Lecture 5: Functions

Last time

```r
nsim <- 1000
n <- 100 # sample size
beta0 <- 0.5 # intercept
beta1 <- 1 # slope
results <- rep(NA, nsim)

for(i in 1:nsim){
  x <- runif(n, min=0, max=1)
  noise <- rchisq(n, 1)
  y <- beta0 + beta1*x + noise

  lm_mod <- lm(y ~ x)
  ci <- confint(lm_mod, "x", level = 0.95)

  results[i] <- ci[1] < 1 & ci[2] > 1
}
mean(results)
```

What if I want to repeat my simulations with a different sample size n?

Simulation code for multiple sample sizes

```
nsim <- 1000
beta0 <- 0.5 # intercept
beta1 <- 1 # slope
results <- rep(NA, nsim)

n <- 100 # sample size
for(i in 1:nsim){
  ...
}

n <- 200 # new sample size
for(i in 1:nsim){
  ...
}
```

Are there any issues with this code?

Coding best practices

So far:

- No magic numbers
- Comment your code
- Use informative names
- Set a seed for reproducibility

Also: *don't repeat the same chunk of code multiple times*

Functions

```r
assess_coverage <- function(n, nsim){
  results <- rep(NA, nsim)

  for(i in 1:nsim){
    x <- runif(n, min=0, max=1)
    noise <- rchisq(n, 1)
    y <- 0.5 + 1*x + noise

    lm_mod <- lm(y ~ x)
    ci <- confint(lm_mod, "x", level = 0.95)

    results[i] <- ci[1] < 1 & ci[2] > 1
  }
  return(mean(results))
}

assess_coverage(n = 100, nsim = 1000)
```

```
[1] 0.955
```

Functions

Now I can change the value of `n` without re-writing all the code!

```
1 assess_coverage(n = 100, nsim = 1000)
```

```
[1] 0.938
```

```
1 assess_coverage(n = 200, nsim = 1000)
```

```
[1] 0.94
```

Function components

Here is a simple function to calculate the absolute value:

```r
my_abs <- function(x){
  return(ifelse(x >= 0, x, -1*x))
}

my_abs(-3)
```

```
[1] 3
```

```r
my_abs(c(-2, 5))
```

```
[1] 2 5
```

- **name:** `my_abs`
- **arguments:** `x`
- **body:** everything in the curly braces `{ }`

Function arguments

- The *arguments* `n` and `nsim` allow us to change the sample size and number of simulations
- What other parts of the simulation might we want to change?

```r
assess_coverage <- function(n, nsim){
  results <- rep(NA, nsim)

  for(i in 1:nsim){
    x <- runif(n, min=0, max=1)
    noise <- rchisq(n, 1)
    y <- 0.5 + 1*x + noise

    lm_mod <- lm(y ~ x)
    ci <- confint(lm_mod, "x", level = 0.95)

    results[i] <- ci[1] < 1 & ci[2] > 1
  }
```

Function arguments

```r
assess_coverage <- function(n, nsim, beta0, beta1){
  results <- rep(NA, nsim)

  for(i in 1:nsim){
    x <- runif(n, min=0, max=1)
    noise <- rchisq(n, 1)
    y <- beta0 + beta1*x + noise

    lm_mod <- lm(y ~ x)
    ci <- confint(lm_mod, "x", level = 0.95)

    results[i] <- ci[1] < beta1 & ci[2] > beta1
  }
  return(mean(results))
}
```

Ordering and arguments

```
my_power <- function(x, y){
  return(x^y)
}
```

```
my_power(x = 2, y = 3)
```

```
[1] 8
```

```
my_power(y = 3, x = 2)
```

```
[1] 8
```

```
my_power(2, 3)
```

```
[1] 8
```

```
my_power(3, 2)
```

```
[1] 9
```

- If you don't name the arguments when calling a function, R assumes you passed them in the order of the function definition

Function defaults

```
1 my_power <- function(x, y){
2   return(x^y)
3 }
```

What will happen when I run the following code?

```
1 my_power(3)
```

Function defaults

```
my_power <- function(x, y){
  return(x^y)
}
```

What will happen when I run the following code?

```
my_power(3)
```

```
Error in my_power(3): argument "y" is missing, with no default
```

Function defaults

```
1 my_power <- function(x, y=2){
2   return(x^y)
3 }
```

What will happen when I run the following code?

```
1 my_power(3)
```

Function defaults

```
1 my_power <- function(x, y=2){
2   return(x^y)
3 }
```

What will happen when I run the following code?

```
1 my_power(3)
```

```
[1] 9
```

Function defaults

```
1 my_power <- function(x, y=2){
2   return(x^y)
3 }
```

What will happen when I run the following code?

```
1 my_power(2, 3)
```

Function defaults

```
1 my_power <- function(x, y=2){
2   return(x^y)
3 }
```

What will happen when I run the following code?

```
1 my_power(2, 3)
```

```
[1] 8
```

Function defaults

```
1 my_power <- function(x, y){
2   return(x^y)
3 }
```

What will happen when I run the following code?

```
1 my_power(3)
```

Function defaults

```
my_power <- function(x, y){
  return(x^y)
}
```

What will happen when I run the following code?

```
my_power(3)
```

```
Error in my_power(3): argument "y" is missing, with no default
```

Function defaults

```
1 my_power <- function(x=2, y=4){
2   return(x^y)
3 }
```

What will happen when I run the following code?

```
1 my_power()
```

Function defaults

```
1 my_power <- function(x=2, y=4){
2   return(x^y)
3 }
```

What will happen when I run the following code?

```
1 my_power()
```

```
[1] 16
```

Class activity

https://sta279-s24.github.io/class_activities/ca_lecture_5.html