Advanced Data Management (CSCI 680/490)

Data Wrangling

Dr. David Koop
NumPy

- Fast **vectorized** array operations for data munging and cleaning, subsetting and filtering, transformation, and any other kinds of computations
- Common array algorithms like sorting, unique, and set operations
- Efficient descriptive statistics and aggregating/summarizing data
- Data alignment and relational data manipulations for merging and joining together heterogeneous data sets
- Expressing conditional logic as array expressions instead of loops with `if-elif-else` branches
- Group-wise data manipulations (aggregation, transformation, function application).

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

- **What is this data?**

<table>
<thead>
<tr>
<th></th>
<th>42ND STREET &amp; 8TH AVENUE</th>
<th>42ND STREET &amp; GRAND CENTRAL</th>
<th>14TH STREET-UNION SQUARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>R011</td>
<td>00228985</td>
<td>00207758</td>
<td>00224603</td>
</tr>
<tr>
<td>R170</td>
<td>0008471</td>
<td>00080983</td>
<td>0011051</td>
</tr>
<tr>
<td>R046</td>
<td>00000441</td>
<td>00000323</td>
<td>00000827</td>
</tr>
<tr>
<td></td>
<td>00001455</td>
<td>00001183</td>
<td>00003026</td>
</tr>
<tr>
<td></td>
<td>00000134</td>
<td>000003001</td>
<td>00000660</td>
</tr>
<tr>
<td></td>
<td>00033341</td>
<td>00040759</td>
<td>00089367</td>
</tr>
<tr>
<td></td>
<td>00071255</td>
<td>00096613</td>
<td>00199841</td>
</tr>
</tbody>
</table>

- **Semantics**: real-world meaning of the data
- **Type**: structural or mathematical interpretation
- **Both often require metadata**
  - Sometimes we can infer some of this information
  - Line between data and metadata isn’t always clear
Semantics

• The meaning of the data
• Example: 94023, 90210, 02747, 60115
Semantics

- The meaning of the data
- Example: 94023, 90210, 02747, 60115
  - Attendance at college football games?
Semantics

• The meaning of the data
• Example: 94023, 90210, 02747, 60115
  - Attendance at college football games?
  - Salaries?
Semantics

• The meaning of the data
• Example: 94023, 90210, 02747, 60115
  - Attendance at college football games?
  - Salaries?
  - Zip codes?
• Cannot always infer based on what the data looks like
• Often require semantics to better understand data, column names help
• May also include rules about data: a zip code is part of an address that uniquely identifies a residence
• Useful for asking good questions about the data
Data Terminology

• Items
  - An **item** is an individual discrete entity
  - e.g., a row in a table

• Attributes
  - An **attribute** is some specific property that can be measured, observed, or logged
  - a.k.a. variable, (data) dimension
  - e.g., a column in a table
Tables

**Flat**
- Data organized by rows & columns
  - row ~ item (usually)
  - column ~ attribute
  - label ~ attribute name
- Key: identifies each item (row)
  - Usually *unique*
  - Allows *join* of data from 2+ tables
- Compound key: key split among multiple columns, e.g. (state, year) for population

**Multidimensional**
- Split compound key

---

[Munzner (ill. Maguire), 2014]
Attribute Types

⇒ Categorical

⇒ Ordered

⇒ Ordinal

⇒ Quantitative

[Munzner (ill. Maguire), 2014]
Assignment 2

- Basically the same as Assignment 1, now with pandas and duckdb
- Can either do each task at the same time (one in pandas, one in duckdb), or all tasks in pandas then all tasks in duckdb
Test 1

• Next Wednesday, Feb. 23
• In-class, 3:30-4:45pm in PM 153
• Format:
  - Multiple Choice
  - Free Response
• More information to be posted online soon
Reading

- Wednesday
- Discussing paper:
  - "Wrangler: Interactive Visual Specification of Data Transformation Scripts"
  - Kandel et al.
- Read
- Come prepared with questions, thoughts
  - Compare with how things work in pandas
pandas

- Contains high-level data structures and manipulation tools designed to make data analysis fast and easy in Python
- Built on top of NumPy
- Requirements:
  - Data structures with labeled axes (aligning data)
  - Time series data
  - Arithmetic operations that include metadata (labels)
  - Handle missing data
  - Merge and relational operations
Series

