pandas_withassignment

January 16, 2022

Title= “Mr”

Name= “Syed Saad ul Hassan”

email = “saadulhassanis@gmail.com”

whatsapp = “+491729024676”

Task: Pandas Hands on Practise

1. Libraries

Intalling libraries libraries
Importing libraries
Define a Series (a column list with a Not A Number)

[ ]:
```
# pip install numpy
# pip install pandas
import numpy as np
import pandas as pd
s = pd.Series([1,3,np.nan,5,7,8,9])
s
```

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

2. Generating Data Series/Frames

Printing Dates in a series

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

[ ]:
```
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')
```

Generating a Data Frame using the Dates as Index of that Data set

[ ]:
```
# np.random.randn(20,4) this indicates that keep index = 20 (row split) of␣
 ↪table, where as 4 is the column split
df = pd.DataFrame(data=np.random.randn(20,4), index=dates,␣
 ↪columns=list("ABCD"), dtype=float, copy=None)
df
```

[ ]:

	A	B	C	D
2022-01-01	-0.807823	-0.067819	-1.330167	0.074816
2022-01-02	0.470514	0.100841	1.017651	-1.245134
2022-01-03	-0.040778	1.192630	0.935529	-1.628175
2022-01-04	-0.493067	-0.660789	-0.601053	-1.287295
2022-01-05	0.689037	-0.448571	-0.712814	0.695055
2022-01-06	-1.302253	-1.658869	-1.811578	0.671019
2022-01-07	0.838272	0.111321	-0.809320	-0.475192
2022-01-08	-0.377271	0.510839	0.041828	-0.060369
2022-01-09	0.396088	1.338157	-0.759130	1.828155
2022-01-10	-0.557112	-0.273150	2.068674	0.741978
2022-01-11	0.550546	-0.565107	-0.085478	-0.423174
2022-01-12	-0.083672	-1.471417	-0.039479	0.111297
2022-01-13	0.074566	-0.125679	1.443004	0.034680
2022-01-14	1.265377	0.554316	1.188476	-2.156915
2022-01-15	1.275653	-0.101605	0.780956	0.675729
2022-01-16	-1.602749	0.251363	0.271529	2.246843
2022-01-17	-1.566494	0.545225	-0.388877	-0.317478
2022-01-18	1.251463	1.940221	-1.155982	-1.181810
2022-01-19	0.062309	-1.656761	0.531129	-1.674739
2022-01-20	1.019831	-0.298755	-0.044523	0.686477

Checking the data type of Data frame

[ ]:
```
df.dtypes
```

[ ]:
```
A    float64
B    float64
C    float64
D    float64
dtype: object
```

Generating a Data Frame using Dictionary Method (Key=Column names)

[ ]:
```
df2 = pd.DataFrame(
    {
        "A":2.5,
        "B":pd.Timestamp("20220114"),
        "C": pd.Series(1,index=list(range(4)),dtype="float32"),
        "D":np.array([3]*4, dtype="int32"),
        "E":pd.Categorical(["boy","baba","sakht londay","sigma male"]),
        "F": "Males",

    }

)
df2
```

[ ]:

	A	B	C	D	E	F
0	2.5	2022-01-14	1.0	3	boy	Males
1	2.5	2022-01-14	1.0	3	baba	Males
2	2.5	2022-01-14	1.0	3	sakht londay	Males
3	2.5	2022-01-14	1.0	3	sigma male	Males

[ ]:
```
df7 = pd.DataFrame(
    {
        "A":2.5,
        "B":pd.Timestamp("20220114"),
        "C": pd.Series(1,index=list(range(4)),dtype="float32"),
        "D":np.array([3]*4, dtype="int32"),
        "E":pd.Categorical(["boy","baba","sakht londay","sigma male"]),
        "F": "Males",

    }
    ,index=['first', 'second','third','four']

)
df7
```

[ ]:

	A	B	C	D	E	F
first	2.5	2022-01-14	NaN	3	boy	Males
second	2.5	2022-01-14	NaN	3	baba	Males
third	2.5	2022-01-14	NaN	3	sakht londay	Males
four	2.5	2022-01-14	NaN	3	sigma male	Males

[ ]:
```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df8 = pd.DataFrame(data, index=['first', 'second'])
print(df8)
```

```
        a   b     c
first   1   2   NaN
```

```
second  5  10  20.0
```

