Interactive Data Visualization Using R Shiny

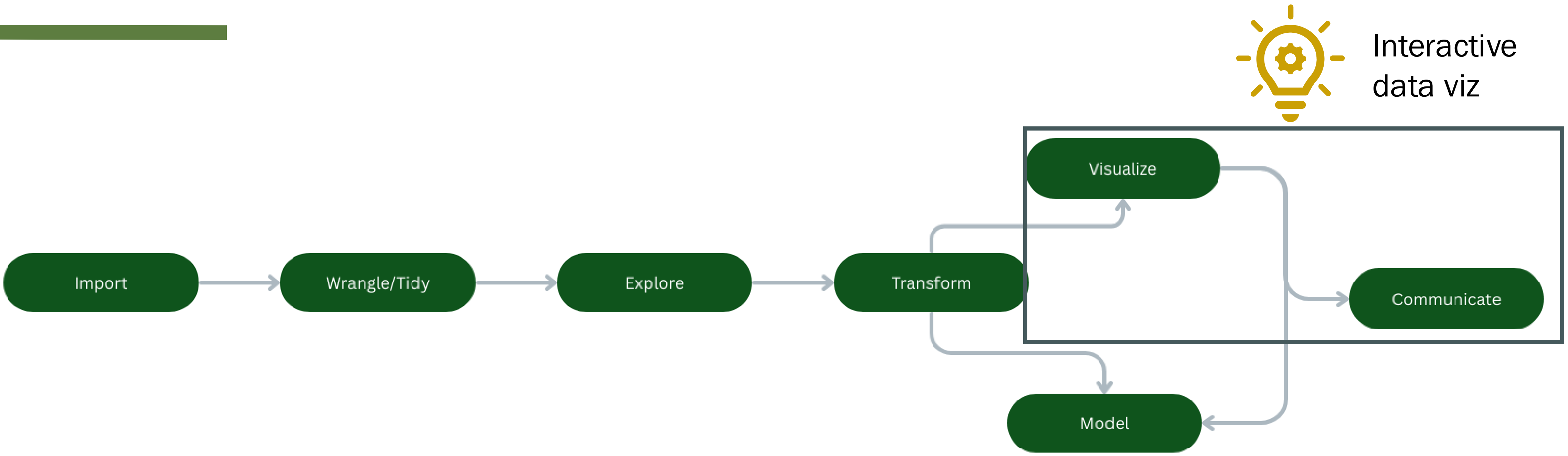
A Case study of ABP Certifying Exam Prediction Tool

Hope Adegoke
Ph.D. Student, University of North Carolina, Greensboro
Graduate Research Assistant, American Board of Pediatrics (ABP)

Agenda

- Data Visualization
- Understanding the structure of a Shiny app
- ABP Cert Prediction Tool Overview
- Final tips & takeaways

The Data Science Process



Why Interactive Data Visualization?

- More engaging and better user experience
- Empower the user
- Storytelling
- Easier to understand and remember
- Straight to the point communication of data

+ a b | e a u

Zoho Analytics

Power BI

Shiny

sisense

Qlik

infogram

Why R Shiny?

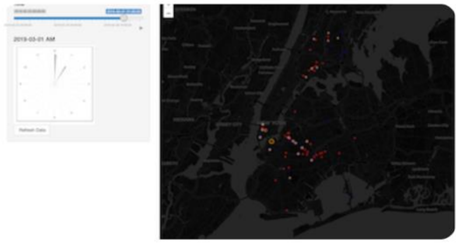
R Shiny is a web application framework for R that helps turn data analyses into interactive web applications.

Benefits

- Open Source and a big community.
- Develop for the web using the same language meant for data exploration and modelling.
- No HTML, CSS, or JavaScript knowledge required**.
- Pre-build widget for building an elegant and powerful application with little effort.

Shiny Examples With Codes

See more here

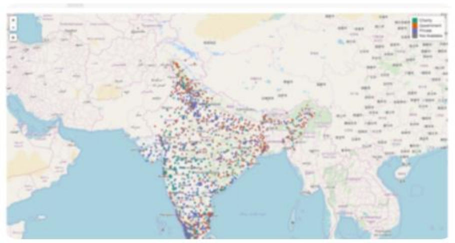

Animated NYC metro traffic

City Cycle Race (with STRAVA) - Compare the cycling speed of cities

Crime Watch

Dublin Transport Info

Freedom of Press Index


Locating Blood Banks in India

https://shiny.posit.co/r/gallery/#feature-demos

Building a Shiny App

```r
install.packages("shiny")
library(shiny)
runApp("app.R")
```

As a minimum, every shiny app has two parts (named ui and server) saved within the same directory. You can either have the ui and server as separate scripts or combine them into one script.

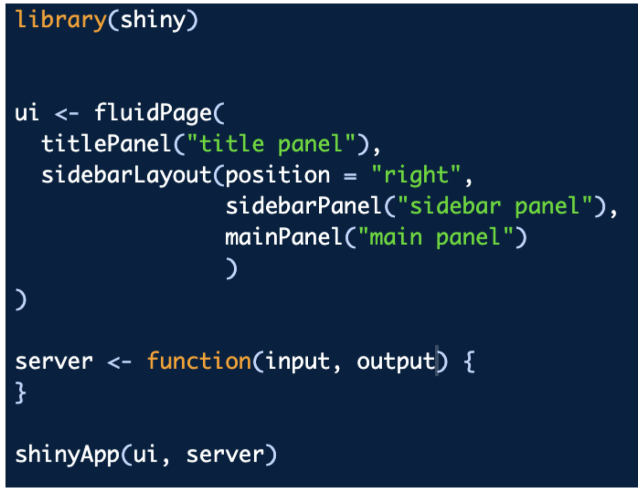
```r
ui <- fluidPage(
)
```

The fluidPage function lets the display adjust automatically to the browser dimensions.

```r
server <- function(input, output, session) {
}
```

