Solutions: A First Guide to Programming in R

2022-01-25

Contents

2 The Basics

Exercise 2.1. In the following code box, create three new variables named 'name', 'age', and 'birthday' (in a MM/DD/YYYY format), containing your info. What data types are these? When you're done, run the code to make sure you've declared your variables correctly.

Exercise 2.2. Use the `max` and `min` functions to find the largest value of `list_of_numbers` (see below), then store those two values in two different variables. Then find the product of the max and min, save it to another variable, and print it using the `print` function (note that both `max`, `min`, and `print` can all take in one parameter).

Exercise 2.3. Write a function called `quad_form` that accepts three parameters (`a, b, c`) and return the positive solution of the quadratic formula (so return $x = \frac{-b+\sqrt{b^2-4ac}}{2a}$). A few hints:

- The square root command is `sqrt()` (alternatively, you can raise something to the 0.5 power with the carat).
- Order of operations **does** work as you've been taught in school (fun fact: multiplication has higher precedence than division in R). However, you need to be explicit when writing multiplication, i.e. you need to use the asterisk to show that you are multiplying two numbers together.
- Be sure to explicitly return the number you calculate! R will actually return the output of the last line that you write so you don't need to type `return`, but it is very good practice to do so.

```
quad_form = function(a, b, c){
 x = (-b + sqrt(b^2 - 4*a*c))/(2*a)
 return(x)
}

# test if this works
quad_form(1, 4, 4)
```

```
## [1] -2
```

Exercise 2.4. Predict the output of this program! To check your answer, just copy the code and run it.

```
print("Green eggs and ham.")
```

```
## [1] "Green eggs and ham."
```

```
if(1 > 2 | "red" == "blue"){
  print("I do not like green eggs and ham.")
}

if("red" == "green" | 2 > 1){
  print("I do not like them, Sam-I-Am.")
}
```

```
## [1] "I do not like them, Sam-I-Am."
```

```
# note: when you have a (NOT) in front of a bunch of things,
# you can evaluate from inside-out
if(!(3 != 3 | "pass" == "fail")){
  print("Would you like them here or there?")
}
```

```
## [1] "Would you like them here or there?"
```

```
x = -1
y = 3

if(x < y & x > y){
  print("I would not like them here or there.")
} else if (x < y){
  print("I would not like them anywhere.")
} else if(x > y){
  print("Would you like them in a house?")
} else{
  print("Would you like them with a mouse?")
}
```

```
## [1] "I would not like them anywhere."
```

Exercise 2.5. Add random numbers to `x` until `x` is greater than 5, printing `x` after every addition. Use the function `runif(1)` to generate one random number at a time.

Note: your output might vary.

```
x = 0
while(x < 5){
  x = x + runif(1) # no += here :(
}
x
```

```
## [1] 5.060138
```

Exercise 2.6. What happens when you run the following code? Notice that the for loop works, even though you don't use `i` at all in the loop.

```
for (i in 1:5){
  print("USC")
}
```

```
## [1] "USC"
## [1] "USC"
## [1] "USC"
## [1] "USC"
## [1] "USC"
```

Exercise 2.7. Print out 10, 9, 8, . . . , 1 in descending order. Can you think of more than one way to do it?

```
# with a for loop, with some math; this does it line by line
# don't do this in practice
for(i in 1:10){
  print(10 - i + 1)
}
```

```
## [1] 10
## [1] 9
## [1] 8
## [1] 7
## [1] 6
## [1] 5
## [1] 4
## [1] 3
## [1] 2
## [1] 1
```

```
# the "R" way to do it
print(10:1)
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```
# using the seq function
print(seq(10, 1))
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

Exercise 2.8. (nested for loops) Print out all the possible sums from rolling two dice, with repetition (i.e. 1+1, 1+2, etc.). You should see 36 values.

```
# with two for loops
for(i in 1:6){
  for(j in 1:6){
    print(i+j)
  }
}
```

```
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 5
```

```
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
```

```
# with one for loop, formatted differently
# this is faster but still not ideal
# I can't think of a better solution in R though
for(i in 1:6){
  print(i + 1:6)
}
```

```
## [1] 2 3 4 5 6 7
## [1] 3 4 5 6 7 8
## [1] 4 5 6 7 8 9
## [1]  5  6  7  8  9 10
## [1]  6  7  8  9 10 11
## [1]  7  8  9 10 11 12
```

Exercise 2.9. Given the following 10×10 matrix, calculate the mean across each row and down each column. Store these store values two separate numeric vectors without using a for loop.