Checking the Data type of data frame created with Dictionary

[ ]:
```
df2.describe()
```

[ ]:
```
          A    C    D
count  4.0  4.0  4.0
mean   2.5  1.0  3.0
std    0.0  0.0  0.0
min    2.5  1.0  3.0
25%    2.5  1.0  3.0
50%    2.5  1.0  3.0
75%    2.5  1.0  3.0
max    2.5  1.0  3.0
```

[ ]:
```
df2.dtypes
```

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

Another Data Frame generation (Mapping according to columns)

[ ]:
```
data = [['Alex',10],['Bob',12],['Clarke',13]]
df6 = pd.DataFrame(data,columns=['Name','Age'])
print(df6)
```

```
     Name  Age
0    Alex   10
1     Bob   12
2  Clarke   13
```

[ ]:
```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df15 = pd.DataFrame(data)
print(df15)
```

```
    Name  Age
0    Tom   28
1   Jack   34
2  Steve   29
3  Ricky   42
```

Converting data frame (df) to numpy (Array)

[ ]:
```
f= df.to_numpy()
f
```

[ ]:
```
array([[-0.80782255, -0.06781884, -1.33016708,  0.07481607],
       [ 0.47051419,  0.10084056,  1.01765052, -1.24513377],
       [-0.04077768,  1.19262986,  0.93552934, -1.62817466],
       [-0.4930668 , -0.6607893 , -0.60105347, -1.28729493],
       [ 0.68903713, -0.4485706 , -0.71281443,  0.69505513],
       [-1.30225331, -1.65886928, -1.81157792,  0.67101941],
       [ 0.83827152,  0.11132093, -0.80932006, -0.47519209],
       [-0.37727078,  0.51083937,  0.04182803, -0.0603689 ],
       [ 0.39608832,  1.33815701, -0.75913005,  1.82815546],
       [-0.5571121 , -0.27315032,  2.0686735 ,  0.7419781 ],
       [ 0.55054643, -0.56510706, -0.08547801, -0.42317445],
       [-0.08367169, -1.47141736, -0.03947904,  0.11129695],
       [ 0.07456605, -0.12567864,  1.44300411,  0.03468045],
       [ 1.26537728,  0.5543164 ,  1.18847556, -2.15691471],
       [ 1.2756534 , -0.10160487,  0.78095598,  0.67572942],
       [-1.60274865,  0.25136267,  0.27152883,  2.24684335],
       [-1.56649381,  0.54522529, -0.3888765 , -0.31747769],
       [ 1.25146326,  1.94022066, -1.15598215, -1.18181022],
       [ 0.0623093 , -1.65676135,  0.53112893, -1.67473858],
       [ 1.01983136, -0.29875472, -0.04452312,  0.68647655]])
```

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

[ ]:
```
array([[2.5, Timestamp('2022-01-14 00:00:00'), 1.0, 3, 'boy', 'Males'],
       [2.5, Timestamp('2022-01-14 00:00:00'), 1.0, 3, 'baba', 'Males'],
       [2.5, Timestamp('2022-01-14 00:00:00'), 1.0, 3, 'sakht londay',
        'Males'],
       [2.5, Timestamp('2022-01-14 00:00:00'), 1.0, 3, 'sigma male',
        'Males']], dtype=object)
```

Transpose

[ ]:
```
# to transpose
df2.T
```

[ ]:
```
                    0                   1                   2  \
A                 2.5                 2.5                 2.5
B  2022-01-14 00:00:00  2022-01-14 00:00:00  2022-01-14 00:00:00
C                 1.0                 1.0                 1.0
D                   3                   3                   3
E                 boy                baba         sakht londay
F               Males               Males               Males

                    3
A                 2.5
```

```
B  2022-01-14 00:00:00
C                  1.0
D                    3
E           sigma male
F                Males
```

