Advanced Data Management (CSCI 680/490)

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`
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 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          5000
dtype: float64

In [30]: obj3 + obj4
Out[30]:
Ohio          35000
Oregon        16000
Texas         71000
Utah          5000
California    NaN
dtype: float64
```

- 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  1  2
   Ohio  3  4  5
   Texas 6  7  8
   Name: Oregon, dtype: float64
```

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

- DataFrame 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]:                           Out[172]:
  d  a  b  c                          a  b  c  d
  one    4  5  6  7                   three  1  2  3  0
  three  0  1  2  3                   one    5  6  7  4
  ```

- 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
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
• Information 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
```

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

In [204]: df.describe()
Out[204]:
<table>
<thead>
<tr>
<th></th>
<th>one</th>
<th>two</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>3.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>mean</td>
<td>3.083333</td>
<td>-2.900000</td>
</tr>
<tr>
<td>std</td>
<td>3.493685</td>
<td>2.262742</td>
</tr>
<tr>
<td>min</td>
<td>0.750000</td>
<td>-4.500000</td>
</tr>
<tr>
<td>25%</td>
<td>1.075000</td>
<td>-3.700000</td>
</tr>
<tr>
<td>50%</td>
<td>1.400000</td>
<td>-2.900000</td>
</tr>
<tr>
<td>75%</td>
<td>4.250000</td>
<td>-2.100000</td>
</tr>
<tr>
<td>max</td>
<td>7.100000</td>
<td>-1.300000</td>
</tr>
</tbody>
</table>

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    |

In [208]: returns = price.pct_change()
In [209]: returns.tail()
Out[209]:
<table>
<thead>
<tr>
<th></th>
<th>AAPL</th>
<th>GOOG</th>
<th>IBM</th>
<th>MSFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>2014-07-07</td>
<td>0.020632</td>
<td>-0.004241</td>
<td>-0.002599</td>
</tr>
<tr>
<td></td>
<td>2014-07-08</td>
<td>-0.006460</td>
<td>-0.019167</td>
<td>-0.004361</td>
</tr>
<tr>
<td></td>
<td>2014-07-09</td>
<td>0.000420</td>
<td>0.008738</td>
<td>0.006410</td>
</tr>
<tr>
<td></td>
<td>2014-07-10</td>
<td>-0.003669</td>
<td>-0.008645</td>
<td>-0.003821</td>
</tr>
<tr>
<td></td>
<td>2014-07-11</td>
<td>0.001894</td>
<td>0.014148</td>
<td>0.001598</td>
</tr>
</tbody>
</table>

In [210]: returns.MSFT.corr(returns.IBM)
Out[210]: 0.51360438136345077

In [211]: returns.MSFT.cov(returns.IBM)
Out[211]: 8.4825099973219876e-05

---

[W. McKinney, Python for Data Analysis]
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>notnull</td>
<td>Negation of isnull.</td>
</tr>
</tbody>
</table>

[W. McKinney, Python for Data Analysis]
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.]
This takes a lot of time!

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%

[CrowdFlower Data Science Report, 2016]
...and it isn't the most fun thing to do

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
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

[J. Canny et al.]
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

[J. Canny et al.]
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 Keihanaikukauakahihuliheekahaunaele" becomes "Janice Keihanaikukauakahihuliheek" 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(”’) 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>

*(Example erroneous values)*

---

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

<table>
<thead>
<tr>
<th>Transform</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format $\phi(R, i, f)$</td>
<td>${ (a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n, f(a_i)) \mid (a_1, \ldots, a_n) \in R }$</td>
</tr>
<tr>
<td>Add $\alpha(R, x)$</td>
<td>${ (a_1, \ldots, a_n, x) \mid (a_1, \ldots, a_n) \in R }$</td>
</tr>
<tr>
<td>Drop $\pi(R, i)$</td>
<td>${ (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R }$</td>
</tr>
<tr>
<td>Copy $\kappa((a_1, \ldots, a_n), i)$</td>
<td>${ (a_1, \ldots, a_n, a_i) \mid (a_1, \ldots, a_n) \in R }$</td>
</tr>
<tr>
<td>Merge $\mu((a_1, \ldots, a_n), i, j, \text{glue})$</td>
<td>${ (a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_j, a_{j+1}, \ldots, a_n, a_i \oplus \text{glue} \oplus a_j) \mid (a_1, \ldots, a_n) \in R }$</td>
</tr>
<tr>
<td>Split $\omega((a_1, \ldots, a_n), i, \text{splitter})$</td>
<td>${ (a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n, \text{left}(a_i, \text{splitter}), \text{right}(a_i, \text{splitter})) \mid (a_1, \ldots, a_n) \in R }$</td>
</tr>
<tr>
<td>Divide $\delta((a_1, \ldots, a_n), i, \text{pred})$</td>
<td>${ (a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n, \text{null}) \mid (a_1, \ldots, a_n) \in R \wedge \text{pred}(a_i) } \cup$</td>
</tr>
<tr>
<td></td>
<td>${ (a_1, \ldots, a_i, a_{i+1}, \ldots, a_n, \text{null}, a_i) \mid (a_1, \ldots, a_n) \in R \wedge \neg\text{pred}(a_i) }$</td>
</tr>
<tr>
<td>Fold $\lambda(R, i_1, i_2, \ldots, i_k)$</td>
<td>${ (a_1, \ldots, a_{i_1-1}, a_{i_1}, a_{i_1+1}, \ldots, a_{i_2-1}, a_{i_2}, a_{i_2+1}, \ldots, a_{i_k-1}, a_{i_k}, a_{i_k+1}, \ldots, a_n, a_{i_i}) \mid$</td>
</tr>
<tr>
<td></td>
<td>$(a_1, \ldots, a_n) \in R \wedge 1 \leq l \leq k }$</td>
</tr>
<tr>
<td>Select $\sigma(R, \text{pred})$</td>
<td>${ (a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R \wedge \text{pred}(a_1, \ldots, a_n) }$</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 '
2 \1'

