Build a Shiny App Demo

Impressing Hiring Teams with R + Shiny

Javier Orraca-Deatcu
javierorracadeatcu.com
orraca.javier@gmail.com

SoCal RUG
2023-03-21
Meetup: meetup.com/socal-rug
Website: socalr.org

Presentation Overview

1. About Javier
2. Shiny overview
3. Fork and clone a GitHub repo
4. Edit the Shiny demo app
5. App deployment
6. Resources

About Javier

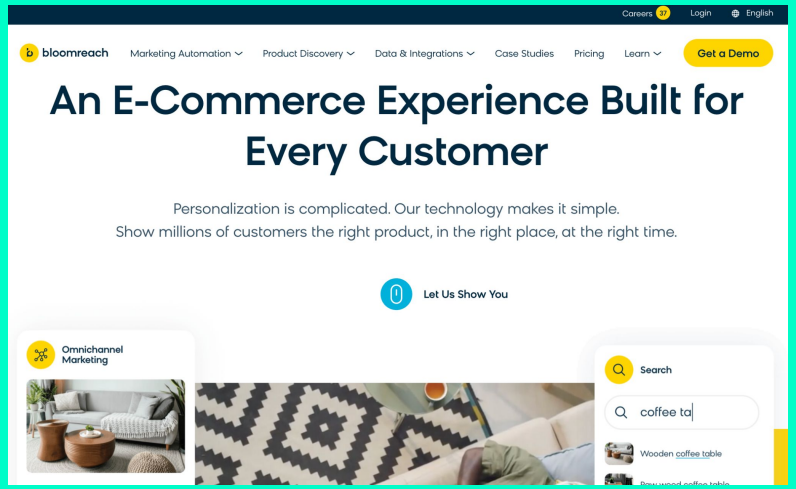
Excel → Access → SQL → R → Shiny → MLOps

```
💡 Javier Orraca-Deatcu
├─🚀 Professional Overview
│  ├─💼 Career
│  │  ├─ Centene / Lead Machine Learning Engineer
│  │  ├─ Bloomreach / Sr Manager, Data Ops & Analytics
│  │  ├─ Centene / Data Scientist III, Strategic Insights
│  │  ├─ KPMG / Sr Manager, Economics & Valuation
│  │  ├─ PG&E / Supervisor, Capital Recovery & Analysis
│  │  └─ EY / Manager, Valuation, Modeling, & Economics
│  └─📚 Education
│     ├─ UC Irvine / MSc Business Analytics
│     └─ Georgia Tech / BSc Management & Finance
├─📦 Projects
│  ├─ Personal Blog - Data science blog
│  ├─ Scatter Podcast - Analytics & data science in business
│  └─ SoCal R Users Group - Events coordinator & volunteer
```

Shiny Demo As a "Resume Accessory"

- Most corporate dashboards feel clunky or boxy, e.g., MicroStrategy, Power BI, Tableau
- Dashboards typically struggle with real-time data manipulation and calculation
- Demonstrate how easy it is to develop and deploy a Shiny web app styled with a company's logo and branding aesthetics
- This motivated the Bloomreach Shiny demo app: https://javierorraca.shinyapps.io/Bloomreach_Shiny_App/

Bloomreach's Homepage

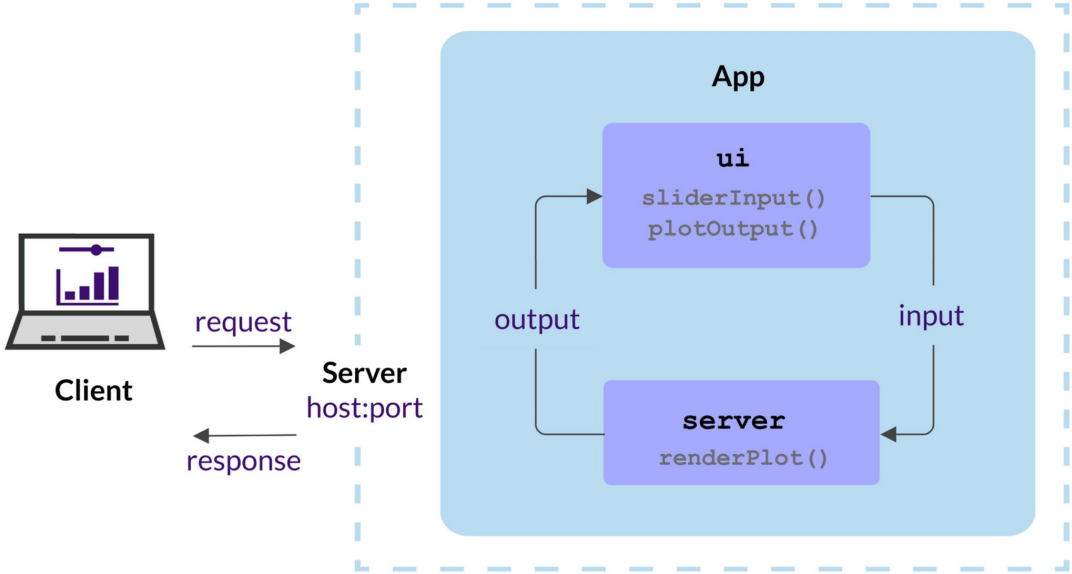


My Shiny App Demo

Shiny Overview

Shiny Overview 🚀 ☁️

- Shiny is an R package that makes it easy to build customized web apps with little to no web development experience
- Use a reactive programming model that allows for dynamic, real-time updates to the app based on user input
- Extend your Shiny apps with CSS, Sass variables, HTML widgets, JavaScript actions, and more



