

# Rmarkdown

*Simeon Markind*

*23 March, 2017*

## Setup

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>. Cheatsheet can be found <http://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf> Reference guide can be found <https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>

Today we will talk about pdf Rmarkdown documents. Your final project is required to be done in Rmarkdown. You will want to use the pdf\_document type for your write-up. Your slides for the presentation should be pdf, of type beamer.

For today's lecture you will need to open up an Rmarkdown document. Go to File -> New File -> R Markdown and create an PDF document with your name. Title it "R Markdown Practice."

At the top of the Rmd script you should see a section that looks like the following:

```
—
title: "Rmarkdown"
author: "Simeon Markind"
date: '23 March, 2017'
output: html_document
—
```

This is the YAML section of the document. We will discuss it later in this lecture. Your date argument should look different, we will talk about what my date argument does at the end of the lecture.

Below this YAML section you should have the basic text that Rstudio puts in to all new Rmarkdown documents and an R chunk that calls the `knitr::opts_chunk$set()` command. We will also talk about this later but for now you can simply ignore all the text that came with the document. Before making any changes make sure that the pdf will compile and click on the Knit button at the top of your editor window. You should get a pdf document to pop up onto your screen. Let one of the TAs know if this does not work for you.

Rmarkdown provides a ready made environment in which you can seamlessly embed code, text, and graphics. Although Rmarkdown comes with Rstudio and by default runs R code, you can also run many other languages in a markdown document. To see which engines are available to you type `names(knitr::knit_engines$get())` and you should get a list similar to the following: awk, bash, coffee, gawk, groovy, haskell, lein, mysql, node, octave, perl, psql, python, Rscript, ruby, sas, scala, sed, sh, stata, zsh, highlight, Repp, tikz, dot, c, fortran, fortran95, asy, cat, asis, stan, block, block2, js, css, sql

## Basics

An Rmarkdown document has two modes, we will refer to them as "text" and "code." Text mode is what you are reading now. Default settings for text mode are black text on a white background. (We will discuss how to change this for html later)

In the text mode of a markdown document you can type text as you would in a word document and the output document simply returns exactly what you put in, typos and all. Please refer to the reference guide linked above to see more options for what you can do in this part of Rmarkdown.

## Text Mode

- Formatting Summary
  - Two spaces at the end of a line indicate a new line.
  - Pound symbols, #, at the beginning of a line indicate headers.
    - \* You can use up to six # signs, for each additional # sign used the text becomes smaller.
  - Surrounding your text in \* or \_ makes italics.
    - \* i.e. \*italics\* gives *italics* and \_italics\_ does as well
  - \*\* will bold your text as does \_\_
    - \* \*\*bold\*\* and \_\_bold\_\_ gives **bold**
  - The ^ creates superscripts so x^2^ gives x<sup>2</sup>
  - The ~ gives strikethrough
    - \* ~I have never used this~ ~~but maybe you will~~
  - To insert links use the following syntax [text] (webaddress)
    - \* To learn more go here
  - Inline equations use Latex math mode, the \$ sign.
    - \* \$y = \pi\*r^{2}\$
      - gives  $y = \pi * r^2$
    - \* If you have used LaTeX before you will be able to use many of the latex typesetting commands and functions in markdown.
  - The backslash is the escape character and allows you to show \\*text\\* that would otherwise involve *formatting*.
  - You can insert tables by hand as shown on the reference guide but I would recommend against that.
  - Bullets are made with a single \* sign followed by a space
    - \* sub-bullets are made by indenting directly below the preceding text and using a + symbol
  - Ordered lists follow the same format but instead of using \* as the symbol use a number.
    - \* Note that you can sub-number a numbered list
- 1. Ordered list
  - 1. first sub-bullet
    - 1a. sub-sub-bullet

In your Rmarkdown file add the following lines, be sure to format them so they are identical when you knit the document.

