Data Aggregation and Group Operations

Part 4

Apply: General split-apply-combine

Part 2

Example: Group Weighted Average and Correlation

 Under the split-apply-combine paradigm of groupby, operations between columns in a DataFrame or two Series, such as a group weighted average, are possible.

```
In [68]: df = pd.DataFrame({'category': ['a', 'a', 'a', 'a',
                                             'b', 'b', 'b', 'b'],
                               'data': np.random.randn(8),
                               'weights': np.random.rand(8)})
          df
Out[68]:
             category
                          data weights
                   a 1.561587 0.957515
          1
                   a 1.219984 0.347267
                   a -0.482239 0.581362
                   a 0.315667 0.217091
                   b -0.047852 0.894406
                   b -0.454145 0.918564
                   b -0.556774 0.277825
          7
                   b 0.253321 0.955905
```

• The group weighted average by category would then be:

```
In [69]: grouped = df.groupby('category')
   get_wavg = lambda g: np.average(g['data'], weights=g['weights'])
   grouped.apply(get_wavg)

Out[69]: category
   a    0.811643
   b   -0.122262
   dtype: float64
```

 As another example, consider a financial dataset originally obtained from Yahoo! Finance containing end-of-day prices for a few stocks and the S&P 500 index (the SPX symbol):

```
In [70]: close px = pd.read csv('examples/stock px 2.csv', parse dates=True,
                                 index col=0)
In [71]: close px.info()
         <class 'pandas.core.frame.DataFrame'>
         DatetimeIndex: 2214 entries, 2003-01-02 to 2011-10-14
         Data columns (total 4 columns):
                2214 non-null float64
                 2214 non-null float64
                 2214 non-null float64
                 2214 non-null float64
         dtypes: float64(4)
         memory usage: 86.5 KB
In [72]: close px[-4:]
Out[72]:
                     AAPL MSFT XOM
                                         SPX
          2011-10-11 400.29 27.00 76.27 1195.54
          2011-10-12 402.19 26.96 77.16 1207.25
          2011-10-13 408.43 27.18 76.37 1203.66
          2011-10-14 422.00 27.27 78.11 1224.58
```

- One task of interest might be to compute a DataFrame consisting of the yearly correlations of daily returns (computed from percent changes) with SPX.
- As one way to do this, we first create a function that computes the pairwise correlation of each column with the 'SPX' column:

```
In [73]: spx_corr = lambda x: x.corrwith(x['SPX'])
```

• Next, we compute percent change on close_px using pct change:

```
In [74]: rets = close_px.pct_change().dropna()
```

 Lastly, we group these percent changes by year, which can be extracted from each row label with a one-line function that returns the year attribute of each datetime label:



- You could also compute inter-column correlations.
- Here we compute the annual correlation between Apple and Microsoft:

```
In [76]: by year.apply(lambda g: g['AAPL'].corr(g['MSFT']))
Out[76]: 2003
                 0.480868
         2004
                 0.259024
         2005
                 0.300093
         2006
                 0.161735
         2007
                 0.417738
                 0.611901
         2008
         2009
                 0.432738
         2010
                 0.571946
         2011
                 0.581987
         dtype: float64
```

Example: Group-Wise Linear Regression

- In the same theme as the previous example, you can use groupby to perform more complex group-wise statistical analysis, as long as the function returns a pandas object or scalar value.
- For example, we can define the following regress function (using the statsmodels econometrics library), which executes an ordinary least squares (OLS) regression on each chunk of data:

```
In [77]: import statsmodels.api as sm
def regress(data, yvar, xvars):
    Y = data[yvar]
    X = data[xvars]
    X['intercept'] = 1.
    result = sm.OLS(Y, X).fit()
    return result.params
```

 Now, to run a yearly linear regression of AAPL on SPX returns, execute:

In [78]:	by_ye	ar.apply	(regress
Out[78]:			
		SPX	intercept
	2003	1.195406	0.000710
	2004	1.363463	0.004201
	2005	1.766415	0.003246
	2006	1.645496	0.000080
	2007	1.198761	0.003438
	2008	0.968016	-0.001110
	2009	0.879103	0.002954
	2010	1.052608	0.001261
	2011	0.806605	0.001514

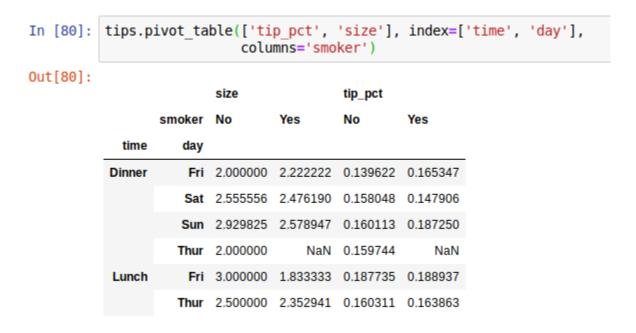
Pivot Tables and Cross-Tabulation

- A *pivot table* is a data summarization tool frequently found in spreadsheet programs and other data analysis software.
- It aggregates a table of data by one or more keys, arranging the data in a rectangle with some of the group keys along the rows and some along the columns.
- Pivot tables in Python with pandas are made possible through the groupby facility combined with reshape operations utilizing hierarchical indexing.
- DataFrame has a pivot_table method, and there is also a top-level pandas.pivot table function.
- In addition to providing a convenience interface to groupby, pivot_table can add partial totals, also known as margins.

