Simple Linear Regression

Metric Predicted Variable With One Metric Predictor Variable

Tim Frasier

Goals & General Idea

Goals

When would we use this type of analysis?

- Simple linear regression
  - Relationship between two variables (how does one change in relation to the other?)
  - Predict values of one parameter based on values of the other

General Idea

- Trying to quantify the relationship between two different sets of data
  - One (we'll call `y`) is the response (or predicted) variable
  - The other (we'll call `x`) is the predictor variable

- Examples:
  - Height versus weight
  - GPA versus SAT scores
  - etc.

Equation

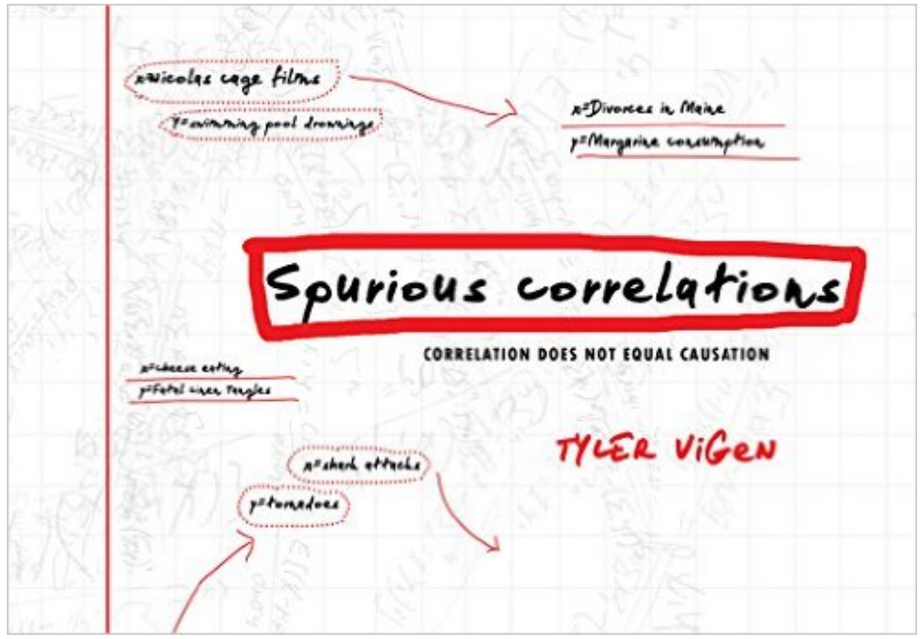
$$y = \beta_0 + \beta_1 x$$

- β_0 the *y*-intercept (the value of `y` when `x = 0`)
- β_1 the slope of the line, indicating how much `y` increases when `x` increases by 1 unit

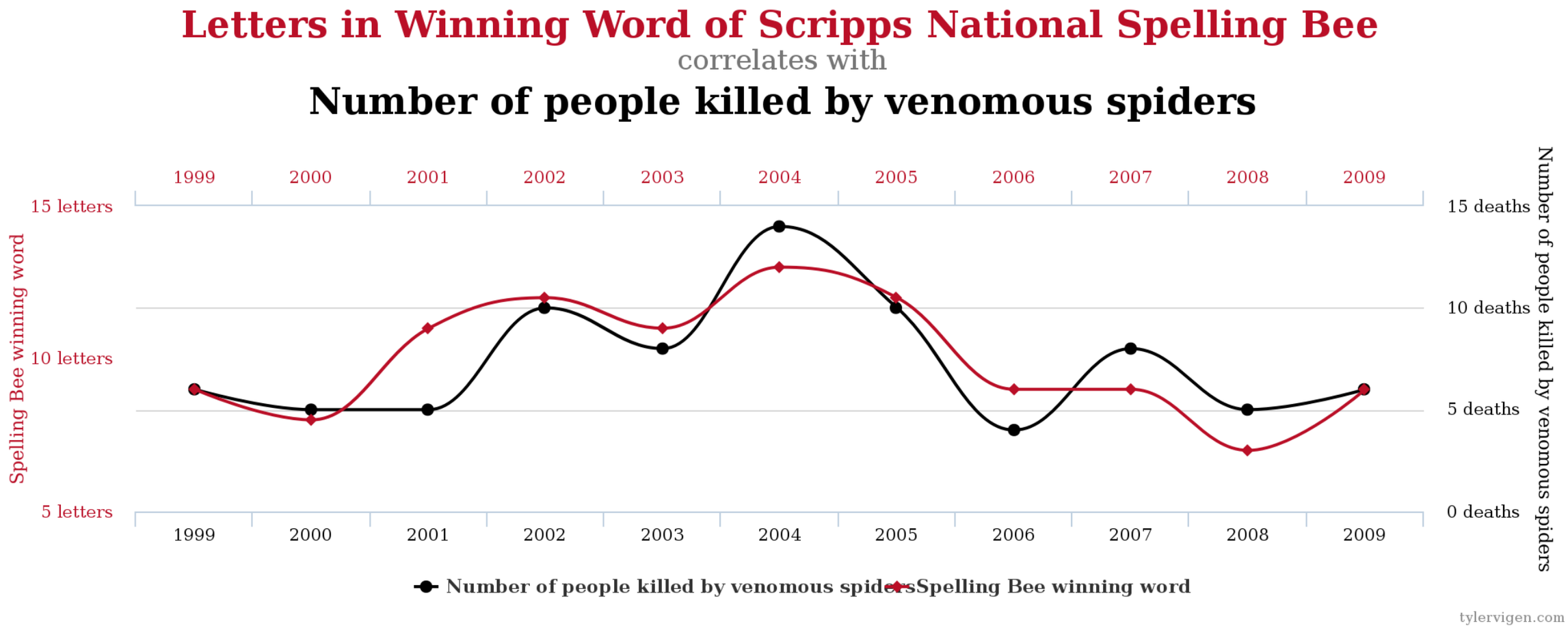
Equation

$$y = \beta_0 + \beta_1 x$$

- β_0 the *y*-intercept (the value of `y` when `x = 0`)
- β_1 the slope of the line, indicating how much `y` increases when `x` increases by 1 unit
- Assumes homogeneity of variance (at every value of `x`, the variance of `y` is the same)
- Describes tendencies, not causation (that would require other data)



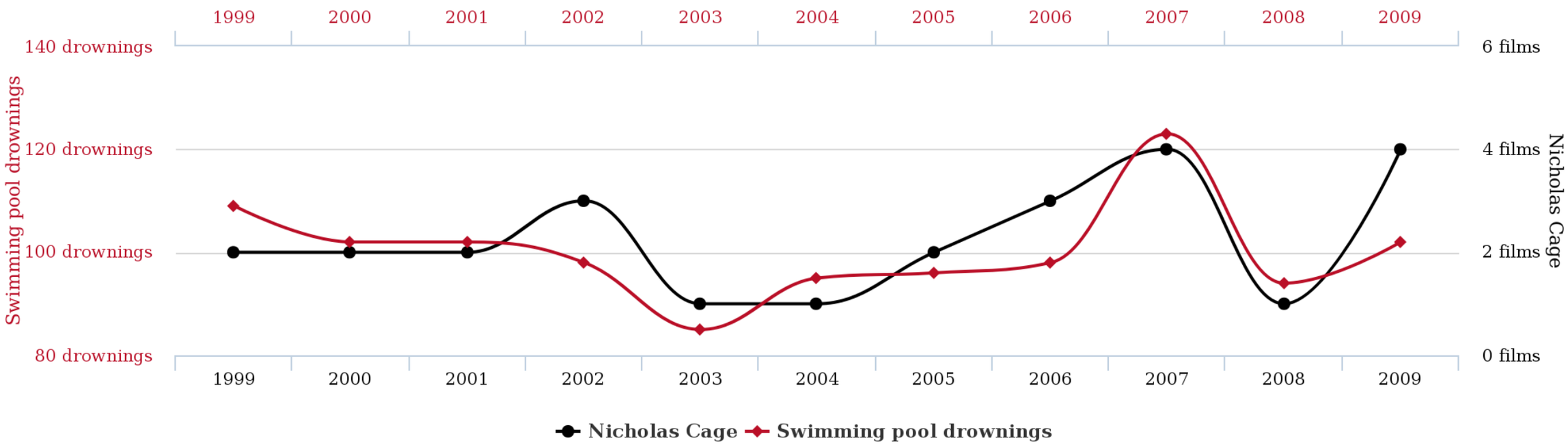
tylervigen.com


Letters in Winning Word of Scripps National Spelling Bee
correlates with
Number of people killed by venomous spiders
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
15 letters
10 letters
5 letters
Spelling Bee winning word
15 deaths
10 deaths
5 deaths
0 deaths
Number of people killed by venomous spiders
Number of people killed by venomous spiders
Spelling Bee winning word
tylervigen.com

Number of people who drowned by falling into a pool

correlates with

Films Nicolas Cage appeared in



tylervigen.com

General Idea

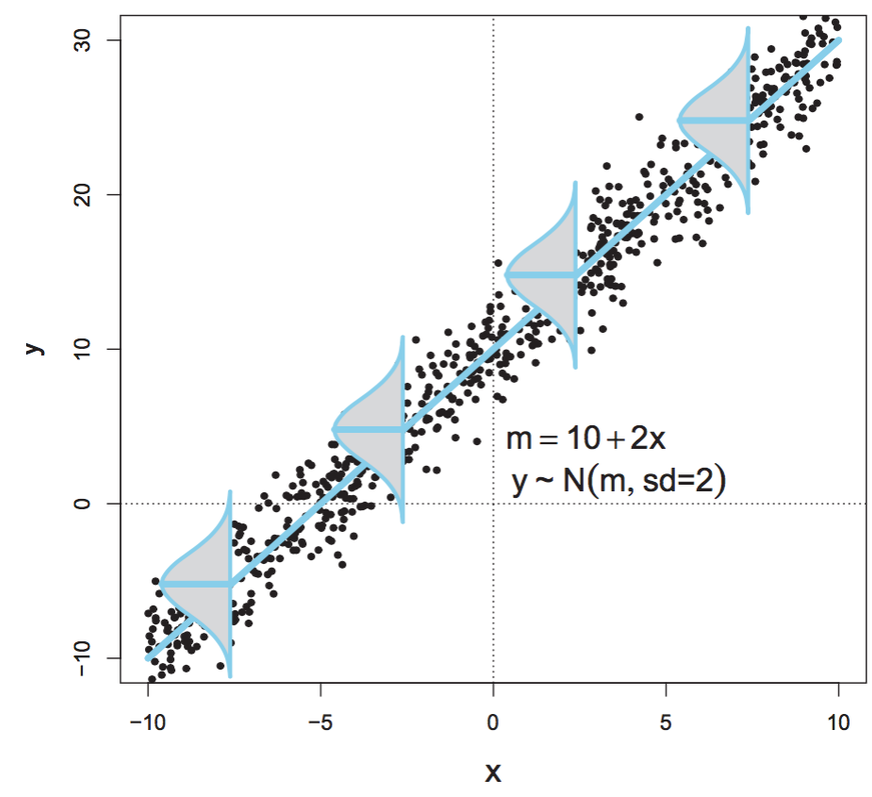
$$y = \beta_0 + \beta_1 x$$

- Our data will be the `x` and `y` values
- Our goal is to estimate the β_0 and β_1 parameters

General Idea

$$y = \beta_0 + \beta_1 x$$

- Our data will be the `x` and `y` values
- Our goal is to estimate the β_0 and β_1 parameters
- We do not expect a perfect match for all points. Instead, we're really trying to predict the mean `y` value for a given `x` value, and the noise around that mean



Kruschke (2015) p. 478

General Idea

$$y = \beta_0 + \beta_1 x$$

- Our data will be the `x` and `y` values
- Our goal is to estimate the β_0 and β_1 parameters
- We do not expect a perfect match for all points. Instead, we're really trying to predict the mean `y` value for a given `x` value, and the noise around that mean
- We'll assume a normal distribution of variation around these predicted values for now.

General Idea

- Mathematically, this is

mean *sd*

$$y \sim \text{normal}(\text{mu}, \text{sigma})$$

where

$$\text{mu} = \beta_0 + \beta_1 x$$

- Our goal is to determine what combinations of β_0, β_1 and sigma are most credible given the data

General Idea

- Mathematically, this is

mean

Stochastic

$$y \sim \text{normal}(\text{mu}, \text{sigma})$$

where

$$\text{mu} = \beta_0 + \beta_1 x$$

Deterministic

- Ou vhat combinations of β_0, β_1 and sigma are mo ata

Data

Data

- Height and weight data from Kruschke (2011)
- "`HtWt.csv`"

Data

- Read data into R

```
htwt = read.table("HtWt.csv", header = TRUE, sep = ",")
```

htwt

	height	weight
1	62.9	133.3
2	62.2	128.7
3	62.1	148.0
4	61.6	102.7
5	69.0	202.4
6	67.9	204.0
7	65.6	142.1
8	69.7	150.7
9	67.2	137.2
10	67.2	127.9
11	67.0	161.9
12	71.7	185.0
13	71.0	171.9

Data

- Let's get a feel for the data

```
summary(htwt)

   height          weight
 Min.   :58.50   Min.   : 86.1
 1st Qu.:64.08   1st Qu.:129.6
 Median :66.35   Median :150.3
 Mean   :66.65   Mean   :159.2
 3rd Qu.:69.05   3rd Qu.:184.8
 Max.   :78.10   Max.   :272.5
```

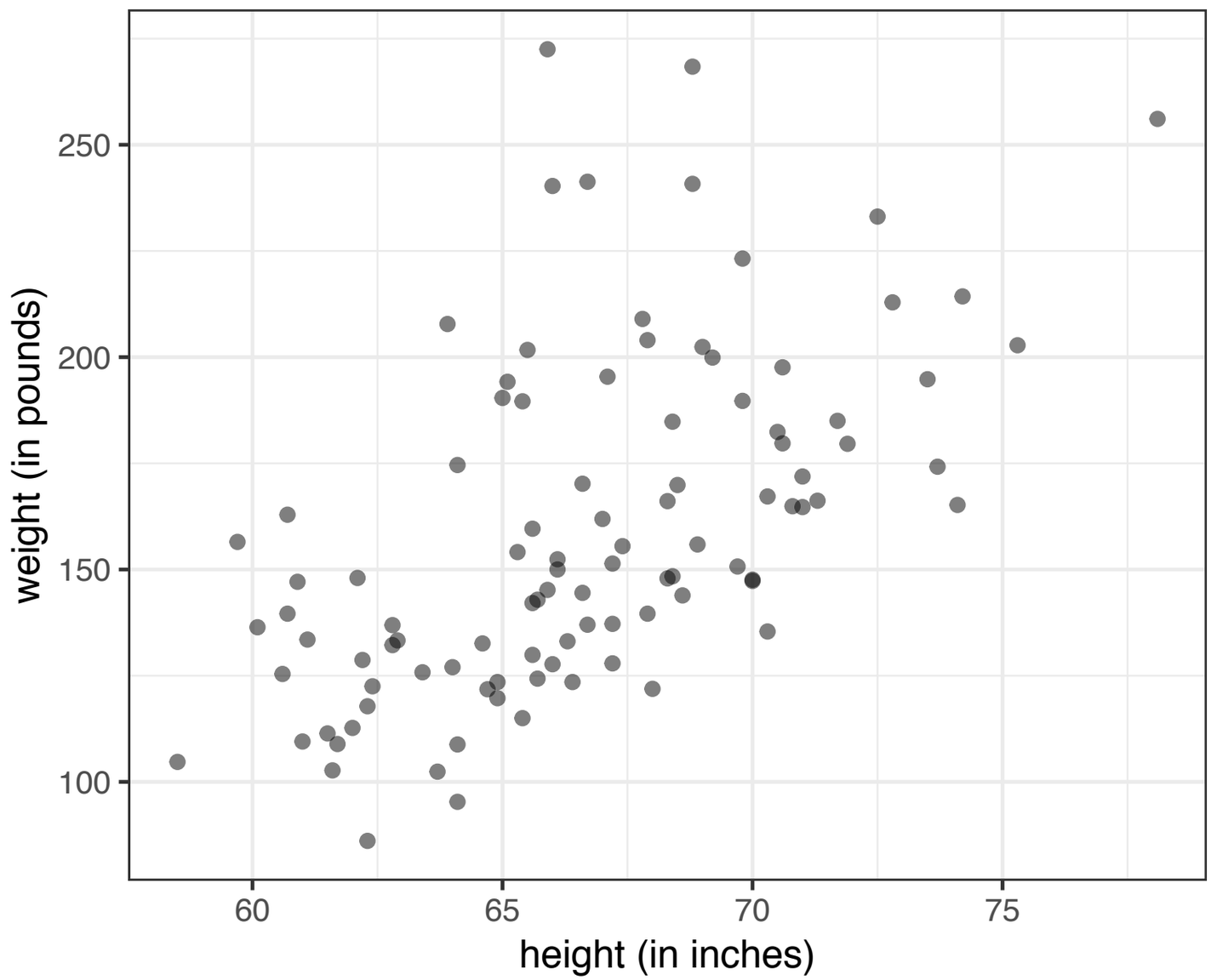
Data

```
sd(htwt$height)
[1] 3.8668
```

```
sd(htwt$weight)
[1] 39.65889
```

Data

```
ggplot(htwt, aes(x = height, y = weight)) +
  theme_bw() +
  geom_point(alpha = 0.5) +
  ylab("weight (in pounds)") +
  xlab("height (in inches)")
```



Frequentist Approach

Frequentist Approach

`lm` function (linear model)

```
model = lm(htwt$weight ~ htwt$height)

summary(model)

Call:
lm(formula = htwt$weight ~ htwt$height)

Residuals:
    Min      1Q  Median      3Q     Max 
-48.910 -22.624  -7.591  15.405 117.731 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) -231.8232    56.7373  -4.086 8.99e-05 ***
htwt$height    5.8663     0.8499   6.903 5.10e-10 ***
---
Signif. codes:  
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 32.7 on 98 degrees of freedom
Multiple R-squared:  0.3271,	Adjusted R-squared:  0.3203 
F-statistic: 47.65 on 1 and 98 DF,  p-value: 5.102e-10
```

Frequentist Approach

`lm` function (linear model)

```
model = lm(htwt$weigh

summary(model)

Call:
lm(formula = htwt$wei

Residuals:
    Min      1Q  Median      3Q     Max
-48.910 -22.624  -7.591  15.405 117.731

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -231.8232    56.7373  -4.086 8.99e-05 ***
htwt$height    5.8663     0.8499   6.903 5.10e-10 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 32.7 on 98 degrees of freedom
Multiple R-squared:  0.3271,  Adjusted R-squared:  0.3203
F-statistic: 47.65 on 1 and 98 DF,  p-value: 5.102e-10
```

Estimate for β_0. A person will weigh -232 pounds when at a height of 0 inches. Note that this value will often be biologically meaningless.

Frequentist Approach

`lm` function (linear model)

```
model = lm(htwt$weight ~ htwt$height)

summary(model)

Call:
lm(formula = htwt

Residuals:
    Min      1Q  M
-48.910 -22.624  -7.591  15.405 117.731

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -231.8232    56.7373  -4.086 8.99e-05 ***
htwt$height    5.8663     0.8499   6.903 5.10e-10 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 32.7 on 98 degrees of freedom
Multiple R-squared:  0.3271,  Adjusted R-squared:  0.3203
F-statistic: 47.65 on 1 and 98 DF,  p-value: 5.102e-10
```

Estimate for β_1. For each 1 inch increase in height, weight increases by ~6 pounds.

