Pandas

pandas DataFrame

```
import numpy as np
import pandas as pd
# importing NumPy & pandas
```

```
student_values = [["Rayan", 53, "DBDA"], ["Nelly", 56, "DAC"], ["Eminem", 57, "DITISS"]]
# creating a 2-D list for DataFrame
df = pd.DataFrame(student_values, columns=['Name', 'RollNo', 'Course'])
# using 2-D DataFrame to create DataFrame with columns 'Name', 'RollNo', 'Course'
df
```

	Name	RollNo	Course
0	Rayan	53	DBDA
1	Nelly	56	DAC
2	Eminem	57	DITISS

```
df = pd.DataFrame([['Sudeep', 3545, 'MBA'], ['Ruchika', 1254, 'MCA'], ['Rahul', 6547, 'MS'], ['Purnima', 1389, 'BE']], columns=['Name', 'RollNo', 'Qualification'])
# Creating 2nd DataFrame with columns 'Name', 'RollNo', 'Qualification'
df
```

	Name	RollNo	Qualification
0	Sudeep	3545	MBA
1	Ruchika	1254	MCA
2	Rahul	6547	MS
3	Purnima	1389	BE

DataFrame.iloc[Integer Locator]

```
mydf = df.iloc[[0, 2]]
# df.iloc(integer location/index or array of indexes or slice of indices)
# here, internal bracket is list/array of indices
# creating a new DataFrame mydf with rows 0 & 2 from previous DataFrame df
mydf
```

```
Name  RollNo  Qualification
```

```
mydf.iloc[1]
# printing 1st row from mydf DataFrame using iloc property
```

```
Name             Rahul
RollNo            6547
Qualification       MS
Name: 2, dtype: object
```

```
import os
# iporting os package
os.getcwd()
# checking current working directory
```

```
'/content'
```

```
from google.colab import files
uploaded=files.upload()
# ERPData.xlsx
# to be used with google colab

# os.chdir(r'D:/advanced-analytics-files/day03')
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving ERPData.xlsx to ERPData.xlsx

pd.read_excel('filename.xlsx')

- to open a WorkBook as a DataFrame

```
df = pd.read_excel('ERPData.xlsx')
# reading excel file to DataFrame
df.head()
# returns first 5 records from DataFrame
```

	MaterialID	Location	Quantity
0	TMI-43T	MWH-4	34
1	AXCP-78	MWH-1	67
2	LXCV-21	MWH-2	27
3	AXCP-78	MWH-5	65
4	AXCP-78	MWH-4	36

DataFrame.Shape

```
df.shape
# prints the shape in (r, c) format for DataFrame
```

```
(50, 3)
```

DataFrame['columnName']

```
df['Quantity']
# method to fetch one column from dataFrame as Series Type
# df.Quantity   # alt method to fetch one column
```

```
0      34
1      67
2      27
3      65
4      36
5      78
6      31
7      29
8      10
9     120
10     34
11     58
12     76
13     32
14     65
15     87
16     12
17    102
18     34
19     52
20     39
21     75
22     48
23     71
24    152
25    109
26     57
27     83
28     34
29     57
30     27
31     65
32     43
33    112
34     26
35     31
36     28
37     65
38     70
39    145
40     34
41     57
42     31
43     38
44     42
45     69
```

```
46     85
47     29
48     75
49     39
Name: Quantity, dtype: int64
```

```
x = list(df['Quantity'])
# df['Quantity'] returns a series type which is now converted into list
```

```
x[:5]
# slicing the list
```

```
[34, 67, 27, 65, 36]
```

```
len(x)
# finding length of list
```

```
50
```

Q. find the count of records with quantity >=75

```
arr1 = np.array(x)
# converting list of quantities to ndarray
idx = np.where(arr1 >= 75)
# using where conditon with ndarray to find quantities  >= 75
len(idx[0])
# finding length of index array

# alt method
# len(np.where(np.array(x)>=75)[0])
```

```
13
```

Q. find the count of records with quantity >=75

```
(arr1>=75).sum()
# alt method
```

```
13
```

Q. find count of records where location is MWH-3

```
arr1 = np.array(df.Location)
# creating an ndarray with Location column from DataFrame
(arr1=='MWH-3').sum()
# using condition on ndarray and then sum() operator to count records
```

```
4
```

```
# alt method
(df.Location == 'MWH-3').sum()
```

```
4
```

```
idx
# printing the indices that satisfy the condition
```

```
(array([ 5,  9, 12, 15, 17, 21, 24, 25, 27, 33, 39, 46, 48]),)
```

```
mydf = df.iloc[idx]
# using indices in iloc to print the records that satisfy the condition
mydf
# note that the indices are improper and according to the previous DataFrame
```

	MaterialID	Location	Quantity
5	TMI-43T	MWH-4	78
9	TMI-43T	MWH-1	120
12	TMI-43T	MWH-4	76
15	GCVB-79	MWH-2	87
17	SDRT-67	MWH-5	102
21	TMI-43T	MWH-4	75
24	AXCP-78	MWH-1	152
25	AXCP-78	MWH-1	109
27	LXCV-21	MWH-5	83
33	TMI-43T	MWH-5	112
39	AXCP-78	MWH-1	145
46	AXCP-78	MWH-2	85
48	AXCP-78	MWH-1	75

