pandas_handson

January 15, 2022

Title= “Mr”

Name= “Syed Saad ul Hassan”

email = “saadulhassanis@gmail.com”

whatsapp = “+491729024676”

Task: Pandas Hands on Practise

0.1 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
```

0.2 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	-0.192360	-0.302980	0.114508	0.021737
2022-01-02	-0.732154	0.675539	-0.089061	-0.714485
2022-01-03	0.920697	0.124596	-0.344493	-0.800064
2022-01-04	0.183988	1.037623	0.077137	1.320106
2022-01-05	-0.296815	2.586747	0.545023	-0.228992
2022-01-06	1.472178	-0.823835	-2.118638	-0.090927
2022-01-07	0.864926	0.214271	1.871359	0.077449
2022-01-08	1.833678	0.442293	2.072272	2.129630
2022-01-09	0.690248	0.698545	1.151583	0.413596
2022-01-10	-2.494094	0.101315	-0.462702	-0.607321
2022-01-11	-0.689498	-0.265338	0.152879	0.080922
2022-01-12	0.877331	1.090692	-0.310215	-0.471069
2022-01-13	-0.013573	0.982974	-1.068835	2.155889
2022-01-14	0.573002	-0.993907	1.870721	0.532708
2022-01-15	-0.048105	0.918509	0.757474	-0.614617
2022-01-16	-0.791288	0.089095	-1.111961	0.901136
2022-01-17	1.140474	1.291287	-0.120033	-1.059595
2022-01-18	-1.199316	0.303289	0.676231	0.280090
2022-01-19	0.571447	-0.800374	-0.800749	-0.594903
2022-01-20	0.845535	1.075723	-1.578412	-0.592417

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([[-0.19236037, -0.30298046,  0.11450808,  0.02173655],
       [-0.73215396,  0.67553874, -0.08906137, -0.71448545],
       [ 0.92069666,  0.12459613, -0.34449267, -0.80006434],
       [ 0.18398778,  1.03762291,  0.07713743,  1.32010566],
       [-0.29681544,  2.58674725,  0.54502258, -0.22899186],
       [ 1.47217771, -0.82383473, -2.1186384 , -0.09092708],
       [ 0.86492625,  0.21427089,  1.87135877,  0.07744906],
       [ 1.83367845,  0.44229333,  2.07227195,  2.12962989],
       [ 0.69024797,  0.69854488,  1.15158338,  0.41359637],
       [-2.49409397,  0.10131488, -0.46270209, -0.60732072],
       [-0.68949819, -0.26533776,  0.15287932,  0.08092192],
       [ 0.87733115,  1.09069157, -0.31021478, -0.47106938],
       [-0.01357257,  0.98297428, -1.06883539,  2.15588947],
       [ 0.573002  , -0.9939066 ,  1.8707213 ,  0.53270836],
       [-0.0481049 ,  0.91850916,  0.75747394, -0.61461735],
       [-0.79128826,  0.08909467, -1.11196068,  0.90113589],
       [ 1.14047406,  1.29128666, -0.12003268, -1.05959519],
       [-1.19931628,  0.30328896,  0.67623098,  0.28009001],
       [ 0.57144725, -0.80037379, -0.80074913, -0.59490279],
       [ 0.84553537,  1.07572315, -1.57841159, -0.59241705]])
```

[ ]:
```
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	0.845535	1.075723	-1.578412	-0.592417
2022-01-19	0.571447	-0.800374	-0.800749	-0.594903
2022-01-18	-1.199316	0.303289	0.676231	0.280090
2022-01-17	1.140474	1.291287	-0.120033	-1.059595
2022-01-16	-0.791288	0.089095	-1.111961	0.901136
2022-01-15	-0.048105	0.918509	0.757474	-0.614617
2022-01-14	0.573002	-0.993907	1.870721	0.532708
2022-01-13	-0.013573	0.982974	-1.068835	2.155889
2022-01-12	0.877331	1.090692	-0.310215	-0.471069
2022-01-11	-0.689498	-0.265338	0.152879	0.080922
2022-01-10	-2.494094	0.101315	-0.462702	-0.607321
2022-01-09	0.690248	0.698545	1.151583	0.413596
2022-01-08	1.833678	0.442293	2.072272	2.129630
2022-01-07	0.864926	0.214271	1.871359	0.077449
2022-01-06	1.472178	-0.823835	-2.118638	-0.090927
2022-01-05	-0.296815	2.586747	0.545023	-0.228992
2022-01-04	0.183988	1.037623	0.077137	1.320106
2022-01-03	0.920697	0.124596	-0.344493	-0.800064
2022-01-02	-0.732154	0.675539	-0.089061	-0.714485
2022-01-01	-0.192360	-0.302980	0.114508	0.021737

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

[ ]:

	A	B	C	D
2022-01-01	-0.192360	-0.302980	0.114508	0.021737
2022-01-02	-0.732154	0.675539	-0.089061	-0.714485
2022-01-03	0.920697	0.124596	-0.344493	-0.800064
2022-01-04	0.183988	1.037623	0.077137	1.320106
2022-01-05	-0.296815	2.586747	0.545023	-0.228992
2022-01-06	1.472178	-0.823835	-2.118638	-0.090927
2022-01-07	0.864926	0.214271	1.871359	0.077449
2022-01-08	1.833678	0.442293	2.072272	2.129630
2022-01-09	0.690248	0.698545	1.151583	0.413596
2022-01-10	-2.494094	0.101315	-0.462702	-0.607321
2022-01-11	-0.689498	-0.265338	0.152879	0.080922

