Practice7

September 11, 2025

[2]:
```
import pandas as pd
import numpy as np
ser=pd.Series(('ankit','kiio',np.nan,'summi'),index=['a','b','c','d'])
print(ser)
```

```
a    ankit
b     kiio
c      NaN
d    summi
dtype: object
```

[3]:
```
print(ser.isnull())
print(ser.isna())
print(ser.isnull().sum())
```

```
a    False
b    False
c     True
d    False
dtype: bool
a    False
b    False
c     True
d    False
dtype: bool
1
```

[4]:
```
print(ser.dropna(axis=0)) # any row with np.nan
print(ser[ser.notnull()])
```

```
a    ankit
b     kiio
d    summi
dtype: object
a    ankit
b     kiio
d    summi
dtype: object
```

[5]:
```
df=pd.DataFrame([['ankit',np.nan,'ISI'],['soumi',100,np.
↪nan],['summi',100,'ECIL'],[np.nan,np.nan,np.nan],\
                 ␣
↪['kiio',100,'IIIT-D']],columns=['Name','Marks','Institute'],index=['a','b','c','d','e'])
print(df)
```

```
    Name  Marks Institute
a  ankit    NaN       ISI
b  soumi  100.0       NaN
c  summi  100.0      ECIL
d    NaN    NaN       NaN
e   kiio  100.0    IIIT-D
```

[6]:
```
print(df.dropna(how='all',axis=0)) # drop all rows with null
```

```
    Name  Marks Institute
a  ankit    NaN       ISI
b  soumi  100.0       NaN
c  summi  100.0      ECIL
e   kiio  100.0    IIIT-D
```

[7]:
```
print(df.dropna(thresh=3)) # rows with atleast 3 without np.nan values
```

```
    Name  Marks Institute
c  summi  100.0      ECIL
e   kiio  100.0    IIIT-D
```

[8]:
```
data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],'k2': [1, 1, 2, 3, 3,␣
↪4, 4]})
print(data)
```

```
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
4  one   3
5  two   4
6  two   4
```

[9]:
```
print(data.duplicated())
```

```
0    False
1    False
2    False
3    False
4    False
5    False
```

```
6     True
dtype: bool
```

[10]:
```
print(data.drop_duplicates())
```

```
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
4  one   3
5  two   4
```

[11]:
```
print(data.drop_duplicates(['k1']))
```

```
    k1  k2
0  one   1
1  two   1
```

[12]:
```
print(data.drop_duplicates(['k1', 'k2'], keep='last'))
```

```
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
4  one   3
6  two   4
```

[13]:
```
data=pd.DataFrame(['ankit','kiio','summi'],columns=['Name'],index=['a','b','c'])
print(data)
```

```
    Name
a  ankit
b   kiio
c  summi
```

[14]:
```
data['Marks']=data['Name'].map({k:(0 if k=='ankit' else 100) for k in␣
↪data['Name'] })
print(data)
```

```
    Name  Marks
a  ankit      0
b   kiio    100
c  summi    100
```

[15]:
```
data['Marks']=data['Name'].map(lambda x:0 if x=='ankit' else 100)
print(data)
```

```
    Name  Marks
a  ankit      0
b   kiio    100
c  summi    100
```

[16]:
```
data=pd.Series([1,999,34,21])
data=data.replace([999,34],np.nan)
print(data)
```

```
0     1.0
1     NaN
2     NaN
3    21.0
dtype: float64
```

[17]:
```
data=pd.DataFrame(['ankit','kiio','summi'],columns=['Name'],index=['a','b','c'])
print(data)
data['Name'].replace('summi','soumi',inplace=True)
print(data)
```

```
    Name
a  ankit
b   kiio
c  summi
    Name
a  ankit
b   kiio
c  soumi
```

[18]:
```
data=pd.
  ↪DataFrame(['ankit','kiio','summi'],columns=['Name'],index=['abc','bcd','cde'])
data.rename(index=str.capitalize,columns=str.upper)
```

[18]:
```
       NAME
Abc  ankit
Bcd   kiio
Cde  summi
```

[19]:
```
data.rename(index={'abc':'new','cde':'new_index'},columns={'Name':
  ↪'New_Name'},inplace=True)
print(data)
```

```
          New_Name
new          ankit
bcd           kiio
new_index    summi
```

Making bins of data

[20]:
```
b=pd.
↪cut([12,2,1,22,11,34,32,1,41,48,45,51,58,61,66,67,69,70,71,72,90,91,81,81,11,2,3,2,1,2,0],\
        [0,10,20,30,40,50,60,70,80,90,100],right=True,
       ␣
↪labels=['0-10','11-20','21-30','31-40','41-50','51-60','61-70','71-80','81-90','91-100'])
```

[21]:
```
b
```

[21]:
```
['11-20', '0-10', '0-10', '21-30', '11-20', …, '0-10', '0-10', '0-10', '0-10',
NaN]
Length: 31
Categories (10, object): ['0-10' < '11-20' < '21-30' < '31-40' … '61-70' <
'71-80' < '81-90' < '91-100']
```

