STAT 33B Workbook 8

Ming Fong (3035619833)

Oct 22, 2020

This workbook is due **Oct 22, 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.

Printing Output

Watch the "Printing Output" lecture video.

Exercise 1

The `scan` function provides a way to collect *input* from the user.

Write a program that loops endlessly and:

1. Asks the user to input their name.
2. Uses `scan` to collect their input.
3. Prints `"Hello, NAME!"` where `NAME` is replaced by the collected name.

The program should exit the loop (and quit) if the user enters the name `quit`.

YOUR ANSWER GOES HERE:

```
while (TRUE) {
  name = scan(what  = character(), n = 1)
  if (name == "quit") {
    break
```

```
  }
  cat("Hello,", name)
}
```

Messages, Warnings, and Errors

Watch the "Messages, Warnings, and Errors" lecture video.

Exercise 2

A vectorized implementation of the `to_kelvin` function from Workbook 6 is:

```
to_kelvin = function(temperature, unit) {
  unit = match.arg(unit, c("celsius", "fahrenheit"), several.ok = TRUE)

  # First convert Fahrenheit to Celsius.
  is_f = unit == "fahrenheit"
  temperature[is_f] = (temperature[is_f] - 32) * 5 / 9

  temperature + 273.15
}
```

Write a modified version of `to_kelvin` that checks for potential problems. In particular, your version should check the assumptions that:

- `temperature` is numeric.
- `temperature` and `unit` are the same length, or `unit` has length 1.

Your version should raise an error (with a descriptive message) if either of these assumptions don't hold.

Test your function to show that it checks for potential problems. You can use `error = TRUE` on an RMarkdown code chunk to allow errors when knitting.

YOUR ANSWER GOES HERE:

```
to_kelvin = function(temperature, unit) {
  unit = match.arg(unit, c("celsius", "fahrenheit"), several.ok = TRUE)

  # Check for errors in params
  if (class(temperature) != "numeric") {
    stop("temperature must be numeric")
  }
  if (length(temperature) != length(unit) | length(unit) != 1) {
    stop("unit must be the same length as temperature or be length 1")
  }

  # First convert Fahrenheit to Celsius.
  is_f = unit == "fahrenheit"
  temperature[is_f] = (temperature[is_f] - 32) * 5 / 9

  temperature + 273.15
}
to_kelvin(temperature = c(1, 2, 3), unit = c("c", "c"))
```

```
## Error in to_kelvin(temperature = c(1, 2, 3), unit = c("c", "c")): unit must be the same length as te
```

```
to_kelvin(temperature = "Apple pie", unit = "c")
```

```
## Error in to_kelvin(temperature = "Apple pie", unit = "c"): temperature must be numeric
```

Handling Errors

Watch the "Handling Errors" lecture video.

No exercises for this section.

Global Options

Watch the "Global Options" lecture video.

Exercise 3

Skim `?options` and `?Startup`.

Create or edit your `.Rprofile` file to set an option (or several options).

Check that your option is actually set when you restart R (you can call `options` without any arguments to see your current options).

In your answer here, describe which option you set and include the code you added to `.Rprofile`.

YOUR ANSWER GOES HERE:

I changed the `warn` option to 1 as recommended by the professor. I also printed a welcome message on start.

```
# Put your new .Rprofile code here.
# Things you might want to change

# options(papersize="a4")
# options(editor="notepad")
# options(pager="internal")

# set the default help type
# options(help_type="text")
  options(help_type="html")

# set a site library
# .Library.site <- file.path(chartr("\\", "/", R.home()), "site-library")

# set a CRAN mirror
# local({r <- getOption("repos")
#       r["CRAN"] <- "http://my.local.cran"
#       options(repos=r)})

# Give a fortune cookie, but only to interactive sessions
# (This would need the fortunes package to be installed.)
#  if (interactive())
#    fortunes::fortune()

options(warn = 1)

.First = function() {
```

```
  cat("Welcome Ming Fong", date(), "\n")
}
```

Preventing Bugs

Watch the "Preventing Bugs" lecture video.

No exercises for this section.

The R Debugger

Watch the "The R Debugger" lecture video.

No exercises for this section.

Other Debugging Functions

Watch the "Other Debugging Functions" lecture video.

No exercises for this section. You're done!