• A one-dimensional array (with a type) with an index
• Index defaults to numbers but can also be text (like a dictionary)
• Allows easier reference to specific items
• `obj = pd.Series([7,14,-2,1])`

• Basically two arrays: `obj.values` and `obj.index`
• Can specify the index explicitly and use strings
  • `obj2 = pd.Series([4, 7, -5, 3],
                      index=['d', 'b', 'a', 'c'])`

• Kind of like fixed-length, ordered dictionary + can create from a dictionary
• `obj3 = pd.Series({'Ohio': 35000, 'Texas': 71000,
                   'Oregon': 16000, 'Utah': 5000})`
Series

- **Indexing:** `s[1]` or `s['Oregon']`
- **Can check for missing data:** `pd.isnull(s)` or `pd.notnull(s)`
- **Both index and values can have an associated name:**
  - `s.name = 'population'; s.index.name = 'state'`
- **Addition and NumPy ops work as expected and preserve the index-value link**
- **These operations **align:**

<table>
<thead>
<tr>
<th>In [28]: obj3</th>
<th>In [29]: obj4</th>
<th>In [30]: obj3 + obj4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ohio</td>
<td>35000</td>
<td>California     NaN</td>
</tr>
<tr>
<td>Oregon</td>
<td>16000</td>
<td>Ohio           35000</td>
</tr>
<tr>
<td>Texas</td>
<td>71000</td>
<td>Oregon         16000</td>
</tr>
<tr>
<td>Utah</td>
<td>5000</td>
<td>Texas           71000</td>
</tr>
<tr>
<td>dtype: int64</td>
<td></td>
<td>dtype: float64</td>
</tr>
</tbody>
</table>

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

- A dictionary of Series (labels for each series)
- A spreadsheet with column headers
- Has an index shared with each series
- Allows easy reference to any cell

- Index is automatically assigned just as with a series but can be passed in as well via index kwarg
- Can reassign column names by passing columns kwarg
```python
df = pd.read_csv('penguins_lter.csv')
```

<table>
<thead>
<tr>
<th>studyName</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PAL0708</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PAL0708</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PAL0708</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PAL0708</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PAL0708</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>339</td>
<td>PAL0910</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>340</td>
<td>PAL0910</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
<td></td>
</tr>
<tr>
<td>341</td>
<td>PAL0910</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
<td></td>
</tr>
<tr>
<td>342</td>
<td>PAL0910</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
<td></td>
</tr>
<tr>
<td>343</td>
<td>PAL0910</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
<td></td>
</tr>
</tbody>
</table>

344 rows x 17 columns
df = pd.read_csv('penguins_lter.csv')

<table>
<thead>
<tr>
<th>studyName</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL0708</td>
<td>1</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>PAL0708</td>
<td>2</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>PAL0708</td>
<td>3</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>PAL0708</td>
<td>4</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0708</td>
<td>5</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>PAL0910</td>
<td>120</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0910</td>
<td>121</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>PAL0910</td>
<td>122</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
<tr>
<td>PAL0910</td>
<td>124</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
</tr>
</tbody>
</table>

344 rows x 17 columns
# Data Frame

```python
df = pd.read_csv('penguins_lter.csv')
```

<table>
<thead>
<tr>
<th>Index</th>
<th>Study Name</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PAL0708</td>
<td>1</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>1</td>
<td>PAL0708</td>
<td>2</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>2</td>
<td>PAL0708</td>
<td>3</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>3</td>
<td>PAL0708</td>
<td>4</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>PAL0708</td>
<td>5</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
</tr>
<tr>
<td>339</td>
<td>PAL0910</td>
<td>120</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>340</td>
<td>PAL0910</td>
<td>121</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>341</td>
<td>PAL0910</td>
<td>122</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>342</td>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
<tr>
<td>343</td>
<td>PAL0910</td>
<td>124</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
</tr>
</tbody>
</table>

344 rows x 17 columns
### Data Frame

```python
df = pd.read_csv('penguins_lter.csv')
```

<table>
<thead>
<tr>
<th></th>
<th>studyName</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PAL0708</td>
<td>1</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>1</td>
<td>PAL0708</td>
<td>2</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>2</td>
<td>PAL0708</td>
<td>3</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>3</td>
<td>PAL0708</td>
<td>4</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>PAL0708</td>
<td>5</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>339</td>
<td>PAL0910</td>
<td>120</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>340</td>
<td>PAL0910</td>
<td>121</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>341</td>
<td>PAL0910</td>
<td>122</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>342</td>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
<tr>
<td>343</td>
<td>PAL0910</td>
<td>124</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
</tr>
</tbody>
</table>

