DS-GA 3001.009: Responsible Data Science

Data Profiling

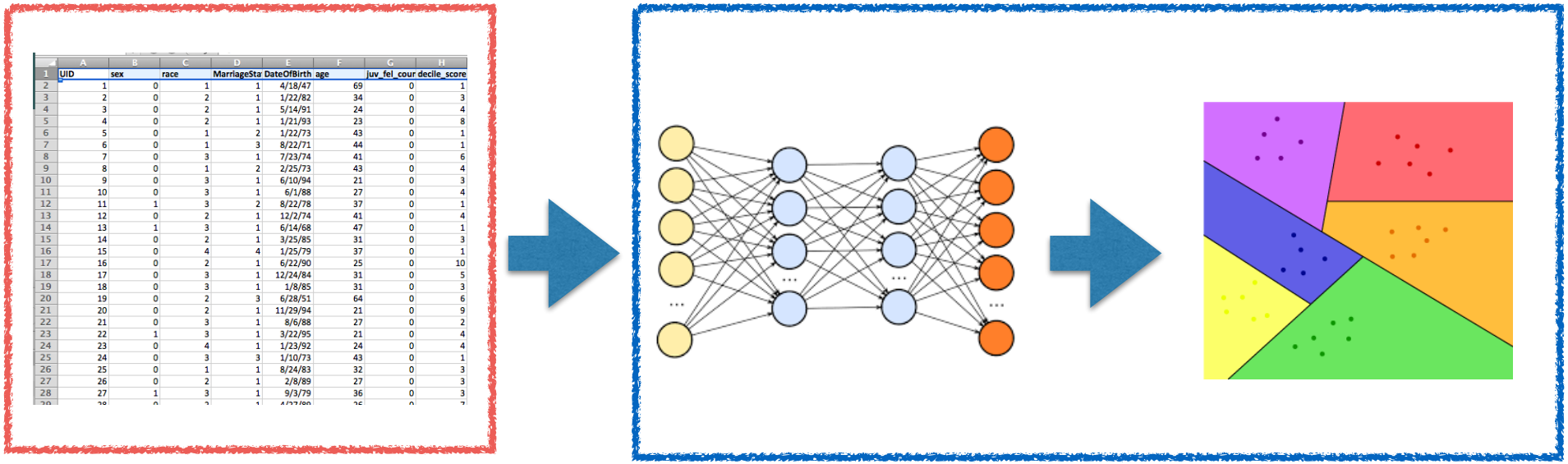
Prof. Julia Stoyanovich
Center for Data Science
Computer Science and Engineering at Tandon

@stoyanoj

http://stoyanovich.org/
https://dataresponsibly.github.io/

Responsible data science

- Be **transparent** and **accountable**
- Achieve **equitable** resource distribution
- Be cognizant of the **rights** and **preferences** of individuals

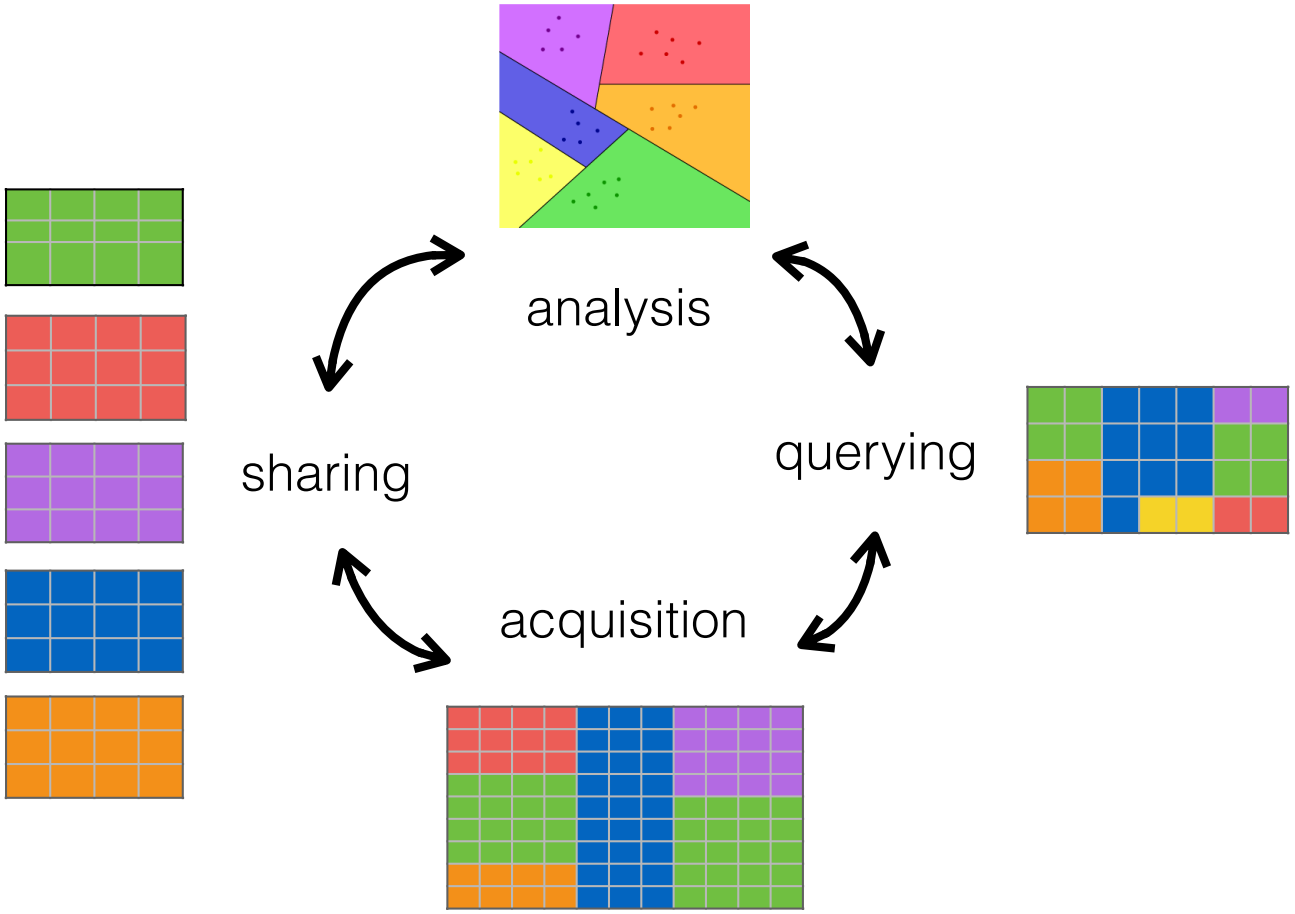


UID	sex	race	MarriageSta	DateOfBirth	age	juv_fel_cour	decile_score
1	0	1	1	4/18/47	69	0	1
2	0	2	1	1/22/82	34	0	3
3	0	2	1	5/14/91	24	0	4
4	0	2	1	1/21/93	23	0	8
5	0	1	2	1/22/73	43	0	1
6	0	1	3	8/22/71	44	0	1
7	0	3	1	7/23/74	41	0	6
8	0	1	2	2/25/73	43	0	4
9	0	3	1	6/10/94	21	0	3
10	0	3	1	6/1/88	27	0	4
11	1	3	2	8/22/78	37	0	1
12	0	2	1	12/2/74	41	0	4
13	1	3	1	6/14/68	47	0	1
14	0	2	1	3/25/85	31	0	3
15	0	4	4	1/25/79	37	0	1
16	0	2	1	6/22/90	25	0	10
17	0	3	1	12/24/84	31	0	5
18	0	3	1	1/8/85	31	0	3
19	0	2	3	6/28/51	64	0	6
20	0	2	1	11/29/94	21	0	9
21	0	3	1	8/6/88	27	0	2
22	1	3	1	3/22/95	21	0	4
23	0	4	1	1/23/92	24	0	4
24	0	3	3	1/10/73	43	0	1
25	0	1	1	8/24/83	32	0	3
26	0	2	1	2/8/89	27	0	3
27	1	3	1	9/3/79	36	0	3

done?

but where does the data come from?

A holistic view of the lifecycle



Understand your data!

CRA
Computing Research Association

"Given the heterogeneity of the flood of data, it is **not enough merely to record it and throw it into a repository**. Consider, for example, data from a range of scientific experiments. If we just have a bunch of data sets in a repository, it is **unlikely anyone will ever be able to find, let alone reuse**, any of this data. With adequate **metadata**, there is some hope, but even so, challenges will remain due to differences in experimental details and in data record structure."

https://cra.org/ccc/wp-content/uploads/sites/2/2015/05/bigdatawhitepaper.pdf

Understand your data!

Need **metadata** to:

- enable data **re-use** (have to be able to find it!)
- determine **fitness for use** of a dataset in a task
- help establish **trust** in the data analysis process and its outcomes

Data is considered to be of high quality if it's "**fit for intended uses** in operations, decision making and planning"

Thomas C. Redman, "Data Driven: Profiting from Your Most Important Business Asset." 2013]

NYC | OpenData 311 | Search all NYC.gov websites

NYC OpenData

Home Data About Learn Contact Us Blog | Sign In

Open Data for All New Yorkers

Open Data is free public data published by New York City agencies and other partners. **Sign up for the NYC Open Data mailing list** to learn about training opportunities and upcoming events!

Search Open Data for things like 311, Buildings, Crime

How You Can Get Involved

New to Open Data
Learn what data is and how to get started with our How To.

Data Veterans
View details on Open Data APIs and check status alerts.

Get in Touch
Ask a question, leave a comment, or suggest a dataset to the NYC Open Data team.

Dive into the Data
Already know what you're looking for? Browse the data catalog now.

2010 SAT (College Board) School Level Results

Education

NYC OpenData

New York City school level College Board SAT results for the graduating seniors of 2010. Records contain 2010 College-bound seniors mean SAT scores.

summary

Records with 5 or fewer students are suppressed (marked 's').

privacy

College-bound seniors are those students that complete the SAT Questionnaire when they register for the SAT and identify that they will graduate from high school in a specific year. For example, the 2010 college-bound seniors are those students that self-reported they would graduate in 2010. Students are not required to complete the SAT Questionnaire in order to register for the SAT. Students who do not indicate which year they will graduate from high school will not be included in any college-bound senior report.

Students are linked to schools by identifying which school they attend when registering for a College Board exam. A student is only included in a school's report if he/she self-reports being enrolled at that school.

description

Updated September 17, 2018

freshness

Data Provided by Department of Education (DOE)

source

2010 SAT (College Board) School Level Results

Education

OpenData

About this Dataset

Updated
September 17, 2018

Data Last Updated
October 6, 2011

Metadata Last Updated
September 17, 2018

Date Created
October 6, 2011

Views
23.7K

Downloads
9,076

Data Provided by
Department of Education (DOE)

Dataset Owner
NYC OpenData

Update

Update Frequency	Historical Data
Automation	No
Date Made Public	10/11/2011

Dataset Information

Agency	Department of Education (DOE)

Attachments

SAT Data Dictionary.xlsx

Topics

Category	Education
Tags	doe, sat

2010 SAT (College Board) School Level Results

Education

NYC OpenData

What's in this Dataset?

Rows	Columns
460	6

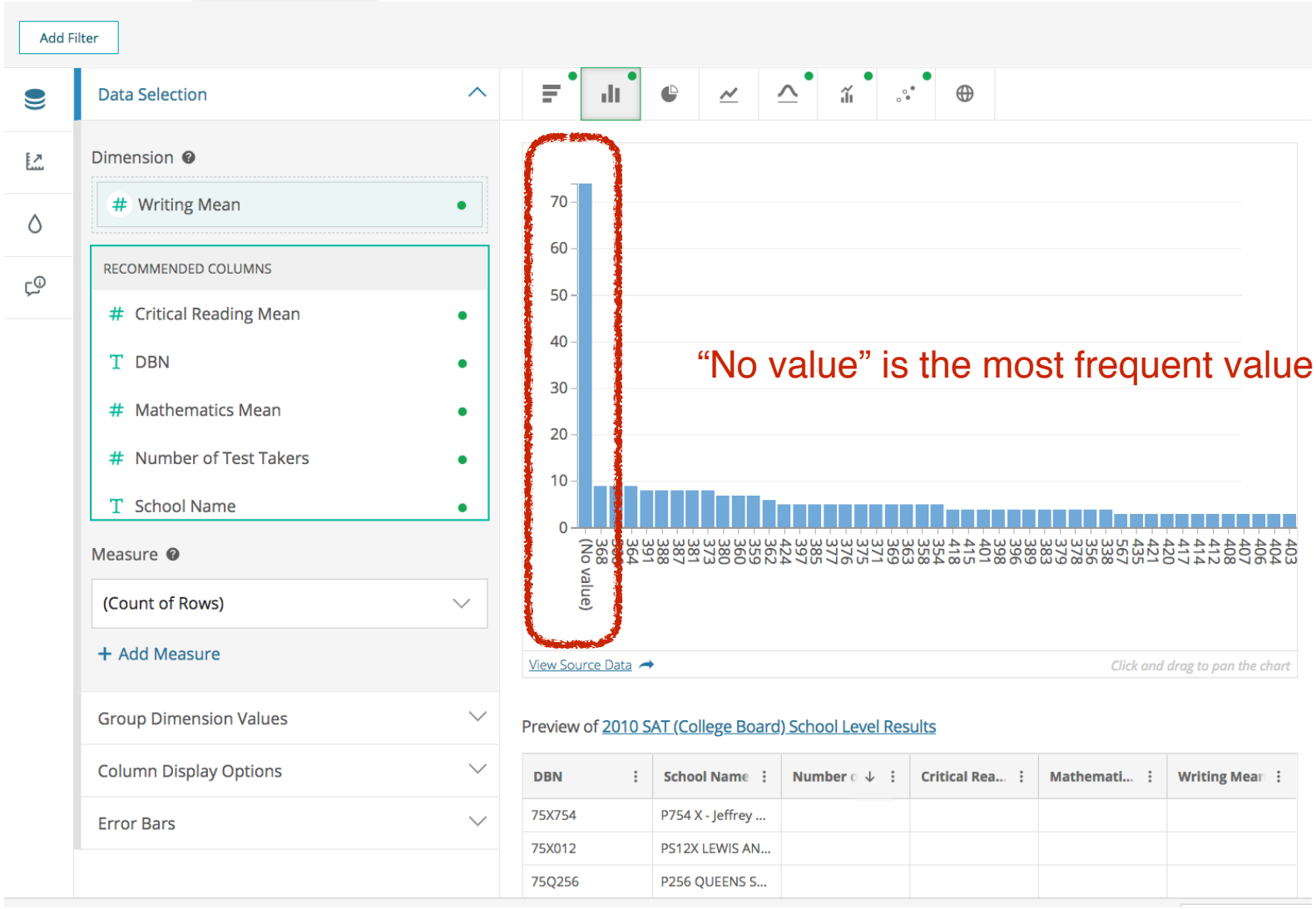
Columns in this Dataset

Column Name	Description	Type
DBN		Plain Text T
School Name		Plain Text T
Number of Test Takers		Number #
Critical Reading Mean		Number #
Mathematics Mean		Number #
Writing Mean		Number #

2010 SAT (College Board) School Level Results

Education

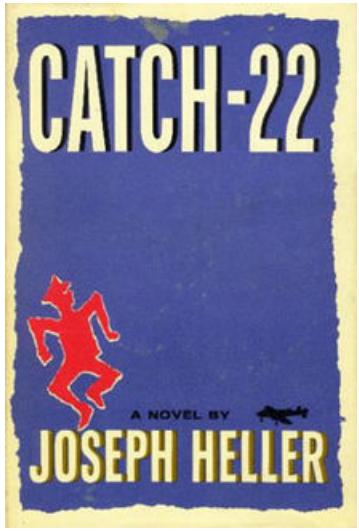
NYC OpenData



Data profiling

- A set of activities and processes to determine the metadata about a given dataset
- Metadata summarizes the data, summaries should be **small** but **informative**.
- What is informative depends on the task, or set of tasks, we have in mind.

should profiling be task-agnostic or task-specific?



- A related activity is **data cleaning**

Data cleaning

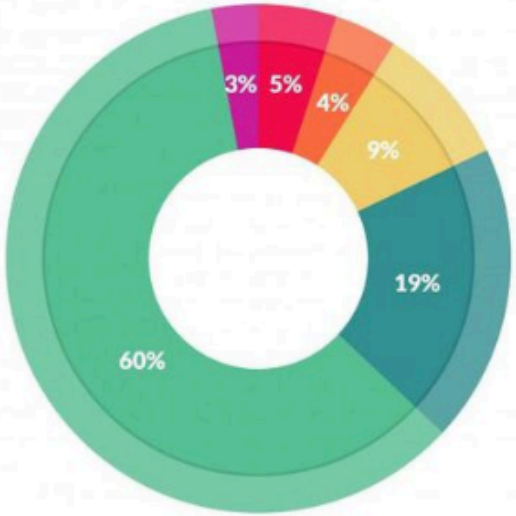
52,423 views | Mar 23, 2016, 09:33am

Forbes

Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says

Gil Press Contributor ⓘ

I write about technology, entrepreneurs and innovation.