3. Sorting (Index Based) Row / Column heads only

Sorting Ascending/Descending Row index (row head) Wise

[ ]:
```
df.sort_index(axis=0, ascending=False)
```

[ ]:
```
                   A         B         C         D
2022-01-20  1.019831 -0.298755 -0.044523  0.686477
2022-01-19  0.062309 -1.656761  0.531129 -1.674739
2022-01-18  1.251463  1.940221 -1.155982 -1.181810
2022-01-17 -1.566494  0.545225 -0.388877 -0.317478
2022-01-16 -1.602749  0.251363  0.271529  2.246843
2022-01-15  1.275653 -0.101605  0.780956  0.675729
2022-01-14  1.265377  0.554316  1.188476 -2.156915
2022-01-13  0.074566 -0.125679  1.443004  0.034680
2022-01-12 -0.083672 -1.471417 -0.039479  0.111297
2022-01-11  0.550546 -0.565107 -0.085478 -0.423174
2022-01-10 -0.557112 -0.273150  2.068674  0.741978
2022-01-09  0.396088  1.338157 -0.759130  1.828155
2022-01-08 -0.377271  0.510839  0.041828 -0.060369
2022-01-07  0.838272  0.111321 -0.809320 -0.475192
2022-01-06 -1.302253 -1.658869 -1.811578  0.671019
2022-01-05  0.689037 -0.448571 -0.712814  0.695055
2022-01-04 -0.493067 -0.660789 -0.601053 -1.287295
2022-01-03 -0.040778  1.192630  0.935529 -1.628175
2022-01-02  0.470514  0.100841  1.017651 -1.245134
2022-01-01 -0.807823 -0.067819 -1.330167  0.074816
```

[ ]:
```
df.sort_index(axis=0, ascending=True)
```

[ ]:
```
                   A         B         C         D
2022-01-01 -0.807823 -0.067819 -1.330167  0.074816
2022-01-02  0.470514  0.100841  1.017651 -1.245134
2022-01-03 -0.040778  1.192630  0.935529 -1.628175
2022-01-04 -0.493067 -0.660789 -0.601053 -1.287295
2022-01-05  0.689037 -0.448571 -0.712814  0.695055
2022-01-06 -1.302253 -1.658869 -1.811578  0.671019
2022-01-07  0.838272  0.111321 -0.809320 -0.475192
2022-01-08 -0.377271  0.510839  0.041828 -0.060369
2022-01-09  0.396088  1.338157 -0.759130  1.828155
2022-01-10 -0.557112 -0.273150  2.068674  0.741978
2022-01-11  0.550546 -0.565107 -0.085478 -0.423174
```

```
2022-01-12 -0.083672 -1.471417 -0.039479  0.111297
2022-01-13  0.074566 -0.125679  1.443004  0.034680
2022-01-14  1.265377  0.554316  1.188476 -2.156915
2022-01-15  1.275653 -0.101605  0.780956  0.675729
2022-01-16 -1.602749  0.251363  0.271529  2.246843
2022-01-17 -1.566494  0.545225 -0.388877 -0.317478
2022-01-18  1.251463  1.940221 -1.155982 -1.181810
2022-01-19  0.062309 -1.656761  0.531129 -1.674739
2022-01-20  1.019831 -0.298755 -0.044523  0.686477
```

Sorting Ascending/Descending (Only Column Heads) not Values

[ ]:
```
df.sort_index(axis=1, ascending=False)
```

[ ]:
```
                   D         C         B         A
2022-01-01  0.074816 -1.330167 -0.067819 -0.807823
2022-01-02 -1.245134  1.017651  0.100841  0.470514
2022-01-03 -1.628175  0.935529  1.192630 -0.040778
2022-01-04 -1.287295 -0.601053 -0.660789 -0.493067
2022-01-05  0.695055 -0.712814 -0.448571  0.689037
2022-01-06  0.671019 -1.811578 -1.658869 -1.302253
2022-01-07 -0.475192 -0.809320  0.111321  0.838272
2022-01-08 -0.060369  0.041828  0.510839 -0.377271
2022-01-09  1.828155 -0.759130  1.338157  0.396088
2022-01-10  0.741978  2.068674 -0.273150 -0.557112
2022-01-11 -0.423174 -0.085478 -0.565107  0.550546
2022-01-12  0.111297 -0.039479 -1.471417 -0.083672
2022-01-13  0.034680  1.443004 -0.125679  0.074566
2022-01-14 -2.156915  1.188476  0.554316  1.265377
2022-01-15  0.675729  0.780956 -0.101605  1.275653
2022-01-16  2.246843  0.271529  0.251363 -1.602749
2022-01-17 -0.317478 -0.388877  0.545225 -1.566494
2022-01-18 -1.181810 -1.155982  1.940221  1.251463
2022-01-19 -1.674739  0.531129 -1.656761  0.062309
2022-01-20  0.686477 -0.044523 -0.298755  1.019831
```