Structure of a Shiny web application // © Analythium

Shiny Overview: App Structure

- Key components:
  1. **Load package(s)**
  2. **Define the UI object**
  3. **Create the server function**
- The UI object controls the layout, inputs, and displays the rendered graphics
- The server function handles the app's reactivity, performs computations, and renders graphics
- Run the app by passing the UI object and server functions to `shinyApp()`

```
library(shiny)

ui <- ...

server <- function(input, output)

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

Shiny Overview: App.R

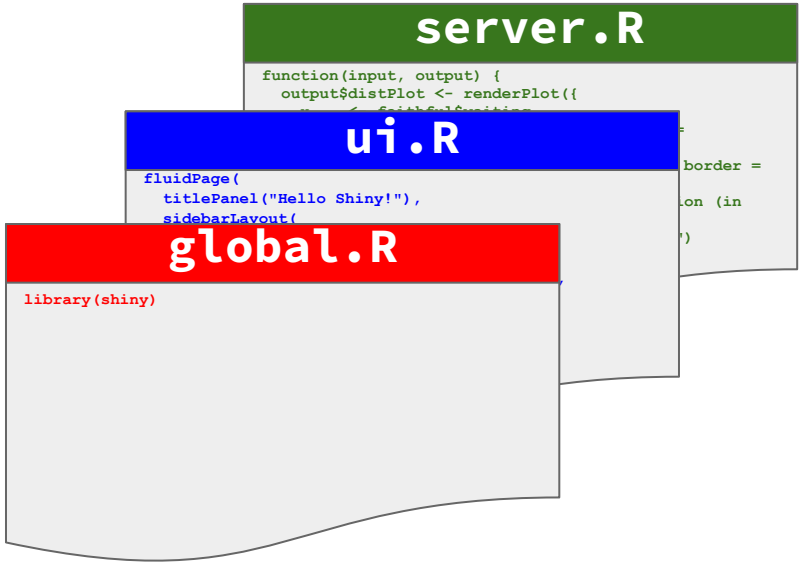
- Key components:
  1. **Load package(s)**
  2. **Define the UI object**
  3. **Create the server function**
- The UI object controls the layout, inputs, and displays the rendered graphics
- The server function handles the app's reactivity, performs computations, and renders graphics
- Run the app by passing the UI object and server functions to `shinyApp()`

app.R

```
library(shiny)

ui <- fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),
    mainPanel(
      plotOutput(outputId = "distPlot")
    )
  )
)

server <- function(input, output) {
  output$distPlot <- renderPlot({
    x    <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = "#007bc2", border = "white",
         xlab = "Waiting time to next eruption (in mins)",
         main = "Histogram of waiting times")
    })
}

shinyApp(ui, server)
```

Shiny Overview: Modular App Framework

- The Bloomreach Shiny demo app makes use of a modular framework
- Code reusability and app maintenance are easier with modular structures
- Key components are placed into their own files:
  1. `global.R`
  2. `ui.R`
  3. `server.R`
- Run the app by passing the UI object and server functions to `shinyApp()`



Shiny Overview: "Reactivity"

- A reactive expression is an R expression that uses widget input and returns a value
- The reactive expression updates the value when the widget changes
- Shiny input values are stored as elements within the `input` object, e.g., `input$bins`

app.R

```
library(shiny)

ui <- fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),
    mainPanel(
      plotOutput(outputId = "distPlot")
    )
  )
)

server <- function(input, output) {
  output$distPlot <- renderPlot({
    x    <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = "#007bc2", border = "white",
         xlab = "Waiting time to next eruption (in mins)",
         main = "Histogram of waiting times")
    })
}

shinyApp(ui, server)
```

Shiny Overview: "Reactivity"

- A reactive expression is an R expression that uses widget input and returns a value
- The reactive expression updates the value when the widget changes
- Shiny input values are stored as elements within the `input` object, e.g., `input$bins`
- Make your R expressions `reactive()` in the Shiny server by surrounding them in curly braces

mod_Overview.R

```
# in the UI of the module

selectInput(inputId = ns("region_select"),
            label = tags$b("Select Region:"),
            choices = c("All Regions", "Australia", "APAC",
                        "Europe", "North America", "South
                         America"),
            selected = "All Regions")



# in the Server of the module

industry_options <- reactive({
        req(diamonds_alt, diamonds_alt_sampled,
input$region_select)
        if(input$region_select == "All Regions"){
          diamonds_alt_sampled |>
            distinct(Industry) |>
            arrange(Industry) |>
            pull()
        }else{
          diamonds_alt |>
            filter(cut == input$region_select) |>
            distinct(Industry) |>
            arrange(Industry) |>
            pull()
        }
      })
```

Shiny Overview: (Hard) Lessons Learned

- When starting out, ***keep it simple!***
- My first enterprise Shiny app was over 5,000 lines in one `app.R` *(and still smoothly in production)* 🤯
- I've spent countless hours troubleshooting Shiny issues (***lots of StackOverflow*** *and Google)* but this made me a better Shiny dev
- I was fortunate to have a team of experienced Shiny devs teach me about Shiny extension packages and the benefits of modular app design
- You learn a lot from reviewing the code of other devs!

GitHub

Fork & Clone my Shiny Project

GitHub: Fork your own copy of my repository

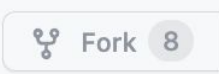
- The GitHub “fork” is a new repo that shares code and visibility settings with the original repo
- Forks let you make changes to a project and code without affecting the original repo, also known as the "upstream" repo
- After you fork a repo, you can fetch updates from the upstream repo to keep your fork current
- You can propose changes from your fork to the upstream repository with pull requests

GitHub: Fork your own copy of my repository

- The GitHub “fork” is a new repo that shares code and visibility settings with the original repo
- Forks let you make changes to a project and code without affecting the original repo, also known as the "upstream" repo
- After you fork a repo, you can fetch updates from the upstream repo to keep your fork current
- You can propose changes from your fork to the upstream repository with pull requests

Steps to fork my repo:

1. Navigate to my GitHub repo github.com/JavOrraca/bslib_demo_shiny
2. Click on the **Fork** button in the top-right of the GitHub navbar

   - This will prompt you to *“Create a new fork”* and I recommend keeping the default settings
3. Hit the **Create Fork** button on the bottom of the screen
4. You should see a newly created repo titled **bslib_demo_shiny** in your GitHub repositories

GitHub: Clone project repo to Your local system

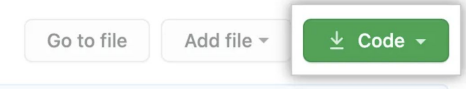
- Your newly created GitHub repo exists as a remote repository
- You can clone your repo to create a local copy on your computer
- Cloning a repo pulls down a full copy of all the data that the GitHub repo has at that snapshot in time
- Jenny Bryan's *Happy Git and GitHub for the useR* is a great resource with much more info
  - *Introduce Yourself to Git*
  - *Connect RStudio to Git & GitHub*

GitHub: Clone project repo to Your local system

- Your newly created GitHub repo exists as a remote repository
- You can clone your repo to create a local copy on your computer
- Cloning a repo pulls down a full copy of all the data that the GitHub repo has at that snapshot in time
- Jenny Bryan's *Happy Git and GitHub for the useR* is a great resource with much more info
  - *Introduce Yourself to Git*
  - *Connect RStudio to Git & GitHub*

Steps to clone your repo (or mine):

1. Navigate to your GitHub repo (or github.com/JavOrraca/bslib_demo_shiny)
2. Click the **Code** button then copy your repo's URL (HTTPS or SSH)



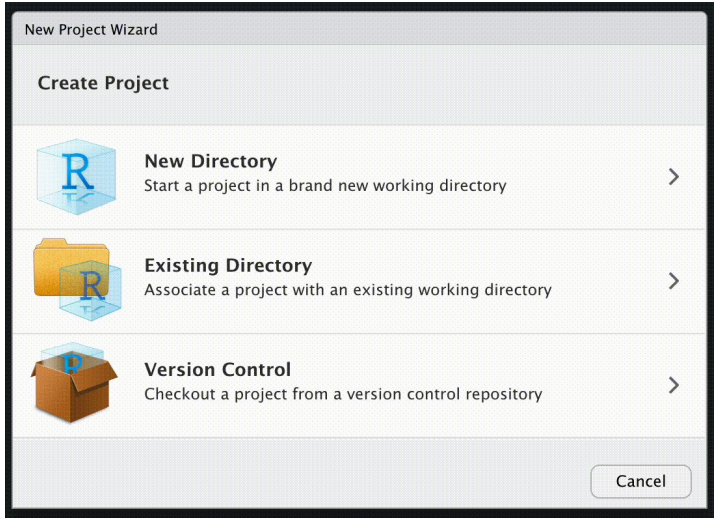
3. Using the RStudio IDE, navigate to **File** -> **New Project** -> **Version Control** -> **Git**, paste your repo's clone URL, choose your local path of choice, and click on **Create Project**.

GitHub: Clone project repo to Your local system

- Your newly created GitHub repo exists as a remote repository
- You can clone your repo to create a local copy on your computer
- Cloning a repo pulls down a full copy of all the data that the GitHub repo has at that snapshot in time
- Jenny Bryan's *Happy Git and GitHub for the useR* is a great resource with much more info
  - *Introduce Yourself to Git*
  - *Connect RStudio to Git & GitHub*

Steps to clone your repo (or mine):

4. RStudio IDE's *New Project Wizard* will launch in a popup window after selecting **New Project** (from Step #3):



Edit the Shiny demo app

(to your liking, Except this time 🙃)

Edit the Shiny demo app: Cloned File Structure

```
├── Dockerfile
├── LICENSE
├── README.md
├── bslib_demo_shiny.Rproj
├── global.R
├── helpers/
│   ├── custom_theme.R
│   ├── footer.R
│   └── navbar.R
├── modules/
│   ├── mod_About.R
│   ├── mod_Mapping.R
│   ├── mod_Overview.R
│   └── mod_Reports.R
•
•
•
```

```
├── renv/
│   ├── activate.R
│   ├── library/
│   │   └── R-4.2
│   ├── sandbox/
│   │   └── R-4.2
│   ├── settings.dcf
│   └── staging/
├── renv.lock
├── server.R
├── ui.R
├── www/
│   ├── Shiny_Demo_Preview.png
│   ├── blr_logo-primary.png
│   └── styles.scss
```

Edit the Shiny demo app: Installing R Packages

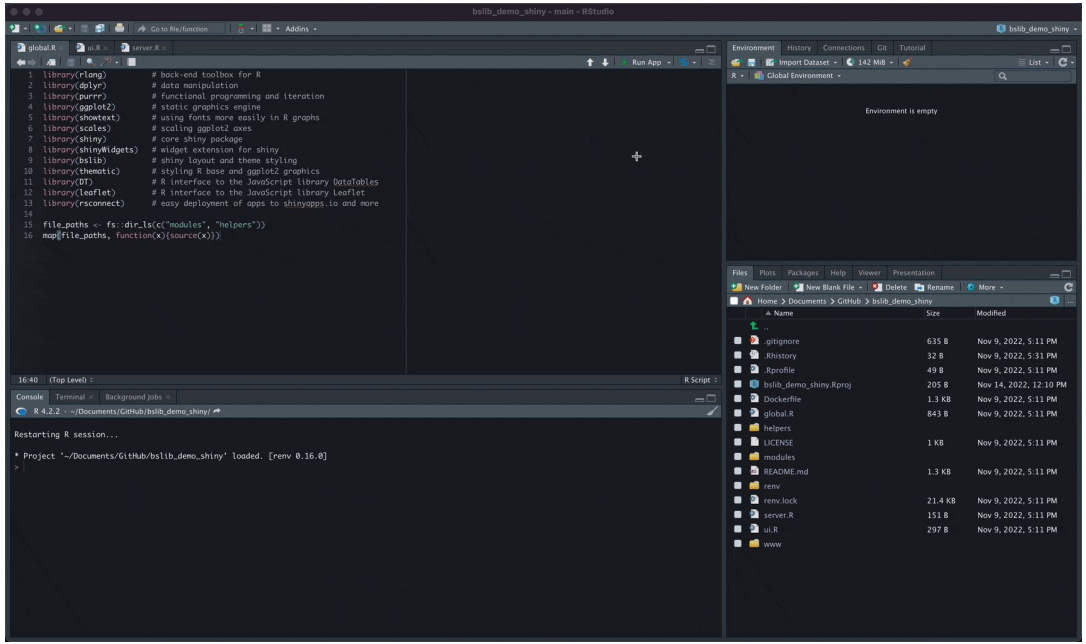
- Project-package dependencies can create production headaches 🤕
- The **{renv}** framework helps you create a **r**eproducible **env**ironment
- After cloning the repo, run `renv::restore()` to restore the package state as recorded in the `renv.lock` file

renv.lock


```
{
  "R": {
    "Version": "4.2.2",
    "Repositories": [
      {
        "Name": "CRAN",
        "URL": "https://cloud.r-project.org"
      }
    ]
  },
  "Packages": {
    "shiny": {
      "Package": "shiny",
      "Version": "1.7.3",
      "Source": "Repository",
      "Repository": "CRAN",
      "Requirements": [
        "R",
        "R6",
        "bslib",
        "cachem",
        "commonmark",
        "crayon",
        "ellipsis",
        "fastmap",
        "fontawesome",
        "glue",
        "grDevices",
        "htmltools",
        "httpuv",
        "jsonlite",
        "later",
        "lifecycle",
        "methods",
        "mime",
        "promises",
        "rlang",
        "sourcetools",
        "tools",
        "utils",
        "withr",
        "xtable"
      ],
      "Hash": "c6c3b5f560ee4..."
    }
```

Edit the Shiny demo app: Install R Packages + launch demo app

- Project-package dependencies can create production nightmares 🤕
- The {renv} framework helps you create a reproducible environment
- After cloning the repo, run `renv::restore()` to restore the package state as recorded in the `renv.lock` file
- Execute `shiny::runApp()` to launch the Shiny app or click on the "Run App" button in the RStudio IDE

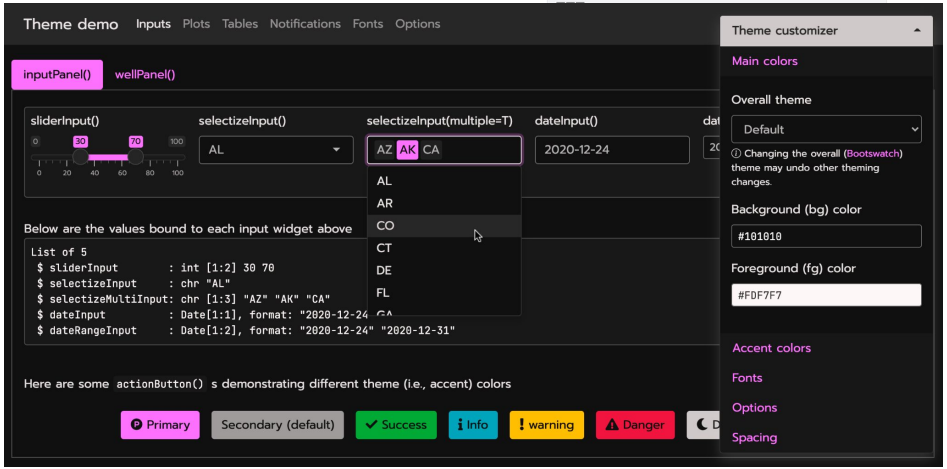


Edit the Shiny demo app: Introducing {bslib}

- {bslib} is a package for customizing Bootstrap themes and it introduces Sass variables to Shiny and R Markdown
- Use `bs_theme_preview()` to test custom themes in "real-time" via an interactive Shiny app
- See example 👉👉👉👉👉👉👉

```
# Shiny usage
navbarPage(
  theme = bs_theme(
    bg = "#101010",
    fg = "#FDF7F7",
    primary = "#ED79F9",
    base_font = font_google("Prompt"),
    code_font = font_google("JetBrains Mono")
  ),
  ...
)
```

```
# R Markdown usage
---
output:
  html_document:
    theme:
      bg: "#101010"
      fg: "#FDF7F7"
      primary: "#ED79F9"
      base_font:
        google: "Prompt"
      code_font:
        google: "JetBrains Mono"
---
```



https://rstudio.github.io/bslib // {bslib} site

Edit the Shiny demo app: {bslib} for Custom Themes

- {bslib} is a package for customizing Bootstrap themes and it introduces Sass variables to Shiny and R Markdown
- {bslib} makes app customization a unified and centralized process
- **helpers/custom_theme.R** contains two helper functions created for theming
  - **fn_custom_theme()** is called in L3 of ui.R
  - **fn_thematic_theme()** is called in L56 of modules/mod_Overview.R

helpers/custom_theme.R

```
# Overarching bslib theme
fn_custom_theme <- function() {
  bslib::bs_theme(
    version = "4",
    base_font = sass::font_google("Open Sans"),
    bg = "#ffffff",
    fg = "#1d2d42",
    primary = "#f3d436",
    secondary = "#1d2d42",
    success = "#1d2d42") |>
    bs_add_variables("border-bottom-width" = "6px",
                     "border-color" = "$primary",
                     .where = "declarations") |>
    bs_add_rules(sass::sass_file("www/styles.scss"))
}

# Thematic theme for ggplot2 outputs
fn_thematic_theme <- function() {
  thematic::thematic_theme(
    bg = "#ffffff",
    fg = "#1d2d42",
    accent = "#f3d436",
    font = font_spec(sass::font_google("Open Sans"),
                     scale = 1.75)
  )
}
```

Edit the Shiny demo app: {bslib} for Custom Themes

- {bslib} is a package for customizing Bootstrap themes and it introduces Sass variables to Shiny and R Markdown
- {bslib} makes app customization a unified and centralized process
- **helpers/custom_theme.R** contains two helper functions created for theming
  - **fn_custom_theme()** is called in L3 of ui.R
  - **fn_thematic_theme()** is called in L56 of modules/mod_Overview.R

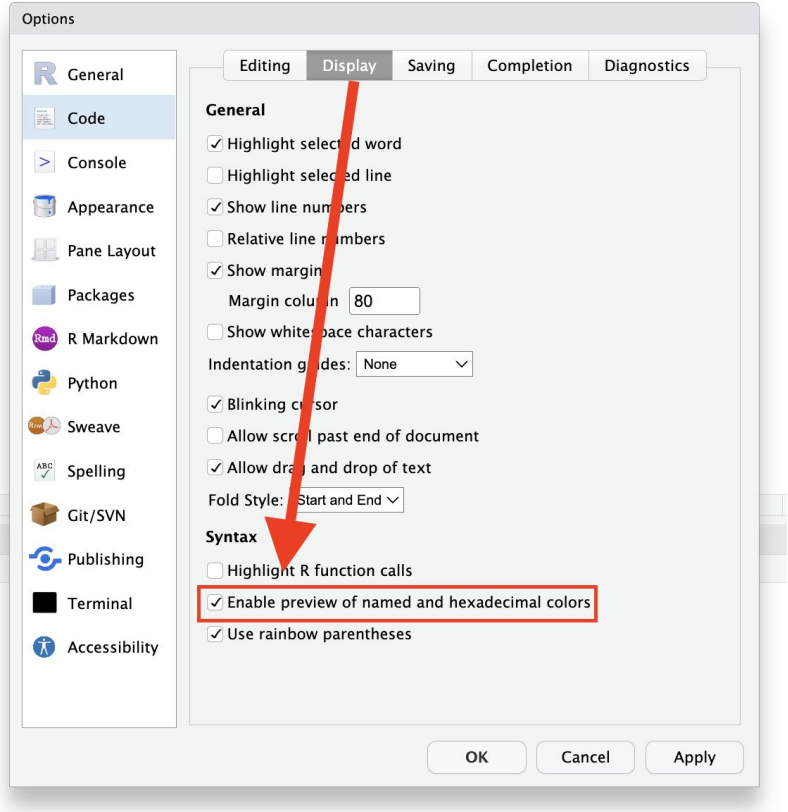
helpers/custom_theme.R

ui.R

```
tagList(
  navbarPage(
    theme = fn_custom_theme(),
    id = "navbar",
    title = fn_navbar(),
    windowTitle = "Analytics Dashboard",
    footer = fn_footer(),

    #--- M O D U L E S ---#
    ui_Overview("main"),
    ui_Mapping("main"),
    ui_Reports("main"),
    ui_About("main")
  )
)
```

🚨 RStudio IDE Pro Tip 🚨 *Color Preview!*

- To preview named or hex colors in the RStudio IDE, enable color previews in the Global Options
- Navigate to **Tools** -> **Global Options** -> **Code** -> **Display**
- Click the checkbox to *"Enable preview of named and hexadecimal colors"*



🚨 RStudio IDE Pro Tip 🚨 *color preview!*

- To preview named or hex colors in the RStudio IDE, enable color previews in the Global Options
- Navigate to **Tools** -> **Global Options** -> **Code** -> **Display**
- Click the checkbox to *"Enable preview of named and hexadecimal colors"*

custom_theme.R

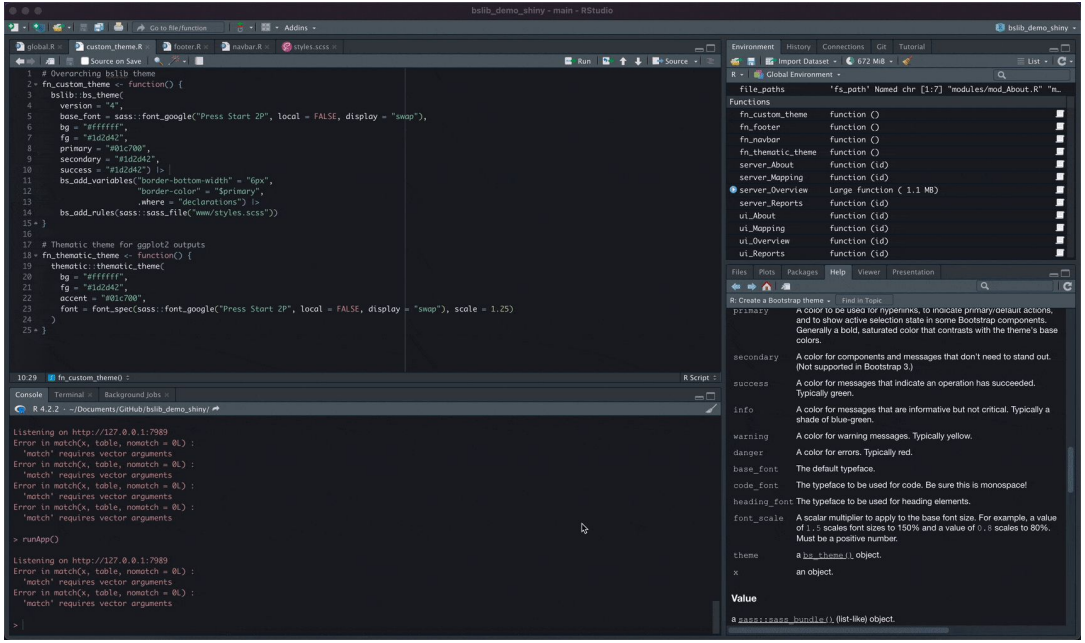
Source on Save

```r
# Overarching bslib theme
fn_custom_theme <- function() {
  bslib::bs_theme(
    version = "4",
    base_font = sass::font_google("Open Sans"),
    bg = "#ffffff",
    fg = "#1d2d42",
    primary = "#f3d436",
    secondary = "#1d2d42",
    success = "#1d2d42") |>
    bs_add_variables("border-bottom-width" = "6px",
                     "border-color" = "$primary",
                     .where = "declarations") |>
    bs_add_rules(sass::sass_file("www/styles.scss"))
}

# Thematic theme for ggplot2 outputs
fn_thematic_theme <- function() {
  thematic::thematic_theme(
    bg = "#ffffff",
    fg = "#1d2d42",
    accent = "#f3d436",
    font = font_spec(sass::font_google("Open Sans"), scale = 1.75)
  )
}
```

Edit the Shiny demo app: from bloomreach to apple

- To transform the app from a Bloomreach to Apple theme, below are the files we will modify:

  - `helpers/custom_theme.R`
  - `helpers/footer.R`
  - `helpers/navbar.R`
  - `www/retro_apple_logo.png`
  - `modules/About.R`



Edit the Shiny demo app: from bloomreach to apple

- To transform the app from a Bloomreach to Apple theme, below are the files we will modify:

  - **helpers/custom_theme.R**
  - helpers/footer.R
  - helpers/navbar.R
  - www/retro_apple_logo.png
  - modules/About.R

helpers/custom_theme.R

```
# Overarching bslib theme
fn_custom_theme <- function() {
  bslib::bs_theme(
    version = "4",
    base_font = sass::font_google("Press Start 2P", local = F, display = "swap"),
    bg = "#ffffff",
    fg = "#1d2d42",
    primary = "#01c700",
    secondary = "#1d2d42",
    success = "#1d2d42") |>
    bs_add_variables("border-bottom-width" = "6px",
                     "border-color" = "$primary",
                     .where = "declarations") |>
    bs_add_rules(sass::sass_file("www/styles.scss"))
}

# Thematic theme for ggplot2 outputs
fn_thematic_theme <- function() {
  thematic::thematic_theme(
    bg = "#ffffff",
    fg = "#1d2d42",
    accent = "#01c700",
    font = font_spec(sass::font_google("Open Sans"),
                     scale = 1.75)
  )
}
```

Edit the Shiny demo app: from bloomreach to apple

- To transform the app from a Bloomreach to Apple theme, below are the files we will modify:

  - helpers/custom_theme.R
  - **helpers/footer.R**
  - helpers/navbar.R
  - www/retro_apple_logo.png
  - modules/About.R

helpers/footer.R

```
fn_footer <- function(){
  tags$footer(
    HTML(
      "This web app demo was developed by
      <a href='http://www.woz.org'>Steve Wozniak</a>
      with R + Shiny"
    )
  )
}
```

Edit the Shiny demo app: from bloomreach to apple

- To transform the app from a Bloomreach to Apple theme, below are the files we will modify:

  - helpers/custom_theme.R
  - helpers/footer.R
  - **helpers/navbar.R**
  - www/retro_apple_logo.png
  - modules/About.R

helpers/navbar.R

```
fn_navbar <- function(){
  div(
    class = "retro_apple",
    a(href = "https://www.apple.com/",
      img(src = "retro_apple_logo.png",
          title = "Apple")
    )
  )
}
```

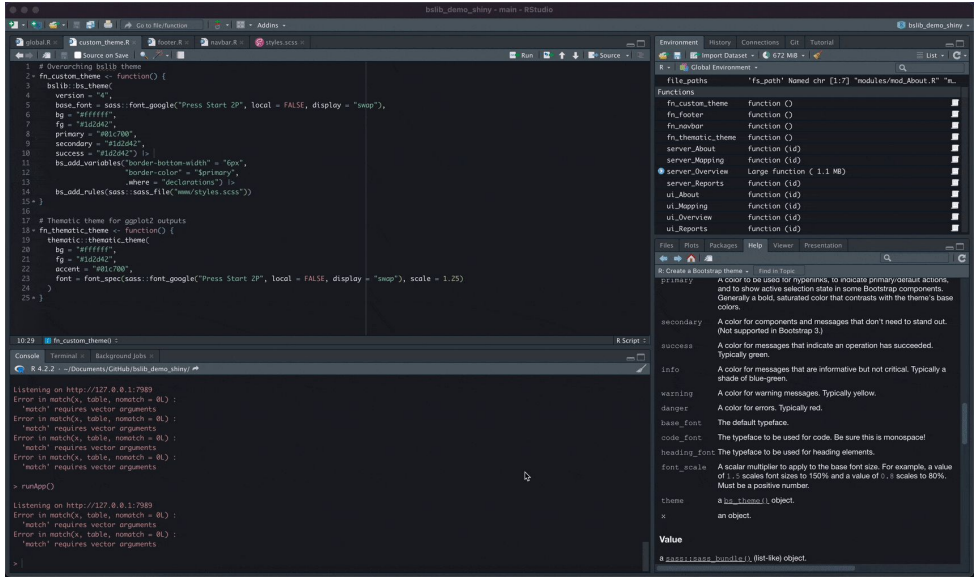
Edit the Shiny demo app: from bloomreach to apple

- To transform the app from a Bloomreach to Apple theme, below are the files we will modify:

  - helpers/custom_theme.R
  - helpers/footer.R
  - helpers/navbar.R
  - **www/retro_apple_logo.png**
  - modules/About.R

- 💡In Shiny projects, the **www** directory is used for storing elements that will be rendered in the web browser and not from script outputs

```
├── www/
│   ├── retro_apple_logo.png
│   └── styles.scss
```



Edit the Shiny demo app: from bloomreach to apple

- To transform the app from a Bloomreach to Apple theme, below are the files we will modify:

  - helpers/custom_theme.R
  - helpers/footer.R
  - helpers/navbar.R
  - www/retro_apple_logo.png
  - modules/About.R

modules/About.R

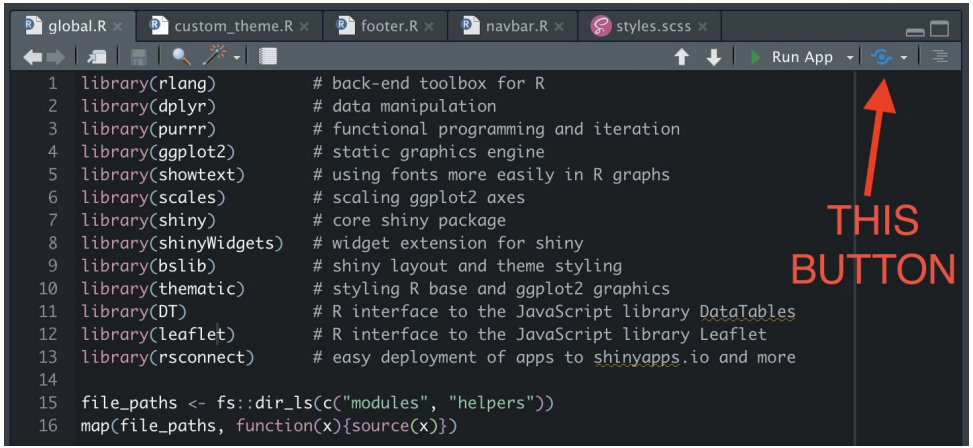
```
ui_About <- function(id){
  ns <- NS(id)

  tabPanel(
    "About",
    fluidRow(
      column(3),
      column(6,
             p(),
             h2("About The Woz"),
             "Hi, I'm Steve! I co-founded Apple. Life is good.",
      ),
      column(3)
    )
  )
}

server_About <- function(id){}
```

App Deployment

using shinyapps.io

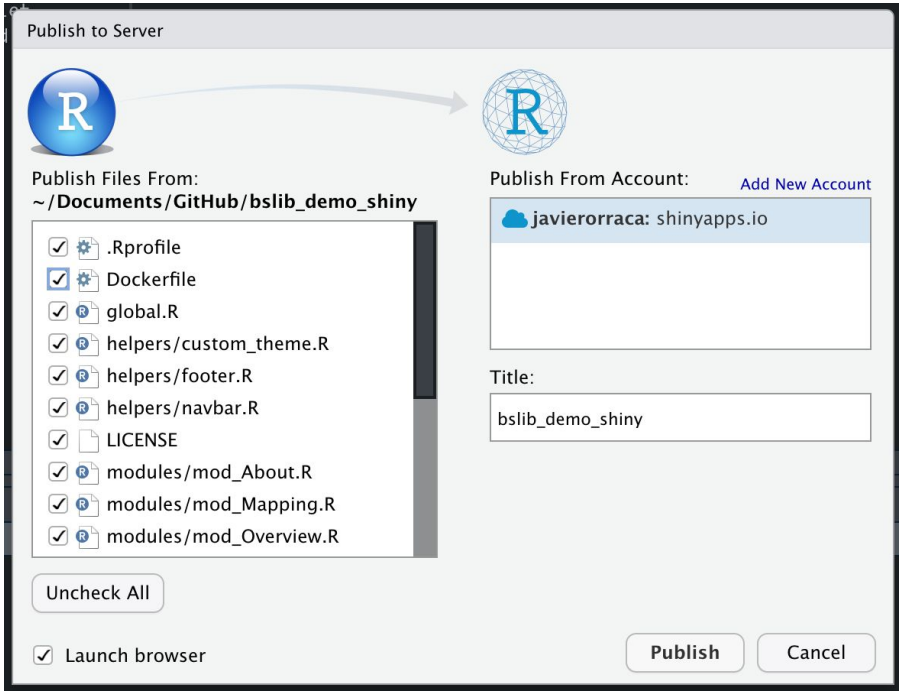
App Deployment

- shinyapps.io is a hosting platform by Posit that allows for push-button deployment from the RStudio IDE
- The free tier is sufficient for resume or “cover letter accessory” needs with up to:
  - 5 Shiny apps
  - 25 active hours/month
- If you do not have an account or you have not synced your RStudio IDE to shinyapps.io, please review their *Getting Started* documentation
- The **blue Publish button** in the RStudio IDE gives you the power of push-button deployment



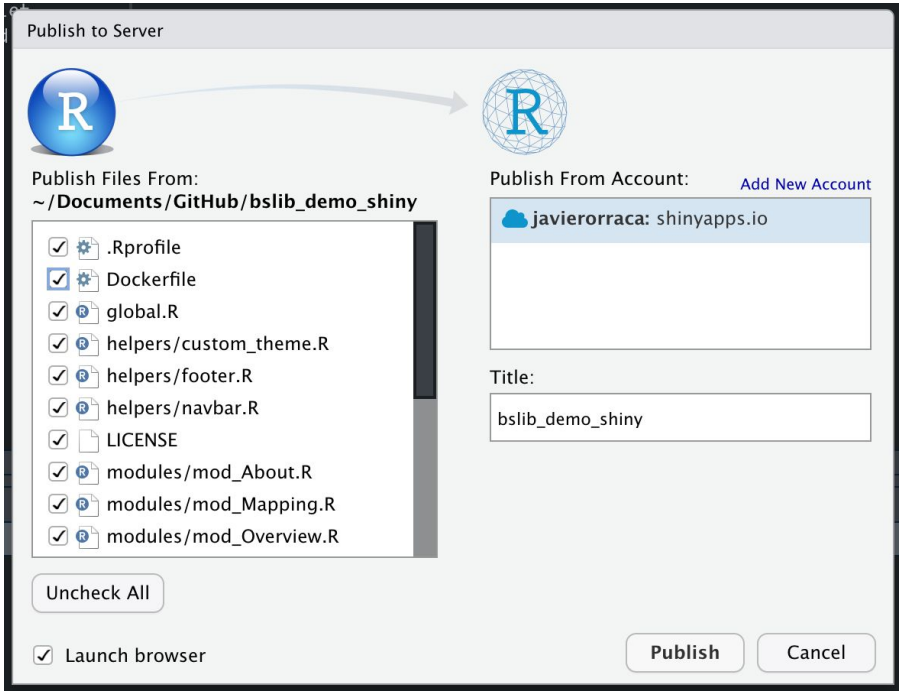
App Deployment

- Clicking the **blue Publish button** will open a new window to help you publish your app to the shinyapps.io servers
- The title you select will be the name of the application on your shinyapps.io dashboard and it will also be included in the URL of your app
- For example, I named my app "Bloomreach_Shiny_App" so the URL to visit my app is javierorraca.shinyapps.io/Bloomreach_Shiny_App



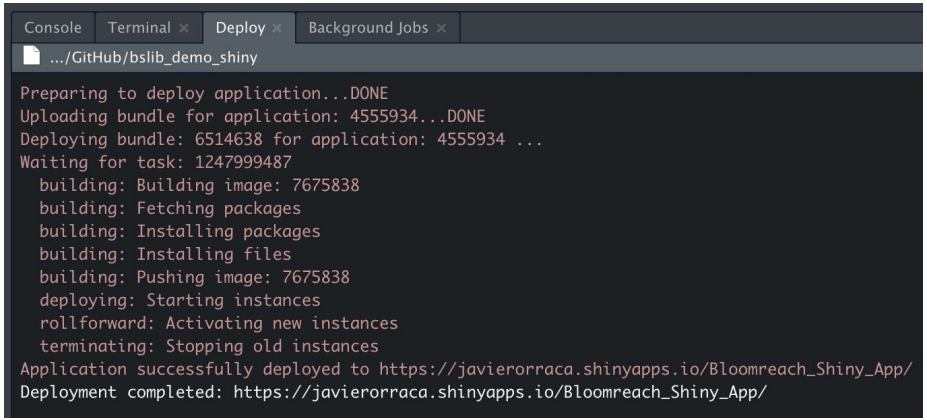
App Deployment

- If you used the push-button deployment method and already had an app on shinyapps.io with the same title, you will force overwrite the prior app
- The `Dockerfile` is not relevant to the Shiny demo app so un-check it from the files



App Deployment

- When you click on **Publish**, a new tab, *Deploy*, will be visible in the bottom-left quadrant of the RStudio IDE
- This process takes several minutes
- shinyapps.io will install an R image with the required package versions, build the Shiny app, and deploy



App Deployment

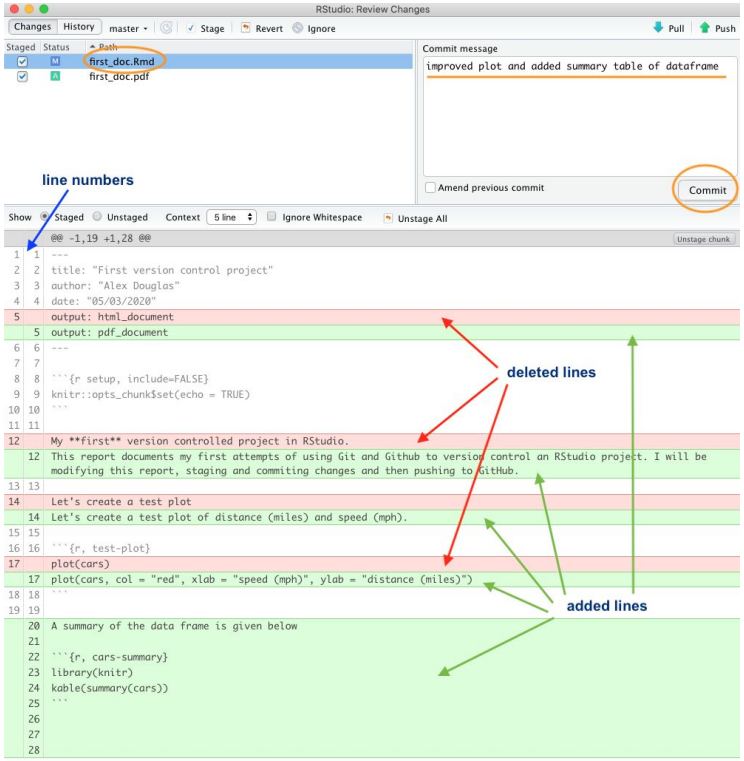
Cleaning up with Git

- Wrap up your project by saving your local changes and pushing them back to your GitHub repo

- Under the *Git* pane, click on **Commit** to open a new staging and inspection window
  - select the files you want to stage for commit and click the **Stage** button

- Write a commit message, click on the **Commit** button under the message box

- Click the **Push** button to push your locally committed changes to your remote GitHub repo



[9.6.1 Tracking Changes](#) // *An Introduction to R*

Resources

Resources

Beginner Resources

- Official Shiny website: https://shiny.rstudio.com
  - Includes example galleries for inspiration

- Deploying your Shiny app to shinyapps.io
  - https://shiny.rstudio.com/articles/shinyapps.html

- Shiny extension packages:
  - {bs4Dash}: How to start? (step-by-step)
    - Brings Bootstrap 4 support to Shiny themes
    - Relies on AdminLTE HTML template
  - {bslib}: Customizing "stock" Shiny apps
    - Globally style your Shiny Bootstrap themes
    - Use Sass variables to further customize your apps with “Sassy CSS” (*.scss)

Resources

Advanced Resources

- Hadley Wickham's *Mastering Shiny*
  - https://mastering-shiny.org/
- {golem} for modularizing and packaging Shiny apps
  - https://thinkr-open.github.io/golem/index.html
- *Engineering Production-Grade Shiny Apps*
  - https://engineering-shiny.org/
- *Outstanding User Interfaces with Shiny*
  - https://unleash-shiny.rinterface.com/
- *Automating Dockerfile creation for Shiny apps*
  - https://www.jumpingrivers.com/blog/shiny-auto-docker

Resources

Shiny Inspiration & Javier's Step-by-Step Tutorial

- Posit's Shiny gallery
  - https://shiny.rstudio.com/gallery/

- Appsilon's Shiny demos
  - https://demo.appsilon.com/

- Shiny Contest submissions by Stefan Schliebs:
  - 2021: New Zealand Commute Explorer
  - 2020: R Blog Explorer
  - 2019: R Package Explorer

- Javier's *Build a Shiny App Demo* materials
  - Live app: https://javierorraca.shinyapps.io/Bloomreach_Shiny_App
  - GitHub: https://github.com/JavOrraca/bslib_demo_shiny
  - Blog post from Nov 2022: Build a Shiny App Demo