```
import pandas as pd
import numpy as np

names = ['Arun',"Varun",'Ram',"Mohon"]
age=[29,21,32,51]

df = pd.DataFrame(list(zip(names,age)))
df
```

```
        0   1
0    Arun  29
1   Varun  21
2     Ram  32
3   Mohon  51
```

```
df = pd.DataFrame(list(zip(names,age)),columns=["Name",'Val'])
df
```

```
    Name  Val
0   Arun   29
1  Varun   21
2    Ram   32
3  Mohon   51
```

```
lst =[
    {'name':"Arun",'age':29,"gender":"M"},
        {'name':"Varun",'age':21,"gender":"M"},
        {'name':"Ram",'age':32,"gender":"M"}
]

df = pd.DataFrame(lst)
df
```

```
    name  age gender
0   Arun   29      M
1  Varun   21      M
2    Ram   32      M
```

```
df =pd.read_csv('harry.csv')
df
```

```
   Unnamed: 0.3  Unnamed: 0.2  Unnamed: 0.1  Unnamed: 0  Train No
Name  \
0             0             0             0           0     12655
Nazmul
1             1             1             1           1     46645
Siam
2             2             2             2           2     54646
Sajid

   Marks       City
0     90  Madaripur
```

```
1     80   Barishal
2     95  Jheniadoh
```

```
df =pd.DataFrame #Blank DataFrame
df
```

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

```
df = pd.DataFrame(columns=['name','age']) #Blank DataFrame
df
```

```
Empty DataFrame
Columns: [name, age]
Index: []
```

```
df = pd.read_csv('name_age.csv')
df
```

```
    name  age        dob gender
0   Rita   23   20/02/97      f
1   Arun   29   1/7/1991      m
2   Sita   14   7/7/2006      f
3  Varun   21   1/5/1999      m
4    Ram   32  7/11/1988      m
5  Radha   23   6/9/1997      f
6  Mohan   51   3/3/1969      m
7   Devi   20   5/1/2000      f
8  Nidhi   29  10/1/1991      f
```

```
df.shape #output is row by column ,Output is a tuple
```

```
(9, 4)
```

```
rows,column =df.shape
```

```
rows
```

```
9
```

```
column
```

```
4
```

```
df.head()
```

```
    name  age        dob gender
0   Rita   23   20/02/97      f
1   Arun   29   1/7/1991      m
2   Sita   14   7/7/2006      f
3  Varun   21   1/5/1999      m
4    Ram   32  7/11/1988      m
```

```
df.head(3)
```

```
   name  age        dob gender
0  Rita   23   20/02/97      f
1  Arun   29   1/7/1991      m
2  Sita   14   7/7/2006      f
```

```python
df.tail()
```

```
    name  age        dob gender
4    Ram   32  7/11/1988      m
5  Radha   23   6/9/1997      f
6  Mohan   51   3/3/1969      m
7   Devi   20   5/1/2000      f
8  Nidhi   29  10/1/1991      f
```

```python
df[5:8] # include row #5 an exclude row #8
```

```
    name  age       dob gender
5  Radha   23  6/9/1997      f
6  Mohan   51  3/3/1969      m
7   Devi   20  5/1/2000      f
```

```python
df[:] #All rows
```

```
    name  age        dob gender
0   Rita   23   20/02/97      f
1   Arun   29   1/7/1991      m
2   Sita   14   7/7/2006      f
3  Varun   21   1/5/1999      m
4    Ram   32  7/11/1988      m
5  Radha   23   6/9/1997      f
6  Mohan   51   3/3/1969      m
7   Devi   20   5/1/2000      f
8  Nidhi   29  10/1/1991      f
```

```python
df.columns
```

```
Index(['name', 'age', 'dob', 'gender'], dtype='object')
```

```python
# Read the CSV file
df = pd.read_csv('name_age.csv')  # Replace with your file path
df
```

```
    name  age        dob gender
0   Rita   23   20/02/97      f
1   Arun   29   1/7/1991      m
2   Sita   14   7/7/2006      f
3  Varun   21   1/5/1999      m
4    Ram   32  7/11/1988      m
5  Radha   23   6/9/1997      f
6  Mohan   51   3/3/1969      m
```

```
7   Devi   20   5/1/2000      f
8  Nidhi   29  10/1/1991      f
```

```
df.columns
```

```
Index(['name', 'age', 'dob', 'gender'], dtype='object')
```

```
df
```

```
    name  age        dob gender
0   Rita   23   20/02/97      f
1   Arun   29   1/7/1991      m
2   Sita   14   7/7/2006      f
3  Varun   21   1/5/1999      m
4    Ram   32  7/11/1988      m
5  Radha   23   6/9/1997      f
6  Mohan   51   3/3/1969      m
7   Devi   20   5/1/2000      f
8  Nidhi   29  10/1/1991      f
```

```
df["name"]
```

```
0     Rita
1     Arun
2     Sita
3    Varun
4      Ram
5    Radha
6    Mohan
7     Devi
8    Nidhi
Name: name, dtype: object
```

```
df["age"]
```

```
0    23
1    29
2    14
3    21
4    32
5    23
6    51
7    20
8    29
Name: age, dtype: int64
```

```
df.name
```

```
0     Rita
1     Arun
2     Sita
```

```
3     Varun
4       Ram
5     Radha
6     Mohan
7      Devi
8     Nidhi
Name: name, dtype: object
```

```
df.age
```

```
0    23
1    29
2    14
3    21
4    32
5    23
6    51
7    20
8    29
Name: age, dtype: int64
```

```
df[['name','age']].head() #Column name for accessing column
                          #Range for accessing row
```

```
    name  age
0   Rita   23
1   Arun   29
2   Sita   14
3  Varun   21
4    Ram   32
```

```
type(df)
```

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

```
type(df["age"])
```

```
pandas.core.series.Series
```

```
df['age'].max()
```

```
51
```

```
df['age'].min()
```

```
14
```

```
df['age'].mean()
```

```
26.88888888888889
```

```
df["age"].std()
```

```
10.576441325470071
```

```
df.describe()
```

```
             age
count   9.000000
mean   26.888889
std    10.576441
min    14.000000
25%    21.000000
50%    23.000000
75%    29.000000
max    51.000000
```

```
df[df['age']>30]
```

```
    name  age        dob gender
4    Ram   32  7/11/1988      m
6  Mohan   51   3/3/1969      m
```

```
df[df['age']==df['age'].min()]
```

```
   name  age       dob gender
2  Sita   14  7/7/2006      f
```

```
df['name'][df['age'] > 30]
```

```
4      Ram
6    Mohan
Name: name, dtype: object
```

```
df[['name','dob','age']][df['age'] < 30] #multiple bracket for
multiple column
                                        #single bracket for accessing
single column
```

```
    name        dob  age
0   Rita   20/02/97   23
1   Arun   1/7/1991   29
2   Sita   7/7/2006   14
3  Varun   1/5/1999   21
5  Radha   6/9/1997   23
7   Devi   5/1/2000   20
8  Nidhi  10/1/1991   29
```

```
# accessing row with loc and iloc
 # Loc takes the index value and gives the result for that index,
whereas iloc
  # takes the position of an index and gives the row.
df.loc[0]
```

```
name          Rita
age             23
dob       20/02/97
gender           f
Name: 0, dtype: object
```

```
df.iloc[1]
```

```
name          Arun
age             29
dob       1/7/1991
gender           m
Name: 1, dtype: object
```

```
df.index
```

```
RangeIndex(start=0, stop=9, step=1)
```

```
#df=df.set_index('name')
```

```
df.set_index('name',inplace=True)
```

```
df.head()
```

```
       age         dob gender
name
Rita    23   20/02/97      f
Arun    29   1/7/1991      m
Sita    14   7/7/2006      f
Varun   21   1/5/1999      m
Ram     32  7/11/1988      m
```

```
df.loc["Arun"]
```

```
age             29
dob       1/7/1991
gender           m
Name: Arun, dtype: object
```

```
df.iloc[0]
```

```
age             23
dob       20/02/97
gender           f
Name: Rita, dtype: object
```

```
df=pd.read_csv('name_age.csv',header=None)
df
```

```
       0    1          2       3
0   name  age        dob  gender
1   Rita   23   20/02/97       f
```

```
2   Arun   29   1/7/1991       m
3   Sita   14   7/7/2006       f
4  Varun   21   1/5/1999       m
5    Ram   32  7/11/1988       m
6  Radha   23   6/9/1997       f
7  Mohan   51   3/3/1969       m
8   Devi   20   5/1/2000       f
9  Nidhi   29  10/1/1991       f
```

```
# Adding custom header
df=pd.read_csv('name_age.csv',
names=['Name','Age','dateOfBirth',"GENDER"])
df.head()
```

```
    Name  Age dateOfBirth  GENDER
0   name  age         dob  gender
1   Rita   23    20/02/97       f
2   Arun   29    1/7/1991       m
3   Sita   14    7/7/2006       f
4  Varun   21    1/5/1999       m
```

```
# ‘nrows’ will be the number of rows we want to see
df = pd.read_csv('name_age.csv',nrows=4)
df.head()
```

```
    name  age       dob gender
0   Rita   23  20/02/97      f
1   Arun   29  1/7/1991      m
2   Sita   14  7/7/2006      f
3  Varun   21  1/5/1999      m
```

```
df= pd.read_csv('prac2.csv')
df.head()
```

```
    name Unnamed: 1 Unnamed: 2      Unnamed: 3
0    NaN        NaN        NaN             NaN
1   Arun         29   1/7/1991               m
2   Sita         14   7/7/2006               f
3  Varun         21   1/5/1999               m
4    Ram         32  7/11/1988  not avaialable
```

```
df = pd.read_csv('prac2.csv',skiprows=2) #skip first two rows
df
```

```
    Arun  29   1/7/1991               m
0   Sita  14   7/7/2006               f
1  Varun  21   1/5/1999               m
2    Ram  32  7/11/1988  not avaialable
3  Radha  na   6/9/1997               f
4  Mohan  51   3/3/1969               m
```

```
5   Devi  20   5/1/2000               f
6  Nidhi  29  10/1/1991               f
```

```
df = pd.read_csv("name_age.csv")
df
```

```
    name  age        dob gender
0   Rita   23   20/02/97      f
1   Arun   29   1/7/1991      m
2   Sita   14   7/7/2006      f
3  Varun   21   1/5/1999      m
4    Ram   32  7/11/1988      m
5  Radha   23   6/9/1997      f
6  Mohan   51   3/3/1969      m
7   Devi   20   5/1/2000      f
8  Nidhi   29  10/1/1991      f
```

```
df = pd.read_csv("name_age.csv", skiprows=2)  #Skip two rows
df
```

```
    Arun  29   1/7/1991  m
0   Sita  14   7/7/2006  f
1  Varun  21   1/5/1999  m
2    Ram  32  7/11/1988  m
3  Radha  23   6/9/1997  f
4  Mohan  51   3/3/1969  m
5   Devi  20   5/1/2000  f
6  Nidhi  29  10/1/1991  f
```

```
df = pd.read_csv("name_age.csv", header = 2) #Skip two rows
df
```

```
    Arun  29   1/7/1991  m
0   Sita  14   7/7/2006  f
1  Varun  21   1/5/1999  m
2    Ram  32  7/11/1988  m
3  Radha  23   6/9/1997  f
4  Mohan  51   3/3/1969  m
5   Devi  20   5/1/2000  f
6  Nidhi  29  10/1/1991  f
```

```
df = pd.read_csv('prac2.csv')
df
```

```
    name Unnamed: 1 Unnamed: 2      Unnamed: 3
0    NaN        NaN        NaN             NaN
1   Arun         29   1/7/1991               m
2   Sita         14   7/7/2006               f
3  Varun         21   1/5/1999               m
4    Ram         32  7/11/1988  not avaialable
5  Radha         na   6/9/1997               f
```

```
6  Mohan         51   3/3/1969              m
7   Devi         20   5/1/2000              f
8  Nidhi         29  10/1/1991              f
```

```
df = pd.read_csv('prac2.csv',na_values=['na','not avaialable'])
df
```

```
    name  Unnamed: 1 Unnamed: 2 Unnamed: 3
0    NaN         NaN        NaN        NaN
1   Arun        29.0   1/7/1991          m
2   Sita        14.0   7/7/2006          f
3  Varun        21.0   1/5/1999          m
4    Ram        32.0  7/11/1988        NaN
5  Radha         NaN   6/9/1997          f
6  Mohan        51.0   3/3/1969          m
7   Devi        20.0   5/1/2000          f
8  Nidhi        29.0  10/1/1991          f
```

```
df = pd.read_csv('name_age.csv')
df
# cleaning has been applied to a specific column
```

```
    name  age        dob gender
0   Rita   23   20/02/97      f
1   Arun   29   1/7/1991      m
2   Sita   14   7/7/2006      f
3  Varun   21   1/5/1999      m
4    Ram   32  7/11/1988      m
5  Radha   23   6/9/1997      f
6  Mohan   51   3/3/1969      m
7   Devi   20   5/1/2000      f
8  Nidhi   29  10/1/1991      f
```

```
df.to_csv('newCSV.csv') #df is created a new file newCSV
```

```
!dir "newCSV.csv"
```

```
 Volume in drive C is Acer
 Volume Serial Number is D6B1-BC2D

 Directory of C:\Users\USER

05/14/2025  08:18 PM               225 newCSV.csv
               1 File(s)            225 bytes
               0 Dir(s)  76,663,754,752 bytes free
```

```
df
```

```
    name  age        dob gender
0   Rita   23   20/02/97      f
1   Arun   29   1/7/1991      m
```

```
2   Sita   14   7/7/2006      f
3  Varun   21   1/5/1999      m
4    Ram   32  7/11/1988      m
5  Radha   23   6/9/1997      f
6  Mohan   51   3/3/1969      m
7   Devi   20   5/1/2000      f
8  Nidhi   29  10/1/1991      f
```

```
df.to_csv('newCSV.csv',index=False)
df.to_csv()
```

```
',name,age,dob,gender\r\n0,Rita,23,20/02/97,f\r\n1,Arun,29,1/7/1991,m\
r\n2,Sita,14,7/7/2006,f\r\n3,Varun,21,1/5/1999,m\r\
n4,Ram,32,7/11/1988,m\r\n5,Radha,23,6/9/1997,f\r\
n6,Mohan,51,3/3/1969,m\r\n7,Devi,20,5/1/2000,f\r\
n8,Nidhi,29,10/1/1991,f\r\n'
```

```
df = pd.read_csv('newCSV.csv')
print(df.to_csv(index=False))
```

```
name,age,dob,gender
Rita,23,20/02/97,f
Arun,29,1/7/1991,m
Sita,14,7/7/2006,f
Varun,21,1/5/1999,m
Ram,32,7/11/1988,m
Radha,23,6/9/1997,f
Mohan,51,3/3/1969,m
Devi,20,5/1/2000,f
Nidhi,29,10/1/1991,f
```

```
df.to_csv("newCSV.csv", columns=["name", "age", 'dob'], index=False)
df_new = pd.read_csv('newCSV.csv')
df_new.head()
```

```
    name  age        dob
0   Rita   23   20/02/97
1   Arun   29   1/7/1991
2   Sita   14   7/7/2006
3  Varun   21   1/5/1999
4    Ram   32  7/11/1988
```

```
df.to_csv("newCSV.csv", header=False, index=False)
df_new = pd.read_csv('newCSV.csv')
df_new
```

```
    Rita  23   20/02/97  f
0   Arun  29   1/7/1991  m
```

```
1   Sita  14   7/7/2006  f
2  Varun  21   1/5/1999  m
3    Ram  32  7/11/1988  m
4  Radha  23   6/9/1997  f
5  Mohan  51   3/3/1969  m
6   Devi  20   5/1/2000  f
7  Nidhi  29  10/1/1991  f
```

```
import pandas as pd

# Sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['NY', 'LA', 'Chicago']}
df = pd.DataFrame(data)

# Export WITHOUT headers
df.to_csv("no_headers.csv", index=False, header=False)

# Show file content
print("CSV WITHOUT headers:")
with open("no_headers.csv") as f:
    print(f.read())
```

```
CSV WITHOUT headers:
Alice,25,NY
Bob,30,LA
Charlie,35,Chicago
```

```
# Export WITH headers (default behavior)
df.to_csv("with_headers.csv", index=False)  # header=True is default

# Show file content
print("CSV WITH headers:")
with open("with_headers.csv") as f:
    print(f.read())
```

```
CSV WITH headers:
Name,Age,City
Alice,25,NY
Bob,30,LA
Charlie,35,Chicago
```

```
df_new = pd.read_excel('prac3.xlsx')
df_new.head()
```

```
    name  age                  dob gender
0   Rita   23  2015-01-01 00:00:00      f
1   Arun   29  2017-02-14 00:00:00      m
```

```
2   Sita   14  1999-07-07 00:00:00      f
3  Varun   21                4//5/8      m
4    Ram   32  1997-09-09 00:00:00      m
```

```
df_new = pd.read_excel('prac3.xlsx','Sheet2')
df_new
```

```
  Name        dob  class
0   AB 2025-08-07      8
1   CD 2025-04-07      9
```

```
def changeClass(cell):
    if cell==8:
        return 'Eight'
    elif cell==9:
        return 'Nine'
    else:
        return 'NA'


df_new = pd.read_excel('prac3.xlsx',sheet_name='Sheet2',converters ={
    'class' : changeClass
})
df_new.head()
```

```
  Name        dob  class
0   AB 2025-08-07  Eight
1   CD 2025-04-07   Nine
```

```
df_new= pd.read_excel("newExcel.xlsx","Sheet2")
df_new
```

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

```
df_new.to_excel("newExcel.xlsx",sheet_name='Sheet2',index=False,header
=False)
df_new
#overwrite the entire file
```

