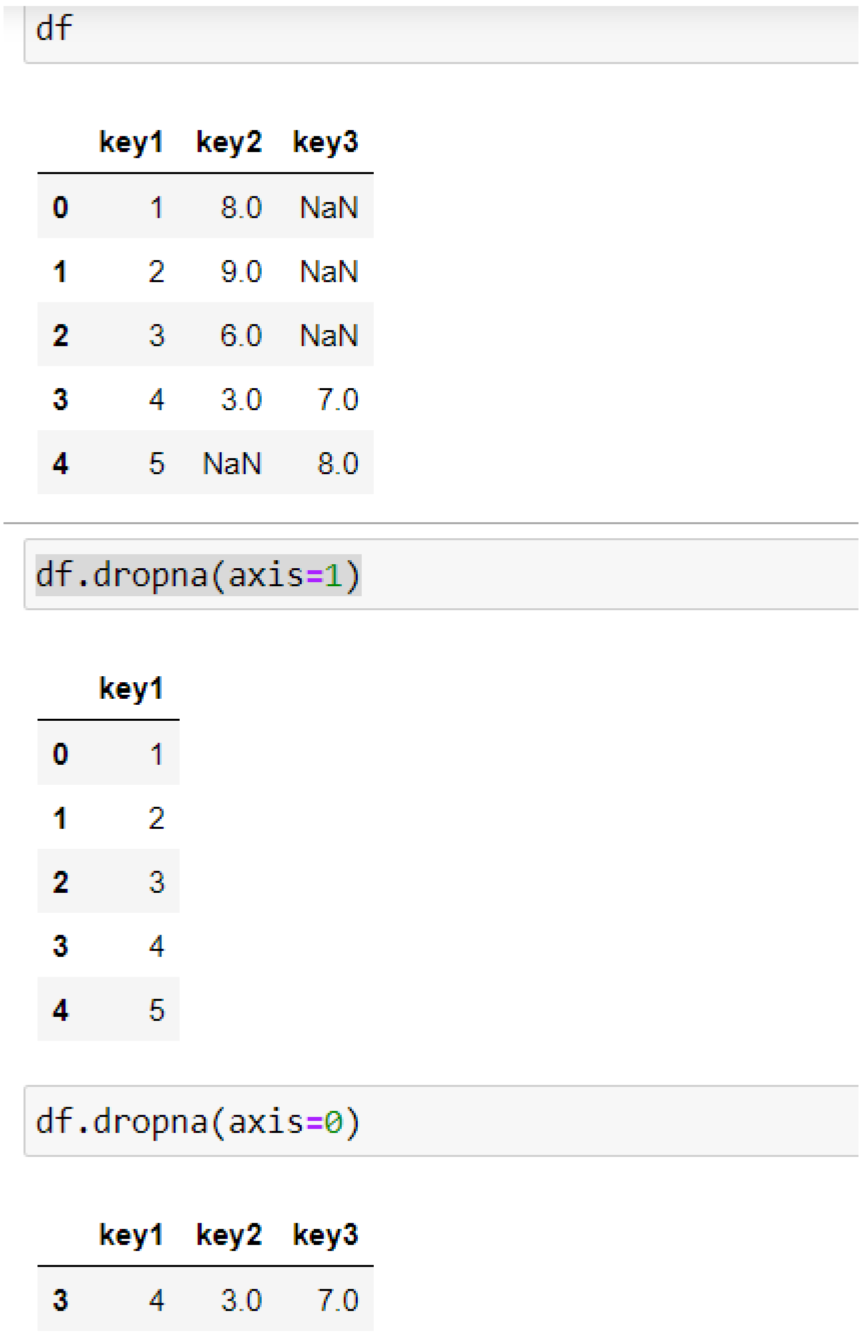
```
df
```

	key1	key2	key3
0	1	NaN	NaN
1	2	NaN	NaN
2	3	NaN	NaN
3	4	3.0	NaN
4	5	4.0	8.0

```
df.dropna(inplace=True)
```

```
df
```

	key1	key2	key3
4	5	4.0	8.0



```python
df
```

	key1	key2	key3
0	1	8.0	NaN
1	2	9.0	NaN
2	3	6.0	NaN
3	4	3.0	7.0
4	5	NaN	8.0

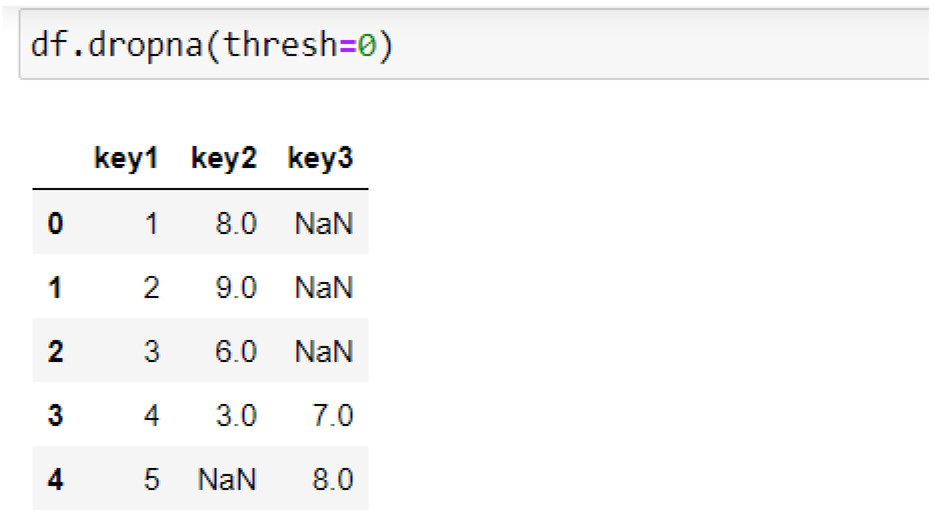
```python
df.dropna(axis=1)
```

	key1
0	1
1	2
2	3
3	4
4	5

```python
df.dropna(axis=0)
```

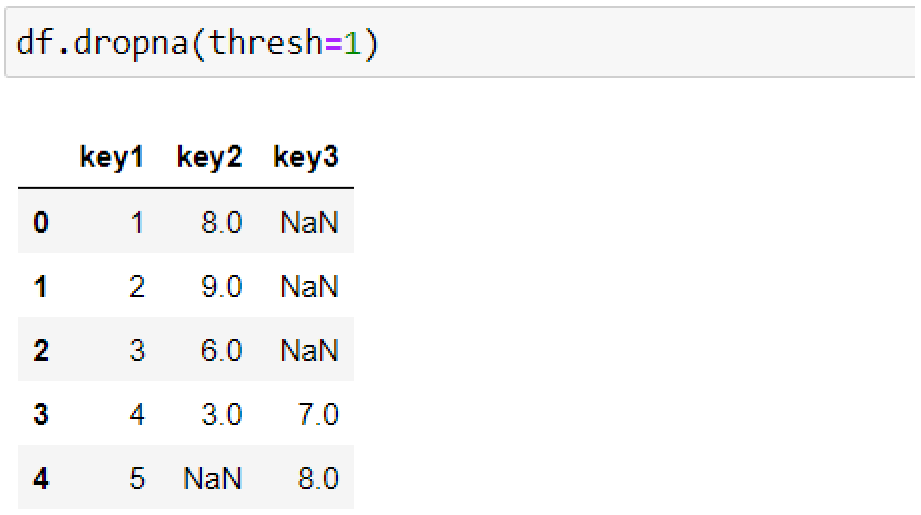
	key1	key2	key3
3	4	3.0	7.0

- thresh parameter is used when we check the at least that number of not nan value here(axis=0)
- thresh=0 ---> here axis=0 means it will check in the row at least zero not nan value is required.
- thresh=1 ---> it will check in the row at least 1 not nan value is required.
- thresh=2 ---> it will check in the row at least 2 not nan value is required.
- If we use thresh in dropna function e.g np.dropna(thresh=0) by default axis=0 it means it will check in each row that at least 0 not nan value is required.