The function contains the code needed to run the app

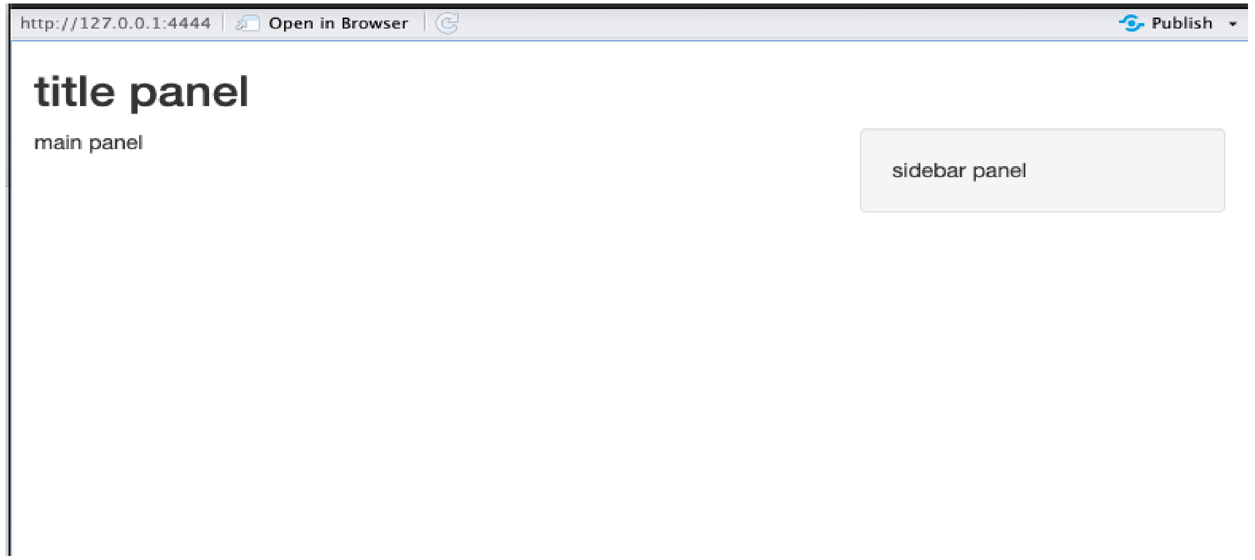
The User Interface (UI)



```r
library(shiny)


ui <- fluidPage(
  titlePanel("title panel"),
  sidebarLayout(position = "right",
                sidebarPanel("sidebar panel"),
                mainPanel("main panel")
                )
)

server <- function(input, output) {
}

shinyApp(ui, server)
```

A series of nested Shiny functions controls the layout of the content.

sidebarLayout is a function with two compulsory arguments, both of which are functions (sidebarPanel and mainPanel). An optional argument controls the position of the panels.



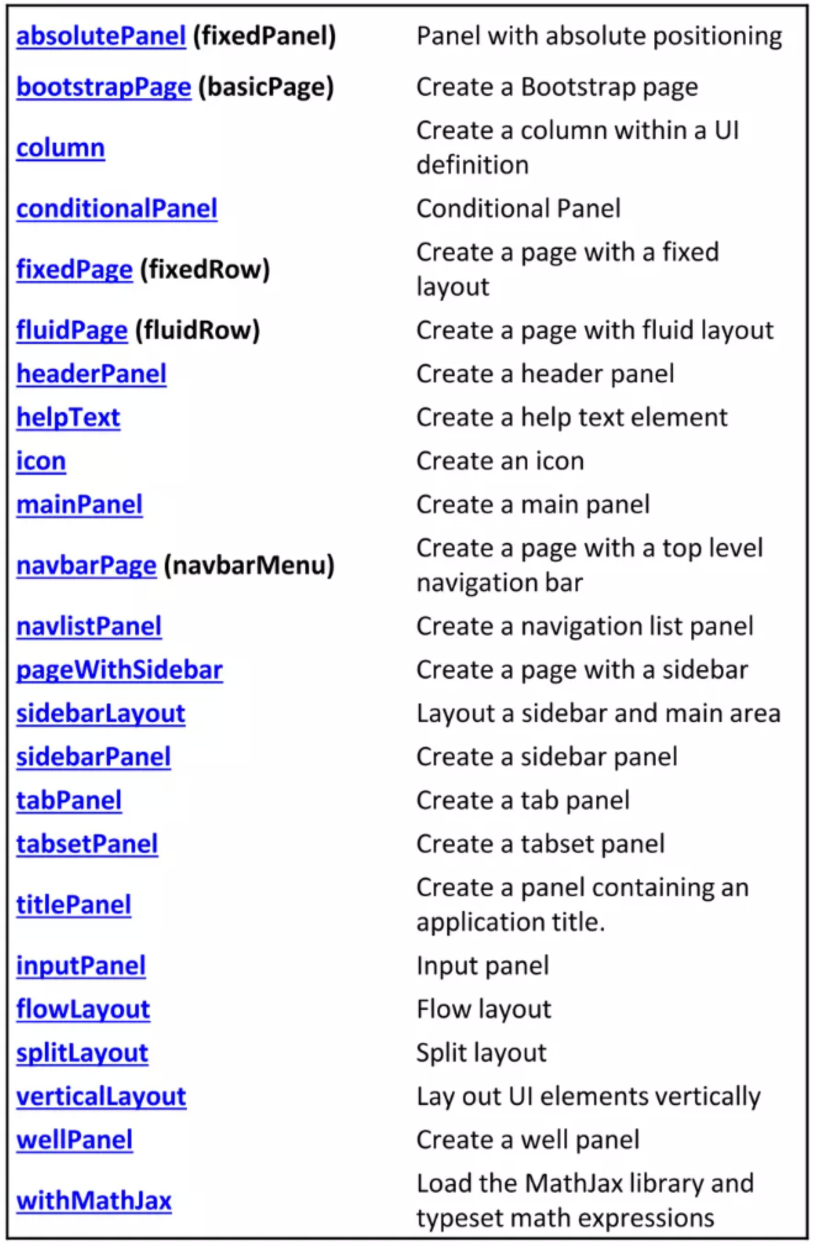
title panel

main panel

sidebar panel

Shiny User Interface Functions

https://shiny.posit.co/r/reference/shiny/1.6.0/

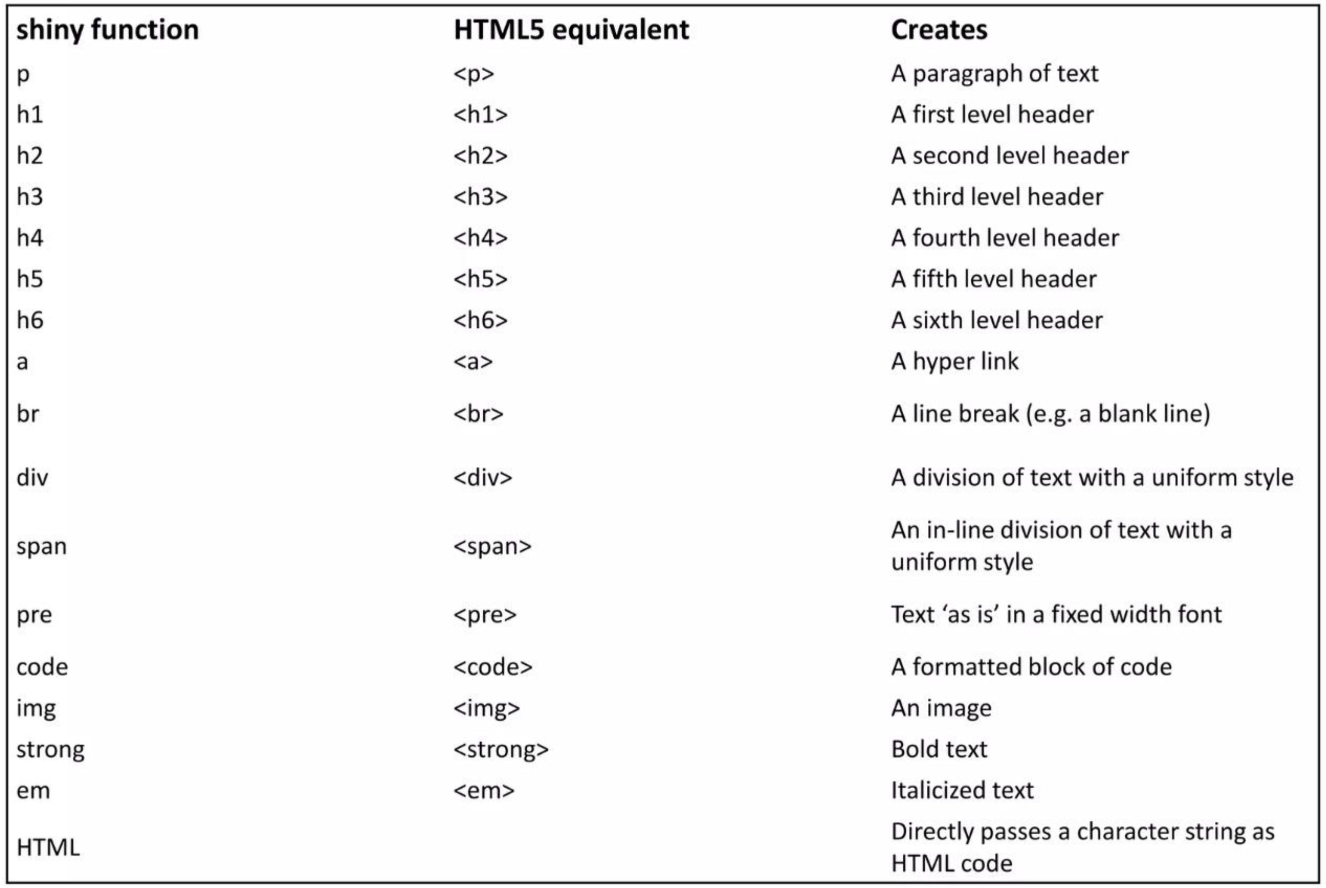


absolutePanel (fixedPanel)	Panel with absolute positioning
bootstrapPage (basicPage)	Create a Bootstrap page
column	Create a column within a UI definition
conditionalPanel	Conditional Panel
fixedPage (fixedRow)	Create a page with a fixed layout
fluidPage (fluidRow)	Create a page with fluid layout
headerPanel	Create a header panel
helpText	Create a help text element
icon	Create an icon
mainPanel	Create a main panel
navbarPage (navbarMenu)	Create a page with a top level navigation bar
navlistPanel	Create a navigation list panel
pageWithSidebar	Create a page with a sidebar
sidebarLayout	Layout a sidebar and main area
sidebarPanel	Create a sidebar panel
tabPanel	Create a tab panel
tabsetPanel	Create a tabset panel
titlePanel	Create a panel containing an application title.
inputPanel	Input panel
flowLayout	Flow layout
splitLayout	Split layout
verticalLayout	Lay out UI elements vertically
wellPanel	Create a well panel
withMathJax	Load the MathJax library and typeset math expressions

Formatting Text

To make things look a bit more interesting, there are a lot of shiny commands that can modify the text color, style, and size.

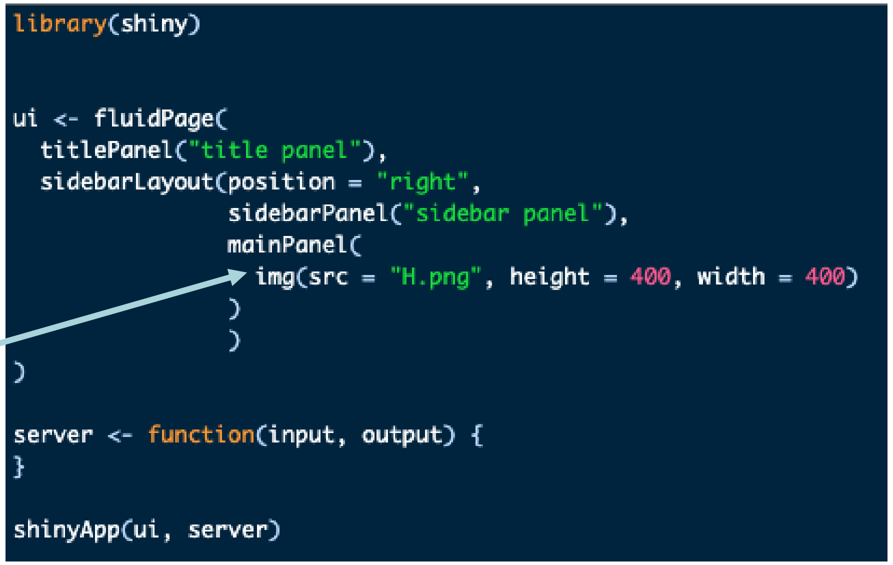
Alternatively, if you are familiar with HTML, you can write HTML code directly inside HTML(“”).



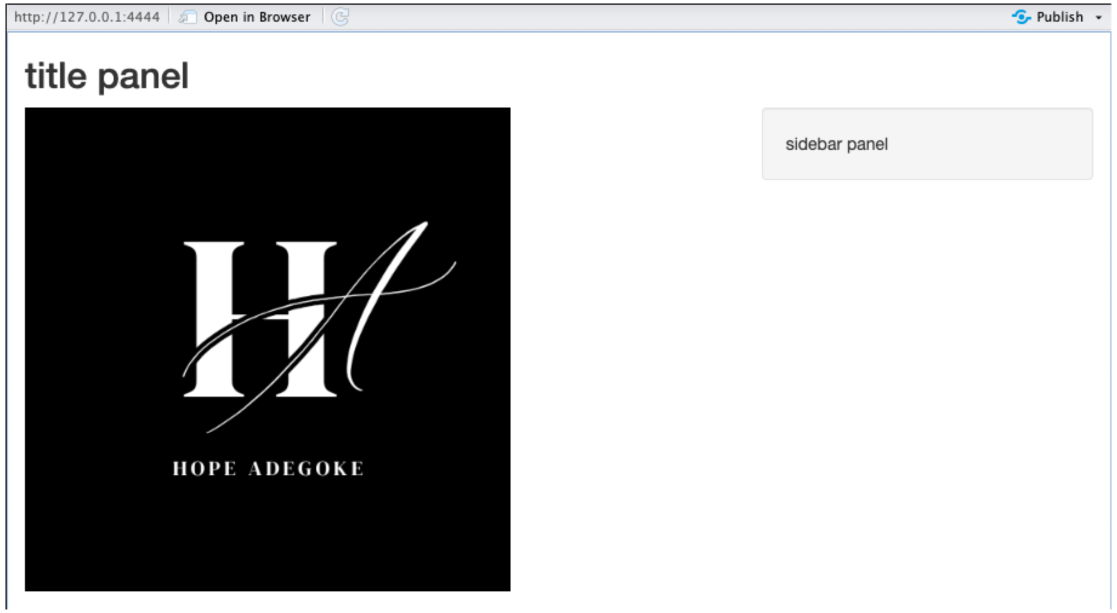
shiny function	HTML5 equivalent	Creates
p	<p>	A paragraph of text
h1	<h1>	A first level header
h2	<h2>	A second level header
h3	<h3>	A third level header
h4	<h4>	A fourth level header
h5	<h5>	A fifth level header
h6	<h6>	A sixth level header
a	<a>	A hyper link
br	<br>	A line break (e.g. a blank line)
div	<div>	A division of text with a uniform style
span	<span>	An in-line division of text with a uniform style
pre	<pre>	Text ‘as is’ in a fixed width font
code	<code>	A formatted block of code
img	<img>	An image
strong	<strong>	Bold text
em	<em>	Italicized text
HTML		Directly passes a character string as HTML code

Adding Images

The image file must be placed inside a folder named www within the same directory as the ui and server scripts. Any file placed in the www folder will be shared with the web browser.



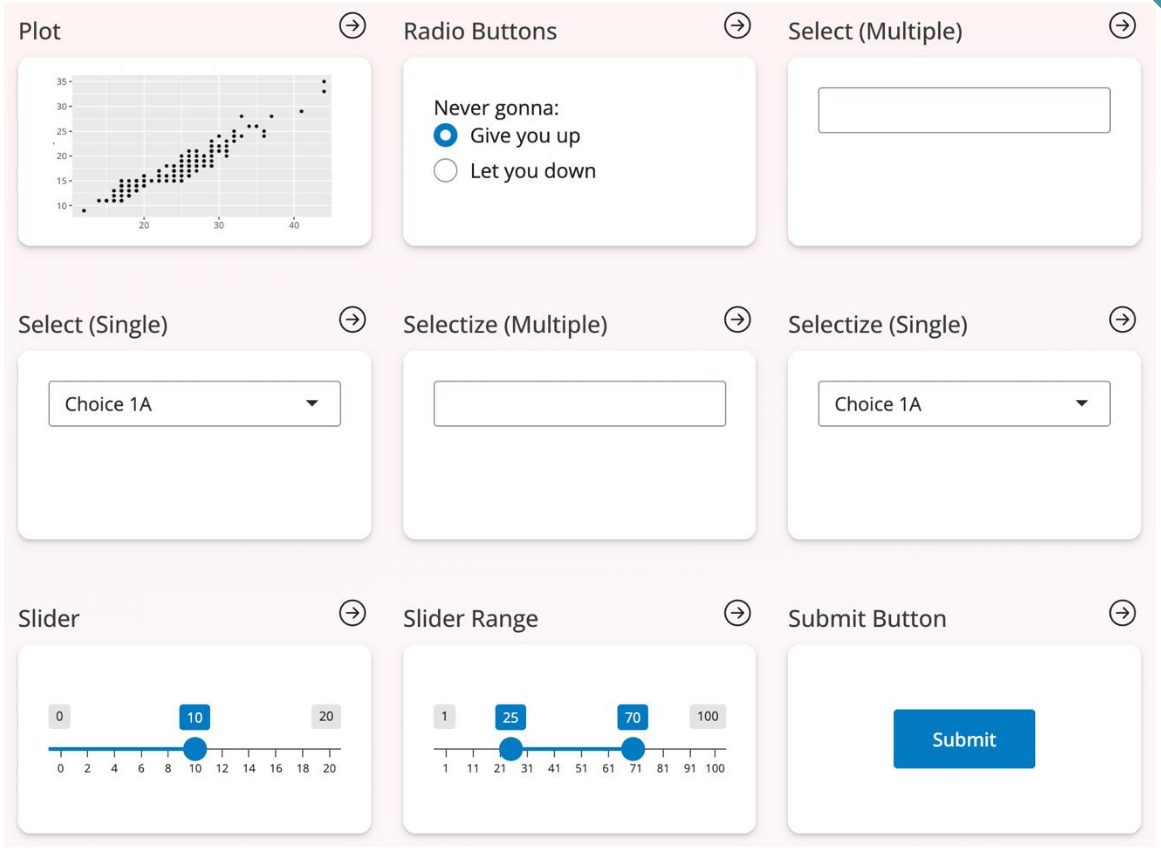
```r
library(shiny)


ui <- fluidPage(
  titlePanel("title panel"),
  sidebarLayout(position = "right",
                sidebarPanel("sidebar panel"),
                mainPanel(
                  img(src = "H.png", height = 400, width = 400)
                )
                )
)

server <- function(input, output) {
}

shinyApp(ui, server)
```



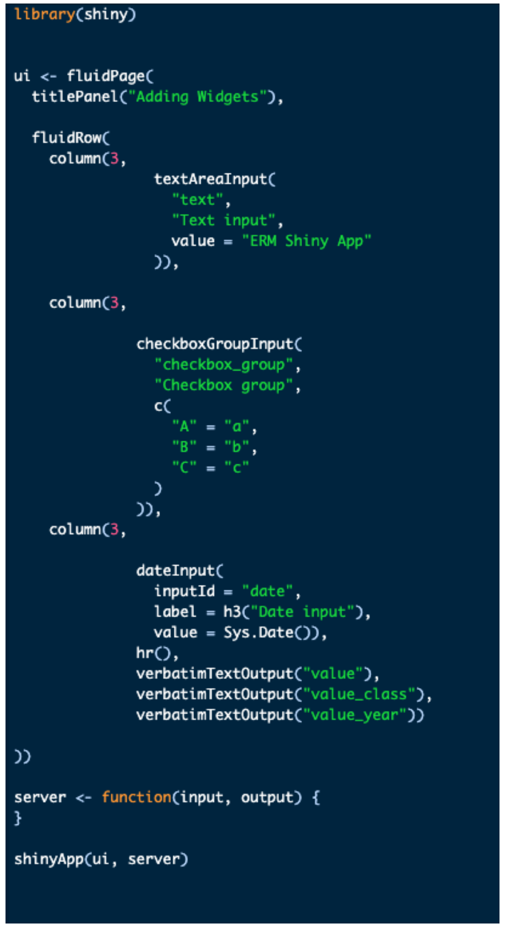
Adding Widgets

Widgets are interactive web elements. A series of inbuilt shiny functions allows widget to be easily added to webpages. Each function has several arguments. It must have at least a name (used to access its value, but not visible), and a label (visible on the page). Both are character strings. Other arguments are widget specific.

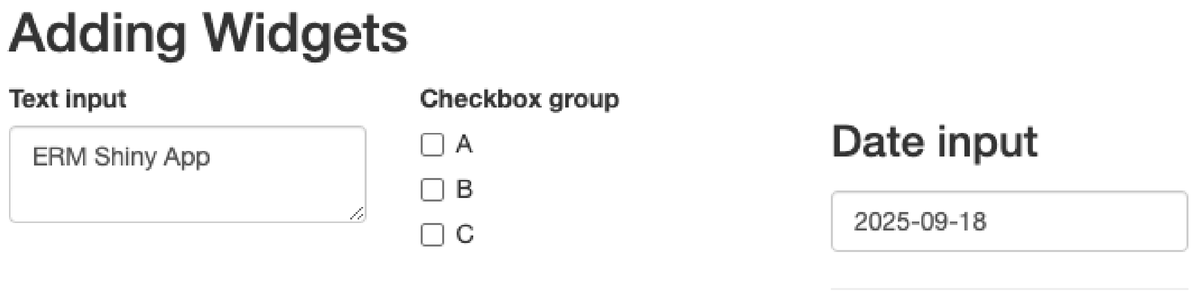
https://shiny.posit.co/r/components/



Adding Widgets



```r
library(shiny)


ui <- fluidPage(
  titlePanel("Adding Widgets"),

  fluidRow(
    column(3,
                textAreaInput(
                  "text",
                  "Text input",
                  value = "ERM Shiny App"
                )),

    column(3,

              checkboxGroupInput(
                "checkbox_group",
                "Checkbox group",
                c(
                  "A" = "a",
                  "B" = "b",
                  "C" = "c"
                )
              )),
    column(3,

              dateInput(
                inputId = "date",
                label = h3("Date input"),
                value = Sys.Date()),
              hr(),
              verbatimTextOutput("value"),
              verbatimTextOutput("value_class"),
              verbatimTextOutput("value_year"))

))

server <- function(input, output) {
}

shinyApp(ui, server)
```



Adding Widgets

Text input

ERM Shiny App

Checkbox group

☐ A
☐ B
☐ C

Date input

2025-09-18

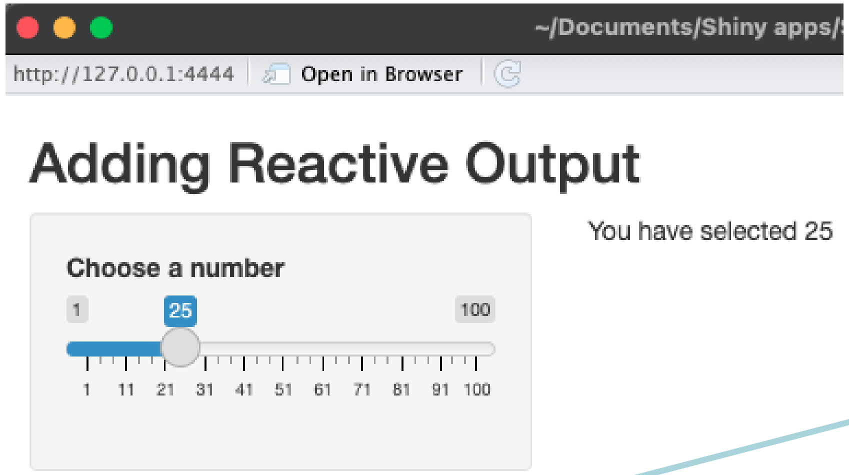
Adding Reactive Output

Widgets allow users to input their choices. Reactive output is a response to those choices. Programming it into an app is a two-step process:

- Add an R object to the user interface to indicate where it should be displayed.
- Add instructions on how to build the object in the server

Output function	Creates
htmlOutput	Raw HTML
imageOutput	Image
plotOutput	Plot
TableOutput	Table
textOutput	Text

