Python Pandas की समीक्षा

सीबीएसई पाठ्यक्रम पर आधारित

इन्फार्मेटिक्स प्रैक्टिसेज कक्षा -12

अध्याय -1

द्वारा:
संजीव भदौरिया
स्नातकोत्तर शिक्षक (संगणक विज्ञान)
के० वि० बाराबंकी (लखनऊ संभाग)

Python Pandas (एक नज़र)

- Data को analysis करने का एक अत्यंत महत्वपूर्ण हिस्सा होता है- Data Processing क्यों कि data हमेशा इच्छित format में नहीं होता है |
- Data को analysis करने से पहले बहुत सारी processing की ज़रुरत होती है जैसे - Cleaning, Restructuring या merging इत्यादि |
- Data को fast process करने के लिए python में बहुत सारे tools उपलब्ध हैं – जैसे - Numpy, Scipy, Cython और Pandas.
- Pandas को Numpy के ऊपर रखा गया है |
- इस ध्याय में हम Python Pandas Data Series और DataFrames के basic concepts सीखेंगे जिन्हें हमने कक्षा - 11 में भी सीखा था |

Python Pandas

- Pandas एक open-source python की library है जो अपने powerful data-structure का प्रयोग करके data manipulation और उसको analysis करने की सुविधा प्रदान करती है |
- इनकी performance बहुत अच्छी होती है |
- Pandas विभिन्न प्रकार के data को process करने के लिए कई functions प्रदान करता है |
- Data analysis करते समय यह बहुत ध्यान रखना होता है की आप सही data type को प्रयोग कर रहे हैं अन्यथा आपको कई errors का सामना करना पड सकता है | pandas के द्वारा निम्न data type पर आसानी से काम किया जा सकता है |

Pandas dtype	Python type	NumPy type	Usage
object	str	string_, unicode_	Text
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64	Integer numbers
float64	float	float_, float16, float32, float64	Floating point numbers
bool	bool	bool_	True/False values
datetime64	NA	datetime64[ns]	Date and time values
timedelta[ns]	NA	NA	Differences between two datetimes
category	NA	NA	Finite list of text values

Pandas Series

- Pandas की primary building block, ***Series*** ही होती है |
- ***Series*** एक labeled ***One-Dimensional Array*** होती है जो किसी भी data type को hold कर सकती है |
- Series का data हमेशा mutable होता है अर्थात इनको बदला जा सकता है |
- लेकिन Series के data का size immutable होता है |
- यह एक प्रकार से 2 arrays का structure प्रतीत होता है जिसमे एक index होता है और दूसरे में actual values.
- Series में row labels को ***index*** कहा जाता है |
- निम्न उदहारण data types हैं जिनसे series बनाई जा सकती है और ऐसा करने के लिए ***series()*** का प्रयोग किया जाता है |

```
Num = [23, 54, 34, 44, 35, 66, 27, 88, 69, 54]   # a list with homogeneous data
Emp = ['A V Raman', 35, 'Finance', 45670.00]   # a list with heterogeneous data
Marks = {"ELENA JOSE" : 450, "PARAS GUPTA" : 467, "JOEFFIN JOSEPH" : 480} # a dictionary
Num1 = (23, 54, 34, 44, 35, 66, 27, 88, 69, 54) # a tuple with homogeneous data
Std = ('AKYHA KUMAR', 78.0, 79.0, 89.0, 88.0, 91.0)      # a list with heterogeneous data
```

Series Objects को बनाना

– एक series type object को कई तरीकों से बनाया जा सकता है |

1. Series ( ) फंक्शन का प्रयोग करके -

<Series Object> = pandas.Series( ) यह empty series बनाएगा |

```
>>> import pandas as pd
>>> ob = pd.Series()
>>> ob
Series([], dtype: float64)
```

2. Non-empty series बनाना –

Import pandas as pd

<Series Object> = pd.Series(data, index=idx) जहाँ data कोई भी python sequence, ndarray, python dictionary या scaler value हो सकता है |

Index

```
>>> import pandas as pd
>>> ob = pd.Series(range(5))
>>> ob
0    0
1    1
2    2
3    3
4    4
dtype: int64
```

Index

```
>>> import pandas as pd
>>> obj=pd.Series([3,5,4,4.5])
>>> obj
0    3.0
1    5.0
2    4.0
3    4.5
dtype: float64
```

Series Objects को बनाना

1. Dictionary के साथ series बनाना

Index of Keys

```
>>> import pandas as pd
>>> obj=pd.Series({'Jan':31,'Feb':28,'Mar':31})
>>> obj
Jan    31
Feb    28
Mar    31
dtype: int64
```

2. Scalar value के साथ series बनाना -

```
>>> import pandas as pd
>>> a=pd.Series(10,index=range(0,3))
>>> a
0    10
1    10
2    10
dtype: int64
```

```
>>> import pandas as pd
>>> b=pd.Series(15,index=range(1,6,2))
>>> b
1    15
3    15
5    15
dtype: int64
```

```
>>> import pandas as pd
>>> c=pd.Series('Welcome to BBK', index=['Hema','Rahul','Anup'])
>>> c
Hema     Welcome to BBK
Rahul    Welcome to BBK
Anup     Welcome to BBK
dtype: object
```

Series Object Attributes

3. कुछ common attributes निम्न हैं-

<series object>.<AttributeName>

Attribute	Description
Series.index	Returns index of the series
Series.values	Returns ndarray
Series.dtype	Returns dtype object of the underlying data
Series.shape	Returns tuple of the shape of underlying data
Series.nbytes	Return number of bytes of underlying data
Series.ndim	Returns the number of dimention
Series.size	Returns number of elements
Series.intemsize	Returns the size of the dtype
Series.hasnans	Returns true if there are any NaN
Series.empty	Returns true if series object is empty

Series Object Attributes

```
>>> import pandas as pd
>>> s=pd.Series(range(1,15,3), index=[x for x in 'abcde'])
>>> s.index
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
>>> s.values
array([ 1,  4,  7, 10, 13], dtype=int64)
>>> s.shape
(5,)
>>> s.size
5
>>> s.nbytes
40
>>> s.ndim
1
>>> s.itemsize
```

Series Object को access करना

```
>>> import pandas as pd
>>> import numpy as np
>>> a=np.arange(9,13)
>>> ob=pd.Series(index=a, data=a**2)
>>> ob
9      81
10    100
11    121
12    144
dtype: int32
>>> ob[10]
100
```

ऑब्जेक्ट की values को print करना

Individual value को print करना

```
>>> ob[2:4]
11    121
12    144
dtype: int32
>>> ob[1:]
10    100
11    121
12    144
dtype: int32
>>> ob[0::2]
9      81
11    121
dtype: int32
```

Objects की slicing

```
>>> ob[::-1]
12    144
11    121
10    100
9      81
dtype: int32
```

Objects की slicing करना बहुत आसान है बस निम्न syntax का अनुसरण करिए

<objectName>[<start>:<stop>:<step >]

head() और tail () Function

1. head(<n>) function शुरू से n elements return करता है यदि n न दे तो अपने आप शुरू के 5 record दिखायेगा |
2. tail(<n>) function आखिरी से n elements return करता है यदि n न दे तो अपने आप आखिरी के 5 record दिखायेगा |

```
>>> import pandas as pd
>>> import math
>>> s=pd.Series(data=[math.sqrt(x) for x in range(1,10)],index=[x for x in range
(1,10)])
```

```
>>> s
1    1.000000
2    1.414214
3    1.732051
4    2.000000
5    2.236068
6    2.449490
7    2.645751
8    2.828427
9    3.000000
dtype: float64
```

```
>>> s.head(6)
1    1.000000
2    1.414214
3    1.732051
4    2.000000
5    2.236068
6    2.449490
dtype: float64
```

```
>>> s.tail(7)
3    1.732051
4    2.000000
5    2.236068
6    2.449490
7    2.645751
8    2.828427
9    3.000000
dtype: float64
```

```
>>> s.head()
1    1.000000
2    1.414214
3    1.732051
4    2.000000
5    2.236068
dtype: float64
>>> s.tail()
5    2.236068
6    2.449490
7    2.645751
8    2.828427
9    3.000000
dtype: float64
```

NumPy array और Series objects में अंतर

1. ndarray के case में आप तभी vector operation कर सकते हैं जब दोनों ndarray के shape सामान हों | जबकि series ऑब्जेक्ट के केस में अगर मैचिंग index के साथ ही align होगा अन्यथा NaN return होता है |

```
>>> import numpy as np
>>> a=np.array([1,2,3])
>>> b=np.array([1,2,3,45,5])
>>> a+b
Traceback (most recent call last):
  File "<pyshell#143>", line 1, in <module>
    a+b
ValueError: operands could not be broadcast together with shapes (3,) (5,)
```

2. ndarray में index हमेशा 0 से शुरू होता है और सदैव numeric ही होगा |लेकिन series में numbers के आलावा किसी भी type का index हो सकता है और ज़रूरी नहीं की index 0 से शुरू हो |

DataFrame

- Pandas का मुख्य object *DataFrame* होता है | और यह pandas का सबसे अधिक प्रयोग किया जाने वाला Data Structure है |
- *DataFrame* एक *Two -Dimensional Array* होता है जो किसी भी data type को hold कर सकती है | और यह tabular format में data को store करता है |
- Finance, Statistics, Social Science और कई engineering branch में इसका प्रयोग अधिकता में किया जाता है |
- DataFrame में data और इसका size दोनों ही mutable होते हैं अर्थात इन्हें बदला जा सकता है |
- DataFrame में दो विभिन्न indexes होते हैं - *row index* और *column index* |

A DataFrame with two-dimensional array with heterogeneous data.

Country	Population	BirthRate	UpdateDate
China	1,379,750,000	14.00	2016-08-11
India	1,330,780,000	21.76	2016-08-11
United States	324,882,000	13.82	2016-08-11
Indonesia	260,581,000	18.84	2016-01-07
Brazil	206,918,000	18.43	2016-08-11
Pakistan	194,754,000	27.62	2016-08-11

DataFrame को बनाना और प्रदर्शित करना

- 2 D फॉर्मेट में डाटा को pass करके एक DataFrame object बनाया जा सकता है |

```
import pandas as pd
<dataFrameObject> = pd.DataFrame(<a 2D Data Structure>,\ [columns=<column sequence>],[index=<index sequence>])
```

- आप DataFrame को कई तरीके से data values pass करके बना सकते हैं जैसे –
  - 2D dictionaries
  - 2D ndarrays
  - Series type object
  - Another DataFrame object

2D Dictionary से DataFrame बनाना

A. List या ndarrays की dictionary से DataFrame बनाना |

```
>>> import pandas as pd
>>> dict={'Students':['Pratibha','Ritika','Saumya','Aryan','Keshwam','Priyanka']
,'Marks':[69,65,64,59,59,40], 'Sports':['TQ','TT','KB','VB','CR','KO']}
>>> dtf=pd.DataFrame(dict)
>>> dtf
   Students  Marks Sports
0  Pratibha     69     TQ
1    Ritika     65     TT
2    Saumya     64     KB
3     Aryan     59     VB
4   Keshwam     59     CR
5  Priyanka     40     KO
```

उपरोक्त उदाहरण में index स्वतः 0 से 5 तक आगये तथा column के name स्वतः वाही आये जो dictionary में keys थीं |

np.range(n) का प्रयोग करके indexes स्वतः निर्मित हो गयीं |

2D Dictionary के keys column के नाम हो गए

```
>>> import pandas as pd
>>> dict={'Students':['Pratibha','Ritika','Saumya','Aryan','Keshwam','Priyanka'
,'Marks':[69,65,64,59,59,40], 'Sports':['TQ','TT','KB','VB','CR','KO']}
>>> dtf=pd.DataFrame(dict,index=['I','II','III','IV','V','VI'])
>>> dtf
     Students  Marks Sports
I    Pratibha     69     TQ
II     Ritika     65     TT
III    Saumya     64     KB
IV      Aryan     59     VB
V     Keshwam     59     CR
VI   Priyanka     40     KO
```

यहाँ indexes आपने specify किये हैं |

अर्थात यदि आप index का sequence देते हैं तो index आपके द्वारा दिया गया ही set होगा अन्यथा वह स्वतः 0 से n-1 तक का index लेलेगा |

2D Dictionary से DataFrame बनाना

B. Dictionaries की dictionary से DataFrame बनाना |

```
>>> yr2015={'Qtr1':40000, 'Qtr2':35000,'Qtr3': 47000,'Qtr4':45000}
>>> yr2016={'Qtr1':42000, 'Qtr2':37000,'Qtr3': 49000,'Qtr4':47000}
>>> yr2017={'Qtr1':43000, 'Qtr2':38000,'Qtr3': 50000,'Qtr4':48000}
>>> kvfee={2015:yr2015,2016:yr2016,2017:yr2017}
>>> dtFee=pd.DataFrame(kvfee)
>>> dtFee
       2015   2016   2017
Qtr1  40000  42000  43000
Qtr2  35000  37000  38000
Qtr3  47000  49000  50000
Qtr4  45000  47000  48000
```

यह 2D Dictionary है जो ऊपर की dictionaries से मिलकर बनी है |

DataFrame ऑब्जेक्ट create हो गया |

यहाँ आप index और column name देख कर समझ सकते हैं की कैसे assign हुए

यदि यहाँ yr2015, yr2016 और yr2017 के keys अलग होते तो dataframe के rows और column ज्यादा बढ़ जाते और बिना match वाले row और column में NaN store हो जाता |

2D ndarray से DataFrame बनाना

```
>>> import pandas as pd
>>> import numpy as np
>>> narr=np.array([[1,2,3],[4,5,6]],np.int32)
>>> narr.shape
(2, 3)
>>> dtf=pd.DataFrame(narr)
>>> dtf
   0  1  2
0  1  2  3
1  4  5  6
```

इसमें column name और index स्वतः अगये|

```
>>> import pandas as pd
>>> import numpy as np
>>> narr=np.array([[1,2,3],[4,5,6]],np.int32)
>>> dtf=pd.DataFrame(narr, columns=['One','Two','Three'])
>>> dtf
   One  Two  Three
0    1    2      3
1    4    5      6
```

इसमें column name user ने दिए हैं |

```
>>> dtf=pd.DataFrame(narr, columns=['One','Two','Three'],index=['A','B'])
>>> dtf
   One  Two  Three
A    1    2      3
B    4    5      6
```

इसमें column name तथा index दोनों user ने दिए हैं |

Series Object की 2D Dictionary से DataFrame बनाना

```
>>> import pandas as pd
>>> population=pd.Series([35,39,34,64],index=['Class12','Class11','Class10','Cla
ss9'])
>>> AvgMarks=pd.Series([350,390,340,400],index=['Class12','Class11','Class10','C
lass9'])
>>> dict={0:population,1:AvgMarks}
>>> dtf=pd.DataFrame(dict)
>>> dtf
          0    1
Class12  35  350
Class11  39  390
Class10  34  340
Class9   64  400
```

यह 2D Dictionary है जो ऊपर की Series से मिलकर बनी है |

DataFrame ऑब्जेक्ट create हो गया |

```
>>> dict={'Population':population,'AverageMarks':AvgMarks}
>>> dtf=pd.DataFrame(dict)
>>> dtf
         Population  AverageMarks
Class12          35           350
Class11          39           390
Class10          34           340
Class9           64           400
```

DataFrame ऑब्जेक्ट को इस प्रकार भी बनाया जा सकता है |

दूसरे DataFrame ऑब्जेक्ट से DataFrame बनाना

```
>>> import pandas as pd
>>> import numpy as np
>>> narr=np.array([[1,2,3],[4,5,6]])
>>> dtf=pd.DataFrame(narr,columns=['first','Second','Third'],index=['A','B'])
>>> dtf
   first  Second  Third
A      1       2      3
B      4       5      6
>>> dtf2=pd.DataFrame(dtf)
>>> dtf2
   first  Second  Third
A      1       2      3
B      4       5      6
```

