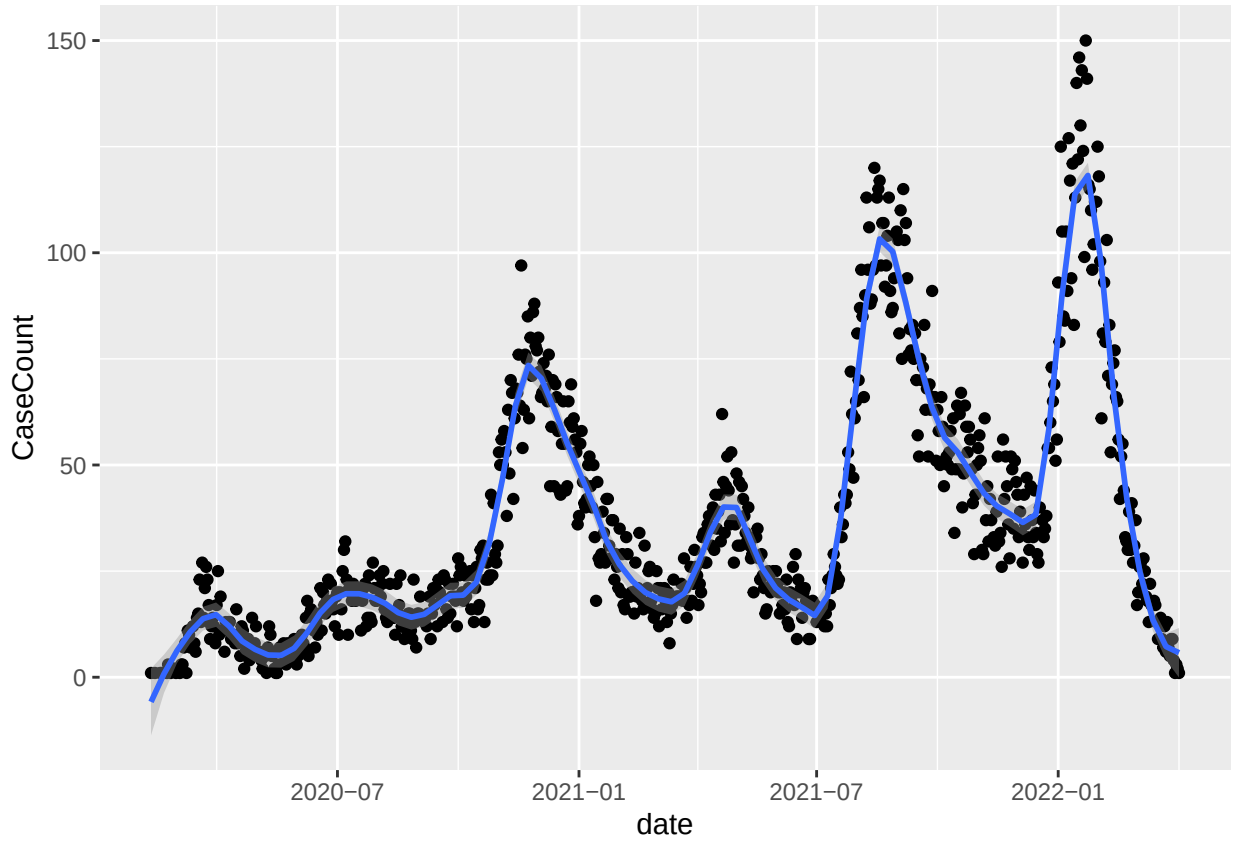
Wednesday Class 4/06/2022

Ian McConachie

More COVID data

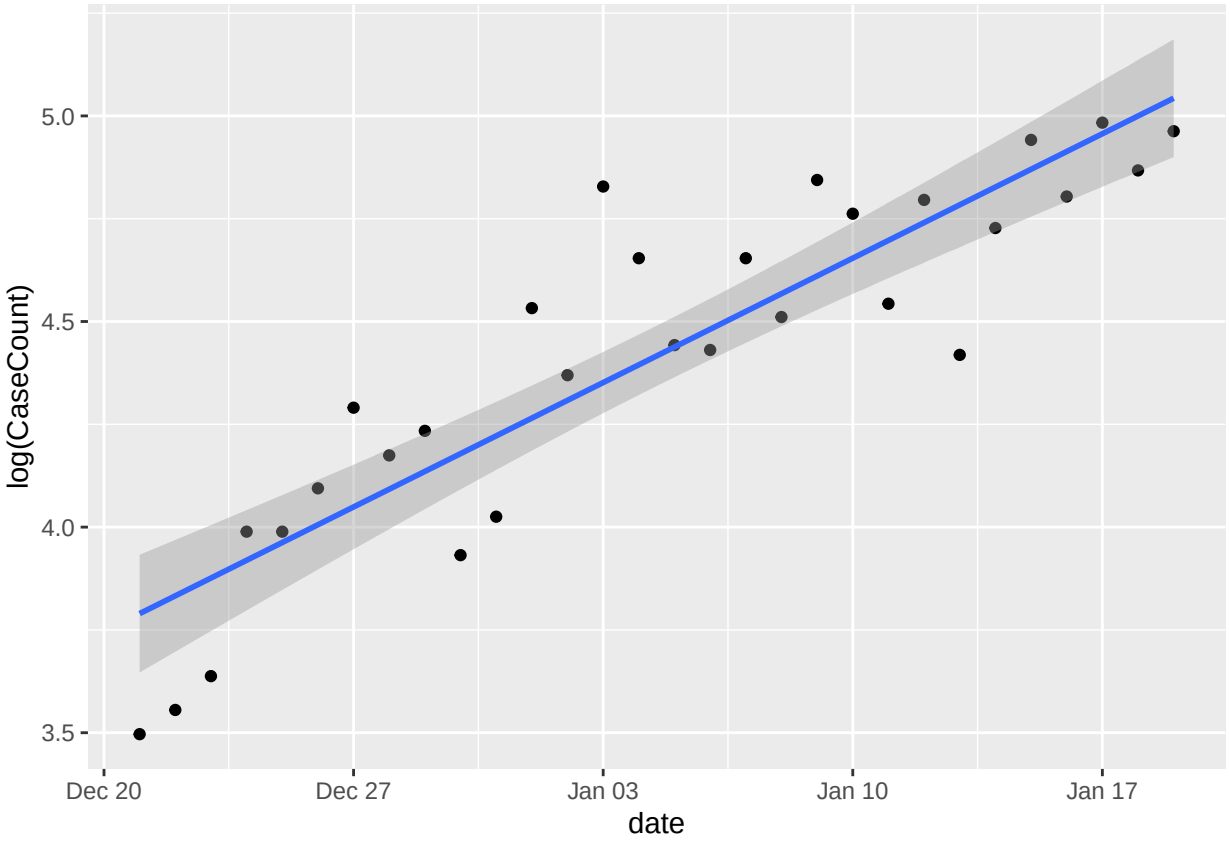
Data for Oregon hospitalizations can be found at http://pages.uoregon.edu/dlevin/DATA/oregon.txt

```
library(dplyr)
orcvd <- read.table("http://pages.uoregon.edu/dlevin/DATA/oregon.txt",sep="\t",header=TRUE)
orcvd <- orcvd %>% filter(HospitalizationStatus=="Y")
orcvd$date <- as.Date(orcvd$Onset,format = "%m/%d/%y")
ggplot(orcvd, aes(date,CaseCount)) + geom_point() + geom_smooth(method="loess",span=0.1)
```



Focus on the data from the Omicron wave:

```
ggplot(subset(orcvd,(date>"2021-12-20")&(date<"2022-01-20")), aes(date,log(CaseCount))) + geom_point() +
```



Using the data for hospitalizations in first 31 days of the Delta wave in Oregon, formulate a prior distribution for the rate of growth of the Omicron wave.

Fit a Bayesian model and give a 95% uncertainty interval for the growth rate of Omicron based on the estimate. How different is the estimate if a non-informative prior is used? How different is the estimate if a weakly-informative prior (e.g., the default prior) is used?

Use `posterior_linpred` to estimate the expected number of hospitalizations 10 days into the Omicron wave, and give a 95% uncertainty interval around that estimate.

