Intermediate R Workshop Notes

Hena R. Ramay

2020-05-21

Contents

Chapter 1

Introduction

This notebook is created so that the particpants of the workshop can follow along with the instructors. We will constantly update this notebook during the workshop to include answers to the questions asked during the workshop. So Please ask a lot of questions!!

1.1 Workshop Schedule

We will try to cover the following material in the course:

Day 1

Time	Topic
9:00	Introductions and workshop overview
9:30	Conditionals and Loops
10:45	Functions
13:00	Apply family
14:00	Advanced plotting ggplot2

Day 2

Time	Topic
9:00	R markdown basics
10:45	Knitr chunks
13:00	Rmarkdown themes, adding html and custom css styling

Time	Topic
14:00	A short project to bring it all together!

1.2 Important links

Overview and setup

Lets get started!

1.3 Setting up a project

1.4 Learning Objectives

- Create directories and files
- Understand the difference between absolute and relative paths

Before we start using R, we will first review the basics of the Unix shell by setting up our project. Open the terminal in OS X or Linux, or Git Bash in Windows.

Our project will explore the citations and alternative metrics (altmetrics) for articles published in the PLOS family of journals between 2003 and 2010. The data set was compiled by Priem et al. 2012 (publication, code).

First create a new directory to store the project files called `altmetrics` and then change to that directory.

```
mkdir altmetrics
cd altmetrics
```

Repeat this process to create a subdirectory to store the data files.

```
mkdir data
cd data
```

1.5 Finding your location

Recall that you can always determine where you are by running the command `pwd`, which stands for "print working directory". Also, if you run `cd` with no arguments, it takes you to your home directory.

Download the two data files using the links below. Save them to the data subdirectory.

- counts-raw
- counts-norm

1.6 Downloading files from the command line

When possible, it is best to download files from the command line because that makes the analysis more reproducible. Different systems have different tools installed, which is why we manually downloaded the files for this lesson. Some options include `wget`, `curl`, and `rsync`. As an example, here is how to perform the download using `wget`.

```
wget https://raw.githubusercontent.com/jdblischak/r-intermediate-altmetrics/gh-pages/data/counts-
wget https://raw.githubusercontent.com/jdblischak/r-intermediate-altmetrics/gh-pages/data/counts-
```

If `wget` isn't installed on your machine, try `curl`:

```
curl -O "https://raw.githubusercontent.com/jdblischak/r-intermediate-altmetrics/gh-pages/data/coun
curl -O "https://raw.githubusercontent.com/jdblischak/r-intermediate-altmetrics/gh-pages/data/cou
```

The first file, `counts-raw.txt.gz`, contains the raw counts for each of the articles across all the metrics. The second file, `counts-norm.txt.gz`, contains the counts for each of the articles across all the metrics after they have been normalized across disciplines and years.

Confirm that the files downloaded by listing the files in `data`.

```
ls
```

```
counts-norm.txt.gz counts-raw.txt.gz
```

Now move back up a directory to `altmetrics`. One option would be to specify the absolute path to this directory, e.g. `~/altmetrics/` if you created the directory in your home folder. However, an easier option is to use a relative path, which is dependent on the directory you are currently in. The shortcut to move to the directory above is two periods: `..`.

```
cd ..
```

Now you should be in the `altmetrics` directory. If you ran the above command again, you would be moved to the directory where you created `altmetrics`.

To create files, we'll use the simple text editor `nano`. As an argument, you provide the name of an existing file to edit or the name of a new file to create. If you call `nano` without specifying a filename, it will prompt you for a filename when saving. Create a file to practice.

```
nano example-file
```

The commands are listed at the bottom of the screen. The `^` stands for `Ctrl`, thus to save type `Ctrl-W` and to exit type `Ctrl-X`.

Remove the file with `rm`.

```
rm example-file
```

1.7 Organizing a larger project

Projects can grow quickly. For ideas on organizing your own projects, see A Quick Guide to Organizing Computational Biology Projects by William Noble.

1.8 Create a README file

It is a convention to have a file named `README` in a project directory to explain what it contains (both for others and your future self). Use `nano` to create a README file. Include the date and explain that this directory was created for a Software Carpentry workshop.

