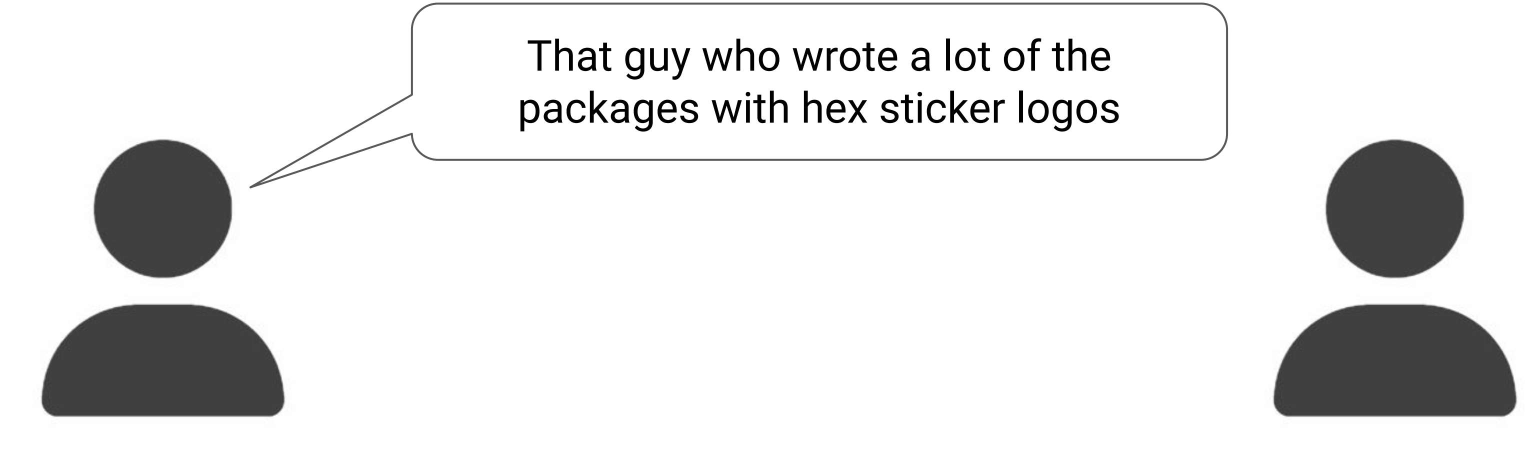
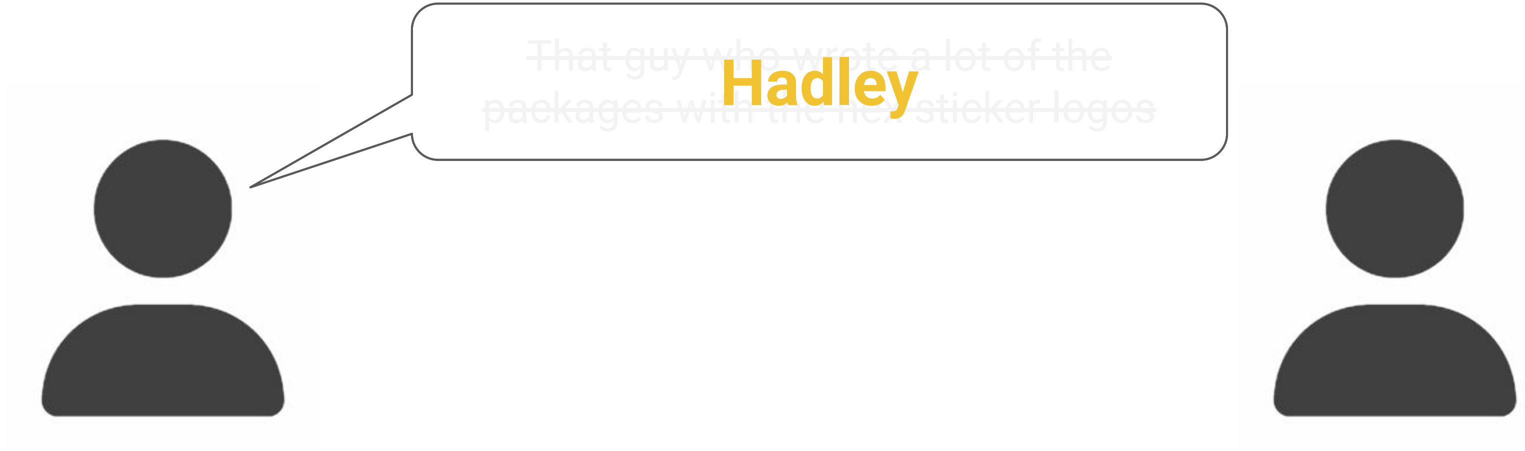
```
{“talk_title":
“starts_with(language):
    Translating select helpers to dbt”,


 “talk_author”: {
   “author_name”: “Emily Riederer”,
   “author_hndl”: “@emilyriederer”,
  },
 “talk_forum”: {
   “forum_name”: “posit::conf(2023)”,
   “forum_date”: “2023-09-19”
  }

}
```