344 rows x 17 columns

**Column:** `df['Island']`
<table>
<thead>
<tr>
<th>Index</th>
<th>Row: df.loc[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PAL0708</td>
</tr>
<tr>
<td>1</td>
<td>PAL0708</td>
</tr>
<tr>
<td>2</td>
<td>PAL0708</td>
</tr>
<tr>
<td>3</td>
<td>PAL0708</td>
</tr>
<tr>
<td>4</td>
<td>PAL0708</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>339</td>
<td>PAL0910</td>
</tr>
<tr>
<td>340</td>
<td>PAL0910</td>
</tr>
<tr>
<td>341</td>
<td>PAL0910</td>
</tr>
<tr>
<td>342</td>
<td>PAL0910</td>
</tr>
<tr>
<td>343</td>
<td>PAL0910</td>
</tr>
</tbody>
</table>

**Column Names**

<table>
<thead>
<tr>
<th>studyName</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL0708</td>
<td>1</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>PAL0708</td>
<td>2</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>PAL0708</td>
<td>3</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>PAL0708</td>
<td>4</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0708</td>
<td>5</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
</tr>
<tr>
<td>PAL0910</td>
<td>120</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0910</td>
<td>121</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>PAL0910</td>
<td>122</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
<tr>
<td>PAL0910</td>
<td>124</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
</tr>
</tbody>
</table>

344 rows x 17 columns

**Column: df['Island']**

\[
df = pd.read_csv('penguins_lter.csv')
\]
Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

<table>
<thead>
<tr>
<th>studyName</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Clummen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL0708</td>
<td>1</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>PAL0708</td>
<td>2</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>PAL0708</td>
<td>3</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>PAL0708</td>
<td>4</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0708</td>
<td>5</td>
<td>Adelie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>36.7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>PAL0910</td>
<td>120</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>PAL0910</td>
<td>121</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>PAL0910</td>
<td>124</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
</tbody>
</table>

344 rows x 17 columns

Row: `df.loc[2]`

Cell: `df.loc[341, 'Species']`

Column: `df['Island']`
## Data Frame

```python
df = pd.read_csv('penguins_lter.csv')
```

### Column Names

<table>
<thead>
<tr>
<th>Index</th>
<th>Study Name</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PAL0708</td>
<td>1</td>
<td>Adélie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A1</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.1</td>
</tr>
<tr>
<td>1</td>
<td>PAL0708</td>
<td>2</td>
<td>Adélie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N1A2</td>
<td>Yes</td>
<td>11/11/07</td>
<td>39.5</td>
</tr>
<tr>
<td>2</td>
<td>PAL0708</td>
<td>3</td>
<td>Adélie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
<tr>
<td>3</td>
<td>PAL0708</td>
<td>4</td>
<td>Adélie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A2</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>PAL0708</td>
<td>5</td>
<td>Adélie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N3A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>NaN</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>339</td>
<td>PAL0910</td>
<td>120</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N38A2</td>
<td>No</td>
<td>12/1/09</td>
<td>NaN</td>
</tr>
<tr>
<td>340</td>
<td>PAL0910</td>
<td>121</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>46.8</td>
</tr>
<tr>
<td>342</td>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N39A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>50.4</td>
</tr>
<tr>
<td>343</td>
<td>PAL0910</td>
<td>124</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A1</td>
<td>Yes</td>
<td>11/22/09</td>
<td>45.2</td>
</tr>
</tbody>
</table>

**344 rows x 17 columns**

### Row: `df.loc[2]`

<table>
<thead>
<tr>
<th>Index</th>
<th>Study Name</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>PAL0708</td>
<td>3</td>
<td>Adélie Penguin (Pygoscelis adeliae)</td>
<td>Anvers</td>
<td>Torgersen</td>
<td>Adult, 1 Egg Stage</td>
<td>N2A1</td>
<td>Yes</td>
<td>11/16/07</td>
<td>40.3</td>
</tr>
</tbody>
</table>

