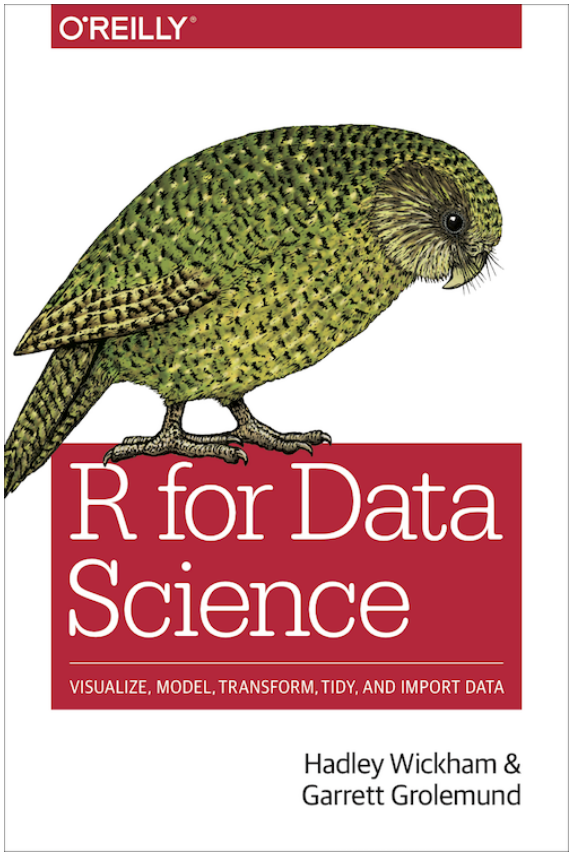
Socio-Informatics 348

Program
Functions

Dr Lisa Martin

Department of Information Science
Stellenbosch University

Today's Reading



R for Data Science, Chapter 25

Functions

- Functions are a way to organise and reuse code
- Benefits over copy-pasting code:
  - As requirements change, you only need to update code in one place, instead of many.
  - You eliminate the chance of making incidental mistakes when you copy and paste
  - It makes it easier to reuse work from project-to-project, increasing your productivity over time.
  - You can give a function an evocative name that makes your code easier to understand.

Types of Functions

- Vector functions take one or more vectors as input and return a vector as output.
- Data frame functions take a data frame as input and return a data frame as output.
- Plot functions that take a data frame as input and return a plot as output.

Vector Functions

Example:

```r
df <- tibble(
  a = rnorm(5),
  b = rnorm(5),
  c = rnorm(5),
  d = rnorm(5),
)

df |> mutate(
  a = (a - min(a, na.rm = TRUE)) /
    (max(a, na.rm = TRUE) - min(a, na.rm = TRUE)),
  b = (b - min(a, na.rm = TRUE)) /
    (max(b, na.rm = TRUE) - min(b, na.rm = TRUE)),
  c = (c - min(c, na.rm = TRUE)) /
    (max(c, na.rm = TRUE) - min(c, na.rm = TRUE)),
  d = (d - min(d, na.rm = TRUE)) /
    (max(d, na.rm = TRUE) - min(d, na.rm = TRUE)),
)
```

```
#> # A tibble: 5 × 4
#>       a       b     c     d
#>   <dbl>   <dbl> <dbl> <dbl>
#> 1 0.339  0.387  0.291 0
#> 2 0.880 -0.613  0.611 0.557
#> 3 0     -0.0833 1     0.752
#> 4 0.795 -0.0822 0     1
#> 5 1     -0.0952 0.580 0.394
```

Vector Functions

What is being repeated?

```
(a - min(a, na.rm = TRUE)) / (max(a, na.rm = TRUE) - min(a, na.rm = TRUE))
(b - min(b, na.rm = TRUE)) / (max(b, na.rm = TRUE) - min(b, na.rm = TRUE))
(c - min(c, na.rm = TRUE)) / (max(c, na.rm = TRUE) - min(c, na.rm = TRUE))
(d - min(d, na.rm = TRUE)) / (max(d, na.rm = TRUE) - min(d, na.rm = TRUE))
```

What is changing each time?

```
(_ - min(_, na.rm = TRUE)) / (max(_, na.rm = TRUE) - min(_, na.rm = TRUE))
```

Vector Functions

Parts of a function:

- Name
- Arguments (inputs)
- Body (code that does the work)

```
name <- function(arguments) {
  body
}
```

