Pandas: Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas? Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

What Can Pandas Do? Pandas gives you answers about the data. Like: • Is there a correlation between two or more columns? • What is average value? • Max value? • Min value? Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Installation of Pandas:

If you have Python and PIP already installed on a system, then installation of Pandas is very easy. Install it using this command: pip install pandas If this command fails, then use a python distribution that already has Pandas installed like, Anaconda, Spyder etc.

```
pip install pandas
```

Import Pandas Once Pandas is installed, import it in your applications by adding the import keyword: import pandas Now Pandas is imported and ready to use. alias pd is used for pandas

import pandas as pd

Pandas Series: It is like a col in a table. it is like a one dimensional array holding data of any type.

In [11]:
```
# create a simple pandas series from a list
import pandas as pd
a=[10,'ab',30,'COMPUTER']
mycol=pd.Series(a)
print(mycol)
```

```
0          10
1          ab
2          30
3    COMPUTER
dtype: object
```

Labels: it is like index. if nothing is passed then col elements will have normal indexing which starts from 0 to the (number_of_elements-1). it can be used to access any element of the series.

In [14]:
```
#return the element of series using index(label)
print(mycol[3])
```

```
COMPUTER
```

In [49]:
```
#creating your own index
import pandas as pd
b=[1,5,6]
s=pd.Series(b,index=['x','y','z'])
print(s)
```

```
x    1
y    5
z    6
dtype: int64
```

In [16]:
```
print(s['x'])
print(s[1])
```

```
1
5
```

In [26]:
```
#creating series through an array
import pandas as pd
import numpy as np
data=np.array([5,6,7,8])
ds=pd.Series(data,index=[10,20,30,40])
print(ds)
```

```
10    5
20    6
30    7
40    8
dtype: int32
```

In [50]:
```
#creating a series through a dictionary
dict={'a':10,'b':20,'c':30}
ds1=pd.Series(dict)
print(ds1)
```

```
a    10
b    20
c    30
dtype: int64
```

In [20]:
```
#creating series through dictionary using fewer keys as index
dict={'a':10,'b':20,'c':30}
ds2=pd.Series(dict,index=['a','b',1,'c'])
print(ds2)
```

```
a    10.0
b    20.0
1     NaN
c    30.0
dtype: float64
```

In [24]:
```
#creating series with repeated index
#index are not unique
dict={'a':10,'b':20,'c':30}
ds3=pd.Series(dict,index=['a','b','c','a','b','c'])
print(ds3)
```

```
a    10
b    20
c    30
a    10
b    20
c    30
dtype: int64
```

In [20]:
```
#creating series of a particular scalar value with your own index
ds4=pd.Series(20,index=[10,20,30,40,50])
print(ds4)
```

```
10    20
20    20
30    20
40    20
50    20
dtype: int64
```

In [51]:
```
#creating series using numpy fuctions
ds5=pd.Series(np.zeros(10))
print(ds5)
ds6=pd.Series(np.arange(12,25))
print(ds6)
```

```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
5    0.0
6    0.0
7    0.0
8    0.0
9    0.0
dtype: float64
0     12
1     13
2     14
3     15
4     16
5     17
6     18
7     19
8     20
9     21
10    22
11    23
12    24
dtype: int32
```

DataFrames: Data sets in Pandas are usually multi-dimensional tables, called DataFrames. Series is like a column, a DataFrame is the whole table. A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns. Data frame is hetrogeneous i.e collection of different type of value

In [2]:
```
#creating a dataframe from three series(list)/ dictionary list:
import pandas as pd
data = {
    "Names" :['abhishek','harsh','nikhil'],
    "Rollno":[100,101,102],
    'Marks':[90,88,85]
}
df=pd.DataFrame(data,index=[1,2,3])
print(df)
```

```
      Names  Rollno  Marks
1  abhishek     100     90
2     harsh     101     88
3    nikhil     102     85
```