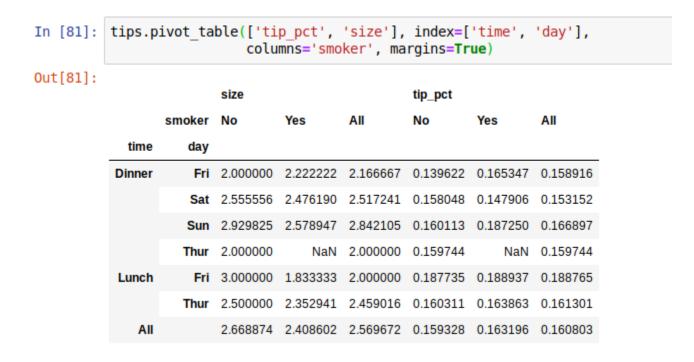
• Returning to the tipping dataset, suppose you wanted to compute a table of group means (the default pivot_table aggregation type) arranged by day and smoker on the rows:

In [79]:	<pre>tips.pivot_table(index=['day', 'smoker'])</pre>							
Out[79]:			size	tip	tip_pct	total_bill		
	day	smoker						
	Fri	No	2.250000	2.812500	0.151650	18.420000		
		Yes	2.066667	2.714000	0.174783	16.813333		
	Sat	No	2.555556	3.102889	0.158048	19.661778		
		Yes	2.476190	2.875476	0.147906	21.276667		
	Sun	No	2.929825	3.167895	0.160113	20.506667		
		Yes	2.578947	3.516842	0.187250	24.120000		
	Thur	No	2.488889	2.673778	0.160298	17.113111		
		Yes	2.352941	3.030000	0.163863	19.190588		

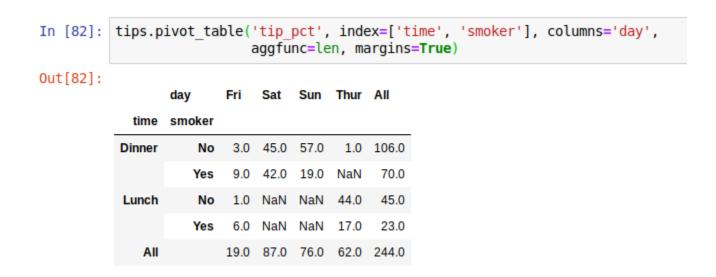
- Now, suppose we want to aggregate only tip_pct and size, and additionally group by time.
- We'll put smoker in the table columns and day in the rows:



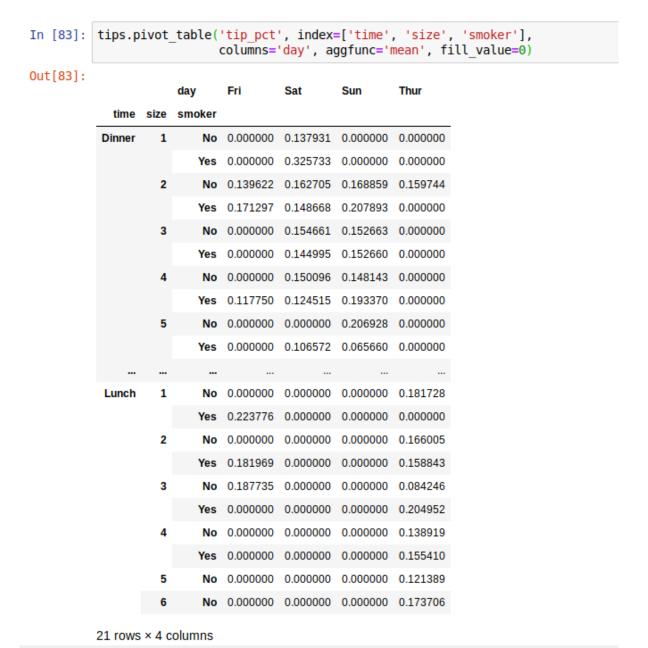
- We could augment this table to include partial totals by passing margins=True.
- This has the effect of adding All row and column labels, with corresponding values being the group statistics for all the data within a single tier:



- To use a different aggregation function, pass it to aggfunc.
- For example, 'count' or len will give you a cross-tabulation (count or frequency) of group sizes:

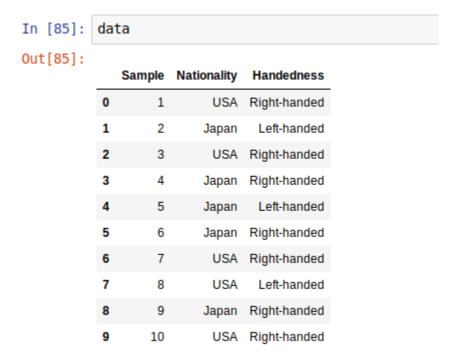


• If some combinations are empty (or otherwise NA), you may wish to pass a fill value:

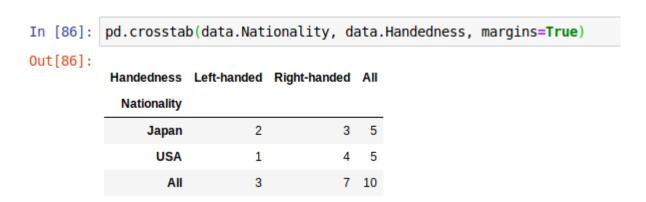


Cross-Tabulations: Crosstab

- A cross-tabulation (or crosstab for short) is a special case of a pivot table that computes group frequencies.
- Here is an example:



- As part of some survey analysis, we might want to summarize this data by nationality and handedness.
- You could use pivot_table to do this, but the pandas.crosstab function can be more convenient:



- The first two arguments to crosstab can each either be an array or Series or a list of arrays.
- As in the tips data:

