

Resources for S043/Stat151: Multilevel and Longitudinal Models

Luke Miratrix and Friends

2023-09-30

Table of contents

Preface

This online book has a bunch of resources for S-043: Multilevel and Longitudinal Models. The book is written in [Quarto](#), and is basically a bunch of handouts stapled together. It is very much a work in progress. If you notice errors, please notify us at:

lmiratrix@gse.harvard.edu

joshua_gilbert@g.harvard.edu

Overview

There are several parts to the book, loosely arranged by type of handout. We list the parts next, with some brief overview.

R & R Markdown

The book starts with material on just using R and making tables and whatnot.

Using ggPlot

The ggPlot section's handouts are on using ggPlot, with an emphasis on using small multiples and other tricks to plot clustered or longitudinal data and results from multilevel data analysis. This section also includes how to use prediction to visualize a model's fit, which is especially important for longitudinal data, and plotting growth curves.

Model Fitting and Interpretation

This section has information on how to deal with the results from a `lmer()` call, and also has material connecting the code to the mathematical model. There are handouts on how to interpret parameters as well. This has help for much of the core content of the course.

Worked Examples

The section has several case studies that illustrate things such as three-level models. It also has all the code to replicate the High School and Beyond example from Chapter 4 of Raudenbush and Bryk,

Visualizations

The visualization section has some interactive visualiations made by Josh Gilbert that can bring some of the ideas of this course to life.

Math Derivations

The math derivations at the end show how some of the variance decomposition stuff works, or error correlation matrices are built.

Acknowledgements

Some of these handouts, or early drafts of these handouts, were written by the many prior TFs of this course. We have attributed authorship where we had it, but input from TFs has improved pretty much everything you see here. Thanks also to the many prior students who have asked for these handouts, given feedback, and overall have helped this course be what it is today (which, as you can tell from the number of handouts, is a lot).

Do I take S-043? A self-assessment

I received numerous inquiries as to whether a given background is enough to take this course. Let me elaborate at length. The stated prerequisite to this course is S052, Stat 139, or equivalent, i.e., you have gone a bit beyond an applied course on linear regression (S040). Technically, if you have taken S040 or equivalent, you could take this course; the “a bit beyond” part is just wanting a bit more comfort with these foundational skills. More specifically, we greatly prefer you to have the following skills as a prerequisite for this course:

- You can calculate summary statistics and visualizations for data (the mean, standard deviation, scatterplots, histograms) using some sort of statistical software.
- You can load data into some sort of statistical software, use that software to fit a regression model, and then interpret the results of the model.
- You know how to include categorical covariates in a linear regression.
- You know what an interaction term in a linear regression model is, and ideally can add one to a model you are fitting.
- You can interpret confidence intervals and p-values.
- You can write down a regression equation and identify what the covariates and parameters are in a presented regression equation.

You will have an easier time if you have some of the following skills and experiences as well:

- You have some experience with doing things in R (even a little helps here).
- You know what logistic regression is, and know how to fit a logistic regression model using some sort of statistical software.
- You have seen random intercept models before, as perhaps in S-052.
- You have seen regression with fixed effects for groups, as perhaps in an econometrics course.
- You have some experience doing quantitative research with real data in almost any capacity.

If you have a strong mathematical background, or strong comfort with math, or if you have a strong computer programming background, or strong comfort with that, then you can likely get a lot out of this course even if you do not have some of the above skills and knowledge.

People from many, many different backgrounds take S043/Stat151. In general, the more background experience you have, the easier the course. Even if you don't have as solid a background, you can still get a lot out of this course if you are willing to sign on for an intense experience. But *please* don't be surprised if you sign up for an intense experience... and it is really intense!

To get a very rough sense of what it might feel like, take the quiz below.

A self-assessment quiz

Trying to figure out how S43 will be for you? Add up the points across the following categories:

Work Experience

- +0 I have no work experience with quantitative data, or
- +1 I have some work experience with quantitative data, or
- +2 I have substantial work experience with quantitative data

Stat Courses

- 2 I have no prior stat experience under my belt, or
- +1 I have taken a very intro stat course (e.g., S12) or have at least a little stat knowledge, or
- +3 I have taken a linear regression course (e.g., S30, S40), or
- +5 I have taken an intermediate or advanced stats course (e.g., S52 or beyond)
- +1 bonus if concurrently enrolling in S052
- +1 bonus if classes were comfortable for you

Programming

- +0 I have basically no experience doing computer programming, or
- +1 I have some experience doing computer programming, or
- +2 I have a lot of experience doing computer programming

R Skills

- +0 I have no real experience with R, or
 - +1 I have a small bit of experience with R (e.g., ran scripts of it in S040), or
 - +2 I have some experience with R (e.g., played around with scripts a bit in S040), or
 - +3 I have substantial experience with R (e.g., write my own R code for my work)
- +1 bonus if learning R has been comfortable for you
+1 bonus if you are comfortable with something like STATA

Math

- +0 I don't recall much math from my past education, or
 - +1 I have taken (and somewhat remember) Calculus, or
 - +2 I have taken classes beyond Calculus
- +1 bonus if classes were comfortable for you

Other

- +2 I am content with getting a B or taking the class SAT/UNSAT

Total the above

A VERY ROUGH recommendation is:

< 4: **Danger!** Please email the instructor to talk about how this might go for you.

4-6: **It will be really hard!** You could take this course, but will likely find it will take much more time than a typical course. We would support you through this, but be warned that this could feel like a lot to take on.

7-9: **It will be hard!** There are lots of folks like you who are taking this course. The course will likely be a fair bit of work and could feel confusing/overwhelming at times. We would support you through this. At the end you will have learned a lot if you stick with it.

10+: **It will probably feel like a normal course.** No reservations. You can either work a reasonable amount and learn a lot about data science, or dig deeper to really go far with the skills we cover.

14+: **It will probably be a cake-walk.** You will learn some concepts but not have to work particularly hard in this course. We would still love to have you!

Students have historically said this course teaches you a lot; the question is just whether you have the time to allocate for the course. This quiz helps assess the time.

Getting Started with R

To get started with R, first download and install R and RStudio (the IDE, or integrated development environment, that makes using R much, much more pleasant). See links at [this HGSE software support page](#).

Some Resources and Directions

The following are ways of getting help with R.

Handouts

This textbook has many handouts written by the teaching team that illustrates how to use R for a variety of tasks. Also see the Resources page on Canvas for further commentary and organization.

Class Code

Each class has a R script that shows how to do the stuff from that class. See the Packets on Canvas for this code.

A Very Important Online Textbook

See [R for Data Science..](#). This textbook provides important information on wrangling data, making plots, and doing statistical programming. It is full of examples and code snippets you can steal.

Class Sections

Sections will typically have a hands-on component which will give you a chance to try things out yourself. These sections will also publish the final R code for future reference. See the section pages on Canvas to get this information.

Office Hours

Office hours are fine time to get help troubleshooting a specific or script you are working on.

GSE Stat Help Desk

Education students can write to stathelp@gse.harvard.edu to get help getting started with R. If groups of 3 or more want some tutorials, they can ask for them as well.

The Help Desk sometimes has Intro to R workshops. For example, grab some old workshop materials here: <http://its.gse.harvard.edu/gentle-introduction-r>

Assignment Formatting Guidelines

The following describes our expectations on what turned in assignments should look like. Please read this document carefully before your first assignment. Note that many of these requirements will be automatically fulfilled if you use R Markdown.

Guiding Principle

Your submitted work should present your work clearly and without undue clutter, be easy to navigate, and adhere to basic publication norms.

The following are some specific details that further the guiding principle. It is not an exhaustive list.

Formatting

- Start each question on a new page.
- Provide at least a brief title for the question or sub-question along with the question number. E.g. “(a). Association between graduation rates and school type.” You can copy the entire prompt for your future reference if you want, but abbreviated prompts are fine with us!
- Use headings or other means to highlight problem titles (e.g., bold, italic, etc.).
- Make sure you answer all parts of each question, and do so under the proper labeled subpart. For each question or sub-question, include any R code, R output and answer.
- Use different fonts/formatting for your R code, R output and answers and use font formatting consistently throughout each assignment.
- Use single line spacing and normal 1-inch margins.
- Include page numbers.
- *Let people say of your work: “There is no bombast, no similes, flowers, digressions, or unnecessary descriptions. Everything tends directly to the catastrophe.” – Horace Walpole, The Castle of Otranto*

Numbers

- Round your final answers, not intermediary steps.

- Put a zero in front of a decimal place (e.g. 0.2 instead of .2). This is optional for bounded numbers (e.g., p-values or proportions).
- Round to the nearest meaningful digit. What this means is a little hard to say, and different people can have different standards. However, if you have a statistic with a standard error of 1, it's not meaningful to report decimals because the estimate simply isn't precise enough to estimate numbers that small. Similarly, if you're reporting average salaries, it's usually not meaningful to report past the hundreds place regardless of your precision, because tens of dollars are too small to matter. Because this is so imprecise, we'll give a lot of lee-way, but spurious precision will annoy. If you're not sure what this means in practice, feel free to ask a TF.
- Format your p-values! Round p-values to 3 places, and never report a p-value of 0. Instead say, for example, “ $p < 0.001$ ” or “ $p < 10^{-r}$ ” for some r .
- “*Numbers have life; they’re not just symbols on paper.*” – Shakuntala Devi

Plots

- Make sure your plots are well labelled, including title, axis, legends and any other elements you choose to include.
- Your plots should be self-explanatory.
- Include notes and captions as necessary.
- Try to make plots easier to compare when you have multiple plots. For example, it is nice to have the same X -axis bounds if giving two histograms.
- Do not include best fit lines unless you have some reason, e.g. a significant p -value or a scientific basis for understanding an association.
- “*Above all else show the data.*” –Edward Tufte

Avoiding bias

- Use plural phrases, nouns or pronouns, e.g. “children and their toys” for “a child and his toy.” You may use the singular “they” pronoun (“a child and their toy”), but be warned that broader academic communities are still in flux regarding this usage.
- Try to avoid biased forms of language concerning race, gender, disability and sexuality. A reasonable list of case studies to consider on this topic is <https://academicguides.waldenu.edu/writingcenter/bias>. The APA also has a guide for race at <https://apastyle.apa.org/style-grammar-guidelines/bias-free-language/racial-ethnic-minorities>.
- “*I have yet to see a piece of writing, political or non-political, that does not have a slant. All writing slants the way a writer leans, and no man is born perpendicular.*” E.B. White

Conciseness

- “*Brevity is the soul of wit.*” –Hamlet

Part I

R & R MARKDOWN

1 Intro to R Markdown

1.1 Overview

R Markdown (and its newer cousin Quarto) 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.

1.2 Getting started

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

```
---
```

```
title: "My perfect homework"
author: "R master"
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:

2 A big section

while this

```
## A small section
```

produces this:

2.1 A small 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*` or `_italicize_`. You can **bold** words with `**bold**` or `__bold__`. N.B. Newer versions of Markdown and Quarto have a [visual editor](#) that allows you to format things in the usual way, e.g., control-B for bold. We generally recommend using the visual editor.

You can add superscripts ($E=mc^2$) by writing `E=mc^2^`.

You can create unordered lists:

- Item 1
- Item 2
- Item 3

to get

- Item 1
- Item 2
- Item 3

Or ordered lists:

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` (not relevant for HTML output).

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.

You can also add links and images, and do many other things beyond what we'll show you in this class. There are many resources out there, but [here's](#) one place you can start.

To compile or knit the document, click on the button that says **Knit** or **Render**, or Shift + Ctrl/Cmd + K.

2.2 Embedding R code

There are two main ways to embed R code in R Markdown, code chunks or inline.

2.2.1 Code chunks

To insert a code chunk click on **Insert** on the top right corner of your R Markdown file and select **R**. Or use keyboard shortcuts: Ctrl + Alt + I for PC and Cmd + Option + I for Mac:

Code chunks have a number of different options. The most important ones for us are:

- `eval = TRUE`, which means every time you knit the file, the code inside the R code chunk will get evaluated. This is the default.
- `echo = TRUE`, which means every time you knit the file, the code inside the R code chunk will be rendered, and you can see both the code itself and the results from evaluating the code.

For class, you should keep `echo = TRUE`, so that we can see your code and be able to tell what went wrong, if something did. You can set `echo = FALSE` for code chunks that load and manipulate data.

Other code chunks options you may see in class are:

- `warning = FALSE`, which means warning messages generated by the code will not be displayed.
- `results = 'asis'`, which means results will not be reformatted when the file is compiled (useful if results return raw HTML).

- `fig.height` and `fig.width`, which specify the height and width (in inches) of plots created by the chunk.

Let's try loading some data:

```
library(haven)
dat <- read_dta("data/neighborhood.dta")
```

You can see the code is displayed, and the command is carried out. The file `dat` is loaded in the R environment.

Instead of specifying code chunks options every time, you can specify them globally in the setup chunk by using `knitr::opts_chunk$set(echo = TRUE, eval = TRUE)`. You can then add additional options only to relevant chunks. If you want to exclude specific chunks, you can re-set `echo = FALSE` and `eval = FALSE` for those specific chunks.

Running code chunks: A good practice is to run individual code chunks to make sure they are doing what you want them to do. You can do this by executing individual lines of code, or whole chunks. Go to Run in the upper right corner and select what chunks to execute, e.g. Run Current Chunk, Run Next Chunk, etc.

2.2.2 Inline code

Code results can also be inserted directly in the text of your R Markdown file. This is particularly useful when you are extracting and interpreting model parameters. You can extract the coefficient from the model and use inline code to report it. If the data or model change, *the text will change too* when you knit the document.

To add inline code, enclose it in ``r``. For example, to report the mean reading score, you can use

```
`r mean(dat$p7read)`
```

Which will produce `-0.0443549`. That's a few too many decimals, let's round it off, using

```
`r round(mean(dat$p7read), 2)`
```

which produces “`-0.04`.”

Here we used two commands: `round` and `mean`. You can use more commands and write more complex inline code, depending on what you want to report.

2.3 Embedding plots

Plots are easy to embed. For example,

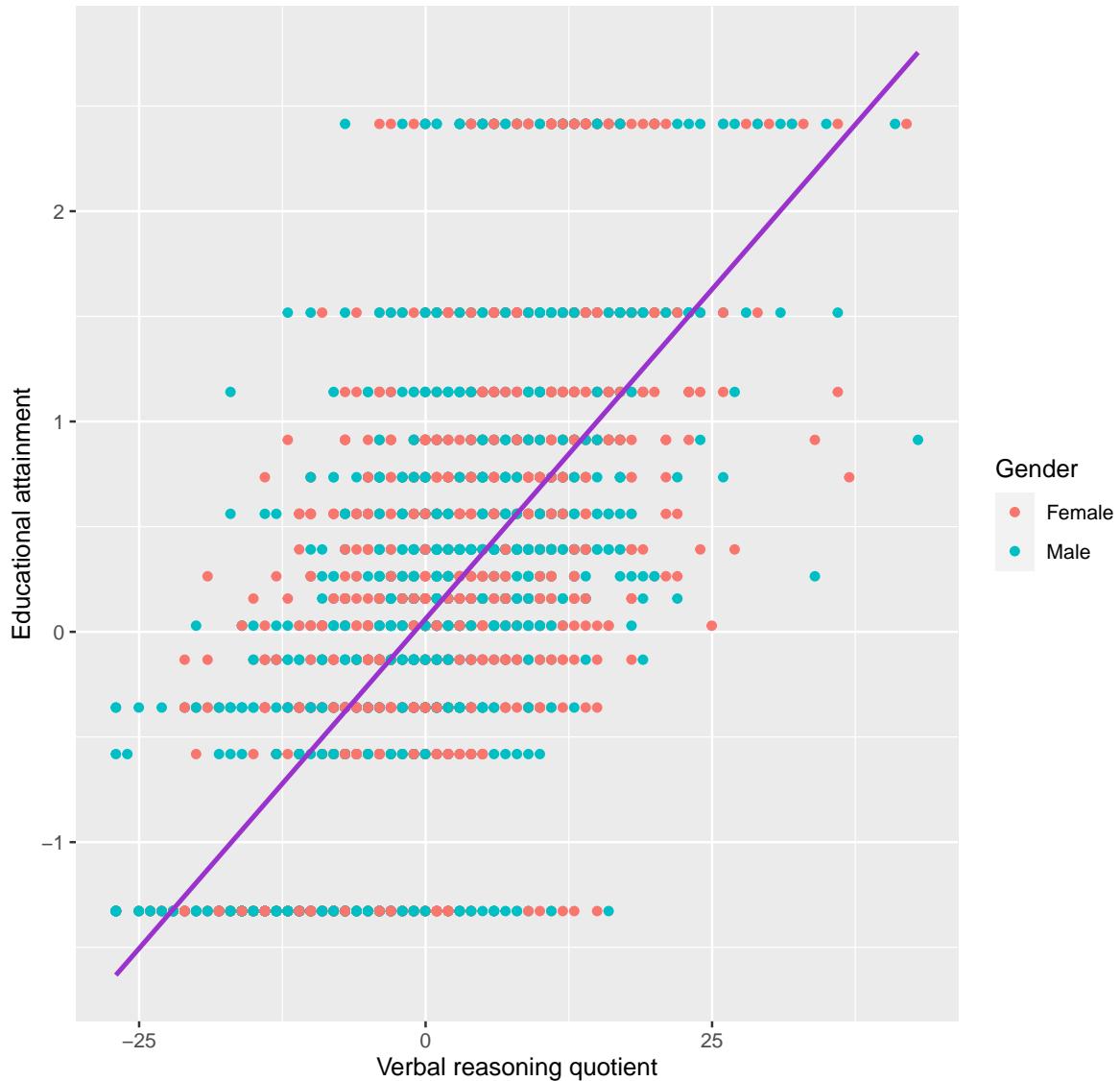
```
library(ggplot2)

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

ggplot(data=dat, aes(p7vrq, attain, colour=males)) +
  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")

`geom_smooth()` using formula = 'y ~ x'
```

Attainment as a function of verbal reasoning



Girls are rendered as coral, boys are rendered 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 loaded in a previous chunk. When knitting the document, code chunks get executed in order and the results persist throughout the R Markdown document.

For the purposes of class, we want to see both your plot code and the plot itself. It's not uncommon to use wrong code to create a plot that looks correct (at least visually).

2.4 Embedding tables

You can directly render tables in R Markdown. The idea is, inside an R chunk, you call a command that prints out a table. The report then takes this printout and integrates it into your overall report. There are many different packages to make tables, but in class we'll mostly use `texreg`, `stargazer`, and the `tab_model()` function in `sjPlot`.

You can use these packages to create a descriptive table. For example:

```
# Here I specified results = asis. If I didn't stargazer will just render the table in html.  
# This is bad if you are making a pdf document.  
# I also specified message = FALSE, so that the package citation gets suppressed  
library(stargazer)  
stargazer(dat, type = "latex")
```

% Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail:
marek.hlavac at gmail.com % Date and time: Sun, Sep 10, 2023 - 22:29:17

Table 2.1

Statistic	N	Mean	St. Dev.	Min	Max
-----------	---	------	----------	-----	-----

See Chapter ?? for more on making various tables.

We can also use `texreg` or `stargazer` to create a taxonomy of regression models. We recommend `texreg`, which automatically outputs the variances of random effects (more on this soon).

For example:

```
library(texreg)

# 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 education"))
```

	Model 1	Model 2	Model 3
Intercept	0.15 *** (0.03)	0.03 (0.03)	-0.02 (0.03)
Male	-0.12 ** (0.04)	-0.12 ** (0.04)	-0.12 ** (0.04)
Maternal education		0.49 *** (0.05)	0.24 *** (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

*** p < 0.001; ** p < 0.01; * p < 0.05

Both packages include a lot of options and make it easy to produce publication-quality tables with little effort. We have provided more resources on Canvas and in other portions of this book (see ([make-regression-tables?](#)), in particular).

2.5 Embedding mathematical models

We'll be writing a lot of mathematical models in class. R Markdown can use [LaTeX](#) style math-writing to display mathematical script. Another chapter in the book has more resources with [LaTeXsyntax](#) for the mostly commonly used models in the class. Similar to code chunks and inline code, you can use [LaTeX](#) for single or multiple equations, or for individual parameters embedded in the text.

For example, the following statement

```
 $$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$
```

compiles to

$$Y_i = \beta_0 + \beta_1 Y_i + \epsilon_i$$

And the following statement `μ` compiles to μ . This will be very helpful when we ask you to match R output to model parameters in homework.

2.6 Help! R Markdown report generation doesn't work

Don't put "View()" in your Markdown file when loading your csv file. Just put in the `read_csv` line. Otherwise you will not be able to knit.

Also watch for the `skim()` command—it can crash report generation as well.

If you can't knit PDFs you need to install latex (tex). Once you do, reboot your computer. If things don't work, then knit to Microsoft word (or, failing that, html as a last resort), print to pdf, and turn that in. But then ask a teaching fellow to help get things set up, since PDFs make for much more readable reports.

2.6.1

3 Intro to Regression

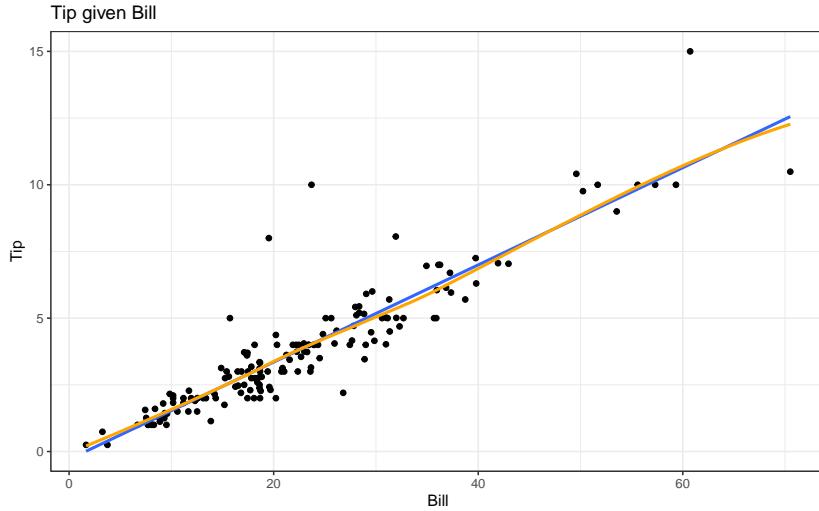
This walkthrough shows how to fit simple linear regression models in R. Linear regression is the main way researchers tend to examine the relationships between multiple variables. This document runs through some code without too much discussion, with the assumption that you are already familiar with interpretation of such models.

3.1 Simple Regression

We are going to use an example dataset, `RestaurantTips`, that records tip amounts for a series of bills. Let's first regress `Tip` on `Bill`. Before doing regression, we should plot the data to make sure using simple linear regression is reasonable. For kicks, we add in an automatic regression line as well by taking advantage of ggplot's `geom_smooth()` method:

```
# load the data into memory
data(RestaurantTips)

# plot Tip on Bill
ggplot( RestaurantTips, aes(x = Bill, y = Tip) ) +
  geom_point() +
  geom_smooth( method="lm", se=FALSE ) +
  geom_smooth( method="loess", se=FALSE, col="orange" ) +
  labs(title = "Tip given Bill")
```



That looks pretty darn linear! There are a few unusually large tips, but no extreme outliers, and variability appears to be constant at all levels of `Bill`, so we proceed:

```
# fit the linear model
mod <- lm(Tip ~ Bill, data = RestaurantTips)
summary(mod)
```

```
Call:
lm(formula = Tip ~ Bill, data = RestaurantTips)

Residuals:
    Min      1Q  Median      3Q     Max 
-2.391 -0.489 -0.111  0.284  5.974 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -0.29227   0.16616  -1.76   0.081 .  
Bill         0.18221   0.00645  28.25  <2e-16 *** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.98 on 155 degrees of freedom
Multiple R-squared:  0.837, Adjusted R-squared:  0.836 
F-statistic: 798 on 1 and 155 DF,  p-value: <2e-16
```

The first line tells R to fit the regression. The thing on the left of the `~` is our outcome, the things on the right are our covariates or predictors. R then saves the results of all that work under the name `mod` (short for model - you can call it anything you want). Once we fit the model, we used `summary()` command to print the output to the screen.

Results relevant to the intercept are in the `(Intercept)` row and results relevant to the slope are in the `Bill` row (`Bill` is the explanatory variable). The `Estimate` column gives the estimated coefficients, the `Std. Error` column gives the standard error for these estimates, the `t value` is simply estimate/SE, and the p-value is the result of a hypothesis test testing whether that coefficient is significantly different from 0.

We also see the RMSE as `Residual standard error` and R^2 as `Multiple R-squared`. The last line of the regression output gives details relevant to an ANOVA table for testing our model against no model. It has the F-statistic, degrees of freedom, and p-value.

You can pull the coefficients of your model out with the `coef()` command:

```
coef(mod)
```

(Intercept)	Bill
-0.292	0.182

```
coef(mod)[1] # intercept
```

(Intercept)
-0.292

```
coef(mod)[2] # slope
```

Bill
0.182

```
coef(mod)["Bill"] # alternate way.
```

Bill
0.182

Alternatively, you can use the `tidy()` function from `broom` to turn the regression results into a tidy data frame, which makes it easier to work with:

```
tidy(mod)
```

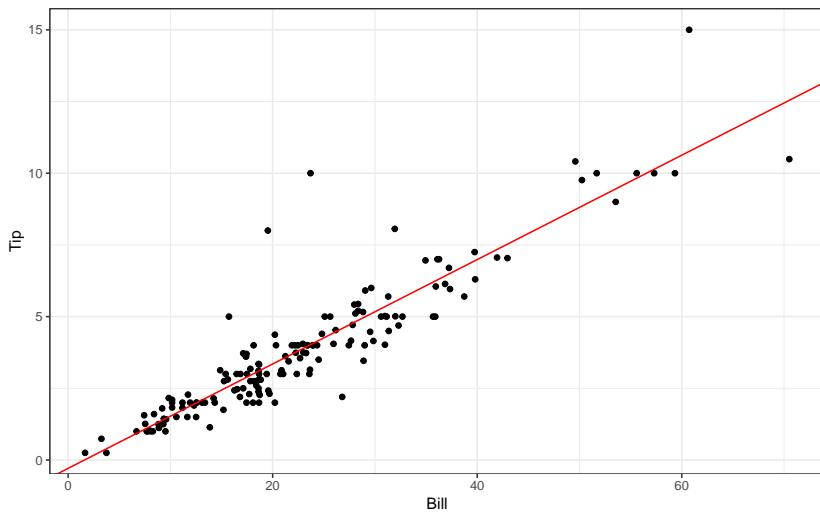
```
# A tibble: 2 x 5
  term      estimate std.error statistic p.value
  <chr>      <dbl>     <dbl>      <dbl>    <dbl>
1 (Intercept) -0.292    0.166     -1.76 8.06e- 2
2 Bill         0.182    0.00645    28.2  5.24e-63
```

```
tidy(mod)[[2,2]] # slope
```

```
[1] 0.182
```

We can plot our regression line on top of the scatterplot manually using the `geom_abline()` layer in ggplot:

```
ggplot( RestaurantTips, aes( Bill, Tip ) ) +
  geom_point() +
  geom_abline( intercept = -0.292, slope = 0.182, col="red" )
```



3.2 Multiple Regression

We now include the additional explanatory variables of number in party (`Guests`) and whether or not they pay with a credit card (`Credit`):

```
tip.mod <- lm(Tip ~ Bill + Guests + Credit, data=RestaurantTips )
summary(tip.mod)
```

Call:
lm(formula = Tip ~ Bill + Guests + Credit, data = RestaurantTips)

Residuals:

Min	1Q	Median	3Q	Max
-2.384	-0.478	-0.108	0.272	5.984

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.25468	0.20273	-1.26	0.21
Bill	0.18302	0.00846	21.64	<2e-16 ***
Guests	-0.03319	0.10282	-0.32	0.75
Credit	0.04217	0.18282	0.23	0.82

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

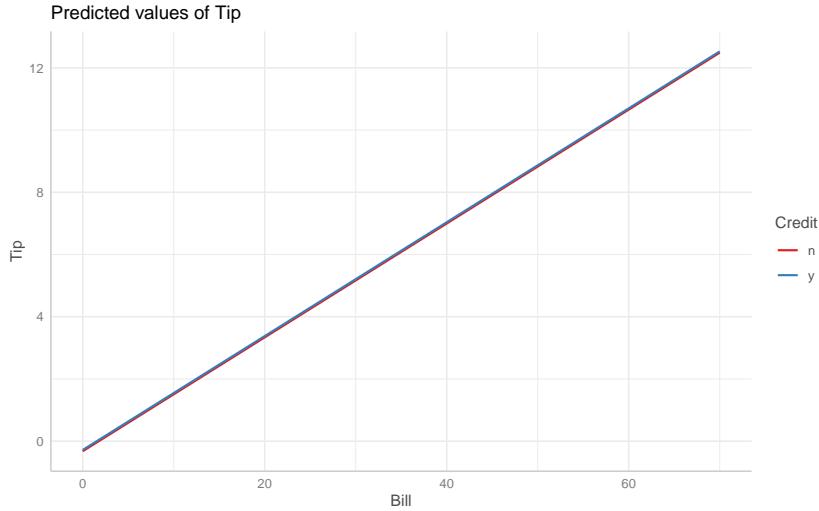
Residual standard error: 0.985 on 153 degrees of freedom
Multiple R-squared: 0.838, Adjusted R-squared: 0.834
F-statistic: 263 on 3 and 153 DF, p-value: <2e-16

This output should look very similar to the output for one variable, except now there is a row corresponding to each explanatory variable. Our two-category (y, n) Credit variable was automatically converted to a 0-1 dummy variable (with “y” being 1 and “n” our baseline).

You can make plots and tables of your fit models. For one easy kind of regression graph, try ggeffects:

```
# graph model 2, with Bill on X, Credit as color, and Guests held constant at the mean
ggeffect(tip.mod, terms = c("Bill", "Credit")) |>
  plot(add.data = TRUE, ci = FALSE)
```

Raw data not available.



For making tables, Chapter ??.

3.3 Categorical Variables (and Factors)

You can include any explanatory categorical variable in a multiple regression model, and R will automatically create corresponding 0/1 variables. For example, if you were to include gender coded as male/female, R would create a variable GenderMale that is 1 for males and 0 for females.

3.3.1 Numbers Coding Categories.

If you have multiple levels of a category, but your levels are coded with numbers you have to be a bit careful because R can treat this as a quantitative (continuous) variable by mistake in some cases. You will know it did this if you only see the single variable on one line of your output. For categorical variables with k categories, you should see $k - 1$ lines.

To make a variable categorical, even if the levels are numbers, convert the variable to a factor with `as.factor` or `factor`:

```
# load the US states data
data( USStates )

# convert Region to a factor
USStates <- USStates |>
  mutate(Region = factor(Region))
```

3.3.2 Setting new baselines.

We can reorder the levels if desired (the first is our baseline).

```
levels( USStates$Region )
```

```
[1] "MW" "NE" "S" "W"
```

```
USStates$Region = relevel(USStates$Region, "S" )
levels( USStates$Region )
```

```
[1] "S" "MW" "NE" "W"
```

Now any regression will use the south as baseline.

3.3.3 Testing for significance of a categorical variable.

When deciding whether to keep a categorical variable, we need to test how important all the dummy variables for that category are to the model all at once. We do this with ANOVA. Here we examine whether region is useful for predicting the percent vote for Clinton in 2016:

```
mlm = lm( ClintonVote ~ Region, data=USStates)
anova( mlm )
```

Analysis of Variance Table

```
Response: ClintonVote
          Df Sum Sq Mean Sq F value    Pr(>F)
Region      3   1643     548     6.99 0.00057 ***
Residuals  46   3603      78
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It is quite important.

We can also compare for region beyond some other variable:

```

m1m2 = lm( ClintonVote ~ HouseholdIncome + HouseholdIncome + HighSchool +
           EighthGradeMath, data=USStates)

m1m3 = lm( ClintonVote ~ HouseholdIncome + HouseholdIncome + HighSchool +
           EighthGradeMath + Region, data=USStates)
anova( m1m2, m1m3 )

```

Analysis of Variance Table

```

Model 1: ClintonVote ~ HouseholdIncome + HouseholdIncome + HighSchool +
          EighthGradeMath
Model 2: ClintonVote ~ HouseholdIncome + HouseholdIncome + HighSchool +
          EighthGradeMath + Region
Res.Df   RSS Df Sum of Sq    F Pr(>F)
1      46 3287
2      43 2649  3       638 3.45  0.025 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Region is still important, beyond including some further controls. Interpreting this mess of a regression is not part of this document; this document shows you how to run regressions but it doesn't discuss whether you should or not.

3.3.4 Missing levels in a factor

R often treats categorical variables as factors. This is often useful, but sometimes annoying. A factor has different **levels** which are the different values it can be. For example:

```

data(FishGills3)
levels(FishGills3$Calcium)

```

```
[1] ""        "High"     "Low"      "Medium"
```

```
table(FishGills3$Calcium)
```

	High	Low	Medium
0	30	30	30

Note the weird nameless level; it also has no actual observations in it. Nevertheless, if you make a boxplot, you will get an empty plot in addition to the other three. This error was likely due to some past data entry issue. You can drop the unused level:

```
FishGills3$Calcium = droplevels(FishGills3$Calcium)
```

You can also turn a categorical variable into a numeric one like so:

```
summary( FishGills3$Calcium )
```

High	Low	Medium
30	30	30

```
asnum = as.numeric( FishGills3$Calcium )  
asnum
```

Regression on only a categorical variable is fine:

```
mylm = lm( GillRate ~ Calcium, data=FishGills3 )
mylm
```

```
Call:  
lm(formula = GillRate ~ Calcium, data = FishGills3)
```

Coefficients:

(Intercept) CalciumLow CalciumMedium
58.2 10.3 0.5

R has made you a bunch of dummy variables automatically. Here “high” is the baseline, selected automatically. We can also force it so there is no baseline by removing the intercept, in which case the coefficients are the means of each group.

```
mymm = lm( GillRate ~ 0 + Calcium, data=FishGills3 )
mymm
```

```

Call:
lm(formula = GillRate ~ 0 + Calcium, data = FishGills3)

Coefficients:
          CalciumHigh    CalciumLow   CalciumMedium
                      58.2        68.5        58.7

```

3.4 Some extensions (optional)

3.4.1 Confidence Intervals

To get confidence intervals around each parameter in your model, try this:

```
confint(tip.mod)
```

	2.5 %	97.5 %
(Intercept)	-0.655	0.146
Bill	0.166	0.200
Guests	-0.236	0.170
Credity	-0.319	0.403

You can also create them easily using `tidy` and `mutate`:

```

tip.mod |>
  tidy() |>
  mutate(upper = estimate + 1.96*std.error,
        lower = estimate - 1.96*std.error)

```

# A tibble: 4 x 7	term	estimate	std.error	statistic	p.value	upper	lower
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
1 (Intercept)	-0.255	0.203	-1.26	2.11e- 1	0.143	-0.652	
2 Bill	0.183	0.00846	21.6	2.07e-48	0.200	0.166	
3 Guests	-0.0332	0.103	-0.323	7.47e- 1	0.168	-0.235	
4 Credity	0.0422	0.183	0.231	8.18e- 1	0.400	-0.316	

3.4.2 Prediction

Suppose a server at this bistro is about to deliver a \$20 bill, and wants to predict their tip. They can get a predicted value and 95% (this is the default level, change with level) prediction interval with

```
new.dat = data.frame( Bill = c(20) )
predict(mod,new.dat,interval = "prediction")
```

```
fit lwr upr
1 3.35 1.41 5.29
```

They should expect a tip somewhere between \$1.41 and \$5.30.

If we know a bit more we can use our more complex model called `tip.mod` from above:

```
new.dat = data.frame( Bill = c(20), Guests=c(1), Credit=c("n") )
predict(tip.mod,new.dat,interval = "prediction")
```

```
fit lwr upr
1 3.37 1.41 5.34
```

This is the predicted tip for one guest paying with cash for a \$20 tip. It is wider than our original interval because our model is a bit more unstable (it turns out guest number and credit card aren't that relevant or helpful).

Compare the prediction interval to the confidence interval

```
new.dat = data.frame( Bill = c(20), Guests=c(1), Credit=c("n") )
predict(tip.mod, new.dat, interval = "confidence")
```

```
fit lwr upr
1 3.37 3.09 3.65
```

This predicts the mean tip for all single guests who pay a \$20 bill with cash. Our interval is smaller because we are generating a confidence interval for where the mean is, and are ignoring that individuals will vary around that mean. Confidence intervals are different from prediction intervals.

3.4.3 Removing Outliers

If you can identify which rows the outliers are on, you can do this by hand (say the rows are 5, 10, 12).

```
new.data = old.data[ -c(5,10,12), ]  
lm( Y ~ X, data=new.data )
```

Some technical details: The `c(5,10,12)` is a list of 3 numbers. The `c()` is the concatenation function that takes things makes lists out of them. The “-list” notation means give me my old data, but without rows 5, 10, and 12. Note the comma after the list. This is because we identify elements in a dataframe with row, column notation. So `old.data[1,3]` would be row 1, column 3.

If you notice your points all have X bigger than some value, say 20.5, you could use filtering to keep everything less than some value:

```
new.data = filter( old.data, X <= 20.5 )
```

3.4.4 Missing data

If you have missing data, `lm` will automatically drop those cases because it doesn’t know what else to do. It will tell you this, however, with the `summary` command.

```
data(AllCountries)  
dev.lm = lm( BirthRate ~ Rural + Health + ElderlyPop, data=AllCountries )  
summary( dev.lm )
```

Call:

```
lm(formula = BirthRate ~ Rural + Health + ElderlyPop, data = AllCountries)
```

Residuals:

Min	1Q	Median	3Q	Max
-16.592	-3.728	-0.791	3.909	16.218

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	26.5763	1.6795	15.82	< 2e-16 ***
Rural	0.0985	0.0224	4.40	1.9e-05 ***
Health	-0.0995	0.0930	-1.07	0.29

```

ElderlyPop   -1.0249      0.0881  -11.64  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

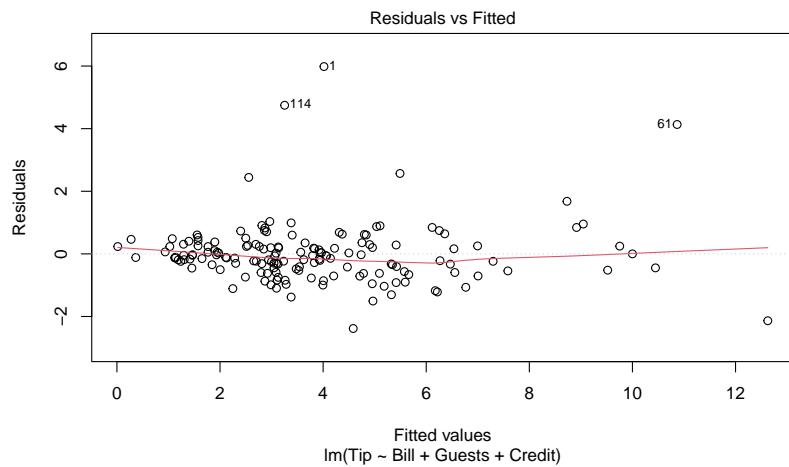
Residual standard error: 5.83 on 174 degrees of freedom
(39 observations deleted due to missingness)
Multiple R-squared:  0.663, Adjusted R-squared:  0.657
F-statistic: 114 on 3 and 174 DF,  p-value: <2e-16

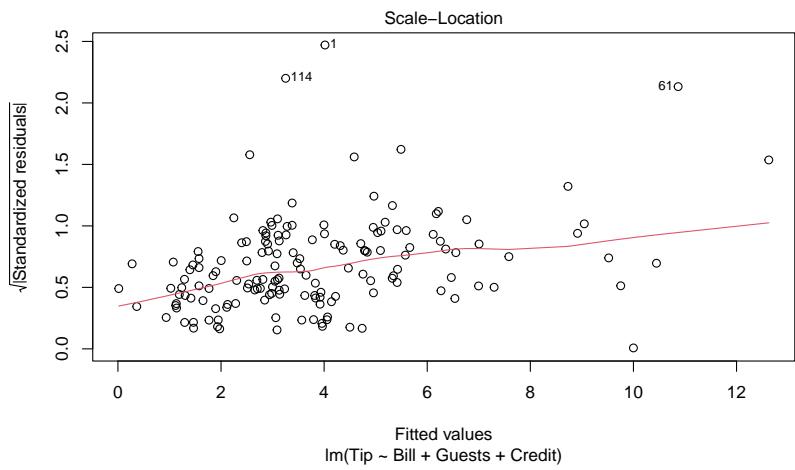
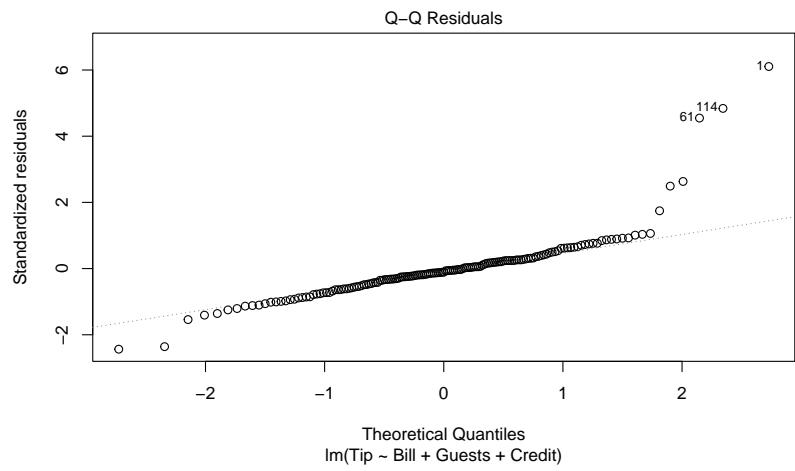
```

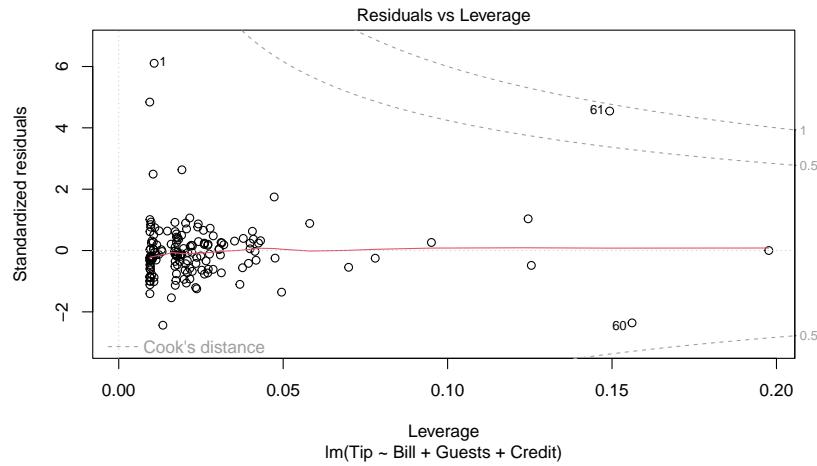
3.4.5 Residual plots and model fit

If we throw out model into the `plot` function, we get some nice regression diagnostics.

```
plot(tip.mod)
```



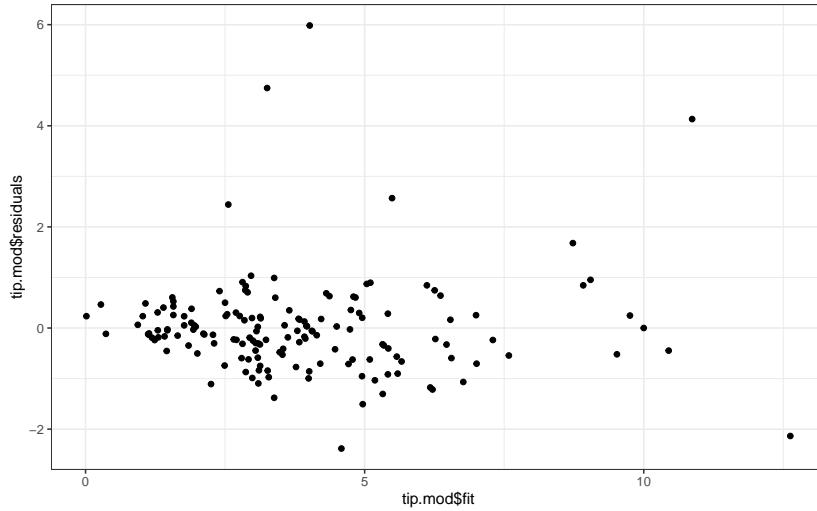




To generate classic model fit diagnostics with more control, we need to calculate residuals, make a residual versus fitted values plot, and make a histogram of the residuals. We can make some quick and dirty plots with `qplot` (standing for “quick plot”) like so:

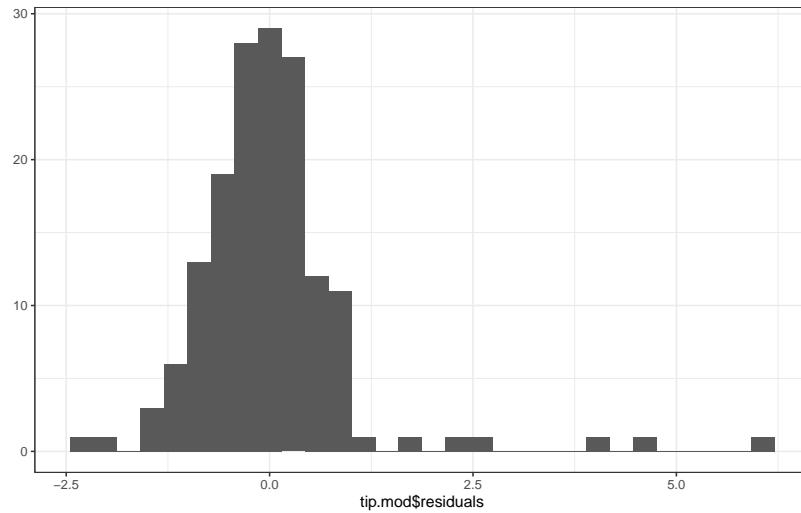
```
qplot(tip.mod$fit, tip.mod$residuals )
```

Warning: `qplot()` was deprecated in ggplot2 3.4.0.



and

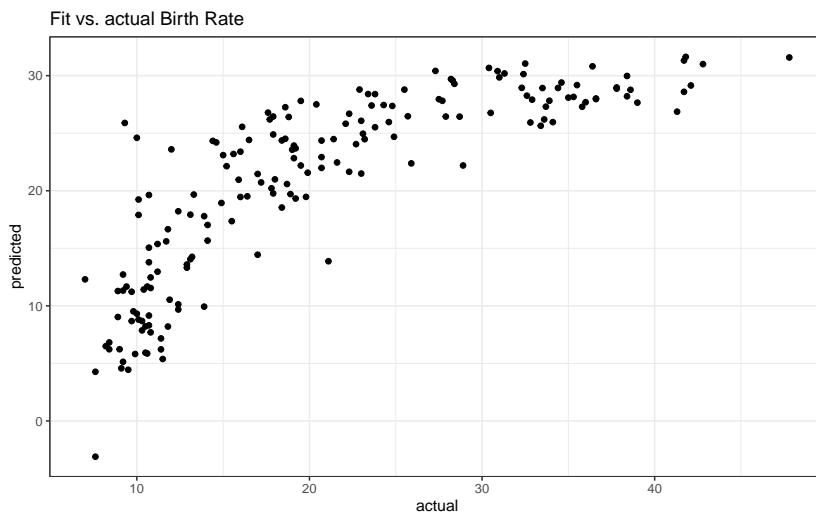
```
qplot(tip.mod$residuals, bins=30)
```



We see no real pattern other than some extreme outliers. The residual histogram suggests we are not really normally distributed, so we should treat our SEs and p -values with caution. These plots are the canonical “model-checking” plots you might use.

Another is the “fitted outcomes vs. actual outcomes” plot of:

```
predicted = predict( dev.lm )
actual = dev.lm$model$BirthRate
qplot( actual, predicted, main="Fit vs. actual Birth Rate" )
```



Note the `dev.lm` variable has a `model` variable inside it. This is a data frame of the **used** data for the model (i.e., if cases were dropped due to missingness, they will not be in the model). We then grab the birth rates from this, and make a scatterplot. If we tried to skip this, and use the original data, we would get an error because our original data set has some observations that were dropped.

Note we can't just add our predictions to `AllCountries` since we would get an error due to this dropped data issue:

```
AllCountries$predicted = predict( dev.lm )
```

```
Error in `$<-data.frame`(`*tmp*`, predicted, value = c(`1` = 31.630301617421, :  
replacement has 179 rows, data has 217
```

We can, however, predict like this:

```
AllCountries$predicted = predict( dev.lm, newdata=AllCountries )
```

The `newdata` tells `predict` to generate a prediction for each row in `AllCountries` rather than each row in the left over data after `lm` dropped cases with missing values.

4 Simple tables, plots, and model diagnostics

In this document we give a few simple plots and summary tables that may be useful for final projects and other things as well. This includes a few simple model diagnostic plots to check for extreme outliers and whatnot.

It is a bit of a hodge-podge, but skimming to get some ideas is definitely worthwhile.

4.1 National Youth Survey Example

Our running example is the National Youth Survey (NYS) data as described in Raudenbush and Bryk, page 190. This data comes from a survey in which the same students were asked yearly about their acceptance of 9 “deviant” behaviors (such as smoking marijuana, stealing, etc.). The study began in 1976, and followed two cohorts of children, starting at ages 11 and 14 respectively. We will analyze the first 5 years of data.

At each time point, we have measures of:

- ATTIT, the attitude towards deviance, with higher numbers implying higher tolerance for deviant behaviors.
- EXPO, the “exposure”, based on asking the children how many friends they had who had engaged in each of the “deviant” behaviors.

Both of these variables have been transformed to a logarithmic scale to reduce skew.

For each student, we have:

- Gender (binary)
- Minority status (binary)
- Family income, in units of \$10K (this can be either categorical or continuous).

4.1.1 Getting the data ready

We'll focus on the first cohort, from ages 11-15. First, let's read the data. Note that this data frame is in “wide format”. That is, there is only one row for each student, with all the different observations for that student in different columns of that one row.

```

nyswide <- read_csv("data/nyswide.csv")
head(nyswide)

# A tibble: 6 x 14
  ID ATTIT.11 EXP0.11 ATTIT.12 EXP0.12 ATTIT.13 EXP0.13 ATTIT.14 EXP0.14
<dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
1     3      0.11   -0.37     0.2     -0.27      0     -0.37      0     -0.27
2     8      0.29    0.42     0.29     0.2      0.11     0.42     0.51     0.2
3     9      0.8     0.47     0.58     0.52      0.64     0.2      0.75     0.47
4    15      0.44    0.07     0.44     0.32      0.89     0.47     0.75     0.26
5    33      0.2     -0.27    0.64     -0.27      0.69     -0.27     NA      NA
6    45      0.11    0.26     0.37     -0.17      0.37     0.14     0.37     0.14
# i 5 more variables: ATTIT.15 <dbl>, EXP0.15 <dbl>, FEMALE <dbl>,
# MINORITY <dbl>, INCOME <dbl>

```

For our purposes, we want it in “long format”, i.e. each student has multiple rows for the different observations. The `pivot_longer()` command does this for us.

```

nys1 <- nyswide |>
  pivot_longer(ATTIT.11:EXP0.15, names_to = "score") |>
  mutate(outcome = word(score, 1, 1, sep = "\\".), 
         age = as.numeric(word(score, 2, 2, sep = "\\".)), 
         age_fac = factor(age)) |>
  select(-score) |>
  pivot_wider(names_from = outcome) |>
  # drop missing ATTIT values
  drop_na(ATTIT)

head( nys1 )

```

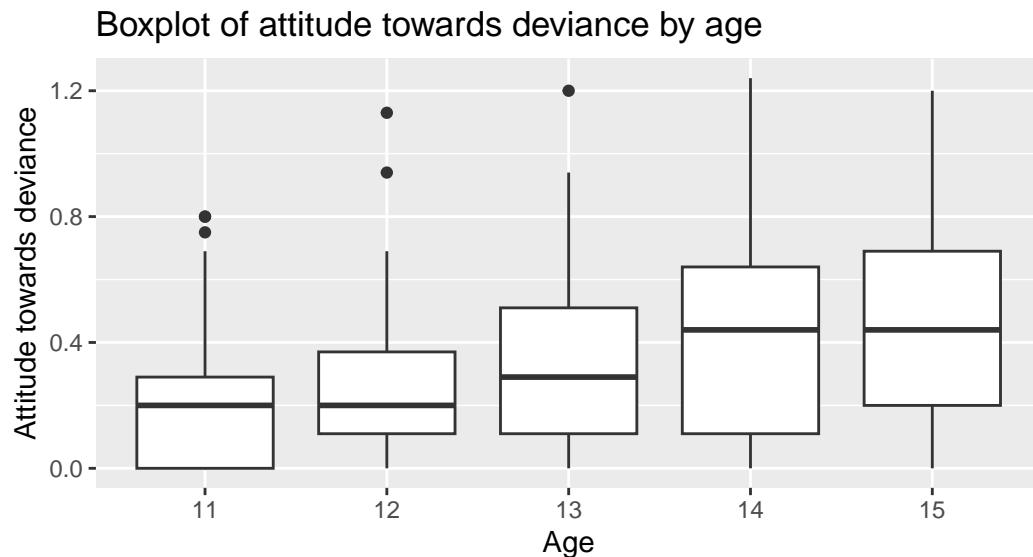
```

# A tibble: 6 x 8
  ID FEMALE MINORITY INCOME    age age_fac ATTIT    EXP0
<dbl>  <dbl>    <dbl>  <dbl> <dbl> <fct>  <dbl>  <dbl>
1     3      1      0      3     11  11     0.11  -0.37
2     3      1      0      3     12  12     0.2    -0.27
3     3      1      0      3     13  13     0     -0.37
4     3      1      0      3     14  14     0     -0.27
5     3      1      0      3     15  15     0.11  -0.17
6     8      0      0      4     11  11     0.29   0.42

```

Just to get a sense of the data, let’s plot each age as a boxplot

```
ggplot(nys1, aes(age_fac, ATTIT)) +
  geom_boxplot() +
  labs(title = "Boxplot of attitude towards deviance by age",
       x = "Age", y = "Attitude towards deviance")
```



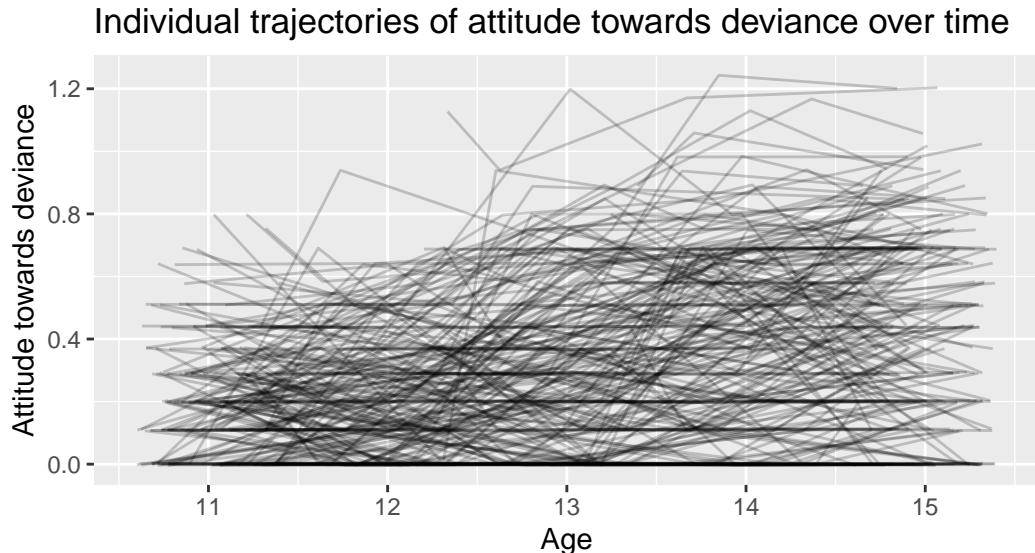
Note: The boxplot's “x” variable is the group. You get one box per group. The “y” variable is the data we are making boxplots of.

Note some features of the data:

- First, we see that ATTIT goes up over time.
- Second, we see the variation of points also goes up over time. This is evidence of heteroskedasticity.

If we plot individual lines we have:

```
nys1 |>
  drop_na() |>
  ggplot(aes(age, ATTIT, group=ID)) +
  geom_line(alpha=0.2, position = "jitter") +
  labs(title = "Individual trajectories of attitude towards deviance over time",
       x = "Age",
       y ="Attitude towards deviance")
```



Note how we have correlation of residuals: some students have systematically lower trajectories and some students have systematically higher trajectories (although there is a lot of bouncing around).

4.2 Tabulating data (Categorical variables)

We can tabulate data as so:

```
table(nys1$age)
```

```
11 12 13 14 15
202 209 230 220 218
```

or

```
table(nys1$MINORITY, nys1$age)
```

```
11 12 13 14 15
0 159 165 182 175 175
1 43 44 48 45 43
```

Interestingly, we have more observations for later ages.

We can make “proportion tables” as well:

```
prop.table( table( nys1$MINORITY, nys1$INCOME ), margin=1 )
```

	1	2	3	4	5	6	7	8	9
0	0.06075	0.13551	0.18341	0.18107	0.14369	0.10981	0.06893	0.05257	0.00935
1	0.28251	0.41704	0.12556	0.05830	0.05830	0.02242	0.01345	0.00000	0.00000

	10
0	0.05491
1	0.02242

The margin determines what adds up to 100%.

4.3 Summary stats (continuous variables)

The `tableone` package is useful:

```
library(tableone)

# sample mean
CreateTableOne(data = nys1,
               vars = c("ATTIT"))


```

Overall	
n	1079
ATTIT (mean (SD))	0.33 (0.27)


```
# you can also stratify by a variables of interest
CreateTableOne(data = nys1,
               vars = c("ATTIT"),
               strata = c("FEMALE"))
```

Stratified by FEMALE			
		p	test
n	559 520		
ATTIT (mean (SD))	0.37 (0.27) 0.29 (0.27)	<0.001	

```
# you can also include both binary variables
CreateTableOne(data = nys1,
               vars = c("ATTIT", "agefac"), # include both binary and continuous variables
               factorVars = c("agefac"), # include only binary variables here
               strata = c("FEMALE"))
```

Warning in ModuleReturnVarsExist(vars, data): The data frame does not have:
agefac Dropped

Warning in ModuleReturnVarsExist(factorVars, data): The data frame does not have: agefac Dropped

Stratified by FEMALE			
	0	1	p test
n	559	520	
ATTIT (mean (SD))	0.37 (0.27)	0.29 (0.27)	<0.001

4.4 Table of summary stats

You can easily make pretty tables using the `stargazer` package:

```
library(stargazer)
```

Please cite as:

Hlavac, Marek (2022). `stargazer`: Well-Formatted Regression and Summary Statistics Tables.

R package version 5.2.3. <https://CRAN.R-project.org/package=stargazer>

```
# to include all variables
stargazer(nys1, header = FALSE)
```

You can include only some of the variables and omit stats that are not of interest:

```
# to include only variables of interest
stargazer(nys1[2:7], header=FALSE,
          omit.summary.stat = c("p25", "p75", "min", "max"), # to omit percentiles
          title = "Table 1: Descriptive statistics")
```

Table 4.1

Statistic	N	Mean	St. Dev.	Min	Max
-----------	---	------	----------	-----	-----

Table 4.2: Table 1: Descriptive statistics

Statistic	N	Mean	St. Dev.
-----------	---	------	----------

See the `stargazer` help file for how to set/change more of the options: <https://cran.r-project.org/web/packages/stargazer/stargazer.pdf>

4.5 High School and Beyond Example

For this part, we'll use the HSB data to summarize variables by group/school.

```
# load data
dat <- read_dta("data/hsb.dta")
```

4.6 Summarizing by group

To plot summaries by group, first aggregate your data, and plot the results. Like so:

```
aggdat = dat %>%
  group_by(schoolid, sector) %>%
  summarise( per.fem = mean( female ) )

`summarise()` has grouped output by 'schoolid'. You can override using the
`.groups` argument.

head( aggdat )

# A tibble: 6 x 3
# Groups:   schoolid [6]
  schoolid sector per.fem
    <dbl>   <dbl>   <dbl>
```

```

1   1224      0   0.596
2   1288      0   0.44
3   1296      0   0.646
4   1308      1   0
5   1317      1   1
6   1358      0   0.367

```

The including sector (a level 2 variable) is a way to ensure it gets carried through to the aggregated results. Neat trick.

Anyway, we then plot:

```

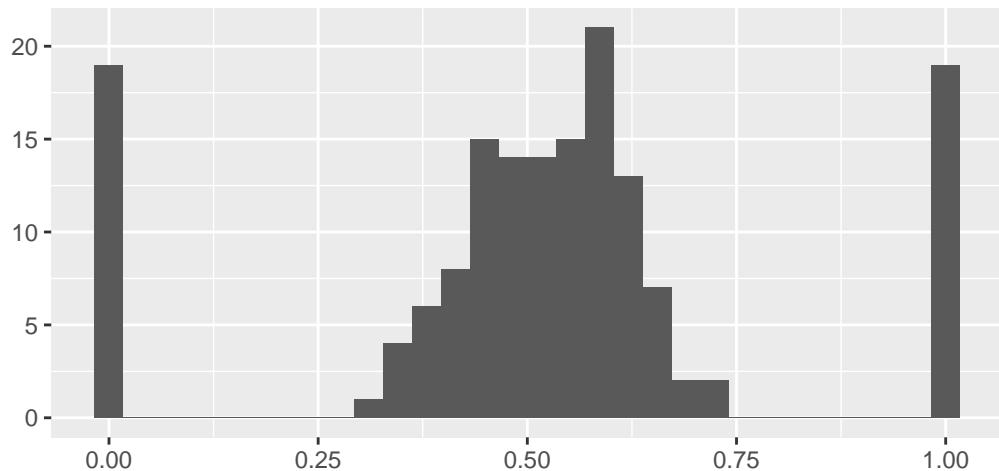
qplot( aggdat$per.fem,
       main = "Percent female students",
       xlab = "")

```

Warning: `qplot()` was deprecated in ggplot2 3.4.0.

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

Percent female students



Note the single sex (catholic) schools. We can facet to see both groups:

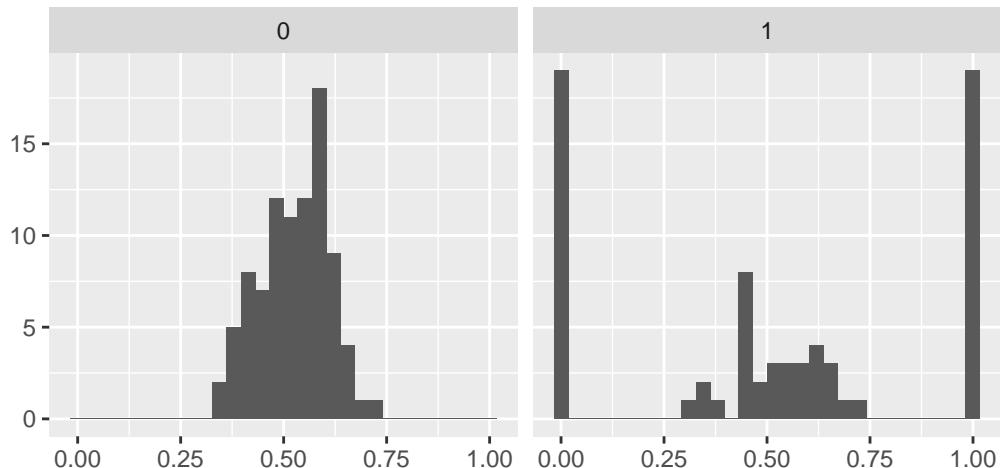
```

qplot( per.fem, data=aggdat,
       main = "Percent female students",
       xlab = "") +
  facet_wrap( ~ sector )

```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

Percent female students



4.7 Diagnostic plots

We can also make some diagnostic plots for our model. first, let's fit a random intercept model.

