< 层次化的索引 | 目录 | 组合数据集：Merge 和 Join >

Open in Colab

Combining Datasets: Concat and Append

组合数据集：Concat 和 Append

> Some of the most interesting studies of data come from combining different data sources. These operations can involve anything from very straightforward concatenation of two different datasets, to more complicated database-style joins and merges that correctly handle any overlaps between the datasets. `Series` and `DataFrame`s are built with this type of operation in mind, and Pandas includes functions and methods that make this sort of data wrangling fast and straightforward.

很多对数据进行的有趣的研究都来源自不同数据源的组合。这些组合操作包括很直接的连接两个不同的数据集，到更复杂的数据库风格的联表和组合可以正确的处理数据集之间的重复部分。`Series`和`DataFrame`內建了对这些操作的支持，Pandas提供的函数和方法能够让这种数据操作高效而直接。

> Here we'll take a look at simple concatenation of `Series` and `DataFrame`s with the `pd.concat` function; later we'll dive into more sophisticated in-memory merges and joins implemented in Pandas.

本节中我们会简单介绍使用`pd.concat`函数对`Series`和`DataFrame`进行连接；然后我们深入讨论Pandas中复杂的内存级别的合并及联表操作。

> We begin with the standard imports:

首先还是标准载入：

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

> For convenience, we'll define this function which creates a `DataFrame` of a particular form that will be useful below:

为了方便起见，我们定义下面这个函数用来创建一个`DataFrame`，本节后续的`DataFrame`都来源自该函数：

```
In [2]: def make_df(cols, ind):
            """Quickly make a DataFrame"""
            data = {c: [str(c) + str(i) for i in ind]
                    for c in cols}
            return pd.DataFrame(data, ind)

        # example DataFrame
        make_df('ABC', range(3))
```

```
Out[2]:
            A   B   C
        0   A0  B0  C0
        1   A1  B1  C1
        2   A2  B2  C2
```

> In addition, we'll create a quick class that allows us to display multiple `DataFrame`s side by side. The code makes use of the special `_repr_html_` method, which IPython uses to implement its rich object display:

除此之外，我们还要创建一个类，用来将多个`DataFrame`紧靠着进行展示。下面的代码实现了特殊的`_repr_html_`方法，IPython使用这个方法来展示对象的HTML格式：

```
In [3]: class display(object):
            """多个对象的HTML格式展示"""
            template = """<div style="float: left; padding: 10px;">
            <p style='font-family:"Courier New", Courier, monospace'>{0}</p>{1}
            </div>"""
            def __init__(self, *args):
                self.args = args

            def _repr_html_(self):
                return '\n'.join(self.template.format(a, eval(a)._repr_html_())
                                 for a in self.args)

            def __repr__(self):
                return '\n\n'.join(a + '\n' + repr(eval(a))
                                   for a in self.args)
```

> The use of this will become clearer as we continue our discussion in the following section.

这个类的使用方式会在后续进一步介绍。

Recall: Concatenation of NumPy Arrays

复习：NumPy数组的连接

> Concatenation of `Series` and `DataFrame` objects is very similar to concatenation of Numpy arrays, which can be done via the `np.concatenate` function as discussed in The Basics of NumPy Arrays. Recall that with it, you can combine the contents of two or more arrays into a single array:

`Series`和`DataFrame`对象的连接与NumPy数组的连接非常相似，NumPy数组我们可以通过NumPy数组基础一节中介绍过的`np.concatenate`函数来实现。回忆一下，你可以将两个或多个数组连接成一个数组：

```
In [4]: x = [1, 2, 3]
        y = [4, 5, 6]
        z = [7, 8, 9]
        np.concatenate([x, y, z])

Out[4]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

> The first argument is a list or tuple of arrays to concatenate. Additionally, it takes an `axis` keyword that allows you to specify the axis along which the result will be concatenated:

第一个参数是需要进行连接的数组的元组或列表。函数还可以提供一个`axis`关键字参数来指定沿着哪个维度方向对数组进行连接：

```
In [5]: x = [[1, 2],
             [3, 4]]
        np.concatenate([x, x], axis=1)

Out[5]: array([[1, 2, 1, 2],
               [3, 4, 3, 4]])
```

Simple Concatenation with `pd.concat`

使用`pd.concat`进行简单连接

> Pandas has a function, `pd.concat()`, which has a similar syntax to `np.concatenate` but contains a number of options that we'll discuss momentarily:

Pandas有相应的函数`pd.concat()`，与`np.concatenate`有着相似的语法，但是有一些参数我们需要深入讨论：

