PANDAS

Pandas stand for Panel Data System

Pandas is an open source library for data analysis, Data manipulation and Data Visualization.

(OR) Pandas provide powerful data structures for data analysis, time series and statistics.

Pandas works on the top numpy and matplotlib.

Features of pandas

1. Handling huge amount data
2. Missing Data
3. Cleaning up data
4. Alignment and indexing
5. Merging and joining
6. Grouping and Visualizing data
7. Time Series Functionality
8. Allows to load data from multiple file formats
9. Input and Output Tools

Pandas library is used by scikit-learn for ML

Applications of Pandas

1. Recommendation Systems
2. Stock Prediction
3. Big Data and Data Science
4. NLP (Natural Language Processing)
5. Statistics and Analytics
6. Neuroscience

Important data structures of Pandas are,

1. Series
2. DataFrame

Q: What is data analysis?

Data analysis is process of collecting, transforming, cleaning and modeling the data with goal of discovering required information.

Data analysis process consists of the following steps.

1. Data Requirement Specifications
2. Data Collection
3. Data Processing
4. Data Cleaning

5. Data Analysis
6. Communication

What is Series?
Pandas series is a one dimensional array object, this object can hold data of any type. It can be integers, floats, string or python objects.
Pandas series represents or equal to a column in any data base (MsExcel, Oracle, MySQL, SQLServer,..)
What is DataFrame?
DataFrame is a two dimensional array object or data structure. Data stored tabular format, which is rows and columns.
The Dataframe consist of 3 components.

1. Data
2. Rows
3. Columns

How to install pandas?
Other than jupyter and googlecolab, it is required to install pandas lib.

pip install pandas

What is Colab?
Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

Zero configuration required
Free access to GPUs
Easy sharing
Whether you're a student, a data scientist or an AI researcher, Colab can make your work easier.

https://colab.research.google.com/?utm_source=scs-index

Pandas Series
Series is single dimension array like object with homogeneous or heterogeneous data.
Series object can be created in different ways.

1. Using array
2. Using Dictionary
3. Using Scalar values
4. Using other iterables

Series is name of the class or type which is used to construct Series object.

Syntax: Series(data,index,dtype)

Data : the source using which series object is created
Index : index values must hashable and must be unique
dtype: type of the series is defined using dtype.

Creating Empty Series

```
import pandas as pd
import numpy as np
s1=pd.Series(dtype=np.int8)
print(s1)
```

```
Series([], dtype: int8)
```

Creating Series using List object

```
s2=pd.Series([10,20,30,40,50])
print(s2)
s3=pd.Series([10,20,30,40,50],index=['a','b','c','d','e'])
print(s3)
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
a    10
b    20
c    30
d    40
e    50
dtype: int64
```

Creating Series using ndarray

```
a=np.ndarray(shape=(5,))
i=0
for value in range(10,60,10):
    a[i]=value
    i+=1
print(a)
print(type(a))
s=pd.Series(a)
print(s)
```

```
[10. 20. 30. 40. 50.]
<class 'numpy.ndarray'>
0    10.0
1    20.0
2    30.0
3    40.0
4    50.0
dtype: float64
```

Creating Series Using Dictionary

We can create series using dictionary (OR) we can pass the dictionary object to series.

Series object is using dictionary values as data and dictionary keys as index labels.

```
sales_dict={2018:50000,2019:60000,2020:75000}
s=pd.Series(sales_dict)
print(s)
emp_dict={'naresh':5000,'suresh':6000,'kishore':9000}
s=pd.Series(emp_dict)
print(s)
```

```
2018    50000
2019    60000
2020    75000
dtype: int64
naresh     5000
suresh     6000
kishore    9000
dtype: int64
```

Creating Series using Scalar values

If the series is created using scalar values we must define index. This index defines the length of series.

```
s=pd.Series(15,index=[0,1,2,3,4])
print(s)
```

```
0    15
1    15
2    15
3    15
4    15
dtype: int64
```

Accessing Data from Series

Series is index based collection, we can read and manipulate data using index.

