Shiny

2024-11-15

What is the difference between Hadley_1 and Hadley_2? Use the functions Katia showed last Wednesday to investigate the difference.

```
library(shiny)
ui <- fluidPage(
  selectInput("dataset", label = "Dataset", choices = ls("package:datasets")),
  verbatimTextOutput("summary"),
  tableOutput("table")
)
```

Hadley_1

```
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)
```

Dataset

ability.cov

```
       Length Class  Mode
cov    36     -none- numeric
center  6     -none- numeric
n.obs   1     -none- numeric
```

cov.general	cov.picture	cov.blocks	cov.maze	cov.reading	cov.vocab	center	n.
24.64	5.99	33.52	6.02	20.75	29.70	0.00	11
5.99	6.70	18.14	1.78	4.94	7.20	0.00	11
33.52	18.14	149.83	19.42	31.43	50.75	0.00	11
6.02	1.78	19.42	12.71	4.76	9.07	0.00	11
20.75	4.94	31.43	4.76	52.60	66.76	0.00	11

Hadley_2

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

  output$summary <- renderPrint({
    # Use a reactive expression by calling it like a function
    summary(dataset())
  })

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

ability.cov

```
       Length Class  Mode
cov    36     -none- numeric
center  6     -none- numeric
n.obs   1     -none- numeric
```

cov.general	cov.picture	cov.blocks	cov.maze	cov.reading	cov.vocab	center	n.
24.64	5.99	33.52	6.02	20.75	29.70	0.00	11
5.99	6.70	18.14	1.78	4.94	7.20	0.00	11
33.52	18.14	149.83	19.42	31.43	50.75	0.00	11
6.02	1.78	19.42	12.71	4.76	9.07	0.00	11
20.75	4.94	31.43	4.76	52.60	66.76	0.00	11
29.70	7.20	50.75	9.07	66.76	135.29	0.00	11

```
library(microbenchmark)

input <- list(dataset = "mtcars")

microbenchmark(
  Hadley_1 = {
    dataset1 <- get(input$dataset, "package:datasets")
    summary(dataset1)
    dataset1
  },
  Hadley_2 = {
    dataset2 <- get(input$dataset, "package:datasets") # No need for reactive()
    summary(dataset2)
    dataset2
  }
)
```

```
## Unit: milliseconds
##      expr    min     lq     mean  median      uq    max neval
##  Hadley_1 2.3305 2.3658 2.517903 2.41180 2.52865 5.1407   100
##  Hadley_2 2.3272 2.3711 2.543557 2.42695 2.55795 4.5086   100
```

Hadley_1 performs better in terms of raw speed due to the absence of the reactive overhead. However, this comes at the cost of code redundancy and maintainability.

Hadley_2, while slightly slower, offers better scalability and efficiency for Shiny apps with shared or complex dependencies.

Prepare Chapters 2-4 from Mastering Shiny. complete in submit the homework in sections 2.3.5, 3.3.6, and 4.8.

2.3.5

1.

a verbatimTextOutput()

b textOutput()

c verbatimTextOutput()

d textOutput()

2.

