Best practices for building, branding, and testing interactive web applications in R

Abstract

This talk will feature a gentle introduction to interactive web application development in the R programming language with two open source package libraries: `shiny` and `shinytest`. Both packages are used widely in the data science community and are actively maintained by engineering teams at RStudio, PBC. Shiny is an R package that makes it easy to build interactive web apps straight from R, without prerequisite knowledge of web development skills such as HTML, javascript and CSS. Shiny applications are particularly useful for developing dashboards and performing exploratory data analysis through interactive filtering and visualization. The `shinytest` package provides tools for creating and running automated tests on Shiny applications and the underlying code used to create them. Testing code is a general best practice for all forms of development, and Shiny applications are not an exception.

Introduction

RStudio, PBC maintains free and open source resources and documentation for the `shiny` package, which are contributed to by members of the RStudio Shiny development team, education team, and solutions engineering team, among others (RStudio 2020b).

Use of Shiny Applications for Clinical Data Science

Shiny is an R package that makes it easy to build interactive web apps straight from R, without prerequisite knowledge of web development skills such as HTML, javascript and CSS. Shiny applications are particularly useful for developing dashboards and performing exploratory data analysis through interactive filtering and visualization (Wickham 2020).

Shiny applications are used in a variety of industry verticals including pharmaceuticals, health care, medicine and clinical data science. In a 2019 joint webinar between RStudio, Roche/Genentech, Sebastian Wolf, a Scientific Software Developer at Roche Diagnostics, delivered an introduction to how Roche leverages Shiny for providing self-service statistics applications. Wolf described three hard functional requirements for the shiny web application framework to be successful in a clinical data science context. Shiny applications must be: fast, robust, and well constructed to be adopted as successful workflow tooling at Roche. In turn, successful adoption yeilds significant productivity gains, which Roche has experienced through R and `shiny` development lead by Wolf and others.

> *QA processes in Roche Diagnostics are dependent on statistical evaluations. As such evaluations follow certain standard operating procedures, the biostats department decided to enable users doing them by themselves. The R-Shiny app bioWARP brings standard procedures such as linear regression or equivalence tests to people who cannot code R. It saves them time they would have spent consulting a Biostatistician or using validated Excel sheets.* (Wolf 2019)

The joint RStudio, Roche/Genentech webinar, *"The Role of R in Drug Discovery, Research, and Development"* was recorded, and is freely available to watch at the website: `resources.rstudio.com/webinars`.

The webinar featured four speakers from Roche/Genentech and was orchestrated by the RStudio Director of Life Sciences, Phil Bowsher. Talks focused on the diverse ways R can be used in different stages of the pharma lifecycle, spanning research and discovery, through to development and market access. Contact information for Phil Bowsher can be found at the end of this paper, following the Acknowledgements section.

Building Shiny Applications

Chapter 2 of the book, *Mastering Shiny*, currently under development with O'Reilly Media and written by Hadley Whickam, Chief Data Scientist at RStudio, is an excellent starting primer for learning the basics of Shiny Application development. *Mastering Shiny* is freely available online at `mastering-shiny.org` and is the primary reference for this section.

Steps to create a simple Shiny Application

Run the following commands to install and load the `shiny` package in your current R session:

```
install.packages('shiny')
library(shiny)
```

Create a new directory for your application, and put a single file called `app.R` in it:

```
#Contents of the app.R file:

library(shiny)

ui <- fluidPage(
  "Hello, world!"
)
server <- function(input, output, session) {
}
shinyApp(ui, server)
```

This is a complete shiny application. The code in `app.R` accomplishes four things:

- Makes a call to library(shiny) to load the shiny package.
- Defines the user interface, the HTML webpage that humans interact with. In this case, a page containing the words "Hello, world!".
- Specifies the behavior of our app by defining a server function (currently empty - no behavior defined).
- Executes `shinyApp(ui, server)` to construct and start a Shiny application from the UI and server functions.

To learn more about adding UI controls, interactive behavior, and basic reactivity, continue reading *Mastering Shiny* Chapter 2, "Your first Shiny app" available at *mastering-shiny.org* (Wickham 2020).

Testing Shiny Applications

The `shinytest` package provides tools for creating and running automated tests on Shiny applications and the underlying code used to create them by way of a snapshot-based testing strategy. The first time it runs a set of tests for an application, it performs some scripted interactions with the app and takes one or more snapshots of the application's state. These snapshots are saved to disk so that future runs of the tests can compare their results to them.

Why should you test Shiny Applications?

After you get your Shiny application to a state where it works, it's often useful to have an automated system that checks that it continues to work as expected. There are many possible reasons for an application to stop working. These reasons include:

- An upgraded R package has different behavior.
- You make modifications to your application.
- An external data source stops working, or returns data in a changed format.

One way to detect these problems is with manual testing – in other words, by having a person interact with the app in a browser – but this can be time-intensive, inconsistent, and imprecise. Having automated tests can alert you to these kinds of problems quickly and with almost zero effort, after the tests have been created.

Reference: `shinytest` documentation website (RStudio 2020d)

Basic testing proceedure

The RStudio IDE provides a graphical interface for recording tests of Shiny applications. These `shinytest` integration features are available in version 1.2 of the IDE and above.

1. Run `recordTest()` to launch the app in a test recorder.
2. Create tests by interacting with the application and telling the recorder to snapshot the state at various points.
3. Quit the test recorder. Upon exiting the recording session, three things will happen:

- The test script will be saved in a `.R` file in a subdirectory of the application named `tests/`.
- If you are running in the RStudio IDE, it will automatically open this file in the editor.
- The test script will be run, and the snapshots will be saved in a subdirectory of the `tests/` directory.

Reference: `shinytest` documentation website (RStudio 2020d)

Automated testing

Test automation is by no means necessary, but it can be powerful, especially when developing applications that might need to be maintained and improved over time. Testing procedures can be cumbersome if they become part of daily work, but are easily automated with the help of continuous integration (CI) platforms.

The `shinytest` vignette for continuous integration covers how to use Travis CI, which is commonly paired with GitHub. Use of Tavis CI and GitHub are not required; the pattern described can be replicated using many differnent code management and CI platforms. The CI vignette contains instructions for testing applications that exist as components of R packages as well as ones that stand alone, i.e., applications comprised of files at the top level of the repository, outside of a package framework.

The overall procedure for enabling tests on a CI platform is as follows:

- Create tests locally and save the expected results as detailed in the previous section.
- Commit the expected results into the project's version control repository.
- Enable a CI platform integration for the project repository.

Once continuous integration is enabled, the typical development cycle is as follows:

- Make a development code change to the project, then commit, and push the changes.
- The CI platform will automatically do a build, in which it downloads the code and runs tests. If the tests fail, it will raise an alert (typically email-based).

As application development matures over time, it may become appropriate to add, remove, or modify tests. It is always possible to re-run tests and save new expected results.

- Reference: `shinytest` documentation website (RStudio 2020d)

Note: As of this writing, the `shinytest` CI vignette references use of a package called `packrat` for maintaining a list of packages used in an application, and the version of each package currently installed in the development environment. Use of `packrat` for maintaining package environment information on a project basis will still work, but is no longer recommended by RStudio as a best practice. Instead, we recommend the use of a newer package, `renv` (Ushey 2019). A link to an updated template project that utilizes `renv` over `packrat` is provided below:

Updated GitHub and Travis CI template for getting started with `shinytest` automation and `renv`: `github.com/kellobri/shinytest-travis-ci` (O'Briant 2019)

Conclusion

The Importance of Reproducibility

Great data science work should be reproducible. Being able to repeat experiments is the foundation of all science. Reproducing work is also critical for business applications: scheduled reporting, team collaboration, project validation (RStudio 2020a).

At RStudio, our thesis is that data science, as a scientific process, requires writing code. We build software that serves the day to day needs of data science teams that do their work in code. Data science is all about finding meaningful opportunities in the systems that exist around us. Scientific data discovery is inherently a complex and messy process. Without process and rigor, reproducibility is at risk, which can have serious implications ranging from financial to core credibility.

At the 2019 *RStudio Conference* in Austin Texas, Dr. Garrett Grolemund, Data Scientist and Lead Educator at RStudio, presented a talk titled, *"R Markdown: The bigger picture"*, which details a compelling argument about why it is critical for data science to be reproducible and how the R statistical programming language leverages an open source package ecosystem to support those reproducibility ideals:

> *Statistics has made science resemble math, so much so that we've begun to conflate p-values with mathematical proofs. We need to return to evaluating a scientific discovery by its reproducibility, which will require a change in how we report scientific results. This change will be a windfall to commercial data scientists because reproducible means repeatable, automatable, parameterizable, and schedulable.* (Grolemund 2019)

The vibrant R community that exists in clinical data science and health care industries has been leading the way when it comes to creative thinking about reproducibility in Shiny applicaitons. At the 2019 *R in Pharma* conferenece held at Harvard University, Chief Technology Officer and creator of the `shiny` R paackage, Joe Cheng, debuted a workshop on `shinymeta`, a new package developed by RStudio with inspiration from thought leaders in the Pharma space.

The `shinymeta` R package provides tools for capturing logic in a Shiny app and exposing it as code that can be run outside of Shiny (e.g., from an R console). It also provides tools for bundling both the code and results to the end user (RStudio 2020c).

While `shinymeta` falls outside the scope of this paper and talk, we hope that you will explore this package as it relates to creating reproducible code derived from Shiny applications. The package documentation website for `shinymeta` is available here: `rstudio.github.io/shinymeta/`, and recorded talks on this package and usecases will soon be made available from *RStudio Conference 2020*.

Acknowledgements

This paper has drawn on packages, vignettes, and work produced by many individuals employed by RStudio, a Public Benefit Corporation. RStudio documentation is written and maintained by groups of individuals over time, including members of various engineering teams, including the team I am affiliated with, RStudio Solutions Engineering.

Contact Information

Comments and questions can be directed to the following individuals at RStudio, PBC:

Phil Bowsher, Director of Life Sciences & Healthcare
`phil@rstudio.com`
Web: `linkedin.com/in/philip-bowsher-67151015/`

Kelly O'Briant, Solutions Engineer
`kelly@rstudio.com`
Web: `solutions.rstudio.com`

References

Grolemund, Garrett. 2019. "R Markdown: The Bigger Picture." RStudio, PBC. https://resources.rstudio.com/rstudio-conf-2019/r-markdown-the-bigger-picture.

O'Briant, Kelly. 2019. "A Template for Setting up Shinytest with Travis Ci and Renv." RStudio, PBC. https://github.com/kellobri/shinytest-travis-ci.

RStudio. 2020a. "Reproducible Environments: Manage Environments for Data Science." RStudio, PBC. https://environments.rstudio.com/.

———. 2020b. "Shiny Documentation." RStudio, PBC. https://shiny.rstudio.com/.

———. 2020c. "Shinymeta Documentation." RStudio, PBC. https://rstudio.github.io/shinymeta/.

———. 2020d. "Shinytest Documentation." RStudio, PBC. https://rstudio.github.io/shinytest/index.html.

Ushey, Kevin. 2019. "Renv: Project Environments for R." RStudio, PBC. https://blog.rstudio.com/2019/11/06/renv-project-environments-for-r/.

Wickham, Hadley. 2020. *Mastery Shiny*. 1st ed. O'Reilly Media. https://mastering-shiny.org/.

Wolf, Sebastian. 2019. "Self-Service Statistics with Shiny." RStudio, PBC. https://resources.rstudio.com/webinars/the-role-of-r-in-drug-discovery-research-and-development.