What data scientists spend the most time doing

- *Building training sets: 3%*
- *Cleaning and organizing data: 60%*
- *Collecting data sets; 19%*
- *Mining data for patterns: 9%*
- *Refining algorithms: 4%*
- *Other: 5%*

Spend most time doing
Collecting data (19%)
Cleaning and organizing data (60%)

based on a slide by Heiko Mueller

Data cleaning

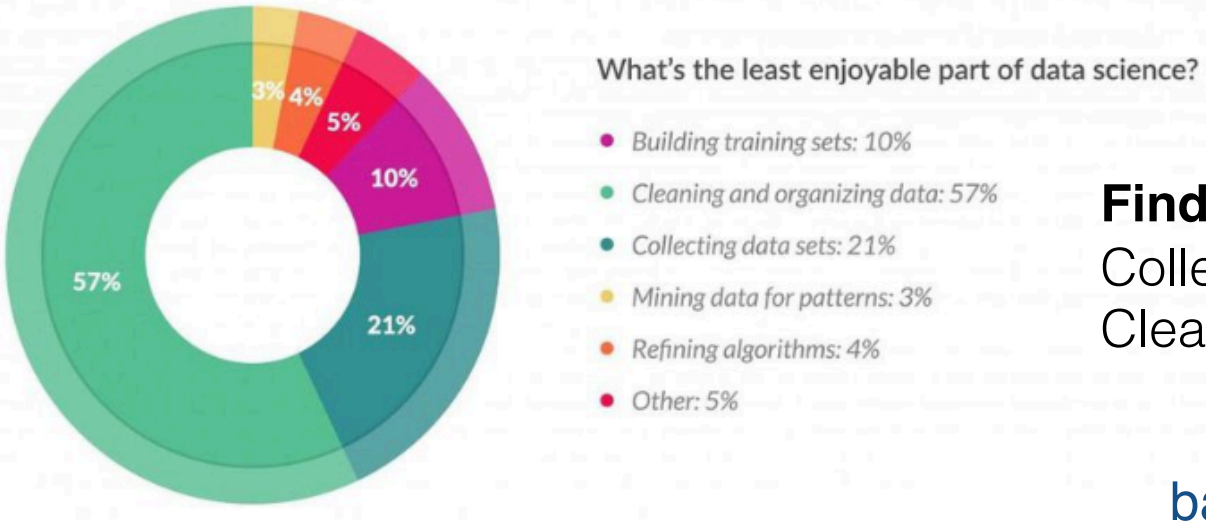
52,423 views | Mar 23, 2016, 09:33am

Forbes

Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says

Gil Press Contributor ⓘ

I write about technology, entrepreneurs and innovation.



Find least enjoyable

Collecting data (21%)

Cleaning and organizing data (57%)

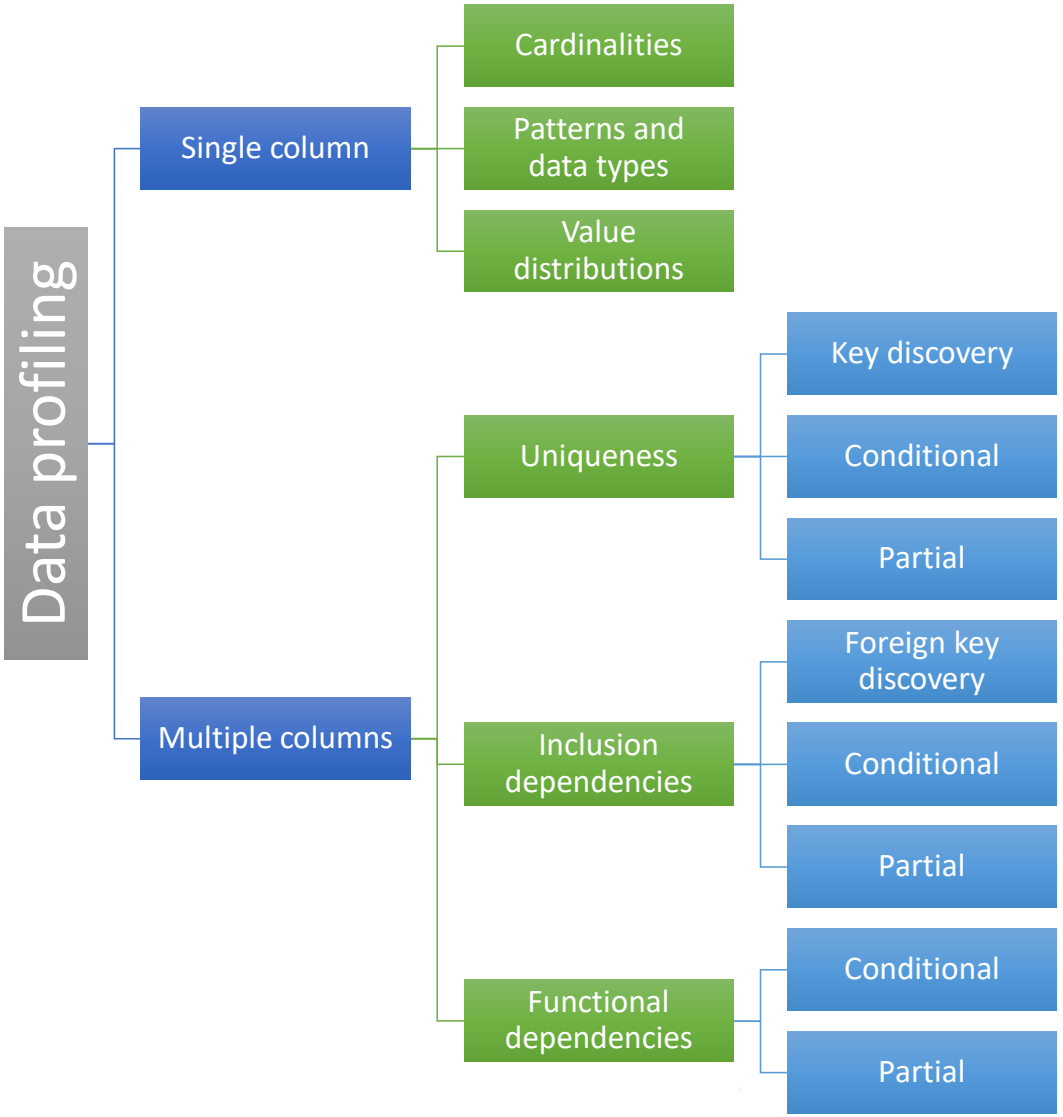
based on a slide by Heiko Mueller

A classification of data profiling tasks

[Abedjan, Golab, Naumann; *SIGMOD 2017*]

	A	B	C	D	E	F	G	H
1	UID	sex	race	MarriageSta	DateOfBirth	age	juv_fel_cour	decile_score
2	1	0	1	1	4/18/47	69	0	1
3	2	0	2	1	1/22/82	34	0	3
4	3	0	2	1	5/14/91	24	0	4
5	4	0	2	1	1/21/93	23	0	8
6	5	0	1	2	1/22/73	43	0	1
7	6	0	1	3	8/22/71	44	0	1
8	7	0	3	1	7/23/74	41	0	6
9	8	0	1	2	2/25/73	43	0	4
10	9	0	3	1	6/10/94	21	0	3
11	10	0	3	1	6/1/88	27	0	4
12	11	1	3	2	8/22/78	37	0	1
13	12	0	2	1	12/2/74	41	0	4
14	13	1	3	1	6/14/68	47	0	1
15	14	0	2	1	3/25/85	31	0	3
16	15	0	4	4	1/25/79	37	0	1
17	16	0	2	1	6/22/90	25	0	10
18	17	0	3	1	12/24/84	31	0	5
19	18	0	3	1	1/8/85	31	0	3
20	19	0	2	3	6/28/51	64	0	6
21	20	0	2	1	11/29/94	21	0	9
22	21	0	3	1	8/6/88	27	0	2
23	22	1	3	1	3/22/95	21	0	4
24	23	0	4	1	1/23/92	24	0	4
25	24	0	3	3	1/10/73	43	0	1
26	25	0	1	1	8/24/83	32	0	3
27	26	0	2	1	2/8/89	27	0	3
28	27	1	3	1	9/3/79	36	0	3

relational data (here: just one table)



An alternative classification

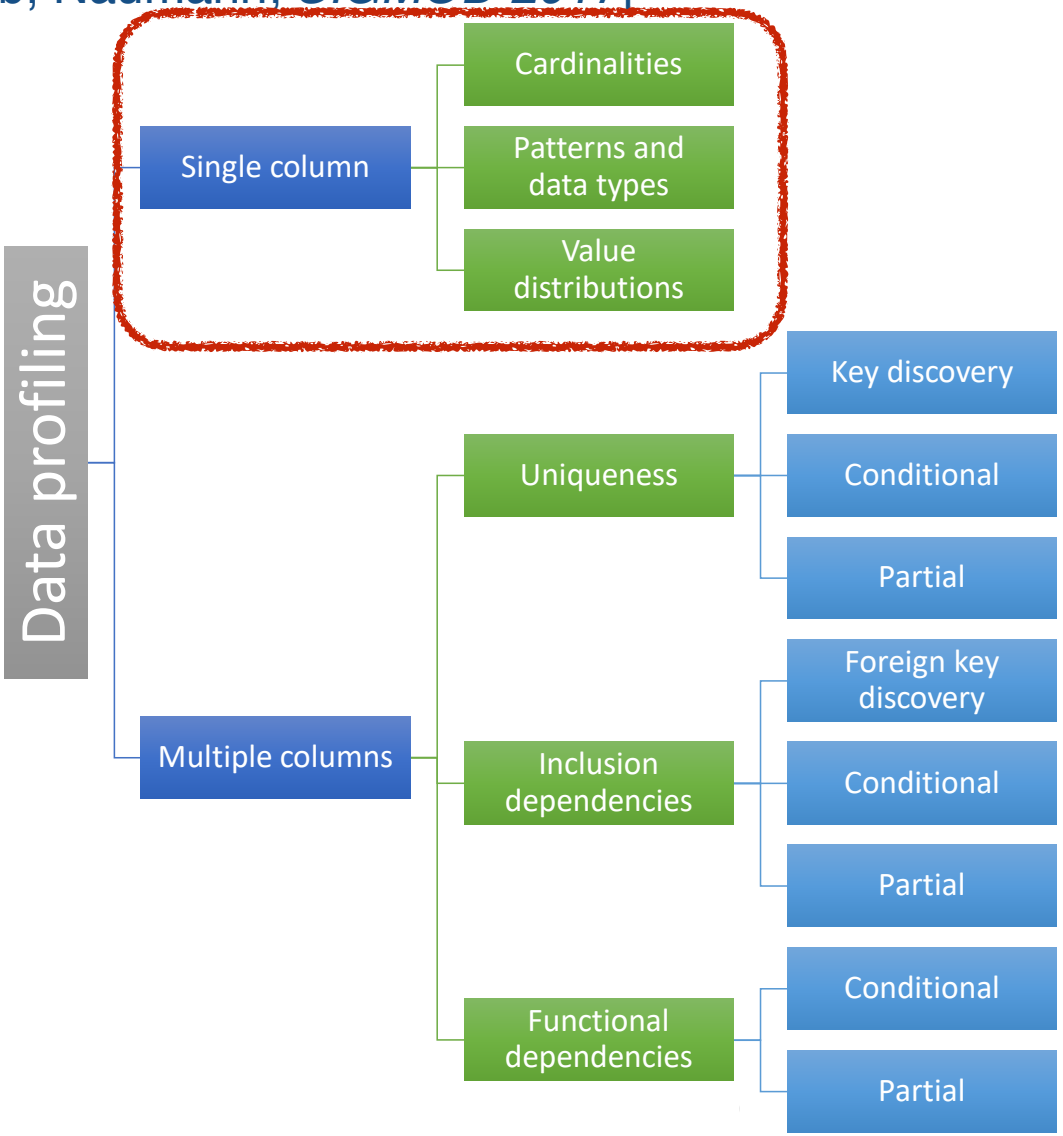
- To help understand the **statistics**, we look at value ranges, data types, value distributions per column or across columns, etc
- To help understand the **structure** - the (business) rules that generated the data - we look at unique columns / column combinations, dependencies between columns, etc - **reverse-engineer the relational schema** of the data we have
- We need both statistics and structure, they are mutually-reinforcing, and help us understand the **semantics** of the data - it's meaning

A classification of data profiling tasks

[Abedjan, Golab, Naumann; *SIGMOD 2017*]

	A	B	C	D	E	F	G	H
1	UID	sex	race	MarriageSta	DateOfBirth	age	juv_fel_cour	decile_score
2	1	0	1	1	4/18/47	69	0	1
3	2	0	2	1	1/22/82	34	0	3
4	3	0	2	1	5/14/91	24	0	4
5	4	0	2	1	1/21/93	23	0	8
6	5	0	1	2	1/22/73	43	0	1
7	6	0	1	3	8/22/71	44	0	1
8	7	0	3	1	7/23/74	41	0	6
9	8	0	1	2	2/25/73	43	0	4
10	9	0	3	1	6/10/94	21	0	3
11	10	0	3	1	6/1/88	27	0	4
12	11	1	3	2	8/22/78	37	0	1
13	12	0	2	1	12/2/74	41	0	4
14	13	1	3	1	6/14/68	47	0	1
15	14	0	2	1	3/25/85	31	0	3
16	15	0	4	4	1/25/79	37	0	1
17	16	0	2	1	6/22/90	25	0	10
18	17	0	3	1	12/24/84	31	0	5
19	18	0	3	1	1/8/85	31	0	3
20	19	0	2	3	6/28/51	64	0	6
21	20	0	2	1	11/29/94	21	0	9
22	21	0	3	1	8/6/88	27	0	2
23	22	1	3	1	3/22/95	21	0	4
24	23	0	4	1	1/23/92	24	0	4
25	24	0	3	3	1/10/73	43	0	1
26	25	0	1	1	8/24/83	32	0	3
27	26	0	2	1	2/8/89	27	0	3
28	27	1	3	1	9/3/79	36	0	3

relational data (here: just one table)



Single column: cardinalities, data types

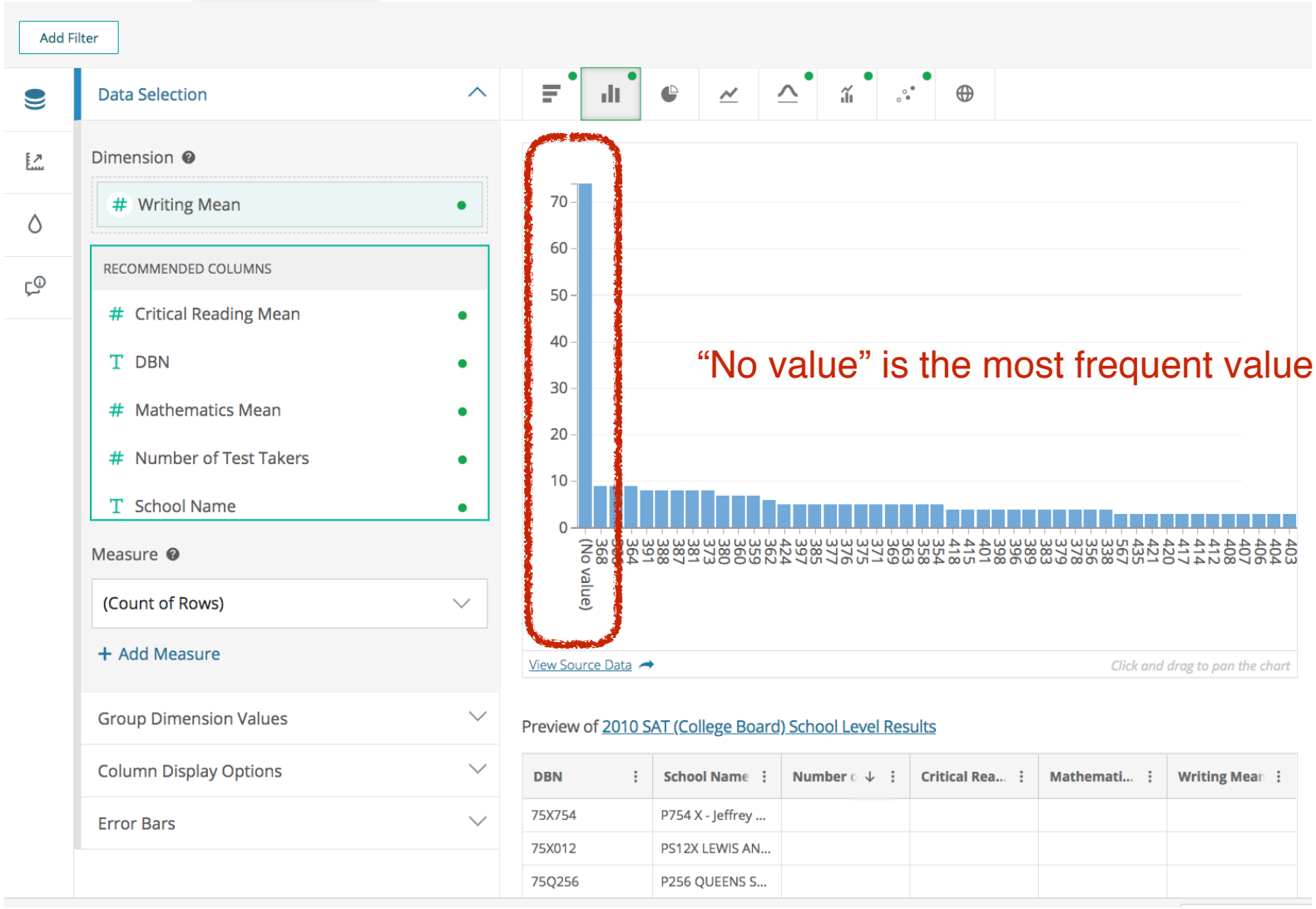
[Abedjan, Golab, Naumann; *SIGMOD 2017*]

- cardinality of relation **R** - number of rows
- domain cardinality of a column **R.a** - number of **distinct** values
- attribute value **length**: min, max, average, median
- **basic data type**: string, numeric, date, time, ….
- number of percentage of **null** values of a given attribute
- regular expressions
- semantic domain: SSN, phone number
- ….

