# Getting Started wiht R Markdown

*Lily Bliznashka, based on content by Joe McIntyre and Sophie Litschwartz*

## Overview

R Markdown is a simple but powerful markdown language which you can use to create documents with inline R code and results. This makes it much easier for you to complete homework assignments and reports; makes it much less likely that your work will include errors; and makes your work much easier to reproduce. For example, if you find you have to drop cases from your dataset, you can simply add that line of code to your document, and recompile your document; any text that's drawn directly from your analyses will be automatically updated.

Other R packages, such as Sweave and knitr, allow you to do the same things, but R Markdown has the added advantage of being relatively simple to use. This document will show you how to use R Markdown to create documents which draw directly on your data to produce reports.

## Getting started

You should take a look at the rmarkdown-cheatsheet.pdf and rmarkdown-reference.pdf documents on Canvas (under Files/Handouts). They have more information than this document does.

Every R Markdown document starts with a header. Headers look like this:

```
---
title: "Don't panic"
author: "R whiz wannabe"
output: pdf_document
---
```

A header can contain more or less information, as you see fit. Your computer needs to have a copy of LaTeX installed in order to output .pdf documents. If you don't, you should change `output: pdf_document` to `output: html_document` or `output: word_document`.

You identify sections of the document using hashtags; more hashtags indicate less important sections.

For example, this:

```
# A big section
```

produces this:

## A big section

while this

```
###### A little section
```

produces this:

A little section

If your document includes a table of contents, the sections get used to automatically generate the table of contents.

You can *italicize* words by writing `*italicize*`. You can **bold** words with `**bold**`.

Frequently in this class, we'll be writing models. R Markdown can use LaTeX style math-writing to display mathematical script. We'll provide a handout with LaTeX syntax for the mosty commonly used models in the class.

For example, the following statement

```
$$Achieve_i = \beta_0 + \beta_1 Gender_i + \epsilon_i$$
```

compiles to

$$Achieve_i = \beta_0 + \beta_1 Gender_i + \epsilon_i$$

If it's a fitted model rather than a population model, we might write

```
$$\hat{Achieve}_i = \hat{\beta}_0 + \hat{\beta}_1 Gener_i$$
```

which gives us

$$\hat{Achieve}_i = \hat{\beta}_0 + \hat{\beta}_1 Gener_i$$

Lists are easy in R Markdown as well. For unordered lists, write

```
- Item 1
- Item 2
- Item 3
```

to get

- Item 1
- Item 2
- Item 3

For ordered lists, write

```
1. Item 1
2. Item 2
3. Item 3
```

to get

1. Item 1
2. Item 2
3. Item 3

To start a new page, just type `\newpage`.

As you may have noticed, one of the driving ideas behind R Markdown is that the text should be interpretable even if it's not compiled. A person should be able to read this text file and understand the basic organization and what all of the symbols denote.

# Embedding R code

## The basics

Of course, the real goal of using R Markdown is to embed R code; otherwise you'd be better off just using a word processor.

The simplest way to embed R code is inline. If you write, for example, `` `r mean(c(1, 2, 3))` `` , your document will display 2. That's a pretty silly use of R Markdown, but if you wanted to display the mean of a variable, and that variable might change (for example, if you created a document to generate a dynamic report from a school's data, and you wanted to apply it to a new school), then it makes a lot more sense.

For larger tasks, you'll want to use an R code chunk. Code chunks have a number of different options. The most important ones for us right now are the `eval` and `echo` options. By default, `eval=TRUE`, which means every time you recompile your Markdown file, the code inside your R code chunk will be evaluated. If you set `echo=TRUE`, then R Markdown will render not only the results of the code, but also all of the code itself. For example, if I read in data in a code chunk where `echo` is set to true, this is what I get:

```
library(readstata13)
dat <- read.dta13("neighborhood.dta")
```

Not only is the code displayed, the command has actually been carried out, so I can now access `dat` directly from my document. For the purposes of the class, we'll keep `echo` set to true so you can see code chunks and follow along easier. For assignments, you should also leave `echo` set to true, so that we can tell why things have gone wrong, if they have. If you don't want to display the code (e.g., code chunks that load and manipulate data), you can set `echo=FALSE`. For example, if I set my code chunk options as `{r, echo = FALSE}`, and inside the chunk have

```
cat("The mean school attainment is", round(mean(dat$attain),1 ))
```

then the document will display

```
## The mean school attainment is 0.1
```

If the data change *so will the text* (at least as soon as I recompile/knit the document)!

I can also run code segments individually to test out code. If I look in the top right corner there is a little button that says run. If I click the button there are options to run the current code chunk, the previous code chunk, and the next code chunk.

## Embedding plots

We can embed plots in exactly the same way. For example,

```
library(tidyverse)  # load ggplot, notice you can comment in R code chunks
```

```
## -- Attaching packages ---------------------------------------------------------- tidyverse 1.2.1 -
```

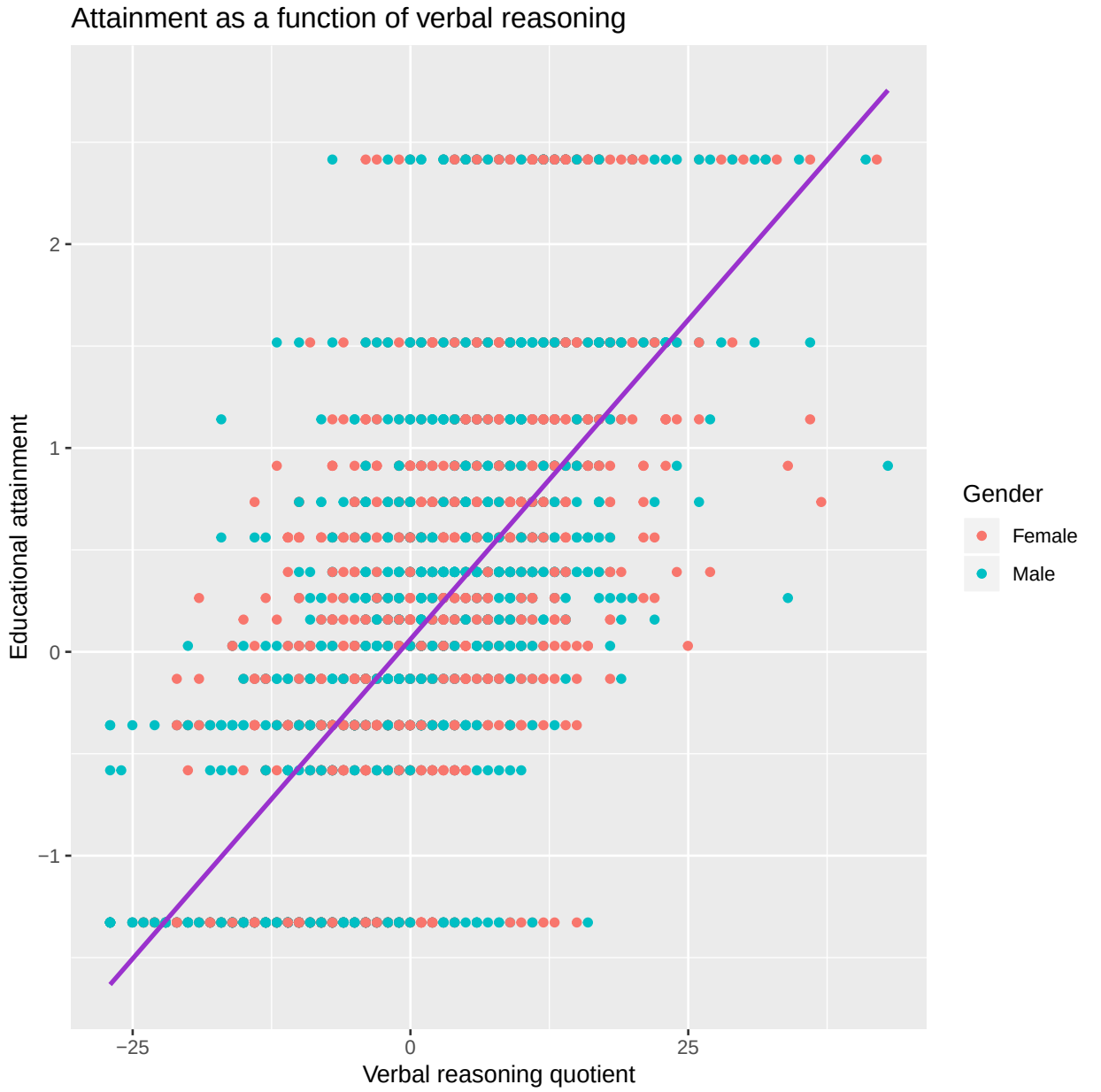
```
## v ggplot2 3.2.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------ tidyverse_conflicts() -
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
dat$male <- factor(dat$male, levels = c(0, 1), labels = c("Female", "Male"))
```

```
ggplot(data=dat, aes(p7vrq, attain, colour=male)) +
  geom_point() +
```

```
    labs(title="Attainment as a function of verbal reasoning",
        x = "Verbal reasoning quotient", y = "Educational attainment", colour="Gender") +
    geom_smooth(method="lm", se=FALSE, colour="darkorchid3")
```

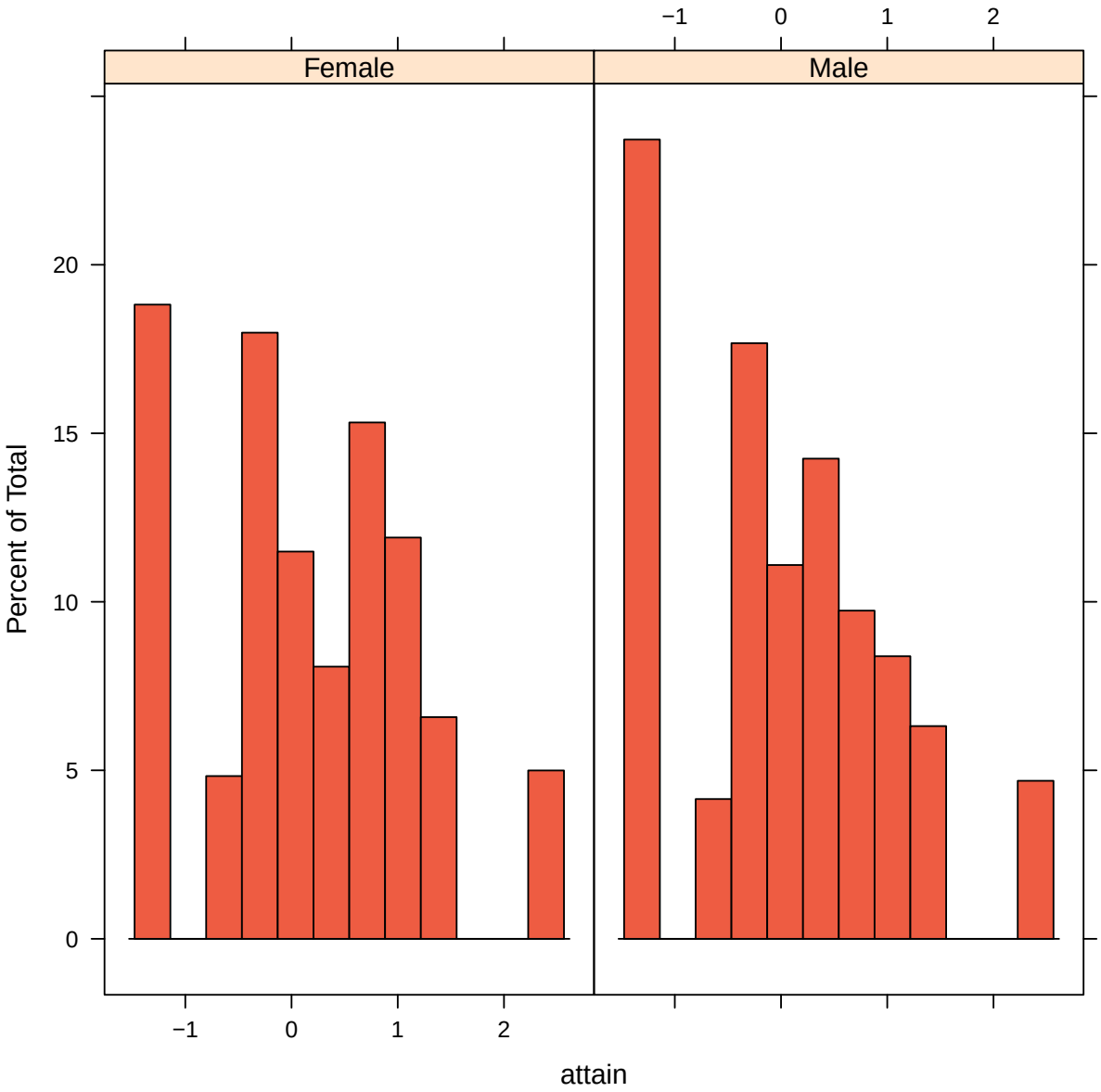



Girls are rendered as peach, boys are rendedered in turquoise, and the line of best fit is drawn in `darkorchid3` (because why not). Just because you have a lot of colors and plotting characters to work with doesn't mean you need to use them all. In the options, I specified `fig.width = 7` and `fig.height = 7`. Notice that this command draws on `dat`, which we already defined in a previous code chuck; it will persist throughout the R Markdown document until we're done with it.

Heres another example, this time a lattice histogram.

```
  library(lattice)

  histogram(~attain | male, data=dat, col="tomato2")
```


Female
Male
Percent of Total
attain
0
5
10
15
20
−1
0
1
2

## Embedding tables

You can directly tell R Markdown how to render a table, but there are also a ton of great packages to do that for you. For descriptive tables, you can use the `tableone` package to output the table. For example:

```
library(tableone)

CreateTableOne(data = dat, vars = c("attain", "p7vrq"), strata = c("male"))
```

```
##                      Stratified by male
##                       Female       Male          p      test
##   n                    1201          1109
##   attain (mean (SD)) 0.15 (0.99)   0.03 (1.01)   0.004
##   p7vrq (mean (SD))  1.23 (10.38) -0.28 (10.88)  0.001
```

You can also use a nifty package, `stargazer` or `texreg`, to output an attractive taxonomy of regresison models. We recommend `texreg`, which automatically outputs the variances of random effects (more on this in a week or two).

For example:

```
library(texreg)
```

```
## Version:  1.36.23
## Date:     2017-03-03
## Author:   Philip Leifeld (University of Glasgow)
##
## Please cite the JSS article in your publications -- see citation("texreg").
```

```
##
## Attaching package: 'texreg'
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
# fit some models
m1 <- lm(attain ~ male, data=dat)
m2 <- lm(attain ~ male + momed, data=dat)
m3 <- lm(attain ~ male + momed + daded, data=dat)

screenreg(list(m1,m2,m3), custom.coef.names=c("Intercept", "Male", "Maternal education", "Paternal edu
```

```
##
## ============================================================
##                      Model 1      Model 2      Model 3
## ------------------------------------------------------------
## Intercept              0.15 ***     0.03        -0.02
##                       (0.03)       (0.03)       (0.03)
## Male                  -0.12 **     -0.12 **     -0.12 **
##                       (0.04)       (0.04)       (0.04)
## Maternal education                  0.49 ***     0.24 ***
##                                     (0.05)       (0.05)
## Paternal education                               0.54 ***
##                                                 (0.06)
## ------------------------------------------------------------
## R^2                    0.00         0.05         0.09
## Adj. R^2               0.00         0.05         0.08
## Num. obs.           2310         2310         2310
## RMSE                   1.00         0.98         0.96
## ============================================================
## *** p < 0.001, ** p < 0.01, * p < 0.05
```

Both pacakge include a lot of options and make it easy to produce publication-quality tables with little effort. We have provided more resources on Canvas.