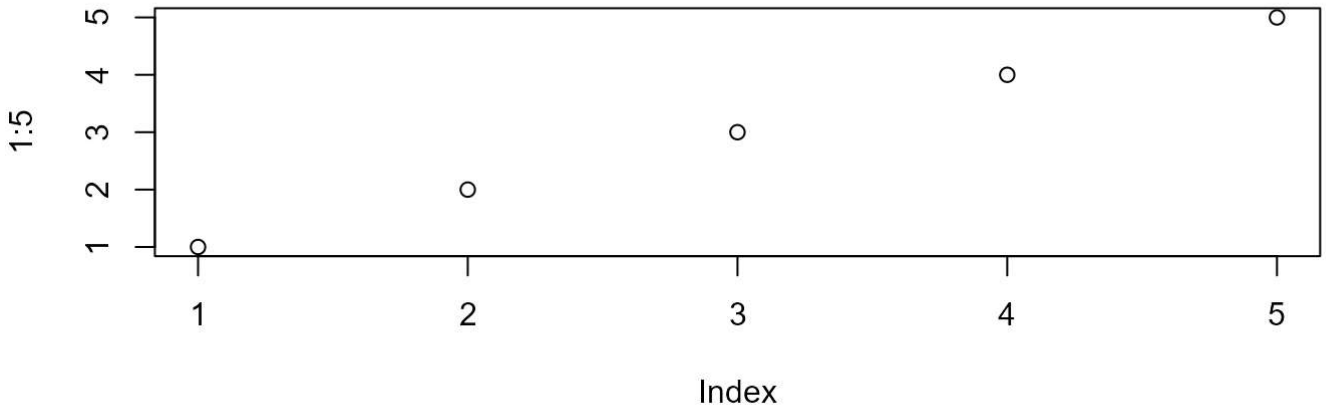
```
library(shiny)

ui <- fluidPage(
  # Hidden text description for screen readers
  tags$div(id = "plotDescription", style = "position: absolute; left: -9999px;",
           "Scatterplot of five random numbers from 1 to 5"),

  # Wrap plotOutput with a div that includes aria-describedby
  tagList(
    tags$div(
      plotOutput("plot", width = "700px", height = "300px"),
      `aria-describedby` = "plotDescription"
    )
  )
)

server <- function(input, output, session) {
  output$plot <- renderPlot({
    plot(1:5)
  }, res = 96)
}

shinyApp(ui, server)
```



3.

```
library(shiny)
library(DT)
```

```
##
## 载入程序包：'DT'
```

```
## The following objects are masked from 'package:shiny':
##
##     dataTableOutput, renderDataTable
```

```
ui <- fluidPage(
  DTOutput("table")
)

server <- function(input, output, session) {
  output$table <- renderDataTable({
    datatable(mtcars, options = list(
      pageLength = 5,       # Number of rows to display initially
      searching = FALSE,    # Remove search box
      ordering = FALSE,     # Disable column sorting
      paging = FALSE,       # Remove pagination controls
      info = FALSE          # Remove table information
    ))
  })
}

shinyApp(ui, server)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.46	20.22	1	0	3	1
Duster 360	14.3	8	360	245	3.21	3.57	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.19	20	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.15	22.9	1	0	4	2

4.

```
library(shiny)
library(reactable)
```

```
## Warning: 程序包'reactable'是用R版本4.4.2 来建造的
```

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

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

shinyApp(ui, server)
```

	mpg	cyl	disp	hp	drat
Mazda RX4	21	6	160	110	3.9
Mazda RX4 Wag	21	6	160	110	3.9
Datsun 710	22.8	4	108	93	3.85
Hornet 4 Drive	21.4	6	258	110	3.08
Hornet Sportabout	18.7	8	360	175	3.15
Valiant	18.1	6	225	105	2.76
Duster 360	14.3	8	360	245	3.21
Merc 240D	24.4	4	146.7	62	3.69
Merc 230	22.8	4	140.8	95	3.92

3.3.6

1.

```
#Fix code
server1 <- function(input, output, server) {
  output$greeting <- renderText(paste0("Hello ", input$name))
}

server2 <- function(input, output, server) {
  greeting <- reactive(paste0("Hello ", input$name))
  output$greeting <- renderText(greeting())
}

server3 <- function(input, output, server) {
  output$greeting <- renderText(paste0("Hello ", input$name))
}
```

