

Stat545

*Dookyung Kim*



# Contents

<b>Welcome</b>	<b>5</b>
<b>1 cm001 - introduction</b>	<b>7</b>
1.1 Intro to course; S/W install; acct sign-ups . . . . .	7
1.2 Deep Thoughts about data analytic work; intro to R and RStudio . . . . .	13
<b>2 R Markdown test drive</b>	<b>19</b>



# Welcome

\_\_\_\_“Stat545.com”\_\_\_\_ .  
<http://stat545.com/index.html>



# Chapter 1

## cm001 - introduction

### 1.1 Intro to course; S/W install; acct sign-ups

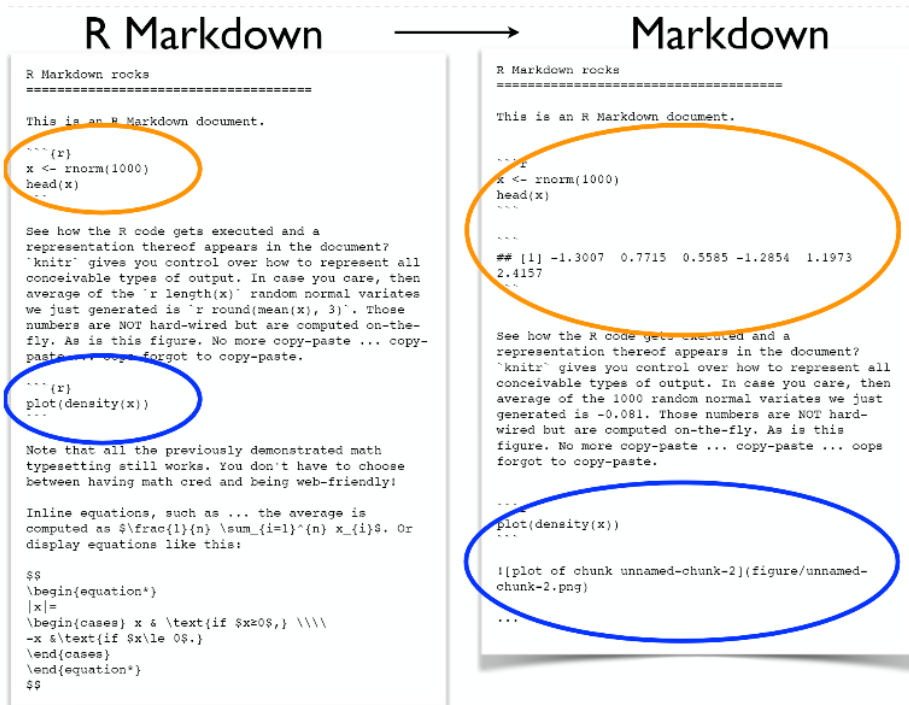
#### 1.1.1 Data Domination

“Software programming, algorithm development and other technological skills can give scientists an edge in their fields.” — Careers, 31 August 2017

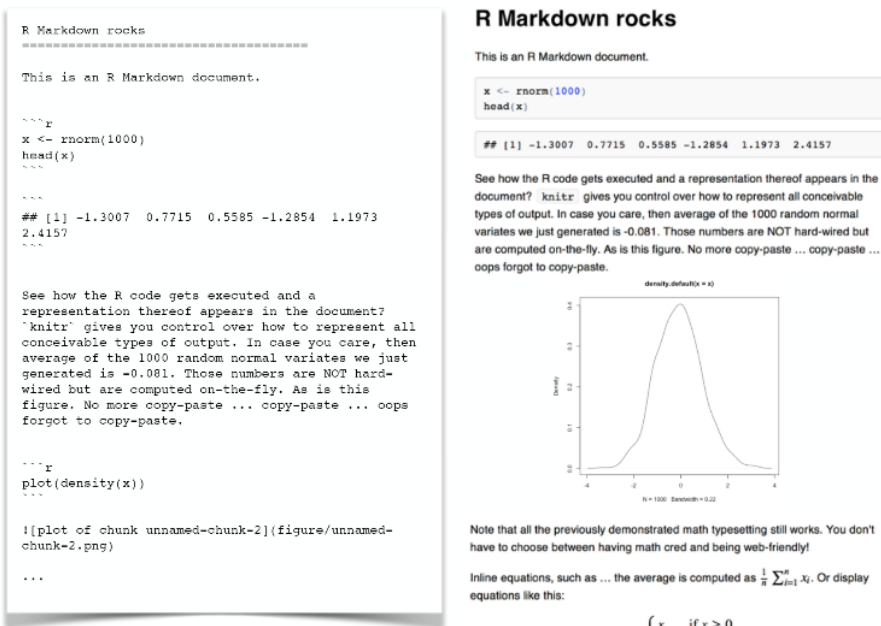
“A picture is worth a thousand words.”

