Data Wrangling: Join, Combine, and Reshape

Part 3

Combining and Merging Datasets

Part 2

Concatenating Along an Axis

- Another kind of data combination operation is referred to interchangeably as concatenation, binding, or stacking.

- NumPy's `concatenate` function can do this with NumPy arrays:

```
In [85]: arr = np.arange(12).reshape((3, 4))

In [86]: arr
Out[86]: array([[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11]])

In [87]: np.concatenate([arr, arr], axis=1)
Out[87]: array([[ 0,  1,  2,  3,  0,  1,  2,  3],
                [ 4,  5,  6,  7,  4,  5,  6,  7],
                [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

- In the context of pandas objects such as Series and DataFrame, having labeled axes enable you to further generalize array concatenation.
- In particular, you have a number of additional things to think about:
  - If the objects are indexed differently on the other axes, should we combine the distinct elements in these axes or use only the shared values (the intersection)?
  - Do the concatenated chunks of data need to be identifiable in the resulting object?
  - Does the “concatenation axis” contain data that needs to be preserved? In many cases, the default integer labels in a DataFrame are best discarded during concatenation.

- Suppose we have three Series with no index overlap:

```
In [88]: s1 = pd.Series([0, 1], index=['a', 'b'])

In [89]: s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])

In [90]: s3 = pd.Series([5, 6], index=['f', 'g'])

In [91]: pd.concat([s1, s2, s3])

Out[91]: a    0
         b    1
         c    2
         d    3
         e    4
         f    5
         g    6
         dtype: int64
```

- By default concat works along `axis=0`, producing another Series.
- If you pass `axis=1`, the result will instead be a DataFrame (`axis=1` is the columns):

```
In [92]: pd.concat([s1, s2, s3], axis=1)
```

```
/home/joshua/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: Sorting because non-concatena
tion axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  """Entry point for launching an IPython kernel.
```

Out[92]:

	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

In [93]:
```
s4 = pd.concat([s1, s3])
```

In [94]:
```
s4
```

Out[94]:
```
a    0
b    1
f    5
g    6
dtype: int64
```

In [95]:
```
pd.concat([s1, s4], axis=1)
```

```
/home/joshua/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: Fut
tion axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  """Entry point for launching an IPython kernel.
```

Out[95]:

	0	1
a	0.0	0
b	1.0	1
f	NaN	5
g	NaN	6

In [96]:
```
pd.concat([s1, s4], axis=1, join='inner')
```

Out[96]:

	0	1
a	0	0
b	1	1

- You can even specify the axes to be used on the other axes with `join_axes`:

```
In [97]: s1
Out[97]: a    0
         b    1
         dtype: int64

In [98]: s4
Out[98]: a    0
         b    1
         f    5
         g    6
         dtype: int64

In [99]: pd.concat([s1, s4], axis=1, join_axes=[['a', 'c', 'b', 'e']])
Out[99]:
```

	0	1
a	0.0	0.0
c	NaN	NaN
b	1.0	1.0
e	NaN	NaN

- A potential issue is that the concatenated pieces are not identifiable in the result.

```
In [104]: s1

Out[104]: a    0
          b    1
          dtype: int64

In [105]: s3

Out[105]: f    5
          g    6
          dtype: int64

In [106]: pd.concat([s1, s1, s3])

Out[106]: a    0
          b    1
          a    0
          b    1
          f    5
          g    6
          dtype: int64
```

- Suppose instead you wanted to create a hierarchical index on the concatenation axis.
- To do this, use the `keys` argument:

```
In [107]: s1
Out[107]: a    0
          b    1
          dtype: int64

In [108]: s3
Out[108]: f    5
          g    6
          dtype: int64
```

```
In [109]: result = pd.concat([s1, s1, s3], keys=['one', 'two', 'three'])

In [110]: result
Out[110]: one    a    0
                 b    1
          two    a    0
                 b    1
          three  f    5
                 g    6
          dtype: int64

In [111]: result.unstack()
Out[111]:
```

	a	b	f	g
one	0.0	1.0	NaN	NaN
two	0.0	1.0	NaN	NaN
three	NaN	NaN	5.0	6.0

- In the case of combining Series along `axis=1`, the keys become the DataFrame column headers:

In [112]:
```
pd.concat([s1, s2, s3], axis=1, keys=['one', 'two', 'three'])
```

```
/home/joshua/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: Sorting because non-concatena
tion axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  """Entry point for launching an IPython kernel.
```

Out[112]:

	one	two	three
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

- The same logic extends to DataFrame objects:

```
In [113]: df1 = pd.DataFrame(np.arange(6).reshape(3, 2), index=['a', 'b', 'c'],
                             columns=['one', 'two'])

In [114]: df2 = pd.DataFrame(5 + np.arange(4).reshape(2, 2), index=['a', 'c'],
                             columns=['three', 'four'])

In [115]: df1
```

Out[115]:

	one	two
a	0	1
b	2	3
c	4	5

```
In [116]: df2
```

Out[116]:

	three	four
a	5	6
c	7	8

```
In [117]: pd.concat([df1, df2], axis=1, keys=['level1', 'level2'])
```

```
/home/joshua/anaconda3/lib/python3.7/site-packages/ipykernel_
tion axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass

  """Entry point for launching an IPython kernel.
```

Out[117]:

	level1		level2	
	one	two	three	four
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0

- If you pass a dict of objects instead of a list, the dict's keys will be used for the `keys` option:

In [118]:
```
pd.concat({'level1': df1, 'level2': df2}, axis=1)
```

```
/home/joshua/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: Sorting because non-concatena
tion axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  """Entry point for launching an IPython kernel.
```

Out[118]:

	level1		level2	
	one	two	three	four
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0

- We can name the created axis levels with the `names` argument:

In [119]:
```
pd.concat([df1, df2], axis=1, keys=['level1', 'level2'],
          names=['upper', 'lower'])
```

```
/home/joshua/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: FutureWarning: Sorting because non-concatena
tion axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.
```

Out[119]:

upper	level1		level2	
lower	one	two	three	four
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0

- A last consideration concerns DataFrames in which the row index does not contain any relevant data:

```
In [120]: df1 = pd.DataFrame(np.random.randn(3, 4), columns=['a', 'b', 'c', 'd'])

In [121]: df2 = pd.DataFrame(np.random.randn(2, 3), columns=['b', 'd', 'a'])

In [122]: df1
```

Out[122]:

	a	b	c	d
0	1.246435	1.007189	-1.296221	0.274992
1	0.228913	1.352917	0.886429	-2.001637
2	-0.371843	1.669025	-0.438570	-0.539741

```
In [123]: df2
```

Out[123]:

	b	d	a
0	0.476985	3.248944	-1.021228
1	-0.577087	0.124121	0.302614

```
In [124]: pd.concat([df1, df2])
```

```
/home/joshua/anaconda3/lib/python3.7/site-packages/ipyke
tion axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning,

  """Entry point for launching an IPython kernel.
```

Out[124]:

	a	b	c	d
0	1.246435	1.007189	-1.296221	0.274992
1	0.228913	1.352917	0.886429	-2.001637
2	-0.371843	1.669025	-0.438570	-0.539741
0	-1.021228	0.476985	NaN	3.248944
1	0.302614	-0.577087	NaN	0.124121

- In this case, you can pass `ignore_index=True`:

In [125]:
```
pd.concat([df1, df2], ignore_index=True)
```

```
/home/joshua/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: Sorting because non-concatena
tion axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  """Entry point for launching an IPython kernel.
```

Out[125]:

	a	b	c	d
0	1.246435	1.007189	-1.296221	0.274992
1	0.228913	1.352917	0.886429	-2.001637
2	-0.371843	1.669025	-0.438570	-0.539741
3	-1.021228	0.476985	NaN	3.248944
4	0.302614	-0.577087	NaN	0.124121

Combining Data with Overlap

- There is another data combination situation that can’t be expressed as either a merge or concatenation operation.
- You may have two datasets whose indexes overlap in full or part.

- As a motivating example, consider NumPy's `where` function, which performs the array-oriented equivalent of an if-else expression:

```
In [132]: a = pd.Series([np.nan, 2.5, 0.0, 3.5, 4.5, np.nan],
                        index=['f', 'e', 'd', 'c', 'b', 'a'])

In [133]: b = pd.Series([0., np.nan, 2., np.nan, np.nan, 5.],
                        index=['a', 'b', 'c', 'd', 'e', 'f'])

In [134]: a

Out[134]: f    NaN
          e    2.5
          d    0.0
          c    3.5
          b    4.5
          a    NaN
          dtype: float64

In [135]: b

Out[135]: a    0.0
          b    NaN
          c    2.0
          d    NaN
          e    NaN
          f    5.0
          dtype: float64

In [136]: np.where(pd.isnull(a), b, a)

Out[136]: array([0. , 2.5, 0. , 3.5, 4.5, 5. ])
```

- Series has a `combine_first` method, which performs the equivalent of this operation along with pandas's usual data alignment logic:

```
In [134]: a
Out[134]: f    NaN
          e    2.5
          d    0.0
          c    3.5
          b    4.5
          a    NaN
          dtype: float64

In [135]: b
Out[135]: a    0.0
          b    NaN
          c    2.0
          d    NaN
          e    NaN
          f    5.0
          dtype: float64
```

```
In [137]: b.combine_first(a)
Out[137]: a    0.0
          b    4.5
          c    2.0
          d    0.0
          e    2.5
          f    5.0
          dtype: float64
```

- With DataFrames, `combine_first` does the same thing column by column, so you can think of it as "patching" missing data in the calling object with data from the object you pass:

```
In [138]: df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],
                              'b': [np.nan, 2., np.nan, 6.],
                              'c': range(2, 18, 4)})

In [139]: df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],
                              'b': [np.nan, 3., 4., 6., 8.]})

In [140]: df1
```

Out[140]:

	a	b	c
0	1.0	NaN	2
1	NaN	2.0	6
2	5.0	NaN	10
3	NaN	6.0	14

```
In [141]: df2
```

Out[141]:

	a	b
0	5.0	NaN
1	4.0	3.0
2	NaN	4.0
3	3.0	6.0
4	7.0	8.0

```
In [142]: df1.combine_first(df2)
```

Out[142]:

	a	b	c
0	1.0	NaN	2.0
1	4.0	2.0	6.0
2	5.0	4.0	10.0
3	3.0	6.0	14.0
4	7.0	8.0	NaN