2010 SAT (College Board) School Level Results

Education

NYC OpenData



The trouble with *null* values

A CRITIQUE OF

THE SQL DATABASE LANGUAGE

C.J.Date

PO Box 2647, Saratoga
California 95070, USA

December 1983

* Null values

I have argued against null values at length elsewhere [6], and I will not repeat those arguments here. In my opinion the null value concept is far more trouble than it is worth. Certainly it has never been properly thought through in the existing SQL implementations (see the discussion under "Lack of Orthogonality: Miscellaneous Items", earlier). For example, the fact that functions such as AVG simply ignore null values in their argument violates what should surely be a fundamental principle, viz: The system should never produce a (spuriously) precise answer to a query when the data involved in that query is itself imprecise. At least the system should offer the user the explicit option either to ignore nulls or to treat their presence as an exception.

50 shades of *null*

- **Unknown** - some value definitely belongs here, but I don't know what it is (e.g., unknown birthdate)
- **Inapplicable** - no value makes sense here (e.g., if marital status = single then spouse name should not have a value)
- **Unintentionally omitted** - values is left unspecified unintentionally, by mistake
- **Optional** - a value may legitimately be left unspecified (e.g., middle name)
- **Intentionally withheld** (e.g., an unlisted phone number)
- …..

(this selection is mine, see reference below for a slightly different list)

https://www.vertabelo.com/blog/technical-articles/50-shades-of-null-or-how-a-billion-dollar-mistake-has-been-stalking-a-whole-industry-for-decades

50 shades of *null*… and it gets worse!

- **Hidden missing values** -
  - 99999 for zip code, Alabama for state
  - need data cleaning….
- lots of houses in Philadelphia, PA were built in 1934

how do we detect hidden missing values?

Single column: cardinalities, data types

[Abedjan, Golab, Naumann; *SIGMOD 2017*]

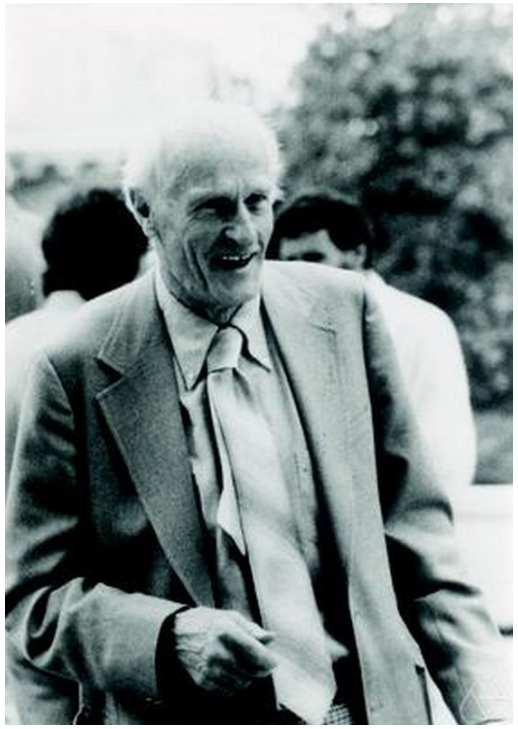
- cardinality of relation **R** - number of rows
- domain cardinality of a column **R.a** - number of **distinct** values
- attribute value **length**: min, max, average, median
- **basic data type**: string, numeric, date, time, ….
- number of percentage of **null** values of a given attribute
- **regular expressions**
- semantic domain: SSN, phone number
- ….

Regular expressions

- some attributes will have values that follow a regular format, e.g, telephone numbers: 212-864-0355 or (212) 864-0355 or 1.212.864-0355
- we may want to identify a small set of **regular expressions** that match all (or most) values in a column
- challenging - **very many possibilities!**

A **regular expression**, **regex** or **regexp** … is a sequence of characters that define a *search pattern*. Usually this pattern is used by string searching algorithms for "find" or "find and replace" operations on strings, or for input validation. It is a technique that developed in theoretical computer science and formal language theory.

https://en.wikipedia.org/wiki/Regular_expression



Stephen Kleene

Inferring regular expressions

- we may want to identify a small set of **regular expressions** that match all (or most) values in a column
- challenging - **very many possibilities!**

Example Regular Expression Language

.	Matches any character
abc	Sequence of characters
[abc]	Matches any of the characters inside **[ ]**
*****	Previous character matched zero or more times
?	Previous character matched zero or one time
{m}	Exactly **m** repetitions of previous character
^	Matches beginning of a line
$	Matches end of a line
\d	Matches any decimal digit
\s	Matches any whitespace character
\w	Matches any alphanumeric character

telephone
(201) 368-1000
(201)373-9599
(718) 206-1088
(718) 206-1121
(718) 206-1420
(718) 206-4420
(718) 206-4481
(718) 262-9072
(718) 868-2300
(718) 206-0545
(814) 681-6200
(888) 8NYC-TRS
800-624-4143

based on a slide by Heiko Mueller

Oakham's razor

Lex parsimoniae

If multiple hypotheses explain an observation, the simplest one should be preferred.

Ockham's motivation: can one prove the existence of God?

Used as a heuristic to help identify a promising hypothesis to test

Many applications today: biology, probability theory, ethics - also good for inferring regular expressions

William of Ockham
(1285-1347)

Inferring regular expressions

telephone
800-624-4143
(201)373-9599
(201) 368-1000
(718) 206-1088
(718) 206-1121
(718) 206-1420
(718) 206-4420
(718) 206-4481
(718) 262-9072
(718) 868-2300
(718) 206-0545
(814) 681-6200
(888) 8NYC-TRS

Simple Algorithm

(1) Group values by length

(2) Find pattern for each group

- Ignore small groups
- Find most specific character at each position

(	2	0	1	)		3	6	8	-	1	0	0	0
(	2	0	1	)		2	0	6	-	1	0	8	8
(	7	1	8	)		2	0	6	-	1	1	2	1
(	7	1	8	)		2	0	6	-	1	4	2	0
(	7	1	8	)		2	0	6	-	4	4	2	0
(	7	1	8	)		2	0	6	-	4	4	8	1
(	7	1	8	)		2	6	2	-	9	0	7	2
(	7	1	8	)		8	6	8	-	2	3	0	0
(	7	1	8	)		2	0	6	-	0	5	4	5
(	8	1	4	)		6	8	1	-	6	2	0	0
(	8	8	8	)		8	N	Y	C	-	T	R	S
(	\d	\d	\d	)		\d	\w	\w	.	.	\w	\w	\w

based on a slide by Heiko Mueller

Inferring regular expressions

telephone
800-624-4143
(201)373-9599
(201) 368-1000
(718) 206-1088
(718) 206-1121
(718) 206-1420
(718) 206-4420
(718) 206-4481
(718) 262-9072
(718) 868-2300
(718) 206-0545
(814) 681-6200
(888) 8NYC-TRS

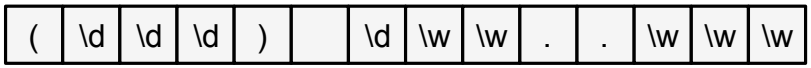
Simple Algorithm

(1) Group values by length

(2) Find pattern for each group

- **Ignore small groups**
- Find most specific character at each position

ignoring small groups: alternatives?




```
(\d{3}) \d\w{2}.{2}\w{3}
```

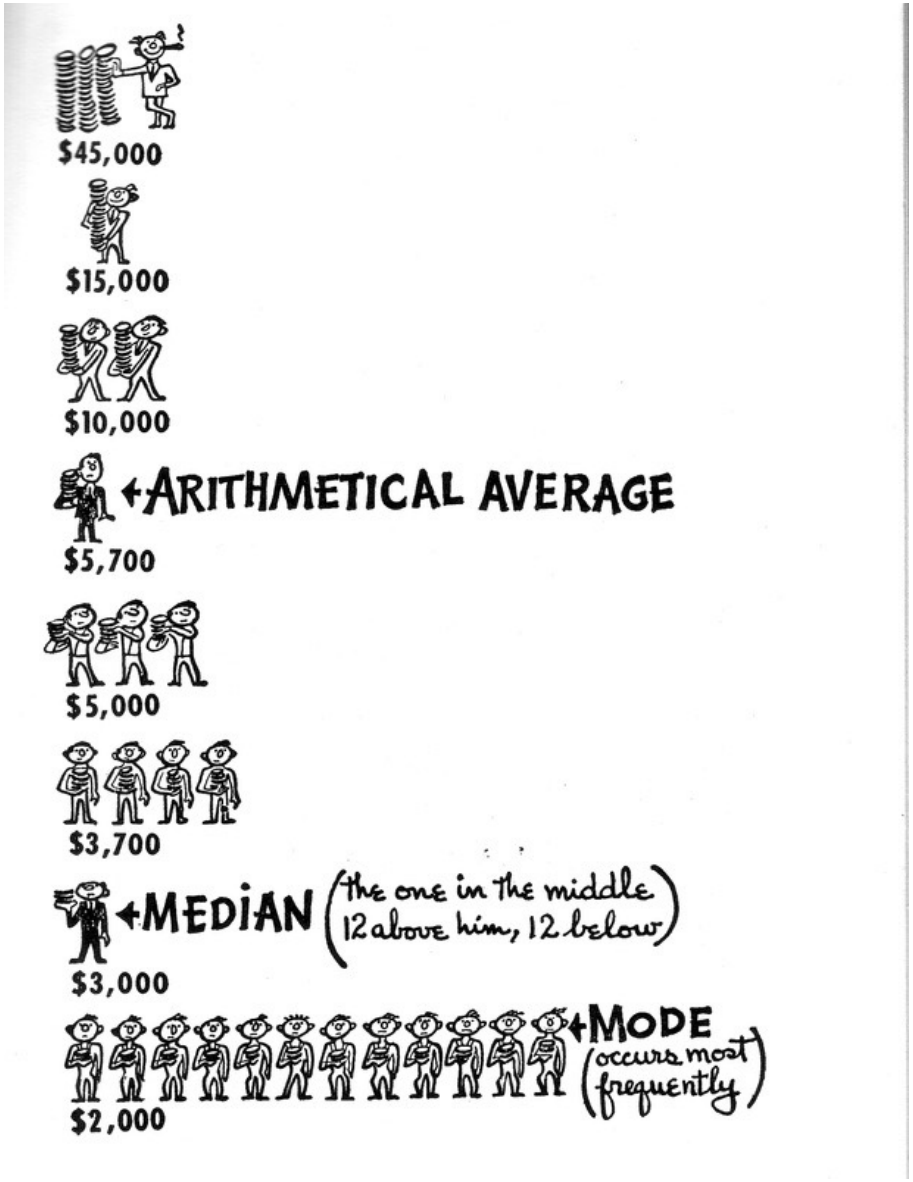
based on a slide by Heiko Mueller

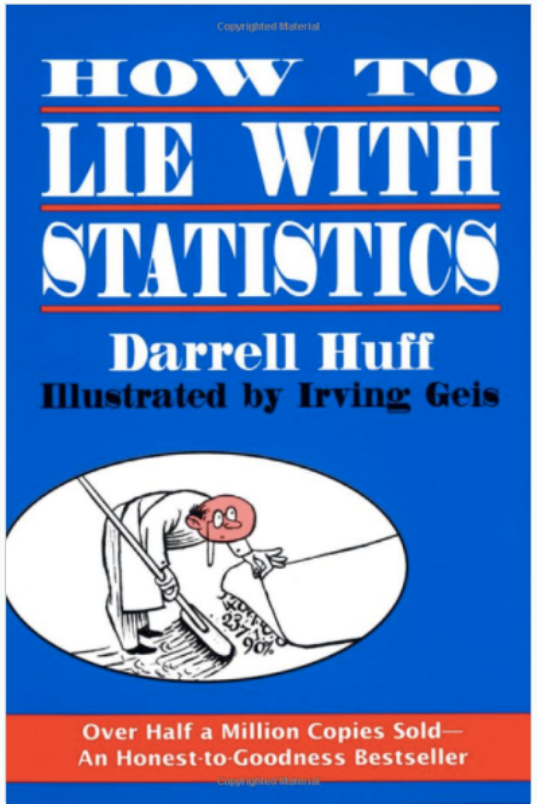
Single column: basic stats, distributions

[Abedjan, Golab, Naumann; *SIGMOD 2017*]

- min, max, **average**, median value of **R.a**
- **histogram**
  - equi-width - (approximately) the same number of distinct values in each bucket (e.g., age broken down into 5-year windows)
  - equi-depth (approximately) the same number of tuples in each bucket
  - biased histograms use different granularities for different parts of the value range to provide better accuracy
- quartiles - three points that divide the numeric values into four equal groups - a kind of an equi-depth histogram
- **first digit** - distribution of first digit in numeric values, to check Benford law
- …

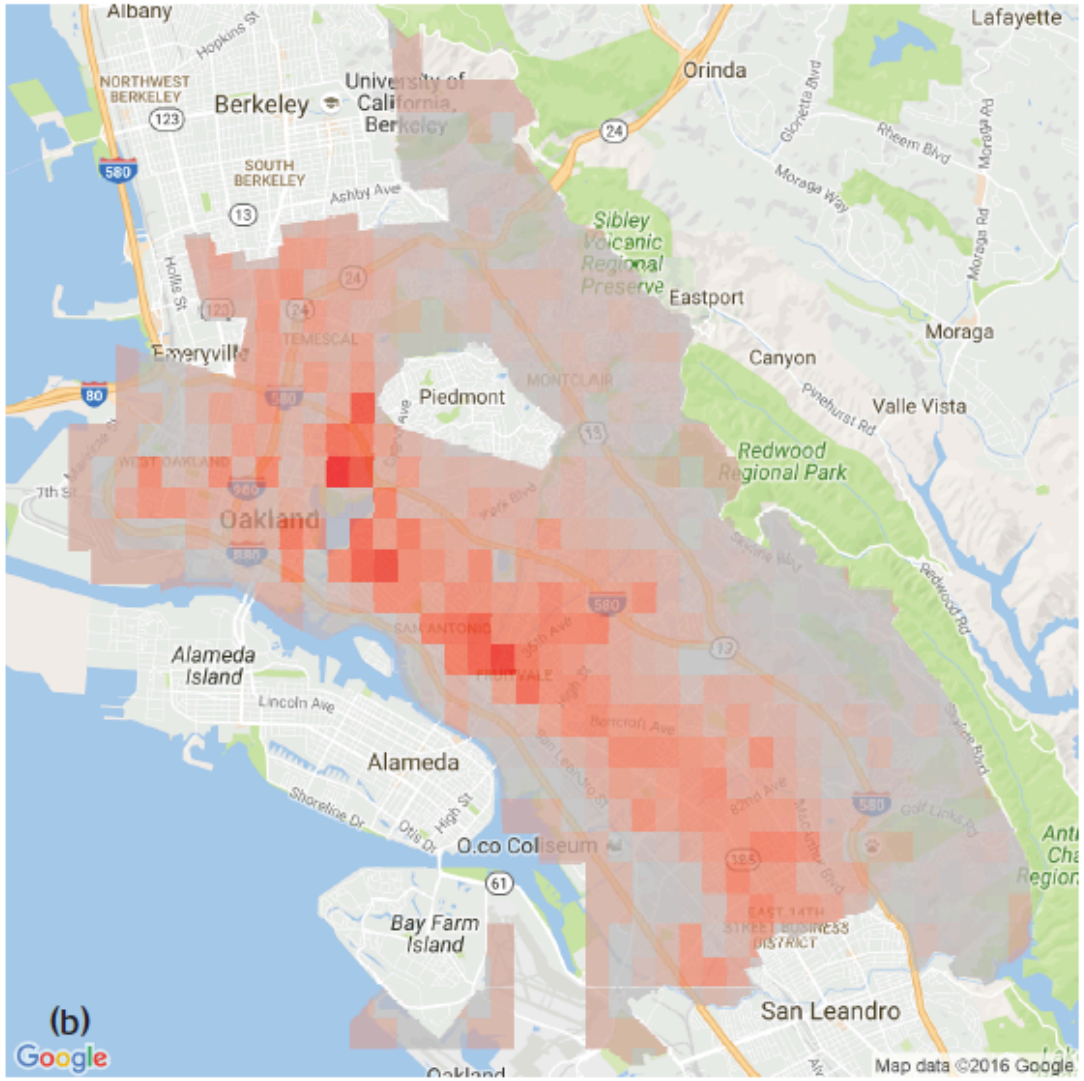
What's in an average?