```
2022-01-12  0.877331  1.090692 -0.310215 -0.471069
2022-01-13 -0.013573  0.982974 -1.068835  2.155889
2022-01-14  0.573002 -0.993907  1.870721  0.532708
2022-01-15 -0.048105  0.918509  0.757474 -0.614617
2022-01-16 -0.791288  0.089095 -1.111961  0.901136
2022-01-17  1.140474  1.291287 -0.120033 -1.059595
2022-01-18 -1.199316  0.303289  0.676231  0.280090
2022-01-19  0.571447 -0.800374 -0.800749 -0.594903
2022-01-20  0.845535  1.075723 -1.578412 -0.592417
```

Sorting Ascending/Descending (Only Column Heads) not Values

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

[ ]:
```
                   D         C         B         A
2022-01-01  0.021737  0.114508 -0.302980 -0.192360
2022-01-02 -0.714485 -0.089061  0.675539 -0.732154
2022-01-03 -0.800064 -0.344493  0.124596  0.920697
2022-01-04  1.320106  0.077137  1.037623  0.183988
2022-01-05 -0.228992  0.545023  2.586747 -0.296815
2022-01-06 -0.090927 -2.118638 -0.823835  1.472178
2022-01-07  0.077449  1.871359  0.214271  0.864926
2022-01-08  2.129630  2.072272  0.442293  1.833678
2022-01-09  0.413596  1.151583  0.698545  0.690248
2022-01-10 -0.607321 -0.462702  0.101315 -2.494094
2022-01-11  0.080922  0.152879 -0.265338 -0.689498
2022-01-12 -0.471069 -0.310215  1.090692  0.877331
2022-01-13  2.155889 -1.068835  0.982974 -0.013573
2022-01-14  0.532708  1.870721 -0.993907  0.573002
2022-01-15 -0.614617  0.757474  0.918509 -0.048105
2022-01-16  0.901136 -1.111961  0.089095 -0.791288
2022-01-17 -1.059595 -0.120033  1.291287  1.140474
2022-01-18  0.280090  0.676231  0.303289 -1.199316
2022-01-19 -0.594903 -0.800749 -0.800374  0.571447
2022-01-20 -0.592417 -1.578412  1.075723  0.845535
```

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

[ ]:
```
                   A         B         C         D
2022-01-01 -0.192360 -0.302980  0.114508  0.021737
2022-01-02 -0.732154  0.675539 -0.089061 -0.714485
2022-01-03  0.920697  0.124596 -0.344493 -0.800064
2022-01-04  0.183988  1.037623  0.077137  1.320106
2022-01-05 -0.296815  2.586747  0.545023 -0.228992
2022-01-06  1.472178 -0.823835 -2.118638 -0.090927
2022-01-07  0.864926  0.214271  1.871359  0.077449
2022-01-08  1.833678  0.442293  2.072272  2.129630
2022-01-09  0.690248  0.698545  1.151583  0.413596
```

