````
---
title: "Shiny_AE"
author: "Guangyu Ding"
date: "2024-11-15"
output: word_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

```{r}
library(tidyverse)
library(shiny)
library(microbenchmark)
library(reactable)
library(vroom)
library(devtools)
```
Problem 1
```{r}
Hadley_1 <- microbenchmark({
  ui <- fluidPage(
  selectInput("dataset", label = "Dataset", choices = ls("package:datasets")),
  verbatimTextOutput("summary"),
  tableOutput("table")
)
server <- function(input, output, session) {
  output$summary <- renderPrint({
    dataset <- get(input$dataset, "package:datasets")
    summary(dataset)
  })

  output$table <- renderTable({
    dataset <- get(input$dataset, "package:datasets")
    dataset
  })
}
shinyApp(ui, server)
})
summary(Hadley_1)
Hadley_2 <- microbenchmark({
ui <- fluidPage(
  selectInput("dataset", label = "Dataset", choices = ls("package:datasets")),
  verbatimTextOutput("summary"),
  tableOutput("table")
)
````
````

```
server <- function(input, output, session) {
  dataset <- reactive({
    get(input$dataset, "package:datasets")
  })

  output$summary <- renderPrint({
    summary(dataset())
  })

  output$table <- renderTable({
    dataset()
  })
}
shinyApp(ui, server)
})
summary(Hadley_2)

```


Problem 2
Exercise 2.3.5

1
a.renderPrint(summary(mtcars)) should be paired with verbatimTextOutput since it is console output.
b.renderText("Good morning!") should be paired with textOutput since it is regular text.
c.renderPrint(t.test(1:5, 2:6)) should be paired with verbatimTextOutput since it is console output.
d.renderText(str(lm(mpg ~ wt, data = mtcars))) should be paired with verbatimTextOutput since it is console output.

2
```{r}
library(shiny)

ui <- fluidPage(
    plotOutput("plot", width = "700px", height = "300px")
)

server <- function(input, output, session) {
    output$plot <- renderPlot(plot(1:5), res = 96,
                              alt = "Scatterplot of 5 random numbers")
}

shinyApp(ui, server)
```

3
```{r}
library(shiny)

ui <- fluidPage(
    dataTableOutput("table")
)

server <- function(input, output, session) {
    output$table <- renderDataTable(mtcars,
                                    options = list(pageLength = 5,
                                                   ordering = FALSE,
                                                   searching = FALSE))
}

shinyApp(ui, server)
```

4
```{r}
library(shiny)
library(reactable)

ui <- fluidPage(
  reactableOutput("table")
)

server <- function(input, output) {
  output$table <- renderReactable({
    reactable(mtcars)
  })
}

shinyApp(ui, server)
```

Exercise 3.3.6

1
```{r}
#input$greeting -> output$greeting
#Inside renderText, name -> input$name
#Fixed code:
server1 <- function(input, output, server) {
  output$greeting <- renderText(paste0("Hello ", input$name))
}
```

```
#Make greeting a reactive: greeting <- reactive(paste0("Hello ", input$name))
#Since greeting is now a reactive, add parenthesis around it: output$greeting <-
renderText(greeting())
#Fixed code:
server2 <- function(input, output, server) {
  greeting <- reactive(paste0("Hello ", input$name))
  output$greeting <- renderText(greeting())
}

#Spelling error: output$greting -> output$greeting
#Missing renderText()
#Fixed code:
server3 <- function(input, output, server) {
  output$greeting <- renderText(paste0("Hello ", input$name))
}
```


2
```{r}
# Server 1
#c <- reactive(input$a + input$b)
#e <- reactive(c() + input$d)
#output$f <- renderText(e())

#input$a --> c --> e --> output$f
#input$b --> c
#input$d --> e

# Server 2
#x <- reactive(input$x1 + input$x2 + input$x3)
#y <- reactive(input$y1 + input$y2)
#output$z <- renderText(x() / y())

#input$x1 --> x --> output$z
#input$x2 --> x
#input$x3 --> x
#input$y1 --> y --> output$z
#input$y2 --> y

# Server 3
#d <- reactive((c()) ^ input$d)
#a <- reactive(input$a * 10)
#c <- reactive(b() / input$c)
#b <- reactive(a() + input$b)

#input$a --> a --> b --> c --> d
```

```
#input$b --> b
#input$c --> c
#input$d --> d
```

3
range is a base R function used to calculate the range of a vector, and var is a base R function used to calculate the variance of a numeric vector. Naming reactives range and var causes confusion and potential unexpected behavior when these names are used elsewhere in the app, as the reactive objects would override the base functions.