[ ]:
```
df.sort_index(axis=1, ascending=True)
```

[ ]:
```
                   A         B         C         D
2022-01-01 -0.807823 -0.067819 -1.330167  0.074816
2022-01-02  0.470514  0.100841  1.017651 -1.245134
2022-01-03 -0.040778  1.192630  0.935529 -1.628175
2022-01-04 -0.493067 -0.660789 -0.601053 -1.287295
2022-01-05  0.689037 -0.448571 -0.712814  0.695055
2022-01-06 -1.302253 -1.658869 -1.811578  0.671019
2022-01-07  0.838272  0.111321 -0.809320 -0.475192
2022-01-08 -0.377271  0.510839  0.041828 -0.060369
2022-01-09  0.396088  1.338157 -0.759130  1.828155
```

```
2022-01-10 -0.557112 -0.273150  2.068674  0.741978
2022-01-11  0.550546 -0.565107 -0.085478 -0.423174
2022-01-12 -0.083672 -1.471417 -0.039479  0.111297
2022-01-13  0.074566 -0.125679  1.443004  0.034680
2022-01-14  1.265377  0.554316  1.188476 -2.156915
2022-01-15  1.275653 -0.101605  0.780956  0.675729
2022-01-16 -1.602749  0.251363  0.271529  2.246843
2022-01-17 -1.566494  0.545225 -0.388877 -0.317478
2022-01-18  1.251463  1.940221 -1.155982 -1.181810
2022-01-19  0.062309 -1.656761  0.531129 -1.674739
2022-01-20  1.019831 -0.298755 -0.044523  0.686477
```

Sorting a specified Column of Data Frame sorting its values

[ ]:
```
df.sort_values('B',axis=0, ascending=True )
```

[ ]:
```
                   A         B         C         D
2022-01-06 -1.302253 -1.658869 -1.811578  0.671019
2022-01-19  0.062309 -1.656761  0.531129 -1.674739
2022-01-12 -0.083672 -1.471417 -0.039479  0.111297
2022-01-04 -0.493067 -0.660789 -0.601053 -1.287295
2022-01-11  0.550546 -0.565107 -0.085478 -0.423174
2022-01-05  0.689037 -0.448571 -0.712814  0.695055
2022-01-20  1.019831 -0.298755 -0.044523  0.686477
2022-01-10 -0.557112 -0.273150  2.068674  0.741978
2022-01-13  0.074566 -0.125679  1.443004  0.034680
2022-01-15  1.275653 -0.101605  0.780956  0.675729
2022-01-01 -0.807823 -0.067819 -1.330167  0.074816
2022-01-02  0.470514  0.100841  1.017651 -1.245134
2022-01-07  0.838272  0.111321 -0.809320 -0.475192
2022-01-16 -1.602749  0.251363  0.271529  2.246843
2022-01-08 -0.377271  0.510839  0.041828 -0.060369
2022-01-17 -1.566494  0.545225 -0.388877 -0.317478
2022-01-14  1.265377  0.554316  1.188476 -2.156915
2022-01-03 -0.040778  1.192630  0.935529 -1.628175
2022-01-09  0.396088  1.338157 -0.759130  1.828155
2022-01-18  1.251463  1.940221 -1.155982 -1.181810
```

[ ]:
```
df.sort_values(by=['B', 'A'])
```

[ ]:
```
                   A         B         C         D
2022-01-06 -1.302253 -1.658869 -1.811578  0.671019
2022-01-19  0.062309 -1.656761  0.531129 -1.674739
2022-01-12 -0.083672 -1.471417 -0.039479  0.111297
2022-01-04 -0.493067 -0.660789 -0.601053 -1.287295
2022-01-11  0.550546 -0.565107 -0.085478 -0.423174
2022-01-05  0.689037 -0.448571 -0.712814  0.695055
2022-01-20  1.019831 -0.298755 -0.044523  0.686477
```

```
2022-01-10 -0.557112 -0.273150  2.068674  0.741978
2022-01-13  0.074566 -0.125679  1.443004  0.034680
2022-01-15  1.275653 -0.101605  0.780956  0.675729
2022-01-01 -0.807823 -0.067819 -1.330167  0.074816
2022-01-02  0.470514  0.100841  1.017651 -1.245134
2022-01-07  0.838272  0.111321 -0.809320 -0.475192
2022-01-16 -1.602749  0.251363  0.271529  2.246843
2022-01-08 -0.377271  0.510839  0.041828 -0.060369
2022-01-17 -1.566494  0.545225 -0.388877 -0.317478
2022-01-14  1.265377  0.554316  1.188476 -2.156915
2022-01-03 -0.040778  1.192630  0.935529 -1.628175
2022-01-09  0.396088  1.338157 -0.759130  1.828155
2022-01-18  1.251463  1.940221 -1.155982 -1.181810
```