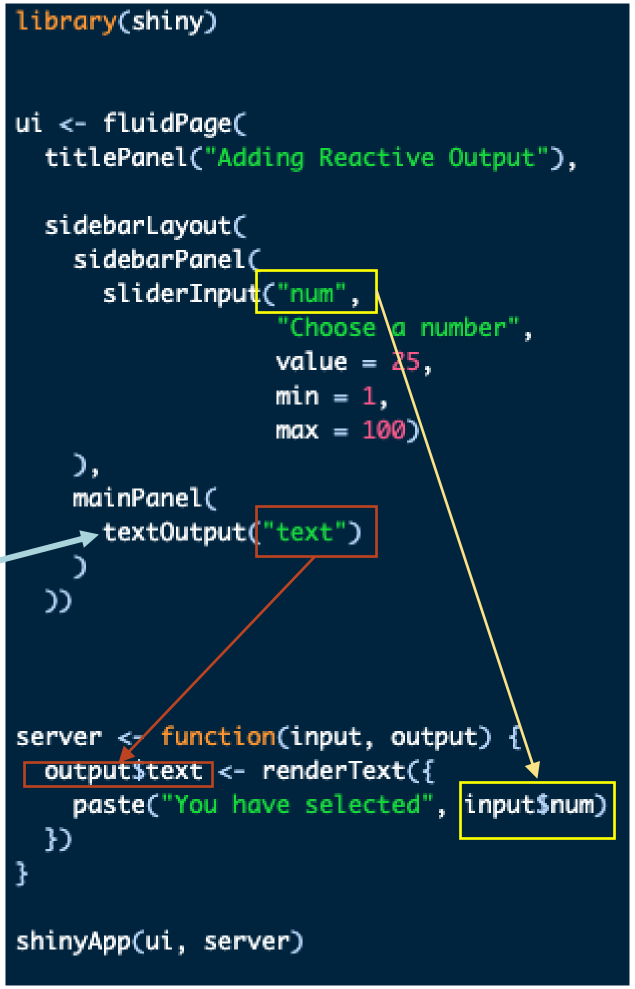
Adding Reactive Output


http://127.0.0.1:4444 Open in Browser

Adding Reactive Output

Choose a number

You have selected 25


```
library(shiny)


ui <- fluidPage(
  titlePanel("Adding Reactive Output"),

  sidebarLayout(
    sidebarPanel(
      sliderInput("num",
                  "Choose a number",
                  value = 25,
                  min = 1,
                  max = 100)
    ),
    mainPanel(
      textOutput("text")
    )
  ))


server <- function(input, output) {
  output$text <- renderText({
    paste("You have selected", input$num)
  })
}

shinyApp(ui, server)
```

Output functions have one argument, which is a name (character string). In this case, “text”. This is used in the server to identify the output.

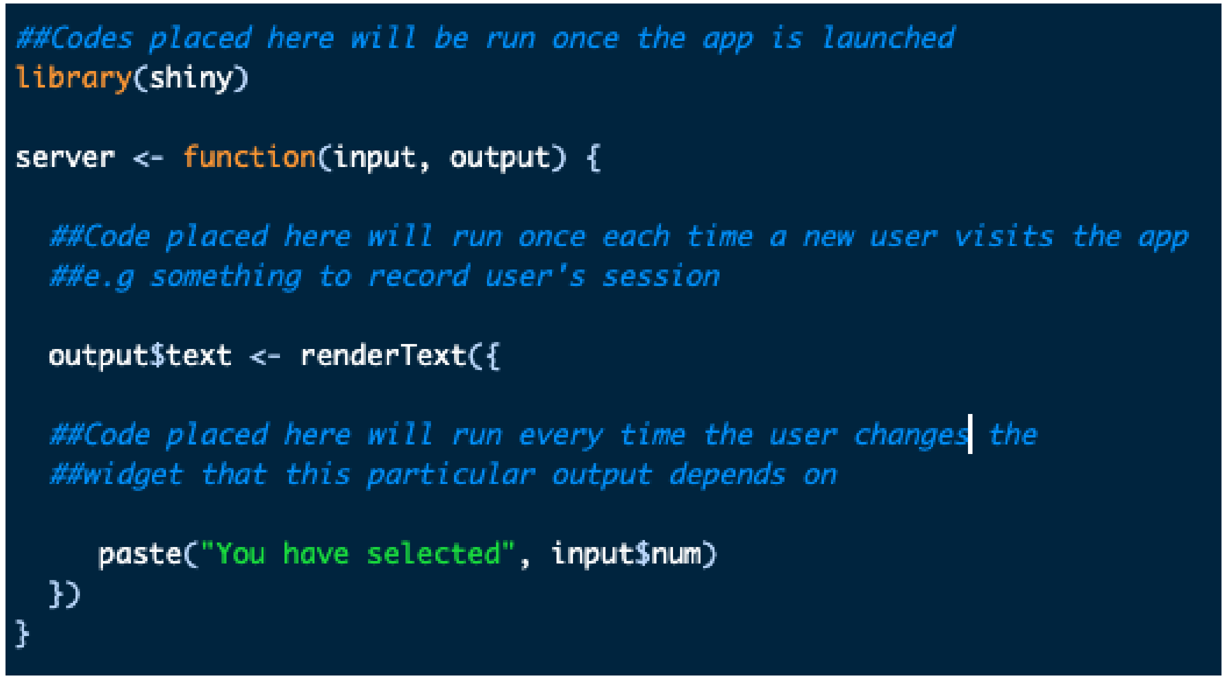
Adding Reactive Output

The function inside the Shiny Server contains all the code that needs to be updated when a user accesses the app. All R output objects used in the ui need to be defined in the server using the prefix output$ followed by the name of the object e.g., output$text

The element should be defined using one of the shiny render* function. This should be chosen to reflect the type of output. Each render function takes a single argument surrounded by braces.

Render function	Creates
renderImage	Images (saved as a link to the source file)
renderPlot	Plots
renderPrint	Any printed output
renderTable	Data frame, matrix, other table like structures
renderText	Character strings
renderUI	A shiny tag object or HTML

The server is where all the code for executing the app is located. How frequently code is run depends on where it is placed within the server script



```
##Codes placed here will be run once the app is launched
library(shiny)

server <- function(input, output) {

  ##Code placed here will run once each time a new user visits the app
  ##e.g something to record user's session

  output$text <- renderText({

  ##Code placed here will run every time the user changes the
  ##widget that this particular output depends on

     paste("You have selected", input$num)
  })
}
```

The Server

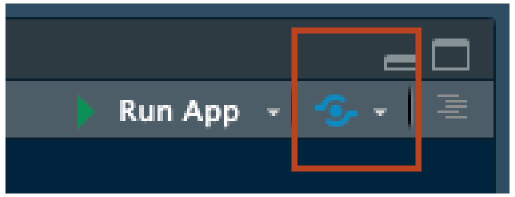
Other Shiny Functions

- **Reactive Expressions** – You decide which part of the app update when
- **Absolute Positioning** – Personalization of the UI using a grid-based system
- **Interactive Visualizations** – Interactive maps and charts
- **Progress Bars** – The User can know that something is happening while loading
- **Integration with R Markdown** – For interactive documents
- **Shiny Server Pro** – Allows user authentication
- **Etc….**

Sample App

Let's go into R and see how everything comes together with a quick, simple example.

Publishing your App

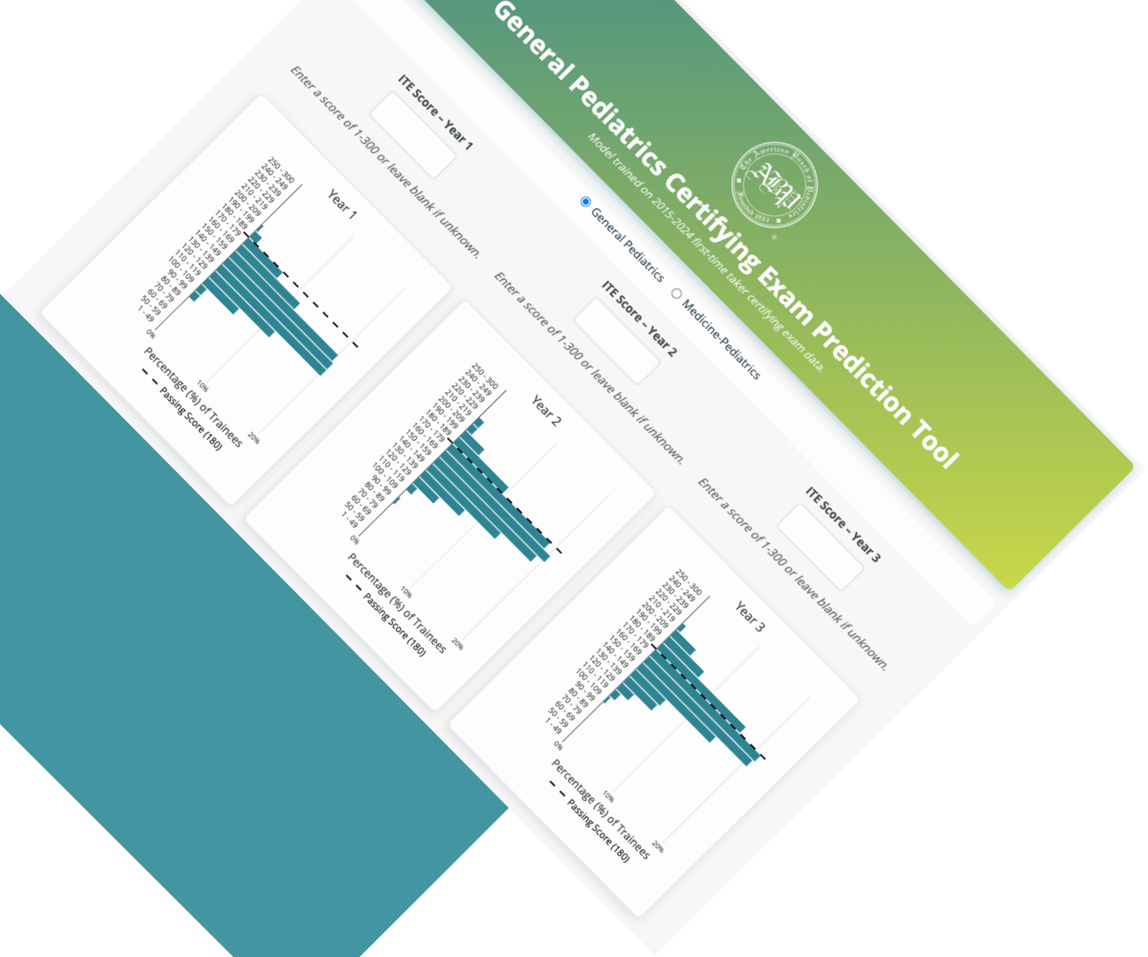


- Create and account in shinyapps.io
- Use the blue button to commence publishing
- Follow all the steps to get your app published

Shinyapps.io

Resources

- https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/
- https://mastering-shiny.org/ - Rshiny Textbook
- AI – Use AI to make your work faster, cleaner, and efficient.


General Pediatrics Certifying Exam Prediction Tool
Model trained on 2015-2024 first-time taker certifying exam data.
General Pediatrics
Medicine-Pediatrics
ITE Score – Year 1
Enter a score of 1-300 or leave blank if unknown.
ITE Score – Year 2
Enter a score of 1-300 or leave blank if unknown.
ITE Score – Year 3
Enter a score of 1-300 or leave blank if unknown.
Year 1
Year 2
Year 3
Percentage (%) of Trainees
Passing Score (180)

Case study of ABP Certifying Exam Prediction Tool

https://www.abp.org/dashboards/certifying-exam-prediction-tool

Back Story

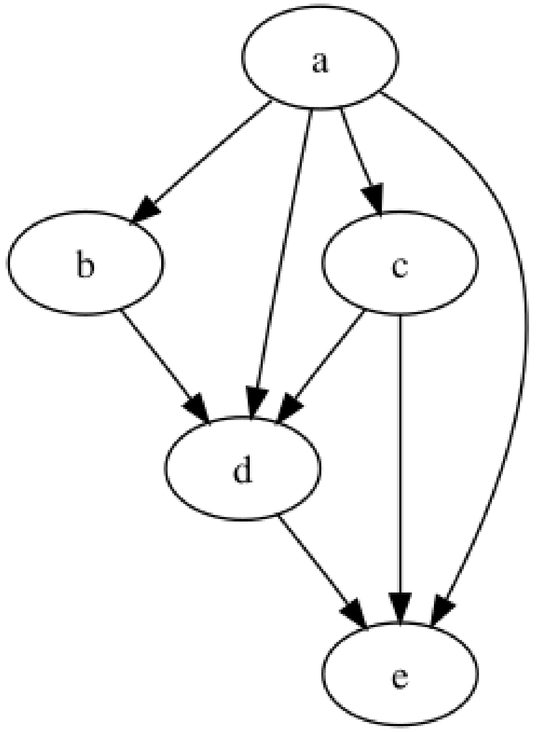
ITE uses retired certifying exam questions and ensures they match the content specifications.

ITE scores have moderate to high correlations with scores on the certifying examination.

Training programs use ITE to identify trainees who need additional support and remediation.

ITE is a valid feedback tool to indicate overall preparedness for the certifying examinations.

Bayesian Network



Directed Acyclic Graph

A probabilistic graphical model that represents a set of variables and their conditional dependencies through a directed acyclic graph (DAG)

Bayesian Networks can be learned from data or defined based on expert knowledge.

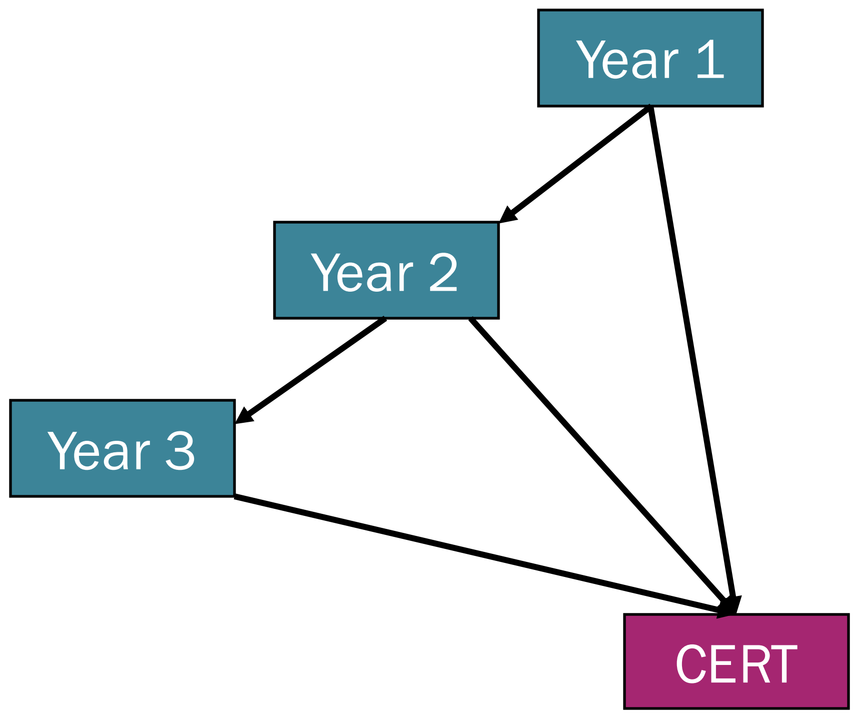
No strict distributional assumption, uses conditional probability distributions.

Highly flexible; can model complex dependencies and update beliefs dynamically

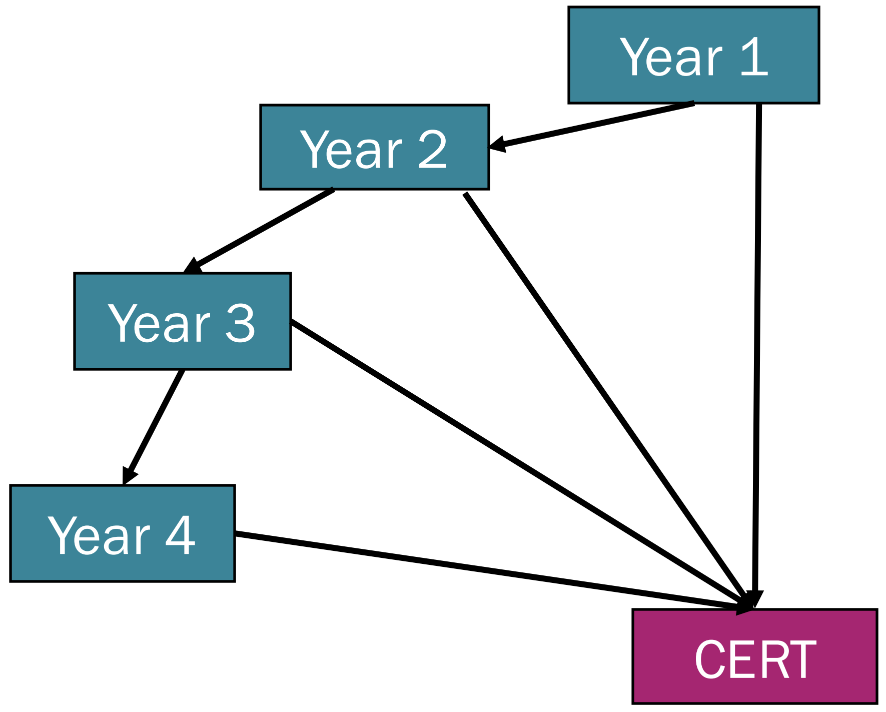
Useful for small-sample exams

Provides accurate, data-driven score forecasts

GPITE Score Prediction Model

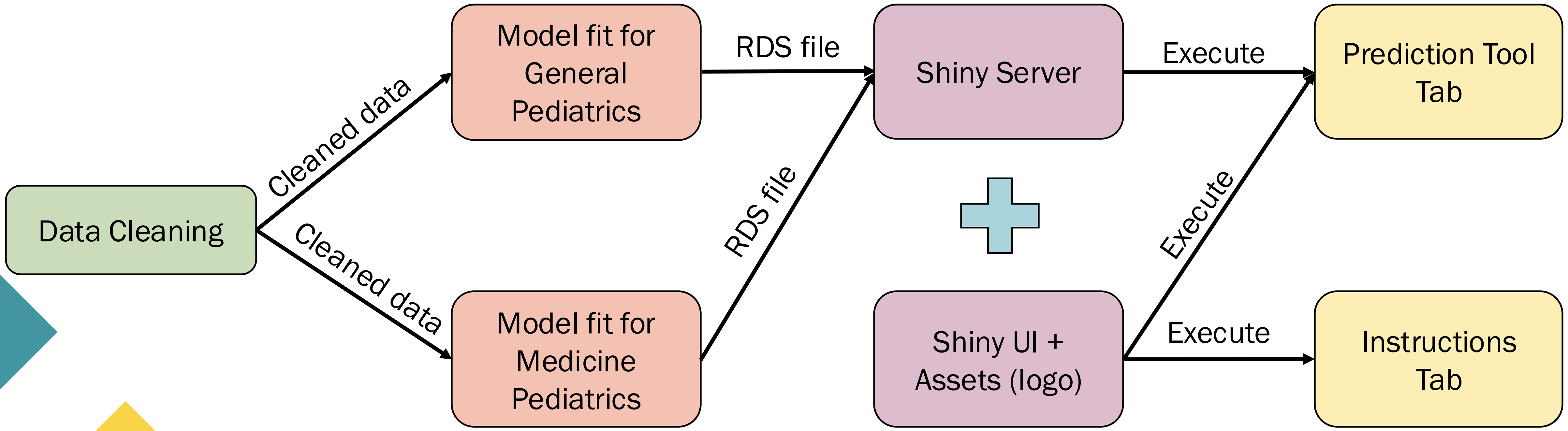


General Pediatrics Model



Medicine-Pediatrics Model

ABP App Build Workflow



Demo: https://www.abp.org/dashboards/certifying-exam-prediction-tool

Key Takeaways & Final Tips

- Build a minimal working app before adding complexity.
- Use reactive expressions wisely: Prevent redundant computations and speed up the app.
- Break large apps into modules or separate scripts.
- Leverage packages.
- Modify existing apps from Posit’s gallery that fit your use case. https://shiny.posit.co/r/gallery/#feature-demos

https://github.com/Hope112/R-Shiny-Dashboard-Presentation

Thank you

Hope Adegoke, M.Sc., Ph.D. Student

Email: hoadegoke@uncg.edu

Work email: hadegoke@abpeds.org

Website: www.hopeadegoke.com

HA's Presentation Feedback Form