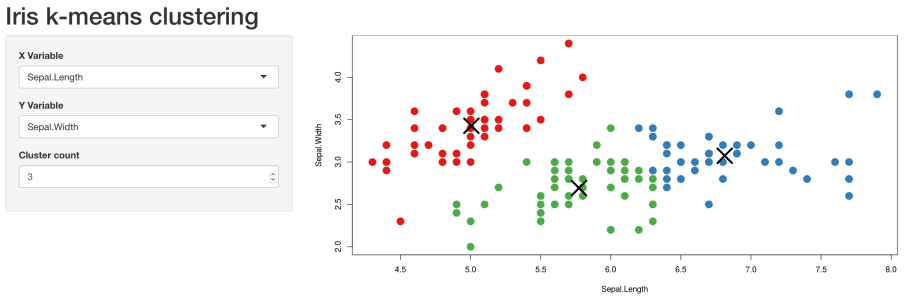
Statistical Computing and Data Visualization in R

Lecture 10

Interactive visualizations and Markup

Interactive Graphs: Shiny

- You can use the shiny package to create interactive apps with a friendly user interface and that are served over the internet.
- A shiny app consists of two pieces (files):
  - A user interface.
  - A set of R server instructions.



http://shiny.rstudio.com/gallery/kmeans-example.html

Interactive Graphs: Shiny

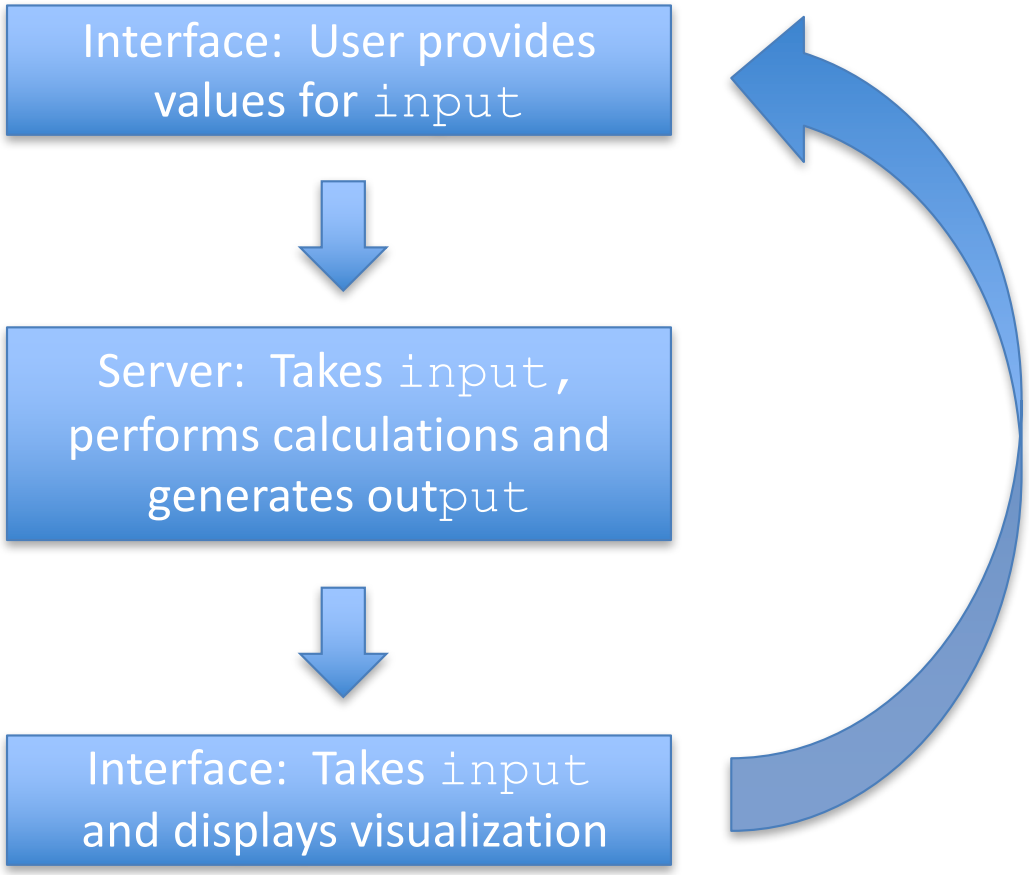
R server

- Contains all the R code that is used to generate the visualization, including any intermediate computation.
- Takes the object "`input`" generated by the interface as argument to define the parameters of the calculations
- Creates an object called "`output`" that contains the visualization and any numeric output that is to be displayed.

