Lecture 4: 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

```
1  nsim <- 1000
2  beta0 <- 0.5 # intercept
3  beta1 <- 1 # slope
4  results <- rep(NA, nsim)
5
6  n <- 100 # sample size
7  for(i in 1:nsim){
8    ...
9  }
10
11 n <- 200 # new sample size
12 for(i in 1:nsim){
13   ...
14 }
```

these portions of code will be the same

Are there any issues with this code?

Repeated block of code can be issue:

1. Easier to make errors
2. Hard to keep track of any changes
3. Harder to read 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

defining function

arguments

```
1  assess_coverage <- function(n, nsim){
2    results <- rep(NA, nsim)
3
4    for(i in 1:nsim){
5      x <- runif(n, min=0, max=1)
6      noise <- rchisq(n, 1)
7      y <- 0.5 + 1*x + noise
8
9      lm_mod <- lm(y ~ x)
10     ci <- confint(lm_mod, "x", level = 0.95)
11
12     results[i] <- ci[1] < 1 & ci[2] > 1
13   }
14   return(mean(results))
15 }
16
17 assess_coverage(n = 100, nsim = 1000)
```

```
[1] 0.947
```

body (what the function does)

calling function

what the function returns (the output)

specify the arguments

input : n , nsim

do : simulate data & estimate coverage

output : estimated coverage

Functions

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

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

```
[1] 0.957
```

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

```
[1] 0.947
```

Function components

Here is a simple function to calculate the absolute value:

```
1 my_abs <- function(x){
2   return(ifelse(x >= 0, x, -1*x))
3 }
4
5 my_abs(-3)
```

arguments

body

output: absolute value of x

```
[1] 3
```

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

```
[1] 2 5
```

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

making a new vector : c(...)

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

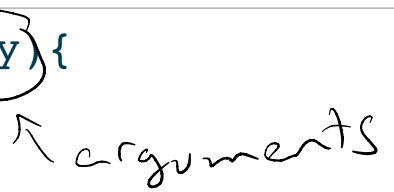
```r
assess_coverage <- function(n, nsim, beta0, beta1){
  results <- rep(NA, nsim)

  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))
}
```

arguments now include beta0 and beta1

function is written in terms of beta0 and beta1 so can specify these inputs

Ordering and arguments

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

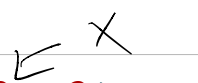



```
1 my_power(x = 2, y = 3)
```

```
[1] 8
```

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

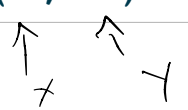
```
[1] 8
```




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


```
[1] 8
```

```
1 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

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

What will happen when I run the following code?

```
1 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 }
```

default value of y (the value of y if not specified when calling the function)

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)
```

8? 4?

Function defaults

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

default value

What will happen when I run the following code?

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

```
[1] 8
```

x=2 y=3

Can use values other than the default

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

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

What will happen when I run the following code?

```
1 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 }
```

default values

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
```

assumes x=2 y=4 (defaults)

Function arguments

We can also pass functions as arguments!

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

  for(i in 1:nsim){
    x <- runif(n, min=0, max=1)
    noise <- noise_dist(n)
    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))
}
```

this will be a function!

generate ε_i according to whatever distribution I want

```r
assess_coverage(n = 100, nsim = 1000, beta0 = 0.5, beta1 = 1,
                noise_dist = rexp)
```

```
[1] 0.948
```

name of the function to generate ε_i

Summary

- Functions can be used to avoid repeating code
- Arguments allow us specify the inputs when we call a function
- If inputs are not named when calling the function, R uses the ordering from the function definition
- All arguments must be specified when calling a function
- Default arguments can be specified when the function is defined
- The input to a function can be a function!

Class activity

https://sta279-f23.github.io/class_activities/ca_lecture_4.html

- If finished early, you may work on homework
- Solutions will be posted on course website