RLadies August Meeting: Functions

Ellen Fitzsimmons & Hope Snyder

8/25/20

What is a function?

- A function is a set of statements organized together to perform a certain task

$$y = mx + b$$

Basic Parts

- Name
- Input
- Computation
- Output

Computing the Slope

Name:

```
slope <- function(){}
```

Computing the Slope

Input:

```r
slope <- function(X, Y, B){}
```

```r
slope <- function(coords, constant){}
```

Computing the Slope

Computations:

```r
slope <- function(X, Y, B){
  (Y-B)/X
}
```

Computing the Slope

Output:

```
slope <- function(X, Y, B){
  M = (Y-B)/X

  print("The slope is equal to ", M)
}
```

Computing the Slope

Use:

```r
slope <- function(X, Y, B){
  M = (Y-B)/X

  print(paste("The slope is equal to ", M))
}

slope(2,5,7)
```

```
## [1] "The slope is equal to  -1"
```

When to Use

- Almost whenever you want!
- When your code repeats a lot
- Something you will want to use in the future
- When you want to apply the code to multiple things at once
- Writing a package

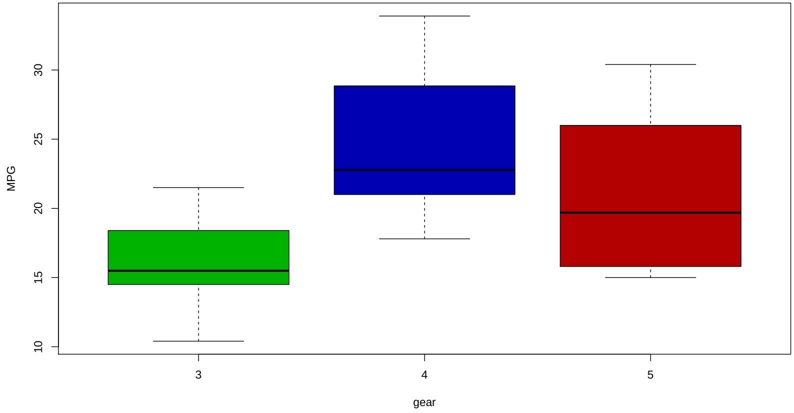
Nesting functions

Functions can be called from within other functions!

```r
dat <- mtcars

pick_a_color <- function(x){
  I=length(unique(x))
  myColor=hsv((1:I)/I,1,.7,1)
  return(myColor)
}

My_plot=function(x,y,...){
  color = pick_a_color(x)

  boxplot(y~x,col=color,... )
}
```

Nesting functions

```r
My_plot(dat$gear,dat$mpg,xlab="gear", ylab="MPG")
```



purrr

You can apply functions to a table of values!

```
lm(dat$mpg ~ dat$gear)
```

```
##
## Call:
## lm(formula = dat$mpg ~ dat$gear)
##
## Coefficients:
## (Intercept)     dat$gear
##       5.623        3.923
```

purrr

```
slope <- function(Y,X){
  (Y-5.623)/X
}

library(purrr)
mat <- purrr::map2_dbl(dat$mpg, dat$gear, slope)
```

purrr

```
mat <- matrix(mat, ncol = 4, byrow = TRUE)
mat
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,] 3.844250 3.844250 4.294250 5.259000
## [2,] 4.359000 4.159000 2.892333 4.694250
## [3,] 4.294250 3.394250 3.044250 3.592333
## [4,] 3.892333 3.192333 1.592333 1.592333
## [5,] 3.025667 6.694250 6.194250 7.069250
## [6,] 5.292333 3.292333 3.192333 2.559000
## [7,] 4.525667 5.419250 4.075400 4.955400
## [8,] 2.035400 2.815400 1.875400 3.944250
```