Class XII, IP, Python Notes Chapter II – Python Pandas

by V. Khatri, PGT CS,KV1 Jammu

Pandas : Pandas is an open-source library of python providing high-performance data manipulation and analysis tool using its powerful data structure, there are many tools available in python to process the data fast Like-Numpy, Scipy, Cython and Pandas(Series and DataFrame).
Data of Series is always mutable . It means, it can be changed. But the size of data of Series is size immutable , means can not be changed.

DataFrame -It is a 2-dimensional data structure with columns of different types. It is just similar to a spreadsheet or SQL tabl. It is generally the most commonly used pandas object. It has index values as well as columns name.

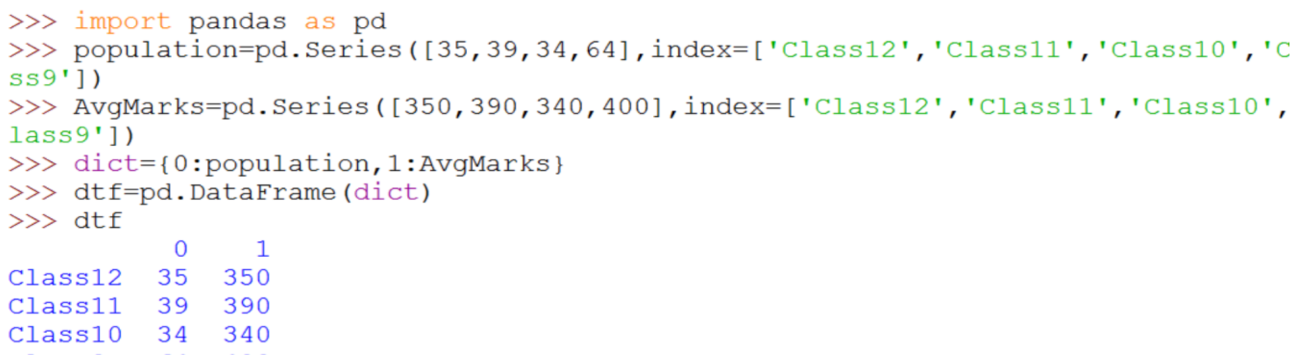
Series : It is also Pandas Data structure that contains one dimensional array like objects, it uses index for accessing items, it does not have columns name like Dataframe

You can create a DataFrame by various methods by passing data values. Like-

• **2D dictionaries** - d = {'2016':{'A':25000,'B':30000},'2017':{'A':36000,'B':34000}} it will create a DataFrame with index A and B, coloumns will be 2016 and 2017

• **2D ndarrays :** a= np.array([1,2,3],[4,5,6]) df=pd.DataFrame(a) other examples were explained in Numpy