Use `posterior_predict` to estimate the number of hospitalizations 10 days into the Omicron wave, and explain the difference between this estimate and the last one

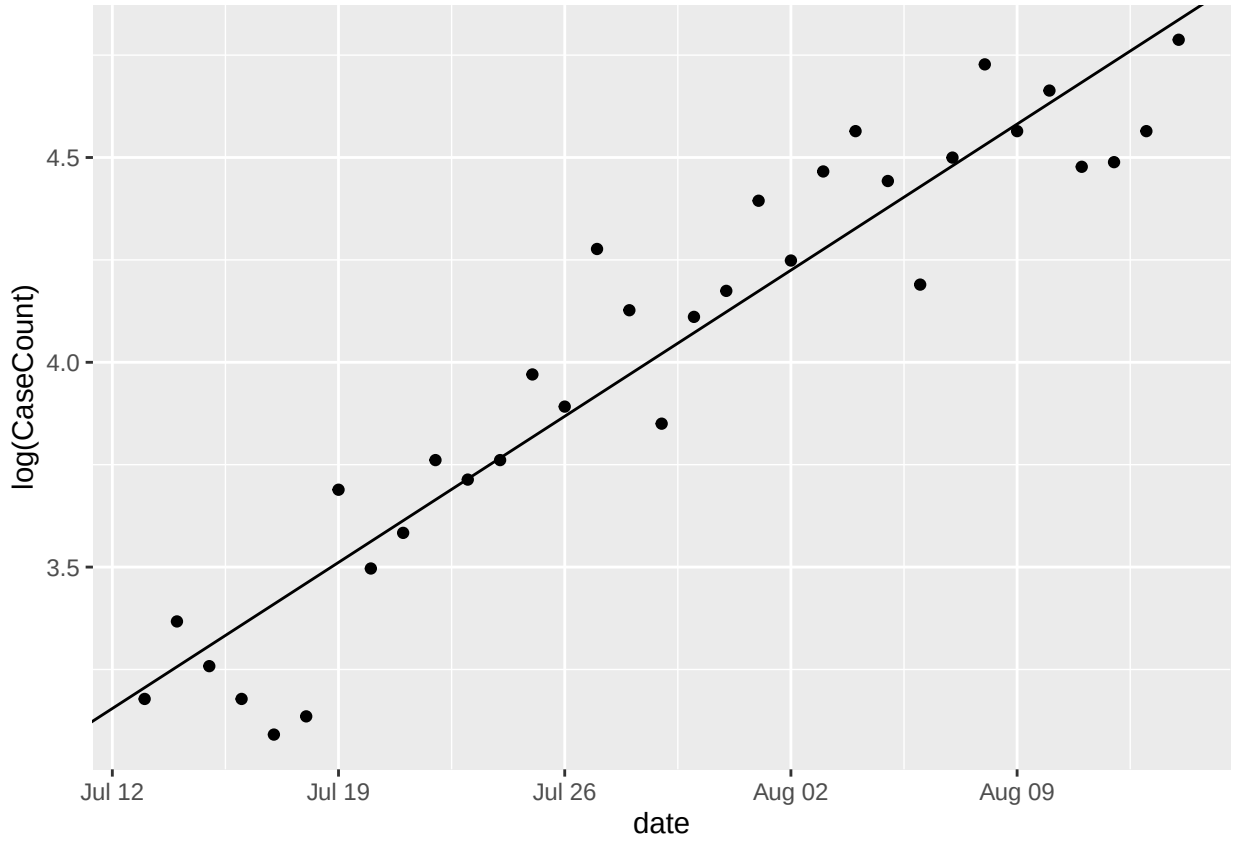
SOLUTION

First, let us take a subset of the hospitalization data given to be representative of the first 31 days of the Omicron variant wave in Oregon.

```
library(rstanarm)
omwave <- subset(orcvd,(date>"2021-12-20")&(date<"2022-01-20"))
```

Now, here is some code to take a subset of the data that is representative of the hospitalizations in the first 31(?) days of the Delta wave COVID epidemic in Oregon. The code then plots the data as a scatter plot and makes a linear model for it with the classical statistical methods used in the lm function.

```
deltawave <- subset(orcvd,(date>"2021-7-12")&(date<"2021-08-15"))
fdelta <- lm(log(CaseCount)~date, data=deltawave)
fDel = coef(fdelta)
ggplot(deltawave, aes(y=log(CaseCount), x=date)) + geom_point() + geom_abline(slope = fDel[2], intercep
```



Now we can use the linear model we fitted to the data above to make a reasonable prior for modeling the beginning of the Omicron wave using Bayesian statistics.

Let us assume that the exponential growth rate of hospitalizations in Oregon at the beginning of a new variant wave is taken from a normal distribution. We can find low and high end estimates for this rate using some assumptions (that come with uncertainty) about its relation to the exponential growth rate seen with the Delta wave.

We can take the high end estimate to be two standard errors (of the Delta rate) above the Delta rate and multiplied by 4 and then 0.75. Four is a high estimate for the factor at which the Omicron variant is more infectious than Delta, and 0.75 is a high estimate for the rate of hospitalization per case in the Omicron wave compared to the Delta wave. The low estimate can be set as two standard errors below the least-squares rate for the Delta wave, multiplied by low estimates for the statistics previously mentioned (those estimates are 3 and 0.25 respectively).

Using the low and high estimates for the rate of Omicron's exponential growth, we can estimate the mean of the normal distribution we're using to describe the rate's prior by finding the average of these two values.

The standard deviation of the prior can be estimated by setting our high and low estimates to be 2 standard deviations from the mean on opposite sides (because we are assuming the majority of possible values lie between them). To calculate this estimate, we can simply subtract the estimated mean from the high end estimate and then divide by 2 to get the value for standard deviation.

```r
# Some code to find a reasonable prior for the omicron rate
r = summary(fdelta)$coefficients[2,1]
r_se = summary(fdelta)$coefficients[2,2]
high_end = (r + 2*r_se)*4*0.75
low_end = (r - 2*r_se)*3*0.25
mu1 = (high_end + low_end)/2
sig1 = (high_end - mu1)/2
```

Now we can use our estimated mean and standard deviation for the prior distribution of the Omicron exponential growth rate in hospitalizations in the stan_glm function to simulate drawing from the prior to estimate the posterior and produce a linear model relating log(hospitalizations) and date.

```r
f1 <- stan_glm(log(CaseCount)~date, data=omwave, prior=normal(location=mu1, scale=sig1))
```

Now we can construct a 95% uncertainty interval using the summary of f1 (the Bayesian fit we generated above) to get the estimate for the rate of exponential increase and its standard error. A 95% uncertainty interval is then constructed by adding 1.96 standard deviations on either side of the estimate. This calculation can be done with the following code:

```r
# Code for 95% uncertainty interval with calculated prior
r_est = summary(f1)[2,1]
r_est_sd = summary(f1)[2,3]
ubound = r_est + (1.96 * r_est_sd)
lbound = r_est - (1.96 * r_est_sd)
sprintf("The 95 percent uncertainty interval for exponential rate of Omicron hospitilization increase wi
```

```
## [1] "The 95 percent uncertainty interval for exponential rate of Omicron hospitilization increase wi
```

We can also generate linear models with priors that are weakly informative (normal distributions with large standard deviations) and non-informative (improper uniform distributions) which have their own uncertainty intervals to compare with the above interval

```r
# Bayesian fit with weakly informative prior
f2 <- stan_glm(log(CaseCount)~date, data=omwave)
# Bayesian fit with non-informative (improper) prior
f3 <- stan_glm(log(CaseCount)~date, data=omwave, prior=NULL)
```

```r
# Generating an uncertainty interval for the fit made with the weakly informative prior
Wr_est = summary(f2)[2,1]
Wr_est_sd = summary(f2)[2,3]
Wubound = Wr_est + (1.96 * Wr_est_sd)
Wlbound = Wr_est - (1.96 * Wr_est_sd)
sprintf("The 95 percent uncertainty interval for exponential rate of Omicron hospitilization increase wi
```

```
## [1] "The 95 percent uncertainty interval for exponential rate of Omicron hospitilization increase wi
```

```r
# Generating an uncertainty interval for the fit made with the non-informative prior
Nr_est = summary(f3)[2,1]
Nr_est_sd = summary(f3)[2,3]
Nubound = Nr_est + (1.96 * Nr_est_sd)
Nlbound = Nr_est - (1.96 * Nr_est_sd)
sprintf("The 95 percent uncertainty interval for exponential rate of Omicron hospitilization increase wi
```

```
## [1] "The 95 percent uncertainty interval for exponential rate of Omicron hospitilization increase wi
```

These uncertainty intervals will not always be the same due to the stochastic nature of the Bayesian statistics used in stan_glm's regression modeling. However, from all the values I have observed, the uncertainty intervals are relatively close across all three priors (informative, weakly informative, and non-informative). Theoretically, the informative prior gives us the best interval with the most accurate picture of the uncertainty in this data.

The reason the intervals are so similar may be that the amount of data is enough (even with just 31 data points) to sufficiently reduce the impact of the prior on the resulting posterior distribution. Another possible explanation is that our prior simply isn't informative enough and its standard deviation is too great to have a meaningful effect on the resulting 95% uncertainty interval.

To further this analysis, we can use posterior_linpred to predict the expected value for hospitalizations in Oregon on the 10th day of the Omicron variant wave of COVID-19.

```
exppV = posterior_linpred(f1, newdata=data.frame(date=as.Date("2021-12-30")))
# Have to take the exponent of the prediction because our model is based on log(CaseCount)
exp_pred = exp(mean(exppV))
sprintf("The estimated expected value of hospitalizations in Oregon on the 10th day of the Omicron wave
```

```
## [1] "The estimated expected value of hospitalizations in Oregon on the 10th day of the Omicron wave
```

We can then predict the observed value of hospitalizations in Oregon on the 10th day of the Omicron wave by using the rstanarm function posterior_predict

```
ppV = posterior_predict(f1, newdata=data.frame(date=as.Date("2021-12-30")))
# Have to take the exponent of the prediction because our model is based on log(CaseCount)
ob_pred = exp(mean(ppV))
sprintf("The estimated number of hospitalizations in Oregon on the 10th day of the Omicron wave of COVI
```

```
## [1] "The estimated number of hospitalizations in Oregon on the 10th day of the Omicron wave of COVID
```

The difference between these two estimates is that the first (made with posterior_linpred) is attempting to estimate the expected value of hospitalizations on the 10th day of Omicron, while posterior_predict is attempting to predict an observed value that retains the error inherit in observed random variables. Because of this distinction, the predict function adds some random error epsilon centered around 0 with standard deviation sigma (sigma calculated as part of stan_glm).

This addition of randomness is evident in the fact that if you run posterior_predict several times with the exact same linear model, you will get different answers every time. If you were to run posterior_predict many times and find the sample average of those trials, you would expect to see the posterior_linpred prediction due to the weak law of large numbers.

Text problems:

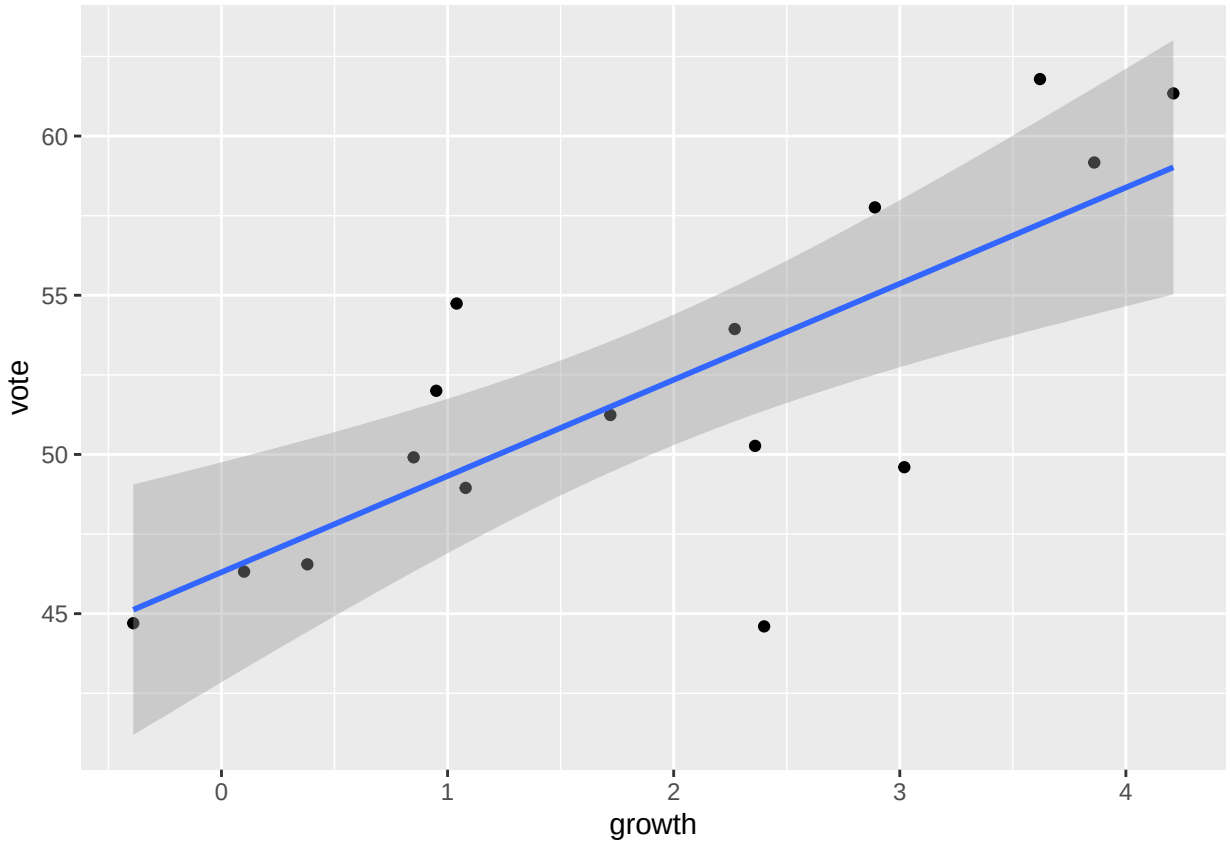
9.3: Uncertainty in the predicted expectation and the forecast

First we need to write some code to import the data the problem is talking about and to store it in a data frame

```
# Reading in the data
myfile <- "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/ElectionsEconomy/data/hibbs.da
hibbs <- read.table(myfile, header = T)
```

Now we can fit a linear model to this data using the Bayesian statistics built into the stan_glm function.

```
# Using the Bayesian approach of stan_glm to fit a regression line to this data
hibbs_fit = stan_glm(vote~growth, data=hibbs)
ggplot(hibbs, aes(y=vote, x=growth)) + geom_point() + geom_smooth(method = "stan_glm")
```



We can use the model generated above to predict the expected value and potential realized value at 2% economic growth using the slope and intercept coefficients the linear modeling program produced. The following code does this for us automatically:

```
hSumm = summary(hibbs_fit)
inter = hSumm[1,1]
slope = hSumm[2,1]
exp_2 = inter + (slope * 2)
sprintf("Our prediction for expected value of vote percentage with 2 percent economic growth is %f", ex
```

```
## [1] "Our prediction for expected value of vote percentage with 2 percent economic growth is 52.35916
```

We can then calculate the standard deviation of this prediction of expected value at 2% economic growth with formula 9.1 from the textbook. To do this, we need an estimate for error in the linear model (which

stan_glm automatically produces for us), as well as several other statistics about the data in question (here vote percentages). This calculation is automated with the following code:

```
# First we get the error sigma, found by stan_glm
ex_sig = hSumm[3,1]
# Number of data points is n
n = nrow(hibbs)
x_bar = mean(hibbs$vote)
sq_err = sum((hibbs$vote - x_bar)^2)
exp_sd = ex_sig * sqrt((1/n) + ((exp_2 - x_bar)^2)/(sq_err))
sprintf("The standard deviation for our prediction of expected value at 2 percent economic growth is %f
```

```
## [1] "The standard deviation for our prediction of expected value at 2 percent economic growth is 1.0
```

In addition to the expected value of vote percentage at 2% economic growth, we can produce a prediction for the actual observed value at 2% economic growth. To do this, we want to simulate error which is inherit in an observed value from a model with random variables.

The simulated error will be drawn from a normal distribution centered at 0, with the standard deviation the same estimate of sigma that we used above in calculating the standard deviation of the expected value prediction. The code below will calculate and print a prediction for observed vote percentage (note that this value will be different every time because of the simulated randomness that has been introduced).

```
sig = hSumm[3,1]
ob_2 = inter + (slope * 2) + rnorm(1,0,sig)
sprintf("The predicted observed value for vote percentage at 2 percent economic growth is %f", ob_2)
```

```
## [1] "The predicted observed value for vote percentage at 2 percent economic growth is 47.311994"
```

We can also calculate the standard deviation of the predicted observed value with formula 9.2. To do this, we will again use the estimated sigma value that we used above for finding the standard deviation of the expected value prediction. The calculation ends up being very similar with the only difference being a 1 added to the sum under the radical.

```
# First we get the error sigma, found by stan_glm
ex_sig = hSumm[3,1]
# Number of data points is n
n = nrow(hibbs)
x_bar = mean(hibbs$vote)
sq_err = sum((hibbs$vote - x_bar)^2)
exp_sd = ex_sig * sqrt(1 + (1/n) + ((exp_2 - x_bar)^2)/(sq_err))
sprintf("The standard deviation for our prediction of observed value at 2 percent economic growth is %f
```

```
## [1] "The standard deviation for our prediction of observed value at 2 percent economic growth is 4.1
```

Now, we can make the same predictions as above, but with the relevant prediction functions that R has already provided us. Namely, we will be using posterior_linpred to predict the estimated expected value at 2% economic growth and posterior_predict to generate a predicted observed value at that economic growth level.

