Getting Started with R, RStudio, & R Markdown for ENVH 556: Part I

Lianne Sheppard

Created for Winter 2021; printed 26 December, 2020

1 Why R Markdown and reproducible reports?

All good data analysis should be reproducible. Markdown is a popular variant of the syntax often used in Wikis, such as Wikipedia. R Markdown is an extension of Markdown to support the execution and rendering of R code within the document. It integrates easily with R to allow you to produce reproducible reports. We will use these tools and reproducible research practices to support your development in this course.

1.1 Credit

This document was modified and expanded from one created by Emily Voldal in fall 2018. It also incorporates input from Brian High.

The output (e.g. `.html` or `.pdf`) and source (`.Rmd`) documents are optimally used side-by-side so that you can see both the code and its result. Before you knit this yourself, you will need to install the packages `rmarkdown` and `knitr`. (These are already installed on UW-supported RStudio servers.)

1.2 What is Markdown, and why bother?

Just like an R script is better than typing code in the console, using R Markdown is better than using an R script. R Markdown documents allow you to save your code, the output that corresponds to your code, and a record of how each calculation and figure was created. Not only will this help you with the homework for this class, it's good practice for doing reproducible research.

Although a `.Rmd` document looks a lot different from an R script, the basics of R Markdown are straightforward compared to the rest of R. After you become familiar with the tricks of R Markdown, you may find it easier and faster to use than an R script.

2 Getting started

2.1 Log onto RStudio Server

In ENVH 556 we will ensure that all applications and labs work on either of 2 UW RStudio servers running on virtual Linux machines:

- Plasmid, for DEOHS and EPI students: plasmid.deohs.washington.edu.
- SPH server, for all students: rstudio.sph.washington.edu

For either of these UW servers you will need to install and sign into Husky OnNet VPN using your UW NetID.

Another (non-UW) RStudio server option is RStudio Cloud.

2.2 Install applications locally on your computer

You are welcome to do your work locally on your laptop, but we will not be able to provide support for local installations. You will need the most recent version of RStudio and R. An easy way to do this is to follow directions on a tutorial program (Swirl) which begins by walking you through the process of downloading R and RStudio. Here is the link to the Swirl tutorial: https://swirlstats.com/students.html. The first few Swirl lessons cover the "Basics of R Programming" which may help new R users become familiar with R and RStudio.

To work locally, make sure you have installed the `rmarkdown` package. For this document, you will also need to install `knitr`. If, at any point during this process, R tells you that you need to install other packages, do so.

2.3 Become familiar with R Markdown

To open a new R Markdown document (`.Rmd`), select 'File', 'New File', 'R Markdown'. You will see a window asking you for some information about your document; R will use this information to fill in some code in the `.Rmd` file, which you can change at any time. For now, leave the default HTML setting and fill in whatever you want for the title and author. Save this file, just like you would save an R script.

This is an R Markdown document. The following text is the beginning explanation in the text that is included in every new R Markdown documents you open:

> Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see https://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.

Every time you open a new `.Rmd` file, you will see this example. To turn this code into a nice document, press the 'knit' button at the top of the panel. A window will pop up with the knitted document. Every time you knit a document, in addition to showing you the preview in R, it will save the knitted document in the same location as the `.Rmd` file.

You can run code line-by-line from the `.Rmd` document as you build it. However, this runs in a different environment than the one used by `knitr`. So you may have different errors in each. It can be helpful to knit often.

Here are several good R Markdown resources from RStudio to help you get started:

- The Website
- The Reference Guide
- The Cheat Sheet
- The Definitive Guide

The post Getting Started with R Markdown — Guide and Cheatsheet on R-bloggers is also very helpful.

3 Anatomy of a `.Rmd` file

3.1 YAML header

The header is enclosed by dashes and is always at the top of the `.Rmd` file. By default, it will include a title, author, date, and what type of file the `.Rmd` will knit to. You can change the text of the title, author, and date here any time. These will show up at the top of your knitted document. You can also change the file type at any time. For example, if I write `word_document` instead of `html_document`, I will get a Word file. However, be aware that some commands are specific to certain document types, or just show up differently. It appears that `html_document` is the most flexible and least fussy type of output.

