```
In [10]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import glob
         import re
         import math
```

```
In [11]: # CREATING AN ARRAY
```

```
In [12]: a = np.array([1,2,3,4,5,6,7])
         a

Out[12]: array([1, 2, 3, 4, 5, 6, 7])
```

```
In [13]: # CREATING A SERIES FROM THE NUMPY ARRAY
```

```
In [14]: a1 = pd.Series(a) # this gives the labels of the values or the index values
         a1

Out[14]: 0    1
         1    2
         2    3
         3    4
         4    5
         5    6
         6    7
         dtype: int32
```

```
In [15]: # DATATYPE OF SERIES
```

```
In [16]: a1.dtype

Out[16]: dtype('int32')
```

```
In [17]: # CHECK HOW MANY BYTES OF MEMORY DOES THE SERIES TAKE UP
```

```
In [18]: a1.nbytes

Out[18]: 28
```

```
In [19]: # CHECKING THE DIMENSION / SHAPE OF THE SERIES
```

```
In [20]: a1.shape

Out[20]: (7,)
```

```
In [21]: # CHECKING THE NUMBER OF DIMENSIONS
```

```
In [22]: a1.ndim

Out[22]: 1
```

```
In [23]: # CHECKING THE LENGTH OF THE SERIES
```

In [24]:
```
len(a1)
```

Out[24]:
```
7
```

In [25]:
```
# CHECKING HOW MANY VALUES ARE THERE IN THE SERIES
```

In [26]:
```
a1.count()
```

Out[26]:
```
7
```

In [27]:
```
# CHECKING THE SIZE OF THE SERIES
```

In [28]:
```
a1.size
```

Out[28]:
```
7
```

In [29]:
```
# CREATING A SERIES FROM THE LIST
```

In [30]:
```
b = [1,2,3]
b
```

Out[30]:
```
[1, 2, 3]
```

In [31]:
```
b1 = pd.Series(b, index = ['a', 'b', 'c'])
b1 # we are assigning different index values to the values in the list
```

Out[31]:
```
a    1
b    2
c    3
dtype: int64
```

In [32]:
```
# MODIFYING INDEX IN SERIES
```

In [33]:
```
X = np.array(['a','b','c','d','e','f','g'])
a1.index = X
a1
```

Out[33]:
```
a    1
b    2
c    3
d    4
e    5
f    6
g    7
dtype: int32
```

In [34]:
```
# CREATING SERIES USING RANDOM AND RANGE FUNCTIONS
```

In [35]:
```
c = np.random.random(10) # print random values from 1 to 9 (n-1=10-1=9)
c1 = np.arange(0,10) # print range of numbers from 0 to 9 (n-1=10-1=9)
c2 = pd.Series(c,c1)
c
```

Out[35]:
```
array([0.4586024 , 0.4378271 , 0.68681806, 0.11715971, 0.8870549 ,
       0.13624054, 0.32513687, 0.70583831, 0.69177117, 0.51753175])
```

In [36]:
```
c1,c
```

Out[36]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 array([0.4586024 , 0.4378271 , 0.68681806, 0.11715971, 0.8870549 ,
        0.13624054, 0.32513687, 0.70583831, 0.69177117, 0.51753175]))
```

In [37]:
```
d = {'a1':10, 'a2':20, 'a3':30, 'a4':40}
d1 = pd.Series(d)
d1
```

Out[37]:
```
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

In [38]:
```
pd.Series(99, index = [0,1,2,3,4,5])
```

Out[38]:
```
0    99
1    99
2    99
3    99
4    99
5    99
dtype: int64
```

1) DIFFERENT THINGS ON SERIES WITH PANDAS

1.1 Slicing Series

In [41]:
```
c2
```

Out[41]:
```
0    0.458602
1    0.437827
2    0.686818
3    0.117160
4    0.887055
5    0.136241
6    0.325137
7    0.705838
8    0.691771
9    0.517532
dtype: float64
```

In [42]:
```
# RETURN ALL THE VALUES OF THE SERIES
```

In [43]:
```
c2[:]
```

Out[43]:
```
0    0.458602
1    0.437827
2    0.686818
3    0.117160
4    0.887055
5    0.136241
6    0.325137
7    0.705838
8    0.691771
9    0.517532
dtype: float64
```

In [44]:
```
# RETURN FIRST 3 VALUES OF THE SERIES
```

In [45]:
```
c2[0:3]
```

Out[45]:
```
0    0.458602
1    0.437827
2    0.686818
dtype: float64
```

In [46]:
```
# RETURN LAST VALUE OF THE SERIES
```

In [47]:
```
c2[-1:]
```

Out[47]:
```
9    0.517532
dtype: float64
```

In [48]:
```
# RETURN FIRST 4 VALUES OF THE SERIES
```

In [49]:
```
c2[:4]
```

Out[49]:
```
0    0.458602
1    0.437827
2    0.686818
3    0.117160
dtype: float64
```

In [50]:
```
# RETURN ALL VALUES OF THE SERIES EXCEPT LAST 2 VALUES
```

In [51]:
```
c2[:-2]
```

Out[51]:
```
0    0.458602
1    0.437827
2    0.686818
3    0.117160
4    0.887055
5    0.136241
6    0.325137
7    0.705838
dtype: float64
```

In [52]:
```
# RETURN ALL VALUES OF THE SERIES EXCEPT LAST 1 VALUE
```

In [53]:
```
c2[:-1]
```

```
Out[53]: 0    0.458602
         1    0.437827
         2    0.686818
         3    0.117160
         4    0.887055
         5    0.136241
         6    0.325137
         7    0.705838
         8    0.691771
         dtype: float64
```

```
In [54]: # RETURN LAST 2 VALUES OF THE SERIES
```

```
In [55]: c2[-2:]
```

```
Out[55]: 8    0.691771
         9    0.517532
         dtype: float64
```

```
In [56]: # RETURN LAST VALUE OF THE SERIES
```

```
In [57]: c2[-1:]
```

```
Out[57]: 9    0.517532
         dtype: float64
```

```
In [58]: c2[-3:-1]
```

```
Out[58]: 7    0.705838
         8    0.691771
         dtype: float64
```

1.2 Append Series

```
In [60]: e = a1.copy()
         e
```

```
Out[60]: a    1
         b    2
         c    3
         d    4
         e    5
         f    6
         g    7
         dtype: int32
```

```
In [61]: d1
```

```
Out[61]: a1    10
         a2    20
         a3    30
         a4    40
         dtype: int64
```

```
In [62]: # APPEND 2 SERIES
```

