PH 718 Data Management and Visualization in R

Part 6: Interactive Plotting

Zhiyang Zhou (zhou67@uwm.edu, zhiyanggeezhou.github.io)

2025/04/14 21:16:20

Examples of interactive plotting

Example 1: Interactive 3D plot of surface z=x^2 + y^2 (also in Part 4: Introduction to tidyverse)

```
x <- seq(-pi, pi, length = 50)
y <- x
z_matrix <- outer(x, y, function(x, y) x^2 + y^2)

plotly::plot_ly(
  x = ~x, y = ~y, z = ~z_matrix,
  type = "surface",
  colorscale = "Viridis"
) |> plotly::layout(
  title = "3D Surface Plot: f(x, y) = x^2 + y^2",
  scene = list(
    xaxis = list(title = "X-axis"),
    yaxis = list(title = "Y-axis"),
    zaxis = list(title = "Z-axis")
  )
)
```

Example 2: Interactive scatter plot deployed online

https://shiny.posit.co/

Example 3: Interactive maps developed by the US Census Bureau

https://www.census.gov/programs-surveys/geography/data/interactive-maps.html

Why interactive plotting is beneficial

- Enhances user engagement by allowing direct interaction with visualizations.
- Improves exploration of complex data through dynamic filtering, zooming, and detailed tooltips, uncovering patterns and insights not immediately apparent from static plots.
- Facilitates the presentation of plots without code adjustments.

With acknowledgment to Dr. Kourosh Ravvaz for sharing course materials.

`plotly::ggplotly`: converting `ggplot2` figures to interactive ones

Interactive scatter plot

```
library(tidyverse)
p <- ggplot(mtcars, aes(x = wt, y = mpg, color = factor(cyl))) +
  geom_point(size = 3) +
  labs(title = "Scatter plot of MPG vs Weight",
       x = "Weight (1000 lbs)",
       y = "Miles per Gallon",
       color = "Cylinders") +
  theme_minimal()
p  # Basic scatter plot (static)
plotly::ggplotly(p) # Convert to an interactive plot
```

The above interactive scatter plot allows:

- Hovering over points to see exact values.
- Zooming and panning dynamically.
- Interactive legend for highlighting different cylinder groups.

Interactive scatter plot with customized tooltips

```
p <- mtcars |>
  ggplot(aes(
    x = wt, y = mpg, color = factor(cyl),
    text = paste("Model:", rownames(mtcars), "<br>", # HTML tag <br> to initiate a new line
                 "Cylinders:", cyl, "<br>",
                 "Horsepower:", hp, "<br>",
                 "Transmission:", ifelse(am == 0, "Automatic", "Manual"), "<br>",
                 "Gear:", gear, "<br>",
                 "Carburetors:", carb)
  )) +
  geom_point(size = 3) +
  labs(title = "Scatter plot of MPG vs Weight",
       x = "Weight (1000 lbs)",
       y = "Miles per Gallon",
       color = "Cylinders") +
  theme_minimal()
plotly::ggplotly(
  p,
  # tooltip = c("text", "x", "y") # Hiding factor(cyl)
)
```

Interactive histogram

```
hist_plot <- ggplot(mtcars, aes(x = mpg, fill = factor(cyl))) +
  geom_histogram(alpha = 0.8, bins = 10) +
  labs(title = "Histogram of MPG by Cylinders",
       x = "Miles per Gallon",
       y = "Count",
       fill = "Cylinders") +
  theme_minimal()
plotly::ggplotly(hist_plot)
```

Interactive boxplot

```
box_plot <- ggplot(mtcars, aes(x = factor(cyl), y = mpg, fill = factor(cyl))) +
  geom_boxplot() +
  labs(title = "MPG distribution by Cylinders",
       x = "Cylinders",
       y = "Miles per Gallon") +
  theme_minimal()
plotly::ggplotly(box_plot)
```