User interface

- Defines the layout of the interface, including the mechanisms by which the user provides information dynamically.
- Creates an object called "`input`" that is used by the server to run any R function and generate the plot.
- Displays the components of an object called "`output`" that is created by the server.

Interactive Graphs: Shiny



Go through these three steps in order when you are designing your interactive visualization!

Interactive Graphs: Shiny

R server instructions

```
function(input, output, session) {
  # Combine the selected variables into a new data
frame
  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]})

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)})

  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A",
"#984EA3", "#FF7F00", "#FFFF33", "#A65628",
"#F781BF", "#999999"))
    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4,
lwd = 4)
  })
}
```

User interface code

```
pageWithSidebar(
  headerPanel('Iris k-means clustering'),
  sidebarPanel(
    selectInput('xcol', 'X Variable',
names(iris)),
    selectInput('ycol', 'Y Variable', names(iris),
                selected=names(iris)[[2]]),
    numericInput('clusters', 'Cluster count', 3,
                 min = 1, max = 9)
  ),
  mainPanel(
    plotOutput('plot1')
  )
)
```

Interactive Graphs: Shiny

Drop down selection box

R server instructions

```
function(input, output, session) {
  # Combine the selected variables into a new data
frame
  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]})

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)})

  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A",
"#984EA3", "#FF7F00", "#FFFF33", "#A65628",
"#F781BF", "#999999"))
    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4,
lwd = 4)
  })
}
```

User interface code

```
pageWithSidebar(
  headerPanel('Iris k-means clustering'),
  sidebarPanel(
    selectInput('xcol', 'X Variable',
names(iris)),
    selectInput('ycol', 'Y Variable', names(iris),
                selected=names(iris)[[2]]),
    numericInput('clusters', 'Cluster count', 3,
                 min = 1, max = 9)
  ),
  mainPanel(
    plotOutput('plot1')
  )
)
```

Open ended numeric input box

Interactive Graphs: Shiny

R server instructions

```
function(input, output, session) {
  # Combine the selected variables into a new data
frame
  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]})

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)})

  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A",
"#984EA3", "#FF7F00", "#FFFF33", "#A65628",
"#F781BF", "#999999"))
    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4,
lwd = 4)
  })
}
```

User interface code

Names of the components of the list "`input`"

```
pageWithSidebar(
  headerPanel('Iris k-means clustering'),
  sidebarPanel(
    selectInput('xcol', 'X Variable',
names(iris)),
    selectInput('ycol', 'Y Variable', names(iris),
                selected=names(iris)[[2]]),
    numericInput('clusters', 'Cluster count', 3,
                 min = 1, max = 9)
  ),
  mainPanel(
    plotOutput('plot1')
  )
)
```

Interactive Graphs: Shiny

Preliminary calculations needed for graph

R server instructions

```
function(input, output, session) {
  # Combine the selected variables into a new data
frame
  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]})

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)})

  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A",
"#984EA3", "#FF7F00", "#FFFF33", "#A65628",
"#F781BF", "#999999"))
    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4,
lwd = 4)
  })
}
```

Preliminary calculations needed for graph

User interface code

```
pageWithSidebar(
  headerPanel('Iris k-means clustering'),
  sidebarPanel(
    selectInput('xcol', 'X Variable',
names(iris)),
    selectInput('ycol', 'Y Variable', names(iris),
                selected=names(iris)[[2]]),
    numericInput('clusters', 'Cluster count', 3,
                 min = 1, max = 9)
  ),
  mainPanel(
    plotOutput('plot1')
  )
)
```

Reactive() specifies sections of code that need to be recalculated every time there is a change in the object `input`

Interactive Graphs: Shiny

- To run a local shiny app you need to create a folder in your current working directory (e.g., `/simple_shiny_app`) where you place the two files, `server.R` and `ui.R`.
- To run the app, use `runApp("simple_shiny_app")`
  - Remember to firt load the library with `library(shiny)`.
  - Your R session will be locked until you finish working with the app.