In [1]:
```
#creating a dataframe from two series(list)/ dictionary list: with your owr
import pandas as pd
data = {
    "Names" :['abhishek','harsh','nikhil'],
    "Rollno":[100,101,102],
    'Marks':[90,88,85]
}
df1=pd.DataFrame(data,index=['st1','st2','st3'])
print(df1)
```

```
        Names  Rollno  Marks
st1  abhishek     100     90
st2     harsh     101     88
st3    nikhil     102     85
```

In [2]:
```
# loc attribute is used for Locating a row in dataframe
#the output is a series of a particular row
print(df1.loc['st2'])
```

```
Names     harsh
Rollno      101
Marks        88
Name: st2, dtype: object
```

In [4]:
```
print(df1['Marks'])
#print(df1['st1'])--> not allowed
```

```
st1    90
st2    88
st3    85
Name: Marks, dtype: int64
```

In [26]:
```
print(df1.loc[['st1','st2']])
```

```
        Names  Rollno  Marks
st1  abhishek     100     90
st2     harsh     101     88
```

In [27]:
```
#creating dataframes using numpy arrays
import numpy as np
import pandas as pd
npar=np.array([['abc','xyz','uvw'],[101,102,103],[90,88,85]])
df2=pd.DataFrame(npar,columns=['ID1','ID2','ID3'],index=['Names','Rollno',
print(df2)
```

```
        ID1  ID2  ID3
Names   abc  xyz  uvw
Rollno  101  102  103
Marks    90   88   85
```

In [19]:
```
#creating dataframe using numpy dictionary
import numpy as np
import pandas as pd
npar=np.array([['abc','xyz','uvw'],[101,102,103],[90,88,85]])
np_dict={'Names':npar[0],'RollNo':npar[1],'Marks':npar[2]}
df3=pd.DataFrame(np_dict)
print(df3)
```

```
  Names RollNo Marks
0   abc    101    90
1   xyz    102    88
2   uvw    103    85
```

In [29]:
```
#creating DF using list of lists
import pandas as pd
list_of_list=[['abc',101,90],['xyz',102,88],['uvw',103,85]]
df4=pd.DataFrame(list_of_list,columns=['Names','Rollno','Marks'])
print(df4)
```

```
  Names  Rollno  Marks
0   abc     101     90
1   xyz     102     88
2   uvw     103     85
```

In [6]:
```
#creating DF using list of Dictionary.
import pandas as pd
list_of_dict=[{'Names':'abc','Rollno':101,'Marks':90},
              {'Names':'xyz','Rollno':102,'Marks':88},
              {'Names':'uvw','Rollno':103,'Marks':85}]
df5=pd.DataFrame(list_of_dict)
print(df5)
```

```
  Names  Rollno  Marks
0   abc     101     90
1   xyz     102     88
2   uvw     103     85
```

In [5]:
```
#creating DF using dictionary of series.
s1=pd.Series(['abc','xyz','uvw'])
s2=pd.Series([101,102,103])
s3=pd.Series([90,88,85])
dict_of_series={'Names':s1,'Rollno':s2,'marks':s3}
df6=pd.DataFrame(dict_of_series)
print(df6)
```

```
  Names  Rollno  marks
0   abc     101     90
1   xyz     102     88
2   uvw     103     85
```

In [4]:
```
#accessing index of a Dataframes
print(df6.index)
```

```
RangeIndex(start=0, stop=3, step=1)
```

In [5]:
```
#Accessing Columns of a Data Frame
print(df6.columns)
```

```
Index(['Names', 'Rollno', 'marks'], dtype='object')
```

In [10]:
```
#indexing and slicing is allowed on col header
print(df6['Names'])
#print(df6['Names'][1])
print(df6['Names'][0:2:2])
```

```
0    abc
1    xyz
2    uvw
Name: Names, dtype: object
0    abc
Name: Names, dtype: object
```

In [11]:
```
#inserting a new col in existing DF
#new col will be added as right most col
df6['Address']=['noida','delhi','gzb']
print(df6)
```

```
  Names  Rollno  marks Address
0   abc     101     90   noida
1   xyz     102     88   delhi
2   uvw     103     85     gzb
```