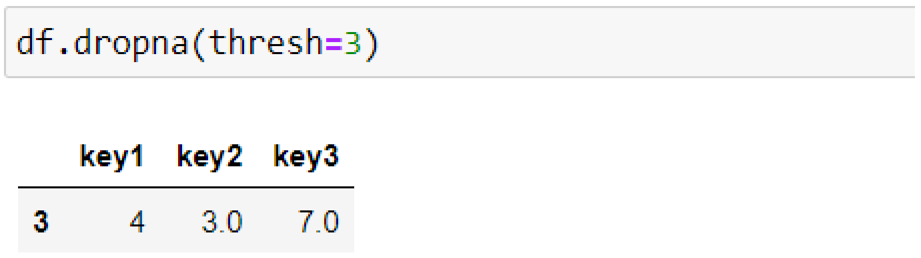
```python
df.dropna(thresh=0)
```

	key1	key2	key3
0	1	8.0	NaN
1	2	9.0	NaN
2	3	6.0	NaN
3	4	3.0	7.0
4	5	NaN	8.0



```python
df.dropna(thresh=1)
```

	key1	key2	key3
0	1	8.0	NaN
1	2	9.0	NaN
2	3	6.0	NaN
3	4	3.0	7.0
4	5	NaN	8.0



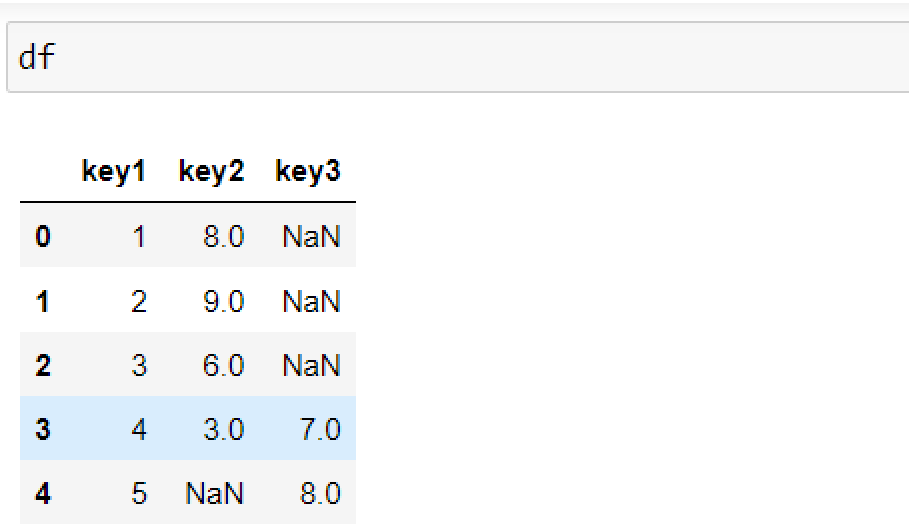
```python
df.dropna(thresh=3)
```

	key1	key2	key3
3	4	3.0	7.0

Same we can do for column also we can check not nan value in column also by adding axis=1

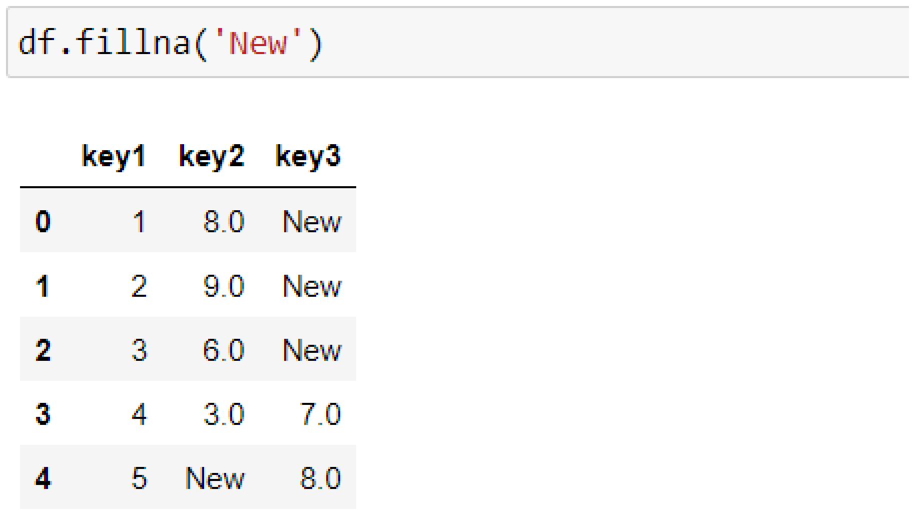
##Fillna Function

- If we want to fill the value wherever nan value is there then use fillna function



```
df
```

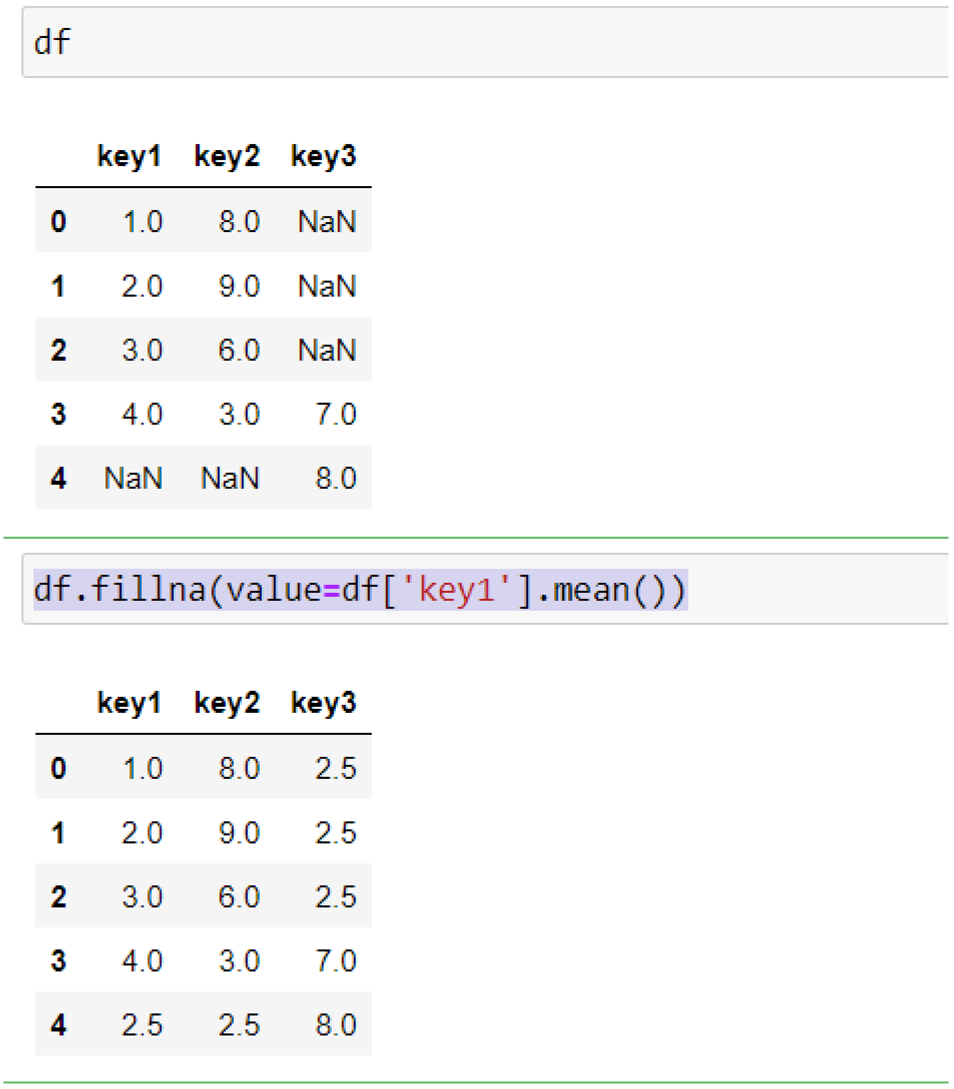
	key1	key2	key3
0	1	8.0	NaN
1	2	9.0	NaN
2	3	6.0	NaN
3	4	3.0	7.0
4	5	NaN	8.0



```
df.fillna('New')
```