### Cell: `df.loc[341, 'Species']`

<table>
<thead>
<tr>
<th>Study Name</th>
<th>Sample Number</th>
<th>Species</th>
<th>Region</th>
<th>Island</th>
<th>Stage</th>
<th>Individual ID</th>
<th>Clutch Completion</th>
<th>Date Egg</th>
<th>Culmen Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL0910</td>
<td>123</td>
<td>Gentoo penguin (Pygoscelis papua)</td>
<td>Anvers</td>
<td>Biscoe</td>
<td>Adult, 1 Egg Stage</td>
<td>N43A2</td>
<td>Yes</td>
<td>11/22/09</td>
<td>49.9</td>
</tr>
</tbody>
</table>

### Column: `df['Island']`

- **Torgersen**
- **Biscoe**
## DataFrame Constructor Inputs

<table>
<thead>
<tr>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D ndarray</td>
<td>A matrix of data, passing optional row and column labels</td>
</tr>
<tr>
<td>dict of arrays, lists, or tuples</td>
<td>Each sequence becomes a column in the DataFrame. All sequences must be the same length.</td>
</tr>
<tr>
<td>NumPy structured/record array</td>
<td>Treated as the “dict of arrays” case</td>
</tr>
<tr>
<td>dict of Series</td>
<td>Each value becomes a column. Indexes from each Series are unioned together to form the result’s row index if no explicit index is passed.</td>
</tr>
<tr>
<td>dict of dicts</td>
<td>Each inner dict becomes a column. Keys are unioned to form the row index as in the “dict of Series” case.</td>
</tr>
<tr>
<td>list of dicts or Series</td>
<td>Each item becomes a row in the DataFrame. Union of dict keys or Series indexes become the DataFrame’s column labels</td>
</tr>
<tr>
<td>List of lists or tuples</td>
<td>Treated as the “2D ndarray” case</td>
</tr>
<tr>
<td>Another DataFrame</td>
<td>The DataFrame’s indexes are used unless different ones are passed</td>
</tr>
<tr>
<td>NumPy MaskedArray</td>
<td>Like the “2D ndarray” case except masked values become NA/missing in the DataFrame result</td>
</tr>
</tbody>
</table>

---

D. Koop, CSCI 680/490, Spring 2022
DataFrame Access and Manipulation

- `df.values` → 2D NumPy array

- Accessing a column:
  - `df["<column>"]`
  - `df.<column>`

  - Both return Series
  - Dot syntax only works when the column is a valid identifier

- Assigning to a column:
  - `df["<column>"] = <scalar>` # all cells set to same value
  - `df["<column>"] = <array>` # values set in order
  - `df["<column>"] = <series>` # values set according to match
    # between df and series indexes
DataFrame Index

- Similar to index for Series
- Immutable
- Can be shared with multiple structures (DataFrames or Series)
- `in` operator works with: 'Ohio' in df.index
# Index methods and properties

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>Concatenate with additional Index objects, producing a new Index</td>
</tr>
<tr>
<td>diff</td>
<td>Compute set difference as an Index</td>
</tr>
<tr>
<td>intersection</td>
<td>Compute set intersection</td>
</tr>
<tr>
<td>union</td>
<td>Compute set union</td>
</tr>
<tr>
<td>isin</td>
<td>Compute boolean array indicating whether each value is contained in the passed collection</td>
</tr>
<tr>
<td>delete</td>
<td>Compute new Index with element at index $i$ deleted</td>
</tr>
<tr>
<td>drop</td>
<td>Compute new index by deleting passed values</td>
</tr>
<tr>
<td>insert</td>
<td>Compute new Index by inserting element at index $i$</td>
</tr>
<tr>
<td>is_monotonic</td>
<td>Returns True if each element is greater than or equal to the previous element</td>
</tr>
<tr>
<td>is_unique</td>
<td>Returns True if the Index has no duplicate values</td>
</tr>
<tr>
<td>unique</td>
<td>Compute the array of unique values in the Index</td>
</tr>
</tbody>
</table>

---

In [78]: obj = Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])

In [79]: obj

Out[79]:

<table>
<thead>
<tr>
<th></th>
<th>d</th>
<th>b</th>
<th>a</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.5</td>
<td>7.2</td>
<td>-5.3</td>
<td>3.6</td>
</tr>
<tr>
<td>dtype: float64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

In [80]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])

In [81]: obj2

Out[81]:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-5.3</td>
<td>7.2</td>
<td>3.6</td>
<td>4.5</td>
<td>NaN</td>
</tr>
<tr>
<td>dtype: float64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reindexing

