Shiny

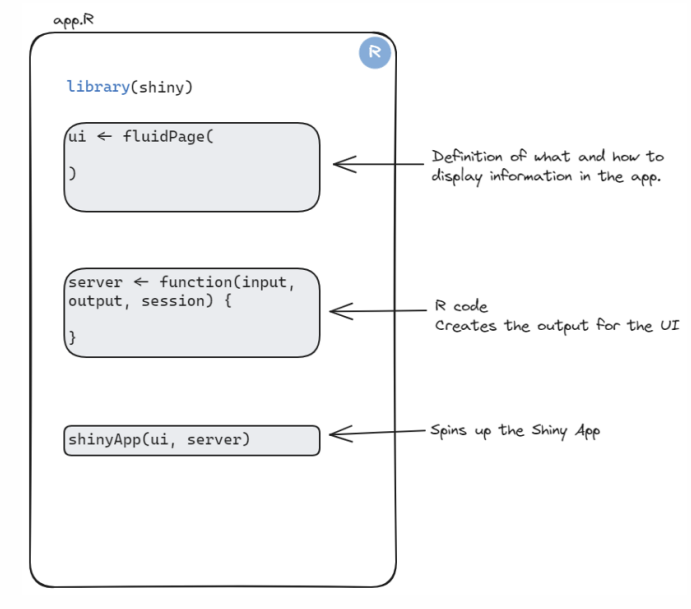
Ggplot

The ggplot2 package is built on the *Grammar of Graphics*, which describes how data visualizations are constructed using seven key layers:

1. **Data** — The dataset you want to visualize.
2. **Aesthetics (aes)** — Defines how data is mapped to visual properties such as the x-axis, y-axis, color, size, or shape.
3. **Geometry (geom)** — Specifies the geometric object used to represent data (e.g., points, lines, bars).
4. **Facets** — Splits the plot into multiple panels to compare subsets of the data.
5. **Statistics** — Adds statistical transformations such as smoothing lines, counts, or summaries.
6. **Coordinates** — Defines the coordinate system (e.g., Cartesian, polar).
7. **Themes** — Controls non-data elements like fonts, colors, backgrounds, and grid lines.

[Create Elegant Data Visualisations Using the Grammar of Graphics • ggplot2]

Rshiny



ui <- fluidPage()

➔ Note that fluidPage is a function Uses (), so arguments need to be comma separated

server <- function(input, output) {}

➔ Note that server defines a new function Uses {} (curly brackets), so code is separated by line

shinyApp(ui = ui, server = server)

➔ Spins up the shiny App

INPUT & OUTPUT in Shiny

In a Shiny app, the **UI** (what the user sees) and the **server** (the R code that does the work) are constantly *talking to each other*.

They do this using **input** and **output**.

Think of it like a two-way conversation:

- **UI → sends input → Server**
- **Server → sends output → UI**

What the UI does

The UI is the *front end* — sliders, buttons, text boxes, plots, etc.

The UI:

1. Listens to output

It waits to receive results (plots, tables, text) from the server so it can show them on the screen.

2. Sends user actions as input

Whenever a user moves a slider, clicks a button, or selects something:

- that action is stored in input$...
- then sent to the server so the server knows what changed

Example:

input$my_slider is the current value of the slider called *my_slider*.

What the Server does

The server is the *back end* — it performs calculations, generates plots, etc.

The Server:

1. Listens to input

Whenever input changes (like a slider value), the server reacts.

2. Creates output

The server takes inputs, does the work, and sends results (like plots/tables) back to the UI.

Example:

output$my_plot is where the server stores the plot that will appear in the UI.

Example with Biopics dataset:

```r
biopics <- read.csv("C:/Users/dinok/OneDrive/Desktop/Uni/1 Semester/Database Management/Shiny/biopics.csv")
View(biopics)
```

Clean the data

```r
#Clean the data
library(tidyverse)
biopics <- biopics %>%
  mutate(box_office = gsub("\\$", "", box_office)) %>%
  mutate(box_office = gsub("m", "", box_office)) %>%
  mutate(box_office = gsub("-", "", box_office)) %>%
  mutate(box_office = as.numeric(box_office))
view(biopics)
```

Whats happening:

mutate() → Adds a new column or modifies an existing one. Here, we are modifying box_office.

gsub("\\$", "", box_office)

- Removes the $ sign from the box_office column.
- Example: $120m → 120m.

gsub("m", "", box_office)

- Removes the letter m (millions).
- Example: 120m → 120.

gsub("-", "", box_office)

- Removes any dashes - (probably for missing or unknown values).
- Example: - → `` (empty string).

as.numeric(box_office)

- Converts the cleaned strings into numeric values so you can plot or do calculations.
- Example: "120" → 120.

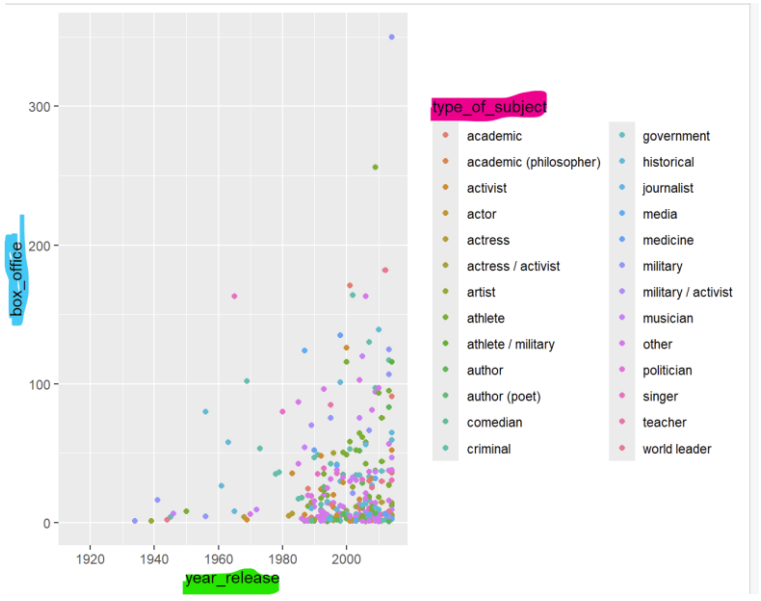
Create a GGPlot

```r
library(ggplot2)
my_plot <-  ggplot(biopics) +
  aes(
    x=year_release,
    y=box_office,
    color= type_of_subject) +
  geom_point()

my_plot
```

You are starting a ggplot and telling it what data to use.

This is the aesthetic mapping layer — telling ggplot which columns go to which visual component.
x = year_release → horizontal axis
y = box_office → vertical axis
color = type_of_subject → color each point based on category
This doesn't draw anything yet — it only sets up the rules.

This is the geometry layer — the shape used to represent data points.
geom_point() means: "Draw a dot for each row in the dataset."



Adding a Plot to App

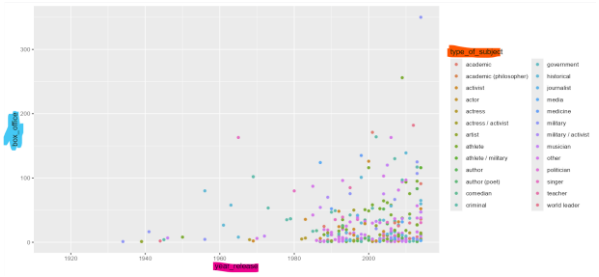
```
# Adding a Plot to App
ui <- fluidPage(
  plotOutput("movie_plot")
)

server <- function(input, output) {

  output$movie_plot <- renderPlot({

    ggplot(biopics) +
      aes(x=year_release,
          y=box_office,
          color=type_of_subject) +
      geom_point()

  })

}

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

fluidPage() creates a responsive web page that automatically adjusts to your browser size
plotOutput("movie_plot") reserves a space in the UI for a plot.
"movie_plot" is the output ID, which the server will use to send the plot to the UI.

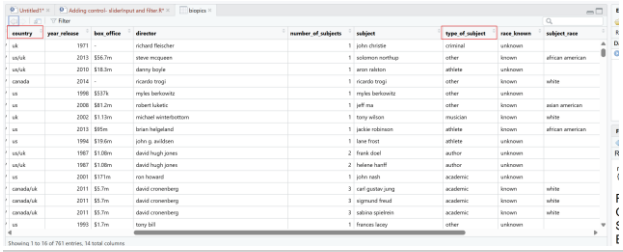
Generating the Plot
ggplot(biopics) → Use your dataset biopics.
geom_point() → Draw each movie as a point on the scatter plot.

This launches the app in a browser or RStudio Viewer.

Running this automatically opens



Adding a Control



```
categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- fluidPage(
  plotOutput("movie_plot"),
  selectInput(
    inputId = "color_select",
    label = "Select Categorical Variable",
    choices = categoricalVars)
)

server <- function(input, output) {

  output$movie_plot <- renderPlot({

    ggplot(biopics) +
      aes(x=year_release,
          y=box_office,
          color=.data[[input$color_select]]) +
      geom_point()
  })

}

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

Creates a vector of column names from the biopics dataset

fluidPage() → Creates a responsive web page.

