Advanced data visualisation in R: Shiny

Nadia and Michael

23/11/2016

Introduction

What is this session about:

- a little bit of appreciation towards basic plotting in R
- some words about visualisation methods with their pros and cons
- interactive plots and web-apps with Shiny

!!! NO GGPLOT TUTORIAL (as everyone is supposed to know it from BTM :p)

To start with: this presentation and all the codes are available in my Github repository.

Data visualisation: methods

Main graphical libraries:

- graphics (`plot`) – check out Uni Pennsylvania tutorial, Harvard tutorial

Data visualisation: methods

Main graphical libraries:

- graphics (`plot`) – check out Uni Pennsylvania tutorial, Harvard tutorial
- lattice (`xyplot`) – check out Uni Pennsylvania tutorial, Deepayan Sarkar's tutorial, University of British Columbia lectures

Data visualisation: methods

Main graphical libraries:

- graphics (`plot`) – check out Uni Pennsylvania tutorial, Harvard tutorial
- lattice (`xyplot`) – check out Uni Pennsylvania tutorial, Deepayan Sarkar's tutorial, University of British Columbia lectures
- ggplot2 (`ggplot`) – check out Uni Pennsylvania tutorial, Zev Ross' cheatsheet, Harvard tutorial

Basic data visualisation: reminder

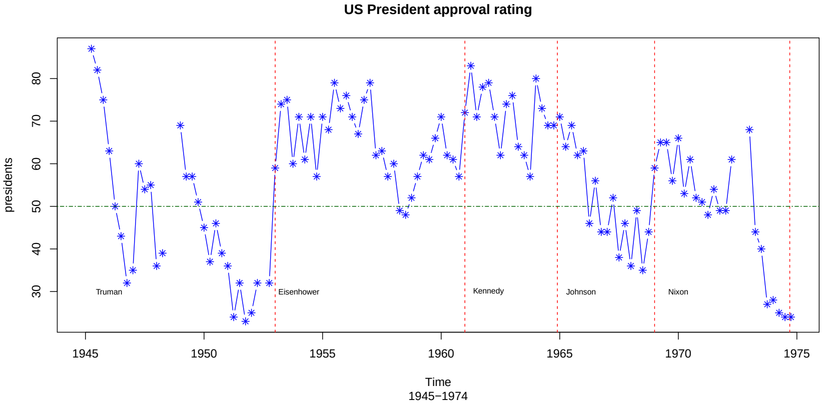
DIY: Plot US president quarterly approval ratings in 1945-1974.

Add lines for 50% apporval and separate different presidents.

(Dataset `presidents`, president change: Jan. 1953, Jan. 1961, Nov. 1964, Jan. 1969, Aug. 1974).

Basic data visualisation: reminder

```
plot(presidents, type="b", cex=0.2, pch=8,
     main = "US President approval rating",
     sub = "1945-1974", col="blue")
abline(v = c(1953,1961,1964.9,1969,1974.7),col="red", lty=2)
abline(h = 50, col="darkgreen", lty=6)
text(x = c(1945,1953,1961,1964.9,1969)+1, y = 30, cex = 0.7,
     labels = c("Truman", "Eisenhower", "Kennedy", "Johnson", "Nixon"))
```



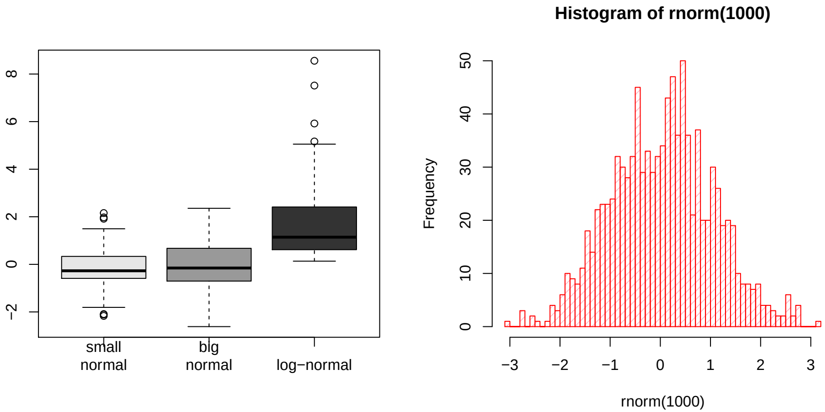
Basic data visualisation: reminder

DIY: visualize with boxplots: small sample of normal distribution, large samples of normal distribution and a sample of log-normal distribution.

DIY: make a histogram of random sample from normal distribution.

Basic data visualisation: reminder

```
par(mfrow = c(1,2))
# Boxplot of normal and lognormal distributions
boxplot(rnorm(50),rnorm(100),rlnorm(100),
        names=c("small\nnormal","big\nnormal","log-normal"),
        col=grey(c(0.9, 0.6, 0.2)),border = "black")
# histogram of normal distribution
hist(rnorm(1000), breaks = 50, density = 15, col="pink", border = "red")
```

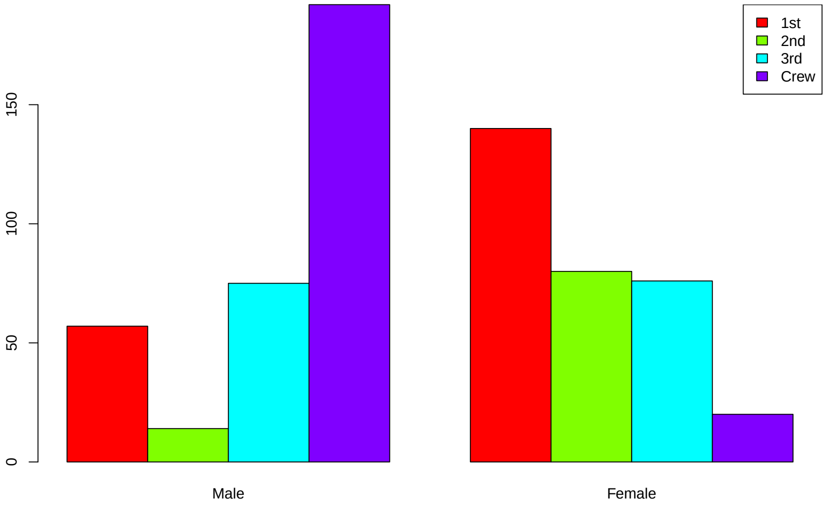


Basic data visualisation: reminder

DIY: visualize with bar charts the numbers of men and women survived on Titanic per class (dataset Titanic). Don't forget to add the legend.

Basic data visualisation: reminder

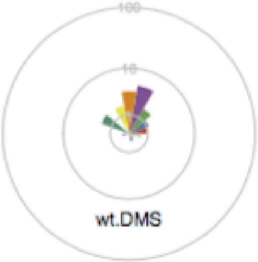
```
# Titanic survival per class
barplot(Titanic[,,Age="Adult",Survived="Yes"],beside = T,col=rainbow(4))
legend("topright",legend = dimnames(Titanic)$Class, fill = rainbow(4))
```



Basic data visualisation: reminder

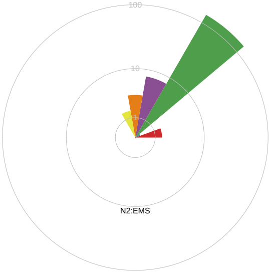
DIY: Visualize spectra of mutational effects provided in the file `spectra.csv` with ggplot2 as a barplot and as a piechart preserving the scale.

```r
load("shiny.plotting.Rdata")
```