4. Displaying Data in a Data frames

To Display an entire column

[ ]:
```
df["A"]
```

[ ]:
```
2022-01-01   -0.807823
2022-01-02    0.470514
2022-01-03   -0.040778
2022-01-04   -0.493067
2022-01-05    0.689037
2022-01-06   -1.302253
2022-01-07    0.838272
2022-01-08   -0.377271
2022-01-09    0.396088
2022-01-10   -0.557112
2022-01-11    0.550546
2022-01-12   -0.083672
2022-01-13    0.074566
2022-01-14    1.265377
2022-01-15    1.275653
2022-01-16   -1.602749
2022-01-17   -1.566494
2022-01-18    1.251463
2022-01-19    0.062309
2022-01-20    1.019831
Freq: D, Name: A, dtype: float64
```

To display selected rows

[ ]:
```
df[0:2]
```

[ ]:
```
                   A         B         C         D
2022-01-01 -0.807823 -0.067819 -1.330167  0.074816
```

```
2022-01-02  0.470514  0.100841  1.017651 -1.245134
```

[ ]:
```
# 2 indicates starting row for frames and 10 will print 10 index values
df[2:10]
```

[ ]:
```
                   A         B         C         D
2022-01-03 -0.040778  1.192630  0.935529 -1.628175
2022-01-04 -0.493067 -0.660789 -0.601053 -1.287295
2022-01-05  0.689037 -0.448571 -0.712814  0.695055
2022-01-06 -1.302253 -1.658869 -1.811578  0.671019
2022-01-07  0.838272  0.111321 -0.809320 -0.475192
2022-01-08 -0.377271  0.510839  0.041828 -0.060369
2022-01-09  0.396088  1.338157 -0.759130  1.828155
2022-01-10 -0.557112 -0.273150  2.068674  0.741978
```

Reaching a specific value in Table (Interpret it as 2D array indexing)

[ ]:
```
df.at[dates[5],"C"]
```

[ ]:
```
-1.8115779218408021
```

5. Targeted Index:Column data Filtration using Loc and ILoc functions

The main distinction between loc and iloc is: loc is label-based, which means that you have to specify rows and columns based on their row and column labels. iloc is integer position-based, so you have to specify rows and columns by their integer position values (0-based integer position).

This displays row 5 column values in vertical order

[ ]:
```
# row 5 ka column A,B,C,D parameters have been generated
df.loc[dates[5]]
```

[ ]:
```
A   -1.302253
B   -1.658869
C   -1.811578
D    0.671019
Name: 2022-01-06 00:00:00, dtype: float64
```

Display chunk of Data using loc command
limited operation on Columns as range cannot be defined like iloc

[ ]:
```
# row index (3 to 6) par only column A and B displayed
df.loc[dates[3:6],["A","C"]]
```

[ ]:
```
                   A         C
2022-01-04 -0.493067 -0.601053
2022-01-05  0.689037 -0.712814
2022-01-06 -1.302253 -1.811578
```

```
[ ]: # specific row and specific column
     df.loc[["20220105","20220107"],["A","C"]]
```

```
[ ]:                    A         C
    2022-01-05  0.689037 -0.712814
    2022-01-07  0.838272 -0.809320
```

Display chunk of Data using iloc command
Independant operation on Columns as range can be defined

```
[ ]: # another way of targeted filtration (row x column filters)
     # row bhe limited and coolumn bhe limited
     df.iloc[3:10,1:4]
```

```
[ ]:                    B         C         D
    2022-01-04 -0.660789 -0.601053 -1.287295
    2022-01-05 -0.448571 -0.712814  0.695055
    2022-01-06 -1.658869 -1.811578  0.671019
    2022-01-07  0.111321 -0.809320 -0.475192
    2022-01-08  0.510839  0.041828 -0.060369
    2022-01-09  1.338157 -0.759130  1.828155
    2022-01-10 -0.273150  2.068674  0.741978
```

Reaching a specific value in Table (Interpret it as 2D array indexing)

```
[ ]: df.at[dates[5],"C"]
```

```
[ ]: -1.8115779218408021
```

6. Condition (<,>) Checking

```
[ ]: df["A"]>1.5
```

```
[ ]: 2022-01-01    False
     2022-01-02    False
     2022-01-03    False
     2022-01-04    False
     2022-01-05    False
     2022-01-06    False
     2022-01-07    False
     2022-01-08    False
     2022-01-09    False
     2022-01-10    False
     2022-01-11    False
     2022-01-12    False
     2022-01-13    False
     2022-01-14    False
     2022-01-15    False
```

```
2022-01-16    False
2022-01-17    False
2022-01-18    False
2022-01-19    False
2022-01-20    False
Freq: D, Name: A, dtype: bool
```

6a. This is most important

you were facing error becoz of column ki data type and tmhe is se related google par bhe kuch nh mila tu ye yaad rakho

To sort column on the basis of a condition applied on a specific Column

[ ]:
```
df[df["A"] >0.1 ]
```

[ ]:
```
                   A         B         C         D
2022-01-02  0.470514  0.100841  1.017651 -1.245134
2022-01-05  0.689037 -0.448571 -0.712814  0.695055
2022-01-07  0.838272  0.111321 -0.809320 -0.475192
2022-01-09  0.396088  1.338157 -0.759130  1.828155
2022-01-11  0.550546 -0.565107 -0.085478 -0.423174
2022-01-14  1.265377  0.554316  1.188476 -2.156915
2022-01-15  1.275653 -0.101605  0.780956  0.675729
2022-01-18  1.251463  1.940221 -1.155982 -1.181810
2022-01-20  1.019831 -0.298755 -0.044523  0.686477
```

To ***display certain columns based on condition applied on another column***

[ ]:
```
criterion = df['A'].map(lambda x: x>0)
df.loc[criterion & (df['B'] > 0.3), 'C':'D']
```

[ ]:
```
                   C         D
2022-01-09 -0.759130  1.828155
2022-01-14  1.188476 -2.156915
2022-01-18 -1.155982 -1.181810
```

Hamesha yaad rakhna Saad k jab bhe bool milen tu loc se khel k real value get karna hay

To check multiple condition and display multiple values

[ ]:
```
s = (df['A'] > 0) & (df['B'] > 0)
print(s)
print("\n \n The sorted value that satisifies condition in A is")
e=df.loc[s]

df.loc[s,'A']
```

```
2022-01-01    False
2022-01-02     True
2022-01-03    False
2022-01-04    False
2022-01-05    False
2022-01-06    False
2022-01-07     True
2022-01-08    False
2022-01-09     True
2022-01-10    False
2022-01-11    False
2022-01-12    False
2022-01-13    False
2022-01-14     True
2022-01-15    False
2022-01-16    False
2022-01-17    False
2022-01-18     True
2022-01-19    False
2022-01-20    False
Freq: D, dtype: bool
```

The sorted value that satisifies condition in A is

[ ]:
```
2022-01-02    0.470514
2022-01-07    0.838272
2022-01-09    0.396088
2022-01-14    1.265377
2022-01-18    1.251463
Name: A, dtype: float64
```

[ ]:
```
e
```

[ ]:
```
                   A         B         C         D
2022-01-02  0.470514  0.100841  1.017651 -1.245134
2022-01-07  0.838272  0.111321 -0.809320 -0.475192
2022-01-09  0.396088  1.338157 -0.759130  1.828155
2022-01-14  1.265377  0.554316  1.188476 -2.156915
2022-01-18  1.251463  1.940221 -1.155982 -1.181810
```