- `reindex` creates a new object with the data conformed to new index
- `obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])`

- Missing values: handle with kwargs
  - `fill_value`: fill any missing value with a specific value
  - `method='ffill'`: fill values forward
  - `method='bfill'`: fill values backward

- Data Frames:
  - reindex rows as with series
  - reindex columns using columns kwarg
Dropping entries

• Can drop one or more entries

• Series:
  - `new_obj = obj.drop('c')`
  - `new_obj = obj.drop(['d', 'c'])`

• Data Frames:
  - `axis` keyword defines which axis to drop (default 0)
  - `axis=0 → rows, axis=1 → columns`
  - `axis = 'columns'`
Indexing

• Same as with NumPy arrays but can use Series's index labels
• Slicing with labels: NumPy is exclusive, Pandas is inclusive!
  
  - `s = Series(np.arange(4))
    
    s[0:2] # gives two values like numpy
  
  - `s = Series(np.arange(4), index=['a', 'b', 'c', 'd'])`
    
    s['a':'c'] # gives three values, not two!

• Obtaining data subsets
  
  - `[]`: get columns by label
  
  - `loc`: get rows/cols by label
  
  - `iloc`: get rows/cols by position (integer index)
  
  - For single cells (scalars), also have `at` and `iat`
Indexing

- `s = Series(np.arange(4.), index=[4,3,2,1])`
- `s[3]`
- `s.loc[3]`
- `s.iloc[3]`
- `s2 = pd.Series(np.arange(4), index=['a','b','c','d'])`
- `s2[3]`
Filtering

• Same as with numpy arrays but allows use of column-based criteria
  - data[data < 5] = 0
  - data[data['three'] > 5]
  - data < 5 → boolean data frame, can be used to select specific elements
### Arithmetic

- Add, subtract, multiply, and divide are element-wise like numpy
- ...but use labels to align
- ...and missing labels lead to \( \text{NaN} \) (not a number) values

```
In [28]: obj3
Out[28]:
Ohio     35000
Oregon    16000
Texas     71000
Utah      5000
dtype: int64

In [29]: obj4
Out[29]:
Ohio     35000
Oregon    16000
Texas     71000
Utah      5000
California       NaN
dtype: float64

In [30]: obj3 + obj4
Out[30]:
Ohio     70000
Oregon    32000
Texas     142000
Utah      NaN
California       NaN
```

- also have `.add`, `.subtract`, ... that allow `fill_value` argument
- `obj3.add(obj4, fill_value=0)`
Arithmetic between DataFrames and Series

- **Broadcasting:** e.g. apply single row operation across all rows

- **Example:**

  In [148]: frame
  Out[148]:
  b   d   e
  Utah 0   1   2
  Ohio 3   4   5
  Texas 6   7   8
  Oregon 9   10   11

  In [149]: series
  Out[149]:
  b   d   e
  Utah 0   1   2
  Ohio 3   4   5
  Texas 6   7   8
  Name: Utah, dtype: float64

  In [150]: frame - series
  Out[150]:
  b   d   e
  Utah 0   0   0
  Ohio 0   0   0
  Texas 0   0   0
  Oregon 0   0   0

- To broadcast over **columns**, use methods (`.add, ...`)

  In [154]: frame
  Out[154]:
  b   d   e
  Utah 0   1   2
  Ohio 3   4   5
  Texas 6   7   8
  Oregon 9   10   11

  In [155]: series3
  Out[155]:
  b   d   e   f
  Utah 0   1   2   1
  Ohio 3   4   5   2
  Texas 6   7   8   3
  Oregon 9   10   11  4

  In [156]: frame.add(series3, axis=0)
  Out[156]:
  b   d   e   f
  Utah 0   1   2   2
  Ohio 3   4   5   2
  Texas 6   7   8   4
  Oregon 9   10   11  7
Sorting by Index (sort_index)

- Sort by index (lexicographical):

  ```python
  In [168]: obj = Series(range(4), index=['d', 'a', 'b', 'c'])
  In [169]: obj.sort_index()
  Out[169]:
  a    1
  b    2
  c    3
  d    0
  dtype: int64
  ```

