Intelligent pattern profiling

Trifacta (@trifacta)
Karthik Sethuraman
Sean Kandel

How is your data structured?

How is your data structured?

Is all your data structured identically?

How is your data structured?

Is all your data structured identically?

How to map input structure(s) to desired output?

How is your data structured?

Is all your data structured identically?

How to map input structure(s) to desired output?

IP	DATE	TZ	METHOD	PATH
2,451 Categories	Oct 25 - Nov 14	1 Category	1 Category	103 Categories
668.667.44.3	25/Oct/2011:07:38:30	-0500	GET	/download/downloa
13.386.648.380	25/Oct/2011:17:06:00	-0500	GET	/download/downloa
06.670.03.40	26/Oct/2011:13:24:00	-0500	GET	/product/demos/pr
18.656.618.46	26/Oct/2011:17:15:30	-0500	GET	/download/downloa
14.688.663.667	26/Oct/2011:21:02:30	-0500	GET	/news
13.07.338.684	26/Oct/2011:21:02:30	-0500	GET	/download
14.688.663.667	26/Oct/2011:21:02:30	-0500	GET	/news
688.615.03.332	26/Oct/2011:21:02:30	-0500	GET	/product/product1
688.615.03.332	26/Oct/2011:21:02:32	-0500	GET	/product/product1
688.615.03.332	26/Oct/2011:21:02:34	-0500	GET	/products/demos
13.07.338.684	26/Oct/2011:21:02:37	-0500	GET	/download
55.3.658.53	26/Oct/2011:21:06:30	-0500	GET	/buy
55.3.658.53	26/Oct/2011:21:06:56	-0500	GET	/buy
14.323.74.653	26/Oct/2011:21:07:00	-0500	GET	/demo
14.323.74.653	26/Oct/2011:21:08:00	-0500	GET	/demo
14.323.74.653	26/Oct/2011:21:09:00	-0500	GET	/demo
14.323.74.653	26/Oct/2011:21:10:03	-0500	GET	/demo

column1

6 Keys

{"IP":"668.667.44.3","DATE":"25/Oct/2011:07:38:30","TZ":"-0500","METHOD":"GET","PATH":"/download/download3.zip","St
{"IP":"13.386.648.380","DATE":"25/Oct/2011:17:06:00","TZ":"-0500","METHOD":"GET","PATH":"/download/download6.zip","
{"IP":"06.670.03.40","DATE":"26/Oct/2011:13:24:00","TZ":"-0500","METHOD":"GET","PATH":"/product/demos/product2","St
{"IP":"18.656.618.46","DATE":"26/Oct/2011:17:15:30","TZ":"-0500","METHOD":"GET","PATH":"/download/download4.zip","S
{"IP":"14.688.663.667","DATE":"26/Oct/2011:21:02:30","TZ":"-0500","METHOD":"GET","PATH":"/news","Status":"200"}
{"IP":"13.07.338.684","DATE":"26/Oct/2011:21:02:30","TZ":"-0500","METHOD":"GET","PATH":"/download","Status":"200"}
{"IP":"14.688.663.667","DATE":"26/Oct/2011:21:02:30","TZ":"-0500","METHOD":"GET","PATH":"/news","Status":"200"}

{
  "IP": "688.615.03.332",
  "DATE": "26/Oct/2011:21:02:30",
  "TZ": "-0500",
  "METHOD": "GET",
  "PATH": "/product/product1",
  "Status": "200"
}

ABC **column1**

29,171 Categories

```
668.667.44.3·-·-·[25/Oct/2011:07:38:30·-0500]·GET·/download/download3.zip·HTTP/1.1·200·0·8751003045·"-"·"Mozil
13.386.648.380·-·-·[25/Oct/2011:17:06:00·-0500]·GET·/download/download6.zip·HTTP/1.1·200·0·8751000945·"-"·"Moz
06.670.03.40·-·-·[26/Oct/2011:13:24:00·-0500]·GET·/product/demos/product2·HTTP/1.1·200·0·8751000710·"-"·"Mozil
18.656.618.46·-·-·[26/Oct/2011:17:15:30·-0500]·GET·/download/download4.zip·HTTP/1.1·200·0·8751007754·"-"·"Mozi
14.688.663.667·-·-·[26/Oct/2011:21:02:30·-0500]·GET·/news·HTTP/1.1·200·0·8751001909·"-"·"Mozilla/5.0·(compatib
13.07.338.684·-·-·[26/Oct/2011:21:02:30·-0500]·GET·/download·HTTP/1.1·200·0·8751005416·"-"·"Mozilla/4.0·(compa
14.688.663.667·-·-·[26/Oct/2011:21:02:30·-0500]·GET·/news·HTTP/1.1·200·0·8751002457·"/news"·"Mozilla/5.0·(comp
688.615.03.332·-·-·[26/Oct/2011:21:02:30·-0500]·GET·/product/product1·HTTP/1.1·200·0·8751009644·"-"·"Mozilla/5
688.615.03.332·-·-·[26/Oct/2011:21:02:32·-0500]·GET·/product/product1·HTTP/1.1·200·0·8751004802·"/product/prod
688.615.03.332·-·-·[26/Oct/2011:21:02:34·-0500]·GET·/products/demos·HTTP/1.1·200·0·8751007570·"/product/produc
13.07.338.684·-·-·[26/Oct/2011:21:02:37·-0500]·GET·/download·HTTP/1.1·307·0·8751009145·"/download"·"Mozilla/4.
55.3.658.53·-·-·[26/Oct/2011:21:06:30·-0500]·GET·/buy·HTTP/1.1·200·0·8751006005·"-"·"Mozilla/5.0·(Windows;·U;·
55.3.658.53·-·-·[26/Oct/2011:21:06:56·-0500]·GET·/buy·HTTP/1.1·200·0·8751000279·"/buy"·"Mozilla/5.0·(Windows;·
14.323.74.653·-·-·[26/Oct/2011:21:07:00·-0500]·GET·/demo·HTTP/1.1·200·0·8751007058·"-"·"Jakarta·Commons-HttpCl
14.323.74.653·-·-·[26/Oct/2011:21:08:00·-0500]·GET·/demo·HTTP/1.1·200·0·8751008681·"/demo"·"Jakarta·Commons-Ht
14.323.74.653·-·-·[26/Oct/2011:21:09:00·-0500]·GET·/demo·HTTP/1.1·200·0·8751005626·"/demo"·"Jakarta·Commons-Ht
14.323.74.653·-·-·[26/Oct/2011:21:10:03·-0500]·GET·/demo·HTTP/1.1·200·0·8751008648·"/demo"·"Jakarta·Commons-Ht
53.667.16.82·-·-·[26/Oct/2011:21:10:30·-0500]·GET·/demo·HTTP/1.1·200·0·8751001359·"-"·"Jakarta·Commons-HttpCli
```

How is your data structured?

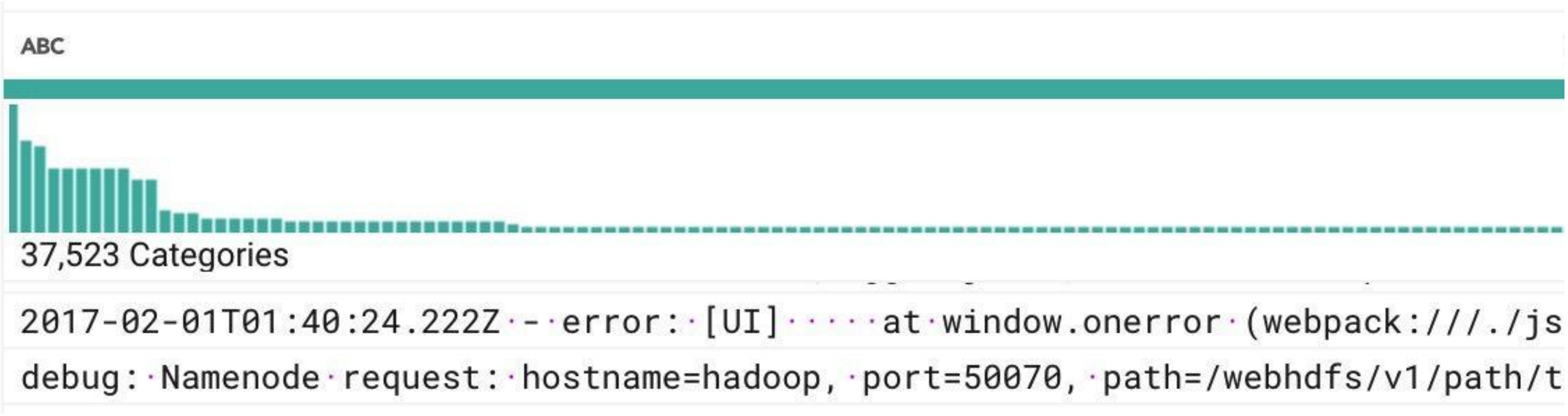
Is all your data structured identically?

How to map input structure(s) to desired output?