DataFrame ऑब्जेक्ट को दूसरे DataFrame ऑब्जेक्ट से बनाया गया है|

DataFrame ऑब्जेक्ट को display करना|

```
>>> dtf
   first  Second  Third
A      1       2      3
B      4       5      6
>>> dtf2=pd.DataFrame(dtf)
>>> dtf2
   first  Second  Third
A      1       2      3
B      4       5      6
```

DataFrame ऑब्जेक्ट को display करने का syntax ये है |

DataFrame Attributes

- जब आप एक DataFrame ऑब्जेक्ट बनाते हैं तो इससे सम्बंधित समस्त सूचना जैसे - size, इसका datatype इत्यादि , attributes के द्वारा प्राप्त किये जा सकते हैं |

  <DataFrame Object>.<attribute name>

- कुछ attributes निम्न हैं -

Attribute	Description
index	यह dataframe के index मतलब (row lebels) को दिखता है |
columns	यह dataframe के column lebels को दिखता है |
axes	यह दोनों axes अर्थात index और column को return करता है |
dtypes	यह dataframe के अन्दर रखे डाटा का datatype return करता है|
size	ऑब्जेक्ट में उपस्थित elements की संख्या return करता है |
shape	यह dataframe की dimention की tuple return करता है |
values	यह dataframe का numpy रूप return करता है |
empty	यह एक सूचक है की dataframe empty है या नहीं|
ndim	यह axes/array की dimention को return करता है |
T	यह index और column को transpose कर देता है|

DataFrame Attributes

```
>>> dtf.index
Index(['A', 'B'], dtype='object')
>>> dtf.columns
Index(['first', 'Second', 'Third'], dtype='object')
>>> dtf.axes
[Index(['A', 'B'], dtype='object'), Index(['first', 'Second', 'Third'], dtype='object')
]
>>> dtf.dtypes
first     int32
Second    int32
Third     int32
dtype: object
>>> dtf.size
6
>>> dtf.shape
(2, 3)
>>> dtf.ndim
2
```

```
>>> dtf.empty
False
>>> dtf.count()
first     2
Second    2
Third     2
dtype: int64
>>> dtf.T
          A  B
first     1  4
Second    2  5
Third     3  6
```

```
>>> dtf.values
array([[1, 2, 3],
       [4, 5, 6]])
```

DataFrame से Selecting और Accessing

- Column को select करना

<DataFrame Object>[<column name>]

या <DataFrame Object>.<column name>

एक column को select करने के लिए

<DataFrame Object>[column name की list]

कई column को select करने के लिए

```
>>> dtf.first
<bound method NDFrame.first of    first  Second  Third
A       1       2       3
B       4       5       6>
>>> dtf['Second']
A    2
B    5
Name: Second, dtype: int32
```

```
>>> dtf[['Second','first']]
   Second  first
A       2      1
B       5      4
```

कई column में हम क्रम बदल सकते हैं|

DataFrame से subset को select करना

<DataFrameObject>.loc [<StartRow> : <EndRow>, <StartCol> : <EndCol>]

```
>>> dtf
           Population  Avg Income  Per Capita Income
Delhi            1001       45000           44.955045
Mumbai           2005       56000           27.930175
Chennai         30236       57000            1.885170
Kolkata          4662       46000            9.867010
```

```
>>> dtf.loc['Delhi',:]
Population              1001.000000
Avg Income             45000.000000
Per Capita Income         44.955045
Name: Delhi, dtype: float64
```

```
>>> dtf.loc[:,'Population':'Per Capita Income']
           Population  Avg Income  Per Capita Income
Delhi            1001       45000           44.955045
Mumbai           2005       56000           27.930175
Chennai         30236       57000            1.885170
Kolkata          4662       46000            9.867010
```

```
>>> dtf.loc['Mumbai':'Kolkata',:]
           Population  Avg Income  Per Capita Income
Mumbai           2005       56000           27.930175
Chennai         30236       57000            1.885170
Kolkata          4662       46000            9.867010
```

```
>>> dtf.loc['Delhi':'Mumbai','Population':'Avg Income']
           Population  Avg Income
Delhi            1001       45000
Mumbai           2005       56000
```

DataFrame से subset को select करना

<DataFrameObject> .iloc [<Row Index> : <RowIndex>, <ColIndex> : <ColIndex>]

```
>>> dtf.iloc[0:2,1:3]
        Avg Income  Per Capita Income
Delhi        45000          44.955045
Mumbai       56000          27.930175
```

```
>>> dtf.iloc[0:2,1:2]
        Avg Income
Delhi        45000
Mumbai       56000
```

DataFrame से Individual Value को select करना

<DFObject>. <col name.[row name or row index]

या

<DFObject> . at [<row name>,<col name>]

या

<DFObject> iat[<row index>, <col index>]

```
>>> dtf
   One  Two  Three
A    1    2      3
B    4    5      6
>>> dtf.Two['B']
5
```

```
>>> dtf.Two[0]
2
```

```
>>> dtf.at['A','Three']
3
>>> dtf.iat[1,2]
6
```

DataFrame में values को access करना व् modify करना

a) नए column को change या add करने के लिए निम्न syntax का प्रयोग करें |

<DFObject>.<Col Name>[<row label>]=<new value>

```
>>> dtf
   One  Two  Three
A    1    2      3
B    4    5      6
C    7    8      9
>>> dtf['Four']=44
>>> dtf
   One  Two  Three  Four
A    1    2      3    44
B    4    5      6    44
C    7    8      9    44
```

चूँकि इसमें 'Four' नाम से कोई column नहीं है तो नया column add हो गया |

चूँकि इसमें 'Four' नाम से column है तो column के मानों में बदलाव हो गया |

```
>>> dtf['Four']=66
>>> dtf
   One  Two  Three  Four
A    1    2      3    66
B    4    5      6    66
C    7    8      9    66
>>>
```

DataFrame में values को access करना व् modify करना

b) नए row को change या add करने के लिए निम्न syntax का प्रयोग करें |

<DFObject> at[<RowName>, :] =<new value>

या

<DFObject> loc[<RowName>, :] =<new value>

```
>>> dtf.at['D',:]=88
>>> dtf
    One   Two  Three  Four
A   1.0   2.0    3.0  66.0
B   4.0   5.0    6.0  66.0
C   7.0   8.0    9.0  66.0
D  88.0  88.0   88.0  88.0
```

चूँकि इसमें 'D' नाम से कोई row नहीं थी तो नयी row add हो गयी |

```
>>> dtf.at['D',:]=99
>>> dtf
    One   Two  Three  Four
A   1.0   2.0    3.0  66.0
B   4.0   5.0    6.0  66.0
C   7.0   8.0    9.0  66.0
D  99.0  99.0   99.0  99.0
```

चूँकि इसमें 'D' नाम से row नहीं तो row की values change हो गयी |

DataFrame में values को access करना व् modify करना

c) Single value को change करने के लिए निम्न syntax का प्रयोग करें |

<DFObject>.<ColName>[<RowName/Lebel>]

```
>>> dtf
    One   Two  Three  Four
A   1.0   2.0    3.0  66.0
B   4.0   5.0    6.0  66.0
C   7.0   8.0    9.0  66.0
D  99.0  99.0   99.0  99.0
>>> dtf.Three['D']=100
>>> dtf
    One   Two  Three  Four
A   1.0   2.0    3.0  66.0
B   4.0   5.0    6.0  66.0
C   7.0   8.0    9.0  66.0
D  99.0  99.0  100.0  99.0
```

इसमें D वाले row में 'Three' वाले column में value बदल गयी |

```
>>> dtf['Four']=[10,11,12,13]
>>> dtf.at['D']=[13,14,15,16]
>>> dtf
    One   Two  Three  Four
A   1.0   2.0    3.0    10
B   4.0   5.0    6.0    11
C   7.0   8.0    9.0    12
D  13.0  14.0   15.0    16
```

Values को ऐसे भी change किया जा सकता है | जिसमे row या column के अलग अलग values भी दिए जा सकते हैं |

DataFrame में values को access करना व् modify करना

Column को delete करने के लिए निम्न syntax का प्रयोग करें |

del <DFObject>[<ColName>] या

df.drop([<Col1Name>,<Col2Name>, . .], axis=1)

```
>>> del dtf['Four']
>>> dtf
    One   Two  Three
A   1.0   2.0    3.0
B   4.0   5.0    6.0
C   7.0   8.0    9.0
D  13.0  14.0   15.0
>>> dtf.drop(['Two','Three'],axis=1)
    One
A   1.0
B   4.0
C   7.0
D  13.0
```

axis =1 यह बताता है की column को delete करना है

del कमांड delete करने के बाद value return नहीं करता है जबकि drop method delete करने के बाद तुरंत बची हुई dataframe को return करता है|

DataFrame में Iteration

- कभी कभी हमें पूरे dataframe पर iteration करना पड़ता है ऐसे में अलग से values को access करने के लिए code लिखना और समस्यात्मक हो जाता है | इसलिए dataframe पर itration करना आवश्यक हो जाता है जिसे हम निम्न तरीके से कर सकते है |

- <DFObject>.iterrows( ) यह dataframe को row-wise subsets में देखता है

- <DFObject>.iteritems( ) यह dataframe को column-wise subsets देखता है|

pandas.iterrows () function का प्रयोग

iter.py - C:/Users/KVBBKServer/AppData/Local/Programs/Python/Python36/iter.py (3.6.5)

File Edit Format Run Options Window Help

```
import pandas as pd
disales={2015:{'Qtr1':34500,'Qtr2':56000,'Qtr3':47000,'Qtr4':49000},\
         2016:{'Qtr1':44500,'Qtr2':46100,'Qtr3':57000,'Qtr4':59000},\
         2017:{'Qtr1':54500,'Qtr2':51000,'Qtr3':47000,'Qtr4':58500}}
df1=pd.DataFrame(disales)
for (row,rowSeries) in df1.iterrows():
    print("RowIndex : ",row)
    print("Containing : ")
    print(rowSeries)
```

```
>>> df1
       2015   2016   2017
Qtr1  34500  44500  54500
Qtr2  56000  46100  51000
Qtr3  47000  57000  47000
Qtr4  49000  59000  58500
```

ये df1 की values हैं जिसे एक एक करके ऐसे प्रोसेस किया गया है|

```
RowIndex :  Qtr1
Containing :
2015    34500
2016    44500
2017    54500
Name: Qtr1, dtype: int64
RowIndex :  Qtr2
Containing :
2015    56000
2016    46100
2017    51000
Name: Qtr2, dtype: int64
RowIndex :  Qtr3
Containing :
2015    47000
2016    57000
2017    47000
Name: Qtr3, dtype: int64
RowIndex :  Qtr4
Containing :
2015    49000
2016    59000
2017    58500
Name: Qtr4, dtype: int64
```

DataFrame बनने के बाद नीचे वाले code को try करिए |

```
for (row,rowSeries) in df1.iterrows():
    print("RowIndex : ",row)
    print("Containing : ")
    i=0
    for val in rowSeries:
        print("At",i,"Position:",val)
        i=i+1
```

pandas.iteritems() function का प्रयोग

```
import pandas as pd
disales={2015:{'Qtr1':34500,'Qtr2':56000,'Qtr3':47000,'Qtr4':49000},\
         2016:{'Qtr1':44500,'Qtr2':46100,'Qtr3':57000,'Qtr4':59000},\
         2017:{'Qtr1':54500,'Qtr2':51000,'Qtr3':47000,'Qtr4':58500}}
df1=pd.DataFrame(disales)
for (col,colSeries) in df1.iteritems():
    print("Column Index : ", col)
    print("Containing : ")
    print(colSeries)
```

```
>>> df1
       2015   2016   2017
Qtr1  34500  44500  54500
Qtr2  56000  46100  51000
Qtr3  47000  57000  47000
Qtr4  49000  59000  58500
```

ये df1 की values हैं जिसे एक एक करके ऐसे प्रोसेस किया गया है|

```
Column Index :  2015
Containing :
Qtr1    34500
Qtr2    56000
Qtr3    47000
Qtr4    49000
Name: 2015, dtype: int64
Column Index :  2016
Containing :
Qtr1    44500
Qtr2    46100
Qtr3    57000
Qtr4    59000
Name: 2016, dtype: int64
Column Index :  2017
Containing :
Qtr1    54500
Qtr2    51000
Qtr3    47000
Qtr4    58500
Name: 2017, dtype: int64
>>>
```

DataFrame बनने के बाद नीचे वाले code को try करिए |

```
df1=pd.DataFrame(disales)
for (col,colSeries) in df1.iteritems():
    print("Column Index : ", col)
    print("Containing : ")
    i=0
    for val in colSeries:
        print("At row",i," : ", val)
        i=i+1
```

Program for iteration

- Write a program to iterate over a dataframe containing names and marks, then calculates grades as per marks (as per guideline below) and adds them to the grade column.

Marks > =90	Grade A+
Marks 70 – 90	Grade A
Marks 60 – 70	Grade B
Marks 50 – 60	Grade C
Marks 40 – 50	Grade D
Marks < 40	Grade F

Program for iteration

```
import pandas as pd
import numpy as np
names=pd.Series(['Sanjeev','Rajeev','Sanjay','Abhay'])
marks=pd.Series([76,86,55,54])
stud={'Name':names,'Marks':marks}
df=pd.DataFrame(stud,columns=['Name','Marks'])
df['Grade']=np.NaN        #this will add NaN to all records of dataframe
print("Initial values in DataFrame")
print(df)
for (col,colSeries) in df.iteritems():
    length=len(colSeries)
    if col=='Marks':
        lstMrks=[]
        for row in range(length):
            mrks=colSeries[row]
            if mrks>=90:
                lstMrks.append('A+')
            elif mrks>=70:
                lstMrks.append('A')
            elif mrks>=60:
                lstMrks.append('B')
            elif mrks>=50:
                lstMrks.append('C')
            elif mrks>=40:
                lstMrks.append('D')
            else:
                lstMrks.append('F')
df['Grade']=lstMrks
print("\n\nDataFrame after calculation of Grades")
print(df)
```

```
Initial values in DataFrame
      Name  Marks  Grade
0  Sanjeev     76    NaN
1   Rajeev     86    NaN
2   Sanjay     55    NaN
3    Abhay     54    NaN


DataFrame after calculation of Grades
      Name  Marks Grade
0  Sanjeev     76     A
1   Rajeev     86     A
2   Sanjay     55     C
3    Abhay     54     C
```

कुछ अन्य ज़रूरी functions

DataFrame में कुछ ज़रूरी functions निम्न हैं-

<DF>.info ( )

<DF>.describe ( )

```
>>> df1
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
>>> df2
    A   B   C
0  10  20  30
1  40  50  60
2  70  80  90
```

```
>>> df3
     A    B    C
0  100  200  300
1  400  500  600
```

```
>>> df4
      A     B
0  1000  2000
1  3000  4000
2  5000  6000
```

```
>>> df1.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
A    3 non-null int32
B    3 non-null int32
C    3 non-null int32
dtypes: int32(3)
memory usage: 116.0 bytes
```

```
>>> df1.describe()
              A    B    C
count   3.0  3.0  3.0
mean    4.0  5.0  6.0
std     3.0  3.0  3.0
min     1.0  2.0  3.0
25%     2.5  3.5  4.5
50%     4.0  5.0  6.0
75%     5.5  6.5  7.5
max     7.0  8.0  9.0
```

कुछ अन्य ज़रूरी functions

DataFrame में कुछ ज़रूरी functions निम्न हैं-

<DF>.head ([n=<n>]) यहाँ n की default value 5 होती है |

<DF>.tail ([n=<n>])

```
>>> df1
   A   B   C
0  1   2   3
1  4   5   6
2  7   8   9
3  10  20  30
4  40  50  60
5  70  80  90
6  100 200 300
7  400 500 600
8  700 800 900
```

```
>>> df1.head()
   A   B   C
0  1   2   3
1  4   5   6
2  7   8   9
3  10  20  30
4  40  50  60
>>> df1.tail()
   A   B   C
4  40  50  60
5  70  80  90
6  100 200 300
7  400 500 600
8  700 800 900
```

```
>>> df1.head(n=3)
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
>>> df1.tail(n=4)
   A   B   C
5  70  80  90
6  100 200 300
7  400 500 600
8  700 800 900
```

Cumulative Calculations Functions

DataFrame में cumulative sum के लिए निम्न फंक्शन है -

<DF>.cumsum([axis = None]) यहाँ axis argument वैकल्पिक है |

```
>>> df1
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
>>> df1.cumsum()
    A   B   C
0   1   2   3
1   5   7   9
2  12  15  18
```

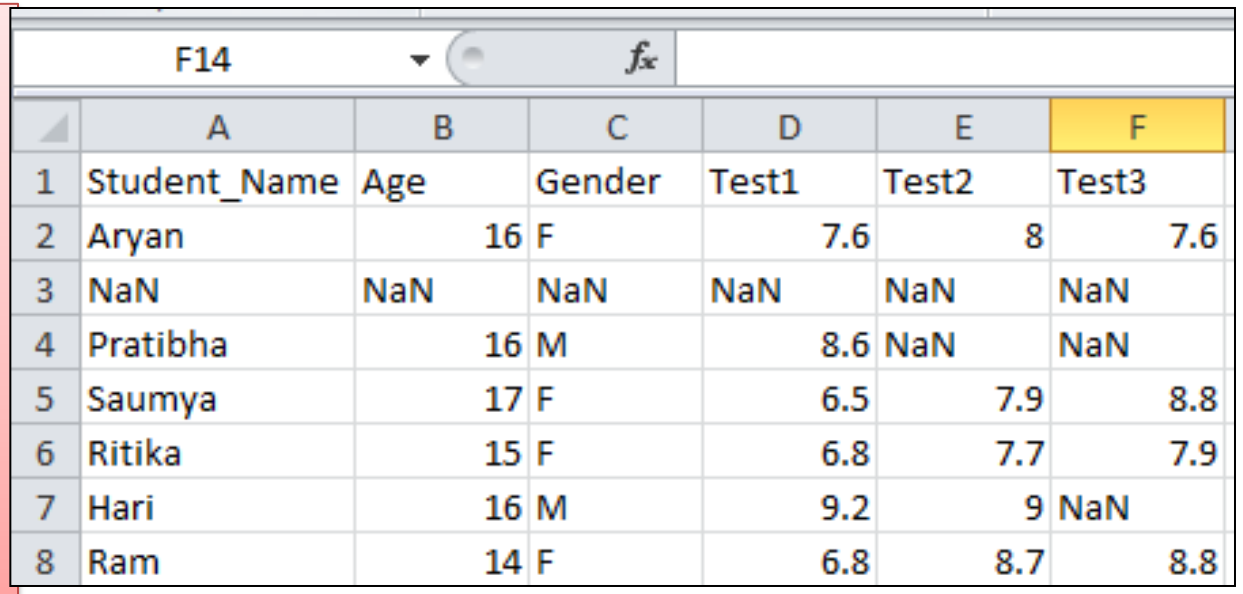
```
>>> df1.cumsum(axis='rows')
    A   B   C
0   1   2   3
1   5   7   9
2  12  15  18
>>> df1.cumsum(axis='columns')
   A   B   C
0  1   3   6
1  4   9  15
2  7  15  24
```

Data Aggregation Functions

- Data analysis के लिए पाइथन एक बहुत अच्छी कंप्यूटर भाषा है |
- Python pandas इस प्रकार के data को analyze करने के लिए कई प्रकार के data aggregation function प्रदान करता है |
- Analysis का एक महत्वपूर्ण कार्य होता है बड़े dataset का summarization जिसे aggregations को compute करना कहते हैं जैसे- sum(), mean(), median(), min() और max() जिसमे विशाल dataset से एक संख्या मिल जाती है |
- इस अध्याय में हम इन्ही data aggregate functions का प्रयोग करना सीखेंगे |
- Aggregation का अर्थ होता है एक विशाल dataset की values को process करके एक value प्राप्त करना |
- Data aggregation में हमेशा multivalued functions दिए जाते हैं जो एक single value return करते हैं |
- दिया जाने वाला dataset या तो series या DataFrame होता है |

Data Aggregation Functions

- हम निम्न excel sheet से एक dataframe बनाते हैं – जिसका process निम्न है|



F14

	A	B	C	D	E	F
1	Student_Name	Age	Gender	Test1	Test2	Test3
2	Aryan	16	F	7.6	8	7.6
3	NaN	NaN	NaN	NaN	NaN	NaN
4	Pratibha	16	M	8.6	NaN	NaN
5	Saumya	17	F	6.5	7.9	8.8
6	Ritika	15	F	6.8	7.7	7.9
7	Hari	16	M	9.2	9	NaN
8	Ram	14	F	6.8	8.7	8.8

```
>>> import pandas as pd
>>> df=pd.read_csv("C:\\Users\\KVBBKServer\\Desktop\\Student.csv")
>>> df
  Student_Name   Age Gender  Test1  Test2  Test3
0        Aryan  16.0      F    7.6    8.0    7.6
1          NaN   NaN    NaN    NaN    NaN    NaN
2     Pratibha  16.0      M    8.6    NaN    NaN
3       Saumya  17.0      F    6.5    7.9    8.8
4       Ritika  15.0      F    6.8    7.7    7.9
5         Hari  16.0      M    9.2    9.0    NaN
6          Ram  14.0      F    6.8    8.7    8.8
```

Data Aggregation Functions

DataFrame "df" में प्रत्येक column में प्रत्येक row के लिए उपस्थित non-NaN items को गिनकर प्रदर्शित करना |

```
>>> df.count()
Student_Name      6
Age               6
Gender            6
Test1             6
Test2             5
Test3             4
dtype: int64
```

उपरोक्त उदहारण में आपने देखा की प्रत्येक column में बिना NaN वाली values को ही गिना गया है |

DataFrame "df" में से किसी एक column के लिए उपस्थित non-NaN items को गिनकर प्रदर्शित करना हो तो निम्न दो तरीकों में से कोई भी तरीका अपना सकते हैं |

```
>>> df.Age.count()
6
>>> df['Age'].count()
6
>>> df.Test3.count()
4
>>> df['Test3'].count()
4
```

DataFrame के column सभी non-NaN values को जोड़ने के लिए निम्न तरीका अपनाया जा सकता है |

```
>>> df.sum()
Age          94.0
Test1        45.5
Test2        41.3
Test3        33.1
dtype: float64
```

DataFrame के row के सभी non-NaN values को जोड़ने के लिए निम्न तरीका अपनाया जा सकता है |

```
>>> df.sum(axis=1)
0    39.2
1     0.0
2    24.6
3    40.2
4    37.4
5    34.2
6    38.3
```

Data Aggregation Functions

```
>>> S=pd.Series([5,10,15,20,25])
>>> S.count()
5
>>> S.sum()
75
>>> S.mean()
15.0
>>> S.median()
15.0
```

```
>>> S.max()
25
>>> S.min()
5
>>> S.std()
7.905694150420948
>>> S.var()
62.5
```

```
>>> S=pd.Series([5,10,15,10,10,10,25])
>>> S.mode()
0    10
dtype: int64
```

Aggregation	Description
count()	कुल items की संख्या
sum()	दिए गए संख्याओं के सेट के items का sum को हल करता है |
mean()	दिए गए संख्याओं के सेट के items का mean या average को हल करता है |
median()	दिए गए संख्याओं के सेट के items के बीच की संख्या या median को हल करता है |
mode()	दिए गए संख्याओं के सेट के items में सबसे ज्यादा repeated value या mode को हल करता है |
max()	दिए गए संख्याओं के सेट में से सबसे बड़ी संख्या को खोजता है |
min()	दिए गए संख्याओं के सेट में से सबसे छोटी संख्या को खोजता है |
std()	दिए गए संख्याओं के सेट के items का standared deviation calculate करता है |
var()	दिए गए संख्याओं के सेट के items का variance calculate करता है |

Data Aggregation Functions

```
>>> df.min()
Age       14.0
Test1      6.5
Test2      7.7
Test3      7.6
dtype: float64
>>> df.Age.min()
14.0
>>> df.Age.std()
1.0327955589886446
```

```
>>> df.var()
Age       1.066667
Test1     1.209667
Test2     0.313000
Test3     0.382500
dtype: float64
>>> df.Age.var()
1.0666666666666669
```

Assignment:

State wise sales values of an item is given below-

State	Sales
Goa	650000
Delhi	692400
Odisha	750000
Haryana	867000
Bihar	920000
Kerala	939000
Tamil Nadu	1015000
West Bengal	1553000
Maharashtra	2176000

Write Commands for the following (Dataframe name is dfA)

(a) Count the number of observation in dfA.
(b) Count the number of observation in column state of dfA.
(c) Sum of the non-null value across the row axis for dfA.
(d) Calculate the mean of Sales columns in dfA.
(e) Add a commission column into dfA. (Commission = sales*0.04)
(f) Calculate the mean of all numeric columns in dfA.
(g) Calculate the mean of sales column.
(h) Find maximum sales and commission values.
(i) Find minimum sales and commission values.
(j) Find the standard deviation of commissions.

Quantiles with Pandas

- Statistics में 3 शब्द बहुत प्रयोग में लाये जाते हैं - Quartile, Quantile और percentile. इनमे निम्न अंतर होता है |
  - 0 quartile = 0 quantile = 0 percentile
  - 1 quartile = 0.25 quantile = 25 percentile
  - 2 quartile = 0.50 quantile = 50 percentile
  - 3 quartile = 0.75 quantile = 75 percentile
  - 4 quartile = 1 quantile = 100 percentile
- Statistics में ***Quartile*** आपके data को 4 हिस्सों (quarter) में बाँट देता है |
- ***Percentile*** एक संख्या है, जहां एक निश्चित percentage स्कोर, उस संख्या से नीचे आता है | इनका प्रयोग अधिकतर परीक्षा इत्यादि में स्कोर को रिपोर्ट करने में किया जाता है |
  - Percentile = ((N-your Rank)/N)*100
  - Your rank = (percentile/100) * number of items
- Quantiles एक distribution में वह point हैं जो उस distribution में values के rank order से संबंधित हैं |

Pandas में Quantile () का प्रयोग

- Pandas का quantile() function किसी request किये गए axis के लिए float अथवा series values को return करता है | request किया गया axis एक numpy percentile होता है | यह एक probabilities की list होती है जिस पर quantile compute किया जाना है |
  - माना यदि percentile = 25 है तो यह पहला quartile अथवा lower quartile होगा |
  - यदि percentile = 50 है तो यह दूसरा quartile अथवा median होगा |
  - यदि percentile = 75 है तो यह तीसरा quartile अथवा upper quartile होगा |
- इसका syntax निम्न है –

  DataFrame.quantile(q=0.5, axis=0, numeric_only=True, interpolation = 'linear')
- जहाँ -
  - q एक float है या array के जैसा, इसका default 0.5 (50% quantile) और 0<=q<=1
  - axis : 0 अथवा 'index' और 1 अथवा 'columns' | इसका default 0 होता है |
  - numeric_only : boolean, default True होता है |
  - interpolation: {'linear', 'lower', 'higher', 'midpoint', 'nearest'}. यह वैकल्पिक (optional) होता है |

Pandas में Quantile () का प्रयोग

Find the Quantile of an odd series s given bellow:

```
>>> s=pd.Series([15,16,18,27,29,32,36])
>>> s.quantile(.25)
17.0
>>> s.quantile([0.25,0.5,0.75])
0.25    17.0
0.50    27.0
0.75    30.5
dtype: float64
```

आप निम्न कमांड भी try कर सकते हैं |

```
>>> import numpy as np
>>> S=pd.Series([15,16,18,27,29,32,36]
>>> np.percentile(S,25)
17.0
```

Find the Quantile of an even series s given bellow:

```
>>> P=pd.Series([15,16,18,29,32,36])
>>> P.quantile([.25,.5,.75])
0.25    16.50
0.50    23.50
0.75    31.25
dtype: float64
```

आप निम्न कमांड भी try कर सकते हैं |

```
>>> import numpy as np
>>> S=pd.Series([15,16,18,29,32,36])
>>> print(np.percentile(P,[25,50,75]))
[16.5  23.5  31.25]
```

Pandas में Quantile () का प्रयोग

Find the quantile of DataFrame "df"

```
>>> import pandas as pd
>>> df=pd.read_csv("C:\\Users\\KVBBKServer\\Desktop\\Student.csv")
>>> df
  Student_Name   Age Gender  Test1  Test2  Test3
0        Aryan  16.0      F    7.6    8.0    7.6
1          NaN   NaN    NaN    NaN    NaN    NaN
2      Pratibha  16.0      M    8.6    NaN    NaN
3       Saumya  17.0      F    6.5    7.9    8.8
4       Ritika  15.0      F    6.8    7.7    7.9
5         Hari  16.0      M    9.2    9.0    NaN
6          Ram  14.0      F    6.8    8.7    8.8
>>> df.quantile(.25)
Age      15.250
Test1     6.800
Test2     7.900
Test3     7.825
Name: 0.25, dtype: float64
>>> df.quantile([.25,.5,.75])
        Age  Test1  Test2  Test3
0.25  15.25   6.80    7.9  7.825
0.50  16.00   7.20    8.0  8.350
0.75  16.00   8.35    8.7  8.800
```

```
>>> quants=[0.05,0.25,0.5,0.75,0.95]
>>> q=df.quantile(quants)
>>> print(q)
        Age  Test1  Test2  Test3
0.05  14.25  6.575   7.74  7.645
0.25  15.25  6.800   7.90  7.825
0.50  16.00  7.200   8.00  8.350
0.75  16.00  8.350   8.70  8.800
0.95  16.75  9.050   8.94  8.800
```

Descriptive Statistics

- Python – pandas में descriptive या summary statistics के लिए ***describe ( )*** function का प्रयोग किया जाता है |
- Describe ( ) के द्वारा mean , std और interquartile (IQR) values को हासिल किया जा सकता है |
- Describe ( ) एक numeric columns पर काम करने के लिए बहुत ही आसन function होता है |
- किसी column के basic statistics को देखे के लिए आप describe ( ) function का प्रयोग कर सकते हैं –जैसे - mean, min, max इत्यादि |
- सामान्यतया describe ( ) character columns को छोड़ देता है और सिर्फ numeric columns पर कार्य करता है |
- किसी dataframe को describe करने पर सिर्फ numeric fields ही return होती है | और यह 8 प्रकार की statistical properties को दर्शाता है -

count( )	mean()
std()	min()
25^{th} percentile	50^{th} percentile
75^{th} percentile	max()

Descriptive Statistics

- Describe ( ) function का syntax निम्नवत है -

DataFrame.describe(percentile = None, include = None, exclude=None)

जहाँ : -

- percentile : default है [0.25, 0.5, 0.75]
- Include: default None है, अन्य में 'All' होसकता है |
- Exclude : भी default None है , यह तब प्रयोग में लाया जाता है जब आप किसी भी column को इन्क्लुड़े न करना चाहें |

```
>>> S=pd.Series([5,10,15,20,25])
>>> S.describe()
count      5.000000
mean      15.000000
std        7.905694
min        5.000000
25%       10.000000
50%       15.000000
75%       20.000000
max       25.000000
dtype: float64
```