This index starts with 0.

```
s1=pd.Series([100,200,300,400,500])
print(s1)
print(s1[0],s1[1],s1[2],s1[3],s1[4])
s2=pd.Series([1000,2000,3000,4000,5000],index=['a','b','c','d','e'])
print(s2['a'],s2['b'],s2['c'],s2['d'],s2['e'])
print(s2[0],s2[1],s2[2],s2[3],s2[4])
```

```
0    100
1    200
2    300
3    400
4    500
dtype: int64
100 200 300 400 500
1000 2000 3000 4000 5000
1000 2000 3000 4000 5000
```

Reading multiple elements/values from series

Series allows reading multiple elements by defining index labels within list.

```
s1=pd.Series(range(100,1000,100))
print(s1)
print(s1[[0,3,6,8]])
s2=pd.Series([100,200,300,400,500],index=['a','b','c','d','e'])
print(s2)
print(s2[['a','c','e']])
```

```
0    100
1    200
2    300
3    400
4    500
5    600
6    700
7    800
8    900
dtype: int64
0    100
3    400
6    700
8    900
dtype: int64
a    100
b    200
c    300
d    400
e    500
```

✓ 0s completed at 7:01 PM

Series allows slicing, to read multiple elements/values.

```
s1=pd.Series(range(100,1000,100))
print(s1)
print(s1[:3])
print(s1[-3:])
print(s1[-1::-1])
```

```
0    100
1    200
2    300
3    400
4    500
5    600
6    700
7    800
8    900
dtype: int64
0    100
1    200
2    300
dtype: int64
6    700
7    800
8    900
dtype: int64
8    900
7    800
6    700
5    600
```

DataFrame

DataFrame is two dimensional array object with heterogeneous data. In DataFrame data is stored in the form of rows and columns.

How to create DataFrame?
DataFrame can be created in different ways.

1. Series
2. Lists
3. Dictionary
4. Numpy array
5. From another dataframe
6. Data can read from files or database

"DataFrame" is type or class name, to create dataframe object
Syntax:
DataFrame(data,index,columns,dtype)

data : data is taken from various sources
Index : row labels
columns : columns labels
dtype: data type of each column

Creating empty dataframe

```
import pandas as pd
#creating empty dataframe
df=pd.DataFrame()
print(df)
```

```
Empty DataFrame
Columns: []
Index: []
```

Creating DataFrame using dictionary
Dictionary consist of key and values.
Dictionary keys as columns headers and values are columns values

```
d={'empno':[1,2,3,4,5],'ename':['naresh','suresh','rajesh','kishore','raman'],'sal':[5000,6000,7000,9000,6000]}
df=pd.DataFrame(d)
print(df)
```

```
   empno    ename   sal
0      1   naresh  5000
1      2   suresh  6000
2      3   rajesh  7000
3      4  kishore  9000
4      5    raman  6000
```

Create DataFrame using List
A nested list represents the content of dataframe.
Each list within list is represented as row.

```
person_list=[['naresh',50],['suresh',45],['kishore',35]]
df=pd.DataFrame(person_list,columns=['name','age'],dtype=float)
print(df)
```

```
      name   age
0   naresh  50.0
1   suresh  45.0
2  kishore  35.0
```

DataFrame created with missing data

Missing data is identified with NaN(Not a Number)