```
mydf.iloc[[0, 1, 2]]
# accessing mydf DataFrame using default indices,
# but it prints indices from previous indices
```

	MaterialID	Location	Quantity
5	TMI-43T	MWH-4	78
9	TMI-43T	MWH-1	120
12	TMI-43T	MWH-4	76

▾ Q. find records with quantity > 100 and location == MWH-5

```
ind = np.where((df['Quantity'] >= 100) & (df['Location'] == 'MWH-5'))
# using two conditions in np.where() using logical '&' operator
ind
# printing the array of indices satisfying the condition
```

```
(array([17, 33]),)
```

```
mydf = df.iloc[ind]
# using indices in iloc to get values from previous DataFrame
mydf
# here also, it shows indices as per previous DataFrame
```

	MaterialID	Location	Quantity
17	SDRT-67	MWH-5	102
33	TMI-43T	MWH-5	112

Q. records where either material id is tmi43t or the quantity >=100

```
ind = np.where((df['MaterialID'] == 'TMI-43T') | (df['Quantity'] >= 100))
# using two conditions in np.where() using logical '|' operator
ind
# printing the array of indices satisfying the condition
```

```
(array([ 0,  5,  6,  7,  9, 12, 13, 14, 17, 18, 19, 20, 21, 24, 25, 31, 32,
        33, 39]),)
```

```
len(ind[0])
# getting the count of records where condition satisfied
```

```
19
```

```
mydf = df.iloc[ind]
# using indices in iloc to get values from previous DataFrame
mydf
# here also, it shows indices as per previous DataFrame
```

	MaterialID	Location	Quantity
0	TMI-43T	MWH-4	34
5	TMI-43T	MWH-4	78
6	TMI-43T	MWH-4	31
7	TMI-43T	MWH-2	29
9	TMI-43T	MWH-1	120
12	TMI-43T	MWH-4	76
13	TMI-43T	MWH-4	32
14	TMI-43T	MWH-3	65
17	SDRT-67	MWH-5	102
18	TMI-43T	MWH-4	34
19	TMI-43T	MWH-4	52

Q. find the count of records with quantity <=25

```
ind = np.where(df['Quantity']<= 25)
# using conditions in np.where() over 'Quantity' column from DataFrame df
ind
```

```
(array([ 8, 16]),)
```

```
len(ind[0])
# count of indices which satisfy the condition
```

```
2
```

DataFrame.drop(index)

```
mydf = df.drop(ind[0])
# creates a new Dataframe dropping the specifying the index
mydf.shape
# prints the shape of DataFrame
```

```
(48, 3)
```

```
df.iloc[ind]
```

	MaterialID	Location	Quantity
8	GCVB-79	MWH-2	10
16	SDRT-67	MWH-5	12

DataFrame.head()

```
df.head()
```

	MaterialID	Location	Quantity
0	TMI-43T	MWH-4	34
1	AXCP-78	MWH-1	67
2	LXCV-21	MWH-2	27
3	AXCP-78	MWH-5	65
4	AXCP-78	MWH-4	36

Q. add another column 'Status' , and put value 'Scrap' if Qty<=50 else put value 'Useful'

```
# (df.Quantity<=50)-> df['Status']=Scrap
# else df['Status']=useful

qty_lst = list(df['Quantity'])
#qty_lst
sts_lst = []
for a in qty_lst:
    if a <=50:
        sts_lst.append('Scrap')
    else:
        sts_lst.append("Useful")
# sts_lst
df['Status'] = sts_lst
df.head()
```

	MaterialID	Location	Quantity	Status
0	TMI-43T	MWH-4	34	Scrap
1	AXCP-78	MWH-1	67	Useful
2	LXCV-21	MWH-2	27	Scrap
3	AXCP-78	MWH-5	65	Useful
4	AXCP-78	MWH-4	36	Scrap

```
mydf = df[['Location', 'Status']]
# creating a new dataFrame with only two columns from previous DataFrame
mydf.head()
# printing only top 5 records from DataFrame
```

	Location	Status
0	MWH-4	Scrap
1	MWH-1	Useful
2	MWH-2	Scrap

DataFrame('ColumnName', axis=1)

```
mydf = df.drop('Location', axis=1)
# dropping a row/column based on axis
mydf.head()
# printing top 5 records from DataFrame
```

	MaterialID	Quantity	Status
0	TMI-43T	34	Scrap
1	AXCP-78	67	Useful
2	LXCV-21	27	Scrap
3	AXCP-78	65	Useful
4	AXCP-78	36	Scrap

DataFrame.columns

```
mydf.columns
# prints an index of column names
```

```
Index(['MaterialID', 'Quantity', 'Status'], dtype='object')
```

renaming column names in DataFrame

```
x = list(mydf.columns)
# converts index of headers in DataFrame
x
```

```
['MaterialID', 'Quantity', 'Status']
```

```
x[0] = 'Name'
# updating value of first column name in list for index of
# column Names in DataFrame
mydf.columns = x
# updating the column Name index with list containing updated list names
```

```
mydf.head()
# printing first five records to check if column names have been updated
```

	Name	Quantity	Status
0	TMI-43T	34	Scrap
1	AXCP-78	67	Useful
2	LXCV-21	27	Scrap
3	AXCP-78	65	Useful
4	AXCP-78	36	Scrap

Data Cleaning