With our example:

```
rescale01 <- function(x) {
  (x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
}
```

Vector Functions

Using the function:

```
df |> mutate(
  a = rescale01(a),
  b = rescale01(b),
  c = rescale01(c),
  d = rescale01(d),
)
#> # A tibble: 5 × 4
#>       a     b     c     d
#>   <dbl> <dbl> <dbl> <dbl>
#> 1 0.339 1     0.291 0
#> 2 0.880 0     0.611 0.557
#> 3 0     0.530 1     0.752
#> 4 0.795 0.531 0     1
#> 5 1     0.518 0.580 0.394
```

Vector Functions

Improving the function:

```
rescale01 <- function(x) {
  rng <- range(x, na.rm = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}
```

```
x <- c(1:10, Inf)
rescale01(x)
#>  [1]   0   0   0   0   0   0   0   0   0   0 NaN
```

```
rescale01 <- function(x) {
  rng <- range(x, na.rm = TRUE, finite = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}

rescale01(x)
#>  [1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667
#>  [8] 0.7777778 0.8888889 1.0000000       Inf
```

Vector Functions

Mutate vs Summarise functions:

Mutate functions return a vector of the same length as the input vectors.

```r
z_score <- function(x) {
  (x - mean(x, na.rm = TRUE)) / sd(x, na.rm = TRUE)
}
```

```r
clamp <- function(x, min, max) {
  case_when(
    x < min ~ min,
    x > max ~ max,
    .default = x
  )
}

clamp(1:10, min = 3, max = 7)
#>  [1] 3 3 3 4 5 6 7 7 7 7
```

Vector Functions

Mutate vs Summarise functions:

Summarise functions return a vector of length 1.

```
commas <- function(x) {
  str_flatten(x, collapse = ", ", last = " and ")
}

commas(c("cat", "dog", "pigeon"))
#> [1] "cat, dog and pigeon"
```

```
cv <- function(x, na.rm = FALSE) {
  sd(x, na.rm = na.rm) / mean(x, na.rm = na.rm)
}

cv(runif(100, min = 0, max = 50))
#> [1] 0.5196276
cv(runif(100, min = 0, max = 500))
#> [1] 0.5652554
```

Dataframe Functions

- Data frame functions take a data frame as input and return a data frame as output.
- They are often used to encapsulate a series of data transformation steps.
- Problem with functions and dplyr: **indirection**

Dataframe Functions

```r
grouped_mean <- function(df, group_var, mean_var) {
  df |>
    group_by(group_var) |>
    summarize(mean(mean_var))
}
```

```r
diamonds |> grouped_mean(cut, carat)
#> Error in `group_by()`:
#> ! Must group by variables found in `.data`.
#> ✖ Column `group_var` is not found.
```

Dataframe Functions

Solution: Embracing

```
df <- tibble(
  mean_var = 1,
  group_var = "g",
  group = 1,
  x = 10,
  y = 100
)

df |> grouped_mean(group, x)
#> # A tibble: 1 × 2
#>   group_var `mean(mean_var)`
#>   <chr>                <dbl>
#> 1 g                        1
df |> grouped_mean(group, y)
#> # A tibble: 1 × 2
#>   group_var `mean(mean_var)`
#>   <chr>                <dbl>
#> 1 g                        1
```

Dataframe Functions

```
grouped_mean <- function(df, group_var, mean_var) {
  df |>
    group_by({{ group_var }}) |>
    summarize(mean({{ mean_var }}))
}

df |> grouped_mean(group, x)
#> # A tibble: 1 × 2
#>   group `mean(x)`
#>   <dbl>     <dbl>
#> 1     1        10
```

Dataframe Functions

```r
summary6 <- function(data, var) {
  data |> summarize(
    min = min({{ var }}, na.rm = TRUE),
    mean = mean({{ var }}, na.rm = TRUE),
    median = median({{ var }}, na.rm = TRUE),
    max = max({{ var }}, na.rm = TRUE),
    n = n(),
    n_miss = sum(is.na({{ var }})),
    .groups = "drop"
  )
}

diamonds |> summary6(carat)
#> # A tibble: 1 × 6
#>     min  mean median   max     n n_miss
#>   <dbl> <dbl>  <dbl> <dbl> <int>  <int>
#> 1   0.2 0.798    0.7  5.01 53940      0
```

Good practice: Use `.groups = "drop"` in functions that use `summarize()`.