Exercise 4.8

1
```{r}
# input$code --> selected --> diag
#                   |       body_part
#                   |       location
#                   |
#                   +--> summary --> age_sex

# input$code --> selected --> diag
#                   |         body_part
#                   |         location
#                   |
#                   +--> summary --> age_sex
# input$y --> age_sex

# input$code --> selected --> diag
#                   |         body_part
#                   |         location
#                   +--> summary --> age_sex
#                   +--> narrative
# input$y --> summary
# input$story --> narrative
```

2
```{r}
library(tidyverse)
dir.create("neiss")
#> Warning in dir.create("neiss"): 'neiss' already exists
download <- function(name) {
  url <- "https://github.com/hadley/mastering-shiny/raw/master/neiss/"
  download.file(paste0(url, name), paste0("neiss/", name), quiet = TRUE)
}
download("injuries.tsv.gz")
```

```
download("population.tsv")
download("products.tsv")


injuries <- vroom::vroom("neiss/injuries.tsv.gz")
injuries

# Original code
injuries %>%
  mutate(diag = fct_lump(fct_infreq(diag), n = 5)) %>%
  group_by(diag) %>%
  summarise(n = as.integer(sum(weight)))

# Flipped code
injuries %>%
  mutate(diag = fct_infreq(fct_lump(diag, n = 5))) %>%
  group_by(diag) %>%
  summarise(n = as.integer(sum(weight)))
```

3
```{r}
library(dplyr)
library(ggplot2)
library(forcats)
library(vroom)
library(shiny)

injuries <- vroom::vroom("neiss/injuries.tsv.gz")
products <- vroom::vroom("neiss/products.tsv")
population <- vroom::vroom("neiss/population.tsv")


ui <- fluidPage(
  fluidRow(
    column(8,
           selectInput("code", "Product",
                       choices = setNames(products$prod_code, products$title),
                       width = "100%"
           )
    ),
    column(2, selectInput("y", "Y axis", c("rate", "count"))),
    # lets the user decide how many rows to show in the summary tables
    column(2, numericInput("num_rows", "Number of Rows", value = 5, min = 0, max = 6))
  ),
  fluidRow(
    column(4, tableOutput("diag")),
```

```
    column(4, tableOutput("body_part")),
    column(4, tableOutput("location"))
  ),
  fluidRow(
    column(12, plotOutput("age_sex"))
  ),
  fluidRow(
    column(2, actionButton("story", "Tell me a story")),
    column(10, textOutput("narrative"))
  )
)

count_top <- function(df, var, n = 5) {
  df %>%
    mutate({{ var }} := fct_lump(fct_infreq({{ var }}), n = n)) %>%
    group_by({{ var }}) %>%
    summarise(n = as.integer(sum(weight)))
}

server <- function(input, output, session) {
  selected <- reactive(injuries %>% filter(prod_code == input$code))

  output$diag <- renderTable(count_top(selected(), diag) %>% slice(1:input$num_rows),
width = "100%")
  output$body_part    <-    renderTable(count_top(selected(),    body_part)    %>%
slice(1:input$num_rows), width = "100%")
  output$location    <-    renderTable(count_top(selected(),    location)    %>%
slice(1:input$num_rows), width = "100%")

  summary <- reactive({
    selected() %>%
      count(age, sex, wt = weight) %>%
      left_join(population, by = c("age", "sex")) %>%
      mutate(rate = n / population * 1e4)
  })

  output$age_sex <- renderPlot({
    if (input$y == "count") {
      summary() %>%
        ggplot(aes(age, n, colour = sex)) +
        geom_line() +
        labs(y = "Estimated number of injuries")
    } else {
      summary() %>%
        ggplot(aes(age, rate, colour = sex)) +
        geom_line(na.rm = TRUE) +
        labs(y = "Injuries per 10,000 people")
```

```
    }
  }, res = 96)

  narrative_sample <- eventReactive(
    list(input$story, selected()),
    selected() %>% pull(narrative) %>% sample(1)
  )
  output$narrative <- renderText(narrative_sample())
}

shinyApp(ui, server)
```