Interactive Graphs: Shiny

- It is easy to proceed by modifying existing Apps. A number of examples, including code, are available at http://shiny.rstudio.com/gallery/
- A full tutorial for shiny is available at http://shiny.rstudio.com/tutorial/
- From within RStudio you can create a new app directly from the menu (this will create the directory and the necessary files)
  - File -> New File -> Shiny Web App

Reproducible research using knitr

- The package `knitr` allows you integrate R code (and its output) directly into LaTeX, html or a general markdown document.
- Dynamic report generation is quite useful for sharing results that are reproducible
- I will focus on creating LaTeX documents with embedded R code using Rstudio, but the principles (although not the syntax) are very similar for html and markdown.
- Using `knitr` directly with your favorite LaTeX editor is tricky …

Reproducible research using knitr

- You need to start by installing the package knitr,

  -> Tools -> Install Packages

  and setting it up as the default "weaving" package:

  -> Tools -> Global Options -> Sweave -> Weave Rnw files using

- Once the package has been installed you need to create a new markdown document
  - File -> New File -> R Sweave: LaTeX document.
  - File -> New File -> R html: html document.
  - File -> New File -> R Markdown: General markdown document.

Reproducible research using knitr

- R code can be included as a “chunk”, which looks like this:

```
<<>>=
x = seq(0, 2*pi, length=100)
y = sin(x)
print(mean(y))
@
```

Opening string

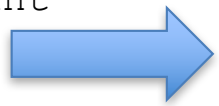
Closing string

- You can insert new chunks from the menu:

  Code -> Insert Chunk

Reproducible research using knitr

```
\documentclass{article}
\begin{document}

Let's start with a very simple
example

<<>>=
# You can have comments if you want
x = seq(0, 2*pi, length=100)
y = sin(x)
print(mean(y))
@

What do you think?

\end{document}
```



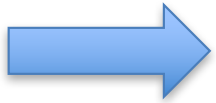
Let's start with a very simple example

```
# Ensure consistent values
x = seq(0, 2*pi, length=100)
y = sin(x)
print(mean(y))

## [1] 2.326786e-18
```

What do you think?

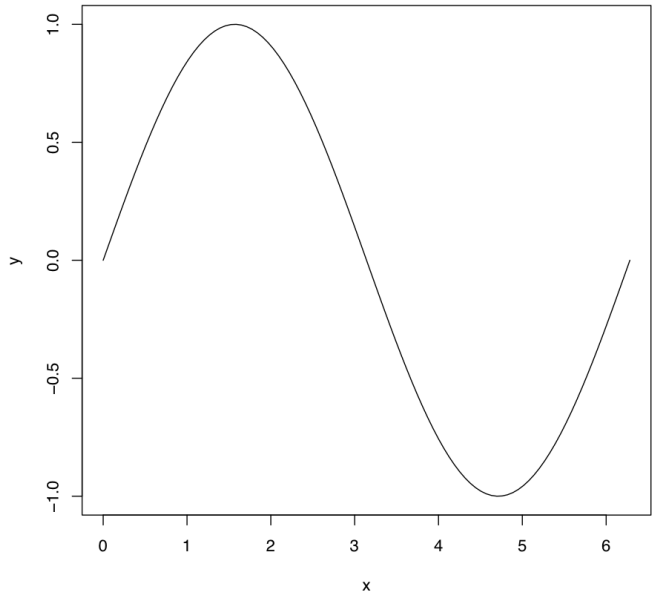
Reproducible research using knitr

```
\documentclass{article}
\begin{document}

Now with a graph

<<singraph>>=
# You can have comments if you want
x = seq(0, 2*pi, length=100)
y = sin(x)
plot(x,y)
@

What do you think?

\end{document}
```



Now with a graph

```
# Ensure consistent values
x = seq(0, 2*pi, length=100)
y = sin(x)
plot(x, y, type="l")
```



What do you think?

Reproducible research using knitr

- The first element within <<>> is the label:.
  - It is useful to have labels in long documents (specially if you want to reference a figure or the chunk itself) but, as the previous examples shows, they are not required.
- Other options can be included inside the header <<>>:
  - `<<eval=FALSE>>` lets you present the code without the performing any evaluation.
  - `<<echo=FALSE>>` does the opposite, it lets you present the results without showing the underlying code that generated it. Use `<<echo=c(2,4)>>` to echo only the second and fourth expression in a chunk.
  - `<<results=XXX>>` puts results in special environment (if `XXX` is "markup"), writes raw results into the document (if `XXX` is "asis") or pushes all output pieces to the end of a chunk (if `XXX` is "hold").

Reproducible research using knitr

- Even more options for the header <<>>:
  - `<<highlight=TRUE>>` different pieces of the code are highlighted using standard colors (e.g., objects in black, character strings in red, etc).
  - `<<tidy=TRUE>>` whether the code should be tidied up using the `tidy_source()` function.
  - `<<fig.show=XXX>>` shows plots as they are (if `XXX` is "asis"), holds all plots and outputs them at the end of the code chunk (if `XXX` is "hold") or wraps all plots into an animation (if `XXX` is "animate" and there is more than one plot.)
  - `<<fig.path=XXX>>` path of the directory where figures will be stores, relative to the current working directory.
  - `<<pdf.options(useDingbats = TRUE)>>` to use Dignbats font

Reproducible research using knitr

- Even more options for the header <<>>:
  - `<<fig.width=XXX>>` and `<<fig.height=XXX>>` control the width and height of the figures.
  - `<<out.width=XXX>>` and `<<out.height=XXX>>` control the width and height of the plot in the final output file. It could be a percentage of line width (e.g., "40%" is translated as `0.4\linewidth`).
  - `<<fig.pos=XXX>>` position of the figure ("h", "t", "b", "p").
  - `<<out.extra="angle=90">>` rotates figures 90 degrees.
  - `<<fig.cap="My caption">>` provides the text for the caption of the figure environment.
  - `<<fig.lp="fig:">>` combined with the label of the chunk, gives you the LaTeX label to use with \ref{}.

Reproducible research using knitr

- When including figures, use a separate chunk of code for each set of figures to be included in the same environment.
- You can include the results of evaluating an R expression into your text without including a chunk using `\Sexpr{}`.

Reproducible research using knitr

```
\documentclass{article}
\begin{document}

Figure \ref{fi:sinx} shows a graph of
the sine function over the $(0,\pi)$
interval:

<<sinx,fig.lp="fi:",fig.cap="The sine
function",echo=FALSE,fig.pos="h",fig.wi
dth=10,fig.height=6.5,out.width="50%",f
ig.align="center">>=
x = seq(0, 2*pi, length=100)
y = sin(x)
z = integrate(sin, 0, pi/3)

plot(x, y, type="l",
mar=c(3,3,1,1)+0.2)
@

Note that $\sin(\pi/4) =
\Sexpr{sin(pi/4)}$ and that the area
between $0$ and $\pi/3$ is
\Sexpr{round(z$value,3)}.

\end{document}
```

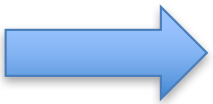


Figure 1 shows a graph of the sine function over the $(0, \pi)$ interval:

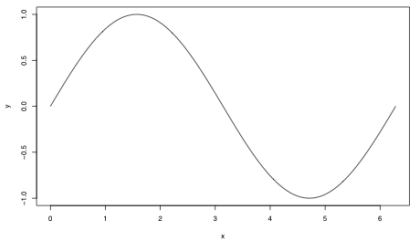


Figure 1: The sine function

Note that $\sin(\pi/4) = 0.7071068$ and that the area between 0 and $\pi/3$ is 0.5.