Chapter 2

Conditionals and Loops

2.1 Learning Objectives

- Write an if…else statement.
-

2.2 Write a for loop to modify a data frame.

2.3 Conditional statements

Decision making is an important part of programming. This can be achieved in R programming using conditional statements such as `if` and `if...else`.

2.4 if

The syntax of an if statement is:

```r
if (test_expression) {
  do_this
}
```

```
x <- 5
if (x > 0) {
  print("positive number")
}
```

```
## [1] "positive number"
```

2.5 if...else

The syntax of an if...else statement is:

```
if (test_expression) {
  do_this
} else {
    do_that
}
```

The else part is optional and is only evaluated if test_expression is FALSE. It is important that the `else` word be in the same line as the closing brace of the `if` statement.

```
x <- 1
if (x > 0){
  print("positive number")
} else {
    print("negative number")
}
```

```
## [1] "positive number"
```

2.6 Nested if...else statements

You can have more than two test expressions:

```
if (test_expression1) {
  statement1
} else if (test_expression2) {
    statement2
} else {
      statement4
}
```

```
x <- 0
if (x < 0) {
   print("negative number")
```

```r
} else if (x > 0) {
   print("positive number")
} else {
   print("zero")
}
```

```
## [1] "zero"
```

2.6.1 EXERCISE

Write a simple if…else statement to check if 5 is an odd number and if it is print "I am odd", otherwise print "I am even".

2.7 Loops

Conceptually, a loop is a way to repeat a sequence of instructions under certain conditions. They allow you to automate parts of your code that are in need of repetition.

The easiest and most frequently used loop in R is a for loop. Here is a demonstration of using loops.

```r
year <- c(2015,2016,2017,2018)
for(i in 1:length(year)) {
  print(year[i])
}
```

```
## [1] 2015
## [1] 2016
## [1] 2017
## [1] 2018
```

```r
for(i in 1:length(year)) {
  print(paste0("the year is ",year[i]))
}
```

```
## [1] "the year is 2015"
## [1] "the year is 2016"
## [1] "the year is 2017"
## [1] "the year is 2018"
```

2.7.1 EXERCISE

Challenging. Using the *E. coli* metadata from the last section, write a `for` loop containing an `if...else` statement to change the genome size into categorical variable with two levels: large and small.

Do this by translating the following sentence into R code: for every element in the genome_size variable, if the genome size is greater than the mean, change it to “large”, otherwise, change it to “small”.

Chapter 3

Functions

"Intelligence is the ability to avoid doing work, yet getting the work done."
Linus Torvalds

3.1 Learning Objectives

- Functions have two parts: arguments and body
- Functions have their own environment
- Functions help you repeat code chunks

In the last lesson we learnt to write loops to avoid writing the same code multiple time. Now lets learn how to make our code even more efficient and re-use the same code multiple times without copy-paste!

3.1.1 The parts of a function

We've already been using built-in R functions: `read.csv`, `mean`, `length`, etc. These functions allow us to run the same routine with different inputs.

Let's explore `read.csv` further. All functions in R have two parts: the input arguments and the body. We can see the arguments of a function with the `args`.

```
args(read.csv)
```

```
function (file, header = TRUE, sep = ",", quote = "\"", dec = ".",
    fill = TRUE, comment.char = "", ...)
NULL
```

So when we pass a character vector like `"data/counts-raw."`, this gets assigned to the argument `file`. All the other arguments have defaults set, so we do not need to assign them a value.

After the arguments have been assigned values, then the body of the function is executed. We can view the body of a function with `body`.

```
body(read.csv)
```

```
read.table(file = file, header = header, sep = sep, quote = quote,
    dec = dec, fill = fill, comment.char = comment.char, ...)
```

`read.csv` is very short. It just calls another function, `read.table`, using the input file and the default arguments as the arguments passed to `read.table`.