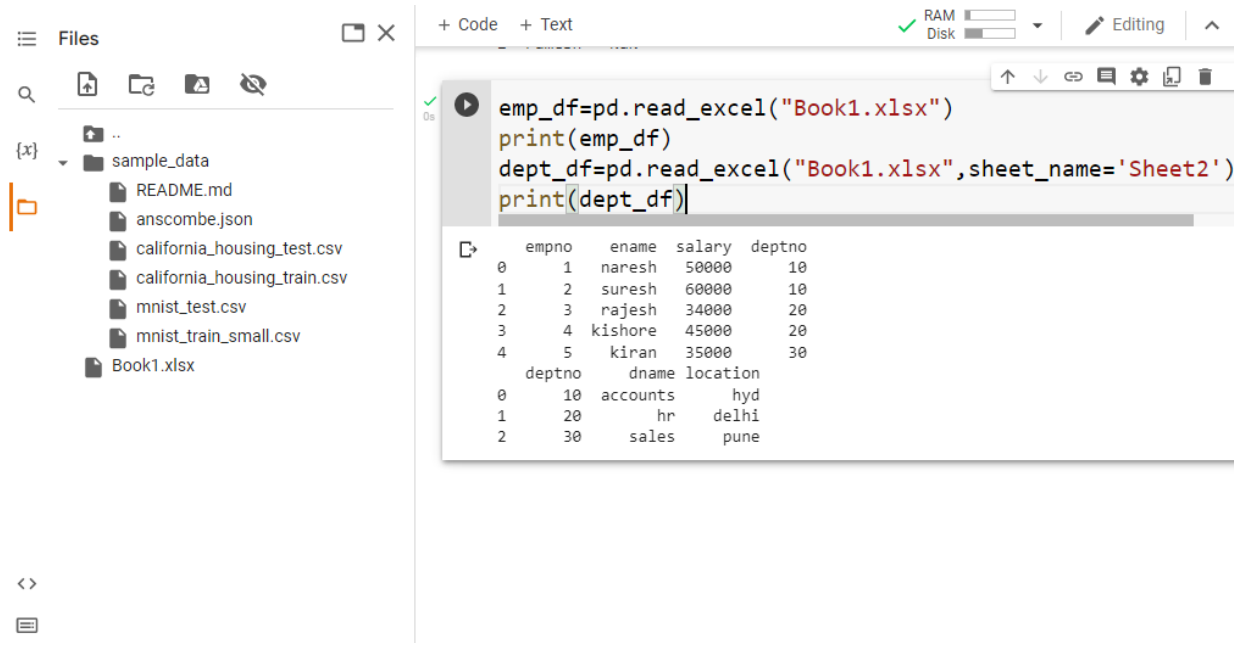
```
data=[['naresh',45],['suresh',56],['kishore',65],['rajesh']]
df=pd.DataFrame(data,columns=['name','age'])
print(df)
```

```
      name   age
0   naresh  45.0
1   suresh  56.0
2  kishore  65.0
3   rajesh   NaN
```

```
data=[{'name':'naresh','age':45},{'name':'kishore'},{'name':'suresh'},{'age':50},{}]
df=pd.DataFrame(data,index=['p1','p2','p3','p4','p5'])
print(df)
```

```
      name   age
p1   naresh  45.0
p2  kishore   NaN
p3   suresh   NaN
p4      NaN  50.0
p5      NaN   NaN
```

Reading/loading data from ms-excel



Selecting Data

1. Row Selection
2. Column Selection

Column Selection

Selecting columns from DataFrame can be done using column header.

```
data=[{'name':'naresh','age':45},{'name':'kishore'},{'name':'suresh'},{'age':50},{}]
df=pd.DataFrame(data)
print(df)
c1=df['name']
c2=df['age']
print(type(c1),type(c2))
print(c1,c2)
```

```
      name   age
0   naresh  45.0
1  kishore   NaN
2   suresh   NaN
3      NaN  50.0
4      NaN   NaN
<class 'pandas.core.series.Series'> <class 'pandas.core.series.Series'>
0     naresh
1    kishore
2     suresh
3        NaN
4        NaN
Name: name, dtype: object 0    45.0
1     NaN
2     NaN
3    50.0
4     NaN
Name: age, dtype: float64
```

Reading multiple columns from DataFrame

In order to read multiple columns, the column names must be defined as a list. It return multiple columns as a dataframe.

single column it read as a series.

```
data={'a':[1,2,3,4,5],'b':[100,200,300,400,500],'c':[1000,2000,3000,4000,5000],'d':[10000,20000,30000,40000,5000]}
df=pd.DataFrame(data)
print(df)
print(df[['a','c']])
r=df[['a','c']]
print(r)
print(type(r))
```

```
   a    b     c      d
0  1  100  1000  10000
1  2  200  2000  20000
2  3  300  3000  30000
3  4  400  4000  40000
4  5  500  5000   5000
   a     c
0  1  1000
1  2  2000
2  3  3000
3  4  4000
4  5  5000
   a     c
0  1  1000
1  2  2000
2  3  3000
3  4  4000
4  5  5000
<class 'pandas.core.frame.DataFrame'>
```

Column Addition

Adding new column to the existing DataFrame.

```
data={'col1':pd.Series([1,2,3]),
      'col2':pd.Series([10,20,30])}
df=pd.DataFrame(data)
print(df)
df['col3']=pd.Series([100,200,300])
print(df)
df['col4']=df['col2']+df['col3']
print(df)
```

```
   col1  col2
0     1    10
1     2    20
2     3    30
   col1  col2  col3
0     1    10   100
1     2    20   200
2     3    30   300
   col1  col2  col3  col4
0     1    10   100   110
1     2    20   200   220
2     3    30   300   330
```

Column Deletion

The column deletion is done using del keyword.
It allows deleting one or more than one columns.