```
mean_mat = matrix(1:100, nrow = 10, ncol = 10)  # data frame of values

# the solution is very unexciting
row_means = rowMeans(mean_mat)
col_means = colMeans(mean_mat)
```

3 R-Specific Syntax

Exercise 3.1. Use these three commands to look at `mtcars`! Match the outputs to the ones you see below. Be sure to load in `tidyverse` using `library()`!

```
print("(1)")
str(mtcars)

print("(2)")
print(head(mtcars))

print("(3)")
# glimpse is my preferred way to look at the data
# you'll need to have tidyverse loaded though
```

```
library(tidyverse)
glimpse(mtcars)
```

Exercise 3.2. Create a new variable `mtcars_4`, and assign to it fourth column from the `mtcars` data set. Do it using the `$` syntax and the square bracket index! (You may need use one of the `names()`, `head()`, `str()`, or `glimpse()` functions to identify the name of the fourth column).

```
mtcars_4 = mtcars$hp
mtcars_4 = mtcars[, 4]
```

Exercise 3.3. Get the third element of the `hp` column in `mtcars`. There's two ways to do it: can you think of both?

```
mtcars[3,4]
```

```
## [1] 93
```

```
mtcars$hp[3]
```

```
## [1] 93
```

Exercise 3.4. Run `head(mtcars)` again. Suppose I accessed the value `3.440` from the `wt` column. How would I have accessed this value?

```
head(mtcars)
```

```
mtcars[5, 6]
```

```
## [1] 3.44
```

```
mtcars$wt[5]
```

```
## [1] 3.44
```

Exercise 3.5. Now, do the following to practice accessing data in data frames.

1. Get the first three columns of `mtcars`.
2. Get the first two rows of the last four columns of `mtcars`. (Hint: use the `dim()` command to see how many columns are in `mtcars`.)
3. Get the even-numbered rows and from `mtcars`. (Hint 1: you can create a new variable that contain even numbers only, and put that variable as the rows.) (Hint 2: The variable is equal to `1:16 * 2`).

```
# 1
mtcars[, 1:3]
```

```
# 2
mtcars[1:2, 8:11]
```

```
# 3
mtcars[2*(1:(nrow(mtcars)/2)), ] # or mtcars[2 * (1:16), ]
```

Exercise 3.6. Halve the mileage in `mtcars_copy`, then set the `super_mpg` to the original `mpg` column in `mtcars`.

```
# not running this code since this is a different document
mtcars_copy$mpg = mtcars_copy$mpg / 2
mtcars_copy$super_mpg = mtcars$mpg
```

Exercise 3.7. We have included the `cereal.csv` data set, which contains nutritional information about a set of cereals. For information about what the columns represent, see here. We'll be using it for the rest of the notebook!

1. Read in the cereal data into an object named `cereal` (refer to the previous line of code). Refer to the `getwd()` and `setwd()` functions mentioned previously if you get an error relating to not being able to find the file.
2. Get a summary of the data using the function of your choice.
3. Create a new variable, `cereal_names`, which contains the names of the cereals.
4. Look through `cereal_names` and identify your three favorite cereals (or just pick any three that aren't too close together). Create a new vector, `favorite_cereals`, which contains your three favorite cereals. (Use the `c("cereal 1", "cereal 2", "cereal 3")` syntax to make this variable, replacing cereal 1/2/3 with the names of your favorites).