```
2022-01-10 -2.494094  0.101315 -0.462702 -0.607321
2022-01-11 -0.689498 -0.265338  0.152879  0.080922
2022-01-12  0.877331  1.090692 -0.310215 -0.471069
2022-01-13 -0.013573  0.982974 -1.068835  2.155889
2022-01-14  0.573002 -0.993907  1.870721  0.532708
2022-01-15 -0.048105  0.918509  0.757474 -0.614617
2022-01-16 -0.791288  0.089095 -1.111961  0.901136
2022-01-17  1.140474  1.291287 -0.120033 -1.059595
2022-01-18 -1.199316  0.303289  0.676231  0.280090
2022-01-19  0.571447 -0.800374 -0.800749 -0.594903
2022-01-20  0.845535  1.075723 -1.578412 -0.592417
```

Sorting a specified Column of Data Frame sorting its values

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

[ ]:
```
                   A         B         C         D
2022-01-14  0.573002 -0.993907  1.870721  0.532708
2022-01-06  1.472178 -0.823835 -2.118638 -0.090927
2022-01-19  0.571447 -0.800374 -0.800749 -0.594903
2022-01-01 -0.192360 -0.302980  0.114508  0.021737
2022-01-11 -0.689498 -0.265338  0.152879  0.080922
2022-01-16 -0.791288  0.089095 -1.111961  0.901136
2022-01-10 -2.494094  0.101315 -0.462702 -0.607321
2022-01-03  0.920697  0.124596 -0.344493 -0.800064
2022-01-07  0.864926  0.214271  1.871359  0.077449
2022-01-18 -1.199316  0.303289  0.676231  0.280090
2022-01-08  1.833678  0.442293  2.072272  2.129630
2022-01-02 -0.732154  0.675539 -0.089061 -0.714485
2022-01-09  0.690248  0.698545  1.151583  0.413596
2022-01-15 -0.048105  0.918509  0.757474 -0.614617
2022-01-13 -0.013573  0.982974 -1.068835  2.155889
2022-01-04  0.183988  1.037623  0.077137  1.320106
2022-01-20  0.845535  1.075723 -1.578412 -0.592417
2022-01-12  0.877331  1.090692 -0.310215 -0.471069
2022-01-17  1.140474  1.291287 -0.120033 -1.059595
2022-01-05 -0.296815  2.586747  0.545023 -0.228992
```

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

[ ]:
```
                   A         B         C         D
2022-01-14  0.573002 -0.993907  1.870721  0.532708
2022-01-06  1.472178 -0.823835 -2.118638 -0.090927
2022-01-19  0.571447 -0.800374 -0.800749 -0.594903
2022-01-01 -0.192360 -0.302980  0.114508  0.021737
2022-01-11 -0.689498 -0.265338  0.152879  0.080922
2022-01-16 -0.791288  0.089095 -1.111961  0.901136
2022-01-10 -2.494094  0.101315 -0.462702 -0.607321
```

```
2022-01-03  0.920697  0.124596 -0.344493 -0.800064
2022-01-07  0.864926  0.214271  1.871359  0.077449
2022-01-18 -1.199316  0.303289  0.676231  0.280090
2022-01-08  1.833678  0.442293  2.072272  2.129630
2022-01-02 -0.732154  0.675539 -0.089061 -0.714485
2022-01-09  0.690248  0.698545  1.151583  0.413596
2022-01-15 -0.048105  0.918509  0.757474 -0.614617
2022-01-13 -0.013573  0.982974 -1.068835  2.155889
2022-01-04  0.183988  1.037623  0.077137  1.320106
2022-01-20  0.845535  1.075723 -1.578412 -0.592417
2022-01-12  0.877331  1.090692 -0.310215 -0.471069
2022-01-17  1.140474  1.291287 -0.120033 -1.059595
2022-01-05 -0.296815  2.586747  0.545023 -0.228992
```

0.3 4. Displaying Data in a Data frames

To Display an entire column

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

[ ]:
```
2022-01-01   -0.192360
2022-01-02   -0.732154
2022-01-03    0.920697
2022-01-04    0.183988
2022-01-05   -0.296815
2022-01-06    1.472178
2022-01-07    0.864926
2022-01-08    1.833678
2022-01-09    0.690248
2022-01-10   -2.494094
2022-01-11   -0.689498
2022-01-12    0.877331
2022-01-13   -0.013573
2022-01-14    0.573002
2022-01-15   -0.048105
2022-01-16   -0.791288
2022-01-17    1.140474
2022-01-18   -1.199316
2022-01-19    0.571447
2022-01-20    0.845535
Freq: D, Name: A, dtype: float64
```

To display selected rows

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

[ ]:
```
                   A         B         C         D
2022-01-01 -0.192360 -0.302980  0.114508  0.021737
```

```
2022-01-02 -0.732154  0.675539 -0.089061 -0.714485
```

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

[ ]:
```
                   A         B         C         D
2022-01-03  0.920697  0.124596 -0.344493 -0.800064
2022-01-04  0.183988  1.037623  0.077137  1.320106
2022-01-05 -0.296815  2.586747  0.545023 -0.228992
2022-01-06  1.472178 -0.823835 -2.118638 -0.090927
2022-01-07  0.864926  0.214271  1.871359  0.077449
2022-01-08  1.833678  0.442293  2.072272  2.129630
2022-01-09  0.690248  0.698545  1.151583  0.413596
2022-01-10 -2.494094  0.101315 -0.462702 -0.607321
```

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

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

[ ]:
```
-2.118638403342511
```

0.4 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    1.472178
B   -0.823835
C   -2.118638
D   -0.090927
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  0.183988  0.077137
2022-01-05 -0.296815  0.545023
2022-01-06  1.472178 -2.118638
```

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

```
                   A         C
2022-01-05 -0.296815  0.545023
2022-01-07  0.864926  1.871359
```

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  1.037623  0.077137  1.320106
2022-01-05  2.586747  0.545023 -0.228992
2022-01-06 -0.823835 -2.118638 -0.090927
2022-01-07  0.214271  1.871359  0.077449
2022-01-08  0.442293  2.072272  2.129630
2022-01-09  0.698545  1.151583  0.413596
2022-01-10  0.101315 -0.462702 -0.607321
```

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

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

```
-2.118638403342511
```

0.5 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     True
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
```

0.5.1 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
2022-01-03  0.920697  0.124596 -0.344493 -0.800064
2022-01-04  0.183988  1.037623  0.077137  1.320106
2022-01-06  1.472178 -0.823835 -2.118638 -0.090927
2022-01-07  0.864926  0.214271  1.871359  0.077449
2022-01-08  1.833678  0.442293  2.072272  2.129630
2022-01-09  0.690248  0.698545  1.151583  0.413596
2022-01-12  0.877331  1.090692 -0.310215 -0.471069
2022-01-14  0.573002 -0.993907  1.870721  0.532708
2022-01-17  1.140474  1.291287 -0.120033 -1.059595
2022-01-19  0.571447 -0.800374 -0.800749 -0.594903
2022-01-20  0.845535  1.075723 -1.578412 -0.592417
```

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
2022-01-04  0.077137  1.320106
2022-01-08  2.072272  2.129630
2022-01-09  1.151583  0.413596
2022-01-12 -0.310215 -0.471069
2022-01-17 -0.120033 -1.059595
2022-01-20 -1.578412 -0.592417
```

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

The sorted value that satisifies condition in A is

[ ]:
```
2022-01-03    0.920697
2022-01-04    0.183988
2022-01-07    0.864926
2022-01-08    1.833678
2022-01-09    0.690248
2022-01-12    0.877331
2022-01-17    1.140474
2022-01-20    0.845535
Name: A, dtype: float64
```

[ ]:
```
e
```

[ ]:
```
                   A         B         C         D
2022-01-03  0.920697  0.124596 -0.344493 -0.800064
2022-01-04  0.183988  1.037623  0.077137  1.320106
2022-01-07  0.864926  0.214271  1.871359  0.077449
2022-01-08  1.833678  0.442293  2.072272  2.129630
2022-01-09  0.690248  0.698545  1.151583  0.413596
2022-01-12  0.877331  1.090692 -0.310215 -0.471069
```

```
2022-01-17  1.140474  1.291287 -0.120033 -1.059595
2022-01-20  0.845535  1.075723 -1.578412 -0.592417
```

To find values greater or less than a specific number

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

[ ]:
```
                   A         B         C         D
2022-01-01       NaN       NaN  0.114508  0.021737
2022-01-02       NaN  0.675539       NaN       NaN
2022-01-03  0.920697  0.124596       NaN       NaN
2022-01-04  0.183988  1.037623  0.077137  1.320106
2022-01-05       NaN  2.586747  0.545023       NaN
2022-01-06  1.472178       NaN       NaN       NaN
2022-01-07  0.864926  0.214271  1.871359  0.077449
2022-01-08  1.833678  0.442293  2.072272  2.129630
2022-01-09  0.690248  0.698545  1.151583  0.413596
2022-01-10       NaN  0.101315       NaN       NaN
2022-01-11       NaN       NaN  0.152879  0.080922
2022-01-12  0.877331  1.090692       NaN       NaN
2022-01-13       NaN  0.982974       NaN  2.155889
2022-01-14  0.573002       NaN  1.870721  0.532708
2022-01-15       NaN  0.918509  0.757474       NaN
2022-01-16       NaN  0.089095       NaN  0.901136
2022-01-17  1.140474  1.291287       NaN       NaN
2022-01-18       NaN  0.303289  0.676231  0.280090
2022-01-19  0.571447       NaN       NaN       NaN
2022-01-20  0.845535  1.075723       NaN       NaN
```

0.6 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 -0.192360 -0.302980  0.114508  0.021737    one
2022-01-02 -0.732154  0.675539 -0.089061 -0.714485    two
2022-01-03  0.920697  0.124596 -0.344493 -0.800064  three
2022-01-04  0.183988  1.037623  0.077137  1.320106   four
2022-01-05 -0.296815  2.586747  0.545023 -0.228992   five
2022-01-06  1.472178 -0.823835 -2.118638 -0.090927    one
2022-01-07  0.864926  0.214271  1.871359  0.077449    two
2022-01-08  1.833678  0.442293  2.072272  2.129630  three
2022-01-09  0.690248  0.698545  1.151583  0.413596   four
```

```
2022-01-10 -2.494094  0.101315 -0.462702 -0.607321   five
2022-01-11 -0.689498 -0.265338  0.152879  0.080922    one
2022-01-12  0.877331  1.090692 -0.310215 -0.471069    two
2022-01-13 -0.013573  0.982974 -1.068835  2.155889  three
2022-01-14  0.573002 -0.993907  1.870721  0.532708   four
2022-01-15 -0.048105  0.918509  0.757474 -0.614617   five
2022-01-16 -0.791288  0.089095 -1.111961  0.901136    one
2022-01-17  1.140474  1.291287 -0.120033 -1.059595    two
2022-01-18 -1.199316  0.303289  0.676231  0.280090  three
2022-01-19  0.571447 -0.800374 -0.800749 -0.594903   four
2022-01-20  0.845535  1.075723 -1.578412 -0.592417   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 -0.192360 -0.302980  0.114508  0.021737
2022-01-02 -0.732154  0.675539 -0.089061 -0.714485
2022-01-03  0.920697  0.124596 -0.344493 -0.800064
2022-01-04  0.183988  1.037623  0.077137  1.320106
2022-01-05 -0.296815  2.586747  0.545023 -0.228992
2022-01-06  1.472178 -0.823835 -2.118638 -0.090927
2022-01-07  0.864926  0.214271  1.871359  0.077449
2022-01-08  1.833678  0.442293  2.072272  2.129630
2022-01-09  0.690248  0.698545  1.151583  0.413596
2022-01-10 -2.494094  0.101315 -0.462702 -0.607321
2022-01-11 -0.689498 -0.265338  0.152879  0.080922
2022-01-12  0.877331  1.090692 -0.310215 -0.471069
2022-01-13 -0.013573  0.982974 -1.068835  2.155889
2022-01-14  0.573002 -0.993907  1.870721  0.532708
2022-01-15 -0.048105  0.918509  0.757474 -0.614617
2022-01-16 -0.791288  0.089095 -1.111961  0.901136
2022-01-17  1.140474  1.291287 -0.120033 -1.059595
2022-01-18 -1.199316  0.303289  0.676231  0.280090
2022-01-19  0.571447 -0.800374 -0.800749 -0.594903
2022-01-20  0.845535  1.075723 -1.578412 -0.592417
```

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 -0.192360 -0.302980  0.114508  0.021737    one -0.089774
2022-01-02 -0.732154  0.675539 -0.089061 -0.714485    two -0.215041
```

```
2022-01-03  0.920697  0.124596 -0.344493 -0.800064  three -0.024816
2022-01-04  0.183988  1.037623  0.077137  1.320106   four  0.654713
2022-01-05 -0.296815  2.586747  0.545023 -0.228992   five  0.651491
2022-01-06  1.472178 -0.823835 -2.118638 -0.090927    one -0.390306
2022-01-07  0.864926  0.214271  1.871359  0.077449    two  0.757001
2022-01-08  1.833678  0.442293  2.072272  2.129630  three  1.619468
2022-01-09  0.690248  0.698545  1.151583  0.413596   four  0.738493
2022-01-10 -2.494094  0.101315 -0.462702 -0.607321   five -0.865700
2022-01-11 -0.689498 -0.265338  0.152879  0.080922    one -0.180259
2022-01-12  0.877331  1.090692 -0.310215 -0.471069    two  0.296685
2022-01-13 -0.013573  0.982974 -1.068835  2.155889  three  0.514114
2022-01-14  0.573002 -0.993907  1.870721  0.532708   four  0.495631
2022-01-15 -0.048105  0.918509  0.757474 -0.614617   five  0.253315
2022-01-16 -0.791288  0.089095 -1.111961  0.901136    one -0.228255
2022-01-17  1.140474  1.291287 -0.120033 -1.059595    two  0.313033
2022-01-18 -1.199316  0.303289  0.676231  0.280090  three  0.015073
2022-01-19  0.571447 -0.800374 -0.800749 -0.594903   four -0.406145
2022-01-20  0.845535  1.075723 -1.578412 -0.592417   five -0.062393
```

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
```

0.7 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
```

[ ]:
```

```