Making interactive web-apps in R using Shiny

Stuart Lacy

17/02/2021

Introduction

- Learning objective: understand what Shiny is, the potential uses it has for research, and be able to setup a basic app after this talk
- Assumes you have some familiarity with R, but no experience with web development is required
- Follow along with the examples by cloning this repository:
- File -> New Project -> Version Control -> Git -> Repository URL: `https://github.com/stulacy/shiny-introduction.git`

What is Shiny and why would I want to use it?

- Web-app framework using R
- Don't need to know any Javascript/CSS/web hosting
- Developed by the RStudio/tidyverse team
- Use cases:
  - Exploring large datasets
  - Visualise live data
  - Provide tools to accompany published research
  - Establish an online presence

Example Shiny apps

What does Shiny code look like?

- Code is organised into 2 files:
- 'ui.R'
  - Defines the widgets and their layout on the page
  - Widgets can be outputs such as plots, tables, text
  - And inputs, such as buttons, sliders, text fields etc...
- 'server.R'
  - Defines all back end logic needed to process incoming inputs and generate the required outputs
- For every widget there is a corresponding function call in the UI (`renderX`) and in the server (`xOutput`)

Example: 1_basic

Adding interactivity

- We can take advantage of Javascript to add interactivity to output visual elements
- There are libraries that provide convenient wrappers for interactive widgets:
  - `plotly`: general plotting library with controls
  - `DataTables`: tables
  - `leaflet`: maps
  - `networkD3`: network diagrams
  - `diagrammeR`: graphs and flowcharts

Example: 2_interactive

User inputs

- We can pass user inputs from `ui.R` to `server.R`, process the input accordingly, and return an updated output
- In Shiny, this is done with the concept of reactivity
- UI elements are referenced in `server.R` through the `input` object, i.e. the current value of a dropdown menu
- Any output expressions that use an input value are reactive, and will re-evaluate when the input value changes

Example: 3_inputs

Input types

- Lots of input types:
  - Buttons
  - Checkboxes
  - Date selection/range
  - File upload/download
  - Text input
  - Radio buttons
  - Dropdown menu
  - Sliders

Customising your app's appearance

- By default all Shiny apps are responsive, i.e. they adapt their layout to the size of the viewing device
- To add more structure to an app, you can use a 'sidebarLayout' to separate inputs from outputs
- You can partition your app further with tabbed output using `tablistPanel`
- You can create entirely independent pages with `navbarPage`

Example: 4_ui_sidebar

Example: 5_ui_tabs

Example: 6_ui_pages

Further visual customisation

- You can organise elements into rows (`fluidRow`) and columns (`column`)
- The bslib package provides different themes
- shinydashboard package provides new UI elements for creating dashboard style displays
- Can add CSS to have fine control over each element's appearance
- Add flourishes with Javascript (`shinyjs` package provides some useful features such as loading spinners)

More advanced tips

- You can create your own reactive elements, rather than just using `input`. See the Shiny documentation
- You can dynamically create UI elements using `renderUI` and `uiOutput`
- Code can start to get messy with larger multi-page apps - recommend putting each page into its own files and using `source` to load them in (example here) or using modules

Example: 7_reactive_part2

Hosting:

- University provide free hosting at shiny.york.ac.uk
- 2 methods:
  - Managed: provide them access to a GitHub repo and it will automatically update whenever there is a change to the main branch
  - Self-hosted: You get given access to a folder where you can put your app and any dependencies/data you need
- Email `itsupport@york.ac.uk` for access
- Shinyapps.io has free hosting provided by RStudio team

Conclusions

- Biggest strength: Shiny makes it easy to get apps up and running with a wide range of features to cover most use cases
- Biggest weakness: For larger apps, the code can become hard to navigate, particularly if you start adding in custom JS