Frequentist Approach

`lm` function (linear model)

```
model = lm(htwt$weight ~ htwt$height)

summary(model)

 Call
 lm(f

 Resi

-48.910 -22.624  -7.591  15.405 117.731

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -231.8232    56.7373  -4.086 8.99e-05 ***
htwt$height    5.8663     0.8499   6.903 5.10e-10 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 32.7 on 98 degrees of freedom
Multiple R-squared:  0.3271,  Adjusted R-squared:  0.3203
F-statistic: 47.65 on 1 and 98 DF,  p-value: 5.102e-10
```

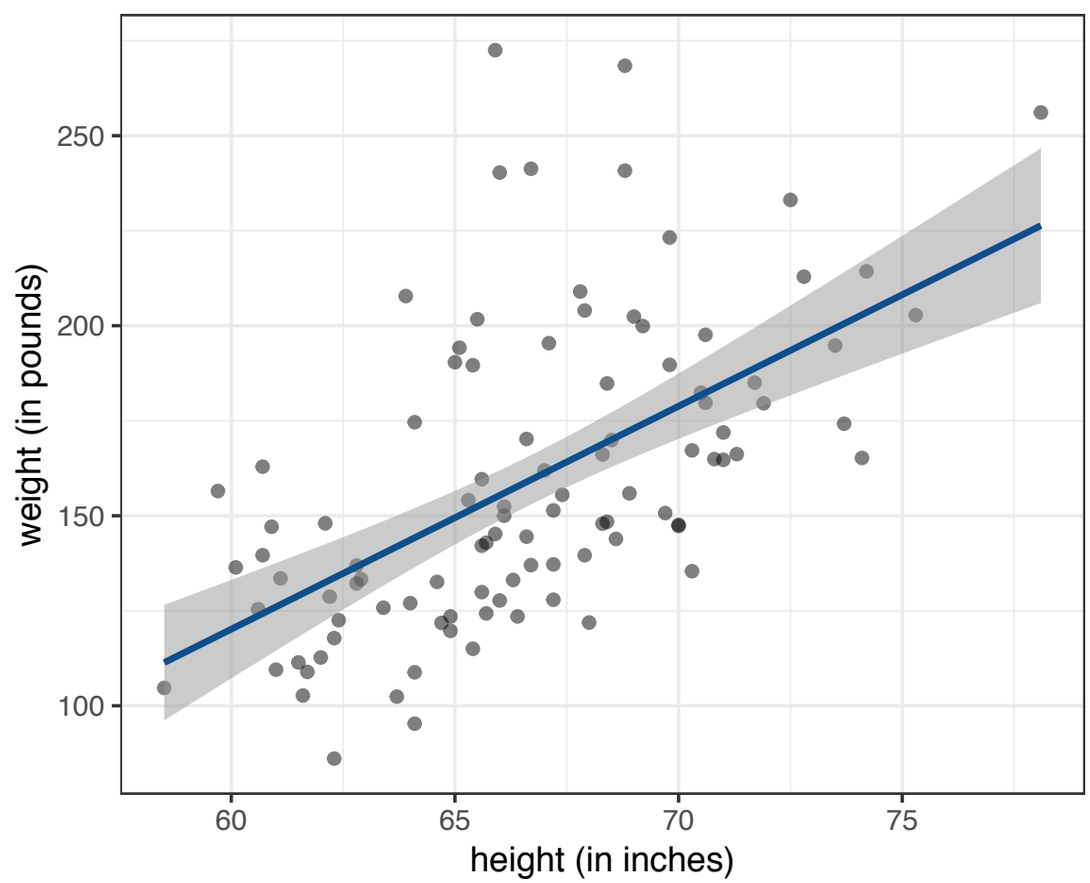
Slope is not zero (super duperly not zero!)

Frequentist Approach

`lm` function (linear model)

- Can plot regression line based on this model

```
ggplot(htwt, aes(x = height, y = weight)) +
  theme_bw() +
  geom_point(alpha = 0.5) +
  geom_smooth(method = lm, colour = "dodgerblue4", alpha = 0.5) +
  ylab("weight (in pounds)") +
  xlab("height (in inches)")
```



Bayesian Approach

Standardize the Data

- Markov Chain will perform much better if we standardize the data first
  - We're just re-scaling it, not changing it!!!
- Also makes for clearer choice of priors
  - Just need to convert it back to original scale prior to interpreting results!

$$z_{xi} = \frac{(x_i - \overline{x})}{\sigma_x}$$

- Mean will be 0, and sd will be ~1

Standardize the Data

- Markov Chain will perform much better if we standardize the data first
  - We're just re-scaling it, not changing it!!!
- Also makes for clearer choice of priors
  - Just need to convert it back to original scale prior to interpreting results!

each original x value

mean of x

each new standardized x value

$$z_{xi} = \frac{(x_i - \bar{x})}{\sigma_x}$$

sd of x

  - Mean will be 0, and sd will be ~1

Standardize the Data

- Should **always** standardize metric data before analyses

```
y = htwt$weight
yM = mean(y)
ySD = sd(y)
zy = (y - yM) / ySD
```

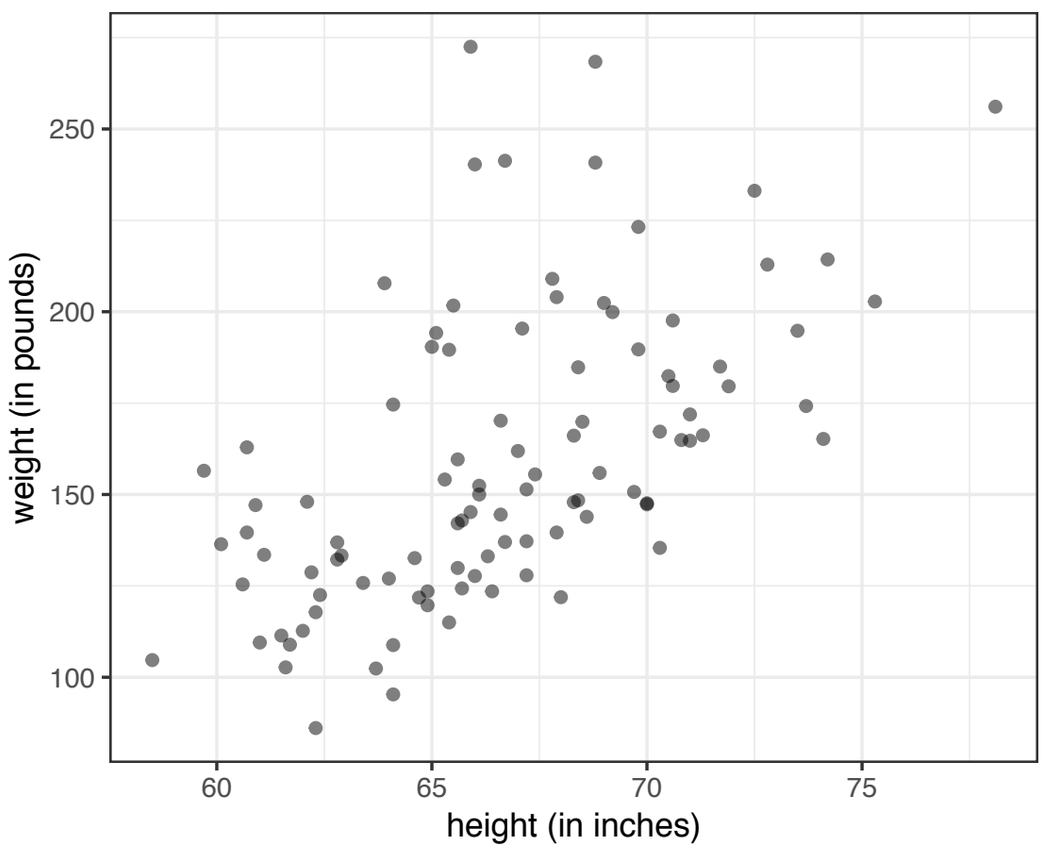
```
x = htwt$height
xM = mean(x)
xSD = sd(x)
zx = (x - xM) / xSD
```

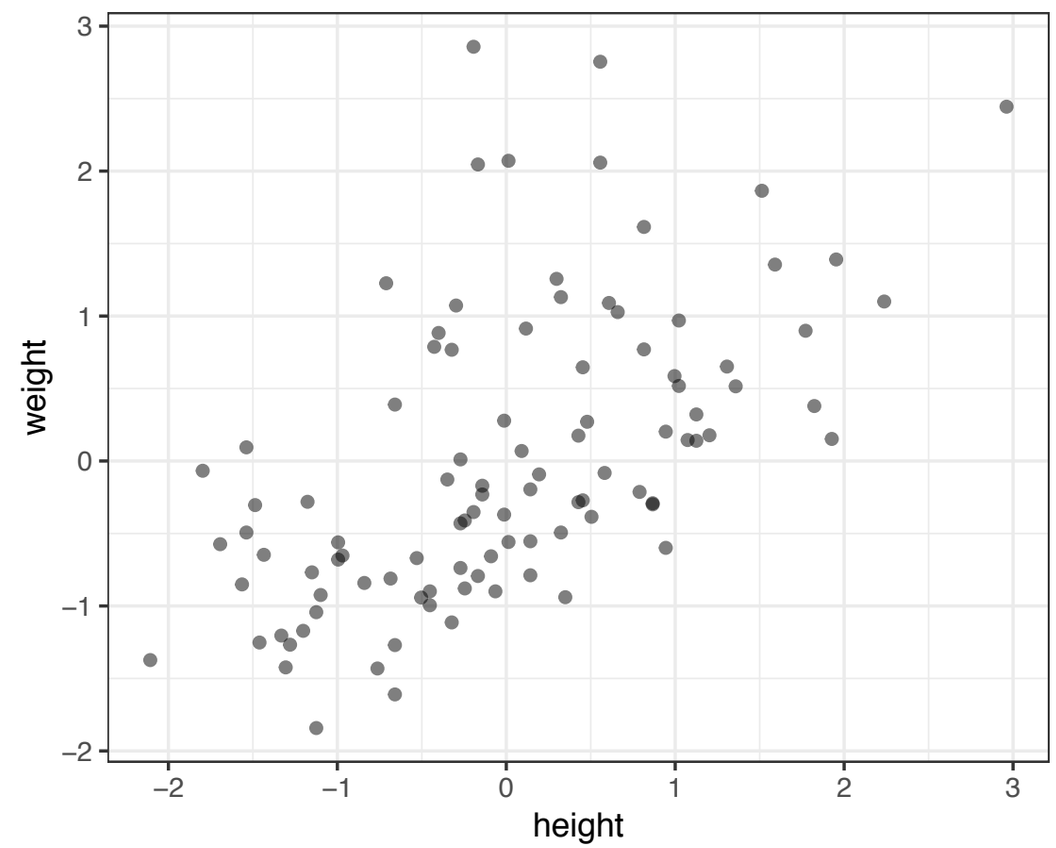
```
N = length(zy)
```

Standardize the Data

- Convince yourself this hasn't changed the structure of the data

```
data = data.frame(zy, zx)
ggplot(data, aes(x = zx, y = zy)) +
  theme_bw() +
  geom_point(alpha = 0.5) +
  ylab("weight") +
  xlab("height")
```





Analyses With Stan (or any MCMC process)

1. Prepare data for Stan
2. Build/define model
3. Run model
4. Assess MCMC process
5. Tentatively evaluate results
6. Conduct posterior predictive checks
7. Accept results or go back to step 2 to refine model

1. Prepare data for Stan

1. Prepare Data for Stan

Remember

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$

where

$$\text{mu} = \beta_0 + \beta_1 zx$$

1. Prepare Data for Stan

Remember

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$

Data *- the standardized weight data*

where

$$\text{mu} = \beta_0 + \beta_1 zx$$

Data *- the standardized height data*

1. Prepare Data for Stan

Remember

Parameter *- the sd around our predicted values*

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$

Data *- the standardized weight data*

where

$$\text{mu} = \beta_0 + \beta_1 zx$$

Data *- the standardized height data*

Parameter *- the "intercept": mean weight when height = 0*

Parameter *- the "slope": the mean increase in weight when height increases by 1 unit*

1. Prepare Data for Stan

```
dataList = list (
  y = zy,
  x = zx,
  N = N
)
```

2. Build/Define the Model

2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **data** block

```
data {
  int N;
  vector[N] y;
  vector[N] x;
}
```

2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$

These are no longer single values, but rather vectors of data (have to define their length, or any dimensions)

- The **data** block

```
data {
    int N;
    vector[N] y;
    vector[N] x;
}
```

2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$

Need to define `N` first, if it is used in other definitions!!!

- The **data** block

```
data {
  int N;
  vector[N] y;
  vector[N] x;
}
```

2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **parameters** block

```
parameters {
  real b0;
  real b1;
  real<lower=0> sigma;
}
```

2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **parameters** block

Standard deviations cannot be negative. Has cool implications for later!

```
parameters {
    real b0;
    real b1;
    real<lower=0> sigma;
  }
```

2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **model** block

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    mu = b0 + (b1 * x);
    y ~ normal(mu, sigma);

    // Priors
    b0 ~ ????
    b1 ~ ????
    sigma ~ ????
}
```

2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **model** block

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    mu = b0 + (b1 * x);
    y ~ normal(mu, sigma);

    // Priors
    b0 ~ ????
    b1 ~ ????
    sigma ~ ????
}
```

We are *calculating* `mu`, but before we do, we have to define it for Stan.

Note that it is **not** a **parameter** that we are estimating, so it does not belong in the parameters block.

2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **model** block

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    mu = b0 + (b1 * x);
    y ~ normal(mu, sigma);

    // Priors
    b0 ~ ????
    b1 ~ ????
    sigma ~ ????
}
```

What seems appropriate?

2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **model** block

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    mu = b0 + (b1 * x);
    y ~ normal(mu, sigma);

    // Priors
    b0 ~ normal(0, 1);
    b1 ~ ????
    sigma ~ ????
}
```

2. Build/Define The Model

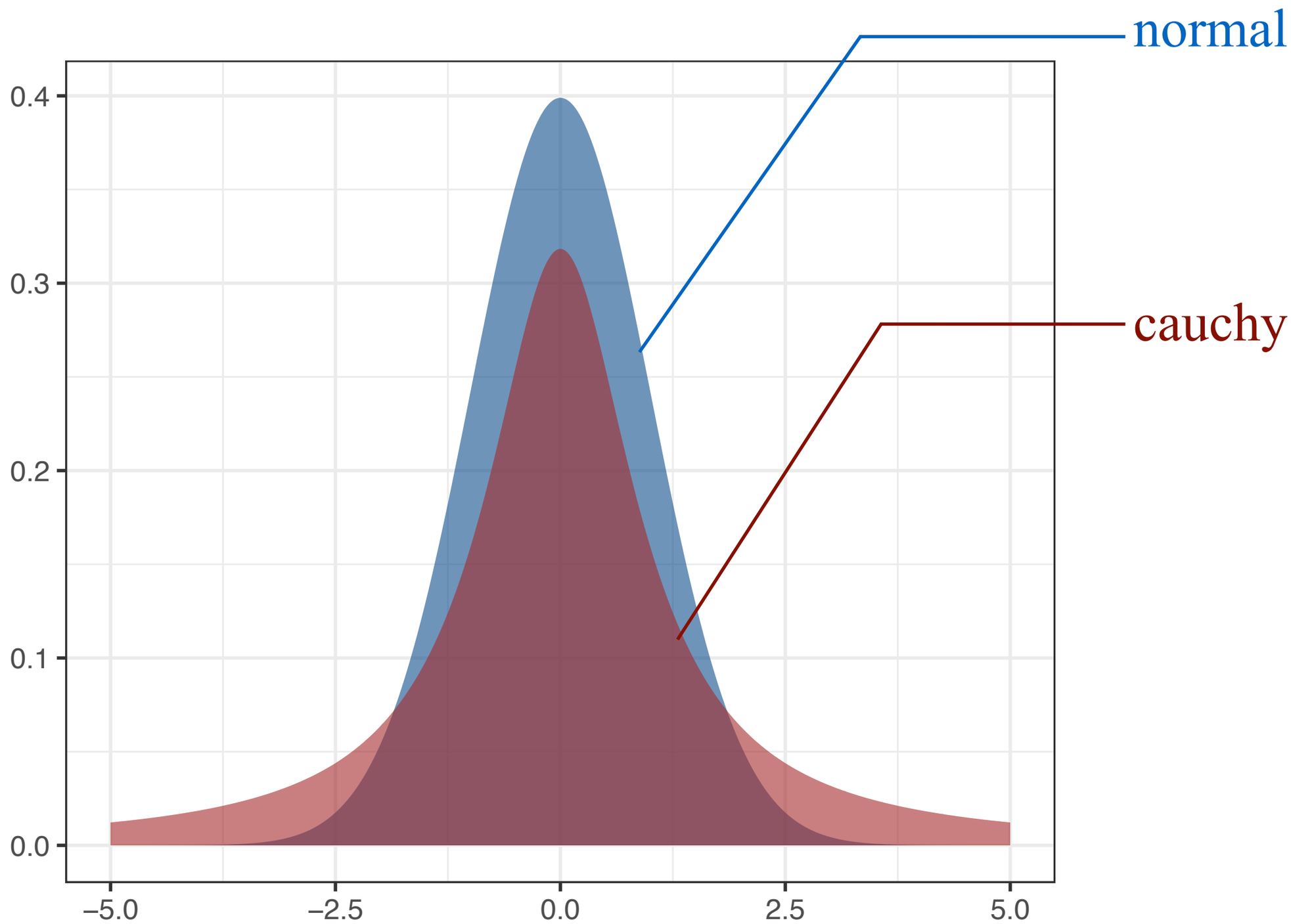
$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **model** block

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    mu = b0 + (b1 * x);
    y ~ normal(mu, sigma);

    // Priors
    b0 ~ normal(0, 1);
    b1 ~ ????
    sigma ~ ????
}
```

What seems appropriate?

2. Build/Define The Model



2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **model** block

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    mu = b0 + (b1 * x);
    y ~ normal(mu, sigma);

    // Priors
    b0 ~ normal(0, 1);
    b1 ~ cauchy(0, 1);
    sigma ~ ????
}
```

2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **model** block

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    mu = b0 + (b1 * x);
    y ~ normal(mu, sigma);