The column is deleted with column name or column labels.

```
import pandas as pd
l=[['naresh',45],['suresh',50],['ramesh',60]]
df=pd.DataFrame(l,columns=['name','age'])
print(df)
del df['name']
print(df)
```

```
     name  age
0  naresh   45
1  suresh   50
2  ramesh   60
   age
0   45
1   50
2   60
```

Row Selection, Addition and Deletion

Each row is identified with index or label. We can read rows from dataframe using index or label.

DataFrame provide two methods to perform this operation.

1. loc
2. iloc

loc() is used to read the rows using label

iloc() is used to read the rows using index

```
student_data={'rno':[1,2,3,4,5],
              'name':['naresh','suresh','ramesh','rajesh','kiran']}
df=pd.DataFrame(student_data,index=['s1','s2','s3','s4','s5'])
print(df)
print(df.loc['s1'])
row=df.loc['s1']
print(row)
print(type(row))
print(row[0],row[1])
```

```
    rno    name
s1    1  naresh
s2    2  suresh
s3    3  ramesh
s4    4  rajesh
s5    5   kiran
rno          1
name    naresh
Name: s1, dtype: object
rno          1
name    naresh
Name: s1, dtype: object
<class 'pandas.core.series.Series'>
1 naresh
```

```
student_data={'rno':[1,2,3,4,5],
              'name':['naresh','suresh','ramesh','rajesh','kiran']}
df=pd.DataFrame(student_data,index=['s1','s2','s3','s4','s5'])
print(df)
print(df.iloc[0])
print(df.iloc[1])
```

```
    rno    name
s1    1  naresh
s2    2  suresh
s3    3  ramesh
s4    4  rajesh
s5    5   kiran
rno          1
name    naresh
Name: s1, dtype: object
rno          2
name    suresh
Name: s2, dtype: object
```

Slicing is used to read more than one row

```
student_data={'rno':[1,2,3,4,5],
              'name':['naresh','suresh','ramesh','rajesh','kiran']}
df=pd.DataFrame(student_data,index=['s1','s2','s3','s4','s5'])
print(df)
print(df[0:3])
print(df[0::2])
df1=df[0:3]
print(type(df1))
```

```
    rno    name
s1    1  naresh
s2    2  suresh
s3    3  ramesh
s4    4  rajesh
s5    5   kiran
    rno    name
s1    1  naresh
s2    2  suresh
s3    3  ramesh
    rno    name
s1    1  naresh
s3    3  ramesh
s5    5   kiran
<class 'pandas.core.frame.DataFrame'>
```

Append Row

After creating data frame we can add a new row using append method. This method will add row at the end of dataframe.

dataframe.append(row)

Row is represented as a dataframe.

```
l=[['naresh',45],['suresh',50],['ramesh',60]]
df=pd.DataFrame(l,columns=['name','age'])
df1=pd.DataFrame([['rajesh',60],['kishore',60]],columns=['name','age'])
df2=df.append(df1)
print(df2)
print(df2.iloc[0])
print(df2.iloc[3])
```

```
      name  age
0   naresh   45
1   suresh   50
2   ramesh   60
0   rajesh   60
1  kishore   60
name    naresh
age         45
Name: 0, dtype: object
name    rajesh
age         60
Name: 0, dtype: object
```

Deletion of rows

Deletion of rows are done using a method drop().
It delete only one row.
Deleting is done using row labels/index.
It row labels are duplicated it remove multiple rows.

```
l=[['naresh',45],['suresh',50],['ramesh',60]]
df=pd.DataFrame(l,columns=['name','age'])
df1=pd.DataFrame([['rajesh',60],['kishore',60]],columns=['name','age'])
df2=df.append(df1)
print(df2)
df3=df2.drop(0)
df4=df2.drop(1)
print(df3)
print(df4)
print(df2)
```

```
      name  age
0   naresh   45
1   suresh   50
2   ramesh   60
0   rajesh   60
1  kishore   60
      name  age
1   suresh   50
2   ramesh   60
1  kishore   60
     name  age
0  naresh   45
2  ramesh   60
0  rajesh   60
      name  age
```

head and tail methods of DataFrame

head and tail are the methods of DataFrame object.

head() returns first n number of rows
tail() returns last n number of rows

```
person_dict={'name':pd.Series(['naresh','ramesh','kishore','ramesh']),
             'grade':pd.Series([45,67,88,34])}
df=pd.DataFrame(person_dict)
print(df)
df1=df.head(2)
df2=df.tail(2)
print(df1)
print(df2)
```

```
      name  grade
0   naresh     45
1   ramesh     67
2  kishore     88
3   ramesh     34
     name  grade
0  naresh     45
1  ramesh     67
      name  grade
2  kishore     88
3   ramesh     34
```

Other Operations of DataFrame

sum(): This function return sum

```
import pandas as pd
df=pd.DataFrame({'sales':[10000,2000,3000,4000,5000,60000]})
print(df)
s=df.sum()
print("Total is",s)
```

```
   sales
0  10000
1   2000
2   3000
3   4000
4   5000
5  60000
Total is sales    84000
dtype: int64
```

```
import pandas as pd
df=pd.DataFrame({'sales':[10000,2000,3000,4000,5000,60000]})
print(df)
s=df.sum()
print("Total is",s)
df=pd.DataFrame({'name':['naresh','suresh','rajesh'],'age':[45,40,35]},columns=['name','age'])
print(df)
s=df.sum()
print(s)
```

```
   sales
0  10000
1   2000
2   3000
3   4000
4   5000
5  60000
Total is sales    84000
dtype: int64
     name  age
0  naresh   45
1  suresh   40
2  rajesh   35
name    nareshsureshrajesh
age                    120
dtype: object
```

describe(): This function perform statistical operations on dataframe.

```
df=pd.DataFrame({'sales':[1000,2000,3000,4000,5000,6000,7000]})
print(df)
print(df.describe())
x=df.describe()
print(type(x))
print(x.iloc[0])
print(x.loc['mean'])
```

```
   sales
0   1000
1   2000
2   3000
3   4000
4   5000
5   6000
6   7000
             sales
count     7.000000
mean   4000.000000
std    2160.246899
min    1000.000000
25%    2500.000000
50%    4000.000000
75%    5500.000000
max    7000.000000
<class 'pandas.core.frame.DataFrame'>
sales    7.0
```

```
def total(a,b):
  return a+b
df=pd.DataFrame({'col1':[10,20,30,40,50],'col2':[100,200,300,400,500]})
print(df)
df.pipe(total,10)
```

```
   col1  col2
0    10   100
1    20   200
2    30   300
3    40   400
4    50   500
```

	col1	col2
0	20	110
1	30	210
2	40	310
3	50	410
4	60	510

DataFrame.apply(func, axis=0)
Apply a function along an axis of the DataFrame.
function to apply to each column or row.

```
import numpy as np
df=pd.DataFrame({'col1':[1,2,3,4,5],'col2':[10,20,30,40,50]})
print(df)
print(df.apply(np.sqrt))
print(df.apply(np.sum,axis=0))
print(df.apply(np.sum,axis=1))
```

```
   col1  col2
0     1    10
1     2    20
2     3    30
3     4    40
4     5    50
       col1      col2
0  1.000000  3.162278
1  1.414214  4.472136
2  1.732051  5.477226
3  2.000000  6.324555
4  2.236068  7.071068
col1     15
col2    150
dtype: int64
0    11
1    22
2    33
3    44
```

DataFrame.applymap(func)
Apply a function to a Dataframe elementwise.
This method applies a function that accepts and returns a scalar to every element of a DataFrame.

```
df=pd.DataFrame({'c1':[1,2,3],'c2':[4,5,6]})
print(df)
print(df.applymap(str))
df=pd.DataFrame({'c1':['aaa','bbb','ccc']})
print(df)
print(df.applymap(str.upper))
```

```
   c1  c2
0   1   4
1   2   5
2   3   6
  c1 c2
0  1  4
1  2  5
2  3  6
    c1
0  aaa
1  bbb
2  ccc
    c1
0  AAA
1  BBB
2  CCC
```

DataFrame.fillna(value=None)

