Phân tích dữ liệu

30-12-2020

1 Giới thiệu

Tổng hợp thông tin là một phần quan trọng trong việc phân tích dữ liệu. Trong bài này, chúng ta sẽ học một số phương thức tổng hợp thông tin từ `DataFrame`.

1.1 Dữ liệu

Chúng ta sẽ sử dụng dữ liệu được thu thập về các hành tinh ngoài hệ mặt trời (exoplanet).

```
In [1]: import seaborn as sns
        planets = sns.load_dataset('planets')
        planets.shape
```

```
Out[1]: (1035, 6)
```

```
In [2]: planets.head()
```

```
Out[2]:              method  number  orbital_period   mass  distance  year
        0  Radial Velocity       1         269.300   7.10     77.40  2006
        1  Radial Velocity       1         874.774   2.21     56.95  2008
        2  Radial Velocity       1         763.000   2.60     19.84  2011
        3  Radial Velocity       1         326.030  19.40    110.62  2007
        4  Radial Velocity       1         516.220  10.50    119.47  2009
```

2 Tính toán với `Series` và `DataFrame`

2.1 Tính toán cơ bản

Người dùng có thể thực hiện các phép tính với `Series` một cách tiện lợi

```
In [3]: import pandas as pd
```

```
In [4]: s = pd.Series([1, 2, 3])
        # thực hiện phép tính giữa Series và một số
        s / 2
```

```
Out[4]: 0    0.5
        1    1.0
        2    1.5
        dtype: float64
```

```
In [5]: s = pd.Series([1996, 2020])
        s.astype('str').str[0:3] + '0s'

Out[5]: 0    1990s
        1    2020s
        dtype: object

In [6]: # thực hiện phép tính giữa Series và Series khác
        s1 = pd.Series([1, 2, 3])
        s2 = pd.Series([1, 2, 5])

        s1 * s2

Out[6]: 0     1
        1     4
        2    15
        dtype: int64
```

2.2 Sử dụng `.apply()`

Đôi khi một số thao tác tính toán không thể được thực hiện qua các phép tính đơn giản mà phải làm thông qua một hàm nào đó (có sẵn hoặc người dùng định nghĩa), chúng ta có thể dùng phương thức lặp qua từng phần tử của `Series` (hoặc từng dòng của `DataFrame`) để thực hiện thao tác tính toán đó.

```
In [7]: import numpy as np
        # tính căn bậc hai từng số trong Series
        t = [np.sqrt(i) for i in s]
        pd.Series(t)

Out[7]: 0    44.676616
        1    44.944410
        dtype: float64
```

Tuy nhiên, `pandas` cung cấp một phương thức tiện lợi, đó là `.apply(<hàm_cần_dùng>)`. `.apply(<hàm_cần_dùng>)` sẽ áp dụng `<hàm_cần_dùng>` cho từng phần tử của `Series` (hoặc từng dòng của `DataFrame`).

Cú pháp của `.apply()` như sau: `.apply(<hàm_cần_dùng>, args, **kargs)`

Trong đó:

- `<hàm_cần_dùng>`: là tên hàm muốn sử dụng (không có `()`).
- `args`: là danh sách các tham số (truyền theo vị trí) bổ sung của `<hàm_cần_dùng>`.
- `**kargs`: là các tham số (truyền theo tên) bổ sung của `<hàm_cần_dùng>`.

```
In [8]: # ví dụ, tính căn bậc hai từng số trong Series
        s.apply(np.sqrt)

Out[8]: 0    44.676616
        1    44.944410
        dtype: float64
```

```
In [9]: # ví dụ, dùng hàm người dùng định nghĩa
        def calculate_decade(x):
            return str(x // 10 * 10) + 's'

        s3 = pd.Series([1991, 2002, 2027])
        s3.apply(calculate_decade)

Out[9]: 0    1990s
        1    2000s
        2    2020s
        dtype: object

In [10]: # ví dụ dùng args
         def find_max(a, b):
             if a > b:
                 return a
             return b

         s4 = pd.Series([1, 2, 3, 4])
         s4.apply(find_max, args =[2])

Out[10]: 0    2
         1    2
         2    3
         3    4
         dtype: int64
```