    // Priors
    b0 ~ normal(0, 1);
    b1 ~ cauchy(0, 1);
    sigma ~ ????
}
```

What seems appropriate?

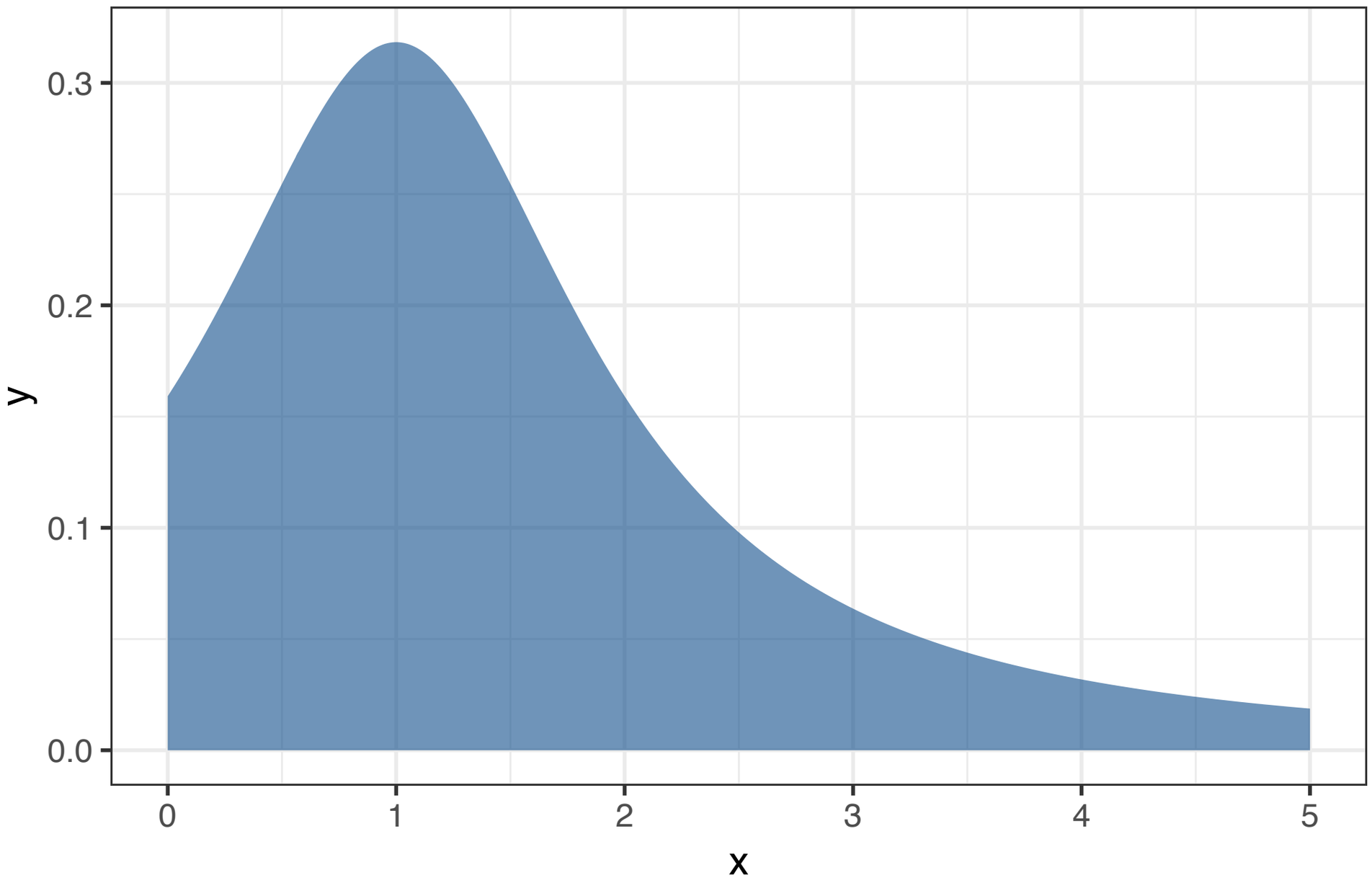
2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **model** block

```
model {
    // Definitions
    vector[N] mu;

    // Likelihood
    mu = b0 + (b1 * x);
    y ~ normal(mu, sigma);

    // Priors
    b0 ~ normal(0, 1);
    b1 ~ cauchy(0, 1);
    sigma ~ cauchy(1, 1);
}
```

2. Build/Define The Model



2. Build/Define The Model

$$zy \sim \text{normal}(\text{mu}, \text{sigma})$$
$$\text{mu} = \beta_0 + \beta_1 zx$$

- The **generated quantities** block

```
generated quantities {
  vector[N] y_signal;
  vector[N] y_pred;

  for (n in 1:N) {
      y_signal[n] = b0 + (b1 * x[n]);
      y_pred[n] = normal_rng(b0 + (b1 * x[n]), sigma);
}
```

2. Build/Define The Model

zy ~ normal(mu, sigma)

mu

For each step in the chain, and for each `x` value, generate a `y` value based on deterministic part of model. Useful for plotting/evaluating uncertainty around deterministic component.

- The **generated quantities** bl

```
generated quantities {
  vector[N] y_signal;
  vector[N] y_pred;

  for (n in 1:N) {
      y_signal[n] = b0 + (b1 * x[n]);
      y_pred[n] = normal_rng(b0 + (b1 * x[n]), sigma);
}
```

2. Build/Define The Model

$zy \sim$ norm

mu

For each step in the chain, and for each `x` value, generate a `y` value based on full model (deterministic + stochastic). Useful for plotting/evaluating how well model captures the data.

- The **generated quantities** bl

```
generated quantities {
  vector[N] y_signal;
  vector[N] y_pred;

  for (n in 1:N) {
      y_signal[n] = b0 + (b1 * x[n]);
      y_pred[n] = normal_rng(b0 + (b1 * x[n]), sigma);
}
```

```
modelString = "
  data {
    int N;
    vector[N] y;
    vector[N] x;
  }

  parameters {
    real b0;
    real b1;
    real<lower=0> sigma;
  }

  model {
    // Definitions
    vector[N] mu;

    // Likelihood
    mu = b0 + (b1 * x);
    y ~ normal(mu, sigma);

    // Priors
    b0 ~ normal(0, 1);
    b1 ~ cauchy(0, 1);
    sigma ~ cauchy(1, 1);
  }

  generated quantities {
    vector[N] y_signal;
    vector[N] y_pred;

    for (n in 1:N) {
        y_signal[n] = b0 + (b1 * x[n]);
        y_pred[n] = normal_rng(b0 + (b1 * x[n]), sigma);
  }
"
writeLines(modelString, con="model.stan")
```

3. Run the Model

3. Run The Model

```
stanFit <- stan(file = "model.stan",
                data = dataList,
                pars = c("b0", "b1", "sigma", “y_signal", "y_pred"),
                warmup = 2000,
                iter = 7000,
                chains = 3)
```

4. Assess Performance of MCMC Process

4. Assess MCMC Process

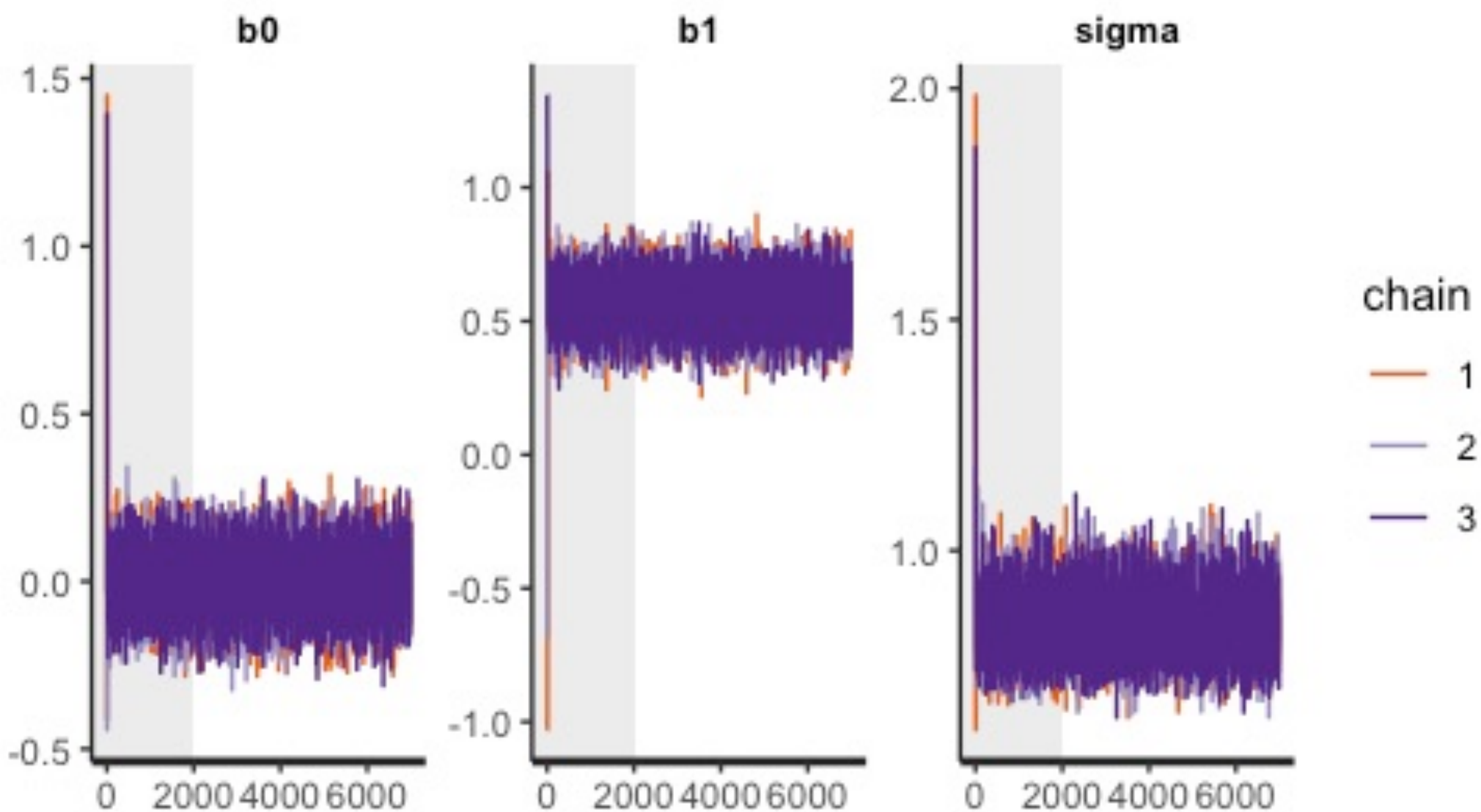
- Check Rhat and effective sample size stats

```
print(stanFit)

Inference for Stan model: model.
3 chains, each with iter=7000; warmup=2000; thin=1;
post-warmup draws per chain=5000, total post-warmup draws=15000.

                 mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
b0               0.00    0.00 0.08  -0.16  -0.06   0.00   0.06   0.17 17259    1
b1               0.57    0.00 0.08   0.40   0.51   0.57   0.62   0.73 16430    1
sigma            0.84    0.00 0.06   0.73   0.79   0.83   0.87   0.97 14591    1
y_signal[1]     -0.55    0.00 0.12  -0.78  -0.63  -0.55  -0.47  -0.32 17148    1
y_signal[2]     -0.65    0.00 0.13  -0.90  -0.74  -0.65  -0.57  -0.40 17091    1
y_signal[3]     -0.67    0.00 0.13  -0.92  -0.75  -0.67  -0.58  -0.41 17083    1
...
```

4. Assess MCMC Process

- Check trace plots

```
stan_trace(stanFit, pars = c("b0", "b1", "sigma"), inc_warmup = TRUE)
```



5. Tentatively Interpret Results

5. Tentatively Interpret Results

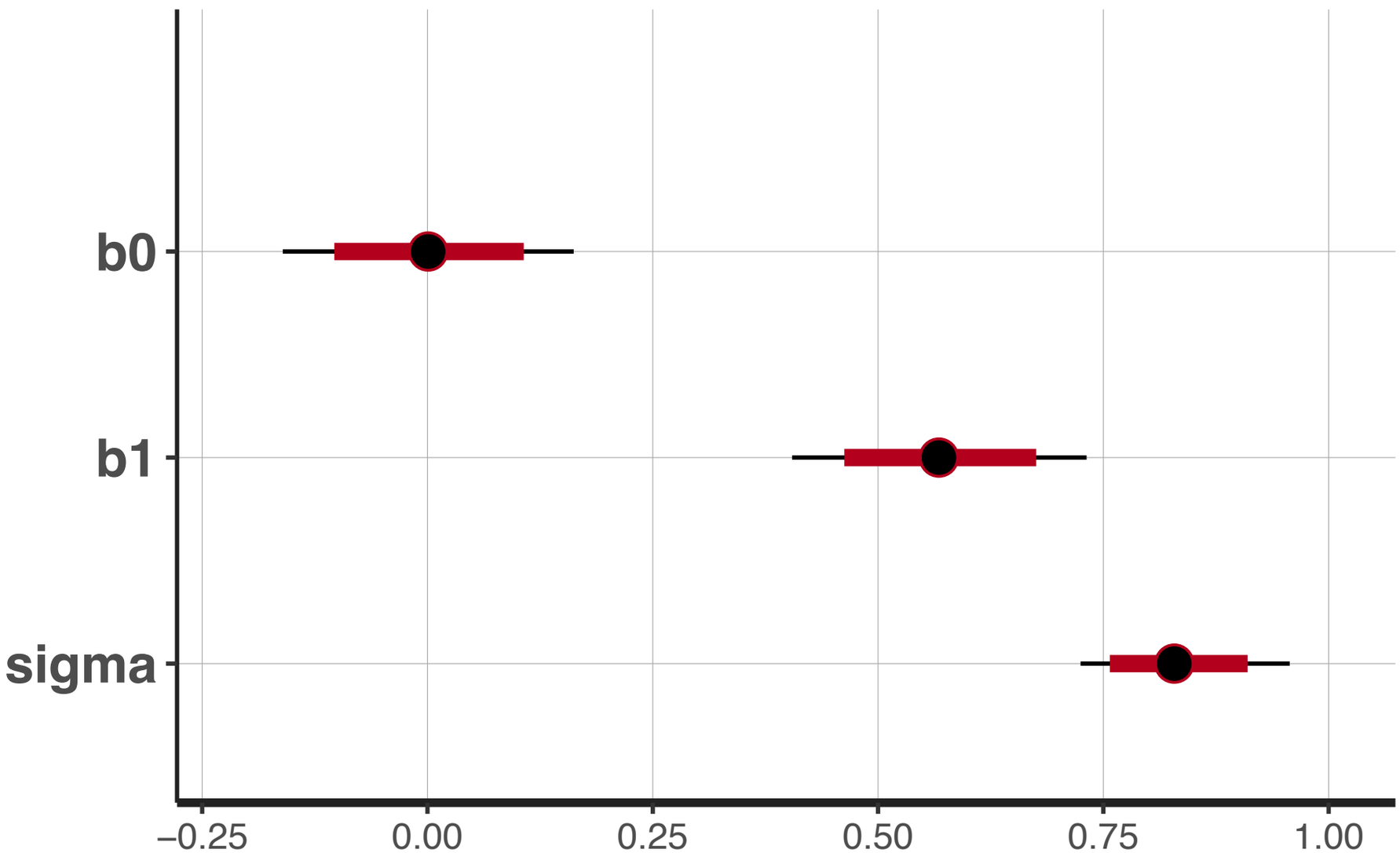
View stats

```
print(stanFit)

Inference for Stan model: model.
3 chains, each with iter=7000; warmup=2000; thin=1;
post-warmup draws per chain=5000, total post-warmup draws=15000.

                mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
b0              0.00    0.00 0.08  -0.16  -0.06   0.00   0.06   0.17 17259    1
b1              0.57    0.00 0.08   0.40   0.51   0.57   0.62   0.73 16430    1
sigma           0.84    0.00 0.06   0.73   0.79   0.83   0.87   0.97 14591    1
y_signal[1]    -0.55    0.00 0.12  -0.78  -0.63  -0.55  -0.47  -0.32 17148    1
y_signal[2]    -0.65    0.00 0.13  -0.90  -0.74  -0.65  -0.57  -0.40 17091    1
y_signal[3]    -0.67    0.00 0.13  -0.92  -0.75  -0.67  -0.58  -0.41 17083    1...
```

5. Tentatively Interpret Results

Plot with `rstan` functions

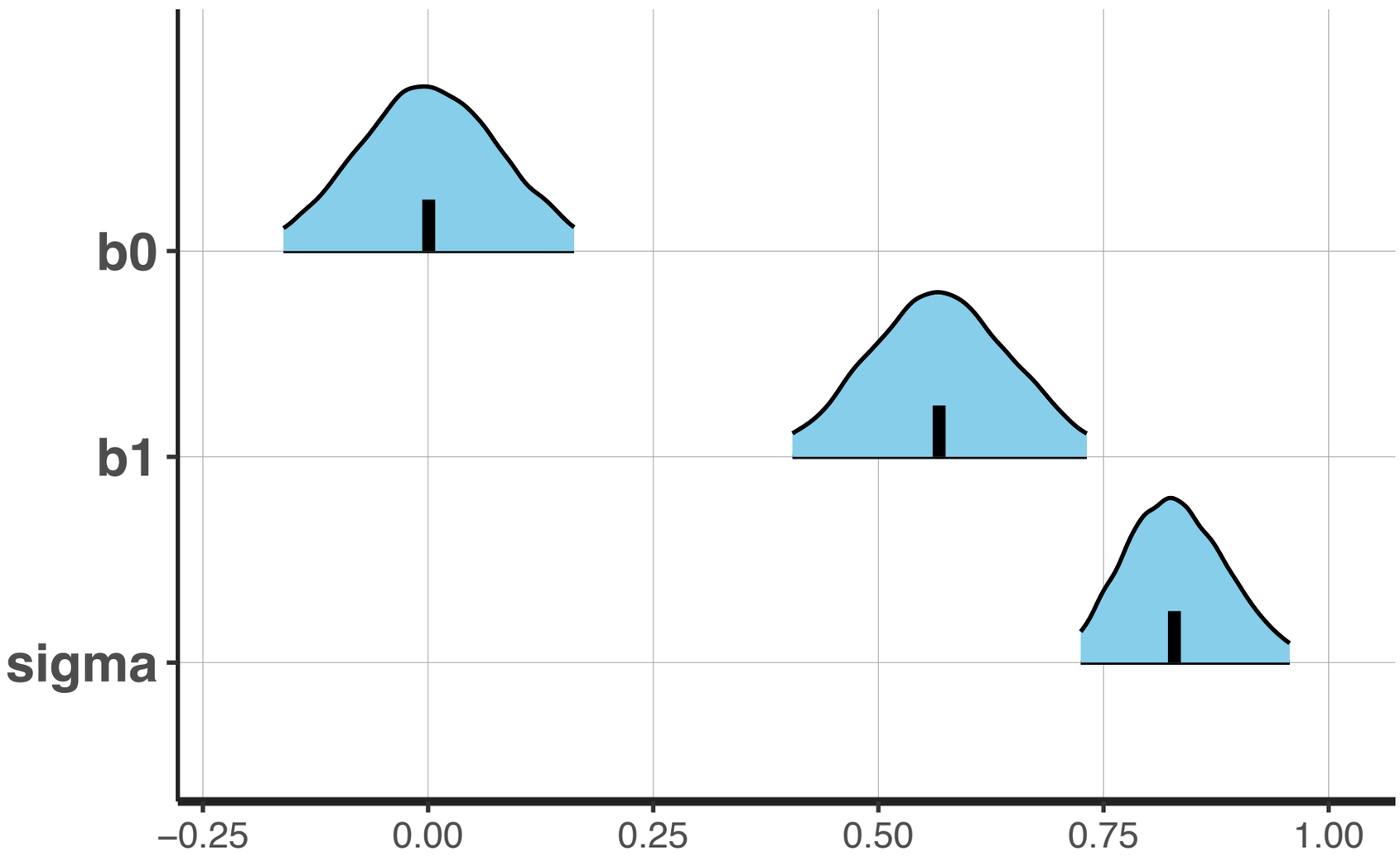
```
stan_plot(stanFit, par = c("b0", "b1", "sigma"))
```



5. Tentatively Interpret Results

Plot with `rstan` functions