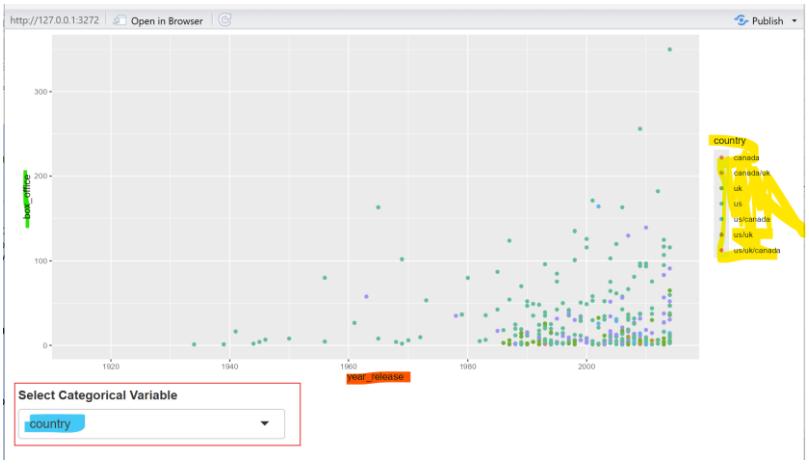
Adds a dropdown menu for the user to choose a categorical variable:
inputId = "color_select" → This is how the server knows what the user picked.
label = "Select Categorical Variable",
choices = categoricalVars → Dropdown options come from the vector we defined earlier.

server <- function(input, output) → Backend that reacts to user inputs.

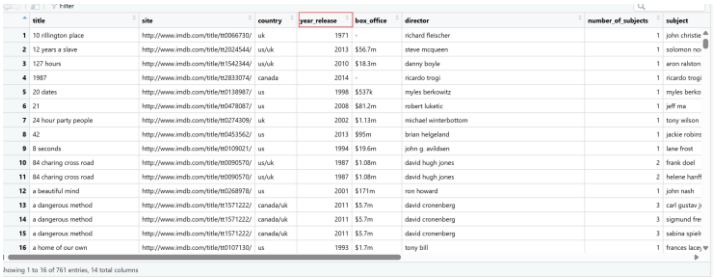
output$movie_plot <- renderPlot({ ... }) → Generates the plot for the UI.

color = .data[[input$color_select]] → dynamically use whatever column the user selected to color points.

Launches the app in a browser or RStudio Viewer.



Making a Dataset Filterable



	title	site	country	year_release	box_office	director	number_of_subjects	subject
1	10 rillington place	http://www.imdb.com/title/tt0066730/	uk	1971	-	richard fleischer	1	john christie
2	12 years a slave	http://www.imdb.com/title/tt2024544/	us/uk	2013	$56.7m	steve mcqueen	1	solomon no
3	127 hours	http://www.imdb.com/title/tt1542344/	us/uk	2010	$18.3m	danny boyle	1	aron ralston
4	1987	http://www.imdb.com/title/tt2833074/	canada	2014	-	ricardo trogi	1	ricardo trogi
5	20 dates	http://www.imdb.com/title/tt0138987/	us	1998	$537k	myles berkowitz	1	myles berko
6	21	http://www.imdb.com/title/tt0478087/	us	2008	$81.2m	robert luketic	1	jeff ma
7	24 hour party people	http://www.imdb.com/title/tt0274309/	uk	2002	$1.13m	michael winterbottom	1	tony wilson
8	42	http://www.imdb.com/title/tt0453562/	us	2013	$95m	brian helgeland	1	jackie robins
9	8 seconds	http://www.imdb.com/title/tt0109021/	us	1994	$19.6m	john g. avildsen	1	lane frost
10	84 charing cross road	http://www.imdb.com/title/tt0090570/	us/uk	1987	$1.08m	david hugh jones	2	frank doel
11	84 charing cross road	http://www.imdb.com/title/tt0090570/	us/uk	1987	$1.08m	david hugh jones	2	helene hanff
12	a beautiful mind	http://www.imdb.com/title/tt0268978/	us	2001	$171m	ron howard	1	john nash
13	a dangerous method	http://www.imdb.com/title/tt1571222/	canada/uk	2011	$5.7m	david cronenberg	3	carl gustav j
14	a dangerous method	http://www.imdb.com/title/tt1571222/	canada/uk	2011	$5.7m	david cronenberg	3	sigmund fre
15	a dangerous method	http://www.imdb.com/title/tt1571222/	canada/uk	2011	$5.7m	david cronenberg	3	sabina spiel
16	a home of our own	http://www.imdb.com/title/tt0107130/	us	1993	$1.7m	tony bill	1	frances lacey

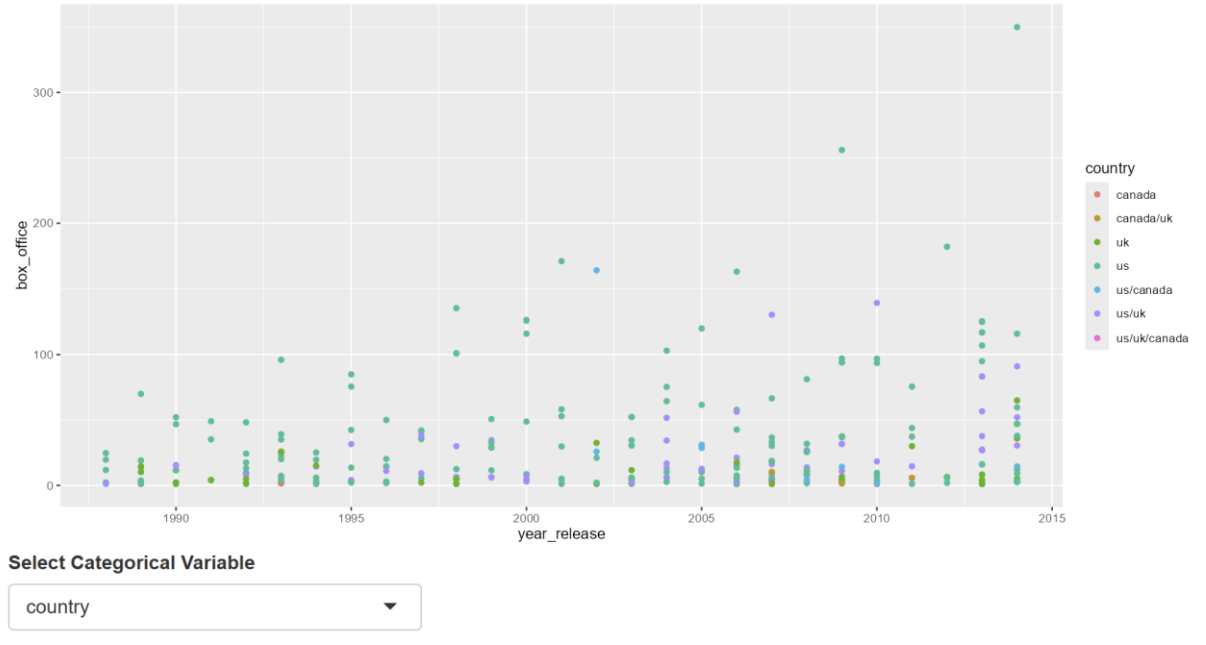
First create a variable with a filter

```
biopics_filtered <- biopics %>%
  filter(year_release > 1987)
```

Run the code with the filter variable

```
#2. Run the code from above where the user can select
categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- fluidPage(
  plotOutput("movie_plot"),
  selectInput(
    inputId = "color_select",
    label = "Select Categorical Variable",
    choices = categoricalVars)
)

server <- function(input, output) {

  output$movie_plot <- renderPlot({

    ggplot(biopics_filtered) +
      aes(x=year_release,
          y=box_office,
          color=.data[[input$color_select]]) +

      geom_point()
  })

}

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

Now we just see data where year_release > 1987



Adding control: sliderInput()

```r
# Adding control: sliderInput() ####

ui <- fluidPage(

  plotOutput("movie_plot"),
  sliderInput("year_filter",
              "Select Lowest Year",
              min = 1915,
              max=2014,
              value = 1915)
)

server <- function(input, output) {

  biopics_filtered <- reactive({

    biopics %>%
      filter(year_release > input$year_filter)

  })

  output$movie_plot <- renderPlot({

    ggplot(biopics_filtered()) +
      aes(x=year_release,
          y=box_office) +

      geom_point()
  })

