Experiment No. 6:Merging / Joining and Concatenation

TE

Prof. Dhanashree Salvi

Python Pandas - Merging/Joining

Pandas has full-featured, high performance in-memory join operations idiomatically very similar to relational databases like SQL.

Pandas provides a single function, **merge**, as the entry point for all standard database join operations between DataFrame objects −

```
pd.merge(left, right, how='inner', on=None, left_on=None,
right_on=None,
left_index=False, right_index=False, sort=True)
```

Here, we have used the following parameters −

- **left** − A DataFrame object.
- **right** − Another Data Frame object.
- **on** − Columns (names) to join on. Must be found in both the left and right Data Frame objects.
- **left_on** − Columns from the left DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame.
- **right_on** − Columns from the right DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame.
- **left_index** − If **True,** use the index (row labels) from the left DataFrame as its join key(s). In case of a DataFrame with a MultiIndex (hierarchical), the number of levels must match the number of join keys from the right DataFrame.
- **right_index** − Same usage as **left_index** for the right DataFrame.
- **how** − One of 'left', 'right', 'outer', 'inner'. Defaults to inner. Each method has been described below.
- **sort** − Sort the result DataFrame by the join keys in lexicographical order. Defaults to True, setting to False will improve the performance substantially in many cases.

Let us now create two different DataFrames and perform the merging operations on it.

```
# import the pandas library
import pandas as pd
```

```
left = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame(
   {'id':[1,2,3,4,5],
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print left
print right
```

Its **output** is as follows −

```
    Name  id   subject_id
0   Alex   1         sub1
1    Amy   2         sub2
2  Allen   3         sub4
3  Alice   4         sub6
4  Ayoung  5         sub5

    Name  id   subject_id
0  Billy   1         sub2
1  Brian   2         sub4
2  Bran    3         sub3
3  Bryce   4         sub6
4  Betty   5         sub5
```

Merge Two DataFrames on a Key

```
import pandas as pd
left = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
      'id':[1,2,3,4,5],
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left,right,on='id')
```

Its **output** is as follows −

```
   Name_x   id  subject_id_x   Name_y   subject_id_y
0  Alex      1          sub1    Billy           sub2
1  Amy       2          sub2    Brian           sub4
2  Allen     3          sub4     Bran           sub3
3  Alice     4          sub6    Bryce           sub6
4  Ayoung    5          sub5    Betty           sub5
```

Merge Two DataFrames on Multiple Keys

```
import pandas as pd
left = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
```

```
      'id':[1,2,3,4,5],
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left,right,on=['id','subject_id'])
```

Its **output** is as follows −

```
    Name_x   id   subject_id   Name_y
0    Alice    4         sub6    Bryce
1   Ayoung    5         sub5    Betty
```

Merge Using 'how' Argument

The **how** argument to merge specifies how to determine which keys are to be included in the resulting table. If a key combination does not appear in either the left or the right tables, the values in the joined table will be NA.

Here is a summary of the **how** options and their SQL equivalent names −

Merge Method	SQL Equivalent	Description
left	LEFT OUTER JOIN	Use keys from left object
right	RIGHT OUTER JOIN	Use keys from right object
outer	FULL OUTER JOIN	Use union of keys
inner	INNER JOIN	Use intersection of keys

Left Join

```
import pandas as pd
left = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left, right, on='subject_id', how='left')
```

Its **output** is as follows −

```
    Name_x   id_x   subject_id   Name_y   id_y
0     Alex      1         sub1      NaN    NaN
1      Amy      2         sub2    Billy    1.0
2    Allen      3         sub4    Brian    2.0
3    Alice      4         sub6    Bryce    4.0
```

```
4   Ayoung      5          sub5     Betty     5.0
```

Right Join

```
import pandas as pd
left = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left, right, on='subject_id', how='right')
```

Its **output** is as follows −

```
    Name_x  id_x   subject_id   Name_y   id_y
0      Amy   2.0         sub2    Billy      1
1    Allen   3.0         sub4    Brian      2
2    Alice   4.0         sub6    Bryce      4
3   Ayoung   5.0         sub5    Betty      5
4      NaN   NaN         sub3     Bran      3
```

Outer Join

```
import pandas as pd
left = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left, right, how='outer', on='subject_id')
```

Its **output** is as follows −

```
    Name_x  id_x   subject_id   Name_y   id_y
0     Alex   1.0         sub1      NaN    NaN
1      Amy   2.0         sub2    Billy    1.0
2    Allen   3.0         sub4    Brian    2.0
3    Alice   4.0         sub6    Bryce    4.0
4   Ayoung   5.0         sub5    Betty    5.0
5      NaN   NaN         sub3     Bran    3.0
```