Whenever you open a new `.Rmd` file and see the example, leave the header and delete the rest of the example below the last `---`.

3.2 Text

To put plain text into an R Markdown document, you don't need anything special. Text that is black in the `.Rmd` document is plain text in the knitted document. Blue text in the `.Rmd` indicates that it has been formatted in some way by using special characters (for example, the knitted text may be bold). Part II of this document gives details on how to format text.

3.3 Chunks

Interspersed in the text are lines of code; these may have a shaded background in your `.Rmd` file; these are called 'chunks'. Chunks start with ` ```{r} ` and end with ` ``` `, each at the beginning of the line. The position of each chunk determines where its output ends up in the knitted document. We can control what the code and output of each chunk look like by changing 'chunk options'. If we removed all the text from a `.Rmd`, the chunks would make up the complete R script for that analysis. (The code appendix does this for you automatically; see below.)

Here is an example code chunk to generate some data and calculate its mean:

```
#-----example chunk-----------

# Code goes here; output appears below.  (Details about the code in this chunk
# are in Part II.)
set.seed(45)
a <- rnorm(mean=0, sd=2, n=20)
mean(a)
```

```
## [1] 0.22
```

There is more information on chunks in Part II.

4 R Markdown strategies to enhance reproducibility

4.1 Benefits of reproducible reports

Reproducible reports with embedded data analyses have many benefits, including:

- Eliminates typos and transcription errors by pulling results directly from R
- Automatically updates your results if you change other code (i.e. if I decided to remove one observation from my data set, I wouldn't have to re-type all my numbers)
- Creates a record of exactly how you calculated every number (so another scientist could easily reproduce your entire analysis, and you will never forget how you calculated something)

One way to ensure reproducibility is to use in-line code. This incorporates R results directly within text. To include code output in a sentence, we use the format of one backtick followed by "r", a space, and then some R code, and ending with a second backtick. For example, we can write: "The mean of the data is 0.22." See Part II for details on inline code.

In the remainder of this section, we suggest strategies that support reproducibility.

4.2 Use projects and relative file paths within them

An important principle is to keep your scripts, data files, and all other inputs and outputs within a project folder. Using an RStudio "project" makes this easy, decreasing reliance on "setwd" commands in your scripts. You would not refer to files outside of this project folder, but instead would use subfolders (e.g., "data", "images", etc.) to organize your work within your project. Using project folders also works nicely with version control (Git) as the project folder becomes your version control "repository" for that project.

Let's make a new RStudio project named "new_project". In the GUI click:

File –> New Project…

In the dialogue box that appears click:

New Directory –> New Project

We can make "new_project" a subdirectory of: "~/Home" and click "Create Project"

From here if your create a new `.Rmd` file it will automatically be placed within the project's working directory. We will also set up a new project in the first lab.

4.3 Automate preparation of the working environment

When you write code in R Markdown it needs to be completely self-contained - that is, it can't rely on anything you loaded, imported, or ran outside of the R Markdown document, e.g., in your RStudio session. An excellent principle is to automate the preparation of the working environment in your R Markdown document. This includes setting options and installing software packages.

4.3.1 Set options

The following chunk is an example of setting `knitr` options. (Note: We already set `knitr` options at the beginning of this document.)

```
#-----set knitr options-----------

suppressMessages(library(knitr))
opts_chunk$set(tidy=FALSE, cache=FALSE, echo=TRUE, message=FALSE)
```

You can adjust these to suit your needs at the time of rendering, such as disabling echo to make a report for someone who might be distracted by seeing R code. Or enable cache once your script is complete and working to allow you to more quickly render the script, such as when knitting a slide presentation just before you delivering a talk.

For more guidance on `knitr` options, see knitr options or the knitr Cheat Sheet.

4.3.2 Install and load packages

A key principle is to only load the packages you will need for your project.
To facilitate this, use the `pacman` package in R instead of `install.packages()` and `library()`. This will allow your script to automatically install any packages it needs to without forcing the installation of a package which has already been installed. Here is an example:

```
#-----setup pacman-----------

# Not evaluated here since done at the beginning of the file
# Load pacman into memory, installing as needed
my_repo <- 'http://cran.r-project.org'
if (!require("pacman")) {install.packages("pacman", repos = my_repo)}

# Load the other packages, installing as needed
# Key principle:  Only load the packages you will need
pacman::p_load(knitr, tidyverse)
```

The first part installs `pacman` if it is missing, then the second part installs and loads the other packages as needed. If you do this at the top of your script for any packages needed later in your script, it makes it really easy for people to see what packages your script depends on. This approach will make it much more likely someone, particularly a new R user, will be able to run your script and reproduce your results. Many new R users get completely stuck if they run code that bombs simply because a package has not been installed. And just putting in `install.packages()` calls "just in case" will needlessly slow down your script (each and every time it is run) if the packages have already been installed.

4.4 Use relative file paths, not absolute file paths

Best practice is to not include full paths ("C://Project_1/data/raw") to your files, but uses relative paths ("data/raw") instead, so they will be more portable – able to run on someone else's system. Even better, use `file.path()` to construct these paths to make them platform independent, so a person can use, e.g., Windows, macOS, or Linux to reproduce your results. Windows users take note: paths like "C:\data\raw" or even "data\raw will" not work on a Mac. Use "data/raw", or better yet, `file.path("data", "raw")` to solve this problem.

```
#-----file paths---------

# Only works on Windows, as other modern operating systems (macOS, Linux) do
# not support "drive letters", such as "P:"
full_path <- 'P:\\ENVH556'
full_path <- 'P:/ENVH556'

# Will not work on Windows, exposes your username, and won't work for other
# users
full_path <- '/Users/joanna/ENVH556'

# Will work on Windows, macOS, and Linux, etc., if the file is in the user's
# "home directory"
relative_path <- '~/ENVH556'

# If you are curious about what the "~" expands to, you can use path.expand()
full_path_to_home <- path.expand('~')
full_path_to_data <- path.expand('~/ENVH556')

# This will work on any system if "ENVH556" is one level below the current folder
relative_path <- 'ENVH556'

# This will work if "ENVH556" is at the same level as current folder, where ".."
# means "parent folder", since "ENVH556" will share the same parent as your
# current "working directory"
relative_path <- '../ENVH556'
```

4.4.1 Using the working directory

Using working directories (folders) means you don't have to use folder names when accessing files. A working directory is the place that R will look for files if you don't give a file path. R is always set to a certain working directory; you can check where that is like this:

```
#-----getwd---------

getwd()
```

You can make your default working directory the location of your `.Rmd` file; it's different when you're running code in your console or in a .R file. If you want to change that default, you can use `setwd()`. For example, let's say my RDS file isn't in the same folder as my `.Rmd`:

```
#-----setwd example---------

# Here we use a full file path, but it would be better to use a relative one
setwd("P:/ENVH556")

# Here we use a relative file path, where ".." means "one folder up"
setwd("../ENVH556")

# Here we use a relative file path with file.path(), the recommended method
# because this method supports muliple computing platforms (Windows, Mac, etc.)
setwd(file.path("..", "ENVH556"))

# Now we are ready to read the file
DEMS <- readRDS("DEMSCombinedPersonal.rds")

# You can see that my working directory changed with getwd()
getwd()
```

Using working directories would be especially helpful if you needed to load lots of different data files and you didn't want to type out all the file paths. If you always put your `.Rmd` and data files in the same location, you should be able to use the default working directory without typing file paths.

One mild word of warning: there are some issues with working directories in R Markdown. If you are having trouble, you may want to make sure that `setwd()` is in the same chunk as your `readRDS()` command. There is also a more elegant solution to this.

4.5 Reporting code in the Appendix

When you turn in assignments, in addition to your well-written answers (which should not have any code or raw output), you will need to turn in the actual code you used. The lab template for ENVH 556 provides you with some standard appendix code to use in every ENVH 556 document. The chunk options to accomplish this are:

```
ref.label=knitr::all_labels(), echo=TRUE, eval=FALSE, include=TRUE
```

For an example of how to use this, I added a code appendix at the very end of this document (see last chunk at the end). Note that you will still want to …

4.6 Document your code

We encourage you to adopt these best practices when comment your code to improve readability:

- Comment to keep a running commentary on what your code does.
- The level of detail should be enough to clarify but not to enough to annoy.
- Insert comments immediately before the line(s) of code to which they apply.
- Add spaces and blank lines as needed to separate code and comments.
- Avoid "side-commenting", putting comments at the end of a line of code.
- Comments should state what the code does, in the form of an imperative.

Here is an example of these put into practice:

```
#-----comment example-----------

# Create a vector of temperatures in degrees Celcius
temps <- c(21, 22, 20, 19, 19, 19, 22, 19)

# Calculate the mean temperature and standard deviation
mean(temps)
sd(temps)
```

4.7 Follow a consistent style

Choose a code style and be consistent within your .Rmd. Following established styles will make your code more readable and easier to follow - for both you and others. Here are two popular examples:

- The Tidyverse Style Guide
- Google's R Style Guide

5 R Markdown resources

5.1 Tutorials for R Markdown:

- The official tutorial - This tutorial has lots of pictures and is well-organized, but only covers the basics.
- Getting Started, by 'John' - This is really great for someone who has never used R Markdown before, and includes tutorials on setting up R Markdown and fixing problems. It also has a great table.
- A book by Yihui Xie - This goes into a lot of detail, and is really well organized and clear.
- Using R Markdown for Class Reports, by Cosma Shalizi - This covers many R Markdown capabilities in R Markdown. It doesn't provide a lot of detail, but it is a good starting point if you're wondering whether you can do something in R Markdown.

5.2 Useful R Markdown cheat sheets and reference guides:

- Official cheat sheet
- Reference guide
- Another cheat sheet

5.3 R for Data Science

The R for Data Science (R4DS) book by Hadley Wickham and Garrett Grolemund is the current modern standard for using R.

- Chapter 8 describes projects and why you should use them
- Chapter 26 is the introduction to the Communication section with a brief overview of the following chapters
- Chapter 27 introduces R Markdown
- Chapter 30 covers R Markdown workflow

6 Appendix

At a minimum, record version numbers of R and your packages with `sessionInfo()` at the end of your script and record the output as an appendix.

```
#-----print session information-----

sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Catalina 10.15.7
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] tinytex_0.28  formatR_1.7   ggplot2_3.3.2 dplyr_1.0.2   readr_1.4.0
## [6] knitr_1.30    pacman_0.5.1
##
## loaded via a namespace (and not attached):
##  [1] magrittr_2.0.1   hms_0.5.3        munsell_0.5.0    tidyselect_1.1.0
##  [5] colorspace_2.0-0 R6_2.5.0         rlang_0.4.9      stringr_1.4.0
##  [9] tools_4.0.3      grid_4.0.3       gtable_0.3.0     xfun_0.19
## [13] withr_2.3.0      htmltools_0.5.0  ellipsis_0.3.1   yaml_2.2.1
## [17] digest_0.6.27    tibble_3.0.4     lifecycle_0.2.0  crayon_1.3.4
## [21] purrr_0.3.4      vctrs_0.3.6      glue_1.4.2       evaluate_0.14
## [25] rmarkdown_2.6    stringi_1.5.3    compiler_4.0.3   pillar_1.4.7
## [29] scales_1.1.1     generics_0.1.0   pkgconfig_2.0.3
```

For ENVH 556, we also want to see all of your code consolidated at the end of your R Markdown output. The following code will compile all your code into an appendix code listing. (This next version is for display in the rendered document and not for execution. It is followed by a working version that creates the appendix. Note that the chunk header containing these options should be a single line of code with no line-wrap.)

````
```{r appendix, ref.label=knitr::all_labels(), echo=TRUE, eval=FALSE,
include=TRUE}
```
````

For ENVH 556, if not already in a template, copy and paste the following chunk into the end of every lab assignment: (This is the version that executes.)

```
#-----setup------

# Set knitr options:
knitr::opts_chunk$set(echo = TRUE, warning = FALSE, messages = FALSE,
                      tidy.opts=list(width.cutoff = 80, blank = TRUE) )

# Set R option: show only 2 digits when displaying
options(digits = 2)

# Clear workspace of all objects and unload all extra (non-base) packages
rm(list = ls(all = TRUE))
if (!is.null(sessionInfo()$otherPkgs)) {
    res <- suppressWarnings(
        lapply(paste('package:', names(sessionInfo()$otherPkgs), sep=""),
               detach, character.only=TRUE, unload=TRUE, force=TRUE))
}

# Load key packages using pacman (see below for explanation)
my_repo <- 'http://cran.r-project.org'
if (!require("pacman")) {install.packages("pacman", repos = my_repo)}

# Load the other packages, installing as needed
# Key principle:  Only load the packages you will need
pacman::p_load(knitr, readr, dplyr, ggplot2, formatR,tinytex)

#-----example chunk-----------

# Code goes here; output appears below.  (Details about the code in this chunk
# are in Part II.)
set.seed(45)
a <- rnorm(mean=0, sd=2, n=20)
mean(a)

#-----set knitr options-----------

suppressMessages(library(knitr))
opts_chunk$set(tidy=FALSE, cache=FALSE, echo=TRUE, message=FALSE)

#-----setup pacman-----------

# Not evaluated here since done at the beginning of the file
# Load pacman into memory, installing as needed
my_repo <- 'http://cran.r-project.org'
if (!require("pacman")) {install.packages("pacman", repos = my_repo)}

# Load the other packages, installing as needed
# Key principle:  Only load the packages you will need
pacman::p_load(knitr, tidyverse)

#-----file paths---------

# Only works on Windows, as other modern operating systems (macOS, Linux) do
# not support "drive letters", such as "P:"
full_path <- 'P:\\ENVH556'
full_path <- 'P:/ENVH556'

# Will not work on Windows, exposes your username, and won't work for other
# users
full_path <- '/Users/joanna/ENVH556'

# Will work on Windows, macOS, and Linux, etc., if the file is in the user's
# "home directory"
relative_path <- '~/ENVH556'

# If you are curious about what the "~" expands to, you can use path.expand()
full_path_to_home <- path.expand('~')
full_path_to_data <- path.expand('~/ENVH556')

# This will work on any system if "ENVH556" is one level below the current folder
relative_path <- 'ENVH556'

# This will work if "ENVH556" is at the same level as current folder, where ".."
# means "parent folder", since "ENVH556" will share the same parent as your
# current "working directory"
relative_path <- '../ENVH556'

#-----getwd---------

getwd()

#-----setwd example---------

# Here we use a full file path, but it would be better to use a relative one
setwd("P:/ENVH556")

# Here we use a relative file path, where ".." means "one folder up"
setwd("../ENVH556")

# Here we use a relative file path with file.path(), the recommended method
# because this method supports muliple computing platforms (Windows, Mac, etc.)
setwd(file.path("..", "ENVH556"))

# Now we are ready to read the file
DEMS <- readRDS("DEMSCombinedPersonal.rds")

# You can see that my working directory changed with getwd()
getwd()

#-----comment example-----------

# Create a vector of temperatures in degrees Celcius
temps <- c(21, 22, 20, 19, 19, 19, 22, 19)

# Calculate the mean temperature and standard deviation
mean(temps)
sd(temps)

#-----print session information-----

sessionInfo()

#-----appendix-----
```