Combine two columns of text in pandas dataframe

Asked 9 years, 1 month ago Modified 6 months ago Viewed 1.6m times

888

I have a 20 x 4000 dataframe in Python using pandas. Two of these columns are named `Year` and `quarter`. I'd like to create a variable called `period` that makes `Year = 2000` and `quarter= q2` into `2000q2`.

Can anyone help with that?

python pandas dataframe

Share Edit Follow

edited Aug 13, 2020 at 23:27
desertnaut
55.3k 21 132 163

asked Oct 15, 2013 at 9:42
user2866103
9,357 6 15 14

Searchers: here's a similar question with more answers – MEHOV Oct 18 at 19:40

20 Answers

Sorted by: Highest score (default)

1100

If both columns are strings, you can concatenate them directly:

```
df["period"] = df["Year"] + df["quarter"]
```

If one (or both) of the columns are not string typed, you should convert it (them) first,

```
df["period"] = df["Year"].astype(str) + df["quarter"]
```

Beware of NaNs when doing this!

If you need to join multiple string columns, you can use `agg`:

```
df['period'] = df[['Year', 'quarter', ...]].agg('-'.join, axis=1)
```

Where "-" is the separator.

Share Edit Follow

edited Nov 11, 2021 at 11:20
iacob
16.4k 5 73 103

answered Oct 15, 2013 at 10:09
silvado
16.3k 2 30 46

23 Is it possible to add multiple columns together without typing out all the columns? Let's say `add(dataframe.iloc[:, 0:10])` for example? – Heisenberg May 9, 2015 at 19:15

7 @Heisenberg That should be possible with the Python builtin `sum`. – silvado May 11, 2015 at 11:06

8 @silvado could you please make an example for adding multiple columns? Thank you – c1c1c1 Oct 25, 2016 at 16:45

15 Be careful, you need to apply map(str) to all columns that are not string in the first place. if quarter was a number you would do `dataframe["period"] = dataframe["Year"].map(str) + dataframe["quarter"].map(str)` map is just applying string conversion to all entries. – Ozgur Ozturk Feb 1, 2017 at 21:17

25 This solution can create problems iy you have nan values, e careful – user2270655 Dec 27, 2017 at 17:14

389

Small data-sets (< 150rows)

```
[''.join(i) for i in zip(df["Year"].map(str),df["quarter"])]
```

