

# Learning Tidy Evaluation by Reimplementing {dplyr}

Daniel Chen @chendaniely

Bureau of Labor Statistics

2021-02-09

hi!

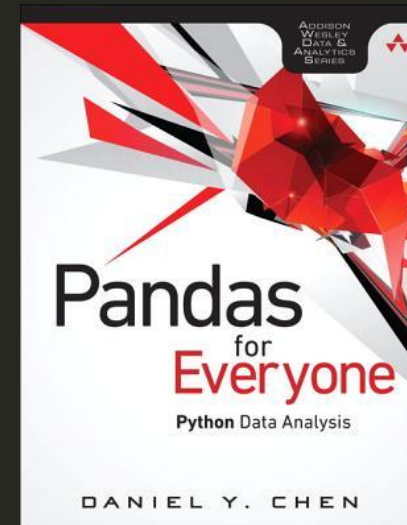
# I'm Daniel



PhD Student: Virginia Tech (Winter 2021)

- Data Science education & pedagogy
- Medical, Biomedical, Health Sciences
- [ds4biomed.tech](https://ds4biomed.tech)

- Inten at RStudio
  - [gradethis](https://gradethis.com)
  - Code grader for [learnr](https://learnr.rstudio.com) documents
- The Carpentries
  - Instructor
  - Trainer
  - Community Maintainer Lead
- Author:



# Why Tidy Evaluation

1. {tidyverse} compatible functions
2. Same **user** interface with different backends

But...

1. Easier for the user, harder for the developer.

# Parts of Tidy Evaluation

1. Quasiquotation
  - Quotation
2. Quosures
  - Quotation
  - Closures
  - Environments
3. Data mask
  - Environments

# Learning Tidy Evaluation

- Using `{dplyr}` as an example
  - Something we are familiar with
- Really only using `dplyr::select()`

# The Famous Penguin Dataset

```
library(palmerpenguins)
penguins
```

```
## # A tibble: 344 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##   <fct>    <fct>         <dbl>         <dbl>         <int>         <int>
## 1 Adelie  Torge~          39.1           18.7           181          3750
## 2 Adelie  Torge~          39.5           17.4           186          3800
## 3 Adelie  Torge~          40.3           18            195          3250
## 4 Adelie  Torge~          NA            NA            NA            NA
## 5 Adelie  Torge~          36.7           19.3           193          3450
## 6 Adelie  Torge~          39.3           20.6           190          3650
## 7 Adelie  Torge~          38.9           17.8           181          3625
## 8 Adelie  Torge~          39.2           19.6           195          4675
## 9 Adelie  Torge~          34.1           18.1           193          3475
## 10 Adelie Torge~          42            20.2           190          4250
## # ... with 334 more rows, and 2 more variables: sex <fct>, year <int>
```

# Selecting Columns



# Selecting columns: [row, col, drop]

```
head(penguins[, "species"]) # for tibble
```

```
## # A tibble: 6 x 1
##   species
##   <fct>
## 1 Adelie
## 2 Adelie
## 3 Adelie
## 4 Adelie
## 5 Adelie
## 6 Adelie
```

```
head(penguins[, c("species", "island")])
```

```
## # A tibble: 6 x 2
##   species island
##   <fct>   <fct>
## 1 Adelie Torgersen
## 2 Adelie Torgersen
## 3 Adelie Torgersen
## 4 Adelie Torgersen
## 5 Adelie Torgersen
## 6 Adelie Torgersen
```

```
head(penguins[, "species", drop = FALSE]) # for data.frame
```

```
## # A tibble: 6 x 1
##   species
##   <fct>
## 1 Adelie
## 2 Adelie
## 3 Adelie
## 4 Adelie
```

# Selecting columns: \$, drop=TRUE

```
penguins$species
```

```
penguins[, "species", drop = TRUE]
```

```
##      [1] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##      [8] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [15] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [22] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [29] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [36] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [43] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [50] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [57] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [64] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [71] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [78] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [85] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [92] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##     [99] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##    [106] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##    [113] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
##    [120] Adelie      Adelie      Adelie      Adelie      Adelie      Adelie      Adelie
```

# Selecting columns: `base::subset()`

```
base::subset(  
  penguins,  
  select = species)
```

```
## # A tibble: 344 x 1  
##   species  
##   <fct>  
## 1 Adelie  
## 2 Adelie  
## 3 Adelie  
## 4 Adelie  
## 5 Adelie  
## 6 Adelie  
## 7 Adelie  
## 8 Adelie  
## 9 Adelie  
## 10 Adelie  
## # ... with 334 more rows
```

```
base::subset(  
  x = penguins,  
  select = c(species, bill_length_mm))
```

```
## # A tibble: 344 x 2  
##   species bill_length_mm  
##   <fct>         <dbl>  
## 1 Adelie         39.1  
## 2 Adelie         39.5  
## 3 Adelie         40.3  
## 4 Adelie         NA  
## 5 Adelie         36.7  
## 6 Adelie         39.3  
## 7 Adelie         38.9  
## 8 Adelie         39.2  
## 9 Adelie         34.1  
## 10 Adelie         42  
## # ... with 334 more rows
```