When we define our own functions, we use the syntax below. We list the arguments, separated by commas, within the parentheses. The body follows, contained within curly brackets `{}`.

```
function_name <- function(args) {
  body
}
```

3.1.2 The principle of encapsulation

An important feature of functions is the principle of encapsulation: the environment inside the function is distinct from the environment outside the function. In other words, variables defined inside a function are separate from variables defined outside the function.

Here's an small example to demonstrate this idea. The function `ex_fun` takes two input arguments, `x` and `y`. It calculates `z` and returns its value.

```
ex_fun <- function(x, y) {
  z <- x - y
  return(z)
}
```

When we run `ex_fun`, the only thing returned to the global environment is the value that was assigned to `z`. The variable `z` itself was only defined in the function environment, and does not exist in the global environment.

```
ex_fun(3, 10)
```

```
[1] -7
```

```
z
```

```
Error in eval(expr, envir, enclos): object 'z' not found
```

3.2 Environments are complicated

The situation presented above is a simplified version of environments which will serve you well if you treat functions as truly encapsulated. In reality, things are more complicated. For example, if inside a function you have a variable that has not been defined in the function, it will actually search the global environment for this variable. To learn the advanced details, see the chapter Environments in Advanced R by Hadley Wickham.

3.3 The return statement

R provides the shortcut of not needing to use `return` at the end of the function. Instead, the variable on the last line of the body of the function is returned. This is useful for writing very small functions, but in these lessons we will use `return` to be more explicit about what is happening.

3.3.1 Challenges

3.4 Write your own function

Write your own function to calculate the mean called `my_mean`. It should take one input argument, `x`, which is a numeric vector. Compare your results with the results from R's function `mean`. Do you receive the same answer?

Chapter 4

Apply Functions with purrr package

https://towardsdatascience.com/functional-programming-in-r-with-purrr-469e597d0229

It's easier to see the intent of your code, because your eyes are drawn to what's different, not what stays the same.

It's easier to respond to changes in requirements. As your needs change, you only need to make changes in one place, rather than remembering to change every place that you copied-and-pasted the code.

You're likely to have fewer bugs because each line of code is used in more places.

Chapter 5

Advanced Plotting

"The greatest value of a picture is when it forces us to notice what we never expected to see" -John Tukey

5.1 Setup

1. Install the `tidyverse` package.

```
library(tidyverse)
```

2. Download datasets.

We will be using a dataset containing citation and alternative metrics for articles published in the PLOS family of journals between 2003 and 2010. The data set was compiled by Priem et al 2012 (publication).

Download the data onto your computer from this dropbox link and move it into a directory on your computer that makes sense.

3. Read in data into R.

```
counts <- read.delim("data/counts-raw.txt")

research <- filter(counts, articleType == "Research Article")
```

5.2 Review of ggplot2 basics

`ggplot2` is a plotting package that makes it simple to create complex plots from data in a data frame. Graphics are built step by step by adding new elements.

Adding layers in this fashion allows for extensive flexibility and customization of plots.

A plot can be divided into different fundamental parts:

Plot = data + aesthetics + geom

Required building blocks:

- data
- aesthetics - describe how data are mapped to colour, size, shape, location
- geoms - geometric objects like points, lines, shapes

Optional building blocks:

- facets - describes how panel plots should be constructed
- stats - statistical transformations like binning, quantiles, smoothing
- coordinates - describes the system in which the locations of the geoms will be drawn
- scales - what scale an aesthetic map uses (ex. male = red, female = blue)

To build a ggplot, we will use the following basic template that can be used for different types of plots:

```
ggplot(data = <DATA>, mapping = aes(<MAPPINGS>)) +  <GEOM_FUNCTION>()
```