• **Creation of DataFarme from 2D Dictionary of same Series Object :**



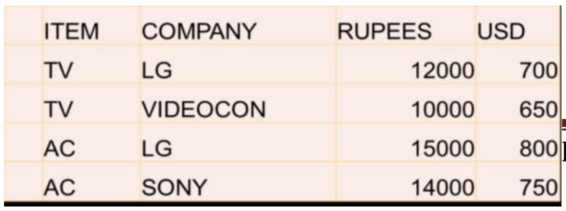
```
>>> import pandas as pd
>>> population=pd.Series([35,39,34,64],index=['Class12','Class11','Class10','C
ss9'])
>>> AvgMarks=pd.Series([350,390,340,400],index=['Class12','Class11','Class10',
lass9'])
>>> dict={0:population,1:AvgMarks}
>>> dtf=pd.DataFrame(dict)
>>> dtf
          0    1
Class12  35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

• **Another DataFrame object :** Df1=pd.DataFrame(df) where df is a already created DataFrame

Pivot function – Pivot reshapes data and uses unique values from index/ columns to form axes of the resulting**, pandas.pivot(index, columns, values)** function produces pivot table based on 3 columns of the DataFrame. Uses unique values from index / columns and fills with values also it produces Pivot table which is used to summarize and aggregate data inside dataframe.
There are two functions available in python for pivoting dataframe.

1. pivot() - This function is used to create a new derived table(pivot) from existing dataframe. It takes 3 arguments : index,columns, and values

Given DataFrame view like -



ITEM	COMPANY	RUPEES	USD
TV	LG	12000	700
TV	VIDEOCON	10000	650
AC	LG	15000	800
AC	SONY	14000	750

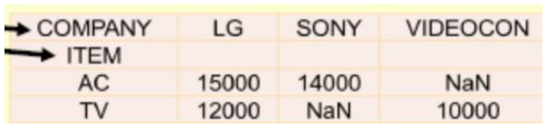
table = {"ITEM",:['TV', 'TV', 'AC', 'AC'], 'COMPANY':['LG', 'VIDEOCON', 'LG', 'SONY'], 'RUPEES': ['12000', '10000', '15000', '14000'], 'USD': ['700', '650', '800', '750']}

d = pd.DataFrame(table)

print(d) it will show Dataframe d as stated here

p = d.pivot(index='ITEM', columns='COMPANY', values='RUPEES')

it will show output as given in diagram, If we don't mention Values argument in Pivot function then it Will show the following pivot.

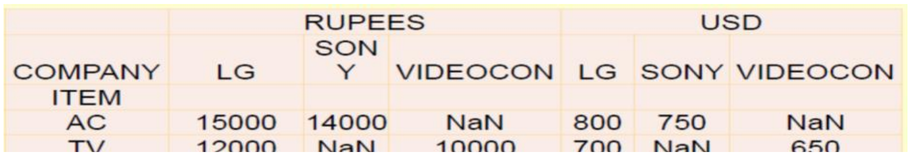


COMPANY	LG	SONY	VIDEOCON
ITEM			
AC	15000	14000	NaN
TV	12000	NaN	10000

If we command p=pd.pivot(index='ITEM',columns='COMPANY',values='RUPEES'.fillna=(' ')

This command will show all Nan values in pivot table to blank, other value will be same

When there are different Values for each item and And for similar company then

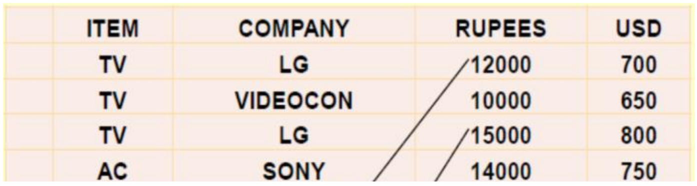


	RUPEES			USD		
COMPANY	LG	SONY	VIDEOCON	LG	SONY	VIDEOCON
ITEM						
AC	15000	14000	NaN	800	750	NaN
TV	12000	NaN	10000	700	NaN	650

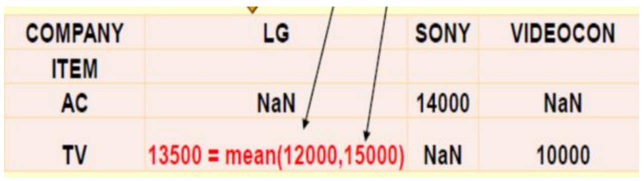
We will use pivot table()

Function instead of pivot() it will take average values of similar records as

d.pivot_table(index='ITEM', columns='COMPANY', values='RUPEES',aggfunc='mean'))



ITEM	COMPANY	RUPEES	USD
TV	LG	12000	700
TV	VIDEOCON	10000	650
TV	LG	15000	800
AC	SONY	14000	750



COMPANY	LG	SONY	VIDEOCON
ITEM			
AC	NaN	14000	NaN
TV	13500 = mean(12000,15000)	NaN	10000

We can mention other functions too like sum, count, for calculating values in aggfunc, by default it is mean that is if we don't mention aggfunc then it will take by default mean.

d.pivot_table(index='ITEM', columns='COMPANY', values='RUPEES',aggfunc='sum'))

Output : AC 290000

TV 12000

Multiple Index can be given also like :

Df.pivot_talbe(index=['Item'','country'],columns='company' values='rupees')

Data Frame Operations by using below Data Frame:

data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],

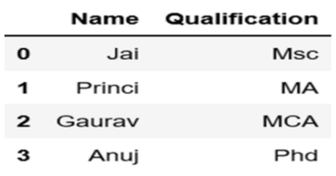
'Age':[27, 24, 22, 32],

'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],

'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

df = pd.DataFrame(data) # its index will be by default 0 to 3

print(df[['Name:'Qualification'] will show DataFrame taking Name&Qualification



	Name	Qualification
0	Jai	Msc
1	Princi	MA
2	Gaurav	MCA
3	Anuj	Phd

print(df.Name[0]] it will show Jai as output

del df['Age'] # it will delete Age column

Iterating (Looping over a DataFrame) : For Iterating over a DataFrame we use tow functions as iterows() and iteritems(), using iterrows() first we access values rows wise, after first row, second rows elements will be accessed, in iteritems() values will be accessed column wise, after completing first columns it goes to second columns, as example –

```
dict = {'Name':["Aparna", "pankaj", "sudhir", "Geeku"],
        'Degree': ["MBA", "BCA", "M.Tech" "MBA"],
        'Score':[90, 40, 80, 98]}
 df = pd.DataFrame(dict,index=['A','B','C'])              # it gives      name    aparna
for (i, j) in df.iterrows():                              degree  MBA
 print(i, j)                                              score    90
print() # Here i represent index name and j represents row wise column values, loop will run until
last row or index in the DataFrame.
```

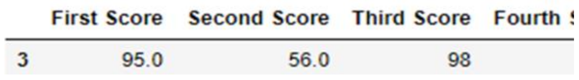
Now we iterate through columns in order to iterate through columns we use iteritems() function, like-

```
for (i,j) in df.items():              Output will be  Column index is  Name
 print('columns index is',i)           Column Values is              A   Aparna
 print('column values is',j)                                         B   MBA
                                                                     C    90
                                                  Column index is  Degree (and so on, it will continue)
```

Dropping missing values using dropna() :

In order to drop a null values from a Dataframe, we used dropna() function this function drop Rows/Columns of datasets with Null values in different ways.

