R Shiny Workshop

Welcome to the introductory data visualization in R workshop offered by the Carnegie Mellon University Libraries. This workshop will focus on introduction to data visualization techniques in RStudio for R Shiny, including various introductory exercises. This handout will focus on these exercises, including walking you step by step through creating various visualizations.

This handout follows the LinkedInLearning Tutorial for R Shiny, “[Building Data Apps with R and Shiny: Essential Training](https://www.linkedin.com/learning/building-data-apps-with-r-and-shiny-essential-training/)” by [Charlie Joey Hadley](https://www.linkedin.com/learning/instructors/charlie-joey-hadley?u=42257553)

You will need to have downloaded RStudio to complete these exercises: https://www.rstudio.com/products/rstudio/download/

Notes:

The following exercises assume that you have a basic understanding of R coding, and will focus on introducing students to data visualization in R only.

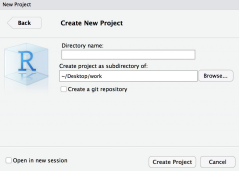
Many exercises were modeled off of the Data Carpentries workshop ‘Data Analysis and Visualization in R for Ecologists’, François Michonneau & Auriel Fournier (Lesson Maintainers) (von Hardenberg A et al. 2018).

Exercise 1: Setting up a project in RStudio

Let’s start by creating a new R directory and R project.

Within RStudio go to ‘File’ , ‘New Project’. This will open up a new window ‘Create Project’. Choose ‘New Directory’.

This will open up a new option ‘Project Type’. Select ‘New Project’.



You will now have the option to set up a new project giving it a ‘Directory name’. You are also offered the opportunity to select where this project will be stored.

You will now be able to work within this project for the remainder of the workshop.

In order to make a clean working procedure, create new files within this project where we can store our scripts. You can do this by clicking on the ‘New Folder’ button in the sub-window at the bottom right hand corner of your RStudio page. Create new files called ‘data’, ‘outputs’ .

## Exercise 1: Adding our data

For this workshop, we will be using data supplied by the LinekdInLearningearning session “[Building Data Apps with R and Shiny: Essential Training](https://www.linkedin.com/learning/building-data-apps-with-r-and-shiny-essential-training/)” .

You will need to have download the corresponding files mentioned in parts of that tutorial ([Folder](https://drive.google.com/drive/folders/1j0oQl8RLNTRkAtbTeGOwXQcM97E_-u86?usp=sharing); [1.1](https://drive.google.com/drive/folders/1BCiFA4xTB1kDqkaiyJNDHjCb0QJxtJa7?usp=drive_link), [2.1](https://drive.google.com/drive/folders/193wjPMdbaXokwzjECG_0Bm2EkwYBHdxg?usp=sharing), [3.1](https://drive.google.com/drive/folders/1SF2u_7fAbuQMhyB3pZPoINbBujyOZ-Ob?usp=sharing),[4.2](https://drive.google.com/drive/folders/1XtWMeNXBFJ92BCzp5PQVVifmIzL3lXOW?usp=drive_link), [4.4](https://drive.google.com/drive/folders/1IYiVwHIPJzB-wmFo-V-z071Ave4fziRy?usp=drive_link) [5.2](https://drive.google.com/drive/folders/1oVcJpFV_IgRXxVKUgtB3Dq7RJ8QiBtX4?usp=sharing), [6.1](https://drive.google.com/drive/folders/1KZbnxaN6SxDopl1jb3H7-7DWiuUv-SI6?usp=drive_link), [6.2](https://drive.google.com/drive/folders/1ea1xi91TfYtOBoOjZU-w-3ECNOFDK8MZ?usp=drive_link), [6.3](https://drive.google.com/drive/folders/1BWykrEhoeoTGHn2taL1ImkbqORG3YPKU?usp=drive_link)).

## Exercise 2: Working with R Shiny

*What is Shiny?*: Shiny is a self-contained web framework for building interactive web apps with R code. An individual shiny app is an interactive web app viewed in a web browser.

Shiny apps are fully self-contained, and all that is needed to run a Shiny app on a local machine is R downloaded and the R Shiny server product. Shiny apps run their R code on the server they are connected to - not within the web browser of the end user. The R package will run the server package on the local host machine.

If you want your shiny app to run in a web browser so others can interact with it on their machines, you can use shinyapps.io. Shinyapps.io is provided by R Studio, but is only available for free for limited tasks, personal use, or testing capabilities. The free teer caps how many apps you can host at any given time. R Studio allows you to manage shiny apps that can then be made public in shinyapps.io

There are two different forms Shiny Apps can take: (1) Single-File app or an (2) Split-File App.

A single-file configuration puts both the server and user interface code in a single app.R file, whereas the split-file configuration puts them in their own separate files. Functionally, these configurations will produce the same app.

What belongs in an ui.R file?

The ui.R file defines the user experience or interface of your shiny app.

In these files, you need to include the controls your users will use while engaging with the app. These are called *input widgets*.

ui.R files do not allow for the direct creation of charts or other R outputs. These will be rendered by the server.R file. You can also not access direct values from input widgets in ui.R

What belongs in an server.R file?

The server.R file is what allows you to run code in your shiny app, and allow for analytic functions to work in a shiny app. As such, this file is usually larger and more complex than the ui.R file.

Your server.R file is also where you can engage with underlying data files, including updating content that can be evaluated by input widgets. [ You can do this using the updateSelectInput() function]

To explore both types of apps, you can either use the app.R file in folder 2.1 for the single-file app example. We will be building a split-file app in our next exercise.

### 2.1 Testing Shiny Apps:

To begin working with R Shiny, we need to first install and load Tidyverse which will allow us to work with the files we downloaded.

Install.packages(“tidyverse”)

library(tidyverse)

Once we have installed and loaded the tidyverse, let's first see what R Shiny looks like under the hood. By opening from file folder ***1.1 the 01\_01.Rproj***, ***server.R*** file, and ***data-processing.R*** files we can launch an R Shiny app.

In the server.R file, we are given the option to select the ‘Run App’ button. This will open the R Shiny app in a new pop-up page. From here, you will be asked to choose a continent, a country, and an indicator. Then select the update chart to view the interactive graphs.

By exploring this tool in R Shiny, we can begin to learn about what is possible for us to do in tools we design ourselves.

### 2.2 Building a Split-File Shiny App:

To build a Split-File shiny app, follow these steps:

1. In a new R Project, create two script files. The first file we will create as ‘ui.R’. The second file should be saved as ‘server.R’.

You will notice that once these two files have been created, inside the ‘server.R’ script file, instead of the typical ‘Run’ button, you will see a ‘Run App’ button.

1. Starting in the ui.R file, we need to set the parameters for how our shiny app will look.

With the function navbarPage () we can set up a horizontal navigation bar. For example, we can add a title and some content for our navigation bar.

navbarPage (

“My First Shiny App”,

tabPanel (“About”),

tabPanel (“Contact”) )

1. To actually see the above text in play, we will need to create an R server file.

In the server.R file, open up a server using the following code:

function(input, output, session){

}

Once we add this content, we can now click the ‘Run App’ button and it will result in a

new shiny app.

### 2.3. Building A More Complex Split-File

Let’s practice building a more complex Split-File shiny app, following these steps:

1. In folder [3.1](https://drive.google.com/drive/folders/1SF2u_7fAbuQMhyB3pZPoINbBujyOZ-Ob?usp=drive_link), open the ui.R and the server.R files.

You can also open curve-plot.R, which contains the code for the graphing function we

will be turning into a shiny app.

Our shiny app will explore the ability for us to change the exponent in this graph using an

R Shiny slider widget.

1. In the ui.R file, we can add the slider widget which will enable us to change the exponent. When working with the sliderInput, the first thing we need to specify is what is the Input ID for our control.

fluidPage(

sliderInput(“exponent”, label = “Choose an Exponent”,

min = 1,

max = 5,

value = 2)

)

1. Now we need to create the curve plot in our server.R file so it can be run through the ui.R file.

function(input, output, session){

output$curve\_plot <- renderPlot({

curve(x^2, from = -5, to = 5)

})

1. Going back to our ui.R file, we can now add content related specifically to the curve function.

fluidPage(

sliderInput(“exponent”, label = “Choose an Exponent”,

min = 1,

max = 5,

value = 2),

plotOutput(“curve\_plot”)

)

1. To ensure that our curve\_plot will update based on our slider, we go back to the survey.R file.

function(input, output, session){

output$curve\_plot <- renderPlot({

curve(x^input$exponent, from = -5, to = 5)

})

## 3. Input and Output Variables

Every single input widget in R Shiny has at least one input variable in the resulting app. As such, input widgets can also be referred to as control widgets. Outputs are created in the server.R file so we can render content to be sent to the client through the ui.R file.

NEVER included input variables, such as input$var. NEVER include output variables such as output$plot. Instead, always add quotation marks on either side of the variables such as *plotOutput(“plot”)*.

If you want to create new content to display in the client side of the app, you need to add them as an output object by assigning values to them. You will also need to choose the right render function for what you need to send to the client side of the app.

To display content from your server.R file in the ui.R file you must use pairs of render and output functions. For example, in the server.R file you could use the code renderPlot AND in the ui.R file you could use the code plotOutput. Or to render text in your shiny app, you could use renderText in the server.R file AND textOutput in the ui.R file.

It is important to remember that you cannot duplicate inputs or outputs.

## 4. Loading Data into R Shiny

All data processing, importing and accessing happens in the server.R file regardless of the data source. Data files can be directly entered into a shiny app directory, but it is not recommended. Instead, external data can be accessed via API or database connections, which is supported by R Studio as well.

When adding data into a Shiny project, it is a good idea to add the file into the same working directory on your hard drive as the server.R and ui.R files. R conventions suggest that you put all of your data files into a sub-folder in this directory called ‘data’.

Lets see how this works in action for working with interactive data tables.

1. Open file folder [5.2](https://drive.google.com/drive/folders/1oVcJpFV_IgRXxVKUgtB3Dq7RJ8QiBtX4?usp=drive_link) and open the server.R and ui.R scripts.
2. To integrate our data we will need to install the DT package, as well as load the library tidyverse.

install.package(“DT”)

library(DT)

library(tidyverse)

1. We can then create our variable called wdi\_data <- read\_csv(“data/wdi\_data.csv”)

wdi\_data <- read\_csv(“data/wdi\_data.csv”)

1. In the console, we can then check to see the data is uploaded without messing with our server.R or ui.R script files by typing

Wdi\_data %>% datatable()

1. Once we have confirmed that the data loaded works, we can add it into our server.R code

function(input, output, session) {

output$wdi\_table <- renderDT({wdi\_data %>% datatable(rownames = FALSE)

})

### 4.1 Populate Tables with Data

Lets try to use some data examples in context!

1. Load file folder [6.1](https://drive.google.com/drive/folders/1KZbnxaN6SxDopl1jb3H7-7DWiuUv-SI6?usp=drive_link) and then open the server.R and ui.R files. You can then run the app to see what data is included already in the Shiny app.
2. When we open the ui.R file, we can see that the code is difficult to read as we have listed all of the country names individually. Here, it would be better to have a pull down data as right now all of these countries are hard coded and any new countries added to the data set would not be shown.We can update our ui.R file to do this.
3. To update the ui.R file with a pull down add this code to the tabPanel function:

tabPanel (“With updateSelectInput(),

fluidPage(

selectInput(“select\_country\_with\_updateSelectInput,

Label = “Choose a Country:”,

choices = NULL ) ,

plotOutput(“internet\_use\_with\_updateSelectInput”,

choices = unique(wdi\_data$country))

1. In the corresponding server.R file, add to the Shiny server function (new code in green):

function(input, output, session) {

updateSelectInput(session,

“select\_country\_with\_updateSelectInput”)

output$internet\_use\_without\_updateSelectInput <- renderPlot({

wdi\_data %>%

filter(

country == input$select\_country\_without\_updateSelectInput,

indicator == "IT.NET.USER.ZS"

) %>%

filter(!is.na(value)) %>%

ggplot(aes(x = year, y = value)) +

geom\_path() +

labs(

title = paste("Individuals using the Internet (% of population)", "in", input$select\_country\_without\_updateSelectInput),

subtitle = "Data source: WDI Package, see data/world-bank.R for

details"

)

})

1. Then we need to go to our server.R file to add in a chart, so we can see the data. We can start by copy and pasting the output$internet\_use\_without\_updateSelectInput below the original text and updating the text to be with instead of without… (new code in green)

function(input, output, session) {

updateSelectInput(session,

“select\_country\_with\_updateSelectInput”)

output$internet\_use\_without\_updateSelectInput <- renderPlot({

wdi\_data %>%

filter(

country == input$select\_country\_without\_updateSelectInput,

indicator == "IT.NET.USER.ZS"

) %>%

filter(!is.na(value)) %>%

ggplot(aes(x = year, y = value)) +

geom\_path() +

labs(

title = paste("Individuals using the Internet (% of population)", "in", input$select\_country\_without\_updateSelectInput),

subtitle = "Data source: WDI Package, see data/world-bank.R for

details"

)

output$internet\_use\_with\_updateSelectInput <- renderPlot({

wdi\_data %>%

filter(

country == input$select\_country\_with\_updateSelectInput,

indicator == "IT.NET.USER.ZS"

) %>%

filter(!is.na(value)) %>%

ggplot(aes(x = year, y = value)) +

geom\_path() +

labs(

title = paste("Individuals using the Internet (% of population)", "in", input$select\_country\_with\_updateSelectInput),

subtitle = "Data source: WDI Package, see data/world-bank.R for

details"

)

})

### 4.2 Labeling Choices in Selecting Data

Let’s try adding labels with meaning…

(1) Load file folder [6.2](https://drive.google.com/drive/folders/1ea1xi91TfYtOBoOjZU-w-3ECNOFDK8MZ?usp=drive_link) and then open the server.R and ui.R files. You can then run the app to see what data is included already in the Shiny app.

(2) In the server.R file, scroll down to lines 21 to 25. Here you can see which values are set to represent the second pull down box in our shiny app, otherwise known as our named vectors.

(3) We can create our own named vectors through the following process.

1. Load the indicators csv file from our data folder into the R Project.

Library(tidyverse)

indicators <- read\_csv(“data/indicators.csv”)

1. Then in a new script file or in your console, we can set indicators as names vectors using:

setNames(indicators$indicator\_code, indicators$indicator\_name)

(4) In our server.r file we can add the code on line 19:

updateSelectIndicator(session,

“selected\_indicator\_labelled\_from\_data”,

choices = setNames(indicators$indicator\_code, indicators$indicator\_name)

)

(5) We can then create our data visualization in our server.R file. We can do so by copying and pasting again the following code from the server.R file with the following changes: [new code in green]

wdi\_data %>%

filter(

country == input$select\_country\_with\_manual\_labels

) %>%

filter(indicator == input$selected\_indicator\_labelled\_manual) %>%

filter(!is.na(value)) %>%

ggplot(aes(x = year, y = value)) +

geom\_path() +

labs(

title = paste(

input$selected\_indicator\_labelled\_manual,

"in",

input$select\_country\_with\_manual\_labels

),

subtitle = "Data source: WDI Package, see data/world-bank.R for details"

)

wdi\_data %>%

filter(

country == input$select\_country\_with\_data\_labels

) %>%

filter(indicator == input$selected\_indicator\_labelled\_from\_data) %>%

filter(!is.na(value)) %>%

ggplot(aes(x = year, y = value)) +

geom\_path() +

labs(

title = paste(

input$selected\_indicator\_labelled\_from\_data,

"in",

input$select\_country\_with\_datal\_labels

),

subtitle = "Data source: WDI Package, see data/world-bank.R for details"

)

})

(6) Finally, it is always important to ensure that our title matches the content of our Shiny app, if only so we can remember what the app is evaluating. Look into the code for renderPlot and create new variable for selected indicator name: [new code in green]

output$chart\_manual\_labels <- renderPlot({

selected\_indicator\_name<- indicators %>%

filter(indicator\_code == input$selected\_indicator\_labelled\_from\_data) %>%

pull(indicator\_name)

wdi\_data %>%

filter(

country == input$select\_country\_with\_data\_labels

) %>%

filter(indicator == input$selected\_indicator\_labelled\_from\_data) %>%

filter(!is.na(value)) %>%

ggplot(aes(x = year, y = value)) +

geom\_path() +

labs(

title = paste(

input$selected\_indicator\_labelled\_from\_data,

"in",

input$select\_country\_with\_datal\_labels

),

subtitle = "Data source: WDI Package, see data/world-bank.R for details"

)

})

### 4.3 Independent Controls to Filter Variables

Let's practice building some independent controls for filtering:

1. Load file folder [6.3](https://drive.google.com/drive/folders/1BWykrEhoeoTGHn2taL1ImkbqORG3YPKU?usp=drive_link) and then open the server.R and ui.R files. You can then run the app to see what data is included already in the Shiny app.
2. We need to add elements to help us filter out the content we need, related to the continent and country. To do se we can build an observed event (adding code to replace the second updateSelectInputs function):

observeEvent(c(input$selected\_continent), {

countries\_in\_continent <- %>%

filter(continent == input$selected\_continent)

pull(country)

UpdateSelectInput(session,

“selected\_countries”,

choices = countries\_in\_continent)

})

## 5. Building Tables in Shiny Apps

Lets try making some tables in R Shiny…

1. Load file folder [4.2](https://drive.google.com/drive/folders/1XtWMeNXBFJ92BCzp5PQVVifmIzL3lXOW?usp=drive_link) and then open the server.R and ui.R files. You can then run the app to see what data is included already in the Shiny app.
2. To customize the table that shows star wars characters by height by using the striped function to add clarity in table design. [new code in green]

function(input, output, session){

output$star\_narrow <- renderTable({

starwars %>%

select(name, species, homeworld, height) %>%

filter(height <= input$height\_limit\_star\_narrow) %>%

arrange(desc(height))

},

striped = TRUE

)

1. We can further modify the tables appearance and function by using hover() and assigning a value to missing varribels.

function(input, output, session){

output$star\_narrow <- renderTable({

starwars %>%

select(name, species, homeworld, height) %>%

filter(height <= input$height\_limit\_star\_narrow) %>%

arrange(desc(height))

},

striped = TRUE,

hover = TRUE,

na = “[Missing]”

)

### 5.1 We can also make an interactive table

1. Load file folder [4.4](https://drive.google.com/drive/folders/1IYiVwHIPJzB-wmFo-V-z071Ave4fziRy?usp=drive_link) and then open the server.R and ui.R files. You can then run the app to see what data is included already in the Shiny app.
2. In the server.R file, load libraries tidyverse and DT

library(tidyverse)

library(DT)

1. Then we can assign values from our star wars table to build the base of our shiny app visualization of an interactive table:

function(input, output, session) {

output$star\_narrow <- renderDT({

starwars %>%

select(name, height, homeworld) %>%

arrange(desc(height))

})

)

1. We can add more to our code to correct an annoying feature by default, which is that the row name for each feature is shown. We can remove that by doing the following [new code in green]:

function(input, output, session) {

output$star\_narrow <- renderDT({

starwars %>%

select(name, height, homeworld) %>%

arrange(desc(height)),

rownames = FALSE)

})

)

1. We can add more detail to our column by adding new code that focuses on having a wider table and shifting row names, giving us more control is what is seen in the table, by adding the following:

function(input, output, session) {

output$star\_narrow <- renderDT({

starwars %>%

select(name, height, homeworld) %>%

arrange(desc(height)),

rownames = FALSE)

})

)

output$star\_wide <- renderDT({

starwars %>%

select(name:homeworld) %>%

arrange(desc(height)) %>%

datatable(rownames = input$show\_rownames)

})

)

1. We can also make the table responses to window size, showing all data even when the pop up is to narrow.

function(input, output, session) {

output$star\_narrow <- renderDT({

starwars %>%

select(name, height, homeworld) %>%

arrange(desc(height)),

rownames = FALSE)

})

)

output$star\_wide <- renderDT({

starwars %>%

select(name:homeworld) %>%

arrange(desc(height)) %>%

datatable(rownames = input$show\_rownames,

extensions = “Responsive” )

})

)

1. We can also filter out information to focus on the films themselves rather than a basic list we have seen before, creating a detailed list column:

function(input, output, session) {

output$star\_narrow <- renderDT({

starwars %>%

select(name, height, homeworld) %>%

arrange(desc(height)),

rownames = FALSE)

})

)

output$star\_wide <- renderDT({

starwars %>%

select(name:homeworld) %>%

arrange(desc(height)) %>%

datatable(rownames = input$show\_rownames,

extensions = “Responsive” )

})

output$star\_list <- renderDT({

starwars %>%

select(name, film:starships) %>%

datatable(rownames = FALSE)

})

)

If you have any additional questions or wish to attend office hours to receive additional support with visualization you made today or for a project, please email: eslayton@andrew.cmu.edu.

SICSS Specific Example

The following example showcases building an R Shiny web app that can display different facets of the [WPRDC](https://data.wprdc.org/dataset/) [Pittsburgh Police Arrest dataset](https://data.wprdc.org/dataset/arrest-data).

## Exercise 1: Loading Packages

For this particular exploration of R Shiny, we need to make sure that all the libraries and packages we need are loaded. For this example we will need:

library(shiny)

library(shinythemes)

library(ggplot2)

library(dplyr)

library(readr)

library(lubridate)

library(DT)

Please use install.packages() to load any libraries you have not already accessed on your computer.

## Exercise 2: Building an R space

We can then create a specific r file that can be used to house the code we would normally place in an ui.r and server.r script file.

### 2.1a: Defining the ui

First, we can set up a fluidPage in our AI using the following code. Elements included in the UI description are the definition of out pages title, the sidebar layout which will include all of our toggles for different inputs and data loading, as well as our panel outputs.

ui <- fluidPage(

# Title

titlePanel("Pittsburgh Arrests by Gender and Age Groups (2016-2023)"),

# Layout: sidebar with controls, main panel with plot

sidebarLayout(

# SIDEBAR: All the input controls

sidebarPanel(

# Year filter

sliderInput("yearRange",

"Year Range:",

min = 2016, max = 2023,

value = c(2016, 2023)),

# Age group selection

checkboxGroupInput("ageGroups",

"Age Groups:",

choices = c("0-17", "18-35", "36-50", "51-75", "75+"),

selected = c("0-17", "18-35", "36-50", "51-75", "75+")),

# Gender selection

checkboxGroupInput("genderFilter",

"Gender:",

choices = c("M", "F"),

selected = c("M", "F")),

# View option

radioButtons("viewBy",

"View by:",

choices = c("Year", "Season"),

selected = "Year")

),

# MAIN PANEL: Output visualizations

mainPanel(

# Loading message while data is being prepared

conditionalPanel(

condition = "!output.dataReady",

h4("Loading data, please wait...")

),

# Plot output (only shown when data is ready)

conditionalPanel(

condition = "output.dataReady",

plotOutput("arrestsPlot", height = "600px")

)

)

)

)

### 2.1b Defining the server

The following code explores everything you need to add to indicate the server component of the R Shiny Application. This code should follow the above provided for the ui, in the same r script. The server, as a reminder, accesses inputs selected by the user to perform computations and specifies how outputs laid out in the UI should be updated. As such, the code details the data that will be uploaded as well as how the ui interacts with that data.

# Server logic

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

# Variable to track if data is ready to display

dataReady <- reactiveVal(FALSE)

# Create reactive output for UI conditional panel

output$dataReady <- reactive({ dataReady() })

outputOptions(output, "dataReady", suspendWhenHidden = FALSE)

# Load data once at startup

arrests\_data <- reactive({

# Show message that app is loading data

withProgress(message = 'Loading data...', {

# Load WPRDC data

arrests <- read.csv("https://data.wprdc.org/datastore/dump/e03a89dd-134a-4ee8-a2bd-62c40aeebc6f")

# Clean the data (simplified version of your cleaning process)

arrests\_cleaned <- arrests %>%

mutate(

ARRESTTIME = ymd\_hms(ARRESTTIME),

year = year(ARRESTTIME),

season = case\_when(

month(ARRESTTIME) %in% c(12, 1, 2) ~ "Winter",

month(ARRESTTIME) %in% c(3, 4, 5) ~ "Spring",

month(ARRESTTIME) %in% c(6, 7, 8) ~ "Summer",

TRUE ~ "Fall"

),

ages = case\_when(

AGE >= 0 & AGE <= 17 ~ "0-17",

AGE >= 18 & AGE <= 35 ~ "18-35",

AGE >= 36 & AGE <= 50 ~ "36-50",

AGE >= 51 & AGE <= 75 ~ "51-75",

AGE > 75 ~ "75+",

TRUE ~ "Unknown"

)

) %>%

filter(year >= 2016 & year <= 2023,

!is.na(AGE), AGE <= 100,

GENDER %in% c("M", "F"))

})

# Signal that data is ready to display

dataReady(TRUE)

return(arrests\_cleaned)

})

# Create filtered dataset based on user inputs

filtered\_data <- reactive({

# Get base dataset

data <- arrests\_data()

# Apply user filters

data %>%

filter(year >= input$yearRange[1] & year <= input$yearRange[2]) %>%

filter(ages %in% input$ageGroups) %>%

filter(GENDER %in% input$genderFilter)

})

# Generate the plot

output$arrestsPlot <- renderPlot({

# Get filtered data

data <- filtered\_data()

# Determine facet variable based on view selection

facet\_var <- if(input$viewBy == "Year") "year" else "season"

# Order age groups correctly (for x-axis)

data$ages <- factor(data$ages,

levels = c("0-17", "18-35", "36-50", "51-75", "75+"))

# Create appropriate facet column based on user selection

if(facet\_var == "year") {

data$facet\_group <- factor(data$year)

facet\_title <- "Year"

} else {

data$facet\_group <- factor(data$season,

levels = c("Winter", "Spring", "Summer", "Fall"))

facet\_title <- "Season"

}

# Create the plot

ggplot(data, aes(x = ages, fill = GENDER)) +

geom\_bar(alpha = 0.7) +

facet\_wrap(~ facet\_group) +

scale\_fill\_manual(values = c("M" = "lightblue", "F" = "darkgreen")) +

labs(

title = "Count of Arrests by Gender and Age Group",

x = "Age Group",

y = "Count",

fill = "Gender"

) +

theme\_minimal() +

theme(

plot.title = element\_text(hjust = 0.5, size = 16),

axis.text.x = element\_text(angle = 45, hjust = 1)

)

})

}

# Run the application

shinyApp(ui = ui, server = server)

### 2.2a Defining the ui (Long form)

We can start by specifying all the details that will need to come in our UI.

First, we can set up a fluidPage in our AI using the following code, including incorporating an nicer theme for our use. Elements included in the UI description are the definition of out pages title, the sidebar layout which will include all of our toggles for different inputs and data loading, as well as our panel outputs. Parts of this code also explore the additional informational pages connected to the data, including a data preview as well as information about the data and its collection / ownership.

Code:

ui <- fluidPage(

# Use a nicer theme

theme = shinytheme("flatly"),

# Application title

titlePanel("Pittsburgh Arrests Analysis by Gender and Age (2016-2023)"),

# Sidebar layout

sidebarLayout(

# Sidebar panel for inputs

sidebarPanel(

# Direct data load option

actionButton("loadData", "Load WPRDC Arrests Data",

icon = icon("database"),

style = "color: #fff; background-color: #337ab7; border-color: #2e6da4"),

tags$div(style = "margin-top: 10px;"),

# Alternative file upload input

fileInput("dataFile", "Or Upload Your Own CSV File",

accept = c(".csv"),

placeholder = "Select arrests data file..."),

tags$hr(),

# Year range filter

sliderInput("yearRange",

"Select Year Range:",

min = 2016,

max = 2023,

value = c(2016, 2023),

step = 1,

sep = ""),

# Age groups to include

checkboxGroupInput("ageGroups",

"Age Groups to Display:",

choices = c(

"0-17" = "0-17",

"18-35" = "18-35",

"36-50" = "36-50",

"51-75" = "51-75",

"75+" = "75+"

),

selected = c("0-17", "18-35", "36-50", "51-75", "75+")),

# Add seasonal breakdown option

checkboxInput("showSeasonal", "Show Seasonal Breakdown Instead of Years", value = FALSE),

# Gender filter

checkboxGroupInput("genderFilter",

"Gender:",

choices = c("M", "F"),

selected = c("M", "F")),

# Add a horizontal line for visual separation

tags$hr(),

# Add a download button for the plot

downloadButton("downloadPlot", "Download Plot"),

# Brief instructions

tags$div(

style = "margin-top: 20px;",

tags$p("Click 'Load WPRDC Arrests Data' to automatically fetch the Pittsburgh police arrests data from WPRDC, or upload your own CSV file."),

tags$p("Use the filters to customize the visualization.")

)

),

# Main panel for outputs

mainPanel(

# Create tabset for different views

tabsetPanel(

# First tab shows the main plot

tabPanel("Plot",

br(),

# Show a loading message while the plot is being generated

conditionalPanel(

condition = "!output.dataLoaded",

h4("Please load data to display the plot")

),

# Container for the plot

plotOutput("arrestsPlot", height = "600px"),

# Add some explanatory text below the plot

conditionalPanel(

condition = "output.dataLoaded",

tags$div(

style = "margin-top: 20px;",

h4("About This Visualization"),

p("This plot shows the distribution of arrests by gender across different age groups."),

p("The data is faceted by year or season, depending on your selection."),

p("The bar heights represent the count of arrests for each gender and age group combination.")

)

)

),

# Second tab for data preview

tabPanel("Data Preview",

br(),

# Container for the data table

conditionalPanel(

condition = "output.dataLoaded",

DTOutput("dataTable")

)

),

# Third tab with information about the app

tabPanel("About",

br(),

h3("About This Application"),

p("This Shiny application visualizes Pittsburgh Bureau of Police arrest data trends by gender and age group over time (2016-2023)."),

h4("Data Source"),

p("The application is designed to work with Pittsburgh Bureau of Police arrest data from the Western Pennsylvania Regional Data Center (WPRDC)."),

p("Data URL: ",

a("https://data.wprdc.org/datastore/dump/e03a89dd-134a-4ee8-a2bd-62c40aeebc6f",

href = "https://data.wprdc.org/datastore/dump/e03a89dd-134a-4ee8-a2bd-62c40aeebc6f",

target = "\_blank")),

p("Dataset Page: ",

a("https://data.wprdc.org/dataset/arrest-data",

href = "https://data.wprdc.org/dataset/arrest-data",

target = "\_blank")),

#data cleaning

h4("Data Cleaning"),

p("The raw data undergoes the following transformations:"),

tags$pre(

'arrests\_cleaned <- arrests |>

mutate(

ARRESTTIME = parse\_date\_time(ARRESTTIME, orders = "Ymd HMS"),

year\_month = as.Date(floor\_date(ARRESTTIME, "month")),

seasonal = case\_when(

month(ARRESTTIME) %in% c(12, 1, 2) ~ "Winter",

month(ARRESTTIME) %in% c(3, 4, 5) ~ "Spring",

month(ARRESTTIME) %in% c(6, 7, 8) ~ "Summer",

TRUE ~ "Fall"

),

year = year(ARRESTTIME),

ages = case\_when(

AGE >= 0 & AGE <= 17 ~ "0-17",

AGE >= 18 & AGE <= 35 ~ "18-35",

AGE >= 36 & AGE <= 50 ~ "36-50",

AGE >= 51 & AGE <= 75 ~ "51-75",

AGE > 75 ~ "75+",

TRUE ~ "Unknown"

)) |>

filter(year(ARRESTTIME) >= 2016 & year(ARRESTTIME) <= 2023, AGE <= 100, GENDER != "U") |>

filter(Y <= 41.2, X <= -79)'

),

)

)

)

)

)

### 

### 2.2b Defining the Server (Longform)

The following code explores everything you need to add to indicate the server component of the R Shiny Application. This code should follow the above provided for the ui, in the same r script. The server, as a reminder, accesses inputs selected by the user to perform computations and specifies how outputs laid out in the UI should be updated. As such, the code details the data that will be uploaded as well as how the ui interacts with that data.