Basic data visualisation: reminder

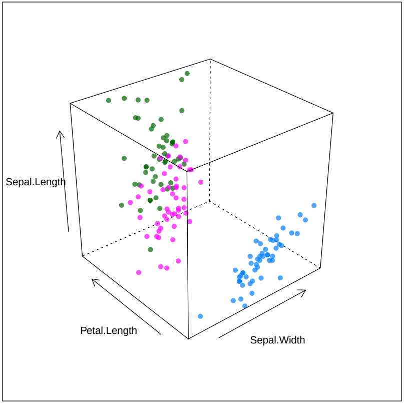
```r
myplot <- function(effects, gen, mut) {
  colors <- c("#CD2A2B","#3D77A6","#4E9E4C","#8A4E93","#E57E17",
              "#E5E540","#95552F","#248E6F","#C36015","#6F6BA1",
              "#D03685","#629527","#CF9F16","#956F26","#5C5C5C",
              "#E2A8A4","#A5BACC","#BAD3B5")
  par(xpd = NA, mar = c(5,5,5,5))
  c <- t(effects)
  s <- stars(rescale(c), draw.segments=TRUE, col.segments=colors, scale=FALSE, col.lines=0, lty=0, labels=
             locations=data.frame(Var1=1,Var2=1))
  for(k in c(0:round(log10( max(effects) )))){
    l = 10**k
    symbols(1, 1, circles=rescale(l), add=TRUE, inches=FALSE, fg="grey")
    text(1, 1 + rescale(l), l, col="grey")
  }
  text(s[,1],s[,2]-.5,paste(gen,":",mut,sep=""), pos=1)
}
myplot(E["EMS",],gen="N2",mut="EMS")
```



Data visualisation: lattice

Just an example of 3D plot with `Iris` dataset

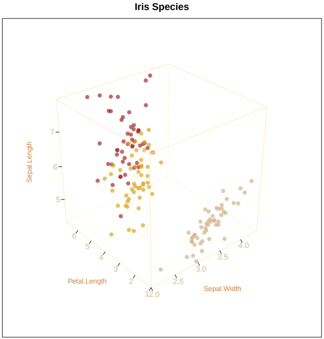
```
library(lattice)
library(plyr)
cloud(Sepal.Length~Sepal.Width*Petal.Length,iris,pch=16,alpha=0.7,cex=1,groups=iris$Species)
```



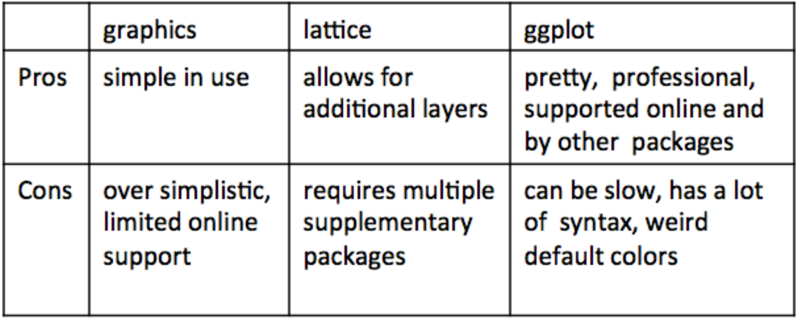
Data visualisation: lattice

Just an example of 3D plot with Iris dataset

```
cloud(Sepal.Length~Sepal.Width*Petal.Length,
      iris,pch=16,alpha=0.7,cex=1,groups=iris$Species,
      main="Iris Species",scales=list(arrows=FALSE,
                                      x=list(cex=0.9,tck=0.5,col="wheat3"),
                                      y=list(cex=0.9,tck=0.5,col="wheat3"),
                                      z=list(cex=0.9,tck=0.5,col="wheat3")),
      par.settings=list(box.3d=list(col="wheat1"),
                        par.xlab.text=list(cex=0.8,col="tan3"),
                        par.ylab.text=list(cex=0.8,col="tan3"),
                        par.zlab.text=list(cex=0.8,col="tan3",rot=90)),
      col=c("tan","goldenrod","brown"))
```