In [63]:

```
d2 = pd.concat([e,d1])
d2
```

Out[63]:

```
a      1
b      2
c      3
d      4
e      5
f      6
g      7
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

In [64]:

```
# WHEN "inplace = False" IT WILL RETURN A NEW COPY OF DATA WITH THE OPERATION PE
```

In [65]:

```
d2.drop('a4', inplace = False)
```

Out[65]:

```
a      1
b      2
c      3
d      4
e      5
f      6
g      7
a1    10
a2    20
a3    30
dtype: int64
```

In [66]:

```
d2
```

Out[66]:

```
a      1
b      2
c      3
d      4
e      5
f      6
g      7
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

In [67]:

```
# WHEN WE USE "inplace = True" IT WILL AFFECT THE DATAFRAME
```

In [68]:

```
d2.drop('a4', inplace = True)
d2
```

Out[68]:

```
a       1
b       2
c       3
d       4
e       5
f       6
g       7
a1     10
a2     20
a3     30
dtype: int64
```

In [69]:

```
d2 = pd.concat([d2, pd.Series({'a4': 7})])
d2
```

Out[69]:

```
a       1
b       2
c       3
d       4
e       5
f       6
g       7
a1     10
a2     20
a3     30
a4      7
dtype: int64
```

1.3 Operation on Series

In [71]:

```
v1 = np.array([10,20,30])
v2 = np.array([1,2,3])
s1 = pd.Series(v1)
s2 = pd.Series(v2)
s1, s2
```

Out[71]:

```
(0    10
 1    20
 2    30
 dtype: int32,
 0    1
 1    2
 2    3
 dtype: int32)
```

In [72]:

```
# ADDITION OF 2 SERIES
```

In [73]:

```
s1.add(s2)
```

Out[73]:

```
0    11
1    22
2    33
dtype: int32
```

In [74]:

```
# SUBTRACTION OF 2 SERIES
```

In [75]:

```
s1.sub(s2)
```

```
Out[75]: 0     9
         1    18
         2    27
         dtype: int32
```

```
In [76]: # INCREMENT ALL NUMBERS IN A SERIES BY 9
```

```
In [77]: s1.add(9)
```

```
Out[77]: 0    19
         1    29
         2    39
         dtype: int32
```

```
In [78]: # MULTIPLICATION OF 2 SERIES
```

```
In [79]: s1.mul(s2)
```

```
Out[79]: 0    10
         1    40
         2    90
         dtype: int32
```

```
In [80]: # MULTIPLY EACH ELEMENT BY 1000
```

```
In [81]: s1.mul(1000)
```

```
Out[81]: 0    10000
         1    20000
         2    30000
         dtype: int32
```

```
In [82]: # DIVISION OF 2 SERIES
```

```
In [83]: s1.div(s2)
```

```
Out[83]: 0    10.0
         1    10.0
         2    10.0
         dtype: float64
```

```
In [84]: # MAX NUMBER IN A SERIES
```

```
In [85]: s1.max()
```

```
Out[85]: 30
```

```
In [86]: # MIN NUMBER IN A SERIES
```

```
In [87]: s1.min()
```

```
Out[87]: 10
```

```
In [88]: # FIND MEAN OF THE SERIES
```

```
In [89]: s1.mean()
```

Out[89]:
```
20.0
```

In [90]:
```
# FIND MEDIAN OF THE SERIES
```

In [91]:
```
s1.median()
```

Out[91]:
```
20.0
```

In [92]:
```
# FIND STANDARD DEVIATION OF THE SERIES
```

In [93]:
```
s1.std()
```

Out[93]:
```
10.0
```

In [94]:
```
# COMPARING 2 SERIES
```

In [95]:
```
s1.equals(s2)
```

Out[95]:
```
False
```

In [96]:
```
s4 = s1
```

In [97]:
```
s1.equals(s4)
```

Out[97]:
```
True
```

In [98]:
```
s5 = pd.Series([1,1,2,2,3,3], index = [0,1,2,3,4,5])
s5
```

Out[98]:
```
0    1
1    1
2    2
3    2
4    3
5    3
dtype: int64
```

In [99]:
```
# FIND THE FREQUENCY OF THE VALUES IN THE SERIES (REPAEATING)
```

In [100…
```
s5.value_counts()
```

Out[100…
```
1    2
2    2
3    2
Name: count, dtype: int64
```

2) DATAFRAME

2.1 Create DataFrame

In [103…
```
df = pd.DataFrame()
df
```

Out[103...

```
—
```

In [104...

```
# CREATE DATAFRAME USING LIST
```

In [105...

```
l = ['Java', 'Pyhton', 'C', 'C++']
df = pd.DataFrame(l)
df
```

Out[105...

	0
0	Java
1	Pyhton
2	C
3	C++

In [106...

```
# ADD COLUMN IN THE DATAFRAME
```

In [107...

```
rating = [1,2,3,4]
df[1] = rating
df
```

Out[107...

	0	1
0	Java	1
1	Pyhton	2
2	C	3
3	C++	4

In [108...

```
df.columns = ['Language', 'Rating']
df
```

Out[108...

	Language	Rating
0	Java	1
1	Pyhton	2
2	C	3
3	C++	4

In [109...

```
# CREATING DATAFRAME USING DICT
```

In [110...

```
data = [{'a':1, 'b':2}, {'a':5, 'b':10, 'c':20}]
df2  = pd.DataFrame(data)
df3  = pd.DataFrame(data, index=['row1', 'row2'], columns = ['a','b'])
df4  = pd.DataFrame(data, index=['row1', 'row2'], columns = ['a','b','c'])
df5  = pd.DataFrame(data, index=['row1', 'row2'], columns = ['a','b','c','d'])
```

In [111...

```
df2
```

Out[111…

```
   a   b     c
0  1   2   NaN
1  5  10  20.0
```

In [112…

```
df3
```

Out[112…

```
      a   b
row1  1   2
row2  5  10
```

In [113…

```
df4
```

Out[113…

```
      a   b     c
row1  1   2   NaN
row2  5  10  20.0
```

In [114…

```
df5
```

Out[114…

```
      a   b     c    d
row1  1   2   NaN  NaN
row2  5  10  20.0  NaN
```

In [115…

```
df0 = pd.DataFrame({'ID':[1,2,3,4], 'Name':['Aryan','Nayan','John','Rose']})
df0
```