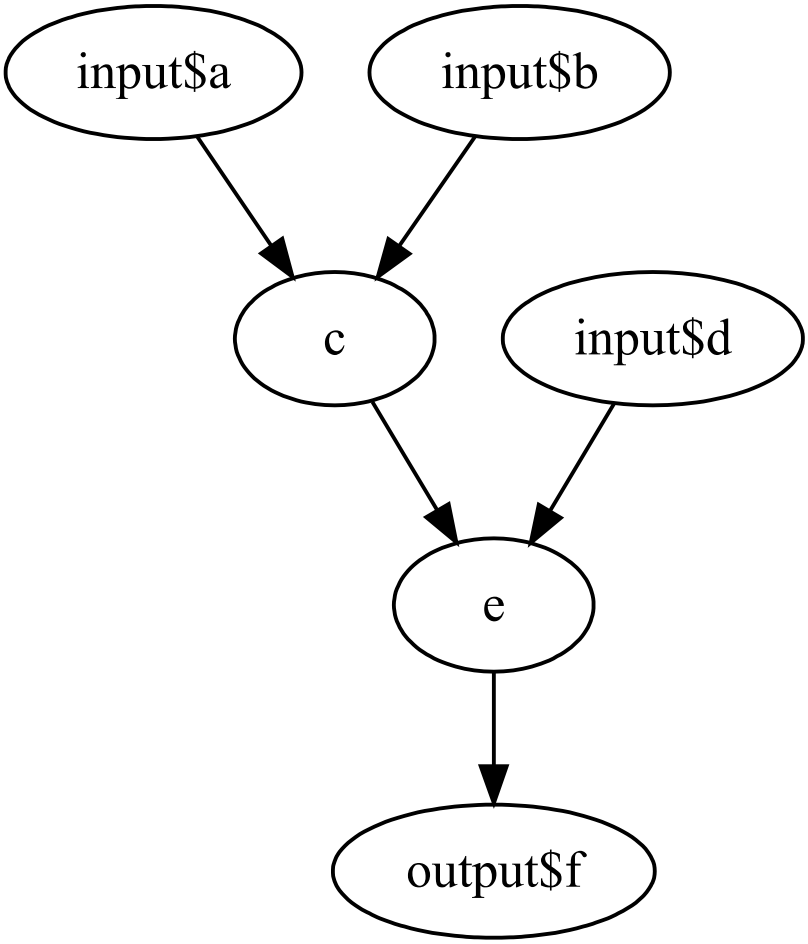
2.

```
library(DiagrammeR)
```

```
## Warning: 程序包'DiagrammeR'是用R版本4.4.2 来建造的
```

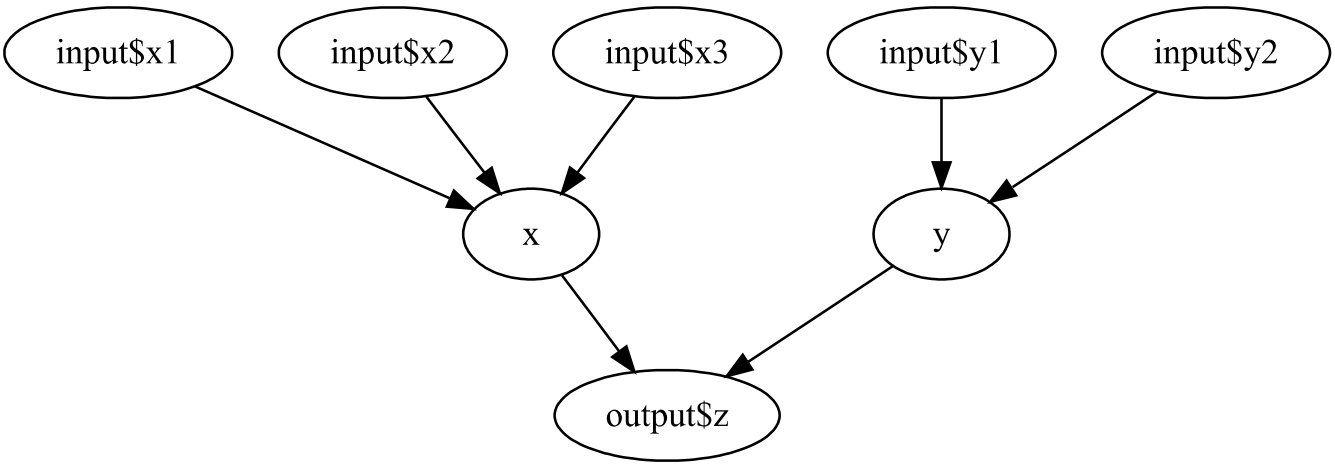
```
grViz("
digraph server1 {
  graph [layout = dot]

  'input$a' -> 'c'
  'input$b' -> 'c'
  'c' -> 'e'
  'input$d' -> 'e'
  'e' -> 'output$f'
}
")
```



