Default arguments

INTRODUCTION TO WRITING FUNCTIONS IN R

Richie Cotton
Curriculum Architect at DataCamp

toss_coin() troubles

```
toss_coin <- function(n_flips, p_head) {
  coin_sides <- c("head", "tail")
  weights <- c(p_head, 1 - p_head)
  sample(coin_sides, n_flips, replace = TRUE, prob = weights)
}
```

Set the default in the signature

```
toss_coin <- function(n_flips, p_head = 0.5) {
  coin_sides <- c("head", "tail")
  weights <- c(p_head, 1 - p_head)
  sample(coin_sides, n_flips, replace = TRUE, prob = weights)
}
```

A template with defaults

```
my_fun <- function(data_arg1, data_arg2, detail_arg1 = default1) {
  # Do something
}
```

Other types of default

```
args(median)
```

```
function (x, na.rm = FALSE, ...)
```

```
library(jsonlite)
args(fromJSON)
```

```
function (txt, simplifyVector = TRUE, simplifyDataFrame = simplifyVector,
    simplifyMatrix = simplifyVector, flatten = FALSE, ...)
```

NULL defaults

By convention, this means

> The function will do some special handling of this argument. Please read the docs.

```
args(set.seed)
```

```
function (seed, kind = NULL, normal.kind = NULL)
```

Categorical defaults

1. Pass a character vector in the signature.
2. Call `match.arg()` in the body.

```
args(prop.test)
```

```
function (x, n, p = NULL, alternative = c("two.sided", "less", "greater"),
  conf.level = 0.95, correct = TRUE)
```

Inside the body

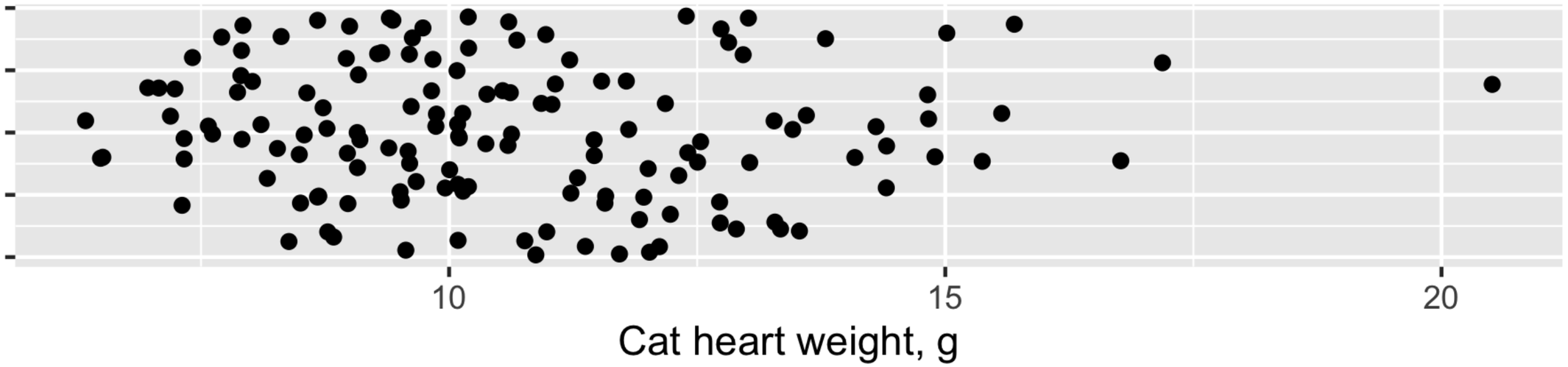
```
alternative <- match.arg(alternative)
```

Cutting a vector by quantile

```
cut_by_quantile <- function(x, n, na.rm, labels, interval_type) {
  probs <- seq(0, 1, length.out = n + 1)
  quantiles <- quantile(x, probs, na.rm = na.rm, names = FALSE)
  right <- switch(interval_type, "(lo, hi]" = TRUE, "[lo, hi)" = FALSE)
  cut(x, quantiles, labels = labels, right = right, include.lowest = TRUE)
}
```

- `x` : A numeric vector to cut
- `n` : The number of categories to cut `x` into
- `na.rm` : Should missing value be removed?
- `labels` : Character labels for the categories
- `interval_type` : Should ranges be open on the left or right?