To find values greater or less than a specific number

[ ]:
```
df[df>0]
```

[ ]:
```
                   A         B         C         D
2022-01-01       NaN       NaN       NaN  0.074816
2022-01-02  0.470514  0.100841  1.017651       NaN
```

```
2022-01-03       NaN  1.192630  0.935529       NaN
2022-01-04       NaN       NaN       NaN       NaN
2022-01-05  0.689037       NaN       NaN  0.695055
2022-01-06       NaN       NaN       NaN  0.671019
2022-01-07  0.838272  0.111321       NaN       NaN
2022-01-08       NaN  0.510839  0.041828       NaN
2022-01-09  0.396088  1.338157       NaN  1.828155
2022-01-10       NaN       NaN  2.068674  0.741978
2022-01-11  0.550546       NaN       NaN       NaN
2022-01-12       NaN       NaN       NaN  0.111297
2022-01-13  0.074566       NaN  1.443004  0.034680
2022-01-14  1.265377  0.554316  1.188476       NaN
2022-01-15  1.275653       NaN  0.780956  0.675729
2022-01-16       NaN  0.251363  0.271529  2.246843
2022-01-17       NaN  0.545225       NaN       NaN
2022-01-18  1.251463  1.940221       NaN       NaN
2022-01-19  0.062309       NaN  0.531129       NaN
2022-01-20  1.019831       NaN       NaN  0.686477
```

7. Adding/ Removing Data Columns and Recreating New Data Frame

Creating a new DF with old data frame and appending a new column

[ ]:
```
df3 = df.copy()
df3["E"]=["one","two","three","four","five",
"one","two","three","four","five","one","two","three","four","five","one","two","three","four"
df3
```

[ ]:
```
                   A         B         C         D      E
2022-01-01 -0.807823 -0.067819 -1.330167  0.074816    one
2022-01-02  0.470514  0.100841  1.017651 -1.245134    two
2022-01-03 -0.040778  1.192630  0.935529 -1.628175  three
2022-01-04 -0.493067 -0.660789 -0.601053 -1.287295   four
2022-01-05  0.689037 -0.448571 -0.712814  0.695055   five
2022-01-06 -1.302253 -1.658869 -1.811578  0.671019    one
2022-01-07  0.838272  0.111321 -0.809320 -0.475192    two
2022-01-08 -0.377271  0.510839  0.041828 -0.060369  three
2022-01-09  0.396088  1.338157 -0.759130  1.828155   four
2022-01-10 -0.557112 -0.273150  2.068674  0.741978   five
2022-01-11  0.550546 -0.565107 -0.085478 -0.423174    one
2022-01-12 -0.083672 -1.471417 -0.039479  0.111297    two
2022-01-13  0.074566 -0.125679  1.443004  0.034680  three
2022-01-14  1.265377  0.554316  1.188476 -2.156915   four
2022-01-15  1.275653 -0.101605  0.780956  0.675729   five
2022-01-16 -1.602749  0.251363  0.271529  2.246843    one
2022-01-17 -1.566494  0.545225 -0.388877 -0.317478    two
2022-01-18  1.251463  1.940221 -1.155982 -1.181810  three
```

```
2022-01-19  0.062309 -1.656761  0.531129 -1.674739   four
2022-01-20  1.019831 -0.298755 -0.044523  0.686477   five
```

Creating a reduced DF from a existing long data frame (data set)

[ ]:
```
df4=df3.iloc[:,0:4]
df4
```

[ ]:
```
                   A         B         C         D
2022-01-01 -0.807823 -0.067819 -1.330167  0.074816
2022-01-02  0.470514  0.100841  1.017651 -1.245134
2022-01-03 -0.040778  1.192630  0.935529 -1.628175
2022-01-04 -0.493067 -0.660789 -0.601053 -1.287295
2022-01-05  0.689037 -0.448571 -0.712814  0.695055
2022-01-06 -1.302253 -1.658869 -1.811578  0.671019
2022-01-07  0.838272  0.111321 -0.809320 -0.475192
2022-01-08 -0.377271  0.510839  0.041828 -0.060369
2022-01-09  0.396088  1.338157 -0.759130  1.828155
2022-01-10 -0.557112 -0.273150  2.068674  0.741978
2022-01-11  0.550546 -0.565107 -0.085478 -0.423174
2022-01-12 -0.083672 -1.471417 -0.039479  0.111297
2022-01-13  0.074566 -0.125679  1.443004  0.034680
2022-01-14  1.265377  0.554316  1.188476 -2.156915
2022-01-15  1.275653 -0.101605  0.780956  0.675729
2022-01-16 -1.602749  0.251363  0.271529  2.246843
2022-01-17 -1.566494  0.545225 -0.388877 -0.317478
2022-01-18  1.251463  1.940221 -1.155982 -1.181810
2022-01-19  0.062309 -1.656761  0.531129 -1.674739
2022-01-20  1.019831 -0.298755 -0.044523  0.686477
```