Out[115…

```
   ID   Name
0   1  Aryan
1   2  Nayan
2   3   John
3   4   Rose
```

In [116…

```
# CREATING DATAFRAME FROM DICT OF SERIES
```

In [117…

```
dict = {'A':pd.Series([1,2,3,], index = ['a','b','c']),
        'B':pd.Series([1,2,3,4], index = ['a','b','c','d'])}
df1 = pd.DataFrame(dict)
df1
```

Out[117…

	A	B
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

2.2 Dataframe of Random Numbers with Date Indices

In [119…

```
dates = pd.date_range(start = '2024-11-20', end = '2024-11-26')
dates
```

Out[119…

```
DatetimeIndex(['2024-11-20', '2024-11-21', '2024-11-22', '2024-11-23',
               '2024-11-24', '2024-11-25', '2024-11-26'],
              dtype='datetime64[ns]', freq='D')
```

In [120…

```
dates = pd.date_range('today',periods= 7)
dates
```

Out[120…

```
DatetimeIndex(['2024-11-26 12:49:43.980186', '2024-11-27 12:49:43.980186',
               '2024-11-28 12:49:43.980186', '2024-11-29 12:49:43.980186',
               '2024-11-30 12:49:43.980186', '2024-12-01 12:49:43.980186',
               '2024-12-02 12:49:43.980186'],
              dtype='datetime64[ns]', freq='D')
```

In [121…

```
dates = pd.date_range(start='2024-11-26',periods= 7)
dates
```

Out[121…

```
DatetimeIndex(['2024-11-26', '2024-11-27', '2024-11-28', '2024-11-29',
               '2024-11-30', '2024-12-01', '2024-12-02'],
              dtype='datetime64[ns]', freq='D')
```

In [122…

```
M = np.random.random((7,7))
M
```

Out[122…

```
array([[0.23059865, 0.37026093, 0.02115781, 0.11875937, 0.76520621,
        0.70393982, 0.36832157],
       [0.96671514, 0.40788785, 0.30748419, 0.99563924, 0.11773271,
        0.33679117, 0.71054754],
       [0.35877304, 0.0439565 , 0.53890988, 0.76264063, 0.8057598 ,
        0.78257715, 0.07697249],
       [0.736959  , 0.38881012, 0.81017295, 0.44876564, 0.68971536,
        0.70504611, 0.16506348],
       [0.309816  , 0.75046975, 0.8519451 , 0.11609   , 0.90378549,
        0.2098155 , 0.19288771],
       [0.67873418, 0.26908651, 0.68202749, 0.64653277, 0.73612505,
        0.3682643 , 0.0269601 ],
       [0.57812265, 0.81999298, 0.83206236, 0.25218467, 0.00463752,
        0.75614716, 0.67058964]])
```

In [123…

```
dframe = pd.DataFrame(M, index = dates)
dframe
```

Out[123…

	0	1	2	3	4	5	6
2024-11-26	0.230599	0.370261	0.021158	0.118759	0.765206	0.703940	0.368322
2024-11-27	0.966715	0.407888	0.307484	0.995639	0.117733	0.336791	0.710548
2024-11-28	0.358773	0.043957	0.538910	0.762641	0.805760	0.782577	0.076972
2024-11-29	0.736959	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-30	0.309816	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-12-01	0.678734	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	0.578123	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [124…

```
# CHANGING COLUMN NAMES
```

In [125…

```
dframe.columns = ['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7']
dframe
```

Out[125…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	0.230599	0.370261	0.021158	0.118759	0.765206	0.703940	0.368322
2024-11-27	0.966715	0.407888	0.307484	0.995639	0.117733	0.336791	0.710548
2024-11-28	0.358773	0.043957	0.538910	0.762641	0.805760	0.782577	0.076972
2024-11-29	0.736959	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-30	0.309816	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-12-01	0.678734	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	0.578123	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [126…

```
# GETTING THE INDEX VALUES
```

In [127…

```
dframe.index
```

Out[127…

```
DatetimeIndex(['2024-11-26', '2024-11-27', '2024-11-28', '2024-11-29',
               '2024-11-30', '2024-12-01', '2024-12-02'],
              dtype='datetime64[ns]', freq='D')
```

In [128…

```
# GETTING THE COLUMN NAMES
```

In [129…

```
dframe.columns
```

Out[129…

```
Index(['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7'], dtype='object')
```

In [130…

```
# GETTING THE DATATYPE OF EACH COLUMN
```

In [131…

```
dframe.dtypes
```

Out[131…

```
C1    float64
C2    float64
C3    float64
C4    float64
C5    float64
C6    float64
C7    float64
dtype: object
```

In [132…

```
# SORT DATAFRAME BY COLUMN 'C1' IN ASCENDING ORDER
```

In [133…

```
dframe.sort_values(by = 'C1') # sorts the C1 values in ascending order
```

Out[133…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	0.230599	0.370261	0.021158	0.118759	0.765206	0.703940	0.368322
2024-11-30	0.309816	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-11-28	0.358773	0.043957	0.538910	0.762641	0.805760	0.782577	0.076972
2024-12-02	0.578123	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590
2024-12-01	0.678734	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-11-29	0.736959	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-27	0.966715	0.407888	0.307484	0.995639	0.117733	0.336791	0.710548

In [134…

```
# SORT DATAFRAME BY COLUMN 'C1' IN DESCENDING ORDER
```

In [135…

```
dframe.sort_values(by = 'C1', ascending = False) # sorts the C1 values in descea
```

Out[135…

	C1	C2	C3	C4	C5	C6	C7
2024-11-27	0.966715	0.407888	0.307484	0.995639	0.117733	0.336791	0.710548
2024-11-29	0.736959	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-12-01	0.678734	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	0.578123	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590
2024-11-28	0.358773	0.043957	0.538910	0.762641	0.805760	0.782577	0.076972
2024-11-30	0.309816	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-11-26	0.230599	0.370261	0.021158	0.118759	0.765206	0.703940	0.368322