}

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

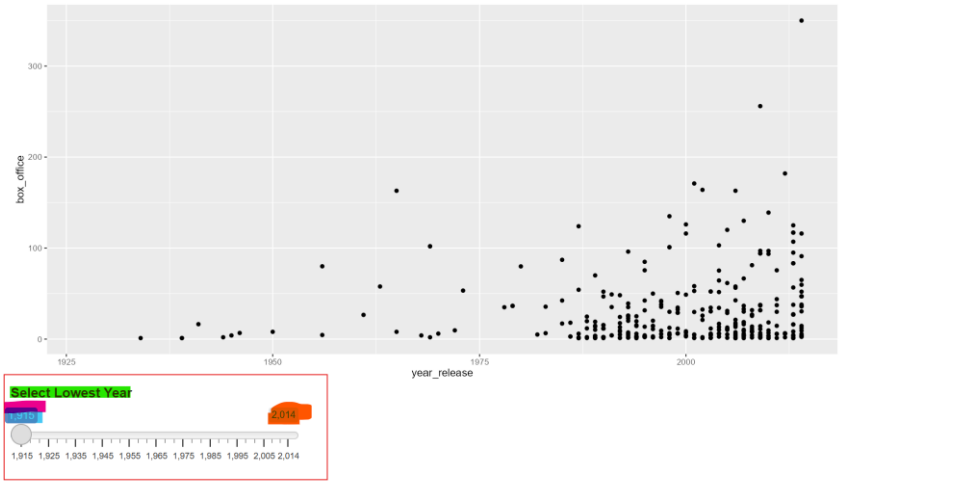
sliderInput("year_filter", ...) This adds a slider control to the page.
"year_filter" → input ID
"Select Lowest Year" → label shown to the user
min = 1915 → slider starts at 1915
max = 2014 → slider ends at 2014
value = 1915 → slider starts at 1915 by default

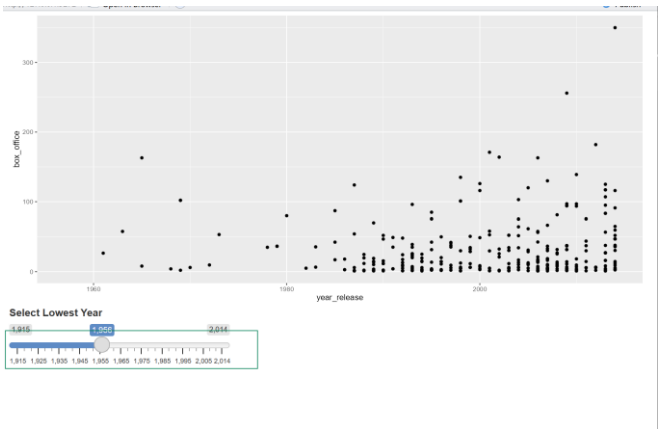
The server contains the code that reacts to user actions and generates outputs.

reactive({ ... }) creates a reactive expression. It automatically updates whenever a related input changes

filter(year_release > input$year_filter)
This removes movies that were released before the chosen year.
If the user moves the slider to 1950, then:
Chosen filter:
input$year_filter = 1950
Filter rule:
year_release > 1950
Result:
Only movies released after 1950 will appear in the plot.

Creating the plot





Putting it all together

```
categoricalVars <- c("country", "type_of_subject", "subject_sex")
```

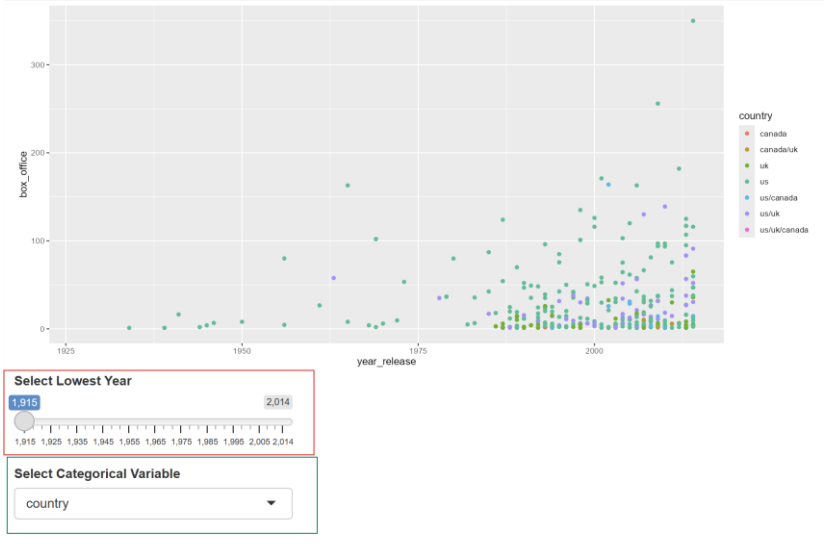
```
# Putting it together ####

ui <- fluidPage(

  plotOutput("movie_plot"),
  sliderInput("year_filter",
              "Select Lowest Year",
              min = 1915,
              max=2014,
              value = 1915),
  selectInput(
    inputId = "color_select",
    label = "Select Categorical Variable",
    choices = categoricalVars,
    selected = 1)
)

server <- function(input, output) {

  biopics_filtered <- reactive({

    biopics %>%
      filter(year_release > input$year_filter)

  })

  output$movie_plot <- renderPlot({

    ggplot(biopics_filtered()) +
      aes(x=year_release,
          y=box_office,
          color=.data[[input$color_select]]) +

      geom_point()
  })

}

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

- ✔ filters the dataset
- ✔ based on the selected year
- ✔ automatically updates when the slider changes

Rendering the Plot

What this does:
① Uses the filtered dataset biopics_filtered()
② Plots a scatterplot
x = year_release
y = box_office
③ Colors points dynamically

Running the App



Reactivity

Reactive programming is a style of programming that focuses on values that change over time, and calculations and actions that depend on those values.

What this means:

- In normal R programming, we run code once and get a result.
- In Shiny, values can change (like input sliders, inputs from users).
- When inputs change, Shiny must automatically update the outputs.

Reactivity = automatic updating when inputs change

→ You don't re-run the code manually.

→ Shiny figures out what depends on what.

Why normal R codes are not reactive

```
216 # Reactivity
217   temp_c <- 10
218   temp_f <- (temp_c * 9 / 5) + 32
219   temp_f   # 50
220
221   temp_c <- 30
222   temp_f   # still 50
```

```
>   temp_c <- 10
>   temp_f <- (temp_c * 9 / 5) + 32
>   temp_f   # 50
[1] 50
>
>   temp_c <- 30
>   temp_f   # still 50
[1] 50
```

- Because temp_f was computed once, when you first ran the code.
- Changing temp_c later does not trigger an update.

Functions as partial solution

```
224 # funtions as sultion
225 temp_c <- 10
226 temp_f <- function() {
227   message("Converting")
228   (temp_c * 9 / 5) + 32
229 }
230 temp_f()
231 # Converting
232 # 50
```

- A function recalculates every time it's called → so it always uses the current value of temp_c.
- So functions solve only one of the two Shiny problems:
    - ✓ Problem 1 solved:
        - You get updated values every time.
    - ✗ Problem 2 NOT solved:
        - The function recomputes every time, even when nothing changed.
        - This is inefficient when your "function" is actually something heavier

We need reactive expression

```
235 #Reactive expression
236 temp_f <- reactive({
237   (input$temp_c * 9/5) + 32
238 })
```

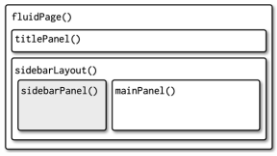
- ✓ Lazy
    - It only runs when needed.
    - It waits until something calls it (e.g., output).
- ✓ Cached
    - It only re-runs when its input changes.
    - If you call it 10 times but the input hasn't changed → it uses the saved result.

Single Page Layout fluidPage sidebarLayout (with titlePanel(), sidebarPanel(), and mainPanel())

Using layout functions *inside* fluidPage()

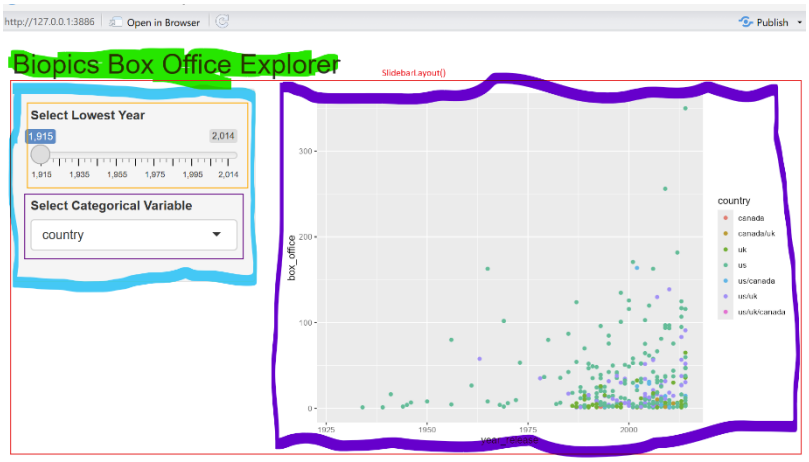
- This creates a typical Shiny UI:
    - Left column: inputs
    - Right column: output(s)

```
fluidPage(
  titlePanel(
    # app title/description
  ),
  sidebarLayout(
    sidebarPanel(
      # inputs
    ),
    mainPanel(
      # outputs
    )
  )
)
```




```
library(shiny)
library(ggplot2)
library(dplyr)

categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- fluidPage(

  # App title
  titlePanel("Biopics Box Office Explorer"),

  # Sidebar layout = left side inputs, right side plot
  sidebarLayout(

    # Left side (inputs)
    sidebarPanel(
      sliderInput(
        "year_filter",
        "Select Lowest Year",
        min = 1915,
        max = 2014,
        value = 1915
      ),

      selectInput(
        inputId = "color_select",
        label = "Select Categorical Variable",
        choices = categoricalVars,
        selected = 1
      )
    ),

    # Right side (outputs)
    mainPanel(
      plotOutput("movie_plot")
    )
  )
)

server <- function(input, output) {

  # Reactive dataset that updates when slider changes
  biopics_filtered <- reactive({
    biopics %>%
      filter(year_release > input$year_filter)
  })

  # Output plot that updates when:
  # - year_filter changes
  # - color_select changes
  # - underlying reactive data changes
  output$movie_plot <- renderPlot({
    ggplot(biopics_filtered()) +
      aes(
        x = year_release,
        y = box_office,
        color = .data[[input$color_select]]
      ) +
      geom_point()
  })
}

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

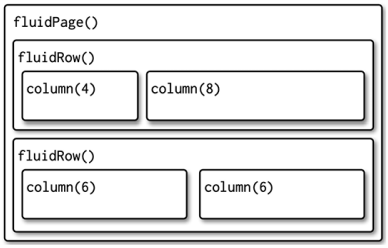
It creates a space in the UI (in the browser) where the plot will appear.
It does NOT draw the plot by itself.
It does NOT update the plot by itself.
It is only a placeholder — a location for the plot to appear.



Single Page Layout (fluid Row)

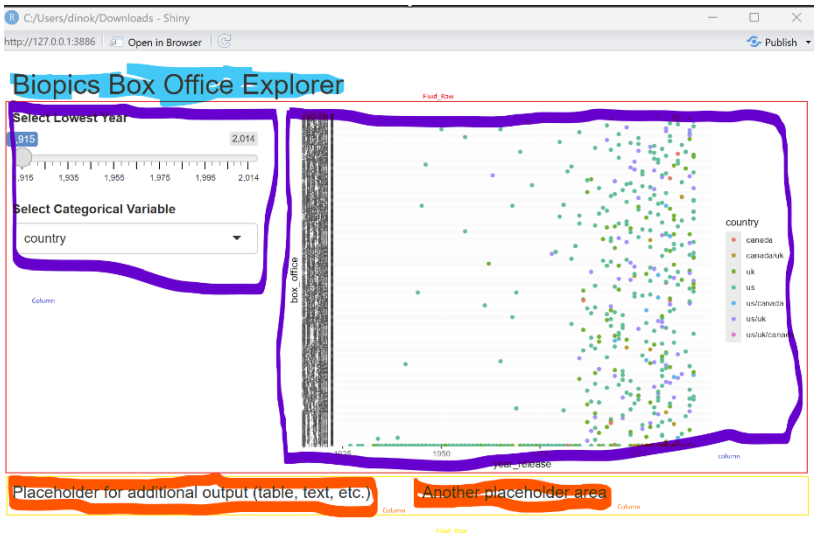
This is more flexible than sidebarLayout(), and it is how advanced Shiny layouts are typically created.

```
fluidPage(
  fluidRow(
    column(4,
      ...
    ),
    column(8,
      ...
    )
  ),
  fluidRow(
    column(6,
      ...
    ),
    column(6,
      ...
    )
  )
)
```



Your app rewritten using `fluidRow()` + `column()`

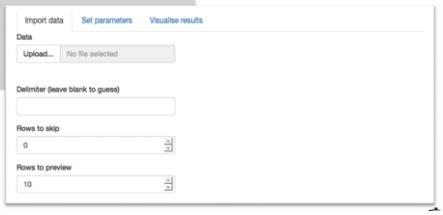
```r
library(shiny)
library(ggplot2)
library(dplyr)

categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- fluidPage(

  # App title
  titlePanel("Biopics Box Office Explorer"),

  # ---- First row: Inputs (left) and Plot (right)
  fluidRow(
    column(
      width = 4,  # Left side = 4/12 of the width

      # Inputs
      sliderInput(
        "year_filter",
        "Select Lowest Year",
        min = 1915,
        max = 2014,
        value = 1915
      ),

      selectInput(
        inputId = "color_select",
        label = "Select Categorical Variable",
        choices = categoricalVars,
        selected = 1
      )
    ),

    column(
      width = 8,  # Right side = 8/12 of the width
      plotOutput("movie_plot")
    )
  ),

  # ---- Second row: (example space)
  # You can add more outputs later
  fluidRow(
    column(
      width = 6,
      h4("Placeholder for additional output (table, text, etc.)")
    ),
    column(
      width = 6,
      h4("Another placeholder area")
    )
  )
)

server <- function(input, output) {

  biopics_filtered <- reactive({
    biopics %>%
      filter(year_release > input$year_filter)
  })

  output$movie_plot <- renderPlot({
    ggplot(biopics_filtered()) +
      aes(
        x = year_release,
        y = box_office,
        color = .data[[input$color_select]]
      ) +
      geom_point()
  })
}

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



Multi Page Layout tabsetPanel() & tabPanel

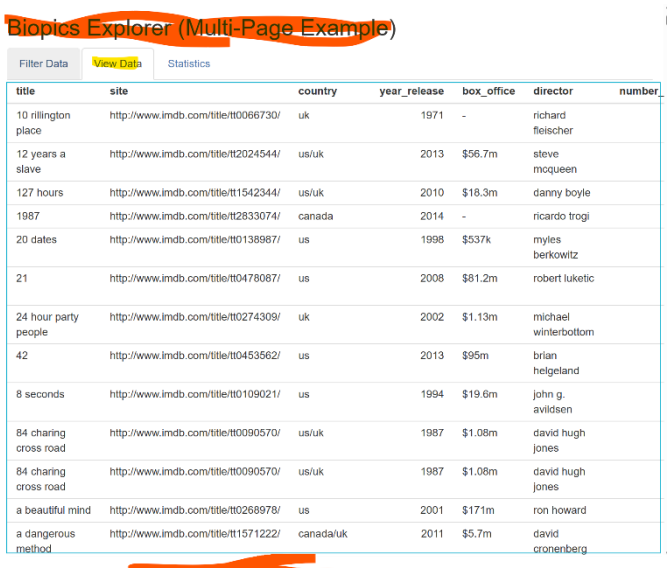
tabsetPanel() & tabPanel.

```
ui <- fluidPage(
  tabsetPanel(
    tabPanel("Import data",
      fileInput("file", "Data", buttonLabel = "Upload..."),
      textInput("delim", "Delimiter (leave blank to guess)", ""),
      numericInput("skip", "Rows to skip", 0, min = 0),
      numericInput("rows", "Rows to preview", 10, min = 1)
    ),
    tabPanel("Set parameters"),
    tabPanel("Visualise results")
  )
)
```

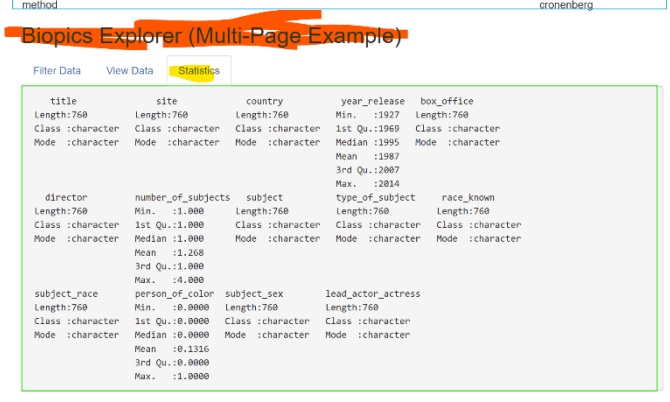


```
r
Code kopieren

library(shiny)
library(ggplot2)
library(dplyr)

categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- fluidPage(
  titlePanel("Biopics Explorer (Multi-Page Example)"),

  tabsetPanel(

    # --- PAGE 1: Filters ---
    tabPanel("Filter Data",

      fluidRow(
        column(6,
          sliderInput(
            "year_filter",
            "Select Lowest Year",
            min = 1915,
            max = 2014,
            value = 1915
          ),

          selectInput(
            inputId = "color_select",
            label = "Select Categorical Variable",
            choices = categoricalVars,
            selected = 1
          )
        ),

        column(6,
          h4("Filtered Plot Preview"),
          plotOutput("movie_plot")
        )
      )
    ),

    # --- PAGE 2: Data Table ---
    tabPanel("View Data",
      tableOutput("movie_table")
    ),

    # --- PAGE 3: Summary Statistics ---
    tabPanel("Statistics",
      verbatimTextOutput("summary_text")
    )
  )
)

server <- function(input, output) {

  biopics_filtered <- reactive({
    biopics %>%
      filter(year_release > input$year_filter)
  })

  output$movie_plot <- renderPlot({
    ggplot(biopics_filtered()) +
      aes(
        x = year_release,
        y = box_office,
        color = .data[[input$color_select]]
      ) +
      geom_point()
  })

  output$movie_table <- renderTable({
    biopics_filtered()
  })

  output$summary_text <- renderPrint({
    summary(biopics_filtered())
  })
}

shinyApp(ui, server)
```

This creates multiple pages



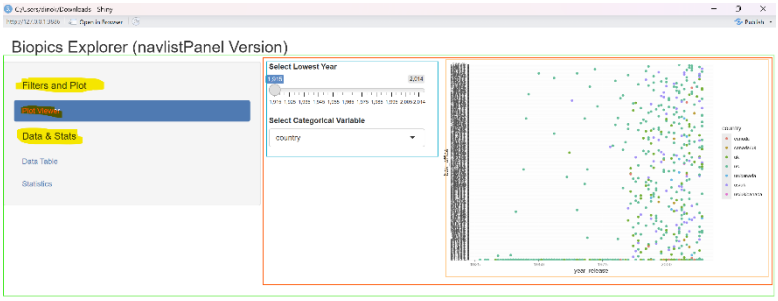
title	site	country	year_release	box_office	director	number_
10 rillington place	http://www.imdb.com/title/tt0066730/	uk	1971	-	richard fleischer	
12 years a slave	http://www.imdb.com/title/tt2024544/	us/uk	2013	$56.7m	steve mcqueen	
127 hours	http://www.imdb.com/title/tt1542344/	us/uk	2010	$18.3m	danny boyle	
1987	http://www.imdb.com/title/tt2833074/	canada	2014	-	ricardo trogi	
20 dates	http://www.imdb.com/title/tt0138987/	us	1998	$537k	myles berkowitz	
21	http://www.imdb.com/title/tt0478087/	us	2008	$81.2m	robert luketic	
24 hour party people	http://www.imdb.com/title/tt0274309/	uk	2002	$1.13m	michael winterbottom	
42	http://www.imdb.com/title/tt0453562/	us	2013	$95m	brian helgeland	
8 seconds	http://www.imdb.com/title/tt0109021/	us	1994	$19.6m	john g. avildsen	
84 charing cross road	http://www.imdb.com/title/tt0090570/	us/uk	1987	$1.08m	david hugh jones	
84 charing cross road	http://www.imdb.com/title/tt0090570/	us/uk	1987	$1.08m	david hugh jones	
a beautiful mind	http://www.imdb.com/title/tt0268978/	us	2001	$171m	ron howard	
a dangerous method	http://www.imdb.com/title/tt1571222/	canada/uk	2011	$5.7m	david cronenberg	

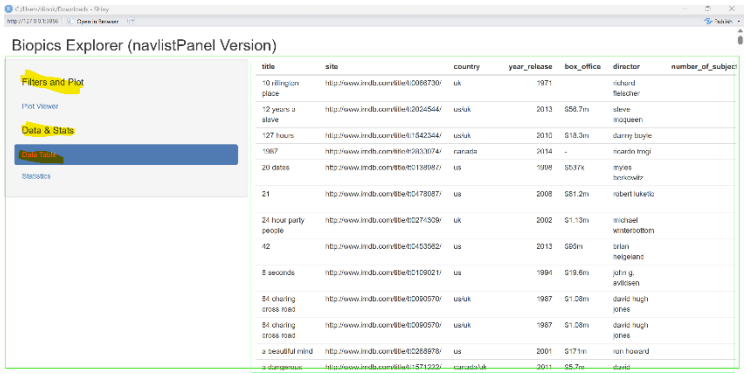


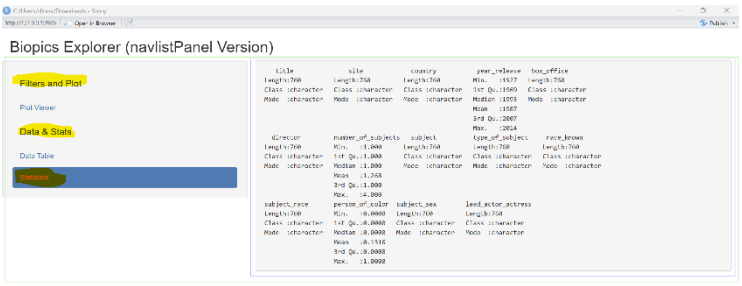

```
     title               site              country          year_release   box_office
 Length:760         Length:760         Length:760         Min.   :1927   Length:760
 Class :character   Class :character   Class :character   1st Qu.:1969   Class :character
 Mode  :character   Mode  :character   Mode  :character   Median :1995   Mode  :character
                                                          Mean   :1987
                                                          3rd Qu.:2007
                                                          Max.   :2014
   director         number_of_subjects   subject          type_of_subject     race_known
 Length:760         Min.   :1.000      Length:760         Length:760         Length:760
 Class :character   1st Qu.:1.000      Class :character   Class :character   Class :character
 Mode  :character   Median :1.000      Mode  :character   Mode  :character   Mode  :character
                    Mean   :1.268
                    3rd Qu.:1.000
                    Max.   :4.000
 subject_race       person_of_color  subject_sex        lead_actor_actress
 Length:760         Min.   :0.0000   Length:760         Length:760
 Class :character   1st Qu.:0.0000   Class :character   Class :character
 Mode  :character   Median :0.0000   Mode  :character   Mode  :character
                    Mean   :0.1316
                    3rd Qu.:0.0000
                    Max.   :1.0000
```

Multi Page Layout Navlists and Navbars

```r
library(shiny)
library(ggplot2)
library(dplyr)

categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- fluidPage(

  titlePanel("Biopics Explorer (navlistPanel Version)"),

  navlistPanel(
    id = "navlist",

    "Filters and Plot",
    tabPanel("Plot Viewer",
      fluidRow(
        column(4,
          sliderInput("year_filter", "Select Lowest Year",
              min = 1915, max = 2014, value = 1915),

          selectInput(
            "color_select",
            "Select Categorical Variable",
            choices = categoricalVars,
            selected = 1
          )
        ),
        column(8,
          plotOutput("movie_plot")
        )
      )
    ),

    "Data & Stats",
    tabPanel("Data Table",
      tableOutput("movie_table")
    ),
    tabPanel("Statistics",
      verbatimTextOutput("summary_text")
    )
  )
)

server <- function(input, output) {

  biopics_filtered <- reactive({
    biopics %>% filter(year_release > input$year_filter)
  })

  output$movie_plot <- renderPlot({
    ggplot(biopics_filtered()) +
      aes(x = year_release,
          y = box_office,
          color = .data[[input$color_select]]) +
      geom_point()
  })

  output$movie_table <- renderTable({
    biopics_filtered()
  })

  output$summary_text <- renderPrint({
    summary(biopics_filtered())
  })
}

shinyApp(ui, server)
```



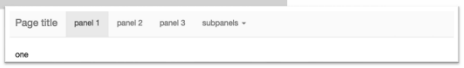




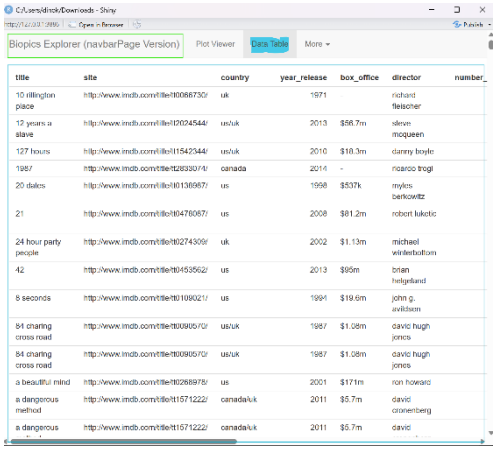
Multi Page Layout dropdownwith navbarMenu()

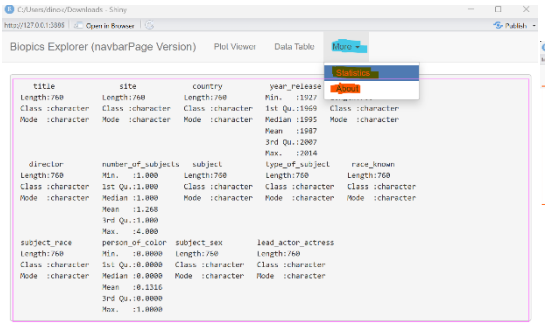
dropdown with navbarMenu()

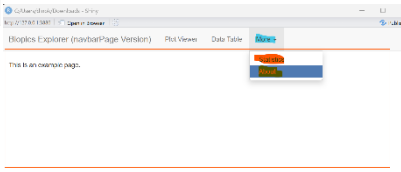
```
ui <- navbarPage(
  "Page title",
  tabPanel("panel 1", "one"),
  tabPanel("panel 2", "two"),
  tabPanel("panel 3", "three"),
  navbarMenu("subpanels",
    tabPanel("panel 4a", "four-a"),
    tabPanel("panel 4b", "four-b"),
    tabPanel("panel 4c", "four-c")
  )
)
```




```
library(shiny)
library(ggplot2)
library(dplyr)

categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- navbarPage(
  "Biopics Explorer (navbarPage Version)",

  # ---- MAIN PAGE 1 ----
  tabPanel("Plot Viewer",
    sidebarLayout(
      sidebarPanel(
        sliderInput("year_filter", "Select Lowest Year",
            min = 1915, max = 2014, value = 1915),

        selectInput(
          "color_select",
          "Select Categorical Variable",
          choices = categoricalVars,
          selected = 1
        )
      ),
      mainPanel(
        plotOutput("movie_plot")
      )
    )
  ),

