Advanced Data Management (CSCI 490/680)

Data Cleaning

Dr. David Koop

Comma-separated values (CSV) Format

- Comma is a field separator, newlines denote records
  - ```
    a,b,c,d,message
    1,2,3,4,hello
    5,6,7,8,world
    9,10,11,12,foo
    ```
- May have a header (`a,b,c,d,message`), but not required
- No type information: we do not know what the columns are (numbers, strings, floating point, etc.)
  - Default: just keep everything as a string
  - Type inference: Figure out the type to make each column based on values
- What about commas in a value? → double quotes
    ```

Reading & Writing Data in Pandas

Format	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

[https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html]

read_csv

- Convenient method to read csv files
- Lots of different options to help get data into the desired format
- Basic: `df = pd.read_csv(fname)`
- Parameters:
  - `path`: where to read the data from
  - `sep` (or `delimiter`): the delimiter (`','`, `' '`, `'\t'`, `'\s+'`)
  - `header`: if `None`, no header
  - `index_col`: which column to use as the row index
  - `names`: list of header names (e.g. if the file has no header)
  - `skiprows`: number of list of lines to skip

Writing CSV data with pandas

- Basic: `df.to_csv(<fname>)`
- Change delimiter with sep kwarg:
  - `df.to_csv('example.dsv', sep='|')`
- Change missing value representation
  - `df.to_csv('example.dsv', na_rep='NULL')`
- Don't write row or column labels:
  - `df.to_csv('example.csv', index=False, header=False)`
- Series may also be written to csv

JavaScript Object Notation (JSON)

- A format for web data
- Looks very similar to python dictionaries and lists
- Example:

```
- {"name": "Wes",
   "places_lived": ["United States", "Spain", "Germany"],
   "pet": null,
   "siblings": [{"name": "Scott", "age": 25, "pet": "Zuko"},
                {"name": "Katie", "age": 33, "pet": "Cisco"}] }
```

- Only contains literals (no variables) but allows null
- Values: strings, arrays, dictionaries, numbers, booleans, or null
  - Dictionary keys must be strings
  - Quotation marks help differentiate string or numeric values

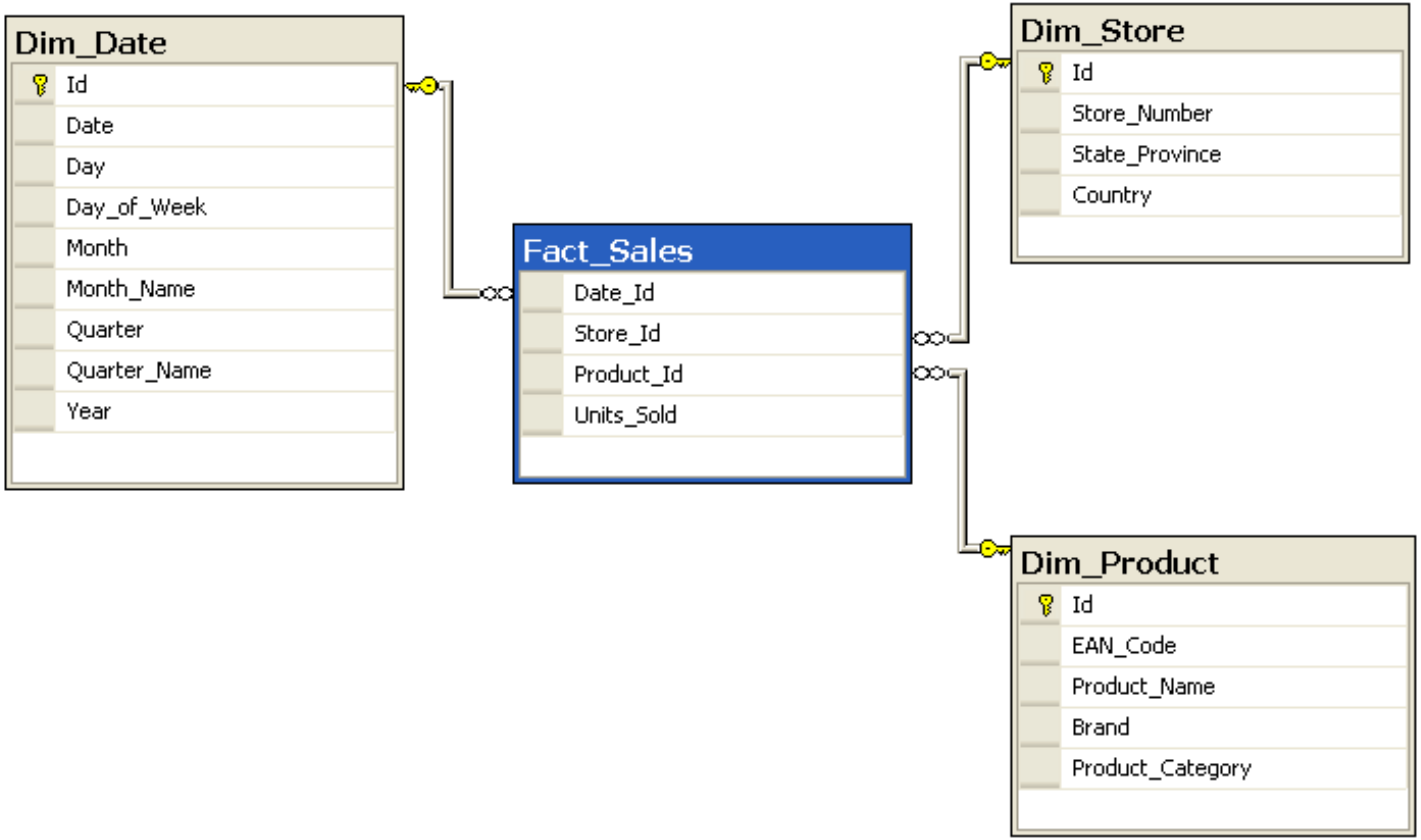
JSON Orientation

- Indication of expected JSON string format. Compatible JSON strings can be produced by `to_json()` with a corresponding orient value. The set of possible orients is:
  - `split`: dict like `{index -> [index],`
    `columns -> [columns],`
    `data -> [values]}`
  - `records`: list like `[{column -> value, ... , column -> value}]`
  - `index`: dict like `{index -> {column -> value}}`
  - `columns`: dict like `{column -> {index -> value}}`
  - `values`: just the values array

Binary Formats

- CSV, JSON, and XML are all text formats
- What is a binary format?
- Pickle: Python's built-in serialization
- HDF5: Library for storing large scientific data
  - Hierarchical Data Format, supports **compression**
  - Interfaces in C, Java, MATLAB, etc.
  - Use `pd.HDFStore` to access, shortcuts: `read_hdf/to_hdf`,
- Excel: need to specify sheet when a spreadsheet has multiple sheets
  - `pd.ExcelFile` or `pd.read_excel`
- Parquet: big data format, can use compression

Databases



[Wikipedia]

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.]

Dirty Data: Data Scientist's View

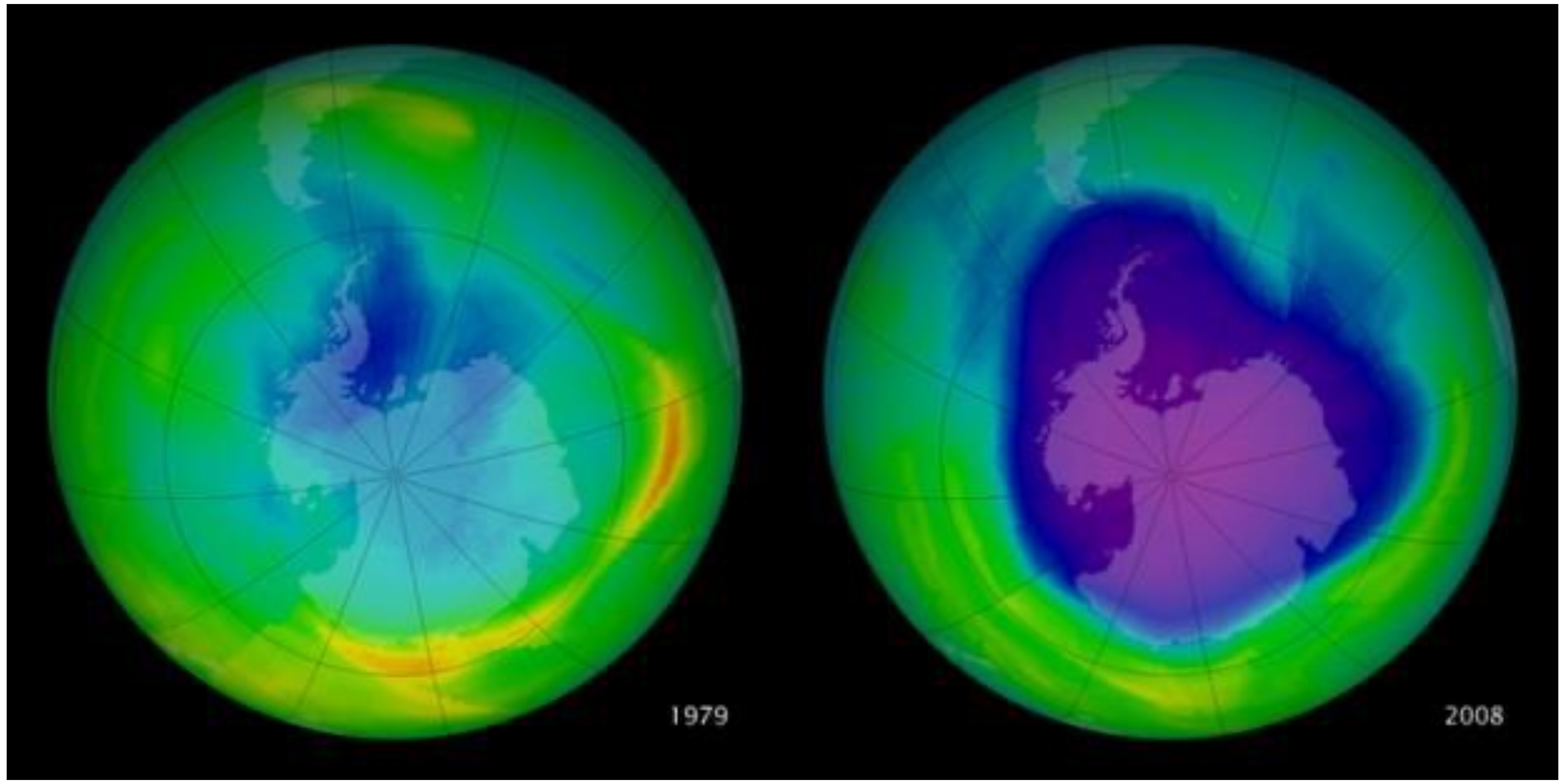
- Combination of:
  - Statistician's View: data has non-ideal samples for model
  - Database Expert's View: missing data, corrupted data
