# Higher Level Data Analysis for the Iris Dataset: Data Summary Layout, Display Table, and Row Selection for Scatter Plot

*Katie Li*

*12/3/2017*

## Introduction

- The purpose of the post is to explore **alternative**, sometimes **higher-level** data analysis methods for Shiny App than the ones explored in class. In the last homework assignment, we created scatter plot, histogram, and barchart for a psuedo stats133 dataset. Here, using the iris dataset, a built in dataset in R, I tried to use alternative data analysis methods to make meaning of "iris". The post begins with the creation of a data summary page, followed by a table disyplay, and ends with row selections of table that corresponds to a scatterplot. The content builds on top of what we covered so far in class and demonstrates the vast functions in the Shiny App package. Note that the version of the R Studio used here is 1.1.383 for Mac OS X 10.6+. In addtion, the references that are used are linked collectively in the end of the blog post.

## Required Package

- Before being able to replicate results of this blog post, make sure that you have downloded both the Shiny package (here, I am using 1.0.5) and a package called DT (version 0.2), which provides an R interface that allows R data objects to be displayed as tables on HTML pages. Under the DT package, DataTable provides filtering, sorting, and many other features that do not come automatically under the Shiny package.
- After you finished installing the packages using `install.packages()`, make sure to load the two packages that we will be using in the rest of the blog post.

`library(shiny)` `library(DT)`

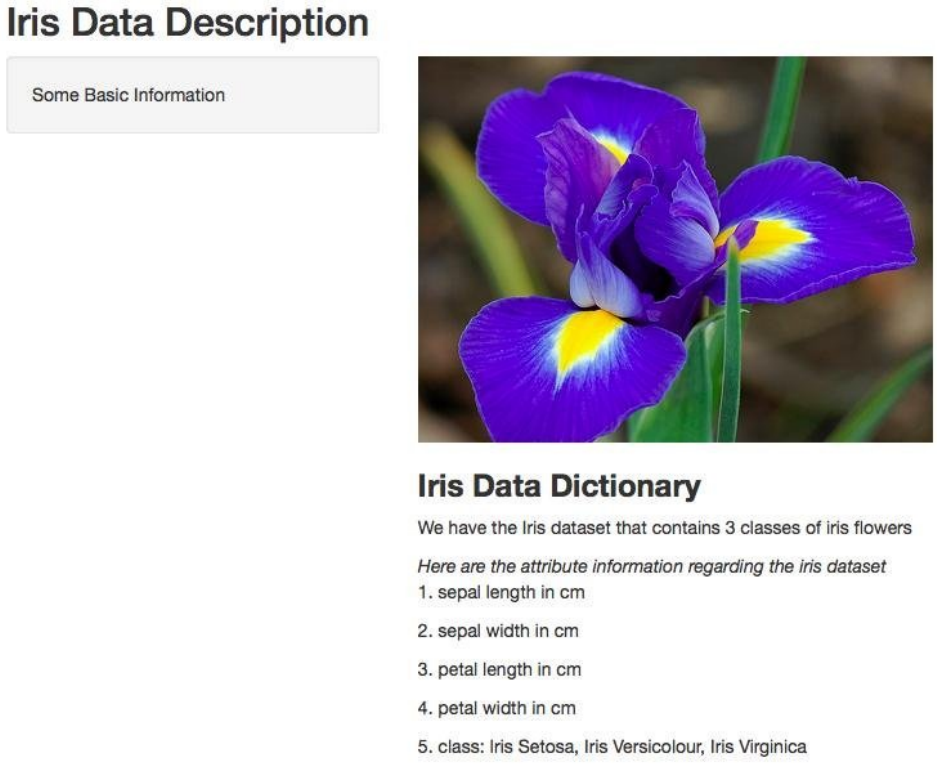
## Required Dataset: Iris

- Iris is a built-in dataset in R. It describes 3 iris species and their petal length, petal width, sepal length, and sepal width. Please load data using `data("iris"")`. You could also see how the first couple rows look like by coding `head(iris)`.

## Example 1: Data Summary Page - Explaining What the Iris Dataset is About

```
ui <- fluidPage(
  titlePanel("Iris Data Description"),
  sidebarLayout(
    sidebarPanel("Some Basic Information"),
    mainPanel(
      img(src = "https://i.pinimg.com/originals/06/b8/5c/06b85ccba2a1799ac42173bdb0476878.jpg", height = 300, wid
th = 400),
      h3(p(strong("Iris Data Dictionary"))),
      p("We have the Iris dataset that contains 3 classes of iris flowers"),
      em("Here are the attribute information regarding the iris dataset"),
      p("1. sepal length in cm"),
      p("2. sepal width in cm"),
      p("3. petal length in cm"),
      p("4. petal width in cm"),
      p("5. class: Iris Setosa, Iris Versicolour, Iris Virginica")
    )
  )
)

server <- function(input, output) {
  output <- renderText({
    "input text"
  })
}

shinyApp(ui = ui, server = server)
```

first image




iris

- The Data Summary Page is similar to the data dictionary text file that we learned in class, except this can be explicitly shown in Shiny App. A pain point that I encounter in a regular data dictionary is its poor readibility. Readers are often presented with a data dictionary that is cramed with text with little to none visually appealing components. To resolve the issue, I did some research on adding beautiful images to the Data Summary Page on Shiny App by relying on `img()` in the ui layout.

# Reproducing the Above Data Summary Page

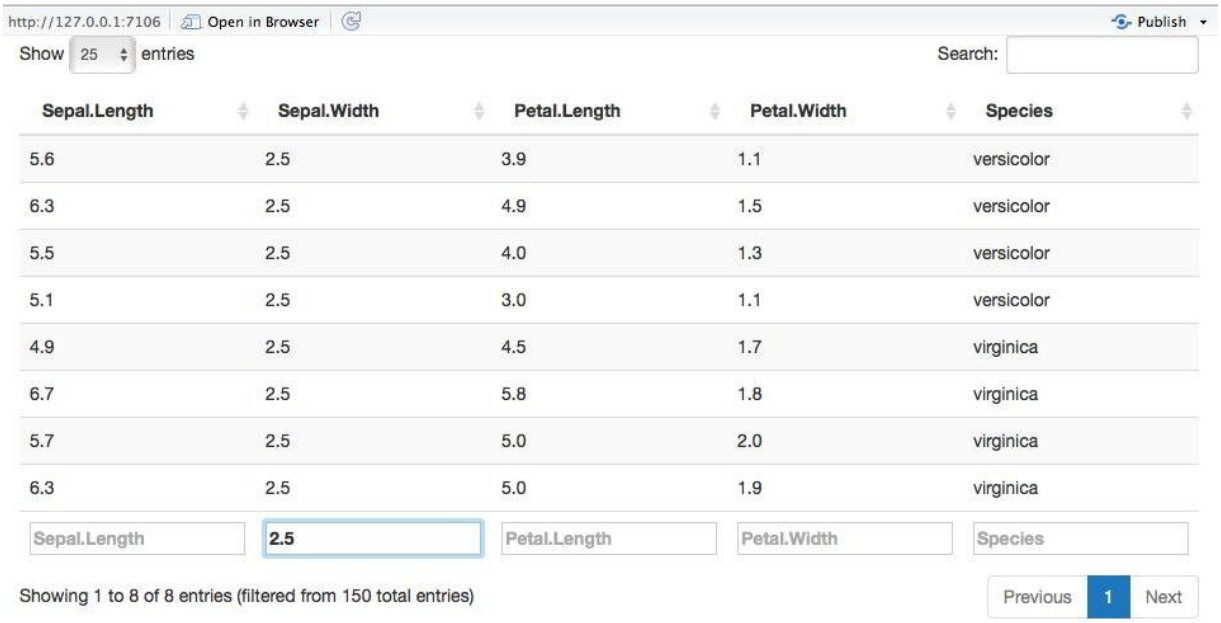
- Step 1: Define the user interface (ui), including the titlePanel and sidebarPanel that label the title and purpose of the iris dataset.
- Step 2: In mainPanel, add an image of the iris flower by using `img(src = , height = , width = )`, where the image url goes into src, and the height and width define the size of the image. Be mindful also of the code chunks such as `h3(p(strong("Iris Data Dictionary")))` that are used (which I learned from the R Studio website), where h3 determines the size of the text (2rd level), p indicates the beginning of a new paragraph, and strong indicates the bolding of certain words. `em()`, similarily, commands certian words to be italicized.
- Step 3: Here, we define the server, which relies on renderText since our output is plainly data summary texts that describe the overall content (including variables, etc.) of the iris data set.

# Example 2: Data Table - Display Iris Dataset in an Intelligent Manner

```r
# Data Table for Iris
  shinyApp(
    ui = fluidPage(
      fluidRow(
        column(12,
           dataTableOutput('table')
        )
      )
    ),
    server = function(input, output){
      output$table <- renderDataTable(iris)
    }
  )
```

second image

- So far in the course, we learned to display data in 3 formats under Shiny App: histogram, bar chart, as well as scatter plot. To me, having a data table is just as, if not more important for displaying complext data results. After searching for a function that would implement such results on RDocumentation, I used it in the code chunk above to display results. As seen in the image below, it is possible to enter specific sepal length (note: sepal is a part of flowers), petal lengths, etc. to see the corresponding results for all variables. Having a function as such is convenient to benchmark results with 1 variable held constant and see how other varaibles are affected. In the image below, with the sepal width as 2.5 (where sepal width is the control variable), we see how sepal length, petal length, and petal width all vary slighty and that there are only two species of iris (versicolor and virginica) that fall under this specification.




Iris Table

# Reproducing the Above Iris Data Table

- Step 1: We begin with the ui component and use dataTableOutput to specify that our user interface layout would look like a data table.
- Step 2: We set server and define the output to be a table in `output$table`, explicitly calling the iris dataset “iris” in `renderDataTable` to return a data table with searchable variables.
- Step 3: Run code chunk and data table such as the one below returns.

# Example 3: Row Selection and Scatter Plot

```
colnames(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

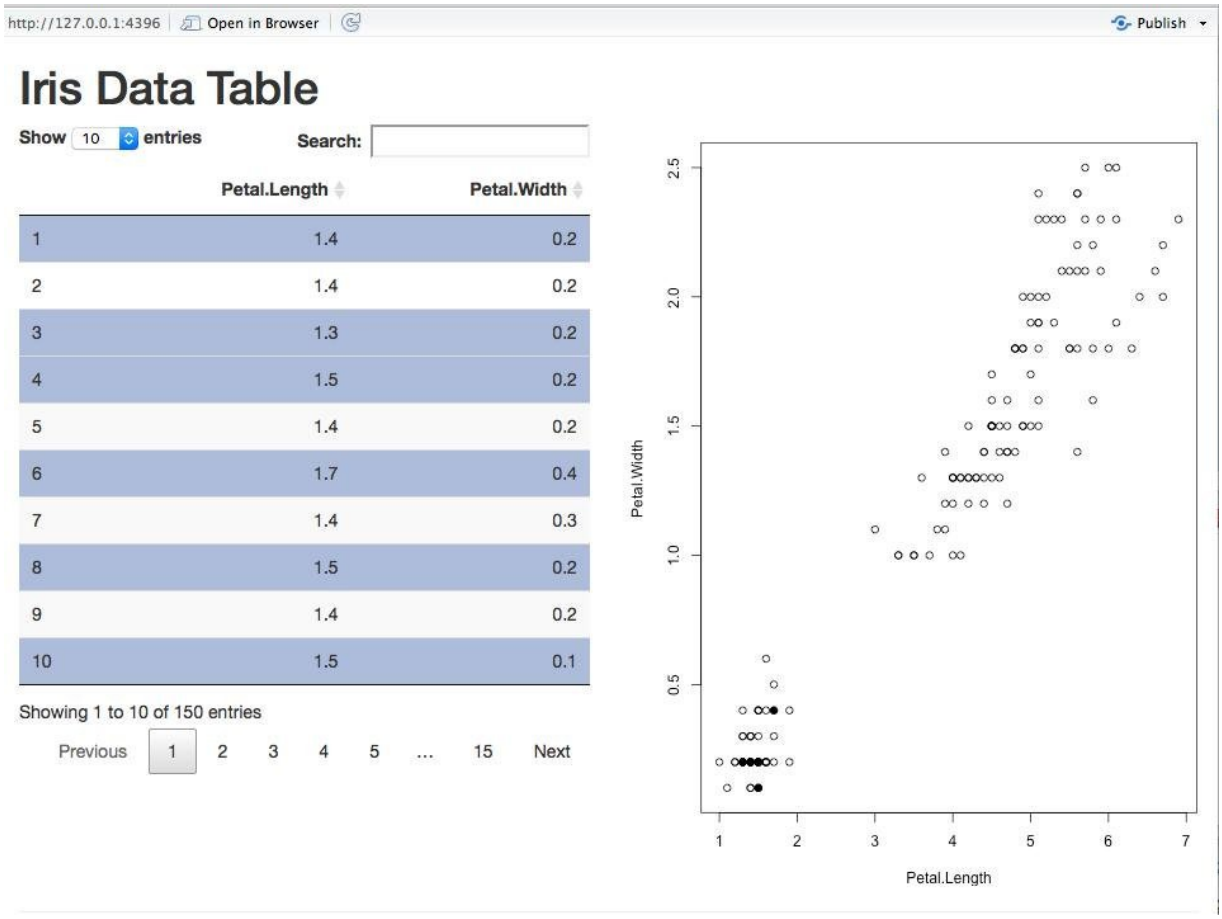
```
# select specific rows in the iris dataset
iris_two_cols <- iris[, c("Petal.Length", "Petal.Width")]
iris_two_cols
```

select two columns

```
ui <- fluidPage(

  title = 'Select Table Rows',

  h1('Iris Data Table'),

 fluidRow(
    column(6, DT::dataTableOutput('x1')),
    column(6, plotOutput('x2', height = 600))
  ),

  hr(),

  fluidRow(
    column(9, DT::dataTableOutput('x3')),
    column(3, verbatimTextOutput('x4'))
  )
)

server <- shinyServer(function(input, output, session) {

  output$x1 = DT::renderDataTable(iris_two_cols, server = FALSE)

  # highlight selected rows in the scatterplot
  output$x2 = renderPlot({
    s = input$x1_rows_selected
    par(mar = c(4, 4, 1, .1))
    plot(iris_two_cols)
    if (length(s)) points(iris_two_cols[s, , drop = FALSE], pch = 19, cex = 1)
  })

  # print the selected indices
  output$x4 = renderPrint({
    s = input$x3_rows_selected
    if (length(s)) {
      cat('These rows were selected:\n\n')
      cat(s, sep = ', ')
    }
  })
})
shinyApp(ui = ui, server = server)
```

code

- So far we learned the rudimentary concepts of Shiny App in class, where we take a dataset and translate it into some kind of graphical / visual representation. What if we want to go beyond that and display results that are more advanced? With this goal in mind, I did additional research on the DT package in R (explained in the introduction on what it is) and found additional sources on baoruidata that demonstrates additional usuage. By coding `DT::datatable()`, we are able to customize DataTables and display them in advanced formats. As seen in the image below, by selecting specific rows in the data table, we are able to simultaneously display the corresponding points on the scatter plot, specified by ones where the points are filled (the non-selected points display as transparent colors). This is not only a useful way to more conveniently display data, but it also effectively shows how different points in the dataset display on a scatter plot in real time. Most importantly, the row selection is a concept that is not extensively covered but would benefit many students.




scatter plot from selected rows

# Reproducing the Above Iris Data Table

- Step 1: In contrary to the first two demos in this blog post, I will be selecting two columns (Petal Length and Petal Width) by coding `iris_two_cols <- iris[, c("Petal.Length", "Petal.Width")]`, as a scatter plot only takes in two variabales to display the graphical results.
- Step 2: Define ui. By using code chunks such as `h1('Iris Data Table'), fluidRow( column(6, DT::dataTableOutput('x1')), column(6, plotOutput('x2', height = 600)) ),` we define sepcificly how the layout would look like. Here, th height, and the graphical output (table & graph) are specified using `DT::dataTableOutput` and `plotOutput`, respectively.
- Step 3: Define server by highlighting selected rows in the scatterplot. This is perhaps the most important step in this graphical output. Here, by defining `renderPlot(s = input$x1_rows_selected)`, we command R to produce a scatter plot that links specific rows in our data table to specific points on the scatter point. In the if statement, by specifying `pch` and `cex`, we define the level of transparency and the size of the points that would be highlighted in the scatter plot.
- Step 4: Run app and play around with selected rows!

# Conclusion

In conclusion, the purpose of the blog post is to explore alternative methods to analyze data using Shiny App. In the first section, we went over creating a data summary page, which not only contains a vivid image of an iris, making the data dicionary more visually appealing, but also displays other important information of the dataset, such as the variables that are involved. In the data table and scatter plot, I did reserach on what is already covered in class and added additional twists to them. With the DT package, we saw that it is possible to enter values for certain variable and see the corresponding values for all other variables. With the scatterplot, we are able to see the selected rows being displayed in real time on the scatter plot. These alternative methods to exploring data ultimately make the data analysis process even more comprehensive than what we already covered in class.

# Further Discussion

- In the future, I would like to see if it is possible to make products on Shiny App more visually appealing. Something that I love from this class is ggplot2 and how aesthetic it looks. In the future, I would like to take what I learned from Shiny App and add more colorful components to them.

# Refernces Used


- https://shiny.rstudio.com/tutorial/written-tutorial/lesson2/
- http://www.sthda.com/english/wiki/r-built-in-data-sets
- https://www.youtube.com/watch?v=6F5_jbBmeJU&ab_channel=AbhinavAgrawal
- https://yihui.shinyapps.io/DT-rows/
- https://rstudio.github.io/DT/
- https://datatables.net/examples/api/select_row.html
- https://www.rdocumentation.org/packages/DT/versions/0.2/topics/dataTableOutput

- http://www.baoruidata.com/examples/018-datatable-options/