Advanced Data Management (CSCI 640/490)

Data Wrangling

Dr. David Koop
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
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
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 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]:
    California    NaN
    Ohio          35000
    Oregon        16000
    Texas         71000
    Utah           NaN
    dtype: float64

    In [30]: obj3 + obj4
    Out[30]:
    California    NaN
    Ohio          70000
    Oregon        32000
    Texas         142000
    Utah           NaN
    dtype: float64
```

- also have .add, .subtract, ... that allow fill_value argument
- obj3.add(obj4, fill_value=0)
Mutating Dataframes

• assign allows new columns to be created, returns "new" dataframe
  - df2 = df.assign(Total=df.Points1 + df.Points2)

• More reusable:
  - df2 = df.assign(Total=lambda df: df.Points1 + df.Points2)

• If you have columns that are not proper identifiers, can use **kwargs
  - df2 = df.assign(**{"Total Points": lambda df: df.Points1 + df.Points2})
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 Methods

• Can manipulate columns of strings
  - Use the .str modifier
• Most string and regex operations are available
• Examples:
  - df.first_name.str.startswith("Jo")
  - df.name.str.split(' ')
  - df.name.str.split(' ')
DuckDB

- SQL syntax with extras
  - `read_csv_auto`
  - similar string, datetime, array functions to pandas
Ibis

- More Pythonic interface to database or dataframe systems
- Uses DuckDB as default backend, can be configured to use others
- Syntax aligns better with SQL, potentially clearer
  - select
  - filter
  - mutate
  - group_by
  - order_by
  - unnest
Assignment 2

- Assignment 1 Questions with pandas, DuckDB, and Ibis
- CS 640 students do all, CS 490 do pandas & DuckDB (Ibis is EC)
- Can work by framework or by query
- Most questions can be answered with a single statement… but that statement can take a while to write
  - Read documentation
  - Check hints
Test 1

- Monday, Feb. 27
- In-class, 9:30-10:45am
- Format:
  - Multiple Choice
  - Free Response
- Information will be posted online
Statistics

- **sum**: column sums (axis=1 gives sums over rows)
- missing values are excluded unless the whole slice is NaN
- **idxmax, idxmin** are like argmax, argmin (return index)
- **describe**: shortcut for easy stats!

```
In [204]: df.describe()
Out[204]:
    one       two
count 3.000000  2.000000
mean  3.083333 -2.900000
std   3.493685  2.262742
min   0.750000 -4.500000
25%   1.075000 -3.700000
50%   1.400000 -2.900000
75%   4.250000 -2.100000
max   7.100000 -1.300000
```

On non-numeric data, `describe` produces alternate summary statistics:

```
In [205]: obj = Series(['a', 'a', 'b', 'c'] * 4)
In [206]: obj.describe()
Out[206]:
count 16
unique  3
top a
top freq  8
dtype: object
```

See Table 5-10 for a full list of summary statistics and related methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Number of non-NA values</td>
</tr>
<tr>
<td>describe</td>
<td>Compute set of summary statistics for Series or each DataFrame column</td>
</tr>
<tr>
<td>min, max</td>
<td>Compute minimum and maximum values</td>
</tr>
<tr>
<td>argmin, argmax</td>
<td>Compute index locations (integers) at which minimum or maximum value obtained, respectively</td>
</tr>
<tr>
<td>idxmin, idxmax</td>
<td>Compute index values at which minimum or maximum value obtained, respectively</td>
</tr>
<tr>
<td>quantile</td>
<td>Compute sample quantile ranging from 0 to 1</td>
</tr>
<tr>
<td>sum</td>
<td>Sum of values</td>
</tr>
<tr>
<td>mean</td>
<td>Mean of values</td>
</tr>
<tr>
<td>median</td>