```
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, 40, 80, 98],
        'Fourth Score':[np.nan, np.nan, np.nan, 65]}
 df = pd.DataFrame(dict)
df.dropna()
```



	First Score	Second Score	Third Score	Fourth S
3	95.0	56.0	98	

it will delete all rows containing of none value and will output as above.

Filling missing values using fillna() function:

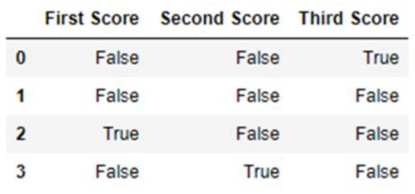
In order to fill null values in a datasets, we use fillna() function these function replace NaN values with some value of their own. This function help in filling a null values in datasets of a DataFrame. Interpolate() function is basically used to fill NA values in the dataframe but it uses various interpolation technique to fill the missing values rather than hard-coding the value.

```
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],'Third Score':[np.nan, 40, 80, 98]}
```

df = pd.DataFrame(dict)

df.fillna(0)   # it will fill 0.0 in every place of np.nan

df.isnull()   # this command checks null value, null values will be shown as True and other values will be shown as False



	First Score	Second Score	Third Score
0	False	False	True
1	False	False	False
2	True	False	False
3	False	True	False

loc command : if d={'Name':['jai','gourav','diksha'], 'Roll':[1,2,3],'Address':['jammu','delhi','jaipur']}

df=pd.DataFrame(d,index=['A','B','C'])

When we want to apply conditions on both rows and columns we use loc commands as -

print(df.loc['A' :'C', 'Name':'Address']   # Notice that comma is used to separate Row data and column Data also it will show Name to Address column including Roll also rows from A to C including B, in every Row and Column combined address Row address should be given first.

Print(df.loc[['A' :'B', :]]   # it will give A and B Rows information showing all column information.

iloc command : it uses index instead of Rows name and columns name as :

print(df.iloc[0:2, 1:3] it will show rows from index 0 to 1 and columns from 1 to 2

Topic - at and iat command : Syntax of these commands are

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

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

Example : df.at['B','Roll'] it will give output as 2

Also df.iat(2,2) # it will give output as Jaipur which is at 2 index Row and 2 Column.

Note : df['subject']=['ip','cs','maths']   # it will create another column as Subject with values.

Also in at and iat command we can give rows and columns values and index

df.at['A','Name':'Address']=['Manav',4,'Kota'] it will change first Row information

df.at['D',:]=['Man',5,'Delhi'] it will create a new Row of Named D with index value as 3

df.iat[1:2]='Goa'   # it will change index 1 i.e. B, Address value to Goa in place of Delhi.

Notice : iat command accept index only in figure not in range, if we give range of rows or columns it will show error like df.iat[0:2,3] or df.iat[2,1:2]

Making DataFrame by fetching data from a Excel file with extension .csv

data = pd.read_csv("nba.csv", index_col ="Name")

# This command will access nba.csv file which will be created in Excel with .csv extension and data will be created as a DataFrame, its index column will be Name which should be as a column in .csv file extension.

Descriptive (Aggregate) Functions : Min(), Max(), mean(), mode(),median(),count(), sum() etc.

d = {'2016':{'A':25000,'B':30000},'2017':{'A':36000,'B':34000}}

df = pd.DataFrame(d) #ThisDataFrame has A,B AND C as indexes and 2016, 2017 as columns

df.min() -it will take axis default as 0 and give give min value in each column, like   A   25000

B   30000