STRING टाइप के series के लिए |

```
>>> str=pd.Series(['a','a','b','b','c','d'])
>>> str.describe()
count     6
unique    4
top       a
freq      2
dtype: object
```

Descriptive Statistics

```
>>> import pandas as pd
>>> df=pd.read_csv("C:\\Users\\KVBBKServer\\Desktop\\Student.csv")
>>> df
  Student_Name   Age Gender  Test1  Test2  Test3
0        Aryan  16.0      F    7.6    8.0    7.6
1          NaN   NaN    NaN    NaN    NaN    NaN
2     Pratibha  16.0      M    8.6    NaN    NaN
3       Saumya  17.0      F    6.5    7.9    8.8
4       Ritika  15.0      F    6.8    7.7    7.9
5         Hari  16.0      M    9.2    9.0    NaN
6          Ram  14.0      F    6.8    8.7    8.8
>>> df.describe()
             Age      Test1      Test2      Test3
count   6.000000   6.000000   5.000000   4.000000
mean   15.666667   7.583333   8.260000   8.275000
std     1.032796   1.099848   0.559464   0.618466
min    14.000000   6.500000   7.700000   7.600000
25%    15.250000   6.800000   7.900000   7.825000
50%    16.000000   7.200000   8.000000   8.350000
75%    16.000000   8.350000   8.700000   8.800000
max    17.000000   9.200000   9.000000   8.800000
```

DataFrame के लिए describe( ) function.

```
>>> df['Age'].describe()
count      6.000000
mean      15.666667
std        1.032796
min       14.000000
25%       15.250000
50%       16.000000
75%       16.000000
max       17.000000
Name: Age, dtype: float64
```

Descriptive Statistics (Assignment)

Example A dataframe dfS is given with following data for 10 students:

	Name	Total Marks
0	Amit Kumar	450
1	Vinamr Katyal	476
2	Aasha Goel	426
3	Naina Rawat	476
4	Dawn Sebastian	458
5	Riya Mary	446
6	Aashna Sharma	461
7	Piush Cocher	464
8	Om Berma	476
9	Manali Sovani	452

Write commands for the following:

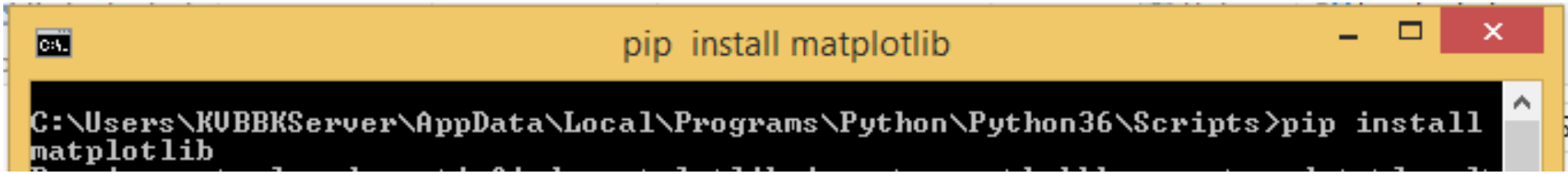
(*a*) Find the default quantile of the DataFrame.
(*b*) Find the [.25, .5, .75] quantiles of the DataFrame.
(*c*) Write the summary of statistics pertaining to the dataframe column.
(*d*) Get the full summary of statistics to the dataframe.
(*e*) Find the 50th percentile of the dataframe.

Histogram

- Histogram किसी data के distribution को analyze करने के लिए एक powerful टूल है |
- एक histogram plot को सामान्यतया किसी संख्या की frequency को दर्शाने के लिए प्रयोग किया जाता है |
- इसके द्वारा user को data का विभिन्न catagory में distribution आसानी से समझ में आजाता है | तथा साथ ही data की median और range भी समझ आजाती है |
- Histogram बनाने के लिए, पहले, हम values की पूरी range को intervals की एक series में विभाजित करते हैं। और दूसरा, हम गिनते हैं कि प्रत्येक interval में कितने values आते हैं|
- Matplotlib फिर bins में उन categories या intervals को call करता है | bins, variables के continuous और non-overlapping intervals होते हैं। वे एक दूसरे के ठीक लगे हुए (adjacent) और बराबर size के होने चाहिए|
- Histogram में -
  - **X – axis**: observation के intervals को दर्शाता है |
  - **Y- axis:** यह frequency के घनत्व (density) को दर्शाता है |

Matplotlib

- Matplotlib पाइथन की एक अग्रणी visualization library है जो कि एक powerful और two dimentional plotting library है |
- यह numpy arrays के आधार पर बनी हुई एक multi-platform data visualization library है |
- यह सभी प्रकार के graph, plots, charts, histograms इत्यादि बनाने में सक्षम है |
- इसके लिए आपको अपने system में pip कमांड के द्वारा matplotlib library को install करना होता है |




```
>>> import matplotlib
>>> print(matplotlib.__version__)
3.0.1
```

इस कमांड से हम पता कर सकते हैं की matplotlib का कौन सा version installed है |

Histogram बनाना

- Histogram बनाने के लिए syntax निम्न है -

```
DataFrame.hist(column=None, by=None, grid=True, xlabelsize=None, xrot=None, ylabelsize=None, yrot=None, ax=None, sharex=False, sharey=False, figsize=None, layout=None, bins=10, **kwds)
```

- column: is the dataframe column name to create a histogram.
- by: If passed, then used to form histograms for separate groups. The by option will take an object by which the data can be grouped.
- grid: takes a Boolean value, i.e., to enable (if **True**) or disable (if **False**) the grid.
- xlabelsize, ylabelsize: these options change the size of x and y label text size.
- sharex, sharey: to set both of the axes to the same range and scale.
- bin: isumber of histogram bins to be used. The default value is 10.
- fill: to fillis the dataframe column name to create a histogram.

- Example के लिए हम निम्न DataFrame लेते हैं -

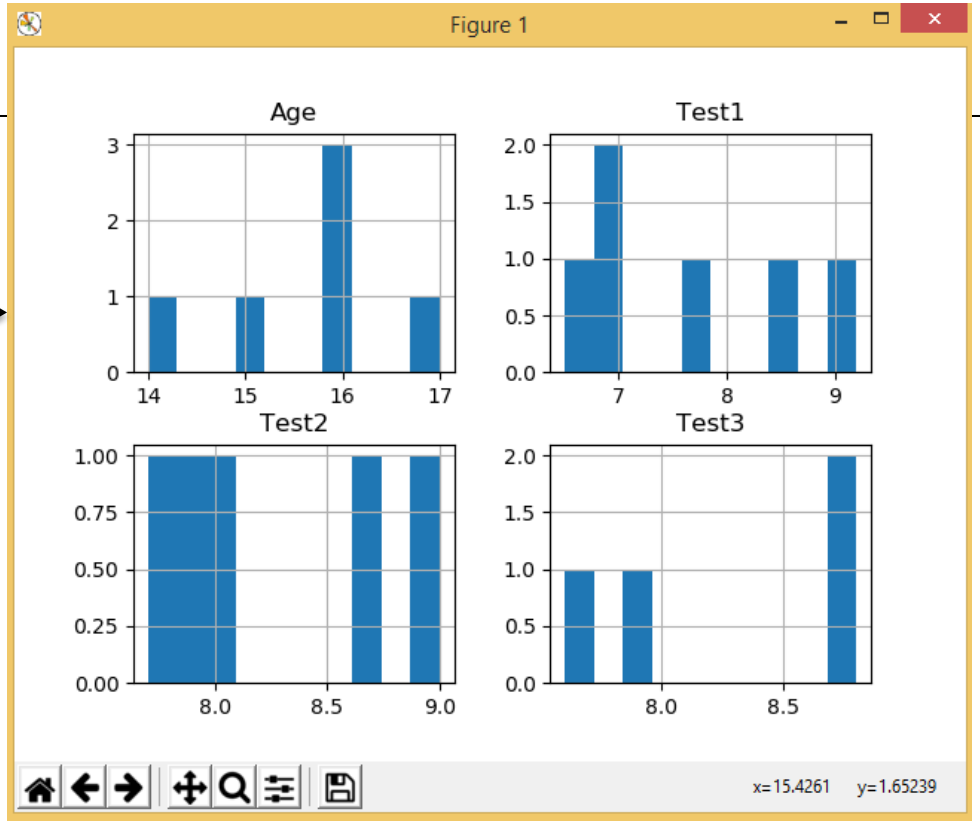
```
>>> df
  Student_Name   Age Gender  Test1  Test2  Test3
0        Aryan  16.0      F    7.6    8.0    7.6
1          NaN   NaN    NaN    NaN    NaN    NaN
2     Pratibha  16.0      M    8.6    NaN    NaN
3       Saumya  17.0      F    6.5    7.9    8.8
4       Ritika  15.0      F    6.8    7.7    7.9
5         Hari  16.0      M    9.2    9.0    NaN
6          Ram  14.0      F    6.8    8.7    8.8
```

Histogram बनाना

```
>>> import matplotlib.pyplot as plt
>>> df.hist()
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000009B7FCB0080>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000009B7FD1A7F0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000009B7FE83D68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000009B7FEB3320>]],
      dtype=object)
>>> plt.show()
```

Histogram बनाने के लिए hist( ) function का प्रयोग करते हैं |

उपरोक्त कमांड चलाने पर बगल में दर्शाई गयी छवि जो histogram बन कर आरहा है वही output आपको प्रदर्शित होता है |

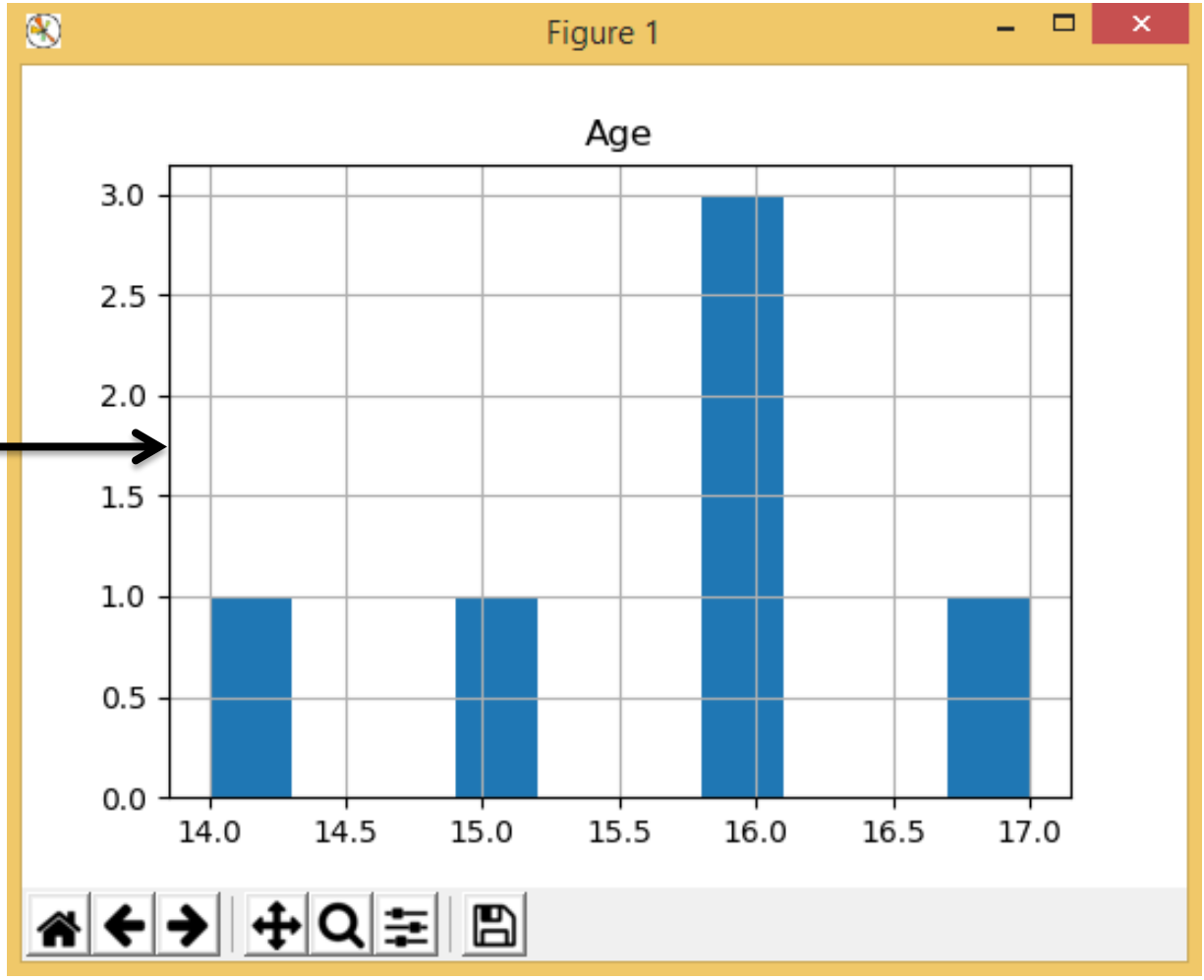


Pandas dataframe से single Histogram बनाना

hist ( ) function में column pass कर देते हैं |

```
>>> df.hist(column='Age')
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000009B7FBB2630>]],
      dtype=object)
>>> plt.show()
```

उपरोक्त कमांड चलाने पर बगल में दर्शाई गयी छवि जो histogram बन कर आरहा है वही output आपको प्रदर्शित होता है |

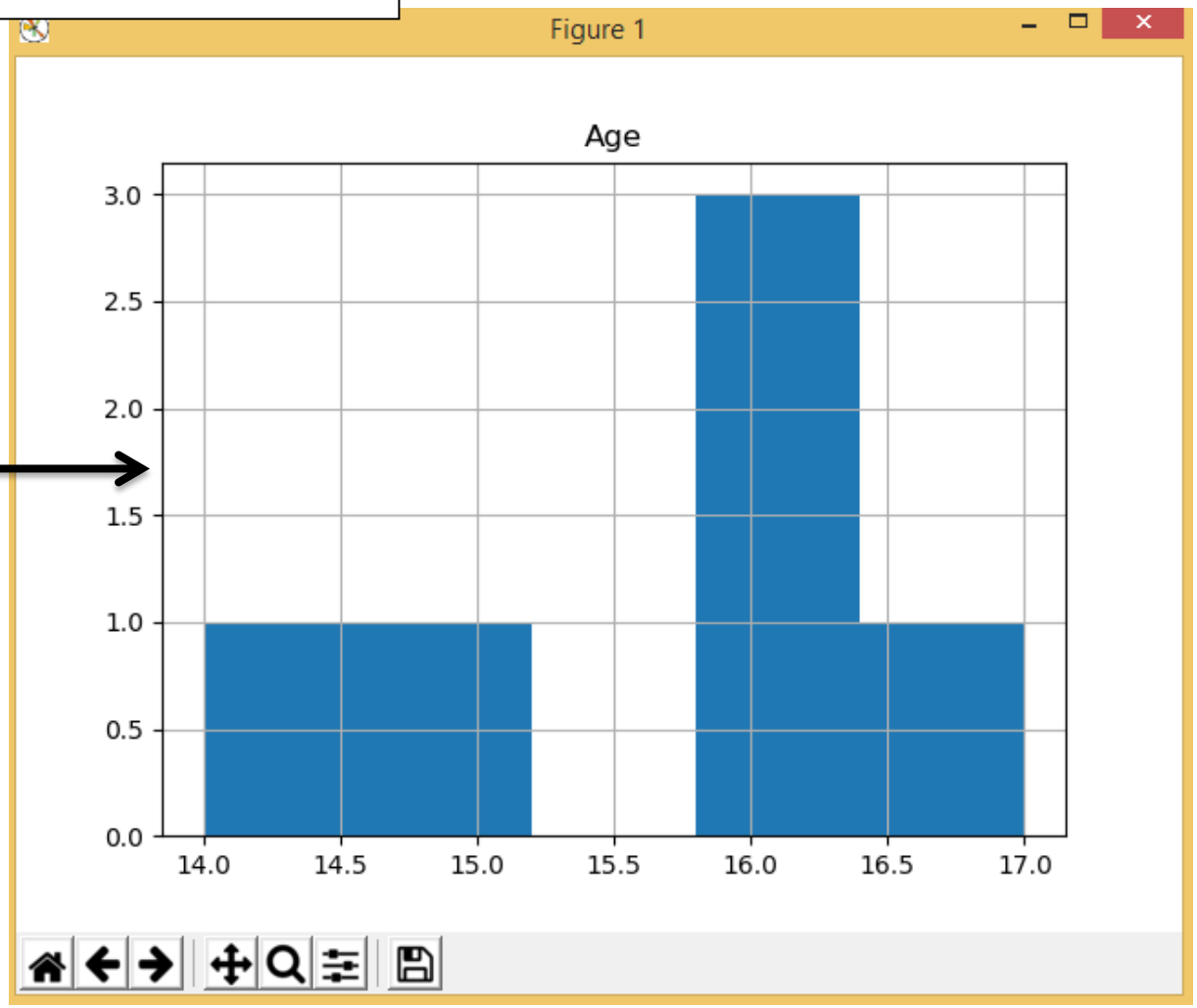


Histogram के bins के आकार को बदलना

```
>>> df.hist(column='Age', bins=5)
array([[<matplotlib.axes._subplot
       dtype=object)
>>> plt.show()
```

hist ( ) function में column के साथ bins को भी pass कर देते हैं |

उपरोक्त कमांड चलाने पर बगल में दर्शाई गयी छवि जो histogram बन कर आरहा है वही output आपको प्रदर्शित होता है |

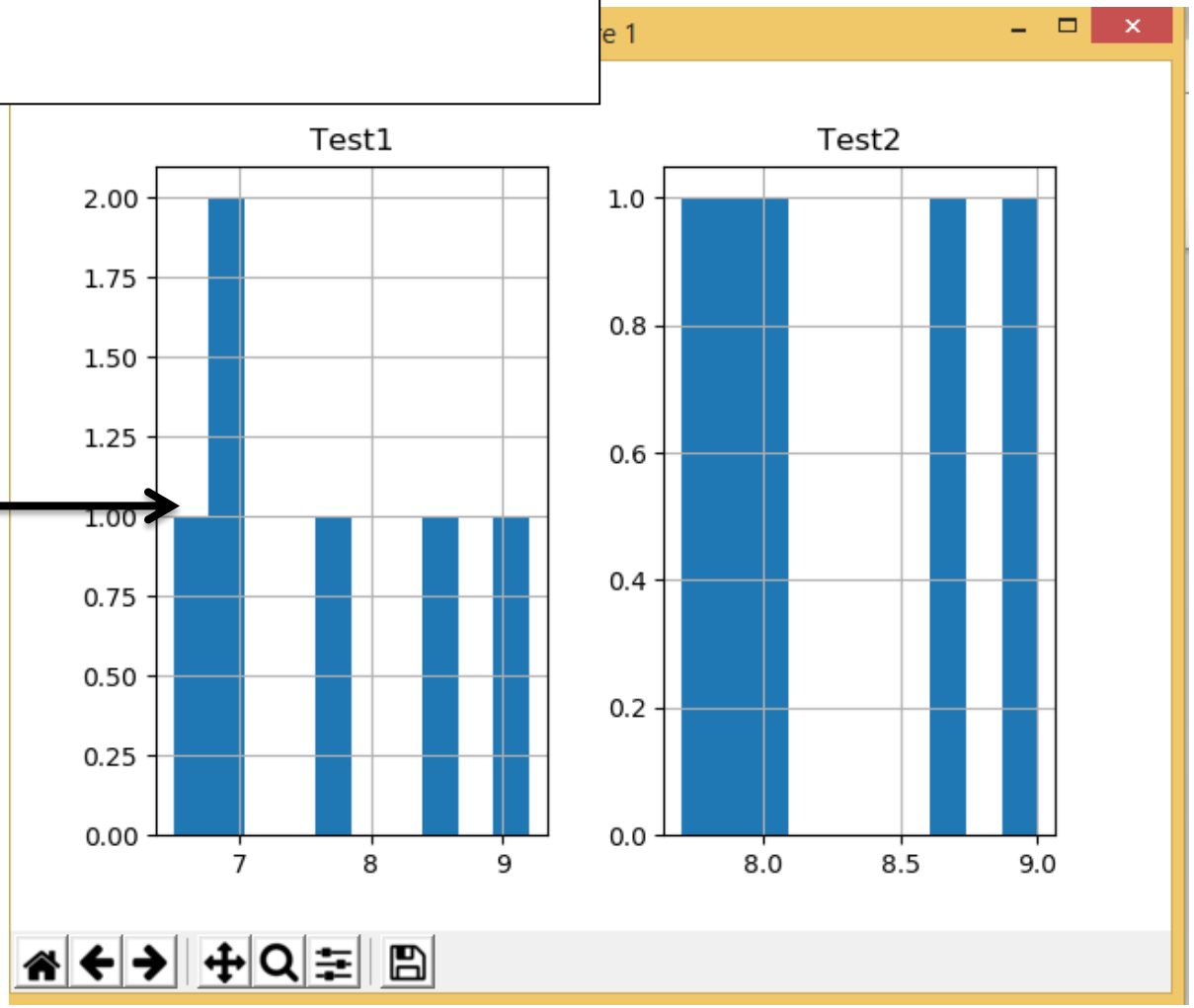


Multiple pandas Histogram

```
>>> df.hist(column=["Test1","Test2"])
array([[<matplotlib.axes._subplots.Axe
        <matplotlib.axes._subplots.Axe
       dtype=object)
>>> plt.show()
```

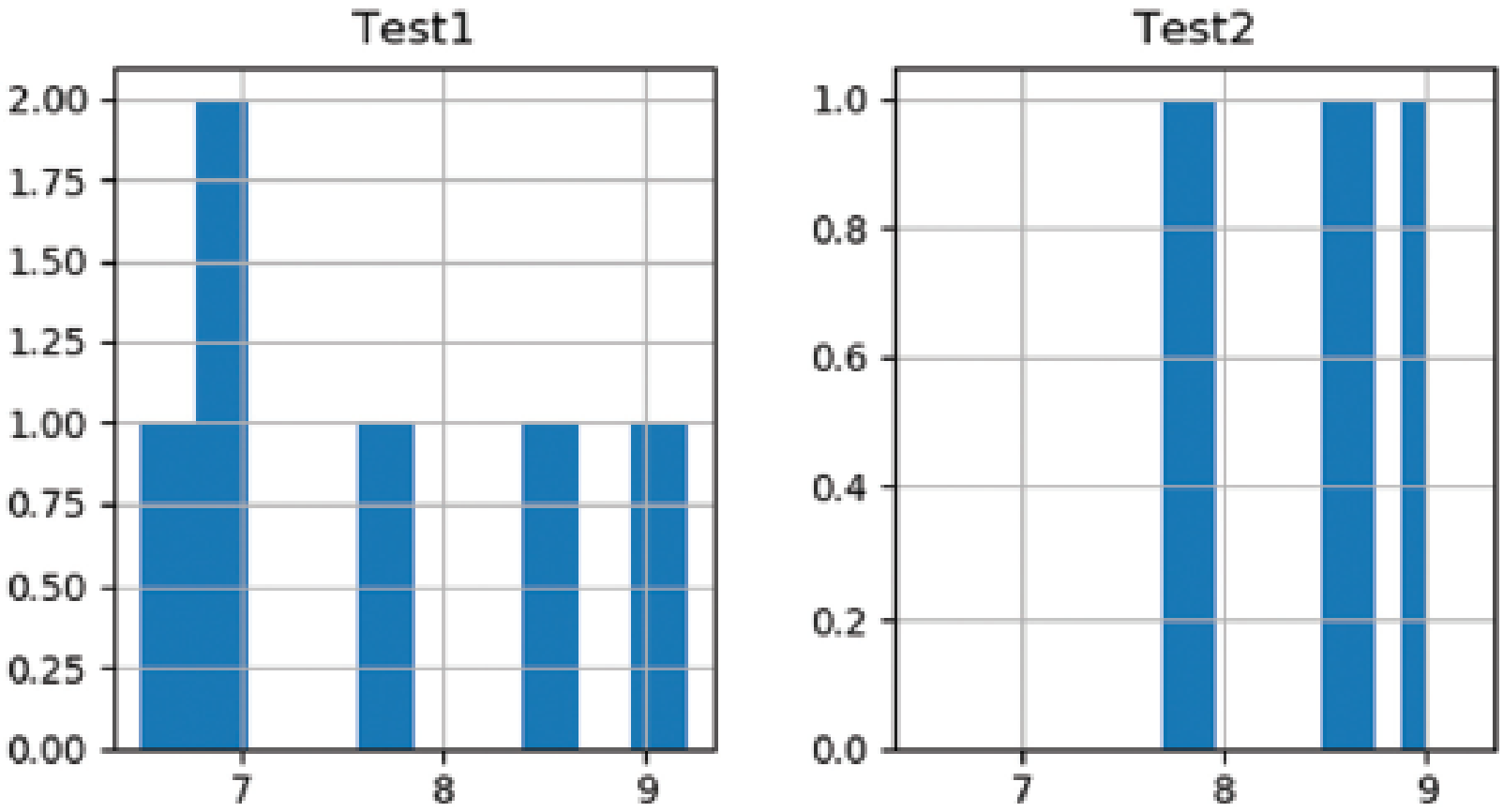
hist ( ) function में column को भी pass कर देते हैं |

उपरोक्त कमांड चलाने पर बगल में दर्शाई गयी छवि जो histogram बन कर आरहा है वही output आपको प्रदर्शित होता है |



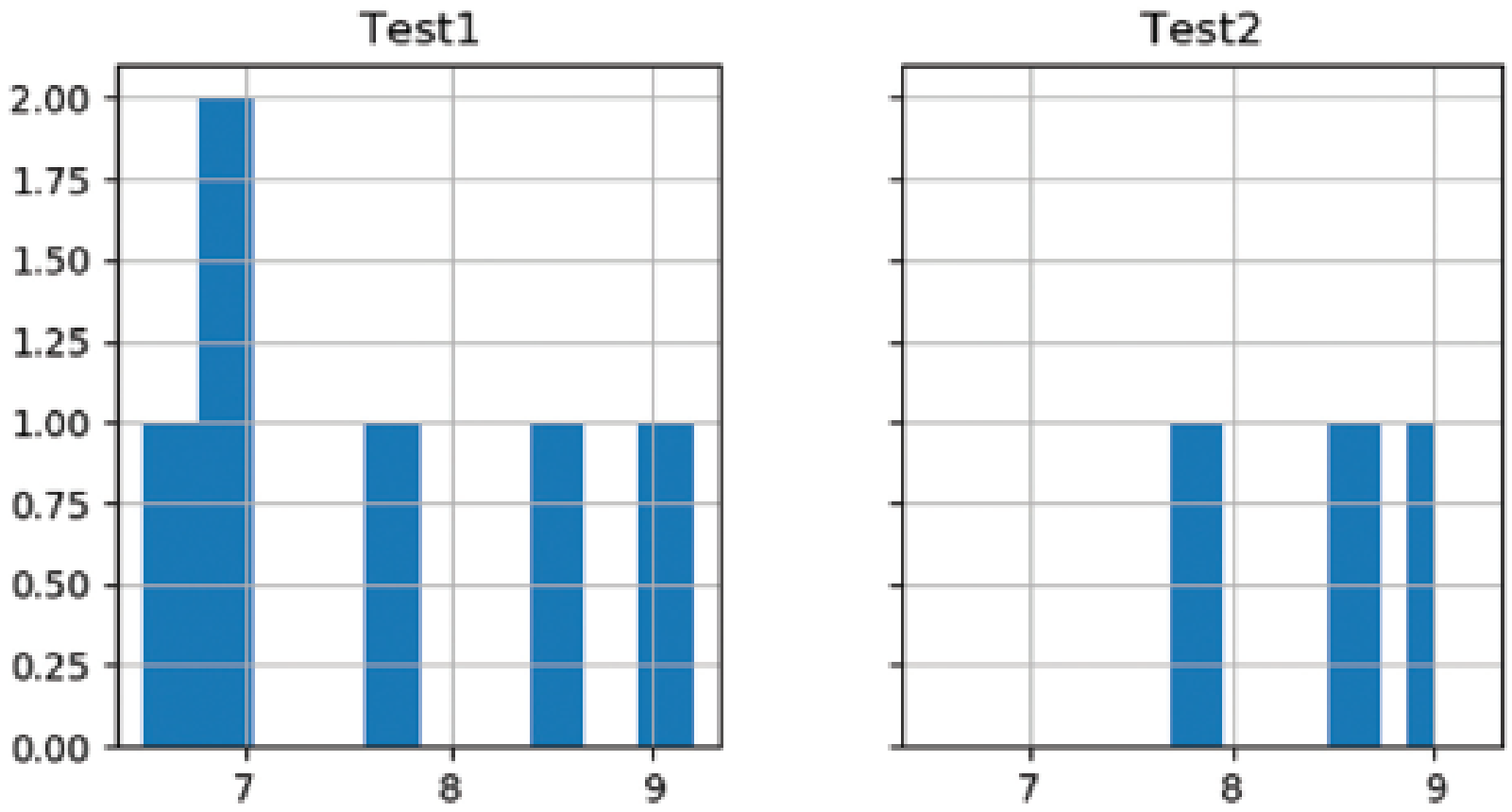
Histogram Axes को बदलना

```
>>> df.hist(column=["Test1", "Test2"], sharex=True)  # Share only x axis
>>> plt.show()
```