In [13]:
```
df6['pra_status']=df6['marks']>80
print(df6)
```

```
  Names  Rollno  marks Address  pra_status
0   abc     101     90   noida       False
1   xyz     102     88   delhi       False
2   uvw     103     85     gzb       False
```

In [14]:
```
#insertion of new column at desired location in a dataframe
df6.insert(2,'Section',['A','B','A'])#insert(location,col_name,values)
print(df6)
```

```
  Names  Rollno Section  marks Address  pra_status
0   abc     101       A     90   noida       False
1   xyz     102       B     88   delhi       False
2   uvw     103       A     85     gzb       False
```

In [38]:
```
#dropping or deleting any col from a DF
#del keyword
del df6['pra_status']
print(df6)
```

```
  Names Section  Rollno  marks Address
0   abc       A     101     90   noida
1   xyz       B     102     88   delhi
2   uvw       A     103     85     gzb
```

In [39]:
```
#dropping a col with pop
df7=df6.pop('Section')
print(df6)
print(df7)
```

```
  Names  Rollno  marks Address
0   abc     101     90   noida
1   xyz     102     88   delhi
2   uvw     103     85     gzb
0    A
1    B
2    A
Name: Section, dtype: object
```

In [15]:
```
#dropping a col or row from DF using drop()
print(df6)
df6.drop('marks', axis=1,inplace=)
print(df6)
```

```
  Names  Rollno Section  marks Address  pra_status
0   abc     101       A     90   noida       False
1   xyz     102       B     88   delhi       False
2   uvw     103       A     85     gzb       False
  Names  Rollno Section Address  pra_status
0   abc     101       A   noida       False
1   xyz     102       B   delhi       False
2   uvw     103       A     gzb       False
```

In [41]:
```
#dropping a row
df6.drop(0,inplace=True)
print(df6)
```

```
  Names  Rollno Address
1   xyz     102   delhi
2   uvw     103     gzb
```

In [42]:
```
#reseting index to default
df6.reset_index(drop=True,inplace=True)
print(df6)
```

```
  Names  Rollno Address
0   xyz     102   delhi
1   uvw     103     gzb
```

In [16]:
```
#renaming a col of a DF
df6.rename(columns={'Names':'Stu_Name'})
```

Out[16]:

	Stu_Name	Rollno	Section	Address	pra_status
0	abc	101	A	noida	False
1	xyz	102	B	delhi	False
2	uvw	103	A	gzb	False

In [44]:
```
print(df6)
```

```
  Names  Rollno Address
0   xyz     102   delhi
1   uvw     103     gzb
```

In [17]:
```
#for permanent change we must set inplace as TRUE
df6.rename(columns={'Names':'Stu_Name'},inplace=True)
print(df6)
```

```
  Stu_Name  Rollno Section Address  pra_status
0      abc     101       A   noida       False
1      xyz     102       B   delhi       False
2      uvw     103       A     gzb       False
```

In [24]:
```
#creating a DF using numpy random
dfr=pd.DataFrame(np.random.rand(250,5))
print(dfr)#it will print first and last five rows of DF
#print(dfr.to_string())#to_string method we can display complete DF
dfr1=pd.DataFrame(np.random.randint(2,10,size=(3,5)))
print(dfr1)
```

```
            0         1         2         3         4
0    0.501294  0.905376  0.049323  0.393475  0.446488
1    0.902987  0.875587  0.964971  0.281807  0.146739
2    0.366424  0.725206  0.493195  0.732623  0.401131
3    0.350906  0.809631  0.586045  0.191506  0.985471
4    0.820636  0.127417  0.577871  0.188145  0.088218
..        ...       ...       ...       ...       ...
245  0.074677  0.280098  0.736023  0.037943  0.124636
246  0.228597  0.951596  0.338354  0.159433  0.712031
247  0.186327  0.539442  0.151226  0.253499  0.858925
248  0.445907  0.690805  0.313935  0.795078  0.261670
249  0.727315  0.730549  0.255405  0.738341  0.132371

[250 rows x 5 columns]
   0  1  2  3  4
0  6  6  7  9  2
1  7  7  3  4  8
2  6  3  3  3  3
```