Dataframe Functions

Because this function uses `summarize()`, it works with grouped data.

```
diamonds |>
  group_by(cut) |>
  summary6(carat)
#> # A tibble: 5 × 7
#>   cut         min  mean median   max     n n_miss
#>   <ord>     <dbl> <dbl>  <dbl> <dbl> <int>  <int>
#> 1 Fair       0.22 1.05    1     5.01  1610      0
#> 2 Good       0.23 0.849   0.82  3.01  4906      0
#> 3 Very Good  0.2  0.806   0.71  4    12082      0
#> 4 Premium    0.2  0.892   0.86  4.01 13791      0
#> 5 Ideal      0.2  0.703   0.54  3.5  21551      0
```

Dataframe Functions

Quickly calculate count and proportion:

```
# https://twitter.com/Diabb6/status/1571635146658402309
count_prop <- function(df, var, sort = FALSE) {
  df |>
    count({{ var }}, sort = sort) |>
    mutate(prop = n / sum(n))
}

diamonds |> count_prop(clarity)
#> # A tibble: 8 × 3
#>   clarity     n   prop
#>   <ord>   <int>  <dbl>
#> 1 I1        741 0.0137
#> 2 SI2      9194 0.170
#> 3 SI1     13065 0.242
#> 4 VS2     12258 0.227
#> 5 VS1      8171 0.151
#> 6 VVS2     5066 0.0939
#> # ℹ 2 more rows
```

Dataframe Functions

Quickly get unique observations meeting a condition:

```
unique_where <- function(df, condition, var) {
  df |>
    filter({{ condition }}) |>
    distinct({{ var }}) |>
    arrange({{ var }})
}

# Find all the destinations in December
flights |> unique_where(month == 12, dest)
#> # A tibble: 96 × 1
#>   dest
#>   <chr>
#> 1 ABQ
#> 2 ALB
#> 3 ATL
#> 4 AUS
#> 5 AVL
#> 6 BDL
#> # i 90 more rows
```

Dataframe Functions

If it is a very specific case that will only ever be used on one dataset, hard-coding could save you having to retype the dataset name when calling the function:

```
subset_flights <- function(rows, cols) {
  flights |>
    filter({{ rows }}) |>
    select(time_hour, carrier, flight, {{ cols }})
}
```

Dataframe Functions

Providing a vector of variable names?

```
count_missing <- function(df, group_vars, x_var) {
  df |>
    group_by({{ group_vars }}) |>
    summarize(
      n_miss = sum(is.na({{ x_var }})),
      .groups = "drop"
    )
}

flights |>
  count_missing(c(year, month, day), dep_time)
#> Error in `group_by()`:
#> ℹ In argument: `c(year, month, day)`.
#> Caused by error:
#> ! `c(year, month, day)` must be size 336776 or 1, not 1010328.
```

```
group_by(c(year, month, day))
instead of
group_by(year, month, day)
```

Dataframe Functions

`pick()`

```
count_missing <- function(df, group_vars, x_var) {
  df |>
    group_by(pick({{ group_vars }})) |>
    summarize(
      n_miss = sum(is.na({{ x_var }})),
      .groups = "drop"
  )
}

flights |>
  count_missing(c(year, month, day), dep_time)
#> # A tibble: 365 × 4
#>    year month   day n_miss
#>   <int> <int> <int>  <int>
#> 1  2013     1     1      4
#> 2  2013     1     2      8
#> 3  2013     1     3     10
```

Plot Functions

Instead of returning a data frame, you might want to return a plot.

```
diamonds |>
  ggplot(aes(x = carat)) +
  geom_histogram(binwidth = 0.1)

diamonds |>
  ggplot(aes(x = carat)) +
  geom_histogram(binwidth = 0.05)
```

```
histogram <- function(df, var, binwidth = NULL) {
  df |>
    ggplot(aes(x = {{ var }})) +
    geom_histogram(binwidth = binwidth)
}

diamonds |> histogram(carat, 0.1)
```

```
diamonds |>
  histogram(carat, 0.1) +
  labs(x = "Size (in carats)", y = "Number of diamonds")
```

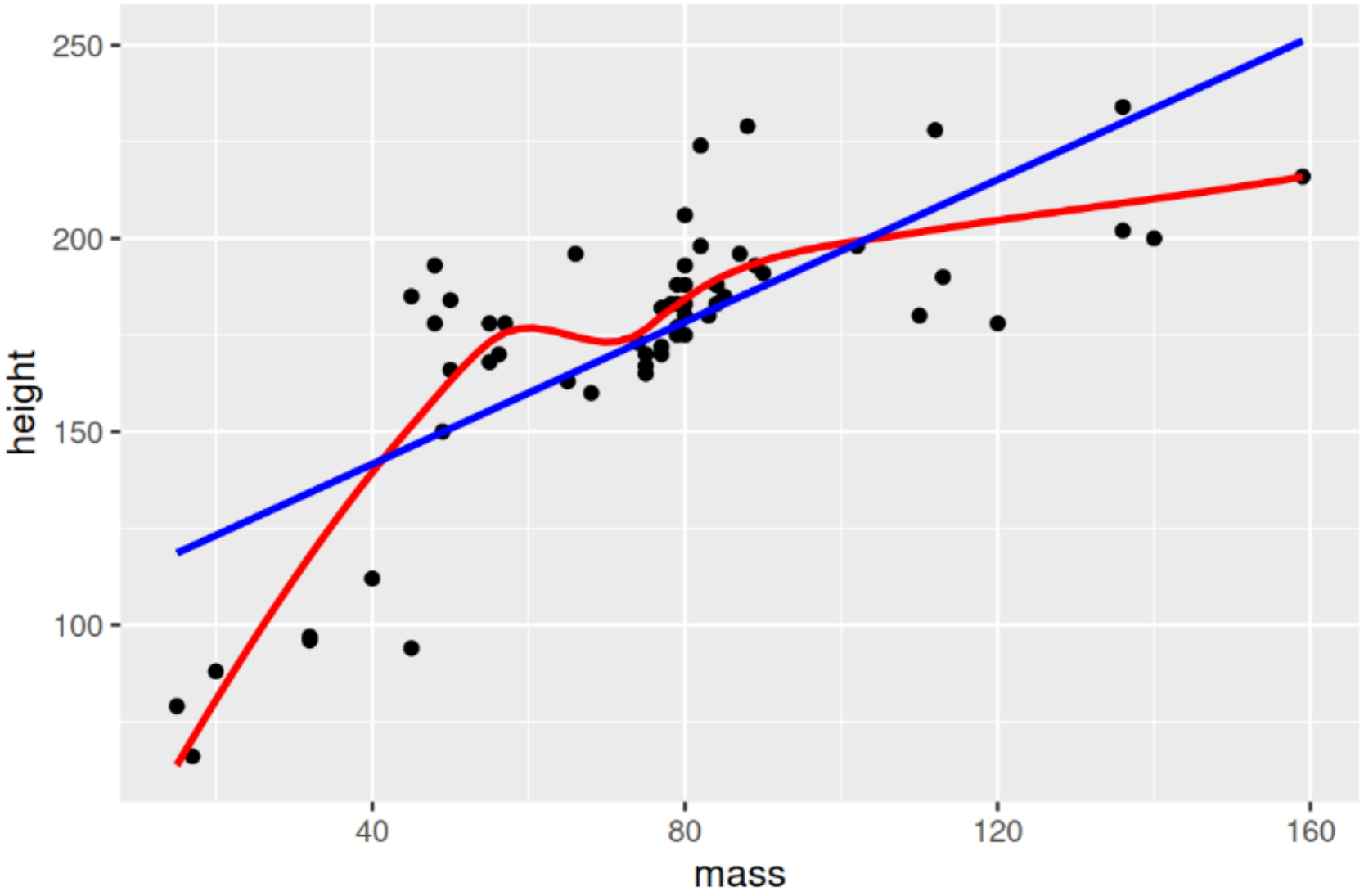
Plot Functions

Using multiple variables:

```
# https://twitter.com/tyler_js_smith/status/1574377116988104704
linearity_check <- function(df, x, y) {
  df |>
    ggplot(aes(x = {{ x }}, y = {{ y }})) +
    geom_point() +
    geom_smooth(method = "loess", formula = y ~ x, color = "red", se = FALSE) +
    geom_smooth(method = "lm", formula = y ~ x, color = "blue", se = FALSE)
}

starwars |>
  filter(mass < 1000) |>
  linearity_check(mass, height)
```