2.3 Delete Column in DataFrame

In [137…

```
df1
```

Out[137…

	A	B
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

In [138…

```
del df1['B']
```

In [139…

```
df1
```

Out[139…

	A
a	1.0
b	2.0
c	3.0
d	NaN

In [140…

```
df5
```

Out[140…

	a	b	c	d
row1	1	2	NaN	NaN
row2	5	10	20.0	NaN

In [141…

```
# DELETE COLUMNS USING POP() FUNCTION
```

In [142…

```
df5.pop('c')
```

Out[142…

```
row1     NaN
row2    20.0
Name: c, dtype: float64
```

In [143…

```
df5
```

Out[143…

	a	b	d
row1	1	2	NaN
row2	5	10	NaN

2.4 Data Selection in Dataframe

In [145…

```
df
```

Out[145…

	Language	Rating
0	Java	1
1	Pyhton	2
2	C	3
3	C++	4

In [146…

```
df.index = [1,2,3,4]
df # changing the index values
```

Out[146…

	Language	Rating
1	Java	1
2	Pyhton	2
3	C	3
4	C++	4

In [147…

```
# DATA SELECTION USING ROW LABEL
```

In [148…

```
df.loc[1] # accessing using index values
```

Out[148…

```
Language    Java
Rating         1
Name: 1, dtype: object
```

In [149…

```
df.iloc[1] # accessing using just the values
```

Out[149…

```
Language    Pyhton
Rating           2
Name: 2, dtype: object
```

In [150…

```
df.loc[1:2]
```

Out[150…

	Language	Rating
1	Java	1
2	Pyhton	2

In [151…

```
df.iloc[1:2]
```

Out[151…

	Language	Rating
2	Pyhton	2

In [152…

```
# DATA SELECTION BASED ON CONDITION
```

In [153…

```
df.loc[df.Rating>2]
```

Out[153…

	Language	Rating
3	C	3
4	C++	4

In [154…

```
df1
```

Out[154…

	A
a	1.0
b	2.0
c	3.0
d	NaN

In [155…

```
df1.loc['a']
```

Out[155…

```
A    1.0
Name: a, dtype: float64
```

In [156…

```
dframe
```

Out[156…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	0.230599	0.370261	0.021158	0.118759	0.765206	0.703940	0.368322
2024-11-27	0.966715	0.407888	0.307484	0.995639	0.117733	0.336791	0.710548
2024-11-28	0.358773	0.043957	0.538910	0.762641	0.805760	0.782577	0.076972
2024-11-29	0.736959	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-30	0.309816	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-12-01	0.678734	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	0.578123	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [157…

```
# DATA SELECTION USING ROW LABEL
```

In [158…

```
dframe['2024-11-26':'2024-11-29']
```

Out[158…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	0.230599	0.370261	0.021158	0.118759	0.765206	0.703940	0.368322
2024-11-27	0.966715	0.407888	0.307484	0.995639	0.117733	0.336791	0.710548
2024-11-28	0.358773	0.043957	0.538910	0.762641	0.805760	0.782577	0.076972
2024-11-29	0.736959	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063

In [159…

```
# SELECTING ALL ROWS AND SELCTED COLUMNS
```

In [160…

```
dframe.loc[:, ['C1','C7']]
```

Out[160…

	C1	C7
2024-11-26	0.230599	0.368322
2024-11-27	0.966715	0.710548
2024-11-28	0.358773	0.076972
2024-11-29	0.736959	0.165063
2024-11-30	0.309816	0.192888
2024-12-01	0.678734	0.026960
2024-12-02	0.578123	0.670590

In [304…

```
# SELECTING ROWS AND COLUMNS BASED ON LABELS
```

In [308…

```
dframe.loc['2024-11-26':'2024-11-28', ['C1','C7']]
```

Out[308…

	C1	C7
2024-11-26	0.230599	0.368322
2024-11-27	0.966715	0.710548
2024-11-28	0.358773	0.076972

In [310…

```
# DATA SELECTION BASED ON CONDITION
```

In [314…

```
dframe[dframe['C1']>0.5] # returns the column values which are greter than 0.5
```

Out[314…

	C1	C2	C3	C4	C5	C6	C7
2024-11-27	0.966715	0.407888	0.307484	0.995639	0.117733	0.336791	0.710548
2024-11-29	0.736959	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-12-01	0.678734	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	0.578123	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [316…

```
# DATA SELECTION BASED ON CONDITION
```

In [318…

```
dframe[(dframe['C1']>0.5) & (dframe['C4']>0.5)] # returns the column values whic
```

Out[318…

	C1	C2	C3	C4	C5	C6	C7
2024-11-27	0.966715	0.407888	0.307484	0.995639	0.117733	0.336791	0.710548
2024-12-01	0.678734	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960

In [320…

```
# DATA SELECTION USING POSITION ( INTEGER INDEX BASED )
```

In [322…

```
dframe.iloc[0][0]
```

```
C:\Users\AKSHAY\AppData\Local\Temp\ipykernel_2552\1918434869.py:1: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  dframe.iloc[0][0]
```

Out[322…

```
0.2305986543638956
```

In [324…

```
# SELECT ALL ROWS AND FIRST 3 COLUMNSZ
```

In [326…

```
dframe.iloc[:,0:3]
```

Out[326…

	C1	C2	C3
2024-11-26	0.230599	0.370261	0.021158
2024-11-27	0.966715	0.407888	0.307484
2024-11-28	0.358773	0.043957	0.538910
2024-11-29	0.736959	0.388810	0.810173
2024-11-30	0.309816	0.750470	0.851945
2024-12-01	0.678734	0.269087	0.682027
2024-12-02	0.578123	0.819993	0.832062

In [330…

```
dframe.iloc[0,0] = 10 # changing the value to 10
```

In [332…

```
dframe
```

Out[332…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	10.000000	0.370261	0.021158	0.118759	0.765206	0.703940	0.368322
2024-11-27	0.966715	0.407888	0.307484	0.995639	0.117733	0.336791	0.710548
2024-11-28	0.358773	0.043957	0.538910	0.762641	0.805760	0.782577	0.076972
2024-11-29	0.736959	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-30	0.309816	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-12-01	0.678734	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	0.578123	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [334…

```
# DISPLAY ALL ROWS WHERE C1 HAS VALUE OF 10 OR 20
```

In [342…

```
dframe[dframe['C1'].isin([10])] # as there is 10 value it prints the row
```

Out[342…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	10.0	0.370261	0.021158	0.118759	0.765206	0.70394	0.368322

In [344…

```
dframe[dframe['C1'].isin([20])] # as there is no 20 value it prints nothing
```

Out[344…

	C1	C2	C3	C4	C5	C6	C7

2.5 Set Value

In [347…

```
# SET VALUE 888 FOR ALL ROWS IN COLUMN C1
```

In [349…

```
dframe['C1'] = 888
dframe
```

Out[349…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	0.021158	0.118759	0.765206	0.703940	0.368322
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	0.336791	0.710548
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	0.782577	0.076972
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-12-01	888	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [351…

```
# SET VALUE 777 FOR FIRST 3 ROWS IN COLUMN C6
```

In [367…

```
dframe.loc[dframe.index[0:3], 'C6'] = 777
dframe
```

Out[367…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	0.021158	0.118759	0.765206	777.000000	0.368322
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	777.000000	0.710548
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	0.076972
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-12-01	888	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [373…

```
# SET VALUE 333 IN 1st ROW AND 3rd COLUMN
```

In [371…

```
dframe.iat[0,2] = 333
dframe
```

Out[371…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	333.000000	0.118759	0.765206	777.000000	0.368322
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	777.000000	0.710548
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	0.076972
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-12-01	888	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [375…

```
# SET VALUE 555 IN 1st ROW AND 3rd COLUMN
```

In [381…

```
dframe.iloc[0,2] = 555
dframe
```

Out[381…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	555.000000	0.118759	0.765206	777.000000	0.368322
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	777.000000	0.710548
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	0.076972
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-12-01	888	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [383…

```
# CREATE COPY OF THE CALLING OBJECTS DATA ALONG WITH INDICES
```

In [393…

```
dframe1 = dframe.copy(deep = True)
dframe1
```

Out[393…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	555.000000	0.118759	0.765206	777.000000	0.368322
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	777.000000	0.710548
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	0.076972
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-12-01	888	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [397…

```
dframe1[(dframe1['C1']>0.5) & (dframe['C4']>0.5)] = 0
```

In [399…

```
dframe1[dframe1['C1'] == 0]
```

Out[399…

	C1	C2	C3	C4	C5	C6	C7
2024-11-27	0	0.0	0.0	0.0	0.0	0.0	0.0
2024-11-28	0	0.0	0.0	0.0	0.0	0.0	0.0
2024-12-01	0	0.0	0.0	0.0	0.0	0.0	0.0

In [401…

```
# REPLACE ZEROS IN C1 COLUMN WITH 99
```

In [405…

```
dframe1[dframe1['C1'].isin([0])] = 99
dframe1
```

Out[405…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	555.000000	0.118759	0.765206	777.000000	0.368322
2024-11-27	99	99.000000	99.000000	99.000000	99.000000	99.000000	99.000000
2024-11-28	99	99.000000	99.000000	99.000000	99.000000	99.000000	99.000000
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-12-01	99	99.000000	99.000000	99.000000	99.000000	99.000000	99.000000
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [407…

```
dframe
```

Out[407…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	555.000000	0.118759	0.765206	777.000000	0.368322
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	777.000000	0.710548
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	0.076972
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	0.165063
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	0.192888
2024-12-01	888	0.269087	0.682027	0.646533	0.736125	0.368264	0.026960
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	0.670590

In [409…

```
# DISPLAY ALL ROWS WHERE VALUE OF C1 IS 99
```

In [415…

```
dframe1[dframe1['C1'] == 99]
```

Out[415…

	C1	C2	C3	C4	C5	C6	C7	888	99
2024-11-27	99	99.0	99.0	99.0	99.0	99.0	99.0	99	99
2024-11-28	99	99.0	99.0	99.0	99.0	99.0	99.0	99	99
2024-12-01	99	99.0	99.0	99.0	99.0	99.0	99.0	99	99

2.6 Dealing with NULL Values

In [430…

```
dframe.loc[dframe.index[0:8],'C7'] = np.NaN
dframe.loc[dframe.index[0:2],'C6'] = np.NaN
dframe.loc[dframe.index[5:6],'C5'] = np.NaN
dframe # replaces value to NaN in the respective field
```

Out[430…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	555.000000	0.118759	0.765206	NaN	NaN
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	NaN	NaN
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	NaN
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	NaN
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	NaN
2024-12-01	888	0.269087	0.682027	0.646533	NaN	0.368264	NaN
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	NaN

In [432…

```
# DETECT NON-MISSING VALUES
```

In [438…

```
# IT WILL RETURN TRUE FOR NOT-NULL VALUES AND FALSE FOR NULL VALUES
dframe.notna()
```

Out[438…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	True	True	True	True	True	False	False
2024-11-27	True	True	True	True	True	False	False
2024-11-28	True	True	True	True	True	True	False
2024-11-29	True	True	True	True	True	True	False
2024-11-30	True	True	True	True	True	True	False
2024-12-01	True	True	True	True	False	True	False
2024-12-02	True	True	True	True	True	True	False

In [440…

```
# DETECT MISSING OR NULL VALUES
```

In [444…

```
# IT WILL RETURN TRUE FOR NULL VALUES AND FALSE FOR NOT-NULL VALUES
dframe.isna()
```

Out[444…

```
               C1     C2     C3     C4     C5     C6    C7
2024-11-26  False  False  False  False  False   True  True
2024-11-27  False  False  False  False  False   True  True
2024-11-28  False  False  False  False  False  False  True
2024-11-29  False  False  False  False  False  False  True
2024-11-30  False  False  False  False  False  False  True
2024-12-01  False  False  False  False   True  False  True
2024-12-02  False  False  False  False  False  False  True
```

In [446…

```
# FILL ALL NULL VALUES WITH 1020
```

In [448…

```
dframe = dframe.fillna(1020)
dframe
```

Out[448…

```
             C1        C2          C3        C4           C5           C6      C7
2024-11-26  888  0.370261  555.000000  0.118759     0.765206  1020.000000  1020.0
2024-11-27  888  0.407888    0.307484  0.995639     0.117733  1020.000000  1020.0
2024-11-28  888  0.043957    0.538910  0.762641     0.805760   777.000000  1020.0
2024-11-29  888  0.388810    0.810173  0.448766     0.689715     0.705046  1020.0
2024-11-30  888  0.750470    0.851945  0.116090     0.903785     0.209815  1020.0
2024-12-01  888  0.269087    0.682027  0.646533  1020.000000     0.368264  1020.0
2024-12-02  888  0.819993    0.832062  0.252185     0.004638     0.756147  1020.0
```

In [452…

```
dframe.loc[dframe.index[0:5] , 'C7'] = np.NaN
dframe.loc[dframe.index[0:2] , 'C6'] = np.NaN
dframe.loc[dframe.index[5:6] , 'C5'] = np.NaN
dframe # replaces value to NaN in the respective field
```