```
import numpy as np
import pandas as pd
# importing NumPy & pandas
```

```
from google.colab import files
uploaded=files.upload()
# emp_info.xlsx
# to be used with google colab

# os.chdir(r'D:/advanced-analytics-files/day03')
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving emp_info.xlsx to emp_info.xlsx

```
df = pd.read_excel('emp_info.xlsx')
# loading excel file into dataframe
df
```

	Name	EmpID	Deptt	Passport
0	Sudeep	A342	Quality	DF453278
1	Ruchika	C578	Sales	CA567657
2	NaN	J436	Admin	DF453278
3	Arjun	A746	NaN	RF453432
4	Rishi	R475	Prodn	DF453278
5	Chetan	NaN	NaN	ER534867
6	Deepak	NaN	Quality	RF453432

DataFrame['ColumnName'].isnull()

```
ind = np.where(df['Deptt'].isnull())
# using np.where() with isnull() condition
ind
# printing indices which have null values
```

```
(array([3, 5]),)
```

```
# deptt is not missing / data filtered without null values
df1 = df.drop(ind[0], axis=0)
# creating a new DataFrame while dropping indices which have null values
df1
# printing new DataFrame that has removed null values
```

	Name	EmpID	Deptt	Passport
0	Sudeep	A342	Quality	DF453278
1	Ruchika	C578	Sales	CA567657
2	NaN	J436	Admin	DF453278
4	Rishi	R475	Prodn	DF453278
6	Deepak	NaN	Quality	RF453432

```
# deptt is missing / data filtered with null values
df2 = df.iloc[ind]
# creating a new DataFrame using indices which have null values
df2
# printing new DataFrame which contains null values
```

	Name	EmpID	Deptt	Passport
3	Arjun	A746	NaN	RF453432
5	Chetan	NaN	NaN	ER534867

▼ Q. select the rows where either empid is missing or deptt is missing

```
ind = np.where( (df['EmpID'].isnull()) | (df['Deptt'].isnull()) )
# creating a new DataFrame using where() specifying
# either EmpID or Deptt is null
ind
# printing indices which satisfies condition
```

```
(array([3, 5, 6]),)
```

```
# either empid is missing or deptt is missing
df2 = df.iloc[ind]
# creating a new DataFrame using indices in df.iloc which satisfies
# either EmpID or Deptt is null
df2
# printing Dataframe which satisfies condition
```

	Name	EmpID	Deptt	Passport
3	Arjun	A746	NaN	RF453432
5	Chetan	NaN	NaN	ER534867
6	Deepak	NaN	Quality	RF453432

Q. neither empid is missing nor deptt is missing

```
df1 = df.drop(ind[0], axis=0)
# dropping indices with null vlaue condition to create
# a new DataFrame without null values
df1
# printing Dataframe which satisfies condition for withour null values
```

	Name	EmpID	Deptt	Passport
0	Sudeep	A342	Quality	DF453278
1	Ruchika	C578	Sales	CA567657
2	NaN	J436	Admin	DF453278
4	Rishi	R475	Prodn	DF453278

Q. select the rows where none of the values is missing

DataFrame.dropna()

```
mydf = df.dropna()
# DataFrame.dropna() removes all the records with any null values
mydf
# printing new DataFrame without any null values
```

	Name	EmpID	Deptt	Passport
0	Sudeep	A342	Quality	DF453278
1	Ruchika	C578	Sales	CA567657
4	Rishi	R475	Prodn	DF453278

```
df
# printing original DataFrame
```

	Name	EmpID	Deptt	Passport
0	Sudeep	A342	Quality	DF453278
1	Ruchika	C578	Sales	CA567657
2	NaN	J436	Admin	DF453278
3	Arjun	A746	NaN	RF453432
4	Rishi	R475	Prodn	DF453278
5	Chetan	NaN	NaN	ER534867
6	Deepak	NaN	Quality	RF453432

Q. remove duplicate values on Passport column

DataFrame['columnName'].duplicated(keep=False)

```
ind = np.where(df['Passport'].duplicated(keep=False))
# create index of column 'Passport' from DataFrame, without duplicate values
# .duplicated(keep=False) will mark all the duplicates as True
ind
# printing indices which satisfy the condition
```

```
(array([0, 2, 3, 4, 6]),)
```

```
df1 = df.drop(ind[0], axis=0)
# unique values on Passport column
# using index of column 'Passport' from DataFrame to drop records without
#  duplicate values or unique values
df1
# printing new DataFrame which contains unique values
```

	Name	EmpID	Deptt	Passport
1	Ruchika	C578	Sales	CA567657
5	Chetan	NaN	NaN	ER534867

```
# non-unique values on Passport column
df2 = df.iloc[ind]
# creating a new DataFrame with iloc using indices with duplicate entries
df2
# printing the new DataFrame with duplicate entries
```

	Name	EmpID	Deptt	Passport
0	Sudeep	A342	Quality	DF453278
2	NaN	J436	Admin	DF453278
3	Arjun	A746	NaN	RF453432

DataFrame['columnName'].duplicated(keep='last')

```
ind = np.where(df['Passport'].duplicated(keep='last'))
ind
# keep='last' marks all duplicates, but not the last one
# considers last duplicate occurence as correct
# while other duplicate values are removed
```

```
(array([0, 2, 3]),)
```

```
keep_last_mydf = df.drop(ind[0], axis=0)
#
keep_last_mydf
#
```

