Georg-August Universität Göttingen

Statistical Programming with R

Gotta Read 'Em All: An RStudio Add-In to visually read different file-formats into R

Author:
Stanislaus Stadlmann,
Student ID: 21144637

Supervisor
Paul Wiemann, M.Sc.

Submitted on August 3, 2016

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Contents

List of Figures

List of Tables

1 Motivation

R is a statistical software with an almost uncountable number of functions for different statistical methods, procedures and graphs. On the Comprehensive R Archive Network (CRAN) alone, the most popular network for adding new features to R via so called "packages", more than 8000 packages are ready to be downloaded (footnote CRAN). Each of them provide a variety of functions to solve different tasks. For example, a package called "Vector Generalized Linear and Additive Models" (or short VGAM) can be easily installed via the R command `install.packages("VGAM")` and, once loaded in R, provides functions to estimate a variety of different regression models (Zitat?).

These aforementioned packages with the underlying functions make R the popular statistical package it is today. But most of these additional features require some data to be of any use, most prominently regression model estimation functions. There exist datasets shipped with R, but for most academical purposes, external data will be required to generate new insights.

To read data into R, there are multiple ways, depending on the data type (e.g. .csv, .xlsx) and also the data size (big data, small data). Reading a .csv file, for example, can be achieved via the built-in R function `base::read.table ("filename.csv")`. Big .csv files can be read very quick with the function `data.table::fread("bigfile.csv")` from the data.table package. Other packages provide even more functionality, e.g. for dealing with strings or a smaller number of required arguments inside of a function. Most of those packages are also available on CRAN.

The availability of packages to read different filetypes in numerous ways is very helpful for the advanced R user, because there is almost no filetype that cannot be read via an R function. But it also poses a problem: If there are so many ways that a user can read a file into R, how will she/he remember all the necessary packages and functions, and their function arguments? This problem is often encountered by new R users, who want to use R's extended functionalities but fail at importing data into their working environment.

An answer to this problem is provided by Thomas Leeper's R package called "rio", which tries to minimize redundancy by wrapping R reading functions into one import `rio::import()` and one export function `rio::export()`.

The R package introduced with this paper takes it one step further. Built on the Shiny Framework and implemented as an RStudio Add-In, “Gotta Read ’Em All” provides a GUI for reading all different file-formats into R.

The general process is the following: In the beginning, the user selects a file on her/his computer. After some adjustments (which are done interactively), the proper function to read the file is pasted into the console, with an object name that can be specified by the user. In between, the user can always head to the preview to see what the parsed file would look like with the current options.

Using this Add-In, the user can now read data into R without remembering any code, but still obtains the correct R code to re-parse the data at a later point.

2 Underlying Frameworks

2.1 The Shiny Framework

The Shiny Framework is in itself an R package designed to create interactive visualisations with R functions and HTML code. The author describes the package as “combining the computational power of R with the interactivity of the modern web” (citing a web page blabla).The goal is to create applications with clickable interfaces quickly showcasing different scenarios. This is done via reactive R functions, which are run everytime a user interacts with the GUI.

Figure 1 shows an example Shiny Application, consisting of some user interface (UI) control elements (a select button, and tick boxes) and a graph. The UI is reactive; whenever the user ticks a box or selects a different value in the first box, the graph changes its look. This is built upon reactive functions, which run every time a value inside them changes. The values that are allowed to change are therefore bound to the UI elements.

Shiny Apps consist of two elements: The UI and a “server” side. The UI includes functions which wrap HTML to build the viewable part of the App, for example buttons and placement of graphs. The server side consists of functions that specify the reactive R functions which create dynamic output displayed on the UI side, and when the functions should be run. Both sides then are able

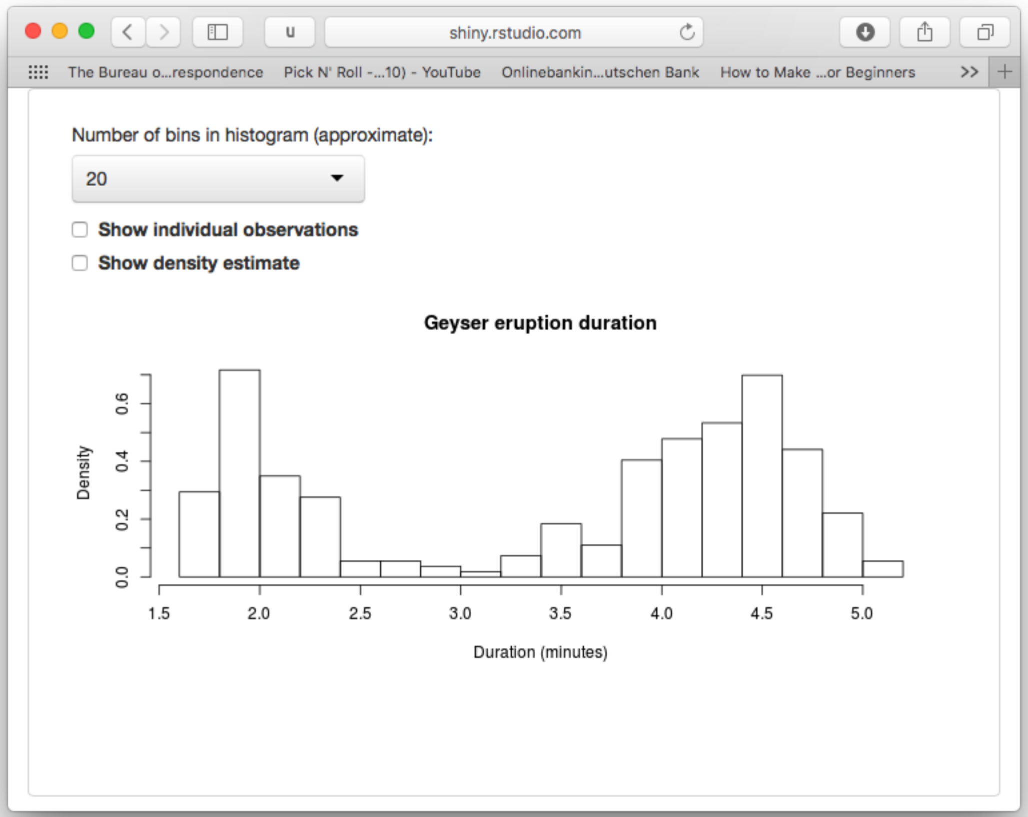
shiny.rstudio.com
The Bureau o...respondence
Pick N' Roll -...10) - YouTube
Onlinebankin...utschen Bank
How to Make ...or Beginners
Number of bins in histogram (approximate):
20
Show individual observations
Show density estimate
Geyser eruption duration
Density
0.0
0.2
0.4
0.6
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Duration (minutes)

Figure 1: An example Shiny App.

to automatically communicate with each other to create a smooth interactive experience for the user. Code for an example Shiny app is attached below:

```r
# Define UI
ui <- bootstrapPage(
  numericInput("n", "Number of obs", 100),
  plotOutput("plot")
)
# Define Server
server <-  function(input, output) {
  output$plot <- renderPlot({ hist(runif(input$n)) })
}
app <- shinyApp(ui, server)
runApp(app)
```

We can now see that both the UI and server elements are R objects, while the server also resembles an R function. `runApp()` then opens up the Application.

2.2 Shiny Gadgets

2.3 RStudio Add-Ins

3 Implementation

3.1 The Main Function: GREA()

3.2 The Reading Function: GREA_read()

3.3 Helper Functions

4 Usage

Appendix

References

[1] blubb