  # ---- MAIN PAGE 2 ----
  tabPanel("Data Table",
    tableOutput("movie_table")
  ),

  # ---- DROPDOWN MENU ----
  navbarMenu("More",
    tabPanel("Statistics",
      verbatimTextOutput("summary_text")
    ),
    tabPanel("About",
      p("This is an example page.")
    )
  )
)

server <- function(input, output) {

  biopics_filtered <- reactive({
    biopics %>% filter(year_release > input$year_filter)
  })

  output$movie_plot <- renderPlot({
    ggplot(biopics_filtered()) +
      aes(x = year_release,
          y = box_office,
          color = .data[[input$color_select]]) +
      geom_point()
  })

  output$movie_table <- renderTable({
    biopics_filtered()
  })

  output$summary_text <- renderPrint({
    summary(biopics_filtered())
  })
}

shinyApp(ui, server)
```

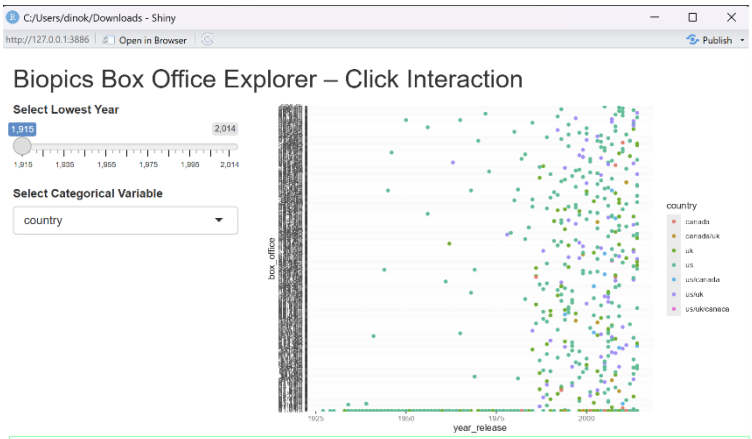






Graphics – Clicking

```r
library(shiny)
library(ggplot2)
library(dplyr)

# Read data
biopics <- read.csv("C:/Users/dinok/OneDrive/Desktop/Uni/1 Semester/Database Management/Shiny/

categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- fluidPage(

  titlePanel("Biopics Box Office Explorer – Click Interaction"),

  fluidRow(
    column(
      width = 4,

      sliderInput(
        "year_filter",
        "Select Lowest Year",
        min = 1915,
        max = 2014,
        value = 1915
      ),

      selectInput(
        "color_select",
        "Select Categorical Variable",
        choices = categoricalVars
      )
    ),

    column(
      width = 8,
      plotOutput("movie_plot", click = "plot_click")
    )
  ),

  fluidRow(
    column(
      width = 12,
      h4("Movie selected by clicking on plot"),
      tableOutput("clicked_movie")
    )
  )
)

server <- function(input, output) {

  # Reactive dataset filtered by year
  biopics_filtered <- reactive({
    biopics %>%
      filter(year_release > input$year_filter)
  })

  # Plot
  output$movie_plot <- renderPlot({
    ggplot(biopics_filtered()) +
      aes(
        x = year_release,
        y = box_office,
        color = .data[[input$color_select]]
      ) +
      geom_point()
  })

  # Table shows closest point to click
  output$clicked_movie <- renderTable({
    req(input$plot_click)
    nearPoints(biopics_filtered(), input$plot_click)
  })
}

shinyApp(ui, server)
```

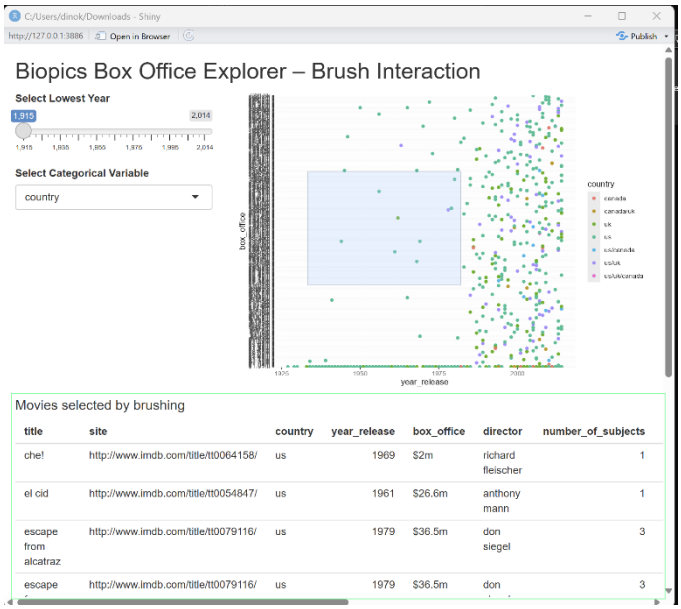


Movie selected by clicking on plot

title	site	country	year_release	box_office	director	number_of_subjects	sub
sergeant york	http://www.imdb.com/title/tt0034167/	us	1941	$16.4m	howard hawks	1	alvir york

Graphics – Brushing

```r
library(shiny)
library(ggplot2)
library(dplyr)

biopics <- read.csv("C:/Users/dinok/OneDrive/Desktop/Uni/1 Semester/Database Management/Shiny/

categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- fluidPage(

  titlePanel("Biopics Box Office Explorer – Brush Interaction"),

  fluidRow(
    column(
      width = 4,

      sliderInput(
        "year_filter",
        "Select Lowest Year",
        min = 1915,
        max = 2014,
        value = 1915
      ),

      selectInput(
        "color_select",
        "Select Categorical Variable",
        choices = categoricalVars
      )
    ),

    column(
      width = 8,
      plotOutput("movie_plot", brush = "plot_brush")
    )
  ),

  fluidRow(
    column(
      width = 12,
      h4("Movies selected by brushing"),
      tableOutput("brushed_movies")
    )
  )
)

server <- function(input, output) {

  biopics_filtered <- reactive({
    biopics %>%
      filter(year_release > input$year_filter)
  })

  output$movie_plot <- renderPlot({
    ggplot(biopics_filtered()) +
      aes(
        x = year_release,
        y = box_office,
        color = .data[[input$color_select]]
      ) +
      geom_point()
  })

  # Table shows all brushed points
  output$brushed_movies <- renderTable({
    req(input$plot_brush)
    brushedPoints(biopics_filtered(), input$plot_brush)
  })
}

shinyApp(ui, server)
```



Graphics Validation & Notification

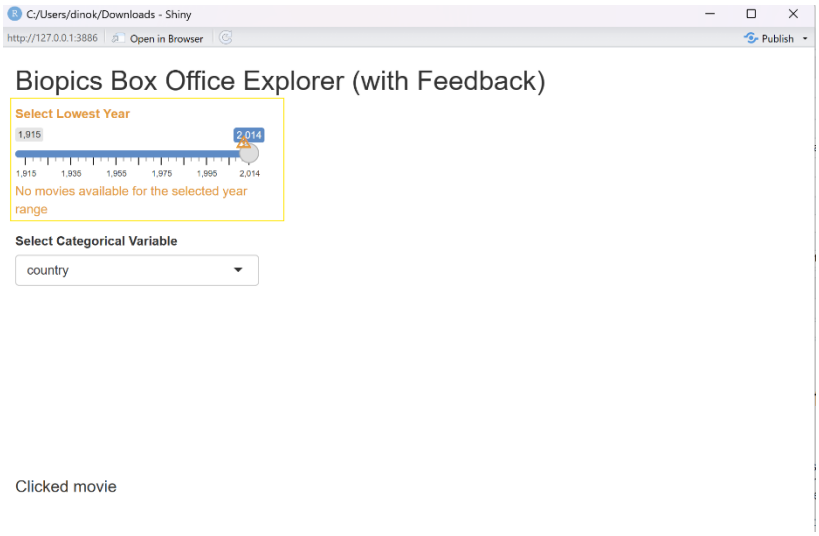
```r
biopics <- read.csv("C:/Users/dinok/OneDrive/Desktop/Uni/1 Semester/Database Management/Shiny/

categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- fluidPage(

  # ---- Enable feedback functionality ----
  useShinyFeedback(),

  titlePanel("Biopics Box Office Explorer (with Feedback)"),

  fluidRow(
    column(
      width = 4,

      sliderInput(
        "year_filter",
        "Select Lowest Year",
        min = 1915,
        max = 2014,
        value = 1915
      ),

      selectInput(
        inputId = "color_select",
        label = "Select Categorical Variable",
        choices = categoricalVars
      )
    ),

    column(
      width = 8,
      plotOutput(
        "movie_plot",
        click = "plot_click"
      )
    )
  ),

  fluidRow(
    column(
      width = 12,
      h4("Clicked movie"),
      tableOutput("clicked_movie")
    )
  )
)

server <- function(input, output, session) {

  # ---- Reactive data with validation ----
  biopics_filtered <- reactive({

    data <- biopics %>%
      filter(year_release > input$year_filter)

    # ---- VALIDATION FEEDBACK ----
    feedbackWarning(
      inputId = "year_filter",
      show = nrow(data) == 0,
      text = "No movies available for the selected year range"
    )

    # Stop downstream code if no data
    req(nrow(data) > 0)

    data
  })

  # ---- Plot ----
  output$movie_plot <- renderPlot({
    ggplot(biopics_filtered()) +
      aes(
        x = year_release,
        y = box_office,
        color = .data[[input$color_select]]
      ) +
      geom_point()
  })

  # ---- Click interaction + notification ----
  output$clicked_movie <- renderTable({
    req(input$plot_click)

    showNotification(
      "Movie selected from plot",
      type = "message",
      duration = 2
    )

    nearPoints(biopics_filtered(), input$plot_click)
  })
}

shinyApp(ui, server)
```

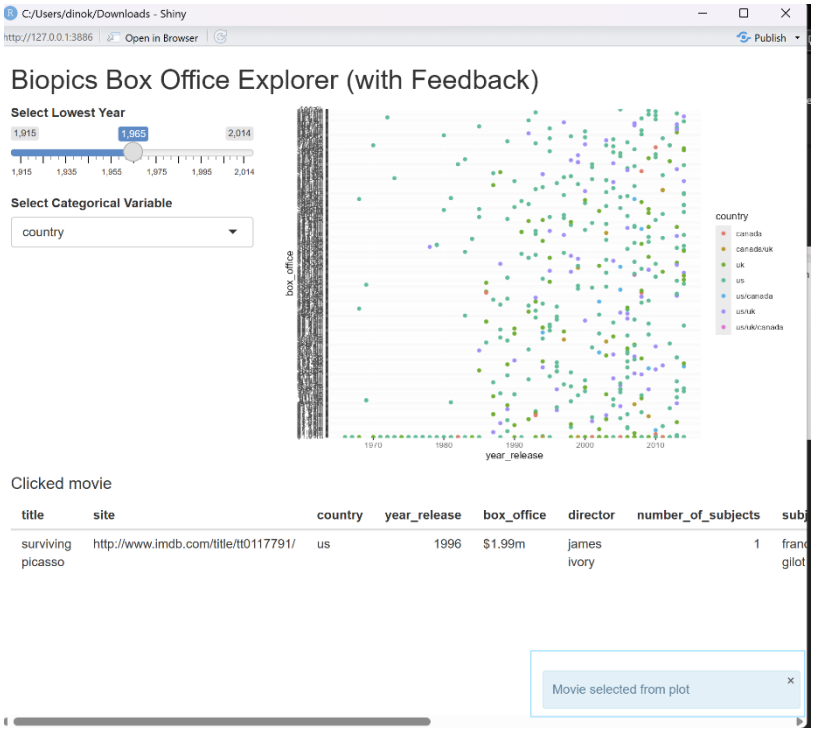
Read this like a sentence:
"Show a warning under the year slider IF the filtered data has zero rows."
Breakdown:
inputId = "year_filter" → attach warning to the slider
nrow(data) == 0 → when no data remains
text → message user sees
✅ If data exists → warning disappears
❌ If data empty → warning appears
This is non-blocking feedback — the app is still running.

Validation

What req() does:
"Do NOT allow any output below me to run unless this condition is TRUE."
So:
If nrow(data) > 0 ✅ → continue
If nrow(data) == 0 ❌ → STOP here

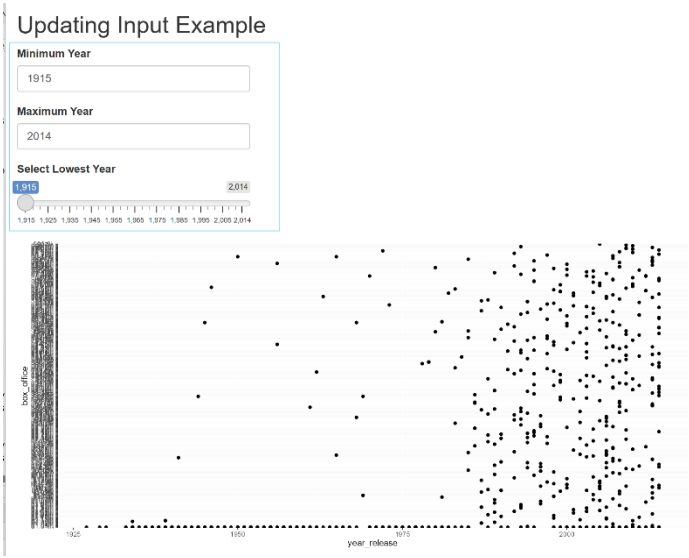
Notification

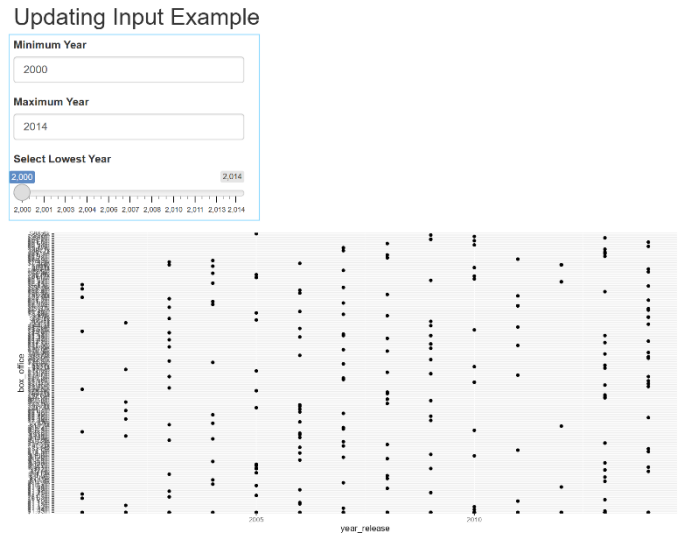




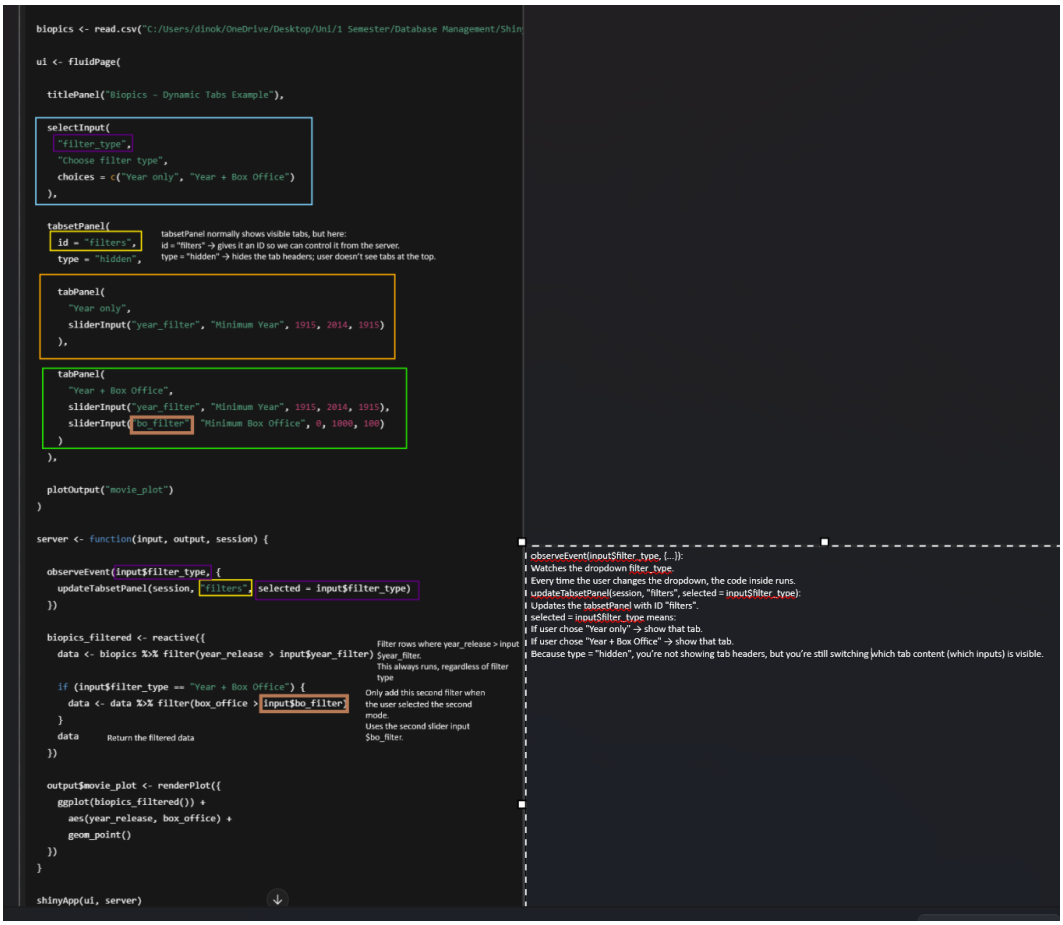
UPDATING INPUTS (updateSliderInput())

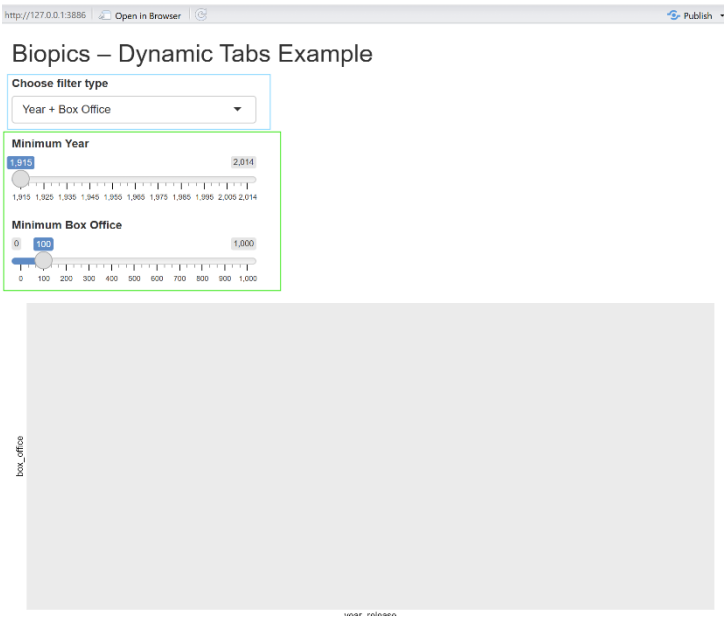
```r
library(shiny)
library(dplyr)
library(ggplot2)