	Name	EmpID	Deptt	Passport
1	Ruchika	C578	Sales	CA567657
4	Rishi	R475	Prodn	DF453278
5	Chetan	NaN	NaN	ER534867
6	Deepak	NaN	Quality	RF453432

DataFrame['columnName'].duplicated(keep='first')

```
ind = np.where(df['Passport'].duplicated(keep='first'))
ind
# keep='first' considers last duplicate occurence is considered correct
# while other duplicate values are removed
```

```
(array([2, 4, 6]),)
```

DataFrame.drop(index, axis=0)

```
keep_first_mydf = df.drop(ind[0], axis=0)
# dropping indices of duplicates keeping 'first' as original value
# but all other as duplicate values
keep_first_mydf
# printing DataFrame with duplicates as 'first' as removed
```

	Name	EmpID	Deptt	Passport
1	Ruchika	C578	Sales	CA567657
4	Rishi	R475	Prodn	DF453278
5	Chetan	NaN	NaN	ER534867
6	Deepak	NaN	Quality	RF453432

DataFrame GroupBy

```
import numpy as np
import pandas as pd
# importing NumPy & pandas
```

```
from google.colab import files
uploaded=files.upload()
# ERPData.xlsx
# to be used with google colab

# os.chdir(r'D:/advanced-analytics-files/day03')
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving ERPData.xlsx to ERPData.xlsx

```
df = pd.read_excel('ERPData.xlsx')
# using pd.read_excel('filename') to load excel file to DataFrame
df.head()
# printing first 5 records from DataFrame
```

	MaterialID	Location	Quantity
0	TMI-43T	MWH-4	34
1	AXCP-78	MWH-1	67
2	LXCV-21	MWH-2	27
3	AXCP-78	MWH-5	65
4	AXCP-78	MWH-4	36

DataFrame.groupby('ColumnName')

- groupby: Grouped information based on columns
- groups created can be used to perfrom aggregate functions

```
grp1 = df.groupby('Location')
# creates a DataFrame GroupBy on Location column
grp1
# prints object information for DataFrame GroupBy
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f0b49ad8eb0>
```

DataFrameGroupBy.groups

```
grp1.groups
# gruoups property lsts down all the groups in a DataFrame Group
```

```
{'MWH-1': [1, 9, 10, 24, 25, 26, 39, 40, 41, 48, 49], 'MWH-2': [2, 7, 8, 15, 35, 36, 42, 45, 46], 'MWH-3': [14, 43, 44, 47], 'MWH-4': [0, 4, 5, 6, 11, 12, 13, 18, 19, 20, 21, 22, 30, 31, 32],
'MWH-5': [3, 16, 17, 23, 27, 28, 29, 33, 34, 37, 38]}
```

```
len(grp1.groups)
# prints count of unique groups in the column where GroupBy is created
```

```
5
```

```
type(grp1)
# type of groupby object is DataFrameGroupBy
```

```
pandas.core.groupby.generic.DataFrameGroupBy
```

DataFrameGroupBy.get_group('UniqueValue')

```
grp1.get_group('MWH-2')
# uses DataFrameGroupBy.get_group property to fetch records
# for a specified unique value / group only
```

	MaterialID	Location	Quantity
2	LXCV-21	MWH-2	27
7	TMI-43T	MWH-2	29
8	GCVB-79	MWH-2	10
15	GCVB-79	MWH-2	87
35	GCVB-79	MWH-2	31
36	GCVB-79	MWH-2	28
42	SDRT-67	MWH-2	31
45	DDBN-89	MWH-2	69
46	AXCP-78	MWH-2	85

DataFrameGroupBy.agg(AggregateFunction)

- works with DataFrameGroupBy, but returns a DataFrame

```
grp1.agg(np.mean)
# can specify the aggregate function to be used with DataFrame GroupBy
# printing the mean of each of the groups created on location column
```

```
<ipython-input-9-1c597bc92aa3>:1: FutureWarning: The operation <function mean at 0x7f0b899
  grp1.agg(np.mean)
```

Location	Quantity
MWH-1	80.818182
MWH-2	44.111111
MWH-3	43.500000
MWH-4	48.533333
MWH-5	63.363636

```
grp1.agg(np.size)
# printing the size/length of each of the groups created on location column

# NOTE that, now it prints size for MaterialID column too,
# because it can count all type of values,
# but for numeric aggregate functions like mean, etc. ,
# it takes up column which have numeric values only
```

Location	MaterialID	Quantity
MWH-1	11	11
MWH-2	9	9
MWH-3	4	4
MWH-4	15	15
MWH-5	11	11

```
grp1.agg(np.sum)
# printing the sum of each of the groups created on location column

# Here again, it'll take coulumns which have numeric values only
# for numeric functions like sum, mean
```

```
<ipython-input-11-70f497a5407a>:1: FutureWarning: The operation <function sum at 0x7f0b899
  grp1.agg(np.sum)
```

Location	Quantity
MWH-1	889
MWH-2	397

DataFrameGroupBy.agg([ListOfAggreagateFunctions])

Location	Quantity
MWH-5	697

