DATA INTEGRATION

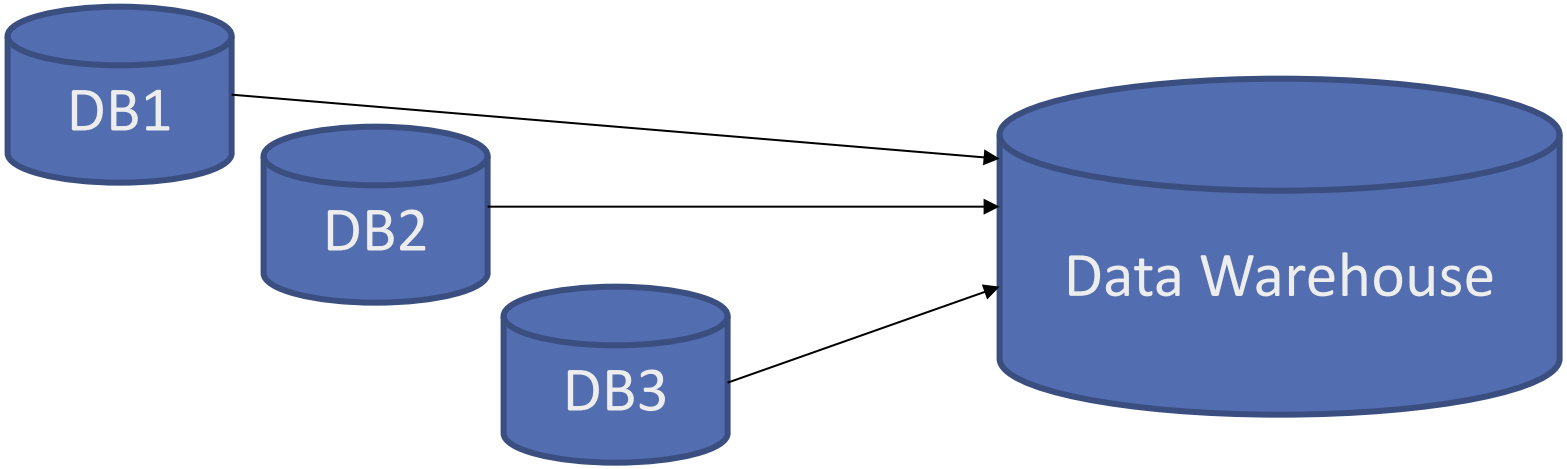
INTRODUCTION TO DATA SCIENCE

CARSTEN BINNIG

BROWN UNIVERSITY

DATA WAREHOUSING

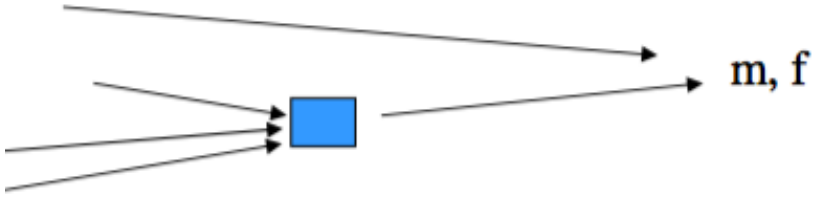
A data warehouse **integrates (inconsistent) data** coming from different sources in a **consistent way**



DB 1 – m, f

DB 2 – male, female

DB 3 – 1,0



inconsistent **consistent**

DATA INTEGRATION

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

- **Data Wrangling**
  - "Hipster" term
  - Less structured
  - Ad-hoc / Incremental

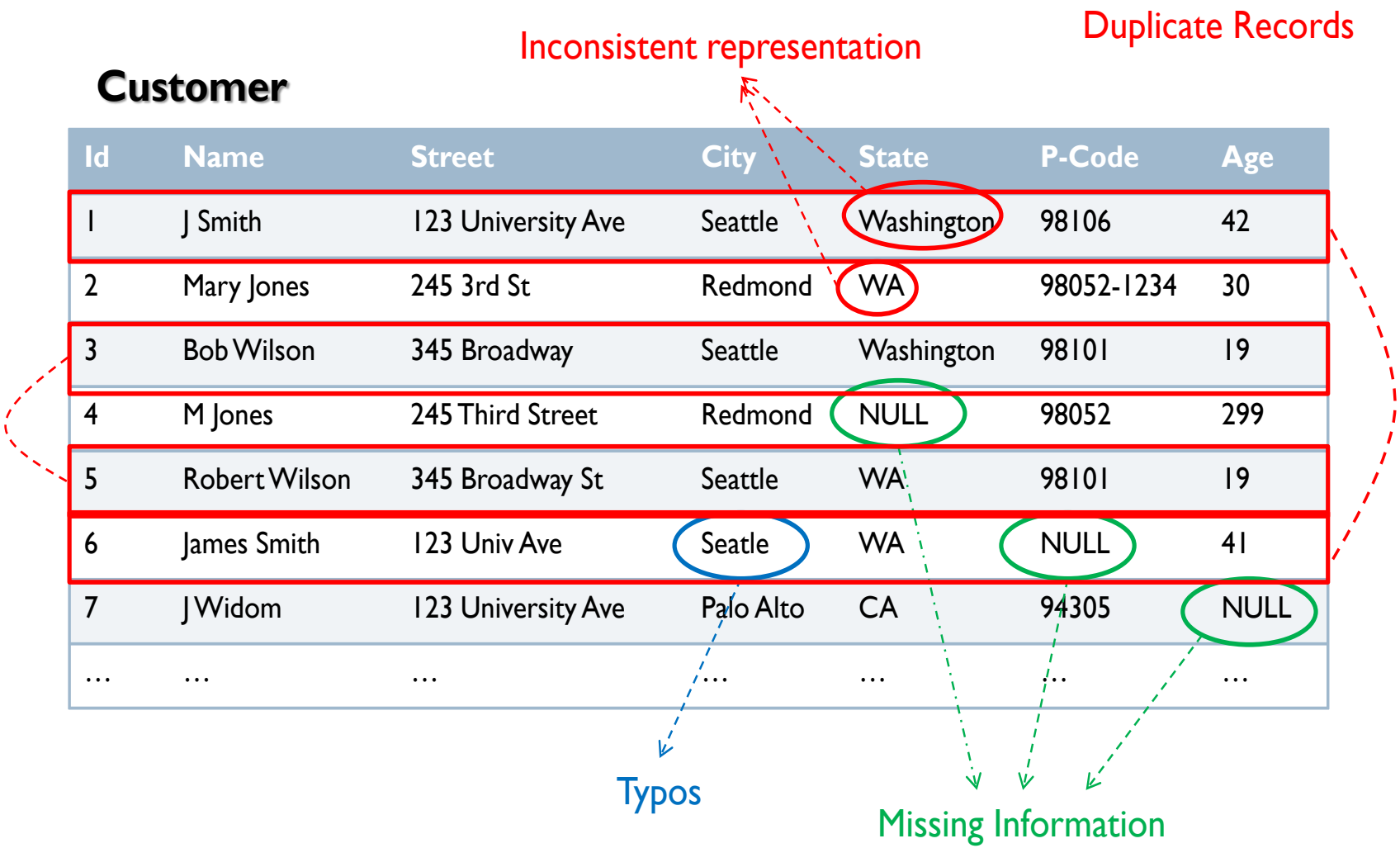
DATA INTEGRATION

REAL WORLD DATA

What is wrong here?

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

REAL WORLD DATA



REAL WORLD DATA

- How many customers do I have?

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

Wrong answer because of duplicate records!

- How many customers by state?

```
select count(*)
from customer
group by state
```

State	Count
AL	60
…	…
…	…
WA	1200
Washington	50
Wasington	2

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)

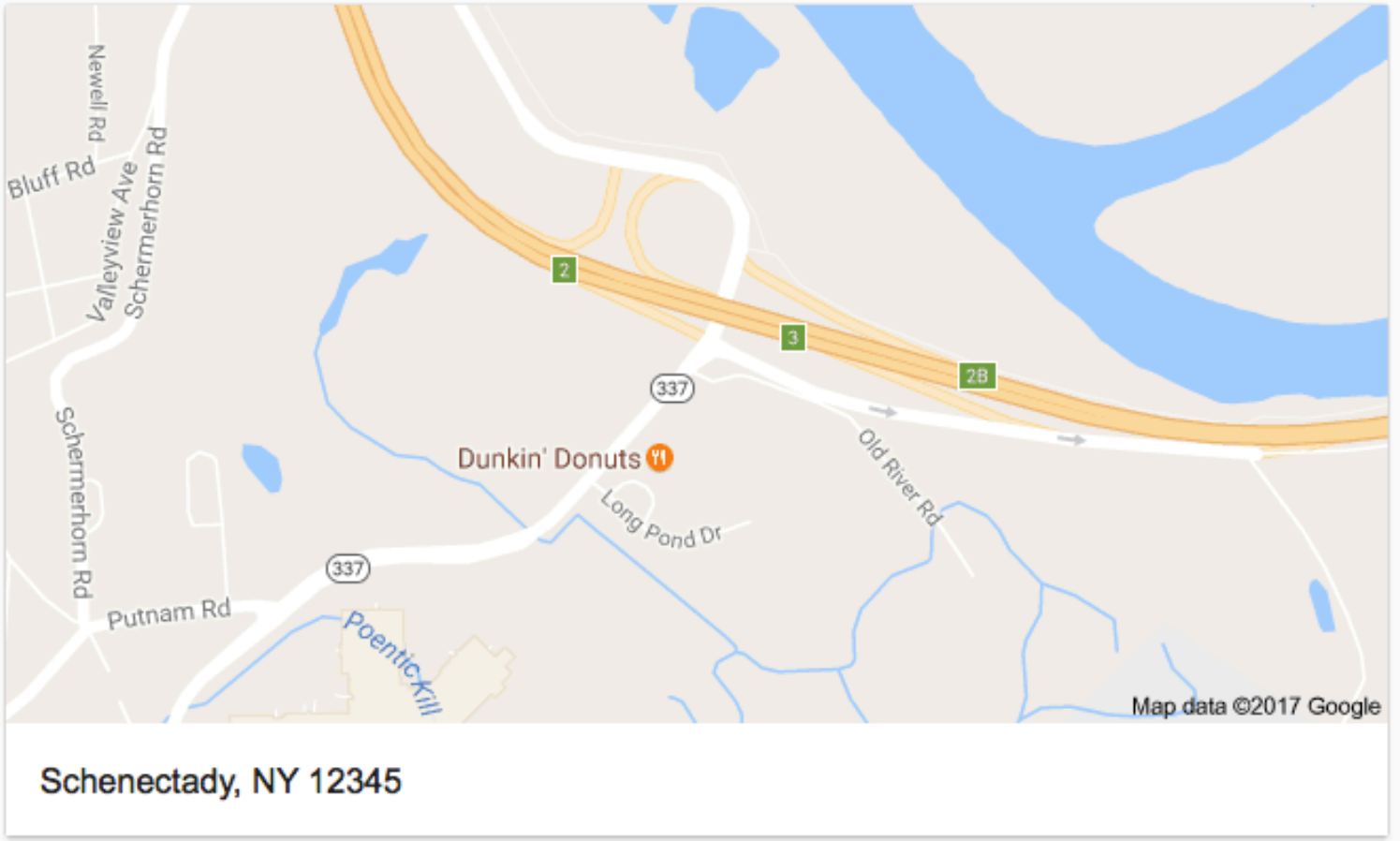
Any combination of the above

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?

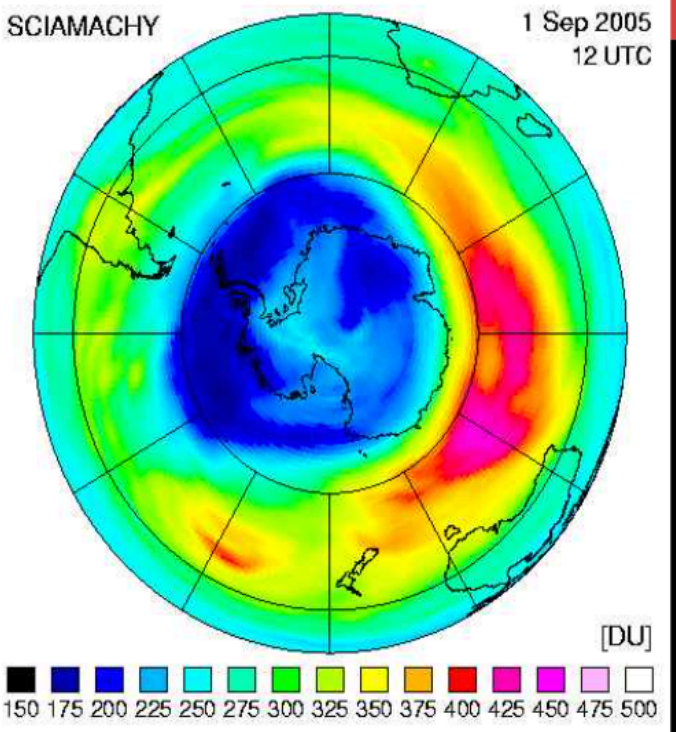


SOLUTION: DETECT OUTLIERS?

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.

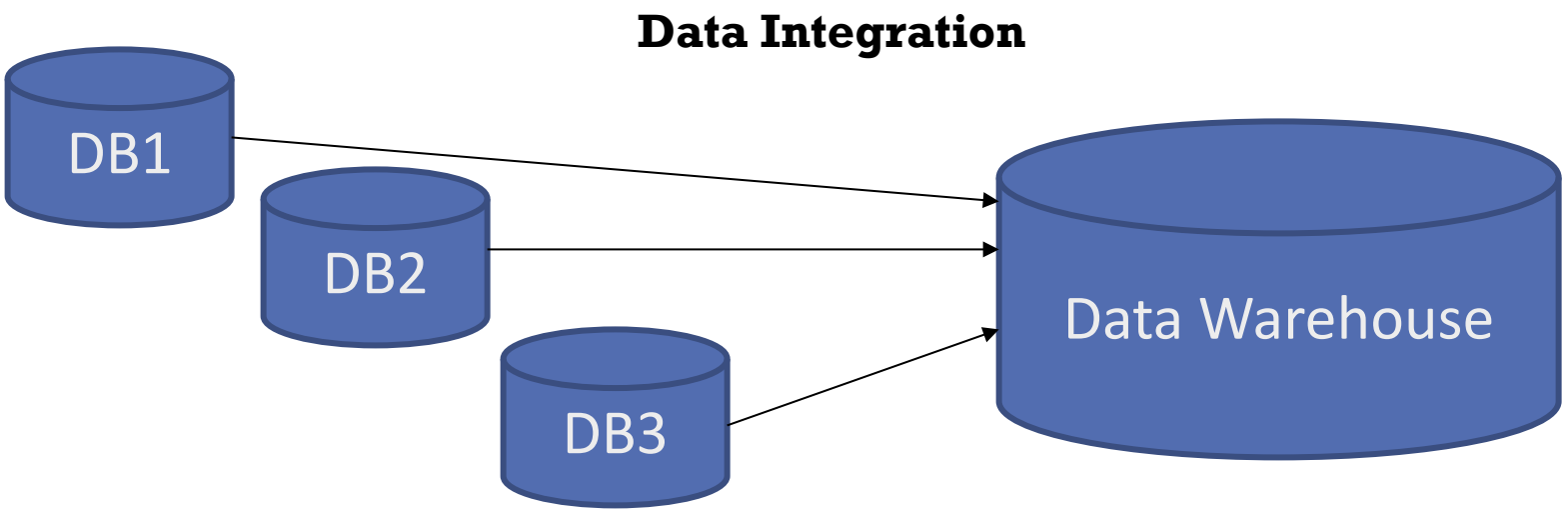
National Center for Atmospheric Research



In fact, the data were **rejected as unreasonable** by data quality control algorithms

DATA INTEGRATION

DATA WAREHOUSING



Multiple Independent Schemata

One Integrated Schemata

SCHEMA MATCHING

ID	Name	Address	Zip	State	City	Phone	E-Mail
1	Tim Kraska	135 Watermann St,	02906	Providence	RI	+1 234234 234	Tim_kraska@brown.edu
…	…	…				..	..

ID	Name	Address	Phone	E-Mail
1	Tim Kraska	135 Watermann St, 02906 Providence, RI	+1 234234 234	Tim_kraska@brown.edu
…	…	…	..	..

ID	Name
1	Tim Kraska
…	…

AddressID	Person-ID	Address	Phone-Nb	E-Mail
1	1	135 Watermann St, 02906 Providence	+1 234234 234	Tim_kraska@brown.edu
1	1	222 Hope St, 02906 Providence	980 – 0803284	tim_kraska@brown.edu

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?

ID	Name	Address	Zip	State	City	Phone	E-Mail
1	Tim Kraska	135 Watermann St,	02906	Providence	RI	+1 234234 234	Tim_kraska @brown.edu
…	…	…				..	..

ID	Name	Addr	Mobile	E-Mail
1	Tim Kraska	135 Watermann St, 02906 Providence, RI	+1 234234 234	Tim_kraska @brown.edu
…	…	…	..	..

ID	Name
1	Tim Kraska
…	…

Address ID	Person-ID	Address	Phone-Nb	E-Mail
1	1	135 Watermann St, 02906 Providence	+1 234234 234	Tim_kraska @brown.edu
1	1	222 Hope St, 02906 Providence	980 – 0803284	tim_kraska @brown.edu

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 (pervious matching results, user input,..)

DATA INTEGRATION

deduplication, entity clustering, merge/purge, record linkage, approximate match...

EXAMPLE

ID	Product Name	Price
r1	iPad Two 16GB WiFi White	$490
r2	iPad 2nd generation 16GB WiFi White	$469
r3	iPhone 4th generation White 16GB	$545
r4	Apple iPhone 4 16GB White	$520
r5	Apple iPhone 3rd generation Black 16GB	$375
r6	iPhone 4 32GB White	$599
r7	Apple iPad2 16GB WiFi White	$499
r8	Apple iPod shuffle 2GB Blue	$49
r9	Apple iPod shuffle USB Cable	$19

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
- …

TEXT SIMILARITY

Customer

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

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) = EditDstance(s2, s1)

"Distance" = opposite of similarity

EDIT DISTANCE

EditDistance ("Provdince", "Providence") = 2

Provdince → Providince → Providence

EditDistance("Seattle", "Redmond") = 6

Seattle → Reattle → Redttle

Redmtle → Redmole → Redmone

→ Redmond

EDIT DISTANCE

115th Waterman St., Providence, RI

EditDistance = 1

110th Waterman St., Providence, RI

Waterman Street, Providence, RI

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**

$$Jaccard(s1, s2) = \frac{|s1 \cap s2|}{|s1 \cup s2|}$$

- **Range of values = [0,1]**

JACCARD SIMILARITY

148th Ave NE, Redmond, WA

140th Ave NE, Redmond, WA

$$Jaccard = \frac{4}{4+2} \approx 0.66$$

JACCARD SIMILARITY

148th Ave NE, Redmond, WA

NE 148th Ave, Redmond, WA

$$Jaccard = \frac{5}{5} = 1.0$$

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

↕ Jaccard = 1/3

Microsoft Corp

B)

Microsoft Corporation

↕ Jaccard = 1/3

Oracle Corporation

C)

Waterman 115 St

↕ Jaccard = 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

$$Weight\ Function = wt: Elements \rightarrow \mathbb{R}^{+}$$

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

$$wt(s) = \sum_{e \in s} wt(e)$$

wt("Microsoft") > wt("Corporation")

Wt("Oracle") > wt("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.

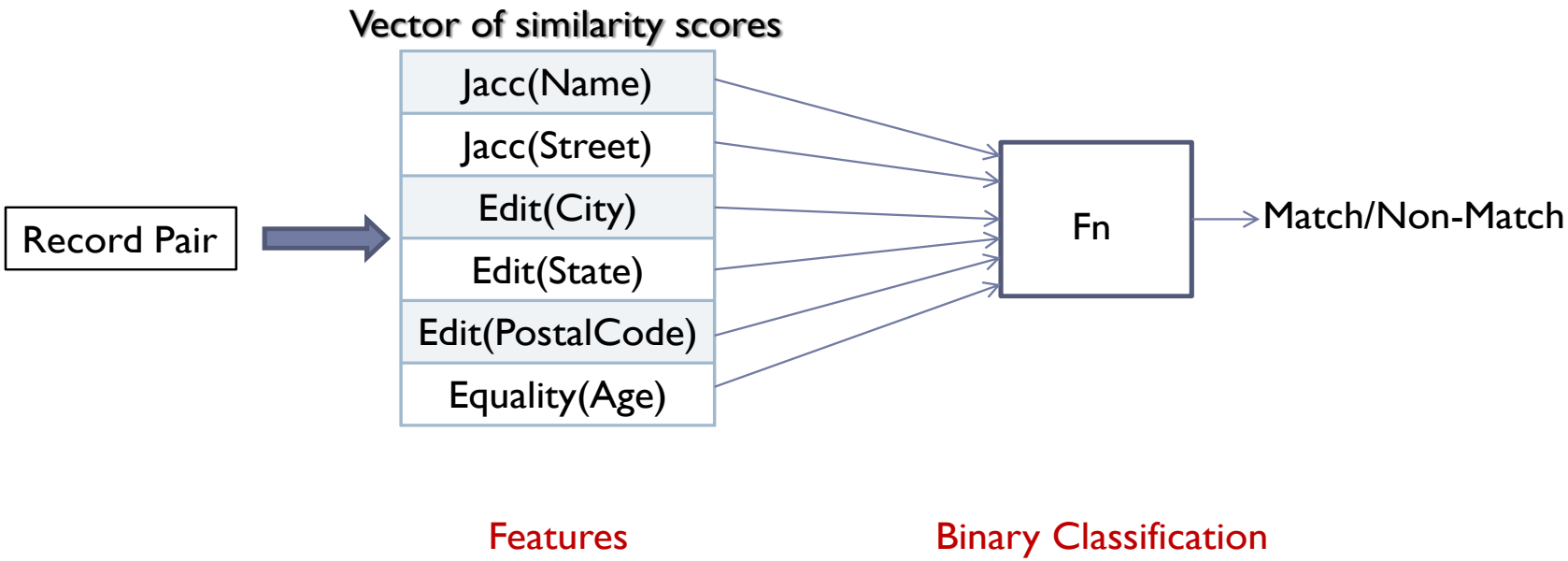
RECORD MATCHING PROBLEMS

Customer

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

$WtJaccard =$ 0.57 0.91 1.0 0.0 1.0 1.0

COMBINING SIMILARITY FUNCTIONS



Idea: Weighted sum of per attribute similarity + threshold?

LEARNING-BASED APPROACH

Bob Wilson	345 Broadway	Seattle	Washington	98101	19
Robert Wilson	345 Broadway St	Seattle	WA	98101	19

Match

B Wilson	123 Broadway	Boise	Idaho	83712	19
Robert Wilson	345 Broadway St	Seattle	WA	98101	19

Non-Match

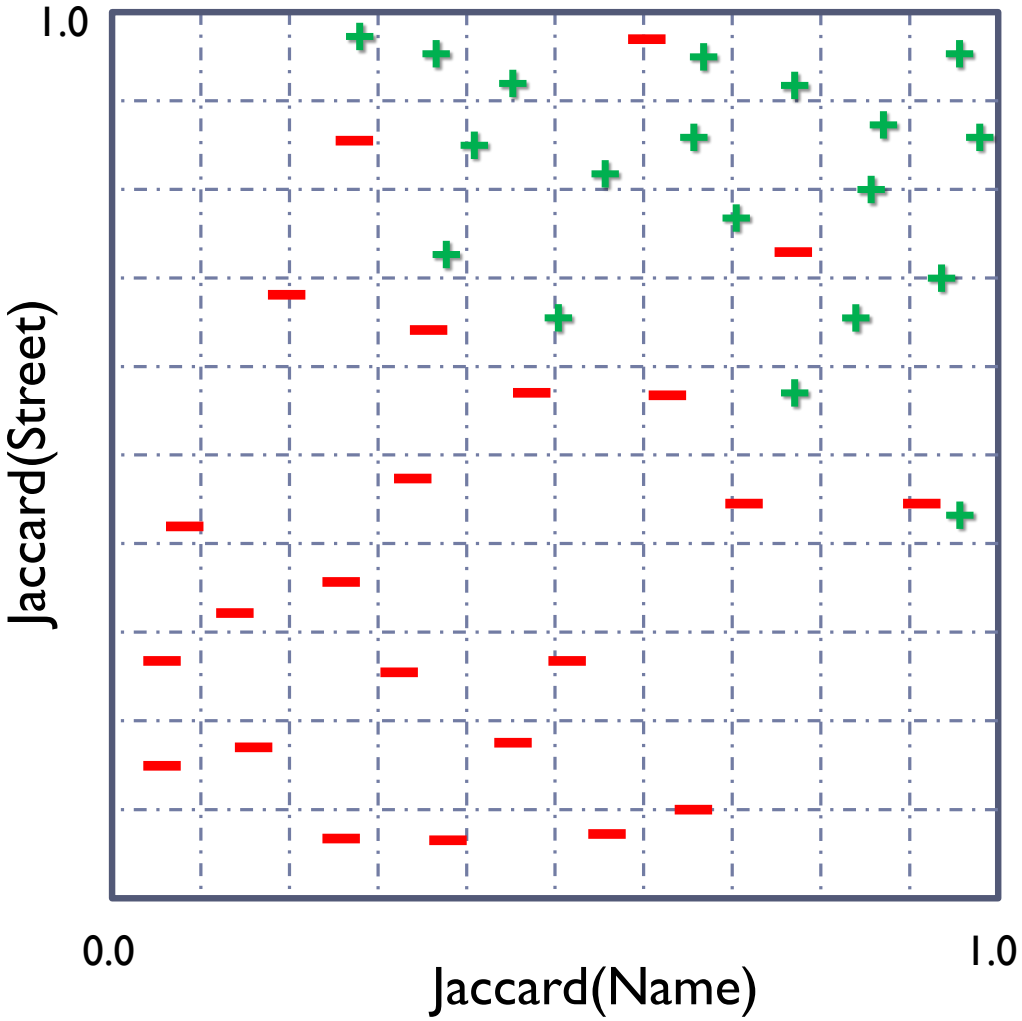
Mary Jones	245 3rd St	Redmond	WA	98052-1234	30
M Jones	245 Third Street	Redmond	NULL	98052	299

Match

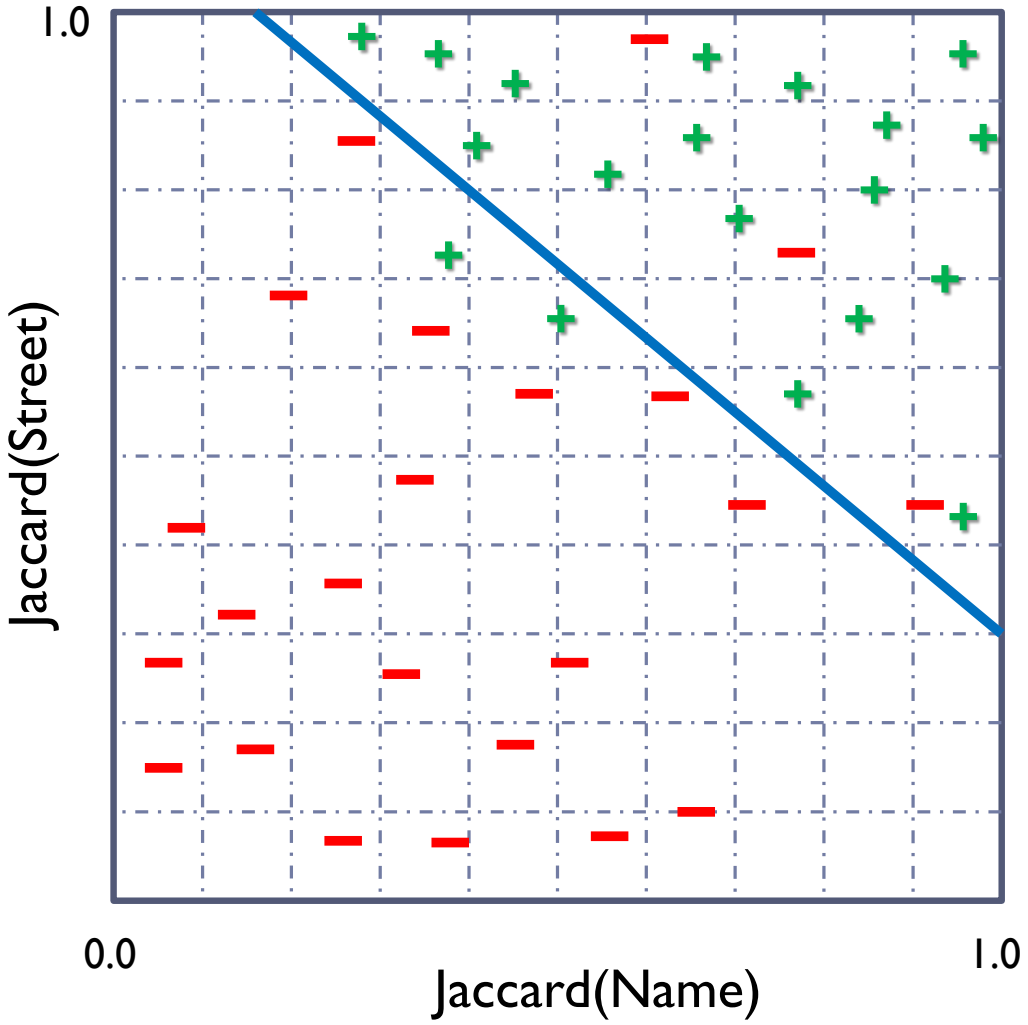
Mary Jones	245 3rd St	Redmond	WA	98052-1234	30
Robert Wilson	345 Broadway St	Seattle	WA	98101	19

Non-Match

LEARNING BASED APPROACH



LEARNING BASED APPROACH

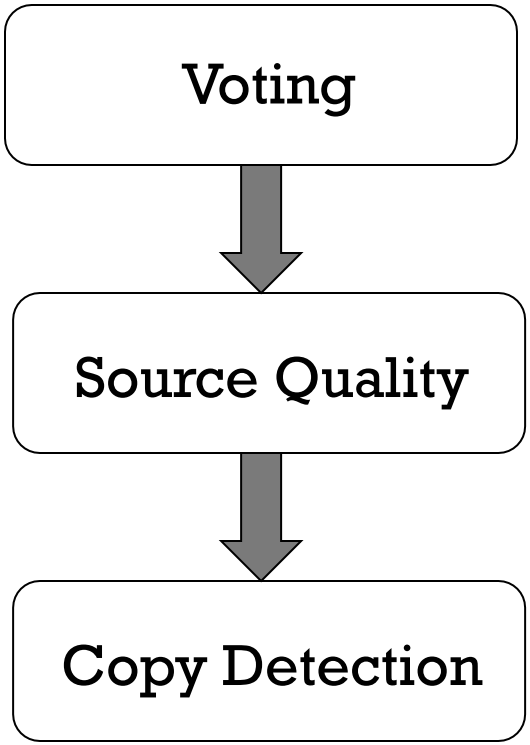


DATA INTEGRATION

DATA FUSION'S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

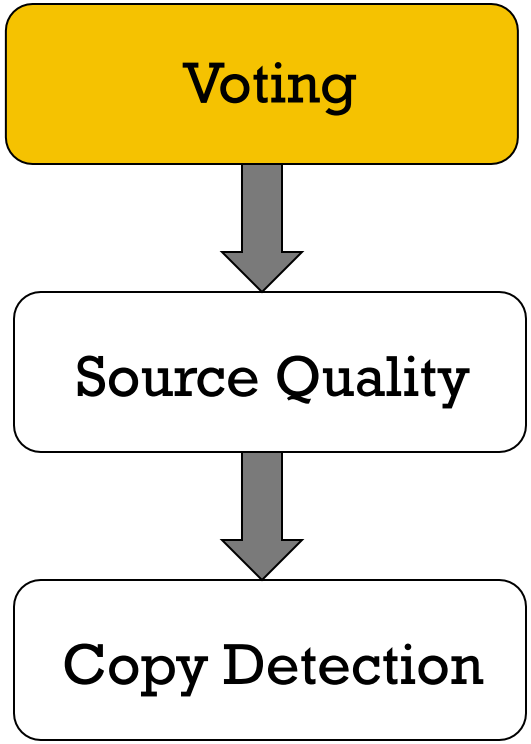
- Resolves inconsistency across diversity of sources



	S1	S2	S3	S4	S5
Jagadish	UM	ATT	UM	UM	UI
Dewitt	MSR	MSR	UW	UW	UW
Bernstein	MSR	MSR	MSR	MSR	MSR
Carey	UCI	ATT	BEA	BEA	BEA
Franklin	UCB	UCB	UMD	UMD	UMD

DATA FUSION'S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

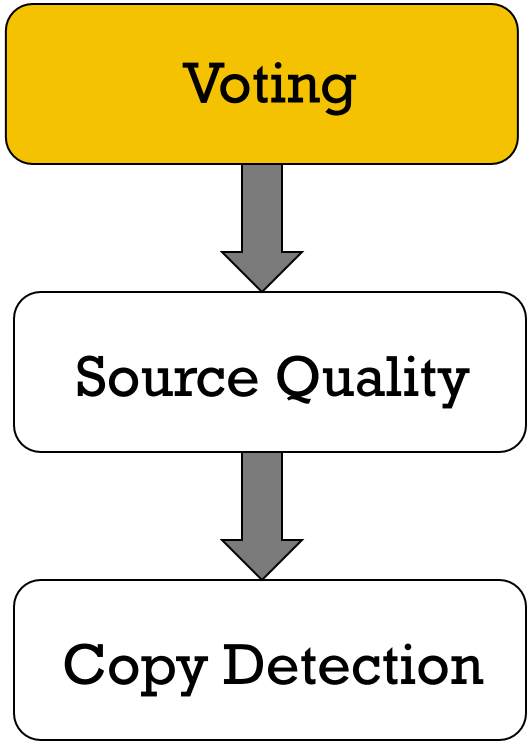


	S1	S2	S3
Jagadish	UM	ATT	UM
Dewitt	MSR	MSR	UW
Bernstein	MSR	MSR	MSR
Carey	UCI	ATT	BEA
Franklin	UCB	UCB	UMD

DATA FUSION'S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

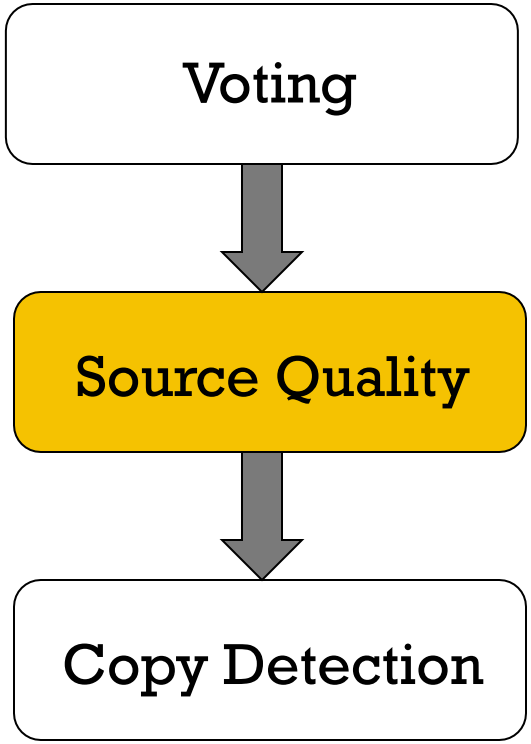
- Supports difference of opinion



	S1	S2	S3
Jagadish	UM	ATT	UM
Dewitt	MSR	MSR	UW
Bernstein	MSR	MSR	MSR
Carey	UCI	ATT	BEA
Franklin	UCB	UCB	UMD

DATA FUSION'S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

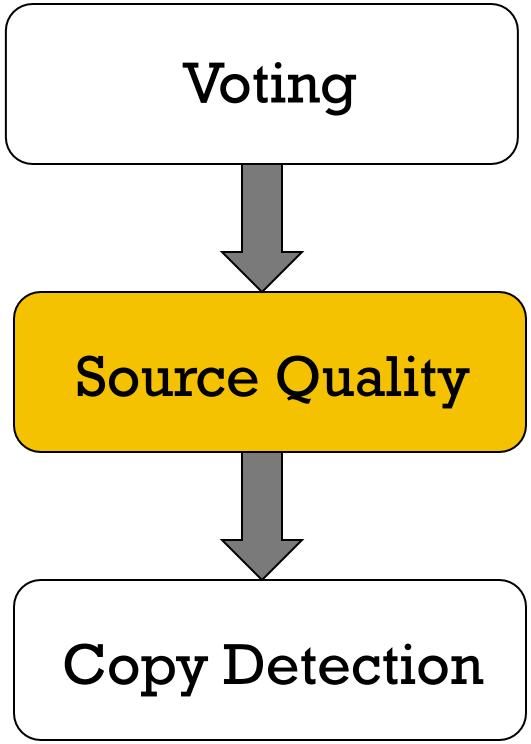


	S1	S2	S3
Jagadish	UM	ATT	UM
Dewitt	MSR	MSR	UW
Bernstein	MSR	MSR	MSR
Carey	UCI	ATT	BEA
Franklin	UCB	UCB	UMD

DATA FUSION'S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

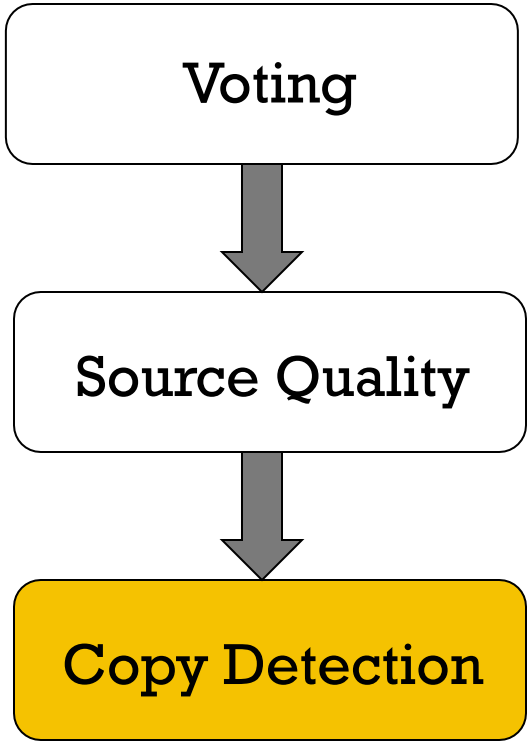
- Gives more weight to knowledgeable sources



	S1	S2	S3
Jagadish	UM	ATT	UM
Dewitt	MSR	MSR	UW
Bernstein	MSR	MSR	MSR
Carey	UCI	ATT	BEA
Franklin	UCB	UCB	UMD

DATA FUSION'S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

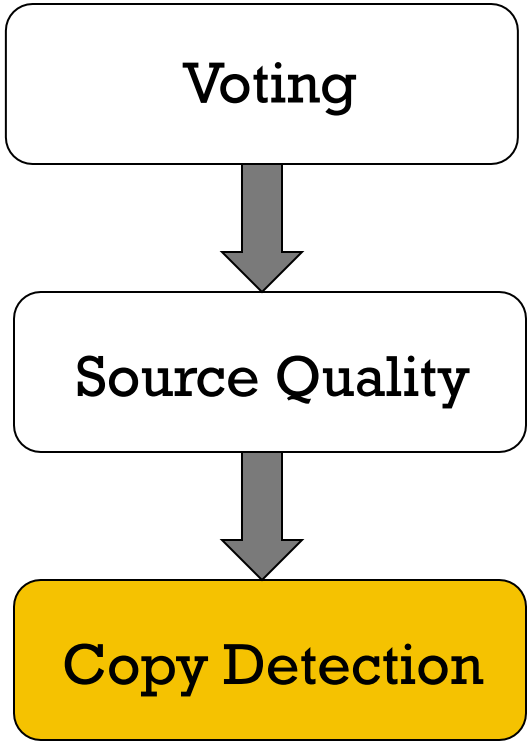


	S1	S2	S3	S4	S5
Jagadish	UM	ATT	UM	UM	UI
Dewitt	MSR	MSR	UW	UW	UW
Bernstein	MSR	MSR	MSR	MSR	MSR
Carey	UCI	ATT	BEA	BEA	BEA
Franklin	UCB	UCB	UMD	UMD	UMD

DATA FUSION'S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

- Reduces weight of copier sources



	S1	S2	S3	S4	S5
Jagadish	UM	ATT	UM	UM	UI
Dewitt	MSR	MSR	UW	UW	UW
Bernstein	MSR	MSR	MSR	MSR	MSR
Carey	UCI	ATT	BEA	BEA	BEA
Franklin	UCB	UCB	UMD	UMD	UMD

DATA INTEGRATION SO FAR

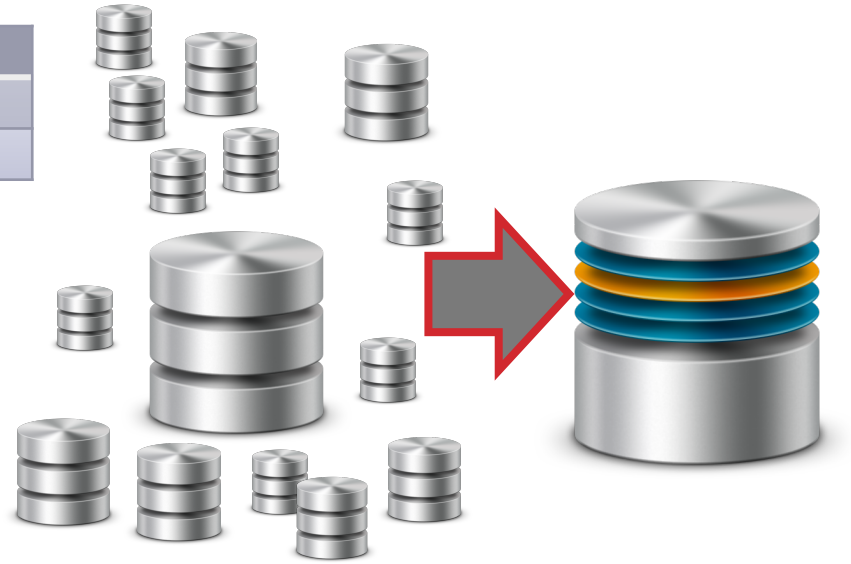
But what are some practical tools?

SO FAR: RELATIONAL DATA

A	B	C	D
1	b	b	b
2	s	e	f

F	A	G
a	1	g
a	1	g

A	B	F
a	2	c
3	2	v



```
{|class="wikitable sortable"
|-
!Appearances
!Team
!Wins
!Losses
!Winning<br />percentage
!Season(s)
|-align=center
| {{Sort|0862|8}}||align=left style="background:#fcc;"|[[Pittsburgh Steelers]]<sup>†</sup><ref group=note name=e />
| 6|| 2|| .750
|align=left|{{Sort|1974 02|'''[[Super Bowl IX|1974]]''',<sup>†</sup> '''[[Super Bowl X|1975]]''',<sup>†</sup> '''[[Super Bowl
XIII|1978]]''',<sup>†</sup> '''[[Super Bowl XIV|1979]]''',<sup>†</sup> [[Super Bowl XXX|1995]],<sup>†</sup> '''[[Super Bowl
XL|2005]]''',<sup>†</sup><ref group=note name=c /> '''[[Super Bowl XLIII|2008]]''',<sup>†</sup> [[Super Bowl
XLV|2010]]<sup>†</sup>}}
|-align=center
|{{Sort|0853|8}}||align=left style="background:#d0e7ff;"|[[Dallas Cowboys]]<sup>*</sup>
| 5|| 3|| .625
|align=left|{{Sort|1970 02|[[Super Bowl V|1970]],<sup>*</sup> '''[[Super Bowl VI|1971]]''',<sup>*</sup> [[Super Bowl
X|1975]],<sup>*</sup><ref group=note name=c /> '''[[Super Bowl XII|1977]]''',<sup>*</sup> [[Super Bowl XIII|
....
```

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

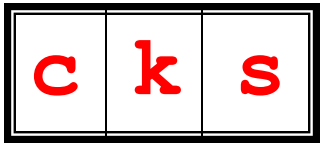
Regular expressions are endemic to Unix

- vi, ed, sed, and emacs
- awk, tcl, perl and Python
- grep, egrep, fgrep
- compilers

REGULAR EXPRESSIONS

The simplest regular expressions are a string of literal characters to match.

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

regular expression → 


```
UNIX Tools rocks.
```

match

```
UNIX Tools sucks.
```

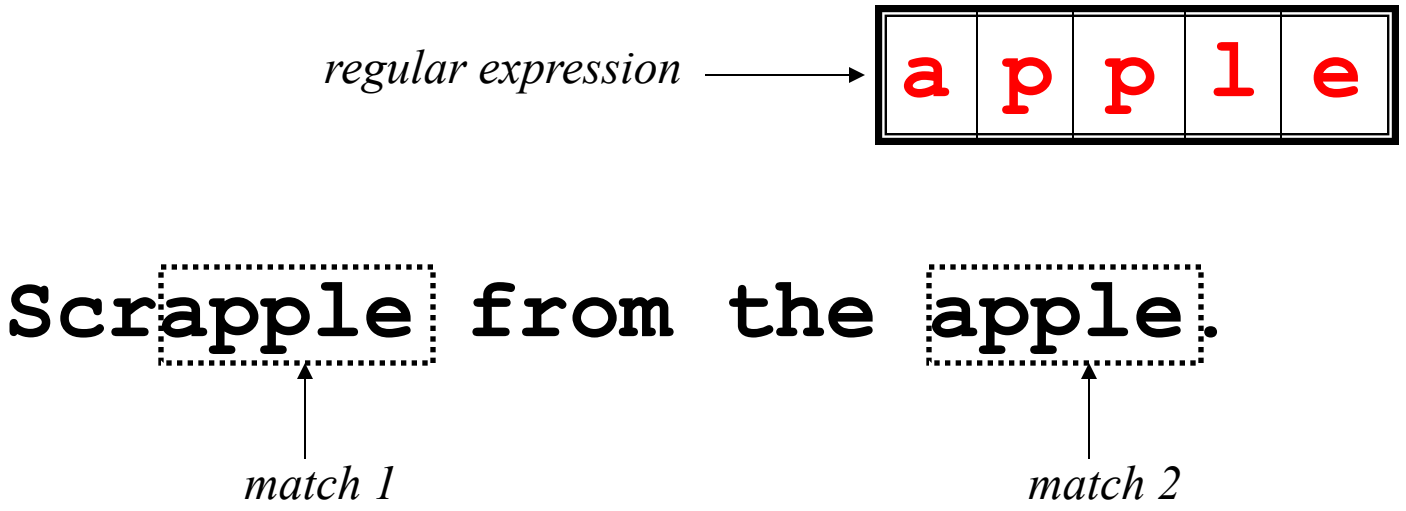
match

```
UNIX Tools is okay.
```

no match

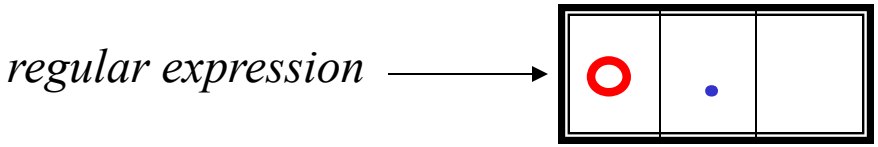
REGULAR EXPRESSIONS

A regular expression can match a string in more than one place.



REGULAR EXPRESSIONS

The . regular expression can be used to match any character.



OR

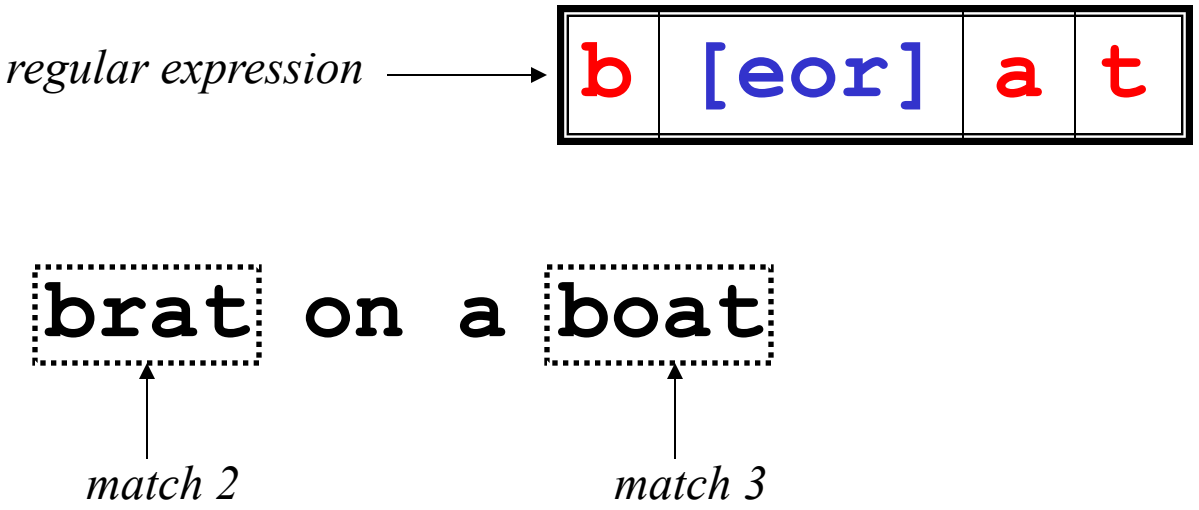
a|b* denotes {ε, "a", "b", "bb", "bbb", ...}

(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", ...}

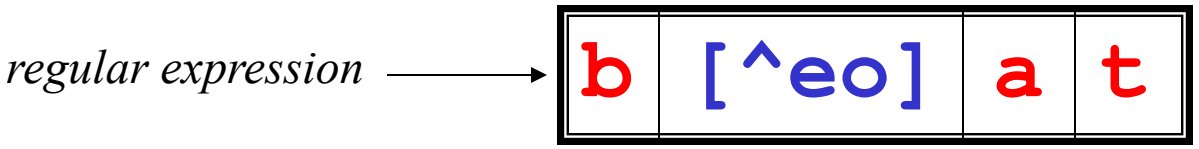
CHARACTER CLASSES

Character classes [] can be used to match any specific set of characters.



NEGATED CHARACTER CLASSES

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




```
beat a brat on a boat
```

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*:]

- [a-zA-Z] [[:alpha:]]
- [a-zA-Z0-9] [[:alnum:]]
- [45a-z] [45[:lower:]]

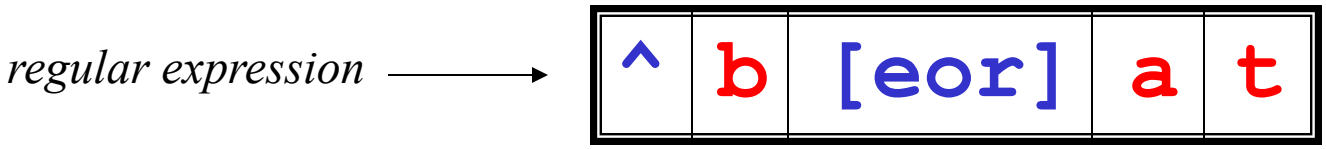
Important for portability across languages

ANCHORS

Anchors are used to match at the beginning or end of a line (or both).

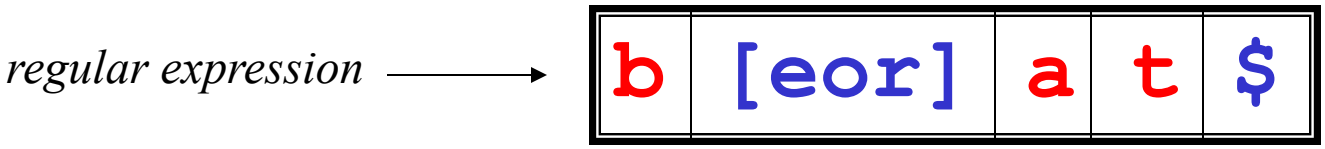
^ means beginning of the line

$ means end of the line




```
beat a brat on a boat
```

match




```
beat a brat on a boat
```

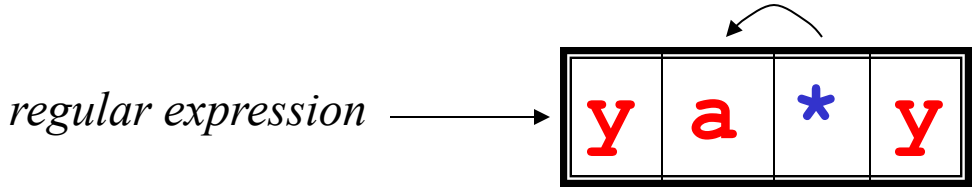
match

`^word$`

`^$`

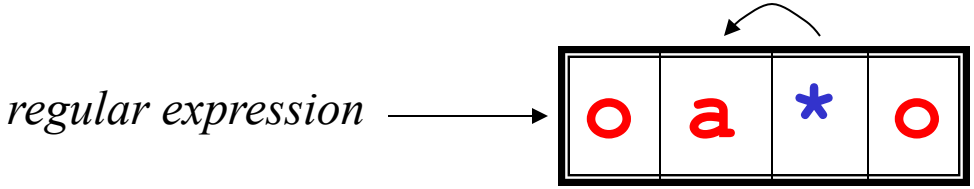
REPETITION

The * is used to define zero or more occurrences of the *single* regular expression preceding it.



I got mail, yaaaaaaaaaay!

match



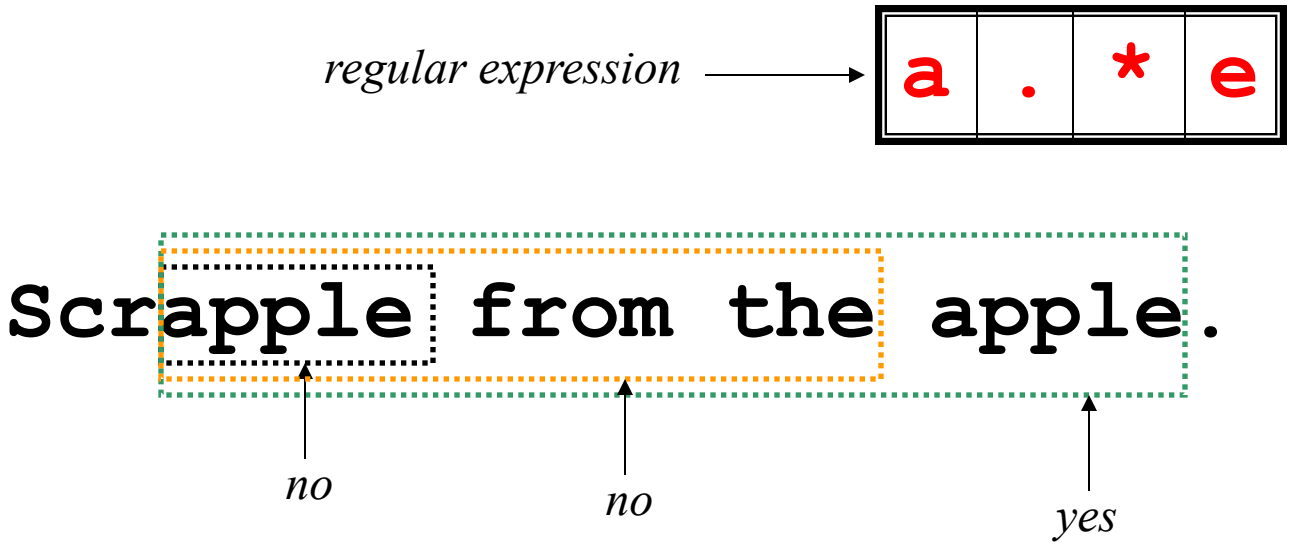
For me to poop on.

match

.*

MATCH LENGTH

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



REPETITION RANGES

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

GREP: BACKREFERENCES

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]>

CLICKER QUESTION I

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

CLICKER QUESTION I

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

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

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 intput, 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.