Plot Functions



Plot Functions

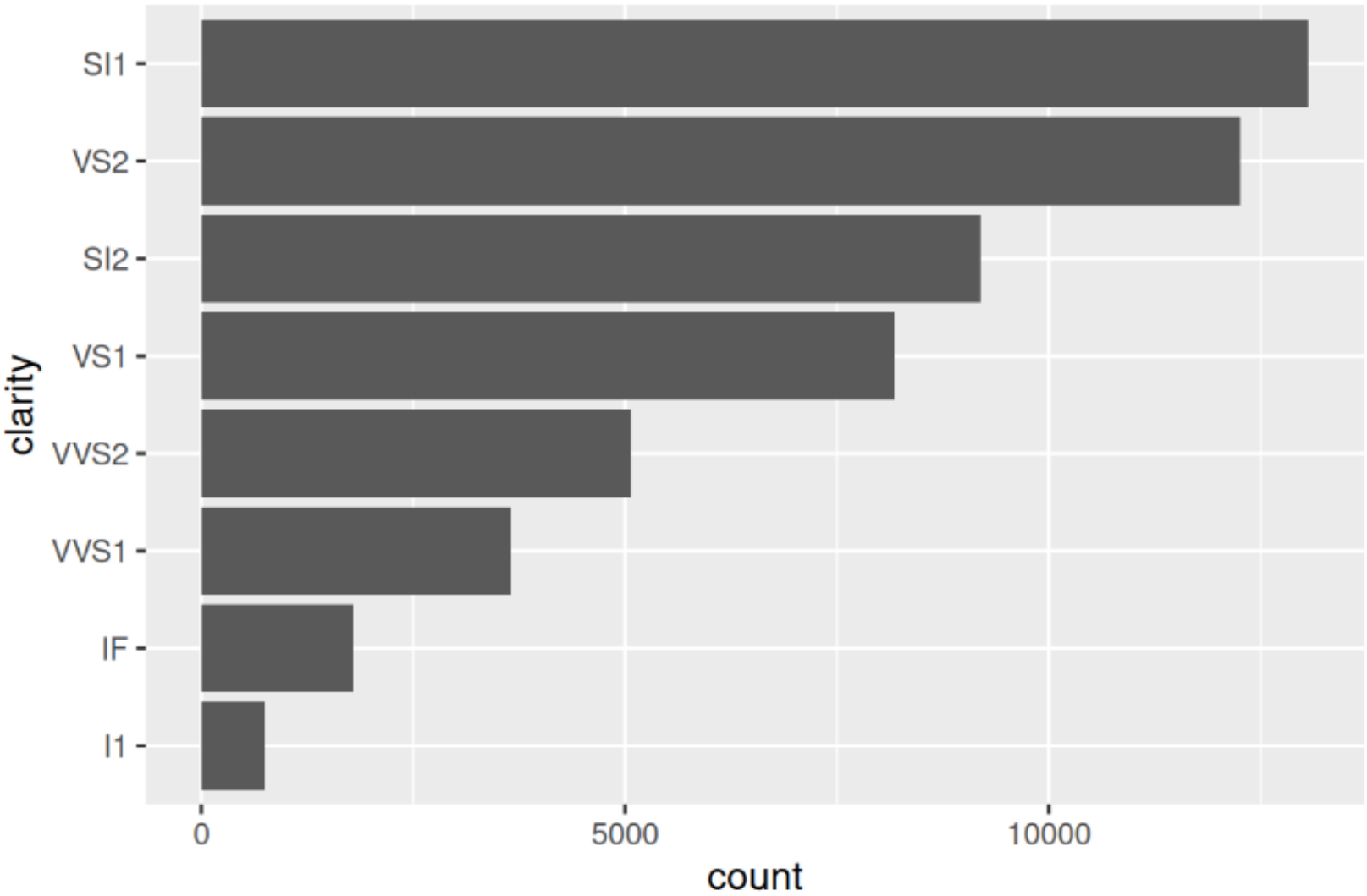
Combine with `mutate()`:

```
sorted_bars <- function(df, var) {
  df |>
    mutate({{ var }} := fct_rev(fct_infreq({{ var }}))) |>
    ggplot(aes(y = {{ var }})) +
    geom_bar()
}

diamonds |> sorted_bars(clarity)
```

Note: You need to use the walrus operator `:=` to mutate the variable.
Why? R only accepts a single literal name on the left of the = operator.