```
grp1.agg([np.sum, np.mean])
# printing the sum & mean of each of the groups created on location column
# but it'll work on numeric columns only
```

```
<ipython-input-12-61ce2952447e>:1: FutureWarning: ['MaterialID'] did not aggregate success
  grp1.agg([np.sum, np.mean])
```

Location	Quantity	
	sum	**mean**
MWH-1	889	80.818182
MWH-2	397	44.111111
MWH-3	174	43.500000
MWH-4	728	48.533333
MWH-5	697	63.363636

```
r = grp1.agg([np.sum, np.mean])
# returns a DatFrame from DaaFrameGroupBy using aggregate functions
r
# printing DataFrame r
```

```
<ipython-input-13-aa90d8c6701d>:1: FutureWarning: ['MaterialID'] did not aggregate success
```

```
type(r)
# checking type of DataFrame containing aggregate information
#  returned by DataFrameGroupBy.agg()
```

```
pandas.core.frame.DataFrame
```

```
ind = np.where(r.index == 'MWH-3')
# using np.where() to find the indices of records of
# DataFrame which has index = 'MWH-3'
ind
# printing indices of records satisfying the condition
```

```
(array([2]),)
```

```
r.iloc[ind]
# using iloc on DataFrame to print the records on indices
# satisfying the condition
```

	Quantity	
	sum	mean
Location		
MWH-3	174	43.5

```
grp2 = df.groupby(['MaterialID', 'Location'])
# creating another group using DataFrame.groupby() to make
# groups by 'MaterialID' column, and then sub-grouping by 'Location' column
grp2.groups
```

```
{('AXCP-78', 'MWH-1'): [1, 24, 25, 39, 48], ('AXCP-78', 'MWH-2'): [46], ('AXCP-78', 'MWH-3'): [47], ('AXCP-78', 'MWH-4'): [4], ('AXCP-78', 'MWH-5'): [3, 37, 38], ('DDBN-89', 'MWH-1'): [26, 40, 49], ('DDBN-89', 'MWH-2'): [45], ('DDBN-89', 'MWH-3'): [43], ('DDBN-89', 'MWH-4'): [22, 30], ('DDBN-89', 'MWH-5'): [23], ('GCVB-79', 'MWH-2'): [8, 15, 35, 36], ('GCVB-79', 'MWH-5'): [34], ('LXCV-21', 'MWH-2'): [2], ('LXCV-21', 'MWH-3'): [44], ('LXCV-21', 'MWH-5'): [27, 28, 29], ('SDRT-67', 'MWH-1'): [10, 41], ('SDRT-67', 'MWH-2'): [42], ('SDRT-67', 'MWH-4'): [11], ('SDRT-67', 'MWH-5'): [16, 17], ('TMI-43T', 'MWH-1'): [9], ('TMI-43T', 'MWH-2'): [7], ('TMI-43T', 'MWH-3'): [14], ('TMI-43T', 'MWH-4'): [0, 5, 6, 12, 13, 18, 19, 20, 21, 31, 32], ('TMI-43T', 'MWH-5'): [33]}
```

```
grp3 = df.groupby(['Location', 'MaterialID'])
# creating another group using DataFrame.groupby() to make
# groups by 'Location' column, and then sub-grouping by 'MaterialID' column
grp3.groups
```

```
{('MWH-1', 'AXCP-78'): [1, 24, 25, 39, 48], ('MWH-1', 'DDBN-89'): [26, 40, 49], ('MWH-1', 'SDRT-67'): [10, 41], ('MWH-1', 'TMI-43T'): [9], ('MWH-2', 'AXCP-78'): [46], ('MWH-2', 'DDBN-89'): [45], ('MWH-2', 'GCVB-79'): [8, 15, 35, 36], ('MWH-2', 'LXCV-21'): [2], ('MWH-2', 'SDRT-67'): [42], ('MWH-2', 'TMI-43T'): [7], ('MWH-3', 'AXCP-78'): [47], ('MWH-3', 'DDBN-89'): [43], ('MWH-3', 'LXCV-21'): [44], ('MWH-3', 'TMI-43T'): [14], ('MWH-4', 'AXCP-78'): [4], ('MWH-4', 'DDBN-89'): [22, 30], ('MWH-4', 'SDRT-67'): [11], ('MWH-4', 'TMI-43T'): [0, 5, 6, 12, 13, 18, 19, 20, 21, 31, 32], ('MWH-5', 'AXCP-78'): [3, 37, 38], ('MWH-5', 'DDBN-89'): [23], ('MWH-5', 'GCVB-79'): [34], ('MWH-5', 'LXCV-21'): [27, 28, 29], ('MWH-5', 'SDRT-67'): [16, 17], ('MWH-5', 'TMI-43T'): [33]}
```

```
grp2.agg([np.sum, np.size])
# using aggregate functions on DataFrameGroupBy which is grouped
# on two columns of DataFrame,
# so there will be sub-groups for groups,
# and then it'll execute aggregate functions for each of the sub-groups