```
plot(stanFit, par = c("b0", "b1", "sigma"), show_density = TRUE,
    ci_level = 0.95, fill_color = "skyblue")
```



5. Tentatively Interpret Results

Make custom plots (base R)

- Extract the data as a data frame and take a look at it's structure

```
mcmcChains = as.data.frame(stanFit)
head(mcmcChains)

           b0        b1     sigma y_signal[1] y_signal[2] y_signal[3] y_signal[4] ...
1 -0.01503022 0.6156512 0.8442957  -0.6119446  -0.7233983  -0.7393203  -0.8189301 ...
2  0.09802777 0.3644079 0.9305011  -0.2552896  -0.3212598  -0.3306841  -0.3778056 ...
3 -0.05129244 0.4105070 0.8049662  -0.4493059  -0.5236216  -0.5342381  -0.5873208 ...
4  0.01711922 0.6390345 0.9414948  -0.6024668  -0.7181537  -0.7346804  -0.8173139 ...
```

5. Tentatively Interpret Results

Make custom plots (base R)

- Separate out the desired parameters

```
zb0 = mcmcChains$b0
zb1 = mcmcChains$b1
zsigma = mcmcChains$sigma

ysignal = mcmcChains[, 4:103]

ypred = mcmcChains[, 104:203]
```

5. Tentatively Interpret Results

Make custom plots (base R)

- Convert back to original scale*

```
sigma = zsigma * ySD
b1 = zb1 * ySD / xSD
b0 = ((zb0 * ySD) + yM - (zb1 * ySD * xM / xSD))
```

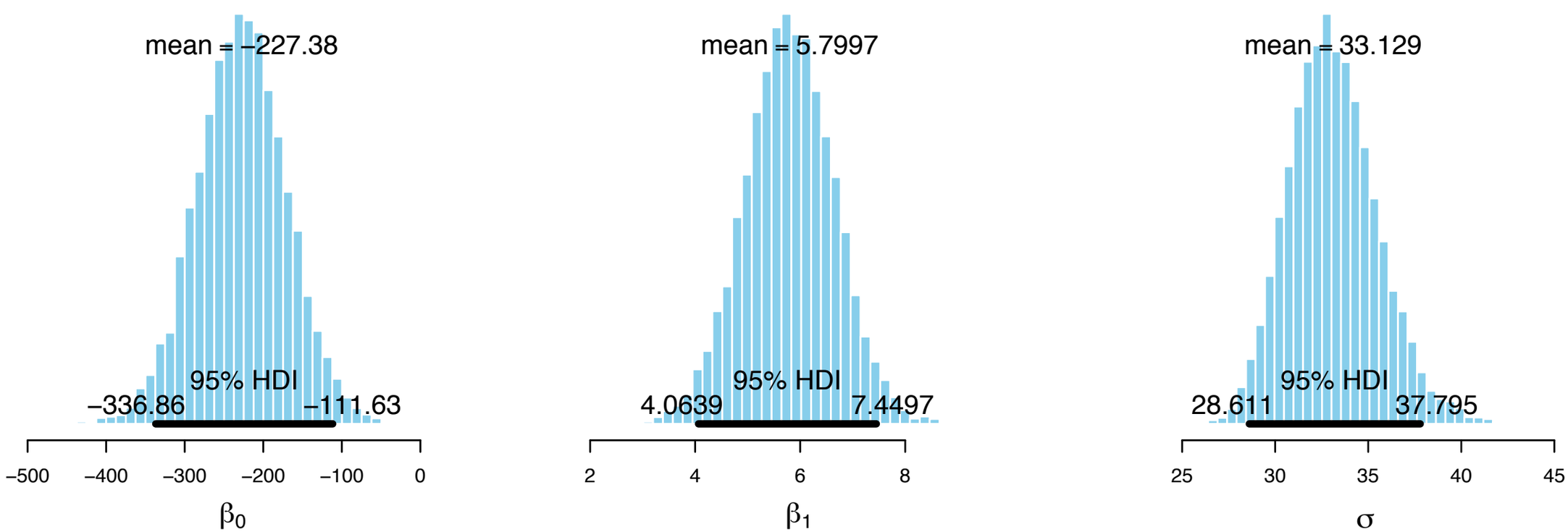
* See Kruschke (2015) pp. 485-487 for explanation

5. Tentatively Interpret Results

Make custom plots (base R)

- Plot the posteriors

```
source("plotPost.R")
par(mfrow = c(1, 3))
histInfo = plotPost(b0, xlab = bquote(beta[0]))
histInfo = plotPost(b1, xlab = bquote(beta[1]))
histInfo = plotPost(sigma, xlab = bquote(sigma))
```



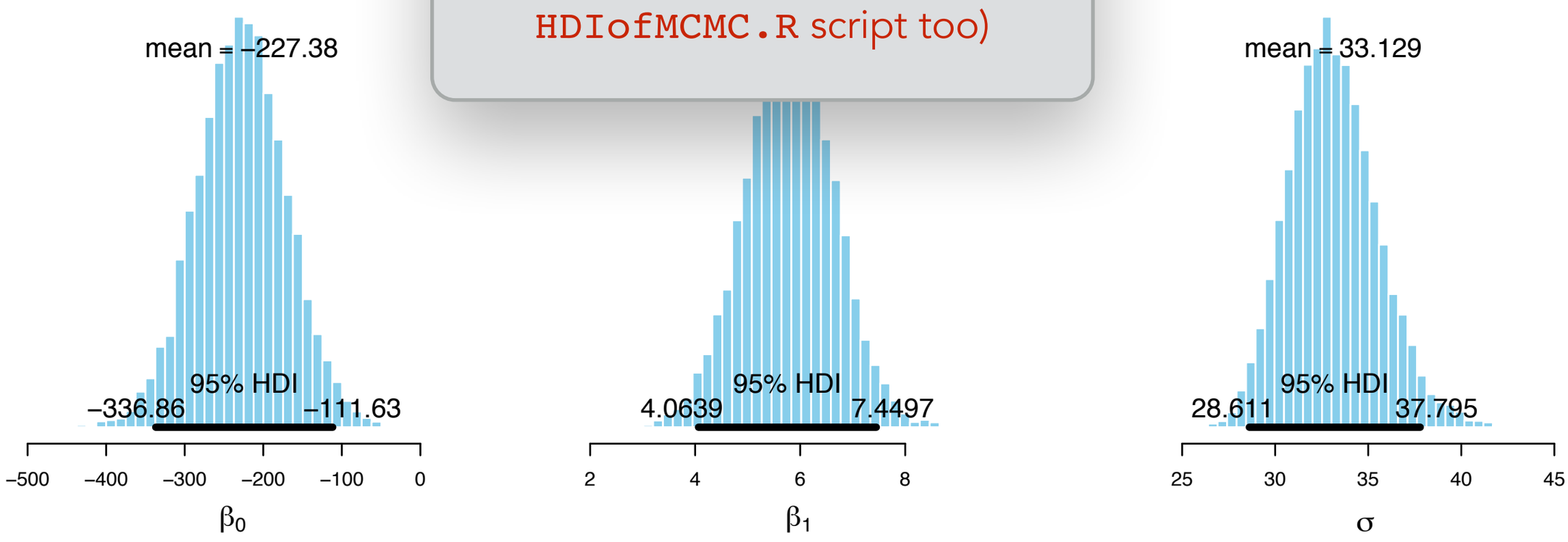
5. Tentatively Interpret Results

Make custom plots (base R)

- Plot the posteriors

```
source("plotPost.R")
par(mfrow = c(1, 3))
histInfo = plotPost(b0, xlab = bquote(beta[0]))
histInfo = plotPost(b1, xlab = bquote(beta[1]))
histInfo = plotPost(sigma, xlab = bquote(sigma))
```

A plotting function from the textbook (requires the `HDIofMCMC.R` script too)



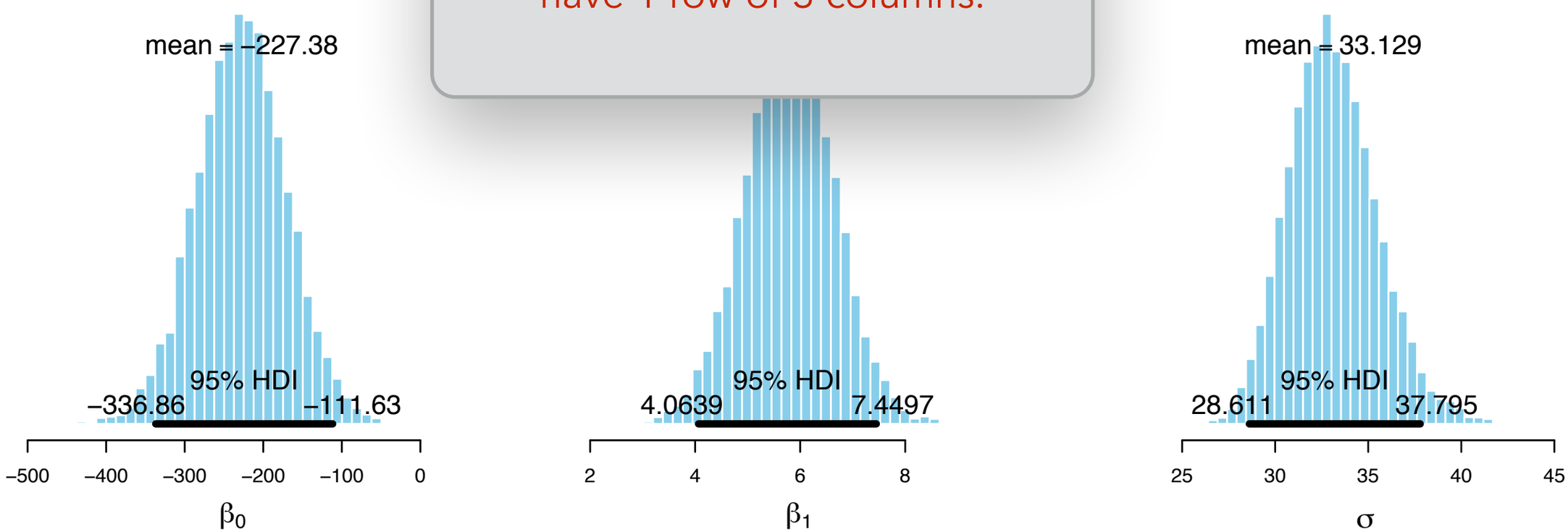
5. Tentatively Interpret Results

Make custom plots (base R)

- Plot the posteriors

```
source("plotPost.R")
par(mfrow = c(1, 3))
histInfo = plotPost(b0, xlab = bquote(beta[0]))
histInfo = plotPost(b1, xlab = bquote(beta[1]))
histInfo = plotPost(sigma, xlab = bquote(sigma))
```

Set the plotting window. Here we have 1 row of 3 columns.



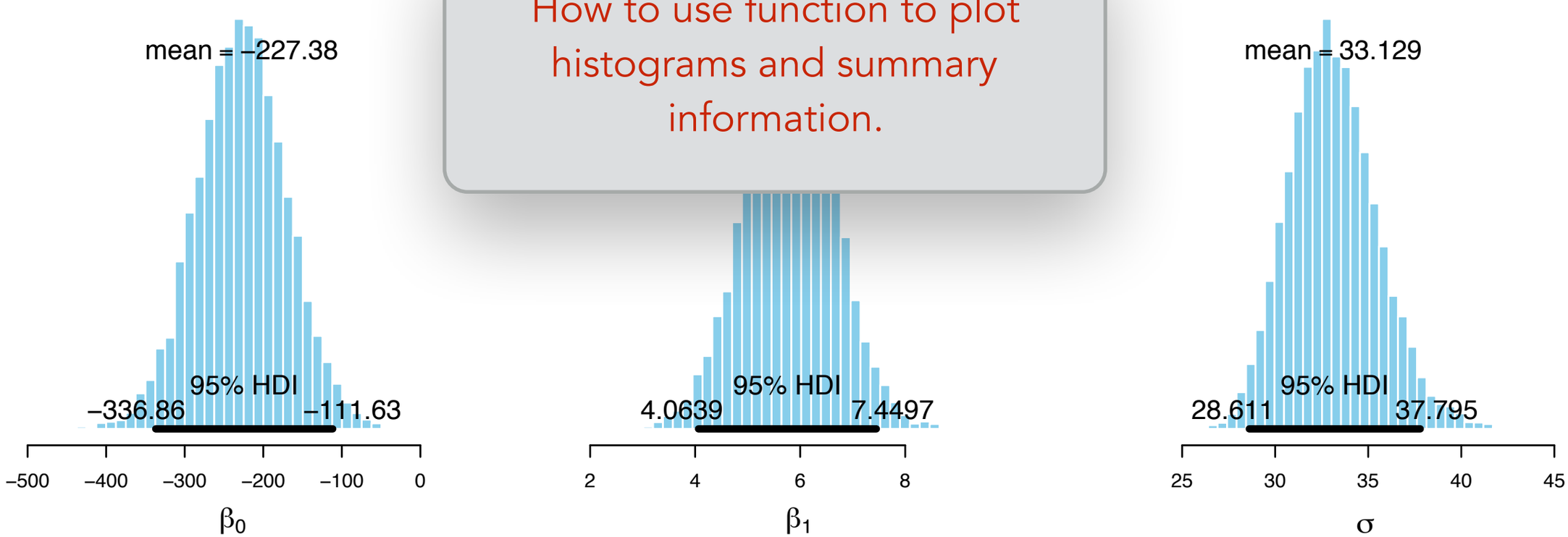
5. Tentatively Interpret Results

Make custom plots (base R)

- Plot the posteriors

```
source("plotPost.R")
par(mfrow = c(1, 3))
histInfo = plotPost(b0, xlab = bquote(beta[0]))
histInfo = plotPost(b1, xlab = bquote(beta[1]))
histInfo = plotPost(sigma, xlab = bquote(sigma))
```

How to use function to plot histograms and summary information.



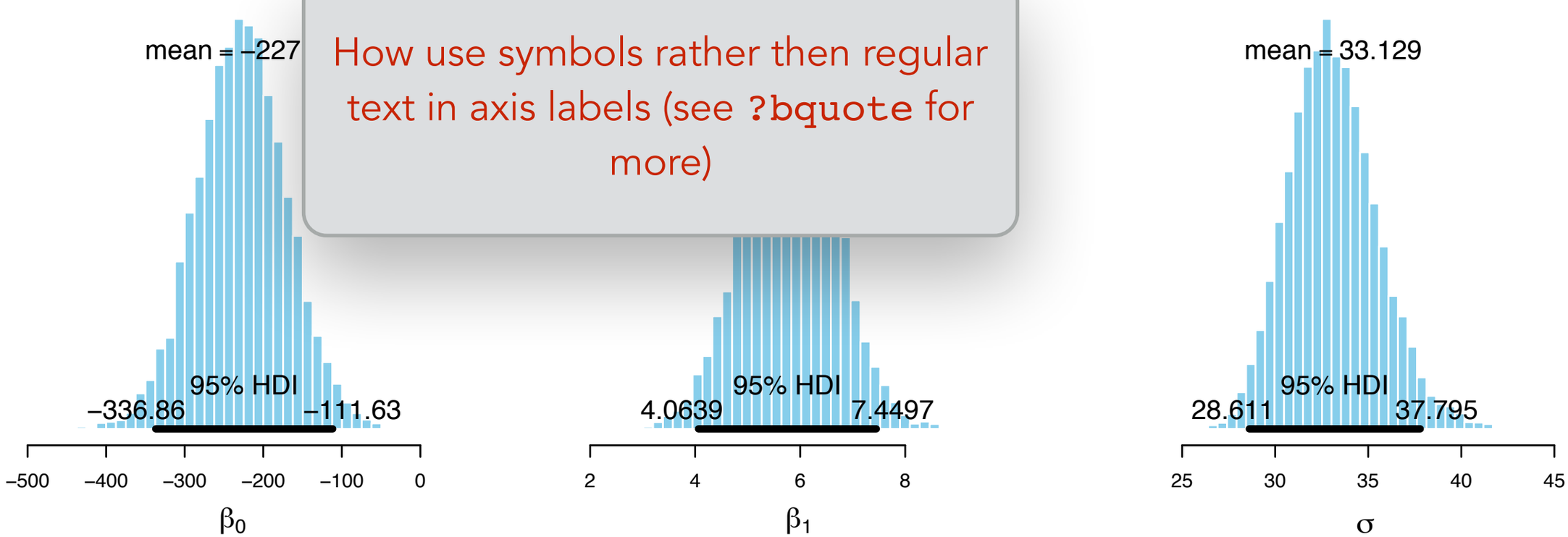
5. Tentatively Interpret Results

Make custom plots (base R)

- Plot the posteriors

```
source("plotPost.R")
par(mfrow = c(1, 3))
histInfo = plotPost(b0, xlab = bquote(beta[0]))
histInfo = plotPost(b1, xlab = bquote(beta[1]))
histInfo = plotPost(sigma, xlab = bquote(sigma))
```

How use symbols rather then regular text in axis labels (see `?bquote` for more)



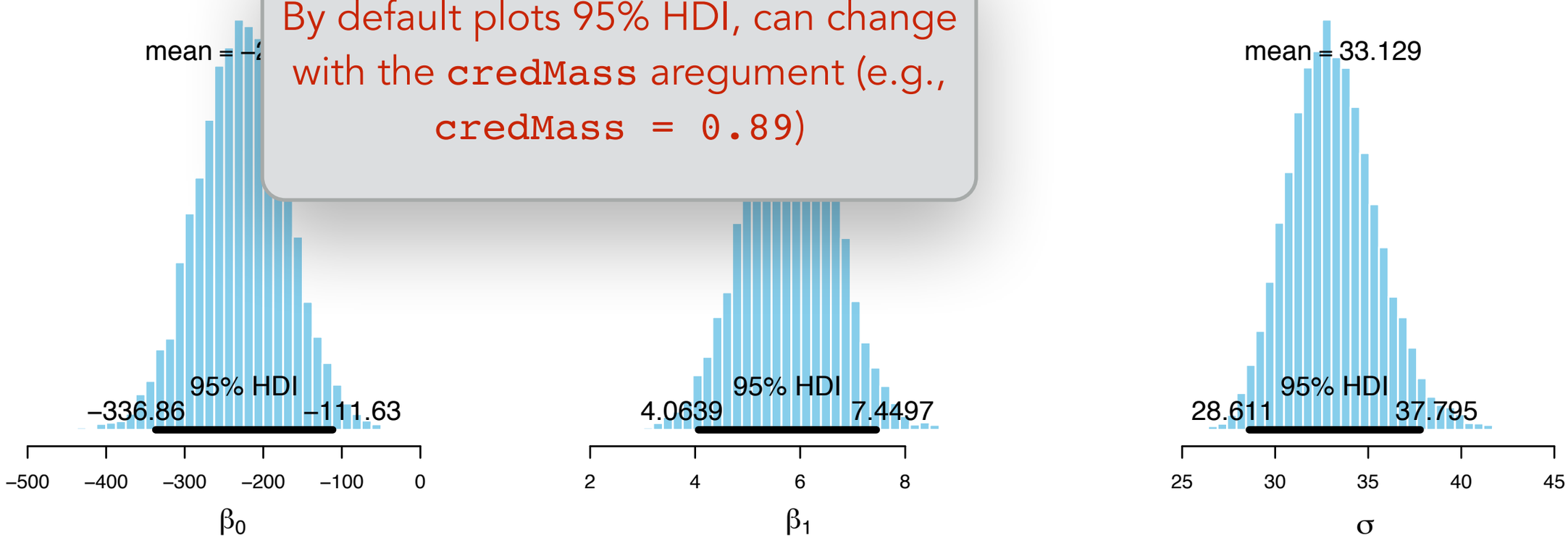
5. Tentatively Interpret Results

Make custom plots (base R)

- Plot the posteriors

