Data Wrangling: Join, Combine, and Reshape

Part 4

Reshaping and Pivoting

Reshaping with Hierarchical Indexing

- Hierarchical indexing provides a consistent way to rearrange data in a DataFrame.
- There are two primary actions:
  - `stack`
    - This “rotates” or pivots from the columns in the data to the rows
  - `unstack`
    - This pivots from the rows into the columns

- Consider a small DataFrame with string arrays as row and column indexes:

```
In [143]: data = pd.DataFrame(np.arange(6).reshape((2, 3)),
                              index=pd.Index(['Ohio', 'Colorado'], name='state'),
                              columns=pd.Index(['one', 'two', 'three'],
                              name='number'))

In [144]: data
```

Out[144]:

number	one	two	three
state			
Ohio	0	1	2
Colorado	3	4	5

- Using the `stack` method on this data pivots the columns into the rows, producing a Series:

```
In [144]: data
```

Out[144]:

number	one	two	three
state			
Ohio	0	1	2
Colorado	3	4	5

```
In [145]: result = data.stack()

In [146]: result
Out[146]: state     number
          Ohio      one       0
                    two       1
                    three     2
          Colorado  one       3
                    two       4
                    three     5
          dtype: int64
```

- From a hierarchically indexed Series, you can rearrange the data back into a DataFrame with `unstack`:

```
In [146]: result
Out[146]: state     number
          Ohio      one       0
                    two       1
                    three     2
          Colorado  one       3
                    two       4
                    three     5
          dtype: int64

In [147]: result.unstack()
Out[147]:
```

number	one	two	three
state			
Ohio	0	1	2
Colorado	3	4	5

- By default the innermost level is unstacked (same with `stack`).
- You can unstack a different level by passing a level number or name:

```
In [146]: result
Out[146]: state     number
          Ohio      one       0
                    two       1
                    three     2
          Colorado  one       3
                    two       4
                    three     5
          dtype: int64
```

```
In [148]: result.unstack(0)
```

Out[148]:

state	Ohio	Colorado
number		
one	0	3
two	1	4
three	2	5

```
In [149]: result.unstack('state')
```

Out[149]:

state	Ohio	Colorado
number		
one	0	3
two	1	4
three	2	5

- Unstacking might introduce missing data if all of the values in the level aren’t found in each of the subgroups:

```
In [150]: s1 = pd.Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])

In [151]: s2 = pd.Series([4, 5, 6], index=['c', 'd', 'e'])

In [152]: data2 = pd.concat([s1, s2], keys=['one', 'two'])

In [153]: data2

Out[153]: one  a    0
          b    1
          c    2
          d    3
     two  c    4
          d    5
          e    6
     dtype: int64

In [154]: data2.unstack()
```

Out[154]:

	a	b	c	d	e
one	0.0	1.0	2.0	3.0	NaN
two	NaN	NaN	4.0	5.0	6.0

- Stacking filters out missing data by default.

```
In [155]: data2.unstack()
```

Out[155]:

	a	b	c	d	e
one	0.0	1.0	2.0	3.0	NaN
two	NaN	NaN	4.0	5.0	6.0

```
In [156]: data2.unstack().stack()
```

```
Out[156]: one  a    0.0
          b    1.0
          c    2.0
          d    3.0
          two  c    4.0
          d    5.0
          e    6.0
          dtype: float64
```

```
In [157]: data2.unstack().stack(dropna=False)
```

```
Out[157]: one  a    0.0
          b    1.0
          c    2.0
          d    3.0
          e    NaN
          two  a    NaN
          b    NaN
          c    4.0
          d    5.0
          e    6.0
          dtype: float64
```

- When you unstack in a DataFrame, the level unstacked becomes the lowest level in the result:

```
In [158]: df = pd.DataFrame({'left': result, 'right': result + 5},
                            columns=pd.Index(['left', 'right'], name='side'))

In [159]: df
```

Out[159]:

side		left	right
state	number		
Ohio	one	0	5
	two	1	6
	three	2	7
Colorado	one	3	8
	two	4	9
	three	5	10

```
In [160]: df.unstack('state')
```

Out[160]:

side	left		right	
state	Ohio	Colorado	Ohio	Colorado
number				
one	0	3	5	8
two	1	4	6	9
three	2	5	7	10

- When calling `stack`, we can indicate the name of the axis to stack:

```
In [160]: df.unstack('state')
```

Out[160]:

side	left		right	
state	Ohio	Colorado	Ohio	Colorado
number				
one	0	3	5	8
two	1	4	6	9
three	2	5	7	10

```
In [161]: df.unstack('state').stack('side')
```

Out[161]:

	state	Colorado	Ohio
number	side		
one	left	3	0
	right	8	5
two	left	4	1
	right	9	6
three	left	5	2
	right	10	7

Pivoting “Long” to “Wide” Format

- A common way to store multiple time series in databases and CSV is in so-called *long* or *stacked* format.

