Pandas tutorial (Day-11)

this tutorial explains pandas

First install libraries

pip isntall numpy & pip install pandas

import libraries

In [3]:
```
import pandas as pd
import numpy as np
```

In [4]:
```
# Object creation series is a colum and shift enter give instat result

s = pd.Series([1,3, np.nan , 5,7,8,9])
s
```

Out[4]:
```
0    1.0
1    3.0
2    NaN
3    5.0
4    7.0
5    8.0
6    9.0
dtype: float64
```

In [5]:
```
dates = pd.date_range("20220101" , periods=20)
dates
```

Out[5]:
```
DatetimeIndex(['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04',
               '2022-01-05', '2022-01-06', '2022-01-07', '2022-01-08',
               '2022-01-09', '2022-01-10', '2022-01-11', '2022-01-12',
               '2022-01-13', '2022-01-14', '2022-01-15', '2022-01-16',
               '2022-01-17', '2022-01-18', '2022-01-19', '2022-01-20'],
              dtype='datetime64[ns]', freq='D')
```

In [6]:
```
df = pd.DataFrame(np.random.randn(20,4), index=dates, columns=list("ABCD"))
df
# yaha pe error aya ta wo islye k number of rows thek likny hain randn(rows, 
# yaha per (rows to pehly waly periods 20 hain or coulmn ABCD 4 hain)
# random array hamny banai jisme 20 rows hain date wise or 4 column hain, inde
```

Out[6]:

	A	B	C	D
2022-01-01	0.373736	0.480652	1.142781	-0.887483
2022-01-02	-1.491390	-0.786223	-0.365551	-1.228022
2022-01-03	-0.979261	-0.485096	1.289152	0.343405

	A	B	C	D
2022-01-04	-0.209437	-1.218131	-0.808479	-1.347075
2022-01-05	-0.943479	1.573199	-0.427349	0.474966
2022-01-06	0.514389	0.214347	-0.319681	0.407004
2022-01-07	-0.505157	-0.771696	0.302403	0.559630
2022-01-08	1.314176	-1.589684	-0.326924	0.011162
2022-01-09	1.828487	0.775517	-0.325338	1.500496
2022-01-10	-0.466249	-1.059650	-0.839302	1.019707
2022-01-11	0.147976	-1.935537	0.611068	-0.013197
2022-01-12	1.847803	0.063300	0.442557	0.784059
2022-01-13	0.340435	1.531455	3.095419	-0.212860
2022-01-14	-1.487640	0.765940	-0.620060	0.344466
2022-01-15	0.463026	-1.735056	0.558428	0.869181
2022-01-16	-0.714746	-0.869723	1.388101	1.039998
2022-01-17	0.030559	-1.371300	-0.525500	-0.016866
2022-01-18	-0.017294	-1.312327	0.301922	-0.655952
2022-01-19	0.322134	0.348061	-0.300517	0.035281

In [7]:
```
df2 =pd.DataFrame(
    {
        "A" : 1.0,
        "B" : pd.Timestamp("20220111"),
        "C" : pd.Series(1, index=list(range(4)), dtype="float32"),
        "D" : np.array([3] *4, dtype="int32"),
        "E" : pd.Categorical(["girl" , "woman", "girl" , "woman"]),
        "F" : "females",
    }
)
print(df2)
```

```
     A          B    C  D      E        F
0  1.0 2022-01-11  1.0  3   girl  females
1  1.0 2022-01-11  1.0  3  woman  females
2  1.0 2022-01-11  1.0  3   girl  females
3  1.0 2022-01-11  1.0  3  woman  females
```