```
source("plotPost.R")
par(mfrow = c(1, 3))
histInfo = plotPost(b0, xlab = bquote(beta[0]))
histInfo = plotPost(b1, xlab = bquote(beta[1]))
histInfo = plotPost(sigma, xlab = bquote(sigma))
```

By default plots 95% HDI, can change with the `credMass` aregument (e.g., `credMass = 0.89`)



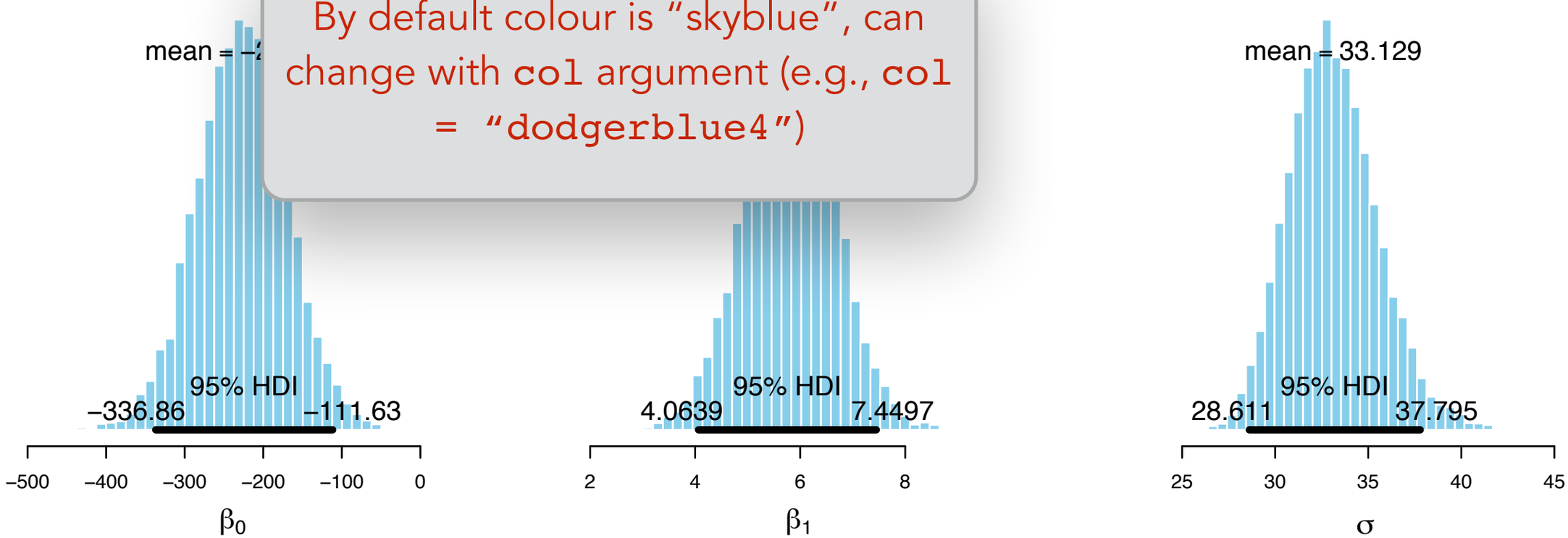
5. Tentatively Interpret Results

Make custom plots (base R)

- Plot the posteriors

```
source("plotPost.R")
par(mfrow = c(1, 3))
histInfo = plotPost(b0, xlab = bquote(beta[0]))
histInfo = plotPost(b1, xlab = bquote(beta[1]))
histInfo = plotPost(sigma, xlab = bquote(sigma))
```

By default colour is "skyblue", can change with `col` argument (e.g., `col = "dodgerblue4"`)



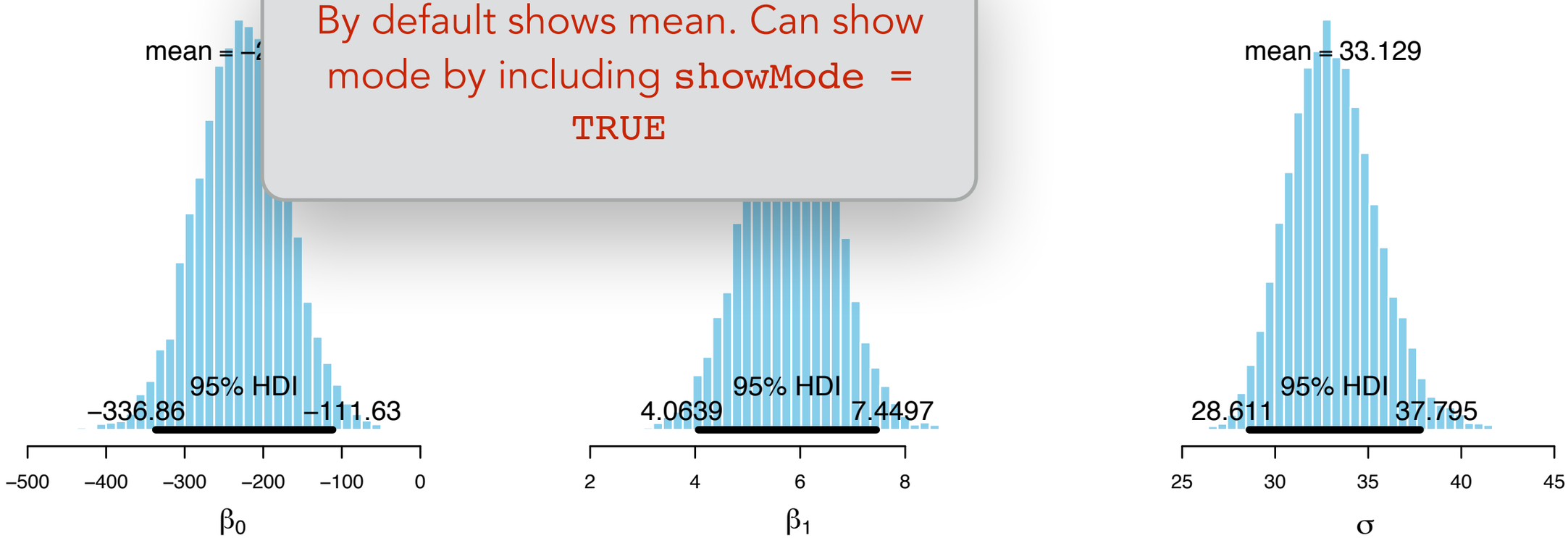
5. Tentatively Interpret Results

Make custom plots (base R)

- Plot the posteriors

```
source("plotPost.R")
par(mfrow = c(1, 3))
histInfo = plotPost(b0, xlab = bquote(beta[0]))
histInfo = plotPost(b1, xlab = bquote(beta[1]))
histInfo = plotPost(sigma, xlab = bquote(sigma))
```

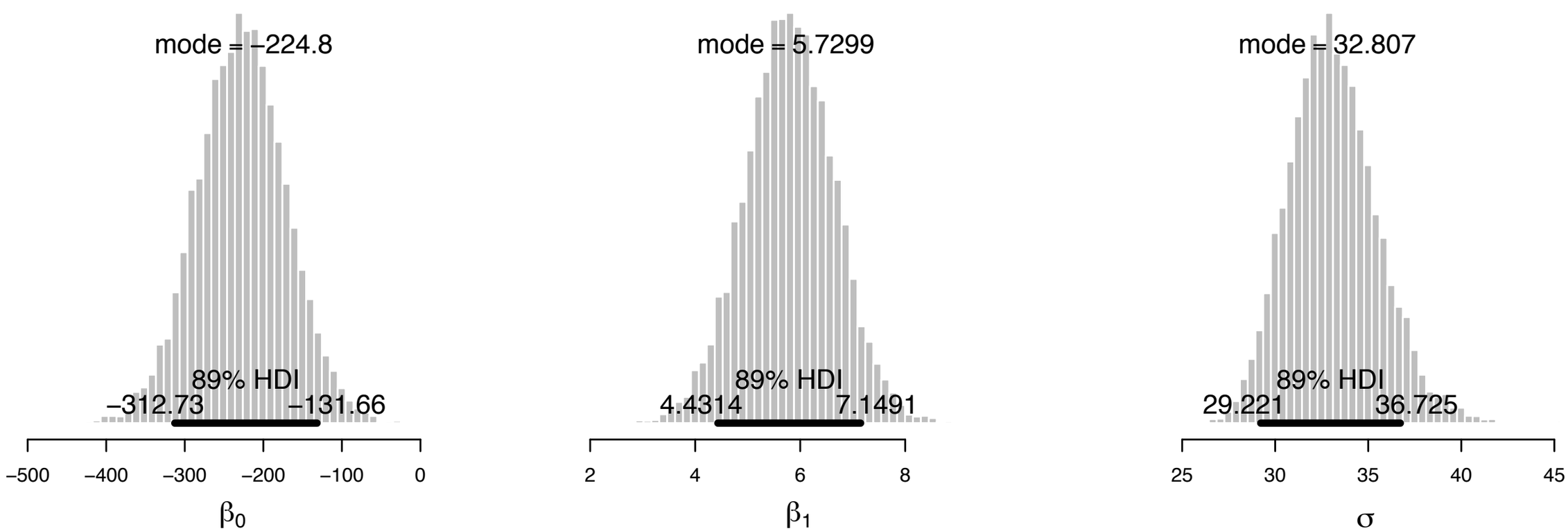
By default shows mean. Can show mode by including `showMode = TRUE`



5. Tentatively Interpret Results

Make custom plots (base R)

```
histInfo = plotPost(b0, xlab = bquote(beta[0]), credMass = 0.89, col =
  "gray", showMode = TRUE)
histInfo = plotPost(b1, xlab = bquote(beta[1]), credMass = 0.89, col =
  "gray", showMode = TRUE)
histInfo = plotPost(sigma, xlab = bquote(sigma), credMass = 0.89, col =
  “gray”, showMode = TRUE)
```



Colours in R

- Can use colours by name
- Can also use RGB scale

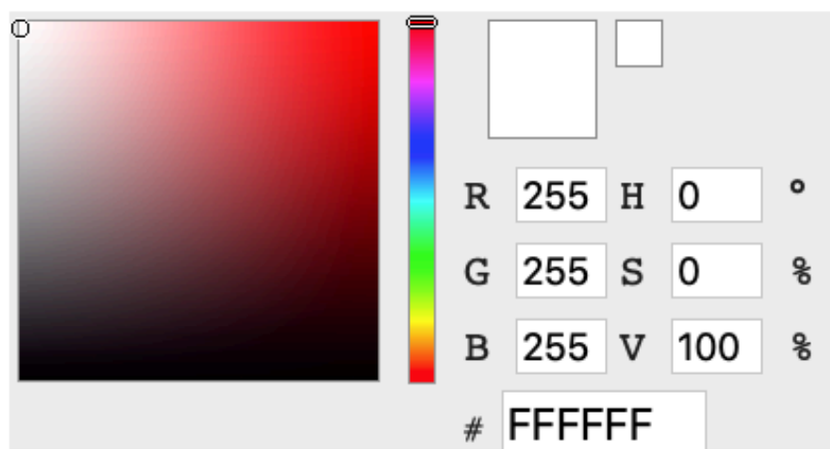
```
rgb(red, green, blue, alpha)
```

Each on a scale of 0 to 1

- Real RGB scale is 0 to 255, so have to scale appropriately

Colours in R

RGB color picker

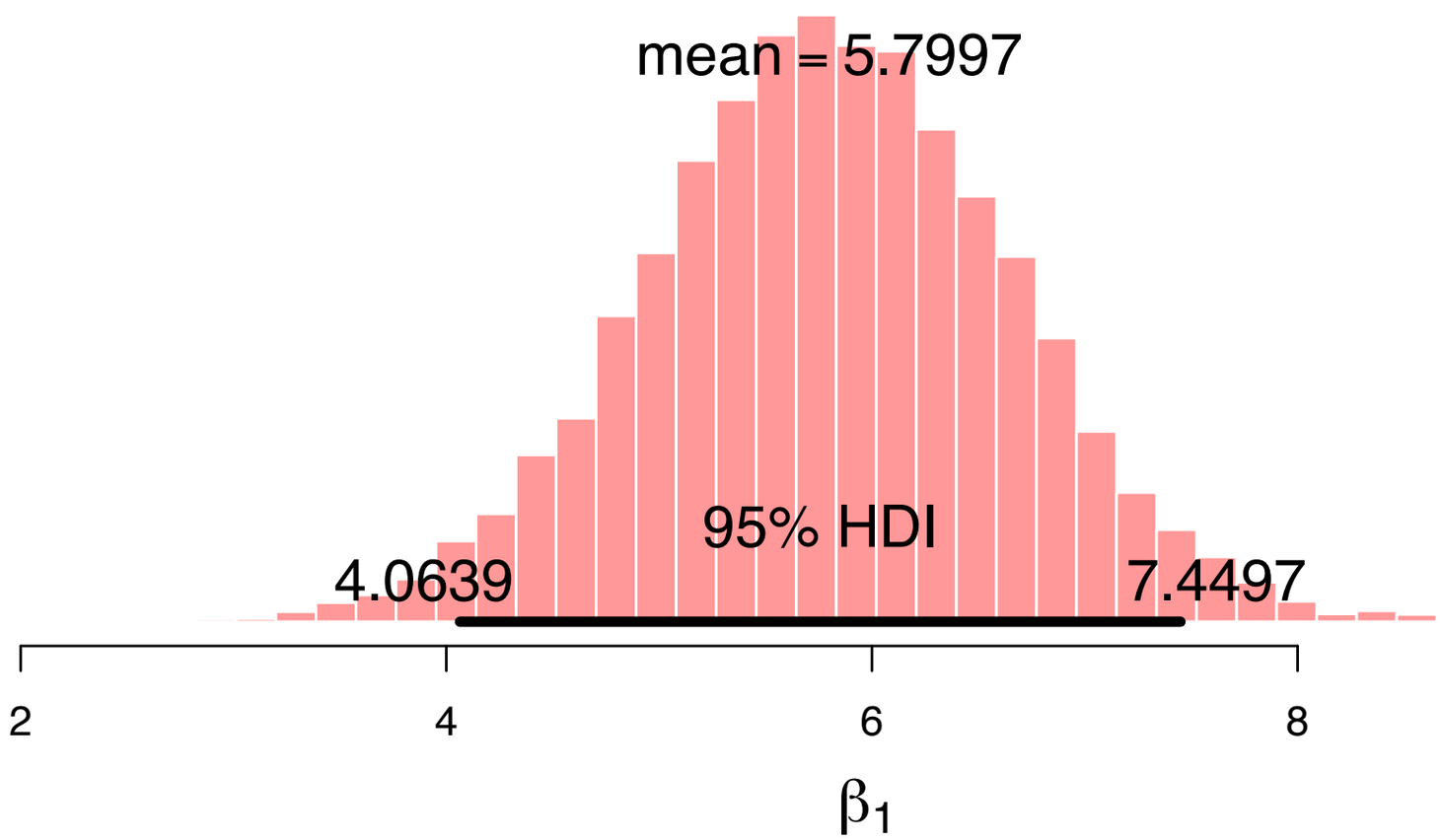


RGB color codes chart

Hover with cursor on color to get the hex and decimal color codes below:

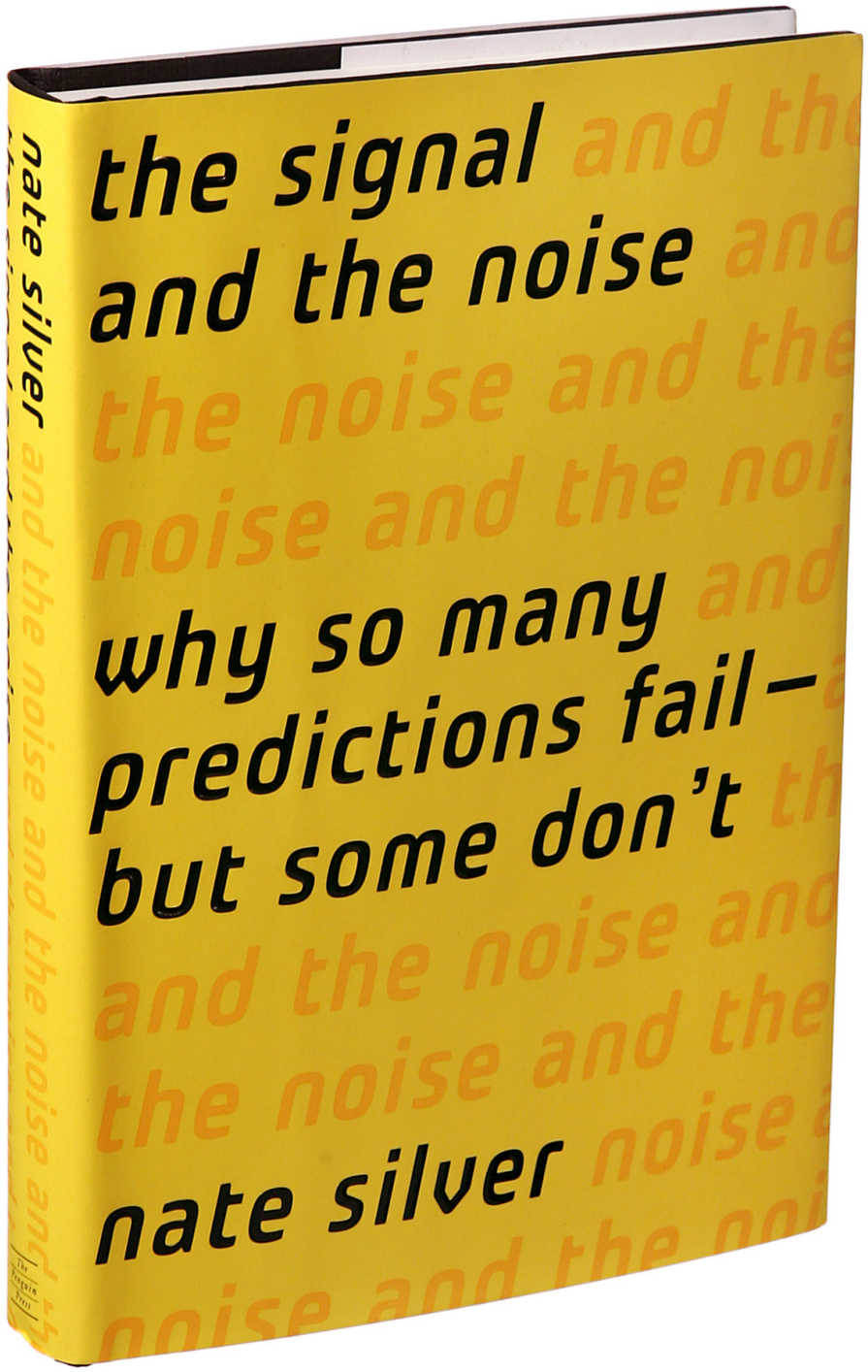
Colours in R

```
par(mfrow = c(1, 1))
histInfo = plotPost(b1, xlab = bquote(beta[1]), col = rgb(1, 0.2, 0.2,
  0.5))
```



6. Conduct Posterior Predictive Checks:

Evaluating the signal and the noise


the signal
and the noise
why so many
predictions fail—
but some don't
nate silver
nate silver and the noise

6. Posterior Predictive Checks
Deterministic Component

- Our generated quantity, `y_signal`, has a predicted value for `y` (weight) for each `x` value (height) for every step of the chain (using the b0 and b1 values for each step)

`y_signal[x]` where x = 1-100

predicted y for each step

y_sim[1]	y_sim[2]	y_sim[3]	y_sim[4]	y_sim[5]	y_sim[6]	y_sim[7]
-0.3576661	-0.4473600	-0.4601734	-0.5242405	0.42395217	0.283004607	-0.011703928
-0.5849479	-0.6854168	-0.6997695	-0.7715329	0.29056643	0.132686796	-0.197425168
-0.4048725	-0.5245287	-0.5416225	-0.6270912	0.63784633	0.449815075	0.056658809
-0.7121148	-0.7971175	-0.8092607	-0.8699769	0.02862265	-0.104952964	-0.384247421
-0.7554739	-0.8548864	-0.8690882	-0.9400972	0.11083519	-0.045384480	-0.372025618
-0.6715561	-0.7691671	-0.7831116	-0.8528337	0.17905425	0.025665497	-0.295056441
-0.5293173	-0.6208777	-0.6339578	-0.6993581	0.26856631	0.124685658	-0.176155710
-0.6874285	-0.8076957	-0.8248768	-0.9107820	0.36061480	0.171623392	-0.223540470
-0.4986728	-0.6084201	-0.6240983	-0.7024892	0.45769665	0.285236593	-0.075361715
-0.6309496	-0.7307121	-0.7449639	-0.8162228	0.23840953	0.081639855	-0.246151276
-0.6100997	-0.7439155	-0.7630321	-0.8586148	0.55600981	0.345727774	-0.093952858
-0.5936546	-0.7190449	-0.7369578	-0.8265223	0.49903220	0.301990317	-0.110006342
-0.4414907	-0.5452086	-0.5600254	-0.6341096	0.46233657	0.299351323	-0.041436007

6. Posterior Predictive Checks

Deterministic Component

- Can plot these, and their 95% HDI (or whatever) against the data to see how it fits

- `ysignal` is on the standardized scale!!!

6. Posterior Predictive Checks
Deterministic Component

- Calculate desired summary statistics

```
signalMeans = apply(ysignal, 2, mean)
```

6. Posterior Predictive Checks
Deterministic Component

- Calculate desired summary statistics

```
signalMeans = apply(ysignal, 2, mean)
```

An R function that will **apply** a function across all rows or columns of a data frame or matrix.

6. Posterior Predictive Checks
Deterministic Component

- Calculate desired summary statistics

```
signalMeans = apply(ysignal, 2, mean)
```

The data frame or matrix across which to apply the function.

6. Posterior Predictive Checks
Deterministic Component

- Calculate desired summary statistics

```
signalMeans = apply(ysignal, 2, mean)
```

Indicator of whether to apply the function to rows (1) or columns (2) of the data.

6. Posterior Predictive Checks
Deterministic Component

- Calculate desired summary statistics

```
signalMeans = apply(ysignal, 2, mean)
```

What function to apply.

6. Posterior Predictive Checks
Deterministic Component

- Calculate desired summary statistics

```
signalMeans = apply(ysignal, 2, mean)
```

Will calculate the **mean** value across all steps of the chains for each x value (height value)

6. Posterior Predictive Checks
Deterministic Component

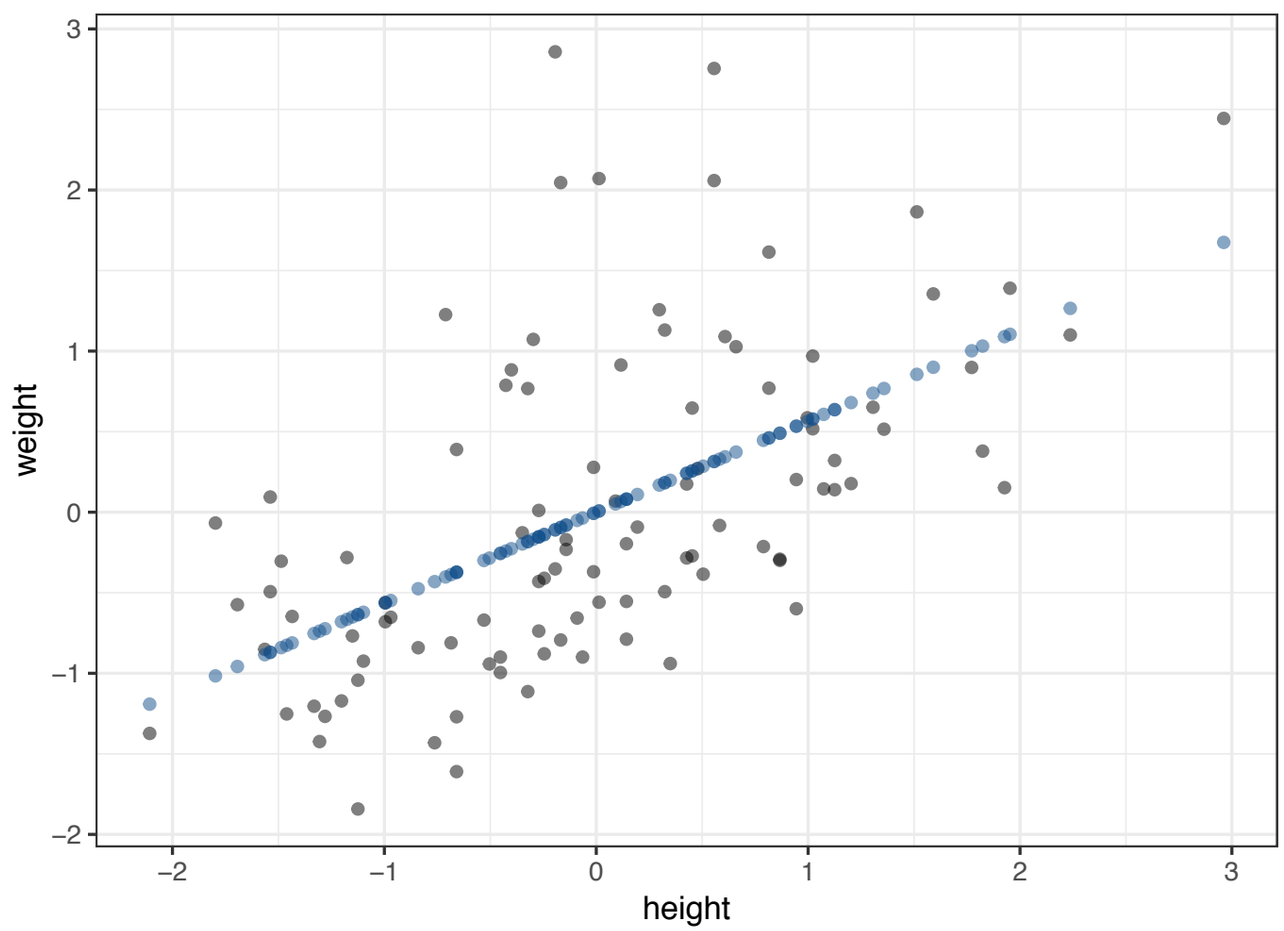
- Calculate desired summary statistics

```
signalHDIMin = apply(ysignal, 2, quantile, probs = 0.025)
signalHDIMax = apply(ysignal, 2, quantile, probs = 0.975)
signalSummary = data.frame(signalMeans, signalHDIMin, signalHDIMax, zx, zy)
```

6. Posterior Predictive Checks
Deterministic Component

- Plot raw data with mean predicted values

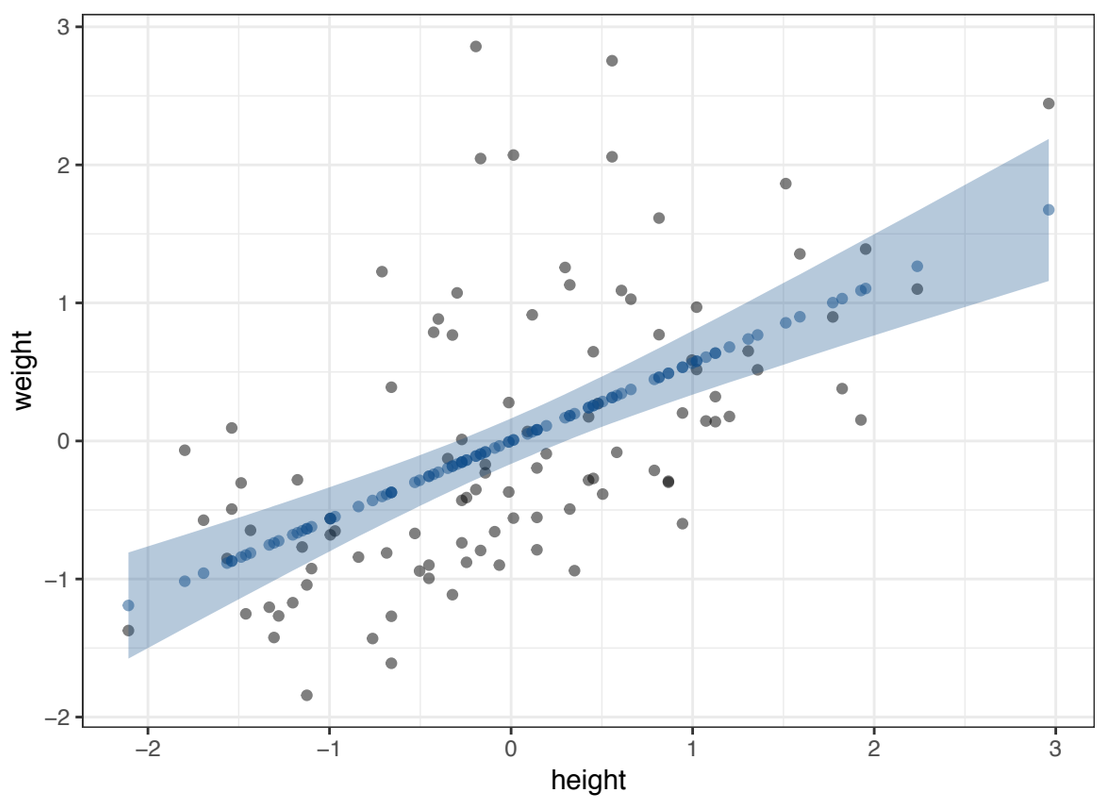
```
ggplot(signalSummary) +
  theme_bw() +
  geom_point(aes(x = zx, y = zy), alpha = 0.5) +
  geom_point(aes(x = zx, y = signalMeans), colour = "dodgerblue4", alpha = 0.5) +
  ylab("weight") +
  xlab("height")
```



6. Posterior Predictive Checks
Deterministic Component

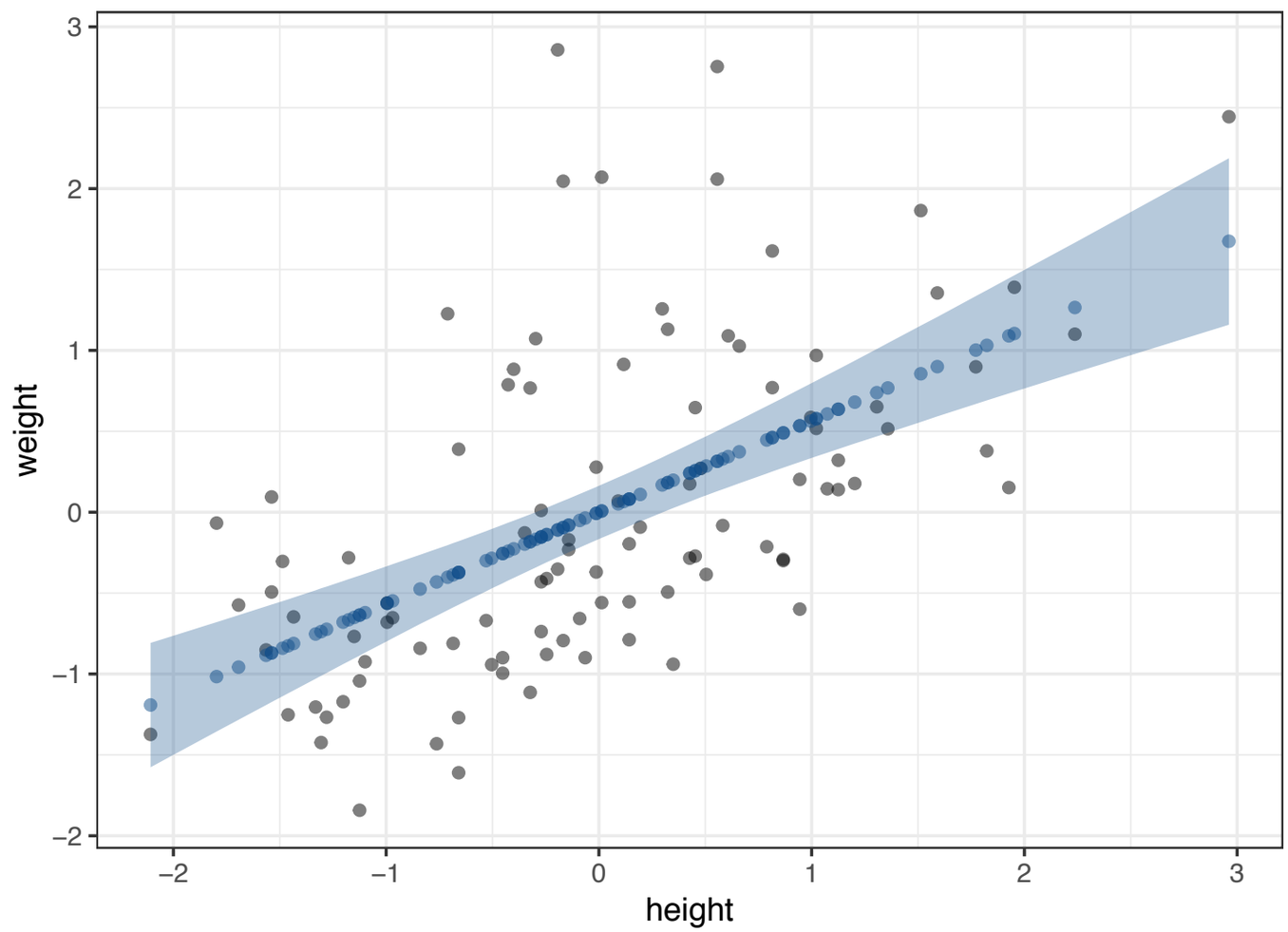
- Add a ribbon for 95% HDI

```
ggplot(signalSummary) +
  theme_bw() +
  geom_point(aes(x = zx, y = zy), alpha = 0.5) +
  geom_point(aes(x = zx, y = signalMeans), colour = "dodgerblue4", alpha = 0.5) +
  geom_ribbon(aes(x = zx, ymin = signalHDIMin, ymax = signalHDIMax),
    fill = "dodgerblue4", alpha = 0.3) +
  ylab("weight") +
  xlab("height")
```



6. Posterior Predictive Checks
Deterministic Component

- Estimates of deterministic component pretty tight
- Clearly missing some important parameters from our model
  - A lot of unexplained variation in weight (shocking!)



6. Posterior Predictive Checks

Full Model

- Can now check how well full model fits the data
- If we've done well, 95% HDI should capture 95% of all data points
- Considers both the signal *and the noise*

6. Posterior Predictive Checks

Full Model

- Calculate desired summary statistics

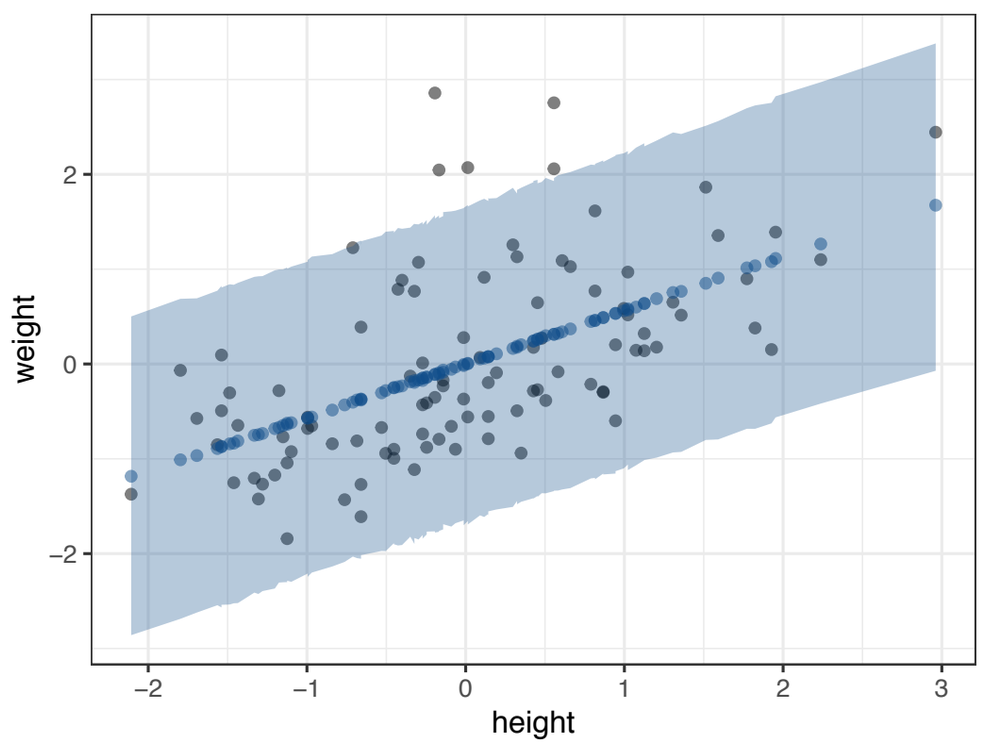
```
predMeans = apply(ypred, 2, mean)
predHDIMin = apply(ypred, 2, quantile, probs = 0.025)
predHDIMax = apply(ypred, 2, quantile, probs = 0.975)
predictions = data.frame(predMeans, predHDIMin, predHDIMax, zx, zy)
```

6. Posterior Predictive Checks

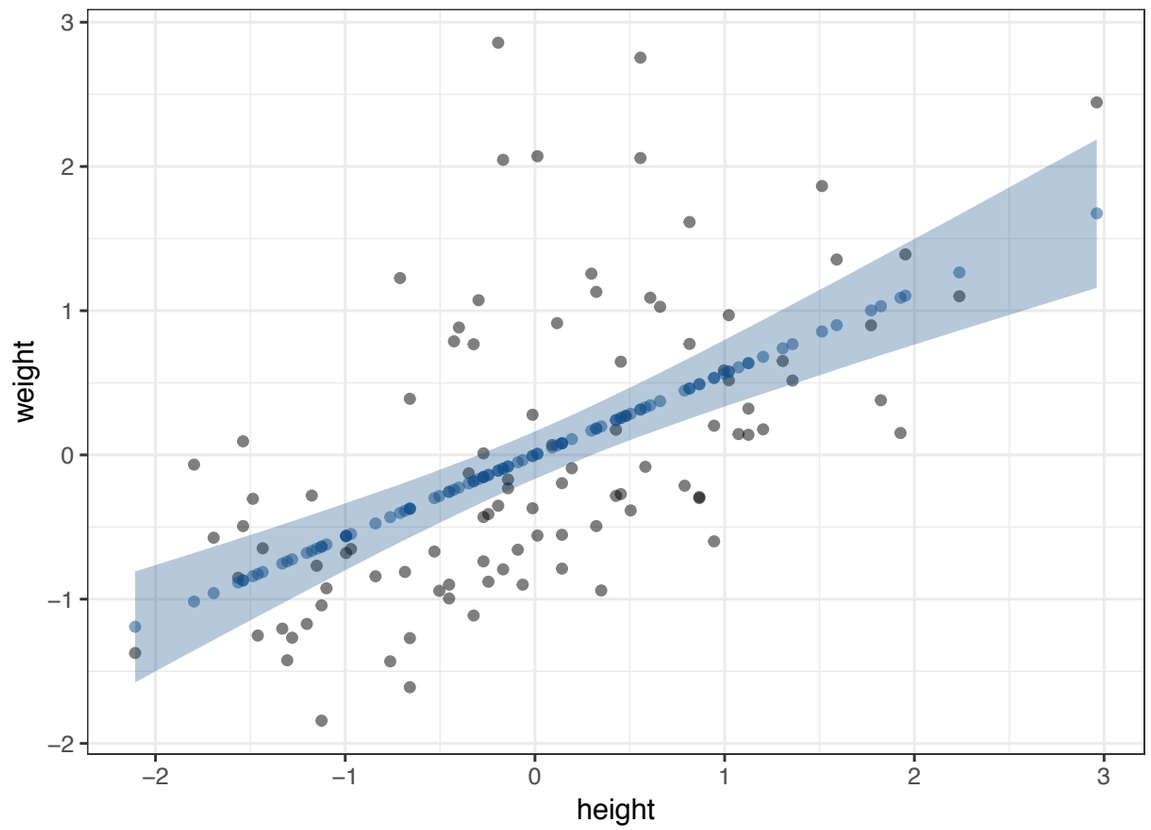
Full Model

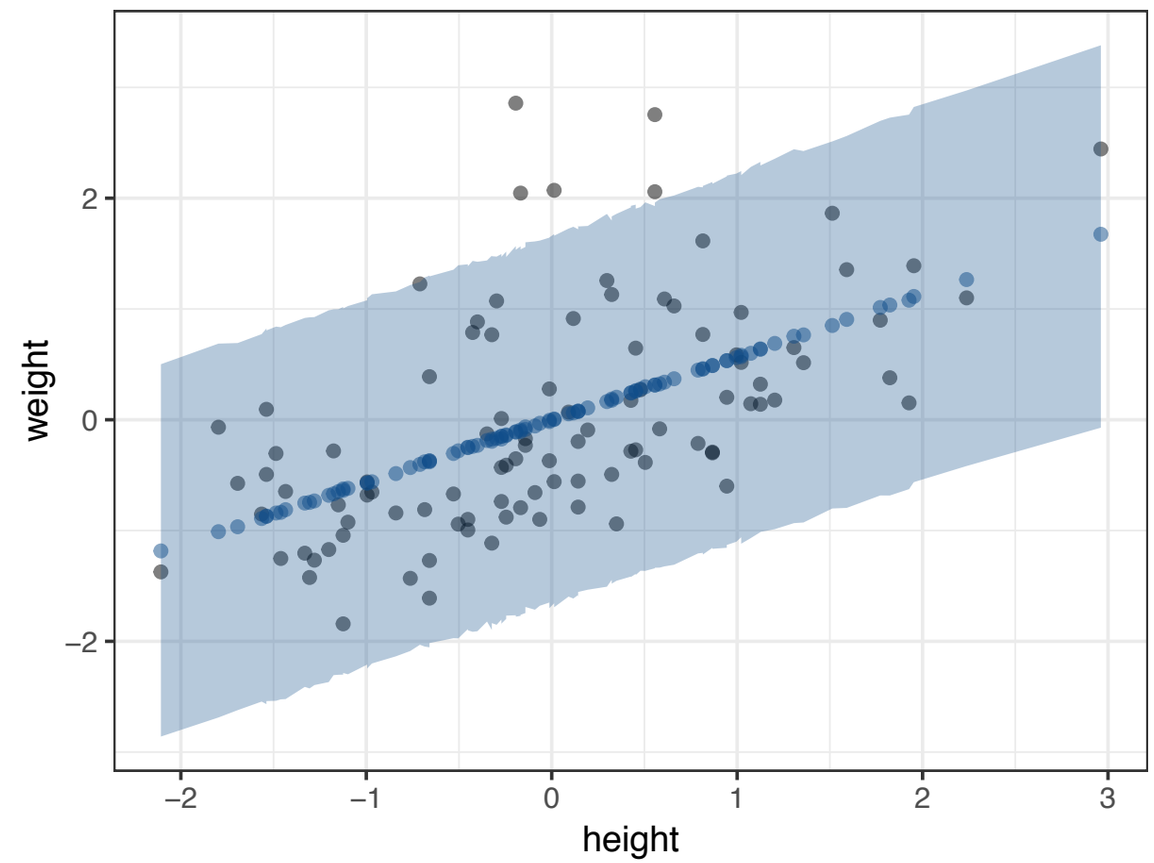
- Plot predictions

```
ggplot(predictions) +
  theme_bw() +
  geom_point(aes(x = zx, y = zy), alpha = 0.5) +
  geom_point(aes(x = zx, y = predMeans), colour = "dodgerblue4", alpha = 0.5) +
  geom_ribbon(aes(x = zx, ymin = predHDIMin, ymax = predHDIMax),
    fill = "dodgerblue4", alpha = 0.3) +
  ylab("weight") +
  xlab("height")
```



6. Posterior Predictive Checks





Predicting Weight for New Height Values

- In many cases, once we have estimates for β_0 and β_1 for a system, we then want to use that information to make predictions based on new data (and our uncertainty about those predictions)

- Our original data contained heights ranging from 58.5 to 78.1 inches

- Let's predict the weights of 50 new individuals ranging in height from 70 to 90 inches (7.5 ft!)

Predicting Weight for New Height Values

- Generate new x (height) values

```
xNew = runif(n = 50, min = 70, max = 90)
xNew

[1] 72.36499 70.03866 88.58214 85.86236 88.68099 75.34898 70.56079 78.43638
 [9] 72.23287 75.99434 83.90202 86.71789 78.46509 85.60824 83.00595 80.80889
[17] 73.21614 86.29524 78.14187 85.92406 82.17720 89.37008 88.66640 78.02620
[25] 87.58855 85.41556 73.17865 70.03777 73.33176 85.63155 74.26131 75.38690
[33] 71.43466 72.37566 78.06732 74.53229 82.95678 76.72291 81.88241 85.24027
[41] 79.15880 77.20354 70.57621 86.33159 80.02201 82.85431 89.50417 77.48950
[49] 86.06753 75.68026
```

Predicting Weight for New Height Values

- Order for easier visualization later

```
xNew = sort(xNew)
xNew

[1] 70.03777 70.03866 70.56079 70.57621 71.43466 72.23287 72.36499 72.37566
 [9] 73.17865 73.21614 73.33176 74.26131 74.53229 75.34898 75.38690 75.68026
[17] 75.99434 76.72291 77.20354 77.48950 78.02620 78.06732 78.14187 78.43638
[25] 78.46509 79.15880 80.02201 80.80889 81.88241 82.17720 82.85431 82.95678
[33] 83.00595 83.90202 85.24027 85.41556 85.60824 85.63155 85.86236 85.92406
[41] 86.06753 86.29524 86.33159 86.71789 87.58855 88.58214 88.66640 88.68099
[49] 89.37008 89.50417
```

Predicting Weight for New Height Values

- Next, define a matrix that will hold all of the predicted `y` values
  - Number of rows is the number of `x` values for prediction
  - Number of columns is the number of `y` values generated from the MCMC process
  - We'll start with the matrix filled with zeros, but will fill it in later

```
postSampSize = length(b1)

yNew = matrix(0, nrow = length(xNew), ncol = postSampSize)
```

Predicting Weight for New Height Values

- Define a matrix for holding the HDI limits of the predicted y values
  - Same number of rows as above
  - Only two columns (one for each end of the HDI)

```
yHDIlim = matrix(0, nrow = length(xNew), ncol = 2)
```

Predicting Weight for New Height Values

- Now, populate the `yNew` matrix by generating one predicted y value for each step in the chain

```
for (i in 1:length(xNew)) {
  for (j in 1:postSampSize) {
    yNew[i, j] = rnorm(n = 1, mean = b0[j] + b1[j] * xNew[i], sd = sigma[j])
  }
}
```

Predicting Weight for New Height Values

- Now, populate the `yNew` matrix by generating one predicted y value for each step in the chain

```
for (i in 1:length(xNew)) {
  for (j in 1:postSampSize) {
    yNew[i, j] = rnorm(n = 1, mean = b0[j] + b1[j] * xNew[i], sd = sigma[j])
  }
}
```

For each new `x` value …

Predicting Weight for New Height Values

- Now, populate the `yNew` matrix by generating one predicted y value for each step in the chain

```
for (i in 1:length(xNew)) {
  for (j in 1:postSampSize) {
    yNew[i, j] = rnorm(n = 1, mean = b0[j] + b1[j] * xNew[i], sd = sigma[j])
  }
}
```

For every step in the chain...

Predicting Weight for New Height Values

- Now, populate the `yNew` matrix by generating one predicted y value for each step in the chain

```
for (i in 1:length(xNew)) {
  for (j in 1:postSampSize) {
    yNew[i, j] = rnorm(n = 1, mean = b0[j] + b1[j] * xNew[i], sd = sigma[j])
  }
}
```

Generate a new `y` value by randomly sampling one value from a normal distribution...

Predicting Weight for New Height Values

- Now, populate the `yNew` matrix by generating one predicted y value for each step in the chain

```
for (i in 1:length(xNew)) {
  for (j in 1:postSampSize) {
    yNew[i, j] = rnorm(n = 1, mean = b0[j] + b1[j] * xNew[i], sd = sigma[j])
  }
}
```

With a mean equal to `b0 + b1*x` (using the b0 and b1 values from that step in the chain)...

Predicting Weight for New Height Values

- Now, populate the `yNew` matrix by generating one predicted y value for each step in the chain

```
for (i in 1:length(xNew)) {
  for (j in 1:postSampSize) {
    yNew[i, j] = rnorm(n = 1, mean = b0[j] + b1[j] * xNew[i], sd = sigma[j])
  }
}
```

And a standard deviation based on that step in the chain

Predicting Weight for New Height Values

- Calculate means for each prediction, and the associated low and high 95% HDI estimates

```
means = rowMeans(yNew)

for (i in 1:length(xNew)) {
  yHDIlim[i, ] = quantile(yNew[i, ], probs = c(0.025, 0.975))
}
```

Predicting Weight for New Height Values

- Combine the data

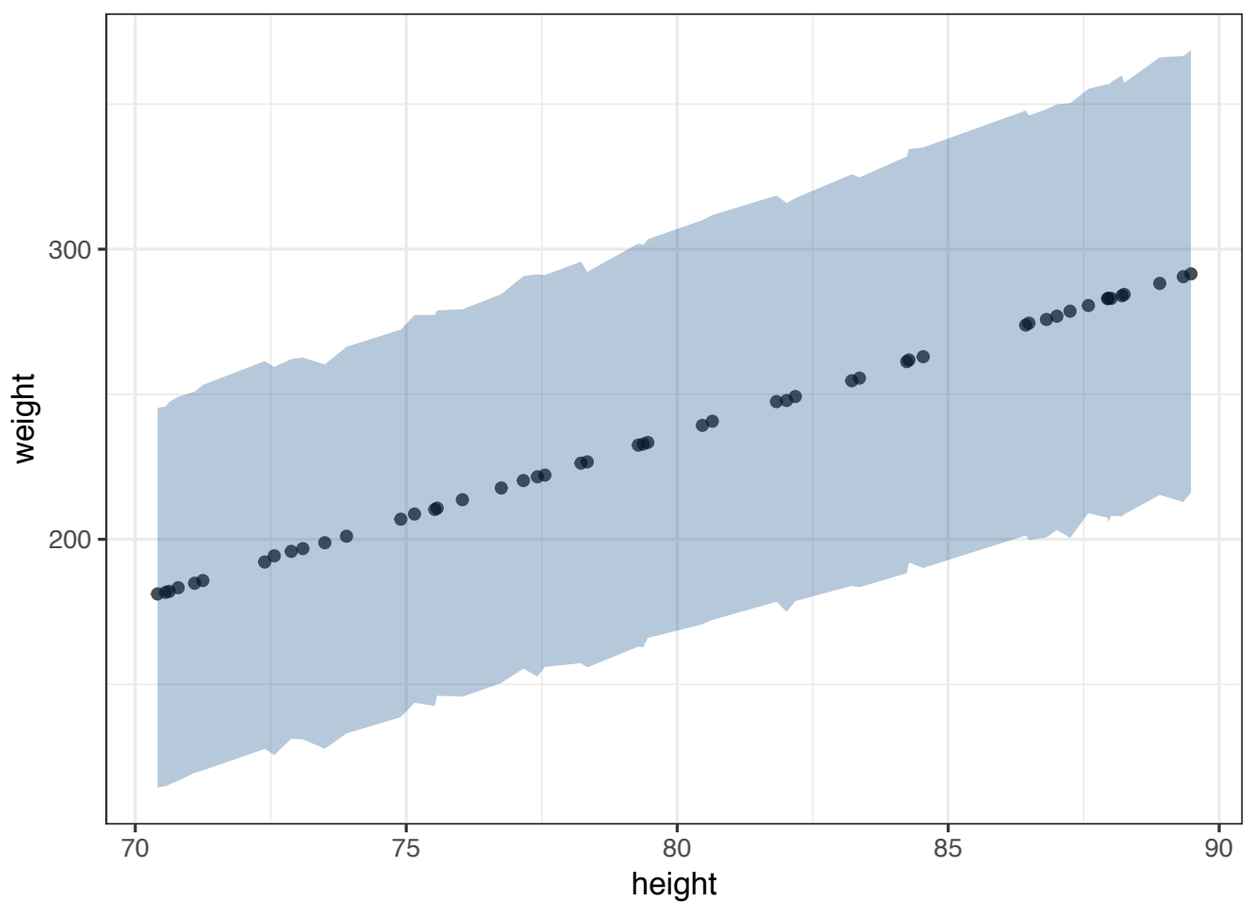
```
predTable = data.frame(xNew, means, yHDIlim)
```

xNew	means	V3	V4
70.41042	181.1716	114.4159	245.3072
70.55985	181.7180	114.9396	245.7509
70.62610	182.0531	115.4858	247.3272
70.79192	183.3163	116.7804	249.1335
71.09461	184.8677	119.5066	250.8810
71.24652	185.7985	120.3223	253.1887
72.38824	192.1810	127.7456	261.4491
72.56511	194.3028	125.6432	259.4161
72.87910	195.8427	131.2240	262.1318
73.09264	196.7971	131.0395	262.6172
73.49713	198.8184	127.8091	260.2962
73.89766	201.0719	133.0695	266.3588
74.90008	206.9332	138.6882	272.3089

Predicting Weight for New Height Values

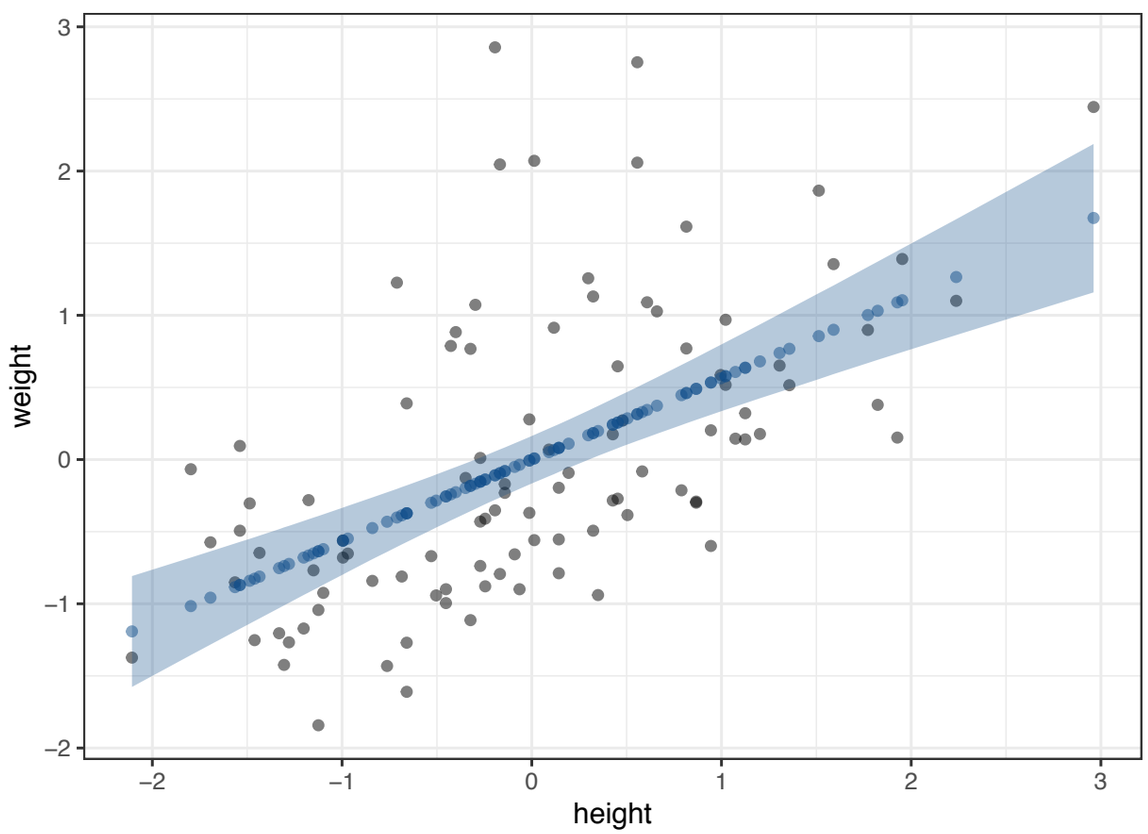
- Plot the results

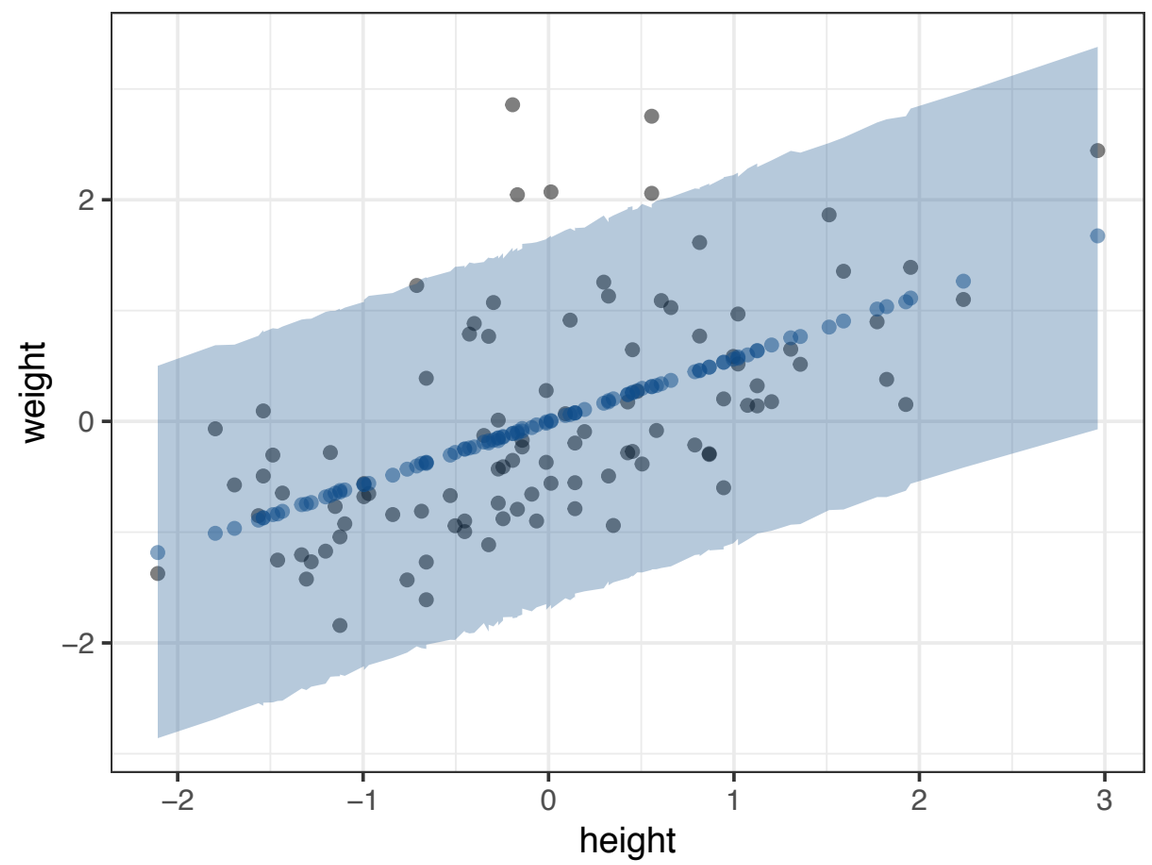
```
ggplot(predTable) +
  theme_bw() +
  geom_point(aes(x = xNew, y = means), alpha = 0.7) +
  geom_ribbon(aes(x = xNew, ymin = predTable[, 3], ymax = predTable[, 4]),
    fill = "dodgerblue4", alpha = 0.3) +
  ylab("weight") +
  xlab("height")
```



Predicting Weight for New Height Values

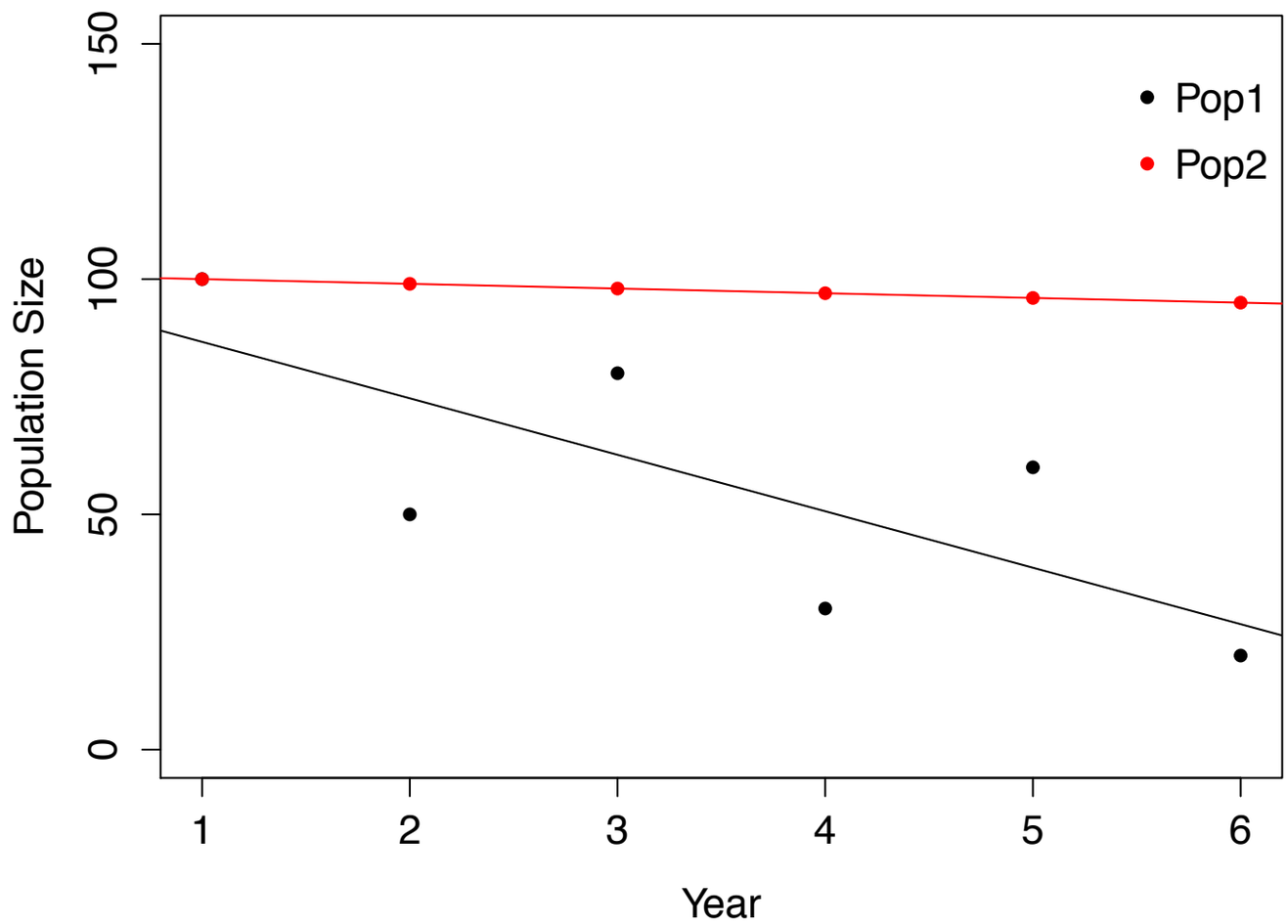
- Note how important accounting for "unexplained variance" is when making predictions
  - Can you even do this with a frequentist approach?





Re-Analysis of Some "Old" Data

Remember This?



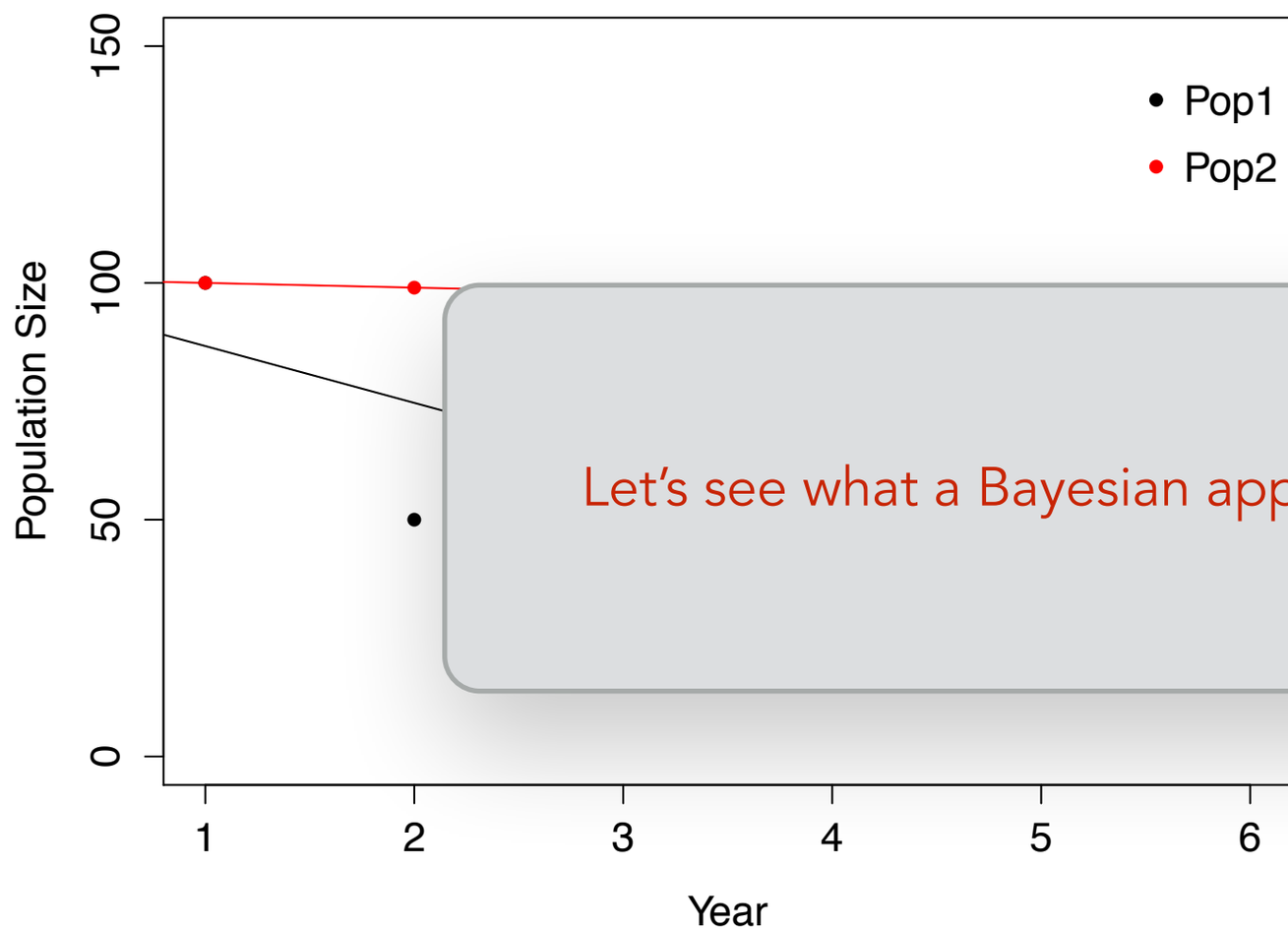
Year	Pop1	Pop2
1	100	100
2	50	99
3	80	98
4	30	97
5	60	96
6	20	95

Pop1 $p = 0.08$

Pop2 $p = 2.2 \times 10^{-16}$

Huh?

Remember This?



Year	Pop1	Pop2
1	100	100
2	50	99
	80	98
	30	97
	60	96
	20	95

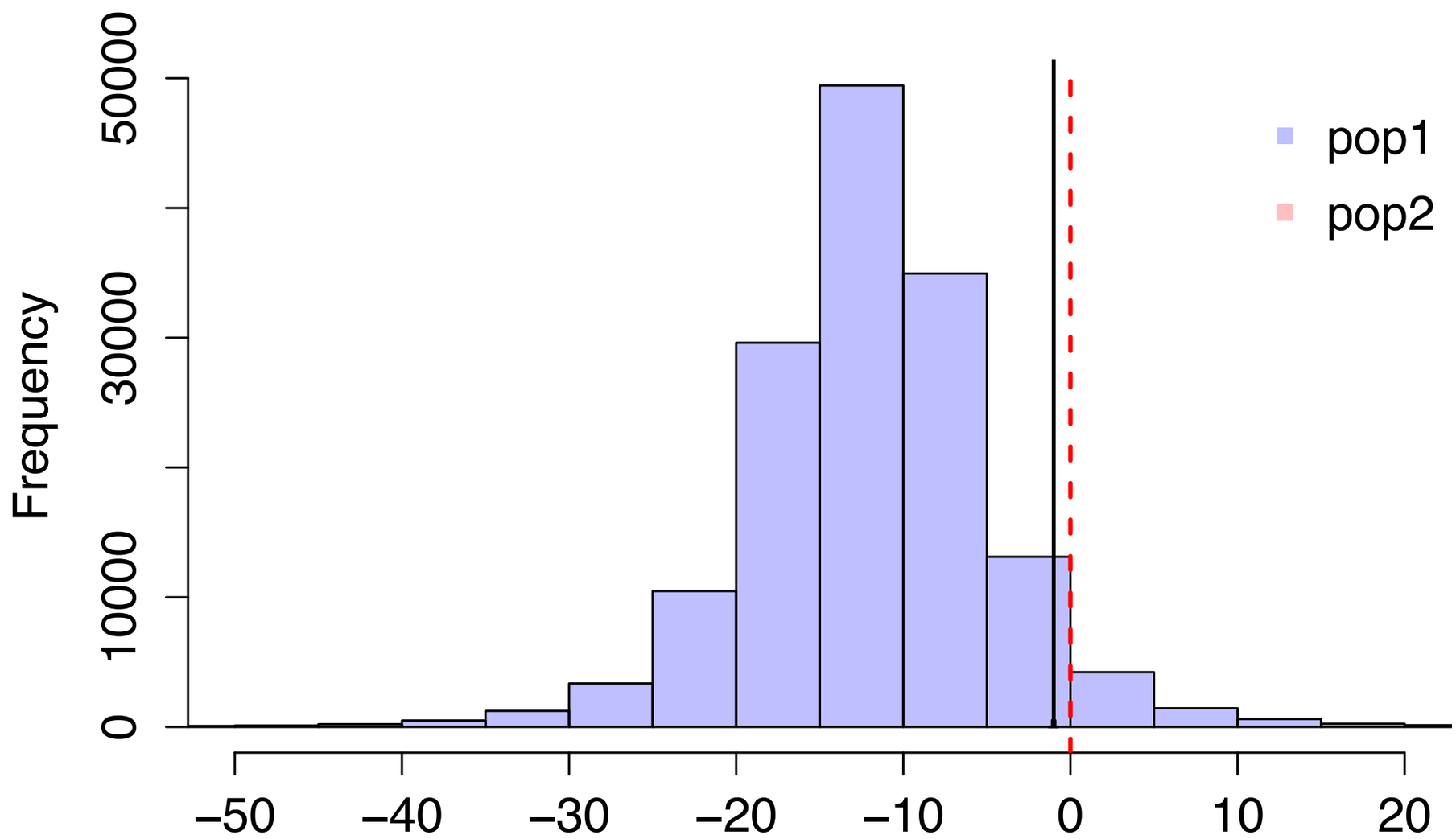
Let's see what a Bayesian approach would say!

Pop1 p = 0.08

Pop2 p = 2.2×10^{-16}

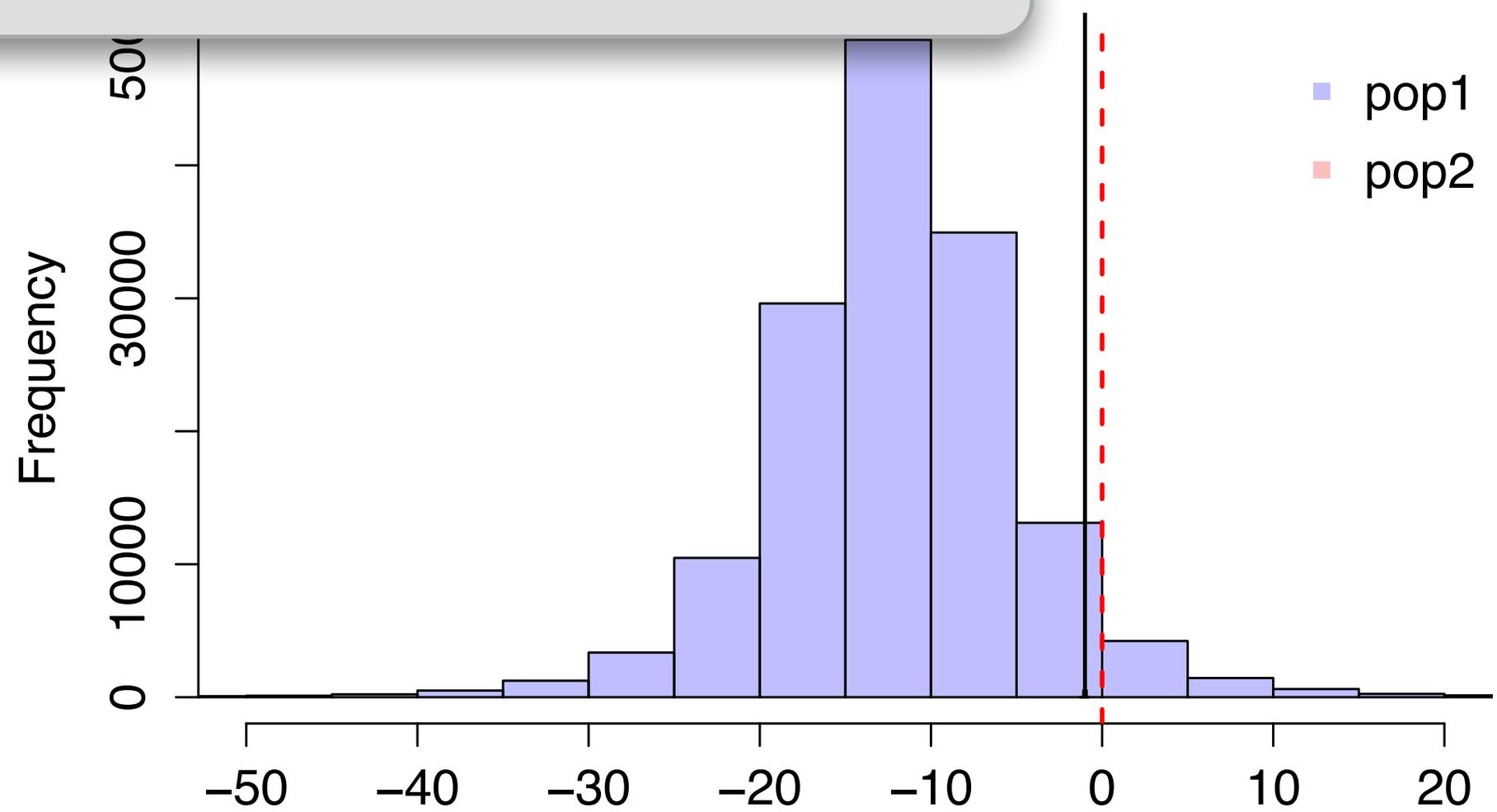
Huh?

Compare Trends of Both Populations



Compare Trends of Both Populations

- Both are clearly declining
  - pop1 more rapidly than pop2
  - Less certain about actual rate of pop1 than pop2



Questions?