In [47]:
```
dfr.head()#it will provide first five rows
```

Out[47]:

	0	1	2	3	4
0	0.118188	0.738726	0.536305	0.696957	0.485691
1	0.087714	0.068397	0.802042	0.462471	0.107957
2	0.277934	0.288766	0.375352	0.449396	0.794294
3	0.936131	0.274651	0.050710	0.722961	0.884917
4	0.342906	0.333072	0.478259	0.240857	0.260225

In [48]:
```
dfr.tail()#it will provide last five rows
```

Out[48]:

	0	1	2	3	4
245	0.515896	0.725178	0.020433	0.252714	0.703838
246	0.568715	0.705708	0.007391	0.560841	0.762710
247	0.809811	0.858970	0.645272	0.947625	0.011744
248	0.031755	0.630835	0.226820	0.815927	0.397492
249	0.659800	0.589403	0.945276	0.773535	0.912125

In [49]:
```
print(dfr.head(10))#for first 10 rows
print(dfr.tail(15))#for last 15 rows
```

```
          0         1         2         3         4
0  0.118188  0.738726  0.536305  0.696957  0.485691
1  0.087714  0.068397  0.802042  0.462471  0.107957
2  0.277934  0.288766  0.375352  0.449396  0.794294
3  0.936131  0.274651  0.050710  0.722961  0.884917
4  0.342906  0.333072  0.478259  0.240857  0.260225
5  0.330090  0.039013  0.888633  0.200711  0.056866
6  0.250550  0.512536  0.647614  0.099011  0.436954
7  0.393599  0.611261  0.142363  0.610543  0.274917
8  0.977425  0.626738  0.277000  0.162345  0.533917
9  0.030888  0.393670  0.388361  0.418927  0.371608
            0         1         2         3         4
235  0.010231  0.937293  0.195192  0.715973  0.091497
236  0.001091  0.125901  0.279095  0.309217  0.029799
237  0.015323  0.566269  0.218144  0.228086  0.792246
238  0.412259  0.705126  0.820062  0.414508  0.746717
239  0.214710  0.651711  0.182186  0.428158  0.108934
240  0.409179  0.249154  0.463424  0.144854  0.884721
241  0.485064  0.590841  0.035129  0.084302  0.386779
242  0.397736  0.250944  0.653726  0.364660  0.947238
243  0.771878  0.583874  0.024450  0.931386  0.797897
244  0.000755  0.504769  0.805539  0.061454  0.610468
245  0.515896  0.725178  0.020433  0.252714  0.703838
246  0.568715  0.705708  0.007391  0.560841  0.762710
247  0.809811  0.858970  0.645272  0.947625  0.011744
248  0.031755  0.630835  0.226820  0.815927  0.397492
249  0.659800  0.589403  0.945276  0.773535  0.912125
```

In [7]:
```
print(df5)
```

```
  Names  Rollno  Marks
0   abc     101     90
1   xyz     102     88
2   uvw     103     85
```

In [51]:
```
df5.index=['st1','st2','st3']
print(df5)
```

```
    Names  Rollno  Marks
st1   abc     101     90
st2   xyz     102     88
st3   uvw     103     85
```

In [52]:
```
print(df5.loc['st1','Rollno'])# loc: provides data by using labels
```

```
101
```

In [53]:
```
print(df5.iloc[0,1])#iloc: provides data by using index
```

```
101
```

In [54]:
```
df5.iloc[:2,:2]
```

Out[54]:

	Names	Rollno
st1	abc	101
st2	xyz	102

Writing any existing Data Frame on a file.(.csv, .xlsx,.js)

In [9]:
```
#writing on a csv file
df5.to_csv(r"C:\desktop\AIML_A.csv")
#provide path where you want to save the data frame file
```

