Pivot Tables by Hand

To start learning more about this data, we might begin by grouping it according to gender, survival status, or some combination thereof. If you have read the previous section, you might be tempted to apply a `GroupBy` operation—for example, let's look at survival rate by gender:

```
In[3]: titanic.groupby('sex')[['survived']].mean()
```

```
Out[3]:        survived
        sex
        female  0.742038
        male    0.188908
```

This immediately gives us some insight: overall, three of every four females on board survived, while only one in five males survived!

This is useful, but we might like to go one step deeper and look at survival by both sex and, say, class. Using the vocabulary of `GroupBy`, we might proceed using something like this: we *group by* class and gender, *select* survival, *apply* a mean aggregate, *combine* the resulting groups, and then *unstack* the hierarchical index to reveal the hidden multidimensionality. In code:

```
In[4]: titanic.groupby(['sex', 'class'])['survived'].aggregate('mean').unstack()
```

```
Out[4]: class      First    Second     Third
        sex
        female  0.968085  0.921053  0.500000
        male    0.368852  0.157407  0.135447
```

This gives us a better idea of how both gender and class affected survival, but the code is starting to look a bit garbled. While each step of this pipeline makes sense in light of the tools we've previously discussed, the long string of code is not particularly easy to read or use. This two-dimensional `GroupBy` is common enough that Pandas includes a convenience routine, `pivot_table`, which succinctly handles this type of multidimensional aggregation.

Pivot Table Syntax

Here is the equivalent to the preceding operation using the `pivot_table` method of `DataFrame`s:

```
In[5]: titanic.pivot_table('survived', index='sex', columns='class')
```

```
Out[5]: class      First    Second     Third
        sex
        female  0.968085  0.921053  0.500000
        male    0.368852  0.157407  0.135447
```

This is eminently more readable than the `GroupBy` approach, and produces the same result. As you might expect of an early 20th-century transatlantic cruise, the survival gradient favors both women and higher classes. First-class women survived with near certainty (hi, Rose!), while only one in ten third-class men survived (sorry, Jack!).

Multilevel pivot tables

Just as in the `GroupBy`, the grouping in pivot tables can be specified with multiple levels, and via a number of options. For example, we might be interested in looking at age as a third dimension. We'll bin the age using the `pd.cut` function:

```
In[6]: age = pd.cut(titanic['age'], [0, 18, 80])
       titanic.pivot_table('survived', ['sex', age], 'class')

Out[6]:   class               First    Second     Third
          sex    age
          female (0, 18]   0.909091  1.000000  0.511628
                 (18, 80]  0.972973  0.900000  0.423729
          male   (0, 18]   0.800000  0.600000  0.215686
                 (18, 80]  0.375000  0.071429  0.133663
```

We can apply this same strategy when working with the columns as well; let's add info on the fare paid using `pd.qcut` to automatically compute quantiles:

```
In[7]: fare = pd.qcut(titanic['fare'], 2)
       titanic.pivot_table('survived', ['sex', age], [fare, 'class'])

Out[7]:
fare            [0, 14.454]
class                 First    Second     Third      \\
sex    age
female (0, 18]          NaN  1.000000  0.714286
       (18, 80]         NaN  0.880000  0.444444
male   (0, 18]          NaN  0.000000  0.260870
       (18, 80]         0.0  0.098039  0.125000

fare            (14.454, 512.329]
class                 First    Second     Third
sex    age
female (0, 18]     0.909091  1.000000  0.318182
       (18, 80]    0.972973  0.914286  0.391304
male   (0, 18]     0.800000  0.818182  0.178571
       (18, 80]    0.391304  0.030303  0.192308
```

The result is a four-dimensional aggregation with hierarchical indices (see "Hierarchical Indexing" on page 128), shown in a grid demonstrating the relationship between the values.