# Note that, here the order of grouping is 'MaterialID' then 'Location'
```

```
# groups are arranged aplhabetically, so access time is reduced
# and DataFrame becomes optimized
```

		Quantity	
		sum	size
MaterialID	Location		
AXCP-78	MWH-1	548	5
	MWH-2	85	1
	MWH-3	29	1
	MWH-4	36	1
	MWH-5	200	3
DDBN-89	MWH-1	130	3
	MWH-2	69	1
	MWH-3	38	1
	MWH-4	75	2
	MWH-5	71	1
GCVB-79	MWH-2	156	4
	MWH-5	26	1
LXCV-21	MWH-2	27	1
	MWH-3	42	1
	MWH-5	174	3
SDRT-67	MWH-1	91	2
	MWH-2	31	1
	MWH-4	58	1
	MWH-5	114	2
TMI-43T	MWH-1	120	1
	MWH-2	29	1
	MWH-3	65	1
	MWH-4	559	11
	MWH-5	112	1

```
grp3.agg([np.sum, np.mean])
# Note that, here the order of grouping is 'Location' then 'MaterialID'
```

		Quantity	
		sum	mean
Location	MaterialID		
MWH-1	AXCP-78	548	109.600000
	DDBN-89	130	43.333333
	SDRT-67	91	45.500000
	TMI-43T	120	120.000000
MWH-2	AXCP-78	85	85.000000
	DDBN-89	69	69.000000
	GCVB-79	156	39.000000
	LXCV-21	27	27.000000
	SDRT-67	31	31.000000
	TMI-43T	29	29.000000
MWH-3	AXCP-78	29	29.000000
	DDBN-89	38	38.000000
	LXCV-21	42	42.000000
	TMI-43T	65	65.000000
MWH-4	AXCP-78	36	36.000000
	DDBN-89	75	37.500000
	SDRT-67	58	58.000000
	TMI-43T	559	50.818182
MWH-5	AXCP-78	200	66.666667
	DDBN-89	71	71.000000
	GCVB-79	26	26.000000
	LXCV-21	174	58.000000

```
1 df.head()
2 # first 5 records from original DataFrame
```

	MaterialID	Location	Quantity
0	TMI-43T	MWH-4	34
1	AXCP-78	MWH-1	67
2	LXCV-21	MWH-2	27
3	AXCP-78	MWH-5	65
4	AXCP-78	MWH-4	36

DataFrame.sort_values('columnName')

```
mydf = df.sort_values('Quantity')
# creating a new DataFrame while sorting on
# column 'Quantity' from original DataFrame
```

DataFrame.head(n)

- `DataFrame.head()` : by default prints first 5 records
- `DataFrame.head(n)` : prints first n records

```
mydf.head()
# printing first 5 records from DataFrame
```

	MaterialID	Location	Quantity
8	GCVB-79	MWH-2	10
16	SDRT-67	MWH-5	12
34	GCVB-79	MWH-5	26
2	LXCV-21	MWH-2	27
30	DDBN-89	MWH-4	27

DataFrame.tail(n)

- `DataFrame.tail()` : by default prints last 5 records
- `DataFrame.tail(n)` : prints last n records

```
mydf.tail()
# printing last 5 records from DataFrame
```

	MaterialID	Location	Quantity
25	AXCP-78	MWH-1	109
33	TMI-43T	MWH-5	112
9	TMI-43T	MWH-1	120
39	AXCP-78	MWH-1	145
24	AXCP-78	MWH-1	152

DataFrame.sort_values([listOfColumns])

```

```

```
mydf = df.sort_values(['Location', 'MaterialID', 'Quantity'])
# creating a new Dataframe with sorting original DataFrame on three columns
mydf
# printing sorted DataFrame
```

MaterialID Location Quantity

Pandas merge

- used to merge two DataFrames
- acts as SQL Join from

39 AXCP-78 MWH-1 145

pd.merge with 'Pune' & 'Mumbai'

```
import numpy as np
import pandas as pd
```

```
from google.colab import files
uploaded=files.upload()
# locations.xlsx
# to be used with google colab

# os.chdir(r'D:/advanced-analytics-files/day03')
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving locations.xlsx to locations.xlsx

pd.read_excel('filename.xlsx', sheet_name='WorkSheet')

42 SDRT-67 MWH-2 31

```
df1 = pd.read_excel('locations.xlsx', sheet_name='Pune')
# taking workSheet 'Pune' from WorkBook 'locations.xlsx' into DataFrame
df2 = pd.read_excel('locations.xlsx', sheet_name='Mumbai')
# taking workSheet 'Mumbai' from WorkBook 'locations.xlsx' into DataFrame
```

```
df1
# printing DataFrame df1
```

	Name	Subject	Grade
0	Rakesh	Python	A
1	Manoj	MLPython	C
2	Vaibhav	Statistics	B
3	Hitesh	CommSkills	A
4	Suyash	ProjectMgmt	B

0 TMI-431 MWH-4 34

```
df2
# printing DataFrame df2
```

	Name	Subject	Grade
0	Vaibhav	Six Sigma	A
1	Deepika	Statistics	B
2	Arjun	CommSkills	A
3	Chetan	Python	A
4	Abhishek	MLPython	B

pd.merge(DataFrame1, DataFrame2, on='CommonColumnName')

- produces inner join of DataFrame1 & DataFrame2 on common column

```
pd.merge(df1, df2, on='Subject')
# returns resultant DataFrame which joins df1 with df2 on column subject
# produces inner join by default