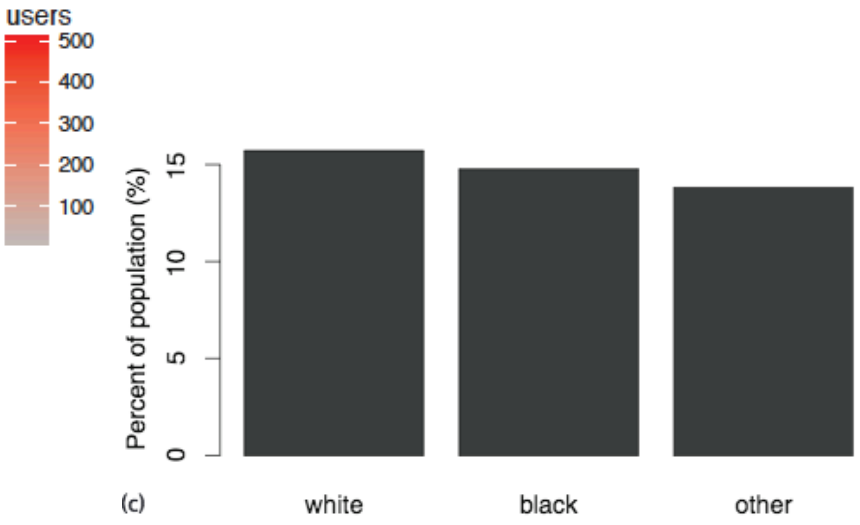


Is my data biased? (histograms + geo)

[Lum, Isaac; *Significance, 2016*]



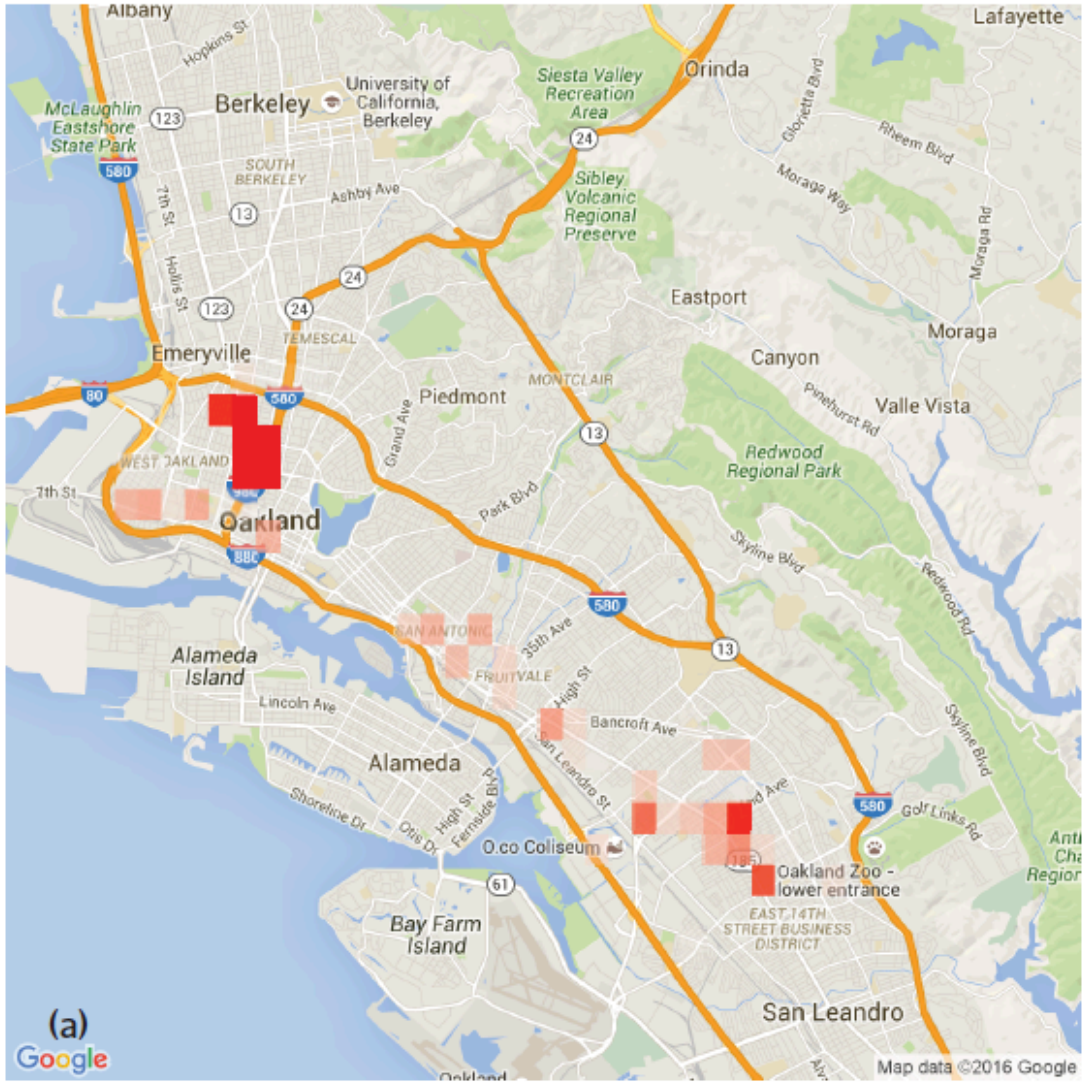
Estimated number of drug users, based on 2011 National Survey on Drug Use and Health, in Oakland, CA



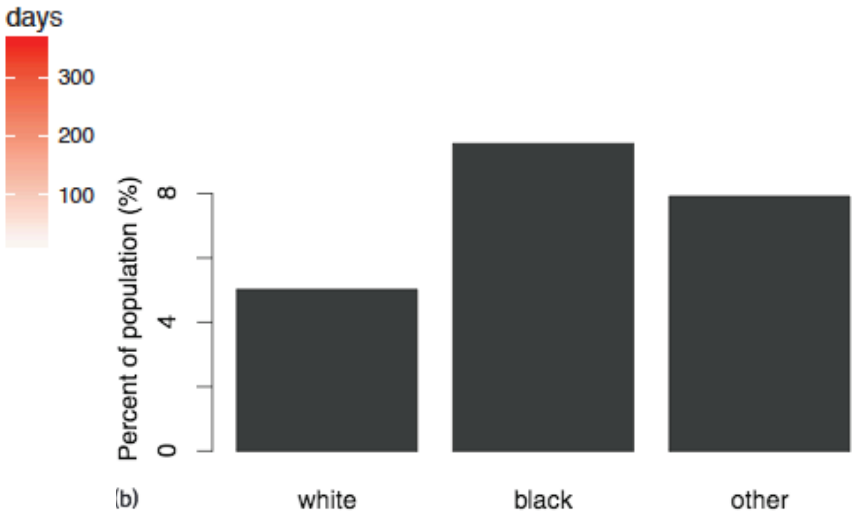
Estimated drug use by race

Is my data biased? (histograms + geo)

[Lum, Isaac; *Significance, 2016*]



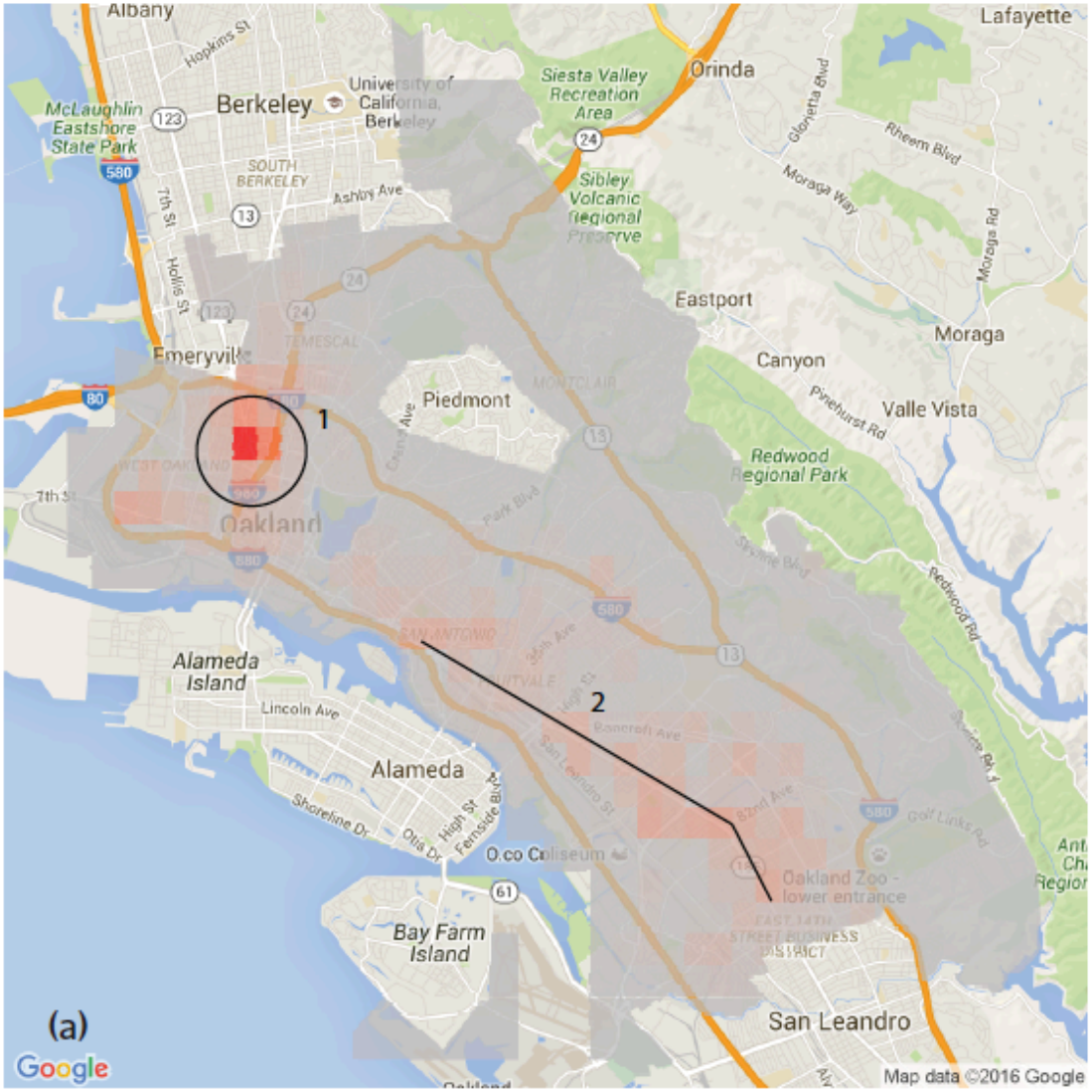
Number of days with targeted policing for drug crimes in areas flagged by PredPol analysis of Oakland, CA, police data for 2011



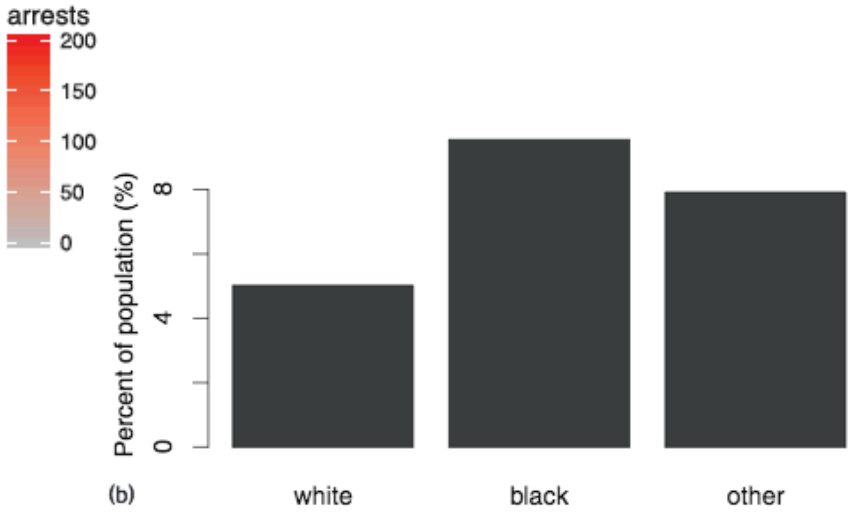
Targeted policing for drug crimes by race

Is my data biased? (histograms + geo)

[Lum. Isaac: *Significance. 2016*]



Number of drug arrests made by the Oakland, CA, police department in 2010



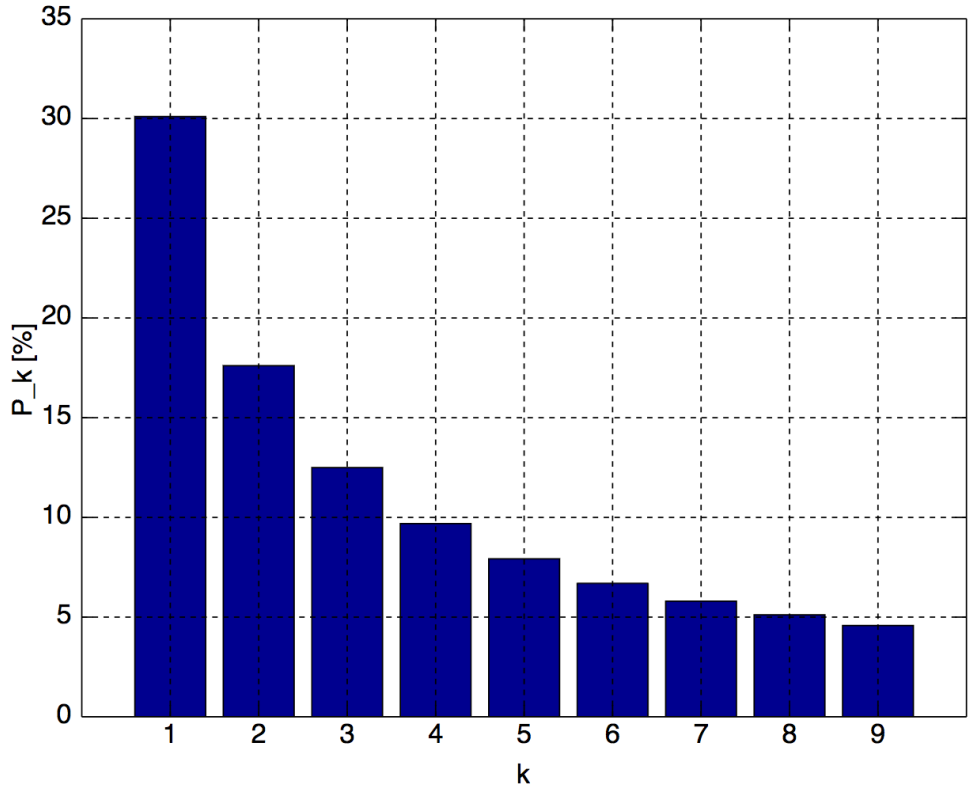
Targeted policing for drug crimes by race

Benford Law (first digit law)

[Benford: "The law of anomalous numbers" *Proc. Am. Philos. Soc.*, *1938*]

The distribution of the first digit **d** of a number, in many naturally occurring domains, approximately follows

$$P(d) = \log_{10}\left(1 + \frac{1}{d}\right)$$



1 is the most frequent leading digit, followed by 2, etc.

https://en.wikipedia.org/wiki/Benford%27s_law

Benford Law (first digit law)

[Benford: "The law of anomalous numbers" *Proc. Am. Philos. Soc.*, *1938*]

