

```
library(lubridate)
```

```
##  
## Attaching package: 'lubridate'  
  
## The following object is masked from 'package:base':  
##  
##     date
```

```
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.3.0 --  
  
## v ggplot2 3.2.1      v purrr  0.3.3  
## v tibble  2.1.3      v dplyr  0.8.4  
## v tidyr   1.0.2      v stringr 1.4.0  
## v readr   1.3.1      v forcats 0.4.0  
  
## -- Conflicts ----- tidyverse_conflicts() --  
## x lubridate::as.difftime() masks base::as.difftime()  
## x lubridate::date()        masks base::date()  
## x dplyr::filter()          masks stats::filter()  
## x lubridate::intersect()   masks base::intersect()  
## x dplyr::lag()             masks stats::lag()  
## x lubridate::setdiff()     masks base::setdiff()  
## x lubridate::union()       masks base::union()
```

1. Rewrite `rescale_minmax` so that `-Inf` is set to 0, and `Inf` is mapped to 1.

```
rescale_minmax <- function(x) {  
  rng <- range(x, na.rm = TRUE, finite = TRUE)  
  (x - rng[1]) / (rng[2] - rng[1])  
  x[x == Inf] <- 1  
  x[x == -Inf] <- 0  
  x  
}  
rescale_minmax(c(Inf, -Inf, 0:5, NA))
```

```
## [1] 1 0 0 1 2 3 4 5 NA
```

2. Practice turning the following code snippets into functions. Think about what each function does. What would you call it? How many arguments does it need? Can you rewrite it to be more expressive or less duplicative?

```
x <- 1:10  
mean(is.na(x))
```

```
## [1] 0
```

```
x / sum(x, na.rm = TRUE)
```

```
## [1] 0.01818182 0.03636364 0.05454545 0.07272727 0.09090909 0.10909091  
## [7] 0.12727273 0.14545455 0.16363636 0.18181818
```

```
sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE)
```

```
## [1] 0.5504819
```

Implementation

```
prop_miss <- function(x) {  
  mean(is.na(x))  
}  
my_mean <- function(x) {  
  x / sum(x, na.rm = TRUE)  
}  
my_var <- function(x) {  
  sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE)  
}
```

3. Exercise 19.2.1, Question 4

```
variance <- function(x) {  
  n <- length(x)  
  m <- mean(x)  
  (1/(n - 1)) * sum((x - m)^2)  
}  
  
variance(c(1,2,3))
```

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

```
var(c(1,2,3))
```

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

```
skewness <- function(x) {  
  n <- length(x)  
  v <- var(x)  
  m <- mean(x)  
  third.moment <- (1/(n - 2)) * sum((x - m)^3)  
  third.moment/(var(x)^(3/2))  
}  
  
skewness(c(1, 2, 3, 4, 5))
```

```
## [1] 0
```

Regression problem

```

cal_reg <- function(x, y){
  nx <- length(x)
  ny <- length(y)

  x.intercept <- rep(1, nx)
  cbindx <- cbind(x.intercept, x)
  ymat<- matrix(y, ncol=1)
  beta <- solve(t(cbindx) %*% cbindx) %*% t(cbindx) %*% y
  beta

}

data(trees)
cal_reg(trees$Height, trees$Volume)

```

```

##           [,1]
## x.intercept -87.12361
## x           1.54335

```

```
lm(trees$Volume~ trees$Height)
```

```

##
## Call:
## lm(formula = trees$Volume ~ trees$Height)
##
## Coefficients:
## (Intercept) trees$Height
##      -87.124      1.543

```

R for Data Science-Exercises 9.4.4 - Q2

```

greeter <- function(now = now()) {
  if (between(hour(now), 8, 13)) {
    print("Good morning")
  } else if (between(hour(now), 13, 18)) {
    print("Good afternoon")
  } else {
    print("Good evening")
  }
}

greeter(now())

```

```
## [1] "Good afternoon"
```

Write a function to count the number of even numbers in a vector.

```
count_even <- function(x){  
  cnt <- 0  
  len.x <- length(x)  
  for(j in 1:len.x){  
    if (x[j] %% 2 == 0){  
      cnt = cnt + 1  
    }  
  }  
  cnt  
}  
  
count_even(1:10)
```

```
## [1] 5
```