Additional pivot table options

The full call signature of the `pivot_table` method of `DataFrame`s is as follows:

```
# call signature as of Pandas 0.18
DataFrame.pivot_table(data, values=None, index=None, columns=None,
                      aggfunc='mean', fill_value=None, margins=False,
                      dropna=True, margins_name='All')
```

We've already seen examples of the first three arguments; here we'll take a quick look at the remaining ones. Two of the options, `fill_value` and `dropna`, have to do with missing data and are fairly straightforward; we will not show examples of them here.

The `aggfunc` keyword controls what type of aggregation is applied, which is a mean by default. As in the `GroupBy`, the aggregation specification can be a string representing one of several common choices (`'sum'`, `'mean'`, `'count'`, `'min'`, `'max'`, etc.) or a function that implements an aggregation (`np.sum()`, `min()`, `sum()`, etc.). Additionally, it can be specified as a dictionary mapping a column to any of the above desired options:

```
In[8]: titanic.pivot_table(index='sex', columns='class',
                           aggfunc={'survived':sum, 'fare':'mean'})
```

```
Out[8]:              fare                          survived
        class        First     Second      Third    First Second Third
        sex
        female  106.125798  21.970121  16.118810     91.0   70.0  72.0
        male     67.226127  19.741782  12.661633     45.0   17.0  47.0
```

Notice also here that we've omitted the `values` keyword; when you're specifying a mapping for `aggfunc`, this is determined automatically.

At times it's useful to compute totals along each grouping. This can be done via the `margins` keyword:

```
In[9]: titanic.pivot_table('survived', index='sex', columns='class', margins=True)
```

```
Out[9]: class      First    Second     Third       All
        sex
        female  0.968085  0.921053  0.500000  0.742038
        male    0.368852  0.157407  0.135447  0.188908
        All     0.629630  0.472826  0.242363  0.383838
```

Here this automatically gives us information about the class-agnostic survival rate by gender, the gender-agnostic survival rate by class, and the overall survival rate of 38%. The margin label can be specified with the `margins_name` keyword, which defaults to `"All"`.

Example: Birthrate Data

As a more interesting example, let's take a look at the freely available data on births in the United States, provided by the Centers for Disease Control (CDC). This data can be found at *https://raw.githubusercontent.com/jakevdp/data-CDCbirths/master/births.csv* (this dataset has been analyzed rather extensively by Andrew Gelman and his group; see, for example, this blog post):

```
In[10]:
# shell command to download the data:
# !curl -O https://raw.githubusercontent.com/jakevdp/data-CDCbirths/
# master/births.csv
```

```
In[11]: births = pd.read_csv('births.csv')
```

Taking a look at the data, we see that it's relatively simple—it contains the number of births grouped by date and gender:

```
In[12]: births.head()
```

```
Out[12]:   year  month day gender  births
         0  1969      1   1      F    4046
         1  1969      1   1      M    4440
         2  1969      1   2      F    4454
         3  1969      1   2      M    4548
         4  1969      1   3      F    4548
```

We can start to understand this data a bit more by using a pivot table. Let's add a decade column, and take a look at male and female births as a function of decade:

```
In[13]:
births['decade'] = 10 * (births['year'] // 10)
births.pivot_table('births', index='decade', columns='gender', aggfunc='sum')
```

```
Out[13]: gender         F         M
         decade
         1960     1753634   1846572
         1970    16263075  17121550
         1980    18310351  19243452
         1990    19479454  20420553
         2000    18229309  19106428
```

We immediately see that male births outnumber female births in every decade. To see this trend a bit more clearly, we can use the built-in plotting tools in Pandas to visualize the total number of births by year (Figure 3-2; see Chapter 4 for a discussion of plotting with Matplotlib):

```
In[14]:
%matplotlib inline
import matplotlib.pyplot as plt
sns.set()  # use Seaborn styles
births.pivot_table('births', index='year', columns='gender', aggfunc='sum').plot()
plt.ylabel('total births per year');
```