Interactive line plot

```
line_plot <- ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line(color = "steelblue") +
  labs(title = "Unemployment over Time",
       x = "Year",
       y = "Number of Unemployed (thousands)") +
  theme_minimal()
plotly::ggplotly(line_plot)
```

Interactive jitter plot

```
p = starwars %>%
  filter(!is.na(height) & !is.na(mass) & mass < 1000) %>%
  ggplot(
    aes(x = mass, y = height, color = sex)
  ) +
  geom_point() +
  geom_jitter(
    width= .2, # each point is randomly moved up to 0.2 units left or right.
    height= .2, # each point is randomly moved up to 0.2 units up or down.
    alpha = .2, # the transparency
    color = "purple"  # color for jittered points
  ) +
  labs(
    title = "Height vs. Mass",
    x = "Mass (kg)", y = "Height (cm)"
  ) +
  theme_bw()
plotly::ggplotly(p)
```

Interactive histogram with frequency polygon

```
p = starwars %>%
  filter(!is.na(height)) %>%
  ggplot(aes(x = height)) +
  geom_histogram(bins = 10, fill = "steelblue", color = "black") +
  geom_freqpoly(bins = 10, color = "orange", size = 1) +
  labs(title = "Distribution of Character Heights",
       x = "Height (cm)", y = "Count") +
  theme_minimal()
plotly::ggplotly(p)
```

Interactive stacked bar plot

```
p = starwars %>%
  ggplot(aes(
    x = fct_rev(species), # Reverses the order of species to correct flipping
    fill = sex
  )) +
  geom_bar(position = "stack") +  # position = "fill" or "stack"
  labs(title = "Stacked Bar Chart",
       x = "Species", y = "Count") +
  theme_minimal() +
  coord_flip()
plotly::ggplotly(p)
```

Interactive density plot

```
p = starwars %>%
  filter(!is.na(mass) & !is.na(sex) & mass<1000) %>%
  ggplot(aes(x = mass, fill = sex)) +
  geom_density(alpha = 0.6) + # alpha controls the transparency
  labs(title = "Density Plot of Mass by sex",
       x = "Mass (kg)",
       y = "Density") +
  theme_minimal()
plotly::ggplotly(p)
```

Interactive violin plot with a box plot inside

```
p = starwars %>%
  ggplot(aes(x = sex, y = height, fill = sex)) +
  geom_violin(alpha = 0.5) +
  geom_boxplot(width = 0.1) +
  labs(title = "Violin Plot with Box Plot",
       x = "sex", y = "Height (cm)") +
  theme_minimal()
plotly::ggplotly(p)
```

Interactive heatmap

```
cor_df <- as.data.frame(cor(mtcars)) %>%
  rownames_to_column(var = "Var1") %>%
  pivot_longer(cols = -Var1, names_to = "Var2", values_to = "Correlation")
cor_df <- cor_df %>%
  mutate(Var2 = factor(Var2, levels = rev(sort(unique(Var2)))))
p = ggplot(cor_df, aes(Var1, Var2, fill = Correlation)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red") +
  labs(title = "Heatmap of mtcars Correlation Matrix",
       x = "Variable", y = "Variable") +
  theme_minimal()
plotly::ggplotly(p)
```

`plotly::plot_ly`: creating custom interactive plots from scratch

Interactive scatter plot

```
plotly::plot_ly(
  data = mtcars,
  x = ~wt,
  y = ~mpg,
  type = "scatter", mode = "markers", # Scatter plot
  color = ~factor(cyl),           # Color by cylinder (categorical)
  marker = list(size = 10),       # Marker size
  text = ~paste("Cylinders:", cyl,
                "<br>MPG:", mpg,
                "<br>Weight:", wt),  # Tooltip
  hoverinfo = "text"              # Show only custom text on hover
) %>% plotly::layout(
    title = "Scatter plot of MPG vs Weight",
    xaxis = list(title = "Weight (1000 lbs)"),
    yaxis = list(title = "Miles per Gallon"),
    legend = list(title = list(text = "Cylinders"))
  )
```