In [8]:
```
df2.dtypes
```

Out[8]:
```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

In [9]:
```
df.head(2) # df.head or df.tail to check the data or see it
```

Out[9]:

	A	B	C	D
2022-01-01	0.373736	0.480652	1.142781	-0.887483
2022-01-02	-1.491390	-0.786223	-0.365551	-1.228022

In [10]:
```
df.index
```

Out[10]:
```
DatetimeIndex(['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04',
               '2022-01-05', '2022-01-06', '2022-01-07', '2022-01-08',
               '2022-01-09', '2022-01-10', '2022-01-11', '2022-01-12',
               '2022-01-13', '2022-01-14', '2022-01-15', '2022-01-16',
               '2022-01-17', '2022-01-18', '2022-01-19', '2022-01-20'],
              dtype='datetime64[ns]', freq='D')
```

In [11]:
```
df2.index
```

Out[11]:
```
Int64Index([0, 1, 2, 3], dtype='int64')
```

In [12]:
```
df.to_numpy()
```

Out[12]:
```
array([[ 0.37373584,  0.48065187,  1.14278146, -0.88748326],
       [-1.49138962, -0.78622307, -0.36555097, -1.22802181],
       [-0.97926141, -0.48509619,  1.28915245,  0.34340484],
       [-0.20943663, -1.21813095, -0.80847893, -1.34707479],
       [-0.9434791 ,  1.57319869, -0.42734917,  0.47496625],
       [ 0.51438936,  0.21434663, -0.31968084,  0.40700371],
       [-0.50515665, -0.77169606,  0.30240296,  0.55963032],
       [ 1.31417615, -1.58968426, -0.32692362,  0.01116192],
       [ 1.8284867 ,  0.77551729, -0.32533826,  1.5004957 ],
       [-0.46624932, -1.05965016, -0.83930161,  1.01970706],
       [ 0.14797604, -1.93553678,  0.61106767, -0.01319688],
       [ 1.84780343,  0.0633005 ,  0.4425571 ,  0.78405898],
       [ 0.34043479,  1.53145468,  3.09541933, -0.21286022],
       [-1.48763981,  0.76594041, -0.62006048,  0.34446556],
       [ 0.46302579, -1.73505573,  0.55842761,  0.86918089],
       [-0.71474615, -0.86972305,  1.38810051,  1.03999777],
       [ 0.03055866, -1.37129961, -0.52549951, -0.0168662 ],
       [-0.01729449, -1.31232687,  0.30192203, -0.65595249],
       [ 0.32213397,  0.34806054, -0.30051689,  0.03528095],
       [-1.01869603, -0.28482044, -0.8932819 ,  0.4615885 ]])
```

In [13]:
```
df2.to_numpy()
```

Out[13]:
```
array([[1.0, Timestamp('2022-01-11 00:00:00'), 1.0, 3, 'girl', 'females'],
       [1.0, Timestamp('2022-01-11 00:00:00'), 1.0, 3, 'woman',
        'females'],
       [1.0, Timestamp('2022-01-11 00:00:00'), 1.0, 3, 'girl', 'females'],
       [1.0, Timestamp('2022-01-11 00:00:00'), 1.0, 3, 'woman',
        'females']], dtype=object)
```

In [14]:
```
df.describe() #must use bracket in end
```

Out[14]:

	A	B	C	D
count	20.000000	20.000000	20.000000	20.000000
mean	-0.032531	-0.383339	0.168992	0.174474
std	0.964544	1.057586	0.986940	0.756587
min	-1.491390	-1.935537	-0.893282	-1.347075
25%	-0.771929	-1.241680	-0.451887	-0.065865
50%	0.006632	-0.628396	-0.310099	0.343935
75%	0.396058	0.381208	0.571588	0.615737
max	1.847803	1.573199	3.095419	1.500496

In [15]:

```
# we can take transpose or matrix
# change column to rows

df2.T
```

Out[15]:

	0	1	2	3
A	1.0	1.0	1.0	1.0
B	2022-01-11 00:00:00	2022-01-11 00:00:00	2022-01-11 00:00:00	2022-01-11 00:00:00
C	1.0	1.0	1.0	1.0
D	3	3	3	3
E	girl	woman	girl	woman
F	females	females	females	females

In [16]:

```
# we can sorting of data

df.sort_index(axis=0, ascending=True)
# here this ascending true and false change order of ABCD because of axis=1
# on zero axis it will do row wise
```

Out[16]:

	A	B	C	D
2022-01-01	0.373736	0.480652	1.142781	-0.887483
2022-01-02	-1.491390	-0.786223	-0.365551	-1.228022
2022-01-03	-0.979261	-0.485096	1.289152	0.343405
2022-01-04	-0.209437	-1.218131	-0.808479	-1.347075
2022-01-05	-0.943479	1.573199	-0.427349	0.474966
2022-01-06	0.514389	0.214347	-0.319681	0.407004
2022-01-07	-0.505157	-0.771696	0.302403	0.559630
2022-01-08	1.314176	-1.589684	-0.326924	0.011162

	A	B	C	D
2022-01-09	1.828487	0.775517	-0.325338	1.500496
2022-01-10	-0.466249	-1.059650	-0.839302	1.019707
2022-01-11	0.147976	-1.935537	0.611068	-0.013197
2022-01-12	1.847803	0.063300	0.442557	0.784059
2022-01-13	0.340435	1.531455	3.095419	-0.212860
2022-01-14	-1.487640	0.765940	-0.620060	0.344466
2022-01-15	0.463026	-1.735056	0.558428	0.869181
2022-01-16	-0.714746	-0.869723	1.388101	1.039998
2022-01-17	0.030559	-1.371300	-0.525500	-0.016866
2022-01-18	-0.017294	-1.312327	0.301922	-0.655952
2022-01-19	0.322134	0.348061	-0.300517	0.035281

In [17]:
```
# we can sort value wise as well
df.sort_index(axis=0, ascending=True)
```

Out[17]:

	A	B	C	D
2022-01-01	0.373736	0.480652	1.142781	-0.887483
2022-01-02	-1.491390	-0.786223	-0.365551	-1.228022
2022-01-03	-0.979261	-0.485096	1.289152	0.343405
2022-01-04	-0.209437	-1.218131	-0.808479	-1.347075
2022-01-05	-0.943479	1.573199	-0.427349	0.474966
2022-01-06	0.514389	0.214347	-0.319681	0.407004
2022-01-07	-0.505157	-0.771696	0.302403	0.559630
2022-01-08	1.314176	-1.589684	-0.326924	0.011162
2022-01-09	1.828487	0.775517	-0.325338	1.500496
2022-01-10	-0.466249	-1.059650	-0.839302	1.019707
2022-01-11	0.147976	-1.935537	0.611068	-0.013197
2022-01-12	1.847803	0.063300	0.442557	0.784059
2022-01-13	0.340435	1.531455	3.095419	-0.212860
2022-01-14	-1.487640	0.765940	-0.620060	0.344466
2022-01-15	0.463026	-1.735056	0.558428	0.869181
2022-01-16	-0.714746	-0.869723	1.388101	1.039998
2022-01-17	0.030559	-1.371300	-0.525500	-0.016866
2022-01-18	-0.017294	-1.312327	0.301922	-0.655952

	A	B	C	D
2022-01-19	0.322134	0.348061	-0.300517	0.035281

In [18]:
```
df.sort_values(by="B" , ascending=True)
```

Out[18]:

	A	B	C	D
2022-01-11	0.147976	-1.935537	0.611068	-0.013197
2022-01-15	0.463026	-1.735056	0.558428	0.869181
2022-01-08	1.314176	-1.589684	-0.326924	0.011162
2022-01-17	0.030559	-1.371300	-0.525500	-0.016866
2022-01-18	-0.017294	-1.312327	0.301922	-0.655952
2022-01-04	-0.209437	-1.218131	-0.808479	-1.347075
2022-01-10	-0.466249	-1.059650	-0.839302	1.019707
2022-01-16	-0.714746	-0.869723	1.388101	1.039998
2022-01-02	-1.491390	-0.786223	-0.365551	-1.228022
2022-01-07	-0.505157	-0.771696	0.302403	0.559630
2022-01-03	-0.979261	-0.485096	1.289152	0.343405
2022-01-20	-1.018696	-0.284820	-0.893282	0.461589
2022-01-12	1.847803	0.063300	0.442557	0.784059
2022-01-06	0.514389	0.214347	-0.319681	0.407004
2022-01-19	0.322134	0.348061	-0.300517	0.035281
2022-01-01	0.373736	0.480652	1.142781	-0.887483
2022-01-14	-1.487640	0.765940	-0.620060	0.344466
2022-01-09	1.828487	0.775517	-0.325338	1.500496
2022-01-13	0.340435	1.531455	3.095419	-0.212860
2022-01-05	-0.943479	1.573199	-0.427349	0.474966

In [19]:
```
df["A"] # we can do different selection by this method
```

Out[19]:
```
2022-01-01    0.373736
2022-01-02   -1.491390
2022-01-03   -0.979261
2022-01-04   -0.209437
2022-01-05   -0.943479
2022-01-06    0.514389
2022-01-07   -0.505157
2022-01-08    1.314176
2022-01-09    1.828487
2022-01-10   -0.466249
2022-01-11    0.147976
2022-01-12    1.847803
```

```
2022-01-13    0.340435
2022-01-14   -1.487640
2022-01-15    0.463026
2022-01-16   -0.714746
2022-01-17    0.030559
2022-01-18   -0.017294
2022-01-19    0.322134
2022-01-20   -1.018696
Freq: D, Name: A, dtype: float64
```

In [20]:
```
df[0:2] # row wise selection we can sort index wise also , which is date or r
```

Out[20]:

	A	B	C	D
2022-01-01	0.373736	0.480652	1.142781	-0.887483
2022-01-02	-1.491390	-0.786223	-0.365551	-1.228022

In [21]:
```
# Select by labels
df.loc[dates[0]]
# here its extracting data row wise
# here it extract data from date-wise or index wise

# agar 1 dalain to 2nd january ka data ayega
```

Out[21]:
```
A    0.373736
B    0.480652
C    1.142781
D   -0.887483
Name: 2022-01-01 00:00:00, dtype: float64
```

In [22]:
```
df.loc[: , ["A" , "B"]]
# here it does slicing column wise,
```

Out[22]:

	A	B
2022-01-01	0.373736	0.480652
2022-01-02	-1.491390	-0.786223
2022-01-03	-0.979261	-0.485096
2022-01-04	-0.209437	-1.218131
2022-01-05	-0.943479	1.573199
2022-01-06	0.514389	0.214347
2022-01-07	-0.505157	-0.771696
2022-01-08	1.314176	-1.589684
2022-01-09	1.828487	0.775517
2022-01-10	-0.466249	-1.059650
2022-01-11	0.147976	-1.935537
2022-01-12	1.847803	0.063300

	A	B
2022-01-13	0.340435	1.531455
2022-01-14	-1.487640	0.765940
2022-01-15	0.463026	-1.735056
2022-01-16	-0.714746	-0.869723
2022-01-17	0.030559	-1.371300
2022-01-18	-0.017294	-1.312327
2022-01-19	0.322134	0.348061

In [23]:
```
df.loc["20220102" : "20220104" , ["A" , "B"]]
# this gave cross section data of row and column
```

Out[23]:

	A	B
2022-01-02	-1.491390	-0.786223
2022-01-03	-0.979261	-0.485096
2022-01-04	-0.209437	-1.218131

In [24]:
```
df.loc[["20220102" , "20220104"] , ["A" , "B"]]
#  now see the difference this give data of only specific index
```

Out[24]:

	A	B
2022-01-02	-1.491390	-0.786223
2022-01-04	-0.209437	-1.218131

In [25]:
```
df.loc["20220102" , ["A" , "B", "C"]]
```

Out[25]:
```
A   -1.491390
B   -0.786223
C   -0.365551
Name: 2022-01-02 00:00:00, dtype: float64
```

In [26]:
```
df.at[dates[0], "A"]
```

Out[26]:
```
0.3737358373179265
```

In [27]:
```
df.iloc[3]
```

Out[27]:
```
A   -0.209437
B   -1.218131
C   -0.808479
D   -1.347075
Name: 2022-01-04 00:00:00, dtype: float64
```

In [28]:
```
df.iloc[3:10]
```

Out[28]:

	A	B	C	D
2022-01-04	-0.209437	-1.218131	-0.808479	-1.347075
2022-01-05	-0.943479	1.573199	-0.427349	0.474966
2022-01-06	0.514389	0.214347	-0.319681	0.407004
2022-01-07	-0.505157	-0.771696	0.302403	0.559630
2022-01-08	1.314176	-1.589684	-0.326924	0.011162
2022-01-09	1.828487	0.775517	-0.325338	1.500496
2022-01-10	-0.466249	-1.059650	-0.839302	1.019707

In [29]:
```
df.iloc[0:5 , 0:3]
```

Out[29]:

	A	B	C
2022-01-01	0.373736	0.480652	1.142781
2022-01-02	-1.491390	-0.786223	-0.365551
2022-01-03	-0.979261	-0.485096	1.289152
2022-01-04	-0.209437	-1.218131	-0.808479
2022-01-05	-0.943479	1.573199	-0.427349

In [30]:
```
# we can do implicity by keeping on side of colon empty
# df.iloc[0:5 , : ] , take all rows
# df.iloc[: , 0:3] , take 3 columns
df.iloc[: , 0:2]
```

Out[30]:

	A	B
2022-01-01	0.373736	0.480652
2022-01-02	-1.491390	-0.786223
2022-01-03	-0.979261	-0.485096
2022-01-04	-0.209437	-1.218131
2022-01-05	-0.943479	1.573199