```
cereal = read.csv("cereal.csv")
str(cereal)
```

```
## 'data.frame':    77 obs. of  16 variables:
##  $ name    : chr  "100% Bran" "100% Natural Bran" "All-Bran" "All-Bran with Extra Fiber" ...
##  $ mfr     : chr  "N" "Q" "K" "K" ...
##  $ type    : chr  "C" "C" "C" "C" ...
##  $ calories: int  70 120 70 50 110 110 110 130 90 90 ...
##  $ protein : int  4 3 4 4 2 2 2 3 2 3 ...
##  $ fat     : int  1 5 1 0 2 2 0 2 1 0 ...
##  $ sodium  : int  130 15 260 140 200 180 125 210 200 210 ...
##  $ fiber   : num  10 2 9 14 1 1.5 1 2 4 5 ...
##  $ carbo   : num  5 8 7 8 14 10.5 11 18 15 13 ...
##  $ sugars  : int  6 8 5 0 8 10 14 8 6 5 ...
##  $ potass  : int  280 135 320 330 -1 70 30 100 125 190 ...
##  $ vitamins: int  25 0 25 25 25 25 25 25 25 25 ...
##  $ shelf   : int  3 3 3 3 3 1 2 3 1 3 ...
##  $ weight  : num  1 1 1 1 1 1 1 1.33 1 1 ...
##  $ cups    : num  0.33 1 0.33 0.5 0.75 0.75 1 0.75 0.67 0.67 ...
##  $ rating  : num  68.4 34 59.4 93.7 34.4 ...
```

```
cereal_names = cereal$name
favorite_cereals = c("Cinnamon Toast Crunch", "Cocoa Puffs", "Honey Nut Cheerios")
```

4 Vectors, Vectorized Operators, and Boolean Indexing

Exercise 4.1. Create a vector containing the strings `"cyan"`, `"yellow"`, "`magenta`", and `"key"`. Select the middle two values (yellow and magenta).

```
colors = c("cyan", "yellow", "magenta", "key")
selected_colors = colors[2:3]
```

Exercise 4.2. Given the following numeric data, remove the values that contain NA values.

```
numeric_values = c(1, 2, NA, 3, NA, NA, NA, 4)
numeric_values = numeric_values[!is.na(numeric_values)]
numeric_values
```

```
## [1] 1 2 3 4
```

Exercise 4.3 (Cereal, Booleans)**.** Let's apply these skills in a practice context! Recall that for out cereal, we made a `favorite_cerals` vector. Let's select our favorite cereals.

1. Use the `%in%` operator to determine which of the `cereal_names` are in your favorites. (You may be confused about which order to put the vectors. Experiment with both orders and compare the outputs! Think about why one order works and the other doesn't).

2. Create a new data frame, `favorite_cereal_data`, that contains the data in `cereal` that corresponds to your favorite cereals. (Hint: use the `[rows, columns]` syntax. If you pass a boolean vector to rows/columns, it will select those rows/columns, just the same way it works with a 1-dimensional vector. Reference the previous set of commands!)
3. Examine the dimensions of `favorite_cereal_data` by calling the `dim()` function on it. If you did everything correctly, it should be 3 x 16!

```
# 1
names_in_favs = cereal$name %in% favorite_cereals
# 2
favorite_cereal_data = cereal[names_in_favs, ]
# 3
dim(favorite_cereal_data)
```

```
## [1]  3 16
```

Exercise 4.4. Given the following numeric data, convert the NA values to -1. (You might wonder what to pass to the third argument. Think about what might make sense and try it!)

Note: there is an approach to do so by boolean indexing – can you think of how to do it without using "ifelse"?

```
numeric_values = c(1, 2, NA, 3, NA, NA, NA, 4)
numeric_values = ifelse(is.na(numeric_values),
                        -1,
                        numeric_values)

# alternatively
numeric_values = c(1, 2, NA, 3, NA, NA, NA, 4)
numeric_values[is.na(numeric_values)] = -1
numeric_values
```

```
## [1]  1  2 -1  3 -1 -1 -1  4
```

Exercise 4.5. Given the same numeric data, remove the NA values. Then create a new vector where all values smaller than the mean are replaced with `"small"`; otherwise, they are replaced with `"large"`.

```
numeric_values = c(1, 2, NA, 3, NA, NA, NA, 4)
numeric_values = numeric_values[!is.na(numeric_values)]

mean_value = mean(numeric_values)

numeric_values = ifelse(numeric_values < mean_value, "small", "large")
numeric_values
```