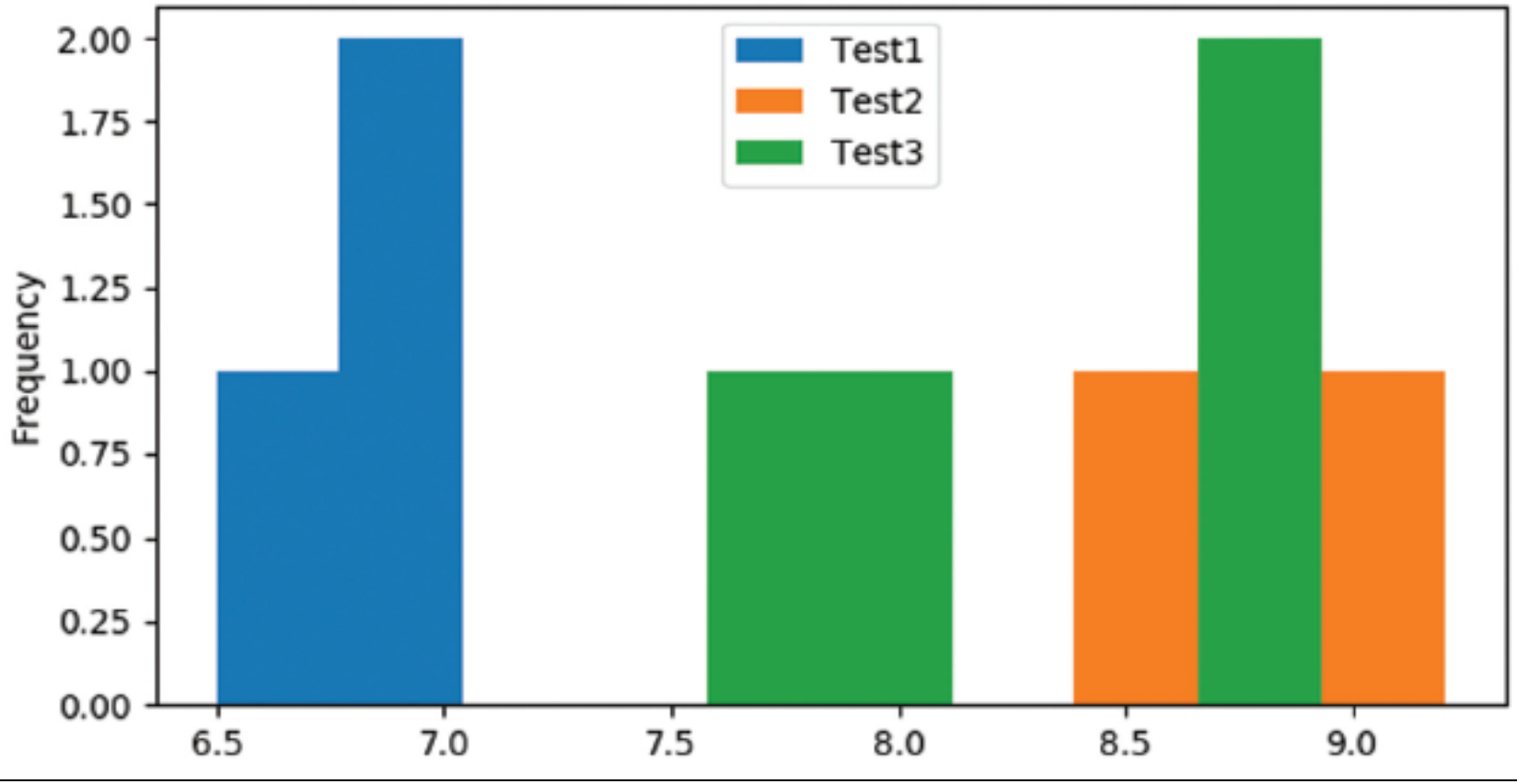
Histogram Axes को बदलना

```
>>> df.hist(column=["Test1", "Test2"], sharex=True, sharey=True)  # Share x and y axis
>>> plt.show()
```



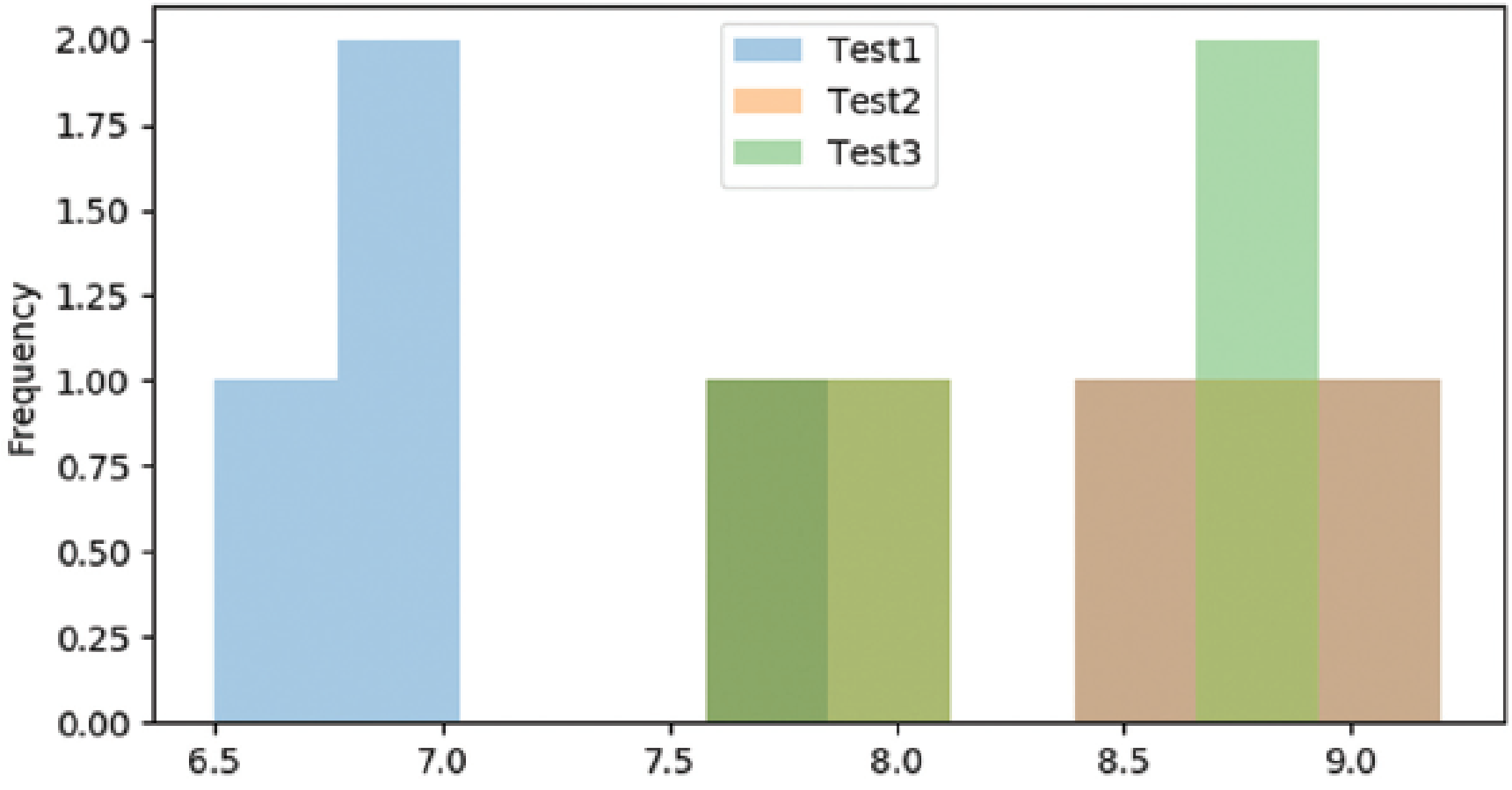
Multiple features in one plot

```
>>> df[["Test1", "Test2", "Test3"]].plot.hist()  # Note slicing is performed on df itself
>>> plt.show()
```



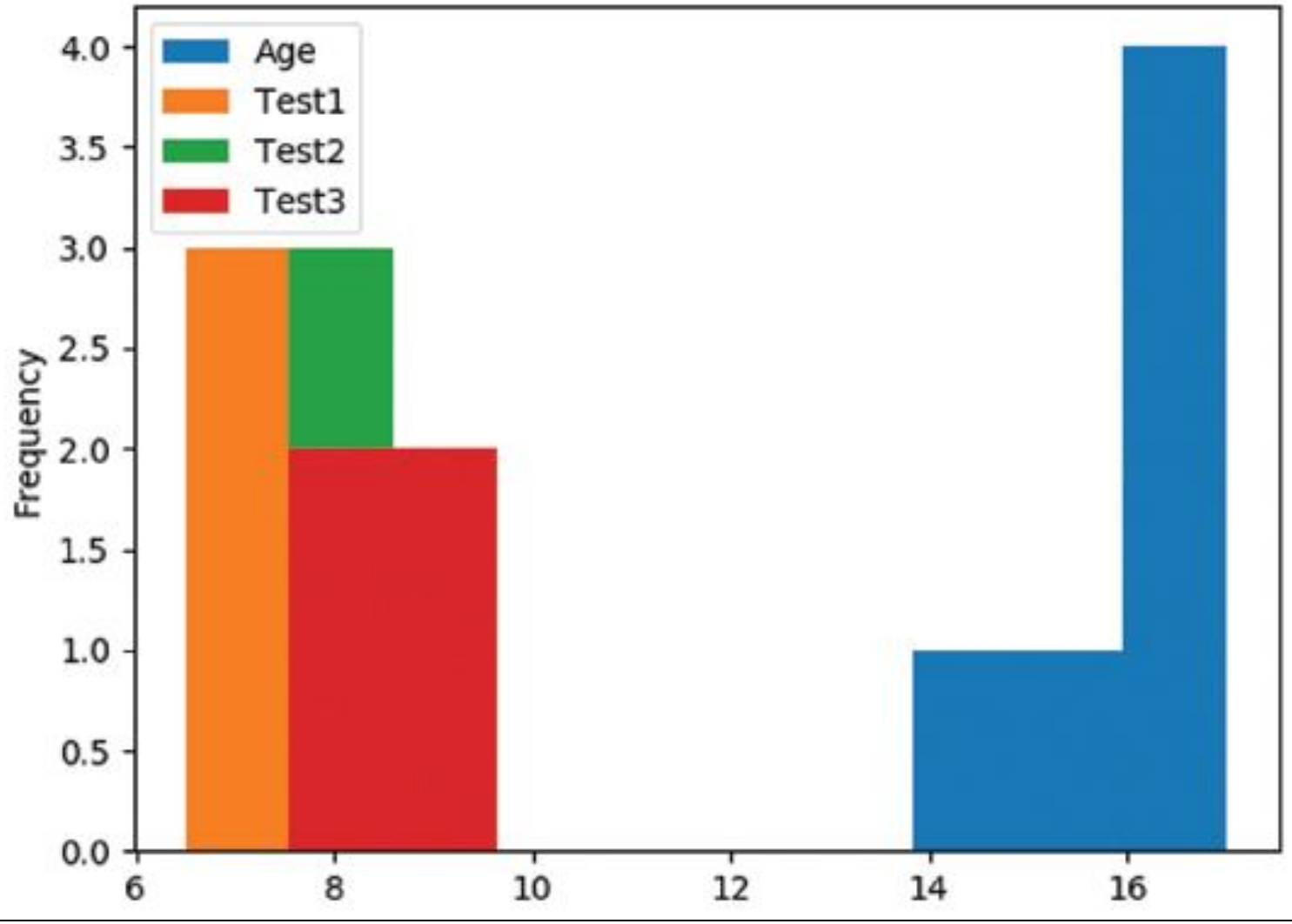
Multiple features in one plot

```
>>> df[["Test1", "Test2", "Test3"]].plot.hist(alpha=0.4)  # Plot at 40% opacity
>>> plt.show()
```



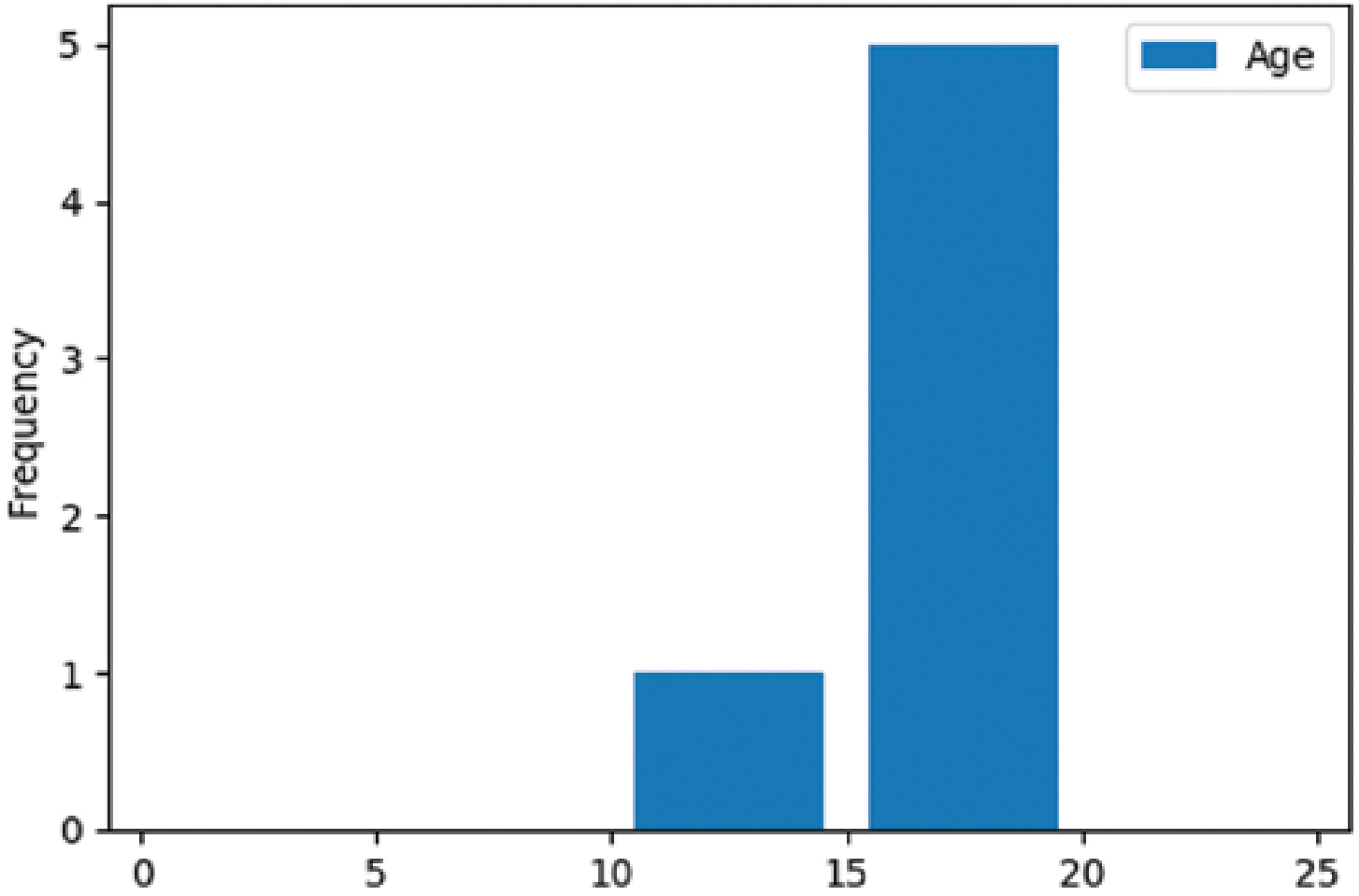
Plotting DataFrame Columns using DataFrame plot () Method

```
>>> df.plot(kind='hist')
>>> plt.show()
```



Histogram using single/multiple column

```
>>> df[['Age']].plot(kind='hist',bins=[0,5,10,15,20, 25],rwidth=0.8)
>>> plt.show()
```



Assignment

Example **Using previous dataframe dfS, write the command to create following histograms:**

(*a*) Crete a histogram plot to show Total Marks

(*b*) Create a histogram plot to show Total Marks with bins 100.

(*c*) Create a histogram plot to show Total Marks with bins [400,420,440,460,480,500].