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

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>Anna</th>
<th>Davis</th>
<th>Bob</th>
<th>Stewart</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Joan</td>
<td>Marsh</td>
<td></td>
<td></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>Blair, John</td>
</tr>
<tr>
<td>(</td>
<td>is user specified split position)</td>
<td>(&lt;\xi^* &gt; &lt;', 'Money &gt;)</td>
</tr>
<tr>
<td>MAA</td>
<td>to</td>
<td>SIN</td>
</tr>
</tbody>
</table>
| 321 Blake #7 | , Berkeley | , CA 94720 | 719 MLK Road | , Fremont | , CA 95743 | \(<number \xi^* > <', 'word> <', (2 letter word) (5 letter integer)\>) | Parsing is easy because of consistent delimiter. | [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
- Visual Transformation Previews
- Transformation History

0. Fill Bangladesh by copying values from above

1. Fill Bangladesh by interpolating values from above

2. Fill Bangladesh by averaging the 5 values from above

3. Automated Transformation Suggestions

4. Editable Natural Language Descriptions

5. Interface

6. Visual Transformation Previews

7. Transformation History

8. [S. Kandel et al., 2011]
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

<table>
<thead>
<tr>
<th>(a)</th>
<th>Reported crime in <strong>Alabama</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>before:</td>
<td>{('in', ' ')}</td>
</tr>
<tr>
<td>(b) selection:</td>
<td>{‘Alabama’}</td>
</tr>
<tr>
<td>after:</td>
<td>{}</td>
</tr>
<tr>
<td>before:</td>
<td>{(' '), ('in', ' '), (word, ' '), (lowercase, ' ')}</td>
</tr>
<tr>
<td>(c) selection:</td>
<td>{('Alabama'), (word)}</td>
</tr>
<tr>
<td>after:</td>
<td>{}</td>
</tr>
<tr>
<td>{(),('Alabama'),()}</td>
<td>{(),(word),()}</td>
</tr>
<tr>
<td>{(' '),O,O}</td>
<td>{(word, ' '),O,O}</td>
</tr>
<tr>
<td>{(' '),('Alabama'),()}</td>
<td>{((word, ' '),('Alabama'),())}</td>
</tr>
<tr>
<td>(d) {('in', ' '),O,O}</td>
<td>{(lowercase, ' '),O,O}</td>
</tr>
<tr>
<td>{('in', ' '),('Alabama'),()}</td>
<td>{(lowercase, ' '),('Alabama'),()}</td>
</tr>
<tr>
<td>{('in', ' '),word),()}</td>
<td>{(lowercase, ' '),word),()}</td>
</tr>
<tr>
<td>(e) {((lowercase, ' '),('Alabama'),())} → /[a-z]+ (Alabama)/</td>
<td></td>
</tr>
</tbody>
</table>
Data Wrangler Demo

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

## Transform Script

- Split **data repeatedly** on newline into rows
- Split **split repeatedly** on ','
- Promote **row 0** to header
- Delete **row 7**
- Delete **empty rows**
- Fill **row 7** by copying values from above

### Table

<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>0</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>0</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>
Evaluation

• Compare with Excel

• Tests:
  - Extract text from a single string entry
  - Fill in missing values with estimates
  - Reshape tables

• Allowed users to ask questions about Excel, not Wrangler

• Found significant effect of tool and users found previews and suggestions helpful

• Complaint: No manual fallback, make implications of user choices more obvious for users
COMPARATIVE EVALUATION WITH EXCEL

We acknowledge that this is not an ideal cleaning solution for the market or the needs of the average user. For Wrangler, our approach is more flexible and easier to use, but Excel still has some limitations. In contrast, Wrangler offers a more intuitive way to perform these tasks. Overall, Wrangler is a more powerful tool for data cleaning, especially for users who are not familiar with programming.
Improvements in Prediction

Update suggestions when given more information

[Heer et al., 2015]
Data Wrangling Tasks

- Unboxing: Discovery & Assessment: What's in there? (types, distribution)
- Structuring: Restructure data (table, nested data, pivot tables)
- Cleaning: does data match expectations (often involves user)
- Enriching & Blending: Adding new data
- Optimizing & Publishing: Structure for storage or visualization

[J. M. Hellerstein et al., 2018]
Differences with Extract-Transform-Load (ETL)

• ETL:
  - Who: IT Professionals
  - Why: Create static data pipeline
  - What: Structured data
  - Where: Data centers

• "Modern Data Preparation":
  - Who: Analysts
  - Why: Solve problems by designing recipes to use data
  - What: Original, custom data blended with other data
  - Where: Cloud, desktop

[J. M. Hellerstein et al., 2018]
Trifacta Wrangler