10 minutes to pandas

Đây là phần giới thiệu ngắn về gấu trúc, chủ yếu dành cho người dùng mới. Bạn có thể xem các công thức nấu ăn phức tạp hơn trong Sách dạy nấu ăn .

Nhập như sau:

```
import numpy as np
import pandas as pd
```

Tạo đối tượng

Tạo một **Series** bằng cách chuyển một danh sách các giá trị, cho phép gấu trúc tạo một danh sách số nguyên mặc định:

```
s = pd.Series([1, 3, 5, np.nan, 6, 8])

s
Out[4]:
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

Tạo một DataFrame bằng cách truyền một mảng NumPy, với chỉ mục ngày giờ và các cột được gắn nhãn:

```
dates = pd.date_range("20130101", periods=6)

dates
Out[6]:
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')

df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))

df
Out[8]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
```

Tạo một DataFrame bằng cách chuyển một lệnh của các đối tượng có thể được chuyển đổi thành dạng chuỗi.

```
df2 = pd.DataFrame(
    {
        "A": 1.0,
        "B": pd.Timestamp("20130102"),
        "C": pd.Series(1, index=list(range(4)), dtype="float32"),
        "D": np.array([3] * 4, dtype="int32"),
        "E": pd.Categorical(["test", "train", "test", "train"]),
        "F": "foo",
    }
)


df2
Out[10]:
     A          B    C  D      E    F
0  1.0 2013-01-02  1.0  3   test  foo
1  1.0 2013-01-02  1.0  3  train  foo
2  1.0 2013-01-02  1.0  3   test  foo
3  1.0 2013-01-02  1.0  3  train  foo
```

Các cột của kết quả DataFrame có khác nhau dtypes .

```
df2.dtypes
Out[11]:
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

Nếu bạn đang sử dụng IPython, tính năng hoàn thành tab cho tên cột (cũng như các thuộc tính công khai) sẽ tự động được bật. Dưới đây là một tập hợp con của các thuộc tính sẽ được hoàn thành:

```
df2.<TAB>  # noqa: E225, E999
df2.A                  df2.bool
df2.abs                df2.boxplot
df2.add                df2.C
df2.add_prefix         df2.clip
df2.add_suffix         df2.columns
df2.align              df2.copy
df2.all                df2.count
df2.any                df2.combine
df2.append             df2.D
df2.apply              df2.describe
df2.applymap           df2.diff
df2.B                  df2.duplicated
```

Có thể thấy, các cột A, B, C, và D sẽ được tự động tab hoàn thành. E và F ở đó; phần còn lại của các thuộc tính đã được cắt bớt cho ngắn gọn.

Xem dữ liệu

Xem phần Kiến thức cơ bản . Đây là cách xem các hàng trên cùng và dưới cùng của khung:

```
df.head()
Out[13]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401

df.tail(3)
Out[14]:
                   A         B         C         D
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
```

Hiển thị chỉ mục, các cột:

```
df.index
Out[15]:
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')

df.columns
Out[16]: Index(['A', 'B', 'C', 'D'], dtype='object')
```

DataFrame.to_numpy() cung cấp một biểu diễn NumPy của dữ liệu cơ bản. Lưu ý rằng đây có thể là một thao tác tốn kém khi DataFramecác cột của bạn có các kiểu dữ liệu khác nhau, dẫn đến sự khác biệt cơ bản giữa gấu trúc và NumPy: Mảng NumPy có một kiểu cho toàn bộ mảng, trong khi các DataFrames của gấu trúc có một kiểu cho mỗi cột . Khi bạn gọi DataFrame.to_numpy(), gấu trúc sẽ tìm thấy kiểu NumPy có thể chứa tất cả các kiểu trong DataFrame. Điều này có thể kết thúc object, yêu cầu truyền mọi giá trị cho một đối tượng Python.