Interactive histogram

```
plotly::plot_ly(
  data = mtcars,
  x = ~mpg,
  color = ~factor(cyl),
  type = "histogram",
  nbinsx = 20,
  opacity = 0.8
) %>% plotly::layout(
  title = "Histogram of MPG by Cylinders",
  xaxis = list(title = "Miles per Gallon"),
  yaxis = list(title = "Count"),
  barmode = "stack",  # stacked histogram
  legend = list(title = list(text = "Cylinders"))
)
```

Interactive boxplot

```
plotly::plot_ly(
  data = mtcars,
  x = ~factor(cyl),
  y = ~mpg,
  color = ~factor(cyl),
  type = "box"
) %>% plotly::layout(
  title = "MPG distribution by Cylinders",
  xaxis = list(title = "Cylinders"),
  yaxis = list(title = "Miles per Gallon"),
  legend = list(title = list(text = "Cylinders"))
)
```

Interactive line plot

```
plotly::plot_ly(
  data = economics,
  x = ~date,
  y = ~unemploy,
  type = "scatter",
  mode = "lines",
  line = list(color = "steelblue")
) %>% plotly::layout(
  title = "Unemployment over Time",
  xaxis = list(title = "Year"),
  yaxis = list(title = "Number of Unemployed (thousands)")
)
```

Interactive stacked bar plot

```
starwars %>%
  filter(!is.na(species), !is.na(sex)) %>%
  count(species, sex) %>%
  mutate(species = fct_rev(as.factor(species))) %>%
  plotly::plot_ly(
    x = ~n,
    y = ~species,
    color = ~sex,
    type = "bar",
    orientation = "h"
  ) %>% plotly::layout(
    barmode = "stack",
    title = "Stacked Bar Chart",
    xaxis = list(title = "Count"),
    yaxis = list(title = "Species"),
    legend = list(title = list(text = "Sex")),
    template = "plotly_white"
  )
```

Interactive violin plot with a box plot inside

```
starwars %>%
  filter(!is.na(height), !is.na(sex)) |>
  plotly::plot_ly(
    x = ~sex,
    y = ~height,
    type = "violin",
    split = ~sex,
    box = list(visible = TRUE),          # adds boxplot inside
    meanline = list(visible = TRUE),     # optional: shows mean
    fillcolor = "rgba(100,100,255,0.5)",# semi-transparent fill
    opacity = 0.5,
    line = list(color = "black"),
    showlegend = FALSE
  ) %>% plotly::layout(
    title = "Violin Plot with Box Plot",
```

```
    xaxis = list(title = "Sex"),
    yaxis = list(title = "Height (cm)"),
    template = "plotly_white"
  )
```

Interactive heatmap

```
cor_df <- as.data.frame(cor(mtcars)) %>%
  rownames_to_column(var = "Var1") %>%
  pivot_longer(cols = -Var1, names_to = "Var2", values_to = "Correlation")
cor_df <- cor_df %>%
  mutate(Var2 = factor(Var2, levels = rev(sort(unique(Var2)))))
plotly::plot_ly(
  data = cor_df,
  x = ~Var1,
  y = ~Var2,
  z = ~Correlation,
  type = "heatmap",
  colors = c("white", "red"),
  colorbar = list(title = "Correlation")
) %>% plotly::layout(
  title = "Heatmap of mtcars Correlation Matrix",
  xaxis = list(title = "Variable"),
  yaxis = list(title = "Variable")
)
```

Interactive 3D scatter plot

```
plotly::plot_ly(
  data = mtcars,
  x = ~wt,
  y = ~mpg,
  z = ~hp,
  type = "scatter3d",
  mode = "markers",
  color = ~factor(cyl)
)
```

Interactive 3D surface plot—visualizing a bivariate function

```
x <- seq(-10, 10, length.out = 50)
y <- x
z <- outer(x, y, function(x, y) cos(sqrt(x^2 + y^2)))
plotly::plot_ly(
  x = ~x, y = ~y, z = ~z,
  type = "surface"
) %>% plotly::layout(title = "3D Surface Plot")
```

Interactive geographic map: hightlighting coordinates

```
df <- data.frame(
  city = c("New York", "London", "Tokyo"),
```

```
  lat = c(40.7128, 51.5074, 35.6895),
  lon = c(-74.0060, -0.1278, 139.6917)
)
plotly::plot_ly(
  data = df,
  type = "scattergeo",
  mode = "markers",
  lat = ~lat,
  lon = ~lon,
  text = ~city,
  marker = list(size = 10, color = "blue")
) %>% plotly::layout(
  geo = list(
    projection = list(type = "orthographic"), "natural earth" or "orthographic"
    showland = TRUE
  ),
  title = "Cities on a Globe"
)
```

Interactive geographic map: hightlighting states in US

```
plotly::plot_ly(
  type = "choropleth",
  locations = c("USA", "CAN", "AUS"), # ISO 3166-1 alpha-3 codes
  locationmode = "ISO-3",
  z = c(1,2,3),  # c(1,2,3): means one color for each country
  colorscale = list(c(0, "lightgray"), c(1, "steelblue")),  # Color scale
  showscale = F
) |> plotly::layout(
  geo = list(
    showframe = T,
    showcoastlines = TRUE,
    projection = list(type = "orthographic")  #  "natural earth" or "orthographic"
  ),
  title = ""
)
```

Interactive geographic map: hightlighting Wisconsin on a USA Map

```
plotly::plot_ly(
  type = "choropleth",
  locations = c("WI", "AK"),
  locationmode = "USA-states",
  z = c(1,1),  # c(1,1): means one color for all
  colorscale = list(c(0, "lightgray"), c(1, "forestgreen")),  # Color scale
  showscale = F
) |> plotly::layout(
  geo = list(
    scope = "usa",
    showlakes = TRUE,
    lakecolor = "lightblue"
  ),
  title = "Wisconsin and Alaska"
```

```
)
```

A dashboard combining multiple interactive plots into a single layout

```
p1 <- plotly::plot_ly(
  mtcars,
  x = ~mpg,
  type = "histogram",
  name = "MPG"
)
p2 <- plotly::plot_ly(
  mtcars,
  x = ~wt, y = ~mpg,
  type = "scatter", mode = "markers",
  name = "MPG vs Weight"
)
p3 <- plotly::plot_ly(
  mtcars,
  x = ~cyl, y = ~hp,
  type = "box",
  name = "HP by Cylinders"
)
plotly::subplot(
  p1, p2, p3,
  nrows = 2, margin = 0.05,
  titleX = TRUE, titleY = TRUE
) %>% plotly::layout(
  title = "Mini Dashboard with plotly"
)
```

Summary: `plotly::plot_ly` vs `plotly::ggplotly`

Quick Comparison: `plot_ly()` vs `ggplotly()`

Feature	`plot_ly()`	`ggplotly()`
Style	Build from scratch	Convert from `ggplot2`
Syntax	Plotly-native	`ggplot2`-based
Flexibility	High (fine control)	Moderate (tied to `ggplot2`)
Tooltips	Custom via `text`, `tooltip`	From `aes(text = ...)`
Layout control	Full via `layout()`	Partial via `theme()`
Best for	Dashboards, 3D, maps, full control	Quick interactivity for `ggplot2` plots

When to use which?

- Use `ggplotly` when:
  - You already have a `ggplot2` plot and want to add interactivity
  - You want a fast way to enable zoom, pan, hover, and tooltips.
  - You're more comfortable with ggplot2 syntax.
- Use `plot_ly` when:
  - You want full control over every interactive aspect (like hover behavior, annotations, animations).
  - You're building web dashboards (e.g., with shiny) and want lightweight plots.
  - You're working with `plotly`-specific types like 3D plots and maps.

What is R Shiny

- An R package allowing users to create interactive web applications (apps) directly from R without the need for extensive web development knowledge
- References:
  - Mastering Shiny: https://mastering-shiny.org/
  - Cheatsheet: https://raw.githubusercontent.com/rstudio/cheatsheets/main/shiny.pdf

Key features of R Shiny

- Create sophisticated interactive web apps directly from R
- Integrate seamlessly with other languages like HTML, CSS, and JavaScript
- Facilitate easy sharing of apps both online and internally
- Supports reactive programming for dynamic content

Structure of a R Shiny app

- User interface (UI): defines the layout and appearance of the app
- Server: Contains R code and reactive expressions that generate the app's outputs based on user interactions

The first R Shiny app: interactive bar plot of WorldPhones data

Step 1

```
library(shiny)
# Use a fluid page layout
ui <- fluidPage()
# Define a server for the Shiny app
server <- function(input, output) {}
# Return a Shiny app object
shinyApp(ui = ui, server = server)
```

Step 2

```
library(shiny)
# Use a fluid page layout
ui <- fluidPage(
  # Give the page a title
  titlePanel("Telephones by region")
)
# Define a server for the Shiny app
server <- function(input, output) {}
# Return a Shiny app object
shinyApp(ui = ui, server = server)
```

Step 3

```
library(shiny)
# Use a fluid page layout
ui <- fluidPage(
  # Give the page a title
  titlePanel("Telephones by region"),
  # Layout a sidebar and main area
  sidebarLayout(
    # Define the sidebar
    sidebarPanel(
      selectInput(
        inputId = "region",
        label = "Region:",
        choices=colnames(WorldPhones)
      )
    ),
    # Create a spot for the main area
    mainPanel(
      plotOutput("phonePlot")
    )
  )
)
# Define a server for the Shiny app
server <- function(input, output) {
}
# Return a Shiny app object
shinyApp(ui = ui, server = server)
```

Step 4

```
library(shiny)
# Use a fluid page layout
ui <- fluidPage(
  # Give the page a title
  titlePanel("Telephones by region"),
  # Layout a sidebar and main area
  sidebarLayout(
    # Define the sidebar
    sidebarPanel(
      selectInput(
        inputId = "region",
        label = "Region:",
        choices=colnames(WorldPhones)
      )
    ),
    # Create a spot for the main area
    mainPanel(
      plotOutput("phonePlot")
    )
  )
)
# Define a server for the Shiny app
```

```
server <- function(input, output) {
  # Fill in the spot we created for a plot
  output$phonePlot <- renderPlot(
    # Render a barplot
    barplot(
      WorldPhones[,input$region]*1000,
      main=input$region,
      ylab="Number of Telephones",
      xlab="Year"
    )
  )
}
# Return a Shiny app object
shinyApp(ui = ui, server = server)
```

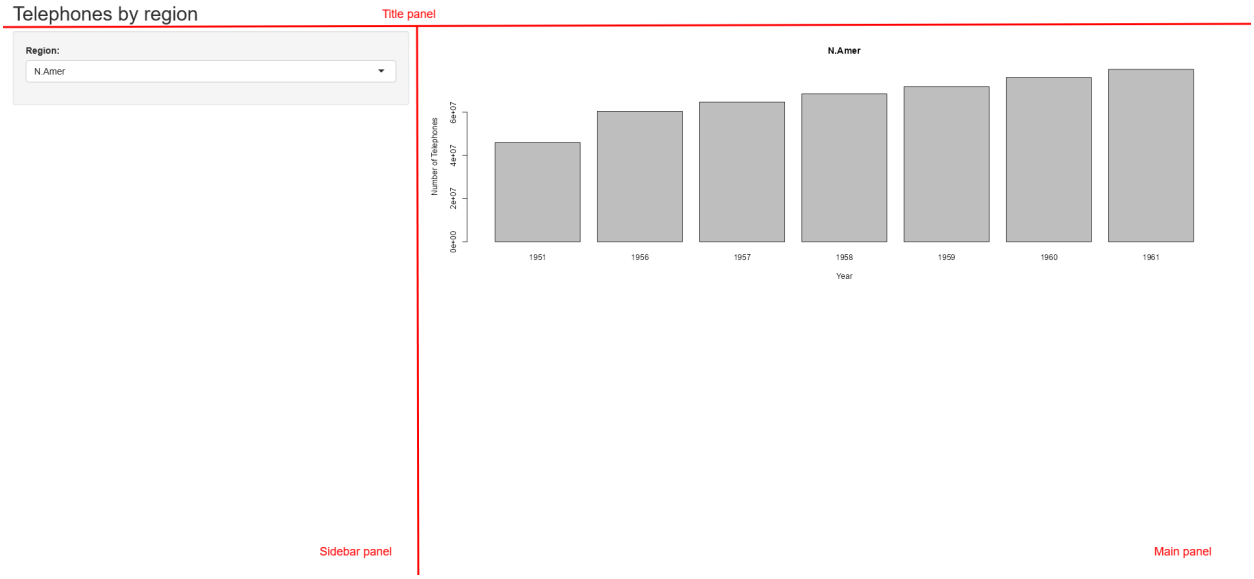

Figure 1: Three panels

Step 5

```
library(shiny)
# Use a fluid page layout
ui <- fluidPage(
  # Give the page a title
  titlePanel("Telephones by region"),
  # Layout a sidebar and main area
  sidebarLayout(
    # Define the sidebar
    sidebarPanel(
      selectInput(
        inputId = "region",
        label = "Region:",
        choices=colnames(WorldPhones)
      )
```

```
    ),
    # Create a spot for the main area
    mainPanel(
      plotOutput("phonePlot")
    )
  )
)
# Define a server for the Shiny app
server <- function(input, output) {
  # Fill in the spot we created for a plot
  output$phonePlot <- renderPlot(
    # Render a barplot
    barplot(
      WorldPhones[,input$region]*1000,
      main=input$region,
      ylab="Number of Telephones",
      xlab="Year"
    )
  )
}
# Return a Shiny app object
shinyApp(ui = ui, server = server)
```

Step 6

```
library(shiny)
# Use a fluid page layout
ui <- fluidPage(
  # Give the page a title
  titlePanel("Telephones by region"),
  # Layout a sidebar and main area
  sidebarLayout(
    # Define the sidebar
    sidebarPanel(
      selectInput(
        inputId = "region",
        label = "Region:",
        choices=colnames(WorldPhones)
      ),
      helpText("Data from AT&T (1961) The World's Telephones.")
    ),
    # Create a spot for the main area
    mainPanel(
      plotOutput("phonePlot")
    )
  )
)
# Define a server for the Shiny app
server <- function(input, output) {
  # Fill in the spot we created for a plot
  output$phonePlot <- renderPlot(
    # Render a barplot
    barplot(
```

```
      WorldPhones[,input$region]*1000,
      main = input$region,
      ylab = "Number of Telephones",
      xlab = "Year"
    )
  )
}
# Return a Shiny app object
shinyApp(ui = ui, server = server)
```

Step 7: if using bar plots generated via ggplot2

```
library(shiny)
library(tidyverse)
# Use a fluid page layout
ui <- fluidPage(
  # Give the page a title
  titlePanel("Telephones by region"),
  # Layout a sidebar and main area
  sidebarLayout(
    # Define the sidebar
    sidebarPanel(
      selectInput(
        inputId = "region",
        label = "Region:",
        choices=colnames(WorldPhones)
      ),
      helpText("Data from AT&T (1961) The World's Telephones.")
    ),
    # Create a spot for the main area
    mainPanel(
      plotOutput("phonePlot")
    )
  )
)
# Define a server for the Shiny app
server <- function(input, output) {
  # Fill in the spot we created for a plot
  output$phonePlot <- renderPlot(
    # Render a ggplot2 bar plot
    WorldPhones |>
      ggplot(aes(x = rownames(WorldPhones), y = WorldPhones[, input$region] * 1000)) +
      geom_bar(stat = "identity") +
      labs(
        title = input$region,
        x = "Year",
        y = "Number of Telephones"
      )
  )
}
# Return a Shiny app object
shinyApp(ui = ui, server = server)
```

Step 8: if using bar plots generated via `plot_ly`

```
library(shiny)
library(plotly)
# Use a fluid page layout
ui <- fluidPage(
  # Give the page a title
  titlePanel("Telephones by region"),
  # Layout a sidebar panel and main panel
  sidebarLayout(
    # Define the sidebar
    sidebarPanel(
      selectInput(
        inputId = "region",
        label = "Region:",
        choices=colnames(WorldPhones)
      ),
      helpText("Data from AT&T (1961) The World's Telephones.")
    ),
    # Create a spot for the main area
    mainPanel(
      plotlyOutput("phonePlot") # plotOutpu -> plotlyOutput
    )
  )
)
# Define a server for the Shiny app
server <- function(input, output) {
  # Fill in the spot we created for a plot
  output$phonePlot <- renderPlotly( # renderPlot -> renderPlotly
    # Render a plot_ly barplot
    plot_ly(
      x = rownames(WorldPhones),  # Years from row names
      y = WorldPhones[, input$region] * 1000,  # Data for selected region, scaled
      type = "bar",
      name = input$region
    ) %>% layout(
      title = input$region,
      xaxis = list(title = "Year"),
      yaxis = list(title = "Number of Telephones")
    )
  )
}
# Return a Shiny app object
shinyApp(ui = ui, server = server)
```

The second R Shiny app: interactive plots of `agridat::beaven.barley` data

```
library(shiny)
library(tidyverse)

Barley <- as.data.frame(agridat::beaven.barley)
ui <- fluidPage(
  titlePanel("Barley Yield"),
```

```
  sidebarLayout(
    sidebarPanel(
      selectInput(
        inputId = "gen",  # Give the input a name "genotype"
        label = "1. Select genotype",
        choices = c(
          "A" = "a",
          "B" = "b",
          "C" = "c",
          "D" = "d",
          "E" = "e",
          "F" = "f",
          "G" = "g",
          "H" = "h"
        ),
        selected = "a"
      ),
      selectInput(
        inputId = "color",
        label = "2. Select the color of histogram frame",
        choices = c("blue","green","red","purple","grey"),
        selected = "grey"
      ),
      selectInput(
        inputId = "fill",
        label = "3. Select the color of histogram bins",
        choices = c("blue","green","red","purple","grey"),
        selected = "grey"
      ),
      sliderInput(
        inputId = "bin",
        label = "4. Select number of histogram bins",
        min = 1, max= 25, step = 2, round = T,
        value= 10 # Initial value
      ),
      textInput(
        inputId = "text",
        label = "4. Enter some text to be displayed",
        value = "" # Initial value
      )
    ),
    mainPanel(
      plotOutput("myhist"),
      tableOutput("mytable"),
      textOutput("mytext")
    )
  )
)

server <- function(input, output) {
  output$myhist <- renderPlot(
    ggplot(Barley, aes(x = yield)) +
      geom_histogram(
```

```
        bins = input$bin, group=input$gen,
        data=Barley[Barley$gen == input$gen,],
        color = input$color,
        fill = input$fill
      )
    )
  output$mytext <- renderText(input$text)
  output$mytable <- renderTable({
    Barley %>%
      filter(gen == input$gen) %>%
      summarise(
        "Mean" = mean(yield),
        "Median" = median(yield),
        "STDEV" = sd(yield),
        "Min" = min(yield),
        "Max" = max(yield)
      )
    })
}

# Return a Shiny app object
shinyApp(ui = ui, server = server)
```

The third R Shiny app: visualizing movie ratings from IMDb

https://shiny.posit.co/r/getstarted/build-an-app/hello-shiny/getting-started.html

```
library(shiny)
library(tidyverse)

# Get the data
# An .RData file is a file format used in R to save and store R objects
# (such as data 3 frames, lists, vectors, functions, or entire workspaces)
# E.g., use save.image(file = "workspace.RData") to save the entire workspace.

url <- "https://github.com/rstudio-education/shiny-course/raw/main/movies.RData"
destfile <- "movies.RData"
download.file(url, destfile) # save movies.RData in a temporary dir
load(destfile)

ui <- fluidPage(sidebarLayout(
  sidebarPanel(
    selectInput(
      inputId = "x",
      label = "X-axis:",
      choices = c(
        "IMDB rating"          = "imdb_rating",
        "IMDB number of votes" = "imdb_num_votes",
        "Critics score"        = "critics_score",
        "Audience score"       = "audience_score",
        "Runtime"              = "runtime"
      ),
      selected = "audience_score"
    ),
```

```
    selectInput(
      inputId = "y",
      label = "Y-axis:",
      choices = c(
        "IMDB rating"          = "imdb_rating",
        "IMDB number of votes" = "imdb_num_votes",
        "Critics score"        = "critics_score",
        "Audience score"       = "audience_score",
        "Runtime"              = "runtime"
      ),
      selected = "critics_score"
    ),
    selectInput(
      inputId = "z",
      label = "Color by:",
      choices = c(
        "Title type" = "title_type",
        "Genre" = "genre",
        "MPAA rating" = "mpaa_rating",
        "Critics rating" = "critics_rating",
        "Audience rating" = "audience_rating"
      ),
      selected = "mpaa_rating"
    ),
    sliderInput(
      inputId = "opacity",
      label = "Opacity:",
      min = 0, max = 1,
      value = 0.5
    ),
    dateRangeInput(
      inputId = "date",
      label = "Select dates:",
      start = min(as.Date(movies$thtr_rel_date)),
      end = max(as.Date(movies$thtr_rel_date)),
      min = min(as.Date(movies$thtr_rel_date)),
      max = max(as.Date(movies$thtr_rel_date)),
      startview = "year"
    ),
    checkboxInput(
      inputId = "show_data",
      label = "Show data table",
      value = TRUE
    ),
    downloadButton(outputId = "download_data", label = "Download Data")
  ),
  mainPanel(
    plotOutput(outputId = "scatterplot"),
    plotOutput(outputId = "densityplot", height = 200),
    DT::dataTableOutput(outputId = "moviestable")
  )
))
```

```
server <- function(input, output) {
  output$scatterplot <- renderPlot({
    movies |> filter(
      thtr_rel_date >= as.POSIXct(input$date[1]) &
      thtr_rel_date <= as.POSIXct(input$date[2])
    ) |>  ggplot(
      aes(x = .data[[input$x]], y = .data[[input$y]], color = .data[[input$z]])
    ) +
    geom_point(alpha = input$opacity)
  })

  output$densityplot <- renderPlot({
    movies |> filter(
      thtr_rel_date >= as.POSIXct(input$date[1]) &
      thtr_rel_date <= as.POSIXct(input$date[2])
    ) |>
      ggplot(aes(x = .data[[input$x]])) +
      geom_density()
  })

  output$moviestable <- DT::renderDT({
    if(input$show_data == T){
      movies |> filter(
        thtr_rel_date >= as.POSIXct(input$date[1]) &
        thtr_rel_date <= as.POSIXct(input$date[2])
      ) |> DT::datatable(
        options = list(pageLength = 10), rownames = FALSE)
    }
  })

  output$download_data <- downloadHandler(
    filename = function() {
      paste("movies_filtered_", Sys.Date(), ".csv", sep = "")
    },
    content = function(file) {
      write.csv(
        movies |> filter(
          thtr_rel_date >= as.POSIXct(input$date[1]) &
          thtr_rel_date <= as.POSIXct(input$date[2])
        ),
        file,
        row.names = FALSE
      )
    }
  )
}

shinyApp(ui = ui, server = server)
```