Out[452…

```
             C1        C2          C3        C4        C5          C6      C7
2024-11-26  888  0.370261  555.000000  0.118759  0.765206         NaN     NaN
2024-11-27  888  0.407888    0.307484  0.995639  0.117733         NaN     NaN
2024-11-28  888  0.043957    0.538910  0.762641  0.805760  777.000000     NaN
2024-11-29  888  0.388810    0.810173  0.448766  0.689715    0.705046     NaN
2024-11-30  888  0.750470    0.851945  0.116090  0.903785    0.209815     NaN
2024-12-01  888  0.269087    0.682027  0.646533       NaN    0.368264  1020.0
2024-12-02  888  0.819993    0.832062  0.252185  0.004638    0.756147  1020.0
```

In [454…

```
# REPLACE NULL VALUES IN C5 COLUMN WITH 123
# REPLACE NULL VALUES IN C6 COLUMN WITH 789
```

In [456…

```python
dframe.fillna(value = {'C5':123, 'C6':789})
```

Out[456…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	555.000000	0.118759	0.765206	789.000000	NaN
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	789.000000	NaN
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	NaN
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	NaN
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	NaN
2024-12-01	888	0.269087	0.682027	0.646533	123.000000	0.368264	1020.0
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	1020.0

In [458…

```python
# REPLACE 1st NULL VALUE IN C7 COLUMN WITH 789
```

In [462…

```python
dframe.fillna(value = {'C7':789}, limit = 1)
```

Out[462…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	555.000000	0.118759	0.765206	NaN	789.0
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	NaN	NaN
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	NaN
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	NaN
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	NaN
2024-12-01	888	0.269087	0.682027	0.646533	NaN	0.368264	1020.0
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	1020.0

In [464…

```python
# DROP ROWS WITH NULL VALUES
```

In [466…

```python
dframe.dropna()
```

Out[466…

	C1	C2	C3	C4	C5	C6	C7
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	1020.0

In [468…

```python
# DROP COLUMNS WITH NULL VALUES
```

In [470…

```python
dframe.dropna(axis = 'columns')
```

Out[470…

	C1	C2	C3	C4
2024-11-26	888	0.370261	555.000000	0.118759
2024-11-27	888	0.407888	0.307484	0.995639
2024-11-28	888	0.043957	0.538910	0.762641
2024-11-29	888	0.388810	0.810173	0.448766
2024-11-30	888	0.750470	0.851945	0.116090
2024-12-01	888	0.269087	0.682027	0.646533
2024-12-02	888	0.819993	0.832062	0.252185

In [472…

```
dframe
```

Out[472…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	555.000000	0.118759	0.765206	NaN	NaN
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	NaN	NaN
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	NaN
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	NaN
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	NaN
2024-12-01	888	0.269087	0.682027	0.646533	NaN	0.368264	1020.0
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	1020.0

In [474…

```
# DROP ROWS WITH NULL VALUES PRESENT IN C5 OR C6
```

In [478…

```
dframe.dropna(subset = ['C5','C6'])
```

Out[478…

	C1	C2	C3	C4	C5	C6	C7
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	NaN
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	NaN
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	NaN
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	1020.0

2.7 Descriptive Statistics

In [481…

```
# FILL NULL VALUES WITH 55
```

In [483…

```
dframe.fillna(55, inplace = True)
dframe
```

Out[483…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	555.000000	0.118759	0.765206	55.000000	55.0
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	55.000000	55.0
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	55.0
2024-11-29	888	0.388810	0.810173	0.448766	0.689715	0.705046	55.0
2024-11-30	888	0.750470	0.851945	0.116090	0.903785	0.209815	55.0
2024-12-01	888	0.269087	0.682027	0.646533	55.000000	0.368264	1020.0
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	1020.0

In [485…

```
# MEAN OF ALL COLUMNS
```

In [487…

```
dframe.mean()
```

Out[487…

```
C1    888.000000
C2      0.435781
C3     79.860372
C4      0.477230
C5      8.326691
C6    127.005610
C7    330.714286
dtype: float64
```

In [489…

```
# MAX VALUE IN ALL COLUMNS
```

In [491…

```
dframe.max()
```

Out[491…

```
C1     888.000000
C2       0.819993
C3     555.000000
C4       0.995639
C5      55.000000
C6     777.000000
C7    1020.000000
dtype: float64
```

In [493…

```
# MIN VALUE IN ALL COLUMNS
```

In [495…

```
dframe.min()
```

Out[495…

```
C1    888.000000
C2      0.043957
C3      0.307484
C4      0.116090
C5      0.004638
C6      0.209815
C7     55.000000
dtype: float64
```

In [497…

```
# MEDIAN OF ALL COLUMNS
```

In [501…

```
dframe.median()
```

```
Out[501…  C1    888.000000
          C2      0.388810
          C3      0.810173
          C4      0.448766
          C5      0.765206
          C6      0.756147
          C7     55.000000
          dtype: float64
```

```
In [503…  # STANDARD DEVIATION
```

```
In [505…  dframe.std()
```

```
Out[505…  C1      0.000000
          C2      0.269332
          C3    209.516973
          C4      0.338932
          C5     20.583993
          C6    287.769373
          C7    470.871785
          dtype: float64
```

```
In [507…  # VARIANCE
```

```
In [509…  dframe.var()
```

```
Out[509…  C1         0.000000
          C2         0.072540
          C3     43897.361977
          C4         0.114875
          C5       423.700787
          C6     82811.212152
          C7    221720.238095
          dtype: float64
```

```
In [521…  # LOWER QUARTILE / FIRST QUARTILE
```

```
In [523…  dframe.quantile(0.25)
```

```
Out[523…  C1    888.000000
          C2      0.319674
          C3      0.610469
          C4      0.185472
          C5      0.403724
          C6      0.536655
          C7     55.000000
          Name: 0.25, dtype: float64
```

```
In [525…  # SECOND QUARTILE / MEDIAN
          dframe.quantile(0.5)
```

```
Out[525…  C1    888.000000
          C2      0.388810
          C3      0.810173
          C4      0.448766
          C5      0.765206
          C6      0.756147
          C7     55.000000
          Name: 0.5, dtype: float64
```

In [527…

```
# UPPER QUARTILE
```

In [529…

```
dframe.quantile(0.75)
```

Out[529…

```
C1    888.000000
C2      0.579179
C3      0.842004
C4      0.704587
C5      0.854773
C6     55.000000
C7    537.500000
Name: 0.75, dtype: float64
```

In [532…

```
# INTER QUARTILE RANGE ( IQR )
```

In [534…

```
dframe.quantile(0.75) - dframe.quantile(0.25)
```

Out[534…

```
C1      0.000000
C2      0.259505
C3      0.231535
C4      0.519115
C5      0.451049
C6     54.463345
C7    482.500000
dtype: float64
```