In [8]:
```
#writing on an excel file
df5.to_excel(r"C:\desktop\dataxl.xlsx")
```

```
---------------------------------------------------------------------------
--
ModuleNotFoundError                       Traceback (most recent call las
t)
Input In [8], in <module>
      1 #writing on an excel file
----> 2 df5.to_excel(r"C:\desktop\dataxl.xlsx")

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\c
ore\generic.py:2345, in NDFrame.to_excel(self, excel_writer, sheet_name,
na_rep, float_format, columns, header, index, index_label, startrow, star
tcol, engine, merge_cells, encoding, inf_rep, verbose, freeze_panes, stor
age_options)
   2332 from pandas.io.formats.excel import ExcelFormatter
   2334 formatter = ExcelFormatter(
   2335     df,
   2336     na_rep=na_rep,
   (...)
   2343     inf_rep=inf_rep,
   2344 )
-> 2345 formatter.write(
   2346     excel_writer,
   2347     sheet_name=sheet_name,
   2348     startrow=startrow,
   2349     startcol=startcol,
   2350     freeze_panes=freeze_panes,
   2351     engine=engine,
   2352     storage_options=storage_options,
   2353 )

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\i
o\formats\excel.py:888, in ExcelFormatter.write(self, writer, sheet_name,
startrow, startcol, freeze_panes, engine, storage_options)
    884     need_save = False
    885 else:
    886     # error: Cannot instantiate abstract class 'ExcelWriter' with
abstract
    887     # attributes 'engine', 'save', 'supported_extensions' and 'wr
ite_cells'
--> 888     writer = ExcelWriter(  # type: ignore[abstract]
    889         writer, engine=engine, storage_options=storage_options
    890     )
    891     need_save = True
    893 try:

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\i
o\excel\_openpyxl.py:49, in OpenpyxlWriter.__init__(self, path, engine, d
ate_format, datetime_format, mode, storage_options, if_sheet_exists, engi
ne_kwargs, **kwargs)
     36 def __init__(
     37     self,
     38     path,
   (...)
     47 ):
     48     # Use the openpyxl module as the Excel writer.
---> 49     from openpyxl.workbook import Workbook
     51     engine_kwargs = combine_kwargs(engine_kwargs, kwargs)
```

```
     53     super().__init__(
     54         path,
     55         mode=mode,
    (...)
     58         engine_kwargs=engine_kwargs,
     59     )

ModuleNotFoundError: No module named 'openpyxl'
```

#Reading any csv, xlsx,js files and converting it into DF

In [10]:
```
#reading a csv file
dfc=pd.read_csv(r"C:\desktop\AIML_A.csv")
print(dfc)
```

```
   Unnamed: 0 Names  Rollno  Marks
0           0   abc     101     90
1           1   xyz     102     88
2           2   uvw     103     85
```

In [10]:
```
#reading a csv file
dfcs=pd.read_csv(r"C:\desktop\coursera report.csv")
print(dfcs)
#print(dfcs.to_string())
```

```
        Noida Institute of Engineering and Technology   Unnamed: 1 Unna
med: 2  \
0   19, Institutional Area, Knowledge Park II, Gre...          NaN
NaN
1                                     Coursera Report          NaN
NaN
2                            Mechanical Engineering            NaN
NaN
3                                               NaN          NaN
NaN
4                                             S.No.  AdmissionNo
RollNo
5                                               NaN          NaN
NaN
6                                                 1   0211MEC005
[]
7                                                 2   0211MEC032
[]
8                                                 3   0211MEC019
```

In [ ]:
```
pip install pandas --upgrade pandas
```

Aggregation: It is a function used to apply some aggregation across one or more columns
Frequently used functions: sum: min: max: mean: count:

In [11]:
```
dfc1=pd.read_csv(r"C:\desktop\AIML_A.csv")
```

In [12]:
```
print(dfc1)
```

```
   Unnamed: 0 Names  Rollno  Marks
0           0   abc     101     90
1           1   xyz     102     88
2           2   uvw     103     85
```