Relational data often has multi-structured columns

ABC name	ABC address	ABC phone
2 Categories	2 Categories	2 Categories
Jane Doe	1601 W. Broadway, Anaheim, CA, 92108	123-456-7890
Doe, John	132 Sansom Street, Philadelphia, PA	1 (123) 456-7890

Log data contains many types of events



How is your data structured?

Is all your data structured identically?

How to map input structure(s) to desired output?

Analysis and visualization tools generally require tabular and homogenous data

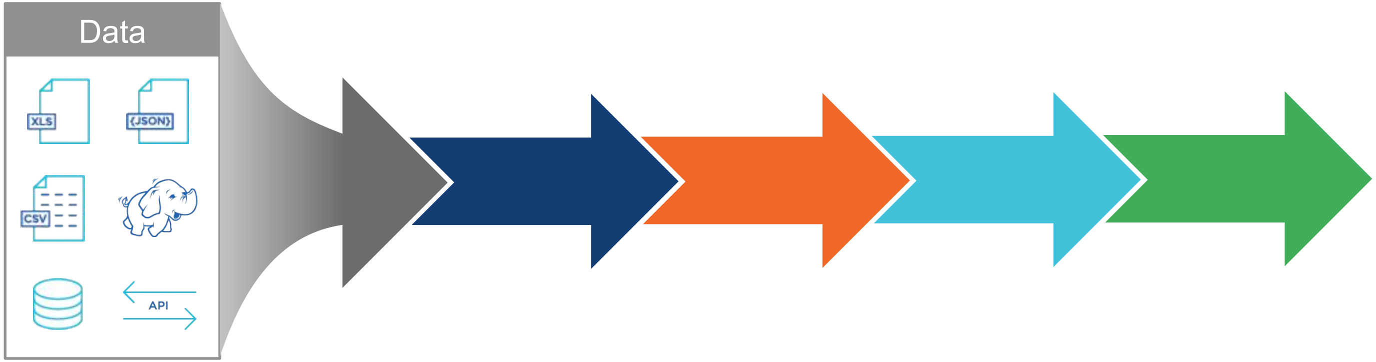
Data is increasingly

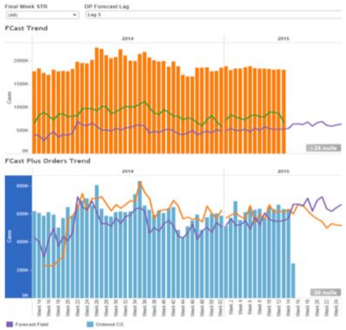
non-tabular or heterogeneous

The challenge

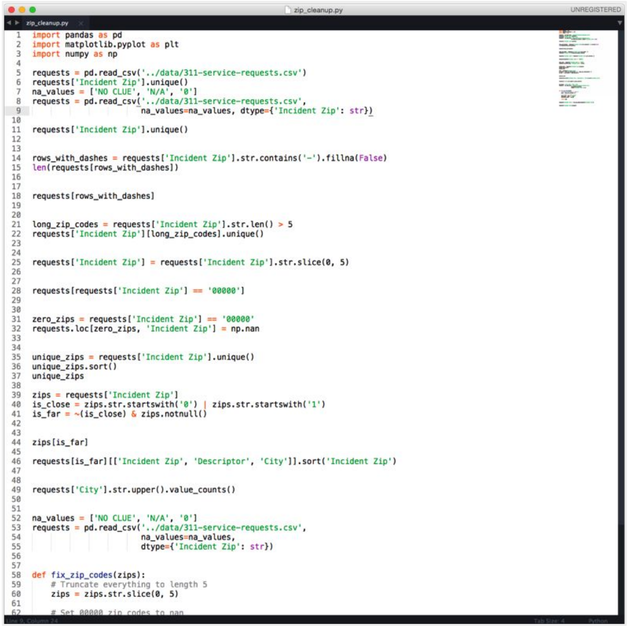
How do we make this more efficient?

To drive more value here!

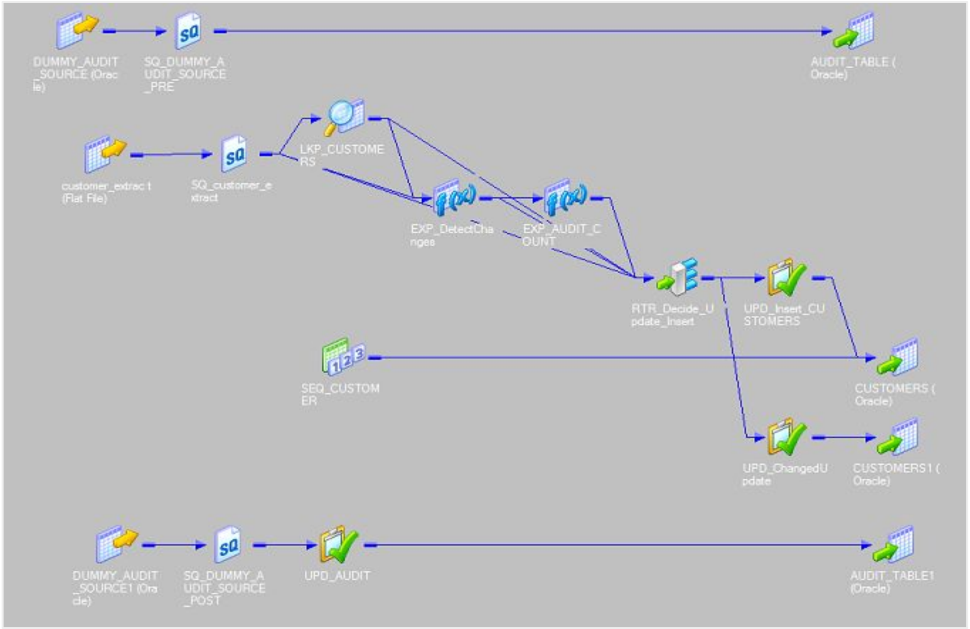




Conventional approaches inhibit self-service



Hand-Coding



Search / ETL

Conventional approaches inhibit self-service

Users must understand complex regular expressions.

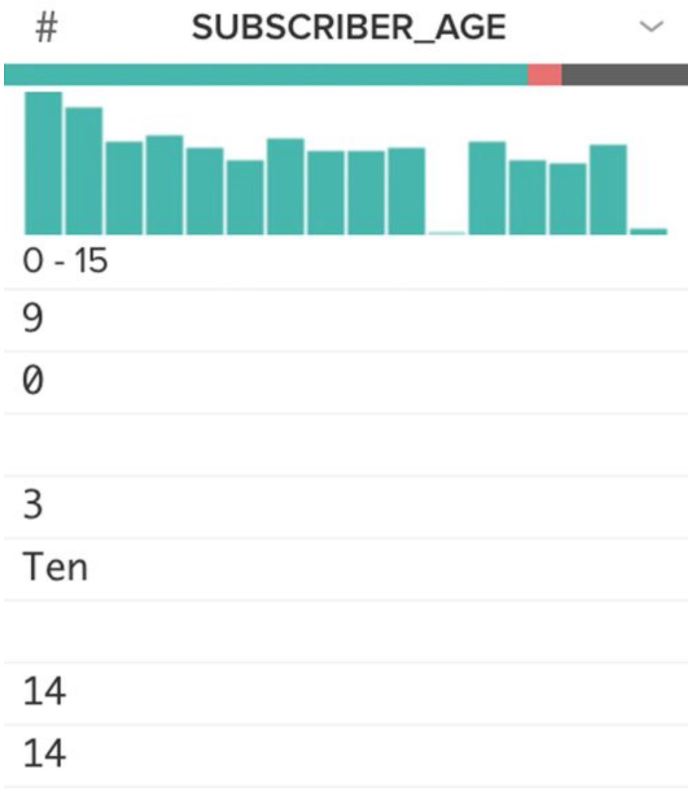
```
MOC|SFR/621630263|/
MTC|ORG1/638590539|/
SMS-MT|SMSC/600000000|BOY/658510643
SMS-MO|SMSC/600000000|SFR/634989093
SMS-MO|SMSC/600000000|ORG1/608564604
```

↓

```
[A-Z]+(\-[A-Z]+)?\|[A-Z0-9]+/[0-9]{9}/([A-Z]+/[0-9]+)?
```

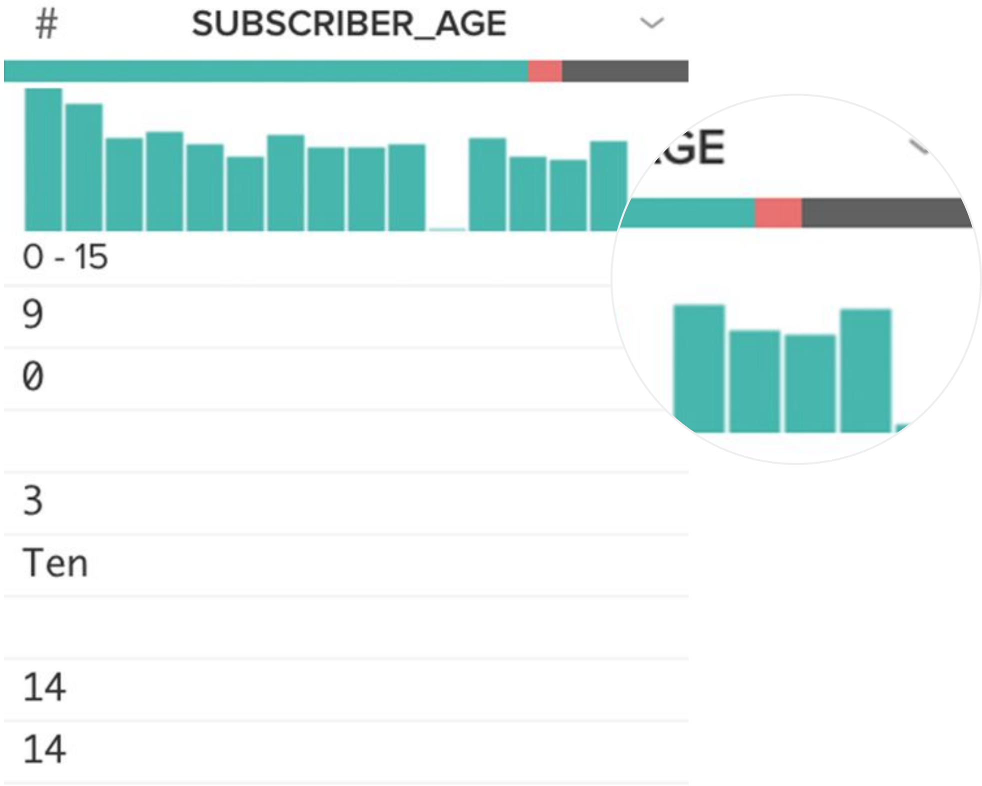
Pattern profiling combines automatic pattern discovery with interactive visualizations

Understanding data through profiling



For **ordinal data** (numbers, datetimes, ...), profiles can compactly and efficiently convey **data distribution**.

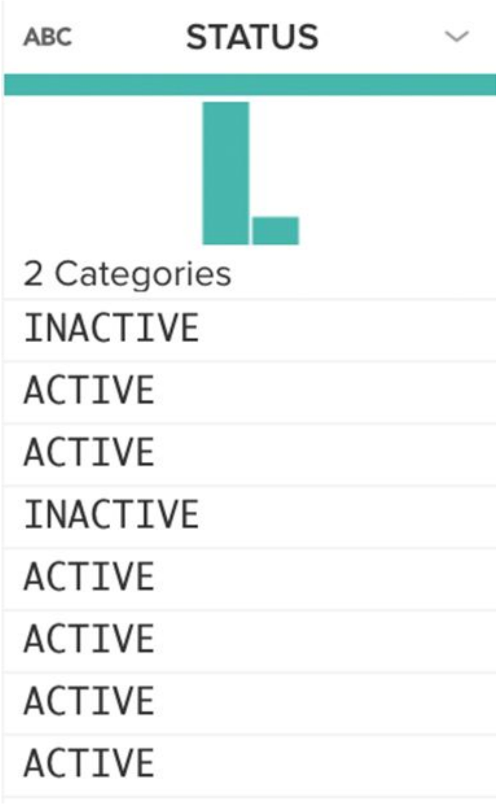
Understanding data through profiling



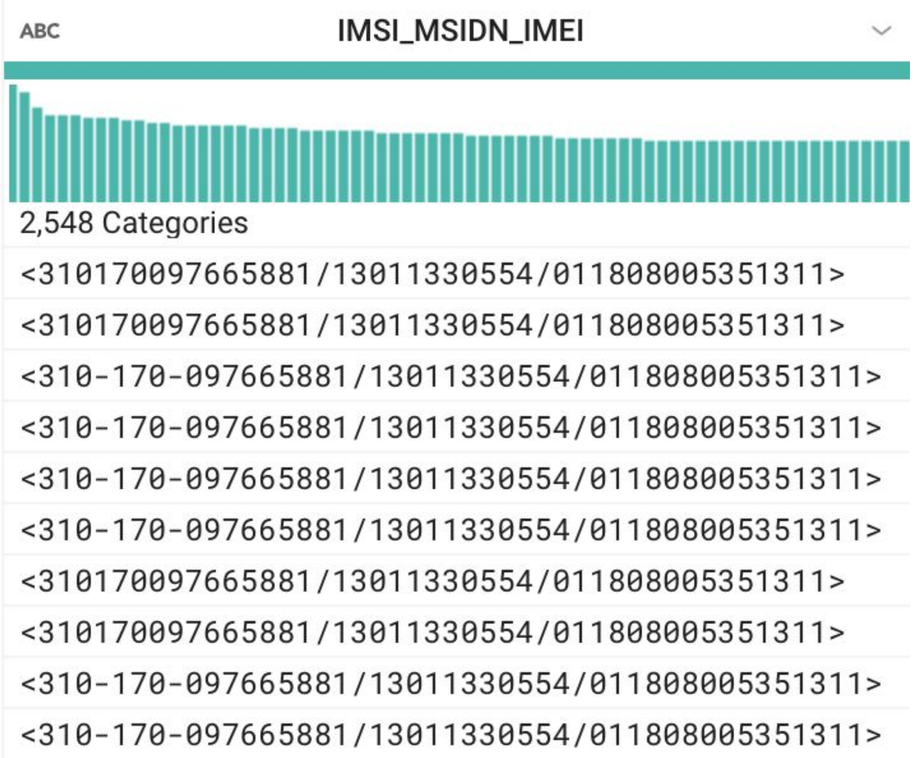
For **well typed data**, profiles can indicate the prevalence of **valid**, **invalid**, and **null** values.

Understanding data through profiling

For **categorical data** with a limited number of categories, profiles provide information on the **prevalence of each category**.



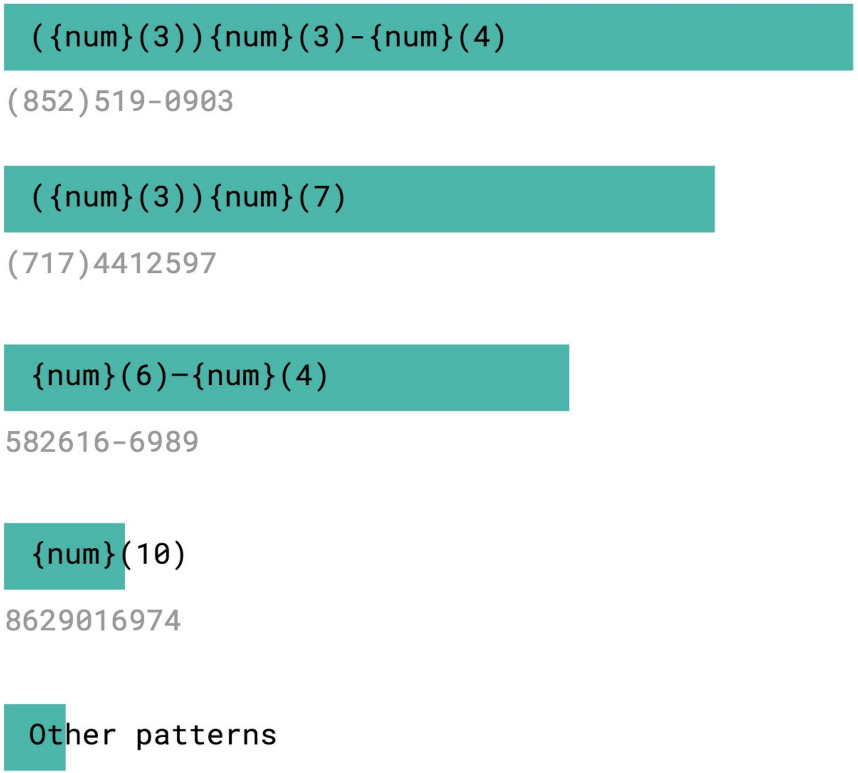
The nasty case of text



Categorical profiles fall back to **top k values**.