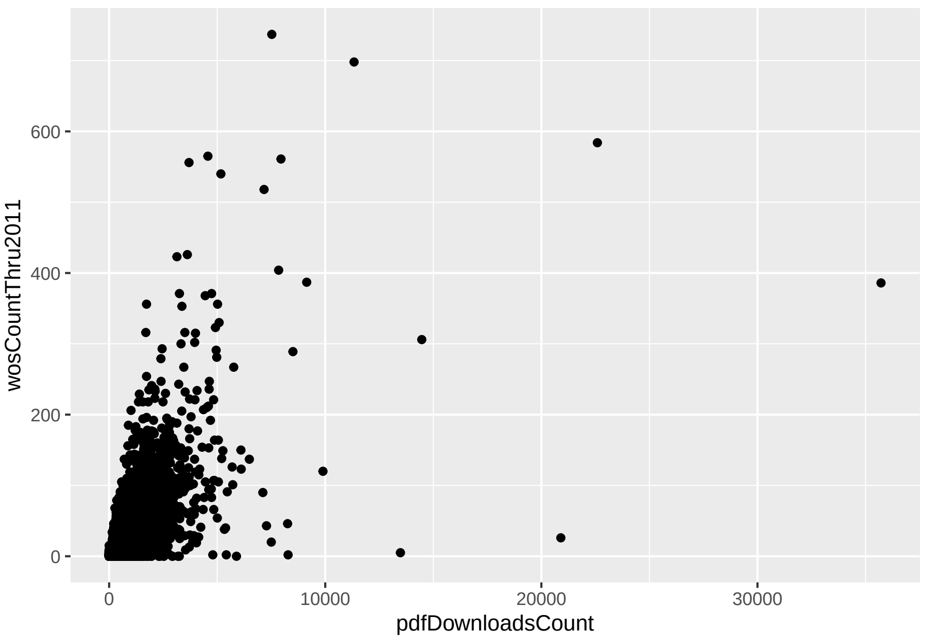
1. Specify which data set to use for the plot using the `data` argument.

2. Define a "mapping" (using the aesthetic (`aes`) function), by selecting the variables to be plotted and specifying how to present them in the graph, e.g. as x/y positions or characteristics such as size, shape, color, etc.

3. Add "geoms" – graphical representations of the data in the plot (points, lines, bars). `ggplot2` offers many different geoms; common ones include:

- `geom_point()` for scatter plots, dot plots, etc.
- `geom_boxplot()` for boxplots.
- `geom_histogram()` for histograms.
- `geom_barplot()` for barplots.
- `geom_line()` for trend lines, time series, etc.

```
p <- ggplot(research, aes( x = pdfDownloadsCount, y = wosCountThru2011)) + geom_point(
p
```



5.3 Scales

5.4 Faceting

5.5 Themes

In addition to `theme_bw()`, which changes the plot background to white, `ggplot2` comes with several other themes which can be useful to quickly change the look of your visualization. The complete list of themes is available at https://ggplot2.tidyverse.org/reference/ggtheme.html. `theme_minimal()` and `theme_light()` are popular, and `theme_void()` can be useful as a starting point to create a new hand-crafted theme.

The `ggthemes` package provides a wide variety of options (including an Excel 2003 theme). The `ggplot2` extensions website provides a list of packages that extend the capabilities of `ggplot2`, including additional themes.

5.6 Color palettes

You can create your own color palettes using the `colorRamp()` or `colorRampPalette()` function.

`colorRamp()` returns a function that takes values between 0 and 1, indicating the extremes of the color palette.

`colorRampPalette()` returns a function that takes integer arguments and returns a vector of colours.

```
cols <- colorRamp(c("red", "blue"))
cols(0)
```

```
##      [,1] [,2] [,3]
## [1,]  255    0    0
```

```
cols(0.5)
```

```
##       [,1] [,2]  [,3]
## [1,] 127.5    0 127.5
```

```
cols(1)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0  255
```

```
cols <- colorRampPalette(c("red", "blue"))
cols(2)
```

```
## [1] "#FF0000" "#0000FF"
```

