Lab 10 - Using R Markdown

Bree Bang-Jensen

1/8/2019

R Markdown Basics

We will now be doing all of our labs and homework in R Markdown, rather than in a basic .R script file. This is also how you will write the code for your own analyses for your projects. Welcome to the future!

Go ahead and open a new R Markdown file – it's in the pull down menu where you've previously been selecting a new R Script file.

The basic point of R Markdown is to discuss code, write code, show results easily, and do all of this in one place. Let's break that down.

Discuss Your Code

This is as simple as typing in a Word document. To structure your discussion, you can add headers and subheaders (as above), *italics*, and **bold**. You can also add block quotes.

> Here's a thing I'd like to emphasize. Maybe a formula. Or an important fact.

We can also add horizontal lines. There are other formatting options, and the R Markdown cheat sheet is a handy way to keep them all straight (I've also added this PDF to the lab blog for this week: (https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf)

Do Everything All in One Place

When you click the **Knit** button and knit to HTML a document will be generated (and saved in your working directory) that includes both the written content, as well as any embedded code chunks and the output of those code chunks within the document.

But wait: what do you mean *embedded code chunks*?

Write Code & Print Results

You can add chunks of code to your R Markdown document using the **Insert** button, and selecting R. This is what a basic piece of embedded R code looks like:

```
x <- 10
x
```

```
## [1] 10
```

```
print("sup")
```

```
## [1] "sup"
```

Your code should all go on lines between the triple accent marks.

There are some important options available to you for each code chunk:

- **eval** (default is TRUE): Whether to evaluate the code and include its results
- **echo** (default is TRUE): Whether to display code along with its results
- **warning** (default is TRUE): Whether to display warnings
- **error** (default is FALSE): Whether to display errors
- **message** (default is TRUE): Whether to display messages
- **tidy** (default is FALSE): Whether to reformat code in a tidy way when displaying it
- **include**(default is TRUE): Whether to include all output (code, results, messages, etc.)

Here's a demonstration of where these options go in your code chunk: the header.

```
y <- 20
y
```

```
## [1] 20
```

This will print the results in the knitted R Markdown document – in this case, the value of y – without printing the code that generated the results.

In general, you do not want to display errors, warnings, or messages in your R Markdown document, so you should set those options to FALSE as needed.

How Not to Use R Markdown

Let's look at an embedded code chunk using a variant of code from an old homework. Here's a good example of how to make a messy and sad R Markdown homework.

So this is problem 4:

```
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

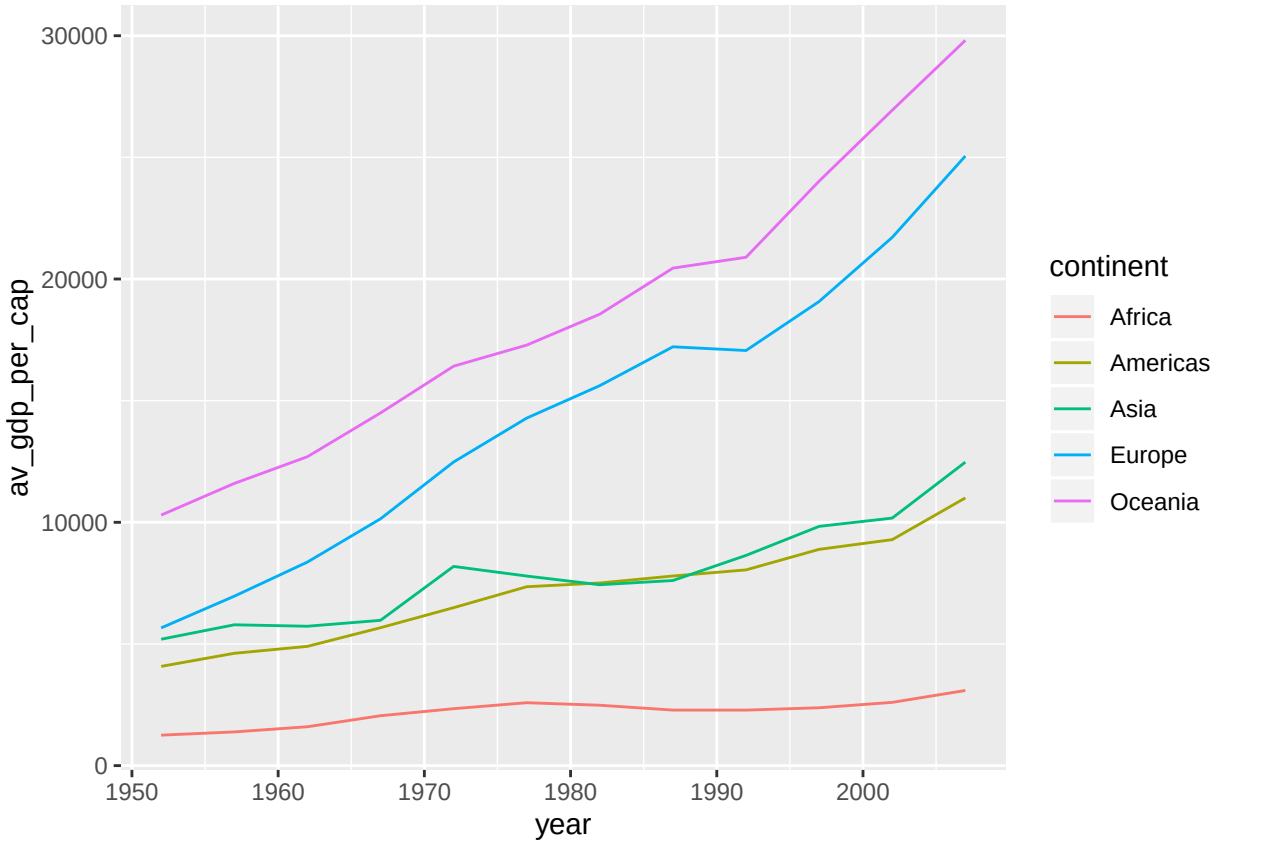
```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library("gapminder")
library("ggplot2")

gdp_table <- gapminder %>%
  group_by(year, continent) %>%
  summarize(av_gdp_per_cap = mean(gdpPercap))

gdp_table
```

```
## # A tibble: 60 x 3
## # Groups:   year [12]
##     year continent av_gdp_per_cap
##    <int> <fct>              <dbl>
##  1  1952 Africa             1253.
```

```
##  2  1952 Americas           4079.
##  3  1952 Asia               5195.
##  4  1952 Europe             5661.
##  5  1952 Oceania           10298.
##  6  1957 Africa             1385.
##  7  1957 Americas           4616.
##  8  1957 Asia               5788.
##  9  1957 Europe             6963.
## 10  1957 Oceania           11599.
## # ... with 50 more rows
```

```
ggplot(gdp_table, aes(x = year, y = av_gdp_per_cap, color = continent)) + geom_line()
```



A Better Way

Problem 4

First, I will load the required packages: dplyr, ggplot2, and the gapminder data set.

```
library("dplyr")
library("gapminder")
library("ggplot2")
```

Then, I will group the gapminder data by year, and then by continent, and then use the summarize function on each "year-continent" group in order to calculate the mean.

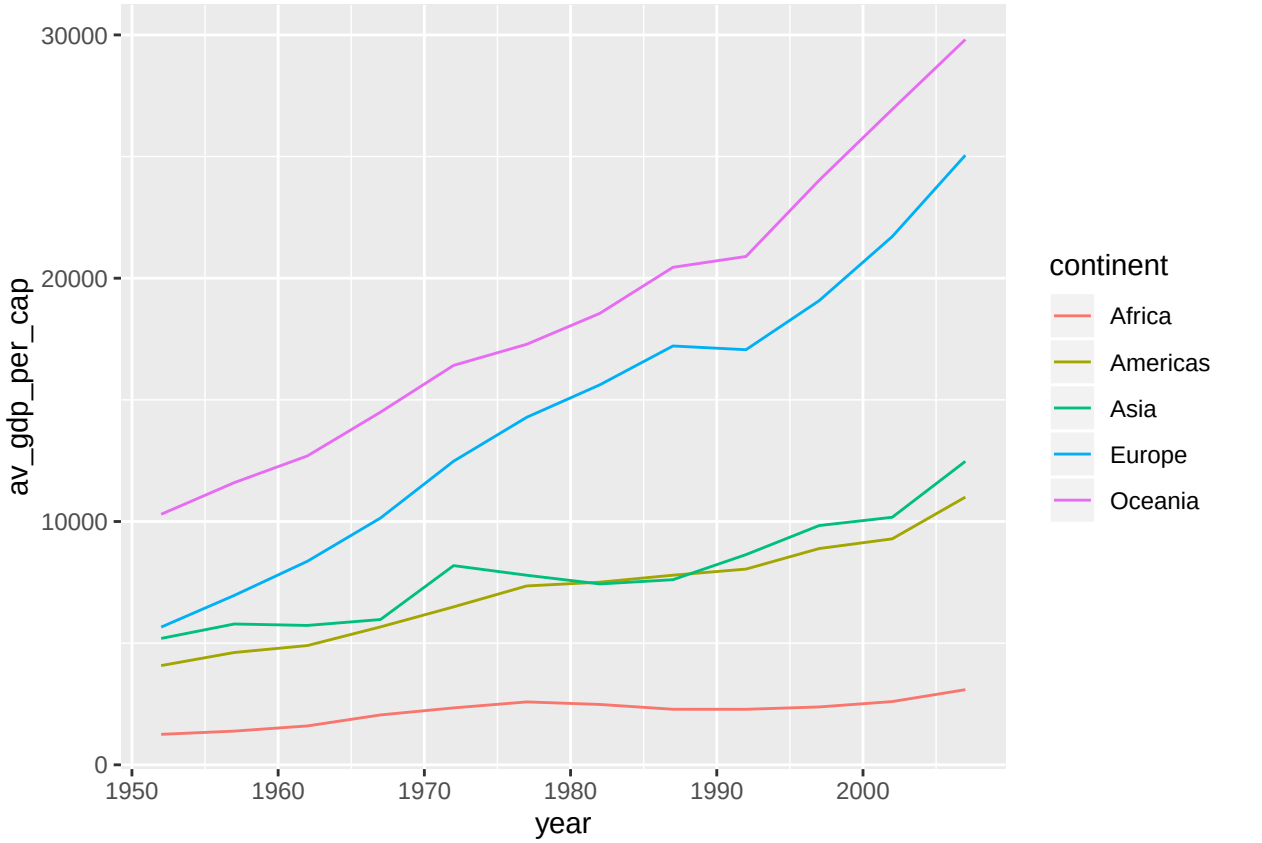
```r
gdp_table <- gapminder %>%
  group_by(year, continent) %>%
  summarize(av_gdp_per_cap = mean(gdpPercap))
```

Here is the head of the resulting table of average GDP by continent by year.

```
## # A tibble: 6 x 3
## # Groups:   year [2]
##    year continent av_gdp_per_cap
##   <int> <fct>              <dbl>
## 1  1952 Africa             1253.
## 2  1952 Americas           4079.
## 3  1952 Asia               5195.
## 4  1952 Europe             5661.
## 5  1952 Oceania           10298.
## 6  1957 Africa             1385.
```

This table is not the easiest way to conceptualize the data, though: a graph showing change over time in average GDP for each continent is much easier to understand.

```r
ggplot(gdp_table, aes(x = year, y = av_gdp_per_cap, color = continent)) + geom_line()
```



Couple Additional Notes (courtesy of Karl Broman)

Naming code chunks: It's usually best to give each code chunk a name, like simulate_data and chunk_name above. The name is optional; if included, each code chunk needs a distinct name. The advantage of giving each chunk a name is that it will be easier to understand where to look for errors, should they occur. Also, any figures that are created will be given names based on the name of the code chunk that produced them.

Here's where the name would go:

```
x
```

```
## [1] 10
```

Setting global options: You may be inclined to use largely the same set of chunk options throughout a document. But it would be a pain to retype those options in every chunk. Thus, you want to set some global chunk options at the top of your document.

It could look something like this:

```
knitr::opts_chunk$set(echo = FALSE, warning = FALSE, message = FALSE)
```

Because the global chunk options become the defaults for the rest of the document, you have to specify a different option (within that chunk) if you want a particular chunk to have a different behavior.

Using In-Line Code: When discussing the results of a given calculation in your written commentary, we want to be referencing R calculations, not simply typing numbers that appeared in the console.

That's the point of using in-line code. You'd write something like "There are 1704 observations in the data." Another example: "The estimated correlation between x and y was 0.2273181."