The distribution of the first digit **d** of a number, in many naturally occurring domains, approximately follows

$$P(d) = \log_{10}\left(1 + \frac{1}{d}\right)$$

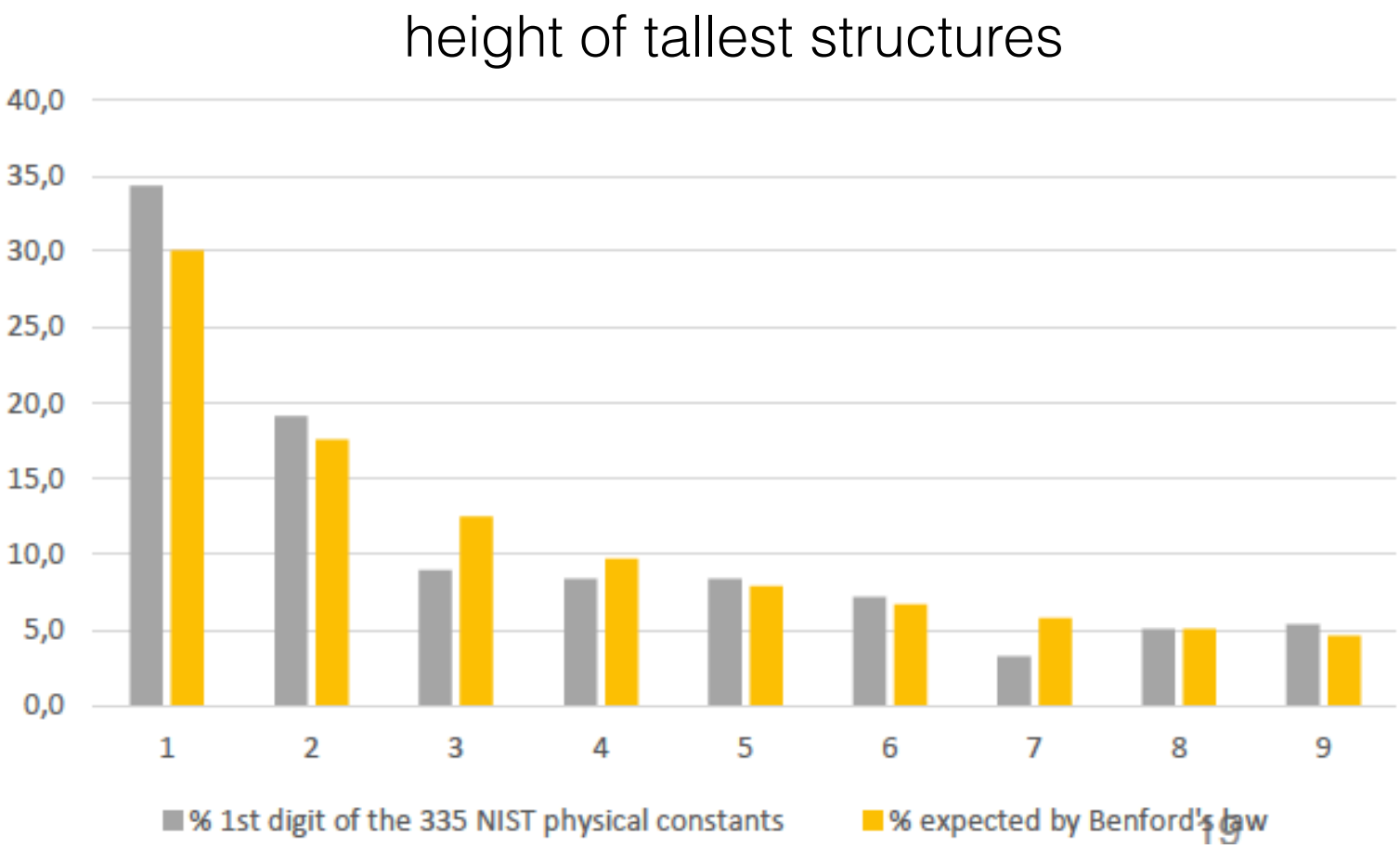
Holds if log(x) is uniformly distributed. Most accurate when values are distributed across multiple orders of magnitude, especially if the process generating the numbers is described by a power law (common in nature)

A logarithmic scale bar. Picking a random x position uniformly on this number line, roughly 30% of the time the first digit of the number will be 1.

https://en.wikipedia.org/wiki/Benford%27s_law

Benford Law: an example

[Abedjan, Golab, Naumann; *SIGMOD 2017*]



Benford Law: other examples

[Abedjan, Golab, Naumann; *SIGMOD 2017*]

- surface area of 355 rivers
- sizes of 3,259 US populations
- 104 physical constants
- 1,800 molecular weights
- 308 numbers contained in an issue of Reader's Digest
- Street addresses of the first 342 persons listed in American Men of Science
- ….

used in fraud detection!

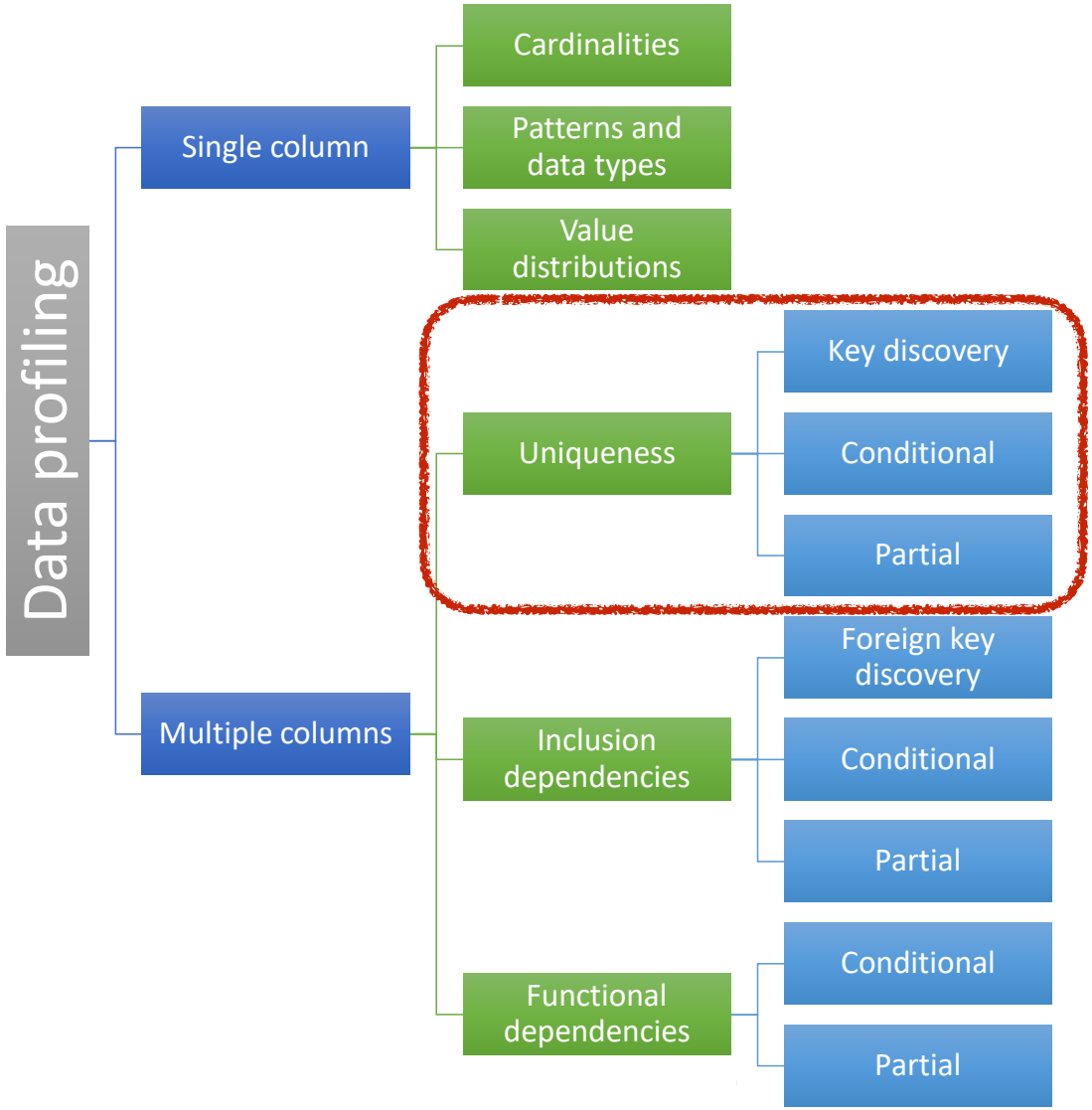
[Benford: "The law of anomalous numbers" *Proc. Am. Philos. Soc.*, *1938*]

Classification of data profiling tasks

[Abedjan, Golab, Naumann; *SIGMOD 2017*]

	A	B	C	D	E	F	G	H
1	UID	sex	race	MarriageSta	DateOfBirth	age	juv_fel_cour	decile_score
2	1	0	1	1	4/18/47	69	0	1
3	2	0	2	1	1/22/82	34	0	3
4	3	0	2	1	5/14/91	24	0	4
5	4	0	2	1	1/21/93	23	0	8
6	5	0	1	2	1/22/73	43	0	1
7	6	0	1	3	8/22/71	44	0	1
8	7	0	3	1	7/23/74	41	0	6
9	8	0	1	2	2/25/73	43	0	4
10	9	0	3	1	6/10/94	21	0	3
11	10	0	3	1	6/1/88	27	0	4
12	11	1	3	2	8/22/78	37	0	1
13	12	0	2	1	12/2/74	41	0	4
14	13	1	3	1	6/14/68	47	0	1
15	14	0	2	1	3/25/85	31	0	3
16	15	0	4	4	1/25/79	37	0	1
17	16	0	2	1	6/22/90	25	0	10
18	17	0	3	1	12/24/84	31	0	5
19	18	0	3	1	1/8/85	31	0	3
20	19	0	2	3	6/28/51	64	0	6
21	20	0	2	1	11/29/94	21	0	9
22	21	0	3	1	8/6/88	27	0	2
23	22	1	3	1	3/22/95	21	0	4
24	23	0	4	1	1/23/92	24	0	4
25	24	0	3	3	1/10/73	43	0	1
26	25	0	1	1	8/24/83	32	0	3
27	26	0	2	1	2/8/89	27	0	3
28	27	1	3	1	9/3/79	36	0	3

relational data (here: just one table)



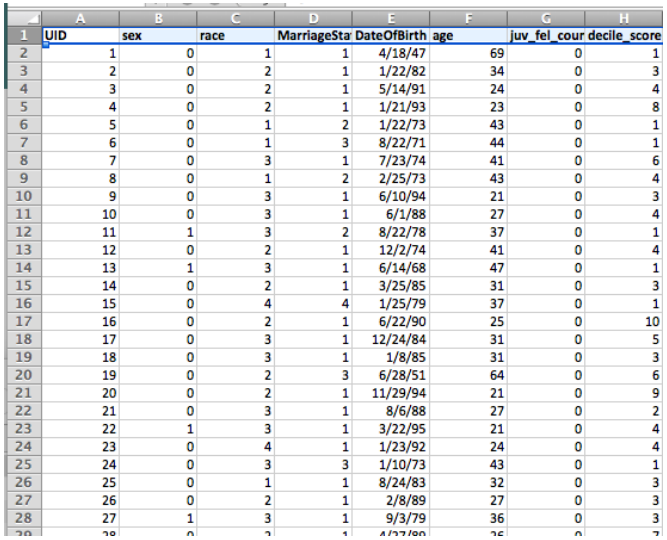
An alternative classification

- To help understand the **statistics**, we look at value ranges, data types, value distributions per column or across columns, etc
- To help understand the **structure** - the (business) rules that generated the data - we look at unique columns / column combinations, dependencies between columns, etc - **reverse-engineer the relational schema** of the data we have
- We need both statistics and structure, they are mutually-reinforcing, and help us understand the **semantics** of the data - it's meaning

next up: relational model basics

The relational model

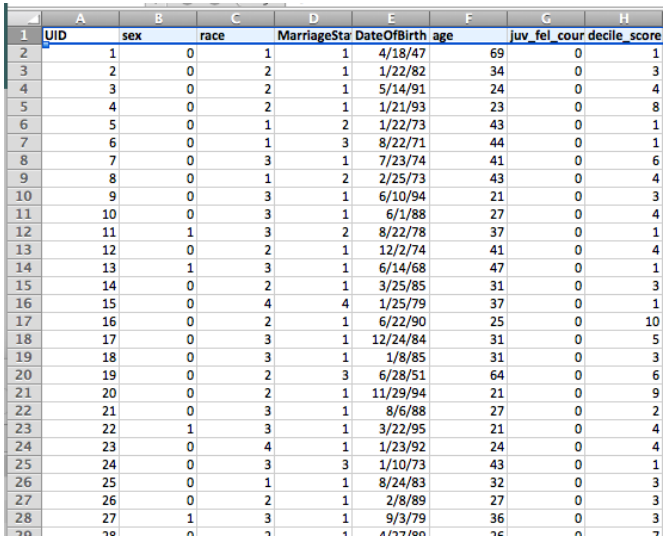
- Introduced by Edgar F. Codd in 1970 (Turing award)
- At the heart of relational database management systems (RDBMS)
  - a database consists of a collection of **relations** (tables)
  - **tuples** are stored in table rows
  - **attributes** of tuples are stored in table columns



	A	B	C	D	E	F	G	H
1	UID	sex	race	MarriageSta	DateOfBirth	age	juv_fel_cour	decile_score
2	1	0	1	1	4/18/47	69	0	1
3	2	0	2	1	1/22/82	34	0	3
4	3	0	2	1	5/14/91	24	0	4
5	4	0	2	1	1/21/93	23	0	8
6	5	0	1	2	1/22/73	43	0	1
7	6	0	1	3	8/22/71	44	0	1
8	7	0	3	1	7/23/74	41	0	6
9	8	0	1	2	2/25/73	43	0	4
10	9	0	3	1	6/10/94	21	0	3
11	10	0	3	1	6/1/88	27	0	4
12	11	1	3	2	8/22/78	37	0	1
13	12	0	2	1	12/2/74	41	0	4
14	13	1	3	1	6/14/68	47	0	1
15	14	0	2	1	3/25/85	31	0	3
16	15	0	4	4	1/25/79	37	0	1
17	16	0	2	1	6/22/90	25	0	10
18	17	0	3	1	12/24/84	31	0	5
19	18	0	3	1	1/8/85	31	0	3
20	19	0	2	3	6/28/51	64	0	6
21	20	0	2	1	11/29/94	21	0	9
22	21	0	3	1	8/6/88	27	0	2
23	22	1	3	1	3/22/95	21	0	4
24	23	0	4	1	1/23/92	24	0	4
25	24	0	3	3	1/10/73	43	0	1
26	25	0	1	1	8/24/83	32	0	3
27	26	0	2	1	2/8/89	27	0	3
28	27	1	3	1	9/3/79	36	0	3

The relational model

- Relations are **unordered collections** of tuples
  - conceptually, a relation is a **set** of tuples
  - however, SQL implements a relation as a **multiset** (bag) of tuples
- Why this model?
  - Simple yet powerful. Great for processing very large data sets in bulk



	A	B	C	D	E	F	G	H
1	UID	sex	race	MarriageSta	DateOfBirth	age	juv_fel_cour	decile_score
2	1	0	1	1	4/18/47	69	0	1
3	2	0	2	1	1/22/82	34	0	3
4	3	0	2	1	5/14/91	24	0	4
5	4	0	2	1	1/21/93	23	0	8
6	5	0	1	2	1/22/73	43	0	1
7	6	0	1	3	8/22/71	44	0	1
8	7	0	3	1	7/23/74	41	0	6
9	8	0	1	2	2/25/73	43	0	4
10	9	0	3	1	6/10/94	21	0	3
11	10	0	3	1	6/1/88	27	0	4
12	11	1	3	2	8/22/78	37	0	1
13	12	0	2	1	12/2/74	41	0	4
14	13	1	3	1	6/14/68	47	0	1
15	14	0	2	1	3/25/85	31	0	3
16	15	0	4	4	1/25/79	37	0	1
17	16	0	2	1	6/22/90	25	0	10
18	17	0	3	1	12/24/84	31	0	5
19	18	0	3	1	1/8/85	31	0	3
20	19	0	2	3	6/28/51	64	0	6
21	20	0	2	1	11/29/94	21	0	9
22	21	0	3	1	8/6/88	27	0	2
23	22	1	3	1	3/22/95	21	0	4
24	23	0	4	1	1/23/92	24	0	4
25	24	0	3	3	1/10/73	43	0	1
26	25	0	1	1	8/24/83	32	0	3
27	26	0	2	1	2/8/89	27	0	3
28	27	1	3	1	9/3/79	36	0	3