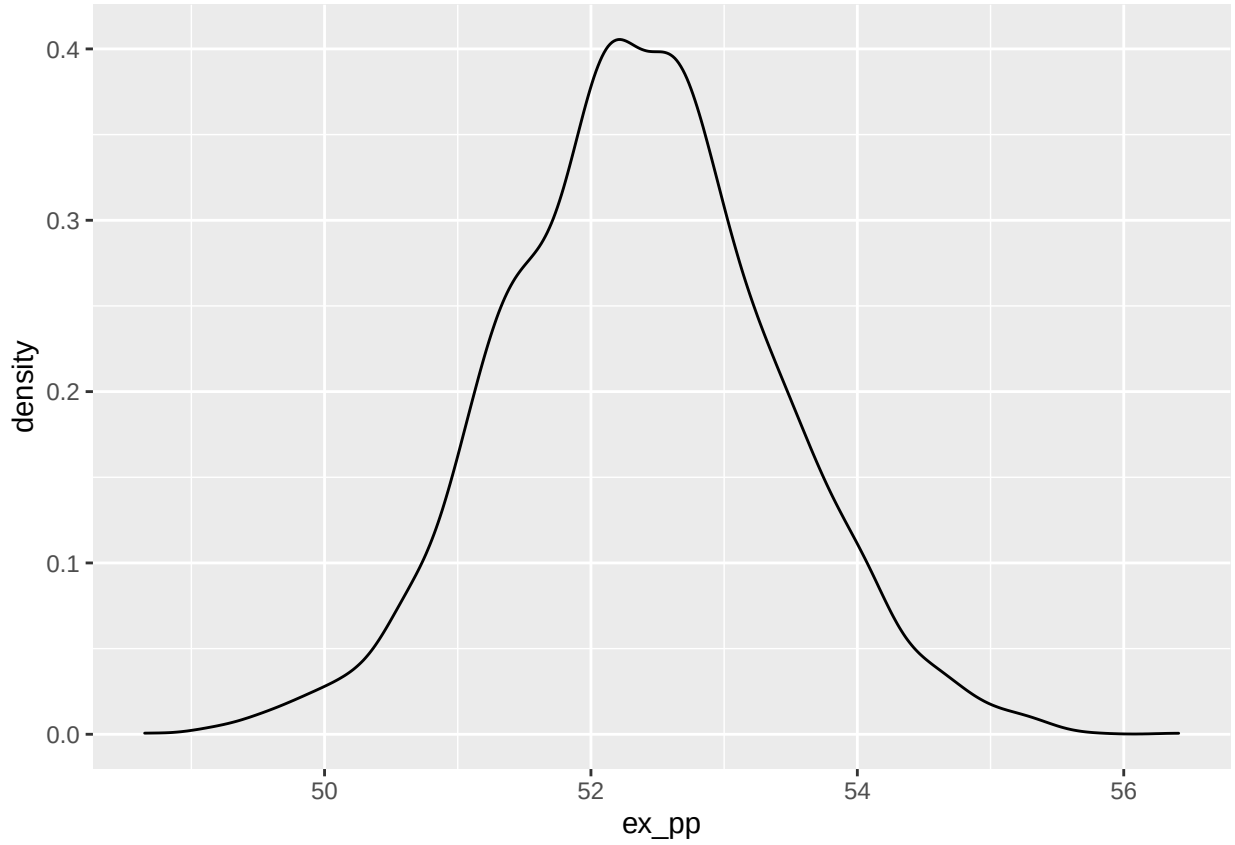
```r
# This function will return a vector of simulated expected values
# at 2% economic growth
ex_pp <- posterior_linpred(hibbs_fit, newdata=data.frame(growth=2))
# We then take the mean of this vector to get our prediction
ex_pred = mean(ex_pp)
sprintf("Our prediction for expected value of vote percentage with 2 percent economic growth is %f", ex
```

```
## [1] "Our prediction for expected value of vote percentage with 2 percent economic growth is 52.35916
```

```r
# Finding the standard deviation of the prediction consists of finding the
# sample variance of the vector of simulated expected values
ex_SD = var(ex_pp)
sprintf("The standard deviation for our expected value prediction at 2 percent economic growth is %f", 
```

```
## [1] "The standard deviation for our expected value prediction at 2 percent economic growth is 1.0378
```

```r
ggplot(data.frame(ex_pp),aes(ex_pp)) + geom_density()
```



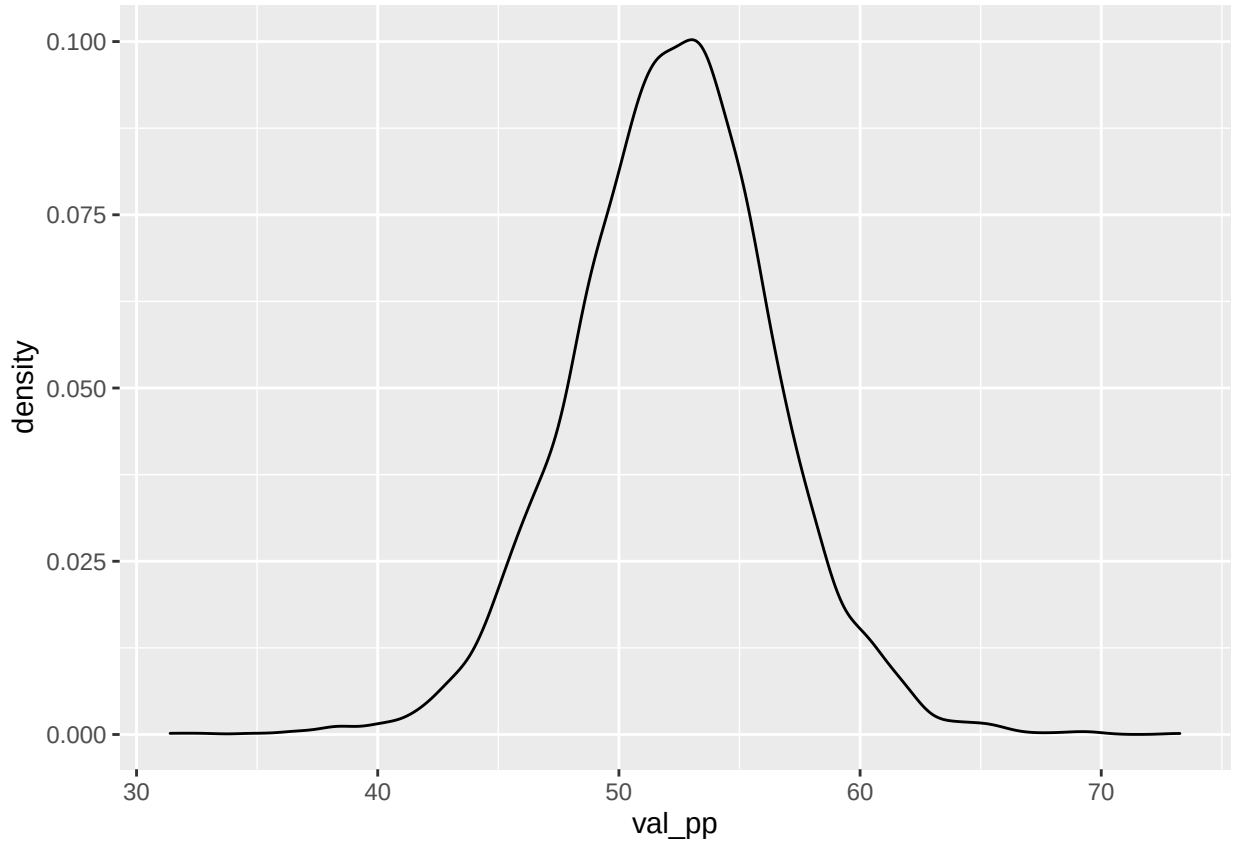

```r
# This function will return a vector of simulated expected values
# at 2% economic growth
val_pp <- posterior_predict(hibbs_fit, newdata=data.frame(growth=2))
val_pred = mean(val_pp)
sprintf("Our prediction for obersved value of vote percentage with 2 percent economic growth is %f", val
```

```
## [1] "Our prediction for obersved value of vote percentage with 2 percent economic growth is 52.25473
```

```
val_SD = var(val_pp)
sprintf("The standard deviation for our observd value prediction at 2 percet economic growth is %f", val
```

```
## [1] "The standard deviation for our observd value prediction at 2 percet economic growth is 17.47328
```

```
ggplot(data.frame(val_pp),aes(val_pp)) + geom_density()
```



Note that the above code also produced two graphs. These graphs model the predictions and standard deviations for predicted expected and observed value of vote percentage at the specified growth. Here, the standard deviations are easier to interpret as they are shown visually.

We can note that the predictions for observed and expected values are roughly the same, but the observed prediction has a much larger standard deviation. This makes sense intuitively as we would expect to see a larger degree of variation with observed values than with expected values. Mathematically, this larger variation can be explained by the simulated randomness that the posterior_predict algorithm adds to its values.

9.4: Partial pooling

We know from the textbook that we can use the following formula to calculate the posterior mean estimate:

$$\hat{\theta} = \left(\frac{1}{(se_{prior})^2} \hat{\theta}_{prior} + \frac{1}{(se_{data})^2} \hat{\theta}_{data} \right) / \left(\frac{1}{(se_{prior})^2} + \frac{1}{(se_{data})^2} \right)$$

It has been given to us that the posterior mean is 49% meaning that

$$\hat{\theta} = 0.49$$

Furthermore, we know that the mean and standard deviation of the prior are 42% and 5% respectively (here we can consider se_{prior} the same as the standard deviation because the prior is just a distribution without a sample size n). We have also been given that the observed vote percentage is 54%. This gives us the following equation:

$$0.49 = \left(\frac{1}{(0.05)^2}(0.42) + \frac{1}{(se_{data})^2}(0.54)\right) / \left(\frac{1}{(0.05)^2} + \frac{1}{(se_{data})^2}\right)$$

If we solve the above equation for the standard error of the data, we end up with:

$$(se_{data})^2 = 0.00178571 \implies se_{data} = \pm 0.0422577$$

Since standard error must be positive, we can ignore the negative solution to this equation. Percentage of vote can be modeled with a binomial distribution, meaning the standard error of the data would be equal to:

$$se_{data} = \sqrt{\frac{(\theta_{data})(1-\theta_{data})}{n}}$$

Using the calculated standard error of data and the given observed vote percentage in the data we get the following equation for the standard error of this data:

$$se_{data} = 0.0422577 = \sqrt{\frac{(0.54)(1-0.54)}{n}} \implies n = 139.104 \approx 139$$

Therefore, the sample size of the data must be 139. With the known information, we can now calculate the standard error of the posterior distribution with formula 9.4 which states that:

$$se_{Bayes} = 1 \Bigg/ \sqrt{\frac{1}{se_{prior}^2} + \frac{1}{se_{data}^2}}$$

We know that the standard error of the prior is 0.05 and we have calculated that the standard error of the data is 0.0422577. Therefore, we end up with the following equation for standard error of the posterior distribution:

$$se_{Bayes} = 1 \Bigg/ \sqrt{\frac{1}{0.05^2} + \frac{1}{0.04225771^2}} = 0.0322749$$

When we solve for the posterior standard error with the above equation, we end up with 0.0322749. We can assume that the posterior distribution follows a roughly normal pdf and use this assumption to calculate the posterior probability that this candidate wins the election.

Let us establish that to win the election, the candidate must get at least 50% of the vote. The expected value of the posterior distribution has been given to us to be 0.49, so that means that to win the election, the candidate must have 0.01 more than the expected vote percentage to win. With our calculated standard error of the posterior distribution, we can translate this to be 0.3098382954 standard deviations above the mean. Using the pnorm function in R, we know that the probability that an observation of a normal distribution falling 0.3098382954 or more above its mean is 0.378342. We therefore can conclude that that the posterior probability that this candidate wins the election is approximately 37.83%.

9.6: Bayesian inference with a zero-centered informative prior on the log scale

First consider the given information: the point estimate of the multiplicative effect is 1.42 with a 95% confidence interval of [1.02, 1.98], on a scale for which 1.0 corresponds to a multiplying by 1, or no effect. For the first part of the problem, we want to take the natural log of the point estimate and its confidence interval bounds to get the error and standard estimate of the log of the multiplicative effect of the treatment.

```
# Here the log function in R does a natural log by default
log_est = log(1.42)
log_ubound = log(1.98)
log_lbound = log(1.02)
```

This calculation tells us that the estimate of the log of the multiplicative effect of the treatment is 0.3506569 and it has a 95% confidence interval of [0.01980263, 0.6830968].

We can approach this data from a Bayesian point of view using the following code to calculate the mean of the posterior for the log multiplicative effect of the treatment:

```
# gap is the distance of half the 95% uncertainty interval
gap = log_ubound - log_est
# We divide this gap by 1.96 because 1.96 sd's on both sides generates a 95% interval
se_data = (gap/1.96)
se_prior = 0.1
# We can then use formula 9.3 from the textbook to calculate the Bayes' estimate
bayes_est = ((1/(se_data^2))*(0.3506569))/((1/(se_prior^2)) + 1/(se_data^2))
```

We can then use the code below to calculate the value of exponentiated mean. We have to raise e to our estimate from above because the estimate above is an estimate for log of multiplicative effect.

```
mult_eff = exp(bayes_est)
sprintf("The estimated multiplicative effect from a Bayesian point of view is %f", mult_eff)
```

```
## [1] "The estimated multiplicative effect from a Bayesian point of view is 1.094666"
```

Note that it does equal 1.09 (an estimated treatment effect of +9%), which is what we expected according to the problem description.

We can compute the standard error of this estimate using a Bayesian approach with the following calculation:

```
# This line of code uses theorem 9.4 from the textbook
se_bayes = 1/sqrt((1/(se_prior^2)) + (1/(se_data^2)))
# We then add 1.96 of these standard errors to both sides to generate a
# 95% uncertainty interval for the estimate
bayes_ubound = bayes_est + (1.96 * se_bayes)
bayes_lbound = bayes_est - (1.96 * se_bayes)
exp_u = exp(bayes_ubound)
exp_l = exp(bayes_lbound)
sprintf("Our Bayesian 95 percent uncertainty interval is then [%f, %f]", exp_l, exp_u)
```

```
## [1] "Our Bayesian 95 percent uncertainty interval is then [0.924603, 1.296009]"
```

Here we get that our Bayes 95% interval comes to [0.9246034, 1.296009] when we exponentiate it. Note that this is slightly different than the textbook's answer, but it is a sufficiently small difference that we could attribute it to rounding error.

With this small note in mind, we can see that following the book's instructions, we end up with the same estimate and confidence interval as expected by taking taking the log of the original interval and using Bayesian statistics and exponentiation to construct a new interval.

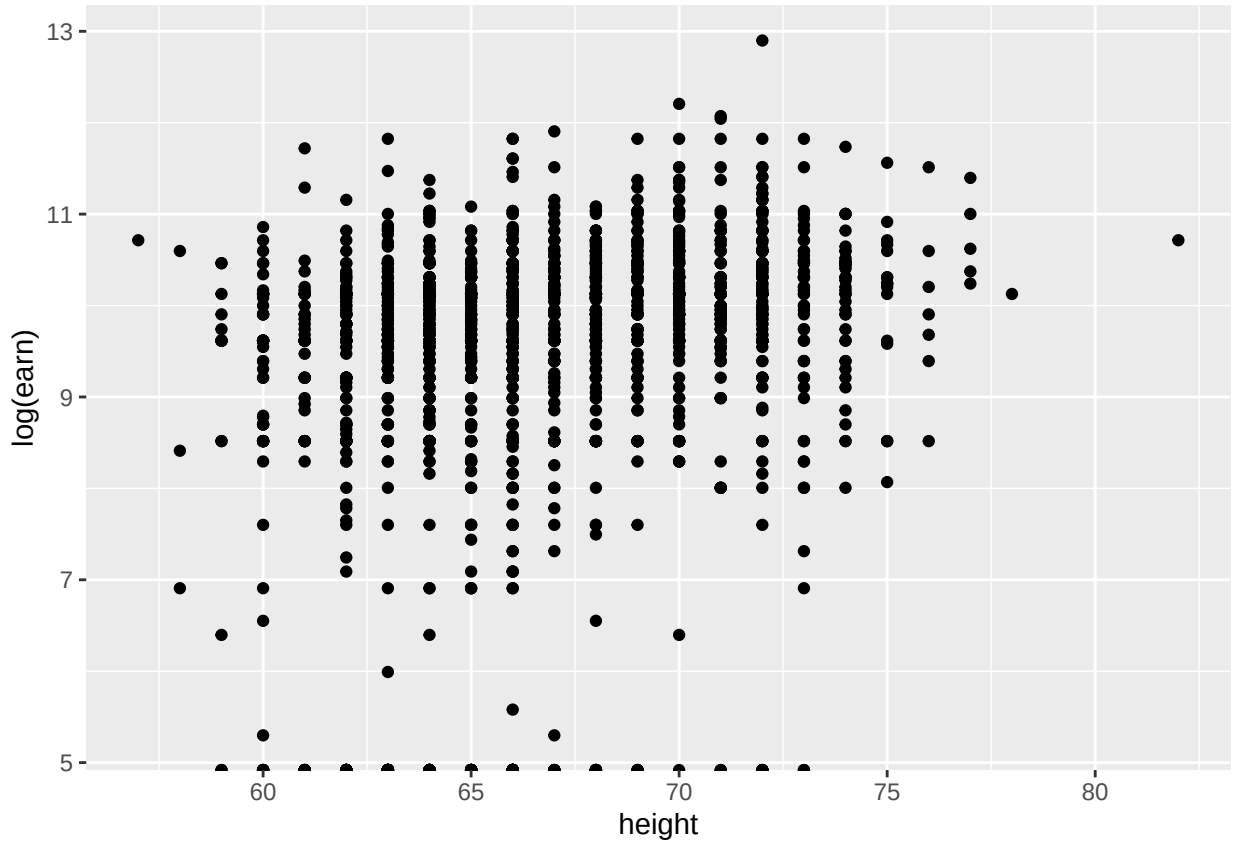
9.7: Uniform, weakly informative, and informative priors

First, let's import the data from the authors' GitHub page and format it as a data frame:

```
earnings = read.csv("https://raw.githubusercontent.com/avehtari/ROS-Examples/master/Earnings/data/earni
```

The earnings variable is spread out over several magnitudes, so let's take the natural log of it to get a better picture of the potential linear relationship between height and earnings. The scatter plot of height against the log of earnings is displayed below:

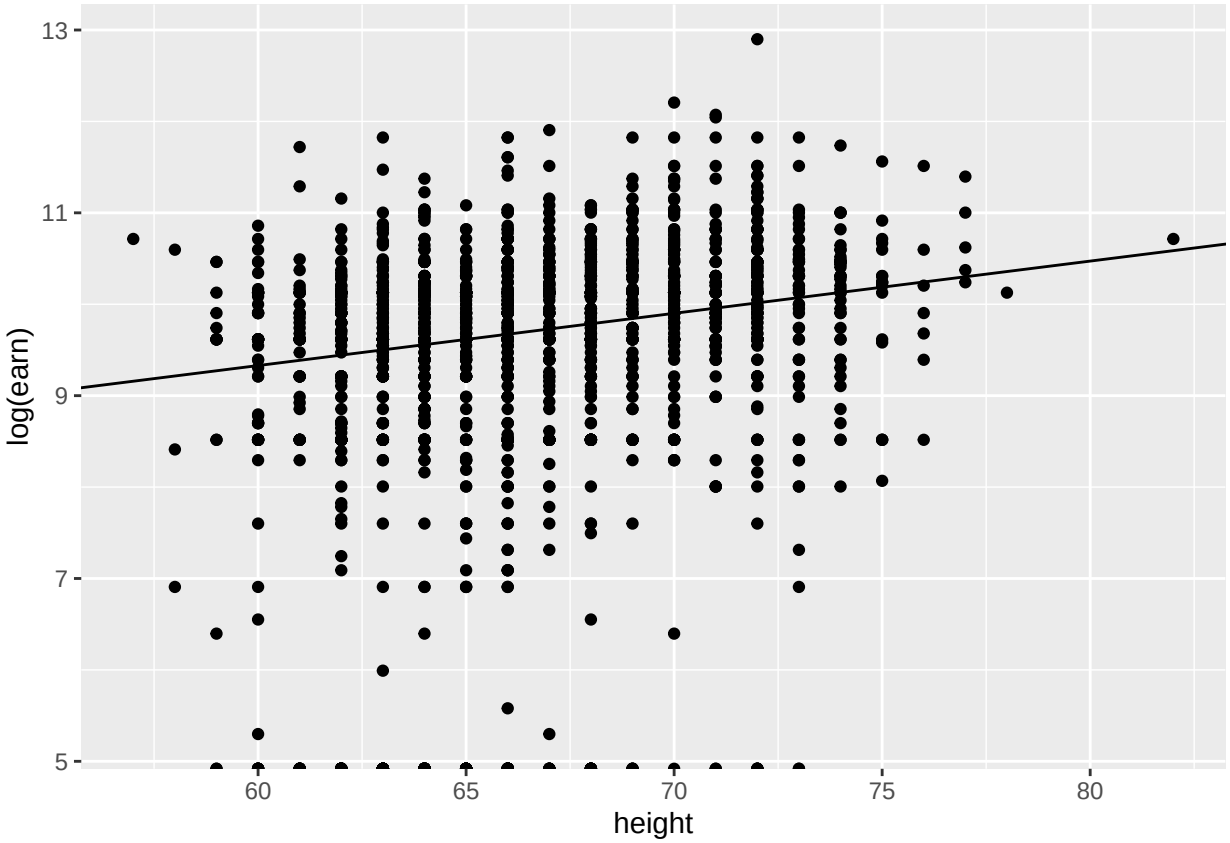
```
ggplot(earnings, aes(x=height,y=log(earn))) + geom_point()
```



For the uniform (improper) prior, we can use stan_glm and set the prior to NULL.

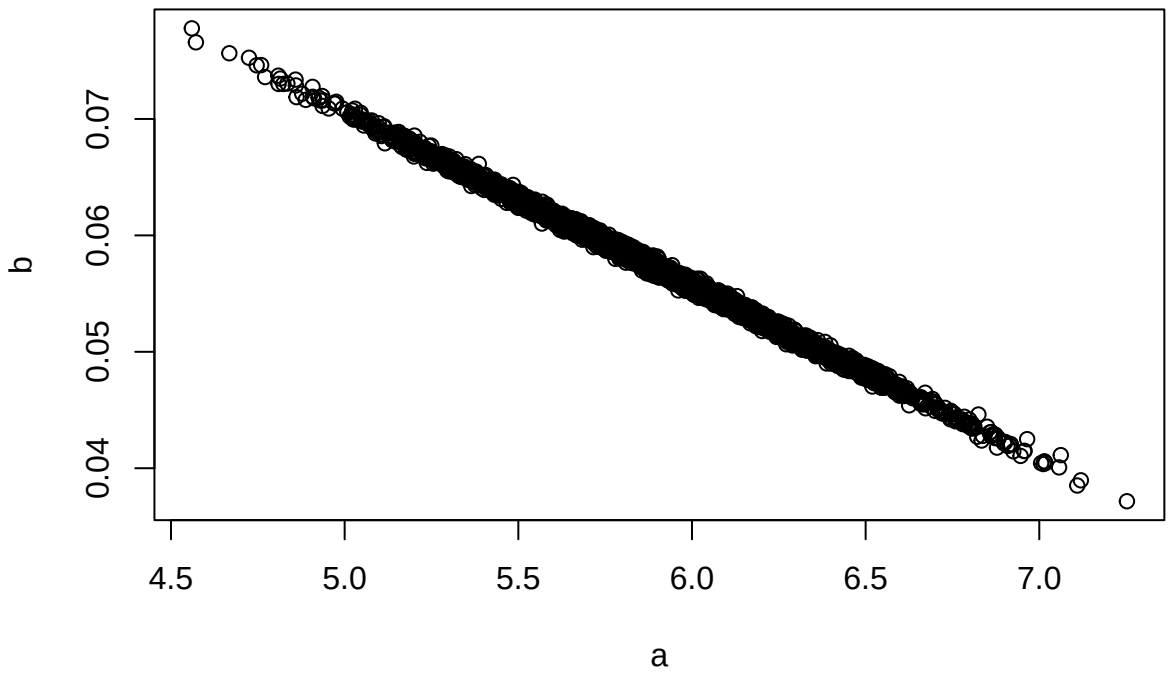
```
library(dplyr)
# We want to filter out the zero values for the earnings, so not to cause
# problems when we take the log of the earnings values
earn_subset = filter(earnings, earn > 0)
```

```
# Fitting a linear model using our assumptions and Bayesian inference
uni_earn_fit <- stan_glm(log(earn)~height, data=earn_subset, prior=NULL)
coefs = coef(uni_earn_fit)
ggplot(earnings, aes(x=height,y=log(earn))) + geom_point() + geom_abline(slope = coefs[2], intercept = c
```



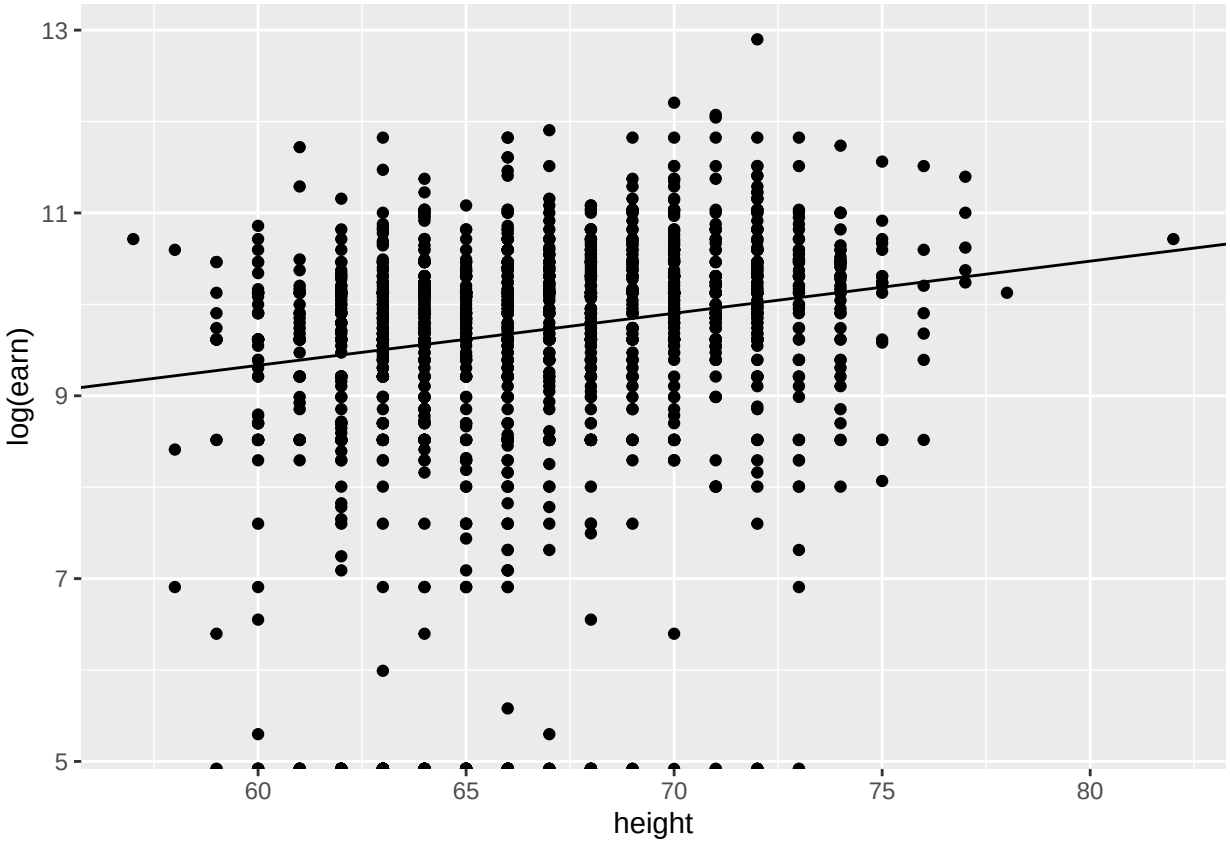
We can then display the Bayesian random simulations used with this improper prior by using the same data visualization techniques that the book makes use of in section 9.5

```
uni_sims <- as.data.frame(uni_earn_fit)
a <- uni_sims[,1]
b <- uni_sims[,2]
plot(a, b)
```



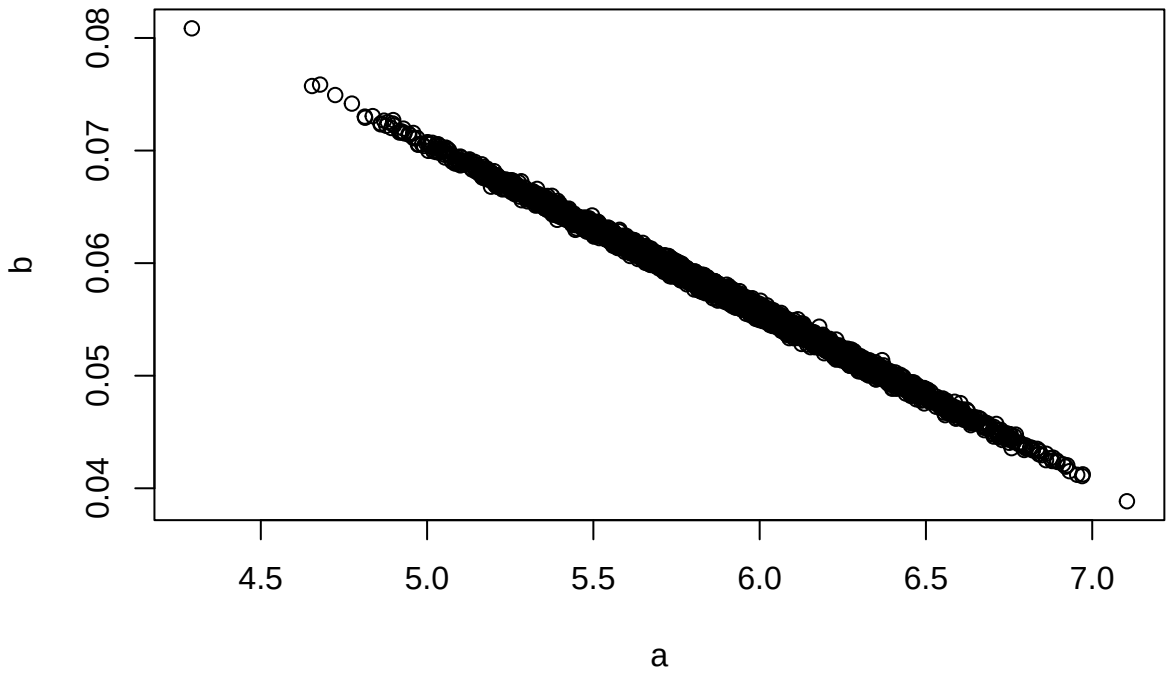
The next step up in specifying prior distributions is using the default weak prior distribution that is used by stan_glm. Stan functions will implicitly apply this kind of weak prior, so we can generate this fit by not specifying any prior at all when creating our model. In reality, this default is not really Bayesian because it is determined in part by the data itself–making it not really a "prior" distribution. The expected result of this linear regression fitting is not much different than the uniform prior used above.

```
def_earn_fit <- stan_glm(log(earn)~height, data=earn_subset)
coefs = coef(def_earn_fit)
ggplot(earnings, aes(x=height,y=log(earn))) + geom_point() + geom_abline(slope = coefs[2], intercept = c
```



Just like we did above, we can visualize these simulations by extracting the data from the Bayesian simulations into a data frame and plotting it:

```
def_sims <- as.data.frame(def_earn_fit)
a <- def_sims[,1]
b <- def_sims[,2]
plot(a, b)
```

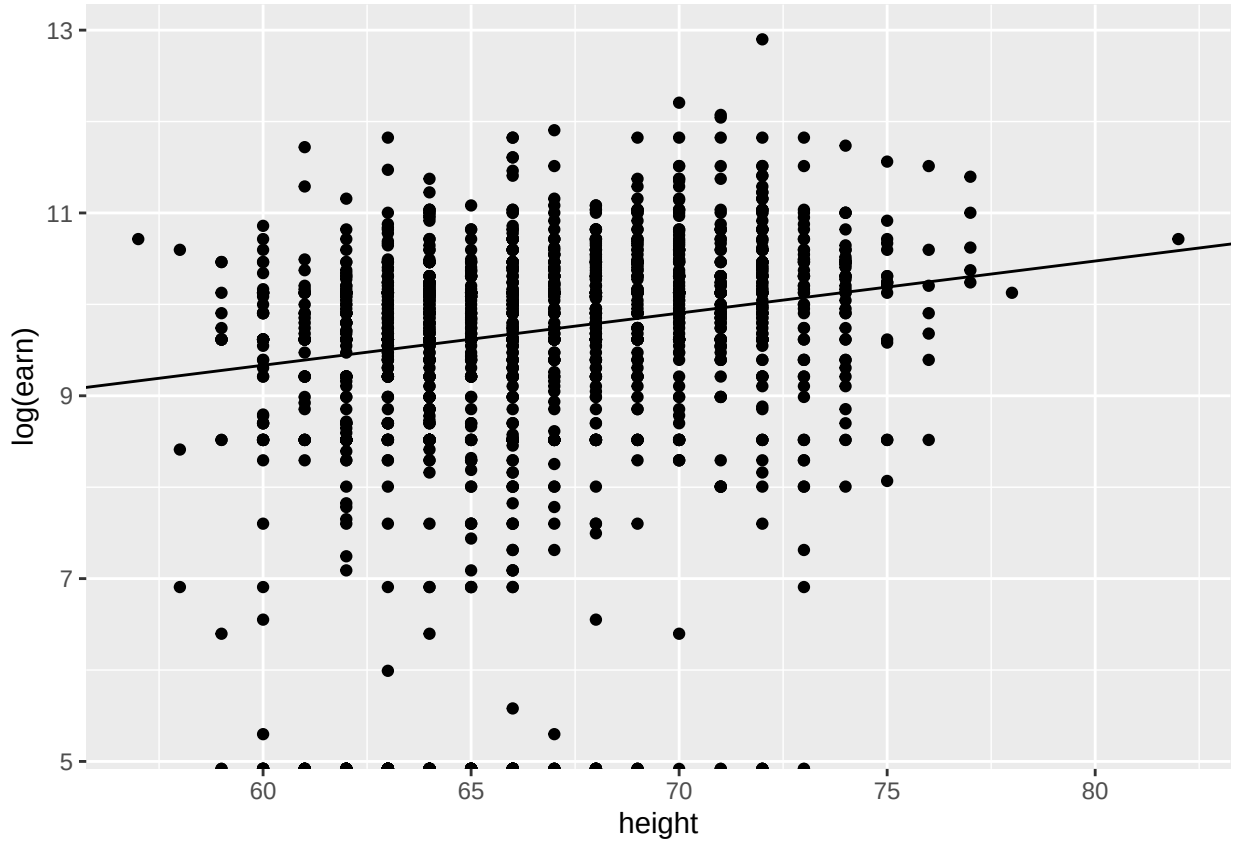


For the weakly informative prior, we can note that there is no reason to think that there exists a linear relationship between height and earnings–whether that be positive or negative. Because of this, we can center our prior around 0 (which corresponds to a multiplicative effect of 1 when we exponentiate out the log).

For the standard deviation of the prior, we can reasonably assume that the multiplicative effect of height on earnings is between 0.5 and 2. That is, we would not expect to find that tall people make twice as much money or half as much money as shorter people based on our experience in everyday life. We can set these two ends of the spectrum as 2 standard deviations away from our mean of 1. When we consider the log of earnings, this translates to a standard deviation of 0.345.

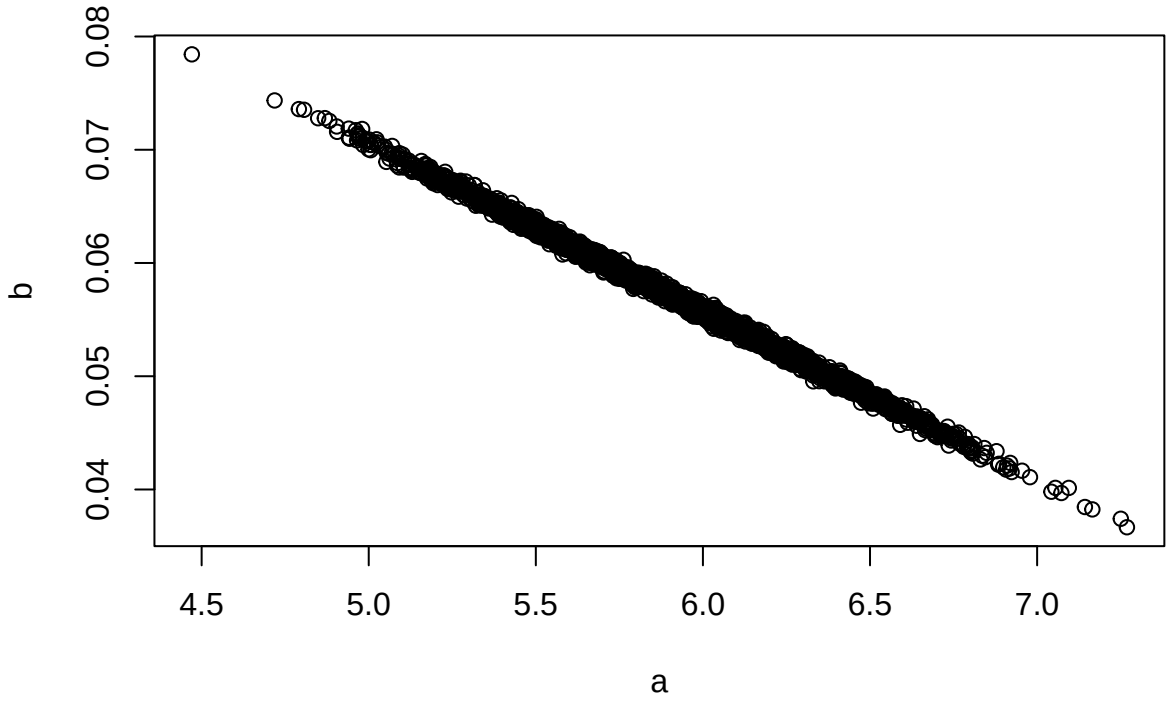
We don't necessarily care too much about the intercept in this problem (no one has a height of 0 so it is meaningless), so we will focus only on establishing a weak prior for the slope. Using our assumptions above we can fit a Bayesian model to this data with a weak prior using the following code:

```
wk_earn_fit <- stan_glm(log(earn)~height, data=earn_subset, prior=normal(location=0, scale=0.345))
coefs = coef(wk_earn_fit)
ggplot(earnings, aes(x=height,y=log(earn))) + geom_point() + geom_abline(slope = coefs[2], intercept = c
```



Now we can do the same data visualization as above:

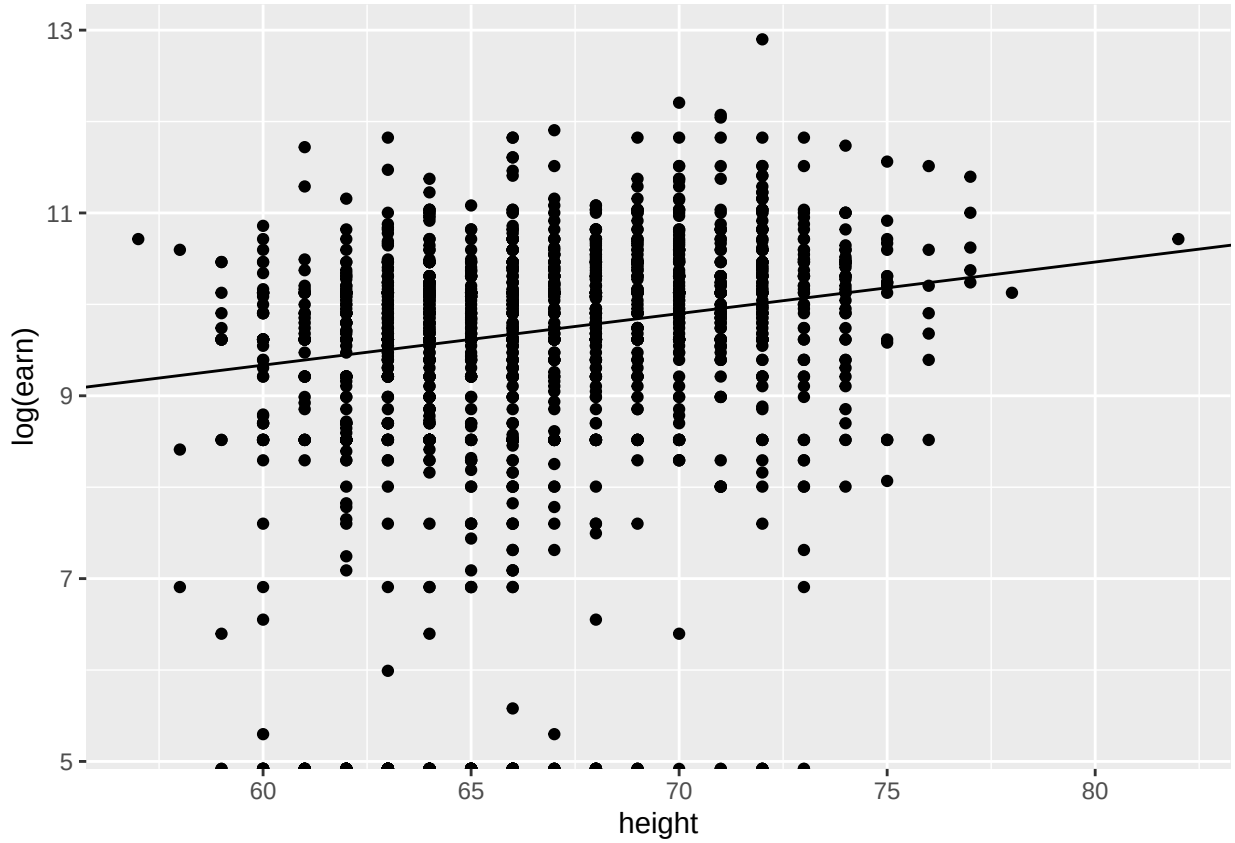
```
wk_sims <- as.data.frame(wk_earn_fit)
a <- wk_sims[,1]
b <- wk_sims[,2]
plot(a, b)
```



For the informative prior distribution let's assume that we have reason to believe that height does not affect earnings. Because of this, we can make the expected value of our prior 0 with a reasonably small standard deviation of 0.05 (which is some indication of how confident we are in our belief that there is no linear correlation between these two variables). Let us also assume that the prior distribution is roughly normal.

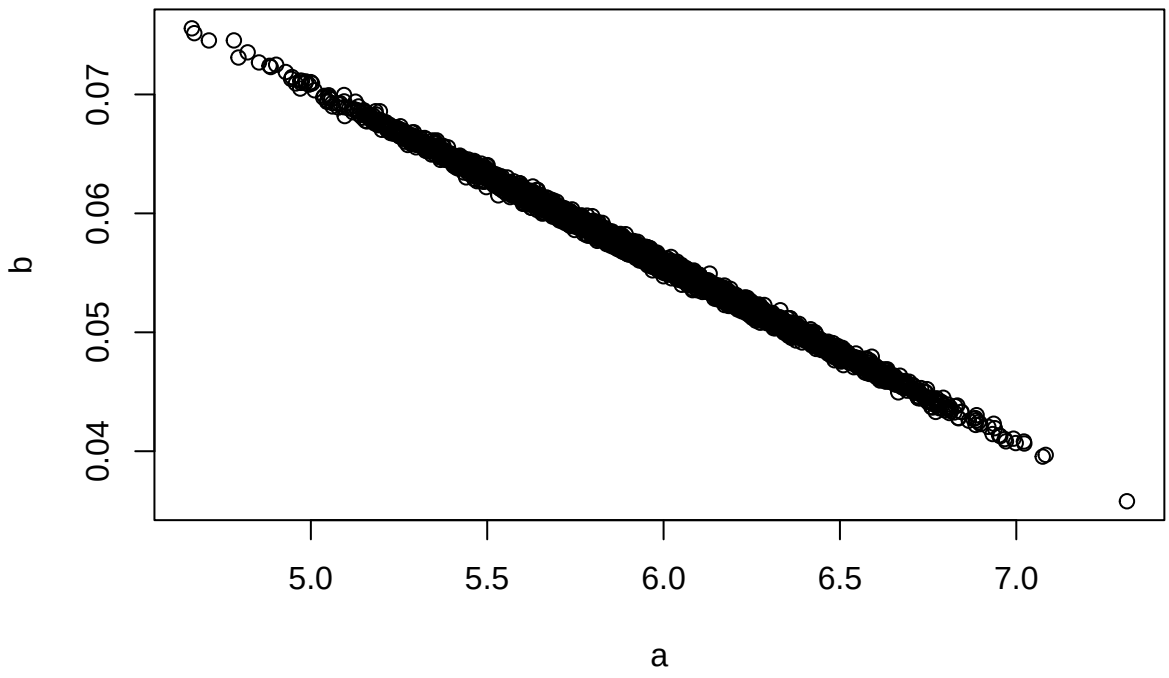
With these assumptions about the prior distribution of the effect size of height on earnings we can do the following Bayesian estimation of the effect size with stan_glm:

```
# Fitting a linear model using our assumptions and Bayesian inference
inf_earn_fit <- stan_glm(log(earn)~height, data=earn_subset, prior=normal(location=0, scale=0.05))
coefs = coef(inf_earn_fit)
ggplot(earnings, aes(x=height,y=log(earn))) + geom_point() + geom_abline(slope = coefs[2], intercept = c
```



We can then visualize this fit with the same visualization scheme as above:

```
inf_sims <- as.data.frame(inf_earn_fit)
a <- inf_sims[,1]
b <- inf_sims[,2]
plot(a, b)
```



We can directly compare the estimated slopes of the linear model that results each of the above priors by printing all of them together with the following code:

```
sprintf("The predicted multiplcative effect of height on income with the uniform prior is %f",
        exp(coef(uni_earn_fit)[2]))
```

```
## [1] "The predicted multiplcative effect of height on income with the uniform prior is 1.058804"
```

```
sprintf("The predicted multiplcative effect of height on income with the default prior is %f",
        exp(coef(def_earn_fit)[2]))
```

```
## [1] "The predicted multiplcative effect of height on income with the default prior is 1.058651"
```

```
sprintf("The predicted multiplcative effect of height on income with the weakly informative prior is %f
        exp(coef(wk_earn_fit)[2]))
```

```
## [1] "The predicted multiplcative effect of height on income with the weakly informative prior is 1.05
```

```
sprintf("The predicted multiplcative effect of height on income with the informative prior is %f",
        exp(coef(inf_earn_fit)[2]))
```

```
## [1] "The predicted multiplcative effect of height on income with the informative prior is 1.058030"
```

The estimated slopes for all of these will be something different every time due to the simulated randomness included in Bayesian R functions like stan_glm. However, after running this a few times, I usually end up with similar slopes estimated multiplicative effects across the different priors. This may suggest that I should use more specific priors or that the data is enough so that the prior doesn't have as substantial of an effect on the posterior as in other cases.‘