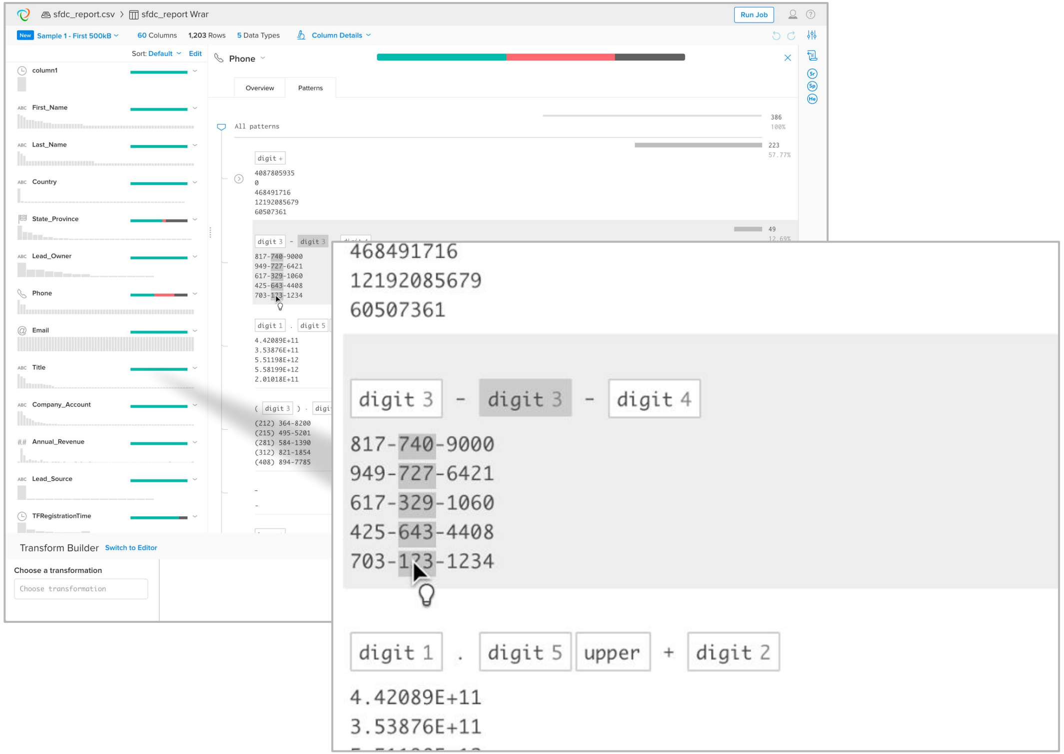
Data quality can only differentiate between null and non-null values.

Understanding data through patterns



Through **pattern based representations** of text, we can see **common** and **anomalous** patterns and drill down to specific records more easily.

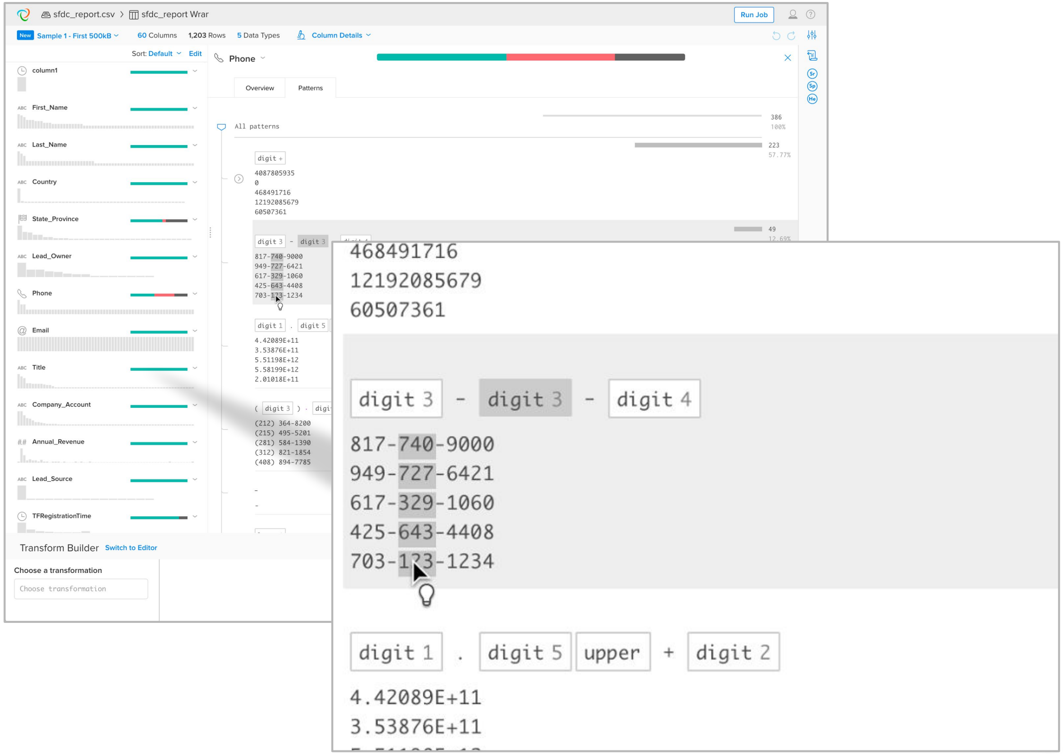
Pattern profiling



What?

- Automatically detects and displays formatting patterns within a column
- Visually summarizes content of a column into common and anomalous patterns

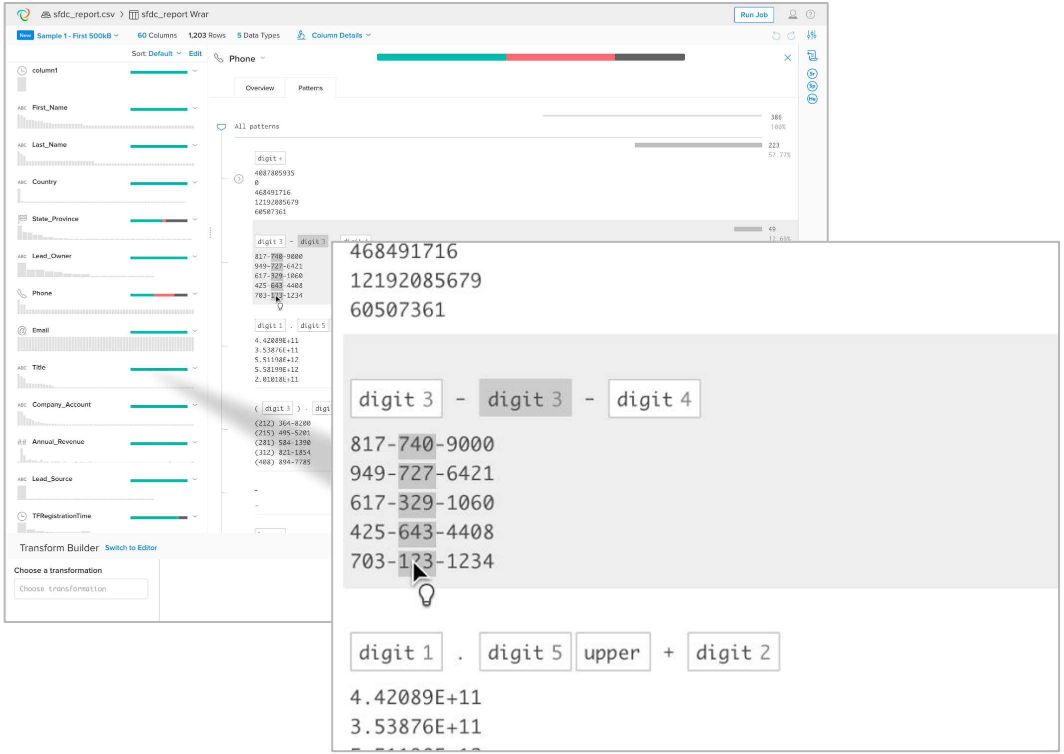
Pattern profiling



Why?

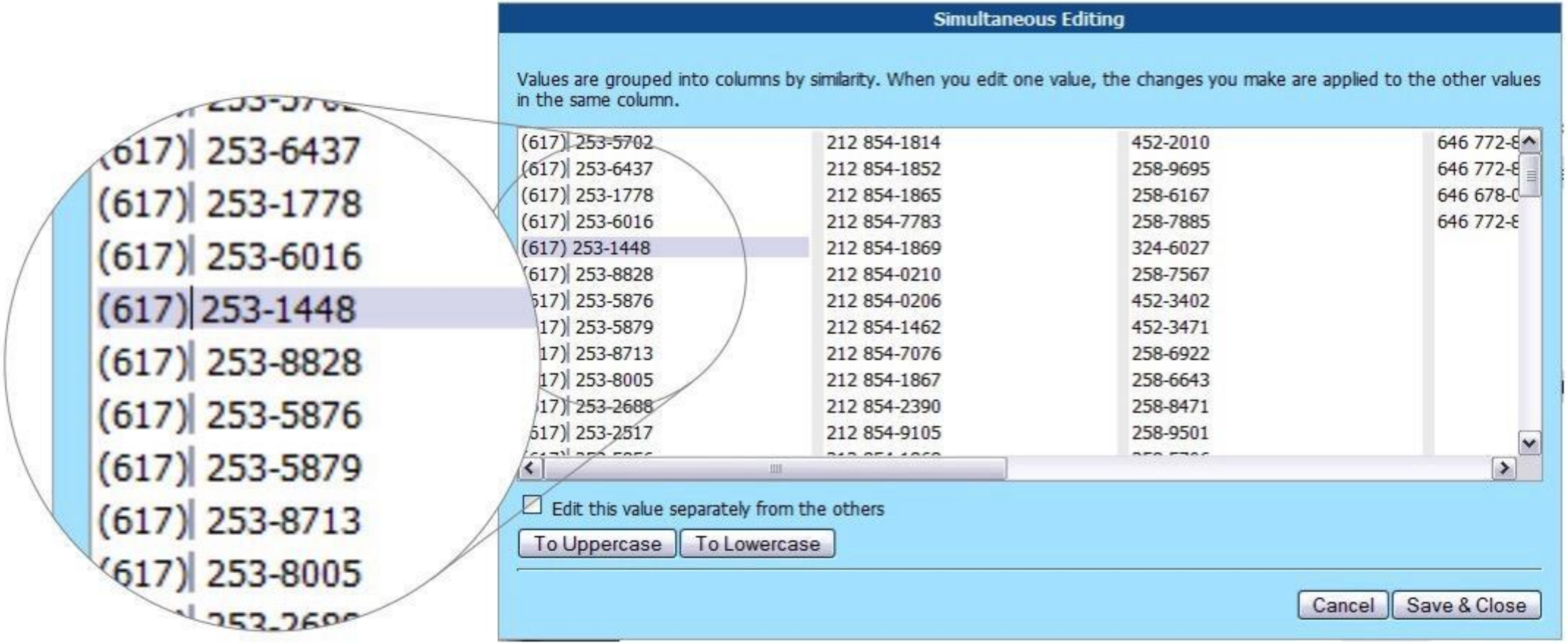
- Users can quickly understand and correct discrepancies within each column
- Provides starting point for users to identify and select subsets of records to transform

Pattern profiling



How?

- Cluster records into meaningful sub-groups
- Users interact with sub-groups and example records
- Predict transformations to apply across the data



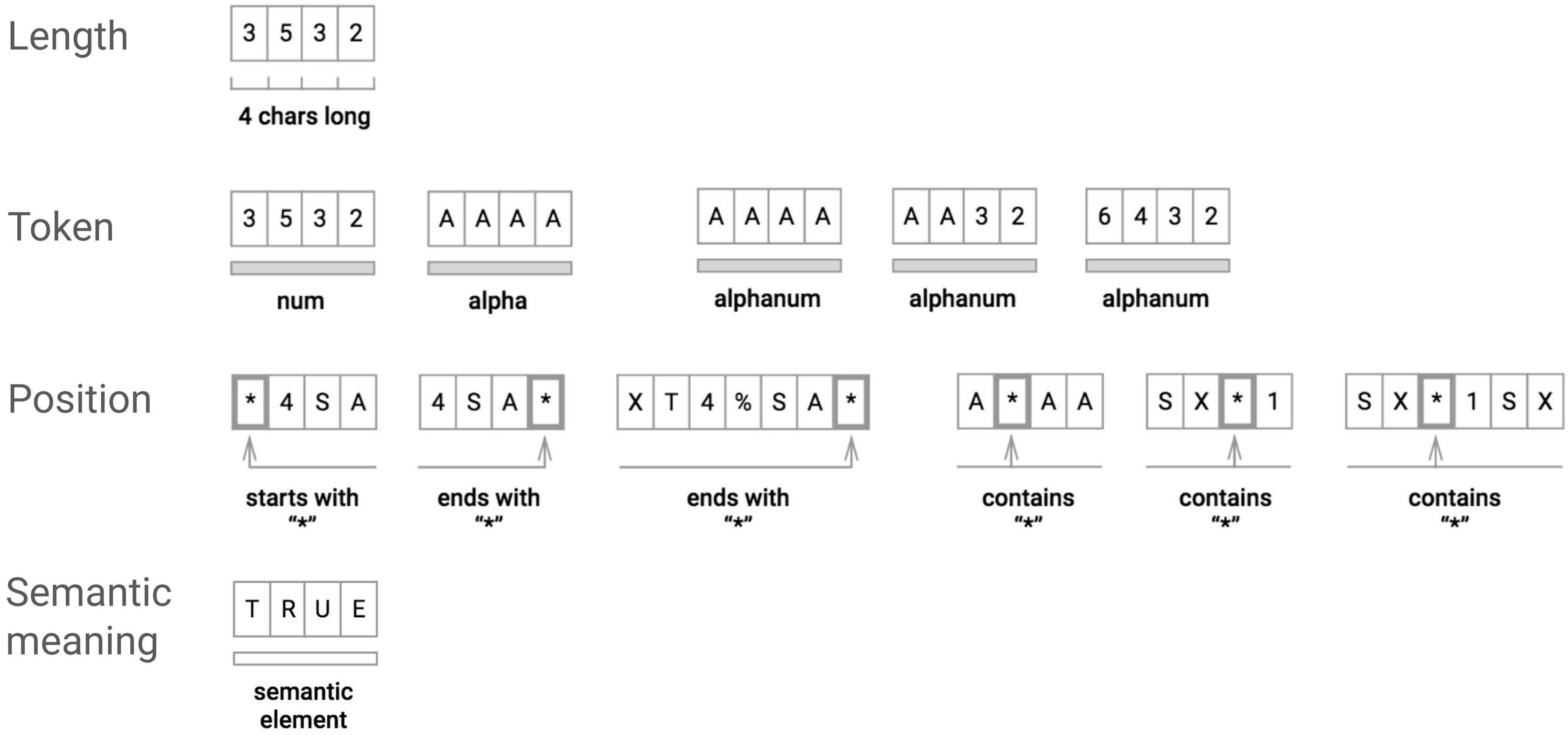
Hyunh, Miller, and Karger, ISWC 2007.

Potluck: Data Mash-Up Tool for Casual Users.

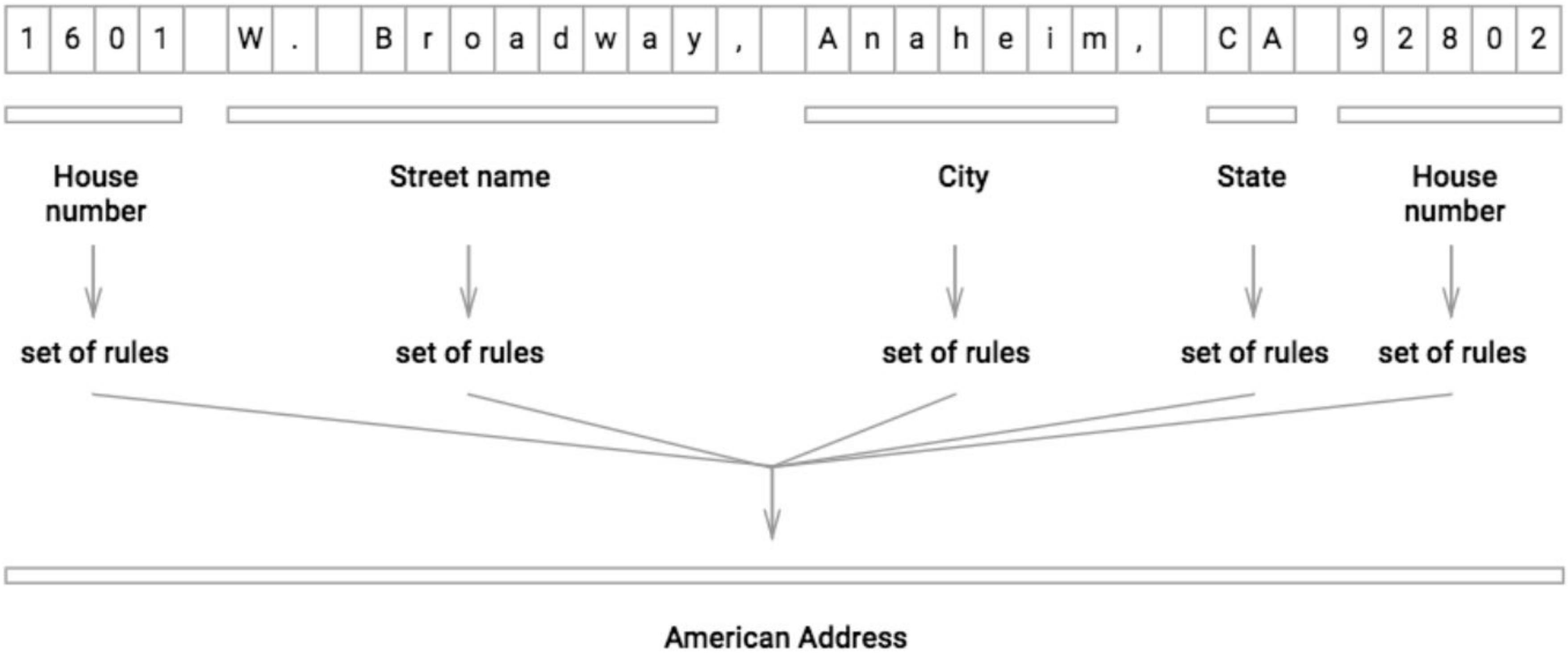
References:

Demo

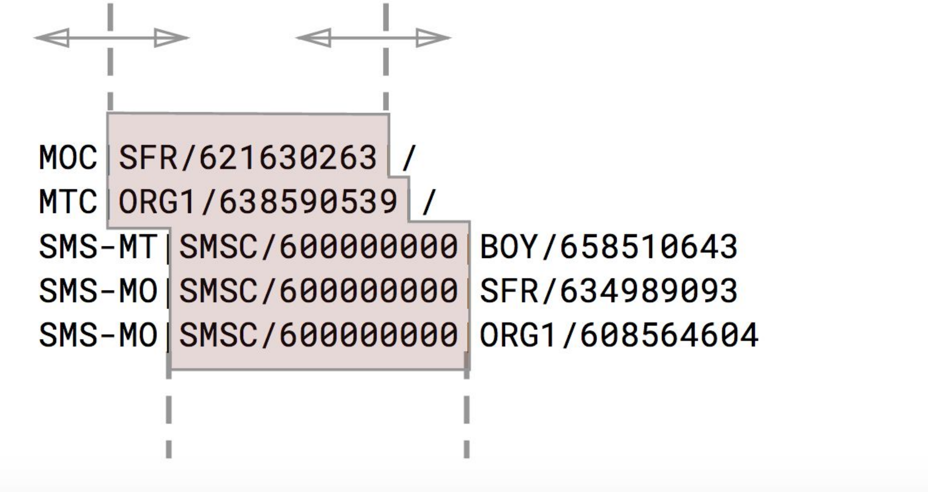
What is in a pattern?



What is in a pattern?



Capturing semi-structure with patterns

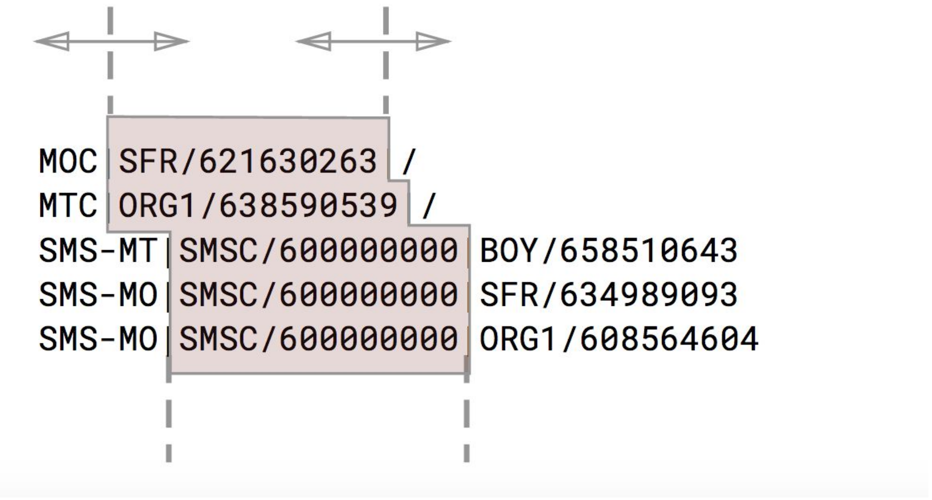


[A-Z]+(\-[A-Z]+)?\|[A-Z0-9]+/[0-9]{9}/([A-Z]+/[0-9]+)? (all **five** rows)

[A-Z]+|[A-Z0-9]+/[0-9]{9}/ (first **two** rows)

[A-Z]+\-[A-Z]+|[A-Z0-9]+/[0-9]{9}/[A-Z]+/[0-9]+ (last **three** rows)

Capturing semi-structure with patterns

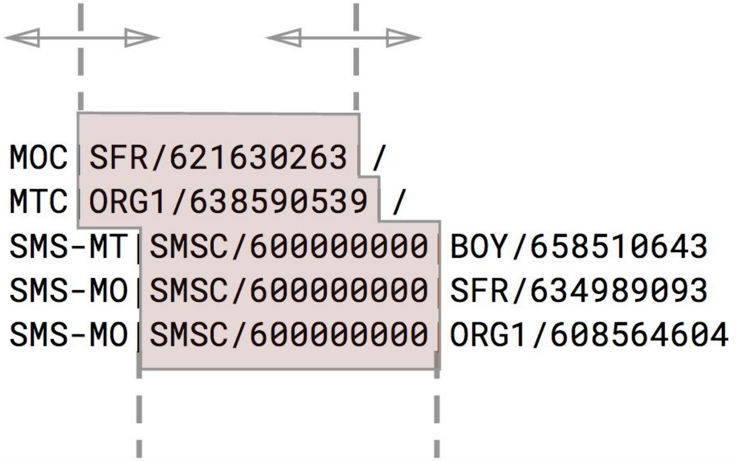


...<Product Code>...

[A-Z]+|<Product Code>/

[A-Z]+\-[A-Z]+|<Product Code>/[A-Z]+/[0-9]+

Making sense of hierarchical pattern structures



...<Product Code>... (union of **two** structures)

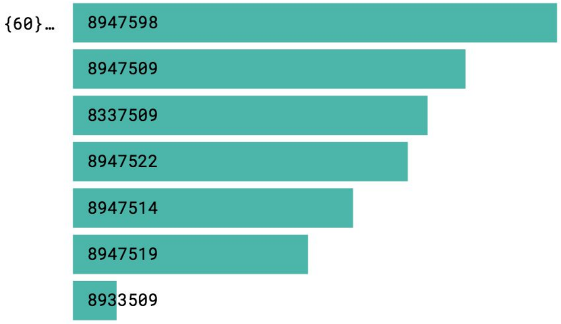
[A-Z]+|<Product Code>/

[A-Z]+\-[A-Z]+|<Product Code>/[A-Z]+/[0-9]+

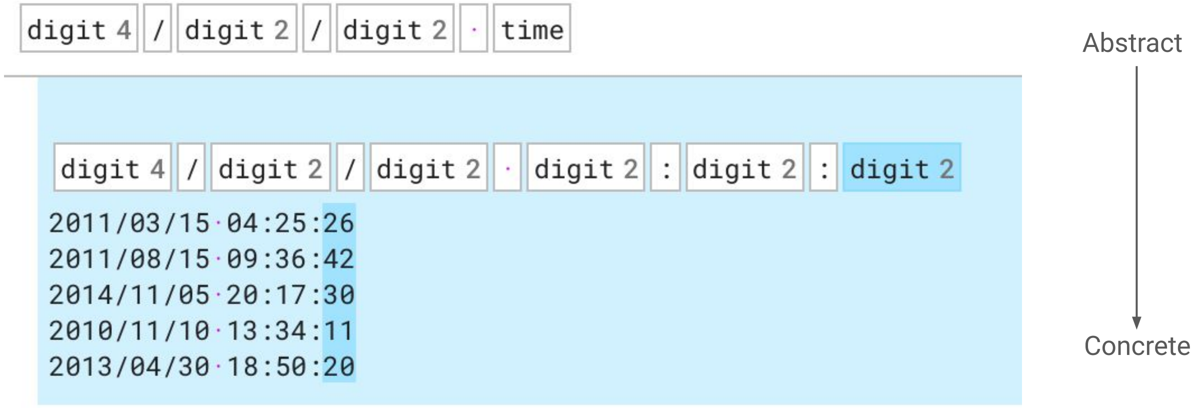
Making sense of hierarchical pattern structures

Cluster more **concrete, lower-level** patterns into more **abstract, higher-level** representations to give overviews of data.

Help the user transition from the higher-level **token** and **wild-card** representations to the **literal** data they encounter.

