DSC 201: Data Analysis & Visualization

Review

Dr. David Koop
Can use time as an **index**

```python
data = [('2017-11-30', 48),
        ('2017-12-02', 45),
        ('2017-12-03', 44),
        ('2017-12-04', 48)]
dates, temps = zip(*data)
s = pd.Series(temps, pd.to_datetime(dates))
```

Accessing a particular time or checking equivalence allows any string that can be interpreted as a date:

- `s['12/04/2017']` or `s['20171204']`

Using a less specific string will get all matching data:

- `s['2017-12']` returns the three December entries

Time slices do not need to exist:

- `s['2017-12-01':'2017-12-31']`
Timedelta

- Compute differences between dates
- Lives in `datetime` module
- `diff = parse_date("1 Jan 2017") - datetime.now().date() 
diff.days`
- Also a `pd.Timedelta` object that take strings:
  - `datetime.now().date() + pd.Timedelta("4 days")`
- Also, Roll dates using anchored offsets

```
from pandas.tseries.offsets import Day, MonthEnd
now = datetime(2011, 11, 17)
In [107]: now + MonthEnd(2)
Out[107]: Timestamp('2011-12-31 00:00:00')
```
Time Zones

• Coordinated Universal Time (UTC) is the standard time (basically equivalent to Greenwich Mean Time (GMT))

• Other time zones are UTC +/- a number in [1,12]

• Dartmouth is UTC-5 (aka US/Eastern)

• The pytz module keeps track of all of the time zone parameters
  - even Daylight Savings Time

• Localize a timestamp using `tz_localize`
  - `ts = pd.Timestamp("1 Dec 2016 12:30 PM")`
    `ts = ts.tz_localize("US/Eastern")`

• Convert a timestamp using `tz_convert`
  - `ts.tz_convert("Europe/Budapest")`

• Operations involving timestamps from different time zones become UTC
Frequency

• Generic time series in pandas are **irregular**
  - there is no fixed frequency
  - we don't necessarily have data for every day/hour/etc.

• Date ranges have frequency

```
In [76]: pd.date_range(start='2012-04-01', periods=20)
Out[76]:
DatetimeIndex(['2012-04-01', '2012-04-02', '2012-04-03', '2012-04-04',
              '2012-04-05', '2012-04-06', '2012-04-07', '2012-04-08',
              '2012-04-09', '2012-04-10', '2012-04-11', '2012-04-12',
              '2012-04-17', '2012-04-18', '2012-04-19', '2012-04-20'],
    dtype='datetime64[ns]', freq='D')
```
Shifting Data

• Leading or Lagging Data

In [95]: ts = Series(np.random.randn(4),
   ....:       index=pd.date_range('1/1/2000', periods=4, freq='M'))

In [96]: ts
Out[96]:
2000-01-31   -0.066748
2000-02-29    0.838639
2000-03-31   -0.117388
2000-04-30   -0.517795
Freq: M, dtype: float64

In [97]: ts.shift(2)
Out[97]:
2000-03-31   -0.066748
2000-04-30    0.838639
2000-05-31   -0.117388
2000-06-30   -0.517795
Freq: M, dtype: float64

In [98]: ts.shift(-2)
Out[98]:
2000-01-31 01:30:00   -0.066748
2000-02-29 01:30:00    0.838639
2000-03-31 01:30:00   -0.117388
2000-04-30 01:30:00   -0.517795
Freq: M, dtype: float64

• Shifting by time:

In [99]: ts.shift(2, freq='M')
Out[99]:
2000-03-31   -0.066748
2000-04-30    0.838639
2000-05-31   -0.117388
2000-06-30   -0.517795
Freq: M, dtype: float64

[W. McKinney, Python for Data Analysis]
Resampling

• Could be
  - downsample: higher frequency to lower frequency
  - upsample: lower frequency to higher frequency
  - neither: e.g. Wednesdays to Fridays