```

	Name_x	Subject	Grade_x	Name_y	Grade_y
0	Rakesh	Python	A	Chetan	A
1	Manoj	MLPython	C	Abhishek	B
2	Vaibhav	Statistics	B	Deepika	B
3	Hitesh	CommSkills	A	Arjun	A

pd.merge(DataFrame1, DataFrame2, on='CommonColumnName', how='JoinType')

```
pd.merge(df1, df2, on='Subject', how='inner')
# returns resultant DataFrame which joins df1 with df2
# on column 'subject' by 'inner join'

# performs intersection of two DataFrames
```

	Name_x	Subject	Grade_x	Name_y	Grade_y
0	Rakesh	Python	A	Chetan	A
1	Manoj	MLPython	C	Abhishek	B
2	Vaibhav	Statistics	B	Deepika	B
3	Hitesh	CommSkills	A	Arjun	A

```

```

```
pd.merge(df1, df2, on='Subject', how='outer')
# returns resultant DataFrame which joins df1 with df2
# on column 'subject' by 'outer join'

# performs union of two DataFrames
```

	Name_x	Subject	Grade_x	Name_y	Grade_y
0	Rakesh	Python	A	Chetan	A
1	Manoj	MLPython	C	Abhishek	B
2	Vaibhav	Statistics	B	Deepika	B
3	Hitesh	CommSkills	A	Arjun	A
4	Suyash	ProjectMgmt	B	NaN	NaN
5	NaN	Six Sigma	NaN	Vaibhav	A

```
pd.merge(df1, df2, on='Subject', how='left')
# returns resultant DataFrame which joins df1 with df2
# on column 'subject' by 'left join'
```

	Name_x	Subject	Grade_x	Name_y	Grade_y
0	Rakesh	Python	A	Chetan	A
1	Manoj	MLPython	C	Abhishek	B
2	Vaibhav	Statistics	B	Deepika	B
3	Hitesh	CommSkills	A	Arjun	A
4	Suyash	ProjectMgmt	B	NaN	NaN

```
pd.merge(df1, df2, on='Subject', how='right')
# returns resultant DataFrame which maps df1 with df2
# on column 'subject' by 'right join'
```

	Name_x	Subject	Grade_x	Name_y	Grade_y
0	NaN	Six Sigma	NaN	Vaibhav	A
1	Vaibhav	Statistics	B	Deepika	B
2	Hitesh	CommSkills	A	Arjun	A
3	Rakesh	Python	A	Chetan	A
4	Manoj	MLPython	C	Abhishek	B

```
df1
# printing actual DataFrame df1
```

	Name	Subject	Grade
0	Rakesh	Python	A
1	Manoj	MLPython	C
2	Vaibhav	Statistics	B
3	Hitesh	CommSkills	A

```
df2
# printing actual DataFrame df2
```

	Name	Subject	Grade
0	Vaibhav	Six Sigma	A
1	Deepika	Statistics	B
2	Arjun	CommSkills	A
3	Chetan	Python	A
4	Abhishek	MLPython	B

pd.merge(DataFrame1, DataFrame2, on=[listOfColumns], how='JoinType')

```
pd.merge(df1, df2, on=['Subject', 'Grade'], how='inner')
# returns resultant DataFrame which joins df1 with df2
# on two columns 'Subject' & 'Grade' by 'inner join'
```

	Name_x	Subject	Grade	Name_y
0	Rakesh	Python	A	Chetan
1	Vaibhav	Statistics	B	Deepika
2	Hitesh	CommSkills	A	Arjun

```
pd.merge(df1, df2, on=['Subject', 'Grade'], how='outer')
# returns resultant DataFrame which joins df1 with df2
# on two columns 'Subject' & 'Grade' by 'outer join'
```

	Name_x	Subject	Grade	Name_y
0	Rakesh	Python	A	Chetan
1	Manoj	MLPython	C	NaN
2	Vaibhav	Statistics	B	Deepika
3	Hitesh	CommSkills	A	Arjun
4	Suyash	ProjectMgmt	B	NaN
5	NaN	Six Sigma	A	Vaibhav
6	NaN	MLPython	B	Abhishek

```
pd.merge(df1, df2, on=['Subject', 'Grade'], how='left')
# returns resultant DataFrame which joins df1 with df2
# on two columns 'Subject' & 'Grade' by 'left join'
```

	Name_x	Subject	Grade	Name_y
0	Rakesh	Python	A	Chetan
1	Manoj	MLPython	C	NaN
2	Vaibhav	Statistics	B	Deepika
3	Hitesh	CommSkills	A	Arjun
4	Suyash	ProjectMgmt	B	NaN

```
pd.merge(df1, df2, on=['Subject', 'Grade'], how='right')
# returns resultant DataFrame which joins df1 with df2
# on two columns 'Subject' & 'Grade' by 'right join'
```

	Name_x	Subject	Grade	Name_y
0	NaN	Six Sigma	A	Vaibhav
1	Vaibhav	Statistics	B	Deepika
2	Hitesh	CommSkills	A	Arjun
3	Rakesh	Python	A	Chetan
4	NaN	MLPython	B	Abhishek

pd.merge with 'Salary' & 'Deptt'

```
import numpy as np
import  pandas as pd
```

```
from google.colab import files
uploaded=files.upload()
# locations.xlsx
# to be used with google colab

# os.chdir(r'D:/advanced-analytics-files/day03')
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving locations.xlsx to locations (1).xlsx

