# R-Shiny

Alastair Kerr
Bioinformatics Core Facility Manager


Wellcome Trust
Centre for Cell Biology

# Where to get help

- General
  - http://shiny.rstudio.com
    - http://shiny.rstudio.com/tutorial/
- Administrators Guide
  - http://rstudio.github.io/shiny-server/latest/

# Motivation for adopting R-Shiny

- I run an analysis service for biologists
  - Tables and graphs are key deliverables
- Excel has no place in bioinformatics
  - Truncating large data sets
  - Limited: applications, programmable ...
- Static images are often more work for multifaceted data
  - I have created over 300 images in the past for a single collaboration
- Server side memory and cycles benefit visualisation on low spec'ed users computers

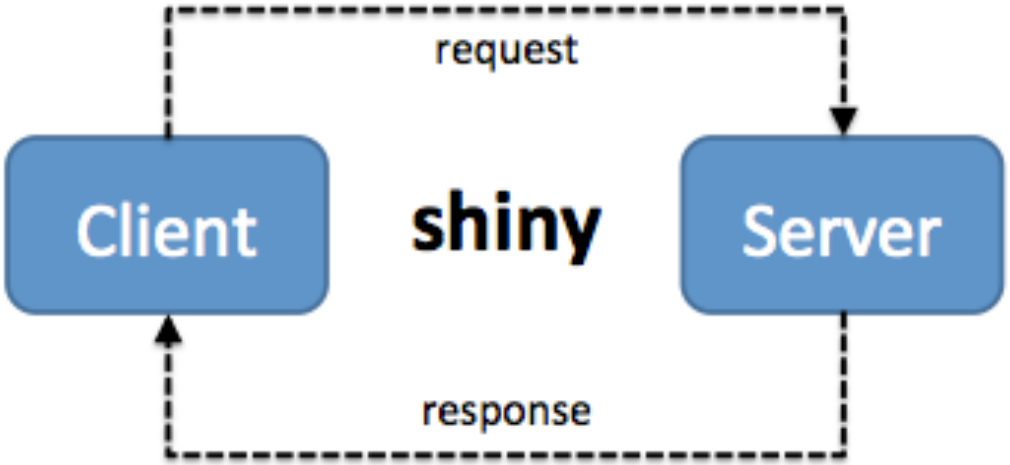
# Example Cases

- Plot selected facets of multi faceted data
- Compare latest samples to any previously run
  - R reads files in a directory but I could also have used a database
- Interactive spreadsheets
  - can be combined with a plot on the same page

# Example: using Shiny in our Core Facility

- Live Demo of R-shiny using ggplot2
- and  shiny tables

# Client-server infrastructure




- **Shiny app Server**
  - Was nodejs
  - Now a dedicated binary
    - “shiny-server” debian package

- **Client**
  - Web browser

# Two server versions

- **Open Source Edition**
  - Great for hosting lightweight public applications
  - Does not support authentication or SSL
  - Single R process per application
- **Paid [Professional] edition**
  - Supports authentication and SSL
  - Includes admin dashboard with both realtime and historical performance data
  - Can use multiple R processes per app
- Shiny app hosting service also provided by **Shinyapps.io**

# Simple Installation

Ubuntu/Debian

- Shiny Apps
  - sudo su -  -c "R -e \"install.packages('shiny', repos='http://cran.rstudio.com/')\""
- Server:
  - sudo apt-get install gdebi-core
  - wget http://download3.rstudio.org/ubuntu-12.04/x86_64/shiny-server-1.3.0.403-amd64.deb
  - sudo gdebi shiny-server-1.3.0.403-amd64.deb
  - Default app location: “/srv/shiny-server”
    - Default port 3838:
      - URL: http://hostname:3838/DIRECTORY_NAME

# A basic Shiny app

**Key files in app directory**

- **ui.R**
  - Contains all the code to generate the GUI
  - Run ONCE at start of the app
  - contains the shinyUI() function
- **server.R**
  - Contains the "shinyServer()" function
    - Code **outside** this function is run ONCE at start of app
- **global.R [optional]**
  - Objects created here are visible to ui.R and server.R
  - Limited benefit
  - Server caching may not detect changes in global.R

# TWO key objects

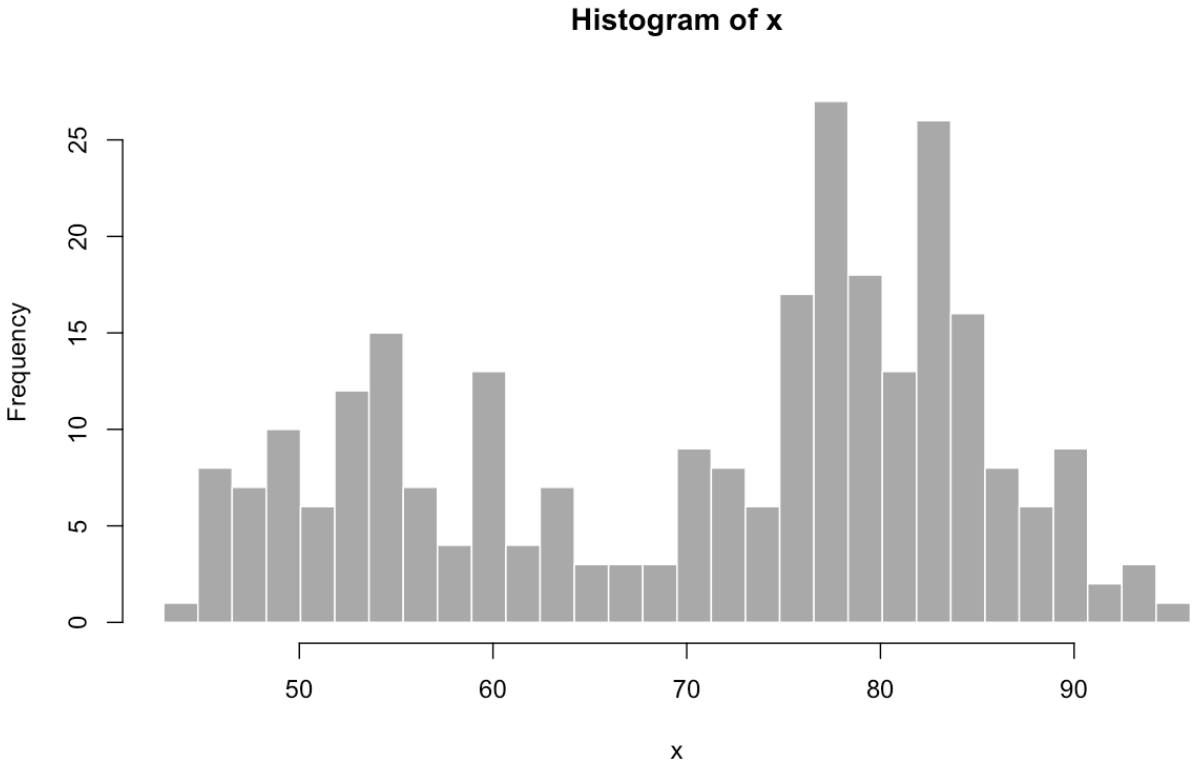
- **input**
  - Created from ui.R
    - Items set as input$ITEM
  - Used by server.R
- **output**
  - Created from server.R
  - Used by ui.R
    - used for plotting and reactive elements in the UI

# Tutorial App

## Hello Shiny!

Number of bins:

1 30 50



# ui.R

```
library(shiny)
# Define UI for application that draws a histogram
shinyUI(fluidPage(
  # Application title
  titlePanel("Hello Shiny!"),
  # Sidebar with a slider input for the 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")
    )
  )
))
```

# Customising the UI

Widgets
http://shiny.rstudio.com/gallery/widget-gallery.html

Layout
http://shiny.rstudio.com/articles/layout-guide.html
includes examples on column and wellpanel

# server.R

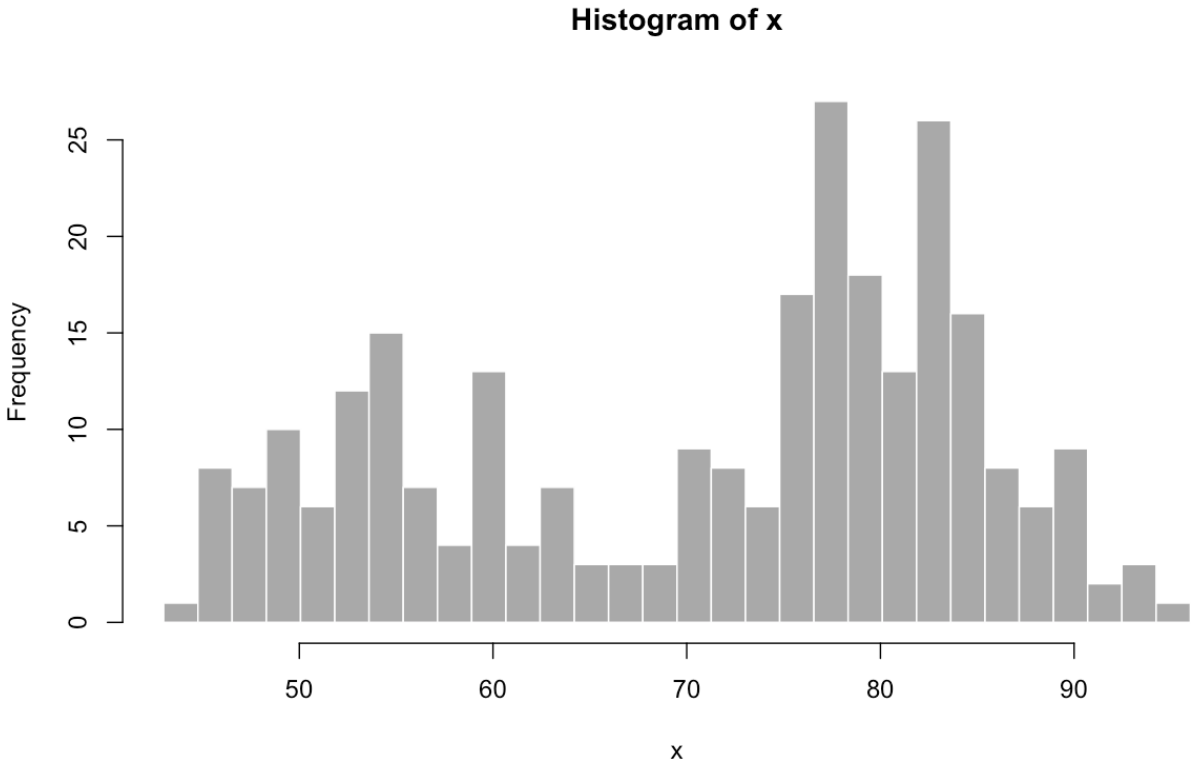
```
library(shiny)

# Define server logic required to draw a histogram
shinyServer(function(input, output) {
  # Expression that generates a histogram. The expression is
  # wrapped in a call to renderPlot to indicate that:
  #  1) It is "reactive" and therefore should re-execute automatically
  #     when inputs change
  #  2) Its output type is a plot

  output$distPlot <- renderPlot({
    x    <- faithful[, 2]  # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

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

# Tutorial App

## Hello Shiny!

Number of bins:

1 30 50



# Render Functions in server.R

| | |
|---|---|
| **renderPlot** | plots |
| **renderTable** | data frame, matrix, other table like structures |
| **renderUI** | a Shiny tag object or HTML |
| **renderImage** | images (saved as a link to a source file) |
| **renderText** | character strings |
| **renderPrint** | any printed output |

# Corresponding ui.R functions

| | |
|---|---|
| **plotOutput** | plot |
| **tableOutput** | table |
| **htmlOutput** | raw HTML |
| **uiOutput** | raw HTML |
| **imageOutput** | image |
| **textOutput** | text |
| **verbatimTextOutput** | text |

# “Reactive” functions/expressions

- Create reactive expressions with
  - **reactive({ })**
- Will **re-calculate** if their **input has changed**
- Use: Call from
  - **Render** function
  - or **other reactive** functions
- Examples
  - Updating data in a graph
  - Creating new inputs for ui.R widget
  - Caching data to use in another reactive function

```
##ui.R
htmlOutput("selectUI1"),
checkboxInput(inputId = "all",  label = "L1", value = FALSE),
checkboxInput(inputId = "DNA",  label = "L2",  value = TRUE),
##server.R
output$selectUI1 <- renderUI({
       selectInput("Experiment1", "Select first set to plot",
                   searchResult() )
   })
searchResult<- reactive({
    tmp <- myfiles
    if(input$all){tmp <- subset(tmp, grepl("all", tmp)) }
    if(input$DNA){tmp <- subset(tmp, grepl("DNA", tmp)) }
    tmp
})
```

# Another code example if time permits

# Take care with

- Commas, brackets, commas, commas and commas
  - did I mention commas….
- environment() for ggplot using local variables
  - ggplot(..., environment=environment())

# So much more:

- For Inspiration
  - http://shiny.rstudio.com/gallery/
- Can use html5 in ui.R to create custom pages
- htmlwidgets: JavaScript data visualization for R
  - http://www.htmlwidgets.org/
  - Use JavaScript visualization libraries at the R console, just like plots
  - Embed widgets in R Markdown documents and Shiny web applications
  - Develop new widgets using a framework that seamlessly bridges R and JavaScript