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

```
df1=pd.read_excel('prac4.xlsx',sheet_name='Sheet1')
df1.head()
```

```
    name  age                  dob gender
0   Rita   23             20/02/97      f
1   Arun   29  1991-01-07 00:00:00      m
2   Sita   14  2006-07-07 00:00:00      f
```

```
3  Varun   21  1999-01-05 00:00:00      m
4    Ram   32  1988-07-11 00:00:00      m
```

```
df2=pd.read_excel('prac4.xlsx',sheet_name='Sheet2')
df2.head()
```

```
   name  age                  dob gender
0  Rita   23             20/02/97      f
1  Arun   29  1991-01-07 00:00:00      m
2  Sita   14  2006-07-07 00:00:00      f
```

```
with pd.ExcelWriter("multipleSheet.xlsx") as writer:
    df1.to_excel(writer,sheet_name='Sheet1',index=False)
    df2.to_excel(writer,sheet_name='Sheet2',index=False)
```

```
print(df1.to_csv(index=False))
```

```
name,age,dob,gender
Rita,23,20/02/97,f
Arun,29,1991-01-07 00:00:00,m
Sita,14,2006-07-07 00:00:00,f
Varun,21,1999-01-05 00:00:00,m
Ram,32,1988-07-11 00:00:00,m
Radha,23,1997-06-09 00:00:00,f
Mohan,51,1969-03-03 00:00:00,m
Devi,20,2000-05-01 00:00:00,f
Nidhi,29,1991-10-01 00:00:00,f
```

```
df = pd.read_excel('multipleSheet.xlsx',sheet_name='Sheet1')
df
```

```
    name  age                  dob gender
0   Rita   23             20/02/97      f
1   Arun   29  1991-01-07 00:00:00      m
2   Sita   14  2006-07-07 00:00:00      f
3  Varun   21  1999-01-05 00:00:00      m
4    Ram   32  1988-07-11 00:00:00      m
5  Radha   23  1997-06-09 00:00:00      f
6  Mohan   51  1969-03-03 00:00:00      m
7   Devi   20  2000-05-01 00:00:00      f
8  Nidhi   29  1991-10-01 00:00:00      f
```

```
df.to_csv('test1.txt',sep='\t',index=False)
```

```
df=pd.read_csv('test1.txt')
df
```

```
              name\tage\tdob\tgender
0              Rita\t23\t20/02/97\tf
```

```
1   Arun\t29\t1991-01-07 00:00:00\tm
2   Sita\t14\t2006-07-07 00:00:00\tf
3  Varun\t21\t1999-01-05 00:00:00\tm
4    Ram\t32\t1988-07-11 00:00:00\tm
5  Radha\t23\t1997-06-09 00:00:00\tf
6  Mohan\t51\t1969-03-03 00:00:00\tm
7   Devi\t20\t2000-05-01 00:00:00\tf
8  Nidhi\t29\t1991-10-01 00:00:00\tf
```

```
df = pd.read_csv('weather.csv',parse_dates=["date"])
```

```
df
```

```
        date  temperature  windSpeed  status  Unnamed: 4
0 2020-05-06      35.6582  10.788378   sunny         NaN
1 2019-01-30          NaN        NaN     NaN         NaN
2 2023-10-27      30.9343        NaN   rainy         NaN
3 2024-11-29          NaN   6.889682  cloudy         NaN
4 2025-08-11      13.9082  19.012990   rainy         NaN
5 2024-09-09      23.9382        NaN   sunny         NaN
```

```
df.set_index('date',inplace=True)
df
```

```
            temperature  windSpeed  status  Unnamed: 4
date
2020-05-06      35.6582  10.788378   sunny         NaN
2019-01-30          NaN        NaN     NaN         NaN
2023-10-27      30.9343        NaN   rainy         NaN
2024-11-29          NaN   6.889682  cloudy         NaN
2025-08-11      13.9082  19.012990   rainy         NaN
2024-09-09      23.9382        NaN   sunny         NaN
```

```
pd.isna(df['temperature'])
```

```
date
2020-05-06    False
2019-01-30     True
2023-10-27    False
2024-11-29     True
2025-08-11    False
2024-09-09    False
Name: temperature, dtype: bool
```

```
df['temperature'].notna()
```

```
date
2020-05-06     True
2019-01-30    False
2023-10-27     True
2024-11-29    False
```

```
2025-08-11     True
2024-09-09     True
Name: temperature, dtype: bool
```

```
pd.notna(df['windSpeed'])
```

```
date
2020-05-06     True
2019-01-30    False
2023-10-27    False
2024-11-29     True
2025-08-11     True
2024-09-09    False
Name: windSpeed, dtype: bool
```

```
df.isna()
```

```
            temperature  windSpeed  status  Unnamed: 4
date
2020-05-06        False      False   False        True
2019-01-30         True       True    True        True
2023-10-27        False       True   False        True
2024-11-29         True      False   False        True
2025-08-11        False      False   False        True
2024-09-09        False       True   False        True
```

```
df.notna()
```

```
            temperature  windSpeed  status  Unnamed: 4
date
2020-05-06         True       True    True       False
2019-01-30        False      False   False       False
2023-10-27         True      False    True       False
2024-11-29        False       True    True       False
2025-08-11         True       True    True       False
2024-09-09         True      False    True       False
```

```
None==None
```

```
True
```

```
np.nan == np.nan
```

```
False
```

```
np.nan != np.nan
```

```
True
```

```
df['temperature']==np.nan
```

```
date
2020-05-06    False
2019-01-30    False
2023-10-27    False
2024-11-29    False
2025-08-11    False
2024-09-09    False
Name: temperature, dtype: bool
```

```
import pandas as pd

data = {
    'date': ["20200508", "20200509", "20200510", "20200511",
"20200512", "20200513"],  # Filled missing dates with adjacent days
    'temperature': [35.6582, 30.0, 30.9343, 25.0, np.nan, 23.9382],  #
Replaced NaN with approximate values
    'windSpeed': [10.788378, 5.0, 8.0, np.nan, 19.012990, 7.0],      #
Replaced NaN with typical wind speeds
    'status': ["sunny",np.nan, "rainy", "cloudy", np.nan, np.nan]  #
Replaced NaN with plausible weather
}

df = pd.DataFrame(data)
print(df)
```

```
       date  temperature  windSpeed  status
0  20200508      35.6582  10.788378   sunny
1  20200509      30.0000   5.000000     NaN
2  20200510      30.9343   8.000000   rainy
3  20200511      25.0000        NaN  cloudy
4  20200512          NaN  19.012990     NaN
5  20200513      23.9382   7.000000     NaN
```

```
df['date'] = pd.to_datetime(df['date'], format='%Y%m%d')
df
```

```
        date  temperature  windSpeed  status
0 2020-05-08      35.6582  10.788378   sunny
1 2020-05-09      30.0000   5.000000     NaN
2 2020-05-10      30.9343   8.000000   rainy
3 2020-05-11      25.0000        NaN  cloudy
4 2020-05-12          NaN  19.012990     NaN
5 2020-05-13      23.9382   7.000000     NaN
```

```
df.loc[[0,4,5],['date']] = np.nan
df
```

```
        date  temperature  windSpeed  status
0        NaT      35.6582  10.788378   sunny
1 2020-05-09      30.0000   5.000000     NaN
2 2020-05-10      30.9343   8.000000   rainy
```

```
3 2020-05-11      25.0000        NaN  cloudy
4        NaT          NaN  19.012990     NaN
5        NaT      23.9382   7.000000     NaN
```

```
s =pd.Series(['aa','bb','cc'])
s.loc[0]=None
s.loc[1]=np.nan
s
```

```
0    None
1     NaN
2      cc
dtype: object
```

```
s =pd.Series([11,22,33])
s.loc[0]=None     # missing number always show NaN
s.loc[1]=np.nan   #misssing datetime always show NaT
s
```

```
0     NaN
1     NaN
2    33.0
dtype: float64
```

```
df.set_index('date',inplace=True)
df
```

```
            temperature  windSpeed  status
date
NaT             35.6582  10.788378   sunny
2020-05-09      30.0000   5.000000     NaN
2020-05-10      30.9343   8.000000   rainy
2020-05-11      25.0000        NaN  cloudy
NaT                 NaN  19.012990     NaN
NaT             23.9382   7.000000     NaN
```

```
df.fillna(0)
```

```
            temperature  windSpeed  status
date
NaT             35.6582  10.788378   sunny
2020-05-09      30.0000   5.000000       0
2020-05-10      30.9343   8.000000   rainy
2020-05-11      25.0000   0.000000  cloudy
NaT              0.0000  19.012990       0
NaT             23.9382   7.000000       0
```

```
df.fillna({
    'temperature':0,
        'windSpeed':0,
           'status':'sunny'
})
```

```
            temperature  windSpeed  status
date
NaT             35.6582  10.788378   sunny
2020-05-09      30.0000   5.000000   sunny
2020-05-10      30.9343   8.000000   rainy
2020-05-11      25.0000   0.000000  cloudy
NaT              0.0000  19.012990   sunny
NaT             23.9382   7.000000   sunny
```

```
df['status'].fillna('new status')
```

```
date
NaT                 sunny
2020-05-09     new status
2020-05-10          rainy
2020-05-11         cloudy
NaT            new status
NaT            new status
Name: status, dtype: object
```

```
df.fillna(method='ffill')
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1193302488.py:1:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will
raise in a future version. Use obj.ffill() or obj.bfill() instead.
  df.fillna(method='ffill')
```

```
            temperature  windSpeed  status
date
NaT             35.6582  10.788378   sunny
2020-05-09      30.0000   5.000000   sunny
2020-05-10      30.9343   8.000000   rainy
2020-05-11      25.0000   8.000000  cloudy
NaT             25.0000  19.012990  cloudy
NaT             23.9382   7.000000  cloudy
```

```
df
```

```
            temperature  windSpeed  status
date
NaT             35.6582  10.788378   sunny
2020-05-09      30.0000   5.000000     NaN
2020-05-10      30.9343   8.000000   rainy
2020-05-11      25.0000        NaN  cloudy
NaT                 NaN  19.012990     NaN
NaT             23.9382   7.000000     NaN
```

```
df.fillna(method='bfill')
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\2831856154.py:1:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will
raise in a future version. Use obj.ffill() or obj.bfill() instead.
  df.fillna(method='bfill')
```

```
            temperature  windSpeed  status
date
NaT             35.6582  10.788378   sunny
2020-05-09      30.0000   5.000000   rainy
2020-05-10      30.9343   8.000000   rainy
2020-05-11      25.0000  19.012990  cloudy
NaT             23.9382  19.012990     NaN
NaT             23.9382   7.000000     NaN
```

```
df.fillna(method='ffill',axis='columns')
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1140326825.py:1:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will
raise in a future version. Use obj.ffill() or obj.bfill() instead.
  df.fillna(method='ffill',axis='columns')
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1140326825.py:1:
FutureWarning: Downcasting object dtype arrays
on .fillna, .ffill, .bfill is deprecated and will change in a future
version. Call result.infer_objects(copy=False) instead. To opt-in to
the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  df.fillna(method='ffill',axis='columns')
```

```
           temperature  windSpeed    status
date
NaT            35.6582  10.788378     sunny
2020-05-09        30.0        5.0       5.0
2020-05-10     30.9343        8.0     rainy
2020-05-11        25.0       25.0    cloudy
NaT                NaN   19.01299  19.01299
NaT            23.9382        7.0       7.0
```

```
df.fillna(method='bfill',axis='columns')
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\3576732574.py:1:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will
raise in a future version. Use obj.ffill() or obj.bfill() instead.
  df.fillna(method='bfill',axis='columns')
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\3576732574.py:1:
FutureWarning: Downcasting object dtype arrays
on .fillna, .ffill, .bfill is deprecated and will change in a future
version. Call result.infer_objects(copy=False) instead. To opt-in to
the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  df.fillna(method='bfill',axis='columns')
```

```
            temperature  windSpeed  status
date                                      
NaT             35.6582  10.788378   sunny
2020-05-09         30.0        5.0     NaN
2020-05-10      30.9343        8.0   rainy
2020-05-11         25.0     cloudy  cloudy
NaT            19.01299   19.01299     NaN
NaT             23.9382        7.0     NaN
```

```
df
```

```
            temperature  windSpeed  status
date                                      
NaT             35.6582  10.788378   sunny
2020-05-09      30.0000   5.000000     NaN
2020-05-10      30.9343   8.000000   rainy
2020-05-11      25.0000        NaN  cloudy
NaT                 NaN  19.012990     NaN
NaT             23.9382   7.000000     NaN
```

```
df.fillna(method='ffill',limit=1)
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\393035141.py:1: 
FutureWarning: DataFrame.fillna with 'method' is deprecated and will 
raise in a future version. Use obj.ffill() or obj.bfill() instead.
  df.fillna(method='ffill',limit=1)
```

```
            temperature  windSpeed  status
date                                      
NaT             35.6582  10.788378   sunny
2020-05-09      30.0000   5.000000   sunny
2020-05-10      30.9343   8.000000   rainy
2020-05-11      25.0000   8.000000  cloudy
NaT             25.0000  19.012990  cloudy
NaT             23.9382   7.000000     NaN
```

```
df
```

```
            temperature  windSpeed  status
date                                      
NaT             35.6582  10.788378   sunny
2020-05-09      30.0000   5.000000     NaN
2020-05-10      30.9343   8.000000   rainy
2020-05-11      25.0000        NaN  cloudy
NaT                 NaN  19.012990     NaN
NaT             23.9382   7.000000     NaN
```

```
cols = ['temperature', 'windSpeed']
df[cols] = df[cols].fillna(df[cols].mean()) #multiple columns mean at 
a time
df
```

```
            temperature  windSpeed  status
date
NaT            35.65820  10.788378   sunny
2020-05-09     30.00000   5.000000     NaN
2020-05-10     30.93430   8.000000   rainy
2020-05-11     25.00000   9.960274  cloudy
NaT            29.10614  19.012990     NaN
NaT            23.93820   7.000000     NaN
```

```
df['windSpeed'].mean()
```

```
9.9602736
```

```
df['temperature'].mean()
```

```
29.10614
```

```
df
```

```
            temperature  windSpeed  status
date
NaT            35.65820  10.788378   sunny
2020-05-09     30.00000   5.000000     NaN
2020-05-10     30.93430   8.000000   rainy
2020-05-11     25.00000   9.960274  cloudy
NaT            29.10614  19.012990     NaN
NaT            23.93820   7.000000     NaN
```

```
import pandas as pd

data = {
    'date': ['2020-05-06', '2020-05-09', '2020-05-10', '2020-05-11',
'2020-05-12', '2020-05-13'],
    'temperature': [35.658200, 32.115275, 30.934300, 22.421250,
13.908200, 23.938200],
    'windSpeed': [10.788378, 8.449161, 7.669422, 6.889682, 19.012990,
19.012990],
    'status': ['sunny', None, 'rainy', 'cloudy', 'rainy', 'sunny']
}

df = pd.DataFrame(data)
print(df)
```

```
         date  temperature  windSpeed  status
0  2020-05-06    35.658200  10.788378   sunny
1  2020-05-09    32.115275   8.449161    None
2  2020-05-10    30.934300   7.669422   rainy
3  2020-05-11    22.421250   6.889682  cloudy
4  2020-05-12    13.908200  19.012990   rainy
5  2020-05-13    23.938200  19.012990   sunny
```

```
df.interpolate()
```

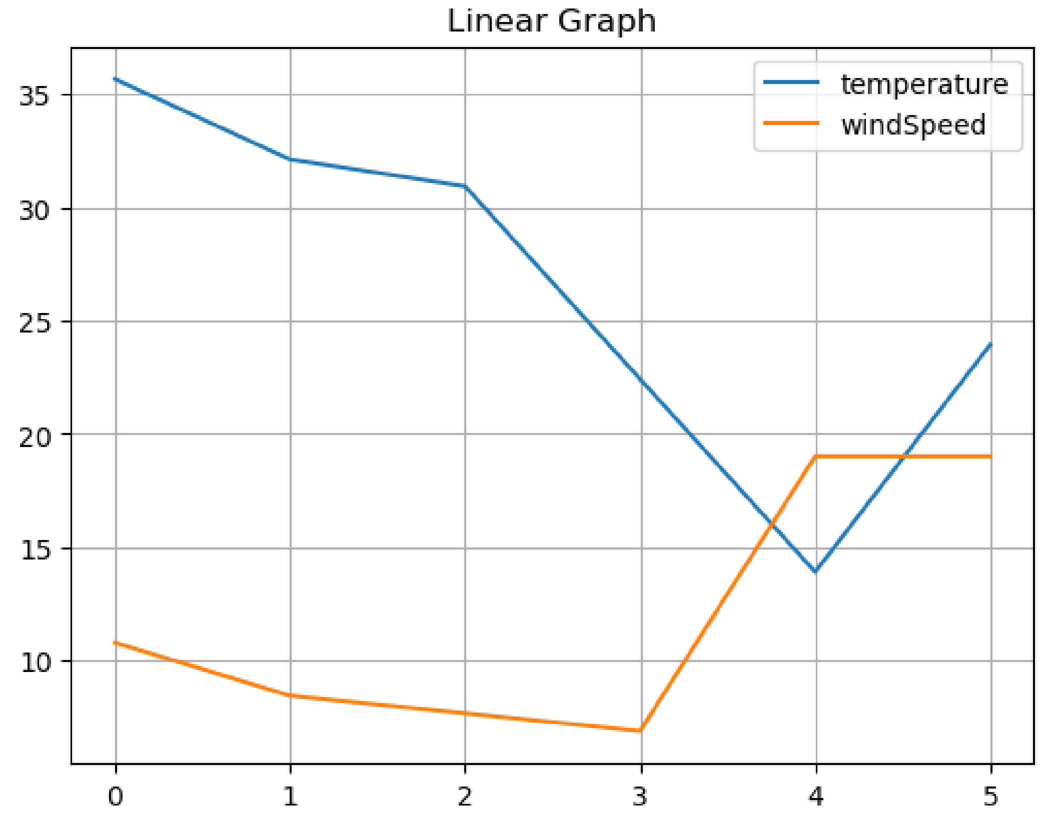
```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\4002874584.py:1:
FutureWarning: DataFrame.interpolate with object dtype is deprecated
and will raise in a future version. Call obj.infer_objects(copy=False)
before interpolating instead.
  df.interpolate()
```

```
         date  temperature  windSpeed  status
0  2020-05-06    35.658200  10.788378   sunny
1  2020-05-09    32.115275   8.449161    None
2  2020-05-10    30.934300   7.669422   rainy
3  2020-05-11    22.421250   6.889682  cloudy
4  2020-05-12    13.908200  19.012990   rainy
5  2020-05-13    23.938200  19.012990   sunny
```

```
df.interpolate().plot(title='Linear Graph',grid=1)
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\3340644183.py:1:
FutureWarning: DataFrame.interpolate with object dtype is deprecated
and will raise in a future version. Call obj.infer_objects(copy=False)
before interpolating instead.
  df.interpolate().plot(title='Linear Graph',grid=1)
```

```
<Axes: title={'center': 'Linear Graph'}>
```



```
df=df.set_index('date')
df
```

```
            temperature  windSpeed  status
date
2020-05-06    35.658200  10.788378   sunny
2020-05-09    32.115275   8.449161    None
2020-05-10    30.934300   7.669422   rainy
2020-05-11    22.421250   6.889682  cloudy
2020-05-12    13.908200  19.012990   rainy
2020-05-13    23.938200  19.012990   sunny
```

```
df.columns
```

