# Creating an empty Pandas DataFrame, then filling it?

324

104

I'm starting from the pandas DataFrame docs here: http://pandas.pydata.org/pandas-docs/stable/dsintro.html

I'd like to iteratively fill the DataFrame with values in a time series kind of calculation. So basically, I'd like to initialize the DataFrame with columns A, B and timestamp rows, all 0 or all NaN.

I'd then add initial values and go over this data calculating the new row from the row before, say `row[A][t] = row[A][t-1]+1` or so.

I'm currently using the code as below, but I feel it's kind of ugly and there must be a way to do this with a DataFrame directly, or just a better way in general. Note: I'm using Python 2.7.

```
import datetime as dt
import pandas as pd
import scipy as s

if __name__ == '__main__':
    base = dt.datetime.today().date()
    dates = [ base - dt.timedelta(days=x) for x in range(0,10) ]
    dates.sort()

    valdict = {}
    symbols = ['A','B', 'C']
    for symb in symbols:
        valdict[symb] = pd.Series( s.zeros( len(dates)), dates )

    for thedate in dates:
        if thedate > dates[0]:
            for symb in valdict:
                valdict[symb][thedate] = 1+valdict[symb][thedate -
dt.timedelta(days=1)]

    print valdict
```

python dataframe pandas

edited Feb 25 at 17:29
Daniel Holmes
462 5 14

asked Dec 9 '12 at 2:50
Matthias Kauer
1,901 4 12 15

## 4 Answers

255

Here's a couple of suggestions:

Use `date_range` for the index:

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

✓

```
todays_date = datetime.datetime.now().date()
index = pd.date_range(todays_date-datetime.timedelta(10), periods=10, freq='D')

columns = ['A','B', 'C']
```

*Note: we could create an empty DataFrame (with `NaN` s) simply by writing:*

```
df_ = pd.DataFrame(index=index, columns=columns)
df_ = df_.fillna(0) # with 0s rather than NaNs
```

To do these type of calculations for the data, use a numpy array:

```
data = np.array([np.arange(10)]*3).T
```

Hence we can create the DataFrame:

```
In [10]: df = pd.DataFrame(data, index=index, columns=columns)

In [11]: df
Out[11]:
            A  B  C
2012-11-29  0  0  0
2012-11-30  1  1  1
2012-12-01  2  2  2
2012-12-02  3  3  3
2012-12-03  4  4  4
2012-12-04  5  5  5
2012-12-05  6  6  6
2012-12-06  7  7  7
2012-12-07  8  8  8
2012-12-08  9  9  9
```

edited Jan 31 '17 at 14:26
Ninjakannon
2,771 4 31 53

answered Dec 9 '12 at 9:40
Andy Hayden
201k 56 452 443

2 pd.date_range() does not work for me. I tried with DateRange (from eclipse's autocompletion), but that works with strings as date format, right? The overall approach works though (I changed index to something else). – Matthias Kauer Dec 15 '12 at 8:42

2 date_range is a factory function for creating datetime indexes and was a new feature in 0.8.0, I would definitely recommend upgrading to the latest stable release (0.9.1) there are many bug fixes and new features. :) – Andy Hayden Dec 15 '12 at 9:52

25 In my experiences, creating a data frame of the necessary size filled with NaNs, and then filling up with values is much-much slower than creating a data frame with `index x 0` dimensions ( `columns = []` ), and attaching one column in each turn of a loop. I mean `df[col_name] = pandas.Series([...])` in a loop iterating through column names. In the former case, not only the memory allocation takes time, but replacing NaNs with new values seems extremely slow. – deeenes Mar 3 '15 at 16:33

5 @deeenes definitely. this answer should probably make that clearer - you very rarely (if ever) want to do create an empty Dataframe (of NaNs). – Andy Hayden Mar 3 '15 at 17:33

1 As per this answer stackoverflow.com/a/30267881/2302569 You need to assign the result of fillna, or pass param inplace=True – JayJay Jan 2 '17 at 20:09

121

***If you simply want to create an empty data frame and fill it with some incoming data frames later, try this:***

In this example I am using this pandas doc to create a new data frame and then using append to write to the newDF with data from oldDF.

**Have a look at this**

```
newDF = pd.DataFrame() #creates a new dataframe that's empty
newDF = newDF.append(oldDF, ignore_index = True) # ignoring index is optional
# try printing some data from newDF
print newDF.head() #again optional
```

- if I have to keep appending new data into this newDF from more than one oldDFs, I just use a for loop to iterate over pandas.DataFrame.append()

edited May 23 '17 at 15:56 answered Jan 8 '17 at 4:18
geekidharsh
1,767 1 11 16

13 Please note that `append` (and similarly `concat` ) copies the full dataset to a new object every time, hence, iterating and appending can and will cause a major performance hit. for more info refer to: pandas.pydata.org/pandas-docs/stable/merging.html – MoustafaAAtta Sep 18 '17 at 12:21

2 @MoustafaAAtta What are the alternatives to append iteratively data to the dataframe ? – MysteryGuy Aug 13 '18 at 11:24

2 @MoustafaAAtta Is Fred answer in this post : stackoverflow.com/questions/10715965/… better on this point of view ? – MysteryGuy Aug 13 '18 at 11:29

90

***If you want to have your column names in place from the start, use this approach:***

```
import pandas as pd

col_names =  ['A', 'B', 'C']
my_df  = pd.DataFrame(columns = col_names)
my_df
```

***If you want to add a record to the dataframe it would be better to use:***

```
my_df.loc[len(my_df)] = [2, 4, 5]
```

You also might want to pass a dictionary:

```
my_dic = {'A':2, 'B':4, 'C':5}
my_df.loc[len(my_df)] = my_dic
```

***However if you want to add another dataframe to my_df do as follows:***

```
col_names =  ['A', 'B', 'C']
my_df2  = pd.DataFrame(columns = col_names)
my_df = my_df.append(my_df2)
```

***If you are adding rows inside a loop consider performance issues:***
For around the first 1000 records "my_df.loc" performance is better, but it gradually becomes slower by increasing the number of records in the loop.

***If you plan to do thins inside a big loop (say 10M records or so):***
You are better off using a mixture of these two; fill a dataframe with iloc until the size gets around 1000, then append it to the original dataframe, and empty the temp dataframe. This would boost your performance by around 10 times.

edited May 21 at 21:05 — DanTan **188** 4 13

answered Apr 23 '18 at 5:29 — Afshin Amiri **1,624** 6 16

1 This is absolutely perfect! Thank you! – amc Feb 3 at 22:36

-1

Assume a dataframe with 19 rows

```
index=range(0,19)
index

columns=['A']
test = pd.DataFrame(index=index, columns=columns)
```

Keeping Column A as a constant

```
test['A']=10
```

Keeping column b as a variable given by a loop

```
for x in range(0,19):
    test.loc[[x], 'b'] = pd.Series([x], index = [x])
```

You can replace the first x in `pd.Series([x], index = [x])` with any value

edited Mar 16 at 7:35 — AkshayNevrekar **6,821** 10 22 43

answered Aug 29 '18 at 11:06 — Ajay Ohri **2,105** 2 21 53

**protected** by cs95 May 17 '18 at 7:29

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).