STAT 29000 Project 8

## Topics: R, Shiny

**Motivation:** Sometimes it is useful to break down your shiny app into smaller parts. The source function allows us to do this. We can import and use custom functions and code simply by using source. In addition, reactive expressions are expressions that allow you to control when different parts of your app update. Often times, you will want to “react” to a change in data by outputting something new. This is where reactive expressions are useful.

**Context:** We’ve jumped right into building shiny apps. We’ve learned about the different parts of a shiny app, how to control the layout, etc. In this project, we will continue to practice what we learned before, as well as introduce the source function and reactive expressions.

**Scope:** Using the shiny package to build simple web apps.

Rstudio provides *excellent* examples that are instantly available for you to play around with:

library(shiny)  
runExample("01\_hello") # a histogram  
runExample("02\_text") # tables and data frames  
runExample("03\_reactivity") # a reactive expression  
runExample("04\_mpg") # global variables  
runExample("05\_sliders") # slider bars  
runExample("06\_tabsets") # tabbed panels  
runExample("07\_widgets") # help text and submit buttons  
runExample("08\_html") # Shiny app built from HTML  
runExample("09\_upload") # file upload wizard  
runExample("10\_download") # file download wizard  
runExample("11\_timer") # an automated timer

By running one of the examples you are immediately presented with an app for you to test out, as well as the associated (copy-and-pastable) code that you can use to run the app yourself.

The [written tutorial](https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/) does a much better job of introducing shiny than an examples file would do. For each problem or sub-problem, I will provide you with the lesson(s) that you will be required to read in order to be able to finish the problem or sub-problem.

Use the template found [here](https://datamine.purdue.edu/seminars/spring2020/stat29000project08template.R) or on scholar:

/class/datamine/data/spring2020/stat29000project08template.R

to submit your solutions.

After each problem, we’ve provided you with a list of keywords. These keywords could be package names, functions, or important terms that will point you in the right direction when trying to solve the problem, and give you accurate terminology that you can further look into online. You are not required to utilize all the given keywords. You will receive full points as long as your code gives the correct result.

## Question 1:

For this project you will submit a single app.R file that contains the answers (as comments) to all of the sub-problems of question 1. In addition, it will contain a single ui\_2 fluidPage, a single server\_2 server function, and a line that runs: shinyApp(ui = ui\_2, server = server\_2). Lastly, you will also submit a modified rt.R file.

Read [lesson 5](https://shiny.rstudio.com/tutorial/written-tutorial/lesson5/).

**1a.** *(.5 pt)* When, and how many times does the shinyApp function run?

**Item(s) to submit:**

* A single comment answering the question in the app.R file.

**1b.** *(.5 pt)* When, and how many times does the server function run?

**Item(s) to submit:**

* A single comment answering the question in the app.R file.

**1c.** *(.5 pt)* When, and how many times does any function starting with “render” run? For example, renderPlot.

**Item(s) to submit:**

* A single comment answering the question in the app.R file.

**1d.** *(.5 pt)* Explain what the source function does in 1-2 sentences.

**Item(s) to submit:**

* A single comment answering the question in the app.R file.

**1e.** *(.5 pt)* The source function requires a path to a file or filename. In your shiny app, when commands like source are run in the app.R file, where does shiny assume the file is relative to the app.R file?

**Item(s) to submit:**

* A single comment answering the question in the app.R file.

## Question 2:

Read [lesson 6](https://shiny.rstudio.com/tutorial/written-tutorial/lesson6/).

**2a.** *(2 pts)* We’ve provided you with a concept version of an app [here](https://datamine.purdue.edu/seminars/spring2020/stat29000project08question02.R) or on scholar:

/class/datamine/data/spring2020/stat29000project08question02.R. Copy and paste the contents after your solutions to question 1 in your app.R file.

In addition, we’ve provided you with an R script that comes pre-loaded with functions that scrape various parts of <https://rottentomatoes.com> [here](https://datamine.purdue.edu/seminars/spring2020/rt.R) or on scholar:

/class/datamine/data/spring2020/rt.R. Copy and paste this into the same folder as your app.R file.

Given a rotten tomatoes id, our app scrapes some things from <https://rottentomatoes.com/>, and displays them. Run the app and test out a few rotten tomatoes ids, just to get the feel for it: wall\_e, moana\_2016, holes.