<td>Arithmetic median (50% quantile) of values</td>
</tr>
<tr>
<td>mad</td>
<td>Mean absolute deviation from mean value</td>
</tr>
<tr>
<td>var</td>
<td>Sample variance of values</td>
</tr>
<tr>
<td>std</td>
<td>Sample standard deviation of values</td>
</tr>
</tbody>
</table>
Statistics

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Number of non-NA values</td>
</tr>
<tr>
<td>describe</td>
<td>Compute set of summary statistics for Series or each DataFrame column</td>
</tr>
<tr>
<td>min, max</td>
<td>Compute minimum and maximum values</td>
</tr>
<tr>
<td>argmin, argmax</td>
<td>Compute index locations (integers) at which minimum or maximum value obtained, respectively</td>
</tr>
<tr>
<td>idxmin, idxmax</td>
<td>Compute index values at which minimum or maximum value obtained, respectively</td>
</tr>
<tr>
<td>quantile</td>
<td>Compute sample quantile ranging from 0 to 1</td>
</tr>
<tr>
<td>sum</td>
<td>Sum of values</td>
</tr>
<tr>
<td>mean</td>
<td>Mean of values</td>
</tr>
<tr>
<td>median</td>
<td>Arithmetic median (50% quantile) of values</td>
</tr>
<tr>
<td>mad</td>
<td>Mean absolute deviation from mean value</td>
</tr>
<tr>
<td>var</td>
<td>Sample variance of values</td>
</tr>
<tr>
<td>std</td>
<td>Sample standard deviation of values</td>
</tr>
<tr>
<td>skew</td>
<td>Sample skewness (3rd moment) of values</td>
</tr>
<tr>
<td>kurt</td>
<td>Sample kurtosis (4th moment) of values</td>
</tr>
<tr>
<td>cumsum</td>
<td>Cumulative sum of values</td>
</tr>
<tr>
<td>cummin, cummax</td>
<td>Cumulative minimum or maximum of values, respectively</td>
</tr>
<tr>
<td>cumprod</td>
<td>Cumulative product of values</td>
</tr>
<tr>
<td>diff</td>
<td>Compute 1st arithmetic difference (useful for time series)</td>
</tr>
<tr>
<td>pct_change</td>
<td>Compute percent changes</td>
</tr>
</tbody>
</table>

Another type of method is neither a reduction nor an accumulation. Describe is one such example, producing multiple summary statistics in one shot:

```python
In [204]: df.describe()
Out[204]:
          one        two
count    3.000000  2.000000
mean      3.083333 -2.900000
std       3.493685  2.262742
min       0.750000 -4.500000
25%       1.075000 -3.700000
50%       1.400000 -2.900000
75%       4.250000 -2.100000
max       7.100000 -1.300000
```

On non-numeric data, describe produces alternate summary statistics:

```python
In [205]: obj = Series(["a", "a", "b", "c"] * 4)
In [206]: obj.describe()
Out[206]:
count     16
unique     3
top        a
freq       8
dtype: object
```

See Table 5-10 for a full list of summary statistics and related methods.

Table 5-10. Descriptive and summary statistics

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Number of non-NA values</td>
</tr>
<tr>
<td>describe</td>
<td>Compute set of summary statistics for Series or each DataFrame column</td>
</tr>
<tr>
<td>min, max</td>
<td>Compute minimum and maximum values</td>
</tr>
<tr>
<td>argmin, argmax</td>
<td>Compute index locations (integers) at which minimum or maximum value obtained, respectively</td>
</tr>
<tr>
<td>idxmin, idxmax</td>
<td>Compute index values at which minimum or maximum value obtained, respectively</td>
</tr>
<tr>
<td>quantile</td>
<td>Compute sample quantile ranging from 0 to 1</td>
</tr>
<tr>
<td>sum</td>
<td>Sum of values</td>
</tr>
<tr>
<td>mean</td>
<td>Mean of values</td>
</tr>
<tr>
<td>median</td>
<td>Arithmetic median (50% quantile) of values</td>
</tr>
<tr>
<td>mad</td>
<td>Mean absolute deviation from mean value</td>
</tr>
<tr>
<td>var</td>
<td>Sample variance of values</td>
</tr>
<tr>
<td>std</td>
<td>Sample standard deviation of values</td>
</tr>
<tr>
<td>skew</td>
<td>Sample skewness (3rd moment) of values</td>
