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

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 and

Figure 2: Selfie Cat

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.

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

```
## [1] "banana"
```

To add in new code chunks, there is a button with a "C" in the top menu. Notice how in the RMarkdown file the chunk has a different shading.

I can also use code chunks to perform R functions:

```
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

```
library(dplyr)

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

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

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

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

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

Displaying/Hiding R Code

It is also possible only to display the output from the R code, while hiding the actual code itself. To do this, you can enter "echo=FALSE" within the {r} brackets at the start of a code chunk (i.e. {r, echo=FALSE}).

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 two code chunks below after knitting.

Example 1:

```
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:

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

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.

```
#install.packages("ggplot2") #ggplot2 is a commonly used package for making plots
library(ggplot2)
```

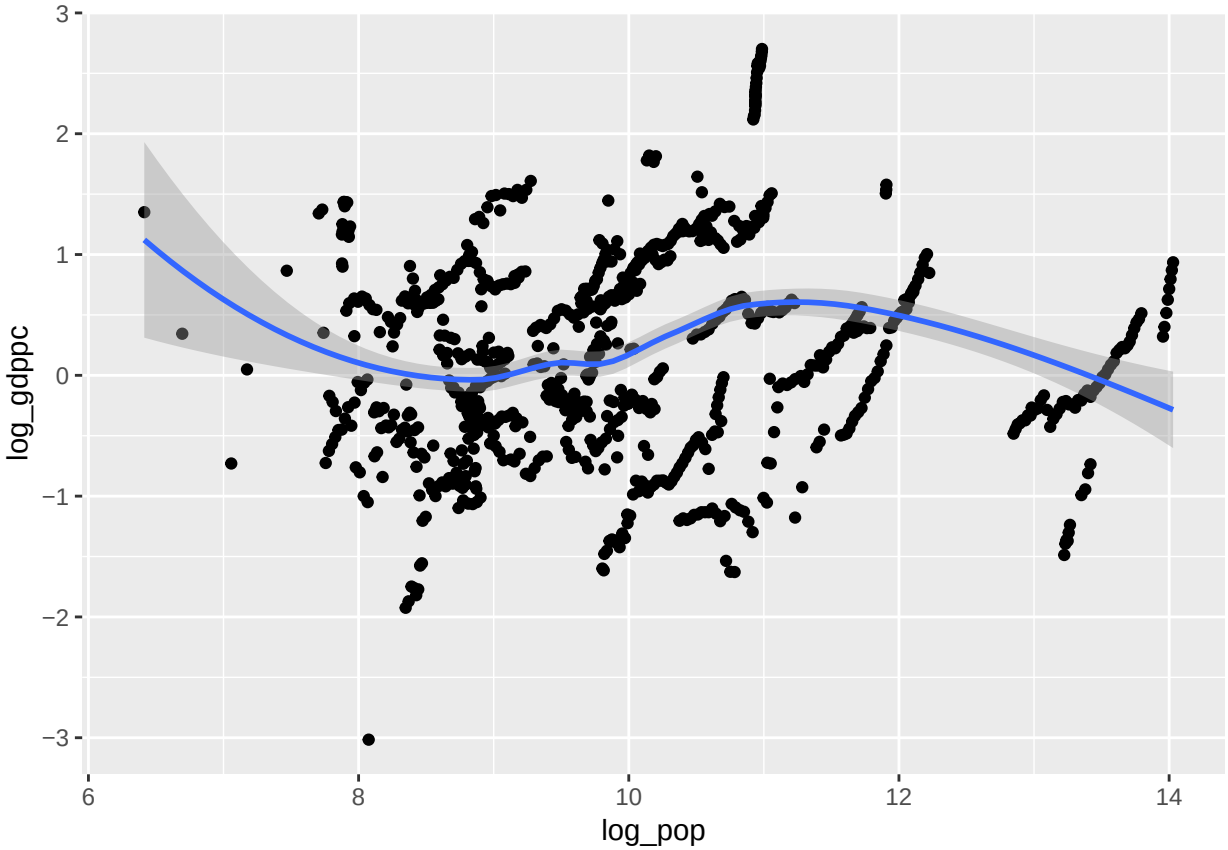
Example 1:

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



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'
```

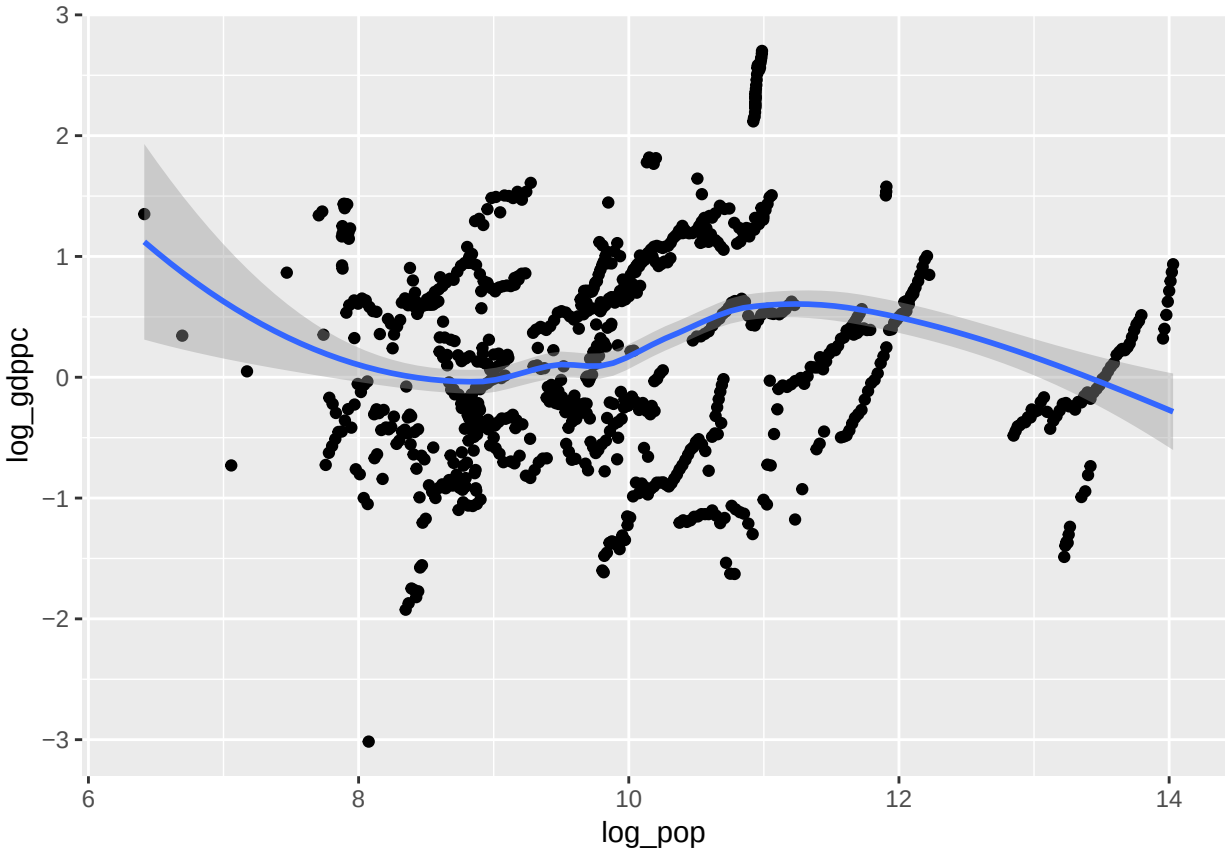
```
## Warning: Removed 59 rows containing non-finite values (stat_smooth).
```

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



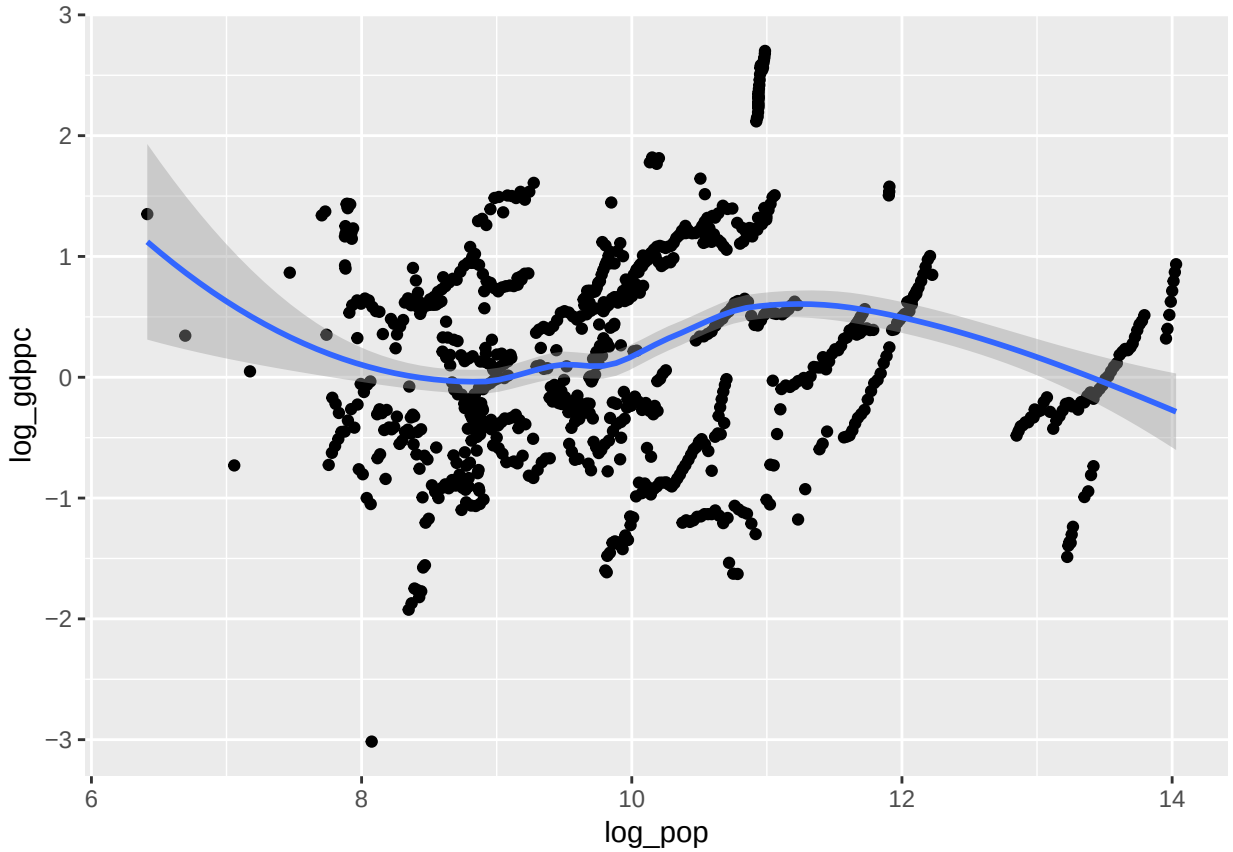
Example 2:

```
## `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:



On the other hand, if I just want to show the code and not run it, I can use the eval= FALSE option

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

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

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.

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

Often you want to go through each observation, to iterate through observations. We can think of iteration most simply when we think of going through a list of elements. We can iterate through any set of elements.

The syntax starts with the word `for`; then comes the parentheses `(iterate_through_something)`; and then a curly brace where you can put the commands you want iterated. A curly brace also closes the process.

Within the parentheses, you first have an *index variable*, often called i That 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, "\n") #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\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") #\n is a line break
  cubed <- foo^3 #each time we assign a new value to squared
  cat(foo, "cubed is:", cubed, "\n\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
```

Often we use `for` loop to iterate through a data set. Here we are going to create a data set of the first and last year of each country at war.

First, we create an empty data set that we are going to write values into. Then, fill up that data set by iterating through the original data set.

To create the data set, we use the `matrix` function to create a empty matrix of the same number of rows as there are countries. We also we create three columns. We then give these columns names.

```
all_countries <- unique(as.character(cw_alt$country)) #we want the country name not the factor level

first_last_year <- data.frame(matrix(NA,
                                     nrow =length(unique(all_countries)), #number of rows
                                     ncol = 3)) #number of columns

names(first_last_year) <- c("country", "first_year", "last_year") #set the names of the columns
```

Then, we loop through each of the countries in the data set. As we loop, we subset the data set and calculate the `min` and the `max` of each subset of the data. (Note in this case we could also use `tapply`)

```
for(i in 1:length(all_countries)) {
  cat(all_countries[i], "\n") # see where we are in the process

  first_last_year$country[i] <- all_countries[i] #set the value for "country" in the new df

  one_country_only <- filter(cw_alt, country == all_countries[i]) #filter only the data associated with

  first_last_year$first_year[i] <- min(one_country_only$year) #set the value for the earliest year for
  first_last_year$last_year[i] <-  max(one_country_only$year) #set the value for the latest year for th
}
```

```
## CUBA
## HAITI
## DOMINICAN REP.
## GUATEMALA
## EL SALVADOR
## NICARAGUA
## COSTARICA
## COLOMBIA
## PERU
## BOLIVIA
## PARAGUAY
## ARGENTINA
## UK
## CROATIA
```

```
## YUGOSLAVIA
## BOSNIA
## GREECE
## CYPRUS
## MOLDOVA
## RUSSIA
## GEORGIA
## AZERBAIJAN
## GUINEA BISSAU
## MALI
## SENEGAL
## LIBERIA
## SIERRA LEONE
## NIGERIA
## CENTRAL AFRICAN REP.
## CHAD
## CONGO
## DEM. REP. CONGO
## UGANDA
## BURUNDI
## RWANDA
## SOMALIA
## DJIBOUTI
## ETHIOPIA
## ANGOLA
## MOZAMBIQUE
## ZIMBABWE
## SOUTH AFRICA
## MOROCCO
## ALGERIA
## SUDAN
## IRAN
## TURKEY
## IRAQ
## LEBANON
## JORDAN
## YEMEN ARAB REP.
## YEMEN
## YEMEN PEOP. REP.
## AFGHANISTAN
## TAJIKISTAN
## CHINA
## KOREA, S.
## INDIA
## PAKISTAN
## BANGLADESH
## BURMA
## SRI LANKA
## NEPAL
## CAMBODIA
## LAOS
## VIETNAM, S.
## PHILIPPINES
## INDONESIA
```

```
## PAPUA N.G.
```

```
head(first_last_year)
```

```
##          country first_year last_year
## 1           CUBA       1958      1959
## 2          HAITI       1991      1995
## 3 DOMINICAN REP.       1965      1965
## 4      GUATEMALA       1968      1996
## 5    EL SALVADOR       1979      1992
## 6      NICARAGUA       1978      1988
```

We now see that we have created a new data set with the first and last year for each country.

Problem 4

1. Create a dataframe called cw_1990 from the civil war data with only values from the year 1990.
2. Using a `for` loop, create a new dataframe from cw_1990 called min_max_population that stores the name of the region, the highest population value of that region, and the lowest population value of that region. (Three columns: region, max_pop and min_pop)
3. Print the first few rows of the dataframe you created using head()

Answer to Problem 4

```
cw_1990 <- filter(cw,
             year == 1990)

all_regions <- unique(as.character(cw_1990$region)) #we want the country name not the factor level

min_max_population <- data.frame(matrix(NA,
                                        nrow =length(unique(all_regions)),
                                        ncol = 3))

names(min_max_population) <- c("region", "max_pop", "min_pop")


for(i in 1:length(all_regions)) {
  cat(all_regions[i], "\n") # see where we are in the process

  min_max_population$region[i] <- all_regions[i]

  one_region_only <- filter(cw, region == all_regions[i])

  min_max_population$max_pop[i] <- max(one_region_only$pop, na.rm = T)
  min_max_population$min_pop[i] <-  min(one_region_only$pop, na.rm = T)
}
```

```
## western democracies and japan
## latin america and the caribbean
## e. europe and the former soviet union
```

```
## n. africa and the middle east
## sub-saharan africa
## asia
```

```
head(min_max_population)
```

```
##                                region   max_pop min_pop
## 1         western democracies and japan  270029.0    1688
## 2       latin america and the caribbean  165873.6     621
## 3 e. europe and the former soviet union  287630.0    1114
## 4         n. africa and the middle east   63451.0     222
## 5                   sub-saharan africa  121257.3     270
## 6                                 asia 1238599.4     520
```