Calculating Mean on selected columns and generating a new Column (Assignment Qs)

[ ]:
```
df3['average'] = df3.iloc[:, [0,1,2,3]].mean(axis=1)
df3
```

[ ]:
```
                   A         B         C         D      E   average
2022-01-01 -0.807823 -0.067819 -1.330167  0.074816    one -0.532748
2022-01-02  0.470514  0.100841  1.017651 -1.245134    two  0.085968
2022-01-03 -0.040778  1.192630  0.935529 -1.628175  three  0.114802
2022-01-04 -0.493067 -0.660789 -0.601053 -1.287295   four -0.760551
2022-01-05  0.689037 -0.448571 -0.712814  0.695055   five  0.055677
2022-01-06 -1.302253 -1.658869 -1.811578  0.671019    one -1.025420
2022-01-07  0.838272  0.111321 -0.809320 -0.475192    two -0.083730
2022-01-08 -0.377271  0.510839  0.041828 -0.060369  three  0.028757
2022-01-09  0.396088  1.338157 -0.759130  1.828155   four  0.700818
2022-01-10 -0.557112 -0.273150  2.068674  0.741978   five  0.495097
2022-01-11  0.550546 -0.565107 -0.085478 -0.423174    one -0.130803
```

```
2022-01-12 -0.083672 -1.471417 -0.039479  0.111297    two -0.370818
2022-01-13  0.074566 -0.125679  1.443004  0.034680  three  0.356643
2022-01-14  1.265377  0.554316  1.188476 -2.156915   four  0.212814
2022-01-15  1.275653 -0.101605  0.780956  0.675729   five  0.657683
2022-01-16 -1.602749  0.251363  0.271529  2.246843    one  0.291747
2022-01-17 -1.566494  0.545225 -0.388877 -0.317478    two -0.431906
2022-01-18  1.251463  1.940221 -1.155982 -1.181810  three  0.213473
2022-01-19  0.062309 -1.656761  0.531129 -1.674739   four -0.684515
2022-01-20  1.019831 -0.298755 -0.044523  0.686477   five  0.340758
```

Appending one Data frame into another DF1 into DF2

[ ]:
```
df10 = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df11 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])

df10 = df10.append(df11)
print(df10)
```

```
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
```

[ ]:
```
df10 = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df11 = pd.DataFrame([[5, 6], [7, 8]], columns = ['c','d'])

df10 = df10.append(df11)
print(df10)
```

```
     a    b    c    d
0  1.0  2.0  NaN  NaN
1  3.0  4.0  NaN  NaN
0  NaN  NaN  5.0  6.0
1  NaN  NaN  7.0  8.0
```

[ ]:
```
df10 = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df11 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','c'])

df10 = df10.append(df11)
print(df10)
```

```
   a    b    c
0  1  2.0  NaN
1  3  4.0  NaN
0  5  NaN  6.0
1  7  NaN  8.0
```

8. Other Functions (Delete/Pop/Drop)

Deleting a column using del and POP command

[ ]:
```
# Using the previous DataFrame, we will delete a column
# using del function
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
   'three' : pd.Series([10,20,30], index=['a','b','c'])}

df9 = pd.DataFrame(d)
print ("Our dataframe is:")
print(df9)

# using del function
print ("Deleting the first column using DEL function:")
del df9['one']
print(df9)

# using pop function
print ("Deleting another column using POP function:")
df9.pop('two')
print(df9)
```

```
Our dataframe is:
   one  two  three
a  1.0    1   10.0
b  2.0    2   20.0
c  3.0    3   30.0
d  NaN    4    NaN
Deleting the first column using DEL function:
   two  three
a    1   10.0
b    2   20.0
c    3   30.0
d    4    NaN
Deleting another column using POP function:
   three
a   10.0
b   20.0
c   30.0
d    NaN
```

Drop or delete a specific row

[ ]:
```
df10 = df10.drop(0)
print(df10)
```

```
   a    b    c
1  3  4.0  NaN
1  7  NaN  8.0
```