In [13]:
```
print(dfc1.Rollno.count())
print(dfc1.Marks.count())
print(dfc1.Marks.sum())
print(dfc1.Marks.max())
print(dfc1.Marks.min())
```

```
3
3
263
90
85
```

In [ ]:
```
#aggregate(): it is used to apply more than one function on desired col
print(dfc1.aggregate([sum,max,min,'mean','count']))
```

In [ ]:
```
#applying aggregate on selected columns and selected functions
print(dfc1.aggregate({'Number':['min','max'],'Age':['max', 'mean'],'Salary'
```

In [ ]:
```
#applying aggregate on selected columns and selected functions
print(dfc1.aggregate(x=('Number','min'),y=('Age','max'),z=('Salary','sum'))
```

In [ ]:
```
#agg(): can also be used for aggregation on complete DF
dfc1.agg('mean', axis='columns')
```

In [ ]:
```
df=pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]],columns=['A','B','C'])
print(df)
```

In [ ]:
```
print(df.agg('mean', axis='columns'))# will provide value of aggfun rowwise
print(df.agg('mean'))# will provide value of aggfun colwise
```

Grouping: groupby() will be used to perform grouping : by this we are refering to a process involving one or more of the following steps: > Splitting the data into groups based on some criteria > Applying a function on each group independently > combining the resuls into a data structure

In [16]:
```
import numpy as np
dfg=pd.DataFrame({
    'A':['foo','bar','foo','bar','bar','foo','foo','bar','foo'],
    'B':['one','one','one','two','three','two','three','one','two'],
    'C':np.random.rand(),
    'D':np.random.randint(-23,16,size=(9))
})
print(dfg)
```

```
     A      B         C   D
0  foo    one  0.093879  13
1  bar    one  0.093879 -20
2  foo    one  0.093879  -2
3  bar    two  0.093879  13
4  bar  three  0.093879 -17
5  foo    two  0.093879  -4
6  foo  three  0.093879   6
7  bar    one  0.093879   4
8  foo    two  0.093879  13
```

In [ ]:
```
#groupby(): is used to create groups among entries of DF based on values ir
g=dfg.groupby('A')
print(g)
```

In [ ]:
```
print(g.groups)# to check the groups inside groupby data frame
```

In [ ]:
```
#iterating through the groups df
for i,j in g:
    print(i)
    print(j)
```

In [ ]:
```
g1=dfg.groupby('B')
```

In [ ]:
```
print(g1.groups)
```

In [ ]:
```
for i,j in g1:
    print(i)
    print(j)
```

In [ ]:
```
#applying aggregation function on groups
print(g.sum())
print(g1.sum())
print(g.max())
print(g1.max())
print(g.min())
print(g1.min())
print(g['C'].max())# can perform agg. on a specific col among all the group
```

In [ ]:
```
# Grouping can be done by multiple columns too.
#it form a hierarichal index
print(dfg)
g2=dfg.groupby(['A','B'])
print(g2.groups)
```

In [ ]:
```
for i,j in g2:
    print(i)
    print(j)
```

In [ ]:
```
#agg on multiple col groups
print(g2.sum())
```

In [ ]:
```
#get_group: is used to access a particular group
g3=g.get_group('foo')
print(g3)
```

Merging two Data Frames

In [19]:
```
sts=pd.DataFrame({
    'Names':['john','mary','henry','akash','vaibhav'],
    'Teams':['alpha','beta','gama','delta','ita']
})
print(sts)
```

```
     Names  Teams
0     john  alpha
1     mary   beta
2    henry   gama
3    akash  delta
4  vaibhav    ita
```

In [20]:
```
st=pd.DataFrame({
    'Names':['john','mary','akash','vaibhav','tony','harsh'],
    'Marks':[56,78,89,67,59,75]
})
print(st)
```

```
     Names  Marks
0     john     56
1     mary     78
2    akash     89
3  vaibhav     67
4     tony     59
5    harsh     75
```