Plot Functions



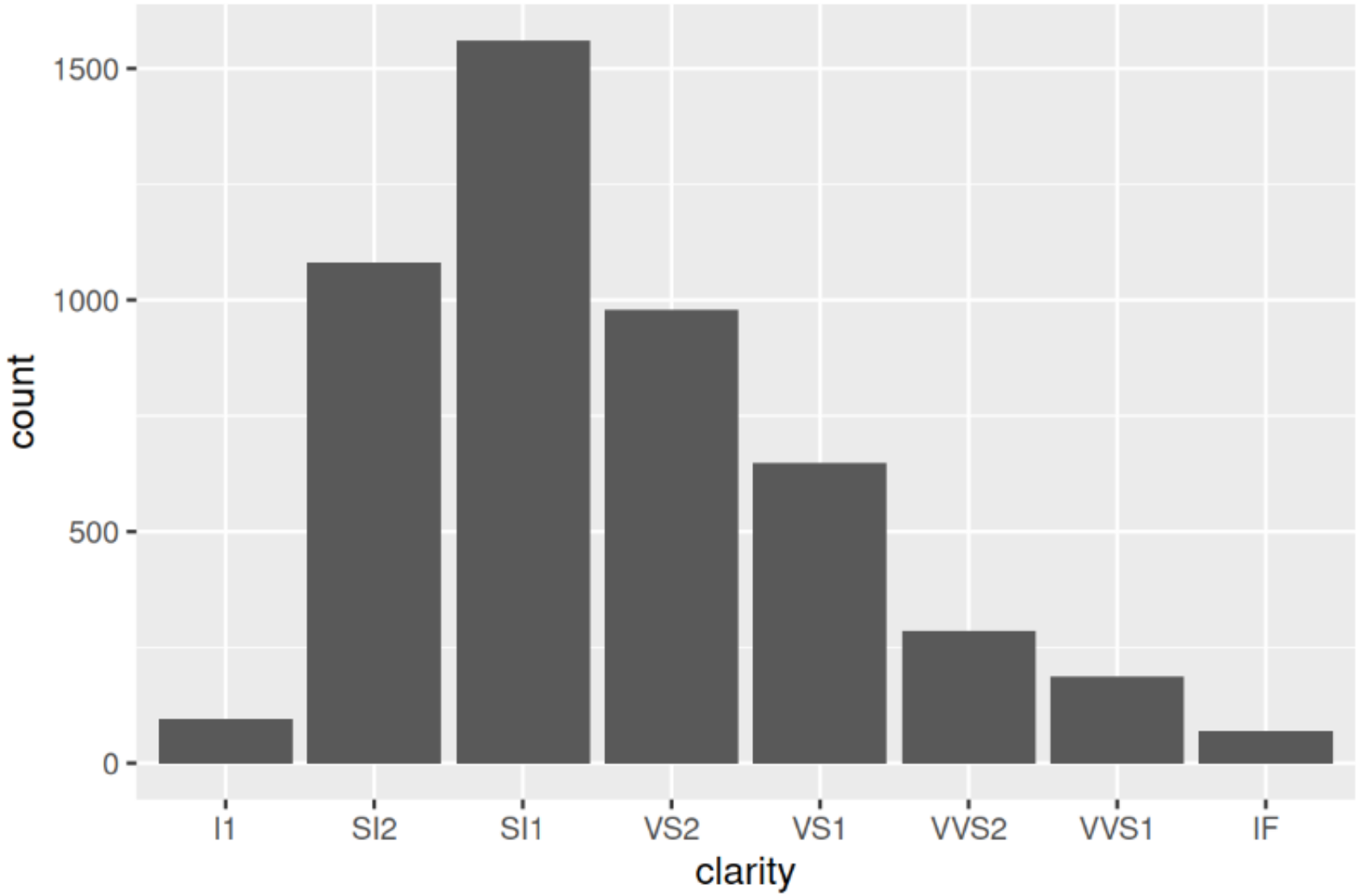
Plot Functions

Combine with a condition:

```
conditional_bars <- function(df, condition, var) {
  df |>
    filter({{ condition }}) |>
    ggplot(aes(x = {{ var }})) +
    geom_bar()
}

diamonds |> conditional_bars(cut == "Good", clarity)
```

