A data warehouse integrates (inconsistent) data coming from different sources in a consistent way.
DATA INTEGRATION

- **Extract-Transform-Load**
  - “Old” term
  - Schema-centric
  - Big Band Integration / All at once

- **Data Wrangling**
  - “Hipster” term
  - Less structured
  - Ad-hoc / Incremental
DATA INTEGRATION

Data Cleaning → Schema Matching → Entity Resolution → Data Fusion
What is wrong here?

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>P-Code</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J Smith</td>
<td>123 University Ave</td>
<td>Seattle</td>
<td>Washington</td>
<td>98106</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>Mary Jones</td>
<td>245 3rd St</td>
<td>Redmond</td>
<td>WA</td>
<td>98052-1234</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Bob Wilson</td>
<td>345 Broadway</td>
<td>Seattle</td>
<td>Washington</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>M Jones</td>
<td>245 Third Street</td>
<td>Redmond</td>
<td>NULL</td>
<td>98052</td>
<td>299</td>
</tr>
<tr>
<td>5</td>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>James Smith</td>
<td>123 Univ Ave</td>
<td>Seattle</td>
<td>WA</td>
<td>NULL</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>J Widom</td>
<td>123 University Ave</td>
<td>Palo Alto</td>
<td>CA</td>
<td>94305</td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
### Real World Data

#### Customer Dataset

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>P-Code</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J Smith</td>
<td>123 University Ave</td>
<td>Seattle</td>
<td>Washington</td>
<td>98106</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>Mary Jones</td>
<td>245 3rd St</td>
<td>Redmond</td>
<td>WA</td>
<td>98052-1234</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Bob Wilson</td>
<td>345 Broadway</td>
<td>Seattle</td>
<td>Washington</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>M Jones</td>
<td>245 Third Street</td>
<td>Redmond</td>
<td>NULL</td>
<td>98052</td>
<td>299</td>
</tr>
<tr>
<td>5</td>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>James Smith</td>
<td>123 Univ Ave</td>
<td>Seatle</td>
<td>WA</td>
<td>NULL</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>J Widom</td>
<td>123 University Ave</td>
<td>Palo Alto</td>
<td>CA</td>
<td>94305</td>
<td>NULL</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
- How many customers do I have?

Wrong answer because of duplicate records!

- How many customers by state?

```
select count(*)
from customer
```

<table>
<thead>
<tr>
<th>State</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>60</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WA</td>
<td>1200</td>
</tr>
<tr>
<td>Washington</td>
<td>50</td>
</tr>
<tr>
<td>Washington</td>
<td>2</td>
</tr>
</tbody>
</table>

What about if you give this data to a ML algorithm?
THE DATA QUALITY PROBLEM

Data is dirty on its own

Data sets are clean but integration (i.e., combining them) screws them up (e.g., duplicates are created)

Old data rots, i.e., it loses its value over time (storing amounts without currency conversion of that time)

Any combination of the above
DIRTY DATA PROBLEMS

1) Parsing input data (e.g., separator issues)
2) Naming conventions: NYC vs New York
3) Formatting issues – esp. dates
4) Missing values and required fields (e.g., always use 0)
5) Different representations (2 vs Two)
6) Fields too long (get truncated)
7) Primary key violations (from data merging)
8) Redundant Records (from data merging)
TYPICAL PROBLEMS: DATA ENTRY

Why are so many of our customers in Schenectady, NY?
SOLUTION: DETECT OUTLIERS?

ages of employees (US)
DATA CLEANING MAKES EVERYTHING OKAY?

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

In fact, the data were rejected as unreasonable by data quality control algorithms.
DATA WAREHOUSING

Data Integration

Multiple Independent Schemata

One Integrated Schemata
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Address</th>
<th>Zip</th>
<th>State</th>
<th>City</th>
<th>Phone</th>
<th>E-Mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tim Kraska</td>
<td>135 Watermann St, 02906</td>
<td>RI</td>
<td>Providence</td>
<td>RI</td>
<td>+1 234234 234</td>
<td><a href="mailto:Tim_kraska@brown.edu">Tim_kraska@brown.edu</a></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Address</th>
<th>Phone</th>
<th>E-Mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tim Kraska</td>
<td>135 Watermann St, 02906</td>
<td>+1 234234 234</td>
<td><a href="mailto:Tim_kraska@brown.edu">Tim_kraska@brown.edu</a></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Address</th>
<th>Phone-Nb</th>
<th>E-Mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>135 Watermann St, 02906</td>
<td>+1 234234 234</td>
<td><a href="mailto:Tim_kraska@brown.edu">Tim_kraska@brown.edu</a></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>222 Hope St, 02906</td>
<td>980 – 0803284</td>
<td><a href="mailto:tim_kraska@brown.edu">tim_kraska@brown.edu</a></td>
</tr>
</tbody>
</table>
CLICKER QUESTION:

How Many Tables has a (typical) SAP’s ERP Installation?

(a) 100 - 1.000
(b) 1.000 - 10.000
(c) 10.000 - 100.000
(d) > 100.000
CLICKER QUESTION:

How Many Tables has a (typical) SAP’s ERP Installation?

(a) 100 - 1.000
(b) 1.000 - 10.000
(c) 10.000 - 100.000
(d) > 100.000

70.000 – 140.000
SCHEMA MATCHING: IDEAS?

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Address</th>
<th>Zip</th>
<th>State</th>
<th>City</th>
<th>Phone</th>
<th>E-Mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tim Kraska</td>
<td>135 Watermann St,</td>
<td>02906</td>
<td>Providence</td>
<td>RI</td>
<td>+1 234234 234</td>
<td><a href="mailto:Tim_kraska@brown.edu">Tim_kraska@brown.edu</a></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Addr</th>
<th>Mobile</th>
<th>E-Mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tim Kraska</td>
<td>135 Watermann St,</td>
<td>+1 234234 234</td>
<td><a href="mailto:Tim_kraska@brown.edu">Tim_kraska@brown.edu</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>02906 Providence,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address ID</th>
<th>Person-ID</th>
<th>Address</th>
<th>Phone-Nb</th>
<th>E-Mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>135 Watermann St,</td>
<td>+1 234234 234</td>
<td><a href="mailto:Tim_kraska@brown.edu">Tim_kraska@brown.edu</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>02906 Providence,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>222 Hope St,</td>
<td>980 – 0803284</td>
<td><a href="mailto:tim_kraska@brown.edu">tim_kraska@brown.edu</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>02906 Providence</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SCHEMA MATCHING - TECHNIQUES

- **Instance vs Schema**: consider instance data or schema information.

- **Element vs Structure**: matching performed for individual schema element (attribute), or for combinations of elements (structure).

- **Use domain information**: use linguistic information (dictionaries) or constraint information (key, relationship).

- **Using cardinality information**: the overall match result may relate one or more elements of one schema to one or more elements of the other (1:1, 1:n, m:n).

- **Other auxiliary information**: the use of auxiliary information (previous matching results, user input,..)
DATA INTEGRATION

- Data Cleaning
- Schema Matching
- Entity Resolution
- Data Fusion

deduplication, entity clustering, merge/purge, record linkage, approximate match...
<table>
<thead>
<tr>
<th>ID</th>
<th>Product Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>iPad Two 16GB WiFi White</td>
<td>$490</td>
</tr>
<tr>
<td>r2</td>
<td>iPad 2nd generation 16GB WiFi White</td>
<td>$469</td>
</tr>
<tr>
<td>r3</td>
<td>iPhone 4th generation White 16GB</td>
<td>$545</td>
</tr>
<tr>
<td>r4</td>
<td>Apple iPhone 4 16GB White</td>
<td>$520</td>
</tr>
<tr>
<td>r5</td>
<td>Apple iPhone 3rd generation Black 16GB</td>
<td>$375</td>
</tr>
<tr>
<td>r6</td>
<td>iPhone 4 32GB White</td>
<td>$599</td>
</tr>
<tr>
<td>r7</td>
<td>Apple iPad2 16GB WiFi White</td>
<td>$499</td>
</tr>
<tr>
<td>r8</td>
<td>Apple iPod shuffle 2GB Blue</td>
<td>$49</td>
</tr>
<tr>
<td>r9</td>
<td>Apple iPod shuffle USB Cable</td>
<td>$19</td>
</tr>
</tbody>
</table>
ENTITY RESOLUTION

“[The] problem of identifying and linking/grouping different manifestations of the same real world object.”

Challenges

• Diversity in representations (format, truncation, ambiguity)
• Data entry errors
• Missing data
• Records from different times
• …
<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>P-Code</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J Smith</td>
<td>123 University Ave</td>
<td>Seattle</td>
<td>Washington</td>
<td>98106</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>Mary Jones</td>
<td>245 3rd St</td>
<td>Redmond</td>
<td>WA</td>
<td>98052-1234</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Bob Wilson</td>
<td>345 Broadway</td>
<td>Seattle</td>
<td>Washington</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>M Jones</td>
<td>245 Third Street</td>
<td>Redmond</td>
<td>NULL</td>
<td>98052</td>
<td>299</td>
</tr>
<tr>
<td>5</td>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>James Smith</td>
<td>123 Univ Ave</td>
<td>Seattle</td>
<td>WA</td>
<td>NULL</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>J Widom</td>
<td>123 University Ave</td>
<td>Palo Alto</td>
<td>CA</td>
<td>94305</td>
<td>NULL</td>
</tr>
</tbody>
</table>
TEXTUAL SIMILARITY

String Similarity function:

- $Sim(string, string) \rightarrow numeric\ value$

A “good” similarity function:

- Strings representing the same concept $\Rightarrow$ high similarity
- Strings representing different concepts $\Rightarrow$ low similarity
EDIT DISTANCE

EditDistance(s1, s2):

- Minimum number of edits to transform s1 to s2

Edit:

- Insert a character
- Delete a character
- Substitute a character

Note: EditDistance(s1, s2) = EditDistance(s2, s1)
“Distance” = opposite of similarity
**EDIT DISTANCE**

\[ \text{EditDistance} \left( \text{“Provdince”}, \text{“Providence”} \right) = 2 \]

Provdince $\rightarrow$ Providince $\rightarrow$ Providence

\[ \text{EditDistance} \left( \text{“Seattle”}, \text{“Redmond”} \right) = 6 \]

Seattle $\rightarrow$ Reattle $\rightarrow$ Redttle
Redmtle $\rightarrow$ Redmole $\rightarrow$ Redmone
$\rightarrow$ Redmond
EDIT DISTANCE

115\textsuperscript{th} Waterman St., Providence, RI

\textbf{EditDistance} = 1

110\textsuperscript{th} Waterman St., Providence, RI

\textbf{EditDistance} = 4

Waterman Street, Providence, RI

\textbf{EditDistance} = 4

Waterman St, Providence, RI
EDIT DISTANCE

148th Ave NE, Redmond, WA

EditDist = 0

148th Ave NE, Redmond, WA

148th Ave NE, Redmond, WA

EditDist = 4

NE 148th Ave, Redmond, WA

Order sensitive Similarity?
JACCARD SIMILARITY

- Statistical measure
- Originally defined over sets
- String = set of words

\[
Jaccard(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|}
\]

- Range of values = [0,1]
Jaccard Similarity

\[ Jaccard = \frac{4}{4 + 2} \approx 0.66 \]

148th Ave NE, Redmond, WA

140th Ave NE, Redmond, WA
JACCARD SIMILARITY

\[ Jaccard = \frac{5}{5} = 1.0 \]

I 48th Ave NE, Redmond, WA

NE 148th Ave, Redmond, WA
CLICKER QUESTION I:

What is the Jaccard Similarity between:

- iPad Two 16GB WiFi White
- iPad 2nd generation 16GB Wifi White

(a) 3 / 8
(b) 4 / 11
(c) 4 / 7
CLICKER QUESTION I:

What is the Jaccard Similarity between:

• iPad Two 16GB WiFi White
• iPad 2nd generation 16GB Wifi White

(a) 3 / 8
(b) 4 / 11
(c) 4 / 7
CLICKER QUESTION II

Which jaccard similarity is wrong?

A) Microsoft Corporation \[\text{Jaccard} = \frac{1}{3}\]
   Microsoft Corp
B) Microsoft Corporation \[\text{Jaccard} = \frac{1}{3}\]
   Oracle Corporation
C) Waterman 115 St \[\text{Jaccard} = \frac{1}{4}\]
   115 Waterman Street
CLICKER QUESTION II

Which Jaccard similarity is wrong?

A) **Microsoft Corporation** ↔ **Jaccard = 1/3**
   **Microsoft Corp**

B) **Microsoft Corporation** ↔ **Jaccard = 1/3**
   **Oracle Corporation**

C) **Waterman 115 St** ↔ **Jaccard = 1/4**
   **115 Waterman Street**
WHAT CAN WE DO ABOUT?

Microsoft Corporation

Microsoft Corp

Microsoft Corporation

Oracle Corporation
JACCARD SIMILARITY

Weight Function = \( wt: \text{Elements} \rightarrow \mathbb{R}^+ \)

\[
WtJaccard(s1, s2) = \frac{wt(s1 \cap s2)}{wt(s1 \cup s2)}
\]

\[
wt(s) = \sum_{e \in s} wt(e)
\]

\[
wt(\text{"Microsoft"}) > wt(\text{"Corporation"})
\]

\[
Wt(\text{"Oracle"}) > wt(\text{"Corporation"})
\]
OTHER SIMILARITY FUNCTIONS

- Affine edit distance
- Cosine similarity
- Hamming distance
- Generalized edit distance
- Jaro distance
- Monge-Elkan distance
- Q-gram
- Smith-Warerman distance
- Soundex distance
- TF/IDF
- …many more

• No universally good similarity function
• Choice of similarity function depends on domains of interest, data instances, etc.
## RECORD MATCHING PROBLEMS

### Customer

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>P-Code</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J Smith</td>
<td>123 University Ave</td>
<td>Seattle</td>
<td>Washington</td>
<td>98106</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>Mary Jones</td>
<td>245 3rd St</td>
<td>Redmond</td>
<td>WA</td>
<td>98052-1234</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Bob Wilson</td>
<td>345 Broadway</td>
<td>Seattle</td>
<td>Washington</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>M Jones</td>
<td>245 Third Street</td>
<td>Redmond</td>
<td>NULL</td>
<td>98052</td>
<td>299</td>
</tr>
<tr>
<td>5</td>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>James Smith</td>
<td>123 Univ Ave</td>
<td>Seattle</td>
<td>WA</td>
<td>NULL</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>J Widom</td>
<td>123 University Ave</td>
<td>Palo Alto</td>
<td>CA</td>
<td>94305</td>
<td>NULL</td>
</tr>
</tbody>
</table>

... ... ... ... ... ...

\[
W_t Jaccard = 0.57 \quad 0.91 \quad 1.0 \quad 0.0 \quad 1.0 \quad 1.0 \quad ...
\]
COMBINING SIMILARITY FUNCTIONS

Vector of similarity scores

Record Pair → \[ \text{Jacc(Name)} \]
\[ \text{Jacc(Street)} \]
\[ \text{Edit(City)} \]
\[ \text{Edit(State)} \]
\[ \text{Edit(PostalCode)} \]
\[ \text{Equality(Age)} \]
→ Function (\( F_n \)) → Match/Non-Match

Features → Binary Classification

Idea: Weighted sum of per attribute similarity + threshold?
## LEARNING-BASED APPROACH

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob Wilson</td>
<td>345 Broadway</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>B Wilson</td>
<td>123 Broadway</td>
<td>Boise</td>
<td>Idaho</td>
<td>83712</td>
<td>19</td>
</tr>
<tr>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
<tr>
<td>Mary Jones</td>
<td>245 3rd St</td>
<td>Redmond</td>
<td>WA</td>
<td>98052-1234</td>
<td>30</td>
</tr>
<tr>
<td>M Jones</td>
<td>245 Third Street</td>
<td>Redmond</td>
<td>NULL</td>
<td>98052</td>
<td>299</td>
</tr>
<tr>
<td>Mary Jones</td>
<td>245 3rd St</td>
<td>Redmond</td>
<td>WA</td>
<td>98052-1234</td>
<td>30</td>
</tr>
<tr>
<td>Robert Wilson</td>
<td>345 Broadway St</td>
<td>Seattle</td>
<td>WA</td>
<td>98101</td>
<td>19</td>
</tr>
</tbody>
</table>
LEARNING BASED APPROACH
LEARNING BASED APPROACH

![Graph showing Jaccard Similarity between 'Name' and 'Street' with a diagonal line indicating points above the line having higher Jaccard Similarity for 'Name' than 'Street' and vice versa.]
DATA INTEGRATION

Data Cleaning  Schema Matching  Entity Resolution  Data Fusion
DATA FUSION’S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

- Resolves inconsistency across diversity of sources
DATA FUSION’S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

Voting

Source Quality

Copy Detection

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
</tr>
</tbody>
</table>
DATA FUSION’S THREE COMPONENTS

Data fusion: voting + source quality + copy detection
• Supports difference of opinion

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
</tr>
</tbody>
</table>
DATA FUSION’S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
</tr>
</tbody>
</table>
DATA FUSION’S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

• Gives more weight to knowledgeable sources

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
</tr>
</tbody>
</table>
DATA FUSION’S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

Voting

Source Quality

Copy Detection

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
<td>UM</td>
<td>UI</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
<td>UMD</td>
<td>UMD</td>
</tr>
</tbody>
</table>
Data fusion: voting + source quality + copy detection

- Reduces weight of copier sources

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jagadish</td>
<td>UM</td>
<td>ATT</td>
<td>UM</td>
<td>UM</td>
<td>UI</td>
</tr>
<tr>
<td>Dewitt</td>
<td>MSR</td>
<td>MSR</td>
<td>UW</td>
<td>UW</td>
<td>UW</td>
</tr>
<tr>
<td>Bernstein</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
<td>MSR</td>
</tr>
<tr>
<td>Carey</td>
<td>UCI</td>
<td>ATT</td>
<td>BEA</td>
<td>BEA</td>
<td>BEA</td>
</tr>
<tr>
<td>Franklin</td>
<td>UCB</td>
<td>UCB</td>
<td>UMD</td>
<td>UMD</td>
<td>UMD</td>
</tr>
</tbody>
</table>
DATA INTEGRATION SO FAR

But what are some practical tools?
### SO FAR: RELATIONAL DATA

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>s</td>
<td>e</td>
<td>f</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>g</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>g</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>v</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Appearances</strong></td>
<td><strong>Team</strong></td>
<td><strong>Wins</strong></td>
</tr>
<tr>
<td><strong>Season(s)</strong></td>
<td><strong>Winning percentage</strong></td>
<td><strong>Losses</strong></td>
</tr>
</tbody>
</table>
| **Dallas Cowboys**<sup>*</sup> | 5 | 3 | .625 | 1970 02 | 1971 | 1975 | 1977 | 1978 | 2010 | 2011 | 2012 |...
THREE EXTREMELY POWERFUL TOOLS

1) grep

Basic syntax:

grep 'regexp' filename

or equivalently (using UNIX pipelining):

cat filename | grep 'regexp'
WHAT IS A REGULAR EXPRESSION?

A regular expression (regex) describes a set of possible input strings.

Regular expressions descend from a fundamental concept in Computer Science called finite automata theory.

Regular expressions are endemic to Unix

- vi, ed, sed, and emacs
- awk, tcl, perl, and Python
- grep, egrep, fgrep
- compilers
The simplest regular expressions are a string of literal characters to match.

The string *matches* the regular expression if it contains the substring.
UNIX Tools rocks.

match

UNIX Tools sucks.

match

UNIX Tools is okay.

no match
A regular expression can match a string in more than one place.
The . regular expression can be used to match any character.
a|b* denotes \{ε, "a", "b", "bb", "bbb", ...\}

(a|b)* denotes the set of all strings with no symbols other than "a" and "b", including the empty string: \{ε, "a", "b", "aa", "ab", "ba", "bb", "aaa", ...\}

ab*(c) denotes the set of strings starting with "a", then zero or more "b"s and finally optionally a "c": {"a", "ac", "ab", "abc", "abb", "abbc", ...}
Character classes [] can be used to match any specific set of characters.

regular expression → \texttt{b [eor] a t}

\begin{itemize}
  \item \texttt{beat} → match 1
  \item \texttt{brat} → match 2
  \item \texttt{boat} → match 3
\end{itemize}
NEGATED CHARACTER CLASSES

Character classes can be negated with the \[^\] syntax.

\[\text{regular expression} \quad b \ [^eo] \ a \ t\]

\text{beat a \underline{brat} on a boat}

\text{match}
MORE ABOUT CHARACTER CLASSES

- [aeiou] will match any of the characters a, e, i, o, or u
- [kK]orn will match korn or Korn

Ranges can also be specified in character classes

- [1-9] is the same as [123456789]
- [abcde] is equivalent to [a-e]
- You can also combine multiple ranges
  - [abcde123456789] is equivalent to [a-e1-9]
- Note that the - character has a special meaning in a character class but only if it is used within a range, [-123] would match the characters -, 1, 2, or 3
NAMED CHARACTER CLASSES

Commonly used character classes can be referred to by name (*alpha*, *lower*, *upper*, *alnum*, *digit*, *punct*, *cntrl*)

Syntax [:name:] :

- [:alpha:] 
- [:alnum:] 
- [:lower:] 

Important for portability across languages
Anchors are used to match at the beginning or end of a line (or both).

^ means beginning of the line

$ means end of the line
regular expression: `^b [eor] a t$

match: beat a brat on a boat

regular expression: `b [eor] a t $

match: beat a brat on a boat

`^word$ ^$
The * is used to define zero or more occurrences of the single regular expression preceding it.
I got mail, yaaaaaaaaaaaaay!

For me to poop on.
MATCH LENGTH

A match will be the longest string that satisfies the regular expression.

Scrapple from the apple.

A . * e

regular expression

yes

no

no

yes
Ranges can also be specified

- `{ }` notation can specify a range of repetitions for the immediately preceding regex
- `{n}` means exactly $n$ occurrences
- `{n,}` means at least $n$ occurrences
- `{n,m}` means at least $n$ occurrences but no more than $m$ occurrences

Example:

- `{0,}` same as .* 
- `a{2,}` same as aaa*
GREP

• grep comes from the ed (Unix text editor) search command “global regular expression print” or g/re/p

• This was such a useful command that it was written as a standalone utility

• There are two other variants, egrep and fgrep that comprise the grep family

• *grep* is the answer to the moments where you know you want the file that contains a specific phrase but you can’t remember its name
FAMILY DIFFERENCES

- **grep** - uses regular expressions for pattern matching

- **fgrep** - file grep, does not use regular expressions, only matches fixed strings but can get search strings from a file

- **egrep** - extended grep, uses a more powerful set of regular expressions but does not support backreferencing, generally the fastest member of the grep family

- **agrep** – approximate grep; not standard
Sometimes it is handy to be able to refer to a match that was made earlier in a regex.

This is done using **backreferences**

- \n is the backreference specifier, where \n is a number

Looks for \n-th subexpression

For example, to find if the first word of a line is the same as the last:

- `^([[[:alpha:]]){1,}) .* \1$
- The `([[[:alpha:]]){1,})` matches 1 or more letters
PRACTICAL REGEX EXAMPLES

Dollar amount with optional cents
• \$[0-9]+(\.[0-9][0-9])?

Time of day
• (1[012]|1-[9]):[0-5][0-9] (am|pm)

HTML headers <h1> <H1> <h2> ...
• <[hH][1-4]>
Select the string for which the regular expression ‘..\19..’ would find a match:

a) “12.1000”
b) “123.1900”
c) “12.2000”
d) the regular expression does not find a match for any of the strings above
SELECT THE STRING FOR WHICH THE REGULAR EXPRESSION ‘..\.19..’ WOULD FIND A MATCH:

a) “12.1000”

b) “123.1900”

c) “12.2000”

d) the regular expression does not find a match for any of the strings above
Choose the pattern that finds all filenames in which

1. the first letters of the filename are chap,
2. followed by two digits,
3. followed by some additional text,
4. and ending with a file extension of .doc

For example: chap23Production.doc

a) chap[0-9]*.doc
b) chap*[0-9]doc
c) chap[0-9][0-9].*\.doc
d) chap*doc
CLICKER QUESTION II

Choose the pattern that finds all filenames in which

1. the first letters of the filename are chap,
2. followed by two digits,
3. followed by some additional text,
4. and ending with a file extension of .doc

For example: chap23Production.doc

a) chap[0-9]*.doc
b) chap*[0-9]doc
c) chap[0-9][0-9].*\.doc
d) chap*doc
GREP FAMILY

Syntax

grep [-hilnv] [-e expression] [filename]

egrep [-hilnv] [-e expression] [-f filename] [expression] [filename]

fgrep [-hilnxv] [-e string] [-f filename] [string] [filename]

• -h     Do not display filenames
• -i     Ignore case
• -l     List only filenames containing matching lines
• -n     Precede each matching line with its line number
• -v     Negate matches
• -x     Match whole line only (fgrep only)
• -e expression     Specify expression as option
• -f filename     Take the regular expression (egrep) or a list of strings (fgrep) from filename
THREE EXTREMELY POWERFUL TOOLS

1) **grep**
   Basic syntax:
   ```
   grep 'regexp' filename
   ```
   or equivalently (using UNIX pipelining):
   ```
   cat filename | grep 'regexp'
   ```

2) **sed – stream editor**
   Basic syntax
   ```
   sed 's/regexp/replacement/g' filename
   ```
   For each line in the input, the portion of the line that matches regexp (if any) is replaced with replacement.
   Sed is quite powerful within the limits of operating on single line at a time.
   You can use \( \) to refer to parts of the pattern match.
Awk

Finally, awk is a powerful scripting language (not unlike perl). The basic syntax of awk is:

```
awk -F',' 'BEGIN{commands}
  /regexp1/ {command1} /regexp2/ {command2}
  END{commands}'
```

- For each line, the regular expressions are matched in order, and if there is a match, the corresponding command is executed (multiple commands may be executed for the same line).
- BEGIN and END are both optional.
- The -F',' specifies that the lines should be split into fields using the separator ",", and those fields are available to the regular expressions and the commands as $1, $2, etc.
- See the manual (man awk) or online resources for further details.
grep "created_at" twitter.json
   | sed 's/.*"user":{"id":([0-9]*)}.*/1/'
   | sort | uniq -c | sort -n | tail -5"
# DATA WRANGLER / TRIFACTA

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

## Transformer

**Mobile Campaign Project**

<table>
<thead>
<tr>
<th>abc</th>
<th>Event_ID</th>
<th>User_Email</th>
<th>Access_Date</th>
<th>Screen_Detail</th>
<th>Device_Manufacturer</th>
<th>Device_OS_Versi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DCA1000049804</td>
<td>lucitus.vulpate nisi@felis _</td>
<td>2012-09-13</td>
<td>17:37:34</td>
<td>samsung</td>
<td>android 4.3</td>
</tr>
<tr>
<td>2</td>
<td>DCA1000049805</td>
<td><a href="mailto:velis@nunculpinerian.du">velis@nunculpinerian.du</a></td>
<td>2012-10-17</td>
<td>02:43:32</td>
<td>adtran.name-u-target&amp;adtran.so</td>
<td>android 4.0.2</td>
</tr>
<tr>
<td>3</td>
<td>DCA1000049806</td>
<td>nunc.risus.varius@nullolvelpu</td>
<td>2012-11-28</td>
<td>16:43:16</td>
<td>adtran.name-holidaypromo&amp;adtran.so</td>
<td>android 4.0.2</td>
</tr>
<tr>
<td>4</td>
<td>DCA1000049807</td>
<td>fermentum.vel@turpiniscornuil</td>
<td>2012-10-15</td>
<td>05:44:38</td>
<td>adtran.name-holidaypromo&amp;adtran.so</td>
<td>android 4.0.2</td>
</tr>
<tr>
<td>5</td>
<td>DCA1000049808</td>
<td>vulputat.ornare@veliumvelcinici</td>
<td>2012-10-14</td>
<td>16:32:41</td>
<td>adtran.name-holidaypromo&amp;adtran.so</td>
<td>android 4.0.2</td>
</tr>
<tr>
<td>6</td>
<td>DCA1000049809</td>
<td><a href="mailto:Ditis.elementum@Mauris.eu">Ditis.elementum@Mauris.eu</a></td>
<td>2012-11-03</td>
<td>08:22:33</td>
<td>adtran.name-u-target&amp;adtran.so Nokia</td>
<td>android 4.0.2</td>
</tr>
<tr>
<td>7</td>
<td>DCA1000049808</td>
<td>non.ocularis@Vivamus@Proinvslic</td>
<td>2012-10-23</td>
<td>14:56:07</td>
<td>Nokia</td>
<td>android 4.0.2</td>
</tr>
<tr>
<td>8</td>
<td>DCA1000049811</td>
<td><a href="mailto:nec@dictum.ca">nec@dictum.ca</a></td>
<td>2012-11-18</td>
<td>17:16:43</td>
<td>adtran.name-holidaypromo&amp;adtran.so Nokia</td>
<td>android 4.0.2</td>
</tr>
<tr>
<td>9</td>
<td>DCA1000049812</td>
<td>Aenean@Vivamus@nisi</td>
<td>2012-09-27</td>
<td>02:24:50</td>
<td>samsung</td>
<td>android 4.1.1</td>
</tr>
<tr>
<td>10</td>
<td>DCA1000049813</td>
<td>in.hendrerit.consectetuer@eu</td>
<td>2012-10-17</td>
<td>16:36:26</td>
<td>Nokia</td>
<td>android 4.1.1</td>
</tr>
<tr>
<td>11</td>
<td>DCA1000049814</td>
<td>urna.Nunc@ac</td>
<td>2012-10-22</td>
<td>12:49:38</td>
<td>adtran.name-holidaypromo&amp;adtran.so null</td>
<td>android 4.1.1</td>
</tr>
<tr>
<td>12</td>
<td>DCA1000049815</td>
<td>faucibus.Luctus@portitorero</td>
<td>2012-11-12</td>
<td>04:09:55</td>
<td>adtran.name-holidaypromo&amp;adtran.so null</td>
<td>android 4.1.1</td>
</tr>
<tr>
<td>13</td>
<td>DCA1000049816</td>
<td>Donec@amet</td>
<td>2012-12-19</td>
<td>12:55:48</td>
<td>null</td>
<td>android 4.0.2</td>
</tr>
<tr>
<td>14</td>
<td>DCA1000049817</td>
<td><a href="mailto:lobortis@sed.co">lobortis@sed.co</a></td>
<td>2012-12-12</td>
<td>10:16:56</td>
<td>adtran.name-u-target&amp;adtran.so Nokia</td>
<td>windows phone 8.1</td>
</tr>
<tr>
<td>15</td>
<td>DCA1000049818</td>
<td>anet.risus.Donec@Integerlinc</td>
<td>2012-12-16</td>
<td>18:28:18</td>
<td>samsung</td>
<td>windows phone 8.1</td>
</tr>
<tr>
<td>16</td>
<td>DCA1000049819</td>
<td>mollis@<a href="mailto:turpis@nulla.co">turpis@nulla.co</a></td>
<td>2012-10-16</td>
<td>04:17:49</td>
<td>adtran.name-holidaypromo&amp;adtran.so null</td>
<td>windows phone 8.1</td>
</tr>
<tr>
<td>17</td>
<td>DCA1000049820</td>
<td><a href="mailto:orci.adipiscing.non@nassa.co">orci.adipiscing.non@nassa.co</a></td>
<td>2012-11-03</td>
<td>11:47:35</td>
<td>motorola</td>
<td>windows phone 8.1</td>
</tr>
<tr>
<td>18</td>
<td>DCA1000049821</td>
<td>blandit@<a href="mailto:Phasellus@Nulla.co">Phasellus@Nulla.co</a></td>
<td>2012-09-14</td>
<td>02:24:31</td>
<td>adtran.name-holidaypromo&amp;adtran.so null</td>
<td>windows phone 8.1</td>
</tr>
<tr>
<td>19</td>
<td>DCA1000049822</td>
<td>tincidunt.adipiscing.Mauris@</td>
<td>2012-10-13</td>
<td>13:46:24</td>
<td>motorola</td>
<td>windows phone 8.1</td>
</tr>
<tr>
<td>20</td>
<td>DCA1000049823</td>
<td>vel@<a href="mailto:lobortis@Uspidepe.de">lobortis@Uspidepe.de</a></td>
<td>2012-11-11</td>
<td>05:06:07</td>
<td>adtran.name-holidaypromo&amp;adtran.so null</td>
<td>windows phone 8.1</td>
</tr>
<tr>
<td>21</td>
<td>DCA1000049824</td>
<td>Nulla.eu.neque@<a href="mailto:nemo@ullis.co">nemo@ullis.co</a></td>
<td>2012-11-28</td>
<td>20:50:25</td>
<td>adtran.name-holidaypromo&amp;adtran.so null</td>
<td>windows phone 8.1</td>
</tr>
<tr>
<td>22</td>
<td>DCA1000049825</td>
<td>fringilla@<a href="mailto:nulla@eet.org">nulla@eet.org</a></td>
<td>2012-10-08</td>
<td>14:15:43</td>
<td>samsung</td>
<td>android 4.1.1</td>
</tr>
<tr>
<td>23</td>
<td>DCA1000049826</td>
<td>faucebus.Luctus@architectum@nunc</td>
<td>2012-11-14</td>
<td>21:51:54</td>
<td>adtran.name-holidaypromo&amp;adtran.so null</td>
<td>android 4.1.1</td>
</tr>
<tr>
<td>24</td>
<td>DCA1000049827</td>
<td><a href="mailto:nisl.Cum@Donecestmauris.com">nisl.Cum@Donecestmauris.com</a></td>
<td>2012-10-16</td>
<td>14:38:37</td>
<td>ricinus</td>
<td>android 4.1.1</td>
</tr>
<tr>
<td>25</td>
<td>DCA1000049828</td>
<td>parturient.montes.nec@sectetuer@p</td>
<td>2012-10-16</td>
<td>04:06:42</td>
<td>ricinus</td>
<td>android 4.1.1</td>
</tr>
<tr>
<td>26</td>
<td>DCA1000049829</td>
<td>nisl.Quique@fringilla@conse</td>
<td>2012-10-31</td>
<td>03:01:30</td>
<td>ricinus</td>
<td>android 4.1.1</td>
</tr>
</tbody>
</table>

### Transform Editor

- **highlight row**: (date(2012, 11, 7) <= Access_Date) && (Access_Date < date(2012, 12, 27))

### Suggested Transforms

- **highlight row**: (date(2012, 11, 7) <= Access_Date) && (Access_Date < date(2012, 12, 27))
- **split rows** (col1 on: "Row")
- **split col1 on**: "Row" split
- **header**
- **split col**: Access_Time at: 10:11
- **rename col**: column2 to: "Access_Date"
Watch: 10-minute tour of pandas
http://vimeo.com/59324550
IDF WEIGHTED

• IDF: Inverse Document Frequency

\[ wt(\text{word}) = \log_e \left( \frac{\text{size of corpus}}{\text{frequency}(\text{word})} \right) \]

• frequency(word) = defined using some “corpus”:
  • large table of records
  • Wikipedia?
**IDF WEIGHTED JACCARD**

**Microsoft Corporation**

\[
WtJaccard = \frac{12.21}{12.21 + 4.21 + 4.38} = \frac{12.21}{20.8} = 0.59
\]

**Microsoft Corp**

**Microsoft Corporation**

\[
WtJaccard = \frac{4.21}{26.57} = 0.16
\]

**Oracle Corporation**

<table>
<thead>
<tr>
<th>Word</th>
<th>Freq</th>
<th>IDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td>5</td>
<td>12.21</td>
</tr>
<tr>
<td>Oracle</td>
<td>39</td>
<td>10.15</td>
</tr>
<tr>
<td>Corporation</td>
<td>14782</td>
<td>4.21</td>
</tr>
<tr>
<td>Corp</td>
<td>12496</td>
<td>4.38</td>
</tr>
</tbody>
</table>

Corpus size = 1M records