```
## [1] "small" "small" "large" "large"
```

Exercise 4.6 (Cereal, ifelse())**.** In the cereal data, the `type` column represents if the cereal should be eaten hot or cold, which is encoded by `"C"` or `"H"`. Create a new column, `type_full`, which writes out `type` in full (i.e. convert `"C"` to `"Cold"` and `"H"` to `"Hot"`).

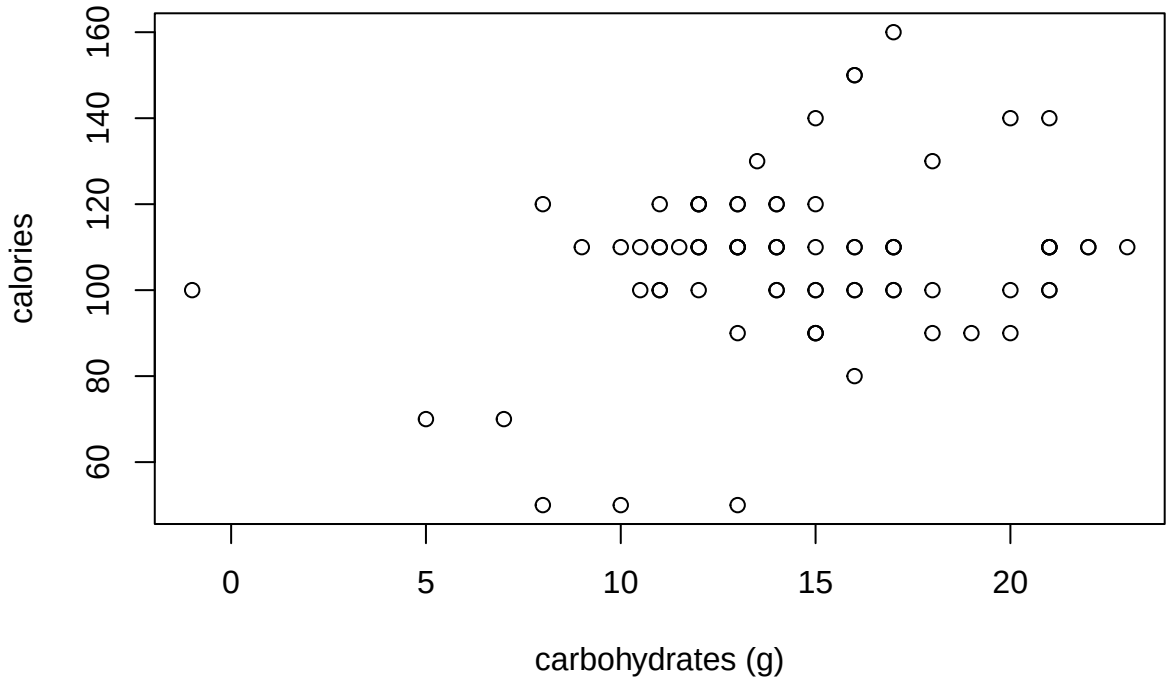
```
cereal$type_full = ifelse(cereal$type == "C", "Cold", "Hot")
cereal
```

5 Plotting Data

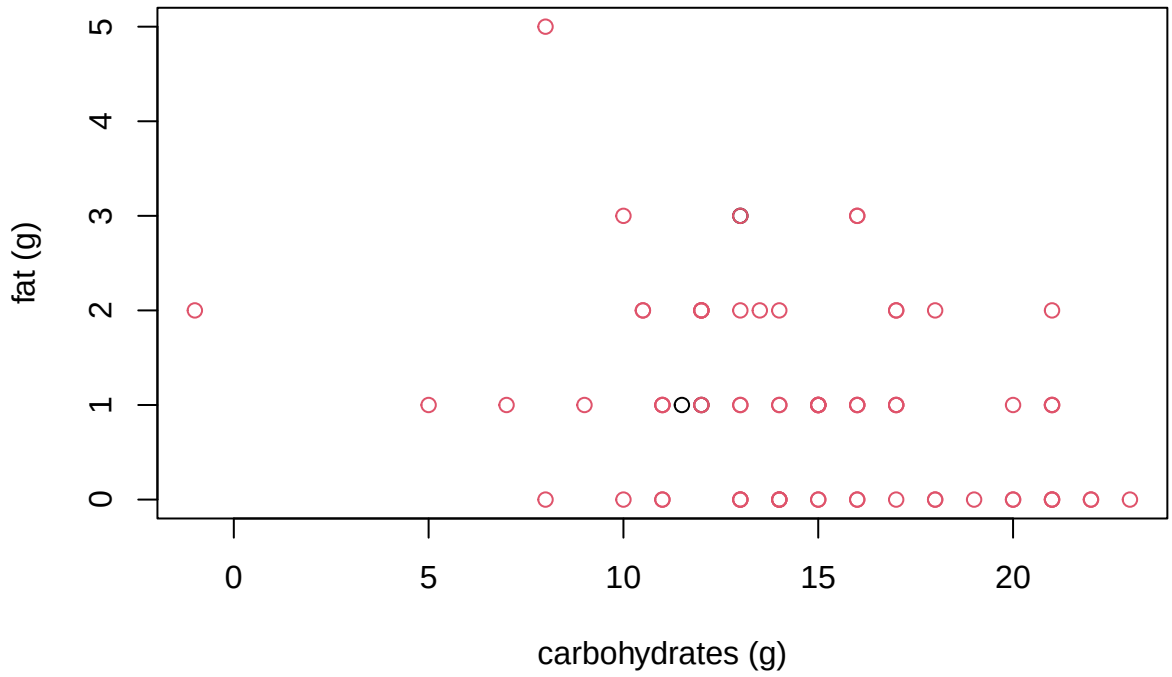
Exercise 5.1 (Cereal Plotting)**.** Let's put everything together with the cereal data set.

1. Create a new column in the original `cereal` data frame called `is_favorite`. `is_favorite` should be `"favorite"` if it was one of your favorites, and `not favorite` if it was not. (Hint: you've done something very similar to this a few exercises ago.)
2. Create a scatter plot of the carbohydrates on the x-axis vs. the calories on the y-axis. Make sure the axes are labeled appropriately!