# Define server logic

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

# Reactive value to track if data is loaded

dataLoaded <- reactiveVal(FALSE)

# Reactive value to store the data

arrestsData <- reactiveVal(NULL)

# Output value to control UI conditional panels

output$dataLoaded <- reactive({

return(dataLoaded())

})

outputOptions(output, "dataLoaded", suspendWhenHidden = FALSE)

# Observer for the load data button

observeEvent(input$loadData, {

# Show a notification that data is loading

id <- showNotification("Loading data from WPRDC...", type = "message", duration = NULL)

# Try to load the data from WPRDC

tryCatch({

# Load the data

arrests <- read\_csv("https://data.wprdc.org/datastore/dump/e03a89dd-134a-4ee8-a2bd-62c40aeebc6f")

# Process the data using the provided cleaning code

arrests\_cleaned <- arrests %>%

mutate(

ARRESTTIME = parse\_date\_time(ARRESTTIME, orders = "Ymd HMS"),

year\_month = as.Date(floor\_date(ARRESTTIME, "month")),

seasonal = case\_when(

month(ARRESTTIME) %in% c(12, 1, 2) ~ "Winter",

month(ARRESTTIME) %in% c(3, 4, 5) ~ "Spring",

month(ARRESTTIME) %in% c(6, 7, 8) ~ "Summer",

TRUE ~ "Fall"

),

year = year(ARRESTTIME),

ages = case\_when(

AGE >= 0 & AGE <= 17 ~ "0-17",

AGE >= 18 & AGE <= 35 ~ "18-35",

AGE >= 36 & AGE <= 50 ~ "36-50",

AGE >= 51 & AGE <= 75 ~ "51-75",

AGE > 75 ~ "75+",

TRUE ~ "Unknown"

)

) %>%

filter(year(ARRESTTIME) >= 2016 & year(ARRESTTIME) <= 2023, AGE <= 100, GENDER != "U") %>%

filter(Y <= 41.2, X <= -79)

# Store the cleaned data

arrestsData(arrests\_cleaned)

# Set dataLoaded to TRUE

dataLoaded(TRUE)

# Show success notification

removeNotification(id)

showNotification("Data loaded successfully!", type = "message", duration = 3)

}, error = function(e) {

# Remove the loading notification

removeNotification(id)

# Show error notification

showNotification(paste("Error loading data:", e$message), type = "error", duration = NULL)

})

})

# Observer for the file upload

observeEvent(input$dataFile, {

# Check if a file was uploaded

if (is.null(input$dataFile)) return()

# Show a notification that data is loading

id <- showNotification("Processing uploaded file...", type = "message", duration = NULL)

# Try to process the uploaded file

tryCatch({

# Read the CSV file

arrests <- read\_csv(input$dataFile$datapath)

# Check if required columns exist

required\_cols <- c("AGE", "GENDER", "ARRESTTIME")

missing\_cols <- required\_cols[!required\_cols %in% names(arrests)]

if (length(missing\_cols) > 0) {

# Remove the loading notification

removeNotification(id)

# Show error notification

showNotification(paste("Missing required columns:", paste(missing\_cols, collapse = ", ")),

type = "error", duration = NULL)

return()

}

# Process the data using similar cleaning logic

arrests\_cleaned <- arrests %>%

mutate(

ARRESTTIME = parse\_date\_time(ARRESTTIME, orders = "Ymd HMS"),

year\_month = as.Date(floor\_date(ARRESTTIME, "month")),

seasonal = case\_when(

month(ARRESTTIME) %in% c(12, 1, 2) ~ "Winter",

month(ARRESTTIME) %in% c(3, 4, 5) ~ "Spring",

month(ARRESTTIME) %in% c(6, 7, 8) ~ "Summer",

TRUE ~ "Fall"

),

year = year(ARRESTTIME),

ages = case\_when(

AGE >= 0 & AGE <= 17 ~ "0-17",

AGE >= 18 & AGE <= 35 ~ "18-35",

AGE >= 36 & AGE <= 50 ~ "36-50",

AGE >= 51 & AGE <= 75 ~ "51-75",

AGE > 75 ~ "75+",

TRUE ~ "Unknown"

)

) %>%

filter(!is.na(AGE) & !is.na(GENDER) & !is.na(ARRESTTIME)) %>%

filter(AGE <= 100, GENDER != "U")

# Store the cleaned data

arrestsData(arrests\_cleaned)

# Set dataLoaded to TRUE

dataLoaded(TRUE)

# Show success notification

removeNotification(id)

showNotification("Data loaded successfully!", type = "message", duration = 3)

}, error = function(e) {

# Remove the loading notification

removeNotification(id)

# Show error notification

showNotification(paste("Error processing file:", e$message), type = "error", duration = NULL)

})

})

# Filtered data based on user inputs

filteredData <- reactive({

# Get the full dataset

data <- arrestsData()

# Return NULL if data is NULL

if (is.null(data)) return(NULL)

# Apply filters

data %>%

filter(year >= input$yearRange[1] & year <= input$yearRange[2]) %>%

filter(ages %in% input$ageGroups) %>%

filter(GENDER %in% input$genderFilter)

})

# Render the plot

output$arrestsPlot <- renderPlot({

# Get filtered data

data <- filteredData()

# Return empty plot if data is NULL or empty

if (is.null(data) || nrow(data) == 0) {

return(ggplot() +

theme\_void() +

annotate("text", x = 0.5, y = 0.5, label = "No data available with current filters"))

}

# Determine facet variable based on showSeasonal input

facet\_var <- if (input$showSeasonal) "seasonal" else "year"

# Create the plotting data

plot\_data <- data

# Ensure facet variable is a factor with ordered levels

if (facet\_var == "year") {

plot\_data$facet\_column <- factor(plot\_data$year, levels = sort(unique(plot\_data$year)))

facet\_title <- "Year"

} else {

plot\_data$facet\_column <- factor(plot\_data$seasonal,

levels = c("Winter", "Spring", "Summer", "Fall"))

facet\_title <- "Season"

}

# Ensure age groups are ordered logically

age\_order <- c("0-17", "18-35", "36-50", "51-75", "75+", "Unknown")

plot\_data$ages <- factor(plot\_data$ages, levels = age\_order)

# Create the plot

ggplot(plot\_data, aes(x = ages, fill = GENDER)) +

geom\_bar(alpha = 0.7) +

facet\_wrap(~ facet\_column) +

scale\_fill\_manual(values = c("M" = "#4169E1", "F" = "#FF69B4")) +

labs(

title = "Count of Arrests for Men and Women by Age Group",

subtitle = if (facet\_var == "year") {

paste0("Years: ", input$yearRange[1], " to ", input$yearRange[2])

} else {

paste0("Seasonal Breakdown (", input$yearRange[1], " to ", input$yearRange[2], ")")

},

x = "Age Group",

y = "Count",

fill = "Gender"

) +

theme\_minimal() +

theme(

plot.title = element\_text(hjust = 0.5, size = 16, face = "bold"),

plot.subtitle = element\_text(hjust = 0.5, size = 14),

axis.title = element\_text(size = 14),

axis.text.x = element\_text(angle = 45, hjust = 1, size = 10),

strip.text = element\_text(size = 12, face = "bold"),

legend.position = "top"

)

})

# Render the data table preview

output$dataTable <- renderDT({

# Get the full dataset

data <- arrestsData()

# Return NULL if data is NULL

if (is.null(data)) return(NULL)

# Select relevant columns and show a preview

data %>%

select(year, seasonal, AGE, ages, GENDER, ARRESTTIME, OFFENSES, INCIDENTNEIGHBORHOOD) %>%

head(1000) %>%

datatable(options = list(

pageLength = 10,

scrollX = TRUE,

scrollY = "500px"

))

})

# Download handler for the plot

output$downloadPlot <- downloadHandler(

filename = function() {

paste("pittsburgh-arrests-", Sys.Date(), ".png", sep = "")

},

content = function(file) {

# Get filtered data

data <- filteredData()

# Handle empty data case

if (is.null(data) || nrow(data) == 0) {

# Create an empty plot with a message

p <- ggplot() +

theme\_void() +

annotate("text", x = 0.5, y = 0.5, label = "No data available with current filters")

ggsave(file, plot = p, width = 10, height = 7, dpi = 300)

return()

}

# Determine facet variable based on showSeasonal input

facet\_var <- if (input$showSeasonal) "seasonal" else "year"

# Create the plotting data

plot\_data <- data

# Ensure facet variable is a factor with ordered levels

if (facet\_var == "year") {

plot\_data$facet\_column <- factor(plot\_data$year, levels = sort(unique(plot\_data$year)))

facet\_title <- "Year"

} else {

plot\_data$facet\_column <- factor(plot\_data$seasonal,

levels = c("Winter", "Spring", "Summer", "Fall"))

facet\_title <- "Season"

}

# Ensure age groups are ordered logically

age\_order <- c("0-17", "18-35", "36-50", "51-75", "75+", "Unknown")

plot\_data$ages <- factor(plot\_data$ages, levels = age\_order)

# Create the plot

p <- ggplot(plot\_data, aes(x = ages, fill = GENDER)) +

geom\_bar(alpha = 0.7) +

facet\_wrap(~ facet\_column) +

scale\_fill\_manual(values = c("M" = "#4169E1", "F" = "#FF69B4")) +

labs(

title = "Count of Arrests for Men and Women by Age Group",

subtitle = if (facet\_var == "year") {

paste0("Years: ", input$yearRange[1], " to ", input$yearRange[2])

} else {

paste0("Seasonal Breakdown (", input$yearRange[1], " to ", input$yearRange[2], ")")

},

x = "Age Group",

y = "Count",

fill = "Gender"

) +

theme\_minimal() +

theme(

plot.title = element\_text(hjust = 0.5, size = 16, face = "bold"),

plot.subtitle = element\_text(hjust = 0.5, size = 14),

axis.title = element\_text(size = 14),

axis.text.x = element\_text(angle = 45, hjust = 1, size = 10),

strip.text = element\_text(size = 12, face = "bold"),

legend.position = "top"

)

# Save the plot

ggsave(file, plot = p, width = 10, height = 7, dpi = 300)

}

)

}

# Run the application

shinyApp(ui = ui, server = server)