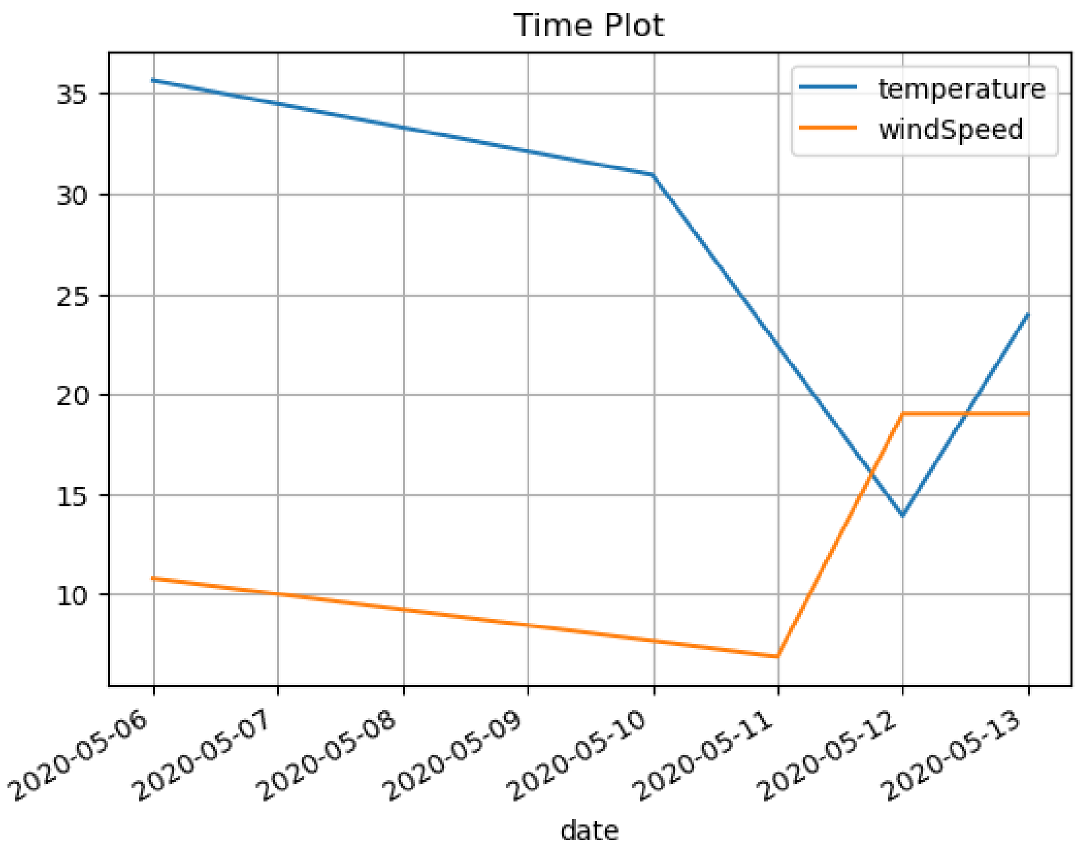
```
Index(['temperature', 'windSpeed', 'status'], dtype='object')
```

```
import pandas as pd

# Sample data with NaN in 'status' (non-numeric) and missing
temperature (numeric)
data = {
    'date': ['2020-05-06', '2020-05-09', '2020-05-10', '2020-05-11',
'2020-05-12', '2020-05-13'],
    'temperature': [35.658200, 32.115275, 30.934300, 22.421250,
13.908200, 23.938200],
    'windSpeed': [10.788378, 8.449161, 7.669422, 6.889682, 19.012990,
19.012990],
    'status': ['sunny', None, 'rainy', 'cloudy', 'rainy', 'sunny']
}

# Convert to DataFrame and set 'date' as datetime index
df = pd.DataFrame(data)
df['date'] = pd.to_datetime(df['date'])  # Convert to datetime
df.set_index('date', inplace=True)       # Set as index

# Interpolate ONLY numeric columns (temperature, windSpeed)
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
df[numeric_cols] = df[numeric_cols].interpolate(method='time')

print(df)
```

```
            temperature  windSpeed  status
date
2020-05-06    35.658200  10.788378   sunny
2020-05-09    32.115275   8.449161    None
2020-05-10    30.934300   7.669422   rainy
2020-05-11    22.421250   6.889682  cloudy
2020-05-12    13.908200  19.012990   rainy
2020-05-13    23.938200  19.012990   sunny
```

```
import pandas as pd

# Sample data
data = {
    'date': ['2020-05-06', '2020-05-09', '2020-05-10', '2020-05-11',
'2020-05-12', '2020-05-13'],
    'temperature': [35.658200, 32.115275, 30.934300, 22.421250,
13.908200, 23.938200],
    'windSpeed': [10.788378, 8.449161, 7.669422, 6.889682, 19.012990,
19.012990],
    'status': ['sunny', None, 'rainy', 'cloudy', 'rainy', 'sunny']
}

# Convert to DataFrame and set 'date' as datetime index
df = pd.DataFrame(data)
df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace=True)

# Interpolate numeric columns and plot
(df.select_dtypes(include=['float64', 'int64'])  # Select only numeric
columns
   .interpolate(method='time')                  # Interpolate missing
values
   .plot(title='Time Plot', grid=1)          # Plot with title and
grid
)
```

```
<Axes: title={'center': 'Time Plot'}, xlabel='date'>
```



```
df1 = pd.DataFrame({'A':[2,3.3,np.nan,3.7,7.6,8.8],
                    'B':[1.25,np.nan,np.nan,4.3,14.23,16.4]})
df1

     A      B
0  2.0   1.25
1  3.3    NaN
2  NaN    NaN
3  3.7   4.30
4  7.6  14.23
5  8.8  16.40

df1.interpolate()

     A          B
0  2.0   1.250000
1  3.3   2.266667
2  3.5   3.283333
3  3.7   4.300000
4  7.6  14.230000
5  8.8  16.400000

df1.interpolate(method='barycentric')
```

```
      A      B
0  2.00   1.25
1  3.30  -9.52
2  2.23  -6.06
3  3.70   4.30
4  7.60  14.23
5  8.80  16.40
```

```
df1.interpolate(method= 'pchip')
```

```
          A          B
0  2.000000   1.250000
1  3.300000   1.566495
2  3.488632   2.560768
3  3.700000   4.300000
4  7.600000  14.230000
5  8.800000  16.400000
```

```
df1.interpolate(method='akima')
```

```
          A          B
0  2.000000   1.250000
1  3.300000  -0.772951
2  3.444216   0.175210
3  3.700000   4.300000
4  7.600000  14.230000
5  8.800000  16.400000
```

```
df1.interpolate(method='spline',order=2)
```

```
          A          B
0  2.000000   1.250000
1  3.300000  -0.855010
2  3.440909   0.547068
3  3.700000   4.300000
4  7.600000  14.230000
5  8.800000  16.400000
```

```
df1.interpolate(method='polynomial',order=2)
```

```
          A          B
0  2.000000   1.250000
1  3.300000  -3.129744
2  2.905405  -2.113077
3  3.700000   4.300000
4  7.600000  14.230000
5  8.800000  16.400000
```

```
df1
```

```
     A      B
0  2.0   1.25
1  3.3    NaN
2  NaN    NaN
3  3.7   4.30
4  7.6  14.23
5  8.8  16.40
```

```
df1.interpolate(limit=1)
```

```
     A          B
0  2.0   1.250000
1  3.3   2.266667
2  3.5        NaN
3  3.7   4.300000
4  7.6  14.230000
5  8.8  16.400000
```

```
df1.interpolate(limit=1,limit_direction='backward')
```

```
     A          B
0  2.0   1.250000
1  3.3        NaN
2  3.5   3.283333
3  3.7   4.300000
4  7.6  14.230000
5  8.8  16.400000
```

```
df1
```

```
     A      B
0  2.0   1.25
1  3.3    NaN
2  NaN    NaN
3  3.7   4.30
4  7.6  14.23
5  8.8  16.40
```

```
df1.interpolate(limit=1,limit_direction='both')
```

```
     A          B
0  2.0   1.250000
1  3.3   2.266667
2  3.5   3.283333
3  3.7   4.300000
4  7.6  14.230000
5  8.8  16.400000
```

```
dff = pd.Series([np.nan,np.nan,35,np.nan,np.nan,55,np.nan,np.nan])
dff
```

```
0     NaN
1     NaN
2    35.0
3     NaN
4     NaN
5    55.0
6     NaN
7     NaN
dtype: float64
```

```
dff.interpolate(limit_direction = 'both',limit_area =
'inside',limit=1)
```

```
0          NaN
1          NaN
2    35.000000
3    41.666667
4    48.333333
5    55.000000
6          NaN
7          NaN
dtype: float64
```

```
dff.interpolate(limit_direction='both',limit_area='outside',limit=1)
```

```
0     NaN
1    35.0
2    35.0
3     NaN
4     NaN
5    55.0
6    55.0
7     NaN
dtype: float64
```

```
dff.interpolate(limit_direction='both',limit_area='outside',limit=1)
```

```
0     NaN
1    35.0
2    35.0
3     NaN
4     NaN
5    55.0
6    55.0
7     NaN
dtype: float64
```

```
import pandas as pd

data = {
    'date': ["20200506", "20200509", "20200510", "20200511",
"20200512", "20200513"],  # Filled missing dates with adjacent days
    'temperature': [35.6582, np.nan, 30.9343, np.nan, 13.9082,
23.9382],  # Replaced NaN with approximate values
    'windSpeed': [10.788378, np.nan, np.nan, 6.889682, 19.012990,
np.nan],      # Replaced NaN with typical wind speeds
    'status': ["sunny",np.nan, "rainy", "cloudy", 'rainy', 'sunny']  #
Replaced NaN with plausible weather
}

df = pd.DataFrame(data)
print(df)
```

```
       date  temperature  windSpeed  status
0  20200506      35.6582  10.788378   sunny
1  20200509          NaN        NaN     NaN
2  20200510      30.9343        NaN   rainy
3  20200511          NaN   6.889682  cloudy
4  20200512      13.9082  19.012990   rainy
5  20200513      23.9382        NaN   sunny
```

```
df['date'] = pd.to_datetime(df['date'], format='%Y%m%d')
df
```

```
        date  temperature  windSpeed  status
0 2020-05-06      35.6582  10.788378   sunny
1 2020-05-09          NaN        NaN     NaN
2 2020-05-10      30.9343        NaN   rainy
3 2020-05-11          NaN   6.889682  cloudy
4 2020-05-12      13.9082  19.012990   rainy
5 2020-05-13      23.9382        NaN   sunny
```

```
df=df.set_index('date')
df
```

```
            temperature  windSpeed  status
date
2020-05-06      35.6582  10.788378   sunny
2020-05-09          NaN        NaN     NaN
2020-05-10      30.9343        NaN   rainy
2020-05-11          NaN   6.889682  cloudy
2020-05-12      13.9082  19.012990   rainy
2020-05-13      23.9382        NaN   sunny
```

```
df.dropna() #drops/removes row contain minimum 1 NaN
```

```
            temperature  windSpeed status
date
2020-05-06      35.6582  10.788378  sunny
2020-05-12      13.9082  19.012990  rainy
```

```
df.dropna(how='all') #date 9 is dropped where all values are NaN
```

```
            temperature  windSpeed  status
date
2020-05-06      35.6582  10.788378   sunny
2020-05-10      30.9343        NaN   rainy
2020-05-11          NaN   6.889682  cloudy
2020-05-12      13.9082  19.012990   rainy
2020-05-13      23.9382        NaN   sunny
```

```
df.dropna(thresh=1) #If minimum 1 data is present then they will not
remove,others will remove
```

```
            temperature  windSpeed  status
date
2020-05-06      35.6582  10.788378   sunny
2020-05-10      30.9343        NaN   rainy
2020-05-11          NaN   6.889682  cloudy
2020-05-12      13.9082  19.012990   rainy
2020-05-13      23.9382        NaN   sunny
```

```
df.dropna(thresh=3) #1/2 data containing row will remove/drop
```

```
            temperature  windSpeed status
date
2020-05-06      35.6582  10.788378  sunny
2020-05-12      13.9082  19.012990  rainy
```

```
ranks = ['a','b','c','d']
names = ['Raju','Ramu','Priya','Sneha']

dfnew = pd.DataFrame(list(zip(names,ranks)),
columns=['names','ranks'])
dfnew
```

```
   names ranks
0   Raju     a
1   Ramu     b
2  Priya     c
3  Sneha     d
```

```
dfnew = dfnew.replace(['a','b','c','d'],[1,2,3,4])
dfnew
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\769724644.py:1:
FutureWarning: Downcasting behavior in `replace` is deprecated and
will be removed in a future version. To retain the old behavior,
explicitly call `result.infer_objects(copy=False)`. To opt-in to the
future behavior, set `pd.set_option('future.no_silent_downcasting',
True)`
  dfnew = dfnew.replace(['a','b','c','d'],[1,2,3,4])
```

```
   names  ranks
0   Raju      1
1   Ramu      2
2  Priya      3
3  Sneha      4
```

```
dfnew.replace({3:10,1:100}) # Replace with mapping dictionary
```

```
   names  ranks
0   Raju    100
1   Ramu      2
2  Priya     10
3  Sneha      4
```

```
df_replace = pd.read_csv('weather_replace.csv')
df_replace
```

```
       date temperature       windSpeed  status
0  20200506   35.6582 c  10.788378 kmph   sunny
1  20200509          -1            xxxx       0
2  20200510   30.9343 c            xxxx   rainy
3  20200511          -1  6.8896825 kmph  cloudy
4  20200512   13.9082 c   19.01299 kmph   rainy
5  20200513           0              -1   sunny
```

```
df_replace.replace('-1',np.nan)
```

```
       date temperature       windSpeed  status
0  20200506   35.6582 c  10.788378 kmph   sunny
1  20200509         NaN            xxxx       0
2  20200510   30.9343 c            xxxx   rainy
3  20200511         NaN  6.8896825 kmph  cloudy
4  20200512   13.9082 c   19.01299 kmph   rainy
5  20200513           0             NaN   sunny
```

```
df_replace.replace({
    '-1': np.nan,
    'xxxx': np.nan})
```

```
       date temperature       windSpeed  status
0  20200506   35.6582 c  10.788378 kmph   sunny
1  20200509         NaN             NaN       0
2  20200510   30.9343 c             NaN   rainy
3  20200511         NaN  6.8896825 kmph  cloudy
4  20200512   13.9082 c   19.01299 kmph   rainy
5  20200513           0             NaN   sunny
```

```
df_replace.replace(['-1','xxxx'],np.nan)
```

```
       date temperature       windSpeed  status
0  20200506   35.6582 c  10.788378 kmph   sunny
```

```
1  20200509         NaN              NaN       0
2  20200510   30.9343 c              NaN   rainy
3  20200511         NaN  6.8896825 kmph  cloudy
4  20200512   13.9082 c   19.01299 kmph   rainy
5  20200513           0              NaN   sunny
```

```
df_replace.replace(['-1','xxxx','0'],np.nan) #0 in temperature was
valid
```

```
       date temperature       windSpeed  status
0  20200506   35.6582 c  10.788378 kmph   sunny
1  20200509         NaN              NaN     NaN
2  20200510   30.9343 c              NaN   rainy
3  20200511         NaN  6.8896825 kmph  cloudy
4  20200512   13.9082 c   19.01299 kmph   rainy
5  20200513         NaN              NaN   sunny
```

```
#Replacing data as per columns
df_new = df_replace.replace({'temperature':'-1','windSpeed':['xxxx','-
1'],'status':'0'},np.nan)
df_new
```

```
       date temperature       windSpeed  status
0  20200506   35.6582 c  10.788378 kmph   sunny
1  20200509         NaN              NaN     NaN
2  20200510   30.9343 c              NaN   rainy
3  20200511         NaN  6.8896825 kmph  cloudy
4  20200512   13.9082 c   19.01299 kmph   rainy
5  20200513           0              NaN   sunny
```

```
#Regex on specific columns
df_new.replace({
    'temperature':'[A-Za-z]',
    'windSpeed': '[A-Za-z]',
},'', regex=True)       #remove the word 'c' and 'kmph'
```

```
       date temperature   windSpeed  status
0  20200506    35.6582   10.788378    sunny
1  20200509         NaN         NaN     NaN
2  20200510    30.9343          NaN   rainy
3  20200511         NaN  6.8896825   cloudy
4  20200512    13.9082    19.01299    rainy
5  20200513           0         NaN   sunny
```

```
students_df = pd.read_csv('students.csv')
students_df
```

```
     name    subject  sem1  sem2
0   Nisha    Physics    88    91
1    Arun    Physics    92    95
2    Neha    Physics    78    81
```

```
3   Varun    Physics    60    63
4   Nisha  Chemistry    61    64
5    Arun  Chemistry    72    75
6    Neha  Chemistry    82    85
7   Varun  Chemistry    59    62
8   Nisha      Maths    70    73
9    Arun      Maths    48    51
10   Neha      Maths    83    86
11  Varun      Maths    63    66
12  Nisha    Biology    71    74
13   Arun    Biology    84    87
14   Neha    Biology    57    60
15  Varun    Biology    71    74
```

```
students = students_df.groupby(['name'])
students
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at
0x000001D1AF462540>
```

```
students.first()
```

```
       subject  sem1  sem2
name
Arun   Physics    92    95
Neha   Physics    78    81
Nisha  Physics    88    91
Varun  Physics    60    63
```

```
students.last()
```

```
       subject  sem1  sem2
name
Arun   Biology    84    87
Neha   Biology    57    60
Nisha  Biology    71    74
Varun  Biology    71    74
```

```
students['sem1'].min()
```

```
name
Arun     48
Neha     57
Nisha    61
Varun    59
Name: sem1, dtype: int64
```

```
students['sem1'].mean()
```

```
name
Arun     74.00
```

```
Neha     75.00
Nisha    72.50
Varun    63.25
Name: sem1, dtype: float64
```

```
students.groups # group's name position at index
```

```
{'Arun': [1, 5, 9, 13], 'Neha': [2, 6, 10, 14], 'Nisha': [0, 4, 8,
12], 'Varun': [3, 7, 11, 15]}
```

```
for student,students_df in students: #using for loop in 'students'---
>4 times
    print(student)                   #loop work in 'students_df'
    print(students_df)
```

```
('Arun',)
    name    subject  sem1  sem2
1   Arun    Physics    92    95
5   Arun  Chemistry    72    75
9   Arun      Maths    48    51
13  Arun    Biology    84    87
('Neha',)
    name    subject  sem1  sem2
2   Neha    Physics    78    81
6   Neha  Chemistry    82    85
10  Neha      Maths    83    86
14  Neha    Biology    57    60
('Nisha',)
     name    subject  sem1  sem2
0   Nisha    Physics    88    91
4   Nisha  Chemistry    61    64
8   Nisha      Maths    70    73
12  Nisha    Biology    71    74
('Varun',)
     name    subject  sem1  sem2
3   Varun    Physics    60    63
7   Varun  Chemistry    59    62
11  Varun      Maths    63    66
15  Varun    Biology    71    74
```

```
students.describe()
```

```
       sem1                                                       sem2
\
      count   mean        std   min    25%   50%    75%   max  count
mean
name

Arun    4.0  74.00  19.183326  48.0  66.00  78.0  86.00  92.0    4.0
77.00
Neha    4.0  75.00  12.192894  57.0  72.75  80.0  82.25  83.0    4.0
```

```
78.00
Nisha   4.0  72.50  11.269428  61.0  67.75  70.5  75.25  88.0   4.0
75.50
Varun   4.0  63.25   5.439056  59.0  59.75  61.5  65.00  71.0   4.0
66.25


             std   min    25%   50%    75%   max
name
Arun   19.183326  51.0  69.00  81.0  89.00  95.0
Neha   12.192894  60.0  75.75  83.0  85.25  86.0
Nisha  11.269428  64.0  70.75  73.5  78.25  91.0
Varun   5.439056  62.0  62.75  64.5  68.00  74.0
```

```
students_df = pd.read_csv('students.csv')
```

```
students_df.groupby(['name']).sum()
```

```
                            subject  sem1  sem2
name
Arun   PhysicsChemistryMathsBiology   296   308
Neha   PhysicsChemistryMathsBiology   300   312
Nisha  PhysicsChemistryMathsBiology   290   302
Varun  PhysicsChemistryMathsBiology   253   265
```

```
students_df.groupby(['name'],sort=False).sum()
```

```
                            subject  sem1  sem2
name
Nisha  PhysicsChemistryMathsBiology   290   302
Arun   PhysicsChemistryMathsBiology   296   308
Neha   PhysicsChemistryMathsBiology   300   312
Varun  PhysicsChemistryMathsBiology   253   265
```

```
dir(students)      # method is available
```

```
['_DataFrameGroupBy__examples_dataframe_doc',
 '__annotations__',
 '__class__',
 '__class_getitem__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattr__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__orig_bases__',
 '__parameters__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_accessors',
 '_agg_examples_doc',
 '_agg_general',
 '_agg_py_fallback',
 '_aggregate_frame',
 '_aggregate_with_numba',
 '_apply_filter',
 '_apply_to_column_groupbys',
 '_ascending_count',
 '_cache',
 '_choose_path',
 '_concat_objects',
 '_constructor',
 '_cumcount_array',
 '_cython_agg_general',
 '_cython_transform',
 '_define_paths',
 '_deprecate_axis',
 '_descending_count',
 '_dir_additions',
 '_dir_deletions',
 '_fill',
 '_get_data_to_aggregate',
 '_get_index',
 '_get_indices',
 '_gotitem',
 '_grouper',
 '_hidden_attrs',
 '_idxmax_idxmin',
 '_infer_selection',
 '_insert_inaxis_grouper',
 '_internal_names',
 '_internal_names_set',
 '_make_mask_from_int',
 '_make_mask_from_list',
 '_make_mask_from_positional_indexer',
 '_make_mask_from_slice',
 '_make_mask_from_tuple',
 '_mask_selected_obj',
 '_maybe_transpose_result',
 '_nth',
 '_numba_agg_general',
 '_numba_prep',
 '_obj_1d_constructor',
 '_obj_with_exclusions',
 '_op_via_apply',
 '_positional_selector',
 '_python_agg_general',
 '_python_apply_general',
 '_reindex_output',
 '_reset_cache',
 '_selected_obj',
 '_selection',
 '_selection_list',
 '_set_result_index_ordered',
 '_transform',
 '_transform_general',
 '_transform_with_numba',
 '_value_counts',
 '_wrap_agged_manager',
 '_wrap_aggregated_output',
 '_wrap_applied_output',
 '_wrap_applied_output_series',
 '_wrap_idxmax_idxmin',
 '_wrap_transform_fast_result',
 'agg',
 'aggregate',
 'all',
 'any',
 'apply',
 'bfill',
 'boxplot',
 'corr',
 'corrwith',
 'count',
 'cov',
 'cumcount',
 'cummax',
 'cummin',
 'cumprod',
 'cumsum',
 'describe',
 'diff',
 'dtypes',
 'ewm',
 'expanding',
 'ffill',
 'fillna',
 'filter',
 'first',
 'get_group',
 'groups',
 'head',
 'hist',
 'idxmax',
 'idxmin',
 'indices',
 'last',
 'max',
 'mean',
 'median',
 'min',
 'name',
 'ndim',
 'ngroup',
 'ngroups',
 'nth',
 'nunique',
 'ohlc',
 'pct_change',
 'pipe',
 'plot',
 'prod',
 'quantile',
 'rank',
 'resample',
 'rolling',
 'sample',
 'sem',
 'sem1',
 'sem2',
 'shift',
 'size',
 'skew',
 'std',
 'subject',
 'sum',
 'tail',
 'take',
 'transform',
 'value_counts',
 'var']
```

```
len(students) # i.e number of groups
```

```
4
```

```
len(students.get_group('Arun'))     #length of each group element
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\2600640423.py:1:
FutureWarning: When grouping with a length-1 list-like, you will need
to pass a length-1 tuple to get_group in a future version of pandas.
Pass `(name,)` instead of `name` to silence this warning.
  len(students.get_group('Arun'))     #length of each group element
```

```
4
```

```
students_df.sort_values(by=['name'],inplace=True)
students_df.set_index(['name','subject'],inplace=True)
students_df
```

```
                 sem1  sem2
name  subject
Arun  Physics      92    95
      Chemistry    72    75
      Maths        48    51
      Biology      84    87
Neha  Physics      78    81
      Chemistry    82    85
      Maths        83    86
      Biology      57    60
Nisha Physics      88    91
      Chemistry    61    64
      Maths        70    73
      Biology      71    74
Varun Physics      60    63
      Chemistry    59    62
      Maths        63    66
      Biology      71    74
```

```
grouped = students_df.groupby(level=0)   # here index is 'name'
grouped.sum()
```

```
       sem1  sem2
name
Arun    296   308
Neha    300   312
```

```
Nisha   290   302
Varun   253   265
```

```
grouped = students_df.groupby(level=1)  # here index is 'subject'
grouped.sum()
```

```
           sem1  sem2
subject
Biology     283   295
Chemistry   274   286
Maths       264   276
Physics     318   330
```

```
grouped = students_df.groupby(level = 'name')
grouped.sum()
```

```
       sem1  sem2
name
Arun    296   308
Neha    300   312
Nisha   290   302
Varun   253   265
```

```
grouped = students_df.groupby(level = 'subject')
grouped.sum()
```

```
           sem1  sem2
subject
Biology     283   295
Chemistry   274   286
Maths       264   276
Physics     318   330
```

```
students_df
```

```
                 sem1  sem2
name  subject
Arun  Physics      92    95
      Chemistry    72    75
      Maths        48    51
      Biology      84    87
Neha  Physics      78    81
      Chemistry    82    85
      Maths        83    86
      Biology      57    60
Nisha Physics      88    91
      Chemistry    61    64
      Maths        70    73
      Biology      71    74
Varun Physics      60    63
      Chemistry    59    62
      Maths        63    66
      Biology      71    74
```

```
#  Grouping DataFrame with index level and columns
students_df.groupby([pd.Grouper(level = 'name'), 'sem1']).sum()
```

```
            sem2
name  sem1
Arun  48      51
      72      75
      84      87
      92      95
Neha  57      60
      78      81
      82      85
      83      86
Nisha 61      64
      70      73
      71      74
      88      91
Varun 59      62
      60      63
      63      66
      71      74
```

```
students_df.groupby(['name','sem1']).sum()
```

```
            sem2
name  sem1
Arun  48      51
      72      75
      84      87
      92      95
Neha  57      60
      78      81
      82      85
      83      86
Nisha 61      64
      70      73
      71      74
      88      91
Varun 59      62
      60      63
      63      66
      71      74
```

```
students.size() #Size of each group
```

```
name
Arun     4
Neha     4
```

```
Nisha    4
Varun    4
dtype: int64
```

```
students.aggregate(np.size) #SIze of each group column-wise
```

```
       subject  sem1  sem2
name                      
Arun         4     4     4
Neha         4     4     4
Nisha        4     4     4
Varun        4     4     4
```

```
students[['sem1', 'sem2']].mean()
```

```
        sem1   sem2
name               
Arun   74.00  77.00
Neha   75.00  78.00
Nisha  72.50  75.50
Varun  63.25  66.25
```

```
students[['sem1', 'sem2']].agg(['mean','max'])
```

```
        sem1        sem2    
        mean max    mean max
name                        
Arun   74.00  92  77.00  95
Neha   75.00  83  78.00  86
Nisha  72.50  88  75.50  91
Varun  63.25  71  66.25  74
```

```
students['sem1'].agg([np.sum, np.mean, np.std])
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\700931850.py:1:
FutureWarning: The provided callable <function sum at
0x000001D1A9D6F920> is currently using SeriesGroupBy.sum. In a future
version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "sum" instead.
  students['sem1'].agg([np.sum, np.mean, np.std])
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\700931850.py:1:
FutureWarning: The provided callable <function mean at
0x000001D1A9D90A40> is currently using SeriesGroupBy.mean. In a future
version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "mean" instead.
  students['sem1'].agg([np.sum, np.mean, np.std])
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\700931850.py:1:
FutureWarning: The provided callable <function std at
0x000001D1A9D90B80> is currently using SeriesGroupBy.std. In a future
version of pandas, the provided callable will be used directly. To
```

```
keep current behavior pass the string "std" instead.
  students['sem1'].agg([np.sum, np.mean, np.std])
```

```
       sum   mean        std
name
Arun   296  74.00  19.183326
Neha   300  75.00  12.192894
Nisha  290  72.50  11.269428
Varun  253  63.25   5.439056
```

```
#  Renaming the column names for aggregate functions
students['sem1'].agg([np.sum,np.mean,np.std]).rename(columns={
    'sum': 'total',
    'mean': 'average',
    'std': 'standardDeviation'
})
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\3735640168.py:2:
FutureWarning: The provided callable <function sum at
0x000001D1A9D6F920> is currently using SeriesGroupBy.sum. In a future
version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "sum" instead.
  students['sem1'].agg([np.sum,np.mean,np.std]).rename(columns={
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\3735640168.py:2:
FutureWarning: The provided callable <function mean at
0x000001D1A9D90A40> is currently using SeriesGroupBy.mean. In a future
version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "mean" instead.
  students['sem1'].agg([np.sum,np.mean,np.std]).rename(columns={
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\3735640168.py:2:
FutureWarning: The provided callable <function std at
0x000001D1A9D90B80> is currently using SeriesGroupBy.std. In a future
version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "std" instead.
  students['sem1'].agg([np.sum,np.mean,np.std]).rename(columns={
```

```
       total  average  standardDeviation
name
Arun     296    74.00          19.183326
Neha     300    75.00          12.192894
Nisha    290    72.50          11.269428
Varun    253    63.25           5.439056
```

```
# apply aggregated functions to different columns
students.agg(
    sem1_min_marks=pd.NamedAgg(column='sem1', aggfunc='min'),
    sem2_max_marks=pd.NamedAgg(column='sem2', aggfunc='max'),
    sem1_avg_marks=pd.NamedAgg(column='sem1', aggfunc='mean'),
    sem2_avg_marks=pd.NamedAgg(column='sem2', aggfunc='mean'),
)
```

```
       sem1_min_marks  sem2_max_marks  sem1_avg_marks  sem2_avg_marks
name
Arun                48              95           74.00           77.00
Neha                57              86           75.00           78.00
Nisha               61              91           72.50           75.50
Varun               59              74           63.25           66.25
```

```
students.agg({
    'sem1' :'sum',
    'sem2' :lambda x: np.std(x,ddof=1)
})
```

```
       sem1       sem2
name
Arun    296  19.183326
Neha    300  12.192894
Nisha   290  11.269428
Varun   253   5.439056
```

```
students_df = pd.read_csv('students.csv')
```

```
students2_df =students_df.reset_index()
students2 = students2_df.groupby('name')
```

```
students2.mean(['sem1','sem2'])
```

```
       index   sem1   sem2
name
Arun     7.0  74.00  77.00
Neha     8.0  75.00  78.00
Nisha    6.0  72.50  75.50
Varun    9.0  63.25  66.25
```

```
# TRANSFORMATION
students2[['sem1', 'sem2']].transform(np.mean) #16 rows which is same
as original data
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\492706235.py:2:
FutureWarning: The provided callable <function mean at
0x000001D1A9D90A40> is currently using DataFrameGroupBy.mean. In a
future version of pandas, the provided callable will be used directly.
To keep current behavior pass the string "mean" instead.
  students2[['sem1', 'sem2']].transform(np.mean) #16 rows which is
same as original data
```

```
    sem1   sem2
0   72.50  75.50
1   74.00  77.00
2   75.00  78.00
3   63.25  66.25
4   72.50  75.50
```

```
5   74.00  77.00
6   75.00  78.00
7   63.25  66.25
8   72.50  75.50
9   74.00  77.00
10  75.00  78.00
11  63.25  66.25
12  72.50  75.50
13  74.00  77.00
14  75.00  78.00
15  63.25  66.25
```

```
score =lambda x: (x-x.mean())/x.std()*10
score2 = lambda x: (x.max()-x.min())
```

```
score3 = lambda x: x.fillna(x.mean())
students2[['sem1','sem2']].transform(score3)
```

```
    sem1  sem2
0     88    91
1     92    95
2     78    81
3     60    63
4     61    64
5     72    75
6     82    85
7     59    62
8     70    73
9     48    51
10    83    86
11    63    66
12    71    74
13    84    87
14    57    60
15    71    74
```

```
students2[['sem1','sem2']].transform(score2)
```

```
    sem1  sem2
0     27    27
1     44    44
2     26    26
3     12    12
4     27    27
5     44    44
6     26    26
7     12    12
8     27    27
9     44    44
10    26    26
```

```
11    12    12
12    27    27
13    44    44
14    26    26
15    12    12
```

```
students2_df[['name','sem1','sem2']].groupby('name').apply(print)
```

```
    name  sem1  sem2
1   Arun    92    95
5   Arun    72    75
9   Arun    48    51
13  Arun    84    87
    name  sem1  sem2
2   Neha    78    81
6   Neha    82    85
10  Neha    83    86
14  Neha    57    60
     name  sem1  sem2
0   Nisha    88    91
4   Nisha    61    64
8   Nisha    70    73
12  Nisha    71    74
     name  sem1  sem2
3   Varun    60    63
7   Varun    59    62
11  Varun    63    66
15  Varun    71    74
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\3555616080.py:1:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping
columns. This behavior is deprecated, and in a future version of
pandas the grouping columns will be excluded from the operation.
Either pass `include_groups=False` to exclude the groupings or
explicitly select the grouping columns after groupby to silence this
warning.
  students2_df[['name','sem1','sem2']].groupby('name').apply(print)
```

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

```
#Rolling
```

```
students2_df.groupby('name').rolling(3).sem1.sum() #For each student,
it looks at the current exam and the previous 2 exams. rolling(3) adds
                                                    # row1+row2+row3
```

```
name
Arun   1       NaN
       5       NaN
       9     212.0
       13    204.0
Neha   2       NaN
       6       NaN
       10    243.0
       14    222.0
Nisha  0       NaN
       4       NaN
       8     219.0
       12    202.0
Varun  3       NaN
       7       NaN
       11    182.0
       15    193.0
Name: sem1, dtype: float64
```

```
# expanding

students2_df.groupby('name')[['sem1','sem2']].expanding().sum() #sum
all row for previous row
```

```
            sem1   sem2
name
Arun  1     92.0   95.0
      5    164.0  170.0
      9    212.0  221.0
      13   296.0  308.0
Neha  2     78.0   81.0
      6    160.0  166.0
      10   243.0  252.0
      14   300.0  312.0
Nisha 0     88.0   91.0
      4    149.0  155.0
      8    219.0  228.0
      12   290.0  302.0
Varun 3     60.0   63.0
      7    119.0  125.0
      11   182.0  191.0
      15   253.0  265.0
```

```
students_df.set_index('name')
```

```
          subject  sem1  sem2
name
Nisha     Physics    88    91
Arun      Physics    92    95
Neha      Physics    78    81
Varun     Physics    60    63
Nisha   Chemistry    61    64
Arun    Chemistry    72    75
```

```
Neha   Chemistry    82    85
Varun  Chemistry    59    62
Nisha      Maths    70    73
Arun       Maths    48    51
Neha       Maths    83    86
Varun      Maths    63    66
Nisha    Biology    71    74
Arun     Biology    84    87
Neha     Biology    57    60
Varun    Biology    71    74
```

```
students_df.groupby('name').filter(lambda x: any('run' in name for
name in x['name'].unique())) # Filtering
```

```
     name    subject  sem1  sem2
1    Arun    Physics    92    95
3   Varun    Physics    60    63
5    Arun  Chemistry    72    75
7   Varun  Chemistry    59    62
9    Arun      Maths    48    51
11  Varun      Maths    63    66
13   Arun    Biology    84    87
15  Varun    Biology    71    74
```

```
filtering = pd.Series([10, 11, 12, 13, 14, 15])

result=filtering.groupby(filtering).filter(lambda x: x.sum() >12)
#.sum() lets you evaluate the entire group at once.
display(result)
```

```
3    13
4    14
5    15
dtype: int64
```

```
students_df = pd.read_csv('students.csv')
students_df.set_index('name')
```

```
         subject  sem1  sem2
name
Nisha    Physics    88    91
Arun     Physics    92    95
Neha     Physics    78    81
Varun    Physics    60    63
Nisha  Chemistry    61    64
Arun   Chemistry    72    75
Neha   Chemistry    82    85
Varun  Chemistry    59    62
Nisha      Maths    70    73
Arun       Maths    48    51
Neha       Maths    83    86
```

```
Varun      Maths    63    66
Nisha    Biology    71    74
Arun     Biology    84    87
Neha     Biology    57    60
Varun    Biology    71    74
```

```
students_df.groupby(['name'])
students_df.set_index('name')
```

```
         subject  sem1  sem2
name
Nisha    Physics    88    91
Arun     Physics    92    95
Neha     Physics    78    81
Varun    Physics    60    63
Nisha  Chemistry    61    64
Arun   Chemistry    72    75
Neha   Chemistry    82    85
Varun  Chemistry    59    62
Nisha      Maths    70    73
Arun       Maths    48    51
Neha       Maths    83    86
Varun      Maths    63    66
Nisha    Biology    71    74
Arun     Biology    84    87
Neha     Biology    57    60
Varun    Biology    71    74
```

```
students_df[['sem1','sem2']].sum()
```

```
sem1    1139
sem2    1187
dtype: int64
```

```
students[['sem1', 'sem2']].sum()
```

```
       sem1  sem2
name
Arun    296   308
Neha    300   312
Nisha   290   302
Varun   253   265
```

```
students_df.set_index(['name','subject'],inplace=True)
```

```
students_df
```

```
                 sem1  sem2
name  subject
Nisha Physics      88    91
Arun  Physics      92    95
```

```
Neha  Physics      78    81
Varun Physics      60    63
Nisha Chemistry    61    64
Arun  Chemistry    72    75
Neha  Chemistry    82    85
Varun Chemistry    59    62
Nisha Maths        70    73
Arun  Maths        48    51
Neha  Maths        83    86
Varun Maths        63    66
Nisha Biology      71    74
Arun  Biology      84    87
Neha  Biology      57    60
Varun Biology      71    74
```

```
students_df.fillna(method='ffill')
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\717292758.py:1:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will
raise in a future version. Use obj.ffill() or obj.bfill() instead.
  students_df.fillna(method='ffill')
```

```
                 sem1  sem2
name  subject
Nisha Physics      88    91
Arun  Physics      92    95
Neha  Physics      78    81
Varun Physics      60    63
Nisha Chemistry    61    64
Arun  Chemistry    72    75
Neha  Chemistry    82    85
Varun Chemistry    59    62
Nisha Maths        70    73
Arun  Maths        48    51
Neha  Maths        83    86
Varun Maths        63    66
Nisha Biology      71    74
Arun  Biology      84    87
Neha  Biology      57    60
Varun Biology      71    74
```

```
# Fetching ,  We can fetch the nth row of each groups
students_df.groupby('name').nth(1)
```

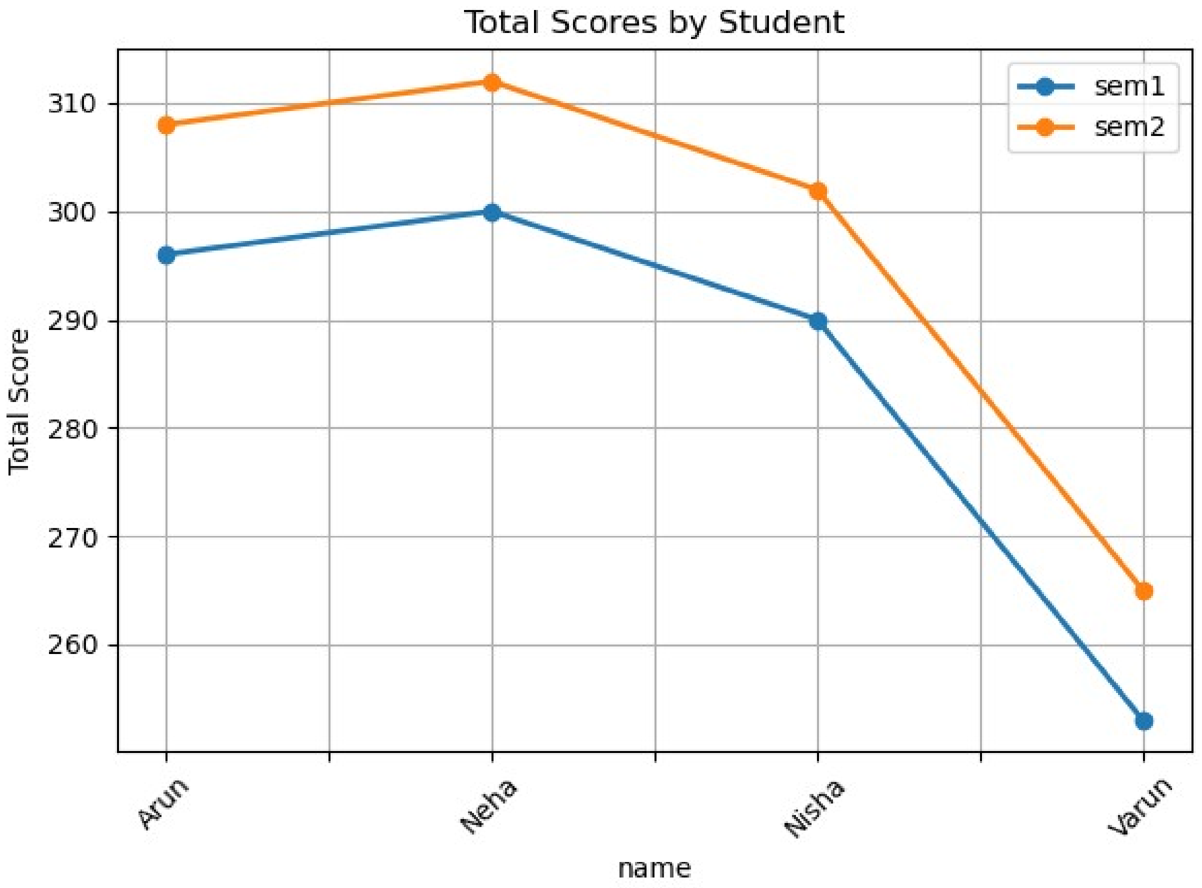
```
                 sem1  sem2
name  subject
Nisha Chemistry    61    64
Arun  Chemistry    72    75
Neha  Chemistry    82    85
Varun Chemistry    59    62
```

```python
def f(group):
    return pd.DataFrame({
        'original': group,
        'reduced': group - group.mean()
    })

# Proper way to apply to groups
result = students_df.groupby('name')['sem1'].apply(f)
print(result)
```

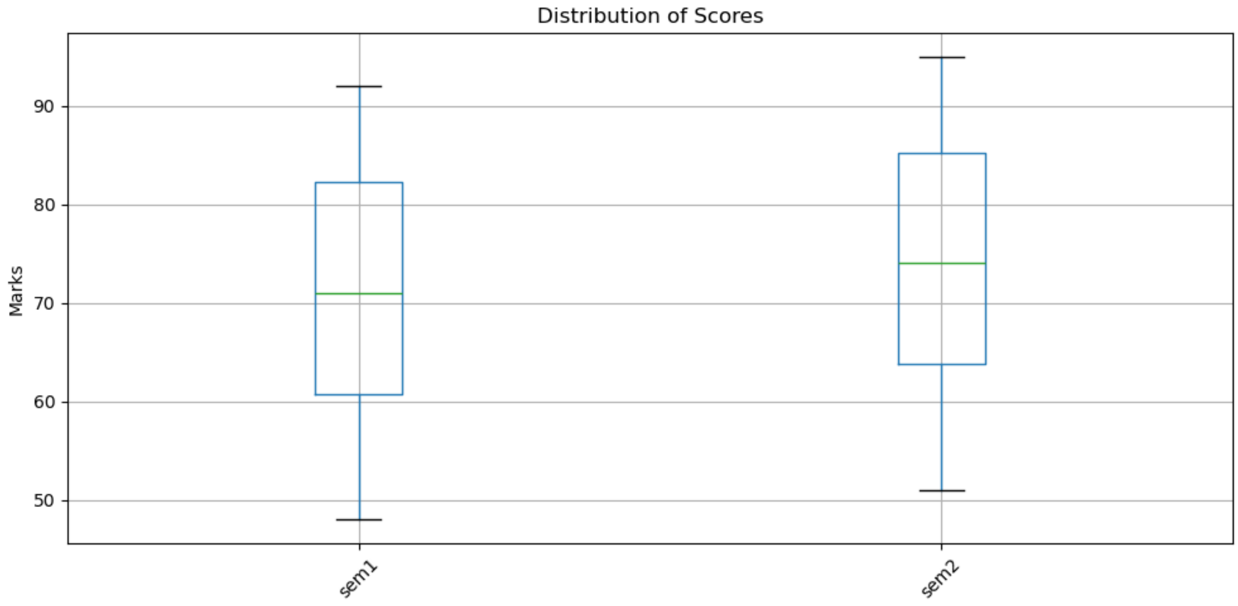
```
                        original  reduced
name  name  subject
Arun  Arun  Physics           92    18.00
            Chemistry         72    -2.00
            Maths             48   -26.00
            Biology           84    10.00
Neha  Neha  Physics           78     3.00
            Chemistry         82     7.00
            Maths             83     8.00
            Biology           57   -18.00
Nisha Nisha Physics           88    15.50
            Chemistry         61   -11.50
            Maths             70    -2.50
            Biology           71    -1.50
Varun Varun Physics           60    -3.25
            Chemistry         59    -4.25
            Maths             63    -0.25
            Biology           71     7.75
```

```python
import matplotlib.pyplot as plt
%matplotlib inline

# Create figure with appropriate size
plt.figure(figsize=(10, 4))

# Group by name and sum all numeric columns, then plot as line graph
students_df.groupby('name').sum().plot(
    kind='line',  # *** Changed from 'bar' to 'line' ***
    grid=True,
    rot=45,
    title='Total Scores by Student',
    ylabel='Total Score',
    marker='o',  # Add markers for each data point
    linestyle='-',  # Solid line
    linewidth=2  # Thicker line
)

# Adjust layout
plt.tight_layout()
plt.show()
```

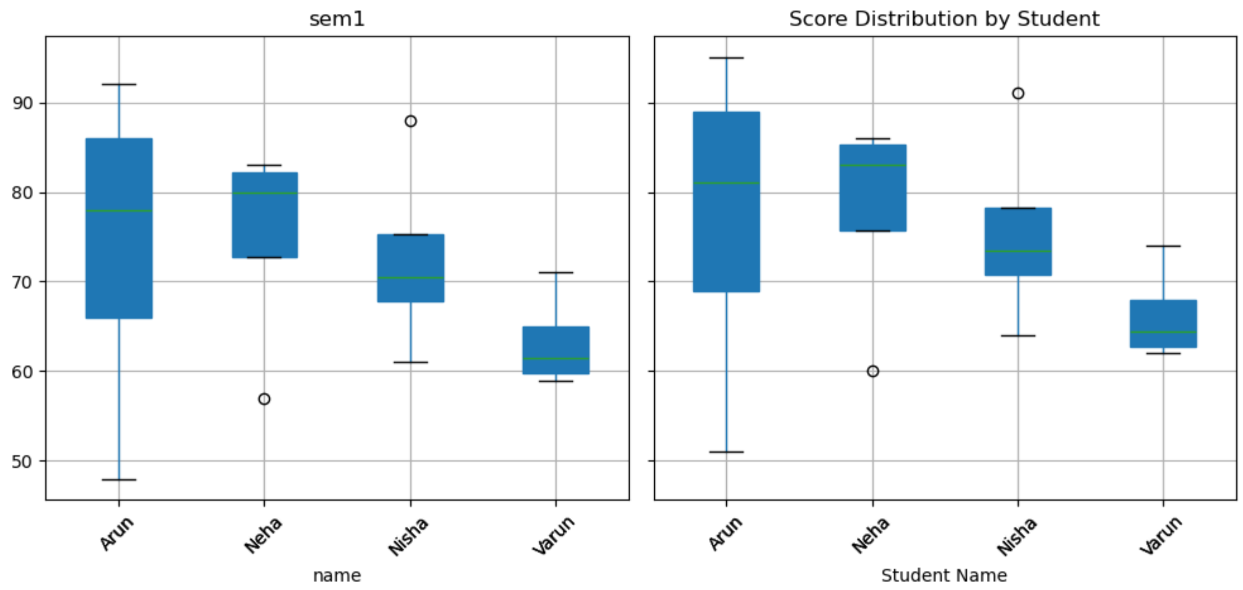
```
<Figure size 1000x400 with 0 Axes>
```



```
import matplotlib.pyplot as plt
%matplotlib inline

# Simple boxplot of all numeric columns
students_df.boxplot(figsize=(10, 5), grid=True)
plt.title('Distribution of Scores')
plt.ylabel('Marks')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



```python
import matplotlib.pyplot as plt
%matplotlib inline

# Boxplot of scores grouped by student name
students_df.boxplot(
    column=['sem1', 'sem2'],  # Columns to plot
    by='name',                # Grouping variable
    figsize=(10, 5),          # Figure size
    grid=True,
    patch_artist=True,        # Color the boxes
    rot=45                    # Rotate labels
)

plt.title('Score Distribution by Student')
plt.suptitle('')  # Remove default subtitle
plt.xlabel('Student Name')
plt.ylabel('Score')
plt.tight_layout()
plt.show()
```



```
import pandas as pd
import numpy as np

arun = pd.Series(['a','r','u','n'])
neha = pd.Series(['n','e','h','a'])
pd.concat([arun,neha])
```

```
0    a
1    r
2    u
3    n
0    n
1    e
2    h
3    a
dtype: object
```

```
arun_scores = {
    'subjects':['maths','physics','chemistry','biology'],
    'sem1':[60,70,80,90],
    'sem2':[63,71,85,89]
}
neha_scores ={
    'subjects':['maths','physics','chemistry','computers'],
    'sem1':[60,70,80,90],
    'sem2':[63,77,89,92]
}

arun_df = pd.DataFrame(arun_scores)
arun_df
```

```
    subjects  sem1  sem2
0      maths    60    63
1    physics    70    71
2  chemistry    80    85
3    biology    90    89
```

```python
neha_df = pd.DataFrame(arun_scores)
neha_df
```

```
    subjects  sem1  sem2
0      maths    60    63
1    physics    70    71
2  chemistry    80    85
3    biology    90    89
```

```python
df = pd.concat([arun_df,neha_df])
df
```

```
    subjects  sem1  sem2
0      maths    60    63
1    physics    70    71
2  chemistry    80    85
3    biology    90    89
0      maths    60    63
1    physics    70    71
2  chemistry    80    85
3    biology    90    89
```

```python
df = pd.concat([arun_df, neha_df], ignore_index = True)
df #  The duplicate index are now gone as we ignored
```

```
    subjects  sem1  sem2
0      maths    60    63
1    physics    70    71
2  chemistry    80    85
3    biology    90    89
4      maths    60    63
5    physics    70    71
6  chemistry    80    85
7    biology    90    89
```

```python
# Adding keys to DataFrames
df = pd.concat([arun_df, neha_df], keys = ['arun','neha'])
df
```

```
         subjects  sem1  sem2
arun 0      maths    60    63
     1    physics    70    71
     2  chemistry    80    85
     3    biology    90    89
neha 0      maths    60    63
```

```
      1    physics    70    71
      2  chemistry    80    85
      3    biology    90    89

df.loc['arun'] # access keys

    subjects  sem1  sem2
0      maths    60    63
1    physics    70    71
2  chemistry    80    85
3    biology    90    89

arun_sem3_scores = {
    'subjects': ['maths','physics','chemistry','biology'],
    'sem3':[50, 64, 88, 81]
}
df_additional = pd.DataFrame(arun_sem3_scores)
df_additional

    subjects  sem3
0      maths    50
1    physics    64
2  chemistry    88
3    biology    81

df = pd.concat([arun_df, df_additional])
df

    subjects  sem1  sem2  sem3
0      maths  60.0  63.0   NaN
1    physics  70.0  71.0   NaN
2  chemistry  80.0  85.0   NaN
3    biology  90.0  89.0   NaN
0      maths   NaN   NaN  50.0
1    physics   NaN   NaN  64.0
2  chemistry   NaN   NaN  88.0
3    biology   NaN   NaN  81.0

df = pd.concat([arun_df, df_additional], axis=1) #  same data to be a
new column
df

    subjects  sem1  sem2   subjects  sem3
0      maths    60    63      maths    50
1    physics    70    71    physics    64
2  chemistry    80    85  chemistry    88
3    biology    90    89    biology    81

pd.concat([arun , neha],axis=1)
```

```
   0  1
0  a  n
1  r  e
2  u  h
3  n  a
```

```
#  Rearranging the order of column
arun_sem3_scores = {
    'subjects':['physics', 'chemistry', 'maths', 'biology'],
    'sem3':[50, 64, 88, 81]
}
df_additional = pd.DataFrame(arun_sem3_scores)
df_additional
```

```
    subjects  sem3
0    physics    50
1  chemistry    64
2      maths    88
3    biology    81
```

```
df = pd.concat([arun_df, df_additional], axis=1)
df
```

```
    subjects  sem1  sem2   subjects  sem3
0      maths    60    63    physics    50
1    physics    70    71  chemistry    64
2  chemistry    80    85      maths    88
3    biology    90    89    biology    81
```

```
arun_sem3_scores = {
    'subjects':['physics', 'chemistry', 'maths', 'biology'],
    'sem3':[50, 64, 88, 81]
}
df_additional = pd.DataFrame(arun_sem3_scores, index=[1,2,0,3])
df_additional
```

```
    subjects  sem3
1    physics    50
2  chemistry    64
0      maths    88
3    biology    81
```

```
arun_df
```

```
    subjects  sem1  sem2
0      maths    60    63
1    physics    70    71
2  chemistry    80    85
3    biology    90    89
```

```
df_additional
```

```
    subjects  sem3
1    physics    50
2  chemistry    64
0      maths    88
3    biology    81
```

```
arun_sem3_scores = {
    'subjects':['physics', 'chemistry', 'maths', 'biology'],
    'sem3':[50, 64, 88, 81]
}
df_additional = pd.DataFrame(arun_sem3_scores, index=[1,2,0,3]) #
maths changed to index 0,physics---->1
df_additional
```

```
    subjects  sem3
1    physics    50
2  chemistry    64
0      maths    88
3    biology    81
```

```
df = pd.concat([arun_df, df_additional],axis=1)
df
```

```
    subjects  sem1  sem2   subjects  sem3
0      maths    60    63      maths    88
1    physics    70    71    physics    50
2  chemistry    80    85  chemistry    64
3    biology    90    89    biology    81
```

```
s = pd.Series([88, 76, 74, 72], name='sem4')
s
```

```
0    88
1    76
2    74
3    72
Name: sem4, dtype: int64
```

```
df = pd.concat([arun_df, s], axis=1)
df
```

```
    subjects  sem1  sem2  sem4
0      maths    60    63    88
1    physics    70    71    76
2  chemistry    80    85    74
3    biology    90    89    72
```

```
# Concatenating multiple DataFrames /series
df = pd.concat([arun_df, df_additional['sem3'],s],axis=1)
df
```

```
    subjects  sem1  sem2  sem3  sem4
0      maths    60    63    88    88
1    physics    70    71    50    76
2  chemistry    80    85    64    74
3    biology    90    89    81    72
```

```
arun_sem1_scores = {
    'subjects':['maths','physics','chemistry','biology'],
    'sem1':[60,70,80,90],

}
sem1_df = pd.DataFrame(arun_sem1_scores)
sem1_df
```

```
    subjects  sem1
0      maths    60
1    physics    70
2  chemistry    80
3    biology    90
```

```
arun_sem2_scores = {
    'subjects':['physics','chemistry','maths','biology'],
    'sem2':[73,81,88,83],

}
sem2_df = pd.DataFrame(arun_sem2_scores)
sem2_df
```

```
    subjects  sem2
0    physics    73
1  chemistry    81
2      maths    88
3    biology    83
```

```
# merge() combines the DataFrames on the basis of values  of common
columns whereas
# concat() just appends the DataFrames

df = pd.merge(sem1_df, sem2_df, on='subjects')
df
```

```
    subjects  sem1  sem2
0      maths    60    88
1    physics    70    73
2  chemistry    80    81
3    biology    90    83
```

```
arun_sem1_scores = {
    'subjects':['maths','physics','chemistry', 'litrature'],
    'sem1':[60,70,80,55],
}
sem1_df = pd.DataFrame(arun_sem1_scores)
sem1_df
```

```
    subjects  sem1
0      maths    60
1    physics    70
2  chemistry    80
3  litrature    55
```

```
arun_sem2_scores = {
    "subjects": ['physics', 'chemistry', 'maths', 'biology',
'computers'],
    "sem2":[73, 81, 88, 83, 88]
}
sem2_df = pd.DataFrame(arun_sem2_scores)
sem2_df
```

```
    subjects  sem2
0    physics    73
1  chemistry    81
2      maths    88
3    biology    83
4  computers    88
```

```
df = pd.merge(sem1_df,sem2_df, on= 'subjects') # inner join:This
provides the result common to both the DataFrames.
df
```

```
    subjects  sem1  sem2
0      maths    60    88
1    physics    70    73
2  chemistry    80    81
```

```
#Merging with outer join(all subjects)
df = pd.merge(sem1_df, sem2_df, on='subjects', how= 'outer')
df
```

```
    subjects  sem1  sem2
0    biology   NaN  83.0
1  chemistry  80.0  81.0
2  computers   NaN  88.0
3  litrature  55.0   NaN
4      maths  60.0  88.0
5    physics  70.0  73.0
```

```
sem1_df
```

```
    subjects  sem1
0      maths    60
1    physics    70
```

```
2  chemistry    80
3  litrature    55

sem2_df

    subjects  sem2
0    physics    73
1  chemistry    81
2      maths    88
3    biology    83
4  computers    88

#    Merging with left join ( one dataframe + common value from both
dataframe)

df = pd.merge(sem1_df, sem2_df, on= 'subjects',how = 'left')
df

    subjects  sem1  sem2
0      maths    60  88.0
1    physics    70  73.0
2  chemistry    80  81.0
3  litrature    55   NaN

#     Merging with right join,  all the rows (subjects) from the right
DataFrame (sem2_df)

df = pd.merge(sem1_df, sem2_df, on= 'subjects',how = 'right')
df

    subjects  sem1  sem2
0    physics  70.0    73
1  chemistry  80.0    81
2      maths  60.0    88
3    biology   NaN    83
4  computers   NaN    88

# Knowing the source DataFrame after merge [Uses of indicator]
df = pd.merge(sem1_df, sem2_df, on='subjects', how='outer', indicator=
True)
df

    subjects  sem1  sem2      _merge
0    biology   NaN  83.0  right_only
1  chemistry  80.0  81.0        both
2  computers   NaN  88.0  right_only
3  litrature  55.0   NaN   left_only
4      maths  60.0  88.0        both
5    physics  70.0  73.0        both
```

```
neha_sem1_scores= {
    'subjects' : ['maths','physics','chemistry','computers'],
    'sem1': [65,75,83,80]
}
neha_sem1_df = pd.DataFrame(neha_sem1_scores)
neha_sem1_df
```

```
    subjects  sem1
0      maths    65
1    physics    75
2  chemistry    83
3  computers    80
```

```
arun_sem1_scores = {
    'subjects':['maths','physics','chemistry', 'litrature'],
    'sem1':[60,70,80,55],

}
sem1_df = pd.DataFrame(arun_sem1_scores)
sem1_df
```

```
    subjects  sem1
0      maths    60
1    physics    70
2  chemistry    80
3  litrature    55
```

```
df = pd.merge(sem1_df, neha_sem1_df, on= 'subjects', how='outer')
df #Same column which is 'sem1' will automatically have _x and _y
```

```
    subjects  sem1_x  sem1_y
0  chemistry    80.0    83.0
1  computers     NaN    80.0
2  litrature    55.0     NaN
3      maths    60.0    65.0
4    physics    70.0    75.0
```

```
# custom suffixes instead of _x and _y
df = pd.merge(sem1_df, neha_sem1_df, on= 'subjects', how='outer',
suffixes=('_arun','_neha'))
df
```

```
    subjects  sem1_arun  sem1_neha
0  chemistry       80.0       83.0
1  computers        NaN       80.0
2  litrature       55.0        NaN
3      maths       60.0       65.0
4    physics       70.0       75.0
```

```
sem1_df = sem1_df.set_index('subjects')
sem1_df
```

```
           sem1
subjects
maths        60
physics      70
chemistry    80
litrature    55

sem2_df = sem2_df.set_index('subjects')
sem2_df

           sem2
subjects
physics      73
chemistry    81
maths        88
biology      83
computers    88

# Join method
sem1_df.join(sem2_df, how='outer') # by default used inner join

           sem1  sem2
subjects
biology     NaN  83.0
chemistry  80.0  81.0
computers   NaN  88.0
litrature  55.0   NaN
maths      60.0  88.0
physics    70.0  73.0

#  Append is a shortcut to concat()
# These concat only along axix=0 i.e. only rows

# sem1_df.append(sem2_df)

#  Pivot

df = pd.read_csv('weather_pivot.csv')
df

       date     city  temperature  windspeed
0  01/03/20   mumbai           32          9
1  02/03/20   mumbai           35          8
2  03/03/20   mumbai           33          6
3  01/03/20    delhi           40          7
4  02/03/20    delhi           38          9
5  03/03/20    delhi           37          8
6  01/03/20  kolkata           35          9
7  02/03/20  kolkata           36          6
8  03/03/20  kolkata           35          7
```

```
type(df['date'][0]) # string format
```

```
str
```

```
# Change data to datetime format
df['date'] = pd.to_datetime(df['date'])
df
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\4021415496.py:2:
UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is
consistent and as-expected, please specify a format.
  df['date'] = pd.to_datetime(df['date'])
```

```
        date     city  temperature  windspeed
0 2020-01-03   mumbai           32          9
1 2020-02-03   mumbai           35          8
2 2020-03-03   mumbai           33          6
3 2020-01-03    delhi           40          7
4 2020-02-03    delhi           38          9
5 2020-03-03    delhi           37          8
6 2020-01-03  kolkata           35          9
7 2020-02-03  kolkata           36          6
8 2020-03-03  kolkata           35          7
```

```
type(df['date'][0]) # timestamp format
```

```
pandas._libs.tslibs.timestamps.Timestamp
```

```
# Multilevel columns
```

```
df.pivot(index='date', columns='city')  # temperature and windspeed
has become one level of column
                                          # whereas city became another
```

```
           temperature                windspeed
city             delhi kolkata mumbai     delhi kolkata mumbai
date
2020-01-03          40      35     32         7       9      9
2020-02-03          38      36     35         9       6      8
2020-03-03          37      35     33         8       7      6
```

```
# to select specific value (column)
```

```
df.pivot(index='date',columns='city',values = 'windspeed')
```

```
city        delhi  kolkata  mumbai
date
2020-01-03      7        9       9
2020-02-03      9        6       8
2020-03-03      8        7       6
```

```
# alternatively

df.pivot(index='date', columns = 'city')['windspeed']
```

```
city        delhi  kolkata  mumbai
date
2020-01-03      7        9       9
2020-02-03      9        6       8
2020-03-03      8        7       6
```

```
df = pd.DataFrame({
    'first':list('aabbcc'),
    'second':list('xxyyzz'),
    'third':[1,2,3,4,5,6]
})
df
```

```
  first second  third
0     a      x      1
1     a      x      2
2     b      y      3
3     b      y      4
4     c      z      5
5     c      z      6
```

```
# A valueError is raised if there are any duplicates.

# df.pivot(index='first',columns='second')
# ValueError: Index contains duplicate entries, cannot reshape

df = pd.read_csv('weather_pivotTable.csv')
df['date'] = pd.to_datetime(df['date'])
df
```

```
         date     city  temperature  windspeed     time
0  2020-01-03   mumbai           43          9  morning
1  2020-01-03   mumbai           42         11  evening
2  2020-01-03    delhi           40          8  morning
3  2020-01-03    delhi           42          8  evening
4  2020-01-03  kolkata           38          6  morning
5  2020-01-03  kolkata           37          8  evening
6  2019-01-12   mumbai           22         12  morning
7  2019-01-12   mumbai           20         10  evening
8  2019-01-12    delhi           18          9  morning
9  2019-01-12    delhi           19          7  evening
10 2019-01-12  kolkata           21          7  morning
11 2019-01-12  kolkata           23         10  evening
```

```
# pivot table for the data
# by default takes mean on the values
```

```
df.pivot_table(index='city',
columns='date',values=['temperature','windspeed'])

        temperature              windspeed
date     2019-01-12 2020-01-03 2019-01-12 2020-01-03
city
delhi          18.5       41.0        8.0        8.0
kolkata        22.0       37.5        8.5        7.0
mumbai         21.0       42.5       11.0       10.0

df.pivot_table(index='date',
columns='city',values=['temperature','windspeed'])

# By default the above result provides us the mean of the temperature
and windspeed

           temperature                windspeed
city             delhi kolkata mumbai     delhi kolkata mumbai
date
2019-01-12        18.5    22.0   21.0       8.0     8.5   11.0
2020-01-03        41.0    37.5   42.5       8.0     7.0   10.0

#  Aggregate function
df.pivot_table(index='city',columns='date',aggfunc='sum',values=['temp
erature','windspeed'])

        temperature              windspeed
date     2019-01-12 2020-01-03 2019-01-12 2020-01-03
city
delhi            37         82         16         16
kolkata          44         75         17         14
mumbai           42         85         22         20

df.pivot_table(index='city',columns='date', aggfunc='count')

# count. i.e. the number of dates available for that city

        temperature                  time                windspeed

date     2019-01-12 2020-01-03 2019-01-12 2020-01-03 2019-01-12 2020-
01-03
city

delhi             2          2          2          2          2
2
kolkata           2          2          2          2          2
2
mumbai            2          2          2          2          2
2
```

```
df.pivot_table(index='city',columns='date',aggfunc=[min,max,sum],value
s=['temperature','windspeed'])
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1369685544.py:1:
FutureWarning: The provided callable <built-in function min> is
currently using DataFrameGroupBy.min. In a future version of pandas,
the provided callable will be used directly. To keep current behavior
pass the string "min" instead.

df.pivot_table(index='city',columns='date',aggfunc=[min,max,sum],value
s=['temperature','windspeed'])
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1369685544.py:1:
FutureWarning: The provided callable <built-in function max> is
currently using DataFrameGroupBy.max. In a future version of pandas,
the provided callable will be used directly. To keep current behavior
pass the string "max" instead.

df.pivot_table(index='city',columns='date',aggfunc=[min,max,sum],value
s=['temperature','windspeed'])
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1369685544.py:1:
FutureWarning: The provided callable <built-in function sum> is
currently using DataFrameGroupBy.sum. In a future version of pandas,
the provided callable will be used directly. To keep current behavior
pass the string "sum" instead.

df.pivot_table(index='city',columns='date',aggfunc=[min,max,sum],value
s=['temperature','windspeed'])

                min                                          max
\
        temperature             windspeed            temperature

date     2019-01-12 2020-01-03 2019-01-12 2020-01-03  2019-01-12 2020-
01-03
city

delhi            18         40          7          8          19
42
kolkata          21         37          7          6          23
38
mumbai           20         42         10          9          22
43

                                      sum

         windspeed            temperature             windspeed

date    2019-01-12 2020-01-03  2019-01-12 2020-01-03 2019-01-12 2020-
01-03
city
```

```
delhi            9          8         37         82         16
16
kolkata         10          8         44         75         17
14
mumbai          12         11         42         85         22
20
```

```
#   Custom functions to individual columns
df.pivot_table(index='city',columns='date', aggfunc={
    'temperature':[min,max,'mean'],
    'windspeed': 'sum'
})

#a dictionary with keys as column name and values as the function
# to be applied
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1742279413.py:2:
FutureWarning: The provided callable <built-in function min> is
currently using SeriesGroupBy.min. In a future version of pandas, the
provided callable will be used directly. To keep current behavior pass
the string "min" instead.
  df.pivot_table(index='city',columns='date', aggfunc={
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1742279413.py:2:
FutureWarning: The provided callable <built-in function max> is
currently using SeriesGroupBy.max. In a future version of pandas, the
provided callable will be used directly. To keep current behavior pass
the string "max" instead.
  df.pivot_table(index='city',columns='date', aggfunc={
```

```
        temperature
\
                max                  mean                   min

date     2019-01-12 2020-01-03 2019-01-12 2020-01-03 2019-01-12 2020-
01-03
city

delhi            19         42       18.5       41.0         18
40
kolkata          23         38       22.0       37.5         21
37
mumbai           22         43       21.0       42.5         20
42

         windspeed
               sum
date    2019-01-12 2020-01-03
city
delhi           16         16
```

```
kolkata          17          14
mumbai           22          20
```

```
#   Apply pivot_table() on desired columns

df.pivot_table(index='city', columns='date', aggfunc='sum',
values='windspeed')
```

```
date      2019-01-12  2020-01-03
city
delhi             16          16
kolkata           17          14
mumbai            22          20
```

```
# ALternatively
df.pivot_table(index='city', columns='date', aggfunc='sum')
['windspeed']
```

```
date      2019-01-12  2020-01-03
city
delhi             16          16
kolkata           17          14
mumbai            22          20
```

```
df
```

```
         date     city  temperature  windspeed     time
0  2020-01-03   mumbai           43          9  morning
1  2020-01-03   mumbai           42         11  evening
2  2020-01-03    delhi           40          8  morning
3  2020-01-03    delhi           42          8  evening
4  2020-01-03  kolkata           38          6  morning
5  2020-01-03  kolkata           37          8  evening
6  2019-01-12   mumbai           22         12  morning
7  2019-01-12   mumbai           20         10  evening
8  2019-01-12    delhi           18          9  morning
9  2019-01-12    delhi           19          7  evening
10 2019-01-12  kolkata           21          7  morning
11 2019-01-12  kolkata           23         10  evening
```

```
df['date'] = pd.to_datetime(df['date'])
df
```

```
         date     city  temperature  windspeed     time
0  2020-01-03   mumbai           43          9  morning
1  2020-01-03   mumbai           42         11  evening
2  2020-01-03    delhi           40          8  morning
3  2020-01-03    delhi           42          8  evening
4  2020-01-03  kolkata           38          6  morning
5  2020-01-03  kolkata           37          8  evening
6  2019-01-12   mumbai           22         12  morning
```

```
7  2019-01-12   mumbai           20         10  evening
8  2019-01-12    delhi           18          9  morning
9  2019-01-12    delhi           19          7  evening
10 2019-01-12  kolkata           21          7  morning
11 2019-01-12  kolkata           23         10  evening

# mean of row in 'All' column
# average of values on two different dates i.e. average of each
row(named as 'All' by default)
df.pivot_table(index='city', columns='date', values='temperature',
margins=True)

date     2019-01-12 00:00:00  2020-01-03 00:00:00        All
city
delhi                   18.5            41.000000  29.750000
kolkata                 22.0            37.500000  29.750000
mumbai                  21.0            42.500000  31.750000
All                     20.5            40.333333  30.416667

# Giving margin's name
df.pivot_table(index='city', columns='date', values='temperature',
margins=True,margins_name= 'average')

date     2019-01-12 00:00:00  2020-01-03 00:00:00    average
city
delhi                   18.5            41.000000  29.750000
kolkata                 22.0            37.500000  29.750000
mumbai                  21.0            42.500000  31.750000
average                 20.5            40.333333  30.416667

df

         date     city  temperature  windspeed     time
0  2020-01-03   mumbai           43          9  morning
1  2020-01-03   mumbai           42         11  evening
2  2020-01-03    delhi           40          8  morning
3  2020-01-03    delhi           42          8  evening
4  2020-01-03  kolkata           38          6  morning
5  2020-01-03  kolkata           37          8  evening
6  2019-01-12   mumbai           22         12  morning
7  2019-01-12   mumbai           20         10  evening
8  2019-01-12    delhi           18          9  morning
9  2019-01-12    delhi           19          7  evening
10 2019-01-12  kolkata           21          7  morning
11 2019-01-12  kolkata           23         10  evening

#  [[[[[ Grouper ]]]]]

import pandas as pd

# Convert 'date' to datetime if not already
```

```
df['date'] = pd.to_datetime(df['date'])

# Create pivot table with yearly aggregation
pivot = df.pivot_table(
    index=pd.Grouper(freq='Y', key='date'),  # Yearly grouping
    columns='city',
    values=['temperature', 'windspeed'],     # Values to aggregate
    aggfunc='mean'                           # Default is mean, but
explicit here
)

print(pivot)
```

```
           temperature                windspeed
city             delhi kolkata mumbai     delhi kolkata mumbai
date
2019-12-31        18.5    22.0   21.0       8.0     8.5   11.0
2020-12-31        41.0    37.5   42.5       8.0     7.0   10.0
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\4142982490.py:10:
FutureWarning: 'Y' is deprecated and will be removed in a future
version, please use 'YE' instead.
  index=pd.Grouper(freq='Y', key='date'),  # Yearly grouping
```

```
df = pd.DataFrame({
     "first":list("aaabbbccd"),
     "second":list("xyzxyzxyy"),
     "third":[1,2,3,4,5,6,7,8,9]
 })
df
```

```
  first second  third
0     a      x      1
1     a      y      2
2     a      z      3
3     b      x      4
4     b      y      5
5     b      z      6
6     c      x      7
7     c      y      8
8     d      y      9
```

```
df.pivot_table(index='first', columns='second')
```

```
       third
second     x    y    z
first
a        1.0  2.0  3.0
b        4.0  5.0  6.0
c        7.0  8.0  NaN
d        NaN  9.0  NaN
```

```
df.pivot_table(index= 'first' , columns = 'second', fill_value="NILL
Data")
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1873935781.py:1:
FutureWarning: Downcasting object dtype arrays
on .fillna, .ffill, .bfill is deprecated and will change in a future
version. Call result.infer_objects(copy=False) instead. To opt-in to
the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  df.pivot_table(index= 'first' , columns = 'second', fill_value="NILL
Data")
```

```
            third
second          x    y          z
first
a             1.0  2.0        3.0
b             4.0  5.0        6.0
c             7.0  8.0  NILL Data
d       NILL Data  9.0  NILL Data
```

```
# Reshape DataFrame using melt

df = pd.read_csv('subject_melt.csv')
df
```

```
    subjects  arun  varun  neha
0      maths    72     88    87
1    physics    92     74    81
2  chemistry    55     69    78
3    biology    82     77    89
4  computers    68     71    76
```

```
df1 = pd.melt(df, id_vars=['subjects'])
df1
```

```
     subjects variable  value
0       maths     arun     72
1     physics     arun     92
2   chemistry     arun     55
3     biology     arun     82
4   computers     arun     68
5       maths    varun     88
6     physics    varun     74
7   chemistry    varun     69
8     biology    varun     77
9   computers    varun     71
10      maths     neha     87
11    physics     neha     81
12  chemistry     neha     78
13    biology     neha     89
14  computers     neha     76
```

```
df1[df1['subjects']=='maths']
```

```
   subjects variable  value
0     maths     arun     72
5     maths    varun     88
10    maths     neha     87
```

```
# Melt for only one colum
df1 = pd.melt(df, id_vars=['subjects'], value_vars=['arun'])
df1
```

```
    subjects variable  value
0      maths     arun     72
1    physics     arun     92
2  chemistry     arun     55
3    biology     arun     82
4  computers     arun     68
```

```
#  Melt multiple columns
df1 = pd.melt(df, id_vars=['subjects'], value_vars=['arun','neha'])
df1
```

```
    subjects variable  value
0      maths     arun     72
1    physics     arun     92
2  chemistry     arun     55
3    biology     arun     82
4  computers     arun     68
5      maths     neha     87
6    physics     neha     81
7  chemistry     neha     78
8    biology     neha     89
9  computers     neha     76
```

```
df1 = pd.melt(df, id_vars=['subjects'], var_name='nAMe',
value_name='Marks')
df1
```

```
     subjects   nAMe  Marks
0       maths   arun     72
1     physics   arun     92
2   chemistry   arun     55
3     biology   arun     82
4   computers   arun     68
5       maths  varun     88
6     physics  varun     74
7   chemistry  varun     69
8     biology  varun     77
9   computers  varun     71
10      maths   neha     87
11    physics   neha     81
```

```
12  chemistry   neha     78
13    biology   neha     89
14  computers   neha     76
```

```
# Reshaping using stack and unstack
# converting columns to rows and rows to columns by stack and unstack
respectively

df
```

```
    subjects  arun  varun  neha
0      maths    72     88    87
1    physics    92     74    81
2  chemistry    55     69    78
3    biology    82     77    89
4  computers    68     71    76
```

```
df = pd.read_excel('students_stack.xlsx',header=[0,1,2], index_col=0)
df
```

```
      name
      arun                        varun
     maths physics chemistry maths physics chemistry
sem1    60      63        62    58      66        65
sem2    58      61        60    56      64        63
sem3    62      65        64    60      68        67
sem4    67      70        69    65      73        72
```

```
df.stack() # by default stacks the last level of header , here last
header is row 2
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1380500203.py:1:
FutureWarning: The previous implementation of stack is deprecated and
will be removed in a future version of pandas. See the What's New
notes for pandas 2.1.0 for details. Specify future_stack=True to adopt
the new implementation and silence this warning.
  df.stack() # by default stacks the last level of header , here last
header is row 2
```

```
               name
               arun varun
sem1 chemistry   62    65
     maths       60    58
     physics     63    66
sem2 chemistry   60    63
     maths       58    56
     physics     61    64
sem3 chemistry   64    67
     maths       62    60
     physics     65    68
sem4 chemistry   69    72
```

```
     maths       67    65
     physics     70    73
```

```python
df_stacked = df.stack(level=1)
df_stacked
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\2980282631.py:1:
FutureWarning: The previous implementation of stack is deprecated and
will be removed in a future version of pandas. See the What's New
notes for pandas 2.1.0 for details. Specify future_stack=True to adopt
the new implementation and silence this warning.
  df_stacked = df.stack(level=1)
```

```
            name
           maths physics chemistry
sem1 arun     60      63        62
     varun    58      66        65
sem2 arun     58      61        60
     varun    56      64        63
sem3 arun     62      65        64
     varun    60      68        67
sem4 arun     67      70        69
     varun    65      73        72
```

```python
# ALternatively
df_stacked = df.stack(1)
df_stacked
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\457864059.py:2:
FutureWarning: The previous implementation of stack is deprecated and
will be removed in a future version of pandas. See the What's New
notes for pandas 2.1.0 for details. Specify future_stack=True to adopt
the new implementation and silence this warning.
  df_stacked = df.stack(1)
```

```
            name
           maths physics chemistry
sem1 arun     60      63        62
     varun    58      66        65
sem2 arun     58      61        60
     varun    56      64        63
sem3 arun     62      65        64
     varun    60      68        67
sem4 arun     67      70        69
     varun    65      73        72
```

```python
#  Stack on multiple levels of column
df_stacked = df.stack(level=[1,2]) # Or, df_stacked([1,2)
df_stacked                         # df_stacked
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\155454708.py:2:
FutureWarning: The previous implementation of stack is deprecated and
will be removed in a future version of pandas. See the What's New
notes for pandas 2.1.0 for details. Specify future_stack=True to adopt
the new implementation and silence this warning.
  df_stacked = df.stack(level=[1,2]) # Or, df_stacked([1,2)
```

```
                      name
sem1 arun  chemistry    62
           maths        60
           physics      63
     varun chemistry    65
           maths        58
           physics      66
sem2 arun  chemistry    60
           maths        58
           physics      61
     varun chemistry    63
           maths        56
           physics      64
sem3 arun  chemistry    64
           maths        62
           physics      65
     varun chemistry    67
           maths        60
           physics      68
sem4 arun  chemistry    69
           maths        67
           physics      70
     varun chemistry    72
           maths        65
           physics      73
```

```
#  Stack by default removes the row with all missing values
#  but still we can control the behaviour by parameter called ‘dropna’

columns2 = pd.MultiIndex.from_tuples([('weight', 'kilogram'),
                                     ('height', 'meter')]) # First
column: Level 0='weight', Level 1='kilogram'
animals_df = pd.DataFrame([[1.3,None], [8, 2.8]],           #  Second
column: Level 0='height', Level 1='meter'
                          index=['rat','dog'],
                          columns= columns2
                         )                         #columns=columns2 sets
the column headers
animals_df
```

```
       weight height
     kilogram  meter
```

```
rat       1.3    NaN
dog       8.0    2.8
```

```
animals_df.stack() #Header is used i.e, kilogram and meter
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\34287367.py:1:
FutureWarning: The previous implementation of stack is deprecated and
will be removed in a future version of pandas. See the What's New
notes for pandas 2.1.0 for details. Specify future_stack=True to adopt
the new implementation and silence this warning.
  animals_df.stack() #Header is used i.e, kilogram and meter
```

```
              weight  height
rat kilogram     1.3     NaN
dog kilogram     8.0     NaN
    meter        NaN     2.8
```

```
animals_df.stack(dropna = False) # rat [height and meter both are
none]
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1013946429.py:1:
FutureWarning: The previous implementation of stack is deprecated and
will be removed in a future version of pandas. See the What's New
notes for pandas 2.1.0 for details. Specify future_stack=True to adopt
the new implementation and silence this warning.
  animals_df.stack(dropna = False) # rat [height and meter both are
none]
```

```
              weight  height
rat kilogram     1.3     NaN
    meter        NaN     NaN
dog kilogram     8.0     NaN
    meter        NaN     2.8
```

```
# Unstack the stacked DataFrame
```

```
df = pd.read_excel('students_stack.xlsx',header=[0,1,2], index_col=0)
df
```

```
      name
      arun                       varun
     maths physics chemistry maths physics chemistry
sem1    60      63        62    58      66        65
sem2    58      61        60    56      64        63
sem3    62      65        64    60      68        67
sem4    67      70        69    65      73        72
```

```
df_stacked = df.stack(2)
df_stacked
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1602624969.py:1:
FutureWarning: The previous implementation of stack is deprecated and
```

```
will be removed in a future version of pandas. See the What's New
notes for pandas 2.1.0 for details. Specify future_stack=True to adopt
the new implementation and silence this warning.
  df_stacked = df.stack(2)
```

```
               name
               arun varun
sem1 chemistry   62    65
     maths       60    58
     physics     63    66
sem2 chemistry   60    63
     maths       58    56
     physics     61    64
sem3 chemistry   64    67
     maths       62    60
     physics     65    68
sem4 chemistry   69    72
     maths       67    65
     physics     70    73
```

```
df_stacked.unstack()
```

```
          name
          arun                       varun
     chemistry maths physics chemistry maths physics
sem1        62    60      63        65    58      66
sem2        60    58      61        63    56      64
sem3        64    62      65        67    60      68
sem4        69    67      70        72    65      73
```

```
df = pd.read_excel('students_stack.xlsx',header=[0,1,2], index_col=0)
df
```

```
      name
      arun                     varun
     maths physics chemistry maths physics chemistry
sem1    60      63        62    58      66        65
sem2    58      61        60    56      64        63
sem3    62      65        64    60      68        67
sem4    67      70        69    65      73        72
```

```
df_stacked = df.stack(level = [1,2])
df_stacked
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\425497088.py:1:
FutureWarning: The previous implementation of stack is deprecated and
will be removed in a future version of pandas. See the What's New
notes for pandas 2.1.0 for details. Specify future_stack=True to adopt
the new implementation and silence this warning.
  df_stacked = df.stack(level = [1,2])
```

```
                      name
sem1 arun  chemistry    62
           maths        60
           physics      63
     varun chemistry    65
           maths        58
           physics      66
sem2 arun  chemistry    60
           maths        58
           physics      61
     varun chemistry    63
           maths        56
           physics      64
sem3 arun  chemistry    64
           maths        62
           physics      65
     varun chemistry    67
           maths        60
           physics      68
sem4 arun  chemistry    69
           maths        67
           physics      70
     varun chemistry    72
           maths        65
           physics      73
```

```python
df_stacked.unstack(level=0)    # sem index will be column i.e, 0
```

```
                name
                sem1 sem2 sem3 sem4
arun  chemistry   62   60   64   69
      maths       60   58   62   67
      physics     63   61   65   70
varun chemistry   65   63   67   72
      maths       58   56   60   65
      physics     66   64   68   73
```

```python
df_stacked.unstack(1)
```

```
               name
               arun varun
sem1 chemistry   62    65
     maths       60    58
     physics     63    66
sem2 chemistry   60    63
     maths       58    56
     physics     61    64
sem3 chemistry   64    67
     maths       62    60
     physics     65    68
```

```
sem4 chemistry   69    72
     maths       67    65
     physics     70    73

# Unstack multiple indexes

df_stacked.unstack(level=[1,2])

          name
          arun                       varun
     chemistry maths physics chemistry maths physics
sem1        62    60      63        65    58      66
sem2        60    58      61        63    56      64
sem3        64    62      65        67    60      68
sem4        69    67      70        72    65      73

#  Frequency distribution of DataFrame column
#  interrelation between two variables and can help find interactions
between them

df = pd.read_csv('hair_color.csv')
df

      name country gender  age hair_color
0      Ram   India      M   23      black
1   Mathew      UK      M   27      brown
2  Gillian      UK      F   43      brown
3      Tom     USA      M   33      brown
4     Anna     USA      F   25     blonde
5   Sophia     USA      F   27     blonde
6     Emma      UK      F   52     blonde
7    Sweta   India      F   23      black
8    Mohan   India      M   44      black
9   Amelia      UK      F   24     blonde

pd.crosstab(df.country, df.hair_color)

#  The 1st parameter to crosstab function is the ‘index’ and the 2nd
parameter is
#  the ‘column’ on which we want to apply the frequency distribution

hair_color  black  blonde  brown
country
India           3       0      0
UK              0       2      2
USA             0       2      1

pd.crosstab(df.gender, df.hair_color) # the parameter passed are the
DataFrame column (as df.gender etc.)
```

```
hair_color  black  blonde  brown
gender
F               1       4      1
M               2       0      2

pd.crosstab(df.gender, df.hair_color, margins = True) # Get total of
rows/columns

hair_color  black  blonde  brown  All
gender
F               1       4      1    6
M               2       0      2    4
All             3       4      3   10

#  Multilevel columns
pd.crosstab(df.country, [df.gender, df.hair_color])  # M has no blonde
color

gender         F                  M
hair_color black blonde brown black brown
country
India          1      0     0     2     0
UK             0      2     1     0     1
USA            0      2     0     0     1

# Multilevel indexes
pd.crosstab([df.gender,df.country], df.hair_color) #Index are
df.gender,df.country

hair_color      black  blonde  brown
gender country
F      India        1       0      0
       UK           0       2      1
       USA          0       2      0
M      India        2       0      0
       UK           0       0      1
       USA          0       0      1

pd.crosstab([df.gender, df.country],
            df.hair_color,
            rownames = ['GENDER', 'COUNTRY'],
            colnames = ['HAIRcolor'])


# We have provided two names to rownames as we have two indexes and 1
to
# colnames as we have only one column

HAIRcolor       black  blonde  brown
GENDER COUNTRY
F      India        1       0      0
```

```
       UK             0       2      1
       USA            0       2      0
M      India          2       0      0
       UK             0       0      1
       USA            0       0      1

#   Normalize (percentage) of the frequency

pd.crosstab([df.gender], df.hair_color, normalize='index')

hair_color     black    blonde     brown
gender
F           0.166667  0.666667  0.166667
M           0.500000  0.000000  0.500000

pd.crosstab(df.gender,
            df.hair_color,
            values = df.age,
            aggfunc = 'mean')

hair_color  black  blonde  brown
gender
F            23.0    32.0   43.0
M            33.5     NaN   30.0

# Drop unwanted rows/columns
df = pd.read_csv('hair_color.csv')
df

      name country gender  age hair_color
0      Ram   India      M   23      black
1   Mathew      UK      M   27      brown
2  Gillian      UK      F   43      brown
3      Tom     USA      M   33      brown
4     Anna     USA      F   25     blonde
5   Sophia     USA      F   27     blonde
6     Emma      UK      F   52     blonde
7    Sweta   India      F   23      black
8    Mohan   India      M   44      black
9   Amelia      UK      F   24     blonde

df_drop = pd.crosstab([df.gender, df.country], df.hair_color)
df_drop

hair_color      black  blonde  brown
gender country
F      India        1       0      0
       UK           0       2      1
       USA          0       2      0
M      India        2       0      0
```

```
       UK            0       0      1
       USA           0       0      1

df_drop.drop('M') # by default axis = 0 [row]

hair_color      black  blonde  brown
gender country
F      India         1       0      0
       UK            0       2      1
       USA           0       2      0

#Delete rows of custom index level

df_drop.drop(index='India', level= 1)

#  we have deleted the row with index ‘India’ at level 1

hair_color      black  blonde  brown
gender country
F      UK            0       2      1
       USA           0       2      0
M      UK            0       0      1
       USA           0       0      1

#  Delete multiple rows

df_drop.drop(index= ['India','UK'], level=1)

hair_color      black  blonde  brown
gender country
F      USA           0       2      0
M      USA           0       0      1

#  Drop column
# These oepartion are related to crosstab
df_drop.drop(columns= 'brown')

hair_color      black  blonde
gender country
F      India         1       0
       UK            0       2
       USA           0       2
M      India         2       0
       UK            0       0
       USA           0       0

# Alternatively

df_drop.drop('brown', axis=1)

hair_color      black  blonde
gender country
```

```
F      India        1       0
       UK           0       2
       USA          0       2
M      India        2       0
       UK           0       0
       USA          0       0

# Delete multilevel columns
df_drop = pd.crosstab(df.country,
                      [df.gender, df.hair_color])
df_drop

gender          F                  M
hair_color black blonde brown black brown
country
India           1      0     0     2     0
UK              0      2     1     0     1
USA             0      2     0     0     1

df_drop.drop(columns = ['black', 'blonde'] , level=1 ) #all column
with 'black' and 'brown' name were removed

gender          F     M
hair_color brown brown
country
India           0     0
UK              1     1
USA             0     1

df_drop = pd.crosstab([df.country], [df.gender,df.hair_color])
df_drop

gender          F                  M
hair_color black blonde brown black brown
country
India           1      0     0     2     0
UK              0      2     1     0     1
USA             0      2     0     0     1

#   Delete both rows & columns

# pass both ‘index’ (rows) and ‘columns’ as parameter
# to delete both rows and columns together

df_drop.drop(index= 'UK', columns='M')

gender          F
hair_color black blonde brown
country
India           1      0     0
USA             0      2     0
```

```
# Remove duplicate values

df = pd.DataFrame({
 "name":['arun', 'varun', 'neha', 'varun', 'varun', 'arun'],
 'instruments':['violin', 'drum', 'flute', 'guitar', 'bongo','tabla'],
 'start_date': ['Jan 10, 2020', 'Mar 3, 2003', 'Feb 6, 2005', 'Dec 8,
2008',
 'Nov 5, 2011', 'Mar 10, 2011']
 })
df.start_date= pd.to_datetime(df.start_date)
df

    name instruments start_date
0   arun      violin 2020-01-10
1  varun        drum 2003-03-03
2   neha       flute 2005-02-06
3  varun      guitar 2008-12-08
4  varun       bongo 2011-11-05
5   arun       tabla 2011-03-10

# Remove duplicate

df.drop_duplicates(subset = 'name') #  remaining rows are the rows
with their first occurrence

    name instruments start_date
0   arun      violin 2020-01-10
1  varun        drum 2003-03-03
2   neha       flute 2005-02-06

# ALternatively
df.drop_duplicates('name')

    name instruments start_date
0   arun      violin 2020-01-10
1  varun        drum 2003-03-03
2   neha       flute 2005-02-06

#  Fetch custom occurrence of data

df.drop_duplicates('name', keep = 'first') #by default keeps first
occurence

    name instruments start_date
0   arun      violin 2020-01-10
1  varun        drum 2003-03-03
2   neha       flute 2005-02-06

df

    name instruments start_date
0   arun      violin 2020-01-10
```

```
1  varun        drum 2003-03-03
2   neha       flute 2005-02-06
3  varun      guitar 2008-12-08
4  varun       bongo 2011-11-05
5   arun       tabla 2011-03-10
```

```python
df.drop_duplicates('name', keep= 'last') #Index has no sequence,which 
problem will solve by ignore_index
```

```
    name instruments start_date
2   neha       flute 2005-02-06
4  varun       bongo 2011-11-05
5   arun       tabla 2011-03-10
```

```python
# Remove all duplicates
df.drop_duplicates('name' , keep = False)
```

```
   name instruments start_date
2  neha       flute 2005-02-06
```

```python
# while deleting the duplicates the indexes becomes nonsequential.This 
can be
# handled by a parameter ‘ignore_index’.

# non-sequential : not following a particular order, or not following 
one after the other in order

#Ignore Index

df.drop_duplicates('name',
                   keep='last',
                   ignore_index=True) #index 0,1,2
```

```
    name instruments start_date
0   neha       flute 2005-02-06
1  varun       bongo 2011-11-05
2   arun       tabla 2011-03-10
```

```python
# Sort the data

df = pd.DataFrame({
    'alphabet': list('dpbtbkc'),
    'num1':[1,2,np.nan,4,3,7,2],
    'num2':[3,4,3,4,7,5,4]
})
df #In book  'num2':[3,4,3,4,2,5,4] which is changed for understanding 
if 2nd column has no order with 1st columns
```

```
  alphabet  num1  num2
0        d   1.0     3
1        p   2.0     4
2        b   NaN     3
```

```
3         t   4.0     4
4         b   3.0     7
5         k   7.0     5
6         c   2.0     4

#Sort columns
# columns: axis=0 [by default]
# rows: axis=1 [if we need]

df.sort_values(by='alphabet')

# OR, df.sort_values(by=['alphabet'])
# OR, df.sort_values('alphabet')

  alphabet  num1  num2
2        b   NaN     3
4        b   3.0     7
6        c   2.0     4
0        d   1.0     3
5        k   7.0     5
1        p   2.0     4
3        t   4.0     4

 # Sorting multiple columns
df.sort_values(by=['alphabet', 'num2'])

# OR, df.sort_values(by=['alphabet', 'num2']) ; 1st sort alphabet
column then if
# posiible num2 is sorted(maybe all values will not be sorted)

  alphabet  num1  num2
2        b   NaN     3
4        b   3.0     7
6        c   2.0     4
0        d   1.0     3
5        k   7.0     5
1        p   2.0     4
3        t   4.0     4

# Sorting order

df.sort_values(by='alphabet', ascending=False) # descending

  alphabet  num1  num2
3        t   4.0     4
1        p   2.0     4
5        k   7.0     5
0        d   1.0     3
6        c   2.0     4
2        b   NaN     3
4        b   3.0     7
```

```
# by default missing value is at the end

df.sort_values('num1')
```

```
  alphabet  num1  num2
0        d   1.0     3
1        p   2.0     4
6        c   2.0     4
4        b   3.0     7
3        t   4.0     4
5        k   7.0     5
2        b   NaN     3
```

```
# Positioning missing value
df.sort_values('num1',na_position = 'first')
```

```
  alphabet  num1  num2
2        b   NaN     3
0        d   1.0     3
1        p   2.0     4
6        c   2.0     4
4        b   3.0     7
3        t   4.0     4
5        k   7.0     5
```

```
#  Working with date and time

df = pd.read_csv('TCS_data.csv')
df
```

```
           Date     Open     High      Low    Close   Volume
0    2019/04/01  2010.00  2039.95  2008.25  2031.65  2095740
1    2019/04/02  2037.10  2086.00  2037.00  2079.30  3719663
2    2019/04/03  2085.00  2089.60  2058.10  2079.30  2939886
3    2019/04/04  2078.15  2079.70  2007.40  2014.50  4397518
4    2019/04/05  2028.65  2054.40  2018.80  2048.30  3152103
..          ...      ...      ...      ...      ...      ...
240  2020/03/24  1653.05  1770.00  1632.85  1703.15  6354209
241  2020/03/25  1700.00  1810.00  1680.00  1750.30  2765527
242  2020/03/26  1831.60  1832.05  1722.55  1790.95  4556071
243  2020/03/27  1820.00  1850.00  1750.40  1824.50  4331310
244  2020/03/30  1766.00  1905.00  1763.55  1778.50  8513608

[245 rows x 6 columns]
```

```
type(df['Date'][0])
```

```
str
```

```
# Converting date to timestamp and set as index

df = pd.read_csv('TCS_data.csv', parse_dates=['Date'],
index_col='Date')
df.head()
```

```
               Open      High       Low     Close   Volume
Date
2019-04-01  2010.00  2039.95  2008.25  2031.65  2095740
2019-04-02  2037.10  2086.00  2037.00  2079.30  3719663
2019-04-03  2085.00  2089.60  2058.10  2079.30  2939886
2019-04-04  2078.15  2079.70  2007.40  2014.50  4397518
2019-04-05  2028.65  2054.40  2018.80  2048.30  3152103
```

```
df.index
```

```
DatetimeIndex(['2019-04-01', '2019-04-02', '2019-04-03', '2019-04-04',
               '2019-04-05', '2019-04-08', '2019-04-09', '2019-04-10',
               '2019-04-11', '2019-04-12',
               ...
               '2020-03-17', '2020-03-18', '2020-03-19', '2020-03-20',
               '2020-03-23', '2020-03-24', '2020-03-25', '2020-03-26',
               '2020-03-27', '2020-03-30'],
              dtype='datetime64[ns]', name='Date', length=245,
freq=None)
```

```
#   Access data for particular year
df.loc['2020']
```

```
               Open      High       Low     Close   Volume
Date
2020-01-01  2168.00  2183.90  2154.00  2167.60  1354908
2020-01-02  2179.95  2179.95  2149.20  2157.65  2380752
2020-01-03  2164.00  2223.00  2164.00  2200.65  4655761
2020-01-06  2205.00  2225.95  2187.90  2200.45  3023209
2020-01-07  2200.50  2214.65  2183.80  2205.85  2429317
...             ...      ...      ...      ...      ...
2020-03-24  1653.05  1770.00  1632.85  1703.15  6354209
2020-03-25  1700.00  1810.00  1680.00  1750.30  2765527
2020-03-26  1831.60  1832.05  1722.55  1790.95  4556071
2020-03-27  1820.00  1850.00  1750.40  1824.50  4331310
2020-03-30  1766.00  1905.00  1763.55  1778.50  8513608

[63 rows x 5 columns]
```

```
#   Access data for particular month
df.loc['2020-1'].head()
```

```
               Open      High      Low    Close   Volume
Date
2020-01-01  2168.00  2183.90  2154.0  2167.60  1354908
2020-01-02  2179.95  2179.95  2149.2  2157.65  2380752
2020-01-03  2164.00  2223.00  2164.0  2200.65  4655761
```

```
2020-01-06  2205.00  2225.95  2187.9  2200.45  3023209
2020-01-07  2200.50  2214.65  2183.8  2205.85  2429317

df.loc['2020-01'].Close.mean() # Calculating average closing price for
any month

2188.8934782608694

#  Access a date range
df.loc['2020-01-01':'2020-01-07']

               Open     High     Low    Close   Volume
Date
2020-01-01  2168.00  2183.90  2154.0  2167.60  1354908
2020-01-02  2179.95  2179.95  2149.2  2157.65  2380752
2020-01-03  2164.00  2223.00  2164.0  2200.65  4655761
2020-01-06  2205.00  2225.95  2187.9  2200.45  3023209
2020-01-07  2200.50  2214.65  2183.8  2205.85  2429317

#  Resampling the data
df.Close.resample('M').mean() # ‘M’

C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1914949084.py:2:
FutureWarning: 'M' is deprecated and will be removed in a future
version, please use 'ME' instead.
  df.Close.resample('M').mean() # ‘M’

Date
2019-04-30    2109.397368
2019-05-31    2120.722727
2019-06-30    2241.686842
2019-07-31    2145.432609
2019-08-31    2219.017500
2019-09-30    2131.200000
2019-10-31    2080.752500
2019-11-30    2131.205000
2019-12-31    2134.290000
2020-01-31    2188.893478
2020-02-29    2146.592500
2020-03-31    1841.205000
Freq: ME, Name: Close, dtype: float64

#Business days are the weekdays when most companies operate, typically
Monday through Friday, excluding weekends
# and public holidays

#'WOM" The X-th Day of the Y-th Week of Each Month
# Y = The week number (1st, 2nd, 3rd, 4th, or last week).
# X = The day of the week (Monday=0, Tuesday=1, ..., Sunday=6)

#  'M' calendar month end
# 'M': "calendar month end" refers to the last day of each month

#'BM' stands for "Business Month End", which refers to the last
business day (weekday) of each month

#'SM' always generates two timestamps per month:
# 15th day (mid-month).
# Last calendar day (e.g., 2019-03-31)

# 'Q' stands for calendar quarter-end, which marks the last day of
each financial quarter (regardless of
#  weekends/holidays)

df.Close.resample('SM').mean()
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\3815591759.py:1:
FutureWarning: 'SM' is deprecated and will be removed in a future
version, please use 'SME' instead.
  df.Close.resample('SM').mean()

Date
2019-03-31    2048.955000
2019-04-15    2166.081250
2019-04-30    2159.985000
2019-05-15    2093.320833
2019-05-31    2222.360000
2019-06-15    2256.500000
2019-06-30    2176.215000
2019-07-15    2114.758333
2019-07-31    2217.620000
2019-08-15    2219.077273
2019-08-31    2190.275000
2019-09-15    2087.130000
2019-09-30    2042.105556
2019-10-15    2096.886364
2019-10-31    2184.650000
2019-11-15    2095.204545
2019-11-30    2064.638889
2019-12-15    2194.235000
2019-12-31    2197.663636
2020-01-15    2187.741667
2020-01-31    2144.833333
2020-02-15    2141.433333
2020-02-29    1984.933333
2020-03-15    1723.609091
Freq: SME-15, Name: Close, dtype: float64
```

```
# Quarterly frequency
df.Close.resample('Q').mean() # Jan,Feb,Mar=1 quarter
```

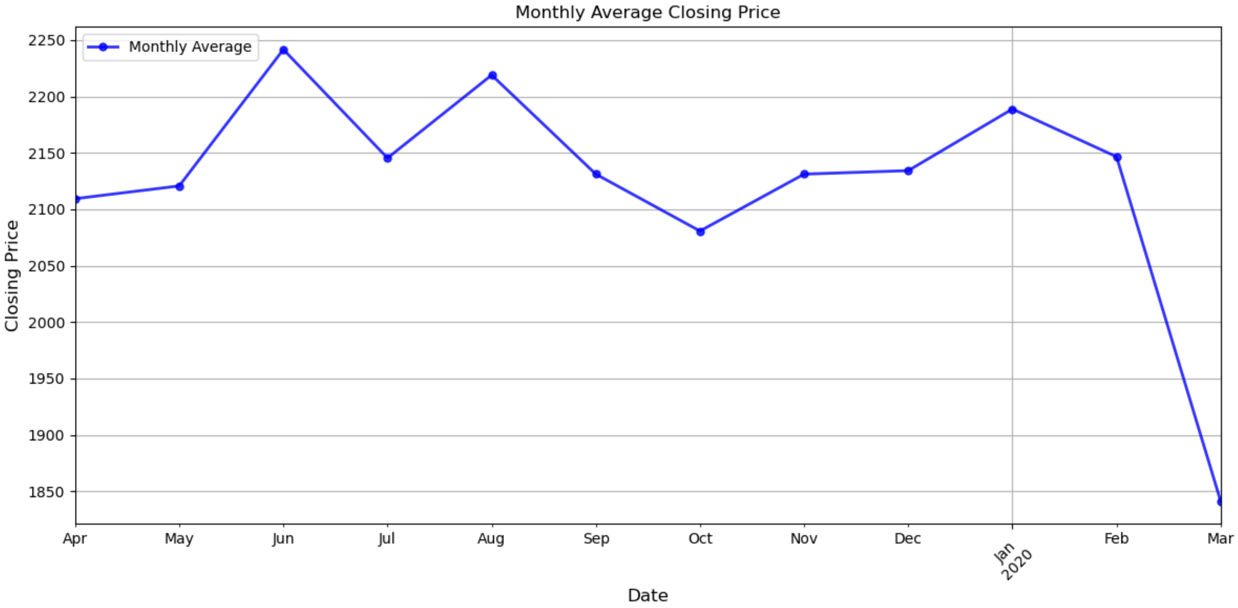
```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\365181218.py:2:
FutureWarning: 'Q' is deprecated and will be removed in a future
version, please use 'QE' instead.
  df.Close.resample('Q').mean() # Jan,Feb,Mar=1 quarter

Date
2019-06-30    2155.441667
2019-09-30    2164.808065
2019-12-31    2115.415833
2020-03-31    2065.087302
Freq: QE-DEC, Name: Close, dtype: float64
```

```python
df.Close.resample('A').mean()

#'A' stands for calendar year-end, which refers to December 31 of each
year (regardless of weekends/holidays)
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\36632666.py:1:
FutureWarning: 'A' is deprecated and will be removed in a future
version, please use 'YE' instead.
  df.Close.resample('A').mean()

Date
2019-12-31    2145.437088
2020-12-31    2065.087302
Freq: YE-DEC, Name: Close, dtype: float64
```

```python
#  Plotting the resampled data

import pandas as pd
import matplotlib.pyplot as plt

# Assuming df is your DataFrame with a datetime index and 'Close'
column
monthly_avg = df['Close'].resample('M').mean()  # Calculate monthly
averages

# Create the plot with proper formatting
ax = monthly_avg.plot(
    figsize=(12, 6),                # Set figure size
    title='Monthly Average Closing Price',  # Add title
    grid=True,                     # Show grid lines
    color='blue',                  # Line color
    linewidth=2,                   # Line thickness
    style='-o',                    # Line with circle markers
    markersize=5,                  # Marker size
    alpha=0.8                      # Slight transparency
)

# Customize the plot further
ax.set_xlabel('Date', fontsize=12)
ax.set_ylabel('Closing Price', fontsize=12)
ax.tick_params(axis='both', which='major', labelsize=10)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Add a legend
ax.legend(['Monthly Average'], loc='upper left')

# Adjust layout to prevent label cutoff
plt.tight_layout()

# Show the plot
plt.show()
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1964578811.py:7: 
FutureWarning: 'M' is deprecated and will be removed in a future 
version, please use 'ME' instead.
  monthly_avg = df['Close'].resample('M').mean()  # Calculate monthly 
averages
```



```
df.Close.resample('Q').mean()
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\3015364846.py:1: 
FutureWarning: 'Q' is deprecated and will be removed in a future 
version, please use 'QE' instead.
  df.Close.resample('Q').mean()

Date
2019-06-30    2155.441667
2019-09-30    2164.808065
```

```
2019-12-31    2115.415833
2020-03-31    2065.087302
Freq: QE-DEC, Name: Close, dtype: float64

# df.Close.resample('Q').mean().plot(kind='bar')

df = pd.read_csv('TCS_data_withoutdate.csv')
df.head()

      Open     High     Low    Close   Volume
0  2168.00  2183.90  2154.0  2167.60  1354908
1  2179.95  2179.95  2149.2  2157.65  2380752
2  2164.00  2223.00  2164.0  2200.65  4655761
3  2205.00  2225.95  2187.9  2200.45  3023209
4  2200.50  2214.65  2183.8  2205.85  2429317

rng = pd.date_range(start='01/01/2020', end='01/31/2020', freq='B')
rng

DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-06',
               '2020-01-07', '2020-01-08', '2020-01-09', '2020-01-10',
               '2020-01-13', '2020-01-14', '2020-01-15', '2020-01-16',
               '2020-01-17', '2020-01-20', '2020-01-21', '2020-01-22',
               '2020-01-23', '2020-01-24', '2020-01-27', '2020-01-28',
               '2020-01-29', '2020-01-30', '2020-01-31'],
              dtype='datetime64[ns]', freq='B')

# Apply the above date range to our data

df.set_index(rng, inplace=True)
df.head(10)

               Open     High      Low    Close   Volume
2020-01-01  2168.00  2183.90  2154.00  2167.60  1354908
2020-01-02  2179.95  2179.95  2149.20  2157.65  2380752
2020-01-03  2164.00  2223.00  2164.00  2200.65  4655761
2020-01-06  2205.00  2225.95  2187.90  2200.45  3023209
2020-01-07  2200.50  2214.65  2183.80  2205.85  2429317
2020-01-08  2205.00  2260.00  2202.05  2255.25  5197454
2020-01-09  2248.75  2251.95  2210.00  2214.35  3734173
2020-01-10  2228.00  2234.00  2208.00  2213.55  1915807
2020-01-13  2217.85  2218.95  2184.70  2190.35  2843893
2020-01-14  2195.00  2229.80  2195.00  2206.90  2948452

 rng = pd.date_range(start="01/01/2020", end="01/31/2020", freq="D")
rng

# This would have given all the dates, but this can’t be applied
because we
# have data for only working days. By above method we would have
```

```
generated
# the dates but would have not able to map it with the data.

DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
               '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08',
               '2020-01-09', '2020-01-10', '2020-01-11', '2020-01-12',
               '2020-01-13', '2020-01-14', '2020-01-15', '2020-01-16',
               '2020-01-17', '2020-01-18', '2020-01-19', '2020-01-20',
               '2020-01-21', '2020-01-22', '2020-01-23', '2020-01-24',
               '2020-01-25', '2020-01-26', '2020-01-27', '2020-01-28',
               '2020-01-29', '2020-01-30', '2020-01-31'],
              dtype='datetime64[ns]', freq='D')

#   Generate the missing data with missing dates by 'asfreq()'
function

df.asfreq('D').head(7)

               Open     High     Low    Close     Volume
2020-01-01  2168.00  2183.90  2154.0  2167.60  1354908.0
2020-01-02  2179.95  2179.95  2149.2  2157.65  2380752.0
2020-01-03  2164.00  2223.00  2164.0  2200.65  4655761.0
2020-01-04      NaN      NaN     NaN      NaN        NaN
2020-01-05      NaN      NaN     NaN      NaN        NaN
2020-01-06  2205.00  2225.95  2187.9  2200.45  3023209.0
2020-01-07  2200.50  2214.65  2183.8  2205.85  2429317.0

#  To fill the data we can do various methods as we learned in
previous chapters

df.asfreq('D', method='pad') # 'pad' or 'ffill'

               Open     High      Low    Close   Volume
2020-01-01  2168.00  2183.90  2154.00  2167.60  1354908
2020-01-02  2179.95  2179.95  2149.20  2157.65  2380752
2020-01-03  2164.00  2223.00  2164.00  2200.65  4655761
2020-01-04  2164.00  2223.00  2164.00  2200.65  4655761
2020-01-05  2164.00  2223.00  2164.00  2200.65  4655761
2020-01-06  2205.00  2225.95  2187.90  2200.45  3023209
2020-01-07  2200.50  2214.65  2183.80  2205.85  2429317
2020-01-08  2205.00  2260.00  2202.05  2255.25  5197454
2020-01-09  2248.75  2251.95  2210.00  2214.35  3734173
2020-01-10  2228.00  2234.00  2208.00  2213.55  1915807
2020-01-11  2228.00  2234.00  2208.00  2213.55  1915807
2020-01-12  2228.00  2234.00  2208.00  2213.55  1915807
2020-01-13  2217.85  2218.95  2184.70  2190.35  2843893
2020-01-14  2195.00  2229.80  2195.00  2206.90  2948452
2020-01-15  2213.00  2231.00  2194.20  2226.90  2620681
2020-01-16  2226.95  2249.00  2215.00  2238.80  3117214
2020-01-17  2240.75  2253.55  2213.00  2219.10  3281059
2020-01-18  2240.75  2253.55  2213.00  2219.10  3281059
```

```
2020-01-19  2240.75  2253.55  2213.00  2219.10  3281059
2020-01-20  2194.90  2242.20  2156.20  2170.35  5817599
2020-01-21  2169.95  2186.55  2158.05  2171.05  1902980
2020-01-22  2181.00  2210.00  2173.70  2206.90  1773686
2020-01-23  2209.80  2217.75  2183.70  2190.95  2069866
2020-01-24  2190.95  2190.95  2170.00  2183.40  1319430
2020-01-25  2190.95  2190.95  2170.00  2183.40  1319430
2020-01-26  2190.95  2190.95  2170.00  2183.40  1319430
2020-01-27  2189.70  2193.45  2165.00  2169.25  1549101
2020-01-28  2174.00  2187.80  2152.00  2183.75  1743024
2020-01-29  2185.00  2186.95  2150.00  2154.60  2306761
2020-01-30  2160.00  2165.00  2125.00  2137.85  2098567
2020-01-31  2139.40  2144.35  2071.60  2079.05  3287223
```

```
df.asfreq('D', method='ffill').head(7)
```

```
               Open     High     Low    Close   Volume
2020-01-01  2168.00  2183.90  2154.0  2167.60  1354908
2020-01-02  2179.95  2179.95  2149.2  2157.65  2380752
2020-01-03  2164.00  2223.00  2164.0  2200.65  4655761
2020-01-04  2164.00  2223.00  2164.0  2200.65  4655761
2020-01-05  2164.00  2223.00  2164.0  2200.65  4655761
2020-01-06  2205.00  2225.95  2187.9  2200.45  3023209
2020-01-07  2200.50  2214.65  2183.8  2205.85  2429317
```

```
# We can get the weekly prices too

df.asfreq('W', method='pad')  # Sunday
```

```
               Open     High     Low    Close   Volume
2020-01-05  2164.00  2223.00  2164.0  2200.65  4655761
2020-01-12  2228.00  2234.00  2208.0  2213.55  1915807
2020-01-19  2240.75  2253.55  2213.0  2219.10  3281059
2020-01-26  2190.95  2190.95  2170.0  2183.40  1319430
```

```
#   Date range with periods ( one more way to generate the date range
with date_range() function)

rng = pd.date_range(start='01/01/2020', periods=23, freq='B') #pass
‘periods’ parameter instead of  ‘end’ parameter

rng
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-06',
               '2020-01-07', '2020-01-08', '2020-01-09', '2020-01-10',
               '2020-01-13', '2020-01-14', '2020-01-15', '2020-01-16',
               '2020-01-17', '2020-01-20', '2020-01-21', '2020-01-22',
               '2020-01-23', '2020-01-24', '2020-01-27', '2020-01-28',
               '2020-01-29', '2020-01-30', '2020-01-31'],
              dtype='datetime64[ns]', freq='B')
```

```
df.set_index(rng, inplace=True)
df.head(8)

               Open     High      Low    Close   Volume
2020-01-01  2168.00  2183.90  2154.00  2167.60  1354908
2020-01-02  2179.95  2179.95  2149.20  2157.65  2380752
2020-01-03  2164.00  2223.00  2164.00  2200.65  4655761
2020-01-06  2205.00  2225.95  2187.90  2200.45  3023209
2020-01-07  2200.50  2214.65  2183.80  2205.85  2429317
2020-01-08  2205.00  2260.00  2202.05  2255.25  5197454
2020-01-09  2248.75  2251.95  2210.00  2214.35  3734173
2020-01-10  2228.00  2234.00  2208.00  2213.55  1915807

# Working with custom holidays

pd.date_range(start='12-01-2019', end='12-31-2019', freq='B') #month-
date-year
                                                              #output:
working date

DatetimeIndex(['2019-12-02', '2019-12-03', '2019-12-04', '2019-12-05',
               '2019-12-06', '2019-12-09', '2019-12-10', '2019-12-11',
               '2019-12-12', '2019-12-13', '2019-12-16', '2019-12-17',
               '2019-12-18', '2019-12-19', '2019-12-20', '2019-12-23',
               '2019-12-24', '2019-12-25', '2019-12-26', '2019-12-27',
               '2019-12-30', '2019-12-31'],
              dtype='datetime64[ns]', freq='B')

#   Adding US holidays
from pandas.tseries.holiday import USFederalHolidayCalendar
from pandas.tseries.offsets import CustomBusinessDay

usb = CustomBusinessDay(calendar = USFederalHolidayCalendar())
usb

<CustomBusinessDay>

# The 25th has been removed as it was Christmas in US

rng = pd.date_range(start='12/01/2019', end='12/31/2019', freq=usb)
rng

DatetimeIndex(['2019-12-02', '2019-12-03', '2019-12-04', '2019-12-05',
               '2019-12-06', '2019-12-09', '2019-12-10', '2019-12-11',
               '2019-12-12', '2019-12-13', '2019-12-16', '2019-12-17',
               '2019-12-18', '2019-12-19', '2019-12-20', '2019-12-23',
               '2019-12-24', '2019-12-26', '2019-12-27', '2019-12-30',
               '2019-12-31'],
              dtype='datetime64[ns]', freq='C')

# Creating custom calendar
```

```
from pandas.tseries.holiday import Holiday, \
AbstractHolidayCalendar, \
nearest_workday

class myHolidayCalendar(AbstractHolidayCalendar):
    rules = [
        Holiday('ExampleHoliday1', month=1, day=2),
        Holiday('ExampleHoliday1', month=1, day=7)
        ]
myc = CustomBusinessDay(calendar= myHolidayCalendar())
myc
```

```
<CustomBusinessDay>
```

```
pd.date_range(start='1-1-2020',end='1/11/2020', freq=myc)

# we have provided as holidays in myHolidayCalender (2nd and 7th) have
been removed from the date range
```

```
DatetimeIndex(['2020-01-01', '2020-01-03', '2020-01-06', '2020-01-08',
               '2020-01-09', '2020-01-10'],
              dtype='datetime64[ns]', freq='C')
```

```
# Observance rule
from pandas.tseries.holiday import Holiday, AbstractHolidayCalendar,
nearest_workday
from pandas.tseries.offsets import CustomBusinessDay

class myHolidayCalendar(AbstractHolidayCalendar):
    rules = [
        Holiday('ExampleHoliday1', month=1, day=5,
observance=nearest_workday),
        Holiday('ExampleHoliday2', month=1, day=9)
    ]

myc = CustomBusinessDay(calendar=myHolidayCalendar())
myc
```

```
<CustomBusinessDay>
```

```
# The 5th was sunday thus the nearest weekday was 6th(monday)
# If we would have provided holiday on 4th then 3rd would have been
nearest holiday

pd.date_range(start='1-1-2020',end='1-11-2020', freq= myc)
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-07',
               '2020-01-08', '2020-01-10'],
              dtype='datetime64[ns]', freq='C')
```

```
# Custom week days
```

```
custom_weekdays = 'Sun Mon Tue Wed Thu'
custom_businessDays = CustomBusinessDay(weekmask=custom_weekdays)
pd.date_range(start='1/1/2020', end='1/11/2020', freq=
custom_businessDays)

# In This example Friday(3rd) and Saturday(4th) are the weekends

DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-05', '2020-01-06',
               '2020-01-07', '2020-01-08', '2020-01-09'],
              dtype='datetime64[ns]', freq='C')

#  Custom holiday

custom_weekdays = 'Sun Mon Tue Wed Thu'
custom_businessDays = CustomBusinessDay(weekmask=custom_weekdays ,
holidays=['2020-01-06'])
pd.date_range(start='1/1/2020', end='1/11/2020', freq=
custom_businessDays)

# passed 6th as a holiday to ‘holiday’

DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-05', '2020-01-07',
               '2020-01-08', '2020-01-09'],
              dtype='datetime64[ns]', freq='C')

#  Working with date formats

#Converting to a common format
import pandas as pd

dates = ['2020-03-10', 'Mar 10, 2020', '03/10/2020', '2020.03.10',
'2020/03/10', '20200310']
pd.to_datetime(dates, format='mixed')

DatetimeIndex(['2020-03-10', '2020-03-10', '2020-03-10', '2020-03-10',
               '2020-03-10', '2020-03-10'],
              dtype='datetime64[ns]', freq=None)

#   Time conversion

dates =  ['2020-03-10 04:30:00 PM',
          'Mar 10, 2020 16:30:00',
          '03/10/2020',
          '2020.03.10',
          '2020/03/10',
          '20200310']
pd.to_datetime(dates, format='mixed')  #both the different time
formats (04:30 PM and 16:30) have became a single format (16:30)

DatetimeIndex(['2020-03-10 16:30:00', '2020-03-10 16:30:00',
               '2020-03-10 00:00:00', '2020-03-10 00:00:00',
               '2020-03-10 00:00:00', '2020-03-10 00:00:00'],
              dtype='datetime64[ns]', freq=None)
```

```
#10th March,2020
date = '10/03/2020'  # month/date/year
pd.to_datetime(date) # Normal form
```

```
Timestamp('2020-10-03 00:00:00')
```

```
#    Dayfirst formats
pd.to_datetime(date, dayfirst=True)
```

```
Timestamp('2020-03-10 00:00:00')
```

```
#   Remove custom delimiter in date

date = '10@03@2020'
pd.to_datetime(date, format='%d@%m@%Y')
```

```
Timestamp('2020-03-10 00:00:00')
```

```
date = '10@03@2020'
pd.to_datetime(date, format='%d@%m@%Y')
```

```
Timestamp('2020-03-10 00:00:00')
```

```
#   Remove custom delimiter in time

import pandas as pd

date = '10@03@2020 04&30'
pd.to_datetime(date, format='%d@%m@%Y %H&%M')
```

```
Timestamp('2020-03-10 04:30:00')
```

```
#  Handling errors in datetime

dates = ['Mar 10,2020', None]
pd.to_datetime(dates, format='%b %d,%Y', errors='coerce')

#%b  Abbreviated month name (3 letters)
```

```
DatetimeIndex(['2020-03-10', 'NaT'], dtype='datetime64[ns]',
freq=None)
```

```
# by default this will raise am error
# to avoid we can pass errors=ignore
# but none of the other conversions will be done

dates = ['Mar 10, 2020', 'xyz']
pd.to_datetime(dates, format='%b %d,%Y',errors='ignore')
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\3481787600.py:6:
FutureWarning: errors='ignore' is deprecated and will raise in a
future version. Use to_datetime without passing `errors` and catch
exceptions explicitly instead
  pd.to_datetime(dates, format='%b %d,%Y',errors='ignore')

Index(['Mar 10, 2020', 'xyz'], dtype='object')

#  Coerce the error
dates = ['Mar 10, 2020', 'xyz']
pd.to_datetime(dates, errors='coerce') # We can see that the
conversion to the valid dates have been done but the
                                       # garbage one has been
converted to NaT

DatetimeIndex(['2020-03-10', 'NaT'], dtype='datetime64[ns]',
freq=None)

#  Epoch to datetime [ number to date]

t=1
pd.to_datetime(t)

Timestamp('1970-01-01 00:00:00.000000001')

# by default in ns(nano second)
t = 1583861574
pd.to_datetime(t)

Timestamp('1970-01-01 00:00:01.583861574')

# unit = 's' for sec
# unit = 'ms' for milli sec

dt = pd.to_datetime([t], unit='s')
dt

# nix timestamp, representing the number of seconds since January 1,
1970, 00:00:00 UTC (the Unix epoch)

DatetimeIndex(['2020-03-10 17:32:54'], dtype='datetime64[ns]',
freq=None)

#  datetime to Epoch [date to number]

dt.view('int64')  #  This gives back, the epoch time

array([1583861574000000000], dtype=int64)

# Annual Period

y_2020 = pd.Period('2020')  # Period covers a range
y_2020
```

```
# output: 'Y-DEC': The frequency is yearly (Y)

Period('2020', 'Y-DEC')

y_2020.start_time  # Operations in annual period

Timestamp('2020-01-01 00:00:00')

y_2020.end_time    # Operations in annual period

Timestamp('2020-12-31 23:59:59.999999999')

#  Addition/Subtraction to annual period:

y_2019 = pd.Period('2019')
y_2019 + 1

Period('2020', 'Y-DEC')

#  Monthly period
m_2020 = pd.Period('2020-01')
m_2020

Period('2020-01', 'M')

m_2020 = pd.Period('2020-2', freq='3M')  #  February, March and April
m_2020

Period('2020-02', '3M')

m_2020.start_time

Timestamp('2020-02-01 00:00:00')

m_2020.end_time

Timestamp('2020-04-30 23:59:59.999999999')

# Addition/Subtraction on monthly period:

m_2020 = pd.Period('2020-01')
m_2020 - 1

Period('2019-12', 'M')

# Daily period

d_2020 = pd.Period('2020-03-01')
d_2020

Period('2020-03-01', 'D')
```

```
d_2020 = pd.Period('2020-03-01', freq='3D')
d_2020

Period('2020-03-01', '3D')

d_2020.start_time

Timestamp('2020-03-01 00:00:00')

d_2020.end_time

Timestamp('2020-03-03 23:59:59.999999999')

d_2020

Period('2020-03-01', '3D')

#  Adding/Subtracting days:
d_2020+2 # date: 1 + 3*2 ( freq='3D')

Period('2020-03-07', '3D')

#   Hourly period

h_2020 = pd.Period('2020-03-01 04:00')
h_2020

Period('2020-03-01 04:00', 'min')

h_2020 = pd.Period('2020-03-01 04:00', freq='H')
h_2020

C:\Users\USER\AppData\Local\Temp\ipykernel_9660\3817598763.py:1:
FutureWarning: 'H' is deprecated and will be removed in a future
version, please use 'h' instead.
  h_2020 = pd.Period('2020-03-01 04:00', freq='H')

Period('2020-03-01 04:00', 'h')

#  Adding/Subtracting hours

h_2020 - 1

Period('2020-03-01 03:00', 'h')

# Quarterly period

# A year has four quarters (Jan to Mar, Apr to Jun, Jul to Sept, Oct
to Dec by default)

# Only 1st Quarter

q1_2020 = pd.Period('2020', freq='Q')
q1_2020
```

```
Period('2020Q1', 'Q-DEC')

q1_2020.start_time

Timestamp('2020-01-01 00:00:00')

q1_2020.end_time

Timestamp('2020-03-31 23:59:59.999999999')

#  Multiple Quarters

q_2020 = pd.Period('2020', freq='3Q')
q_2020

Period('2020Q1', '3Q-DEC')

q_2020.start_time

Timestamp('2020-01-01 00:00:00')

q_2020.end_time #'3Q' 1st 9 months of a year [1 quarter = 3 month]

Timestamp('2020-09-30 23:59:59.999999999')

#  Defined Quarter

q3_2020 = pd.Period('2020Q3', freq='Q') # 3rd quarter=July to
September
q3_2020

Period('2020Q3', 'Q-DEC')

q3_2020.start_time

Timestamp('2020-07-01 00:00:00')

q3_2020.end_time

Timestamp('2020-09-30 23:59:59.999999999')

# Custom quarters

#To get a period where the quarter starts with April we have to pass
the ‘freq’ parameter as ‘Q-MAR’.

q1_2020 = pd.Period('2020Q1', freq='Q-MAR')  # Fiscal year 2020
(ending March 2020) [*** began April 2019]
q1_2020

Period('2020Q1', 'Q-MAR')

q1_2020.start_time
```

```
Timestamp('2019-04-01 00:00:00')

q1_2020.end_time

Timestamp('2019-06-30 23:59:59.999999999')

q1_2020

Period('2020Q1', 'Q-MAR')

# Converting one frequency to another

#  We’ll take the quarterly frequency we generated in above topic and
convert that to monthly.

q1_2020.asfreq('M')  # output shows the end of the period.

Period('2019-06', 'M')

q1_2020.asfreq('M', how='start')

Period('2019-04', 'M')

q1_2020.asfreq('M', how="end")

# OR,  q1_2020.asfreq('M', 'e')

Period('2019-06', 'M')

#   Arithmetic between two periods

q1_2020 = pd.Period('2020Q1', freq='Q-MAR')
q_new = pd.Period('2018Q3', freq='Q-MAR')
q1_2020 - q_new            #  This shows the difference between is of
6 quarters.

<6 * QuarterEnds: startingMonth=3>

q1_2020.start_time  # start April 2019

Timestamp('2019-04-01 00:00:00')

q1_2020.end_time   # end at June 2019

Timestamp('2019-06-30 23:59:59.999999999')

q_new.start_time # start at October 2017

Timestamp('2017-10-01 00:00:00')

q_new.end_time # end at December 2017

Timestamp('2017-12-31 23:59:59.999999999')
```

```
# October 2017 to June 2019 have 6 quarter
# Oct-Dec 2017 (2018Q3) → Jan-Mar 2018 (2018Q4) → Apr-Jun 2018
(2019Q1) →
# Jul-Sep 2018 (2019Q2) → Oct-Dec 2018 (2019Q3) → Jan-Mar 2019
(2019Q4) →
# Apr-Jun 2019 (2020Q1)

# Period Index

idx = pd.period_range('2015','2020',freq='Q')
idx

PeriodIndex(['2015Q1', '2015Q2', '2015Q3', '2015Q4', '2016Q1',
'2016Q2',
             '2016Q3', '2016Q4', '2017Q1', '2017Q2', '2017Q3',
'2017Q4',
             '2018Q1', '2018Q2', '2018Q3', '2018Q4', '2019Q1',
'2019Q2',
             '2019Q3', '2019Q4', '2020Q1'],
            dtype='period[Q-DEC]')

#  Getting given number of periods

idx = pd.period_range('2015', periods=5, freq='Q')
idx

# This provides us 5 periods, starting from 2015.

PeriodIndex(['2015Q1', '2015Q2', '2015Q3', '2015Q4', '2016Q1'],
dtype='period[Q-DEC]')

#   Period index to DataFrame

ps = pd.Series(np.random.randn(len(idx)), idx)
ps

#  We have created a DataFrame with random numbers and added the
period index to that.

2015Q1   -0.283528
2015Q2   -1.045122
2015Q3    0.419786
2015Q4    0.403627
2016Q1    0.279300
Freq: Q-DEC, dtype: float64

# Extract annual data

ps['2015']

2015Q1   -0.283528
2015Q2   -1.045122
```

```
2015Q3    0.419786
2015Q4    0.403627
Freq: Q-DEC, dtype: float64
```

```
ps['2016']
```

```
2016Q1    0.2793
Freq: Q-DEC, dtype: float64
```

```
# Extract a range of periods data

ps['2015Q3' : '2016'] #  range from 2015 Q3 to 2016 Q1
```

```
2015Q3    0.419786
2015Q4    0.403627
2016Q1    0.279300
Freq: Q-DEC, dtype: float64
```

```
# Convert DatetimeIndex to PeriodIndex

import pandas as pd

# Create a DateTimeIndex
dti = pd.date_range('2023-01-01', periods=5, freq='D')
print("DateTimeIndex:", dti)

# Convert to PeriodIndex with daily frequency
period_index = dti.to_period('D')
print("PeriodIndex:", period_index)
```

```
DateTimeIndex: DatetimeIndex(['2023-01-01', '2023-01-02', '2023-01-
03', '2023-01-04',
               '2023-01-05'],
              dtype='datetime64[ns]', freq='D')
PeriodIndex: PeriodIndex(['2023-01-01', '2023-01-02', '2023-01-03',
'2023-01-04',
             '2023-01-05'],
            dtype='period[D]')
```

```
#  Working with time zones
# Two types of times: 1,time zone aware...2,time zone unaware

df = pd.read_csv('timezone.csv',
                 index_col='date',
                 parse_dates=(['date']))
df
```

```
                     price
date
2020-10-03 01:00:00     57
2020-10-03 02:00:00     58
2020-10-03 03:00:00     59
```

```
2020-10-03 04:00:00     60
2020-10-03 05:00:00     61
2020-10-03 06:00:00     62
2020-10-03 07:00:00     63
2020-10-03 08:00:00     64
2020-10-03 09:00:00     65
2020-10-03 10:00:00     66

# Make naïve time to time zone aware

df1 = df.tz_localize(tz='US/Eastern')
df1.index

# EST (Eastern Standard Time) = UTC−5 (used in winter, no DST).
# EDT (Eastern Daylight Time) = UTC−4 (used in summer, DST active).

DatetimeIndex(['2020-10-03 01:00:00-04:00', '2020-10-03 02:00:00-
04:00',
               '2020-10-03 03:00:00-04:00', '2020-10-03 04:00:00-
04:00',
               '2020-10-03 05:00:00-04:00', '2020-10-03 06:00:00-
04:00',
               '2020-10-03 07:00:00-04:00', '2020-10-03 08:00:00-
04:00',
               '2020-10-03 09:00:00-04:00', '2020-10-03 10:00:00-
04:00'],
              dtype='datetime64[ns, US/Eastern]', name='date',
freq=None)

#  Available timezones

import pytz
pytz.all_timezones[::30]

['Africa/Abidjan',
 'Africa/Kinshasa',
 'America/Argentina/Catamarca',
 'America/Catamarca',
 'America/Guadeloupe',
 'America/Maceio',
 'America/Paramaribo',
 'America/Tegucigalpa',
 'Asia/Aqtobe',
 'Asia/Hovd',
 'Asia/Pontianak',
 'Asia/Yakutsk',
 'Australia/North',
 'Eire',
 'Etc/GMT0',
 'Europe/Kyiv',
 'Europe/Uzhgorod',
 'Indian/Reunion',
 'Pacific/Gambier',
 'Pacific/Wallis']

# For INDIA:
pytz.country_timezones('IN')

['Asia/Kolkata']

# For Bangladesh:
pytz.country_timezones('BD')

['Asia/Dhaka']

df1.index

DatetimeIndex(['2020-10-03 01:00:00-04:00', '2020-10-03 02:00:00-
04:00',
               '2020-10-03 03:00:00-04:00', '2020-10-03 04:00:00-
04:00',
               '2020-10-03 05:00:00-04:00', '2020-10-03 06:00:00-
04:00',
               '2020-10-03 07:00:00-04:00', '2020-10-03 08:00:00-
04:00',
               '2020-10-03 09:00:00-04:00', '2020-10-03 10:00:00-
04:00'],
              dtype='datetime64[ns, US/Eastern]', name='date',
freq=None)

# Convert on time zone to other

df = df1.tz_convert(tz ='Europe/Guernsey')   # df1 was in
‘US/Eastern’ time zone and is getting converted to
                                              # ‘Europe/Guernsey’ time
zone
df.index

DatetimeIndex(['2020-10-03 06:00:00+01:00', '2020-10-03
07:00:00+01:00',
               '2020-10-03 08:00:00+01:00', '2020-10-03
09:00:00+01:00',
               '2020-10-03 10:00:00+01:00', '2020-10-03
11:00:00+01:00',
               '2020-10-03 12:00:00+01:00', '2020-10-03
13:00:00+01:00',
               '2020-10-03 14:00:00+01:00', '2020-10-03
15:00:00+01:00'],
              dtype='datetime64[ns, Europe/Guernsey]', name='date',
freq=None)
```

```
# Time zone in a date range

rng  =  pd.date_range(start='10/3/2020', periods=10, freq='H',
tz='dateutil/Asia/Kolkata')
rng   # [ 1 to 8 hour ] + 5: 30,
      # IST is UTC+5:30 (always, no DST).
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_9660\1899864123.py:3:
FutureWarning: 'H' is deprecated and will be removed in a future
version, please use 'h' instead.
  rng  =  pd.date_range(start='10/3/2020', periods=10, freq='H',
tz='dateutil/Asia/Kolkata')
```

```
DatetimeIndex(['2020-10-03 00:00:00+05:30', '2020-10-03
01:00:00+05:30',
               '2020-10-03 02:00:00+05:30', '2020-10-03
03:00:00+05:30',
               '2020-10-03 04:00:00+05:30', '2020-10-03
05:00:00+05:30',
               '2020-10-03 06:00:00+05:30', '2020-10-03
07:00:00+05:30',
               '2020-10-03 08:00:00+05:30', '2020-10-03
09:00:00+05:30'],
              dtype='datetime64[ns, tzfile('Asia/Calcutta')]',
freq='h')
```

```
# Data shifts in DataFrame

df = pd.read_csv('shifting.csv', index_col='date', parse_dates =
(['date']))
df
```

```
            price
date
2020-03-10     43
2020-03-11     54
2020-03-12     73
2020-03-13     85
2020-03-14     53
2020-03-15     74
2020-03-16     76
2020-03-17     44
2020-03-18     62
2020-03-19     84
```

```
#    Shifting the price down

df.shift()

# the price for date 10 March, 2020 has been shifted to 11 march, 2020
```

```
            price
date
2020-03-10    NaN
2020-03-11   43.0
2020-03-12   54.0
2020-03-13   73.0
2020-03-14   85.0
2020-03-15   53.0
2020-03-16   74.0
2020-03-17   76.0
2020-03-18   44.0
2020-03-19   62.0

# Shifting by multiple rows

df.shift(3)

            price
date
2020-03-10    NaN
2020-03-11    NaN
2020-03-12    NaN
2020-03-13   43.0
2020-03-14   54.0
2020-03-15   73.0
2020-03-16   85.0
2020-03-17   53.0
2020-03-18   74.0
2020-03-19   76.0

# Reverse shifting
df.shift(-3)

#  the price which was for 19 is now for 17 and so on

            price
date
2020-03-10   85.0
2020-03-11   53.0
2020-03-12   74.0
2020-03-13   76.0
2020-03-14   44.0
2020-03-15   62.0
2020-03-16   84.0
2020-03-17    NaN
2020-03-18    NaN
2020-03-19    NaN

df = pd.read_csv('shifting.csv', index_col='date', parse_dates =
(['date']))
df
```

```
            price
date
2020-03-10     43
2020-03-11     54
2020-03-12     73
2020-03-13     85
2020-03-14     53
2020-03-15     74
2020-03-16     76
2020-03-17     44
2020-03-18     62
2020-03-19     84

#  New column with last day price:

df['previous_day_prices'] = df.shift()
df

            price  previous_day_prices
date
2020-03-10     43                  NaN
2020-03-11     54                 43.0
2020-03-12     73                 54.0
2020-03-13     85                 73.0
2020-03-14     53                 85.0
2020-03-15     74                 53.0
2020-03-16     76                 74.0
2020-03-17     44                 76.0
2020-03-18     62                 44.0
2020-03-19     84                 62.0

# New column with change in the price from last day:

df['changed_price'] =df['price'] - df['previous_day_prices']
df

            price  previous_day_prices  changed_price
date
2020-03-10     43                  NaN            NaN
2020-03-11     54                 43.0           11.0
2020-03-12     73                 54.0           19.0
2020-03-13     85                 73.0           12.0
2020-03-14     53                 85.0          -32.0
2020-03-15     74                 53.0           21.0
2020-03-16     76                 74.0            2.0
2020-03-17     44                 76.0          -32.0
2020-03-18     62                 44.0           18.0
2020-03-19     84                 62.0           22.0

df ['return_3days'] = (df['price']-
df['price'].shift(3)*100/df['price'].shift(3))
```

```
df

# (85 - 43) * 100 / 43 ≈ 97.67%
# (53 - 54) * 100 / 54 ≈ -1.85%

            price  previous_day_prices  changed_price  return_3days
date
2020-03-10     43                  NaN            NaN           NaN
2020-03-11     54                 43.0           11.0           NaN
2020-03-12     73                 54.0           19.0           NaN
2020-03-13     85                 73.0           12.0         -15.0
2020-03-14     53                 85.0          -32.0         -47.0
2020-03-15     74                 53.0           21.0         -26.0
2020-03-16     76                 74.0            2.0         -24.0
2020-03-17     44                 76.0          -32.0         -56.0
2020-03-18     62                 44.0           18.0         -38.0
2020-03-19     84                 62.0           22.0         -16.0

df

            price  previous_day_prices  changed_price  return_3days
date
2020-03-10     43                  NaN            NaN           NaN
2020-03-11     54                 43.0           11.0           NaN
2020-03-12     73                 54.0           19.0           NaN
2020-03-13     85                 73.0           12.0         -15.0
2020-03-14     53                 85.0          -32.0         -47.0
2020-03-15     74                 53.0           21.0         -26.0
2020-03-16     76                 74.0            2.0         -24.0
2020-03-17     44                 76.0          -32.0         -56.0
2020-03-18     62                 44.0           18.0         -38.0
2020-03-19     84                 62.0           22.0         -16.0

# DatetimeIndex shift

df.shift(freq='D')

#  the index has been shifted by one row down and day 10 has gone and
taken over by 11

            price  previous_day_prices  changed_price  return_3days
date
2020-03-11     43                  NaN            NaN           NaN
2020-03-12     54                 43.0           11.0           NaN
2020-03-13     73                 54.0           19.0           NaN
2020-03-14     85                 73.0           12.0         -15.0
2020-03-15     53                 85.0          -32.0         -47.0
2020-03-16     74                 53.0           21.0         -26.0
2020-03-17     76                 74.0            2.0         -24.0
2020-03-18     44                 76.0          -32.0         -56.0
```

```
2020-03-19     62                  44.0           18.0         -38.0
2020-03-20     84                  62.0           22.0         -16.0
```

```
# Reverse DatetimeIndex shift

df.shift(freq='-1D') # not working: '-D'

#  vanished 19 from the index.
```

```
            price  previous_day_prices  changed_price  return_3days
date
2020-03-09     43                  NaN            NaN           NaN
2020-03-10     54                 43.0           11.0           NaN
2020-03-11     73                 54.0           19.0           NaN
2020-03-12     85                 73.0           12.0         -15.0
2020-03-13     53                 85.0          -32.0         -47.0
2020-03-14     74                 53.0           21.0         -26.0
2020-03-15     76                 74.0            2.0         -24.0
2020-03-16     44                 76.0          -32.0         -56.0
2020-03-17     62                 44.0           18.0         -38.0
2020-03-18     84                 62.0           22.0         -16.0
```

```
# Working with MySQL

import pandas as pdf
import sqlalchemy

# Create connection

engine =
sqlalchemy.create_engine('mysql+pymysql://root:NAZMULhasan11#@localhos
t:3306/students_db')
engine
```

```
Engine(mysql+pymysql://root:***@localhost:3306/students_db)
```

```
# Read table data

df_students = pd.read_sql_table("students", engine)
df_students
```

```
   id student_name  age
0   1         arun   29
1   2        varun   20
2   3         neha   22
3   4        nisha   35
4   5       robert   38
5   6      michael   27
6   7      gillian   52
7   8       graeme   49
8   9        rohan   22
```

```
9   10       robert   38
10  11      michael   27
11  12      gillian   52
12  13       graeme   49
13  14        rohan   22
14  15       robert   38
15  16      michael   27
16  17      gillian   52
17  18       graeme   49
18  19        rohan   22
```

```
# Fetching specific columns from table

df_students = pd.read_sql_table('students',
                                engine,
                                columns = ['student_name'])
df_students
```

```
   student_name
0          arun
1         varun
2          neha
3         nisha
4        robert
5       michael
6       gillian
7        graeme
8         rohan
9        robert
10      michael
11      gillian
12       graeme
13        rohan
14       robert
15      michael
16      gillian
17       graeme
18        rohan
```

```
# Execute a query
query ='''
select student_name, age from students where student_name='arun'
'''
df_query = pd.read_sql(query, engine)
df_query
```

```
  student_name  age
0         arun   29
```

```
#   Insert data to table
```

```
students = {
  "robert":38,
  "michael":27,
  "gillian":52,
  "graeme":49,
  "rohan": 22,
}
df = pd.DataFrame(students.items(), columns=['name', 'Age'])
df

# students.items() - This converts the dictionary into a list of
tuples:
# Output: [('robert', 38), ('michael', 27), ...]
# Each tuple contains (key, value) pairs from the dictionary
```

```
      name  Age
0   robert   38
1  michael   27
2  gillian   52
3   graeme   49
4    rohan   22
```

```
# Rename column name

df.rename(columns={
    'name':'student_name',
    'Age':'age'
}, inplace=True)
df
```

```
  student_name  age
0       robert   38
1      michael   27
2      gillian   52
3       graeme   49
4        rohan   22
```

```
# Once the column names are matching then we can insert the data.

df.to_sql(
    name = 'students',
    con = engine,
    index =False,
    if_exists = 'append'
)
```

```
5
```

```
df_students = pd.read_sql_table('students',
                                engine)
df_students.head()
```

```
   id student_name  age
0   1         arun   29
1   2        varun   20
2   3         neha   22
3   4        nisha   35
4   5       robert   38
```

```python
#    Common function to read table and query
# 'read_sql_table' or, 'read_sql_query' =read_sql()

# read_sql() to fetch table data

df_students_table = pd.read_sql('students', engine)
df_students_table.head()
```

```
   id student_name  age
0   1         arun   29
1   2        varun   20
2   3         neha   22
3   4        nisha   35
4   5       robert   38
```

```python
# read_sql() to fetch query data

query = '''
select student_name, age
from students
where student_name ='arun'
'''
df_students_query = pd.read_sql(query, engine)
df_students_query
```

```
  student_name  age
0         arun   29
```

```python
#   Working with MongoDB
# !pip install pymongo

# Create connection
from pymongo import MongoClient

client = MongoClient()
db = client.students_db
collection = db.students

from pymongo import MongoClient

# Connect to MongoDB
client = MongoClient()
db = client.students_db  # Database name
collection = db.students  # Collection name
```

```
# Insert a sample document (creates the collection automatically)
insert_result = collection.insert_one({"student_name": "Alice", "age":
24, "grade": "A"})

# Simply return/print the collection object
print(collection)

# Or to verify the collection exists (without updating anything)
print("Collections in 'students_db':", db.list_collection_names())
```

```
Collection(Database(MongoClient(host=['localhost:27017'],
document_class=dict, tz_aware=False, connect=True), 'students_db'),
'students')
Collections in 'students_db': ['students']
```

```
import pandas as pd
from pymongo import MongoClient

# Connect and query with limit
collection = MongoClient().students_db.students
data = pd.DataFrame(list(collection.find().limit(1)))  # ← Only get 1
document

print(data.head())  # Shows first 5 rows
```

```
                        _id student_name  age grade
0  6824a5e70428e16ab6d6c92f        Alice   24     A
```

```
# Fetching specific columns

data = pd.DataFrame(list(
    collection.find({}, {'age': 1, '_id': 0}).limit(2)  # ← Limits to
2 docs at DB level
))
print(data)
```

```
   age
0   24
1   24
```

```
# Insert records

students = [
   {"student_name":"robert", "age":38},
   {"student_name":"michael", "age":27},
   {"student_name":"gillian", "age":52},
   {"student_name":"graeme", "age":49},
   {"student_name":"rohan", "age":22},
]
collection.insert_many(students)
```

```
InsertManyResult([ObjectId('6824a64672eda7ac2bb4752d'),
ObjectId('6824a64672eda7ac2bb4752e'),
ObjectId('6824a64672eda7ac2bb4752f'),
ObjectId('6824a64672eda7ac2bb47530'),
ObjectId('6824a64672eda7ac2bb47531')], acknowledged=True)
```

```
collection.delete_many({
    "$or": [
        {"student_name": {"$ne": "Alice"}},
        {"student_name": {"$exists": False}}
    ]
})
```

```
DeleteResult({'n': 5, 'ok': 1.0}, acknowledged=True)
```

```
#   Delete records
data = pd.DataFrame(list(collection.find().limit(1)))
display(data)
```

```
                        _id student_name  age grade
0  6824a5e70428e16ab6d6c92f        Alice   24     A
```