## This is the largest header size

I have really enjoyed this course on ~~python~~, sorry, R  
 $5^2 + 4 = 29$

- I like the following:
  - lists
    - \* sub-bullets
- 1. numbered lists are good too
- 2. They make it easy to organize things.

## Some LaTeX Basics

L<sup>A</sup>T<sub>E</sub>X is a powerful typesetting language used throughout academia and the professional world to easily display mathematical functions, graphics, images, etc... alongside text. Rmarkdown is built in part on

L<sup>A</sup>T<sub>E</sub>X which is why many L<sup>A</sup>T<sub>E</sub>X commands are also Rmarkdown text commands. The formatting above are L<sup>A</sup>T<sub>E</sub>X commands. One of the most important features of L<sup>A</sup>T<sub>E</sub>X for our purposes will be the use of math mode. This mode allows us to add equations both inline and for display in our document. For your final project you should use these modes to explain the models used for your analysis.

## Inline Equations

Inline equations occur within the text smoothly. For example if I wanted to tell you that  $y = \sum_{i=1}^n mx_i^2 + b$  in the middle of a sentence I could do that. This is not an equation that has any meaning to us but is a good display. Inline math mode is accessed using a single \$ sign surrounding each end of the equation. So, for my code above I typed: `$y = \sum_{i=1}^n mx_i^2 + b$`. The `\sum` gives the sigma character and the `x_i^2` returns the  $x_i^2$  symbol.

- `\sum` is a built-in symbol to LaTeX. And gives  $\sum$ 
  - There are many others such as `\infty` which gives  $\infty$
- The single `_` character allows for subscripts as we saw above. So `$\sum_1$` gives us  $\sum_1$ . (I will no longer explicitly show the \$ symbols in the explanation but remember that you need them to display math symbols inline.)
- The `^` sign gives superscript and the brackets take the numbers to display in the superscript. So `x^2` gives  $x^2$ 
  - The brackets can accept multiple numbers, so you could theoretically type `x^{2,3}` which gives  $x^{2,3}$  which makes no sense.
  - However, this ability is important as we will see in a minute with subscripting. To show element A in row i and column j in a matrix you would type `A_{i,j}` and show  $A_{i,j}$

Putting this all together we can show another example of inline math.  $\int_0^1 x^2 dx = \frac{x^2}{2} \Big|_0^1 = \frac{1}{2} - 0 = \frac{1}{2}$ . The code for which is: `$\int_0^1 x^2 dx = \frac{x^2}{2} \Big|_0^1 = \frac{1}{2} - 0 = \frac{1}{2}$`. Here the `|` character (pipe) gives us the evaluation bar for x once we have evaluated it. To see more examples of using different latex functions check out the webpage: <http://www.calvin.edu/~rpruim/courses/m343/F12/RStudio/LatexExamples.html>. If you do not find the math symbols you are looking for there, try this CU webpage [http://www.colorado.edu/physics/phys4610/phys4610\\_sp15/PHYS4610\\_sp15/Home\\_files/LaTeXSymbols.pdf](http://www.colorado.edu/physics/phys4610/phys4610_sp15/PHYS4610_sp15/Home_files/LaTeXSymbols.pdf). If that fails ask google.

Note that Latex symbols follow a basic pattern when defining them of `\function{input}` with the braces delineating inputs. For example  $\cos(x^2)$  is `$\cos\{x^2\}$`

## Display Equations

In addition to inline math mode, equations in Latex can also be shown in display mode. In this mode the expressions are centered and in a separate line from the main text.

$$\sum_{i=1}^n X_i$$

. This mode is activated by using two \$ signs surrounding the formula in place of one.

