HW 1 Solutions

9/7/2020

Disclaimer: Please read this first!

Hi everyone! This is the solutions document for your first homework.

This is the biggest and first assignment you'd have gotten in MA 678, and also will include a lot of very important R concepts that will be used a lot throughout. So, we wanted to make sure that this document was especially thorough to introduce a lot of the concepts, and to help highlight why things are the way they are too. It's is to help show solutions and explain them.

This means: - Your Homework doesn't need to be nearly this extensive. Homework is 5% of your grade overall, and the important thing is to practice and do what you can! - Other Homework solutions most likely will not be this extensive in the future. - If you have any questions about anything here, feel free to reach out to either TA!

7.2

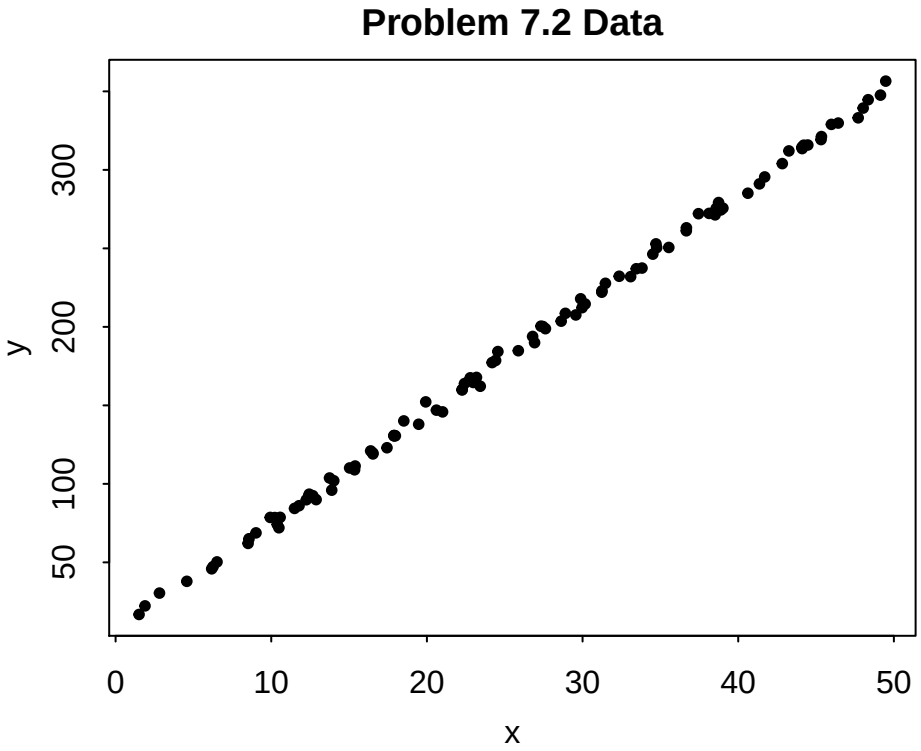
Fake-data simulation and regression: Simulate 100 data points from the linear model, y = a + bx + error, with a = 5, b = 7, the values of x being sampled at random from a uniform distribution on the range [0, 50], and errors that are normally distributed with mean 0 and standard deviation 3.

To simulate our data, we can set our constraints. We can write them in code, but I'll walk through them here first: The problem defines a and b as a = 5 and b = 7. X follows uniform distribution with 100 data points, from 0 to 50. This can be done with `runif()`, which allows us to draw random values from a uniform distribution. The error has a normal distribution with mean 0 and standard deviation of 3. This can be done with `rnorm()`, which draws sampled values from a normal distribution.

I'd also encourage you to use ?runif and ?rnorm to see how the codes work too! These help commands are very nice!

With all of that, we can calculate y by plugging our variables into the equation, $y = a + bx + error$. I'll also go straight to plotting the results, using the plot() function.

```
set.seed(100)
a = 5
b = 7
x = runif(100,0,50)
error = rnorm(100,0,3)

y = a + b*x + error

par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(x,y,pch=20,main = "Problem 7.2 Data") # Main sets the title
```




```
#pch=20 makes the points the dots that they are!
```

We can see these points are following a straight line!

7.2a

Fit a regression line to these data and display the output.

Now we can write the model. We'll use stan_glm() for now, following along with what the textbook does.

First, I'll put x and y into a data frame. You don't need to, but doing so stops rstanarm from yelling at you and giving you warnings. We'll also use print() to see the output.

```
data = data.frame(x,y)
M7.2 <- stan_glm(y~x, data=data,refresh=0) #refresh=0 hides a lot of the rstanarm output.
print(M7.2)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      y ~ x
##  observations: 100
##  predictors:   2
## ------
##             Median MAD_SD
## (Intercept) 4.8    0.7
## x           7.0    0.0
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 3.1    0.2
##
```

```
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

This output includes a lot of the simple For now, I'll just talk about the coefficients under "estimate". We can see that they'fe 4.8 and 7, which are not exactly the same as our preset a and b, but are extremely close! This means that our model is capturing the linear pattern, but is off due to some uncertainty.

If you wanted to look at or draw coefficients separately, you can also do so using the coef() function!

```
coef(M7.2)
```

```
## (Intercept)           x
##    4.817971    6.996281
```

7.2b and c

Graph a scatterplot of the data and the regression line. Use the text function in R to add the formula of the fitted line to the graph.

You can use plot() like before to make a scatterplot. To add a line, you can use the abline() function, and text() to add a label somewhere on the graph. Again, you can use ?abline and ?text to look closer.

abline() is relatively straightforward to implement. You need to put in the intercept and slope for a line that you want. We can use coef() to draw these coefficients from our model and directly input them into abline! I'll show this after showing how to make text.

The text part is more tedious and probably more tricky. The best way to make it is by combining preset characters and R output values using paste() (?paste if you want to look closer). You can separate the objects you want to put together using commas, and then set how they're separated. I'll make my label using the code below:

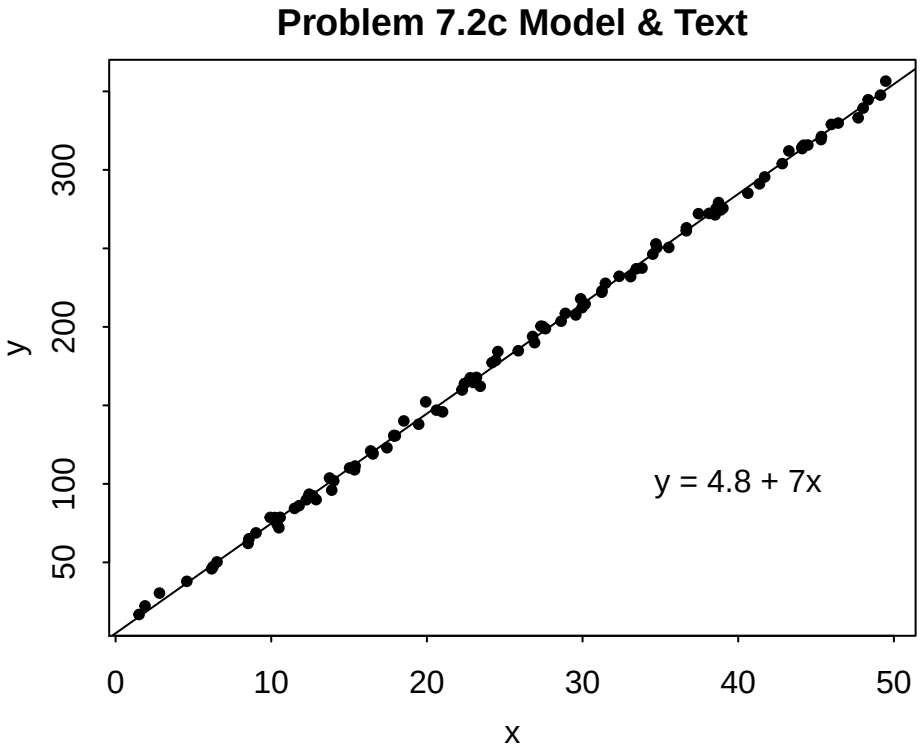
```
label = paste("y = ", round(coef(M7.2)[1],1), " + ", round(coef(M7.2)[2],1), "x", sep="")
label
```

```
## [1] "y = 4.8 + 7x"
```

And you can see it comes out very well! To describe what's happening more in depth, coef(M7.2)[1] and coef(M7.2)[2] take the first and second values from the coefficients. You can use these square brackets after an item with multiple objects in it (like a vector) to take out a specific item, or range of items from it. Also, round(), as its name would suggest rounds the value to a set number of decimal points. In this case, I rounded it to 1.

We can now put it all together using plot(), abline(), and text():

```
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(x,y,pch=20,main = "Problem 7.2c Model & Text")
abline(coef(M7.2))
text(40,100,label)
```



7.3

Fake-data simulation and fitting the wrong model: Simulate 100 data points from the model, $y = a + bx + cx^2 + error$, with the values of x being sampled at random from a uniform distribution on the range [0, 50], errors that are normally distributed with mean 0 and standard deviation 3, and a, b, c chosen so that a scatterplot of the data shows a clear nonlinear curve.

Like last time, we set up the data first. x follows a uniform distribtion, the error follows a normal distriution, we choose a, b, and c, and then we can calculate our line using the equation, $y = a + bx + cx^2 + error$. I'll make a=5, b=7, and c=10. Let's see how this looks!

We can see that the data clearly curves, displaying a nonlinear pattern!

7.3 a

Fit a regression line stan_glm(y ~ x) to these data and display the output.

Now, the question wants us to try to fit a linear model, even though this is very much not a linear pattern. We can write the same stan_glm() code regardless.

```
data = data.frame(x=x,y=y)

M7.3 = stan_glm(y~x,data=data,refresh=0)

print(M7.3)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      y ~ x
##  observations: 100
```

```
##  predictors:   2
## ------
##             Median  MAD_SD
## (Intercept) -5283.3   365.3
## x             535.6    12.0
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 1645.9  114.2
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

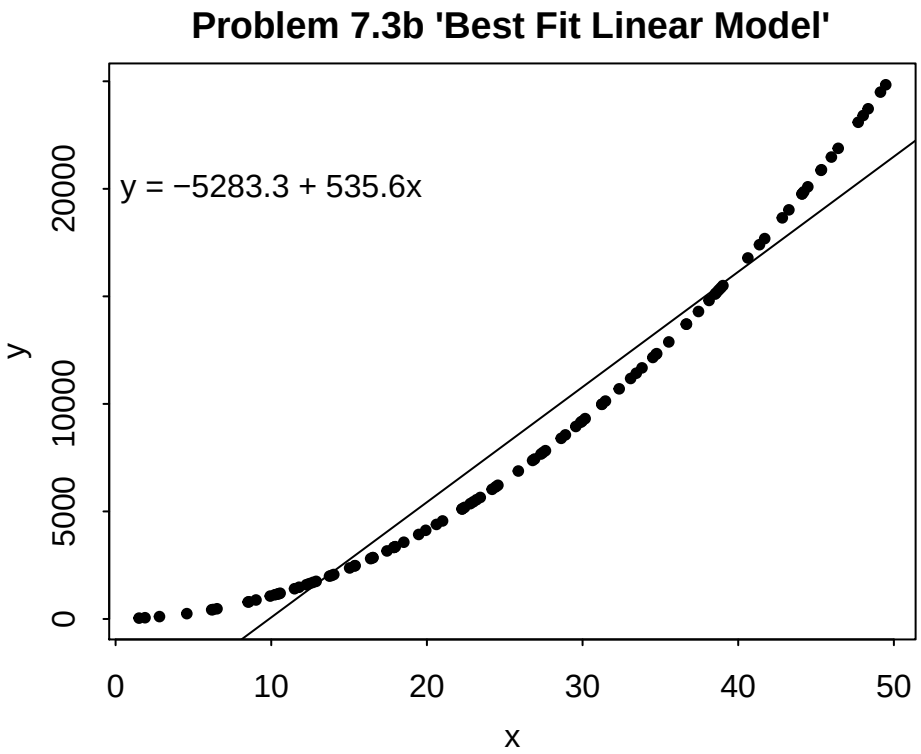
We can see that the model still fits. The errors and all the values are bigger, but that's due to the huge y-scale of our y's. We'll visualize the model in 7.3b!

7.3b

Graph a scatterplot of the data and the regression line. This is the best-fit linear regression. What does "best-fit" mean in this context?

Let's graph our "best-fit" regression! You'll be able to confirm here that the data doesn't fit at all!

```
label = paste("y = ", round(coef(M7.3)[1],1), " + ", round(coef(M7.3)[2],1), "x", sep="")
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(x,y,pch=20,main="Problem 7.3b 'Best Fit Linear Model'")
abline(coef(M7.3))
text(10,20000,label)
```



So this "best fit linear regression doesn't fit at all! The reason for this is that a"best fit" model is a version of

the model that tries to minimize some type of error. Most of the time of the time, they try to minimize a value called mean squared error (MSE), which effectively looks for the line with the lowest total distance between the observed data points and predicted points.

The "best-fit linear regression" would be the linear regression with the lowest MSE or SSE for this dataset. By default, since the pattern is not linear, it can only do so well, and even at its best it doesn't fit well. This is why it's always good to display your data and regression and verify if your regression is fitting closely. Without visualizing it, it can be hard to tell when it's messing up!

You'll also be learning to diagnose these kinds of patterns using residuals too.

7.6

Formulating comparisons as regression models: Take the election forecasting model and simplify it by creating a binary predictor defined as x = 0 if income growth is less than 2% and x = 1 if income growth is more than 2%.

First thing's first, I'll be reading in the data and looking over it. There are two easy ways to get the data!

- Go to this page: https://avehtari.github.io/ROS-Examples/examples.html.
- Find Chapter 7.
- Go to the link to the example (we're looking at ElectionsEconomy). If you want to see the code for teh examples, click on the html links.
- To get the data, click on the link at the top. You'll be taken to a github page. Click data.
- You should see the data file! Click on it, and then click on the "Raw" button.
- You can either save this raw file, or use the page url to download the file.
- In my case, I'm lazy and want a consistent method, so I'll use the url. The main downside for this is that you don't always get the chance to read more about the data, if it's included.
- I'll do it for the HW because I want it to be easy to move around, but whenever you're working on files, I'd encourage you to download it to your computer. This way, it's easier to know what the file is, and you'll keep the project all in one place even if github randomly decides to disappear.

I'll also use head() to look at the first 6 rows, to get an idea of what's in the data.

```
#hibbs <- read.table("https://raw.githubusercontent.com/avehtari/ROS-Examples/master/ElectionsEconomy/d

ghv_data_dir <- "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/"
hibbs <- read.table (paste0(ghv_data_dir,"ElectionsEconomy/data/hibbs.dat"), header=T)
#URL is:
#"https://raw.githubusercontent.com/avehtari/ROS-Examples/master/ElectionsEconomy/data/hibbs.dat"
head(hibbs)
```

```
##   year growth  vote inc_party_candidate other_candidate
## 1 1952   2.40 44.60           Stevenson      Eisenhower
## 2 1956   2.89 57.76          Eisenhower       Stevenson
## 3 1960   0.85 49.91               Nixon         Kennedy
## 4 1964   4.21 61.34             Johnson       Goldwater
## 5 1968   3.02 49.60            Humphrey           Nixon
## 6 1972   3.62 61.79               Nixon        McGovern
```

We can see a few variables here: year, inc_party_candidate (the selected candidate), other_candidate (the other one), vote (the proportion of votes), and "growth" (income growth). For this problem, we're going to be trying to associate income growth with the vote proportions.

As by the question, we need to make an indicator variable for income growth, which is 1 if income growth is more than 2% and 0 if it's less. This can be done very easily with an ifelse() statement (?ifelse)! With ifelse, we can set a condition and set two different outputs depending on if the condition is met or not!

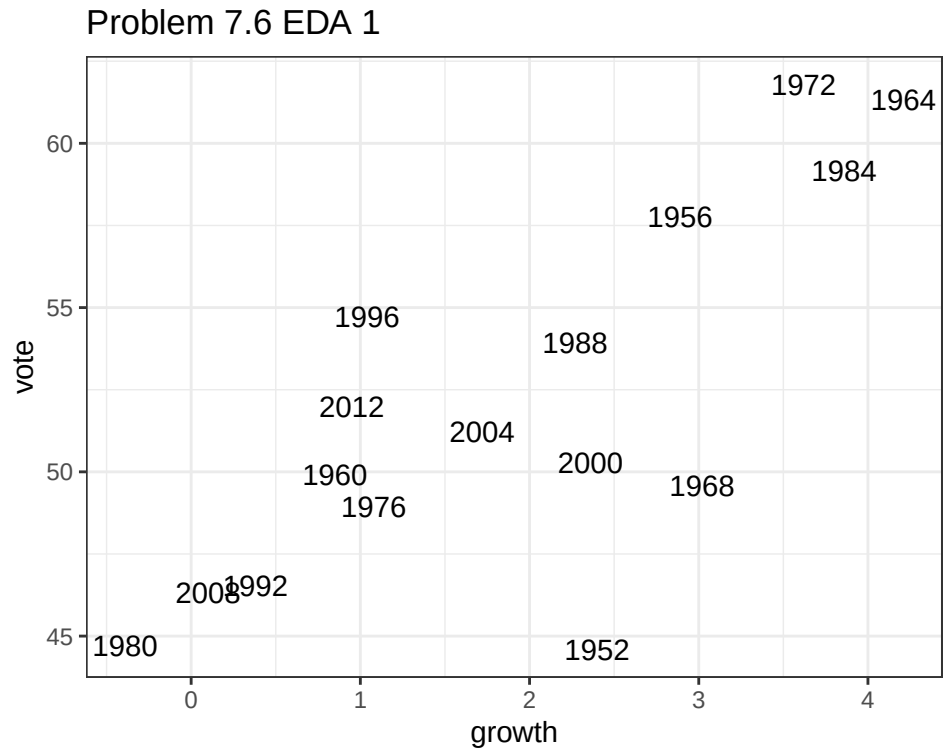
```
hibbs$x <- ifelse(hibbs$growth>=2,1,0)
head(hibbs)
```

```
##   year growth  vote inc_party_candidate other_candidate x
## 1 1952   2.40 44.60           Stevenson      Eisenhower 1
## 2 1956   2.89 57.76          Eisenhower       Stevenson 1
## 3 1960   0.85 49.91               Nixon         Kennedy 0
## 4 1964   4.21 61.34             Johnson       Goldwater 1
## 5 1968   3.02 49.60            Humphrey           Nixon 1
## 6 1972   3.62 61.79               Nixon        McGovern 1
```

We can't see too well from head, but it looks like the x's are matching our condition!

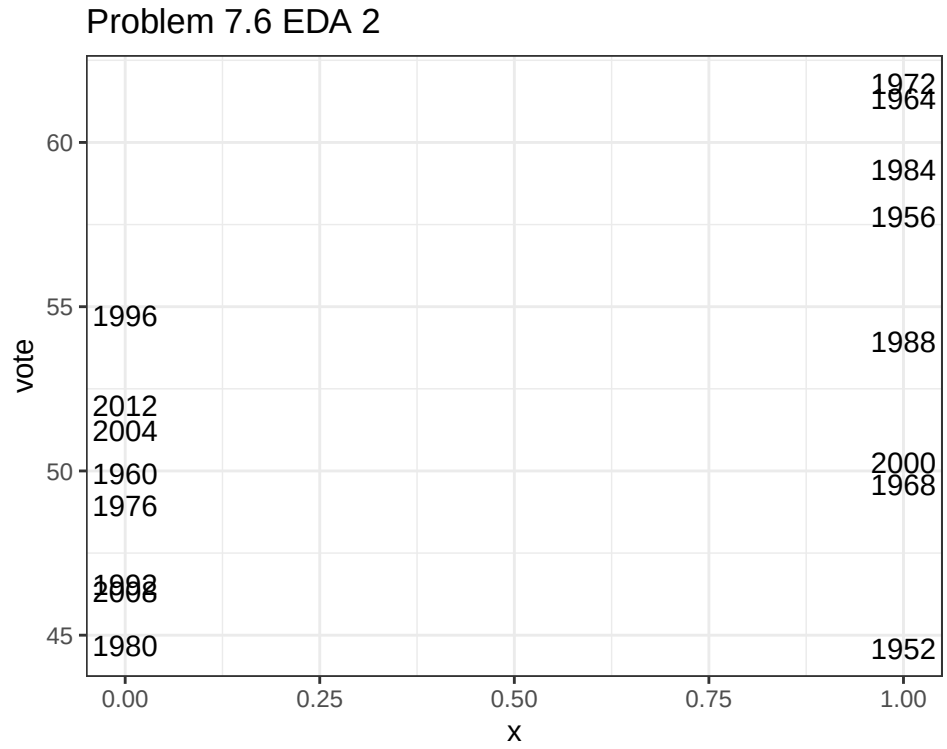
Lastly, I'm going to do a bit of exploration and compare growth to vote, labeling the points by year.

```
ggplot(hibbs) + geom_text(aes(x=growth,y=vote,label=year)) +
  theme_bw() + labs(title="Problem 7.6 EDA 1")
```



We can see some general linear relationship between the two! If we want to visualize a comparison using our new "x" instead, we can!

```
ggplot(hibbs) + geom_text(aes(x=x,y=vote,label=year)) +
  theme_bw()  + labs(title="Problem 7.6 EDA 2")
```



This is what it looks like now! Here, we can see that both values include low data points (1952 and 1980, for example), but we can see that x=1 contains more years with higher vote proportions. In the rest of the problem, we can see how this turns out with the model.

7.6a

Compute the difference in incumbent party's vote share on average, comparing those two groups of elections, and determine the standard error for this difference.

This part is kind of hard to understand at first - we basically want to find the difference in mean vote share between the x=1 and x=0 groups, and its standard deviation. This is going to take some probability math.

Let's do the easy thing first, and separate the data into two groups: group 1, where x=1, and group 2, where x=0.

```
group1 <- hibbs[hibbs$x==1,]$vote
group2 <- hibbs[hibbs$x==0,]$vote
```

Like before, we can use conditions to select rows of the data. The "==" checks if the data is identical to what follows (in this case, 1), and sets a true or false condition to set the groups. In this case, it picks everything if it's true. The "$vote" afterwards means that it's looking specifically at the vote column.

You can also see that I have a comma in these brackets: that's because this list has rows AND columns. For lists that have multiple dimensions, you need commas to separate them. In front of the comma is the row, and the back is the column. For example, hibbs[4,3] would select the value at the fourth row and third column.

We now have our groups. Let's do the easy thing first and look at the difference between their means. We can simply use the mean() function, which will take the average of the group.

```
groupmean = mean(group1) - mean(group2)
groupmean
```

```
## [1] 5.5075
```

The standard error is a little more tricky. The reason is that we split the data into two groups, and need to figure out the standard error when comparing them. The way we estimate this is by calculating the "pooled standard error" of the two groups, which is calculated using this equation equation:

$$\sqrt{\frac{\sigma_1^2 * (n_1 - 1) + \sigma_2^2 * (n_2 - 1)}{n_1 + n_2 - 2}} \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

In this case, σ_1 and σ_2 are group 1 and 2's standard deviations, and n_1 and n_2 are the number of individuals in each group.

This basically looks at the variance of the two groups, and then calculates a weighted average between the two. If one group has more points, that group's variance will have more weight on the final variance. In this case, n_1 is 8 and n_2 is 8, so they should be weighed equally.

We can use R to carry out this math! We can use the length() function to figure out how many items are in each group to set n_1 and n_2!

```
n1 = length(group1)
n2 = length(group2)
groupvar = (var(group1)*(n1-1) + var(group2)*(n2-1))/(n1+n2-2)
groupsd = sqrt(groupvar)
groupse=groupsd* sqrt(1/n1 + 1/n2)
groupse
```

```
## [1] 2.502052
```

And that's the standard error!

On a side note, I want to explore one more thing that I think a lot of people may also notice. Let's compare the standard deviation we just calculated (5.0041039)to the standard deviation of the vote column of the whole dataset.

```
sd(hibbs$vote)
```

```
## [1] 5.608951
```

We can see that these values are different! The main reason is based on populations. Calculating the standard deviation of all of the data as one group assumes it all comes from population. However, when we're comparing the means, we're usually making an implicit assumption that these two groups come from different populations with their own parameters. These assumptions lead to different numbers!

7.6b

Regress incumbent party's vote share on the binary predictor of income growth and check that the resulting estimate and standard error are the same as above.

Now, let's turn this into a regression. We can use `stan_glm()` and regress vote ~ x.

```
set.seed(100)
M7.6 <- stan_glm(vote ~ x, data=hibbs,refresh=0)
print(M7.6)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      vote ~ x
##  observations: 16
```

```
##  predictors:   2
## ------
##             Median MAD_SD
## (Intercept) 49.4    1.8
## x            5.5    2.6
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 5.2    1.0
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

To interpret what this means, remember that this is a comparison, and that x represents our groups.

- Group 2, which is when x = 0 (and when income growth is less than 2), would have an expected vote share of 49.4 + 5.5*0, or 49.4.
- Group 1, which is when x = 1 (and income growth is more than 2), would have an expected vote share of 49.4 + 5.5*1, or 4.94 + 5.5. This means that our coefficient for x represents the predicted difference between group 1 and group 2.
- Our auxiliary parameter for sigma is approximately 5.2. This would be the expected standard deviation between the two groups.

Compare these coefficients to our calculated mean and variance, 5.5 and 5. These values again aren't exact but are very close! The "MAD_SD" value for x is also close to our pooled standard eroor too!

Let's also do this in base `lm()`.

```
M7.6_v <- lm(vote ~ x, data=hibbs)
summary(M7.6_v)
```

```
##
## Call:
## lm(formula = vote ~ x, data = hibbs)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.2087  -3.3706   0.1287   3.3037   6.9812
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   49.301      1.769  27.866 1.15e-13 ***
## x              5.508      2.502   2.201    0.045 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.004 on 14 degrees of freedom
## Multiple R-squared:  0.2571, Adjusted R-squared:  0.204
## F-statistic: 4.845 on 1 and 14 DF,  p-value: 0.045
```

In fact the numbers match up (up to rounding difference).

8.8

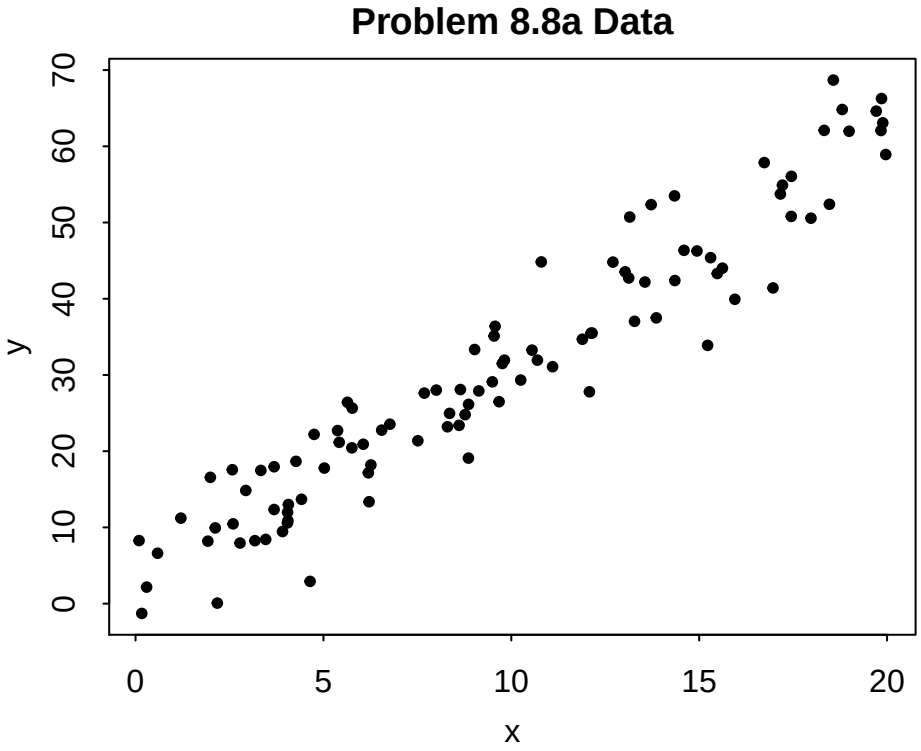
Comparing `lm` and `stan_glm`: Use simulated data to compare least squares estimation to default Bayesian regression:

8.8a

Simulate 100 data points from the model, y = 2 + 3x + error, with predictors x drawn from a uniform distribution from 0 to 20, and with independent errors drawn from the normal distribution with mean 0 and standard deviation 5. Fit the regression of y on x data using lm and stan_glm (using its default settings) and check that the two programs give nearly identical results.

Let's simulate some more data! - a = 2 - b = 3 - x is 100 points based on a uniform distribution from 0 to 20 - The error is normally distributed, has mean 0, standard deviation 5 - y = a + bx + error

```
a = 2
b = 3
x = runif(100,0,20)
error = rnorm(100,0,5)

y=a + b*x + error

data=data.frame(x,y)
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(x,y,pch=20,main="Problem 8.8a Data")
```



Our data has a wide-ish spread, but is still linear.

Now, let's try making our regressions! I'll be printing the coef() values separately first.

```
lm8.8 <- lm(y~x, data=data)
coef(lm8.8)
```

```
## (Intercept)           x
##    2.249946    2.957865
```

```
stan8.8 <- stan_glm(y~x,data=data,refresh=0)
coef(stan8.8)
```

```
## (Intercept)           x
##    2.266115    2.955114
```

We can see that their coefficients are very very similar! This is because at the end of the day, they're trying to estimate similar values. We can look further into this by comparing them with print().

```
print(lm8.8)
```

```
##
## Call:
## lm(formula = y ~ x, data = data)
##
## Coefficients:
## (Intercept)            x
##       2.250        2.958
```

```
print(stan8.8)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      y ~ x
##  observations: 100
##  predictors:   2
## ------
##             Median MAD_SD
## (Intercept) 2.3    1.0
## x           3.0    0.1
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 5.1    0.4
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

You can kind of guess based off of the display, but lm is more streamlined, while stan_glm provides more information on the distribution and variances of each parameter. The reason for this is that stan_glm has many background processes, which makes it take longer to run, but will also allow you to do more things with your predictions than lm() does. These are more technical, and we'll talk about them later in the course!

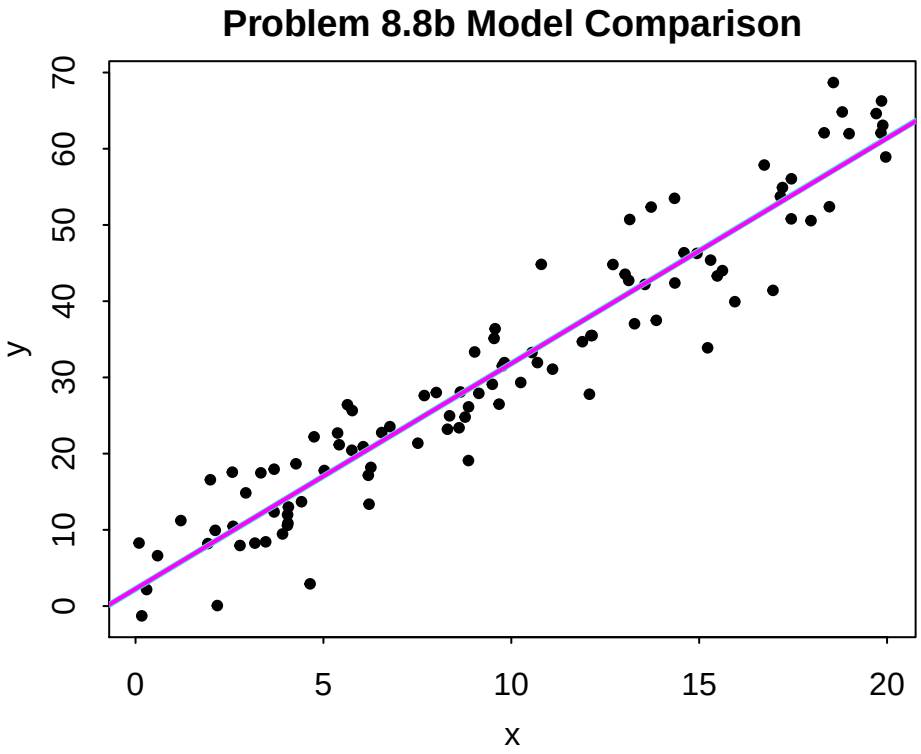
8.8b

Plot the simulated data and the two fitted regression lines.

Let's use abline() to plot them. We can set colors for each line to distinguish them. Let's make stan_glm() purple and lm() blue!

```
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(x,y,pch=20,main="Problem 8.8b Model Comparison")
```

```
abline(coef(lm8.8),col="turquoise",lwd=3)
abline(coef(stan8.8),col="magenta",lwd=2)
```



Both lines are overlapping to the point where we can hardly see the blue one! I had to make it thicker to make it even slightly visible!

8.8c

Repeat the two steps above, but try to create conditions for your simulation so that lm and stan_glm give much different results.

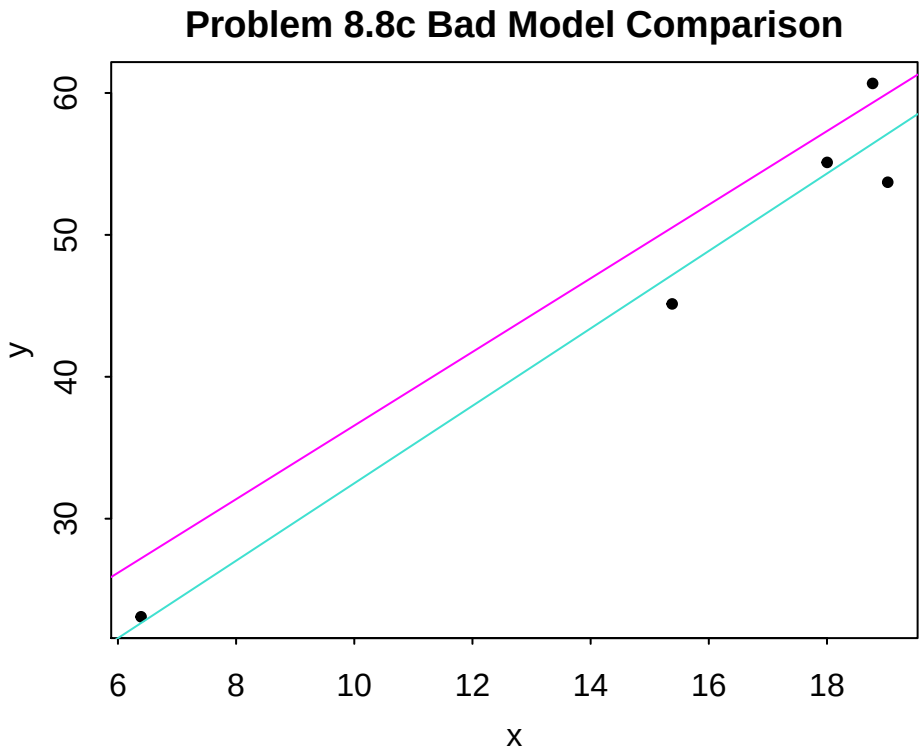
Now, let's try to make them different! I'll write the code in one chunk to make it easier to run, and will skip ahead to the display rather than focusing on the coefficients. There will be warnings.

There are a lot of ways to go about this and it's surprisingly hard (at least for me), and I'm interested to see the things you try. My method is: - Now there are only 5 data points. - change the number of iterations stan is using from the normal value to 20 in order to make it much more unstable and much worse. Stan will yell at me. I used 20 because I tried using 10 and the second line didn't even make it onto the graph. If you want to see more about this, look at question 10.8!

```
a = 2
b = 3
x = runif(5,0,20)
error = rnorm(5,0,10)

y=a + b*x + error

data=data.frame(x,y)

lm8.8 <- lm(y~x, data=data)
stan8.8 <- stan_glm(y~x, data=data,iter=20,  refresh=0)
```

```
## Warning: There were 1 chains where the estimated Bayesian Fraction of Missing Information was low. Se
## http://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: The largest R-hat is 1.61, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess

## Warning: Markov chains did not converge! Do not analyze results!
```

```
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(x,y,pch=20,main="Problem 8.8c Bad Model Comparison")
abline(coef(lm8.8),col="turquoise",lwd=1)
abline(coef(stan8.8),col="magenta",lwd=1)
```



And the lines are different! Let's look at the coefficients.

```
coef(lm8.8)
```

```
## (Intercept)           x
##    5.208227    2.728635
```

```
coef(stan8.8)
```

```
## (Intercept)           x
##   10.622874    2.593877
```

10.1

Regression with interactions: Simulate 100 data points from the model, $y = b_0 + b_1 x + b_2 z + b_3 xz + error$, with a continuous predictor x and a binary predictor z, coefficients b=c(1,2,-1,-2), and errors drawn independently from a normal distribution with mean 0 and standard deviation 3, as follows. or each data point i, first draw z_i, equally likely to take on the values 0 and 1. Then draw x_i from a normal distribution with mean z_i and standard deviation 1. Then draw the error from its normal distribution and compute y_i.

Once again, let's start with simulations! This one is definitely a little bit more tricky to code, and there are a lot of ways to go about doing this. This process invovles: - Setting up your coefficients for β. We can put these all in one vector. - Setting up error with rnorm. - Randomly producing each z. We can use rbinom() for this. - Randomly producing each x based off of each z. - Calculating for y based off of the first set of values. We can use brackets around β for each coefficient.

There are other ways you can do this using other features, like for loops or other things, but this is the most straightforward way I found!

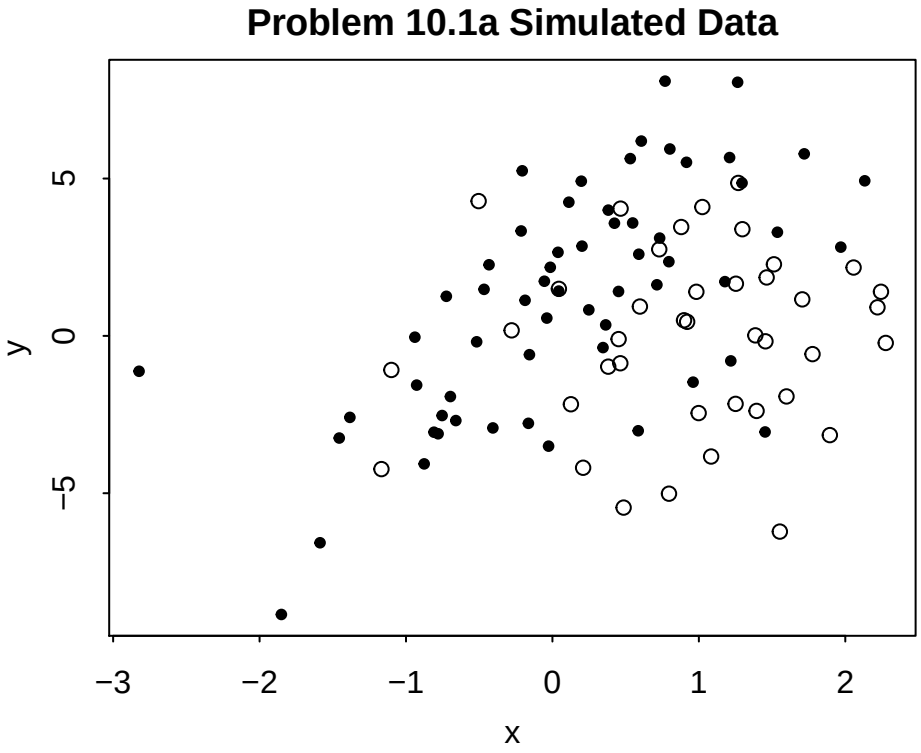
```
set.seed(101)
b = c(1,2,-1,-2)
error = rnorm(100,0,3)
z <- rbinom(100,1,0.5)
x <- rnorm(100,z,1)

y = b[1] + b[2]*x + b[3]*z + b[4]*x*z + error
```

10.1a

Display your simulated data as a graph of y vs. x, using dots and circles for the points with z = 0 and 1, respectively.

Now, we can plot them. To change the shape, we can use the "pch" argument in plot(). We can google online and see that the open points are pch=1, and the dots are pch=20. We need to set pch to 1 when z=1, and to 20 when z=0. We can do this using an ifelse statement, and then put the argument into the pch part to have it give each point its desired shape!

```
shape = ifelse(z==1,1,20)
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(x,y,pch=shape,main = "Problem 10.1a Simulated Data")
```



We can see that the points are scattered. Even though most of the z=1 (open) points are higher on the x scale, there's still a lot of overlap!

10.1b

Fit a regression predicting y from x and z with no interaction. Make a graph with the data and two parallel lines showing the fitted model.

Let's fit our regression with no interaction first. Again, I'll put the data into a data frame first!

```
data = data.frame(y=y,x=x,z=z)

M10.1 <- stan_glm(y ~ z + x,data=data,refresh=0)

print(M10.1)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      y ~ z + x
##  observations: 100
##  predictors:   3
## ------
##             Median MAD_SD
## (Intercept)  0.9    0.4
## z           -2.6    0.7
## x            1.7    0.3
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 3.0    0.2
##
```

```
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

Without the interaction, we can see that it still fits a model. We can look closely at our coefficients too using coef()! I'll put them in an object named b_hat for future use.

```
b_hat <- coef(M10.1)
b_hat
```

```
## (Intercept)           z           x
##   0.9163943  -2.6387167   1.7000134
```

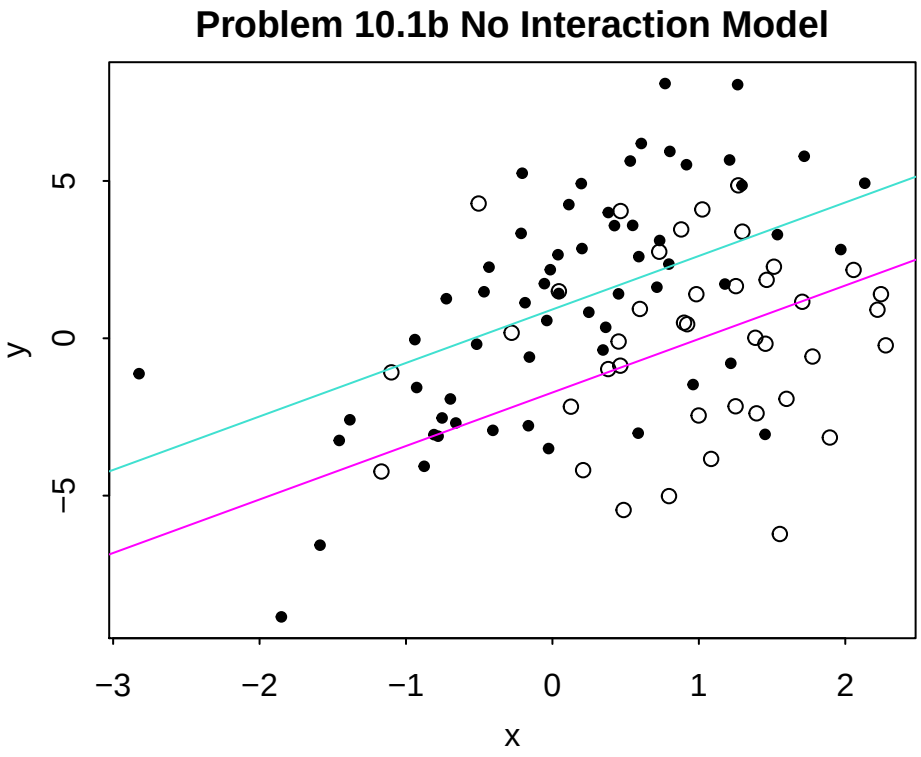
We can see our intercept, and coefficients for z and x. Let's try to interpret it!

- Our baseline is the intercept coefficient, 0.9.
- z is a constant coefficient, -2.6. When z is 0, nothing happens. When z=1, it's present in the equation, and changes the line's intercept into -1.7.
- x is the only changing value that would have a slope. This slope would be 1.7, and this slope is consistent for z=1 and z=0.

We can see that we'll have two lines with different intercepts but the same slope. We can plot these using abline(). Rather than using coef() like before, we'll have to bring out the individual coefficients. We can do that by calling b_hat and using brackets!

I'll make the line magenta when z=1, and turquoise when z=0.

```
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(x,y,pch=shape, main = "Problem 10.1b No Interaction Model")
abline(b_hat[1] + b_hat[2], b_hat[3],col="magenta")
abline(b_hat[1], b_hat[3],col="turquoise")
```



Again, we can see these lines are parallel, due to the same slope, but it estimates the z=1 group to have a lower intercept!

10.1c

Fit a regression predicting y from x, z, and their interaction. Make a graph with the data and two lines showing the fitted model.

Now, let's move onto interactions! To write this, you can either use z + x + z:x or z*x. I like using the former, so that's how I'll write it, but both work!

```
M10.1 <- stan_glm(y ~ z + x + z:x,data=data,refresh=0)
print(M10.1)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      y ~ z + x + z:x
##  observations: 100
##  predictors:   4
## ------
##             Median MAD_SD
## (Intercept)  0.8    0.4
## z           -1.3    0.8
## x            2.3    0.4
## z:x         -1.9    0.7
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 2.8    0.2
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

We now have one more coefficient for our interaction term, z:x! Let's look more closely!

```
b_hat <- coef(M10.1)
b_hat
```

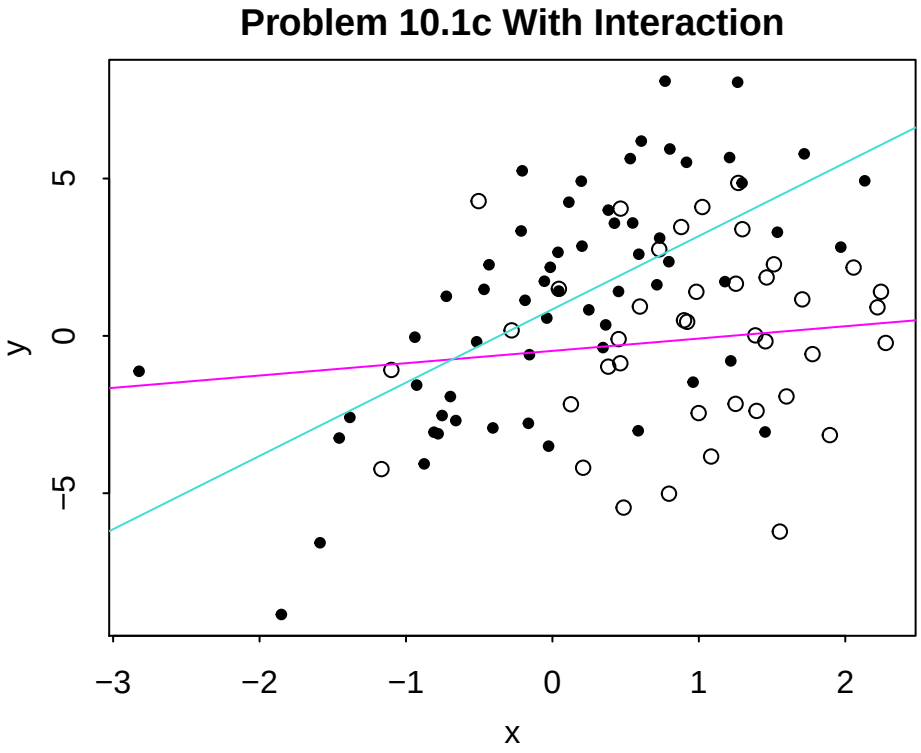
```
## (Intercept)           z           x         z:x
##   0.8438291  -1.3204556   2.3292545  -1.9378032
```

I'll talk about them more in-depth! I'll refer to the above coefficients as β_0, β_1, β_2, and β_3 respectively for this explanation.

- If z=0, the z and z:x terms would be eliminated from the equation. That means the z=0 group's line would have intercept 0.8 and slope 2.3.
- If z=1, the z term will modify the intercept, and the z:x term would modify the slope. Our new line would therefore have an intercept $\beta_0 + \beta_1$, or -0.5, and slope $\beta_2 + \beta_3$, or 0.4.

Again, we use these formulas and ab_line() to make our plots!! Again, let's let our z=1 line be magenta and our z=0 line be turquoise.

```
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(x,y,pch=shape, main = "Problem 10.1c With Interaction")
abline(b_hat[1] + b_hat[2], b_hat[3] + b_hat[4],col="magenta")
abline(b_hat[1], b_hat[3],col="turquoise")
```



We can see that our intercept now allows our lines to have completely different slopes, and capture two different trends altogether!

10.2

Regression with interactions: Here is the output from a fitted linear regression of outcome y on pre-treatment predictor x, treatment indicator z, and their interaction:

10.2a

Write the equation of the estimated regression line of y on x for the treatment group and the control group, and the equation of the estimated regression line of y on x for the control group.

First, I'll copy the table into R! I'll put it into a data frame manually.

```
Table = data.frame(Median=c(1.2,1.6,2.7,0.7),
                   MAD_SD=c(0.2,0.4,0.3,0.5))
row.names(Table) = c("(Intercept)","x","z","x:z")
Table
```

```
##             Median MAD_SD
## (Intercept)    1.2    0.2
## x              1.6    0.4
## z              2.7    0.3
## x:z            0.7    0.5
```

This table shows the output for our regression. To review: - Our intercept and our x are always present. These would be the intercept and slope of our "control". - z, the treatment indicator variable, is present or not present. It will affect the intercept for the treatment. The intercept for the treatment group is the model intercept + z. - x:z, our interaction, shows how our slope changes for the treatment group. It is calculated using x + x:z.

Therefore: Our Control group has intercept 1.2 and slope 1.6. This equation is $y_{control} = 1.2 + 1.6x$.

Our Treatment group has intercept 1.2 + 2.7 = 3.9, and slope 1.6 + 0.7 = 2.3. This equation is $y_{treatment} = 3.9 + 2.3x$.

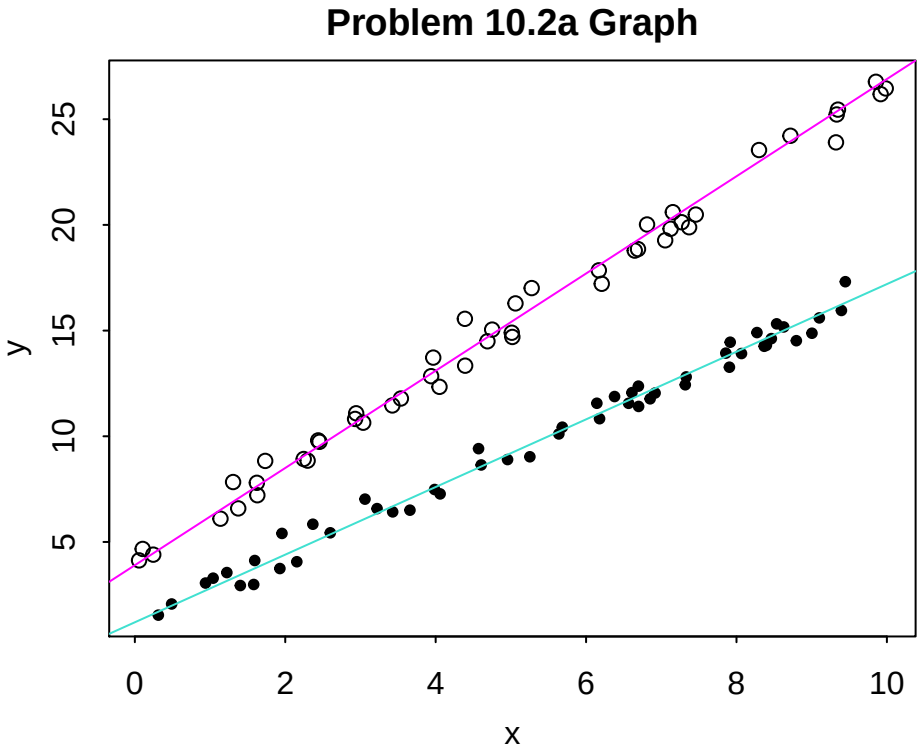
10.2b

Graph with pen on paper the two regression lines, assuming the values of x fall in the range (0, 10). On this graph also include a scatterplot of data (using open circles for treated units and dots for controls) that are consistent with the fitted model.

We can't sketch on an RMD file, but we can make graphs!

To do this, I'll follow a general process: I'll make 100 random x's using a `runif()`, with the range of 0 to 10, and 100 random z's using `rbinom()`. I'll set up an error term with a normal distribution, using variance 0.5 based off of the sigma coefficient listed in the table. I'll then calculate y using an ifelse statement. If z=0, I'll use the control equation. If z=1, I'll use the treatment equation. I'll then set shapes and make the plot!

```
bhat <- Table$Median
x = runif(100,0,10)
z = rbinom(100,1,0.5)
error = rnorm(100,0,0.5)
y = ifelse(z==0, bhat[1] + bhat[2]*x + error, bhat[1] + bhat[3] + (bhat[2]+bhat[4])*x + error )

shape = ifelse(z==0,20,1)
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(x,y,pch=shape,main="Problem 10.2a Graph")
abline(bhat[1],bhat[2],col="turquoise")
abline(bhat[1]+bhat[3],bhat[2]+bhat[4],col="magenta")
```



This displays what our plots will look like! For fun, let's see what happens when we run this new data through rstanarm!

```
data = data.frame(y,x,z)
M10.2 <- stan_glm(y~ x+ z + x:z,data=data,refresh=0)
print(M10.2)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      y ~ x + z + x:z
##  observations: 100
##  predictors:   4
## ------
##             Median MAD_SD
## (Intercept) 1.3    0.2
## x           1.6    0.0
## z           2.8    0.2
## x:z         0.7    0.0
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 0.5    0.0
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

Our output may not be the exact same, but is almost identical. This means that our data generating pattern worked!

10.5

Regression modeling and prediction: The folder KidIQ contains a subset of the children and mother data discussed earlier in the chapter. You have access to children's test scores at age 3, mother's education, and the mother's age at the time she gave birth for a sample of 400 children.

First, I'll download the kid IQ dataset using the process described in 7.6!

```
ghv_data_dir <- "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/"
kidiq <- read.csv (paste0(ghv_data_dir,"KidIQ/data/kidiq.csv"), header=T)
#kidiq <- read.csv("https://raw.githubusercontent.com/avehtari/ROS-Examples/master/KidIQ/data/kidiq.csv
#URL:
#"https://raw.githubusercontent.com/avehtari/ROS-Examples/master/KidIQ/data/kidiq.csv"
head(kidiq)
```

```
##   kid_score mom_hs    mom_iq mom_work mom_age
## 1        65      1 121.11753        4      27
## 2        98      1  89.36188        4      25
## 3        85      1 115.44316        4      27
## 4        83      1  99.44964        3      25
## 5       115      1  92.74571        4      27
## 6        98      0 107.90184        1      18
```

We can see we have a few things! - First, the kid's IQ score. - Second, an indicator variable showing if the mom graduated high school. - Third, the mom's IQ score. - Fourth, a categorical variable corresponding to the mother's working status. - Lastly, the mother's age.

We're in the MSSP program, so before I get to the rest of the problem, I'll explore the data a little bit first! This isn't required for HW, but is good practice to do (if you wanted to/had the time or energy! Again, not

required, but still encouraged if you can!)! Masanao talked about it before in class, and I'll just look at a few other things.

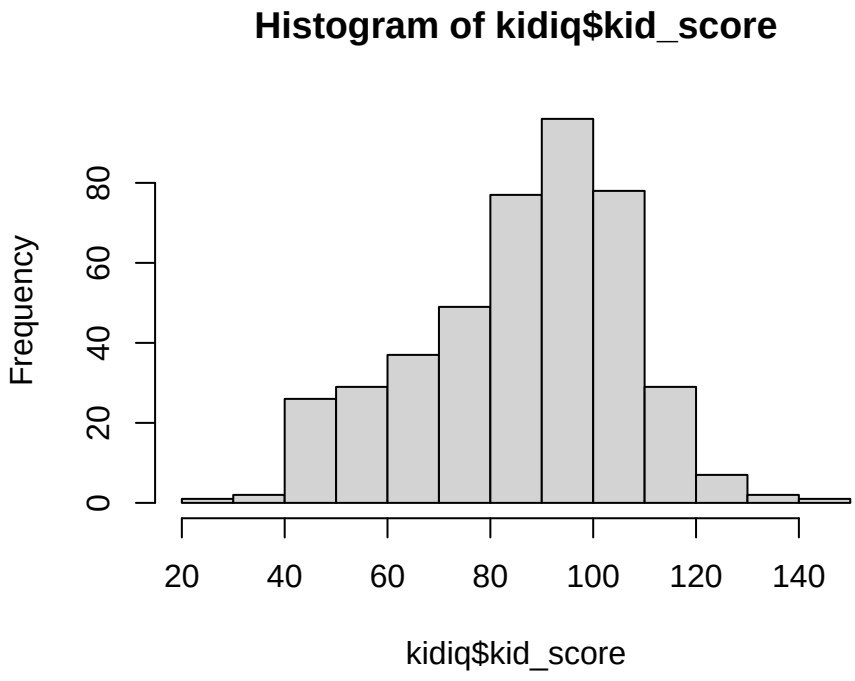
First, let's recall that we're trying to see how different aspects of the mother may impact the IQ of their kid. I'm going to list a few things that I think are important to consider for this question!

- Kid's age
- Kid's current education level
- Number of times kid's took an IQ test
- If the kids have a tutor
- Current living condition of the mother and the kids
- How much time the kids spend at school
- The "quality" of the school the kid goes to
- Kid's test-taking skills and overall grades

Basically, everything about the kid seems to be missing from the dataset other than the IQ score. If we're trying to answer questions about the IQ of children, considering information about the kids may be pretty important. Leaving out this information leads to two problems: - Without these here, any of the correlations and conclusions we make could be caused by these effects, and we'd have no way of knowing. A common example is a dataset that found that a lot of shark attacks happened on days where people bought lots of ice cream. However, this isn't because that eating ice cream leads to shark attacks, this is because a lot of people went to the beach on HOT days. So in fact, the correlation was the heat. In the IQ dataset, the correlation would probably be the kids. - The second thing is that this dataset ends up being intentionally misleading. Even though having reduced variables is very good for homework, we're only allowed to make comparisons to the mother. This kind of limitation can be very misguiding and deceptive, and can encourage incorrect conclusions. It's important not just to think about what is IN the dataset, but also what's NOT there.

Okay, now let's explore! I'm going to look over the Kid's IQ scores first!

```
hist(kidiq$kid_score)
```



This is a distribution of the kid's IQ scores visualized! We seem to have an average of 86.8. However, I'm personally not an expert about IQ scores, so I'm not quite sure what this means! Let's try to put it in context first.

Doing some quick googling, it says on Healthline.com (https://www.healthline.com/health/average-iq#

average-iq): "IQ tests are made to have an average score of 100. Psychologists revise the test every few years in order to maintain 100 as the average. Most people (about 68 percent) have an IQ between 85 and 115. Only a small fraction of people have a very low IQ (below 70) or a very high IQ (above 130)."

I don't know how valid this is, but for the sake of me not going down too much of a rabbit hole, I'm going to just use this as my reference for IQ scores. The thing I'll notice is that "a small fraction of people have a very low IQ (below 70)". Well, our dataset seems to have quite a few! Let's see how many using length and some brackets!

```
length(kidiq$kid_score[kidiq$kid_score<70])/length(kidiq$kid_score)
```

```
## [1] 0.2142857
```

```
# This uses length and division to see the proportion of kids who have scores less than 70. The numerat
```

So, according to this, ~21% of our database falls under a "small fraction of people"! I don't have the best understanding of IQ scores and won't try to make direct conclusions around this, but this, again, probably points to some kind of underlying demographic or something else in the dataset. You will all be talking about ethics, understanding biases, and considering all of this when making conclusions, but these kinds of things can be found pretty often. Overall, it's important to be aware of the context of what you're looking at, and be sure that you understand it when you start making conclusions.

I will also quickly note to those who have been reading this, it's very rare that people can carry out experiments where they can get all the information they want. However, it's crucial to acknowledge and embrace these limitations for both your analysis and the conclusions that you make, rather than ignore them.

There is definitely more that can be explored, but I'll leave it for here for now. This kind of exploration isn't required for homework unless specifically listed, so don't feel pressured to put time into this. However, if you do find yourself curious you can also probably explore and find out more about the database on your own as well! These kinds of data explorations can be surprisingly interesting.

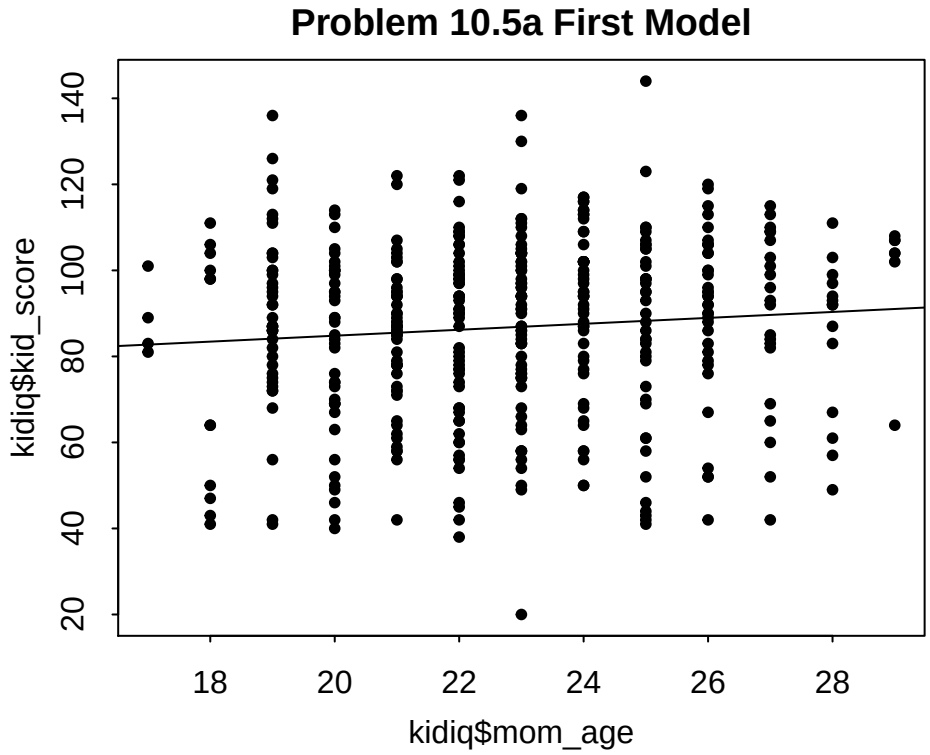
Through the questions, I'll talk about some more of my thoughts, but these are just my thoughts, and remember that you don't have to do all of this to get a good grade on these homeworks!

10.5a

Fit a regression of child test scores on mother' age, display the data and fitted model, check assumptions, and interpret the slope coefficient. Based on this analysis, when do you recommend mothers should give birth? What are you assuming in making this recommendation?

Alright, exploration aside, let's make the model how Gelman wants.

```
M1 <- stan_glm(kid_score~mom_age,data=kidiq,refresh=0)
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(kidiq$mom_age,kidiq$kid_score,pch=20,main="Problem 10.5a First Model")
abline(coef(M1))
```



This is going to look a little strange. This is because ages are discrete and not continuous, so we'll see the gaps between the points.

Let's look at our coefficients!

```
print(M1)
```

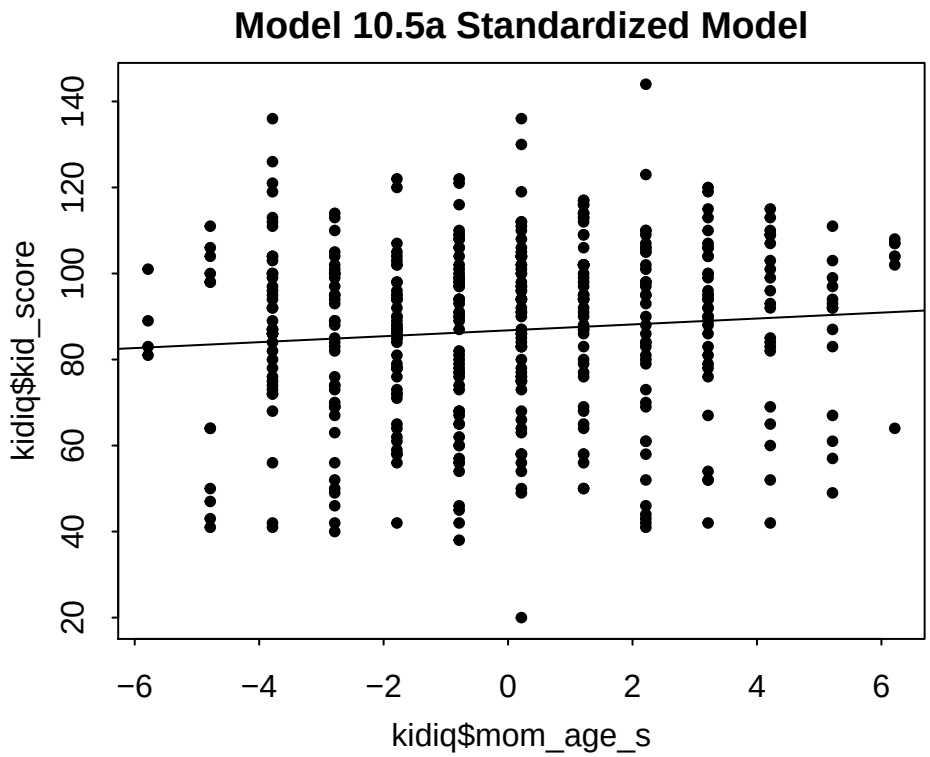
```
## stan_glm
##  family:       gaussian [identity]
##  formula:      kid_score ~ mom_age
##  observations: 434
##  predictors:   2
## ------
##             Median MAD_SD
## (Intercept) 71.0    8.2
## mom_age      0.7    0.4
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 20.4    0.7
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

So let's try to interpret this. The coefficient is with mom's age, meaning a mom 1 year older would be expected to have a kid with an IQ 0.7 higher. However, the error for it seems to be high, too, meaning that this may not be the most stable or interpretable estimate. Looking at our graphical display, it seems to be the case, too, since a lot of the data seems to fall both above and below the line.

We can also see that the intercept is weird. We talked about centering data in class, where we subtract the average age of the mothers in the dataset to standardize them, and I'm going to choose to apply it it here for

obvious reasons and rerun the model!

```
kidiq$mom_age_s <- kidiq$mom_age - mean(kidiq$mom_age)

M1 <- stan_glm(kid_score~mom_age_s,data=kidiq,refresh=0)
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(kidiq$mom_age_s,kidiq$kid_score,pch=20, main="Model 10.5a Standardized Model")
abline(coef(M1))
```



In our new plot, we can see that the x-axis has adjusted, and shows how many years above or below the mean the mom's age is!

Let's look at our coefficients!

```
print(M1)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      kid_score ~ mom_age_s
##  observations: 434
##  predictors:   2
## ------
##             Median MAD_SD
## (Intercept) 86.8    1.0
## mom_age_s    0.7    0.4
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 20.4    0.7
##
## ------
## * For help interpreting the printed output see ?print.stanreg
```

```
## * For info on the priors used see ?prior_summary.stanreg
```

Now, our intercept has increased to 86.8, which is the "Expected" IQ of the child of a mother from an average age (weird but I don't know how else to put it). We can also see that our slope hasn't changed at all! That's because we simply moved all of the points the same distance down the x axis. This affects the intercept, but not the slope!

I'll also mention here that the auxillary sigma is also pretty high. This is because the data has a very wide spread, and would naturally have a lot of variation.

Now comes the interpretation of Gelman's questions. Again, as I kind of pointed to in the exploration portion, I don't think that there's much that I personally would be confident in saying. For example, we do see a positive correlation with age, but it's entirely possible that older kids on average get higher IQ scores than younger kids!

One thing I will point out though is being careful of questions. If someone asks you something along the lines of "when do you recommend mothers should give birth?", the first thing I would recommend is running away. This dataset is based on 434 kids and their IQ tests, so something of this scale and magnitude is completely out of the scale of the data.

10.5b & c

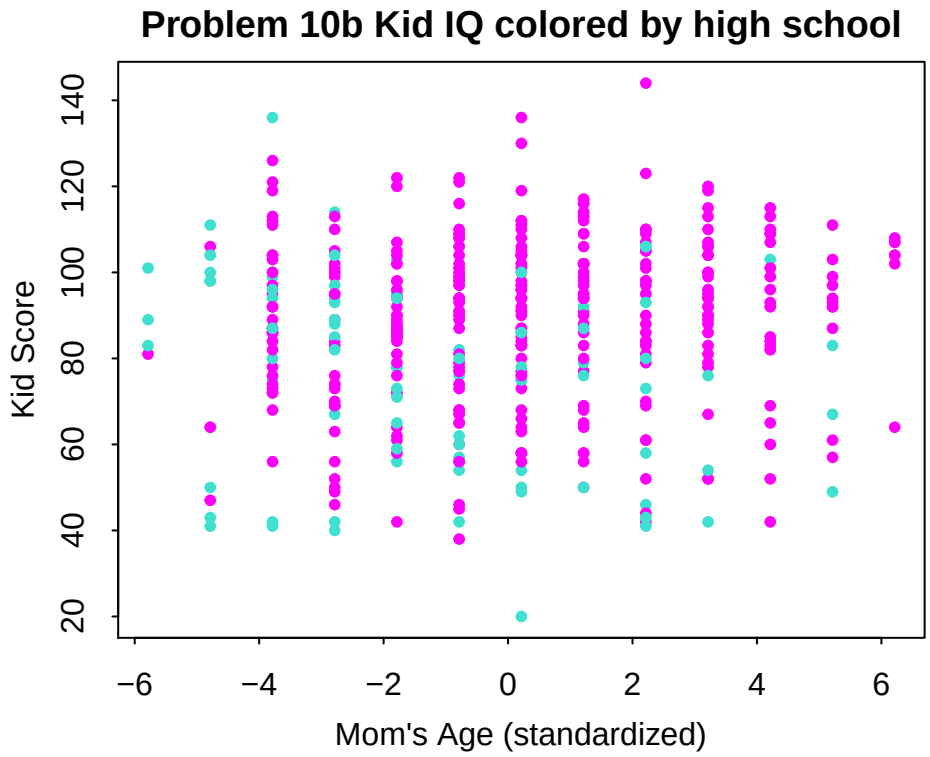
Repeat this for a regression that further includes mother's education, interpreting both slope coefficients in this model. Have your conclusions about the timing of birth changed?

Now create an indicator variable reflecting whether the mother has completed high school or not. Consider interactions between high school completion and mother's age. Also create a plot that shows the separate regression lines for each high school completion status group.

The dataset I found on github doesn't include the raw education data and includes the indicator variable, so I'll just be using that. I'll keep using the standardized age.

First, let's look at mothers' high school graduation status.

```
colors = ifelse(kidiq$mom_hs==1,"magenta","turquoise")
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(kidiq$mom_age_s,kidiq$kid_score,col=colors,pch=20, main = "Problem 10b Kid IQ colored by high scho
```



The high school graduate mothers are magenta, and the ones that did not are blue. The main thing I'll note for now is that there are many more mothers who seem to have graduated from high school than those that did not. We can still make our regression, but if we do choose to compare the groups, it's important to keep this in mind as well.

However, let's first make a model without interaction. I'll continue using the standardized age.

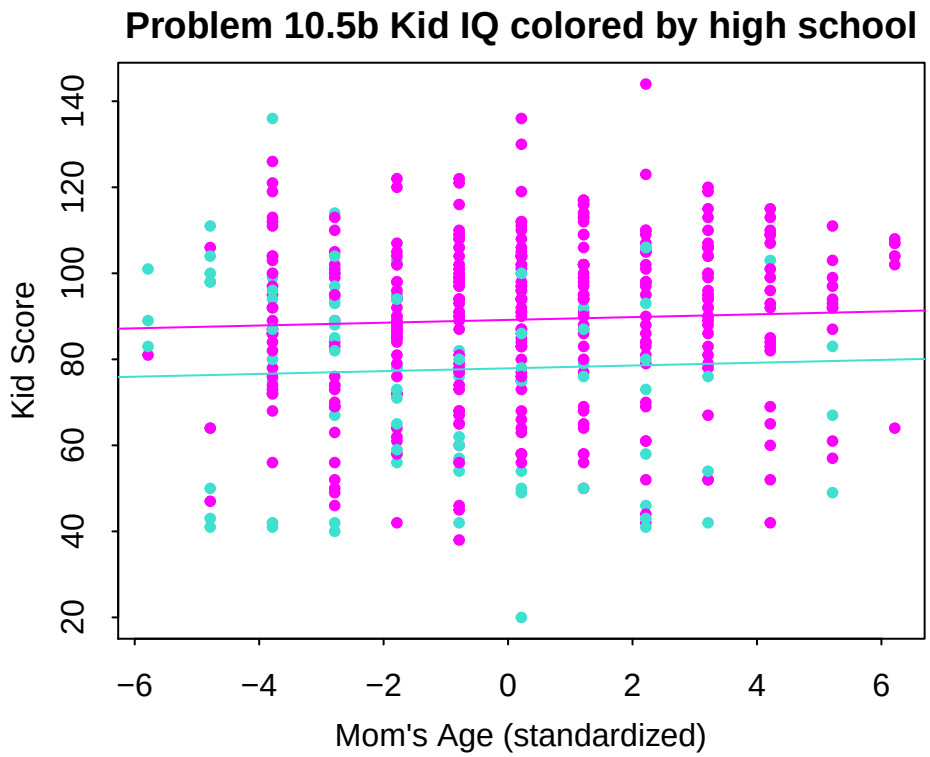
```
M2 <- stan_glm(kid_score~mom_age_s + mom_hs,data=kidiq,refresh=0)
print(M2)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      kid_score ~ mom_age_s + mom_hs
##  observations: 434
##  predictors:   3
## ------
##             Median MAD_SD
## (Intercept) 77.9    2.1
## mom_age_s    0.3    0.4
## mom_hs      11.3    2.4
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 19.9    0.6
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

First, we can see that our sigma is still pretty high, reflecting how variable the data set is. We can also see that the standardized age coefficient is smaller than it's error, which may signal that conclusions around it may be unreliable. Our model also seems to think that kid's associated with highschool graduate mothers

have an average IQ 11.3 higher! Wow! Let's visualize this!

```
b_hat <- coef(M2)
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(kidiq$mom_age_s,kidiq$kid_score,col=colors,pch=20, main = "Problem 10.5b Kid IQ colored by high sc
abline(b_hat[1] + b_hat[3], b_hat[2],col="magenta")
abline(b_hat[1], b_hat[2],col="turquoise")
```



We can see that the slope is almost flat now. And again, because the variation is so large, even though the lines are different, a lot of the data points still fall far from the line.

Let's try again with interactions!

```
M2 <- stan_glm(kid_score~mom_age_s + mom_hs + mom_age_s:mom_hs,data=kidiq,refresh=0)
print(M2)
```

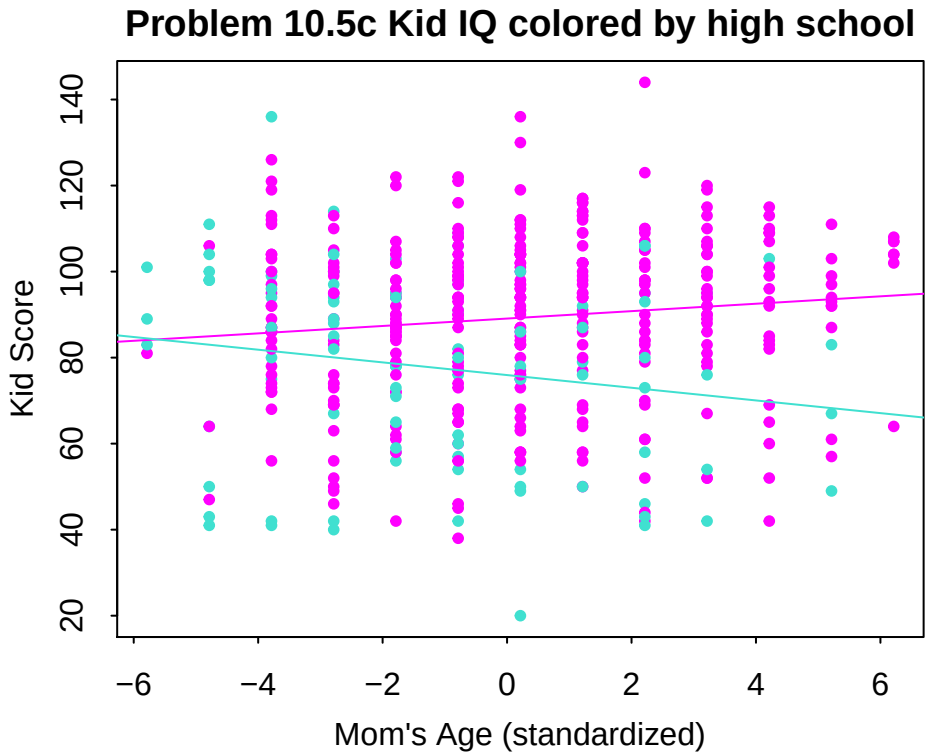
```
## stan_glm
##  family:       gaussian [identity]
##  formula:      kid_score ~ mom_age_s + mom_hs + mom_age_s:mom_hs
##  observations: 434
##  predictors:   4
## ------
##                  Median MAD_SD
## (Intercept)      75.9    2.2
## mom_age_s        -1.5    0.7
## mom_hs           13.1    2.5
## mom_age_s:mom_hs  2.3    0.9
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 19.7    0.7
##
```

```
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

Again, we now have the interaction term, and very different coefficients. Sigma is still high because the variance is still present.

Now, let's plot this!

```
b_hat <- coef(M2)
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(kidiq$mom_age_s,kidiq$kid_score,col=colors,pch=20, main = "Problem 10.5c Kid IQ colored by high sc
abline(b_hat[1] + b_hat[3], b_hat[2] + b_hat[4],col="magenta")
abline(b_hat[1], b_hat[2],col="turquoise")
```



This model now shows a positive slope for kids with highschool graduate mothers, and a negative slope for the other kids. Again, I think it's hard to make any conclusions out of this based on this data. There are so many more kids with high school graduate mothers, and again, we don't really know any of the features of the kids themselves!

Again, there is a lot of context wrapped around these kinds of topics. It's important to understand the context and what's missing, and not to jump to conclusions!

10.5d

Finally, fit a regression of child test scores on mother's age and education level for the first 200 children and use this model to predict test scores for the next 200. Graphically display comparisons of the predicted and actual scores for the final 200 children.

Now comes to the prediction part! This is basically a practice of testing the model. - First, we create a "training" set to build the model. - We then use it to predict values based on a "test" set of data, and use it to see how well the model does!

We'll use the first 200 kids as the training set and the next 200 for the test set. The last 34 kids won't be used. We'll then go straight to building our interaction model using our training data!

```
training = kidiq[1:200,]
test = kidiq[201:400,]
M3 <- stan_glm(kid_score ~ mom_age_s + mom_hs + mom_age_s:mom_hs, data = training,refresh=0)
print(M3)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      kid_score ~ mom_age_s + mom_hs + mom_age_s:mom_hs
##  observations: 200
##  predictors:   4
## ------
##                  Median MAD_SD
## (Intercept)      88.2    3.1
## mom_age_s        -1.3    1.0
## mom_hs            4.8    3.4
## mom_age_s:mom_hs  2.0    1.1
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 17.6    0.9
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

We can see that these coefficients are have changed from our past model That's because we're working with a different set of kids than before!

Now we go onto making our predictions. Luckily, because of the nature of rstanarm, this is pretty straightforward, and it makes all the background processing pay off!

We can use posterior_predict() to draw from our model's posterior distribution and predict multiple iterations of data. We'll discuss more of it in-depth in the course, but it basically produces a distribution as its output, which lets us draw multiple simulations. This also lets us calculate and display the uncertainty of our estimate.

Let's use posterior_predict() using our model and our test dataset. I'll also show the dimensions of the table that is produced.

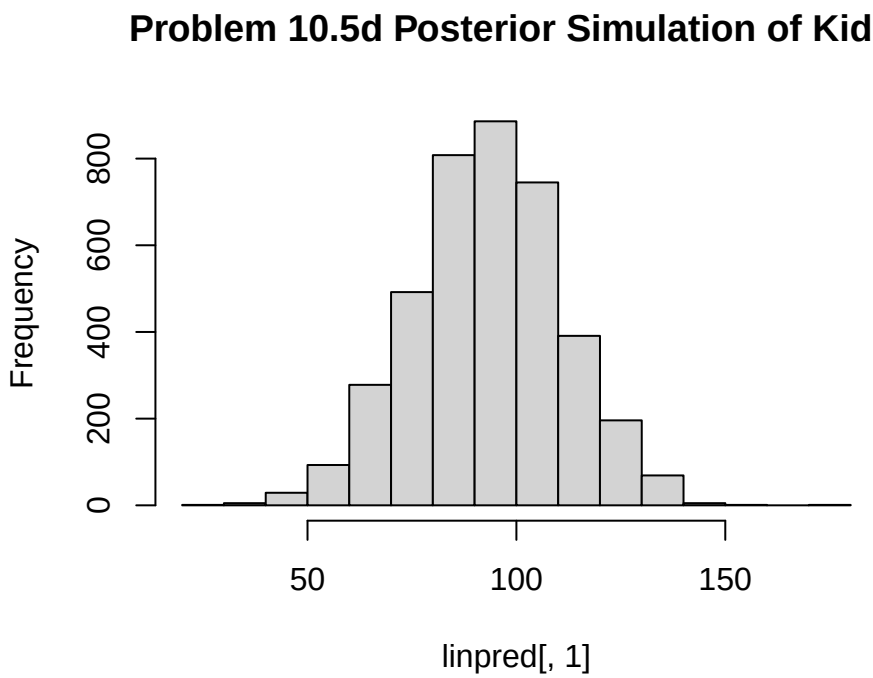
```
linpred <- posterior_predict(M3,test)
dim(linpred)
```

```
## [1] 4000  200
```

This posterior prediction is a 4000 x 200 dataset! What the heck?

What posterior_predict() does is looks at our new dataset, sees that there are 200 kids, and creates 4000 simulations for each child. Each row is a simulation, and each column represents each kid's data. Again, these are simulations, so let's look closely at the 4000 simulations for kid 1!

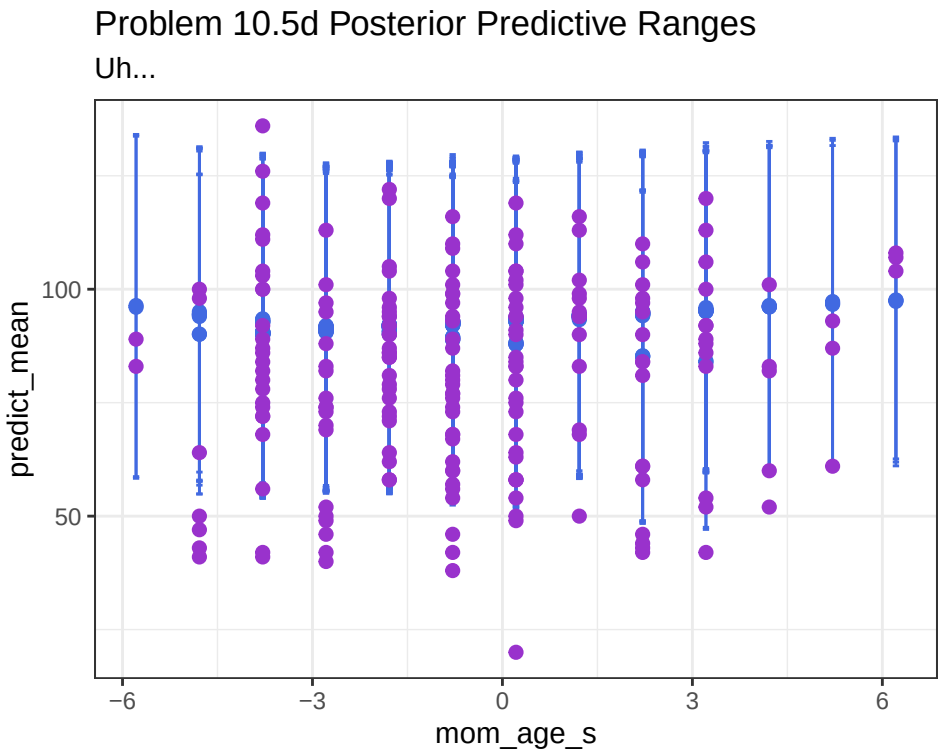
```
hist(linpred[,1],main="Problem 10.5d Posterior Simulation of Kid 1")
```



This histogram reflects the 4000 simulations for the first kid in the dataset. This distribution is a little skewed, but still seems close to normal!

Now comes to graphically comparing them. I'll use a few things: - the apply() function allows you to carry out a function on a data frame's row or column. In this case, I'll use it to get the mean and standard deviation for all simulations of each child. - I'll then use ggplot() to display it! I'll include each child's real points, the model's simulated points, and error bars to show the predictive range. - I'll also use a 95% predictive interval, which has a 95% chance of containing the real data point. This will basically follow the normal distribution pattern of being 2 standard deviations below and above the mean. Be warned, it's gonna be big.

```
test$predict_mean <- apply(linpred,2,mean)
test$predict_sd <- apply(linpred,2,sd)

ggplot(test) + aes(x=mom_age_s) +
  geom_errorbar(aes(ymin=predict_mean-2*predict_sd,ymax=predict_mean+2*predict_sd),color="royalblue",wi
  geom_point(aes(y=predict_mean),color="royalblue",size=2) +
    geom_point(aes(y=kid_score),color="darkorchid",size=2) +
  theme_bw() + labs(title="Problem 10.5d Posterior Predictive Ranges",subtitle="Uh...")
```



The blue points are the posterior predictions (with the bars being the predictive interval), and the purple points are the observed data. We can see two things: our model has very large uncertainty for each data point, and even with that, many observed values fall outside of our predictions. This, again, highlights that our model may have limitations.

10.6

Regression models with interactions: The folder Beauty contains data (use file beauty.csv) Beauty and teaching evaluations from Hamermesh and Parker (2005) on student evaluations of instructors' beauty and teaching quality for several courses at the University of Texas. The teaching evaluations were conducted at the end of the semester, and the beauty judgments were made later, by six students who had not attended the classes and were not aware of the course evaluations.

Alright, onto a new dataset! We're going to download this one directly from github.

```
#beauty <- read.csv("https://raw.githubusercontent.com/avehtari/ROS-Examples/master/Beauty/data/beauty.
ghv_data_dir <- "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/"
beauty <- read.csv (paste0(ghv_data_dir,"Beauty/data/beauty.csv"), header=T)
#URL is:
#https://raw.githubusercontent.com/avehtari/ROS-Examples/master/Beauty/data/beauty.csv
head(beauty)
```

```
##   eval     beauty female age minority nonenglish lower course_id
## 1  4.3  0.2015666      1  36        1          0     0         3
## 2  4.5 -0.8260813      0  59        0          0     0         0
## 3  3.7 -0.6603327      0  51        0          0     0         4
## 4  4.3 -0.7663125      1  40        0          0     0         2
## 5  4.4  1.4214450      1  31        0          0     0         0
## 6  4.2  0.5002196      0  62        0          0     0         0
```

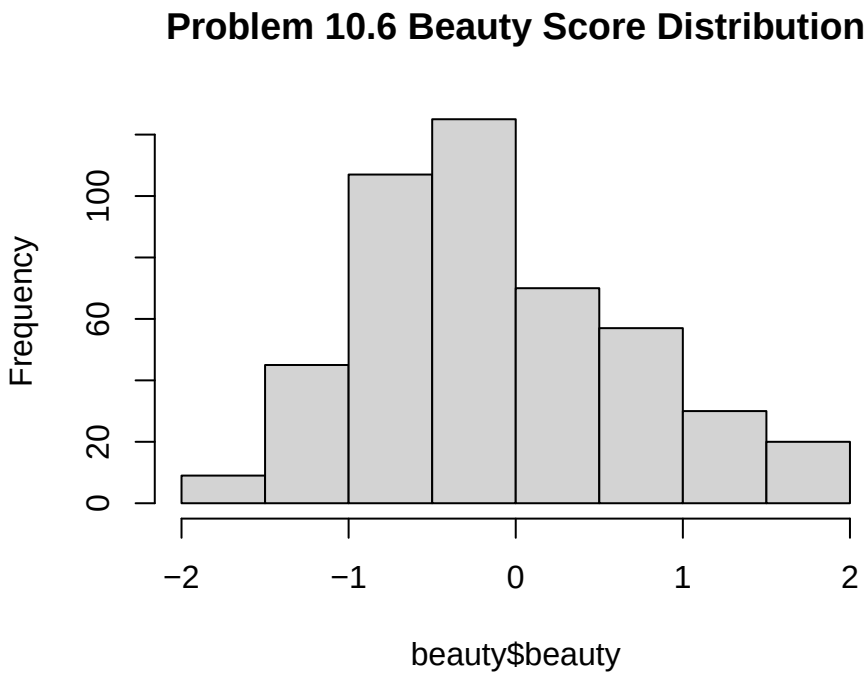
We can look at our data and look more closely: eval is the teacher's evaluation score, which seems to max at

5. female is a binary variable, which is 1 if the person is female. age is the teacher's age minority probably indicates if the teacher is from a minority. nonenglish indicates if the teacher is not a native english speaker. I don't know what lower is. course_id indicates which course they taught (I think).

For the sake of length of this, I won't do a thorough deep dive into this one. Again, a lot of information seems to be missing, so it's good to be careful of the conclusions and interpretations that you make, and you can feel free to explore it on your own!

I will, however, take a quick look at beauty! This is a variable that should indicate beauty, but I'm not quite sure how it works! Let's see how its data is distributed.

```
hist(beauty$beauty,main="Problem 10.6 Beauty Score Distribution")
```



This metric seems to be standardized at 0, with most of the populous being below 0. Most likely, this measurement had already been standardized. The mean of beauty is -0.1, which is close to 0, so it was closely standardized to the mean. However, we don't know two things: how this was calculated, and what a change in 1 means. I'd guess it would be 1 standard deviation of whatever the scale of the score is, but it's so difficult to say and makes it so much harder to interpret.

10.6a

Run a regression using beauty (the variable beauty) to predict course evaluations (eval), adjusting for various other predictors. Graph the data and fitted model, and explain the meaning of each of the coefficients along with the residual standard deviation. Plot the residuals versus fitted values.

Regardless, let's make our first model!

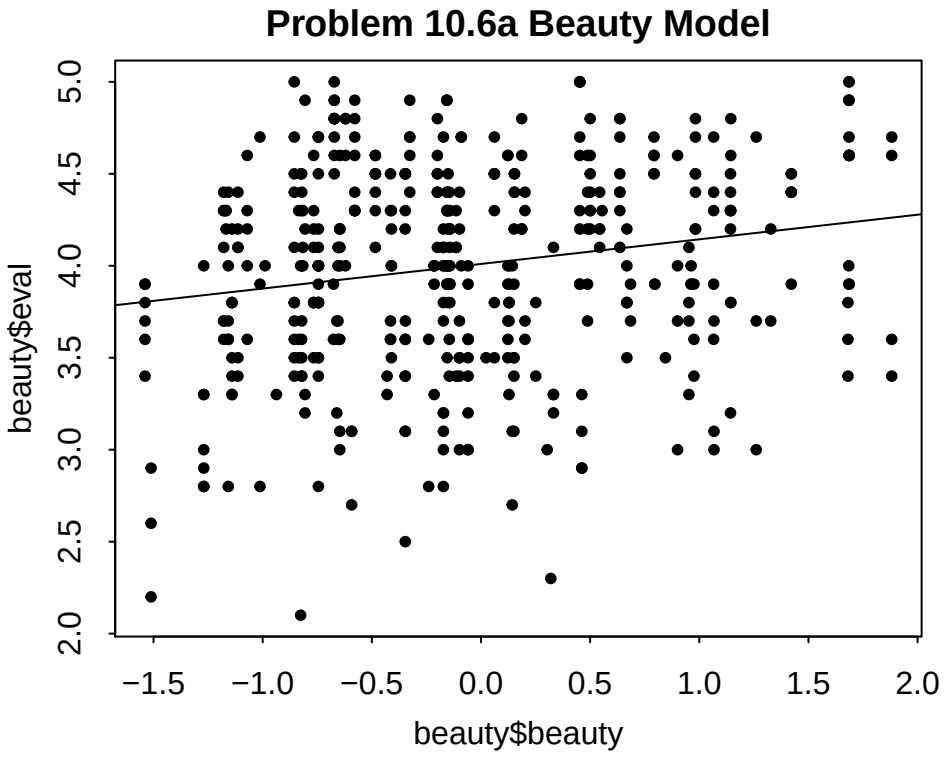
```
M10.6 <- stan_glm(eval ~ beauty,data=beauty,refresh=0)
print(M10.6)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      eval ~ beauty
##  observations: 463
##  predictors:   2
```

```
## ------
##             Median MAD_SD
## (Intercept) 4.0    0.0
## beauty      0.1    0.0
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 0.5    0.0
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

Again, the intercept in this case represent the average evaluation of a teacher with a beauty of 0 - or the average evaluation of the "average" teacher. The beauty of 0.1 indicates that teachers with higher beauty are expected to have higher evaluations, but only slightly. Let's display this with the data!

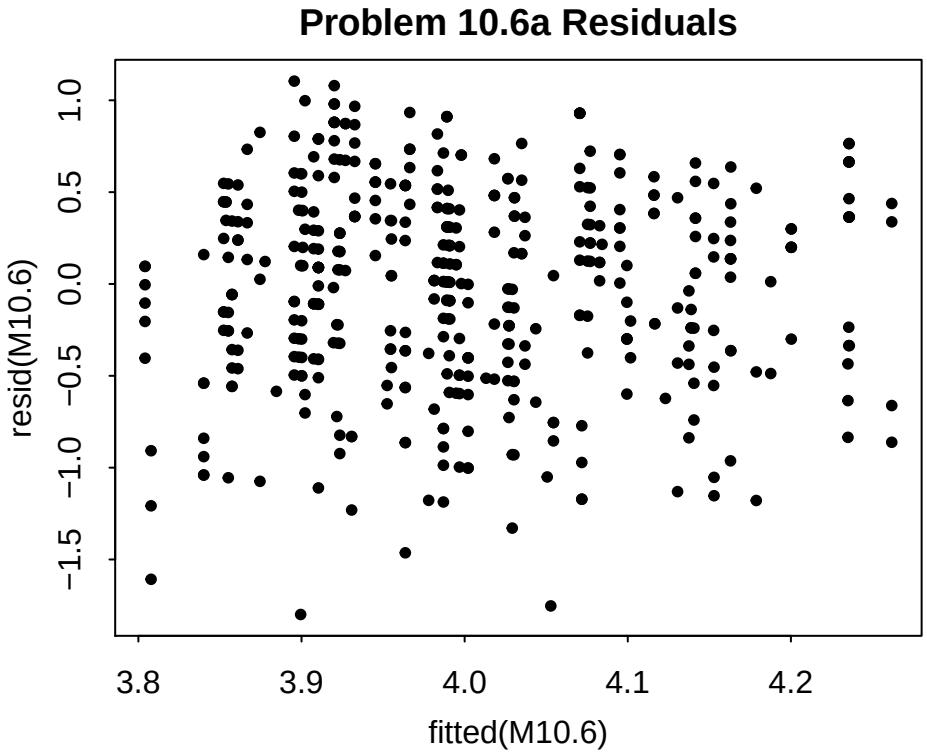
```
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(beauty$beauty,beauty$eval,main="Problem 10.6a Beauty Model",pch=20)
abline(coef(M10.6))
```



Our data seems to have a wide distribution around our line, and the slope doesn't seems relatively large given the scale, but still small.

Now, let's look at our residuals! The way residuals work is that they look at how off your plots are. Plotting your fitted values together with your residual can highlight on underlying patterns that your model isn't capturing, like nonlinearity, for example. Let's look here! You can make residuals using plot, fitted(), and resid()!

```
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(fitted(M10.6),resid(M10.6),pch=20,main="Problem 10.6a Residuals")
```



Ideally, the residuals should be a cloud of points with no obvious patterns, and this seems to fit the bill!

10.6b

Fit some other models, including beauty and also other predictors. Consider at least one model with interactions. For each model, explain the meaning of each of its estimated coefficients.

There's a lot that you can do for additional models. Because they're the first two predictor variables, what I'll do here is make a model using the beauty, female, and an interaction between the two!

```
M10.6b <- stan_glm(eval ~ beauty + female + beauty:female,data=beauty,refresh=0)
print(M10.6b)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      eval ~ beauty + female + beauty:female
##  observations: 463
##  predictors:   4
## ------
##                Median MAD_SD
## (Intercept)     4.1    0.0
## beauty          0.2    0.0
## female         -0.2    0.1
## beauty:female -0.1    0.1
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 0.5    0.0
##
## ------
```

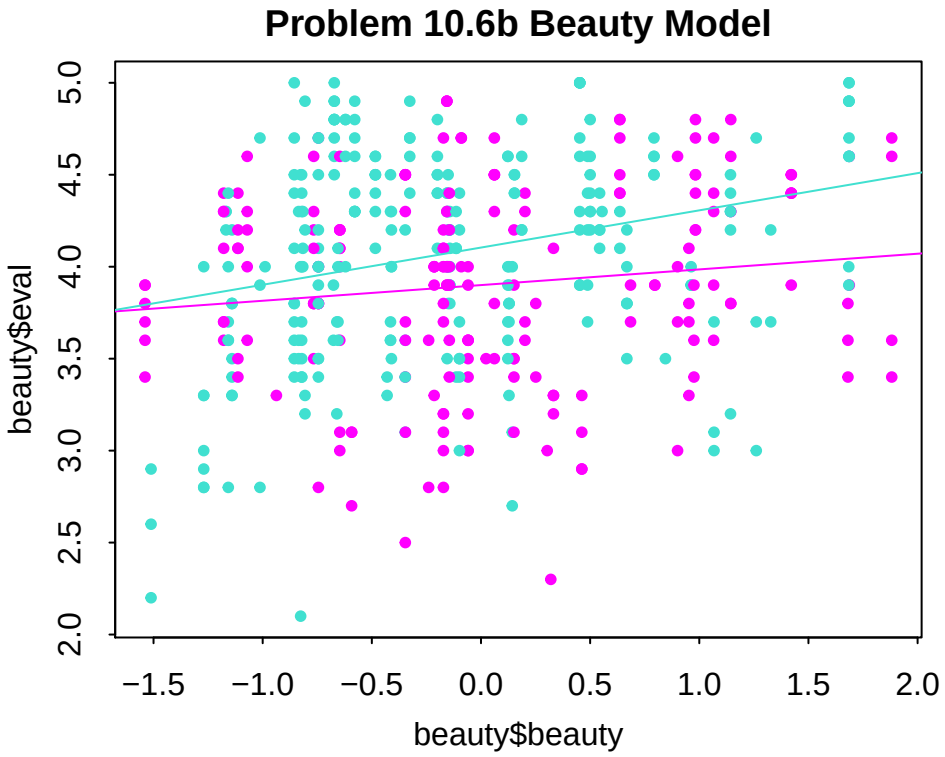
```
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

Let's look at our coefficients. Effectively, if the professor is male with average beauty, they would be expected to have an evaluation of 4.1. The slope coefficient of 0.2 means that professors with 1 more point in beauty score seem to have evaluations 0.2 points higher.

In comparison, female professors with average beauty are expected to have an average evaluation 0.2 lower than males, at 3.9. Their slope is also lowered to 0.1, meaning it has a lower rate of change. The standard deviation for this coefficient estimation is high, though, so the conclusions around this may be unstable.

Let's visualize it! Let's make the magenta females and turquoise non-females.

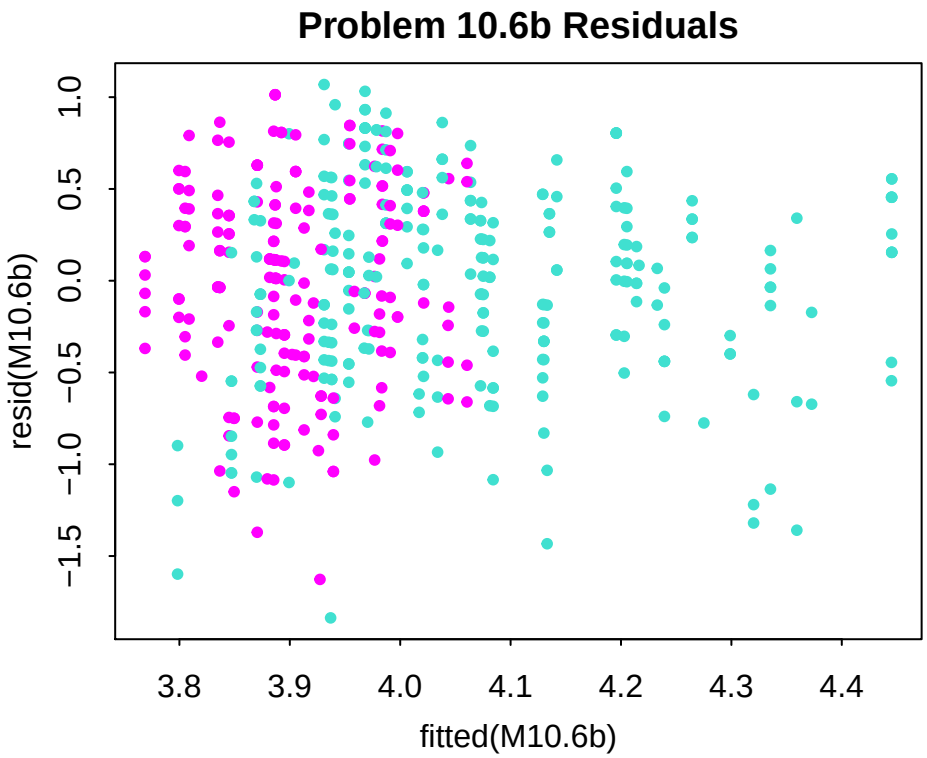
```
b_hat <- coef(M10.6b)
colors = ifelse(beauty$female==1,"magenta","turquoise")
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(beauty$beauty,beauty$eval,main="Problem 10.6b Beauty Model",pch=20,col=colors)
abline(b_hat[1] + b_hat[3],b_hat[2]+b_hat[4],col="magenta")
abline(b_hat[1],b_hat[2],col="turquoise")
```



So, the model seems to be suggesting that on average and given equal beauty score, non-females get better evaluations than females. Again, there are a lot of reasons that this could be this way that need to account for university envornment and the way evaluations are given out, as well as course difficulty, but we don't have that information, so I'm personally not comfortable making conclusions!

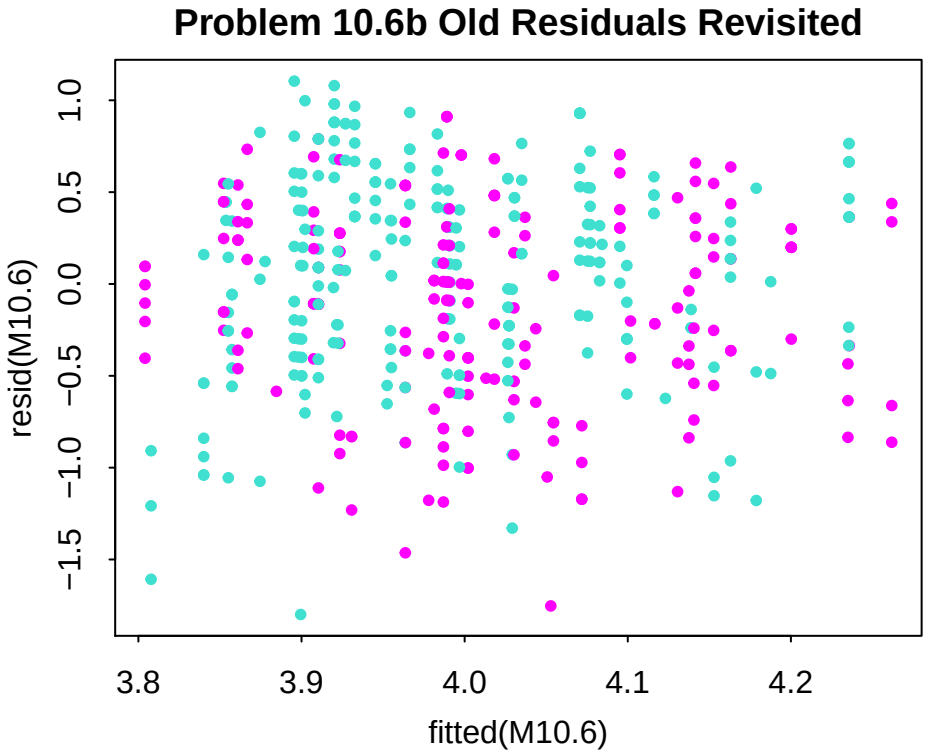
Lastly, let's look over our model's residuals. I'll color the points by the female indicator!

```
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(fitted(M10.6b),resid(M10.6b),pch=20,col=colors,main="Problem 10.6b Residuals")
```



We can see that our residuals are in a cloud. However, we can see that the colors are pretty separated! This is relatively weird. Was this present in the last model we did? Let's check!

```
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01)
plot(fitted(M10.6),resid(M10.6),pch=20,col=colors,main="Problem 10.6b Old Residuals Revisited")
```



It actually wasn't! It seems that including the interaction actually separated the groups in the residuals!

I'm not sure if one is necessarily more appropriate than the other, but these are things that you need to keep track of whenever you're incorporating interactions and other factors into the models. Looking at residuals can help highlight weird things that your model may be doing, and can help guide you on how to fix them!

10.7

Predictive simulation for linear regression: Take one of the models from the previous exercise.

10.7a

Instructor A is a 50-year-old woman who is a native English speaker and has a beauty score of -1. Instructor B is a 60-year-old man who is a native English speaker and has a beauty score of -0.5. Simulate 1000 random draws of the course evaluation rating of these two instructors. In your simulation, use posterior_predict to account for the uncertainty in the regression parameters as well as predictive uncertainty.

First, let's create the data for our instructors. We can use simple one-row data frames! In data frames, you can set column names and the values that are in them. If you use a single value, it'll produce one row. Here, I'll manually put in the column names to match the information from the book!

```
instA <- data.frame(beauty=-1,female=1,age=50,minority=0,nonenglish=0)
instB <- data.frame(beauty=-0.5,female=0,age=60,minority=0,nonenglish=0)
```

Note: you can go about this by using one data frame, but I wanted to showcase this!

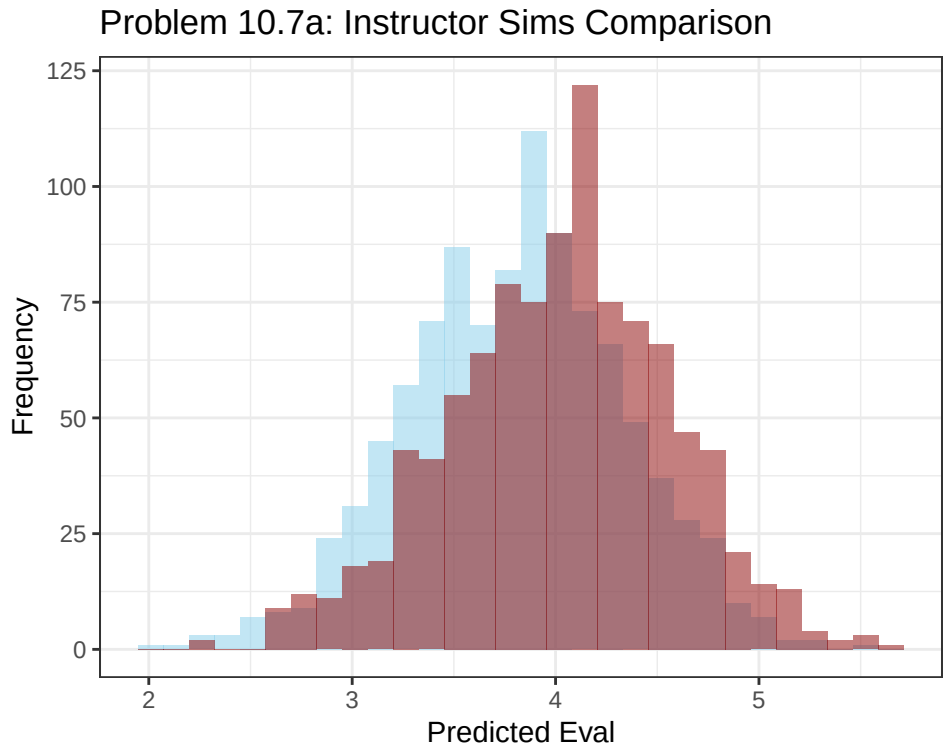
Now, let's use posterior_predict() using our the model from 10.6b. We can use the "draws" argument to draw 1000 simulations instead.

```
simA <- posterior_predict(M10.6b,newdata=instA,draws=1000)
simB <- posterior_predict(M10.6b,newdata=instB,draws=1000)
```

Let's visualize our findings!

```
ggplot() + geom_histogram(aes(simA[,1]),fill="skyblue",alpha=0.5) +geom_histogram(aes(simB[,1]),fill="r
```

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



Instructor A is in blue, and Instructor B is red. We can see that these simulations have wide spreads in their distribution, even predicting scores higher than any of the professors in our dataset! We can also see that even though blue seems to be slightly smaller, most of it is overlap.

10.7b

Make a histogram of the difference between the course evaluations for A and B. What is the probability that A will have a higher evaluation?

Now comes to comparing Instructor A and B. Again, the output of an `rstanarm()` model's predictions will be a probability distribution. Here, we're comparing two different distributions for two different professors, and can make predictions using probability!

We can solve this problem two ways: one is with math, and the other is with simulation.

10.7b: Probability

Based off of the above distributions, we can guess that these simulations seem to follow a normal distribution. We can use `apply()` to calculate the mean and standard deviations of each one, which I'll show below.

```
probA <- c(apply(simA,2,mean),apply(simA,2,sd))
probB <- c(apply(simB,2,mean),apply(simB,2,sd))
```

Next, we want to see if A > B. Funnily enough, the comparison A > B is comparing two probabilities. This may seem weird, but we can rewrite it as A - B > 0. This means we're looking at a new distribution, A - B, and how often it's more than 0!

So first, let's look at A - B. This is the sum of two normal distributions, A - B, which will have two properties: its mean will be the sum of its means, and its variance will be the sum of its variances (these are traits for the sum of any set of normal distributions). We can calculate those in code!

```
probC <- c(probA[1]-probB[1],sqrt(probA[2]^2+probB[2]^2))
```

Lastly, we want to figure out how often A - B > 0. Using R code, we can look at the different values. The closest would be pnorm(), which gives the probability that the distribution would produce a value equal to or lower to a set value. That means it can show us when A - B =< 0. However, we can find the probability we're looking for by subtracting this probability from 1, or by using 1 - pnorm().

```
1 - pnorm(0,probC[1],probC[2])
```

```
## [1] 0.3891175
```

If our assumptions about the simulations being normal hold, this should be the probability A > B!

10.7b: Simulation

We can use the following process to simulate the probability. - Pick random simulated values from simulations A and B separately. - See how many times one is bigger than the other. - Use mean() to see the proportion of times it was 1.

We can do this using sample(). We draw from each distribution a large number of times (I'm doing 10000!), and set replace=T to allow it to draw the same value twice.

```
a <- sample(simA,10000,replace=T)
b <- sample(simB,10000,replace=T)


mean(a>b)
```

```
## [1] 0.3884
```

These values are very close, indicating that it's drawing from a similar process! It will never quite be exact because, rather than drawing from a known distribution, this is drawing from samples of the simulation, and will have some sampling uncertainty.

One note, these are random for both the simulations and the output for this, so if your numbers are slightly different from mine, that's okay! The important thing is the code.

10.8

How many simulation draws: Take the model from Exercise 10.6 that predicts course evaluations from beauty and other predictors.

10.8a

Display and discuss the fitted model. Focus on the estimate and standard error for the coefficient of beauty.

I did this pretty thoroughly back in 10.6b. I'll print the output but won't talk about it here!

```
print(M10.6b)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      eval ~ beauty + female + beauty:female
##  observations: 463
##  predictors:   4
## ------
##                Median MAD_SD
## (Intercept)     4.1    0.0
## beauty          0.2    0.0
## female         -0.2    0.1
## beauty:female -0.1    0.1
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 0.5    0.0
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

10.8b

Compute the median and mad sd of the posterior simulations of the coefficient of beauty, and check that these are the same as the output from printing the fit.

The as.matrix() function can let us look at the posterior simulations of the model itself, similar to posterior_predict(), but for the values you used to fit it. We can use this and apply() to find the MEDIAN and MAD_SD values, using the median and mad functions respectively!

```
sims <-as.matrix(M10.6b)

MEDIAN <- apply(sims,2,median)
MAD_SD <- apply(sims,2,mad)
print(cbind(round(MEDIAN,1),round(MAD_SD,1)))
```

```
##                [,1] [,2]
## (Intercept)     4.1  0.0
## beauty          0.2  0.0
## female         -0.2  0.1
## beauty:female -0.1  0.1
## sigma           0.5  0.0
```

These seem identical or very close the values from our model!

10.8c

Fit again, this time setting iter = 1000 in your stan_glm call. Do this a few times in order to get a sense of the simulation variability.

We can use the "iter" argument to change the number of iterations our model runs! To look at the variability, I'm going to use a for loop! A for loop basically allows you to run the same value multiple times, and can be useful if you're repeating a process. It's harder to explain in text, but the basics are: - you set i to go from a range of numbers. - If you want to store values, you need an empty object with spaces that can be filled each

time you run it. In my case, I need to make spaces for 5 lists. - You'll also see double brackets in this one. That's because I need to specify that I'm storing the output in a list slot. It's weird.

```
M10.8c
```

```
## [[1]]
##   (Intercept)        beauty        female beauty:female
##     4.1013728     0.1979259    -0.2033161    -0.1096042
##
## [[2]]
##   (Intercept)        beauty        female beauty:female
##     4.1052462     0.1998731    -0.2031604    -0.1101934
##
## [[3]]
##   (Intercept)        beauty        female beauty:female
##     4.1022647     0.2007651    -0.2060161    -0.1123873
##
## [[4]]
##   (Intercept)        beauty        female beauty:female
##     4.1049494     0.1952814    -0.2058892    -0.1091996
##
## [[5]]
##   (Intercept)        beauty        female beauty:female
##     4.1020808     0.1983558    -0.2044882    -0.1105216
```

The other advantage for the for loop is that I get to print all my outputs at once!

Here we can see that there is some variation, but all of it is small.

10.8d

Repeat the previous step, setting iter = 100 and then iter = 10.

Let's do it again with 100 iterations!

```
M10.8d1
```

```
## [[1]]
##   (Intercept)        beauty        female beauty:female
##     4.1058074     0.2017964    -0.2068400    -0.1201664
##
## [[2]]
##   (Intercept)        beauty        female beauty:female
##     4.1058561     0.1982433    -0.2043001    -0.1163667
##
## [[3]]
##   (Intercept)        beauty        female beauty:female
##     4.1115132     0.2059329    -0.2102636    -0.1158283
##
## [[4]]
##   (Intercept)        beauty        female beauty:female
##     4.1038665     0.1986429    -0.2070642    -0.1166799
##
## [[5]]
##   (Intercept)        beauty        female beauty:female
##     4.1002431     0.1976654    -0.1964406    -0.1088490
```

Again, it seems pretty consistent overall, but definitely a little less than before!

What about with 10 iterations?

```
M10.8d2
```

```
## [[1]]
##   (Intercept)        beauty        female beauty:female
##     1.1626085     0.1037335    -1.7251221     1.6604912
##
## [[2]]
##   (Intercept)        beauty        female beauty:female
##    -0.3568315     1.5397270     2.6754747    -0.4511328
##
## [[3]]
##   (Intercept)        beauty        female beauty:female
##     1.8666285     0.1265665    -2.1668890    -0.9431330
##
## [[4]]
##   (Intercept)        beauty        female beauty:female
##     0.7986649     0.4476768     0.4473728    -1.0674583
##
## [[5]]
##   (Intercept)        beauty        female beauty:female
##     2.2568864     0.6453408    -4.1583308     0.5799900
```

We'll see two things: Stan yells when it's unhappy. A LOT. Especially if you use loops. The coefficients vary a LOT with 10 iterations.

The reasons for this are linked. The way STAN works is by basically drawing estimations for the variables multiple times using an algorithm, and after a while, the algorithm converges onto an estimate. In the first steps, usually it varies a lot before it converges, and that's what happens when you make the number of iterations too small.

10.8e

How many simulations were needed to give a good approximation to the mean and standard error for the coefficient of beauty?

However, picking a number of iterations is really weird. For simpler models, you can probably get away with doing fewer. However, more complex models may require more. There isn't a direct "rule of thumb" for this. Generally, the default or 1000 iterations can be good, but just watch for when rstan yells at you!

In this case, I probably would want to use 1000 or more, but usually if you get warnings from STAN, that means you don't have enough or something else is wrong!