



R Functions

**Why should you
write a function?**

What does this code do?

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

What does this code do?

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

What does this code do?

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

Duplication hides the intent

What does this code do?

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

Did you spot the mistake?

How was this code written?

```
> df$a <- (df$a - min(df$a, na.rm = TRUE)) /  
          (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
```

Do one column

How was this code written?

```
> df$a <- (df$a - min(df$a, na.rm = TRUE)) /  
          (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))  
  
> df$a <- (df$a - min(df$a, na.rm = TRUE)) /  
          (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
```

Copy-and-paste

How was this code written?

```
> df$a <- (df$a - min(df$a, na.rm = TRUE)) /  
           (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))  
  
> df$b <- (df$b - min(df$b, na.rm = TRUE)) /  
           (max(df$b, na.rm = TRUE) - min(df$b, na.rm = TRUE))
```

[Edit for next column](#)

How was this code written?

```
> df$a <- (df$a - min(df$a, na.rm = TRUE)) /  
           (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))  
  
> df$b <- (df$b - min(df$b, na.rm = TRUE)) /  
           (max(df$b, na.rm = TRUE) - min(df$b, na.rm = TRUE))  
  
> df$c <- (df$c - min(df$c, na.rm = TRUE)) /  
           (max(df$c, na.rm = TRUE) - min(df$c, na.rm = TRUE))
```

Repeat

When should you write a function?

If you have copied-and-pasted twice,
it's time to write a function

Writing a function makes the intent clearer

```
> df$a <- rescale01(df$a)
> df$b <- rescale01(df$b)
> df$c <- rescale01(df$c)
> df$d <- rescale01(df$d)
```

- Reduces mistakes from copy and pasting
- Makes updating code easier

Functional programming further reduces duplication

```
> library(purrr)
> df[] <- map(df, rescale01)
```

Chapter 3: Functional Programming



R Functions

Let's practice!



R Functions

**How should you write
a function?**

Start with a simple problem

```
> df <- data.frame(  
  a = rnorm(10),  
  b = rnorm(10),  
  c = rnorm(10),  
  d = rnorm(10)  
)  
  
# Rescale the a column in df to a 0-1 range
```

Start with a simple problem

```
> df <- data.frame(  
  a = 1:11,  
  b = rnorm(11),  
  c = rnorm(11),  
  d = rnorm(11)  
)  
  
# Rescale the a column in df to a 0-1 range  
  
# Output should be:  
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```


Get a working snippet of code

```
> (df$a - min(df$a, na.rm = TRUE)) /  
  (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
```

Rewrite to use temporary variables

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

Rewrite to use temporary variables

```
> x <- df$a  
  
> ( x - min( x , na.rm = TRUE)) /  
  (max( x , na.rm = TRUE) - min( x , na.rm = TRUE))
```

Rewrite for clarity

```
> x <- df$a  
  
> rng <- range(x, na.rm = TRUE)  
  (x - rng[1]) / (rng[2] - rng[1])
```

Finally, turn it into a function

```
> x <- df$a

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

> rescale01(x)
```

How should you write a function?

- Start with a simple problem
- Get a working snippet of code
- Rewrite to use temporary variables
- Rewrite for clarity
- Finally, turn into a function



R Functions

Let's practice!



R Functions

**How can you write
a good function?**

What makes a good function?

- Correct
- Understandable
- Functions are for humans and computers
- Correct + Understandable = Obviously correct

What does this code do?

```
> baz <- foo(bar, qux)
```

Who knows?

```
> df2 <- arrange(df, qux)
```

**Good names make code
understandable with minimal
context**

Naming principles

- Pick a consistent style for long names

```
# Good
> col_mins()
> row_maxes()

# Bad
> newData <- c(old.data, todays_log)
```

* Same whether objects, functions, or arguments

- Do not override existing variables or functions

```
# Bad
> T <- FALSE
> c <- 10
> mean <- function(x) sum(x)
```

Function names

- Should generally be verbs

```
# Good  
> impute_missing()  
# Bad  
> imputed()
```

- Should be descriptive

```
# Good  
> collapse_years()  
# Bad  
> f()  
> my_awesome_function()
```

Argument names

- Should generally be nouns
- Use the very common short names when appropriate:
 - `x`, `y`, `z`: vectors
 - `df`: a data frame
 - `i`, `j`: numeric indices (typically rows and columns)
 - `n`: length, or number of rows
 - `p`: number of columns

Argument order

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

```
t.test(x, y = NULL,  
       alternative = c("two.sided", "less", "greater"),  
       mu = 0, paired = FALSE, var.equal = FALSE,  
       conf.level = 0.95, ...)
```

- Data arguments come first **Data arguments: supply data to compute on**
- Detail arguments should have sensible defaults **Detail Arguments: supply arguments that control the details of the computation**

What makes a good function?

- Use good names for functions and arguments
- Use an intuitive argument order and reasonable default values
- Make it clear what the function returns
- Use good style inside the body of the function



R Functions

Let's practice!