The relational model

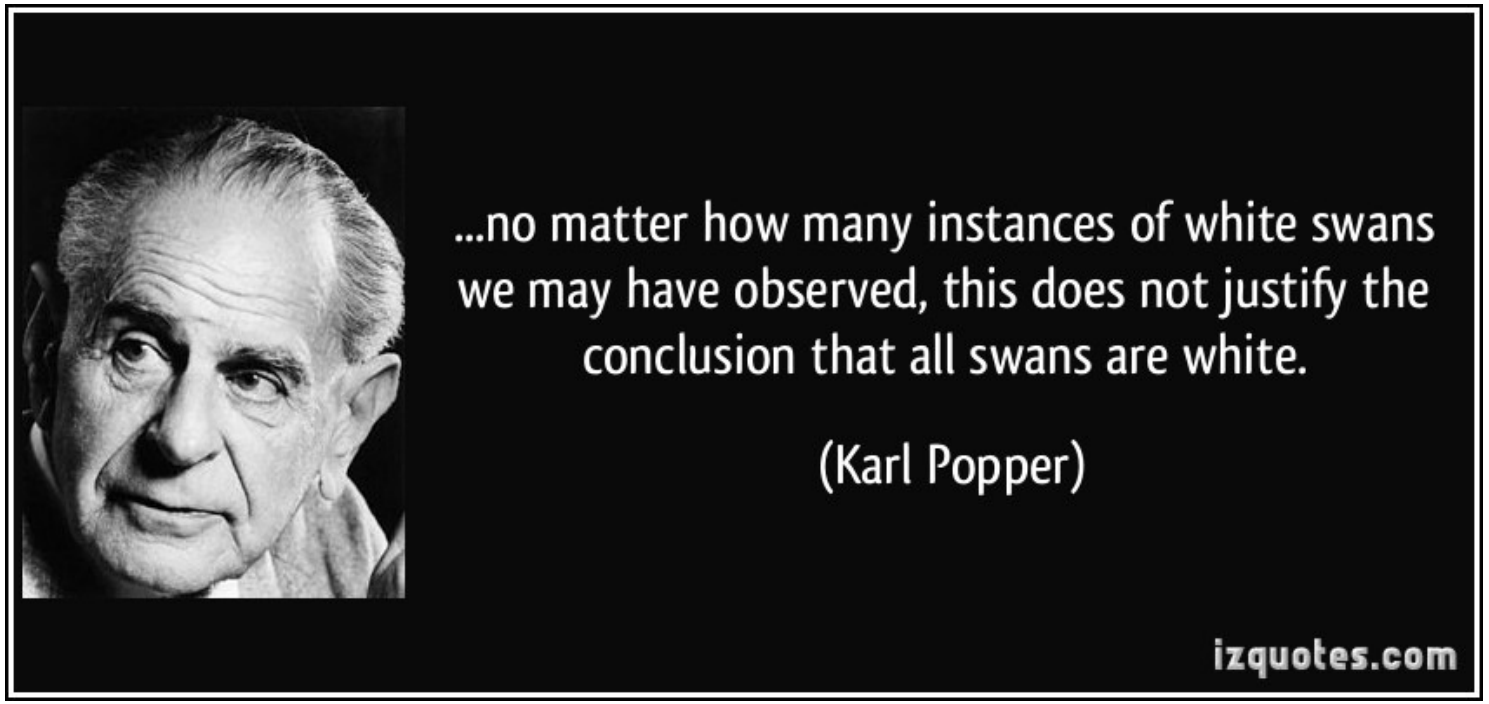
Episodes (season: int, num: int, title: string, viewers: long)

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

- **Relation**: a set or tuples - order doesn't matter, all tuples are distinct
- **Attribute**: a column in a relation (e.g., season)
- **Domain**: data type of an attribute (e.g., season: int)
- **Tuple**: a row in a relation, e.g., (1, 2, The Kingsroad, 2.2 M)

Schema vs. instances

Relation schema is a description of a relation in terms of relation name, attribute names, attribute datatypes, constraints (e.g., keys). A schema describes **all valid instances** of a relation.



Schema vs. instances

Relation schema is a description of a relation in terms of relation name, attribute names, attribute datatypes, constraints (e.g., keys). A schema describes **all valid instances** of a relation.

schema Episodes (season: integer, num: integer, title: string, viewers: integer)

instance 1

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M

instance 2

season	num	title	viewers
1	20	Blah, Blah and Blah	0
4	7	Yet Another Title	10 B

instance 3

season	num	title	viewers

Integrity constraints

- Ensure that data adheres to the rules of the application
- Specified **when schema is defined**
- Checked and enforced by the DBMS when relations are modified (tuples added / removed / updated)
- Must **hold on every valid instance** of the database

1. **Domain constraints** - specify valid data types for each attribute, e.g., Students (sid: integer, name: string, gpa: decimal)
2. **Key constraints** - define a unique identifier for each tuple
3. **Referential integrity constraints** - specify links between tuples
4. **Functional dependencies** - show relationships within a table

Key constraints

A set of attributes is a **candidate key** for a relation if:
(1) no two distinct tuples can have the same values for all key attributes (candidate key **uniquely identifies** a tuple), *and*
(2) this is not true for any subset of the key attributes (candidate key **is minimal**)

- If condition (2) is not met, we have a **superkey**
- There may be more than one candidate key for a relation, if so, one is designated as the **primary key**
- All candidate key should be known to property enforce data integrity

Example: name possible candidate keys

Students (sid: integer, login: string, name: string, dob: date)

Key constraints

Example: Students (sid: integer, login: string, name: string, dob: date)

three possible SQL implementations

```
create table Students (
   sid   integer      primary key,
   login varchar(128) unique,
   name  varchar(128),
   dob   date,
   gpa   decimal,
   unique (name, dob) );
```

```
create table Students (
   sid   integer      unique,
   login varchar(128) primary key,
   name  varchar(128),
   dob   date,
   gpa   decimal,
   unique (name, dob) );
```

```
create table Students (
   sid   integer       unique,
   login varchar(128)  unique,
   name  varchar(128),
   dob   date,
   gpa   decimal,
   primary key (name, dob) );
```

NB: every relation must have exactly one primary key

DB 101: Where do business rules come from?

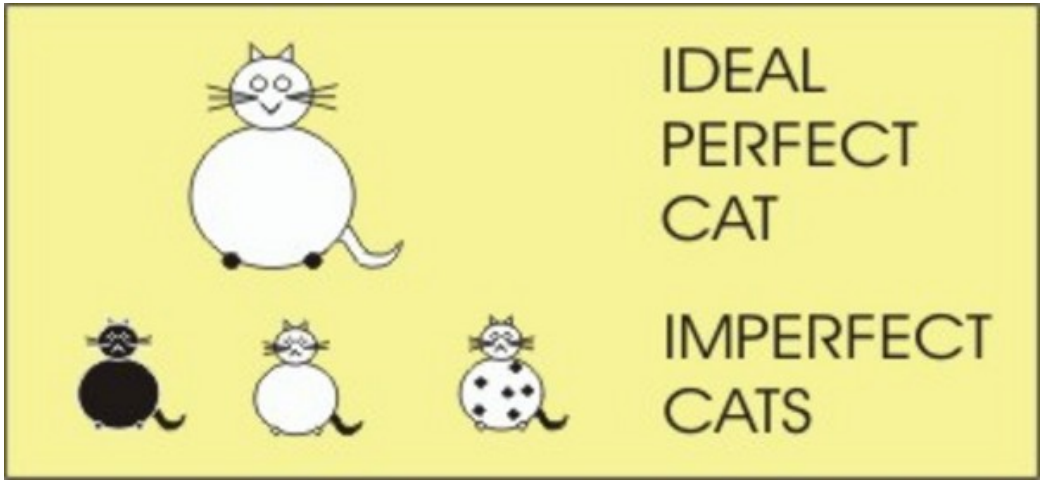
- **Business rules are given**: by the client, by the application designer, by your boss
- Once you know the rules, you create a **relational schema** that encodes these business rules (sometimes starting with the entity-relationship model, sometimes with the relational model directly)
- We can **never-ever-ever deduce business rules by looking at an instance** of a relation!
- We can sometimes know which rules do not hold, but we cannot be sure which rules do hold

Employee

id	login	name
1	jim	Jim Morrison
2	amy	Amy Winehouse
3	amy	Amy Pohler
4	raj	Raj Kapoor

1. Which column **is not** a candidate key?
2. Which column(s) **may be** a candidate key?
3. Give 2 create table statements for which this instance is valid.

DB (databases) vs. DS (data science)



https://midnightmediamusings.wordpress.com/2014/07/01/plato-and-the-theory-of-forms/

- **DB**: start with the schema, admit only data that fits; iterative refinement is possible, and common, but we are still schema-first
- **DS**: start with the data, figure out what schema it fits, or almost fits - reasons of usability, repurposing, low start-up cost

the “right” approach is somewhere between these two, data profiling aims to bridge between the two worlds / points of view / methodologies

Discovering uniques

Given a relation schema ***R*** *(A, B, C, D)* and a relation instance ***r***, a **unique column combination** (or a **"unique"** for short) is a set of attributes ***X*** whose **projection** contains no duplicates in ***r***

$Episodes(season, num, title, viewers)$

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M

Projection is a relational algebra operation that takes as input relation ***R*** and returns a new relation ***R'*** with a subset of the columns of ***R***.

$\pi_{season}(Episodes)$

season
1
1
2

$\pi_{season,num}(Episodes)$

season	num
1	1
1	2
2	1

$\pi_{title}(Episodes)$

title
Winter is Coming
The Kingsroad
The North Remembers

Discovering uniques

Given a relation schema ***R*** *(A, B, C, D)* and a relation instance ***r***, a **unique column combination** (or a **"unique"** for short) is a set of attributes ***X*** whose **projection** contains no duplicates in ***r***

$Episodes(season, num, title, viewers)$

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M

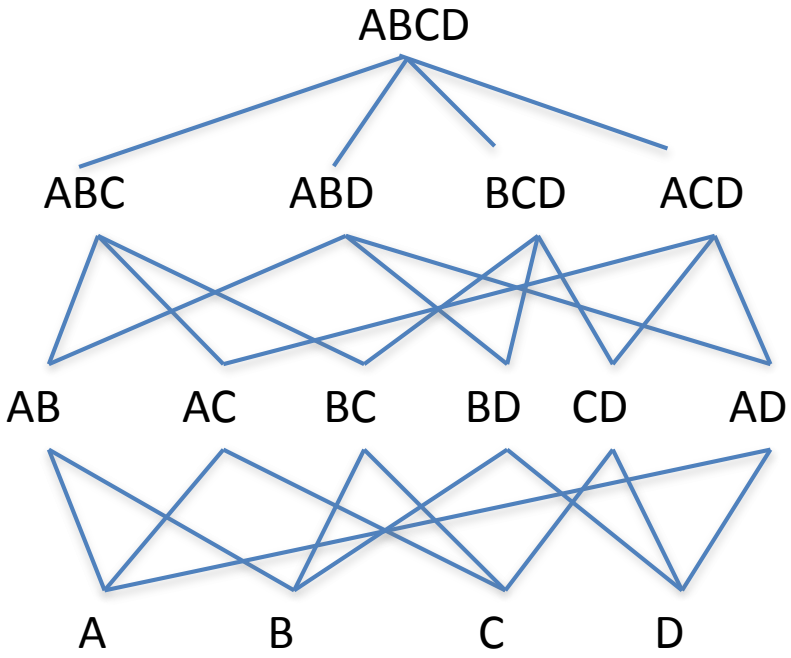
Projection is a relational algebra operation that takes as input relation ***R*** and returns a new relation ***R'*** with a subset of the columns of ***R***.

- Recall that more than one set of attributes **X** may be unique
- It may be the case that **X** and **Y** are both unique, and that they are not disjoint. When is this interesting?

Discovering uniques

R (A, B, C, D)

attribute lattice of ***R***



$$\binom{4}{4} = 1$$

$$\binom{4}{3} = 4$$

$$\binom{4}{2} = 6$$

$$\binom{4}{4} = 4$$

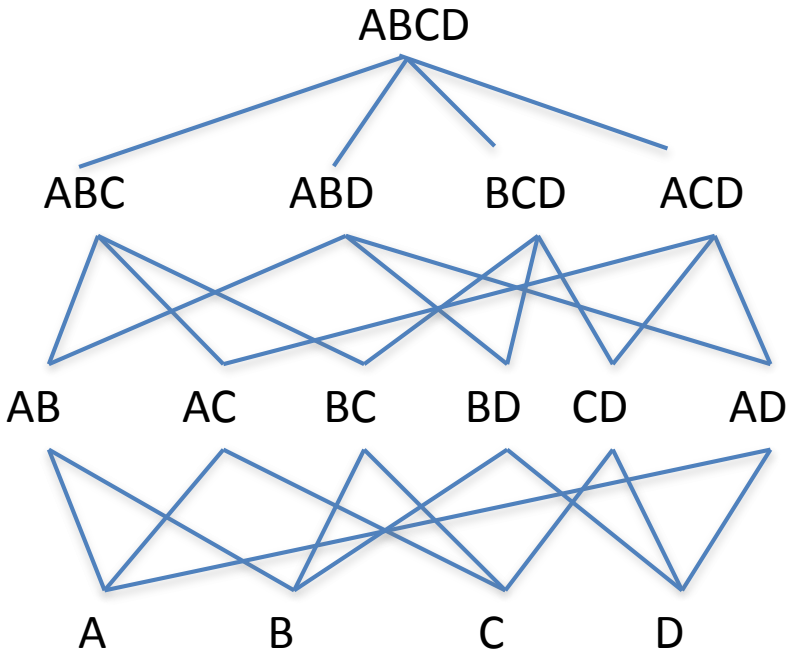
What's the size of the attribute lattice of ***R***?

Look at all attribute combinations?

Discovering uniques

R (A, B, C, D)

attribute lattice of R



- If **X** is unique, then what can we say about its **superset Y**?
- If **X** is non-unique, then what can we say about its **subset Z**?

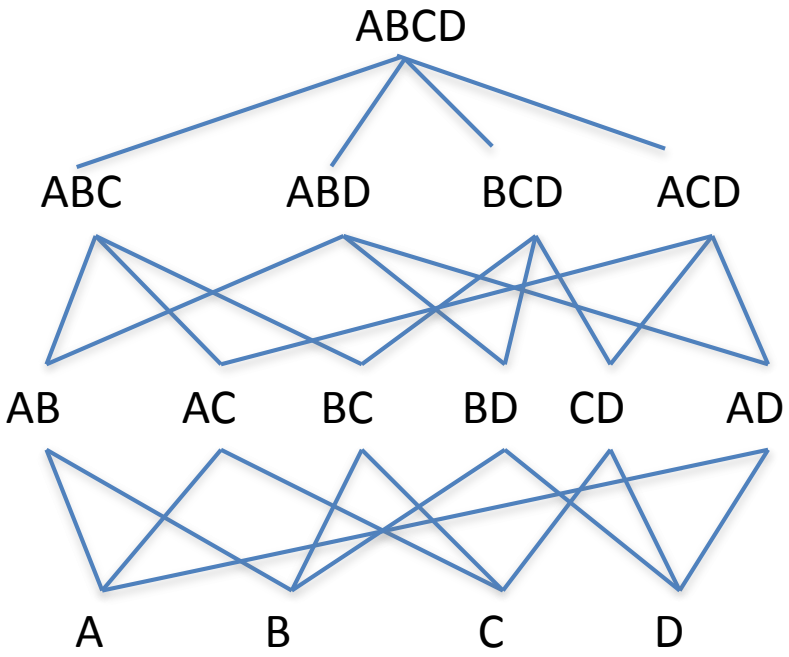
Discovering uniques

Given a relation schema ***R*** *(A, B, C, D)* and a relation instance ***r***, a **unique column combination** (or a **"unique"** for short) is a set of attributes ***X*** whose **projection** contains no duplicates in ***r***

Given a relation schema ***R*** *(A, B, C, D)* and a relation instance ***r***, a set of attributes ***Y*** is **non-unique** if its projection contains duplicates in ***r***

X is **minimal unique** if every subset ***Y*** of ***X*** is non-unique

Y is maximal non-unique if every superset ***X*** of ***Y*** is unique



From uniques to candidate keys

Given a relation schema ***R*** *(A, B, C, D)* and a relation instance ***r***, a **unique column combination** is a set of attributes ***X*** whose **projection** contains no duplicates in ***r***

$$Episodes(season, num, title, viewers)$$

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M

A set of attributes is a **candidate key** for a relation if:
(1) no two distinct tuples can have the same values for all key attributes (candidate key **uniquely identifies** a tuple), *and*
(2) this is not true for any subset of the key attributes (candidate key **is minimal**)

A minimal unique of a relation instance is a (possible) candidate key of the relation schema. To find all possible candidate keys, find all minimal uniques in a relation instance.

Pivot: Frequent itemsets & association rules

- Problem formulation due to Agrawal, Imielinski, Swami, SIGMOD 1993
- Solution: the **Apriori** algorithm by Agrawal & Srikant, VLDB 1994
- Initially for **market-basket data** analysis, has many other applications, we'll see one today
- We wish to answer two related questions:
  - **Frequent itemsets:** Which items are often purchased together, e.g., milk and cookies are often bought together
  - **Association rules:** Which items will likely be purchased, based on other purchased items, e.g., if diapers are bought in a transaction, beer is also likely bought in the same transaction

Market-basket data

- $I = \{i_1, i_2, \ldots, i_m\}$ is the set of available items, e.g., a product catalog of a store
- $X \subseteq I$ is an **itemset**, e.g., {milk, bread, cereal}
- **Transaction** t is a set of items purchased together, $t \subseteq I$, has a transaction id (TID)

  t_1: {bread, cheese, milk}

  t_2: {apple, eggs, salt, yogurt}

  t_3: {biscuit, cheese, eggs, milk}
- Database T is a set of transactions $\{t_1, t_2, \ldots, t_n\}$
- A transaction t **supports** an itemset X if $X \subseteq t$
- Itemsets supported by at least ***minSupp*** transactions are called **frequent itemsets**

minSupp, which can be a number or a percentage, is specified by the user

Itemsets

TID	Items
1	A
2	A C
3	A B D
4	A C
5	A B C
6	A B C

minSupp = 2 transactions

How many possible itemsets are there (excluding the empty itemset)?

$2^4 - 1 = 15$

itemset	support
A	6
B	3
C	4
D	1
A B	3
A C	4
A D	1
B C	2
B D	1
C D	0
A B C	2
A B D	1
B C D	0
A C D	0
A B C D	0

Association rules

An **association rule** is an implication $X \rightarrow Y$, where $X, Y \subset I$, *and* $X \cap Y = \varnothing$

example: {milk, bread}→{cereal}

"A customer who purchased X is also likely to have purchased Y in the same transaction"

we are interested in rules with a **single item** in Y

can we represent {milk, bread} → {cereal, cheese}?

Rule $X \rightarrow Y$ holds with **support** *supp* in T if *supp* of transactions contain $X \cup Y$

Rule $X \rightarrow Y$ holds with confidence *conf* in T if *conf* % of transactions that contain X also contain Y

$conf \approx \Pr(Y \mid X)$

$conf\ (X \rightarrow Y) = supp\ (X \cup Y)\ /\ supp\ (X)$

Association rules

minSupp = 2 transactions
minConf = 0.75

itemset	support
A	6
B	3
C	4
D	1
A B	3
A C	4
A D	1
B C	2
B D	1
C D	0
A B C	2
A B D	1
B C D	0
A C D	0
A B C D	0

supp = 3
$A \rightarrow B$ conf = 3 / 6 = 0.5
$B \rightarrow A$ conf = 3 / 3 = 1.0 ★

supp = 2
$B \rightarrow C$ conf = 2 / 3 = 0.67
$C \rightarrow B$ conf = 2 / 4 = 0.5

supp = 4
$A \rightarrow C$ conf = 4 / 6 = 0.67
$C \rightarrow A$ conf = 4 / 4 = 1.0 ★

supp = 2
$AB \rightarrow C$ conf = 2 / 3 = 0.67
$AC \rightarrow B$ conf = 2 / 4 = 0.5
$BC \rightarrow A$ conf = 2 / 2 = 1.0 ★

conf $(X \rightarrow Y)$ **=** ***supp*** **(X U Y) /** ***supp*** **(X)**

Association rule mining

- Goal: find all association rules that satisfy the user-specified minimum support and minimum confidence

- Algorithm outline
  - Step 1: find all frequent itemsets
  - Step 2: find association rules

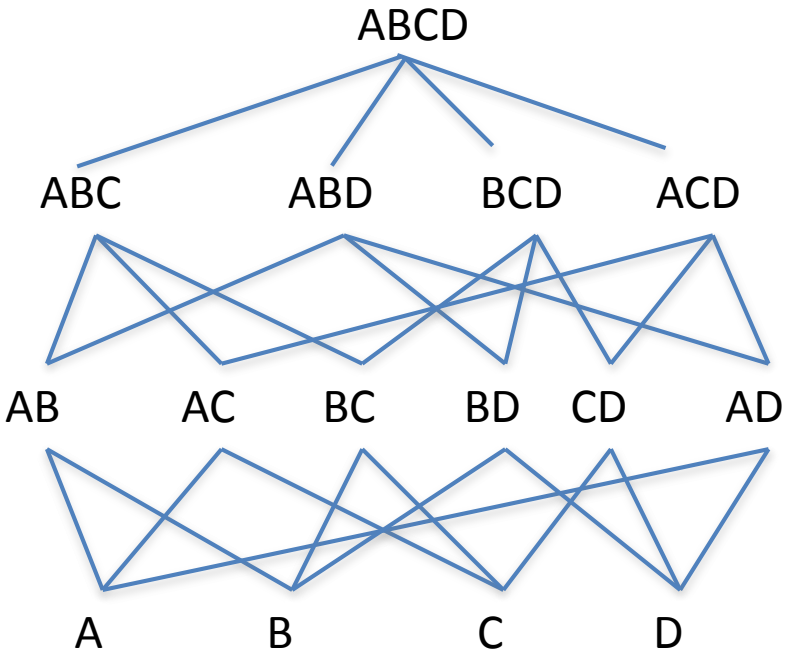
- Take 1: naïve algorithm for frequent itemset mining
  - Enumerate all subsets of ***I***, check their support in ***T***
  - **What is the complexity?**

Key idea: downward closure

itemset	support
A	6
B	3
C	4
D	1
A B	3
A C	4
A D	1
B C	2
B D	1
C D	0
A B C	2
A B D	1
B C D	0
A C D	0
A B C D	0

All subsets of a frequent itemset ***X*** are themselves frequent

So, if some subset of X is infrequent, then X cannot be frequent, we know this **apriori**



The converse is not true! If all subsets of ***X*** are frequent, ***X*** is not guaranteed to be frequent

The *Apriori* algorithm

Algorithm Apriori(*T*, *minSupp*)

F_1 = {*frequent 1-itemsets*};

for (k = 2; $F_{k-1} \neq \emptyset$; k++) **do**

$C_k \leftarrow$ **candidate-gen**(F_{k-1});

for each transaction $t \in T$ **do**

for each candidate $c \in C_k$ **do**

if c is contained in t **then**

c.count++;

end

end

$F_k \leftarrow \{c \in C_k \mid c.count \geq minSupp\}$

end

return $F \leftarrow \bigcup_k F_k$;

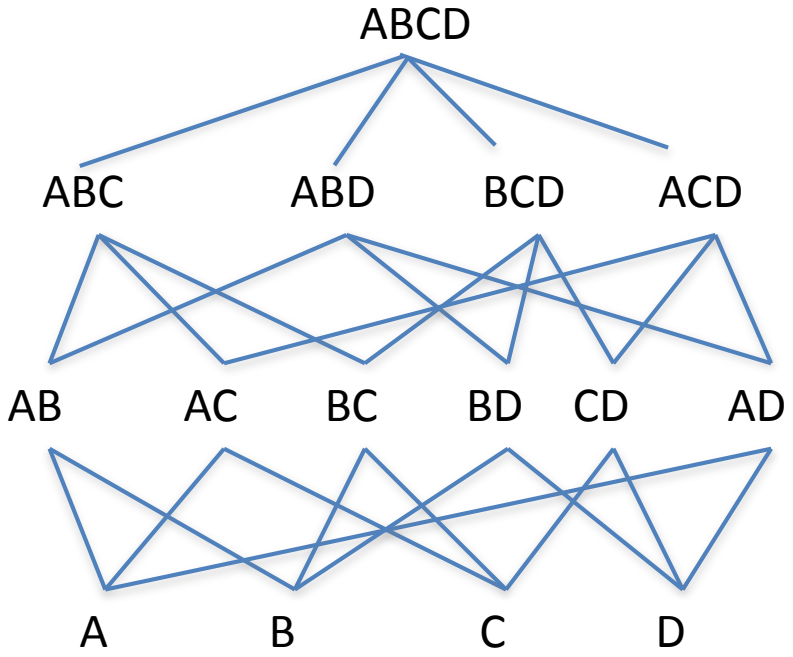
itemset	support
A	6
B	3
C	4
D	1
A B	3
A C	4
A D	1
B C	2
B D	1
C D	0
A B C	2
A B D	1
B C D	0
A C D	0
A B C D	0

Candidate generation

The **candidate-gen** function takes F_{k-1} and returns a superset (called the candidates) of the set of all frequent k-itemsets. It has two steps:

Join: generate all possible candidate itemsets C_k of length k

Prune: optionally remove those candidates in C_k that have infrequent subsets



Candidate generation

Assume a lexicographic ordering of the items

Join

```
Insert into C_k (
  select   p.item_1, p.item_2, …, p.item_{k-1}, q.item_{k-1}
  from     F_{k-1} p, F_{k-1} q
  where    p.item_1 = q.item_1
  and      p.item_2 = q.item_2
  and      …
  and      p.item_{k-1} < q.item_{k-1} )
```

why not $p.item_{k-1} \neq q.item_{k-1}$?

Prune

for each c in C_k **do**
  for each (k-1) subset s of c **do**
    if (s not in F_{k-1}) **then**
      delete c from C_k

Generating association rules

Rules = ∅

for each frequent *k-itemset* X **do**

  for each 1-itemset A ⊂ X **do**

    compute conf (X / A → A) = supp(X) / sup (X / A)

    if conf (X / A → A) ≥ minConf **then**

      Rules ← "X / A → A"

  end

end

end

return Rules

Performance of *Apriori*

- The possible number of frequent itemsets is exponential, O(2^m), where m is the number of items
- Apriori exploits sparseness and locality of data
  - Still, it may produce a large number of rules: thousands, tens of thousands, ….
  - So, thresholds should be set carefully. What are some good heuristics?

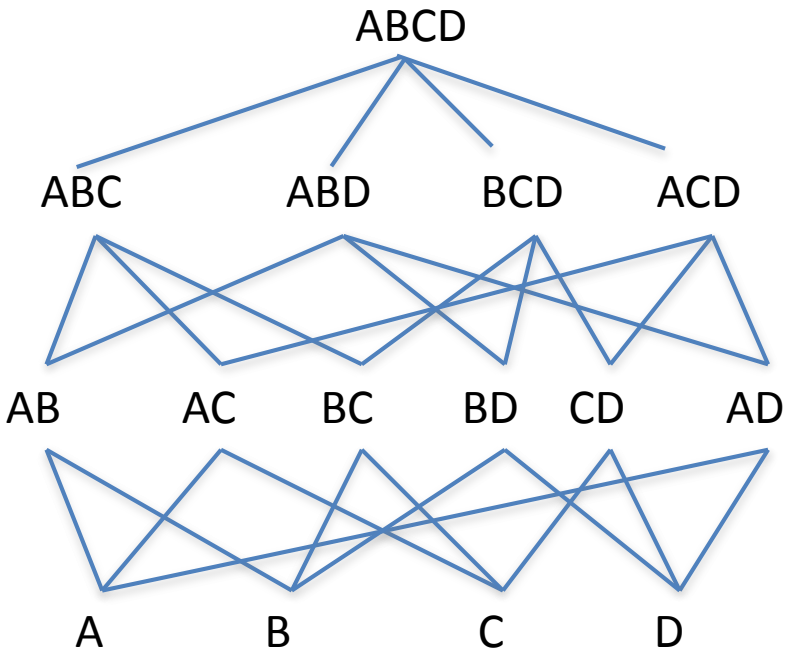
Back to data profiling: Discovering uniques

Given a relation schema ***R*** *(A, B, C, D)* and a relation instance ***r***, a **unique column combination** (or a **"unique"** for short) is a set of attributes ***X*** whose **projection** contains no duplicates in ***r***

Given a relation schema ***R*** *(A, B, C, D)* and a relation instance ***r***, a set of attributes ***Y*** is **non-unique** if its projection contains duplicates in ***r***

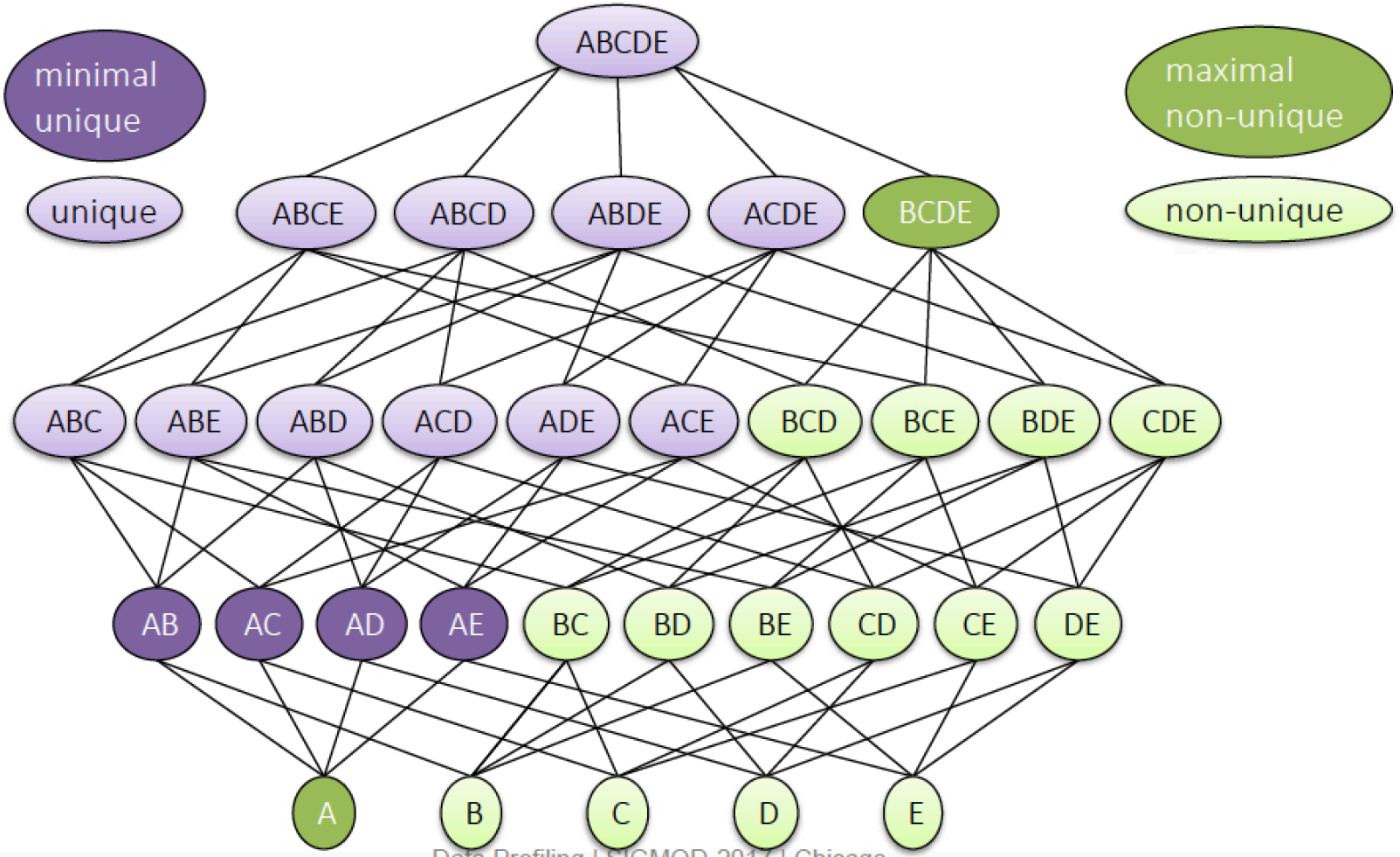
X is **minimal unique** if every subset ***Y*** of ***X*** is non-unique

Y is maximal non-unique if every superset ***X*** of ***Y*** is unique



[Abedjan, Golab, Naumann; *SIGMOD 2017*]

Output



From uniques to candidate keys

Given a relation schema ***R*** *(A, B, C, D)* and a relation instance ***r***, a **unique column combination** is a set of attributes ***X*** whose **projection** contains no duplicates in ***r***

$$Episodes(season, num, title, viewers)$$

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M

A set of attributes is a **candidate key** for a relation if:

(1) no two distinct tuples can have the value values for all key attributes (candidate key **uniquely identifies** a tuple), *and*

(2) this is not true for any subset of the key attributes (candidate key **is minimal**)

A minimal unique of a relation instance is a (possible) candidate key of the relation schema. To find such possible candidate keys, find all minimal uniques in a given relation instance.

Apriori-style uniques discovery

[Abedjan, Golab, Naumann; *SIGMOD 2017*]

A minimal unique of a relation instance is a **(possible) candidate key** of the relation schema.

Algorithm Uniques **// sketch, similar to HCA**

U_1 = {1-uniques} N_1 = {1-non-uniques}

for (k = 2; $N_{k-1} \neq \emptyset$; k++) **do**

$C_k \leftarrow$ **candidate-gen**(N_{k-1})

$U_k \leftarrow$ **prune-then-check** (C_k)

// prune candidates with unique sub-sets, and with **value distributions that cannot be unique**

// check each candidate in pruned set for uniqueness

$N_k \leftarrow C_k \setminus U_k$

end

return $U \leftarrow \bigcup_k U_k$;

Referential integrity constraints

A **foreign key** defines a directed link between a pair of tuples, R.t and S.t'. Foreign keys enforce **referential integrity**, a requirement that a value in an attribute or attributes of R.t must appear as a value in S.t'.