  - Domain Expert's View: data doesn't pass the smell test
- All of the views present problems with the data
- The goal may dictate the solutions:
  - Median value: don't worry too much about crazy outliers
  - Generally, aggregation is less susceptible by numeric errors
  - Be careful, the data may be correct…

[J. Canny et al.]

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



[Wikimedia]

Assignment 2

- Same data as A1, different version of the dataset
- Dealing with the raw data now
- Same questions as A1, but use pandas
- CS680 students + some questions about problems with the data

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"

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

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

[V. Raman and J. Hellerstein, 2001]

Potter's Wheel: Transforms

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

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 glue are values. f is a function mapping values to values. $x \oplus y$ concatenates x and y. splitter is a position in a string or a regular expression, left$(x, \text{splitter})$ is the left part of x after splitting by splitter. pred is a function returning a boolean.

[V. Raman and J. Hellerstein, 2001]

Potter's Wheel: Example

		Stewart,Bob
Anna	Davis	
		Dole,Jerry
Joan	Marsh	

Format
'(.*), (.*)' to '\2 \1'

		Bob Stewart
Anna	Davis	
		Jerry Dole
Joan	Marsh	

Split at ' '

		Bob	Stewart
Anna	Davis		
		Jerry	Dole
Joan	Marsh		

2 Merges

Bob	Stewart
Anna	Davis
Jerry	Dole
Joan	Marsh

[V. Raman and J. Hellerstein, 2001]

Potter's Wheel: Inferring Structure from Examples

Example Values Split By User (| is user specified split position)	Inferred Structure	Comments