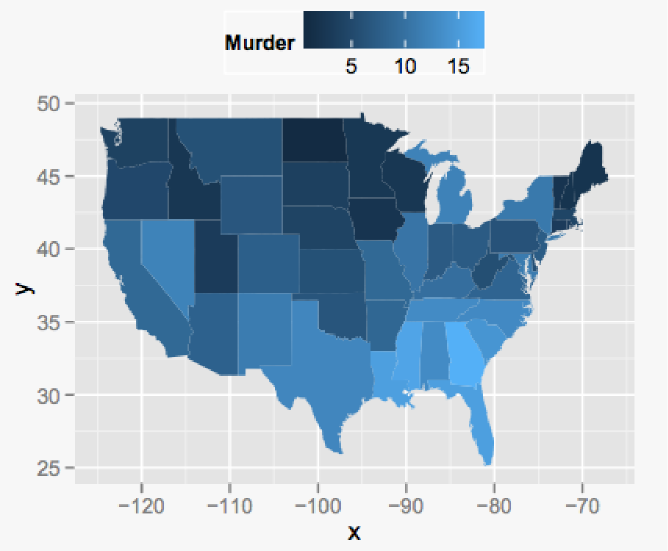


Data visualisation: pros and cons



	graphics	lattice	ggplot
Pros	simple in use	allows for additional layers	pretty, professional, supported online and by other packages
Cons	over simplistic, limited online support	requires multiple supplementary packages	can be slow, has a lot of syntax, weird default colors

Data visualisation: wonderful ggplot



Source

Some additional hints for spatial visualization

Packages:

- rworldmap
- maps
- `ggmap`

Data visualisation: summary

What is Shiny

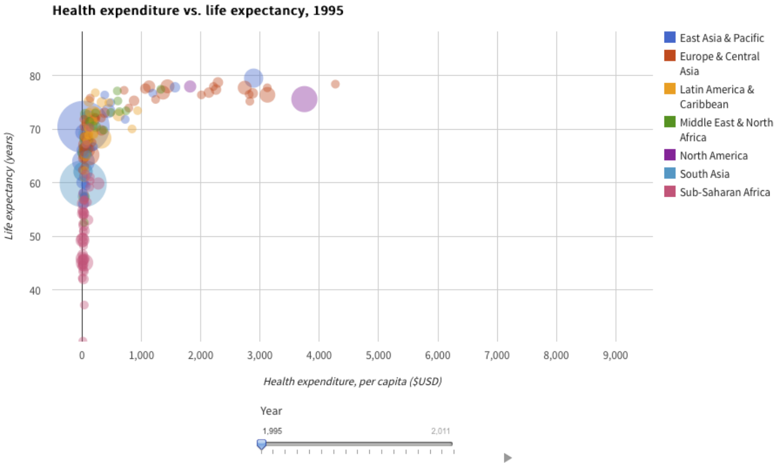
Shiny – web application framework for R.

- it allows to make pretty interactive applications
- it does not require any CSS, HTML or JavaScript skills

```r
install.packages("shiny")
library(shiny)
```

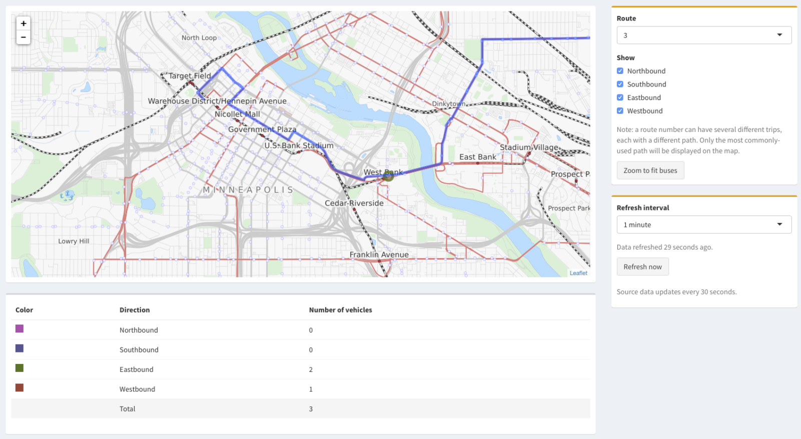
Shiny Showcase:
www.rstudio.com/products/shiny/shiny-user-showcase/

Shiny examples



Source

Shiny examples



Route

3 ▾

Show

☑ Northbound

☑ Southbound

☑ Eastbound

☑ Westbound

Note: a route number can have several different trips, each with a different path. Only the most commonly-used path will be displayed on the map.

Zoom to fit buses

Refresh interval

1 minute ▾

Data refreshed 29 seconds ago.

Refresh now

Source data updates every 30 seconds.

Color	Direction	Number of vehicles
🟪	Northbound	0
🟦	Southbound	0
🟩	Eastbound	2
🟥	Westbound	1
	Total	3

Source

Shiny examples: Hello Shiny

Hello Shiny - draw histograms

```
library(shiny)
runExample("01_hello")
```

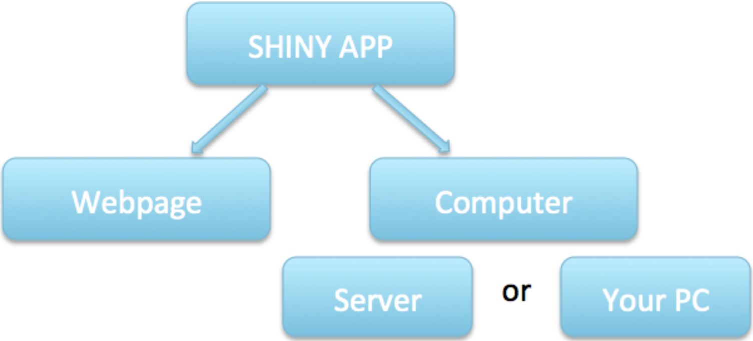
Print dataset in text form

```
runExample("02_text")
```

Reactivity add-on

```
runExample("03_reactivity")
```

How does Shiny work



How to use Shiny

Shiny applications have two components:

- user-interface definition (source file named `ui.R`)
  - HTML (written with Shiny functions) responsible for layout
  - ordering of things in the app
- server script (source file named `server.R`)
  - logic of the app
  - instructions for reaction to user actions

Note that inputs and outputs are connected together "live": changes are propagated immediately (without reloading the whole page).

Shiny also uses **reactive** programming: only the necessary parts of the code will be re-executed in response to input data changes.

Shiny app template

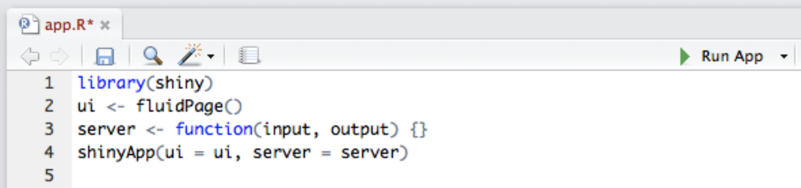
```r
library(shiny)
# Initialise empty IU
ui <- fluidPage()
# Initialise empty server
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

NB: this script should be saved as **app.R**, otherwise Shiny will not recognize it.

NB2: You can do it with RStudio: *File* > *New Project* > *New Directory* > *Shiny Web Application*.

Running a Shiny app

Press “Run App” button!



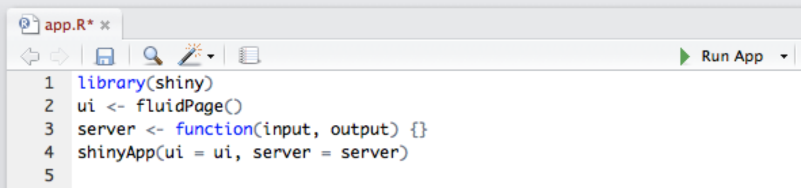
```r
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)

```

Or just run `shiny::runApp()`,

- What happens?

Running a Shiny app

Press “Run App” button!



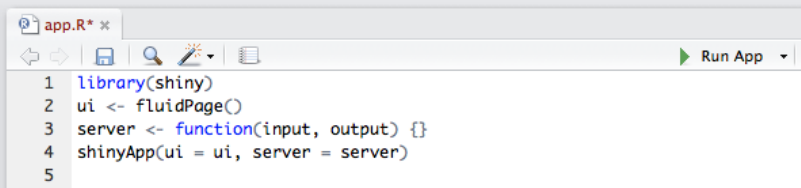
```r
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)

```

Or just run `shiny::runApp()`,

- What happens?
- Console prints where is the server your R studion is listening to

Running a Shiny app

Press “Run App” button!



```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

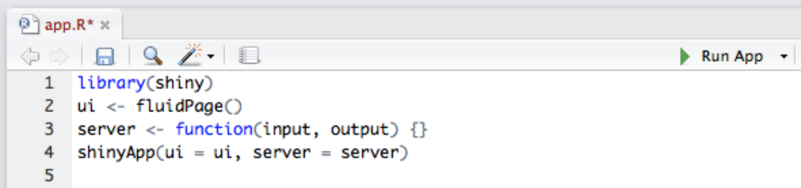
Or just run `shiny::runApp()`,

- What happens?

- Console prints where is the server your R studion is listening to
- “Stop” button appears - RStudio is busy running the app

Running a Shiny app

Press “Run App” button!



```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)

```

Or just run `shiny::runApp()`,

- What happens?

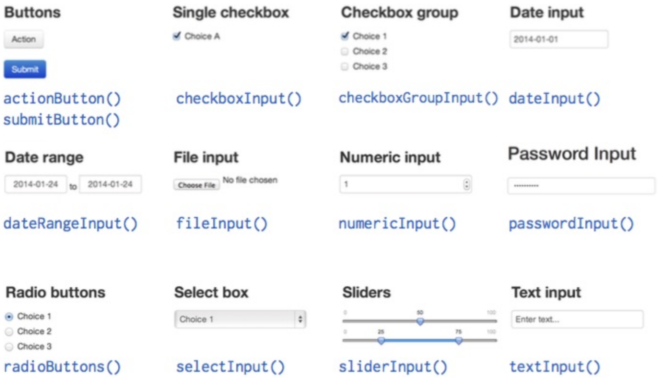
- Console prints where is the server your R studion is listening to
- “Stop” button appears - RStudio is busy running the app
- Click “Stop” or press *Esc*.

Building an app in Shiny

- Add elements to the app as arguments to `fluidPage()`

```
ui = fluidPage(
  titlePanel(...),   # set up a title
  sidebarLayout( # not necessary, creates a sidebar where you ca
    sidebarPanel(
      # Input*() functions (coming!)
    ),
    mainPanel(
      # *Output() functions (coming!)
    ),
    position = (...) # align it wherever you like
  )
)
```

Building an app in Shiny: Inputs



`actionButton()` `submitButton()`

`checkboxInput()`

`checkboxGroupInput()`

`dateInput()`

`dateRangeInput()`

`fileInput()`

`numericInput()`

`passwordInput()`

`radioButtons()`

`selectInput()`

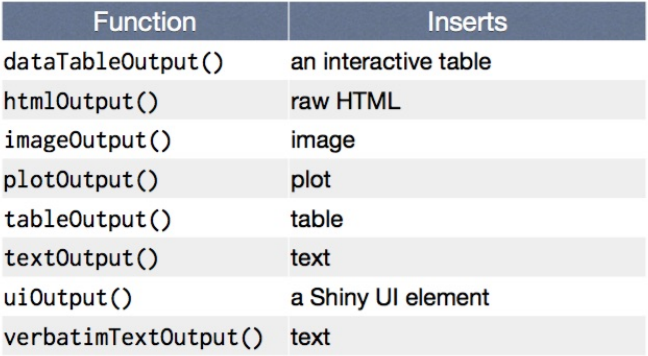
`sliderInput()`

`textInput()`

Syntax:

```r
selectInput(inputId = "gen",
            label = "Choose genetic factor",
            value = genes)
```

Building an app in Shiny: Outputs



Function	Inserts
`dataTableOutput()`	an interactive table
`htmlOutput()`	raw HTML
`imageOutput()`	image
`plotOutput()`	plot
`tableOutput()`	table
`textOutput()`	text
`uiOutput()`	a Shiny UI element
`verbatimTextOutput()`	text

```
plotOutput("myplot")
```

Building an app in Shiny: User interface altogether

```r
ui <- fluidPage(
  titlePanel("Mutational signatures catalogue"),
  selectInput(inputId = "gen",
              label = "Choose genetic factor",
              choices = genotypes),
  plotOutput("myplot")
)
```

General workflow:

- begin with template
- create reactive inputs
- create reactive outputs
- assemble outputs from inputs in server functions

Building an app in Shiny: Server

3 main rules:

- Save objects to display to `output$`

```
output$myplot
```

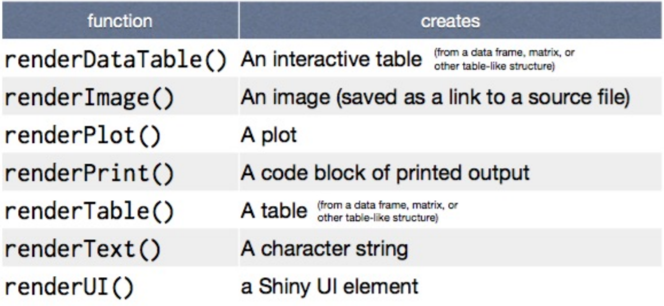
- Build objects to display with `render...()`: it is "reactive" and therefore should be automatically re-executed when inputs change

```
output$myplot <- renderPlot({
  # ...
})
```

- Access input values with `input$`

```
input$gen
```

Building an app in Shiny: Server and rendering



function	creates
`renderDataTable()`	An interactive table (from a data frame, matrix, or other table-like structure)
`renderImage()`	An image (saved as a link to a source file)
`renderPlot()`	A plot
`renderPrint()`	A code block of printed output
`renderTable()`	A table (from a data frame, matrix, or other table-like structure)
`renderText()`	A character string
`renderUI()`	a Shiny UI element

Building an app in Shiny: Server commands altogether

```r
server <- function(input, output) {
  output$myplot <- renderPlot({
    myplot(E[input$gen,], gen=input$gen)
  })
}
```

Try yourself:

Task: make an app that will show spectrum of mutational effects for a given combination of factors.

It's dangerous to go alone, so take again the spectra matrix and the plotting function we made earlier:

```r
# download effect matrix
load("shiny.plotting.Rdata")
# grab the function we created earlier
source("myplot.R")
```

If you have troubles - check out the codes in my Github repository.

User-interface definition

```
ui <- fluidPage(
  titlePanel("Mutational signatures catalogue"),
  sidebarLayout(
    sidebarPanel(
      selectInput(inputId="gen",
                  label="Choose genotype:",
                  choices = genotypes,
                  selected = "N2"),
      selectInput(inputId="mut",
                  label="Choose mutagen:",
                  choices = c("NA",mutagens),
                  selected = "NA")),
  # Show a plot of the generated distribution
    mainPanel(plotOutput("myplot"))
  )
)
```

Server script

```r
server <- function(input, output) {
  output$myplot <- renderPlot({
    if (input$mut!="NA") {
      effects <- E[paste(input$gen,input$mut,sep="."),]
    } else {
      effects <- E[input$gen,]
    }
    myplot(effects, input$gen, input$mut)
  })
}
```

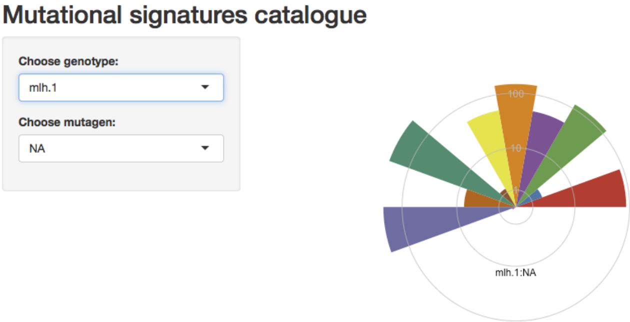
Now RUN SHINY

Click the “Run App” button!

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

- What do you see?

How it should look like



Mutational signatures catalogue

Choose genotype:

mlh.1

Choose mutagen:

NA

Sharing your apps

ShinyApps – can be tied to Github account

Step 1. Install `rsconnect` package

```
install.packages('rsconnect')
```

Step 2. Authorize

```
rsconnect::setAccountInfo(name='USERNAME',
                          token='YOUR_TOKEN',
                          secret='<SECRET>')
```

Step 3. Deploy

```
library(rsconnect)
rsconnect::deployApp('path/to/your/app')
```

Useful links

- Tutorials on various R graphics:
  - Uni Pennsylvania tutorial
  - Harvard tutorial
  - University of British Columbia lectures
  - Zev Ross' cheatsheet
  - Harvard tutorial
- Tutorials on Shiny:
  - Official website
  - Developer tutorial – mostly used in this presentation
  - ShinyHelper
  - Shiny in RMarkdown
- Hints and tricks:
  - Understand Reactivity
  - Debugging Shiny apps
  - Solving common problems

The end!