Đối với **df**, **DataFrame** tất cả các giá trị dấu phẩy động của chúng tôi , *DataFrame.to_numpy()* nhanh chóng và không yêu cầu sao chép dữ liệu.

```
df.to_numpy()
Out[17]:
array([[ 0.4691, -0.2829, -1.5091, -1.1356],
       [ 1.2121, -0.1732,  0.1192, -1.0442],
       [-0.8618, -2.1046, -0.4949,  1.0718],
       [ 0.7216, -0.7068, -1.0396,  0.2719],
       [-0.425 ,  0.567 ,  0.2762, -1.0874],
       [-0.6737,  0.1136, -1.4784,  0.525 ]])
```

Đối với df2, DataFrame với nhiều loại, DataFrame.to_numpy() là tương đối tốn kém.

```
df2.to_numpy()
Out[18]:
array([[1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo']],
      dtype=object)
```

> Ghi chú DataFrame.to_numpy() không bao gồm các chỉ số hoặc cột nhãn trong đầu ra.

describe() hiển thị tóm tắt thống kê nhanh về dữ liệu của bạn:

```
df.describe()
Out[19]:
              A         B         C         D
count  6.000000  6.000000  6.000000  6.000000
mean   0.073711 -0.431125 -0.687758 -0.233103
std    0.843157  0.922818  0.779887  0.973118
min   -0.861849 -2.104569 -1.509059 -1.135632
25%   -0.611510 -0.600794 -1.368714 -1.076610
50%    0.022070 -0.228039 -0.767252 -0.386188
75%    0.658444  0.041933 -0.034326  0.461706
max    1.212112  0.567020  0.276232  1.071804
```

Chuyển vị dữ liệu:

```
df.T
Out[20]:
   2013-01-01  2013-01-02  2013-01-03  2013-01-04  2013-01-05  2013-01-06
A    0.469112    1.212112   -0.861849    0.721555   -0.424972   -0.673690
B   -0.282863   -0.173215   -2.104569   -0.706771    0.567020    0.113648
C   -1.509059    0.119209   -0.494929   -1.039575    0.276232   -1.478427
D   -1.135632   -1.044236    1.071804    0.271860   -1.087401    0.524988
```

Sắp xếp theo trục tọa độ:

```
df.sort_index(axis=1, ascending=False)
Out[21]:
                   D         C         B         A
2013-01-01 -1.135632 -1.509059 -0.282863  0.469112
2013-01-02 -1.044236  0.119209 -0.173215  1.212112
2013-01-03  1.071804 -0.494929 -2.104569 -0.861849
2013-01-04  0.271860 -1.039575 -0.706771  0.721555
2013-01-05 -1.087401  0.276232  0.567020 -0.424972
2013-01-06  0.524988 -1.478427  0.113648 -0.673690
```

Sắp xếp theo giá trị:

```
df.sort_values(by="B")
Out[22]:
                   A         B         C         D
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
```

Lựa chọn

> Ghi chú Trong khi tiêu chuẩn biểu thức Python / NumPy để lựa chọn và thiết lập trực quan và có ích cho công việc tương tác, cho mã sản xuất, chúng tôi khuyên các phương pháp truy cập dữ liệu gấu trúc được tối ưu hóa, .at, .iat, .locvà .iloc.

Bắt đầu tìm hiểu

Chọn một cột duy nhất, mang lại một Series, tương đương với df.A:

```
df["A"]
Out[23]:
2013-01-01    0.469112
2013-01-02    1.212112
2013-01-03   -0.861849
2013-01-04    0.721555
2013-01-05   -0.424972
2013-01-06   -0.673690
Freq: D, Name: A, dtype: float64
```

Chọn thông qua mảng [], để cắt các hàng.

```
df[0:3]
Out[24]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804

df["20130102":"20130104"]
Out[25]:
                   A         B         C         D
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
```

Lựa chọn theo nhãn

Để có được một mặt cắt bằng cách sử dụng một nhãn:

```
df.loc[dates[0]]
Out[26]:
A    0.469112
B   -0.282863
C   -1.509059
D   -1.135632
Name: 2013-01-01 00:00:00, dtype: float64
```

Chọn trên nhiều trục theo nhãn:

```
df.loc[:, ["A", "B"]]
Out[27]:
                   A         B
2013-01-01  0.469112 -0.282863
2013-01-02  1.212112 -0.173215
2013-01-03 -0.861849 -2.104569
2013-01-04  0.721555 -0.706771
2013-01-05 -0.424972  0.567020
2013-01-06 -0.673690  0.113648
```

Hiển thị cắt nhãn, cả hai điểm cuối đều được bao gồm :

```
df.loc["20130102":"20130104", ["A", "B"]]
Out[28]:
                   A         B
2013-01-02  1.212112 -0.173215
2013-01-03 -0.861849 -2.104569
2013-01-04  0.721555 -0.706771
```

Giảm kích thước của đối tượng trả về:

```
df.loc["20130102", ["A", "B"]]
Out[29]:
A    1.212112
B   -0.173215
Name: 2013-01-02 00:00:00, dtype: float64
```