</tr>
<tr>
<td>kurt</td>
<td>Sample kurtosis (4th moment) of values</td>
</tr>
<tr>
<td>cumsum</td>
<td>Cumulative sum of values</td>
</tr>
<tr>
<td>cummin, cummax</td>
<td>Cumulative minimum or maximum of values, respectively</td>
</tr>
<tr>
<td>cumprod</td>
<td>Cumulative product of values</td>
</tr>
<tr>
<td>diff</td>
<td>Compute 1st arithmetic difference (useful for time series)</td>
</tr>
<tr>
<td>pct_change</td>
<td>Compute percent changes</td>
</tr>
</tbody>
</table>

Correlation and Covariance

Some summary statistics, like correlation and covariance, are computed from pairs of arguments. Let's consider some DataFrames of stock prices and volumes obtained from Yahoo! Finance:

```python
import pandas.io.data as web
all_data = {}
for ticker in ["AAPL", "IBM", "MSFT", "GOOG"]:
    all_data[ticker] = web.get_data_yahoo(ticker)
price = DataFrame({tic: data["Adj Close"]
for tic, data in all_data.iteritems()})
volume = DataFrame({tic: data["Volume"]
for tic, data in all_data.iteritems()})
```

I now compute percent changes of the prices:

```python
In [208]: returns = price.pct_change()
In [209]: returns.tail()
Out[209]:
          AAPL       GOOG       IBM       MSFT
Date
2014-07-07  0.020632 -0.004241 -0.002599  0.004545
2014-07-08 -0.006460 -0.019167 -0.004361 -0.005001
2014-07-09  0.000420  0.008738  0.006410 -0.002633
2014-07-10 -0.003669 -0.008645 -0.003821  0.000480
2014-07-11  0.001894  0.014148  0.001598  0.009595
```

The `corr` method of Series computes the correlation of the overlapping, non-NA, aligned-by-index values in two Series. Relatedly, `cov` computes the covariance:

```python
In [210]: returns.MSFT.corr(returns.IBM)
Out[210]: 0.51360438136345077
In [211]: returns.MSFT.cov(returns.IBM)
Out[211]: 8.4825099973219876e-05
```

DataFrame's `corr` and `cov` methods, on the other hand, return a full correlation or covariance matrix as a DataFrame, respectively:
Unique Values and Value Counts

• `unique` returns an array with only the unique values (no index)
  
  ```python
  s = Series(['c','a','d','a','a','b','b','c','c'])
  s.unique() # array(['c', 'a', 'd', 'b'])
  ```

• Data Frames use `drop_duplicates`

• `value_counts` returns a Series with index frequencies:
  
  ```python
  s.value_counts() # Series({'c': 3,'a': 3,'b': 2,'d': 1})
  ```
Handling Missing Data

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dropna</td>
<td>Filter axis labels based on whether values for each label have missing data, with varying thresholds for how much missing data to tolerate.</td>
</tr>
<tr>
<td>fillna</td>
<td>Fill in missing data with some value or using an interpolation method such as 'ffill' or 'bfill'.</td>
</tr>
<tr>
<td>isnull</td>
<td>Return like-type object containing boolean values indicating which values are missing / NA.</td>
</tr>
<tr>
<td>nonnull</td>
<td>Negation of isnull.</td>
</tr>
</tbody>
</table>
What if data isn't correct/trustworthy/in the right format?
Dirty Data
Geolocation Errors

• Maxmind helps companies determine where users are located based on IP address

• "How a quiet Kansas home wound up with 600 million IP addresses and a world of trouble" [Washington Post, 2016]
Numeric Outliers

ages of employees (US)

- median 37
- mean 58.52632
- variance 9252.041

[J. Hellerstein via J. Canny et al.]
FINDINGS

we got about the future of the data science, the most salient takeaway was how excited our respondents were about the evolution of the field. They cited things in their own practice, how they saw their jobs getting more interesting and less repetitive, all while expressing a real and broad enthusiasm about the value of the work in their organization.