In [162]:
```
data = pd.read_csv('examples/macrodata.csv')
```

In [163]:
```
data.head()
```

Out[163]:

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	unemp	pop	infl	realint
0	1959.0	1.0	2710.349	1707.4	286.898	470.045	1886.9	28.98	139.7	2.82	5.8	177.146	0.00	0.00
1	1959.0	2.0	2778.801	1733.7	310.859	481.301	1919.7	29.15	141.7	3.08	5.1	177.830	2.34	0.74
2	1959.0	3.0	2775.488	1751.8	289.226	491.260	1916.4	29.35	140.5	3.82	5.3	178.657	2.74	1.09
3	1959.0	4.0	2785.204	1753.7	299.356	484.052	1931.3	29.37	140.0	4.33	5.6	179.386	0.27	4.06
4	1960.0	1.0	2847.699	1770.5	331.722	462.199	1955.5	29.54	139.6	3.50	5.2	180.007	2.31	1.19

In [164]:
```
periods = pd.PeriodIndex(year=data.year, quarter=data.quarter,
                         name='date')
```

In [165]:
```
columns = pd.Index(['realgdp', 'infl', 'unemp'], name='item')
```

In [166]:
```
data = data.reindex(columns=columns)
```

In [167]:
```
data.index = periods.to_timestamp('D', 'end')
```

In [168]:
```
ldata = data.stack().reset_index().rename(columns={0: 'value'})
```

```
In [169]: ldata[:10]
```

Out[169]:

	date	item	value
0	1959-03-31 23:59:59.999999999	realgdp	2710.349
1	1959-03-31 23:59:59.999999999	infl	0.000
2	1959-03-31 23:59:59.999999999	unemp	5.800
3	1959-06-30 23:59:59.999999999	realgdp	2778.801
4	1959-06-30 23:59:59.999999999	infl	2.340
5	1959-06-30 23:59:59.999999999	unemp	5.100
6	1959-09-30 23:59:59.999999999	realgdp	2775.488
7	1959-09-30 23:59:59.999999999	infl	2.740
8	1959-09-30 23:59:59.999999999	unemp	5.300
9	1959-12-31 23:59:59.999999999	realgdp	2785.204

- This is the so-called *long* format for multiple time series, or other observational data with two or more keys (here, our keys are date and item).
- Each row in the table represents a single observation.

- In some cases, the data may be more difficult to work with in this format; you might prefer to have a DataFrame containing one column per distinct `item` value indexed by timestamps in the `date` column.
- DataFrame's `pivot` method performs exactly this transformation:

```
In [170]: pivoted = ldata.pivot('date', 'item', 'value')

In [171]: pivoted
Out[171]:
```

item	infl	realgdp	unemp
date			
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2
1960-06-30 23:59:59.999999999	0.14	2834.390	5.2
1960-09-30 23:59:59.999999999	2.70	2839.022	5.6
1960-12-31 23:59:59.999999999	1.21	2802.616	6.3
1961-03-31 23:59:59.999999999	-0.40	2819.264	6.8
1961-06-30 23:59:59.999999999	1.47	2872.005	7.0
...	...	...	...
2007-06-30 23:59:59.999999999	2.75	13203.977	4.5
2007-09-30 23:59:59.999999999	3.45	13321.109	4.7
2007-12-31 23:59:59.999999999	6.38	13391.249	4.8
2008-03-31 23:59:59.999999999	2.82	13366.865	4.9
2008-06-30 23:59:59.999999999	8.53	13415.266	5.4
2008-09-30 23:59:59.999999999	-3.16	13324.600	6.0
2008-12-31 23:59:59.999999999	-8.79	13141.920	6.9
2009-03-31 23:59:59.999999999	0.94	12925.410	8.1
2009-06-30 23:59:59.999999999	3.37	12901.504	9.2
2009-09-30 23:59:59.999999999	3.56	12990.341	9.6

203 rows × 3 columns

- Suppose you had two value columns that you wanted to reshape simultaneously:

```
In [172]: ldata['value2'] = np.random.randn(len(ldata))

In [173]: ldata[:10]
```

