shinymgr: A Framework for Building, Managing, and Stitching Shiny Modules into Reproducible Workflows

by Laurence A. Clarfeld, Caroline Tang, and Therese Donovan

Abstract The R package shinymgr provides a unifying framework that allows Shiny developers to create, manage, and deploy a master Shiny application comprised of one or more "apps", where an "app" is a tab-based workflow that guides end-users through a step-by-step analysis. Each tab in a given "app" consists of one or more Shiny modules. The shinymgr app builder allows developers to "stitch" Shiny modules together so that outputs from one module serve as inputs to the next, creating an analysis pipeline that is easy to implement and maintain. Apps developed using shinymgr can be incorporated into R packages or deployed on a server, where they are accessible to end-users. Users of shinymgr apps can save analyses as an RDS file that fully reproduces the analytic steps and can be ingested into an RMarkdown or Quarto report for rapid reporting. In short, developers use the shinymgr framework to write Shiny modules and seamlessly combine them into Shiny apps, and end-users of these apps can execute reproducible analyses that can be incorporated into reports for rapid dissemination. A comprehensive overview of the package is provided by 12 learnr tutorials.

1 Introduction

The **shiny** R package allows users to build interactive web apps straight from R, without advanced knowledge of HTML or JavaScript (Chang et al., 2022). A *shiny* web app can permit an expedient analysis pipeline or workflow. Ideally, the pipeline can produce outputs that are fully reproducible (Peng, 2011; Gentleman and Lang, 2007; Alston and Rick, 2021). Moreover, the pipeline can permit rapid reporting to convey the results of an analysis workflow to a target audience (Stoudt et al., 2021) (Figure 1).

shiny applications range from simple to complex, each with an intended purpose developed for an intended user audience. Several R packages provide a development framework for building multi-faceted master applications, including **shinipsum** for prototyping (Fay and Rochette, 2020), **golem** (Fay et al., 2021), and **rhino** (Żyła et al., 2023).

From the developer's perspective, complex *shiny* applications can result in many lines of code, creating challenges for collaborating, debugging, streamlining, and maintaining the overall product. *shiny* modules are a solution to this problem. As stated by Winston Chang (shi, 2020), "A *shiny* module is a piece of a *shiny* app. It can't be directly run, as a *shiny* app can. Instead, it is included as part of a larger app . . . Once created, a *shiny* module can be easily reused – whether across different apps, or multiple times in a single app." *shiny* modules, and modularization in general, are a core element of agile software development practices (Larman, 2004). Several authors have contributed R packages for distributing pre-written *shiny* modules for general use, including the **datamods** (Perrier et al., 2022), **shiny.reglog** (Kosinski, 2022), **periscope** (Brett and Neuhaus, 2022), **shinyauthr** (Campbell, 2021), and

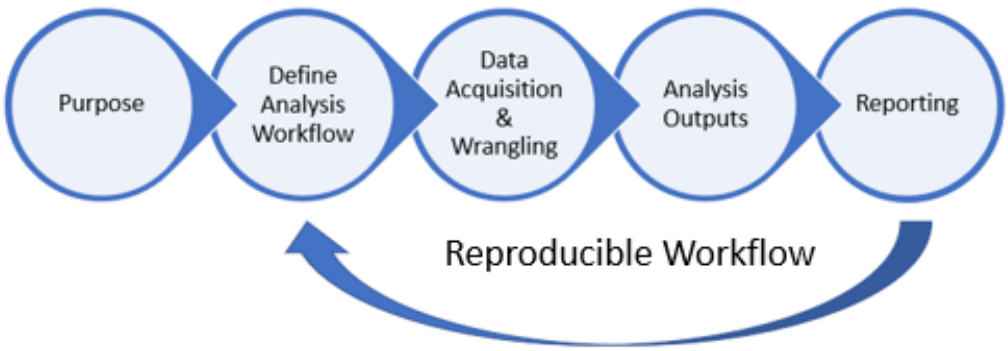

Figure 1: Stages of a reproducible workflow, a process that moves an inquiry from raw data to insightful contribution.

jsmodule (Kim and Lee, 2022) packages.

However, as the number of available modules increases, there is a pressing need for documenting available *shiny* modules and easily incorporating them into new workflows. For example, consider a toy modular-based app that guides a user through an analysis of the famous "Iris Dataset," which contains 150 records of 3 species of iris, including measurements of the length and width of the flowers' sepals and petals (Fisher, 1936). The app, called "Iris Explorer," consists of 5 tabs to be worked through in sequence (Figure 2, top).

Tab 1 displays instructions for use, while tab 2 performs a k-means clustering of the data, where k is specified by the user. The resulting clusters are displayed with two variables of the user's choosing as depicted in Figure 2. In tab 3, the user will choose a value n, indicating the number of rows by which to randomly subset the data, and in tab 4 the user selects a single variable to be plotted as a bar chart. Finally, in tab 5 the user can save their outputs as an RDS file. This contrived example includes some key elements of a typical workflow in that the five tabs introduce a dataset, guide the user through light data wrangling, produce analysis outputs, and offer the ability to save the results.

The app's blueprint (Figure 2, bottom) identifies the *shiny* modules in each tab, showing how outputs from one module can serve as inputs to the next. Note that while this example shows a single module in each tab with differing inputs/outputs, in the general case tabs can contain an arbitrary number of *shiny* modules (including multiple instances of the same module) and each module can have multiple inputs/outputs.

While two of the *shiny* modules within the "iris_explorer" app pertain to the iris dataset specifically ("iris_intro" and "iris_cluster"), the remaining *shiny* modules ("subset_rows", "single_column_plot", and "save") may be incorporated into other apps.

Figure 2: Top: The "iris_explorer" app guides a user through an analysis of the iris dataset in a tab-based sequence. Bottom: A blueprint of the "iris_explorer" app shows the 5 tabs, each containing a single module identified by name within blue ovals. Some of the shiny modules require inputs and generate outputs as identified in gray polygons.

Developers who utilize the same *shiny* modules within different apps will naturally be faced with several questions:

1. Which *shiny* modules have been written? Are they well documented with unit testing?
2. What are the module's inputs (arguments) and outputs (returns)?
3. Where are the *shiny* modules stored?
4. How can *shiny* modules be combined into a cohesive, well-documented app?
5. How can production-ready apps be deployed for end-users?

Users of an app created with the *shinymgr* framework may wish to know:

6. Can analysis outputs be saved as a fully reproducible workflow?
7. Can outputs be ingested into a *Rmarkdown* or *Quarto* template for rapid reporting?

1.1 Introducing *shinymgr*

The R package, **shinymgr**, was developed to meet these challenges (Clarfeld et al., 2024). The *shinymgr* package includes a general framework that allows developers to create *shiny* modules, stitch them together as individual "apps" that are embedded within the master *shiny* application, and then deploy them on a *shiny* server or incorporate them into R packages. *shinymgr* was motivated from our first-hand experience in our work building tools that assist scientists in remote wildlife monitoring with the R package *AMMonitor* (Balantic and Donovan, 2020). Dependencies of *shinymgr* include the packages **DBI** (R Special Interest Group on Databases (R-SIG-DB) et al., 2022), **reactable** (Lin, 2022), **RSQLite** (Müller et al., 2022), **renv** (Ushey, 2023), **shiny** (Chang et al., 2022), **shinyjs** (Attali, 2021), and **shinydashboard** (Chang and Borges Ribeiro, 2021).

From the developer's perspective, an "app" consists of an ordered set of tabs, each of which contain specified *shiny* modules. *shiny* modules are the basic element in the *shinymgr* framework; they can be used and re-used across different tabs and different apps. Information about each module and app is stored in a SQLite database (Hipp, 2020). The *shinymgr* app builder "stitches" *shiny* modules together so that outputs from one module serve as inputs to the next, creating an analysis pipeline that is easy to implement and maintain. When apps are production-ready , developers can deploy a stand-alone *shiny* application independent of *shinymgr* on a server or within an R package. From the end-user's perspective, an "app" created with the *shinymgr* framework consists of an ordered series of *shiny* tabs, establishing an analysis. Users can save their inputs and outputs as an RDS file to ensure full reproducibility. Furthermore, the RDS file may be loaded into an R Markdown (Rmd) or Quarto (qmd) template for rapid reporting. We are unaware of existing packages that unify the elements of modularization, documentation, reproducibility, and reporting in a single framework.

We introduce *shinymgr* in sections 2-4 below. In section 2 we describe how developers can create apps using the *shinymgr* framework. In section 3 we describe how developers can deploy a *shinymgr* project on a local machine, server, or within an R package. In section 4 describes the end-user experience, where end-users execute an "app" and store results for reproducibility and reporting. The package tutorials and cheat sheet are described in section 5. The *shinymgr* package comes with a series of **learnr** (Schloerke et al., 2020) tutorials described at the end of the paper.

2 Developing *shinymgr* apps

2.1 Setting up *shinymgr*

The canonical home of *shinymgr* is `https://code.usgs.gov/vtcfwru/shinymgr/` where *shinymgr* users may post merge requests and bug fix requests. *shinymgr* may also be downloaded from CRAN.

```
install.packages("shinymgr")
```

The development version can be downloaded with:

```
remotes::install_gitlab(
  repo = "vtcfwru/shinymgr",
  auth_token = Sys.getenv("GITLAB_PAT"),
  host = "code.usgs.gov",
  build_vignettes = FALSE)
```

Once installed, a new *shinymgr* project can be created within a parent directory:

```
# set the directory path that will house the shinymgr project
parentPath <- getwd()

# set up raw directories and fresh database
shinymgr_setup(
  parentPath = parentPath,
  demo = TRUE)
```

The `shinymgr_setup()` function produces the following directory structure within the primary "shinymgr" directory. This structure consists of 3 files that make up the "master" app (global.R, server.R, and ui.R), and 9 directories. If the argument demo is set to FALSE, these directories will be largely empty, except for the "modules_mgr" and "database" directories, which will contain *shiny* modules for rendering *shinymgr*'s UI and an empty SQLite database, respectively. If the argument demo is set to TRUE, each directory will include several demo files as shown, including a pre-populated database. Here, we highlight a subset of the demo files related to the "iris_explorer" app to guide developers through the key elements of *shinymgr* (additional demo files come with package but are omitted here for clarity).

```
shinymgr
+-- analyses
|   \-- iris_explorer_Gandalf_2023_06_05_16_30.RDS
+-- data
|   \-- iris.RData
+-- database
|   \-- shinymgr.sqlite
+-- global.R
+-- modules
|   +-- iris_cluster.R
|   +-- iris_intro.R
|   +-- single_column_plot.R
|   \-- subset_rows.R
+-- modules_app
|   \-- iris_explorer.R
+-- modules_mgr
|   +-- add_app.R
|   +-- add_mod.R
|   +-- add_report.R
|   +-- add_tab.R
|   +-- app_builder.R
|   +-- my_db.R
|   +-- new_analysis.R
|   +-- new_report.R
|   +-- queries.R
|   +-- save_analysis.R
|   +-- stitch_script.R
|   \-- table.R
+-- reports
|   \-- iris_explorer
|       \-- iris_explorer_report.Rmd
+-- server.R
+-- tests
|   +-- shinytest
|   |   +-- test-iris_explorer-expected
|   |   |   +-- 001.json
|   |   |   +-- 001.png
|   |   |   +-- 002.json
|   |   |   \-- 002.png
|   |   \-- test-iris_explorer.R
|   +-- shinytest.R
|   +-- testthat
|   |   +-- test-iris_cluster.R
|   |   \-- test-subset_rows.R
|   \-- testthat.R
+-- ui.R
```

```
\-- www
    +-- dark_mode.css
    \-- shinymgr-hexsticker.png
```

The directory structure produced by `shinymgr_setup()` includes the following:

- The **analyses** directory provides the developer an example of a previously run analysis that was created using the *shinymgr* framework (an RDS file). An analysis file name includes the app name (e.g. "iris_explorer"), the name of the person who ran the analysis (e.g. "Gandalf"), and the date and time of the analysis (e.g., "iris_explorer_Gandalf_2023_06_05_16_30.RDS").
- The **data** directory stores RData files that can be used by various *shinymgr* apps (e.g., "iris.RData").
- The **database** directory stores the *shinymgr* SQLite database, named "shinymgr.sqlite." The database is used by the developer to track all *shiny* modules, their arguments (inputs), returns (outputs), and how they are combined into *shinymgr* apps.
- The **modules** directory stores stand-alone *shiny* modules. These files are largely written by the developer with the help of the `mod_init()` function, and are registered in the database with the `mod_register()` function. Four of the example *shiny* modules listed are used in the "iris_explorer" app.
- The **modules_app** directory stores *shiny* modules that are *shinymgr* "apps" – the stitching together of *shiny* modules into a tab-based layout that provides an analysis workflow (Figure 2 shows the "iris_explorer" app layout). Files within the "modules_app" directory are not written by hand - instead, they are created with the *shinymgr* "app builder."
- The **modules_mgr** directory stores *shiny* modules that build the overall *shinymgr* framework.
- The **reports** directory provides an example of an *RMarkdown* (Rmd) template (e.g., "iris_explorer_report.Rmd"), allowing for rapid reporting by an end-user.
- The **tests** directory stores both **testthat** (Wickham, 2011) and **shinytest** (Chang et al., 2021) code testing scripts.
- The **www** directory stores images that may be used by a *shiny* app.
- In addition to these directories, three files are created for launching the master *shinymgr shiny* application:

  1. **ui.R** - This file contains code to set the user interface for the master *shinymgr* app.
  2. **server.R** - The master server file.
  3. **global.R** - The global.R file is sourced into the server.R file at start-up. It sources all of the *shiny* modules within the *shinymgr* framework so they are available when *shinymgr* is launched.

2.2 The *shinymgr* developer's portal

Once set-up is complete, the `launch_shinymgr()` function will launch the *shinymgr* "Developer's Portal" UI, allowing developers to create and test new *shinymgr* apps.

```
# launch shinymgr
launch_shinymgr(shinyMgrPath = paste0(parentPath, "/shinymgr"))
```

The portal is recognizable by the *shinymgr* logo in the upper left corner (Figure 3). The portal consists of three main tabs in the left menu. The "Developer Tools" tab is used to create apps, view the *shinymgr* database, and register reports, while the "Analysis (beta)" and "Reports (beta)" tabs allow developers to evaluate apps from the user's perspective.

The "Developer Tools" section includes 4 tabs for app development: The "Build App" tab allows the developer to create new *shinymgr* apps from existing modules using the *shinymgr* app builder; the "Database" tab displays the *shinymgr* database tables, the "Queries" tab contains a set of standard database queries, and the "Add Reports" tab allows the developer to link a report (Rmd or qmd) to a given *shinymgr* app (Figure 3), as described below.

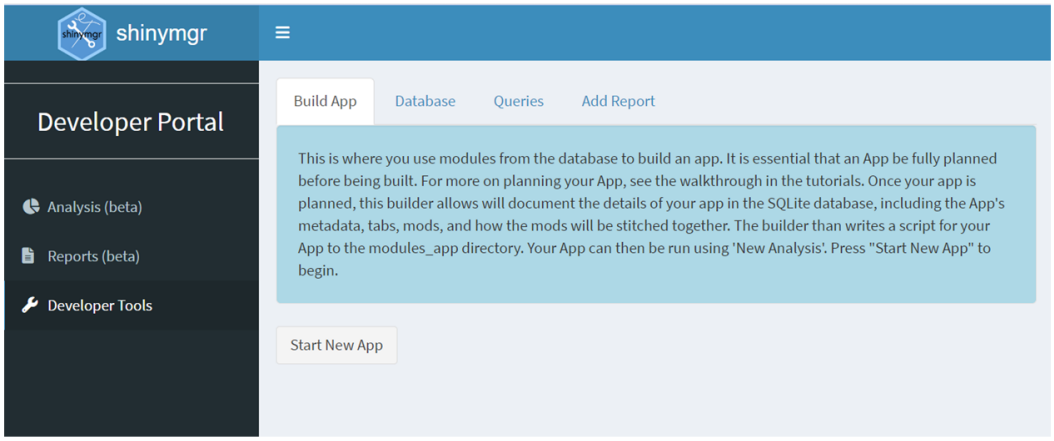

Figure 3: The shinymgr Developer Portal consists of a sidebar panel where developers can create new shiny modules and new apps, and test-drive analyses and reports from the user's perspective. The main panel shows the "Build App" tab within the "Developer Tools" section.

2.3 The *shinymgr* database

The *shinymgr* SQLite database ("shinymgr.sqlite") is a single file created by the `shinymgr_setup()` function. The database tracks all *shiny* modules, their arguments (inputs), returns (outputs), their package dependencies and version numbers, how they are combined into an "app," and any reports that are associated with apps. The database tables are populated via dedicated *shinymgr* functions.

The *shinymgr* database consists of 11 tables in total (Figure 4). These tables are connected to each other as a typical relational database, with primary keys establishing unique records in each table, and foreign keys that reference primary keys in other tables (see Appendix A for a full database schema and the "database" *learnr* tutorial for additional information).

The "apps," "appReports," "reports," "appTabs," and "tabs" tables largely store information on what a user would see when they run an analysis. The table "apps" stores information about apps such as "iris_explorer." Apps consist of tabs, which are listed in the "tabs" table. Tabs are linked to apps via the "appTabs" table. The table "reports" lists any Rmd or qmd files that serve as a report template, and the table "appReports" links a specific report with a specific app.

Four of the 11 database tables focus on modules, highlighting that *shiny* modules are basic building blocks of any *shinymgr* app. Developers create new *shiny* modules with the `mod_init()` function, which copies a *shinymgr* module template (an R file template) that includes a header with key-value that describe the module, including the module name, display name, description, citation, notes, and module arguments and returns (if any). For example, the header of the iris_cluster module is:

```
#!! ModName = iris_cluster
#!! ModDisplayName = Iris K-Means Clustering
#!! ModDescription = Clusters iris data based on 2 attributes
#!! ModCitation = Baggins, Bilbo.  (2023). iris_cluster. [Source code].
```

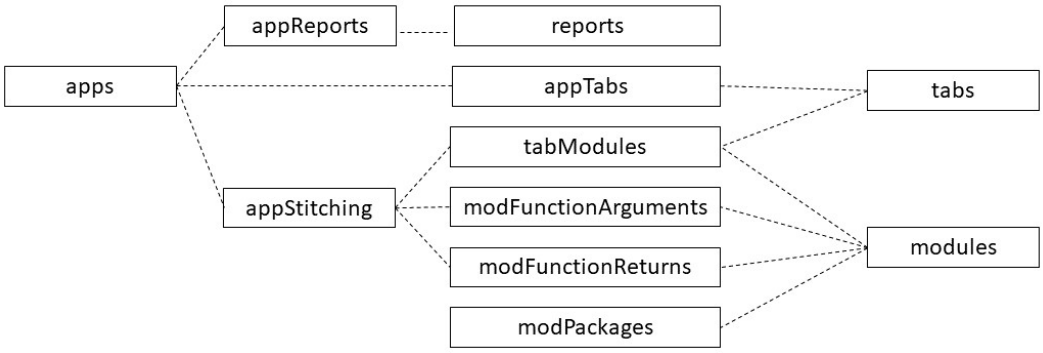

Figure 4: The 11 tables of the shinymgr SQLite database. Lines indicate how the tables are related to each other.

```
#!! ModNotes = Demo module for the shinymgr package.
#!! ModActive = 1
#!! FunctionReturn = returndf !! selected attributes and their assigned clusters !! data.frame
```

The module code is written beneath the header (see Appendix B for an example). Function calls within the module code should be written with `package::function()` notation, making explicit any R package dependencies. Once the module is completed, unit tests can written and stored in the *shinymgr* project's "tests" directory. The final module file is saved to the "modules" directory and registered into the database with the `mod_register()` function. The `mod_register()` function populates the modules, "modFunctionArguments", and "modFunctionReturns" SQLite database tables. Further, it uses the *renv* package to identify any package dependencies and inserts them into the modPackages table. Readers are referred to the "modules" "tests", and "shinymgr_modules" *learnr* tutorials that come with the *shinymgr* package for more details.

Once modules are registered in the database, the developer can incorporate them into new apps. As *shiny* modules and apps in the database represent files that contain their scripts, deleting a module or an app from the database will delete all downstream database entries as well as (optionally) the actual files themselves. Deletion of a module will fail if it is being used in other apps. Module updates can be versioned by creating a new module and then referencing its precursor in the "modules" database table.

2.4 The *shinymgr* app builder

Once developers create and register their own stand-alone *shiny* modules, apps are generated with *shinymgr*'s app builder (Figure 5).

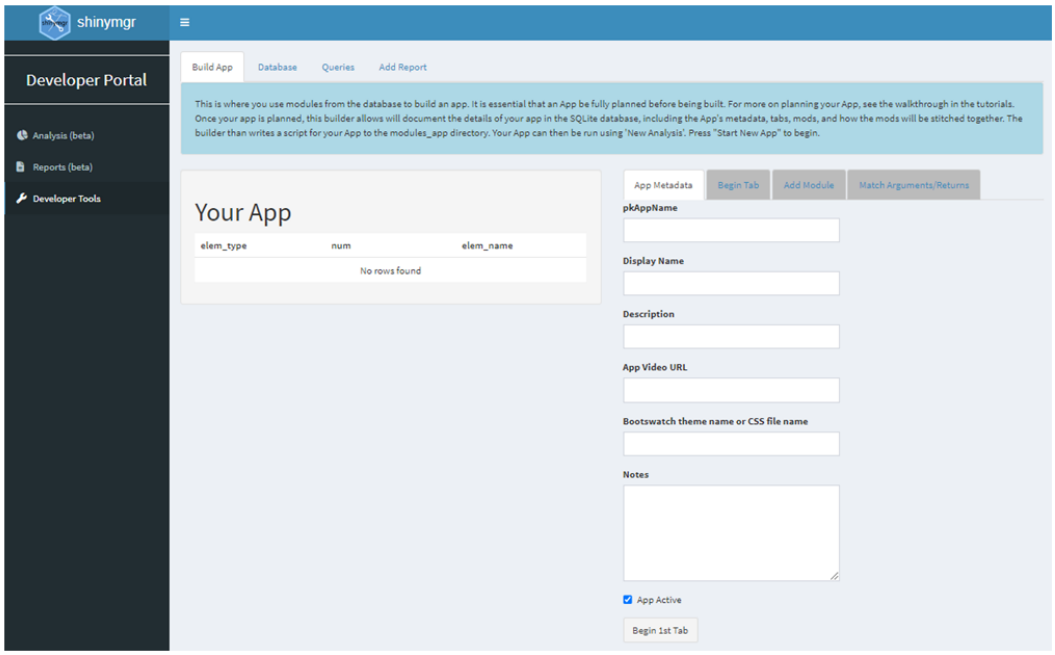

Figure 5: The shinymgr Developer Portal layout, showing the app builder in the Developer Tools.

Developers are guided through a process where they design their app from *shiny* modules they have registered. The builder then populates the *shinymgr* database with instructions on how to construct the app and writes the app's script based on those instructions. The newly created script is saved to the "modules_app" directory. Through this structured process, apps produced by the builder are well-documented and generate highly reproducible analyses. Readers are encouraged to peruse the tutorial, "apps", for more information.

The `qry_app_flow()` function will query the database to return a list of the *shiny* modules and tabs included in a specified app, such as "iris_explorer":

```
# look at the appTabs table in the database
qry_app_flow("iris_explorer", shinyMgrPath = paste0(getwd(),"/shinymgr"))

      fkAppName      fkTabName tabOrder       fkModuleName modOrder
```

```
1 iris_explorer       IE_intro        1         iris_intro        1
2 iris_explorer   IE_iris_data        2       iris_cluster        1
3 iris_explorer IE_subset_rows        3        subset_rows        1
4 iris_explorer   IE_plot_data        4 single_column_plot        1
```

As shown in Figure 2, this app has 5 tabs, and each tab features a single module. The "Save" tab is the final tab in all *shinymgr* apps and is not listed in the query result.

Developers can "beta test" apps prior to deployment by selecting the Analysis (beta) tab in the Developer's Portal (Figure 3). They can also create *RMarkdown* or *Quarto* report templates that accept the outputs from an analysis and incorporate them into a report. Report metadata are logged in the "reports" table of the database, and then linked with a specific app in the "appReports" table. An end-user will run an analysis and render a report, a process described more fully in the "Using *shinymgr* Apps" section below.

To summarize this section, developers use the `shinymgr_setup()` function to create the directory structure and underlying database needed to build and run *shiny* apps with *shinymgr*. Developers use the `mod_init()` and `mod_register()` functions to create modules and make them available for inclusion in new apps built with the *shinymgr* app builder. A developer can create as many *shinymgr* projects as needed. In each case, the *shinymgr* project is simply a fixed directory structure with three R files (ui.R, server.R, and global.R), and a series of subdirectories that contain the apps and *shiny* modules created by the developer, along with a database for tracking everything.

3 Deploying *shinymgr* projects

Once development is completed, developers can deploy their *shinymgr* project on a server or within an R package by copying portions of the *shinymgr* project to a new location while retaining the original project for future development. Once deployed, a *shinymgr* project no longer requires the *shinymgr* package or database to be run. Thus, the files and directories to be copied for deployment include only:

```
shinymgr
+-- data
+-- global.R
+-- modules
+-- modules_app
+-- modules_mgr
+-- reports
+-- server.R
+-- ui.R
\-- www
```

The master app files, ui.R, global.R, and server.R, are needed to run the *shinymgr* framework.

When deploying a *shinymgr* project within an R package, objects within the data folder should be copied into the package's "data" folder. The remaining files should be copied into a directory within the package's "inst" folder that will house the master *shiny* application. Deployment on a server such as shinyapps.io will require similar adjustments.

After files are copied to the correct location, a few key adjustments are needed. First, the "modules_app" directory should contain only those apps (and dependent modules and reports) that can be used by end-users; unused apps, modules, and reports can be deleted. Second, the new.analysis.R script within the modules_mgr folder will require minor updates to remove dependencies on the *shinymgr* database. Third, the ui.R and server.R scripts should be updated to no longer showcase *shinymgr* and the Developer's Portal; rather, it should be customized by the developer to create their own purpose-driven apps. For example, Figure 6 shows a hypothetical deployment of the master app titled "Deployed Project" that is based on the *shinymgr* framework. Notice the absence of the Developer Tools tab and the absence of references to *shinymgr*. The "deployment" *learnr* tutorial provides more in-depth discussion.

To summarize this section, deploying the *shinymgr* framework involves copying key elements of the *shinymgr* developer project into package or server directories, updated as needed for use by end-users. Readers are referred to the "deployment" tutorial for further information.

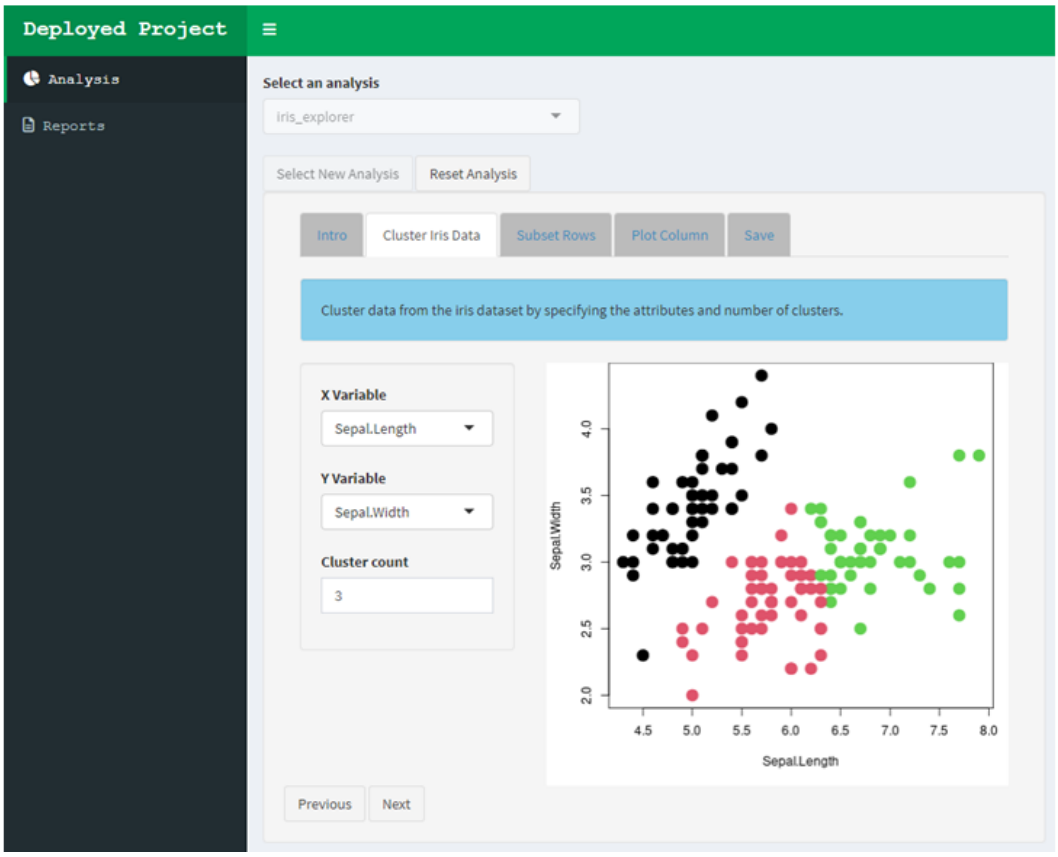

Figure 6: An example of a deployed shinymgr app. The deployed version excludes the Developers Tools tab and is an example of what the end user sees when using a deployed app.

4 Using *shinymgr* apps

Apps built with *shinymgr* can appeal to various types of end-users. When deployed as part of an R package, end-users would be anyone who uses that package. Apps may also be distributed as stand-alone scripts, or hosted on a server, as described above. Developers may also use *shinymgr* to produce apps for their own use (i.e., the developer *is* the end-user). Regardless of who the intended end-user is, this section discusses that user's experience after the master app is deployed.

Whoever the intended audience for the app, this section discusses how an app can be used *after* it has been deployed.

4.1 Reproducible analyses

The final tab in any *shinymgr* app provides the opportunity to save the analysis itself. Reproducibility is a core tenet of *shinymgr*. Therefore, a robust set of metadata are saved as an RDS file to allow a user to understand and replicate their results. An example of a completed analysis is the file, "iris_explorer_Gandalf_2023_06_05_16_30.RDS," which stores a user's analytic steps for a run of the "iris explorer" app. The code below reads in this example file, and shows the structure (a list with 23 elements):

```
rds_filepath <- paste0(getwd(),"/shinymgr/analyses/iris_explorer_Gandalf_2023_06_05_16_30.RDS")
old_analysis <- readRDS(rds_filepath)
str(old_analysis, max.level = 2, nchar.max = 20, vec.len = 15)

List of 23
 $ analysisName                    : chr "iri"| __truncated__
 $ app                             : chr "iris_explorer"
 $ username                        : chr "Gandalf"
 $ mod2-clusters                   : int 3
 $ mod2-xcol                       : chr "Sepal.Length"
```

```
$ mod2-ycol                            : chr "Petal.Length"
$ mod3-full_table__reactable__pageSize  : int 10
$ mod3-resample                        : 'shinyActionButtonValue' int 1
$ mod3-full_table__reactable__pages     : int 15
$ mod3-subset_table__reactable__page    : int 1
$ mod3-full_table__reactable__page      : int 1
$ mod3-sample_num                      : int 20
$ mod3-subset_table__reactable__pages   : int 2
$ mod3-subset_table__reactable__pageSize: int 10
$ returns                              :List of 3
 ..$ data1:List of 1
 ..$ data2:List of 1
 ..$ data3:List of 2
$ notes                                : chr "Thi"| __truncated__
$ timestamp                            : POSIXct[1:1], format: "202"| __truncated__
$ metadata                             :List of 6
 ..$ appDescription: chr "Clu"| __truncated__
 ..$ mod1          :List of 7
 ..$ mod2          :List of 7
 ..$ mod3          :List of 7
 ..$ mod4          :List of 7
 ..$ lockfile      :List of 2
$ app_code                             : chr "# T"| __truncated__
$ iris_intro_code                      : chr "#!!"| __truncated__
$ iris_cluster_code                    : chr "#!!"| __truncated__
$ subset_rows_code                     : chr "#!!"| __truncated__
$ single_column_plot_code              : chr "#!!"| __truncated__
```

The list stores a great deal of information:

- **analysisName** is the name of the analysis and is equivalent to the filename of the RDS file (without the extension)
- **app** is the name of the app that produced the saved analysis results.
- **username** was entered in the "Save" tab when the analysis was performed.
- **mod#-value** indicate the values of each *shiny* module's arguments (inputs), if any exist, at the time the analysis was saved.
- **returns** includes values of all outputs (returns) of each module.
- **notes** were entered in the "Save" tab when the analysis was performed.
- **timestamp** is the date/time when the analysis was saved.
- **metadata** includes robust information about each module, including the app description and the description of each module as it was originally stored in the *shinymgr* database tables. The metadata list element also includes an *renv* "lockfile": a list that describes the R version and R package dependencies (including *shinymgr*) used by the app itself. The lockfile captures the state of the app's package dependencies at the time of its creation; in the case of *shinymgr*, it contains the dependencies used by the developer who created the app. Each lockfile record includes the name and version of the package and their installation source.
- ***_code** attributes with this format contain the source code for the app.

The code list element allows an end user to revisit the full analysis with *shinymgr*'s `rerun_analysis()` function, supplying the file path to a saved *shinymgr* analysis (RDS file).

```
rerun_analysis(analysis_path = rds_filepath)
```

The `rerun_analysis()` function will launch a *shiny* app with two tabs (Figure 7); it can only be run during an interactive R session, with no other *shiny* apps running.

The first tab is called "The App", and will be visible when the `rerun_analysis()` function is called. It contains a header with the app's name, a subheading of "Analysis Rerun," and a fully functioning, identical copy of the *shiny* app used to generate the saved analysis. Below that, a disclaimer appears, indicating the app was produced from a saved analysis. A summary of the analysis is presented on the second tab that displays the values used to produce the given analysis output.

If the `rerun_analysis()` function fails, it could be due to a change in R and package versions currently installed on the end-user's machine. To that end, the lockfile that is included in the metadata section of the RDS file can be used to restore the necessary R packages and R version with the `restore_analysis()` function. This function will attempt to create a self-contained *renv* R project that

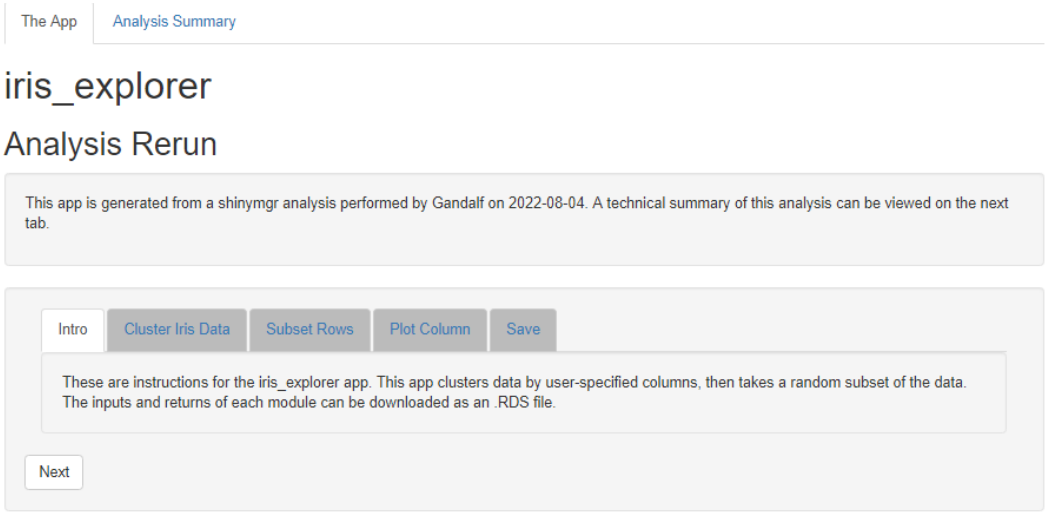

Figure 7: A screenshot of the rerun_analysis() function, as called on the saved analysis from the iris_explorer app (RDS file). The active tab, called "The App", allows a user to rerun a previously executed analysis. The "Analysis Summary" tab displays the values of all module arguments and returns, captured when the analysis was saved, along with a detailed description of the app, it's modules, the App's source code, and all package dependencies.

includes all of the packages and the R version used by the developer when the app was created. The analysis RDS is added to this new project, where the `rerun_analysis()` function can be attempted again. Readers are referred to the "analyses" tutorial for further information.

4.2 Rapid reporting

Another important feature of *shinymgr* is the ability to share results of an analysis with others in a friendly, readable format with *RMarkdown* or *Quarto*. Apps produce an RDS file, which may be passed into an Rmd or qmd file as a parameterized input. For example, the demo database includes a report template called "iris_explorer_report.Rmd." This file, with code shown below, allows users to navigate to the RDS file produced by the "iris explorer" app and render the rapid report.

```
---
title: 'Annual Report for Iris Explorer'
output: html_document
params:
  user:
    label: "User"
    value: "Bilbo"
    placeholder: "Enter user name"
  year:
    label: "Year"
    value: 2017
    input: slider
    min: 2010
    max: 2018
    step: 1
    sep: ""
  file:
   input: file
   label: "Choose RDS"
   value: ""
   multiple: FALSE
   buttonLabel: "Browse to analysis output..."
---

```{r setup, include=FALSE}
```
```

```
knitr::opts_chunk$set(echo = FALSE)
library(knitr)
ps <- readRDS(params$file)
```

This report summarizes an analysis of iris data by
`r params$user` conducted in `r params$year`. Iris
data was clustered into `r ps$'mod2-clusters'` groups
based on `r ps$'mod2-xcol'` and `r ps$'mod2-ycol'`.
A random sample of `r ps$'mod3-sample_num'` records
were collected, with sample sizes shown in the pie
chart below:

```{r}
pie_data <- table(ps$returns$data2$subset_data$cluster)
pie(
  x = pie_data,
  labels = as.character(pie_data),
  col = rainbow(length(pie_data)),
  main = "Number of random samples by cluster"
)
legend(
  x = "topright",
  legend = names(pie_data),
  fill = rainbow(length(pie_data))
)

```

Some things to note about this analysis are: `r ps$notes`

Respectfully submitted,

Gandalf

Reports may be run within the deployed version of *shinymgr* (e.g., left menu of Figure 6), or may be run directly in R by opening the Rmd file and navigating to the RDS as a file input. Users who run a report can download it to their local machine as a HTML, PDF, or Word file, where they can further customize the output.

To summarize this section, users of *shinymgr* "apps" created with the *shinymgr* framework are presented with a series of *shiny* tabs that establish an analysis workflow. Users can save their inputs and outputs as an RDS file to ensure full reproducibility. Further, the RDS file may be loaded into an R Markdown (Rmd) or Quarto (qmd) template for rapid reporting.

5 Tutorials and cheatsheet

with the package. Below is a list of current tutorials, intended to be worked through in order:

```
Available tutorials:
* shinymgr
  - intro            : "shinymgr-01: Introduction"
  - shiny            : "shinymgr-02: Shiny"
  - modules          : "shinymgr-03: Modules"
  - app_modules      : "shinymgr-04: App modules"
  - tests            : "shinymgr-05: Tests"
  - shinymgr         : "shinymgr-06: shinymgr"
  - database         : "shinymgr-07: Database"
  - shinymgr_modules : "shinymgr-08: shinymgr_modules "
  - apps             : "shinymgr-09: Apps"
  - analyses         : "shinymgr-10: Analyses"
  - reports          : "shinymgr-11: Reports"
  - deployment       : "shinymgr-12: Deployment"
```

The "intro" tutorial gives a general overview. Tutorials 2-5 are aimed at developers who are new to *shiny*, while tutorials 6 – 12 focus on the *shinymgr* package.

Launch a tutorial with the *learnr* `run_tutorial()` function, providing the name of the module to launch. The tutorial should launch in a browser, which has the benefit of being able to print the tutorial to PDF upon completion:

```
learnr::run_tutorial(
  name = "modules",
  package = "shinymgr")
```

Additionally, the package cheatsheet can be found with:

```
browseURL(paste0(find.package("shinymgr"), "/extdata/shinymgr_cheatsheet.pdf"))
```

Contributions are welcome from the community. Questions can be asked on the issues page at https://code.usgs.gov/vtcfwru/shinymgr/issues.

6 Acknowledgments

We thank Cathleen Balantic and Jim Hines for feedback on the overall package and package tutorials. *shinymgr* was prototyped by Therese Donovan at a *shiny* workshop taught by Chris Dorich and Matthew Ross at Colorado State University in 2020 (pre-pandemic). We thank the instructors for feedback and initial coding assistance. Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government. The Vermont Cooperative Fish and Wildlife Research Unit is jointly supported by the U.S. Geological Survey, University of Vermont, Vermont Fish and Wildlife Department, and Wildlife Management Institute.

7 Bibliography

8 Appendix A

Entity relationship diagram for the *shinymgr* database, which tracks all components of an apps and modules (Figure 8). The database consists of 11 tables. Primary keys are referenced with a "pk" prefix, while foreign keys are referenced with an "fk" prefix. A full description of the database is contained in the "database" *learnr* tutorial that comes with the *shinymgr* package

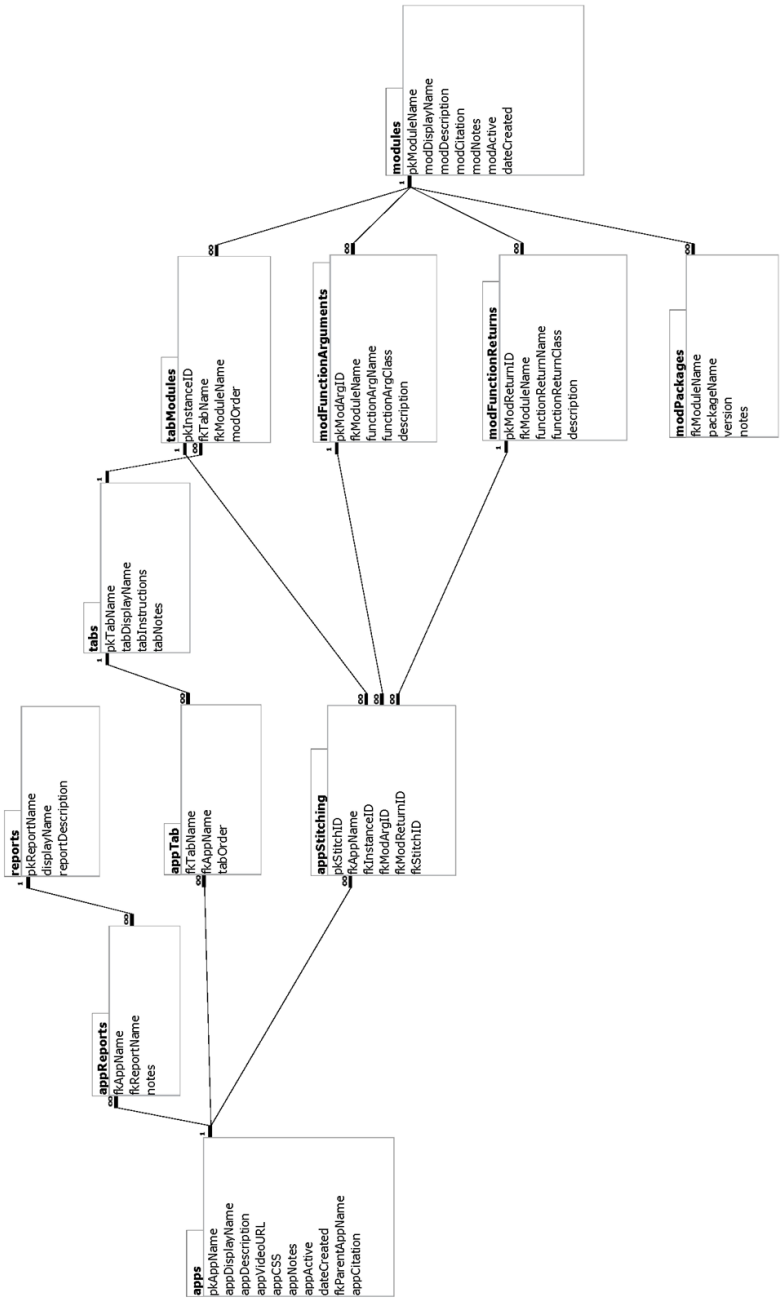

Figure 8: Entity relationship diagram for the shinymgr database, which tracks all components of an apps and modules. The database consists of 11 tables. Primary keys are referenced with a "pk" prefix, while foreign keys are referenced with an "fk" prefix. A full description of the database is contained in the "database" learnr tutorial that comes with the shinymgr package.

9 Appendix B

Modules in *shinymgr* are written by developers for their own purposes. The `shinymgr::mod_init()` function creates a template for module development. The header is a series of key-value pairs that the developer fills out (typically after the module code is written and tested). The "iris_cluster" module is presented below as an example. The module consists of two paired functions: here, `iris_cluster_ui(id)` and `iris_cluster_server()`. The UI is a function with an argument called id, which is turned into module's "namespace" with the `NS()` function. A namespace is simply the module's identifier and ensures that function and object names within a given module do not conflict with function and object names in other modules. The Id's for each input and output in the UI must be wrapped in a `ns()` function call to make explicit that these inputs are assigned to the module's namespace. All UI elements are wrapped in a `tagList()` function, where a `tagList` allows one to combine multiple UI elements into a single R object. Readers should consult the "modules," "tests," and "shinymgr_modules" tutorials for additional information.

```
#!! ModName = iris_cluster
#!! ModDisplayName = Iris K-Means Clustering
#!! ModDescription = Clusters iris data based on 2 attributes
#!! ModCitation = Baggins, Bilbo.  (2022). iris_cluster. [Source code].
#!! ModNotes =
#!! ModActive = 1
#!! FunctionReturn = returndf !! selected attributes and their assigned clusters !! data.frame

iris_cluster_ui <- function(id){
  # create the module's namespace
  ns <- NS(id)

  tagList(
    sidebarLayout(
      sidebarPanel(
        # add the dropdown for the X variable
        selectInput(
          ns("xcol"),
          label = "X Variable",
          choices = c(
            "Sepal.Length",
            "Sepal.Width",
            "Petal.Length",
            "Petal.Width"
          ),
          selected = "Sepal.Length"
        ),

        # add the dropdown for the Y variable
        selectInput(
          ns("ycol"),
          label = "Y Variable",
          choices = c(
            "Sepal.Length",
            "Sepal.Width",
            "Petal.Length",
            "Petal.Width"
          ),
          selected = "Sepal.Width"
        ),
        # add input box for the cluster number

        numericInput(
          ns("clusters"),
          label = "Cluster count",
          value = 3,
          min = 1,
          max = 9
        )
```

```
      ), # end of sidebarPanel

      mainPanel(
        # create outputs
        plotOutput(
          ns("plot1")
        )
      ) # end of mainPanel
    ) # end of sidebarLayout
  ) # end of tagList
} # end of UI function

iris_cluster_server <- function(id) {

  moduleServer(id, function(input, output, session) {

    # combine variables into new data frame
    selectedData <- reactive({
      iris[, c(input$xcol, input$ycol)]
    })

    # run kmeans algorithm
    clusters <- reactive({
      kmeans(
        x = selectedData(),
        centers = input$clusters
      )
    })

    output$plot1 <- renderPlot({
      par(mar = c(5.1, 4.1, 0, 1))
      plot(
        selectedData(),
        col = clusters()$cluster,
        pch = 20,
        cex = 3
      )
    })

    return(
      reactiveValues(
        returndf = reactive({
          cbind(
            selectedData(),
            cluster = clusters()$cluster
          )
        })
      )
    )

  }) # end of moduleServer function

} # end of irisCluster function
```

References

Modularizing shiny app code, 2020. URL https://shiny.posit.co/r/articles/improve/modules/. Accessed: 2010-09-30. [p157]

J. M. Alston and J. A. Rick. A beginner's guide to conducting reproducible research. *The Bulletin of the Ecological Society of America*, 102(2):e01801, 2021. doi: https://doi.org/10.1002/bes2.1801. URL https://esajournals.onlinelibrary.wiley.com/doi/abs/10.1002/bes2.1801. [p157]

D. Attali. *shinyjs: Easily Improve the User Experience of Your Shiny Apps in Seconds*, 2021. URL https://CRAN.R-project.org/package=shinyjs. R package version 2.1.0. [p159]

C. Balantic and T. Donovan. Ammonitor: Remote monitoring of biodiversity in an adaptive framework with r. *Methods in Ecology and Evolution*, 11(7):869–877, 2020. doi: https://doi.org/10.1111/2041-210X.13397. [p159]

C. Brett and I. Neuhaus. *periscope: Enterprise Streamlined 'Shiny' Application Framework*, 2022. URL https://CRAN.R-project.org/package=periscope. R package version 1.0.1. [p157]

P. Campbell. *shinyauthr: 'Shiny' Authentication Modules*, 2021. URL https://CRAN.R-project.org/package=shinyauthr. R package version 1.0.0. [p157]

W. Chang and B. Borges Ribeiro. *shinydashboard: Create Dashboards with 'Shiny'*, 2021. URL https://CRAN.R-project.org/package=shinydashboard. R package version 0.7.2. [p159]

W. Chang, G. Csárdi, and H. Wickham. *shinytest: Test Shiny Apps*, 2021. URL https://CRAN.R-project.org/package=shinytest. R package version 1.5.1. [p161]

W. Chang, J. Cheng, J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges. *shiny: Web Application Framework for R*, 2022. URL https://CRAN.R-project.org/package=shiny. R package version 1.7.3. [p157, 159]

L. Clarfeld, C. Tang, and T. Donovan. *shinymgr: A framework for building, managing, and stitching shiny modules into reproducible workflows.*, 2024. R package version 1.1.0. [p159]

C. Fay and S. Rochette. *shinipsum: Lorem-Ipsum Helper Function for 'shiny' Prototyping*, 2020. URL https://cran.r-project.org/web/packages/shinipsum/index.html. R package version 0.1.0. [p157]

C. Fay, S. Rochette, V. Guyader, and C. Girard. *Engineering Production-Grade Shiny Apps*. Chapman and Hall/CRC, 2021. doi: https://doi.org/10.1201/9781003029878. [p157]

R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2): 179–188, 1936. doi: https://doi.org/10.1111/j.1469-1809.1936.tb02137.x. [p158]

R. Gentleman and D. T. Lang. Statistical analyses and reproducible research. *Journal of Computational and Graphical Statistics*, 16(1):1–23, 2007. doi: 10.1198/106186007X178663. URL https://doi.org/10.1198/106186007X178663. [p157]

R. D. Hipp. SQLite, 2020. URL https://www.sqlite.org/index.html. [p159]

J. Kim and H. Lee. *jsmodule: 'RStudio' Addins and 'Shiny' Modules for Medical Research*, 2022. URL https://CRAN.R-project.org/package=jsmodule. R package version 1.3.0. [p158]

M. Kosinski. *shiny.reglog: Optional Login and Registration Module System for ShinyApps*, 2022. URL https://statismike.github.io/shiny.reglog/. R package version 0.5.2. [p157]

C. Larman. *Agile and iterative development: a manager's guide*. Addison-Wesley Professional, 2004. [p157]

G. Lin. *reactable: Interactive Data Tables Based on 'React Table'*, 2022. URL https://CRAN.R-project.org/package=reactable. R package version 0.3.0. [p159]

K. Müller, H. Wickham, D. A. James, and S. Falcon. *RSQLite: SQLite Interface for R*, 2022. URL https://CRAN.R-project.org/package=RSQLite. R package version 2.2.14. [p159]

R. D. Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011. doi: 10.1126/science.1213847. URL https://www.science.org/doi/abs/10.1126/science.1213847. [p157]

V. Perrier, F. Meyer, and Z. S. Abeer. *datamods: Modules to Import and Manipulate Data in 'Shiny'*, 2022. URL https://CRAN.R-project.org/package=datamods. R package version 1.3.3. [p157]

R Special Interest Group on Databases (R-SIG-DB), H. Wickham, and K. Müller. *DBI: R Database Interface*, 2022. URL https://CRAN.R-project.org/package=DBI. R package version 1.1.3. [p159]

B. Schloerke, J. Allaire, and B. Borges. *learnr: Interactive Tutorials for R*, 2020. URL https://CRAN.R-project.org/package=learnr. R package version 0.10.1. [p159]

S. Stoudt, V. N. Vásquez, and C. C. Martinez. Principles for data analysis workflows. *PLOS Computational Biology*, 17(3):e1008770, 2021. doi: https://doi.org/10.1371/journal.pcbi.1008770. [p157]

K. Ushey. *renv: Project Environments*, 2023. URL https://rstudio.github.io/renv/. R package version 0.17.3. [p159]

H. Wickham. testthat: Get started with testing. *The R Journal*, 3:5–10, 2011. URL https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf. [p161]

K. Żyła, J. Nowicki, L. Siemiński, M. Rogala, R. Vibal, and T. Makowski. *rhino: A Framework for Enterprise Shiny Applications*, 2023. https://appsilon.github.io/rhino/, https://github.com/Appsilon/rhino. [p157]

Laurence A. Clarfeld
Vermont Cooperative Fish and Wildlife Research Unit
302 Aiken Center, University of Vermont
Burlington, VT 05405 USA
ORCiD: 0000-0002-3927-9411
laurence.clarfeld@uvm.edu

Caroline Tang
Queen's University
Biology Department
116 Barrie St, Kingston, ON K7L 3N6
ORCiD: 0000-0001-7966-5854
17ct24@queensu.ca

Therese Donovan
U.S. Geological Survey, Vermont Cooperative Fish and Wildlife Research Unit
302 Aiken Center, University of Vermont
Burlington, VT 05405 USA
ORCiD: 0000-0001-8124-9251
tdonovan@uvm.edu