Indexes Binning Outliers and Sampling

October 8, 2019

0.1 Environment

```
In [3]: import numpy as np
        import pandas as pd
        PREVIOUS_MAX_ROWS = pd.options.display.max_rows
        pd.options.display.max_rows = 20
        np.random.seed(12345)
        import matplotlib.pyplot as plt
        plt.rc('figure', figsize=(10, 6))
        np.set_printoptions(precision=4, suppress=True)
```

0.1.1 Rename axis indexes

```
In [8]: data=pd.DataFrame(np.arange(12).reshape((3,4)),
                          index=['ant','bee','cat'],
                          columns=['one','two','three','four'])
```

```
In [9]: data
```

```
Out[9]:      one  two  three  four
        ant    0    1      2     3
        bee    4    5      6     7
        cat    8    9     10    11
```

```
In [11]: transform=lambda x: x[:4].upper()  # Define a function that changes indexes into uppe
         data.index.map(transform)
```

```
Out[11]: Index(['ANT', 'BEE', 'CAT'], dtype='object')
```

```
In [13]: data.index=data.index.map(transform)
         data
```

```
Out[13]:      one  two  three  four
         ANT    0    1      2     3
         BEE    4    5      6     7
         CAT    8    9     10    11
```

```
In [14]: data.rename(index=str.title, columns=str.upper)
```

```
Out[14]:      ONE  TWO  THREE  FOUR
         Ant    0    1      2     3
         Bee    4    5      6     7
         Cat    8    9     10    11
```

```
In [18]: data.rename(index={'ANT':'Ant-eater'},
                     columns={'three':'two-n-a-half'})
```

```
Out[18]:            one  two  two-n-a-half  four
         Ant-eater    0    1             2     3
         BEE          4    5             6     7
         CAT          8    9            10    11
```

```
In [19]: data.rename(index={'ANT':'Antelope'}, inplace=True)
         data
```

```
Out[19]:           one  two  three  four
         Antelope    0    1      2     3
         BEE         4    5      6     7
         CAT         8    9     10    11
```

0.1.2 Discretization and Binning

```
In [37]: ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32] # alternatively can use ages=
         ages
```

```
Out[37]: [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

```
In [39]: bins=[18,25,35,60,100]
         cats=pd.cut(ages,bins) #categories with open lower but closed upper limits
         cats
```

```
Out[39]: [(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60]
         Length: 12
         Categories (4, interval[int64]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
```

```
In [40]: cats.codes
```

```
Out[40]: array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)
```

```
In [41]: cats.categories
```

```
Out[41]: IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]],
                       closed='right',
                       dtype='interval[int64]')
```

```
In [42]: pd.value_counts(cats) # frequency coutns
```

```
Out[42]: (18, 25]     5
         (35, 60]     3
         (25, 35]     3
         (60, 100]    1
         dtype: int64
```

```
In [43]: pd.cut(ages,[18,26,36,61,100],right=False) #categories with closed lower but open upp

Out[43]: [[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), ..., [26, 36), [61, 100), [36, 61)
         Length: 12
         Categories (4, interval[int64]): [[18, 26) < [26, 36) < [36, 61) < [61, 100)]

In [44]: group_names=['Youth', 'YoungAdult', 'MiddleAged', 'Senior']
         pd.cut(ages, bins, labels=group_names)

Out[44]: [Youth, Youth, Youth, YoungAdult, Youth, ..., YoungAdult, Senior, MiddleAged, MiddleAg
         Length: 12
         Categories (4, object): [Youth < YoungAdult < MiddleAged < Senior]

In [46]: data=np.random.rand(20) #uniform random 20 numbers
         data

Out[46]: array([0.8374, 0.3832, 0.2988, 0.0063, 0.4376, 0.7379, 0.3758, 0.4932,
                0.014 , 0.2494, 0.3471, 0.7967, 0.9384, 0.1005, 0.7354, 0.9764,
                0.7059, 0.9518, 0.9278, 0.41  ])

In [48]: pd.cut(data, 4, precision=2)

Out[48]: [(0.73, 0.98], (0.25, 0.49], (0.25, 0.49], (0.0053, 0.25], (0.25, 0.49], ..., (0.73,
         Length: 20
         Categories (4, interval[float64]): [(0.0053, 0.25] < (0.25, 0.49] < (0.49, 0.73] < (0

In [52]: data=np.random.randn(1000) #1000 random normal numbers
         cats=pd.qcut(data,4) #Group into quartiles
         cats
         pd.value_counts(cats)

Out[52]: (0.663, 3.389]       250
         (-0.0374, 0.663]     250
         (-0.673, -0.0374]    250
         (-3.665, -0.673]     250
         dtype: int64

In [57]: newcats=pd.qcut(data,[0,0.2,0.5,0.8,1.])
         newcats

Out[57]: [(-0.863, -0.0374], (-0.863, -0.0374], (-3.665, -0.863], (0.83, 3.389], (0.83, 3.389]
         Length: 1000
         Categories (4, interval[float64]): [(-3.665, -0.863] < (-0.863, -0.0374] < (-0.0374,

In [58]: pd.value_counts(newcats)