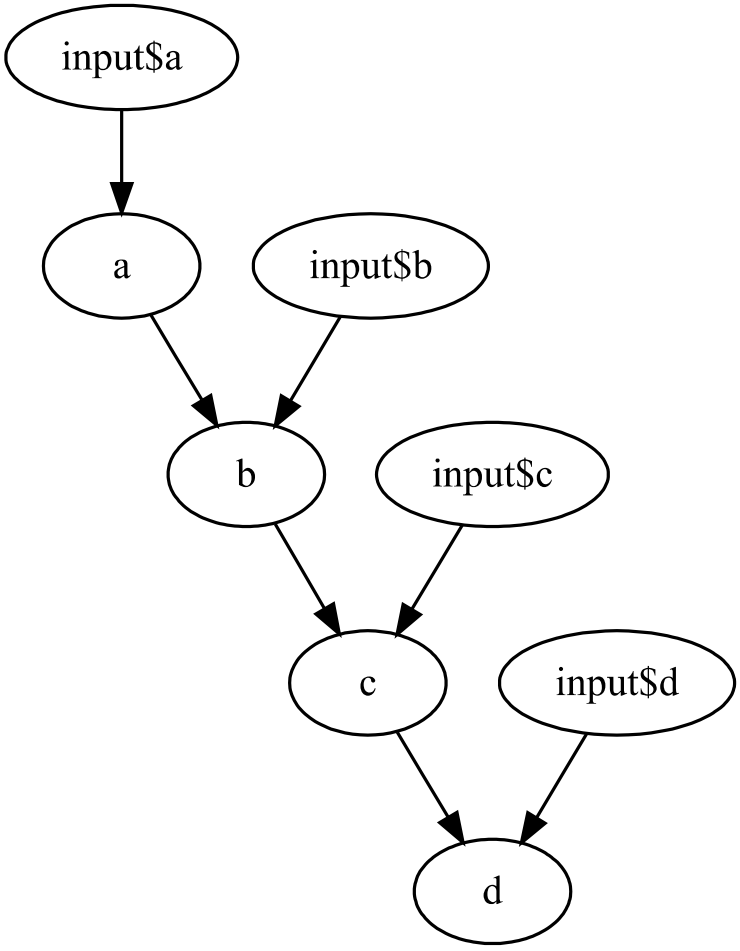
```
grViz("
digraph server2 {
  graph [layout = dot]

  'input$x1' -> 'x'
  'input$x2' -> 'x'
  'input$x3' -> 'x'
  'x' -> 'output$z'
  'input$y1' -> 'y'
  'input$y2' -> 'y'
  'y' -> 'output$z'
}
")
```



```
grViz("
digraph server3 {
  graph [layout = dot]

  'input$a' -> 'a'
  'a' -> 'b'
  'input$b' -> 'b'
  'b' -> 'c'
  'input$c' -> 'c'
  'c' -> 'd'
  'input$d' -> 'd'
}
")
```