Để nhận giá trị vô hướng:

```
df.loc[dates[0], "A"]
Out[30]: 0.4691122999071863
```

Để truy cập nhanh vào một phương pháp vô hướng (tương đương với phương pháp trước):

```
df.at[dates[0], "A"]
Out[31]: 0.4691122999071863
```

Lựa chọn theo vị trí

Xem thêm trong Lựa chọn theo vị trí . Chọn qua vị trí của các số nguyên đã chuyển:

```
df.iloc[3]
Out[32]:
A    0.721555
B   -0.706771
C   -1.039575
D    0.271860
Name: 2013-01-04 00:00:00, dtype: float64
```

Theo các lát số nguyên, hoạt động tương tự như NumPy / Python:

```
df.iloc[3:5, 0:2]
Out[33]:
                   A         B
2013-01-04  0.721555 -0.706771
2013-01-05 -0.424972  0.567020
```

Theo danh sách các vị trí vị trí số nguyên, tương tự như kiểu NumPy / Python:

```
df.iloc[[1, 2, 4], [0, 2]]
Out[34]:
                   A         C
2013-01-02  1.212112  0.119209
2013-01-03 -0.861849 -0.494929
2013-01-05 -0.424972  0.276232
```

Để cắt các hàng một cách rõ ràng:

```
df.iloc[1:3, :]
Out[35]:
                   A         B         C         D
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
```

Để cắt các cột một cách rõ ràng:

```
df.iloc[:, 1:3]
Out[36]:
                   B         C
2013-01-01 -0.282863 -1.509059
2013-01-02 -0.173215  0.119209
2013-01-03 -2.104569 -0.494929
2013-01-04 -0.706771 -1.039575
2013-01-05  0.567020  0.276232
2013-01-06  0.113648 -1.478427
```

Để nhận giá trị một cách rõ ràng:

```
df.iloc[1, 1]
Out[37]: -0.17321464905330858
```

Để truy cập nhanh vào một phương pháp vô hướng (tương đương với phương pháp trước):

```
df.iat[1, 1]
Out[38]: -0.17321464905330858
```

Lập chỉ mục Boolean

Sử dụng các giá trị của một cột để chọn dữ liệu.

```
df[df["A"] > 0]
Out[39]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
```

Chọn các giá trị từ DataFrame đáp ứng điều kiện boolean.

```
df[df > 0]
Out[40]:
                   A         B         C         D
2013-01-01  0.469112       NaN       NaN       NaN
2013-01-02  1.212112       NaN  0.119209       NaN
2013-01-03       NaN       NaN       NaN  1.071804
2013-01-04  0.721555       NaN       NaN  0.271860
2013-01-05       NaN  0.567020  0.276232       NaN
2013-01-06       NaN  0.113648       NaN  0.524988
```

Sử dụng phương pháp isin() để lọc:

```
df2 = df.copy()

df2["E"] = ["one", "one", "two", "three", "four", "three"]

df2
Out[43]:
                   A         B         C         D      E
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632    one
2013-01-02  1.212112 -0.173215  0.119209 -1.044236    one
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804    two
2013-01-04  0.721555 -0.706771 -1.039575  0.271860  three
2013-01-05 -0.424972  0.567020  0.276232 -1.087401   four
2013-01-06 -0.673690  0.113648 -1.478427  0.524988  three

df2[df2["E"].isin(["two", "four"])]
Out[44]:
                   A         B         C         D     E
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804   two
2013-01-05 -0.424972  0.567020  0.276232 -1.087401  four
```

Cài đặt

Đặt một cột mới sẽ tự động căn chỉnh dữ liệu theo các chỉ mục.

```
s1 = pd.Series([1, 2, 3, 4, 5, 6], index=pd.date_range("20130102", periods=6))

s1
Out[46]:
2013-01-02    1
2013-01-03    2
2013-01-04    3
2013-01-05    4
2013-01-06    5
2013-01-07    6
Freq: D, dtype: int64

df["F"] = s1
```

Đặt giá trị theo nhãn:

```
df.at[dates[0], "A"] = 0
```

Đặt giá trị theo vị trí:

```
df.iat[0, 1] = 0
```

Cài đặt bằng cách gán với một mảng NumPy:

```
df.loc[:, "D"] = np.array([5] * len(df))
```

