Advanced Data Management (CSCI 640/490)

Data Wrangling

Dr. David Koop

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

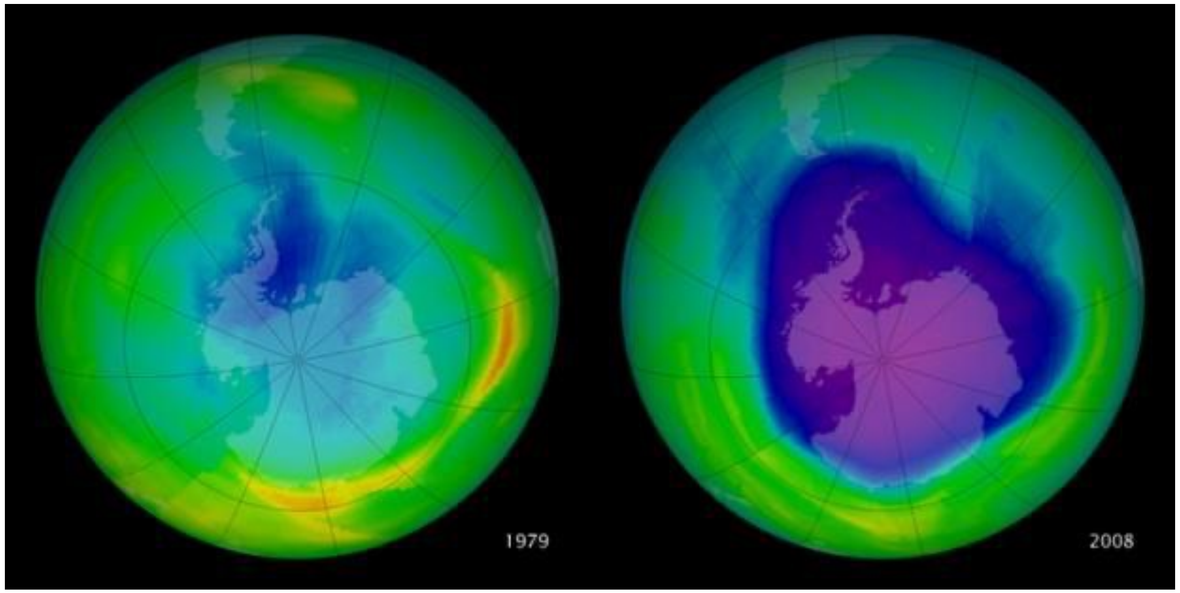
- Combination of:
  - Statistician's View: data has non-ideal samples for model
  - Database Expert's View: missing data, corrupted data
  - Domain Expert's View: data doesn't pass the smell test
- All of the views present problems with the data
- The goal may dictate the solutions:
  - Median value: 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

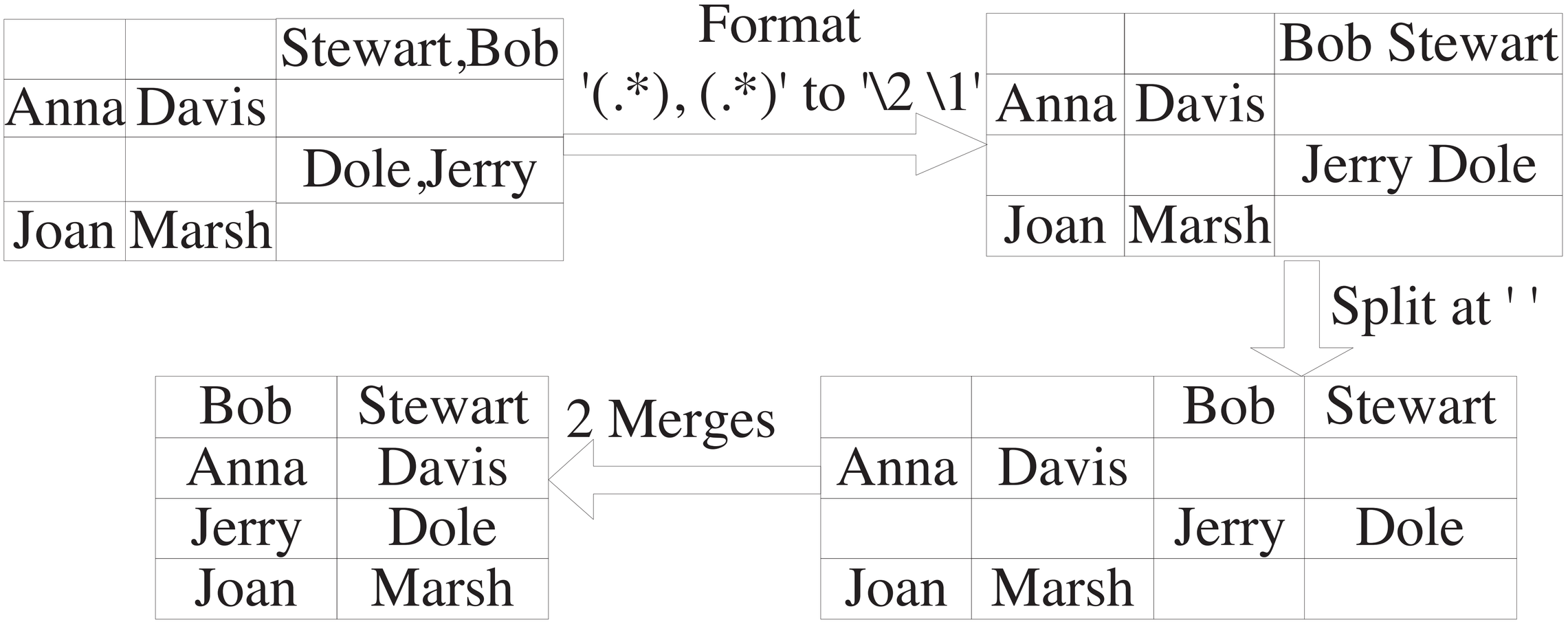


[Wikimedia]

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"

Potter's Wheel: Example



[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, $\text{left}(x, \text{splitter})$ is the left part of x after splitting by splitter. pred is a function returning a boolean.

[V. Raman and J. Hellerstein, 2001]

Interface

- Automated Transformation Suggestions
- Editable Natural Language Explanations

- 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]

Data Wrangler Demo

- http://vis.stanford.edu/wrangler/app/

Transform Script Import Export

- Split **data repeatedly** on **newline** into **rows**
- Split **split repeatedly** on **','**
- Promote **row 0** to header

Text | Columns | Rows | Table | Clear

Delete **row 7**

Delete **empty rows**

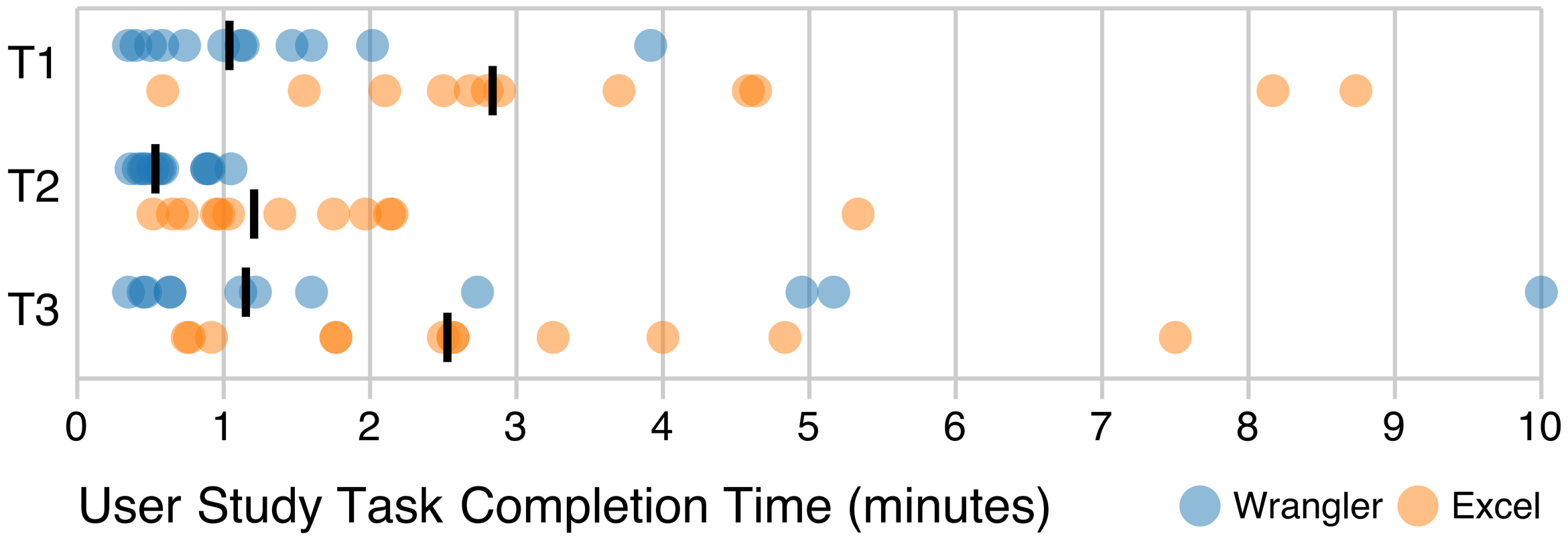
Fill **row 7** by **copying** values from **above**

	Year	Property_crime_rate
0	Reported crime in Alabama	
1		
2	2004	4029.3
3	2005	3900
4	2006	3937
5	2007	3974.9
6	2008	4081.9
7		
8	Reported crime in Alaska	
9		
10	2004	3370.9
11	2005	3615
12	2006	3582

Evaluation

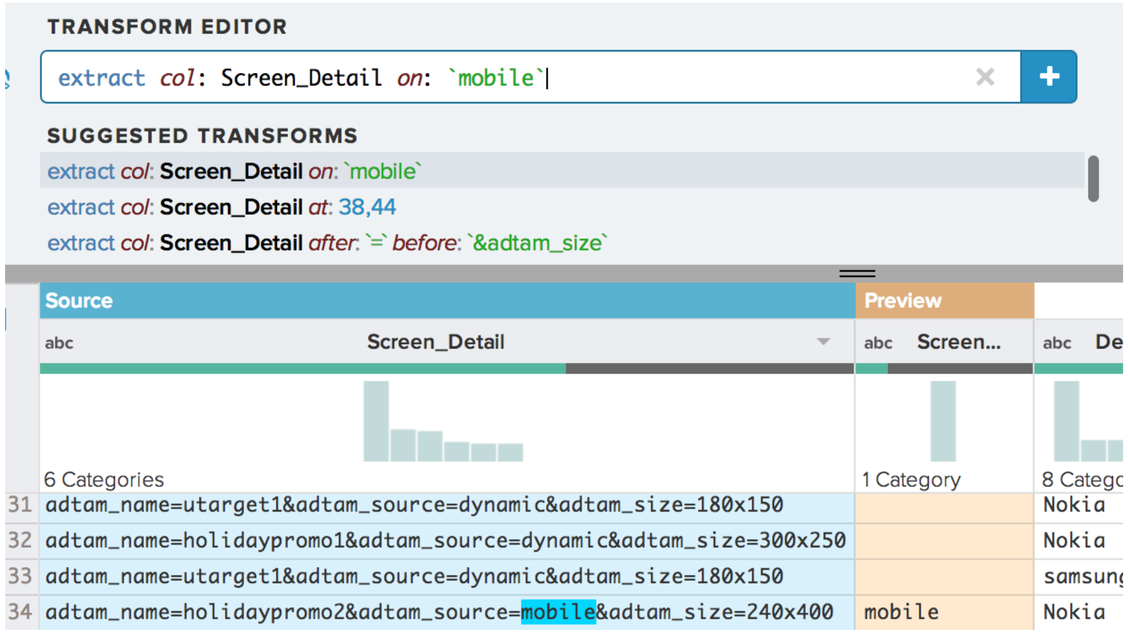
- Compare with Excel
- Tests:
  - Extract text from a single string entry
  - Fill in missing values with estimates
  - Reshape tables
- Allowed users to ask questions about Excel, not Wrangler
- Found significant effect of tool and users found previews and suggestions helpful
- Complaint: No manual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

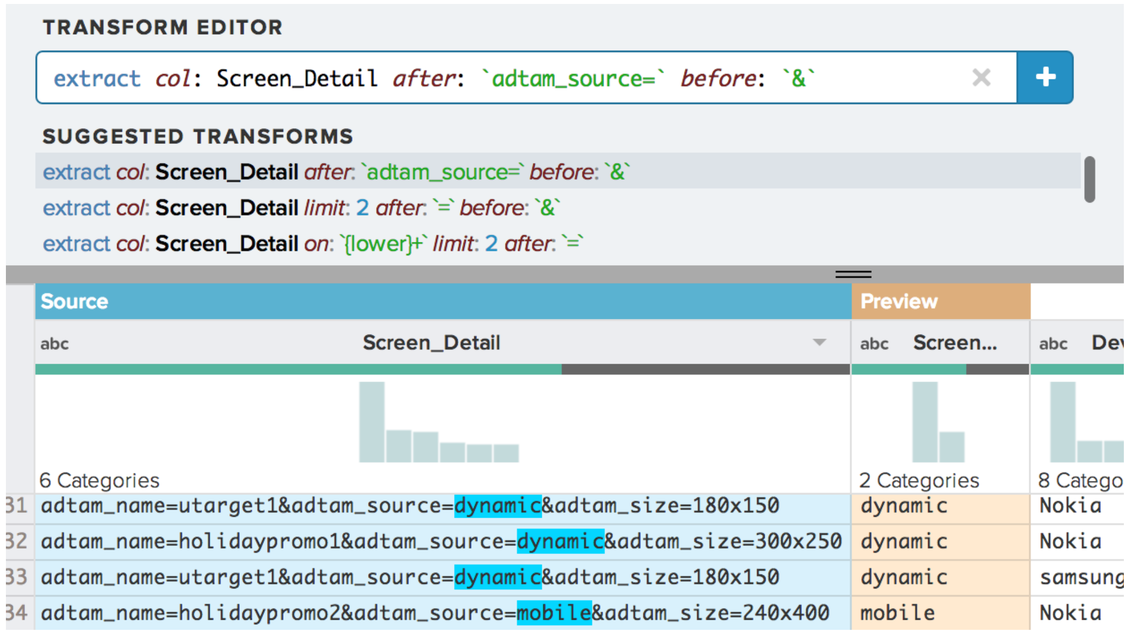
Task Completion Times



[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

Test 1

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

Remote Office Hours Today

- Due to family illness, need to conduct office hours remotely today (Zoom)
- Please email me with questions or for appointments

Reading Wednesday

- Read the paper
- Write a critique (like I did for Trifacta)
- Think about differences from transformations to reformating

Data Formats

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

Delimiter-separated Values

- Comma is a **delimiter**, specifies boundary between fields
- Could be a tab, pipe (|), or perhaps spaces instead
- All of these follow similar styles to CSV

Fixed-width Format

- Old school
- Each field gets a certain number of spots in the file
- Example:

```
- id8141    360.242940   149.910199   11950.7
  id1594    444.953632   166.985655   11788.4
  id1849    364.136849   183.628767   11806.2
  id1230    413.836124   184.375703   11916.8
  id1948    502.953953   173.237159   12468.3
```

- Specify exact character ranges for each field, e.g. 0-6 is the id

Reading & Writing Data

Reading Data in Python

- Use the `open()` method to open a file for reading
  - `f = open('huck-finn.txt')`
- Usually, add an `'r'` as the second parameter to indicate "read"
- Can iterate through the file (think of the file as a collection of lines):
  - ```
    f = open('huck-finn.txt', 'r')
    for line in f:
        if 'Huckleberry' in line:
            print(line.strip())
    ```
- Using `line.strip()` because the read includes the newline, and print writes a newline so we would have double-spaced text
- Closing the file: `f.close()`
    ```

With Statement: Improved File Handling

- With statement does "enter" and "exit" handling (similar to the finally clause):
- In the previous example, we need to remember to call `f.close()`
- Using a with statement, this is done automatically:

```
- with open('huck-finn.txt', 'r') as f:
      for line in f:
          if 'Huckleberry' in line:
              print(line.strip())
```

- This is more important for writing files!

```
- with open('output.txt', 'w') as f:
      for k, v in counts.items():
          f.write(k + ': ' + v + '\n')
```

- Without `with`, we need `f.close()`

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]

Types of arguments for readers

- Indexing: choose a column to index the data, get column names from file or user
- Type inference and data conversion: automatic or user-defined
- Datetime parsing: can combine information from multiple columns
- Iterating: deal with very large files
- Unclean Data: skip rows (e.g. comments) or deal with formatted numbers (e.g. 1,000,345)

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

More read_csv/read_tables arguments

Argument	Description
`skiprows`	Number of rows at beginning of file to ignore or list of row numbers (starting from 0) to skip.
`na_values`	Sequence of values to replace with NA.
`comment`	Character(s) to split comments off the end of lines.
`parse_dates`	Attempt to parse data to `datetime`; `False` by default. If `True`, will attempt to parse all columns. Otherwise can specify a list of column numbers or name to parse. If element of list is tuple or list, will combine multiple columns together and parse to date (e.g., if date/time split across two columns).
`keep_date_col`	If joining columns to parse date, keep the joined columns; `False` by default.
`converters`	Dict containing column number of name mapping to functions (e.g., `{'foo': f}` would apply the function `f` to all values in the `'foo'` column).
`dayfirst`	When parsing potentially ambiguous dates, treat as international format (e.g., 7/6/2012 -> June 7, 2012); `False` by default.
`date_parser`	Function to use to parse dates.
`nrows`	Number of rows to read from beginning of file.
`iterator`	Return a `TextParser` object for reading file piecemeal.
`chunksize`	For iteration, size of file chunks.

[W. McKinney, Python for Data Analysis]

Chunked Reads

- With very large files, we may not want to read the entire file
- Why?
  - Time
  - Want to understand part of data before processing all of it
- Reading only a few rows:
  - `df = pd.read_csv('example.csv', nrows=5)`
- Reading chunks:
  - Get an iterator that returns the next chunk of the file
  - `chunker = pd.read_csv('example.csv', chunksize=1000)`
  -
```
for piece in chunker:
    process_data(piece)
```

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

Reading/Writing CSV Data with DuckDB

- Importing:
  - `read_csv` method with parameters for delimter, header, etc.
  - `read_csv_auto` automatically **infer** these parameters
  - ```
    CREATE TABLE ontime AS SELECT * FROM
    read_csv_auto('flights.csv');
    ```
- Exporting:
  - Use the `COPY` function
  - `COPY tbl TO 'output.csv' (HEADER, DELIMITER ',');`
    ```

eXtensible Markup Language (XML)

- Older, self-describing format with nesting; each field has tags
- Example:

```
- <INDICATOR>
    <INDICATOR_SEQ>373889</INDICATOR_SEQ>
    <PARENT_SEQ></PARENT_SEQ>
    <AGENCY_NAME>Metro-North Railroad</AGENCY_NAME>
    <INDICATOR_NAME>Escalator Avail.</INDICATOR_NAME>
    <PERIOD_YEAR>2011</PERIOD_YEAR>
    <PERIOD_MONTH>12</PERIOD_MONTH>
    <CATEGORY>Service Indicators</CATEGORY>
    <FREQUENCY>M</FREQUENCY>
    <YTD_TARGET>97.00</YTD_TARGET>
  </INDICATOR>
```

- Top element is the **root**

XML

- No built-in method
- Use lxml library (also can use ElementTree)
- ```
  from lxml import objectify
  path = 'datasets/mta_perf/Performance_MNR.xml'
  parsed = objectify.parse(open(path))
  root = parsed.getroot()
  data = []
  skip_fields = ['PARENT_SEQ', 'INDICATOR_SEQ',
                 'DESIRED_CHANGE','DECIMAL_PLACES']
  for elt in root.INDICATOR:
      el_data = {}
      for child in elt.getchildren():
          if child.tag in skip_fields:
              continue
          el_data[child.tag] = child.pyval
      data.append(el_data)
  perf = pd.DataFrame(data)
  ```

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

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

What is the problem with reading this data?

```
• [{"name": "Wes",
    "places_lived": ["United States", "Spain", "Germany"],
    "pet": null,
    "siblings": [
       {"name": "Scott", "age": 25, "pet": "Zuko"},
       {"name": "Katie", "age": 33, "pet": "Cisco"}]
  },
  {"name": "Nia",
   "address": {"street": "143 Main",
               "city": "New York",
               "state": "New York"},
   "pet": "Fido",
   "siblings": [
       {"name": "Jacques", "age": 15, "pet": "Fido"}]
  },
 …
 ]
```

Reading JSON data

- Python has a built-in `json` module
  -
```
with open('example.json') as f:
    data = json.load(f)
```
  - Can also load/dump to strings:
    - `json.loads, json.dumps`
- Pandas has `read_json, to_json` methods

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`, need to specify object
- Excel: need to specify sheet when a spreadsheet has multiple sheets
  - `pd.ExcelFile` or `pd.read_excel`

Parquet

- "Open source, column-oriented data file format designed for efficient data storage and retrieval" [parquet.apache.org]
- Available in multiple languages including python
- Binary format
- Column-oriented: can read a column at a time (e.g. from the cloud)
- Self-describing (schema can be embedded)
- Supports compression
- Also supported via Apache Arrow (pyarrow in python) with zero-copy reads

Parquet/CSV Comparison

Dataset	Size on Amazon S3	Query Run time	Data Scanned	Cost
Data stored as CSV files	1 TB	236 seconds	1.15 TB	$5.75
Data stored in Apache Parquet format*	**130 GB**	**6.78 seconds**	**2.51 GB**	**$0.01**
Savings / Speedup	**87% less with Parquet**	**34x faster**	**99% less data scanned**	**99.7% savings**

Dataset	Columns	Size on Amazon S3	Data scanned	Cost (1TB = $5)
Data stored as CSV file	4	4TB	4TB	$20
Data stored as GZIP CSV file	4	1TB	1TB	$5
Data stored as Parquet file	4	1TB	0.25TB	$1.25

[T. Spicer]

Parquet Support

- Pandas:
  - Install pyarrow
  - `df = pd.read_parquet('input.parquet')`
  - `df.to_parquet('output.parquet')`
- DuckDB
  - `CREATE TABLE new_tbl AS SELECT * FROM read_parquet('input.parquet');`
  - `COPY tbl TO 'output.parquet' (FORMAT PARQUET);`

Transform Data by Example

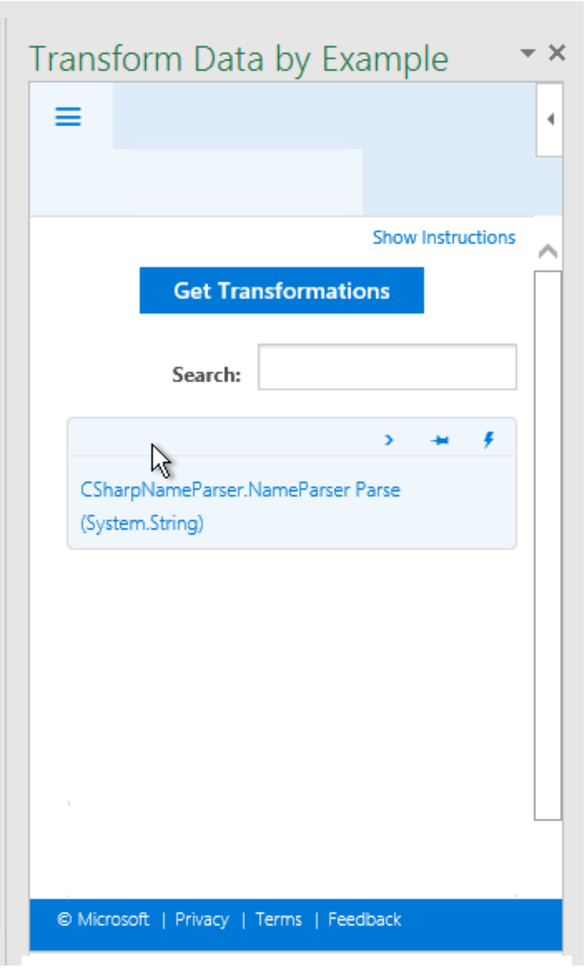
Wrangler

- Have to know what operations to apply
- What about an example-based approach instead?

Microsoft's Transform by Example

TDE: Transform Data by Example

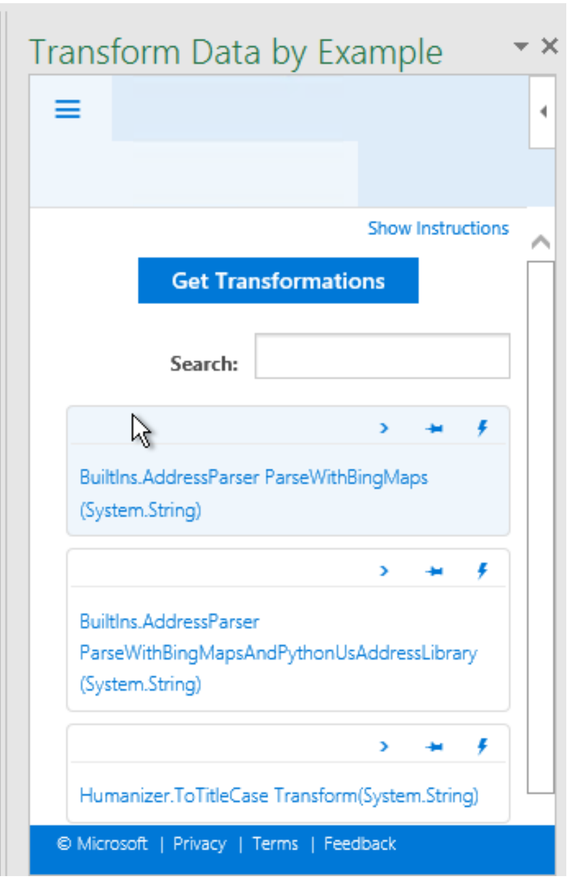
C	D
Customer Name	**Output**
John K. **Doe** Jr.	Doe, John
Mr. **Doe, John**	Doe, John
Jane A. **Smith**	Smith, Jane
MS. **Jane Smith**	Smith, Jane
Smith, Jane	Smith, Jane
Dr **Anthony** R **Von Fange** III	Von Fange, Anthony
Peter Tyson	Tyson, Peter
Dan E. **Williams**	Williams, Dan
James Davis Sr.	Davis, James
James J. **Davis**	Davis, James
Mr. **Donald** Edward **Miller**	Miller, Donald



[Y. He et al., 2018]

TDE: Transform Data by Example

C	D
Address	**Output**
4297 148th Avenue NE L105, Bellevue, WA 98007	Bellevue, WA, 98007
2720 N Mesa St, El Paso, 79902, USA	El Paso, TX, 79902
3524 W Shore Rd APT 1002, Warwick,02886	Warwick, RI, 02886
4740 N 132nd St, Omaha, 68164	Omaha, NE, 68164
10508 Prairie Ln, Oklahoma City	Oklahoma City, OK, 73162
525 1st St, Marysville, WA 95901	Marysville, CA, 95901
211 W Ridge Dr, Waukon,52172	Waukon, IA, 52172
1008 Whitlock Ave NW, Marietta, 30064	Marietta, GA, 30064
602 Highland Ave, Shinnston, 26431	Shinnston, WV, 26431
840 W Star St, Greenville, 27834	Greenville, NC, 27834

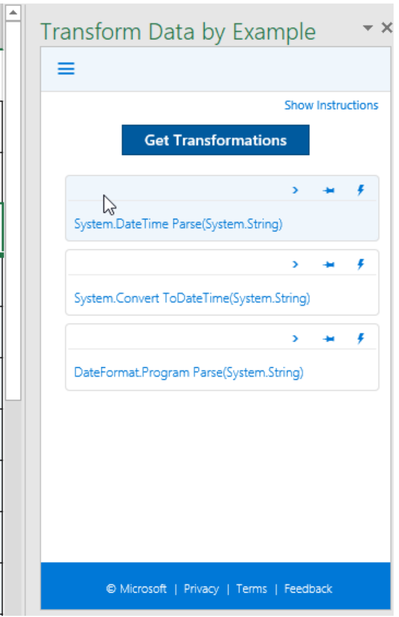


[Y. He et al., 2018]

TDE: Transform Data by Example

C	D
Transaction Date	**output**
Wed, 12 Jan 2011	2011-01-12-Wednesday
Thu, 15 Sep 2011	2011-09-15-Thursday
Mon, 17 Sep 2012	
2010-Nov-30 11:10:41	
2011-Jan-11 02:27:21	
2011-Jan-12	
2010-Dec-24	
9/22/2011	
7/11/2012	
2/12/2012	

C	D
Transaction Date	**output**
Wed, 12 Jan 2011	2011-01-12-Wednesday
Thu, 15 Sep 2011	2011-09-15-Thursday
Mon, 17 Sep 2012	2012-09-17-Monday
2010-Nov-30 11:10:41	2010-11-30-Tuesday
2011-Jan-11 02:27:21	2011-01-11-Tuesday
2011-Jan-12	2011-01-12-Wednesday
2010-Dec-24	2010-12-24-Friday
9/22/2011	2011-09-22-Thursday
7/11/2012	2012-07-11-Wednesday
2/12/2012	2012-02-12-Sunday



[Y. He et al., 2018]

TDE: Synthesized Function

Input Examples
Wed, 12 Jan 2011
Thu, 15 Sep 2011
Mon, 17 Sep 2012
2010-Nov-30 21:10:41
2011-Jan-11 02:27:21
2011-Jan-12

f

Return Object Dump					
Year	Month	Day	Day-of-week	Day-of-Year	...
2011	01	12	Wednesday	12	...
2011	09	15	Thursday	258	...
2012	09	17	Monday	261	...
2010	11	30	Tuesday	334	...
2011	01	11	Tuesday	11	...
2011	01	12	Wednesday	12	...

Member method result dump				
ToLongDateString()	ToTimeStr()	ToUTC()	ToBinary()	...
Wednesday, January 12, 2011	12:00:00 AM	...	...	...
Thursday, September 15, 2011	12:00:00 AM	...	...	...
Monday, September 17, 2012	12:00:00 AM	...	...	...
Tuesday, November 30, 2010	09:10:41 PM	...	...	...
Tuesday, January 11, 2011	02:27:21 AM	...	...	...
Wednesday, January 12, 2012	12:00:00 AM	...	...	...

Synthesize

Desired Output
2011-01-12 (Wed)
2011-09-15 (Thu)

[Y. He et al., 2018]

TDE: Transform Data by Example

- Row-to-row translation only
- Search System, GitHub, and StackOverflow for functions
- Given dataset with examples
  - Use L1 from library
  - Compose synthesized programs (L2)
  - Rank best transformations

TDE Benchmarks

System	Total cases (239)	FF-GR-Trifacta (46)	Head cases (44)	StackOverflow (49)	BingQL-Unit (50)	BingQL-Other (50)
TDE	**72% (173)**	**91% (42)**	**82% (36)**	**63% (31)**	**96% (48)**	**32% (16)**
TDE-NF	53% (128)	87% (40)	41% (18)	35% (17)	96% (48)	10% (5)
FlashFill	23% (56)	57% (26)	34% (15)	31% (15)	0% (0)	0% (0)
Foofah	3% (7)	9% (4)	2% (1)	4% (2)	0% (0)	0% (0)
DataXFormer-UB	38% (90)	7% (3)	36% (16)	35% (17)	62% (31)	**46% (23)**
System-A	13% (30)	52% (24)	2% (1)	10% (5)	0% (0)	0% (0)
OpenRefine-Menu[8]	4% (9)	13% (6)	2% (1)	4% (2)	0% (0)	0% (0)

- TDE and FlashFill focused on row-to-row transformations
- Foofah considers a wider range of transformations (table reformatting)

[Y. He et al., 2018]

Trifacta's Transform by Example

Transform by Pattern (TBP)

TBP Use Cases

- Auto-Unify

S-timestamp	S-phone	S-coordinates
2019-12-23	(425) 882-8080	(38°57'N, 95°15'W)
2019-12-24	(425) 882-8080	(38°61'N, 95°21'W)
2019-12-23	(206) 876-1800	(39°19'N, 95°18'W)
2019-12-24	(206) 876-1800	(39°26'N, 95°23'W)
2019-12-23	(206) 903-8010	(39°42'N, 96°38'W)

R-timestamp	R-phone	R-coordinates
Nov. 16 2019	650-853-1300	N37°31' W122°14'
Nov. 17 2019	650-853-1300	N37°18' W122°19'
Nov. 16 2019	425-421-1225	N37°48' W122°17'
Nov. 17 2019	425-421-1225	N37°60' W123°08'
Nov. 16 2019	650-253-0827	N37°01' W123°72'

- Auto-Repair

Date	Opponents
January 12, 1997	Venezuela
February 12, 1997	Peru
April 2, 1997	Colombia
1997-06-04	United States
1997-06-11	Chile
1997-06-14	Ecuador

(a) EN-Wiki: Dates

Year	Artist	Issue Price (BU)
1989	John Mardon	$16.25
1990	D.J. Craig	$16.75
1991	D.J. Craig	$16.75
1992	Karsten Smith	17.50
1993	Stewart Sherwood	$17.50
1994	Ian D. Sparkes	$17.95

(b) EN-Wiki: Currency values

Women's winner	Time
Anikó Kálovics	2:31:24
Lenah Cheruiyot	2:27:02
Lenah Cheruiyot	2:33.44
Emily Kimuria	2:28.42
Jane Ekimat	2:32.08

(c) EN-wiki:time

#	Original air date[1]
12	March 23, 2008
13	March 30, 2008
14	April 6, 2008
15	13 April 2008
16	20 April 2008

(d) EN-Wiki: Date

TBP Learning from Tables

T_1

Name	#	Born	Died
Washington, George	USA President (1)	02/22/1732	12/14/1799
Adams, John	USA President (2), VP (1)	10/30/1735	07/04/1826
Jefferson, Thomas	USA President (3), VP (2)	04/13/1743	07/04/1826
Madison, James	USA President (4)	03/16/1751	06/28/1836
Monroe, James	USA President (5)	04/28/1758	07/04/1851

T_2

Date of birth	President	Birthplace	State† of birth
February 22, 1732	George Washington	Westmoreland County	Virginia†
October 30, 1735	John Adams	Braintree	Massachusetts†

T_3

30.	**George Washington**	–	57y, 10d	22.02.1732	14.12.1799
31.	**John Quincy Adams**	Nat-Rep	57y, 7m, 20d	11.07.1767	23.02.1848
32.	**Thomas Jefferson**	Dem-Rep	57y, 10m, 18d	13.04.1743	04.07.1826
33.	**James Madison**	Dem-Rep	57y, 11m, 15d	16.03.1751	28.06.1836
34.	**James Monroe**	Dem-Rep	58y, 10m, 3d	28.04.1758	04.07.1831

T_4

1.	George Washington	Virginia	Feb. 22, 1732	Dec. 14, 1797
3.	Thomas Jefferson	Virginia	Apr. 13, 1743	July 4, 1826
4.	James Madison	Virginia	Mar. 16, 1751	June 28, 1836
6.	John Quincy Adams	Massachusetts	July 11, 1767	Feb. 23, 1848

T_5

	Name and (party)[1]	Term	State of birth	Born	Died	Religion[2]	Age at inaug.	Age at death
1.	Washington (F)[3]	1789–1797	Va.	2/22/1732	12/14/1799	Episcopalian	57	67
2.	J. Adams (F)	1797–1801	Mass.	10/30/1735	7/4/1826	Unitarian	61	90

T_6

PRESIDENT	BIRTH DATE	BIRTH PLACE	DEATH DATE	LOCATION OF DEATH
George Washington	Feb 22, 1732	Westmoreland Co., Va.	Dec 14, 1799	Mount Vernon, Va.
John Adams	Oct 30, 1735	Quincy, Mass.	July 4, 1826	Quincy, Mass.

TBP Programs and Triples

Table 1: An example repository of TBP programs (P_s, P_t, T), where each line is a TBP program. The first three programs can be used to auto-unify the two tables shown in Figure 2.

TBP-id	Source-pattern (P_s)	Target-pattern (P_t)	(T)
TBP-1	<letter>{3}. <digit>{2}, <digit>{4}	<digit>{4}-<digit>{2}-<digit>{2}	...
TBP-2	(<digit>{3}) <digit>{3}-<digit>{4}	<letter>{3}-<digit>{3}-<digit>{4}	...
TBP-3	(<digit>+°<num>'<letter>{1}, <digit>+°<num>'<letter>{1})	<letter>{1}<digit>+°<num>' <letter>{1}<digit>+°<num>'	...
...	...	...	...
TBP-7	<digit>{4}/<digit>{2}/<digit>{2}	<letter>{3} <digit>{2}	...
TBP-8	<num> kg	<num> lb	...
TBP-9	<num> lb	<num> lb <num> oz	...
...	...	...	...
TBP-15	<num> kg	<num>公斤	...
TBP-16	<letter>+ de <digit>{4}	<digit>{4}	...
...	...	...	...

CCT-id	Input-column (C)	Output-column (C')	Program (T)
CCT-1	(C_1) "Born" = {"02/22/1732", "10/30/1735", ... }	(C'_1) "Date of birth" = {"February 22, 1732", ... }	Listing 1
CCT-2	(C_2) "Date of birth" = {"February 22, 1732", ... }	(C'_2) "Born" = {"02/22/1732", "10/30/1735", ... }	...
CCT-3	(C_3) "Died" = {"02/14/1799", "07/04/1826", ... }	(C'_3) "Date of birth" = {"February 22, 1732", ... }	...
CCT-4	(C_4) "Date" = {"11/01/2019", "12/01/2019", ... }	(C'_4) "Date-2" = {"November 01, 2019", ... }	Listing 1
...	...	...	...
CCT-9	(C_9) "Name" = {"Washington, George", "Adam, John", ... }	(C'_9) "Date of birth" = {"February 22, 1732", ... }	∅
...	...	...	...

Comments/Critique?