get more out of R: from good stats to better science


p.mckenzie@columbia.edu


github.com/pmckenz1


@patrick_mcken_z

https://www.github.com/pmckenz1/weedsprez2018

this is only an overview – if you want to follow through with anything specific, **google** it

(or ask me later)

introduction

Kind of scattered, but based on the premise that:

R stats are everywhere

Best practices are not

You've probably heard of these tools but might not have used them

Goal of today:

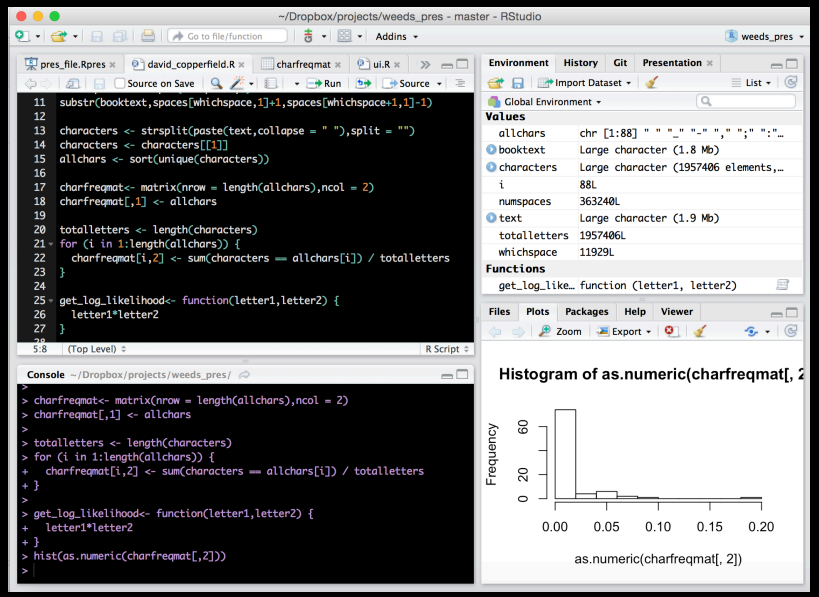
Quick survey of (imo) underused tools in RStudio

What today is not:

General coding lesson

Essential or in-depth

```
Last login: Mon Feb 12 23:24:44 on ttys000
Patricks-MacBook-Pro:~ pmckenz1$ R

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

>
```



what is R?

R is a programming language for statistics and graphing.

We interact with R through an Interactive Development Environment (IDE) called **RStudio**, which has many features to make coding easier.

RStudio®

Scripting and console

Run code line-by-line

Built-in plotting and help features

Tracking of environmental variables

And more: R projects, R Markdown, R Shiny, and easy R packages

to success

- Organization
- Readability
- Reproducibility
- Accessibility

Can you find your code from 2 years ago?

Is your old code still useful to you?

Is your code easy to understand?

Can others find and use your code?

Can your mom use your code?

Everything happens in the context of a “self-contained project”

organization

Use a consistent file structure

Keep a directory for each project

Keep a **README.txt** file
with every project

Tool #1: R projects

An R project is just a way to save your work.

Rather than opening R files one-by-one from different places, open them within an R project.

Your previously open files will be restored when you go back to work.

Starting an R project is easy: **file -> new project**

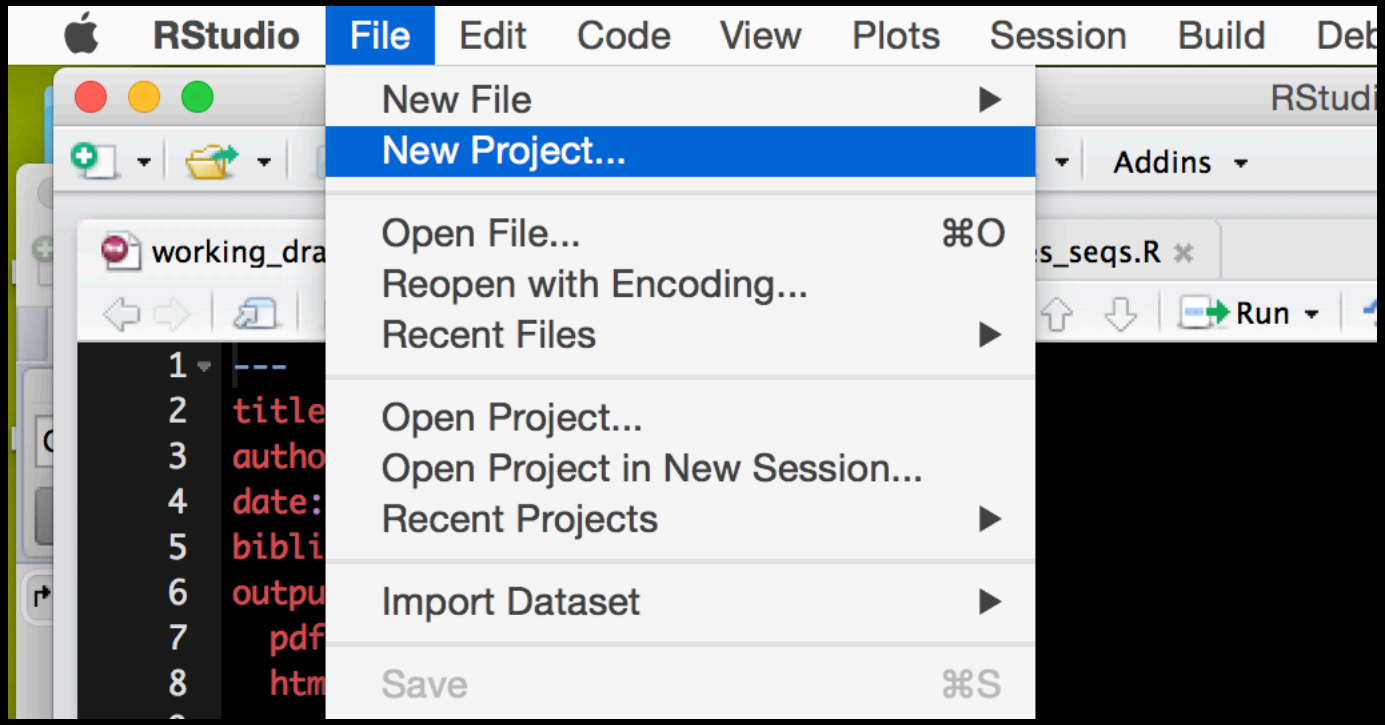
Incredibly simple.