Khi sử dụng `.apply()` với `DataFrame`, chúng ta còn một tham số, đó là `axis`, tham số này sẽ chỉ ra các áp dụng <hàm_cần_dùng> theo hàng (`axis = 'columns'`) hay theo cột (`axis = 'index'`) của `DataFrame`. Mặc định sẽ là `axis = 'index'`.

```
In [11]: # dùng .apply() trên DataFrame trên từng cột
         df = pd.DataFrame({
             'new': [1, 2, 3],
             'old': [2, 3, 4]
         })

         df.apply(np.sqrt)

Out[11]:         new       old
         0  1.000000  1.414214
         1  1.414214  1.732051
         2  1.732051  2.000000

In [12]: # dùng .apply() trên DataFrame theo dòng
         def calculate_percent(x):
             t = (x.new - x.old) / x.old * 100
             return round(t, 2)

         df = pd.DataFrame({
             'new': [1, 2, 3],
```

```
            'old': [2, 3, 4]
        })

        df.apply(calculate_percent, axis = 'columns')
```

```
Out[12]: 0   -50.00
         1   -33.33
         2   -25.00
         dtype: float64
```

3 Tổng hợp thông tin cơ bản

Nhắc lại, chúng ta có các phương thức tổng hợp cơ bản sau:

- `.count()`: số lượng phần tử khác `NaN`.
- `.nunique()`: số lượng các giá trị khác nhau.
- `.min()`, `.max()`: giá trị nhỏ nhất, giá trị lớn nhất.
- `.mean()`, `.median()`: giá trị trung bình, giá trị trung vị.
- `.var()`, `.std()`: phương sai, độ lệch chuẩn.
- `.sum()`: tổng các phần tử.
- `.prod()`: tích các phần tử.

Các phương thức trên đều có thể dùng trên `Series` và `DataFrame`.
Nếu dùng trên `Series`, các phương thức trên sẽ trả về một kết quả duy nhất.

```
In [13]: m = planets.mass
         print('Số lượng hành tinh có khối lượng', m.count())
         print('Khối lượng lớn nhất là', m.max())
         print('Khối lượng bé nhất là', m.min())
         print('Khối lượng trung bình là', m.mean())
         print('Phương sai là', m.var())
         print('Độ lệch chuẩn là', m.std())
```

```
Số lượng hành tinh có khối lượng 513
Khối lượng lớn nhất là 25.0
Khối lượng bé nhất là 0.0036
Khối lượng trung bình là 2.6381605847953216
Phương sai là 14.58183312700122
Độ lệch chuẩn là 3.8186166509616046
```

Khi dùng các phương thức trên, `pandas` sẽ tự động bỏ qua các giá trị `NaN`.

Khi dùng các phương thức tổng hợp thông tin của `DataFrame`, kết quả là một `Series` với index là các cột (hoặc dòng) của `DataFrame` và data là kết quả của phương thức tổng hợp.

Ngoài ra, khi làm việc với `DataFrame`, các phương thức có một tham số là `axis`. Tham số này chỉ ra nên áp dụng phương thức theo dòng (`axis = 'columns'`) hay theo cột (`axis = 'index'`). Mặc định luôn là `axis = 'index'`.

```
In [14]: planets.mean()
```

```
Out[14]: number              1.785507
         orbital_period    2002.917596
         mass                 2.638161
         distance           264.069282
         year              2009.070531
         dtype: float64

In [15]: planets.mean(axis = 'columns')

Out[15]: 0       472.160000
         1       588.586800
         2       559.488000
         3       492.810000
         4       531.238000
                    …
         1030    545.735377
         1031    539.653966
         1032    546.297881
         1033    576.531271
         1034    568.296939
         Length: 1035, dtype: float64
```

Chúng ta còn có các phương thức tổng hợp khác:

- `.cov()`: hiệp phương sai.
- `.corr()`: hệ số tương quan.