biopics <- read.csv("C:/Users/dinok/OneDrive/Desktop/Uni/1 Semester/Database Management/Shiny/

ui <- fluidPage(

  titlePanel("Updating Input Example"),

  numericInput("min_year", "Minimum Year", 1915),
  numericInput("max_year", "Maximum Year", 2014),

  sliderInput(
    "year_filter",
    "Select Lowest Year",
    min = 1915,
    max = 2014,
    value = 1915
  ),

  plotOutput("plot")
)

server <- function(input, output, session) {

  observeEvent(input$min_year, {
    updateSliderInput(session, "year_filter", min = input$min_year)
  })

  observeEvent(input$max_year, {
    updateSliderInput(session, "year_filter", max = input$max_year)
  })

  output$plot <- renderPlot({
    biopics %>%
      filter(year_release > input$year_filter) %>%
      ggplot(aes(year_release, box_office)) +
      geom_point()
  })
}

shinyApp(ui, server)
```

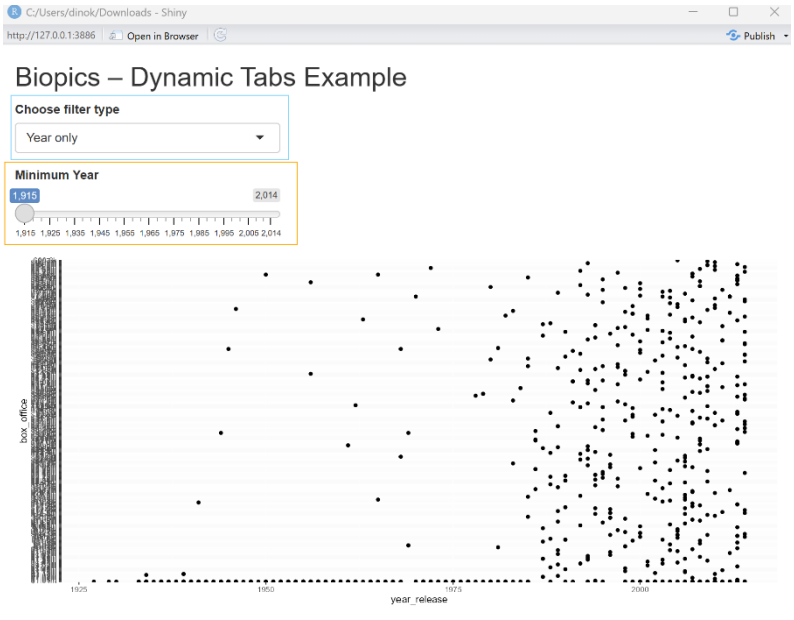




Updating Inputs DYNAMIC TABS (tabsetPanel(type = "hidden"))



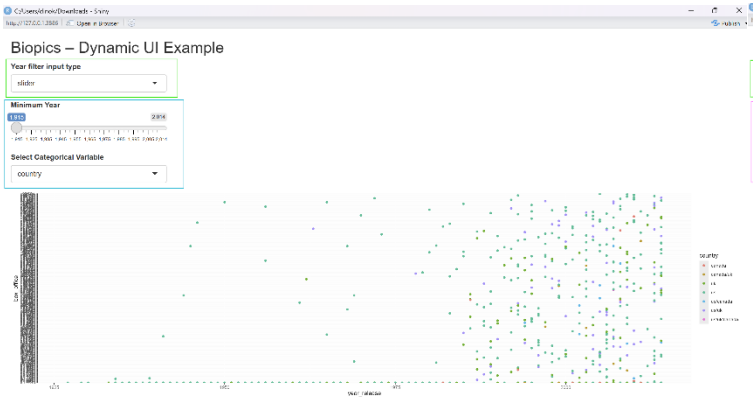


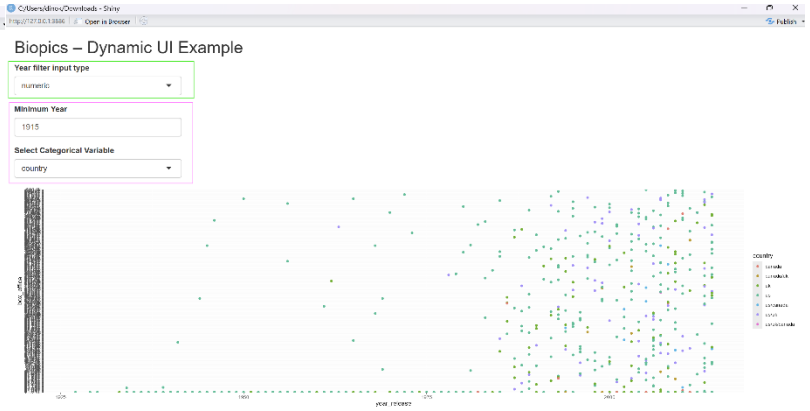


Updating inputs DYNAMIC UI (uiOutput() + renderUI())

```r
biopics <- read.csv("C:/Users/dinok/OneDrive/Desktop/Uni/1 Semester/Database Management/Sh

categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- fluidPage(

  titlePanel("Biopics - Dynamic UI Example"),

  selectInput(
    "input_type",
    "Year filter input type",
    choices = c("slider", "numeric")
  ),

  uiOutput("year_ui"),

  selectInput(
    "color_select",
    "Select Categorical Variable",
    choices = categoricalVars
  ),

  plotOutput("movie_plot")
)

server <- function(input, output, session) {

  output$year_ui <- renderUI({
    if (input$input_type == "slider") {
      sliderInput("year_filter", "Minimum Year", 1915, 2014, 1915)
    } else {
      numericInput("year_filter", "Minimum Year", 1915)
    }
  })

  biopics_filtered <- reactive({
    biopics %>% filter(year_release > input$year_filter)
  })

  output$movie_plot <- renderPlot({
    ggplot(biopics_filtered()) +
      aes(year_release, box_office, color = .data[[input$color_select]]) +
      geom_point()
  })
}

shinyApp(ui, server)
```

outputSyear_ui corresponds to uiOutput("year_ui") in the UI.
renderUI({...}) returns UI elements instead of text/plots/tables.
Inside:
if (inputSinput_type == "slider") { ... } else { ... }
Checks what the user selected in the first dropdown:
If "slider" → create a slider input.
If "numeric" → create a numeric input.
When "slider" is chosen:
sliderInput("year_filter", "Minimum Year", 1915, 2014, 1915)
Creates a slider with:
inputId = "year_filter"
Label: "Minimum Year"
Min = 1915, max = 2014, default value = 1915
When "numeric" is chosen:
numericInput("year_filter", "Minimum Year", 1915)
Creates a numeric input box with:
inputId = "year_filter"
Label: "Minimum Year"
Default value = 1915





Bookmark

Turn the ui into a function

```r
library(shiny)
library(ggplot2)
library(dplyr)

biopics <- read.csv("C:/Users/dinok/OneDrive/Desktop/Uni/1 Semester/Database Management/Shiny/

categoricalVars <- c("country", "type_of_subject", "subject_sex")

ui <- function(request) {
  fluidPage(

    titlePanel("Biopics Explorer (Bookmarkable)"),

    fluidRow(
      column(
        4,
        sliderInput("year_filter", "Minimum Year", 1915, 2014, 1915),
        selectInput("color_select", "Color by", choices = categoricalVars),
        bookmarkButton()
      ),
      column(
        8,
        plotOutput("movie_plot")
      )
    )
  )
}

server <- function(input, output, session) {

  biopics_filtered <- reactive({
    biopics %>% filter(year_release > input$year_filter)
  })

  output$movie_plot <- renderPlot({
    ggplot(biopics_filtered()) +
      aes(year_release, box_office, color = .data[[input$color_select]]) +
      geom_point()
  })
}

shinyApp(ui, server, enableBookmarking = "url")
```