10. Lọc dữ liệu trong pandas

Trong các bài trước về pandas, tôi đã giới thiệu các cách truy nhập tự nhiên cho dataframe hay series, có thể nhận thấy rằng chúng rất tương đồng với cách truy cập mảng, list hay dictionary.

Trong phần này tôi sẽ hướng dẫn các kĩ thuật filtering để có thể select một phần dữ liệu trong Dataframe dựa vào nội dung của dữ liệu chứ không chỉ dựa vào index hay label như các bài trước.

.where()

Việc select các giá trị từ một Series với một Filter (véc tơ boolean) thường trả về một tập con của dữ liệu. Để đảm bảo rằng đầu ra của select có cùng hình dạng với dữ liệu ban đầu, bạn có thể sử dụng phương thức where trong Series và Dataframe. Ví dụ:

```
>>> s = pd.Series([12,34,56,78],index = [1,2,3,4])
>>> s
1    12
2    34
3    56
4    78
dtype: int64
>>> s>50
1    False
2    False
3     True
4     True
dtype: bool
>>> s[s>50]
3    56
4    78
dtype: int64
>>> s.where(s>50)
1     NaN
2     NaN
3    56.0
4    78.0
dtype: float64
>>>
```

Select các giá trị từ một dataframe khi truyền vào một Filter (Boolean vector) bây giờ cũng bảo tồn hình dạng dữ liệu đầu vào. “where” được sử dụng trong ví dụ dưới đây cho kết quả tương đương.

```
>>> df = pd.DataFrame(np.random.randn(5, 4)*100,columns = list("ABCD"))
>>> df
           A          B           C          D
0  102.502787 -68.034861  215.551919  -39.416507
1   89.068861  30.578270 -169.886261   32.943052
```

```
2   26.351777 -88.681893   42.678767   51.788262
3  -28.990230   8.767649   -5.079649  106.257066
4 -163.709170 -87.866018   43.412728  -22.622010
>>> df[df>0]
          A         B          C          D
0  102.502787       NaN  215.551919        NaN
1   89.068861  30.578270        NaN   32.943052
2   26.351777       NaN   42.678767   51.788262
3        NaN   8.767649        NaN  106.257066
4        NaN       NaN   43.412728        NaN
>>> df.where(df>0)
          A         B          C          D
0  102.502787       NaN  215.551919        NaN
1   89.068861  30.578270        NaN   32.943052
2   26.351777       NaN   42.678767   51.788262
3        NaN   8.767649        NaN  106.257066
4        NaN       NaN   43.412728        NaN
>>>
```

Tuy nhiên, trong trường hợp dùng “where”có một đối số tùy chọn khác để thay thế các giá trị mà điều kiện là False, trong bản sao được trả về.

```
>>> df.where(df>0,-df)
          A         B          C          D
0  102.502787  68.034861  215.551919   39.416507
1   89.068861  30.578270  169.886261   32.943052
2   26.351777  88.681893   42.678767   51.788262
3   28.990230   8.767649    5.079649  106.257066
4  163.709170  87.866018   43.412728   22.622010
>>>
```

Chú ý, tùy chọn inplace=True giúp where chỉnh sửa tại chỗ dataframe đang query.

.query(expr, inplace=False, **kwargs)

Các đối tượng dataframe có một phương thức query() cho phép select dữ liệu thông qua việc sử dụng một biểu thức.

Bạn có thể lấy giá trị của dataframe mà cột b có các giá trị nằm giữa các giá trị của các cột a và c. Ví dụ sau đây sẽ trình bày 2 cách cho cùng một kết quả trả về. Cách một dùng cú pháp giống numy thông thường, cách còn lại dùng theo cú pháp sql.

```
>>> df = pd.DataFrame(np.random.randint(100, size=(10, 3)), columns=list('abc'))
>>> df
   a   b   c
0  51  91  35
1  57  12  91
2  40  64  16
3  30  36  28
4  11  69  89
```

```
5  96   7  72
6  66  20  44
7  85  16  29
8  16  12  21
9  80  48  91
>>> df[(df.a < df.b) & (df.b < df.c)]
   a   b   c
4  11  69  89
>>> print("cach lam tuong duong khi dung query")
cach lam tuong duong khi dung query
>>> df.query('a < b and b < c')
   a   b   c
4  11  69  89
>>>
```