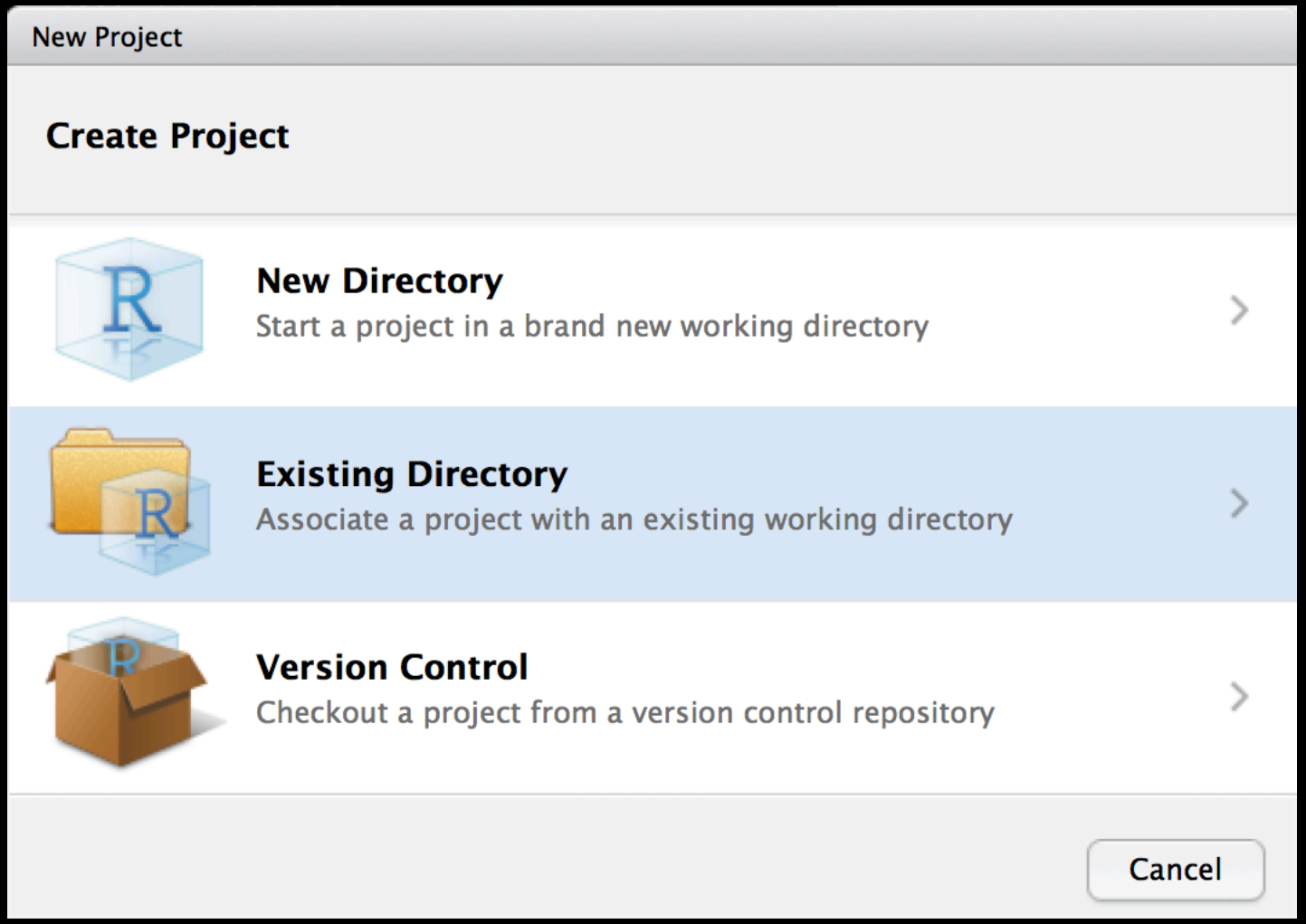
1) Default working domain
2) Default files
3) Self-contained workspace

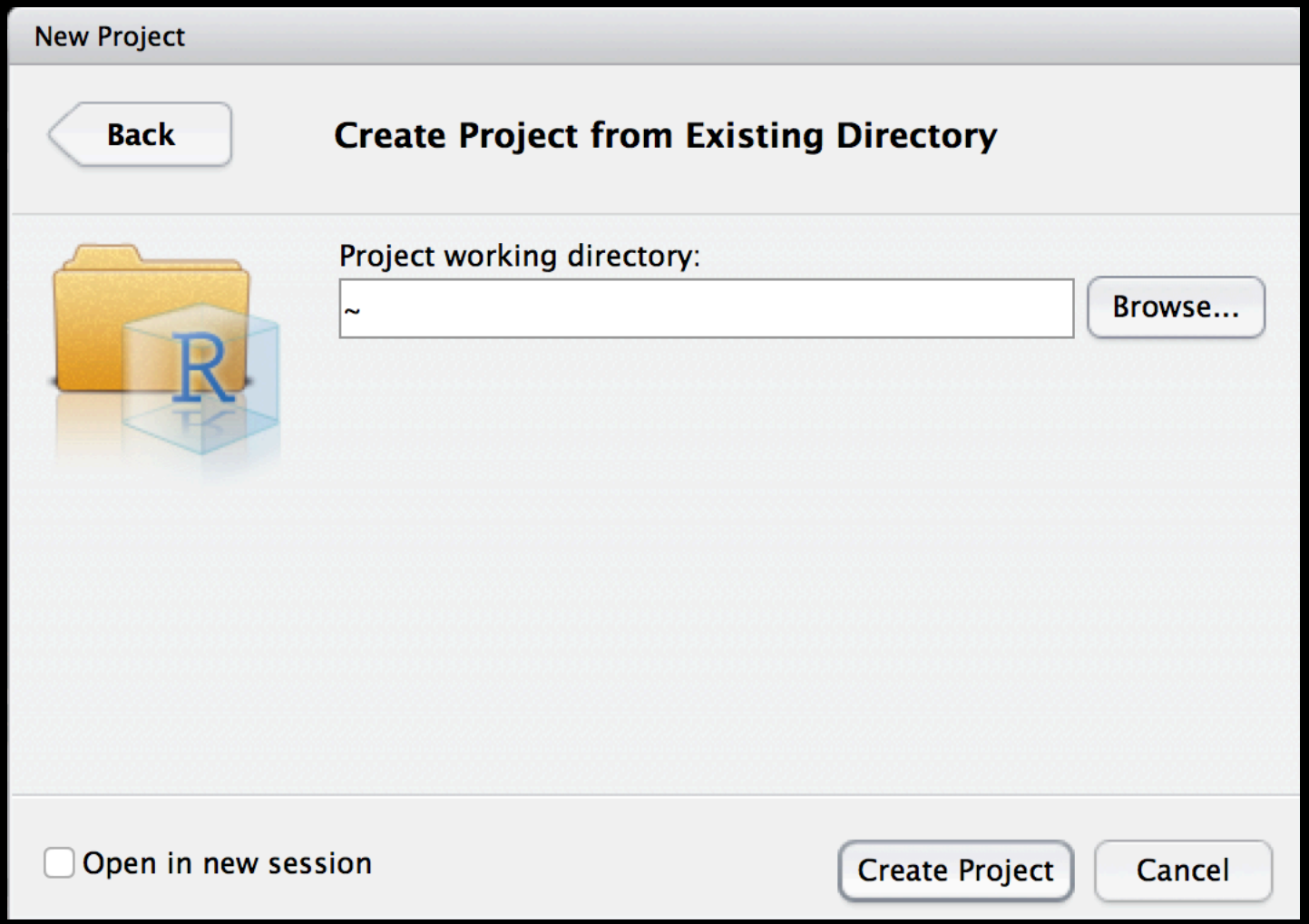
Implications?

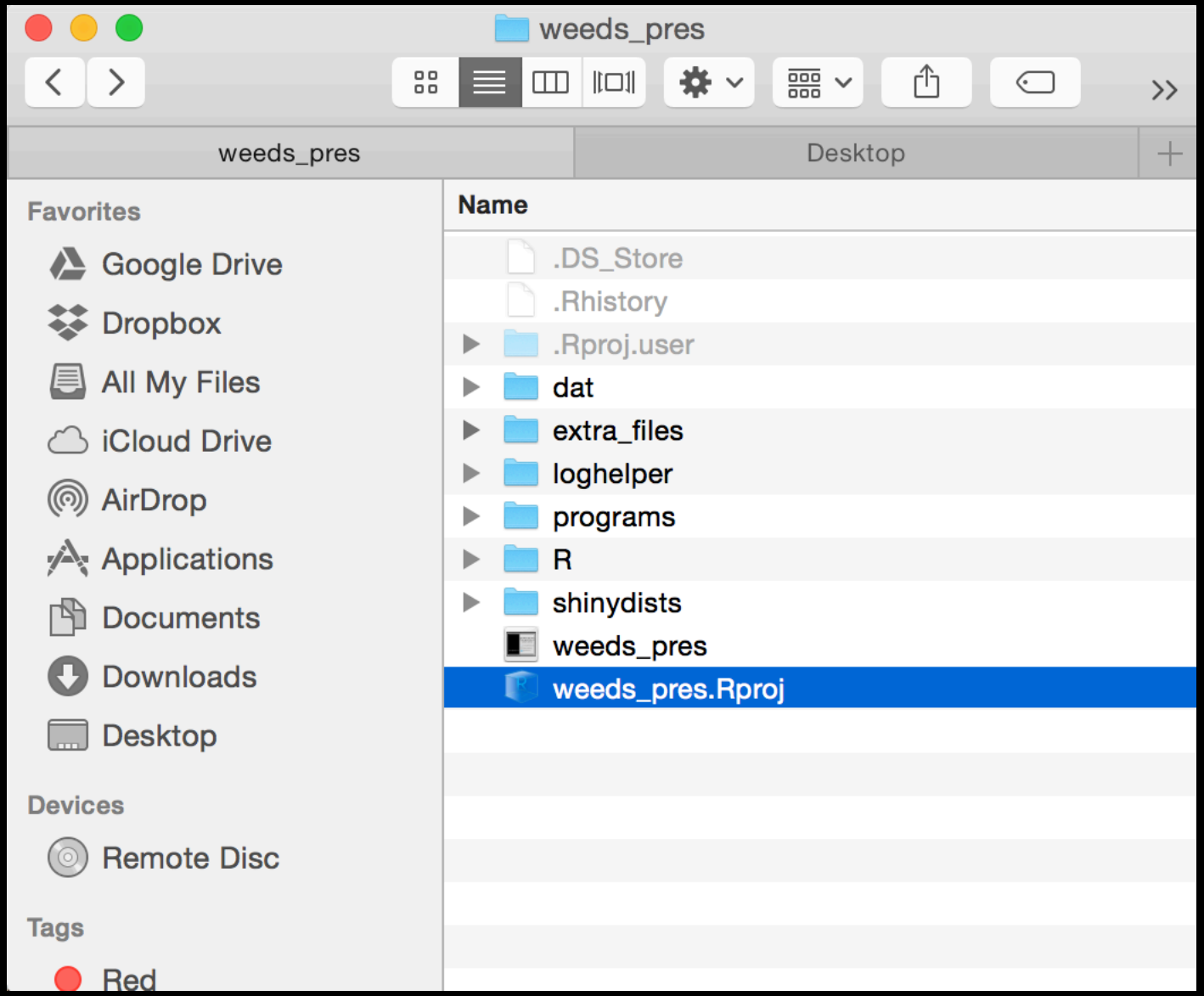
Makes it easy to bounce between multiple projects.