[22]:
```
b.codes
```

[22]:
```
array([ 1,  0,  0,  2,  1,  3,  3,  0,  4,  4,  4,  5,  5,  6,  6,  6,  6,
        6,  7,  7,  8,  9,  8,  8,  1,  0,  0,  0,  0,  0, -1], dtype=int8)
```

[23]:
```
b.categories
```

[23]:
```
Index(['0-10', '11-20', '21-30', '31-40', '41-50', '51-60', '61-70', '71-80',
       '81-90', '91-100'],
      dtype='object')
```

[24]:
```
pd.value_counts(b)
```

[24]:
```
0-10      8
61-70     5
11-20     3
41-50     3
81-90     3
31-40     2
51-60     2
71-80     2
21-30     1
91-100    1
dtype: int64
```

A closely related function, qcut, bins the data based on sample quantiles. Depending on the distribution of the data, using cut will not usually result in each bin having the same number of data points. Since qcut uses sample quantiles instead, by definition you will obtain roughly equal-size bins

[25]:
```
data = np.random.randn(1000) # Normally distributed
cats = pd.qcut(data, 4) # Cut into quartiles
print(cats)
```

```
print(pd.value_counts(cats))
```

```
[(-0.685, 0.00999], (-0.685, 0.00999], (0.629, 3.114], (-0.685, 0.00999],
(0.629, 3.114], …, (0.629, 3.114], (-2.9099999999999997, -0.685],
(-2.9099999999999997, -0.685], (-2.9099999999999997, -0.685],
(-2.9099999999999997, -0.685]]
Length: 1000
Categories (4, interval[float64, right]): [(-2.9099999999999997, -0.685] <
(-0.685, 0.00999] < (0.00999, 0.629] < (0.629, 3.114]]
(-2.9099999999999997, -0.685]    250
(-0.685, 0.00999]                250
(0.00999, 0.629]                 250
(0.629, 3.114]                   250
dtype: int64
```

[26]:
```
cats.categories
```

[26]:
```
IntervalIndex([(-2.9099999999999997, -0.685], (-0.685, 0.00999], (0.00999,
0.629], (0.629, 3.114]], dtype='interval[float64, right]')
```

[27]:
```
# Similar to cut you can pass your own quantiles (numbers between 0 and 1,␣
 ↪inclusive):

pd.qcut(data, [0, 0.1, 0.5, 0.9, 1.])
```

[27]:
```
[(-1.256, 0.00999], (-1.256, 0.00999], (0.00999, 1.223], (-1.256, 0.00999],
(1.223, 3.114], …, (1.223, 3.114], (-1.256, 0.00999], (-2.9099999999999997,
-1.256], (-1.256, 0.00999], (-2.9099999999999997, -1.256]]
Length: 1000
Categories (4, interval[float64, right]): [(-2.9099999999999997, -1.256] <
(-1.256, 0.00999] < (0.00999, 1.223] < (1.223, 3.114]]
```

Detecting and Filtering Outliers

Suppose you wanted to find values in one of the columns exceeding 3 in absolute value

[28]:
```
data = pd.DataFrame(np.random.randn(1000, 4))
data
```

[28]:

	0	1	2	3
0	0.891199	-1.210597	0.564329	1.730597
1	-0.078886	-0.893330	-1.070863	2.936057
2	-0.811553	1.349532	0.288253	-0.510670
3	-0.037945	-0.018861	-0.119692	0.427349
4	-0.301513	-3.514392	0.616071	-0.778039
..	…	…	…	…
995	0.377140	-2.146521	0.470708	0.911578
996	-0.713249	-0.387552	0.220569	1.796074

```
997  2.186931 -2.333990 -0.336696  0.081126
998  0.141597 -0.426379 -1.120187 -0.511060
999  1.571511 -0.160032 -0.123935 -1.426114

[1000 rows x 4 columns]
```

[29]:
```
data[(np.abs(data) > 0).any(1)]
```

```
/tmp/ipykernel_4175/1506215609.py:1: FutureWarning: In a future version of
pandas all arguments of DataFrame.any and Series.any will be keyword-only.
  data[(np.abs(data) > 0).any(1)]
```

[29]:
```
            0         1         2         3
0    0.891199 -1.210597  0.564329  1.730597
1   -0.078886 -0.893330 -1.070863  2.936057
2   -0.811553  1.349532  0.288253 -0.510670
3   -0.037945 -0.018861 -0.119692  0.427349
4   -0.301513 -3.514392  0.616071 -0.778039
..        …         …         …         …
995  0.377140 -2.146521  0.470708  0.911578
996 -0.713249 -0.387552  0.220569  1.796074
997  2.186931 -2.333990 -0.336696  0.081126
998  0.141597 -0.426379 -1.120187 -0.511060
999  1.571511 -0.160032 -0.123935 -1.426114

[1000 rows x 4 columns]
```

[ ]:
```
np.sign(data) * 3
# The statement np.sign(data) produces 1 and -1 values based on whether the␣
 ↪values in data are positive or negative
```