	key1	key2	key3
0	1	8.0	New
1	2	9.0	New
2	3	6.0	New
3	4	3.0	7.0
4	5	New	8.0

➢ If we want to fill with the mean value of particular column



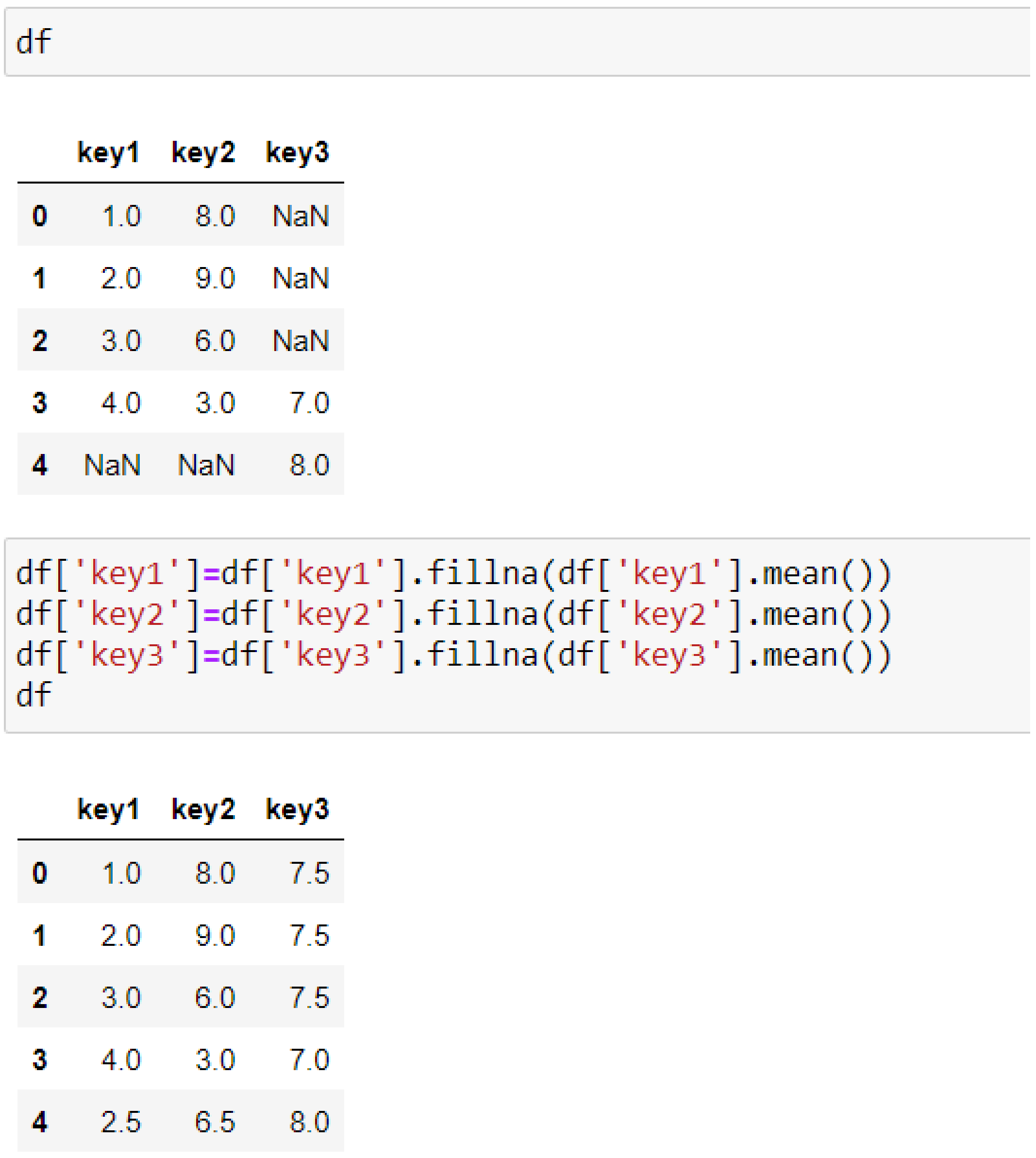
```
df
```

	key1	key2	key3
0	1.0	8.0	NaN
1	2.0	9.0	NaN
2	3.0	6.0	NaN
3	4.0	3.0	7.0
4	NaN	NaN	8.0

```
df.fillna(value=df['key1'].mean())
```

	key1	key2	key3
0	1.0	8.0	2.5
1	2.0	9.0	2.5
2	3.0	6.0	2.5
3	4.0	3.0	7.0
4	2.5	2.5	8.0

➢ If we want to fill the nan value of particular column by average of the column



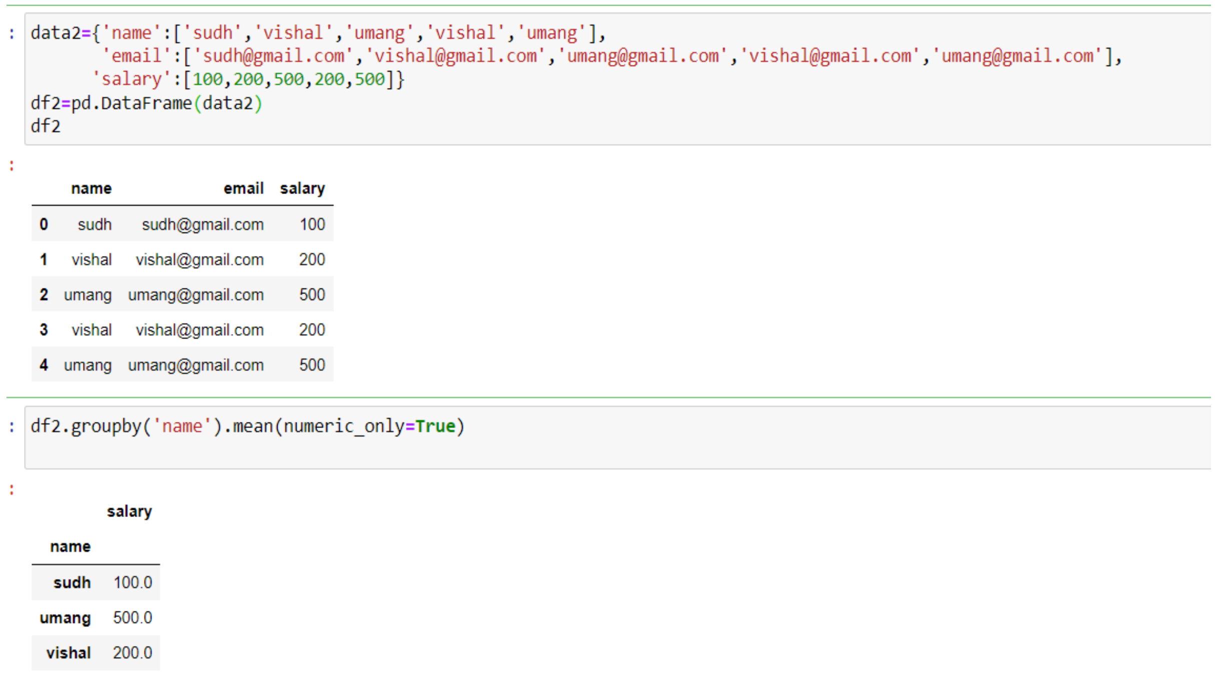
```
df
```

	key1	key2	key3
0	1.0	8.0	NaN
1	2.0	9.0	NaN
2	3.0	6.0	NaN
3	4.0	3.0	7.0
4	NaN	NaN	8.0

```
df['key1']=df['key1'].fillna(df['key1'].mean())
df['key2']=df['key2'].fillna(df['key2'].mean())
df['key3']=df['key3'].fillna(df['key3'].mean())
df
```

	key1	key2	key3
0	1.0	8.0	7.5
1	2.0	9.0	7.5
2	3.0	6.0	7.5
3	4.0	3.0	7.0
4	2.5	6.5	8.0

##GROUPBY Function



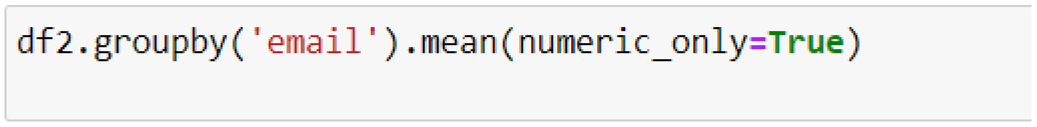
```
data2={'name':['sudh','vishal','umang','vishal','umang'],
       'email':['sudh@gmail.com','vishal@gmail.com','umang@gmail.com','vishal@gmail.com','umang@gmail.com'],
      'salary':[100,200,500,200,500]}
df2=pd.DataFrame(data2)
df2
```

	name	email	salary
0	sudh	sudh@gmail.com	100
1	vishal	vishal@gmail.com	200
2	umang	umang@gmail.com	500
3	vishal	vishal@gmail.com	200
4	umang	umang@gmail.com	500

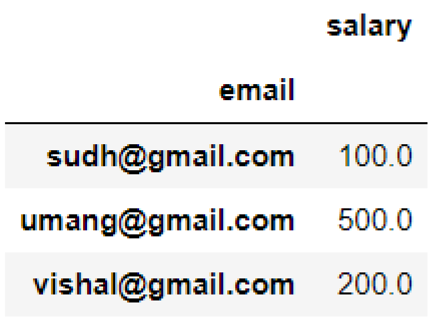
```
df2.groupby('name').mean(numeric_only=True)
```

	salary
name	
sudh	100.0
umang	500.0
vishal	200.0

I use numeric_only bcz email column is not numeric so if we not use numeric_only then it will give error
As like that we do median, sum,etc.



```
df2.groupby('email').mean(numeric_only=True)
```



	salary
email	
sudh@gmail.com	100.0
umang@gmail.com	500.0
vishal@gmail.com	200.0

```
df2.groupby('email').describe()
```

	salary							
	count	mean	std	min	25%	50%	75%	max
email								
sudh@gmail.com	1.0	100.0	NaN	100.0	100.0	100.0	100.0	100.0
umang@gmail.com	2.0	500.0	0.0	500.0	500.0	500.0	500.0	500.0
vishal@gmail.com	2.0	200.0	0.0	200.0	200.0	200.0	200.0	200.0

```
# if we want to get the data of umang
df2.groupby('email').describe().loc['umang@gmail.com']
```

```
salary  count      2.0
        mean     500.0
        std        0.0
        min      500.0
        25%      500.0
        50%      500.0
        75%      500.0
        max      500.0
Name: umang@gmail.com, dtype: float64
```

If we want to get the mean from all the info

```
# if we want to get the data of umang
df2.groupby('email').describe().loc['umang@gmail.com']['salary']['mean']
```

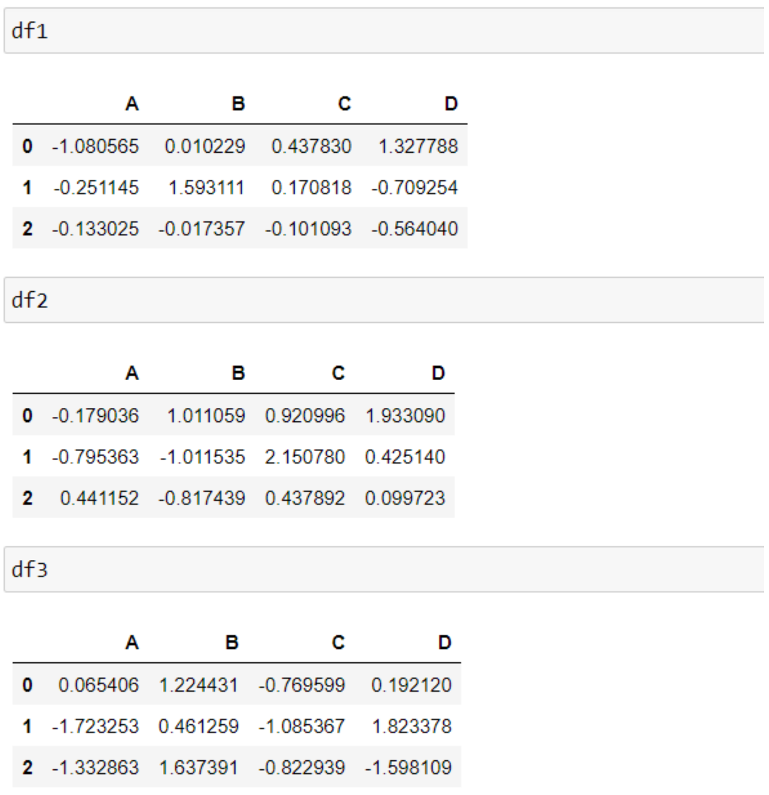
```
500.0
```

```
# if we want to convert row into column and column into row
df2.groupby('email').describe().T
```

	email	sudh@gmail.com	umang@gmail.com	vishal@gmail.com
salary	**count**	1.0	2.0	2.0
	mean	100.0	500.0	200.0
	std	NaN	0.0	0.0
	min	100.0	500.0	200.0
	25%	100.0	500.0	200.0
	50%	100.0	500.0	200.0
	75%	100.0	500.0	200.0
	max	100.0	500.0	200.0

## Concat Function

```
df1=pd.DataFrame(np.random.randn(3,4),columns=['A','B','C','D'],index=[0,1,2])
df2=pd.DataFrame(np.random.randn(3,4),columns=['A','B','C','D'],index=[0,1,2])
df3=pd.DataFrame(np.random.randn(3,4),columns=['A','B','C','D'],index=[0,1,2])
```



```
df1
```

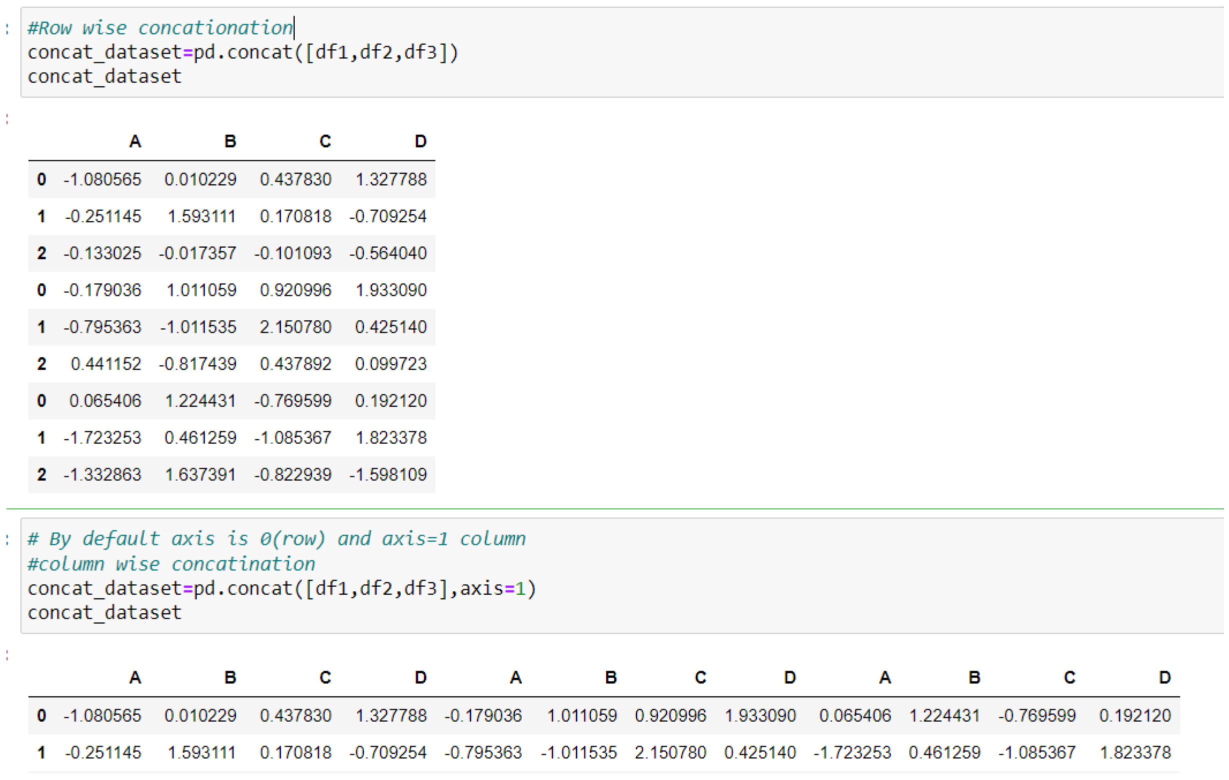
	A	B	C	D
0	-1.080565	0.010229	0.437830	1.327788
1	-0.251145	1.593111	0.170818	-0.709254
2	-0.133025	-0.017357	-0.101093	-0.564040

```
df2
```

	A	B	C	D
0	-0.179036	1.011059	0.920996	1.933090
1	-0.795363	-1.011535	2.150780	0.425140
2	0.441152	-0.817439	0.437892	0.099723

```
df3
```

	A	B	C	D
0	0.065406	1.224431	-0.769599	0.192120
1	-1.723253	0.461259	-1.085367	1.823378
2	-1.332863	1.637391	-0.822939	-1.598109



```
#Row wise concationation
concat_dataset=pd.concat([df1,df2,df3])
concat_dataset
```

	A	B	C	D
0	-1.080565	0.010229	0.437830	1.327788
1	-0.251145	1.593111	0.170818	-0.709254
2	-0.133025	-0.017357	-0.101093	-0.564040
0	-0.179036	1.011059	0.920996	1.933090
1	-0.795363	-1.011535	2.150780	0.425140
2	0.441152	-0.817439	0.437892	0.099723
0	0.065406	1.224431	-0.769599	0.192120
1	-1.723253	0.461259	-1.085367	1.823378
2	-1.332863	1.637391	-0.822939	-1.598109

```
# By default axis is 0(row) and axis=1 column
#column wise concatination
concat_dataset=pd.concat([df1,df2,df3],axis=1)
concat_dataset
```

	A	B	C	D	A	B	C	D	A	B	C	D
0	-1.080565	0.010229	0.437830	1.327788	-0.179036	1.011059	0.920996	1.933090	0.065406	1.224431	-0.769599	0.192120
1	-0.251145	1.593111	0.170818	-0.709254	-0.795363	-1.011535	2.150780	0.425140	-1.723253	0.461259	-1.085367	1.823378

Loc and iloc function in concated data

```
# it is giving data index wise
concat_dataset_row.iloc[0]
```

```
A   -1.080565
B    0.010229
C    0.437830
D    1.327788
Name: 0, dtype: float64
```

```
#loc takes name index data thats why it is giving us multiple data
concat_dataset_row.loc[0]
```

	A	B	C	D
0	-1.080565	0.010229	0.437830	1.327788
0	-0.179036	1.011059	0.920996	1.933090
0	0.065406	1.224431	-0.769599	0.192120

```
concat_dataset
```

	A	B	C	D	A	B	C	D	A	B	C	D
0	-1.080565	0.010229	0.437830	1.327788	-0.179036	1.011059	0.920996	1.933090	0.065406	1.224431	-0.769599	0.192120
1	-0.251145	1.593111	0.170818	-0.709254	-0.795363	-1.011535	2.150780	0.425140	-1.723253	0.461259	-1.085367	1.823378
2	-0.133025	-0.017357	-0.101093	-0.564040	0.441152	-0.817439	0.437892	0.099723	-1.332863	1.637391	-0.822939	-1.598109

```
concat_dataset['A']
```

	A	A	A
0	-1.080565	-0.179036	0.065406
1	-0.251145	-0.795363	-1.723253
2	-0.133025	0.441152	-1.332863

```
# if we want to get only the second A data
concat_dataset.iloc[:,4]
```

```
0   -0.179036
1   -0.795363
2    0.441152
Name: A, dtype: float64
```

e.g of concat

```
import pandas as pd
import numpy as np
df1=pd.DataFrame(np.random.randn(3,4),columns=['A','B','C','D'],index=[0,1,2])
df2=pd.DataFrame(np.random.randn(3,4),columns=['A','B','C','D'],index=[3,4,5])
df3=pd.DataFrame(np.random.randn(3,4),columns=['A','B','C','D'],index=[6,7,8])
```

```
df1
```

	A	B	C	D
0	-1.027757	-0.291438	-0.451213	-0.609088
1	-0.513293	-1.259538	-0.464329	0.984869
2	-0.777857	0.493746	-1.178063	1.735518

```
df2
```