if we give commands as df.min(axix=1) then it will calculate columns wise value and give output as 2016 25000,   2017   34000

Other function like mean(), mode() and Median(), count() and sum() etc. may be applied same.

Df.count()

Df.sum(axis=1)

Df.count()	Df.sum(axis=1)
2016 2	A 61000
2017 =2	B 64000

df.columns =['Col_1', 'Col_2', 'Col_3', 'Col_4']

# This will change columns Name

df.index = ['Row_1', 'Row_2', 'Row_3', 'Row_4'] # This will change index names

other function is std() which denotes standard deviation, it can be calculated row wise or by columns:

df.std()   # This will show standard deviation row wise like A   Std. Dev then B   Std. Dev. etc

and df.std(axis=1) # This will calculate column wise like 2019 Std. Dev. Then 2017   Std, Dev and so on

mad()   # This is a function to calculate mean absolute deviation, like –

df.mad(axis=1, skipna=None) this will calculate column wise also it will not skip na or None values.

Code for renaming index and columns name in DataFrame by using rename (), reindex(), reindex_like() etc:

Like above example, other way to change index name or columns name by using rename() is -

df.rename(index={index={"A": "a", "B": "b", "C": "c"} columns={'Name':"nm','Age':'ag','Score':'sc'}, ,inplace=True)   # This will change index and columns name as specified in code

When we write inplace=True then it will not create another DataFrame and changes will be seen in current DataFrame but when we specify inplaced=False then it will return another DataFrame

df1=df.rename(index={'Name':"nm','Age':'ag','Score':'sc'}, columns={"A": "a", "B": "b", "C": "c"},inplace=False)   # Here inplace False is mentioned

Print(df1)   # it will show changed columns in DataFrame otherwise will be same for df

We can also change indexes by using reindex() function

As df.reindex(['a','b','c,'d'])   it will there are only three indexes then d index will show Nan values

In DataFrame, we can fill Nan values with specified value by using fill_value command, example

As df.reindex(['a','b','c,'d'],fillValue=1000) # it will show all values 10000 in d index row which was showing Nan value previously.

Another function is reindex_like() : It will match two DataFrame, and first DataFrame will be made equal to second DatafFrame, index will be from second, also same columns will be made, example

df = {'2016':{'A':25000,'B':30000},'2017':{'A':36000,'B':34000}}

df1 = {'2019':{'A':25000,'B':30000},'2017':{'A':36000,'C':34000}} df1.reindex_like(df)

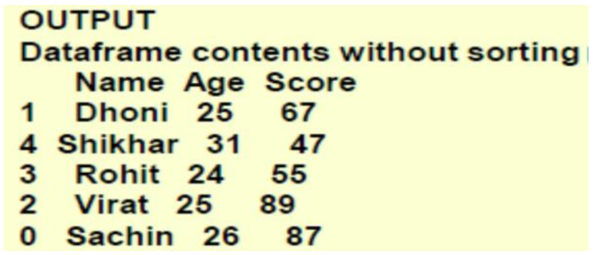
then it will show output as –

	2016	2017
A	NaN	36000
B	NaN	NaN

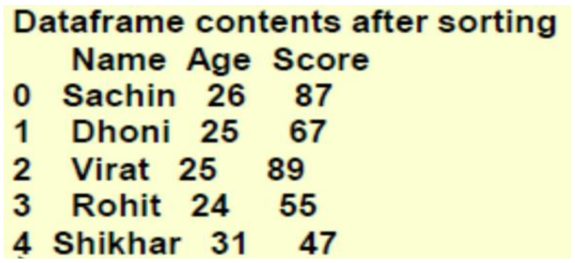
Sorting - DataFrame

Sorting means arranging the contents in ascending or descending order. There are two kinds of sorting available in pandas(Dataframe).

1. **By value(column)** d={'Name':pd.['Sachin','Dhoni','Virat','Rohit','Shikhar'],\
   'Age':[26,27,25,24,31],'Score':[87,89,67,55,47]}
   df = pd.DataFrame(d) df=df.sort_values(by='Score')
   It will make DataFrame df by sorting Score in ascending order
   If we give commands as df=df.sort_values(by='Score',ascending=0)
   then it wil show DataFrame by Showing score values in Descending order.
   df=df.sort_values(by=['Age', 'Score'],ascending=[True,False])
   It will show Age in ascending order but if two persons age is same then it will take first whose Score is High, as here Score is in descending order.
2. **By index -** Sorting over dataframe index sort_index() is supported by sort_index() method
   df=df.reindex([1,4,3,2,0]) it will change the index of the DataFrame according to given index
   df1=df.sort_index() Changed DataFrame will be again sorted by index in ascending order



