rstd.io/tidy-eval-context

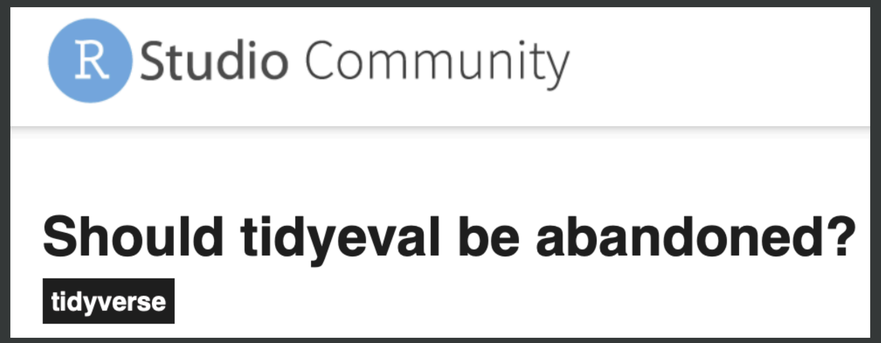
Jennifer Bryan

RStudio®

@jennybc

@JennyBryan

rstd.io/tidy-eval-context



What is tidy evaluation?

What is tidy evaluation?

Is it going to kill you or us?

hold on, wait ...

May have more bang bang for the buck than learning tidy eval:

1. How to write functions
2. Domain-specific tooling (maps, time series, etc.)
3. Lists, list-columns, nesting, unnesting
4. Functional programming with purrr
5. Scoped dplyr verbs, e.g. `mutate_at()`

What is tidy evaluation?

Tidy eval is:
a toolkit for metaprogramming in R.

Is something about the toolkit ***tidy***?

Yeah, I think so! But also ...

The tidyverse makes heavy use of metaprogramming, behind the scenes.

Tidy eval powers all of that.

```r
library(tidyverse)


starwars %>%
  filter(homeworld == "Tatooine") %>%
  arrange(height) %>%
  select(name, ends_with("color"))


ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point()
```

metaprogramming

≈

nonstandard evaluation (NSE)

≈

unquoted variable names

* this is technically WRONG but useful

To evaluate an expression, you
search environments for name-value bindings.

Nonstandard evaluation means you might:

- modify the expression first
- modify the chain of searched environments

Functions that accept unquoted variable names (+ an associated data frame) must implement NSE.

If you wrap such a function, you're obligated to deal with the NSE.

If you make direct specification of variable names extremely easy ...

it makes indirect specification harder.

Examples of indirect specification:

- names stored in an object
- names passed as function arguments

Is this challenge unique to the tidyverse?

No, it's present in base R as well.

```
lm(lifeExp ~ year, weights = pop, data = gapminder)

subset(gapminder, country == "Chad", select = year:pop)

transform(gapminder, GDP = gdpPercap * pop)

with(gapminder, lifeExp[country == "Chad" & year < 1980])
```

`?subset`, `?transform`, `?with`

⚠️ Warning ⚠️

This is a convenience function intended for use interactively. For programming it is better to use the standard subsetting functions like `[`, and in particular the non-standard evaluation of argument `subset` can have unanticipated consequences.

```
lm(lifeExp ~ poly(I(year - 1952), degree = 2))
```

$$\text{life expectancy} = \beta_0 + \beta_1 * \text{year} + \beta_2 * \text{year}^2 + \varepsilon$$

Want to fit model to each Gapminder country?

1. Wrap `lm()` in a function.
2. Drop into an iterative machine.

```
fit_fun <- function(df) {
  lm(lifeExp ~ poly(I(year - 1952), degree = 2), data = df)
}

by(gapminder, gapminder$country, fit_fun)
```

```r
fit_fun <- function(df) {
  lm(lifeExp ~ poly(I(year - 1952), degree = 2), data = df)
}

by(gapminder, gapminder$country, fit_fun)
#> gapminder$country: Afghanistan
#>
#> Call:
#> lm(formula = lifeExp ~ poly(I(year - 1952), degree = 2), data = df)
#>
#> Coefficients:
#>    (Intercept)  <blah>1  <blah>2
#>         37.479   16.462   -3.445
#> ------------------------------------------------------------
#> gapminder$country: Albania
#>
#> and so on ...
```

$$Y = \beta_0 + \beta_1 * X + \beta_2 * X^2 + \varepsilon$$

```
fit_fun <- function(df, y, x) {
  lm(y ~ poly(x, degree = 2), data = df)
}
```

```r
library(gapminder)

nope <- function(df, y, x) {
  lm(y ~ poly(x, degree = 2), data = df)
}

## will this work?
nope(gapminder, lifeExp, year)
#> Error in eval(predvars, data, env):
#> object 'year' not found

## do quotes help?
nope(gapminder, "lifeExp", "year")
#> Error in poly(x, degree = 2): 'degree'
#> must be less than number of unique points
```

This works, but 😬

```
wow <- function(df, y, x) {
  lm_formula <- substitute(
    y ~ poly(x, degree),
    list(y = substitute(y), x = substitute(x), degree = 2)
  )
  eval(lm(lm_formula, data = df))
}
```

Payoff: wow() is pleasant to use!

```
wow(gapminder,   y = lifeExp, x = year - 1952)

wow(gapminder, y = gdpPercap, x = year - 1952)

wow(gapminder,   y = lifeExp, x =   gdpPercap)
```

In base R, programming around NSE-using functions has been explicitly or implicitly discouraged.

The messy eval era

`ggplot2::aes_string()` vs. `aes()`

`dplyr::select_()` vs. `select()`

etc.

Not predictable for users

Not pleasant to maintain

Good news:

The tidyverse prioritizes usability, such as a data mask and unquoted variable names.

Bad news:

Programming around this is harder.

Good news:

We provide ourselves & you a toolkit for this.

rlang.r-lib.org

rlang provides the toolkit

but most people don't need to make direct use of rlang

What do you want to do?

I'll tell you how much tidy eval you need to know.

You want to:

Use existing tidyverse functions to analyze data.

You need to know this much tidy eval:

None. Congrats! Rock on.

You want to:

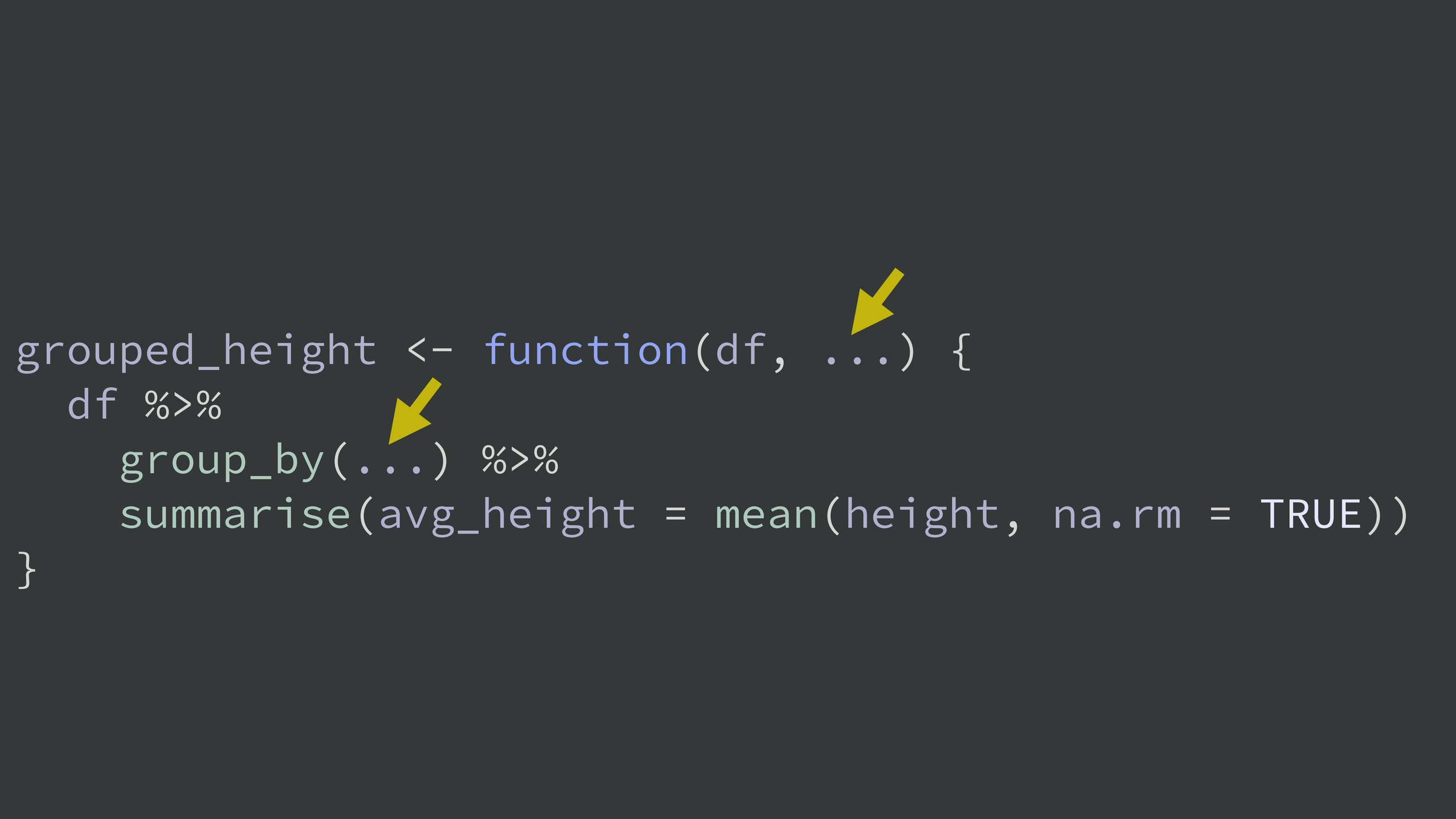
Write simple functions to reduce duplication.

You need to know this much tidy eval:

Perhaps none!

"Pass the dots".

You do not need rlang.


grouped_height <- function(df, ...) {
df %>%
group_by(...) %>%
summarise(avg_height = mean(height, na.rm = TRUE))
}

```
grouped_height(starwars, homeworld)
#> # A tibble: 49 x 2
#>    homeworld      avg_height
#>    <chr>               <dbl>
#>  1 <NA>                139.
#>  2 Alderaan            176.
#>  ...

grouped_height(starwars, species)
#> # A tibble: 38 x 2
#>    species   avg_height
#>    <chr>          <dbl>
#>  1 <NA>             160
#>  2 Aleena            79
#>  ...
```

You want to:

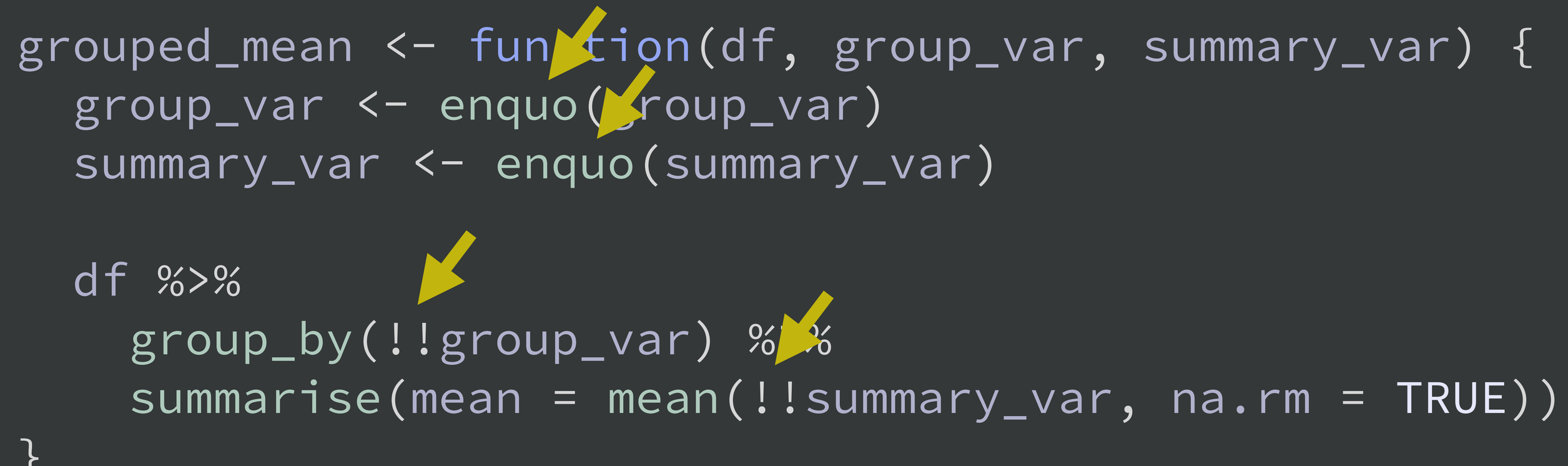
Write simple functions to reduce duplication.

You need to know this much tidy eval:

`enquo()` and `!!`

dplyr, ggplot2, and tidyr expose this.

You do not need rlang.


grouped_mean <- function(df, group_var, summary_var) {
group_var <- enquo(group_var)
summary_var <- enquo(summary_var)
df %>%
group_by(!!group_var) %>%
summarise(mean = mean(!!summary_var, na.rm = TRUE))
}

```
grouped_mean(starwars, homeworld, height)
#> # A tibble: 49 x 2
#>    homeworld        mean
#>    <chr>           <dbl>
#>  1 <NA>             139.
#>  2 Alderaan         176.
#>  3 ...

grouped_mean(starwars, homeworld, mass)
#> # A tibble: 49 x 2
#>    homeworld        mean
#>    <chr>           <dbl>
#>  1 <NA>               82
#>  2 Alderaan           64
#>  3 ...
```

You want to:

Write functions that make names from user input.

You need to know this much more tidy eval:

:=

dplyr, ggplot2, and tidyr expose this.

You do not need rlang.

You want to:

Compute on expressions & manipulate environments.

You need to know this much more tidy eval:

You do need to understand the theory.

You need rlang.

Helpful resources written by others:

Standard nonstandard evaluation rules
Thomas Lumley (2003)
http://developer.r-project.org/nonstandard-eval.pdf

Scoping Rules and NSE
Thomas Mailund
https://mailund.dk/posts/scoping-rules-and-nse/

Yet Another Introduction to Tidy Eval
Hiroaki Yutani
https://speakerdeck.com/yutannihilation/yet-another-introduction-to-tidyeval

Helpful resources from tidy eval creators:

Metaprogramming chapters of Advanced R, 2nd edition
Hadley Wickham
https://adv-r.hadley.nz/introduction-16.html

Tidy evaluation
Lionel Henry
https://tidyeval.tidyverse.org

RStudio community thread
https://community.rstudio.com/t/interesting-tidy-eval-use-cases

rstd.io/tidy-eval-context

Jennifer Bryan

RStudio®

@jennybc

@JennyBryan