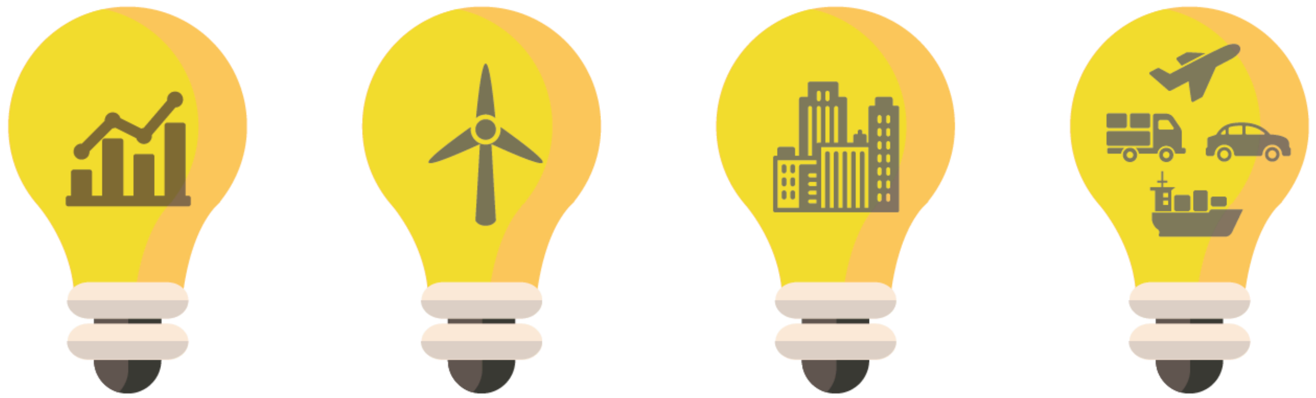
BENV0091 Lecture 7: Publishing Data Science Online

Patrick de Mars



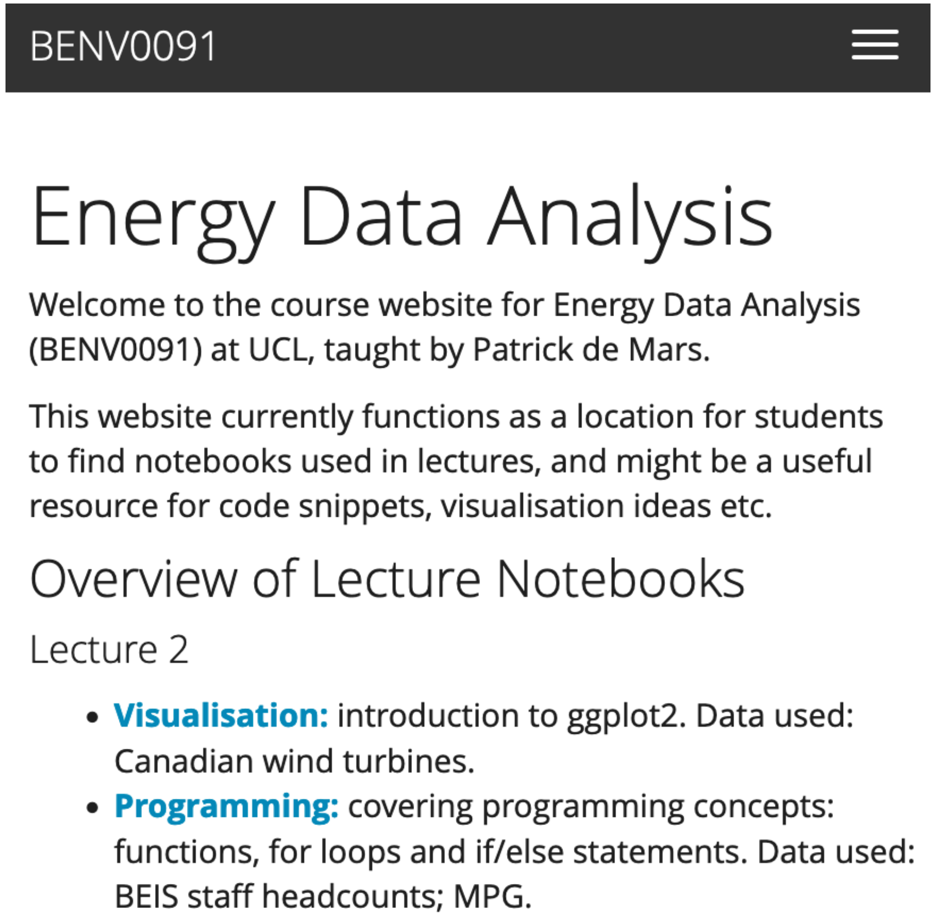
Lecture Overview

- Simple websites with R Markdown and Github Pages
- Shiny

Github Pages

- With Github Pages, you can create a static website for your repository (for free!) at: **https://{username}.github.io/{project name}**
- Github will look (**recursively**) for any HTML files in a specified directory in your repository and publish them on your Github Pages site
- **index.html** (or index.md) will be used as the homepage

The course website is hosted on GH Pages



Energy Data Analysis

Welcome to the course website for Energy Data Analysis (BENV0091) at UCL, taught by Patrick de Mars.

This website currently functions as a location for students to find notebooks used in lectures, and might be a useful resource for code snippets, visualisation ideas etc.

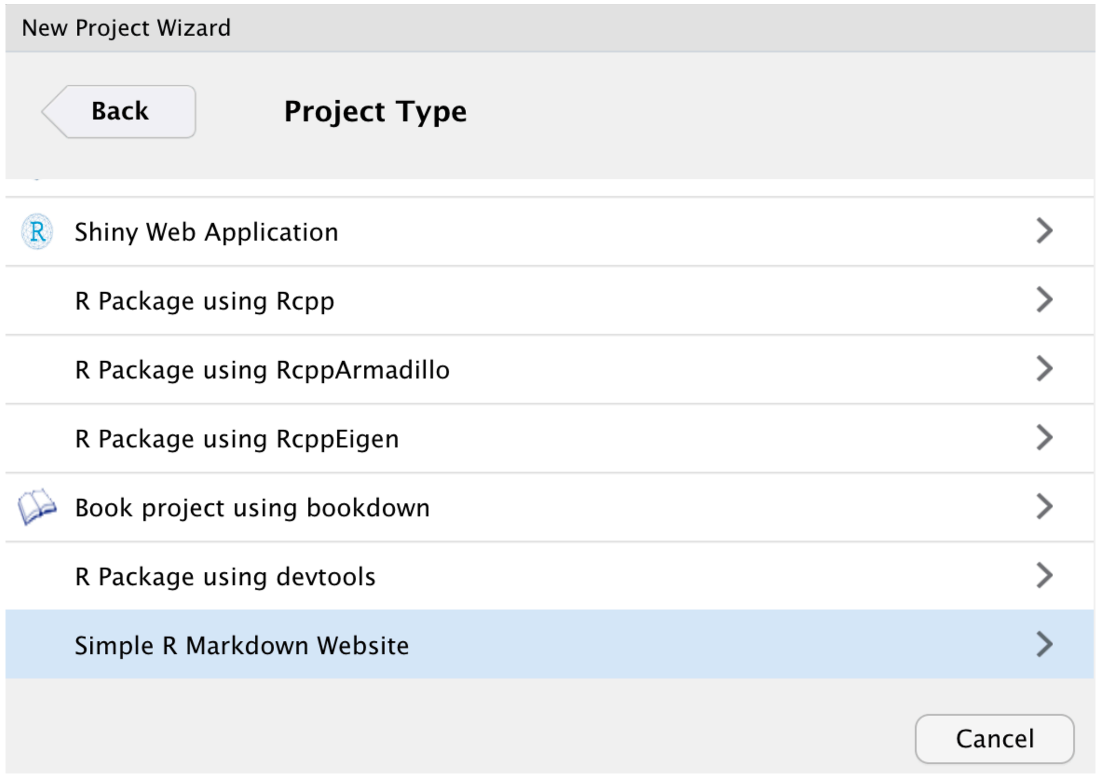
Overview of Lecture Notebooks

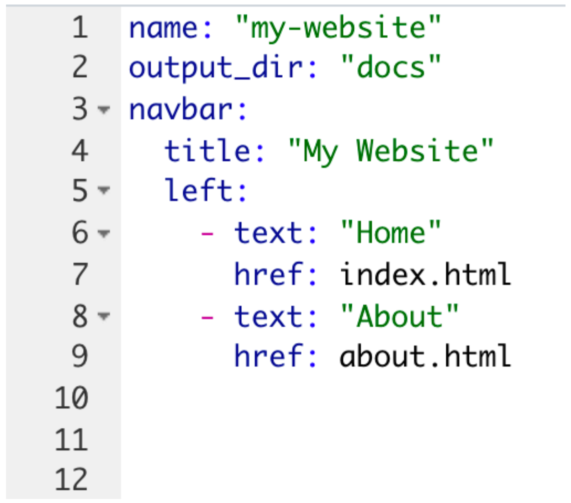
Lecture 2

- **Visualisation:** introduction to ggplot2. Data used: Canadian wind turbines.
- **Programming:** covering programming concepts: functions, for loops and if/else statements. Data used: BEIS staff headcounts; MPG.

Simple Markdown Website

- Create a New Project > Simple R Markdown Website
- In the _site.yml file, add a line: output_dir: ”docs”
- Run `rmarkdown::render_site()` to render the .Rmd files to html
- Note: you can also render your site in the `Build` tab (Build Website), and view the output in the `Viewer` tab
- This converts the .Rmd files to HTML and puts them in `docs`, along with supporting files (e.g. CSS, Javascript)

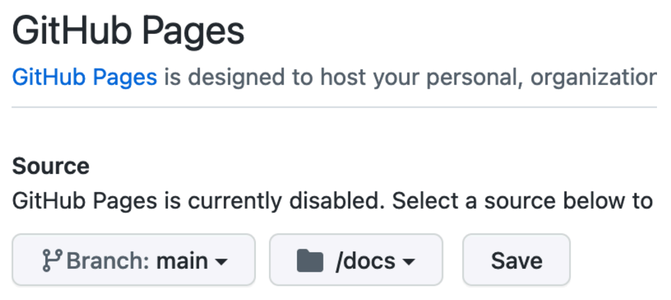

New Project Wizard
Back
Project Type
Shiny Web Application
R Package using Rcpp
R Package using RcppArmadillo
R Package using RcppEigen
Book project using bookdown
R Package using devtools
Simple R Markdown Website
Cancel



```yaml
name: "my-website"
output_dir: "docs"
navbar:
  title: "My Website"
  left:
    - text: "Home"
      href: index.html
    - text: "About"
      href: about.html



```

Github setup

- Now we need to set up our Github repository:
  - In RStudio: go to Tools > Version Control > Project Setup and set the Version Control System to Git
  - In Github Desktop: go to File > Add Local Repository and find your website repository
  - Add all the files and commit your changes
  - Publish your repository to Github (and make sure it is **public**)
  - In Github (browser) go to Settings > Pages
  - Under Source, set branch to `main` and the folder to `/docs` and press Save
  - In RStudio, make a change to your website, **render the site,** then commit and push your change *(this is sometimes necessary to refresh Github Pages during the initial setup)*
- In a few moments, your site should be published – go and visit your site!

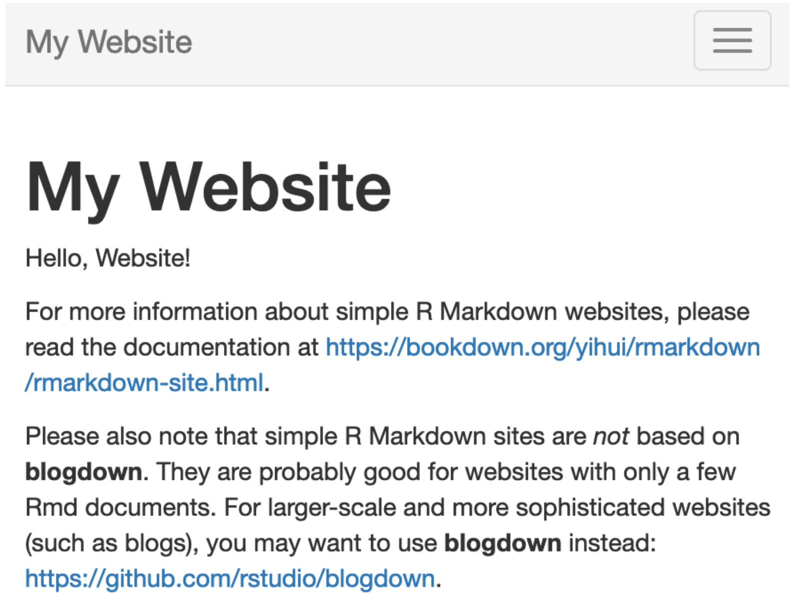


GitHub Pages

GitHub Pages is designed to host your personal, organizatior

Source

GitHub Pages is currently disabled. Select a source below to



My Website

My Website

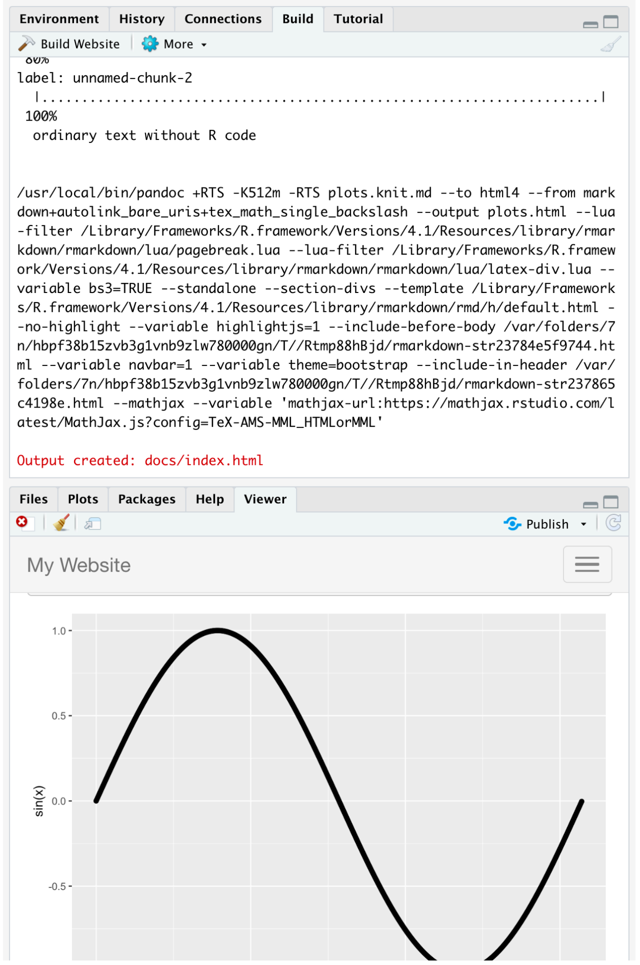
Hello, Website!

For more information about simple R Markdown websites, please read the documentation at https://bookdown.org/yihui/rmarkdown/rmarkdown-site.html.

Please also note that simple R Markdown sites are *not* based on **blogdown**. They are probably good for websites with only a few Rmd documents. For larger-scale and more sophisticated websites (such as blogs), you may want to use **blogdown** instead: https://github.com/rstudio/blogdown.

Adding More Pages

- When you run `render_site()`, all .Rmd files are rendered as html and saved in the output directory specified in your `_site.yml` file (i.e. `docs` in our case)
- Task: create a new file `plots.Rmd`
- Notice that your `_site.yml` also specifies the items in a navbar
- Task: in _site.yml, add the new page to your navbar
- Task: render the site, commit your changes and push them to Github



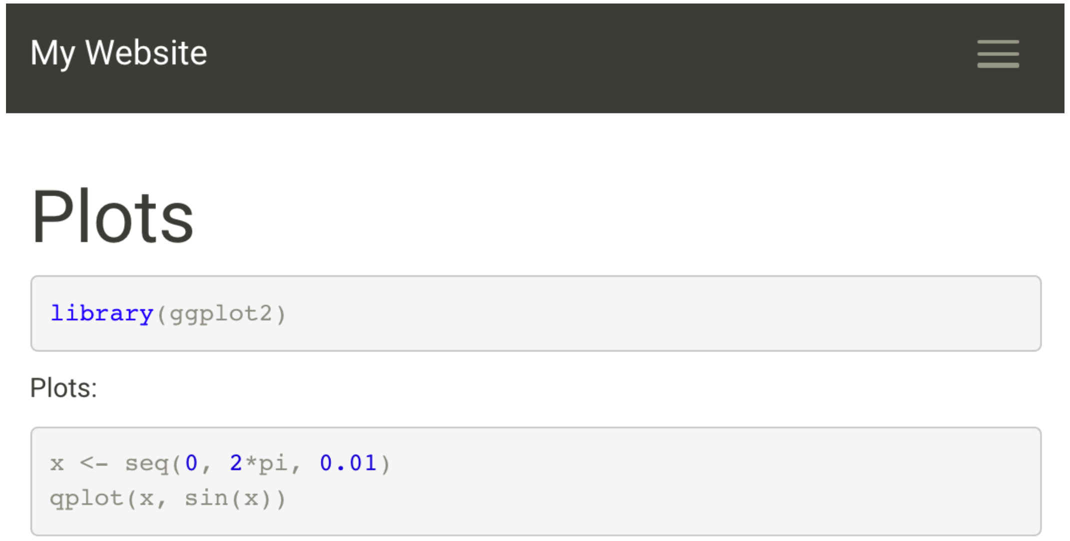
```
 80%
label: unnamed-chunk-2
  |.....................................................................|
 100%
  ordinary text without R code


/usr/local/bin/pandoc +RTS -K512m -RTS plots.knit.md --to html4 --from mark
down+autolink_bare_uris+tex_math_single_backslash --output plots.html --lua
-filter /Library/Frameworks/R.framework/Versions/4.1/Resources/library/rmar
kdown/rmarkdown/lua/pagebreak.lua --lua-filter /Library/Frameworks/R.framew
ork/Versions/4.1/Resources/library/rmarkdown/rmarkdown/lua/latex-div.lua --
variable bs3=TRUE --standalone --section-divs --template /Library/Framework
s/R.framework/Versions/4.1/Resources/library/rmarkdown/rmd/h/default.html -
-no-highlight --variable highlightjs=1 --include-before-body /var/folders/7
n/hbpf38b15zvb3g1vnb9zlw780000gn/T//Rtmp88hBjd/rmarkdown-str23784e5f9744.ht
ml --variable navbar=1 --variable theme=bootstrap --include-in-header /var/
folders/7n/hbpf38b15zvb3g1vnb9zlw780000gn/T//Rtmp88hBjd/rmarkdown-str237865
c4198e.html --mathjax --variable 'mathjax-url:https://mathjax.rstudio.com/l
atest/MathJax.js?config=TeX-AMS-MML_HTMLorMML'

Output created: docs/index.html
```

Customising your Website

- Setting the `theme` option in `_site.yml` allows you to set CSS themes (from the Bootswatch theme library), such as:
  - cerulean
  - journal
  - Darkly
- The `highlight` option can be used to change syntax highlighting options
- You can also use a custom CSS file, e.g. by adding `css: styles.css` to your _site.yml file

Customising _site.yml

```
output:
  html_document:
    theme: sandstone
    highlight: textmate
```



Plots

```
library(ggplot2)
```

Plots:

```
x <- seq(0, 2*pi, 0.01)
qplot(x, sin(x))
```

Adding other HTML files and media

- Github pages will also recognise any other HTML files, including those in sub-directories
- For instance: the file `docs/notebooks/data-cleaning.html` will be found at the URL: https://{username}.github.io/{projectname}/notebooks/data-cleaning.html
- However: .Rmd files in subdirectories will not be rendered to HTML (which is a big limitation of R Markdown sites!)
- As well as the navbar, you can also include links in your .Rmd files (such as index.Rmd) for site navigation

Links in Markdown

```
The analysis section can be found
[here](path/to/analysis)
```

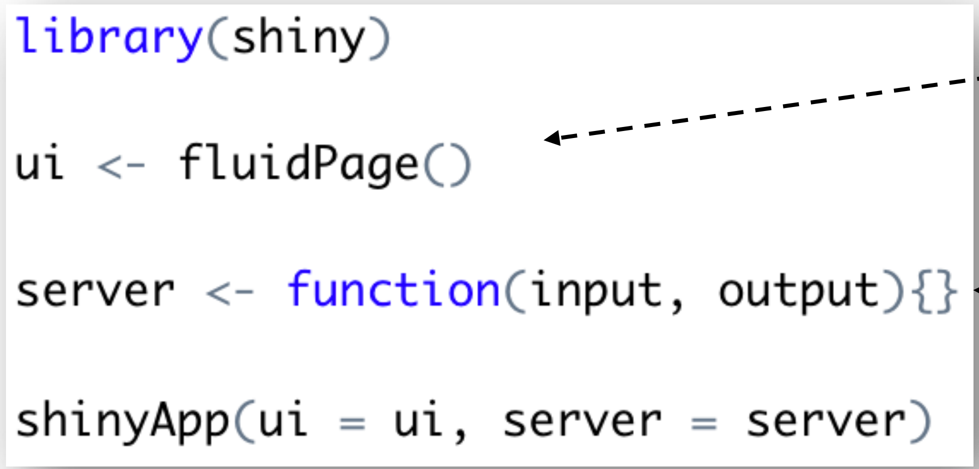
Shiny

What is Shiny?

- Shiny is an R package that lets you build interactive web apps
- The user can change variables using dropdown menus, sliders, buttons etc. in a **user interface (UI)** and view the results
- A Shiny app is maintained by a **server** running an R session
- The code running on the server responds **reactively** to user inputs to produce outputs such as plots, tables, text etc.
- Plots and tables are updated and rendered as HTML

App Template

- You can build a Shiny app with only a few lines of code:



```r
library(shiny)

ui <- fluidPage()

server <- function(input, output){}

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

The 'front end'

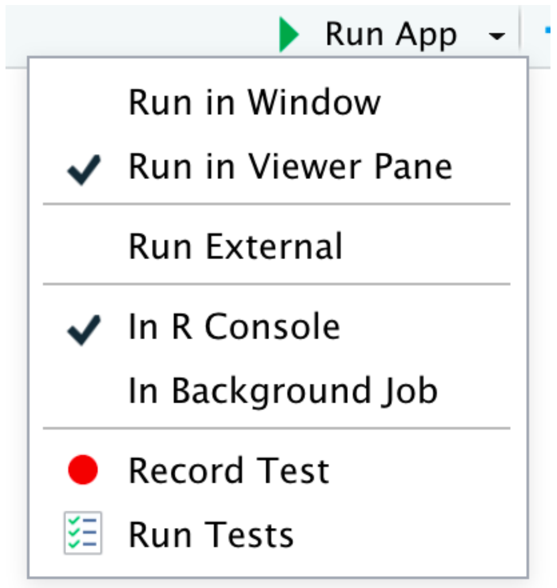
The 'back end': your R code goes here!

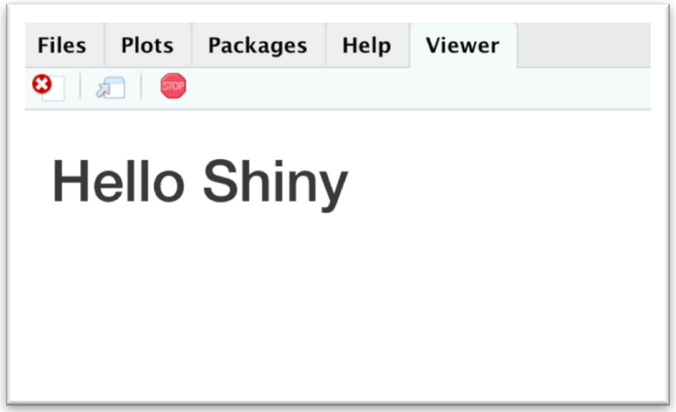
Knitting the UI and server together to run the app

- All of this code should be saved in a file called app.R
- Alternatively, you can split your code over 2 files: ui.R and server.R

Setting up an App

- Task: make sure you have the **shiny** package installed
- We will now set up a simple app:
  - Create a new directory called `my_app`
  - Inside `my_app`, create a file called `app.R`
  - Copy the code from the previous slide and save
  - Add `titlePanel("My App")` as an argument to `fluidPage()`
  - Run your app (top right)!

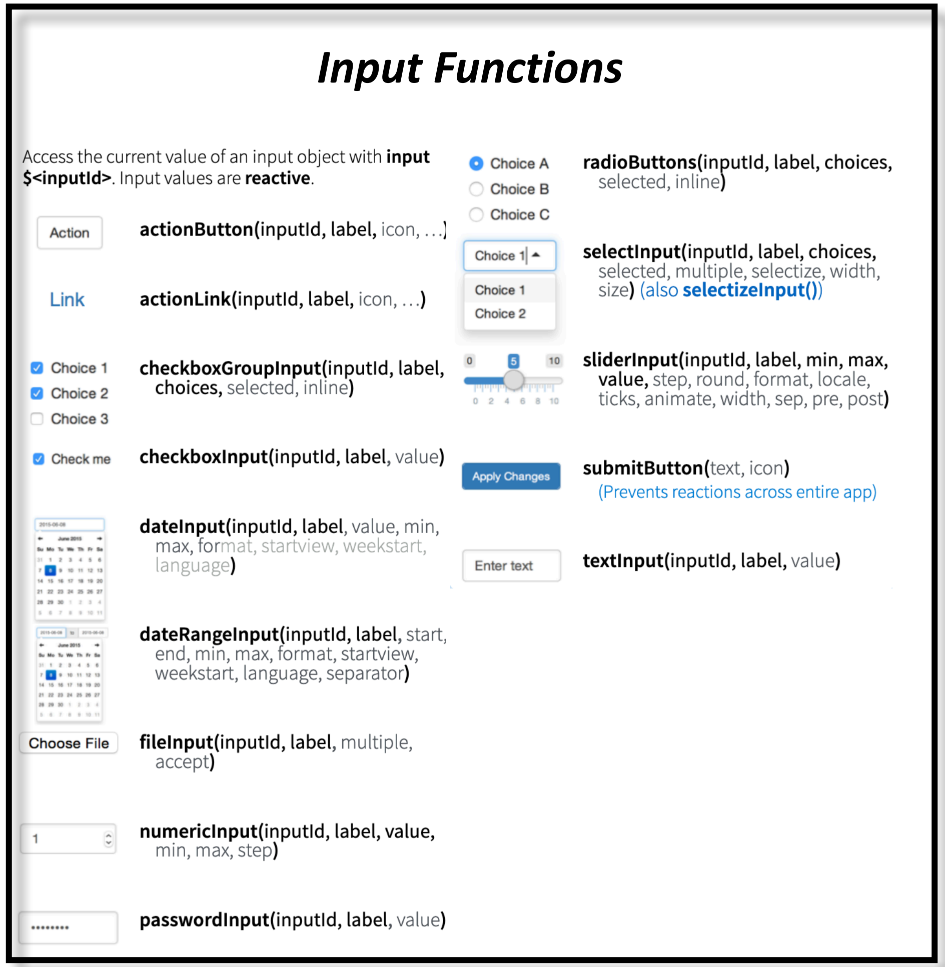




Viewing your app in the Viewer pane

Input

- Adding **input functions** to the UI (`fluidPage`) component of your code creates variables which respond to user inputs
- These **reactive values** can be referenced in your server code using `input$inputId`, where inputId is a specified variable name



Action
Link
Choice 1
Choice 2
Choice 3
Check me
Choose File
Choice A
Choice B
Choice C
Apply Changes
Enter text

Input Functions

Access the current value of an input object with **input $<inputId>**. Input values are **reactive**.

actionButton(inputId, label, icon, …**)**

actionLink(inputId, label, icon, …**)**

checkboxGroupInput(inputId, label, choices, selected, inline**)**

checkboxInput(inputId, label, value**)**

dateInput(inputId, label, value, min, max, format, startview, weekstart, language**)**

dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator**)**

fileInput(inputId, label, multiple, accept**)**

numericInput(inputId, label, value, min, max, step**)**

passwordInput(inputId, label, value**)**

radioButtons(inputId, label, choices, selected, inline**)**

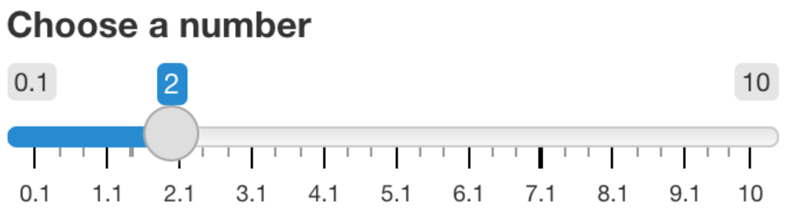
selectInput(inputId, label, choices, selected, multiple, selectize, width, size**)** (also **selectizeInput()**)

sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post**)**

submitButton(text, icon**)**
(Prevents reactions across entire app)

textInput(inputId, label, value**)**

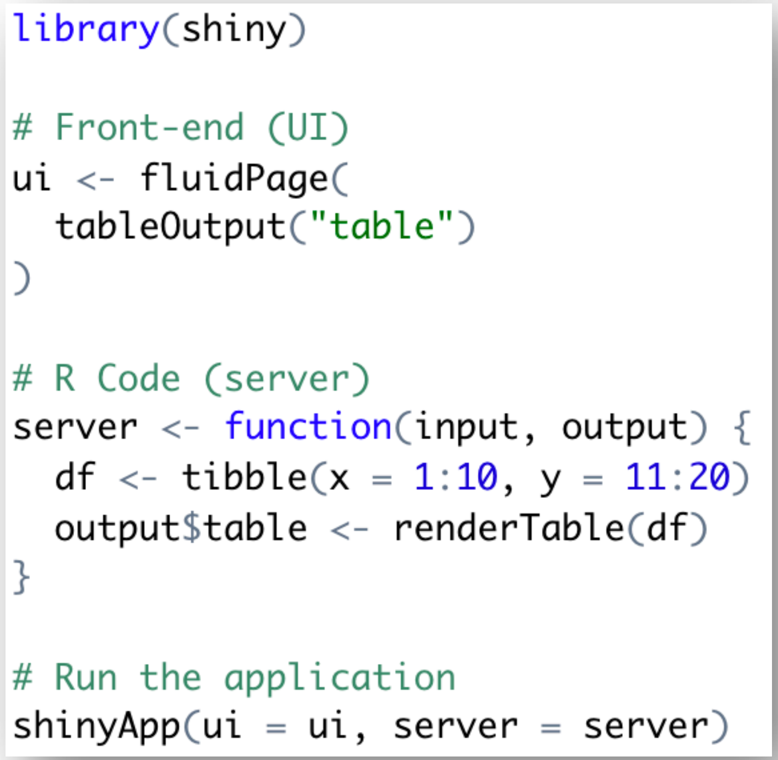
```
sliderInput(inputId = "num",
            label = "Choose a number",
            value = 2, min = 0.1, max = 10),
```



```
input$num = 2
```

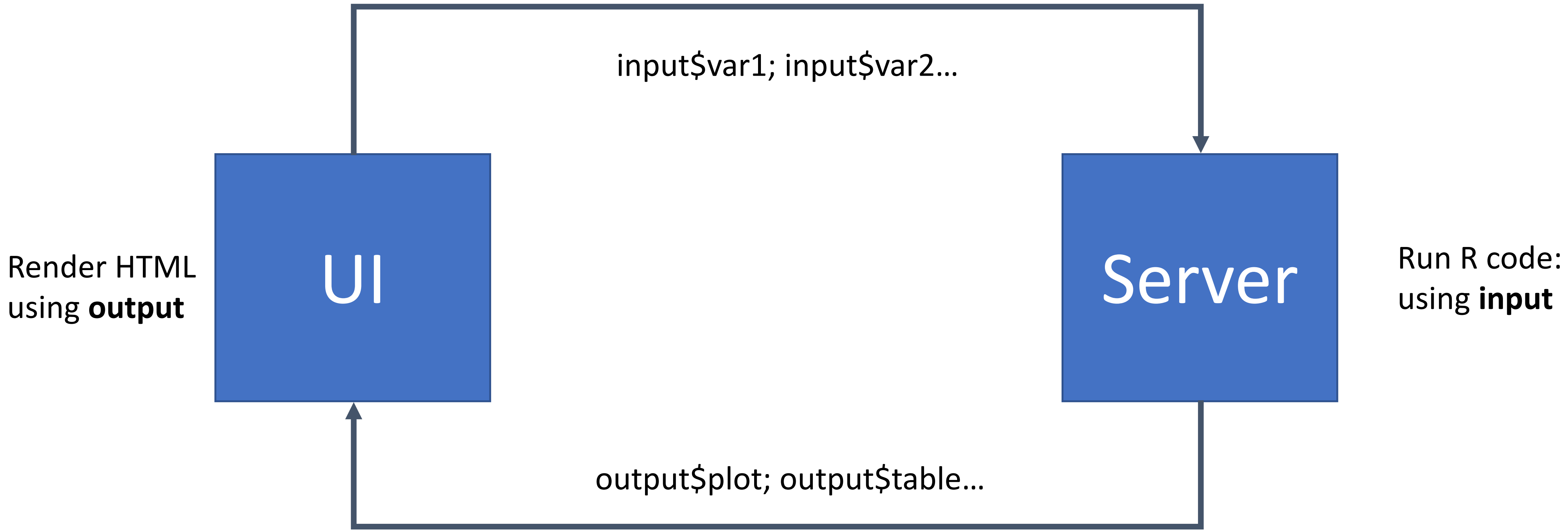
Output

- Similarly, we can create outputs in the server code, which can be referenced in the UI code
- You must use a **render function** to create an object which can be referenced in the UI:
    - renderTable
    - renderPlot
- Then, in the UI, you must use the corresponding **output function** to render the object in the web app
    - tableOutput
    - plotOutput
- The code on the right creates a simple table **(there are no reactive values (inputs in this case)**
- Task: reproduce the web app on the right
- Task: change the code to output a scatter plot of y vs. x



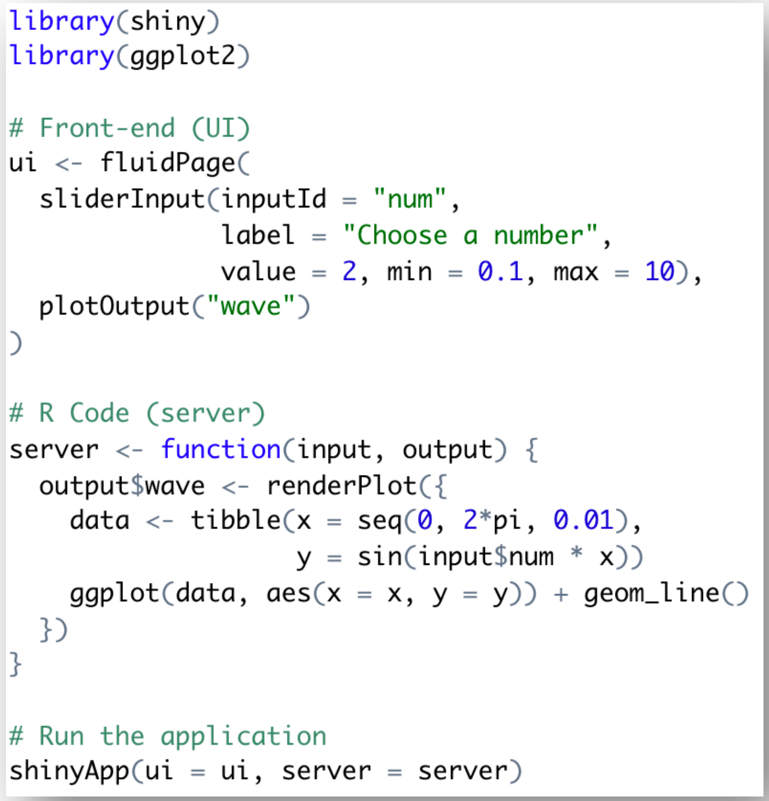
```r
library(shiny)

# Front-end (UI)
ui <- fluidPage(
  tableOutput("table")
)

# R Code (server)
server <- function(input, output) {
  df <- tibble(x = 1:10, y = 11:20)
  output$table <- renderTable(df)
}

# Run the application
shinyApp(ui = ui, server = server)
```

Summary


input$var1; input$var2…
Render HTML
using **output**
UI
Server
Run R code:
using **input**
output$plot; output$table…

Example App: Sine Wave

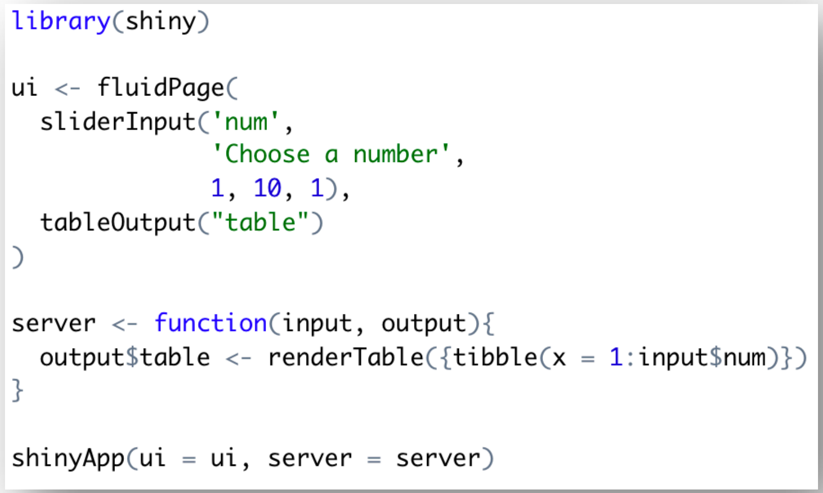
- Now we will combine reactive values (inputs) with outputs
- The example app on the right allows the user to change the value c in the equation $y = \sin(cx)$ and plots the results
- Task: create a new directory `sine_wave_app`
- Task: copy the code on the right into a file `app.R` in the new directory
- Task: run the app
- Task: change the input function to `numericInput`
- Note: the source code is available on the Github repo: https://github.com/pwdemars/BENV0091/tree/main/lecture7/sine_wave_app



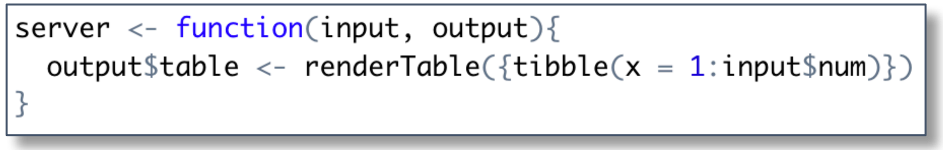
```r
library(shiny)
library(ggplot2)

# Front-end (UI)
ui <- fluidPage(
  sliderInput(inputId = "num",
              label = "Choose a number",
              value = 2, min = 0.1, max = 10),
  plotOutput("wave")
)

# R Code (server)
server <- function(input, output) {
  output$wave <- renderPlot({
    data <- tibble(x = seq(0, 2*pi, 0.01),
                   y = sin(input$num * x))
    ggplot(data, aes(x = x, y = y)) + geom_line()
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Reactive Functions

- Not all code on the server side will respond to a change in `input`
- Functions which respond to reactive values are called **reactive functions,** and include the `render*()` functions
- If you try to reference `input` outside of a reactive function, R will throw an error!

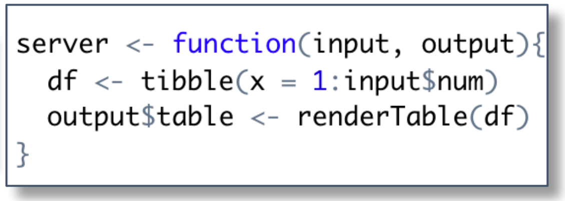


```r
library(shiny)

ui <- fluidPage(
  sliderInput('num',
              'Choose a number',
              1, 10, 1),
  tableOutput("table")
)

server <- function(input, output){
  output$table <- renderTable({tibble(x = 1:input$num)})
}

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



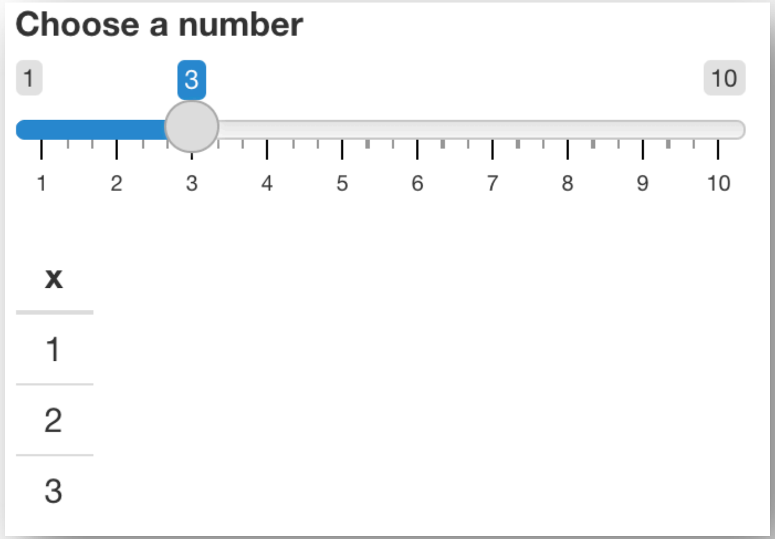
```r
server <- function(input, output){
  output$table <- renderTable({tibble(x = 1:input$num)})
}
```

Inside reactive function



```r
server <- function(input, output){
  df <- tibble(x = 1:input$num)
  output$table <- renderTable(df)
}
```

Outside reactive function

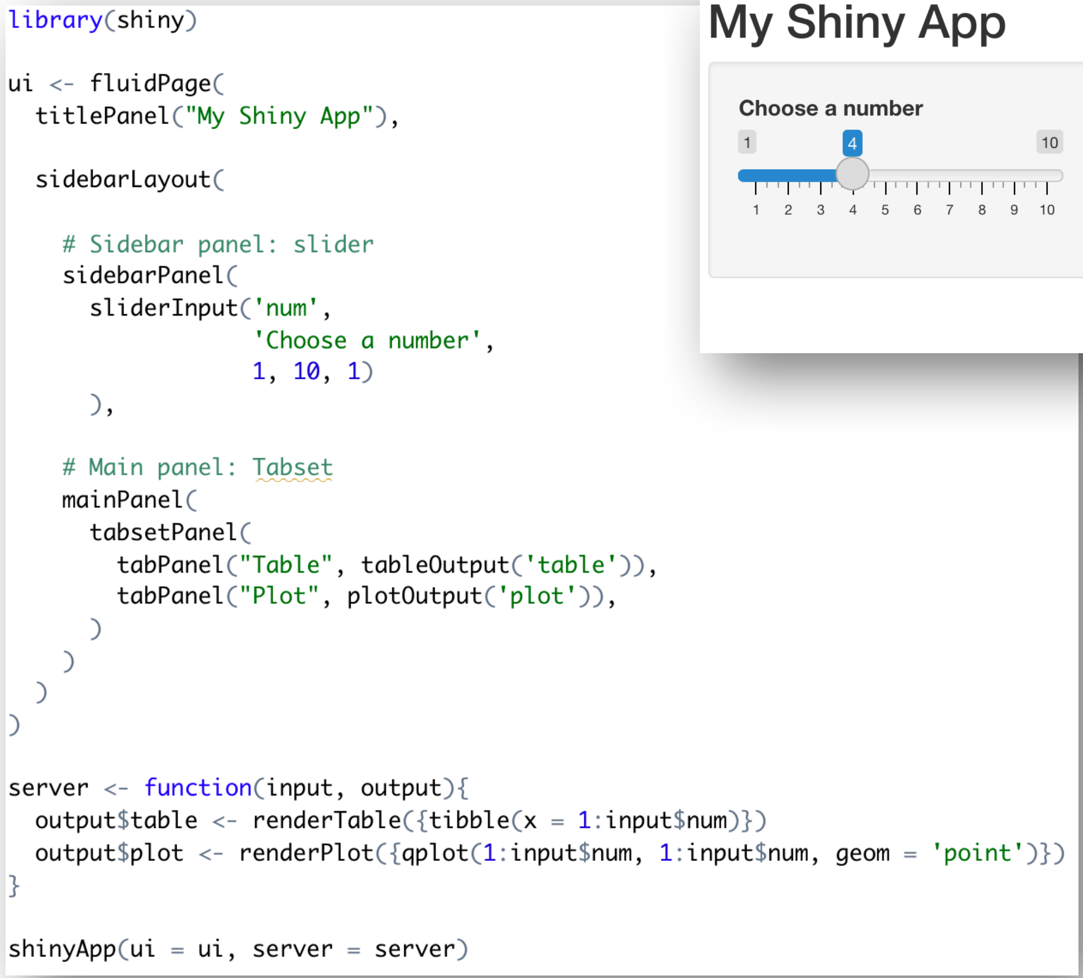


Choose a number

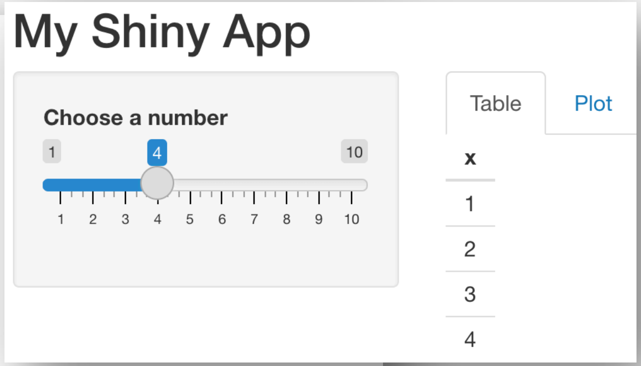
x
1
2
3

Customisation: Layouts

- The `sidebarLayout()` enables you to add a **sidebar panel** next to a **main panel**
- Tabsets can be created with `tabsetPanel()`
- You can also create more complex arrangements with `fluidRow()`, as well as apps with multiple pages using `navbarPage()`
- For a guide to layouts: https://shiny.rstudio.com/articles/layout-guide.html



```r
library(shiny)

ui <- fluidPage(
  titlePanel("My Shiny App"),

  sidebarLayout(

    # Sidebar panel: slider
    sidebarPanel(
      sliderInput('num',
                  'Choose a number',
                  1, 10, 1)
      ),

    # Main panel: Tabset
    mainPanel(
      tabsetPanel(
        tabPanel("Table", tableOutput('table')),
        tabPanel("Plot", plotOutput('plot')),
      )
    )
  )
)

server <- function(input, output){
  output$table <- renderTable({tibble(x = 1:input$num)})
  output$plot <- renderPlot({qplot(1:input$num, 1:input$num, geom = 'point')})
}

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



My Shiny App

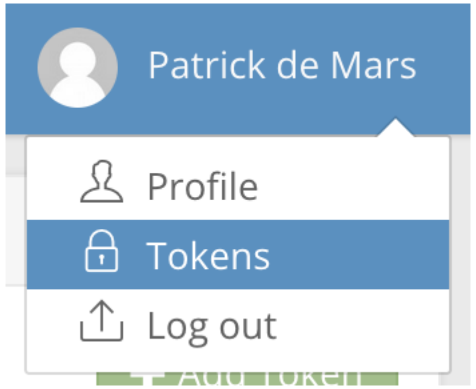
Choose a number

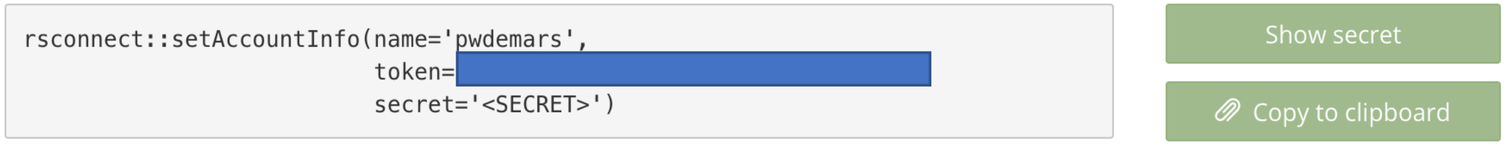
x
1
2
3
4

Sidebar layout with tabs

shinyapps.io

- **shinyapps.io** is a (mostly free) platform for hosting Shiny web apps
- Task: sign up for an account at shinyapps.io
- Shiny uses the **rsconnect** package to securely enable you to make changes and reploy/redeploy your app
- Task: install rsconnect with `install.packages()`
- Once you have signed up to shinyapps.io, you can generate a token and a secret which enables you will need to configure your account with rsconnect
- Task: generate a token and secret and copy the code to run in the R Studio console





```
rsconnect::setAccountInfo(name='pwdemars',
                          token=[redacted]
                          secret='<SECRET>')
```

Deploying your App

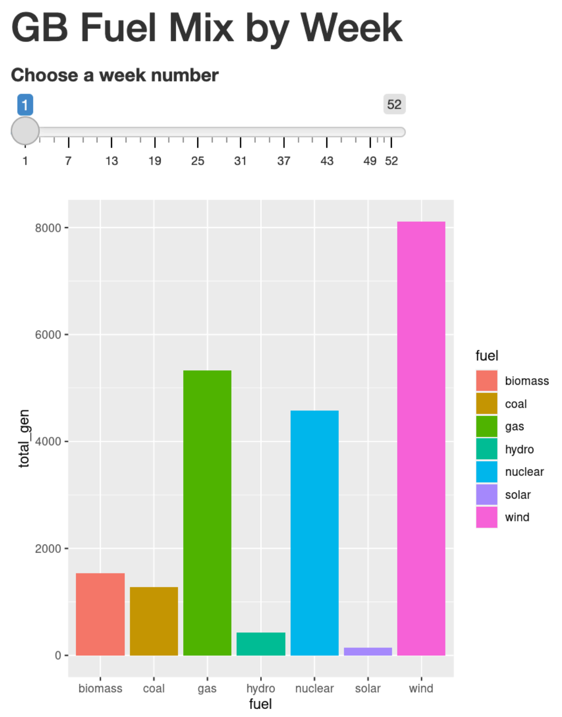
- You are now all set up to deploy your app!
- Running `rsconnect::deployApp()` will deploy your app, which will be found at: https://{username}.shinyapps.io/{app_name}
  - Note that the `app_name` is the name of your directory by default, but can be changed with the argument `appName`
  - See the documentation `?deployApp` for more deployment options
- Task: deploy your app
- Task: check your app runs by visiting the URL in a web browser

Including Data and Additional Files

Source code

https://github.com/pwdemars/BENV0091/tree/main/lecture7/electric_insights_app

- So long as your app is packaged in a single directory with **app.R** at the root, you can include additional files (e.g. data or code)
- Go to: **lecture7/electric_insights_app** in the course repo (see right for URL)
- The app.R file uses `` `source()` `` to load the plotting function in electric_insights_plot.R
- The data is stored in a sub-directory \`data\`



App URL

https://pwdemars.shinyapps.io/electric_insights_app/

Combining Github Pages and Shiny?

- shinyapps.io is unique in offering free hosting of Shiny apps - unfortunately you cannot host your Shiny app through Github Pages ☹
- But you can always include a link from your Github Pages site to shinyapps.io!

Further Reading

- Shiny apps are highly customisable and can be powerful tools for disseminating your data science work
- Some resources:
  - Learn Shiny: https://shiny.rstudio.com/tutorial/
  - shinyapps.io user guide: https://docs.rstudio.com/shinyapps.io/
  - Mastering Shiny – Hadley Wickham