```
OUTPUT
Dataframe contents without sorting
     Name  Age  Score
1   Dhoni   25    67
4 Shikhar   31    47
3   Rohit   24    55
2   Virat   25    89
0  Sachin   26    87
```

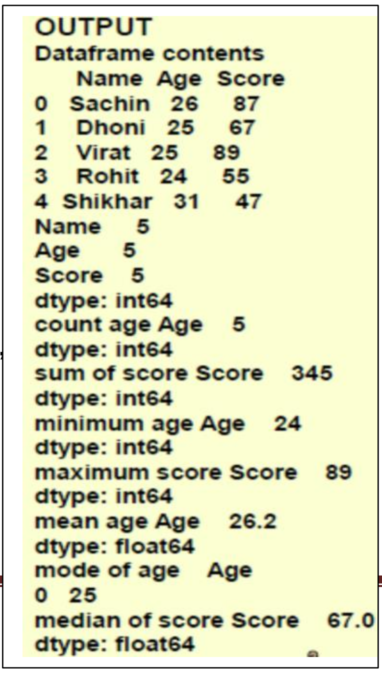


```
Dataframe contents after sorting
     Name  Age  Score
0  Sachin   26    87
1   Dhoni   25    67
2   Virat   25    89
3   Rohit   24    55
4 Shikhar   31    47
```

df1=df.sort_index(ascending=0) this command will show DataFrame in descending order of index.

Data aggregation – Aggregation is the process of turning the values of a dataset (or a subset of it)into one single value or data aggregation is a multivalued function ,which require multiple values and return a single value as a result.There are number of aggregations possible like count,sum,min,max,median, quartile etc, take an example -

d = {'Name':pd.Series(['Sachin','Dhoni','Virat','Rohit','Shikhar']),
'Age':pd.Series([26,25,25,24,31]),



```
OUTPUT
Dataframe contents
     Name  Age  Score
0  Sachin   26    87
1   Dhoni   25    67
2   Virat   25    89
3   Rohit   24    55
4 Shikhar   31    47
Name     5
Age      5
Score    5
dtype: int64
count age Age    5
dtype: int64
sum of score Score    345
dtype: int64
minimum age Age    24
dtype: int64
maximum score Score    89
dtype: int64
mean age Age    26.2
dtype: float64
mode of age    Age
0   25
median of score Score    67.0
dtype: float64
```

'Score':pd.Series([87,67,89,55,47])}

df = pd.DataFrame(d)

print(df.count())

print("count age",df['Age'].count())

print("sum of score",df['Score'].sum())

print("minimum age",df['Age'].min())

print("maximum score",df['Score'].max())

print("mean age",df['Age'].mean())

print("mode of age",df['Age'].mode())

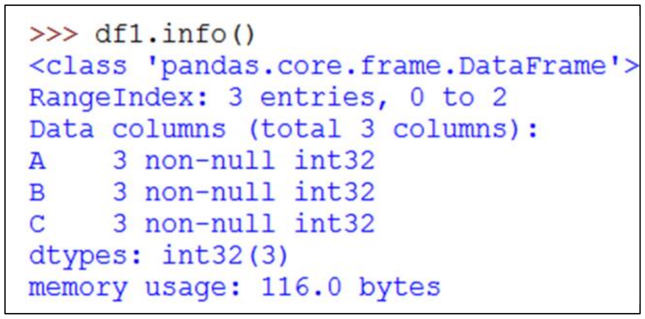
print("median of score",df['Score'].median())

Other important functions of DataFrame are as under-

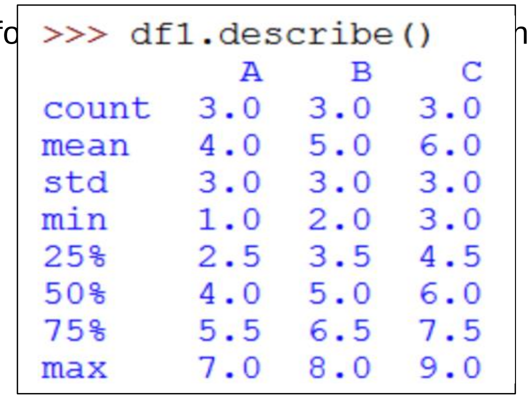
(1) <DF>.info ( )

<DF>.describe ( )

Info function gives information about columns values, here A, B, C are columns of DataFrame and describe function gives all stated functions fo n values.



```
>>> 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
```

(2) head() and tail() : head function gives half of total rows in a DataFrame and tail gives second part of remaining half rows, if total records are in odd numbers then common row records will be in both, suppose total rows data is in 9 rows then head () will show 1 to 5 rows records and tail () will show from 5 to 9 rows dats.

d={'A':[1,2,3,4,5,6],'B':[1,2,3,4,5,6],'C':[1,2,3,4,5,6]}

df=pd.DataFrame(d)

print(df.head(n=3))   # it will show rows from index 0 to 2, total 3 records

print(df.tail(n=2))   # it will show rows from index 3 to 4,total 2 records

3. **idmax() and idmin() function:** This gives maximum and minimum indexes in columns

d={'Roll':[1,2,3,4,5],'Age':[26,27,25,24,31]}

df = pd.DataFrame(d)   This will show output as   Roll   4

print(df.idxmax())   Age   4

print(df.idxmin())   Roll   0   and Age   0

4. **cumsum () function :** It shows row wise or column wise sum of previous rows or columns

if d={'A':[1,2,3],'B':[4,5,6],'C':[7,8,9]}

df = pd.DataFrame(d)

print(df.cumsum()) # # it will sum values column wise

print(df.cumsum(), axis=1) # it will sum values column wise

Cumsum()	A	B	C	**For axis(1)**	A	B	C
0	1	2	3	0	1	3	6
1	5	7	9	1	4	9	15
2	12	15	18	2	7	15	24

Quantile : The word "quantile" comes from the word quantity. means, a quantile is where a sample is divided into equal-sized or subgroups (that's why it's sometimes called a "fractile"). Quantile statistics is a part of a data set. It is used to describe data in a clear and understandable way. The 0,30 quantile is basically saying that 30 % of the observations in our data set is below a given line. It returns the value at the given quanltiles over Requested axis (0 or 1)

The median is a kind of quantile; the median is placed in a probability distribution at center so that exactly half of the data is lower than the median and half of the data is above the median. . Quartiles are quartiles; when they divide the distribution into four equal parts. Deciles are quantiles that divide a distribution into 10 equal parts and Percentiles when that divide a distribution into 100 equal parts . To Know .3 quantile the formula is q(n+1), where q is .3 and n is the total items in the list or DataFrame, i.e. if n=40 then.3(41)=12.3, this proves that 30 % of Data is up to 12.3 and rest above.

s = pd.DataFrame([3, 4, 5,6,8,10,12,16,18,20, 25])

r=s.quantile(.3) # this will calculate r value as =.3(11+1)=3.6

print(r) #taking round of 3.6 is 4 and 4th number in DataFrame is 6, so it the answer.

df = pd.DataFrame(np.array([[1, 1], [2, 10], [3, 100], [4, 1000]]),columns=['a', 'b'])

print(df)

print(df.quantile(0.4)) #DataFrame is a 2X4 Matrix and it will calculate quantile column wise and output will be