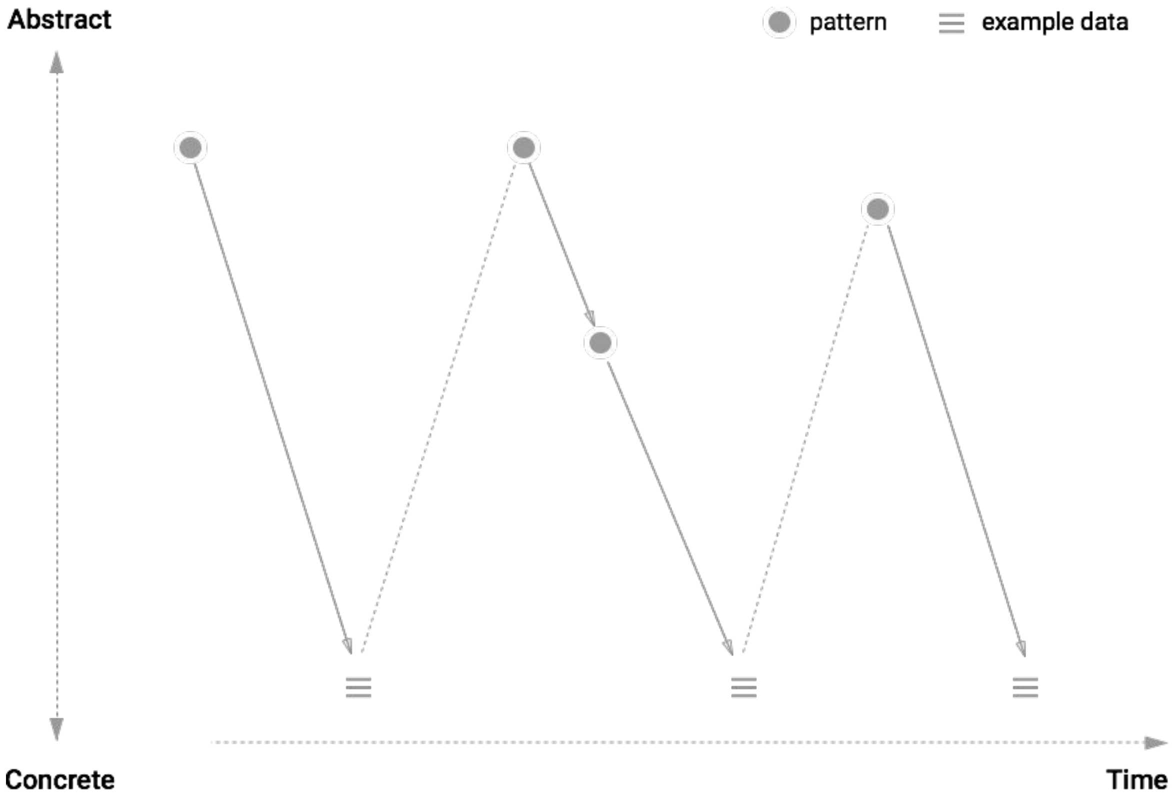


Examples as links between abstract and concrete



Example data can help users traverse the hierarchy of patterns and understand their context.

Examples as links between abstract and concrete



Inspiration from PADS

PADS (processing arbitrary data streams) is a data description language.

Format descriptions written in PADS are then used to process and structure data.

For more see:
* Fisher and Gruber 2003, PADS: Processing Arbitrary Data Streams.
* Fisher *et al.* 2008, From Dirt to Shovels: Fully Automatic Tool Generation from Ad Hoc Data.
* Xi *et al.* 2009, Ad Hoc Data and the Token Ambiguity Problem.

c	::=	$a \mid i \mid s$	(constants)
x			(variables)
p	::=	$c \mid x$	(parameters)

Base types b ::=

`Pint`	(generic, unrefined integer)
| `PintRanged`	(integer with min/max values)
| `Pint32`	(32-bit integer)
| `Pint64`	(64-bit integer)
| `PintConst`	(constant integer)
| `Pfloat`	(floating point number)
| `Palpha`	(alpha-numeric string)
| `Pstring`	(string; terminating character)
| `PstringFW`	(string; fixed width)
| `PstringConst`	(constant string)
| `Pother`	(punctuation character)
| `ComplexB`	(complex base type defined by regexp; *e.g.* date, time, *etc.*)
| `Pvoid`	(parses no characters; fails immediately)
| `Pempty`	(parses no characters; succeeds immediately)

Types T ::=

$b(p_1,\ldots,p_k)$	(parameterized base type)
| $x{:}b(p_1,\ldots,p_k)$	(parameterized base type; underlying value named x)
| `struct` $\{T_1; \ldots T_k;\}$	(fixed sequence of items)
| `array` $\{T;\}$	(array with unbounded repetitions)
| `arrayFW` $\{T;\}[p]$	(array; fixed length)
| `arrayST` $\{T;\}$[sep,term]	(array; separator and terminator)
| `union` $\{T_1; \ldots T_k;\}$	(alternatives)
| `enum` $\{c_1; \ldots c_k;\}$	(enumeration of constants)
| x:`enum` $\{c_1; \ldots c_k;\}$	(enumeration of constants; underlying value named x)
| `option` $\{T;\}$	(type T or nothing)
| `switch` x `of` $\{c_1 \Rightarrow T_1; \ldots c_k \Rightarrow T_k;\}$	(dependent choice)

Representations of parsed data d ::=

c	(constant)
| $\mathrm{in}_i(d)$	(injection into the i^{th} alternative of a union)
| $(d_1,\ldots,d_k)$	(sequence of data items)

Figure 1. Components of the IR (Fisher *et al.* 2008).

From dirt to shovels

Aims to **fully automate** PADS format generation from input data.

Does so by **tokenizing** the input data, **discovering structure** across tokenized records, and **refining** until minimal spec reached.

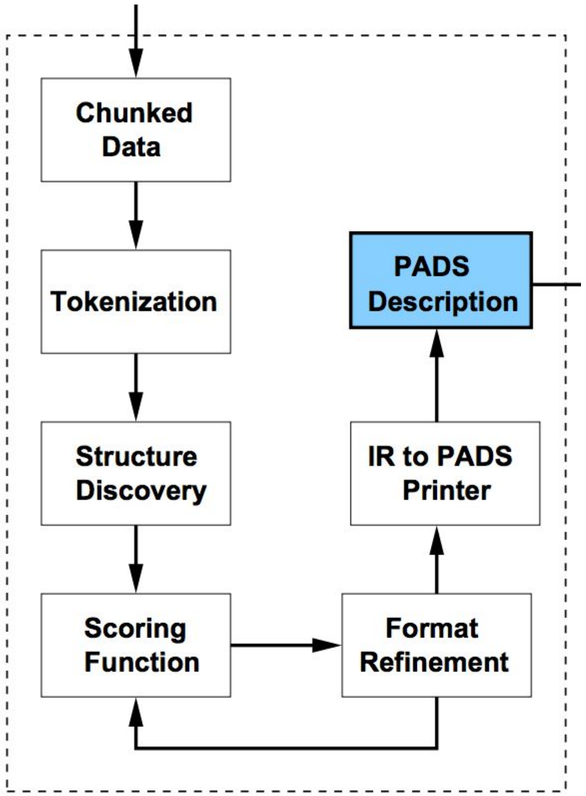


Figure 4. Architecture of the automatic tool-generation engine (Fisher *et al.* 2008).

Tokenization and lexing

Wildcard

{any}

Semantic Tokens

{ip-address}
{time}
{hashtag}
{email}

Character Tokens

{alpha-numeric}
{alpha}
{upper}
{lower}
{digit}
{delim}

Literal Tokens

{A-Za-z}
{0-9}
{/}
{-}
{_}
{|}
{,}
{ }
{<}
{>}

Specify a token tree going from most general (wildcard) to most specific (literal). Where possible, include parent child relationships.

Tokenization and lexing

```
192.168.2.255:GET
193.168.3.344:PUT
193.145.13.45:POST
```

```
{digit}{3}{delim}{digit}{3}{delim}{digit}{delim}{digit}{3}
{:}{upper}{3}
{digit}{3}{delim}{digit}{3}{delim}{digit}{2}{delim}{digit}{2}
{:}{upper}{4}
```

Pattern aggregation

Given **a pair of patterns** describing the data, determine how to **combine** and **generalize**.

Pattern aggregation

Given **a pair of patterns** describing the data, determine how to **combine** and **generalize**.

Use **structure discovery** to suggest possible struct and union tokens.

Structure discovery informs aggregation

Determine for a set of tokens *S* and data *D* which tokens are union tokens and which tokens are struct tokens.

Structure discovery informs aggregation

Determine for a set of tokens *S* and data *D* which tokens are union tokens and which tokens are struct tokens.

Given tokens `{{a}, {b}}`, a union relationship implies **`{a}|{b}`** and a struct relationship implies **`({a}{b})`**.

Structure discovery informs aggregation

Determine for a set of tokens *S* and data *D* which tokens are union tokens and which tokens are struct tokens.

```
for each token T in S:
  define histogram H[T] = distribution of number of matches of T over D

for each histogram H1 in H:
  for each histogram H2 in H:
    define E[H1][H2] = symmetric_relative_entropy(H1, H2)

define clusters C = agglomerative clustering with distance metric E
```

For more on structure discovery see:
Fisher *et al.* 2008, From Dirt to Shovels: Fully Automatic Tool Generation from Ad Hoc Data.

Structure discovery informs aggregation

Determine for a set of tokens *S* and data *D* which tokens are union tokens and which tokens are struct tokens.

```
for each cluster C1 in C:
  if histograms [...C1] have high coverage and narrow distribution:
    C1 is struct
  if histograms [...C1] have lower coverage and wider distribution:
    C1 is union
```

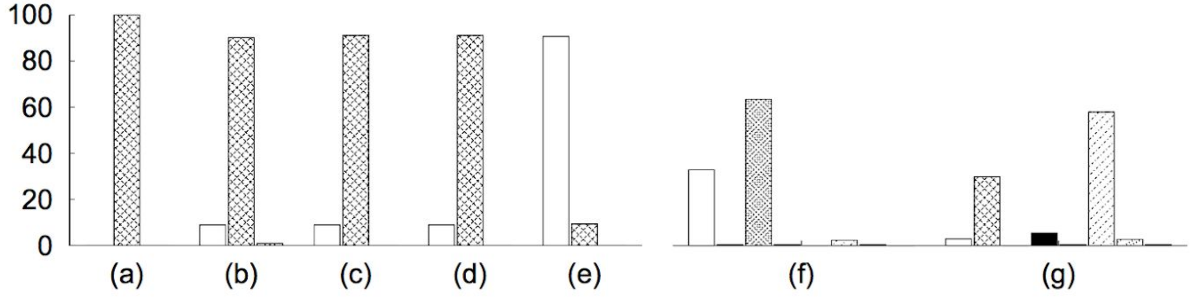


Figure 6. (a)-(e) are struct tokens, (f)-(g) are union tokens (Fisher *et al.* 2008).

Pattern aggregation

Given **a pair of patterns** describing the data, determine how to **combine** and **generalize**.

Use **structure discovery** to suggest possible struct and union tokens.

Align tokens (longest common subsequence) taking into account union and struct tokens.

Pattern aggregation

Given **a pair of patterns** describing the data, determine how to **combine** and **generalize**.

Use **structure discovery** to suggest possible struct and union tokens.

Align tokens (longest common subsequence) taking into account union and struct tokens.

Use **token hierarchy** to combine non-aligned tokens.

Token hierarchy informs aggregation

```
{upper},{lower},{alpha},{foo},{bar},{foobar}
{upper}|{lower} = {alpha}                        IR
{foo}|{bar} = {foobar}
```

With `{upper}{lower}`, `{foo}{bar}` as union clusters

Align `{upper}{foo}`
`{lower}{bar}`

Use semantic token hierarchy from IR `{upper}|{lower} = {alpha}, {foo}|{bar} = {foobar}`

```
  {upper}{foo}
+ {lower}{bar}
= {alpha}{foobar}
```

Pattern aggregation

Given **a pair of patterns** describing the data, determine how to **combine** and **generalize**.

Use **structure discovery** to suggest possible struct and union tokens.

Use **token alignment** (longest common subsequence) along struct and union tokens to find shared structure.

Use **token hierarchy** to combine non-aligned tokens.

Score candidate aggregations.

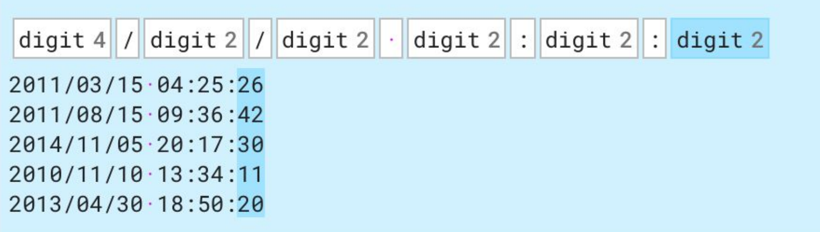
Scoring candidate aggregations

```
define score(pattern):
  n_wildcards  = count([{t} is {any} for {t} in pattern])
  n_kleene     = count([{t} has {*} for {t} in pattern])
  n_semantics  = length(intersection(semantic_tokens, pattern))
  n_basic      = length(intersection(basic_tokens, pattern))
  n_struct     = length(intersection(struct_tokens, pattern))

  score = -(w_1)(n_wildcards) - (w_2)(n_kleene) - (w_3)n_semantics +
           (w_4)(n_basic) + (w_5)(n_struct)

  return score / length(pattern)
```

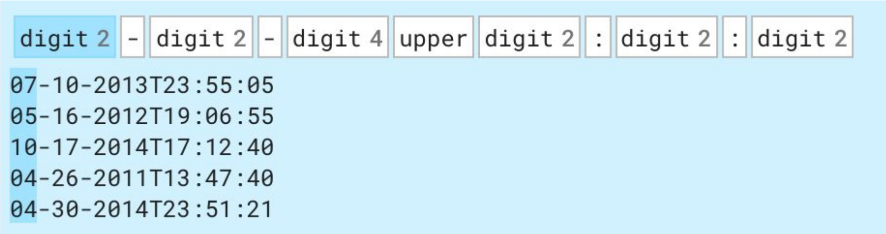
Pattern aggregation continued...

```
for each pattern {a} in patterns:
  for each pattern {b} in patterns:
    {parent}, score = combine_patterns({a}, {b})
    scores[{a}][{b}] = {{parent}, score}

sort scores desc

for each {a}, {b}, {parent}, score in scores:
  # if disjoint (neither child has found a better parent)
  if {a}, {b} in patterns:
    remove {a}, {b} from patterns
    add {parent} to patterns

# stop at the {root} pattern
repeat until only one pattern present
```

Generating interactive examples



For each set of candidate patterns, run **conditional aggregate** queries to accumulate example records.



digit 2 - digit 2 → `([0-9]{2})(-)([0-9]{2})`
$n captures the nth token group

Interact within a pattern by using **capturing groups** and matching across example records.

Extensions

Injecting supervision

Injecting Supervision

Tuning aggregation scoring through **human-in-the-loop feedback**

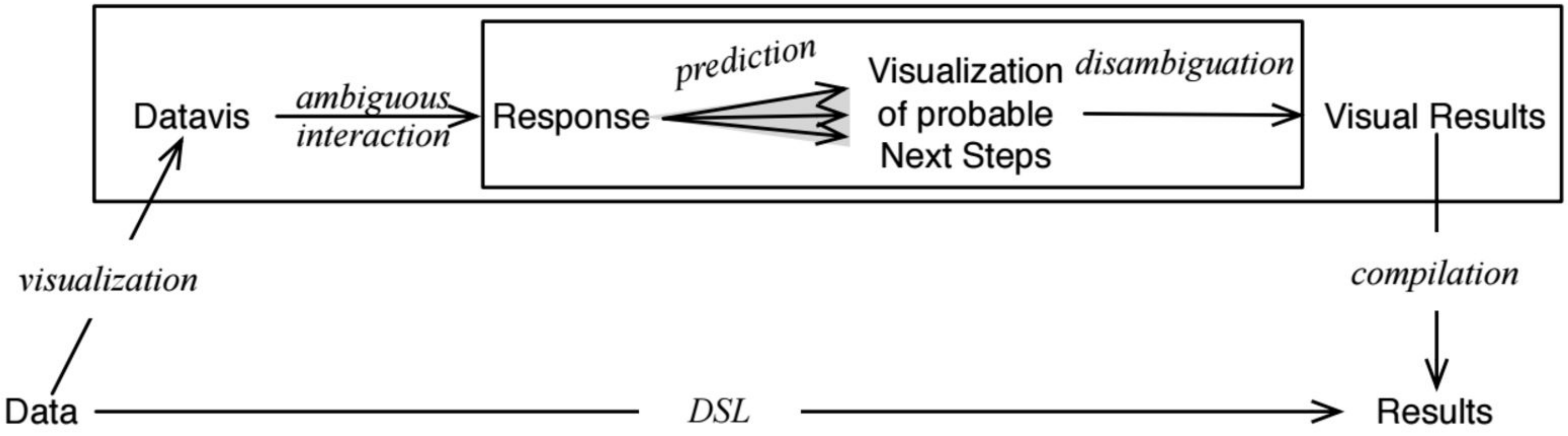


Figure 6. Predictive Interaction: The Guide / Decide Loop (Heer, Hellerstein, Kandel 2015).

Injecting Supervision

The **token ambiguity** problem

```
192.168.2.255
Option 1: int ‘.’ int ‘.’ int ‘.’ int
Option 2: float ‘.’ float
Option 3: ip-address
```

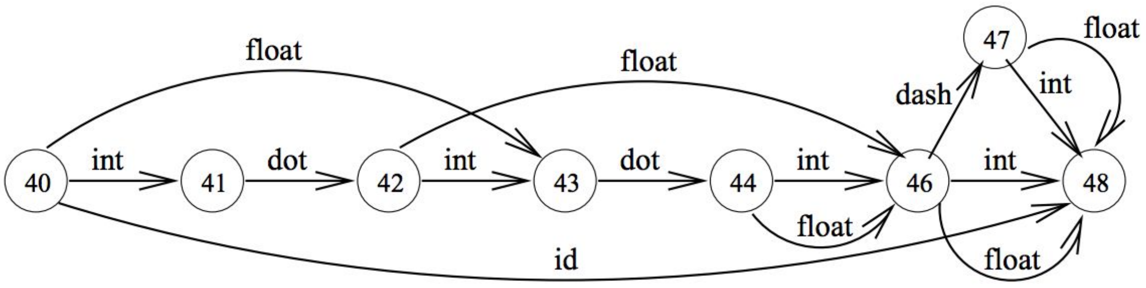


Figure 2. SeqSet from parsing “2.2-13.4” (Xi *et al.* 2009).

Xi *et al.* train HMMs and Hierarchical SVMs to traverse the tokenization paths, assign probabilities, pick best tokenization (2009).

Extensions

Injecting supervision

Improving performance

Thanks!

Michael Minar,
Athena Jiang,
Anish Doshi,
Lionel Michel,
and others @trifacta

TRIFACTA