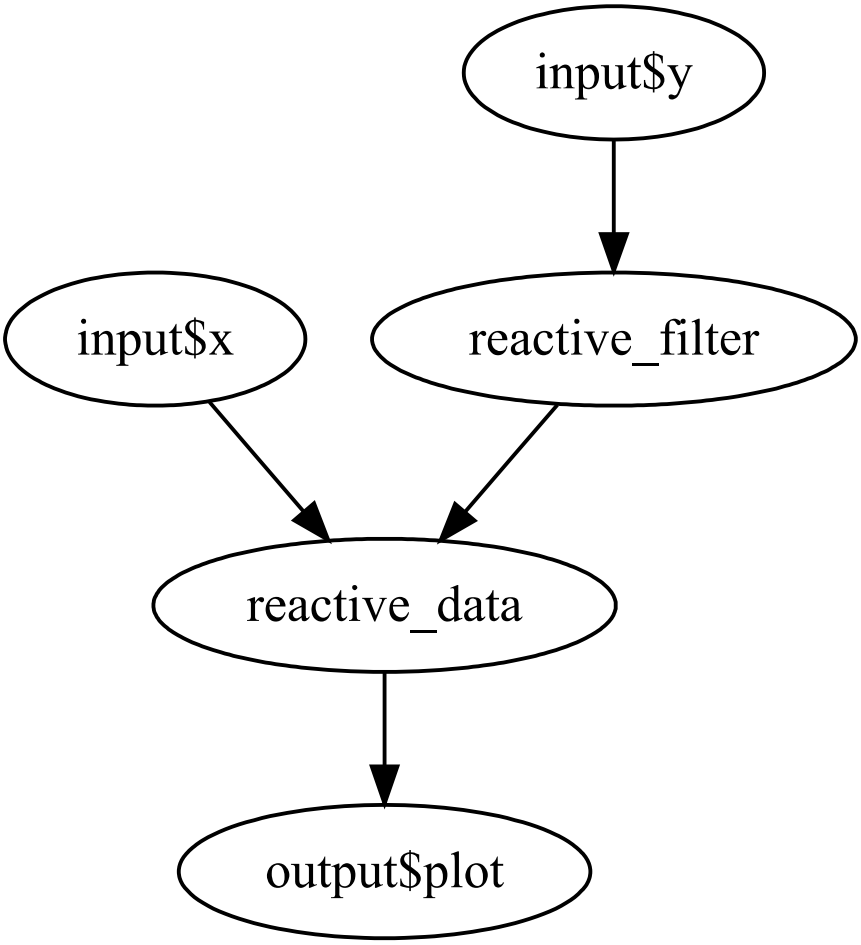


3. This code doesn't work because we called our reactive range, so when we call the range function we're actually calling our new reactive. If we change the name of the reactive from range to col_range then the code will work.

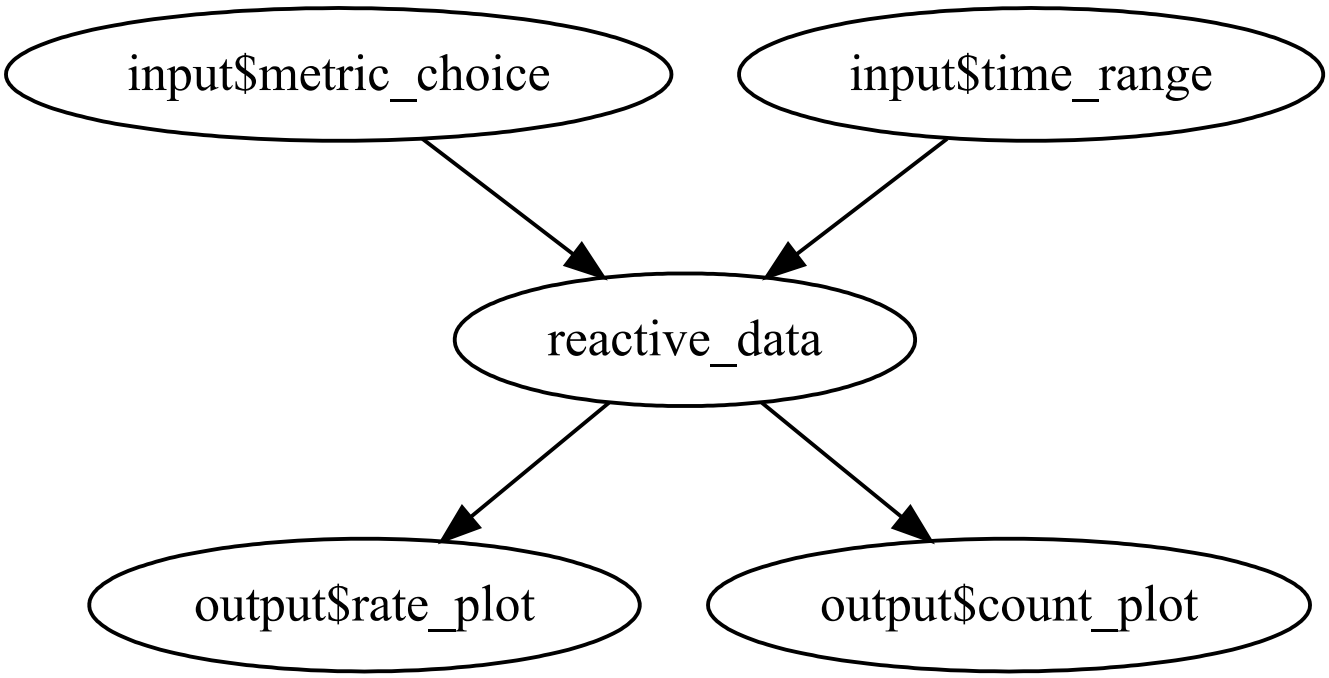
4.8

1.

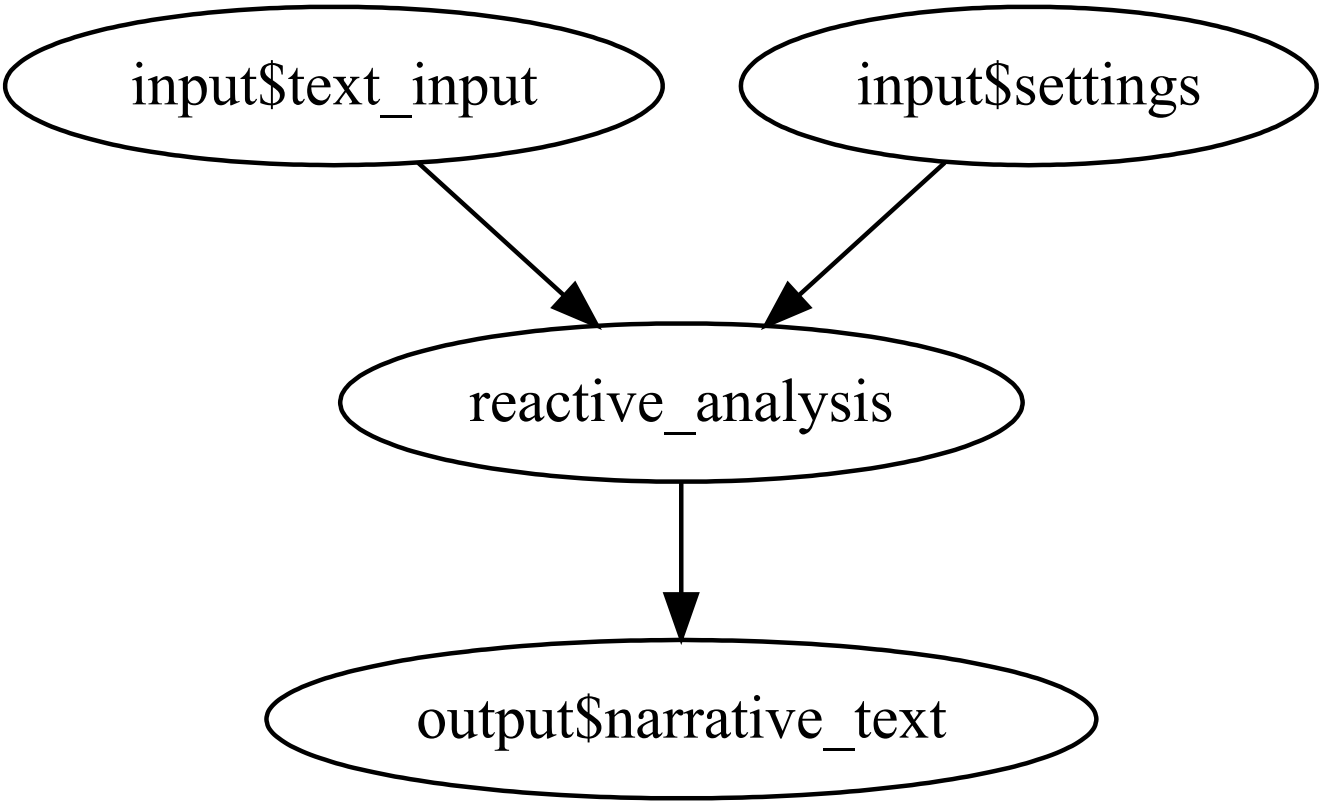
```
library(DiagrammeR)
grViz("
  digraph Prototype {
    'input$x' -> 'reactive_data';
    'input$y' -> 'reactive_filter';
    'reactive_filter' -> 'reactive_data';
    'reactive_data' -> 'output$plot';
  }
")
```




```
grViz("
  digraph Rate_vs_Count {
    'input$metric_choice' -> 'reactive_data';
    'input$time_range' -> 'reactive_data';
    'reactive_data' -> 'output$rate_plot';
    'reactive_data' -> 'output$count_plot';
  }
")
```




```
grViz("
  digraph Narrative {
    'input$text_input' -> 'reactive_analysis';
    'input$settings' -> 'reactive_analysis';
    'reactive_analysis' -> 'output$narrative_text';
  }
")
```



