Lesson 8 (week 10) - Functions and binomials!

YOUR NAME

2023-11-14

Notes for lesson[1]:

- review loops - for and while[2] - before getting started
- functions
- case study using the binomial distribution

Functions

Today we are going to learn about functions! I am going to quote directly from the R god Hadley Wickham:

"Functions allow you to automate common tasks in a more powerful and general way than copy-and-pasting. Writing a function has four big advantages over using copy-and-paste:

1. You can give a function an evocative name that makes your code easier to understand.
2. As requirements change, you only need to update code in one place, instead of many.
3. You eliminate the chance of making incidental mistakes when you copy and paste (i.e. updating a variable name in one place, but not in another).
4. It makes it easier to reuse work from project-to-project, increasing your productivity over time."

If this tutorial makes no sense to you, you can go read his.[3]

All the fun stuff you do with the `tidyverse` packages are just functions **someone else** built and published as a package. And now you are going to learn to do exactly the same thing.

So what is in a function?

Components:

- Function Name: How we will refer to our function.
- Arguments: Variables passed into the function.
- Function Body: The code that performs the task.
- Return Statement: The output of the function.

[1] Developed by Daphne Hansell ('24) and Dr. Bitarello
[2] Previous two labs.
[3] https://r4ds.hadley.nz/functions

```
functionName <- #function name
        function(arg1, arg2, ...) { #arguments
         # Function Body
         # ...
        return(result) #return statement
}
```

Example: The function `addNumbers` adds 2 numbers together and returns the output.

```
addNumbers <- function(num1, num2) { #function name and arguments
  sum <- num1 + num2 #function body
  return(sum) #return statement
} #close the curly brackets!
```

Run it in your console and then try this code:

```
addNumbers(12, 865)
```

```
## [1] 877
```

Woah look at that! It added the numbers and printed the result. You could try again with other numbers but I think you get the idea.

Ex.1: Can you now make a basic function that multiplies two numbers and returns the result? Call it something like `multiplyNumbers`.

```
# function name and arguments
mulitplyNumbers<- function(num1, num2) {
  #function body
  product <- num1 * num2
  #return statement
  return(product)
}
```

Try it with some numbers! You wrote your very first function!!!

Ok, these seem fairly useless – obviously R can add and multiply pretty easily without this. But lets say you're a budding young biologist who has to work with Celsius in their lab but wants to know what the temperatures are in Fahrenheit. You are sick of constantly googling the conversion, so you decide to automate it in R.

Here's the equation to convert Celsius (centigrade) to Fahrenheit.

$$^0F = (1.8 \times ^0 C) + 32$$

Note: we write these equations using a typesetting language called Latex. It's great and awesome but you don't need to know it, but now you know what this is.

Ex.2: Write a simple function that takes in a value in Celsius and converts it into Fahrenheit.

```
#call your function CtoF or anythign meaningful to you
#give it one argument called "c"


#In the body of the function, write out the math of the conversion.
#Use the formula above to create a variable "f" to store the converted temperature.



#Then combine this using the structure from above. Make sure you call return()

CtoF<-function(c){
f<- (c *1.8) + 32
return(f)
}
```

Now you can change your temperatures and convert them easily! Lets do the same for Fahrenheit to Celsius.

$$^0C = \frac{^0F - 32}{1.8}$$

```
#first write out the math of the conversion. Use the formula above and create variables
↪  for the temperatures

#now name your function something like FtoC or anything you'll remember

#Then combine this using the structure from above. Make sure you call return()
FtoC<-function(f){
c<- (f-32)/1.8
return(c)
}
```

What if we wanted to combine these functions to make one big temperature conversion function? You'd have to know how to use conditional statements in a function

##Conditional Statements in Functions It is useful to have functions that can do different things depending on the input. For example, if we had a big temperature conversion function you would want it to differentiate Fahrenheit from Celsius and do different math depending on which you gave it.

For example this function tells you if a number is even. if the number is even, it returns `TRUE`. If it is odd, it returns `FALSE`.

```
isEven <- function(number) {
  if (number %% 2 == 0) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}
```

Run it and try some numbers.