#### 1.1.2 Data Science Tool :

- R
- Rstudio : an integrated development environment (IDE) for R
- Rmarkdown



## Markdown → HTML





R Markdown → Markdown → HTML

foo.rmd → foo.md → foo.html

easy to write  
(and read!)

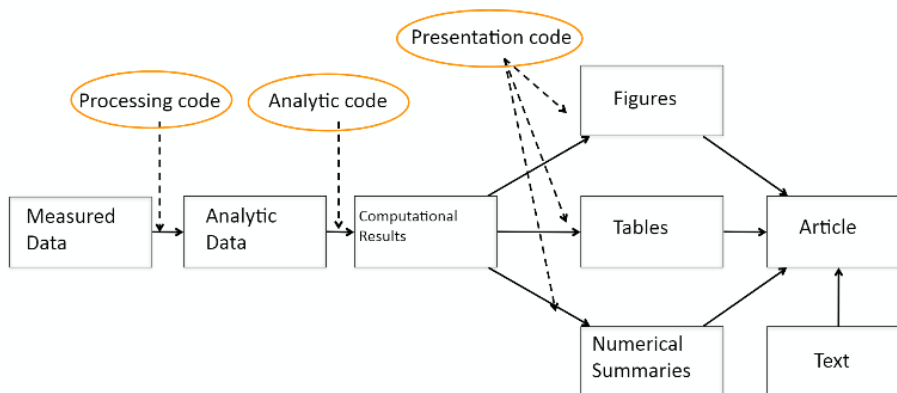
easy to publish  
easy to read in  
browser

- git and github : Version control systems(VCS) and collaboration tools.



- For a quick, stand-alone document that doesn't fit neatly into a repository or project (yet), make it a Gist
- GNU Make
  - How to make end products more integrated and more reproducible?
  - How to keep everything up-to-date?
  - if the data changes, how do we remember to remake the figure 2B and 4?

### R (or Python) scripts



Like Git,  
**GNU Make** is another old  
 school tool that is being  
 repurposed to meet a need in  
 data-intensive workflows.

Originally intended to  
 orchestrate compiling  
 complicated software, it's now  
 used to express what depends  
 on what and keep everything  
 “in sync”.

### Makefile

```

all: data model paper
data: raw.csv
model: model.Rout
paper: plot.Rout paper.pdf

raw.csv: get_data.py
python get_data.py

clean.csv: clean.sh raw.csv
source clean.sh

model.Rout: model.R clean.csv
R CMD BATCH model.R

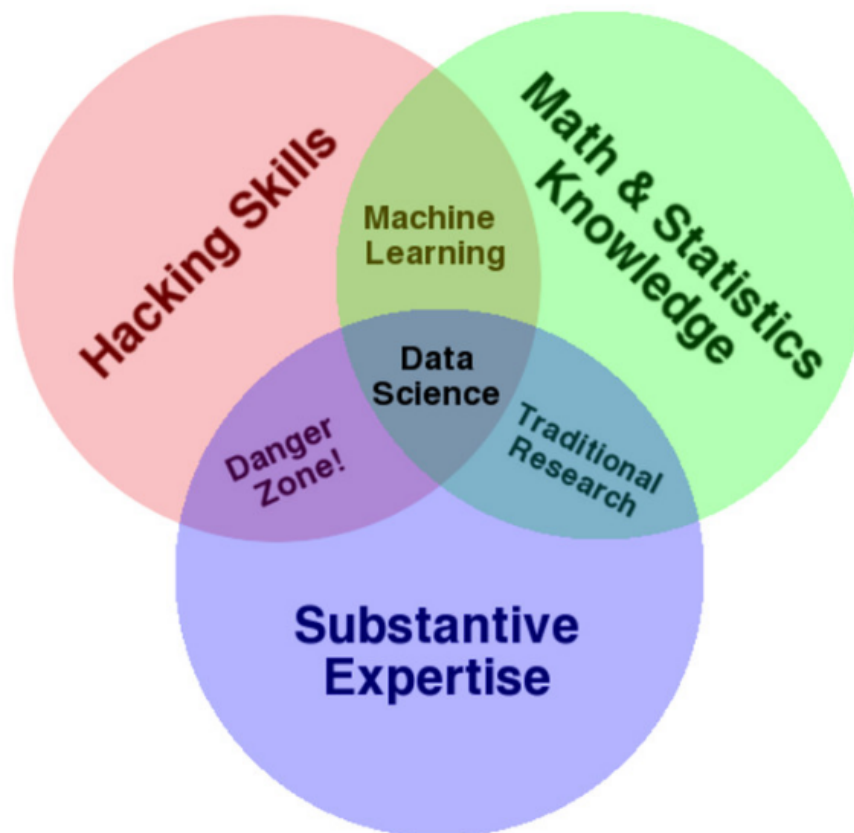
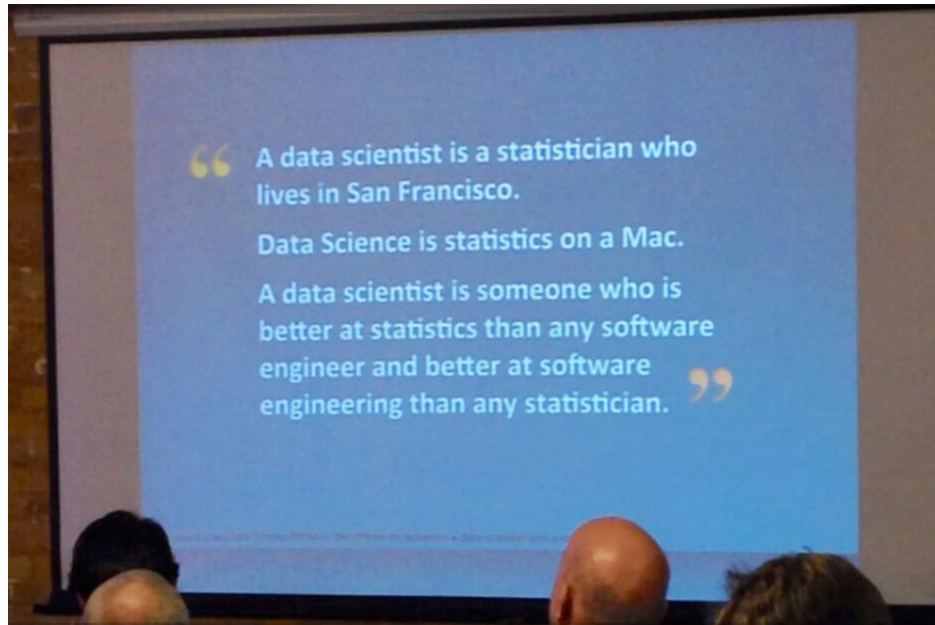
plot.Rout: plot.R model.Rout
R CMD BATCH plot.R

paper.pdf: paper.tex
$(TEXCMD) $<
$(TEXCMD) $<
bibtex *.aux
$(TEXCMD) $<

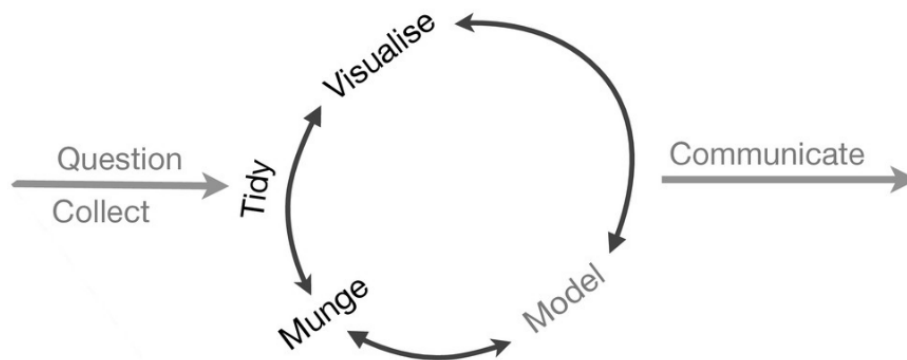
```

<http://zmjones.com/make.html>

### 1.1.3 What is Data Science?



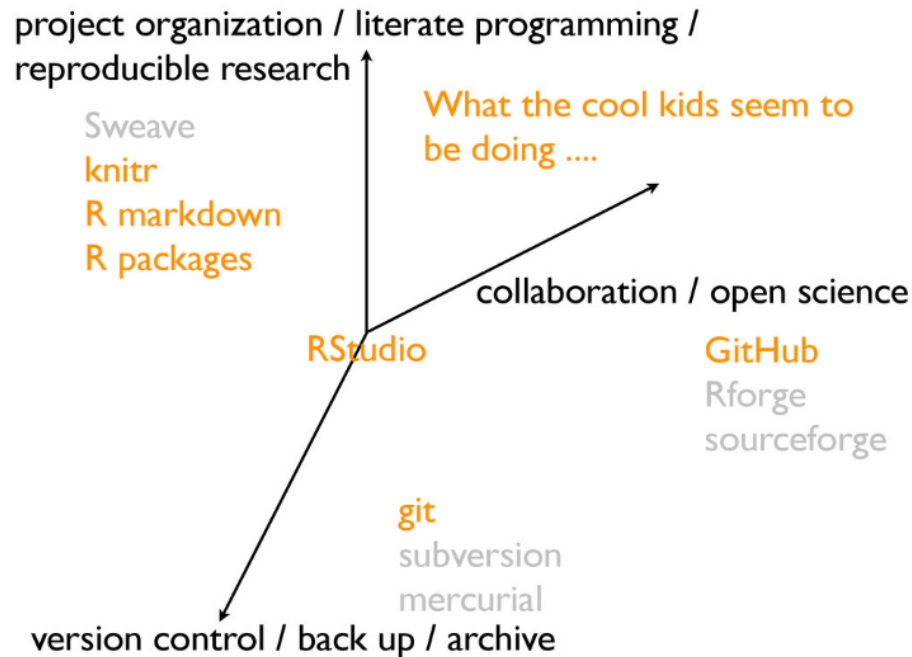
## a horizontal data science workflow



Slides from Hadley Wickham's talk in the Simply Statistics Unconference <http://t.co/D931Og8mq3>

big data  $\neq$  data science

big data  $\subset$  data science



## 1.2 Deep Thoughts about data analytic work; intro to R and RStudio

### 1.2.1 To do before next class:

- swirl: “a software package for the R programming language that turns the R console into an interactive learning environment”

### 1.2.2 Basics of working with R at the command line and RStudio goodies

- Launch RStudio/R and notice the default panes:
- Console (entire left)
- Environment/History (tabbed in upper right)
- Files/Plots/Packages/Help (tabbed in lower right)

Customizing RStudio.

- Make an assignment.

```
x <- 3 * 4
x
#> [1] 12
```

“assignments” form:

```
objectName <- value
```

- Keyboard shortcut for assignment operator <- is Alt + - (the minus sign).
- RStudio offers many handy keyboard shortcuts.
- Alt+Shift+K brings up a keyboard shortcut reference card.
- Object names

```
i_use_snake_case
other.people.use.periods
evenOthersUseCamelCase
```

Make another assignment

```
this_is_a_really_long_name <- 2.5
```

To inspect this, try out RStudio’s completion facility: type the first few characters, press TAB, add characters until you disambiguate, then press return.

Make another assignment

```
jenny_rocks <- 2 ^ 3
```

R has a mind-blowing collection of built-in functions that are accessed like so

```
functionName(arg1 = val1, arg2 = val2, and so on)
```

- seq() which makes regular sequences of numbers
  - Type `se` and hit TAB.

```
seq(1,10)
#> [1] 1 2 3 4 5 6 7 8 9 10
```

```
yo <- "hello world"
```

```
y <- seq(1, 10)
y
#> [1] 1 2 3 4 5 6 7 8 9 10
```

Surrounding the assignment with parentheses, which causes assignment and “print to screen” to happen.

```
(y <- seq(1, 10))
#> [1] 1 2 3 4 5 6 7 8 9 10
```

Not all functions have (or require) arguments:

```
date()
#> [1] "Sat Jul  7 13:48:28 2018"
```

- The workspace is where user-defined objects accumulate. You can also get a listing of these objects with commands:

```
objects()
#> [1] "jenny_rocks"      "this_is_a_really_long_name"
#> [3] "x"                "y"
#> [5] "yo"
ls()
#> [1] "jenny_rocks"      "this_is_a_really_long_name"
#> [3] "x"                "y"
#> [5] "yo"
```

remove the object named y

```
rm(y)
```

To remove everything:

```
rm(list = ls())
```

or click the broom in RStudio's Environment pane.

## 1.2.3 Workspace and working directory

### 1.2.3.1 Workspace, .RData

### 1.2.3.2 Working directory

You can explicitly check your working directory with:

```
getwd()
```

**Although I do not recommend it**, in case you're curious, you can set R's working directory at the command line like so:

```
setwd("~/myCoolProject")
```

**Although I do not recommend it**, you can also use RStudio's Files pane to navigate to a directory and then set it as working directory from the menu: Session → Set Working Directory → To Files Pane Location. (You'll see even more options there). Or within the Files pane, choose **More** and **Set As Working Directory**.

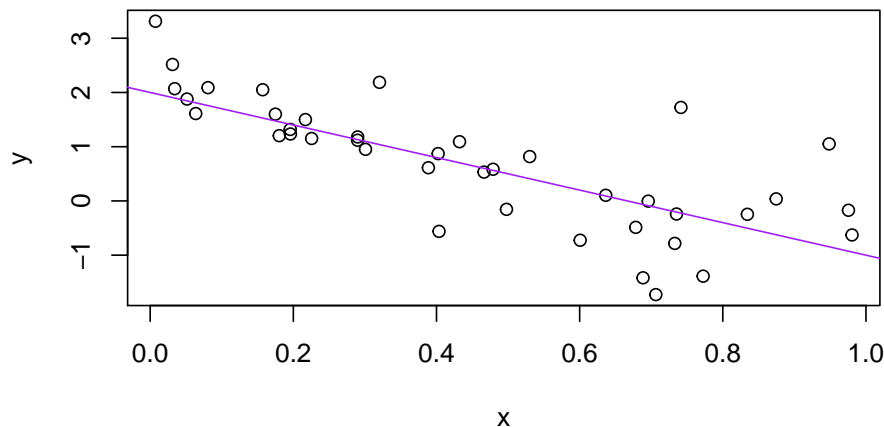
### 1.2.4 RStudio projects

Keeping all the files associated with a project organized together – input data, R scripts, analytical results, figures – is such a wise and common practice that RStudio has built-in support for this via its *projects*.

Using Projects

Let's enter a few commands in the Console, as if we are just beginning a project:

```
a <- 2
b <- -3
sig_sq <- 0.5
x <- runif(40)
y <- a + b * x + rnorm(40, sd = sqrt(sig_sq))
(avg_x <- mean(x))
#> [1] 0.45
write(avg_x, "avg_x.txt")
plot(x, y)
abline(a, b, col = "purple")
```



```
dev.print(pdf, "toy_line_plot.pdf")
#> pdf
#> 2
```

```
source(file.r)
```

In your favorite OS-specific way, search your files for `toy_line_plot.pdf` and presumably you will find the PDF itself (no surprise) but *also the script that created it* (`toy-line.r`). This latter phenomenon is a huge win. One day you will want to remake a figure or just simply understand where it came from. If you rigorously save figures to file **with R code and not ever ever ever the mouse or the clipboard**, you will sing my praises one day. Trust me.

### 1.2.5 stuff

It is traditional to save R scripts with a `.R` or `.r` suffix.

Comments symbol : `#`

de(comment) : `Ctrl+Shift+C`

This workflow will serve you well in the future:



- Create an RStudio project for an analytical project
- Keep inputs there
- Keep scripts there; edit them, run them in bits or as a whole from there
- Keep outputs there (like the PDF written above)

Many long-time users never save the workspace, never save `.RData` files (I'm one of them), never save or consult the history.

Option to disable the loading of `.RData` and permanently suppress the prompt on exit to save the workspace (go to Tools->Options->General).



## Chapter 2

# R Markdown test drive

### 2.0.0.1 Overview

Here is the official R Markdown documentation: <http://rmarkdown.rstudio.com>

### 2.0.0.2 Step 0: Software installation and configuration

### 2.0.0.3 Step 1: Get ready to work

### 2.0.0.4 Step 2: Practice with RStudio's boilerplate R Markdown document

Do this: *File > New File > R Markdown ...*

- Give it an informative title.
- Accept the default Author or edit if you wish.
- Accept the default output format of HTML.
- Click OK.

Click on “Knit HTML” or do *File > Knit Document*. RStudio should display a preview of the resulting HTML.

### 2.0.0.5 Step 3: Take control of the output format

The magical process that turns your R Markdown to HTML is like so: `foo.Rmd --> foo.md --> foo.html`.

#### Output format

html:

```
---
title: "Something fascinating"
author: "Jenny Bryan"
date: "2018-07-07"
output:
  html_document:
    keep_md: true
---
```

GitHub-flavored markdown:

```
---
title: "Something fascinating"
author: "Jenny Bryan"
date: "2018-07-07"
output: github_document
---
```

#### 2.0.0.6 Step 4: Swap out the “guts” of the document

Insert an empty R chunk:

```
```r
## insert your brilliant WORKING code here
```
```

#### 2.0.0.7 Step 5: Develop your report

#### 2.0.0.8 Step 6: Publish your report

Creating, committing, and pushing markdown is a very functional, lightweight publishing strategy.

This is (sort of) another example of keeping things machine- and human-readable, which is bliss. By making `foo.Rmd` available, others can see and run your **actual code**. By sharing `foo.md` and/or `foo.html`, others can casually browse your end product and decide if they even want to bother.

#### 2.0.0.9 Troubleshooting

Make sure RStudio and the `rmarkdown` package (and its dependencies) are up-to-date.

Get rid of your `.Rprofile` at least temporarily.

Insert a chunk in your `.Rmd` document so that it renders even when there are errors.

Tolerate errors in one specific chunk.

```
```r
## your sketchy code goes here ;)
```
```

Check your working directory.

- `getwd()`
- `list.files()`

Don't try to change working directory within an R Markdown document.

Don't be in a hurry to create a complicated sub-directory structure.

### 2.0.1 HTML on GitHub

I have more general ideas about how to make a GitHub repo function as a website.

### 2.0.1.1 Be savvy about your files

Keep files in the plainest, web-friendliest form that is compatible with your main goals. Plain text is the very best. GitHub offers special handling for certain types of files:

- Markdown files, which may be destined for conversion into, e.g., HTML
- Markdown files named `README.md`
- HTML files, often the result of compiling Markdown files
- Source code, such as `.R` files
- Delimited files, containing data one might bring into R via `read.table()`
- PNG files

### 2.0.1.2 Get over your hang ups re: committing derived products

Let's acknowledge the discomfort some people feel about putting derived products under version control.

The taboo of keeping derived products under version control originates from compilation of binary executables from source. Software built on a Mac would not work on Windows and so it made sense to keep these binaries out of the holy source code repository.

### 2.0.1.3 Markdown

**Keep intermediate Markdown.** Commit both `foo.Rmd` and `foo.md`, even if you choose to `.gitignore` the final `foo.html`. As of September 2014, GitHub renders R Markdown files nicely, like Markdown, and with proper syntax highlighting, which is great. But, of course, the code blocks just sit there un-executed, so my advice about keeping intermediate Markdown still holds. You want YAML frontmatter that looks something like this for `.Rmd`:

```
---
title: "Something fascinating"
author: "Jenny Bryan"
date: "`r format(Sys.Date())`"
output:
  html_document:
    keep_md: TRUE
---
```

or like this for `.R`:

```
#' ---
#' title: "Something fascinating"
#' author: "Jenny Bryan"
#' date: "`r format(Sys.Date())`"
#' output:
#'   html_document:
#'     keep_md: TRUE
#' ---
```

### 2.0.1.4 README.md

`README.md` at the top-level of your repo as the *de facto* landing page. This is analogous to `index.html`.

Some repositories consist solely of `README.md`.

- Jeff Leek's write-ups on How to share data with a statistician
- Developing R packages.

README-only repos vs gists :

- PNGs README gallery