```
a    2.2
b   28.0
```

if we command as print(df.quantile(0.4,axis=1)) # it will calculate quantile row wise by taking values of columns for each row, output will be 0 2.2, 1 5.2 2 41.8 3 402.4

Variance : It is used in statistics for probability distribution. Since **variance** measures the variability (volatility) from an average or mean and volatility is a measure of risk, the **variance** statistic can help determine the risk an investor might assume when purchasing a specific security, like ..

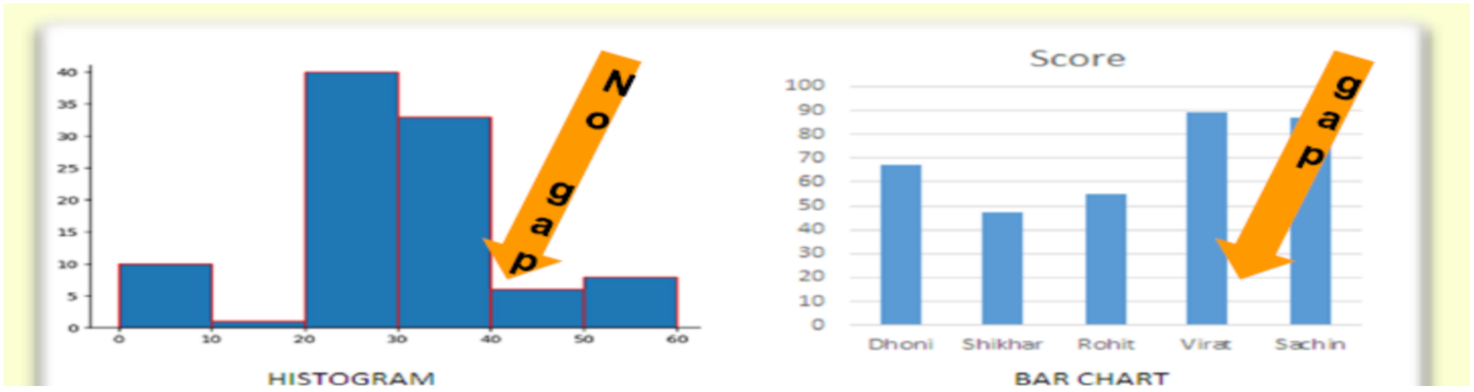
df = pd.DataFrame(np.array([[1, 1], [2, 10], [3, 100],]),columns=['a', 'b'])

print(df.var())   # it will show result as a   1   ,   b   2997.0

print(df.var(axis=1)) # it will calculate variance for each row and output will be

```
0      0
1      3
2   4704.5
```

Histogram : A histogram is a powerful technique in data visualization. It is an accurate graphical representation of the distribution of numerical data. It was first introduced by Karl Pearson. **Matplotlib** can be used to create histograms. A histogram shows the frequency on the vertical axis and the horizontal axis. Usually it has bins, where every bin has a minimum and maximum value. Each bin also has a frequency between x



Difference of Bar Chart and Histogram is that A bar chart majorly represents categorical data while histograms on the other hand, is used to describe distributions.

To draw histogram in python following concepts must be clear

Title –To display heading of the histogram.

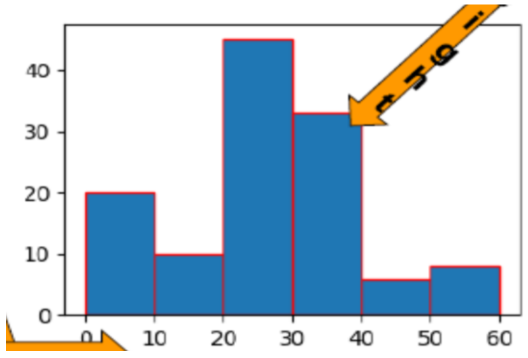
Color – To show the color of the bar.

Axis: y-axis and x-axis.

Data: The data can be represented as an array.

Height and width of bars. This is determined based on the analysis. The width of the bar is called bin or intervals.

Border color –To display border color of the bar.



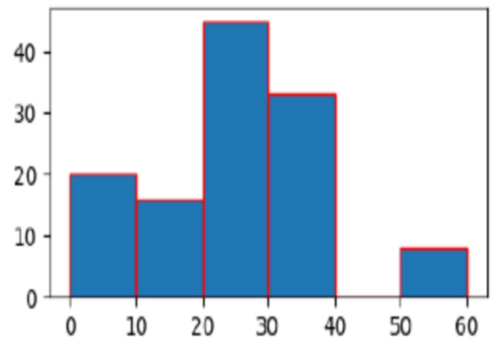
Example : import numpy as np

import matplotlib.pyplot as plt

plt.hist([5,15,25,35,45, 55], bins=[0,10,20,30,40,50, 60],/
 weights=[20,10,45,33,6,8], edgecolor="red")

plt.show()   # Here bins represent the range on x axis and weight represent the values.

If we give plt.hist([5,15,25,35,15, 55], bins=[0,10,20,30,40,50, 60],/
weights=[20,10,45,33,6,8], edgecolor="red")
plt.show()



Here 40 to 50 range is shown blank as value 15 is given twice

Also wrong value or position is shown for this range, So first 15 value is shown by taking weight as 10+6=16

plt.hist([1,11,21,31,41, 51], bins=[0,10,20,30,40,50, 60], weights=[10,1,0,33,6,8], facecolor='y', edgecolor="red") # facecolor yellow shows the histogram in yellow color.

plt.title("Histogram Heading") # This name will appear at the top of Histogram

plt.savefig(“test.jpg") # This will save histogram as jpg file to be stored on python folder.

plt.xlabel('Value')

plt.ylabel('Frequency') # These label will be shown on both x and y axis.

Function Application : In Python, function is a group of related statements that perform a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

```
def greet(name):            # This is a example of an function.
   print("Hello, " + name + ". Good morning!")
```

Pandas provide some important functions namely pipe(), apply() and applymap() :groupby(), transform() 1.Table wise Function Application: pipe() 2. Row or Column Wise Function: apply() 3. Element wise Function Application: applymap() 4. Groupby() (5) transform()

1. **Pipe()** : it is used to take one command or function output as input for another function, lile power(sqrt(n),2) # here sqrt() function value will be used as input for powr() function.
   Df.add(div(power(sqrt(n),2),3),100) these functions can be written using pipe () like
   Df.pipe(sqrt,n).pipe(power,2).pipe(div, 3).pipe(add,100)

```
Example : def adder(adder1,adder2):    # it is a function
              return adder1+adder2
          def divide(adder1,adder2):   # This is another function.
              return adder1/adder2
d = {'A':[20,50], 'B':[89,87]}     and df=pd.DataFrame(d)
df1=df.pipe(adder,5).pipe(divide,2)
print (df1)      It will show first output  After Adder Call      After Divide Call
                                      A          B          A          B
                              0       25         94         12.5       47
                              1       55         92         27.5       46
```

2. **apply() Row or Column Wise Function Application:** apply() function performs the operation over either row wise or column wise data, In above DataFrame if we give command as –

```
r=df.apply(np.mean,axis=1)   # it will give output as -          0      59.5
print(r)                                                         1      73.5
r=df.apply(np.mean)          # it will calculate mean col wise as       A      80
                                                                        B      186
```

3. Element wise Function Application in python pandas: applymap() Function performs the specified operation for all the elements the DataFrame :

r=df.applymap(lambda x:x+3)   # it will apply on each elements of DataFrame and give output as

```
A        B
23       92
53       90
```

Another example is df.apply(np.meam) # it will show mean by all values of every columns like A mean, B Mean

Whereas df.applymap(np.meam) # it will show mean for every value of DataFrame like

```
A            B
23.0         92.0
53.0         90.0
```

4. group by() function : When there are similar values in particular columns, then these vales may be grouped and other aggregation function like min(),max(),count() etc may be applied on these functions : Example

```
table = {"name", ['vishal', 'anil', 'mayur', 'viraj','mahesh'], 'age',[15, 16, 15, 17,16],/
 'weight', [51, 48, 49, 51,48], 'height', [5.1, 5.2, 5.1, 5.3,5.1],/
'runsscored', [55,25, 71, 53,51]}
```

Also df=pd.DataFrame(table)

Here different functions like sum(), count(), max(), Min() etc can be applied on DataFrame like df[‘weight’].sum(), simply df.sum() will show all sums of Numerical valued column. Now we will use group by () then we will apply different aggregate function, as we have applied in MySQL, as age is same as same for two records of 15 and 16.

print(df.groupby('age').max()) # it will show output by grouping age –

```
age      name       weight        height        runscored
15       vishal     51            5.1           71
16       mahesh     48            5.2           51
```

Here vishal comes later then mayur, so it is taken in max()

df.get_group(15)   # this command will show information of age 15 like

g=df.groupby('age')

print(g.get_group(15))   # This will show