Out[58]: (-0.0374, 0.83]      300
         (-0.863, -0.0374]    300
         (0.83, 3.389]        200
         (-3.665, -0.863]     200
         dtype: int64
```

0.1.3 Identify and work with outliers

```
In [60]: data=pd.DataFrame(np.random.randn(1000, 4)) #data matrix with thousand rows and 4 col
         data.describe() #summary statistics
```

```
Out[60]:                  0            1            2            3
         count  1000.000000  1000.000000  1000.000000  1000.000000
         mean     -0.060144    -0.009583     0.004146    -0.020177
         std       1.025283     0.991398     1.010832     1.009819
         min      -3.067430    -3.423739    -2.944923    -3.094371
         25%      -0.756485    -0.671014    -0.674305    -0.688851
         50%      -0.056539     0.032223    -0.015343    -0.026830
         75%       0.651537     0.661970     0.673732     0.672980
         max       3.571767     3.424722     3.893606     4.104784
```

```
In [62]: col=data[2]
         col[np.abs(col)>3] #find values greater than 3 in column 2
```

```
Out[62]: 447    3.354485
         583    3.893606
         Name: 2, dtype: float64
```

```
In [65]: data[(np.abs(data)>3).any(1)] #report rows that have >3 in any column
```

```
Out[65]:             0         1         2         3
         222  0.429820 -0.247168 -1.145995  4.104784
         380 -3.067430  0.043376  0.709777 -1.326205
         412 -0.008728 -3.423739  1.061722 -0.398055
         447 -1.975929  1.117683  3.354485 -1.824912
         460  0.337453 -1.199839 -0.140934  3.216015
         561  3.571767 -0.080974 -0.362215 -1.887861
         571  1.186184  3.162137 -1.811221 -0.295279
         583 -0.477607  0.101242  3.893606  1.048426
         629  0.111325 -0.379214  0.862023 -3.094371
         977 -3.019376 -0.534652  1.155369  1.047623
         986  0.189540  3.424722  0.871550  0.452319
```

```
In [73]: data[np.abs(data)>3]=np.sign(data)*3  #replace outliers with 3 (keep the sign)
         data
```

```
Out[73]:             0         1         2         3
         0    0.405399  2.695794  0.564636  1.593455
         1    0.885846  0.324446  0.606096  0.915896
         2   -1.030575 -1.402759 -0.910587 -0.956956
         3    0.270255 -2.908266  0.460448 -2.941183
         4    1.430784  0.694300 -0.236944 -0.588769
         5    0.675679 -0.896593 -0.928476 -0.582420
         6   -0.465101 -0.484181 -2.008272  1.902356
         7   -0.075422 -1.194952 -0.955007  0.748794
```

```
8    0.434854 -1.295906 -0.060320  0.763275
9   -1.033090  0.277089 -0.866665  0.682160
..        ...       ...       ...       ...
990  0.243201  0.829935  0.662501  0.148890
991  1.358860 -1.216059 -0.703997 -2.141126
992  0.505308  0.188781 -0.541365 -0.686123
993  0.283435  0.345885  0.576676  0.388423
994 -0.004547 -0.626815 -1.372446  0.297942
995 -0.185451  0.130191 -0.424046  1.947049
996 -0.200710  0.909408 -0.599311 -1.033973
997 -0.538521 -2.125265 -0.475660 -0.774994
998  1.202417  0.082925 -0.858229 -1.828928
999 -0.327564 -0.267989  0.845160  0.371163

[1000 rows x 4 columns]
```

```
In [74]: np.sign(data).head() #check the sign only
```

```
Out[74]:      0    1    2    3
         0  1.0  1.0  1.0  1.0
         1  1.0  1.0  1.0  1.0
         2 -1.0 -1.0 -1.0 -1.0
         3  1.0 -1.0  1.0 -1.0
         4  1.0  1.0 -1.0 -1.0
```

0.1.4 Permutation and random sampling from the data

```
In [80]: df=pd.DataFrame(np.arange(5*4).reshape(5,4)) #create 5*4=20 consecutive values then r
         df
```

```
Out[80]:     0   1   2   3
         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
```

```
In [86]: sampler=np.random.permutation(5) #random arrangements of rows
         sampler
```

```
Out[86]: array([4, 1, 0, 3, 2])
```

```
In [87]: df.take(sampler) #arrange rows according to sampler
```

```
Out[87]:     0   1   2   3
         4  16  17  18  19
         1   4   5   6   7
         0   0   1   2   3
         3  12  13  14  15
         2   8   9  10  11
```

```
In [89]: df.sample(n=3) #now sample the first three rows
```

```
Out[89]:    0  1   2   3
         1  4  5   6   7
         2  8  9  10  11
         0  0  1   2   3
```

```
In [93]: # Another example: randomly choose values with replacement
         choices=pd.Series([5,7,-1,6,4]) #some random values
         choices
```

```
Out[93]: 0    5
         1    7
         2   -1
         3    6
         4    4
         dtype: int64
```

```
In [94]: draws=choices.sample(n=10, replace=True)
         draws
```

```
Out[94]: 0    5
         1    7
         0    5
         4    4
         4    4
         3    6
         4    4
         4    4
         4    4
         4    4
         dtype: int64
```