#Types of Merging

inner join outer join left join right join

In [21]:
```
#inner join : is like intersection of dataframes based on a col
#merge(): function is used for merging dfs
rst=pd.merge(sts,st)#by default inner join and first col is considered
print(sts)
print(st)
print(rst)
```

```
     Names  Teams
0     john  alpha
1     mary   beta
2    henry   gama
3    akash  delta
4  vaibhav    ita
     Names  Marks
0     john     56
1     mary     78
2    akash     89
3  vaibhav     67
4     tony     59
5    harsh     75
     Names  Teams  Marks
0     john  alpha     56
1     mary   beta     78
2    akash  delta     89
3  vaibhav    ita     67
```

In [22]:
```
rst=pd.merge(sts,st,on='Names',how='inner')
print(rst)
```

```
     Names  Teams  Marks
0     john  alpha     56
1     mary   beta     78
2    akash  delta     89
3  vaibhav    ita     67
```

In [23]:
```
#outer join: is like union operation
rst=pd.merge(sts,st,on='Names',how='outer')
print(rst)
```

```
     Names  Teams  Marks
0     john  alpha   56.0
1     mary   beta   78.0
2    henry   gama    NaN
3    akash  delta   89.0
4  vaibhav    ita   67.0
5     tony    NaN   59.0
6    harsh    NaN   75.0
```

In [24]:
```
#left join: corresponding to the left dataframe entries
rst=pd.merge(sts,st,on='Names',how='left')
print(rst)
```

```
     Names  Teams  Marks
0     john  alpha   56.0
1     mary   beta   78.0
2    henry   gama    NaN
3    akash  delta   89.0
4  vaibhav    ita   67.0
```

In [ ]:
```
#right join: corresponding to the right dataframe entries
rst=pd.merge(sts,st,on='Names',how='right')
print(rst)
```

In [ ]:
```
#using indicator parameter in merge: set indicator as True then
#it will provide columns existance iorigin in the result of merging
#outer join: is like union operation with indicator
rst=pd.merge(sts,st,on='Names',how='outer',indicator=True)
print(rst)
```

In [ ]:
```
#suffix parameter of merge function
sts['Marks']=[50,55,43,48,67]
print(sts)
```

In [ ]:
```
#outer join: is like union operation
#both DF has common column
rst=pd.merge(sts,st,on='Names',how='outer')
print(rst)
```

In [ ]:
```
#suffix: it will provide name to duplicate cols of DFs
#both DF has common column
rst=pd.merge(sts,st,on='Names',how='outer',suffixes=('_sts','_st'))
print(rst)
```

In [10]:
```
#data Manipulation
#loc and iloc
dfc=pd.read_csv(r'C:\Users\anuj\Desktop\coursera report.csv')
print(dfc)
```

```
        Noida Institute of Engineering and Technology   Unnamed: 1 Unna
med: 2  \
0   19, Institutional Area, Knowledge Park II, Gre...          NaN
NaN
1                                    Coursera Report          NaN
NaN
2                           Mechanical Engineering            NaN
NaN
3                                              NaN          NaN
NaN
4                                            S.No.  AdmissionNo
RollNo
5                                              NaN          NaN
NaN
6                                                1   0211MEC005
[]
7                                                2   0211MEC032
[]
8                                                3   0211MEC019
```

In [ ]:
```
#iloc: it is index(int) based location finder
#select rows and cols based on default index
dfc.iloc[3] #out put in form of a data series
```

In [ ]:
```
dfc.iloc[3]['Team'] #labeling of series generated by iloc
```

In [ ]:
```
#iloc: can access more tahn one row at a time
dfc.iloc[[2,4,5,6]]
```

In [ ]:
```
dfc1=dfc.set_index('Name')
dfc1.head(10)
```

In [ ]:
```
dfc1.iloc[3] #it will still work by ignoring column used as label
```

In [ ]:
```
#dfc1.iloc['John Holland']
#Cannot index by location index with a non-integer key.
```

In [ ]:
```
#slicing: works with iloc
print(dfc.iloc[:4])
```

In [ ]:
```
print(dfc.iloc[:5,1:4])
```

In [ ]:
```
print(dfc.iloc[:5,[4,5]])
```

In [ ]:
```
#loc property:  it is a user defined label based property
# will access a group of rows and col by labeling
print(dfc1.head(10))
```

In [ ]:
```
dfc1.loc['John Holland']
```

In [ ]:
```
dfc1.loc[['John Holland'],['Team','Number']]
# for accessing multiple col values of a located row
```

In [ ]:
```
dfc.loc[2] #it will also work on integer index of non labeled dfs
```

loc	iloc
it is label based.	it is index based

in slicing loc includes end point. iloc does not include end point it uses bot integer/label index. it uses only integer index

In [8]:
```
#slicing with loc
dfc1.loc['John Holland':'Jordan Mickey'] #end point is inclusive
```

```
---------------------------------------------------------------------
--
NameError                                 Traceback (most recent call las
t)
Input In [8], in <module>
      1 #slicing with loc
----> 2 dfc1.loc['John Holland':'Jordan Mickey']

NameError: name 'dfc1' is not defined
```

In [ ]:
```
dfc1.Salary>2500000 #boolean series
```

In [ ]:
```
#all the data of employees having salary greater than 2500000
dfc1.loc[dfc1.Salary>2500000]
```

In [7]:
```
#All the data of employees having salary greater than 2500000
#and weight equal to 180
dfc1.loc[(dfc1.Salary>2500000)&(dfc1.Weight>200)]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call las
t)
Input In [7], in <module>
      1 #All the data of employees having salary greater than 2500000
      2 #and weight equal to 180
----> 3 dfc1.loc[(dfc1.Salary>2500000)&(dfc1.Weight>200)]

NameError: name 'dfc1' is not defined
```

In [ ]:
```
#Adding new rows in existing DF using loc
print(dfc)
dfc.loc[458]=[1,2,3,4,5,6,7,8,9]
print(dfc)
```

In [ ]:
```
#Apply Function on DFs
students=pd.DataFrame({
    'Name':['ram','shyam','raju','sonu','golu','harsh','nikhil','abhi'],
    'Marks1':[50,45,65,76,43,89,64,93],
    'Marks2':[30,40,50,60,70,80,90,20]
})
print(students)
```

In [ ]:
```
#Apply a function to check
#if a person is fail or pass according to Marks1 along with axis
#you have to pass function name as a parameter in apply and set axis.
def check(row):
    if row['Marks1']>=50:
        return 'Pass'
    else:
        return 'Fail'
students['Result']=students.apply(check,axis=1)
print(students)
```

In [ ]:
```
#check whether names of a DF belongs to a list or not
def names(row):
    if(row['Name'] in ['ram','raju','sonu','nikhil','abhi']):
        return 'yes'
    else:
        return 'no'
students['flag']=students.apply(names,axis=1)
print(students)
```

In [ ]:
```
#Insert a col at the end having grades of student according to their marks
def grade(row):
    n=row['Marks1']+row['Marks2']
    p=(n*100)/200
    if p>=90:
        return 'A+'
    elif p>=80:
        return 'A'
    elif p>=70:
        return 'B+'
    elif p>=60:
        return 'B'
    elif p>=50:
        return 'C'
    elif p>=40:
        return 'D'
    else:
        return 'FAIL'

students['Grade']=students.apply(grade,axis=1)
print(students)
```

In [ ]:
```
students['Address']=['noida','delhi','gzb',np.nan,np.nan,'delhi','gzb',np.r
print(students)
```

In [ ]:
```
#isnull(): it will return true if value contained is null
students.isnull()
```

In [ ]:
```
#notnull(): reverse of isnull
students.notnull()
```

In [ ]:
```
#fillna(): is used to write a value inplace of null
students['Address'].fillna(value='Unknown',inplace=True)
print(students)
```

In [ ]:
```
#describe(): statistical data table
students.describe()
```

In [ ]:

```
#End of pandas
#https://pandas.pydata.org/pandas-docs
```