Inner Join

Joining will be performed on index. Join operation honors the object on which it is called. So, **a.join(b)** is not equal to **b.join(a)**.

```
import pandas as pd
left = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
```

```
   'id':[1,2,3,4,5],
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})

print pd.merge(left, right, on='subject_id', how='inner')
```

Its **output** is as follows −

```
    Name_x   id_x   subject_id   Name_y   id_y
0      Amy      2         sub2    Billy      1
1    Allen      3         sub4    Brian      2
2    Alice      4         sub6    Bryce      4
3   Ayoung      5         sub5    Betty      5
```

Python Pandas - Concatenation

Pandas provides various facilities for easily combining together **Series, DataFrame**, and **Panel** objects.

```
 pd.concat(objs,axis=0,join='outer',join_axes=None,
ignore_index=False)
```

- **objs** − This is a sequence or mapping of Series, DataFrame, or Panel objects.
- **axis** − {0, 1, ...}, default 0. This is the axis to concatenate along.
- **join** − {‘inner’, ‘outer’}, default ‘outer’. How to handle indexes on other axis(es). Outer for union and inner for intersection.
- **ignore_index** − boolean, default False. If True, do not use the index values on the concatenation axis. The resulting axis will be labeled 0, ..., n - 1.
- **join_axes** − This is the list of Index objects. Specific indexes to use for the other (n-1) axes instead of performing inner/outer set logic.

Concatenating Objects

The **concat** function does all of the heavy lifting of performing concatenation operations along an axis. Let us create different objects and do concatenation.

```
import pandas as pd

one = pd.DataFrame({
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
```

```
   'subject_id':['sub1','sub2','sub4','sub6','sub5'],
   'Marks_scored':[98,90,87,69,78]},
   index=[1,2,3,4,5])

two = pd.DataFrame({
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5'],
   'Marks_scored':[89,80,79,97,88]},
   index=[1,2,3,4,5])
print pd.concat([one,two])
```

Its **output** is as follows −

```
    Marks_scored     Name   subject_id
1             98     Alex         sub1
2             90      Amy         sub2
3             87    Allen         sub4
4             69    Alice         sub6
5             78   Ayoung         sub5
1             89    Billy         sub2
2             80    Brian         sub4
3             79     Bran         sub3
4             97    Bryce         sub6
5             88    Betty         sub5
```

Suppose we wanted to associate specific keys with each of the pieces of the chopped up DataFrame. We can do this by using the **keys** argument −

```
import pandas as pd

one = pd.DataFrame({
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5'],
   'Marks_scored':[98,90,87,69,78]},
   index=[1,2,3,4,5])

two = pd.DataFrame({
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5'],
   'Marks_scored':[89,80,79,97,88]},
   index=[1,2,3,4,5])
print pd.concat([one,two],keys=['x','y'])
```

Its **output** is as follows −

```
x  1   98    Alex    sub1
   2   90    Amy     sub2
   3   87    Allen   sub4
   4   69    Alice   sub6
   5   78    Ayoung  sub5
y  1   89    Billy   sub2
   2   80    Brian   sub4
   3   79    Bran    sub3
   4   97    Bryce   sub6
   5   88    Betty   sub5
```

The index of the resultant is duplicated; each index is repeated.

If the resultant object has to follow its own indexing, set **ignore_index** to **True**.

```
import pandas as pd

one = pd.DataFrame({
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5'],
   'Marks_scored':[98,90,87,69,78]},
   index=[1,2,3,4,5])

two = pd.DataFrame({
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5'],
   'Marks_scored':[89,80,79,97,88]},
   index=[1,2,3,4,5])
print pd.concat([one,two],keys=['x','y'],ignore_index=True)
```

Its **output** is as follows −

```
    Marks_scored     Name    subject_id
0             98     Alex          sub1
1             90      Amy          sub2
2             87    Allen          sub4
3             69    Alice          sub6
4             78   Ayoung          sub5
5             89    Billy          sub2
6             80    Brian          sub4
7             79     Bran          sub3
8             97    Bryce          sub6
9             88    Betty          sub5
```

Observe, the index changes completely and the Keys are also overridden.

If two objects need to be added along **axis=1**, then the new columns will be appended.

```
import pandas as pd

one = pd.DataFrame({
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5'],
   'Marks_scored':[98,90,87,69,78]},
   index=[1,2,3,4,5])

two = pd.DataFrame({
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5'],
   'Marks_scored':[89,80,79,97,88]},
   index=[1,2,3,4,5])
print pd.concat([one,two],axis=1)
```

Its **output** is as follows −

```
    Marks_scored    Name  subject_id   Marks_scored    Name
subject_id
1           98      Alex      sub1          89         Billy
sub2
2           90       Amy      sub2          80         Brian
sub4
3           87     Allen      sub4          79          Bran
sub3
4           69     Alice      sub6          97         Bryce
sub6
5           78    Ayoung      sub5          88         Betty
sub5
```

Concatenating Using append

A useful shortcut to concat are the append instance methods on Series and DataFrame. These methods actually predated concat. They concatenate along **axis=0**, namely the index −

```
import pandas as pd

one = pd.DataFrame({
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5'],
   'Marks_scored':[98,90,87,69,78]},
   index=[1,2,3,4,5])

two = pd.DataFrame({
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5'],
   'Marks_scored':[89,80,79,97,88]},
   index=[1,2,3,4,5])
print one.append(two)
```

Its **output** is as follows −

```
    Marks_scored    Name  subject_id
1           98      Alex      sub1
2           90       Amy      sub2
3           87     Allen      sub4
4           69     Alice      sub6
5           78    Ayoung      sub5
1           89     Billy      sub2
2           80     Brian      sub4
3           79      Bran      sub3
4           97     Bryce      sub6
5           88     Betty      sub5
```

The **append** function can take multiple objects as well −

```
import pandas as pd

one = pd.DataFrame({
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5'],
   'Marks_scored':[98,90,87,69,78]},
```

```
   index=[1,2,3,4,5])

two = pd.DataFrame({
   'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5'],
   'Marks_scored':[89,80,79,97,88]},
   index=[1,2,3,4,5])
print one.append([two,one,two])
```

Its **output** is as follows −

```
    Marks_scored   Name    subject_id
1             98    Alex          sub1
2             90     Amy          sub2
3             87   Allen          sub4
4             69   Alice          sub6
5             78  Ayoung          sub5
1             89   Billy          sub2
2             80   Brian          sub4
3             79    Bran          sub3
4             97   Bryce          sub6
5             88   Betty          sub5
1             98    Alex          sub1
2             90     Amy          sub2
3             87   Allen          sub4
4             69   Alice          sub6
5             78  Ayoung          sub5
1             89   Billy          sub2
2             80   Brian          sub4
3             79    Bran          sub3
4             97   Bryce          sub6
5             88   Betty          sub5
```

Time Series

Pandas provide a robust tool for working time with Time series data, especially in the financial sector. While working with time series data, we frequently come across the following −

- Generating sequence of time
- Convert the time series to different frequencies

Pandas provides a relatively compact and self-contained set of tools for performing the above tasks.

Get Current Time

datetime.now() gives you the current date and time.

```
import pandas as pd

print pd.datetime.now()
```

Its **output** is as follows −

```
2017-05-11 06:10:13.393147
```

Create a TimeStamp

Time-stamped data is the most basic type of timeseries data that associates values with points in time. For pandas objects, it means using the points in time. Let's take an example −

```
import pandas as pd

print pd.Timestamp('2017-03-01')
```

Its **output** is as follows −

```
2017-03-01 00:00:00
```

It is also possible to convert integer or float epoch times. The default unit for these is nanoseconds (since these are how Timestamps are stored). However, often epochs are stored in another unit which can be specified. Let's take another example

```
import pandas as pd

print pd.Timestamp(1587687255,unit='s')
```

Its **output** is as follows −

```
2020-04-24 00:14:15
```

Create a Range of Time

```
import pandas as pd

print pd.date_range("11:00", "13:30", freq="30min").time
```

Its **output** is as follows −

```
[datetime.time(11, 0) datetime.time(11, 30) datetime.time(12, 0)
datetime.time(12, 30) datetime.time(13, 0) datetime.time(13, 30)]
```

Change the Frequency of Time

```
import pandas as pd

print pd.date_range("11:00", "13:30", freq="H").time
```

Its **output** is as follows −

```
[datetime.time(11, 0) datetime.time(12, 0) datetime.time(13, 0)]
```

Converting to Timestamps

To convert a Series or list-like object of date-like objects, for example strings, epochs, or a mixture, you can use the **to_datetime** function. When passed, this returns a Series (with the same index), while a **list-like** is converted to a **DatetimeIndex**. Take a look at the following example −

```
import pandas as pd
```

```
print pd.to_datetime(pd.Series(['Jul 31, 2009','2010-01-10',
None]))
```

Its **output** is as follows −

```
0  2009-07-31
1  2010-01-10
2         NaT
dtype: datetime64[ns]
```

NaT means **Not a Time** (equivalent to NaN)

Let's take another example.

```
import pandas as pd

print pd.to_datetime(['2005/11/23', '2010.12.31', None])
```

Its **output** is as follows −

```
DatetimeIndex(['2005-11-23', '2010-12-31', 'NaT'],
dtype='datetime64[ns]', freq=None)
```