• resample method: e.g. `ts.resample('M').mean()`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>String or DateOffset indicating desired resampled frequency (e.g., 'M', '5min', or <code>Second(15)</code>)</td>
</tr>
<tr>
<td>axis</td>
<td>Axis to resample on; default axis=0</td>
</tr>
<tr>
<td>fill_method</td>
<td>How to interpolate when upsampling, as in 'ffill' or 'bfill'; by default does no interpolation</td>
</tr>
<tr>
<td>closed</td>
<td>In downsampling, which end of each interval is closed (inclusive), 'right' or 'left'</td>
</tr>
<tr>
<td>label</td>
<td>In downsampling, how to label the aggregated result, with the 'right' or 'left' bin edge (e.g., the 9:30 to 9:35 five-minute interval could be labeled 9:30 or 9:35)</td>
</tr>
<tr>
<td>loffset</td>
<td>Time adjustment to the bin labels, such as '-1s' / <code>Second(-1)</code> to shift the aggregate labels one second earlier</td>
</tr>
<tr>
<td>limit</td>
<td>When forward or backward filling, the maximum number of periods to fill</td>
</tr>
<tr>
<td>kind</td>
<td>Aggregate to periods ('period') or timestamps ('timestamp'); defaults to the type of index the time series has</td>
</tr>
<tr>
<td>convention</td>
<td>When resampling periods, the convention ('start' or 'end') for converting the low-frequency period to high frequency; defaults to 'end'</td>
</tr>
</tbody>
</table>
Downsampling

• Need to define **bin edges** which are used to group the time series into **intervals** that can be aggregated

• Remember:
  - Which side of the interval is closed
  - How to label the aggregated bin (start or end of interval)

```
closed='left'  9:00  9:01  9:02  9:03  9:04  9:05
```

```
closed='right' 9:00  9:01  9:02  9:03  9:04  9:05
```

You also could have accomplished the effect of `loffset` by calling the `shift` method on the result without the `loffset`.

Open-High-Low-Close (OHLC) resampling

In finance, a popular way to aggregate a time series is to compute four values for each bucket: the first (open), last (close), maximum (high), and minimal (low) values. By using the `ohlc` aggregate function you will obtain a DataFrame having columns containing these four aggregates, which are efficiently computed in a single sweep of the data:

```
In [220]: ts.resample('5min').ohlc()
Out[220]:
    open  high  low  close
00:00  9:00  9:00  9:00  9:00
00:05  9:05  9:05  9:05  9:05
00:10  9:10  9:10  9:10  9:10
```

11.6 Resampling and Frequency Conversion | 351
Upsampling

- No aggregation necessary

In [222]: frame
Out[222]:

<table>
<thead>
<tr>
<th></th>
<th>Colorado</th>
<th>Texas</th>
<th>New York</th>
<th>Ohio</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000-01-05</td>
<td>-0.896431</td>
<td>0.677263</td>
<td>0.036503</td>
<td>0.087102</td>
</tr>
<tr>
<td>2000-01-12</td>
<td>-0.046662</td>
<td>0.927238</td>
<td>0.482284</td>
<td>-0.867130</td>
</tr>
</tbody>
</table>

In [223]: df_daily = frame.resample('D').asfreq()

In [224]: df_daily
Out[224]:

<table>
<thead>
<tr>
<th></th>
<th>Colorado</th>
<th>Texas</th>
<th>New York</th>
<th>Ohio</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000-01-05</td>
<td>-0.896431</td>
<td>0.677263</td>
<td>0.036503</td>
<td>0.087102</td>
</tr>
<tr>
<td>2000-01-06</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2000-01-07</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2000-01-08</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2000-01-09</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2000-01-10</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2000-01-11</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2000-01-12</td>
<td>-0.046662</td>
<td>0.927238</td>
<td>0.482284</td>
<td>-0.867130</td>
</tr>
</tbody>
</table>

In [225]: frame.resample('D').ffill()
Out[225]:

<table>
<thead>
<tr>
<th></th>
<th>Colorado</th>
<th>Texas</th>
<th>New York</th>
<th>Ohio</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000-01-05</td>
<td>-0.896431</td>
<td>0.677263</td>
<td>0.036503</td>
<td>0.087102</td>
</tr>
<tr>
<td>2000-01-06</td>
<td>-0.896431</td>
<td>0.677263</td>
<td>0.036503</td>
<td>0.087102</td>
</tr>
<tr>
<td>2000-01-07</td>
<td>-0.896431</td>
<td>0.677263</td>
<td>0.036503</td>
<td>0.087102</td>
</tr>
<tr>
<td>2000-01-08</td>
<td>-0.896431</td>
<td>0.677263</td>
<td>0.036503</td>
<td>0.087102</td>
</tr>
<tr>
<td>2000-01-09</td>
<td>-0.896431</td>
<td>0.677263</td>
<td>0.036503</td>
<td>0.087102</td>
</tr>
<tr>
<td>2000-01-10</td>
<td>-0.896431</td>
<td>0.677263</td>
<td>0.036503</td>
<td>0.087102</td>
</tr>
<tr>
<td>2000-01-11</td>
<td>-0.896431</td>
<td>0.677263</td>
<td>0.036503</td>
<td>0.087102</td>
</tr>
<tr>
<td>2000-01-12</td>
<td>-0.046662</td>
<td>0.927238</td>
<td>0.482284</td>
<td>-0.867130</td>
</tr>
</tbody>
</table>
Interpolation

• Fill in the missing values with computed best estimates using various types of algorithms
• Apply after resample
Sales Data by Month
Resampled Sales Data (ffill)
Resampled with Linear Interpolation (Default)
Resampled with Cubic Interpolation
Piecewise Cubic Hermite Interpolating Polynomial
Window Functions

• Idea: want to aggregate over a window of time, calculate the answer, and then slide that window ahead. Repeat.

• rolling: smooth out data

• Specify the window size in rolling, then an aggregation method

• Result is set to the right edge of window (change with center=True)

• Example:
  - df.rolling('180D').mean()
  - df.rolling('90D').sum()
90-Day Rolling Window (Mean)
180-Day Rolling Window (Mean)
Assignment 5

- Aggregation, Time Series, and Visualization
- Use New York City Subway Turnstile Data
Final Exam

- Course [web page](#) has details
- Wednesday, Dec. 13, 11:30am-2:30pm
- LARTS 201
- Covers all material in the course
- More focus on the last few topics since Test 2
Questions?
Review

• Exploratory Data Analysis
  - Univariate (one attribute)
  - Multivariate (2+ attributes)
  - Non-graphical ~ statistics
  - Graphical ~ visualizations
Common Distributions

- Uniform
- Bernoulli
- Hypergeometric
- Binomial
- Geometric
- Negative Binomial
- Poisson
- Exponential
- Log Normal
- Normal (Gaussian)
- Chi-Squared
- Weibull
- Student’s t
- Gamma
- Beta
Scatterplots and Correlation

- Strong positive correlation
- Moderate positive correlation
- No correlation
- Moderate negative correlation
- Strong negative correlation
- Curvilinear relationship
Review

- Visualization:
  - Why?
  - Tools: matplotlib, Tableau, and more
  - Types of data: categorical, ordinal, quantitative
  - Encoding data: marks & channels
  - Expressiveness and Effectiveness
  - Visual encodings: bar charts, scatterplots, line charts, etc.
  - Color and colormaps
  - Interaction and multiple views
  - Maps
Categorial, Ordinal, and Quantitative

<table>
<thead>
<tr>
<th>Order ID</th>
<th>Order Date</th>
<th>Order Priority</th>
<th>Product Container</th>
<th>Product Base Margin</th>
<th>Ship Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10/14/06</td>
<td>5-Low</td>
<td>Large Box</td>
<td>0.8</td>
<td>10/21/06</td>
</tr>
<tr>
<td>6</td>
<td>2/21/08</td>
<td>4-Not Specified</td>
<td>Small Pack</td>
<td>0.55</td>
<td>2/22/08</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Small Pack</td>
<td>0.79</td>
<td>7/17/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Jumbo Box</td>
<td>0.72</td>
<td>7/17/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.6</td>
<td>7/18/07</td>
</tr>
<tr>
<td>32</td>
<td>7/16/07</td>
<td>2-High</td>
<td>Medium Box</td>
<td>0.65</td>
<td>7/18/07</td>
</tr>
<tr>
<td>35</td>
<td>10/23/07</td>
<td>4-Not Specified</td>
<td>Wrap Bag</td>
<td>0.52</td>
<td>10/24/07</td>
</tr>
<tr>
<td>35</td>
<td>10/23/07</td>
<td>4-Not Specified</td>
<td>Small Box</td>
<td>0.58</td>
<td>10/25/07</td>
</tr>
<tr>
<td>36</td>
<td>11/3/07</td>
<td>1-Urgent</td>
<td>Small Box</td>
<td>0.55</td>
<td>11/3/07</td>
</tr>
<tr>
<td>65</td>
<td>3/18/07</td>
<td>1-Urgent</td>
<td>Small Pack</td>
<td>0.49</td>
<td>3/19/07</td>
</tr>
<tr>
<td>66</td>
<td>1/20/05</td>
<td>5-Low</td>
<td>Wrap Bag</td>
<td>0.56</td>
<td>1/20/05</td>
</tr>
<tr>
<td>69</td>
<td>6/4/05</td>
<td>4-Not Specified</td>
<td>Small Pack</td>
<td>0.44</td>
<td>6/6/05</td>
</tr>
<tr>
<td>69</td>
<td>6/4/05</td>
<td>4-Not Specified</td>
<td>Small Pack</td>
<td>0.6</td>
<td>6/6/05</td>
</tr>
<tr>
<td>70</td>
<td>12/18/06</td>
<td>5-Low</td>
<td>0.59</td>
<td>12/23/06</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>12/18/06</td>
<td>5-Low</td>
<td>0.82</td>
<td>12/23/06</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>4/17/05</td>
<td>2-High</td>
<td>0.55</td>
<td>4/19/05</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>1/29/06</td>
<td>3-Medium</td>
<td>0.38</td>
<td>1/30/06</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>11/19/08</td>
<td>5-Low</td>
<td>0.37</td>
<td>11/28/08</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>0.37</td>
<td>5/9/08</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>0.38</td>
<td>5/10/08</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>5/8/08</td>
<td>2-High</td>
<td>0.6</td>
<td>5/11/08</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>6/11/06</td>
<td>3-Medium</td>
<td>0.6</td>
<td>6/12/06</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>6/11/06</td>
<td>3-Medium</td>
<td>0.38</td>
<td>6/14/06</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>5/1/08</td>
<td>4-Not Specified</td>
<td>0.82</td>
<td>5/3/08</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>10/21/07</td>
<td>4-Not Specified</td>
<td>Large Box</td>
<td>0.64</td>
<td>10/23/07</td>
</tr>
<tr>
<td>166</td>
<td>9/12/07</td>
<td>2-High</td>
<td>0.55</td>
<td>9/14/07</td>
<td></td>
</tr>
<tr>
<td>193</td>
<td>8/8/06</td>
<td>1-Urgent</td>
<td>0.57</td>
<td>8/10/06</td>
<td></td>
</tr>
<tr>
<td>194</td>
<td>4/5/08</td>
<td>3-Medium</td>
<td>0.42</td>
<td>4/7/08</td>
<td></td>
</tr>
</tbody>
</table>

quantitative ordinal categorical
Visual Encoding

- How do we encode data visually?
  - **Marks** are the basic graphical elements in a visualization
  - **Channels** are ways to control the appearance of the marks
- Marks classified by dimensionality:
  - **Points**
  - **Lines**
  - **Areas**
- Also can have surfaces, volumes
- Think of marks as a mathematical definition, or if familiar with tools like Adobe Illustrator or Inkscape, the path & point definitions
Visual Channels by Effectiveness

**Channels:** Expressiveness Types and Effectiveness Ranks

- **Magnitude Channels:** Ordered Attributes
  - Position on common scale
  - Position on unaligned scale
  - Length (1D size)
  - Tilt/angle
  - Area (2D size)
  - Depth (3D position)
  - Color luminance
  - Color saturation
  - Curvature
  - Volume (3D size)

- **Identity Channels:** Categorical Attributes
  - Spatial region
  - Color hue
  - Motion
  - Shape

[Munzner (ill. Maguire), 2014]
Color and Colormaps

- A colormap specifies a mapping between colors and data values
- Channels: hue, saturation, and luminance
- Luminance perception is non-linear…
- Issues with rainbow colormaps
- Segmented vs. continuous
- Univariate and bivariate
Review

• Python
  - Notebooks
  - Types
  - Variables
  - Functions
  - Lists & Tuples (also mutable vs. immutable)
  - Dictionaries & Sets
  - Classes
Review

- NumPy
  - Arrays
  - Why?
- Pandas
  - Series
  - Index
  - Data Frames
- Data
  - Formats: CSV, TSV, JSON, XML
  - Reading data from files
Review

• Data Wrangling:
  - Cleaning
  - Transforming
  - Reshaping
  - Merging
  - Tidy Data
Reshaping

- **stack**: pivots from the columns into rows (may produce a Series!)
- **unstack**: pivots from rows into columns
- unstacking may add missing data
- stacking filters out missing data (unless `dropna=False`)
- can unstack at a different level by passing it (e.g. 0), defaults to innermost level
Merges (aka Joins)

- Need to merge data from one DataFrame with data from another DataFrame
- Example: Football game data merged with temperature data

<table>
<thead>
<tr>
<th>Id</th>
<th>Location</th>
<th>Date</th>
<th>Home</th>
<th>Away</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Boston</td>
<td>9/2</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>Boston</td>
<td>9/9</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Cleveland</td>
<td>9/16</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>San Diego</td>
<td>9/23</td>
<td>21</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>wld</th>
<th>City</th>
<th>Date</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Boston</td>
<td>9/2</td>
<td>72</td>
</tr>
<tr>
<td>1</td>
<td>Boston</td>
<td>9/3</td>
<td>68</td>
</tr>
<tr>
<td>7</td>
<td>Boston</td>
<td>9/9</td>
<td>75</td>
</tr>
<tr>
<td>36</td>
<td>Cleveland</td>
<td>9/16</td>
<td>81</td>
</tr>
</tbody>
</table>

No data for San Diego
## Left Strategy

### Merged

<table>
<thead>
<tr>
<th>Id</th>
<th>Location</th>
<th>Date</th>
<th>Home</th>
<th>Away</th>
<th>Temp</th>
<th>wId</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Boston</td>
<td>9/2</td>
<td>1</td>
<td>15</td>
<td>72</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Boston</td>
<td>9/9</td>
<td>1</td>
<td>7</td>
<td>75</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Cleveland</td>
<td>9/16</td>
<td>12</td>
<td>1</td>
<td>81</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>San Diego</td>
<td>9/23</td>
<td>21</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>
## Tidy Data

**Initial Data**

<table>
<thead>
<tr>
<th>John Smith</th>
<th>Jane Doe</th>
<th>Mary Johnson</th>
</tr>
</thead>
<tbody>
<tr>
<td>treatmenta</td>
<td>—</td>
<td>2</td>
</tr>
<tr>
<td>Jane Doe</td>
<td>16</td>
<td>11</td>
</tr>
<tr>
<td>Mary Johnson</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**Transpose**

<table>
<thead>
<tr>
<th>John Smith</th>
<th>Jane Doe</th>
<th>Mary Johnson</th>
</tr>
</thead>
<tbody>
<tr>
<td>treatmenta</td>
<td>—</td>
<td>16</td>
</tr>
<tr>
<td>treatmentb</td>
<td>2</td>
<td>11</td>
</tr>
</tbody>
</table>

**Tidy Data**

<table>
<thead>
<tr>
<th>name</th>
<th>trt</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>a</td>
<td>—</td>
</tr>
<tr>
<td>Jane Doe</td>
<td>a</td>
<td>16</td>
</tr>
<tr>
<td>Mary Johnson</td>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>John Smith</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>Jane Doe</td>
<td>b</td>
<td>11</td>
</tr>
<tr>
<td>Mary Johnson</td>
<td>b</td>
<td>1</td>
</tr>
</tbody>
</table>

[H. Wickham, 2014]
Review

• Data Aggregation
  - Why?
  - Split-Apply-Combine
  - Pivot Tables & Crosstabs
  - Aggregate Visualizations

• Time Series
  - Representing date and time
  - Frequencies and ranges
  - Time zones
  - Resampling: downsampling and upsampling
  - Window functions
Aggregation of time series data, a special use case of `groupby`, is referred to as `resampling` in this book and will receive separate treatment in Chapter 10.

**GroupBy Mechanics**

Hadley Wickham, an author of many popular packages for the R programming language, coined the term `split-apply-combine` for talking about group operations, and I think that’s a good description of the process. In the first stage of the process, data contained in a pandas object, whether a Series, DataFrame, or otherwise, is split into groups based on one or more keys that you provide. The splitting is performed on a particular axis of an object. For example, a DataFrame can be grouped on its rows (`axis=0`) or its columns (`axis=1`). Once this is done, a function is applied to each group, producing a new value. Finally, the results of all those function applications are combined into a result object. The form of the resulting object will usually depend on what’s being done to the data. See Figure 9-1 for a mockup of a simple group aggregation.

**Figure 9-1. Illustration of a group aggregation**

Each grouping key can take many forms, and the keys do not have to be all of the same type:

- A list or array of values that is the same length as the axis being grouped
- A value indicating a column name in a DataFrame

[W. McKinney, Python for Data Analysis]
Pivot Tables and Crosstabs

- `tips.pivot_table(index=['sex', 'smoker'])`

<table>
<thead>
<tr>
<th>sex</th>
<th>smoker</th>
<th>size</th>
<th>tip</th>
<th>tip_pct</th>
<th>total_bill</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female</td>
<td>No</td>
<td>2.592593</td>
<td>2.773519</td>
<td>0.156921</td>
<td>18.105185</td>
</tr>
<tr>
<td>Female</td>
<td>Yes</td>
<td>2.242424</td>
<td>2.931515</td>
<td>0.182150</td>
<td>17.977879</td>
</tr>
<tr>
<td>Male</td>
<td>No</td>
<td>2.711340</td>
<td>3.113402</td>
<td>0.160669</td>
<td>19.791237</td>
</tr>
<tr>
<td>Male</td>
<td>Yes</td>
<td>2.500000</td>
<td>3.051167</td>
<td>0.152771</td>
<td>22.284500</td>
</tr>
</tbody>
</table>

- `pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)`

```python
# can also unstack this series into a dataframe
result = result.unstack()  
# can get arbitrary quantiles
tips.groupby('smoker')['tip_pct'].quantile(0.9)  
tips.pivot_table(index=['sex', 'smoker'])
```