As data science becomes more commonplace and simultaneously a bit demystified, we expect this trend to continue as well. After all, last year’s respondents were just as excited about their work (about 79% were “satisfied” or better).

How a Data Scientist Spends Their Day

Here's where the popular view of data scientists diverges pretty significantly from reality. Generally, we think of data scientists building algorithms, exploring data, and doing predictive analysis. That's actually not what they spend most of their time doing, however.

As you can see from the chart above, 3 out of every 5 data scientists we surveyed actually spend the most time cleaning and organizing data. You may have heard this referred to as “data wrangling” or compared to digital janitor work. Everything from list verification to removing commas to debugging databases— that time adds up and it adds up immensely. Messy data is by far the more time-consuming aspect of the typical data scientist’s work flow. And nearly 60% said they simply spent too much time doing it.

What data scientists spend the most time doing

- Building training sets: 3%
- Cleaning and organizing data: 60%
- Collecting data sets: 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%

This takes a lot of time!

[CrowdFlower Data Science Report, 2016]
Why That's a Problem

Simply put, data wrangling isn't fun. It takes forever. In fact, a few years back, the New York Times estimated that up to 80% of a data scientist's time is spent doing this sort of work.

Here, it's necessary to point out that data cleaning is incredibly important. You can't do the sort of work data scientists truly enjoy doing with messy data. It needs to be cleaned, labeled, and enriched before you can trust the output.

The problem here is two-fold. One: data scientists simply don't like doing this kind of work, and, as mentioned, this kind of work takes up most of their time. We asked our respondents what was the least enjoyable part of their job.

They had this to say:

Note how those last two charts mirror each other. The things data scientists do most are the things they enjoy least. Last year, we found that respondents far prefer doing the more creative, interesting parts of their job, things like predictive analysis and mining data for patterns. That's where the real value comes. But again, you simply can't do that work unless the data is properly labeled. And nobody likes labeling data.

Do Data Scientists Have What They Need?

With a shortage of data scientists out there in the world, we wanted to find out if they thought they were properly supported in their job. After all, when you need more data scientists, you'll often find a single person doing the work of several.

What's the least enjoyable part of data science?

- Building training sets: 10%
- Cleaning and organizing data: 57%
- Collecting data sets: 21%
- Mining data for patterns: 3%
- Refining algorithms: 4%
- Other: 5%

[CrowdFlower Data Science Report, 2016]
Dirty Data: Statistician's View

• Some process produces the data
• Want a model but have non-ideal samples:
  - Distortion: some samples corrupted by a process
  - Selection bias: likelihood of a sample depends on its value
  - Left and right censorship: users come and go from scrutiny
  - Dependence: samples are not independent (e.g. social networks)
• You can add/augment models for different problems, but cannot model everything
• Trade-off between accuracy and simplicity

[J. Canny et al.]
Dirty Data: Database Expert's View

• Got a dataset
• Some values are missing, corrupted, wrong, duplicated
• Results are absolute (relational model)
• Better answers come from improving the quality of values in the dataset
Dirty Data: Domain Expert's View

- Data doesn't look right
- Answer doesn't look right
- What happened?
- Domain experts carry an implicit model of the data they test against
- You don't always need to be a domain expert to do this
  - Can a person run 50 miles an hour?
  - Can a mountain on Earth be 50,000 feet above sea level?
  - Use common sense
Dirty Data: Data Scientist's View

• Combination of the previous three views
• All of the views present problems with the data
• The goal may dictate the solutions:
  - Median value: don't worry too much about crazy outliers
  - Generally, aggregation is less susceptible by numeric errors
  - Be careful, the data may be correct…
Be careful how you detect dirty data

• The appearance of a hole in the earth’s ozone layer over Antarctica, first detected in 1976, was so unexpected that scientists didn’t pay attention to what their instruments were telling them; they thought their instruments were malfunctioning.

– National Center for Atmospheric Research
Where does dirty data originate?