In [536…

```
# SUM OF ALL COLUMN VALUES
```

In [538…

```
dframe.sum()
```

Out[538…

```
C1    6216.000000
C2       3.050465
C3     559.022602
C4       3.340612
C5      58.286837
C6     889.039273
C7    2315.000000
dtype: float64
```

In [540…

```
# DESCRIPTIVE STATS
```

In [542…

```
dframe.describe()
```

Out[542…

	C1	C2	C3	C4	C5	C6	C7
count	7.0	7.000000	7.000000	7.000000	7.000000	7.000000	7.000000
mean	888.0	0.435781	79.860372	0.477230	8.326691	127.005610	330.714286
std	0.0	0.269332	209.516973	0.338932	20.583993	287.769373	470.871785
min	888.0	0.043957	0.307484	0.116090	0.004638	0.209815	55.000000
25%	888.0	0.319674	0.610469	0.185472	0.403724	0.536655	55.000000
50%	888.0	0.388810	0.810173	0.448766	0.765206	0.756147	55.000000
75%	888.0	0.579179	0.842004	0.704587	0.854773	55.000000	537.500000
max	888.0	0.819993	555.000000	0.995639	55.000000	777.000000	1020.000000

In [544…

```
# RETURN UNBIASED SKEW
```

In [546…

```
dframe.skew()
```

Out[546…

```
C1    0.000000
C2    0.270947
C3    2.645747
C4    0.376582
C5    2.644128
C6    2.602297
C7    1.229634
dtype: float64
```

In [548…

```
# RETURN UNBIASED KURTOSIS USING FISHER'S DEFINITION OF KURTOSIS
```

In [550…

```
dframe.kurt()
```

Out[550…

```
C1    0.000000
C2   -0.436738
C3    6.999981
C4   -1.305932
C5    6.993488
C6    6.820295
C7   -0.840000
dtype: float64
```

In [552…

```
# CORRELATION
```

In [556…

```
dframe.corr()
```

Out[556…

	C1	C2	C3	C4	C5	C6	C7
C1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
C2	NaN	1.000000	-0.106762	-0.602143	-0.279090	-0.665163	0.275855
C3	NaN	-0.106762	1.000000	-0.467153	-0.161956	-0.110673	-0.257916
C4	NaN	-0.602143	-0.467153	1.000000	0.214558	0.393634	-0.056176
C5	NaN	-0.279090	-0.161956	0.214558	1.000000	-0.188719	0.636390
C6	NaN	-0.665163	-0.110673	0.393634	-0.188719	1.000000	-0.300162
C7	NaN	0.275855	-0.257916	-0.056176	0.636390	-0.300162	1.000000

In [558…

```
# COVARIANCE
```

In [560…

```
dframe.cov()
```

Out[560…

	C1	C2	C3	C4	C5	C6	
C1	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
C2	0.0	0.072540	-6.024524	-0.054967	-1.547253	-51.553837	34.9841
C3	0.0	-6.024524	43897.361977	-33.173448	-698.467410	-6672.788652	-25444.9034
C4	0.0	-0.054967	-33.173448	0.114875	1.496878	38.392788	-8.9653
C5	0.0	-1.547253	-698.467410	1.496878	423.700787	-1117.864061	6168.1602
C6	0.0	-51.553837	-6672.788652	38.392788	-1117.864061	82811.212152	-40672.6285
C7	0.0	34.984172	-25444.903450	-8.965369	6168.160259	-40672.628515	221720.2380

In [562…

```
import statistics as st
dframe.loc[dframe.index[3:6],'C1'] = 22
dframe
```

Out[562…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	555.000000	0.118759	0.765206	55.000000	55.0
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	55.000000	55.0
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	55.0
2024-11-29	22	0.388810	0.810173	0.448766	0.689715	0.705046	55.0
2024-11-30	22	0.750470	0.851945	0.116090	0.903785	0.209815	55.0
2024-12-01	22	0.269087	0.682027	0.646533	55.000000	0.368264	1020.0
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	1020.0

In [564…

```
# MEAN
```

In [566…

```
st.mean(dframe['C1'])
```

```
Out[566…   516.8571428571429

In [568…   # HORMONIC MEAN

In [570…   st.harmonic_mean(dframe['C1'])

Out[570…   49.69186046511628

In [572…   # MEDIAN

In [578…   arr = np.array([1,2,3,4,5,6,7,8])
           st.median(arr)

Out[578…   4.5

In [580…   # LOW MEDIAN OF THE DATA WITH EVEN LENGTH

In [582…   st.median_low(arr)

Out[582…   4

In [584…   # HIGH MEDIAN OF THE DATA WITH EVEN LENGTH

In [586…   st.median_high(arr)

Out[586…   5

In [588…   # MODE OF DATAFRAME

In [592…   st.mode(dframe['C7'])

Out[592…   55.0

In [594…   # SMAPLE VARIANCE

In [596…   st.variance(dframe['C1'])

Out[596…   214273.14285714287

In [598…   # POPULATION VARIANCE

In [600…   st.pvariance(dframe['C1'])

Out[600…   183662.693877551

In [602…   # STANDARD DEVIATION

In [608…   st.stdev(dframe['C1'])

Out[608…   462.89647099231905

In [610…   # POPULATION STANDARD DEVIATION

In [612…   st.pstdev(dframe['C1'])
```

Out[612…

```
428.55885695847076
```

3) APPLY FUNCTION ON DATAFRAME

In [617…

```python
dframe
```

Out[617…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888	0.370261	555.000000	0.118759	0.765206	55.000000	55.0
2024-11-27	888	0.407888	0.307484	0.995639	0.117733	55.000000	55.0
2024-11-28	888	0.043957	0.538910	0.762641	0.805760	777.000000	55.0
2024-11-29	22	0.388810	0.810173	0.448766	0.689715	0.705046	55.0
2024-11-30	22	0.750470	0.851945	0.116090	0.903785	0.209815	55.0
2024-12-01	22	0.269087	0.682027	0.646533	55.000000	0.368264	1020.0
2024-12-02	888	0.819993	0.832062	0.252185	0.004638	0.756147	1020.0

In [619…

```python
# FINDING MAX VALUE IN COLUMNS
```

In [621…

```python
dframe.apply(max)
```

Out[621…

```
C1     888.000000
C2       0.819993
C3     555.000000
C4       0.995639
C5      55.000000
C6     777.000000
C7    1020.000000
dtype: float64
```