Taylor, Jane |, $52,072 Blair, John |, $73,238 Tony Smith |, $1,00,533	$(< \xi^* > < \text{','}\ Money >)$	Parsing is doable despite no good delimiter. A *regular expression* domain can infer a structure of $[0-9,]* for last component.
MAA |to| SIN JFK |to| SFO LAX |–| ORD SEA |/| OAK	$(<len\ 3\ identifier> < \xi^* > < len\ 3\ identifier>)$	Parsing is possible despite multiple delimiters.
321 Blake #7 |, Berkeley |, CA 94720 719 MLK Road |, Fremont |, CA 95743	$(<number\ \ \xi^* > < \text{','}\ \ word> <\text{','}\ (2\ letter\ word)\ (5\ letter\ integer)>)$	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
  - Fill **Bangladesh** by **copying** values from **above**
  - Fill **Bangladesh** by [averaging / ✓ copying / interpolating] values from **above**
  - Fill **Bangladesh** by **averaging** the **5** values from **above**
- Visual Transformation Previews
- Transformation History

split	split1	split2	split3	split4
	2004	2004	2004	2003
STATE	Participation Rate 2004	Mean SAT I Verbal	Mean SAT I Math	Participation Rate
New York	87	497	510	82
Connecticut	85	515	515	84
Massachusetts	85	518	523	82
New Jersey	83	501	514	85
New Hampshire	80	522	521	75
D.C.	77	489	476	77
Maine	76	505	501	70
Pennsylvania	74	501	502	73
Delaware	73	500	499	73
Georgia	73	494	493	66

split	fold	fold1	value
New York	2004	Participation Rate 2004	87
New York	2004	Mean SAT I Verbal	497
New York	2004	Mean SAT I Math	510
New York	2003	Participation Rate 2003	82
New York	2003	Mean SAT I Verbal	496
New York	2003	Mean SAT I Math	510
Connecticut	2004	Participation Rate 2004	85
Connecticut	2004	Mean SAT I Verbal	515
Connecticut	2004	Mean SAT I Math	515
Connecticut	2003	Participation Rate 2003	84
Connecticut	2003	Mean SAT I Verbal	512

[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

(a) `Reported crime in Alabama`

(b)

before:	{'in', ' '}	'Alabama' → {'Alabama', *word*}
selection:	{'Alabama'}	'in' → {'in', *word*, *lowercase*}
after:	∅	' ' → {' '}

(c)

before:	{(' '), ('in', ' '), (*word*, ' '), (*lowercase*, ' ')}
selection:	{('Alabama'), (*word*)}
after:	∅

(d)

{(),('Alabama'),()}	~~{(),(*word*),()}~~
~~{(' '),(),()}~~	~~{(*word*, ' '),(),()}~~
{(' '),('Alabama'),()}	{(*word*, ' '),('Alabama'),()}
~~{(' '),(*word*),()}~~	~~{(*word*, ' '),(*word*),()}~~
{('in', ' '),(),()}	{(*lowercase*, ' '),(),()}
{('in', ' '),('Alabama'),()}	{(*lowercase*, ' '),('Alabama'),()}
{('in', ' '),(*word*),()}	~~{(*lowercase*, ' '),(*word*),()}~~

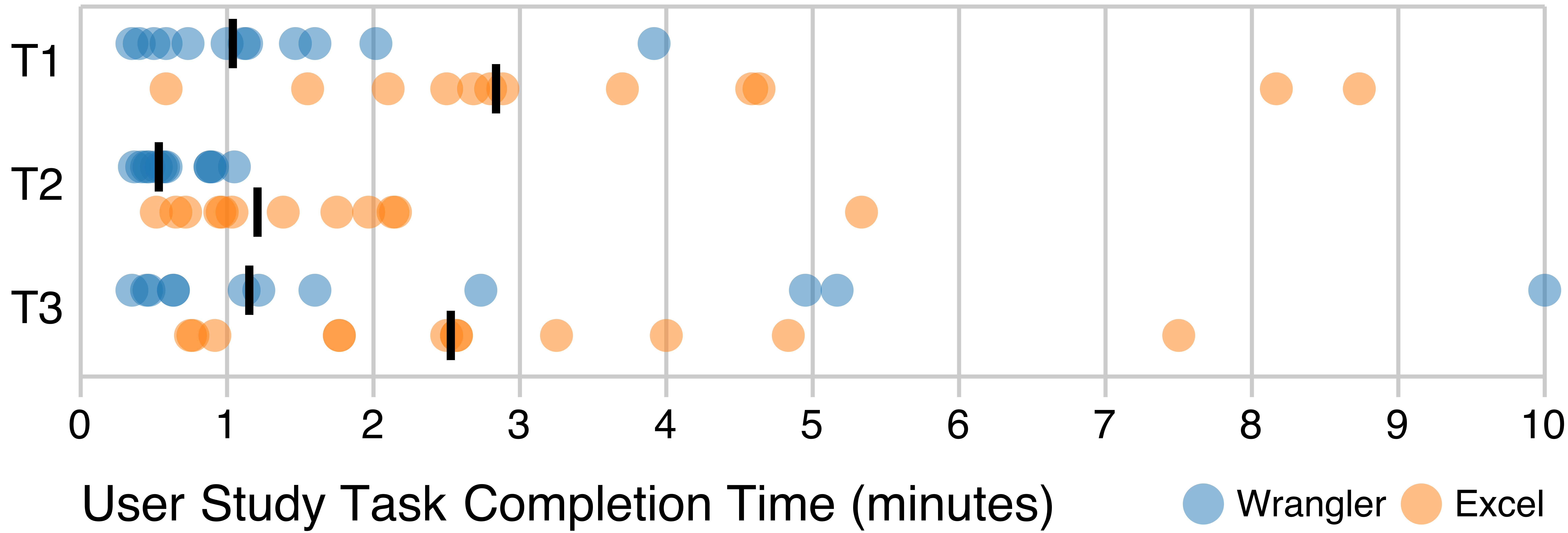
(e) {(*lowercase*, ' '),('Alabama'),()} → /[a-z]+ (Alabama)/

[S. Kandel et al., 2011]

Evaluation

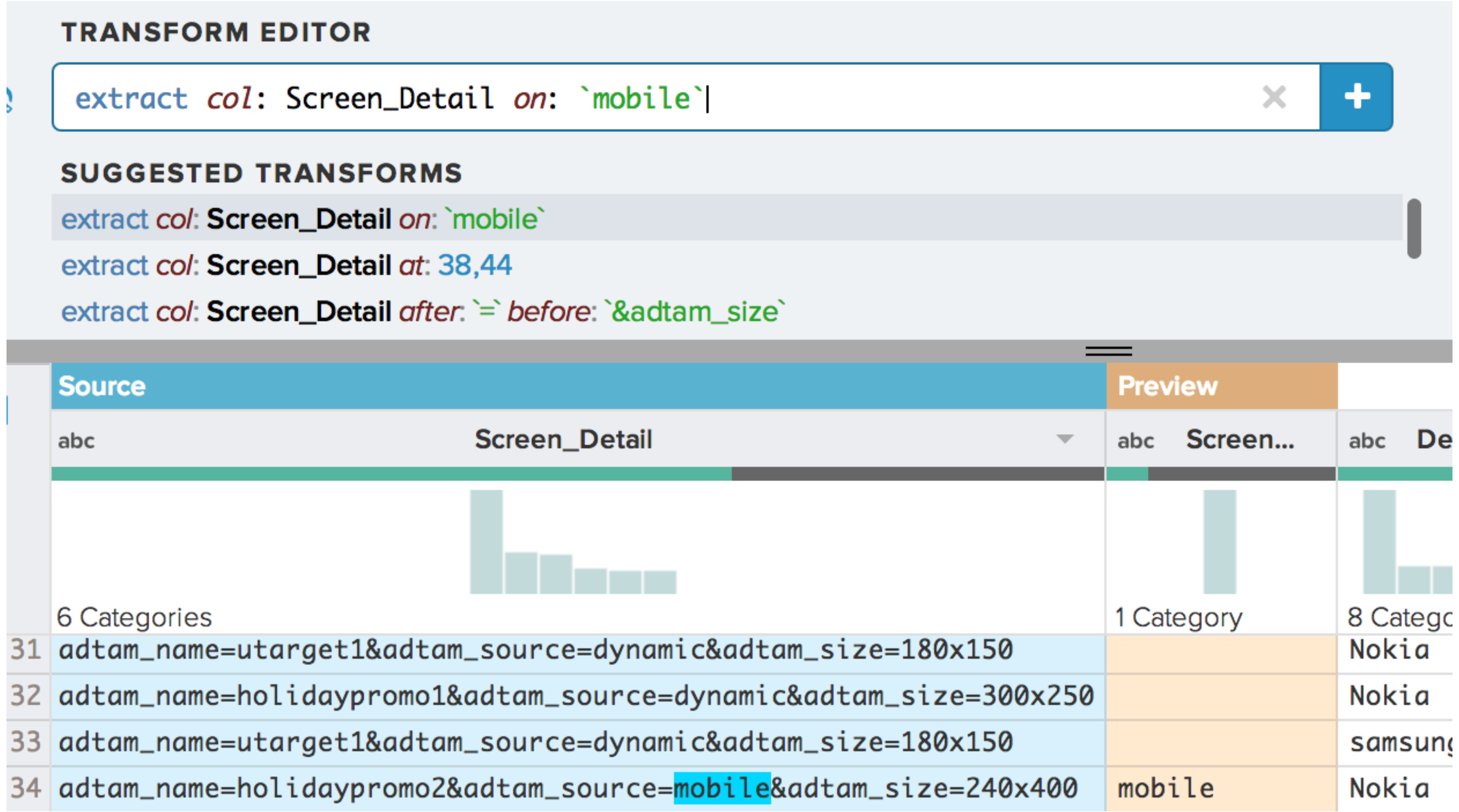
- Compare with Excel
- Tests:
  - Extract text from a single string entry
  - Fill in missing values with estimates
  - Reshape tables
- Allowed users to ask questions about Excel, not Wrangler
- Found significant effect of tool and users found previews and suggestions helpful
- Complaint: No manual fallback, make implications of user choices more obvious for users

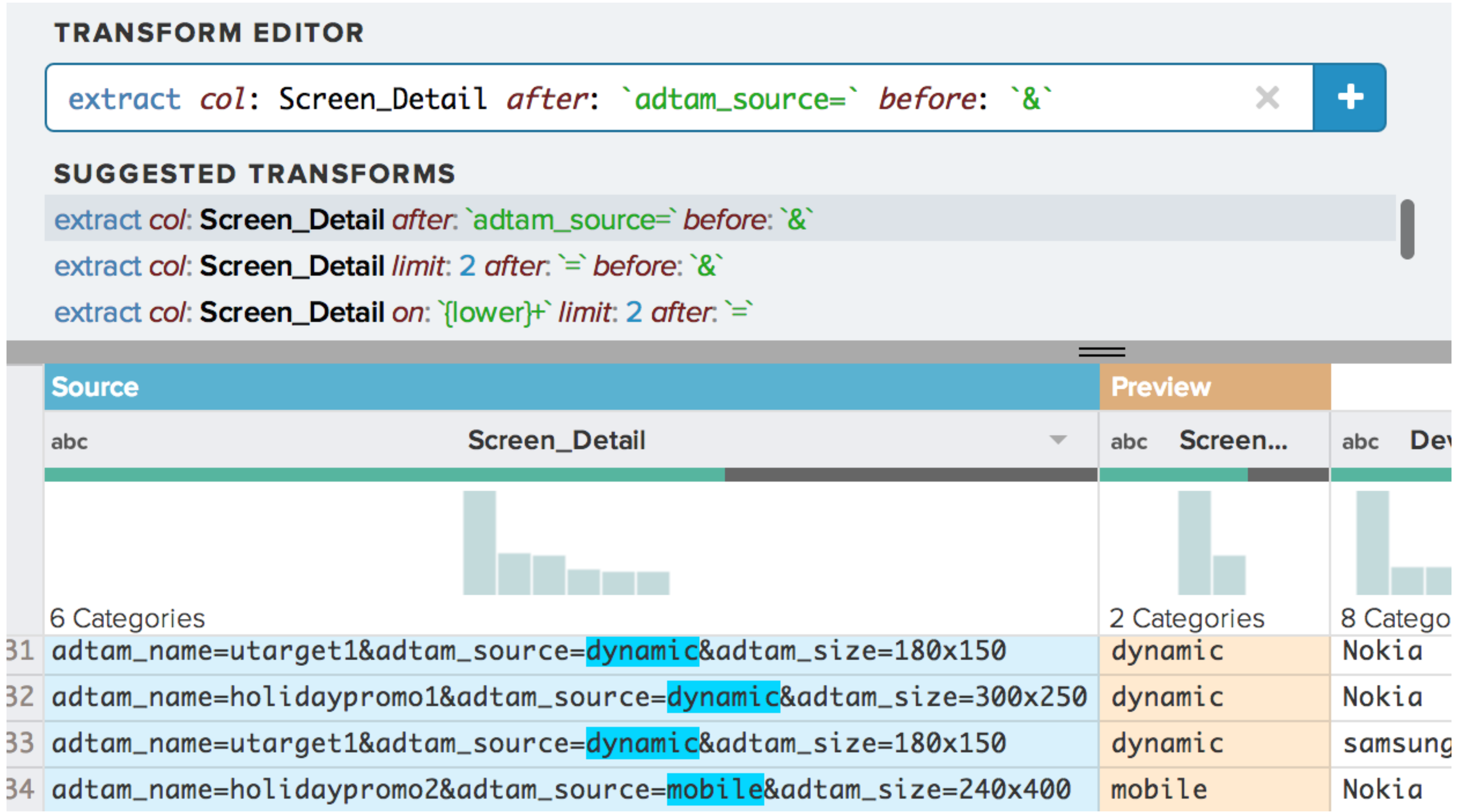
Task Completion Times



[S. Kandel et al., 2011]

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

Paper Critique

- Foofah: Transforming Data By Example, Z. Jin et al., 2017
- Due Wednesday **before** class, submit via Blackboard
- Read the paper
- Look up references if necessary
- Keep track of things you are confused by or that seem problematic
- Write a few sentences summarizing the paper's contribution
- Write more sentences discussing the paper and what you think the paper does well or doesn't do well at