```
# Pandas v0.24.2的函数签名
pd.concat(
    objs,
    axis=0,
    join='outer',
    join_axes=None,
    ignore_index=False,
    keys=None,
    levels=None,
    names=None,
    verify_integrity=False,
    sort=None,
    copy=True,
)
```

> `pd.concat()` can be used for a simple concatenation of `Series` or `DataFrame` objects, just as `np.concatenate()` can be used for simple concatenations of arrays:

`pd.concat()`可以用来对`Series`或`DataFrame`对象进行简单的连接，就像可以用`np.concatenate()`来对数组进行简单连接一样：

```
In [6]: ser1 = pd.Series(['A', 'B', 'C'], index=[1, 2, 3])
        ser2 = pd.Series(['D', 'E', 'F'], index=[4, 5, 6])
        pd.concat([ser1, ser2])

Out[6]: 1    A
        2    B
        3    C
        4    D
        5    E
        6    F
        dtype: object
```

> It also works to concatenate higher-dimensional objects, such as `DataFrame`s:

`pd.concat()`函数也可以应用到高维对象上，例如`DataFrame`：

```
In [7]: df1 = make_df('AB', [1, 2])
        df2 = make_df('AB', [3, 4])
        display('df1', 'df2', 'pd.concat([df1, df2])')
```

```
Out[7]:
         df1          df2          pd.concat([df1, df2])

            A   B        A   B        A   B
         1  A1  B1    3  A3  B3    1  A1  B1
         2  A2  B2    4  A4  B4    2  A2  B2
                                   3  A3  B3
                                   4  A4  B4
```

> By default, the concatenation takes place row-wise within the `DataFrame` (i.e., `axis=0`). Like `np.concatenate`, `pd.concat` allows specification of an axis along which concatenation will take place. Consider the following example:

默认情况下，连接会按照`DataFrame`的行来进行（即`axis=0`）。就像`np.concatenate`那样，`pd.concat`允许指定沿着哪个维度方向进行连接，看下例：

```
In [8]: df3 = make_df('AB', [0, 1])
        df4 = make_df('CD', [0, 1])
        display('df3', 'df4', "pd.concat([df3, df4], axis='columns')")
```

```
Out[8]:
         df3          df4          pd.concat([df3, df4], axis='columns')

            A   B        C   D        A   B   C   D
         0  A0  B0    0  C0  D0    0  A0  B0  C0  D0
         1  A1  B1    1  C1  D1    1  A1  B1  C1  D1
```

> We could have equivalently specified `axis=1`; here we've used the more intuitive `axis='col'`.

我们也可以使用相同的声明方式`axis=1`；这里我们使用了更加直观的方式`axis='columns'`。

译者注：原文中axis的参数是`'col'`，这个参数在新版本的Pandas中已经改为`'columns'`。

Duplicate indices

重复的行索引

> One important difference between `np.concatenate` and `pd.concat` is that Pandas concatenation *preserves indices*, even if the result will have duplicate indices! Consider this simple example:

`np.contenate`和`pd.concat`的一个重要区别是Pandas的连接会*保留行索引*，甚至在结果中包含重复索引的情况下。看下例：

```
In [9]: x = make_df('AB', [0, 1])
        y = make_df('AB', [2, 3])
        y.index = x.index  # make duplicate indices!
        display('x', 'y', 'pd.concat([x, y])')
```

```
Out[9]:
         x            y            pd.concat([x, y])

            A   B        A   B        A   B
         0  A0  B0    0  A2  B2    0  A0  B0
         1  A1  B1    1  A3  B3    1  A1  B1
                                   0  A2  B2
                                   1  A3  B3
```

> Notice the repeated indices in the result. While this is valid within `DataFrame`s, the outcome is often undesirable. `pd.concat()` gives us a few ways to handle it.

注意看到结果中的重复索引。虽然这是`DataFrame`允许的，但是结果通常不是你希望的。`pd.concat()`提供了一些处理这个问题的方法。

Catching the repeats as an error

将重复的索引捕获为错误

> If you'd like to simply verify that the indices in the result of `pd.concat()` do not overlap, you can specify the `verify_integrity` flag. With this set to True, the concatenation will raise an exception if there are duplicate indices. Here is an example, where for clarity we'll catch and print the error message:

如果你希望简单的进行验证`pd.concat()`结果数据集中是否含有重复的索引，你可以传递参数`verify_integrity=True`参数。这时连接结果的数据集中如果存在重复的行索引，将会抛出一个错误。下面这个例子，我们将捕获到这个错误并输出：