In [623…

```python
# FINDING MIN VALUE IN COLUMNS
```

In [625…

```python
dframe.apply(min)
```

Out[625…

```
C1    22.000000
C2     0.043957
C3     0.307484
C4     0.116090
C5     0.004638
C6     0.209815
C7    55.000000
dtype: float64
```

In [627…

```python
# FINDING SUM OF COLUMN VALUES
```

In [629…

```python
dframe.apply(sum)
```

Out[629…

```
C1    3618.000000
C2       3.050465
C3     559.022602
C4       3.340612
C5      58.286837
C6     889.039273
C7    2315.000000
dtype: float64
```

In [631…

```
# FINDING SUM OF ROW VALUES
```

In [633…

```
dframe.apply(sum, axis = 1)
```

Out[633…

```
2024-11-26    1554.254227
2024-11-27     999.828744
2024-11-28    1722.151267
2024-11-29      80.042510
2024-11-30      79.832106
2024-12-01    1098.965911
2024-12-02    1910.665025
Freq: D, dtype: float64
```

In [635…

```
# SQUARE ROOT OF ALL VALUES IN DATAFRAME
```

In [643…

```
dframe.apply(np.sqrt)
```

Out[643…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	29.799329	0.608491	23.558438	0.344615	0.874761	7.416198	7.416198
2024-11-27	29.799329	0.638661	0.554513	0.997817	0.343122	7.416198	7.416198
2024-11-28	29.799329	0.209658	0.734105	0.873293	0.897641	27.874720	7.416198
2024-11-29	4.690416	0.623546	0.900096	0.669900	0.830491	0.839670	7.416198
2024-11-30	4.690416	0.866297	0.923009	0.340720	0.950676	0.458056	7.416198
2024-12-01	4.690416	0.518735	0.825850	0.804073	7.416198	0.606848	31.937439
2024-12-02	29.799329	0.905535	0.912175	0.502180	0.068099	0.869567	31.937439

In [647…

```
dframe.applymap(float)
```

```
C:\Users\AKSHAY\AppData\Local\Temp\ipykernel_2552\4228414899.py:1: FutureWarning:
DataFrame.applymap has been deprecated. Use DataFrame.map instead.
  dframe.applymap(float)
```

Out[647…

	C1	C2	C3	C4	C5	C6	C7
2024-11-26	888.0	0.370261	555.000000	0.118759	0.765206	55.000000	55.0
2024-11-27	888.0	0.407888	0.307484	0.995639	0.117733	55.000000	55.0
2024-11-28	888.0	0.043957	0.538910	0.762641	0.805760	777.000000	55.0
2024-11-29	22.0	0.388810	0.810173	0.448766	0.689715	0.705046	55.0
2024-11-30	22.0	0.750470	0.851945	0.116090	0.903785	0.209815	55.0
2024-12-01	22.0	0.269087	0.682027	0.646533	55.000000	0.368264	1020.0
2024-12-02	888.0	0.819993	0.832062	0.252185	0.004638	0.756147	1020.0

In [649…

```
# USING LAMBDA FUNCTION IN DATAFRAME TO FIND MIN VALUE
```

In [651…

```
dframe.apply(lambda x: min(x))
```

Out[651…

```
C1    22.000000
C2     0.043957
C3     0.307484
C4     0.116090
C5     0.004638
C6     0.209815
C7    55.000000
dtype: float64
```

In [653…

```
# USING LAMBDA DUNCTION IN DATAFRAME O FIND THE SQUARE OF ALL VALUES
```

In [655…

```
dframe.apply(lambda x: x*x)
```

Out[655…

	C1	C2	C3	C4	C5	C6	C
2024-11-26	788544	0.137093	308025.000000	0.014104	0.585541	3025.000000	3025.
2024-11-27	788544	0.166372	0.094547	0.991297	0.013861	3025.000000	3025.
2024-11-28	788544	0.001932	0.290424	0.581621	0.649249	603729.000000	3025.
2024-11-29	484	0.151173	0.656380	0.201391	0.475707	0.497090	3025.
2024-11-30	484	0.563205	0.725810	0.013477	0.816828	0.044023	3025.
2024-12-01	484	0.072408	0.465161	0.418005	3025.000000	0.135619	1040400.
2024-12-02	788544	0.672388	0.692328	0.063597	0.000022	0.571759	1040400.

3.1 Merge DataFrames

In [658…

```
daf1 = pd.DataFrame({'Id': ['1','2','3','4','5'], 'Name': ['Aryan','Rose','Bran'
daf1
```

Out[658…

	Id	Name
0	1	Aryan
1	2	Rose
2	3	Bran
3	4	Ronaldo
4	5	Craig

In [660…

```
daf2 = pd.DataFrame({'Id':['1','2','6','7','8'], 'Score':[40, 60, 80, 90, 70]})
daf2
```

Out[660…

	Id	Score
0	1	40
1	2	60
2	6	80
3	7	90
4	8	70

In [662…

```
# INNER JOIN
```

In [668…

```
pd.merge(daf1,daf2)
#pd.merge(daf1,daf2, on = 'Id', how = 'inner')
```

Out[668…

	Id	Name	Score
0	1	Aryan	40
1	2	Rose	60

In [670…

```
# FULL OUTER JOIN
```

In [672…

```
pd.merge(daf1, daf2, on = 'Id', how = 'outer')
```

Out[672...

	Id	Name	Score
0	1	Aryan	40.0
1	2	Rose	60.0
2	3	Bran	NaN
3	4	Ronaldo	NaN
4	5	Craig	NaN
5	6	NaN	80.0
6	7	NaN	90.0
7	8	NaN	70.0

In [674...

```
# LEFT OUTER JOIN
```

In [680...

```
pd.merge(daf1,daf2, on = 'Id', how = 'left') # only takes daf1 ids
```

Out[680...

	Id	Name	Score
0	1	Aryan	40.0
1	2	Rose	60.0
2	3	Bran	NaN
3	4	Ronaldo	NaN
4	5	Craig	NaN

In [682...

```
# RIGHT OUTER JOIN
```

In [684...

```
pd.merge(daf1,daf2, on = 'Id', how = 'right') # only takes daf2 ids
```

Out[684...

	Id	Name	Score
0	1	Aryan	40
1	2	Rose	60
2	6	NaN	80
3	7	NaN	90
4	8	NaN	70

In [ ]: