PH 718 Data Management and Visualization in R

Part 8: Interactive Plotting

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

2026/05/08 00:30:06

Examples of interactive plotting

Example 1: Interactive 3D plot of surface $z = x^2 + y^2$

```
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.

`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 color
)
```

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 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 3D scatter plot

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

Mini exercise

Create an interactive scatter plot using `moderndive::evals` with:

- `x = bty_avg`
- `y = score`

- points colored by `gender`
- marker size set to 10
- a custom tooltip showing:
  - gender
  - beauty score
  - course evaluation score

Add an informative title and axis labels.

```
library(moderndive)
library(plotly)

plotly::plot_ly(
  data = evals,
  x = ~bty_avg,
  y = ~score,
  type = "scatter", mode = "markers",
  color = ~factor(gender),
  marker = list(size = 10),
  text = ~paste(
    "Gender:", gender,
    "<br>Beauty score:", bty_avg,
    "<br>Course score:", score
  ),
  hoverinfo = "text"
) |> plotly::layout(
  title = "Course Evaluation vs Beauty Rating",
  xaxis = list(title = "Beauty Rating (Average)"),
  yaxis = list(title = "Course Evaluation Score"),
  legend = list(title = list(text = "Gender"))
)
```

Modify the previous scatter plot:

- Change the x-axis to `age` and keep `y = score`.
- Update the tooltip accordingly.

```
library(moderndive)
library(plotly)

plotly::plot_ly(
  data = evals,
  x = ~age,
  y = ~score,
  type = "scatter", mode = "markers",
  color = ~gender,
  marker = list(size = 10),
  text = ~paste(
    "Gender:", gender,
    "<br>Age:", age,
    "<br>Course score:", score
  ),
  hoverinfo = "text"
) |> plotly::layout(
  title = "Course Evaluation vs Instructor Age",
```

```
  xaxis = list(title = "Age"),
  yaxis = list(title = "Course Evaluation Score"),
  legend = list(title = list(text = "Gender"))
)
```

Create a 3D scatter plot with:

- x = bty_avg
- y = age
- z = score
- points colored by gender

```
library(moderndive)
library(plotly)

plotly::plot_ly(
  data = evals,
  x = ~bty_avg,
  y = ~age,
  z = ~score,
  type = "scatter3d", mode = "markers",
  color = ~gender,
  marker = list(size = 5),
  text = ~paste(
    "Gender:", gender,
    "<br>Beauty score:", bty_avg,
    "<br>Age:", age,
    "<br>Course score:", score
  ),
  hoverinfo = "text"
) |> plotly::layout(
  title = "3D Scatter Plot of Course Evaluation, Beauty, and Age",
  legend = list(title = list(text = "Gender"))
)
```

Interactive heatmap

```
library(tidyverse)

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")
)
```

Mini exercise

Task:

- Select only numeric variables from `evals` using `evals_num = evals |> select(where(is.numeric))`
- Compute the correlation matrix.
- Convert to long format.
- Create an interactive heatmap.

```
library(moderndive)
library(tidyverse)
library(plotly)

# Select only numeric variables
evals_num = evals |> select(where(is.numeric))

# Compute correlation matrix and reshape to long format
cor_df = evals_num |>
  cor() |>
  as.data.frame() |>
  rownames_to_column(var = "Var1") |>
  pivot_longer(
    cols = -Var1,
    names_to = "Var2",
    values_to = "Correlation"
  )

# Reorder Var2 so the heatmap reads more naturally
cor_df <- cor_df |>
  mutate(Var2 = factor(Var2, levels = rev(sort(unique(Var2)))))

# Create interactive heatmap
plotly::plot_ly(
  data = cor_df,
  x = ~Var1,
  y = ~Var2,
  z = ~Correlation,
  type = "heatmap",
  colors = colorRamp(c("blue", "white", "red")),
  text = ~paste(
    "Variable 1:", Var1,
    "<br>Variable 2:", Var2,
    "<br>Correlation:", round(Correlation, 2)
  ),
  hoverinfo = "text",
  colorbar = list(title = "Correlation")
```

```
)
```

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 stacked bar plot

```
library(tidyverse)
starwars |>
  filter(!is.na(species), !is.na(sex)) |>
  count(species, sex) %>%
  mutate(species = fct_rev(as.factor(species))) |>
  plotly::plot_ly(
    y = ~species,
    x = ~n,
    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 boxplot

```
plotly::plot_ly(
  data = mtcars,
  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 violin plot with a box plot inside

```
library(tidyverse)
starwars |>
  filter(!is.na(height), !is.na(sex)) |>
  plotly::plot_ly(
    x = ~sex,
    y = ~height,
    type = "violin",
    split = ~sex,
    box = list(visible = T), # adds boxplot inside
    meanline = list(visible = F), # shows mean
    fillcolor = "rgba(100,100,255,0.5)",
    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"
  )
```

Mini exercise

Create an interactive histogram using `moderndive::evals`:

- `x = score`
- color by `gender`

```
plotly::plot_ly(
  data = evals,
  x = ~score,
  color = ~gender,
  type = "histogram",
  nbinsx = 30,
  opacity = 0.7
) |> plotly::layout(
  title = "Distribution of Course Evaluation Scores by Gender",
  xaxis = list(title = "Course Evaluation Score"),
  yaxis = list(title = "Count"),
  barmode = "stack",
  legend = list(title = list(text = "Gender"))
)
```

Create an interactive stacked bar plot using `moderndive::evals`:

- Create counts using `evals |> mutate(score_group = cut(score, breaks = 5)) |> count(score_group, gender)`
- `x = n` (counts)
- `y = score_group`
- color by `gender`
- horizontal stacked bars

```
evals_bar = evals |>
  mutate(score_group = cut(score, breaks = 5)) |>
  count(score_group, gender)

plotly::plot_ly(
  data = evals_bar,
  x = ~n,
  y = ~score_group,
  color = ~gender,
  type = "bar",
  orientation = "h"
) |> plotly::layout(
  title = "Counts of Score Ranges by Gender",
  xaxis = list(title = "Count"),
  yaxis = list(title = "Score Range"),
  barmode = "stack",
  legend = list(title = list(text = "Gender"))
)
```

Create an interactive boxplot using `moderndive::evals`:

- `x = gender`
- `y = score`
- color by `gender`

```
plotly::plot_ly(
  data = evals,
  x = ~gender,
  y = ~score,
  color = ~gender,
  type = "box"
) %>% plotly::layout(
  title = "Boxplot of Course Evaluation Scores by Gender",
  xaxis = list(title = "Gender"),
  yaxis = list(title = "Course Evaluation Score"),
  legend = list(title = list(text = "Gender"))
)
```

Create an interactive violin plot using `moderndive::evals`:

- `x = gender`
- `y = score`
- boxplot inside

```
plotly::plot_ly(
  data = evals,
  x = ~gender,
  y = ~score,
  type = "violin",
  split = ~gender,
  box = list(visible = TRUE),
  meanline = list(visible = TRUE),
  opacity = 0.6
) %>% plotly::layout(
  title = "Violin Plot of Course Evaluation Scores by Gender",
  xaxis = list(title = "Gender"),
  yaxis = list(title = "Course Evaluation Score")
)
```

Interactive line plot

```
plotly::plot_ly(
  data = ggplot2::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)")
)
```

Mini exercise

The following data are derived from `nycflights13::flights` which contains information on commercial flights departing from New York City in 2013.

```
library(tidyverse)

flights_daily <- nycflights13::flights %>%
  mutate(date = make_date(year, month, day)) %>%
  group_by(date) %>%
  summarize(
    avg_dep_delay = mean(dep_delay, na.rm = TRUE) # average departure delay for each day
  ) %>%
  ungroup()
```

Single line plot: create a line plot showing

- `x = date`
- `y = avg_dep_delay`

```
plotly::plot_ly(
  data = flights_daily,
```

```
  x = ~date,
  y = ~avg_dep_delay,
  type = "scatter",
  mode = "lines",
  line = list(color = "steelblue"),
  text = ~paste(
    "Date:", date,
    "<br>Avg delay:", round(avg_dep_delay, 2)
  ),
  hoverinfo = "text"
) %>% plotly::layout(
  title = "Average Daily Departure Delay",
  xaxis = list(title = "Date"),
  yaxis = list(title = "Average Departure Delay (minutes)")
)
```

Compare trajectories of multiple airports

- Recreate a dataset with average departure delay for each `date` and each `origin` (airport)
- `x = date`
- `y = avg_dep_delay`
- color by `origin`

```
library(tidyverse)
flights_daily_origin <- nycflights13::flights %>%
  mutate(date = make_date(year, month, day)) %>%
  group_by(date, origin) %>%
  summarize(
    avg_dep_delay = mean(dep_delay, na.rm = TRUE),
  ) %>%
  ungroup()

plotly::plot_ly(
  data = flights_daily_origin,
  x = ~date,
  y = ~avg_dep_delay,
  color = ~origin,
  type = "scatter",
  mode = "lines"
) %>% plotly::layout(
  title = "Average Daily Departure Delay by Airport",
  xaxis = list(title = "Date"),
  yaxis = list(title = "Average Departure Delay"),
  legend = list(title = list(text = "Airport"))
)
```

Plot of monthly aggregation for each `origin`:

- `x = month`
- `y = average monthly departure delay`

```
flights_monthly_origin <- nycflights13::flights %>%
  group_by(month, origin) %>%
```

```r
  summarize(
    avg_dep_delay = mean(dep_delay, na.rm = TRUE),
  ) %>%
  ungroup()

plotly::plot_ly(
  data = flights_monthly_origin,
  x = ~month,
  y = ~avg_dep_delay,
  color = ~origin,
  type = "scatter",
  mode = "lines+markers"
) %>% plotly::layout(
  title = "Average Monthly Departure Delay",
  xaxis = list(title = "Month"),
  yaxis = list(title = "Average Departure Delay")
)
```

Interactive geographic map: hightlighting coordinates

```r
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 = "natural earth"), # "natural earth" or "orthographic"
    bgcolor = "lightyellow",      # overall background
    showland = TRUE,
    landcolor = "lightgreen",     # land color
    showocean = TRUE,
    oceancolor = "lightblue",     # ocean color
    showlakes = TRUE,
    lakecolor = "blue"            # lake color
  ),
  title = "Cities on a Globe"
)
```

Interactive geographic map: hightlighting countries

```
plotly::plot_ly(
  type = "choropleth",
  locations = c("USA", "CAN", "AUS"), # ISO 3166-1 alpha-3 codes
  locationmode = "ISO-3",
  z = c(1,5,3),  # one color for one value of z
  colorscale = list(c(0, "lightgray"), c(1, "steelblue")),  # Color scale
  showscale = F
) |> plotly::layout(
  geo = list(
    showframe = T,
    showcoastlines = TRUE,
    projection = list(type = "natural earth")  #  "natural earth" or "orthographic"
  ),
  title = ""
)
```

Interactive geographic map: hightlighting Wisconsin on a US Map

```
plotly::plot_ly(
  type = "choropleth",
  locations = c("WI", "AK", "MN"),
  locationmode = "USA-states",
  z = c(10000,5000, 3000),  # one color for one value of z
  colorscale = list(c(0, "lightgray"), c(1, "forestgreen")),  # Color scale
  showscale = F
) |> plotly::layout(
  geo = list(
    scope = "usa"
  ),
  title = "Wisconsin and Alaska"
)
```

Mini exercise

Create a data frame containing the coordinates of airports you have visited, e.g.,

- New York City (JFK): 40.6413, -73.7781
- Chicago (ORD): 41.9742, -87.9073
- Los Angeles (LAX): 33.9416, -118.4085

Then create an interactive geographic map highlighting these airports.

```
airport_df = data.frame(
  airport = c("New York (JFK)", "Chicago (ORD)", "Los Angeles (LAX)"),
  lat = c(40.6413, 41.9742, 33.9416),
  lon = c(-73.7781, -87.9073, -118.4085)
)

plotly::plot_ly(
  data = airport_df,
```

```
  type = "scattergeo",
  mode = "markers",
  lat = ~lat,
  lon = ~lon,
  text = ~airport,
  hoverinfo = "text",
  marker = list(size = 10, color = "blue")
) |> plotly::layout(
  title = "Selected U.S. Airports",
  geo = list(
    projection = list(type = "natural earth"),
    showland = TRUE
  )
)
```

Create a map highlighting countries you have visited

```
plotly::plot_ly(
  type = "choropleth",
  locations = c("USA", "CAN", "MEX"),
  locationmode = "ISO-3",
  z = c(3, 2, 1),
  colorscale = list(
    c(0, "lightgray"),
    c(1, "steelblue")
  ),
  showscale = F
) |> plotly::layout(
  title = "Highlighted Countries in North America",
  geo = list(
    showframe = TRUE,
    showcoastlines = TRUE,
    projection = list(type = "natural earth")
  )
)
```

Create a map highlighting U.S. states with the Top 5 busiest airports

```
plotly::plot_ly(
  type = "choropleth",
  locations = c("GA", "TX", "CO", "IL", "CA"),
  locationmode = "USA-states",
  z = c(1:5),
  colorscale = list(
    c(0, "lightgray"),
    c(1, "forestgreen")
  ),
  showscale = F
) |> plotly::layout(
  title = "Selected U.S. States",
  geo = list(
    scope = "usa"
```

```
  )
)
```

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 = 1, 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()
  )
)
# 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)
```

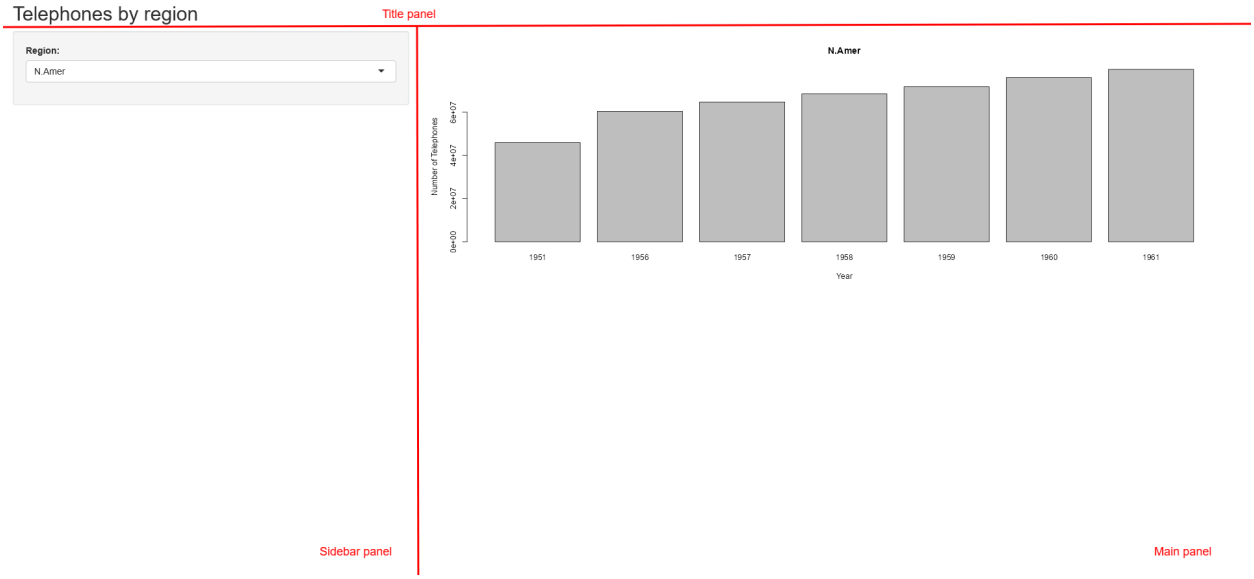

Figure 1: Three panels

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`

Update following functions

- `plotOutput` -> `plotlyOutput`
- `renderPlot` -> `renderPlotly`

```
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") # plotOutput -> 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 = 1, round = T,
        value= 10 # Initial value
      ),
      textInput(
        inputId = "text",
        label = "4. Enter some text to be displayed",
        value = "Feel free to input any text here." # Initial value
      )
    ),
    mainPanel(
      plotOutput("myhist"),
      tableOutput("mytable"),
      textOutput("mytext")
    )
  )
)

server = function(input, output) {
```

```
  output$myhist = renderPlot({
    Barley %>%
      filter(gen == input$gen) %>%
      ggplot(aes(x = yield)) +
      geom_histogram(
        bins = input$bin,
        group = input$gen,
        color = input$color,
        fill = input$fill
      )
  })
  output$mytable = renderTable({
    Barley %>%
      filter(gen == input$gen) %>%
      summarise(
        "Mean" = mean(yield),
        "Median" = median(yield),
        "STDEV" = sd(yield),
        "Min" = min(yield),
        "Max" = max(yield)
      )
  })
  output$mytext = renderText(input$text)
}

# 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)
```

Reactive programming in Shiny

- Reactive programming means: when an input changes, the app automatically updates related outputs.
- Example: user adjusts a slider -> Shiny recalculates something -> Plot/table/text updates automatically
- Main reactive components in Shiny

Function	Purpose
`input$...`	Gets values from UI controls
`output$...`	Stores outputs shown in the app
`render*()`	Creates reactive outputs
`reactive()`	Stores reusable reactive calculations (returns a value)
`observeEvent()`	Runs code only when a specific event happens
`isolate()`	Reads inputs without triggering updates

Use `reactive()` when

- you repeat the same computation multiple times
- multiple outputs depend on the same processed data

```
library(shiny)
library(tidyverse)

url <- "https://github.com/rstudio-education/shiny-course/raw/main/movies.RData"
destfile <- "movies.RData"
download.file(url, destfile)
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) {
  ############## Generate a reusable movies_filtered() #################
  movies_filtered = reactive({
    movies |> filter(
      thtr_rel_date >= as.POSIXct(input$date[1]) &
      thtr_rel_date <= as.POSIXct(input$date[2])
    )
  })
  ######################################################################
```

```
  output$scatterplot <- renderPlot({
    movies_filtered() |>  ggplot(
      aes(x = .data[[input$x]], y = .data[[input$y]], color = .data[[input$z]])
    ) +
    geom_point(alpha = input$opacity)
  })

  output$densityplot <- renderPlot({
    movies_filtered() |>
      ggplot(aes(x = .data[[input$x]])) +
      geom_density()
  })

  output$moviestable <- DT::renderDT({
    if(input$show_data == T){
      movies_filtered() |> 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_filtered(),
        file,
        row.names = FALSE
      )
    }
  )
}

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

Use `observeEvent()` when you only want code to run after ONE specific event

Use `isolate()` when:

- you want to read an input
- but do NOT want it to trigger reactivity

```
library(shiny)
library(tidyverse)

url <- "https://github.com/rstudio-education/shiny-course/raw/main/movies.RData"
destfile <- "movies.RData"
download.file(url, destfile)
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
      ),
      ############ a newly added button ###################
      actionButton(
        inputId = "go",
        label = "Generate"
      ),
      ######################################################
      br(), # initiate a newline for a good UI design
      downloadButton(
        outputId = "download_data",
        label = "Download Data"
      ),
    ),
    mainPanel(
      plotOutput(outputId = "scatterplot"),
      plotOutput(outputId = "densityplot", height = 200),
      DT::dataTableOutput(outputId = "moviestable")
    )
  )
)

server <- function(input, output) {
  ################### No plot until the "Generate" button is hit ###########
  observeEvent(
    eventExpr = input$go,
    handlerExpr = {
      movies_filtered = reactive({
        movies |> filter(
          thtr_rel_date >= as.POSIXct(isolate(input$date[1])) &  # isolate() reads the value without tr
          thtr_rel_date <= as.POSIXct(isolate(input$date[2]))
        )
      })
      output$scatterplot <- renderPlot({
        movies_filtered() |>  ggplot(
          aes(
            x = .data[[isolate(input$x)]],
            y = .data[[isolate(input$y)]],
            color = .data[[isolate(input$z)]]
          )
        ) +
        geom_point(alpha = isolate(input$opacity))
      })
      output$densityplot <- renderPlot({
        movies_filtered() |>
          ggplot(aes(x = .data[[isolate(input$x)]])) +
          geom_density()
      })
      output$moviestable <- DT::renderDT({
        if(input$show_data == T){
          movies_filtered() |> 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_filtered(),
        file,
        row.names = FALSE
      )
    }
  )
}

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

More R Shiny input widgets

https://raw.githubusercontent.com/rstudio/cheatsheets/main/shiny.pdf

bslib for a customized look

- bslib is an R package that enables you to customize the look and feel of Shiny apps and R Markdown documents using Bootstrap themes (pre-designed website templates built using the Bootstrap framework, a popular HTML, CSS, and JavaScript framework for responsive web development)

```
library(shiny)
library(tidyverse)
library(bslib)

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

ui <- fluidPage(
  ########################## Customized theme ##############################
  theme = bs_theme(
    version = 5, # Bootstrap theme version (4 or 5)
    bootswatch = "cosmo", # Prebuilt Bootstrap themes; available themes at
    # print(bootswatch_themes(version = "5")); previews at
    # bs_theme_preview(bs_theme(bootswatch = "superhero"))
    primary = "#198754", # Color; available colors at
    # https://rstudio.github.io/bslib/articles/bs4-variables/index.html
    # https://rstudio.github.io/bslib/articles/bs5-variables/index.html
    base_font = font_google("Special Gothic Expanded One"),
    heading_font = font_google("Merriweather"),
    code_font = font_google("Fira Code")
    # available fonts at https://fonts.google.com
  ),
```

```
################################################################################
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
      ),
      actionButton(
        inputId = "go",
        label = "Generate"
      ),
      br(), # initiate a newline
      downloadButton(
        outputId = "download_data",
        label = "Download Data"
      ),
    ),
    mainPanel(
      plotOutput(outputId = "scatterplot"),
      plotOutput(outputId = "densityplot", height = 200),
      DT::dataTableOutput(outputId = "moviestable")
    )
  )
)

server <- function(input, output) {
  observeEvent(
    eventExpr = input$go,
    handlerExpr = {
      movies_filtered = reactive({
        movies |> filter(
          thtr_rel_date >= as.POSIXct(isolate(input$date[1])) &
          thtr_rel_date <= as.POSIXct(isolate(input$date[2]))
        )
      })
      output$scatterplot <- renderPlot({
        movies_filtered() |>  ggplot(
          aes(
            x = .data[[isolate(input$x)]],
            y = .data[[isolate(input$y)]],
            color = .data[[isolate(input$z)]]
          )
        ) +
        geom_point(alpha = isolate(input$opacity))
      })
      output$densityplot <- renderPlot({
        movies_filtered() |>
          ggplot(aes(x = .data[[isolate(input$x)]])) +
          geom_density()
      })
      output$moviestable <- DT::renderDT({
        if(input$show_data == T){
          movies_filtered() |> 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_filtered(),
        file,
        row.names = FALSE
      )
    }
  )
}

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

Sharing R Shiny apps as webpages

- User-friendly: local installation of R NOT required for users
- Still a bit sticking: required computing power to fit the number of expected users, ...
- Platforms
  - shinyapps.io
    - https://www.shinyapps.io/
    - Posit's hosting service with different plans
      - Free plan limited to 5 apps, 25 active hours per month
  - Shiny server
    - https://posit.co/download/shiny-server/
    - Building a server by yourself to host Shiny apps (free, open source)

Deploying an R Shiny app on shinyapps.io

- Sign up at https://www.shinyapps.io with the free plan.
- Create an R Script including the source code of your app.
  - The code should starts by loading the necessary packages, preprocessing the data, and defining the UI and server settings. Finish it with a call to the shinyApp() function, without anything unnecessary.
- Leave the R script in a separate folder (with no redundant files in the same folder).
- Open the R script in RStudio.
- Test your app locally by clicking the "**Run App**" button (upper-right).
- Click the "**Publish**" button (upper-right), choose shinyapps.io as the server and follow prompts.
  - Go to your shinyapps.io account.
  - Under the "Account" tab, copy the name, token and secret for authentication.
- R will upload your app and provide a unique public URL
- Updated when you republish