Resources for S-043: Multilevel and Longitudinal Models

Luke Miratrix and Friends

Invalid Date

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).

1 Do I take S-043? A self-assessment

1.1 How to decide to take the course?

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 a 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

1.2 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.

Part I

R & R MARKDOWN

2 Intro to R Markdown

2.1 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.

2.2 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:

3 A big section

while this

```
## A small section
```

produces this:

3.1 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.

3.2 Embedding R code

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

3.2.1 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 witll 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 HTLM).

- `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.

3.2.2 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 too using

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

which produces -0.04".

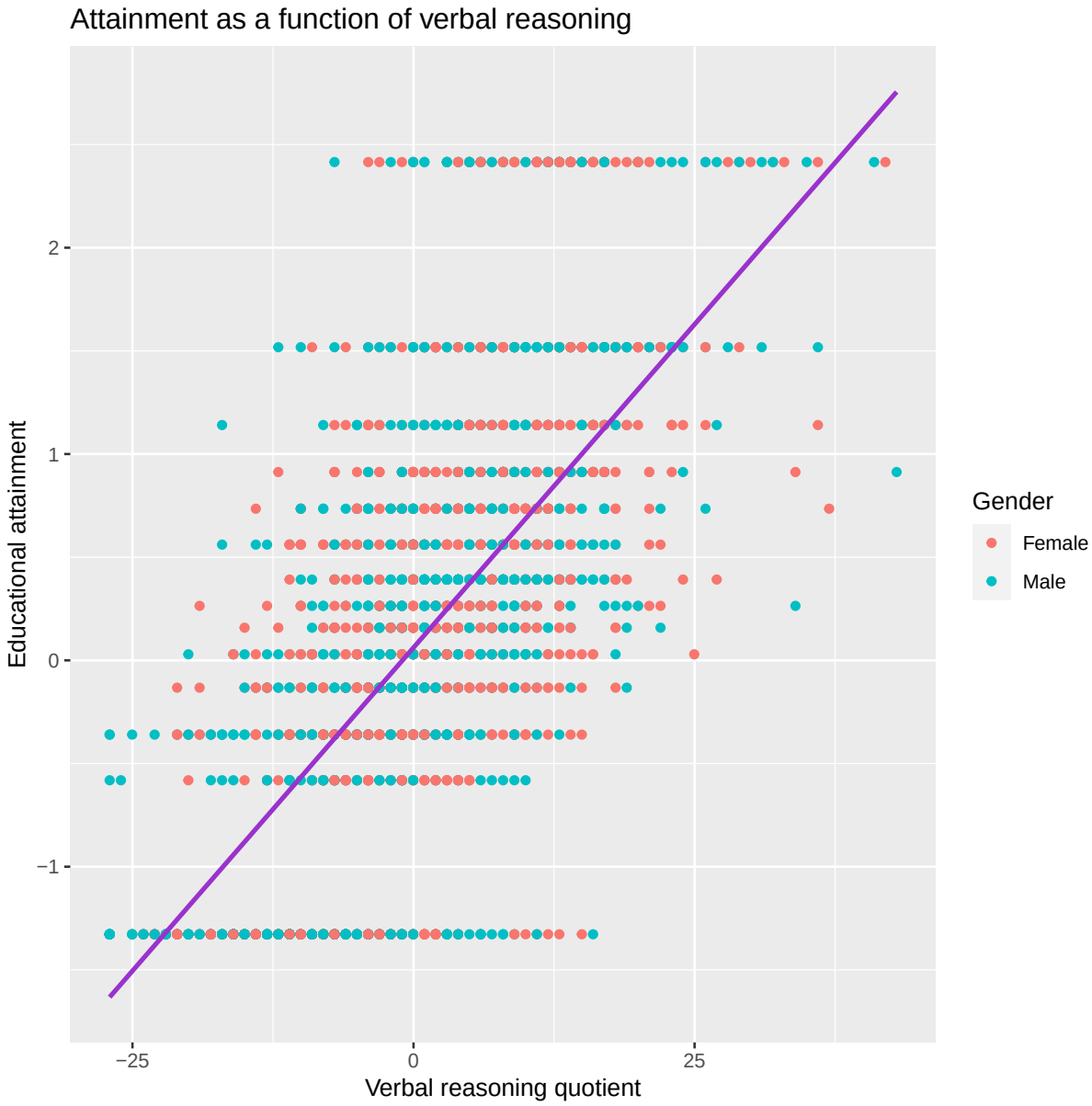
Here we used two commands: `round` and `mean`. You can use more commands and write more complex inline code, depending on what you want to report.

3.3 Embedding plots

Plots are easy to embed too. For example,

```
library(ggplot2)

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

ggplot(data=dat, aes(p7vrq, attain, colour=male)) +
  geom_point() +
  labs(title="Attainment as a function of verbal reasoning",
       x = "Verbal reasoning quotient",
       y = "Educational attainment", colour="Gender") +
  geom_smooth(method="lm", se=FALSE, colour="darkorchid3")
```

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



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).

3.4 Embedding tables

Finally, you can directly render tables in R Markdown. There are many different packages, but in class we'll mostly use `texreg` and `stargazer`. (NB the newer function `tab_model` from `sjPlot` is excellent as well!)

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
# 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: Fri, Aug 25, 2023 - 11:55:34

Table 3.1

Statistic	N	Mean	St. Dev.	Min	Max

We can also use `texreg` and `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         -0.02
```

```
                       (0.03)       (0.03)       (0.03)
Male                   -0.12 **     -0.12 **     -0.12 **
                       (0.04)       (0.04)       (0.04)
Maternal education                   0.49 ***     0.24 ***
                                     (0.05)       (0.05)
Paternal education                                0.54 ***
                                                  (0.06)
-------------------------------------------------------------
R^2                     0.00         0.05         0.09
Adj. R^2                0.00         0.05         0.08
Num. obs.            2310         2310         2310
=============================================================
*** 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.

3.5 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 `LaTeX`syntax 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 `$\mu$` compiles to μ. This will be very helpful when we ask you to match R output to model parameters in homework.

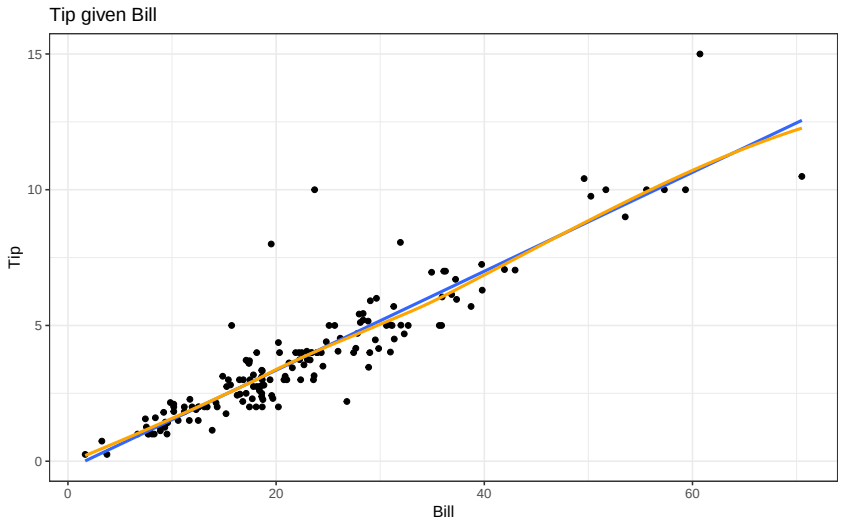
4 Intro to Regression

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.

4.1 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)
```

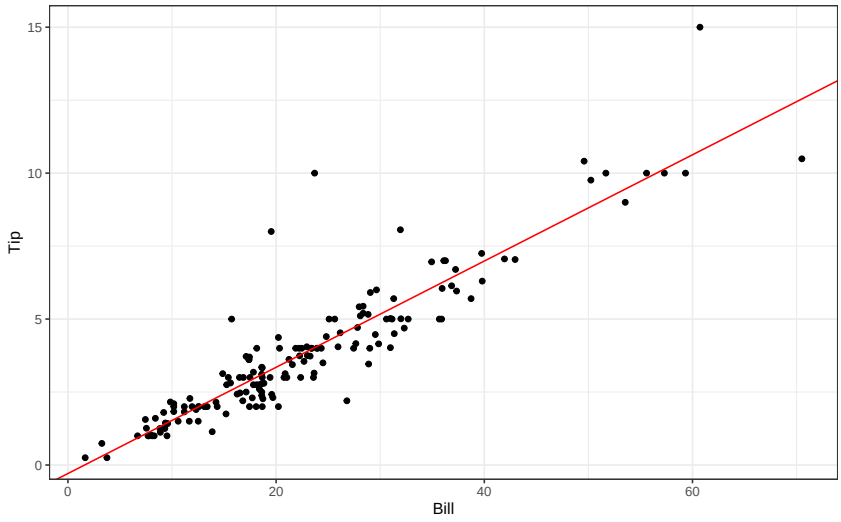
```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)   -0.292   0.166       -1.76 8.06e- 2
2 Bill           0.182   0.00645     28.2  5.24e-63
```

```
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" )
```



4.2 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
Credity      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).

4.2.1 Easy Tabulation and Graphing of Multiple Regression Models

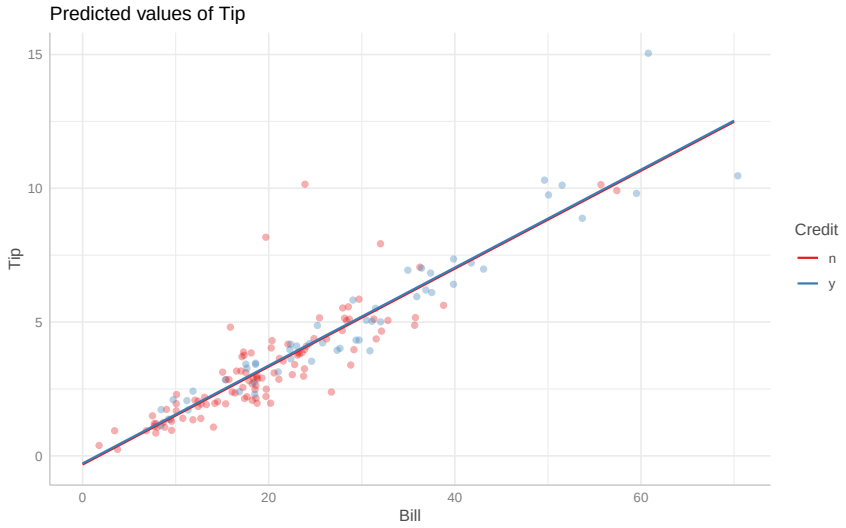
For publication-ready tables and graphics, R has many wonderful packages to automate the process. I (JG) am partial to `tab_model` from `sjPlot` for regression tables, and `ggeffects` for regression graphs, as shown below. See more in the other chapters.

```
# tabulate the results of our two tip models
tab_model(mod, tip.mod)
```

	Tip			Tip		
Predictors	Estimates	CI	p	Estimates	CI	p
(Intercept)	-0.29	-0.62 – 0.04	0.081	-0.25	-0.66 – 0.15	0.211

Bill	0.18	0.17 – 0.19	**<0.001**	0.18	0.17 – 0.20	**<0.001**
Guests				-0.03	-0.24 – 0.17	0.747
Credit [y]				0.04	-0.32 – 0.40	0.818
Observations	157			157		
R^2 / R^2 adjusted	0.837 / 0.836			0.838 / 0.834		

```
# 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)
```



4.3 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.

4.3.1 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))
```

4.3.2 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.

4.3.3 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:

```
mlm2 = lm( ClintonVote ~ HouseholdIncome + HouseholdIncome + HighSchool +
               EighthGradeMath, data=USStates)

mlm3 = lm( ClintonVote ~ HouseholdIncome + HouseholdIncome + HighSchool +
               EighthGradeMath + Region, data=USStates)
anova( mlm2, mlm3 )
```

```
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.

4.3.4 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
```

```
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
[39] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[77] 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

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
```

4.4 Some extra stuff (optional)

4.4.1 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
```

4.4.2 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.

4.4.3 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 )
```

4.4.4 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.

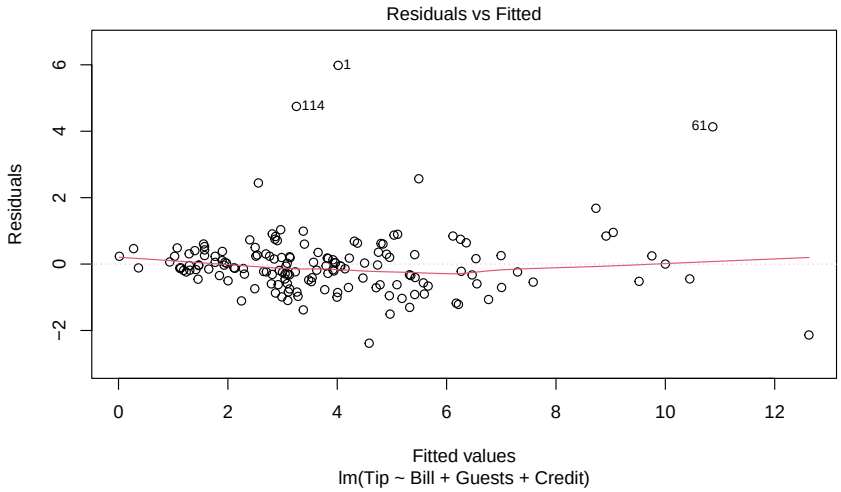
```
data(AllCountries)
dev.lm = lm( BirthRate ~ Rural + Health + ElderlyPop, data=AllCountries )
summary( dev.lm  )
```

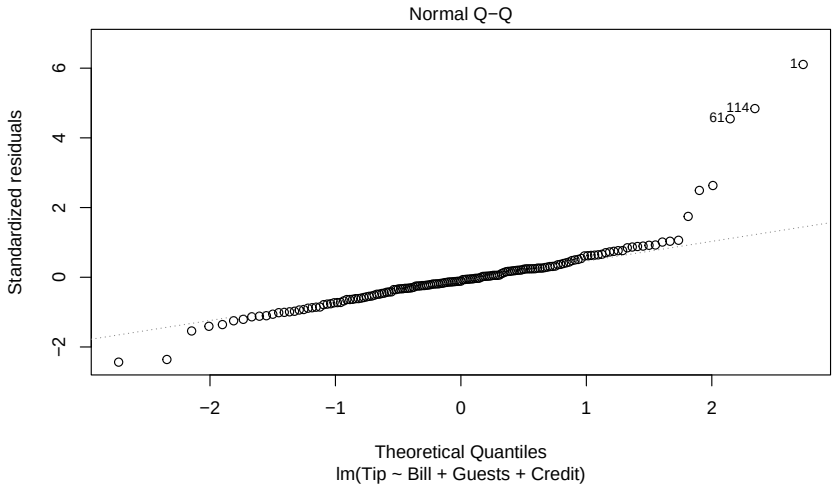
```
Call:
lm(formula = BirthRate ~ Rural + Health + ElderlyPop, data = AllCountries)

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
```

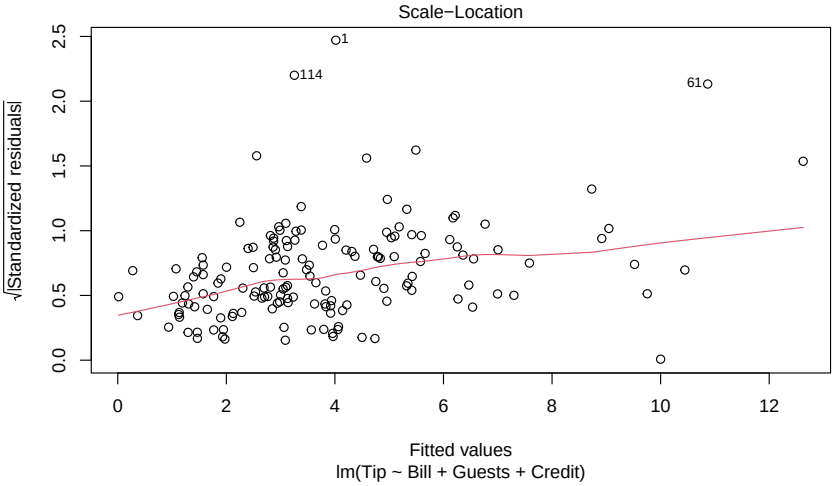
4.4.5 Residual plots and model fit

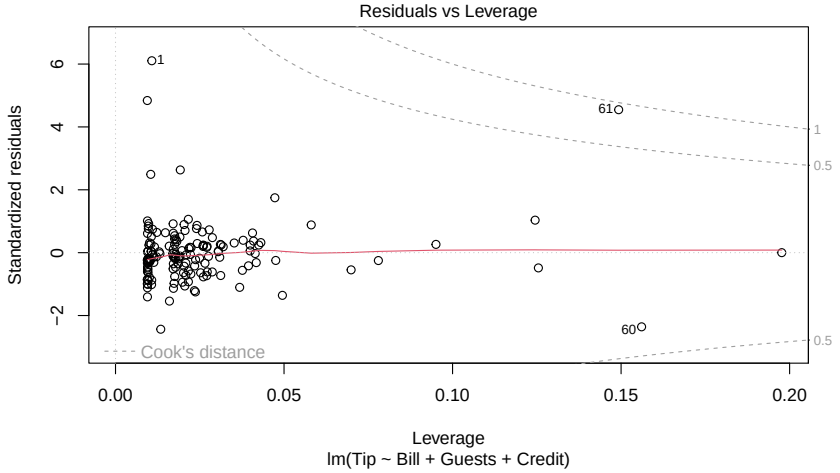
If we throw out model into the `plot` function, we get some nice regression diagnostics.

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




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(Tip ~ Bill + Guests + Credit)
1
114
61

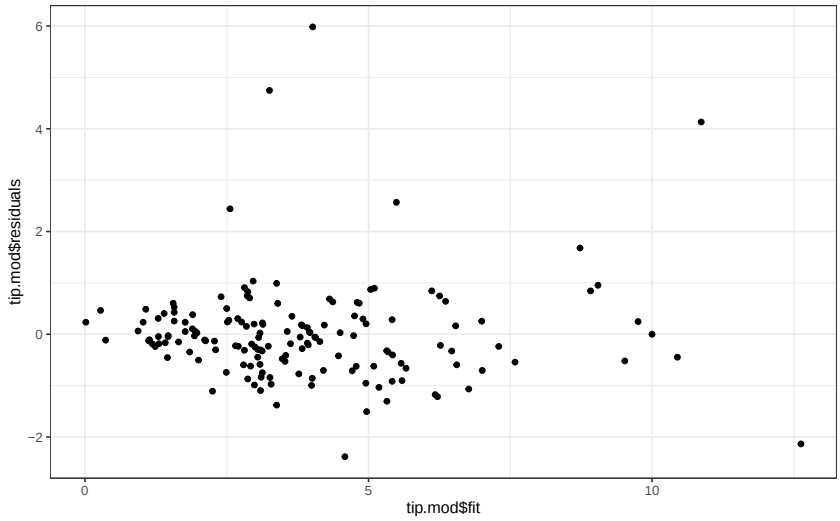

Scale−Location
√|Standardized residuals|
Fitted values
lm(Tip ~ Bill + Guests + Credit)
1
114
61


Residuals vs Leverage
Standardized residuals
Leverage
lm(Tip ~ Bill + Guests + Credit)
Cook's distance
1
0.5
0.5
1
61
60

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:

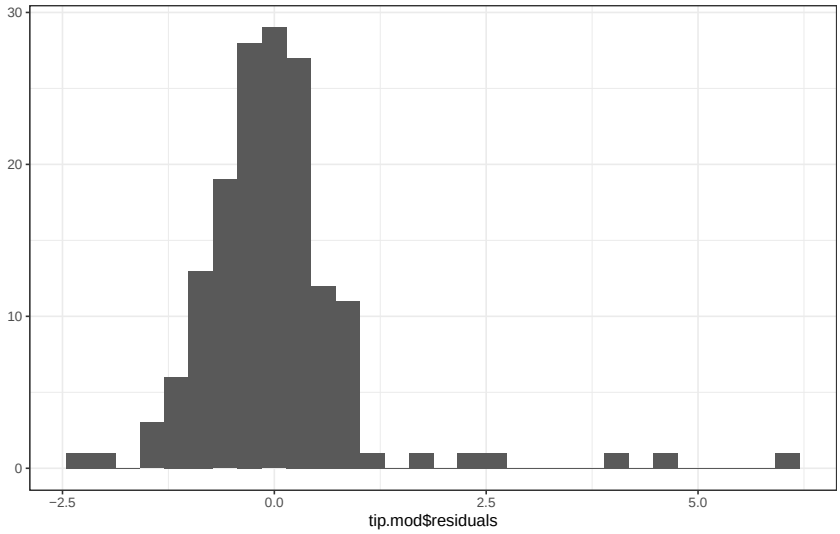
```
qplot(tip.mod$fit, tip.mod$residuals )
```

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



and

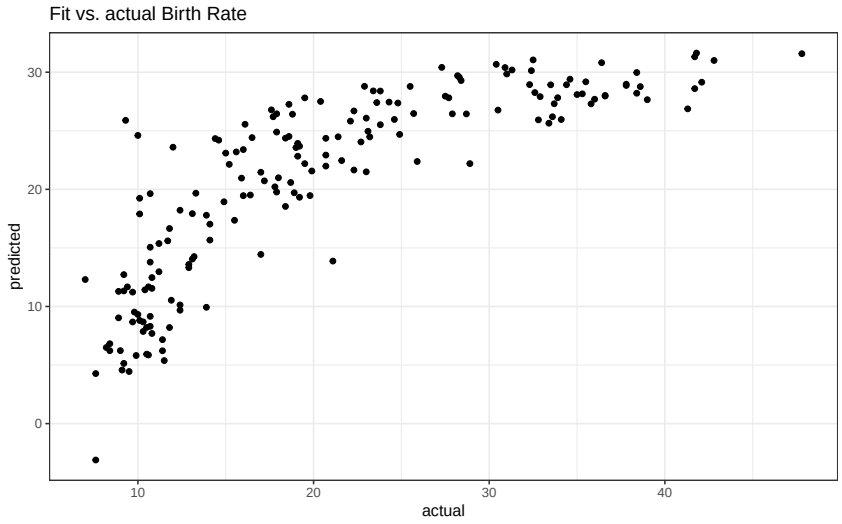
```
qplot(tip.mod$residuals, bins=30)
```



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.