Cách gọi hai phương thức này có sự khác biệt giữa `Series` và `DataFrame`.
Đối với `Series`, bạn cần phải chỉ ra một `Series` khác khi gọi `.cov()` và `.corr()`.

```
In [16]: m = planets.mass
         d = planets.distance

         print('Hiệp phương sai giữa khối lượng và khoảng cách là', m.cov(d))
         print('Hệ số tương quan giữa khối lương và khoảng cách là', m.corr(d))
```

```
Hiệp phương sai giữa khối lượng và khoảng cách là 46.439430678341544
Hệ số tương quan giữa khối lương và khoảng cách là 0.2740824509615061
```

Đối với `DataFrame`, bạn chỉ cần gọi hai phương thức trên, kết quả sẽ là ma trận hiệp phương sai hoặc ma trận tương quan giữa các cột trong `DataFrame`.

```
In [17]: planets.cov()

Out[17]:                      number  orbital_period         mass       distance  \
         number             1.540022   -4.109673e+02    -1.073362     -31.780638
         orbital_period  -410.967320    6.767661e+08  1005.472594 -255305.071038
         mass              -1.073362    1.005473e+03    14.581833      46.439431
         distance         -31.780638   -2.553051e+05    46.439431  537459.792221
         year               0.728876   -3.367808e+03    -1.963811     517.198294

                              year
```

```
number            0.728876
orbital_period -3367.807803
mass              -1.963811
distance         517.198294
year              15.781287
```

```
In [18]: planets.corr()
```

Out[18]:

	number	orbital_period	mass	distance	year
number	1.000000	-0.012570	-0.241429	-0.033638	0.147849
orbital_period	-0.012570	1.000000	0.173725	-0.034365	-0.032333
mass	-0.241429	0.173725	1.000000	0.274082	-0.123787
distance	-0.033638	-0.034365	0.274082	1.000000	0.178922
year	0.147849	-0.032333	-0.123787	0.178922	1.000000

Ngoài ra, chúng ta còn một phương thức tiện lợi để tổng hợp thông tin, đó là `.describe()`. Phương thức này sẽ trả về thống kê mô tả của `Series` và `DataFrame`.

```
In [19]: m.describe()
```

```
Out[19]: count    513.000000
         mean       2.638161
         std        3.818617
         min        0.003600
         25%        0.229000
         50%        1.260000
         75%        3.040000
         max       25.000000
         Name: mass, dtype: float64
```

```
In [20]: planets.describe()
```

Out[20]:

	number	orbital_period	mass	distance	year
count	1035.000000	992.000000	513.000000	808.000000	1035.000000
mean	1.785507	2002.917596	2.638161	264.069282	2009.070531
std	1.240976	26014.728304	3.818617	733.116493	3.972567
min	1.000000	0.090706	0.003600	1.350000	1989.000000
25%	1.000000	5.442540	0.229000	32.560000	2007.000000
50%	1.000000	39.979500	1.260000	55.250000	2010.000000
75%	2.000000	526.005000	3.040000	178.500000	2012.000000
max	7.000000	730000.000000	25.000000	8500.000000	2014.000000

```
In [21]: # bỏ các giá trị NaN
         planets.dropna().describe()
```

Out[21]:

	number	orbital_period	mass	distance	year
count	498.00000	498.000000	498.000000	498.000000	498.000000
mean	1.73494	835.778671	2.509320	52.068213	2007.377510
std	1.17572	1469.128259	3.636274	46.596041	4.167284
min	1.00000	1.328300	0.003600	1.350000	1989.000000
25%	1.00000	38.272250	0.212500	24.497500	2005.000000
50%	1.00000	357.000000	1.245000	39.940000	2009.000000
75%	2.00000	999.600000	2.867500	59.332500	2011.000000
max	6.00000	17337.500000	25.000000	354.000000	2014.000000