	A	B	C	D
3	0.884996	0.739840	-1.370472	1.457025
4	-0.690722	0.122597	0.854740	0.685937
5	0.515301	-0.424854	-1.139134	-0.457750

```
df3
```

	A	B	C	D
6	-1.537026	-0.496684	-0.716174	-0.303836
7	-0.345073	0.696809	0.514315	0.128830
8	0.267714	0.990067	-1.287290	0.758357

```
#Row wise concationation
concat_dataset_row=pd.concat([df1,df2,df3],axis=1)
concat_dataset_row
```

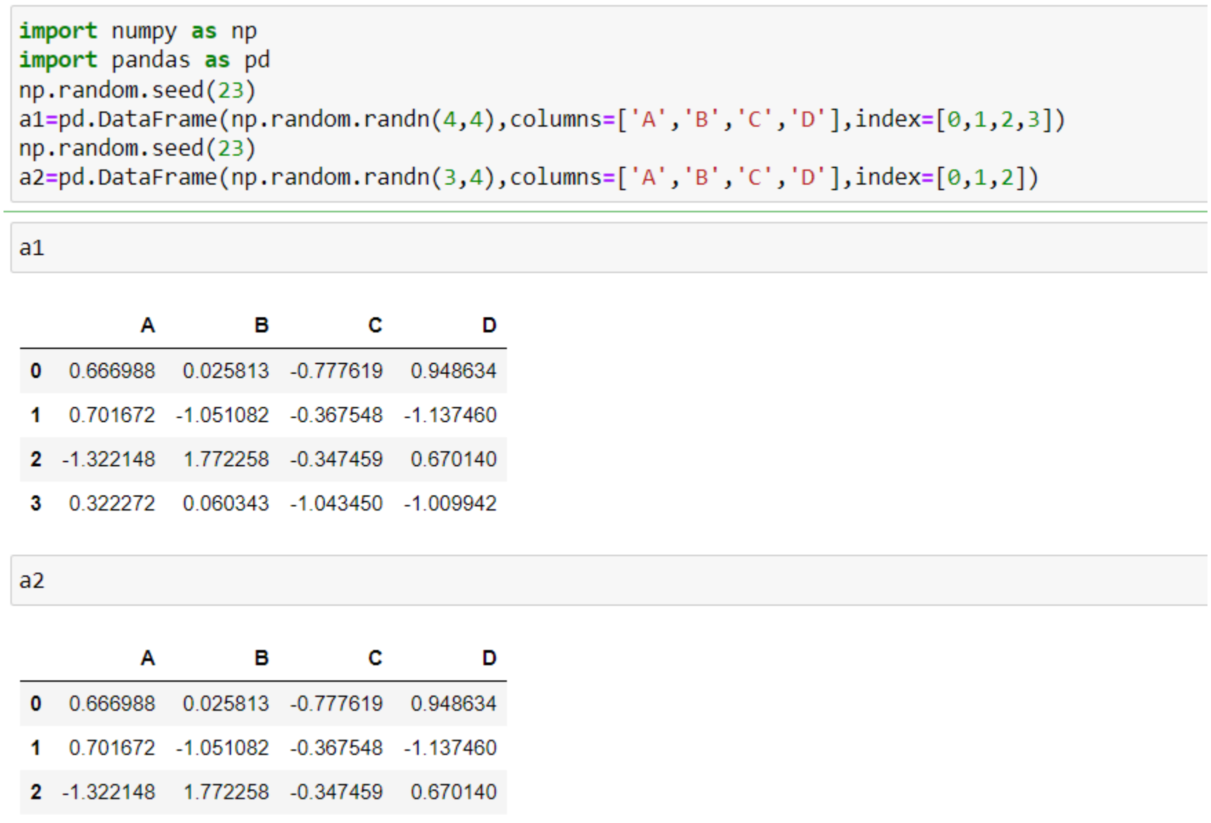
	A	B	C	D	A	B	C	D	A	B	C	D
0	-1.027757	-0.291438	-0.451213	-0.609088	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	-0.513293	-1.259538	-0.464329	0.984869	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	-0.777857	0.493746	-1.178063	1.735518	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	0.884996	0.739840	-1.370472	1.457025	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	-0.690722	0.122597	0.854740	0.685937	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	0.515301	-0.424854	-1.139134	-0.457750	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-1.537026	-0.496684	-0.716174	-0.303836
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-0.345073	0.696809	0.514315	0.128830
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.267714	0.990067	-1.287290	0.758357

```
concat_dataset_row=pd.concat([df1,df2,df3],axis=1).fillna(0)
concat_dataset_row
```

	A	B	C	D	A	B	C	D	A	B	C	D
0	-1.027757	-0.291438	-0.451213	-0.609088	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	-0.513293	-1.259538	-0.464329	0.984869	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2	-0.777857	0.493746	-1.178063	1.735518	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

##MERGE FUNCTION

1)Example



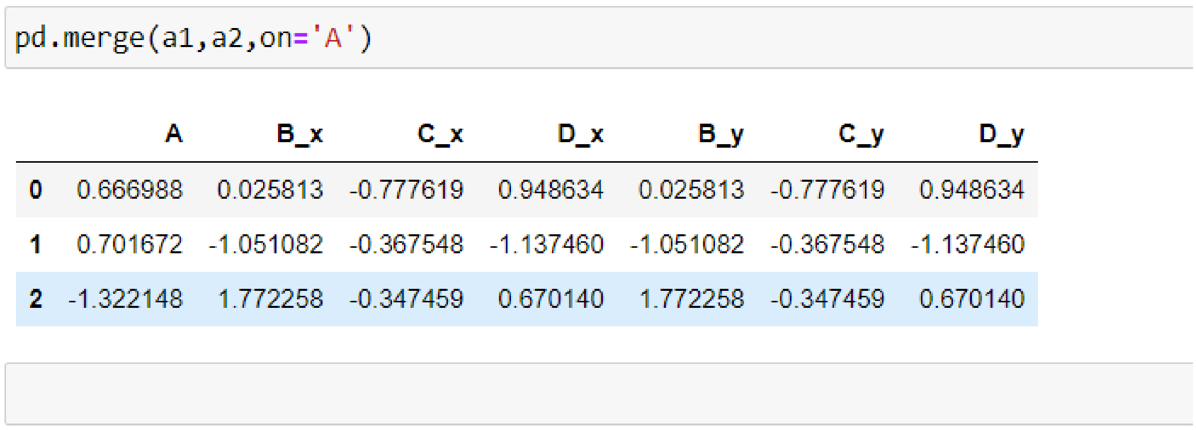
```python
import numpy as np
import pandas as pd
np.random.seed(23)
a1=pd.DataFrame(np.random.randn(4,4),columns=['A','B','C','D'],index=[0,1,2,3])
np.random.seed(23)
a2=pd.DataFrame(np.random.randn(3,4),columns=['A','B','C','D'],index=[0,1,2])
```

```python
a1
```

	A	B	C	D
0	0.666988	0.025813	-0.777619	0.948634
1	0.701672	-1.051082	-0.367548	-1.137460
2	-1.322148	1.772258	-0.347459	0.670140
3	0.322272	0.060343	-1.043450	-1.009942

```python
a2
```

	A	B	C	D
0	0.666988	0.025813	-0.777619	0.948634
1	0.701672	-1.051082	-0.367548	-1.137460
2	-1.322148	1.772258	-0.347459	0.670140