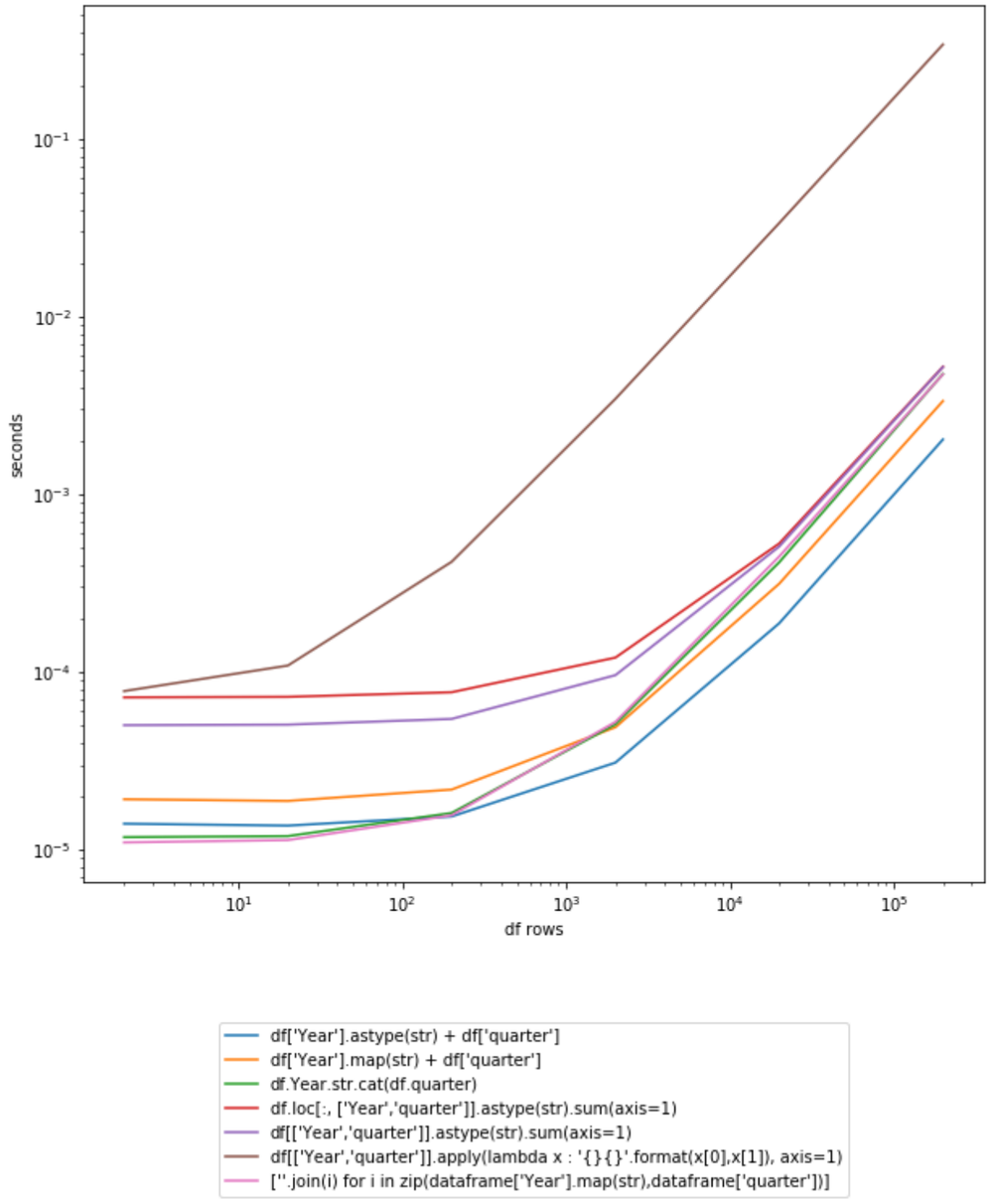
or slightly slower but more compact:

```
df.Year.str.cat(df.quarter)
```

Larger data sets (> 150rows)

```
df['Year'].astype(str) + df['quarter']
```

UPDATE: Timing graph Pandas 0.23.4



Let's test it on 200K rows DF:

```
In [250]: df
Out[250]:
   Year quarter
0  2014      q1
1  2015      q2

In [251]: df = pd.concat([df] * 10**5)

In [252]: df.shape
Out[252]: (200000, 2)
```

UPDATE: new timings using Pandas 0.19.0

Timing without CPU/GPU optimization (sorted from fastest to slowest):

```
In [107]: %timeit df['Year'].astype(str) + df['quarter']
10 loops, best of 3: 131 ms per loop

In [106]: %timeit df['Year'].map(str) + df['quarter']
10 loops, best of 3: 161 ms per loop

In [108]: %timeit df.Year.str.cat(df.quarter)
10 loops, best of 3: 189 ms per loop

In [109]: %timeit df.loc[:, ['Year','quarter']].astype(str).sum(axis=1)
1 loop, best of 3: 567 ms per loop

In [110]: %timeit df[['Year','quarter']].astype(str).sum(axis=1)
1 loop, best of 3: 584 ms per loop

In [111]: %timeit df[['Year','quarter']].apply(lambda x : '{}
{}'.format(x[0],x[1]), axis=1)
1 loop, best of 3: 24.7 s per loop
```

Timing using CPU/GPU optimization:

```
In [113]: %timeit df['Year'].astype(str) + df['quarter']
10 loops, best of 3: 53.3 ms per loop

In [114]: %timeit df['Year'].map(str) + df['quarter']
10 loops, best of 3: 65.5 ms per loop

In [115]: %timeit df.Year.str.cat(df.quarter)
10 loops, best of 3: 79.9 ms per loop

In [116]: %timeit df.loc[:, ['Year','quarter']].astype(str).sum(axis=1)
1 loop, best of 3: 230 ms per loop

In [117]: %timeit df[['Year','quarter']].astype(str).sum(axis=1)
1 loop, best of 3: 230 ms per loop

In [118]: %timeit df[['Year','quarter']].apply(lambda x : '{}
{}'.format(x[0],x[1]), axis=1)
1 loop, best of 3: 9.38 s per loop
```

Answer contribution by @anton-vbr

Share Edit Follow

edited May 30 at 17:37
Sunderam Dubey
1

answered Apr 28, 2016 at 10:02
MaxU - stop genocide of UA
199k 36 364 401

What difference between 261 and 264 in your timing? – Anton Protopopov May 21, 2016 at 19:57

@AntonProtopopov apparently 100ms out of nowhere :) – Dennis Golomazov Oct 10, 2016 at 17:30

@AntonProtopopov, i guess it's a mixture of two timings - one used CPU/GPU optimization, another one didn't. I've updated my answer and put both timing sets there... – MaxU - stop genocide of UA Oct 10, 2016 at 17:45

This use of .sum() fails If all columns look like they could be integers (ie are string forms of integers). Instead, it seems pandas converts them back to numeric before summing! – CPBL May 25, 2017 at 13:06

1 @MaxU How did you go about the CPU/GPU optimization? Is that just a more powerful computer or is it something you did with code? – user3374113 Jul 7, 2017 at 11:30

318

```
df = pd.DataFrame({'Year': ['2014', '2015'], 'quarter': ['q1', 'q2']})
df['period'] = df[['Year', 'quarter']].apply(lambda x: ''.join(x), axis=1)
```

Yields this dataframe

```
   Year quarter  period
0  2014      q1  2014q1
1  2015      q2  2015q2
```

This method generalizes to an arbitrary number of string columns by replacing `df[['Year', 'quarter']]` with any column slice of your dataframe, e.g. `df.iloc[:,0:2].apply(lambda x: ''.join(x), axis=1)`.

You can check more information about apply() method here

Share Edit Follow

edited Jan 8, 2018 at 16:21
kepy97
910 10 12

answered Sep 11, 2015 at 17:36
Russ
3,546 1 12 14

30 `lambda x: ''.join(x)` is just `''.join`, no? – DSM Sep 19, 2016 at 11:54

6 @OzgurOzturk: the point is that the lambda part of the `lambda x: ''.join(x)` construction doesn't do anything; it's like using `lambda x: sum(x)` instead of just `sum`. – DSM Feb 1, 2017 at 21:07

5 Confirmed same result when using `''.join`, i.e.: `df['period'] = df[['Year', 'quarter']].apply(''.join, axis=1)`. – Max Ghenis Oct 10, 2017 at 5:30

1 @Archie `join` takes only `str` instances in an iterable. Use a `map` to convert them all into `str` and then use `join`. – John Strood Mar 27, 2018 at 12:51

23 '-'.join(x.map(str)) – Manjul Sep 3, 2018 at 8:23

194

The method `cat()` of the `.str` accessor works really well for this:

```
>>> import pandas as pd
>>> df = pd.DataFrame([["2014", "q1"],
...                    ["2015", "q3"]],
...                   columns=('Year', 'Quarter'))
>>> print(df)
   Year Quarter
0  2014      q1
1  2015      q3
>>> df['Period'] = df.Year.str.cat(df.Quarter)
>>> print(df)
   Year Quarter  Period
0  2014      q1  2014q1
1  2015      q3  2015q3
```

`cat()` even allows you to add a separator so, for example, suppose you only have integers for year and period, you can do this:

```
>>> import pandas as pd
>>> df = pd.DataFrame([[2014, 1],
...                    [2015, 3]],
...                   columns=('Year', 'Quarter'))
>>> print(df)
   Year Quarter
0  2014       1
1  2015       3
>>> df['Period'] = df.Year.astype(str).str.cat(df.Quarter.astype(str), sep='q')
>>> print(df)
```

```
   Year Quarter  Period
0  2014       1  2014q1
1  2015       3  2015q3
```

Joining multiple columns is just a matter of passing either a list of series or a dataframe containing all but the first column as a parameter to `str.cat()` invoked on the first column (Series):

```
>>> df = pd.DataFrame(
...     [['USA', 'Nevada', 'Las Vegas'],
...      ['Brazil', 'Pernambuco', 'Recife']],
...     columns=['Country', 'State', 'City'],
... )
>>> df['AllTogether'] = df['Country'].str.cat(df[['State', 'City']], sep=' - ')
>>> print(df)
  Country       State       City                   AllTogether
0     USA      Nevada  Las Vegas      USA - Nevada - Las Vegas
1  Brazil  Pernambuco     Recife  Brazil - Pernambuco - Recife
```

Do note that if your pandas dataframe/series has null values, you need to include the parameter na_rep to replace the NaN values with a string, otherwise the combined column will default to NaN.

Share Edit Follow

edited Sep 23, 2018 at 8:49
G. Sliepen
7,338 1 17 31

answered Mar 7, 2016 at 18:04
LeoRochael
13.4k 5 27 38

19 This seems way better (maybe more efficient, too) than `lambda` or `map` ; also it just reads most cleanly. – dwanderson May 22, 2016 at 20:31

1 @ZakS, by passing the remaining columns as a dataframe instead of a series as the first parameter to `str.cat()` . I'll amend the answer – LeoRochael Jul 23, 2018 at 21:42

Which version of pandas are you using? I get ValueError: Did you mean to supply a `sep` keyword? in pandas-0.23.4. Thanks! – Qinqing Liu Dec 5, 2018 at 20:56

@QinqingLiu, I retested these with pandas-0.23.4 and they seem work. The `sep` parameter is only necessary if you intend to separate the parts of the concatenated string. If you get an error, please show us your failing example. – LeoRochael Dec 10, 2018 at 19:34

1 @arun-menon: I don't see why not. In the last example above you could do `.str.cat(df[['State', 'City']], sep ='\n')` , for example. I haven't tested it yet, though. – LeoRochael Jun 21, 2021 at 12:08

43

Use of a lamba function this time with string.format().

```
import pandas as pd
df = pd.DataFrame({'Year': ['2014', '2015'], 'Quarter': ['q1', 'q2']})
print df
df['YearQuarter'] = df[['Year','Quarter']].apply(lambda x : '{}
{}'.format(x[0],x[1]), axis=1)
print df

  Quarter  Year
0      q1  2014
1      q2  2015
  Quarter  Year YearQuarter
0      q1  2014      2014q1
1      q2  2015      2015q2
```

This allows you to work with non-strings and reformat values as needed.

```
import pandas as pd
df = pd.DataFrame({'Year': ['2014', '2015'], 'Quarter': [1, 2]})
```

```
print df.dtypes
print df

df['YearQuarter'] = df[['Year','Quarter']].apply(lambda x :
'{}q{}'.format(x[0],x[1]), axis=1)
print df

Quarter     int64
Year       object
dtype: object
   Quarter  Year
0        1  2014
1        2  2015
   Quarter  Year YearQuarter
0        1  2014      2014q1
1        2  2015      2015q2
```

Share Edit Follow

edited Mar 16, 2016 at 16:57

answered Mar 16, 2016 at 16:43

Bill Gale
1,192 13 14

4 Much quicker: .apply(".join(x), axis=1) – Minions Jul 8, 2019 at 10:31

This solution worked great for my needs since I had to do some formatting. `df_game['formatted_game_time'] = df_game[['wday', 'month', 'day', 'year', 'time']].apply(lambda x: '{}, {}/{}/{} @ {}'.format(x[0], x[1], x[2], x[3], x[4]), axis=1)` – Dan Nov 26 at 17:07

23

generalising to multiple columns, why not:

```
columns = ['whatever', 'columns', 'you', 'choose']
df['period'] = df[columns].astype(str).sum(axis=1)
```

Share Edit Follow

answered Jul 30, 2019 at 10:38

geher
475 1 6 14

6 Looks cool but what if I want to add a delimiter between the strings, like '-'? – Odisseo Oct 2, 2019 at 17:55

@Odisseo maybe create a delimiter column? – Dd H Sep 20, 2021 at 8:15

18

You can use lambda:

```
combine_lambda = lambda x: '{}{}'.format(x.Year, x.quarter)
```

And then use it with creating the new column:

```
df['period'] = df.apply(combine_lambda, axis = 1)
```

Share Edit Follow

edited Jun 1, 2021 at 12:12

buhtz
9,464 14 66 132

answered Feb 28, 2021 at 16:25

Pobaranchuk
789 8 12

Let us suppose your `dataframe` is `df` with columns **`Year`** and **`Quarter`** .

15

```
import pandas as pd
df = pd.DataFrame({'Quarter':'q1 q2 q3 q4'.split(), 'Year':'2000'})
```

Suppose we want to see the dataframe;

```
df
>>>  Quarter    Year
   0    q1      2000
   1    q2      2000
   2    q3      2000
   3    q4      2000
```

Finally, concatenate the **`Year`** and the **`Quarter`** as follows.

```
df['Period'] = df['Year'] + ' ' + df['Quarter']
```

You can now `print` **`df`** to see the resulting dataframe.

```
df
>>>  Quarter    Year    Period
     0   q1      2000    2000 q1
     1   q2      2000    2000 q2
     2   q3      2000    2000 q3
     3   q4      2000    2000 q4
```

If you do not want the space between the year and quarter, simply remove it by doing;

```
df['Period'] = df['Year'] + df['Quarter']
```

Share Edit Follow

edited Jan 24, 2019 at 9:38
cs95
355k 88 655 707

answered Jul 22, 2018 at 5:20
Samuel Nde
2,472 2 23 23

3 Specified as strings `df['Period'] = df['Year'].map(str) + df['Quarter'].map(str)` – Stuber Aug 7, 2018 at 18:58

I'm getting `TypeError: Series cannot perform the operation +` when I run either `df2['filename'] = df2['job_number'] + '.' + df2['task_number']` or `df2['filename'] = df2['job_number'].map(str) + '.' + df2['task_number'].map(str)` . – Karl Baker Mar 3, 2019 at 6:43

1 However, `df2['filename'] = df2['job_number'].astype(str) + '.' + df2['task_number'].astype(str)` did work. – Karl Baker Mar 3, 2019 at 6:51

@KarlBaker, I think you did not have strings in your input. But I am glad you figured that out. If you look at the example `dataframe` that I created above, you will see that all the columns are `string` s. – Samuel Nde Mar 3, 2019 at 17:31

What exactly is the point of this solution, since it's identical to the top answer? – AMC Mar 18, 2020 at 1:22

14

Although the @silvado answer is good if you change `df.map(str)` to `df.astype(str)` it will be faster:

```
import pandas as pd
df = pd.DataFrame({'Year': ['2014', '2015'], 'quarter': ['q1', 'q2']})

In [131]: %timeit df["Year"].map(str)
10000 loops, best of 3: 132 us per loop
```

```
In [132]: %timeit df["Year"].astype(str)
10000 loops, best of 3: 82.2 us per loop
```

Share Edit Follow

answered Nov 25, 2015 at 10:25
Anton Protopopov
28.8k 11 84 91

12

Here is an implementation that I find very versatile:

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

In [2]: df = pd.DataFrame([[0, 'the', 'quick', 'brown'],
   ...:                    [1, 'fox', 'jumps', 'over'],
   ...:                    [2, 'the', 'lazy', 'dog']],
   ...:                   columns=['c0', 'c1', 'c2', 'c3'])

In [3]: def str_join(df, sep, *cols):
   ...:     from functools import reduce
   ...:     return reduce(lambda x, y: x.astype(str).str.cat(y.astype(str),
sep=sep),
   ...:                   [df[col] for col in cols])
   ...:

In [4]: df['cat'] = str_join(df, '-', 'c0', 'c1', 'c2', 'c3')

In [5]: df
Out[5]:
   c0   c1     c2     c3                cat
0   0  the  quick  brown  0-the-quick-brown
1   1  fox  jumps   over   1-fox-jumps-over
2   2  the   lazy    dog     2-the-lazy-dog
```

Share Edit Follow

answered Apr 3, 2017 at 17:05
Pedro M Duarte
25.6k 7 43 43

FYI: This method works great with Python 3, but gives me trouble in Python 2. – Alex P. Miller Jul 31, 2017 at 19:40

11

more efficient is

```
def concat_df_str1(df):
    """ run time: 1.3416s """
    return pd.Series([''.join(row.astype(str)) for row in df.values],
index=df.index)
```

and here is a time test:

```
import numpy as np
import pandas as pd

from time import time


def concat_df_str1(df):
    """ run time: 1.3416s """
    return pd.Series([''.join(row.astype(str)) for row in df.values],
index=df.index)
```

```
def concat_df_str2(df):
    """ run time: 5.2758s """
    return df.astype(str).sum(axis=1)


def concat_df_str3(df):
    """ run time: 5.0076s """
    df = df.astype(str)
    return df[0] + df[1] + df[2] + df[3] + df[4] + \
           df[5] + df[6] + df[7] + df[8] + df[9]


def concat_df_str4(df):
    """ run time: 7.8624s """
    return df.astype(str).apply(lambda x: ''.join(x), axis=1)


def main():
    df = pd.DataFrame(np.zeros(1000000).reshape(100000, 10))
    df = df.astype(int)

    time1 = time()
    df_en = concat_df_str4(df)
    print('run time: %.4fs' % (time() - time1))
    print(df_en.head(10))


if __name__ == '__main__':
    main()
```

final, when `sum` (concat_df_str2) is used, the result is not simply concat, it will trans to integer.

Share Edit Follow

answered Jan 9, 2018 at 2:13

Colin Wang
701 8 13

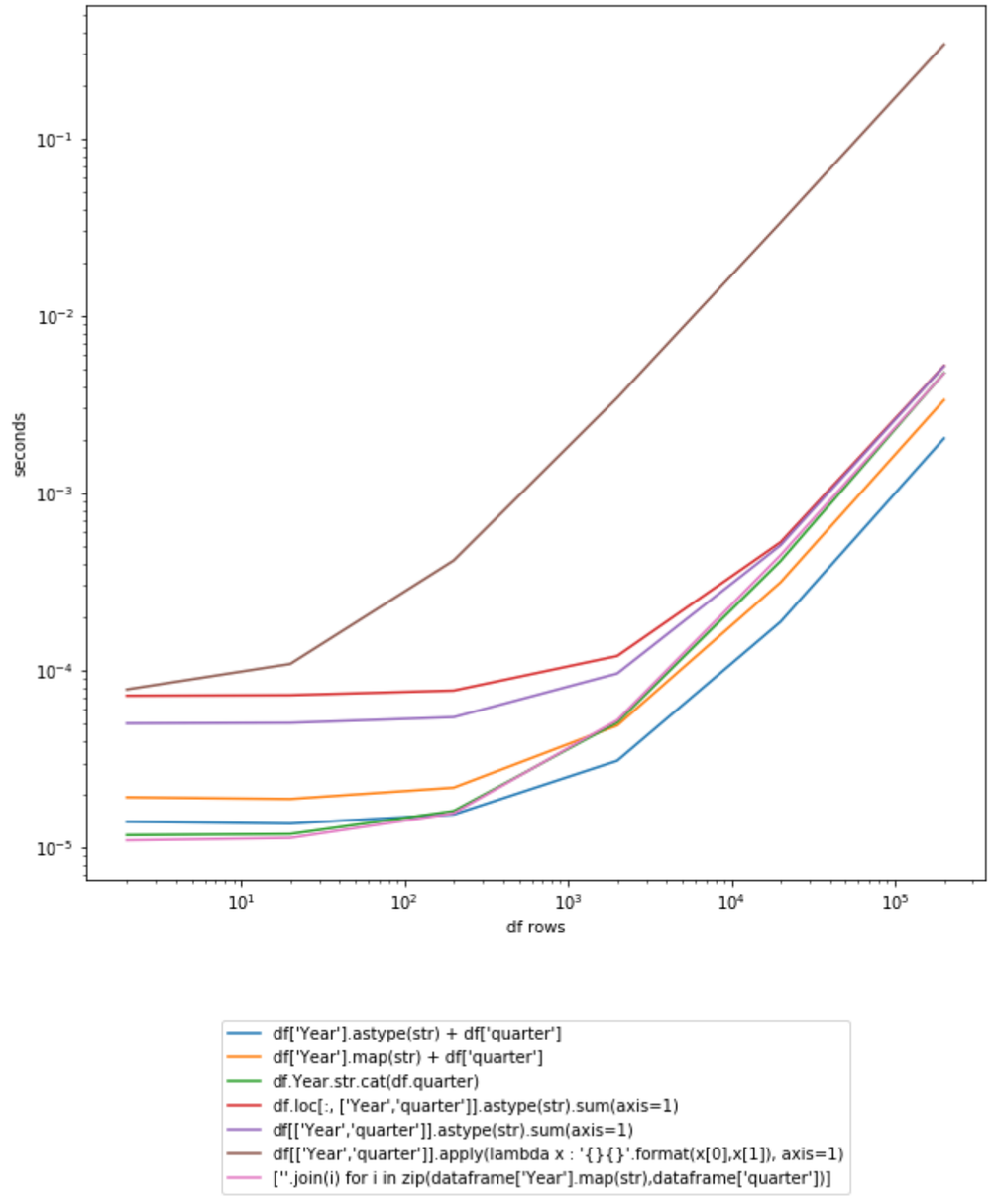
1 +1 Neat solution, this also allows us to specify the columns: e.g. `df.values[:, 0:3]` or `df.values[:, [0,2]]` . – Snow bunting Feb 9, 2018 at 9:51

7

Using `zip` could be even quicker:

```
df["period"] = [''.join(i) for i in zip(df["Year"].map(str),df["quarter"])]
```

Graph:




```
import pandas as pd
import numpy as np
import timeit
import matplotlib.pyplot as plt
from collections import defaultdict

df = pd.DataFrame({'Year': ['2014', '2015'], 'quarter': ['q1', 'q2']})

myfuncs = {
"df['Year'].astype(str) + df['quarter']":
    lambda: df['Year'].astype(str) + df['quarter'],
"df['Year'].map(str) + df['quarter']":
    lambda: df['Year'].map(str) + df['quarter'],
"df.Year.str.cat(df.quarter)":
    lambda: df.Year.str.cat(df.quarter),
"df.loc[:, ['Year','quarter']].astype(str).sum(axis=1)":
    lambda: df.loc[:, ['Year','quarter']].astype(str).sum(axis=1),
"df[['Year','quarter']].astype(str).sum(axis=1)":
    lambda: df[['Year','quarter']].astype(str).sum(axis=1),
    "df[['Year','quarter']].apply(lambda x : '{}{}'.format(x[0],x[1]),
axis=1)":
```

```
    lambda: df[['Year','quarter']].apply(lambda x : '{}{}'.format(x[0],x[1]),
axis=1),
    "['''.join(i) for i in
zip(dataframe['Year'].map(str),dataframe['quarter'])]":
    lambda: ['''.join(i) for i in zip(df["Year"].map(str),df["quarter"])]
}

d = defaultdict(dict)
step = 10
cont = True
while cont:
    lendf = len(df); print(lendf)
    for k,v in myfuncs.items():
        iters = 1
        t = 0
        while t < 0.2:
            ts = timeit.repeat(v, number=iters, repeat=3)
            t = min(ts)
            iters *= 10
        d[k][lendf] = t/iters
        if t > 2: cont = False
    df = pd.concat([df]*step)

pd.DataFrame(d).plot().legend(loc='upper center', bbox_to_anchor=(0.5, -0.15))
plt.yscale('log'); plt.xscale('log'); plt.ylabel('seconds'); plt.xlabel('df
rows')
plt.show()
```

Share Edit Follow

edited Feb 20, 2019 at 17:07

answered May 13, 2018 at 14:33

Anton vBR
17.6k 5 38 46

7

This solution uses an intermediate step **compressing two columns of the DataFrame to a single column containing a list** of the values. This works not only for strings but for all kind of column-dtypes

```
import pandas as pd
df = pd.DataFrame({'Year': ['2014', '2015'], 'quarter': ['q1', 'q2']})
df['list']=df[['Year','quarter']].values.tolist()
df['period']=df['list'].apply(''.join)
print(df)
```

Result:

```
   Year quarter        list  period
0  2014      q1  [2014, q1]  2014q1
1  2015      q2  [2015, q2]  2015q2
```

Share Edit Follow

answered Mar 15, 2019 at 16:37

Markus Dutschke
8,115 2 54 53

looks like other dtypes won't work. I got a TypeError: sequence item 1: expected str instance, float found – Prometheus Apr 10, 2019 at 9:08

apply first a cast to string. The join operation works only for strings – Markus Dutschke Apr 10, 2019 at 10:58

This solution won't work to combine two columns with different dtype, see my answer for the correct solution for such case. – Good Will May 16, 2019 at 13:21

Instead of `.apply(''.join)` why not use `.str.join('')` ? – Bill May 28, 2021 at 0:45

6

Here is my summary of the above solutions to concatenate / combine two columns with int and str value into a new column, using a separator between the values of columns. Three solutions work for this purpose.

```
# be cautious about the separator, some symbols may cause "SyntaxError: EOL
while scanning string literal".
# e.g. ";;" as separator would raise the SyntaxError

separator = "&&"

# pd.Series.str.cat() method does not work to concatenate / combine two columns
with int value and str value. This would raise "AttributeError: Can only use
.cat accessor with a 'category' dtype"

df["period"] = df["Year"].map(str) + separator + df["quarter"]
df["period"] = df[['Year','quarter']].apply(lambda x : '{} &&
{}'.format(x[0],x[1]), axis=1)
df["period"] = df.apply(lambda x: f'{x["Year"]} && {x["quarter"]}', axis=1)
```

Share Edit Follow

answered May 16, 2019 at 13:19

Good Will
1,141 14 10

5

my take....

```
listofcols = ['col1','col2','col3']
df['combined_cols'] = ''

for column in listofcols:
    df['combined_cols'] = df['combined_cols'] + ' ' + df[column]
'''
```

Share Edit Follow

answered Aug 18, 2020 at 4:13

leo
323 4 12

5 You should add an explanation to this code snippet. Adding only code answers encourages people to use code they don't understand and doesn't help them learn.
– annedroiid Aug 18, 2020 at 10:10

2

As many have mentioned previously, you must convert each column to string and then use the plus operator to combine two string columns. You can get a large performance improvement by using NumPy.

```
%timeit df['Year'].values.astype(str) + df.quarter
71.1 ms ± 3.76 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

%timeit df['Year'].astype(str) + df['quarter']
565 ms ± 22.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Share Edit Follow

answered Oct 25, 2017 at 3:21

Ted Petrou
56k 19 124 129

I'd like to use the numpyified version but I'm getting an error: **Input**: `df2['filename'] = df2['job_number'].values.astype(str) + '.' + df2['task_number'].values.astype(str)` --> **Output**: `TypeError: ufunc 'add' did not contain a loop with signature matching types`

`dtype('<U21') dtype('<U21') dtype('<U21')` . Both job_number and task_number are ints. – Karl Baker Mar 3, 2019 at 6:56

That's because you are combining two numpy arrays. It works if you combine an numpy array with pandas Series. as `df['Year'].values.astype(str) + df.quarter` – AbdulRehmanLiaqat Feb 10, 2020 at 11:23

2

One can use *assign* method of *DataFrame*:

```
df= (pd.DataFrame({'Year': ['2014', '2015'], 'quarter': ['q1', 'q2']}).
  assign(period=lambda x: x.Year+x.quarter ))
```

Share Edit Follow

answered Dec 1, 2018 at 10:55
Sergey
477 3 7

1

Similar to @geher answer but with any separator you like:

```
SEP = " "
INPUT_COLUMNS_WITH_SEP = ",sep,".join(INPUT_COLUMNS).split(",")

df.assign(sep=SEP)[INPUT_COLUMNS_WITH_SEP].sum(axis=1)
```

Share Edit Follow

answered Dec 4, 2021 at 12:43
Marc Torrellas Socastro
419 3 7

0

Use `.combine_first` .

```
df['Period'] = df['Year'].combine_first(df['Quarter'])
```

Share Edit Follow

edited Feb 12, 2018 at 7:54
Keiku
7,515 3 37 41

answered Feb 10, 2018 at 4:01
Abul
187 2 4 14

4 This is not correct. `.combine_first` will result in either the value from `'Year'` being stored in `'Period'` , or, if it is Null, the value from `'Quarter'` . It will not concatenate the two strings and store them in `'Period'` . – Steve G Jan 29, 2019 at 20:48

0

```
def madd(x):
    """Performs element-wise string concatenation with multiple input arrays.

    Args:
        x: iterable of np.array.

    Returns: np.array.
    """
    for i, arr in enumerate(x):
        if type(arr.item(0)) is not str:
            x[i] = x[i].astype(str)
    return reduce(np.core.defchararray.add, x)
```

For example:

```
data = list(zip([2000]*4, ['q1', 'q2', 'q3', 'q4']))
df = pd.DataFrame(data=data, columns=['Year', 'quarter'])
df['period'] = madd([df[col].values for col in ['Year', 'quarter']])

df
```

```
    Year    quarter period
0   2000    q1  2000q1
1   2000    q2  2000q2
2   2000    q3  2000q3
3   2000    q4  2000q4
```

Share Edit Follow

edited Jul 21, 2017 at 20:26

answered Jul 21, 2017 at 19:38

BMW
529 3 14

NameError: name 'reduce' is not defined – rubengavidia0x Mar 8 at 20:14

`from functools import reduce` – pauljohn32 May 25 at 4:22