Week 1 Worksheet: Installing R, Rmarkdown, Rbasics

*David Gerard*

2019-01-10

Learning Objectives

- Install R

- Understand the basics of R (Chapters 1, 2, and 4)

- Rmarkdown (Chapter 27)

Installations

- Install R from the Comprehensive R Network (CRAN): https://cran.r-project.org/

    - You will have to choose the appropriate operating system. If you are using Ubuntu, there are special instructions if you want the newest version of R (v3.5.2 as of writing this document) as the default is to install an older version. This is not too important for this class, but some packages (mainly the ones in Bioconductor) require the newest version of R. If you want to use these packages, you should follow these instructions: https://cran.r-project.org/bin/linux/ubuntu/README.html

- Install RStudio: https://www.rstudio.com/

    - RStudio is **different** from R. R is a statistical programming language. RStudio is an Integrated Development Environment (IDE), a software application used to write R code. It contains tools (such as shortcut buttons and syntax autocmpletion and highlighting) that make coding in R much easier. Most data scientists either use RStudio, Jupyter Notebook, or Emacs Speaks Statistics to interact with R.

Getting Started

- When you first open up RStudio, it should look something like this:
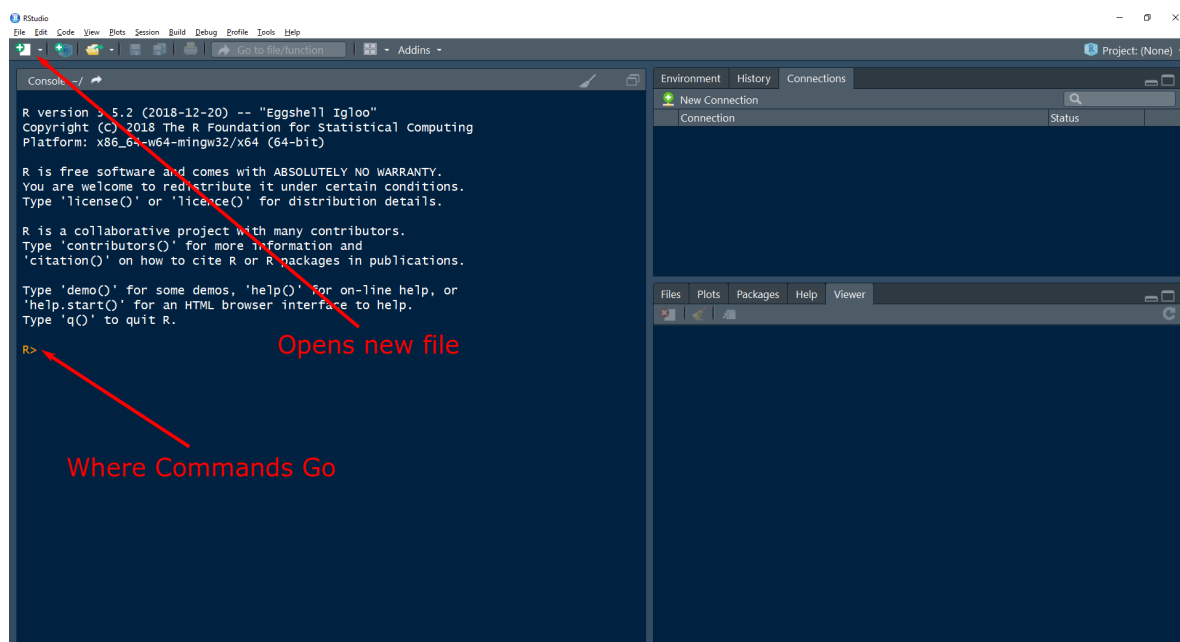
- The area right next to the carrot ">" is called a **prompt**. You insert commands into the prompt.

- You can use R as a powerful calculator. Try typing some of the following commands into the command prompt:

```
3 * 7
9 / 3
4 + 6
3 - 9
(3 + 5) * 6
3 ^ 2
4 ^ 2
```

- What do you think **%%** does? Try it out on a few examples:

```
1 %% 2
2 %% 2
3 %% 2
4 %% 2
1 %% 3
2 %% 3
3 %% 3
4 %% 3
```

- In general if you don't know what something does, there are three things you can do:

  1. Google it. Always add "in R" at the end of your search term. E.g. "%% in R".
  2. Use the `help()` function to see if there is a help page on it. E.g., type `help("%%")` in the command prompt. This is the most important function in R.
  3. Play around with it, changing the inputs until you figure out what it does.

RStudio Options

- RStudio will automatically save a bunch of leftover stuff when you exit it. This is bad for reproducibility because you might be depending on code that you ran in a previous session (which you no longer can view). We'll get rid of these silly options.

- Open up "Tools > Global Options... > General" and

  - **Uncheck** "Restore .RData into workspace at startup".
  - Change "Save workspace to .RData on exit" to "Never".

R Basics

Variables

- R consists of two things: **variables** and **functions** (computer scientists would probably disagree with this categorization).

- A **variable** stores a value. This value can be:

  1. A **numeric** (a number): `1`, `2`, `3`, `4`

  2. A **string** (a word/character). These are always surrounded by quotation marks: `"Hello"`, `"world"`, `"32"`.

  3. A **logical** (either `TRUE` or `FALSE`).

  4. A **factor** (categorical variable), more on this later.

  5. A **vector** of numerics/strings/logicals/factors. A vector is a sequence of something of the same type. E.g.

```
## [1] 1 2 3 4
```

```
## [1] "Hello" "world" "32"
```

6. A **matrix** of numerics/strings/logicals/factors. A matrix is a rectangle of elements of the same type. E.g.

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
##      [,1]   [,2]     [,3]
## [1,] "this" "a"      "of"
## [2,] "is"   "matrix" "strings"
```

7. A **list** of any sequence of objects.

```
## [[1]]
## [1] "hello"
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] TRUE
```

8. A **data frame** where each column is a vector of a certain type (a vector of numerics or strings or logicals or factors).

- You **assign** variables values with `<-` (you can also use `=`, but only noobs do that). For example, assign `x` the value of `7`.

```
x <- 7
```

- If you type `x` now into the command promt, it will return what is stored in `x`:

```
x
```

- Creating variables is useful because it allows you to reuse those values over and over again.

- You can apply arithmetic operations to variables as you would to numbers.

```
y <- 9
x + y
x * y
x / y
x - y
```

- You assign strings and logicals via:

```
z <- "I AM A STRING"
x <- TRUE
```

- Notice that we overwrote `x` to now be `TRUE` instead of `7`. This is possible in R (in other languages the type is pre-defined and not changed easily).

- I've chosen horrible variable names so far. Generally variable names should be:
  - Nouns
  - Informative about what it is storing, yet concise.
  - Use either the `camelCaseSystem` or the `snake_case_system`.
  - You are also able to use periods in variable names, but only noobs do that (it makes things more confusing in more advanced R).

- Variables are **case sensitive**, so `Variable` is different from `variable`. Try this out:

```
variable <- 1
Variable <- 2
variable
Variable
```

- When I write a variable inline, it will look like `this`.

- Exercise: Why doesn't this code work?

```
my_variable <- 10
my_variab1e
```

```
data0 <- 1
dataO
```

Functions

- Functions take objects (such as numbers or variables) as input and output new objects. Let's look at a simple function that takes the log of a number:

```
log(x = 4, base = 2)
```

- The inputs are called "arguments". Generally, every function will be for the form:

```
function_name(arg1 = val1, arg2 = val2, ...)
```

- If you do not specify the name of the argument, R will assume you are assigning in their order.

```
log(4, 2)
```

- You can change the order of the arguments if you specify them.

```
log(base = 2, x = 4)
```

- To see the list of all possible arguments of a function, use the `help()` function:

```
help(log)
```

- In the help file, there are often **default** values for an argument. For example, the following indicates the the default value of `base` is `exp(1)`.

```
log(x, base = exp(1))
```

- This indicates that you can omit the `base` argument and R will assume that it should be `exp(1)`.

```
log(x = 4, base = exp(1))
```

```
## [1] 1.386
```

```
log(x = 4)
```

```
## [1] 1.386
```

- If an argument does not have a default, then it must be specified when calling a function.

- Type this:

```
log(x = 4,
```

- The "+" indicates that R is expecting more input (you forgot either a parentheses or a quotation mark). You can get back to the prompt by hitting the ESCAPE key.

- There are a *lot* of functions that come pre-installed with R. Don't try to memorize the list of all of them functions. The best course of action is to Google the functionality you need and you'll automatically pick up the names as you go.

- But here is a reference card with some basic functions: https://cran.r-project.org/doc/contrib/Short-refcard.pdf

- A **package** is a collection of functions that don't come with R by default.

- There are **many many** packages available. If you need to do any data analysis, there is probably an R package for it.

- Using `install.packages()`, you can install packages that contain functions and datasets that are not available by default. Do this now with the tidyverse package:

```
install.packages("tidyverse")
```

- You will only need to install a package once per computer. Once it is installed you can gain access to all of the functions and datasets in a package by using the `library()` function.

```
library(tidyverse)
```

- You will need to run `library()` at the start of every R session if you want to use the functions in a package.

- When I want to write the name of a function, I will write it like `this()`.

- Generally function names should be:

    - Verbs
    - Informative about what it is does, yet concise.
    - `useTheCamelCaseSystem()` or `use_the_snake_case_system()`.
    - You are also able to use periods in function names, but only noobs do that (it makes things more confusing in more advanced R).
    - But lots of base R functions have periods in them: `is.na()`, `which.max()`, `data.frame()`, etc.

Rmarkdown

- Rmarkdown is a file format that is a combination of plain text and R code.

- You write code and commentary of code in one file. You may then compile (RStudio calls this "Knitting") the Rmarkdown file to many different kinds of output: pdf (including beamer presentations), html (including various presentation formats), Word, PowerPoint, etc.

- Rmarkdown is useful for:

    1. Communication of statistical results.
    2. Collaborating with other data scientists.
    3. Using it as a modern lab notebook to *do* data science.

- *You will use Rmarkdown for all assignments in this class.*

- Open up a new Rmarkdown file:
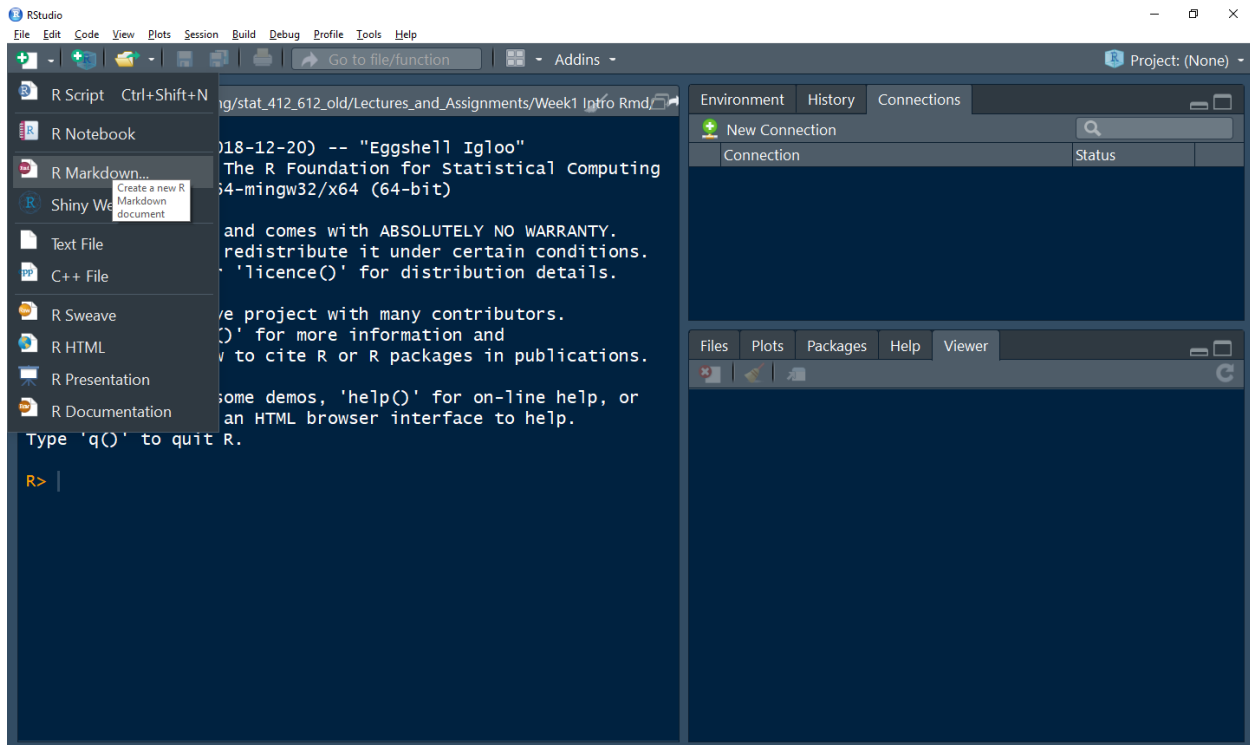
- Choose the options for the type of output you want
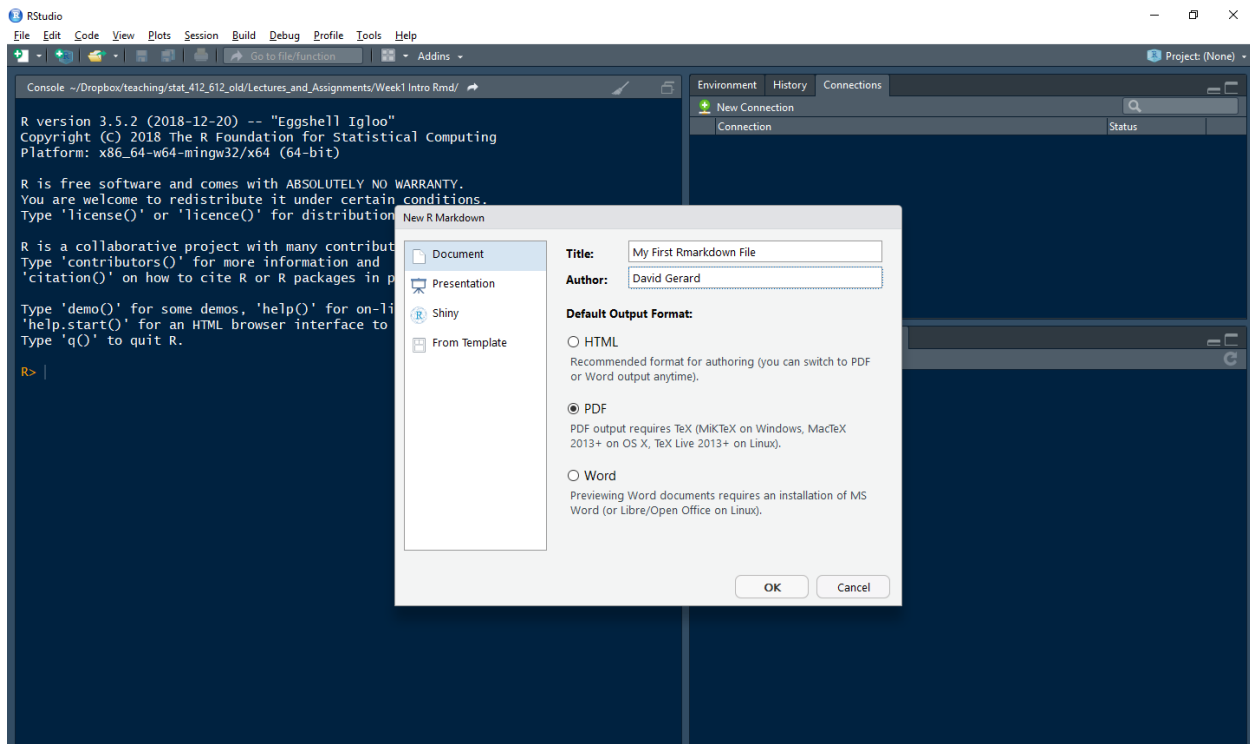
- You should now have a rudimentary Rmarkdown file.

- Save a copy of this file in your "analysis" folder in the "week1" project.

- Rmarkdown contains three things

1. A YAML (Yet Another Markup Language) header that controls options for the Rmarkdown document.
2. Code **chunks** — bits of R code that look like this that are surrounded by "`{r}`" and "``". Only valid R code should go in here.
3. Plain text that contains simple formatting options.

- All of these are are displayed in the default Rmarkdown file. You can compile this file by clicking the "Knit" button at the top of the screen or by typing CONTROL + SHIFT + K. Do this now.

- Here is Hadley's brief intro to formatting text in Rmarkdown:

```
Text formatting
------------------------------------------------------------

*italic*  or _italic_
**bold**   __bold__
`code`
superscript^2^ and subscript~2~

Headings
------------------------------------------------------------

# 1st Level Header

## 2nd Level Header

### 3rd Level Header

Lists
------------------------------------------------------------

*   Bulleted list item 1

*   Item 2

    * Item 2a

    * Item 2b

1.  Numbered list item 1

1.  Item 2. The numbers are incremented automatically in the output.

Links and images
------------------------------------------------------------

<http://example.com>

[linked phrase](http://example.com)

![optional caption text](path/to/img.png)

Tables
------------------------------------------------------------
```

```
First Header   | Second Header
-------------- | --------------
Content Cell   | Content Cell
Content Cell   | Content Cell
```

- You can insert new code-chunks using CONTROL + ALT + i.