Plot Functions



Plot Functions

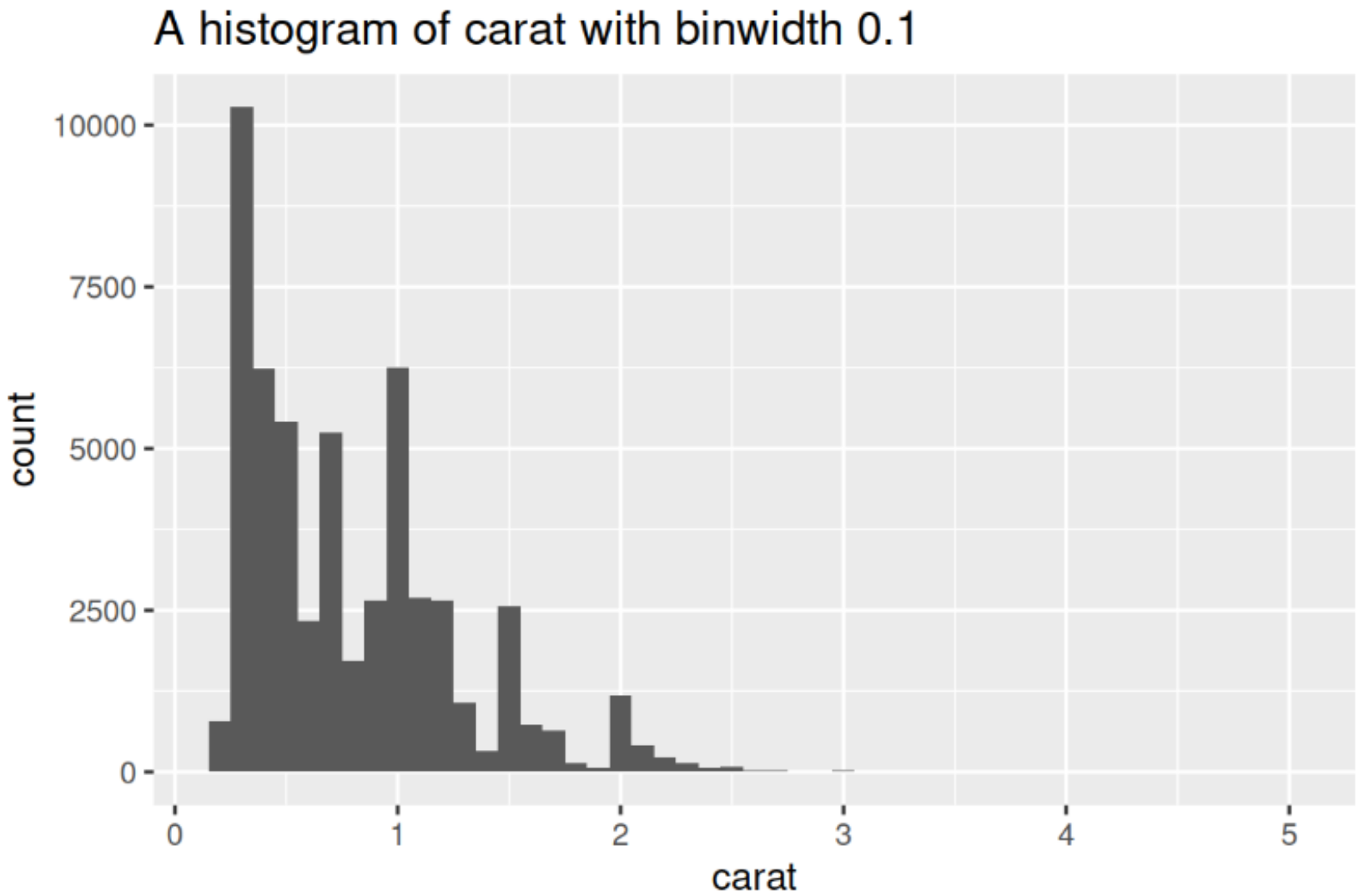
Labelling:

```
histogram <- function(df, var, binwidth) {
  label <- rlang::englue("A histogram of {{var}} with binwidth {binwidth}")

  df |>
    ggplot(aes(x = {{ var }})) +
    geom_histogram(binwidth = binwidth) +
    labs(title = label)
}

diamonds |> histogram(carat, 0.1)
```

Plot Functions



A Note on Style

- R doesn't care what your function or arguments are called but the names make a big difference for humans.
- Keep your code neat, consistent, use comments, and use meaningful names.
- Functions are always followed by squiggly brackets {} and the contents indented by 2 spaces.
- It is recommended to put extra space in the embracing brackets to make them more obvious.