Encourages you to write code using local working directory.


RStudio
File
Edit
Code
View
Plots
Session
Build
Deb
New File
New Project...
Open File... ⌘O
Reopen with Encoding...
Recent Files
Open Project...
Open Project in New Session...
Recent Projects
Import Dataset
Save ⌘S
Addins
Run
working_dra
s_seqs.R
1 ---
2 title
3 autho
4 date:
5 bibli
6 outpu
7 pdf
8 htm


New Project
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel


New Project
Back
Create Project from Existing Directory
Project working directory:
~
Browse...
Open in new session
Create Project
Cancel


weeds_pres
weeds_pres
Desktop
Favorites
Google Drive
Dropbox
All My Files
iCloud Drive
AirDrop
Applications
Documents
Downloads
Desktop
Devices
Remote Disc
Tags
Red
Name
.DS_Store
.Rhistory
.Rproj.user
dat
extra_files
loghelper
programs
R
shinydists
weeds_pres
weeds_pres.Rproj

Examples on my own computer

getwd()
loading files
opening two projects at once

Big Idea: R Projects help you maintain discrete, self-contained projects

Ten papers (or ten computers) down the road, this will be nice.

Are you using a “self-contained project” approach but want to use your same R tools for a bunch of different projects?

Do you want to make your code easy for someone else to use?

Tool #2: R Packages

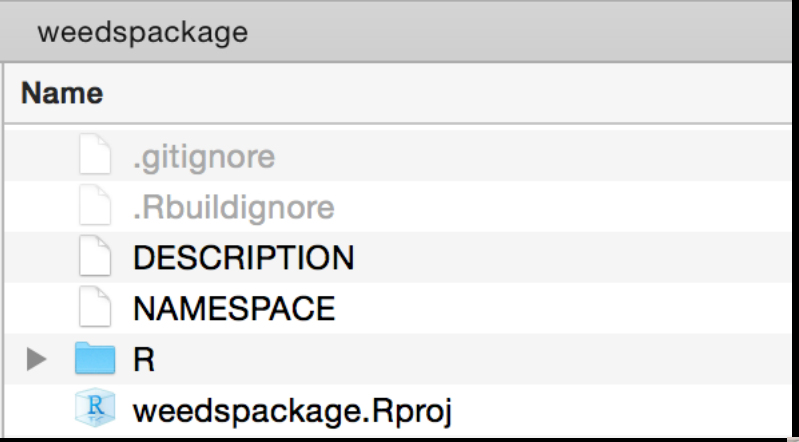
(Organization and reproducibility)

Writing self-contained, well-documented code that is easy to come back to and that others can use.

Installable globally, importable globally, has metadata.

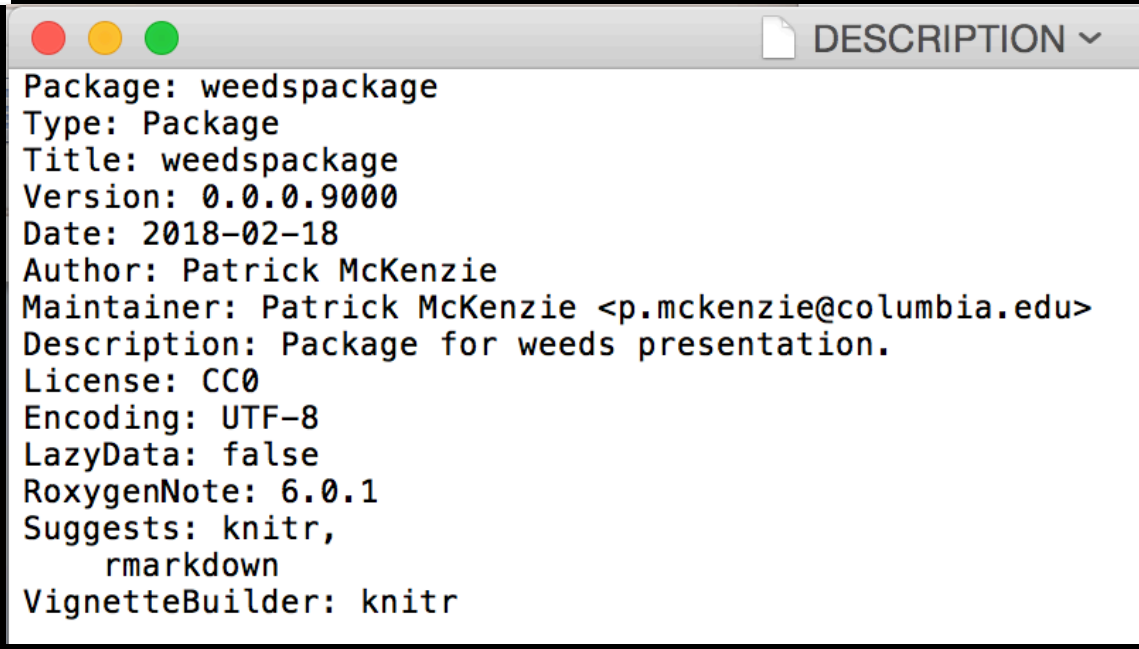
`devtools` and `roxygen2` packages automate the process of making our own R packages

```
1  ## These are the packages we need
2  ## install them if not already done
3  install.packages("devtools")
4  install.packages("roxygen2")
5
6  ## Load packages
7  library(devtools)
8  library(roxygen2)
9
10 ## Makes a directory for our new package
11 devtools::create("weedspackage")
```



This is our new directory, automatically set up for us. Cool.

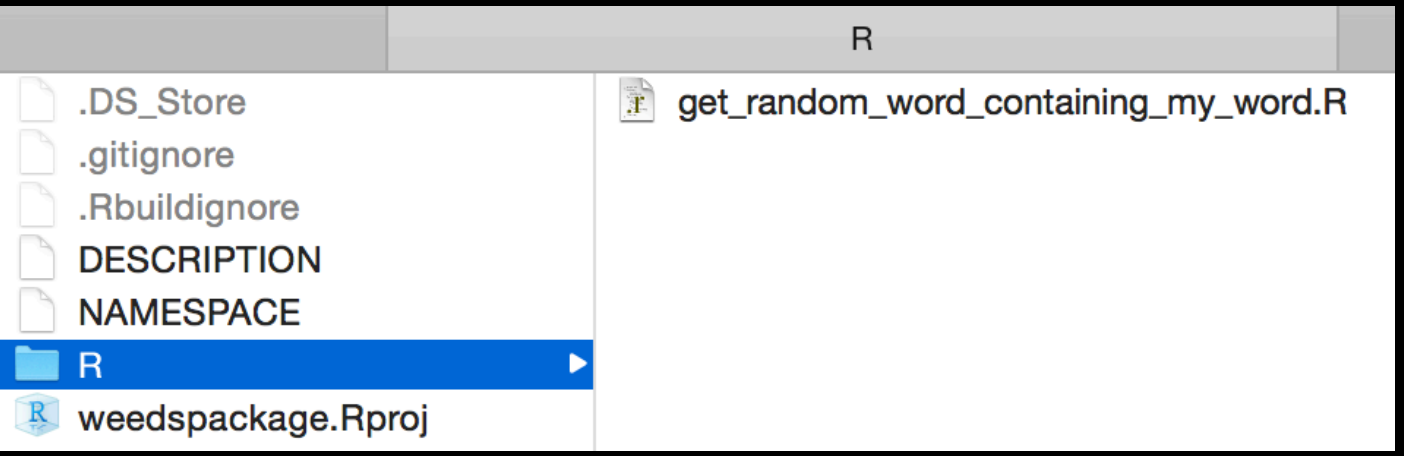
1) Manually edit the **DESCRIPTION** file in your new directory to include stuff like your package name, your name, the date, and the license



2) Write some functions – this is probably already done if you're trying to make a package. **Document them at the top using the format below.**

```r
##' Random word containing provided word
##'
##' This function takes a word as input and outputs a random word that contains the given word.
##' @param myword This is a string.
##' @export
##' @examples
##' get_random_word_containing_my_word("evolution")

get_random_word_containing_my_word <- function(myword) {
  allwords <- readLines("https://raw.githubusercontent.com/dwyl/english-words/master/words.txt")
  indices <- grep(pattern = myword,x = allwords)
  if (length(indices) == 0) {
    return("Sadly, there are no words that contain your word.")
  } else if (length(indices) == 1) {
    return(allwords[indices])
  } else {return(allwords[sample(indices,1)])}
}
```



3) Save your new functions as separate R files in the “R” folder in your new directory.

4) Use the **`document()`** function to automatically create a “man” folder for your documentation

```
> devtools::document()
Updating weedspackage documentation
Loading weedspackage
Writing NAMESPACE
Writing get_random_word_containing_my_word.Rd
Warning message:
In readLines(file) :
  incomplete final line found on '/Users/pmckenz1/Google
Drive/projects/weeds_pres/weedspackage/DESCRIPTION'
>
```

Either:

```
devtools::install("weedspackage")
```

Or (if it's on GitHub):

```
devtools::install_github("pmckenz1/weedspackage")
```

Then use it like a normal package.

```
library(weedsprez)
get_random_word_containing_my_word("evolution")
```

Our package is done!

We can install it locally, or we can push it to a GitHub repo so that anyone can install it

Next steps:

Write a vignette to explain how to use the package

Write unit tests to make sure updates to your package don't cause problems

Big Idea: Packages are easy to make.
Then other people can use your code.
This is good for replicable science.

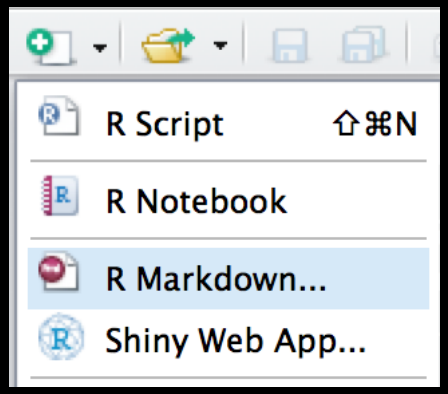
Tool #3: R Markdown

Essential for **readability** and **reproducibility**

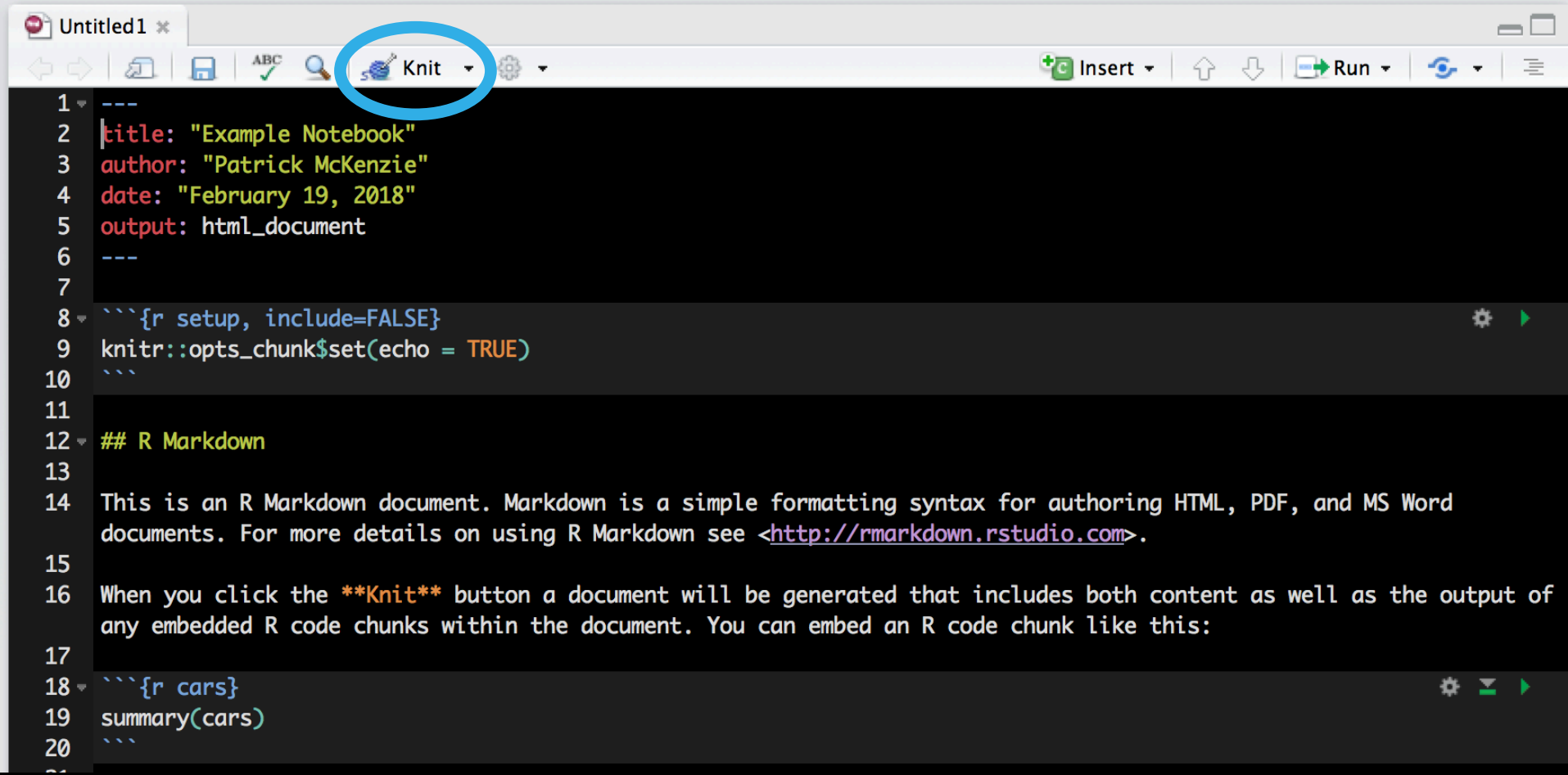
I use these for LITERALLY everything. Research meetings, papers, homework, note-taking, whatever.

Write and format text, code, equations, and plots in the same document.

Output as .pdf or .html



Easy to start a new markdown document from your R project.



collecting_inputs_example.Rmd

Knit | Insert | Run

```
---
title: "Collecting inputs for the biogeography model"
author: "Patrick McKenzie"
date: "June 6, 2017"
output:
  pdf_document: default
  html_document: default
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```{r include = FALSE}
library(raster)
```

This document walks through the process of collecting the two types of inputs for the biogeography model: a single environmental matrix, and a "current distribution" matrix for each species.

Two key functions used are `sdm_function()` and `conv_sdm_rasters_threshold()`. Both of these are included in the "R" folder. The functions use the MaxEnt species distribution modeling program to produce a distribution map for each species.

Choosing the species and the geographic area

For maxent species distribution modeling, we chose 14 species of *Pedicularis*, a large genus of flowering plants concentrated in mid-southern China, from across its phylogeny. The geographic region from which data was collected was bounded by longitudes of 80 and 112 degrees and latitudes of 18 and 40 degrees.

The following code defines the species we're using and creates directories for maxent files.
```{r eval=FALSE}
species <- c("salviiflora","muscoides","trichoglossa","ingens","axillaris","armata","rhinanthoides","rex","integrifolia","verticillata","cheilanthifolia","lyrata","lutescens","kansuensis")
```
```

1:1 | Collecting inputs for the biogeography model | R Markdown

Collecting inputs for the biogeography model

Patrick McKenzie

June 6, 2017

This document walks through the process of collecting the two types of inputs for the biogeography model: a single environmental matrix, and a "current distribution" matrix for each species.

Two key functions used are `sdm_function()` and `conv_sdm_rasters_threshold()`. Both of these are included in the "R" folder. The functions use the MaxEnt species distribution modeling program to produce a distribution map for each species.

Choosing the species and the geographic area

For maxent species distribution modeling, we chose 14 species of *Pedicularis*, a large genus of flowering plants concentrated in mid-southern China, from across its phylogeny. The geographic region from which data was collected was bounded by longitudes of 80 and 112 degrees and latitudes of 18 and 40 degrees.

The following code defines the species we're using and creates directories for maxent files.

```
species <- c("salviiflora","muscoides","trichoglossa","ingens","axillaris","armata","rhinanthoides","
for (i in species) {
  dir.create(paste0("example_species_dist/species_dist_maps/",i))
}
```

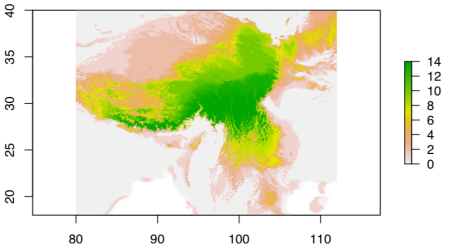
Then we run MaxEnt for each species using `sdm_function()` to produce the distribution maps.

```
dir <- paste0("example_species_dist/species_dist_maps/",species)
speciesnames <- paste0("Pedicularis ",species)
for (i in 1:length(species)) {
  sdm_function(
    directory.path = dir[i],
    coords = c(80,112,18,40),
    species.name = speciesnames[i],
    climate.map.path = "example_species_dist/climate_map/wc2-5"
  )
}
```

Creating the environmental matrix

To make a single environmental matrix for the phylogeography method, we read in each species' maxent output raster individually and apply `conv_sdm_rasters_threshold()`, which uses a threshold value to assign either 1 or 0 to each cell in the maxent output rasters. We then sum all raster layers and use `as.matrix()` to convert the summed distribution rasters to a matrix. This was then saved as a .csv file.

```
rasterfilenames <- paste0(gsub(" ","_",speciesnames),".asc")
specieslists <- list()
for (i in 1:length(species)) {
  raster_path <- paste0("example_species_dist/species_dist_maps/",species[i],"/outputs/",rasterfilena
  results_file <- paste0("example_species_dist/species_dist_maps/",species[i],"/outputs/maxentResults
  specieslists[[i]] <- conv_sdm_rasters_threshold(raster_path = raster_path,
                          results_file = results_file,
                          "balanced")
}
environmental_matrix <- sum(stack(specieslists))
write.csv(as.matrix(environmental_matrix),"example_species_dist/phyloproject_inputs/environmental_matri
```

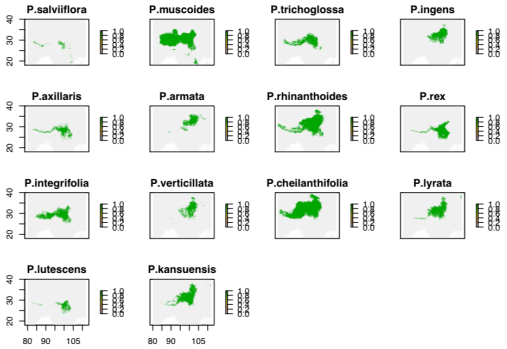


Current species ranges

We want to save matrices for the individual species' ranges as inputs for the biogeography model. Again, we use `conv_sdm_rasters_threshold()`. However, we now use the most conservative threshold, "10pct," and we write each one as a .csv file.

```
rasterfilenames <- paste0(gsub(" ","_",speciesnames),".asc")
for (i in 1:length(species)) {
  raster_path <- paste0("example_species_dist/species_dist_maps/",species[i],"/outputs/",rasterfilename
  results_file <- paste0("example_species_dist/species_dist_maps/",species[i],"/outputs/maxentResults.c
  specieslayer <- conv_sdm_rasters_threshold(raster_path = raster_path,
                                             results_file = results_file,
                                             "10pct")
  write.csv(as.matrix(specieslayer),paste0("example_species_dist/phyloproject_inputs/",species[i],".csv
}
```

Now we can see the current ranges for each species.



R Markdown Cheatsheet

https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf

Big Idea: R Markdown makes presenting your project quick and easy.

Your advisor will thank you.

And, years later, it will help you remember why you wrote your code the way you did.

Tool #4: R Shiny

Good as a learning or teaching tool, and good for sharing your work.

Uses “reactive programming,” where code is written incorporate values from a range of possible user-provided inputs and **automatically update the outputs**.

Easy to host online.

Shiny apps exist inside a dedicated directory that must include **at least these two files:**

ui.R

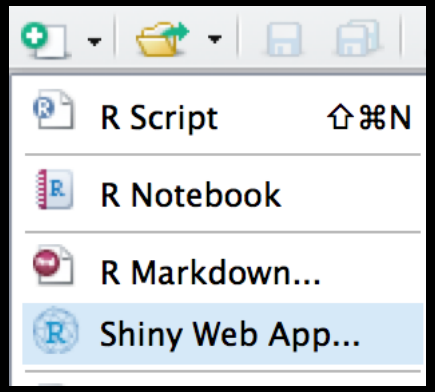
server.R

Or just a single file, **app.R**, that includes **ui** and **server** objects.

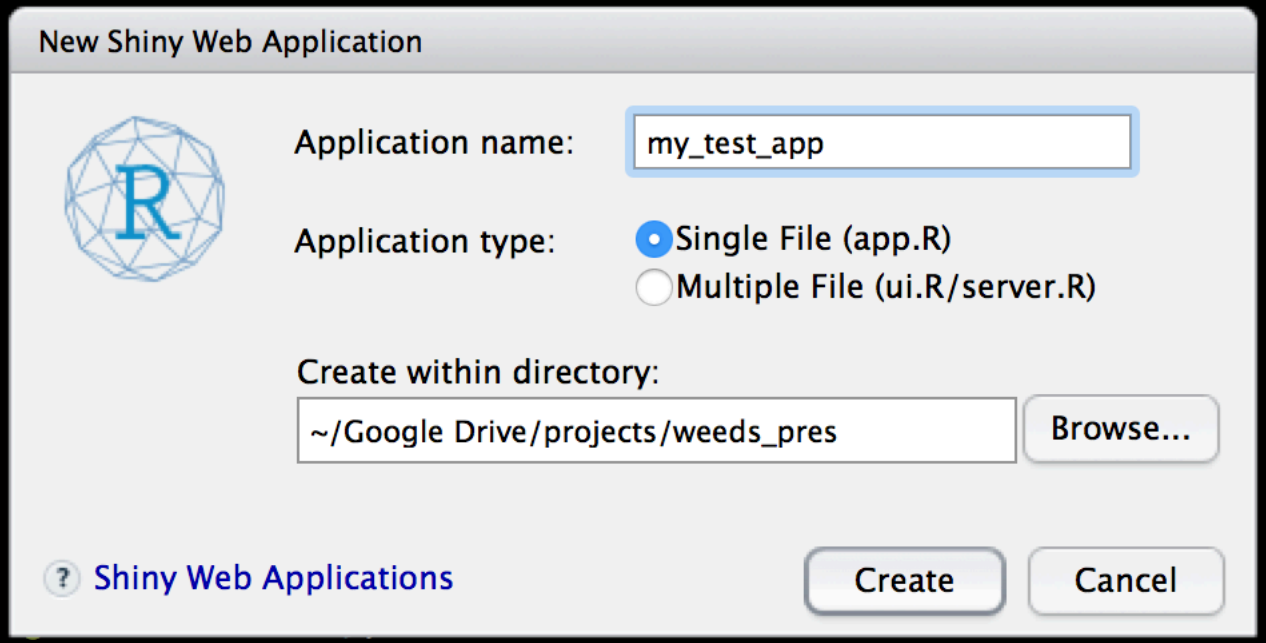
ui defines the layout of the page that the user will see, including inputs (sliders, etc.) and outputs (plots, etc.)

server is a function that accepts inputs from **ui** and returns whatever outputs the user is supposed to see.

Let’s make our own simple Shiny app.



We'll use a single file application type for simplicity. For more complicated apps, it's cleaner to use the two separate files.



This creates a folder with our application name inside the designated directory.

app.R

```
#
# This is a Shiny web application. You can run the application by clicking
# the 'Run App' button above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

   # Application title
   titlePanel("Old Faithful Geyser Data"),

   # Sidebar with a slider input for number of bins
   sidebarLayout(
      sidebarPanel(
         sliderInput("bins",
                     "Number of bins:",
                     min = 1,
                     max = 50,
                     value = 30)
      ),
```

1:1 (Top Level)

```r
ui <- fluidPage(

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for 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")
    )
  )
)
```

ui defines what the app user will see.

This app will have:

- title panel labeled “Old Faithful Geyser Data”
- sidebar with a single slider
- panel to plot outputs

```r
# Define server logic required to draw a histogram
server <- function(input, output) {

   output$distPlot <- renderPlot({
      # generate bins based on input$bins from ui.R
      x    <- faithful[, 2]
      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')
   })
}
```

server is a function that takes user inputs and returns what the user should see.

The input here is the value from the “bins” slider.

The output plot is saved to an output list object with the name **distPlot**.

distPlot is what is plotted on the ui (as we defined in the previous slide).

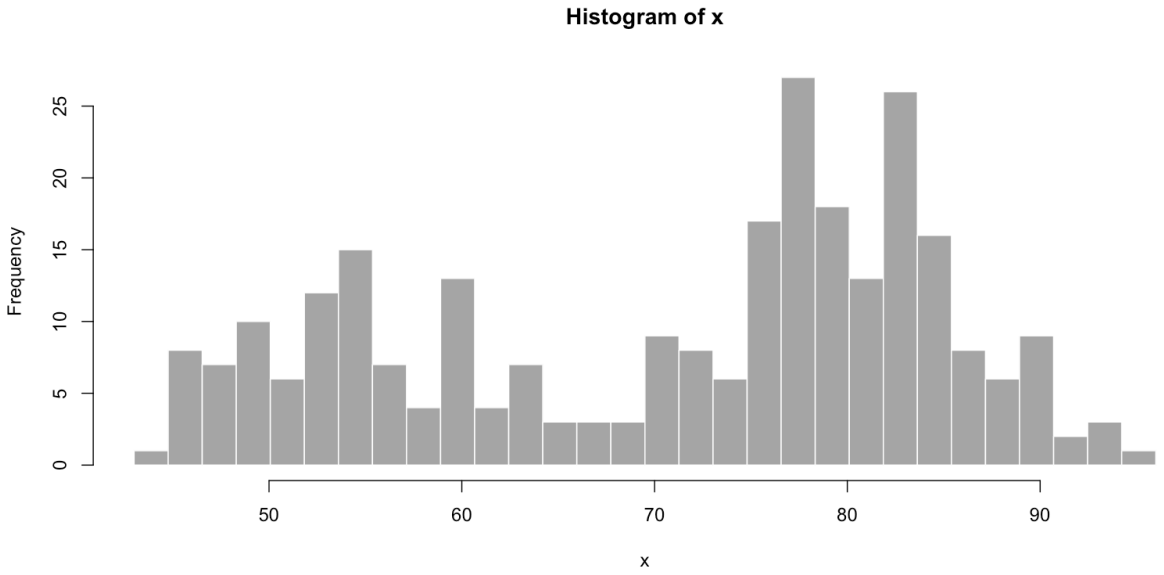
```
#
# This is a Shiny web application. You can run the application by clicking
# the 'Run App' button above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

   # Application title
   titlePanel("Old Faithful Geyser Data"),

   # Sidebar with a slider input for 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")
      )
   )
)

# Define server logic required to draw a histogram
server <- function(input, output) {

   output$distPlot <- renderPlot({
      # generate bins based on input$bins from ui.R
      x    <- faithful[, 2]
      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')
   })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Old Faithful Geyser Data

Number of bins:

1 30 50

1 6 11 16 21 26 31 36 41 46 50



Let’s look at a few examples (not in this presentation file)

https://pmckenz1.shinyapps.io/loghelper/

https://pmckenz1.shinyapps.io/Deterministic_Mutualism_Model/

R Shiny Cheatsheet

http://shiny.rstudio.com/images/shiny-cheatsheet.pdf

R Shiny Gallery

https://shiny.rstudio.com/gallery/

Big Idea: **R Shiny is good for making your work accessible to others.**

Particularly good as a learning tool.

R Projects
R Packages
R Markdown
R Shiny

Conclusion

Disclaimer first:

nothing here is always best for the job

But using these can reinforce **best practices** of coding for science

Organization

Readability

Reproducibility

Accessibility

There are Python equivalents of everything here