

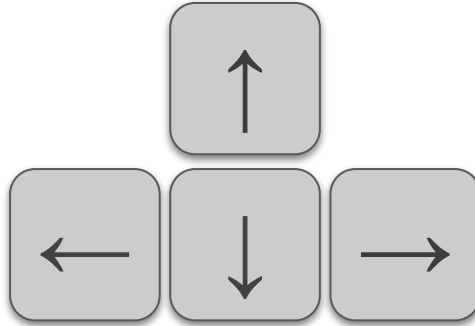
```
{“talk_title”: “Column Names as Contracts”,  
  
  “talk_author”: {  
    “author_name”: “Emily Riederer”,  
    “author_twtr”: “@emilyriederer”,  
    “author_site”: “emily.rbind.io”  
  },  
  “talk_forum”: {  
    “forum_name”: “R User Group”,  
    “forum_locn”: “Los Angeles”,  
    “forum_date”: “2021-04-27”  
  }  
}
```

# User interfaces make performance contracts

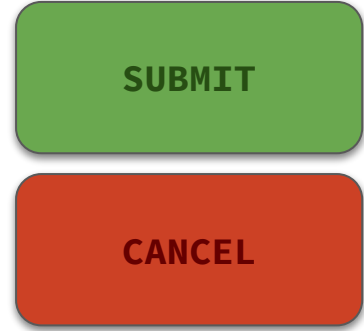
## Universal Symbols



## Grouping



## Aesthetics



Column names are the user interface of our data

A	B	C	D
1	10	11	1
2	20	12	10
3	30	13	100
4	40	14	1,000
5	50	15	10,000
...	...	...	...



User Interface



Functionality

# Subtle design choices challenge analytical (re)producibility

## Origin

Field provenance

When field loads

Unique keys

## Encoding

Indicator encoding

Metric definition

Null handling

## Usage

Feature leakage

Date formats

Allowed operations

# Subtle design choices challenge analytical (re)producibility

## Origin

Field provenance

## Encoding

Indicator encoding

Metric definition

Null handling

## Usage

“We had **a bunch of zeros that should have been coded ones** and the ones should have been coded zeroes.”

[Retraction Watch](#)

Date formats

Allowed operations

“These data sets often have multiple files that...have **unclear and sometimes duplicative variables**. Such complexities are commonplace among many data systems... I would not be surprised if coding errors were fairly common, and that the ones discovered constitute only the “tip of the iceberg.”

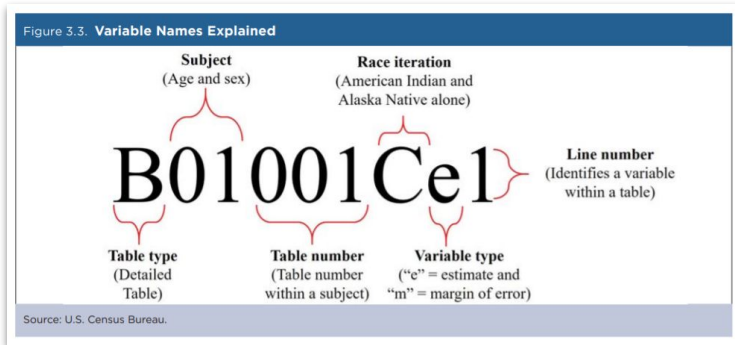
[Retraction Watch](#)

# Column names rarely encode human-interpretable meaning

## US Census Bureau

B19013\_001 (median household income)

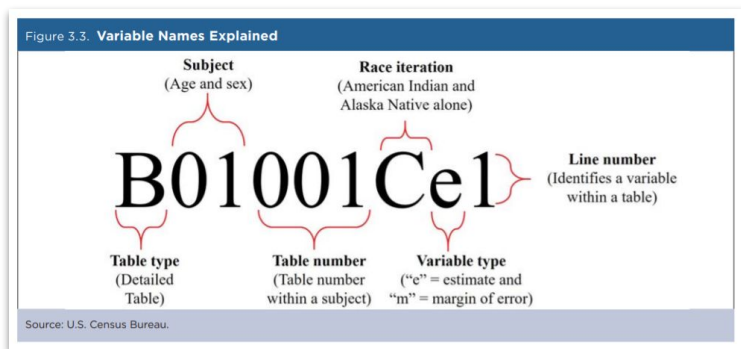
P013001 (median age)



# Column names rarely encode human-interpretable meaning

## US Census Bureau

B19013\_001 (median household income)  
P013001 (median age)



## Cooperative Congressional Election Study

year  
case\_id  
st  
st\_post  
cd\_up  
cd\_up\_post  
gender  
age  
educ  
race  
hispanic  
citizen  
religion  
marstat  
ownhome  
has\_child  
no\_milstat  
faminc  
employ  
union  
economy\_retro  
newsint  
approval\_pres  
approval\_gov  
approval\_sen1  
approval\_sen2

Indicator?  
Categorical?  
Continuous?

Same type,  
different  
conventions

Numeric or  
binary?

# Column names can help align data collectors and consumers

From lab assistant



to PI's desk

From you in the field



to you in the office

From the paper author



to the replicator

From the data engineer



to the analyst



## Using controlled vocabularies for column names

### WHAT

1. Establish a set of well-defined stubs
2. Stubs at different levels encode different semantics
3. Stubs may also carry associated contracts
4. Stubs are composed to communicate complex concepts

## An example vocabulary

<b>Stub</b>
ID
IND / IS
BIN
N
AMT
VAL
DT
TM
CAT
CD

## An example vocabulary

Stub	Semantics
ID	Unique entity identifier
IND / IS	Binary 0/1 indicator; rest of name describes 1 condition
BIN	Binary 0/1 indicator; rest of name describes 1 condition
N	Count of quantity or event occurrences
AMT	Sum-able real number amount (“denominator free”)
VAL	Numeric variables that are not inherently summable
DT	Date of an event
TM	Timestamp of an event
CAT	Human-readable categorical variable; maps to a CD
CD	System-generated categorical variable; maps to a CAT

## An example vocabulary

Stub	Semantics	Contracts
ID	Unique entity identifier	Numeric, primary / surrogate key
IND / IS	Binary 0/1 indicator; rest of name describes 1 condition	Always 0 or 1, non-null
BIN	Binary 0/1 indicator; rest of name describes 1 condition	Always 0 or 1
N	Count of quantity or event occurrences	Non-negative integer, non-null
AMT	Sum-able real number amount (“denominator free”)	Numeric
VAL	Numeric variables that are not inherently summable	Numeric
DT	Date of an event	Date, ISO 8601 (YYYY-MM-DD)
TM	Timestamp of an event	Datetime, YYYY-MM-DD HH:MM:SS
CAT	Human-readable categorical variable; maps to a CD	-
CD	System-generated categorical variable; maps to a CAT	-

# An example vocabulary

Stub	Semantics	Contracts
ID	Unique entity identifier	Numeric, primary / surrogate key
IND / IS	Binary 0/1 indicator; rest of name describes 1 condition	Always 0 or 1, <b>non-null</b>
BIN	Binary 0/1 indicator; rest of name describes 1 condition	Always 0 or 1
N	Count of quantity or event occurrences	Non-negative integer, non-null
AMT	Sum-able real number amount (“denominator free”)	Numeric
VAL	Numeric variables that are <b>not inherently summable</b>	Numeric
DT	Date of an event	Date, <b>ISO 8601</b> (YYYY-MM-DD)
TM	Timestamp of an event	Datetime, YYYY-MM-DD HH:MM:SS
CAT	Human-readable categorical variable; maps to a CD	One-to-one mapping with CD
CD	System-generated categorical variable; maps to a CAT	One-to-one mapping with CAT

## An example vocabulary

<b>Stub</b>
COUNTY
CASE
HOSP
...

## An example vocabulary

Stub	Semantics
COUNTY	Continental US county or county equivalents as defined by the US Census Bureau
CASE	Test-confirmed COVID-19 case as reported by state health department and as aligned by date of testing
HOSP	In-patient COVID-19 hospitalization as reported by the state health department
...	

## An example vocabulary

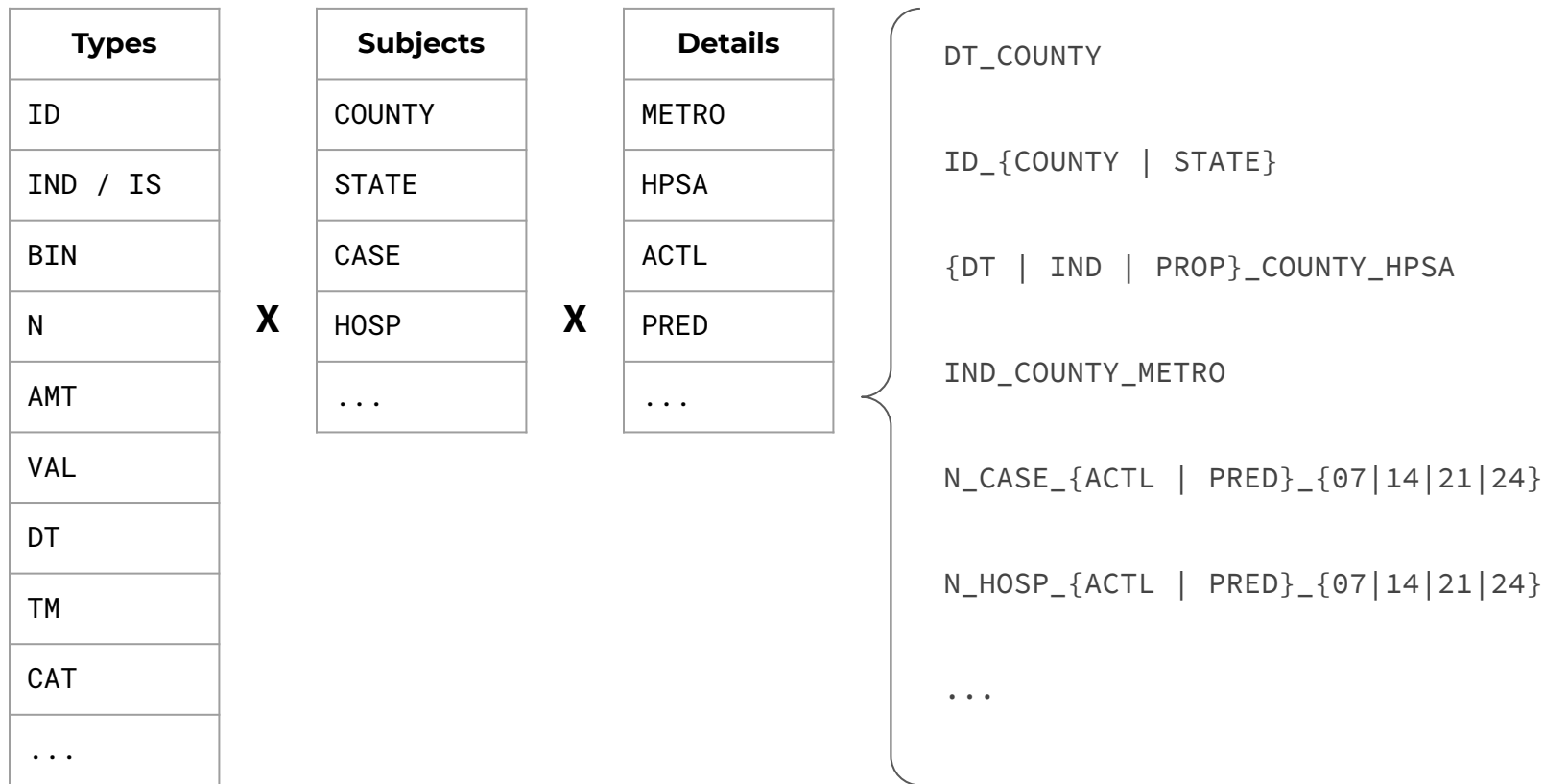
Stub	Semantics
COUNTY	Continental US county or county equivalents as defined by the US Census Bureau
CASE	<b>Test-confirmed</b> COVID-19 case as reported by state health department and as aligned by <b>date of testing</b>
HOSP	In-patient COVID-19 hospitalization as reported by the state health department
...	



## An example vocabulary

Stub	Semantics	Consequence
COUNTY	Continental US county or county equivalents as defined by the US Census Bureau	
CASE	<b>Test-confirmed</b> COVID-19 case as reported by state health department and as aligned by <b>date of testing</b>	Reports may continue to <b>backfill</b> , generally up to 7 days
HOSP	In-patient COVID-19 hospitalization as reported by the state health department	
...		

# An example vocabulary



# Subtle design choices *aid* scientific (re)producibility

## Origin

Field provenance

In 'entity' level

When field loads

In 'entity' level

Unique keys

ID field

## Encoding

Indicator encoding

IND stub -> positive

Metric definition

Clearly composed

Null handling

Non-null guarantees

## Usage

Feature leakage

`select(data, -contains("POST"))`

Date formats

DT stub -> ISO 8601

Allowed operations

VAL stub -> not summable

# Using controlled vocabularies for column names

## WHAT

1. Establish a set of stubs with well-defined meanings
2. Stubs at different levels can encode different semantics
3. Stubs may also carry associated contracts
4. Stubs are composed to communicate complex concepts

## WHY

- Automate maintenance burden for collectors
- Reduce cognitive load for consumers
- Add clarity for reviewers

# Data discoverability & documentation

## Data dictionary

Show 10 ▾ entries

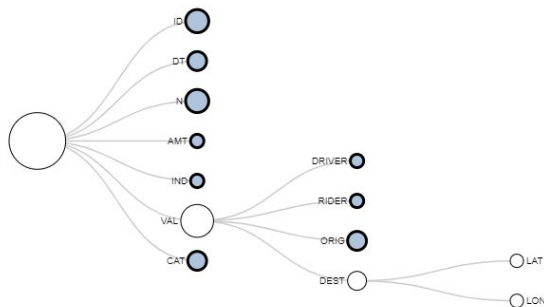
Search:

	variable	level1	level2	level3
	<input type="text" value="All"/>	<input type="text" value="All"/>	<input type="text" value="All"/>	<input type="text" value="All"/>
1	ID_DRIVER	ID	DRIVER	
2	ID_RIDER	ID	RIDER	
3	ID_TRIP	ID	TRIP	
4	DT_ORIG	DT	ORIG	
5	DT_DEST	DT	DEST	
6	N_DRIVER_PASSENGERS	N	DRIVER	PASSENGERS
7	N_TRIP_ORIG	N	TRIP	ORIG
8	N_TRIP_DEST	N	TRIP	DEST
9	AMT_TRIP_DIST	AMT	TRIP	DIST
10	IND_SURGE	IND	SURGE	

Showing 1 to 10 of 18 entries

Previous 1 2 Next

## Variable exploration



## Autocomplete



# Data validation

```
library(pointblank)
```

```
data %>%
```

```
  create_agent(actions =
```

```
    action_levels(stop_at = 0.1)) %>%
```

```
  col_vals_gte(starts_with("N"), 0) %>%
```

```
  col_vals_not_null(starts_with("IND")) %>%
```

```
  col_vals_in_set(starts_with("IND"), c(0,1)) %>%
```













```
  col_is_date(starts_with("DT")) %>%
```

```
  interrogate()
```

## Pointblank Validation

[2021-02-21|20:20:36]

DATA FRAME WARN — STOP 0.10 NOTIFY —

	STEP	COLUMNS	VALUES	TBL	EVAL	...	PASS	FAIL	W	S	N	EXT
1	 col_vals_not_null()	ind_county_hpsa	—	→	✓	5	5 1.00	0 0.00	—	○	—	—
2	 col_vals_in_set()	ind_county_hpsa	0, 1	→	✓	5	0 0.00	5 1.00	—	●	—	CSV
3	 col_vals_gte()	nm_county	0	→	✓	5	5 1.00	0 0.00	—	○	—	—
4	 col_vals_gte()	nm_state	0	→	✓	5	5 1.00	0 0.00	—	○	—	—
5	 col_vals_gte()	n_case_act1	0	→	✓	5	5 1.00	0 0.00	—	○	—	—
6	 col_vals_gte()	n_death_act1	0	→	✓	5	5 1.00	0 0.00	—	○	—	—
7	 col_vals_gte()	n_case_pred_07	0	→	✓	5	5 1.00	0 0.00	—	○	—	—
8	 col_vals_gte()	n_hosp_pred_07	0	→	✓	5	5 1.00	0 0.00	—	○	—	—
9	 col_vals_gte()	n_death_pred_07	0	→	✓	5	5 1.00	0 0.00	—	○	—	—
10	 col_vals_gte()	n_case_pred_14	0	→	✓	5	5 1.00	0 0.00	—	○	—	—
11	 col_vals_gte()	n_hosp_pred_14	0	→	✓	5	5 1.00	0 0.00	—	○	—	—
12	 col_vals_gte()	n_death_pred_14	0	→	✓	5	5 1.00	0 0.00	—	○	—	—

# Data wrangling

```
library(dplyr)
```

```
data %>%
```

```
  group_by(NM_STATE) %>%
```

```
  summarize(
```

```
    across(starts_with("IND"), mean),
```

```
    across(contains("_ACTL_"), sum)
```

```
  )
```

```
#> # A tibble: 51 x 4
```

```
#>   NM_STATE   IND_COUNTY_HPSA  N_CASE_ACTL  N_DEATH_ACTL
```

```
#>   <chr>          <dbl>        <dbl>        <dbl>
```

```
#> 1 Alabama      0.149      455582      7566
```

```
#> 2 Alaska       0.235      51338       250
```

```
#> 3 Arizona      0        753379     13098
```

# Data wrangling

```
library(dplyr)

data %>%

  group_by(NM_STATE) %>%

  summarize(

    across(starts_with("IND"), mean),

    across(contains("_ACTL_"), sum)

  )
```

```
#>   NM_STATE   IND_COUNTY_HPSA  N_CASE_ACTL  N_DEATH_ACTL
#>   <chr>          <dbl>        <dbl>        <dbl>
#> 1 Alabama      0.149         455582        7566
#> 2 Alaska       0.235         51338         250
#> 3 Arizona      0           753379        13098
```

```
import pandas as pd

cols_n = [vbl for vbl in data.columns if vbl[0:2] == 'IND_']

cols_grp = ["NM_STATE"]

data.groupby(cols_grp)[cols_n].mean()
```

```
#>           IND_COUNTY_HPSA
#> NM_STATE
#> Alabama      0.149
#> Alaska       0.235
#> Arizona      0
```



# Data wrangling

```
library(dplyr)

data %>%

  group_by(NM_STATE) %>%

  summarize(

    across(starts_with("IND"), mean, .names = "{gsub('IND', 'PROP', {.col})}")),

    across(contains("_ACTL_"), sum)

  )
```

```
#> # A tibble: 51 x 4
```

```
#>   NM_STATE   PROP_COUNTY_HPSA  N_CASE_ACTL  N_DEATH_ACTL
#>   <chr>         <dbl>         <dbl>         <dbl>
#> 1 Alabama      0.149      455582      7566
#> 2 Alaska       0.235      51338       250
#> 3 Arizona       0       753379     13098
```

# Beyond tabular data

```
{
  "url": "https://api.github.com/repos/octocat/Hello-World/pulls/1347",
  "id": 1,
  "title": "Amazing new feature",
  "user": {
    "login": "octocat",
    "id": 1
  },
  "body": "Please pull these awesome changes in!",
  "milestone": {
    "id": 1002604,
    "title": "v1.0",
    "description": "Tracking milestone for version 1.0",
    "creator": {
      "login": "octocat",
      "id": 1
    }
  },
  "created_at": "2011-04-10T20:09:31Z",
  "created_at": "2011-01-26T19:01:12Z",
  "assignee": {
    "login": "octocat",
    "id": 1,
  },
  "requested_reviewers": [
    {
      "login": "other_user",
      "id": 1
    }
  ],
  "repo": {
    "id": 1296269,
    "node_id": "MDExOjJlcG9zaXRvcnkxMjk2MjY5",
    "name": "Hello-World",
    "full_name": "octocat/Hello-World",
    "owner": {
      "login": "octocat",
    },
    "created_at": "2011-01-26T19:01:12Z",
  },
  "merged_by": {
    "login": "octocat",
    "id": 1,
  },
}
```

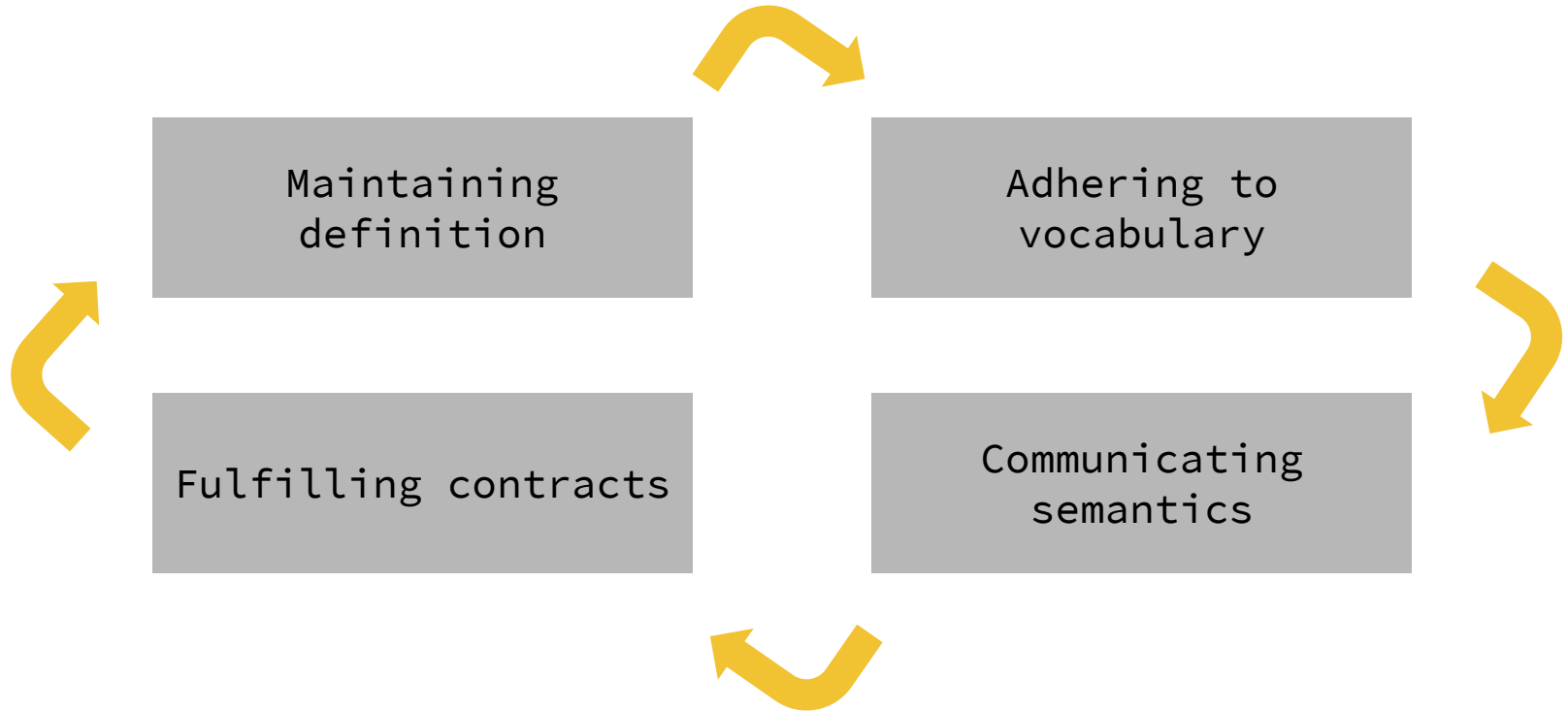
milestone

pull  
request

repo

- 2x “title”
- 6x “login”
- 3x “created”

Bad contracts are worse than no contracts



Automated tools can help us uphold contracts



R package

define & evaluate convo

local data



brings coding practices to SQL

templates, macros, tests

data in RDBMS

# Describe a controlled vocabularies with YAML

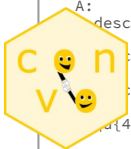
```
level1:  
  ID:  
    desc: Unique identifier  
    valid:  
      - col_vals_not_null()  
      - col_is_numeric()  
      - col_vals_between(1000, 99999)  
  IND:  
    desc: Binary indicator  
    valid:  
      - col_is_numeric()  
      - col_vals_in_set(c(0,1))  
    rename:  
      - when: SUM  
        then: 'N'  
      - when: AVG  
        then: P  
  AMT:  
    desc: Non-negative, summable quantity  
    valid:  
      - col_is_numeric()  
      - col_vals_gte(0)  
  VAL:  
    desc: Value  
    valid:  
      - col_is_numeric()  
    rename:  
      - when: AVG  
        then: VALAV  
  CAT:  
    desc: Category  
    valid:  
      - col_is_character()  
  CD:  
    desc: System-generated code  
    valid:  
      - col_is_character()  
  DT:  
    desc: Calendar date in YYYY-MM-DD format  
    valid:  
      - col_is_date()  
level2:  
  A:  
    desc: Type A  
  C:  
    desc: Type C  
  D:  
    desc: Type D  
  "a{4}": []
```

In YAML file, specify:

- Stub names
- Human-readable descriptions
- Validation contracts
- Renaming mappings

**library**(convo)

convo <- **read\_convo**("my-vocab.yml")



# Assess vocabulary quality

one meaning per stub

one stub per meaning



# Assess vocabulary quality

one meaning per stub

```
bad_convo <- list(  
  c("IND", "AMT", "CAT"),  
  c("DOG", "CAT")  
)  
pivot_convo(bad_convo)
```

```
#> $CAT  
#> [1] 1 2
```

one stub per meaning



# Assess vocabulary quality

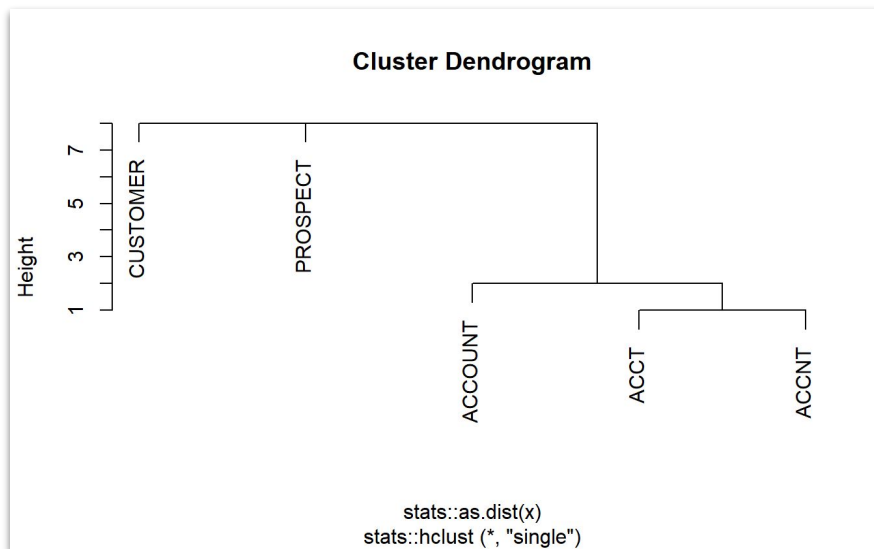
one meaning per stub

```
bad_convo <- list(  
  c("IND", "AMT", "CAT"),  
  c("DOG", "CAT")  
)  
pivot_convo(bad_convo)
```

```
#> $CAT  
#> [1] 1 2
```

one stub per meaning

```
bad_convo <- list(  
  c("IND", "IS", "AMT", "AMOUNT", "CAT", "CD"),  
  c("ACCOUNT", "ACCT", "ACCNT", "PROSPECT", "CUSTOMER")  
)  
clusts <- cluster_convo(bad_convo)  
plot(clusts[[2]])
```





# Challenge vocabulary realizations

evaluate names

discover new stubs



# Challenge vocabulary realizations

evaluate names

discover new stubs

```
col_names <- c(
  "ID_A", "IND_A", "XYZ_D", "AMT_B",
  "AMT_Q", "ID_A_1234", "ID_A_12"
)
evaluate_convo(convo, col_names, sep = "_")
```

```
#> Level 1
#> - XYZ_D
#> Level 2
#> - AMT_B
#> - AMT_Q
#> Level 3
#> - ID_A_12
```



# Challenge vocabulary realizations

evaluate names

```
col_names <- c(
  "ID_A", "IND_A", "XYZ_D", "AMT_B",
  "AMT_Q", "ID_A_1234", "ID_A_12"
)
evaluate_convo(convo, col_names, sep = "_")
```

```
#> Level 1
#> - XYZ_D
#> Level 2
#> - AMT_B
#> - AMT_Q
#> Level 3
#> - ID_A_12
```

discover new stubs

```
convo_colnames <- parse_stubs(col_names)
compare_convo(
  convo_colnames,
  convo,
  fx = "setdiff"
)
```

```
#> Level 1
#> - XYZ
#> Level 2
#> - B
#> - Q
#> Level 3
#> - 12
```








# Validate vocabulary promises

```
data_to_validate <- data.frame(IND_A = 1,  
                                IND_B = 5,  
                                DT_B = as.Date("2020-01-01"))  
agent <- create_pb_agent(convo, data_to_validate)  
pointblank::interrogate(agent)
```

## Pointblank Validation

[2021-02-07|13:05:04]

DATA FRAME tbl

STEP		COLUMNS	VALUES	TBL	EVAL	...	PASS	FAIL	W	S	N	EXT
1		col_is_numeric()	IND_A	—	→	✓	1 1.00	0 0.00	—	—	—	—
2		col_is_numeric()	IND_B	—	→	✓	1 1.00	0 0.00	—	—	—	—
3		col_vals_in_set()	IND_A	0, 1	→	✓	1 1.00	0 0.00	—	—	—	—
4		col_vals_in_set()	IND_B	0, 1	→	✓	1 0.00	0 1.00	—	—	—	CSV
5		col_is_date()	DT_B	—	→	✓	1 1.00	0 0.00	—	—	—	—

2021-02-07 13:05:04 CST

< 1 s

2021-02-07 13:05:05 CST



# Export validation infrastructure

```
convo <- read_convo("my-vocab.yml")
write_pb(convo,
  c("IND_A", "AMT_B"),
  filename = "convo-validation.yml")
```

```
read_fn: ~setNames(as.data.frame(matrix(1,
ncol = 2)), c("IND_A", "AMT_B"))
tbl_name: .na.character
label: '[2021-02-07|13:02:35]'
locale: en
steps:
- col_is_numeric:
  columns: vars(IND_A)
- col_vals_in_set:
  columns: vars(IND_A)
  set:
  - 0.0
  - 1.0
- col_is_numeric:
  columns: vars(AMT_B)
- col_vals_gte:
  columns: vars(AMT_B)
  value: 0.0
```



# Generate data documentation

```
vars <- c("AMT_A_2019", "IND_C_2020")
desc_df <- describe_names(
  vars,
  convo,
  desc_str = "{level1} of {level2} in given year")
DT::datatable(desc_df, escape = FALSE)
```

Show  entries Search:

	var_name	level1	level2	level3	desc
1	AMT_A_2019	AMT	A	2019	Non-negative, summable quantity of Type A in given year
2	IND_C_2020	IND	C	2020	Binary indicator of Type C in given year

Showing 1 to 2 of 2 entries Previous  Next



# Generate dictionary documentation

```
desc_df <- describe_convo(convo,  
                           include_valid = TRUE,  
                           for_DT = TRUE)  
DT::datatable(desc_df, escape = FALSE)
```

Show  entries Search:

level	stub	stub_desc	checks
3	1	AMT	Non-negative, summable quantity Expect that column is of type: numeric. Expect that values should be >= `0`.
5	1	CAT	Category Expect that column is of type: character.
6	1	CD	System-generated code Expect that column is of type: character.
8	1	DT	Calendar date in YYYY-MM-DD format Expect that column is of type: Date.
9	1	ID	Unique identifier Expect that all values should not be NULL. Expect that column is of type: numeric. Expect that values should be between `1000` and `99999`.
10	1	IND	Binary indicator Expect that column is of type: numeric. Expect that values should be in the set of `0`, `1`.
11	1	VAL	Value Expect that column is of type: numeric.
2	2	A	Type A
4	2	C	Type C
7	2	D	Type D

Showing 1 to 10 of 11 entries Previous  2 Next



## Open questions

- Alternative ways to specify convo (spreadsheet? CSV? better R constructors?)
- More convo-level metadata (e.g. delimiters)
- Increased flexibility in convo structure
  - Alternatives orders (e.g. suffix based, hierarchical)
  - Optional levels
  - Better regex support and documentation thereof
- Better aesthetics for documentation
  - Or better integration with fantastic {pointblank} docs





dbt makes it possible to systemized column-name contracts  
further up the pipeline

### Jinja Templating

Makes SQL more  
“programmable” with local  
variables and control flow

### Custom Macros

Enables creation of custom  
helper functions

### In-Pipeline Testing

Inject tests within data  
transformations to detect  
errors before they occur

dbt makes it possible to systemized column-name contracts  
further up the pipeline

## Jinja Templating

Makes SQL more  
“programmable” with local  
variables and control flow



Create valid names and  
avoid typos

## Custom Macros

Enables creation of custom  
helper functions



Operate on columns based on  
names

## In-Pipeline Testing

Inject tests within data  
transformations to detect  
errors before they occur



Test validity of operations  
and contract adherence

# SQL templating standardizes metric definition and naming

```
select
  id_county,
  dt_county,
  {% for l in var('lags') %}
    lag(cases, {{l}})
      over (partition by id_county
            order by dt_county)
      as n_case_pred_{{l}}
  {% if not loop.last %}, {% endif %}
  {% endfor %}
from predictions_source_table
```



```
select
  id_county,
  dt_county,
  lag(cases, 07)
    over (partition by id_county
          order by dt_county)
    as n_case_pred_07,
  lag(cases, 14)
    over (partition by id_county
          order by dt_county)
    as n_cases_pred_14,
  lag(cases, 21)
    over (partition by id_county
          order by dt_county)
    as n_cases_pred_21,
  lag(cases, 24)
    over (partition by id_county
          order by dt_county)
    as n_cases_pred_24,
  ...
```

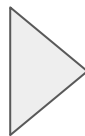
## Custom macros enable programmatic wrangling of data to enforce contracts

```
{% set cols = get_column_names( ref('data') ) %}
{% set cols_n = starts_with(cols, 'n') %}
{% set cols_dt = starts_with(cols, 'dt') %}
{% set cols_ind = starts_with(cols, 'ind') %}
{% set cols_oth =
    not_one_of(cols,
               cols_n + cols_dt + cols_ind %}

select

    {{ across(cols_oth, "{var}") }} ,
    {{ across(cols_n, "cast({var} as int)") }} ,
    {{ across(cols_dt, "date({var}) as {var}") }} ,
    {{ across(cols_ind, "coalesce({c}, 0)") }}

from {{ ref('data') }}
```



```
select

    amt_a,
    amt_b,

    cast(n_a as int64) as n_a,
    cast(n_c as int64) as n_c,

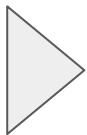
    date(dt_b) as dt_b,
    date(dt_d) as dt_d,

    coalesce(ind_b,0) as ind_b,
    coalesce(ind_c,0) as ind_c

from db.schema.data
```

# Testing confirms any non-enforceable contracts are upheld

```
{% set cols = get_column_names(ref('prep')) %}  
{% set cols_n = starts_with(cols, 'n') %}  
  
select *  
from {{ ref('model_monitor_staging') }}  
where  
  {% for c in cols_n %}  
    abs({{c}} - cast({{c}} as int64)) > 0.01 or  
  {% endfor %}  
  FALSE
```



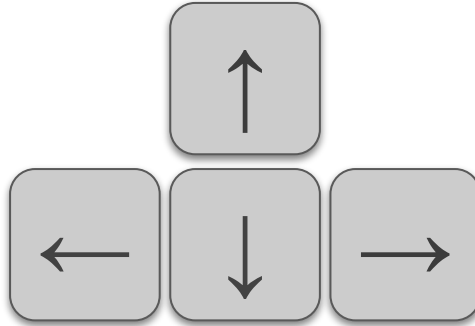
```
with dbt__CTE__INTERNAL_test as (  
  select *  
  from `sonorous-wharf-302611`.`dbt_emily`.`model_monitor_staging`  
  where  
    abs(n_case_act1 - cast(n_case_act1 as int64)) > 0.01 or  
    abs(n_death_act1 - cast(n_death_act1 as int64)) > 0.01 or  
    abs(n_case_pred_07 - cast(n_case_pred_07 as int64)) > 0.01 or  
    abs(n_hosp_pred_07 - cast(n_hosp_pred_07 as int64)) > 0.01 or  
    abs(n_death_pred_07 - cast(n_death_pred_07 as int64)) > 0.01 or  
    abs(n_case_pred_14 - cast(n_case_pred_14 as int64)) > 0.01 or  
    abs(n_hosp_pred_14 - cast(n_hosp_pred_14 as int64)) > 0.01 or  
    abs(n_death_pred_14 - cast(n_death_pred_14 as int64)) > 0.01 or  
    abs(n_case_pred_21 - cast(n_case_pred_21 as int64)) > 0.01 or  
    abs(n_hosp_pred_21 - cast(n_hosp_pred_21 as int64)) > 0.01 or  
    abs(n_death_pred_21 - cast(n_death_pred_21 as int64)) > 0.01 or  
    abs(n_case_pred_28 - cast(n_case_pred_28 as int64)) > 0.01 or  
    abs(n_hosp_pred_28 - cast(n_hosp_pred_28 as int64)) > 0.01 or  
    abs(n_death_pred_28 - cast(n_death_pred_28 as int64)) > 0.01 or  
    FALSE  
)  
select count(*) from dbt__CTE__INTERNAL_test
```

# Column names can make performance contracts

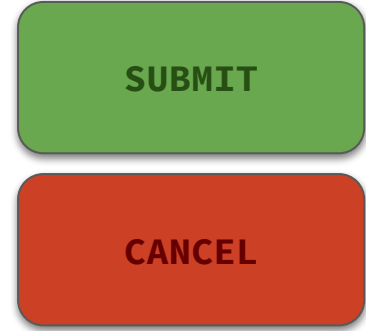
## Universal Symbols



## Grouping



## Aesthetics



# Thank you!

More thoughts on my website:

- Under the `data` tag: [emily.rbind.io/tags/data/](https://emily.rbind.io/tags/data/)
- [Column Names as Contracts](#)
- [Introducing {convo}](#) + open design questions!
- [Embedding column-name contracts in dbt pipelines](#)
- [{convo}](#) package website
- [dbtplyr](#) GitHub repo