Fill NA/NaN values using the specified method.

```
import pandas as pd
list1=[[1,2,3],[4,5,6],[],[1,2]]
df=pd.DataFrame(list1)
print(df)
df1=df.fillna(0)
print(df1)
df2=df.fillna(1)
print(df2)
```

```
     0    1    2
0  1.0  2.0  3.0
1  4.0  5.0  6.0
2  NaN  NaN  NaN
3  1.0  2.0  NaN
     0    1    2
0  1.0  2.0  3.0
1  4.0  5.0  6.0
2  0.0  0.0  0.0
3  1.0  2.0  0.0
     0    1    2
0  1.0  2.0  3.0
1  4.0  5.0  6.0
2  1.0  1.0  1.0
3  1.0  2.0  1.0
```

Pandas groupby example

Start by importing pandas, numpy and creating a data frame.

```
import pandas as pd
```

Our data frame contains simple tabular data:

City	Name
Sydney	Alice
Sydney	Ada
Paris	Mallory
Sydney	Mallory
Sydney	Billy
Paris	Mallory

In code the same table is:

```
import pandas as pd
import numpy as np

df1 = pd.DataFrame( {
    "Name" : ["Alice", "Ada", "Mallory", "Mallory", "Billy" , "Mallory"] ,
    "City" : ["Sydney", "Sydney", "Paris", "Sydney", "Sydney", "Paris"]} )
```

You can then summarize the data using the groupby method. In our example there are two columns: Name and City.

The function .groupby() takes a column as parameter, the column you want to group on.
Then define the column(s) on which you want to do the aggregation.

```
print df1.groupby(["City"])[['Name']].count()
```

This will count the frequency of each city and return a new data frame:

City	Name
Sydney	2
Paris	4

The total code being:

```
import pandas as pd
import numpy as np

df1 = pd.DataFrame( {
    "Name" : ["Alice", "Ada", "Mallory", "Mallory", "Billy" , "Mallory"] ,
    "City" : ["Sydney", "Sydney", "Paris", "Sydney", "Sydney", "Paris"]} )
```

```
df2 = df1.groupby(["City"])[['Name']].count()
print(df2)
```

Example 2

The **groupby()** operation can be applied to any pandas data frame.
Lets do some quick examples.

The data frame below defines a list of animals and their speed measurements.

```
>>> df = pd.DataFrame({'Animal': ['Elephant','Cat','Cat','Horse','Horse','Cheetah', 'Cheet
[20,30,27,50,45,70,66]})
>>> df
     Animal      Speed
0  Elephant         20
1       Cat         30
2     Horse         50
3   Cheetah         70
>>>
```

You can group by animal and the average speed.

```
>>> df.groupby(['Animal']).mean()
          Speed
Animal
Cat        28.5
Cheetah    68.0
Elephant   20.0
Horse      47.5
>>>
```

If you have multiple columns in your table like so:

```
>>> df = pd.DataFrame({'Animal': ['Elephant','Cat','Cat','Horse','Horse','Cheetah',
'Cheetah'],
... 'Speed': [20,30,26,50,45,70,60],
... 'Length': [8,0.5,0.6,2,2.1,1.8,1.7]})
```

Then you can add the column like this:

```
>>> df.groupby("Animal")["Speed"].mean()
>>> df.groupby("Animal")["Length"].mean()
```

Pandas groupby() Syntax

Below is the syntax of the groupby() function, this function takes several params that are explained below and returns DataFrameGroupBy object that contains information about the groups.

```
# Syntax of DataFrame.groupby()
DataFrame.groupby(by=None, axis=0, level=None, as_index=True,
    sort=True, group_keys=True, squeeze=<no_default>,
    observed=False, dropna=True)
```

by – List of column names to group by

axis – Default to 0. It takes *0 or 'index', 1 or 'columns'*

level – Used with MultiIndex.

as_index – sql style grouped otput.

sort – Default to True. Specify whether to sort after group

group_keys – add group keys or not

squeeze – depricated in new versions

observed – This only applies if any of the groupers are Categoricals.

dropna – Default to False. Use True to drop None/Nan on sory keys

In order to explain several examples of how to perform group by, first, let's create a simple DataFrame with the combination of string and numeric columns.

```
import pandas as pd
technologies   = ({
'Courses':["Spark","PySpark","Hadoop","Python","Pandas","Hadoop","Spark","Python","NA"],
   'Fee' :[22000,25000,23000,24000,26000,25000,25000,22000,1500],
'Duration':['30days','50days','55days','40days','60days','35days','30days','50days','40days'],
   'Discount':[1000,2300,1000,1200,2500,None,1400,1600,0]
        })
df = pd.DataFrame(technologies)
print(df)
```