Trường hợp sử dụng query() là khi bạn có một tập hợp dataframe có một tập con của các tên cột (hoặc các level chỉ mục /tên) chung. Bạn có dùng cùng một truy vấn cho cả hai dataframes mà không cần phải chỉ định dataframe mà bạn quan tâm đến việc truy vấn. Ví dụ:

```
>>> df = pd.DataFrame(np.random.randint(100,size = (5, 3)), columns=list('abc'))
>>> df1 = pd.DataFrame(np.random.randint(100,size=(8, 3)), columns=df.columns)
>>> expr = '0.0 <= a <= c <= 100'
>>> map(lambda frame: frame.query(expr), [df, df1])
[   a   b   c
3  21  66  63
6   1  17  62
7   7  50  65,    a   b   c
0  42  40  73
3  27  49  34
4  67  89  76
7  24  46  64]
>>>
```

Toán tử ‘in’ và ‘not in’

Query() cũng hỗ trợ sử dụng toán tử so sánh đặc biệt trong Python “in” và “not in”, cung cấp một cú pháp ngắn gọn để gọi phương thức isin() của Series hoặc DataFrame.

```
>>> df = pd.DataFrame({'a': list('aabbccddeeff'), 'b': list('aaaabbbbcccc'),'c': np.random.randint(5, size=12), 'd': np.random.randint(9, size=12)})
>>> df
   a  b  c  d
0  a  a  1  1
1  a  a  1  4
2  b  a  4  2
3  b  a  2  7
4  c  b  2  2
5  c  b  2  3
6  d  b  2  0
7  d  b  4  0
8  e  c  0  2
9  e  c  1  1
10 f  c  0  3
11 f  c  2  4
>>> df.query('a in b')
  a  b  c  d
0 a  a  1  1
1 a  a  1  4
2 b  a  4  2
3 b  a  2  7
4 c  b  2  2
5 c  b  2  3
>>>
```

Cách tương tự với ví dụ trên nhưng sử dụng isin()

```
>>> df[df.a.isin(df.b)]
  a  b  c  d
0 a  a  1  1
1 a  a  1  4
2 b  a  4  2
3 b  a  2  7
4 c  b  2  2
5 c  b  2  3
>>>
```

Cách dùng tương tự với “not in” được so sánh với ~isin().

```
>>> df.query('a not in b')
   a b c d
6  d b 2 0
7  d b 4 0
8  e c 0 2
9  e c 1 1
10 f c 0 3
11 f c 2 4
>>> df[~df.a.isin(df.b)]
   a b c d
6  d b 2 0
7  d b 4 0
8  e c 0 2
9  e c 1 1
10 f c 0 3
11 f c 2 4
>>>
```

Các bộ Filter có thể được kết hợp với nhau qua các toán tử logic như &, |, ~ hay or, and, != … như trong SQL. Cần chú ý rằng mỗi bộ Filter con cần được đặt trong dấu ngoặc ().

Bạn có thể kết hợp “in” hay “not in” với các biểu thức khác cho các truy vấn rất ngắn gọn:

```
>>> df.query('a in b and c < d')
  a b c d
1 a a 1 4
3 b a 2 7
5 c b 2 3
>>>
```

```
>>> df[df.a.isin(df.b) & (df.c < df.d)]
  a b c d
1 a a 1 4
3 b a 2 7
5 c b 2 3
>>>
```

Khi sử dụng toán tử “==” hay”!=” với list ta có kết quả tương đồng với toán tử “in” và “not in” ví dụ:

```
>>> df.query('a == ["a", "b", "c"]')
  a b c d
0 a a 1 1
1 a a 1 4
2 b a 4 2
3 b a 2 7
4 c b 2 2
5 c b 2 3
>>> df.query('a != ["a", "b", "c"]')
   a b c d
6  d b 2 0
7  d b 4 0
8  e c 0 2
9  e c 1 1
10 f c 0 3
11 f c 2 4
>>>
```

```
>>> df.query('a in ["a", "b", "c"]')
  a b c d
0 a a 1 1
1 a a 1 4
2 b a 4 2
3 b a 2 7
4 c b 2 2
5 c b 2 3
>>> df.query('a not in ["a", "b", "c"]')
   a b c d
6  d b 2 0
7  d b 4 0
8  e c 0 2
9  e c 1 1
10 f c 0 3
11 f c 2 4
>>>
```