Out[173]:

	date	item	value	value2
0	1959-03-31 23:59:59.999999999	realgdp	2710.349	0.523772
1	1959-03-31 23:59:59.999999999	infl	0.000	0.000940
2	1959-03-31 23:59:59.999999999	unemp	5.800	1.343810
3	1959-06-30 23:59:59.999999999	realgdp	2778.801	-0.713544
4	1959-06-30 23:59:59.999999999	infl	2.340	-0.831154
5	1959-06-30 23:59:59.999999999	unemp	5.100	-2.370232
6	1959-09-30 23:59:59.999999999	realgdp	2775.488	-1.860761
7	1959-09-30 23:59:59.999999999	infl	2.740	-0.860757
8	1959-09-30 23:59:59.999999999	unemp	5.300	0.560145
9	1959-12-31 23:59:59.999999999	realgdp	2785.204	-1.265934

- By omitting the last argument, you obtain a DataFrame with hierarchical columns:

```
In [174]: pivoted = ldata.pivot('date', 'item')

In [175]: pivoted[:5]
```

Out[175]:

	value			value2		
item	infl	realgdp	unemp	infl	realgdp	unemp
date						
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	0.000940	0.523772	1.343810
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	-0.831154	-0.713544	-2.370232
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	-0.860757	-1.860761	0.560145
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	0.119827	-1.265934	-1.063512
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	-2.359419	0.332883	-0.199543

```
In [176]: pivoted['value'][:5]
```

Out[176]:

item	infl	realgdp	unemp
date			
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2

- Note that `pivot` is equivalent to creating a hierarchical index using `set_index` followed by a call to `unstack`:

```
In [177]: unstacked = ldata.set_index(['date', 'item']).unstack('item')

In [178]: unstacked[:7]
```

Out[178]:

	value			value2		
item	infl	realgdp	unemp	infl	realgdp	unemp
date						
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	0.000940	0.523772	1.343810
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	-0.831154	-0.713544	-2.370232
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	-0.860757	-1.860761	0.560145
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	0.119827	-1.265934	-1.063512
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	-2.359419	0.332883	-0.199543
1960-06-30 23:59:59.999999999	0.14	2834.390	5.2	-0.970736	-1.541996	-1.307030
1960-09-30 23:59:59.999999999	2.70	2839.022	5.6	0.377984	0.286350	-0.753887

Pivoting “Wide” to “Long” Format

- An inverse operation to `pivot` for DataFrames is `pandas.melt`.
- Rather than transforming one column into many in a new DataFrame, it merges multiple columns into one, producing a DataFrame that is longer than the input.

```
In [179]: df = pd.DataFrame({'key': ['foo', 'bar', 'baz'],
                             'A': [1, 2, 3],
                             'B': [4, 5, 6],
                             'C': [7, 8, 9]})

In [180]: df
```

Out[180]:

	key	A	B	C
0	foo	1	4	7
1	bar	2	5	8
2	baz	3	6	9

- The `'key'` column may be a group indicator, and the other columns are data values.

- When using `pandas.melt`, we must indicate which columns (if any) are group indicators.
- Let's use `'key'` as the only group indicator here:

```
In [181]: melted = pd.melt(df, ['key'])

In [182]: melted
```

Out[182]:

	key	variable	value
0	foo	A	1
1	bar	A	2
2	baz	A	3
3	foo	B	4
4	bar	B	5
5	baz	B	6
6	foo	C	7
7	bar	C	8
8	baz	C	9

- Using `pivot`, we can reshape back to the original layout:

```
In [183]: reshaped = melted.pivot('key', 'variable', 'value')

In [184]: reshaped
```

Out[184]:

variable	A	B	C
key			
bar	2	5	8
baz	3	6	9
foo	1	4	7

- Since the result of `pivot` creates an index from the column used as the row labels, we may want to use `reset_index` to move the data back into a column:

```
In [185]: reshaped.reset_index()
```

Out[185]:

variable	key	A	B	C
0	bar	2	5	8
1	baz	3	6	9
2	foo	1	4	7

- You can also specify a subset of columns to use as value columns:

```
In [186]: pd.melt(df, id_vars=['key'], value_vars=['A', 'B'])
```

Out[186]:

	key	variable	value
0	foo	A	1
1	bar	A	2
2	baz	A	3
3	foo	B	4
4	bar	B	5
5	baz	B	6

- `pandas.melt` can be used without any group identifiers, too:

```
In [187]: pd.melt(df, value_vars=['A', 'B', 'C'])
```

Out[187]:

	variable	value
0	A	1
1	A	2
2	A	3
3	B	4
4	B	5
5	B	6
6	C	7
7	C	8
8	C	9

```
In [188]: pd.melt(df, value_vars=['key', 'A', 'B'])
```

Out[188]:

	variable	value
0	key	foo
1	key	bar
2	key	baz
3	A	1
4	A	2
5	A	3
6	B	4
7	B	5
8	B	6