Introduction to Shiny & psychTestR

Klaus Frieler

Hanover Music Lab

11th March 2022

Agenda

Zeroth part: Preparation

- Hands on: Rstudio, installing packages, GitHub account, creating a repository

First part: Shiny

- Input: Concepts & principles
- Hands on: Your first shiny Shiny app

Second part: psychTestR & psychTestRCAT

- Hands on: Getting started & setting up
- Input: History, purpose & concepts
- Hands on: How to add a questionnaire to psyquest
- Hands on: How to make a test battery

Zeroth Part

Better be prepared

Installation Orgy

What you need

- R stuff
  - R
  - RStudio
  - Packages: tidyverse, shiny, shinythemes, bslib, DT, devtools & more
  - For the second part: psychTestR, psychTestRCAT
- Git stuff
  - Git (Windows: git-bash)
  - Github account

Git

- Git is a version control system.
- De facto standard today (alternatives CVS, SVN, Microsoft's Source Safe).
- Biased towards textual data (diffs), but binary data is also fine.
- Allows to track all changes and version of a project ("Track Changes for projects").
- Allows collaboration of teams on the same (complex) project.
- All changes are tracked in a central repository.
- Repository reside typically in the cloud (e.g. Github, Bitbucket), but can be held locally as well.
- Git is basically a command line tool, but there are GUIs, e.g. RStudio has a good Git integration.

Git Workflow

- Create or fork repository (e.g., on/from Github, Bitbucket)
- Init or clone repository locally.
- Branch out alternative development lines, if necessary
- Enter a typical workflow
  - Pull latest changes from repository
  - (Merge in case of conflicts)
  - Work on your project
  - Commit changes (frequently, based on “logical” work units)
  - Push your commits back to repository (resolve conflicts if any)
  - Repeat

Hands On: GitHub

- Create Github account
- Create Github PAT (Personal Access Token)
- Create new (dummy) repository
- Add README.md. (Markdown file)
- Invite other participants as collaborators
- Fork repository from another participant
- Clone one of your repositories, pull
- Arbitrarily change README
- Commit and push your changes
- Delete all your repositories

Tidyverse Crash Course

- Shiny was developed by Rstudio and is closely tied to the tidyverse.
- Tidyverse was a game-changing set of libraries for the R universe.
- Tidyverse can massively improve productivity and facilitate readability and learnability.
- If you are not familiar with the tidyverse, I strongly recommend to check it out.

Tidyverse Crash Course

- Packages in the tidyverse:
  - `ggplot2` (best plotting there is)
  - `tibble` (a better data.frame)
  - `tidyr` (tools to keep data “tidy”, i.e. data shape transformations)
  - `dplyr` (tools to manipulate data frames, filtering, variable selection, grouping, summarizing, mutating, sorting)
  - `purrr` (applying functions over lists and data frames)
  - `Stringr, glue` (string functions)
  - `readr` (reading data)
  - `forcate` (manipulating factors)
- Workers in the background: `rlang`, `tidyselect`, `vctrs`, `magrittr`…

Tidyverse Crash Course

- Most important tool is the pipe %>%, (taken from magrittr, preceding the tidyverse)
- It allows simple chaining of functions calls.
- Example:

```
x %>% abs %>% sum %>% divide_by(length(.))
```

is equivalent to

```
sum(abs(x))/length(x)
```

- Example:

```
my.data.frame %>% filter(age < 30) %>% select(age)
```

is equivalent to

```
my.data.frame[my.data.frame$age < 30, “age”]
```

Some Assorted Tips

- Use RStudio and Rstudio projects (no setwd(), or rm(list = ls()))
- Use some consistent formatting style to enhance readability (e.g., tidyverse style):
  - Name files, variables, functions etc. always with snake_case (lower letters). Don’t use camelCase, use UpperCamelCase only for classes. Exceptions should be motivated.
- Avoid writing monolithic scripts
  - Break up code into small, manageable and testable functions.
  - Avoid global variables.
  - Split larger projects in multiple files with semantic unity (e.g. data I/O and wrangling, plotting, statistical analysis, “paper production”).
  - Ideally, never save workspace but rebuild it on restart (not always practical).
  - DRY: Don’t repeat yourself

Some Assorted Tips

- Use GitHub if possible
- Always think about other people (e.g. reviewers) reading and potentially using your code.
- Document/comment complicated stuff.
- Good variable and functions names and a clear structure are half of a documentation.
- Good error culture is crucial
  - Making errors is the basically the definition of programming.
  - Find bugs and fix them, no need to blame anyone (particularly not yourself).
  - Test as much as possible (use debugging tools, like browser())
- Golden rule: Make it run, make it right, make it fast.

First Part
Shiny at your hands

Input: Shiny what is it?

- Shiny is a web development framework for R.
- Package Shiny + Shiny Server (integrated in Rstudio for local use).
- Main purpose is building interactive data visualization tools.
- But can be used for much more: https://www.rstudio.com/blog/winners-of-the-2nd-shiny-contest/
- Shiny produces dynamic HTML pages, powered with Bootstrap CSS and jQuery JavaScript.
- Reactive Programming Paradigm: Event-based instead of linear program flow.
- Main components: an User Interface (UI) and a server function.

Quick Tour of the Web

- The World Wide Web is a communication layer on top of the internet (other layers: Mail (POP/SMTP, IMAP), FTP, SCP, SSH, SNMP, TCP, …)
- A web server listens to HTTP requests from clients and sends HTML answers (or errors) back to browsers which render the HTML into web pages.
- HTML: Hypertext Markup Language, uses <tags> to (logically) markup content</tag>.
- CSS (Cascading Stylesheets): Formatting directives for the markup elements.
- JavaScript: scripting language to add interactivity and to dynamically alter content and appearances on the client side.
- HTML documents typically include CSS and JavaScript.
- Hands on: Open www.google.com and press Ctrl + Shift + I (Chrome)

Structure of a HTML document

```
<html>
  <head>
  <!-- meta instructions, external includes, scripts  -->
  </head>
  <body>
    Content goes here...
    <a href = "https://www.google.com">A link to google</a>
    <img  src = "https://www.google.com/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png"/>
    <p style = "color:red">Roter <span style = "font-size:40px">großer</span>Text</p>
    <div style =
           "margin-left:500px;border:solid 1px;padding:5px;max-width:768px;
            font-family:serif;color:teal;font-size:100px">
       A container
      <h6 onclick="if(this.style.color != 'red'){this.style.color = 'red';this.innerHTML = 'Clicked';} else
{this.style.color = 'teal';this.innerHTML = 'Click me!';}">Click me!</h6>
</div>
</body>
```

Shiny App

- A Shiny app is an R script that contains a call to shinyApp(ui, server).
- `ui()` is a function that contains the code to define the user interface, i.e., what is displayed on the Shiny app page.
- server(input, output, session) is function that contains the logic to create output elements and observe and react to reactive elements.
- Two options:
  - Single file app: app.R (preferred)
  - Multi file app: ui.R, server.R
  - All files can include other files via source()
  - globals.R is automatically included
  - Shiny Servers automatically recognize these file names

Shiny Apps: Workings

- Shiny apps can be run
  - locally with Rstudio ("Run App" button, internal or external browser)
  - on a remote server, using a ShinyServer (copy app.R in a dedicated subfolder and that's it!)
- Upon HTTP request, the Shiny Server returns a initial HTML page and opens a WebSocket connection with the client for further communication between shiny/httpuv (R) and jQuery (JavaScript).
- User interacts with app → predefined events are sent to the Shiny Server → reactive environments consume events → new HTML is generated if necessary using rendering functions → results send back to browser/user → web page dynamically updated → loop ad infinitum

Shiny Apps: Workings

- The UI function defines input and output elements with unique ids that are "automagically" bound to input and output objects, as fields with the same name.
- All events from the UI are stored in the **input** object provided to the the **server** function.
- Output is gathered in the **output** object of the server function.
- In the server function, reactive function that contains the **input** object are triggered when used fields of **input** object are changed.

Shiny Apps Demo: ui

```
ui <- fluidPage(
    # Application title
    titlePanel("Old Faithful Geyser Data"),
    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),
        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("distPlot")
        )))
```

Shiny Apps Demo: server

```
server <- function(input, output) {

    output$distPlot <- renderPlot({
        # generate bins based on input$bins from ui.R
        x    <- faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)

        # draw the histogram with the specified number of bins
        hist(x, breaks = bins, col = 'darkgray', border = 'white')
    })
}
```

Input and Output

ui

```r
ui <- fluidPage(
    # Application title
    titlePanel("Old Faithful Geyser Data"),
    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),
        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("distPlot")
        )))
```

server

```r
server <- function(input, output) {

    output$distPlot <- renderPlot({
        # generate bins based on input$bins from ui.R
        x    <- faithful[, 2]

        bins <- seq(min(x),
                    max(x),
                    length.out = input$bins + 1)

        # draw the histogram
        hist(x, breaks = bins, col = 'darkgray',
             border = 'white')
    })
}
```

Hands On: Gettin’ real

- Create new repository for your workshop project (with the same name)
- Clone project into a new folder (using git clone <…> on the command line)
- Launch RStudio and create a new project “in existing directory”, the one you just created.
- File → New File → Shiny Web App (name app.R)
- Run App (button on source window or by typing runApp() in the R console)
- Tip: use external browser instead of inbuilt browser.

Hands on: Tweaking and Fiddling

- Change color of title to red.
- Reduce the width of the main panel to 4 columns (there are 12 columns in total).
- Add a second plot for the other data column in `faithful` data.
- Add a second slider input for bins of second plot.
- Extract function to generate histograms and rewrite plot functions.
- Change plot title to name of data column.
- Add select input for bar colors (red, green, blue) and border colors (white, black).
- Add checkbox to display either a histogram or a density.
- Remove second plot (and slider) and add drop box to select data columns instead.
- Add another drop box to select faithful, mtcars, or diamonds datasets, update variable dropbox dynamically, select only numeric variables, change title color with plot color

UI Design

- UIs can have several layouts:
  - Fluid and fixed pages
  - Sidebar layout
  - Tab sets
  - Nav bars
  - Combinations
  - Dashboard (need extra package)
- Hands on: Change the main panel to a tab set panel with two tabs, “plot” for plot the histogram/density plots and “summary” with a table of descriptive stats (mean, median, summary etc. of the selected column variable).
- Change theme to “yeti” or any other (install “shinythemes” if need be)

UI Input elements

- Shiny provides a range of standard input elements (control widgets)
  - Slider input
  - Select box (drop down)
  - Check box, check box groups
  - Radio buttons
  - Action buttons
  - Download buttons, file input
  - Text, numeric, date input
  - Date range

UI Output Elements

- Shiny provides a range of standard rendering functions
  - Text and verbatim text
  - Plot output
  - Image output
  - Tables (data.frame, matrix, DataTable)
  - HTML/UI output
- Used via corresponding render* functions, e.g. renderPlot(), renderText()
- Hands on: Add a third tab “Data” with a DT data table of the data (install package DT if needed).

Hands on: Make it your own

- Change the data for your own data, import your data at start-up (remove data set selection).
- Insert all numeric variables of your dataset in the variable select box in the sidebar.
- Add a drop down for the values of one your main grouping variables. Use this to filter your data on the fly. Add also a “no filter” option.
- Add a download button for your data on the sidebar or the summary tab.
- Add imprint info on the sidebar and/or the summary tab or add an “info” tab.
- Push all data to GitHub.
- Bonus: Replace hist/density plot with ggplot equivalents, remove filter, add select box for all your grouping variables and facet all plots/summary stats by selected grouping variable
- Commit and push your project to GitHub

Second Part
psychTestR

Hands on: Getting started

- First install psychTestR
- Not on CRAN, install from GitHub using the “remotes” or the “devtools” package, which maybe need to be installed first.
- Go to https://github.com/pmcharrison/psychTestR and follow instructions
- Install psychTestRCAT the same way.
- Go to https://github.com/klausfrieler/psyquest, create a fork and clone it in a good space.

Input: psychTestR

- psychTestR was developed by Peter Harrison in context of the LongGold project, in order to get rid of the ConcertoServer, which was used to implement adaptive tests, as the ConcertoServer had some drawbacks.
- psychTestR is based on Shiny, but you would not easily notice.
- Allows standalone lab-based and massive online experiments.
- Supports building test batteries from modular components.
- Supports internationalization to provide tests in several languages at once.
- psychRCAT is an add-on package to psychTestR to implement adaptive testing in an easy way.
- The LongGold project ported all its tests to psychTestR and abandoned Concerto in 2019.

Input: psychTestR

- All tests and questionnaires are implemented as R packages on GitHub, hence it is open source, easy to install, easy to enhance, and easy to customize.
- This allows for modularization and reusability of test components by plugging them together to test batteries, while adding customer specific parts, if needed.
- Can also be integrated with other online survey platforms.
- In 2021, the DOTS consortium initiated the implementation of a dedicated Shiny test server, powered by the DGM, to allow easy access to these resources for society members.
- Accompanying monitor apps (Shiny) are easily provided.
- Over 40 tests are questionnaire available as of now.
- Many more tests and packages are under development or in planning.

Hands on: Adding a questionnaire to psyquest

- psyquest is the main psychTestR package for LongGold questionnaires (total of 29).
- (Simple) Questionnaires can be easily added by providing basically only a dictionary and an item bank (item definitions).
- Task: Implement one of the tests (Barcelona Reward, Healty/Unhealty Music Scale, Highly Sensitive Person Scale, Profile of Mood States).

Hands on: Adding a questionnaire to psyquest

1. Choose a three letter acronym, e.g. BMR for Barcelona Music Reward Quest).
2. Define a dictionary, use one of the “dict” files in psyquest/data_raw/dicts, e.g. DAC.csv as example or template.
3. Define an item bank using of the item bank files in psyquest/data_raw/item_banks.
4. Item bank define type, scoring, scales of items using and item ID.
5. Dictionaries define textual resources for items using the same ids.

Hands on: Adding a questionnaire to psyquest

Item banks

- Columns; "main_id";"language";"template";"score_func";"subscales";"layout“
- Main id has form T<TESTNAME>_00nn
- Different definitions for languages if necessary, default is “en”.
- Template defines type, mostly “n-option multiple choices” for simple questionnaires.
- Score func for inverting scores or other transformations (R function!)
- Subscales for items, can be more than one, separated by semicolon
- Layout: Vertical or horizontal for n-options multiple choices (vertical mostly fine)

When finished, source “data_raw/item_bank_generator.R” to import item bank

Hands on: Adding a questionnaire to psyquest

Dictionaries

- Edit using a text editor (use/keep utf8 encoding!)
- Entries consist of id and texts in several languages in columns (_f = formal version)
- Items have a prompt and n choices
- The IDs for prompts and choices are coded:
- <ITEMID>_PROMPT
- <ITEMID>_CHOICEn
- <ITEMID> correspond to item id in item bank
- ITEMID with 0000 is reserved for the testname itself, 0001 for instructions, if available.

When finished, source “data_raw/dict_generator.R” to import dictionary

Hands on: Adding a questionnaire to psyquest

- R file
  - Copy R/DAC.R and rename
  - Substitute all occurrences of DAC with your TESTNAME.
- When finished, build psyquest library.
- Test & Debug
- Push new files to GitHub
- File pull request for psyquest

Hands on: GoldMSI and Musical Taste

- Problem: The maker of the GoldMSI claims that it was specifically designed to be independent of musical taste. Is this really true?
- Task: Design and setup a psychTestR battery to test this assertion (English).
- Find all available tests here: http://testing.musikpsychologie.de/
- Consider additional custom questions.
- Start a new project, add a file battery.R

Hands on: GoldMSI and Musical Taste

- A psychTest battery consists of two elements:
  1. A psychTestR::timeline (needed for internationalization) or a simple list of test_elements.
  2. A call to psychTestR::make_test
- For example

```
timeline <- psychTestR::join(
  psychTestR::one_button_page("Page 1"),
  psychTestR::one_button_page("Page 2"),
  psychTestR::final_page("The end")
)
test <- psychTestR::make_test(elts = timeline)
shiny::runApp(test)
```

Hands on: GoldMSI and Musical Taste

- Test elements can be
  - simple pages (predefined in psychTestR or self-defined)
  - control elements (e.g., code_block, conditional, reactivePage)
  - complex modules from other packages
- A timeline is necessary to invoke the internationalization magic
- Timeline objects need a psychTestR::dict object that provides the texts.
- Timelines cannot be nested.
- Timelines must end in a final page.
- Add elts_save_to_disk() at end, if you want to have data…
- make_test can have psychTestR::options for customizing.

Hands on: GoldMSI and Musical Taste

- Setup a test battery.
- Create GitHub repo for this and push (will be published on DOTS)
- Generate some data.
- Get results via admin panel.

Wrap Up

- You hopefully learned:
  - How to use Git and GitHub
  - Basic concepts of Shiny, the Web, and all that.
  - How to implement a simple Shiny App
  - Concepts of psychTestR
  - How to add a questionnaires to psyquest
  - How to set up a psychTestR test battery
- What is missing:
  - A gazillion details, tricks, and fancy packages for enhanced Shiny development
  - How to write a more psychTestR test package from the scratch.

The End