Example: Students (sid: integer) Courses (cid: integer) Enrolled(sid, cid)

Enrolled.sid must refer to an id of some actual student, per Students.sid

Enrolled.cid must refer to an id of some actual course, per Courses.cid

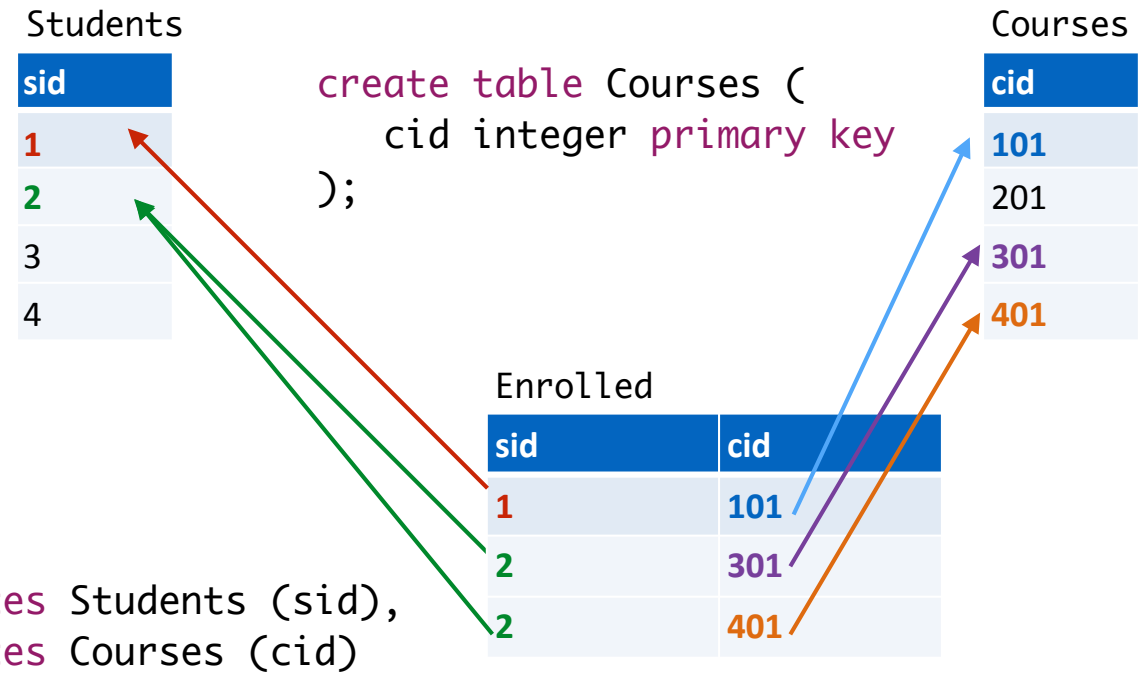
Referential integrity constraints

A **foreign key** defines a directed link between a pair of tuples, R.t and S.t'. Foreign keys enforce **referential integrity**, a requirement that a value in an attribute or attributes of R.t must appear as a value in S.t'.

```
create table Students (
    sid integer primary key
);
```

```
create table Courses (
    cid integer primary key
);
```

```
create table Enrolled (
    sid integer, cid integer,
    primary key (sid, cid),
    foreign key (sid) references Students (sid),
    foreign key (cid) references Courses (cid)
);
```



Referential integrity constraints

- **A foreign key** in a table **must reference** (point to) **a candidate key** in another table.
- That is, the target column(s) are either a primary key or are designated **unique**.
- Some relational systems limit this further: a foreign key must point to a primary key. For efficiency reasons, this is usually a better choice.

```
create table Students (
   sid   integer      primary key,
   login varchar(128) unique,
   name  varchar(128),
   dob   date,
   gpa   decimal,
   unique (name, dob)
);
```

```
create table Sibling_Pairs (
   thing_one   integer,
   thing_two   integer,
   primary key (thing_one, thing_two),
   foreign key (thing_one)
                 references Students(sid),
   foreign key (thing_two)
                 references Students(sid)
);
```

candidate keys: **sid**, login, (name, dob)

Discovering inclusion dependencies

- An inclusion dependency (IND) between attribute ***A*** of relation ***R*** and attribute ***B*** of relation ***S*** asserts that all values of ***R.A*** also appear in ***S.B***:
  R.A $\subseteq$ ***S.B***
  - We can extend this notion of unary (single-attribute) IND to sets of attributes ***R.X*** and ***S.Y***
  - Note that ***R*** and ***S*** are usually of two different relations (e.g,. Students and Enrolled)
- Inclusion dependencies must necessarily hold (but are not sufficient) for a foreign key (aka referential integrity) constraint to hold

Why is inclusion not a sufficient condition for referential integrity?

How might we go about discovering inclusion dependencies?

An inverted index for unary IND

[De Marchi, Lopes, Petit; *J Intell. Inf. Syst. 2009*]

A	B
1	3
1	4
2	3
1	5

relation ***R***

C	D
1	3
2	3
4	4
7	4

relation ***S***

Value	Columns
1	A, C
2	A, C
3	B, D
4	B, D
5	B
7	C

inverted index on values

How do we use this index data structure to find all pairs of attributes over which inclusion dependencies hold?

Across two relations, within a relation?

Which IND rules should we try?

- If our goal is to find candidates for foreign key, then the RHS of the rule should be unique (a candidate key)
  - We can extend this notion of unary (single-attribute) IND to sets of attributes ***R.X*** and ***S.Y***
  - Note that ***R*** and ***S*** are usually of two different relations (e.g,. Students and Enrolled)
- Inclusion dependencies must necessarily hold (but are not sufficient) for a foreign key (aka referential integrity) constraint to hold

Why is inclusion not a sufficient condition for referential integrity?

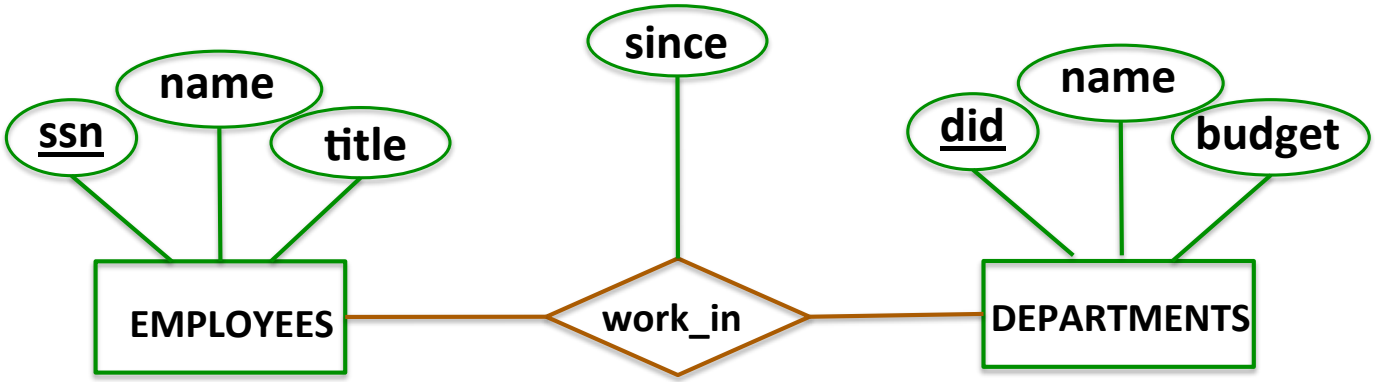
How might we go about discovering inclusion dependencies?

Functional dependencies

- Let's revisit the relational model: the basic abstraction is a relation (a table)
- In a properly designed database, a relation corresponds to a single real-world entity (e.g., student or course) or to a single relationship between entities (e.g., student takes course)
- Sometimes, we represent multiple linked entities, or an entity and some of its relationships, in the same table, but this is only done to encode specific business rules correctly
- Unless there are strong performance reasons to do so, we don't want to encode many-to-many relationships, together with the entities they relate, in the same table, because this gives rise to **redundancy**
- No avoid redundancy, we **decompose** (**normalize**) our relations - break them up into multiple relations
- Functional dependencies guide us in such decompositions

Example of a many-to-many relationship

each entity set (Employees and Departments) and the relationship set (work_in) are mapped to a table of their own



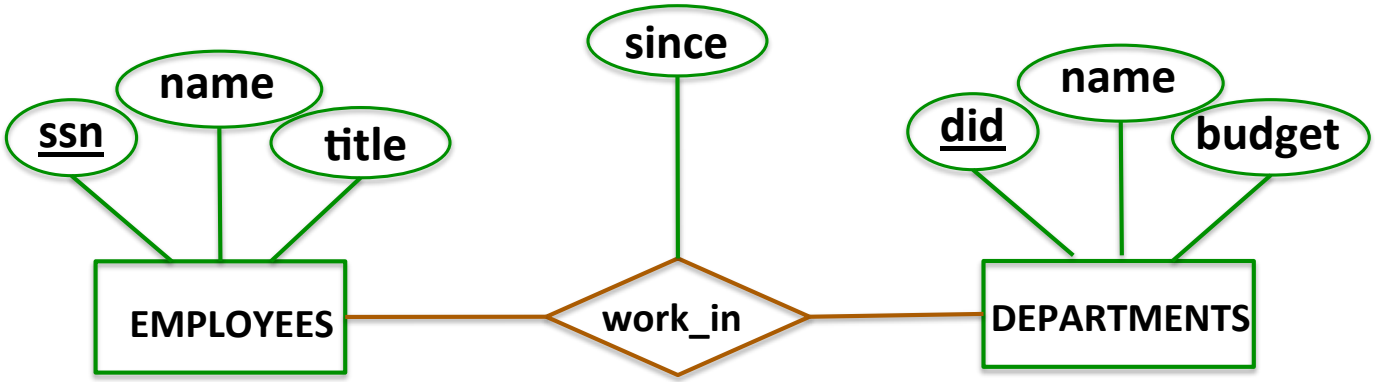

```
create table Employees (
   ssn      char(11)      primary key,
   name     varchar(128) not null,
   title    varchar(128)
);
```

```
create table Departments (
   did      integer      primary key,
   name     varchar(128) not null,
   budget   integer
);
```

```
create table Work_In (
   ssn     char(11),
   did     integer,
   since   date,
   primary key (ssn, did),
   foreign key (ssn) references Employee(ssn),
   foreign key (did) references Department(did)
);
```

This is incorrect!

each entity set (Employees and Departments) and the relationship set (work_in) are mapped to a table of their own




```
create table Employees_Work_In_Departments (
    ssn          char(11),
    empName      varchar(128) not null,
    title        varchar(128),
    did          integer,
    deptName     varchar(128) not null,
    budget       integer,
    since        date,
    primary key (ssn, did)
);
```

Why is this incorrect?
Because of redundancy!

$$ssn \rightarrow empName$$

$$ssn \rightarrow title$$

$$did \rightarrow deptName$$

$$did \rightarrow budget$$

Functional dependencies (FDs)

A **functional dependency** (FD) is an integrity constraint (IC) that generalizes the concept of a key. It's part of a schema or relation ***R***.

The functional dependency $A_1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m$ says:

When two tuples have the same values for all $A = \{A_1, A_2, \ldots, A_n\}$

then they will also have the same values for all $B = \{B_1, B_2, \ldots, B_m\}$

A **candidate key** is a particular kind of an FD: the values of the key attributes, taken together, uniquely identify a tuple in relation R, and so functionally determine all of that tuple's attributes.

Functional dependencies: example

K	A	B	C	D
k1	a1	b1	c1	d1
k2	a1	b1	c1	d2
k3	a1	b2	c2	d1
k4	a1	b3	c3	d1

we know that **K** is a candidate key

Is this FD satisfied: $K \rightarrow ABCD$

yes, **K** is a key

Is this FD satisfied: $AB \rightarrow C$

yes

Is this FD satisfied: $A \rightarrow B$

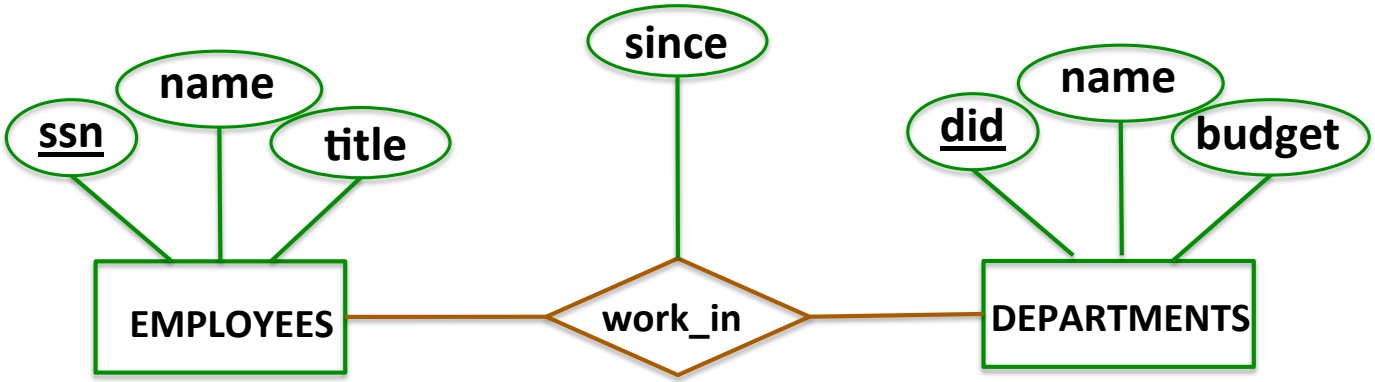
no, violated for tuples k3 and k4

Is this FD satisfied: $AB \rightarrow B$

yes, known as a **trivial** FD

FDs are used for normalization

each entity set (Employees and Departments) and the relationship set (work_in) are mapped to a table of their own




```
create table Employees_Work_In_Departments (
    ssn           char(11),
    emp_name      varchar(128) not null,
    title         varchar(128),
    did           integer,
    dept_name     varchar(128) not null,
    budget        integer,
    since         date,
    primary key (ssn, did)
);
```

$$ssn \rightarrow empName$$

$$ssn \rightarrow title$$

$$did \rightarrow deptName$$

$$did \rightarrow budget$$

if we knew these FDs, we'd decompose this relation to avoid redundancy

FDs and redundancy

$$A_1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m$$

A **candidate key** is a particular kind of an FD: the values of the key attributes, taken together, uniquely identify a tuple in relation ***R***, and so functionally determine all of that tuple's attributes.

- In an ideal world (one with no redundancy), we only have FDs in which a candidate key is on the left-hand-side, and no other FDs. This is known as the **Boyce-Codd Normal Form** (**BCNF**).

- In a somewhat less ideal world, where we know the schema (including the FDs), we can attempt to decompose our relation to have no redundancy (BCNF) or with almost no redundancy (**third normal form, 3NF**).

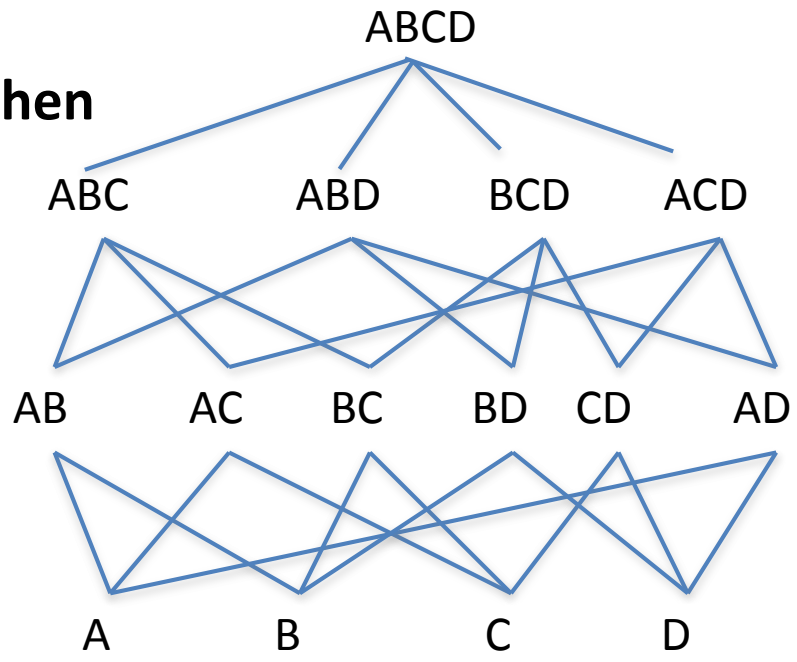
- If only a dataset (an instance or R) is given to us, but not the schema, and no FDs, we can attempt to **infer FDs** that hold, or almost hold, and to then either normalize our data, or to clean it accordingly.

Inferring FDs

Given relation R, detect all **minimal non-trivial FDs** of the form $X \rightarrow A$ where X is a set of attributes and A is a single attribute

```
FDs = ∅
for each A ∈ R do
    for each X ⊂ Powerset( R \ A) do
        if count distinct(X) = count district (XA) then
            FDs ← "X → A"
        end
    end
end
```

ring any bells?



Inferring FDs: TANE

[Huhtala, Kärkkäinen, Porkka, Toivonen; *Computer Journal 1999*]

An Aprior-style algorithm that traverses bottom-up through the attribute lattice

Consider only minimal dependencies: if B → C, don't check BD → C

Consider only non-trivial dependencies: for a set X, test all X \ A → A, A ∈ X