THREE EXTREMELY POWERFUL TOOLS

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.

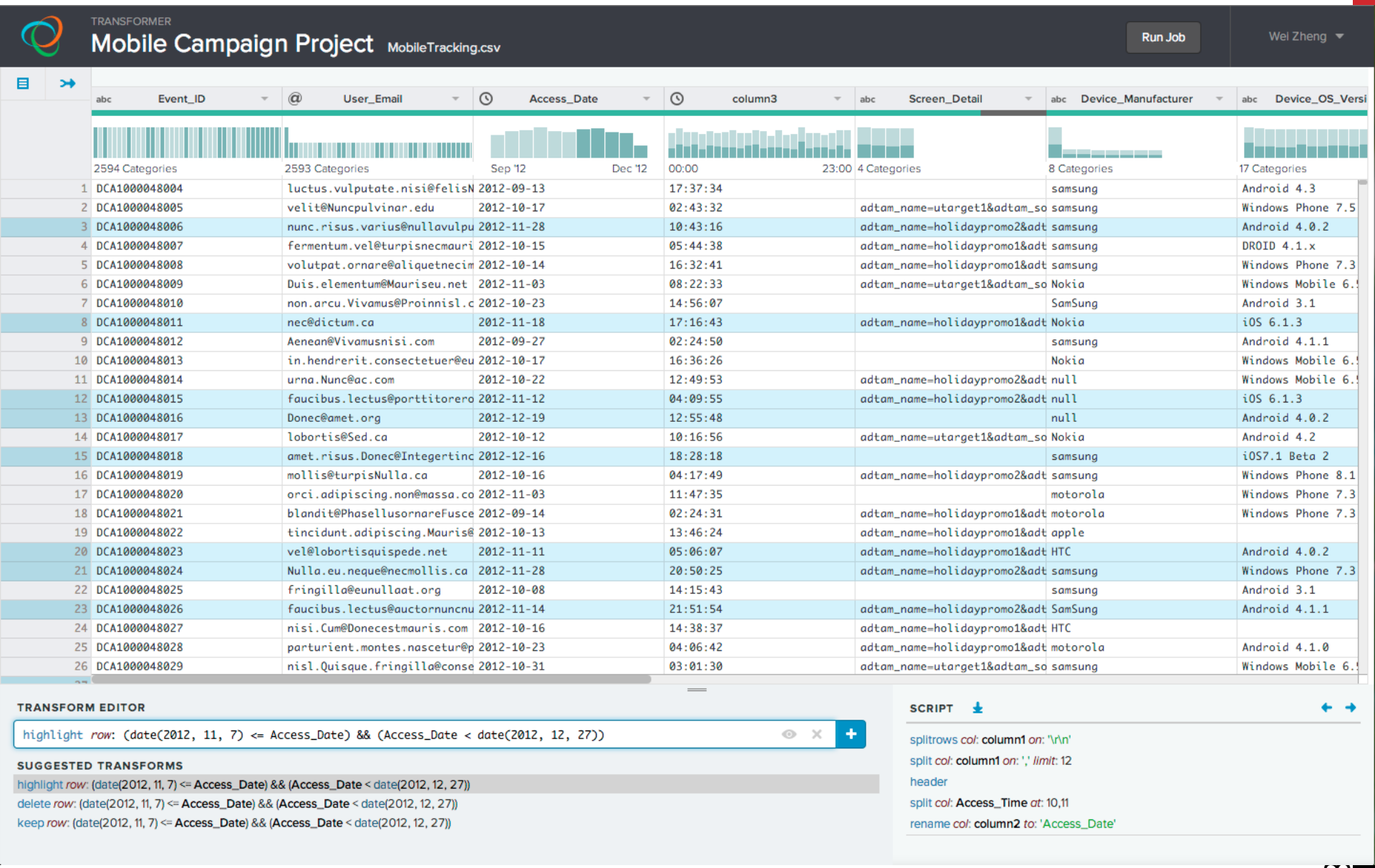
EXAMPLE

```
grep "created\_at" twitter.json
  | sed 's/.*"user":{"id":\([0-9]*\).*/\1/'
  | sort | uniq -c | sort -n | tail -5"
```


```
{"created_at":"Sat Aug 31 06:57:23 +0000
2013","id":373701031776882688,"id_str":"373701031776882688","text":"\u5ddd\u5d0e\u3055\u309
3\u306e\u5bbf\u984c\u3084\u3063\u3066\u304f\u308c\u308b\u5fc3\u512a\u3057\u3044\u65b9\u306f
\u5c45\u3089\u306c\u306e\u3067\u3059\u304b\uff1f\u2190","source":"\u003ca
href=\"http:\/\/twitter.com\/download\/iphone\" rel=\"nofollow\"\u003eTwitter for
iPhone\u003c\/a\u003e","truncated":false,"in_reply_to_status_id":null,"in_reply_to_status_i
d_str":null,"in_reply_to_user_id":null,"in_reply_to_user_id_str":null,"in_reply_to_screen_n
ame":null,"user":{"id":1580127176,"id_str":"1580127176","name":"\u3061\u306e\u3071\u3093","
screen_name":"08_chi_02","location":"","url":null,"description":"\u305f\u3060\u306e\u304a\u
3070\u304b\u3067\u3059\u3002\u306f\u3044\u3002\n\u3078\u3093\u3066\u3053\u6ce8\u610f\u21af"
,"protected":false,"followers_count":130,"friends_count":149,"listed_count":0,"created_at":
"Tue Jul 09 11:23:26 +0000
2013","favourites_count":62,"utc_offset":32400,"time_zone":"Tokyo","geo_enabled":false,"ver
ified":false,"statuses_count":489,"lang":"ja","contributors_enabled":false,"is_translator":
false,"profile_background_color":"C0DEED","profile_background_image_url":"http:\/\/a0.twimg
.com\/images\/themes\/theme1\/bg.png","profile_background_image_url_https":"https:\/\/si0.t
wimg.com\/images\/themes\/theme1\/bg.png","profile_background_tile":false,"profile_image_ur
l":"http:\/\/a0.twimg.com\/profile_images\/378800000306401177\/a8912f698459a84e7343d19ac90f
6fa0_normal.jpeg","profile_image_url_https":"https:\/\/si0.twimg.com\/profile_images\/37880
0000306401177\/a8912f698459a84e7343d19ac90f6fa0_normal.jpeg","profile_banner_url":"https:\/
\/pbs.twimg.com\/profile_banners\/1580127176\/1377526337","profile_link_color":"0084B4","pr
```

DATA WRANGLER / TRIFACTA

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



PANDAS

Watch: 10-minute tour of pandas
http://vimeo.com/59324550

IDF WEIGHTED

- IDF: Inverse Document Frequency

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

- frequency(word) = defined using some “corpus”:
  - large table of records
  - Wikipedia?

IDF WEIGHTED JACCARD

Microsoft Corporation

WtJaccard = 12.21 / (12.21 + 4.21 + 4.38)

= 12.21/20.8 = 0.59

Microsoft Corp

$$\log_e\left(\frac{1{,}000{,}000}{5}\right)$$

Word	Freq	IDF
Microsoft	5	12.21
Oracle	39	10.15
Corporation	14782	4.21
Corp	12496	4.38

Microsoft Corporation

WtJaccard = 4.21/26.57 = 0.16

Oracle Corporation

Corpus size = 1M records