- Source data is bad, e.g. person entered it incorrectly
- Transformations corrupt the data, e.g. certain values processed incorrectly due to a software bug
- Integration of different datasets causes problems
- Error propagation: one error is magnified
Types of Dirty Data Problems

- **Separator Issues**: e.g. CSV without respecting double quotes
  - 12, 13, "Doe, John", 45

- **Naming Conventions**: NYC vs. New York

- **Missing required fields**, e.g. key

- **Different representations**: 2 vs. two

- **Truncated data**: "Janice Keihanaikukauakahihiuliheekahaunaele" becomes "Janice Keihanaikukauakahihiuliheek" on Hawaii license

- **Redundant records**: may be exactly the same or have some overlap

- **Formatting issues**: 2017–11–07 vs. 07/11/2017 vs. 11/07/2017

---

[J. Canny et al.]
Data Wrangling

• Data wrangling: transform raw data to a more meaningful format that can be better analyzed
• Data cleaning: getting rid of inaccurate data
• Data transformations: changing the data from one representation to another
• Data reshaping: reorganizing the data
• Data merging: combining two datasets
Data Cleaning
Wrangler: Interactive Visual Specification of Data Transformation Scripts

S. Kandel, A. Paepcke, J. Hellerstein, J. Heer
Wrangler

• Data cleaning takes a lot of **time** and **human effort**
• "Tedium is the message"
• Repeating this process on multiple data sets is even worse!
• Solution:
  - interactive interface (mixed-initiative)
  - transformation language with natural language "translations"
  - suggestions + "programming by demonstration"
Your Critique/Questions
Example Critique

• Summary: Wrangler tackles data wrangling tasks by combining a language for specifying operations with an interface allowing users to specify the types of changes they are interested; the system can then generate suggested operations and demonstrates them on demand.

• Critique: The suggestions may lead to states that a user cannot recover from easily. Suppose a suggestion looks like it works well, but a user later realizes was incorrect. They can backtrack, but it's often unclear where to and which other path to take. In addition, a user has to have some idea of the constructs of the language in order to edit parameters. Without a good idea of the impact of the parameters, the work may become as tedious as manual correction. Perhaps a more example-based strategy could help.
Previous Work: Potter's Wheel

• V. Raman and J. Hellerstein, 2001
• Defines structure extractions for identifying fields
• Defines transformations on the data
• Allows user interaction
### Potter's Wheel: Structure Extraction

<table>
<thead>
<tr>
<th>Example Column Value (Example erroneous values)</th>
<th># Structures Enumerated</th>
<th>Final Structure Chosen (Punc = Punctuation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-60</td>
<td>5</td>
<td>Integer</td>
</tr>
<tr>
<td>UNITED, DELTA, AMERICAN etc.</td>
<td>5</td>
<td>IspellWord</td>
</tr>
<tr>
<td>SFO, LAX etc. (JFK to OAK)</td>
<td>12</td>
<td>AllCapsWord</td>
</tr>
<tr>
<td>1998/01/12</td>
<td>9</td>