- DataFrames sorting:

  ```python
  In [170]: frame = DataFrame(np.arange(8).reshape((2, 4)), index=['three', 'one'],
  columns=['d', 'a', 'b', 'c'])
  In [171]: frame.sort_index()   In [172]: frame.sort_index(axis=1)
  Out[171]:
     d  a  b  c    
  one  4  5  6  7
  three 0  1  2  3
  Out[172]:
     a  b  c  d
  one 5  6  7  4
  three 1  2  3  0
  ```

- axis controls sort rows (0) vs. sort columns (1)
Sorting by Value (sort_values)

- **sort_values** method on series
  - `obj.sort_values()`

- Missing values (NaN) are at the end by default (`na_position` controls, can be first)

- **sort_values** on DataFrame:
  - `df.sort_values(<list-of-columns>)`
  - `df.sort_values(by=['a', 'b'])`
  - Can also use `axis=1` to sort by index labels
String Transformation

• One of the reasons for Python's popularity is string/text processing
• `split(<delimiter>):` break a string into pieces:
  - `s = "12,13, 14"
    slist = s.split(',') # ["12", "13", " 14"]
• `<delimiter>.join([<str>]):` join several strings by a delimiter
  - `":".join(slist) # "12:13: 14"
• `strip():` remove leading and trailing whitespace
  - `[p.strip() for p in slist] # ["12", "13", "14"]`
String Transformation

- `replace(<from>,<to>):` change substrings to another substring
- `upper()/lower():` casing
- `index(<str>):` find where a substring first occurs (Error if not found)
- `find(<str>):` same as index but \(-1\) if not found
- `startswith()|endswith():` boolean checks for string occurrence
Regular Expressions in Python

• import re
• re.search(<pattern>, <str_to_check>)
  - Returns None if no match, information about the match otherwise
• Capturing information about what is in a string → parentheses
• \(\d+\)/\d+/\d+ will capture information about the month
• match = re.search('\(\d+\)/\d+/\d+','12/31/2016')
  if match:
      match.group() # 12
• re.findall(<pattern>, <str_to_check>)
  - Finds all matches in the string, search only finds the first match
• Can pass in flags to alter methods: e.g. re.IGNORECASE
Pandas String Methods

- Any column or series can have the string methods (e.g. replace, split) applied to the entire series
- Fast (vectorized) on whole columns or datasets
- Use `.str.<method_name>`
- `.str` is **important**!

```python
- data = pd.Series({'Dave': 'dave@google.com',
   'Steve': 'steve@gmail.com',
   'Rob': 'rob@gmail.com',
   'Wes': np.nan})

data.str.contains('gmail')
data.str.split('@').str[1]
data.str[-3:]
```
Regular Expression Methods

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>findall</td>
<td>Return all non-overlapping matching patterns in a string as a list</td>
</tr>
<tr>
<td>finditer</td>
<td>Like findall, but returns an iterator</td>
</tr>
<tr>
<td>match</td>
<td>Match pattern at start of string and optionally segment pattern components into groups; if the pattern matches, returns a match object, and otherwise None</td>
</tr>
<tr>
<td>search</td>
<td>Scan string for match to pattern; returning a match object if so; unlike match, the match can be anywhere in the string as opposed to only at the beginning</td>
</tr>
<tr>
<td>split</td>
<td>Break string into pieces at each occurrence of pattern</td>
</tr>
<tr>
<td>sub, subn</td>
<td>Replace all (sub) or first n occurrences (subn) of pattern in string with replacement expression; use symbols \1, \2, ... to refer to match group elements in the replacement string</td>
</tr>
</tbody>
</table>
Pandas String Methods with Regexs

In [172]: pattern
Out[172]: '([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.(A-Z\{2,4})'

In [173]: data.str.findall(pattern, flags=re.IGNORECASE)
Out[173]:
Dave      [(dave, google, com)]
Rob       [(rob, gmail, com)]
Steve     [(steve, gmail, com)]
Wes       NaN
dtype: object

In [174]: matches = data.str.match(pattern, flags=re.IGNORECASE)

In [175]: matches
Out[175]:
Dave       True
Rob        True
Steve      True
Wes        NaN
dtype: object

[W. McKinney, Python for Data Analysis]
Reading

- Wednesday
- Discussing paper:
  - "Wrangler: Interactive Visual Specification of Data Transformation Scripts"
  - Kandel et al.
- Read
- Come prepared with questions, thoughts
  - Compare with how things work in pandas