RMarkdown Demonstration

Saewon Park

November 21, 2018

R Markdown is a powerful tool that you can use to type up presentable results in R. There are two parts to R Markdown - the R script and text. We will first go over the text part to understand how it works, and then we will cover how to incorporate R script.

To start R Markdown, you need to start a R Markdown script. To do so, click File, New File, R Markdown. You will get a new window asking to name your document and what type of document you want.

First, some general formatting. When you use the hash-tag symbol (#), it creates a new section header:

Section 1

Notice the colour of the text change—this means that R Markdown recognized this as a section header

If you use two hash-tags, you can create a sub-heading.

Sub-Section 1.1

When you click the "Knit" button a document will be generated that includes both written content as well as the output of any embedded R code chunks within the document. You can choose the format at this point as well (HTML, PDF, Word).

Sub-Sub-Section 1.1.1

Here are some ways to format the text section:

This is how you make **bold** text

This is how you make *italic* text

If you want a slide break (horizontal line) do this:

- bullet point
  - subitem (need to put 4 spaces)

1. numbered
2. list

We can add images too:

Problem 1

1. Open a new RMarkdown file. Remove the introductory text that is there when you open the new file.
2. Add in a section called "Cities" (you can put some generic text under this heading if you want) and then two sub-section headings called "Three cities I've been to" and "Three cities I want to visit"

Figure 1: Selfie Cat

3. Under "Three cities I've been to," list three cities you've been to in bullet point format put the most recent city you visited in *italics*. Under "Three cities I want to visit" list three cities you want to visit as a numbered list and put the city you want to visit the most in **bold**.
4. Knit the file to HTML.

Bonus: If you have time, you can try to download any picture and save it into the same folder as the rmd file and add it to the document.

Answer to Problem 1

Cities

I have two lists of cities.

Three cities I've been to

- Boston
- *Seoul*
- Nairobi

Three cities I want to visit

1. San Francisco
2. Kisumu
3. **Rome**

Using R Code in RMarkdown

If you look down at the console of your screen, you see that you have two tabs: Console and R Markdown. The console section is just like what you have with a regular R Script. The R Markdown part is what knits the document.Some people like to just work in R Markdown, other people like to work with a R Script first

Figure 2: Selfie Cat

and then convert it to R Markdown. It just depends what your preferences are. One thing: everything that you include in an R Script, you have to include in a R Markdown file (otherwise it will not knit)! This means your working directory, packages, etc. all have to be in the R Markdown file.

Setting up your RMarkdown file (in the setup code chunk)

In this first code chunk, I load in the data and the packages I will use in the rest of the document. You'll notice that the code chunk below will not show up when we knit the file. This is because we have specified `include = FALSE`. This tells R Markdown to (a) run the code in the code chunk, but (b) don't show the code chunk itself in the knitted document. That is, it tells R Markdown to run the code quietly.

Running code in R code chunks

Now that I have the `cw` dataset and `dplyr`/`ggplot2` packages loaded, I can conduct some analysis. Let's start a new code chunk where we will run some commands. To add in new code chunks, there is a button with a "C" and a plus sign in the top menu. You can also do this manually. Notice how in the RMarkdown file the chunk has a different shading.

Let's look at some simple descriptive statistics:

```
summary(cw$polity2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## -10.000  -7.000  -3.000  -0.413   8.000  10.000      63
```

Notice that I can still run code down in the console while also writing the text. You can also run all of the code by pressing the green arrow at the top right hand corner of the chunk, or you can highlight it and run it that way.

Problem 2

1. Continuing with the RMarkdown file you created in Problem 1, load in your data and the dplyr package.
2. Create a new code chunk after the text you created earlier.
3. In the code chunk, use dplyr's filter function to separate only cases of countries at war. Then, calculate the mean GDP per capita for those countries.

4. Knit the file to see what the output looks like and make sure your calculation worked correctly. Note that you can always check your work before knitting by running the code in the console.

Answer to Problem 2

This would be in their first chunk

```
war <- cw %>%
  filter(war == 1) #filter only countries at war

mean(war$gdppc, na.rm = T) #mean of the GDPpc of countries at war
```

```
## [1] 1.845964
```

Displaying/Hiding R Code

There are a number of options for how R Markdown displays and evaluates code chunks, here are the three most useful:

- `echo = FALSE`: display the output from the R code, while hiding the actual code itself
- `eval = FALSE`: display the code but do not run it, no output is created because nothing has been evaluated
- `include = FALSE`: evaluate the code, but don't display it or the output it creates

The easier way is generally just to click the gear in the code box and select the option you want. This automatically changes the required code in the `{r}` options.

Compare the three code chunks below after knitting.

Example 1 (`echo = TRUE`, the default):

```
cw_alt <- cw %>%
  filter(war == 1) %>%
  mutate(log_gdppc = log(gdppc),
         log_pop = log(pop))

mean(cw_alt$log_gdppc, na.rm = TRUE)
```

```
## [1] 0.2024307
```

Example 2 (`echo = FALSE`):

```
## [1] 0.2024307
```

Example 2 (`eval = FALSE`):

```
cw_alt <- cw %>%
  filter(war == 1) %>%
  mutate(log_gdppc = log(gdppc),
         log_pop = log(pop))

mean(cw_alt$log_gdppc, na.rm = TRUE)
```

Hiding the R code is useful when you add plots, since you may just want to show a plot and not the code that was used to make it.

Example 1 (`echo = TRUE`):

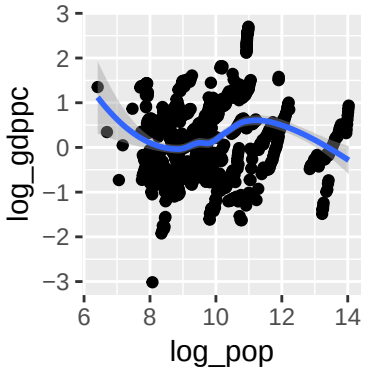
```
#quick review of ggplot
cwplot <- ggplot(data = cw_alt, mapping = aes(log_pop, log_gdppc)) #set the mapping parameters for our

cwplot +
  geom_point() + #tell it how we want to plot the data - here we want a scatter plot
  geom_smooth() #we can add different plotting methods (with the same axes) to one plot - here we want
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

```
## Warning: Removed 59 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 59 rows containing missing values (geom_point).
```

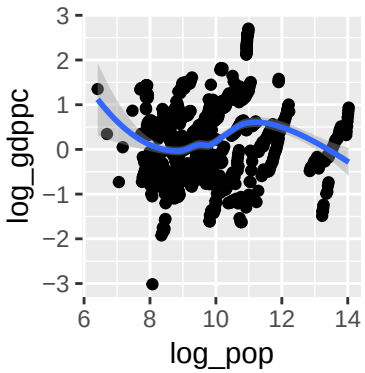


Example 2 (`echo = FALSE`):

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

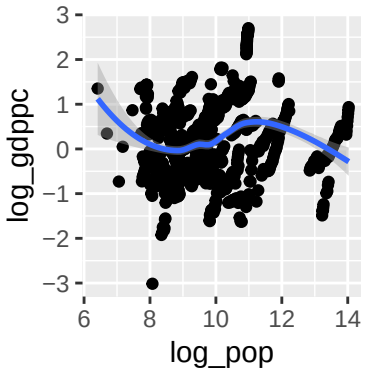
```
## Warning: Removed 59 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 59 rows containing missing values (geom_point).
```



I can also use these options to suppress warning and other messages:

Example 3 (`echo = FALSE, message = FALSE, warning = FALSE`):



Finally, it is possible to set these options globally so that the default for each code chunk is to show or not show the code in the knitted file. To do so, you need to edit the *first* code chunk in your document - either using the gear options or by specifying the necessary arguments (e.g. `echo=FALSE`) within that chunk.

We need to use this in the first chunk:

```
#knitr::opts_chunk$set(echo = FALSE)
```

Problem 3

1. Copy the code chunk you created in Problem 2.
2. In the duplicate code chunk, edit the display options to only display the code output.
3. In another the duplicate code chunk, edit the display options to only display the code without running it. Knit the file and compare how RMarkdown treats each code chunk.
4. Now, change the global options for the file in your first code chunk to only display the code output and re-knit the file. How does this output compare?

Answer for Problem 3

```
library(dplyr)

cw <- read.csv("fearon03.csv")

war <- cw %>%
  filter(war == 1) #filter only countries at war

mean(war$gdppc, na.rm = T) #mean of the GDPpc of countries at war
```

```
## [1] 1.845964
```

```
## [1] 1.845964
```

```
library(dplyr)

cw <- read.csv("fearon03.csv")

war <- cw %>%
```

```
  filter(war == 1) #filter only countries at war

mean(war$gdppc, na.rm = T) #mean of the GDPpc of countries at war
```

Looping

To perform the same function on multiple elements (e.g. vectors, dataframes, lists), we may want to iterate with a for loop. We can think of iteration most simply when we think of going through a list of elements and doing something to each of the elements. We can iterate through any set of elements.

The syntax starts with the word `for`, followed by parentheses `(iterate_through_something)`, and then a curly brace where you can put the commands you want to iterate through. A curly brace also closes the process.

Within the parentheses, you need an *index variable*, often called i (though it is up to you what the index variable is called). This is the thing that is going to change every time you go through the for loop. The second thing is what you are iterating over.

Note that the use of the letter i is arbitrary. We can call i anything. And each time we go through the loop, i takes on a different value.

Let's take a look at a simple `for` loop. Let's just iterate through the numbers 1 through 10 and print out the cube of each number.

```
n <- 10
print(1:n)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
for(i in 1:n) {
  cat("i=", i, "| ") #print to the console \n is a line break
  the_cube <- i^3 #each time we assign a new value to the_square
  cat(i, "cubed is:", the_cube, "\n")
}
```

```
## i= 1 | 1 cubed is: 1
## i= 2 | 2 cubed is: 8
## i= 3 | 3 cubed is: 27
## i= 4 | 4 cubed is: 64
## i= 5 | 5 cubed is: 125
## i= 6 | 6 cubed is: 216
## i= 7 | 7 cubed is: 343
## i= 8 | 8 cubed is: 512
## i= 9 | 9 cubed is: 729
## i= 10 | 10 cubed is: 1000
```

This is the same as looping through an *index variable* named *foo*. We just use i in social statistics because it often makes sense. We can see in the example below that we get exactly the same result if we use *foo*.

```
n <- 10
for(foo in 1:n) {
  cat("i=", foo, "| ") #\n is a line break
  cubed <- foo^3 #each time we assign a new value to squared
  cat(foo, "cubed is:", cubed, "\n")
}
```

```
## i= 1 | 1 cubed is: 1
## i= 2 | 2 cubed is: 8
## i= 3 | 3 cubed is: 27
## i= 4 | 4 cubed is: 64
## i= 5 | 5 cubed is: 125
## i= 6 | 6 cubed is: 216
## i= 7 | 7 cubed is: 343
## i= 8 | 8 cubed is: 512
## i= 9 | 9 cubed is: 729
## i= 10 | 10 cubed is: 1000
```

One more example:

```
year_born <- 1994
current_year <- 2018

for(i in year_born:current_year) {
  age = i - year_born
  cat("In", i, "I was", age, "years old \n")
}
```

```
## In 1994 I was 0 years old
## In 1995 I was 1 years old
## In 1996 I was 2 years old
## In 1997 I was 3 years old
## In 1998 I was 4 years old
## In 1999 I was 5 years old
## In 2000 I was 6 years old
## In 2001 I was 7 years old
## In 2002 I was 8 years old
## In 2003 I was 9 years old
## In 2004 I was 10 years old
## In 2005 I was 11 years old
## In 2006 I was 12 years old
## In 2007 I was 13 years old
## In 2008 I was 14 years old
## In 2009 I was 15 years old
## In 2010 I was 16 years old
## In 2011 I was 17 years old
## In 2012 I was 18 years old
## In 2013 I was 19 years old
## In 2014 I was 20 years old
## In 2015 I was 21 years old
## In 2016 I was 22 years old
## In 2017 I was 23 years old
## In 2018 I was 24 years old
```

Problem 4

1. Create a new code chunk. Create a variable called year_first_grade and assign it the year you entered first grade and also create a variable called year_last_grade and assign the year your graduated from highschool to it.

2. Using a `for` loop, write a code that will output a sentence like "I was in grade X in year X" for each year you were in school.

Answer to Problem 4

```
year_first_grade <- 2001
year_last_grade <- 2012

for(i in year_first_grade:year_last_grade){
  grade = i - year_first_grade + 1
  cat("I was in grade", grade, "in", i, "\n")
}
```

```
## I was in grade 1 in 2001
## I was in grade 2 in 2002
## I was in grade 3 in 2003
## I was in grade 4 in 2004
## I was in grade 5 in 2005
## I was in grade 6 in 2006
## I was in grade 7 in 2007
## I was in grade 8 in 2008
## I was in grade 9 in 2009
## I was in grade 10 in 2010
## I was in grade 11 in 2011
## I was in grade 12 in 2012
```