A community’s shared language makes communication more efficient


That guy who wrote a lot of the packages with hex sticker logos

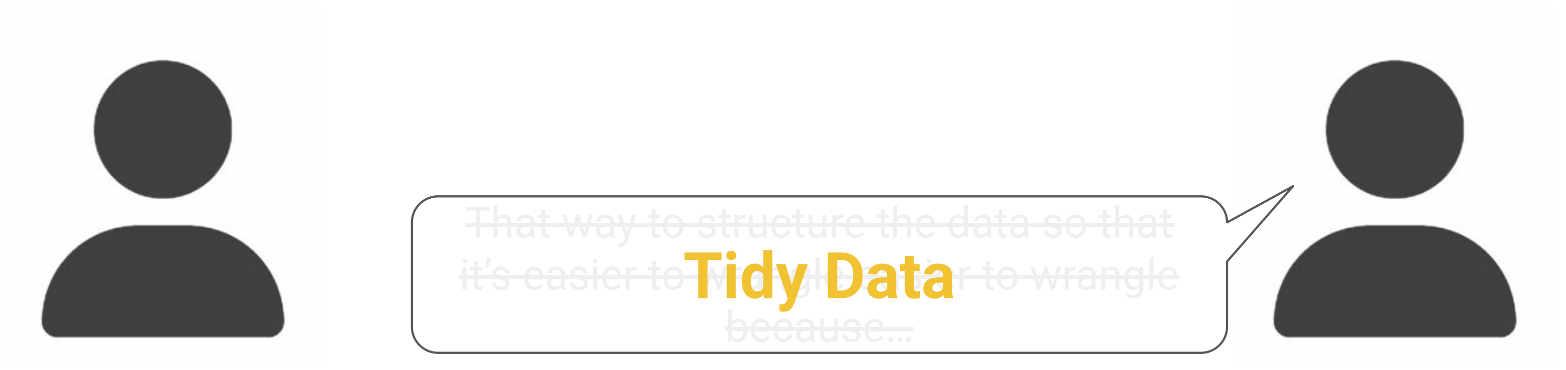
A community’s shared language makes communication more efficient


~~That guy who wrote a lot of the packages with the hex sticker logos~~
Hadley

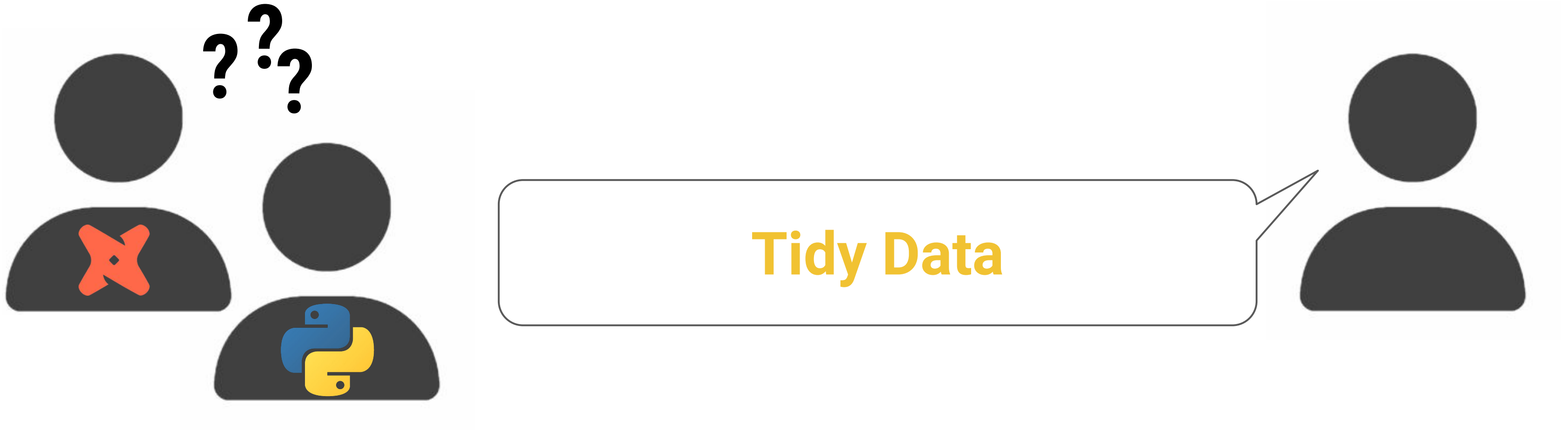
A community’s shared language embeds higher-level concepts

That way to structure the data so that it’s easier to wrangle easier to wrangle because…

A community's shared language embeds higher-level concepts

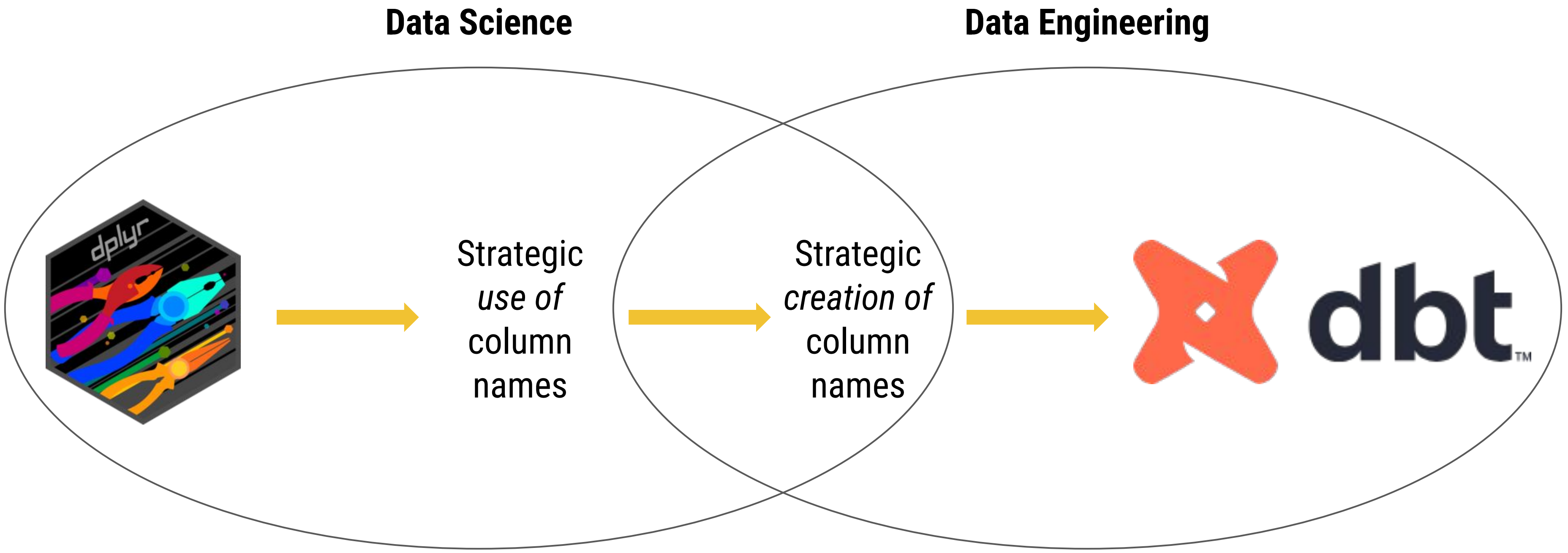

~~That way to structure the data so that it's easier to wrangle easier to wrangle because...~~
Tidy Data

A community's shared language embeds higher-level concepts



Translating **syntax** between **languages** transports **concepts** across **communities**

Language can help us learn, expand, and translate ideas


Data Science
Data Engineering
dplyr
Strategic *use of* column names
Strategic *creation of* column names
dbt

→ learning from dplyr’s language

→ columns names as a language

→ translating to dbt

tidyselect’s helper verbs expect to find meaning in column names

Key Functions	

Subset columns by name	`starts_with()`
	`ends_with()`
	`contains()`
	…

Iterate over transformations	`across()`
	`c_across()`

Iterate over filters	`if_any()`
	`if_all()`

tidyselect’s helper verbs expect to find meaning in column names

	n_user	**dt_spend**	**amt_spend**
`starts_with(‘n_’)`	✓	X	X
`contains(‘_spend_’)`	X	✓	✓
…	…	…	…

Select helpers incentivize strategic column naming

Efficient **transformation**
by acting on **data types**

Defensive **analysis & modeling**
by encoding **semantics**

Comprehensive **data validation**
by forming **contracts**

Find columns - then write more efficient transformation

```
marketing_campaign %>%

  group_by(channel) %>%

  summarize(

    across(starts_with("ind"), mean),
    across(contains("spend_pre_"), sum)

  )
```

Find columns - then prevent modeling feature leakage

```
recipe(ind_resubscribe ~ .,
       data = marketing_campaign) %>%

  update_role(starts_with("id_"),
              new_role = "id variable") %>%

  update_role(contains("_post_"),
              new_role = "metadata")
```

Find columns *dynamically* - then validate comprehensively

```
create_agent(data) %>%

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

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

interrogate()
```

→ learning from dplyr’s language

→ columns names as a language

→ translating to dbt

Column names are themselves a language

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

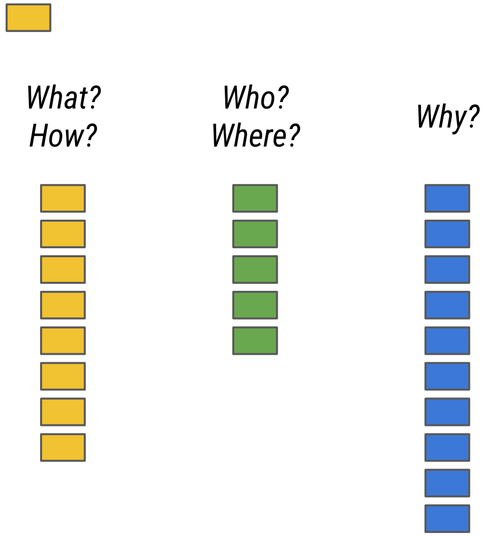
Abstraction

Reality

Columns names can be sentences not just words

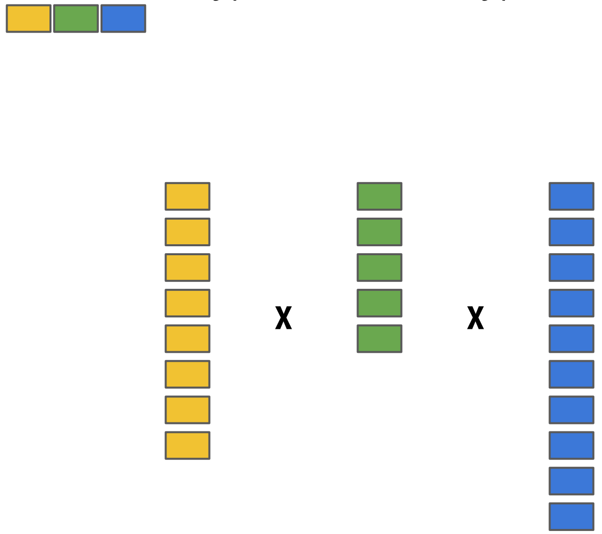
1. Define simple stubs

```
stub = semantics + contracts
```



2. Explain complex concepts

```
name = (type 1 stub)_(type 2 stub)_...
```



An example vocabulary

Stub
ID
IND
N
DT
...

An example vocabulary

Stub	**Semantics**
ID	
IND	Binary 0/1 indicator; **name describes positive case**
N	
DT	
...	

An example vocabulary

Stub	Semantics	Contracts
ID		
IND	Binary 0/1 indicator; name describes positive case	Always 0 or 1, non-null
N		
DT		ISO-8601 format
...		

An example vocabulary

Stub
USER
LOGIN
CLICK

An example vocabulary

Stub	**Semantics**
USER	Unique site visitor, **as determined by IP address**
LOGIN	
CLICK	

An example vocabulary

Stub	**Semantics**	**Consequence**
USER	Unique site visitor, as determined by IP address	**Inconsistent** across devices
LOGIN		
CLICK		

An example vocabulary

Types
ID
IND
N
AMT
VAL
DT
...

x

Entity
USER
LOGIN
SESSION
CLICK
...

x

Details
UTM
DUR
...

```
{DT | TM}_{LOGIN | SESSION}

ID_{USER | SESSION | LOGIN}

AMT_{SESSION | VIEW}_DURATION

...
```

→ learning from dplyr’s language

→ columns names as a language

→ translating to dbt

dbt is an data engineering framework on top of the SQL language

Clean Code

- Variables
- Control flow
- Macros

Organized Projects

- Prescriptive structure
- Monolithic to atomic
- Version control

Developer Workflow

- Dev / prod environments
- Testing
- Orchestration
- Logging

dbt shares values with the R “culture” with a DRYer language

```
select

  coalesce(a, 0) as a,
  coalesce(b, 0) as b,
  coalesce(c, 0) as c,

  (a - lag(a,1) as w) /
      lag(a,1) over w as a_yoy,
  (b - lag(b,1) as w) /
      lag(b,1) over w as b_yoy,
  (c - lag(c,1) as w) /
      lag(c,1) over w as c_yoy

from my_db.my_schema.my_table
```

dbt shares values with the R “culture” with a DRYer language

```
select

  coalesce(a, 0)
  coalesce(b, 0)
  coalesce(c, 0)

  (a - lag(a,1) a
       lag(a,1) c
  (b - lag(b,1) a
       lag(b,1) c
  (c - lag(c,1) a
       lag(c,1) c

from my_db.my_sch
```

```
{% set vars = [‘a’,’b’,’c’]  %}

select

  {% for v in vars %}

  coalesce({{v}}, 0) as {{v}},

  yoy( {{v}} ) as {{v}}_yoy

  {% endfor %}

from {{ ref(‘my_table’) }}
```

1 Variables

2 Control flow

3 Macros & packages

dbtplyr translates select helpers to dbt

Key Functions	

Subset columns by name	`starts_with()`
	`ends_with()`
	`contains()`
	…

Iterate over transformations	`across()`
	`c_across()`

Iterate over filters	`if_any()`
	`if_all()`

It unlocks the same pattern: “find columns, do stuff”

Key Functions

You write…

dbt renders…

Subset columns by name

```
{% set cols =
          dbtplyr.get_column_names(ref(‘data’)) %}
{% set cols_ind =
          dbtplyr.starts_with(cols, ‘ind’) %}
{% set cols_notnull = [‘x’, ‘y’] %}
```

```
[‘x’,‘y’,‘ind_a’,‘ind_b’]
```

Iterate over transformations

Iterate over filters

It unlocks the same pattern: “find columns, do stuff”

Key Functions	You write…	dbt renders…
Subset columns by name	`{% set cols = dbtplyr.get_column_names(ref(‘data’)) %}` `{% set cols_ind = dbtplyr.starts_with(cols, ‘ind’) %}` `{% set cols_notnull = [‘x’, ‘y’] %}`	
Iterate over transformations	`select {{ dbtplyr.across( cols_ind, “avg({{var}}) as prop_{{var}}”) }} from {{ ref('data') }}`	`select avg(ind_a) as prop_ind_a, avg(ind_b) as prop_ind_b from {{ ref('data') }}`
Iterate over filters		

```
{% set cols =
          dbtplyr.get_column_names(ref(‘data’)) %}
{% set cols_ind =
          dbtplyr.starts_with(cols, ‘ind’) %}
{% set cols_notnull = [‘x’, ‘y’] %}
```

```
select
  {{ dbtplyr.across(
          cols_ind,
          “avg({{var}}) as prop_{{var}}”) }}
 from {{ ref('data') }}
```

```
select
 avg(ind_a) as prop_ind_a,
 avg(ind_b) as prop_ind_b
from {{ ref('data') }}
```

It unlocks the same pattern: “find columns, do stuff”