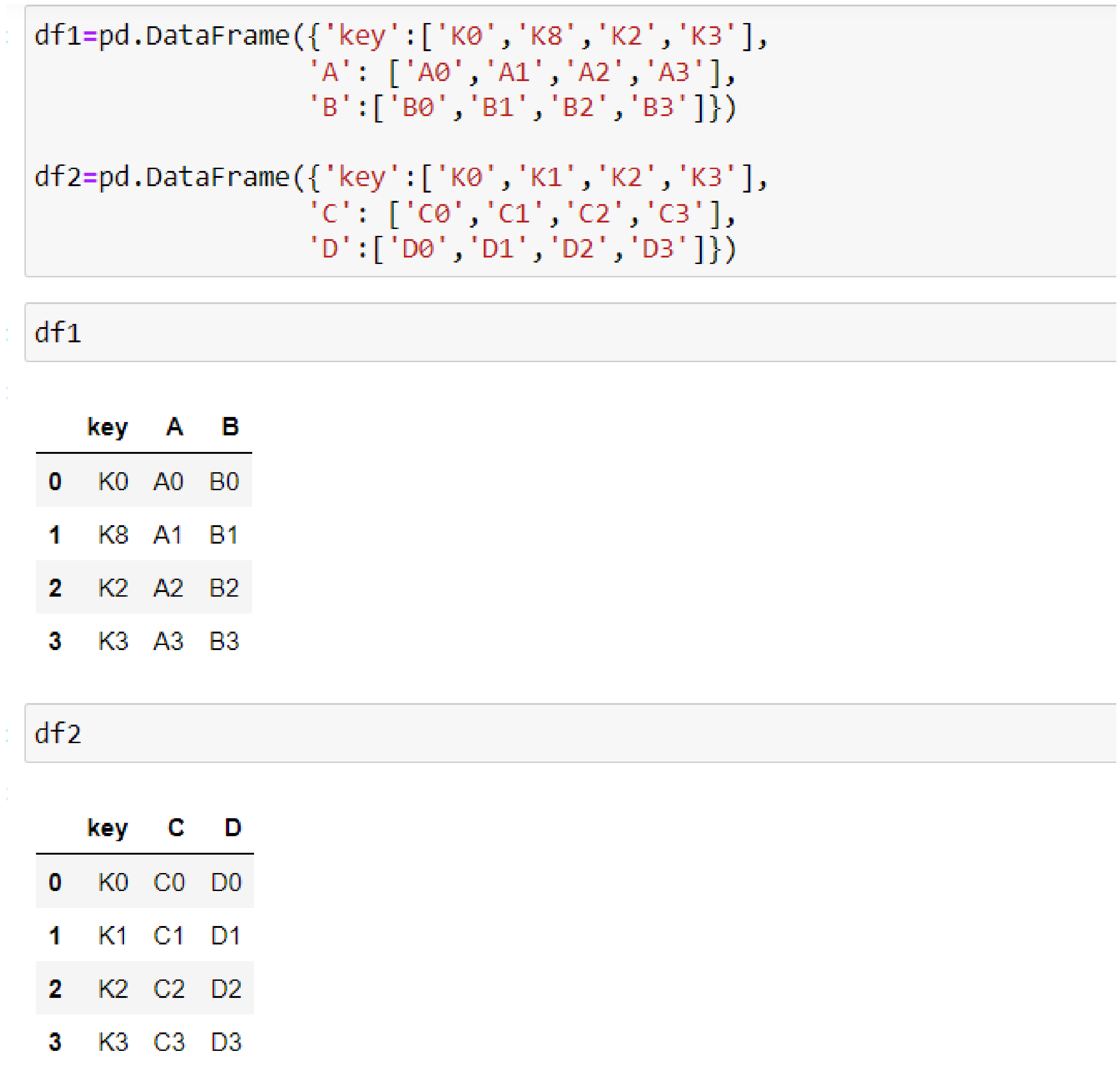


```python
pd.merge(a1,a2,on='A')
```

	A	B_x	C_x	D_x	B_y	C_y	D_y
0	0.666988	0.025813	-0.777619	0.948634	0.025813	-0.777619	0.948634
1	0.701672	-1.051082	-0.367548	-1.137460	-1.051082	-0.367548	-1.137460
2	-1.322148	1.772258	-0.347459	0.670140	1.772258	-0.347459	0.670140

Here it will merge the datasets on the A it means wherever A value is common in the 2 datasets it will merge that values.

2)Example



```
df1=pd.DataFrame({'key':['K0','K8','K2','K3'],
                  'A': ['A0','A1','A2','A3'],
                  'B':['B0','B1','B2','B3']})

df2=pd.DataFrame({'key':['K0','K1','K2','K3'],
                  'C': ['C0','C1','C2','C3'],
                  'D':['D0','D1','D2','D3']})
```

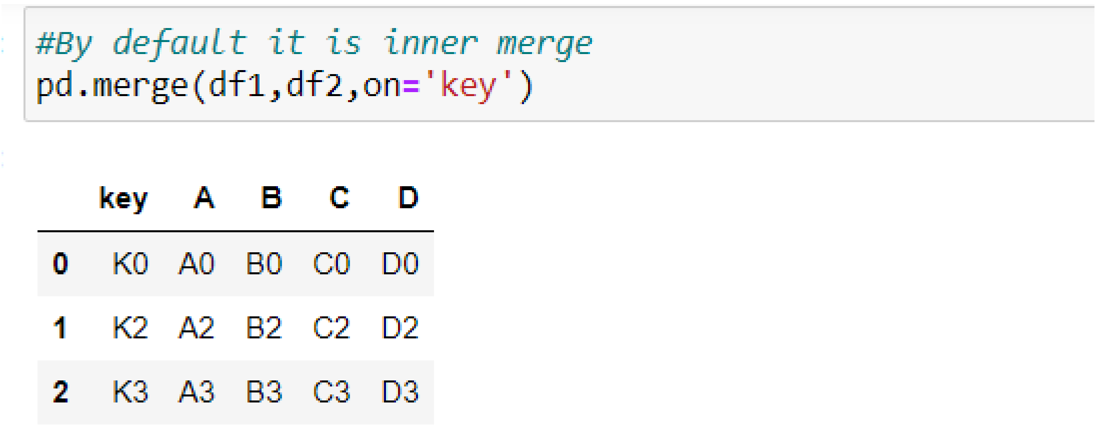
```
df1
```

	key	A	B
0	K0	A0	B0
1	K8	A1	B1
2	K2	A2	B2
3	K3	A3	B3

```
df2
```

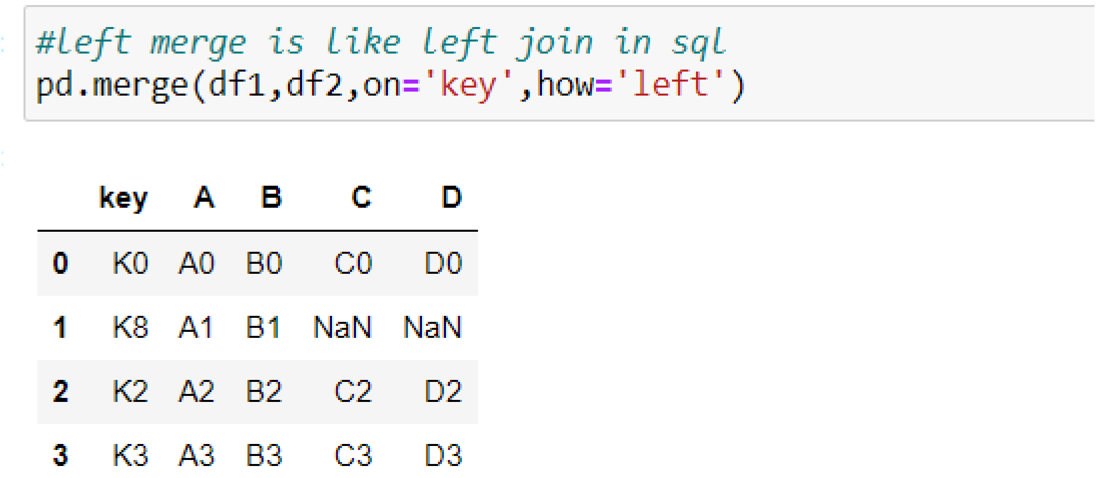
	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

## Left merge,right merge,inner merge,outer merge



```python
#By default it is inner merge
pd.merge(df1,df2,on='key')
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K2	A2	B2	C2	D2
2	K3	A3	B3	C3	D3



```python
#left merge is like left join in sql
pd.merge(df1,df2,on='key',how='left')
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K8	A1	B1	NaN	NaN
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

```
# Right merge is like right join in sql
pd.merge(df1,df2,on='key',how='right')
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	NaN	NaN	C1	D1
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

```
# outer means it will merge all
pd.merge(df1, df2,on='key',how='outer')
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	NaN	NaN	C1	D1
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3
4	K8	A1	B1	NaN	NaN

#Example of merge on 2 columns

```
df1=pd.DataFrame({
    'key1':['K0','K0','K1','K2'],
    'key2':['K0','K1','K0','K1'],
    'A':['A0','A1','A2','A3'],
    'B':['B0','B1','B2','B3']})

df2=pd.DataFrame({
    'key1':['K0','K1','K1','K2'],
    'key2':['K0','K0','K0','K0'],
    'C':['C0','C1','C2','C3'],
    'D':['D0','D1','D2','D3']
})
```

```
df1
```

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

```
df2
```

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

```
pd.merge(df1,df2,on=['key1','key2'])
```

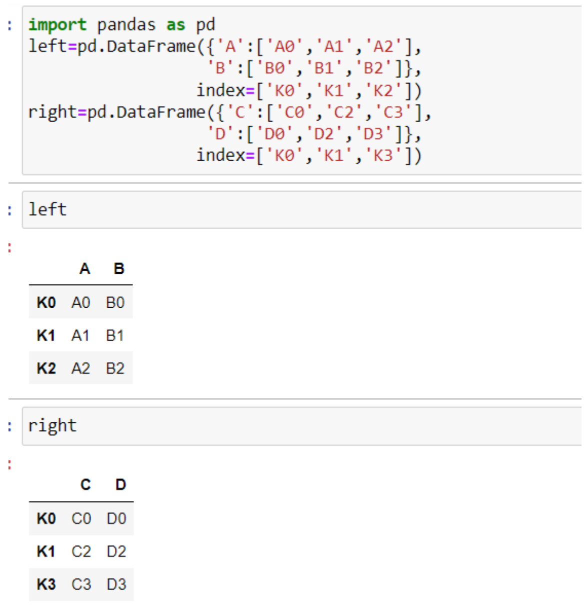
	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

### JOINS IN PANDAS

->Join is same as merge but the only difference is that merge will merge the dataset based out of columns and Join will join the dataset based out of indexs(rows).

EXAMPLE

1)
DATASET



```
import pandas as pd
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','K1','K3'])
```

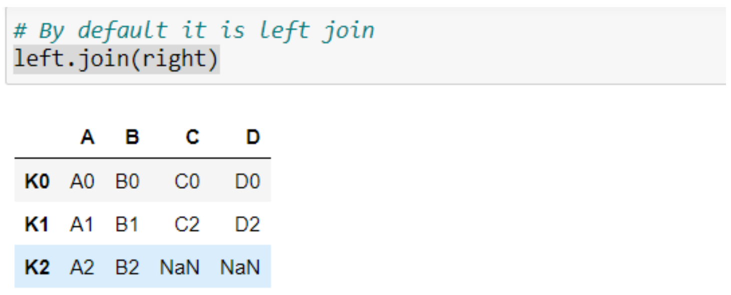
```
left
```

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

```
right
```

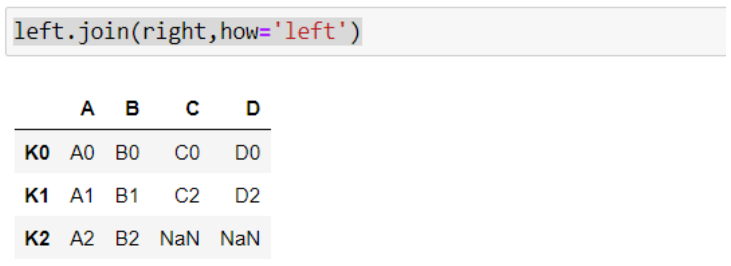
	C	D
K0	C0	D0
K1	C2	D2
K3	C3	D3

LEFT JOIN,INNER JOIN



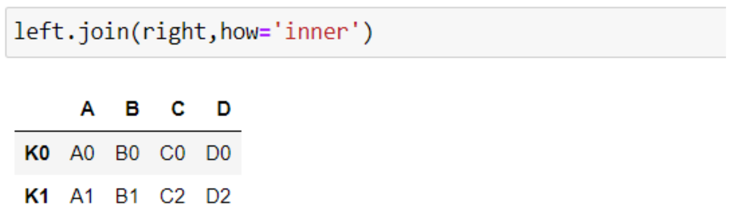
```
# By default it is left join
left.join(right)
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	C2	D2
K2	A2	B2	NaN	NaN



```
left.join(right,how='left')
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	C2	D2
K2	A2	B2	NaN	NaN



```
left.join(right,how='inner')
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	C2	D2

RIGHT JOIN AND OUTER JOIN

```
left.join(right,how='outer')
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	C2	D2
K2	A2	B2	NaN	NaN
K3	NaN	NaN	C3	D3

```
left.join(right,how='right')
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	C2	D2
K3	NaN	NaN	C3	D3

##APPLY FUNCTION

```
df=pd.read_csv(r'C:\Users\HP\Downloads\tips.csv')
def fun(size):
    if size>2:
        return 'Greater than 2'
    else:
        return 'Less than 2'
df['size category']=df['size'].apply(fun)
df
```

	total_bill	tip	sex	smoker	day	time	size	size category
0	16.99	1.01	Female	No	Sun	Dinner	2	Less than 2
1	10.34	1.66	Male	No	Sun	Dinner	3	Greater than 2
2	21.01	3.50	Male	No	Sun	Dinner	3	Greater than 2
3	23.68	3.31	Male	No	Sun	Dinner	2	Less than 2
4	24.59	3.61	Female	No	Sun	Dinner	4	Greater than 2
...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	Greater than 2
240	27.18	2.00	Female	Yes	Sat	Dinner	2	Less than 2
241	22.67	2.00	Male	Yes	Sat	Dinner	2	Less than 2
242	17.82	1.75	Male	No	Sat	Dinner	2	Less than 2
243	18.78	3.00	Female	No	Thur	Dinner	2	Less than 2

244 rows × 8 columns