Làm việc với NaN

30-12-2020

1 Giá trị NaN

1.1 Giới thiệu

Trong bài học này, chúng ta sẽ học một số cách làm việc với giá trị NaN. Đây là một giá trị đặc biệt của Python dùng để ký hiệu giá trị bị thiếu trong dữ liệu.

Để tạo giá trị NaN, chúng ta có 2 cách:

- `float('nan')`
- Dùng `numpy.nan`

```
In [1]: import numpy as np
        import pandas as pd

        a = float('nan')
        b = np.nan

        print(a, b)
        print(type(a), type(b))
```

```
nan nan
<class 'float'> <class 'float'>
```

1.2 Một số đặc điểm của NaN

NaN là một giá trị đặc biệt dùng để đánh dấu dữ liệu bị thiếu, do đó, việc so sánh bằng hai giá trị NaN sẽ trả về False.

```
In [2]: print(np.nan == np.nan)

        print(float('nan') == float('nan'))
```

```
False
False
```

Bạn có thể dùng phép so sánh trên để kiểm tra một giá trị có phải là NaN hay không. Tuy nhiên, khi làm việc với pandas, ta có thể dùng hàm `pd.isna()` để kiểm tra một giá trị có phải là NaN hay không.

```
In [3]: print(pd.isna(np.nan))
```

```
True
```

Các phép toán số học với `NaN` đều trả về `NaN`.

```
In [4]: nan = float('nan')

        print(10 + nan, 10 - nan, 10 * nan, 10 / nan)
        print(nan ** 2, 2 ** nan)
        print(nan / nan)
```

```
nan nan nan nan
nan nan
nan
```

```
In [5]: # tuy nhiên, vẫn có ngoại lệ
        print (1 ** nan)
        print (nan ** 0)
```

```
1.0
1.0
```

Các phép toán logic mà một trong hai vế là `Nan` cũng trả về `NaN`, trừ hai trường hợp:

- `True or np.nan`
- `False and np.nan`

```
In [6]: # hai trường hợp đặc biệt
        print(True or nan)
        print(False and nan)
```

```
True
False
```

```
In [7]: # trường hợp bình thường
        print(nan or nan)
        print(nan and nan)
        print(False or nan)
        print(True and nan)
```

```
nan
nan
nan
nan
```

Khi so sánh, nếu một trong hai vế là `NaN`, ta có:

- Phép khác `!=` trả về `True`.

- Phép >, >=, <, <=, == trả về `False`

```
In [8]: print(nan > 1, nan < 1, nan == 1)
        print(nan != 0, nan != 1, nan != nan)
```

```
False False False
True True True
```

1.3 NaN và pandas

`pandas` cũng dùng `NaN` cho giá trị bị thiếu, tuy nhiên, tùy thuộc vào `dtype` mà `NaN` được hiển thị với tên khác nhau.

```
In [9]: # NaN với dtype là số (float64)
        s = pd.Series([1, 2, nan])
        s
```

```
Out[9]: 0    1.0
        1    2.0
        2    NaN
        dtype: float64
```

```
In [10]: # NaN với dtype là thời điểm
         pd.Series(
             pd.to_datetime(
                 ['2020-12-24', '2020-12-25', 'abc'],
                 errors = 'coerce'
             )
         )
```

```
Out[10]: 0   2020-12-24
         1   2020-12-25
         2          NaT
         dtype: datetime64[ns]
```

```
In [11]: # NaN với dtype là object
         pd.Series([1, 2, nan], dtype = str)
```

```
Out[11]: 0      1
         1      2
         2    NaN
         dtype: object
```

Trong phần kế tiếp, chúng ta sẽ tìm hiểu về cách làm với `NaN` trong `DataFrame`.

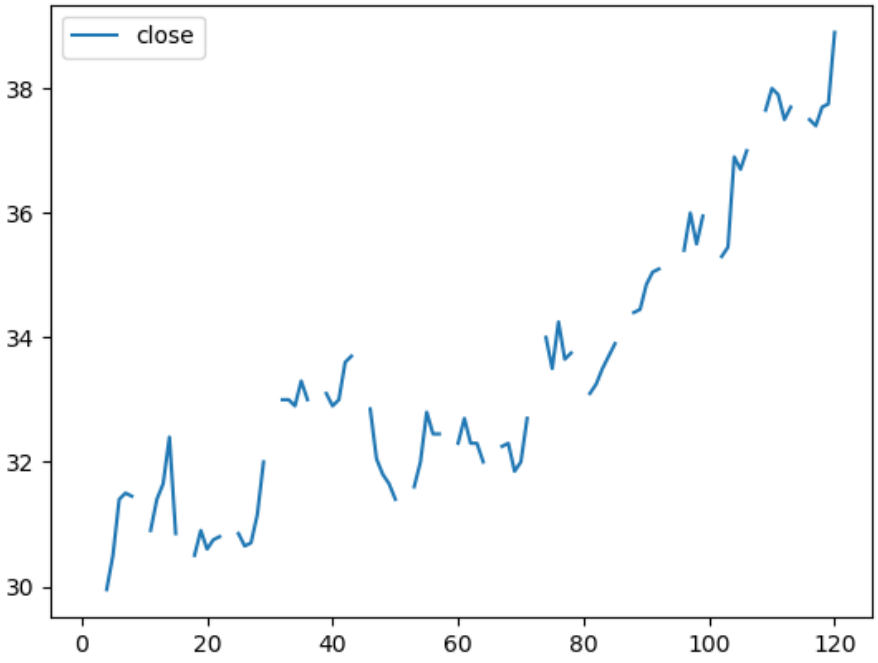
2 Làm việc với NaN

Trong phần này, chúng ta làm việc với dữ liệu sau

```
In [12]: data = pd.read_csv('HPG.csv', index_col = 0)
         print(data.head())
```

```
         date  close
0  2017-06-01    NaN
1  2017-06-02  30.00
2  2017-06-03    NaN
3  2017-06-04    NaN
4  2017-06-05  29.95
```

```
In [14]: import matplotlib.pyplot as plt
         %matplotlib notebook

         data.plot(y = 'close')
```



Dữ liệu gốc

2.1 Kiểm tra giá trị NaN

Để kiểm tra giá trị `NaN` trong một `Series` hay `DataFrame`, ta có thể dùng hàm `pd.isna()` ở trên.

```
In [15]: data[pd.isna(data.close)].head()
```

```
Out[15]:          date  close
         0  2017-06-01    NaN
         2  2017-06-03    NaN
         3  2017-06-04    NaN
         9  2017-06-10    NaN
         10 2017-06-11    NaN
```

Hoặc có thể dùng phương thức `.isna()` của `DataFrame` hoặc `Series`.

```
In [17]: data[data.close.isna()].head()
```

```
Out[17]:           date  close
         0   2017-06-01    NaN
         2   2017-06-03    NaN
         3   2017-06-04    NaN
         9   2017-06-10    NaN
         10  2017-06-11    NaN
```

Ghi chú: phương thức `.count()` sẽ đếm số lượng phần tử khác `NaN` trong mỗi cột.

```
In [18]: data.count()
```

```
Out[18]: date     122
         close     85
         dtype: int64
```

Để xử lý giá trị `NaN` trong `DataFrame`, có 2 phương pháp chung, lần lượt là:

- Xóa dòng hoặc cột chứa `NaN`.
- Thay giá trị `NaN` bằng một giá trị nào đó.

2.2 Xóa giá trị `NaN`

Để xóa các giá trị `NaN` có thể dùng phương thức: `.dropna(axis, how)`
Trong đó:

- `axis`: xóa `NaN` trong dòng hay trong cột. Có 2 giá trị `'columns'` và `'index'`. Mặc định là `'index'`.
- `how`: có 2 giá trị `'any'` và `'all'`. `'any'` sẽ xóa nếu như dòng hoặc có ít nhất 1 giá trị `NaN`. `'all'` sẽ xóa nếu toàn bộ dòng hoặc cột là `NaN`. Mặc định là `'any'`.

```
In [19]: data.dropna(axis = 'columns', how = 'all')
```

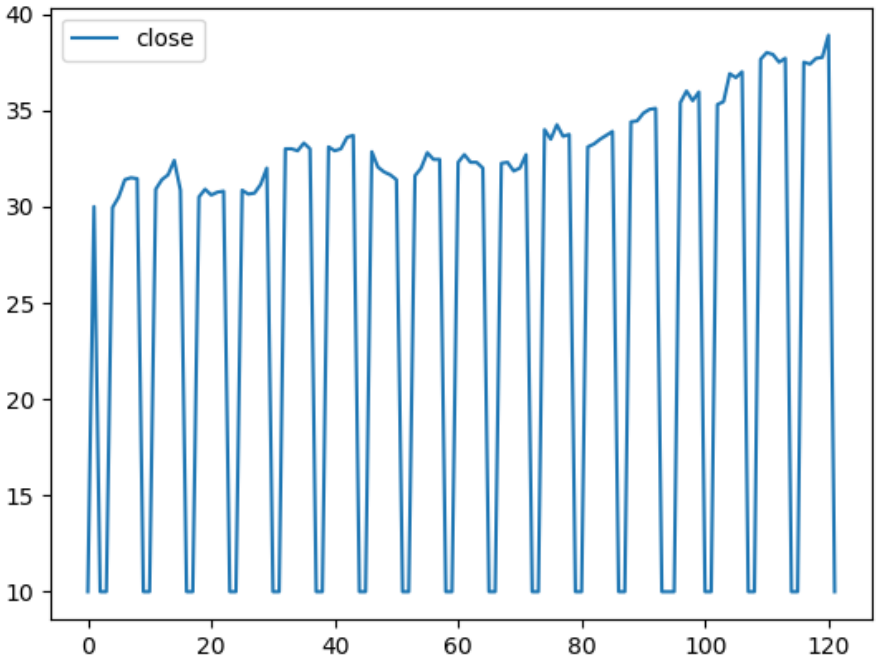
```
Out[19]:           date  close
         0    2017-06-01    NaN
         1    2017-06-02  30.00
         2    2017-06-03    NaN
         3    2017-06-04    NaN
         4    2017-06-05  29.95
         ..          …    …
         117  2017-09-26  37.40
         118  2017-09-27  37.70
         119  2017-09-28  37.75
         120  2017-09-29  38.90
         121  2017-09-30    NaN

         [122 rows x 2 columns]
```

2.3 Thay đổi giá trị NaN

2.3.1 Gán giá trị mới

Nhắc lại, bằng `.loc/.iloc`, ta có thể lấy ra những phần có giá trị `NaN` rồi gán cho nó giá trị mới.

```
In [20]: t = data.copy()
         t.close.loc[t.close.isna()] = 10
         t.plot()
```



`t.close.loc[t.close.isna()] = 10`

Chúng ta còn có thể dùng phương thức `.fillna()` để thay giá trị `NaN`.
Nếu muốn thay bằng một giá trị nào đó, ta có cú pháp: `.fillna(<giá_trị_cần_thay>)`.

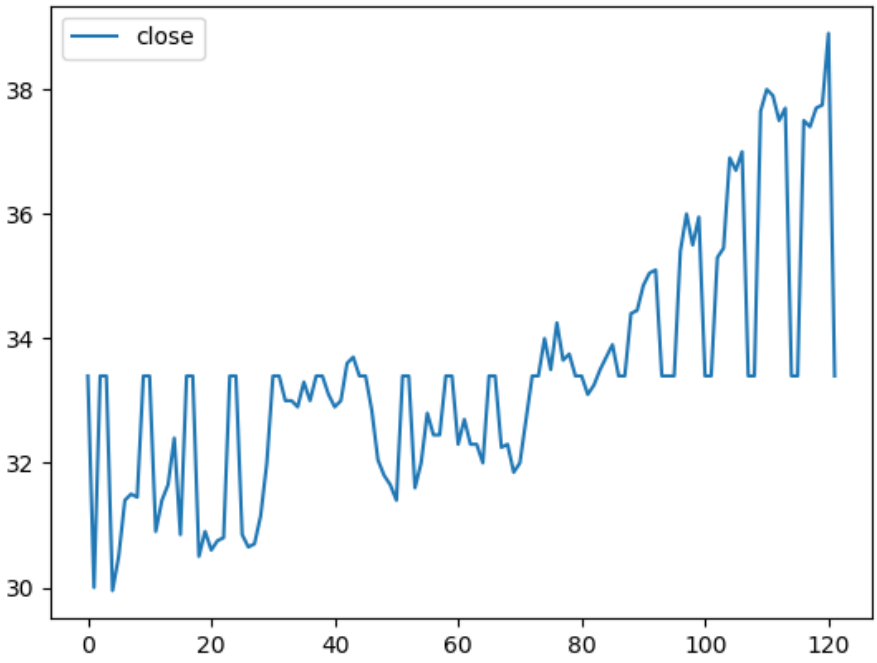
```
In [21]: data.fillna(data.close.mean()).plot()
```

Ngoài ra, `.fillna()` còn cung cấp khả năng thay giá trị `NaN` bằng các giá trị khác `NaN` liền trước hay liền sau với cú pháp `.fillna(method)`
Trong đó:

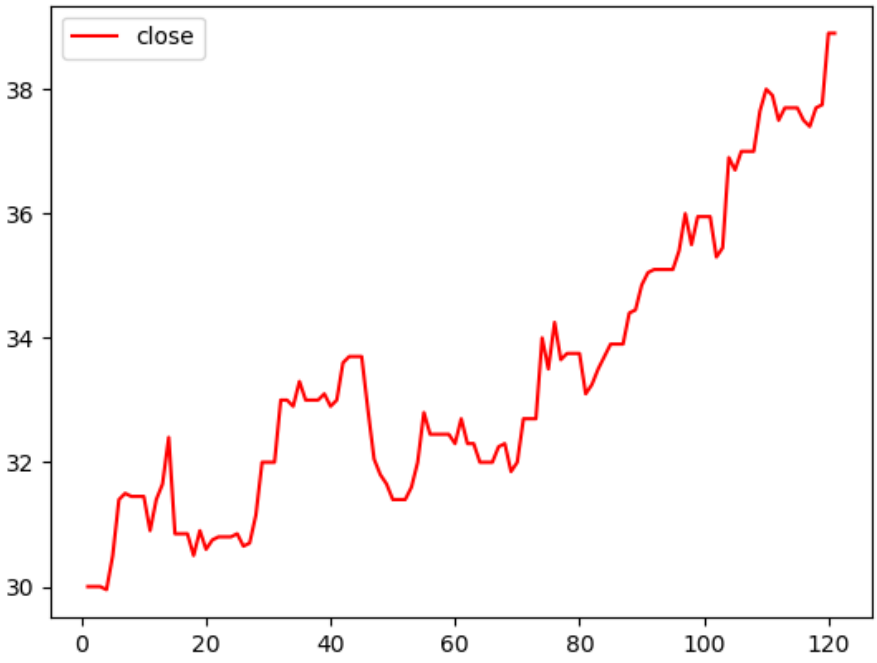
- `method = 'ffill'`, thay `NaN` bằng giá trị khác `NaN` liền trước.
- `method = 'bfill'`, thay `NaN` bằng giá trị khác `NaN` liền sau.

```
In [22]: data.fillna(method = 'ffill').plot(color = 'red')
```

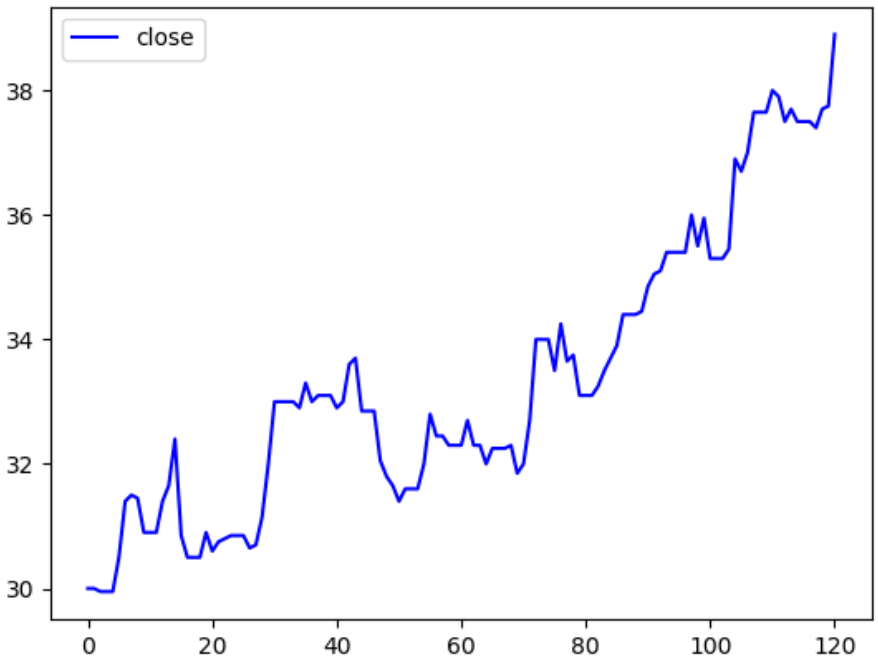
```
In [23]: data.fillna(method = 'bfill').plot(color = 'blue')
```




```
data.fillna(data.close.mean())
```




```
data.fillna(method = 'ffill')
```




```
data.fillna(method = 'bfill')
```