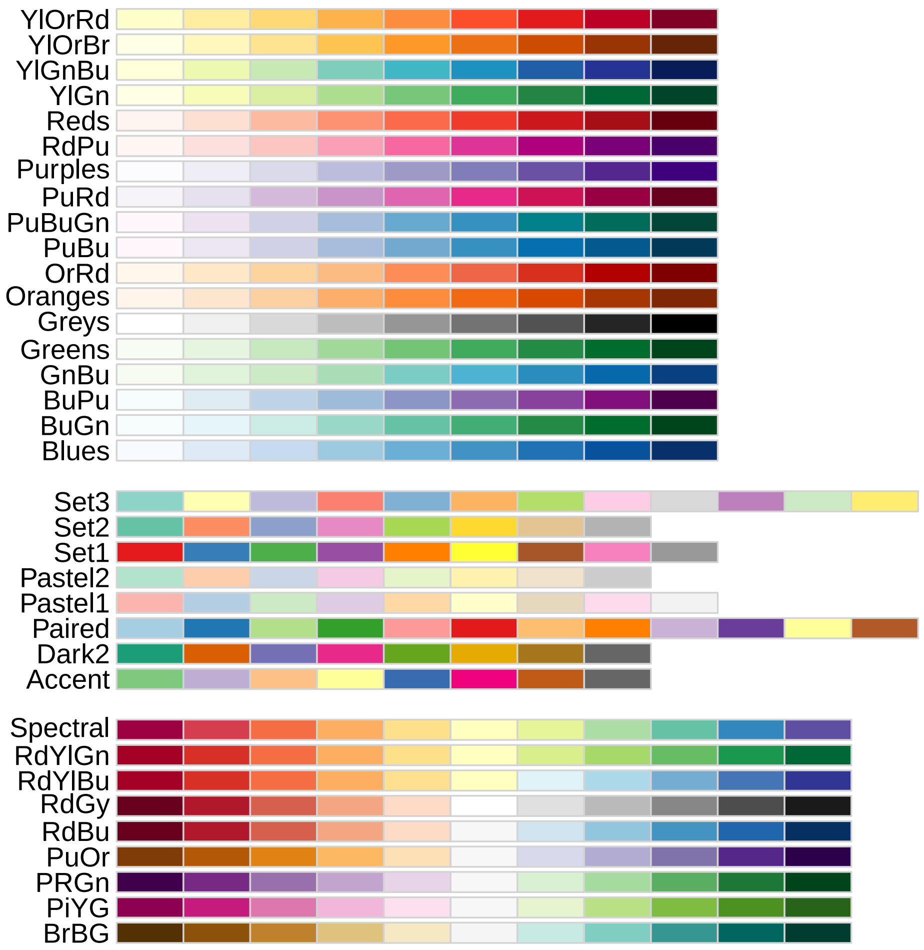
```
cols(10)
```

```
##  [1] "#FF0000" "#E2001C" "#C60038" "#AA0055" "#8D0071" "#71008D" "#5500AA"
##  [8] "#3800C6" "#1C00E2" "#0000FF"
```

Or, you can use the `RColorBrewer` package to get a premade colour palette.

There are three types of palettes: * Sequential * Diverging * Qualitative

```
library(RColorBrewer)
display.brewer.all()
```



5.7 Multiple plots

There are two useful packages for combining multiple plots: `gridExtra` and `cowplot`.

`gridExtra` has two useful functions: `grid.arrange()` and `arrangeGrob()`. However, these functions make no attempt at aligning the plot panels; instead, the plots are simply placed into the grid as they are, so the axes are not aligned. If axis alignment is required, the `plot_grid()` function of the `cowplot` package is better. We will try using both here.

5.7.1 The `gridExtra` package

5.7.2 The `cowplot` package

5.7.3 Saving plots

The easiest way to save a plot is using the `ggsave()` function.

5.8 Additional Resources

http://www.sthda.com/english/wiki/be-awesome-in-ggplot2-a-practical-guide-to-be-highly-effective-r-software-and-data-visualization

http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html

- Mailing list: http://groups.google.com/group/ggplot2
- Wiki: https://github.com/hadley/ggplot2/wiki
- Website: http://had.co.nz/ggplot2/
- StackOverflow: http://stackoverflow.com/questions/tagged/ggplot

Cheatsheet

Chapter 6

Rmarkdown

“If I went back to college again, I’d concentrate on two areas: learning to write and to speak before an audience. Nothing in life is more important than the ability to communicate effectively.” – Gerald R. Ford

6.1 Writing in R Markdown

6.2 Learning Objectives

Learn how to generate reproducible reports that display your code and results.

When you perform wet lab experiments, what information do you put in your lab notebook? You probably include the protocol you used to run the experiment, information about the samples and reagents used in the protocol, and at the end you’ll likely include your results (for instance, a picture of a gel). This essentially creates a report of your experiment.

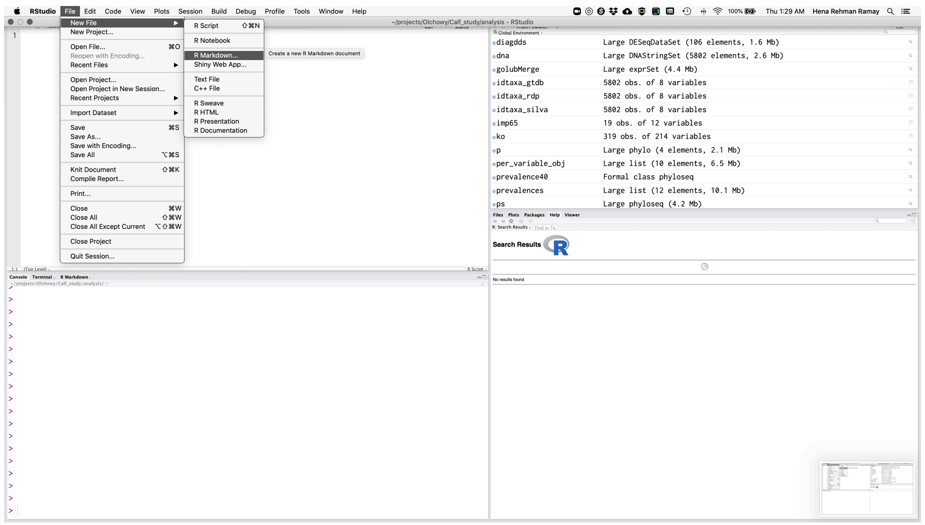
You can do the same with your dry lab analyses using a tool called R Markdown. Why would we want to do this?

- Your method, results, and interpretation are stored in one place
- If you update your methodology, you can easily update your results with the click of a button, rather than copying and pasting.
- You *could* cut and paste your code and results into Word or Power Point, but that will make rerunning your code challenging, as Word often introduces hidden characters.

R Markdown is a fairly simple language you can use to generate reports that incorporate bits of R code along with the output they produce. There are two steps to generating reports with R Markdown and RStudio:

1. Write your code in R Markdown.
2. Assemble your report as either HTML or a PDF using the package rmarkdown.

Next, let's run through the demo R Markdown file to see some of the options. Go up to `File -> New File -> R Markdown`.



```
diagdds            Large DESeqDataSet (106 elements, 1.6 Mb)
dna                Large DNAStringSet (5802 elements, 2.6 Mb)
golubMerge         Large exprSet (4.4 Mb)
idtaxa_gtdb        5802 obs. of 8 variables
idtaxa_rdp         5802 obs. of 8 variables
idtaxa_silva       5802 obs. of 8 variables
imp65              19 obs. of 12 variables
ko                 319 obs. of 214 variables
p                  Large phylo (4 elements, 2.1 Mb)
per_variable_obj   Large list (10 elements, 6.5 Mb)
prevalence40       Formal class phyloseq
prevalences        Large list (12 elements, 10.1 Mb)
ps                 Large phyloseq (4.2 Mb)
```

Figure 6.1: Set up new R Markdown file

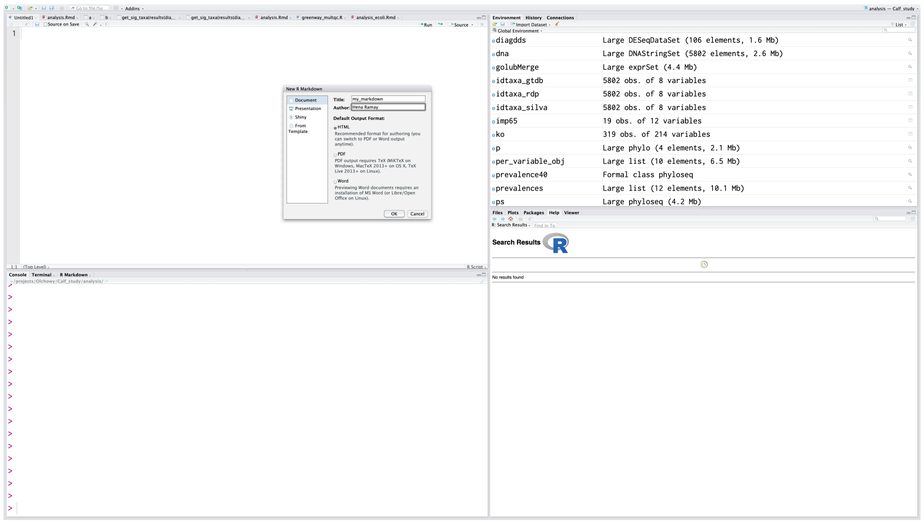
Set up new R Markdown file

A screen will pop up asking us what kind of document we wish to create. Let's name our demo report "Trial Report" and fill in your name. Ensure that "Document" is highlighted to the left and that "HTML" is chosen. Click "Ok".

Choose HTML

Now we have the example R Markdown file open. The first thing you'll notice at the top is a header which includes your name, the title of the document, the date, and a field called output. This header tells the package rmarkdown some information it might need about your document, including what format you want the final report rendered in.

The next thing you'll notice is white space with some text describing an R Markdown document. White space in this document represents text of the report you would like to display. You can put anything here describing your analysis, results, etc. and it will be recognized as text and not R code. This white space is interpreted as Markdown language, so you can use any of the


New R Markdown

Title: my_markdown
Author: Hena Ramay

Default Output Format:

[✓ HTML]
Recommended format for authoring (you can switch to PDF or Word output anytime).

PDF
PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

Word
Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

diagdds	Large DESeqDataSet (106 elements, 1.6 Mb)
dna	Large DNAStringSet (5802 elements, 2.6 Mb)
golubMerge	Large exprSet (4.4 Mb)
idtaxa_gtdb	5802 obs. of 8 variables
idtaxa_rdp	5802 obs. of 8 variables
idtaxa_silva	5802 obs. of 8 variables
imp65	19 obs. of 12 variables
ko	319 obs. of 214 variables
p	Large phylo (4 elements, 2.1 Mb)
per_variable_obj	Large list (10 elements, 6.5 Mb)
prevalence40	Formal class phyloseq
prevalences	Large list (12 elements, 10.1 Mb)
ps	Large phyloseq (4.2 Mb)

Search Results

No results found

Figure 6.2: Choose HTML

tricks we learned in the last lesson to make lists, bold certain words, or create headers in your document.

In this trial script, you'll see how some of these markdown elements are used. For example, the word knit is in bolded (using asterisks), and there are code chucks near the bottom that say echo = FALSE.

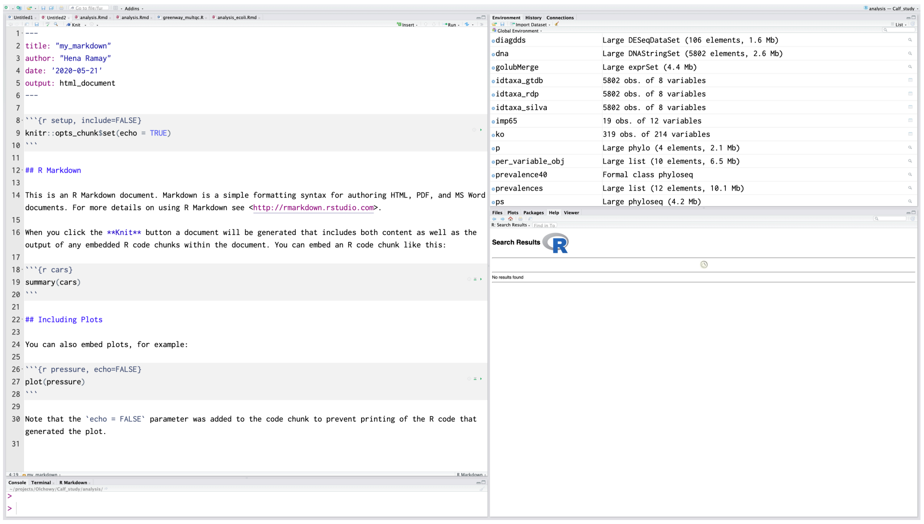
Demo R Markdown Document

In addition to the white space, you'll gray blocks that have "' at the top and bottom. These are called chunks. If the start of a chunk has {r} at the end of the ticks, the code will be run and both it and its output will be displayed in the rendered HTML. In your R Markdown, the code will look like:

~~~
```r
summary(cars)

##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  


In your final report, the code will look like:
~~~



~~~
---
title: "my_markdown"
author: "Hena Ramay"
date: '2020-05-21'
output: html_document
---

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

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word
documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

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. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```

Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```

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

```
diagdds             Large DESeqDataSet (106 elements, 1.6 Mb)
dna                 Large DNAStringSet (5802 elements, 2.6 Mb)
golubMerge          Large exprSet (4.4 Mb)
idtaxa_gtdb         5802 obs. of 8 variables
idtaxa_rdp          5802 obs. of 8 variables
idtaxa_silva        5802 obs. of 8 variables
imp65               19 obs. of 12 variables
ko                  319 obs. of 214 variables
p                   Large phylo (4 elements, 2.1 Mb)
per_variable_obj    Large list (10 elements, 6.5 Mb)
prevalence40        Formal class phyloseq
prevalences         Large list (12 elements, 10.1 Mb)
ps                  Large phyloseq (4.2 Mb)
```

No results found

```
4:19  my_markdown                                   R Markdown
~/projects/Olchowy/Calf_study/analysis/
>
>

Figure 6.3: Demo R Markdown Document

summary(cars)

Let's add a new chunk to end this demo document. To do so, either you can enter three backti

![Insert Chunk](https://jdblischak.github.io/r-intermediate-altmetrics/fig/insert-chunk

Insert Chunk

In the chunk, let's just examine the dimensions of the `car`mdataset:

```r
dim(cars)
```

```
## [1] 50  2
```

You can actually send the code straight from the chunks over to console to be evaluated in t

These are the basics of writing R Markdown, but we still need to generate a report. To do thi

![Knit R Markdown](https://jdblischak.github.io/r-intermediate-altmetrics/fig/knit_rmar

Knit R Markdown

When you click on this link, you see in the console that RStudio is running and rendering yo

The final result is that an HMTL file will pop up where you'll see the report. You can see th

```
Also, if you now look in the altmetrics folder, you'll see an HTML file of the name Rmarkdown_demo.html

Knitr Chunk Options
-------------------

Learning Objectives
-------------------

Learn how format chunks in R Markdown to display only the information you want to display.

You've learned the basics of how to incorporate markdown syntax with code chunks in an R Markdown file

The first thing you may want to consider is naming your code chunks, which makes degubbing easier, esp

Write the name of your chunk after the {r}, like: `{r chunk_name}`

R Markdown:
```

```r
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

```
Rendered:
```

summary(cars)

```
You can use RStudio to navigate to chunks based on their names, which can be especially useful as your

Sometimes you may not want to see the code that produced a particular result in your report. You can ha

R Markdown:
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

```
Rendered:
```

summary(cars)

```
Conversely, sometimes you may want to see the code, but not the output once the code is eval
```

```
R Markdown:
```

```
summary(cars)
```

```
Rendered:
```

summary(cars)

```
Sometimes, you may want to write a report where both the code and the output are suppressed.

R Markdown:

Rendered (I swear there's a chunk after this! It's just invisible!):

There are tons of other options you can include in your chunks: sizing your figures and whet

In addition to code chunks, you may want to include the results of an evaluation in line wit

R Markdown:

The _cars_ dataset included in this analysis contains records for 50 cars.

Rendered:
```

The *cars* dataset included in this analysis contains records for 50 cars. “‘

Chapter 7

Project

2 hours project in which we make plots, write functions and generate a nice html report

Chapter 8

Interesting Packages

1. Cowplot
2. forcats
3. biobroom
4.

Chapter 9

FAQs

A subset of questions that were asked during the session