Wonderful world of R

Kayla Peck

April 29, 2015

1. R Markdown

R markdown is an R format that allows you to create dynamic documents. Include notes, embedded R code and plots, and produce a final output as pdf, html, or MS word.

The first step is to download the R markdown package.

```
install.packages("rmarkdown", repos="http://cran.rstudio.com/")
library("rmarkdown")
```

Text Formatting

Different symbols/characters indicate formatting for your final document. `#`'s can be used for headers (use up to 6 for heading level), while 3 `-`'s below a line of text also acts as a header format.

Asterisks can make text *italic* or **bold**.

A single dash (or asterisk with a space) acts as a bullet point.

- Use tab to create more bullet levels
  - Use 4 spaces if tab does not work

You can evaluate R code inline, for example by stating that 1 plus 1 equals 2. Additionally, you can include equations inline with `$` signs: $1 + 1 = 2$.

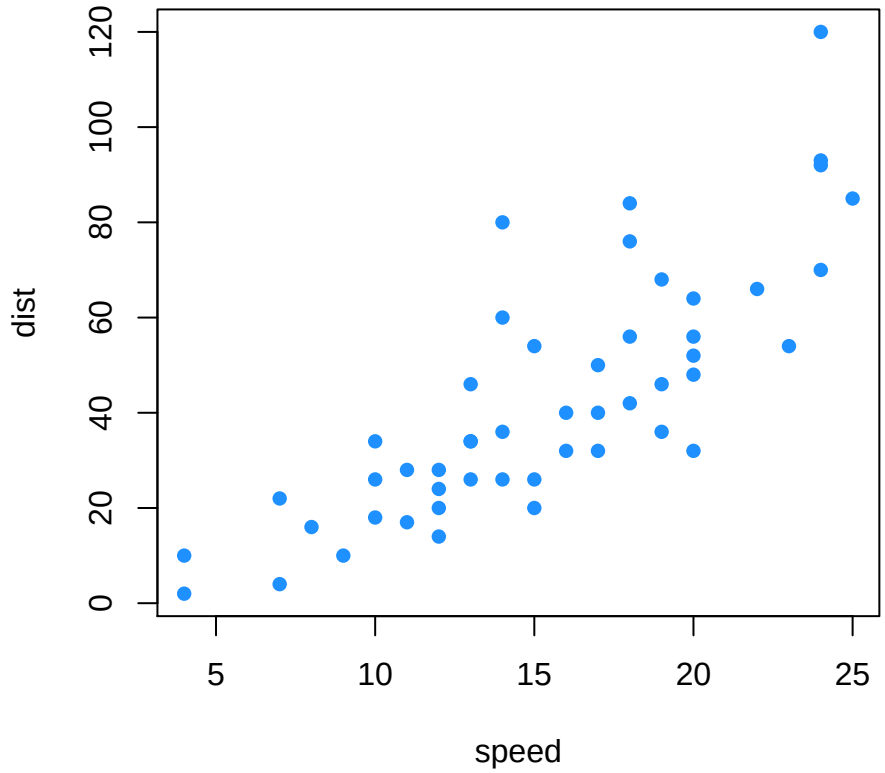
Embedding R code

While you can add notes and format them (as above), one of the cooler features of R markdown is your ability to embed R code.

```
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot. You can also use:

- `eval = FALSE`
  - code will not be evaluated
- `warning = FALSE`
  - warning messages will not be printed (can also set `errors = FALSE` and `messages = FALSE`)
- `results = 'hide'`
  - shows the code and not the results - the opposite of `echo`.
- `include = FALSE`
  - evaluates the code but suppresses **both** the code and its output.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

On an exciting (albeit random) note, you can also embed code from other programming languages in your R Markdown document. For example:

```
food = 'potatoes'
print 'My favorite food is %s!' % food
```

```
## My favorite food is potatoes!
```

For more details on using R Markdown see http://rmarkdown.rstudio.com.

Also checkout R Markdown cheatsheet and reference guides.

2. Version control

Git is a useful version control system that lets you track changes in your code, both for yourself and for others. It can be combined with the website interface GitHub for increased management/sharing.

While git can be used via the command line/terminal, you can also use git directly from within RStudio. This is very fun and reduces the probability of developing carpal tunnel by eliminating window switching!

Install Git

Download git and follow the instructions here to configure git, set up a GitHub account, and generate a SSH key (if needed).

Create a local Git respository

To use GitHub, you first initialize a local repository (repo) that is a directory on your computer that records changes to your code. Later, you can "push" those changes to the online directory for sharing.

In RStudio, go to:

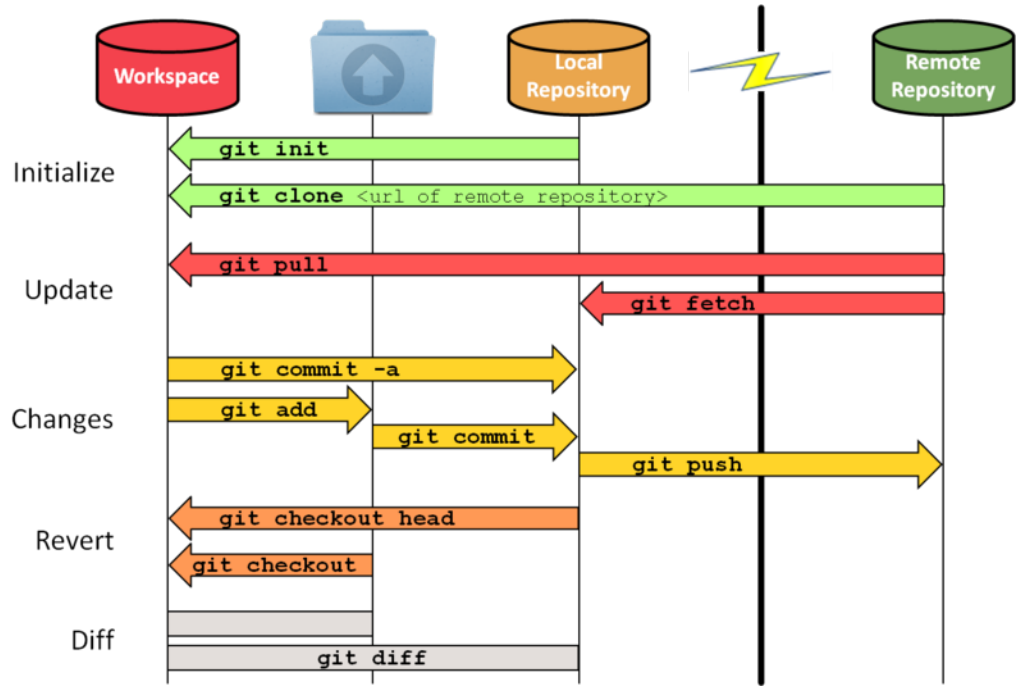
- Tools
  - Project options
    - Git/SVN panel
      - Select "git" under version control system

You should now see a git pane in the top right corner of Rstudio. Here, you can view files that have been modified from the last "commit", untracked files, or deleted files.

Navigating the git pane

Click **Diff** to view changes between your current version and the last "pushed" version.

Click **Commit** when you're ready to "stage" your changes. Here, you can select which files to add to the commit and include a message to be linked to that commit. Click **Push** when you are ready to send off all of your commits.



Selecting the **History** option lets you scroll through previous commits and identify where a mistake occurred. If it was in the *last* commit, you can use the **Revert** option. However, if it is further back, you can copy a past version into the present using its unique id (SHA).

```
git checkout <SHA> <filename>
```

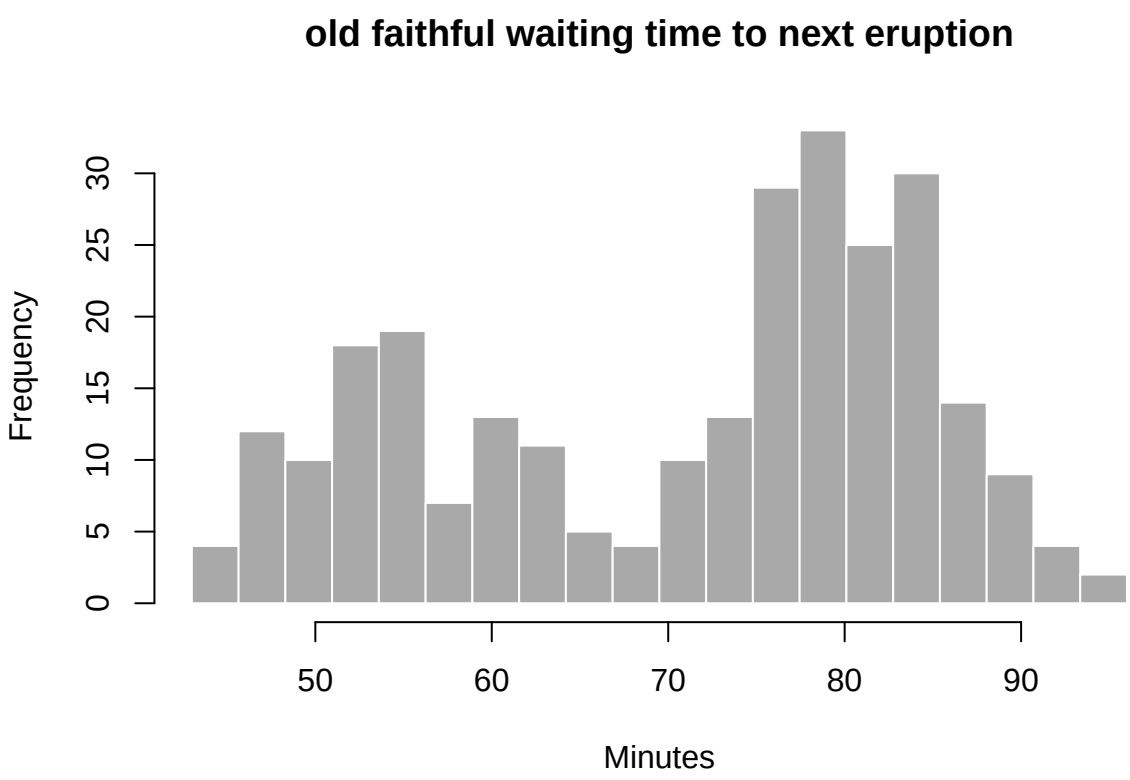
Check out more tutorials on git and git in RStudio.

3. Shiny

```
install.packages("shiny", repos="http://cran.rstudio.com/")
library(shiny)
setwd("C:/Users/kaylap/Dropbox/Kayla/Shiny")
```

Shiny has a few built-in examples to show you its capabilities. First, we'll look at a histogram that is interactive by allowing you to change the number of bins. Let's review the code to generate a histogram.

```
x <- faithful$waiting
bins <- seq(min(x), max(x), length.out=20+1)
hist(x, breaks=bins, col="darkgray", border="white",
     main="old faithful waiting time to next eruption", xlab="Minutes")
```



We can change the bins manually, by increasing or decreasing the number of breaks but Shiny allows the user to change the number of bins in real-time. This uses the input from the user (number of bins), and the *renderPlot()* function to allow the plot to update automatically.

```
runExample("01_hello")
```

Setting up Shiny R scripts

Shiny apps have 2 main components: 1. User-interface script 2. Server script

The User-interface script (which must be saved as **ui.R**) controls the layout and appearance of the app/webpage. It will include Shiny-specific commands such as *sidebarLayout()*, *mainPanel()*, etc.

Let's set up a basic user-interface script:

```
library(shiny)
shinyUI(fluidPage(
  #Main title
  titlePanel("Viral plaque assay replicates"),
  #Sidebar
  sidebarLayout(
    sidebarPanel(
      selectInput("replicate", "Plaque assay replicate:", choices=c("1", "2", "3")),
      hr()
    ),
    #Spot for the plot
```

```
    mainPanel(
      plotOutput("virusPlot")
    )
  )))
```

Next, we'll set up the **server.R** script which has the code needed to actually build the app.

```
library(shiny)
data <- read.csv("plaque_assay.csv")
shinyServer(function(input,output)
{
  #renderPlot indicates that the function is "reactive" - it should automatically
  #  re-execute when the input changes
  output$virusPlot <- renderPlot({

    #render the plot
    replicate = as.numeric(input$replicate)+1
    barplot(data[,replicate],
            main = paste("Replicate ", input$replicate),
            ylab="Virus titer (PFU/mL)",
            xlab="DPP4", names.arg=data$DPP4,
            col=replicate)
  })
})
```

You can run your app using the command *runApp("directory.name")*, where directory.name indicates that name of the folder that contains both the ui.R and the server.R file.

```
runApp("plaque_assay")
```

Translating user input into R output

One thing to note is that the values that are being inputted by the user are going to enter the R code as strings. Thus in the previous example, even though the user is choosing 1, 2, or 3, it's being read in R as "1", "2", "3". This was fixed above by using *as.numeric(input$replicate)*. However, if you are matching the value of a cell to a user-inputted argument, it does not have to be transformed. For example, in the next plot, we will use the user input to subset the data using:

```
virus_of_choice <- subset(dat, dat$virus==input$virus)
```

The plot overall will highlight specific points on a plot given the user's virus of choice. Additionally, users can input which value they would like to be printed on the screen:

```
runApp("virus_identify2")
```

The above examples are gratuitous plots, but here's a useful one that could actually be useful:

```
runApp("chikv_swelling")
```

Email me (kaylap@live.unc.edu) if interested in the code for the above Shiny apps!

Sharing your Shiny app

Once you have completed a graph, you can share it with your colleagues in several ways. The easiest way is to send them the server.R and ui.R files. However, you can also host the code online so that it can be run from anyone using R without the files. For example, github can host the code.

Putting my virus_identify code on github, anyone can run it by typing:

```
runGitHub("shiny", "kmpeck")
```

ShinyApps.io is an online way to host Shiny plots. However, this take a bit of work to be able to use. If interested, see the bottom of this document.

There are other cool things you can do with Shiny.
- You can have users input a file and your app can generate the plot/output
- You can produce tables, texts, images (not just plots)
- You can get fancy with the layout/formatting

Check out the examples below to see some of these options:

```
runExample("02_text")
runExample("06_tabsets")
runExample("09_upload")
```

Browse through the Shiny gallery for more complex/fancy examples.
movies
line chart
google charts

Using ShinyApps.io to host your plots:

Go to: http://cran.r-project.org/bin/windows/Rtools/
Download RTools version 3.1
Run find_rtools() from R
Update devtools package using install_packages("devtools")
Restart R session
Run devtools::install_github('rstudio/shinyapps')
Run library(shinyapps)
Go to shinyapps.io and create an account
Follow the instructions to authorize your computer through a "Token"
In R, run deployApp("directory.name") or use runApp("directory.name) and press the Deploy button at the top of the window. This links in with your shinyapps.io account and allows you to have a url associated with your code. *Note: the latest R version supported for this is 3.1.1*