```

m1 <- lmer(mathach ~ 1 + ses + (1|schoolid), data=dat)
arm::display(m1)

```

```

lmer(formula = mathach ~ 1 + ses + (1 | schoolid), data = dat)
      coef.est coef.se
(Intercept) 12.66     0.19
ses          2.39     0.11

Error terms:
Groups   Name        Std.Dev.

```

```

schoolid (Intercept) 2.18
Residual             6.09
---
number of obs: 7185, groups: schoolid, 160
AIC = 46653.2, DIC = 46637
deviance = 46641.0

```

We can check if some of our assumptions are being grossly violated, i.e. residuals at all levels are normally distributed.

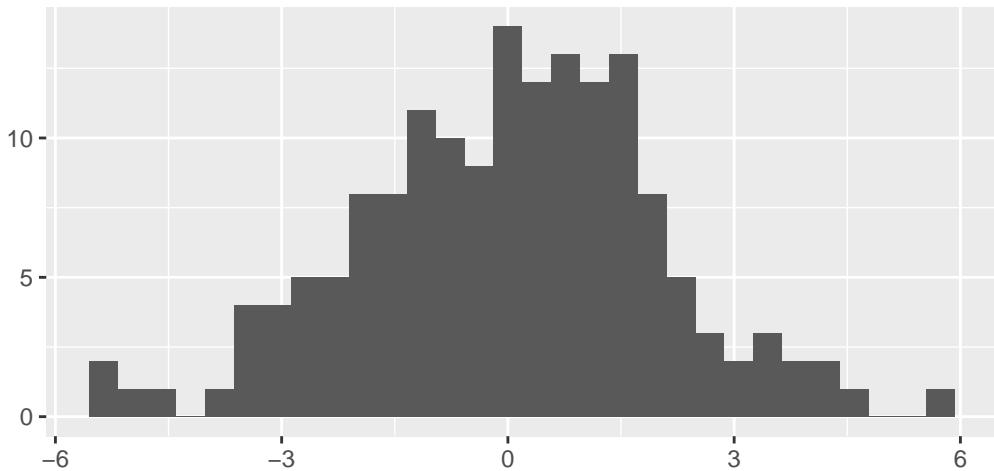
```

qplot(ranef(m1)$schoolid[,1],
      main = "Histogram of random intercepts", xlab="")

```

``stat_bin()` using `bins = 30`.` Pick better value with ``binwidth``.

Histogram of random intercepts



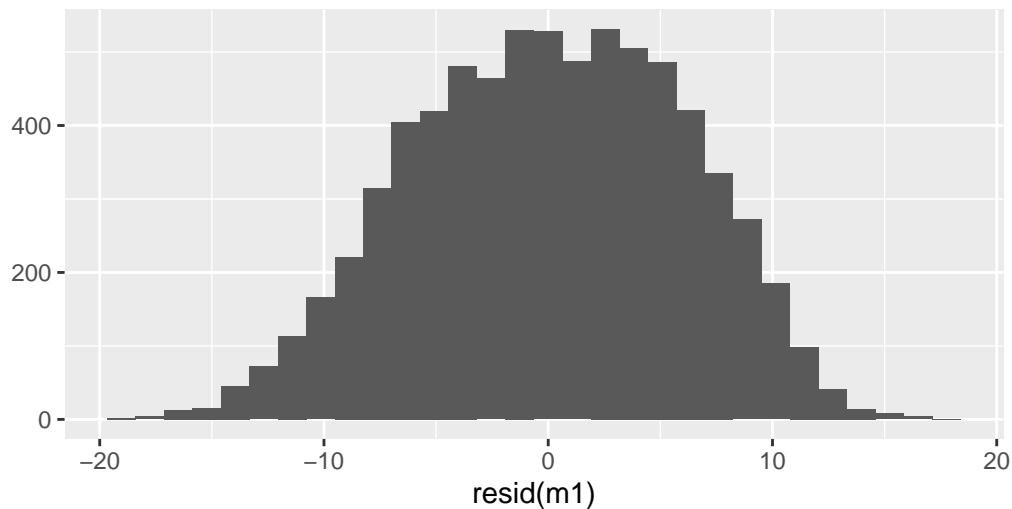
```

qplot(resid(m1),
      main = "Histogram of residuals")

```

``stat_bin()` using `bins = 30`.` Pick better value with ``binwidth``.

Hisogram of residuals



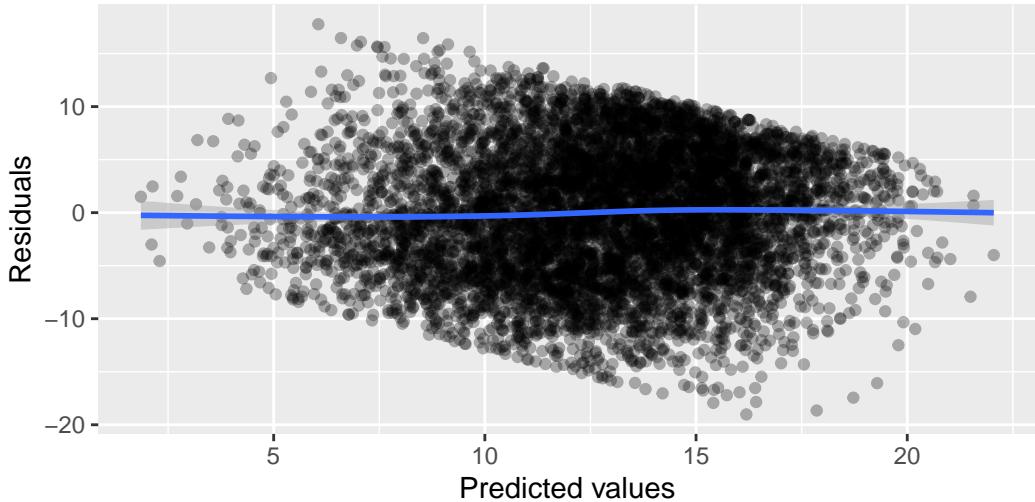
We can check for heteroskedasticity by plotting residuals against predicted values

```
dat$yhat = predict(m1)
dat$resid = resid(m1)

ggplot(dat, aes(yhat, resid)) +
  geom_point(alpha=0.3) +
  geom_smooth() +
  labs(title = "Residuals against predicted values",
       x = "Predicted values", y ="Residuals")

`geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```

Residuals against predicted values

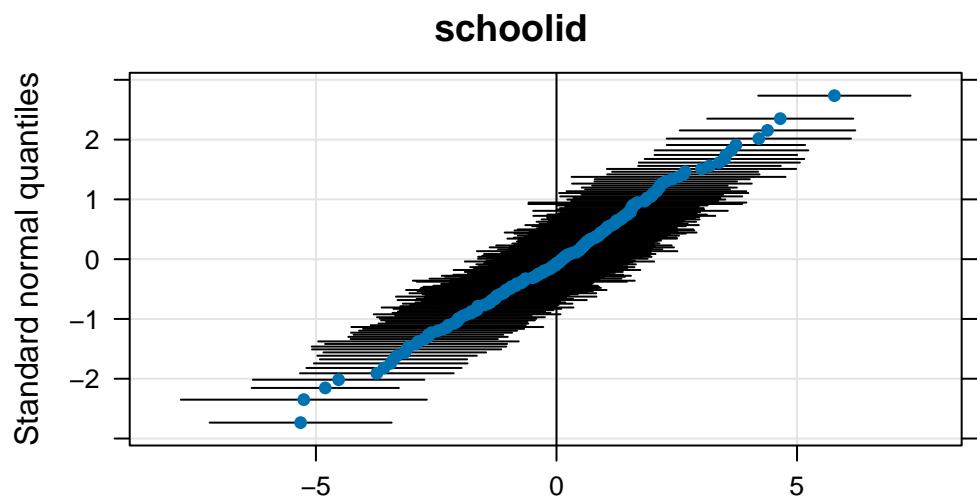


It looks reasonable (up to the discrete and bounded nature of our data). No major weird curves in the loess line through the residuals means linearity is a reasonable assumption. That being said, our nominal SEs around our loess line are tight, so the mild curve is probably evidence of *some* model misfit.

We can also look at the distribution of random effects using the `lattice` package

```
library(lattice)
qqmath(ranef(m1, condVar=TRUE), strip=FALSE)
```

```
$schoolid
```



5 Making tables in Markdown

You might want to make tables. Usually you should probably make charts instead, but every so often a table is a nice thing to have. This chapter is about making generic tables. For regression tables, see Chapter ??.

To illustrate, I make some fake data

```
library( tidyverse )
dat = tibble( G = sample( LETTERS[1:5], 100, replace=TRUE ),
              X = rnorm( 100 ),
              rp = sample( letters[1:3], 100, replace=TRUE ),
              Z = sample( c("tx","co"), 100, replace=TRUE ),
              Y = rnorm( 100 ) )
```

We can make summary of it by our grouping variable:

```
sdat <- dat %>% group_by( G ) %>%
  summarise( EY = mean( Y ),
             pT = mean( Z == "tx" ),
             sdY = sd( Y ) )
```

Our intermediate results:

```
sdat
```



```
# A tibble: 5 x 4
  G       EY     pT    sdY
  <chr>   <dbl> <dbl> <dbl>
1 A      -0.406  0.684 1.34
2 B      0.320   0.476 1.04
3 C      0.0159  0.471 1.09
4 D     -0.0101  0.684 0.911
5 E      0.0167  0.667 0.929
```

Say our grouping variable is a set of codes for something more special. We can merge in better names by first making a small “cross-walk” of the ID codes to the full names, and then merging them to our results:

```
names = tribble( ~ G, ~ name,
                  "A", "fred",
                  "B", "doug",
                  "C", "xiao",
                  "D", "lily",
                  "E", "unknown" )

names
```

```
# A tibble: 5 x 2
  G     name
  <chr> <chr>
1 A     fred
2 B     doug
3 C     xiao
4 D     lily
5 E     unknown
```

```
sdat = left_join( sdat, names ) %>%
    relocate( name)
```

Joining with `by = join_by(G)`

Finally, the easiest way to make a table is with the `kable` command.

```
knitr::kable( sdat, digits=2 )
```

	name	G	EY	pT	sdY
1	fred	A	-0.41	0.68	1.34
2	doug	B	0.32	0.48	1.04
3	xiao	C	0.02	0.47	1.09
4	lily	D	-0.01	0.68	0.91
5	unknown	E	0.02	0.67	0.93

This is a great workhorse table-making tool! There are expansion R packages as well, e.g. `kableExtra`, which can do lots of fancy customizable stuff.

5.1 Making a “table one”

The “table one” is the first table in a lot of papers that show general means of different variables for different groups. The `tableone` package is useful:

```
library(tableone)

# sample mean
CreateTableOne(data = dat,
               vars = c("G", "Z", "X"))
```

Overall	
n	100
G (%)	
A	19 (19.0)
B	21 (21.0)
C	17 (17.0)
D	19 (19.0)
E	24 (24.0)
Z = tx (%)	60 (60.0)
X (mean (SD))	0.08 (1.03)

```
# you can also stratify by a variables of interest
tb <- CreateTableOne(data = dat,
                      vars = c("X", "G", "Y"),
                      strata = c("Z"))
tb
```

Stratified by Z				
	co	tx	p	test
n	40	60		
X (mean (SD))	0.02 (0.96)	0.13 (1.08)	0.606	
G (%)			0.393	
A	6 (15.0)	13 (21.7)		
B	11 (27.5)	10 (16.7)		
C	9 (22.5)	8 (13.3)		
D	6 (15.0)	13 (21.7)		
E	8 (20.0)	16 (26.7)		
Y (mean (SD))	0.09 (1.03)	-0.07 (1.10)	0.456	

You can then use `kable` as so:

```
print(tb$ContTable, printToggle = FALSE) %>%
  knitr::kable()
```

	co	tx	p	test
n	40		60	
X (mean (SD))	0.02 (0.96)	0.13 (1.08)		0.606
Y (mean (SD))	0.09 (1.03)	-0.07 (1.10)		0.456

5.2 Table of summary stats

You can also easily make pretty tables using the `stargazer` package. You need to ensure the data is a `data.frame`, not `tibble`, because `stargazer` is old school. It appears to only do continuous variables.

Finally, you need to modify the R code chunk so it looks like this:

so the output of `stargazer` gets formatted properly in your R Markdown.

```
library(stargazer)

stargazer(as.data.frame(dat))
```

% Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail:
marek.hlavac at gmail.com % Date and time: Sun, Sep 10, 2023 - 22:29:36

Table 5.3

Statistic	N	Mean	St. Dev.	Min	Max
X	100	0.084	1.027	-2.822	2.393
Y	100	-0.005	1.068	-2.783	2.693

You can include only some of the variables and omit stats that are not of interest:

```
# to include only variables of interest
stargazer(as.data.frame(dat), header=FALSE,
          omit.summary.stat = c("p25", "p75", "min", "max"), # to omit percentiles
          title = "Table 1: Descriptive statistics")
```

Table 5.4: Table 1: Descriptive statistics

Statistic	N	Mean	St. Dev.
X	100	0.084	1.027
Y	100	-0.005	1.068

See the **stargazer** help file for how to set/change more of the options: <https://cran.r-project.org/web/packages/stargazer/stargazer.pdf>

6 Making Regression Tables

For publication-ready tables and graphics, R has many wonderful packages to automate the process. Some of these packages are good for html documents (websites) and some are good for making PDF documents (reports and papers). This document demonstrates different ways of making regression tables in your reports, and talks about some weird wrinkles with using them with multilevel modeling.

6.1 The basics of regression tables

For the basics we quickly illustrate regression tables using a subset of the Making Caring Common dataset, which we will eventually discuss in class. This dataset has a measure of emotional safety (our outcome) and we want to see, in a specific school, if this is predicted by gender and/or grade.

Our data look like this:

```
sample_n( sch1, 6 )
```

	ID	esafe	grade	gender	disc	race_white
1	1	3.428571	8	Male	1.444444	1
2	1	4.000000	5	Male	1.000000	1
3	1	3.857143	7	Female	1.888889	1
4	1	2.428571	7	Female	2.222222	1
5	1	4.000000	5	Female	1.000000	1
6	1	4.000000	7	Female	1.000000	1

We fit some models:

```
M_A = lm( esafe ~ grade, data = sch1 )
M_B = lm( esafe ~ grade + gender, data = sch1 )
M_C = lm( esafe ~ grade * gender, data = sch1 )
```

Ok, we have fit our regression models. We can look at big complex printout of a single model like so:

```
summary( M_C )

Call:
lm(formula = esafe ~ grade * gender, data = sch1)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.7894 -0.1570  0.1550  0.2662  0.4938 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 4.12733   0.37946 10.877 <2e-16 ***
grade        -0.07764   0.05859 -1.325  0.1879  
genderMale   -0.72735   0.49762 -1.462  0.1467  
grade:genderMale 0.13327   0.07627  1.747  0.0834 .  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 0.4333 on 108 degrees of freedom
Multiple R-squared: 0.04914, Adjusted R-squared: 0.02273
F-statistic: 1.861 on 3 and 108 DF, p-value: 0.1406

Or we can make *regression tables*. Consider these two packages, the first being `texreg`

```
library( texreg )
screenreg(list(M_A, M_B, M_C))
```

	Model 1	Model 2	Model 3
(Intercept)	3.68 *** (0.25)	3.62 *** (0.25)	4.13 *** (0.38)
grade	0.00 (0.04)	0.00 (0.04)	-0.08 (0.06)
genderMale		0.13 (0.08)	-0.73 (0.50)
grade:genderMale			0.13 (0.08)

R^2	0.00	0.02	0.05
Adj. R^2	-0.01	0.00	0.02
Num. obs.	112	112	112

=====

*** p < 0.001; ** p < 0.01; * p < 0.05

Another is stargazer.

```
library( stargazer )
stargazer( M_A, M_B, M_C, header=FALSE, type='text')
```

Dependent variable:			
	esafe		
	(1)	(2)	(3)
grade	0.004 (0.038)	0.001 (0.038)	-0.078 (0.059)
genderMale		0.130 (0.083)	-0.727 (0.498)
grade:genderMale			0.133* (0.076)
Constant	3.676*** (0.249)	3.624*** (0.250)	4.127*** (0.379)
Observations	112	112	112
R2	0.0001	0.022	0.049
Adjusted R2	-0.009	0.004	0.023
Residual Std. Error	0.440 (df = 110)	0.437 (df = 109)	0.433 (df = 108)
F Statistic	0.009 (df = 1; 110)	1.241 (df = 2; 109)	1.861 (df = 3; 108)

=====

Note: *p<0.1; **p<0.05; ***p<0.01

6.2 Extending to the multilevel model

For our multilevel examples, we use the Making Caring Common data from Project A, and fit data to the 8th grade students only, but do it for all schools. We have made a High School dummy variable.

Our two models we use for demo purposes have a HS term and no HS term:

```
modA <- lmer( esafe ~ 1 + (1 | ID), data=dat.g8)
modB <- lmer( esafe ~ 1 + HS + (1 | ID), data=dat.g8)
```

In the next sections we first show how to get better summary output (according to some folks) and then we walk through making regression tables in a bit more detail than above.

6.3 Getting p-values for lmer output

The `lmerTest` package is a way of making R give you more complete output. We are going to load it, and then put the new lmer models into new variables so we can see how the different model fitting packages work with the regression table packages below.

```
library( lmerTest )
modB.T <- lmer( esafe ~ 1 + HS + (1 | ID), data=dat.g8)
modA.T <- lmer( esafe ~ 1 + (1 | ID), data=dat.g8)

summary( modB.T )
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [  
lmerModLmerTest]  
Formula: esafe ~ 1 + HS + (1 | ID)  
Data: dat.g8
```

```
REML criterion at convergence: 2746.8
```

```
Scaled residuals:  
    Min     1Q   Median     3Q     Max  
-3.3883 -0.6156  0.2021  0.7628  1.7331
```

```
Random effects:  
Groups   Name        Variance Std.Dev.  
ID       (Intercept) 0.04809  0.2193
```

```

Residual           0.46459  0.6816
Number of obs: 1305, groups: ID, 26

Fixed effects:
            Estimate Std. Error      df t value Pr(>|t|)
(Intercept) 3.52798   0.08637 29.91033 40.846 <2e-16 ***
HTRUE       -0.29480   0.10787 25.77814 -2.733  0.0112 *
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
        (Intr)
HTRUE -0.801

```

6.4 The texreg package

In `texreg` there are two primary functions for table making, one is `screenreg()` and the other is `texreg()`.

6.4.1 Using `screenreg()`

Screenreg is fine for MLMs. It looks a bit like raw output, but it is clear and clean. It will take models fit using lmer or lmerTest, no problem.

```
screenreg(list(modA,modB))
```

	Model 1	Model 2
(Intercept)	3.35 *** (0.06)	3.53 *** (0.09)
HTRUE		-0.29 ** (0.11)
AIC	2756.78	2754.79
BIC	2772.30	2775.49
Log Likelihood	-1375.39	-1373.40
Num. obs.	1305	1305
Num. groups: ID	26	26

```

Var: ID (Intercept)      0.07      0.05
Var: Residual           0.46      0.46
=====
*** p < 0.001; ** p < 0.01; * p < 0.05

```

Comment: Note that the number of stars are different for the display vs the summary output! (Look at the HS coefficient for example.) Not good, it would seem.

This is because the p -values are calculated using the normal approximation by the `screenreg` command, and using the t -test with approximate degrees of freedom by `lmerTest`. This makes a difference. Consider the following, using the t statistics for the HS variable:

```
2 * pt( -2.733, df=25.77814 )
```

```
[1] 0.0111831
```

```
2 * pnorm( -2.733 )
```

```
[1] 0.006276033
```

One is below 0.01, and one is not. An extra star!

6.4.2 Using `texreg()` and TeX

The `texreg` command is part of the `texreg` package and can be integrated with `latex` (which you would need to install). Once you do this, when you compile to a pdf, all is well. In the R code chunk you need to include `results="asis"` to get the `latex` to compile right. E.g., “`r, results="asis"`” when you declare a code chunk.

```
texreg(list(modA,modB), table=FALSE)
```

	Model 1	Model 2
(Intercept)	3.35*** (0.06)	3.53*** (0.09)
HTRUE		-0.29** (0.11)
AIC	2756.78	2754.79
BIC	2772.30	2775.49
Log Likelihood	-1375.39	-1373.40
Num. obs.	1305	1305
Num. groups: ID	26	26
Var: ID (Intercept)	0.07	0.05
Var: Residual	0.46	0.46

*** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$

Note that the `table=FALSE` puts the table right where you want it, not at some random spot latex things is nice. Latex likes to have “floating tables,” where it puts the table where there is space; this makes it easier to make the entire formatted page look nice.

6.5 The stargazer package

```
library( stargazer )
stargazer(modA, modB, header=FALSE, type='latex')
```

One issue is stargazer does not include the random effect variances, so the output is quite limited for multilevel modeling. It also has less stringent conditions for when to put down stars. One star is below 0.10, two is below 0.05, and three is below 0.01. This is quite generous. Also it is using the normal approximation.

6.5.1 Stargazer with lmerTest

Stargazer with lmerTest is a bit fussy. This shows how to make it work if you have loaded the lmerTest package. Recall the lmerTest package makes your lmer commands have p-values and whatnot. But this means your new `lmer()` command is not quite the same as the old—and stargazer is expecting the old. You gix this by lying to R, telling it the new thing is the old thing. This basically works.

Now for stargazer, we need to tell it that our models are the right type. First note:

```
class( modB )
```

Table 6.1

<i>Dependent variable:</i>		
	esafe	
	(1)	(2)
HS		−0.295*** (0.108)
Constant	3.346*** (0.059)	3.528*** (0.086)
Observations	1,305	1,305
Log Likelihood	−1,375.388	−1,373.397
Akaike Inf. Crit.	2,756.775	2,754.795
Bayesian Inf. Crit.	2,772.297	2,775.491

Note: *p<0.1; **p<0.05; ***p<0.01

```
[1] "lmerMod"
attr(,"package")
[1] "lme4"
```

```
class( modB.T )
```

```
[1] "lmerModLmerTest"
attr(,"package")
[1] "lmerTest"
```

So we fix as follows:

```
library( stargazer )
class( modB.T ) = "lmerMod"
class( modA.T ) = "lmerMod"
stargazer(modA.T, modB.T, header=FALSE, type='latex' )
```

6.5.2 The sjPlot package

One function, `tab_model` from `sjPlot`, makes nice regression tables:

Table 6.2

<i>Dependent variable:</i>		
	esafe	
	(1)	(2)
HS		-0.295*** (0.108)
Constant	3.346*** (0.059)	3.528*** (0.086)
Observations	1,305	1,305
Log Likelihood	-1,375.388	-1,373.397
Akaike Inf. Crit.	2,756.775	2,754.795
Bayesian Inf. Crit.	2,772.297	2,775.491

Note: *p<0.1; **p<0.05; ***p<0.01

```
# tabulate the results of our two tip models
library( sjPlot )
tab_model(modA.T, modB.T)
```

esafe
esafe
Predictors
Estimates
CI
p
Estimates
CI
p
(Intercept)
3.35

3.23 – 3.46
<0.001
3.53
3.36 – 3.70
<0.001
HTRUE
-0.29
-0.51 – -0.08
0.006
Random Effects
2
0.46
0.46
00
0.07 ID
0.05 ID
ICC
0.12
0.09
N
26 ID
26 ID
Observations
1305
1305
Marginal R² / Conditional R²
0.000 / 0.125
0.028 / 0.119

7 Pretty ANOVA Tables with `kable`

7.1 R Setup

We load the `tidyverse` and `knitr`. The `kable` function from `knitr` makes our tables look nice!

```
library(tidyverse)
library(knitr)
library(broom)
```

7.2 Create fake data

We create a data set called `a` that has 100 observations and specifies our outcome `Y` as a function of two uncorrelated variables `A` and `B`

```
a <- tibble( A = rnorm( 100 ),
             B = rnorm( 100 ),
             Y = A * 0.2 + B * 0.5 + rnorm( 100, 0, 1 ) )
```

7.3 Run the Models

We fit two models, one with `A` and `B`, the other with just `A`.

```
M1 <- lm( Y ~ A + B, data = a )
M2 <- lm( Y ~ A, data = a )
```

7.4 Comparing the Models

We use the `anova` function to compare the two models (see also the chapter on Likelihood Ratio tests). We see that B improves the model fit significantly.

```
aa = anova( M2, M1 )
aa
```

Analysis of Variance Table

```
Model 1: Y ~ A
Model 2: Y ~ A + B
Res.Df   RSS Df Sum of Sq    F    Pr(>F)
1     98 128.50
2     97 105.64  1    22.859 20.989 1.374e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
aa |>
  tidy() |>
  kable()
```

	term	df.residual	rss	df	sumsq	statistic	p.value
	Y ~ A	98	128.5030	NA	NA	NA	NA
	Y ~ A + B	97	105.6436	1	22.8594	20.98907	1.37e-05

7.5 Compare to the Significance test on B

Note that the p value for B is identical to the ANOVA results above. Why bother with ANOVA? It can test more complex hypotheses as well (multiple coefficients, random effects, etc.)

```
M1 |>
  tidy() |>
  kable()
```

	term	estimate	std.error	statistic	p.value
	(Intercept)	-0.1743460	0.1053419	-1.6550482	0.1011469

	term	estimate	std.error	statistic	p.value
A		0.1048881	0.1076018	0.9747807	0.3320937
B		0.4397691	0.0959905	4.5813829	0.0000137

8 A Math Reference: Sample Modeling Equations to Borrow

8.1 Introduction

This document has a bunch of mathematical equations we use in the class. It is a good reference for how to write your own math equations in your life moving forward. Generally, people write math equations using something called Latex. Latex (or Tex) is a way of writing documents where mixed in with the writing of what you want to say are commands (editorial markup, if you will) describing how you want your document to look. This is a very powerful thing: there are Tex editors that allow you to write entire articles, books, reports, poetry, or whatever with extreme control over the typesetting used. It creates beautifully typeset documents that are easy to distinguish from those written in, say, MS Word due to how they adjust whitespace on the page. That being said, it can be a lot to jump in to.

Enter R Markdown. R Markdown is a useful and easy way to take advantage of this syntax without the overhead of writing entire documents in Latex, even if you don't have any R code in your document. Inside R Markdown you can write math equations, and then when you render the report, it not only runs all the R code, but it formats all the math for you as well! You can even have R Markdown render to MS Word to give you a word doc with all your math equations ready to go.

8.1.1 Using this document

You are probably reading the PDF version of this document. But really you should open the .Rmd file that generated this document, so you can cut and paste the relevant equations into your own work, and then modify as necessary. The link to this file [here](#).

8.2 Overview of Using Latex

For math in your writing, you denote the beginning and the end of a math equation in your text using “\$”s—one at the start and one at the stop. E.g., “\$ math stuff \$”. Most greek letters are written as their names with a backslash “\” just before it. E.g., “\alpha”.

So if I want to write an alpha, I write “\$\\alpha\$” and get α .

I can do subscripts by using an underscore. E.g., “\$\\alpha_j\$” gives α_j . I can also do superscripts by using a hat. E.g., “\$\\alpha^2\$” gives α^2 . To put more than one character in a subscript (or superscript), put the stuff to be subscripted in curly braces, e.g., “\$\\alpha_{ij}\$” gives α_{ij} .

8.2.1 Some useful greek letters

Here are some useful greek letters and symbols

Letter	Name
α	\alpha
β	\beta
δ, Δ	\delta, \Delta
ϵ	\epsilon
σ, Σ	\sigma, \Sigma
ρ	\rho
μ	\mu (Meew!)
τ	\tau
\times	\times
\sim	\sim

See many more symbols at, e.g., <https://www.caam.rice.edu/~heinken/latex/symbols.pdf>. This was found by searching “tex symbols” on Google.

8.2.2 Equations on lines by themselves

To write an equation on a line by itself, put the math stuff in between a pair of double “\$”. E.g., if we write:

```
$$ Y = a X + b $$
```

We get

$$Y = aX + b.$$

If we want multiple lines, we have to put our equation between a `\begin{aligned}` and `\end{aligned}` command and use a double backslash (“\\”) to denote each line break (even if we have a line break we have to do this—we have to explicitly tell the program converting our raw text to nice formatted text where the line breaks are). Finally, inside the begin-end block of math, line things up with & symbols on each row of our equation. The & symbols will be lined up vertically.

So if we write

```
$$
\begin{aligned}
Y &= 10 X + 2 \\
Y - 5 &= 3 X^2 + 5
\end{aligned}
$$
```

we get

$$Y = 10X + 2$$

$$Y - 5 = 3X^2 + 5$$

Also consider:

```
$$
\begin{aligned}
a + b + c + d &= c \\
d &= e + f + g + h
\end{aligned}
$$
```

giving

$$a + b + c + d = c$$

$$d = e + f + g + h$$

8.2.3 Normal text in equations

If you put words in your equations, they get all italicized and weird, without their spaces:

```
$$
5 + \text{my dog} = 10
$$
```

$$5 + mydog = 10$$

You can fix using the “\mbox{}” command as so:

```
$$
5 + \mbox{my dog} = 10
$$
```

$$5 + \text{my dog} = 10$$

We next walk through some latex code for the models you will most see.

8.3 Sample code: Random Intercept Model

Our canonical Random Intercept model is as follows. First, our Level 1 model:

```
$$
\begin{aligned}
y_{ij} &= \alpha_j + \beta_1 ses_{ij} + \epsilon_{ij} \\
\epsilon_{ij} &\sim N(0, \sigma^2_y) \\
\end{aligned}
$$
```

$$\begin{aligned} y_{ij} &= \alpha_j + \beta_1 ses_{ij} + \epsilon_{ij} \\ \epsilon_{ij} &\sim N(0, \sigma^2_y) \end{aligned}$$

Our Level 2 model:

```
$$
\begin{aligned}
\alpha_j &= \gamma_0 + \gamma_1 sector_j + u_j \\
u_j &\sim N(0, \sigma^2_\alpha) \\
\end{aligned}
$$
```

$$\begin{aligned} \alpha_j &= \gamma_0 + \gamma_1 sector_j + u_j \\ u_j &\sim N(0, \sigma^2_\alpha) \end{aligned}$$

The Gelman and Hill bracket notation looks like this:

```
$$
\begin{aligned}
\alpha_i &= \alpha_{j[i]} + \beta_1 ses_i + \epsilon_i \\
\epsilon_i &\sim N(0, \sigma^2_y) \\
\alpha_j &= \gamma_0 + \gamma_1 sector_j + u_j \\
u_j &\sim N(0, \sigma^2_\alpha)
\end{aligned}
$$
```

```

u_{j} \sim N( 0, \sigma^2_\alpha ) \\
\end{aligned}
$$

```

$$\begin{aligned}
y_i &= \alpha_{j[i]} + \beta_1 ses_i + \epsilon_i \\
\epsilon_i &\sim N(0, \sigma_y^2) \\
\alpha_j &= \gamma_0 + \gamma_1 sector_j + u_j \\
u_j &\sim N(0, \sigma_\alpha^2)
\end{aligned}$$

The reduced form would look like this:

```

$$
y_{\{i\}} = \gamma_0 + \gamma_1 sector_{j[i]} + \beta_1 ses_{\{i\}} + u_{j[i]} + \epsilon_{\{i\}}
$$

```

$$y_i = \gamma_0 + \gamma_1 sector_{j[i]} + \beta_1 ses_i + u_{j[i]} + \epsilon_i$$

with

```

$$
\epsilon_i \sim N( 0, \sigma_y^2 ), \text{ and } u_j \sim N( 0, \sigma_\alpha^2 )
$$

```

$$\epsilon_i \sim N(0, \sigma_y^2), \text{ and } u_j \sim N(0, \sigma_\alpha^2)$$

If we want to be really prissy, we can write down the i.i.d. aspect of our random effects like this

```

$$
\epsilon_i \stackrel{i.i.d.}{\sim} N( 0, \sigma_y^2 ), \\
u_j \stackrel{i.i.d.}{\sim} N( 0, \sigma_\alpha^2 )
$$

```

$$\epsilon_i \stackrel{i.i.d.}{\sim} N(0, \sigma_y^2), \text{ and } u_j \stackrel{i.i.d.}{\sim} N(0, \sigma_\alpha^2)$$

The `\stackrel{}{\sim}` command takes two bits of latex, each in the curly braces, and stacks them on top of each other.

8.4 Sample code: Random Slope Model

The canonical random slope model for HS&B with `ses` at level 1 and sector at level 2 involves a matrix for the pair of random effects. We have to get a bit fancier with our TeX, therefore!

Level 1 models:

```
$$
\begin{aligned}
y_{ij} &= \beta_{0j} + \beta_{1j} ses_{ij} + \epsilon_{ij} \\
\epsilon_{ij} &\sim N(0, \sigma_y^2)
\end{aligned}
$$
```

$$\begin{aligned} y_{ij} &= \beta_{0j} + \beta_{1j} ses_{ij} + \epsilon_{ij} \\ \epsilon_{ij} &\sim N(0, \sigma_y^2) \end{aligned}$$

Level 2 models:

```
$$
\begin{aligned}
\beta_{0j} &= \gamma_{00} + \gamma_{01} sector_j + u_{0j} \\
\beta_{1j} &= \gamma_{10} + \gamma_{11} sector_j + u_{1j}
\end{aligned}
$$
```

$$\begin{aligned} \beta_{0j} &= \gamma_{00} + \gamma_{01} sector_j + u_{0j} \\ \beta_{1j} &= \gamma_{10} + \gamma_{11} sector_j + u_{1j} \end{aligned}$$

with

```
$$
\begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} \sim N
\begin{pmatrix} 0 \\ 0 \end{pmatrix}

```

```

\end{pmatrix} \! \! \!, &
\begin{pmatrix}
\tau_{00} & \tau_{01} \\
& \tau_{11}
\end{pmatrix}
\end{pmatrix}
\end{bmatrix}
\\
\end{array}
\right.

```

$$\begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} \sim N \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \tau_{00} & \tau_{01} \\ \tau_{10} & \tau_{11} \end{pmatrix} \right]$$

The TeX for the derivation of the reduced form is:

```

\begin{aligned}
y_{ij} &= \beta_{0j} + \beta_{1j} ses_{ij} + \epsilon_{ij} \\
&= (\gamma_{00} + \gamma_{01} sector_j + u_{0j}) + (\gamma_{10} + \gamma_{11} sector_j + u_{1j}) ses_{ij} + \epsilon_{ij} \\
&= \gamma_{00} + \gamma_{01} sector_j + u_{0j} + \gamma_{10} ses_{ij} + \gamma_{11} sector_j ses_{ij} + u_{1j} ses_{ij} + \epsilon_{ij} \\
&= \gamma_{00} + \gamma_{01} sector_j + \gamma_{10} ses_{ij} + \gamma_{11} sector_j ses_{ij} + (u_{0j} + u_{1j} ses_{ij} + \epsilon_{ij})
\end{aligned}

```

$$\begin{aligned}
y_{ij} &= \beta_{0j} + \beta_{1j} ses_{ij} + \epsilon_{ij} \\
&= (\gamma_{00} + \gamma_{01} sector_j + u_{0j}) + (\gamma_{10} + \gamma_{11} sector_j + u_{1j}) ses_{ij} + \epsilon_{ij} \\
&= \gamma_{00} + \gamma_{01} sector_j + u_{0j} + \gamma_{10} ses_{ij} + \gamma_{11} sector_j ses_{ij} + u_{1j} ses_{ij} + \epsilon_{ij} \\
&= \gamma_{00} + \gamma_{01} sector_j + \gamma_{10} ses_{ij} + \gamma_{11} sector_j ses_{ij} + (u_{0j} + u_{1j} ses_{ij} + \epsilon_{ij})
\end{aligned}$$

Commentary: There are various and competing ways of writing the covariance matrix for the random effects. The τ_{**} notation is easy and expands to any sized matrix (if we, for example, have more than one random slope). But all the τ_{**} are variances, not standard deviations, and we often like to talk about random effect variation in terms of standard deviations. We can thus use something like this instead:

```

\begin{pmatrix}
u_{0j} \\
u_{1j}
\end{pmatrix}

```

```

\end{pmatrix} \sim N
\begin{bmatrix}
\begin{pmatrix}
0 \\
0 \\
\end{pmatrix} \! , \! &
\begin{pmatrix}
\sigma_0^2 & \rho\sigma_0\sigma_1 \\
& \sigma_1^2
\end{pmatrix}
\end{bmatrix}
\end{bmatrix}
$$

```

$$\begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} \sim N \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_0^2 & \rho\sigma_0\sigma_1 \\ & \sigma_1^2 \end{pmatrix} \right]$$

Here we have a correlation of random effects, ρ , instead of a covariance, τ_{01} . And we can talk about the standard deviation of, e.g., the random intercepts, as σ_0 rather than $\sqrt{\tau_{00}}$. Different ways of writing the same mathematical thing are called different *parameterizations*; they are equivalent, but are more or less clear for different contexts.

Unfortunately, this means there is no one right answer for how to write down a mathematical equation!

8.5 Summations and fancy stuff

Fractions are as follows:

```

$$
cor( A, B ) = \frac{ cov( A, B ) }{ \sigma_A \sigma_B }
$$

```

$$cor(A, B) = \frac{cov(A, B)}{\sigma_A \sigma_B}$$

For reference, you can do summations and whatnot as follows:

```

$$
Var( Y_{i} ) = \frac{1}{n-1} \sum_{i=1}^n \left( Y_i - \bar{Y} \right)^2
$$

```

$$Var(Y_i) = \frac{1}{n-1} \sum_{i=1}^n (Y_i - \bar{Y})^2$$

And if you have fractions you can have big brackets with “\left(” and “\right)” as follows:

```
$$
X = \left( \frac{1}{2} + y \right)
$$
```

$$X = \left(\frac{1}{2} + y \right)$$

Annoyingly, you always need a pair of these big brackets. If you really don't want one, you use a backslash and a dot, like so:

```
$$
X = \left( \frac{1}{2} + y \right. .
$$
```

$$X = \left(\frac{1}{2} + y \right.$$

The rest you can find on StackOverflow or similar. Or perhaps have ChatGPT help you write your code!

9 pivot_longer and pivot_wider

Generally, you want your data to be in a form where each row is a case and each column is a variable (either explanatory or response). Sometimes your data don't start that way. This section describes how to move your data around to get it in that form. The tidyverse provides a simple method for doing this (`pivot_longer()` and `pivot_wider()`) which you should read about in R for Data Science. There are also "old school" ways of doing this, via a method called `reshape()`; this way is more powerful and useful in some circumstances. See the final section for more on this old-style approach.

But for now, the pivot methods will pretty much do everything you want. Both `pivot_longer` and `pivot_wider` from `tidyverse` are great functions to understand. First, we load `tidyverse` and make some fake data.

```
library(tidyverse)

dat <- data.frame( ID = c( 1:3 ),
                    X = c( 10, 20, 30 ),
                    Y1 = 1:3,
                    Y2 = 10 + 1:3,
                    Y3 = 20 + 1:3 )

dat
```

	ID	X	Y1	Y2	Y3
1	1	10	1	11	21
2	2	20	2	12	22
3	3	30	3	13	23

This data is in wide format, where we have multiple measurements (Y1, Y2, and Y3) for each individual (each row of data).

9.1 Converting wide data to long data

We use `pivot_longer` to take our Y values and nest them within each ID for longitudinal MLM analysis. (NB you can use SEM to fit longitudinal models with wide data; we do not explore that application here.)

```
datL <- pivot_longer(dat, Y1:Y3,
                      names_to = "time",
                      values_to = "front" )

datL
```



```
# A tibble: 9 x 4
  ID      X time   front
  <int> <dbl> <chr> <dbl>
1 1       10 Y1     1
2 1       10 Y2     11
3 1       10 Y3     21
4 2       20 Y1     2
5 2       20 Y2     12
6 2       20 Y3     22
7 3       30 Y1     3
8 3       30 Y2     13
9 3       30 Y3     23
```

9.2 Converting long data to wide data

`pivot_wider` takes us back in the other direction.

```
newdat <- pivot_wider( datL, c(ID, X),
                       names_from=time,
                       values_from=front  )

newdat
```



```
# A tibble: 3 x 5
  ID      X     Y1     Y2     Y3
  <int> <dbl> <dbl> <dbl> <dbl>
1 1       10     1     11     21
2 2       20     2     12     22
3 3       30     3     13     23
```

We then verify our work with a few checks.

```
stopifnot( length( unique( newdat$ID ) ) == nrow( newdat ) )

students = datL %>% dplyr::select( ID, X ) %>%
  unique()
students

# A tibble: 3 x 2
  ID      X
  <int> <dbl>
1     1     10
2     2     20
3     3     30

students = merge( students, newdat, by="ID" )
```

9.3 Optional: wrangling data with reshape

The `reshape()` command is the old-school way of doing things, and it is harder to use but also can be more powerful in some ways (alternatively, there is a long literature on doing fancy stuff with the pivot methods as well). This section is entirely optional and possibly no longer useful.

Anyway, say you have data in a form where a row has a value for a variable for several different points in time. The following code turns it into a data.frame where each row (case) is a value for the variable at that point in time. You also have an ID variable for which Country the GDP came from.

```
dtw = read.csv( "data/fake_country_block.csv", as.is=TRUE )
dtw
```

	Country	X1997	X1998	X1999	X2000	X2001	X2002	X2003	X2004
1	China	0.5	1	2	3.4	4	5.3	6.0	7
2	Morocco	31.9	32	33	34.0	NA	36.0	37.0	NA
3	England	51.3	52	53	54.3	55	56.0	57.3	58

Here we have three rows, but actually a lot of cases if we consider each time point a case. For trying it on your own, get the sample csv file ()[\[here\]](#)

See the website to get the sample csv file \verb|fake_country_block.csv|.

The following *reshapes* our original data by making a case for each time point:

```
dt = reshape( dtw, idvar="Country", timevar="Year", varying=2:9, sep="", direction="long" )
head(dt)
```

	Country	Year	X
China.1997	China	1997	0.5
Morocco.1997	Morocco	1997	31.9
England.1997	England	1997	51.3
China.1998	China	1998	1.0
Morocco.1998	Morocco	1998	32.0
England.1998	England	1998	52.0

Things to notice: each case has a “row name” made out of the country and the Year. The “2:9” indicates a range of columns for the variable that is actually the same variable.

R picked up that, for each of these columns, “X” is the name of the variable and the number is the time, and separated them. You can set the name of your time variable, `\verb|timevar|`, to whatever you want.

The above output is called “long format” and the prior is called “wide format.”
You can go in either direction. Here:

```
dtn = reshape( dt, idvar="Country", timevar="Year" )
dtn
```

	Country	X.1997	X.1998	X.1999	X.2000	X.2001	X.2002	X.2003	X.2004
China.1997	China	0.5	1	2	3.4	4	5.3	6.0	7
Morocco.1997	Morocco	31.9	32	33	34.0	NA	36.0	37.0	NA
England.1997	England	51.3	52	53	54.3	55	56.0	57.3	58

You can reshape on multiple variables. For example:

```
exp.dat = data.frame( ID=c("a","b","c","d"),
  cond = c("AI","DI","DI","AI"),
  trial1 = c("E","U","U","E"),
  dec1 = c(1,1,0,1),
  trial2 = c("U","E","U","E"),
  dec2 = c(0,0,0,1),
  trial3 = c("U","E","E","U"),
  dec3 = c(0,1,0,1),
  trial4 = c("E","U","E","U"),
  dec4 = c(0,1,0,0) )
exp.dat
```

```

ID cond trial1 dec1 trial2 dec2 trial3 dec3 trial4 dec4
1 a AI E 1 U 0 U 0 E 0
2 b DI U 1 E 0 E 1 U 1
3 c DI U 0 U 0 E 0 E 0
4 d AI E 1 E 1 U 1 U 0

```

```

rs = reshape( exp.dat, idvar="ID",
              varying=c( 3:10 ), sep="", direction="long")
head(rs)

```

	ID	cond	time	trial	dec
a.1	a	AI	1	E	1
b.1	b	DI	1	U	1
c.1	c	DI	1	U	0
d.1	d	AI	1	E	1
a.2	a	AI	2	U	0
b.2	b	DI	2	E	0

It sorts out which variables are which. Note the names have to be exactly the same for any group of variables.

Once you have reshaped, you can look at things more easily (I use mosaic's tally instead of the base table):

```
mosaic::tally( trial ~ dec, data=rs )
```

```

dec
trial 0 1
  E 4 4
  U 5 3

```

or

```
mosaic::tally( trial~dec+cond, data=rs )
```

```
, , cond = AI
```

```

dec
trial 0 1
  E 1 3

```

```
U 3 1  
, , cond = DI  
  
dec  
trial 0 1  
E 3 1  
U 2 2
```

10 An Introduction to Missing Data

10.1 Introduction

Handling missing data is the icky, unglamorous part of any statistical analysis. It is where the skeletons lie. There's a range of options available, which are, broadly speaking:

1. Delete the observations with missing covariates (this is a “complete case analysis”)
2. Plug in some kind of reasonable value for the missing covariate. This is called “imputation.” We discuss three ways of doing this that are increasingly sophisticated and layered on each other:
 - a. Mean imputation. Simply take the mean of all the observations where you know the value, and then use that for anything that is missing.
 - b. Regression imputation. You generate regression equations describing how all the variables are connected, and use those to predict any missing value.
 - c. Stochastic regression imputation. Here we use regression imputation, but we also add some residual noise to all our imputed values so that our imputed values have as much variation as our actual values (otherwise our imputed values will tend to be all clumped together).
3. Multiply impute the missing data, by fully modeling the covariate and the missingness, and generating a range of complete datasets under this model. Here you end up with a bunch of complete datasets that are all “reasonable guesses” as to what the full dataset might have been. You then analyze each one, and aggregate your findings across them to get a final answer.

The first two general approaches are imperfect, while the third is often more work than the original analysis that we were hoping to perform. For this course, doing a 2a, 2b, or 2c are all reasonable choices. If you have very little missing data you can often get away with 1. We have no expectations that people will take the plunge into #3 (multiple imputation). In real life, people will often analyze their data with a complete case analysis and some other strategy, and then compare the results. In Education, if missingness is below 10% people usually just do mean imputation, but regression imputation would probably be superior.

This handout provides an introduction to missing data, and includes a few commands to explore and deal with missing data. In this document we first talk about exploring missing

data (in particular getting plots that show you if you have any notable patterns in how things are missing) and then we give a brief walk-through of the 3 methods listed above.

We will use the `mice` and `VIM` packages, which you can install using `install.packages()` if you have not yet done so. These are simple and powerful packages for visualizing and imputing missing data. At the end of this document we also describe the `Amelia` package.

```
library(tidyverse)
library(mice)
library(VIM)
```

Throughout we use a small built-in R dataset on air quality as a working example.

```
data(airquality)
nrow(airquality)
```

```
[1] 153
```

```
head(airquality)
```

	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	36	118	8.0	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4
5	NA	NA	14.3	56	5	5
6	28	NA	14.9	66	5	6

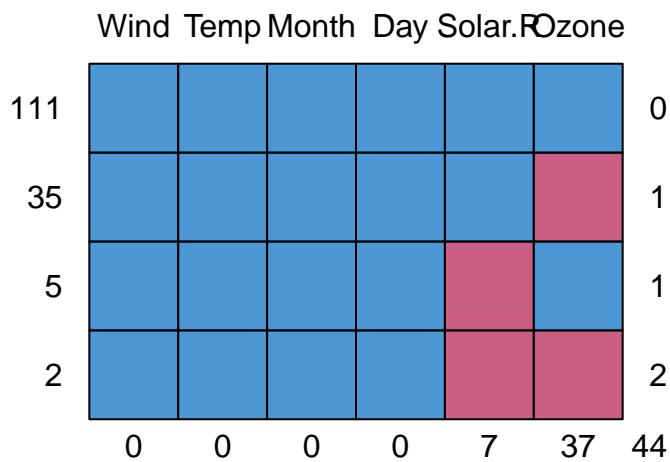
```
summary(airquality[1:4])
```

Ozone	Solar.R	Wind	Temp
Min. : 1.0	Min. : 7	Min. : 1.70	Min. :56.0
1st Qu.: 18.0	1st Qu.:116	1st Qu.: 7.40	1st Qu.:72.0
Median : 31.5	Median :205	Median : 9.70	Median :79.0
Mean : 42.1	Mean :186	Mean : 9.96	Mean :77.9
3rd Qu.: 63.2	3rd Qu.:259	3rd Qu.:11.50	3rd Qu.:85.0
Max. :168.0	Max. :334	Max. :20.70	Max. :97.0
NA's :37	NA's :7		

10.2 Visualizing missing data

Just like with anything in statistics, the first thing to do is to look at our data. We want to know which variables are often missing, and if some variables are often missing together. We also want to know how much data is missing. The mice package has a variety of plots to show us patterns of missingness:

```
md.pattern(airquality)
```



	Wind	Temp	Month	Day	Solar.R	Ozone
111	1	1	1	1	1	1
35	1	1	1	1	1	0
5	1	1	1	1	0	1
2	1	1	1	1	0	0
	0	0	0	0	7	37
						44

This plot gives us the different missing data patterns and the number of observations that have each missing data pattern. For example, the second row in the plot says there are 35 observations that have a missing data pattern where only Ozone is missing.

Easier to understand patterns!

We can also just look at 10 observations to see everything that is going on. Here we take the first 10 rows of our dataset, but could also take a random 10 row with the tidyverse's `sample_n` method.

```
airqualitysub = airquality[1:10, ]
airqualitysub
```

	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	36	118	8.0	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4
5	NA	NA	14.3	56	5	5
6	28	NA	14.9	66	5	6
7	23	299	8.6	65	5	7
8	19	99	13.8	59	5	8
9	8	19	20.1	61	5	9
10	NA	194	8.6	69	5	10

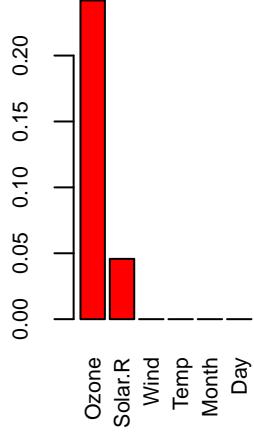
We see that we have one observation missing two covariates and one each of missing Ozone only and Solar.R only.

10.2.1 The VIM Package

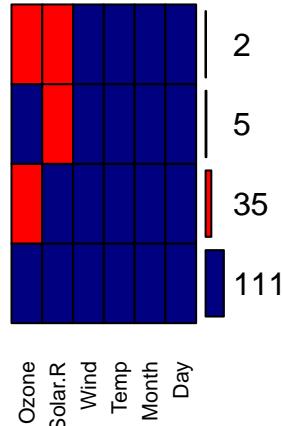
The VIM package gives some alternate plots to explore missing data patterns. For example, `aggr()`:

```
aggr(airquality, col=c('navyblue','red'),
      numbers=TRUE, sortVars=TRUE, labels=names(data),
      cex.axis=.7, gap=3, prop=c(TRUE, FALSE),
      ylab=c("Histogram of missing data","Pattern"))
```

Histogram of missing data



Pattern

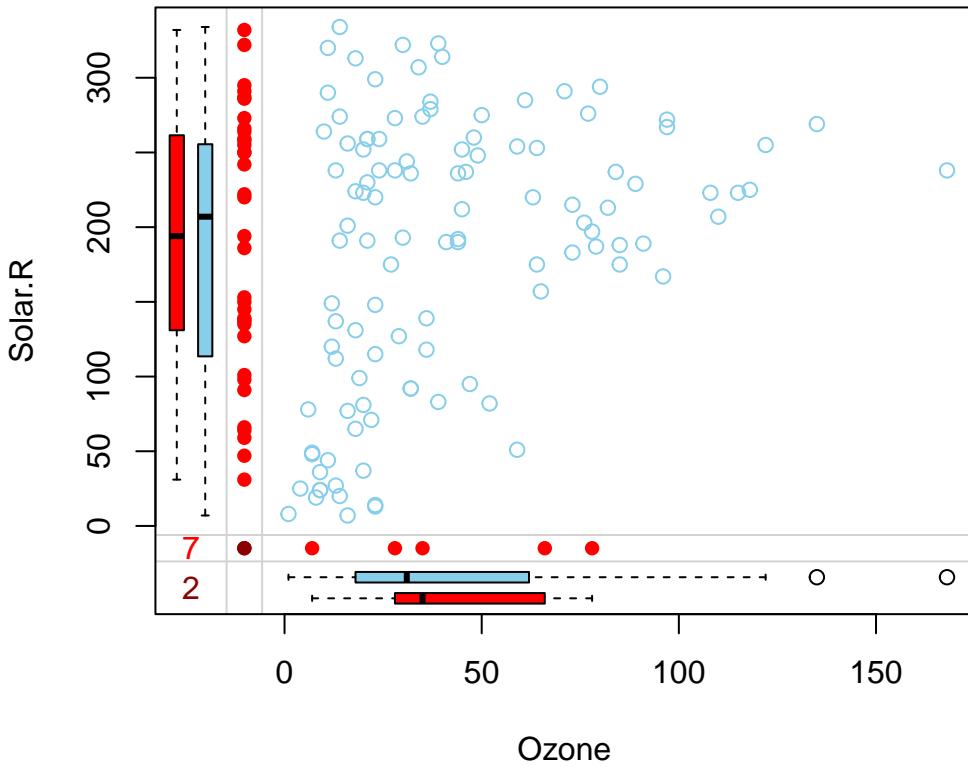


Variables sorted by number of missings:

Variable	Count
Ozone	0.2418
Solar.R	0.0458
Wind	0.0000
Temp	0.0000
Month	0.0000
Day	0.0000

On the left, we have the proportion of missing data for each variable in our dataset. We can see that Ozone and Solar.R have missing values. On the right, we have the joint distribution of missingness. We can see that 111 observations have no missing values. From those with missing values, the majority have missing values for Ozone, some have missing values for Solar.R and only 2 observations have missing values for both Ozone and Solar.R.

```
marginplot(airquality[1:2])
```



Here we have a scatterplot for the first two variables in our dataset: Ozone and Solar.R. These are the variables that have missing data. In addition to the standard scatterplot we are familiar with, information about missingness is shown in the margins. The red dots indicate observations with one or both values missing (so there can be a bunch of dots stacked up in the bottom-left corner). The numbers (37, 7, and 2 tells us how many observations are missing either or both of these variables).

10.3 Complete case analysis

Working with complete cases (dropping observations with any missing data on our outcome and predictors) is always an option. We have been doing this in class and section. However, this can lead to substantial data loss, if we have a lot of missingness and it can heavily bias our results depending on why observations are missing.

Complete case analysis is the R default.

```
fit <- lm(Ozone ~ Wind, data = airquality )
summary(fit)
```

```
Call:
lm(formula = Ozone ~ Wind, data = airquality)

Residuals:
    Min      1Q  Median      3Q     Max 
-51.57 -18.85  -4.87  15.23  90.00 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  96.87     7.24   13.38  < 2e-16 ***
Wind        -5.55     0.69   -8.04  9.3e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 26.5 on 114 degrees of freedom
(37 observations deleted due to missingness)
Multiple R-squared:  0.362, Adjusted R-squared:  0.356 
F-statistic: 64.6 on 1 and 114 DF,  p-value: 9.27e-13
```

Note the listing in the summary of number of items deleted. You can find out which rows were deleted:

```
## which rows/observations were deleted
deleted <- na.action(fit)
deleted
```

5	10	25	26	27	32	33	34	35	36	37	39	42	43	45	46	52	53	54	55
5	10	25	26	27	32	33	34	35	36	37	39	42	43	45	46	52	53	54	55

```

56 57 58 59 60 61 65 72 75 83 84 102 103 107 115 119 150
56 57 58 59 60 61 65 72 75 83 84 102 103 107 115 119 150
attr(,"class")
[1] "omit"

naprint(deleted)

```

```
[1] "37 observations deleted due to missingness"
```

We have more incomplete rows if we add Solar.R as predictor.

```

fit2 <- lm(Ozone ~ Wind+Solar.R, data=airquality)
naprint(na.action(fit2))

```

```
[1] "42 observations deleted due to missingness"
```

We can also drop observations with missing data ourselves instead of letting R do it for us. **Dropping data preemptively is generally a good idea, especially if you plan on using predict().**

```

## complete cases on all variables in the data set
complete.v1 = filter(airquality, complete.cases(airquality) )

## drop observations with missing values, but ignoring a specific variable
complete.v2 = filter(airquality, complete.cases(select(airquality, -Wind)) )

## drop observations with missing values on a specific variable
complete.v3 = filter(airquality, !is.na(Ozone))

```

Once you have subset your data, you just analyze what is left as normal. Easy as pie!

10.4 Mean imputation

Instead of dropping observations with missing values, we can plug in some kind of reasonable value for the missing value, e.g. the grand/global mean. While this can be statistically questionable, it does allow us to use the information provided by that unit's outcome and other covariates, without, we hope, unduly affecting the analysis of the missing covariate.

Generally, people will first plug in the mean value for anything missing, but then also make a dummy variable of whether that observation had a missing value there (or sometimes any missing value). You would then include both the original vector of covariates (with the means plugged in) along with the dummy variable in subsequent regressions and analyses.

10.4.1 Doing Mean Imputation manually

Manually, we can just replace missing values for a variable with the grand/global mean.

```
## make a new copy of the data
data.mean.impute = airquality

## select the observations with missing Ozone
miss.ozone = is.na(data.mean.impute$Ozone)

## replace those NAs with mean(Ozone)
data.mean.impute[miss.ozone, "Ozone"] = mean(airquality$Ozone, na.rm=TRUE)
```

In a multi-level context, it might make more sense to impute using the group mean rather than the grand mean. Here's a generic function to do it. Here we group by month:

```
## a function that replaces missing values in a vector
## by the mean of the other values
mean.impute = function(y) {
  y[is.na(y)] = mean(y, na.rm=TRUE)
  return(y)
}

data.mean.impute = airquality %>% group_by(Month) %>%
  mutate(Ozone = mean.impute(Ozone),
        Solar.R = mean.impute(Solar.R) )
```

We have mean imputed the Ozone column and the Solar.R column

10.4.2 Mean imputation with the Mice package

We can use the `mice` package to do mean imputation. The mice package is a package that can do some quite complex imputation, and so when you call `mice()` (which says “impute missing values please”) you get back a rather complex object telling you what mice imputed, for whom, etc. This object, which is a `mids` object (see `help(mids)`), contains the multiply imputed dataset (or in our case, so far, singly imputed). The `mice` package then provides a lot of nice functions allowing you to get your imputed information out of this object.

We first demonstrate this for the 10 observations sampled above. Mice is generally going to be a two-step process: impute data, get completed dataset.

For step 1:

```
imp <- mice(airqualitysub, method="mean", m=1, maxit=1)
```

```
iter imp variable  
1 1 Ozone Solar.R
```

```
Warning: Number of logged events: 1
```

```
imp
```

```
Class: mids  
Number of multiple imputations: 1  
Imputation methods:  
    Ozone Solar.R     Wind      Temp     Month      Day  
    "mean"   "mean"     ""       ""       ""       ""  
PredictorMatrix:  
    Ozone Solar.R Wind Temp Month Day  
Ozone      0      1     1     1      0     1  
Solar.R     1      0     1     1      0     1  
Wind        1      1     0     1      0     1  
Temp        1      1     1     0      0     1  
Month       1      1     1     1      0     1  
Day         1      1     1     1      0     0  
Number of logged events: 1  
  it im dep      meth    out  
1 0 0      constant Month
```

For step 2:

```
cmp = complete(imp)  
cmp
```

```
Ozone Solar.R Wind Temp Month Day  
1  41.0    190  7.4   67    5   1  
2  36.0    118  8.0   72    5   2  
3  12.0    149  12.6  74    5   3  
4  18.0    313  11.5  62    5   4  
5  23.1    173  14.3  56    5   5  
6  28.0    173  14.9  66    5   6
```

```

7   23.0      299  8.6   65      5   7
8   19.0        99 13.8   59      5   8
9    8.0       19 20.1   61      5   9
10  23.1      194  8.6   69      5  10

```

We see there are no missing values in `cmp`. They were all imputed with the mean of the other non-missing values. This is **mean imputation**.

Now let's impute the full dataset.

```
imp <- mice(airquality, method="mean", m=1, maxit=1)
```

```

iter imp variable
1   1  Ozone Solar.R

```

```
cmp = complete( imp )
```

We next make a dummy variable for each row of our data noting whether anything was imputed or not. We use the `ici` (Incomplete Case Indication) function to list all rows with any missing values.

```
head( ici(airquality) )
```

```
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE
```

Note how we have a TRUE or FALSE for each row of our data.

We then store this as a covariate in our completed dataset:

```

cmp$imputed = ici(airquality)
head( cmp )

```

	Ozone	Solar.R	Wind	Temp	Month	Day	imputed
1	41.0	190	7.4	67	5	1	FALSE
2	36.0	118	8.0	72	5	2	FALSE
3	12.0	149	12.6	74	5	3	FALSE
4	18.0	313	11.5	62	5	4	FALSE
5	42.1	186	14.3	56	5	5	TRUE
6	28.0	186	14.9	66	5	6	TRUE

10.4.2.1 How well did mean imputation work?

Mean imputation has problems. The imputed values will all be the same, and thus when we look at how much variation is in our variables after imputation, it will go down. Compare the SD of our completed dataset Ozone values to the SD of the Ozone values for our non-missing values.

```
sd( airquality$Ozone, na.rm=TRUE )
```

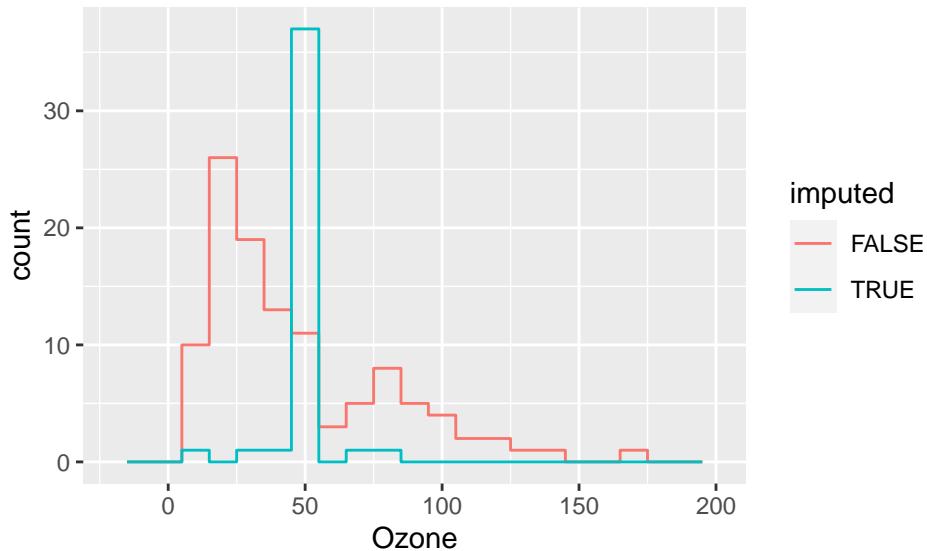
```
[1] 33
```

```
sd( cmp$Ozone )
```

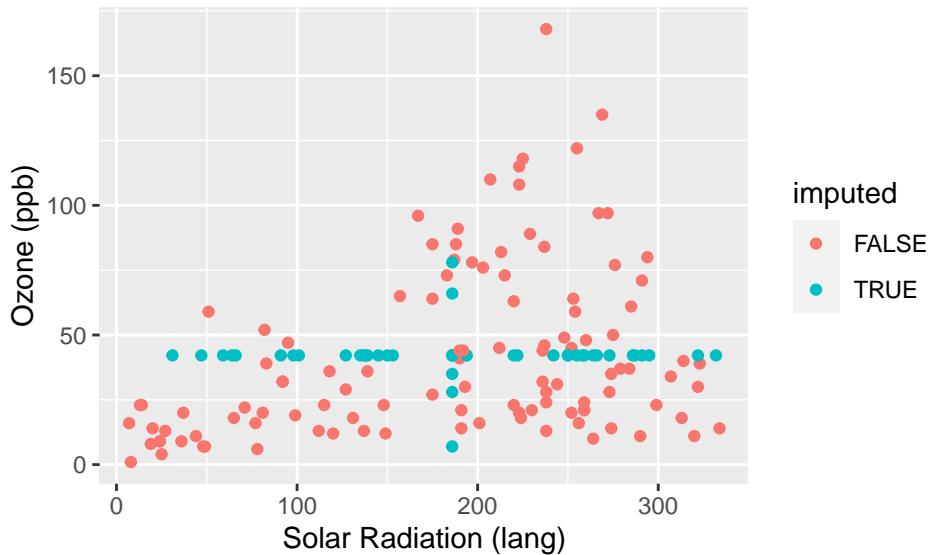
```
[1] 28.7
```

Next, let's look at some plots of our completed data, coloring the points by whether they were imputed.

```
library(ggplot2)
ggplot( cmp, aes(x=Ozone, col=imputed) ) +
  stat_bin( geom="step", position="identity",
            breaks=seq(-20, 200, 10) )
```



```
ggplot( cmp, aes(y=Ozone, x=Solar.R, col=imputed) ) +
  geom_point() +
  labs( y="Ozone (ppb)", x="Solar Radiation (lang)" )
```



What we see in the above plots is that our imputed observations do not look like the rest of our data because one (or both) of their values always is in the exact center. This creates the “+” shape. It also gives the big spike at the mean for the histogram.

10.4.2.2 Important Aside: Namespaces and function collisions

We now need to discuss a sad aspect of R. The short story is, different packages have functions with the same names and so if you have both packages loaded you will need to specify which package to use when calling such a function. You can do this by giving the “surname” of the function at the beginning of the function call (like, I believe, the Chinese). This comes up because for us the method `complete()` exists both in the tidyverse and in mice. In tidyverse, `complete()` fills in rows of missing combinations of values. In mice, `complete()` gives us a completed dataset after we have made an imputation call.

It turns out that since we loaded tidyverse first and mice second, the mice’s `complete()` method is the default. But if we loaded the packages in the other order, we would get strange errors. To be clear, we thus tell R to use `mice` by writing:

```
cmp = mice::complete( imp )
```

In general, you can detect such “namespace collisions” by noticing weird error messages all of a sudden when you don’t expect them. You can then type, for example, `help(complete)` and it will list all the different `completes` around.

```
help( complete )
```

Also when you load a package it will write down what functions are getting mixed up for you. If you were looking at your R code you would get something like this:

```
tidyR::complete() masks mice::complete()
```

10.5 Regression imputation

Regression imputation is half way between mean imputation and multiple imputation. In regression imputation we predict what values we expect for anything missing based on the other values of the observation. For example, if we know that urban/rural is correlated with race, we might impute a different value for race if we know an observation came from an urban environment vs. rural. We do this with regression: we fit a model predicting each variable using the others and then use that regression model to predict any missing values.

We can do this manually, but then it gets very hard when multiple variables are missing for a given observation. The `mice` package is more clever: it does variables one at a time, and the cycles around so everything can get imputed.

10.5.1 Manually

Here is how to use other variables to predict missing values.

```
ic( airqualitysub )
```

	Ozone	Solar.R	Wind	Temp	Month	Day
5	NA	NA	14.3	56	5	5
6	28	NA	14.9	66	5	6
10	NA	194	8.6	69	5	10

```

fit <- lm(Ozone ~ Solar.R, data=airqualitysub)

## predict for missing ozone
need.pred = subset( airqualitysub, is.na( Ozone ) )
need.pred

```

	Ozone	Solar.R	Wind	Temp	Month	Day
5	NA	NA	14.3	56	5	5
10	NA	194	8.6	69	5	10

```

pred <- predict(fit, newdata=need.pred)
pred

```

```

5   10
NA 23.1

```

But now we have to merge back in, and we didn't solve for case 5 because we are missing the variable we would use to predict the other missing variable. Ick. This is where missing data gets *really* hard (when we have multiple missing values on multiple variables). So let's quit now and turn to a package that will handle all of this for us.

10.5.2 Mice

To do regression imputation using mice, we simply call the `mice()` method:

```
imp <- mice(airquality[,1:2], method="norm.predict", m=1, maxit=3, seed=1)
```

```

iter imp variable
1   1  Ozone  Solar.R
2   1  Ozone  Solar.R
3   1  Ozone  Solar.R

```

We have everything! How did it do it? By *chaining equations*. First we start with mean imputation. Then we use our fit model to predict for one covariate, and then we use those predicted scores to predict for the next covariate, and so forth. We cycle back and then everything is jointly predicting everything else.

The `complete()` method gives us a complete dataset with everything imputed. Like so:

```
cdat = mice::complete( imp )
head( cdat )
```

```
Ozone Solar.R
1  41.0     190
2  36.0     118
3  12.0     149
4  18.0     313
5  42.7     186
6  28.0     169
```

```
nrow( cdat )
```

```
[1] 153
```

```
nrow( airquality )
```

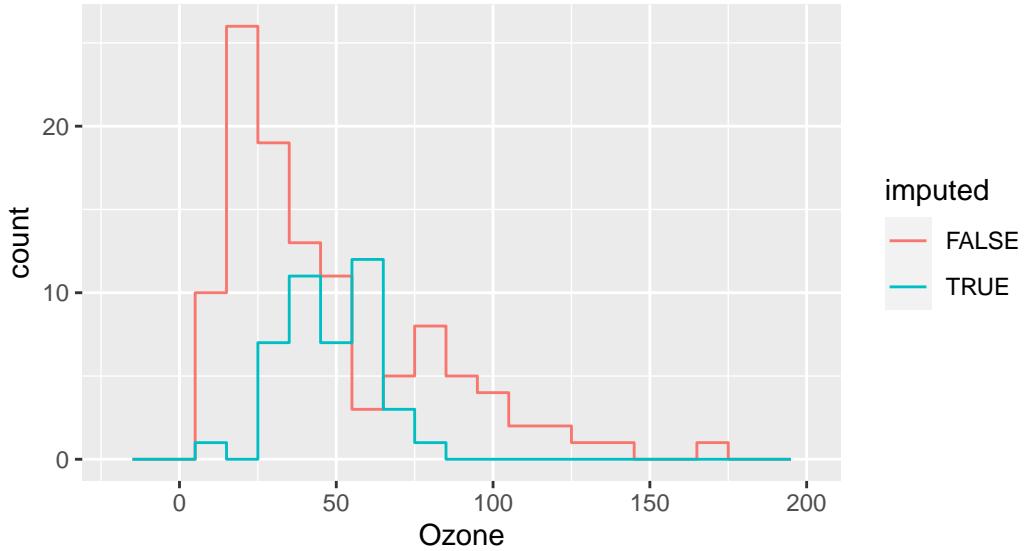
```
[1] 153
```

Next we make a variable of which cases have imputed values and not (any row with missing data must have been partially imputed.)

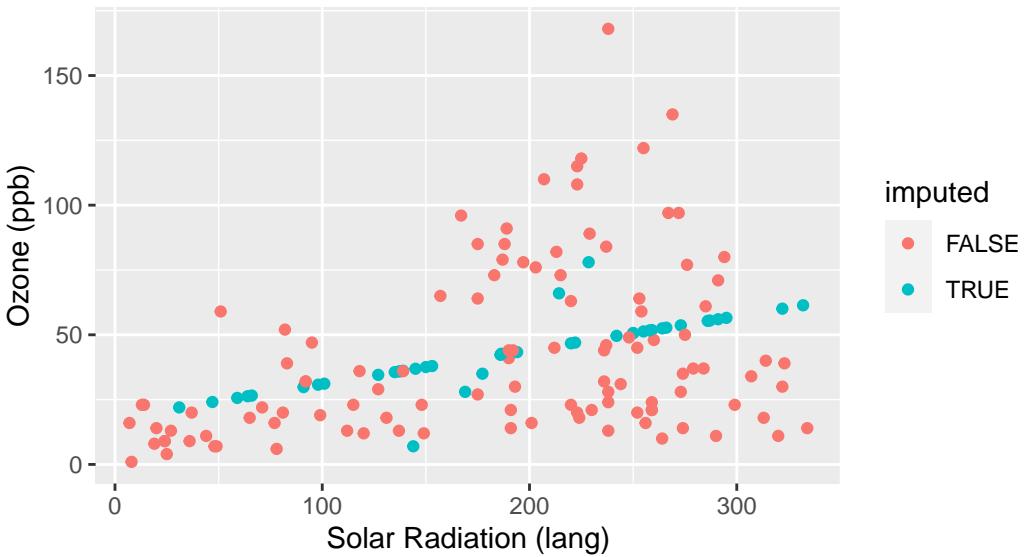
```
cdat$imputed = ici( airquality )
```

And see our results! Compare to mean imputation, above.

```
ggplot( cdat, aes(x=Ozone, col=imputed) ) +
  stat_bin( geom="step", position="identity",
            breaks=seq(-20, 200, 10) )
```



```
ggplot( cdat, aes(y=Ozone, x=Solar.R, col=imputed) ) +
  geom_point() +
  labs( y="Ozone (ppb)", x="Solar Radiation (lang)" )
```



This is better than mean imputation. See how we impute different Ozone for different Solar Radiation values, taking advantage of the information of knowing that they are correlated? But it still is obvious what is mean imputed and what is not. Also, the variance of our imputed

values still does not contain the residual variation around the predicted values that we would get in real data. We can do one more enhancement to fix this.

10.5.3 Stochastic regression imputation

We extend regression imputation by randomly drawing observations that *look like* real ones. See in the two imputations below we get slightly different values for our imputed data.

Here we do it on our mini-dataset and look at the imputed values for our observations with missing values only:

```
imp <- mice(airqualitysub[,1:2],method="norm.nob",m=1,maxit=1,seed=1)
```

```
iter imp variable
 1   1  Ozone  Solar.R
```

```
imp$imp
```

```
$Ozone
 1
5 8.09
10 44.58
```

```
$Solar.R
 1
5 181.2
6 83.7
```

```
imp <- mice(airqualitysub[,1:2],method="norm.nob",m=1,maxit=1,seed=4)
```

```
iter imp variable
 1   1  Ozone  Solar.R
```

```
imp$imp
```

```
$Ozone
 1
5 34.4
10 31.6

$Solar.R
 1
5 381
6 260
```

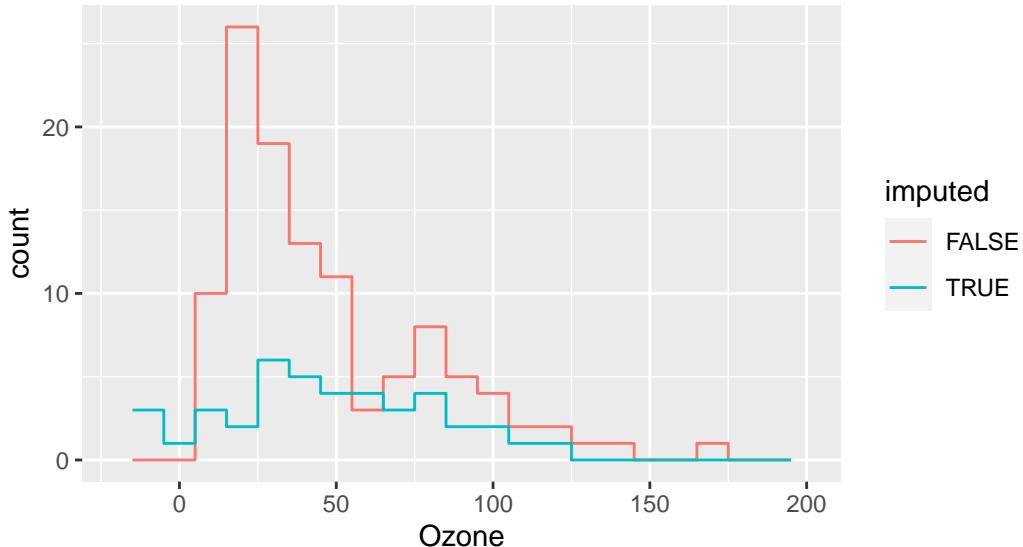
Now let's do it on the full data and look at the imputed values and compare to our plots above.

```
imp <- mice(airquality[,1:2], method="norm.nob", m=1, maxit=1, seed=1)
```

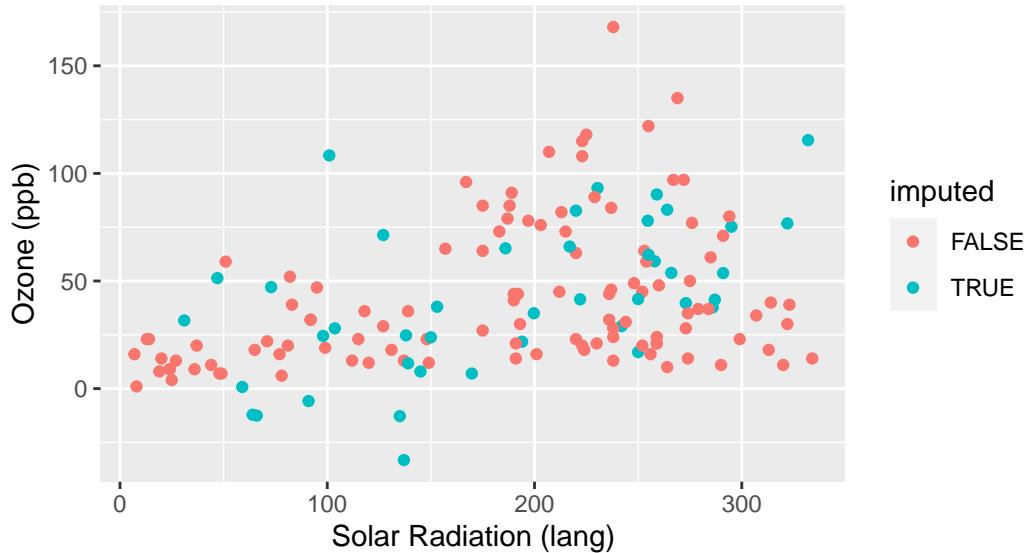
```
iter imp variable
1 1 Ozone Solar.R

cdat = mice:::complete( imp )
cdat$imputed = ici( airquality )

ggplot( cdat, aes(x=Ozone, col=imputed) ) +
  stat_bin( geom="step", position="identity",
            breaks=seq(-20, 200, 10) )
```



```
ggplot( cdat, aes(y=Ozone, x=Solar.R, col=imputed) ) +
  geom_point() +
  labs( y="Ozone (ppb)", x="Solar Radiation (lang)" )
```



Better, but not perfect. What is better? What is still not perfect?

10.6 Multiple imputation

If missing data is a significant issue in your dataset, then mean or regression imputation can be a bit too hacky and approximate. In these contexts, multiple imputation is the way to go.

We do this as follows:

```
imp <- mice(airqualitysub, seed=2, print=FALSE)
```

Warning: Number of logged events: 1

```
imp
```

```

Class: mids
Number of multiple imputations: 5
Imputation methods:
  Ozone Solar.R    Wind    Temp   Month    Day
  "pmm"   "pmm"    ""     ""     ""     ""
PredictorMatrix:
  Ozone Solar.R Wind Temp Month Day
Ozone      0       1   1   1   0   1
Solar.R     1       0   1   1   0   1
Wind        1       1   0   1   0   1
Temp        1       1   1   0   0   1
Month       1       1   1   1   0   1
Day         1       1   1   1   0   0
Number of logged events: 1
  it im dep     meth   out
1 0 0      constant Month

```

```
imp$imp
```

```

$Ozone
  1 2 3 4 5
5 18 41 28 23 23
10 36 18 36 19 28

$Solar.R
  1 2 3 4 5
5 149 99 194 99 19
6 194 19 194 19 19

$Wind
[1] 1 2 3 4 5
<0 rows> (or 0-length row.names)

$Temp
[1] 1 2 3 4 5
<0 rows> (or 0-length row.names)

$Month
[1] 1 2 3 4 5
<0 rows> (or 0-length row.names)

$Day

```

```
[1] 1 2 3 4 5  
<0 rows> (or 0-length row.names)
```

See multiple columns of imputed data? (We have 5 here.)

First aside: All variables you'll be using for your model should be included in the imputation model. Notice we included the full dataset in `mice`, not just the variables with missing values. This way we can account for associations between all the outcome and the predictors in the model we'll be fitting. Your imputation model can be more complicated than your model of interest. That is, you can include additional variables that predict missing values but will not be part of your final model of interest.

Second aside: All variables in your imputation model should be in the correct functional form! Quadratic, higher order polynomials and interaction terms are just another variable that we need to impute. Although it may seem logical to impute your variables first and then calculate the interaction or non-linear term, this can lead to bias.

Third aside: The ordering of the variables in the dataset you are feeding into `mice` can make a difference in results and model convergence. Generally, you want to order your variables from least to most missing. Here, we reorder the variables from least to most missing, and obtain different results.

```
test = airquality[ , c(6,5,4,3,2,1)]  
head(test)
```

	Day	Month	Temp	Wind	Solar.R	Ozone
1	1	5	67	7.4	190	41
2	2	5	72	8.0	118	36
3	3	5	74	12.6	149	12
4	4	5	62	11.5	313	18
5	5	5	56	14.3	NA	NA
6	6	5	66	14.9	NA	28

```
test.imp <- mice(test, seed=2, print=FALSE)
```

Warning: Number of logged events: 1

```
test.imp$imp
```

```
$Day  
[1] 1 2 3 4 5
```

```

<0 rows> (or 0-length row.names)

$Month
[1] 1 2 3 4 5
<0 rows> (or 0-length row.names)

$Temp
[1] 1 2 3 4 5
<0 rows> (or 0-length row.names)

$Wind
[1] 1 2 3 4 5
<0 rows> (or 0-length row.names)

$Solar.R
      1   2   3   4   5
5 194 118 194 313 190
6 118 194 118 118 190

$Ozone
      1   2   3   4   5
5  18  23  23  23  41
10 12   8  18  19   8

```

How to get each complete dataset?

```

## first complete dataset
mice::complete(imp, 1)

```

	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	36	118	8.0	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4
5	18	149	14.3	56	5	5
6	28	194	14.9	66	5	6
7	23	299	8.6	65	5	7
8	19	99	13.8	59	5	8
9	8	19	20.1	61	5	9
10	36	194	8.6	69	5	10

```
## and our second complete dataset
mice::complete(imp, 2)
```

	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	36	118	8.0	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4
5	41	99	14.3	56	5	5
6	28	19	14.9	66	5	6
7	23	299	8.6	65	5	7
8	19	99	13.8	59	5	8
9	8	19	20.1	61	5	9
10	18	194	8.6	69	5	10

See how they are different? They were randomly imputed. We basically used the stochastic regression thing, above, multiple times.

```
mice::complete(imp, 1)[ ici(airqualitysub), ]
```

	Ozone	Solar.R	Wind	Temp	Month	Day
5	18	149	14.3	56	5	5
6	28	194	14.9	66	5	6
10	36	194	8.6	69	5	10

```
mice::complete(imp, 2)[ ici(airqualitysub), ]
```

	Ozone	Solar.R	Wind	Temp	Month	Day
5	41	99	14.3	56	5	5
6	28	19	14.9	66	5	6
10	18	194	8.6	69	5	10

On full data:

```
imp <- mice(airquality, seed=1, print=FALSE)
```

Now we estimate for each imputed dataset using the `with()` method that, in `mice`, will do the regression for each completed dataset. See `help with.mids`.

```

fit <- with(imp, lm(Ozone ~ Wind + Temp + Solar.R))
fit

call :
with.mids(data = imp, expr = lm(Ozone ~ Wind + Temp + Solar.R))

call1 :
mice(data = airquality, printFlag = FALSE, seed = 1)

nmis :
  Ozone Solar.R     Wind      Temp    Month      Day
  37       7        0        0        0        0

analyses :
[[1]]

Call:
lm(formula = Ozone ~ Wind + Temp + Solar.R)

Coefficients:
(Intercept)      Wind      Temp      Solar.R
-66.2402       -2.8219      1.6134      0.0563

[[2]]

Call:
lm(formula = Ozone ~ Wind + Temp + Solar.R)

Coefficients:
(Intercept)      Wind      Temp      Solar.R
-71.2842       -2.9055      1.6749      0.0633

[[3]]

Call:
lm(formula = Ozone ~ Wind + Temp + Solar.R)

Coefficients:
(Intercept)      Wind      Temp      Solar.R
-66.9511       -2.9322      1.6479      0.0543

```

```
[[4]]
```

```
Call:  
lm(formula = Ozone ~ Wind + Temp + Solar.R)
```

```
Coefficients:
```

(Intercept)	Wind	Temp	Solar.R
-33.8480	-3.6628	1.3244	0.0427

```
[[5]]
```

```
Call:  
lm(formula = Ozone ~ Wind + Temp + Solar.R)
```

```
Coefficients:
```

(Intercept)	Wind	Temp	Solar.R
-77.4163	-2.7438	1.7264	0.0663

This can take *any* function call that takes a formula. So `glm`, `lm`, whatever... We can then pool the estimates using the standard theory of combining multiply imputed datasets. The basic idea is to combine the variation/uncertainty of the multiple sets with the average uncertainty we would have for each set if it was truly complete and not imputed.

```
tab <- summary(pool(fit))  
colnames(tab)
```

```
[1] "term"      "estimate"   "std.error"  "statistic" "df"        "p.value"
```

```
tab[,c(1:3,5)]
```

	term	estimate	std.error	df
1	(Intercept)	-63.1480	26.6769	13.8
2	Wind	-3.0132	0.6831	24.0
3	Temp	1.5974	0.2742	19.6
4	Solar.R	0.0566	0.0222	52.4

Aside: You will notice that once we fit our model on the imputed data, `with()` returned an object of class `mira`. `Mira` objects can be pooled to get the pooled estimates, whereas

objects of class `glm`, `lm`, `lmer`, etc. cannot be pooled. You will also notice that you cannot use `predict` with a `mira` object. To use `predict`, you can stack the imputed datasets and fit your model on this complete dataset. Parameter estimates generated by `pool` are the average of the parameter estimates from the model fit on each imputed dataset separately. So your coefficients are fine. However, your SEs will be underestimated. How underestimated your SEs will be depends, to an extent, on how much data is missing and whether it is missing at random.

Our old, sad method:

```
fit <- lm(Ozone~Wind+Temp+Solar.R,data=airquality,na.action=na.omit)
summary( fit )
```

```
Call:
lm(formula = Ozone ~ Wind + Temp + Solar.R, data = airquality,
    na.action = na.omit)

Residuals:
    Min      1Q  Median      3Q     Max 
-40.48 -14.22  -3.55  10.10  95.62 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -64.3421   23.0547  -2.79   0.0062 **  
Wind         -3.3336    0.6544  -5.09  1.5e-06 ***  
Temp          1.6521    0.2535   6.52  2.4e-09 ***  
Solar.R       0.0598    0.0232   2.58   0.0112 *   
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.2 on 107 degrees of freedom
(42 observations deleted due to missingness)
Multiple R-squared:  0.606, Adjusted R-squared:  0.595 
F-statistic: 54.8 on 3 and 107 DF,  p-value: <2e-16
```

```
round(coef(summary(fit)),3)
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-64.34	23.055	-2.79	0.006
Wind	-3.33	0.654	-5.09	0.000
Temp	1.65	0.254	6.52	0.000
Solar.R	0.06	0.023	2.58	0.011

In this case, the missing data estimates are basically the same as the complete case analysis, it appears. We only had 5% missing data though.

10.7 Extensions

10.7.1 Non-continuous variables

Everything shown above can easily be extended to non-continuous variables. The easiest way to do this is using the `mice` package. It allows you to specify the type of variable you are imputing, e.g. dichotomous or categorical. `Mice` will automatically detect and handle non-continuous variables. You can also specify these variables yourself. Here is an example using `nhanes` data (another built-in R dataset).

```
## load data
data(nhanes2)
head(nhanes2)

  age  bmi  hyp chl
1 20-39   NA <NA>  NA
2 40-59 22.7   no 187
3 20-39   NA   no 187
4 60-99   NA <NA>  NA
5 20-39 20.4   no 113
6 60-99   NA <NA> 184

## create some missing values for an ordered categorical variable
nhanes2$age[1:5] = NA
head(nhanes2)

  age  bmi  hyp chl
1 <NA>   NA <NA>  NA
2 <NA> 22.7   no 187
3 <NA>   NA   no 187
4 <NA>   NA <NA>  NA
5 <NA> 20.4   no 113
6 60-99   NA <NA> 184

## impute 5 datasets
imp.cat <- mice(nhanes2, m = 5, print=FALSE)
full.cat = mice::complete(imp.cat)           ## print the first imputed data set
head(full.cat)
```

```

age  bmi hyp chl
1 40-59 21.7 yes 187
2 40-59 22.7 no 187
3 20-39 27.5 no 187
4 60-99 24.9 yes 218
5 40-59 20.4 no 113
6 60-99 24.9 yes 184

```

We can check what imputation method `mice` used for each variable:

```
imp.cat$method
```

```

age      bmi      hyp      chl
"polyreg" "pmm"  "logreg" "pmm"

```

We can see that `mice` used the `polyreg` imputation method for the variable `age`, which means it treated it as an unordered categorical variable. But this is an ordered variable: higher values categories signified older age. We can manually force `mice` to treat `age` as an ordered categorical variable. We will keep the imputation methods for the remaining variables the same.

```
imp.cat2 <- mice(nhanes2, meth=c("polr","pmm","logreg","pmm"), m=5, print=FALSE)
head(mice::complete(imp.cat2, 1))
```

```

age  bmi hyp chl
1 40-59 27.5 yes 184
2 60-99 22.7 no 187
3 60-99 20.4 no 187
4 20-39 35.3 no 184
5 40-59 20.4 no 113
6 60-99 22.7 no 184

```

```
imp.cat2$method
```

```

age      bmi      hyp      chl
"polr"  "pmm"  "logreg" "pmm"

```

10.7.2 Multi-level data

Multilevel data gets more tricky: should we impute taking into account cluster? How do we do that?

For an initial pass, I would recommend simply doing regression imputation *ignoring* cluster/grouping, and then adding in that dummy variable of whether a value is imputed.

10.7.3 Longitudinal data

With longitudinal data we can often use all our data even for individuals with missing data on the outcome, if we assume data are MAR (“Missing at Random”). MAR means that conditional on the observed data, missingness may depend on any observed data, but not on unobserved data. we explore our missing data on individuals over time and on outcomes as above to get a sense of whether MAR is a reasonable assumption or not. Then `lmer` basically handles the rest for us, as far as we have enough observations per individual, on average, to estimate the number of random effects we are trying to estimate. With respect to missing data on covariates or predictors, you can handle those with one of the methods described above.

Here we show how to explore missing data in longitudinal analysis using data on toenail detachment, which you will see in the unit on generalized MLMs. The data is from a RCT where patients were getting a different type of drug to prevent toenail detachment (the outcome).

```
## load data
toes = foreign::read.dta( "data/toenail.dta" )
```

First, let's look at how many times patients were observed.

```
## how many time points per patient?
table( table( toes$patient ) )
```

1	2	3	4	5	6	7
5	3	7	6	10	39	224

We have 224 patients observed at all 7 time points, and the rest of the patients are observed at fewer time points, between 1 and 6.

```

## define function
summarise.patient = function( patient ) {
  pat = rep( ".", 7 )
  pat[ patient$visit ] = 'X'
  paste( pat, collapse="" )
}

## For each patient, this code makes a string of "."
## then it replaces all dots with an "X" if we have data for that visit

## summarize missingness
miss = toes %>% group_by( patient ) %>%
  do( pattern = summarise.patient(.) ) %>%
  unnest(cols = c(pattern))
## Group the data by patient
## Then use the do() command on each chunk of our dataframe
## The "." means "the chunk" (it is a pronoun, essentially).
## This code creates a list of character vectors
## The unnest() takes our character vector out of this list made by "do"

head( miss )

# A tibble: 6 x 2
patient pattern
<dbl> <chr>
1      1 XXXXXX
2      2 XXXXX.
3      3 XXXXXX
4      4 XXXXXX
5      6 XXXXXX
6      7 XXXXXX

```

Here we see the different patterns of missing outcomes, i.e., when patients leave and if they come back. When patients leave and never come back, regardless of the time point (see lines 4 and 5), we have monotone missingness.

```

## sort missing patterns in decreasing order
## starting with no missingness
sort( table( miss$pattern ), decreasing=TRUE )

```

XXXXXXX XXXXX.X XXXX.XX XXX.... X..... XXXXX.. XXXX... XX..... XXX.XXX XXXXX.

224	21	10	6	5	5	4	3	3	3
XXX.X..	XXXX..X	X.XXXXXX	XX..X..	XX.XXX.	XX.XXXX	XXX..XX	XXX.X.X		
2	2	1	1	1	1	1	1		

```
## summarize number of data patterns
miss = miss %>% group_by( pattern ) %>%
  summarise( n=n() )
miss = arrange( miss, -n )
miss

# A tibble: 18 x 2
  pattern     n
  <chr>   <int>
1 XXXXXXXX    224
2 XXXXX.X     21
3 XXXX.XX     10
4 XXX....      6
5 X.....       5
6 XXXXX..      5
7 XXXX...      4
8 XX.....      3
9 XXX.XXX      3
10 XXXXXX.     3
11 XXX.X..     2
12 XXXX..X     2
13 X.XXXXXX    1
14 XX..X..     1
15 XX.XXX.     1
16 XX.XXXX     1
17 XXX..XX     1
18 XXX.X.X     1

## percent missing data (224 complete cases)
224 / sum( miss$n )
```

[1] 0.762

```
## 76% of patients with complete data
```

Second, we look at patterns of missing outcomes. The outcome here is toenail detachment.

```

## reshape data to wide
dat.wide = reshape( toes2, direction="wide", v.names="outcome",
                     idvar="patient", timevar = "visit" )
head( dat.wide )

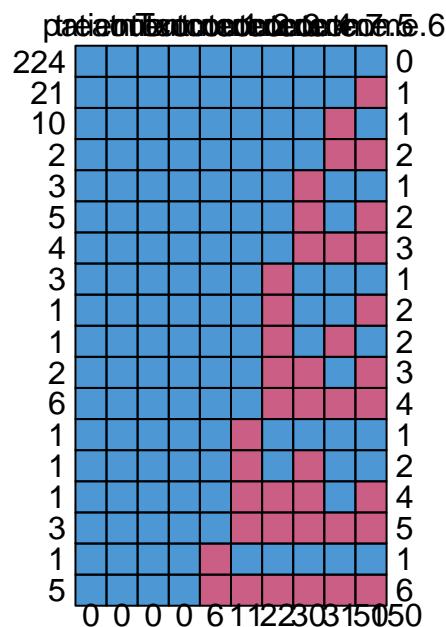
```

patient	treatment	Tx	outcome.1	outcome.2	outcome.3	outcome.4
1	1	Itraconazole	1	1	1	0
8	2	Terbinafine	0	0	1	1
14	3	Terbinafine	0	0	0	0
21	4	Terbinafine	1	0	0	0
28	6	Itraconazole	1	1	1	0
35	7	Itraconazole	0	0	0	0
			outcome.5	outcome.6	outcome.7	
1	0	0	0			
8	0	0	NA			
14	0	0	1			
21	0	0	0			
28	0	0	0			
35	1	1	1			

```

## looking at missing data with mice package
md.pattern( dat.wide )

```



	patient	treatment	Tx	outcome.1	outcome.2	outcome.3	outcome.4	outcome.7
224	1		1	1	1	1	1	1
21	1		1	1	1	1	1	1
10	1		1	1	1	1	1	1
2	1		1	1	1	1	1	1
3	1		1	1	1	1	1	0
5	1		1	1	1	1	1	0
4	1		1	1	1	1	1	0
3	1		1	1	1	1	0	1
1	1		1	1	1	1	0	1
1	1		1	1	1	1	0	1
2	1		1	1	1	1	0	0
6	1		1	1	1	1	0	0
1	1		1	1	1	0	1	1
1	1		1	1	1	0	1	0
1	1		1	1	1	0	0	0
3	1		1	1	1	0	0	0
1	1		1	1	1	1	1	1
5	1		1	1	1	0	0	0
	0		0	0	0	6	11	22
								30
	outcome.5		outcome.6					
224	1		1	0				
21	1		0	1				
10	0		1	1				
2	0		0	2				
3	1		1	1				
5	1		0	2				
4	0		0	3				
3	1		1	1				
1	1		0	2				
1	0		1	2				
2	1		0	3				
6	0		0	4				
1	1		1	1				
1	1		1	2				
1	1		0	4				
3	0		0	5				
1	1		1	1				
5	0		0	6				
	31		50	150				

```

## Another way to generating missingness patterns is to create a function
## This function takes the visits and outcomes and puts a 1 or 0 if there is an
## outcome and a dot if missing.
make.pat = function( visit, outcome ) {
  pat = rep( ".", 7 )
  pat[ visit ] = outcome
  paste( pat, collapse="" )
}

## call our function on all our patients.
outcomes = toes %>% group_by( patient ) %>%
  summarise( tx = Tx[[1]],
             num.obs = n(),
             num.detach = sum( outcome ),
             out = make.pat( visit, outcome ) )

head( outcomes, 20 )

```

	patient	tx	num.obs	num.detach	out
	<dbl>	<fct>	<int>	<dbl>	<chr>
1	1	Itraconazole	7	3	1110000
2	2	Terbinafine	6	2	001100.
3	3	Terbinafine	7	1	0000001
4	4	Terbinafine	7	1	1000000
5	6	Itraconazole	7	3	1110000
6	7	Itraconazole	7	3	0000111
7	9	Itraconazole	7	0	0000000
8	10	Terbinafine	7	0	0000000
9	11	Itraconazole	7	4	1111000
10	12	Terbinafine	7	3	0100110
11	13	Terbinafine	7	4	1111000
12	15	Itraconazole	6	2	11000.0
13	16	Itraconazole	6	1	0000.10
14	17	Terbinafine	6	4	11110.0
15	18	Terbinafine	6	1	001.000
16	19	Itraconazole	7	0	0000000
17	20	Terbinafine	6	0	000.000
18	21	Itraconazole	3	2	110....
19	22	Terbinafine	7	3	1110000
20	23	Itraconazole	7	0	0000000

```
## how many folks have no detachments?
table( outcomes$num.detach )
```

```
0   1   2   3   4   5   6   7
163 25  25  31  30  8   4   8
```

```
163 / nrow(outcomes)
```

```
[1] 0.554
```

```
## how many always detached?
sum( outcomes$num.detach == outcomes$num.obs )
```

```
[1] 16
```

```
16 / nrow(outcomes)
```

```
[1] 0.0544
```

10.8 Further reading

Some further reading on handling missing data. But this is really a course into itself.

- Gelman & Hill Chapter 25 has a more detailed discussion of missing data imputation.
- White IR, Royston P, Wood AM. Multiple imputation using chained equations: issues and guidance for practice. *Statistics in Medicine* 2011;30: 377-399.
- Graham, JW, Olchowski, AE, Gilreath, TD, 2007. How Many Imputations are Really Needed? Some Practical Clarifications of Multiple Imputation Theory 206–213. <https://doi.org/10.1007/s11121-007-0070-9>
- van Buuren S, Groothuis-Oudshoorn K, MICE: Multivariate Imputation by Chained Equations. *Journal of Statistical Software*. 2011;45(3):1-68.
- Grund S, Lüdtke O, Robitzsch A. Multiple Imputation of Missing Data for Multilevel Models: Simulations and Recommendations. DOI: 10.1177/1094428117703686

10.9 Appendix: More about the mice package

The mice package gives back a very complex object that has a lot of information about how values were imputed, which values were imputed, and so forth. In the following we unpack the `imp` variable from above a bit more.

Looking at the imputation object

In the following code, we look at the object we get back from `mice()`. It has lots of parts that we can peek into.

First, the `imp` list inside of `imp` stores all of our newly imputed data. It is itself a list of each variable with their imputed values:

```
imp$imp
```

```
$Ozone
      1   2   3   4   5
 5     6  19  14   8  14
10    12  12   7  23  23
25    14  19  14  19  14
26    37  18  32  32  18
27    11   1  18  13  18
32    65  45  13  28  29
33    22  36  12  18  16
34    13  18   1  13  13
35    63  35  45  52  71
36    23  39  20  59  96
37    24  16  12  34  18
39    64  135  85  80  91
42   115  76  115  37  91
43    66  122  78  64  122
45    44  28  45  23  16
46    23  45  46  45  35
52    20  52  63  47  47
53    59  59  48  115  37
54    40  16  35  37  63
55    40  35  48  39  49
56    23  39  59  59  16
57    44  52  40  52  20
58    30  30  27  14  23
59    45  32  16  16  46
60    44  27  34  28  30
```

```
61   89   64   80   37   64
65   16   16   14   23   29
72   46   52   65   45   35
75   35   64   71   18   78
83   20   40   71   46   59
84   28   63   37   29   63
102  115  78   78   37   66
103  46   29   31   23   40
107  16   30   13   14   22
115  41   12   44   7    22
119  50   78   122  85   50
150  24   12   27   21   12
```

```
$Solar.R
```

```
 1   2   3   4   5
5    7  313  82  13  314
6  322 187 222  24 238
11  66 274 139 135 112
27  20  24   7 238 193
96 175 223 284 197 220
97  51 139 274 237  98
98  98 203 220 188 276
```

```
$Wind
```

```
[1] 1 2 3 4 5
<0 rows> (or 0-length row.names)
```

```
$Temp
```

```
[1] 1 2 3 4 5
<0 rows> (or 0-length row.names)
```

```
$Month
```

```
[1] 1 2 3 4 5
<0 rows> (or 0-length row.names)
```

```
$Day
```

```
[1] 1 2 3 4 5
<0 rows> (or 0-length row.names)
```

```
str( imp$imp )
```

```
List of 6
```

```

$ Ozone  :'data.frame':   37 obs. of  5 variables:
..$ 1: int [1:37] 6 12 14 37 11 65 22 13 63 23 ...
..$ 2: int [1:37] 19 12 19 18 1 45 36 18 35 39 ...
..$ 3: int [1:37] 14 7 14 32 18 13 12 1 45 20 ...
..$ 4: int [1:37] 8 23 19 32 13 28 18 13 52 59 ...
..$ 5: int [1:37] 14 23 14 18 18 29 16 13 71 96 ...
$ Solar.R:'data.frame':    7 obs. of  5 variables:
..$ 1: int [1:7] 7 322 66 20 175 51 98
..$ 2: int [1:7] 313 187 274 24 223 139 203
..$ 3: int [1:7] 82 222 139 7 284 274 220
..$ 4: int [1:7] 13 24 135 238 197 237 188
..$ 5: int [1:7] 314 238 112 193 220 98 276
$ Wind    :'data.frame':    0 obs. of  5 variables:
..$ 1: logi(0)
..$ 2: logi(0)
..$ 3: logi(0)
..$ 4: logi(0)
..$ 5: logi(0)
$ Temp    :'data.frame':    0 obs. of  5 variables:
..$ 1: logi(0)
..$ 2: logi(0)
..$ 3: logi(0)
..$ 4: logi(0)
..$ 5: logi(0)
$ Month   :'data.frame':    0 obs. of  5 variables:
..$ 1: logi(0)
..$ 2: logi(0)
..$ 3: logi(0)
..$ 4: logi(0)
..$ 5: logi(0)
$ Day     :'data.frame':    0 obs. of  5 variables:
..$ 1: logi(0)
..$ 2: logi(0)
..$ 3: logi(0)
..$ 4: logi(0)
..$ 5: logi(0)

```

```
str( imp$imp$Ozone )
```

```

'data.frame':   37 obs. of  5 variables:
$ 1: int  6 12 14 37 11 65 22 13 63 23 ...
$ 2: int  19 12 19 18 1 45 36 18 35 39 ...

```

```
$ 3: int 14 7 14 32 18 13 12 1 45 20 ...
$ 4: int 8 23 19 32 13 28 18 13 52 59 ...
$ 5: int 14 23 14 18 18 29 16 13 71 96 ...
```

We see that Ozone and Solar.R have imputed values, and the other variables do not.

Next, we see two missing observations in our original data and then see the two imputed values for these two missing observations.

```
airqualitysub$Ozone
```

```
[1] 41 36 12 18 NA 28 23 19 8 NA
```

```
imp$imp$Ozone[,1]
```

```
[1] 6 12 14 37 11 65 22 13 63 23 24 64 115 66 44 23 20 59 40
[20] 40 23 44 30 45 44 89 16 46 35 20 28 115 46 16 41 50 24
```

We can make (the hard way) a vector of Ozone by plugging our missing values into the original data. But the `complete()` method, above, is preferred.

```
oz = airqualitysub$Ozone
oz[is.na(oz)] = imp$imp$Ozone[,1]
```

```
Warning in oz[is.na(oz)] = imp$imp$Ozone[, 1]: number of items to replace is
not a multiple of replacement length
```

```
oz
```

```
[1] 41 36 12 18 6 28 23 19 8 12
```

What else is there in `imp`?

```
names(imp)
```

```
[1] "data"           "imp"            "m"              "where"
[5] "blocks"          "call"           "nmis"           "method"
[9] "predictorMatrix" "visitSequence" "formulas"        "post"
[13] "blots"          "ignore"         "seed"           "iteration"
[17] "lastSeedValue"  "chainMean"     "chainVar"       "loggedEvents"
[21] "version"         "date"
```

What was our imputation method?

```
imp$method
```

Ozone	Solar.R	Wind	Temp	Month	Day
"pmm"	"pmm"	""	""	""	""

Mean imputation for each variable with missing values. Later this will say other thing.

What was used to impute what?

```
imp$predictorMatrix
```

	Ozone	Solar.R	Wind	Temp	Month	Day
Ozone	0	1	1	1	1	1
Solar.R	1	0	1	1	1	1
Wind	1	1	0	1	1	1
Temp	1	1	1	0	1	1
Month	1	1	1	1	0	1
Day	1	1	1	1	1	0

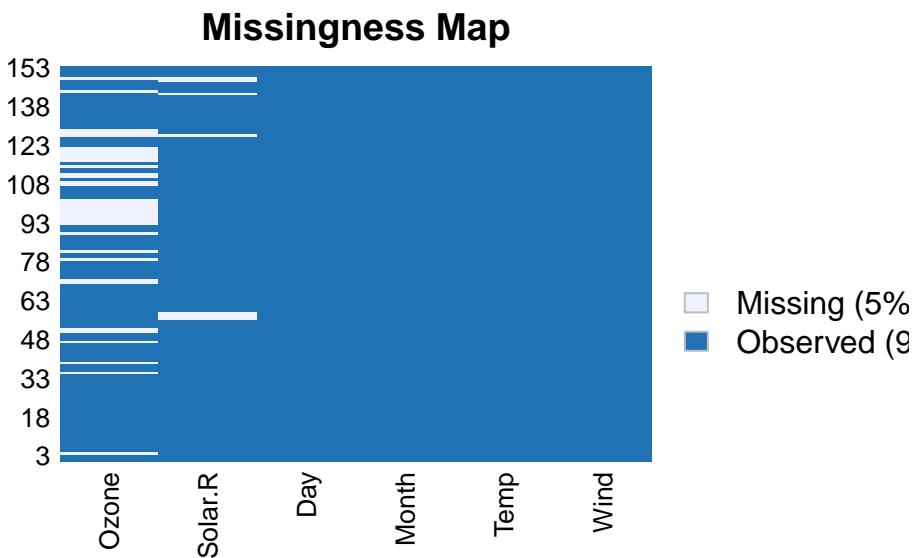
10.10 Appendix: The amelia package

Amelia is another multiple imputation and missing data package. We do not prefer it, but have some demonstration code in the following, for reference.

```
library(Amelia)
```

For missingness we can make the following:

```
missmap(airquality)
```



Each row of the plot is a row of the data, and missing values are shown in brown. But ugly!
And hard to see any trends in the missingness.

You can use the `Amelia` package to do mean imputation.

```
library(dplyr)

## exclude variables that do not vary
a.airquality = airquality %>% dplyr::select(-Month)

## impute data
a.imp <- amelia(a.airquality, m=5)
```

-- Imputation 1 --

```
1 2 3 4 5 6
```

-- Imputation 2 --

```
1 2 3 4 5 6
```

-- Imputation 3 --

```

1 2 3 4 5

-- Imputation 4 --

1 2 3 4 5 6 7

-- Imputation 5 --

1 2 3 4 5 6

a.imp

```

Amelia output with 5 imputed datasets.

Return code: 1

Message: Normal EM convergence.

Chain Lengths:

```

-----
Imputation 1: 6
Imputation 2: 6
Imputation 3: 5
Imputation 4: 7
Imputation 5: 6

```

We can plot our imputed values against our observed values to check that they make sense. We will do this for just one of five datasets we just imputed using **Amelia**.

```

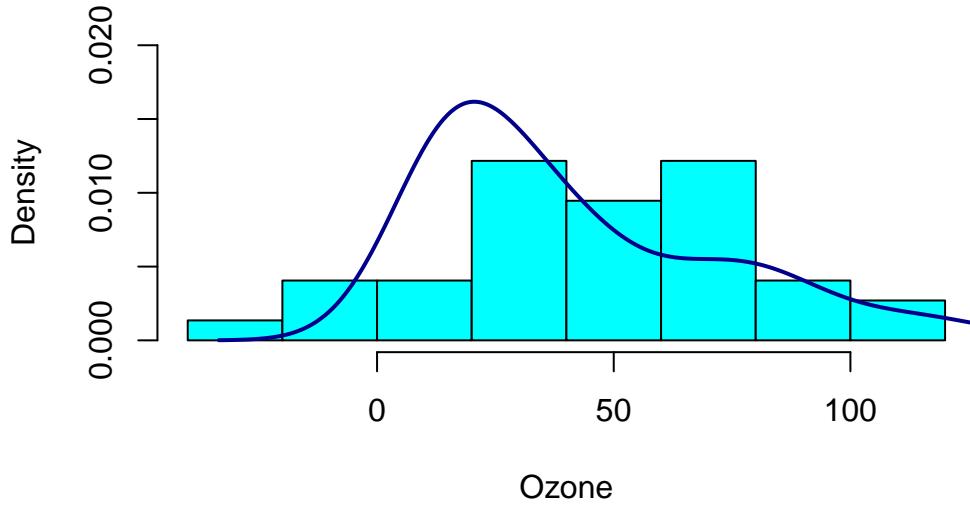
## put imputed values from the third dataset in an object
one_imp <- a.imp$imputations[[3]]$Ozone

## make object with observed values
## from observations without missing Ozone values
obs_data <- a.airquality$Ozone

## make a plot overlaying observed and imputed values
hist(one_imp[is.na(obs_data)], prob=TRUE, xlab="Ozone",
     main="Histogram of Imputed Values in 3rd Imputation \nCompared to Density in Observed
     col="cyan", ylim=c(0,0.02))
lines(density(obs_data[!is.na(obs_data)]), col="darkblue", lwd=2)

```

Histogram of Imputed Values in 3rd Imputation Compared to Density in Observed Data



You can also do multiple imputation in `Amelia`. However, `Amelia` does not have an easy way to combine the estimates from the imputed datasets (no analogue of `with()` in `mice`). You can write a function that fits your model of interest in each imputed dataset and then use a package like `mitools` to pool the estimates and variances.

Much easier to use `mice`!

Aside: A more important limitation of `Amelia` is that the algorithm it uses to impute missing values assumes multivariate normality, which is often questionable, especially when you have binary variables.

11 Tips, Tricks, and Debugging in R

This chapter is a complete hodge-podge of small things that tend to bite students, especially when they are dealing with messy data that they might have for final projects.

In particular, as you learn R, there's lots of good tricks you'll never know about until somebody shows you. Clean code is one such good trick; consider the following: "Your most important collaborator is you from 6 months ago. Unfortunately, you can't ask that-you any questions, because they don't answer their email."

So onwards with a few things that you may find useful, either now or later.

11.1 Some principles to live by

11.1.1 Watch Tricky letter and number confusion in code

The letter "l" looks like the number "1"—watch out for that. Things like "mylm" are usually all letters, with "lm" standing for linear model.

11.1.2 Write in a good R style

Try to do the following

- Comment your code!
- Structure your R file like so:
 - Descriptive comments (including date)
 - Load libraries
 - Constants and script parameters (# iterations, etc.)
 - Functions (with descriptive comment after first line)
 - Everything else
- Naming: variableName / variable.name, FunctionVerb, kConstantName. not_like_this
- Indentation: 2-space indents is nice
- Spaces are ok, but don't go overboard. E.g., `y = (a * x) + b + rnorm(1, sd=sigma)`
- Never use `attach()`

11.1.3 Save and load R objects to save time

If you have the result of something that took awhile to run (e.g., a big multilevel model fit to a lot of data) you can try saving it like so:

```
myBigThing = lm(mpg ~ disp, data=mtcars) #something slow
saveRDS(myBigThing, savedPath)

## Later on:
myBigThing <- readRDS(savedPath)
```

11.1.4 Reproduce randomness with set.seed

If your code uses random numbers, then you should set your seed, which makes your script always generate the same sequence of random numbers.

For example, say your code had this:

```
tryCatch({(1:(1:10)[rpois(1, 3)])}, error=function(e){(e)}) #works?
```

```
[1] 1 2 3
```

```
set.seed(97)
tryCatch({(1:(1:10)[rpois(1, 3)])}, error=function(e){(e)}) #fails!
```

```
<simpleError in 1:(1:10)[rpois(1, 3)]: argument of length 0>
```

(Note the `tryCatch()` method is a way of generating errors and not crashing.)

Key thing to know: **Reproducible results help with debugging.**

If you want to get fancy, try this (after installing the ‘TeachingDemos’ package):

```
TeachingDemos::char2seed("quinn")
# Using your name as a seed says "nothing up my sleeve"
```

11.1.5 Keep your files organized

Ever seen this?

- /My Documents
 - my paper.tex
 - my paper draft 2.tex
 - my paper final.tex
 - my paper final revised.tex
 - my paper final revised 2.tex
 - script.r
 - script 2.r
 - data.csv

Try instead something like:

- /stat 166-Small Data Analysis
 - stat 166.rproj
 - /Empty Project
 - * /code
 - * /data
 - * /text
 - * /figures
 - * readme.txt
 - /HW1
 - * ...

Your `readme.txt` might have informational notes such as “Got data from bit.ly/XYZ.” to remind you of what you were up to.

Your `figures` folder should be full of figures you can easily regenerate with code in your `code` folder.

11.1.6 Make sure your data are numeric

Sometimes when you load data in, R does weird things like decide all your numbers are actually words. This happens if some of your entries are not numbers. Then R makes them all not numbers. You can check this with the `str()` function:

```
str( exp.dat )
```

```
'data.frame': 4 obs. of 10 variables:  
 $ ID    : chr  "a" "b" "c" "d"  
 $ cond   : chr  "AI" "DI" "DI" "AI"  
 $ trial1: chr  "E" "U" "U" "E"  
 $ dec1   : num  1 1 0 1  
 $ trial2: chr  "U" "E" "U" "E"  
 $ dec2   : num  0 0 0 1  
 $ trial3: chr  "U" "E" "E" "U"  
 $ dec3   : num  0 1 0 1  
 $ trial4: chr  "E" "U" "E" "U"  
 $ dec4   : num  0 1 0 0
```

Here we see that we have factors (categorical variables) and numbers (num). All is well.

If something should be a number, then change it like so:

```
lst <- c( 1, 2, 3, "dog", 5, 6 )  
str( lst )
```

```
chr [1:6] "1" "2" "3" "dog" "5" "6"
```

```
lst <- as.numeric( lst )
```

Warning: NAs introduced by coercion

```
lst
```

```
[1] 1 2 3 NA 5 6
```

```
str( lst )
```

```
num [1:6] 1 2 3 NA 5 6
```

Note it warned you that you had non-numbers when you converted. The non-numbers are now missing (NA).

For a dataframe, you fix like this:

```
exp.dat
```

Warning: NAs introduced by coercion

11.1.7 Categories should be words

For categorical variables, don't use numbers, if at all possible. E.g.,

```
levels = c( "Low", "Middle", "High", NA )
```

is better than

```
levels = c(1, 2, 3, 99 )
```

11.2 Data Wrangling

We next give some high level data wrangling advice. But really, check out R for DS for much more and much better on the merging and summarizing topics.

11.2.1 Handling Lagged Data

Sometimes you have multiple times for your units (think country or state), and you want to regress, say, future X on current X. Then you want to have both future and current X for each case.

Here think of a case as a Country at a point in time. E.g., we might have data like this:

```
dtw = read.csv( "data/fake_country_block.csv", as.is=TRUE )
dt = pivot_longer( dtw, cols=X1997:X2004,
                  names_to = "Year", names_prefix = "X",
                  values_to = "X" )
dt$Year = as.numeric( dt$Year )
slice_sample( dt, n=5 )
```

```
# A tibble: 5 x 3
  Country   Year     X
  <chr>    <dbl> <dbl>
1 China     2000    3.4
2 England   1999    53
3 China     2003     6
4 Morocco   1997   31.9
5 England   2003   57.3
```

We then want to know what the X will be 2 years in the future. We can do this with the following trick:

```
dt.fut = dt
dt.fut$Year = dt.fut$Year - 2
head(dt.fut)
```

```
# A tibble: 6 x 3
  Country   Year     X
  <chr>    <dbl> <dbl>
1 China     1995    0.5
2 China     1996    1
3 China     1997    2
4 China     1998    3.4
5 China     1999    4
6 China     2000    5.3
```

```
newdt = left_join( dt, dt.fut,
                    by=c("Country", "Year"), suffix=c("", ".fut") )
head( newdt, 10 )
```

```
# A tibble: 10 x 4
  Country   Year     X X.fut
  <chr>    <dbl> <dbl> <dbl>
1 China     1997    0.5    2
2 China     1998    1      3.4
3 China     1999    2      4
4 China     2000    3.4    5.3
5 China     2001    4      6
6 China     2002    5.3    7
7 China     2003    6      NA
8 China     2004    7      NA
```

```

9 Morocco 1997 31.9 33
10 Morocco 1998 32     34

```

Here we are merging records that match *both* Country and Year.

Note that for the final two China entries, we don't have a future X value. The merge will make it NA indicating it is missing.

How this works: we are tricking the program. We are making a new \verb|dt.lag| data.frame and then putting all the entries into the past by two years. When we merge, and match on Country and Year, the current dataframe and the lagged dataframe get lined up by this shift. Clever, no?

Now we could do regression:

```

my.lm = lm( X.fut ~ X + Country, data=newdt )
summary( my.lm )

```

Call:

```
lm(formula = X.fut ~ X + Country, data = newdt)
```

Residuals:

Min	1Q	Median	3Q	Max
-0.5869	-0.2610	0.0107	0.2753	0.5137

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	1.8684	0.2128	8.78	2.7e-06 ***
X	1.0179	0.0582	17.48	2.3e-09 ***
CountryEngland	-0.8259	2.9704	-0.28	0.79
CountryMorocco	-0.7514	1.7603	-0.43	0.68

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.351 on 11 degrees of freedom
 (9 observations deleted due to missingness)

Multiple R-squared: 1, Adjusted R-squared: 1
 F-statistic: 2.13e+04 on 3 and 11 DF, p-value: <2e-16

11.2.2 Quick overview of merging data

Often you have two datasets that you want to merge. For example, say you want to merge some data you have on a few states with some SAT information from the mosaic package.

```
library( mosaicData )
data( SAT )
head( SAT )
```

```
      state expend ratio salary frac verbal math sat
1   Alabama    4.41  17.2   31.1     8    491   538 1029
2    Alaska     8.96  17.6   48.0    47    445   489  934
3  Arizona     4.78  19.3   32.2    27    448   496  944
4 Arkansas     4.46  17.1   28.9     6    482   523 1005
5 California    4.99  24.0   41.1    45    417   485  902
6 Colorado     5.44  18.4   34.6    29    462   518  980
```

```
df = data.frame( state=c("Alabama","California","Fakus"),
                 A=c(10,20,50),
                 frac=c(0.5, 0.3, 0.4) )
df
```

```
      state A frac
1   Alabama 10  0.5
2 California 20  0.3
3       Fakus 50  0.4
```

```
merge( df, SAT, by="state", all.x=TRUE )
```

```
      state A frac.x expend ratio salary frac.y verbal math sat
1   Alabama 10     0.5    4.41  17.2   31.1     8    491   538 1029
2 California 20     0.3    4.99  24.0   41.1    45    417   485  902
3       Fakus 50     0.4      NA     NA      NA     NA      NA     NA    NA
```

The records are combined by the “by” variable. I.e., each record in df is matched with each record in SAT with the same value of “state.”

Things to note: If you have the same variable in each dataframe, it will keep both, and add a suffix of “.x” and “.y” to indicate where they came from.

The “all.x” means keep all records from your first dataframe (here df) even if there is no match. If you added “all.y=TRUE” then you would get all 50 states from the SAT dataframe even though df doesn’t have most of them. Try it!

You can merge on more than one variable. I.e., if you said `\verb|by=c("A","B")|` then it would match records if they had the same value for both A and B. See below for an example on this.

11.2.3 Summarizing/aggregating Data

Sometimes you want to collapse several cases into one. This is called aggregating. If you install a package called “dplyr” (Run `install.packages("dplyr")` once to install, or better yet simply install `tidyverse`) then you will have great power.

Using `newdt` from above, we can summarize countries across all their time points:

```
newdt %>% group_by( Country ) %>%
  summarise( mean.X = mean(X, na.rm=TRUE) ,
             sd.X = sd( X, na.rm=TRUE ) )
```

```
# A tibble: 3 x 3
  Country mean.X  sd.X
  <chr>     <dbl> <dbl>
1 China      3.65  2.37
2 England    54.6   2.43
3 Morocco    34.0   2.12
```

You can also augment data by adding new variables. You can even do this within groups. Here we subtract the mean from each group:

```
dshift = newdt %>% group_by( Country ) %>%
  mutate( Xm = mean(X, na.rm=TRUE),
         Xc = X - mean(X, na.rm=TRUE) )
head(dshift)
```

```
# A tibble: 6 x 6
# Groups:   Country [1]
  Country  Year     X X.fut     Xm     Xc
  <chr>    <dbl> <dbl> <dbl> <dbl> <dbl>
1 China     1997    0.5    2     3.65 -3.15
2 China     1998    1      3.4    3.65 -2.65
```

3	China	1999	2	4	3.65	-1.65
4	China	2000	3.4	5.3	3.65	-0.25
5	China	2001	4	6	3.65	0.35
6	China	2002	5.3	7	3.65	1.65

11.2.4 Making Data Frames on the fly

For small datasets, you can type in data the hard way like so:

```
exp.dat = data.frame( ID=c("a","b","c","d"),
  cond = c("AI","DI","DI","AI"),
  trial1 = c("E","U","U","E"),
  dec1 = c(1,1,0,1),
  trial2 = c("U","E","U","E"),
  dec2 = c(0,0,0,1),
  trial3 = c("U","E","E","U"),
  dec3 = c(0,1,0,1),
  trial4 = c("E","U","E","U"),
  dec4 = c(0,1,0,0) )
exp.dat
```

	ID	cond	trial1	dec1	trial2	dec2	trial3	dec3	trial4	dec4
1	a	AI	E	1	U	0	U	0	E	0
2	b	DI	U	1	E	0	E	1	U	1
3	c	DI	U	0	U	0	E	0	E	0
4	d	AI	E	1	E	1	U	1	U	0

This is for an experiment on 4 subjects. The first and forth subject got the AI treatment, the second two got the DI treatment. The subjects then had 4 trials each, and they received a “E” choice or a “U” choice, and the decision variable is whether they accepted the choice.

As you can see, data can get a bit complicated!

Part II

USING ggPLOT

12 Intro to ggplot

This is a simple script demonstrating some powerful features of a plotting package `ggplot2`.

`ggplot` can initially seem like a nightmare to some, but once you wrestle it to the ground it is one of the most powerful visualization tools you might have in your toolbox. Happily, it is fairly easy to get some basics up and running once you start looking at the world the way it does. Let's start doing that.

First, `ggplot` thinks of a plot as a collection of layers stacked on top of each other. The way this looks in code is a bunch of weird function calls connected together with `+`. You read this series of calls left to right. The first call is always a statement saying what data you are plotting and what variables you care about. So before you can even plot, you need to make sure your data are in a nice, tidy data frame.

Happily, when you load data, it usually is. For example:

```
dat <- read_dta("data/hsb.dta") |>
  select(1:10)

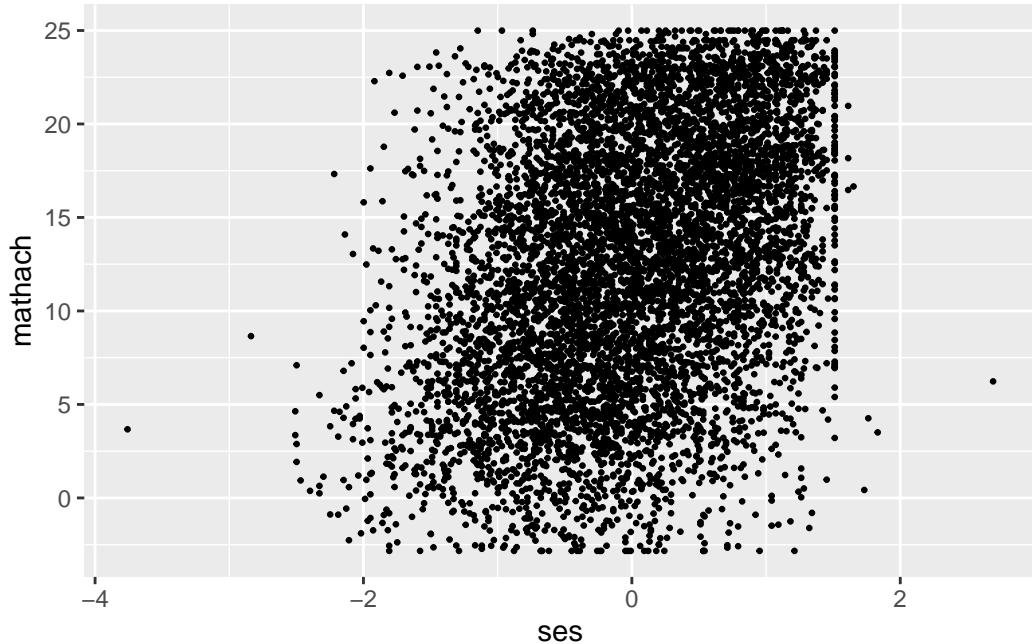
names( dat )
```



```
[1] "minority"  "female"     "ses"        "mathach"    "size"       "sector"
[7] "pracad"    "disclim"    "himinty"    "schoolid"
```

The easiest full plot to make has two elements. The first gives what your variables are, and the second says how to plot them:

```
ggplot(dat, aes(y = mathach, x = ses)) +
  geom_point( cex=0.5)
```

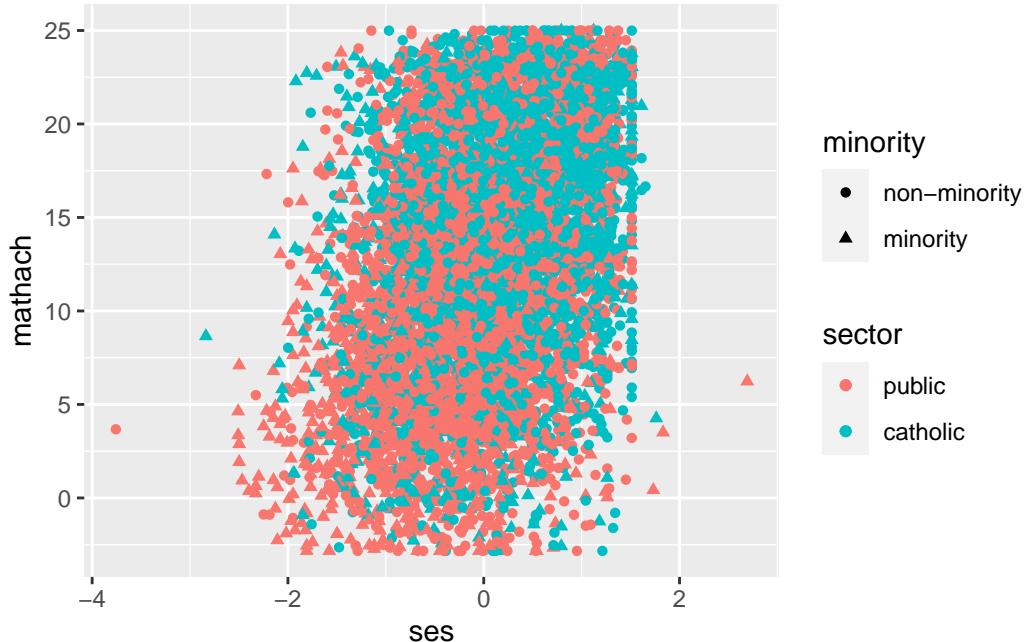


So far, nothing too scary, right? The `ggplot(dat, aes(x=mathach, y=ses))` says “My plot is going to use `dat` for my data, and my y-axis is the `mathach` variable and my x-axis is `ses`.” The `aes()` bit is “aesthetics”—it is a way of tying variables to different kinds of things you could have on your plot: x location, y location, color, plotting symbol, and a few other things.

For example:

```
dat <- dat |>
  mutate(sector = factor(sector, levels=c(0,1), labels=c("public","catholic")),
         minority = factor(minority, labels=c("non-minority","minority")))

ggplot( dat, aes(y=mathach, x=ses, col=sector, pch=minority) ) +
  geom_point()
```

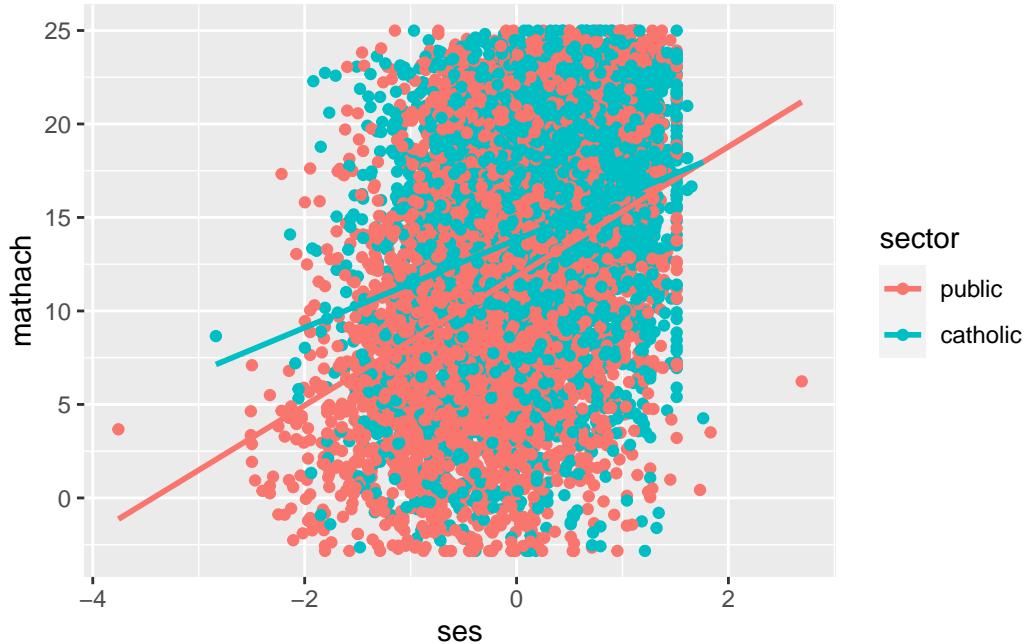


Note that `ggplot` wants the data frame to be neatly put together, including that categorical variables are listed as factors. This is why we convert the dummy `sector` to a factor above. Once you do this, however, it will label things in a nice way.

12.1 Summarizing

You can also automatically add various statistical summaries, such as simple regression lines:

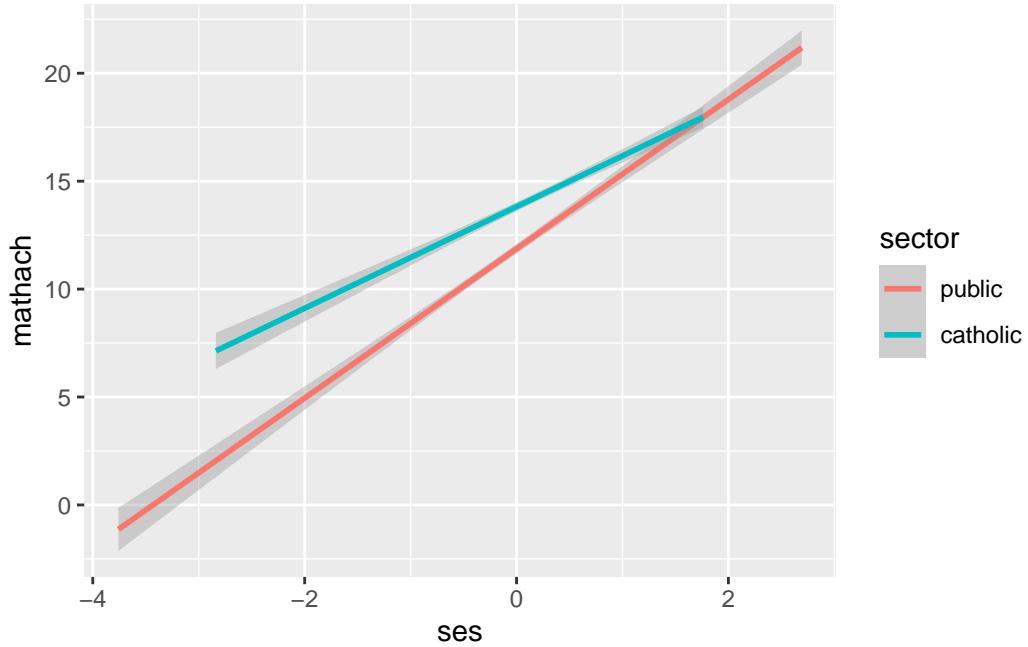
```
ggplot( dat, aes(y=mathach, x=ses, col=sector ) ) +
  geom_point() +
  stat_smooth( method="lm", se = FALSE )
`geom_smooth()` using formula = 'y ~ x'
```



Notice how it automatically realized you have two subgroups of data defined by sector. It gives you a regression line for each group.

The elements of the plot are stacked, and if you remove one of the elements, it will not appear:

```
ggplot( dat, aes(y=mathach, x=ses, col=sector ) ) +
  stat_smooth( method="lm" )
`geom_smooth()` using formula = 'y ~ x'
```



Here we also added some uncertainty bars around the regression lines by not saying `se = FALSE`. (Including uncertainty is the default.)

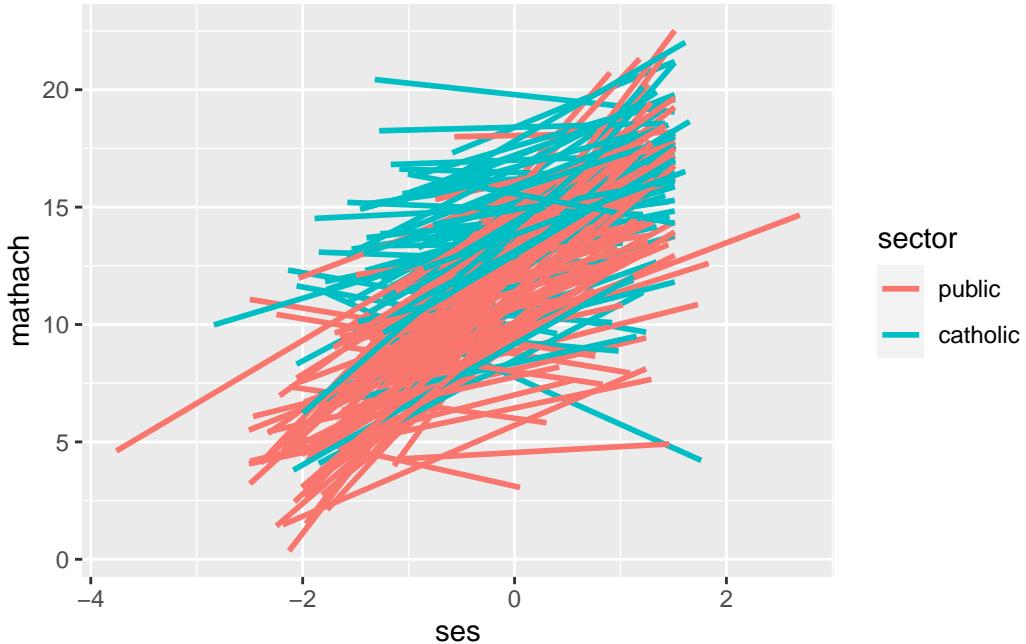
12.2 Grouping

Combining these ideas we can make a trend line for each school:

```
my.plot = ggplot( dat, aes(y=mathach, x=ses, col=sector, group=schoolid ) ) +
  stat_smooth( method="lm", se = FALSE )

my.plot

`geom_smooth()` using formula = 'y ~ x'
```



The trendlines automatically extend to the limits of the data they are run on, hence the different lengths.

Also, notice we “saved” the plot in the variable `my.plot`. Only when we “print” the plot will the plot appear on your display. When we type the name of a variable, it prints. Once you have a plot stored in a variable you can augment it very easily.

As you may now realize, `ggplot2` is very, very powerful.

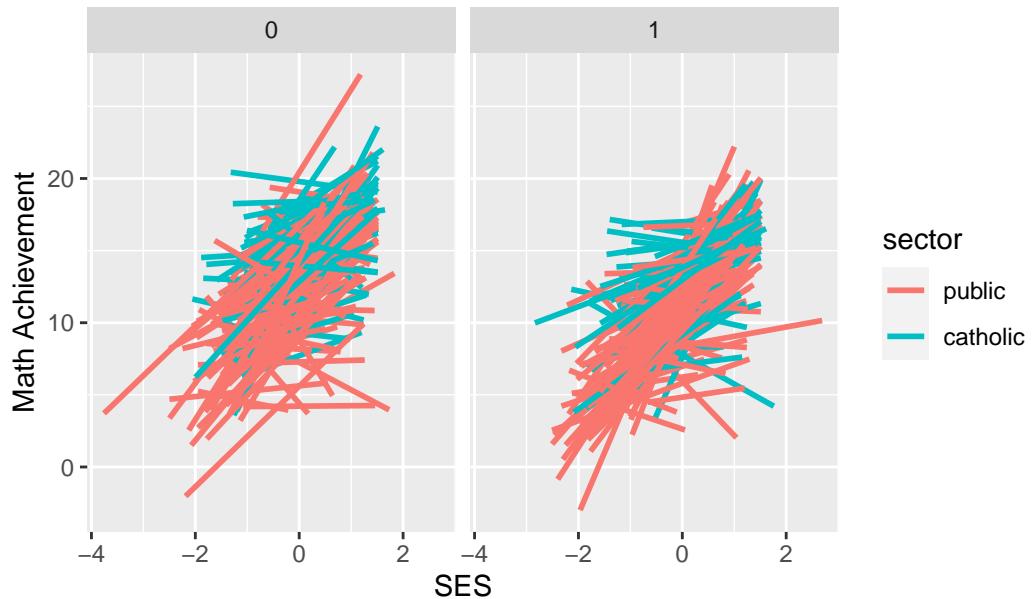
12.3 Customization

We next show some other things you can do. For example, you can make lots of little plots:

```
my.plot +
  facet_grid(~ female) +
  ggttitle("School-level trend lines for their male and female students") +
  labs(x="SES", y="Math Achievement")

`geom_smooth()` using formula = 'y ~ x'
```

School-level trend lines for their male and female students



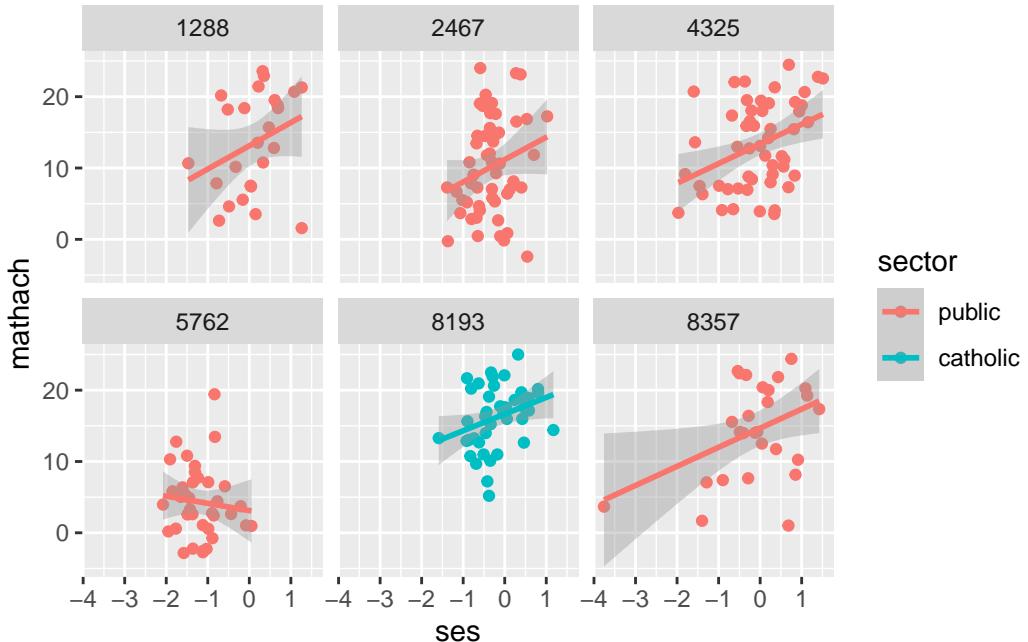
Or,

```
# random subset of schoolid
sch <- sample( unique( dat$schoolid ), 6 )

# pipe into ggplot
my.six.plot <- dat |>
  filter(schoolid %in% sch) |>
  ggplot(aes(y=mathach, x=ses, col=sector ) ) +
  facet_wrap(~ schoolid, ncol=3 ) +
  geom_point() + stat_smooth( method="lm" )

my.six.plot

`geom_smooth()` using formula = 'y ~ x'
```



Also shown are adding titles.

12.4 Themes

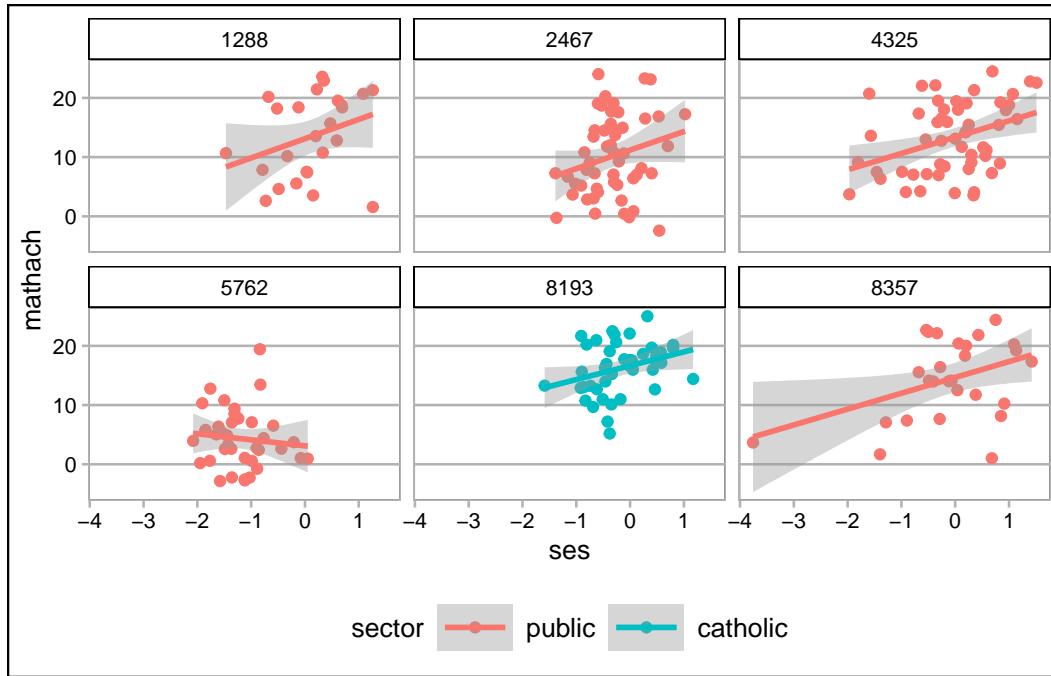
You can very quickly change the entire presentation of your plot using `themes`. There are pre-packaged ones, and you can make your own that you use over and over. Here we set up a theme to be used moving forward

```
library( ggthemes )
my_t = theme_calc() + theme( legend.position="bottom",
                             legend.direction="horizontal",
                             legend.key.width=unit(1,"cm")  )
theme_set( my_t )
```

Compare the same plot from above, now with a new theme.

```
my.six.plot
```

```
`geom_smooth()` using formula = 'y ~ x'
```



Cool, no?

12.5 Next steps

There is a lot of information out there on `ggplot` and my best advice is to find code examples, and then modify them as needed. There are tutorials and blogs that walk through building plots (search for “`ggplot` tutorial” for example), but seeing examples seems to be the best way to learn the stuff. For example, you could use the above code for your project one quite readily. And don’t be afraid to ask how to modify plots on Piazza!

In particular, check out the excellent ‘‘R for Data Science’’ textbook. It extensively uses `ggplot`, starting [here](#).

13 Example of making plots with expand.grid

13.1 Introduction

This script demonstrates using the `predict()` function to make plots with separate lines for different groups. A core element is the `expand.grid()` method. The central idea is this: for each of our groups we manually create a series of points at different levels of our covariate (e.g. ses or time) and then predict the outcome for each of these values. We then plot these predicted points, and it makes a smooth curve for that group.

In this document we start with clustered data (the HS&B dataset) and then illustrate how to this with longitudinal data as well.

13.2 Making plots for the HS&B Dataset

In this section we first look at how to plot the model results by making a tiny dataset from the fixed effects, and then we extend to more powerful plotting of individual schools.

13.2.1 Setting up the HS&B data

The “many small worlds” view says each school has its own regression line. We are going to plot them all. See the lecture code files for how to load the HS&B dataset. For clarity it is omitted from the printout. We end up with this for the schools:

```
head( sdat )
```

	id	size	sector	meanses
1	1224	842	public	-0.428
2	1288	1855	public	0.128
3	1296	1719	public	-0.420
4	1308	716	catholic	0.534
5	1317	455	catholic	0.351
6	1358	1430	public	-0.014

and this for students (we merged in the school info already):

```
head( dat )
```

	id	minority	female	ses	mathach	size	sector	meanses
1	1224	0	1	-1.528	5.876	842	public	-0.428
2	1224	0	1	-0.588	19.708	842	public	-0.428
3	1224	0	0	-0.528	20.349	842	public	-0.428
4	1224	0	0	-0.668	8.781	842	public	-0.428
5	1224	0	0	-0.158	17.898	842	public	-0.428
6	1224	0	0	0.022	4.583	842	public	-0.428

We fit a fancy random slopes model with 2nd level covariates that impact both the overall school means and the ses by math achievement slopes. Our model is

$$\begin{aligned}y_{ij} &= \beta_{0j} + \beta_{1j}ses_{ij} + \epsilon_{ij} \\ \beta_{0j} &= \gamma_{00} + \gamma_{01}sector_j + u_{0j} \\ \beta_{1j} &= \gamma_{10} + \gamma_{11}sector_j + u_{1j}\end{aligned}$$

We omit the equations for the random effect distributions. The ϵ_{ij} are normal, and the (u_{0j}, u_{1j}) are bivariate normal, as usual. We fit the model as so:

```
M1 = lmer( mathach ~ 1 + ses*sector + (1 + ses|id), data=dat )
```

```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.00578925 (tol = 0.002, component 1)
```

```
display( M1 )
```

```
lmer(formula = mathach ~ 1 + ses * sector + (1 + ses | id), data = dat)
```

	coef.est	coef.se
(Intercept)	11.75	0.23
ses	2.96	0.14
sectorcatholic	2.13	0.35
ses:sectorcatholic	-1.31	0.22

Error terms:

Groups	Name	Std.Dev.	Corr
id	(Intercept)	1.95	
	ses	0.28	1.00

```

Residual           6.07
---
number of obs: 7185, groups: id, 160
AIC = 46585.1, DIC = 46557.2
deviance = 46563.2

```

13.2.2 Plotting the model results

We can plot the model results by making a little dataset by hand. This section of the handout illustrates how you can hand-construct plots by directly calculating predicted values from your model. This is a very useful skill, and we recommend studying this area of the handout as a way of learning how to control plotting at a very direct level.

So, to continue, we proceed in three steps.

Step 1: Decide on the plot. Let's make a plot of outcome vs. ses with two lines (one for catholic and one for public). Sometimes it is worth actually sketching the desired plot on scratch paper, identifying the x and y axes and general lines desired.

Step 2: calculate some outcomes using our model. We do this by deciding what values we want to plot, and then making the outcome.

```
quantile( dat$ses, c( 0.05, 0.95 ) )
```

```

5%      95%
-1.318  1.212

```

```

plt = data.frame( ses = c(-1.5, 1.25, -1.5, 1.25 ),
                  catholic = c( 0, 0, 1, 1 ) )
cf = fixef( M1 )
cf

```

	ses	sectorcatholic	ses:sectorcatholic
(Intercept)	11.751789	2.957538	2.129531
			-1.313363

```

plt = mutate( plt,
              Y = cf[[1]] + cf[[2]]*ses + cf[[3]]*catholic + cf[[4]]*ses*catholic )
plt

```

```

ses catholic      Y
1 -1.50          0  7.315482
2  1.25          0 15.448711
3 -1.50          1 11.415057
4  1.25          1 15.936538

```

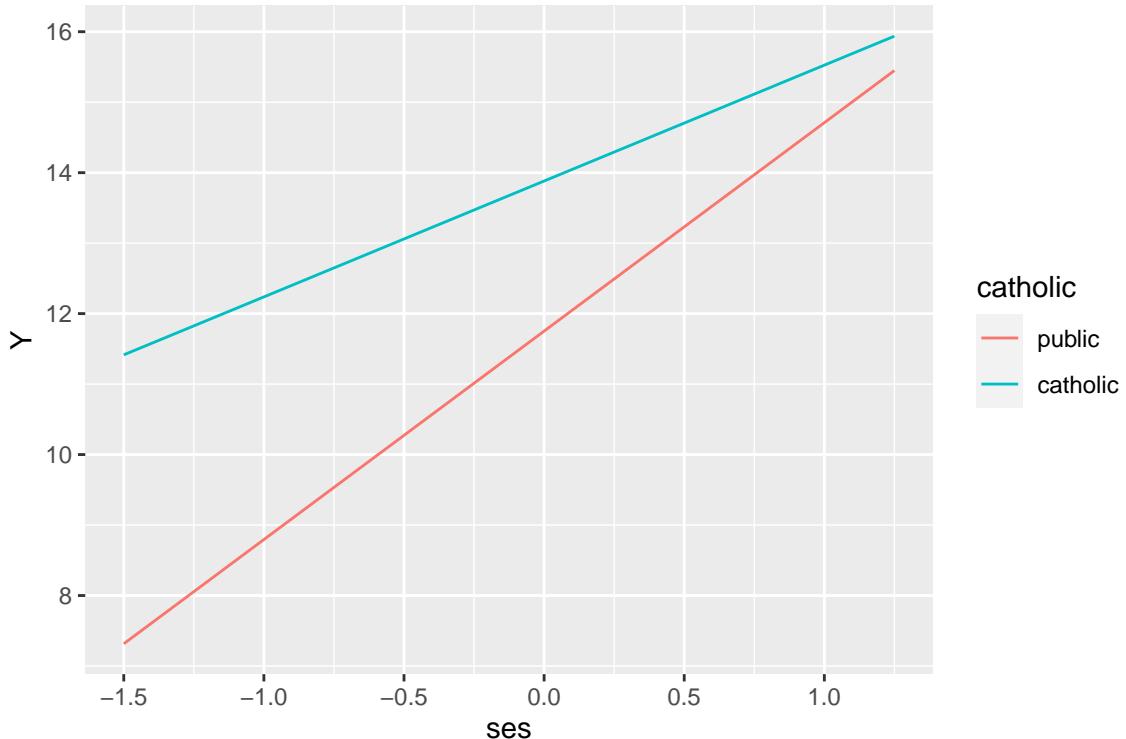
Note that we have made a little mini-dataset with just the points we want to put on our plot. We calculated these points “by hand”. There is no shame in this.

Step 3: plot. We plot using ggplot:

```

plt$catholic = factor( plt$catholic,
                        labels=c("public","catholic"),
                        levels=c(0,1) )
ggplot( plt, aes( ses, Y, col=catholic ) ) +
  geom_line()

```



13.2.2.1 A fancy diversion: categorical variables on the x -axis

Say we decided to fit a model where we have ses **categories**:

```

dat$ses.cat = cut( dat$ses,
                    breaks=quantile( dat$ses, c( 0, 0.33, 0.67, 1 ) ),
                    labels = c( "low","mid","high"),
                    include.lowest = TRUE )
table( dat$ses.cat )

```

```

low  mid high
2371 2462 2352

```

```

M1b = lmer( mathach ~ 1 + ses.cat*sector + (1 + ses|id), data=dat )
display( M1b )

```

```

lmer(formula = mathach ~ 1 + ses.cat * sector + (1 + ses | id),
      data = dat)
              coef.est coef.se
(Intercept)      9.19     0.27
ses.catmid      2.28     0.25
ses.cathigh      5.07     0.29
sectorcatholic   3.44     0.42
ses.catmid:sectorcatholic -0.98    0.38
ses.cathigh:sectorcatholic -2.47    0.42

```

Error terms:

Groups	Name	Std.Dev.	Corr
id	(Intercept)	2.05	
	ses	0.47	0.23
	Residual	6.10	

```

number of obs: 7185, groups: id, 160
AIC = 46691.5, DIC = 46660.7
deviance = 46666.1

```

Make our outcomes:

```

plt = data.frame( ses.mid = c( 0, 1, 0, 0, 1, 0 ),
                  ses.high = c( 0, 0, 1, 0, 0, 1 ),
                  catholic = c( 0, 0, 0, 1, 1, 1 ) )
cf = fixef( M1b )
cf

```

```

(Intercept)          ses.catmid
9.1915044           2.2808807
ses.cathigh         sectorcatholic
5.0721921           3.4398984
ses.catmid:sectorcatholic ses.cathigh:sectorcatholic
-0.9759927          -2.4707460

```

```

plt = mutate( plt,
              Y = cf[[1]] + cf[[2]]*ses.mid + cf[[3]]*ses.high +
                  cf[[4]]*catholic + cf[[5]]*ses.mid*catholic + cf[[6]]*ses.high*catholic )
plt

```

	ses.mid	ses.high	catholic	Y
1	0	0	0	9.191504
2	1	0	0	11.472385
3	0	1	0	14.263697
4	0	0	1	12.631403
5	1	0	1	13.936291
6	0	1	1	15.232849

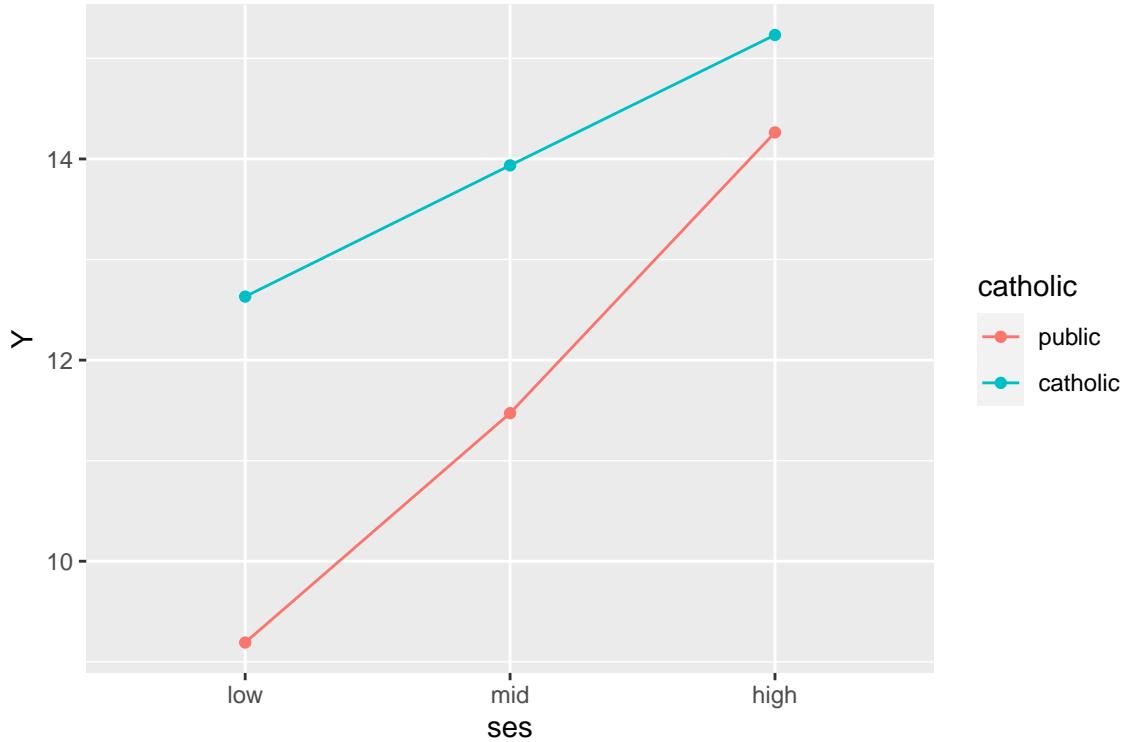
And plot

```

plt$catholic = factor( plt$catholic,
                      labels=c("public","catholic"),
                      levels=c(0,1) )

plt$ses = "low"
plt$ses[plt$ses.mid==1] = "mid"
plt$ses[plt$ses.high==1] = "high"
plt$ses = factor( plt$ses, levels=c("low","mid","high") )
ggplot( plt, aes( ses, Y, col=catholic, group=catholic ) ) +
  geom_line() + geom_point()

```



Note the *very important* `group=catholic` line that tells the plot to group everyone by catholic. If not, it will get confused and note that since ses is categorical, try to group on that. Then it cannot make a line since each group has only a single point.

13.2.3 Plotting individual school regression lines

We can plot the individual lines by hand-calculating the school level slopes and intercepts. This code shows how:

```
coefs = coef( M1 )$id
head( coefs )
```

	(Intercept)	ses	sector	catholic	ses:sector	catholic
1224	11.084408	2.863501		2.129531		-1.313363
1288	12.761032	3.099743		2.129531		-1.313363
1296	9.193415	2.597052		2.129531		-1.313363
1308	12.709882	3.092535		2.129531		-1.313363
1317	10.719013	2.812016		2.129531		-1.313363
1358	11.478455	2.919031		2.129531		-1.313363

```

coefs = rename( coefs,
                gamma.00 = `Intercept``,
                gamma.10 = `ses`,
                gamma.01 = `sectorcatholic`,
                gamma.11 = `ses:sectorcatholic` )
coefs$id = rownames( coefs )
coefs = merge( coefs, sdat, by="id" )
coefs = mutate( coefs,
                beta.0 = gamma.00 + gamma.01 * (sector=="catholic"),
                beta.1 = gamma.10 + gamma.11 * (sector=="catholic") )

```

Note how we have to add up our gammas to get our betas for each school. See our final betas, one set for each school:

```
head( dplyr::select( coefs, -gamma.00, -gamma.10, -gamma.01, -gamma.11 ) )
```

	id	size	sector	meanses	beta.0	beta.1
1	1224	842	public	-0.428	11.084408	2.863501
2	1288	1855	public	0.128	12.761032	3.099743
3	1296	1719	public	-0.420	9.193415	2.597052
4	1308	716	catholic	0.534	14.839413	1.779172
5	1317	455	catholic	0.351	12.848543	1.498653
6	1358	1430	public	-0.014	11.478455	2.919031

Now let's plot a subsample of 20 schools

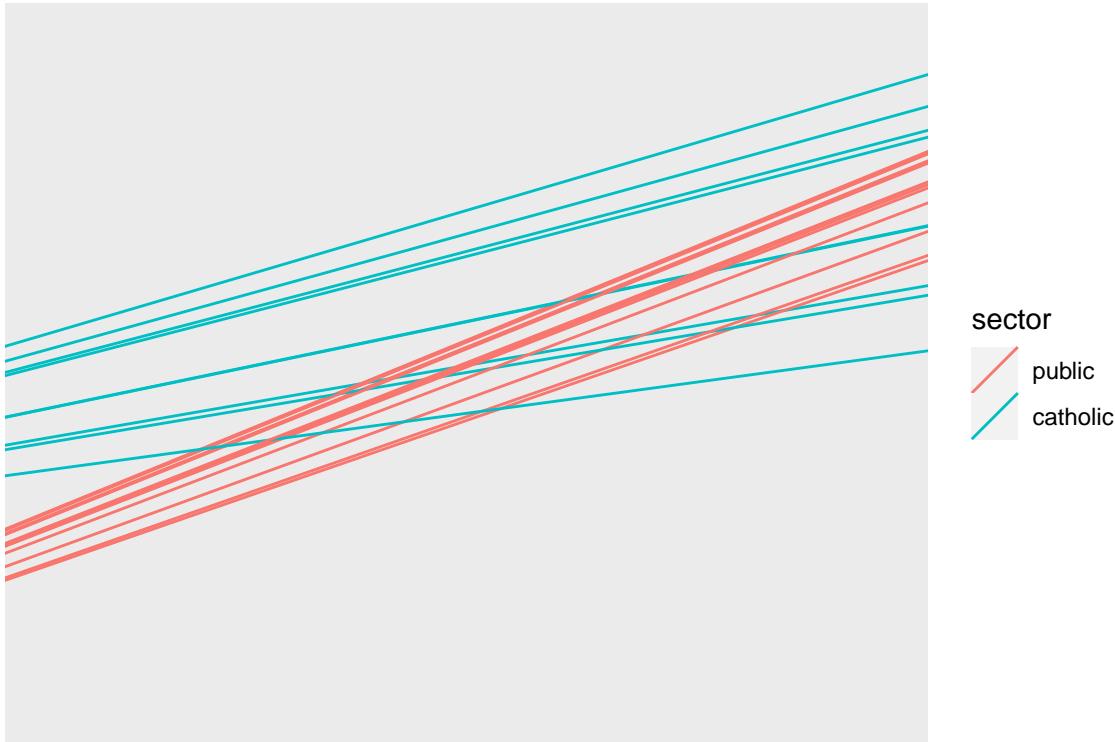
```

set.seed( 102030 )
sub20 = sample( unique( dat$id ), 20 )

coefs.20 = filter( coefs, id %in% sub20 )

ggplot( coefs.20, aes( group=id ) ) +
  geom_abline( aes( slope=beta.1, intercept=beta.0, col=sector) ) +
  coord_cartesian( xlim=c(-2.5,2), ylim=range(dat$mathach) )

```



Commentary: We need to specify the size of the plot since we have no data, just the intercepts and slopes. We are using the Empirical Bayes estimates of the random effects added to our school level fixed effects to get the $\hat{\beta}_{0j}, \hat{\beta}_{1j}$ which define the school-specific regression line for school j .

Our two types of school are clearly separated. Catholic schools have higher average performance, and less of a ses-achievement relationship. Since we have merged in our school level data, we can color the lines by catholic vs public, making our plot easier to read.

13.2.4 Plotting with predict()

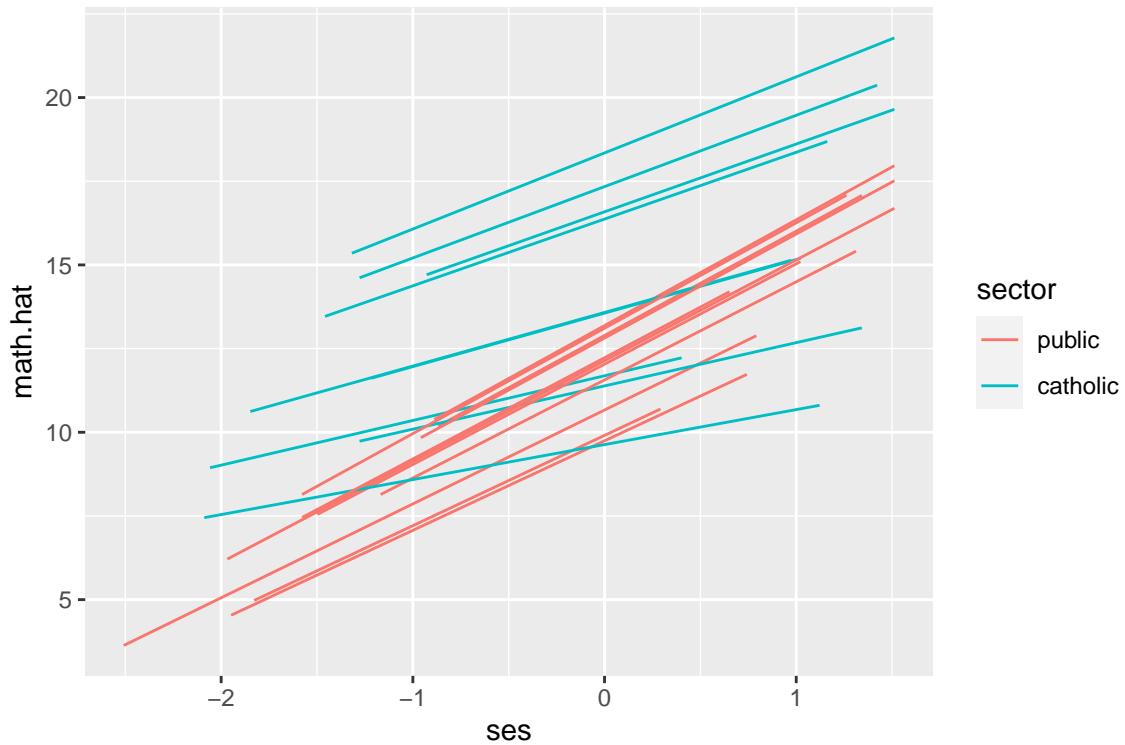
A more general plotting approach is to plot using `predict()`, where for each student we predict the outcome.

```
dat$math.hat = predict( M1 )
```

Now let's plot a subsample of 20 schools

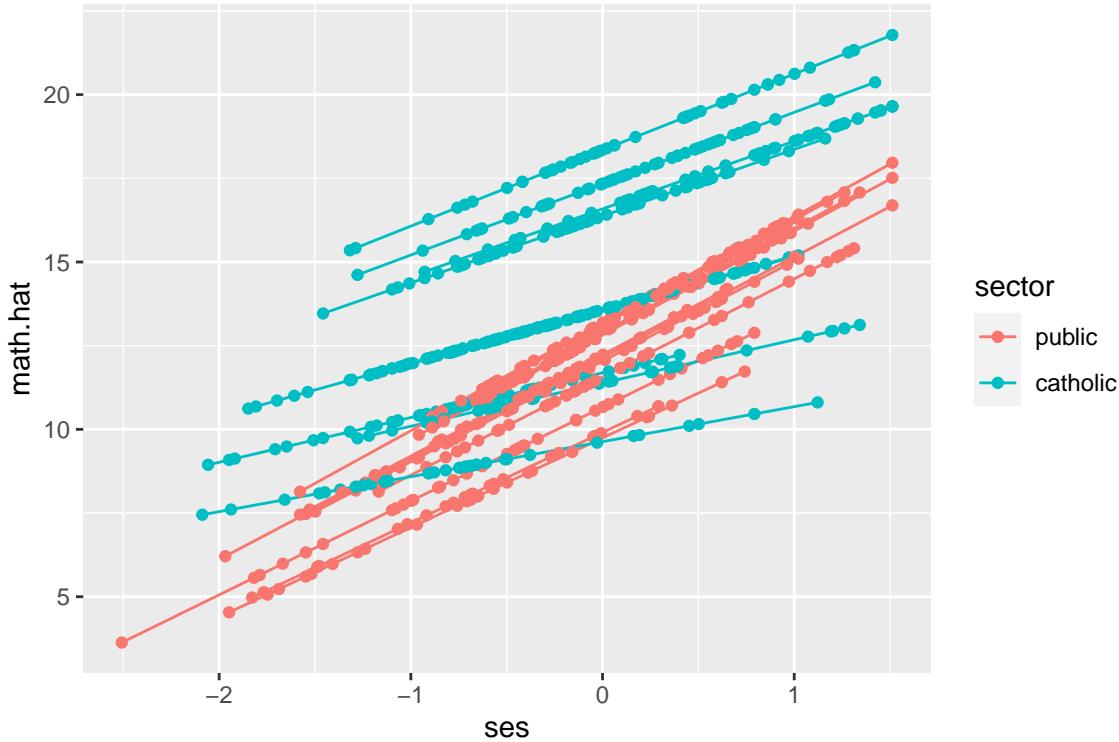
```
dat.20 = filter( dat, id %in% sub20 )

ggplot( dat.20, aes( ses, math.hat, group=id, col=sector ) ) +
  geom_line()
```



But look at how the lines don't go the full distance. What ggplot is doing is plotting the individual students, and connecting them with a line. We can see this by plotting the students as well, like this:

```
ggplot( dat.20, aes( ses, math.hat, group=id, col=sector ) ) +
  geom_line() +
  geom_point()
```



We have a predicted outcome for each student, which removes the student residual, giving just the school trends. If we don't have students for some range of ses for a school, we won't have points in our plot for that range for that school. The lines thus give the ranges (left to right) of the ses values in each school.

13.2.5 Making our lines go the same length with expand.grid()

The way we fix this is we, for each school, make a bunch of fake students with different SES and predict along all those fake students. This will give us equally spaced lines.

That being said: the shorter lines above are also informative, as they give you a sense of what the range of ses for each school actually is. Which approach is somewhat a matter of taste.

We can generate fake children of each group for each school using `expand.grid()`. This method will generate a dataframe with all combinations of the given variables supplied. Here we make all combinations of ses, for a set of ses values, and school id.

```
synth.dat = expand_grid( id = unique( dat$id ),
                         ses = seq( -2.5, 2, length.out=9 ) )
head( synth.dat )
```

```
# A tibble: 6 x 2
  id      ses
  <chr>  <dbl>
1 1224   -2.5
2 1224   -1.94
3 1224   -1.38
4 1224   -0.812
5 1224   -0.25
6 1224    0.312
```

The `seq()` command makes an evenly spaced *sequence* of numbers going from the first to the last, with 9 numbers. E.g.,

```
seq( 1, 10, length.out=4 )
```

```
[1] 1 4 7 10
```

We then merge our school info back in to get sector for each school id:

```
synth.dat = merge( synth.dat, sdat, by="id", all.x=TRUE )
```

We finally predict for each school, predicting outcome for our fake kids in each school.

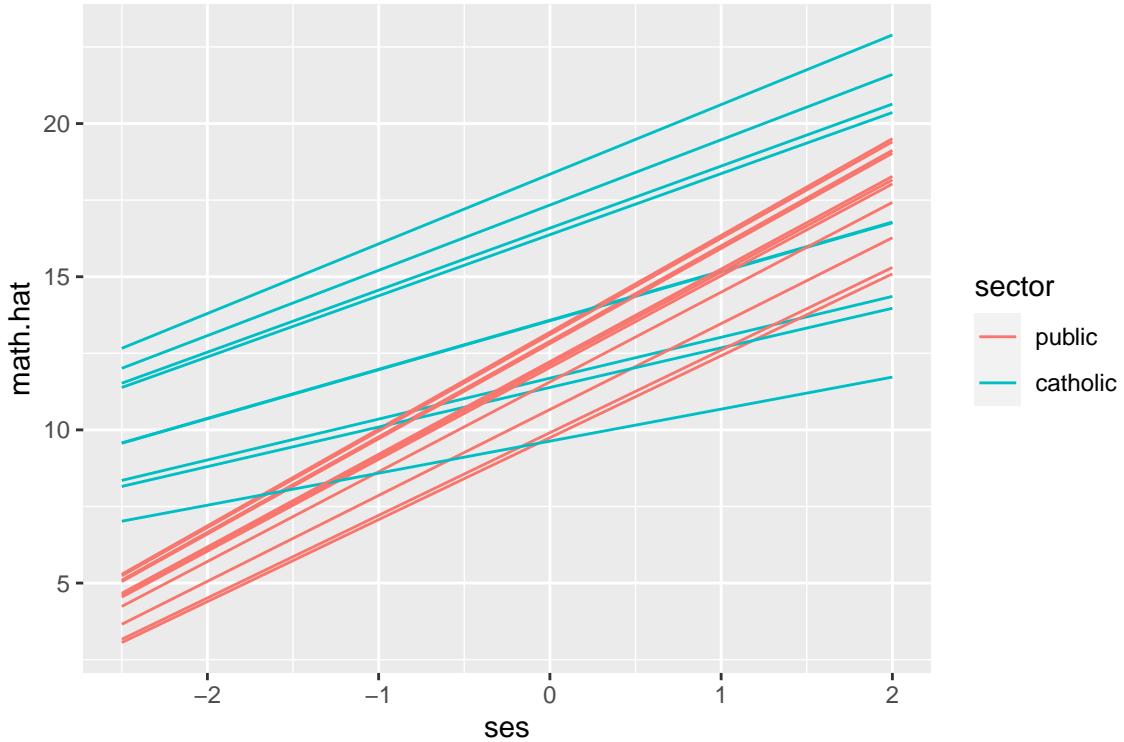
```
synth.dat$math.hat = predict( M1, newdata=synth.dat )
```

We have predictions just as above, just for students that we set for each school. The school random effects and everything remain because we are using the original school ids.

Using our new data, plot 20 random schools—this code is the same as in the prior subsection.

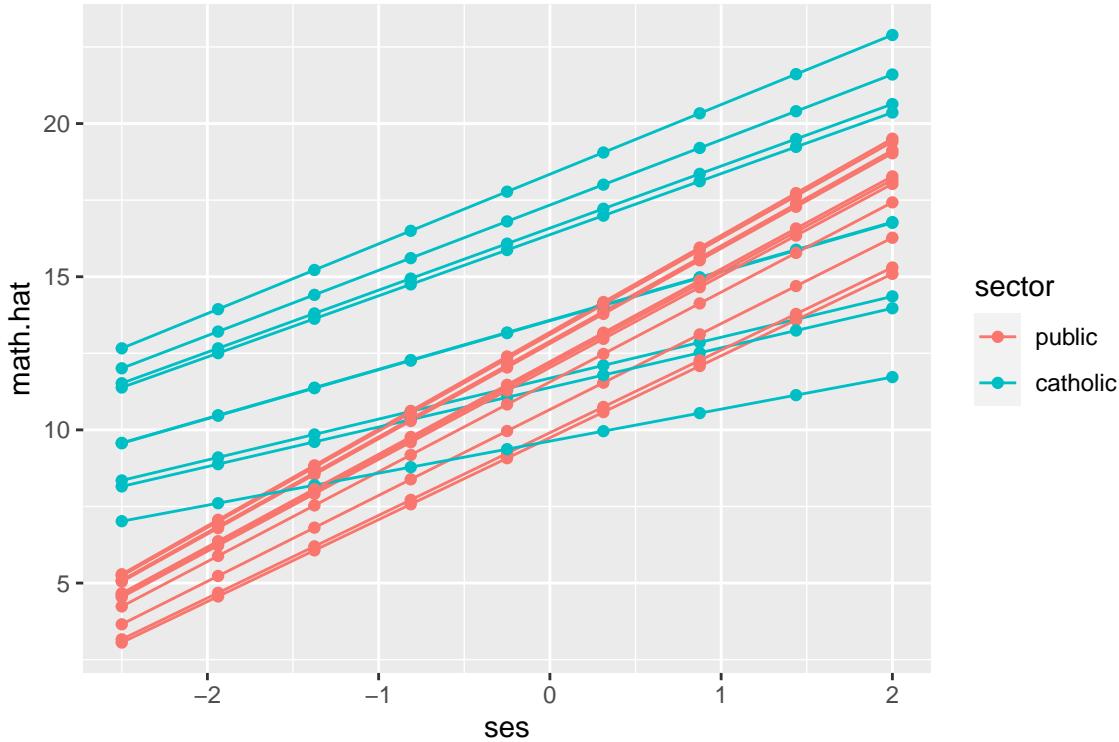
```
synth.dat.20 = filter( synth.dat, id %in% sub20 )

ggplot( synth.dat.20, aes( ses, math.hat, group=id, col=sector ) ) +
  geom_line()
```



But see our equally spaced students?

```
ggplot( synth.dat.20, aes( ses, math.hat, group=id, col=sector ) ) +  
  geom_line() +  
  geom_point()
```



Why do this? The `predict()` approach allows us to avoid working with the gammas and adding them up like we did above. This is a flexible and powerful approach that avoids a lot of work in many cases. In the next section we illustrate by fitting curves rather than lines. This would be very hard to do directly.

13.2.6 Superfancy extra bonus plotting of complex models!

We can use `predict` for weird nonlinear relationships also. This will be important for longitudinal data. To illustrate we fit a model that allows a quadradic relationship between ses and math achievement.

```
dat$ses2 = dat$ses^2
M2 = lmer( mathach ~ 1 + (ses + ses2)*sector + meanses + (1 + ses|id), data=dat )

display( M2 )

lmer(formula = mathach ~ 1 + (ses + ses2) * sector + meanses +
  (1 + ses | id), data = dat)
```

```

            coef.est  coef.se
(Intercept)    12.17     0.21
ses             2.79     0.15
ses2            0.04     0.13
sectorcatholic 1.23     0.33
meanses         3.14     0.38
ses:sectorcatholic -1.35    0.22
ses2:sectorcatholic  0.06    0.21

```

Error terms:

Groups	Name	Std.Dev.	Corr
id	(Intercept)	1.53	
	ses	0.23	0.49
	Residual	6.07	

number of obs: 7185, groups: id, 160

AIC = 46539.7, DIC = 46495.9

deviance = 46506.8

To fit a quadratic model we need our quadratic ses term, which we make by hand. We could also have used `I(ses^2)` in the `lmer()` command directly, but people don't tend to find that easy to read.

And here we predict and plot:

```

synth.dat = expand.grid( id = unique( dat$id ),
                        ses= seq( -2.5, 2, length.out=9 ) )
synth.dat$ses2 = synth.dat$ses^2
synth.dat = merge( synth.dat, sdat, by="id", all.x=TRUE )

```

Note how we make our `ses2` variable out of `ses` just like we did above.

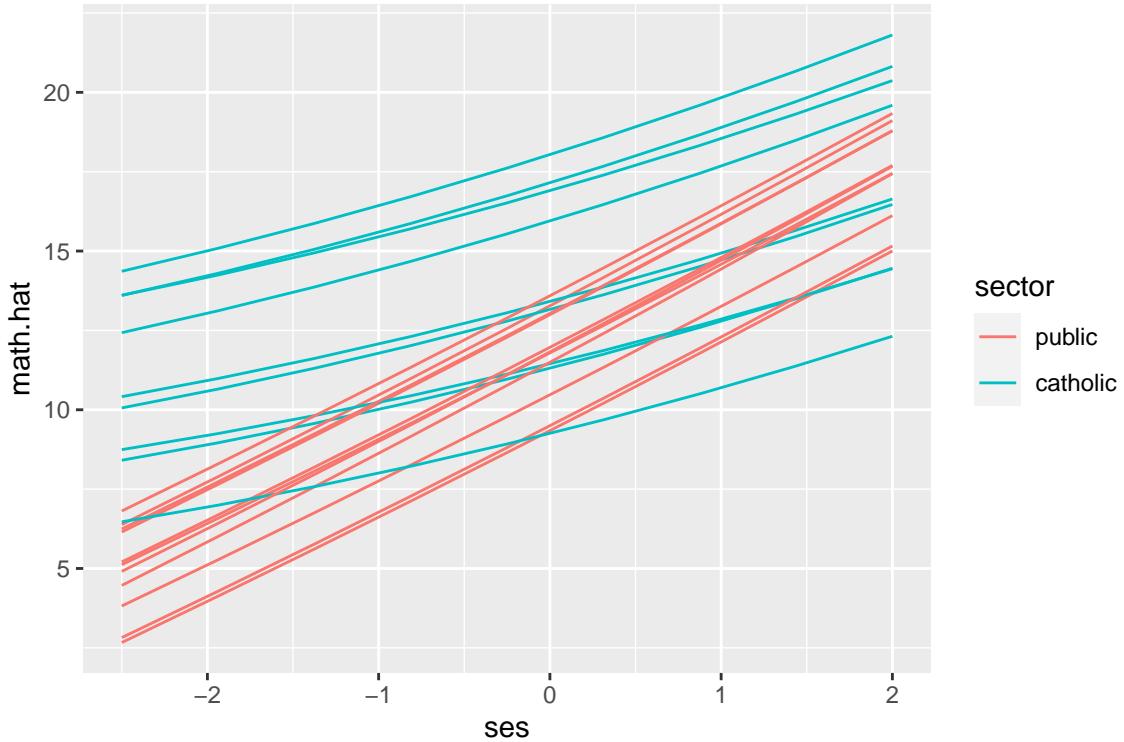
```

synth.dat$math.hat = predict( M2, newdata=synth.dat )

synth.dat.20 = filter( synth.dat, id %in% sub20 )

ggplot( synth.dat.20, aes( ses, math.hat, group=id, col=sector ) ) +
  geom_line()

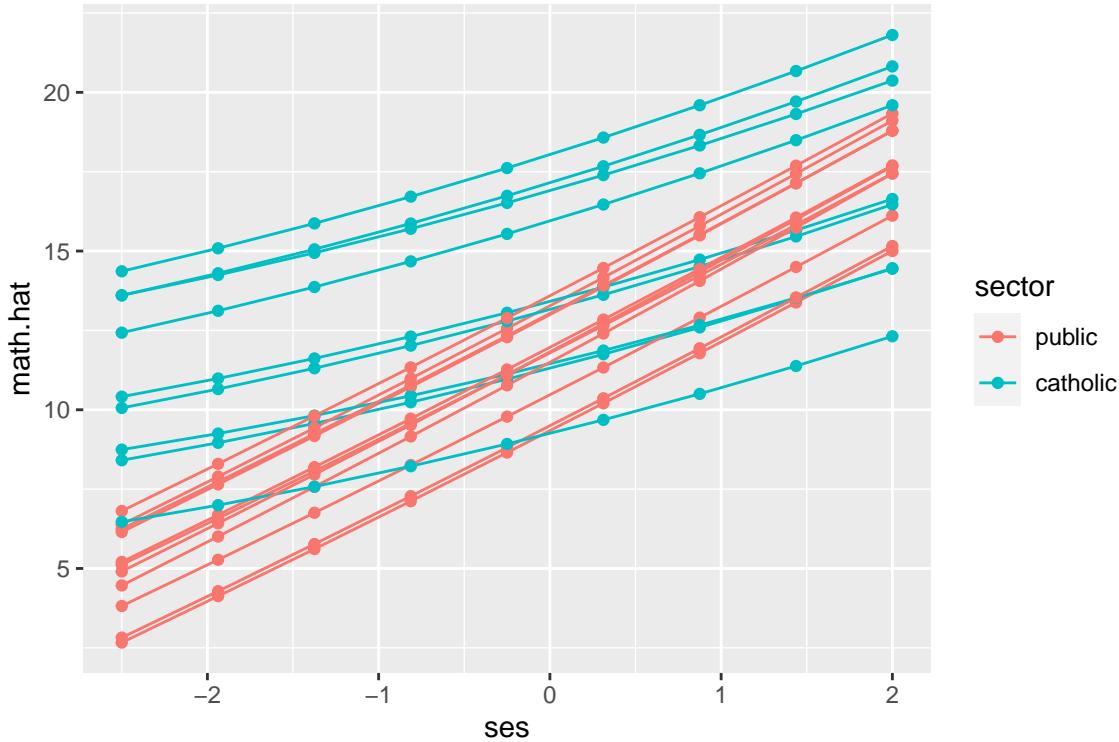
```



This code is the same as above. The prediction handles all our model complexity for us.

Again, we have our equally spaced students:

```
ggplot( synth.dat.20, aes( ses, math.hat, group=id, col=sector ) ) +
  geom_line() +
  geom_point()
```



13.3 Longitudinal Data

We next do the above, but for longitudinal data. The story is basically the same.

13.3.1 The data

We use the “US Sustaining Effects Study” taken from Raudenbush and Bryk (we have not seen these data in class). We have kids in grades nested in schools. So longitudinal data with a clustering on top of that.

```
head( dat )
```

	CHILDDID	SCHOOLID	YEAR	GRADE	MATH	FEMALE	SIZE	RACEETH
1	101480302	3440	-0.5	1	-1.694	1	588	black
2	101480302	3440	0.5	2	-0.211	1	588	black
3	101480302	3440	1.5	3	-0.403	1	588	black
4	101480302	3440	2.5	4	0.501	1	588	black
5	173559292	2820	-0.5	1	-0.194	0	678	white

```
6 173559292      2820      0.5      2  2.140      0   678     white
```

13.3.2 A model

We will be using the following 3-level quadratic growth model:

```
M4 = lmer( MATH ~ 1 + (YEAR + I(YEAR^2)) * (FEMALE * RACEETH) +
            (YEAR|CHIL DID:SCHOOLID) + (YEAR|SCHOOLID), data=dat )
display( M4 )
```

```
lmer(formula = MATH ~ 1 + (YEAR + I(YEAR^2)) * (FEMALE * RACEETH) +
      (YEAR | CHIL DID:SCHOOLID) + (YEAR | SCHOOLID), data = dat)
      coef.est  coef.se
(Intercept) -0.90    0.06
YEAR          0.76    0.02
I(YEAR^2)     -0.04    0.01
FEMALE        0.02    0.05
RACEETHhispanic 0.23    0.10
RACEETHwhite   0.79    0.10
FEMALE:RACEETHhispanic -0.01    0.12
FEMALE:RACEETHwhite   -0.34    0.12
YEAR:FEMALE      0.01    0.02
YEAR:RACEETHhispanic 0.10    0.03
YEAR:RACEETHwhite   0.07    0.03
I(YEAR^2):FEMALE  0.01    0.01
I(YEAR^2):RACEETHhispanic -0.01    0.01
I(YEAR^2):RACEETHwhite   -0.02    0.01
YEAR:FEMALE:RACEETHhispanic -0.01    0.04
YEAR:FEMALE:RACEETHwhite   -0.02    0.04
I(YEAR^2):FEMALE:RACEETHhispanic 0.00    0.02
I(YEAR^2):FEMALE:RACEETHwhite   0.02    0.02
```

Error terms:

Groups	Name	Std.Dev.	Corr
CHIL DID:SCHOOLID	(Intercept)	0.79	
	YEAR	0.11	0.55
SCHOOLID	(Intercept)	0.34	
	YEAR	0.10	0.31
Residual		0.54	

number of obs: 7230, groups: CHIL DID:SCHOOLID, 1721; SCHOOLID, 60

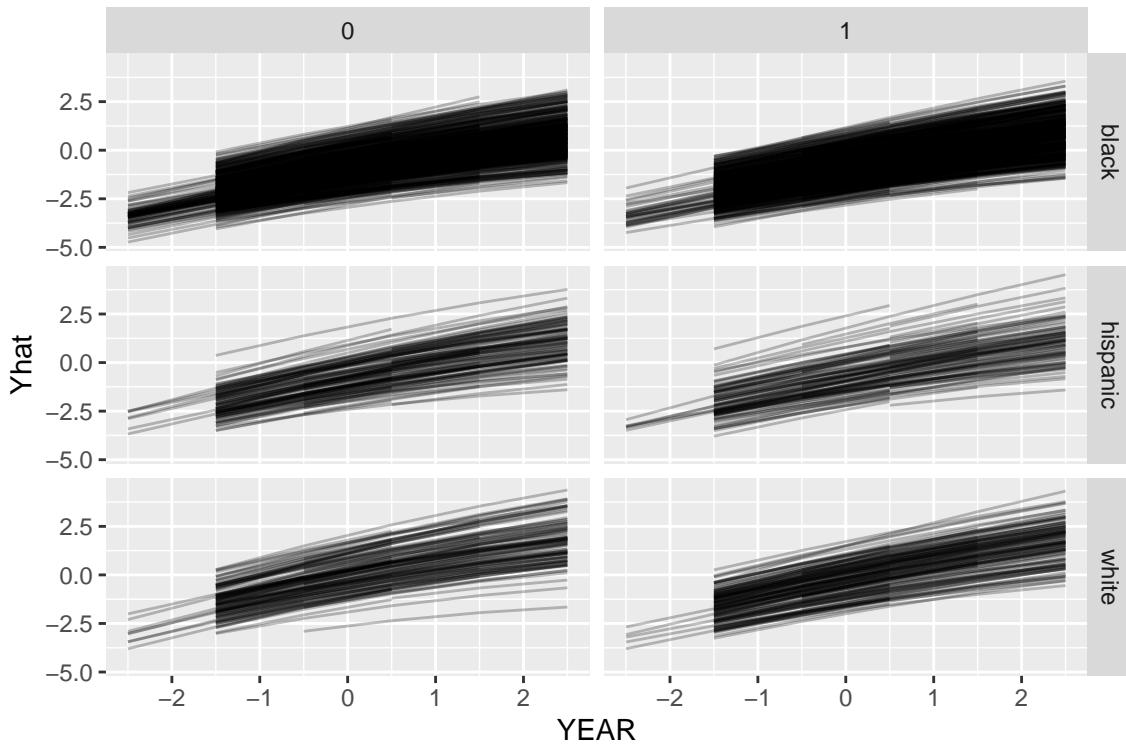
```
AIC = 16259.7, DIC = 16009.6
deviance = 16109.7
```

We are just taking the model as given; this document is about showing the fit of this model. In particular, if you haven't seen 3-level models before, just consider the above as some complex model; the nice thing about `predict()` is you don't even need to understand the model you are using! Note we do have a lot of fixed effect interaction terms, allowing for systematically different trajectories for groups of kids that are grouped on recorded race and gender.

13.3.3 The simple `predict()` approach

We can use our model to predict outcomes for each timepoint in the data. This will smooth out the time to time variation.

```
dat$Yhat = predict( M4 )
ggplot( dat, aes( YEAR, Yhat, group=CHILDDID ) ) +
  facet_grid( RACEETH ~ FEMALE ) +
  geom_line( alpha=0.25 )
```



Note how the growth lines don't go across all years for all kids. This is because we were missing data for those kids in the original dataset at those timepoints, so we didn't predict outcomes when we used the `predict()` function, above.

To fix this we will add in those missing timepoints so we get predictions for all kids for all timepoints.

13.3.4 The `expand.grid()` function

We now want different trajectories for the different groups. We can generate fake children of each group for each school using `expand.grid()`. This method will generate a dataframe with all combinations of the given variables supplied. Here we make all combinations of year, gender, and race/ethnic group for each school.

```
synth.dat = expand.grid( CHILDID = -1,
                         SCHOOLID = levels( dat$SCHOOLID ),
                         YEAR = unique( dat$YEAR ),
                         FEMALE = c( 0, 1 ),
                         RACEETH = levels( dat$RACEETH ) )
head( synth.dat )
```

	CHILDDID	SCHOOLID	YEAR	FEMALE	RACEETH
1	-1	2020	-0.5	0	black
2	-1	2040	-0.5	0	black
3	-1	2180	-0.5	0	black
4	-1	2330	-0.5	0	black
5	-1	2340	-0.5	0	black
6	-1	2380	-0.5	0	black

```
nrow( synth.dat )
```

```
[1] 2160
```

The `CHILDDID = -1` line means we are making up a new child (not using one of the real ones) so the child random effects will be set to 0 in the predictions.

Once we have our dataset, we use `predict` to calculate the predicted outcomes for each student type for each year timepoint for each school:

```

synth.dat = mutate( synth.dat, MATH = predict( M4,
                                              newdata=synth.dat,
                                              allow.new.levels = TRUE) )

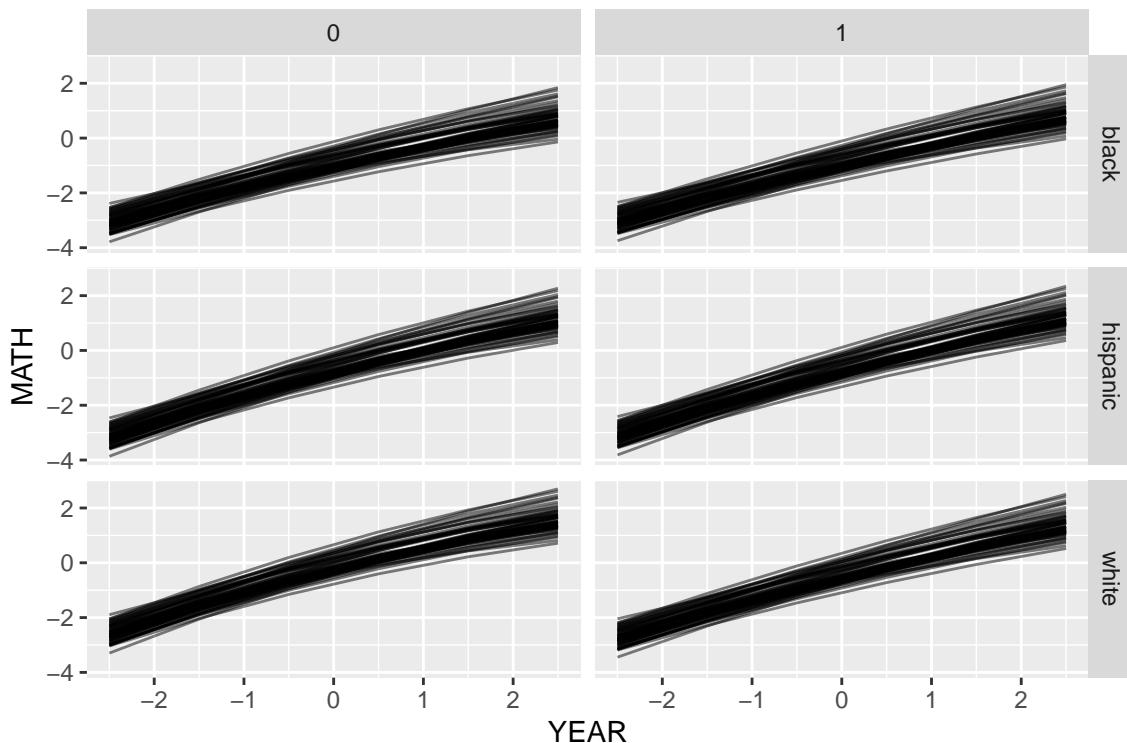
```

Now we can plot with our new predictions

```

ggplot( synth.dat, aes( YEAR, MATH, group=SCHOOLID ) ) +
  facet_grid( RACEETH ~ FEMALE ) +
  geom_line( alpha=0.5 )

```



Here we are seeing the different school trajectories for the six types of kid defined by our student-level demographics.

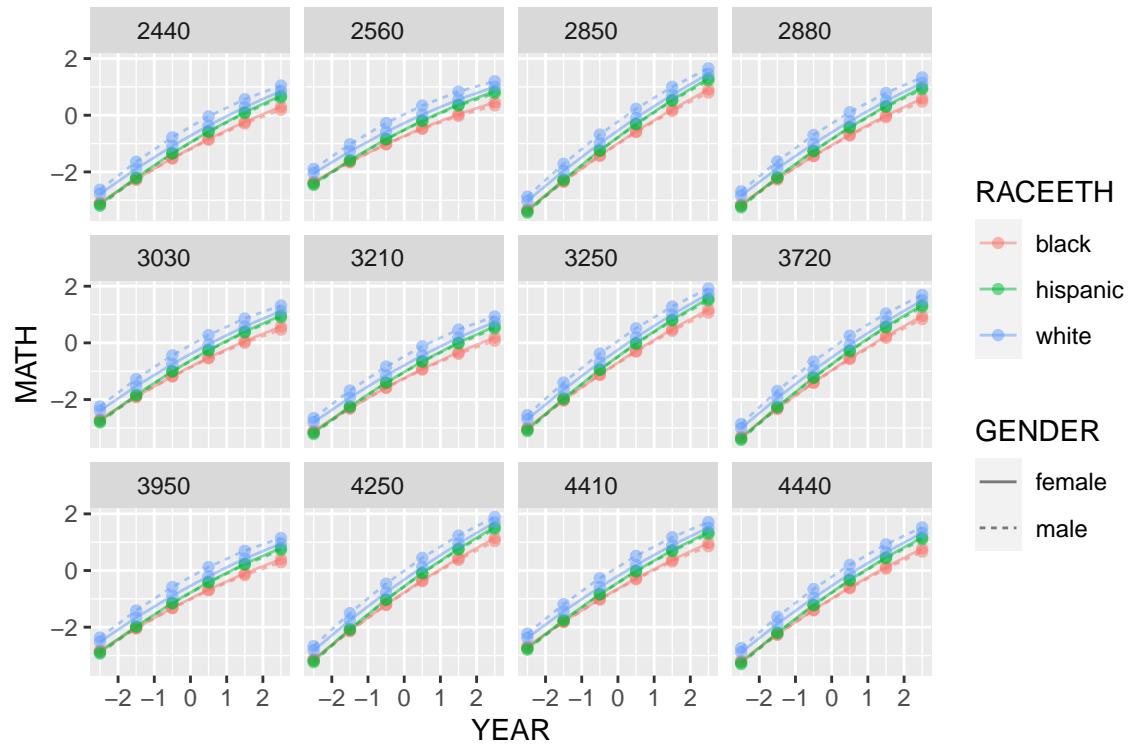
Or, for a subset of schools

```

synth.dat = mutate( synth.dat, GENDER = ifelse( FEMALE, "female", "male" ) )
keepers = sample( unique( synth.dat$SCHOOLID ), 12 )
s2 = filter( synth.dat, SCHOOLID %in% keepers )
ggplot( s2, aes( YEAR, MATH, col=RACEETH, lty=GENDER ) ) +

```

```
facet_wrap( ~ SCHOOLID ) +
geom_line( alpha=0.5) + geom_point( alpha=0.5 )
```



Here we see the six lines for the six groups within each school, plotted in little tiles, one for each school.

13.3.5 Population aggregation

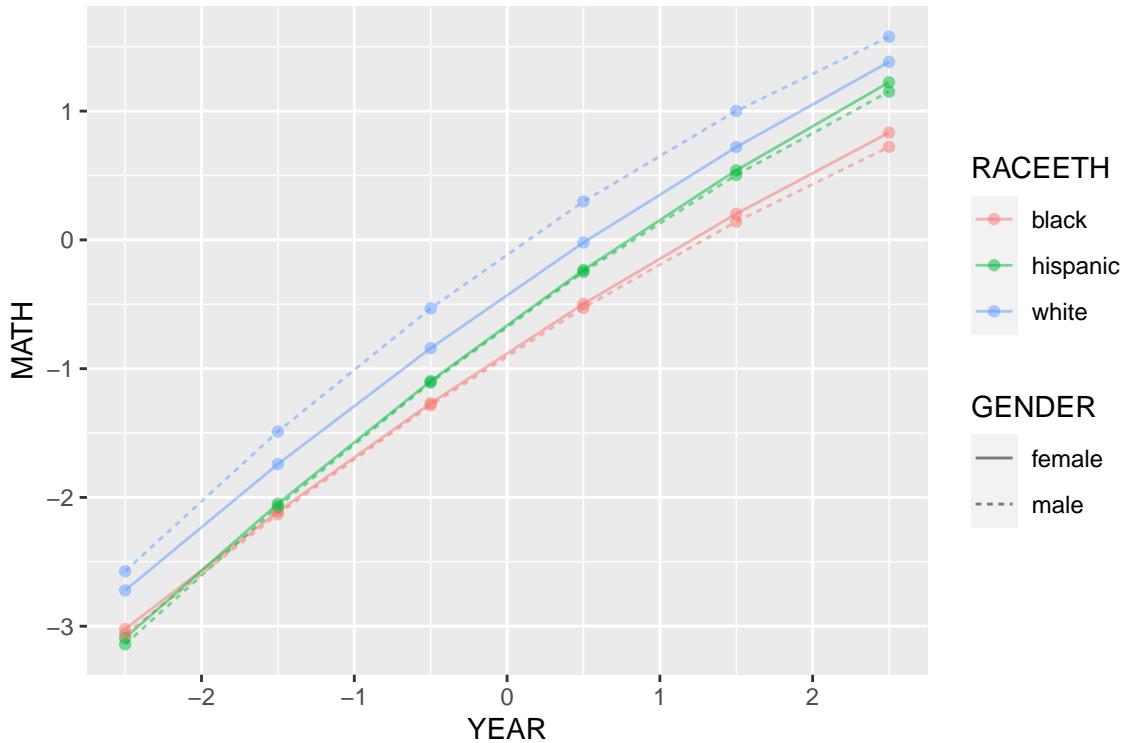
You can also aggregate these predictions. This is the easiest way to get what collection of schools, averaging over their random effects, looks like.

Aggregate with the `group_by()` and the `summarise()` methods:

```
agg.dat = synth.dat %>% group_by( GENDER, RACEETH, YEAR ) %>%
  dplyr::summarise( MATH = mean( MATH ) )
```

`summarise()` has grouped output by 'GENDER', 'RACEETH'. You can override using the `.groups` argument.

```
ggplot( agg.dat, aes( YEAR, MATH, col=RACEETH, lty=GENDER ) ) +
  geom_line( alpha=0.5) + geom_point( alpha=0.5 )
```



Or do this via predict directly, using the prior ideas

```
synth.dat.agg = expand.grid( CHIL DID = -1,
                             SCHOOLID = -1,
                             YEAR = unique( dat$YEAR ),
                             FEMALE = c( 0, 1 ),
                             RACEETH = levels( dat$RACEETH ) )
nrow( synth.dat.agg )
```

[1] 36

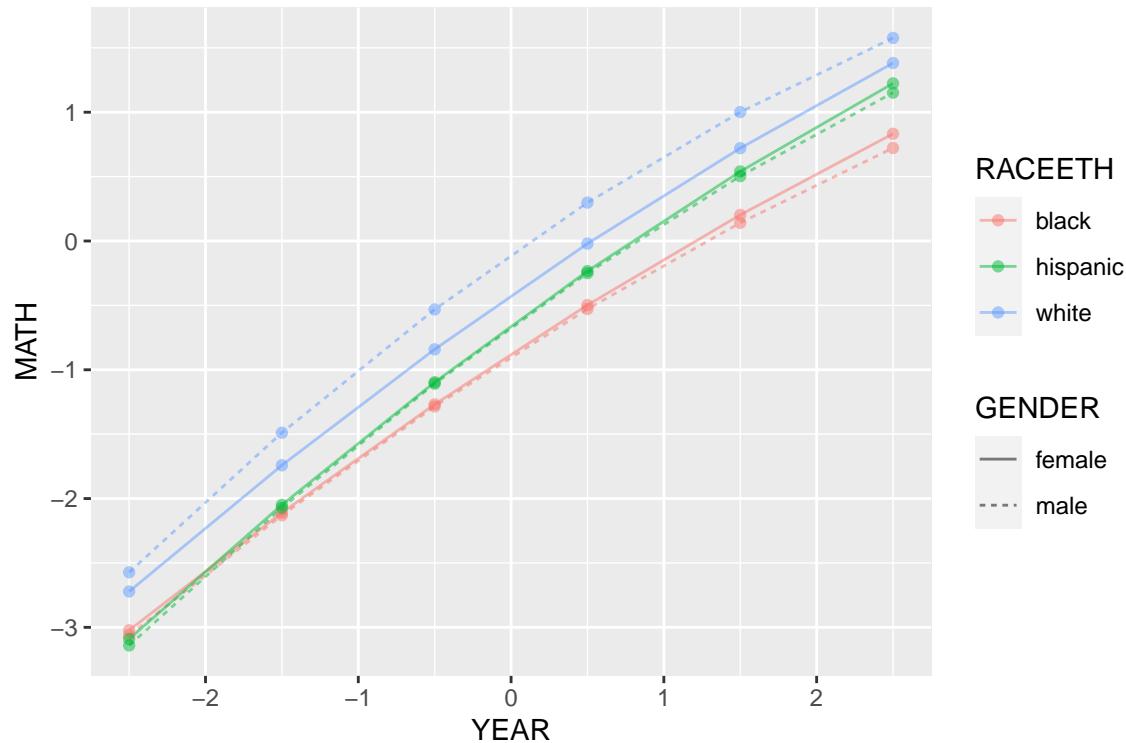
```
synth.dat.agg = mutate( synth.dat.agg,
                        MATH = predict( M4,
                                      newdata=synth.dat.agg,
                                      allow.new.levels = TRUE) )
```

```

synth.dat.agg = mutate( synth.dat.agg, GENDER = ifelse( FEMALE, "female", "male" ) )

ggplot( synth.dat.agg, aes( YEAR, MATH, col=RACEETH, lty=GENDER ) ) +
  geom_line( alpha=0.5) + geom_point( alpha=0.5 )

```



The above plot suggests that the gender gap only exists for the white children. It also shows that there are racial gaps, and that the Black children appear to be falling further behind as time passes.

This block of code is stand-alone, showing the making of fake data and plotting of predictions all in one go. Especially for glms, where there are nonlinearities due to the link function, this will give you the “typical” units, whereas the aggregation method will average over your individuals in the sample.

Finally, we can also make tables to calculate observed gaps (although in many cases you can just read this sort of thing off the regression table). First `spread` our data to get columns for each race

```
s3 = spread( synth.dat.agg, key="RACEETH", value="MATH" )
head( s3 )
```

	CHIL DID	SCHOOLID	YEAR	FEMALE	GENDER	black	hispanic	white
1	-1	-1	-2.5	0	male	-3.062600	-3.140762	-2.5729287
2	-1	-1	-2.5	1	female	-3.022569	-3.090733	-2.7217827
3	-1	-1	-1.5	0	male	-2.129830	-2.071722	-1.4888219
4	-1	-1	-1.5	1	female	-2.110196	-2.048893	-1.7416606
5	-1	-1	-0.5	0	male	-1.284949	-1.107972	-0.5317719
6	-1	-1	-0.5	1	female	-1.268485	-1.096512	-0.8412447

Then summarise:

```
tab = s3 %>% group_by( YEAR ) %>%
  summarise( gap.black.white = mean( white ) - mean( black ),
             gap.hispanic.white = mean( white ) - mean( hispanic ),
             gap.black.hispanic = mean( hispanic ) - mean( black ) )
knitr::kable( tab, digits=2 )
```

YEAR	gap.black.white	gap.hispanic.white	gap.black.hispanic
-2.5	0.40	0.47	-0.07
-1.5	0.50	0.45	0.06
-0.5	0.59	0.42	0.17
0.5	0.65	0.38	0.27
1.5	0.69	0.34	0.35
2.5	0.70	0.29	0.41

This again shows widening gap between Black and White students, and the closing gap of Hispanic and White students.

13.3.6 Plotting random effects by Level 2 variable

You can also look at estimated random effects as a function of level 2 variables. For example, we can see if there is a pattern of average math score for students by year.

```

ranef = ranef( M4 )$SCHOOLID
ranef$SCHOOLID = rownames( ranef )
schools = dat %>% group_by( SCHOOLID ) %>%
  summarise( n = n(),
             size = SIZE[[1]] )
schools = merge( schools, ranef, by="SCHOOLID" )
head( schools )

```

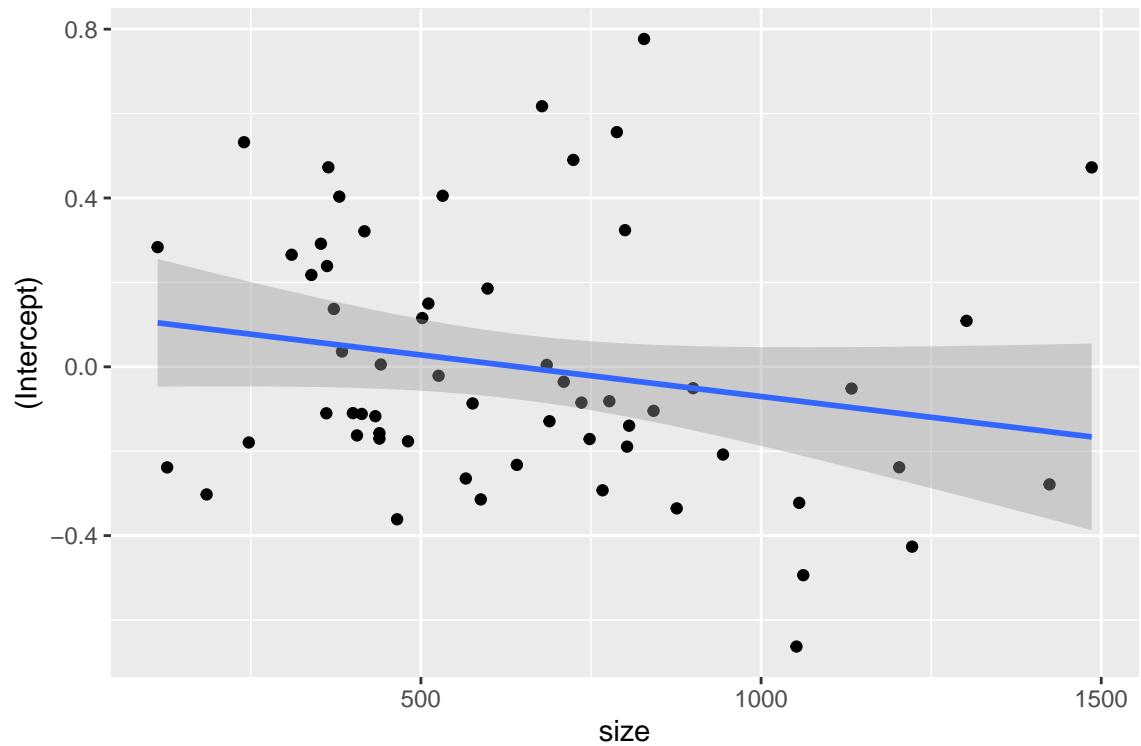
	SCHOOLID	n	size	(Intercept)	YEAR
1	2020	97	380	0.40324907	0.15258009
2	2040	89	502	0.11549704	0.07547476
3	2180	168	777	-0.08150587	-0.08227023
4	2330	150	800	0.32371449	-0.04389409
5	2340	220	1133	-0.05151814	-0.01128432
6	2380	87	439	-0.17017724	0.10803318

```

ggplot( schools, aes( size, `^`(Intercept)`^` ) ) +
  geom_point() +
  geom_smooth(method="lm")

```

```
`geom_smooth()` using formula = 'y ~ x'
```



We see a possible negative trend.

14 Easy Graphing with ggeffects

An awesome convenience function for graphing regression models is the `ggeffects` package. It's the best equivalent I've found in R to Stata's `margins`. Let's demonstrate with the HSB data.

```
# load libraries
library(tidyverse)
library(lme4)
library(ggeffects)
library(sjPlot)
library(haven)

# clear memory
rm(list = ls())

# load HSB data
hsb <- read_dta("data/hsb.dta")
```

14.1 Fit a Series of Models

We can fit 2- and 3-way interactions, but they can be hard to interpret from the coefficients alone (unless you have a lot of practice).

```
m1 <- lmer(mathach ~ ses + (1|schoolid), hsb)
m2 <- lmer(mathach ~ ses + sector + (1|schoolid), hsb)
m3 <- lmer(mathach ~ ses*sector + (1|schoolid), hsb)
m4 <- lmer(mathach ~ ses*sector*female + (1|schoolid), hsb)

# tabulate results with tab_model
tab_model(m1, m2, m3, m4,
          p.style = "stars",
          show.ci = FALSE,
          show.se = TRUE)
```

mathach

mathach

mathach

mathach

Predictors

Estimates

std. Error

Estimates

std. Error

Estimates

std. Error

Estimates

std. Error

(Intercept)

12.66 ***

0.19

11.72 ***

0.23

11.80 ***

0.23

12.41 ***

0.25

ses

2.39 ***

0.11

2.37 ***

0.11

2.95 ***

0.14
 2.73 ***
 0.19
 sector
 2.10 ***
 0.34
 2.14 ***
 0.34
 2.16 ***
 0.38
 ses × sector
 -1.31 ***
 0.21
 -1.26 ***
 0.30
 female
 -1.16 ***
 0.21
 ses × female
 0.34
 0.26
 sector × female
 -0.05
 0.35
 (ses × sector) × female
 -0.05
 0.40
 Random Effects

37.03
37.04
36.84
36.63

00
4.77 schoolid
3.69 schoolid
3.69 schoolid
3.40 schoolid
ICC
0.11
0.09
0.09
0.09
N
160 schoolid
160 schoolid
160 schoolid
160 schoolid
Observations
7185
7185
7185
7185
Marginal R² / Conditional R²
0.077 / 0.182
0.114 / 0.195
0.118 / 0.199
0.127 / 0.202

* p<0.05 ** p<0.01 *** p<0.001

14.2 Graph the Results with ggeffects

If we just call `ggeffect` on the model object, we get a bunch of predicted values:

```
ggeffect(m1)

$ses
# Predicted values of mathach

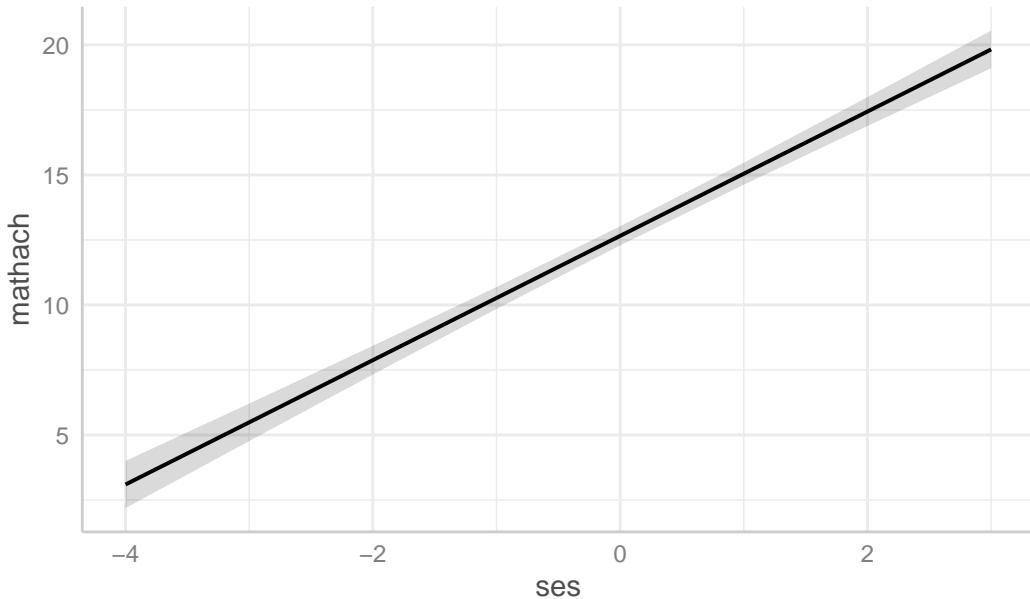
ses | Predicted |      95% CI
-----
-4 |     3.10 | [ 2.19,  4.00]
-3 |     5.49 | [ 4.77,  6.21]
-2 |     7.88 | [ 7.32,  8.43]
-1 |    10.27 | [ 9.85, 10.69]
  0 |    12.66 | [12.29, 13.03]
  1 |    15.05 | [14.62, 15.47]
  2 |    17.44 | [16.88, 17.99]
  3 |    19.83 | [19.10, 20.55]

attr(,"class")
[1] "ggalleffects" "list"
attr(,"model.name")
[1] "m1"
```

We can pipe that into `plot` to get a nice plot:

```
ggeffect(m1) |>
  plot()
```

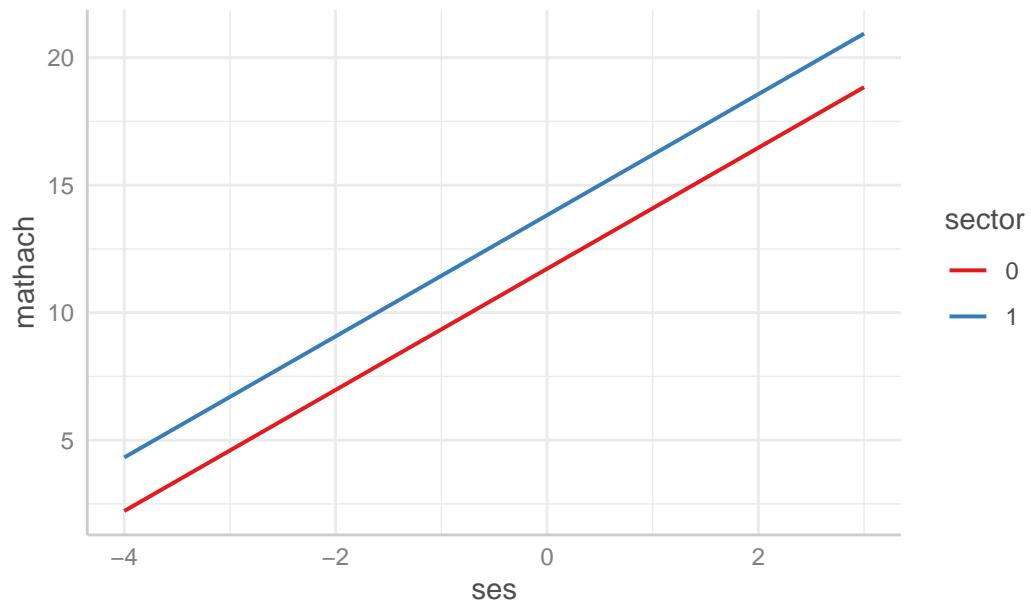
Predicted values of mathach



With multiple covariates, we can call the `terms` argument. The first input is on X, the second is mapped to color, the third to facet. This makes visualizing the interactions super easy! Any covariates included in the model but not included in `terms` are held constant at their means.

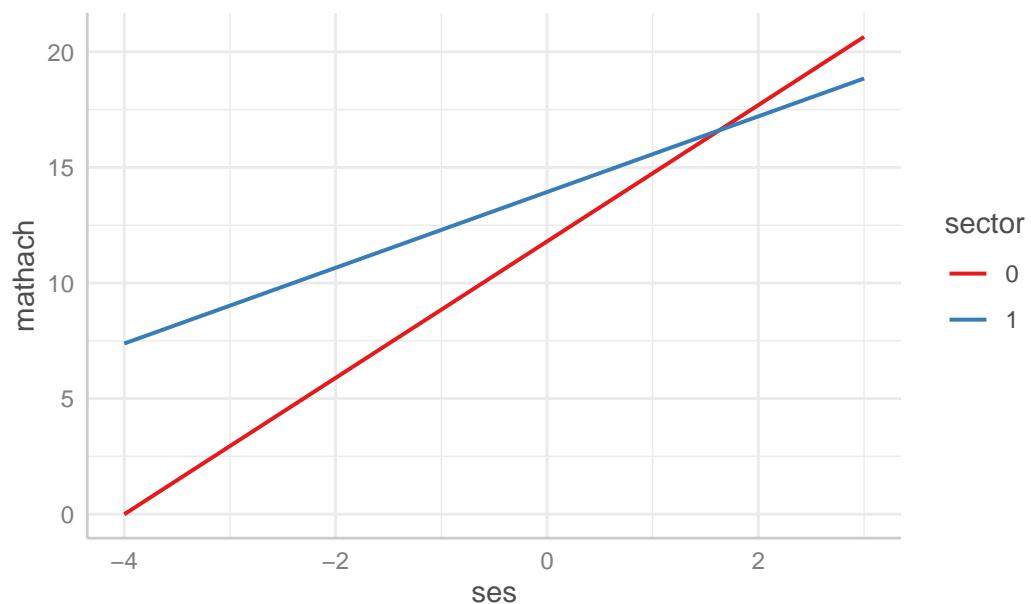
```
ggeffect(m2, terms = c("ses", "sector")) |>  
  plot(ci = FALSE, add.data = TRUE)
```

Predicted values of mathach

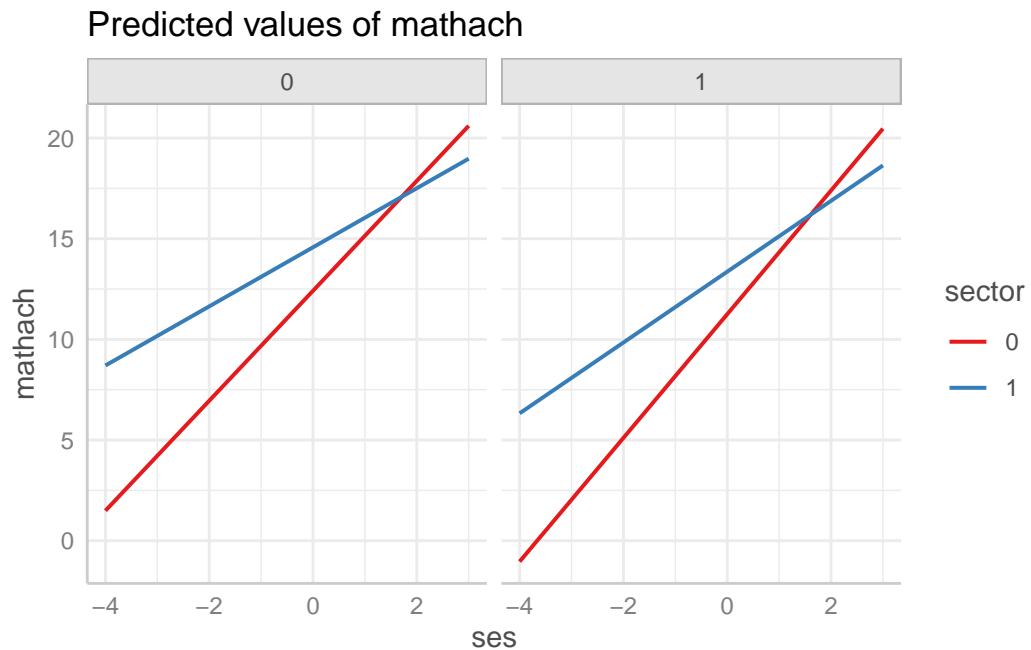


```
ggeffect(m3, terms = c("ses", "sector")) |>  
  plot(ci = FALSE)
```

Predicted values of mathach



```
ggeffect(m4, terms = c("ses", "sector", "female")) |>  
  plot(ci = FALSE)
```



15 Plotting Two Datasets at Once

It's easy (though not always advisable) to plot two data sets at once with `ggplot`. First, we load tidyverse and our HSB data. We then create a school-level aggregate data set of just the mean SES values.

```
library(tidyverse)
library(haven)

# clear memory
rm(list = ls())

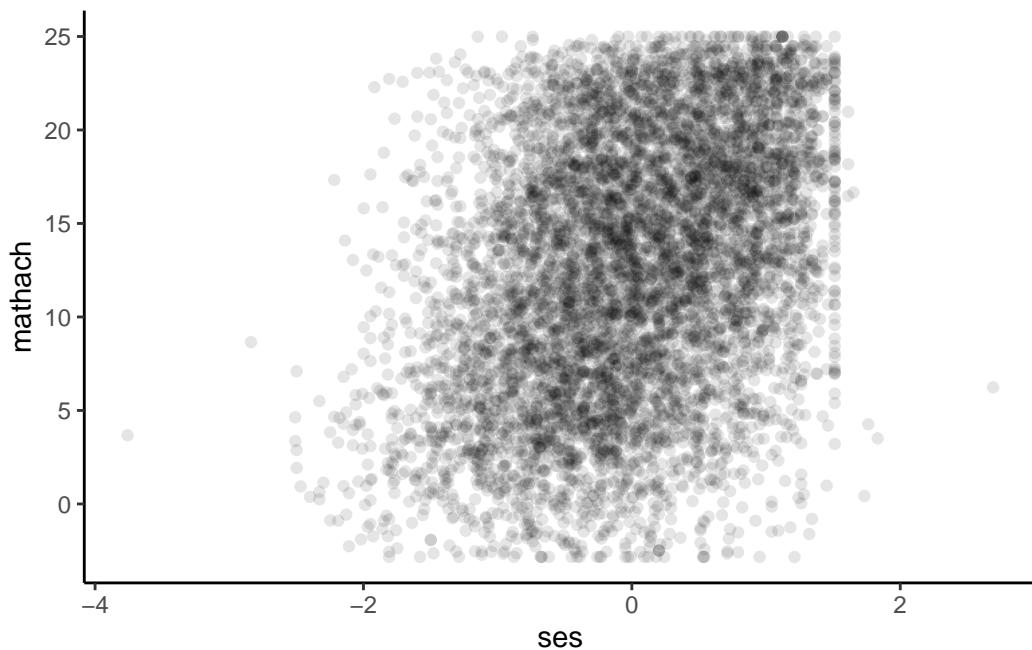
theme_set(theme_classic())

# load HSB data
hsb <- read_dta("data/hsb.dta") |>
  select(mathach, ses, schoolid)

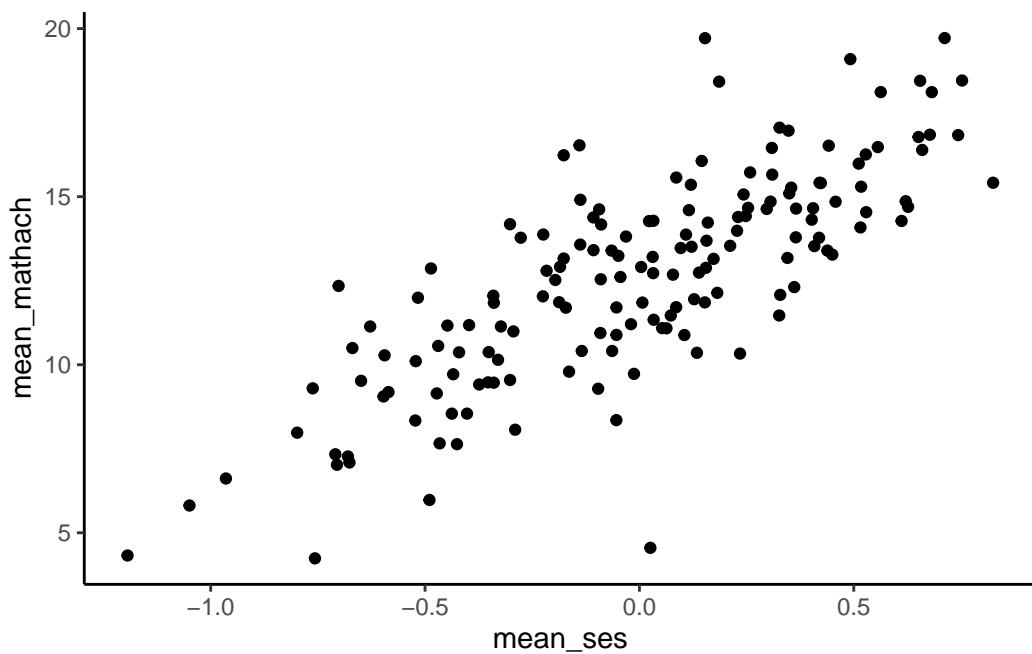
sch <- hsb |>
  group_by(schoolid) |>
  summarise(mean_ses = mean(ses),
            mean_mathach = mean(mathach))
```

Let's say we wanted to plot *both* the individual students *and* the school means. This is easy enough to do separately:

```
ggplot(hsb, aes(x = ses, y = mathach)) +
  geom_point(alpha = 0.1)
```

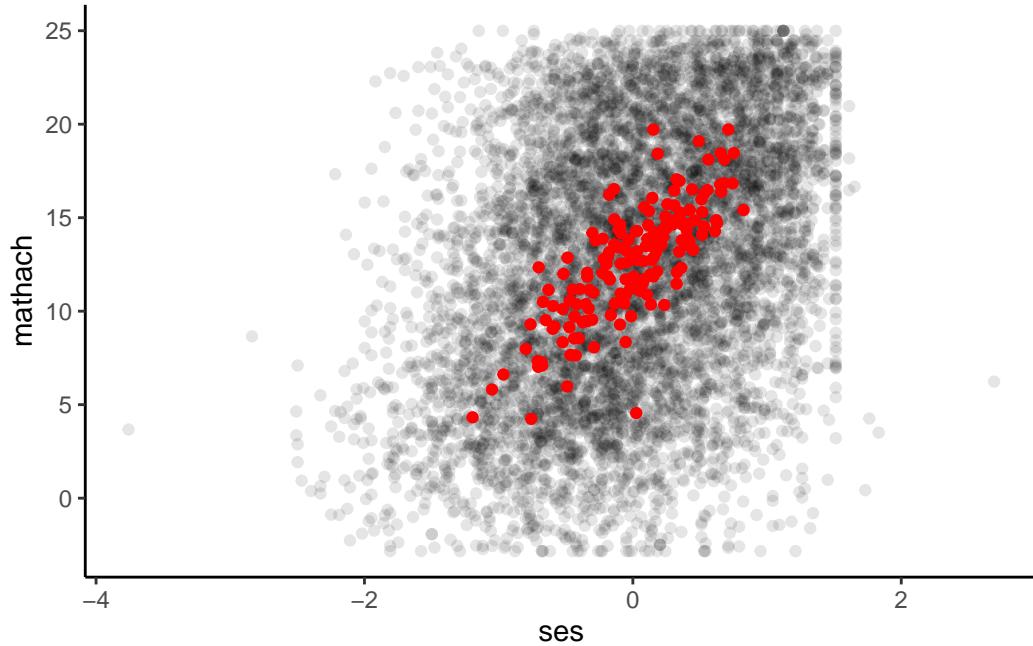


```
ggplot(sch, aes(x = mean_ses, y = mean_mathach)) +  
  geom_point()
```



We can superimpose both plots as follows. Essentially, the first argument in `ggplot` provides the data, and by default, this is passed to all subsequent layers of the plot. We can override this behavior by specifying a different data set (and aesthetic mappings, if desired) *within an individual layer* of `ggplot`, such as `geom_point`.

```
ggplot(hsb, aes(x = ses, y = mathach)) +  
  geom_point(alpha = 0.1) +  
  geom_point(data = sch, aes(x = mean_ses, y = mean_mathach), color = "red")
```



Part III

MODEL FITTING & INTERPRETATION

16 Extracting information from fitted `lmer` models with `broom`

There are three general ways to get information out of a fit model: (1) print it to the screen and read it, (2) use a variety of base R methods to pull information out of the model, and (2) use the `broom` package to pull information out of the model into different kinds of data frames (which is in line with *tidy programming*, and the tidyverse).

This chapter looks at the third way. The following chapter looks at the “base R” way. Which to use is a matter of preference.

16.1 Simple Demonstration

One of my favorite R packages is `broom`, which has many awesome convenience functions for regression models, including MLMs. `broom.mixed` is the extension that specifically works with `lmer` models. It does this via a few core methods that give you the model parameters and information as a nice data frame that you can then use more easily than the original result from your `lmer()` call. Let’s see how it works.

We first load it (and a few other things, and some data):

```
# load libraries
library(tidyverse)
library(broom.mixed)
library(haven)
library(knitr)
library(lme4)

# clear memory
rm(list = ls())

# load HSB data
hsb <- read_dta("data/hsb.dta")
```

16.1.1 tidy

The `tidy()` method takes a model object and returns the output as a tidy tibble (i.e., a data frame), which makes it very easy to work with. Compare the results below:

```
ols <- lm(mathach ~ ses, hsb)

# ugly!
summary(ols)
```

```
Call:
lm(formula = mathach ~ ses, data = hsb)

Residuals:
    Min      1Q  Median      3Q     Max 
-19.4382 -4.7580  0.2334  5.0649 15.9007 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 12.74740   0.07569 168.42   <2e-16 ***
ses          3.18387   0.09712  32.78   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.416 on 7183 degrees of freedom
Multiple R-squared:  0.1301,    Adjusted R-squared:  0.13 
F-statistic: 1075 on 1 and 7183 DF,  p-value: < 2.2e-16
```

```
# beautiful!
tidy(ols)
```

```
# A tibble: 2 x 5
  term       estimate std.error statistic  p.value
  <chr>        <dbl>     <dbl>     <dbl>      <dbl>
1 (Intercept)  12.7     0.0757    168.     0
2 ses          3.18     0.0971    32.8  8.71e-220
```

```
# even better
ols |> tidy() |> kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	12.75	0.08	168.42	0
ses	3.18	0.10	32.78	0

```
# Also works great for MLMs
mlm <- lmer(mathach ~ ses + mnses + (ses|schoolid), hsb)

tidy(mlm)
```

```
# A tibble: 7 x 6
  effect group term          estimate std.error statistic
  <chr>   <chr> <chr>        <dbl>    <dbl>    <dbl>
1 fixed    <NA>   (Intercept)     12.7      0.151    84.2 
2 fixed    <NA>   ses            2.19      0.122    18.0  
3 fixed    <NA>   mnses          3.78      0.383    9.88  
4 ran_pars schoolid sd__(Intercept) 1.64      NA       NA    
5 ran_pars schoolid cor__(Intercept).ses -0.212     NA       NA    
6 ran_pars schoolid sd__ses          0.673     NA       NA    
7 ran_pars Residual sd__Observation 6.07      NA       NA
```

16.1.2 glance

What about model fit stats? That's where `glance` comes in:

```
glance(ols)
```

```
# A tibble: 1 x 12
  r.squared adj.r.squared sigma statistic  p.value      df logLik     AIC     BIC
  <dbl>        <dbl> <dbl>    <dbl>    <dbl>    <dbl> <dbl>    <dbl>    <dbl>
1     0.130        0.130  6.42    1075. 8.71e-220      1 -23549. 47104. 47125.
# i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
glance(mlm) |>
  kable(digits = 2)
```

	nobs	sigma	logLik	AIC	BIC	REMLcrit	df.residual
7185	6.07	-23280.71	46575.42	46623.58	46561.42		7178

16.1.3 augment

What about your estimated random effects? `augment` to the rescue, giving estimates for each random effect:

```
mlm |>
  ranef() |>
  augment() |>
  head() |>
  kable(digits = 2)
```

grp	variable	level	estimate	qq	std.error	lb	ub
schoolid	(Intercept)	8367	-4.14	-0.18	0.78	-5.43	-2.85
schoolid	(Intercept)	4523	-3.09	0.02	0.98	-4.70	-1.47
schoolid	(Intercept)	6990	-2.98	-1.46	0.78	-4.26	-1.70
schoolid	(Intercept)	3705	-2.81	0.28	1.09	-4.61	-1.02
schoolid	(Intercept)	8854	-2.57	-0.85	0.80	-3.89	-1.25
schoolid	(Intercept)	9397	-2.43	-0.65	0.92	-3.94	-0.92

The `level` column are your school IDs, here. If you have multiple sets of random effects, they will all be stacked, and indexed via `grp`.

16.2 Extracting lmer model info

16.2.1 Obtaining Fixed Effects

`lmer` models are in reduced form, so fixed effects include both L1 and L2 predictors. `tidy` denotes the type of effect in a column called `effect`, where `fixed` means fixed, and `ran_pars` means random (standing for “random parameters”)

```
mlm |>
  tidy() |>
  filter(effect == "fixed")
```



```
# A tibble: 3 x 6
  effect group term      estimate std.error statistic
  <chr>  <chr> <chr>      <dbl>     <dbl>      <dbl>
1 fixed   <NA>   (Intercept)  12.7      0.151     84.2
2 fixed   <NA>    ses        2.19      0.122     18.0
3 fixed   <NA>   mnses      3.78      0.383     9.88
```

We can use the `[[[]]]` notation or a pipeline to extract elements from the data frame:

```
# within effect of SES  
tidy(mlm)[[2,4]]
```

```
[1] 2.190349
```

```
# contextual effect of SES  
tidy(mlm)[[3,4]]
```

```
[1] 3.781243
```

```
# using the variable names in a pipeline  
mlm |>  
  tidy() |>  
  filter(term == "ses") |>  
  pull(estimate)
```

```
[1] 2.190349
```

16.2.2 Obtaining Random Effects

`tidy` includes the random effects (SDs and correlations) right there in the output. For example, `sd__ses` is the SD of the SES slope.

```
# display all random effects  
mlm |>  
  tidy() |>  
  filter(effect == "ran_pars")
```

```
# A tibble: 4 x 6  
  effect   group    term          estimate std.error statistic  
  <chr>    <chr>    <chr>        <dbl>     <dbl>      <dbl>  
1 ran_pars schoolid sd__(Intercept)     1.64      NA       NA  
2 ran_pars schoolid cor__(Intercept).ses -0.212     NA       NA  
3 ran_pars schoolid sd__ses            0.673     NA       NA  
4 ran_pars Residual  sd__Observation    6.07      NA       NA
```

```
# pull single number
mlm |>
  tidy() |>
  filter(term == "sd_ses") |>
  pull(estimate)
```

[1] 0.6730818

16.2.3 Obtaining Empirical Bayes Estimates of the Random Effects

This is best done in a pipeline. We first apply `ranef`, then `augment` and get the EB estimates in the `estimate` column, along with the `std.error`, confidence bounds, and `qq` statistics.

```
mlm |>
  ranef() |>
  augment() |>
  head()
```

	grp	variable	level	estimate	qq	std.error	lb	ub
1	schoolid	(Intercept)	8367	-4.137656	-0.1811498	0.7845770	-5.428170	-2.8471413
2	schoolid	(Intercept)	4523	-3.089835	0.0235018	0.9819306	-4.704967	-1.4747032
3	schoolid	(Intercept)	6990	-2.981315	-1.4619679	0.7779876	-4.260991	-1.7016394
4	schoolid	(Intercept)	3705	-2.811935	0.2776904	1.0911916	-4.606785	-1.0170840
5	schoolid	(Intercept)	8854	-2.569302	-0.8528365	0.8045804	-3.892719	-1.2458846
6	schoolid	(Intercept)	9397	-2.431031	-0.6452734	0.9163587	-3.938307	-0.9237553

16.2.4 Intercept-Slope Correlation

The BLUPs are in long form. We can reshape to wide if we want to, for example, visualize the correlation between the random intercepts and slopes.

```

blups <- mlm |>
  ranef() |>
  augment() |>
  dplyr::select(variable, level, estimate) |>
  pivot_wider(names_from = variable, values_from = estimate,
              id_cols = level) |>
  dplyr::rename(schoolid = 1, random_intercept = 2, random_slope = 3)

head(blups)

```

```

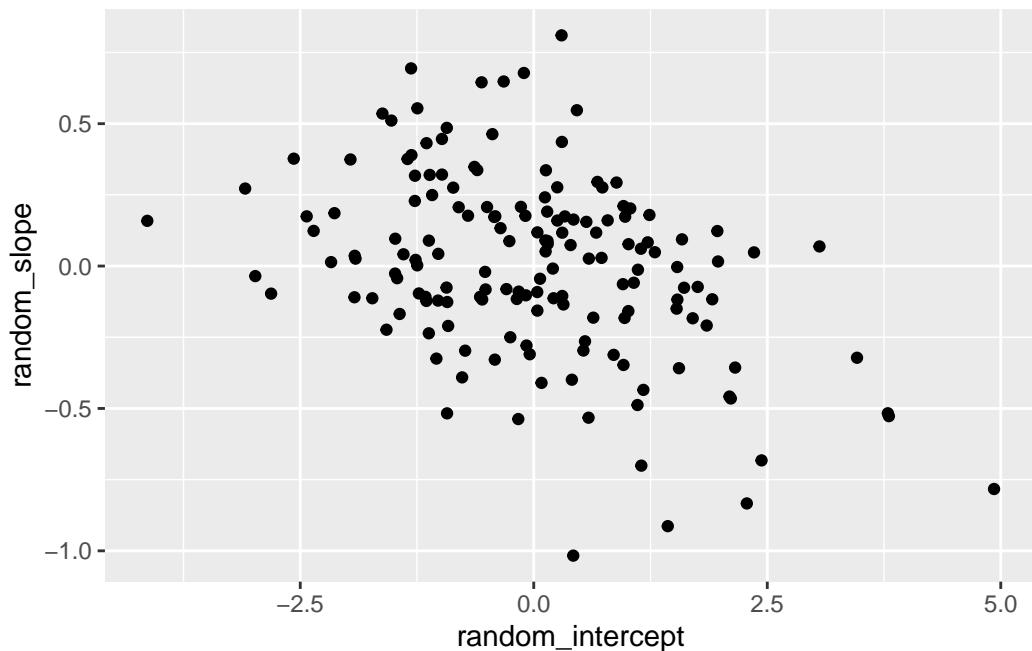
# A tibble: 6 x 3
  schoolid random_intercept random_slope
  <fct>          <dbl>        <dbl>
1 8367            -4.14       0.159
2 4523            -3.09       0.272
3 6990            -2.98      -0.0353
4 3705            -2.81      -0.0968
5 8854            -2.57       0.377
6 9397            -2.43       0.174

```

```

ggplot(blups, aes(x = random_intercept, y = random_slope)) +
  geom_point()

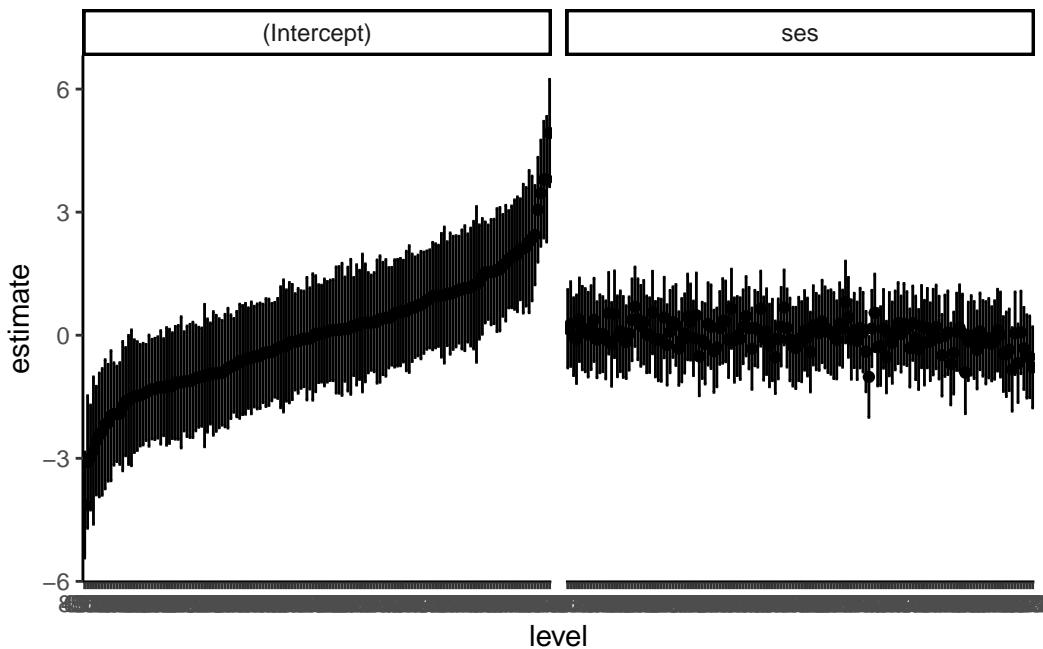
```



16.2.5 Caterpillar Plots

The included information as a data frame makes it easy to construct caterpillar plots!

```
ri <- lmer |>
  ranef() |>
  augment()
ggplot(ri, aes(x = level, y = estimate,
               ymin = lb,
               ymax = ub)) +
  facet_wrap(~ variable, nrow = 1) +
  geom_point() +
  geom_errorbar() +
  theme_classic()
```



16.2.6 Fitted Values

Using `augment` directly on the `lmer` object gives us fitted values (`.fitted`) and residuals (`.resid`). We can use this for residual plots or for plotting lines for each school.

```

mlm |>
  augment() |>
  head()

```

A tibble: 6 x 15

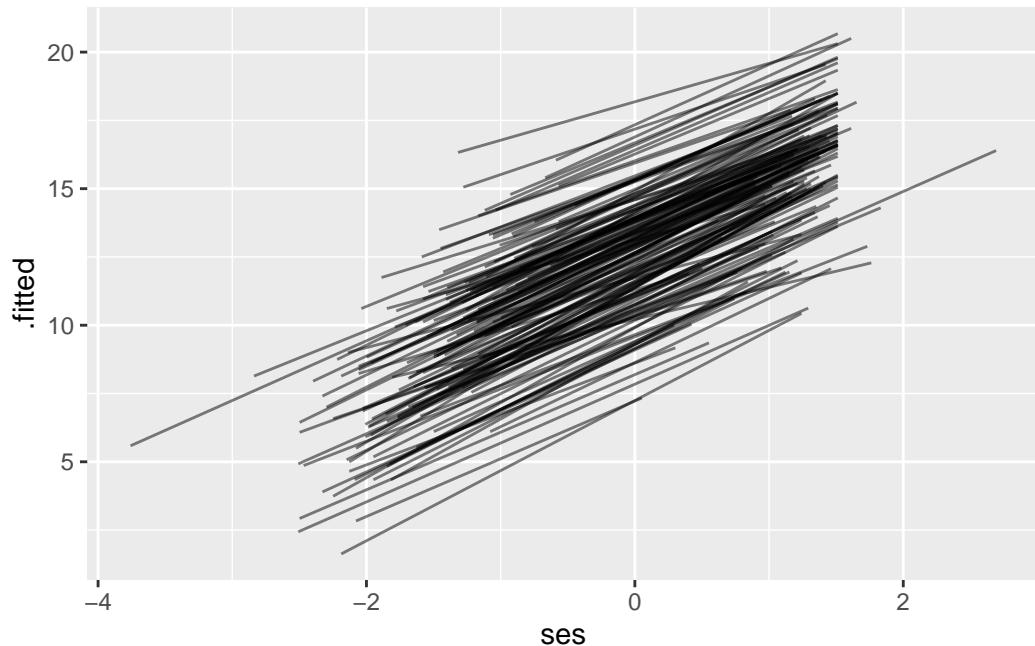
	mathach	ses	mnses	schoolid	.fitted	.resid	.hat	.cooksdi	.fixed	.mu	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	5.88	-1.53	-0.434	1224	7.29	-1.41	0.0325	0.000629	7.68	7.29					
2	19.7	-0.588	-0.434	1224	9.43	10.3	0.0177	0.0175	9.74	9.43					
3	20.3	-0.528	-0.434	1224	9.57	10.8	0.0173	0.0188	9.87	9.57					
4	8.78	-0.668	-0.434	1224	9.25	-0.468	0.0183	0.0000376	9.57	9.25					
5	17.9	-0.158	-0.434	1224	10.4	7.49	0.0164	0.00863	10.7	10.4					
6	4.58	0.0220	-0.434	1224	10.8	-6.24	0.0170	0.00619	11.1	10.8					

i 5 more variables: .offset <dbl>, .sqrtXwt <dbl>, .sqrtrwt <dbl>,
.weights <dbl>, .wtres <dbl>

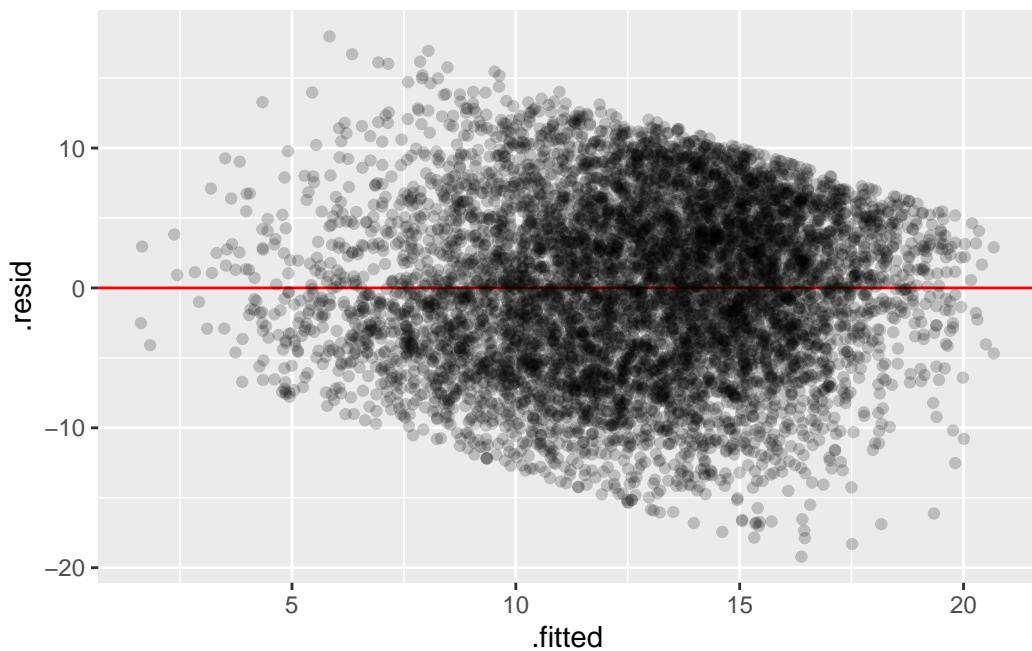
```

# fitted lines
mlm |>
  augment() |>
  ggplot(aes(x = ses, y = .fitted, group = schoolid)) +
  geom_line( alpha=0.5 )

```



```
# residuals
mlm |>
  augment() |>
  ggplot(aes(y = .resid, x = .fitted)) +
  geom_hline(yintercept = 0, color = "red") +
  geom_point(alpha = 0.2)
```



16.3 Additional Resources

I've recently discovered the packaged `mixedup` that has some excellent additional convenience functions for extracting info from `lmer` models: <https://m-clark.github.io/mixedup/index.html>.

It might be worth checking out as well!

17 Extracting information from fitted lmer models using base R

This chapter follows Chapter Chapter ??, and provides an alternate set of ways of pulling information from a fit `lmer` model. In particular, this document walks through various R code to pull information out of a multilevel model (and OLS models as well, since the methods generally work on everything). For illustration, we will use a random-slope model on the HS&B dataset with some level 1 and level 2 fixed effects.

We use the following libraries in this file:

```
library( lme4 )
library( foreign ) ## to load data
library( arm )
library( tidyverse )
```

Loading the data is simple. We read student and school level data and merge:

```
dat = read.spss( "data/hsb1.sav", to.data.frame=TRUE )
sdat = read.spss( "data/hsb2.sav", to.data.frame=TRUE )
```

re-encoding from CP1252

```
dat = merge( dat, sdat, by="id", all.x=TRUE )
head( dat, 3 )
```

```
   id minority female      ses mathach size sector pracad disclim himinty
1 1224        0       1 -1.528   5.876  842        0    0.35   1.597      0
2 1224        0       1 -0.588  19.708  842        0    0.35   1.597      0
3 1224        0       0 -0.528  20.349  842        0    0.35   1.597      0
  meanses
1 -0.428
2 -0.428
3 -0.428
```

17.1 Fitting and viewing the model

Now we fit the random slope model with the level-2 covariates:

```
M1 = lmer( mathach ~ 1 + ses + meanses + (1 + ses|id), data=dat )
```

If we just print the object, e.g., by typing the name of the model on the console, we get minimal information:

```
M1
```

```
Linear mixed model fit by REML ['lmerMod']
Formula: mathach ~ 1 + ses + meanses + (1 + ses | id)
Data: dat
REML criterion at convergence: 46561.42
Random effects:
 Groups   Name        Std.Dev. Corr
 id       (Intercept) 1.6417
           ses          0.6731  -0.21
 Residual            6.0659
Number of obs: 7185, groups: id, 160
Fixed Effects:
(Intercept)      ses        meanses
 12.651       2.190      3.781
```

17.1.1 The `display()` method

The `arm` package's `display()` method gives an overview of what our fitted model is:

```
display( M1 )
```

```
lmer(formula = mathach ~ 1 + ses + meanses + (1 + ses | id),
      data = dat)
      coef.est coef.se
(Intercept) 12.65     0.15
ses         2.19     0.12
meanses     3.78     0.38

Error terms:
 Groups   Name        Std.Dev. Corr

```

```

id      (Intercept) 1.64
       ses          0.67     -0.21
Residual          6.07
---
number of obs: 7185, groups: id, 160
AIC = 46575.4, DIC = 46552.4
deviance = 46556.9

```

17.1.2 The `summary()` method

We can also look at the messier default `summary()` command, which gives you more output. The real win is if we use the `lmerTest` library and fit our model with that package loaded, our `summary()` is more exciting and has *p*-values:

```

library( lmerTest )
M1 = lmer( mathach ~ 1 + ses + meanses + (1 + ses|id), data=dat )
summary( M1 )

```

```

Linear mixed model fit by REML. t-tests use Satterthwaite's method [ lmerModLmerTest]
Formula: mathach ~ 1 + ses + meanses + (1 + ses | id)
Data: dat

REML criterion at convergence: 46561.4

Scaled residuals:
    Min      1Q  Median      3Q      Max 
-3.1671 -0.7270  0.0163  0.7547  2.9646 

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 id       (Intercept) 2.695    1.6417  
           ses          0.453    0.6731   -0.21
 Residual            36.796   6.0659  
Number of obs: 7185, groups: id, 160

Fixed effects:
            Estimate Std. Error    df t value Pr(>|t|)    
(Intercept) 12.6513    0.1506 152.9599 84.000 <2e-16 ***
ses         2.1903    0.1218 178.2055 17.976 <2e-16 ***
meanses    3.7812    0.3826 181.7675  9.883 <2e-16 ***

```

```

---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
  (Intr) ses
ses   -0.080
meanses -0.028 -0.256

```

17.2 Obtaining Fixed Effects

R thinks of all models in reduced form. Thus when we get the fixed effects we get both the level-1 and level-2 fixed effects all together:

```
fixef( M1 )
```

(Intercept)	ses	meanses
12.651300	2.190350	3.781218

The above is a vector of numbers. Each element is named, but we can index them as so:

```
fixef( M1 )[2]
```

ses
2.19035

We can also use the `[[[]]]` which means “give me that element not as a list but as just the element!” When in doubt, if you want one thing out of a list or vector, use `[[[]]]` instead of `[]`:

```
fixef( M1 )[[2]]
```

```
[1] 2.19035
```

See how it gives you the number without the name here?

17.3 Obtaining Variance and Covariance estimates

We can get the Variance-Covariance matrix of the random effects with `VarCorr`.

```
VarCorr( M1 )
```

Groups	Name	Std.Dev.	Corr
id	(Intercept)	1.64174	
	ses	0.67309	-0.212
Residual		6.06594	

It displays nicely if you just print it out, but inside it are covariance matrices for each random effect group. (In our model we only have one group, `id`.) These matrices also have correlation matrices for reference. Here is how to get these pieces:

```
vc = VarCorr( M1 )$id
vc
```

```
(Intercept)      ses
(Intercept)  2.6953203 -0.2339045
ses          -0.2339045  0.4530494
attr(,"stddev")
(Intercept)      ses
1.6417431   0.6730894
attr(,"correlation")
(Intercept)      ses
(Intercept)  1.0000000 -0.2116707
ses          -0.2116707  1.0000000
```

You might be wondering what all the `attr` stuff is. R can “tack on” extra information to a variable via “attributes”. Attributes are not part of the variable exactly, but they follows their variable around. The `attr` (for attribute) method is a way to get these extra bits of information. In the above, R is tacking the correlation matrix on to the variance-covariance matrix to save you the trouble of calculating it yourself. Get it as follows:

```
attr( vc, "correlation" )
```

```
(Intercept)      ses
(Intercept)  1.0000000 -0.2116707
ses          -0.2116707  1.0000000
```

You can also just use the `vc` object as a matrix. Here we take the diagonal of it

```
diag( vc )
```

```
(Intercept)      ses  
2.6953203   0.4530494
```

If you want an element from a matrix use row-column indexing like so:

```
vc[1,2]
```

```
[1] -0.2339045
```

for row 1 and column 2.

17.3.0.1 The sigma.hat() and sigma() methods

If you just want the variances and standard deviations of your random effects, use `sigma.hat()`. This also gives you the residual standard deviation as well. The output is a weird object, with a list of things that are themselves lists in it. Let's examine it. First we look at what the whole thing is:

```
sigma.hat( M1 )
```

```
$sigma  
$sigma$data  
[1] 6.065939
```

```
$sigma$id  
(Intercept)      ses  
1.6417431   0.6730894
```

```
$cors  
$cors$data  
[1] NA  
  
$cors$id  
          (Intercept)      ses  
(Intercept)  1.0000000 -0.2116707  
ses         -0.2116707  1.0000000
```

```
names( sigma.hat( M1 ) )
```

```
[1] "sigma" "cors"
```

```
sigma.hat( M1 )$sigma
```

```
$data
```

```
[1] 6.065939
```

```
$id
```

(Intercept)	ses
1.6417431	0.6730894

Our standard deviations of the random effects are

```
sigma.hat( M1 )$sigma$id
```

(Intercept)	ses
1.6417431	0.6730894

We can get our residual variance by this weird thing (we are getting `data` from the `sigma` inside of `sigma.hat(M1)`):

```
sigma.hat( M1 )$sigma$data
```

```
[1] 6.065939
```

But here is an easier way using the `sigma()` utility function:

```
sigma( M1 )
```

```
[1] 6.065939
```

17.4 Obtaining Empirical Bayes Estimates of the Random Effects

Random effects come out of the `raneff()` method. Each random effect is its own object inside the returned object. You refer to these sets of effects by name. Here our random effect is called `id`.

```

estts = ranef( M1 )$id
head( estts )

(Intercept)      ses
1224 -0.26204371  0.08765385
1288  0.03805199  0.11841938
1296 -1.91525901  0.03572247
1308  0.30485682 -0.10500515
1317 -1.15834807 -0.10815301
1358 -0.98212459  0.44612877

```

Generally, what you get back from these calls is a new data frame with a row for each group. The rows are named with the original id codes for the groups, but if you want to connect it back to your group-level information you are going to want to merge stuff. To do this, and to keep things organized, I recommend adding the id as a column to your dataframe:

```

names(estts) = c( "u0", "u1" )
estts$id = rownames( estts )
head( estts )

```

	u0	u1	id
1224	-0.26204371	0.08765385	1224
1288	0.03805199	0.11841938	1288
1296	-1.91525901	0.03572247	1296
1308	0.30485682	-0.10500515	1308
1317	-1.15834807	-0.10815301	1317
1358	-0.98212459	0.44612877	1358

We also renamed our columns of our dataframe to give them names nicer than `(Intercept)`. You can use these names if you wish, however. You just need to quote them with back ticks (this code is not run):

```
head( estts$`(Intercept)` )
```

17.4.1 The `coef()` method

We can also get a slightly different (but generally easier to use) version these things through `coef()`. What `coef()` does is give you the estimated regression lines for each group in your data by combining the random effect for each group with the corresponding fixed effects. Note how in the following the `meanses` coefficient is the same, but the others vary due to the random slope and random intercept.

```

coefs = coef( M1 )$id
head( coefs )

(Intercept)      ses  meanses
1224     12.38926 2.278004 3.781218
1288     12.68935 2.308769 3.781218
1296     10.73604 2.226072 3.781218
1308     12.95616 2.085345 3.781218
1317     11.49295 2.082197 3.781218
1358     11.66918 2.636479 3.781218

```

Note that if we have level 2 covariates in our model, they are not incorporated in the intercept and slope via `coef()`. We have to do that by hand:

```

names( coefs ) = c( "beta0.adj", "beta.ses", "beta.meanses" )
coefs$id = rownames( coefs )
coefs = merge( coefs, sdat, by="id" )
coefs = mutate( coefs, beta0 = beta0.adj + beta.meanses * meanses )
coefs$beta.meanses = NULL

```

Here we added in the impact of mean ses to the intercept (as specified by our model). Now if we look at the intercepts (the beta0 variables) they will incorporate the level 2 covariate effects. If we then plotted a line using beta0 and beta.ses for each school, we would get the estimated lines for each school including the school-level covariate impacts.

17.5 Obtaining standard errors

We can get an object with all the standard errors of the coefficients, including the individual Empirical Bayes estimates for the individual random effects. This is a lot of information. We first look at the Standard Errors for the fixed effects, and then for the random effects. Standard errors for the variance terms are not given (this is trickier to calculate).

17.5.1 Fixed effect standard errors

```

ses = se.coef( M1 )
names( ses )

```

```
[1] "fixef" "id"
```

Our fixed effect standard errors:

```
ses$fixef
```

```
[1] 0.1506106 0.1218474 0.3826085
```

You can also get the uncertainty estimates of your fixed effects as a variance-covariance matrix:

```
vcov( M1 )
```

```
3 x 3 Matrix of class "dpoMatrix"
      (Intercept)      ses      meances
(Intercept) 0.022683560 -0.001465374 -0.001619405
ses         -0.001465374  0.014846788 -0.011954182
meances     -0.001619405 -0.011954182  0.146389293
```

The standard errors are the diagonal of this matrix, square-rooted. See how they line up?:

```
sqrt( diag( vcov( M1 ) ) )
```

```
(Intercept)      ses      meances
0.1506106   0.1218474  0.3826085
```

17.5.2 Random effect standard errors

Our random effect standard errors for our EB estimates:

```
head( ses$id )
```

```
(Intercept)      ses
1224    0.7845859 0.5804186
1288    0.9819216 0.6277115
1296    0.7779963 0.5766319
1308    1.0911690 0.6556607
1317    0.8045695 0.6188535
1358    0.9163545 0.6173954
```

Warning: these come as a matrix, not data frame. It is probably best to do this:

```
SEs = as.data.frame( se.coef( M1 )$id )
head( SEs )
```

```
(Intercept)      ses
1224    0.7845859  0.5804186
1288    0.9819216  0.6277115
1296    0.7779963  0.5766319
1308    1.0911690  0.6556607
1317    0.8045695  0.6188535
1358    0.9163545  0.6173954
```

17.6 Generating confidence intervals

We can compute profile confidence intervals (warnings have been suppressed)

```
confint( M1 )
```

```
              2.5 %     97.5 %
.sig01      1.4012799  1.8897548
.sig02      -0.8761947  0.1946551
.sig03      0.2165284  0.9849953
.sigma      5.9659922  6.1689341
(Intercept) 12.3559620 12.9462385
ses         1.9512025  2.4296954
meanses    3.0278220  4.5329237
```

17.7 Obtaining fitted values

Fitted values are the predicted value for each individual given the model.

```
yhat = fitted( M1 )
head( yhat )
```

```
1          2          3          4          5          6
7.290105  9.431429  9.568109  9.249189 10.410971 10.821011
```

Residuals are the difference between predicted and observed:

```
resids = resid( M1 )
head( resids )
```

```
1           2           3           4           5           6
-1.4141055 10.2765710 10.7808908 -0.4681887  7.4870293 -6.2380113
```

We can also predict for hypothetical new data. Here we predict the outcome for a random student with ses of -1, 0, and 1 in a school with mean ses of 0:

```
ndat = data.frame( ses = c( -1, 0, 1 ), meanses=c(0,0,0), id = -1 )
predict( M1, newdata=ndat, allow.new.levels=TRUE )
```

```
1           2           3
10.46095 12.65130 14.84165
```

The `allow.new.levels=TRUE` bit says to predict for a new school (our fake school id of -1 in `ndat` above). In this case it assumes the new school is typical, with 0s for the random effect residuals.

If we predict for a current school, the random effect estimates are incorporated:

```
ndat$id = 1296
predict( M1, newdata=ndat )
```

```
1           2           3
8.509969 10.736041 12.962114
```

17.8 Appendix: the guts of the object

When we fit our model and store it in a variable, R stores *a lot* of stuff. The following lists some other functions that pull out bits and pieces of that stuff.

First, to get the model matrix (otherwise called the design matrix)

```
mm = model.matrix( M1 )
head( mm )
```

```
(Intercept)      ses meanses
1            1 -1.528 -0.428
2            1 -0.588 -0.428
3            1 -0.528 -0.428
4            1 -0.668 -0.428
5            1 -0.158 -0.428
6            1  0.022 -0.428
```

This can be useful for predicting individual group mean outcomes, for example.

We can also ask questions such as number of groups, number of individuals:

```
ngrps( M1 )
```

```
id
160
```

```
nobs( M1 )
```

```
[1] 7185
```

We can list all methods for the object (`merMod` is a more generic version of `lmerMod` and has a lot of methods we can use)

```
class( M1 )
```

```
[1] "lmerModLmerTest"
attr(,"package")
[1] "lmerTest"
```

```
methods(class = "lmerMod")
```

```
[1] coerce      coerce<-
[7] getL       mcsamp     se.coef      show        sim         standardize
see '?methods' for accessing help and source code
```

```
methods(class = "merMod")
```

```
[1] anova           as.function    coef          confint        cooks.distance
[6] deviance        df.residual   display        drop1          extractAIC
[11] extractDIC     family         fitted         fixef          formula
[16] getData         getL           getME          hatvalues      influence
[21] isGLMM          isLMM          isNLMM         isREML         logLik
[26] mcsamp          model.frame   model.matrix  ngrps          nobs
[31] plot            predict        print          profile        ranef
[36] refit           refitML       repCA          residuals     rstudent
[41] se.coef          show           sigma.hat     sigma          sim
[46] simulate        standardize  summary        terms          update
[51] VarCorr          vcov           weights
see '?methods' for accessing help and source code
```

18 Interpreting Coefficients

18.1 Interpreting your models (they won't interpret themselves!)

So, multilevel models sure are great, but they can also make interpretations much more challenging. You've done OLS regression, so you have an understanding of how to interpret regression coefficients. However, adding additional levels means that some of our interpretations also need to change. This document is intended to provide a brief guide to how to do that.

Coefficients and indices at various levels of the model

But before we even start, we need to talk about how we use different coefficients and letters at different levels of the model. There isn't a single convention for how to do this, but we'll try to be consistent at least in this class.

We'll distinguish between two basic types of models, those that are multilevel and *not* longitudinal, and those that *are* longitudinal.

As a canonical example of the first type, let's consider the model we use in class, namely

$$\begin{aligned} \text{mathach}_{ij} &= \beta_{0j[i]} + \beta_{1j[i]} \text{SES}_i + \varepsilon_i, \\ \beta_{0j} &= \gamma_{00} + \gamma_{01} \text{sector}_j + u_{0j}, \\ \beta_{1j} &= \gamma_{10} + \gamma_{11} \text{sector}_j + u_{1j}, \\ \varepsilon_i &\sim \text{Normal}(0, \sigma_\varepsilon^2) \\ \begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} &\sim N \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_0^2 & \rho\sigma_0\sigma_1 \\ \rho\sigma_0\sigma_1 & \sigma_1^2 \end{pmatrix} \right] \end{aligned}$$

Here are the features of the model to attend to. When referring to students (or other first-level units), we will use i as a subscript. X_i will indicate a measurement taken for the i th student. When referring to schools (or other second-level units), we will j as a subscript. X_j will indicate a measurement taken for the j th school. When we expand these models to include third-level units (e.g., districts), we will use the subscript k for these units. I don't intend to go past that, although we could. When we introduce cross-classified models (i.e., models will non-nested hierarchies) we'll pick subscripts that are intended to be evocative.

We'll also try to be consistent when using coefficients. We'll use the letter β (beta) to indicate regression coefficients measured at the first level. We'll use the letter γ (gamma) to indicate regression coefficients measured at the second level. Eventually we'll use the letter ξ (xi, or ksi) to indicate regression coefficients measured at the third level.

When we subscript regression coefficients, we'll need a number of subscripts equal to the level of the model at which this coefficient has been entered. The first subscript will indicate the level-1 coefficient with which this particular coefficient is associated, the second subscript will indicate the level-2 coefficient with which it is associated, and so on. This means that each coefficient will have a number of subscripts equal to the level of the model. As a really complicated example, if a coefficient is labeled as ξ_{021} , this indicates that the coefficient is the first slope coefficient (the 1 at the end) in a model for the second level-2 slope coefficient (the 2 in the second position) in a model for the level-1 intercept. Similarly, the first subscript in a random effect will indicate the level-1 coefficient with which it is associated, and the second will indicate the level-2 coefficient with which it is associated. Random effects will always have one fewer subscript than the coefficients at that level. As you can imagine, subscripts quickly get out of hand as we introduce more and more levels to a model.

We'll use σ_p^2 to indicate the variance of the level-2 residual for the p th random effect (starting at 0 for the intercept). I'm not yet sure how to do the subscripting at level-3, and for now am hoping to just wing it. The correlation between the p th and q th random effects will be subscripted pq , and correlations will always be identified with a ρ (rho, not p).

Longitudinal models are similar, except for the subscripting. I'll always (probably) subscript the first level with t , for time. The second level will become i (assuming that we're looking at growth in students or other individuals), followed by j for the third level (we probably won't include a fourth level).

Interpreting fixed effects

Okay, that was complicated, although I think writing down definitions and rules is often more challenging than applying them. Now let's practice some interpretations, going back to our model.

At the first level, we interpret (almost) exactly as we would in a standard regression model. If we have

$$mathach_{ij} = \beta_{0j[i]} + \beta_{1j[i]} SES_i + \varepsilon_i,$$

then we interpret β_{0j} as the predicted value of *mathach* for a student of 0 SES (which represents the grand mean) *who is located in school j*. Because this is a multilevel model, different schools have different intercepts. Similarly, we can interpret $\beta_{1j[i]}$ as the expected difference in math achievement associated with a one-unit difference in SES *for students in school j*. We don't

interpret it, but ε_i indicates the difference between what we observed for this student and what we predicted based on her or his school and SES.

We interpret the level-2 units depending on the coefficients they predict. For the school-intercept we have

$$\beta_{0j} = \gamma_{00} + \gamma_{01} \text{sector}_j + u_{0j}.$$

We can interpret γ_{00} as the predicted intercept for schools for which $\text{sector} = j$ (i.e., public schools). We can interpret γ_{01} as the predicted difference in school intercepts between Catholic and public schools. Although it's less common, we can also interpret the residual for school j , u_{0j} , because you can't tell me what to do. u_{0j} represents the difference between the observed/inferred intercept for school j and the predicted intercept.

Turning to the model for the slope, we have

$$\beta_{1j} = \gamma_{10} + \gamma_{11} \text{sector}_j + u_{1j}.$$

Here γ_{10} is the predicted slope for SES in public schools, while γ_{11} is the mean difference in slopes between Catholic and public school. Finally, u_{1j} is the difference between the slope observed/inferred for school j and the slope predicted by the model.

We can *also* interpret these coefficients at the student level. Rewrite the model by substituting $\beta_{0j} = \gamma_{00} + \gamma_{01} \text{sector}_j + u_{0j}$ and $\beta_{1j} = \gamma_{10} + \gamma_{11} \text{sector}_j + u_{1j}$ to obtain

$$\begin{aligned} \text{mathach}_i &= \gamma_{00} + \gamma_{01} \text{sector}_{j[i]} + u_{0j[i]} + (\gamma_{10} + \gamma_{11} \text{sector}_{j[i]} + u_{1j[i]}) \text{SES}_i + \varepsilon_i \\ &= \gamma_{00} + \gamma_{01} \text{sector}_{j[i]} + \gamma_{10} \text{SES}_i + \gamma_{11} \text{sector}_{j[i]} \text{SES}_i + (u_{0j[i]} + u_{1j[i]} \text{SES}_i + \varepsilon_i). \end{aligned}$$

Now we can interpret these coefficients as in a typical one-level linear regression model.

1. γ_{00} is the predicted mean value of mathach for students of $\text{SES} = 0$ in public schools;
2. γ_{01} is the predicted difference in mathach between students of $\text{SES} = 0$ in Catholic schools and similar peers in public schools;
3. γ_{10} is the predicted difference in mathach associated with a one-unit difference in SES for students in public schools; and
4. γ_{11} is the predicted difference in the above difference between students in Catholic schools and students in public schools.

Either interpretation is acceptable, and you should base your decision on how you're framing your question.

Interpreting variance-covariance parameters

Now we're going to turn to the variance-covariance matrix for the random offsets, namely

$$\Sigma = \begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} \sim N \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_0^2 & \rho\sigma_0\sigma_1 \\ \rho\sigma_0\sigma_1 & \sigma_1^2 \end{pmatrix} \right]$$

The variance of a random offset (e.g., σ_0^2 , the variance of u_{0j}) represents how variable the coefficient associated with that coefficient is, conditional on the variables in the model. The correlations (e.g., ρ , the only correlation in this model) represent the tendency of the random offsets to covary, i.e., to be associated with each other.

19 Within, Between, and Contextual Effects

Many find it hard to keep track of within, between, and contextual effects in MLMs. This short walkthrough shows how to fit and interpret each model using the HSB data.

```
# load libraries
library(tidyverse)
library(lme4)
library(sjPlot)
library(ggeffects)
library(haven)

# clear memory
rm(list = ls())

select <- dplyr::select

# load HSB data
hsb <- read_dta("data/hsb.dta") |>
  select(mathach, ses, schoolid) |>
  group_by(schoolid) |>
  mutate(grp_mean_ses = mean(ses)) |>
  ungroup() |>
  mutate(grp_center_ses = ses - grp_mean_ses)
```

19.1 Fitting the Models

```
ols <- lm(mathach ~ ses, hsb)
fe <- lm(mathach ~ ses + factor(schoolid), hsb)
ri <- lmer(mathach ~ ses + (1|schoolid), hsb)
ri_within <- lmer(mathach ~ grp_center_ses + (1|schoolid), hsb)
ri_between <- lmer(mathach ~ grp_mean_ses + (1|schoolid), hsb)
re_wb <- lmer(mathach ~ grp_center_ses + grp_mean_ses + (1|schoolid), hsb)
contextual <- lmer(mathach ~ ses + grp_mean_ses + (1|schoolid), hsb)
```

```
tab_model(ols, fe, ri, ri_within, ri_between, re_wb, contextual,
          p.style = "stars",
          show.ci = FALSE,
          show.se = TRUE,
          keep = "ses",
          show.dev = TRUE,
          dv.labels = c("OLS",
                      "Fixed Effects",
                      "Rand. Int.",
                      "RI Within",
                      "RI Between",
                      "REWB",
                      "Mundlak"))
```

OLS

Fixed Effects

Rand. Int.

RI Within

RI Between

REWB

Mundlak

Predictors

Estimates

std. Error

Estimates

std. Error

Estimates

std. Error

Estimates

std. Error

Estimates

std. Error
Estimates
std. Error
Estimates
std. Error
ses
3.18 ***
0.10
2.19 ***
0.11
2.39 ***
0.11
2.19 ***
0.11
grp center ses
2.19 ***
0.11
2.19 ***
0.11
grp mean ses
5.86 ***
0.36
5.87 ***
0.36
3.68 ***
0.38
Random Effects

37.03

37.01

39.16

37.02

37.02

00

4.77 schoolid

8.67 schoolid

2.64 schoolid

2.69 schoolid

2.69 schoolid

ICC

0.11

0.19

0.06

0.07

0.07

N

160 schoolid

160 schoolid

160 schoolid

160 schoolid

160 schoolid

Observations

7185

7185

7185

7185

7185

7185

7185

R² / R² adjusted

0.130 / 0.130

0.235 / 0.218

0.077 / 0.182

0.044 / 0.225

0.123 / 0.179

0.167 / 0.224

0.167 / 0.224

Deviance

295643.779

259918.446

46641.008

46720.415

46959.128

46563.821

46563.821

* p<0.05 ** p<0.01 *** p<0.001

19.2 Interpretation

19.2.1 OLS

```
lm(formula = mathach ~ ses, data = hsb)
```

Ignoring school membership, students who are 1-unit higher in SES are predicted to score 3.18 points higher in math. This is generally not a preferred model.

19.2.2 Fixed Effects

```
lm(formula = mathach ~ ses + factor(schoolid), data = hsb)
```

Holding constant school, students who are 1-unit higher in SES are predicted to score 2.19 points higher in math. Fixed effects models focus on within-school comparisons: we are looking at how students within schools relate to each other, and then averaging this relationship across all our schools to get our final estimate.

19.2.3 Random Intercepts

```
lmer(formula = mathach ~ ses + (1 | schoolid), data = hsb)
```

Students who are 1-unit higher in SES are predicted to score 2.39 points higher in math; schools that are 1-unit higher in mean SES are predicted to have mean math scores 2.39 points higher.

The random intercept model gives a precision-weighted average of the within and between effects. Looking at the other models, note that our within effect is 2.19 and our between effect is 5.86. If the RE assumption holds, these are the same in the population, so we get more precision by averaging them together. However, in social science, they are rarely the same, making this model provide a weird blend of two kinds of mechanism.

19.2.4 Random Intercepts, Within Effect

```
lmer(formula = mathach ~ grp_center_ses + (1 | schoolid), data = hsb)
```

Holding constant school, students who are 1-unit higher in SES are predicted to score 2.19 points higher in math. This is the same coefficient as the FE model, but in an RI framework. We have “controlled for school” manually by demeaning the SES variable.

19.2.5 Random Intercepts, Between

```
lmer(formula = mathach ~ grp_mean_ses + (1 | schoolid), data = hsb)
```

Schools that are 1-unit higher in mean SES are predicted to have mean math scores 5.86 points higher.

19.2.6 Random Effects within and Between

```
lmer(formula = mathach ~ grp_center_ses + grp_mean_ses + (1 |  
schoolid), data = hsb)
```

Holding constant school, students who are 1-unit higher in SES are predicted to score 2.19 points higher in math; schools that are 1-unit higher in mean SES are predicted to have mean math scores 5.86 points higher. We get the within and between effects in a single model!

19.2.7 Contextual/Mundlak

```
lmer(formula = mathach ~ ses + grp_mean_ses + (1 | schoolid),  
data = hsb)
```

Holding constant school, students who are 1-unit higher in SES are predicted to score 2.19 points higher in math; *holding constant student SES*, a student that attends a school with 1-unit higher in mean SES are predicted to have mean math scores 3.68 points higher. The contextual effect is the *difference* in the within and between effects (note that $5.86 - 2.19 = 3.68$, up to rounding), and, in principle, its significance test allows us to determine if having both is necessary. Mathematically, the Mundlak model and REWB are *identical*, as you can see from the deviance statistics. You would choose one over the other depending on your preferred interpretation.

19.3 Further Reading

(Antonakis, Bastardoz, and Rönkkö 2019)

20 MLM Assumptions

20.1 Oh, assumptions

There are generally two kinds of assumptions we should worry about the most: omitted variable bias, and independence assumptions. The latter of these is one we should always think about.

Do read Chapter 9 of R&B, paying attention to their examples and not so much to the mathematical formalism. It has some dense prose, but then moves to specific diagnostics that make what they are talking about much more clear (and it also provides things you can do to check assumptions). The *MLM in Plain Language* text has some simpler explanations. Also see below for some further notes.

20.2 Ommitted variable bias

Consider the following numerical example:

```
N = 100
dat = data.frame( X1 = rnorm( N ) )
dat = mutate( dat,
             X2 = X1 + rnorm( N ),
             Y = 3 + 0.5 * X1 + 1.5 * X2 + rnorm( N ) )
```

The above makes an `X2` that is correlated with `X1`, and a `Y` that is a function of both. The true model here is

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \epsilon_i$$

with coefficents $\beta = (3, 0.5, 1.5)$.

We fit two models, one with both covariates, and one with only one:

```
M0 = lm( Y ~ 1 + X1 + X2 , data = dat )
M1 = lm( Y ~ 1 + X1, data = dat )
```

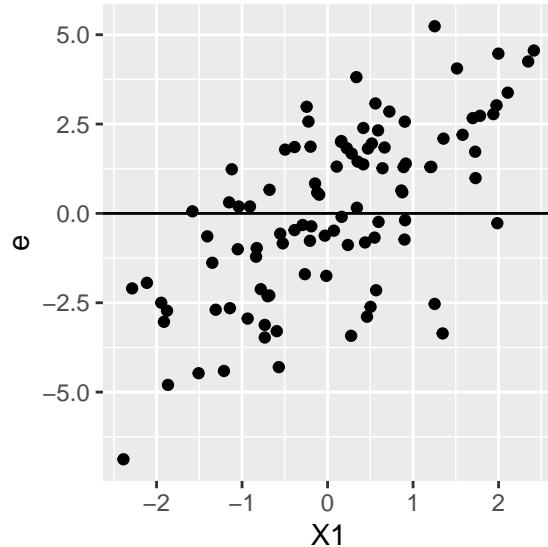
Our results:

```
tab_model(M0, M1, p.style = "stars",
          show.ci = FALSE, show.se = TRUE)
```

	Y		Y	
Predictors	Estimates	std. Error	Estimates	std. Error
(Intercept)	3.10 ***	0.11	2.95 ***	0.18
X1	0.44 **	0.15	1.92 ***	0.16
X2	1.51 ***	0.11		
Observations	100		100	
R ² / R ² adjusted	0.856		0.584	
	/		/	
	0.853		0.580	
	*			
p<0.05	**			
p<0.01	***			
p<0.001				

Note our coefficient is completely wrong when we omit a correlated variable. This is omitted variable bias, and in terms of our assumptions we are in a circumstance where the true residuals in our model are not centered around 0 for all values of X1, since they include the X2 effect which is correlated with X1. We can see this as follows:

```
dat = mutate( dat, e = Y - 3 - 0.5 * X1 )
ggplot( dat, aes( X1, e ) ) +
  geom_point() +
  geom_hline( yintercept = 0 )
```



Note how our residuals (which includes X_2) are positive for bigger X_1 , due to the correlation of X_1 and X_2 .

Conclusion: On one hand, we have the wrong estimate for β_1 . On the other, the estimate we do get is fine if we view it as the best description of the data. We just need to remember that the interpretation of our coefficient includes the confounding effect of X_2 on X_1 .

In this vein:

Q: You mentioned during class that we don't care too much about assumptions when looking at trends in the data. However, if we are trying to draw causal inferences, do these assumptions become more important?

A: Even with the assumptions causal inference would depend on the model. Modern causal inference has a hard time with this, so you need other strategies such as quasiexperimental design. Hence my focus on descriptive aspects of data analysis.

20.3 Independence assumptions

The independence assumptions are key. When we do not take violations of independence into account, we can be overly confident of our estimates.

Generally with MLM we should think of these assumptions in terms of how we sampled our data. If we sampled our data by sampling a collection of schools, and then individuals within those schools, then we have two levels. We then need to ask two questions:

- (1) Were the schools sampled independently?

(2) Were the students sampled independently within the schools?

If yes to both, we have met both our independence assumptions! We have met them even if the students are clustered in classes within their schools. As long as we did not sample using those classes (or other clusters), we are ok as our sample of students will be representative of the school they are in.

One might then ask, more generally, if there is a problem with clustering if it's not part of the sampling plan? E.g. if you sampled at the school level and surveyed all students, there is still natural clustering in classrooms: is that a problem? What about unobserved clustering like families, neighborhoods, etc. which are not part of sampling, but do exist naturally in populations?

E.g., see this [document](#) which says clustered SEs are *not* necessary (in OLS) unless sampling was conducted at the cluster-level and that econometricians often overuse them.

This is indeed correct. That being said, we might want to make clusters to investigate how things vary across those clusters.

Q: More generally, is there a problem with clustering if it's not part of the sampling plan? E.g. if you sampled at the school level and surveyed all students, there is still natural clustering in classrooms: is that a problem? What about unobserved clustering like families, neighborhoods, etc. which are not part of sampling, but do exist naturally in populations? Q: I'll let Luke respond to how this affects the assumptions later on Piazza. My intuition is that we want to cluster/add levels at these different levels if we believe the outcomes or predictors are correlated Q: Thanks. I remember reading this <https://blogs.worldbank.org/impactevaluations/when-should-you-cluster-standard-errors-new-wisdom-econometrics-oracle> a few years ago, which says clustered SEs are *not* necessary (in OLS) unless sampling was conducted at the cluster-level and that econometricians often overuse them, but I am wondering to what degree this holds in MLMs.

A: This is correct. That being said, we might want to make clusters to investigate how things vary across those clusters.

20.4 Number of clusters needed?

Ok, this isn't an assumption per se, but onwards!

Q: Why should you worry if the number of group is small? (referencing the recap slide)

A: With few clusters, estimation is hard just like having a small dataset with OLS. The variance parameters in particular are difficult.

Q: When you say "at least 20" you mean for the number of j's, right?

A: Yes, number of clusters. Mostly Harmless Econometrics readers might recall a discussion of 42 clusters (8.2.3), which contributes to this debate of the appropriate level of j-units

20.5 Testing assumptions

Q: How do we test these assumptions?

A: Often with plots, like with classic OLS. for example we can pot a histogram of the residuals and see if they are normally distributed.

21 Model Representations

This handout walks through the mathematical representation of two core models: the random intercept model and the random slope model. The goal is to very carefully explain all the different math parts, and show how that translates to a `lmer()` call to R for fitting the model.

21.1 The Scenario

We have a collection of schools that we have randomized into treatment and control conditions. The treatment condition is a novel reading program and the control condition is business as usual. We hope that the treatment accomplishes two things: raising reading level overall, and reducing the gap in reading level between “at-risk” kids and not at risk kids (we assume we have a at-risk status as a dummy variable, measured for all kids and treatment and control prior to treatment).

21.2 The Two-Level Random Intercept Model

We will use the “double-indexing” that is the most common notation for multilevel models (not the Gelman and Hill bracket ($j[i]$) notation). Treatment is at the *school level*: let Z_j be an indicator of whether school j was treated (so a 0/1 variable). Then, for student i in school j we have

$$\begin{aligned} Y_{ij} &= \alpha_j + \beta_1 R_{ij} + \beta_2 X_{ij} + \epsilon_{ij} \\ \alpha_j &= \gamma_0 + \gamma_1 Z_j + \gamma_2 S_j + u_j \end{aligned}$$

with Y_{ij} being the reading level of the student, R_{ij} being a dummy variable of student’s “at risk” status, X_{ij} being an important student demographic variable (e.g., prior reading level), and S_j being a school-level covariate (such as a school quality measure).

This is the two-level model. Level 1 is the first equation with the distribution on the residuals of $\epsilon_{ij} \sim N(0, \sigma^2)$. Level 2 is the second equation with a distribution of random effects of

$$u_j \sim N(0, \sigma_\alpha^2).$$

The σ_α^2 is the variance of the random intercept.

Call the β_{0j} the random intercept and u_j the random effect. The u_j is the residual of the level 2 model,. In R, we would say `coef()` for the intercept (including the mean γ_0) and `ranef()` for the random effect. In math, `coef()` gives $\gamma_0 + u_j$ and `ranef()` gives only u_j . Neither include the γ_1 or γ_2 ; these will be separate columns you get from `coef()`.

21.2.0.0.1 Remarks.

We have *completely pooled* the coefficient for R_{ij} and X_{ij} : we are assuming all the schools have the same relationship between the outcome and these covariates.

The intercept α_j is the expected (predicted average) outcome of a not-at-risk student with $X_{ij} = 0$. Different schools have different means. In particular, treatment schools have a mean of γ_1 more than control; this is the treatment impact.

21.2.1 The Reduced Form Model

If we plug in our 2nd level into the first we get the following:

$$\begin{aligned} Y_{ij} &= \beta_{0j} + \beta_1 R_{ij} + \beta_2 X_{ij} + \epsilon_{ij} \\ &= (\gamma_0 + \gamma_1 Z_j + \gamma_2 S_j + u_j) + \beta_1 R_{ij} + \beta_2 X_{ij} + \epsilon_{ij} \\ &= \gamma_0 + \gamma_1 Z_j + \gamma_2 S_j + \beta_1 R_{ij} + \beta_2 X_{ij} + (u_j + \epsilon_{ij}) \end{aligned}$$

The $u_{0j} + \epsilon_{ij}$ is our total random error. It is how much our prediction of a new, unknown student, would differ from their actual score if we didn't know the school's random effect. The rest of the model is the mean model or structural portion of the model.

This is also called the *reduced form*; it is what econometricians work with. They will write the entire residual as ε_{ij} , however:

$$Y_{ij} = \gamma_0 + \gamma_1 Z_j + \gamma_2 S_j + \beta_1 R_{ij} + \beta_2 X_{ij} + \varepsilon_{ij}$$

21.2.1.0.1 Remarks.

The reduced form helps us see our treatment effect more clearly. It is a shift in outcome of γ_1 for treated students.

The γ_0 is the overall mean reading level for students with $X_{ij} = 0$ for not-at-risk students ($R_{ij} = 0$) in control schools with $S_j = 0$.

Note how we subscript school-level covariates with only a j vs. individual-level covariates get an ij . If you want, you can index everything by ij ; the fact that S_{ij} will then be the same for all students i in school j is hidden in the data. But it does make it look very much like OLS with a weird error term:

$$Y_{ij} = \gamma_0 + \gamma_1 Z_{ij} + \gamma_2 S_{ij} + \beta_1 R_{ij} + \beta_2 X_{ij} + (u_j + \epsilon_{ij})$$

Finally, you might call all the different pieces by different letters to indicate whether you care about them or not. E.g.,

$$Y_{ij} = \mu + \tau Z_{ij} + \beta_1 S_{ij} + \beta_2 R_{ij} + \beta_3 X_{ij} + (u_{0j} + \epsilon_{ij}).$$

Here τ is our treatment effects of interest. The β 's are just adjustments to be ignored. The μ is the grand mean (for those not treated, with $S_{ij} = 0$ and $R_{ij} = 0$ and $X_{ij} = 0$). People often use μ for mean and τ for treatment.

21.2.2 Fitting it in lmer

We fit it as:

```
lmer( Y ~ R + Z + X + S + (1|id), data=dat )
```

21.3 The Two-Level Random Slopes Model

Now let's get very complex to really unpack notational stuff. We are going to let treatment not only impact the average outcome in schools, but also allow treatment to differentially impact students who are "at risk". I.e., we are going to have two treatment impacts, one for not at risk, and one for at risk. This is an interaction of risk status and treatment.

Furthermore, we are going to let different schools have different gaps between at risk and not at risk, but allowing a random effect for the at risk coefficient.

Using our same variables as above, we have, for student i in school j

$$\begin{aligned} Y_{ij} &= \beta_{0j} + \beta_{1j} R_{ij} + \beta_2 X_{ij} + \epsilon_{ij} \\ \beta_{0j} &= \gamma_{00} + \gamma_{01} Z_j + \gamma_{02} S_j + u_{0j} \\ \beta_{1j} &= \gamma_{10} + \gamma_{11} Z_j + u_{1j}. \end{aligned}$$

This is the two-level model. Level 1 is the first equation with the distribution on the residuals of $\epsilon_{ij} \sim N(0, \sigma^2)$. Level 2 are the second and third equations, and the distribution of random effects of

$$\begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} \sim N \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \tau_{00} & \tau_{10} \\ \tau_{10} & \tau_{11} \end{pmatrix} \right]$$

The τ_{00} is the variance of the random intercept. τ_{11} is the variance of the random slope. τ_{10} is the covariance (not correlation) of the random effects. To get the *correlation* of random effects we have $\rho = \tau_{10}/\sqrt{\tau_{00}}\sqrt{\tau_{11}}$. (Note that $\tau_{10} = \tau_{01}$, meaning the covariance of A and B is the same as covariance of B and A, so we just write one of them.)

Call the β_{0j} the random intercept and β_{1j} a random coefficient. We might call them both random coefficients. Call the u_{0j}, u_{1j} , which are the residuals of the level 2 models, the random effects. In R, we would say `coef()` for the coefficients (including the means) and `ranef()` for the random effects.

21.3.1 Remarks.

We have *completely pooled* the coefficient for X_{ij} : we are assuming all the schools have the same relationship between the outcome and X_{ij} . This is why we have no level 2 equation for β_2 and we do not index β_2 as β_{2j} .

The intercept β_{0j} is the expected (predicted average) outcome of a not-at-risk student with $X_{ij} = 0$. Different schools have different means.

The achievement gap of at-risk and not-at-risk students for control schools is measured by γ_{10} . For treatment schools it is $\gamma_{10} + \gamma_{11}$.

The γ_{01} is the average treatment effect for not-at-risk students. Then $\gamma_{01} + \gamma_{11}$ is the average treatment effect for the at-risk students. If we find $\gamma_{11} \neq 0$ then the average effects differ for our two types of students, and the change in the achievement gap induced by treatment is measured by γ_{11} .

In this model, the school-level covariate explains overall differences in reading between schools, but does not relate to the size of treatment impact in a school, or relate to the at-risk vs. not-at-risk achievement gap.

21.3.2 The level 2 covariate matrix.

Sometimes people like to write the correlation matrix using other parameterizations. E.g., we might see

$$\begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} \sim N \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_\alpha^2 & \rho\sigma_\alpha\sigma_R \\ \rho\sigma_\alpha\sigma_R & \sigma_R^2 \end{pmatrix} \right]$$

to indicate the cross-school variation in the intercept (α) and the risk gap (R). Now we specifically have written our correlation of random effects as ρ .

21.3.3 The Reduced Form Model

If we plug in our 2nd level into the first we have to plug in both equations. If we do we get... a mess:

$$\begin{aligned}
Y_{ij} &= \beta_{0j} + \beta_{1j}R_{ij} + \beta_2X_{ij} + \epsilon_{ij} \\
&= (\gamma_{00} + \gamma_{01}Z_j + \gamma_{02}S_j + u_{0j}) + (\gamma_{10} + \gamma_{11}Z_j + u_{1j})R_{ij} + \beta_2X_{ij} + \epsilon_{ij} \\
&= \gamma_{00} + \gamma_{01}Z_j + \gamma_{02}S_j + u_{0j} + \gamma_{10}R_{ij} + \gamma_{11}Z_jR_{ij} + u_{1j}R_{ij} + \beta_2X_{ij} + \epsilon_{ij} \\
&= \gamma_{00} + \gamma_{01}Z_j + \gamma_{02}S_j + \gamma_{10}R_{ij} + \gamma_{11}Z_jR_{ij} + \beta_2X_{ij} + u_{0j} + u_{1j}R_{ij} + \epsilon_{ij} \\
&= \gamma_{00} + (\gamma_{01} + \gamma_{11}R_{ij})Z_j + \gamma_{02}S_j + \gamma_{10}R_{ij} + \beta_2X_{ij} + (u_{0j} + u_{1j}R_{ij} + \epsilon_{ij})
\end{aligned}$$

The $u_{0j} + u_{1j}R_{ij} + \epsilon_{ij}$ is our total random error. It is how much our prediction of a new, unknown student, would differ from their actual score if we didn't know the school's random effect. The rest of the model is the mean model or structural portion of the model.

This is our *reduced form*; it is what econometricians work with. They will write the entire residual as ε_{ij} , however:

$$Y_{ij} = \gamma_{00} + (\gamma_{01} + \gamma_{11}R_{ij})Z_j + \gamma_{02}S_j + \gamma_{10}R_{ij} + \beta_2X_{ij} + \varepsilon_{ij}$$

21.3.3.0.1 Remarks.

The reduced form helps us see our treatment effects and treatment variation across groups more clearly. We can put both terms involving the treatment indicator in parenthesis (final line above) to show how treatment is different by γ_{11} for the at-risk students.

We also see that the difference in treatment effects between at-risk and not at-risk is an *interaction* between student risk and treatment assignment of the school (note the Z_jR_{ij} term).

The γ_{00} is the overall mean reading level for students with $X_{ij} = 0$ for not-at-risk students in control schools. The γ_{10} is the average difference between at-risk and not-at-risk students in control schools, across all schools.

We can rearrange our equations above to get

$$Y_{ij} = (\gamma_{00} + u_{0j}) + (\gamma_{01} + \gamma_{11}R_{ij} + u_{1j})Z_j + \gamma_{02}S_j + \gamma_{10}R_{ij} + \beta_2X_{ij} + \epsilon_{ij}.$$

This shows the random intercept and random slope all bundled up.

As with the intercept model, you might call all the different pieces by different letters to indicate whether you care about them or not. E.g.,

$$Y_{ij} = \mu + \tau Z_{ij} + \beta_1 S_{ij} + \beta_2 R_{ij} + \beta_3 X_{ij} + \tau_R Z_{ij}R_{ij} + (u_{0j} + u_{1j}R_{ij} + \epsilon_{ij}).$$

Here τ and τ_R are our treatment effects of interest. The β 's are just adjustments to be ignored. The μ is the grand mean. This model is the same as above, we are just changing names around.

21.3.4 Fitting it in lmer

We fit it as:

```
lmer( Y ~ R * Z + X + S + (R|id), data=dat )
```

Two other ways of saying the same thing:

```
lmer( Y ~ 1 + R * Z + X + S + (1 + R|id), data=dat )
```

and

```
lmer( Y ~ 1 + Z + S + R + Z:R + X + (1 + R|id), data=dat )
```

21.3.5 Remarks.

See how the reduced form and `lmer()` align, especially if we write out what `R` automatically does with `R * Z` (`R` will expand `R*Z` into `R + Z + R:Z` automatically).

21.3.6 Another form for the two-level model

The above is the “double-subscript” way of writing a model. By contrast, Gelman and Hill index with a nifty “bracket notation.” First, let $j[i]$ indicate the school student i is attending. Then we have:

$$\begin{aligned}Y_i &= \beta_{0j[i]} + \beta_{1j[i]}R_i + \beta_2X_i + \epsilon_i \\ \beta_{0j} &= \gamma_{00} + \gamma_{01}Z_j + \gamma_{02}S_j + u_{0j} \\ \beta_{1j} &= \gamma_{10} + \gamma_{11}Z_j + u_{1j},\end{aligned}$$

This is basically identical to the above, but if you are not familiar with the bracketing then things can get messy.

The advantage of this is we can then imagine each student gets their own unique id, i , and then we can query where that student is via $j[i]$. This can be useful when looking at crossed effects models, where units have different random effects for different things (e.g., for a test we might have observation k corresponding to a single answer for a test question, with $i[k]$ being the student who answered it and $q[k]$ being the question item).

22 Connecting the three dots: An HSB Model

This handout shows (1) a mathematical model, (2) the `lmer` syntax for that model, and (3) the output for that model, for the model discussed in Lecture 2.4. This handout is designed to help translate between these three different worlds.

22.1 The mathematical model

Level 1 models:

$$\begin{aligned}y_{ij} &= \beta_{0j} + \beta_{1j}ses_{ij} + \beta_2female_{ij} + \epsilon_{ij} \\ \epsilon_{ij} &\sim N(0, \sigma_y^2)\end{aligned}$$

Level 2 models:

$$\begin{aligned}\beta_{0j} &= \gamma_{00} + \gamma_{01}sector_j + \gamma_{02}meanSES_j + u_{0j} \\ \beta_{1j} &= \gamma_{10} + \gamma_{11}sector_j + u_{1j}\end{aligned}$$

with

$$\begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} \sim N \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \tau_{00} & \tau_{01} \\ \tau_{01} & \tau_{11} \end{pmatrix} \right].$$

The τ_{01} is the covariance of the random intercept and random slope. We usually look at the correlation of

$$\rho = \frac{\tau_{01}}{\sqrt{\tau_{00}\tau_{11}}}.$$

The estimated ρ is what R gives us in the printed output, rather than τ_{01} .

The derivation of the reduced form is:

$$\begin{aligned}y_{ij} &= \beta_{0j} + \beta_{1j}ses_{ij} + \epsilon_{ij} \\ &= (\gamma_{00} + \gamma_{01}sector_j + \gamma_{02}meanSES_j + u_{0j}) + (\gamma_{10} + \gamma_{11}sector_j + u_{1j})ses_{ij} + \beta_2female_{ij} + \epsilon_{ij} \\ &= \gamma_{00} + \gamma_{01}sector_j + \gamma_{02}meanSES_j + u_{0j} + \gamma_{10}ses_{ij} + \gamma_{11}sector_jses_{ij} + u_{1j}ses_{ij} + \beta_2female_{ij} + \epsilon_{ij} \\ &= \gamma_{00} + \gamma_{01}sector_j + \gamma_{02}meanSES_j + \gamma_{10}ses_{ij} + \gamma_{11}sector_jses_{ij} + \beta_2female_{ij} + (u_{0j} + u_{1j}ses_{ij} + \epsilon_{ij})\end{aligned}$$

This formula is what we will give to `lmer()` in R's formula notation.

22.2 The lmer code

```
M1 = lmer( mathach ~ 1 + female + ses*sector +
           meanses + (1+ses|id),
           data = dat )
```

This code is the exact same model, using the fact that `ses*sector` means `ses + sector + ses:sector`. I.e., the above is exactly the same as this more explicitly written R code:

```
M1 = lmer( mathach ~ 1 + sector + meanses + ses + sector:ses + female + (1+ses|id),
           data = dat )
```

Each term in the expanded formula corresponds to a math symbol in the mathematical model. The `(1+ses|id)` make our random effects, and tie to all the τ terms. The residual variance σ_y^2 is the only parameter not explicitly listed in the above model.

22.3 The output

```
display( M1 )
```

```
lmer(formula = mathach ~ 1 + sector + meanses + ses + sector:ses +
      female + (1 + ses | id), data = dat)
      coef.est coef.se
(Intercept) 12.79     0.21
sector       1.29     0.29
meanses      3.04     0.37
ses          2.73     0.14
female      -1.18     0.16
sector:ses   -1.31     0.21

Error terms:
Groups    Name        Std.Dev. Corr
id        (Intercept) 1.45
          ses         0.18     0.65
Residual             6.05
---
number of obs: 7185, groups: id, 160
AIC = 46482.9, DIC = 46445.1
deviance = 46454.0
```

Now, using this output, we have estimates for all our mathematical modeling parameters:

- $\gamma_{00} = 12.79$ - The overall average math achievement for a student with 0 ses in a public school with 0 mean SES.
- $\gamma_{01} = 1.29$ - The average difference between otherwise equivalent catholic and public schools.
- $\gamma_{02} = 3.04$ - The impact on average achievement due to mean SES of schools. Higher SES schools have higher achievement.
- $\gamma_{10} = 2.73$ - The average slope of ses vs. math achievement in public schools.
- $\beta_2 = -1.18$ - The gender gap; girls have lower math scores on average.
- $\gamma_{11} = -1.31$ - The difference in slope between public and catholic schools (catholic schools have flatter slopes).
- $\tau_{00} = 1.45^2$ - Variation in overall intercept of schools (within category of public or catholic, and beyond mean SES).
- $\tau_{11} = 0.18^2$ - The variation in the random slopes for ses vs. math achievement.
- $\rho = 0.65$ - The random intercepts are correlated with random slopes. High achievement schools have more discrepancy between low and high ses students.
- $\sigma_y = 6.05$ - The unexplained student variation within school.

23 Predictors in Longitudinal Growth Models

23.1 Tips for growth models

Start with an unconditional growth model, i.e., don't include any level-1 or level-2 predictors. This model provides useful empirical evidence for determining a proper specification of the individual growth equation and baseline statistics for evaluating more complicated level-2 models.

The nature of the predictor in longitudinal analysis determines where it gets added to the model: Time-invariant predictors always go in level-2 (subject level) model Time-varying predictors can go in level-1 and/or level-2. The level of the predictor dictates which variance component it seeks to describe: Level-2 describes level-2 variances and Level-1 describes level-1 variances. Although the order in which you add these predictors (in a series of successive models) may not ultimately matter, general practice is to add level-2 (time-invariant) predictors first.

How to decide where to add predictors? One strategy:

1. First fit an unconditional (i.e. no predictors) random intercept model. This isn't really predictive, but we can use it as a baseline model that partitions variance into between and within-person variances. Singer & Willett (2003) call this the “unconditional means model”.
2. Calculate the ICC
 1. If most of the variance is between-persons in the random intercept (level-2), then you'll use person-level predictors to reduce that variance (i.e., account for inter-person differences)
 2. If most of the variance is within-person (level-1 residual variance), you'll need time-level predictors to reduce that variance (i.e. account for intra-person differences)

Because the time-specific subscript t can only appear in the level-1 model, all time-varying predictors must appear in the level-1 individual growth model. That is, person-specific predictors that vary over time appear at level-1, not level-2. Time-invariant predictors go in level-2. Furthermore, because they are time-invariant, this means they have no within-person variation to allow for a level-2 residual; thus, the level-2 growth rate parameter corresponding to this time-invariant predictor will not have an error term (i.e. it's assumed to be zero). Interpretation

wise, this assumes the effect of a person-specific effect is constant across population members. For a time-varying predictor, however, the associated level-2 growth parameter equation would have a residual term. This allows the effect of the time-varying predictor to vary randomly across the individuals in the population.

With only a few measurement points per person, we often lack sufficient data to estimate many variance components. Thus, it's suggested that we resist the temptation to automatically allow the effects of time-varying predictors to vary at level-2 unless you have a good reason, and enough data, to do so.

So far in class, we've seen person-specific variables appear in level-2 submodels as predictors for level-1 growth parameters. You might therefore think that substantive predictors must always appear at level-2, but this isn't true!

How inclusion of predictors affect variance components: Generally, when we include time-invariant predictors:

1. the level-1 variance component, σ_e^2 , remains pretty stable because time-invariant predictors can't explain any within-person variation
2. the level-2 variance components, τ_{00} and τ_{01} , will decrease if the time-invariant predictors explain some of the between-person variation in initial status or rates of change, respectively.

When we include time-varying predictors:

1. both level-1 and level-2 variance components might be affected because time-varying predictors vary both within a person and between people
2. we can interpret the resulting decrease in the level-1 variance component as amount of variation in the outcome explained by the time-varying predictors; however, it isn't meaningful to interpret subsequent changes in level-2 variance components because adding the time-varying predictor changes the meaning of the individual growth parameters, which consequently alters the meaning of the level-2 variances, so it doesn't make sense to compare the magnitude of these level-2 variances across successive models.

23.2 Additional Resources

- <https://books.google.com/books?id=PpnA1M8VwR8C&pg=PA168&lpg=PA168&dq=longitudinal+data+analysis+with+mlm+and+latent+growth+models+for+stata+second+edition&hl=en&sa=X&ved=0CCwQ6AEwAWoVChMI5ZLsjKDjyAIVzB0-Ch1s6wGV#v=onepage&q=&f=false>
- http://jonathanTemplin.com/files/mlm/mlm12uga/mlm12uga_section06.pdf
- http://www.lesahoffman.com/944/944_Lecture07_Time-Invariant.pdf

24 Interpreting GLMs

24.1 Poisson regression models

Poisson regression is sometimes used to model count data. The canonical form of a Poisson (log-linear) regression model is

$$\log(E[Y|X]) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p$$

$$Y \sim \text{Poisson}(E[Y|X])$$

The Poisson distribution has only one parameter, the mean, which is also the variance of the distribution. So in estimating $E[Y|X]$, we are also estimating $\text{Var}(Y|X)$. This is a potential drawback to the Poisson model, because there is no variance parameter to estimate, and so incorrect models can give wildly inaccurate standard errors (frequently unrealistically small). A better model is a quasi-Poisson model, for which the variance is proportional to the mean, but not necessarily equal to it. The negative binomial regression model is also commonly used to address over-dispersed count data where the variance exceeds the mean.

The canonical link function for Poisson outcomes is the natural logarithm. When we use a log-link, we can write

$$E[Y|X] = e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}.$$

We can interpret β_0 as follows: for observations which are 0 on all of the predictors, we estimate that the mean (expected) value of the outcome will be e^{β_0} .

We can interpret β_1 as follows: adjusting for the other predictors, a one-unit difference in X_1 predicts a $(e^{\beta_1} - 1) \times 100\%$ difference in the outcome.

Generally, when using a log-link, we assume that differences in the predictors are associated with multiplicative differences in the outcome.

Some advantages to using an exponential link are

1. the model is mathematically more tractable and simpler to fit
2. the model parameters are easy to interpret

3. the mean of Y is guaranteed to be positive for all values of X , which is required by the Poisson distribution

We can fit a Poisson log-linear regression by writing

```
glm(Y ~ X, family = poisson(link = 'log'))
```

To fit a quasi-Poisson model, write

```
glm(Y ~ X, family = quasipoisson(link = 'log'))
```

To fit a negative binomial regression model, write (after loading the MASS library)

```
glm.nb(Y ~ X, link='log')
```

To fit a Poisson regression with an identity link (where coefficients are interpreted as expected differences in the outcome associated with unit differences in the predictor), write

```
glm(Y ~ X, family = poisson(link = 'identity'))
```

To fit a Poisson regression with a square root link, which is vaguely like a compromise between an identity link and a log link (and is harder to interpret than either), write

```
glm(Y ~ X, family = poisson(link = 'sqrt'))
```

To fit a Poisson log-linear model with a random intercept and slope, write

```
glmer(Y ~ X + (X|grp), family = poisson(link = 'log'))
```

24.2 Dichotomous regression models

When predicting either successes and failures, or proportions, we can use a model with a binomial outcome. Here we'll focus on models where the data is represented as individual successes and failures. The canonical model for these data is logistic regression, where

$$\text{logit}(E[Y|X]) \equiv \log \left(\frac{P(Y = 1|X)}{1 - P(Y = 1|X)} \right) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p$$

$$Y \sim \text{Binomial}(1, E[Y|X])$$

We can rewrite this model as

$$\text{odds}(Y) = \frac{P(Y = 1|X)}{1 - P(Y = 1|X)} = e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}$$

or

$$P(Y = 1|X) = \frac{e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}$$

We can interpret β_0 as follows: for observations which are 0 on all of the predictors, we estimate that the mean value of the outcome will be $\frac{e^{\beta_0}}{1+e^{\beta_0}}$. That is, we estimate that the probability of the outcome being a ‘success’ (assuming ‘success’ is coded as a 1) will be $\frac{e^{\beta_0}}{1+e^{\beta_0}}$.

We can interpret β_1 as follows: adjusting for the other predictors, a one-unit difference in X_1 predicts a β_1 difference in the log-odds of the outcome being one, or a $(e^{\beta_1} - 1) \times 100\%$ difference in the odds of the outcome. Unfortunately, the change in probability of a unit change depends on where the starting point is, so there is no easy way to interpret these coefficients in terms of direct probability. One can calculate the estimated change for specific units, however, and look at the distribution of those changes.

Other possible link functions include the probit (which uses a Normal CDF to link $\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p$ to $P(Y = 1|X)$), or the complementary log-log (which allows $P(Y = 1|X)$ to be asymmetric in the predictors), among others.

24.2.1 Example

24.2.2 How to fit a GLM

We can fit a logistic regression model by writing

```
glm(Y ~ X, family = binomial(link = 'logit'))
```

We can fit a probit regression model by writing

```
glm(Y ~ X, family = binomial(link = 'probit'))
```

We can fit a complementary log-log model by writing

```
glm(Y ~ X, family = binomial(link = 'cloglog'))
```

We can allow a random slope and intercept by writing

```
glmer(Y ~ X + (X|grp), family = binomial(link = 'logit'))
```

25 GLMs vs. Transformations

Those of you coming from S40 and S52 may recall that when we have non-linear relationships between X and Y, we can apply a transformation, such as taking the log, to linearize the relationship. In the words of Jimmy Kim, “with transformations, we use the *machinery of linear regression to model non-linear relationships.*” If that’s the case, then what is Poisson regression about, which deals with log counts? This is a topic that confused me for many years so hopefully I can clear it up here.

25.1 Making and Graphing the Data

Let’s start by making some fake data. Here’s the data-generating function, which has the relationship that a 1-unit increase in x will increase the expected count by $e^{.5} = 1.65$.

$$y = \text{Poisson}(e^{0.5x})$$

```
library(tidyverse)
library(sjPlot)
library(ggeffects)

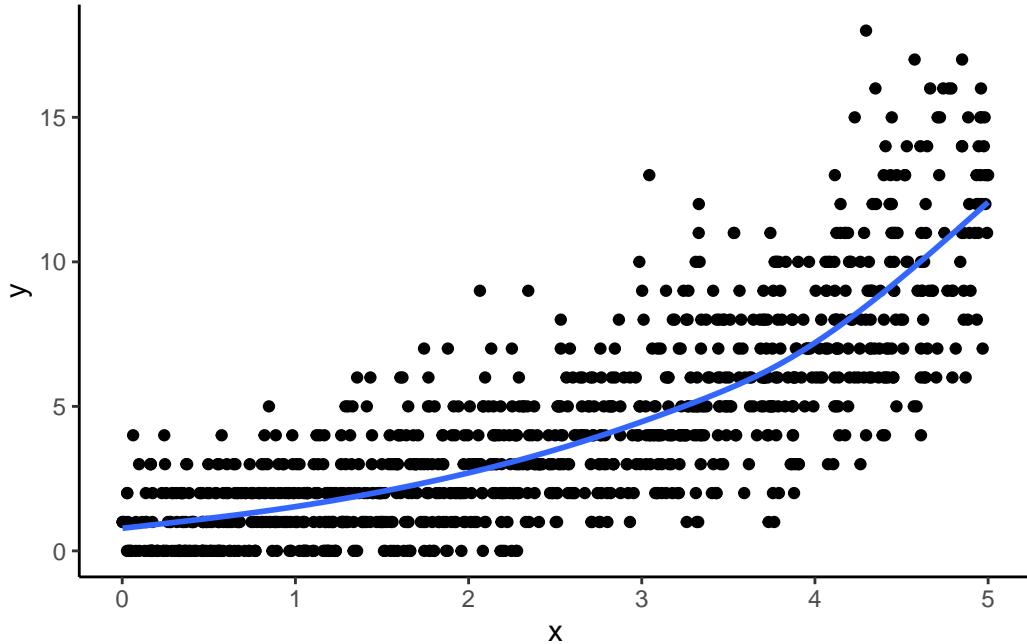
theme_set(theme_classic())

rm(list = ls())

dat <- tibble(
  x = runif(1000, 0, 5),
  y = rpois(1000, exp(0.5*x))
)
```

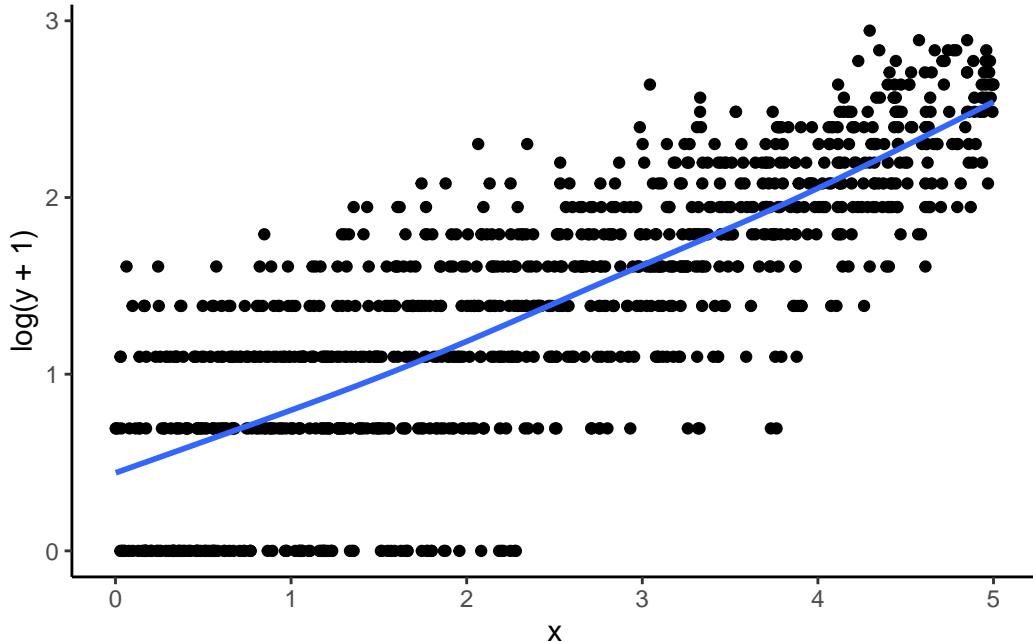
In the graph, we can see that the relationship between x and y is clearly non linear!

```
ggplot(dat, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(se = FALSE)
```



Let's plot $\log_y + 1$ on x . Amazing! The relationship is basically linear, which suggests that a 1-unit increase in x has some multiplicative effect on y .

```
ggplot(dat, aes(x = x, y = log(y + 1))) +  
  geom_point() +  
  geom_smooth(se = FALSE)
```



25.2 Fitting the Regression Models

Let's use both OLS and Poisson regression to fit the data. We see a few things:

1. The Poisson model fits drastically better, both in terms of R^2 and that the coefficients are close to the data-generating values
2. The transformed OLS model understates the slope
3. Both models have (seemingly) similar interpretations: a 1-unit increase in x causes an e^β increase in y . How is this possible?

So what's going on?

The answer is that there is a very subtle difference between a transformed OLS regression and a Poisson regression. In transformed OLS, we are modeling the mean of the log of Y , or $E(\ln(y|x))$. In Poisson, we're modeling the log of the mean of Y , or $\ln(E(y|x))$. These are not equivalent! In essence, Poisson regression is a model for the arithmetic mean, whereas OLS is a model for the geometric mean. This means that when we exponentiate the Poisson model, we can get predicted counts, but this is *not* true of the OLS model.

```
m1 <- lm(log(y + 1) ~ x, dat)
m2 <- glm(y ~ x, dat, family = poisson)
```

```
tab_model(m1, m2,
          p.style = "stars",
          show.ci = FALSE,
          show.se = TRUE,
          digits = 3,
          transform = NULL,
          dv.labels = c("Log(Y+1)", "Poisson"))
```

	Log(Y+1)		Poisson		
Predictors	Estimates	std. Error	Log-Mean	std. Error	
(Intercept)	0.375 ***	0.029	-0.074	0.045	
x	0.420 ***	0.010	0.516 ***	0.012	
Observations	1000		1000		
R ² / R ² adjusted	0.631		0.906		
	/				
	0.631				
	*				
p<0.05	**				
p<0.01	***				
p<0.001					

25.3 More Intuition: An Example with Means

Let's create a super simple data set, s.

```
s <- c(1, 10, 100)
```

It's clearly skewed. But I can still take the mean. I could take the arithmetic mean, or the geometric mean. These are clearly different quantities.

```
mean(s) # arithmetic
```

```
[1] 37
```

```
exp(mean(log(s))) # geometric
```

```
[1] 10
```

The idea of Poisson is to take the log of the mean and fit a linear model for that:

```
log_mean <- log(mean(s))  
log_mean
```

```
[1] 3.610918
```

The idea of transformed OLS is to take the mean of the log and fit a linear model for that:

```
mean_log <- mean(log(s))  
mean_log
```

```
[1] 2.302585
```

When I exponentiate the log of the mean, I get back the original arithmetic mean. This is what Poisson is doing:

```
exp(log_mean)
```

```
[1] 37
```

When I exponentiate the mean of the log, I get back the original geometric mean. This is what transformed OLS is doing:

```
exp(mean_log)
```

```
[1] 10
```

25.4 Further Reading

<https://www.theanalysisfactor.com/the-difference-between-link-functions-and-data-transformations/>

26 Likelihood Ratio Tests

26.1 R Setup

```
# load libraries
library(tidyverse)
library(lme4)
library(haven)
library(sjPlot)

# clear memory
rm(list = ls())

select <- dplyr::select

# load HSB data
hsb <- read_dta("data/hsb.dta") |>
  select(mathach, ses, schoolid)
```

26.2 Why LR Tests?

Our fixed effects coefficients have SEs, z-statistics, and p-values, which allow us to easily test the null hypothesis that the slopes are 0 in the population. No such quantities, however, are provided for the random effects of our model. We can use LR tests to address this issue and test the statistical significance of the various random portions of our model.

We can also use LR tests on fixed effects or sets of fixed effects (like a nested F-test in OLS), but 99.9% of the time, the conclusion will be the same as using the z-statistics.

LR tests require that the models are *nested*, meaning that they use the same data, and one model can be expressed as a constrained version of the other.

26.3 HSB Example

We fit 3 models:

1. Random intercept model
2. Random slope model
3. Random slope model with no correlation between intercepts and slopes

We can see from the model output that the point estimates for the random slope variance τ_{11} and the correlation ρ_{01} are non-zero, but how can we get p-values for these quantities?

```
m1 <- lmer(mathach ~ ses + (1|schoolid), hsb)
m2 <- lmer(mathach ~ ses + (ses|schoolid), hsb)
m3 <- lmer(mathach ~ ses + (ses||schoolid), hsb)

tab_model(m1, m2, m3,
          p.style = "stars",
          show.se = TRUE,
          show.ci = FALSE,
          dv.labels = c("RI", "RS", "No Rho"))
```

RI
RS
No Rho
Predictors
Estimates
std. Error
Estimates
std. Error
Estimates
std. Error
(Intercept)
12.66 ***
0.19

12.67 ***

0.19

12.65 ***

0.19

ses

2.39 ***

0.11

2.39 ***

0.12

2.40 ***

0.12

Random Effects

2

37.03

36.83

36.82

00

4.77 schoolid

4.83 schoolid

4.85 schoolid

11

0.41 schoolid.ses

0.42 schoolid.ses

01

-0.11 schoolid

ICC

```

0.11
0.12
0.12
N
160 schoolid
160 schoolid
160 schoolid
Observations
7185
7185
7185
Marginal R2 / Conditional R2
0.077 / 0.182
0.077 / 0.189
0.077 / 0.185
* p<0.05 ** p<0.01 *** p<0.001

```

26.3.1 Are random ses slopes necessary?

We use `anova` to perform the LR test comparing `m1` and `m2`, and we see that the random slopes are not statistically significant.

```
anova(m1, m2)
```

```

Data: hsb
Models:
m1: mathach ~ ses + (1 | schoolid)
m2: mathach ~ ses + (ses | schoolid)
      npar   AIC   BIC logLik deviance Chisq Df Pr(>Chisq)
m1     4 46649 46677 -23320     46641
m2     6 46648 46690 -23318     46636 4.5354  2     0.1035

```

26.3.2 Is there a correlation between the random intercept and slope for ses?

Similarly, we see that the correlation is non-significant

```
anova(m2, m3)
```

```
Data: hsb  
Models:  
m3: mathach ~ ses + ((1 | schoolid) + (0 + ses | schoolid))  
m2: mathach ~ ses + (ses | schoolid)  
      npar   AIC   BIC logLik deviance Chisq Df Pr(>Chisq)  
m3     5 46647 46681 -23318    46637  
m2     6 46648 46690 -23318    46636 0.2762  1      0.5992
```

26.4 Technical Notes

TL/DR: The traditional LR test provided by `anova` is likely to be conservative for testing the significance of variance components. For the purposes of this course, it is fine.

There is a lot of multilevel literature arguing that testing a null hypothesis on variance components with LR tests is not the best approach. The reason is that variances cannot be negative, so the null hypothesis exists on the “boundary of the parameter space” and therefore is “likely to be conservative” (to use the warning that Stata gives you, i.e., the p-values are too high). The true distribution of a 0 variance component is not a normal distribution, but a mixture distribution with half of the probability mass at 0 and the other half χ^2 . When you’re testing the significance of the random intercepts model, you can divide the p-value by 2 to get the right answer (Stata default), though for more complex models it’s not so simple. There are some R packages that use simulation-based approaches to provide more robust results such as `pbkrtest::PBmodcomp`, but we won’t go into them here. See RH&S pp. 88-89 for a more thorough discussion of this issue. Despite this, the standard LR test remains common in practice.

27 AIC, BIC, and Deviance

In this section, we briefly walk through how to find AIC, BIC, and Deviance to compare models. We have a simple multilevel dataset (we generate through a utility package, `blkvar`, that Miratrix and the C.A.R.E.S. lab has used to explore how multilevel modeling works in practice), and generate a few variables that we will use as predictors. Only the fourth variable is actually useful for prediction! Let's see if our AIC, etc., measures identify which model is superior.

To install a “working package” we use `devtools`:

```
devtools::install_github("https://github.com/lmiratrix/blkvar" )
```

```
library( blkvar )
dd = generate_multilevel_data( J = 40 )
head( dd )
```

	sid	Y0	Y1	Z	Yobs	W
1	1	2.903243	2.1125852	1	2.112585	-1.008438
2	1	1.704167	0.9135087	0	1.704167	-1.008438
3	1	1.358634	0.5679754	0	1.358634	-1.008438
4	1	2.104877	1.3142190	0	2.104877	-1.008438
5	1	1.404675	0.6140171	0	1.404675	-1.008438
6	1	2.610360	1.8197016	1	1.819702	-1.008438

```
dd$X1 = rnorm( nrow(dd) )
dd$X2 = rnorm( nrow(dd) )
dd$X3 = rnorm( nrow(dd) )
dd$X4 = dd$Yobs + rnorm(nrow(dd))

M1 = lmer( Yobs ~ 1 + (1|sid), data=dd )
M2 = lmer( Yobs ~ 1 + X1 + X2 + X3 + (1|sid), data=dd )
M3 = lmer( Yobs ~ 1 + X1 + X2 + X3 + X4 + (1|sid), data=dd )

library( arm )
```

```
Loading required package: MASS
```

```
Attaching package: 'MASS'
```

```
The following object is masked from 'package:dplyr':
```

```
select
```

```
arm (Version 1.13-1, built: 2022-8-25)
```

```
Working directory is /Users/jog1638/Library/CloudStorage/Dropbox/HGSE/HGSE PhD/2023F/S043/MLI
```

```
display(M1)
```

```
lmer(formula = Yobs ~ 1 + (1 | sid), data = dd)
coef.est  coef.se
  0.20      0.14
```

```
Error terms:
```

Groups	Name	Std.Dev.
sid	(Intercept)	0.84
Residual		0.61

```
---
number of obs: 422, groups: sid, 40
AIC = 906.6, DIC = 896.3
deviance = 898.5
```

```
display(M2)
```

```
lmer(formula = Yobs ~ 1 + X1 + X2 + X3 + (1 | sid), data = dd)
coef.est  coef.se
(Intercept)  0.20      0.14
X1          -0.04      0.03
X2           0.04      0.03
X3           -0.01      0.03
```

```
Error terms:
```

Groups	Name	Std.Dev.
--------	------	----------

```

    sid      (Intercept) 0.84
    Residual           0.61
    ---
    number of obs: 422, groups: sid, 40
    AIC = 924.3, DIC = 877.3
    deviance = 894.8

library( texreg )

Version: 1.38.6
Date: 2022-04-06
Author: Philip Leifeld (University of Essex)

Consider submitting praise using the praise or praise_interactive functions.
Please cite the JSS article in your publications -- see citation("texreg").

screenreg( list( M1, M2, M3 ) )

=====

              Model 1   Model 2   Model 3
-----
(Intercept)      0.20      0.20      0.11
                  (0.14)    (0.14)    (0.09)
X1                -0.04     -0.02
                  (0.03)    (0.03)
X2                0.04      0.03
                  (0.03)    (0.03)
X3                -0.01     -0.01
                  (0.03)    (0.02)
X4                  0.34 *** 
                  (0.02)
-----
AIC            906.64    924.33    748.94
BIC            918.78    948.60    777.25
Log Likelihood -450.32   -456.16   -367.47
Num. obs.        422       422       422
Num. groups: sid 40         40         40
Var: sid (Intercept) 0.71      0.71      0.29
Var: Residual     0.37      0.37      0.25
=====

*** p < 0.001; ** p < 0.01; * p < 0.05

```

28 Optimization Algorithms for MLMs

28.1 Convergence and optimization algorithms

Unlike OLS, which has a simple closed-form solution for parameter estimates, multi-level models are complex and often do not have closed-form solutions.¹ As a result, programming languages use optimization algorithms to fit models. These optimization algorithms are typically iterative processes that repeatedly test potential values and eventually converge to the model estimates.

Typically, optimization algorithms involve approximating the log-likelihood function as a multivariate quadratic function. Sometimes this approximation is easy to find and closely matches the true log-likelihood; in these cases, convergence occurs quickly. However, we've seen that convergence is trickier when the log-likelihood function is flat near the maximum; it's also trickier with more complex and fragile likelihoods, like those created by the link functions from Generalized Least Squares (GLS) models.

28.2 What to do when your model won't converge

If your error won't converge, you might get a warning message like this:

```
Warning message: In checkConv(attr(opt, "derivs"), optpar, ctrl = controlcheckConv, : Model failed to converge with max|grad| = 0.0463355 (tol = 0.001, component 1)
```

This warning message tells us two things. First, remember that we are trying to find the maximum of the likelihood function, or the place where the `slope = 0`. In the warning, the `tol = 0.001` tells us that R will be happy if it finds estimates where the `slope ≤ 0.001`. It's also saying that our slope when R stopped converging was `0.0463355`.

Steps that you can take to resolve:

1. Try rescaling variables and refitting your model
2. Try changing your optimizer settings

¹“Closed form” means that there is a formula you can use to simply and directly calculate your estimates. For example, in OLS your matrix equation for $\hat{\beta} = (X'X)^{-1}X'Y$

To address items #2 and #3, you add a `Control` option into your `lme`, `lmer`, or `glmer` function. Each of those functions has its own option, but they all take the same arguments:

1. `lme`: `lmeControl()`
2. `lmer`: `lmerControl()`
3. `glmer`: `glmerControl()`

Below are some other optimizer options that you can try. For simplicity, we're specifying them all as “`glmer`” options, but you could easily adjust them to match whichever model you are trying (but failing) to fit:

```
## Use a Nelder-Mead optimizer
log_mod <- glmer(pass ~ (gender + frl_new + f3) +
  (gender + frl_new + f3|sch),
  data = wide_dat, family = binomial(),
  control = glmerControl(optimizer = 'Nelder_Mead'))


## Use a BFGS optimizer
log_mod <- glmer(pass ~ (gender + frl_new + f3) +
  (gender + frl_new + f3|sch),
  data = wide_dat, family = binomial(),
  control = glmerControl(optimizer="optim", optimMethod = "BFGS"))


#If these aren't working, you can download a special package to use the optimx optimizer
#install.packages('optimx')
library(optimx)
log_mod <- glmer(pass ~ (gender + frl_new + f3) +
  (gender + frl_new + f3|sch),
  data = wide_dat, family = binomial(),
  glmerControl(optimizer = 'optimx', calc.derivs = FALSE,
    optCtrl = list(method = "L-BFGS-B",
      starttests = FALSE,
      kkt = FALSE)))
```

Aside from these examples, there are many other ways to adjust your optimization commands, which can be found here: <https://rdrr.io/cran/lme4/man/lmerControl.html>

28.3 Technical Appendix: Understanding the Types of Optimization Algorithms

There are generally four “types” of algorithms employed to find MLE/REML solutions:

1. Newton methods
2. Quasi-Newton methods
3. EM algorithm
4. Other

28.4 Newton Methods

Newton's method is the most "pure" of these approaches; essentially Newton's method uses a Taylor series approximation to approximate a quadratic function and find its maxima. It involves finding the Hessian (a matrix containing all the second and partial derivatives from your likelihood). An advantage of this approach is that it is theoretically the best of the three named approaches because it will often require fewer iterations to converge. However, there are two drawbacks:

1. When there are a large number of parameters, it is time-consuming to analytically calculate or numerically approximate all second order and mixed derivatives needed for the Hessian matrix.
2. In regions where the log-likelihood function is not sufficiently concave down, there is a tendency to dramatically overshoot because the step size to the next point is proportional to the inverse of the second derivative, resulting in pathological oscillations that would amplify if allowed to continue. Thus, where the log-likelihood function is not well approximated by a second order Taylor expansion, the method tends to fail miserably. This would be the case, for example, if the log-likelihood function was a standard normal density and you started out 2 SD from the mean.

28.5 Quasi-Newton Methods

Quasi-Newton methods start with a "guess" for the Hessian, apply the quadratic formula to attain a new point, update the guess of the Hessian, and repeat until convergence is attained. Importantly, the approximated Hessian will converge to the Hessian so long as the Wolfe conditions (a set of conditions on the likelihood) are satisfied. The easiest guess for the initial Hessian is the identity matrix, making the first step simply a gradient descent. When the identity matrix is used as an initial guess, the quasi-Newton methods converge "super-linearly"—that is it displays linear convergece initially, but approach quadratic convergence as the approximated Hessian updates itself. There are many quasi-Newton methods, but the most common is the "BFGS" updating method.

In terms of time to convergence, quasi-Newton is typically much faster than pure Newton methods. This addresses the first drawback listed for Newton's method, but it is still susceptible

to the second issue. The other potential challenge with Quasi-Newton methods occurs when the Wolfe conditions are not satisfied - the method will typically not converge to the Hessian within a reasonable number of iterations, and can often exceed the maximum iterations set by a program.

28.6 EM (Expectation-Maximization) Algorithm

The EM algorithm is another way of approximating the likelihood function and maximizing that approximation. It does this in a repeating series of steps: the E (Expectation) step and the M (Maximization) step. In random effect models, where normality is assumed, the E-step results in an quadratic function to be maximized in the M-step. Importantly, each iteration of the EM algorithm is guaranteed to increase the likelihood function, a feature that may be too difficult to attain with the Newton methods when a quadratic function is not yet a good approximation. Thus, even if the likelihood function not well approximated by a quadratic function, we are assured to be getting closer to a maximum with the EM algorithm. Thus the EM algorithm fixes the second issue from Newton's method. However, it only displays linear convergence (as opposed to “super linear” or “quadratic”) and can therefore take a very long time to converge.

28.7 Implementation in Different Programs

28.7.1 Stata/MPlus/HLM

Stata, Mplus, and HLM, each use a combination of the EM and the quasi-Newton methods when estimating models with random effects. The algorithms start with the EM algorithm and proceed until there is sufficient concavity to switch a quasi-Newton method. Using a combination of the EM and quasi-Newton methods minimizes computational time while maximizing the opportunity that the algorithm will converge to a maximum. Mplus and HLM will even switch back to the EM algorithm if the Wolfe conditions are not attained in a set amount of time; thus, my experience has been that Mplus and HLM tend to converge the fastest and tend to minimize convergence issues.

Disclaimer: sometimes you may need to manually increase the number of EM iterations allowed to achieve convergence.

28.7.2 R

If I am interpreting the lmerControls documentation correctly, this method starts with the EM algorithm and then applies “unconstrained and box-constrained optimization using PORT

routines” from the `nlminb` function. I’ll classify this algorithm as “other”, as opposed to the three named approaches above.

In my opinion, `lme`’s optimization algorithm is less than ideal for two reasons. First, the number of initial EM steps is fixed and who’s to say that the default number of EM iterations will bring us to a region where the log-likelihood function is sufficiently concave?

Second, HLM and Mplus have been estimating random effect models for a long time, and developers from both have come to the conclusion that the quasi-Newton method as the second method in a combination is the best for these models. I’ll assume this is a very informed decision on the end of these developers. Yet, it does not appear that this is what is occurring in R. Instead, R uses “unconstrained and box-constrained optimization using PORT routines,” whatever that is.

Even according to the “See Also” section in the `nlminb` help file, the `optim` function is listed as preferred over the `nlminb` function. As it turns out, the `optim` function applies the “BFGS” quasi-Newton method as the default, which is consistent with Stata’s approach.

29 Survey Weights

29.1 Multilevel modeling and survey weights

In many circumstances you may be faced with a multilevel modeling project where you also have survey weights. Unfortunately R does not have good support for this hybrid of two worlds (although if you go the econometric direction you can incorporate weights into your robust standard errors).

You basically have two options at this point: you can ignore the weights (defendable, but often upsetting to reviewers and colleagues), or switch to Stata, which allows for both.

29.2 Topline advice

Ignore the weights for your final project and worry about extending to your general population later on, depending on where your research takes you. More important than the weights is making sure you have random effects corresponding to all clustering involved in the way the data were collected. For example, if the program was a sample of states and then a sample of villages, and then households, you would have a 4-level model: states, villages, households, and individuals. You would want a random effect for each level. If you had few states, you could back off and have fixed effects for state and generalize only to the sampled states rather than the full country.

29.3 What are survey weights?

The easy way to think of sample weights (survey weights) is the answer to “how many people does this individual represent in the population?” (Although note that weights will generally be proportional to the answer to this question rather than literally that value.)

For example, if you had three people, the first with a weight of 1, the second with a weight of 0.5 and the third with a weight of 3, then we would think of our population as being $1 + 0.5 + 3 = 4.5$ people, 3 of whom are people similar to our third sampled person, 0.5 of which our second sampled person, and 1 of whom is similar to our first person.

In other words, our first person is sampled proportional to their prevalence in the population. The 0.5 weight person is “oversampled”—we have too many people like this in our sample, as compared to the population so we “downweight” them. The third person is underrepresented, by contrast. We should have had three times as many of these types of people in our sample as we have.

Thus, sampling weights adjust for the probability of selecting an individual from the population when that probability is not constant (this could be due either by design or by chance). For nationally representative data surveys often select a sample where individuals have an unequal probability of being selected. This is done to increase the number of individuals and reduce sampling variability, particularly for certain areas or subgroups of the population. In some cases, corrections for non-response are also built into the weights. Sampling weights are then inversely proportional to the probability of selection.

In some complex surveys, there may be more than one sampling weight when different subsamples are selected. For example, the Demographic and Health Surveys (DHS)¹ select a subsample of adults to be tested for HIV. If 1 in 5 households is selected for HIV testing, then no weighting is needed. But if 1 in 5 urban households and 1 in 2 rural households are selected, then sampling weights need to be applied to both descriptive statistics and model estimates to estimate at national level.

29.4 What happens if you ignore the weights?

In this case (as long as you are modeling the clustering correctly) you are estimating relationships on your sample, rather than the target population. This can be a totally fine thing: if you are interested in how some variables interrelate you might reasonably believe that a found pattern of relationships in the sample is very indicative of how things may play out on a wider stage. It would be odd for (statistically significant) relationships in the sample to not be at least somewhat similar to the population the sample came from. The true magnitudes may shift, but the story should be the same.

For example, if, after ignoring weights, you find an impact of some treatment, then you know the treatment works, at least for those in your sample. Even if your sample is a nonrepresentative sample of your population, it is still some sort of representation, and thus you would believe that your treatment would work to some degree more generally. In this case, any differences between your sample and population would be due to treatment variation, i.e., some in your sample respond differently than some in the population, and so your results in the sample would be weighting some people more than we “should,” causing the discrepancy.

Survey weights are usually much more important when trying to estimate level, or prevalence, of an outcome. If, for example, you are attempting to measure the average literacy in a

¹These are nationally representative surveys conducted in low- and middle-income countries collecting data primarily on maternal and child health.

population, then survey weights will be very important: if the weights of those systematically more (or less) literate are different, then ignoring the weights can cause bias. In addition, if some types of groups or areas are oversampled, then your estimates will tend to be biased towards the levels and relationships in the oversampled group/area. But if you are interested in the relationship between literacy and some covariate, the weights will matter less: it is only if the relationship between these variables is different in your high weight and low weight individuals where you will get bias. This is arguably a less natural phenomenon.

29.5 How to apply weights?

As a rule of thumb, you want to first read any available documentation for the data you are using. You want to understand how the sample was obtained and how the weights were calculated. Publicly available data often comes with manuals on how to handle weights. Some manuals even come with R and Stata code! This is a very important step as sometimes you have to manipulate the weights before you can use them! For example, when using DHS data, you have to divide the weight by 1,000,000 before use. This is a function of how the weights are calculated. In addition, many complex surveys that use weights may also have stratified the sample, and that is also something to account for in your analysis.

When using survey weights it is always advisable to compare the results that include weights with those without them. In general, one should not expect see substantive changes in the point estimates of regression coefficients to the point of dramatically changing one's interpretation of one's results. The model itself is supposed to capture structural relationships between covariates and outcomes, and under correct model specification the weights are superfluous with regards to these coefficients. Where weights could cause change is with descriptives such as the overall averages (e.g., the intercepts, in particular, could be different). We may also see changes in the variance parameters. Finally, with weights, one usually sees an increase in the standard errors.

A limitation with some packages is one might not have an easy way to obtain model fit statistics to help compare models. A clean way to avoid this is to go through the process of model selection using the data in the sample (ignoring the weights) and using the packages and approaches we have seen in class. The findings from such an exploration would be valid for the structure of the data in our sample. Then, as a second step, include the survey weights to move to inference to a larger population (for which the sample is supposed to be representative), taking the preferred model choices from step one and fitting them through a package that allows for survey weights.

29.6 Further references

For some good resources see ([asparouhov2006?](#); [carle2009?](#); [rabe-hesketh2006?](#)). Prior students previously also used ([laukaityte2018importance?](#)) and ([lorah2020estimating?](#)) for some guidance. They then worked with the `BIFIEsurvey` R package to fit multi-level models with survey weights to account for the complex sampling design in their data.

Part IV

FIXED EFFECTS and FRIENDS

30 Pooling

30.1 Pooled/unpooled v.s. fixed/random effects

You may have noticed that we use a couple of different terms interchangeably in this class when it comes to models. Sometimes we talk about coefficients as being completely pooled/partially pooled/unpooled, and sometimes we talk about coefficients as being random or fixed. Yikes, so confusing! Here's a quick document explaining what these various terms mean and what sorts of models they represent. We're only going to be talking about models where the pooling applied to the intercept and slope is the same; most models look like this, and these models are easier to talk about. You should be able to see how you might pool different coefficients differently, though the R code for that can be challenging. We'll use the HSB data, and all of the models we'll consider will look at regressions of math achievement on SES.

30.1.1 Completely pooled

A completely pooled model is a model where we assume that every second-level unit (school) has the same intercept and slope (slopes and intercepts are both completely pooled). This doesn't really have an analog in the fixed/random effects world.

A completely pooled model in this setting might look like

$$\begin{aligned}mathach_i &= \beta_0 + \beta_1 SES_i + \varepsilon_i \\ \varepsilon_i &\sim Normal(0, \sigma^2)\end{aligned}$$

In a completely pooled model we're basically assuming that every school has the same intercept and slope, so we just ignore school membership; notice that we don't even include the j subscript because we're ignoring schools completely. How rude!

We would fit this model with the classic `lm()` call of

```
lm(mathach ~ 1 + ses, data=dat)
```

30.1.2 Partially pooled

A partially pooled model allows for the possibility that different schools might have different slopes and intercepts, but assumes that these slopes and intercepts come from a Normal distribution, which has the effect of pulling them all in towards a grand mean (or *partially pooling* them). This model can also be called a model with random slopes and random intercepts, since we assume that school intercepts and residuals are random draws from a multivariate distribution with means equal to the grand means (and some possibly non-0 correlation). We don't try to estimate these by themselves, only their variances and covariance.

This model can be represented as

$$\begin{aligned} \text{mathach}_{ij} &= \beta_{0j[i]} + \beta_{1j[i]} \text{SES}_i + \varepsilon_i, \\ \beta_{0j} &= \gamma_{00} + u_{0j}, \\ \beta_{1j} &= \gamma_{10} + u_{1j}, \\ \varepsilon_i &\sim \text{Normal}(0, \sigma_\varepsilon^2) \\ \begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} &\sim N \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_0^2 & \rho\sigma_0\sigma_1 \\ \rho\sigma_0\sigma_1 & \sigma_1^2 \end{pmatrix} \right] \end{aligned}$$

We would fit this model with

```
lmer(mathach ~ 1 + ses + (1 + ses|school), data=dat)
```

30.1.3 Unpooled

In an unpooled model, we don't share *any* information across schools about the slopes and intercepts. Instead, we estimate each one separately in each higher-order unit. This is a *fixed-effects* model, because the model treats each school-level slope and intercept as a fixed quantity in the population to be estimated directly. In general parlance, to be a little more precise, a fixed-effects model is a model with unpooled intercepts and completely pooled slopes (although in theory the completely pooled model also has only fixed effects, it's just that those effects are the same in every school; this is why the language of completely pooled, partially pooled, and unpooled coefficients is a little more precise, though it's also less popular).

We could represent an unpooled model as

$$\begin{aligned} \text{mathach}_{ij} &= \beta_{0j[i]} + \beta_{1j[i]} \text{SES}_i + \varepsilon_i \\ \varepsilon_i &\sim \text{Normal}(0, \sigma^2) \end{aligned}$$

We would fit the model with

```
lm(mathach ~ 1 + ses*school)
```

although we might get our estimates in a more useful way by specifying an (identical) model which has no reference school, i.e.,

```
lm(mathach ~ 0 + ses + ses:school)
```

For either of these models to fit you need to ensure that school is coded as a factor and not a number; this is *not* a concern for `lmer()`.

31 Clarification on Fixed Effects and Identification

This chapter talks a bit more about fixed effects. It starts with an overview of the language used to talk about them, gives a brief bit about underidentification, and then moves to looking at how we can have fixed effects interacted with other covariates. The final parts connect to in-class discussion of fixed effects; in particular it gives a reflection on the four concept questions from Packet 1.2 (the live session slides).

31.1 The language of “Fixed Effects”

People will talk about “fixed effects” in (at least) two ways. The first is when you have a dummy variable for each of your clusters, and you are using OLS regression (not multilevel modeling). In this case you are estimating a parameter for each cluster, and we refer to that collection of estimates and parameters that go with these cluster level dummy variables as “fixed effects” and the model is a “fixed effects model.” The second is when you are using multilevel modeling, such as the following:

```
M0 <- lmer(Y ~ 1 + var1 + var2 + var3 + (var1|id), data)
```

When we fit the above model, we will be estimating a grand intercept, and three coefficients for the three variables. Call these β_0 , β_1 , β_2 , and β_3 . We are also estimating a random intercept and random slope for `var1`, with each group defined by the `id` variable having its own random intercept and slope. These are described by a variance-covariance matrix that we have been describing with τ_{00} , τ_{01} , τ_{11} .

Now, the β are the fixed part, or fixed effects, of the model. The τ describe the random part or random effects. This is why, in R, we say `fixef(M0)` to get the β . If we say `ranef(M0)` we get the Empirical Bayes estimates of the random parts for each cluster. If we say `coef(M0)` R adds all this together to give the sum of the fixed part and random part, for each cluster defined by `id`.

Read Gelman and Hill 12.3 for more on this sticky language. G&H do not like “fixed effects” as a description because it is so vague.

31.2 Underidentification

If we fit a model with a dummy variable for each cluster, and a level to variable that does not vary within cluster, we say our model is “underidentified.” We say it is underidentified because no matter how much data we have, we will always have an infinite number of parameter values that can describe our model equally well. For example, say our level 2 variable is a dummy variable (e.g., sector). Then a model where we add five to the coefficient of the level 2 variable, and subtract five from all of the fixed effects for the clusters with sector=1 will fit our data just as well as one where we don’t. We can’t tell the difference! Hence we do not have enough to “identify” the parameter values.

31.3 Model syntax: removing the main ses term vs not

We talked about both these two models:

```
M1 = lm( mathach ~ 0 + ses*id, data=dat.ten )
coef( M1 )
```

```
ses      id1288      id3533      id3881      id4530      id5761      id6074
3.2554487 13.1149374 10.3671216 11.6441421 10.0390287 12.1419451 14.2022643
    id6170      id8800      id9225      id9347 ses:id3533 ses:id3881 ses:id4530
15.6332900  9.1573804 13.9360803 12.9702661 -3.5672188 -0.8647429 -1.6080227
ses:id5761 ses:id6074 ses:id6170 ses:id8800 ses:id9225 ses:id9347
-0.1474381 -1.7263610  1.5563357 -0.6873233 -0.3695565 -0.5694548
```

```
M2 = lm( mathach ~ 0 + ses*id - ses, data=dat.ten )
coef( M2 )
```

```
id1288      id3533      id3881      id4530      id5761      id6074      id6170
13.1149374 10.3671216 11.6441421 10.0390287 12.1419451 14.2022643 15.6332900
    id8800      id9225      id9347 ses:id1288 ses:id3533 ses:id3881 ses:id4530
  9.1573804 13.9360803 12.9702661  3.2554487 -0.3117701  2.3907058  1.6474260
ses:id5761 ses:id6074 ses:id6170 ses:id8800 ses:id9225 ses:id9347
  3.1080106  1.5290877  4.8117843  2.5681254  2.8858922  2.6859939
```

Note how when we remove ses via `- ses` we gain an extra interaction term of `ses:id1288`. In M1, our `ses` coefficient is our baseline slope of school 1288. The ses interaction terms are *slope changes*.

Note how if we add ses to the changes we get back all the slopes in M2:

```
coef( M1 )[12:20] + coef(M1)[[1]]
```

```
ses:id3533 ses:id3881 ses:id4530 ses:id5761 ses:id6074 ses:id6170 ses:id8800  
-0.3117701 2.3907058 1.6474260 3.1080106 1.5290877 4.8117843 2.5681254  
ses:id9225 ses:id9347  
2.8858922 2.6859939
```

Bottom line: M1 and M2 are exactly the same in what they are describing, they are just parameterized differently. Anything we learn from one we could learn from the other.

31.3.1 Plot our model

To plot our model we make a dataset of the intercepts and slopes of each school. Doing this with M2 is much easier than M1, since the coefficients are exactly what we want:

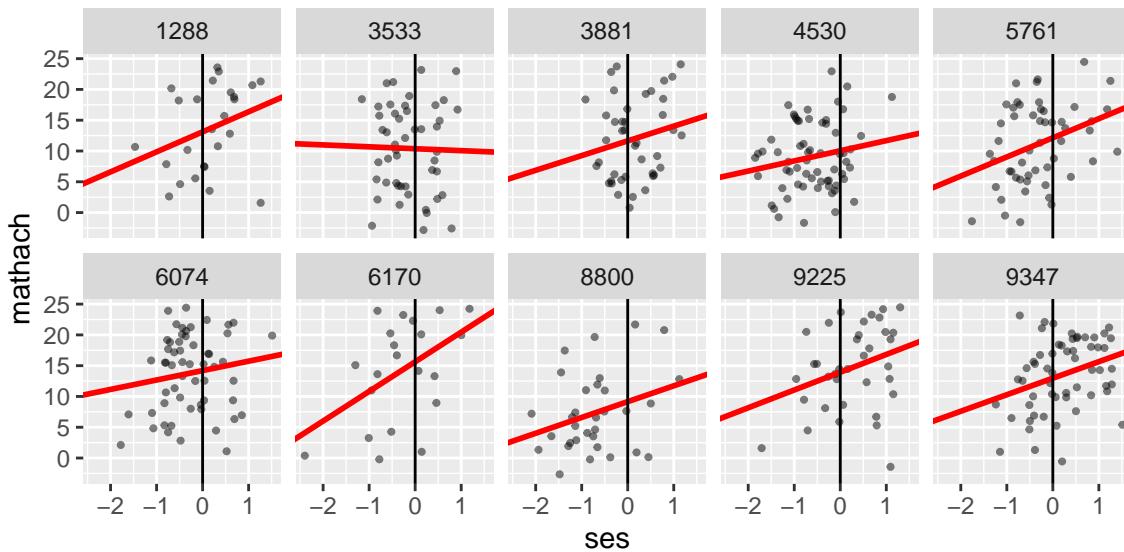
```
lines = data.frame( id = names( coef(M2) )[1:10],  
                    inter = coef(M2)[1:10],  
                    slope = coef(M2)[11:20] )  
  
# we need to fix our IDs. :-(  
lines$id = gsub( "id", "", lines$id)  
head( lines )
```

	id	inter	slope
id1288	1288	13.11494	3.2554487
id3533	3533	10.36712	-0.3117701
id3881	3881	11.64414	2.3907058
id4530	4530	10.03903	1.6474260
id5761	5761	12.14195	3.1080106
id6074	6074	14.20226	1.5290877

(The gsub “substitutes” (replaces) the string “id” with “” in all of our ids so we get back to the actual school ids. Otherwise we will not be able to connect these data to our raw students as easily.)

We now plot!

```
ggplot( dat.ten, aes( ses, mathach ) ) +
  facet_wrap( ~ id, nrow=2 ) +
  geom_point( size=0.75, alpha=0.5 ) +
  geom_abline( data=lines, aes( slope=slope,
                                intercept=inter ),
                col="red", lwd=1 ) +
  geom_vline( xintercept = 0 )
```



31.3.2 What do the intercepts of any of the lines mean?

The intercepts predict what math achievement a student with ses = 0 going to a given school would have. For example, in school 8800, we predict a student with an ses of 0 would have a math achievement of 9.2.

Notice that for some schools the intercept is *extrapolating*. E.g., most of school 8800's students are below 0 for ses, and the intercept is thus describing what we expect for students at the higher end of their range. For school 9225, we are seeing a prediction for students a bit below the middle of their range.

31.3.3 What differences, if any, are there between running a new linear model on each school vs. running the interacted model on the set of 10 schools?

The lines would be *exactly* the same. The standard errors are different. Here is the line on just school 1288:

```
s1288 = filter( dat.ten, id == "1288" )
M_1288 = lm( mathach ~ 1 + ses, data=s1288 )
summary( M_1288 )
```

Call:
`lm(formula = mathach ~ 1 + ses, data = s1288)`

Residuals:

Min	1Q	Median	3Q	Max
-15.648	-5.700	1.048	4.420	9.415

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	13.115	1.387	9.456	2.17e-09 ***
ses	3.255	2.080	1.565	0.131

Signif. codes:	0 '***'	0.001 '**'	0.01 '*'	0.05 '.'
	0.1 ''	1		

Residual standard error: 6.819 on 23 degrees of freedom
Multiple R-squared: 0.09628, Adjusted R-squared: 0.05699
F-statistic: 2.45 on 1 and 23 DF, p-value: 0.1312

The SEs will be different, however. Compare:

```
sum = summary( M2 )
sum$coefficients[ c(1, 11 ), ]
```

	Estimate	Std. Error	t value	Pr(> t)
id1288	13.114937	1.291495	10.154850	9.121968e-22
ses:id1288	3.255449	1.936454	1.681139	9.349559e-02

In this case the SEs are close, but they could be a lot different if we have a lot of heteroskedasticity or the school has few data points so we do a bad job estimating uncertainty.

The key is in the single model we are using *all* the schools to estimate the residual variance, and this is the number that drives our SE estimates.

31.3.4 Do we trust the red lines on the plot? Why or why not?

We trust them because they are driven just by the school data, so they are essentially unbiased. But these are small datasets, so they are unstable.

31.3.5 What about the variability in the slopes and intercepts of the red lines?

The variation is not to be trusted. The slopes are varying because of measurement error. For example, it is unlikely school 3533 really has a negative slope. It is more likely we just got some low performing high ses kids by happenstance in our sample. Similarly, it is unlikely school 6170 has such a steep slope. It has few kids, and the kid with less than -2 ses and a very low math achievement is likely an influential point in that regression.

31.4 Further Reading

(Antonakis, Bastardoz, and Rönkkö 2019)

32 A tour of fixed effects and cluster-robust SEs

This brief handout will walk through fixed effects and cluster robust standard errors, with a stop at aggregation and heteroskedastic standard errors.

Consider the following research question:

RQ: Are there differences in math achievement for Catholic vs public schools, controlling for differences in SES?

32.1 Aggregation

One way forward is to aggregate our HS&B data and merge it into our school-level data, and then analyze the result.

We aggregate as so:

```
col.dat = dat %>% group_by( id ) %>%
  summarize( per.fem = mean(female),
             per.min = mean(minority),
             mean.ses = mean(ses),
             mean.ach = mean(mathach),
             n.stud = n() )

# combine our school-level variables (ours and theirs) into one data.frame
sdat = merge( sdat, col.dat, by="id", all=TRUE )
head( sdat )
```

	id	size	sector	pracad	disclim	himinty	meances	per.fem	per.min
1	1224	842	0	0.35	1.597	0	-0.428	0.5957447	0.08510638
2	1288	1855	0	0.27	0.174	0	0.128	0.4400000	0.12000000
3	1296	1719	0	0.32	-0.137	1	-0.420	0.6458333	0.97916667
4	1308	716	1	0.96	-0.622	0	0.534	0.0000000	0.40000000
5	1317	455	1	0.95	-1.694	1	0.351	1.0000000	0.72916667

```

6 1358 1430      0   0.25   1.535      0  -0.014 0.3666667 0.1000000
    mean.ses  mean.ach n.stud
1 -0.43438298  9.715447    47
2  0.12160000 13.510800    25
3 -0.42550000  7.635958    48
4  0.52800000 16.255500    20
5  0.34533333 13.177687    48
6 -0.01966667 11.206233    30

```

We can now answer our research question with a school-level regression:

```

Magg = lm( mean.ach ~ 1 + sector + mean.ses, data=sdat )

library( sandwich )
library( lmtest )
coeftest( Magg, vcov. = vcovHC, type = "HCO" )

```

t test of coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	12.11950	0.18743	64.6605	< 2.2e-16 ***
sector	1.22194	0.31959	3.8235	0.0001894 ***
mean.ses	5.38738	0.33769	15.9538	< 2.2e-16 ***

Signif. codes:	0 '***'	0.001 '**'	0.01 '*'	0.05 '.'
	0.1	' '	1	

We use heteroskedastic robust standard errors to take into account possible heteroskedasticity due to, for example, some school outcomes being based on smaller numbers of students (and thus having more variation) than other school outcomes.

In this regression we are controlling for school mean SES, not student SES. If anything is going on *within school* between SES and math achievement, and that is maybe different for different sectors, we might be missing it.

32.2 Student level regression and cluster robust SES

Instead of using our aggregated data, we can merge our *school-level* variables into the student data and run a student level regression:

The merge brings in level 2 variables, repeating them for each student in a school:

```
dat = merge( dat, sdat, by="id" )
head( dat )
```

	id	minority	female	ses	mathach	size	sector	pracad	disclim	himinty
1	1224	0	1	-1.528	5.876	842	0	0.35	1.597	0
2	1224	0	1	-0.588	19.708	842	0	0.35	1.597	0
3	1224	0	0	-0.528	20.349	842	0	0.35	1.597	0
4	1224	0	0	-0.668	8.781	842	0	0.35	1.597	0
5	1224	0	0	-0.158	17.898	842	0	0.35	1.597	0
6	1224	0	0	0.022	4.583	842	0	0.35	1.597	0
		meanses	per.fem	per.min	mean.ses	mean.ach	n.stud			
1	-0.428	0.5957447	0.08510638	-0.434383	9.715447		47			
2	-0.428	0.5957447	0.08510638	-0.434383	9.715447		47			
3	-0.428	0.5957447	0.08510638	-0.434383	9.715447		47			
4	-0.428	0.5957447	0.08510638	-0.434383	9.715447		47			
5	-0.428	0.5957447	0.08510638	-0.434383	9.715447		47			
6	-0.428	0.5957447	0.08510638	-0.434383	9.715447		47			

And now our regression

```
Mstud = lm( mathach ~ 1 + sector + ses, data = dat )
summary( Mstud )
```

Call:

```
lm(formula = mathach ~ 1 + sector + ses, data = dat)
```

Residuals:

Min	1Q	Median	3Q	Max
-20.1339	-4.6785	0.1504	4.8894	16.5847

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)							
(Intercept)	11.79325	0.10610	111.15	<2e-16 ***							
sector	1.93501	0.15249	12.69	<2e-16 ***							
ses	2.94856	0.09783	30.14	<2e-16 ***							

Signif. codes:	0	'***'	0.001	'**'	0.01	'*'	0.05	'..'	0.1	' '	1

Residual standard error: 6.345 on 7182 degrees of freedom

Multiple R-squared: 0.1492, Adjusted R-squared: 0.149

F-statistic: 629.8 on 2 and 7182 DF, p-value: < 2.2e-16

32.2.1 Getting the right standard errors

The standard errors for the above regression, however, is **wrong**: we are not taking the clustering into account. We can fix this with cluster-robust standard errors:

```
coeftest( Mstud, vcov. = vcovCL, cluster = dat$id )  
  
t test of coefficients:  
  
              Estimate Std. Error t value Pr(>|t|)  
(Intercept) 11.79325    0.20315 58.0532 < 2.2e-16 ***  
sector       1.93501    0.31718  6.1007 1.111e-09 ***  
ses          2.94856    0.12794 23.0469 < 2.2e-16 ***  
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Catholic schools score 2 points higher than Public, on average, beyond individual level SES.

We can further control for school mean SES, like with aggregation:

```
Mstud2 = lm( mathach ~ 1 + sector + ses + meanses, data = dat )  
coeftest( Mstud2, vcov. = vcovCL, cluster = dat$id )
```

```
t test of coefficients:  
  
              Estimate Std. Error t value Pr(>|t|)  
(Intercept) 12.09792    0.17024 71.0629 < 2.2e-16 ***  
sector       1.28039    0.30008  4.2669 2.008e-05 ***  
ses          2.19116    0.12981 16.8802 < 2.2e-16 ***  
meanses     2.97267    0.36491  8.1464 4.388e-16 ***  
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The contextual value of school mean SES is explaining some of the difference between Catholic and public schools, here: note the reduction of the coefficient for `sector`. That being said, and still accounting for clustering, `sector` is still quite significant.

Confidence intervals give:

```
coefci(Mstud2, vcov. = vcovCL, cluster = dat$id )
```

	2.5 %	97.5 %
(Intercept)	11.7641926	12.431643
sector	0.6921486	1.868627
ses	1.9367009	2.445618
meanses	2.2573428	3.687995

Relative to the overall standard deviation of math achievement we have:

```
sd_math = sd( dat$mathach )
sd_math
```

```
[1] 6.878246
```

```
coefci(Mstud2, vcov. = vcovCL, cluster = dat$id ) / sd_math
```

	2.5 %	97.5 %
(Intercept)	1.7103478	1.8073858
sector	0.1006286	0.2716720
ses	0.2815690	0.3555583
meanses	0.3281858	0.5361825

The difference between Catholic and public schools is somewhere between 0.10 and 0.27 standard deviations, beyond what can be explained by ses.

32.3 And fixed effects?

We can combine fixed effects and cluster robust standard errors quite easily, but we cannot combine fixed effects and level 2 covariates at all. We next look at this latter problem, and then see what combining these options looks like when asking questions that do not rely on level 2 variables for main effects.

32.3.1 The problem of fixed effects and dummy variables

Fixed effects cannot be used to take into account school differences if we are interested in level 2 variables, because the fixed effects and level 2 variables are *co-linear*. Put another way, if we let each school have its own mean outcome (represented by the coefficient for a dummy variable for that school), then we can't have a variable like `sector` to measure how Catholic schools are different from public schools, conditioned on all the school mean outcomes. There is nothing left to explain as, by construction, there are no differences in school mean outcomes once we “control for” the individual school mean outcomes via fixed effects!

What R will do when you give colinear variables is drop the extra ones. Here is a mini-example fake dataset of 4 schools with 3 students in each school:

```
fake = tibble( id = c( 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4),
               mathach = rnorm( 12 ),
               ses = rnorm( 12 ),
               sector = c( 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1 ) )
fake$id = as.factor( fake$id )
fake
```

```
# A tibble: 12 x 4
  id     mathach     ses sector
  <fct>    <dbl>    <dbl>  <dbl>
1 1        0.0372   -0.607    0
2 1       -0.189    1.68     0
3 1       -2.47    -0.494    0
4 2        1.41    -0.533    1
5 2       -0.913    1.84     1
6 2       -1.68    1.60     1
7 3       -0.296    0.608    0
8 3        0.00987  1.79     0
9 3        0.695    0.528    0
10 4      -0.618   -0.188    1
11 4       0.449   -1.34     1
12 4      0.0944   0.543    1
```

And our regression model with fixed effects for school plus our school-level `ses` gives this:

```
lm( mathach ~ 0 + id + ses + sector, data = fake )
```

Call:

```

lm(formula = mathach ~ 0 + id + ses + sector, data = fake)

Coefficients:
      id1      id2      id3      id4      ses    sector
-0.80703 -0.06465  0.46942 -0.13658 -0.34088       NA

```

Note the NA for sector! We cannot estimate it due to colinearity, so it got dropped.

32.3.2 Using fixed effects to handle clustering

That being said, fixed effects are an excellent way to control for school differences when looking at *within-school relationships*. For example, we can ask how math relates to SES within schools, controlling for systematic differences across schools.

Here is the no fixed effect regression, and the fixed effect regression:

```

Mstud3_noFE = lm( mathach ~ 1 + ses, data=dat )

dat$id = as.factor(dat$id)
Mstud3 = lm( mathach ~ 0 + ses + id, data=dat )
head( coef( Mstud3 ) )

ses      id1224      id1288      id1296      id1308      id1317
2.191172 10.667255 13.244353  8.568302 15.098561 12.421003

```

For our fixed effect model, we will have lots of coefficients because we have a fixed effect for each school; the `head()` command is just showing us the first few. We also had to explicitly make our `id` variable a factor (categorical variable), so R doesn't think it is a continuous covariate.

For our standard errors, etc., we can further account for clustering of our residuals above and beyond what can be explained by our fixed effects (even if we subtract out the mean outcome, we might still have dependencies between students within a given school). So we use our cluster-robust standard errors as so:

```

coefs <- coeftest( Mstud3, vcov. = vcovCL, cluster = dat$id )
head( coefs )

Estimate Std. Error   t value   Pr(>|t|)
ses      2.191172 0.13124281 16.69556 2.130074e-61
id1224 10.667255 0.05700964 187.11316 0.000000e+00

```

```

id1288 13.244353 0.01595913 829.89215 0.000000e+00
id1296  8.568302 0.05584382 153.43332 0.000000e+00
id1308 15.098561 0.06929621 217.88439 0.000000e+00
id1317 12.421003 0.04532252 274.05809 0.000000e+00

```

We have again used `head()` to just get the first lines. The whole printout would be one line per school, plus the ses coefficient!

Let's compare our three models (note the way we omit coefficients with `id` to drop our fixed effects from the table):

```

library( texreg )
screenreg( list( `No FE` = Mstud3_noFE, `FE` = Mstud3, `FE + CRVE` = coefs ),
           omit.coef = "id" )

```

	No FE	FE	FE + CRVE
<hr/>			
(Intercept)	12.75 *** (0.08)		
ses	3.18 *** (0.10)	2.19 *** (0.11)	2.19 *** (0.13)
<hr/>			
R^2	0.13	0.83	
Adj. R^2	0.13	0.82	
Num. obs.	7185	7185	
<hr/>			
*** p < 0.001; ** p < 0.01; * p < 0.05			

A few things to note:

- Not having fixed effects means we are getting an estimate of the math-ses relationship *including* school level context. Note the higher point estimate. Often we want to focus on within-school relationships. Fixed effects does this.
- The standard errors are larger once we include fixed effects; the fixed effects are partially accounting for clustering.
- The standard errors are even larger when we include CRVE. It is more fully accounting for the clustering, and the fact that the clusters themselves could vary. In general, one should typically use CRVE in addition to fixed effects, if one wants to view the clusters as representative of a larger population (in this case a larger population of schools).

32.3.3 Bonus: Interactions with level-2 variables are OK, even with fixed effects

If we want to see if the *relationship* of math and SES is different between schools, we can get tricky like so:

```
Mstud4 = lm( mathach ~ 0 + ses + ses:sector + id, data=dat )
head( coef( Mstud4 ) )
```

```
ses      id1224     id1288     id1296     id1308     id1317
2.782105 10.923946 13.172496  8.819744 15.498595 12.682641
```

```
tail( coef( Mstud4 ) )
```

```
id9359      id9397      id9508      id9550      id9586 ses:sector
14.763044   9.982311  13.772485  10.941590  13.973252 -1.348572
```

Note interaction terms always get pushed to the end of the list of estimates by R. So we have to pull them out with `tail()`.

In the following we compare SEs.

```
a <- coeftest( Mstud4, vcov. = vcovCL, cluster = dat$id )
library( texreg )
screenreg( list( wrong=Mstud4, adjusted=a ),
           omit.coef="id", single.row = TRUE )
```

```
=====
          wrong           adjusted
-----
ses            2.78 (0.14) ***    2.78 (0.16) ***
ses:sector    -1.35 (0.22) ***  -1.35 (0.24) ***
-----
R^2            0.83
Adj. R^2       0.82
Num. obs.     7185
=====
*** p < 0.001; ** p < 0.01; * p < 0.05
```

In our second column we are accounting for our clustering with our cluster robust SEs.

33 MLM and Cluster-Robust Standard Errors

We have talked about multilevel modeling, and talked about cluster robust standard errors. We can actually have both. In STATA, you just write “, robust” in your modeling command. In R, you have to use the `clubSandwich` package. Let’s illustrate with the High School and Beyond data.

33.1 Robust standard errors without multilevel modeling

As a reminder, here is how to get classic Huber-White / Sandwich / Heteroskedastic-Robust Standard Errors for vanilla OLS for a single school (school 8857):

First we fit our regression, like we would normally:

```
one.sch = filter( dat, id == "8857" )
nrow( one.sch )
```

```
[1] 64
```

```
M0 <- lm( mathach ~ 1 + ses, dat)
arm::display( M0 )
```

```
lm(formula = mathach ~ 1 + ses, data = dat)
      coef.est coef.se
(Intercept) 12.75     0.08
ses          3.18     0.10
---
n = 7185, k = 2
residual sd = 6.42, R-Squared = 0.13
```

Then we use the `sandwich` and `lmtest` package:

```
library( sandwich )
library( lmtest )
lmtest::coeftest(M0, type = "HC1")
```

```

t test of coefficients:

            Estimate Std. Error t value Pr(>|t|)
(Intercept) 12.747396   0.075686 168.424 < 2.2e-16 ***
ses         3.183870   0.097121  32.782 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

And here is cluster robust standard errors when you have clustered data, but have not bothered with multilevel modeling:

```

M1 = lm( mathach ~ 1 + ses + sector, data = dat )
lmtest::coeftest( M1, type = "CL", cluster = dat$id )

```

```

t test of coefficients:

            Estimate Std. Error t value Pr(>|t|)
(Intercept) 11.793254   0.106102 111.150 < 2.2e-16 ***
ses         2.948558   0.097831  30.139 < 2.2e-16 ***
sector      1.935013   0.152493  12.689 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

33.2 CRSE + Multilevel Modeling

Ok, so we have seen how to get robust standard errors in the above; how do we combine them with multilevel modeling? First, let's fit our multilevel model:

```

M2 = lmer( mathach ~ 1 + ses + sector + (1|id), data=dat )
display( M2 )

```

```

lmer(formula = mathach ~ 1 + ses + sector + (1 | id), data = dat)
            coef.est coef.se
(Intercept) 11.72      0.23
ses          2.37      0.11
sector       2.10      0.34

```

Error terms:

```

Groups      Name           Std.Dev.
id          (Intercept)  1.92
Residual                6.09
---
number of obs: 7185, groups: id, 160
AIC = 46621.2, DIC = 46601.7
deviance = 46606.4

```

If we believe all our MLM assumptions, we can get our vanilla standard errors as so:

```
summary( M2 )$coef
```

	Estimate	Std. Error	t value
(Intercept)	11.718908	0.2280585	51.385537
ses	2.374711	0.1054911	22.511017
sector	2.100837	0.3411243	6.158567

If we don't believe them fully, we might want to make our inference more robust. Before we turn to this, first note that our point estimates can be different for MLM vs OLS:

```
coef( M1 )
```

(Intercept)	ses	sector
11.793254	2.948558	1.935013

```
fixef( M2 )
```

(Intercept)	ses	sector
11.718908	2.374711	2.100837

The assumption of the random effects means we are not weighting all our data the same way. For example, if we find the clusters vary in size a lot, we might weight the clusters more equally when estimating a cluster-level coefficient (e.g., sector) instead of counting on the big clusters more.

Regardless, we might worry that complex dependencies within our clusters are messing up our standard errors in our MLM, however. Fixing that is easy:

```

library( clubSandwich )
club <- coef_test( M2,
  vcov = "CR1S",
  test = "Satterthwaite")
club

  Coef. Estimate     SE t-stat d.f. (Satt) p-val (Satt) Sig.
(Intercept)    11.72 0.226  51.86      88.9 <0.001 ***
ses            2.37 0.119   19.89     142.8 <0.001 ***
sector         2.10 0.347    6.05     149.5 <0.001 ***

```

The `clubSandwich` package works for multi-level models fit with either `lme4::lmer()` or `nlme::lme()`. Note that `coef_test` is *not* the same as `coeftest`. The `vcov = "CR1S"` replicates the Stata SEs (or so it has been speculated, and assuming they use the same correction as for panel data models).

We can compare the SEs as so:

```

rbind( club$SE, se.fixef(M2) )

  (Intercept)      ses      sector
[1,] 0.2259713 0.1193704 0.3474632
[2,] 0.2280585 0.1054911 0.3411243

```

We see that the SEs did not change much from the homoskedastic- and within-cluster-independence-assuming standard errors of the vanilla lmer call in this circumstance.

33.2.1 Misspecified how?

The sorts of misspecification that we might be worried about are things such as the following:

1. Using a random intercept model when the real data-generating process has a random slope;
2. Using a model that assumes homogeneous random effects when the real data-generating process involves heteroskedasticity (e.g., different random effects variances for treatment schools than for control schools);
3. Using a model with a single level of random effects (e.g., school random effects) when the real data-generating process has multiple levels of structure (e.g., school and classroom random effects); or
4. Assuming homoscedastic variance for the lowest-level errors when the real process is heteroskedastic or has some other structure.

Of the above (3) could have “secret clustering” in your clusters, and give you radically incorrect standard errors. The other options are more violations of homoskedasticity, and are likely to not be as serious of concerns. You can also diagnose them with residual plots, and see if you are seeing more scatter in your data for some groups or individuals than others.

Regardless, if you are worried about these things, then the above will give you improved standard errors.

33.3 Some technical notes

So what is this thing even doing? In the following I describe a rough approximation. The key idea is that a multilevel model specification is specifying a parameterized $n \times n$ variance-covariance matrix of the residuals of a generic linear model. Due to our assumption of independent clusters, this matrix is block diagonal with blocks V_1, \dots, V_J , with block V_j corresponding to group j . For a random intercept model, for example, block j would be a $n_j \times n_j$ matrix with $\tau_{00} + \sigma^2$ for the diagonal and τ_{00} for the off-diagonal, with τ_{00} being the variance of the random intercepts and σ^2 being the within-block residual variance.

If we write our multilevel model in reduced form, we can write it as a mini-regression for each group j of:

$$Y_j = X_j \vec{\beta} + Z_j \vec{r}_j + e_j,$$

where Y_j is the vector of outcomes, X_j and Z_j are mini design matrices of covariates (including a column of 1s for the intercept, normally), with X_j being all the covariates and Z_j being those covariates that have corresponding random effects (also with a column of 1s), $\vec{\beta}$ the vector of coefficients (the fixed effects), \vec{r}_j the vector of random effects for group j , and e_j the vector of residuals.

Importantly, the $u_j := Z_j \vec{u}_j + e_j$ is all residual, and $V_j = \text{Var}(u_j)$: the variance-covariance matrix of the residuals is determined by this structure and our assumptions on \vec{u}_j being multivariate normal and the e_j being a vector of independent residual draws (the ϵ_{ij}).

Now, given this view of our multilevel model, we can estimate this with generalized least squares. Generalized least squares is a generic regression technique where, if you have a parameterized covariance matrix on the residuals, you can estimate your regression coefficients taking that correlation structure into account. Think of it as a three-step process: first fit the regression without taking the residuals into account, then use the fit model to estimate our big $n \times n$ variance-covariance matrix, and then use this estimated matrix as a set of weights that we plug back into a least squares estimation.

In particular, the estimator for $\vec{\beta}$ is weighted least squares:

$$\hat{\beta} = (X'WX)^{-1}X'WY,$$

with W a weight matrix that is a $n \times n$ block-diagonal matrix formed from the inverses of the estimated V_j . Now if the random effects structure (the assumed distribution of the r_j) is misspecified or the residual error structure (on the ϵ_{ij}) is wrong, then V_j will be wrong, but $\hat{\beta}$ will still be asymptotically consistent (under some conditions).

Cluster-robust methods use the empirical residuals (the \hat{u}_j) to assess the uncertainty in $\hat{\beta}$ as an estimate of the β as defined by the implied weights W . Even if the random effects part of the model is wrong, the assumption of independent clusters means our inference on this estimand is still right. The key idea is cluster-robust methods take a weighted average of J very badly estimated variance-covariance matrices to get a decent estimate of overall population-level uncertainty.

The main advantage of the `clubSandwich` package is it will take our multilevel model and do this cluster-robust standard error calculation. Even better, however, is it will (using “CR2” adjustment) try to improve the basic sandwich estimator by 1) adjusting the residuals (the \hat{u}_j) a bit so that the variance estimator is exactly unbiased if the working model is exactly correct and b) using Satterthwaite degrees of freedom (or generalizations thereof) for tests/confidence intervals, also derived under the assumption that the working model is exactly correct.

33.4 Acknowledgements

Thanks to James Pustejovsky, the creator of the `clubSandwich` package, for the help in thinking this through. Much of these notes, in particular the reasons for misspecification and much of the technical notes, are liberally stolen from emails with this fine colleague.

Part V

WORKED EXAMPLES

34 Code for HSB Example in Chapter 4 of R&B

This script builds everything from Chapter 4 of Raudenbush and Bryk in R. It is a very useful script for getting pretty much all the code you would need for a conventional multilevel analysis. The code is divided by each table or plot from the chapter.

34.1 R Setup

```
library(foreign) #this lets us read in spss files
library(tidyverse) #this is a broad package that allows us to do lots of data management-y things
library(lme4) #this allows us to run MLM
library(arm) #this allows us to display MLM
library( lmerTest ) # this puts p-values on the summary() command for fixed effects
```

34.2 Load HS&B data

```
# Read student data
stud.dat = read.spss( "data/hsb1.sav", to.data.frame=TRUE )

# Read in school data
sch.dat = read.spss( "data/hsb2.sav", to.data.frame=TRUE )

# Make single data frame with all variables, keep all students even if they
# don't match to a school
dat = merge( stud.dat, sch.dat, by="id", all.x=TRUE )
```

34.3 Table 4.1 Descriptive summaries

```
## Get mean and SD of the Level 1 variables, rounded to 2 decimal places
# math achievement
round(mean(dat$mathach),2)
```

```
[1] 12.75
```

```
round(sd(dat$mathach),2)
```

```
[1] 6.88
```

```
# ses
round(mean(dat$ses),2)
```

```
[1] 0
```

```
round(sd(dat$ses),2)
```

```
[1] 0.78
```

```
## Get mean and SD of Level 2 variables, round to 2 decimal places
# NOTE: we are getting these from the SCHOOL-LEVEL FILE
# sector
round(mean(sch.dat$sector),2) # this answers "what percent of schools are catholic?"
```

```
[1] 0.44
```

```
round(sd(sch.dat$sector),2)
```

```
[1] 0.5
```

```
# mean ses
round(mean(sch.dat$meanses),2) # this answers "what is the average of the school-average SES
```

```
[1] 0
```

```
round(sd(sch.dat$meanses),2)
```

```
[1] 0.41
```

```
# NOTE: if we used the student-level or "dat" file, we would be answering the
# following questions:
# * what percent of students attend a catholic school?
# * what is the average student ses? <- this would match what we calculated
# ourselves if we had the entire school in our sample
```

34.4 Table 4.2: One-Way ANOVA (i.e uncontrolled random intercept)

```
## Fit the model described
mod4.2 <- lmer(mathach ~ 1 + (1|id), data=dat)
# Peek at the results
display(mod4.2)
```

```
lmer(formula = mathach ~ 1 + (1 | id), data = dat)
coef.est  coef.se
12.64     0.24
```

Error terms:

Groups	Name	Std.Dev.
id	(Intercept)	2.93
	Residual	6.26

```
number of obs: 7185, groups: id, 160
AIC = 47122.8, DIC = 47114.8
deviance = 47115.8
```

```
## Extract the fixed effect coefficient (and it's standard error)
fixef(mod4.2) # extracts the fixed effect coefficient(s)
```

```
(Intercept)
```

```
12.63697
```

```
se.coef(mod4.2)$fixef #extracts the standard errors for the fixed effect(s)
```

```
[1] 0.2443936
```

```
## Extract the variance components
```

```
# Note: in the model display, we see the SDs, not the variance
```

```
VarCorr(mod4.2)
```

Groups	Name	Std.Dev.
id	(Intercept)	2.9350
	Residual	6.2569

```
# To get the variances, we extract each part and square it
```

```
# variance of random intercept
```

```
(sigma.hat(mod4.2)$sigma$id)^2
```

```
(Intercept)
```

```
8.614025
```

```
# variance of level 1 residual (easier to extract)
```

```
sigma(mod4.2)^2
```

```
[1] 39.14832
```

```
# could also use the more complicated formula that we used with the intercept.
```

```
# If we do, we get the same thing
```

```
sigma.hat(mod4.2)$sigma$data^2
```

```
[1] 39.14832
```

```
# Inference on the need for a random intercept
```

```
# Thus uses the book's way of calculating a test statistic with a
```

```
# chi-squared distribution.
```

```
schools = dat %>% group_by( id ) %>%
```

```
  summarise( nj = n(),
```

```
            Y.bar.j = mean( mathach ) )
```

```
gamma.00 = fixef( mod4.2 )[[1]]
```

```
sigma.2 = sigma(mod4.2)^2
```

```
H = sum( schools$nj * (schools$Y.bar.j - gamma.00)^2 / sigma.2 )
```

```
H
```

```
[1] 1660.232
```

```
# our p-value
pchisq( H, df = nrow( schools ) - 1, lower.tail = FALSE )
```

```
[1] 4.770612e-248
```

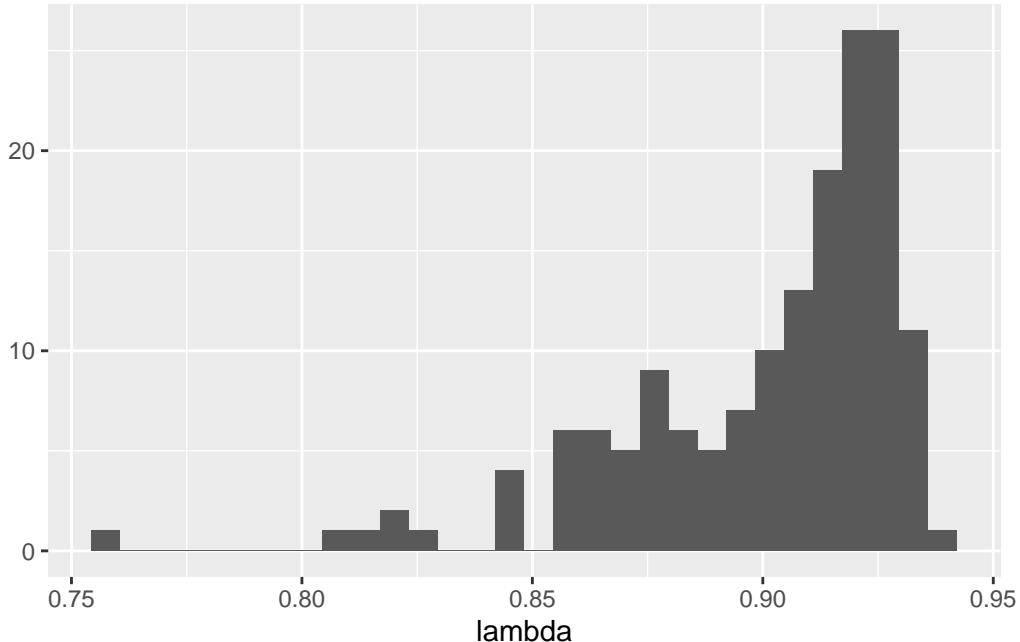
```
# calculating the ICC
tau.00 = VarCorr(mod4.2)$id[1,1]
rho.hat = tau.00 / (tau.00 + sigma.2 )
rho.hat
```

```
[1] 0.1803518
```

```
# Calculating reliability for each school mean. (Here it is purely a function of
# students in the school. More students, more info, and thus more reliable.)
sigma.2 = sigma(mod4.2)^2
tau.00 = VarCorr(mod4.2)$id[1,1]
lambda = tau.00 / ( tau.00 + sigma.2 / schools$nj )
mean( lambda )
```

```
[1] 0.9013773
```

```
# A bonus graph of the reliabilities
qplot( lambda )
```



34.5 Table 4.3 Means as Outcomes Model

```
# (i.e. random intercept with Level 2 predictor)
## Fit the model described
mod4.3 <- lmer(mathach ~ 1 + meanses + (1|id), data=dat)

# Peek at the results
display(mod4.3)
```

```
lmer(formula = mathach ~ 1 + meanses + (1 | id), data = dat)
      coef.est coef.se
(Intercept) 12.65     0.15
meanses      5.86     0.36

Error terms:
Groups   Name        Std.Dev.
id      (Intercept) 1.62
Residual          6.26
---
number of obs: 7185, groups: id, 160
```

```

AIC = 46969.3, DIC = 46956.9
deviance = 46959.1

## Extract the fixed effect coefficients (and standard errors/t-statistics)
fixef(mod4.3) # extracts the fixed effect coefficients

(Intercept)      meanses
12.649435     5.863538

# NOTE: you can call them separately by "indexing" them
# just the intercept
fixef(mod4.3)[1]

(Intercept)
12.64944

# just coefficient on mean ses
fixef(mod4.3)[2]

meanses
5.863538

se.coef(mod4.3)$fixef #extracts the standard errors for the fixed effect(s)

[1] 0.1492801 0.3614580

## Calculate (or extract) the t-ratio (aka the t-statistic)

# NOTE: the author's don't present this for the intercept, because we often
# don't care. But it is presented here for completeness

# tstats for intercept
fixef(mod4.3)[1]/se.coef(mod4.3)$fixef[1]

(Intercept)
84.73622

```

```
# tstat mean ses
fixef(mod4.3)[2]/se.coef(mod4.3)$fixef[2]
```

```
meanses
16.22191
```

```
# tstat extracted - this does both variables at once!
coef(summary(mod4.3))[, "t value"]
```

```
(Intercept)      meanses
84.73622      16.22191
```

```
# NOTE: Let's look at what is happening here:
coef(summary(mod4.3)) # gives us all the fixed effect statistics we could want
```

	Estimate	Std. Error	df	t value	Pr(> t)
(Intercept)	12.649435	0.1492801	153.7425	84.73622	6.032590e-131
meanses	5.863538	0.3614580	153.4067	16.22191	4.267894e-35

```
# the [ ] is called "indexing" - it's a way of subsetting data by telling R
# which [rows,columns] you want to see we are telling R that we want ALL rows "["
# , " but only the column labeled "t value"
```

```
## Extract the variance components
# Note: in the model display, we see the SDs, not the variance
VarCorr(mod4.3)
```

Groups	Name	Std.Dev.
id	(Intercept)	1.6244
	Residual	6.2576

```
# To get the variances, we extract each part and square it
# variance of random intercept
(sigma.hat(mod4.3)$sigma$id)^2
```

```
(Intercept)
2.638708
```

```

# variance of level 1 residual
sigma(mod4.3)^2

[1] 39.15708

# Range of plausible values for school means for schools with mean SES of 0:
# See page 73-74)
fixef( mod4.3 )[[1]] + c(-1.96, 1.96) * (sigma.hat(mod4.3)$sigma$id)

[1] 9.465592 15.833279

# Compare to our model without mean ses
fixef( mod4.2 )[[1]] + c(-1.96, 1.96) * (sigma.hat(mod4.2)$sigma$id)

[1] 6.884441 18.389507

# Proportion reduction in variance or "variance explained" at level 2
tau.00.anova = (sigma.hat(mod4.2)$sigma$id)^2
tau.00.meanses = (sigma.hat(mod4.3)$sigma$id)^2
(tau.00.anova-tau.00.meanses) / tau.00.anova

(Intercept)
0.693673

## Inference on the random effects
schools = merge( schools, sch.dat, by="id" )
gamma.00 = fixef( mod4.3 )[[1]]
gamma.01 = fixef( mod4.3 )[[2]]
schools = mutate( schools, resid = Y.bar.j - gamma.00 - gamma.01*meanses )
H = sum( schools$nj * schools$resid^2 ) / sigma(mod4.3)^2
H

[1] 633.5175

pchisq( H, nrow( schools ) - 2, lower.tail = FALSE )

[1] 3.617696e-58

```

```
## Reliability revisited (from pg 75)
mod4.3
```

```
Linear mixed model fit by REML ['lmerModLmerTest']
Formula: mathach ~ 1 + meanses + (1 | id)
Data: dat
REML criterion at convergence: 46961.28
Random effects:
Groups   Name      Std.Dev.
id       (Intercept) 1.624
Residual           6.258
Number of obs: 7185, groups: id, 160
Fixed Effects:
(Intercept)    meanses
12.649        5.864
```

```
u.hat = coef(mod4.3)$id
head(u.hat)
```

```
(Intercept) meanses
1224     12.32688 5.863538
1288     12.71898 5.863538
1296     10.70101 5.863538
1308     12.92208 5.863538
1317     11.48086 5.863538
1358     11.73878 5.863538
```

```
sigma.2 = sigma(mod4.3)^2
tau.00 = VarCorr(mod4.3)$id[1,1]
sigma.2
```

```
[1] 39.15708
```

```
tau.00
```

```
[1] 2.638708
```

```

# These are the individual reliabilities---how well we can separate schools with the same Mean SES
# (So it is _conditional_ on the mean SES of the schools.)
lambda.j = tau.00 / (tau.00 + (sigma.2 / schools$nj))
mean( lambda.j )

```

```
[1] 0.7400747
```

34.6 Table 4.4 Random coefficient model (i.e. random slope)

```

# group-mean center ses
dat <- dat %>% group_by( id ) %>%
  mutate( ses_grpcenter = ses - mean(ses) )

## Fit the model described
mod4.4 <- lmer(mathach ~ 1 + ses_grpcenter + ( 1 + ses_grpcenter | id ), data=dat)
# Peek at the results
display(mod4.4)

lmer(formula = mathach ~ 1 + ses_grpcenter + (1 + ses_grpcenter |
  id), data = dat)
  coef.est  coef.se
(Intercept) 12.64     0.24
ses_grpcenter 2.19     0.13

Error terms:
Groups   Name        Std.Dev. Corr
id       (Intercept) 2.95
          ses_grpcenter 0.83     0.02
Residual           6.06
---
number of obs: 7185, groups: id, 160
AIC = 46726.2, DIC = 46707.7
deviance = 46711.0

## Extract the fixed effect coefficients (and standard errors/t-statistics)
coef(summary(mod4.4)) #this reproduces the whole first panel, though methods used above also

```

```

              Estimate Std. Error      df t value    Pr(>|t|)
(Intercept) 12.636193 0.2445047 156.7512 51.68077 2.286893e-100
ses_grpcenter 2.193196 0.1282589 155.2166 17.09976 1.582355e-37

```

```

## Extract the variance components
# Note: in the model display, we see the SDs, not the variance
VarCorr(mod4.4)

```

Groups	Name	Std.Dev.	Corr
id	(Intercept)	2.94636	
	ses_grpcenter	0.83307	0.019
Residual		6.05807	

```

# variance of random effects
(sigma.hat(mod4.4)$sigma$id)^2

```

```

(Intercept) ses_grpcenter
8.6810437   0.6939974

```

```

# NOTE: to extract one or the other, you can use indexing
(sigma.hat(mod4.4)$sigma$id[1])^2 #this is just the intercept random effect

```

```

(Intercept)
8.681044

```

```

# variance of level 1 residual
sigma(mod4.4)^2

```

```
[1] 36.70019
```

34.7 Table 4.5 Intercepts and Slopes as Outcomes Model

```

## Fit the model described
mod4.5 <- lmer(mathach ~ 1 + meanses + sector + ses_grpcenter*(meanses + sector) + ( 1 + ses_
# NOTE: The code above allows the coefficients to appear in the same order as in Table 4.5

```

```

# R automatically includes the main effects, so this model can be written more
# concisely as shown below:
#
# lmer(mathach ~ 1 + ses_grpcenter*(meanses + sector) + ( 1 + ses_grpcenter | id ), data=dat)

# Peek at the results
display(mod4.5)

```

```

lmer(formula = mathach ~ 1 + meanses + sector + ses_grpcenter *
(meanses + sector) + (1 + ses_grpcenter | id), data = dat)
            coef.est  coef.se
(Intercept)    12.10    0.20
meanses        5.33    0.37
sector         1.23    0.31
ses_grpcenter   2.94    0.16
meanses:ses_grpcenter 1.04    0.30
sector:ses_grpcenter -1.64    0.24

```

Error terms:

Groups	Name	Std.Dev.	Corr
id	(Intercept)	1.54	
	ses_grpcenter	0.32	0.39
	Residual	6.06	

number of obs: 7185, groups: id, 160
AIC = 46523.7, DIC = 46489.2
deviance = 46496.4

```

## Extract the fixed effect coefficients (and standard errors/t-statistics)
#this reproduces the whole first panel, though methods used above also work
coef(summary(mod4.5))

```

	Estimate	Std. Error	df	t value	Pr(> t)
(Intercept)	12.095997	0.1987329	159.9143	60.865589	1.625102e-112
meanses	5.332898	0.3691567	150.9836	14.446161	2.944282e-30
sector	1.226453	0.3062674	149.6139	4.004518	9.756638e-05
ses_grpcenter	2.938785	0.1550889	139.2935	18.949039	2.197480e-40
meanses:ses_grpcenter	1.038918	0.2988941	160.5429	3.475873	6.550388e-04
sector:ses_grpcenter	-1.642619	0.2397854	143.3351	-6.850371	2.009491e-10

```

# NOTE: there is a slight discrepancy in the estimate for meances:ses_grpcenter and
# the t-statistics for meances:ses_grpcenter and sector:ses_grpcenter; nothing that
# changes the interpretations, however.

# Testing the need for sector (see page 82)
# (We use a likelihood ratio test with the anova() function)
mod4.5.null <- lmer(mathach ~ 1 + meances + ses_grpcenter*(meances) + (1 + ses_grpcenter | id)
anova( mod4.5, mod4.5.null )

Data: dat
Models:
mod4.5.null: mathach ~ 1 + meances + ses_grpcenter * (meances) + (1 + ses_grpcenter | id)
mod4.5: mathach ~ 1 + meances + sector + ses_grpcenter * (meances + sector) + (1 + ses_grpcenter | id)
      npar   AIC   BIC logLik deviance Chisq Df Pr(>Chisq)
mod4.5.null     8 46568 46623 -23276    46552
mod4.5         10 46516 46585 -23248    46496 55.941  2  7.122e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Testing the need for random slope (see page 84)
# (We use a likelihood ratio test with the anova() function)
mod4.5.null.slope <- lmer(mathach ~ 1 + meances + sector + ses_grpcenter*(meances + sector) + (1 + ses_grpcenter | id)
anova( mod4.5, mod4.5.null.slope )

Data: dat
Models:
mod4.5.null.slope: mathach ~ 1 + meances + sector + ses_grpcenter * (meances + sector) + (1 + ses_grpcenter | id)
mod4.5: mathach ~ 1 + meances + sector + ses_grpcenter * (meances + sector) + (1 + ses_grpcenter | id)
      npar   AIC   BIC logLik deviance Chisq Df Pr(>Chisq)
mod4.5.null.slope     8 46513 46568 -23249    46497
mod4.5          10 46516 46585 -23248    46496 1.0039  2      0.6054

```

34.8 Figure 4.1

NOTE: Figure 4.1 is a graphical display using the results from Model/Table 4.5

The solid line represents the slope of the gamma-01 coefficient; this is the same in public and catholic schools. The dotted lines represent the the slope for individual schools with “prototypical” values of meances (-1,0,1 standard deviations from mean)

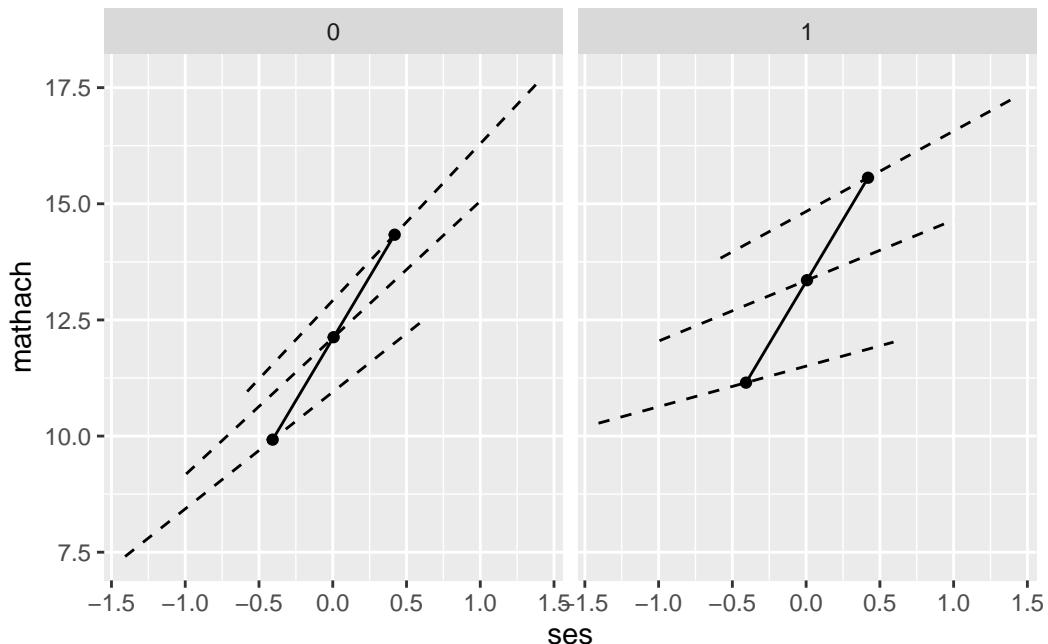
```

# to calculate this, we should note a few values:
avg_meanenses <- mean(dat$meanenses) #average of mean ses var
high_meanenses <- mean(dat$meanenses) + sd(dat$meanenses) # 1 sd above avg meanenses
low_meanenses <- mean(dat$meanenses) - sd(dat$meanenses) # 1 sd below avg meanenses

fake.students = expand.grid( id = -1,
                            meanenses = c( low_meanenses, avg_meanenses, high_meanenses ),
                            sector = c( 0, 1 ),
                            ses_grpcenter = c( -1, 0, 1 ) )
fake.students = mutate( fake.students, ses = meanenses + ses_grpcenter )
fake.students$mathach = predict( mod4.5, newdata=fake.students, allow.new.levels = TRUE )
fake.schools = filter( fake.students, ses_grpcenter == 0 )

ggplot( fake.students, aes( ses, mathach ) ) +
  facet_wrap( ~ sector ) +
  geom_line( aes( group=meanenses ), lty = 2 ) +
  geom_line( data=fake.schools, aes( x = ses, y = mathach ) ) +
  geom_point( data=fake.schools, aes( x = ses, y = mathach ) )

```



34.9 Set-up for remaining tables/figures of chapter

In order to create table 4.6 and the following 2 graphs, we will need to prepare a new dataset. These next lines of code do that.

```
## Start with school level data frame and keep variables interesting to our model comparison
mod.comp <- dplyr::select( sch.dat, id, meanses, sector )

## Add in number of observations per school
n_j <- dat %>% group_by( id ) %>%
  dplyr::summarise(n_j = n())

mod.comp <- merge(mod.comp, n_j, by="id")
head( mod.comp )
```

	id	meanses	sector	n_j
1	1224	-0.428	0	47
2	1288	0.128	0	25
3	1296	-0.420	0	48
4	1308	0.534	1	20
5	1317	0.351	1	48
6	1358	-0.014	0	30

```
## Run site-specific OLS for each school and save estimates

# Calculate global (not group) centered ses
dat$ses_centered <- dat$ses - mean(dat$ses)

# This is the "for loop" method of generating an estimate for each of many small
# worlds (schools). See lecture 2.3 code for the "tidyverse" way.
est.ols <- matrix(nrow=160,ncol=2) #create a matrix to store estimates
se.ols <- matrix(nrow=160,ncol=2) #create matrix to store standard errors

for (i in 1:length(unique(dat$id))){ #looping across the 160 different values of id
  id <- unique(dat$id)[i] #pick the value of id we want
  mod <- lm(mathach ~ 1 + ses_grpcenter, data=dat[dat$id==id,]) #run the model on students
  est.ols[i,] <- coef( mod ) #save the estimates in the matrix we created
  se.ols[i,] <- se.coef( mod ) # and the SEs
}

#convert the matrix to a dataframe and attach the schoolid info
```

```

est.ols <- as.data.frame(est.ols)
est.ols$id <- sch.dat$id
names(est.ols) <- c( 'b0_ols', 'b1_ols', 'id' )

#store standard errors for later
se.ols <- as.data.frame(se.ols)
se.ols$id <- sch.dat$id
names(se.ols) <- c( 'se_b0_ols', 'se_b1_ols', 'id' )

mod.comp <- merge(mod.comp, est.ols, by='id')
mod.comp <- merge(mod.comp, se.ols, by='id' )
head( mod.comp )

```

	id	meanses	sector	n_j	b0_ols	b1_ols	se_b0_ols	se_b1_ols	
1	1224	-0.428		0	47	9.715447	2.5085817	1.0954478	1.765216
2	1288	0.128		0	25	13.510800	3.2554487	1.3637656	2.079675
3	1296	-0.420		0	48	7.635958	1.0759591	0.7740752	1.209016
4	1308	0.534		1	20	16.255500	0.1260242	1.4045813	3.003437
5	1317	0.351		1	48	13.177688	1.2739128	0.7902486	1.435942
6	1358	-0.014		0	30	11.206233	5.0680087	0.8994345	1.391550

We are done running OLS on each of our schools and storing the results.

```

## Extract site-specific coefficients from "unconditional model" (model 4.4)
est4.4 <- coef(mod4.4)$id
names(est4.4) <- c('b0_uncond', 'b1_uncond') #rename
est4.4$id = rownames( est4.4 )

## Extract site-specific coefficients from the "conditional model" (model 4.5)
est4.5 <- coef(mod4.5)$id
head( est4.5 )

```

	(Intercept)	meanses	sector	ses_grpcenter	meanses:ses_grpcenter	
1224	12.02263	5.332898	1.226453		2.933689	1.038918
1288	12.55180	5.332898	1.226453		2.979174	1.038918
1296	10.38509	5.332898	1.226453		2.744066	1.038918
1308	12.12710	5.332898	1.226453		2.923822	1.038918
1317	10.56530	5.332898	1.226453		2.806582	1.038918
1358	11.60500	5.332898	1.226453		2.961265	1.038918
	sector:ses_grpcenter					
1224				-1.642619		

```

1288      -1.642619
1296      -1.642619
1308      -1.642619
1317      -1.642619
1358      -1.642619

```

```

est4.5$id = rownames( est4.5 )

# Now we need to calculate the point estimates using our individual regression equations
# including our level-2 values for each school
# (This is a bit of a pain.)
est4.5 = merge( est4.5, mod.comp, by="id", suffixes = c( "", ".v" ) )
head( est4.5 )

```

	id	(Intercept)	meanses	sector	ses_grpcenter	meanses:ses_grpcenter	sector:ses_grpcenter	meanses.v	sector.v	n_j	b0_ols	b1_ols	se_b0_ols	
1	1224	12.02263	5.332898	1.226453		2.933689				0	47	9.715447	2.5085817	1.0954478
2	1288	12.55180	5.332898	1.226453		2.979174				0	25	13.510800	3.2554487	1.3637656
3	1296	10.38509	5.332898	1.226453		2.744066				0	48	7.635958	1.0759591	0.7740752
4	1308	12.12710	5.332898	1.226453		2.923822				1	20	16.255500	0.1260242	1.4045813
5	1317	10.56530	5.332898	1.226453		2.806582				1	48	13.177688	1.2739128	0.7902486
6	1358	11.60500	5.332898	1.226453		2.961265				0	30	11.206233	5.0680087	0.8994345
		se_b1_ols												
1		1.765216												
2		2.079675												
3		1.209016												
4		3.003437												
5		1.435942												
6		1.391550												

```

est4.5 = mutate( est4.5,
                 b0_cond = `"(Intercept)"` + sector * sector.v + meanses * meanses.v,
                 b1_cond = ses_grpcenter + `sector:ses_grpcenter` * sector.v + `meanses:ses_

```

```

est4.5 = dplyr::select( est4.5, id, b0_cond, b1_cond )

```

```

## Combine the MLM estimates into 1 dataset with ids
est.mlm <- merge( est4.4, est4.5, by="id" )

# Merge all the estimates together by school id
mod.comp <- merge(mod.comp,est.mlm,by = 'id',all=TRUE)

head( mod.comp )

```

	id	meances	sector	n_j	b0_ols	b1_ols	se_b0_ols	se_b1_ols	b0_uncond	
1	1224	-0.428		0	47	9.715447	2.5085817	1.0954478	1.765216	9.956953
2	1288	0.128		0	25	13.510800	3.2554487	1.3637656	2.079675	13.386036
3	1296	-0.420		0	48	7.635958	1.0759591	0.7740752	1.209016	8.039091
4	1308	0.534		1	20	16.255500	0.1260242	1.4045813	3.003437	15.622073
5	1317	0.351		1	48	13.177688	1.2739128	0.7902486	1.435942	13.132771
6	1358	-0.014		0	30	11.206233	5.0680087	0.8994345	1.391550	11.387452
	b1_uncond	b0_cond	b1_cond							
1	2.262837	9.740146	2.489033							
2	2.375964	13.234409	3.112156							
3	1.872247	8.145275	2.307720							
4	2.050193	16.201317	1.835985							
5	1.997129	13.663596	1.528623							
6	2.738390	11.530341	2.946721							

34.10 Table 4.6 Comparing site-specific estimates from different models

```

## Create the list of rows that BER include in the table p. 87
keeprows <- c(4, 15, 17, 22, 27, 53, 69, 75, 81, 90, 135, 153)

## Limit data to the rows of interest, and print the columns in Table 4.6 in the correct order
tab4.6 <- mod.comp[keeprows, c('b0_ols','b1_ols','b0_uncond','b1_uncond','b0_cond','b1_cond')]

## Print Table 4.6 -- the Empirical Bayes from conditional model (b0_cond, b1_cond) are waaaay off
round(tab4.6,2)

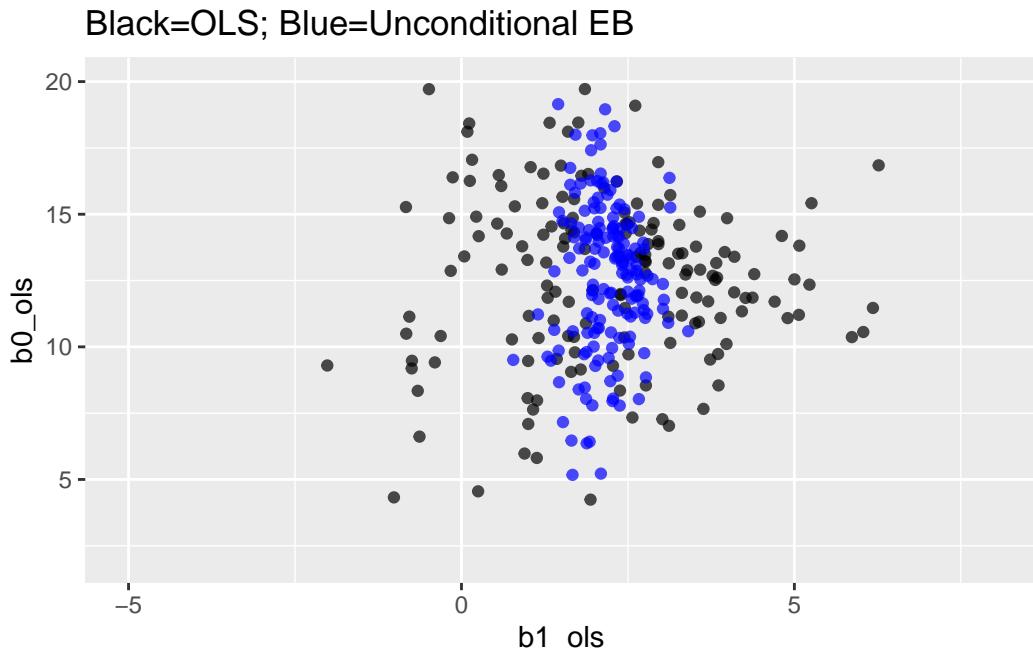
```

	b0_ols	b1_ols	b0_uncond	b1_uncond	b0_cond	b1_cond	n_j	meances	sector
4	16.26	0.13	15.62	2.05	16.20	1.84	20	0.53	1

15	15.98	2.15	15.74	2.19	16.01	1.84	53	0.52	1
17	18.11	0.09	17.41	1.95	17.25	3.71	29	0.69	0
22	11.14	-0.78	11.22	1.15	10.89	0.63	67	-0.62	1
27	13.40	4.10	13.32	2.54	12.95	3.00	38	-0.06	0
53	9.52	3.74	9.76	2.75	9.37	2.42	51	-0.64	0
69	11.47	6.18	11.64	2.72	11.92	3.03	25	0.08	0
75	9.06	1.65	9.28	2.01	9.30	0.67	63	-0.59	1
81	15.42	5.26	15.25	3.14	15.53	1.91	66	0.43	1
90	12.14	1.97	12.18	2.14	12.34	3.03	50	0.19	0
135	4.55	0.25	6.42	1.92	8.55	2.63	14	0.03	0
153	10.28	0.76	10.71	2.06	9.67	2.37	19	-0.59	0

34.11 Figure 4.2 : Scatter plots of the estimates from 2 unconstrained models

```
## Panel (a) and Panel (b) are plotted on the same graph
ggplot(data=mod.comp,aes()) +
  geom_point(aes(x=b1_ols,y=b0_ols),color='black',alpha=0.7) +
  geom_point(aes(x=b1_uncond,y=b0_uncond),color='blue',alpha=0.7) +
  labs(title="Black=OLS; Blue=Unconditional EB") +
  xlim(-5,8) + ylim(2,20)
```



34.12 Figure 4.3 : Scatter plots of residuals from the OLS & Constrained MLM model

```
## Luke: Equation 4.271 and 4.27b (p. 92) are allegedly how we calculate the intercept and s
## But I'm not sure where the estimates for the gamma-hat terms come from; the OLS model only
## individual-level ses
```

```
# trying it here with the predictions from conditional EB
fes = fixef(mod4.5)
fes
```

	meanses	sector
(Intercept)	12.095997	5.332898
ses_grpcenter	2.938785	1.226453
meanses:ses_grpcenter	1.038918	-1.642619

```
mod.comp = mutate(mod.comp,
                  u0_ols = b0_ols - (fes[1] + fes[2]*meanses + fes[3]*sector),
                  u1_ols = b1_ols - (fes[4] + fes[5]*meanses + fes[6]*sector))
```

```

## Panel (a) and (b) plotted on same graph

mod.comp = mutate( mod.comp,
                    u0_cond = b0_cond - (fes[1] + fes[2]*meanses + fes[3]*sector),
                    u1_cond = b1_cond - (fes[4] + fes[5]*meanses + fes[6]*sector) )

head( mod.comp )

```

	id	meanses	sector	n_j	b0_ols	b1_ols	se_b0_ols	se_b1_ols	b0_uncond	b1_uncond	b0_cond	b1_cond	u0_ols	u1_ols	u0_cond	u1_cond	
1	1224	-0.428		0	47	9.715447	2.5085817	1.0954478	1.765216	9.956953	2.262837	9.740146	2.489033	-0.09807014	0.01445354	-0.07337107	-0.005095579
2	1288	0.128		0	25	13.510800	3.2554487	1.3637656	2.079675	13.386036	2.375964	13.234409	3.112156	0.73219201	0.18368221	0.45580075	0.040389349
3	1296	-0.420		0	48	7.635958	1.0759591	0.7740752	1.209016	8.039091	1.872247	8.145275	2.307720	-2.22022179	-1.42648036	-1.71090544	-0.194719194
4	1308	0.534		1	20	16.255500	0.1260242	1.4045813	3.003437	15.622073	2.050193	16.201317	1.835985	0.08528223	-1.72492410	0.03109939	-0.014963432
5	1317	0.351		1	48	13.177688	1.2739128	0.7902486	1.435942	13.132771	1.997129	13.663596	1.528623	-2.01661001	-0.38691354	-1.53070178	-0.132203128
6	1358	-0.014		0	30	11.206233	5.0680087	0.8994345	1.391550	11.387452	2.738390	11.530341	2.946721	-0.81510320	2.14376861	-0.49099553	0.022480378

```

nrow( mod.comp )

```

```

[1] 160

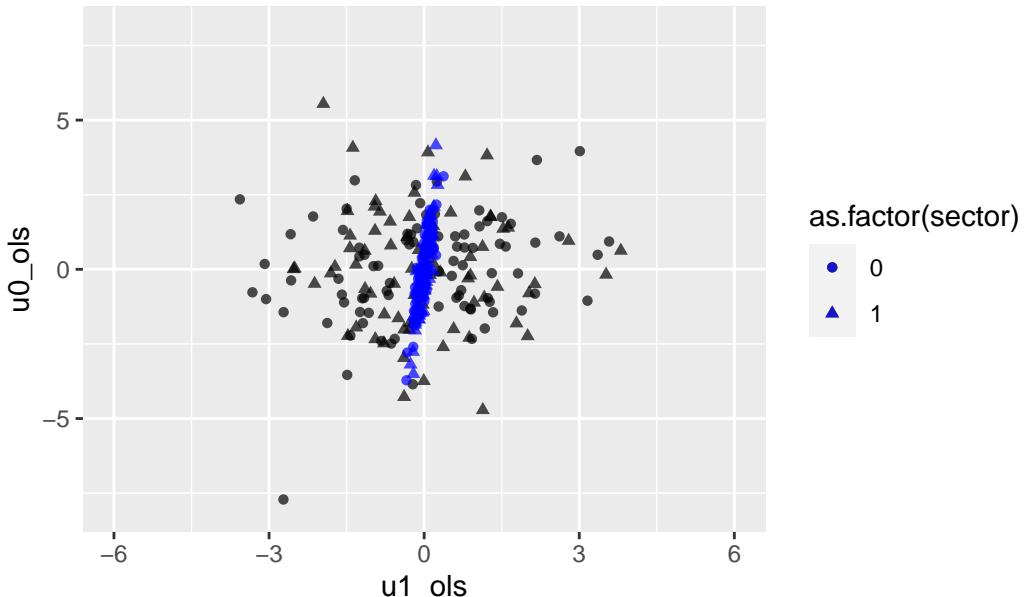
```

```

ggplot(data=mod.comp, aes( pch=as.factor(sector)) ) +
  geom_point(aes(x=u1_ols, y=u0_ols),color='black', alpha=0.7) +
  geom_point(aes(x=u1_cond, y=u0_cond),color='blue', alpha=0.7) +
  labs(title = "Black: OLS, Blue: Conditional EB") +
  xlim(-6,6) + ylim(-8,8)

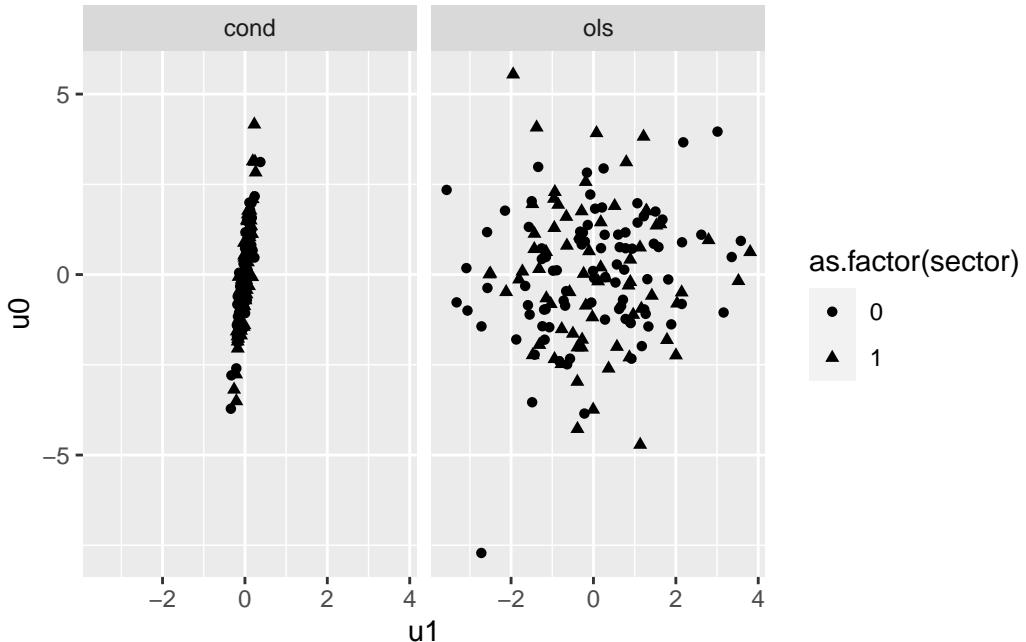
```

Black: OLS, Blue: Conditional EB



```
# To get in two-panel format we need to get our data to long format
mod.comp.ols = data.frame( sector = mod.comp$sector,
                           u0 = mod.comp$u0_ols,
                           u1 = mod.comp$u1_ols )
mod.comp.EB = data.frame( sector = mod.comp$sector,
                           u0 = mod.comp$u0_cond,
                           u1 = mod.comp$u1_cond )
mod.comp.l = bind_rows( ols=mod.comp.ols, cond = mod.comp.EB, .id = "method" )

ggplot(data=mod.comp.l, aes( u1, u0, pch=as.factor(sector)) ) +
  facet_wrap( ~ method ) +
  geom_point()
```



34.13 Table 4.7 : pg 94

```

# This section is not very good--I would skip.
# Generating confidence intervals for individual random intercepts and slopes is a weird bus

# OLS First:

# Doing it by fitting OLS on our subset
sch.2305 = filter( dat, id == 2305 )
head( sch.2305 )

# A tibble: 6 x 13
# Groups:   id [1]
  id  minority female    ses mathach  size sector pracad disclim himinty
  <chr>    <dbl> <dbl>  <dbl>   <dbl>  <dbl>   <dbl>   <dbl>    <dbl>    <dbl>
1 2305      1     1 -0.738   16.4    485     1    0.69   -1.38     1
2 2305      1     1 -1.18    12.8    485     1    0.69   -1.38     1
3 2305      1     1 -0.308   15.3    485     1    0.69   -1.38     1
4 2305      1     1 -0.358   12.7    485     1    0.69   -1.38     1
5 2305      1     1 -1.52    10.2    485     1    0.69   -1.38     1

```

```
6 2305      1      1 -0.518    8.94    485      1  0.69   -1.38      1  
# i 3 more variables: meanses <dbl>, ses_grpcenter <dbl>, ses_centered <dbl>
```

```
M.2305 = lm( mathach ~ ses_grpcenter, data=sch.2305 )  
M.2305
```

Call:

```
lm(formula = mathach ~ ses_grpcenter, data = sch.2305)
```

Coefficients:

(Intercept)	ses_grpcenter
11.1378	-0.7821

```
confint( M.2305 )
```

	2.5 %	97.5 %
(Intercept)	9.911824	12.363698
ses_grpcenter	-2.665989	1.101767

```
sch.8367 = filter( dat, id == 8367 )  
head( sch.8367 )
```

```
# A tibble: 6 x 13  
# Groups: id [1]  
  id minority female    ses mathach size sector pracad disclim himinty  
  <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
1 8367      0     0 -0.228  15.9    153     0     0     1.79     0  
2 8367      0     1 -0.208  1.86    153     0     0     1.79     0  
3 8367      1     0  0.532  -2.83   153     0     0     1.79     0  
4 8367      0     1  0.662   2.12   153     0     0     1.79     0  
5 8367      0     0 -0.228   6.76   153     0     0     1.79     0  
6 8367      0     0 -1.08   0.725  153     0     0     1.79     0  
# i 3 more variables: meanses <dbl>, ses_grpcenter <dbl>, ses_centered <dbl>
```

```
M.8367 = lm( mathach ~ ses_grpcenter, data=sch.8367 )  
M.8367
```

Call:

```
lm(formula = mathach ~ ses_grpcenter, data = sch.8367)
```

Coefficients:

	(Intercept)	ses_grpcenter
	4.5528	0.2504

```
confint( M.8367 )
```

	2.5 %	97.5 %
(Intercept)	1.872974	7.232598
ses_grpcenter	-3.431096	3.931845

```
# Use SE from earlier to get confint
est4.7 <- mod.comp[c(22,135),]
est4.7
```

	id	meanses	sector	n_j	b0_ols	b1_ols	se_b0_ols	se_b1_ols	b0_uncond
22	2305	-0.622	1	67	11.137761	-0.7821112	0.6138468	0.943289	11.222551
135	8367	0.032	0	14	4.552786	0.2503748	1.2299413	1.689668	6.423938
		b1_uncond	b0_cond	b1_cond	u0_ols	u1_ols	u0_cond	u1_cond	
22		1.149555	10.886600	0.6276417	1.132373	-1.432070	0.8812116	-0.02231763	
135		1.924903	8.549007	2.6307047	-7.713864	-2.721656	-3.7176433	-0.34132569	

```
# CI for intercept and slope using our normal and stored SEs.
# (Not taking t distribution into account changes things, as does not
# taking the uncertainty in the fixed effects for the EB CIs. So this is
# very approximate.)
se_uncond = as.data.frame( se.coef(mod4.4)$id )
head( se_uncond )
```

	(Intercept)	ses_grpcenter
1224	0.8464092	0.7189592
1288	1.1205609	0.7593421
1296	0.8382669	0.7111100
1308	1.2307722	0.8005012
1317	0.8382675	0.7377054
1358	1.0354852	0.7489241

```

names( se_uncond ) = c("se_b0_uncond","se_b1_uncond" )
se_cond = as.data.frame( se.coef( mod4.5 )$id )
names( se_cond ) = c("se_b0_cond","se_b1_cond" )
head( se_cond )

```

	se_b0_cond	se_b1_cond
1224	0.7662313	0.2929654
1288	0.9521965	0.2988437
1296	0.7601221	0.2923332
1308	1.0176181	0.3025414
1317	0.7603073	0.2940970
1358	0.8982481	0.2971694

```

se_uncond$id = rownames( se_uncond )
se_cond$id = rownames( se_cond )
est4.7 = merge( est4.7, se_uncond, by="id" )
est4.7 = merge( est4.7, se_cond, by="id" )

est4.7.int = mutate( est4.7,
  CI.low.ols = b0_ols + - 1.96 * se_b0_ols,
  CI.high.ols = b0_ols + 1.96 * se_b0_ols,
  CI.low.uncond = b0_uncond + - 1.96 * se_b0_uncond,
  CI.high.uncond = b0_uncond + 1.96 * se_b0_uncond,
  CI.low.cond = b0_cond + - 1.96 * se_b0_cond,
  CI.high.cond = b0_cond + 1.96 * se_b0_cond )

dplyr::select( est4.7.int, starts_with("CI" ) )

```

	CI.low.ols	CI.high.ols	CI.low.uncond	CI.high.uncond	CI.low.cond	CI.high.cond
1	9.934621	12.340901	9.815648	12.629455	9.579800	12.19340
2	2.142101	6.963471	3.642797	9.205078	6.361492	10.73652

```

est4.7.slope = mutate( est4.7,
  CI.low.ols = b1_ols + - 1.96 * se_b1_ols,
  CI.high.ols = b1_ols + 1.96 * se_b1_ols,
  CI.low.uncond = b1_uncond + - 1.96 * se_b1_uncond,
  CI.high.uncond = b1_uncond + 1.96 * se_b1_uncond,
  CI.low.cond = b1_cond + - 1.96 * se_b1_cond,
  CI.high.cond = b1_cond + 1.96 * se_b1_cond )

dplyr::select( est4.7.slope, starts_with("CI" ) )

```

	CI.low.ols	CI.high.ols	CI.low.uncond	CI.high.uncond	CI.low.cond	CI.high.cond
1	-2.630958	1.066735	-0.1675367	2.466647	0.0629627	1.192321
2	-3.061375	3.562124	0.3960110	3.453795	2.0356645	3.225745

35 Code for Faraway Example

This handout gives the code for the longitudinal data example from Faraway book chapter 9 (see iPac on Canvas). See that chapter to get explanations, etc., or just run code line by line to see what you get! Note: this code uses ggplot. Book uses another plotting package called `lattice`; don't bother with `lattice`.

35.1 R Setup

```
library( arm )
library( ggplot2 )
library( plyr )

# Install package from textbook to get the data by
# running this line once.
#install.packages( "faraway" )
```

35.2 First Example

```
# load the data
library(faraway)
data(psid)
head(psid)
```

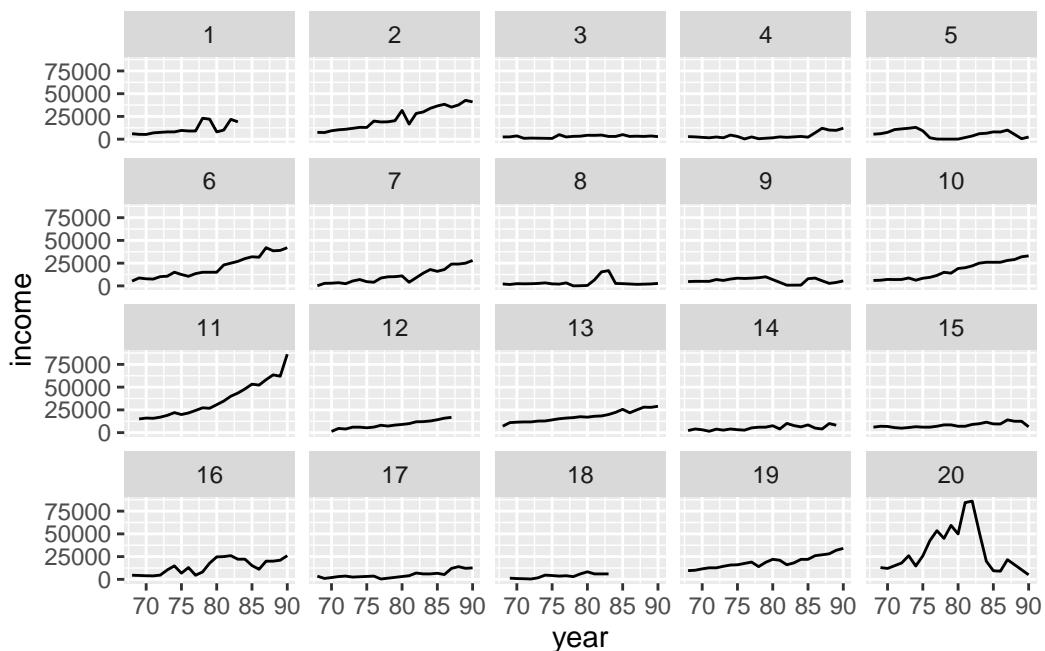
	age	educ	sex	income	year	person
1	31	12	M	6000	68	1
2	31	12	M	5300	69	1
3	31	12	M	5200	70	1
4	31	12	M	6900	71	1
5	31	12	M	7500	72	1
6	31	12	M	8000	73	1

```

# Make log-transform of income
psid$log_income = with( psid, log( income + 100 ) )

# Look at some plots
psid.sub = subset( psid, person < 21 )
ggplot( data=psid.sub, aes( x=year, y=income ) ) +
  facet_wrap( ~ person ) +
  geom_line()

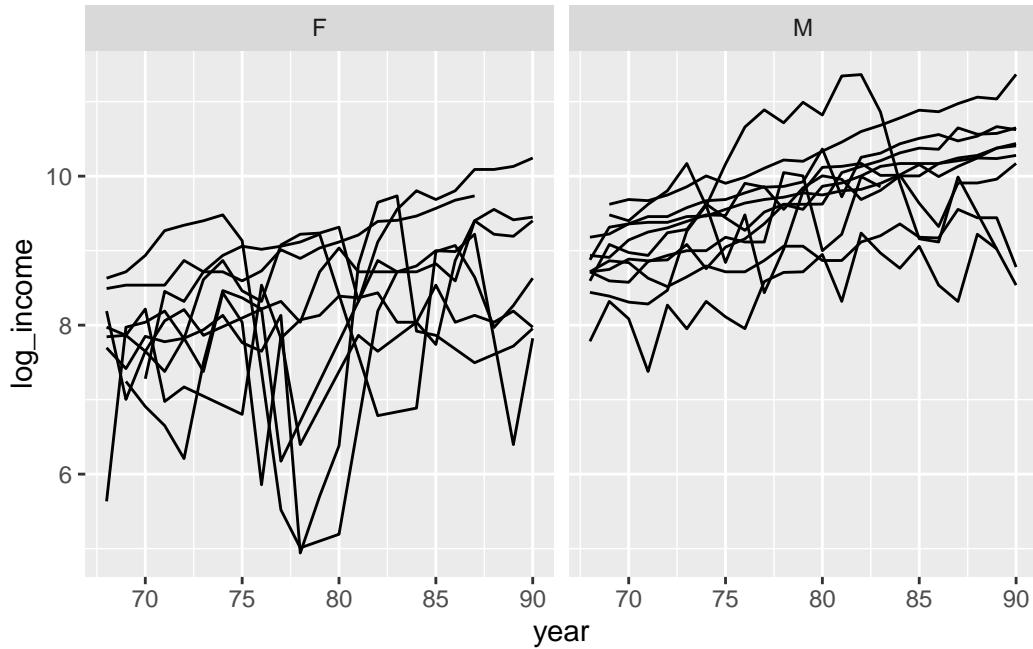
```



```

ggplot( data=psid.sub, aes( x=year, y=log_income, group=person ) ) +
  facet_wrap( ~ sex ) +
  geom_line()

```



```
# Simple regression on a single person
lmod <- lm( log_income ~ I(year-78), subset=(person==1), psid)
coef(lmod)
```

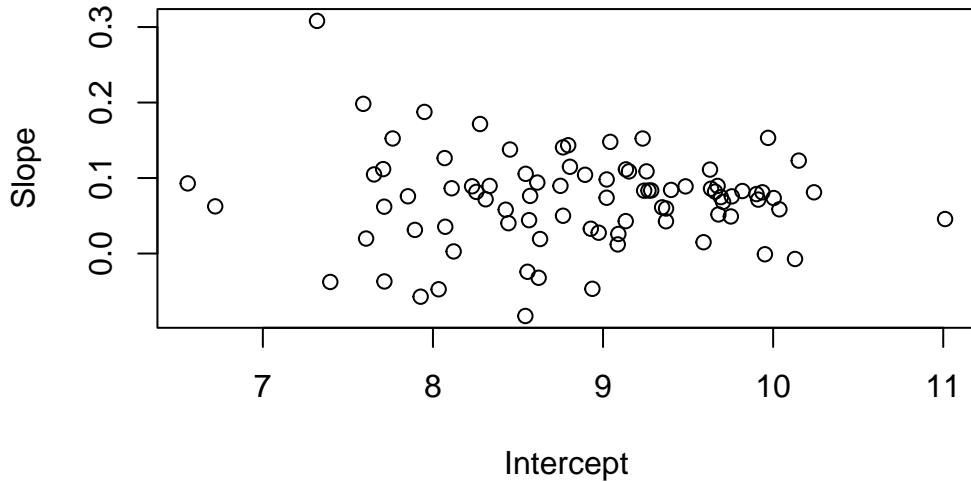
```
(Intercept) I(year - 78)
9.40910950 0.08342068
```

```
# Now do linear regression on everyone
sum.stat = ddply( psid, .(person), function( dat ) {
  lmod <- lm(log(income) ~ I(year-78), data=dat )
  cc = coef(lmod)
  names(cc) = c("intercept","slope")
  c( cc, sex=dat$sex[[1]] )
} )
head( sum.stat )
```

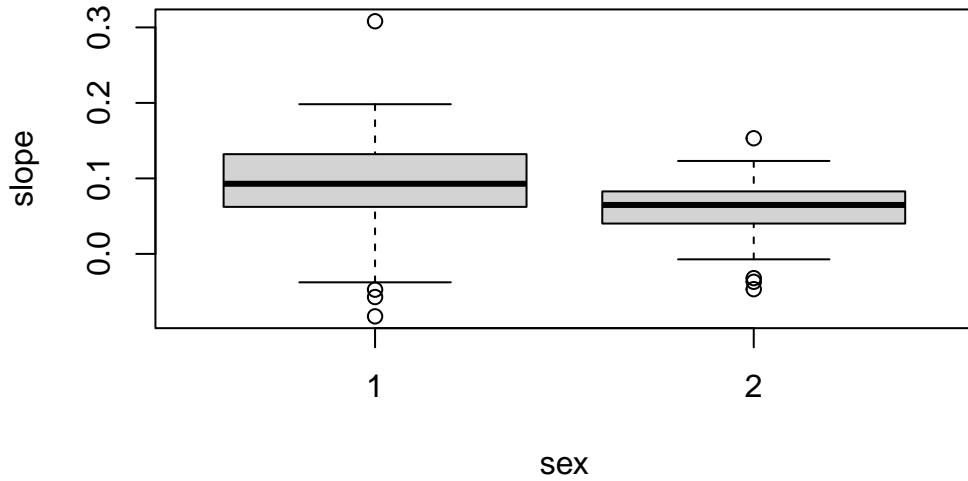
	person	intercept	slope	sex
1	1	9.399957	0.08426670	2
2	2	9.819091	0.08281031	2
3	3	7.893863	0.03131149	1
4	4	7.853027	0.07585135	1

```
5      5 8.033453 -0.04738677   1
6      6 9.673443  0.08953380   2
```

```
plot( slope ~ intercept, data=sum.stat, xlab="Intercept", ylab="Slope")
```



```
boxplot( slope ~ sex, data=sum.stat )
```



```
# Is rate of income growth different by sex?
t.test( slope ~ sex, data=sum.stat )
```

Welch Two Sample t-test

```
data: slope by sex
t = 2.3786, df = 56.736, p-value = 0.02077
alternative hypothesis: true difference in means between group 1 and group 2 is not equal to
95 percent confidence interval:
0.00507729 0.05916871
sample estimates:
mean in group 1 mean in group 2
0.08903346     0.05691046
```

```
# Is initial income different by sex?
t.test( intercept ~ sex, data=sum.stat )
```

Welch Two Sample t-test

```

data: intercept by sex
t = -8.2199, df = 79.719, p-value = 3.065e-12
alternative hypothesis: true difference in means between group 1 and group 2 is not equal to
95 percent confidence interval:
-1.4322218 -0.8738792
sample estimates:
mean in group 1 mean in group 2
8.229275      9.382325

```

35.3 Fitting the model

```

# Fitting our model
library(lme4)
psid$cyear <- psid$year-78
mmod <- lmer(log(income) ~ cyear*sex + age + educ + (cyear|person), psid)
display(mmod)

```

```

lmer(formula = log(income) ~ cyear * sex + age + educ + (cyear |
  person), data = psid)
  coef.est coef.se
(Intercept) 6.67     0.54
cyear        0.09     0.01
sexM         1.15     0.12
age          0.01     0.01
educ         0.10     0.02
cyear:sexM  -0.03     0.01

```

Error terms:

Groups	Name	Std.Dev.	Corr
person	(Intercept)	0.53	
	cyear	0.05	0.19
	Residual	0.68	

number of obs: 1661, groups: person, 85
AIC = 3839.8, DIC = 3751.2
deviance = 3785.5

```

# refit with the lmerTest library to get p-values
library( lmerTest )

```

```
mmod <- lmer(log(income) ~ cyear*sex + age + educ + (cyear|person), psid)
summary(mmod)
```

Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]

Formula: log(income) ~ cyear * sex + age + educ + (cyear | person)

Data: psid

REML criterion at convergence: 3819.8

Scaled residuals:

Min	1Q	Median	3Q	Max
-10.2310	-0.2134	0.0795	0.4147	2.8254

Random effects:

Groups	Name	Variance	Std.Dev.	Corr
person	(Intercept)	0.2817	0.53071	
	cyear	0.0024	0.04899	0.19
Residual		0.4673	0.68357	

Number of obs: 1661, groups: person, 85

Fixed effects:

	Estimate	Std. Error	df	t value	Pr(> t)
(Intercept)	6.674211	0.543323	81.176969	12.284	< 2e-16 ***
cyear	0.085312	0.008999	78.915123	9.480	1.14e-14 ***
sexM	1.150312	0.121292	81.772542	9.484	8.06e-15 ***
age	0.010932	0.013524	80.837433	0.808	0.4213
educ	0.104209	0.021437	80.722317	4.861	5.65e-06 ***
cyear:sexM	-0.026306	0.012238	77.995359	-2.150	0.0347 *

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:

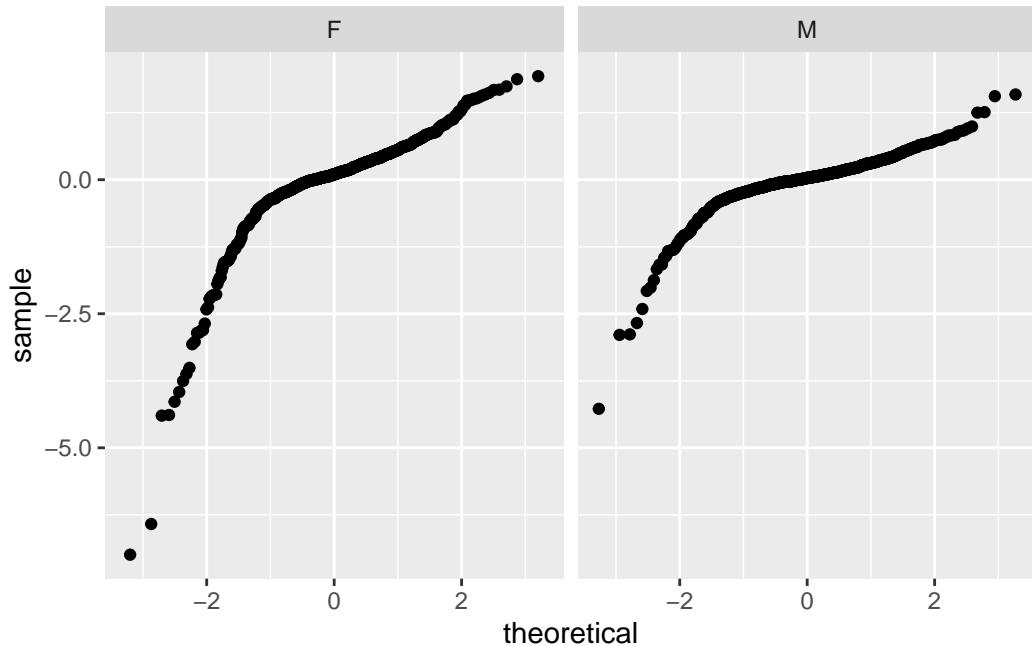
	(Intr)	cyear	sexM	age	educ
cyear	0.020				
sexM	-0.104	-0.098			
age	-0.874	0.002	-0.026		
educ	-0.597	0.000	0.008	0.167	
cyear:sexM	-0.003	-0.735	0.156	-0.010	-0.011

35.4 Model Diagnostics

```
# First add our residuals and fitted values to our original data
# (We can do this since we have no missing data so the rows will line up
# correctly)
psid = transform( psid, resid=resid( mmod ),
                  fit = fitted( mmod ) )
head( psid )
```

	age	educ	sex	income	year	person	log_income	cyear	resid	fit
1	31	12	M	6000	68	1	8.716044	-10	0.06719915	8.632316
2	31	12	M	5300	69	1	8.594154	-9	-0.13201639	8.707478
3	31	12	M	5200	70	1	8.575462	-8	-0.22622748	8.782641
4	31	12	M	6900	71	1	8.853665	-7	-0.01852759	8.857804
5	31	12	M	7500	72	1	8.935904	-6	-0.01030887	8.932967
6	31	12	M	8000	73	1	8.999619	-5	-0.02093325	9.008130

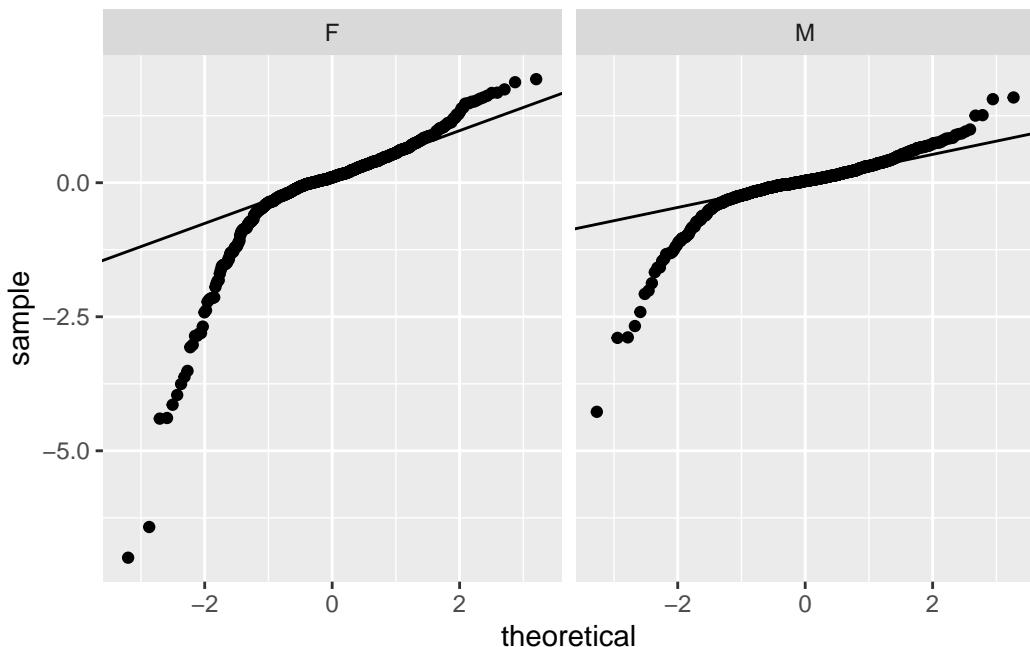
```
# Here is a qqplot for each sex
ggplot( data=psid ) +
  facet_wrap( ~ sex ) +
  stat_qq( aes( sample=resid ) )
```



```
# If you want to add the lines, you have to do a little more work
slopes = ddply( psid, .(sex), function( dat ) {
  y <- quantile(dat$resid, c(0.25, 0.75))
  x <- qnorm(c(0.25, 0.75))
  slope <- as.numeric( diff(y)/diff(x) )
  int <- y[[1]] - slope * x[[1]]
  c( slope=slope, int=int )
} )
slopes
```

	sex	slope	int
1	F	0.4324568	0.10579138
2	M	0.2473357	0.03321435

```
ggplot( data=psid ) +
  facet_wrap( ~ sex ) +
  stat_qq( aes( sample=resid ) ) +
  geom_abline( data=slopes, aes( slope=slope, intercept=int ) )
```

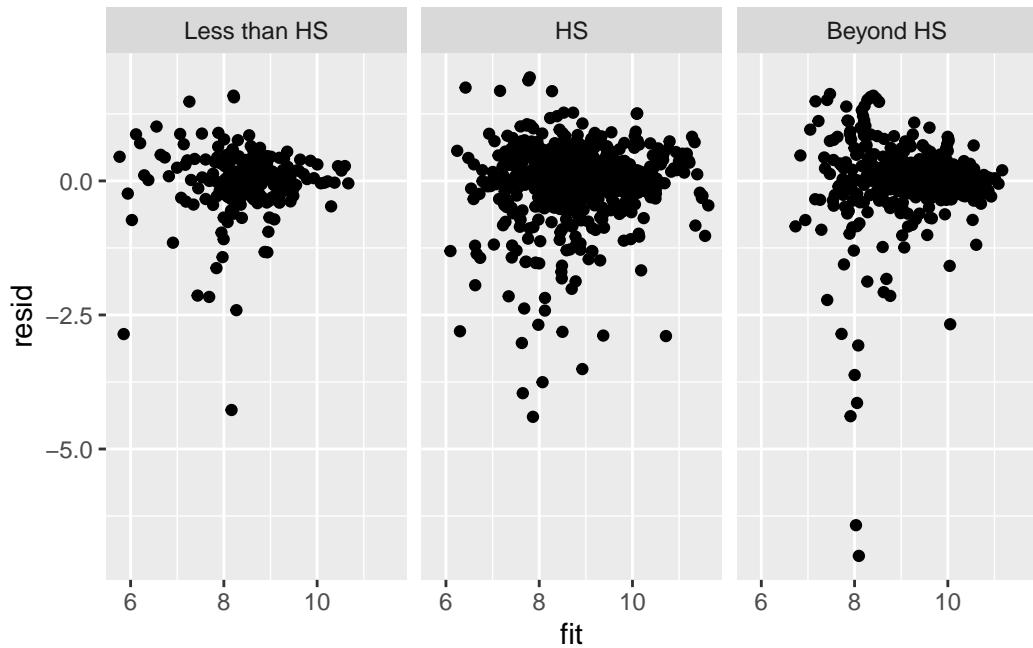


And a residual plot

```

psid$educ_levels = cut(psid$educ, c(0,8.5,12.5,20), labels=c( "Less than HS", "HS", "Beyond HS"))
ggplot( data=psid, aes( x=fit, y=resid ) ) +
  facet_wrap( ~ educ_levels ) +
  geom_point()

```



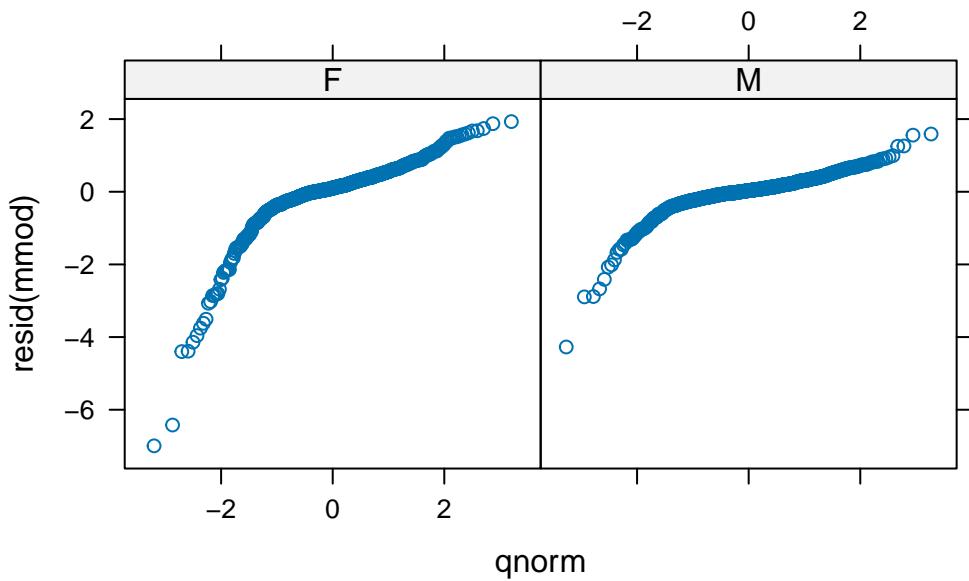
35.4.1 Lattice code

For reference, we can also do this:

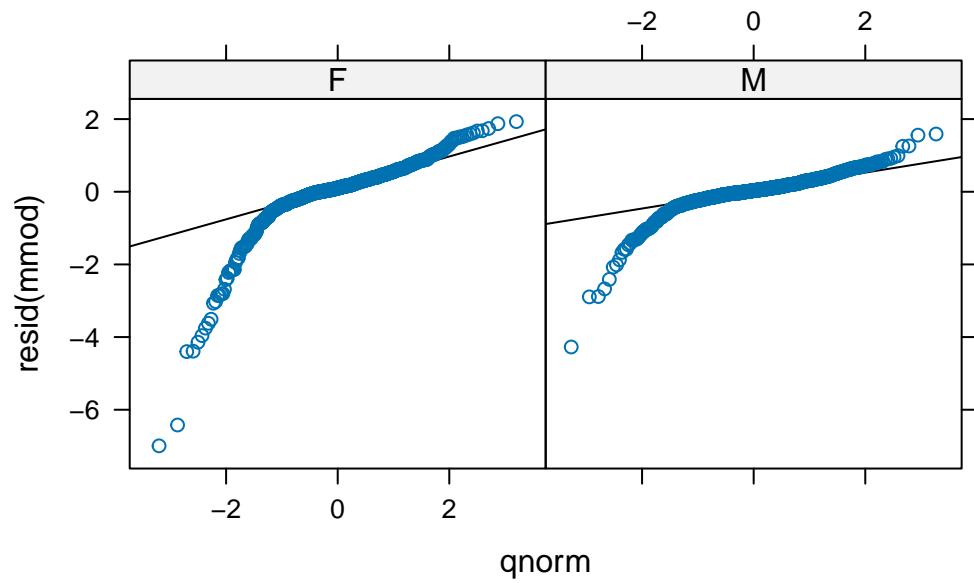
```

# This is doing it from the lattice package
library( lattice )
qqmath(~resid(mmod) | sex, psid)

```



```
# fancier with some lines.  The points should lie on the line
# if we have normal residuals.  (We don't.)
qqmath(~ resid(mmod) | sex, data = psid,
      panel = function(x, ...) {
        panel.qqmathline(x, ...)
        panel.qqmath(x, ...)
      })
```



36 Example of a three-level model of clustered data

This will illustrate fitting a three level model (with clusters inside of clusters) and extracting the various components from it. This is a rough document, but hopefully will be useful.

36.1 Load the data

We first load the data. This is a dataset extensively discussed in Rabe-Hesketh and Skrondal. I am replicating the model they propose in chapter 8.4. This story is as follows: the data set is a collection of measurements for a test-retest of two peak expiratory flow measurement devices (in English, patients were told to exhale into a device to measure their lung capacity, and they did so twice for two different measurement devices). We want to understand whether the types of meter are different, and also understand variation in subjects lung capacities, and variation in the measurement error of the meters.

In the following we load the data and look at the first few lines. We see that each subject had two measurements from the standard and the mini Wright flow meter.

```
pefr = read.dta( "data/pefr.dta" )  
  
head( pefr )
```

```
id wp1 wp2 wm1 wm2  
1 1 494 490 512 525  
2 2 395 397 430 415  
3 3 516 512 520 508  
4 4 434 401 428 444  
5 5 476 470 500 500  
6 6 557 611 600 625
```

We are going to view this as three-level data. We have multiple measurements nested inside device type nested inside subject. We might imagine that different subjects have different lung capacities. We also might imagine that different subjects are going to have different

biases when using the two different meters. The two observations for each meter allows us to understand the variability of measurements for a single meter for a given subject, and looking at how these vary across subjects allows us to understand how much the biases move across individuals.

36.2 Reshape the data (Optional section)

This section illustrates some advanced reshaping techniques. In particular we reshape the data twice to deal with the time and the device as different levels.

Here we go:

```
dat = reshape( pefr, direction="long", idvar = "id",
               varying=list( c("wp1","wm1"), c("wp2","wm2") ),
               times=c("wp","wm"),
               timevar="device",
               v.names=c("time1","time2") )
```

Let's see what we got:

```
head( dat )
```

	<code>id</code>	<code>device</code>	<code>time1</code>	<code>time2</code>
1.wp	1	wp	494	490
2.wp	2	wp	395	397
3.wp	3	wp	516	512
4.wp	4	wp	434	401
5.wp	5	wp	476	470
6.wp	6	wp	557	611

```
subset( dat, id==1 )
```

	<code>id</code>	<code>device</code>	<code>time1</code>	<code>time2</code>
1.wp	1	wp	494	490
1.wm	1	wm	512	525

The second line above shows us our first person now as two new lines, one for each device. We see the measurements correspond to the first row of the original `pefr` data.

Now we have a row for each person for each device. Next we unstack the time (and then look at what we got):

```

dat = reshape( dat, direction="long", idvar=c("id","device"),
               varying=list( c("time1","time2") ),
               v.names=c("flow") )
head( dat )

```

	id	device	time	flow
1.wp.1	1	wp	1	494
2.wp.1	2	wp	1	395
3.wp.1	3	wp	1	516
4.wp.1	4	wp	1	434
5.wp.1	5	wp	1	476
6.wp.1	6	wp	1	557

We look at our second person to see if the measurements have the appropriate labels. They do.

```
subset( dat, id==2 )
```

	id	device	time	flow
2.wp.1	2	wp	1	395
2.wm.1	2	wm	1	430
2.wp.2	2	wp	2	397
2.wm.2	2	wm	2	415

```
subset( pefr, id==2 )
```

	id	wp1	wp2	wm1	wm2
2	2	395	397	430	415

Another sanity check:

```
table( dat$id )
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

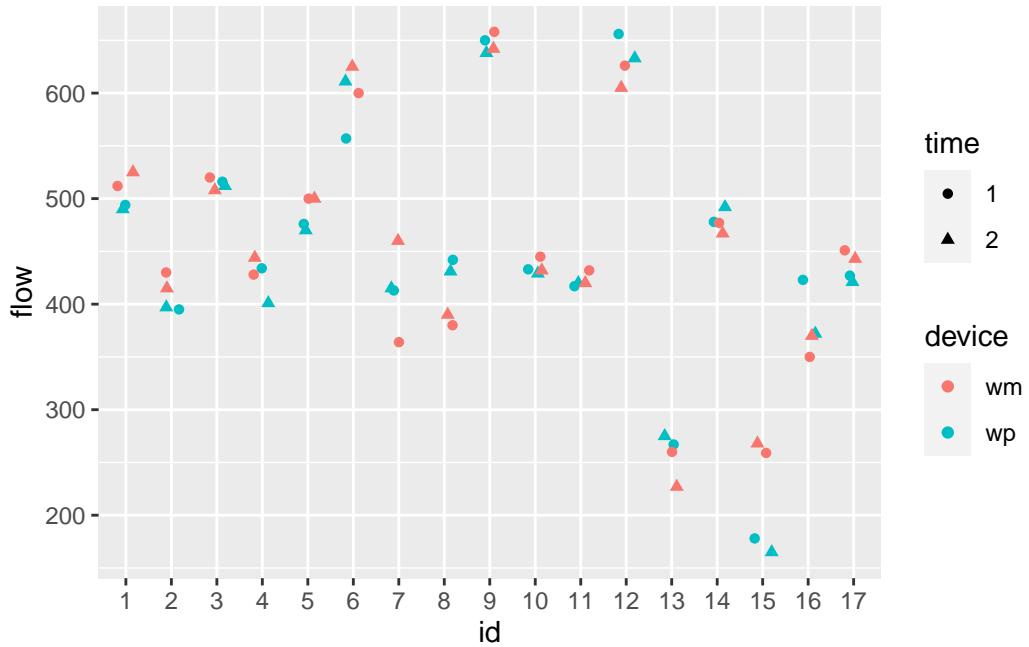
We have four measurements, still, for each person.

When reshaping data, one typically has to fiddle with all of the commands and check the results a few times to get it right.

36.3 Plot the data

We can look at the data. The following illustrates getting different colors and symbols depending on covariate information:

```
dat$id = as.factor( dat$id )
dat$device = as.factor( dat$device )
dat$time = as.factor( dat$time )
ggplot( data=dat, aes( x=id, y=flow, col=device, pch=time ) ) +
  geom_jitter( width=0.2, height=0 )
```



36.4 The mathematical model

Level 1: We have for individual i using machine j at time t : $[Y_{ijt} = \beta_0 + \beta_1 t + \gamma_{ijt}]$ The β_1 allows for a time effect of the second measurement being systematically lower or higher than the first. We pool this across all subjects and machines.

Level 2: Our machine-level intercepts for each subject are $[\beta_0 = \alpha_0 + \alpha_1 D_j + u_{ij}]$ with $D_j = 1\{j = wp\}$ being an indicator (dummy variable) for the second machine. The α_1 allows a systematic bias for the two machines (so the wp machine could tend to give larger readings than the wm machine, for example). Overall, the above says each machine expected

reading varies around the subject's lung capacity, but that these expected readings will vary around the subjects true capacity by the u_{ij} . Actual readings for subject i on machine j will hover around β_{ij} if we had the subject test over and over, according to our model (not including fatigue captured by the time coefficient).

Level 3: Finally our subject intercepts are [$*0i = * + w\{i\} .$] The overall population lung capacity is μ . Subjects have larger or smaller lung capacity depending on their w_i .

The u_{ij} and w_i are each normally distributed, and independent from each other.

The w_i are how the subjects vary (i.e., their different lung capacities). The u_{ij} are the individual biases of a machine for a given subject. Looking at our plot, we see that subjects vary a lot, and machines vary sometimes within a subject (the centers of the pairs of colored points tend to be close, but not always), and the residual variance tends to be small (colored points are close together).

36.5 Fit the model

```
library( lme4 )
M1 = lmer( flow ~ device + time + (1|id) + (1|device:id), data=dat )
display( M1 )
```

```
lmer(formula = flow ~ device + time + (1 | id) + (1 | device:id),
      data = dat)
      coef.est coef.se
(Intercept) 454.43    27.84
devicewp     -6.03     8.05
time2        -1.03     4.37

Error terms:
 Groups     Name        Std.Dev.
 device:id (Intercept) 19.72
 id         (Intercept) 111.99
 Residual            18.01
 ---
 number of obs: 68, groups: device:id, 34; id, 17
 AIC = 682.8, DIC = 709.1
 deviance = 689.9
```

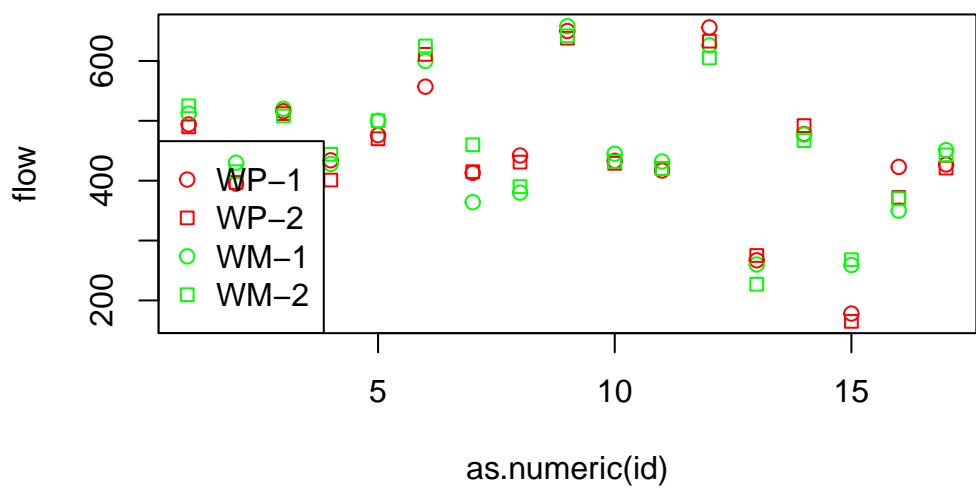
Now let's connect some pieces:

- The main effects estimate $\mu = 455.46$ and $\gamma_1 = -6.03$ and $\beta_1 = -1.03$.
- The z-score of $z = -6.03/8.05 < 1$ means there is no evidence of systematic bias of one machine compared to the other.
- The estimated standard deviation of actual lung capacity is 112.
- The estimated standard deviation of how two different machines will measure the same person is 19.72. Different machines will tend to give different average measurements for the same subject.
- The estimated standard deviation of how much a repeated measurement of the same machine on the same person will vary is 18. The machines are relatively precise, given the variation in the population.
- The amount of variance explained by lung variation is $112^2/(19.72^2 + 111.99^2 + 18.01^2) = 0.94636$, i.e., most of it.

36.6 Appendix (Optional): base plot package

Here is how to build the plot without ggplot:

```
plot( flow ~ as.numeric(id), data=dat, pch=ifelse( time==1, 21, 22 ),
      col=ifelse( device=="wp", "red", "green" ) )
legend( "bottomleft", legend=c("WP-1", "WP-2", "WM-1", "WM-2"),
        pch=c(21,22,21,22),
        col=c("red", "red", "green", "green") )
```



37 Example of a three-level longitudinal model

This will illustrate fitting a three level model (where we have time variation and then clusters) and extracting the various components from it. This is a rough document, but hopefully will be useful. This example is based on a dataset used in Rabe-Hesketh and Skrondal, chapter 8.10, but you don't really need that text.

37.1 Load the data

We first load the data. Shoving a lot of things under the rug, we have five measurements on a collection of kids in Kenya across time. We are interested in the impact of improved nutrition. The children are clustered in schools. This gives a three-level structure. The schools were treated with different nutrition programs.

In the following we load the data and look at the first few lines. Lots of variables! The main ones are id (the identifier of the kid), treatment (the kind of treatment given to the school), schoolid (the identifier of the school), gender (the gender of the kid), and rn (the time variable). Our outcome is ravens (Raven's colored progressive matrices assessment).

```
kenya = read.dta( "data/kenya.dta" )  
  
# look at first 9 variables  
head( kenya[1:9], 3 )
```

	id	schoolid	rn	relyear	ravens	arithmetic	vmeaning	dstotal	age_at_time0
1	1	2	1	-0.15	15	5	25	6	7.19
2	1	2	2	0.14	19	7	39	8	7.19
3	1	2	3	0.46	21	7	33	7	7.19

```
# what times do we have?  
table( kenya$rn ) #time
```

1	2	3	4	5
546	546	546	546	546

```

length( unique( kenya$id ) )
[1] 546

length( unique( kenya$schoolid ) )
[1] 12

```

We see we have 546 kids and 12 schools.

37.2 Plot the data

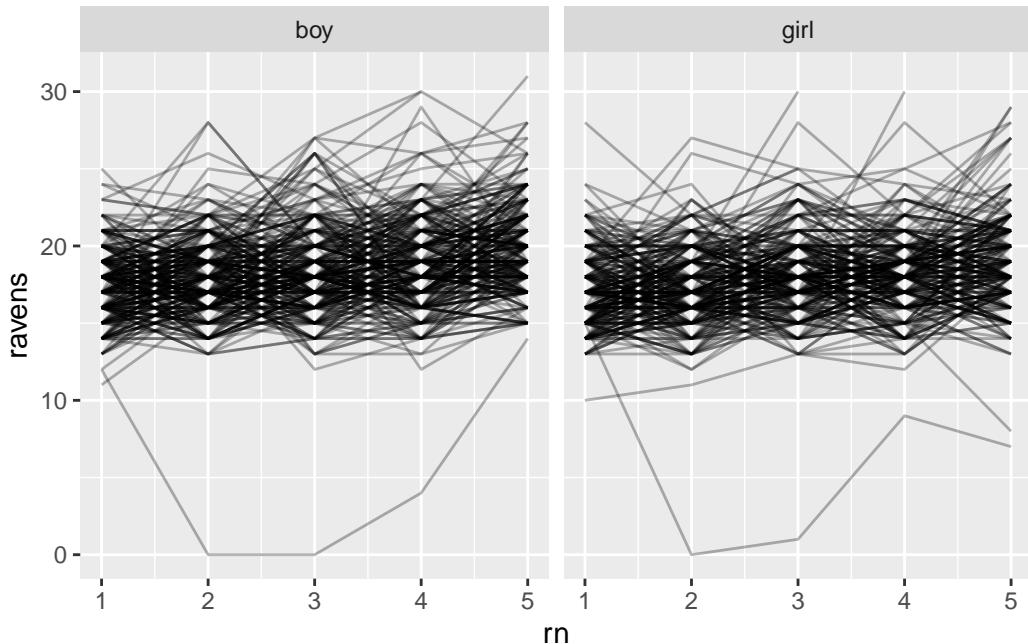
We can look at the data.

```

ggplot( data=kenya, aes( x=rn, y=ravens, group=id ) )+
  facet_wrap( ~ gender ) +
  geom_line( alpha=0.3 )

```

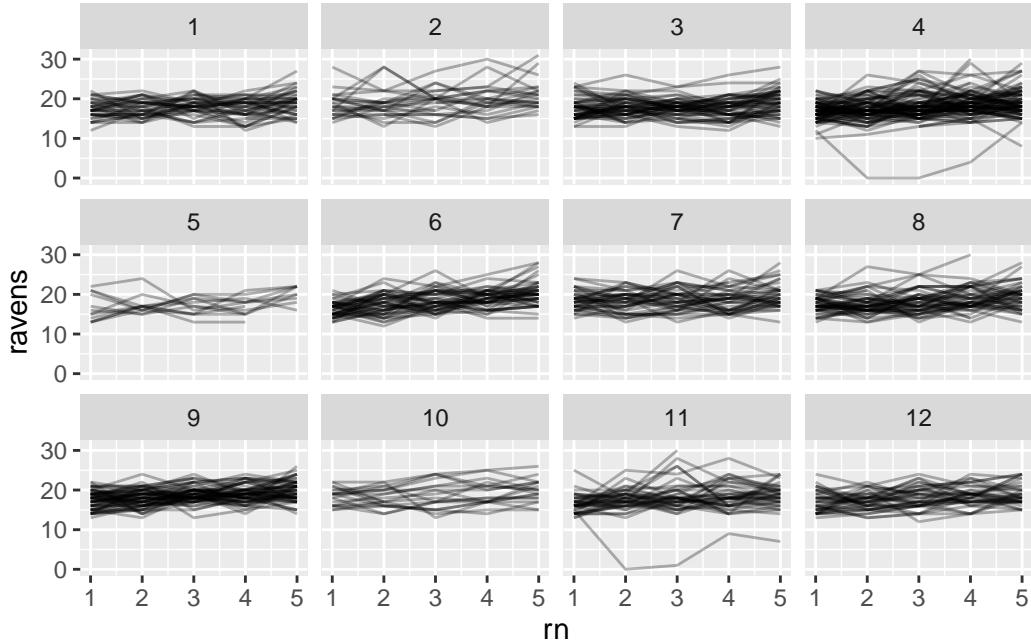
Warning: Removed 114 rows containing missing values (`geom_line()`).



or

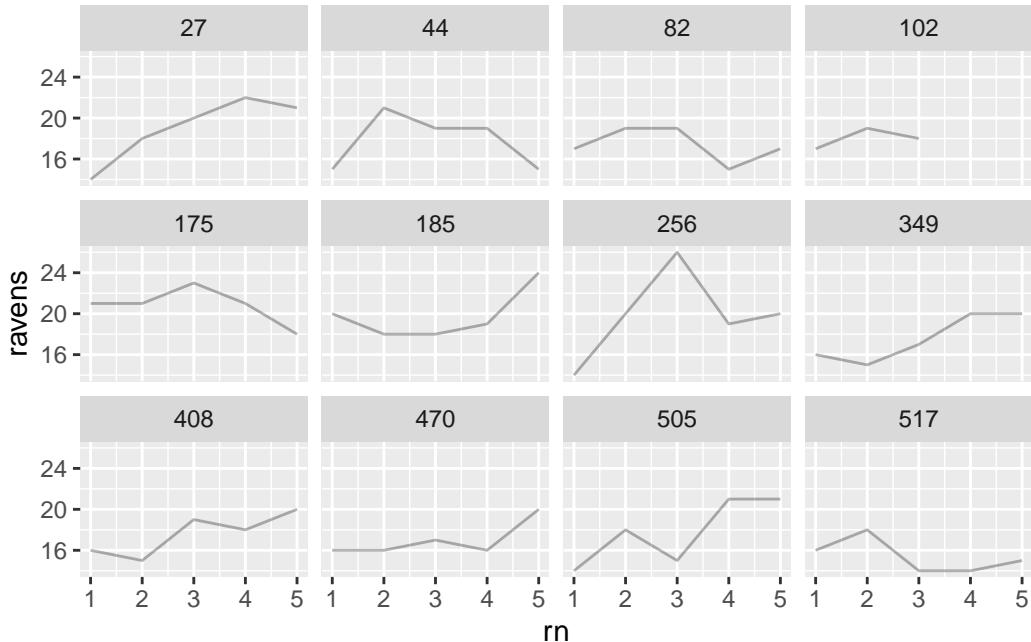
```
ggplot( data=kenya, aes( x=rn, y=ravens, group=id ) ) +  
  facet_wrap( ~ schoolid ) +  
  geom_line( alpha=0.3 )
```

Warning: Removed 114 rows containing missing values (`geom_line()`).



```
id.sub = sample( unique( kenya$id), 12 )  
ken.sub = subset( kenya, id %in% id.sub )  
ggplot( data=ken.sub, aes( x=rn, y=ravens, group=id ) ) +  
  facet_wrap( ~ id ) +  
  geom_line( alpha=0.3 )
```

Warning: Removed 2 rows containing missing values (`geom_line()`).



We have lots of noise! But there is also a trend.

The progression of marginal means show there is growth over time, on average:

```
mosaic::favstats( ravens ~ rn, data=kenya )
```

```
Registered S3 method overwritten by 'mosaic':
method                      from
fortify.SpatialPolygonsDataFrame ggplot2
```

	rn	min	Q1	median	Q3	max	mean	sd	n	missing
1	1	1	16	17	19	28	17.3	2.56	537	9
2	2	0	16	17	19	28	17.7	2.79	529	17
3	3	0	16	18	20	30	18.3	3.04	523	23
4	4	4	17	18	20	30	18.6	2.97	513	33
5	5	7	17	19	21	31	19.5	3.10	496	50

(Using the mosaic package we can do this.)

The above also shows that we have some missing data, more as the study progresses.

We drop these missing observations: