Lab 02 - R File I/O and Plotting

Learning outcomes

In this lab you will learn and practise

- More R syntax,
- Some basic plotting,
- The Shiny package that builds a web framework

Simple R exercises

In the RStudio console, complete the following:

Create the following vectors:

```
(1,2,3,...,19,20)
(20,19, ...., 2,1)
(1,2,3,...,19,20,19,18,..,2,1)
```

Use the `rep()` function

Create a vector `(4,6,3)` and assign it to the variable `tmp`.

Look up the function `rep()` in the Help window, and write down the R code that produces the following vectors using the `tmp` vector above:

```
(4,6,3,4,6,3,...,4,6,3)          # there are 10 occurrences of 4.
(4,6,3,4,6,3,...,4,6,3,4)        # there are 11 occurrences of 4, 10 occurrenc
                                 # and 10 occurrences of 3.
(4,4,...,4,6,6,...,6, 3,3,...,3) # there are 10 occurrences of 4, 20 occurrenc
                                 # and 30 occurrences of 3.
```

Use the `paste()` function

Create the following character vectors of length 30:

```
("label 1" "label 2" .... "label 30")
     # Note that there is a single space between label and the number followir
("fn1" "fn2" ... "fn30")
     # In this case there is no space between fn and the number following, igr
```

Use the `rnorm()` and `sample()` functions

Create a 5×10 matrix with 10 NAs scattered around in the matrix, similar to the one shown below. Note that because of the randomness in the sampling process, your matrix will not look exactly the same each time you run your code.

Imagine that these are patient records, so give each row a name and each column a name. Use the resulted data frame to practise *subsetting* using: - positive integers as indices - negative integers as indices - character strings as indices, and - logic values as indices.

```
##            [,1]        [,2]       [,3]       [,4]       [,5]       [,6]
## [1,]  0.6650485 -1.60234089  1.6962204  0.1305192         NA  0.2591008
## [2,]  0.4063729          NA         NA  0.7378805  1.3074990  0.4969716
## [3,]  1.2116769  0.91862879 -1.5248981 -2.1224044 -0.1485366         NA
## [4,] -1.5053502          NA  0.2147029 -0.2282579  0.6898290  0.2328568
## [5,]  0.3577401 -0.04216234  0.2069640  0.7182465  0.6638631 -0.4434087
##            [,7]      [,8]        [,9]      [,10]
## [1,] -1.0872586 0.5306909          NA -1.9203260
## [2,]         NA 0.6014770 -0.96030573         NA
## [3,] -0.9408726        NA  0.08182994  0.9724694
## [4,]  0.1804114 0.8954715          NA  1.1023495
## [5,]  0.3308494 1.0829487  1.34120763 -0.1074303
```

File I/O

Read files using `read.csv()` and simple plotting

Let's take a look at the *wine quality* dataset available at the UCI Machine Learning database: https://archive.ics.uci.edu/dataset/186/wine+quality

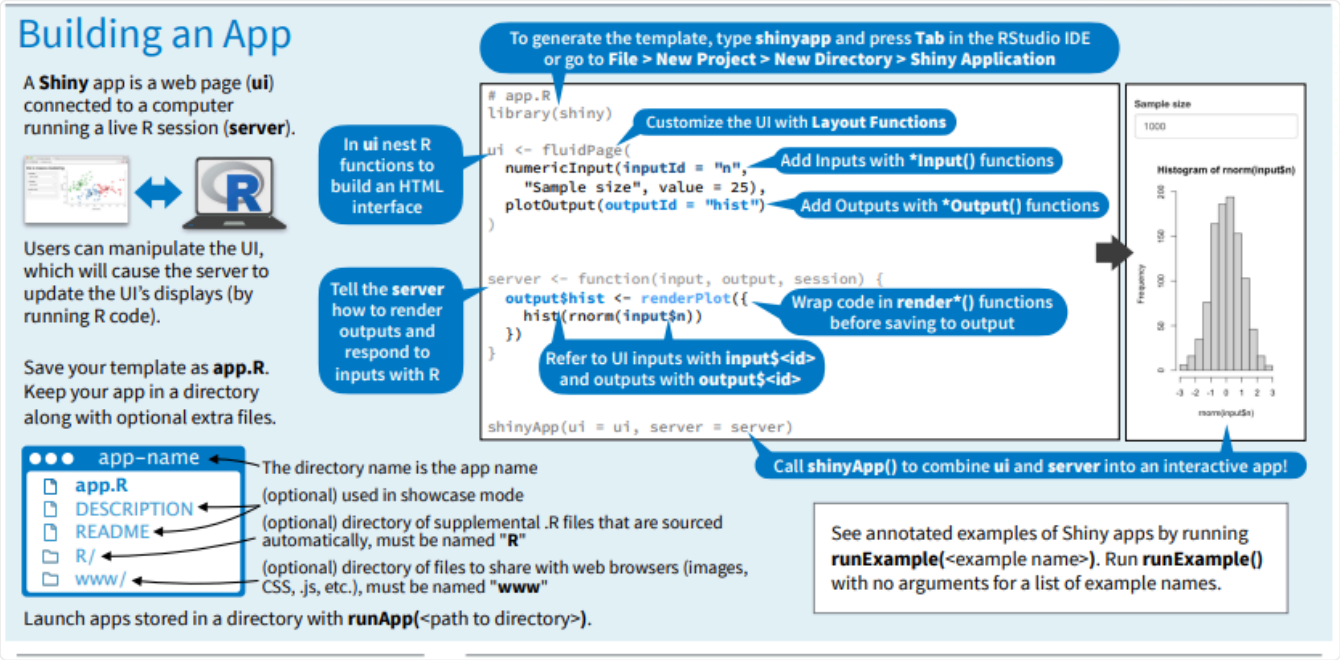
Read the `winequality-red.csv` file using the `read.csv()` function into a data frame called `rw`. Note that the data columns are semi-colon separated, not comma. You can give the URL of the file to the `read.csv()` function. If it doesn't work, then download the `winequality-red.csv` file and read the data from your local disk.

```
winedata <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-database
```

Do the following exercises:

- Use the `head()`, `str()` and `summary()` functions to explore the data in `winedata`. Do we have any categorical variables?
- Use appropriate functions from the `ggplot2` and `gridExtra` libraries to show the histograms for the `residual.sugar` and `fixed.acidity` variables side-by-side.

A First Taste of Shiny App



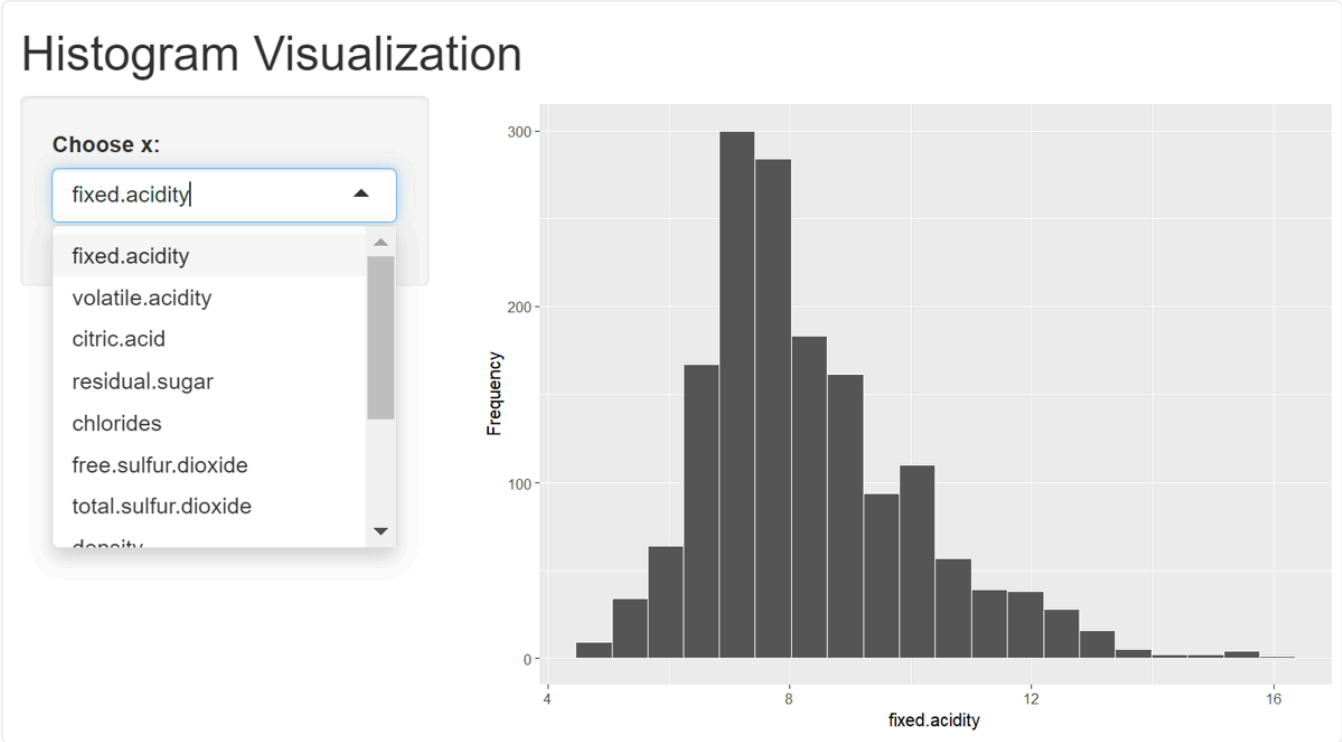
See the full Shiny cheat-sheet at: https://rstudio.github.io/cheatsheets/shiny.pdf (We will introduce more UI controls in later labs).

Create a Shiny App file by following the instructions on RStudio, work out the code segments that need to be modified to turn sample code into the following code to display a histogram and enable user selection for the x-axis variable. Note the flow of execution:

1. the UI uses a sidebarLayout, which has two panels. The sidebarPanel accepts user input through a selection (drop-down) list, that takes values from the `names(df)`. The mainPanel displays the output plot, which is named `histogram`.
2. The server defines a functions called `generate_histogram` which takes `data`, the variable to plot `x_var` and a chart `title`. Then the function is called with the data frame `df`, the UI input `input$x1` and the column name of `input$x1` as chart title. The function call returns an plot object that can be rendered, which is stored as a property for the `output` variable.

```r
# Load required libraries
library(shiny)
library(ggplot2)


# load dataset
df <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine

# Define the UI
ui <- fluidPage(
    titlePanel("Histogram Visualization"),
    sidebarLayout(
        sidebarPanel(
            selectInput(inputId = "x1", label = "Choose x:", choices = names(d
        ),
        mainPanel(
            plotOutput(outputId = "histogram")
        )
    )
)


# Define the server
server <- function(input, output) {
    generate_histogram <- function(data, x_var, title) {
        ggplot(data, aes_string(x = x_var)) +
        geom_histogram(bins = 20, color = "white") +
        labs(title = title, x = x_var, y = "Frequency")
    }
    # Render the side-by-side histograms
    output$histogram <- renderPlot({
        generate_histogram(df, input$x1, colnames(input$x1))
    })
}

# Run the Shiny app
shinyApp(ui = ui, server = server)
```

- When executed, there should be displaying the following plots:

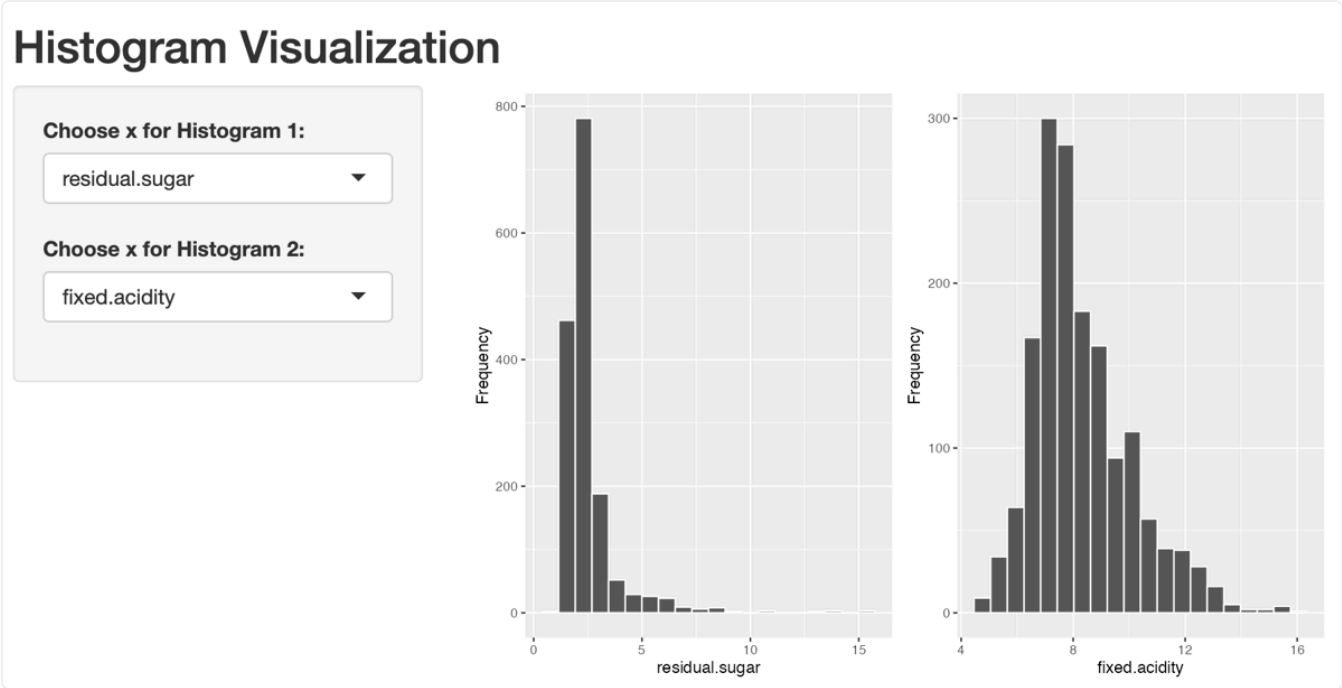


Your Turn: Complete the code below by replacing the "TO DO, add your own codes" and present two histograms side-by-side, enabling the user to choose x-axis variables for each histogram (You need to modify the code below by yourself).

You should attempt this after the lecture covering it.

```r
# Load required libraries
library(shiny)
library(ggplot2)
library(gridExtra)

df <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine
# Define the UI
ui <- fluidPage(
    titlePanel("Histogram Visualization"),
    sidebarLayout(
        sidebarPanel(

            selectInput("x_var1", "Choose x for Histogram 1:", choices = names
            selectInput("x_var2", "Choose x for Histogram 2:", choices = names
        ),
        mainPanel(plotOutput("histogram"))
    )
)

# Define the server
server <- function(input, output){
    # Function to generate the histogram plots
    generate_histograms <- function(data, x_var, title){
        # TO DO, add your own codes
    }
    # Render the side-by-side histograms
    output$histogram <- renderPlot({
        # TO DO, add your own codes
    })
}
# Run the Shiny app
shinyApp(ui = ui, server = server)
```

When executed, there should be a pop-up window that displays the following plots:



The pop-up webpage

- Create a new data frame `df` which has all the `fixed.acidity` values greater than or equal to 7 and the corresponding observations for the `residual.sugar` variable (i.e., `df` should have just two columns).
- Use an appropriate function from `ggplot2` to show a scatter plot of the two variables in the data frame `df`.
- Convert the previous exercise into a Shiny app, enabling the user to choose both x and y variables for the scatter plot.

Input/Output redirection - `source()` and `sink()`

By now, you should have learned how to do some basic coding in R. Rather than typing every R statement in the RStudio interface, you can put your R code into a file. Suppose that you have a file called `lab03.R` in the current working directory. You can open it in RStudio and run through it line by line. Alternatively, you can use the `source()` function to indicate where the source R code comes from, e.g.,

```
source('lab03.R')
```

will run all the R statements in the file from beginning to end (if there is no error in the code). This is how you redirect *standard input* (the keyboard) to a file. If the file is not in your current working directory, you can use the `setwd()` function to change the working directory or you can specify the full path name to the R file that you want to run, e.g., `source('../../project/project1.R')`.

You can also redirect the *standard output* (the computer screen) to a file using the `sink()` function, e.g.,

```
sink('lab03-output.txt')
```

will save all the text output (not any graphics/figures) generated from running R statements (through a file or typing on the keyboard) to the file `lab03-output.txt`. To stop the standard output redirection, simply type:

```
sink()
```

Managing your R workspace

When you exit R or RStudio (by typing `q()`), you would often be asked (this depends on your setting) whether you want to save the workspace image. It is a good practice to save your workspace if there are some variables that have taken you a long time to generate and you want to be able to use them again in the next R session. However, you should also keep in mind that your R workspace is different from other people's R workspace. So if you need to deploy your R code to someone else (e.g., submitting your project for assessment), then while your R code runs perfectly in your R environment, it may not run in a different R environment because of the missing variables. Before submitting your R code for assessment, you should clean up your R workspace and rerun your R code from a fresh environment. This will allow you to detect whether you have forgotten to define any variables in your code.

Several useful and related R commands are given below:

- `ls()` – this function will list all the variables currently defined in your R workspace
- `rm("mpg")` – this function will remove the variable named `mpg` from the R workspace. Note that you need to have the double-quotes around the variable name.
- `rm(list = ls())` – this function will remove all the variables from the R workspace.
- If you only have a text-based terminal window because you are logged on from a remote machine to use R, then it would be easier to just use R instead of RStudio. You can specify to start R with a clean environment as follows:
  `R --no-save --no-restore-data --quiet`
  This command will start R without loading the previously saved workspace. The option `--no-save` indicates that whatever variables that are created in the R session will not be saved when exiting R.
-

While inside RStudio, to clear the workspace is very easy. Just select the `Session` menu then the item `Clear Workspace...`

Continue the learning journey on `swirl`

Type

```
library(swirl)
swirl()
```

and complete lessons 8-12 and then 15 in the *R Programming* course.

R Markdown Files

So far we have seen that you can save the code into a .R file so you can edit and excute later. Another more powerful tool that RStudio provides is R Markdown, .rmd file. R Markdown provides an authoring framework that allows you to combine documentations with excutable code into a single R Markdown file, in a similar fashion as Jupyter Notebook for Python. A file with a `.rmd` extension allows you to

- Save and execute code
- generate high quality reports that can be shared with an audience

For example, all the CITS4009 lecture notes are R Markdown files, that can be "knitted" to HTML files, PDF files or Powerpoint Slides.

Follow the Markdown Basics

> Follow the Markdown Basics subpage to turn what you have done in this lab into a Markdown file, with code chunks and inline code, as well as suitable formatting and show your lab facilitator for feedback.

Previous
R Coding Conventions

Next
Markdown Basics

Last updated 2 months ago