pandas - binning with bins definitions based on value in another column

I am struggling with such task: I need to discretize values in a column from data frame, with bins definition based on value in other column.

For a minimal working example, lets define a simple dataframe:

```
import pandas as pd
df = pd.DataFrame({'A' : ['one', 'one', 'two', 'three'] * 3,'B' : np.random.randn(12)})
```

The dataframe looks like this:

```
        A       B
0       one     2.5772143847077427
1       one     -0.6394141654096013
2       two     0.964652049995486
3       three   -0.3922889559403503
4       one     1.6903991754896424
5       one     0.5741442025742018
6       two     0.6300564981683544
7       three   0.9403680915507433
8       one     0.7044433078166983
9       one     -0.1695006646595688
10      two     0.06376190217285167
11      three   0.277540580579127
```

Now I would like to introduce column `C`, which will contain a bin label, with different bins for each of values in column `A`, i.e.:

- `(-10,-1,0,1,10)` for `A == 'one'`,
- `(-100,0,100)` for `A == 'two'`,
- `(-999,0,1,2,3)` for `A == 'three'`.

A desired output is:

```
        A       B       C
0       one     2.5772143847077427      (1, 10]
1       one     -0.6394141654096013     (-1, 0]
2       two     0.964652049995486       (0, 100]
3       three   -0.3922889559403503     (-999, 0]
4       one     1.6903991754896424      (1, 10]
5       one     0.5741442025742018      (0, 1]
6       two     0.6300564981683544      (0, 100]
7       three   0.9403680915507433      (0, 1]
8       one     0.7044433078166983      (0, 1]
9       one     -0.1695006646595688     (-1, 0]
10      two     0.06376190217285167     (0, 100]
11      three   0.277540580579127       (0, 1]
```

I have tried using `pd.cut` or `np.digitize` with different combinations of `map`, `apply`, but without success.

Currently I am achieving the result by splitting the frame and applying `pd.cut` to each subset separately, and then merging to get the frame back, like this:

```
values_in_column_A = df['A'].unique().tolist()
bins = {'one':(-10,-1,0,1,10),'two':(-100,0,100),'three':(-999,0,1,2,3)}

def binnize(df):

    subdf = []
    for i in range(len(values_in_column_A)):
        subdf.append(df[df['A'] == values_in_column_A[i]])
        subdf[i]['C'] = pd.cut(subdf[i]['B'],bins[values_in_column_A[i]])

    return pd.concat(subdf)
```

This works, but I do not think it is elegant enough, I also anticipate some speed or memory problems in production, when I will have frames with millions of rows. Speaking straight, I guess this could be done better.

I wolud appreciate any help or ideas...

python pandas dataframes binning

edited Aug 13 '13 at 14:07

asked Aug 13 '13 at 13:02

Paweł Rumian
709 7 19

1 Answer

Does this solves your problem?

```
df = pd.DataFrame({'A' : ['one', 'one', 'two', 'three'] * 3,
                   'B' : np.random.randn(12)})
bins = {'one': (-10,-1,0,1,10), 'two':(-100,0,100), 'three':(-999,0,1,2,3)}

def func(row):
    return pd.cut([row['B']], bins=bins[row['A']])[0]

df['C'] = df.apply(func, axis=1)
```

This returns a DataFrame:

```
        A         B          C
0     one  1.440957    (1, 10]
1     one  0.394580     (0, 1]
2     two -0.039619  (-100, 0]
3   three -0.500325  (-999, 0]
4     one  0.497256     (0, 1]
5     one  0.342222     (0, 1]
6     two -0.968390  (-100, 0]
7   three -0.772321  (-999, 0]
8     one  0.803178     (0, 1]
9     one  0.201513     (0, 1]
10    two  1.178546   (0, 100]
11  three -0.149662  (-999, 0]
```

Faster version of binnize:

```
def binize2(df):
    df['C'] = ''
    for key, values in bins.items():
        mask = df['A'] == key
        df.loc[mask, 'C'] = pd.cut(df.loc[mask, 'B'], bins=values)

%%timeit
df3 = binnize(df1)
10 loops, best of 3: 56.2 ms per loop

%%timeit
binize2(df2)
100 loops, best of 3: 6.64 ms per loop
```

This is probably due to the fact that it changes the DataFrame inplace and doesn't create a new one.

edited Aug 13 '13 at 14:32

answered Aug 13 '13 at 13:37

Viktor Kerkez
14.8k 2 32 43

It works, but is painfully slow (over 50 times slower)... For DataFrame with 12k rows: `timeit df['C'] = df.apply(func, axis=1)` , result: `1 loops, best of 3: 4.75 s per loop` To compare: `timeit df2 = binnize(df)` , result `10 loops, best of 3: 95.6 ms per loop` ... – Paweł Rumian Aug 13 '13 at 14:02

I guess it may come from the fact that `pd.cut` is optimized to work on columns, not on single rows... – Paweł Rumian Aug 13 '13 at 14:10

Your right, it seems that the boolean indexing and column wise calculation, works much much faster than the row wise operations... – Viktor Kerkez Aug 13 '13 at 14:24

1 Found a 10x faster version of binnize, will add it to the answer. :) – Viktor Kerkez Aug 13 '13 at 14:30

Yay!!! On a larger dataset it is even close to 20 times faster :) – Paweł Rumian Aug 13 '13 at 15:44