5 Tips, Tricks, and Debugging in R

This chapter is a complete hodge-podge of section covers ways of dealing with data, especially messy data you might have for final projects.

5.1 Some miscellaneous advice

So you are starting to learn R. But 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."

In this section we give a few things that you may find useful, either now or later.

5.1.1 A few random tips

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.

5.1.2 Quick tips regarding R Markdown report generation

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.

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.

5.1.3 Saving R objects

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)
```

5.1.4 R style (based on Google style guide)

Try to do the following

- Comment your code!
- Structure of an R file:
  - Descriptive comments (including date)
  - Load libraries
  - Constants and script parameters (# iterations, etc.)
  - Functions (with descriptive comment after first line)
  - Everything else
- variableName / variable.name, FunctionVerb, kConstantName. not_like_this
- Curly Braces, line breaks: see previous slide
- Consistency: 2-space indents, `y = (a * x) + b + rnorm(1, sd=sigma)`
- Avoid `attach()`

5.1.5 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 4
```

```
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"
```

5.2 File structure: how not to do it

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.

5.2.1 Making Data Frames

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!

5.2.2 Making 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$trial1 = as.numeric( exp.dat$trial1 )
```

```
Warning: NAs introduced by coercion
```

5.2.3 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.

5.2.4 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:

```r
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
```

5.2.5 Summarizing 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:

```r
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. 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
```

5.3 Troubleshooting in R

By now you have gotten to the point where you can get some **really weird** errors in R and they can be quite, quite frustrating. This section talks about how to think about fixing them on your own. It also covers some common mistakes that can happen. Say you have some code that does a bootstrap and prints out a histogram. Nothing seems to work and the `hist` command is giving a strange error.

First step

■ **Put all your code in an R Script!** ■

Put all the commands, start to finish, in your script. The reason for this step is then you know what you are looking at. When scrolling to old commands and trying different things, you can get very tangled up. Anyway, say you do, and you are still getting a strange error:

```
> lovemale = rep(c(0,1,2), c(372, 807,34))
> loveboot = replicate(1000, {
```

```
+     lovesampmale = sample(lovemale, 1000, replace=TRUE)
+     propsampmale = table(lovesampmale)[0]/length(lovesampmale)
+     mean(propsampmale)
+ })
> hist(loveboot, breaks=20)

Error in hist.default(loveboot, breaks = 20) : character(0)
In addition: Warning messages:
1: In min(x) : no non-missing arguments to min; returning Inf
2: In max(x) : no non-missing arguments to max; returning -Inf
```

You might think `hist` is the culprit, but that might not be true.

First step is to check if you have any strange arguments to hist. Try running hist without any arguments other than the data.

■ **Always simplify when things aren't working!** ■

If that doesn't work (and here it won't), then the next step is to see what is going on is to look at what you are making a histogram out of!

```
[1] NaN NaN NaN NaN NaN NaN
```

You can also look at `loveboot` by clicking on it in your 'Workspace' to see if it is weird. If it has a bunch of `NA` or `NaN` then you need to fix your bootstrap code. You are trying to make a histogram out of bad data. Another rule:

■ **Always look at your data and variables!** ■

Those bad data came from somewhere! Let's examine what is happening inside your bootstrap. The easiest way is to run the stuff **inside** your replicate to get one replicate and see what is going on. This illustrates a very important debugging rule:

■ **Break your code down and check each piece.** ■

The code inside your `replicate` should run by itself. So try it, looking at the value each time

```
  lovesampmale = sample(lovemale, 1000, replace=TRUE)
  head(lovesampmale)
```

```
[1] 1 1 1 0 1 0
```

```
  propsampmale = table(lovesampmale)[0]/length(lovesampmale)
  propsampmale
```

```
named numeric(0)
```

```
  mean(propsampmale)
```

```
[1] NaN
```

We see that the `propsampmale` line is going wonky. We unpack the pieces

```
table(lovesampmale)
```

```
lovesampmale
  0   1   2
294 680  26
```

```
length(lovesampmale)
```

```
[1] 1000
```

```
table(lovesampmale)[0]
```

```
named integer(0)
```

We finally find the error. We need quotation marks around the 0. Without the quotes, R interprets "[0]" as taking the 0th entry of the table, which doesn't exist, rather than the entry **named** "0," which does[1]

[1] Why? Because for a table those things at the top are **names** and all names are considered words. We denote words in R with quotation marks

```
table(lovesampmale)["0"]
```

```
  0
294
```

5.3.1 Aside: the table technique

The "table technique" to calculate the proportion of some list of data that has a given value is dangerous. In particular if that value isn't present, then the table could drop it, causing some real trouble. Instead use

```
mean(propsampmale == 0)
```

```
[1] NaN
```

5.3.2 Code redundancies

Sometimes you don't need parts of your code at all! The propsampmale has the answer. No need for the final mean in the above code!

5.3.3 Categories should be words

For categories, don't use numbers. Instead use

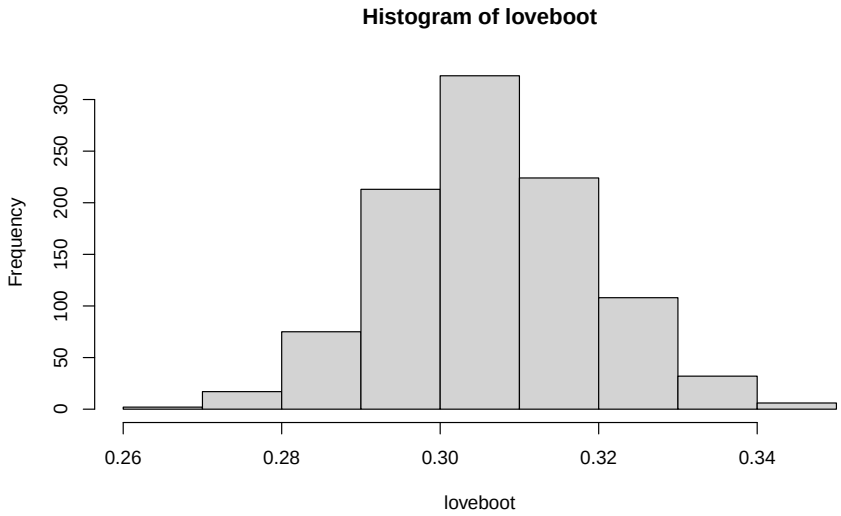
```
lovemale = rep(c("Little", "Some", "Lots"), c(372, 807,34))
```

and then your mean line will be

```
lovemale = rep(c("Little", "Some", "Lots"), c(372, 807,34))
```

giving your final fixed code (plot not shown):

```
lovemale = rep(c("Little", "Some", "Lots"), c(372, 807,34))
loveboot = replicate(1000, {
  lovesampmale = sample( lovemale, replace=TRUE )
  mean(lovesampmale == "Little")
})
hist(loveboot)
```


Histogram of loveboot
Frequency
0
50
100
150
200
250
300
0.26
0.28
0.30
0.32
0.34
loveboot

6 Making tables in Markdown

You might want to make tables. You should probably make charts, but every so often a table is a nice thing to have.

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 summery 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     -0.0390 0.25  0.715
2 B     -0.137  0.6   0.835
3 C      0.362  0.562 0.967
4 D     -0.234  0.4   0.817
5 E      0.0527 0.368 0.862
```

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
fred	A	-0.04	0.25	0.72
doug	B	-0.14	0.60	0.83
xiao	C	0.36	0.56	0.97
lily	D	-0.23	0.40	0.82
unknown	E	0.05	0.37	0.86

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.

6.1 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            20 (20.0)
     B            20 (20.0)
     C            16 (16.0)
     D            25 (25.0)
     E            19 (19.0)
  Z = tx (%)      43 (43.0)
  X (mean (SD)) 0.06 (1.04)
```

```
# 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                 57             43
  X (mean (SD))  0.02 (1.13)  0.12 (0.92)   0.646
  G (%)                                     0.163
     A             15 (26.3)     5 (11.6)
     B              8 (14.0)    12 (27.9)
     C              7 (12.3)     9 (20.9)
     D             15 (26.3)    10 (23.3)
     E             12 (21.1)     7 (16.3)
  Y (mean (SD)) -0.07 (0.84)  0.03 (0.85)   0.589
```

You can then use `kable` as so:

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

	co	tx	p	test
n	57		43	
X (mean (SD))	0.02 (1.13)		0.12 (0.92)	0.646
Y (mean (SD))	-0.07 (0.84)		0.03 (0.85)	0.589

6.2 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: Fri, Aug 25, 2023 - 11:56:14

Table 6.3

Statistic	N	Mean	St. Dev.	Min	Max
X	100	0.062	1.040	−2.463	2.422
Y	100	−0.026	0.842	−1.702	2.154

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 6.4: Table 1: Descriptive statistics

Statistic	N	Mean	St. Dev.
X	100	0.062	1.040
Y	100	−0.026	0.842

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

7 Making Regression Tables and More Complete Summary Output

This document demonstrates the different regression table methods, and talks about some weird wrinkles with using them with multilevel modeling.

7.1 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 3.571429     7 Female 1.111111          1
2  1 4.000000     7   Male 1.000000          0
3  1 3.714286     6 Female 3.555556          0
4  1 3.500000     6   Male 1.111111          1
5  1 4.000000     5   Male 1.000000          1
6  1 2.428571     7 Female 2.222222          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*. There are two packages, one is `texreg`

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

```
===================================================
                  Model 1     Model 2     Model 3
---------------------------------------------------
(Intercept)         3.68 ***    3.62 ***    4.13 ***
                   (0.25)      (0.25)      (0.38)
grade               0.00        0.00       -0.08
                   (0.04)      (0.04)      (0.06)
genderMale                      0.13       -0.73
                               (0.08)      (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            0.001           -0.078
                         (0.038)          (0.038)          (0.059)

genderMale                                 0.130           -0.727
                                          (0.083)          (0.498)

grade:genderMale                                           0.133*
                                                          (0.076)

Constant                 3.676***         3.624***         4.127***
                          (0.249)          (0.250)          (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
```

7.2 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.

7.3 Better summary output for lmer: Getting p-values

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 ***
HSTRUE      -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)
HSTRUE -0.801
```

7.4 The texreg package

There are two options, one is `screenreg` and the other is `texreg()`.

7.4.1 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 ***      3.53 ***
                       (0.06)        (0.09)
HSTRUE                               -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!

7.4.2 Using texreg and latex

The `textreg` 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***	3.53***
	(0.06)	(0.09)
HSTRUE		−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" which it puts where there is space.

7.5 Stargazer

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

Table 7.1

	Dependent variable:	
	esafe	
	(1)	(2)
HS		−0.295***
		(0.108)
Constant	3.346***	3.528***
	(0.059)	(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	

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.

7.5.1 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 )
```

```
[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' )
```

Table 7.2

	Dependent variable:	
	esafe	
	(1)	(2)
HS		-0.295^{***}
		(0.108)
Constant	3.346^{***}	3.528^{***}
	(0.059)	(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	

8 Pretty ANOVA Tables with `kable`

8.1 R Setup

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

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

8.2 Create fake data

We create a data set called `a` that has 100 observations and specifies our outcome `Y` as a funciton 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 ) )
```

8.3 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 )
```

8.4 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     98 133.28
2     97 109.26  1    24.018 21.322 1.192e-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	133.2799	NA	NA	NA	NA
Y ~ A + B		97	109.2623	1	24.01766	21.32221	1.19e-05

8.5 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)	-0.0391383	0.1069672	-0.365891	0.7152432

	term	estimate	std.error	statistic	p.value
A		0.1949385	0.1091445	1.786058	0.0772145
B		0.5391913	0.1167688	4.617598	0.0000119

9 A Math Reference: Sample Modeling Equations to Borrow

9.1 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 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.

9.1.1 Using this document

You are probably reading the PDF version of this document. But really you should open the .Rmd file and cut and paste the relevant equations into your own work, and then modify as necessary.

9.2 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 "α" and get α.

I can do subscripts by using an underscore. E.g., "α_j" gives α_j. I can also do superscripts by using a hat. E.g., "α^2" gives α^2.

9.2.1 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.

9.2.2 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

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

Also consider:

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

giving

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

9.2.3 Normal text in equations

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

```
$$
5 + my dog = 10
$$
```

$$5 + mydog = 10$$

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.

9.3 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}
y_{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}
$$
```

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

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

```
$$
\epsilon_i \sim N( 0, \sigma^2_y ), \mbox{ and } u_{j} \sim N( 0, \sigma^2_\alpha )
$$
```

$$
\epsilon_i \sim N(0, \sigma^2_y), \text{ and } u_j \sim N(0, \sigma^2_\alpha)
$$

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

```
$$
\epsilon_i \stackrel{i.i.d}{\sim} N( 0, \sigma^2_y ),
\mbox{ and } u_{j} \stackrel{i.i.d}{\sim} N( 0, \sigma^2_\alpha )
$$
```

$$
\epsilon_i \overset{i.i.d}{\sim} N(0, \sigma^2_y), \text{ and } u_j \overset{i.i.d}{\sim} N(0, \sigma^2_\alpha)
$$

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

9.4 Random Slope Model

The canonical random slope model for HS&B with `ses` at level 1 and sector at level 2

Level 1 models:

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

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

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{bmatrix}
\begin{pmatrix}
0 \\
0
\end{pmatrix}\!\!,&
```

$$
\begin{pmatrix} u_{0j} \\
u_{1j}
\end{pmatrix} \sim N
\begin{bmatrix}
\begin{pmatrix}
0 \\
0
\end{pmatrix}\!\!,&
$$

```
\begin{pmatrix}
\tau_{00} & \tau_{01}\\
 & \tau_{11} \\
\end{pmatrix}
\end{bmatrix}
$$
```

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

The derivation of the reduced form is:

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

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

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} \sim  N
```

```
\begin{bmatrix}
\begin{pmatrix}
0 \\
0
\end{pmatrix}\!\!,&
\begin{pmatrix}
\sigma^2_{0} & \rho \sigma_0 \sigma_1 \\
 & \sigma^2_{1} \\
\end{pmatrix}
\end{bmatrix}
$$
```

$$\begin{bmatrix}\begin{pmatrix}0 \\ 0\end{pmatrix}\!\!,&\begin{pmatrix}\sigma^2_{0} & \rho \sigma_0 \sigma_1 \\ & \sigma^2_{1} \\\end{pmatrix}\end{bmatrix}$$

$$\begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} \sim N \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma^2_0 & \rho \sigma_0 \sigma_1 \\ & \sigma^2_1 \end{pmatrix} \right]$$

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!

9.5 Summations and fancy stuff

Fractions are as follows:

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

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

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

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

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

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

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

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!

10 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). 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
```

`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" )
```

10.1 Optional: Using old School `reshape`

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 seperated 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
```

10.1.1

11 An Introduction to Missing Data

11.1 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 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
```

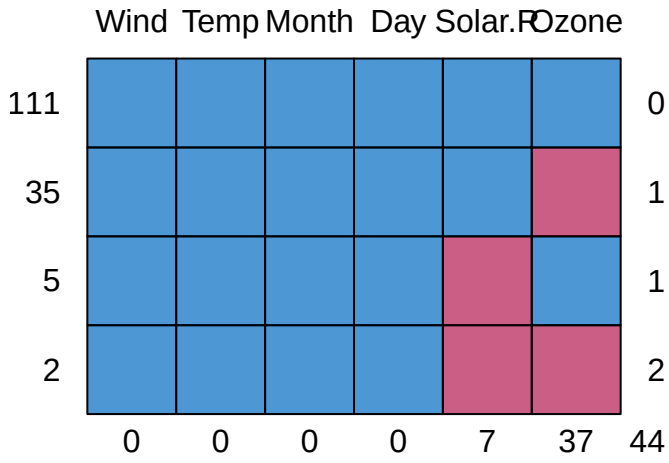
```
  summary(airquality[1:4])
```

```
     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                                   
```

11.2 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  0
35     1    1     1   1       1     0  1
5      1    1     1   1       0     1  1
2      1    1     1   1       0     0  2
       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
```

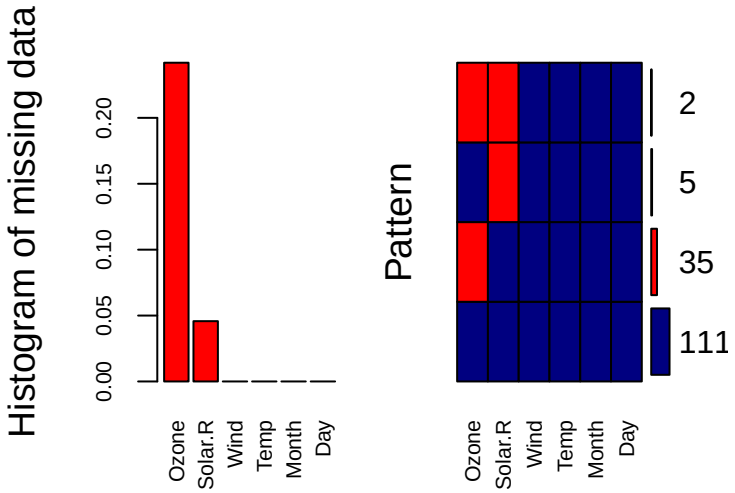
```
   Ozone Solar.R Wind Temp Month Day
1     41     190  7.4   67     5   1
2     36     118  8.0   72     5   2
3     12     149 12.6   74     5   3
4     18     313 11.5   62     5   4
5     NA      NA 14.3   56     5   5
6     28      NA 14.9   66     5   6
7     23     299  8.6   65     5   7
8     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.

11.2.1 The VIM Package

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

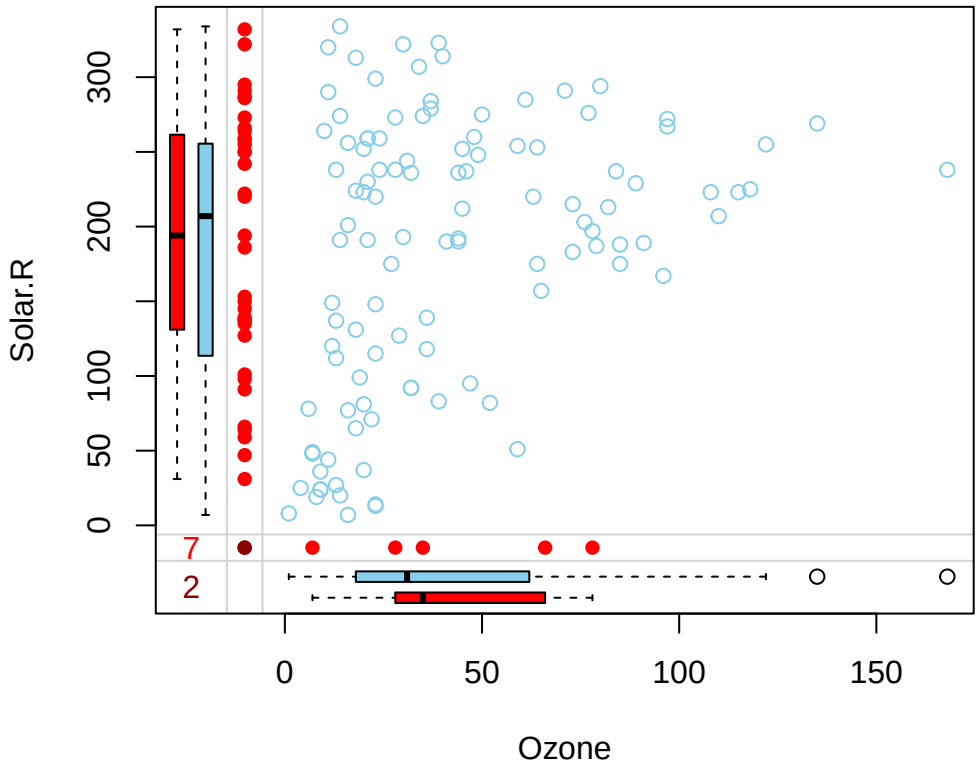
```
aggr(airquality, col=c('navyblue','red'),
     numbers=TRUE, sortVars=TRUE, labels=names(data),
     cex.axis=.7, gap=3, prop=c(TRUE, FALSE),
     ylab=c("Histogram of missing data","Pattern"))
```



```
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).

11.3 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!

11.4 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.

11.4.1 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

11.4.2 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
```

11.4.2.1 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 )
```

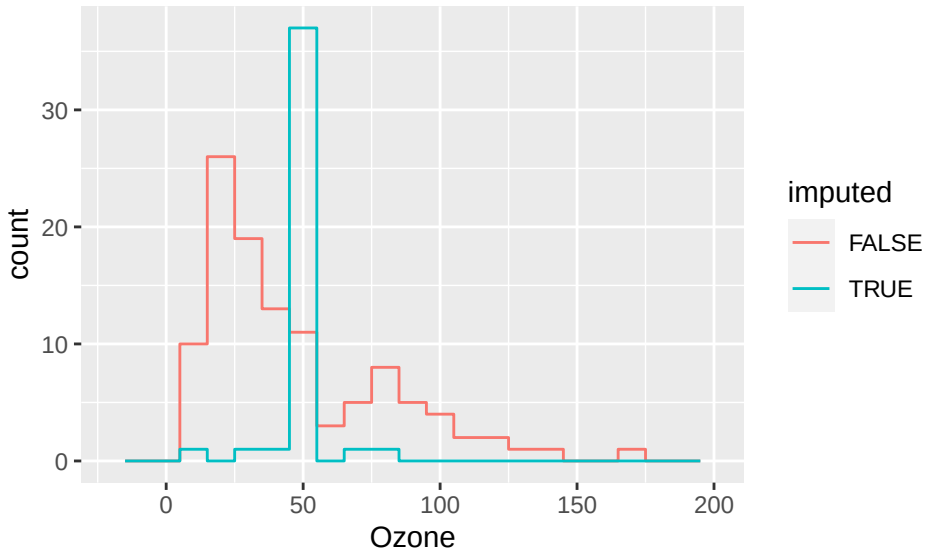
```
[1] 33
```

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

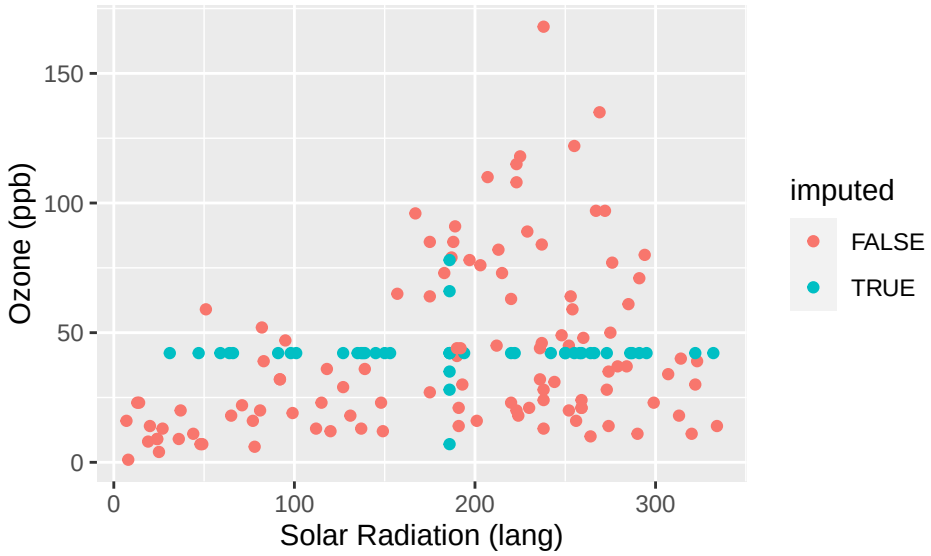
```
[1] 28.7
```

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.

11.4.2.2 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 `complete`s 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()
```

11.5 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.

11.5.1 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.

11.5.2 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 )
```

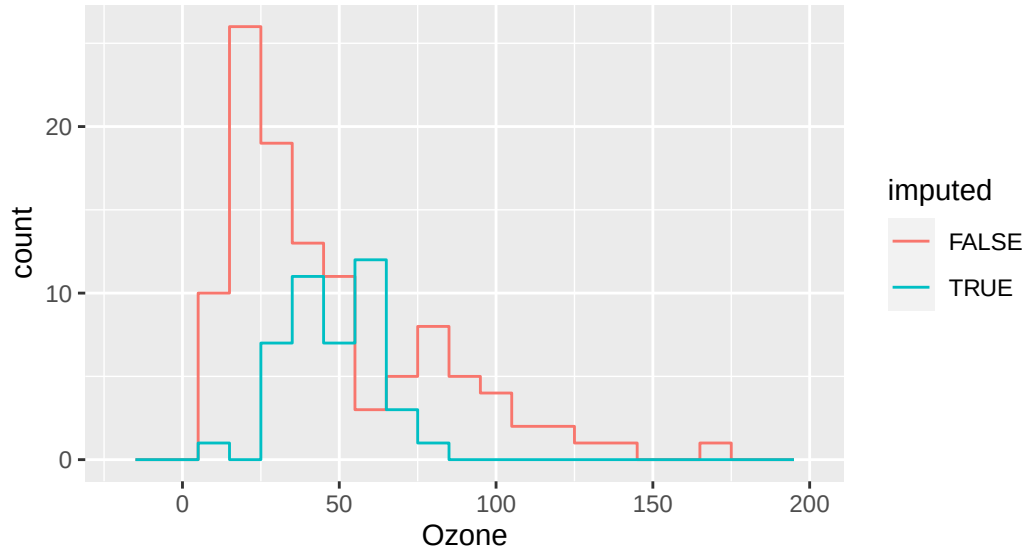
```
[1] 153
```

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

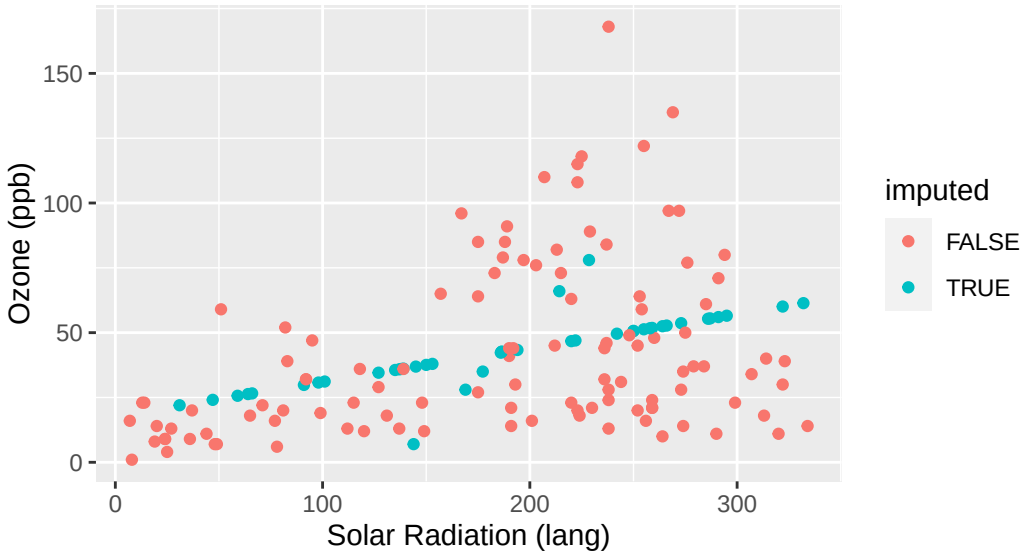
```
  cdat$imputed = ici( airquality )
```

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.

11.5.3 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
```

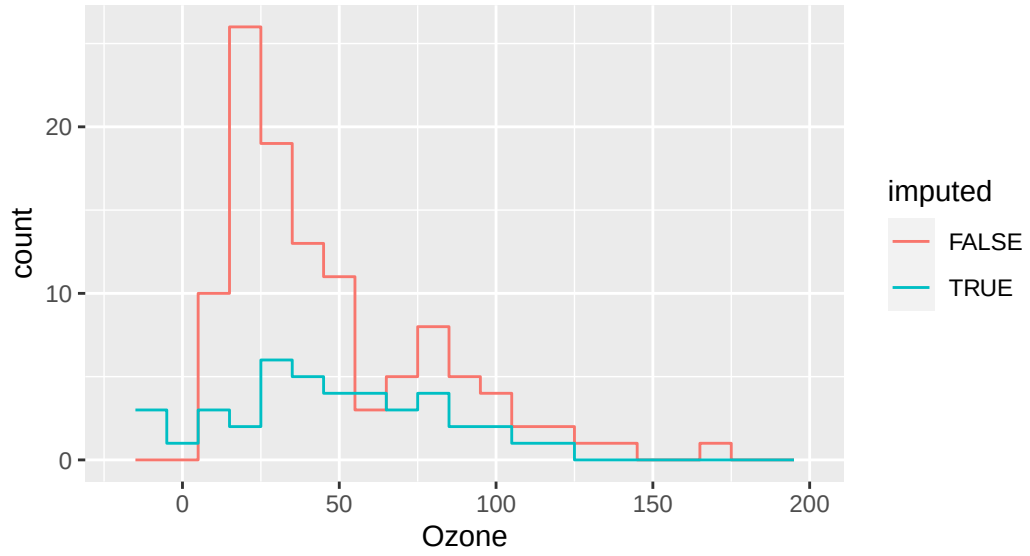
```
$Ozone
      1
5  34.4
10 31.6

$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.

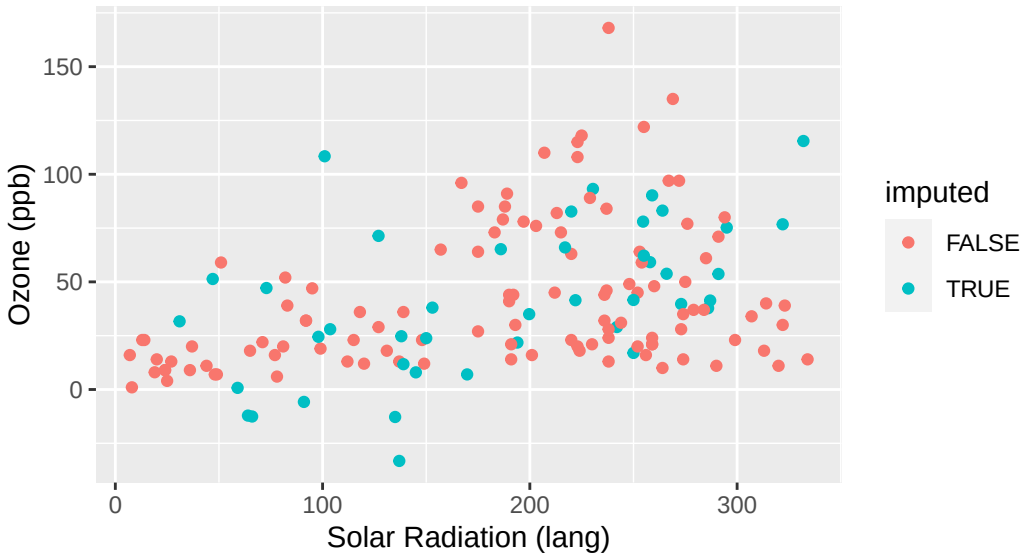
```
  imp <- mice(airquality[,1:2],method="norm.nob",m=1,maxit=1,seed=1)
```

```
 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?

11.6 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 = airqualitysub[,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.

11.7 Extensions

11.7.1 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"
```

11.7.2 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.

11.7.3 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 XXXXXXX
2       2 XXXXXX.
3       3 XXXXXXX
4       4 XXXXXXX
5       6 XXXXXXX
6       7 XXXXXXX
```

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 XXXXXX.
```

```
    224      21      10       6       5       5       4       3       3       3
XXX.X.. XXXX..X X.XXXXX 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 XXXXXXX   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.XXXXX     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
```

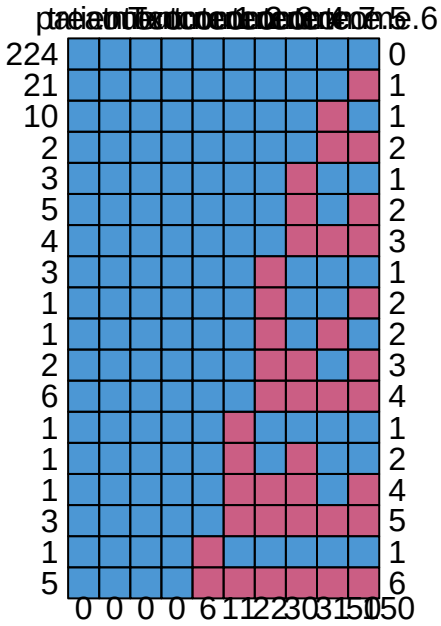
```
  ## 76% of patients with complete data
```

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         1 Itraconazole         1         1         1         0
8        2         0  Terbinafine         0         0         1         1
14       3         0  Terbinafine         0         0         0         0
21       4         0  Terbinafine         1         0         0         0
28       6         1 Itraconazole         1         1         1         0
35       7         1 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	1
21	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	0
5	1	1	1	1	1	1	1	0
4	1	1	1	1	1	1	1	0
3	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	0	1
2	1	1	1	1	1	1	0	0
6	1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	0	1	0
1	1	1	1	1	1	0	0	0
3	1	1	1	1	1	0	0	0
1	1	1	1	1	0	1	1	1
5	1	1	1	1	0	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	

```r
## 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 )
```

```
# A tibble: 20 x 5
   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
```

11.8 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 Buurin 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

11.9 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
```

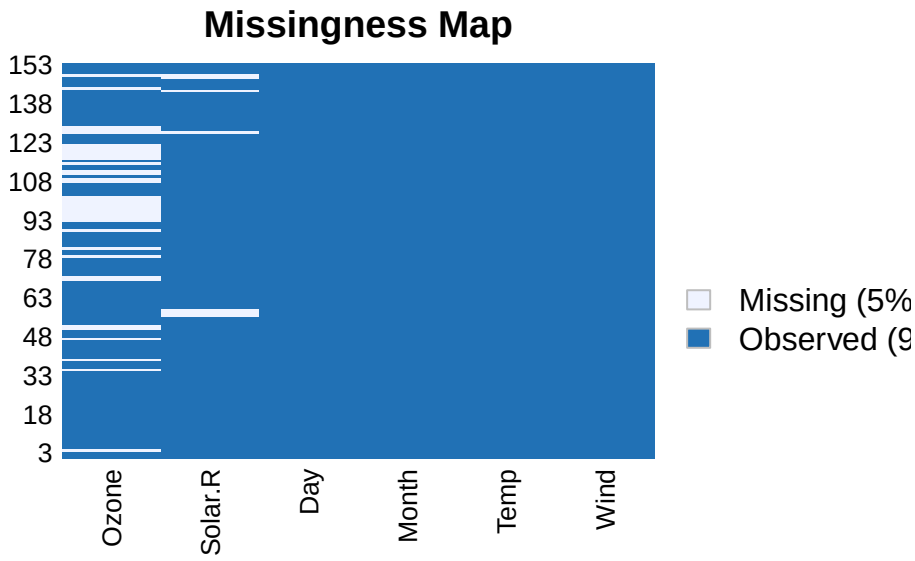
11.10 Appendix: The amelia package

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 --
```

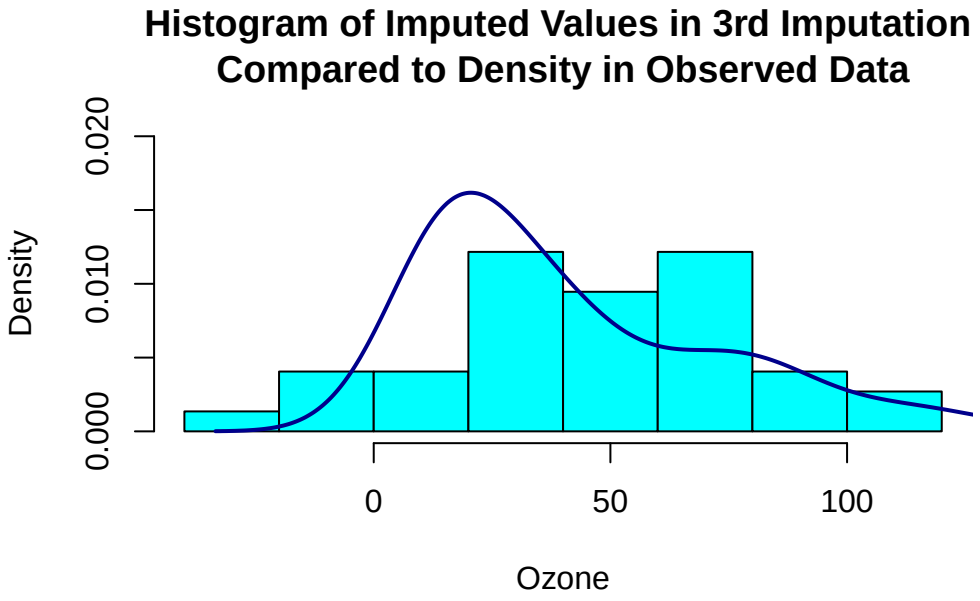
```
  1  2  3  4  5  6  7

-- 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)
```



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.

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.

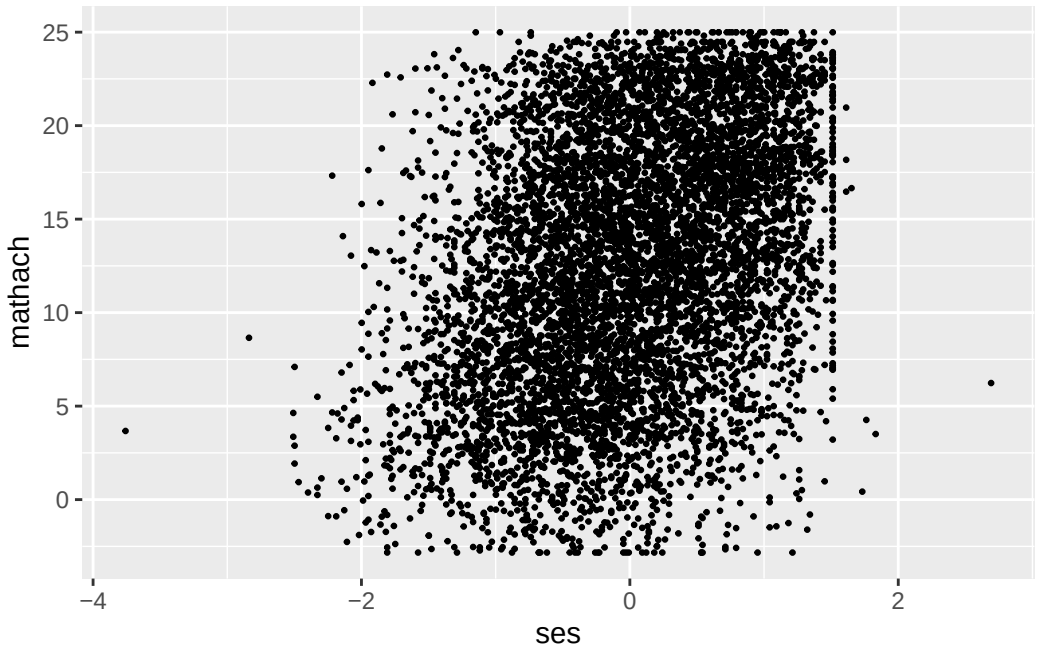
Happily, when you load data, it usually is. For example:

```
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()
```

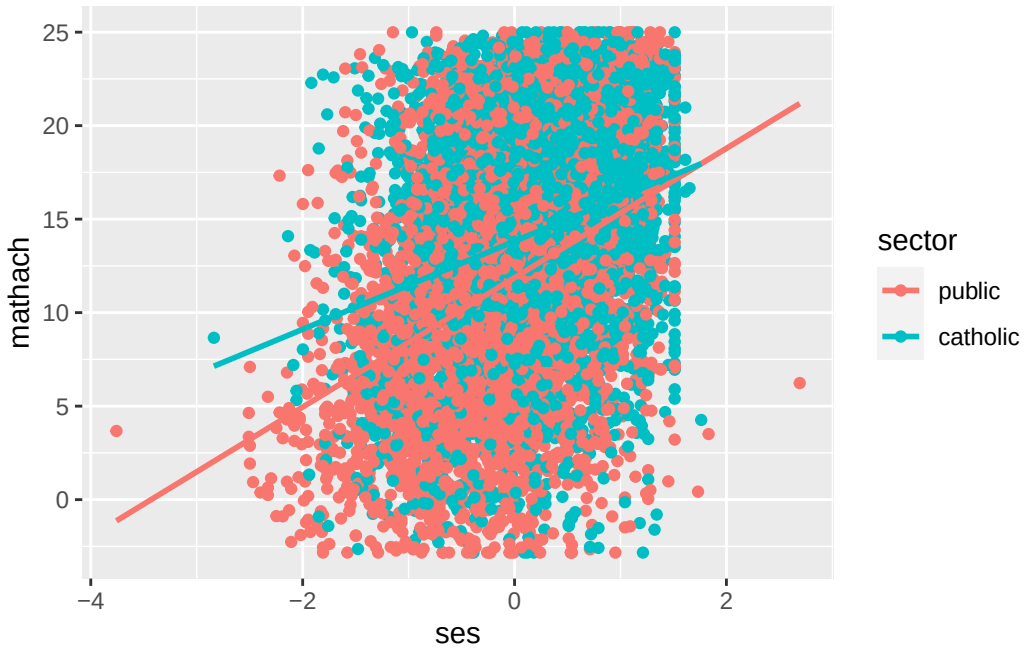
Note that `ggplot` wants the data frame to be neatly put together, including that categorical variables are listed as factors. This is why we convert the dummy `sector` to a factor above. Once you do this, however, it will label things in a nice way.

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'
```

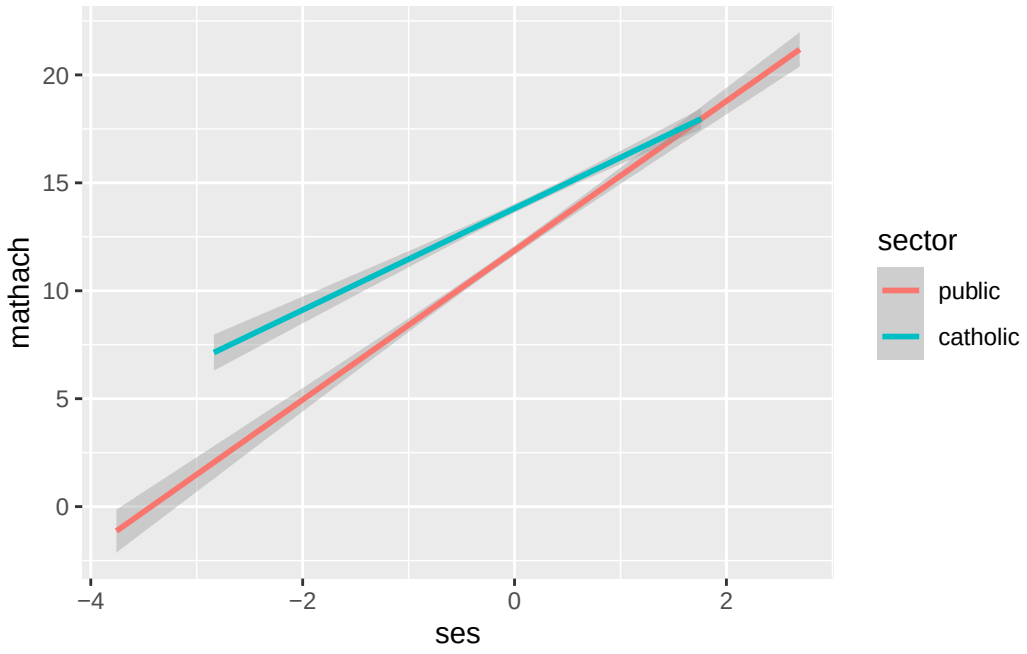


Notice how it automatically realized you have two subgroups of data defined by sector. It gives you a regression line for each group.

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'
```



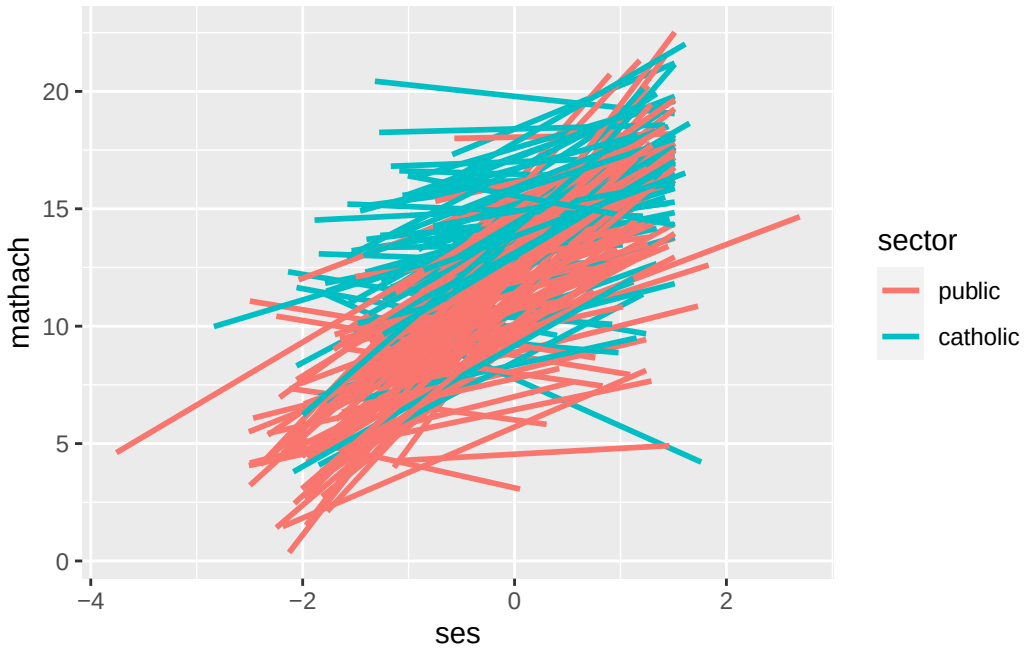
Here we also added some uncertainty bars around the regression lines by not saying `se = FALSE`. (Including uncertainty is the default.)

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:

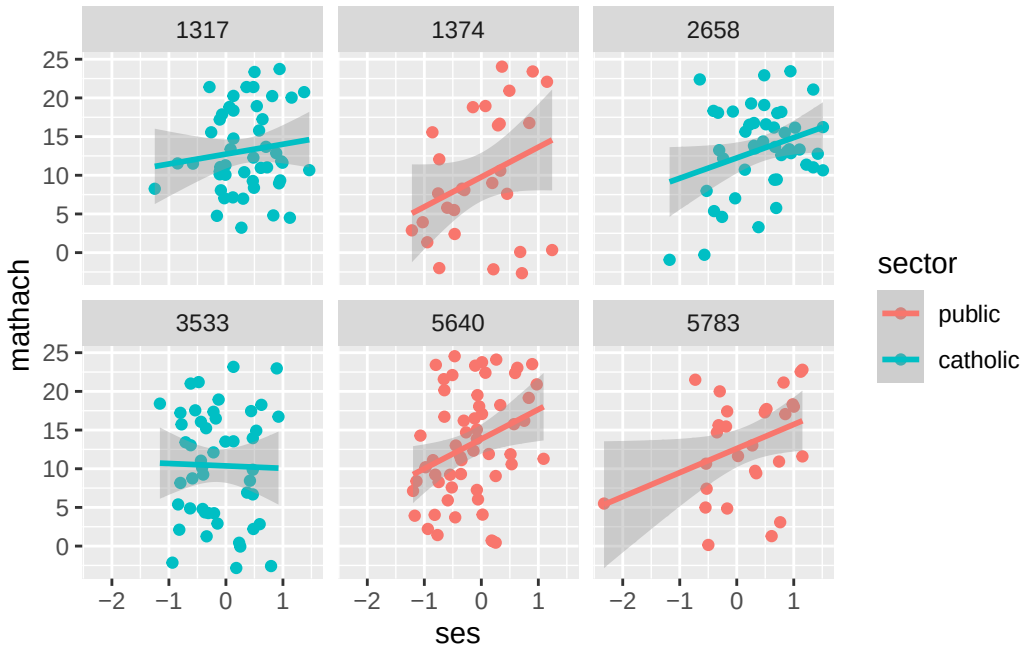
```
my.plot +
  facet_grid( ~ female ) +
    ggtitle("School-level trend lines for their male and female students") +
    labs(x="SES",y="Math Achievement")
```

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

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

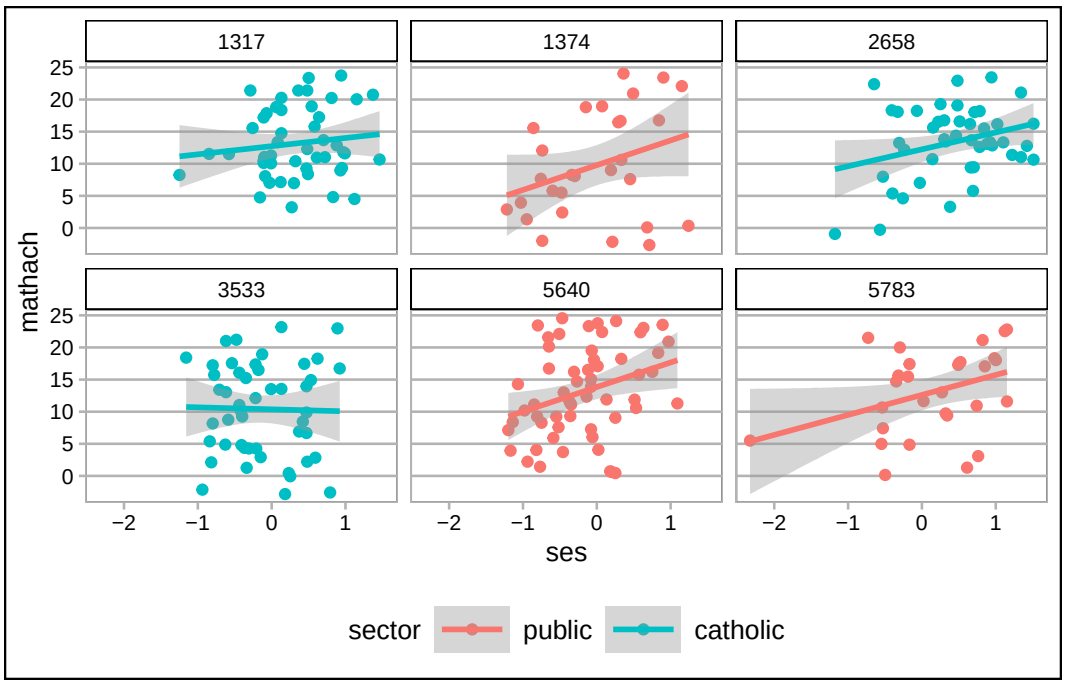
You can very quickly change the entire presentation of your plot using `themes`. There are pre-packaged ones, and you can make your own that you use over and over. Here we set up a theme to be used moving forward

```
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 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.

13.1 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).

13.1.1 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 EXPO.11 ATTIT.12 EXPO.12 ATTIT.13 EXPO.13 ATTIT.14 EXPO.14
  <dbl>    <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>, EXPO.15 <dbl>, FEMALE <dbl>,
#   MINORITY <dbl>, INCOME <dbl>
```

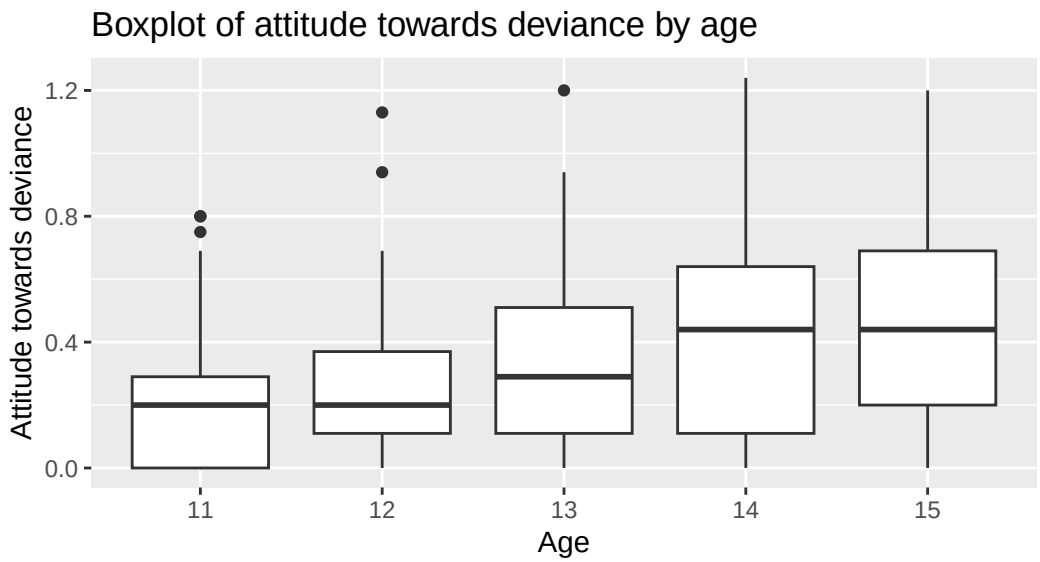
For our purposes, we want it in "long format", i.e. each student has multiple rows for the different observations. The `pivot_longer()` command does this for us.

```
nys1 <- nyswide |>
  pivot_longer(ATTIT.11:EXPO.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  EXPO
  <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")
```



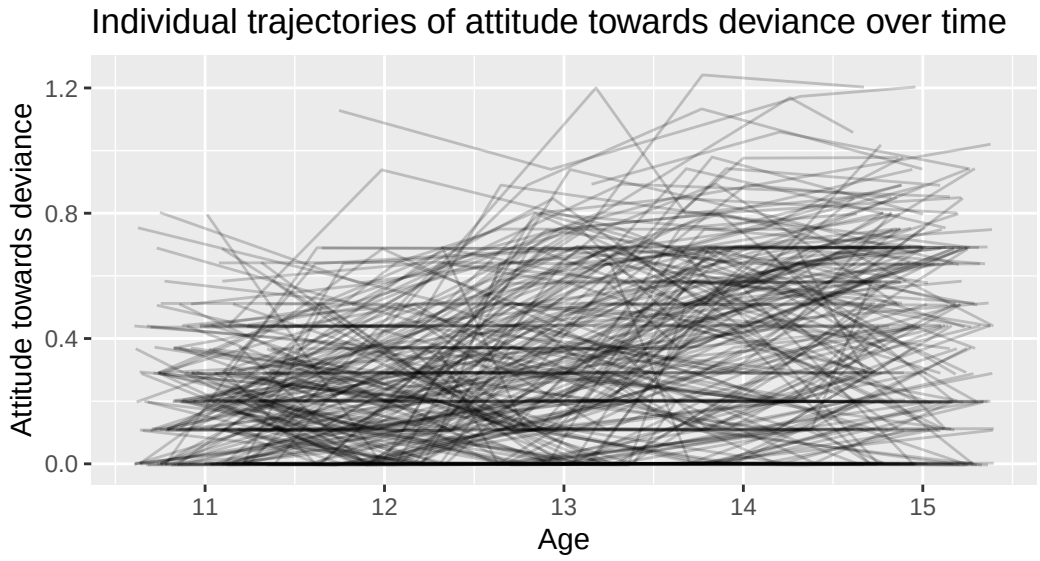
Note: The boxplot's "x" variable is the group. You get one box per group. The "y" variable is the data we are making boxplots of.

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).

13.2 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%.

13.3 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
                     0           1           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 variable
                 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
```

13.4 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 13.1

Statistic	N	Mean	St. Dev.	Min	Max

Table 13.2: 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

13.5 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")
```

13.6 Summarizing by group

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

```
aggdat = dat %>%
  group_by(schoolid, sector) %>%
  summarize( 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>
```

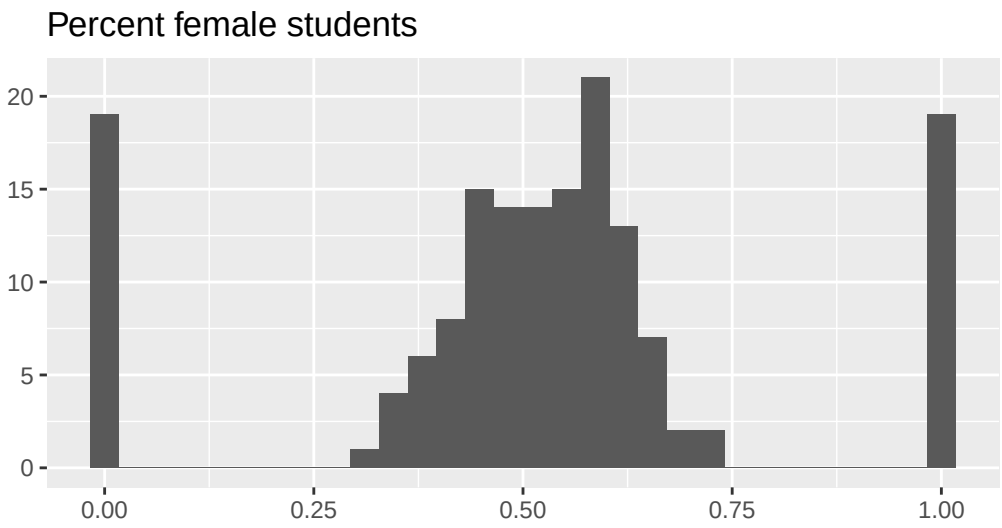
```
1      1224       0    0.596
2      1288       0    0.44
3      1296       0    0.646
4      1308       1    0
5      1317       1    1
6      1358       0    0.367
```

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 = "")
```

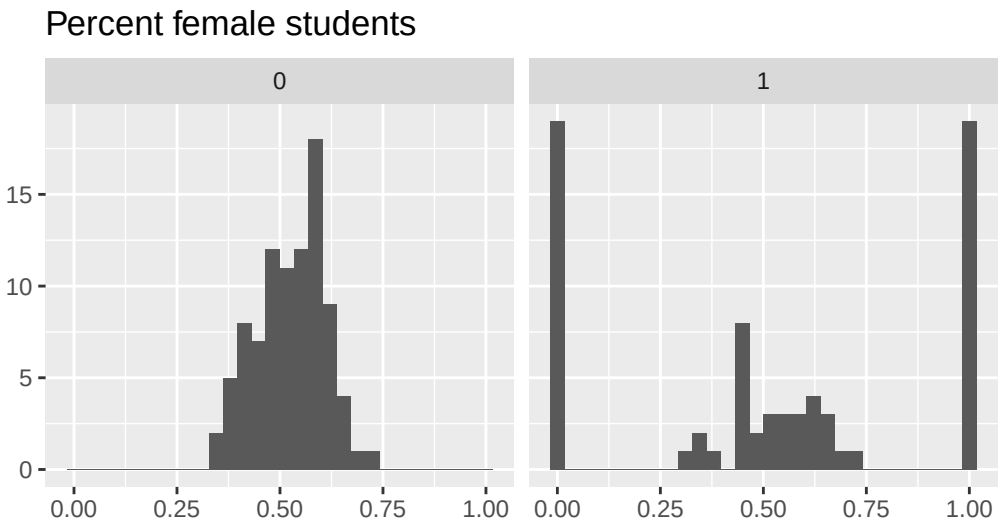
```
Warning: `qplot()` was deprecated in ggplot2 3.4.0.

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



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`.
```



13.7 Diagnostic plots

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

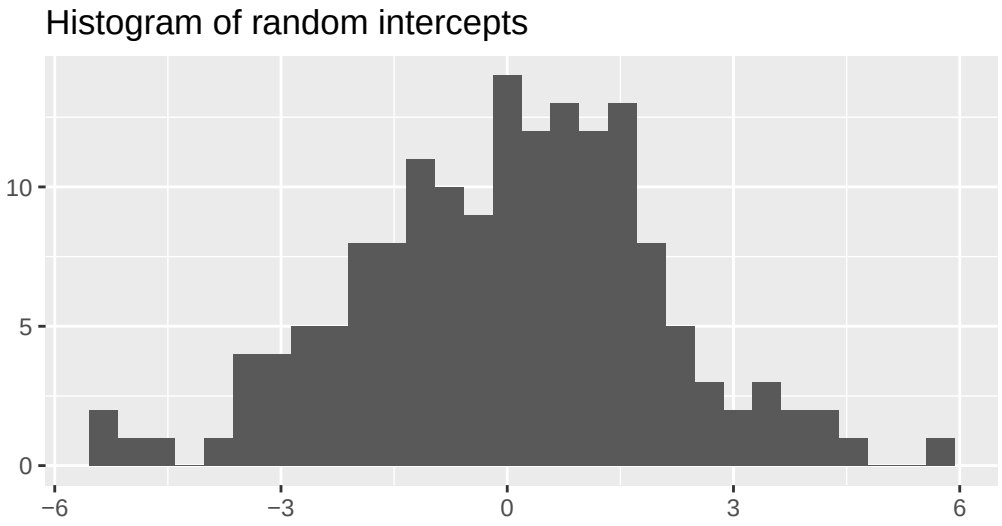
```
m1 <- lmer(mathach ~ 1 + ses + (1|schoolid), data=dat)
arm::display(m1)
```

```
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.

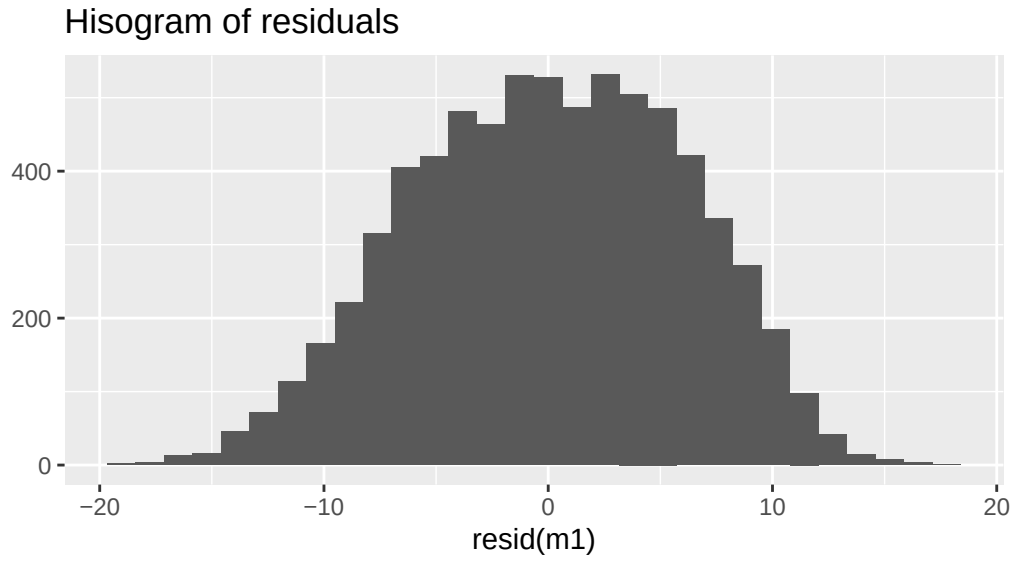
```
  qplot(ranef(m1)$schoolid[,1],
       main = "Histogram of random intercepts", xlab="")
```

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




```
  qplot(resid(m1),
       main = "Hisogram of residuals")
```

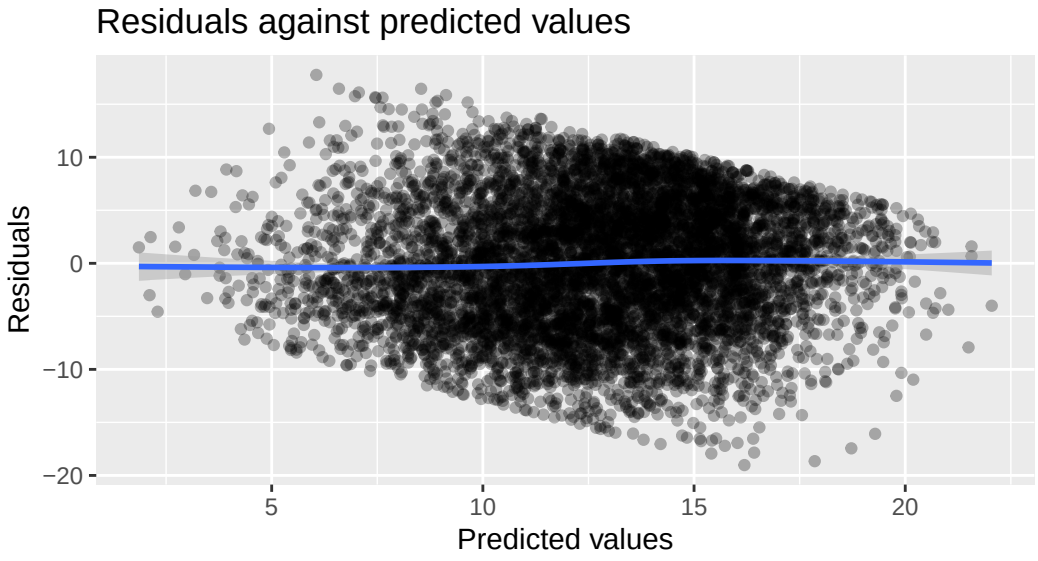
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



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")'
```

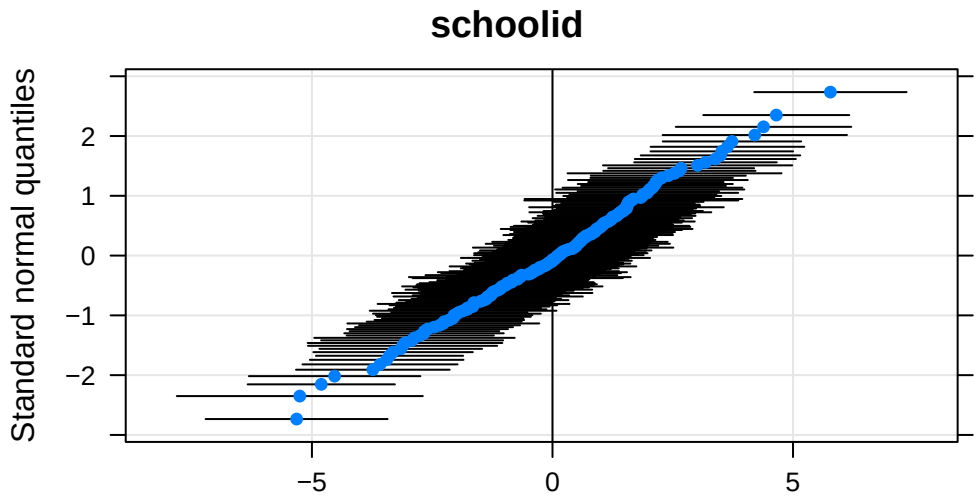


It looks reasonable (up to the discrete and bounded nature of our data). No major weird curves in the loess line through the residuals means linearity is a reasonable assumption. That being said, our nominal SEs around our loess line are tight, so the mild curve is probably evidence of *some* model misfit.

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

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

```
$schoolid
```



14 Example of making plots with `expand.grid`

14.1 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.

14.2 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.

14.2.1 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 achievment 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| = 0.00578929 (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
```

14.2.2 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
```

```
       (Intercept)                ses     sectorcatholic ses:sectorcatholic
         11.751789           2.957538           2.129531          -1.313363
```

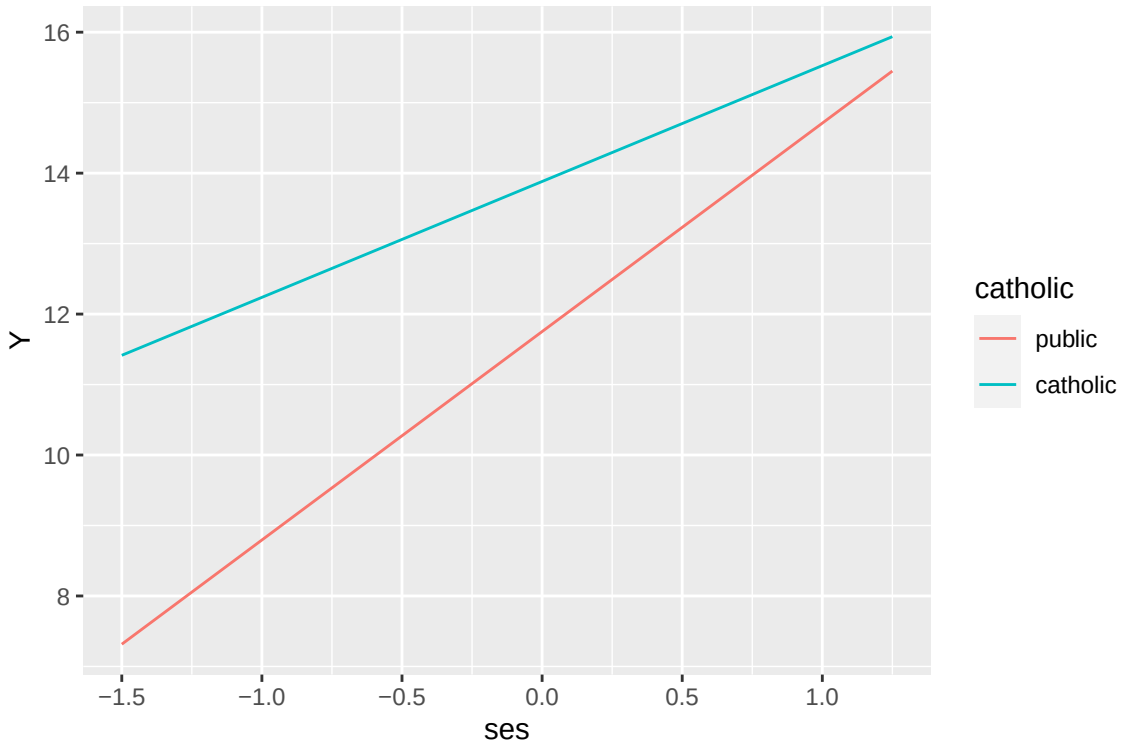
```
plt = mutate( plt,
              Y = cf[[1]] + cf[[2]]*ses + cf[[3]]*catholic + cf[[4]]*ses*catholic )
plt
```

```
    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()
```



14.2.2.1 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
```

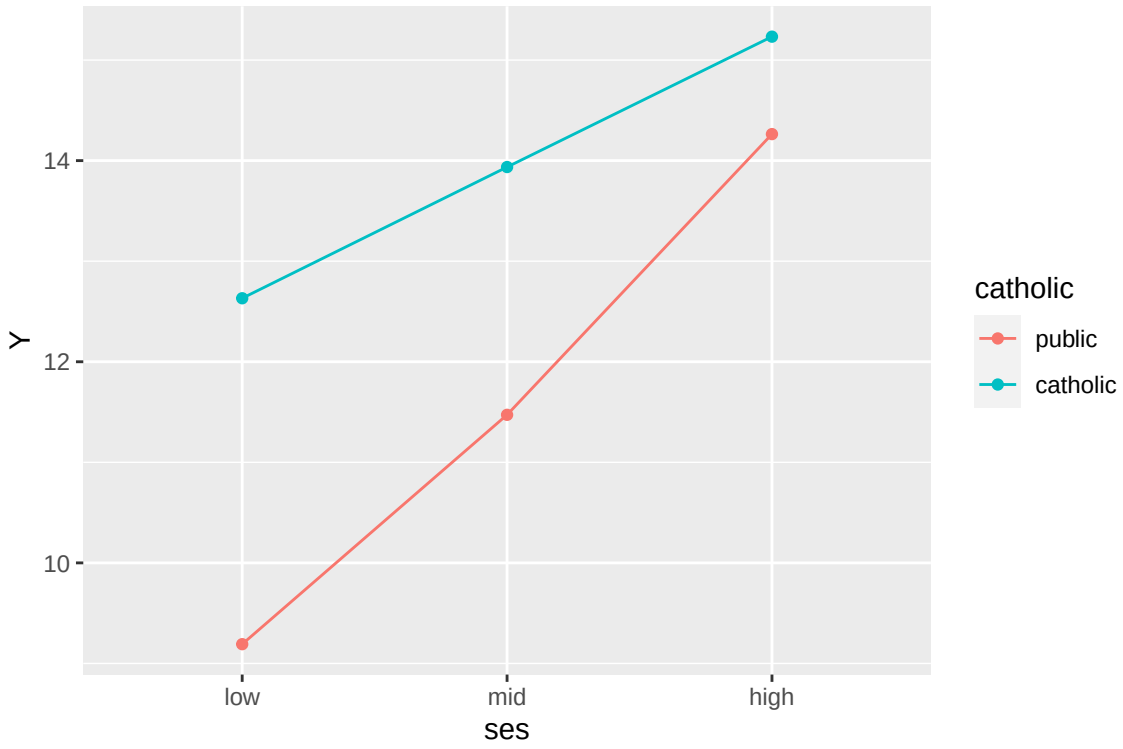
```
               (Intercept)                  ses.catmid
                 9.1915044                   2.2808807
               ses.cathigh              sectorcatholic
                 5.0721921                   3.4398984
 ses.catmid:sectorcatholic ses.cathigh:sectorcatholic
                -0.9759927                  -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.

14.2.3 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 sectorcatholic ses:sectorcatholic
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") )
```

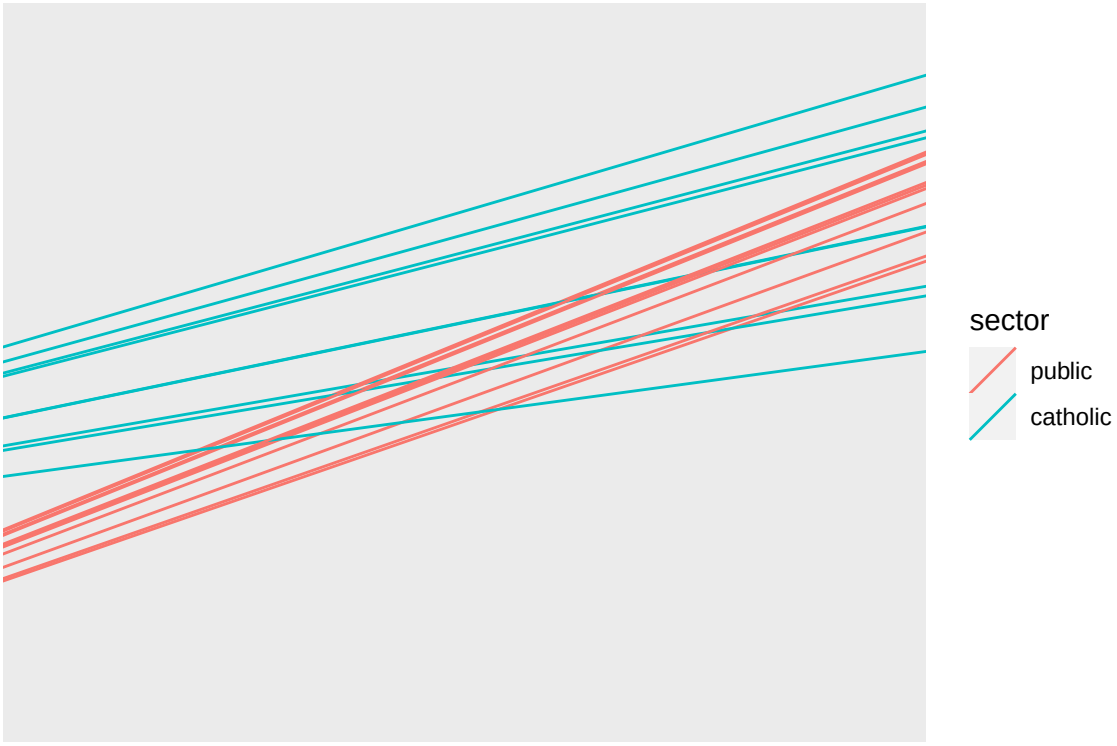
Note how we have to add up our gammas to get our betas for each school. See our final betas, one set for each school:

```
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 Emperical Bayes estimates of the random effects added to our school level fixed effects to get the $\widehat{\beta}_{0j}, \widehat{\beta}_{1j}$ 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.

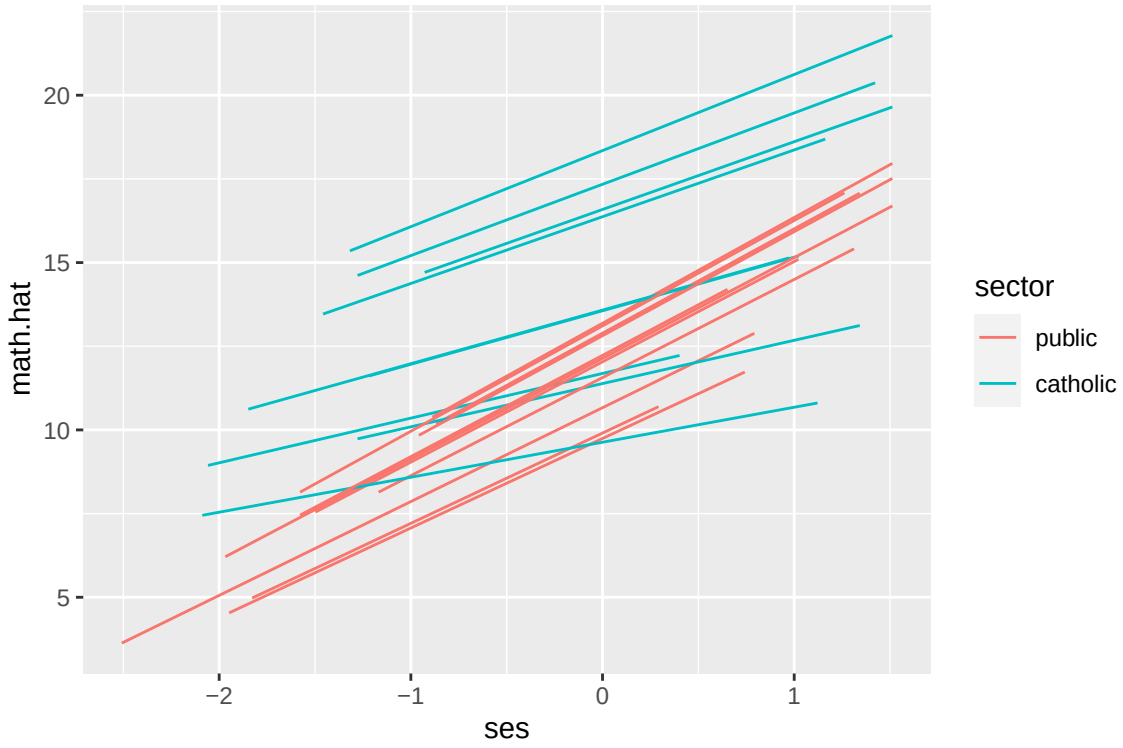
14.2.4 Plotting with predict()

A more general plotitng approach is to plot using `predict()`, where for each student we predict the outcome.

```
dat$math.hat = predict( M1 )
```

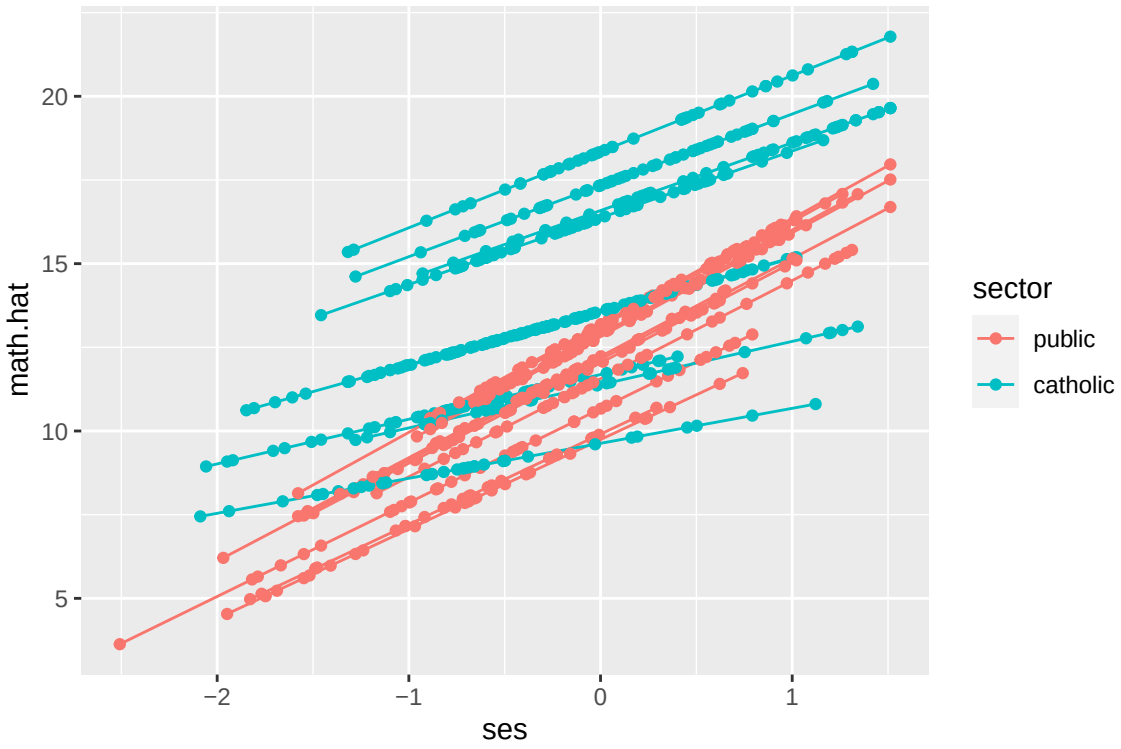
Now let's plot a subsample of 20 schools

```
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.

14.2.5 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 *seq*uence 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 )
```

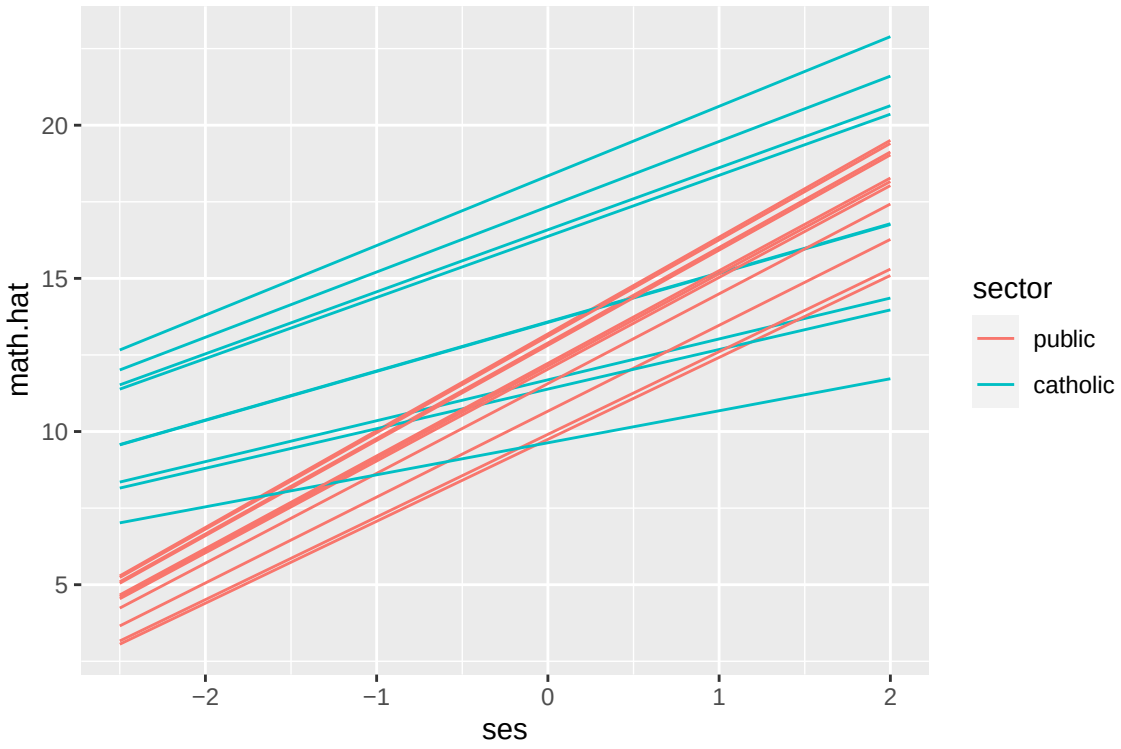
We finally predict for each school, predicting outcome for our fake kids in each school.

```
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.

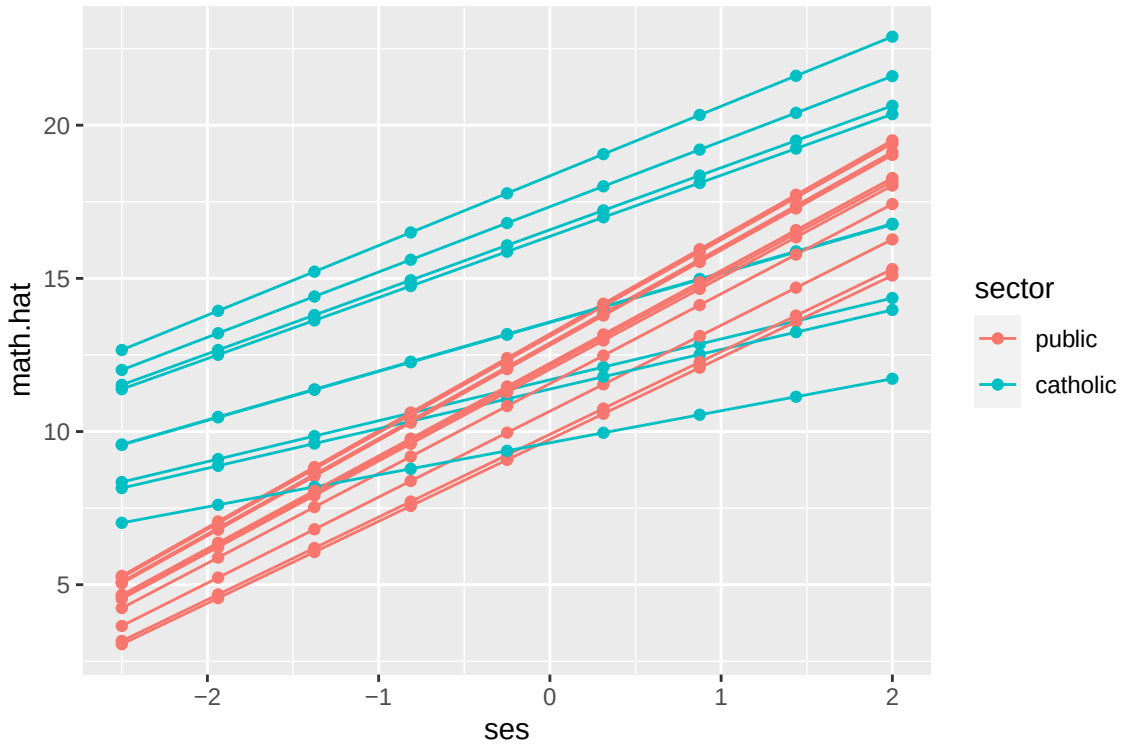
Using our new data, plot 20 random schools–this code is the same as in the prior subsection.

```
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.

14.2.6 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
```

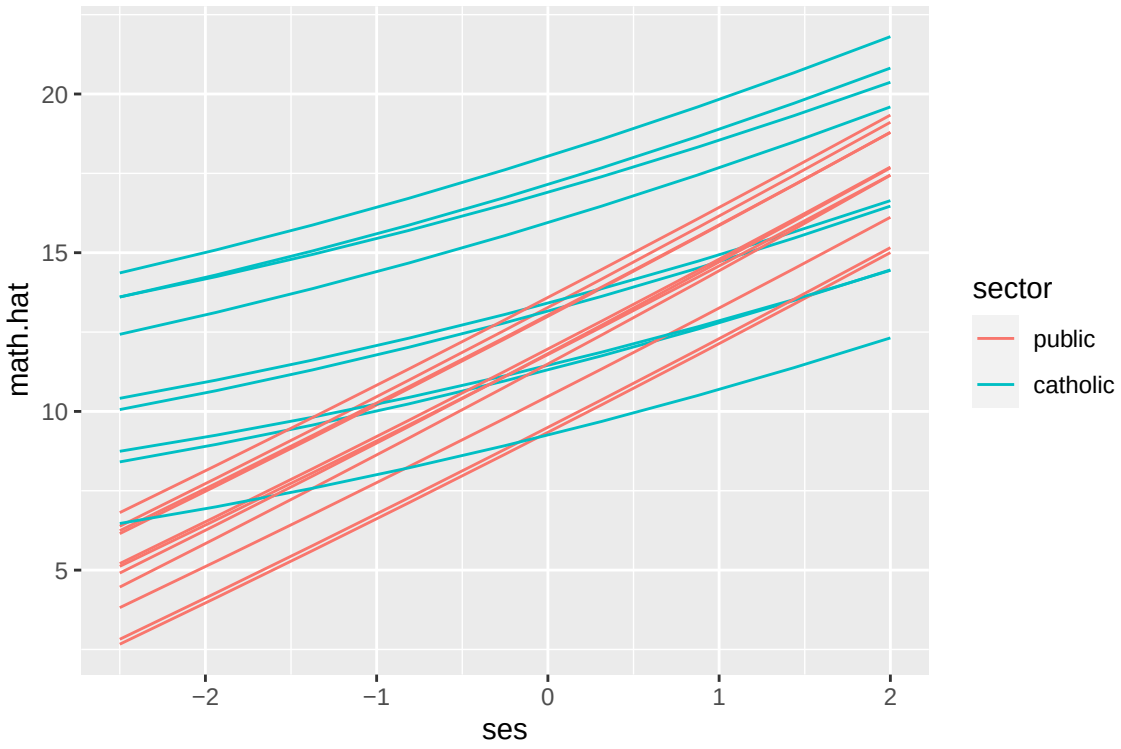
To fit a quadratic model we need our quadratic ses term, which we make by hand. We could also have used `I(ses^2)` in the `lmer()` command directly, but people don't tend to find that easy to read.

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.

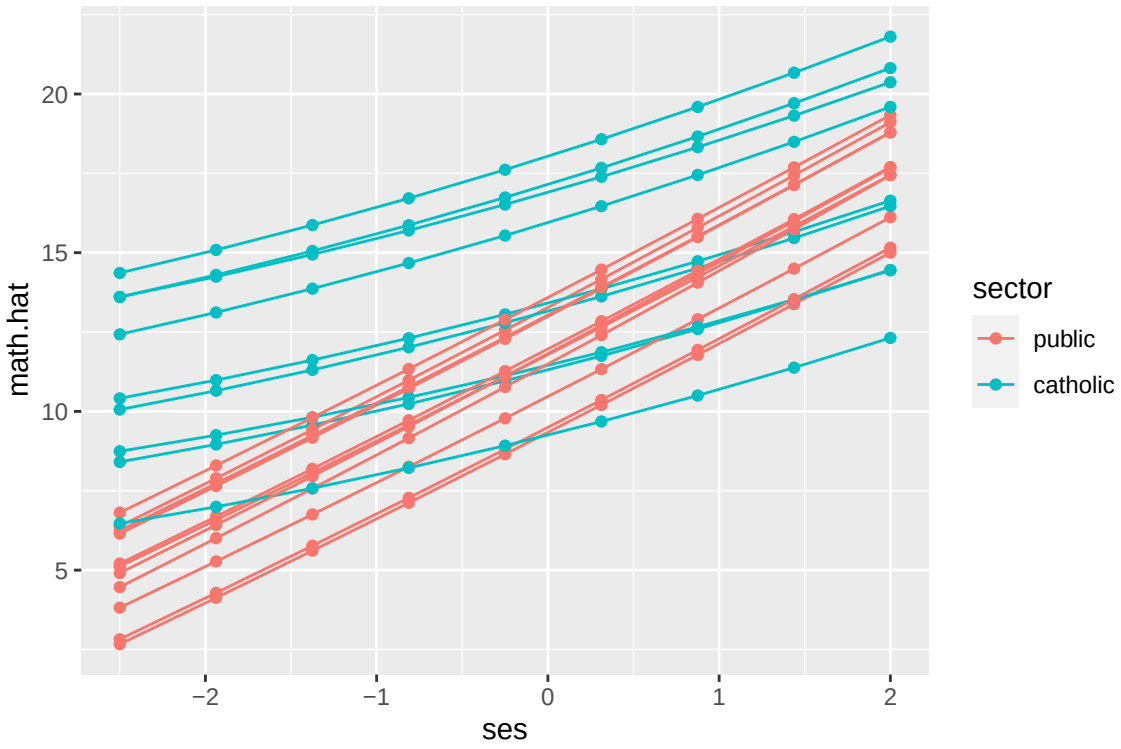
```
synth.dat$math.hat = predict( M2, newdata=synth.dat )

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()
```



14.3 Longitudinal Data

We next do the above, but for longitudinal data. The story is basically the same.

14.3.1 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 )
```

```
       CHILDID     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
```

14.3.2 A model

We will be using the following 3-level quadradic growth model:

```
M4 = lmer( MATH ~ 1 + (YEAR + I(YEAR^2)) * (FEMALE * RACEETH ) +
                (YEAR|CHILDID:SCHOOLID) + (YEAR|SCHOOLID), data=dat )
display( M4 )
```

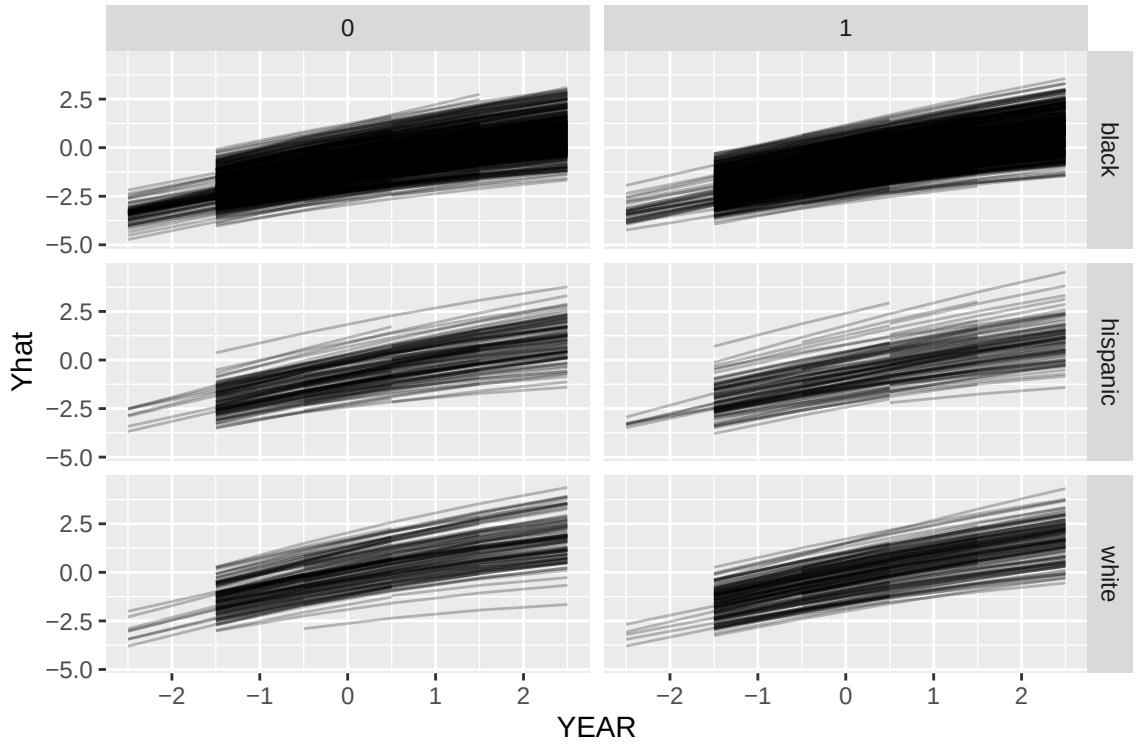
```
lmer(formula = MATH ~ 1 + (YEAR + I(YEAR^2)) * (FEMALE * RACEETH) +
    (YEAR | CHILDID:SCHOOLID) + (YEAR | SCHOOLID), data = dat)
                                coef.est coef.se
(Intercept)                     -0.90     0.06
YEAR                             0.76     0.02
I(YEAR^2)                       -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
 CHILDID: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: CHILDID: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.

14.3.3 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=CHILDID ) ) +
  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.

14.3.4 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 )
```

```
  CHILDID     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
```

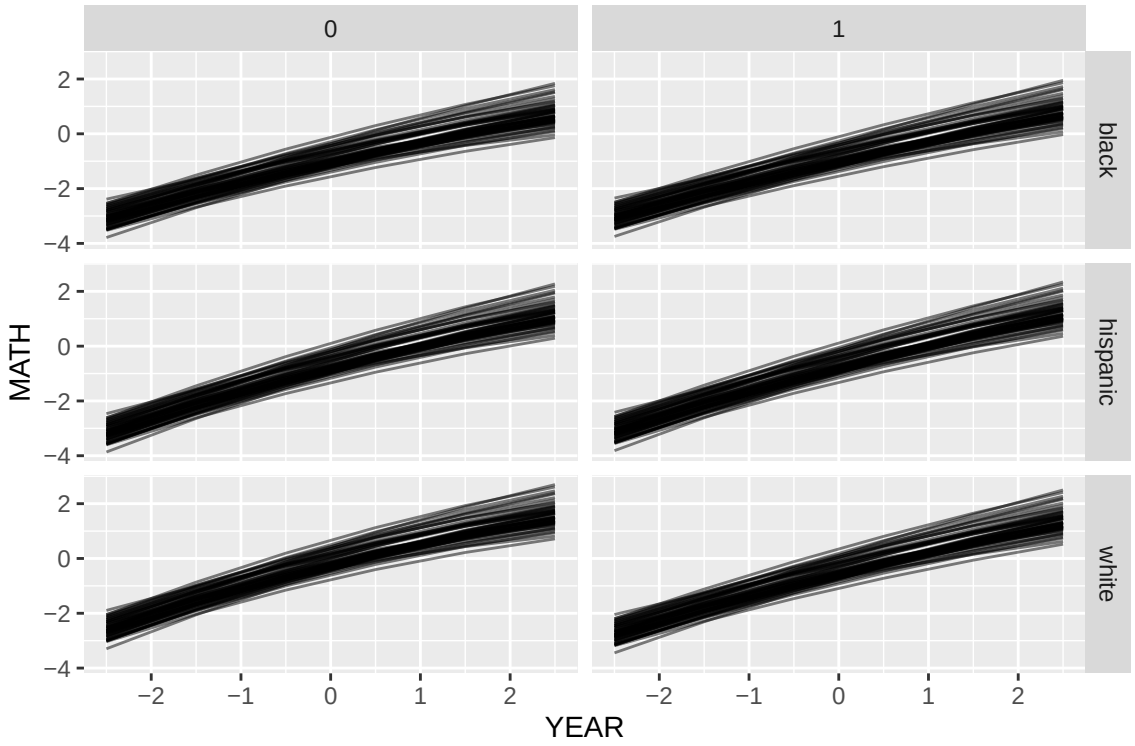
The `CHILDID = -1` line means we are making up a new child (not using one of the real ones) so the child random effects will be set to 0 in the predictions.

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

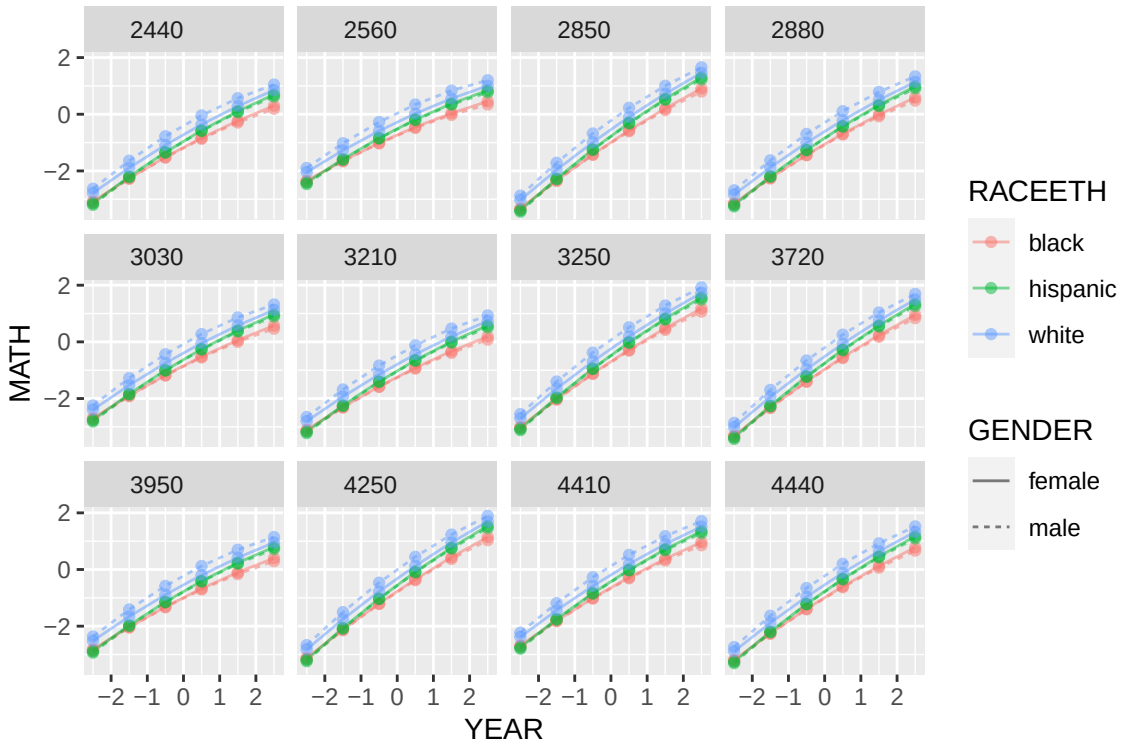
```
ggplot( synth.dat, aes( YEAR, MATH, group=SCHOOLID ) ) +
  facet_grid( RACEETH ~ FEMALE ) +
  geom_line( alpha=0.5 )
```



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.

14.3.5 Population aggregation

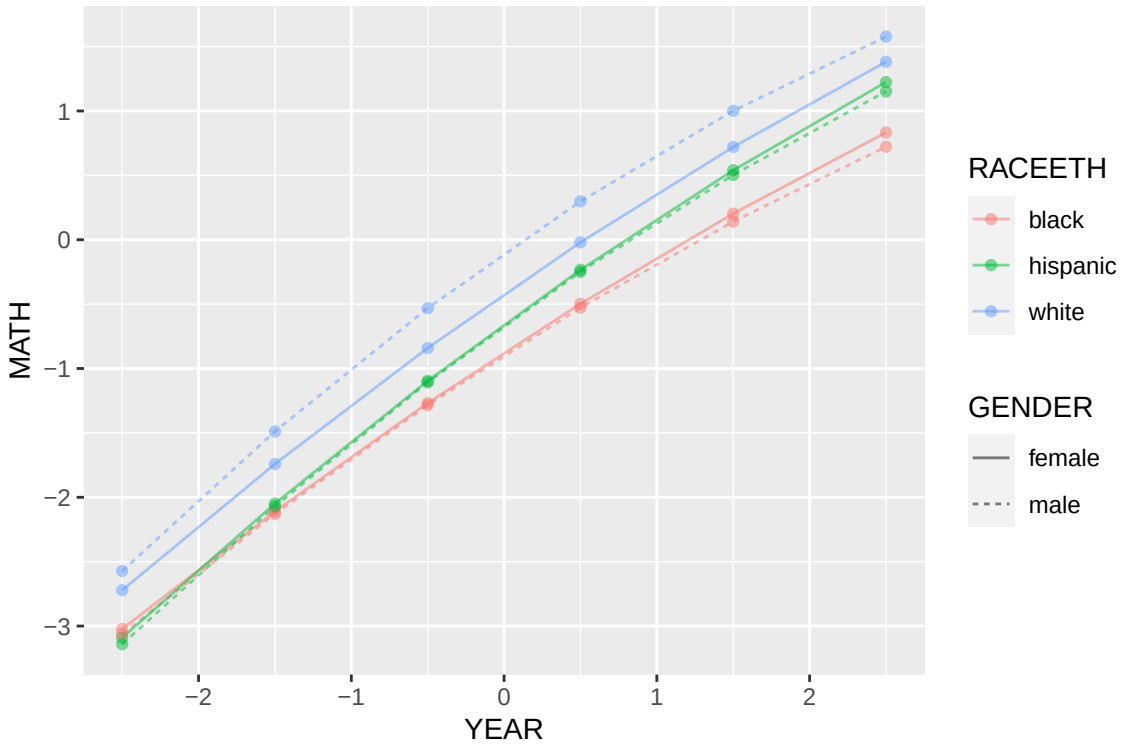
You can also aggregate these predictions. This is the easiest way to get what collection of schools, averaging over their random effects, looks like.

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 )
```

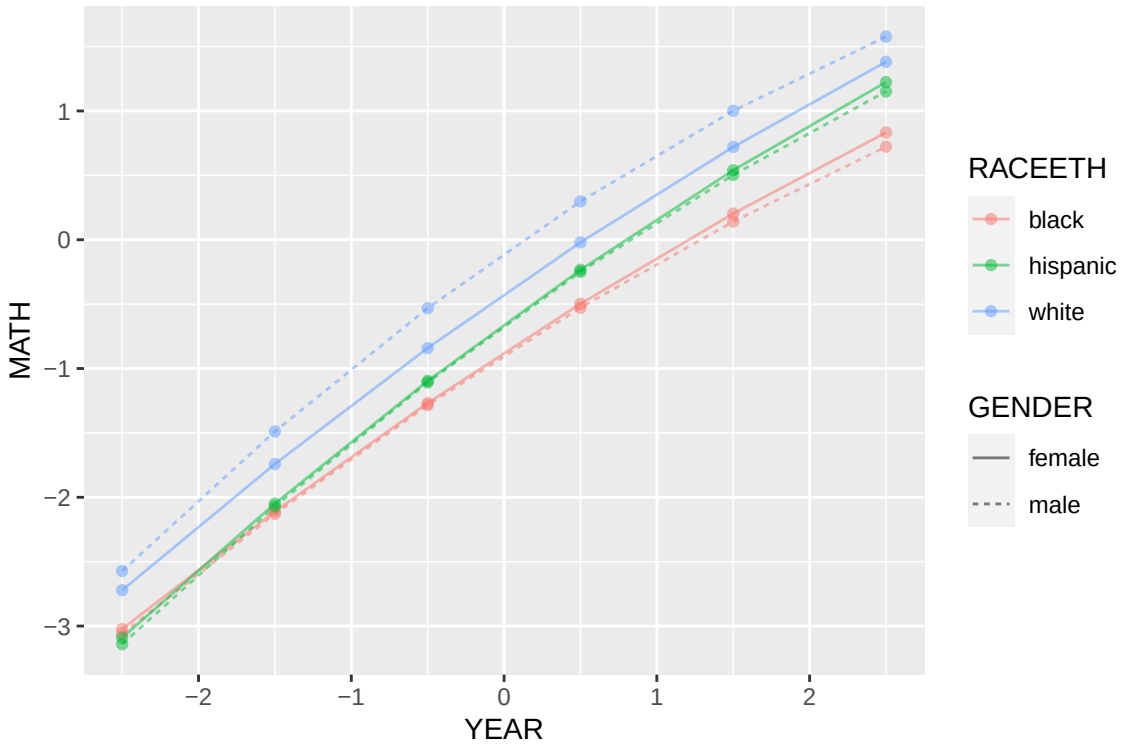


Or do this via predict directly, using the prior ideas

```
synth.dat.agg = expand.grid( CHILDID = -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 )
```

```
  CHILDID SCHOOLID YEAR FEMALE GENDER     black  hispanic      white
1      -1       -1 -2.5      0   male -3.062597 -3.140761 -2.5729327
2      -1       -1 -2.5      1 female -3.022567 -3.090730 -2.7217865
3      -1       -1 -1.5      0   male -2.129829 -2.071721 -1.4888230
4      -1       -1 -1.5      1 female -2.110196 -2.048891 -1.7416614
```

```
5      -1       -1 -0.5      0   male -1.284951 -1.107972 -0.5317700
6      -1       -1 -0.5      1 female -1.268487 -1.096511 -0.8412427
```

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.

14.3.6 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.

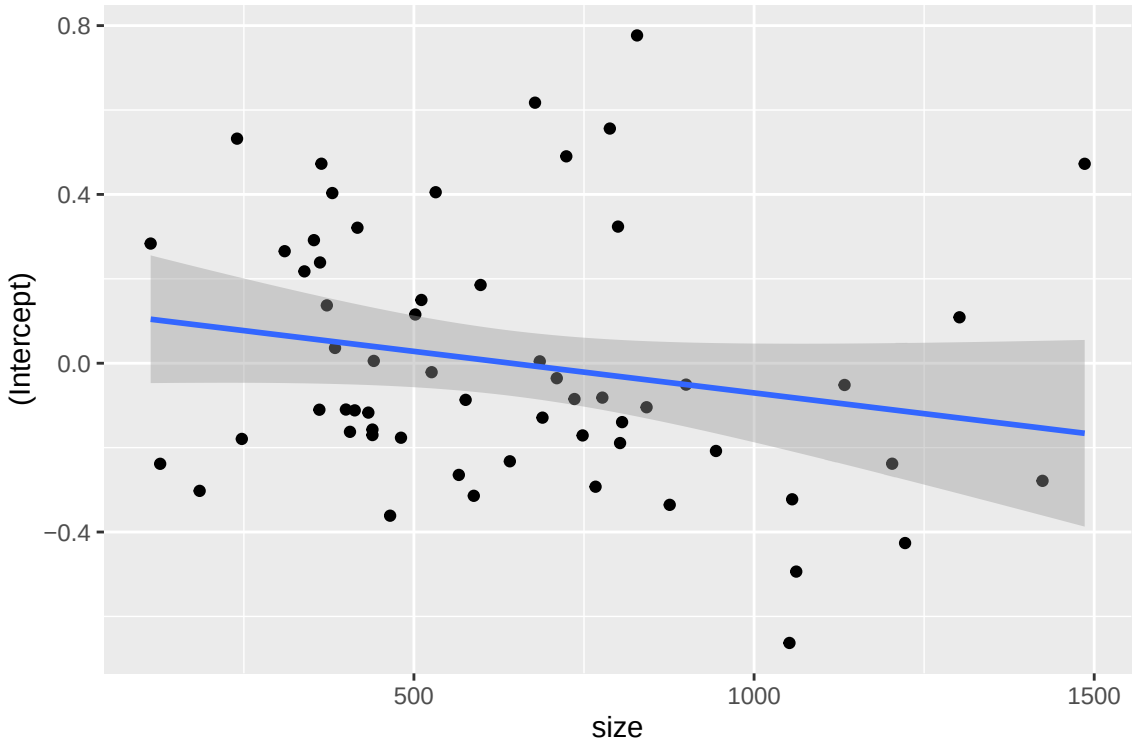
```
ranef = ranef( M4 )$SCHOOLID
ranef$SCHOOLID = rownames( ranef )
schools = dat %>% group_by( SCHOOLID ) %>%
  summarise( n = n(),
             size = SIZE[[1]] )
schools = merge( schools, ranef, by="SCHOOLID" )
head( schools )
```

```
      SCHOOLID   n size (Intercept)        YEAR
1 2020          97  380  0.40323695  0.15257155
2 2040          89  502  0.11549112  0.07547119
3 2180         168  777 -0.08149965 -0.08226575
```

```
4 2330          150  800  0.32372001 -0.04389156
5 2340          220 1133 -0.05151492 -0.01128209
6 2380           87  439 -0.17018815  0.10802722
```

```
ggplot( schools, aes( size, `(Intercept)` ) ) +
  geom_point() +
  geom_smooth(method="lm")
```

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



We see a possible negative trend.

15 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")
```

15.1 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

^{2}

37.03

37.04

36.84

36.63

$_{00}$

4.77 $_{\text{schoolid}}$

3.69 $_{\text{schoolid}}$

3.69 $_{\text{schoolid}}$

3.40 $_{\text{schoolid}}$

ICC

0.11

0.09

0.09

0.09

N

160 $_{\text{schoolid}}$

160 $_{\text{schoolid}}$

160 $_{\text{schoolid}}$

160 $_{\text{schoolid}}$

Observations

7185

7185

7185

7185

Marginal R^2 / Conditional R^2

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

15.2 Graph the Results with `ggeffects`

If we just call `ggeffect` on the model object, we get a bunch of predicted values:

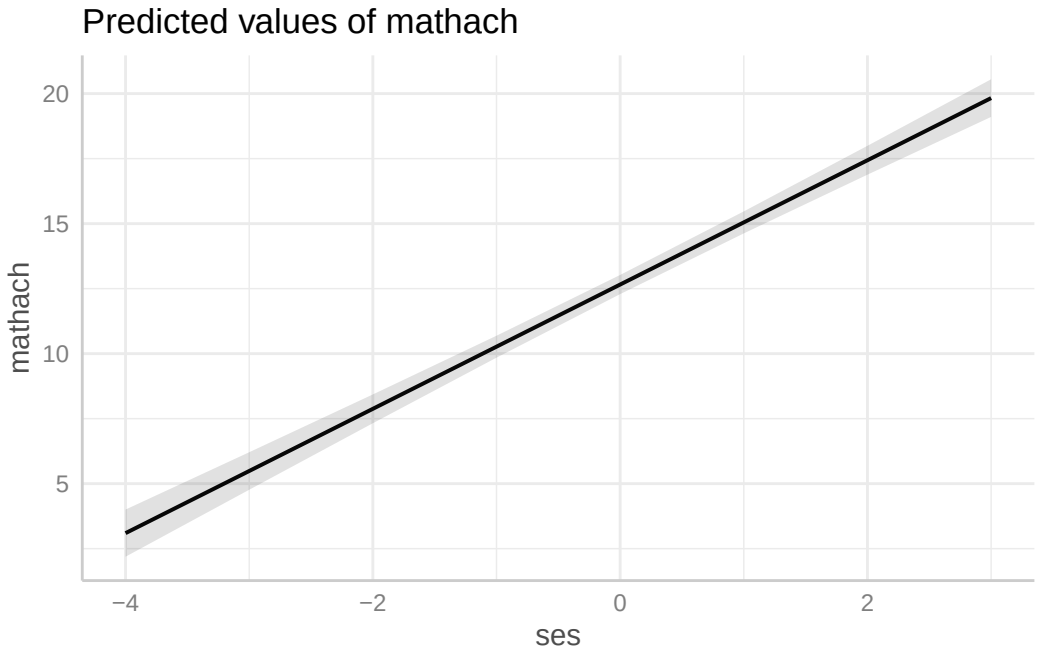
```
ggeffect(m1)
```

```
$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:

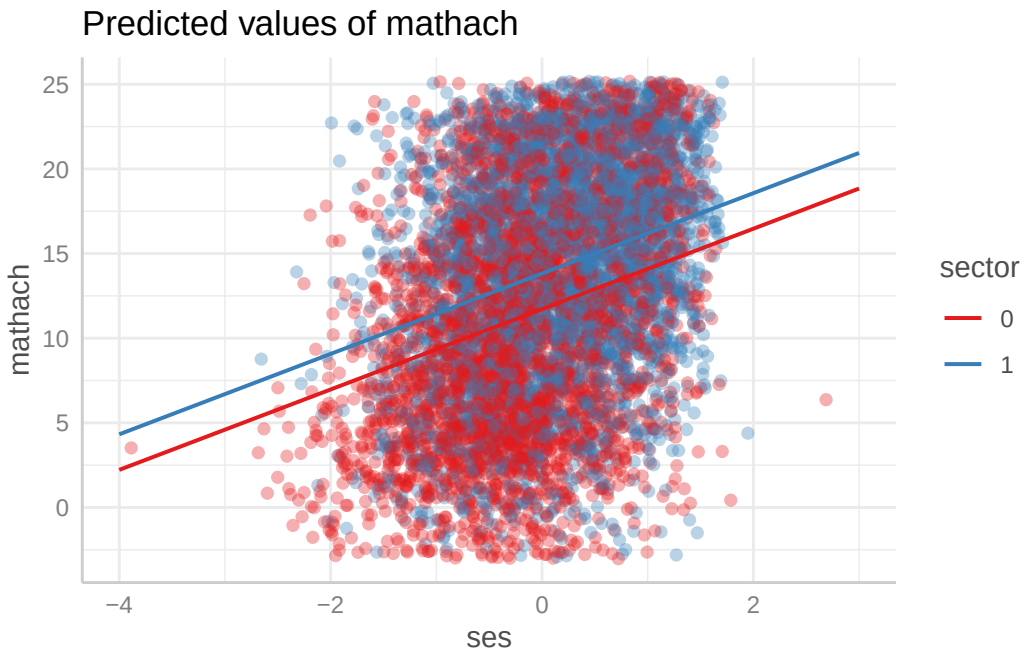
```
ggeffect(m1) |>
  plot()
```

```
$ses
```

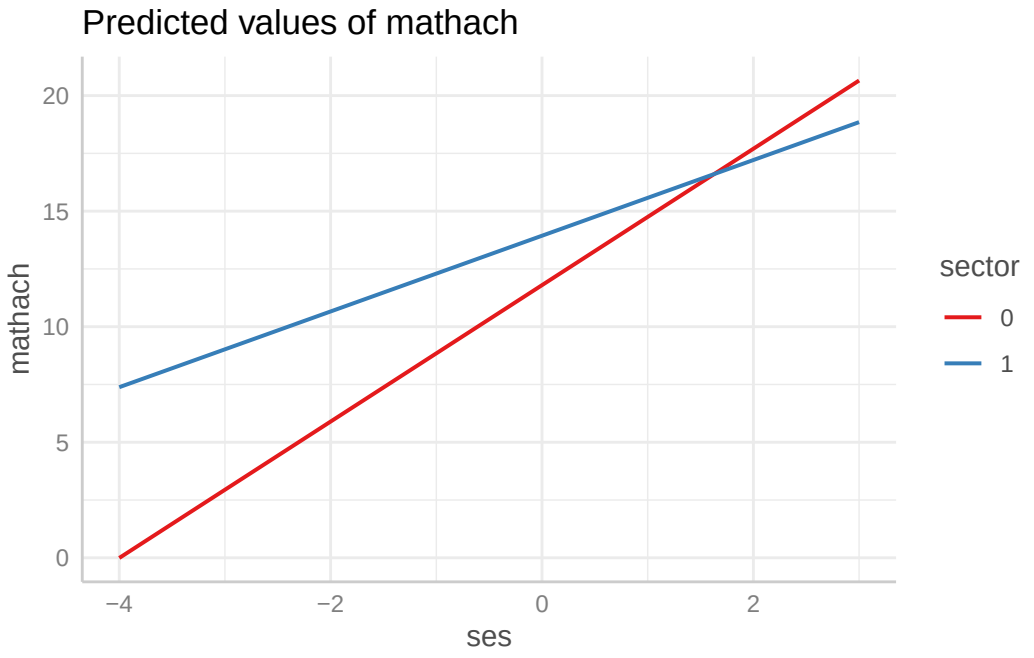


With multiple covariates, we can call the `terms` argument. The first input is on X, the second is mapped to color, the third to facet. This makes visualizing the interactions super easy! Any covariates included in the model but not included in `terms` are held constant at their means.

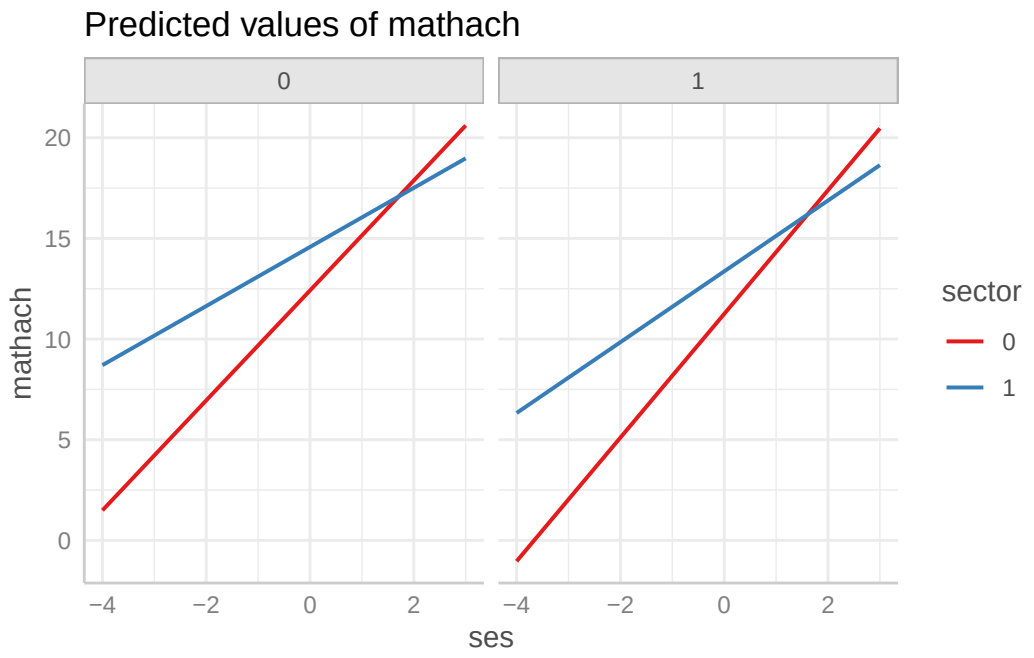
```
ggeffect(m2, terms = c("ses", "sector")) |>
  plot(ci = FALSE, add.data = TRUE)
```




```
ggeffect(m3, terms = c("ses", "sector")) |>
  plot(ci = FALSE)
```



```
ggeffect(m4, terms = c("ses", "sector", "female")) |>
  plot(ci = FALSE)
```



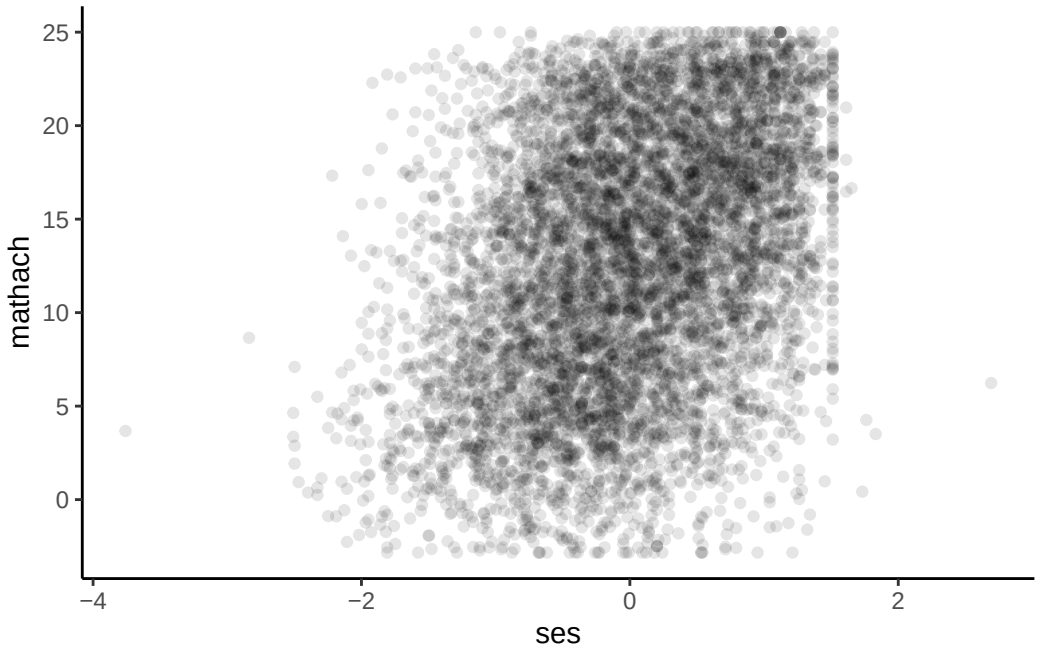
16 Plotting Two Datasets at Once

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.

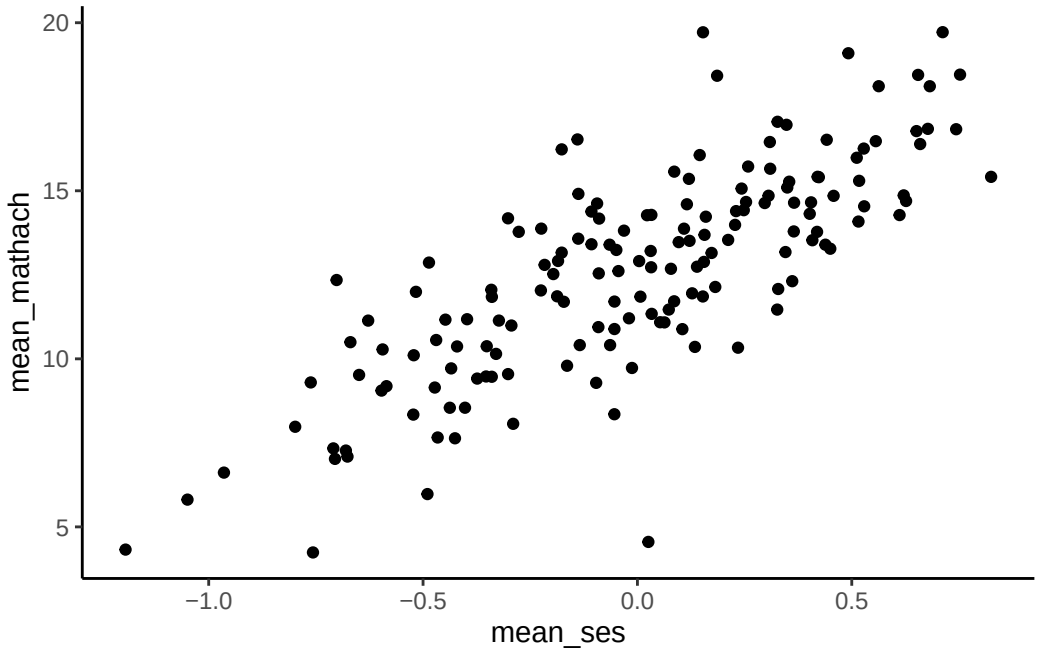
```
library(tidyverse)
library(haven)

# 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)
```



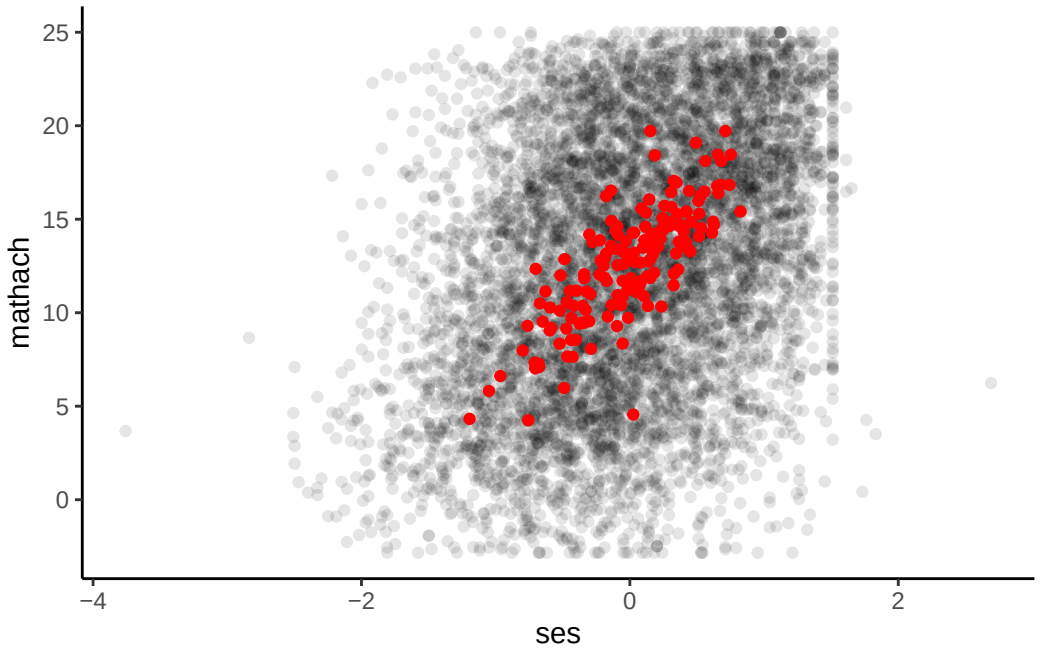

```
ggplot(sch, aes(x = mean_ses, y = mean_mathach)) +
  geom_point()
```



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

17 Extracting model information 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.

17.1 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")
```

17.1.1 `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
```

17.1.2 `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

17.1.3 `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`.

17.2 Extracting `lmer` model info

17.2.1 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
```

17.2.2 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
```

17.2.3 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
1 schoolid (Intercept)  8367 -4.137656 -0.1811498 0.7845770 -5.428170
2 schoolid (Intercept)  4523 -3.089835  0.0235018 0.9819306 -4.704967
3 schoolid (Intercept)  6990 -2.981315 -1.4619679 0.7779876 -4.260991
4 schoolid (Intercept)  3705 -2.811935  0.2776904 1.0911916 -4.606785
5 schoolid (Intercept)  8854 -2.569302 -0.8528365 0.8045804 -3.892719
6 schoolid (Intercept)  9397 -2.431031 -0.6452734 0.9163587 -3.938307
          ub
1 -2.8471413
2 -1.4747032
3 -1.7016394
4 -1.0170840
5 -1.2458846
6 -0.9237553
```

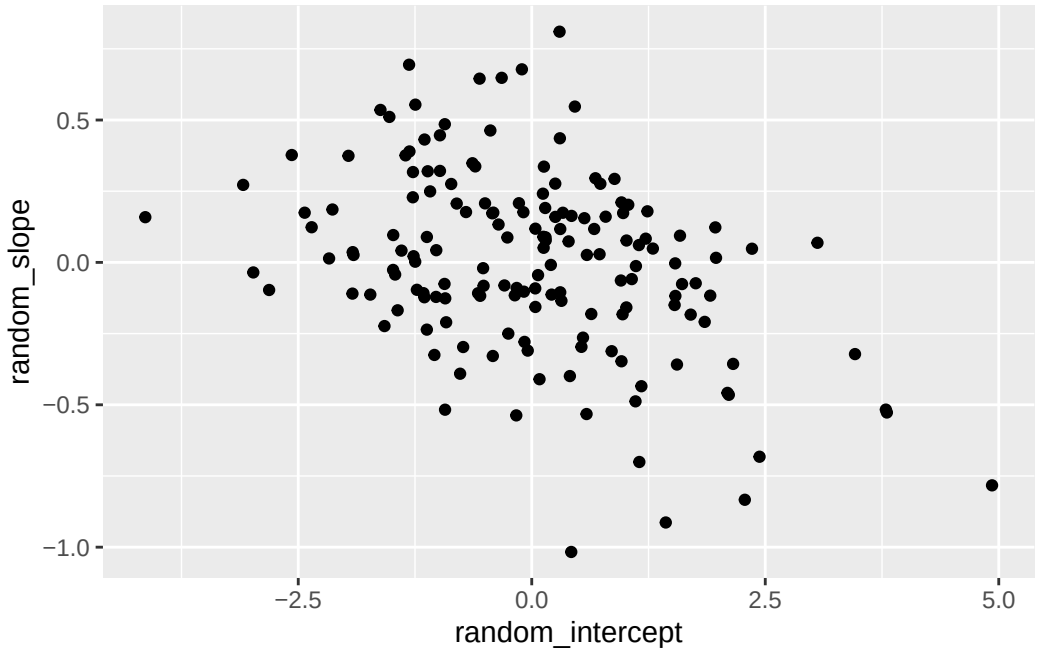
17.2.4 Intercept-Slope Correlation

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.

```r
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
```

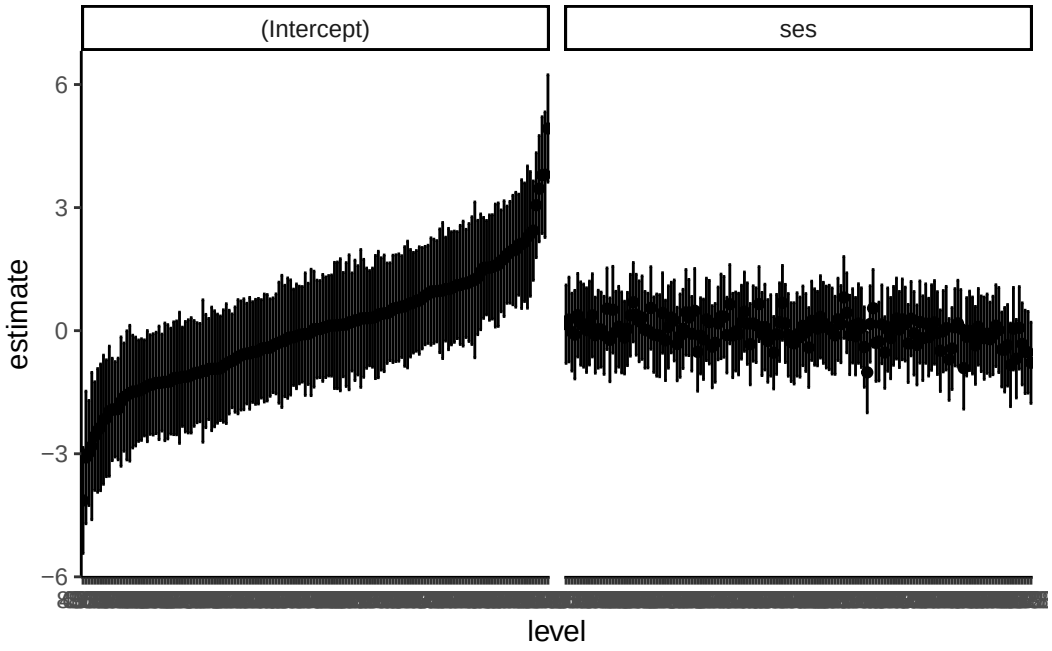
```r
ggplot(blups, aes(x = random_intercept, y = random_slope)) +
  geom_point()
```



17.2.5 Caterpillar Plots

The included information as a data frame makes it easy to construct caterpillar plots!

```
ri <- mlm |>
  ranef() |>
  augment()
ggplot(ri, aes(x = level, y = estimate,
                ymin = lb,
                ymax = ub)) +
  facet_wrap( ~ variable, nrow = 1 ) +
  geom_point() +
  geom_errorbar() +
  theme_classic()
```



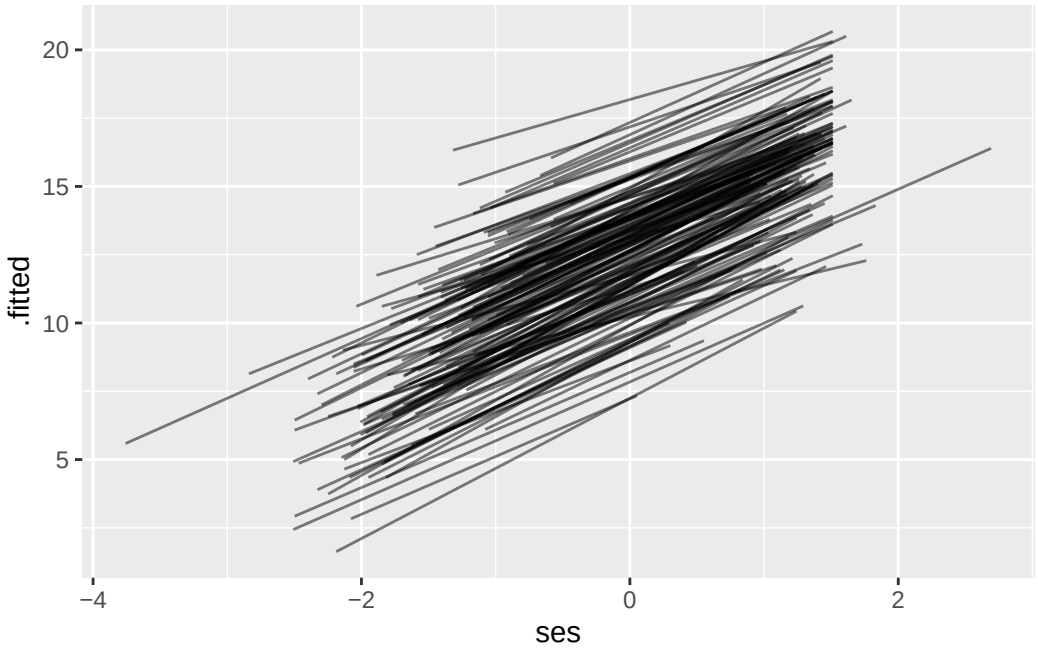
17.2.6 Fitted Values

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()
```

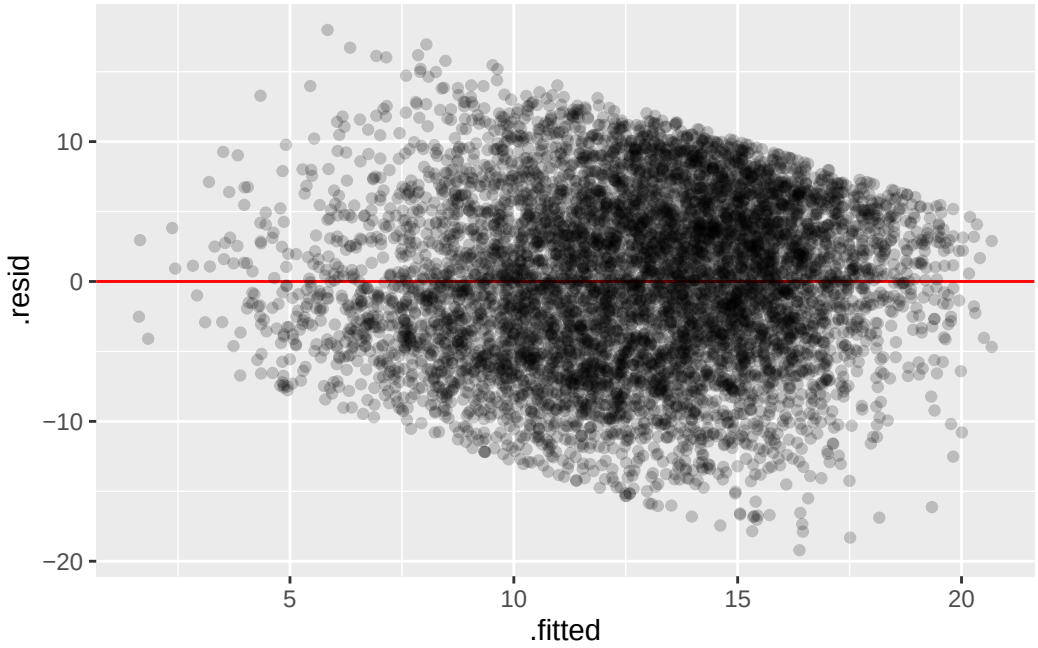
```
# A tibble: 6 x 15
  mathach     ses  mnses schoolid .fitted .resid   .hat   .cooksd .fixed   .mu
    <dbl>   <dbl>  <dbl>    <dbl>   <dbl>  <dbl>  <dbl>     <dbl>  <dbl> <dbl>
1    5.88 -1.53   -0.434     1224    7.29 -1.41  0.0325 0.000629    7.68  7.29
2   19.7  -0.588  -0.434     1224    9.43 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)
```



17.3 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!

18 How to extract information from fitted `lmer` models

Check out the newer chapter on `broom` for a simpler approach to extracting information from `lmer` models.

18.1 Introduction

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.

18.1.1 Libraries

We use the following libraries in this file:

```
library( lme4 )
library( foreign ) ## to load data
library( arm )
library( tidyverse )
```

18.1.2 Loading the data

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
```

18.2 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 )
```

To get an overview of what our fitted model is, use `arm`'s `display()` method:

```
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
```

18.2.1 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
```

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 ['lmerModLmerTest']
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
```

18.3 Obtaining Fixed Effects

R thinks of models in reduced form. Thus when we get the fixed effects we get both the level-1 and level-2 fixed effects

```
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?

18.4 Variance and Covariance estimates of Random Effects

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.

18.4.0.1 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
```

18.5 Obtaining Emperical Bayes Estimates of the Random Effects

Random effects come out of the `ranef()` 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`.

```
ests = ranef( M1 )$id
head( ests )
```

```
     (Intercept)         ses
1224 -0.26204371  0.08765385
1288  0.03805199  0.11841937
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(ests) = c( "u0", "u1" )
ests$id = rownames( ests )
head( ests )
```

```
              u0          u1   id
1224 -0.26204371  0.08765385 1224
1288  0.03805199  0.11841937 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( ests$`(Intercept)` )
```

18.5.1 The `coef()` method

We can also get a slighly 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 incorperated 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 incorperate 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.

18.6 Standard errors

We can get an object with all the standard errors of the coefficients, including the individual Emperical 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 tricker to calculate).

18.6.1 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      meanses
(Intercept)  0.022683560 -0.001465374 -0.001619405
ses         -0.001465374  0.014846788 -0.011954182
meanses     -0.001619405 -0.011954182  0.146389292
```

The standard errors are the diagonal of this matrix, square-rooted. See how they line up?:

```
sqrt( diag( vcov( M1 ) ) )
```

```
(Intercept)         ses     meanses
  0.1506106   0.1218474   0.3826085
```

18.6.2 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
```

18.7 Confidence intervals and uncertainty

We can compute profile confidence intervals (warnings have been suppressed)

```
confint( M1 )
```

```
                  2.5 %     97.5 %
.sig01        1.4012799  1.8897548
.sig02       -0.8733603  0.1945989
.sig03        0.2274189  0.9849964
.sigma        5.9659922  6.1689341
(Intercept) 12.3559620 12.9462385
ses          1.9512025  2.4296954
meanses      3.0278219  4.5329237
```

18.8 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
```

18.9 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"
```

```r
methods(class = "lmerMod")
```

```
 [1] coerce      coerce<-    contest     contest1D   contestMD   display
 [7] getL        mcsamp      se.coef     show        sim         standardize
see '?methods' for accessing help and source code
```

```r
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] fortify        getData        getL           getME          hatvalues
[21] influence      isGLMM         isLMM          isNLMM         isREML
[26] logLik         mcsamp         model.frame    model.matrix   ngrps
[31] nobs           plot           predict        print          profile
[36] ranef          refit          refitML        rePCA          residuals
[41] rstudent       se.coef        show           sigma.hat      sigma
[46] sim            simulate       standardize    summary        terms
[51] update         VarCorr        vcov           weights
see '?methods' for accessing help and source code
```

19 Clarification on Fixed Effects and Identification

19.1 The language of "Fixed Effects"

I wanted to follow-up on a couple of things that I had written down but not sent out.

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 $\tau_{00}, \tau_{01}, \tau_{11}$.

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.

19.2 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.

19.3 Further Reading

(Antonakis, Bastardoz, and Rönkkö 2019)

20 Interpreting Coefficients

20.1 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

$$\begin{aligned} mathach_{ij} &= \beta_{0j[i]} + \beta_{1j[i]} SES_i + \varepsilon_i, \\ \beta_{0j} &= \gamma_{00} + \gamma_{01} sector_j + u_{0j}, \\ \beta_{1j} &= \gamma_{10} + \gamma_{11} sector_j + u_{1j}, \\ \varepsilon_i &\sim Normal(0, \sigma^2_\varepsilon) \\ \begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} &\sim N\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} \sigma^2_0 & \rho\sigma_0\sigma_1 \\ \rho\sigma_0\sigma_1 & \sigma^2_1 \end{pmatrix} \right] \end{aligned}$$

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 ith 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 jth 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 indicates 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 is 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 pth 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 pth and qth 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

$$\beta_{0j} = \gamma_{00} + \gamma_{01} sector_j + u_{0j}.$$

We can interpret γ_{00} as the predicted intercept for schools for which $sector = j$ (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

$$\beta_{1j} = \gamma_{10} + \gamma_{11} sector_j + u_{0j}.$$

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, $u_1 j$ 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 $\beta_{0j} = \gamma_{00} + \gamma_{01} sector_j + u_{0j}$ and $\beta_{1j} = \gamma_{10} + \gamma_{11} sector_j + u_{1j}$ to obtain

$$\begin{aligned} mathach_i &= \gamma_{00} + \gamma_{01} sector_{j[i]} + u_{0j[i]} + (\gamma_{10} + \gamma_{11} sector_{j[i]} + u_{1j[i]}) SES_i + \varepsilon_i \\ &= \gamma_{00} + \gamma_{01} sector_{j[i]} + \gamma_{10} SES_i + \gamma_{11} sector_{j[i]} SES_i + (u_{0j[i]} + u_{1j[i]} SES_i + \varepsilon_i). \end{aligned}$$

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 $SES = 0$ in public schools;
2. γ_{01} is the predicted difference in *mathach* between students of $SES = 0$ 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

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

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.

21 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)
```

21.1 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

^{2}

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

R^2 / R^2 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

21.2 Interpretation

21.2.1 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.

21.2.2 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.

21.2.3 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.

21.2.4 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.

21.2.5 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.

21.2.6 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!

21.2.7 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.

21.3 Further Reading

(Antonakis, Bastardoz, and Rönkkö 2019)

22 Predictors in Longitudinal Growth Models

22.1 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.

22.2 Additional Resources

- https://books.google.com/books?i d=PpnA1M8VwR8C&pg=PA168&lpg=PA168&dq=longitudinal+data QquJF1eE&hl=en&sa=X&ved=0CCwQ6AEwAWoVChMI5ZLsjKDjyAIVzB0-Ch1s6wGV#v=onepage&q
- http://jonathantemplin.com/files/mlm/mlm12uga/mlm12uga_section06.pdf
- http://www.lesahoffman.com/944/944_Lecture07_Time-Invariant.pdf

23 MLM Assumptions

23.1 Oh, assumptions

There are generally two kinds of assumptions we should worry about the most: ommitted 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.

23.2 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:

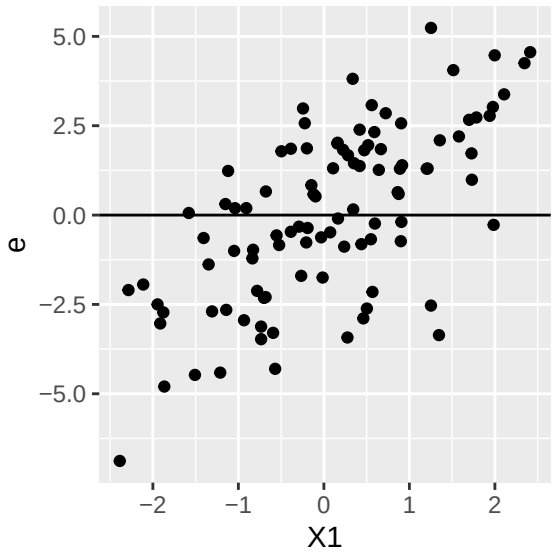
```
tab_model(M0, M1, p.style = "stars",
          show.ci = FALSE, show.se = TRUE)
```

	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 / 0.853		0.584 / 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 `X2`) are positive for bigger `X1`, due to the correlation of `X1` and `X2`.

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 `X2` on `X1`.

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.

23.3 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.

23.4 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

23.5 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.

24 Connecting the three dots: An HSB Model

This handout shows (1) the mathematical model, (2) the `lmer` syntax, and (3) the output for the model, as discussed in class in Lecture 2.4. This handout is designed to help translate between these three different worlds.

24.1 The mathematical model

Level 1 models:

$$y_{ij} = \beta_{0j} + \beta_{1j} ses_{ij} + \beta_2 female_{ij} + \epsilon_{ij}$$
$$\epsilon_{ij} \sim N(0, \sigma_y^2)$$

Level 2 models:

$$\beta_{0j} = \gamma_{00} + \gamma_{01} sector_j + \gamma_{02} meanSES_j + u_{0j}$$
$$\beta_{1j} = \gamma_{10} + \gamma_{11} sector_j + u_{1j}$$

with

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

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:

$$\begin{aligned}
y_{ij} &= \beta_{0j} + \beta_{1j} ses_{ij} + \epsilon_{ij} \\
&= (\gamma_{00} + \gamma_{01} sector_j + \gamma_{02} meanSES_j + u_{0j}) + (\gamma_{10} + \gamma_{11} sector_j + u_{1j}) ses_{ij} + \beta_2 female_{ij} + \epsilon_{ij} \\
&= \gamma_{00} + \gamma_{01} sector_j + \gamma_{02} meanSES_j + u_{0j} + \gamma_{10} ses_{ij} + \gamma_{11} sector_j ses_{ij} + u_{1j} ses_{ij} + \beta_2 female_{ij} + \epsilon_{ij} \\
&= \gamma_{00} + \gamma_{01} sector_j + \gamma_{02} meanSES_j + \gamma_{10} ses_{ij} + \gamma_{11} sector_j ses_{ij} + \beta_2 female_{ij} + (u_{0j} + u_{1j} ses_{ij} + \epsilon_{ij})
\end{aligned}$$

This formula is what we will give to `lmer()` in R's formula notation.

24.2 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.

24.3 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 equivilent 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.

25 Model Representations

25.1 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).

25.2 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

$$Y_{ij} = \alpha_j + \beta_1 R_{ij} + \beta_2 X_{ij} + \epsilon_{ij}$$
$$\alpha_j = \gamma_0 + \gamma_1 Z_j + \gamma_2 S_j + u_j$$

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()`.