STAT 33A Workbook 9

CJ HINES (3034590053)

Oct 29, 2020

This workbook is due **Oct 29, 2020** by 11:59pm PT.

The workbook is organized into sections that correspond to the lecture videos for the week. Watch a video, then do the corresponding exercises *before* moving on to the next video.

Workbooks are graded for completeness, so as long as you make a clear effort to solve each problem, you'll get full credit. That said, make sure you understand the concepts here, because they're likely to reappear in homeworks, quizzes, and later lectures.

As you work, write your answers in this notebook. Answer questions with complete sentences, and put code in code chunks. You can make as many new code chunks as you like.

In the notebook, you can run the line of code where the cursor is by pressing `Ctrl` + `Enter` on Windows or `Cmd` + `Enter` on Mac OS X. You can run an entire code chunk by clicking on the green arrow in the upper right corner of the code chunk.

Please do not delete the exercises already in this notebook, because it may interfere with our grading tools.

You need to submit your work in two places:

- Submit this Rmd file with your edits on bCourses.
- Knit and submit the generated PDF file on Gradescope.

If you have any last-minute trouble knitting, **DON'T PANIC**. Submit your Rmd file on time and follow up in office hours or on Piazza to sort out the PDF.

Functions

Watch the "Functions" lecture video.

Exercise 1

Write a vectorized version of the `is_leap` function.

Show that your function works correctly for a few test cases.

YOUR ANSWER GOES HERE:

```r
is_leap = function(x) {
  result = logical(length(x))
  result[x %% 4 == 0 & x %% 100 != 0] = TRUE
  result[x %% 400 == 0] = TRUE
  result
}

#Tests
years1 = c(2000, 1001, 1300, 1800, 12, 13, 1)
years2 = c(4, 2020, 50)
years3 = c(200, 3000)

is_leap(years1) # [TRUE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE]
```

```
## [1]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE
```

```
is_leap(years2) # [TRUE, TRUE, FALSE]
```

```
## [1]  TRUE  TRUE FALSE
```

```
is_leap(years3) # [FALSE, FALSE]
```

```
## [1] FALSE FALSE
```

Exercise 2

An *anonymous function* is a function that isn't assigned to a variable (and so doesn't have a name). Anonymous functions are occasionally useful as arguments to other functions.

For example, it's common to write an anonymous functions when using `lapply` or `sapply`. The basic syntax is:

```
result = lapply(DATA, function(elt) {
  # elt is a single element of DATA.
  #
  # Your code goes here.
})
```

Consider this data frame:

```
nums = data.frame(
  x = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
  y = seq(6, by = 2, length.out = 10),
  z = c(2, 4, 1, 2, 12, 3, 3, 1, 2, 5)
)
```

Use `sapply` and an anonymous function to compute the number of even elements in each column.

Hint 1: Remember that you can use `sum` to count `TRUE`s in a logical vector.

Hint 2: If you're stuck on this exercise, try doing the rest of the exercises first to get more experience with writing functions.

YOUR ANSWER GOES HERE:

```
yes = sapply(nums, function(ok) {
  ok %% 2 == 0
})

sum(yes)
```

```
## [1] 20
```

Default Arguments

Watch the “Default Arguments” lecture video.

Exercise 3

Write a function `to_kelvin()` to convert temperatures in degrees Celsius to temperatures in degrees Kelvin (you can find the formula online). Your function should have a parameter `celsius` for the temperature to convert.

Make sure that your function is vectorized in `celsius`, so that it can convert an entire vector of temperatures in one call.

Show that your function works correctly for a few test cases.

YOUR ANSWER GOES HERE:

```
to_kelvin = function(celsius) {
  kelvin = numeric(length(celsius))
  kelvin = celsius + 273.15
  kelvin
}

#Tests
c1 = 50
c2 = c(1, 100, 15)
c3 = c(10, 30, 1000, 500, 12, 21, 2, 35)

to_kelvin(c1) # 323.15
```

```
## [1] 323.15
```

```
to_kelvin(c2) # [274.15, 373.15, 288.15]
```

```
## [1] 274.15 373.15 288.15
```

```
to_kelvin(c3) # [283.15, 303.15, 1273.15, 773.15, 285.15, 294.15, 275.15, 308.15]
```

```
## [1]  283.15  303.15 1273.15  773.15  285.15  294.15  275.15  308.15
```

Function Example

Watch the “Function Example” lecture video.

No exercises for this section.

Variables: Scope & Lookup

Watch the “Variables: Scope & Lookup” lecture video.

No exercises for this section.

Using Functions

Watch the “Using Functions” lecture video.

Exercise 4

A better design for `to_kelvin` is to have a `temp` parameter and a `unit` parameter. The function can then check `unit` and choose an appropriate conversion for `temp`. Using this design, rewrite `to_kelvin` so that it can convert to Kelvin from either Celsius or Fahrenheit. Make sure the function is still vectorized in the `temperature` parameter.

The `unit` parameter should be restricted to supported units, which in this case should be `"celsius"` and `"fahrenheit"`. You can use the `match.arg` function to check that an argument matches against a set of candidate arguments.

For instance, `match.arg(x, c("red", "blue"))` checks that the argument to `x` is either `"red"` or `"blue"`. The function also allows for partial matches, so for instance `"r"` matches `"red"`. See the documentation for full details.

Show that your function works correctly for a few test cases.

YOUR ANSWER GOES HERE:

```
to_kelvin = function(temp, unit) {
  result = numeric(length(temp))
  unit = match.arg(unit, c("celsius", "fahrenheit"), several.ok = TRUE);
  if (unit == "celsius") {
    result = temp + 273.15
    } else if (unit == "fahrenheit") {
      result = (temp + 459.67)*(5/9)
      }
  result
}

#Tests
temps1 = c(1, 5, 14, 30, 100)
temps2 = 10
temps3 = c(200, 45, 70)

to_kelvin(temps1, "celsius") # [274.15, 278.15, 287.15, 303.15, 373.15]
```

```
## [1] 274.15 278.15 287.15 303.15 373.15
```

```r
to_kelvin(temps2, "celsius") # 283.15
```

```
## [1] 283.15
```

```r
to_kelvin(temps3, "fahrenheit") # [366.4833, 280.3722, 294.2611]
```

```
## [1] 366.4833 280.3722 294.2611
```