Kết quả của các hoạt động thiết lập trước.

```
df
Out[51]:
                   A         B         C  D    F
2013-01-01  0.000000  0.000000 -1.509059  5  NaN
2013-01-02  1.212112 -0.173215  0.119209  5  1.0
2013-01-03 -0.861849 -2.104569 -0.494929  5  2.0
2013-01-04  0.721555 -0.706771 -1.039575  5  3.0
2013-01-05 -0.424972  0.567020  0.276232  5  4.0
2013-01-06 -0.673690  0.113648 -1.478427  5  5.0
```

Điều kiện where được cài đặt.

```
df2 = df.copy()

df2[df2 > 0] = -df2

df2
Out[54]:
                   A         B         C  D    F
2013-01-01  0.000000  0.000000 -1.509059 -5  NaN
2013-01-02 -1.212112 -0.173215 -0.119209 -5 -1.0
2013-01-03 -0.861849 -2.104569 -0.494929 -5 -2.0
2013-01-04 -0.721555 -0.706771 -1.039575 -5 -3.0
2013-01-05 -0.424972 -0.567020 -0.276232 -5 -4.0
2013-01-06 -0.673690 -0.113648 -1.478427 -5 -5.0
```

Thiếu dữ liệu

gấu trúc chủ yếu sử dụng giá trị np.nan để biểu thị dữ liệu bị thiếu. Theo mặc định, nó không được bao gồm trong tính toán. Xem phần Thiếu dữ liệu .

Lập chỉ mục cho phép bạn thay đổi / thêm / xóa chỉ mục trên một trục xác định. Điều này trả về một bản sao của dữ liệu.

```
df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ["E"])

df1.loc[dates[0] : dates[1], "E"] = 1

df1
Out[57]:
                   A         B         C  D    F    E
2013-01-01  0.000000  0.000000 -1.509059  5  NaN  1.0
2013-01-02  1.212112 -0.173215  0.119209  5  1.0  1.0
2013-01-03 -0.861849 -2.104569 -0.494929  5  2.0  NaN
2013-01-04  0.721555 -0.706771 -1.039575  5  3.0  NaN
```

Để loại bỏ bất kỳ hàng nào có dữ liệu bị thiếu.

```
df1.dropna(how="any")
Out[58]:
                   A         B         C  D    F    E
2013-01-02  1.212112 -0.173215  0.119209  5  1.0  1.0
```

Làm đầy dữ liệu còn thiếu.

```
df1.fillna(value=5)
Out[59]:
                   A         B         C  D    F    E
2013-01-01  0.000000  0.000000 -1.509059  5  5.0  1.0
2013-01-02  1.212112 -0.173215  0.119209  5  1.0  1.0
2013-01-03 -0.861849 -2.104569 -0.494929  5  2.0  5.0
2013-01-04  0.721555 -0.706771 -1.039575  5  3.0  5.0
```

Để lấy mặt nạ boolean ở đó các giá trị nan.

```
pd.isna(df1)
Out[60]:
                A      B      C      D      F      E
2013-01-01  False  False  False  False   True  False
2013-01-02  False  False  False  False  False  False
2013-01-03  False  False  False  False  False   True
2013-01-04  False  False  False  False  False   True
```

Một số phép toán

Số liệu thống kê

Các phép toán nói chung loại trừ dữ liệu bị thiếu.

Thực hiện thống kê mô tả:

```
df.mean()
Out[61]:
A   -0.004474
B   -0.383981
C   -0.687758
D    5.000000
F    3.000000
dtype: float64
```

Thao tác tương tự trên trục khác:

```
df.mean(1)
Out[62]:
2013-01-01    0.872735
2013-01-02    1.431621
2013-01-03    0.707731
2013-01-04    1.395042
2013-01-05    1.883656
2013-01-06    1.592306
Freq: D, dtype: float64
```

Hoạt động với các đối tượng có kích thước khác nhau và cần sự liên kết. Ngoài ra, gấu trúc tự động phát sóng dọc theo kích thước được chỉ định.

```
In [63]: s = pd.Series([1, 3, 5, np.nan, 6, 8], index=dates).shift(2)
In [64]: s
Out[64]:
2013-01-01 NaN
2013-01-02 NaN
2013-01-03 1.0
2013-01-04 3.0
2013-01-05 5.0
2013-01-06 NaN

Freq: D, dtype: float64
In [65]: df.sub(s, axis="index")
Out[65]:
A B C D F
2013-01-01 NaN NaN NaN NaN NaN
2013-01-02 NaN NaN NaN NaN NaN
2013-01-03 -1.861849 -3.104569 -1.494929 4.0 1.0
2013-01-04 -2.278445 -3.706771 -4.039575 2.0 0.0
2013-01-05 -5.424972 -4.432980 -4.723768 0.0 -1.0
2013-01-06 NaN NaN NaN NaN NaN
```