4

```{r}
library(shiny)
library(forcats)
library(dplyr)
library(ggplot2)

count_top <- function(df, var, n = 5) {
  df %>%
    mutate({{ var }} := fct_lump(fct_infreq({{ var }}), n = n)) %>%
    group_by({{ var }}) %>%
    summarise(n = as.integer(sum(weight)))
}

ui <- fluidPage(
  fluidRow(
    column(8, selectInput("code", "Product",
                          choices = setNames(products$prod_code, products$title),
                          width = "100%")
    ),
    column(2, numericInput("rows", "Number of Rows",
                           min = 1, max = 10, value = 5)),
    column(2, selectInput("y", "Y Axis", c("rate", "count")))
  ),
  fluidRow(
    column(4, tableOutput("diag")),
    column(4, tableOutput("body_part")),
    column(4, tableOutput("location"))
  ),
  fluidRow(
    column(12, plotOutput("age_sex"))
  ),
  fluidRow(
    column(2, actionButton("prev_story", "Previous story")),
```

```
    column(2, actionButton("next_story", "Next story")),
    column(8, textOutput("narrative"))
  )
)

server <- function(input, output, session) {
  selected <- reactive(injuries %>% filter(prod_code == input$code))

  # Find the maximum possible of rows.
  max_no_rows <- reactive(
    max(length(unique(selected()$diag)),
        length(unique(selected()$body_part)),
        length(unique(selected()$location)))
  )

  # Update the maximum value for the numericInput based on max_no_rows().
  observeEvent(input$code, {
    updateNumericInput(session, "rows", max = max_no_rows())
  })

  table_rows <- reactive(input$rows - 1)

  output$diag <- renderTable(
    count_top(selected(), diag, n = table_rows()), width = "100%")

  output$body_part <- renderTable(
    count_top(selected(), body_part, n = table_rows()), width = "100%")

  output$location <- renderTable(
    count_top(selected(), location, n = table_rows()), width = "100%")

  summary <- reactive({
    selected() %>%
      count(age, sex, wt = weight) %>%
      left_join(population, by = c("age", "sex")) %>%
      mutate(rate = n / population * 1e4)
  })

  output$age_sex <- renderPlot({
    if (input$y == "count") {
      summary() %>%
        ggplot(aes(age, n, colour = sex)) +
        geom_line() +
        labs(y = "Estimated number of injuries") +
        theme_grey(15)
    } else {
      summary() %>%
```

```
        ggplot(aes(age, rate, colour = sex)) +
        geom_line(na.rm = TRUE) +
        labs(y = "Injuries per 10,000 people") +
        theme_grey(15)
    }
  })

  # Store the maximum posible number of stories.
  max_no_stories <- reactive(length(selected()$narrative))

  # Reactive used to save the current position in the narrative list.
  story <- reactiveVal(1)

  # Reset the story counter if the user changes the product code.
  observeEvent(input$code, {
    story(1)
  })

  # When the user clicks "Next story", increase the current position in the
  # narrative but never go beyond the interval [1, length of the narrative].
  # Note that the mod function (%%) is keeping `current`` within this interval.
  observeEvent(input$next_story, {
    story((story() %% max_no_stories()) + 1)
  })

  # When the user clicks "Previous story" decrease the current position in the
  # narrative. Note that we also take advantage of the mod function.
  observeEvent(input$prev_story, {
    story(((story() - 2) %% max_no_stories()) + 1)
  })

  output$narrative <- renderText({
    selected()$narrative[story()]
  })
}

shinyApp(ui, server)
```

Acknowledgement

I would like to express my deepest appreciation to my peer Songyu Tang for his invaluable contributions and collaborative spirit throughout this assignment. Our collective efforts and synergistic teamwork have significantly enhanced the quality and depth of this study.