Cat heart weights

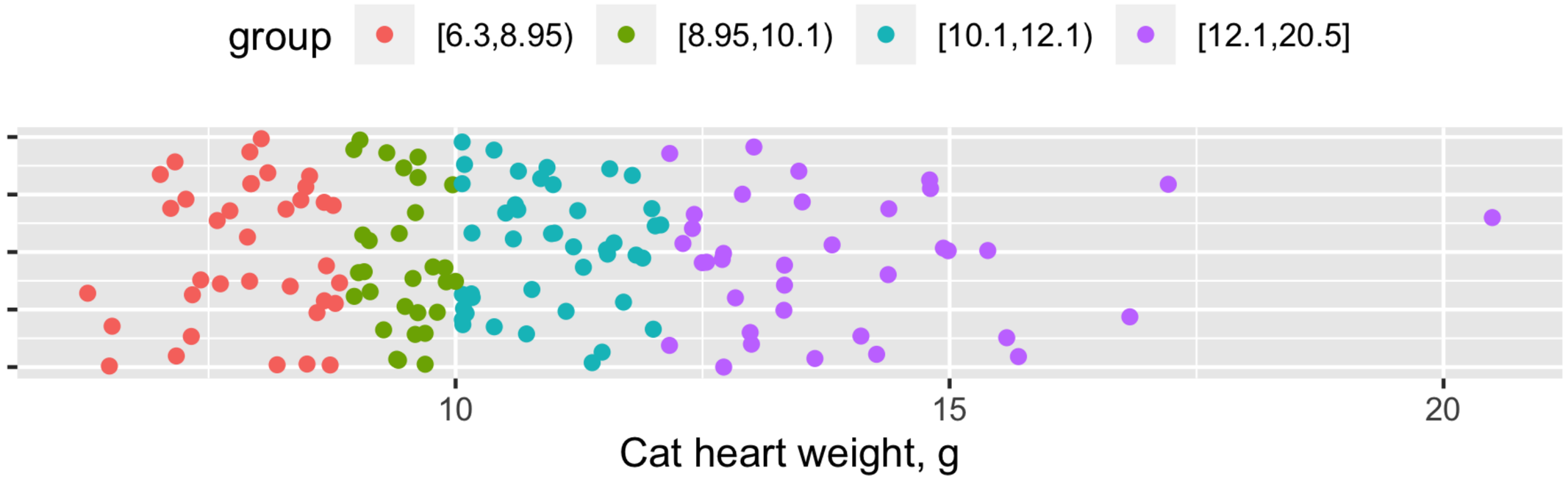



```
quantile(cats$Hwt)
```

```
    0%    25%    50%    75%   100%
 6.300  8.950 10.100 12.125 20.500
```

[1] data(cats, package = "MASS")

Cutting by quantile




```
cut(x, quantile(x))
```

Let's practice!

INTRODUCTION TO WRITING FUNCTIONS IN R

Passing arguments between functions

INTRODUCTION TO WRITING FUNCTIONS IN R

Richie Cotton
Curriculum Architect at DataCamp

Calculating the geometric mean

```
x %>%
  log() %>%
  mean() %>%
  exp()
```

Wrapping this in a function

```
calc_geometric_mean <- function(x) {
  x %>%
    log() %>%
    mean() %>%
    exp()
}
```

Handling missing values

```
calc_geometric_mean <- function(x, na.rm = FALSE) {
  x %>%
    log() %>%
    mean(na.rm = na.rm) %>%
    exp()
}
```

Using ...

```
calc_geometric_mean <- function(x, ...) {
  x %>%
    log() %>%
    mean(...) %>%
    exp()
}
```

The tradeoff

Benefits

- Less typing for you
- No need to match signatures

Drawbacks

- You need to trust the inner function
- The interface is not as obvious to users

Let's practice!

INTRODUCTION TO WRITING FUNCTIONS IN R

Checking arguments

INTRODUCTION TO WRITING FUNCTIONS IN R

Richie Cotton
Curriculum Architect at DataCamp

The geometric mean

```
calc_geometric_mean <- function(x, na.rm = FALSE) {
  x %>%
    log() %>%
    mean(na.rm = na.rm) %>%
    exp()
}
```

```
calc_geometric_mean(letters)
```

```
Error in log(.) : non-numeric argument to mathematical function
```

Checking for numeric values

```
calc_geometric_mean <- function(x, na.rm = FALSE) {
  if(!is.numeric(x)) {
    stop("x is not of class 'numeric'; it has class '", class(x), "'.")
  }
  x %>%
    log() %>%
    mean(na.rm = na.rm) %>%
    exp()
}
```

```
Error in calc_geometric_mean(letters) :
  x is not of class 'numeric'; it has class 'character'.
```

assertive makes errors easy

Writing boilerplate code to handle errors

Handling errors with assertive

Checking types of inputs

- `assert_is_numeric()`
- `assert_is_character()`
- `is_data.frame()`
- …
- `is_two_sided_formula()`
- `is_tskernel()`

Using assertive to check x

```
calc_geometric_mean <- function(x, na.rm = FALSE) {

  assert_is_numeric(x)

  x %>%
    log() %>%
    mean(na.rm = na.rm) %>%
    exp()
}
```

```
 Error in calc_geometric_mean(letters) :
  is_numeric : x is not of class 'numeric'; it has class 'character'.
```

Checking x is positive

```
calc_geometric_mean <- function(x, na.rm = FALSE) {
  assert_is_numeric(x)
  assert_all_are_positive(x)
  x %>%
    log() %>%
    mean(na.rm = na.rm) %>%
    exp()
}
```

```
calc_geometric_mean(c(1, -1))
```

```
Error in calc_geometric_mean(c(1, -1)) :
  is_positive : x contains non-positive values.
There was 1 failure:
  Position Value   Cause
1        2    -1 too low
```

is_* functions

- `assert_is_numeric()`
- `assert_all_are_positive()`

- `is_numeric()` (returns logical value)
- `is_positive()` (returns logical vector)
- `is_non_positive()`

Custom checks

```
calc_geometric_mean <- function(x, na.rm = FALSE) {
  assert_is_numeric(x)
  if(any(is_non_positive(x), na.rm = TRUE)) {
    stop("x contains non-positive values, so the geometric mean makes no sense.")
  }
  x %>%
    log() %>%
    mean(na.rm = na.rm) %>%
    exp()
}
```

```
calc_geometric_mean(c(1, -1))
```

```
Error in calc_geometric_mean(c(1, -1)) :
  x contains non-positive values, so the geometric mean makes no sense.
```

Fixing input

```
use_first(c(1, 4, 9, 16))
```

```
[1] 1
Warning message:
Only the first value of c(1, 4, 9, 16) (= 1) will be used.
```

```
coerce_to(c(1, 4, 9, 16), "character")
```

```
[1] "1"  "4"  "9"  "16"
Warning message:
Coercing c(1, 4, 9, 16) to class ‘character’.
```

Fixing na.rm

```
calc_geometric_mean <- function(x, na.rm = FALSE) {
  assert_is_numeric(x)
  if(any(is_non_positive(x), na.rm = TRUE)) {
    stop("x contains non-positive values, so the geometric mean makes no sense.")
  }
  na.rm <- coerce_to(use_first(na.rm), target_class = "logical")
  x %>%
    log() %>%
    mean(na.rm = na.rm) %>%
    exp()
}
```

```
calc_geometric_mean(1:5, na.rm = 1:5)
```

```
[1] 2.605171
Warning messages:
1: Only the first value of na.rm (= 1) will be used.
2: Coercing use_first(na.rm) to class ‘logical’.
```

Let's practice!

INTRODUCTION TO WRITING FUNCTIONS IN R