Áp dụng

Áp dụng các chức năng cho dữ liệu:

```
In [66]: df.apply(np.cumsum)
Out[66]:
A B C D F
2013-01-01 0.000000 0.000000 -1.509059 5 NaN
2013-01-02 1.212112 -0.173215 -1.389850 10 1.0
2013-01-03 0.350263 -2.277784 -1.884779 15 3.0
2013-01-04 1.071818 -2.984555 -2.924354 20 6.0
2013-01-05 0.646846 -2.417535 -2.648122 25 10.0
2013-01-06 -0.026844 -2.303886 -4.126549 30 15.0
In [67]: df.apply(lambda x: x.max() - x.min())
Out[67]:
A 2.073961
B 2.671590
C 1.785291
D 0.000000
F 4.000000
dtype: float64
```

Lập biểu đồ

```
In [68]: s = pd.Series(np.random.randint(0, 7, size=10))
In [69]: s
Out[69]:
0 4
1 2
2 1
3 2
4 6
5 4
6 4
7 6
8 4
9 4
dtype: int64
In [70]: s.value_counts()
Out[70]:
4 5
2 2
6 2
1 1
dtype: int64
```

Phương thức chuỗi

Chuỗi được trang bị một tập hợp các phương pháp xử lý chuỗi trong str thuộc tính giúp dễ dàng thao tác trên từng phần tử của mảng, như trong đoạn mã bên dưới. Lưu ý rằng đối sánh mẫu strnói chung sử dụng biểu thức chính quy theo mặc định (và trong một số trường hợp luôn sử dụng chúng).

```
In [71]: s = pd.Series(["A", "B", "C", "Aaba", "Baca", np.nan, "CABA", "dog", "cat"])
In [72]: s.str.lower()
Out[72]:
0 a
1 b
2 c
3 aaba
4 baca
5 NaN
6 caba
7 dog
8 cat
dtype: object
```

Kết hợp dữ liệu

Concat pandas cung cấp các phương tiện khác nhau để dễ dàng kết hợp các đối tượng Series và DataFrame với nhau với nhiều loại logic tập hợp khác nhau cho các chỉ mục và hàm đại số quan hệ trong trường hợp các phép toán kiểu nối / kết hợp.

Kết nối các đối tượng gấu trúc với concat():

```
In [73]: df = pd.DataFrame(np.random.randn(10, 4))
In [74]: df
Out[74]:
0 1 2 3
0 -0.548702 1.467327 -1.015962 -0.483075
1 1.637550 -1.217659 -0.291519 -1.745505
2 -0.263952 0.991460 -0.919069 0.266046
3 -0.709661 1.669052 1.037882 -1.705775
4 -0.919854 -0.042379 1.247642 -0.009920
5 0.290213 0.495767 0.362949 1.548106
6 -1.131345 -0.089329 0.337863 -0.945867
7 -0.932132 1.956030 0.017587 -0.016692
8 -0.575247 0.254161 -1.143704 0.215897
9 1.193555 -0.077118 -0.408530 -0.862495
# break it into pieces
In [75]: pieces = [df[:3], df[3:7], df[7:]]
In [76]: pd.concat(pieces)
Out[76]:
0 1 2 3
0 -0.548702 1.467327 -1.015962 -0.483075
1 1.637550 -1.217659 -0.291519 -1.745505
2 -0.263952 0.991460 -0.919069 0.266046
3 -0.709661 1.669052 1.037882 -1.705775
4 -0.919854 -0.042379 1.247642 -0.009920
5 0.290213 0.495767 0.362949 1.548106
6 -1.131345 -0.089329 0.337863 -0.945867
7 -0.932132 1.956030 0.017587 -0.016692
8 -0.575247 0.254161 -1.143704 0.215897
9 1.193555 -0.077118 -0.408530 -0.862495
```

> Ghi chú Thêm một cột vào a DataFrame là tương đối nhanh. Tuy nhiên, việc thêm một hàng yêu cầu một bản sao và có thể tốn kém. Chúng tôi khuyên bạn nên chuyển một danh sách các bản ghi được tạo sẵn cho phương thức DataFramekhởi tạo thay vì tạo một DataFramebằng cách nối các bản ghi vào nó một cách lặp đi lặp lại. Xem Thêm vào khung dữ liệu để biết thêm.