2022-01-06	0.514389	0.214347
2022-01-07	-0.505157	-0.771696
2022-01-08	1.314176	-1.589684
2022-01-09	1.828487	0.775517
2022-01-10	-0.466249	-1.059650

	A	B
2022-01-11	0.147976	-1.935537
2022-01-12	1.847803	0.063300
2022-01-13	0.340435	1.531455
2022-01-14	-1.487640	0.765940
2022-01-15	0.463026	-1.735056
2022-01-16	-0.714746	-0.869723
2022-01-17	0.030559	-1.371300
2022-01-18	-0.017294	-1.312327
2022-01-19	0.322134	0.348061

In [42]:
```
df.iloc[0:5 , : ]
```

Out[42]:

	A	B	C	D
2022-01-01	0.373736	0.480652	1.142781	-0.887483
2022-01-02	-1.491390	-0.786223	-0.365551	-1.228022
2022-01-03	-0.979261	-0.485096	1.289152	0.343405
2022-01-04	-0.209437	-1.218131	-0.808479	-1.347075
2022-01-05	-0.943479	1.573199	-0.427349	0.474966

In [32]:
```
df[df["A"] > 0 ] # gives only those values of A non zero, bllen variable
```

Out[32]:

	A	B	C	D
2022-01-01	0.373736	0.480652	1.142781	-0.887483
2022-01-06	0.514389	0.214347	-0.319681	0.407004
2022-01-08	1.314176	-1.589684	-0.326924	0.011162
2022-01-09	1.828487	0.775517	-0.325338	1.500496
2022-01-11	0.147976	-1.935537	0.611068	-0.013197
2022-01-12	1.847803	0.063300	0.442557	0.784059
2022-01-13	0.340435	1.531455	3.095419	-0.212860
2022-01-15	0.463026	-1.735056	0.558428	0.869181
2022-01-17	0.030559	-1.371300	-0.525500	-0.016866
2022-01-19	0.322134	0.348061	-0.300517	0.035281

In [33]:
```
df[df >0] # gives non zero values of all, and some values will remain missing
```

Out[33]:

	A	B	C	D
2022-01-01	0.373736	0.480652	1.142781	NaN
2022-01-02	NaN	NaN	NaN	NaN
2022-01-03	NaN	NaN	1.289152	0.343405
2022-01-04	NaN	NaN	NaN	NaN
2022-01-05	NaN	1.573199	NaN	0.474966
2022-01-06	0.514389	0.214347	NaN	0.407004
2022-01-07	NaN	NaN	0.302403	0.559630
2022-01-08	1.314176	NaN	NaN	0.011162
2022-01-09	1.828487	0.775517	NaN	1.500496
2022-01-10	NaN	NaN	NaN	1.019707
2022-01-11	0.147976	NaN	0.611068	NaN
2022-01-12	1.847803	0.063300	0.442557	0.784059
2022-01-13	0.340435	1.531455	3.095419	NaN
2022-01-14	NaN	0.765940	NaN	0.344466
2022-01-15	0.463026	NaN	0.558428	0.869181
2022-01-16	NaN	NaN	1.388101	1.039998
2022-01-17	0.030559	NaN	NaN	NaN
2022-01-18	NaN	NaN	0.301922	NaN
2022-01-19	0.322134	0.348061	NaN	0.035281
2022-01-20	NaN	NaN	NaN	0.461589

In [34]:
```
# IS IN method
# we all add one more column

df3 =df.copy()
```

In [35]:
```
# add another new column
# df3 =df.copy()
# df3["Baloch"] = ["one", "one" , "two", "three" , "four", "three" ,
# "one", "one" , "two", "three" , "four", "three",
# "one", "one" , "two", "three" , "four", "three", "four", "three" ]
```

In [36]:
```
# df3["mean"]= [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
```

In [37]:
```
df3["sum"] = df3. sum(axis=1)
```

In [38]:
```
# now here we will have to exclude the column of sum otherwise it will take me

# df3['mean']=df3. mean(axis=1) this wont work

df3["mean"] = df3.iloc[:,0:4].mean(axis=1)
```

In [ ]:
```
# Python list slicing syntax states that for a:b it will get a and everything
# a: will get a and everything after it.
#  :b will get everything before b but not b.
# The list index of -1 refers to the last element.
# :-1 adheres to the same standards as above in that this gets everything bef
#  If you want the last element included use :.
```