Handouts for S022 (Data Science in Education)

Luke Miratrix (with some others for some parts)

2023-04-19

Table of contents

Preface

This "book" is basically a set of handouts generated for a Data Science in Education course taught at the Harvard Graduate School of Education by Luke Miratrix. They are loosely organized, but the primary purpose is to make all the handouts easy to access and find.

The handouts were generated in response to student questions, and aim to short-circuit staring at the possibly confusing world of Stack Overflow. They are curated in the sense that these activities tend to come up over and over in the course of fundamental data science work.

I hope you find this resource useful. Please send feedback to `lmiratrix@g.harvard.edu`.

Part I

On markdown

1 How to Quarto

2 Creating Reports in Quarto

2.1 What is Quarto?

Quarto is the "next-generation" of RMarkdown. Most tutorials on Quarto are intended for people who have experience with RMarkdown, so a great place to start is to watch Luke's video on RMarkdown. You can find it on this page, under the "Do it for Lab" tab.

Once you're familiar with RMarkdown, the tricks you know will almost always work in Quarto. The major difference is that Quarto will render *inside* RStudio, as you write your document, which makes it a little nicer to work with. There are other new (and very cool) features, but they aren't essential for anything in this course.

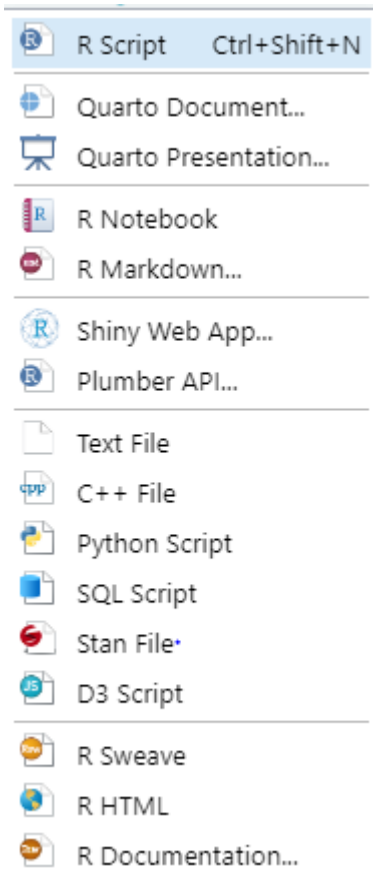
2.2 Using Visual Editor with Rmd

An obvious, cool thing about Quarto is the fancy visual editor, but you can turn this feature on for regular old RMarkdown files, too. Just click on this  , and click "use visual editor." You can also add `editor: visual` to the yaml (top-matter) of an rmd file and it'll open in the visual editor by default.

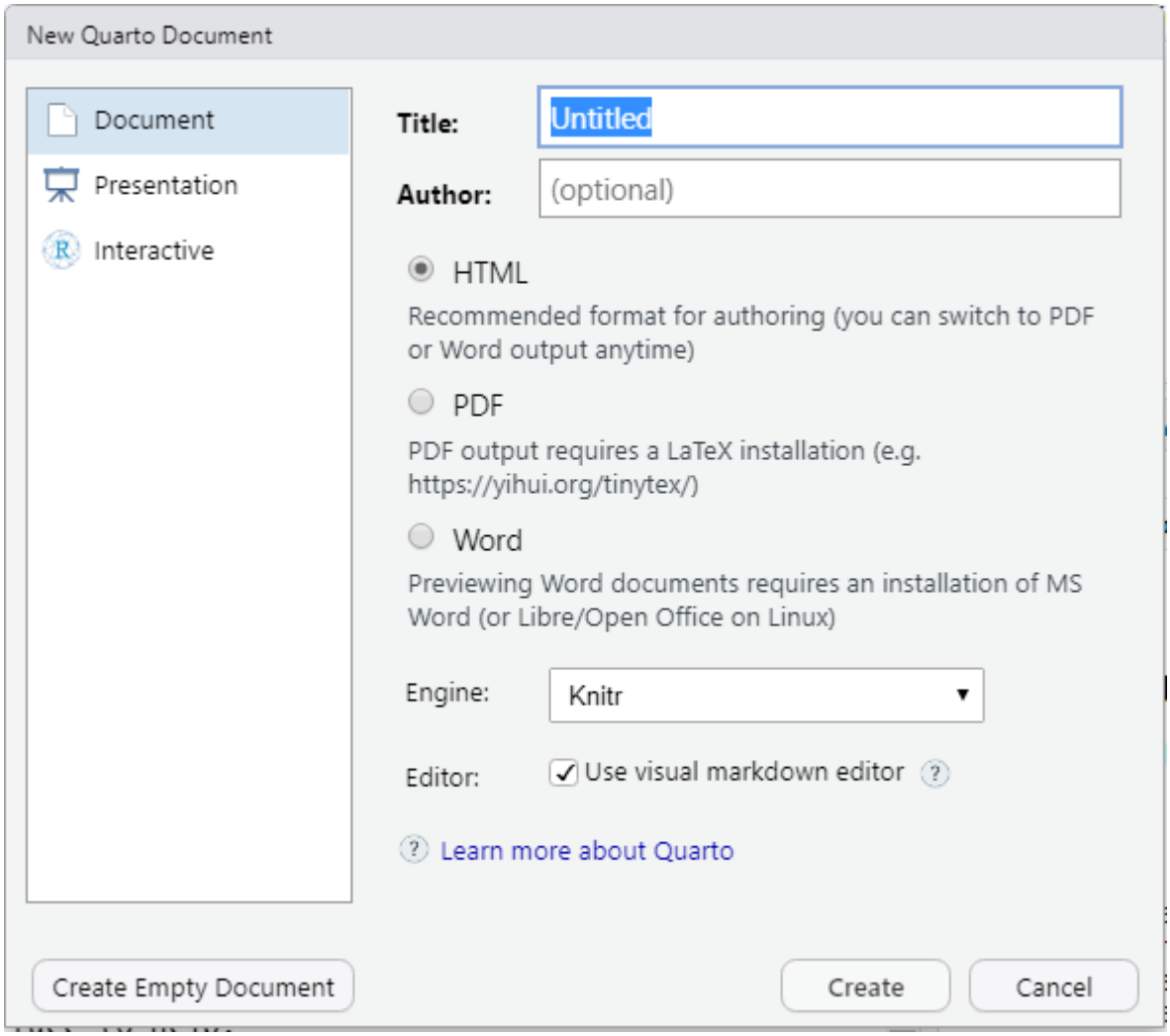
2.3 Creating a Quarto Report

In RStudio, click this icon in the upper left: 

That'll give you the following drop-down menu, where you can select "Quarto Document":



When you click on "Quarto Document..." RStudio might take a few seconds to load. Then you'll see this pop-up:



Fill out the document title and author (just like for RMarkdown). You can always change the title and author later. You'll want to render your reports as PDFs, so select that option. Finally, hit the "Create" button at the bottom.

RStudio will load up a new Quarto doc (with some boilerplate markdown in it). From here, you can treat it like an RMarkdown file.

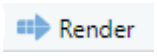
If you want a more thorough introduction to Quarto, check out this tutorial.

2.4 Rendering to PDF

To render your report to PDF, you'll need to have an installation of LaTex. You can set this up from within RStudio.

Down by your console, there's a tab called "Terminal." Click on it to open the terminal. Inside the terminal type the following:
`quarto install tool tinytex`

I recommend restarting your computer after this.

Now you should be able to click 

and get a pdf version of your report. By default, a copy of the pdf will be saved in the same folder as your Quarto document.

2.5 A Couple Quick Tricks

2.5.0.1 Making Code Chunks

Use the keyboard shortcut `ctrl+alt+i` or `command + option + i` to create a new R code chunk.

2.5.0.2 Adding Images

You can copy-paste or drag-and-drop images into a Quarto doc. That's how I put the above screenshots into this document.

2.6 Additional Quarto Resources

Here are a couple more links if you'd like to learn more:

- Grab the RMarkdown Cheatsheet (lots of other great cheatsheets there, too). Most things that work in RMarkdown work in Quarto.
- Quarto Intro Video This video walks through using Quarto for the first time.
- In Depth Quarto Intro This covers the amazing new features (like making interactive html reports).

3 Working with Rmarkdown chunks

3.1 Markdown chunks

3.1.1 Options for including/suppressing code and output

`include`: Should chunk be included in knit file? Defaults to `TRUE`. If `FALSE`, code chunk is run, but chunk and any output is not included in the knit file.

`eval`: Should chunk be evaluated by R? Defaults to `TRUE`. If `FALSE`, code chunk is included in the knit file, but not run.

`echo`: Should the code from this chunk be included in knit file along with output? Defaults to `TRUE`. If `FALSE`, the output from the chunk is included, but the code that created it is not. Most useful for plots.

3.1.2 Options for including/suppressing R messages

R has "errors" meaning it could not run your code, "warnings" meaning that the code was wrong, but there are some potential issues with it, and "messages" which are simply information about what your code ran. You can include or suppress each of these types of message.

`error`: Should R continue knitting if code produces an error? Defaults to `FALSE`. Generally don't want to change this because it means you can miss serious issues with your code.

`warning`: Should R include warnings in knit file? Defaults to `TRUE`.

`message`: Should R include informational messages in knit file? Defaults to `TRUE`. Easy way to clean up your markdowns.

```
#This code produces an error
dat %>%
  filter(dest = 1)
```

```
Error in `filter()`:
! We detected a named input.
i This usually means that you've used `=` instead of `==`.
i Did you mean `dest == 1`?
```

```r
#Example warning
parse_number(c("1", "$3432", "tomato"))
```

```
[1]    1 3432   NA
attr(,"problems")
# A tibble: 1 x 4
    row   col expected actual
  <int> <int> <chr>    <chr>
1     3    NA a number tomato
```

```r
#Example message
library(gridExtra)
```

3.1.3 Options for modifying figure outputs

`out.width`: What percentage of the page width should output take?

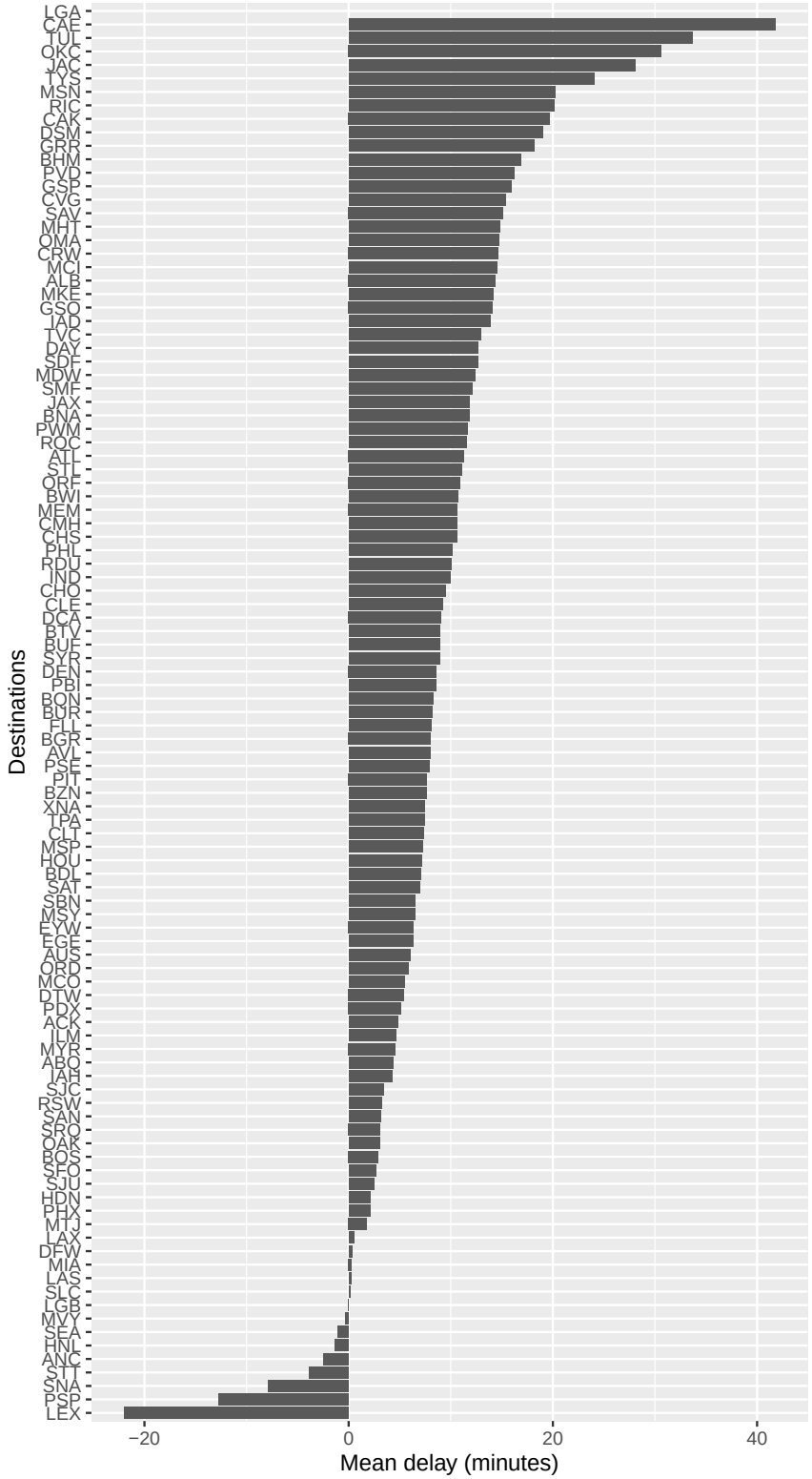
`fig.height`: What should be the height of figures?

`fig.width`: What should be the width of figures?

`fig.asp`: What should be the aspect ratio of figures?

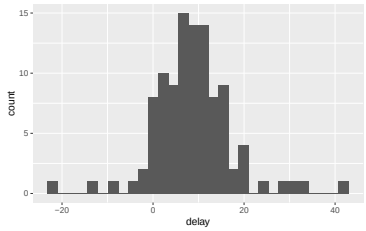
`fig.align`: How should figures be aligned?

We might want a bigger plot for this:


Destinations
LGA
CAE
TUL
OKC
JAC
TYS
MSN
RIC
CAK
DSM
GRR
BHM
PVD
GSP
CVG
SAV
MHT
OMA
CRW
MCI
ALB
MKE
GSO
IAD
TVC
DAY
SDF
MDW
SMF
JAX
BNA
PWM
ROC
ATL
STL
ORF
BWI
MEM
CMH
CHS
PHL
RDU
IND
CHO
CLE
DCA
BTV
BUF
SYR
DEN
PBI
BON
BUR
FLL
BGR
AVL
PSE
PIT
BZN
XNA
TPA
CLT
MSP
HOU
BDL
SAT
SBN
MSY
EYW
EGE
AUS
ORD
MCO
DTW
PDX
ACK
ILM
MYR
ABQ
IAH
SJC
RSW
SAN
SRQ
OAK
BOS
SFO
SJU
HDN
PHX
MTJ
LAX
DFW
MIA
LAS
SLC
LGB
MVY
SEA
HNL
ANC
STT
SNA
PSP
LEX
−20
0
20
40
Mean delay (minutes)

And a smaller plot for this:

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




```
$x
[1] "Minutes of delay"

$y
[1] "Destimations"

attr(,"class")
[1] "labels"
```

3.1.4 Change your defaults

At the beginning of your code, you can set custom defaults. This is handy and means you don't need to repeat the custom arguments in each code chunk.

```
knitr::opts_chunk$set(echo = TRUE,
                      fig.width = 5,
                      fig.height = 3,
                      out.width = "5in",
                      out.height = "3in", fig.align = "center")
```

Part II

On Plotting

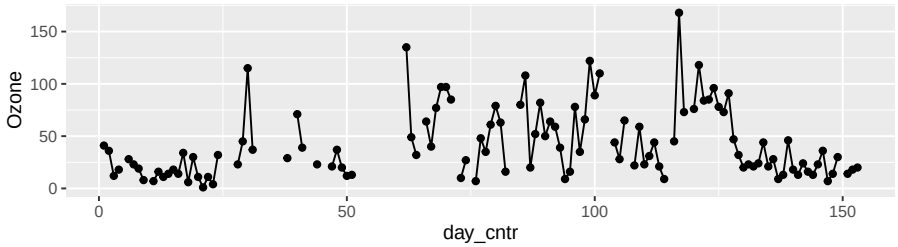
4 Simple plotting tips

This document provides several simple plotting tips for using RStudio to make plots (primarily using the `ggplot2` package in `tidyverse`). These are all very simple tricks that can radically enhance the legibility of a plot. This document also covers how to save your plots so you can use them in presentations and other documents, and how to control the size of your plot in a R Markdown document.

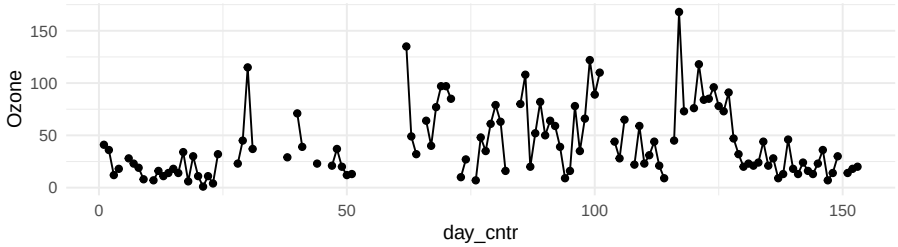
5 Setting a theme

Themes in ggplot give you a bunch of default options for plotting. For example, you can get rid of the grey background of plots with the `theme_minimal()` command.

```
airquality$day_cntr = 1:nrow(airquality)
ggplot( airquality, aes( day_cntr, Ozone ) ) +
    geom_point(na.rm=TRUE) + geom_line()
```

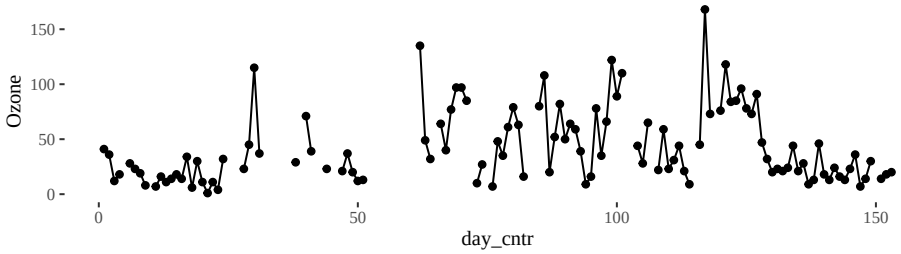



```
ggplot( airquality, aes( day_cntr, Ozone ) ) +
    geom_point(na.rm=TRUE) + geom_line() +
    theme_minimal()
```



I also quite like Tufte theme in the `ggthemes` bonus library (this library is a library of various themes with styles good and bad):

```
ggplot( airquality, aes( day_cntr, Ozone ) ) +
    geom_point(na.rm=TRUE) + geom_line() +
    ggthemes::theme_tufte()
```



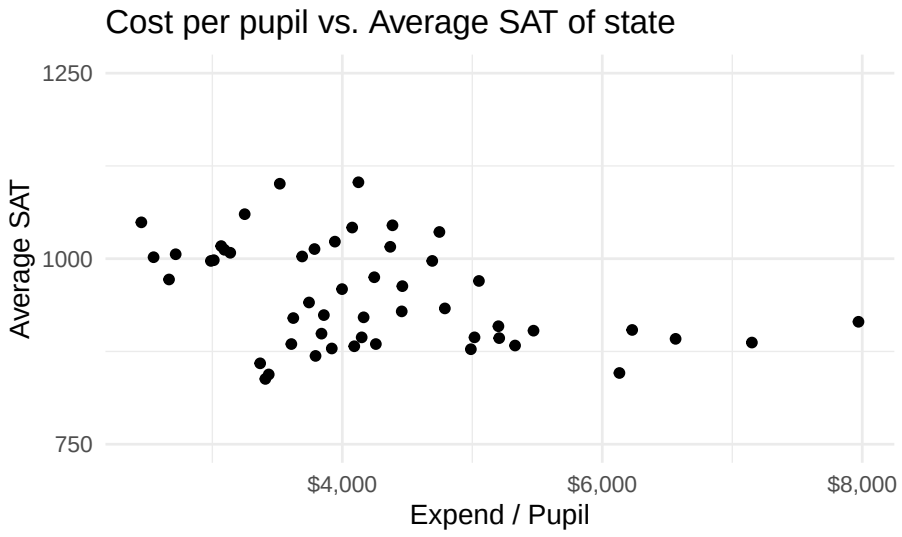
You can also set the theme globally:

```
theme_set( theme_minimal() )
```

6 Formatting axes and making labels

The `labs()` command is critical for getting labels for your plot. The `scale_x_continuous()` (or y) allow you to control the scales on each axis. Also, there is a nice package, `scales` that will format your x and y-axes. Witness!

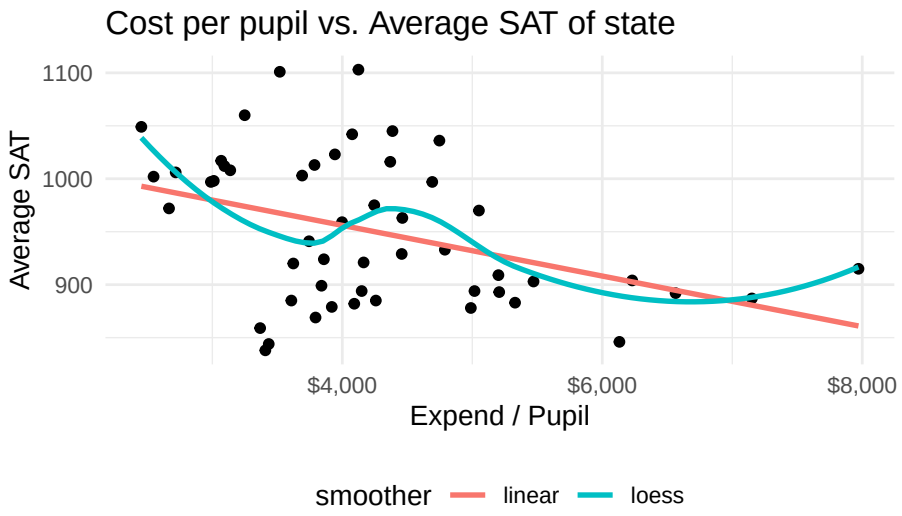
```
library( scales ) # for label_dollar(), below
ggplot( sat93, aes( expend, tot_sat ) ) +
  geom_point() +
  scale_x_continuous( labels = label_dollar() ) +
  labs( title = "Cost per pupil vs. Average SAT of state",
        y = "Average SAT",
        x = "Expend / Pupil" ) +
  theme_minimal() +
  scale_y_continuous( limits = c( 750, 1250),
                      breaks = c( 750, 1000, 1250 ) )
```



7 Making legends using aes()

If you are making a bunch of lines and want to color them, you can do so by giving them all names and then letting R pick the colors for you. This also lets you have a nice legend telling you which color is which. Also note how you can put a name for your legend inside `labs()`. Witness!

```
library( scales ) # for label_dollar(), below
ggplot( sat93, aes( expend, tot_sat ) ) +
  geom_point() +
  scale_x_continuous( labels = label_dollar() ) +
  labs( title = "Cost per pupil vs. Average SAT of state",
        y = "Average SAT",
        x = "Expend / Pupil",
        color = "smoother") +
  geom_smooth( aes( col = "linear" ), method="lm", se=FALSE ) +
  geom_smooth( aes( col = "loess" ), se=FALSE ) +
  theme_minimal() +
  theme(legend.position = "bottom" )
```

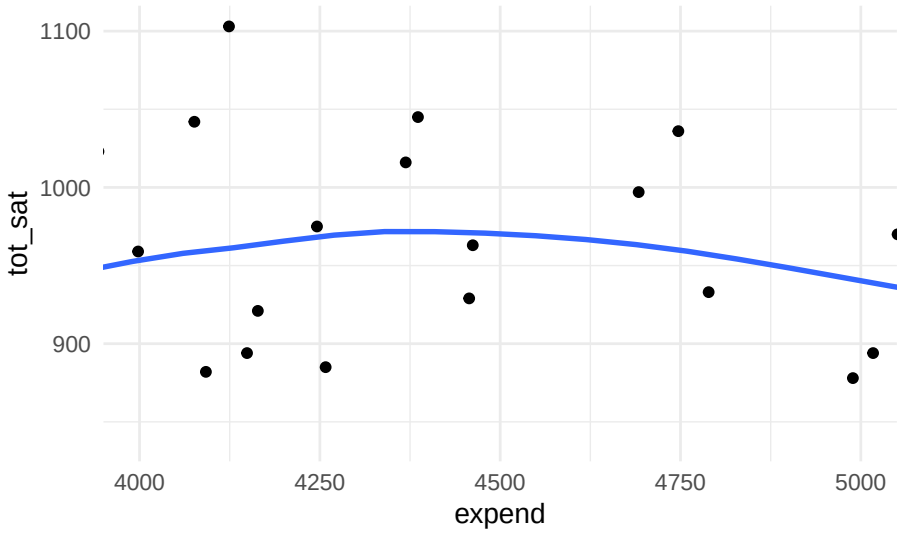


8 Cropping vs. zooming in

If you set the scale of your plot via `scale_x_continuous()` you will drop data from your analysis. If you use `coord_cartesian()` you will zoom in on the specified window, but it will be using all the data for smoothers, etc. This can be an important difference:

```
ggplot( sat93, aes( expend, tot_sat ) ) +
  geom_point() +
  geom_smooth( se=FALSE ) +
  coord_cartesian( xlim=c(4000, 5000 ) )
```

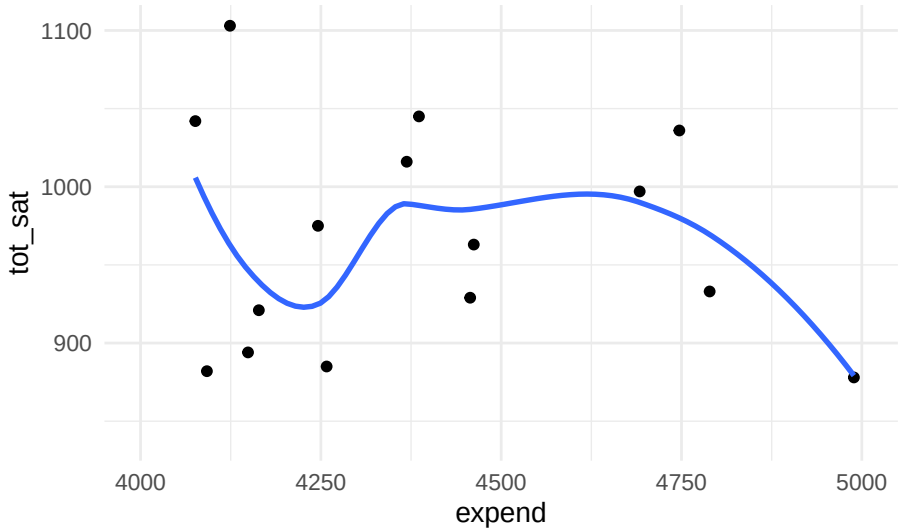
```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```




```
ggplot( sat93, aes( expend, tot_sat ) ) +
  geom_point() +
  geom_smooth( se=FALSE ) +
  scale_x_continuous( limits = c(4000, 5000 ) )
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
Warning: Removed 36 rows containing non-finite outside the scale range
(`stat_smooth()`).

Warning: Removed 36 rows containing missing values or values outside the scale range
(`geom_point()`).
```



9 Controlling size in markdown files

You can add `fig.height=2, fig.width=7` into the header of a chunk. For example, the chunk above had:

{r,r my_figure, fig.height=2, fig.width=7}

You can also set overall size in your setup chunk via

```
knitr::opts_chunk$set(echo = TRUE,
                      fig.width = 5,
                      fig.height = 3,
                      out.width = "75%",
                      fig.align = "center")
```

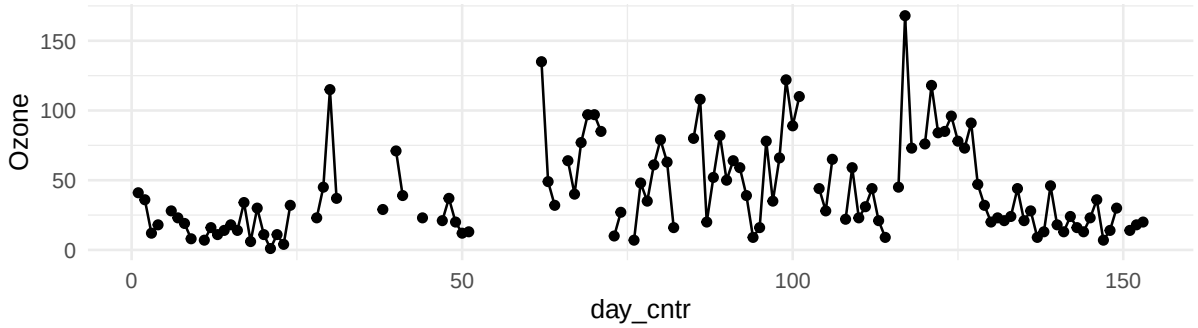
The `fig.width` controls how big the figure is when plotting. The figure is then rescaled to fit into the area dictated by `out.width`. The "75%" can be replaced with, e.g., "4in" for 4 inches. The percent is the percent of text width of the page.

10 Making the fonts of labels, etc., larger

If you make the figure smaller, the axes labels and line thicknesses and everything get larger, relatively speaking. Compare the following, with the header chunks printed out for clarity:

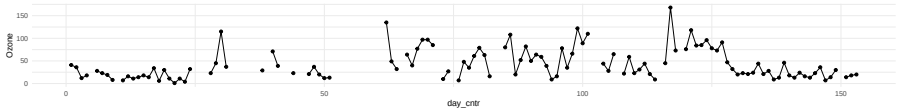
{r,r, fig.height=2, fig.width=7, out.width="100%", out.height="100%"}

```
ggplot( airquality, aes( day_cntr, Ozone ) ) +
    geom_point(na.rm=TRUE) + geom_line(na.rm=TRUE)
```



{r,r, fig.height=2, fig.width=16}

```
ggplot( airquality, aes( day_cntr, Ozone ) ) +
    geom_point(na.rm=TRUE) + geom_line()
```

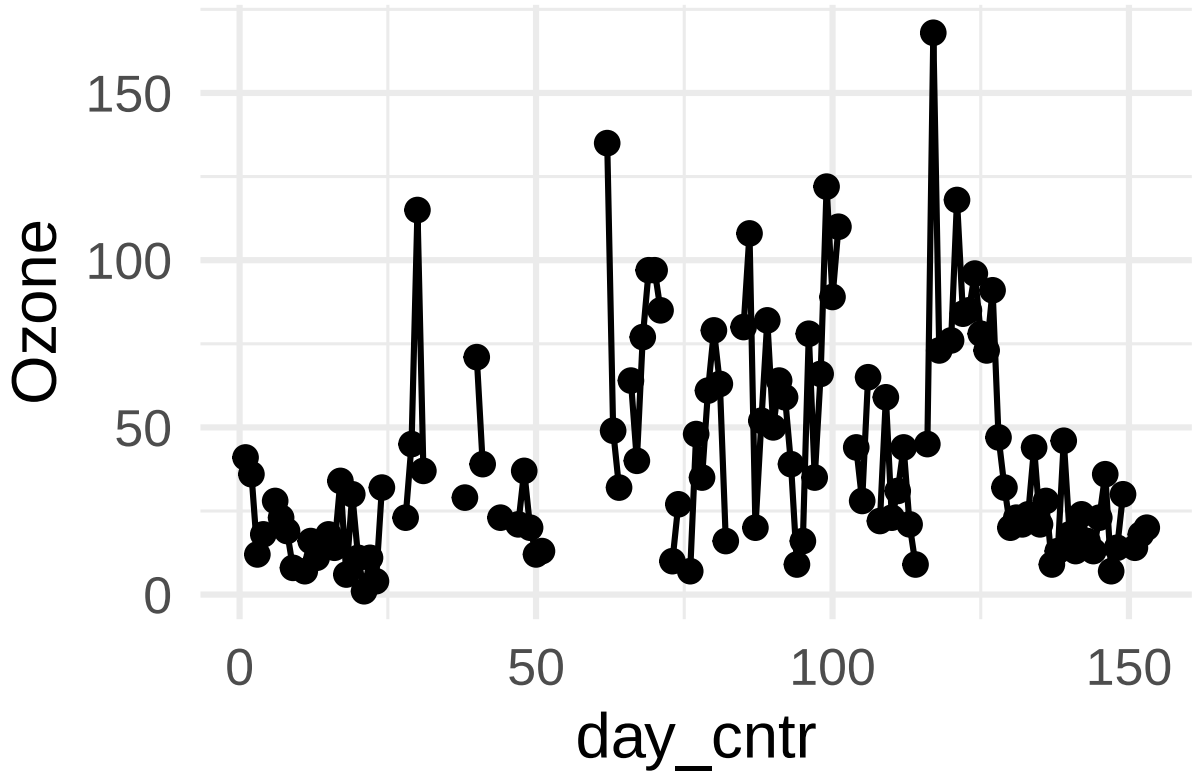


See how the second figure is a bigger figure rescaled to fit into a smaller space (width-wise). Everything thus looks tiny. The other plots in this document are `fig.height=2, fig.width=7`.

If the plot is small, but `out.width` is large, it will not scale up but instead keep to the desired size. The `out.width` only ensures plots are no larger than given. E.g. `fig.width=3, fig.height=2, out.width="100%"` gives:

{r,r, warning = FALSE, fig.width=3, fig.height=2, out.width="100%"}

```
ggplot( airquality, aes( day_cntr, Ozone ) ) +
    geom_point(na.rm=TRUE) + geom_line()
```



11 Saving high resolution plots

There are three ways to export plots from RStudio. You can click on the export button on RStudio. If you say "Copy to Clipboard" you will get a bitmap plot (basically a digital photo of your plot), which will be fuzzy if you blow it up in your report. One way around that is specify a larger width and height in the dialog box before you save. "Save as Image" has the same concern.

The "Save to PDF" will save a vector image of your plot. Vector images remember the dots and lines used for your plot, and will be crisp if you make them large or small.

You can also use `ggsave` as follows:

```
ggplot( airquality, aes( day_cntr, Ozone ) ) +
    geom_point(na.rm=TRUE) + geom_line()
ggsave( "my_ac_plot.pdf", width = 8, height=2 )
```

This allows you to specify the width and height and then, if you want to re-make your plot, it will save the exact same size. This is also good if you have a series of plots you want to make the same size.

I recommend saving to PDF. The image bitmap plots always look a bit lousy.

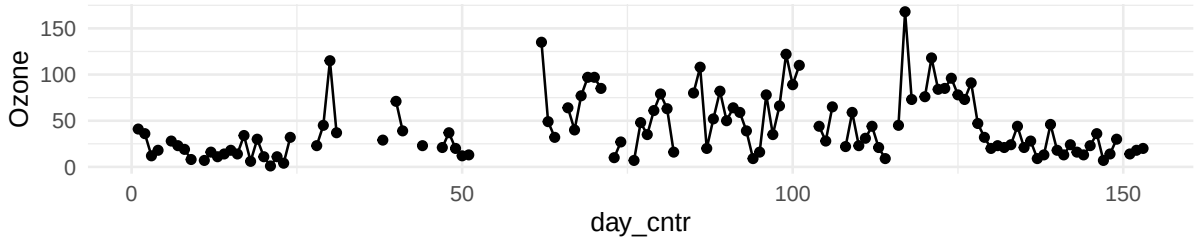
12 Think about the height of a plot.

Very often, making your plot *shorter* will make it better. Big vertical distance amplifies variation and makes it hard to follow trends.

Compare

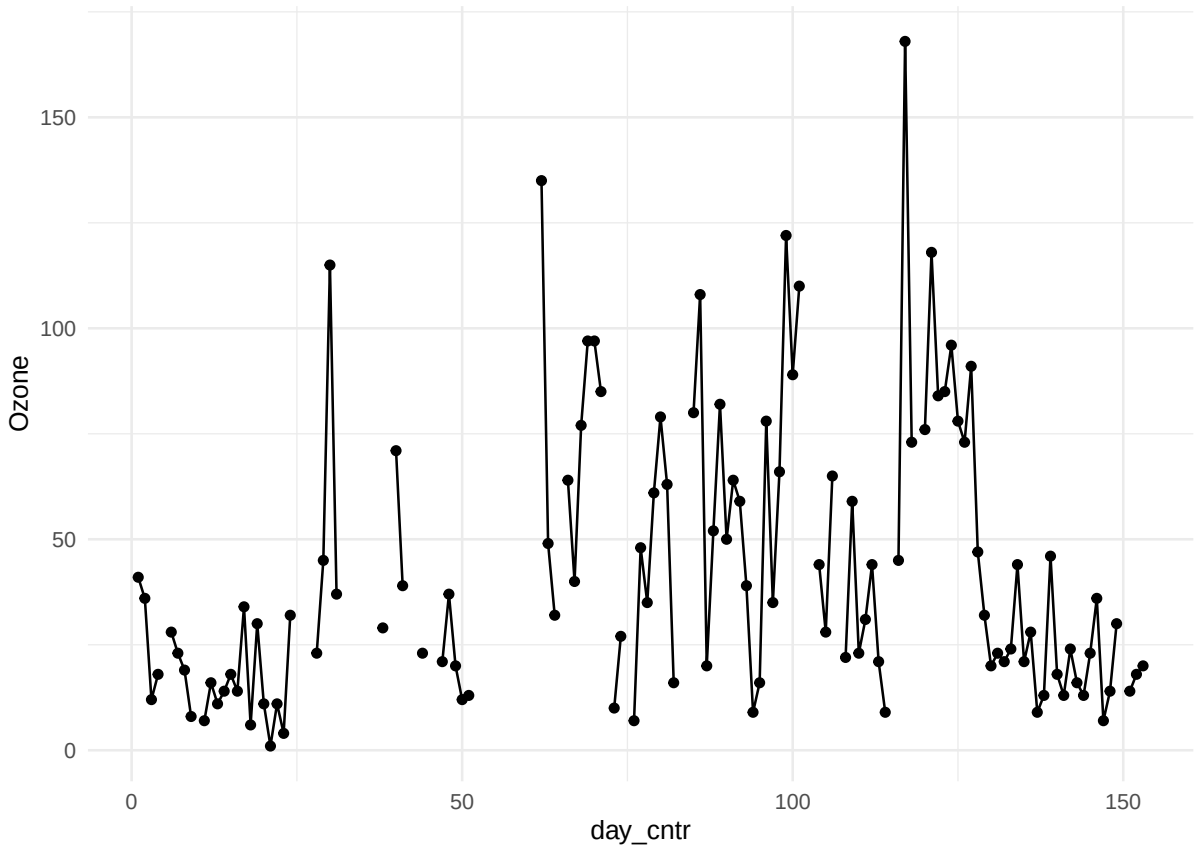
{r,r, fig.height=1.5, fig.width=7, out.width="100%", out.height="100%"}

```
ggplot( airquality, aes( day_cntr, Ozone ) ) +
    geom_point(na.rm=TRUE) + geom_line(na.rm=TRUE)
```



{r,r, fig.height=5, fig.width=7, out.width="100%", out.height="100%"}

```
ggplot( airquality, aes( day_cntr, Ozone ) ) +
    geom_point(na.rm=TRUE) + geom_line(na.rm=TRUE)
```

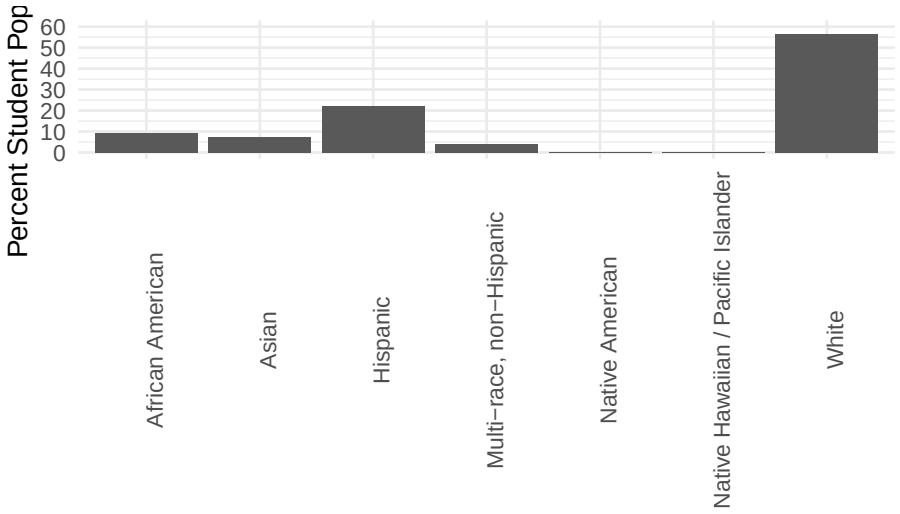

150
100
50
0
Ozone
0
50
100
150
day_cntr

13 Think about the x and y axis of your plot

Similarly, swapping the x and y axes of a plot can make labels *a lot* easier to read.

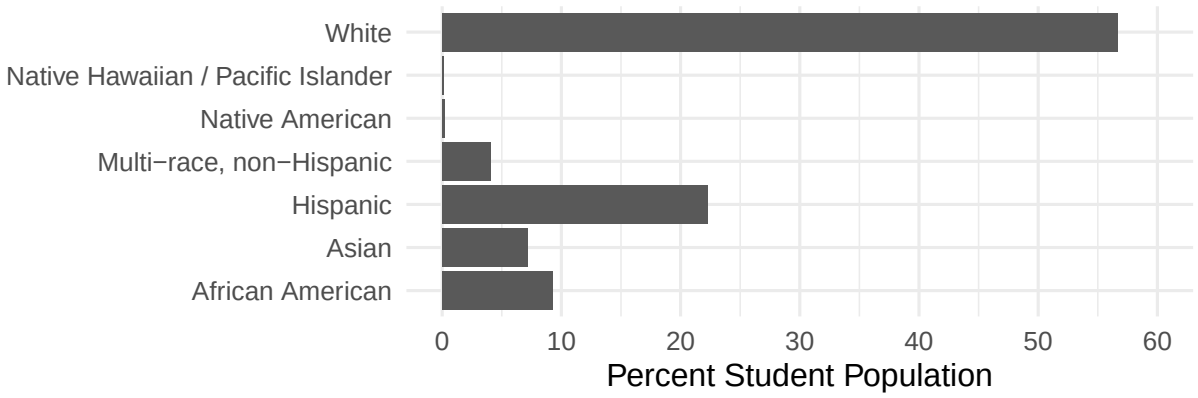
Consider the following two plots:

```
ggplot( dat, aes( x = Category, y=per_students ) ) +
    geom_bar(stat="identity") +
    theme(axis.text.x = element_text(angle = 90))  +
    scale_y_continuous( limits = c(0,60), breaks = c(0,10,20,30,40,50,60) ) +
    labs( y = "Percent Student Population", x="" )
```



{r,r, fig.height=2, fig.width = 6, out.width="100%"}

```
ggplot( dat, aes( x = Category, y=per_students ) ) +
    geom_bar(stat="identity") +
    theme(axis.text.x = element_text(angle = 90))  +
    labs( y = "Percent Student Population", x="" ) +
    scale_y_continuous( limits = c(0,60), breaks = c(0,10,20,30,40,50,60) ) +
    coord_flip() +
    theme(axis.text.x = element_text(angle = 0))
```



Flipping is easy to do. Just add `coord_flip()` to your `ggplot` command! The choice of vertical height of the plot also matters here. Make your bars not too thick, but still a nice amount thick so your labels are well spaced.

For the first plot, we had to rotate the labels so they didn't overlap. Other choices are likely possible.

Also notice the rotating of the axes labels. With `coord_flip` be careful as to what element you are controlling. Usually fiddling around will get it right in no time!

Note: these data are from the Massachusetts Department of Elementary and Secondary Education, school year 2019-20, for the state.

14 Controlling colors, etc

For this, read the "Graphics for Communication" chapter of R for DS: https://r4ds.had.co.nz/graphics-for-communication.html

15 Plotting aggregate data with ggPlot

This handout demonstrates how to create histograms with ggplot when the data have already been aggregated or tallied.

15.0.1 Preface

Before we get started, let's load the packages we'll need.

```
library(tidyverse)
library(ggplot2)
```

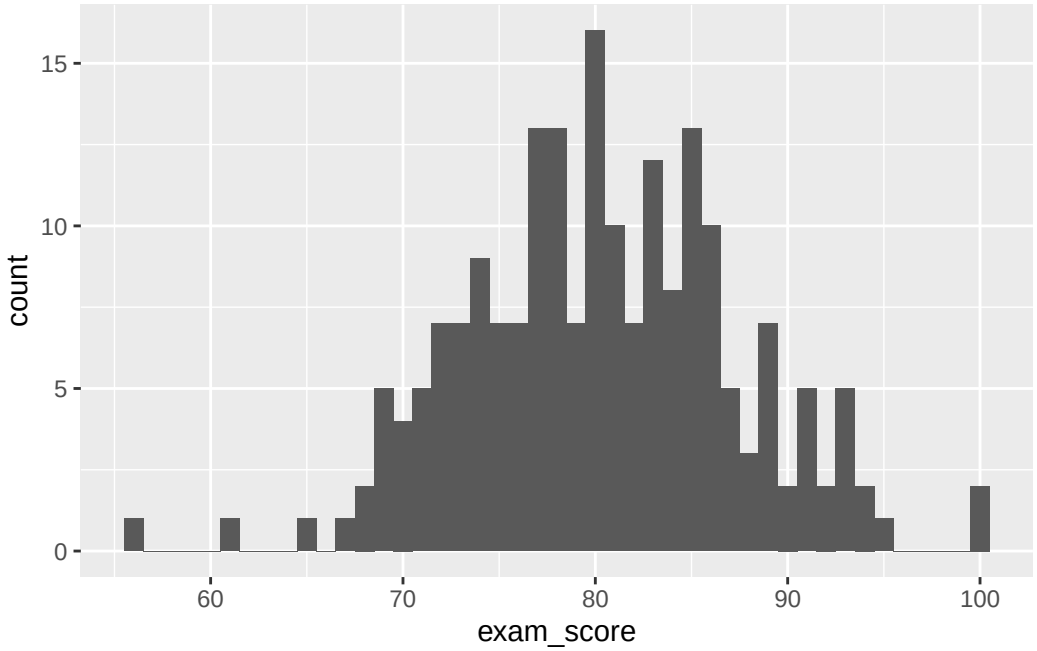
16 The Situation

Typically, we create histograms from individual data points. Suppose we have data on students. The data include a student ID number, a course identifier, the grade the student got in the course, and their final exam score (out of 100 points). Here's some example data.

```
num_students <- 200
df <- data.frame(sid=rep(1:num_students, 2),
                 course=c(rep('math', num_students), rep('ela', num_students)),
                 grade=sample(c('A', 'B', 'C', 'D', 'F'),
                              2*num_students,
                              replace=TRUE),
                 exam_score=sample(0:100, 2*num_students, replace=TRUE,
                                   prob=dnorm(0:100, mean=80, sd=7)))
```

Let's make a histogram of the exam score data for math. This should be familiar.

```
ggplot(df %>% filter(course=='math'), aes(x=exam_score)) +
  geom_histogram(binwidth=1)
```



There's a lot we could do to make this a publication-ready graphic - label the axes, add a title, etc. - but our focus here is on the core plotting, so we'll skip it.

But what if the data were already aggregated by exam score? Let me show you what I mean in code.

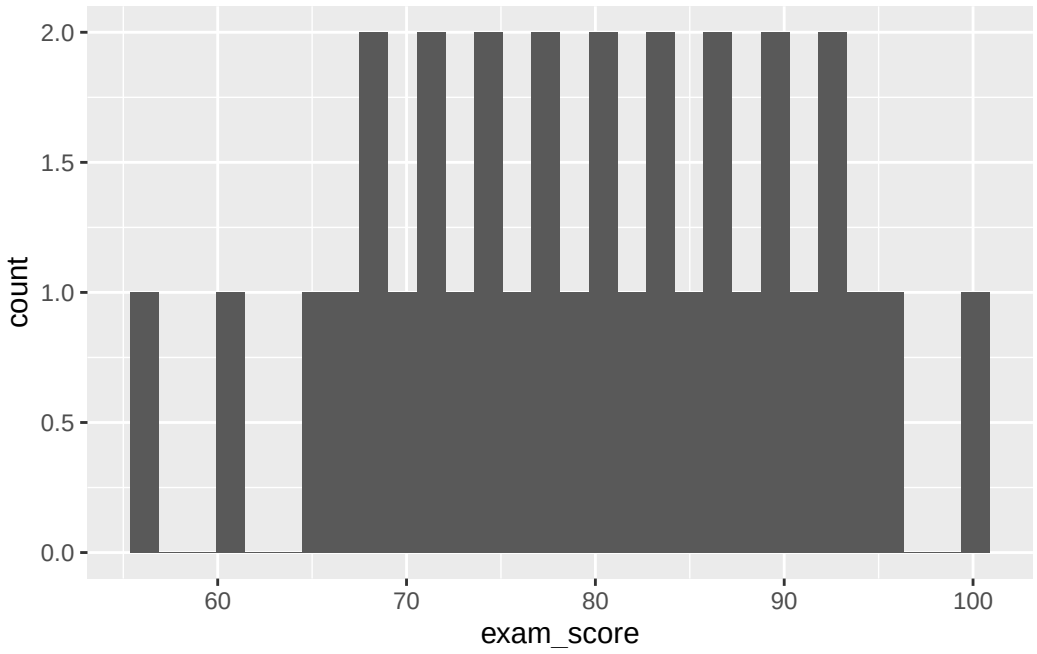
```
exam_agg <- df %>%
  filter(course=='math') %>%
  group_by(exam_score) %>%
  summarize(n=n())
exam_agg
```

```
# A tibble: 33 x 2
   exam_score     n
        <int> <int>
 1         56     1
 2         61     1
 3         65     1
 4         67     1
 5         68     2
 6         69     5
 7         70     4
 8         71     5
 9         72     7
10         73     7
# i 23 more rows
```

The dataset `exam_agg` contains only two variables: the exam score and the number of students who received that score. Sometimes, this is the way we receive data. How can we make a histogram out of this? Let's try it the old way:

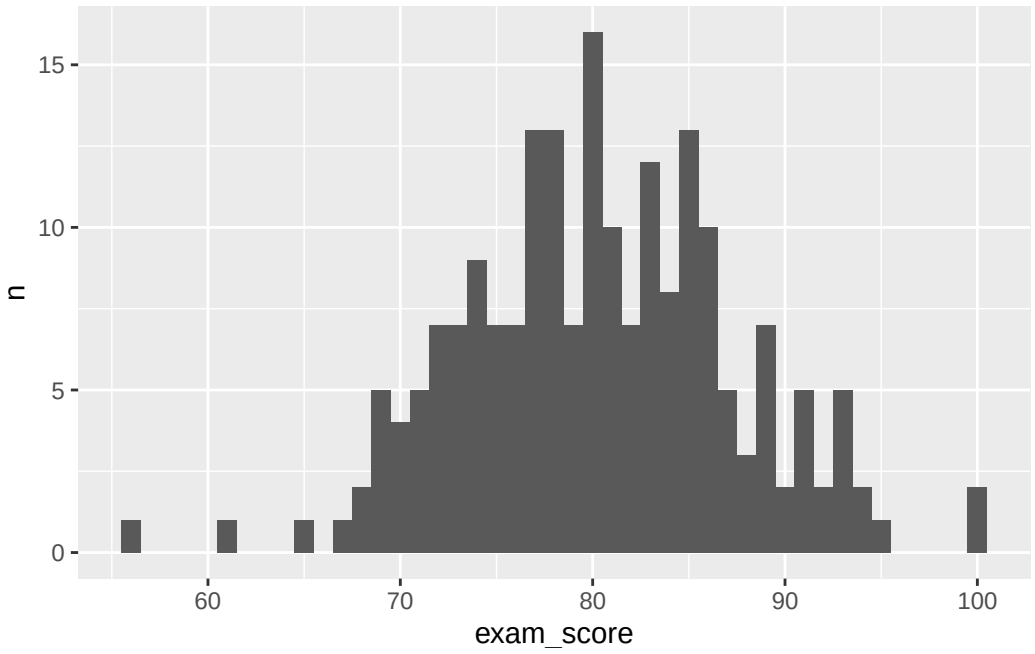
```
ggplot(exam_agg, aes(x=exam_score)) +
  geom_histogram()
```

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



Whoa! That doesn't look right. The problem is that `geom_histogram()` counts how many times each score appears in the data, but after we aggregated, each score only appears once. We need to tell R that the counts are stored in the `n` column of the dataset. Here's how we do it:

```
ggplot(exam_agg, aes(x=exam_score, y=n)) +
  geom_bar(stat='identity', width=1)
```

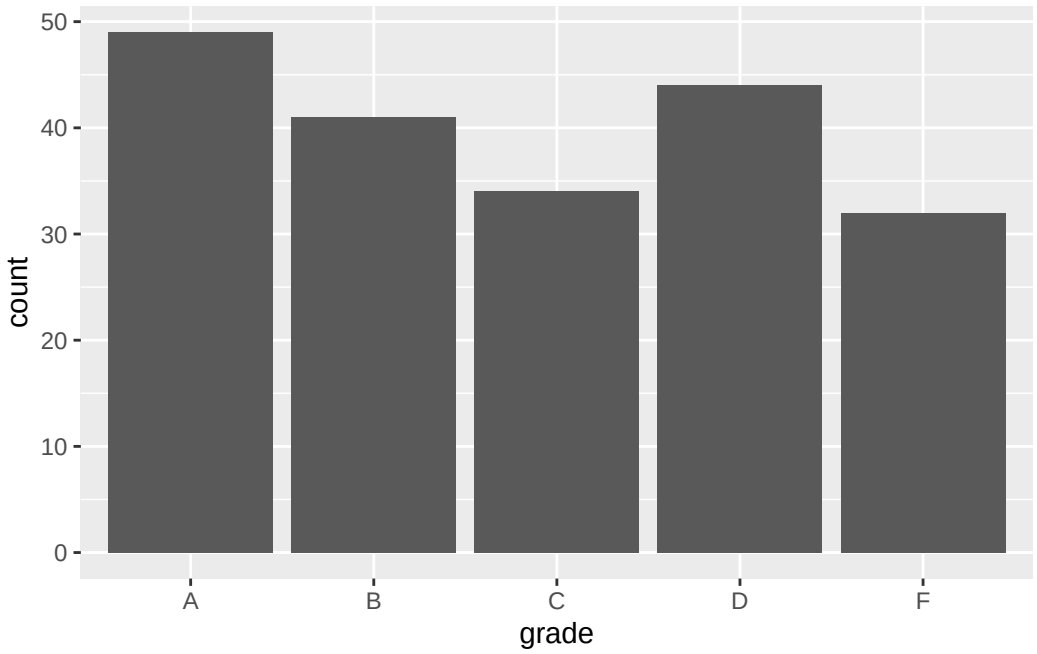


We've switched from `geom_histogram()` to `geom_bar()`. This allows us to specify a separate variable for the y-axis. Inside the `geom_bar()` call, we need to specify that `stat='identity'`. This tells ggplot that it should just use the raw value of the y-variable we gave it. If we don't make this explicit, R doesn't know what to do with the second variable.

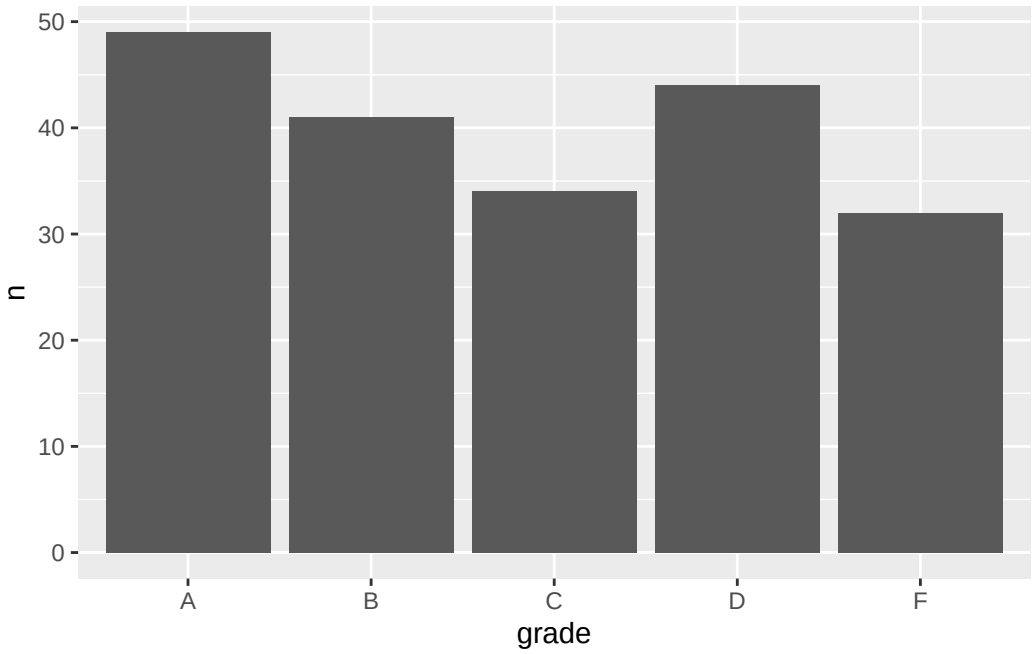
16.1 Categorical data and bar charts

We can do the same thing with a categorical variable, course grades. First we'll plot using the raw data, and then we'll switch to the aggregated version. For the aggregated case, we'll again rely on the `stat` argument of the `geom_bar()` function.

```
# Plot from raw data
# For a categorical variable, we use geom_bar
df %>% filter(course=='ela') %>%
  ggplot(aes(x=grade)) +
  geom_bar()
```




```
# Now let's aggregate the data
grade_agg <- df %>% filter(course=='ela') %>%
  group_by(grade) %>%
  summarize(n=n())

# Now reproduce the bar plot from the aggregated data
ggplot(grade_agg, aes(x=grade, y=n)) + geom_bar(stat='identity')
```



Looks exactly the same! (Except for the y-axis label.)

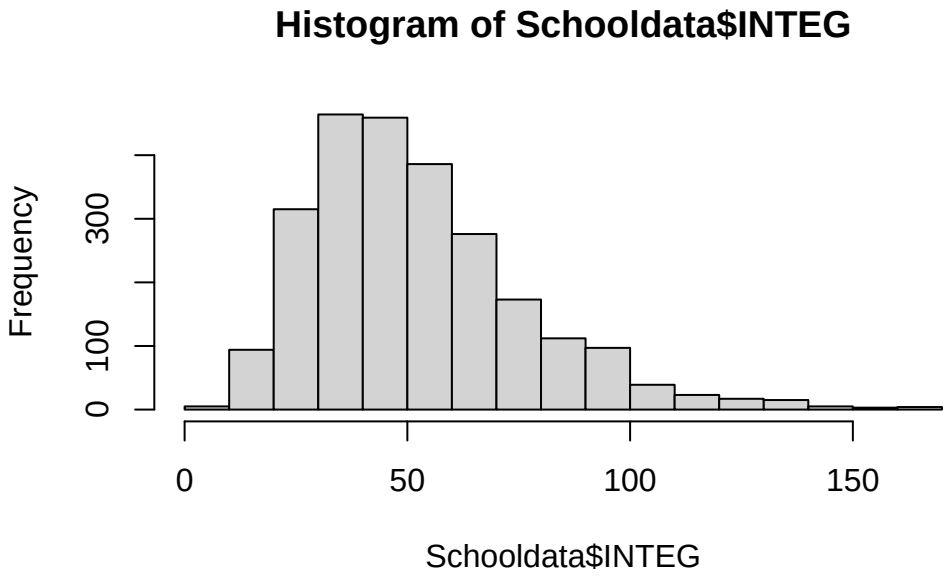
16.2 A Closing Thought

Sometimes the data we get have already been aggregated. In this case, we might still want to visualize distributions of the variables that have been grouped. The above guide introduces the `stat` argument of `geom_bar()` as a way to get this done.

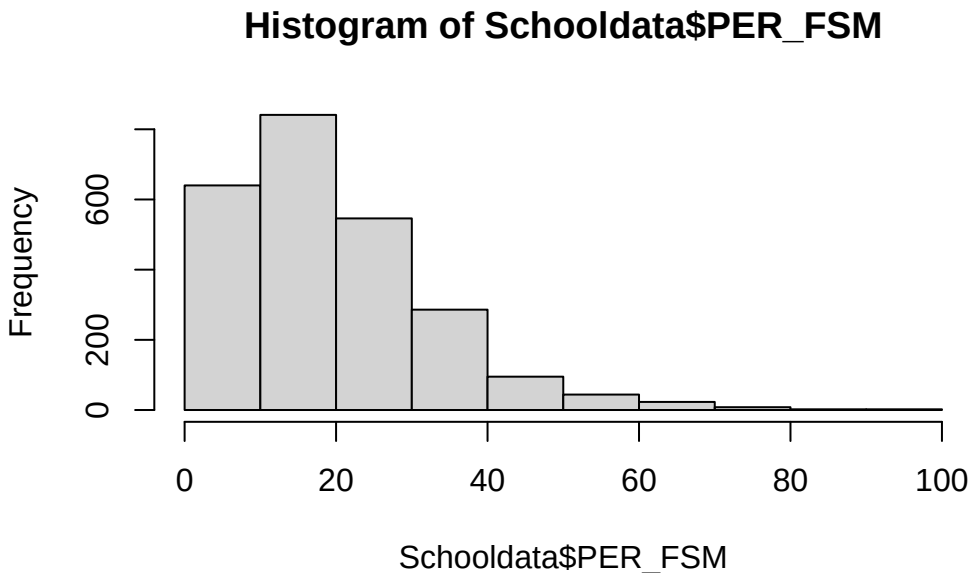
17 Jitter and heatmap examples

18 Jitter examples

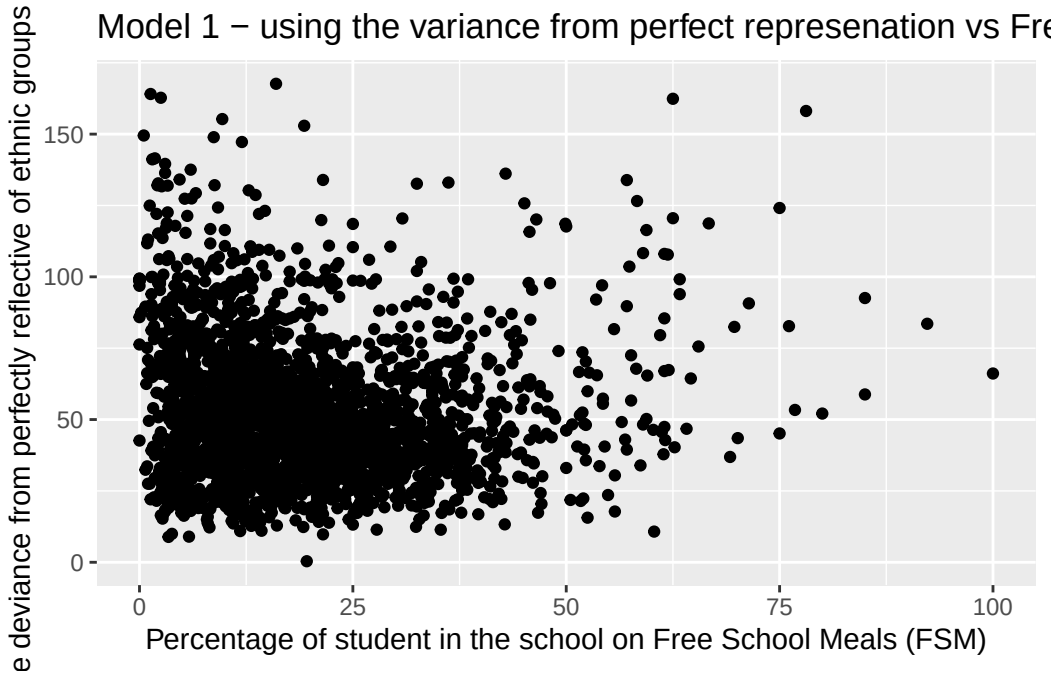
```
# Histogram of ouput variable (INTEG)
hist(Schooldata$INTEG)
```




```
# Histogram of predictor variable (PER_FSM)
hist(Schooldata$PER_FSM)
```




```
# Scatterplot of ouput variable (INTEG) on predictor (FSM)
ggplot(Schooldata, aes(x = PER_FSM, y = INTEG)) +
  geom_point() +
  labs(title= "Model 1 - using the variance from perfect represenation vs Free School Meals"
       x = "Percentage of student in the school on Free School Meals (FSM)",
       y = "Percentage deviance from perfectly reflective of ethnic groups in local area")
```

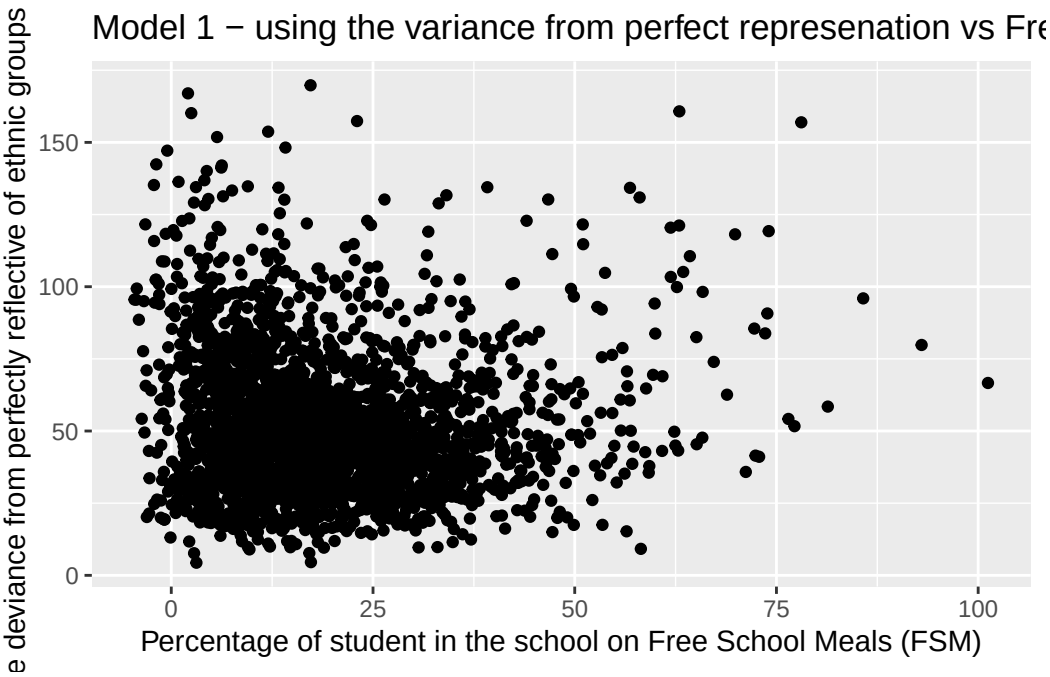


```r
fit1 <- lm(INTEG ~ PER_FSM, data=Schooldata)
summary(fit1)
```

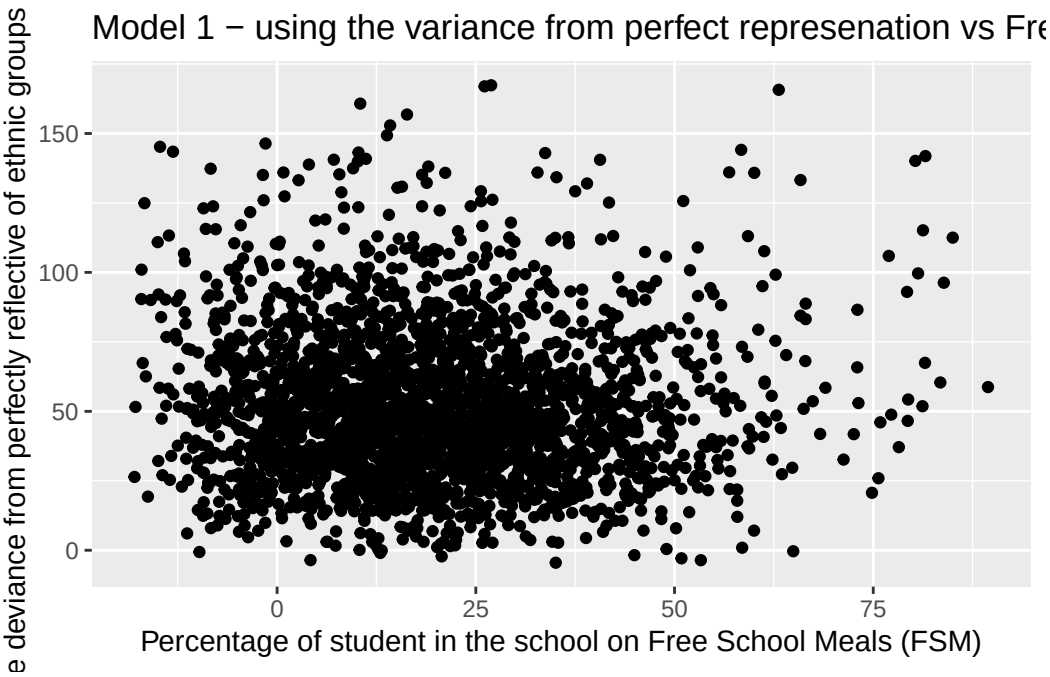
```
Call:
lm(formula = INTEG ~ PER_FSM, data = Schooldata)

Residuals:
    Min      1Q  Median      3Q     Max
-51.901 -17.485  -3.918  12.352 115.029

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 54.10637    0.86768  62.358   <2e-16 ***
PER_FSM     -0.09395    0.03680  -2.553   0.0107 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 24.4 on 2485 degrees of freedom
Multiple R-squared:  0.002616,  Adjusted R-squared:  0.002215
F-statistic: 6.518 on 1 and 2485 DF,  p-value: 0.01074
```

We can try jittering more (jittering here is not a good choice, by the way). The width and height control how much jitter to do both left-right and up-down.

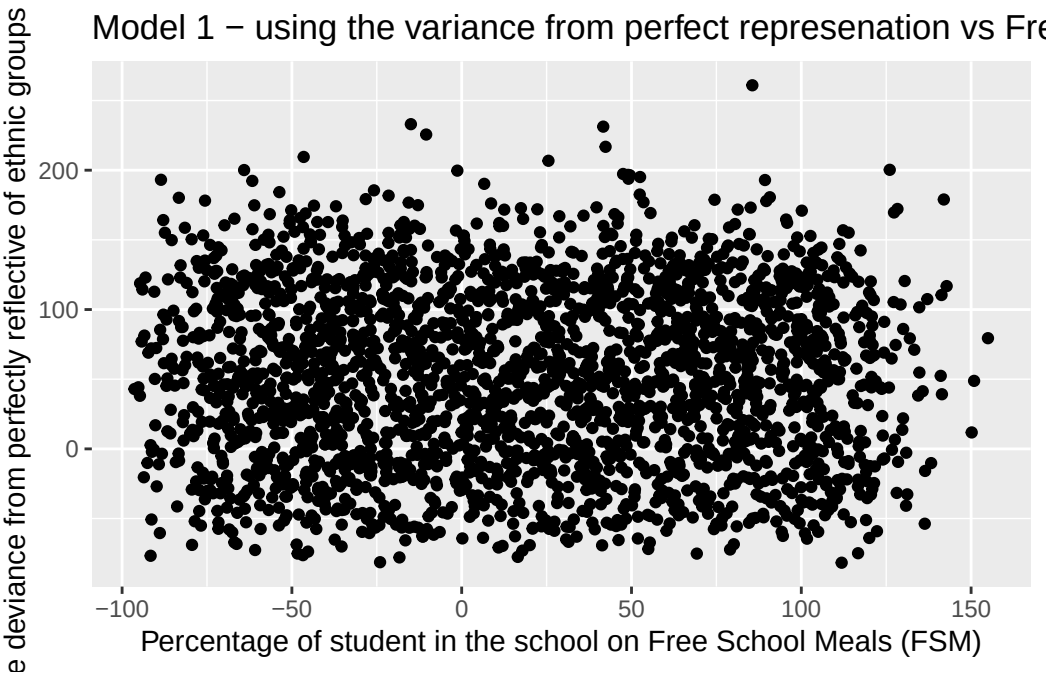
```r
ggplot(Schooldata, aes(x = PER_FSM, y = INTEG)) +
  geom_jitter(width = 5, height=5) +
  labs(title= "Model 1 - using the variance from perfect represenation vs Free School Meals"
       x = "Percentage of student in the school on Free School Meals (FSM)",
       y = "Percentage deviance from perfectly reflective of ethnic groups in local area")
```




```
ggplot(Schooldata, aes(x = PER_FSM, y = INTEG)) +
  geom_jitter(width = 20, height=20) +
  labs(title= "Model 1 - using the variance from perfect represenation vs Free School Meals"
       x = "Percentage of student in the school on Free School Meals (FSM)",
       y = "Percentage deviance from perfectly reflective of ethnic groups in local area")
```




```
ggplot(Schooldata, aes(x = PER_FSM, y = INTEG)) +
  geom_jitter(width = 100, height=100) +
  labs(title= "Model 1 - using the variance from perfect represenation vs Free School Meals"
       x = "Percentage of student in the school on Free School Meals (FSM)",
       y = "Percentage deviance from perfectly reflective of ethnic groups in local area")
```



You can also generate jittered data with `jitter()`:

```
a = 1:10
a
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
jitter( a )
```

```
 [1]  1.053524  2.023156  2.937460  4.139914  4.800838  5.896999  6.956115
 [8]  7.996693  9.092333 10.015065
```

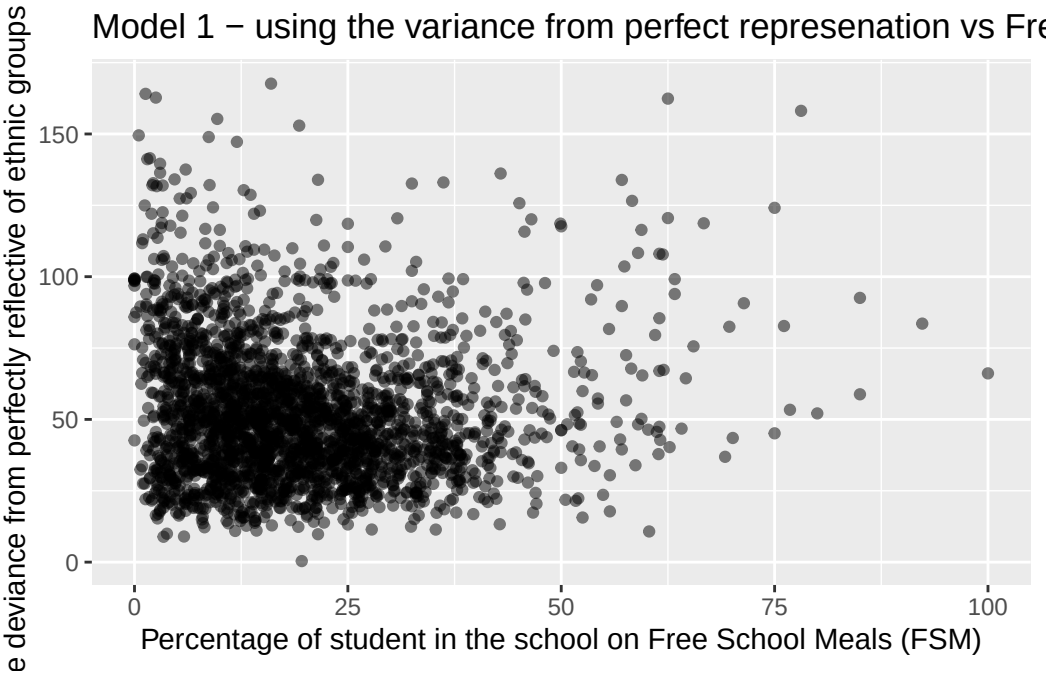
```
jitter( a, factor = 0.5 )
```

```
 [1] 1.090823 1.923515 2.960854 3.942592 5.029437 6.004726 6.902319 7.953197
 [9] 9.022219 9.952510
```

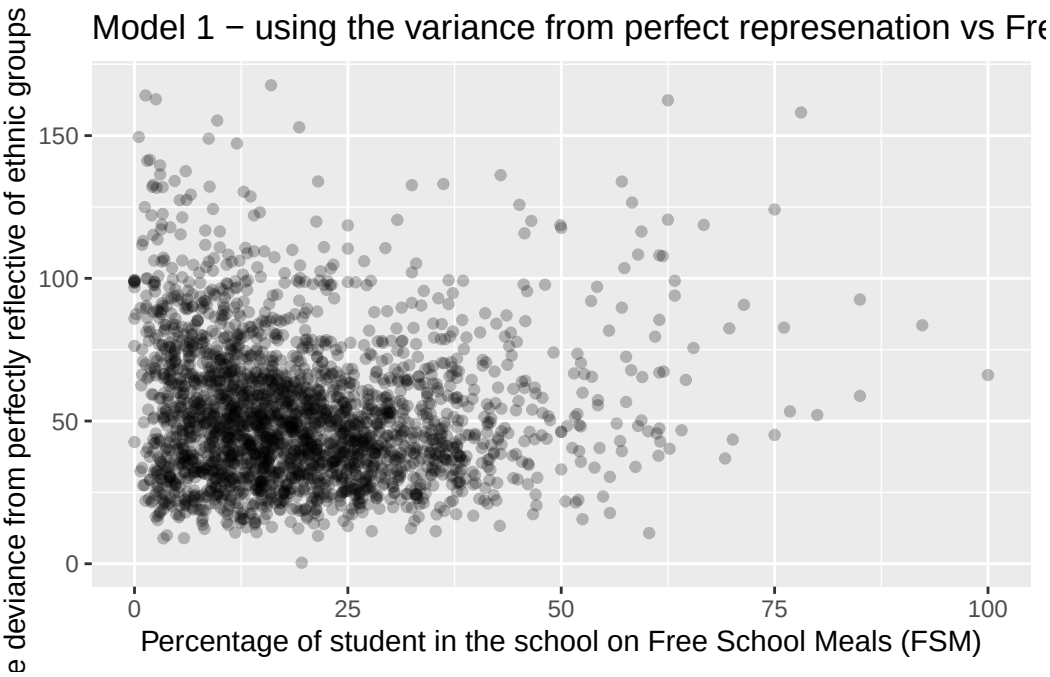
```
jitter( a, factor = 0.0005 )
```

```
 [1]  0.9999284  2.0000583  3.0000589  4.0000290  5.0000213  6.0000827
 [7]  6.9999009  7.9999842  8.9999875 10.0000287
```

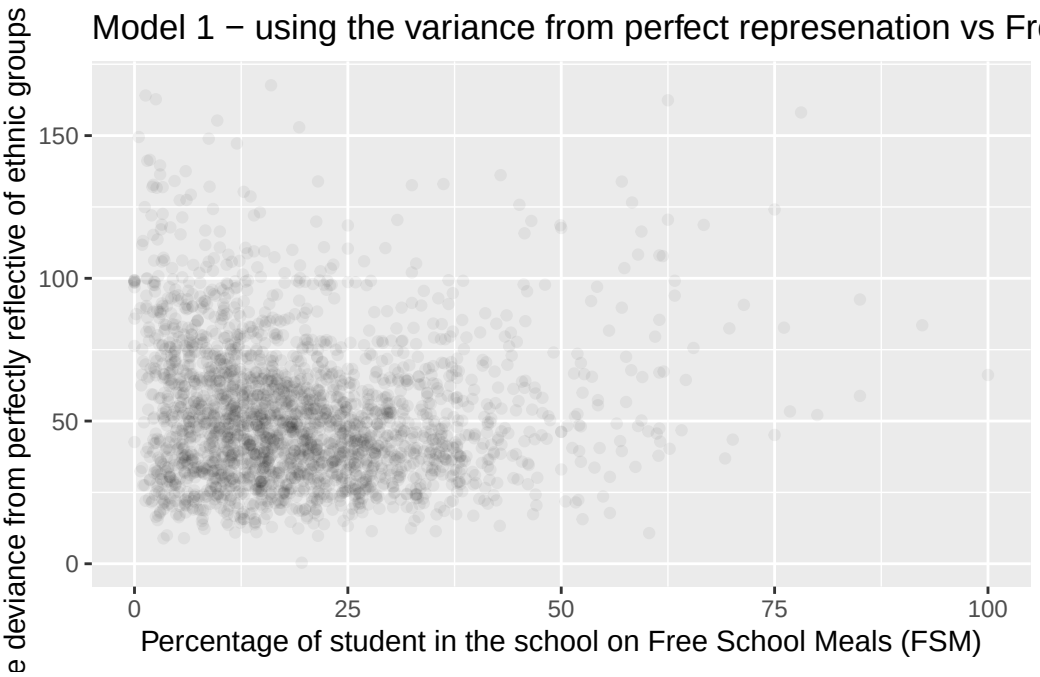
```
## Try alpha

ggplot(Schooldata, aes(x = PER_FSM, y = INTEG)) +
  geom_point(alpha = 0.5) +
  labs(title= "Model 1 - using the variance from perfect represenation vs Free School Meals"
       x = "Percentage of student in the school on Free School Meals (FSM)",
       y = "Percentage deviance from perfectly reflective of ethnic groups in local area")
```




```
ggplot(Schooldata, aes(x = PER_FSM, y = INTEG)) +
  geom_point(alpha = 0.25) +
  labs(title= "Model 1 - using the variance from perfect represenation vs Free School Meals"
       x = "Percentage of student in the school on Free School Meals (FSM)",
       y = "Percentage deviance from perfectly reflective of ethnic groups in local area")
```



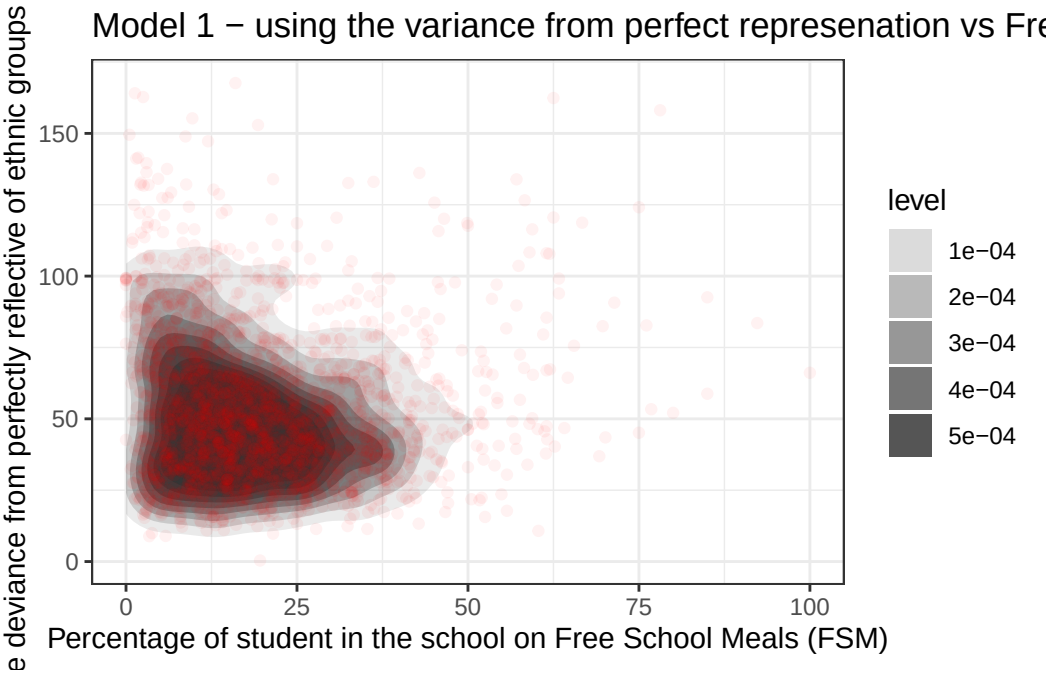

```
ggplot(Schooldata, aes(x = PER_FSM, y = INTEG)) +
  geom_point(alpha = 0.05) +
  labs(title= "Model 1 - using the variance from perfect represenation vs Free School Meals"
       x = "Percentage of student in the school on Free School Meals (FSM)",
       y = "Percentage deviance from perfectly reflective of ethnic groups in local area")
```


Model 1 − using the variance from perfect represenation vs Fre
e deviance from perfectly reflective of ethnic groups
150
100
50
0
0
25
50
75
100
Percentage of student in the school on Free School Meals (FSM)

19 Making a heat map

```
ggplot(Schooldata,aes(x = PER_FSM, y = INTEG))+
  stat_density2d(aes(alpha=..level..), geom="polygon") +
  labs(title= "Model 1 - using the variance from perfect represenation vs Free School Meals"
       x = "Percentage of student in the school on Free School Meals (FSM)",
       y = "Percentage deviance from perfectly reflective of ethnic groups in local area") +
  geom_point(colour="red",alpha=0.05)+
  theme_bw()
```

```
Warning: The dot-dot notation (`..level..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(level)` instead.
```



Part III

On Coding

20 Basic Data Manipulation (tidyverse)

21 R for DS Chapter 5 Core Commands

Let's use this simple table and run through the commands from Chapter 5. This is not a complete reference! See the text for further details.

```
library( tidyverse )
table1
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

21.1 `filter()` (Grab the rows you want, 5.1)

```
filter( table1, year > 1999 )
```

```
# A tibble: 3 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
1 Afghanistan  2000   2666   20595360
2 Brazil       2000  80488  174504898
3 China        2000 213766 1280428583
```

Remember, if you want to save the results of your command, you need to put it in a new variable, like so:

```
my.table <- filter( table1, year > 1999 )
my.table
```

```
# A tibble: 3 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
1 Afghanistan  2000   2666   20595360
2 Brazil       2000  80488  174504898
3 China        2000 213766 1280428583
```

When you do something like this, you should see it appear in your workplace.

21.2 `arrange()` (Sort your rows the way you want, 5.2)

```
arrange( table1, cases )
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

```
arrange( table1, desc( country ), population )
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
1 China        1999 212258 1272915272
2 China        2000 213766 1280428583
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 Afghanistan  1999    745   19987071
6 Afghanistan  2000   2666   20595360
```

21.3 `select()` (Grab the columns you want, 5.3)

```
select( table1, country, population )
```

```
# A tibble: 6 x 2
  country     population
  <chr>            <dbl>
1 Afghanistan   19987071
2 Afghanistan   20595360
3 Brazil       172006362
4 Brazil       174504898
5 China       1272915272
6 China       1280428583
```

21.4 `mutate()` (Make new variables out of your old ones, 5.4)

```
table1 <- mutate( table1, case.per.1000 = 1000 * cases / population )
table1
```

```
# A tibble: 6 x 5
  country      year  cases population case.per.1000
  <chr>       <dbl>  <dbl>      <dbl>         <dbl>
1 Afghanistan  1999    745   19987071        0.0373
2 Afghanistan  2000   2666   20595360        0.129
3 Brazil       1999  37737  172006362        0.219
4 Brazil       2000  80488  174504898        0.461
5 China        1999 212258 1272915272        0.167
6 China        2000 213766 1280428583        0.167
```

(We will use this new variable later, so I am saving it in our table)

21.5 `group_by()` and `summarize` (Summarize your data by subgroup, 5.6)

```
tbl <- group_by( table1, country )
summarize( tbl, av.pop = mean( population ), av.cases = mean( cases ) )
```

```
# A tibble: 3 x 3
  country          av.pop av.cases
  <chr>             <dbl>    <dbl>
1 Afghanistan   20291216.    1706.
2 Brazil       173255630    59112.
3 China       1276671928.  213012
```

Same thing, with the pipe!

```
table1 %>% group_by( country ) %>%
    summarize( av.pop = mean( population ), av.cases = mean( cases ) )
```

```
# A tibble: 3 x 3
  country          av.pop av.cases
  <chr>             <dbl>    <dbl>
1 Afghanistan   20291216.    1706.
2 Brazil       173255630    59112.
3 China       1276671928.  213012
```

21.6 Special: grouped mutates (Making new variables within subgroups, 5.6)

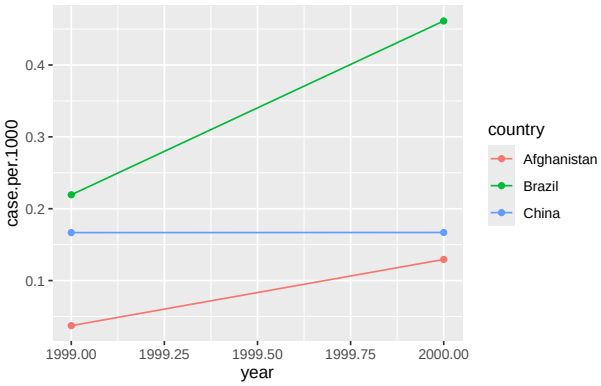
This combo is for doing things like group mean centering your data:

```
table1 %>% group_by( year ) %>% mutate( case.per.1000.cent = case.per.1000 - mean( case.per.
```

```
# A tibble: 6 x 6
# Groups:   year [2]
  country      year  cases population case.per.1000 case.per.1000.cent
  <chr>       <dbl>  <dbl>      <dbl>         <dbl>              <dbl>
1 Afghanistan  1999    745   19987071        0.0373            -0.104
2 Afghanistan  2000   2666   20595360        0.129             -0.123
3 Brazil       1999  37737  172006362        0.219              0.0783
4 Brazil       2000  80488  174504898        0.461              0.209
5 China        1999 212258 1272915272        0.167              0.0256
6 China        2000 213766 1280428583        0.167             -0.0856
```

And a plot

```
ggplot( table1, aes( x=year, y=case.per.1000, col=country ) ) + geom_line() + geom_point()
```


0.4
0.3
0.2
0.1
case.per.1000
1999.00
1999.25
1999.50
1999.75
2000.00
year
country
Afghanistan
Brazil
China

22 Doing things over and over again

In this handout, we will look at several ways of doing things over and over again in R. This comes up all the time. Three main ones we have seen are the following:

- Fit a loess curve to my data for each of a series of smoothing parameters.
- Repeatidly bootstrap my dataset and analyze it.
- Run a simulation where we generate data and analyze it over and over.
- Scrape a series of web pages

Coding wise, there are a few ways of repeating yourself. We are going to walk through them and compare these tools to each other.

23 The `replicate()` command

This simple command repeats a line of code a given number of times. If the line of code gives you a number back each time it is run, then you will end up with a list of numbers.

To illustrate, say we want to look at rolling dice. Here is some code that provides a function that will roll some number of 6-sided dice:

```
roll_dice = function( ndice ) {
    rolls = sample( 1:6, ndice, replace=TRUE )
    sum( rolls )
}

# Roll a single die
roll_dice( 1 )
```

```
[1] 4
```

```
# Roll 3 dice and add them up.
roll_dice( 3 )
```

```
[1] 9
```

If we want to get the sum of three dice over and over, we can replicate:

```
rolls = replicate( 10, roll_dice( 3 ) )
rolls
```

```
 [1] 12 11  9  7 14 10  9 13  8  6
```

Note how the `rolls` variable is a nice numeric vector, easy to work with. It is easy to do calculations with it, like take the average:

```
mean( rolls )
```

```
[1] 9.9
```

Here we use this to see how often we roll above a 15:

```
rolls = replicate( 10000, roll_dice( 3 ) )
mean( rolls > 15 )
```

```
[1] 0.0467
```

24 The `rerun()` command

The `rerun()` command, from the tidyverse, is almost exactly like replicate, but instead of giving a numeric vector of numbers back, it gives an R list. For brevity, I rerun twice in the following:

```
rolls = rerun( 2, roll_dice(3) )
```

```
Warning: `rerun()` was deprecated in purrr 1.0.0.
i Please use `map()` instead.
  # Previously
  rerun(2, roll_dice(3))

  # Now
  map(1:2, ~ roll_dice(3))
```

```
rolls
```

```
[[1]]
[1] 11

[[2]]
[1] 8
```

Each element of the list is itself a list of numbers! Why would someone make such an annoying command like that? This is useful if the function we are rerunning gives us back multiple things. For example:

```
roll_dice_extended = function( ndice ) {
    rolls = sample( 1:6, ndice, replace=TRUE )
    c( mean = mean( rolls ), median = median( rolls ), max = max( rolls ) )
}
roll_dice_extended()
```

```
    mean   median      max
2.833333 3.000000 5.000000
```

Now when we rerun we get this:

```
rolls = rerun( 2, roll_dice_extended(3) )
```

```
Warning: `rerun()` was deprecated in purrr 1.0.0.
i Please use `map()` instead.
  # Previously
  rerun(2, roll_dice_extended(3))

  # Now
  map(1:2, ~ roll_dice_extended(3))
```

```
rolls
```

```
[[1]]
  mean median    max
     4      4      5

[[2]]
    mean   median      max
4.666667 5.000000 5.000000
```

But this is hard to work with. All our numbers are nested and weird. But there is a solution, which is the "rerun + bind_rows combo". Once we have a list of our answers, we can "stack" them with `bind_rows`:

```
bind_rows( rolls )
```

```
# A tibble: 2 x 3
   mean median   max
  <dbl>  <dbl> <dbl>
1  4         4     5
2  4.67      5     5
```

We can also give each row a name:

```
bind_rows( rolls, .id="runID" )
```

```
# A tibble: 2 x 4
  runID  mean median   max
  <chr> <dbl>  <dbl> <dbl>
1 1      4         4     5
2 2      4.67      5     5
```

Side Note: I recommend making your function give back a simple little dataframe of all your stuff. It is less prone to having weird errors. The `bind_rows()` method does really well with data.frames or tibbles. Here is an updated version of the above:

```
roll_dice_extended = function( ndice ) {
    rolls = sample( 1:6, ndice, replace=TRUE )
    data.frame( mean = mean( rolls ), median = median( rolls ), max = max( rolls ) )
}
roll_dice_extended()
```

```
  mean median max
1  3.5    3.5   5
```

```
rolls = rerun( 6, roll_dice_extended(3) )
```

```
Warning: `rerun()` was deprecated in purrr 1.0.0.
i Please use `map()` instead.
  # Previously
  rerun(6, roll_dice_extended(3))

  # Now
  map(1:6, ~ roll_dice_extended(3))
```

```
rolls = bind_rows( rolls, .id="runID" )

rolls
```

```
  runID     mean median max
1     1 1.666667      2   2
2     2 5.000000      5   6
3     3 3.666667      4   5
4     4 3.000000      3   5
5     5 3.333333      2   6
6     6 5.333333      6   6
```

24.1 Warning: `replicate()` doesn't do well with fancy functions

The `replicate()` command doesn't act nice in the following:

```
rolls = replicate( 2, roll_dice_extended(3) )
rolls
```

```
, , 1

  mean median max
1 3    2      6

, , 2

  mean     median max
1 4.666667 5      6
```

That doesn't look like a fun thing to work with. (It is a 3 dimensional array of output, in case you are wondering.) Use `rerun + bind_rows`; it is easier to control and understand.

24.2 Take-away

If your function that you want to repeat returns a single number with each call, use `replicate()`. If it returns more than one thing, use the `rerun() + bind_rows()` combination.

25 `map()`, another way of repeating yourself

The above runs the exact same code over and over. Sometimes you want to run the same function on a collection of different things (e.g., fit a loess line for each of a series of bandwidths). This is done with the `map()` command which takes a list of things, and then calls a function for each thing on that list.

To illustrate, we will see how the largest number rolled changes as a function of the number of dice rolled. To start, let's roll 1 die, then 2 dice, then 3 dice, and each time calculate the average, median, and max:

```
dice = 1:3
result = map( dice, roll_dice_extended )
result
```

```
[[1]]
  mean median max
1    5      5   5

[[2]]
  mean median max
1  3.5    3.5   5

[[3]]
  mean median max
1    2      1   4
```

Note unlike `rerun` or `replicate` we are not **calling** our `roll_dice_extended` function, we are just giving the name of it. You can tell since we do not have the `()` after `roll_dice_extended`, we just pass the name of the function we want to call. We are asking `map` to call `roll_dice_extended` over and over. For `rerun` or `replicate`, by comparison, we provide a stand-alone complete line of code that would run by itself. For `map` we just give the name of a function to run.

We can make our output nicer with the same `bind_rows` trick from `rerun()`:

```
bind_rows( result )
```

```
  mean median max
1  5.0    5.0   5
2  3.5    3.5   5
3  2.0    1.0   4
```

Even better is using the `map_df()` method, which works nicely **provided your function returns a dataframe**:

```
result = map_df( dice, roll_dice_extended )
result$ndice = dice
result
```

```
      mean median max ndice
1 1.000000    1.0   1     1
2 1.500000    1.5   2     2
3 3.333333    3.0   6     3
```

26 Repeating yourself when you are repeating yourself

Our little simulation is a bit sad in that we only have a single trial for each of our number of dice scenarios. What we really want is to know the distribution of the maximum roll for each number of dice. We do this by repeating ourself and then repeating this repeating ourself!

Study this:

```
one_trial = function( ndice ) {
    rolls = rerun( 1000, roll_dice_extended( ndice ) )
    rolls = bind_rows( rolls )
    tbl = table( rolls$max )
    tbl
}
one_trial( 1 )
```

```
Warning: `rerun()` was deprecated in purrr 1.0.0.
i Please use `map()` instead.
  # Previously
  rerun(1000, roll_dice_extended(ndice))

  # Now
  map(1:1000, ~ roll_dice_extended(ndice))
```

```
  1   2   3   4   5   6
166 174 160 163 182 155
```

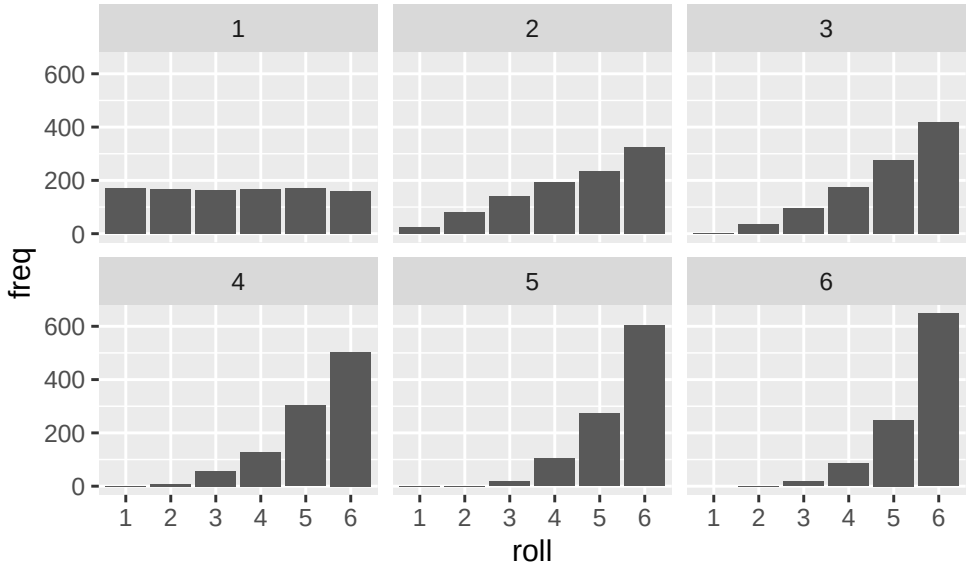
Then this:

```
result = map( 1:6, one_trial )
result = bind_rows( result )
result$ndice = 1:6
result
```

```
# A tibble: 6 x 7
  `1`          `2`          `3`          `4`          `5`          `6`          ndice
  <table[1d]> <table[1d]> <table[1d]> <table[1d]> <table[1d]> <table[1d]> <int>
1 170          167          165          167          170          161              1
2  26           81          142          193          235          323              2
3   2           34           94          174          277          419              3
4   1            8           57          128          302          504              4
5   1            1           18          105          273          602              5
6  NA            1           17           85          248          649              6
```

We can then make a nice plot:

```
resL = pivot_longer( result, `1`:`6`,
                     names_to="roll", values_to="freq" ) %>%
    filter( !is.na( freq ) )
resL$freq = as.numeric( resL$freq )
ggplot( resL, aes( roll, freq ) ) +
    facet_wrap( ~ndice ) +
    geom_col( )
```



Note. The `as.numeric` line is because the table command makes table objects which are really just lists of numbers, but it seems cleaner to tell R to knock it off and just let it be a number. Not doing this still works fine, it just gives a warning in `ggplot()`

27 Advanced stuff

In the following we look at a few cool functions that help with mapping.

27.1 `relocate()`

Not really mapping specific, but still a nice way to move a variable to the start of a data frame.

```
relocate( resL, freq )
```

```
# A tibble: 35 x 3
    freq ndice roll
   <dbl> <int> <chr>
 1   170     1 1
 2   167     1 2
 3   165     1 3
 4   167     1 4
 5   170     1 5
 6   161     1 6
 7    26     2 1
 8    81     2 2
 9   142     2 3
10   193     2 4
# i 25 more rows
```

27.2 `pull()`

This will grab a column from a data frame in a list of pipe commands, which can make it easier to plug into some other tools.

Here we pull the roll column and then hand it to the `table()` command to count the number of instances of each roll.

```
resL %>% pull( roll ) %>%
  table()
```

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

27.3 set_names(list, names)

On the fly name a list before handing to map!

```
dice = 2:6
dice %>% set_names( paste0( dice, " dice" ) ) %>%
  map_df( roll_dice_extended, .id = "scenario" )
```

```
  scenario     mean median max
1   2 dice 3.000000    3.0   5
2   3 dice 3.333333    3.0   4
3   4 dice 2.500000    1.5   6
4   5 dice 3.400000    3.0   4
5   6 dice 3.833333    4.0   6
```

27.4 unpack() and pack()

These will translate a data frame column into individual columns. You can end up with a data frame column if you use map to make a new column in your data:

```
scenarios = tibble( n_dice = 1:6 )
scenarios = scenarios %>%
  mutate( result = map_df( n_dice, roll_dice_extended ) )
scenarios
```

```
# A tibble: 6 x 2
  n_dice result$mean $median  $max
   <int>       <dbl>   <dbl> <int>
1      1        4         4       4
2      2        3.5       3.5     4
3      3        1.67      2       2
4      4        4.25      5       6
5      5        3.8       4       5
6      6        3.17      4       4
```

Notice the weird `$` in the printout? This is because the three columns are inside the dataframe of `result`. You can unpack it like so:

```
unpack( scenarios, result )
```

```
# A tibble: 6 x 4
  n_dice  mean median   max
   <int> <dbl>  <dbl> <int>
1      1  4        4       4
2      2  3.5      3.5     4
3      3  1.67     2       2
4      4  4.25     5       6
5      5  3.8      4       5
6      6  3.17     4       4
```

to get your nice, normal dataframe.

28 Demonstration of the Cluster Bootstrap

29 The Cluster Bootstrap

The cluster bootstrap is a resampling technique commonly used in statistics and machine learning for estimating the variability of statistical estimators, such as mean, variance, or regression coefficients. It is especially useful when the underlying data have a complex dependence structure.

The main idea of the cluster bootstrap is to resample clusters of data points rather than individual data points. A cluster is a group of data points that are correlated with each other, for example, spatially adjacent data points or data points within the same experimental unit.

To perform the cluster bootstrap, we follow these steps:

1. Divide the original data set by the clustering variable, with each cluster containing correlated data points.
2. Randomly select clusters from the original data set with replacement, forming a bootstrap sample with the same number of clusters (but not necessarily the same number of individuals, if cluster sizes vary). If a cluster gets resampled twice, each time would be given a different, new, cluster id, so the data still has the right number of clusters and so clusters don't end up larger than in the original data.
3. Compute the statistical estimator of interest on the bootstrap sample.
4. Repeat steps 2 and 3 a large number of times, say 1000 times, to obtain a distribution of the estimator.
5. Use the distribution of the estimator to construct confidence intervals or calculate standard errors.

The cluster bootstrap is particularly useful when the underlying data have a complex correlation structure that cannot be easily accounted for using standard resampling methods, such as the simple random sampling bootstrap.

The following script is used to explore data from the Tenn Star education randomized controlled trial (RCT). It loads student-level and teacher-level data, merges them by the class ID variable, deletes observations with missing data, and creates an indicator variable for whether the student was in a small class (this is the treatment). The script then fits a linear regression model to estimate the impact of being in a small class on math scores after kindergarten, using small class status, student birth quarter, teacher education level, and teacher experience as predictors.

However, the model is not adjusted for clustering at the class level, so the standard errors may be wrong. To correct for this, we offer two solutions. First, the sandwich package is used to calculate cluster-robust standard errors. This is the classic econometric solution for clustering in a regression. Second, we demonstrate cluster bootstrapping for estimating the standard errors of the model coefficients.

29.1 The Tenn Star Data

The Tennessee Star Experiment was a large-scale randomized controlled trial (RCT) conducted in Tennessee, USA, in the mid-1980s. The experiment aimed to evaluate the effectiveness of class size reduction on student outcome. Students and teachers were randomized to different class sizes, making the treatment assignment effectively assigned at the cluster level when looking at student outcomes.

We load and prepare the data as so:

```
stud <- read.csv('data/tenn_stud_dat.csv') ## student-level data
teach <- read.csv('data/tenn_teach_dat.csv') ## teacher-level data
dat <- merge(stud, teach, by="clid")
dat <- na.omit(dat) ## for simplicity, we'll delete everyone missing any variables

## create an indicator for being in a small class
dat$small_class <- ifelse( dat$cltypek == 'small class', "yes", "no" )

# Drop some extra variables
dat <- dplyr::select( dat, -id, -cltypek, -sesk )

head( dat )
```

```
  clid   ssex srace                   sbirthq sbirthy treadssk tmathssk
1    1   male black 2nd qtr - april,may,june    1980      445      489
2    1   male black  3rd qtr - july,aug,sept    1980      393      429
3    1   male black    4th qtr - oct,nov,dec    1978      395      429
4    1   male black 2nd qtr - april,may,june    1980      403      405
5    1 female black 2nd qtr - april,may,june    1980      424      500
6    1   male black  3rd qtr - july,aug,sept    1980      414      444
      hdegk      cladk totexpk tracek small_class
1 bachelors apprentice       1  black          no
2 bachelors apprentice       1  black          no
3 bachelors apprentice       1  black          no
4 bachelors apprentice       1  black          no
5 bachelors apprentice       1  black          no
```

```
6 bachelors apprentice       1  black          no
```

```
length(unique(dat$clid)) ## 196 classes
```

```
[1] 196
```

```
nrow(dat) ## 3219 students
```

```
[1] 3219
```

Our research question is whether student math achievment was higher for kids in small classrooms vs. large ones.

29.2 The Wrong Method

If we use simple regression we are not taking the correlation of students within a given teacher into account. I.e., say a teacher happened to be effective. Then this teacher being assigned to a small class would have a positive impact on a bunch of treated students due to the single teacher. The student outcomes are correlated with each other by the single teacher. We would need to take this into account when calculating uncertainty: the students are not independent.

```
mod <- lm(tmathssk ~ small_class + sbirthq + hdegk + totexpk,
          data = dat)
res_OLS <- tidy( mod )
knitr::kable( res_OLS, digits = 2 )
```

term	estimate	std.error	statistic	p.value
(Intercept)	486.49	2.33	208.38	0.00
small_classyes	6.39	1.69	3.79	0.00
sbirthq2nd qtr - april,may,june	-5.64	2.33	-2.42	0.02
sbirthq3rd qtr - july,aug,sept	-13.85	2.27	-6.11	0.00
sbirthq4th qtr - oct,nov,dec	10.19	2.49	4.08	0.00
hdegkmaster +	-2.71	5.44	-0.50	0.62
hdegkmasters	-5.25	1.82	-2.89	0.00
hdegkspecialist	16.98	7.62	2.23	0.03
totexpk	0.58	0.15	3.88	0.00

To be clear, the above regression gives a reasonable *estimate* of the *impact* of small class size, but the standard errors, and therefore p-values, etc., are wrong.

29.3 Cluster robust standard errors

One statistical way of handling this is with the `lm_robust` package that uses the `sandwich` package that does cluster robust standard errors:

```
library( estimatr )
mod_CRVE <- lm_robust(tmathssk ~ small_class + sbirthq + hdegk + totexpk,
                  clusters = dat$clid,
                  data = dat) ## another way to get the same result (more or less)
res_CRVE <- tidy(mod_CRVE) %>%
  dplyr::select( term, estimate, std.error, p.value )
knitr::kable( res_CRVE, digits = 2 )
```

term	estimate	std.error	p.value
(Intercept)	486.49	4.26	0.00
small_classyes	6.39	3.86	0.10
sbirthq2nd qtr - april,may,june	-5.64	2.27	0.01
sbirthq3rd qtr - july,aug,sept	-13.85	2.35	0.00
sbirthq4th qtr - oct,nov,dec	10.19	2.48	0.00
hdegkmaster +	-2.71	11.50	0.83
hdegkmasters	-5.25	4.15	0.21
hdegkspecialist	16.98	15.10	0.37
totexpk	0.58	0.35	0.11

29.4 Cluster Bootstrap

Another way is to use the cluster bootstrap. It is a versitile method for getting standard errors on data that is clustered, as it keeps clusters intact.

We write an analysis function as follows:

```
my_analysis <- function( the_dat ) {
    mod <- lm(tmathssk ~  small_class + sbirthq + hdegk + totexpk,
              data = the_dat)
    broom::tidy(mod)
}

my_analysis( dat )
```

```
# A tibble: 9 x 5
  term                                estimate std.error statistic       p.value
```

```
  <chr>                                <dbl>     <dbl>     <dbl>         <dbl>
1 "(Intercept)"                      486.       2.33    208.    0
2 "small_classyes"                     6.39     1.69      3.79  0.000156
3 "sbirthq2nd qtr - april,may,june"   -5.64     2.33     -2.42  0.0155
4 "sbirthq3rd qtr - july,aug,sept"   -13.9      2.27     -6.11  0.00000000110
5 "sbirthq4th qtr - oct,nov,dec"      10.2      2.49      4.08  0.0000452
6 "hdegkmaster + "                    -2.71     5.44     -0.497 0.619
7 "hdegkmasters"                      -5.25     1.82     -2.89  0.00393
8 "hdegkspecialist"                   17.0      7.62      2.23  0.0259
9 "totexpk"                            0.579    0.149     3.88  0.000108
```

We then nest our data so each row is an entire cluster:

```
dat_nst <- dat %>%
    group_by( clid ) %>%
    nest() %>%
    ungroup()
dat_nst
```

```
# A tibble: 196 x 2
    clid data
   <int> <list>
 1     1 <tibble [20 x 11]>
 2     2 <tibble [22 x 11]>
 3     3 <tibble [21 x 11]>
 4     5 <tibble [23 x 11]>
 5     6 <tibble [21 x 11]>
 6     7 <tibble [18 x 11]>
 7     9 <tibble [23 x 11]>
 8    11 <tibble [19 x 11]>
 9    13 <tibble [17 x 11]>
10    14 <tibble [17 x 11]>
# i 186 more rows
```

We can then boostrap our data as so:

```
one_cluster_boot <- function( ) {

    dat_nst_star = slice_sample( dat_nst, n = nrow(dat_nst), replace=TRUE )
    dat_nst_star$clid = 1:nrow(dat_nst_star)

    dat_star <- unnest( dat_nst_star, cols="data" )
```

```
    my_analysis( dat_star )
}
```

Note we are regenerating the cluster ID so if we have the same cluster multiple times, each time gets a new ID.

We bootstrap and analyze a bunch of times and get standard errors:

```
boots = map_df( 1:1000, ~ one_cluster_boot(), .id = "boot" )

res_boot <- boots %>% group_by( term ) %>%
    summarise( SE = sd( estimate ) )
res_boot
```

```
# A tibble: 9 x 2
  term                                    SE
  <chr>                                <dbl>
1 "(Intercept)"                        4.38
2 "hdegkmaster + "                    12.7
3 "hdegkmasters"                       4.08
4 "hdegkspecialist"                   14.0
5 "sbirthq2nd qtr - april,may,june"    2.34
6 "sbirthq3rd qtr - july,aug,sept"     2.34
7 "sbirthq4th qtr - oct,nov,dec"       2.37
8 "small_classyes"                     3.77
9 "totexpk"                            0.363
```

We can compare to the original (WRONG) OLS estimates, the CRVE standard errors, and the bootstrap standard errors:

```
CRVE_sub <- res_CRVE %>%
  dplyr::select( -estimate, -p.value ) %>%
  rename( SE_CRVE = std.error )
OLS_sub <- res_OLS %>%
  dplyr::select( -estimate, -p.value ) %>%
  rename( SE_OLS = std.error )

# Make the table
tbl <- left_join( res_boot, OLS_sub,
                  by = "term" ) %>%
  left_join( CRVE_sub, by = "term" ) %>%
  relocate( term, statistic ) %>%
```

```
  mutate( boot_v_OLS = SE / SE_OLS,
          boot_v_CRVE = SE / SE_CRVE )

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

term	statistic	SE	SE_OLS	SE_CRVE	Boot_v_OLS	Boot_v_CRVE
(Intercept)	208.38	4.38	2.33	4.26	1.88	1.03
hdegkmaster +	-0.50	12.75	5.44	11.50	2.34	1.11
hdegkmasters	-2.89	4.08	1.82	4.15	2.24	0.98
hdegkspecialist	2.23	13.96	7.62	15.10	1.83	0.92
sbirthq2nd qtr - april,may,june	-2.42	2.34	2.33	2.27	1.00	1.03
sbirthq3rd qtr - july,aug,sept	-6.11	2.34	2.27	2.35	1.03	1.00
sbirthq4th qtr - oct,nov,dec	4.08	2.37	2.49	2.48	0.95	0.95
small_classyes	3.79	3.77	1.69	3.86	2.23	0.98
totexpk	3.88	0.36	0.15	0.35	2.43	1.03

A ratio of 1 means the estimated SEs are about the same. More than 1 means the bootstrap is returning larger SEs.

Generally, we see that the bootstrap is increasing the SEs for the level-2 coefficients (those that are talking about how clusters are different). This is good: the OLS SEs are way too small since they are not taking clustering into account.

The CRVE is basically the same as the bootstrap here, with some mild differences. All these methods are for estimating standard errors; they do not change the estimated coefficents themselves.

Bootstrapping is a simple way of getting uncertainty when you don't know how to do that with math or a package. When we can do a mathematical approximation, the bootstrapping might not be worth it, due to the extra computation. But bootstrapping, by direct simulation, can also account for things like heteroskedasticity, outliers, or other weirdness that the mathmatical approximations cannot. It is worth using in many cases due to this general applicability, versitility, and robustness.

Part IV

On Machine Learning

30 A demonstration of different machine learning methods all fit to the same data

31 Overview

This document showcases a bunch of different machine learning tools, all used on the same data set. At the end we compare the different rmse on the validation set.

For context, we will use a classic data set (e.g., Almond et al., 2005) on child birthweight. This data set was originally constructed to estimate the causal effect of maternal smoking on child birthweight; that is not what we are up to now. Our goal is to instead *predict* child birthweight directly based on observable characteristics prior to birth.

The target of interest is the child's birthweight, stored as `child_birthwt`. This is a continuous outcome. All other variables are fair game as predictors.

While this is obviously a simplified data set, the prediction problem is very real – many medical insurers and providers target services based on algorithms similar to what you will put together.

Note: For the purposes of illustration we are going to use a proportionally small training set of 10,000 observations (so we need to regularize and use fancy stuff) and a very large validation set (so we see the real comparison of our different choices). In practice we would try to use as much of our data as possible for training.

32 Setup

Load our libraries, set our random seed for reproducability:

```
library( MASS )
library( rattle )
library( glmnet )
library( tidyverse )
library( modelr )
library( broom )
library( caret )
library( ranger )
set.seed(8675309)

knitr::opts_chunk$set(echo = TRUE,
                      fig.width = 5,
                      out.width = "5in",
                      fig.align = "center")
options(list(dplyr.summarise.inform = FALSE))
theme_set( theme_classic() )
```

32.0.1 Load in the birthweight data.

We have two files: training and holdout. We keep our holdout set for our final validation after we pick our final models. This allows a fair comparison of all the methods at the end as we do not use the holdout set AT ALL for building our models.

```
# Load the data
ca_training = read.csv( "data/lbw_training.csv" )
ca_holdout = read.csv( "data/lbw_holdout.csv" )
```

Let's peek at our list of variables:

```
names( ca_training )
```

```
 [1] "child_birthwt"            "use_tobacco"
 [3] "born_in_hospital"         "MD_at_birth"
 [5] "any_drink_on_avg"         "mother_foreign_born"
 [7] "mother_age"               "father_age"
 [9] "num_prenatal_visits"      "mother_hispanic_other"
[11] "mother_black"             "father_hispanic_other"
[13] "father_black"             "mother_educ_less_than_hs"
[15] "mother_educ_hs"           "father_educ_less_than_hs"
[17] "father_educ_hs"           "adequate_care"
[19] "birth_month"              "birth_weekday"
[21] "child_female"             "mother_anemic"
[23] "mother_cardiac_disease"   "mother_lung"
[25] "mother_herpes"            "mother_diabetes"
```

33 Baseline linear model (OLS)

How do we do with simple linear regression models? We try two versions, one with just the main effects and one with all the pairwise interactions.

```
model_linear <- lm(child_birthwt ~ ., data = ca_training)
```

```
model_linear_int <- lm(child_birthwt ~ .^2, data = ca_training)
```

```
rmse( model_linear, ca_holdout )
```

```
[1] 559.436
```

```
rmse( model_linear_int, ca_holdout )
```

```
[1] 591.2767
```

Note the worse performance of the linear regression with interactions. We are likely overfitting.

34 Forward stepwise selection

Here we search for a good linear model by iteratively adding the "best" covariate until we are not improving on our model performance measure.

```
library(MASS)

# set up simplest and most complex to consider:
# `~ 1` is the "intercept only" model.  Our max model
# has everything (but no interactions).
mod_simple <- lm(child_birthwt ~ 1, data = ca_training)
mod_max <- lm(child_birthwt ~ ., data = ca_training)

# use forward stepwise selection to pick an optimal model (in terms of AIC)
# `trace=0` makes it not print out a lot of stuff to the screen about what it
# is doing.
mod_forward <- stepAIC(mod_simple,
                       scope = list(lower = formula(mod_simple),
                                    upper = formula(mod_max)),
                       direction = "forward",
                       trace = 0 )


summary(mod_forward)
```

```
Call:
lm(formula = child_birthwt ~ use_tobacco + mother_black + num_prenatal_visits +
    child_female + adequate_care + mother_age + born_in_hospital +
    any_drink_on_avg + mother_foreign_born + mother_educ_less_than_hs +
    mother_anemic + MD_at_birth + father_hispanic_other + father_black,
    data = ca_training)

Residuals:
     Min       1Q   Median       3Q      Max
-3038.05  -304.73    26.42   352.09  2399.93
```

```
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)               3254.332     60.917  53.423  < 2e-16 ***
use_tobacco               -225.548     15.441 -14.607  < 2e-16 ***
mother_black              -148.518     40.371  -3.679 0.000236 ***
num_prenatal_visits         27.124      2.088  12.989  < 2e-16 ***
child_female              -127.642     11.152 -11.445  < 2e-16 ***
adequate_careInadequate    208.301     33.101   6.293 3.25e-10 ***
adequate_careIntermediate   78.113     16.530   4.726 2.32e-06 ***
mother_age                   4.934      1.065   4.634 3.63e-06 ***
born_in_hospital          -187.136     45.238  -4.137 3.55e-05 ***
any_drink_on_avg          -242.372     59.992  -4.040 5.38e-05 ***
mother_foreign_born        -63.806     27.741  -2.300 0.021466 *
mother_educ_less_than_hs   -54.475     18.572  -2.933 0.003363 **
mother_anemic             -132.998     56.122  -2.370 0.017817 *
MD_at_birth                -28.245     16.289  -1.734 0.082948 .
father_hispanic_other      -80.786     48.069  -1.681 0.092870 .
father_black               -62.116     38.409  -1.617 0.105865
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 556.9 on 9984 degrees of freedom
Multiple R-squared:  0.08228,   Adjusted R-squared:  0.0809
F-statistic: 59.68 on 15 and 9984 DF,  p-value: < 2.2e-16
```

```
coef( mod_forward )
```

```
             (Intercept)              use_tobacco             mother_black
             3254.331679              -225.548150              -148.518044
     num_prenatal_visits             child_female   adequate_careInadequate
               27.124442              -127.642279               208.301177
adequate_careIntermediate              mother_age         born_in_hospital
               78.113446                4.934118              -187.136158
        any_drink_on_avg      mother_foreign_born  mother_educ_less_than_hs
             -242.371983               -63.806385               -54.475462
           mother_anemic             MD_at_birth     father_hispanic_other
             -132.998442               -28.244929               -80.785733
            father_black
              -62.115737
```

We can examine how many coefficients we zeroed out with this approach:

```r
# Total number of coefficients (remember to subtract off intercept)
length( coef( model_linear ) ) - 1
```

```
[1] 41
```

```r
length(coef(mod_forward)) - 1
```

```
[1] 15
```

We have dropped several covariates.

Finally, how well do we do in terms of predictive performance?

```r
rmse(mod_forward, ca_holdout)
```

```
[1] 558.9989
```

35 Ridge Regression

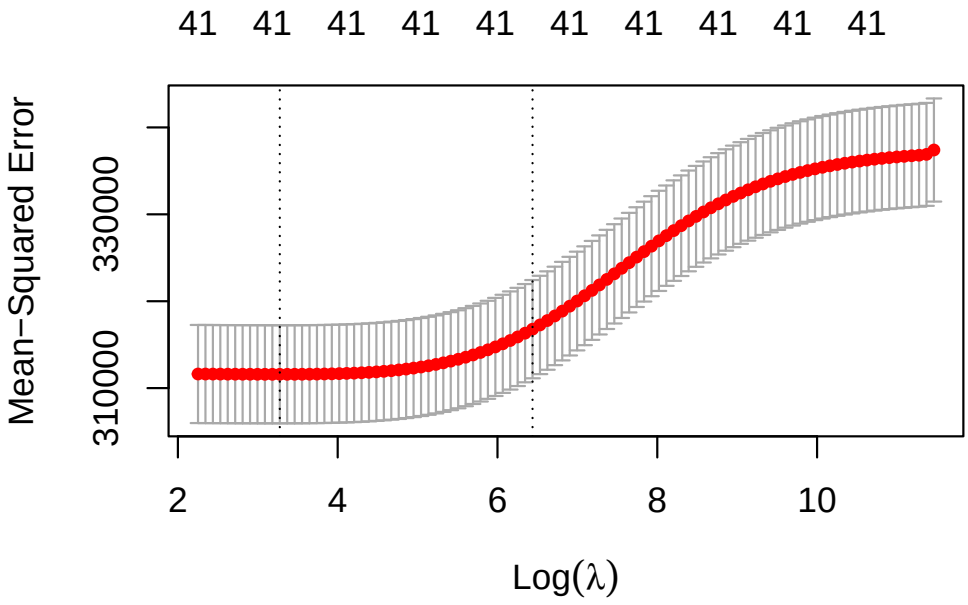
How do we do with ridge? To use `glmnet` we need to make our data into a matrix with no categorical covariates (i.e., we need to convert those to dummy variables). This "design matrix" or "model matrix" is used by those ML packages (in particular glmnet) that do not like formulae. We make this as follows:

```
x <- model.matrix(child_birthwt ~ ., ca_training)[,-1]
y <- as.numeric(ca_training$child_birthwt)

# how many covariates do we have now that we fleshed out our categorical
# ones?
dim( x )
```

```
[1] 10000    41
```

We are going to cross validate to pick the best tuning parameter.

```
model_ridge <- cv.glmnet(x = x, y = y, alpha = 0)

# Plot to see how our CV estimate of performane changes as our tuning parameter changes.
plot(model_ridge)
```



Now let's look at the model corresponding to the "1 SE" rule.

```
# Calculate coefficients at optimal lambda
predict(model_ridge, "coefficients", newx = x, s = "lambda.1se")[,1]
```

```
                (Intercept)                 use_tobacco           born_in_hospital
               3369.117138                 -109.333424                 -110.752801
               MD_at_birth            any_drink_on_avg         mother_foreign_born
                -18.307264                 -150.301619                  -25.061873
                mother_age                  father_age         num_prenatal_visits
                  2.387578                    1.087979                    9.380462
     mother_hispanic_other                mother_black       father_hispanic_other
                 -9.908412                  -73.839709                  -42.635734
              father_black    mother_educ_less_than_hs             mother_educ_hs
                -63.432580                  -35.164414                    3.169975
  father_educ_less_than_hs              father_educ_hs     adequate_careInadequate
                -17.728130                  -11.615696                    9.880843
adequate_careIntermediate             birth_monthM10             birth_monthM11
                 -4.005582                  -12.728006                   12.571847
            birth_monthM12              birth_monthM2              birth_monthM3
                -15.496326                   15.208849                  -10.498643
             birth_monthM4              birth_monthM5              birth_monthM6
                  3.210140                   13.952109                    2.536463
             birth_monthM7              birth_monthM8              birth_monthM9
                  7.638260                   -3.250663                  -11.412587
```

```
birth_weekdayD2          birth_weekdayD3          birth_weekdayD4
      -2.340281                5.611307                17.808560
birth_weekdayD5          birth_weekdayD6          birth_weekdayD7
       2.112503                8.272195                -8.147729
   child_female           mother_anemic    mother_cardiac_disease
     -62.913159              -72.776220                 4.197639
    mother_lung           mother_herpes          mother_diabetes
      12.464810               14.774745                31.563450
```

```
# Re-fit model at optimal
model_ridge_optimal <- glmnet(x = x, y = y, alpha = 0, lambda = model_ridge$lambda.1se)
```

RMSE on holdout:

We first make our holdout data into a matrix just like the training data:

```
# Do the same for holdout.
x_holdout <- model.matrix(child_birthwt ~ ., ca_holdout)[,-1]
y_holdout <- as.numeric(ca_holdout$child_birthwt)
```

Unfortunately, `glmnet` doesn't work with the `rmse()` method, so we have to write our own `rmse`. Sad!

```
# Function to calculate RMSE given our design matrix rather
# than original data.
rmse_by_hand <- function(this_model, x_holdout, y_holdout ){
  this_pred <- predict(this_model, newx = x_holdout)

  sqrt( mean( (this_pred - y_holdout)^2 ) )
}

# Calculate RMSE
rmse_by_hand(model_ridge_optimal, x_holdout, y_holdout)
```

```
[1] 565.0963
```

We can also use the helper function in `caret` that we saw in the random forest case study that takes observed and actual outcomes:

```
preds = as.numeric( predict( model_ridge_optimal, x_holdout ) )
rs = data.frame( obs = y_holdout,
                 pred = preds )
caret::defaultSummary( rs )
```

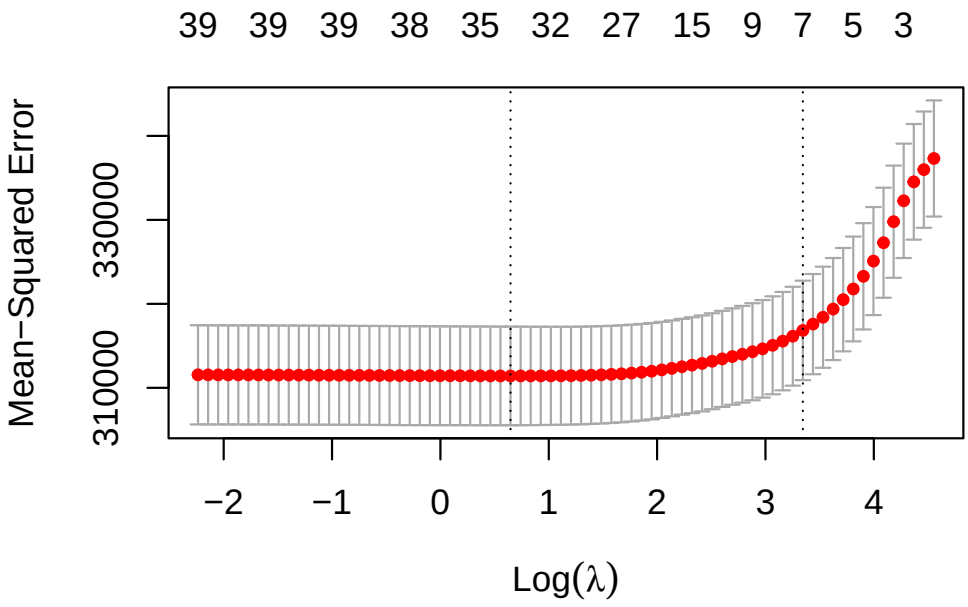
```
        RMSE     Rsquared          MAE
565.09626065   0.07662296 420.94054253
```

Note: The `predict()` method for `glmnet` gives back a matrix, not a vector of numbers. Annoying. We have to use `as.numeric` to get it to change to a list of numbers.

36 Lasso Regression

How do we do with Lasso? It is the same as ridge, except we need to set `alpha=1`.

```
model_lasso <- cv.glmnet(x = x, y = y, alpha = 1)

# Plot output
plot(model_lasso)
```




```
# Calculate coefficients at optimal lambda.  We only look at the non-zero ones.
coef <- predict(model_lasso, "coefficients", newx = x, s = "lambda.1se")[,1]

coef[ coef != 0 ]
```

```
        (Intercept)         use_tobacco    born_in_hospital          mother_age
        3288.089967          -167.324623          -13.895767            1.568242
num_prenatal_visits        mother_black        father_black       child_female
          12.063002          -106.815386          -26.432899          -74.649746
```

```r
# Re-fit model at optimal
model_lasso_optimal <- glmnet(x = x, y = y, alpha = 1, lambda = model_lasso$lambda.1se)

# Calculate RMSE for Lasso
rmse_by_hand(model_lasso_optimal, x_holdout, y_holdout)
```

```
[1] 564.7121
```

37 Single Tree (a CART)

We first do a single tree and prune it, and then we will do random forests later on.

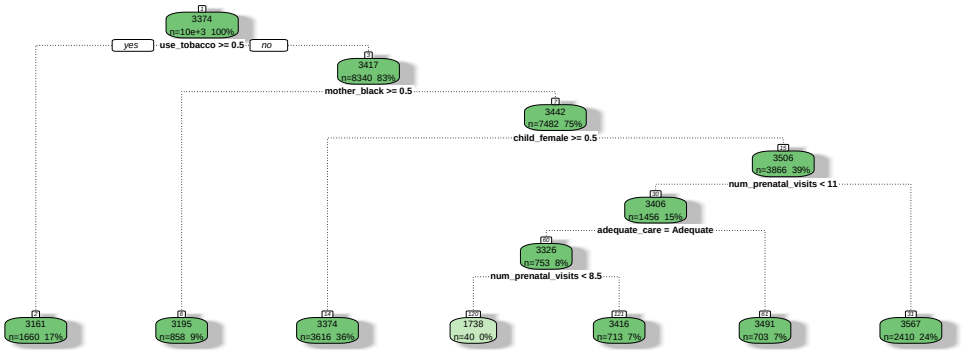
To fit a single tree we use `rpart` (Recursive Partition).

```
library(rpart)
```

```
model_tree <- rpart(child_birthwt ~ ., data = ca_training )
```

We can then plot it:

```
library( rattle )
fancyRpartPlot(model_tree)
```



Rattle 2024−Dec−26 17:42:45 lmiratrix

Our RMSE:

```
# How well do we do on the test set?
rmse(model_tree, ca_holdout)
```

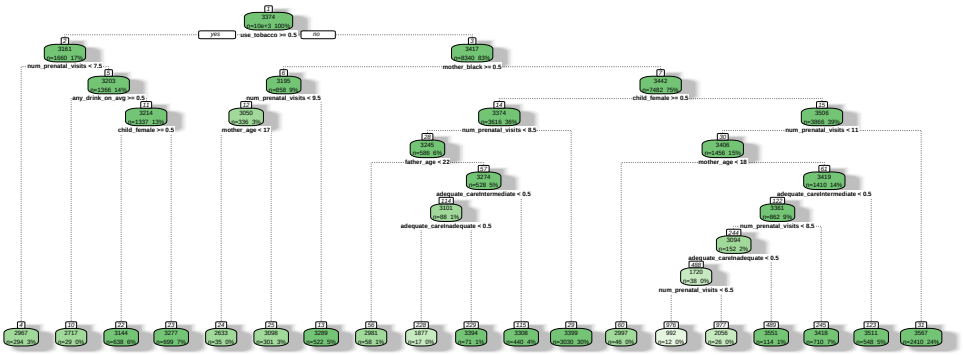
```
[1] 558.7839
```

Let's use cross validation to pick our `cp` tuning parameter (which controls the depth and complexity of our tree):

```
train_control <-
    trainControl(method = "cv",
                 number = 10)

tune_grid <- data.frame(
    cp = exp( seq( log( 0.001 ), log( 0.5 ), length.out=100 ) ) )

rpart_cv <-
    train(child_birthwt ~ ., data = ca_training,
          method = "rpart",
          na.action = "na.omit",
          tuneGrid = tune_grid,
          trControl = train_control)
```

```
Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
: There were missing values in resampled performance measures.
```

```
# Plot the final model (fit using the best chosen tuning parameter automatically)
fancyRpartPlot(rpart_cv$finalModel)
```



Rattle 2024−Dec−26 17:42:49 lmiratrix

Now let's look at the RMSE of our final model. We use predict on the entire `rpart_cv` object since it will automatically give a tree refit to the entire training data using the best selected tuning parameter:

```
rmse(rpart_cv, ca_holdout)
```

```
[1] 548.8857
```

To get the predictions themselves we would use `predict`, handing over `rpart_cv` *not* `rpart_cv$finalModel`:

```
preds = predict( rpart_cv, ca_holdout )
summary( preds )
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    992    3289    3399    3373    3511    3567
```

38 Random Forests

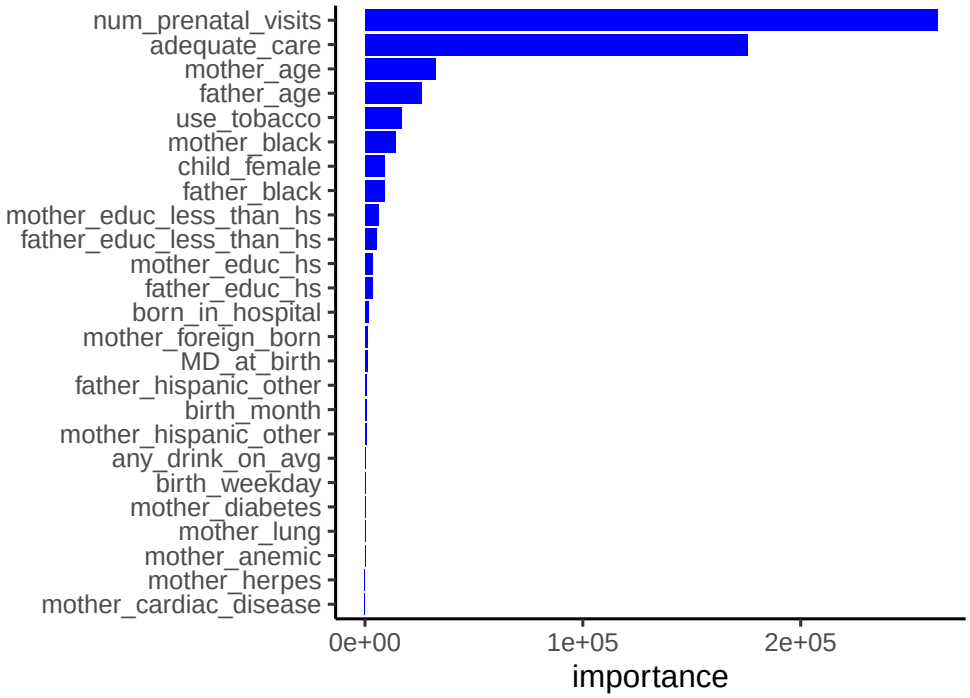
We fit a random forest with the `ranger` package as so:

```
# fit a random forest model
model_rf <- ranger( child_birthwt ~ ., data = ca_training,
                    importance = "permutation" )

model_rf


Ranger result

Call:
 ranger(child_birthwt ~ ., data = ca_training, importance = "permutation")

Type:                             Regression
Number of trees:                  500
Sample size:                      10000
Number of independent variables:  25
Mtry:                             5
Target node size:                 5
Variable importance mode:         permutation
Splitrule:                        variance
OOB prediction error (MSE):       302410.2
R squared (OOB):                  0.1037941

# Table of variable importance
importance(model_rf)

            use_tobacco         born_in_hospital              MD_at_birth
            16588.89005              1693.18649               1116.55033
       any_drink_on_avg      mother_foreign_born              mother_age
              448.65352              1328.86719              32323.92626
             father_age      num_prenatal_visits    mother_hispanic_other
            25935.76551            262591.71389               573.88371
          mother_black    father_hispanic_other             father_black
            13993.30014               792.15976              8945.04138
```

```
mother_educ_less_than_hs          mother_educ_hs father_educ_less_than_hs
              6254.52673              3419.14652               5492.35051
          father_educ_hs          adequate_care             birth_month
              3398.24303             175591.38932                614.54015
           birth_weekday           child_female           mother_anemic
                 380.24615              8977.13883                105.88669
  mother_cardiac_disease             mother_lung           mother_herpes
                -325.34716               144.81925                -46.14257
         mother_diabetes
                313.14919
```

Annoyingly, with the `ranger` package we have to make a variable importance plot by hand. (We could instead use the `randomForest` package, which has some default importance plot code. See prior scripts illustrating this if desired.) The following code makes our plot:

```
imps = importance(model_rf)
imps = tibble( var = names(imps),
               importance = imps )
imps = imps %>% arrange( importance )
ggplot( imps, aes(x = reorder( var, importance ), y = importance)) +
  geom_col(fill = "blue") +
  coord_flip() +
  labs( x = "" )
```



Note: The `reorder()` line in ggplot sets the order of our x variable so we get our variables from least to most important. The `coord_flip` makes our x-axis our y-axis and vice-versa, so we get nice horizontal bars.

The `ranger` package gets fancy with it's `predict` which makes it a bit harder to calculate out of sample RMSE. We do it like so:

```
rf_preds = predict( model_rf, data=ca_holdout )
rf_preds
```

```
Ranger prediction

Type:                             Regression
Sample size:                      118805
Number of independent variables:  25
```

```
head( predictions( rf_preds ) )
```

```
[1] 3481.162 3594.208 3517.143 3432.828 3335.656 3394.036
```

```
RMSE( predictions( rf_preds ), ca_holdout$child_birthwt )
```

```
[1] 546.1326
```

Note how we get a package of predictions from `predict` and then need to get the actual predictions out of the package with `predictions()`. Also note the capital `RMSE` method in `caret` is different from the `rmse` from the `modelr` package.

Different packages are all slightly different. Keep reference code like this to easily remember how to do basic coding tasks.

38.1 Tuning the random forest

We should tune our random forest, to figure out which specification is best. We use `caret`'s `train()` to do this (borrowing code from the case-study walk-through from the Data Science in Education textbook):

```
# setting a seed for reproducibility
set.seed(2020)

# Create a grid of different values of mtry, splitrule, and min.node.size,
# the three tuning parameters for our random forest.
```

```r
tune_grid <-
    expand.grid(
        mtry = c(2, 3, 7, 10),
        splitrule = c("variance"),
        min.node.size = c(10, 20, 50)
    )

# Fit a new model, using the tuning grid we created above.  This will take
# awhile to run.
rf_tuned <-
    train(child_birthwt ~ ., data = ca_training,
          method = "ranger",
          tuneGrid = tune_grid)

rf_tuned
```

```
Random Forest

10000 samples
   25 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 10000, 10000, 10000, 10000, 10000, 10000, ...
Resampling results across tuning parameters:

  mtry  min.node.size  RMSE      Rsquared    MAE
   2    10             560.5576  0.08226263  421.6399
   2    20             560.7701  0.08165209  421.7262
   2    50             561.0561  0.08155258  421.8141
   3    10             557.0580  0.08455732  420.1078
   3    20             557.0095  0.08532631  419.9823
   3    50             557.4392  0.08517044  419.9752
   7    10             557.9592  0.08025062  423.9294
   7    20             556.6819  0.08320896  422.4458
   7    50             555.3160  0.08691731  420.4274
  10    10             559.3643  0.07946736  426.1047
  10    20             557.4722  0.08297020  424.0431
  10    50             555.1987  0.08782704  421.2257

Tuning parameter 'splitrule' was held constant at a value of variance
RMSE was used to select the optimal model using the smallest value.
```

```
The final values used for the model were mtry = 10, splitrule = variance
 and min.node.size = 50.
```

We then take our final random forest, fit to all of our training data and using the winning tuning parameters, by simply using predict from the result of our `train()` call:

```
rmse( rf_tuned, ca_holdout )
```

```
[1] 547.0557
```

39 Comparing our machines

Now let's compare the RMSE for everyone! We add in a null model of predicting the grand mean to give a reference of how well we do when we do nothing. We can even compare all the RMSEs to the one of the mean as a point of reference:

```
model_null = lm( child_birthwt ~ 1, data=ca_training )
rmse_mean = rmse( model_null, ca_holdout )

RMSEs = c( mean = rmse_mean,
           OLS = rmse(model_linear, ca_holdout),
           OLS.quad = rmse( model_linear_int, ca_holdout ),
           forward = rmse(mod_forward, ca_holdout),
           ridge = rmse_by_hand(model_ridge_optimal, x_holdout, y_holdout),
           lasso = rmse_by_hand(model_lasso_optimal, x_holdout, y_holdout),
           CART = rmse(model_tree, ca_holdout),
           CART.prune = rmse(rpart_cv, ca_holdout),
           RF = RMSE( predictions( rf_preds ), ca_holdout$child_birthwt ),
           RF_tuned = rmse( rf_tuned, ca_holdout ) )
```

```
Warning in predict.lm(model, data): prediction from rank-deficient fit; attr(*,
"non-estim") has doubtful cases
```

```
results = tibble( model = names( RMSEs ),
                  RMSE = RMSEs,
                  perDecr = 100 * RMSE / rmse_mean )
knitr::kable( results, digits=1 )
```

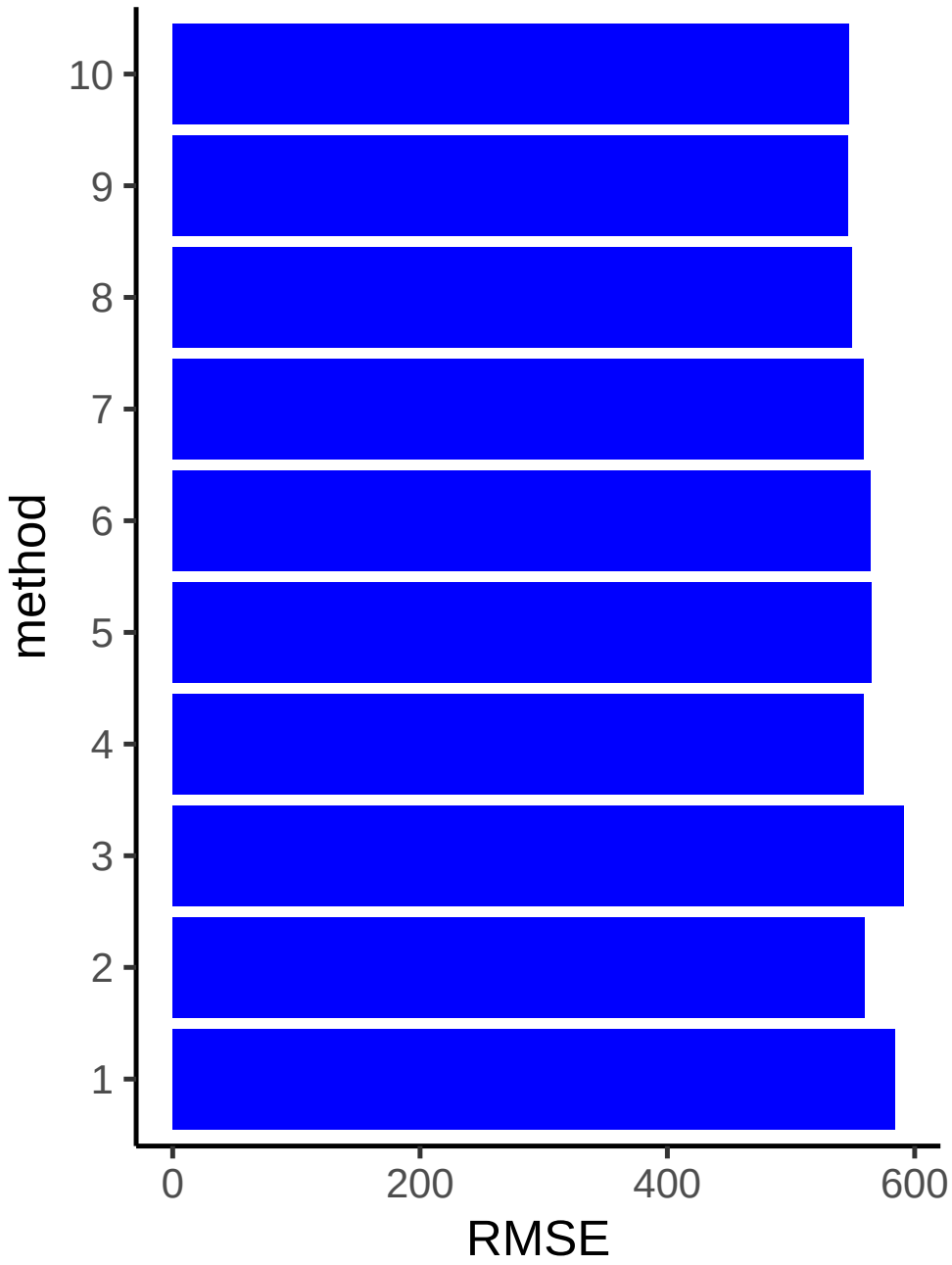
model	RMSE	perDecr
mean	584.1	100.0
OLS	559.4	95.8
OLS.quad	591.3	101.2
forward	559.0	95.7
ridge	565.1	96.8
lasso	564.7	96.7
CART	558.8	95.7

model	RMSE	perDecr
CART.prune	548.9	94.0
RF	546.1	93.5
RF_tuned	547.1	93.7

None of them are spectacular, but of them all, random forest wins, with a reduction of 6.5% in the RMSE over doing nothing. Interestingly, the tuned random forest was a tiny bit worse than the out-of-box random forest. The difference is small, so this is likely just due to instability in the tuning.

A plot of the comparisons follows. Note how the bars are similar heights, indicating that nothing is a slam dunk over, even, just predicting the mean. The interacted OLS model is so overfit it does *worse* than the mean!

```
results = rownames_to_column(results, var="method" )
results$method = reorder( results$method, 1:nrow(results) )
ggplot( results, aes(x = method, y = RMSE)) +
    geom_col(fill = "blue") +
    coord_flip()
```



A different way of thinking about the RMSE is that it is a measure of baseline variation:

```
sd( ca_holdout$child_birthwt )
```

```
[1] 584.053
```

This motivates why it is a reassonable baseline RMSE to compare the other RMSEs to.

40 Reflections on using machine learning (for final projects)

41 Test/train splits, test/train/validate, and cross validation

It is important to track what your machine learning methods are doing in terms of spliting your data inside the call. For example, the `caret` package's `train()` method will often be doing cross validation (or something similar) inside of it. You thus do not need to always do a test/train split outside of it!

Also, don't do a test/train split if you don't have a lot of data. It is too expensive! This is where cross validation is particularly useful, since it lets all of your data be used for testing, and it uses most of your data for fitting your models. It is kind of like a "have your cake and eat it too" situation.

Finally, the full test/train/validate trifecta is only if you are really, really concerned about predicting your future performance of a model. It turns out that if you use cross-validation to select the best set of tuning parameters, the final estimated accuracy from the cross-validation will generally be pretty close to what a final validation set would tell you.

Let's see how the case study in Data Science in Ed, Chapter 14 went about things. They first split the data into a training set and a testing set. They then made a grid of tuning parameters related to tree size and so forth, and then used `train()` to identify what set of tuning parameters was best for the data. In particular, `train()` used this grid and a variant of cross validation (bootstrap resampling) to repeatedly divide the training data into a part used to fit one random forest for each tuning parameter combo and a part used to evaluate all of those random forests on out of sample data. The final table produced gives the estimated error rates, and finally `train()` selects the tuning parameter combo with the best rates.

The book then took the final model (via `rf_fit2$finalModel`), which is a last model fit by `train()` to all the training data using the best found parameter combo, as their answer. To evaluate the quality of their answer, they used `finalModel` to predict outcomes for the test data they set aside at the beginning of their case study.

In our language, their "test" data was used for the final validation step, and the `train()` method was doing little test/train splits inside of it to get the estimates of out of sample error as part of the tuning process.

Note that they also found the predicted error of the finally selected final model was about the same as reported by the out of sample error by `train()`: this is not unusual. If we are not

fitting too many different combos of tuning parameter and so forth, then the estimated error from this process will often be close.

41.1 Cross validation options for caret

The `caret` package does cross validation internally as part of `train()`, if you tell it. If you don't, it does a different kind of internal repeated test/train splitting that it calls bootstrap. This is not bootstrap for inference! What it is doing is resampling the data with replacement to get a training dataset of the same size as it was passed. It will then have about a third of the data not in the training set, and it will use this for out-of-sample testing to estimate the performance of all the models fit. Finally, it repeats this a bunch of times and averages: this means each observation will be in most training sets, but will be used for testing some of the time. This is just like cross validation (except more random)!

Also, `caret`'s cross-validation is, by default, a random cross-validation: repeatedly take 10% of the data as testing, and use the rest for training. Repeat 10 times. This means each observation will be used about once for testing and about nine times for training, but not exactly. Functionally, this will generally be nearly the same as the classic CV where we divide all the data into 10 parts systematically.

It basically does not matter which form of cross-validation or splitting you use. The number of iterations will impact running time: 25 iterations will be 2.5 times longer to run than 10!

41.2 Bootstrap vs. Cross validation

Don't confuse bootstrapping with cross-validiation. Bootstrapping is a way of doing statistical inference: you use it to ask how much an estimate would change if you happened to get a different data set from the same source. This allows you to decide if, for example, a coefficient is "really" positive–if your bootstrapping doesn't really give you any negative estimates, then you can be sure that your estimate is probably not positive due to random chance.

Cross validation, by contrast, is a way of doing a lot of test/train splits because you want to know how a fit model would work on new data. It is focusing on estimating future predictive accuracy, not statistical inference.

42 Determining what variables are important

The Lasso is a sparse regression approach: as part of fitting a lasso model, you are given a subset of your original variables deemed important enough to include in the predictive model. Random forests, by contrast, allow for a second step of generating a variable importance plot, where you get a measure of how useful each variable was for making predictions.

Both these tools can be usefor for identifying factors particularly tied to your outcome. In general, you would expect those variables kept by Lasso to also have high scores in a variable importance plot. Both approaches, however, have some caveats that one should think about when interpreting results.

First, the Lasso approach gives you a final complete model built out of only a few variables. Each kept variable comes with a coefficient, and some of those coefficients will be tiny and others larger. Before directly comparing coefficients, however, spend a moment to think about how the scale of each variable matters.

For example, in the following fake dataset each variable (other than the last) has the same impact on the outcome. Put another way, they are all about the same in terms of their correlation with the outcome, as shown by the last column of the output.

```
library( glmnet )

tb = data.frame( X1 = rnorm( 1000 ),
                 X2 = rnorm( 1000, sd = 10 ),
                 X3 = rnorm( 1000, sd = 100 ),
                 X4 = rnorm( 1000 ),
                 X5 = rnorm( 1000 ) )
tb$X4_proxy = tb$X4 + rnorm( 1000, sd=0.2 )
tb$Y = with(tb, X1 + 0.1 * X2 + 0.01 * X3 + X4 + rnorm( 1000 ) )

cor( tb ) %>%
    knitr::kable( digits=2 )
```

	X1	X2	X3	X4	X5	X4_proxy	Y
X1	1.00	0.03	0.03	0.03	-0.01	0.02	0.49
X2	0.03	1.00	0.00	0.00	-0.02	0.00	0.44
X3	0.03	0.00	1.00	0.00	-0.03	0.00	0.45

	X1	X2	X3	X4	X5	X4_proxy	Y
X4	0.03	0.00	0.00	1.00	-0.04	0.98	0.46
X5	-0.01	-0.02	-0.03	-0.04	1.00	-0.05	-0.05
X4_proxy	0.02	0.00	0.00	0.98	-0.05	1.00	0.45
Y	0.49	0.44	0.45	0.46	-0.05	0.45	1.00

We can fit a Lasso model as so:

```
Xmat = model.matrix( Y ~ ., data=tb )

mod <- cv.glmnet( x = Xmat[,-1], y = tb$Y )

coef( mod )
```

```
7 x 1 sparse Matrix of class "dgCMatrix"
                     s1
(Intercept) 0.010658088
X1          0.903319008
X2          0.087108435
X3          0.009039657
X4          0.888788315
X5          .
X4_proxy    .
```

Note how we have the proxy, a little bit, and also X3. The proxy looks more important than X3 (but actually isn't important). X3 looks unimportant because the scale of X3 is so large. One fix is to standardize your covariates before putting them into your lasso:

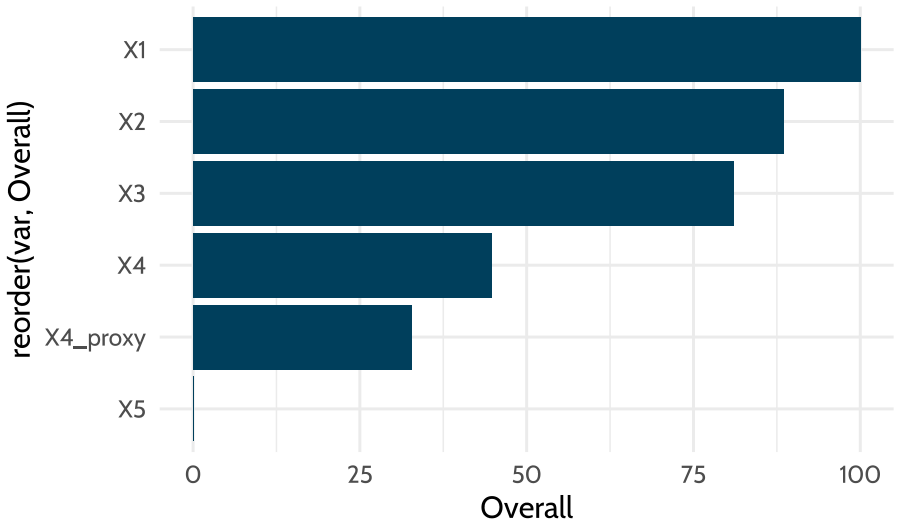
```
tb2 <- tb %>%
    mutate( X2 = scale( X2 ),
            X3 = scale( X3 ) )
Xmat2 = model.matrix( Y ~ ., data=tb )
mod2 <- cv.glmnet( x = Xmat2[,-1], y = tb2$Y )
coef( mod2 )
```

```
7 x 1 sparse Matrix of class "dgCMatrix"
                     s1
(Intercept) 0.010658088
X1          0.903319008
X2          0.087108435
X3          0.009039657
```

```
X4          0.888788315
X5          .
X4_proxy    .
```

If we use random forests and a variable importance plot, we get this:

```
set.seed(2020)
library( caret )

rf_fit2_imp <- train( Y ~ .,
        data = tb,
        method = "ranger",
        importance = "permutation" )

varImp(rf_fit2_imp) %>%
    pluck(1) %>%
    rownames_to_column("var") %>%
    ggplot(aes(x = reorder(var, Overall), y = Overall)) +
    geom_col(fill = dataedu_colors("darkblue")) +
    coord_flip() +
    theme_dataedu()
```



The variable importance plot correctly puts X1, X2, and X3 all as about the same importance (they are: the larger variance perfectly balances out the smaller coefficient). The `X4` variable gets dinged a bit because the `X4_proxy` variable can serve almost as well for prediction, so they both are deemed important.

43 Sensitivity check on trees

Say you want to present a tree in your final project to show how an outcome relates to different variables of interest. Trees, unfortunately, are notoriously unstable and thus you should see if you would get a very different tree with slight changes to your data.

Fortunately, it is easy to check this with the following:

Once you have your final tuning parameters and so forth set, repeat the following 5 times: * Bootstrap your data. * Fit a tree to the bootstrapped data, using the tuning parameters you selected. * Make a plot of your bootstrapped tree.

Now check to see if your five trees are basically similar, or entirely different, from each other. This is called a *stability check* and is a useful way of seeing if your results are particularly sensitive to mild changes in data.

44 A cool and easy way to do causal inference type stuff

Consider the problem of comparing Lyft vs Uber prices given a dataset of rides. You can't just compare the average price in each type because it is possible that, for example, Lyft rides are systematically shorter than Uber.

One direction to take is to fit two random forests, one for your Lyft data and one for your Uber data, to predict price given distance, time, weather, starting location, and anything else you have as a covariate. You then use each random forest to predict price for each observation in your entire dataset. You now have two predictions, one from your Lyft model and one from your Uber model, for each observation. Finally, you compare to see if the predictions are different on average. This gives you a "controlled comparison" where you are asking if there are systematic shifts in cost for otherwise similar things.

45 Interpreting LASSO models, and some other stuff

Back in Lab 10, or thereabouts, we walked through a LASSO model fit to student level data from the PISA exam. For our predictive task, we imagined that these students come from a single country, but in fact, we combined data from multiple nations. I sourced these data using the `learningtower` package.

46 The Story

You are newly hired analyst working in the State Ministry of Education. This morning, you open your email to find a request from your supervisor, your first project in the new job. How exciting!

Your boss writes that the Ministry has received funding to provide a one-on-one tutoring intervention for a modest number of struggling 8th grade math students. It's important that we select the right students for the intervention because we can't afford to give it to everyone. Unfortunately, we don't have previous math scores for students in the state. The first standardized math test they take is at the *end* of 8th grade.

Your boss proposes that you build a predictive model, using the other data the ministry has about students, to estimate who will perform poorly in math at the end of the 8th grade.

The only other requirement is that the model can be explained to policymakers and other higherups in the ministry. No black-box models.

Her email includes an attachment with the data on last year's 8th graders.

Our goal is to understand what is predictive of math scores. More specifically, we want to predict who the struggling students will be. There are a lot of things we could do. Remember whatever we do must be interpretable/explainable. Trees might work. Of course we'll settle on LASSO here. Generally speaking, LASSO is more interpretable than Ridge? (LASSO shrinks many predictors to zero, so we only have to explain the non-zero coefficients, which hopefully will be a small number.)

47 Fitting and interpreting LASSO

Here is a block of code that (without any actual exploration, etc.–please see the lab for that) fits the LASSO model.

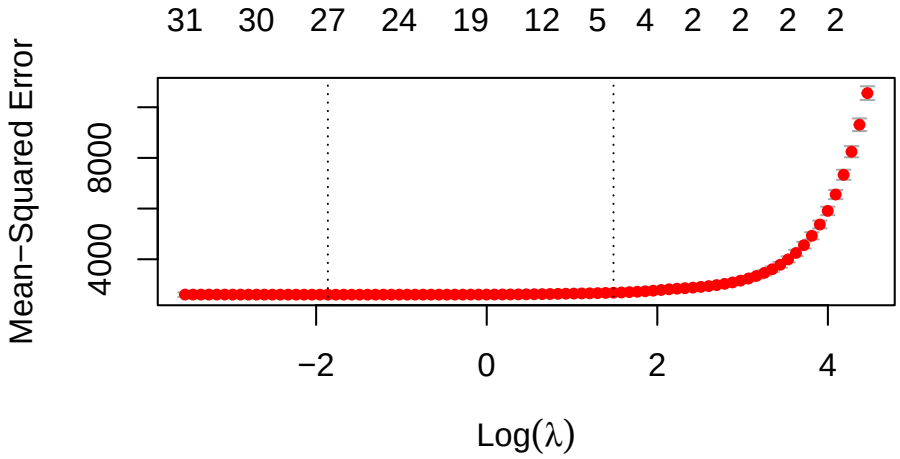
```
stu_data <- read_csv('data/student_data.csv')
stu_data = stu_data[,-c(1:2)]
stu_data <- na.omit(stu_data)

x <- model.matrix(math ~ ., stu_data)[,-1]
y <- stu_data$math

set.seed(42120)

cross_validation_results_lasso <- cv.glmnet(x = x,
                                            y = y,
                                            alpha = 1, # 1 for LASSO, 0 for ridge
                                            nfolds = 10,
                                            type.measure = "mse")

print(cross_validation_results_lasso)
```

```
Call:  cv.glmnet(x = x, y = y, type.measure = "mse", nfolds = 10, alpha = 1)

Measure: Mean-Squared Error

    Lambda Index Measure    SE Nonzero
min  0.155    69    2601 85.10      27
1se  4.422    33    2678 87.96       5
```

In the above we loaded data, dropped anything with missing data (sad!) and fit LASSO, tuning with cross validation. We get an estimated Lambda of 4.422 using the 1SE rule.

We like plots, so we generate this one:

```
plot(cross_validation_results_lasso)
```



What does it tell us? Mainly, it says that small amounts of regularization are about the same as larger amounts–note the long flat part of the curve. This means that we don't lose much if we regularize more (up to where it curves up). Regularizing more means fewer nonzero coefficients, and easier interpretation! If we regularize too much, however, our RMSE goes up quite fast.

But this plot is not particularly useful for our eventual consumers. They don't want to know that you tuned your model; they want to know what your eventual model says! These plots, therefore, do not belong in a final report.

What we want is our final model:

```
best_lambda_lasso <- cross_validation_results_lasso$lambda.1se
final_model_lasso <- glmnet(x = x,
                            y = y,
                            alpha = 1,
                            lambda = best_lambda_lasso)

coef(final_model_lasso)
```

```
32 x 1 sparse Matrix of class "dgCMatrix"
                                    s0
(Intercept)                 78.3051308
mother_educISCED 2           .
mother_educISCED 3A          .
mother_educISCED 3B, C       .
mother_educless than ISCED1  .
father_educISCED 2           .
father_educISCED 3A          .
father_educISCED 3B, C       .
```

```
father_educless than ISCED1  .
gendermale                   9.0804121
computeryes                  0.5051019
internetyes                  .
read                         0.3432561
science                      0.4877308
deskyes                      .
roomyes                      .
television1                  .
television2                  .
television3+                 .
computer_n1                  .
computer_n2                  .
computer_n3+                 .
car1                         .
car2                         .
car3+                        .
book101-200                  .
book11-25                    .
book201-500                  .
book26-100                   .
bookmore than 500            .
wealth                       .
escs                         3.0051742
```

We can print only the nonzero elements as so:

```
cc <- coef(final_model_lasso)
cc <- cc[ cc[,1] != 0, ]
cc
```

```
(Intercept)  gendermale computeryes        read     science        escs
 78.3051308   9.0804121   0.5051019   0.3432561   0.4877308   3.0051742
```

LASSO selected only a few of the many predictors available. In particular, the main predictors of math score are reading in science score, the student's economic, social and cultural status (ESCS), whether they have a computer, and whether they are assigned male in the data. Given this information, the Lasso model is saying that knowing how many televisions, cars, books, or computers are in the family is not useful for predicting math score. Neither is the mothers or fathers education level.

As an aside, it's generally hard to use categorical variables – continuous variables are easier to select because a single variable can describe a lot of variation. So this might be why these other variables are not being picked.

47.1 What variables to include?

We might not want to put all of our available variables into our model. For example, if we are trying to predict math score, we might not want to include other test scores from that same testing occasion, because were looking for contextual factors that lead to a particular outcome. Thus, we might rerun everything we've done without including reading and science, which would force the model to use the contextual variables about the students rather than other measures of their academic ability.

47.2 Coefficient size

The SIZE of the coefficients is going to be too small, in general, because LASSO shrinks coefficients towards zero to stabilize. We should not, for example, expect that the increase in math for each unit increase in reading is 0.3432561. One trick some folks do is *refit* the selected coefficients using OLS to get a new, final model:

```
# Identify nonzero coefficients
coef_indices <- which( colnames(x) %in% names(cc) )
coef_indices
```

```
[1]  9 10 12 13 31
```

```
# Subset data to nonzero covariates
x_sub <- x[, coef_indices]
head( x_sub )
```

```
  gendermale computeryes    read science    escs
1          1           1 347.710 273.827  0.7018
2          0           1 427.958 460.357  0.2091
3          1           1 271.743 337.150 -1.1074
4          1           1 336.271 368.131 -0.4944
5          1           1 402.166 322.163 -0.1840
6          0           1 350.848 342.939 -0.6022
```

```
# Fit OLS model to nonzero covariates
fit_sub <- lm(y ~ ., data = as.data.frame(x_sub))

# Print summary of refit model
summary(fit_sub)
```

```
Call:
lm(formula = y ~ ., data = as.data.frame(x_sub))

Residuals:
     Min       1Q   Median       3Q      Max
-189.522  -32.514   -0.457   32.687  239.605

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 51.00114    5.30280   9.618  < 2e-16 ***
gendermale  19.31204    1.86074  10.379  < 2e-16 ***
computeryes  8.01858    2.71953   2.949  0.00322 **
read         0.38790    0.01883  20.602  < 2e-16 ***
science      0.47816    0.01944  24.596  < 2e-16 ***
escs         4.52125    0.97928   4.617 4.04e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 51.17 on 3309 degrees of freedom
Multiple R-squared:  0.7529,    Adjusted R-squared:  0.7525
F-statistic:  2016 on 5 and 3309 DF,  p-value: < 2.2e-16
```

We can compare how much shrinkage there was:

```
ols_coef = coef( fit_sub )
ctbl = tibble( coef = names(ols_coef),
               OLS = ols_coef,
               Lasso = cc )
ctbl
```

```
# A tibble: 6 x 3
  coef           OLS  Lasso
  <chr>        <dbl>  <dbl>
1 (Intercept) 51.0   78.3
2 gendermale  19.3    9.08
3 computeryes  8.02   0.505
4 read         0.388  0.343
5 science      0.478  0.488
6 escs         4.52   3.01
```

47.3 Assessing stability (a kind of inference)

We might worry that the Lasso model is picking these variables only due to random chance. We could bootstrap this process to see if the same variables tend to get picked each time. Here is a single bootstrap iteration to illustrate:

```
datstar = slice_sample( stu_data, n = nrow(stu_data), replace=TRUE )
xstar <- model.matrix(math ~ ., datstar)[,-1]
ystar <- datstar$math

modstar <- glmnet(x = xstar,
                  y = ystar,
                  lambda = best_lambda_lasso,
                  alpha = 1 )

cc <- coef(modstar)
cc[ cc[,1] != 0, ]
```

```
                (Intercept) father_educless than ISCED1
                 83.6514190                  -2.6175776
                 gendermale                        read
                  6.5947801                   0.3550822
                    science                        escs
                  0.4687787                   3.3871490
```

We generally got the same answers, suggesting stability in what variables are selected. But we should do this a lot, and see how often each variable shows up. To do this, we need a function! And, for our function, life is always easier if we get a data frame back:

```
one_boot <- function( ) {
  datstar = slice_sample( stu_data, n = nrow(stu_data), replace=TRUE )
  xstar <- model.matrix(math ~ ., datstar)[,-1]
  ystar <- datstar$math

  modstar <- glmnet(x = xstar,
                    y = ystar,
                    lambda = best_lambda_lasso,
                    alpha = 1 )

  cc <- coef(modstar)
  cc <- cc[ cc[,1] != 0, ]
  cc = tibble( coef = names(cc), est = cc)
}
```

Then replicate:

```
rps = map_df( 1:100, ~ one_boot() )
rps %>% group_by( coef ) %>%
  summarise( n = n() ) %>%
  arrange( -n ) %>%
  knitr::kable()
```

coef	n
(Intercept)	100
gendermale	100
read	100
science	100
escs	99
computeryes	58
wealth	44
car2	26
internetyes	14
father_educless than ISCED1	13
deskyes	6
mother_educless than ISCED1	5
book201-500	2
computer_n3+	1
mother_educISCED 3A	1

The things at the top were *always* selected–they are clearly important. The things near the bottom are less so. Our initial "computer" variable seems somewhat important, but not as reliably selected as things such as reading and science.

47.4 Last words on writing up results

In writing up your results, consider the following elements:

1. Introduction: Begin by introducing the purpose of your analysis and the dataset you used. For example:

"We fit a Lasso regression model to predict the response variable [name of response variable] based on [list of predictor variables]. The dataset used was [name of dataset] and consisted of [number of observations] observations."

2. Model Selection: Explain the process you used to select the best model, including the choice of the regularization parameter. For example:

"We used cross-validation to select the value of lambda that minimized the mean squared error. The selected lambda was [value]."

3. Model Performance: Report the performance of the Lasso model, including the model's R-squared value, the root mean squared error (RMSE), and any other relevant metrics. For example:

"The Lasso model achieved an R-squared value of [R-squared value], indicating that [percentage of variation in response variable] of the variation in the response variable can be explained by the predictor variables. The RMSE of the model was [RMSE value], indicating an average error of [RMSE value] units in predicting the response variable."

You can calculate R2 by comparing the variance of the predictions to the variance of the original outcome:

$$R^2 = 1 - \frac{Var(predictions)}{Var(Y)}$$

4. Interpretation of Coefficients: Report the estimated coefficients. For example:

"Table XX shows the estimated coefficients for the predictor variables."

You might include OLS values as well, on this table.

5. Conclusion: Summarize the key findings of your analysis and any insights gained from the Lasso model. For example:

"In conclusion, we found that [some variables] predicted our outcome, suggesting [something]. On the other hand, we expected [something else] to be selected, but it never was across the bootstrap samples, indicating that, given the other variables, there is no strong connection between [this thing] and the outcome."

Remember to provide enough detail for the reader to understand your analysis, but keep your report concise and focused on the key points.

Part V

Extra Stuff

48 Main effects, interactions in linear models, and prediction

This document is designed to address a problem that I have seen several times in the final projects. In particular, people will fit a sequence of models, and noticed that the main effect changes massively when they include an interaction term. This document shows a possible solution for this "problem," a solution where you fit a model, and predict two outcomes for each observation, one with a dummy variable of interest and one without. By then looking at the average difference between these predictions, we can recover what we should think of as a main effect.

48.1 Getting the data ready

We use a data set of a bunch of behavioral outcomes measured for how a mother interacts with a child. One question of interest is whether mothers systematically interact with girls differently than they do with boys.

We first load tidyverse and load the data.

```
qm = read.table( "data/KONG.txt" )
names( qm ) = c( "wealth", "age", "fed", "med", "bookread", "hwhelp", "talkchld", "female",
head( qm )
```

```
  wealth       age fed med bookread hwhelp talkchld female sibling
1  11830 11.333330   9   9        2      2        2      0       1
2   8500 12.333330   2   5        2      2        2      1       1
3  20047 13.416670   9   8        2      2        2      1       1
4  18650 11.166670   0  12        2      2        2      1       1
5   6760 11.666670   5   0        1      2        1      0       1
6   4580  9.416667   8   8        2      2        2      1       1
```

```
nrow( qm )
```

```
[1] 1976
```

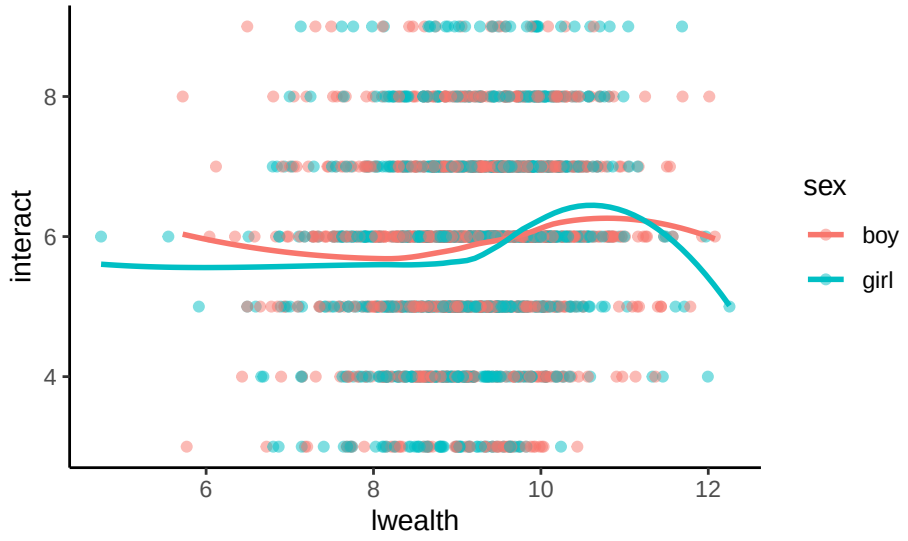
We make an aggregate measure of interaction by adding our three interview questions together. There are better ways of doing this, but this will work for now.

```
qm = mutate( qm, lwealth = log(wealth),
             interact = bookread + hwhelp + talkchld,
             sex = factor( female, levels=c(0,1), labels=c("boy","girl" ) ) )
```

To simplify things, we are going to take the middle 80% of the data based on wealth. (There is odd tail behavior that clouds the trends that I want to avoid for pedagogical purposes.) qm = filter(qm, wealth >= quantile(wealth, 0.1), wealth <= quantile(wealth, 0.9)) nrow(qm)

Lets see what we have:

```
ggplot( qm, aes( x=lwealth, y=interact, col=sex ) ) +
    geom_point( alpha=0.5 ) +
    stat_smooth( aes( group=sex ), method="loess", se=FALSE )
```



Lets look at a simple main effect of sex

```
M0 = lm( interact ~ sex + lwealth + sibling, data=qm )
summary( M0 )
```

```
Call:
lm(formula = interact ~ sex + lwealth + sibling, data = qm)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-3.4280 -0.9432  0.0462  0.9583  3.7106

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.365853   0.311989  13.994  < 2e-16 ***
sexgirl      0.005984   0.062663   0.095    0.924
lwealth      0.197600   0.032674   6.048 1.75e-09 ***
sibling     -0.219954   0.043481  -5.059 4.62e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.367 on 1972 degrees of freedom
Multiple R-squared:  0.03351,   Adjusted R-squared:  0.03203
F-statistic: 22.79 on 3 and 1972 DF,  p-value: 1.679e-14
```

48.2 Fitting an interacted model

Now let's fit a model where we interact sex and wealth. The plot does not look particularly linear, but we will proceed for illustrative purposes.

```
M1 = lm( interact ~ sex * lwealth + sibling, data=qm )
summary( M1 )
```

```
Call:
lm(formula = interact ~ sex * lwealth + sibling, data = qm)

Residuals:
    Min      1Q  Median      3Q     Max
-3.3704 -0.9534  0.0387  0.9547  3.7785

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)      4.80629    0.41588  11.557  < 2e-16 ***
sexgirl         -0.95389    0.60286  -1.582 0.113750
lwealth          0.14987    0.04422   3.389 0.000715 ***
sibling         -0.22096    0.04347  -5.083 4.06e-07 ***
sexgirl:lwealth  0.10468    0.06539   1.601 0.109569
---
```

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

Residual standard error: 1.366 on 1971 degrees of freedom
Multiple R-squared:  0.03476,   Adjusted R-squared:  0.0328
F-statistic: 17.74 on 4 and 1971 DF,  p-value: 2.543e-14
```

Notice the coefficient for sex changes a lot. This is because this is the estimated difference of girls and boys *for those with a lwealth of 0*.

To recover our "main effect" estimate we need to see what the average predicted difference would be for all the individuals in our dataset. We do this by doing the following steps:

(1) Copy our dataset to make two new ones

```
qm.g = qm.b = qm
```

(2) Now make all the folks in the `qm.g` dataset girls, and all folks in the other boys.

```
qm.g$sex = "girl"
qm.b$sex = "boy"
```

(3) Predict interaction assuming *everyone* is a girl and *everyone* is a boy:

```
pred.g = predict( M1, qm.g )
pred.b = predict( M1, qm.b )
```

(4) Then the difference of the predictions is our predicted difference in interaction for two kids with the same covariates except sex:

```
deltas = pred.g - pred.b
sex.diff = mean( deltas )
sex.diff
```

```
[1] 0.006737327
```

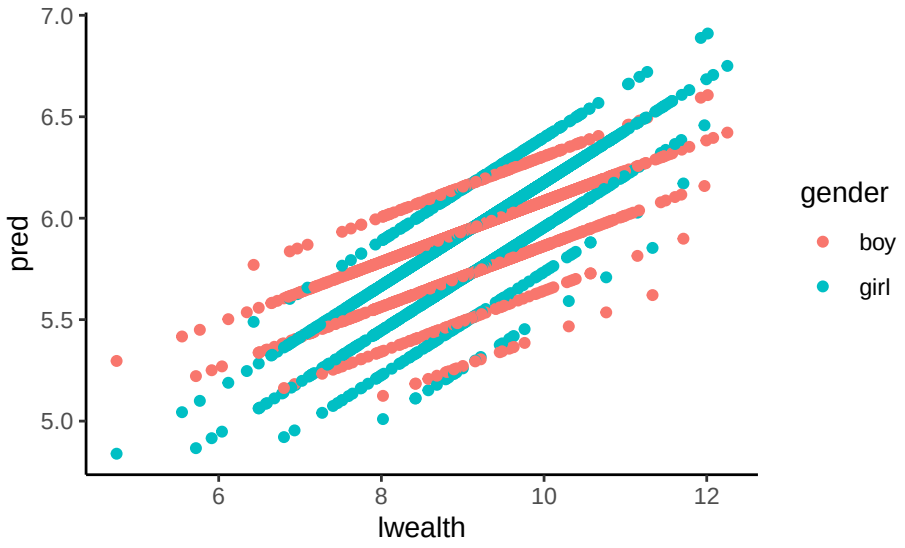
See? Now it matches more closely with the main effect estimate of M0. The core idea here is that the real interpretation of the main effect is a difference in average outcomes (when looking at a dummy varaible) between two groups, when holding all else equal. When you have interactions, the main effect is the predicted difference *for those with zeros for the interaction terms*, which might not be very sensible. We can get back to the difference between the two groups by predicting for everyone and comparing the averages.

48.3 Plotting

As a reminder, we can plot with our predictions in a nice way. The cleanest would be to put all our predictions into a dataframe, convert to long form, and plot.

(Note we use 'gender' as our key, in the following, to avoid colliding with the 'sex' variable name.)

```
qm$girl = pred.g
qm$boy = pred.b
qml = qm %>%
  pivot_longer( cols = c(girl, boy),
                names_to = "gender",
                values_to = "pred" )

ggplot( qml, aes( x=lwealth, y= pred, col=gender ) ) +
    geom_point()
```



We see different stripes for the different numbers of siblings.

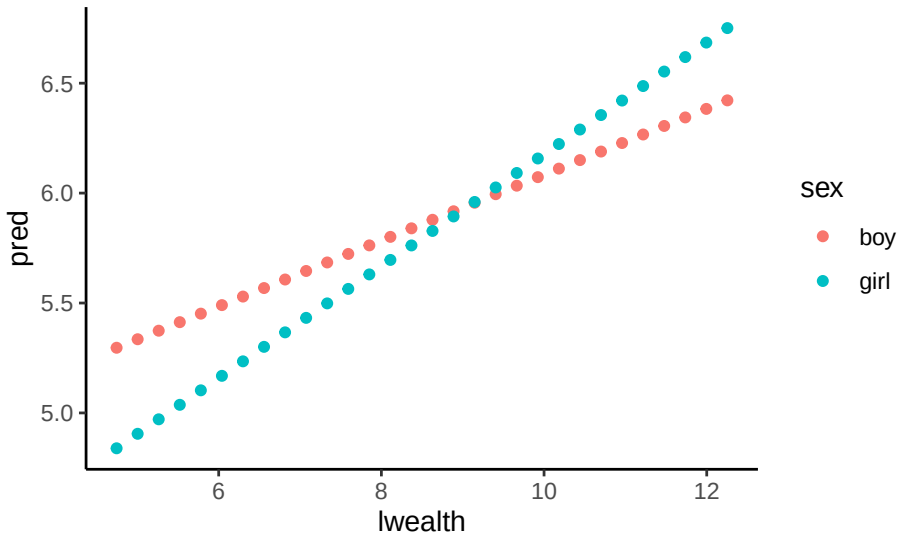
As a better way, we can get fancier with the data grid stuff.

```
library( modelr )
grid = data_grid( qm, lwealth = seq_range( lwealth, 30 ), sex, .model = M1 )
head( grid )
```

```
# A tibble: 6 x 3
  lwealth sex   sibling
    <dbl> <fct>   <dbl>
1    4.74 boy         1
2    4.74 girl        1
3    5.00 boy         1
4    5.00 girl        1
5    5.26 boy         1
6    5.26 girl        1
```

```
grid = add_predictions( grid, M1, "pred" )
```

```
ggplot( grid, aes( x=lwealth, y= pred, col=sex ) ) +
    geom_point()
```



I used `geom_point()` to show the individual predictions. Normally we would use `geom_line()`

48.4 Centering, an alternative approach

You can also center your continuous variable, which means your coefficient for your main effect on your dummy variable will correspond to the average value of the continuous. Much more interpretable!

```
qm = mutate( qm, lwealth.cent = lwealth - mean(lwealth) )
M1b = lm( interact ~ sex * lwealth.cent + sibling, data=qm )

coef( M0 )
```

```
 (Intercept)      sexgirl      lwealth      sibling
 4.365852906  0.005984015  0.197600022 -0.219953588
```

```
coef( M1b )
```

```
         (Intercept)              sexgirl         lwealth.cent
         6.181712978          0.006737327          0.149873199
             sibling sexgirl:lwealth.cent
        -0.220960207          0.104675040
```

See? Much more interpretable!

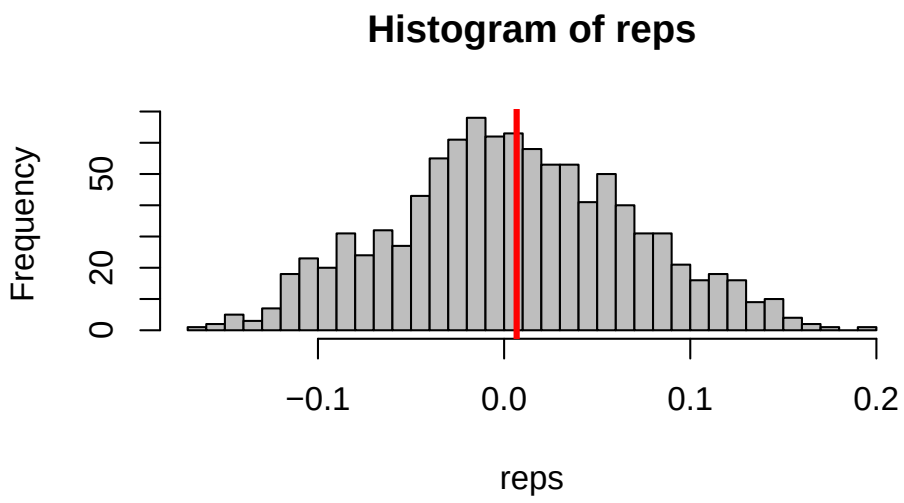
48.5 Confidence intervals for the gap?

A bootstrap is the easiest here.

```
reps = replicate( 1000, {
    qm.star = mosaic::sample( qm, replace=TRUE )

    # refit the original model with the bootstrap data
    M1.star = update( M1, data=qm.star )

    pred.g = predict( M1.star, qm.g )
    pred.b = predict( M1.star, qm.b )

    deltas = pred.g - pred.b
    mean( deltas )
} )
```

Our bootstrap confidence interval: quantile(reps, c(0.025, 0.975))

```
hist( reps, breaks=30, col="grey" )
abline( v=sex.diff, col="red", lwd=3 )
```



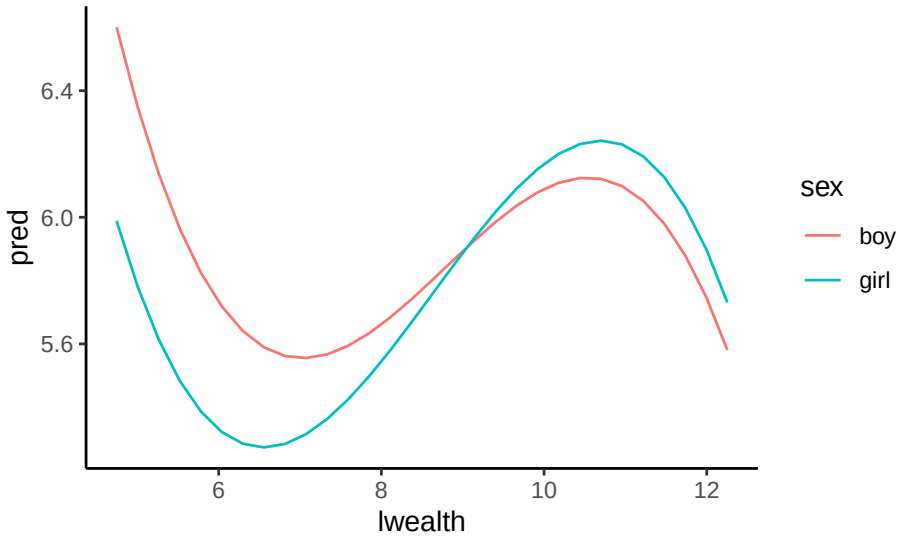
48.6 A final investigation

I don't like the linear relationship. Better to have, perhaps a cubic for both girl and boy?

```
M3 = lm( interact ~ ( lwealth + I((lwealth-mean(lwealth))^2) +
                          I((lwealth-mean(lwealth))^3) ) * sex + sibling,
         data=qm )
coef( M3 )
```

```
                           (Intercept)                                lwealth
                          3.9677413630                           0.2376064190
          I((lwealth - mean(lwealth))^2)          I((lwealth - mean(lwealth))^3)
                          0.0226463368                          -0.0264912833
                               sexgirl                               sibling
                         -1.0082023177                          -0.2159902502
                      lwealth:sexgirl I((lwealth - mean(lwealth))^2):sexgirl
                          0.1117568699                          -0.0122453285
I((lwealth - mean(lwealth))^3):sexgirl
                         -0.0007258967
```

```
grid = add_predictions( grid, M3, "pred" )
ggplot( grid, aes( x=lwealth, y= pred, col=sex ) ) +
    geom_line()
```



In truth, splines would be the best. A topic for another day (or prior handout).

48.7 Disclaimer

The coefficients of the above are not significant, and we are not finding a real difference between girls and boys in this case. But the code is designed to illustrate the core concept of using predict to get access to a real estimate of a main effect when you are fitting a model with an interaction term.

To check if a complex model is an improvement, use anova()

```
anova( M0, M1, M3 )
```

```
Analysis of Variance Table

Model 1: interact ~ sex + lwealth + sibling
Model 2: interact ~ sex * lwealth + sibling
Model 3: interact ~ (lwealth + I((lwealth - mean(lwealth))^2) + I((lwealth -
    mean(lwealth))^3)) * sex + sibling
  Res.Df    RSS Df Sum of Sq      F  Pr(>F)
1   1972 3684.0
2   1971 3679.2  1    4.7838 2.5691 0.10913
3   1967 3662.7  4   16.5032 2.2157 0.06503 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Nope, the interactions (or the more flexible cubic) are not helping us (other than helping our understanding of how to model interactions).

49 Finding and Merging Data Online

An Example with IPEDS Data

49.1 Datasets Posted on the Web

This handout provides a walk-through of downloading publicly available datasets from the web and merging them together. No guide can cover all cases, but this example should give you a sense of what to look for/think about when wrangling data posted on websites. Note that this is *not about scraping*. What we're considering here are datasets posted for download, often - but not always - found on government sites.

49.2 IPEDS

This example uses data from IPEDS, the Integrated Postsecondary Education Data System. These data are made available by NCES, the National Center for Education Statistics, a great place to start looking for education-related data in the US. The data represent a set of surveys, conducted every year, of all colleges, universities, and other post-secondary institutions that take federal student aide money (which is like, many of them). Data are available going back to 1986 (some scattered earlier years also have data). The surveys cover enrollments, graduation rates, prices, and many other things.

There are 12 total survey components that cover 9 major topics:

- Academic Libraries
- Admissions
- Completions
- Enrollment (Fall and 12-Month)
- Finance
- Graduation Rates and Outcome Measures

- Human Resources
- Institutional Characteristics
- Student Financial Aid

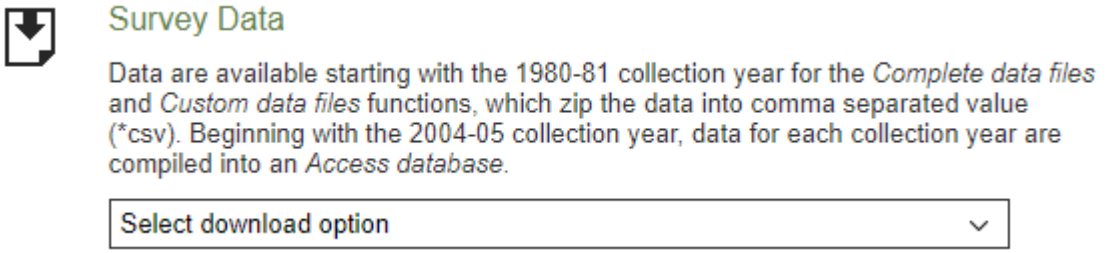
It's worth noting that some info is only collected every other year. In addition, the data are not consistent over time. New survey items get added. Old items get dropped. Definitions change. Identifying codes, like the Classification of Instructional Program (CIP) codes or the Standard Occupational Classification (SOC), are periodically reviewed and updated. Being aware of (and dealing with) these issues is part of our job as data scientists. When you're working with new data, take some time to identify these kinds of challenges before you dig too deeply into analysis.

49.3 Finding IPEDS Data

The landing page for IPEDS is here. When you open the page, you'll see many links. NCES provides a lot of information about their data (and different ways to access it). Sorting through surfeits of info like this is often the biggest challenge of working with public data sources. My description of IPEDS (in the above section) came from digging around these links.[1]

On the IPEDS page, there's a link to their data explorer. When you hit this page, you'll see links to the most recently released surveys. You can filter the datasets by year or by which of the 12 survey components you're interested in. These data are aggregated, but it's a good way to get a sense of what information is available in the individual survey components.

To *get* the survey data, look for this section of the landing page:



In the dropdown menu, select "Complete Data Files". This will take you to a page with download links for each survey component. The files we want are found in the "Data File" column. Let's download this admissions and test scores file,

[1] Along the way, I encountered some broken pages, like this link to the IPEDS data release calendar. Maybe they'll fix it, but you can often find the information you need on some other page. When that doesn't work, ask someone who's familiar with the data. When *that* doesn't work, you can email someone who works for the agency responsible for the data. In my experience, people get back to you reasonably quickly and are happy to be helpful, as long as you write politely to them.

2021	Admissions and Test Scores	Admission considerations, applications, admissions, enrollees and test scores, fall 2021	ADM2021

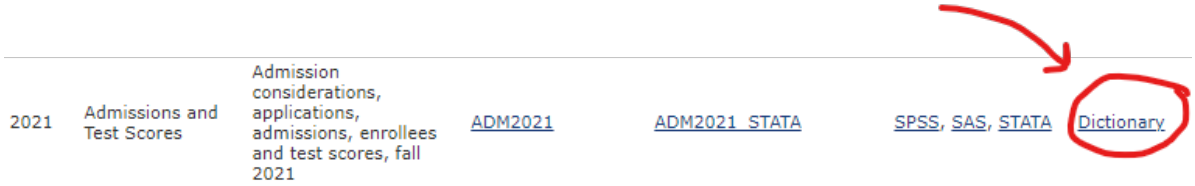
When we download the file, we should put it in a folder on our computer that's dedicated to this project. That'll make things easier once we start writing code. On my laptop, I've made a folder called "ipeds_handout". Inside *that* folder, I made another folder called "data". I put the downloaded file into that "data" folder. When I start writing code, I'll store it in the parent folder ("ipeds_handout").

The downloaded file is a zipped csv. We know how to work with a csv in R, but before we can do that, we have to unzip the file. If you haven't done this before, here's a page that walks through how to unzip files on PC or Mac. Once you have the unzipped csv, you can delete the original zip file to tidy things up.

While we're at it, let's grab one more file, data on instructional staff salaries,

2021	Instructional Staff/Salaries	Number and salary outlays for full-time nonmedical instructional staff, by gender, and academic rank: Academic year 2021-22	SAL2021_IS

Like before, unzip the file and put the csv in a project-specific folder.[2] You should also download and unzip the dictionary files for both datasets.



2021	Admissions and Test Scores	Admission considerations, applications, admissions, enrollees and test scores, fall 2021	ADM2021	ADM2021_STATA	SPSS, SAS, STATA	Dictionary

The dictionaries contain codebooks that explain what each variable and value in the dataset means. These usually aren't big files, so we can open and browse them directly in Excel. I'm going to use them soon to figure out how to find the information I'm interested in.

Now we're ready to get started.

[2]To be clear, this should be the same folder as the first dataset. Also, if you're working with many datasets for a project, it's a good idea to rename them at this stage. Give the datasets clear, descriptive names so you know what they contain. This will make it easier to write and read code later.

49.4 Our Example Case

Let's start by loading our libraries and data.

```
library(tidyverse)
library(ggplot2)
```

```
admit <- read_csv('data/adm2021.csv')
```

```
Rows: 1981 Columns: 68
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (29): XAPPLCN, XAPPLCNM, XAPPLCNW, XADMSSN, XADMSSNM, XADMSSNW, XENRLT, ...
dbl (39): UNITID, ADMCON1, ADMCON2, ADMCON3, ADMCON4, ADMCON5, ADMCON6, ADMC...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
sal <- read_csv('data/sal2021_is.csv')
```

```
Rows: 15382 Columns: 110
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (54): XSAINSTT, XSAINSTM, XSAINSTW, XSA_9MCT, XSA_9MCM, XSA_9MCW, XSATOT...
dbl (56): UNITID, ARANK, SAINSTT, SAINSTM, SAINSTW, SA_9MCT, SA_9MCM, SA_9MC...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

With the datasets we've downloaded, we can explore the relationship between assistant professor salaries and the math ability of incoming Freshmen. To begin, we narrow down each dataset to the variables of interest. To identify these, I looked at the dictionaries we downloaded from IPEDS.

```
# Keep only school IDs and average salary for 9-month
# Contract instructors
sal <- sal %>% filter(ARANK==3) %>%
  # The dictionary tells us that these are assistant profs
  select(UNITID, SA09MAT)

# Keep school Ids, # of applicants, % that submit SATs,
# and 75th pctile of SAT scores
admit <- admit %>% select(UNITID, APPLCN, SATPCT, SATMT75)
```

In other cases, you might have to do much more cleaning before you merge these datasets together. In general, the process will be to clean each dataset so they can be easily merged. (Of course, more data cleaning may be necessary, post-merge.)

49.4.1 Merging the Admissions and Salary Data

It's a good idea to do a full join. This will keep all rows from both datasets and allow us to explore which rows did and did not match, post-merge. That said, I often start with a left or right join just as a quick check of how many records match.

```
df <- left_join(sal, admit, by='UNITID')
sum(is.na(df$SATPCT))
```

```
[1] 1298
```

About half (around 1,300 institutions) don't have a match. This should be explored. Maybe it's from particular types of institutions (e.g. Vocational/Technical Schools). We might have to merge in additional data to explore the matching behavior. We might figure it out by reading carefully through the dictionaries or other documentation on IPEDS. You can take these steps by following the framework laid out above.

For now, we can run a preliminary regression and call it a day.

```
# First let's mean-center both variables
df <- df %>% mutate(across(c(SA09MAT, SATMT75), ~.x-mean(.x, na.rm=TRUE)))

summary(lm(SA09MAT ~ SATMT75, df))
```

```

Call:
lm(formula = SA09MAT ~ SATMT75, data = df)

Residuals:
   Min     1Q Median     3Q    Max
-49118  -8410   -889   8346  52056

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2829.703    459.431   6.159 1.18e-09 ***
SATMT75      164.646      6.186  26.616  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 12740 on 771 degrees of freedom
  (1529 observations deleted due to missingness)
Multiple R-squared:  0.4788,    Adjusted R-squared:  0.4782
F-statistic: 708.4 on 1 and 771 DF,  p-value: < 2.2e-16
```

It looks like assistant professors make more at schools with higher SAT math scores (as measured by the 75th percentile of the distribution). Assistant profs at schools with 100 points higher scores earn about 10k more on average. There are lots of reasons we shouldn't put much weight on this result. Lots of rows didn't merge. We haven't investigated missingness in the data. But the above steps walk through the process of a first, rough-cut analysis with IPEDS data.

50 Applied Prediction Papers

Below is a list of papers related to predictive data science and machine learning in public policy (including education). These are organized into: (1) surveys and reviews and (2) original applications. You may find many of these papers helpful as you are working on your final project and interpreting results - please peruse them with this use case in mind as well as for your general understanding of how these methods are used in the real world.

51 *Surveys and reviews*

The following list includes review articles, textbooks, and textbook chapters that provide an overview of the application of machine learning in various fields related to public policy. These fields include education, public health, urban planning, economics, and urban planning, among many others. Many of these works discuss or cite specific applications of these methods within these fields as well (including some of the references that follow after this section).

51.0.0.1 Athey, Susan. "Beyond Prediction." *Science* (American Association for the Advancement of Science), vol. 355, no. 6324, 2017, pp. 483–85, https://doi.org/10.1126/science.aal4321.

Discussion of the challenges of using machine learning predictions to improve public policies. Includes many references to real-world uses of predictive data science.

51.0.0.2 Athey, Susan. The Impact of Machine Learning on Economics. In: The Economics of Artificial Intelligence: An Agenda [Internet]. University of Chicago Press; 2018 [cited 2023 Mar 31]. p. 507–47. Available from: https://www.nber.org/books-and-chapters/economics-artificial-intelligence-agenda/impact-machine-learning-economics

This chapter discusses use of machine learning in economics and includes a discussion of "prediction policy" applications for informing economic decision-making.

51.0.0.3 Casali, Ylenia, et al. Machine learning for spatial analyses in urban areas: a scoping review. *Sustainable Cities and Society*. 2022 Oct 1;85:104050.

Authors review the use of machine learning in city/urban planning. Specific attention is given to machine learning applications involving spatial data.

51.0.0.4 Hu, Xindi C., et al. "The Utility of Machine Learning Models for Predicting Chemical Contaminants in Drinking Water: Promise, Challenges, and Opportunities." *Current Environmental Health Reports*. 2023 Mar;10(1):45–60.

Review of the use and challenges of machine learning models for predicting chemical contaminants in drinking water. Also discusses their frequent use to guide sampling efforts by prioritizing at-risk areas.

51.0.0.5 Payedimarri, Anil Babu, et al. Prediction Models for Public Health Containment Measures on COVID-19 Using Artificial Intelligence and Machine Learning: A Systematic Review. *International Journal of Environmental Research and Public Health*. 2021 Jan;18(9):4499.

A brief review of the use of machine learning (and artificial intelligence) methods to evaluate public health interventions to contain the spread of SARS-CoV-2.

51.0.0.6 Perry, Walt L. 2013. Predictive Policing: The Role of Crime Forecasting in Law Enforcement Operations. Rand Corporation.

Open access book gives an overview of common predictive policing practices.

51.0.0.7 Williamson, Ben. 2016. "Digital education governance: data visualization, predictive analytics, and 'real-time' policy instruments." Journal of Education Policy 31(2):123-141.

Broad overview of uses of modern education data. Includes a section on predictive analytics. See first full paragraph on p. 136.

51.0.0.8 Baker, R. S., Martin, T., & Rossi, L. M. (2016). Educational Data Mining and Learning Analytics. In *The Wiley Handbook of Cognition and Assessment* (pp. 379–396). John Wiley & Sons, Ltd. https://doi.org/10.1002/9781118956588.ch16

This chapter written by experts in the field provides a nice overview of educational data mining and learning analytics. Data obtained through naturally occurring log data (e.g., from learning management systems) or specifically procured sources (e.g., eyetracking) can be used to both make inferences and predictions on learning behavior and outcomes.

52 *Original applications*

The following list includes papers that apply machine learning methods to specific research questions. All are related to prediction and public policy, but span a similarly wide range of disciplines. While reading these, focus on the descriptions of the methods and results. This may be useful for your final project (and, of course, more generally!).

52.0.0.1 Bansak, Kirk, et al. "Improving Refugee Integration through Data-Driven Algorithmic Assignment." *Science* (American Association for the Advancement of Science), vol. 359, no. 6373, 2018, pp. 325–29.

In this paper, the authors propose a model that aims to predict where refugees will integrate best. They suggest governments take up their approach to make their refugee programs more efficient.

52.0.0.2 Goel S, Rao JM, Shroff R. Precinct or prejudice? Understanding racial disparities in New York City's stop-and-frisk policy. *The Annals of Applied Statistics.* 2016 Mar;10(1):365–94.

The authors of this paper used machine learning to estimate the probability that a detained individual truly has a weapon for stops related to suspicion of criminal weapon possession in New York City. Disproportionate stops by racial/ethnic groups, and the factors resulting in these disparities, are also discussed.

52.0.0.3 Hino, M, et al. Machine learning for environmental monitoring. *Nature Sustainability*. 2018 Oct;1(10):583–8.

Demonstration of how machine learning methods can help allocate resources to more efficiently conduct inspections for violations of the Clean Water Act.

52.0.0.4 Kelly, Sean, et al. "Automatically Measuring Question Authenticity in Real-World Classrooms." *Educational Researcher*, vol. 47, no. 7, 2018, pp. 451–64, Available from: https://doi.org/10.3102/0013189X18785613.

This team uses regression trees to predict which teacher questions are "authentic".

52.0.0.5 Lee Kwang-Sig, et al. Association of Preterm Birth with Depression and Particulate Matter: Machine Learning Analysis Using National Health Insurance Data. *Diagnostics*. 2021 Mar;11(3):555.

Demonstrates a use of machine learning to predict and identify the majors determinants of preterm birth in South Korea. Authors discuss that strategies to reduce air pollution could be an effective intervention based on these findings.

52.0.0.6 Yoo, Sanglim. Investigating important urban characteristics in the formation of urban heat islands: a machine learning approach. *Journal of Big Data*. 2018 Jan 24;5(1):2.

The author presents and discusses use of random forest to predict the formation of urban heat islands in Indianapolis, Indiana.

52.0.0.7 https://chicago.github.io/food-inspections-evaluation/

The city of Chicago uses a predictive model to decide which food establishments are inspected first.

52.0.0.8 Romero, C., López, M. I., Luna, J. M., & Ventura, S. (2013). Predicting students' final performance from participation in on-line discussion forums. *Computers & Education*, *68*, 458-472.

The researchers used classification and clustering to predict student performance based on a collection of features from an online discussion forum. This paper is a good example of the complicated data cleaning and feature pre-processing usually required for learning analytics (LA) work. It also demonstrates the various models that can be used in the process; plus, the paper actually explains what the models are and how they work before going into the results. Overall, it is a good entry paper into the field.

52.0.0.9 Bozick, Robert and Dalton, Benjamin. Balancing Career and Technical Education With Academic Coursework: The Consequences for Mathematics Achievement in High School. *Educational Evaluation and Policy Analysis.* 2013 Jun 1;35(2):123-38.

These researchers assessed the ability of career and technical education courses to improve student learning. Their work provides an example of the use of bootstrapping methods to calculate standard errors of regression coefficients.