# Lesson 14: Advanced Shiny III

Dr. Kam Tin Seong
Assoc. Professor of Information Systems

School of Computing and Information Systems,
Singapore Management University

02 August 2021

# Overview

In this lesson, selected advanced methods of Shiny will be discussed. You will also gain hands-on experiences on using these advanced methods to build Shiny applications.

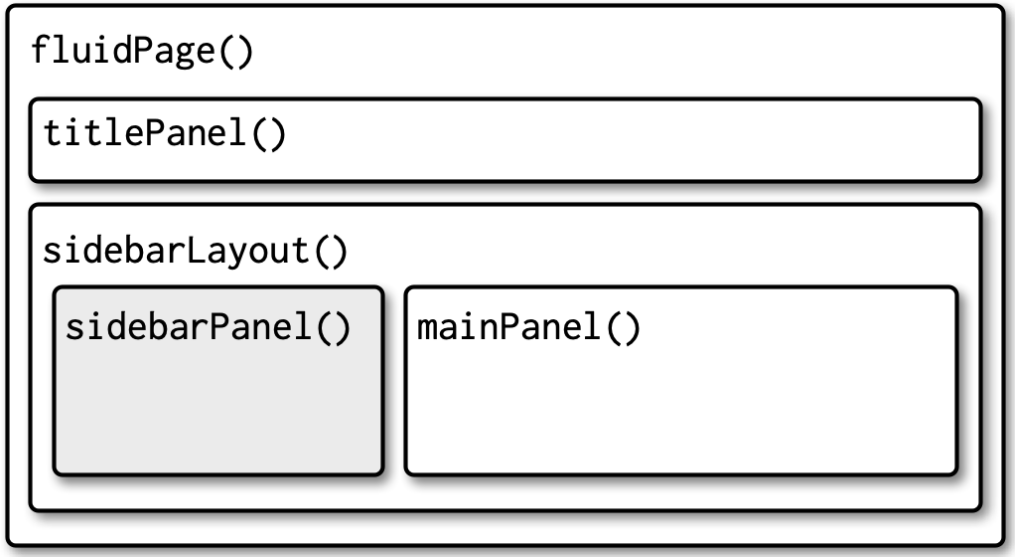
By the end of this lesson, you will be able to:

- understand basic components of Shiny layout and to customise Shiny layouts,
- understand how Shiny themes work and how to customise Shiny theme,
- create professionally looking UI by using shinydashboard package, and
- deploy Shiny App on shinyapps.io service.

# R Shiny Layout

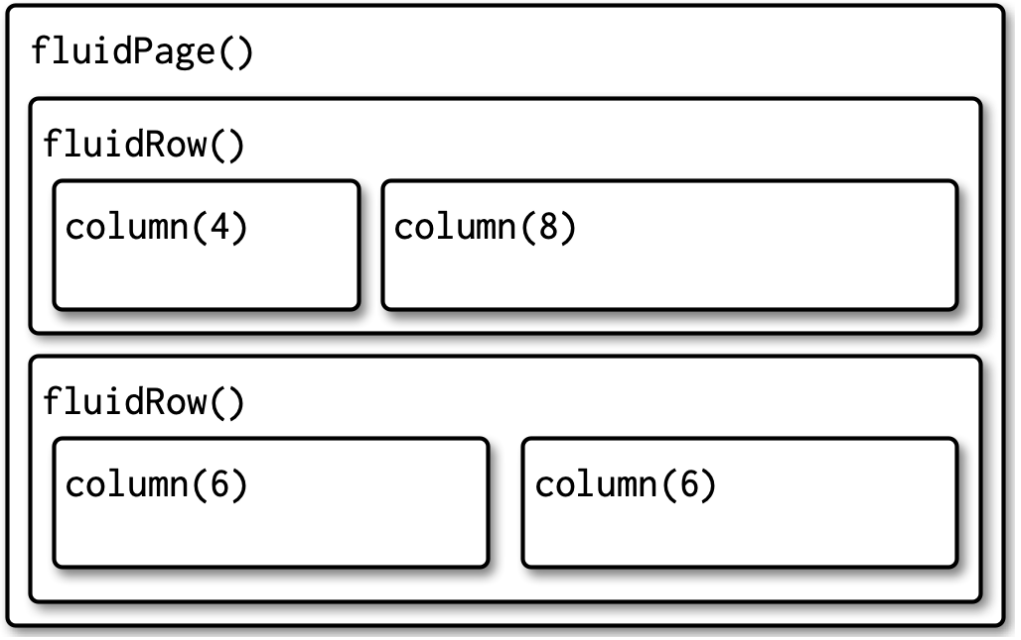
Structure of a basic R Shiny App with sidebar layout.

```
ui <- fluidPage(
    titlePanel("R Shiny Basic Layout"),
    sidebarLayout(
        sidebarPanel("Side bar"
        ),
        mainPanel("Main Display"
        )
    )
)
```



# Multi-row layout: *fluidPage()* and *fluidRow()* method

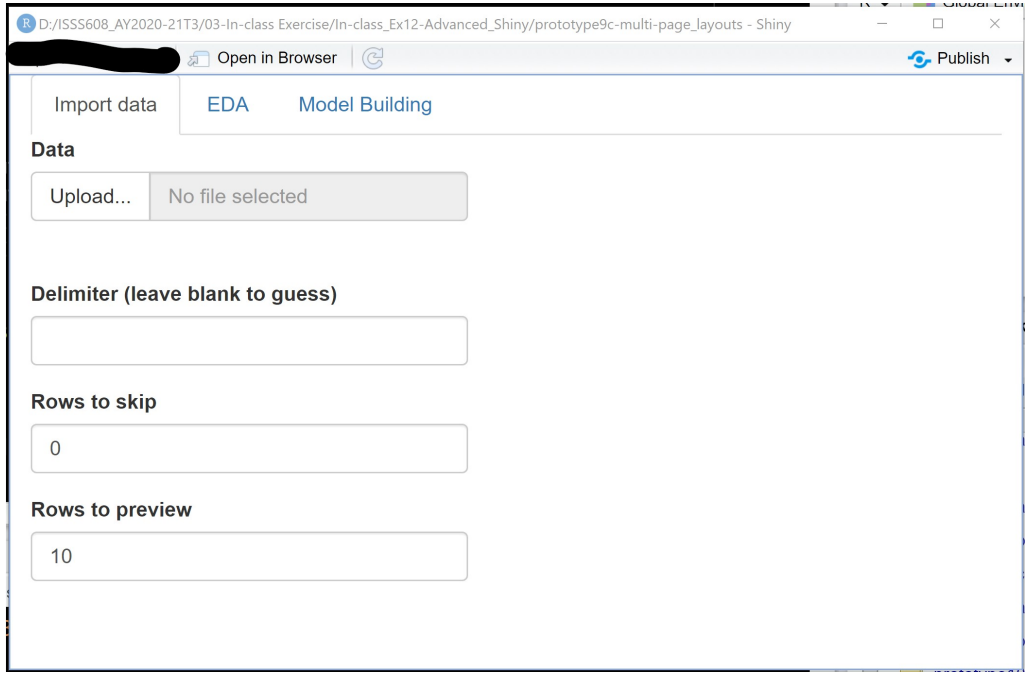
```
fluidPage(
  fluidRow(
    column(4,
      ...
    ),
    column(8,
      ...
    )
  ),
  fluidRow(
    column(6,
      ...
    ),
    column(6,
      ...
    )
  )
)
```



# Multi-page layout: *tabsetPanel()* and *tabPanel()* method

The code chunk below uses *tabsetPanel()* and its close friend *tabPanel()* to create a multi-page layout

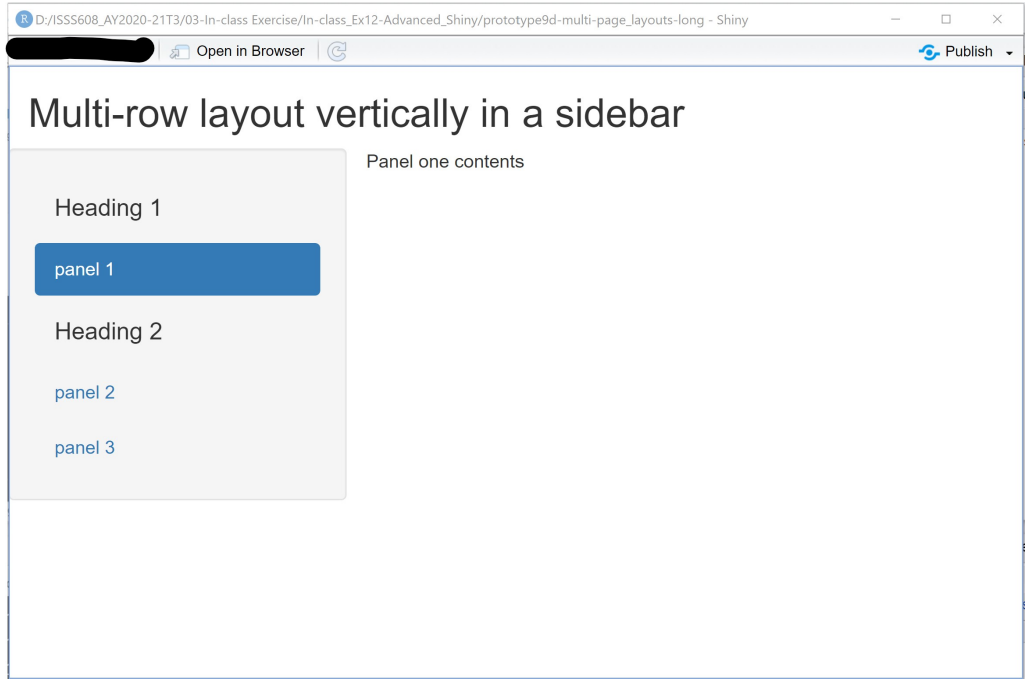
```
ui <- fluidPage(
  tabsetPanel(
    tabPanel("Import data",
             fileInput("file", "Data",
                       buttonLabel = "Upload..
             textInput("delim",
                       "Delimiter (leave blank
                       ""),
             numericInput("skip", "Rows to ski
                          0, min = 0),
             numericInput("rows",
                          "Rows to preview",
                          10, min = 1)
             ),
    tabPanel("EDA"),
    tabPanel("Model Building")
    )
)
```



# Multi-page layout: *navlistPanel()* and *tabPanel()* method

The code chunk below uses *navlistPanel()* and *tabPanel()* to create an alternative layout that let you use more tabs with longer titles.

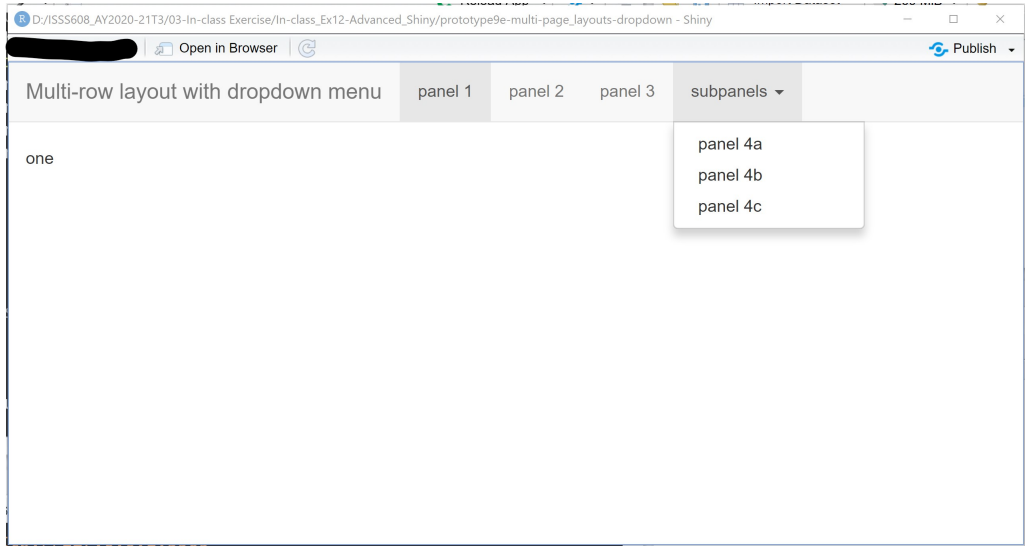
```
ui <- fluidPage(
    titlePanel("Multi-row layout vertically in
    navlistPanel(
        id = "tabset",
        "Heading 1",
        tabPanel("panel 1", "Panel one contents
        "Heading 2",
        tabPanel("panel 2", "Panel two contents
        tabPanel("panel 3", "Panel three conter
    )
)
```



# Multi-page layout - *navbarPage()* and *navbarMenu()* method

In the code chunk below, *navbarPage()* is still runs the tab titles horizontally, but *navbarMenu()* is used to add drop-down menus for an additional level of hierarchy.

```
ui <- navbarPage(
    "Multi-row layout with drop-down menus",
    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")
    )
)
```



# R Shiny Theme

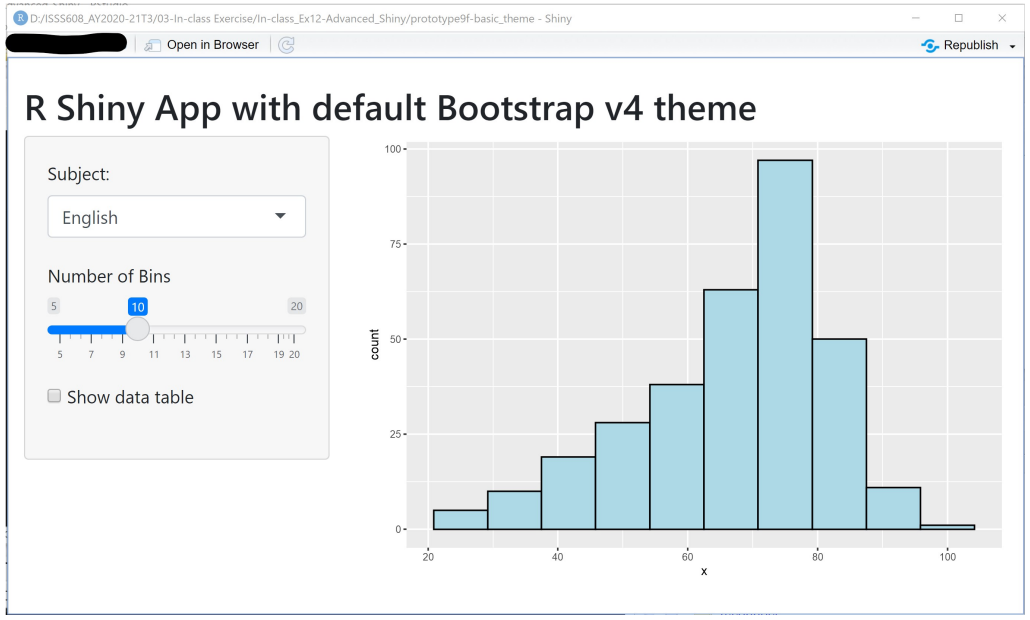
By default, Shiny theme is based on **Bootstrap** framework, a collection of HTML conventions, CSS styles, and JS snippets bundled up into a convenient form.

- The visual appearance of Bootstrap can be customised by using *bslib::bs_theme()*.
- The layouts, inputs, and outputs of Bootstrap names can be customised by using the *class* argument.

# R Shiny Theme - *bslib::bs_theme()* method

- By default, *bslib::bs_theme()*, will use Bootstrap v4.

```
ui <- fluidPage(
    theme = bslib::bs_theme(),
    titlePanel("R Shiny App"),
    sidebarLayout(
```



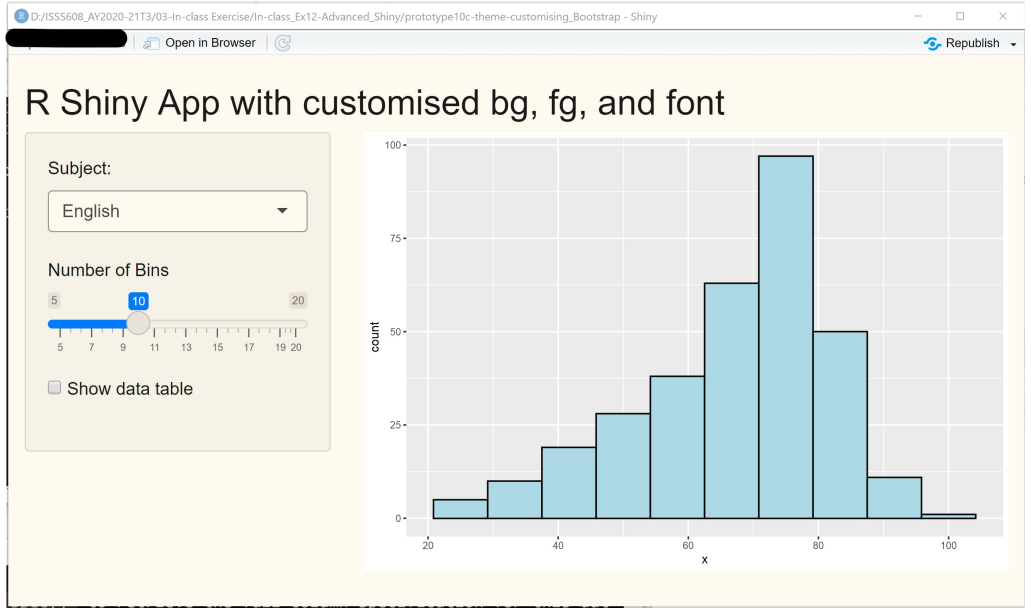

- The easiest way to change the overall look of an Shiny app is to pick a premade **"bootswatch"** theme using the bootswatch argument to *bslib::bs_theme()*.

```
ui <- fluidPage(
    theme = bslib::bs_theme(
      bootswatch = "flatly"),
    titlePanel("R Shiny App"),
    sidebarLayout(
```

# R Shiny Theme: Customising *bs_theme()* argument method

Alternatively, we can construct our own theme using the other arguments to *bs_theme()* like *bg* (background colour), *fg* (foreground colour) and *base_font*:

```
ui <- fluidPage(
  theme = bslib::bs_theme(bg = "#FFFAF0",
                          fg = "black",
                          base_font = "Arial")
  titlePanel("R Shiny App"),
  sidebarLayout(
```




- For colour code, refer to this link.
- For font name, refer to this link

# R Shiny Theme: Customising with thematic package

- **thematic** package auto theming gives R plots the ability to style themselves inside Shiny (via CSS).
- Call *thematic_shiny()* before launching a Shiny app to enable thematic for every *plotOutput()* inside the app.

```
server <- function(input, output){
    output$distPlot <- renderPlot({
        thematic::thematic_shiny()
        x <- unlist(exam[,input$variable])
```

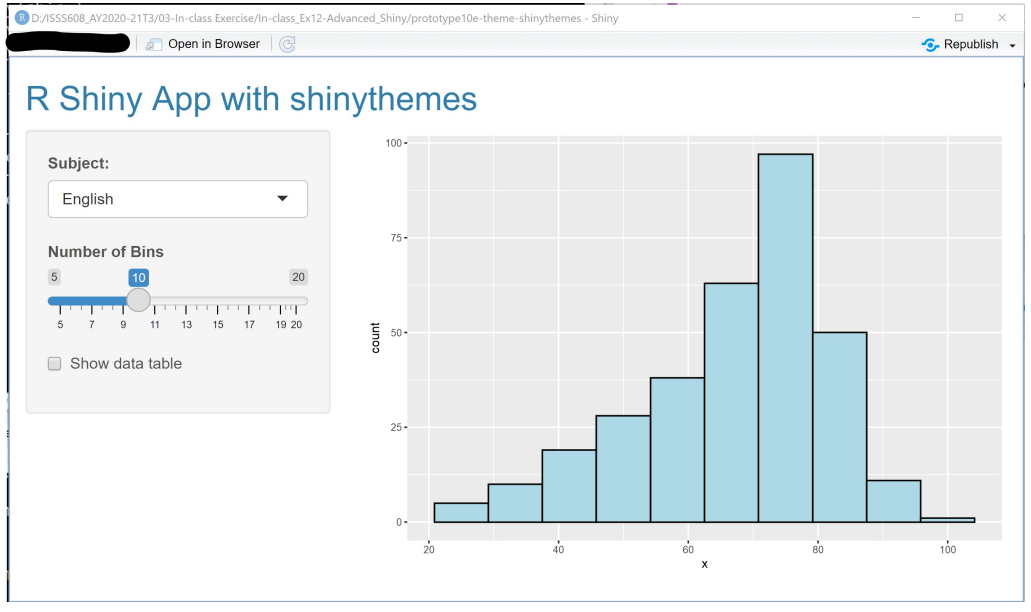

Note that:

- If no values are provided to *thematic_shiny()*, each *plotOutput()* uses the app's CSS colors to inform new R plotting defaults.
- If the app uses *Google Fonts* (and you have **showtext** and/or **ragg** installed), you may safely provide *font = "auto"* to *thematic_shiny()*, which also translates CSS fonts to R.

# R Shiny Theme: shinythemes package

- **shinythemes** package includes several *Bootstrap themes* which are packaged for use with Shiny applications.

```
library(shiny)
library(tidyverse)
library(shinythemes)

exam <- read_csv("data/Exam_data.csv")

ui <- fluidPage(
    theme = shinytheme("cerulean"),
    titlePanel("R Shiny App with shinythemes")
```

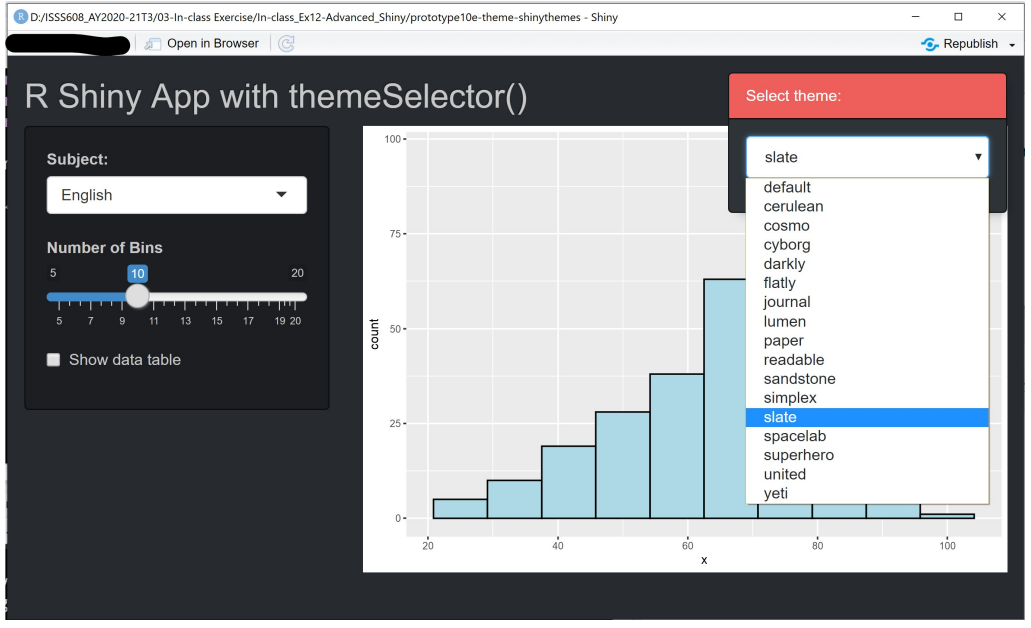



- For detail themes and getting started, refer to the online document.

# R Shiny Theme: *themeSelector()* of shinythemes package

If you want to quickly test out different themes with an application, you can simply add *themeSelector()* somewhere to the UI. This will add a select box which lets you choose the theme. It will change the theme without having to reload or restart your app.

```
ui <- fluidPage(
    shinythemes::themeSelector(),
    titlePanel("R Shiny App with shinythemes")
```

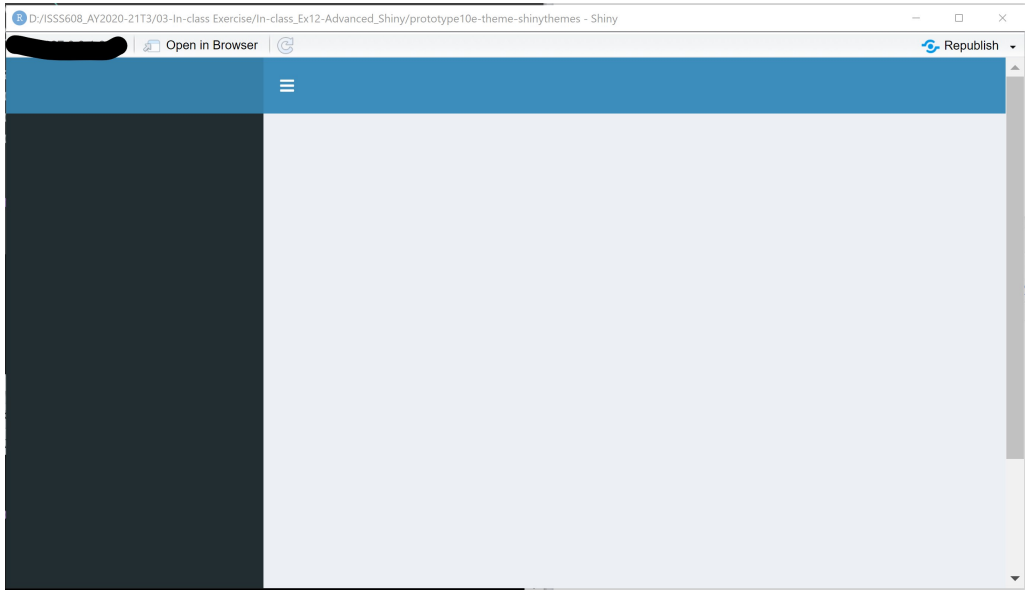



Note: *themeSelector()* is only meant to be used while developing an application. Once you've decided on which theme to use, pass it to the theme argument as described in previous slide.

# shinydashboard

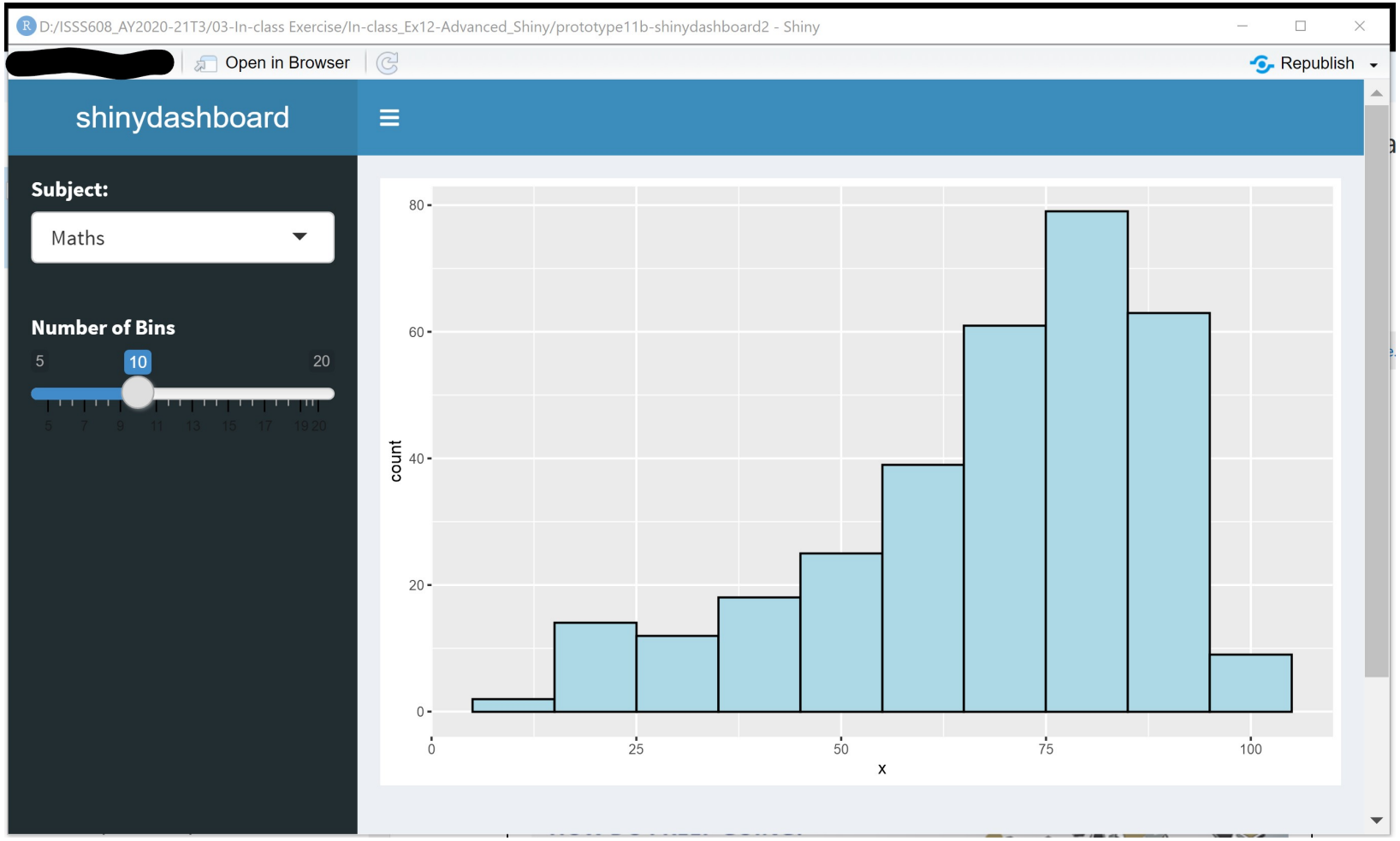
This package provides a theme on top of 'Shiny', making it easy to create attractive dashboards.

```
library(shiny)
library(tidyverse)
library(shinydashboard)

ui <- dashboardPage(
    dashboardHeader(),
    dashboardSidebar(),
    dashboardBody()
)

server <- function(input, output) { }

shinyApp(ui, server)
```



# shinydashboard - Makeover prototype2

In this hands-on exercise, we will makeover prototype2 by using shinydashboard package as shown below.



# shinydashboard - Makeover prototype2

```
library(shiny)
library(shinydashboard)
library(tidyverse)

exam <- read_csv("data/Exam_data.csv")

ui <- dashboardPage(
    dashboardHeader(title = "shinydashboard example"),
    dashboardSidebar(
        selectInput(inputId = "variable",
                    label = "Subject:",
                    choices = list("English" = "ENGLISH",
                                   "Maths" = "MATHS",
                                    "Science" = "SCIENCE"),
                        selected = "ENGLISH"),
        sliderInput(inputId = "bin",
                    label = "Number of Bins",
                    min = 5,
                    max = 20,
                    value = c(10))
        ),
        dashboardBody(
            plotOutput("distPlot")
            )
    )
```

# R Shiny UI Skins

- shinydashboardplus, relies on the same basis as **shinydashboard**, that is the AdminLTE HTML template. It provides extra elements that will help you to develop Shiny apps with a more professional look and feel. Refer to this article for an introduction. For more details, read the vignettes.
- shiny.semantic adds support for a powerful UI library Semantic UI. It also supports universal UI input binding that works with various DOM elements.
- shinyMobile builds on top of framework 7, and is specifically designed for mobile apps. To learn more, start with this article before reading the rest of the vignetts.
- shinymaterial is built on top of Google's Material design framework.

# Deploying Shiny apps to the web

Three ways to deploy Shiny Apps.

- Deploy to the cloud: Shinyapps.io. It is easy to use, secure, and scalable. No hardware, installation, or annual purchase contract required. Free and paid options available.
- Deploy on-premises (open source): Shiny Server. Deploy your Shiny apps and interactive documents on-premises with open source Shiny Server, which offers features such as multiple apps on a single server and deployment of apps behind firewalls.
- Deploy on-premises (commercial): RStudio Connect. With RStudio Connect, you can share Shiny applications, R Markdown reports, dashboards and plots, as well as Python-based content, including Flask, Dash, Streamlit and Bokeh, in one convenient place with push-button publishing from the RStudio IDE. Features include scheduled execution of reports and flexible security policies to bring the power of data science to your entire enterprise.

# Deploying Shiny apps to shinyapps.io

**Step 1:**

Before you get started with shinyapps.io, you will need:

- to install **rsconnect** R package from CRAN, or the latest version from GitHub.

```
install.packages('rsconnect')
```

- After the rsconnect package has been installed, load it into your R session:

```
library(rconnect)
```

# Deploying Shiny apps to shinyapps.io

**Step 2: Create a shinyapps.io account**

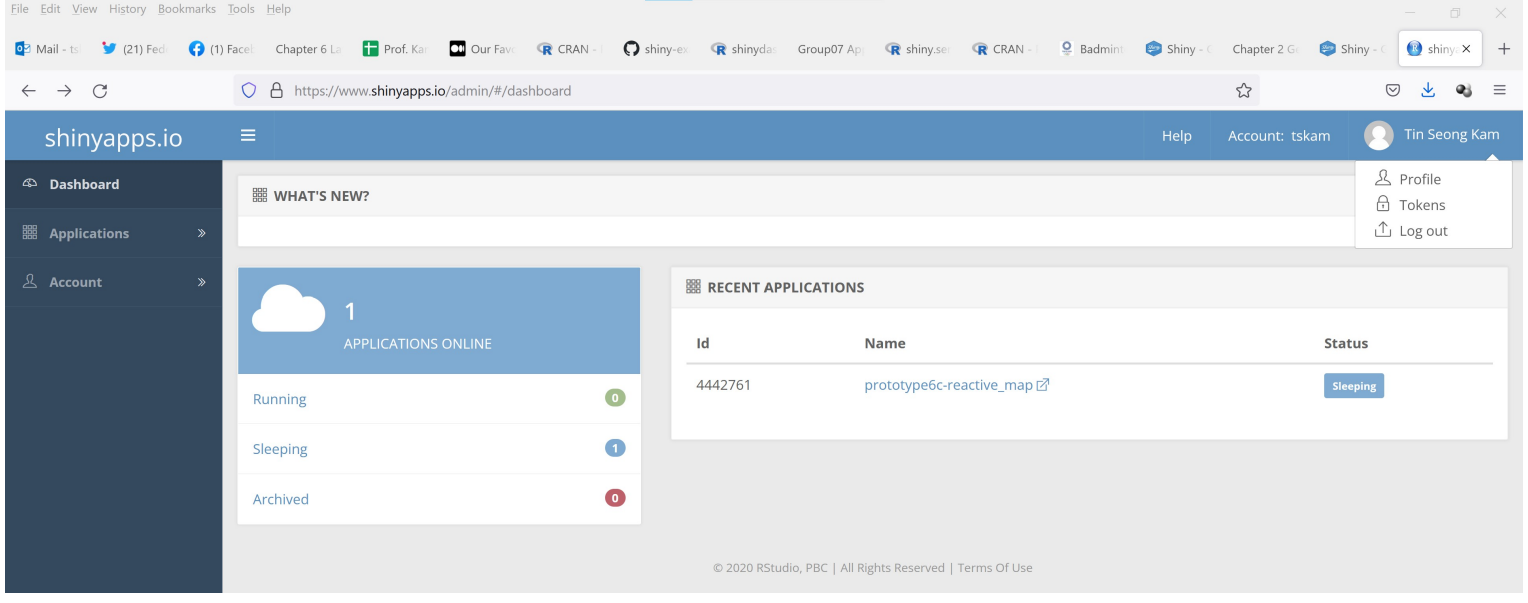
- Go to shinyapps.io and click “Dashboard.” The site will ask you to sign in using your email and password, your Google account, or your GitHub account.
- The first time you sign in, shinyapps.io prompts you to set up your account.
    - Shinyapps.io uses the account name as the domain name for all your apps. Account names must be between four and 63 characters and can contain only letters, numbers, and dashes (-).
    - Account names may not begin with a number or a dash, and they can not end with a dash (see RFC 952). Some account names may be reserved.

# Deploying Shiny apps to shinyapps.io

**Step 3: Retrieve token**

Once you set up your account in shinyapps.io, you can configure the rsconnect package to use your account.

- Shinyapps.io automatically generates a token and secret for you, which the rsconnect package can use to access your account.
- Retrieve your token from the shinyapps.io dashboard. Tokens are listed under Tokens in the menu at the top right of the shinyapps dashboard (under your avatar).

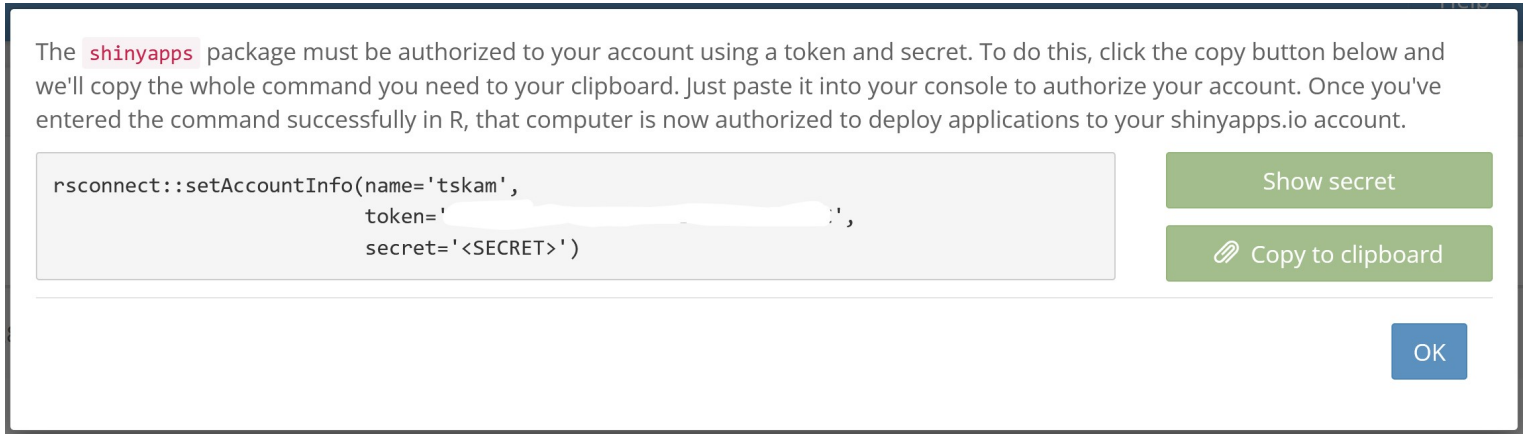


# Deploying Shiny apps to shinyapps.io

**Step 4: Configure rsconnect**

Next, you will configure rconnect:

- Click the show button on the token page. A window will pop up that shows the full command to configure your account using the appropriate parameters for the rsconnect::setAccountInfo function.

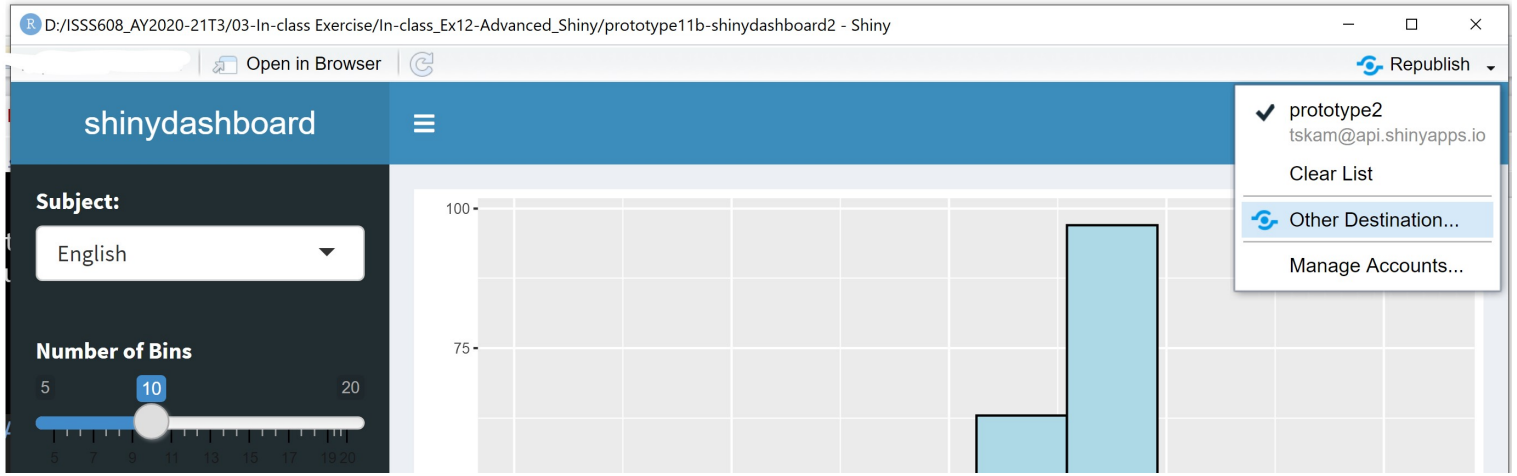



- Copy this command to your clip board, and then paste it into console window of RStudio and click **enter**.

# Deploying Shiny apps to shinyapps.io

**Step 5: Deploying the Shiny App**

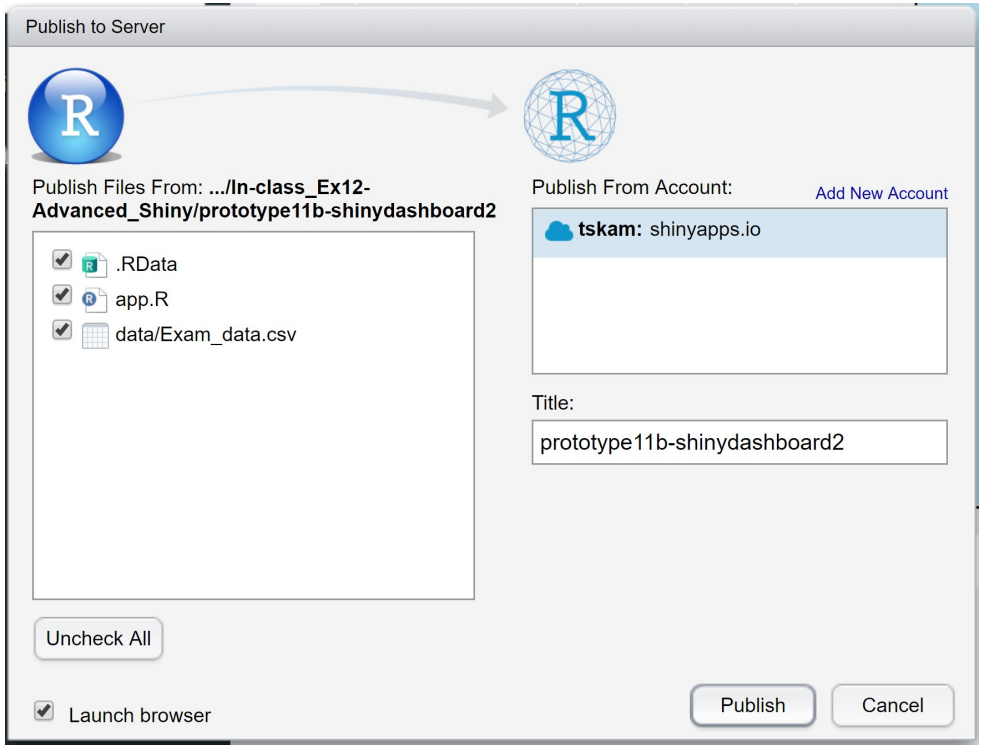
- Test that the Shiny application works by running it locally.
    - At RStudio IDE, click on **Run App** button on the editor toolbar.
- From the upper right corner of the RStudio Browser window, click on the drop-down list and select **Other Destination**.



# Deploying Shiny apps to shinyapps.io

**Step 5: Deploying the Shiny App (continue)**

The **Publish to Server** dialogue window appears.

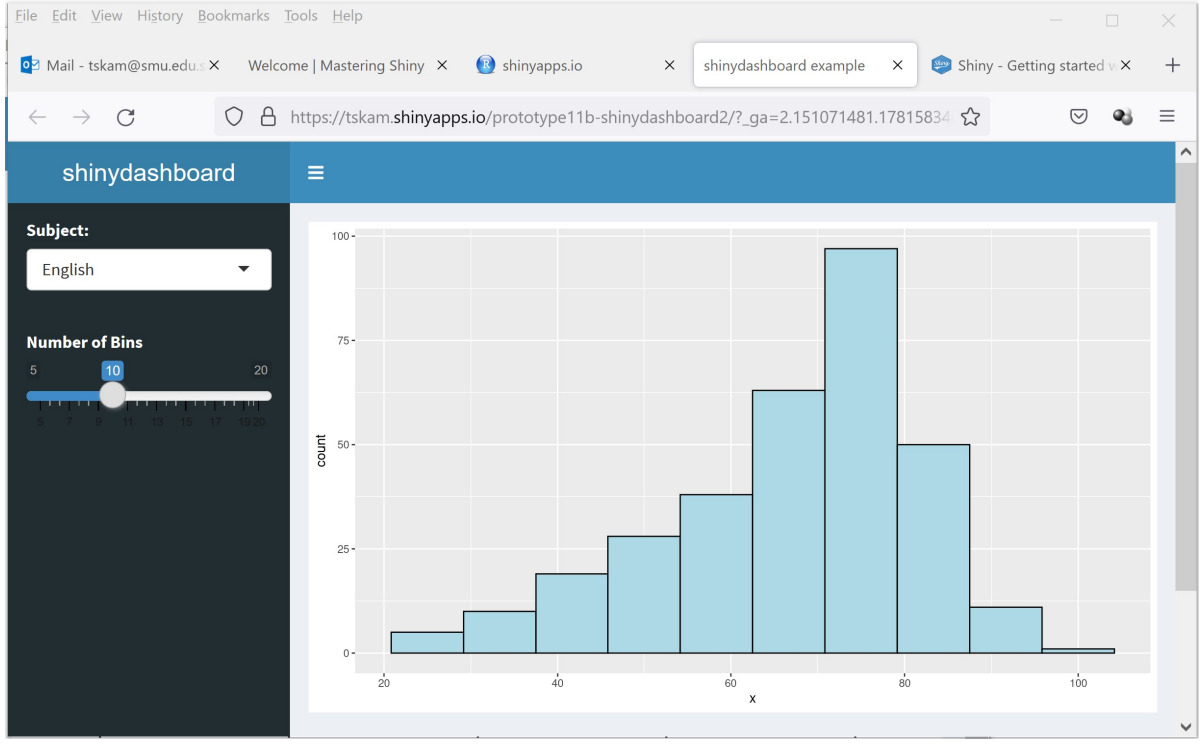



- Click on **Publish** button

# Deploying Shiny apps to shinyapps.io

**Step 5: Deploying the Shiny App (continue)**

Once the deployment finishes, your browser should open automatically to your newly deployed application.




Congratulations! You've deployed your first application. :-)

# References

## HTML and CSS

The beauty of R Shiny is that, as a data analyst and R Shiny application developer, you don't need to learn about the details of HTML CSS. However, if you know some HTML and CSS, it's possible to customise Shiny still further. The links below is a good start to learn HTML and CSS.

- HTML basics
- CSS basics

## R Shiny Productive Extension

- awesome-rshiny, a curated list of resources for R Shiny.
- Awesome Shiny Extensions: This github repository provides a comprehensive list of awesome R packages that offer extended UI or server components for the R web framework Shiny.