DSC 201: Data Analysis & Visualization

Time Series

Dr. David Koop
Pivot Tables

- **tips**

<table>
<thead>
<tr>
<th>total_bill</th>
<th>tip</th>
<th>sex</th>
<th>smoker</th>
<th>day</th>
<th>time</th>
<th>size</th>
<th>tip_pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.99</td>
<td>1.01</td>
<td>Female</td>
<td>No</td>
<td>Sun</td>
<td>Dinner</td>
<td>2</td>
<td>0.059447</td>
</tr>
<tr>
<td>10.34</td>
<td>1.66</td>
<td>Male</td>
<td>No</td>
<td>Sun</td>
<td>Dinner</td>
<td>3</td>
<td>0.160542</td>
</tr>
<tr>
<td>21.01</td>
<td>3.50</td>
<td>Male</td>
<td>No</td>
<td>Sun</td>
<td>Dinner</td>
<td>3</td>
<td>0.166587</td>
</tr>
<tr>
<td>23.68</td>
<td>3.31</td>
<td>Male</td>
<td>No</td>
<td>Sun</td>
<td>Dinner</td>
<td>2</td>
<td>0.139780</td>
</tr>
<tr>
<td>24.59</td>
<td>3.61</td>
<td>Female</td>
<td>No</td>
<td>Sun</td>
<td>Dinner</td>
<td>4</td>
<td>0.146808</td>
</tr>
<tr>
<td>25.29</td>
<td>4.71</td>
<td>Male</td>
<td>No</td>
<td>Sun</td>
<td>Dinner</td>
<td>4</td>
<td>0.186240</td>
</tr>
<tr>
<td>8.77</td>
<td>2.00</td>
<td>Male</td>
<td>No</td>
<td>Sun</td>
<td>Dinner</td>
<td>2</td>
<td>0.228050</td>
</tr>
</tbody>
</table>

- **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></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></td>
<td>Yes</td>
<td>2.500000</td>
<td>3.051167</td>
<td>0.152771</td>
<td>22.284500</td>
</tr>
</tbody>
</table>
Pivot Tables with Margins and Aggfunc

- `tips.pivot_table(["size"], index=["sex", "day"], columns="smoker", aggfunc="sum", margins=True)`

<table>
<thead>
<tr>
<th></th>
<th>size</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>smoker</td>
<td>No</td>
<td>Yes</td>
<td>All</td>
</tr>
<tr>
<td>sex</td>
<td>day</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Female</td>
<td>Fri</td>
<td>2.0</td>
<td>7.0</td>
<td>9.0</td>
</tr>
<tr>
<td></td>
<td>Sat</td>
<td>13.0</td>
<td>15.0</td>
<td>28.0</td>
</tr>
<tr>
<td></td>
<td>Sun</td>
<td>14.0</td>
<td>4.0</td>
<td>18.0</td>
</tr>
<tr>
<td></td>
<td>Thur</td>
<td>25.0</td>
<td>7.0</td>
<td>32.0</td>
</tr>
<tr>
<td>Male</td>
<td>Fri</td>
<td>2.0</td>
<td>8.0</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>Sat</td>
<td>32.0</td>
<td>27.0</td>
<td>59.0</td>
</tr>
<tr>
<td></td>
<td>Sun</td>
<td>43.0</td>
<td>15.0</td>
<td>58.0</td>
</tr>
<tr>
<td></td>
<td>Thur</td>
<td>20.0</td>
<td>10.0</td>
<td>30.0</td>
</tr>
<tr>
<td>All</td>
<td>151.0</td>
<td>93.0</td>
<td>244.0</td>
<td></td>
</tr>
</tbody>
</table>
Crosstab

• A special group case for group frequencies

<table>
<thead>
<tr>
<th></th>
<th>smoker</th>
<th>No</th>
<th>Yes</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>day</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dinner</td>
<td>Fri</td>
<td>3</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Sat</td>
<td>45</td>
<td>42</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>Sun</td>
<td>57</td>
<td>19</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>Thur</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Lunch</td>
<td>Fri</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Thur</td>
<td>44</td>
<td>17</td>
<td>61</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td>151</td>
<td>93</td>
<td>244</td>
</tr>
</tbody>
</table>

• `pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)`
Dates and Times

• What is time to a computer?
  - Can be stored as seconds since Unix Epoch (January 1st, 1970)
• Often useful to break down into minutes, hours, days, months, years…
• Lots of different ways to write dates and times:
  - How could you write "November 29, 2016"?
  - European vs. American ordering…
• What about time zones?
Python Support for Time

• The `datetime` package
  - Has date, time, and datetime classes
  - `.now()` method: the current datetime
  - Can access properties of the time (year, month, seconds, etc.)

• Converting from strings to datetimes:
  - `datetime.strptime`: good for known formats
  - `dateutil.parser.parse`: good for unknown formats

• Converting to strings
  - `str(dt) or dt.strftime(<format>)`

• Differences between times
  - `datetime.timedelta`: can get number of days/hours/etc.
  - deal with issues with different length months, etc.
Pandas Support for Datetime

- `pd.to_datetime`:
  - convenience method
  - can convert an entire column to datetime
- Has a `NaT` to indicate a missing time value (like `NaN` but for timestamps)
- Date Ranges:
  - `index = pd.date_range('4/1/2012', periods=20)`
  - `pd.date_range('1/1/2000','1/3/2000 23:59', freq='4h')"
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']`
Shifting Time Series

- Data:
  
  [('2017-11-30', 48), ('2017-12-02', 45),
   ('2017-12-03', 44), ('2017-12-04', 48)]

- Compute day-to-day difference in high temperature:

  - s - s.shift(1)
    
    2017-11-30    NaN
    2017-12-02   -3.0
    2017-12-03   -1.0
    2017-12-04    4.0

  - s - s.shift(1, 'd')
    
    2017-11-30    NaN
    2017-12-01    NaN
    2017-12-02    NaN
    2017-12-03   -1.0
    2017-12-04    4.0
    2017-12-05    NaN
Shifting Data

• Leading or Lagging Data

```python
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
```

```python
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
```

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

• Shifting by time:

```python
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
```
Assignment 5

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

- Wednesday, Dec. 13
- Covers all material in the courses
- More focus on the last few topics since Test 2
**Timedelta**

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

```python
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

• Why?
• 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)
Python, Pandas, and Time Zones

- Time series in pandas are **time zone native**
- The pytz module keeps track of all of the time zone parameters
  - even Daylight Savings Time
- Localize a timestamp using `tz_localize`
  
  ```python
  ts = pd.Timestamp("1 Dec 2016 12:30 PM")
  ts = ts.tz_localize("US/Eastern")
  ```
- Convert a timestamp using `tz_convert`
  
  ```python
  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

```python
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')
```
Lots of Frequencies (not comprehensive)

<table>
<thead>
<tr>
<th>Alias</th>
<th>Offset type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Day</td>
<td>Calendar daily</td>
</tr>
<tr>
<td>B</td>
<td>BusinessDay</td>
<td>Business daily</td>
</tr>
<tr>
<td>H</td>
<td>Hour</td>
<td>Hourly</td>
</tr>
<tr>
<td>T or m</td>
<td>Minute</td>
<td>Minutely</td>
</tr>
<tr>
<td>S</td>
<td>Second</td>
<td>Secondly</td>
</tr>
<tr>
<td>L or ns</td>
<td>Mill</td>
<td>Millisecond (1/1,000 of 1 second)</td>
</tr>
<tr>
<td>U</td>
<td>Micro</td>
<td>Microsecond (1/1,000,000 of 1 second)</td>
</tr>
<tr>
<td>M</td>
<td>MonthEnd</td>
<td>Last calendar day of month</td>
</tr>
<tr>
<td>BM</td>
<td>BusinessMonthEnd</td>
<td>Last business day (weekday) of month</td>
</tr>
<tr>
<td>MS</td>
<td>MonthBegin</td>
<td>First calendar day of month</td>
</tr>
<tr>
<td>BMS</td>
<td>BusinessMonthBegin</td>
<td>First weekday of month</td>
</tr>
<tr>
<td>W-MON, W-TUE, ...</td>
<td>Week</td>
<td>Weekly on given day of week (MON, TUE, WED, THU, FRI, SAT, or SUN)</td>
</tr>
<tr>
<td>WOM-1MON, WOM-2MON, ...</td>
<td>WeekOfMonth</td>
<td>Generate weekly dates in the first, second, third, or fourth week of the month (e.g., WOM-3FRI for the third Friday of each month)</td>
</tr>
<tr>
<td>Q-JAN, Q-FEB, ...</td>
<td>QuarterEnd</td>
<td>Quarterly dates anchored on last calendar day of each month, for year ending in indicated month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)</td>
</tr>
<tr>
<td>BQ-JAN, BQ-FEB, ...</td>
<td>BusinessQuarterEnd</td>
<td>Quarterly dates anchored on last weekday day of each month, for year ending in indicated month</td>
</tr>
<tr>
<td>QS-JAN, QS-FEB, ...</td>
<td>QuarterBegin</td>
<td>Quarterly dates anchored on first calendar day of each month, for year ending in indicated month</td>
</tr>
<tr>
<td>BQS-JAN, BQS-FEB, ...</td>
<td>BusinessQuarterBegin</td>
<td>Quarterly dates anchored on first weekday day of each month, for year ending in indicated month</td>
</tr>
<tr>
<td>A-JAN, A-FEB, ...</td>
<td>YearEnd</td>
<td>Annual dates anchored on last calendar day of given month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)</td>
</tr>
<tr>
<td>BA-JAN, BA-FEB, ...</td>
<td>BusinessYearEnd</td>
<td>Annual dates anchored on last weekday of given month</td>
</tr>
<tr>
<td>AS-JAN, AS-FEB, ...</td>
<td>YearBegin</td>
<td>Annual dates anchored on first day of given month</td>
</tr>
<tr>
<td>BAS-JAN, BAS-FEB, ...</td>
<td>BusinessYearBegin</td>
<td>Annual dates anchored on first weekday of given month</td>
</tr>
</tbody>
</table>
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                  In [97]: ts.shift(2)         In [98]: ts.shift(-2)
Out[96]:                     Out[97]:                     Out[98]:
2000-01-31   -0.066748       2000-01-31         NaN       2000-01-31   -0.117388
2000-02-29    0.838639       2000-02-29         NaN       2000-02-29   -0.517795
2000-03-31   -0.117388       2000-03-31   -0.066748       2000-03-31         NaN
2000-04-30   -0.517795       2000-04-30    0.838639       2000-04-30         NaN
Freq: M, dtype: float64      Freq: M, dtype: float64      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 Second(15))</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' / Second(-1) 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>

[W. McKinney, Python for Data Analysis]
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
```

Label:
```
label='left'
label='right'
```
Upsampling

- No aggregation necessary

```python
In [222]: frame
Out[222]:
   Colorado    Texas    New York    Ohio
2000-01-05 -0.896431  0.677263  0.036503  0.087102
2000-01-12 -0.046662  0.927238  0.482284 -0.867130

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

In [224]: df_daily
Out[224]:
   Colorado    Texas    New York    Ohio
2000-01-05 -0.896431  0.677263  0.036503  0.087102
2000-01-06  NaN        NaN        NaN        NaN
2000-01-07  NaN        NaN        NaN        NaN
2000-01-08  NaN        NaN        NaN        NaN
2000-01-09  NaN        NaN        NaN        NaN
2000-01-10  NaN        NaN        NaN        NaN
2000-01-11  NaN        NaN        NaN        NaN
2000-01-12 -0.046662  0.927238  0.482284 -0.867130

In [225]: frame.resample('D').ffill()
Out[225]:
   Colorado    Texas    New York    Ohio
2000-01-05 -0.896431  0.677263  0.036503  0.087102
2000-01-06 -0.896431  0.677263  0.036503  0.087102
2000-01-07 -0.896431  0.677263  0.036503  0.087102
2000-01-08 -0.896431  0.677263  0.036503  0.087102
2000-01-09 -0.896431  0.677263  0.036503  0.087102
2000-01-10 -0.896431  0.677263  0.036503  0.087102
2000-01-11 -0.896431  0.677263  0.036503  0.087102
2000-01-12 -0.046662  0.927238  0.482284 -0.867130
```

```
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

• In old versions of pandas (like the book uses), this used to be rolling_count, rolling_sum, rolling_mean

• Specify the window size, then an aggregation method

• Can also specify the window

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

The expression `rolling(250)` is similar in behavior to `groupby`, but instead of grouping it creates an object that enables grouping over a 250-day sliding window. So here we have the 250-day moving window average of Apple's stock price.
Food Inspections Example

• Questions:
  - When was a location was last inspected?
  - How many inspections are done per month/year/etc.?
  - Is there any trend in the number of inspections over time?

• Notebook posted on course web page