- For this response, compare/contrast with Wrangler/Trifacta
- Length: 1/2-1 page

Data Cleaning in pandas

Handling Missing Data

- Filtering out missing data:
  - Can choose rows or columns
- Filling in missing data:
  - with a default value
  - with an interpolated value
- In pandas:

Argument	Description
`dropna`	Filter axis labels based on whether values for each label have missing data, with varying thresholds for how much missing data to tolerate.
`fillna`	Fill in missing data with some value or using an interpolation method such as `'ffill'` or `'bfill'`.
`isnull`	Return boolean values indicating which values are missing/NA.
`notnull`	Negation of `isnull`.

[W. McKinney, Python for Data Analysis]

Filling in missing data

- fillna arguments:

Argument	Description
`value`	Scalar value or dict-like object to use to fill missing values
`method`	Interpolation; by default `'ffill'` if function called with no other arguments
`axis`	Axis to fill on; default `axis=0`
`inplace`	Modify the calling object without producing a copy
`limit`	For forward and backward filling, maximum number of consecutive periods to fill

[W. McKinney, Python for Data Analysis]

Filtering and Cleaning Data

- Find duplicates
  - `duplicated`: returns boolean Series indicating whether row is a duplicate—first instance is **not marked** as a duplicate
- Remove duplicates:
  - `drop_duplicates`: drops all rows where `duplicated` is `True`
  - `keep`: which value to keep (first or last)
- Can pass specific columns to check for duplicates, e.g. check only key column

Changing Data

- Convert strings to upper/lower case
- Convert Fahrenheit temperatures to Celsius
- Create a new column based on another column

```
In [56]: lowercased
Out[56]:
0          bacon
1    pulled pork
2          bacon
3       pastrami
4    corned beef
5          bacon
6       pastrami
7      honey ham
8       nova lox
Name: food, dtype: object
```

```
meat_to_animal = {
  'bacon': 'pig',
  'pulled pork': 'pig',
  'pastrami': 'cow',
  'corned beef': 'cow',
  'honey ham': 'pig',
  'nova lox': 'salmon'
}
```

```
In [57]: data['animal'] = lowercased.map(meat_to_animal)

In [58]: data
Out[58]:
          food  ounces  animal
0        bacon     4.0     pig
1  pulled pork     3.0     pig
2        bacon    12.0     pig
3     Pastrami     6.0     cow
4  corned beef     7.5     cow
5        Bacon     8.0     pig
6     pastrami     3.0     cow
7    honey ham     5.0     pig
8     nova lox     6.0  salmon
```

[W. McKinney, Python for Data Analysis]

Replacing Values

- `fillna` is a special case
- What if `-999` in our dataset was identified as a missing value?

```
In [61]: data
Out[61]:
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

```
In [62]: data.replace(-999, np.nan)
Out[62]:
0       1.0
1       NaN
2       2.0
3       NaN
4   -1000.0
5       3.0
dtype: float64
```

- Can pass list of values or dictionary to change different values

Clamping Values

- Values above or below a specified thresholds are set to a max/min value

```
In [93]: data.describe()
Out[93]:
                 0            1            2            3
count  1000.000000  1000.000000  1000.000000  1000.000000
mean      0.049091     0.026112    -0.002544    -0.051827
std       0.996947     1.007458     0.995232     0.998311
min      -3.645860    -3.184377    -3.745356    -3.428254
25%      -0.599807    -0.612162    -0.687373    -0.747478
50%       0.047101    -0.013609    -0.022158    -0.088274
75%       0.756646     0.695298     0.699046     0.623331
max       2.653656     3.525865     2.735527     3.366626
```

```
In [97]: data[np.abs(data) > 3] = np.sign(data) * 3

In [98]: data.describe()
Out[98]:
                 0            1            2            3
count  1000.000000  1000.000000  1000.000000  1000.000000
mean      0.050286     0.025567    -0.001399    -0.051765
std       0.992920     1.004214     0.991414     0.995761
min      -3.000000    -3.000000    -3.000000    -3.000000
25%      -0.599807    -0.612162    -0.687373    -0.747478
50%       0.047101    -0.013609    -0.022158    -0.088274
75%       0.756646     0.695298     0.699046     0.623331
max       2.653656     3.000000     2.735527     3.000000
```

Computing Indicator Values

- Useful for machine learning
- Want to take possible values and map them to 0-1 indicators
- Example:

```
In [109]: df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
   .....:                    'data1': range(6)})

In [110]: pd.get_dummies(df['key'])
Out[110]:
   a  b  c
0  0  1  0
1  0  1  0
2  1  0  0
3  0  0  1
4  1  0  0
5  0  1  0
```

- Example: Genres in movies

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
  - `s.replace(',', ':') # "12:13: 14"`
- `upper()/lower()`: casing
  - `"AbCd".upper () # "ABCD"`
  - `"AbCd".lower() # "abcd"`

String Transformations

- `index(<str>)`: find where a substring first occurs (Error if not found)

```
- s = "12,13, 14"
  s.index(',') # 2
  s.index(':') # ValueError raised
```

- `find(<str>)`: same as index but `-1` if not found

```
- s.find(',') # 2
  s.find(':') # -1
```

- `startswith()`/`endswith()`: boolean checks for string occurrence

```
- s.startswith("1") # True
  s.endswith("5") # False
```

String Methods

Argument	Description
`count`	Return the number of non-overlapping occurrences of substring in the string.
`endswith`	Returns `True` if string ends with suffix.
`startswith`	Returns `True` if string starts with prefix.
`join`	Use string as delimiter for concatenating a sequence of other strings.
`index`	Return position of first character in substring if found in the string; raises `ValueError` if not found.
`find`	Return position of first character of *first* occurrence of substring in the string; like `index`, but returns –1 if not found.
`rfind`	Return position of first character of *last* occurrence of substring in the string; returns –1 if not found.
`replace`	Replace occurrences of string with another string.
`strip`, `rstrip`, `lstrip`	Trim whitespace, including newlines; equivalent to `x.strip()` (and `rstrip, lstrip`, respectively) for each element.
`split`	Break string into list of substrings using passed delimiter.
`lower`	Convert alphabet characters to lowercase.
`upper`	Convert alphabet characters to uppercase.
`casefold`	Convert characters to lowercase, and convert any region-specific variable character combinations to a common comparable form.
`ljust`, `rjust`	Left justify or right justify, respectively; pad opposite side of string with spaces (or some other fill character) to return a string with a minimum width.

[W. McKinney, Python for Data Analysis]

Regular Expressions

- AKA regex
- A syntax to better specify how to decompose strings
- Look for patterns rather than specific characters
- `"31" in "The last day of December is 12/31/2020."`
- May work for some questions but now suppose I have other lines like:
  `"The last day of September is 9/30/2020."`
- …and I want to find dates that look like:
- `<numbers>/<numbers>/<numbers>`
- Cannot search for every combination!
- `\d+/\d+/\d+`

Regular Expressions

- Character classes:
  - `\d` = digits
  - `\s` = spaces
  - `\w` = word character `[a-zA-Z0-9_]`
  - `[a-z]` = lowercase letters (square brackets indicate a set of chars)
- Repeating characters or patterns
  - `+` = one or more (any number)
  - `*` = zero or more (any number)
  - `?` = zero or one
  - `{<number>}` = a specific number (or range) of occurrences

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**!
  - ```
    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:]
    ```
    ```

Paper Critique

- Foofah: Transforming Data By Example, Z. Jin et al., 2017
- Due Wednesday **before** class, submit via Blackboard
- Read the paper
- Look up references if necessary
- Keep track of things you are confused by or that seem problematic
- Write a few sentences summarizing the paper's contribution
- Write more sentences discussing the paper and what you think the paper does well or doesn't do well at
- For this response, compare/contrast with Wrangler/Trifacta
- Length: 1/2-1 page