Now make the reverse of this function called isOdd. If the number is not divisble by 2 it should return `TRUE`, and if the number is divisible by 2 it should return `FALSE`

```r
isOdd <- function(number) {
  if (number %% 2 == 0) {
    return(FALSE)
  } else {
    return(TRUE)
  }
}
```

Run this one also.

You can see how useful conditional statements are once you can automate them with functions! Remember how you would set your variable to a value, run the conditional then go back and change the original variable if you wanted to do it again? Like this example from lesson 7:

```r
number <-3


if (number > 0) {
  print("The number is positive.")
} else if (number < 0) {
  print("The number is negative.")
} else {
  print("The number is zero.")
}
```

```
## [1] "The number is positive."
```

Try turning this into a function called classifyNumber that uses an input number and returns if it's positive, negative or 0. Hint: You can copy and paste that code and just assign it to a function call

```r
classifyNumber<-function(number){
if (number > 0) {
  print("The number is positive.")
} else if (number < 0) {
  print("The number is negative.")
} else {
  print("The number is zero.")
}
}
```

Mulitple Arguments

So far all of our functions have taken just one or two inputs. Try running one of them without an input. You'll get an error that the argument is missing and has no default. Many of the functions you use regularly have a bunch of arguments that you could customize but don't have to. They have default values that R uses unless you tell it to do something else. For example, type ?mean into your console. You probably only ever include the first argument, the values you want the mean of. But you have the option to exclude some percentage of the values or remove missing data.

So if we wanted to write a function that raises numbers to the nth power but we think we will mostly use it for squaring numbers, we could write it with a default argument.

```
powerraiser<-function(x, power = 2){
        result<-x^power
        return(result)
}
```

Try running it with only one number, then try adding a second argument (that isn't 2).

You can also have the default for an argument be `null`. If you set the default to null, your function will simply ignore that argument. Lets say you want to make a function that calculates the area of a rectangle. Your arguments would be length and width. But if you wanted to calculate the area of a square, you would only need to enter one value. Write such a function

```
#first name your function and add two arguments. Set width equal to NUll
calculate_area <- function(length, width = NULL) {
#then add a conditional statement: If width is NUll, set width equal to length.
  if (is.null(width)) {
    width <- length
  }
 #then calculate the area and don't forget to return your result
 area <- length * width
  return(area)
}
```

Combining everything!

Now lets go back to our temperature functions from earlier.

```
CtoF<-function(c){
f<- (c *1.8) + 32
return(f)
}

FtoC<-function(f){
c<- (f-32)/1.8
return(c)
}
```

We want to make our one big function.

```
#first name it tempchange and set both c and f to NUll
tempchange<-function(c = NULL, f = NULL){
#if f is null that means you are converting from Celsius, so you should be returning
↪  Fahrenheit
if(is.null(f)){
        tf<- (c *1.8) + 32
        return(tf)
#otherwise, you are starting from Fahrenheit, so you should be returning Celsius
}else{
      c<- (f-32)/1.8
        return(c)
}
}
```

Case study: Hospital readmission rates of acute ischemic stroke in California

Chapters covered:

- This covers chapters 3 and 7.

Tools we will use

- Calculating binomial probabilities:

E.g. probability of six successes in 27 trials, where p of success is 0.25.

```
`dbinom(6, size = 27, prob = 0.25)`
```

- Binomial test:

E.g. is finding 10 successes out of 25 trials different from expected number of successes if p of success is 0.061?

```
`binom.test(10, n = 25, p = 0.061,alternative = "two.sided")`
```

- Agresti-Coull 95% confidence interval for the proportion:

E.g. What is the Agresti-Coull 95% confidence interval for a proportion based on 131 successes out of 169 trials?

```
`binom::binom.confint(131, n = 169, method = "ac")`
```

Motivation

According to the Nationwide Readmissions Database of the Healthcare Cost and Utilization Project between 2010 and 2015, the 30-day hospital readmission rate for acute ischemic stroke patients on a national level is 12.4% (Bambhroliya et al. 2018). A researcher wants to test whether the proportion of 30-day hospital readmissions for a California hospital with an "as expected" hospital quality rating differs from the national 30-day readmission proportion.

Questions of interest

Two questions of interest are:

1. Is the proportion of stroke patients readmitted within 30-days of discharge from CA hospital different from the nationwide proportion?
2. What is the 95% confidence interval for the proportion of acute ischemic stroke patients readmitted within 30 days of discharge?

Data

Make sure to install the package `binom` using the `install.packages()` function, then load it.

```r
#install.packages("binom")
library(binom)
```

The data are in csv data file: *readmin.csv*

The data file contains ischemic stroke 30-day hospital readmission incidence data for a random sample of patients in a California hospital with an "as expected" quality rating obtained from a set of hospital records for 2014-2015. The 30-day readmission data from a sample of 50 patients was recorded.

Read in the dataset, which we'll call *readmin*.

```r
# read it in
readmin <- read.csv("readmin.csv")

#have a look using head()
head(readmin) # Shows the first six rows of the data set
```

```
##   ReadmissionStatus
## 1                 0
## 2                 0
## 3                 0
## 4                 0
## 5                 0
## 6                 0
```

The variable *ReadmissionStatus* is a binary variable which is equal to 1 if the patient was readmitted to the hospital within 30 days of discharge and equal to 0 if the patient was not readmitted to the hospital within 30 days of discharge.

Exploring the Data

Let's begin by looking at the variable of interest, readmission.Use the function `table` to have a look at the counts per readmission status.

```r
# Table of Readmission Status
table(readmin$ReadmissionStatus)
```

```
##
##  0  1
## 44  6
```

There are 6 people who are readmitted to the hospital within 30 days and 44 people who are not readmitted.

Data Analysis

Determine the proportion of individuals readmitted in the sample.

Hint: you can use the function `prop.table` on the output of your previous command.

```r
# Proportion table of Readmission Status
prop.table(table(readmin$ReadmissionStatus))
```

```
##
##    0    1
## 0.88 0.12
```

Q2.1. What % pf the sample was readmitted within 30 days?

Answer: 12% (or $p = \frac{6}{50} = 0.12$) of the sample was readmitted within 30 days.

The Binomial test Write the null and alternative hypotheses for this statistical test, comparing the proportion of readmission in our study, to the population proportion of 0.124.

H_0 : The proportion of readmission in the population is 0.124

H_A : The proportion of readmission in the population is not 0.124

Run a two-sided binomial test to see if the sample proportion differs significantly from the population proportion. Hint: look at `?binom.test` and also lecture slides if needed.

```r
# Binomial test of readmission status
binom.test(x = 6, n = 50, p = 0.124,
          alternative = "two.sided")
```

```
##
##  Exact binomial test
##
## data:  6 and 50
## number of successes = 6, number of trials = 50, p-value = 1
## alternative hypothesis: true probability of success is not equal to 0.124
## 95 percent confidence interval:
##  0.04533532 0.24310132
## sample estimates:
## probability of success
##                   0.12
```

Q2.2: How do you interpret the result of this test?

We fail to reject the null hypothesis that the population proportion differs significantly from 0.124 ($P = 0.9316$).

Agresti-Coull 95% Confidence Interval for the proportion Next, calculate the 95% CI for the proportion of stroke patients who were readmitted within 30 days of discharge using the Agresti-Coull method. Hint: the binom package has a function that does this. Read about it with `??binom::binom.confint`

```r
# 95% CI for binomial test
binom.confint(6, n = 50, method = "ac")
```

```
##          method x  n mean      lower     upper
## 1 agresti-coull 6 50 0.12 0.05249712 0.2417271
```

Q2.3: Do we find evidence that the population proportion is significantly different from 0.124? How precise is our CI?

Answer: Based on these results, we find no evidence that the population proportion is significantly different from 0.124. The 95% CI interval ($0.0525 < p < 0.2417$) indicates medium level of precision, given that it covers nearly 20% of possible proportions.

References

Bambhroliya AB, Donnelly JP, Thomas EJ, et al., "Estimates and Temporal Trend for US Nationwide 30-Day Hospital Readmission Among Patients With Ischemic and Hemorrhagic Stroke.", *JAMA Netw Open*, 1[2018]:e181190 https://jamanetwork.com/journals/jamanetworkopen/fullarticle/2696869