```
In [10]: try:
             pd.concat([x, y], verify_integrity=True)
         except ValueError as e:
             print("ValueError:", e)

         ValueError: Indexes have overlapping values: Int64Index([0, 1], dtype='int64')
```

Ignoring the index

忽略行索引

> Sometimes the index itself does not matter, and you would prefer it to simply be ignored. This option can be specified using the `ignore_index` flag. With this set to true, the concatenation will create a new integer index for the resulting `Series`:

有些情况下，索引本身并不重要，那么可以选择忽略它们。给函数传递一个`ignore_index=True`的参数，`pd.concat`函数会忽略连接时的行索引，并在结果中重新创建一个整数的索引值：

```
In [11]: display('x', 'y', 'pd.concat([x, y], ignore_index=True)')
```

```
Out[11]:
         x            y            pd.concat([x, y], ignore_index=True)

            A   B        A   B        A   B
         0  A0  B0    0  A2  B2    0  A0  B0
         1  A1  B1    1  A3  B3    1  A1  B1
                                   2  A2  B2
                                   3  A3  B3
```

Adding MultiIndex keys

增加多重索引标签

> Another option is to use the `keys` option to specify a label for the data sources; the result will be a hierarchically indexed series containing the data:

还有一种方法是使用`keys`参数来指定不同数据集的索引标签；这时`pd.concat`的结果会是包含着连接数据集的多重索引数据集：

```
In [12]: display('x', 'y', "pd.concat([x, y], keys=['x', 'y'])")
```

```
Out[12]:
         x            y            pd.concat([x, y], keys=['x', 'y'])

            A   B        A   B            A   B
         0  A0  B0    0  A2  B2    x  0  A0  B0
         1  A1  B1    1  A3  B3       1  A1  B1
                                   y  0  A2  B2
                                      1  A3  B3
```

> The result is a multiply indexed `DataFrame`, and we can use the tools discussed in Hierarchical Indexing to transform this data into the representation we're interested in.

上例中的结果是一个多重索引的`DataFrame`，我们可以使用层次化的索引中介绍到的方法来转换或者展示连接结果的数据。

Concatenation with joins

使用联表方式连接

> In the simple examples we just looked at, we were mainly concatenating `DataFrame`s with shared column names. In practice, data from different sources might have different sets of column names, and `pd.concat` offers several options in this case. Consider the concatenation of the following two `DataFrame`s, which have some (but not all!) columns in common:

在上面我们看到的简单例子中，我们连接的数据集都具有相同的列及标签。在实际情况中，从不同源得到的数据通常具有不同的列数或者列标签，`pd.concat`提供了几个相应的参数帮助我们完成上面的任务。下例中的两个数据集只有部分（非全部）列和标签相同：

译者注：新版的Pandas修改了`sort`参数的默认值，后续该参数会默认为False。

```
In [13]: df5 = make_df('ABC', [1, 2])
         df6 = make_df('BCD', [3, 4])
         display('df5', 'df6', 'pd.concat([df5, df6])')
```

```
Out[13]:
         df5              df6              pd.concat([df5, df6])

            A   B   C        B   C   D        A    B   C   D
         1  A1  B1  C1    3  B3  C3  D3    1  A1   B1  C1  NaN
         2  A2  B2  C2    4  B4  C4  D4    2  A2   B2  C2  NaN
                                           3  NaN  B3  C3  D3
                                           4  NaN  B4  C4  D4
```

> By default, the entries for which no data is available are filled with NA values. To change this, we can specify one of several options for the `join` and `join_axes` parameters of the concatenate function. By default, the join is a union of the input columns (`join='outer'`), but we can change this to an intersection of the columns using `join='inner'`:

默认情况下，那些对应源数据集中不存在的元素值，将被填充为NA值。如果想改变默认行为，我们可以通过指定`join`和`join_axes`参数来实现。`join`参数默认为`join='outer'`，就像我们上面看到的情况，结果是数据集的并集；如果将`join='inner'`传递给`pd.concat`，那么就会是数据源中相同的列保留在结果中，因此结果是数据集的交集：