```
df1 = pd.read_excel('locations.xlsx', sheet_name='Salary')
# taking workSheet 'Salary' from WorkBook 'locations.xlsx' into DataFrame
```

```
df2 = pd.read_excel('locations.xlsx', sheet_name='Deptt')
# taking workSheet 'Deptt' from WorkBook 'locations.xlsx' into DataFrame
```

```
df1
# printing DataFrame df1
```

	Name	EmpID	Salary
0	Sudeep	A342	56
1	Deepika	J436	45
2	Chetan	B435	76
3	Abhishek	C234	47
4	Rishi	R475	38

```
df2
# printing DataFrame df2
```

	Name	EmpID	Deptt
0	Sudeep	A342	Quality
1	Ruchika	C578	Sales
2	Deepika	J436	Admin
3	Arjun	A342	Procurement
4	Rishi	R475	Prodn

```
pd.merge(df1, df2)
# same as inner merge / join
```

	Name	EmpID	Salary	Deptt
0	Sudeep	A342	56	Quality
1	Deepika	J436	45	Admin
2	Rishi	R475	38	Prodn

```
pd.merge(df1, df2, how='inner')
# inner merge/join acts like intersection
```

	Name	EmpID	Salary	Deptt
0	Sudeep	A342	56	Quality
1	Deepika	J436	45	Admin
2	Rishi	R475	38	Prodn

```
pd.merge(df1, df2, how='outer')
# outer merge/join acts like union
```

	Name	EmpID	Salary	Deptt
0	Sudeep	A342	56.0	Quality
1	Deepika	J436	45.0	Admin
2	Chetan	B435	76.0	NaN
3	Abhishek	C234	47.0	NaN
4	Rishi	R475	38.0	Prodn
5	Ruchika	C578	NaN	Sales
6	Arjun	A342	NaN	Procurement

```
df1
# printing DataFrame df1
```

	Name	EmpID	Salary
0	Sudeep	A342	56
1	Deepika	J436	45
2	Chetan	B435	76
3	Abhishek	C234	47
4	Rishi	R475	38

```
df2
# printing DataFrame df2
```

	Name	EmpID	Deptt
0	Sudeep	A342	Quality
1	Ruchika	C578	Sales
2	Deepika	J436	Admin
3	Arjun	A342	Procurement
4	Rishi	R475	Prodn

```
pd.merge(df1, df2, how='left')
# Note that, indexing based on 'left' DataFrame in case of left merge / join
```

	Name	EmpID	Salary	Deptt
0	Sudeep	A342	56	Quality
1	Deepika	J436	45	Admin

```
pd.merge(df1, df2, how='right')
# Note that, indexing based on 'right' DataFrame in case of right merge / join
```

	Name	EmpID	Salary	Deptt
0	Sudeep	A342	56.0	Quality
1	Ruchika	C578	NaN	Sales
2	Deepika	J436	45.0	Admin
3	Arjun	A342	NaN	Procurement
4	Rishi	R475	38.0	Prodn

pd.merge with 'inventory' & 'Info'

```
import numpy as np
import  pandas as pd
```

```
from google.colab import files
uploaded=files.upload()
# MaterialInfo.xlsx
# to be used with google colab

# os.chdir(r'D:/advanced-analytics-files/day03')
# to change current working directory to specified path
# to be used while running on local system
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving MaterialInfo.xlsx to MaterialInfo.xlsx

```
df1 = pd.read_excel('MaterialInfo.xlsx', sheet_name='inventory')
# taking workSheet 'inventory' from WorkBook 'MaterialInfo.xlsx' into DataFrame
df2 = pd.read_excel('MaterialInfo.xlsx', sheet_name='Info')
# taking workSheet 'Info' from WorkBook 'MaterialInfo.xlsx' into DataFrame
```

```
df1
# printing DataFrame df1
```

	MaterialID	Quantity
0	A	34
1	B	23
2	A	16
3	A	52
4	B	34
5	C	41
6	B	27
7	C	63

```
df2
# printing DataFrame df2
```

	MaterialID	Rate
0	A	1200
1	B	1500
2	C	2400

```
pd.merge(df1, df2)
# sorting will take place automatically as per first column MaterialID
```

	MaterialID	Quantity	Rate
0	A	34	1200
1	A	16	1200
2	A	52	1200
3	A	45	1200
4	A	38	1200
5	B	23	1500
6	B	34	1500
7	B	27	1500
8	B	36	1500
9	C	41	2400
10	C	63	2400
11	C	31	2400

```
pd.merge(df1, df2, how='left')
# indexing / sequencing as per left DataFrame
```

	MaterialID	Quantity	Rate
0	A	34	1200
1	B	23	1500
2	A	16	1200
3	A	52	1200
4	B	34	1500
5	C	41	2400
6	B	27	1500
7	C	63	2400
8	A	45	1200
9	B	36	1500
10	C	31	2400
11	A	38	1200

```
pd.merge(df1, df2, how='right')
# indexing / sequencing as per right DataFrame
```

	MaterialID	Quantity	Rate
0	A	34	1200
1	A	16	1200
2	A	52	1200
3	A	45	1200
4	A	38	1200
5	B	23	1500
6	B	34	1500
7	B	27	1500
8	B	36	1500
9	C	41	2400
10	C	63	2400
11	C	31	2400

```

```