2. Flipping the order of fct_infreq() and fct_lump() will only change the factor levels order. In particular, the function fct_infreq() orders the factor levels by frequency, and the function fct_lump() also orders the factor levels by frequency but it will only keep the top n factors and label the rest as Other.

3.

```
library(shiny)
library(forcats)
library(dplyr)
```

```
##
## 载入程序包：'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)

dir.create("neiss")
```

```
## Warning in dir.create("neiss"): 'neiss'已存在
```

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

```
## Rows: 38 Columns: 2
```

```
## ── Column specification ────────────────────────────────────────────────────────
────────────────────────────
## Delimiter: "\t"
## chr (1): title
## dbl (1): prod_code
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

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

```
## Rows: 255064 Columns: 10
## ── Column specification ────────────────────────────────────────────────────────
────────────────────────────
## Delimiter: "\t"
## chr  (6): sex, race, body_part, diag, location, narrative
## dbl  (3): age, prod_code, weight
## date (1): trmt_date
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
population <- vroom::vroom("neiss/population.tsv")
```

```
## Rows: 170 Columns: 3
## ── Column specification ────────────────────────────────────────────────────
────────────────────────────
## Delimiter: "\t"
## chr (1): sex
## dbl (2): age, population
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
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("story", "Tell me a story")),
    column(10, 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)
    }
  })

  output$narrative <- renderText({
    input$story
    selected() %>% pull(narrative) %>% sample(1)
  })
}

shinyApp(ui, server)
```

Product

knives, not elsewhere classified

Number of Rows

5

Y Axis

rate

diag	n
Laceration	287925
Avulsion	9970
Puncture	5870
Other Or Not Stated	4852
Other	4969

4.

```
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)
```

Product

knives, not elsewhere classified

Number of Rows

5

Y Axis

rate

diag	n
Laceration	287925
Avulsion	9970
Puncture	5870
Other Or Not Stated	4852
Other	4969