Key Functions	You write…	dbt renders…
Subset columns by name	`{% set cols = dbtplyr.get_column_names(ref(‘data’)) %}` `{% set cols_ind = dbtplyr.starts_with(cols, ‘ind’) %}` `{% set cols_notnull = [‘x’, ‘y’] %}`	
Iterate over transformations	`select {{ dbtplyr.across( cols_ind, “avg({{var}}) as prop_{{var}}”) }} from {{ ref('data') }}`	`select avg(ind_a) as prop_ind_a, avg(ind_b) as prop_ind_b from {{ ref('data') }}`
Iterate over filters	`where {{ dbtplyr.if_all( cols_notnull, “not {{var}} is null”) }}`	`where not x is null and not y is null`

```
{% set cols =
        dbtplyr.get_column_names(ref(‘data’)) %}
{% set cols_ind =
        dbtplyr.starts_with(cols, ‘ind’) %}
{% set cols_notnull = [‘x’, ‘y’] %}


select
  {{ dbtplyr.across(
        cols_ind,
        “avg({{var}}) as prop_{{var}}”) }}
 from {{ ref('data') }}

 where

  {{ dbtplyr.if_all(
         cols_notnull,
         “not {{var}} is null”) }}
```

```
select
 avg(ind_a) as prop_ind_a,
 avg(ind_b) as prop_ind_b
from {{ ref('data') }}


where

  not x is null and
  not y is null
```

While dplyr helps scientists ‘ask’ column names, dbtplyr allows engineers to ‘tell’ column names how to act for future users

Consistent Naming

Reliable Meaning

Validated Values

Broken contracts frustrate users

	✓	✓	✗	✗
ID_VARIANT	**N_CLICK_07**	**N_CLICK_14**	**N_CLIK_21**	**N_28_CLICK**
1	100	172	202	291
2	112	136	154	191
3	156	181	202	235

Set parameters - define names

```
select

  id_variant,

    count_if(n_days <= 07)
        as n_click_07,

    count_if(n_days <= 14)
        as n_click_14
```

Set parameters - define names

```
{% set lags %}
  ['07','14','21']
{% endset %}

select

  id_variant,

  {% for l in var('lags') %}

      count_if(
          n_days <= {{l}}
      ) as n_click_{{l}},

  {% endfor %}
```

```
select

  id_variant,

    count_if(n_days <= 07)
        as n_click_07,

    count_if(n_days <= 14)
        as n_click_14
```

Broken contracts lie to users

x

DT_LOGIN	ID_LOGIN	IND_LOGIN
2021-01-01T10:25:28	123	1
2021-01-01T02:10:53	456	1
2021-01-02T07:20:00	789	0

Find columns - enforce contracts

```
select

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

Find columns - enforce contracts

```
{% set cols_dt =
   dbtplyr.starts_with(
     cols, ‘dt’
     )
%}

select

  {{ dbtplyr.across(
       cols_dt,
       “date({{var}})
          as dt_{{var}})”
      )
  }},
```

```
select

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

Overzealous automation can hide errors

N_A	N_B
12.00	3.25
19.00	4.67
27.00	8.99

```
cast({{var}} as int)
```

✓ N_A	✗ N_B
12	3
19	5
27	9

Find columns - confirm assumptions

```
select *
from `db`.`dbt_emily`.`my_source`
where

    abs(n_a - cast(n_a as int64)) > 0.01 or
    abs(n_b - cast(n_b as int64)) > 0.01 or
    abs(n_c - cast(n_c as int64)) > 0.01 or

   FALSE
```

Find columns - confirm assumptions

```
{% set cols_n =
         dbtplyr.starts_with(cols, 'n') %}

select *
from {{ ref('my_source') }}
where

 {%- for c in cols_n %}

  abs({{c}}
      - cast({{c}} as int64)
      ) > 0.01 or

 {% endfor %}

  FALSE
```

```
select *
from `db`.`dbt_emily`.`my_source`
where

    abs(n_a - cast(n_a as int64)) > 0.01 or
    abs(n_b - cast(n_b as int64)) > 0.01 or
    abs(n_c - cast(n_c as int64)) > 0.01 or

   FALSE
```

Translating **syntax** between **languages** transports **concepts** across **communities**

Questions?

↓ Get in touch ↓

@emilyriederer on Web | Twitter | GitHub | LinkedIn | Gmail

↓ Check out these resources ↓

dbt Learning Resources
dbtplyr repo
Blog post with example pipeline
Blog post on column name contracts
Jenny Bryan's talk on Naming Things

```
{“talk_title":
“starts_with(language):
    Translating select helpers to dbt”,


 “talk_author”: {
   “author_name”: “Emily Riederer”,
   “author_hndl”: “@emilyriederer”,
  },
 “talk_forum”: {
   “forum_name”: “posit::conf(2023)”,
   “forum_date”: “2023-09-19”
  }

}
```