```
In [14]: display('df5', 'df6',
                 "pd.concat([df5, df6], join='inner')")
```

```
Out[14]:
         df5              df6              pd.concat([df5, df6], join='inner')

            A   B   C        B   C   D        B   C
         1  A1  B1  C1    3  B3  C3  D3    1  B1  C1
         2  A2  B2  C2    4  B4  C4  D4    2  B2  C2
                                           3  B3  C3
                                           4  B4  C4
```

> Another option is to directly specify the index of the remaininig colums using the `join_axes` argument, which takes a list of index objects. Here we'll specify that the returned columns should be the same as those of the first input:

还可以通过另一个参数`join_axes`来指定结果中保留的列，该参数接受被保留索引标签的列表。下例中我们指定结果中的列和第一个进行连接的数据集完全相同：

译者注：1.0版的pandas已经去掉了`josn_axes`关键字参数，可以通过`reindex`方法达到同样的目的，下面使用了`reindex`语法保留了`df5`的所有列。

```
In [15]: display('df5', 'df6',
                 "pd.concat([df5, df6]).reindex(df5.columns, axis=1)")
```

```
Out[15]:
         df5              df6              pd.concat([df5, df6]).reindex(df5.columns, axis=1)

            A   B   C        B   C   D        A    B   C
         1  A1  B1  C1    3  B3  C3  D3    1  A1   B1  C1
         2  A2  B2  C2    4  B4  C4  D4    2  A2   B2  C2
                                           3  NaN  B3  C3
                                           4  NaN  B4  C4
```

> The combination of options of the `pd.concat` function allows a wide range of possible behaviors when joining two datasets; keep these in mind as you use these tools for your own data.

`pd.concat`函数的参数很多，组合使用它们能解决组合多个数据集中的很多问题；请记住当你在自己的数据上操作时，你可以灵活地应用它们，完成你的工作目标。

The `append()` method

`append()` 方法

> Because direct array concatenation is so common, `Series` and `DataFrame` objects have an `append` method that can accomplish the same thing in fewer keystrokes. For example, rather than calling `pd.concat([df1, df2])`, you can simply call `df1.append(df2)`:

因为数据集的连接操作是很普遍的，`Series`和`DataFrame`对象都有一个`append`方法，它能完成和`pd.concat`一样的功能，并能让让你写代码时节省几次敲击键盘的动作。例如你可以简单是调用`df1.append(df2)`而不是调用`pd.concat([df1, df2])`：

```
In [16]: display('df1', 'df2', 'df1.append(df2)')
```

```
Out[16]:
         df1          df2          df1.append(df2)

            A   B        A   B        A   B
         1  A1  B1    3  A3  B3    1  A1  B1
         2  A2  B2    4  A4  B4    2  A2  B2
                                   3  A3  B3
                                   4  A4  B4
```

> Keep in mind that unlike the `append()` and `extend()` methods of Python lists, the `append()` method in Pandas does not modify the original object–instead it creates a new object with the combined data. It also is not a very efficient method, because it involves creation of a new index *and* data buffer. Thus, if you plan to do multiple `append` operations, it is generally better to build a list of `DataFrame`s and pass them all at once to the `concat()` function.

最后记住不像Python列表的`append()`和`extend`方法，Pandas中的`append()`方法不会修改原始参与运算的数据集，它会为合并后的结果创建一个新的对象。它也不是一个很高性能的方法，因为涉及到新索引和数据缓冲区的创建。因此如果你有需要连接多个数据集时，应该避免多次使用`append`方法，而是将所有需要进行连接的数据集形成一个列表，并传递给`concat`函数来进行连接操作。

> In the next section, we'll look at another more powerful approach to combining data from multiple sources, the database-style merges/joins implemented in `pd.merge`. For more information on `concat()`, `append()`, and related functionality, see the "Merge, Join, and Concatenate" section of the Pandas documentation.

下一节中，我们会介绍另外一种更强大的从不同数据源组合数据的方法，即数据库风格的联表和合并`pd.merge`。需要查阅更多有关`concat()`、`append()`的知识，可以访问Pandas在线文档 - "合并、联表及连接"。

< 层次化的索引 | 目录 | 组合数据集：Merge 和 Join >

Open in Colab