**Ok, now that you have seen how the math modes work how would you write the following in display mode? Write the following equation and knit your document. (Google the LaTeX symbol for a limit if you need)**

$$\lim_{x \rightarrow \infty} \frac{1}{x} \rightarrow 0$$

## Code Mode

Embedded R code is where the power of Rmarkdown really comes through. Up until now we have not really seen the difference between a markdown file and regular Latex. The ability to embed R code, code that can be evaluated in-line with the text, is what makes Rmarkdown so powerful. Similar to text, code chunks can be added either inline or chunk.

### Inline Code

Code made is activated via the ``` character. This is the character at the top left of the number row of your keyboard, on the same key as the `~`. A single ``` followed by text and closed by another ``` creates text on a grey background in html mode, and bolded text with a different font in pdf mode as we see here. This is useful for displaying function names such as `sum()` in a way that readers can understand to be code. Inline code can also be evaluated by adding a marker denoting the language to run immediately following the ``` mark. I.e. `` r sum(c(3,4,5))`` evaluates to 12. We could specify other engines instead of R if desired.

### Code Chunks

While inline code mode is useful, the ability to embed R scripts, essentially in their entirety allows Rmarkdown users to seamlessly use R code and text to create presentations and documents. To create a new code chunk press `Ctrl+Alt+I` or type in three ``` and specify that you want the R engine in braces like on a new line like so:

```
“`{r}
```

All options for the code chunk go between the braces. If you inserted the code chunk with `Ctrl+Alt+I` it will already have a closing in it. If you started the code chunk manually you must close it with three more ``` on an empty line. So your code chunk should look like the following:

```
“` {r options}  
Code  
more code  
“`
```

Within the code chunk you can write R code like you are used to in an R script file. Within the code chunk, the `#` symbol will revert to being the comment symbol as opposed to the header symbol and you can use R functions like normal. The R code chunks, when run, impact the global environment. This means that a table generated in chunk 1 can be accessed later in the script, say in chunk 5, each chunk is not an isolated environment, so changes you make in an early chunk can have an impact “downstream” in your code. When the Rmarkdown file is compiled when you knit, the code chunks are evaluated sequentially so often when you throw an error it is because you changed something upstream that impacted code downstream. It is possible to stop a chunk from being evaluated by changing the chunk options. Before we discuss knitr chunk options, let's get a feel for how the code works.

### Basic R chunk

For this lecture we will explore the economics dataset include with the `ggplot2` package. You can name your chunk with a text string immediately after the engine setting for the chunk. I.e.

```
“`{r name}
```

You do not have to put quotes around your chunk name but you can if desired.

```
“`{r “name”}
```

 is also valid.

Before we continue make sure you have the following libraries:

```
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Conflicts with tidy packages -----

## between():    dplyr, tis, data.table
## changes():    dplyr, frb
## filter():     dplyr, stats
## first():      dplyr, data.table
## groups():     dplyr, frb
## lag():        dplyr, stats
## last():       dplyr, data.table
## rename():     dplyr, frb
## tbl():        dplyr, frb
## transpose():  purrr, data.table
```

```
library(knitr)
```

In your Rmarkdown document create a code chunk and copy in the following code, then knit your document.

```
## First we should take a look at our data
head(economics)
```

```
## # A tibble: 6 x 6
##       date    pce    pop psavert uempmed unemploy
##   <date> <dbl> <int>   <dbl>   <dbl>   <int>
## 1 1967-07-01 507.4 198712    12.5     4.5    2944
## 2 1967-08-01 510.5 198911    12.5     4.7    2945
## 3 1967-09-01 516.3 199113    11.7     4.6    2958
## 4 1967-10-01 512.9 199311    12.5     4.9    3143
## 5 1967-11-01 518.1 199498    12.5     4.7    3066
## 6 1967-12-01 525.8 199657    12.1     4.8    3018
```

```
## Now print out the number of rows in the table
nrow(economics)
```

```
## [1] 574
```

So as you've probably noticed from reading the lecture handouts this semester, code chunks come out in the documents with a grey background. Evaluated code comes out in the document in a box with a white background.