Solution For data: dfA = pd.read_csv('E:/IPSource_XII/IPXIIChap03/Std7.csv')

Assume that the following modules are imported.

```
import pandas as pd
import matplotlib.pyplot as plt
```

(*a*)
```
dfA = dfA.sort_values(by='Total Marks')
    plt.show()
```

(*b*)
```
dfA.hist(column="Total Marks", bins=200)
    plt.show()
```

(*c*)
```
dfA.hist(column="Total Marks", bins=[400,420,440,460,480,500])
    plt.show()
    Or
    dfA[['Total Marks']].plot(kind='hist',bins=[400,420,440,460,480,500])
    plt.show()
```

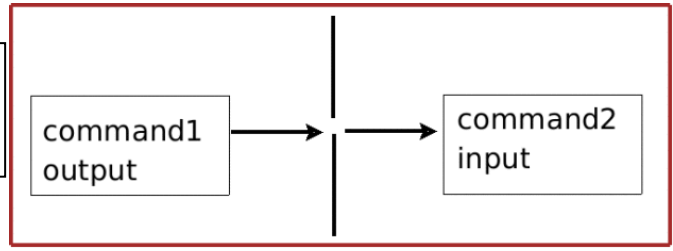
Function Application

- Pandas library कई सुविधाजनक function प्रदान करती है जो data science से सम्बंधित कई कार्यों को करने में सहायता करते हैं | functions के ऐसे कार्यों को function application कहते हैं |
- एक तरह से यह भी कह सकते हिं की functions को dataframe के साथ कई तरीकों से प्रयोग किये जाने को function application कह सकते हैं –जैसे -
  - सम्पूर्ण dataframe पर
  - Row wise अथवा column-wise
  - किसी individual value पर अर्थात element – wise
- इस क्रम में तीन प्रकार के function होते हैं –
  - pipe ( ) → dataframe wise function application
  - apply ( ) → row-wise/column-wise function application
  - applymap ( ) → element-wise function application

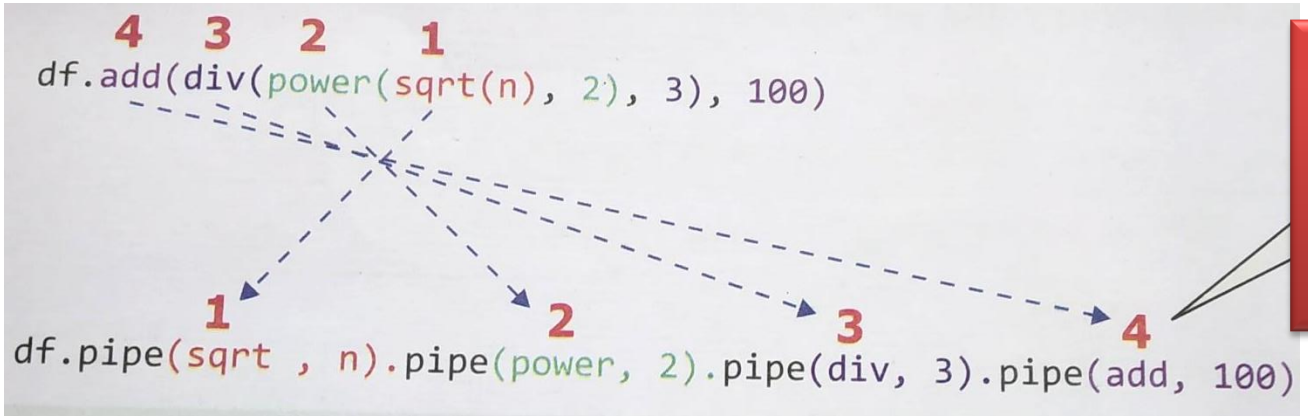
pipe ( ) function

- कई बार हमें ऐसे काम करना पड़ता है कि एक function का output दूसरे function के लिए इनपुट का काम करेगा और तब process आगे बढ़ेगी | इसे ही pipe करना कहते हैं | निम्न उदहारण को देखें -

```
>>> math.pow(math.sqrt(25),3)
125.0
```



- इस उदहारण में पहले sqrt ( ) function का output आयेगा जो की pow ( ) function के लिए इनपुट का काम करेगा | अर्थात ये function, pipe में काम कर रहे हैं |
- pipe ( ) function सारे functions के आर्डर को बदल कर उस क्रम में लाता है जिस क्रम में वह execute हो रहे हैं |



pipe function के अन्दर pass किये गए Function के साथ parenthesis नहीं देते हैं |

pipe ( ) function

```
>>> df5.pipe(np.multiply,2).pipe(np.add,100)
      A     B     C
0   102   104   106
1   108   110   112
2   114   116   118
3   120   140   160
4   180   200   220
5   240   260   280
6   300   500   700
7   900  1100  1300
8  1500  1700  1900
```

```
>>> df5
   A   B   C
0  1   2   3
1  4   5   6
2  7   8   9
3  10  20  30
4  40  50  60
5  70  80  90
6  100 200 300
7  400 500 600
8  700 800 900
```

apply ( ) और applymap ( )function

```
>>> df5
     A    B    C
0    1    2    3
1    4    5    6
2    7    8    9
3   10   20   30
4   40   50   60
5   70   80   90
6  100  200  300
7  400  500  600
8  700  800  900
```

```
>>> df5.apply(np.mean)
A    148.0
B    185.0
C    222.0
dtype: float64
>>> df5.applymap(np.mean)
      A      B      C
0   1.0    2.0    3.0
1   4.0    5.0    6.0
2   7.0    8.0    9.0
3  10.0   20.0   30.0
4  40.0   50.0   60.0
5  70.0   80.0   90.0
6 100.0  200.0  300.0
7 400.0  500.0  600.0
8 700.0  800.0  900.0
```

यहाँ single value प्रति column return हो रही है | क्योंकि apply( ) एक series का function है

जबकि यहाँ single value प्रति element return हो रही है | क्यों कि applymap ( ) एक element का function है |

Missing Data को handle करना

- वह values जो किसी computation में हिस्सा नहीं ले पाती हैं या यों कहे कि missing values वह values होती हैं जिनका कोई computational significance नहीं होता है |
- Missing data को handle करने के लिए निम्न तरीके अपनाये जाते हैं -
  - Dropping missing data
  - Filling missing data (Imputation)

```
>>> df10
      A     B   C
0  1001  2002 NaN
1  3004  4005 NaN
2  5007  6008 NaN
```

```
>>> df11.fillna(0)
      A     B    C
0  1001  2002  0.0
1  3004  4005  0.0
2  5007  6008  0.0
```

```
>>> df11=df10.dropna()
>>> df11
Empty DataFrame
Columns: [A, B, C]
Index: []
>>> df11=df10.dropna(how='all')
>>> df11
      A     B   C
0  1001  2002 NaN
1  3004  4005 NaN
2  5007  6008 NaN
```

Comparison of Pandas Objects

```
>>> df1
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
>>> df2
    A   B   C
0  10  20  30
1  40  50  60
2  70  80  90
```

```
>>> df3
     A    B    C
0  100  200  300
1  400  500  600
```

```
>>> df12
       A      B      C
0  101.0  202.0  303.0
1  404.0  505.0  606.0
2    NaN    NaN    NaN
```

```
>>> df1+df2==df1.add(df2)
      A     B     C
0  True  True  True
1  True  True  True
2  True  True  True
>>> df1+df3==df1.add(df3)
       A      B      C
0   True   True   True
1   True   True   True
2  False  False  False
```

```
>>> df1>5
       A      B      C
0  False  False  False
1  False  False   True
2   True   True   True
```

```
>>> (df1+df3).equals(df1.add(df3))
True
```

equals () दोनों ऑब्जेक्ट की बराबरी check करता है |

Pivoting DataFrame

- ***Data analysis*** के लिए Pandas एक प्रचलित library है |
- किसी भी data analyst के लिए प्रमुख कार्यों में से एक डेटा टेबल को ***pivot*** करना है। अर्थात table data को एक धुरी प्रदान करना जिसके आधार पर database काम करे |
- Pandas का प्रयोग करके MS-Excel के प्रकार के pivot tables बनाये जा सकते हैं |
- ये बड़े data को तुरंत ***summarize*** करके meaningful reports तैयार करने में आपके समय की बहुत बचत करते हैं |
- Pivot table हमें एक बड़े, विस्तृत डेटा सेट से महत्वपूर्ण record निकालने की अनुमति देती है।
- Pivot tables स्वतः sort, count, total इत्यादि कर लेती हैं |
- एक सामान्य बात कहें तो pivot करने का अर्थ है किसी index या column से unique value को प्रयोग करना और DataFrame बनाना |
- Pandas के द्वारा pivot table बनाने के लिए हम ***pivot( )*** या ***pivot_table()*** method का प्रयोग करते हैं |

pivot( ) method का प्रयोग करके Pivoting करना

- pivot() method, column values के आधार पर data को reshape करके नया DataFrame बनाता है |
- यह method 3 arguments लेता है - ***index***, ***columns*** और ***values*** | इनमे से 2 को लेना अनिवार्य है |
- इन arguments की value के रूप में आपको original table के column नाम देने होते हैं |
- तब pivot ( ) एक नयी table बनाता है जिनके row और column के index वही होते हैं जिनको आपने argument के रूप में pass किया है |
- नयी table की cell values उसी column से आयेंगी जिसको आपने parameter के रूप में दिया था | इसका syntax निम्न है –

```
pandas.pivot(index, columns, values)
```

- जहाँ index के द्वारा नए DataFrame का index बनता है जो table से लिया गया column name है |
- जहाँ columns के द्वारा नए DataFrame के columns बनते है जो table से लिए गए column name है |
- जहाँ values के द्वारा नए DataFrame की columns बनती है जो table से लिए गए column name की values हैं |

pivot( ) method का प्रयोग करके Pivoting करना

syntax → pandas.pivot(index, columns, values)

- उदाहरण→ DataFrame बनाना

```
>>> df
      Name Subject  Score Grade
0  Pratibha      CS     99    A1
1    Ritika     Phy     87    A2
2    Saumya    Chem     88    A2
3     Aryan   Maths     67     B
```

```
import pandas as pd
ClassXII={'Name':['Pratibha','Ritika','Saumya','Aryan'],\
          'Subject':['CS','Phy','Chem','Maths'],\
          'Score':[99,87,88,67],\
          'Grade':['A1','A2','A2','B']}
df=pd.DataFrame(ClassXII,columns=['Name','Subject','Score','Grade'])
```

- pivot table बनाना →

```
>>> pv=df.pivot(index='Name',columns='Subject',values='Score')
>>> pv
Subject      CS   Chem  Maths   Phy
Name
Aryan       NaN    NaN   67.0   NaN
Pratibha   99.0    NaN    NaN   NaN
Ritika      NaN    NaN    NaN  87.0
Saumya      NaN   88.0    NaN   NaN
```

हम इस pivot table में देख सकते हैं कि एक नयी table बनी है और Score column की values अलग अलग column में आयीं हैं | जबकि इसकी Name और Subject column, original table से match हो रही हैं| जहाँ values match नहीं हो रही हैं उस जगह NaN (None) स्वतः ही चला गया है |

pivot( ) method का प्रयोग .fillna( ) के साथ

syntax → pandas.pivot(index, columns, values).fillna()

- उदाहरण→ DataFrame बनाना

```
>>> df
      Name Subject  Score Grade
0  Pratibha      CS     99    A1
1    Ritika     Phy     87    A2
2    Saumya    Chem     88    A2
3     Aryan   Maths     67     B
```

```
import pandas as pd
ClassXII={'Name':['Pratibha','Ritika','Saumya','Aryan'],\
          'Subject':['CS','Phy','Chem','Maths'],\
          'Score':[99,87,88,67],\
          'Grade':['A1','A2','A2','B']}
df=pd.DataFrame(ClassXII,columns=['Name','Subject','Score','Grade'])
```

- pivot table .fillna( ) के साथ बनाना बनाना →

```
>>> pv=df.pivot(index='Name',columns='Subject',values='Score').fillna('')
>>> pv
Subject    CS Chem Maths Phy
Name
Aryan                  67
Pratibha   99
Ritika                     87
Saumya          88
```

हम इस pivot table में देख सकते हैं कि एक नयी table बनी है और Score column की values अलग अलग column में आयीं हैं | जबकि इसकी Name और Subject column, original table से match हो रही हैं| जहाँ values match नहीं हो रही हैं उस जगह NaN (None) भी नहीं आया बस रिक्त स्थान आगया है |

Multiple columns के द्वारा pivoting करना

इसमें बस values parameter को हटा देते हैं |

syntax → pandas.pivot(index, columns)

- उदाहरण→ DataFrame बनाना

```
>>> df
      Name Subject  Score Grade
0  Pratibha      CS     99    A1
1    Ritika     Phy     87    A2
2    Saumya    Chem     88    A2
3     Aryan   Maths     67     B
```

```
import pandas as pd
ClassXII={'Name':['Pratibha','Ritika','Saumya','Aryan'],\
          'Subject':['CS','Phy','Chem','Maths'],\
          'Score':[99,87,88,67],\
          'Grade':['A1','A2','A2','B']}
df=pd.DataFrame(ClassXII,columns=['Name','Subject','Score','Grade'])
```

- pivot table .fillna( ) के साथ बनाना बनाना →

```
>>> pv=df.pivot(index='Name',columns='Subject')
>>> pv
          Score                         Grade
Subject      CS  Chem Maths   Phy     CS Chem Maths  Phy
Name
Aryan       NaN   NaN  67.0   NaN    NaN  NaN     B  NaN
Pratibha   99.0   NaN   NaN   NaN     A1  NaN   NaN  NaN
Ritika      NaN   NaN   NaN  87.0    NaN  NaN   NaN   A2
Saumya      NaN  88.0   NaN   NaN    NaN   A2   NaN  NaN
```

Multiple columns के द्वारा pivoting करना. . .

- पिछले उदहारण में हमने देखा कि कई index बन गए Subjects के और उनकी values भी एक बार Score के लिए और एक बार Grade के लिए प्रत्येक नाम के लिए दिखाई गयी है |
- इन्हें हम फ़िल्टर भी कर सकते हैं →

```
>>> pv.Score.fillna('')
Subject    CS Chem Maths Phy
Name
Aryan                  67
Pratibha   99
Ritika                    87
Saumya            88
```

```
>>> pv.Score.CS.fillna('')
Name
Aryan
Pratibha    99
Ritika
Saumya
Name: CS, dtype: object
```

```
>>> pv.Grade.CS.fillna('')
Name
Aryan
Pratibha    A1
Ritika
Saumya
Name: CS, dtype: object
```

```
>>> pv.Grade.Maths.fillna('')
Name
Aryan       B
Pratibha
Ritika
Saumya
Name: Maths, dtype: object
```

Pivot Problem

- हमेशा ध्यान रखिये यदि index और columns के multiple values के साथ combination होंगे तो value error आयेगी |

```
import pandas as pd
ClassXII={'Name':['Pratibha','Ritika','Saumya','Aryan','Pratibha'],\
          'Subject':['CS','Phy','Chem','Maths','CS'],\
          'Score':[99,87,88,67,98],\
          'Grade':['A1','A2','A2','B','A1']}
df=pd.DataFrame(ClassXII,columns=['Name','Subject','Score','Grade'])
```

```
>>> df
       Name Subject  Score Grade
0  Pratibha      CS     99    A1
1    Ritika     Phy     87    A2
2    Saumya    Chem     88    A2
3     Aryan   Maths     67     B
4  Pratibha      CS     98    A1
>>> pv=df.pivot(index='Name',columns='Subject',values='Score')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    pv=df.pivot(index='Name',columns='Subject',values='Score')
  File "C:\Users\KVBBKServer\AppData\Local\Programs\Python\Python36\lib\site-packag
es\pandas\core\frame.py", line 5194, in pivot
    return pivot(self, index=index, columns=columns, values=values)
  File "C:\Users\KVBBKServer\AppData\Local\Programs\Python\Python36\lib\site-packag
es\pandas\core\reshape\reshape.py", line 415, in pivot
    return indexed.unstack(columns)
ValueError: Index contains duplicate entries, cannot reshape
```

stack( ) और unstack( ) methods का प्रयोग

- stack( ) और unstack( ) methods दोनों DataFrame का layout पलट देते हैं अर्थात columns के सारे levels को row में और row के सारे levels को column में पलट देते हैं | DataFrame की ***stacking*** का मतलब है innermost column index को innermost row index की ओर ले जाना | और इसके उलटे क्रम को ***unstacking*** कहते हैं |

```
import pandas as pd
ClassXII={'Name':['Pratibha','Ritika','Saumya','Aryan'],\
          'Subject':['CS','Phy','Chem','Maths'],\
          'Score':[99,87,88,67],\
          'Grade':['A1','A2','A2','B']}
df=pd.DataFrame(ClassXII,columns=['Name','Subject','Score','Grade'])
```

```
>>> pv=df.pivot(index='Name',columns='Subject')
>>> pv
        Score                       Grade
Subject    CS  Chem Maths   Phy     CS Chem Maths  Phy
Name
Aryan     NaN   NaN  67.0   NaN    NaN  NaN     B  NaN
Pratibha 99.0   NaN   NaN   NaN     A1  NaN   NaN  NaN
Ritika    NaN   NaN   NaN  87.0    NaN  NaN   NaN   A2
Saumya    NaN  88.0   NaN   NaN    NaN   A2   NaN  NaN
```

Stack Method का प्रयोग करने पर subjects जो horizontal में थे वो सभी vertical आगये | और यहाँ यह column breakdown में last level लेता है और इसे last row breakdown में बदल देता है

Stack Method का प्रयोग

```
>>> pv.stack()
                  Score Grade
Name     Subject
Aryan    Maths     67.0     B
Pratibha CS        99.0    A1
Ritika   Phy       87.0    A2
Saumya   Chem      88.0    A2
```

stack( ) और unstack( ) methods का प्रयोग

```
import pandas as pd
ClassXII={'Name':['Pratibha','Ritika','Saumya','Aryan'],\
          'Subject':['CS','Phy','Chem','Maths'],\
          'Score':[99,87,88,67],\
          'Grade':['A1','A2','A2','B']}
df=pd.DataFrame(ClassXII,columns=['Name','Subject','Score','Grade'])
```

```
>>> pv=df.pivot(index='Name',columns='Subject')
>>> pv
        Score                         Grade
Subject    CS  Chem Maths   Phy    CS Chem Maths  Phy
Name
Aryan     NaN   NaN  67.0   NaN   NaN  NaN     B  NaN
Pratibha 99.0   NaN   NaN   NaN    A1  NaN   NaN  NaN
Ritika    NaN   NaN   NaN  87.0   NaN  NaN   NaN   A2
Saumya    NaN  88.0   NaN   NaN   NaN   A2   NaN  NaN
```

```
>>> pv.stack()
                  Score Grade
Name     Subject
Aryan    Maths     67.0     B
Pratibha CS        99.0    A1
Ritika   Phy       87.0    A2
Saumya   Chem      88.0    A2
```

```
>>> pv.stack().stack()
Name     Subject
Aryan    Maths    Score    67
                  Grade     B
Pratibha CS       Score    99
                  Grade    A1
Ritika   Phy      Score    87
                  Grade    A2
Saumya   Chem     Score    88
                  Grade    A2
dtype: object
```

इस तरह से भी stack का प्रयोग किया जा सकता है जिसके stacking के बाद पुनः stacking की गयी है तो इसमें यह बचे हुए column level को भी move कर देता है |

stack( ) और unstack( ) methods का प्रयोग

```
import pandas as pd
ClassXII={'Name':['Pratibha','Ritika','Saumya','Aryan'],\
          'Subject':['CS','Phy','Chem','Maths'],\
          'Score':[99,87,88,67],\
          'Grade':['A1','A2','A2','B']}
df=pd.DataFrame(ClassXII,columns=['Name','Subject','Score','Grade'])
```

```
>>> pv=df.pivot(index='Name',columns='Subject')
>>> pv
        Score                         Grade
Subject    CS  Chem Maths   Phy     CS Chem Maths  Phy
Name
Aryan     NaN   NaN  67.0   NaN    NaN  NaN     B  NaN
Pratibha 99.0   NaN   NaN   NaN     A1  NaN   NaN  NaN
Ritika    NaN   NaN   NaN  87.0    NaN  NaN   NaN   A2
Saumya    NaN  88.0   NaN   NaN    NaN   A2   NaN  NaN
```

```
>>> pv.stack(0)
Subject           CS Chem Maths  Phy
Name
Aryan    Grade   NaN  NaN     B  NaN
         Score   NaN  NaN    67  NaN
Pratibha Grade    A1  NaN   NaN  NaN
         Score    99  NaN   NaN  NaN
Ritika   Grade   NaN  NaN   NaN   A2
         Score   NaN  NaN   NaN   87
Saumya   Grade   NaN   A2   NaN  NaN
         Score   NaN   88   NaN  NaN
```

यह unstacking है

```
>>> pv.stack(0).stack()
Name                 Subject
Aryan     Grade      Maths          B
          Score      Maths         67
Pratibha  Grade      CS            A1
          Score      CS            99
Ritika    Grade      Phy           A2
          Score      Phy           87
Saumya    Grade      Chem          A2
          Score      Chem          88
dtype: object
```

Unstacking भी stack के जैसे ही होती है बस उसमे एक argument '0' pass कर दिया जाता है |

pivot_table( ) method का प्रयोग करके Pivoting करना

- यह pivot( ) method का generalization है |
- जब आपके पास एक ही index या column के लिए duplicate values हों तो pivot_table( ) method का प्रयोग किया जाता है |
- एक pivot table में counts, sums तथा table data से सम्बंधित अन्य functions भी उपलब्ध होते हैं |
- pivot_table( ) method एक प्रकार से excel sheet जैसी ही DataFrame बनाती है |
- यह भी row को column में और column को row में बदलने के काम आता है |
- यह किसी भी data field की grouping को allow करता है |
- इसका syntax है →

```
pandas.pivot_table (DataFrame, values=None, index=None,
                    columns=None, aggfunc='mean',
                        fill_value=None, margins=False, dropna=True,
                        margins_name='All')
```

- .pivot_table() method में सारे arguments ज़रूरी नहीं हैं क्योंकि इसके अन्दर कुछ default values होती हैं |

pivot_table( ) method का प्रयोग करके Pivoting करना…

pandas.pivot_table (DataFrame, values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All')

- .pivot_table() method में सारे arguments ज़रूरी नहीं हैं क्योंकि इसके अन्दर कुछ default values होती हैं |
- इसके syntax में -
  - **DataFrame** → एक pandas DataFrame है |
  - **values**→यह optional है और aggregate किये जाने वाला column है |
  - **index**→column, grouper, array, या list का नाम है |
  - **columns**→यह column, grouper, array, या list है |
  - **aggfunc** → aggregation function है |
  - **fill_value**→इसके द्वारा हम default values सेट कर सकते हैं यदि values न दी गयीं हो |
  - **margins**→ यह एक boolean होता है जिसका default false होता है, यदि हम इसे true कर देते हैं तो resulting dataframe में row और column का sum भी जुड़ जाता है |
  - **dropna**→यदि यह true है तो यह missing data वाली row को drop कर देता है |
  - **margins_name='All'** →जब margins true हो तब total वाले row और column के नाम रखता है |

pivot_table( ) method का प्रयोग करके Pivoting करना ...

हम निम्न data को consider करके pivot table बनाते हैं →

```
import pandas as pd
ClassXII={'Name':['Pratibha','Ritika','Saumya','Aryan',\
                  'Pratibha','Ritika','Saumya','Aryan',\
                  'Pratibha','Ritika','Saumya','Aryan',\
                  'Pratibha','Ritika','Saumya','Aryan'],\
          'Test':['Semester1','Semester1','Semester1','Semester1',\
                  'Semester1','Semester1','Semester1','Semester1',\
                  'Semester2','Semester2','Semester2','Semester2',\
                  'Semester2','Semester2','Semester2','Semester2'],\
          'Subject':['CS','CS','CS','CS','IP','IP','IP','IP',\
                     'CS','CS','CS','CS','IP','IP','IP','IP'],\
          'Marks':[99,87,88,67,98,86,88,68,97,85,89,62,94,84,81,69]}
df=pd.DataFrame(ClassXII,columns=['Name','Test','Subject','Marks'])
```

```
>>> df
        Name       Test Subject  Marks
0   Pratibha  Semester1      CS     99
1     Ritika  Semester1      CS     87
2     Saumya  Semester1      CS     88
3      Aryan  Semester1      CS     67
4   Pratibha  Semester1      IP     98
5     Ritika  Semester1      IP     86
6     Saumya  Semester1      IP     88
7      Aryan  Semester1      IP     68
8   Pratibha  Semester2      CS     97
9     Ritika  Semester2      CS     85
10    Saumya  Semester2      CS     89
11     Aryan  Semester2      CS     62
12  Pratibha  Semester2      IP     94
13    Ritika  Semester2      IP     84
14    Saumya  Semester2      IP     81
15     Aryan  Semester2      IP     69
```

हम यह data CSV फाइल से भी le सकते हैं

```
>>> pv=df.pivot_table(index='Name',values='Marks',aggfunc='mean')
```

या

```
>>> pv=df.pivot_table(index='Name',aggfunc='mean')
>>> pv
           Marks
Name
Aryan       66.5
Pratibha    97.0
Ritika      85.5
Saumya      86.5
```

pivot_table( ) method का प्रयोग करके Pivoting करना …

pivot table को हम निम्न तरीके से भी बना सकते हैं →

```
>>> pd.pivot_table(df,index='Name',aggfunc='mean')
         Marks
Name
Aryan     66.5
Pratibha  97.0
Ritika    85.5
Saumya    86.5
```

```
>>> pv=df.pivot_table(index='Name',columns='Subject',aggfunc='mean')
>>> pv
        Marks
Subject    CS    IP
Name
Aryan    64.5  68.5
Pratibha 98.0  96.0
Ritika   86.0  85.0
Saumya   88.5  84.5
```

यहाँ aggfunc के values पर ध्यान दीजिये |

```
>>> pv=df.pivot_table(index='Name',columns='Subject',aggfunc='count')
>>> pv
        Marks     Test
Subject    CS IP   CS IP
Name
Aryan       2  2    2  2
Pratibha    2  2    2  2
Ritika      2  2    2  2
Saumya      2  2    2  2
```

pivot_table( ) method का प्रयोग करके Pivoting करना …

आपके लिए एक excercise →

An Emp table contains the following data:

Empno	Name	Department	Salary	Commission	Job
100	Sunita Sharma	RESEARCH	45600	5600.0	CLERK
101	Ashok Singhal	SALES	43900	3900.0	SALESMAN
102	Sumit Avasti	SALES	27000	7000.0	SALESMAN
103	Jyoti Lamba	RESEARCH	45900	4900.0	MANAGER
104	Martin S.	SALES	32500	3500.0	SALESMAN
105	Binod Goel	SALES	45200	4200.0	MANAGER
106	Chetan Gupta	ACCOUNTS	36800	6800.0	MANAGER
107	Sudhir Rawat	RESEARCH	37000	7000.0	ANALYST
108	Kavita Sharma	ACCOUNTS	42900	4900.0	CLERK
109	Tushar Tiwari	SALES	49500	4500.0	MANAGER
110	Anand Rathi	OPERATIONS	41600	8200.0	SR. MANAGER
111	Sumit Vats	RESEARCH	47800	NaN	SR. MANAGER
112	Manoj Kaushik	OPERATIONS	43600	NaN	CLERK

(a) Using above table create a DataFrame called dfE.
(*b*) Display the department wise total salary.
(*c*) Display the department wise average salary.
(*d*) Display the department wise total and average salary.
(*e*) Display the department wise maximum and minimum salary.
(*f*) Display the department and job wise maximum salary.

pivot_table( ) method का प्रयोग करके Pivoting करना …

Solution→

सबसे पहले आप दी गयी table का DataFrame बनायेंगे pandas का प्रयोग करके |

उसके बाद आपको निम्न function अप्लाई करने हैं -

(*b*) `pd.pivot_table(dfE, index='Department', values='Salary', aggfunc='sum')`

(*c*) `pd.pivot_table(dfE, index='Department', values='Salary')`

Or

`pd.pivot_table(dfE, index='Department', values='Salary', aggfunc='mean')`

(*d*) `pd.pivot_table(dfE, index='Department', values='Salary', aggfunc=['sum', 'mean'])`

(*e*) `pd.pivot_table(dfE, index='Department', values='Salary', aggfunc=['max', 'min'])`

(*f*) `pd.pivot_table(dfE, index=['Department', 'Job'], values='Salary', aggfunc='max')`

DataFrames की Sorting

- DataFrame के data को भी row और column के values के आधार पर sort किया जा सकता है |
- By default sorting, row labels पर होती है वो भी बढ़ते हुए क्रम में |
- Pandas DataFrames के पास दो उपयोगी sort functions होते हैं →
  - sort_values( ): यह function दिए गए column के data को ascending या descending आर्डर में sort करता है |
  - sort_index( ): यह function rows (axis=0) या columns (axis=1) को sort करता है
- इनका syntax निम्नवत है →
- *DataFrame.sort_values(by = None, axis=0, ascending = True, inplace = False)*
- *DataFrame.sort_index(by = None, axis=0, ascending = True, inplace = False)*
- यहाँ –
  - **by:** sort किया जाने वाला column
  - **axis:** यहाँ 0 pass करने का मतलब सोर्टिंग row wise और 1 का मतलब column wise
  - **ascending:** default में ascending true रहता है
  - **inplace:** default false होती है यदि आप नया dataframe नहीं चाहते हैं तो true pass करना होगा |

DataFrames की Sorting...

```
>>> df
      Name Subject  Marks Grade
0  Pratibha      CS     99    A+
1    Ritika      IP     87     A
2    Saumya     PHY     88    B+
3     Aryan    CHEM     67     B
```

यहाँ by default ascending आर्डर में sort हुआ है |

```
>>> dfn=df.sort_values('Name')
```

या

```
>>> dfn=df.sort_values(by='Name')
>>> dfn
      Name Subject  Marks Grade
3     Aryan    CHEM     67     B
0  Pratibha      CS     99    A+
1    Ritika      IP     87     A
2    Saumya     PHY     88    B+
```

Descending आर्डर में sort करने के लिए निम्न उदाहरण है |

```
>>> dfn=df.sort_values(by='Name',ascending=False)
>>> dfn
      Name Subject  Marks Grade
2    Saumya     PHY     88    B+
1    Ritika      IP     87     A
0  Pratibha      CS     99    A+
3     Aryan    CHEM     67     B
```

Ascending parameter की वैल्यू False pass कर दी है |

```
>>> dfn=df.sort_values(['Name','Marks'],ascending=True)
>>> dfn
      Name Subject  Marks Grade
3     Aryan    CHEM     67     B
0  Pratibha      CS     99    A+
1    Ritika      IP     87     A
2    Saumya     PHY     88    B+
```

यहाँ यदि हम दो column इस प्रकार से देदेते हैं तो multiple columns पर sorting apply हो जाती है |

Sort by index

```
>>> dfn=df.sort_index()
>>> dfn
       Name Subject  Marks Grade
0  Pratibha      CS     99    A+
1    Ritika      IP     87     A
2    Saumya     PHY     88    B+
3     Aryan    CHEM     67     B
```

यहाँ यह ascending आर्डर में sorting है |

```
>>> dfn=df.sort_index(ascending=False)
>>> dfn
       Name Subject  Marks Grade
3     Aryan    CHEM     67     B
2    Saumya     PHY     88    B+
1    Ritika      IP     87     A
0  Pratibha      CS     99    A+
```

यहाँ यह descending आर्डर में sorting है |

याद रखने योग्य बातें:

1. pivot( ) method एक नयी table बनाता है जिनके row और column unique होते हैं |
2. pivot( ) method का उपयोग aggregation के बिना pivot के लिए किया जाता है।
3. *stacking* का मतलब है innermost column index को innermost row index की ओर ले जाना |

- कृपया हमारे ब्लॉग को फॉलो करिए और youtube channel को subscribe करिए | ताकि आपको और सारे chapters मिल सकें |

www.pythontrends.wordpress.com