# Selecting columns: `dplyr::select()`

```
dplyr::select(penguins, species)
```

```
## # A tibble: 344 x 1
##   species
##   <fct>
## 1 Adelie
## 2 Adelie
## 3 Adelie
## 4 Adelie
## 5 Adelie
## 6 Adelie
## 7 Adelie
## 8 Adelie
## 9 Adelie
## 10 Adelie
## # ... with 334 more rows
```

```
penguins %>%
  dplyr::select(species, bill_length_mm)
```

```
## # A tibble: 344 x 2
##   species bill_length_mm
##   <fct>         <dbl>
## 1 Adelie         39.1
## 2 Adelie         39.5
## 3 Adelie         40.3
## 4 Adelie         NA
## 5 Adelie         36.7
## 6 Adelie         39.3
## 7 Adelie         38.9
## 8 Adelie         39.2
## 9 Adelie         34.1
## 10 Adelie         42
## # ... with 334 more rows
```

# Selecting columns: index position

```
penguins[, 1, drop = FALSE]
```

```
## # A tibble: 344 x 1
##   species
##   <fct>
## 1 Adelie
## 2 Adelie
## 3 Adelie
## 4 Adelie
## 5 Adelie
## 6 Adelie
## 7 Adelie
## 8 Adelie
## 9 Adelie
## 10 Adelie
## # ... with 334 more rows
```

```
penguins[, c(1, 3, 5)]
```

```
## # A tibble: 344 x 3
##   species bill_length_mm flipper_length_mm
##   <fct>         <dbl>         <int>
## 1 Adelie         39.1           181
## 2 Adelie         39.5           186
## 3 Adelie         40.3           195
## 4 Adelie         NA             NA
## 5 Adelie         36.7           193
## 6 Adelie         39.3           190
## 7 Adelie         38.9           181
## 8 Adelie         39.2           195
## 9 Adelie         34.1           193
## 10 Adelie         42            190
## # ... with 334 more rows
```

# Quasiquotation

What is quoting?

# What is an expression

The code you write that R interprets and evaluates

```
3 + 3
```

```
## [1] 6
```

Only code you write

```
quote(3 + 3)
```

```
## 3 + 3
```

Think of quoting as the string representation of your code. It's not really a string, but it's a reasonable approximation.

Lazy evaluation: the 3+3 isn't evaluated right away

```
ex <- quote(3 + 3)  
ex
```

```
## 3 + 3
```

```
eval(ex)
```

```
## [1] 6
```

# Using {rlang}

```
rlang::expr(3 + 3)
```

```
## 3 + 3
```

```
ex <- rlang::expr(3 + 3)  
ex
```

```
## 3 + 3
```

```
eval(ex)
```

```
## [1] 6
```

```
rlang::eval_tidy(ex)
```

```
## [1] 6
```



# Expressions: call, symbol, constant, pairlist

```
ex <- quote(3 + 3)
str(ex)
```

```
## language 3 + 3
```

```
ex <- quote(3)
str(ex)
```

```
## num 3
```

```
ex <- quote(species)
str(ex)
```

```
## symbol species
```

# Selecting columns

Direct string column

```
penguins[, "species"]
```

```
## # A tibble: 344 x 1
##   species
##   <fct>
## 1 Adelie
## 2 Adelie
## 3 Adelie
## 4 Adelie
## 5 Adelie
## 6 Adelie
## 7 Adelie
## 8 Adelie
## 9 Adelie
## 10 Adelie
## # ... with 334 more rows
```

Passing a variable

```
col <- "species"
penguins[, col]
```

```
## # A tibble: 344 x 1
##   species
##   <fct>
## 1 Adelie
## 2 Adelie
## 3 Adelie
## 4 Adelie
## 5 Adelie
## 6 Adelie
## 7 Adelie
## 8 Adelie
## 9 Adelie
## 10 Adelie
## # ... with 334 more rows
```

# Selecting columns: Variables need to exist

```
penguins[, species]
```

```
## Error in `[.tbl_df`(penguins, , species): object 'species' not found
```

# Selecting: tibble specific (tibble)

```
as.name("species")
```

```
## species
```

```
penguins[, as.name("species")]
```

```
## # A tibble: 344 x 1
##   species
##   <fct>
## 1 Adelie
## 2 Adelie
## 3 Adelie
## 4 Adelie
## 5 Adelie
## 6 Adelie
## 7 Adelie
## 8 Adelie
## 9 Adelie
## 10 Adelie
## # ... with 334 more rows
```

```
quote(species)
```

```
## species
```

```
penguins[, quote(species)]
```

```
## # A tibble: 344 x 1
##   species
##   <fct>
## 1 Adelie
## 2 Adelie
## 3 Adelie
## 4 Adelie
## 5 Adelie
## 6 Adelie
## 7 Adelie
## 8 Adelie
## 9 Adelie
## 10 Adelie
## # ... with 334 more rows
```

# Selecting: tibble specific (data.frame)

```
iris[, as.name("Species")]
```

```
## Error in .subset(x, j): invalid subscript type 'symbol'
```

```
iris[, quote(Species)]
```

```
## Error in .subset(x, j): invalid subscript type 'symbol'
```

# my\_select: try 1

```
my_select <- function(data, col) {  
  return(  
    data[, col, drop = FALSE]  
  )  
}
```

```
my_select(penguins, "species")
```

```
## # A tibble: 344 x 1  
##   species  
##   <fct>  
## 1 Adelie  
## 2 Adelie  
## 3 Adelie  
## 4 Adelie  
## 5 Adelie  
## 6 Adelie  
## 7 Adelie  
## 8 Adelie  
## 9 Adelie  
## 10 Adelie
```

```
# remeber this is tibble input specific  
my_select(penguins, quote(species))
```

```
## # A tibble: 344 x 1  
##   species  
##   <fct>  
## 1 Adelie  
## 2 Adelie  
## 3 Adelie  
## 4 Adelie  
## 5 Adelie  
## 6 Adelie  
## 7 Adelie  
## 8 Adelie  
## 9 Adelie  
## 10 Adelie  
## # ... with 334 more rows
```

# my\_select: try 1 needs to quote

```
my_select(penguins, species)
```

```
## Error in `[.tbl_df`(data, , col, drop = FALSE): object 'species' not found
```

# my\_select: try 2

Oh! I just learned how to quote inputs!

```
my_select <- function(data, col) {  
  return(  
    data[, quote(col), drop = FALSE]  
  )  
}
```

Nope!

We need a way to capture what the user passed, not the function parameter name.

```
my_select(penguins, species) # quote passed in col, not species
```

```
## Error: Can't subset columns that don't exist.  
## x Column `col` doesn't exist.
```



# {rlang} enriched expression

Use `rlang::expr()` to capture expressions outside of a function

Use `rlang::enexpr()` to capture expression **inside** a function

```
ex <- rlang::expr(x)
ex
```

```
## x
```

```
fexpr <- function(x) {
  rlang::expr(x)
}
fexpr(hello)
```

```
## x
```

```
fenexpr <- function(x) {
  rlang::enexpr(x)
}
fenexpr(hello)
```

```
## hello
```

# my\_select: try 3

```
my_select <- function(data, col) {  
  return(  
    data[, rlang::enexpr(col), drop = FALSE]  
  )  
}
```

```
my_select(penguins, species)
```

```
## # A tibble: 344 x 1  
##   species  
##   <fct>  
## 1 Adelie  
## 2 Adelie  
## 3 Adelie  
## 4 Adelie  
## 5 Adelie  
## 6 Adelie  
## 7 Adelie  
## 8 Adelie  
## 9 Adelie  
## 10 Adelie
```

```
my_select <- function(data, col) {  
  col <- rlang::enexpr(col)  
  return(  
    data[, col, drop = FALSE]  
  )  
}
```

```
my_select(penguins, species)
```

```
## # A tibble: 344 x 1  
##   species  
##   <fct>  
## 1 Adelie  
## 2 Adelie  
## 3 Adelie  
## 4 Adelie  
## 5 Adelie  
## 6 Adelie  
## 7 Adelie  
## 8 Adelie  
## 9 Adelie
```

# my\_select: try 3 on data.frame

```
my_select(iris, Species)
```

```
## Error in .subset(x, j): invalid subscript type 'symbol'
```

**Remember how we can select on index positions?**

# my\_select: try 4

```
my_select <- function(data, col) {  
  col <- rlang::enexpr(col)  
  idx <- which(names(data) %in% as.character(col)) # create an index  
  return(  
    data[, idx, drop = FALSE] # subset on the index  
  )  
}
```

Works on a tibble

```
my_select(penguins, species)
```

```
## # A tibble: 344 x 1  
##   species  
##   <fct>  
## 1 Adelie  
## 2 Adelie  
## 3 Adelie  
## 4 Adelie  
## 5 Adelie  
## 6 Adelie
```

Works on a data.frame

```
my_select(iris, Species)
```

```
##      Species  
## 1      setosa  
## 2      setosa  
## 3      setosa  
## 4      setosa  
## 5      setosa  
## 6      setosa  
## 7      setosa  
## 8      setosa
```

# dplyr::select source code

```
select <- function(.data, ...) {  
  UseMethod("select")  
}
```

```
select.data.frame <- function(.data, ...) {  
  loc <- tidyselect::eval_select(expr(c(...)), .data)  
  loc <- ensure_group_vars(loc, .data, notify = TRUE)  
  dplyr_col_select(.data, loc, names(loc))  
}
```

```
dplyr_col_select <- function(.data, loc, names = NULL) {  
  loc <- vec_as_location(loc, n = ncol(.data), na.rm = TRUE)  
  ...  
  ...  
}
```

```
ensure_group_vars <- function(loc, data, notify = FALSE) {  
  group_loc <- match(group_vars(data), names(data))  
  missing <- setdiff(group_loc, loc)  
  
  if (length(missing) > 0) {  
    vars <- names(data)[missing]  
    if (notify) {  
      inform(glue(  
        "Adding missing grouping variables: ",  
        paste0("`", names(data)[missing], "`", collapse = ", ")  
      ))  
    }  
    loc <- c(set_names(missing, vars), loc)  
  }  
  loc  
}
```

# my\_select: try 5: more variables . . .

enexpr for a single variable, enexprs for multiple variables

```
my_select <- function(data, ...) {  
  cols <- rlang::enexprs(...) # the "s" for plural = multiple things  
  cols_char <- as.vector(cols, mode = "character")  
  idx <- which(names(data) %in% cols_char)  
  return(  
    data[, idx, drop = FALSE]  
  )  
}
```

```
my_select(penguins, species, year, island)
```

```
## # A tibble: 344 x 3  
##   species island   year  
##   <fct>   <fct>   <int>  
## 1 Adelie  Torgersen  2007  
## 2 Adelie  Torgersen  2007  
## 3 Adelie  Torgersen  2007  
## 4 Adelie  Torgersen  2007  
## 5 Adelie  Torgersen  2007
```

```
my_select(iris, Species, Petal.Width)
```

```
##   Petal.Width Species  
## 1         0.2   setosa  
## 2         0.2   setosa  
## 3         0.2   setosa  
## 4         0.2   setosa  
## 5         0.2   setosa  
## 6         0.4   setosa  
## 7         0.3   setosa
```

# my\_select: try 6: match not which

```
my_select <- function(data, ...) {  
  cols <- rlang::enexprs(...)  
  cols_char <- as.vector(cols, mode = "character")  
  idx <- match(cols_char, names(data))  
  return(  
    data[, idx, drop = FALSE]  
  )  
}
```

```
my_select(penguins, species, year, island)
```

```
## # A tibble: 344 x 3  
##   species year island  
##   <fct>   <int> <fct>  
## 1 Adelie  2007 Torgersen  
## 2 Adelie  2007 Torgersen  
## 3 Adelie  2007 Torgersen  
## 4 Adelie  2007 Torgersen  
## 5 Adelie  2007 Torgersen  
## 6 Adelie  2007 Torgersen  
## 7 Adelie  2007 Torgersen
```

```
my_select(iris, Species, Petal.Width)
```

```
##   Species Petal.Width  
## 1 setosa      0.2  
## 2 setosa      0.2  
## 3 setosa      0.2  
## 4 setosa      0.2  
## 5 setosa      0.2  
## 6 setosa      0.4  
## 7 setosa      0.3  
## 8 setosa      0.2  
## 9 setosa      0.2
```

# Quasiquotation: ...-quote-unquote-quote...

In the last try we had:

```
my_select <- function(data, ...) {  
  cols <- rlang::enexprs(...)           # handle the "weirdness"  
  cols_char <- as.vector(cols, mode = "character") # unquote the arguments  
  idx <- match(cols_char, names(data))    # do regular R things  
  return(  
    data[, idx, drop = FALSE]  
  )  
}
```



# Variables, quoted, unquoted

```
col_name <- "species"
```

```
penguins[, c(col_name, "island", "year")]
```

```
## # A tibble: 344 x 3
##   species island    year
##   <fct>   <fct>    <int>
## 1 Adelie  Torgersen  2007
## 2 Adelie  Torgersen  2007
## 3 Adelie  Torgersen  2007
## 4 Adelie  Torgersen  2007
## 5 Adelie  Torgersen  2007
## 6 Adelie  Torgersen  2007
## 7 Adelie  Torgersen  2007
## 8 Adelie  Torgersen  2007
## 9 Adelie  Torgersen  2007
## 10 Adelie Torgersen  2007
## # ... with 334 more rows
```

```
penguins %>%
  dplyr::select(col_name, island, year)
```

```
## Note: Using an external vector in selections is ambiguous
## i Use `all_of(col_name)` instead of `col_name` to silence
## i See <https://tidyselect.r-lib.org/reference/faq-external-objects.html>
## This message is displayed once per session.
```

```
## # A tibble: 344 x 3
##   species island    year
##   <fct>   <fct>    <int>
## 1 Adelie  Torgersen  2007
## 2 Adelie  Torgersen  2007
## 3 Adelie  Torgersen  2007
## 4 Adelie  Torgersen  2007
## 5 Adelie  Torgersen  2007
## 6 Adelie  Torgersen  2007
## 7 Adelie  Torgersen  2007
## 8 Adelie  Torgersen  2007
```

# Variables, quoted, unquoted: my\_select

Subsets quoted variables

```
my_select(penguins, island, year)
```

```
## # A tibble: 344 x 2
##   island      year
##   <fct>      <int>
## 1 Torgersen  2007
## 2 Torgersen  2007
## 3 Torgersen  2007
## 4 Torgersen  2007
## 5 Torgersen  2007
## 6 Torgersen  2007
## 7 Torgersen  2007
## 8 Torgersen  2007
## 9 Torgersen  2007
## 10 Torgersen 2007
## # ... with 334 more rows
```

Does not work for unquoted variables.

We want `col_name` to be `species`

```
my_select(penguins, col_name, island, year)
```

```
## Error: Can't use NA as column index with `[` at position 1.
```

# The problem

Need a way to treat the variable as the variable not as the quoted term

## That is...

- Want to unquote `col_name` since the input is automatically quoted using `enexprs`
- Want to replace `col_name` with `species`

## Solution

For a single variable we use `!!col_name` to unquote `col_name`

# bang-bang !!

```
col_name <- "species"
```

```
penguins %>%  
  my_select("island", "year")
```

```
## # A tibble: 344 x 2  
##   island      year  
##   <fct>    <int>  
## 1 Torgersen  2007  
## 2 Torgersen  2007  
## 3 Torgersen  2007  
## 4 Torgersen  2007  
## 5 Torgersen  2007  
## 6 Torgersen  2007  
## 7 Torgersen  2007  
## 8 Torgersen  2007  
## 9 Torgersen  2007  
## 10 Torgersen 2007  
## # ... with 334 more rows
```

```
penguins %>%  
  my_select(!col_name, island, "year")
```

```
## # A tibble: 344 x 3  
##   species island      year  
##   <fct>    <fct>    <int>  
## 1 Adelie   Torgersen  2007  
## 2 Adelie   Torgersen  2007  
## 3 Adelie   Torgersen  2007  
## 4 Adelie   Torgersen  2007  
## 5 Adelie   Torgersen  2007  
## 6 Adelie   Torgersen  2007  
## 7 Adelie   Torgersen  2007  
## 8 Adelie   Torgersen  2007  
## 9 Adelie   Torgersen  2007  
## 10 Adelie   Torgersen  2007  
## # ... with 334 more rows
```

# How to know if a function quotes inputs?

If you pass in the arguments into the function and it works

```
my_select(penguins, island) %>% head(3)
```

```
## # A tibble: 3 x 1
##   island
##   <fct>
## 1 Torgersen
## 2 Torgersen
## 3 Torgersen
```

```
library(rlang)
```

but if you pass the argument outside the function and it fails

```
island
```

```
## Error in eval(expr, envir, enclos): object 'island' not found
```

**!! unquoting: selective evaluation on parts of a quoted expression**

!!! is !! for ...



!!!

```
cols <- exprs(species, island, year)
cols
```

```
## [[1]]
## species
##
## [[2]]
## island
##
## [[3]]
## year
```

```
class(cols)
```

```
## [1] "list"
```

```
my_select(penguins, cols)
```

```
## Error: Can't use NA as column index with `[` at position 1.
```

```
my_select(penguins, !!!cols)
```

```
## # A tibble: 344 x 3
##   species island    year
##   <fct>   <fct>   <int>
## 1 Adelie  Torgersen  2007
## 2 Adelie  Torgersen  2007
## 3 Adelie  Torgersen  2007
## 4 Adelie  Torgersen  2007
## 5 Adelie  Torgersen  2007
## 6 Adelie  Torgersen  2007
## 7 Adelie  Torgersen  2007
## 8 Adelie  Torgersen  2007
## 9 Adelie  Torgersen  2007
## 10 Adelie Torgersen  2007
## # ... with 334 more rows
```

# {dplyr} has one more check

```
col_name <- "species"
```

Bare variable name

```
penguins %>%  
  dplyr::select(col_name, island, year)
```

```
## # A tibble: 344 x 3  
##   species island    year  
##   <fct>   <fct>    <int>  
## 1 Adelie  Torgersen  2007  
## 2 Adelie  Torgersen  2007  
## 3 Adelie  Torgersen  2007  
## 4 Adelie  Torgersen  2007  
## 5 Adelie  Torgersen  2007  
## 6 Adelie  Torgersen  2007  
## 7 Adelie  Torgersen  2007  
## 8 Adelie  Torgersen  2007  
## 9 Adelie  Torgersen  2007  
## 10 Adelie Torgersen  2007  
## #       with 334 more rows
```

Using !! to unquote the variable name

```
penguins %>%  
  dplyr::select(!!col_name, island, year)
```

```
## # A tibble: 344 x 3  
##   species island    year  
##   <fct>   <fct>    <int>  
## 1 Adelie  Torgersen  2007  
## 2 Adelie  Torgersen  2007  
## 3 Adelie  Torgersen  2007  
## 4 Adelie  Torgersen  2007  
## 5 Adelie  Torgersen  2007  
## 6 Adelie  Torgersen  2007  
## 7 Adelie  Torgersen  2007  
## 8 Adelie  Torgersen  2007  
## 9 Adelie  Torgersen  2007  
## 10 Adelie Torgersen  2007  
## #       with 334 more rows
```



**Quosures = quote + closure**

# Closure?!



**Danielle Navarro**

@djnavarro

Following



Roses are red,  
Iris data forgettable.  
Objects of type closure,  
Are not subsettable

#rstats valentine

5:57 AM - 13 Feb 2019

110 Retweets 598 Likes



11



110



598



```
df[1]
```

```
## Error in df[1]: object of type 'closure' is not subsettable
```

`df()` is actually a function in `stats::df()`. You can't subset a function.

- Closure = "thing" (e.g., a function, expression) + environment

# Environments

```
e <- new.env()  
e$x <- 3
```

```
e$x
```

```
## [1] 3
```

```
x
```

```
## Error in eval(expr, envir, enclos): object 'x' not found
```

```
eval(quote(3 + x))
```

```
## Error in eval(quote(3 + x)): object 'x' not found
```

```
eval(quote(3 + x), envir = e)
```

```
## [1] 6
```

# Formulas aren't just for models

<pre>~ 3 + 3</pre>	<p>Extract parts of the formula</p> <pre>form[[1]]</pre>
<pre>## ~3 + 3</pre>	<pre>## `~`</pre>
<pre>form &lt;- ~ 3 + 3 form</pre>	<pre>form[[2]]</pre>
<pre>## ~3 + 3</pre>	<pre>## 3 + 3</pre>
<pre>attributes(form)</pre>	<p>Evaluate the expression of the formula</p>
<pre>## \$class ## [1] "formula" ## ## \$.Environment ## &lt;environment: R_GlobalEnv&gt;</pre>	<pre>eval(form[[2]])</pre> <pre>## [1] 6</pre>

# Formulas = expression + environment

```
form <- ~ 3 + x  
form[[1]]
```

```
## `~`
```

```
form[[2]]
```

```
## 3 + x
```

```
environment(form)
```

```
## <environment: R_GlobalEnv>
```

```
e <- new.env()  
e$x <- 10  
environment(form) <- e
```

```
eval(expr = form[[2]],  
      envir = environment(form))
```

```
## [1] 13
```

```
eval(expr = form[[2]]) # no x in .GlobalEnv
```

```
## Error in eval(expr = form[[2]]): object 'x' not found
```

# Quosure = expression + environment

- We can program with quosures easier than ~
  - Allows quasiquotation
  - User facing: ~
  - Developer facing: quosures

```
form <- ~ 3 + x
e <- rlang::env(x = 10)
environment(form) <- e
```

```
eval(expr = form[[2]],
      envir = environment(form))
```

```
## [1] 13
```

```
q <- rlang::new_quosure(
  expr = rlang::expr(3 + x),
  env = e)
```

```
rlang::eval_tidy(q) # uses the quosure env
```

```
## [1] 13
```

---

Quosures are subclass of formulas

```
class(q) # subclass of formula
```

```
## [1] "quosure" "formula"
```

# Quosures in practice

- In practice we use `enquo()` and `enquos()` within a function definition
  - Remember the `en-` prefix for "enriched" which does the quoting from function arguments
- `quo()`, `quos()`, and `new_quosure()` exist for completeness

# Data Masks



# Data mask

- An object (e.g., usually a dataframe, but can also be a list) where the expression goes to look for values
- Data mask values **superceed** values in the environment

# Quosure + Data Mask example

```
q1 <- rlang::new_quosure(rlang::expr(x * y),  
                        rlang::env(x = 100))  
q1
```

```
## <quosure>  
## expr: ^x * y  
## env: 000000001DE04E60
```

```
df <- data.frame(y = 1:5)  
df
```

```
##   y  
## 1 1  
## 2 2  
## 3 3  
## 4 4  
## 5 5
```

```
rlang::eval_tidy(expr = q1,  
                 data = df)
```

```
## [1] 100 200 300 400 500
```

```
# uses the quosure's env
```

---

```
x * y
```

```
## Error in eval(expr, envir, enclos): object 'x' not found
```

```
100 * y
```

```
## Error in eval(expr, envir, enclos): object 'y' not found
```

```
100 * df$y
```

```
## [1] 100 200 300 400 500
```

# my\_select: previous try

```
col_name <- "species"
```

```
my_select <- function(data, ...) {  
  cols <- rlang::enexprs(...)           # handle the "weirdness"  
  cols_char <- as.vector(cols, mode = "character") # unquote the arguments  
  idx <- match(cols_char, names(data))    # find index positions  
  return(  
    data[, idx, drop = FALSE]           # regular R things: subset on index  
  )  
}
```

```
my_select(penguins, col_name, year, "island")
```

```
## Error: Can't use NA as column index with `[` at position 1.
```

# my\_select:try 7 quosures + data mask

```
my_select <- function(data, ...) {  
  cols <- rlang::enquos(...) # handle the "weirdness"  
  
  vars <- as.list(set_names(seq_along(data), names(data))) # list of columns and their index  
  col_char_num <- purrr::map(cols, rlang::eval_tidy, vars) # tidy evaluate the user inputs  
  idx <- purrr::map_int(col_char_num, # get index based on user input cols  
    function(x){ifelse(is.character(x),  
                        vars[[x]],  
                        x)}})  
  
  return(  
    data[, idx, drop = FALSE] # regular R things: subset on index  
  )  
}
```

```
my_select(penguins, col_name, year, "island")
```

```
## # A tibble: 344 x 3  
##   species year island  
##   <fct>   <int> <fct>  
## 1 Adelie   2007 Torgersen  
## 2 Adelie   2007 Torgersen  
## 3 Adelie   2007 Torgersen
```

# In general...

```
dplyr::select(), dplyr::arrange(), { tidyselect }
```

- Getting the index position of the columns

```
idx <- c(1, 3, 2, 6, 1)
```

- Subsetting using base R on those index positions

```
penguins[, idx, drop = FALSE]
```

```
## # A tibble: 344 x 5
##   species bill_length_mm island   body_mass_g species
##   <fct>         <dbl> <fct>         <int> <fct>
## 1 Adelie         39.1 Torgersen         3750 Adelie
## 2 Adelie         39.5 Torgersen         3800 Adelie
## 3 Adelie         40.3 Torgersen         3250 Adelie
## 4 Adelie         NA   Torgersen          NA Adelie
## 5 Adelie         36.7 Torgersen         3450 Adelie
## 6 Adelie         39.3 Torgersen         3650 Adelie
## 7 Adelie         38.9 Torgersen         3625 Adelie
## 8 Adelie         39.2 Torgersen         4675 Adelie
## 9 Adelie         34.1 Torgersen         3475 Adelie
```

# What I didn't cover

**A lot...**

# For example...

- `:=` "colon equal", let's you quote the left hand side of an equal sign
- `.data` and `.env` pronouns in a data mask
  - <https://adv-r.hadley.nz/evaluation.html#pronouns>



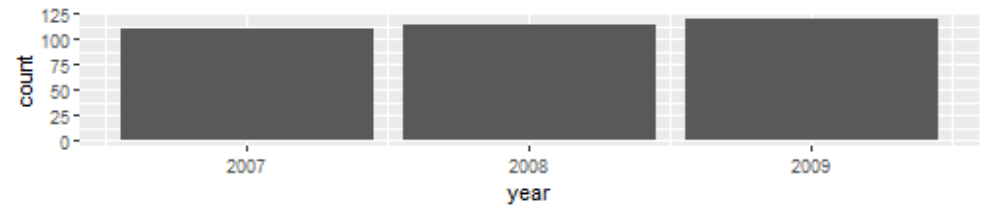
**tl;dr**

# User: string; Function: string

- What we are used to
- Use the string version or string parameter

```
my_func <- function(data, col) {  
  return(  
    ggplot(data = data,  
            aes_string(x = col)) +  
    geom_bar()  
  )  
}
```

```
my_func(penguins, "year")
```

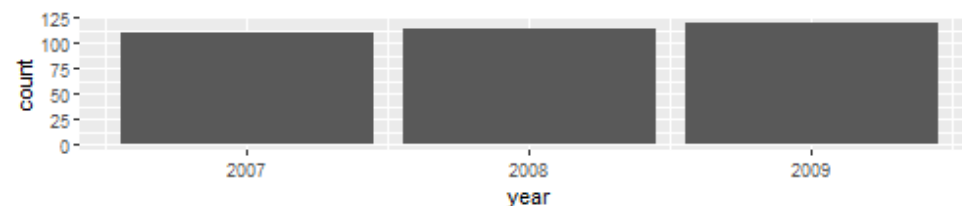


# User: string; Function: quote

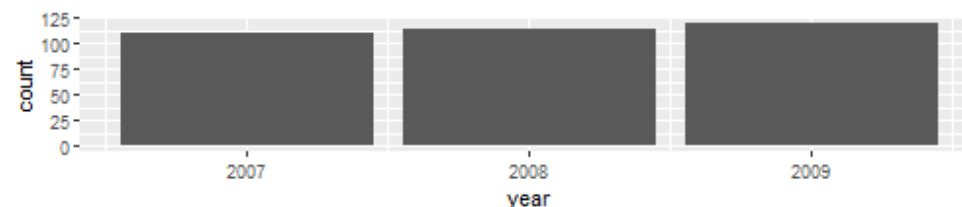
1. Convert to expression
  - `base::parse()`
  - `rlang::parse_expr()`, `rlang::parse_exprs()`, `rlang::parse_quo()`, `rlang::parse_quos()`
2. Unquote expression

```
my_func <- function(data, col) {  
  ex <- rlang::parse_expr(col)  
  return(  
    ggplot(data = data,  
      aes(x = !!ex)) +  
    geom_bar()  
  )  
}
```

```
my_func(penguins, "year")
```



```
var <- "year"  
my_func(penguins, var)
```

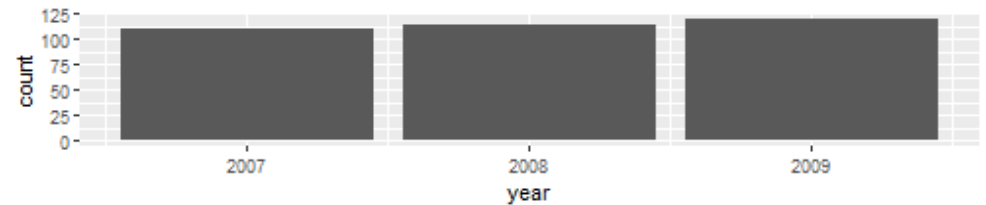


# User: quote; Function: string

- Capture expression: `rlang::enexpr()`
- Convert to string: `rlang::as_string()`

```
my_func <- function(data, col) {  
  q <- rlang::enexpr(col)  
  s <- rlang::as_string(q)  
  return(  
    ggplot(data = data,  
      aes_string(x = s)) +  
    geom_bar()  
  )  
}
```

```
my_func(penguins, year)
```



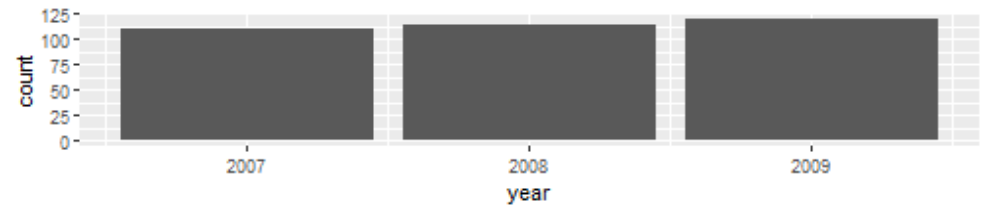
# User: quote; Function: quote

- `rlang::enquo()` + `!!`
- `{{ }}`

```
my_func <- function(data, col) {  
  col_quo <- rlang::enquo(col)  
  return(  
    ggplot(data = data,  
      aes(x = !!col_quo )) +  
    geom_bar()  
  )  
}
```

```
my_func <- function(data, col) {  
  return(  
    ggplot(data = data,  
      aes(x = {{col}} )) +  
    geom_bar()  
  )  
}
```

```
my_func(penguins, year)
```



# Thanks!

@chendaniely

Read, read, and re-read:: <https://adv-r.hadley.nz/metaprogramming.html>

Slides: <https://github.com/chendaniely/2021-02-09-bls-tidyeval>