Series, dataframe cũng cung cấp phương thức get() hoạt động kiểu dictionary và có thể giúp ta gán giá trị mặc định nếu không get được dữ liệu.

```
>>> df.get('d',[1,2,3,4])
0    1
1    4
2    2
3    7
4    2
5    3
6    0
7    0
8    2
```

```
9    1
10   3
11   4
Name: d, dtype: int32
>>> df.get('e',[1,2,3,4])
[1, 2, 3, 4]
>>>
```

.select(function, axis=0)

Một cách khác để truy xuất từ một đối tượng là với phương pháp select của Series, dataframe và Panel. Phương pháp này chỉ nên sử dụng khi không có cách nào trực tiếp. Select có một callable và đối số truyền vào của callable là nhãn các trục theo cột hoặc hàng boolean. Ví dụ:

```
>>> df.select(lambda x: x!='b',axis =1)
   a  c  d
0  a  1  1
1  a  1  4
2  b  4  2
3  b  2  7
4  c  2  2
5  c  2  3
6  d  2  0
7  d  4  0
8  e  0  2
9  e  1  1
10 f  0  3
11 f  2  4
>>>
```

```
>>> df.select(lambda x: x >5,axis = 0)
   a  b  c  d
6  d  b  2  0
7  d  b  4  0
8  e  c  0  2
9  e  c  1  1
10 f  c  0  3
11 f  c  2  4
>>>
```

.lockup(row_lables, col_labels)

Đôi khi bạn muốn trích xuất một tập hợp các giá trị cho một dãy các hàng và các cột, và lookup() cho phép điều này và trả về một mảng numpy của các giá trị tương ứng với mỗi cặp (row,col). Ví dụ,

```
>>> dflookup = pd.DataFrame(np.random.rand(20,4), columns = ['A','B','C','D'])
>>> dflookup
          A         B         C         D
0  0.875261  0.874488  0.056862  0.744732
1  0.028420  0.322262  0.257275  0.315512
2  0.039933  0.747299  0.559197  0.402576
3  0.925743  0.453353  0.699870  0.313097
4  0.089513  0.448037  0.006460  0.967212
5  0.302161  0.374231  0.422943  0.421142
6  0.730934  0.599732  0.247673  0.774336
7  0.240224  0.571432  0.481461  0.310269
8  0.726486  0.539212  0.985579  0.199197
9  0.297361  0.325757  0.130587  0.099352
10 0.756854  0.706253  0.049746  0.439951
11 0.353714  0.201315  0.702448  0.168419
12 0.780751  0.866134  0.779492  0.434104
13 0.621378  0.178430  0.095091  0.613194
14 0.080736  0.889075  0.209379  0.949866
15 0.845428  0.834246  0.404687  0.409837
16 0.177463  0.130453  0.885461  0.963109
17 0.508278  0.552224  0.205997  0.049991
18 0.056802  0.854984  0.383322  0.275022
19 0.804973  0.130914  0.085803  0.894932
>>> list(range(0,10,2))
[0, 2, 4, 6, 8]
>>> dflookup.lookup(list(range(0,10,2)), ['B','C','A','B','D'])
array([ 0.87448836,  0.5591968 ,  0.08951277,  0.5997316 ,  0.19919737])
```

Giá trị 0.87448836 chính là kết quả của cột ‘B’ ứng với hàng có index là 0, bạn có thể giải thích với các số khác.

Kết luận

Có thể thấy Pandas hỗ trợ khá đa dạng các phương thức để truy xuất, filtering một tập dữ liệu trong DataFrame hay series. Trong bài này chúng tôi đã đề cập đến một số phương thức hay dùng trong xử lý dữ liệu như Series.where()/DataFrame.where(), DataFrame.query() với tùy chọn inplace hỗ trợ chỉnh sửa trên chính dữ liệu gốc, đồng thời biểu thức filtering của các hàm này còn có thể sử dụng các toán tử so sánh đặc biệt như 'in'/'not in' và các toán tử logic khác hỗ trợ người dùng nhiều tùy chọn filtering dữ liệu.

Ngoài ra các bạn có thể tham khảo một số cách truy xuất khác có thể sử dụng như DataFrame.get(), DataFrame.lookup() hay DataFrame.lookup()...