As it is a concept version, we could definitely make some improvements. For example, the need to know what a movie’s rotten tomatoes id is, is certainly not ergonomic. Here is a function that will try and “guess” the rotten tomatoes id given a search term:

# this is a function that, given a search term like "lion king"  
# returns a guess for the rotten tomatoe id  
guess\_id <- function(search\_term) {  
 url <- paste("https://www.rottentomatoes.com/napi/search/?query=",   
 gsub(" ", "%20", search\_term), "&offset=0&limit=1", sep="")  
 json <- fromJSON(url)  
  
 # strip the /m/  
 rt\_id <- gsub("/m/", "", json$movies$url)  
 return(rt\_id)  
}

Test it out a few times:

guess\_id("lion king") # the\_lion\_king  
guess\_id("lion king 2") # the\_lion\_king\_ii\_simbas\_pride  
guess\_id("dragon tattoo") # the\_girl\_with\_the\_dragon\_tattoo

What this function is doing is simple. It is using the rotten tomatos search feature and “guessing” that the top result is the result you were looking for. Not perfect, but better.

Include this function in your imported (via source) rt.R file. Modify your app to accept a search term (separated by spaces) instead of a rotten tomatoes id. Be sure to update or remove your default value, and the label for that input.

**Hint:** *You should be able to achieve this by modifying only a single line in your ui\_2, and a single line in your server\_2.*

**Item(s) to submit:**

* Your modified rt.R file.
* Your modified app.R file.

**2b.** *(2.5 pts)* Some people don’t think that the tomatometer scores accurately represent what makes a good movie. Let’s create a reactive expression that creates a new score when the search term changes, or when the tomatometer\_weight changes. The datamine\_score will be a simple weighted average of the audience score and tomatometer score. Here is what you need to do:

1. Add an input that accepts numbers, with an inputId named tomatometer\_weight. See [here](https://shiny.rstudio.com/gallery/widget-gallery.html) for a list of widgets to look at.
2. Using the tomatometer\_weight and the scraped ratings, create a reactive expression that wraps the following calculation:

# strip % sign from scores  
tomatometer\_score <- gsub("%", "", tomatometer\_score)  
audience\_score <- gsub("%", "", audience\_score)  
  
# convert to numeric  
tomatometer\_score <- as.numeric(tomatometer\_score)  
audience\_score <- as.numeric(audience\_score)  
  
# calculate "datamine\_score"  
datamine\_score <- tomatometer\_weight\*tomatometer\_score + (1-tomatometer\_weight)\*audience\_score  
  
# convert back to a percentage  
datamine\_score <- paste0(datamine\_score, "%")

As you can see, this reactive expression will “react” to a change in either the tomatometer\_weight *or* the search term.

1. Create an output just like the output$tomatometer\_score and output$audience\_score called output$datamine\_score.
2. Render you output just like your render the other scores.
3. Don’t forget to show your output in ui\_2``fluidPage.

**Item(s) to submit:**

* Your modified app.R file.

**2c.** *(3 pts)* The app is a little bit better now, great! What it really need now is a makeover. As you can see, the ui\_2 is *very* simple. Use your creativity to reorganize the app and make it more user-friendly and visually-appealing. At a minimum, do the following:

1. *(1 pt)* Add a short 1-2 sentence description of what the app does.
2. *(.25 pts)* Modify the title to be less boring.
3. *(.75 pts)* Add instructions for the user.
4. *(1 pt)* Add labels that convey what the input and output is. For instance, the movie “Wall-e” has a datamine\_score of 92.5%, but the users don’t know what that number is! We need to label things.
5. (optional) Use fluidRow and column to add a structured layout to the app.

Feel free to make any other advancements you think would be useful or fun to do.

**Item(s) to submit:**

* Your modified app.R file.

## Project Submission:

Submit your solutions for the project at this URL: <https://classroom.github.com/a/9w_ofHxH> using the instructions found in the GitHub Classroom instructions folder on Blackboard.

**Important note:** For this project you will submit a single app.R file that contains the answers (as comments) to all of the sub-problems of question 1. In addition, it will contain a single ui\_2 fluidPage, a single server\_2 server function, and a line that runs: shinyApp(ui = ui\_2, server = server\_2). Lastly, you will also submit a modified rt.R file.