<td>Int Punc(/) Int Punc(/) Int</td>
</tr>
<tr>
<td>M, Tu, Thu etc.</td>
<td>5</td>
<td>Capitalized Word</td>
</tr>
<tr>
<td>06:22</td>
<td>5</td>
<td>Int(len 2) Punc(:) Int(len 2)</td>
</tr>
<tr>
<td>12.8.15.147 (ferret03.webtop.com)</td>
<td>9</td>
<td>Double Punc(') Double</td>
</tr>
<tr>
<td>”GET\b (\b)</td>
<td>5</td>
<td>Punc(”&quot;) IspellWord Punc()</td>
</tr>
<tr>
<td>/postmodern/lecs/xia/sld013.htm</td>
<td>4</td>
<td>ξ*</td>
</tr>
<tr>
<td>HTTP</td>
<td>3</td>
<td>AllCapsWord(HTTP)</td>
</tr>
<tr>
<td>/1.0</td>
<td>6</td>
<td>Punc(/) Double(1.0)</td>
</tr>
</tbody>
</table>

[V. Raman and J. Hellerstein, 2001]
Potter's Wheel: Transforms

<table>
<thead>
<tr>
<th>Transform</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format</strong></td>
<td>( \phi(R, i, f) = {(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, f(a_i)) \mid (a_1, \ldots, a_n) \in R} )</td>
</tr>
<tr>
<td><strong>Add</strong></td>
<td>( \alpha(R, x) = {(a_1, \ldots, a_n, x) \mid (a_1, \ldots, a_n) \in R} )</td>
</tr>
<tr>
<td><strong>Drop</strong></td>
<td>( \pi(R, i) = {(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R} )</td>
</tr>
<tr>
<td><strong>Copy</strong></td>
<td>( \kappa((a_1, \ldots, a_n), i) = {(a_1, \ldots, a_n, a_i) \mid (a_1, \ldots, a_n) \in R} )</td>
</tr>
<tr>
<td><strong>Merge</strong></td>
<td>( \mu((a_1, \ldots, a_n), i, j, \text{glue}) = {(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n, a_i \oplus \text{glue} \oplus a_j) \mid (a_1, \ldots, a_n) \in R} )</td>
</tr>
<tr>
<td><strong>Split</strong></td>
<td>( \omega((a_1, \ldots, a_n), i, \text{splitter}) = {(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, \text{left}(a_i, \text{splitter}), \text{right}(a_i, \text{splitter})) \mid (a_1, \ldots, a_n) \in R} )</td>
</tr>
<tr>
<td><strong>Divide</strong></td>
<td>( \delta((a_1, \ldots, a_n), i, \text{pred}) = {(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, a_i, \text{null}) \mid (a_1, \ldots, a_n) \in R \land \text{pred}(a_i)} \cup {(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, \text{null}, a_i) \mid (a_1, \ldots, a_n) \in R \land \neg \text{pred}(a_i)} )</td>
</tr>
<tr>
<td><strong>Fold</strong></td>
<td>( \lambda(R, i_1, i_2, \ldots, i_k) = {(a_1, \ldots, a_{i_1-1}, a_{i_1+1}, \ldots, a_{i_2-1}, a_{i_2+1}, \ldots, a_{i_k-1}, a_{i_k+1}, \ldots, a_n, a_{i_1}) \mid (a_1, a_n) \in R \land 1 \leq l \leq k} )</td>
</tr>
<tr>
<td><strong>Select</strong></td>
<td>( \sigma(R, \text{pred}) = {(a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R \land \text{pred}(a_1, \ldots, a_n)} )</td>
</tr>
</tbody>
</table>

**Notation:** \( R \) is a relation with \( n \) columns. \( i, j \) are column indices and \( a_i \) represents the value of a column in a row. \( x \) and \( \text{glue} \) are values. \( f \) is a function mapping values to values. \( x \oplus y \) concatenates \( x \) and \( y \). \( \text{splitter} \) is a position in a string or a regular expression, \( \text{left}(x, \text{splitter}) \) is the left part of \( x \) after splitting by \( \text{splitter} \). \( \text{pred} \) is a function returning a boolean.