[ ]:
```
       0    1    2    3
0    3.0 -3.0  3.0  3.0
1   -3.0 -3.0 -3.0  3.0
2   -3.0  3.0  3.0 -3.0
3   -3.0 -3.0 -3.0  3.0
4   -3.0 -3.0  3.0 -3.0
..   …   …   …   …
995  3.0 -3.0  3.0  3.0
996 -3.0 -3.0  3.0  3.0
997  3.0 -3.0 -3.0  3.0
998  3.0 -3.0 -3.0 -3.0
999  3.0 -3.0 -3.0 -3.0

[1000 rows x 4 columns]
```

[31]:
```
data[np.abs(data) > 3] = np.sign(data) * 3
```

[32]:
```
data
```

[32]:
```
          0         1         2         3
0    0.891199 -1.210597  0.564329  1.730597
1   -0.078886 -0.893330 -1.070863  2.936057
2   -0.811553  1.349532  0.288253 -0.510670
3   -0.037945 -0.018861 -0.119692  0.427349
4   -0.301513 -3.000000  0.616071 -0.778039
..        …         …         …         …
995  0.377140 -2.146521  0.470708  0.911578
996 -0.713249 -0.387552  0.220569  1.796074
997  2.186931 -2.333990 -0.336696  0.081126
998  0.141597 -0.426379 -1.120187 -0.511060
999  1.571511 -0.160032 -0.123935 -1.426114

[1000 rows x 4 columns]
```

[33]:
```
df = pd.DataFrame(np.arange(5 * 4).reshape((5, 4)),columns=['a','b','c','d'])
print(df)
```

```
    a   b   c   d
0   0   1   2   3
1   4   5   6   7
2   8   9  10  11
3  12  13  14  15
4  16  17  18  19
```

[34]:
```
sampler=np.random.permutation(5)
sampler
```

[34]:
```
array([2, 4, 0, 3, 1])
```

[35]:
```
print(df.take(sampler,axis=0))
```

```
    a   b   c   d
2   8   9  10  11
4  16  17  18  19
0   0   1   2   3
3  12  13  14  15
1   4   5   6   7
```

[36]:
```
# To select a random subset without replacement, you can use the sample method␣
 ↪on Series and DataFrame:
print(df.sample(n=3))
```

```
    a   b   c   d
1   4   5   6   7
```

```
2   8   9  10  11
4  16  17  18  19
```

[37]:
```
print(df.sample(n=3))
```

```
    a   b   c   d
0   0   1   2   3
2   8   9  10  11
4  16  17  18  19
```

[38]:
```
# To generate a sample with replacement (to allow repeat choices), pass␣
↪replace=True to sample:

choices = pd.Series([5, 7, -1, 6, 4])
draws = choices.sample(n=10, replace=True)
print(draws)
```

```
3    6
2   -1
4    4
1    7
3    6
1    7
2   -1
4    4
3    6
4    4
dtype: int64
```

categorical to dummy variables

[39]:
```
df=pd.DataFrame({'Name':['ankit','kiio','summi'],'Marks':range(3)})
print(df)
```

```
    Name  Marks
0  ankit      0
1   kiio      1
2  summi      2
```

[40]:
```
print(pd.get_dummies(df['Name'],prefix='name').join(df[['Marks']]))
```

```
   name_ankit  name_kiio  name_summi  Marks
0           1          0           0      0
1           0          1           0      1
2           0          0           1      2
```

[41]:
```
b=pd.
↪cut([12,2,1,22,11,34,32,1,41,48,45,51,58,61,66,67,69,70,71,72,90,91,81,81,11,2,3,2,1,2,0],
```

```
        [0,10,20,30,40,50,60,70,80,90,100],right=True,
       ␣
  ↪labels=['0-10','11-20','21-30','31-40','41-50','51-60','61-70','71-80','81-90','91-100'])
print(pd.get_dummies(b))
```

	0-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100
0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0	0
5	0	0	0	1	0	0	0	0	0	0
6	0	0	0	1	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	1	0	0	0	0	0
10	0	0	0	0	1	0	0	0	0	0
11	0	0	0	0	0	1	0	0	0	0
12	0	0	0	0	0	1	0	0	0	0
13	0	0	0	0	0	0	1	0	0	0
14	0	0	0	0	0	0	1	0	0	0
15	0	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	0	1	0	0	0
17	0	0	0	0	0	0	1	0	0	0
18	0	0	0	0	0	0	0	1	0	0
19	0	0	0	0	0	0	0	1	0	0
20	0	0	0	0	0	0	0	0	1	0
21	0	0	0	0	0	0	0	0	0	1
22	0	0	0	0	0	0	0	0	1	0
23	0	0	0	0	0	0	0	0	1	0
24	0	1	0	0	0	0	0	0	0	0
25	1	0	0	0	0	0	0	0	0	0
26	1	0	0	0	0	0	0	0	0	0
27	1	0	0	0	0	0	0	0	0	0
28	1	0	0	0	0	0	0	0	0	0
29	1	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0

[ ]: