Concatenating DataFrames

UBCO Master of Data Science – DATA 542

Fatemeh Fard

Concatenation along axes cont.

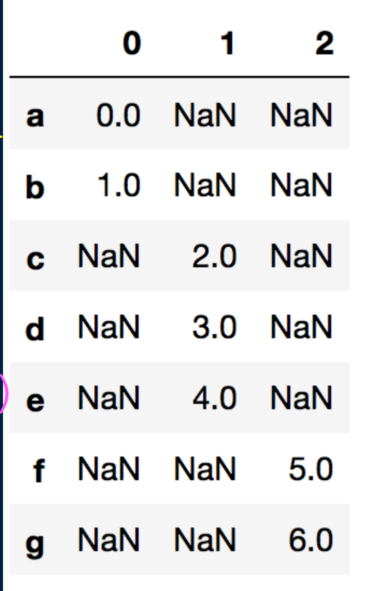
Use `pd.concat()` with the following attributes:

- `obj`: You can pass a list of Pandas objects
- `axis`: The axis along which the join operates
- `keys`: Values to associate with objects being concatenated, forming a hierarchical index along the concatenation axis.
- `how`: One of 'inner', 'outer', 'left', or 'right'; defaults to 'inner’.
- `verify_integrity`: Check new axis in concatenated object for duplicates and raise exception if so; by default (False) allows duplicates.
- `ignore_index`: Do not preserve indexes along concatenation axis, instead producing a new range(total_length) index
- `names`: Names for created hierarchical levels if keys and/or levels passed
- `join_axes`: Specific indexes to use for the other *n*–1 axes instead of performing union/intersection logic

Concatenation example

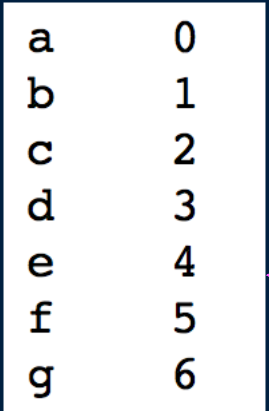
```
s1 = pd.Series([0, 1], index=['a', 'b'])
s2 = pd.Series([2, 3, 4], index=['c','d', 'e'])
s3 = pd.Series([5, 6], index=['f', 'g'])
```

```
pd.concat([s1, s2, s3],
axis=1, sort=False)
```



	0	1	2
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

```
pd.concat([s1, s2, s3])
```

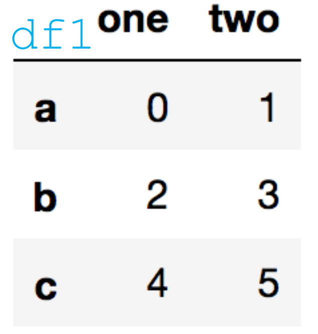


```
a        0
b        1
c        2
d        3
e        4
f        5
g        6
```

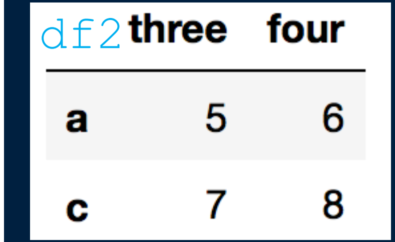
Important NOTE

If indices are not important, turn the flag ignore_index on to create new indices for the concatenated DataFrame

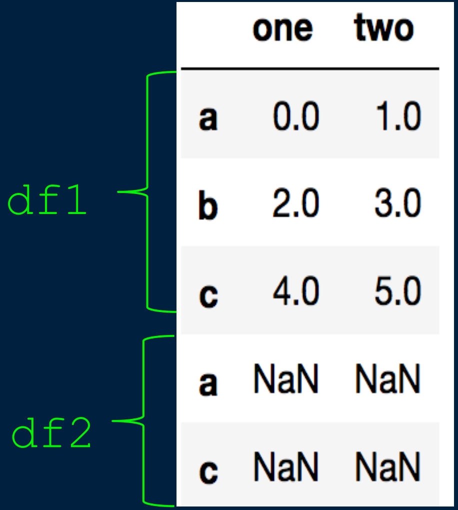
```
pd.concat([df1, df2] ,join_axes=[df1.columns],
ignore_index=True)
```



df1	one	two
a	0	1
b	2	3
c	4	5

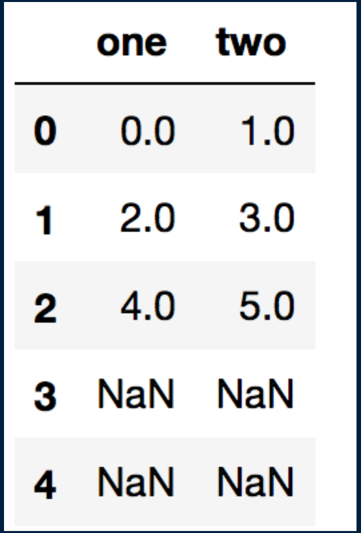


df2	three	four
a	5	6
c	7	8



	one	two
a	0.0	1.0
b	2.0	3.0
c	4.0	5.0
a	NaN	NaN
c	NaN	NaN

→



	one	two
0	0.0	1.0
1	2.0	3.0
2	4.0	5.0
3	NaN	NaN
4	NaN	NaN

Alternation and optimization

Use `append` instead of `concat()`: `df1.append(df2)`

`append` method in Pandas

- Does not modify the original object
- Is not an efficient method

Optimization:

Build a list of DataFrames

Pass them all at once to the `concat()` function.

Example

```
import os
import glob
import pandas as pd
# reading all csv files as a list
files = glob.glob(”PATH_TO_FILES/*.csv")
#concatenating all the csv files as a dataframe df
df = pd.concat([pd.read_csv(f) for f in files], ignore_index=True)
df.to_csv(”all_data.csv")
```

Basic Vectorized Strings Operations and Regex

Question

How can we multiply each element of an array (or a list) by 2?

```
l = [1, 2, 3]
[s*2 for s in l]
Output: [2, 4, 6]


l * 2
Output: [1, 2, 3, 1, 2, 3]
```

Vectorized Operations

```
import numpy as np
x = np.array([2, 3, 5, 7, 11, 13])
x * 2
Output: array([ 4, 6, 10, 14, 22, 26])
```

This *vectorization* of operations simplifies the syntax of operating on arrays of data.

Numpy does NOT provide simple operations on string arrays

String Vectorized Operations

Series and DataFrames have `str` attribute

It has a list of the vectorized string methods

len()	lower()	translate()	islower()
ljust()	upper()	startswith()	isupper()
rjust()	find()	endswith()	isnumeric()
center()	rfind()	isalnum()	isdecimal()
zfill()	index()	isalpha()	split()
strip()	rindex()	isdigit()	rsplit()
rstrip()	capitalize()	isspace()	partition()
lstrip()	swapcase()	istitle()	rpartition()

String Operations by Regular Expressions

A regular expression (or RE) specifies a set of strings that matches it

It lets you check if a particular string matches a given regular expression

. means any

[] means either element in the bracket

https://www.python-course.eu/re.php

Regex

User user! User!

user? Users users!

Letters inside []

Pattern	Matching
[uU]ser	User, user
[1 2 3 4 5 6 7 8 9 0]	Any digits

Ranges [A-Z]

Pattern	Matching
[A-Z]	An upper case letter.
[0-9]	Any digits

TRY IT: regexpal.com

We looked!

Then we saw him step in on the mat.

We looked!

And we saw him!

The cat in the hat!

[Ww]

[em]

[A-Za-z]

[!]

Regex

Negation ^

Pattern	Matching
[^A-Z]	Not an upper case letter
[^Aa]	Not A nor a

Pipe |

Pattern	Matching
Yours|mine	Yours mine
a|b|c	= [abc]

Regex

[^A-Za-z]

[^ !]

at|ook

Some REGEX Character Descriptions

Character	Description	Example
?	Match zero or one repetitions of preceding	"ab?" matches "a" or "ab"
*	Match zero or more repetitions of preceding	"ab*" matches "a", "ab", "abb ", "abbb"...
+	Match one or more repetitions of preceding	"ab+" matches "ab", "abb", "a bbb"... but not "a"
{n}	Match n repetitions of preeeding	"ab{2}" matches "abb"
{m,n}	Match between m and nrepetitions of preceding	"ab{2,3}" matches "abb" or "a bbb"

List of methods that accept regular expressions

Method	Description
match()	Call re.match() on each element, returning a boolean.
extract()	Call re.match() on each element, returning matched groups as strings.
findall()	Call re.findall() on each element
replace()	Replace occurrences of pattern with some other string
contains()	Call re.search() on each element, returning a boolean
count()	Count occurrences of pattern
split()	Equivalent to str.split(), but accepts regexps
rsplit()	Equivalent to str.rsplit(), but accepts regexps

Further Resources

https://www.python-course.eu/re.php

https://docs.python.org/3.6/library/re.html

https://jakevdp.github.io/WhirlwindTourOfPython/14-strings-and-regular-expressions.html

https://www.python-course.eu/python3_re_advanced.php

TRY IT (1 of 2)

```
monte = pd.Series(['Graham Chapman', 'John
Cleese', 'Terry Gilliam', 'Eric Idle', 'Terry
Jones', 'Michael Palin'])

monte.str.lower()

monte.str.len()

monte.str.startswith('T')

monte.str.split()
```

NOTE: You can use more complex functions using the apply method

TRY IT (2 of 2)

extract the first name from each by asking for a contiguous group of characters at the beginning of each element:

```
monte.str.extract('([A-Za-z]+)', expand=False)
```

finding all names that start and end with a consonant, making use of the start-of-string (^) and end-of-string ($) regular expression characters:

```
monte.str.findall(r'^[^AEIOU].*[^aeiou]$')
```

Time series
Strings ⇔ times

UBCO Master of Data Science – DATA 542

Motivation

Timeseries analysis

Date and Time calculations

Various time zones

Calculating periods

Calculating frequencies, elapse time

Sub-setting /slicing using data and time

Input issues:

- Various formats for data
- String inputs
- String inputs with different formats

Definitions

Timestamps , specific instants in time

Fixed periods , such as the month January 2007 or the full year 2010

Intervals of time, indicated by a start and end timestamp. Periods can be thought of as special cases of intervals

Experiment or elapsed time; each timestamp is a measure of time relative to a particular start time (e.g., the diameter of a cookie baking each second since being placed in the oven)

The simplest and most widely used kind of time series are those indexed by timestamp

Review: Python datetime and time

datetime stores both the date and time down to the microsecond

```
from  datetime  import  datetime
now = datetime.now()
now.year
now.month
```

Review: Python datetime and time cont.

timedelta represents the temporal difference between two datetime objects

```
 delta = datetime ( 2018, 12 , 11 ) -
datetime ( 2018 , 9 , 1 , 8 , 15 )
```

```
 delta.days
 delta.seconds
```

Python Timedelta

You can add (or subtract) a timedelta or multiple thereof to a datetime object to yield a new shifted object.

```
from  datetime  import  timedelta
start  =  datetime ( 2018 ,  9 ,  1 )
start  +  timedelta ( 101 )
```

Or, do any arithmetic calculations

TRY IT

Converting Between String and Datetime

Format datetime objects and pandas Timestamp objects, as strings using str() or the strftime() method, passing a format specification

```
stamp = datetime(2019, 12, 21)
str(stamp)
stamp.strftime('%Y- %m- %d’)
stamp.strftime('%Y- %d- %m')
```

Datetime format specification

Converting Between String and Datetime

Use many of the same format codes to convert strings to dates using datetime.strptime()

```
value  =  '2011-01-03'
datetime.strptime(value ,'%Y-%m-%d’)


datestrs = '7/6/2011'
datetime.strptime (datestrs, '%m/%d/%Y')
```

Converting Between String and Datetime cont. Python dateutil

You may get various input formats:

- '2011-01-03’
- 'Jan 31, 1997 10:45 PM’

Use `parser()` in `dateutil` package

```
from dateutil.parser import parse
parse ( '2011-01-03’ )
parse ( 'Jan 31, 1997 10:45 PM' )
```

Pandas datetime

dateutil.parser is a useful but imperfect tool. Notably, it will recognize some strings as dates that you might prefer that it didn’t— for example, '42' will be parsed as the year 2042 with today’s calendar date.

```
parse ( '42-01-03’ )

datetime.datetime(2042, 1, 3, 0, 0)
```

Time series analysis requires arrays of datetimes

Pandas datetime cont.

Create a DatetimeIndex

Use to_datetime()

Handles null values as NaT (Not a Time)

Indexing, slicing, handling null values

```
datestrs = [ '2011-07-06 12:00:00' , '2011-08-06 00:00:00' , None]
```

```
pd.to_datetime ( datestrs )
```

Output:
```
DatetimeIndex(['2011-07-06 12:00:00', '2011-08-06 00:00:00', 'NaT'], dtype='datetime64[ns]', freq=None)
```

TRY IT

Time Series

Simplest time series object in pandas is a Series indexed by timestamps

Arithmetic operations

Slicing `ts[::2]`

```
 dates = [datetime(2011,1,2),datetime(2011,1,5
),datetime(2011, 1 ,7 ),

datetime(2011,1,8),datetime(2011,1,10),datetime(201
1,1,12)]

 ts  =  pd.Series ( np.random.randn(6), index =
dates )
```

Slicing time series

Using previously learned DataFrame slicing (e.g. [])

Pass a string to [] that is interpretable as a date

Pass a year or only a year and month using []

Perform a range query

```
longer_ts[‘2001']
longer_ts['2002-02’]
longer_ts[datetime(2002,9,15):]
longer_ts['2002-9-9':'2002/9/20']
```

Time series in a DataFrame

The same logic applies for time series as index in a DataFrame

Create a DateTimeIndex:

```
 dates = pd.date_range('1,1,2019', periods=100 ,
freq='W-TUE')
```

Frequencies

Frequency refers to examples such as

- Every 15 minutes
- Daily
- Every month

You can resample your data by adding missing frequencies in your time series

```
dates = pd.date_range('1,1,2019', periods=100 , freq='W-TUE’)
```

```
Other frequency formats: ‘H’, ‘D’, ‘M’, ‘BM’
```

Frequencies cont.

Frequency:

- Base frequency (date offset)
- Multiplier

```
from pandas.tseries.offsets import Hour, Minute
Four_hour = Hour(4)


pd.date_range('2014-5-1', '2014/5/2',
freq='1H30Min’)
pd.date_range('2014-5-1', '2015/5/2',
freq='WOM-3FRI’)
```

Shifting and lagging

Time series analysis sometimes requires shifting the data through time

Backward

Forward

Shifting by default does not modify the index

Use frequency to keep the data and shift/modify the time

Resampling

Resampling is similar to groupby

Use resampling and then aggregate on the time

```
df4.resample('5Min', closed='right').sum()
```

Learning outcome

At the end of this lecture you should be able to:

Perform analysis with Datetime

Perform analysis with Datetime, timedelta

Understand the basics of time series

Perform functions on time series

THE UNIVERSITY OF BRITISH COLUMBIA