[V. Raman and J. Hellerstein, 2001]
Potter's Wheel: Example

<table>
<thead>
<tr>
<th>Anna</th>
<th>Davis</th>
<th>Stewart, Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Joan</td>
<td>Marsh</td>
<td></td>
</tr>
</tbody>
</table>

Format
'(.*), (.*)' to '

Split at '

2 Merges

<table>
<thead>
<tr>
<th>Bob</th>
<th>Stewart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Davis</td>
</tr>
<tr>
<td>Jerry</td>
<td>Dole</td>
</tr>
<tr>
<td>Joan</td>
<td>Marsh</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bob</th>
<th>Stewart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Davis</td>
</tr>
<tr>
<td>Jerry</td>
<td>Dole</td>
</tr>
<tr>
<td>Joan</td>
<td>Marsh</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bob</th>
<th>Stewart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Davis</td>
</tr>
<tr>
<td>Jerry</td>
<td>Dole</td>
</tr>
<tr>
<td>Joan</td>
<td>Marsh</td>
</tr>
</tbody>
</table>

[V. Raman and J. Hellerstein, 2001]
**Potter's Wheel: Inferring Structure from Examples**

<table>
<thead>
<tr>
<th>Example Values Split By User</th>
<th>Inferred Structure</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taylor, Jane</td>
<td>$52,072</td>
<td>(&lt;ξ* &gt; &lt;', Money &gt;)</td>
</tr>
<tr>
<td>Blair, John</td>
<td>$73,238</td>
<td>(&lt;len 3 identifier&gt; &lt;ξ* &gt;</td>
</tr>
<tr>
<td>Tony Smith</td>
<td>$1,00,533</td>
<td>&lt; len 3 identifier &gt; )</td>
</tr>
<tr>
<td>MAA [to] SIN JKF [to] SFO LAX [—] ORD SEA [/] OAK</td>
<td>(&lt;number ξ* &gt; &lt;', word &gt;</td>
<td>Parsing is easy because of consistent delimiter.</td>
</tr>
<tr>
<td>321 Blake #7 [, Berkeley</td>
<td>, CA 94720 719 MLK Road [, Fremont</td>
<td>, CA 95743</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;', (2 letter word) (5 letter integer)&gt; )</td>
</tr>
</tbody>
</table>

[V. Raman and J. Hellerstein, 2001]
Wrangler Transformation Language

• Based on Potter's Wheel
• Map: Delete, Extract, Cut, Split, Update
• Lookup/join: Use external data (e.g. from zipcode→state)
• Reshape: Fold and Unfold (aka pivot)
• Positional: Fill and lag
• Sorting, aggregation, key generation, schema transforms
Interface

• Automated Transformation Suggestions
• Editable Natural Language Explanations

- Fill Bangladesh by copying values from above
- Fill Bangladesh by averaging the 5 values from above
- Fill Bangladesh by averaging values from above

• Visual Transformation Previews
• Transformation History
Automation from past actions

- Infer parameter sets from user interaction
- Generating transforms
- Ranking and ordering transformations:
  - Based on user preferences, difficulty, and corpus frequency
  - Sort transforms by type and diversify suggestions

![Reported crime in Alabama](image)

- 'Alabama' → {'Alabama', 'word'}
- 'in' → {'in', 'word', 'lowercase'}
- ' ' → {' '}
- {'in', ' '}, {'Alabama'}, () → {'in', 'Alabama'}
- ('Alabama'), () → {'in', 'Alabama'}
- ('Alabama'), () → {'in', 'Alabama'}
- (lowercase, ' '), (Alabama), () → /[a-z]+ (Alabama)/

[S. Kandel et al., 2011]
Data Wrangler Demo

- [http://vis.stanford.edu/wrangler/app/](http://vis.stanford.edu/wrangler/app/)

![Transform Script](Image)

<table>
<thead>
<tr>
<th>Transform Script</th>
<th>Import</th>
<th>Export</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Split data repeatedly on newline into rows</td>
<td></td>
<td></td>
</tr>
<tr>
<td>▶ Split split repeatedly on ','</td>
<td></td>
<td></td>
</tr>
<tr>
<td>▶ Promote row 0 to header</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Text</th>
<th>Columns</th>
<th>Rows</th>
<th>Table</th>
<th>Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Delete **row 7**

Delete **empty rows**

Fill **row 7 by copying** values from **above**

<table>
<thead>
<tr>
<th>Year</th>
<th>Property_crime_rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reported crime in Alabama</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2004</td>
</tr>
<tr>
<td>3</td>
<td>2005</td>
</tr>
<tr>
<td>4</td>
<td>2006</td>
</tr>
<tr>
<td>5</td>
<td>2007</td>
</tr>
<tr>
<td>6</td>
<td>2008</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Reported crime in Alaska</td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2004</td>
</tr>
<tr>
<td>11</td>
<td>2005</td>
</tr>
<tr>
<td>12</td>
<td>2006</td>
</tr>
</tbody>
</table>