Yields below output.

```
   Courses    Fee Duration  Discount
0    Spark  22000   30days    1000.0
1  PySpark  25000   50days    2300.0
2   Hadoop  23000   55days    1000.0
3   Python  24000   40days    1200.0
4   Pandas  26000   60days    2500.0
5   Hadoop  25000   35days       NaN
6    Spark  25000   30days    1400.0
7   Python  22000   50days    1600.0
```

8 NA 1500 40days 0.0

2. Pandas groupby() Example

As I said above `groupby()` function returns `DataFrameGroupBy` object after collecting the identical data into groups from pandas DataFrame. This object contains several methods (`sum()`, `mean()` e.t.c) that can be used to aggregate the grouped rows.

```
# Use groupby() to compute the sum
df2 =df.groupby(['Courses']).sum()
print(df2)
```

Yields below output.

```
          Fee  Discount
Courses
Hadoop   48000    1000.0
NA        1500       0.0
Pandas   26000    2500.0
PySpark  25000    2300.0
Python   46000    2800.0
Spark    47000    2400.0
```

3. Pandas groupby() on Two or More Columns

Most of the time we would need to perform groupby on multiple columns of DataFrame, you can do this by passing a list of column labels you wanted to perform group by on.

```
# Group by multiple columns
df2 =df.groupby(['Courses', 'Duration']).sum()
print(df2)
```

Yields below output

```
                    Fee  Discount
Courses Duration
Hadoop  35days    25000       0.0
        55days    23000    1000.0
NA      40days     1500       0.0
Pandas  60days    26000    2500.0
PySpark 50days    25000    2300.0
Python  40days    24000    1200.0
        50days    22000    1600.0
Spark   30days    47000    2400.0
```

4. Add Index to the grouped data

By default `groupby()` result doesn't include row Index, you can add the index using DataFrame.reset_index() method.

```
# Add Row Index to the group by result
df2 = df.groupby(['Courses','Duration']).sum().reset_index()
print(df2)
```

Yields below output

```
   Courses Duration    Fee  Discount
0   Hadoop   35days  25000       0.0
1   Hadoop   55days  23000    1000.0
2       NA   40days   1500       0.0
3   Pandas   60days  26000    2500.0
4  PySpark   50days  25000    2300.0
5   Python   40days  24000    1200.0
6   Python   50days  22000    1600.0
7    Spark   30days  47000    2400.0
```

5. Drop NA /None/Nan (on group key) from Result

You can also choose whether to include NA/None/Nan in group keys or not by setting `dropna` parameter. By default the value of `dropna` set to `True`. so to not to include None/Nan values on group keys
set `dropna=False` parameter.

```
# Drop rows that have None/Nan on group keys

df2=df.groupby(by=['Courses'], dropna=False).sum()

print(df2)
```

6. Sort groupby() result by Group Key

By default groupby() function sorts results by group key hence it will take additional time, if you have a performance issue and don't want to sort the group by the result, you can turn this off by using the `sort=False` param.

```
# Remove sorting on grouped results

df2=df.groupby(by=['Courses'], sort=False).sum()

print(df2)
```

If you wanted to sort key descending order, use below.

```
# Sorting group keys on descending order
```

```
groupedDF = df.groupby('Courses',sort=False).sum()
sortedDF=groupedDF.sort_values('Courses', ascending=False)
print(sortedDF)
```

In case you wanted to sort by a different key, you can do so by using DataFrame.apply() function.

```
# Using apply() & lambda
df.groupby('Courses').apply(lambda x: x.sort_values('Fee'))
```

7. Apply More Aggregations

You can also compute several aggregations at the same time in pandas by passing the list of agg functions to the `aggregate()`.

```
# Groupby & multiple aggregations
result = df.groupby('Courses')['Fee'].aggregate(['min','max'])
print(result)
```

Yields below output.

```
        min    max
```

Courses

Hadoop 23000 25000

NA 1500 1500

Pandas 26000 26000

PySpark 25000 25000

Python 22000 24000

Spark 22000 25000

The above example calculates `min` and `max` on the `Fee` column. Let's extend this to compute different aggregations on different columns.