3. Now, choose any two nutritional variables (calories, protein, fat, etc.), and create a scatter plot of these two variables. Color the points based on if they were your favorite or not. (You'll have to convert `is_favorite` into a factor with `is_factor()`).
4. Let's examine the fiber content of the cereals by making a `boxplot` using the `boxplot()` command.
5. Finally, let's save our modified cereal data. Use `write.csv()` to create a new file called `cereal_favorites.csv`!

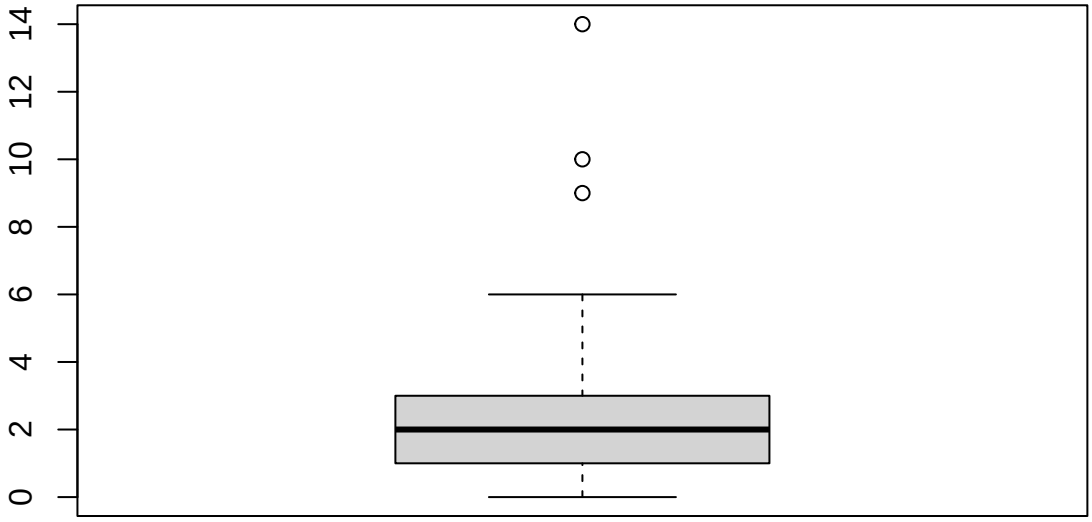
```
# 1
cereal$is_favorite = ifelse(names_in_favs, "favorite", "not favorite")
# 2
plot(x = cereal$carbo, y = cereal$calories,
     xlab = "carbohydrates (g)", ylab = "calories")
```




```
# 3
plot(x = cereal$carbo, y = cereal$fat,
     xlab = "carbohydrates (g)", ylab = "fat (g)",
     col = factor(cereal$is_favorite))
```




```r
# 4
boxplot(cereal$fiber)
```




```r
# 4
write.csv(cereal, "cereal_favorites.csv")
```