## Changing Chunk Options

In addition to being able to name your code chunk you can also pass in knitr options. These options affect whether the code is evaluated or not, whether the code itself is shown in the output document as well as many other controls surrounding the output. Specifying chunk options is done after a comma within the braces of your chunk declaration like so:

```
““{r chunk name, option1 = TRUE, option2 = FALSE, etc}
Code More Code ““
```

Chunk options should be comma separated. Here are some that I frequently use:

1. echo - controls display of code
  - If echo = TRUE the code in the code chunk will be displayed
  - If echo = FALSE the code in the chunk will not be displayed (and the code will still evaluate)
  - Similar to Eval can take a vector argument as well.
2. error - controls error messages
  - If error = TRUE error messages are displayed and code continues to run
  - If error = FALSE error messages are not displayed, code does not continue to run
3. Include
  - If include = TRUE (default) the chunk will be run and included in the final document
  - If include = FALSE the chunk will be run but not included in the final document
    - E.g. plots/tables generated in the chunk are still created and accessible in other chunks
4. Eval
  - If eval = FALSE knitr will not run the code in the code chunk
  - If eval = TRUE (default) the code will be run
  - This is useful for debugging as you can turn eval to FALSE in a chunk and play around with your code but not raise issues compiling your document.
  - This can also take a vector to select which expression(s) to evaluate. e.g. eval = c(2,4,5) or eval = -(4:5)
5. Comment
  - Takes an optional character argument of what to display next to R output. (i.e. comment = “Output >”)

A list of Chunk options can be found in the Rmarkdown reference materials. A full list of chunk options can be found at: [http://yihui.name/knitr/options/#chunk\\_options](http://yihui.name/knitr/options/#chunk_options)

**Look at this webpage and/or the reference materials and create a code block with the following attributes:**

- Output is preceded by the string “Rout >”

**In your code return the subset of the economics dataset with unemployment above 10%**

```
Rout > # A tibble: 101 x 6
Rout >       date    pce    pop psavert uempmed unemploy
Rout >   <date> <dbl> <int>   <dbl>   <dbl>   <int>
Rout > 1 1982-11-01 2154.7 232993    10.3    10.0    11938
Rout > 2 1982-12-01 2167.4 233160    10.3    10.2    12051
Rout > 3 1983-01-01 2180.1 233322    10.4    11.1    11534
Rout > 4 1983-03-01 2208.6 233613    10.0    10.4    11408
Rout > 5 1983-04-01 2231.8 233781     9.6    10.9    11268
Rout > 6 1983-05-01 2251.0 233922     9.4    12.3    11154
Rout > 7 1983-06-01 2280.8 234118     8.5    11.3    11246
Rout > 8 1983-07-01 2309.0 234307     9.0    10.1    10548
Rout > 9 1994-10-01 4833.3 264301     6.5    10.0     7632
Rout > 10 2002-06-01 7366.1 287864     5.4    11.0     8393
Rout > # ... with 91 more rows
```

Chunk options can also be set at the top of the markdown document using the `knitr::opts_chunk$set()` function. This function can be called anywhere in the document but it is a good idea to set it at the beginning since changes set will only affect code downstream. For example, if we wanted to set the default for the echo value to TRUE so that by default all of our code block would be rendered onto the final document and include to be true so that our code would always be run we could do the following:

```
knitr::opts_chunk$set(echo = TRUE, include = TRUE)
```

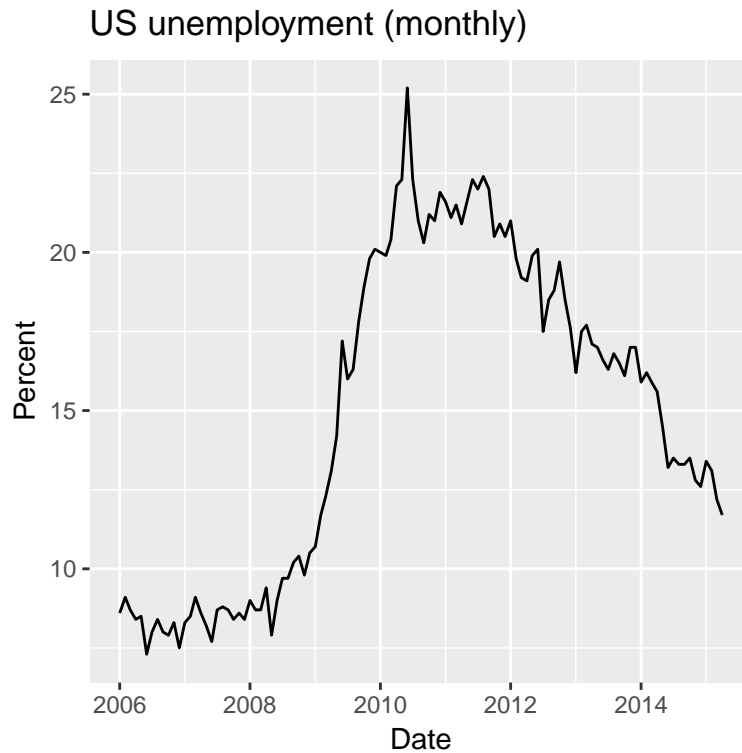


Figure 1: Example caption

## Calling Graphs

Just as we were able to show table output inline with our text above, we can also include graphics in our final output as well. For ggplot objects this simply requires creating the objects and then calling them. Additionally, we can set chunk options either specifically or globally that change the size of our output plots.

```
plot.data <- economics %>%
  filter(date >= as.Date("2006-01-01")) %>%
  select(date, uempmed)

plot1 <- ggplot(plot.data, aes(x = date, y = uempmed)) +
  geom_line() +
  labs(title = "US unemployment (monthly)",
       x = "Date",
       y = "Percent")

plot1
```

You may notice that this figure is smaller than the default, centered, and has a caption underneath.

**Create a plot, called plot2 of the unemploy variable over time from 1996-01-01 to the end of the data with the following attributes**

- Figure height is 4 inches
- Width is 8 inches
- Alignment is right
- Caption reads "Source: Economics dataset"

**Your plot should look like this:**

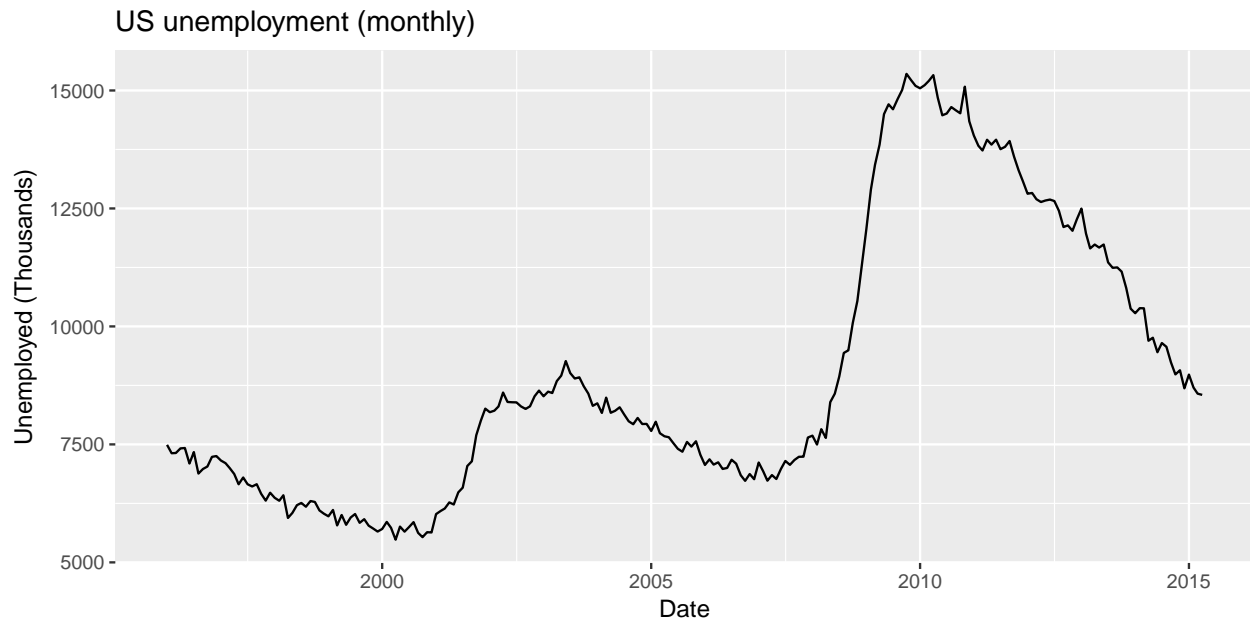


Figure 2: Source: Economics Dataset

## Inserting tables

Just like we can insert graphs into the markdown document so too can we insert tables directly in line with the text. The knitr package comes with the `kable` function that we can use to call a table that is more formatted than standard R output. The `kable` function can take multiple styles if you want your table to have a different look.

```
kable(head(economics, 5), "pandoc", col.names = c("Date", "Pce", "Pop", "Psavert", "uempmed", "unemploy
```

Date	Pce	Pop	Psavert	uempmed	unemploy
1967-07-01	507.4	198712	12.5	4.5	2944
1967-08-01	510.5	198911	12.5	4.7	2945
1967-09-01	516.3	199113	11.7	4.6	2958
1967-10-01	512.9	199311	12.5	4.9	3143
1967-11-01	518.1	199498	12.5	4.7	3066

## Beamer Slides

Most pdf files that you have seen so far have been full length documents, books articles, etc. However, there is also a pdf format for making slideshows, called Beamer. These slides produce similar output to a powerpoint presentation but unfortunately don't come with the fancy Microsoft interface for creating them.

To start a Beamer slide presentation go to File -> New File -> RMarkdown -> Presentation -> PDF (Beamer)

The syntax for making a slideshow is the same as we have covered in this lecture where we made a longer-style document. The main difference is that when making a slideshow, instead of creating a large header, using a single `#` will insert a new slide with whatever title you decide to give it.

## Changing the YAML

Up at the top of the document I showed you that my YAML header has the following command for date: `" r format(Sys.time(), "%d %B, %Y")"` instead of the actual date as you does when you start a new document. As a result the date generated when I knit my document is of the format "day month, year." and this value updates every day because every time I knit my document the YAML makes a call to the `Sys.time()` function and gets today's value. While I could have put any valid R command as the date, e.g. `"date: ' r sum(3,4)'"` and gotten a date of 7 every time I knit, it is important that I quoted the entire command in the YAML



Now go back to your YAML header and make the following changes:

output: html\_document

should now become:

output:

html\_document:

css: style.css

The “html\_document” should be indented 2 spaces beyond “output” and the “css: style.css” should be indented two more spaces than “html\_document.” Make sure that the — ending your YAML header is below “css: style.css”.

Now knit your document, your text should now be red and size 14.

Understanding how css manipulates your html output is important if you want to fine tune your Rmarkdown documents but is beyond the scope of this lecture. To learn more about css files I recommend checking out w3schools: [http://www.w3schools.com/html/html\\_css.asp](http://www.w3schools.com/html/html_css.asp) and looking at their css tutorial. If you have any interest in becoming a web developer this topic will be important.

For your final project as a pdf rather than html, you will not be able to change your styling via css file and will instead need to learn how to change the styling via  $\text{\LaTeX}$  and pandoc if you so desire.

Good luck with your final project. Myself and the rest of the Instructors are looking forward to your presentations next week.