```
          name       age   weight      height         runscored
0         vishal     15    51          5.1            55
2         mayur      15    49          5.1            71
```

print(g.size()) or print(df.groupby[‘age’].size()) # it will show output()

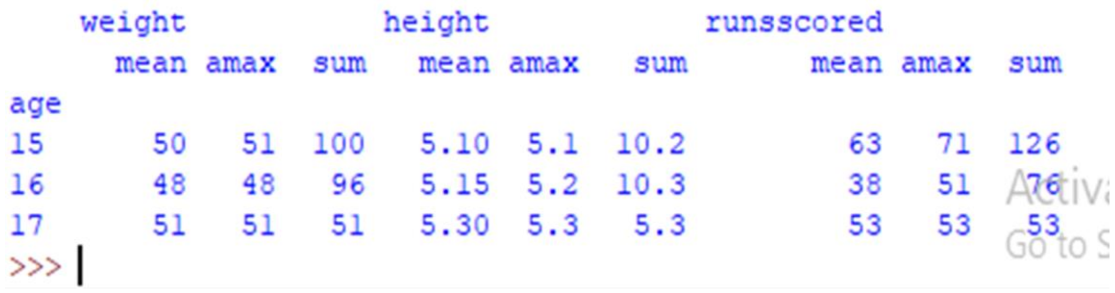
```
15   2
16   2
17   1
```

Other function in group is count(), as print(g.count())

Use of Aggregation functions by group by..

Print(g.agg([np.mean, np.max, np.sum]))

Or we can write it print(df.groupby(‘age’).agg([np.mean, np.max, np.sum])



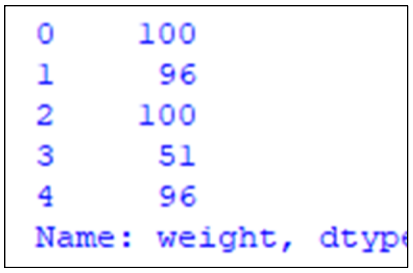
```
    weight            height              runsscored
      mean amax  sum    mean amax   sum         mean amax  sum
age
15      50   51  100    5.10  5.1  10.2           63   71  126
16      48   48   96    5.15  5.2  10.3           38   51   76
17      51   51   51    5.30  5.3   5.3           53   53   53
>>> 
```

5. **Transform –** Transform is an operation used in conjunction with groupby. It is used in given pattern, steps are -

Dataframe -> grouping -> aggregate function on each group value -> then transform that value in each group value.

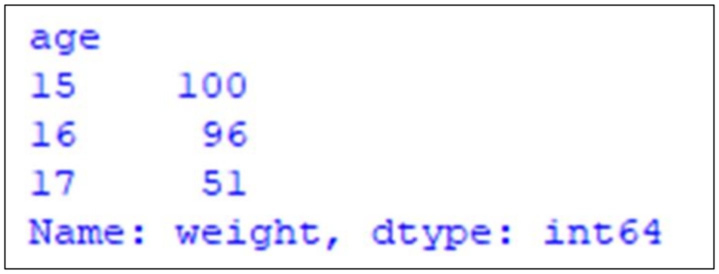
print(df.groupby('age')['weight'].transform('sum')) # **output with transform()**

print(df.groupby('age')['weight'].sum())



```
0     100
1      96
2     100
3      51
4      96
Name: weight, dtype
```

<u>Output produced without transform</u>



```
age
15      100
16       96
17       51
Name: weight, dtype: int64
```

This shows that transform() function Shows all rows information but Applying specified function on each Identical value, whereas without transform() it combines the groped column values.

Unsolved Question of Sumita Arora

Type (B) Ans1. (a) wdf.min()   (b) wdf.max(axis=1)   (c) wdf.var['Rainfall'].var()

(d)   wdf.loc[11: ,].mean()

Ans 2. (a) making pivot_table, taking mean for common values.

(b) it will group by key1 and key2 columns and to display their common values sum in the columns Col A and ColB

(c) Display given rows and columns range information by using loc command

Ans 3. wdf.pipe(sqrt,n).pipe(power,3).pipe(multiply, 10)

Ans4. wdf.apply(np.sqrt)   Ans 5. df.applymap(np.sqrt)

Ans6. As sqrare root of any values takes only one argument, as apply can combine more values also can apply aggregate functions on its value but applymap() takes only one element or value at a time, so the result are same.

Ans7. wdf.reindex(['Row1','Row2','Row3'])

Ans 8. Wdf['D']=[1,2,3,4,5]   # it will add new D column with specified values, it values not specified then it will take Nan values.

Ans9 wdf.reindex([0,1…20,d],fillvalue=10.0)   # as there are 0 to 20 indexes then it will create new index d with 10.0 values for each column.

Ans10.Yes, group by condition can be applied to any column which contains similar values.

Ans11. Import matplotlib.pyplot as plt   plt.hist(ndf)

Note : If any doubt in understanding concept immediately consult the teacher.