Homework1 Bayesian modeling

Table of contents

Load libraries

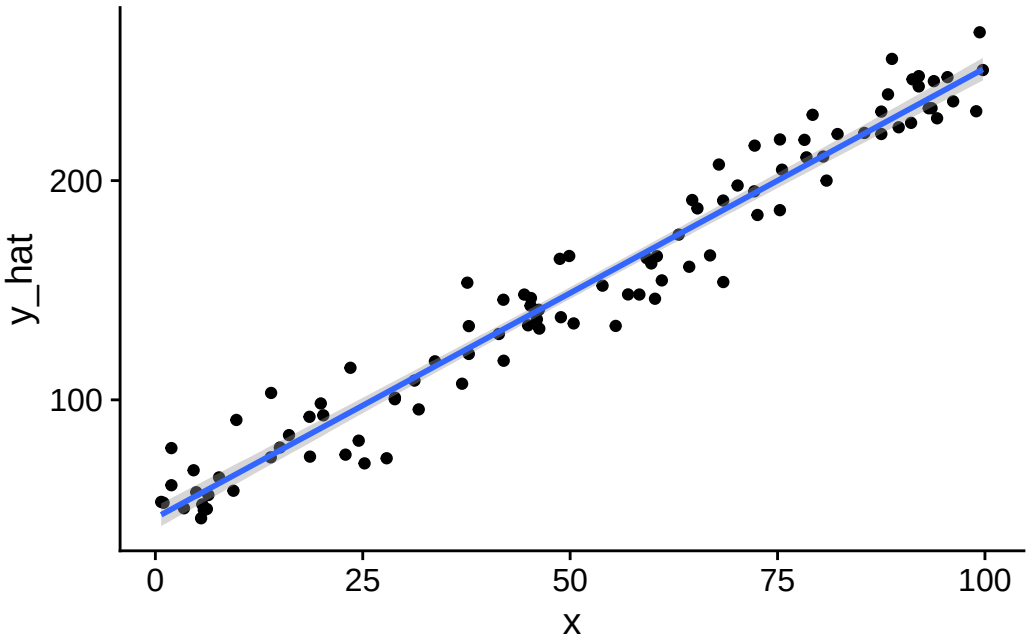
```
library(tidyverse)
library(rstanarm)
library(broom.mixed)
library(ggpubr)
library(cowplot)
ggplot2::theme_set(theme_cowplot())
```

Step 1 - Simulate data for a linear regression

```
sim_data <- function(N = 100, sigma = 15, a = 50, b1 = 2, b2 = -25, b3 = 1,
                     formula = a + b1 * x){
  x <- runif(N, 0, 100)
  binary <- rbinom(N, 1, .5)
  noise <- rnorm(N, 0, sigma)
  y_pred <- eval(substitute(formula))
  y_hat <- y_pred + noise
  tibble(x, binary, y_hat)
}
fake <- sim_data(formula = a + b1 * x)
```

```
ggplot(data = fake, aes(x = x, y = y_hat)) +
  geom_point() +
  geom_smooth(method = "lm")
```

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



Run linear mod to see if we can return simulated parameter value

```
mod1 <- stan_glm(y_hat ~ x, data = fake, refresh = 0)
coef(mod1)
```

```
(Intercept)           x
  46.210531    2.049547
```

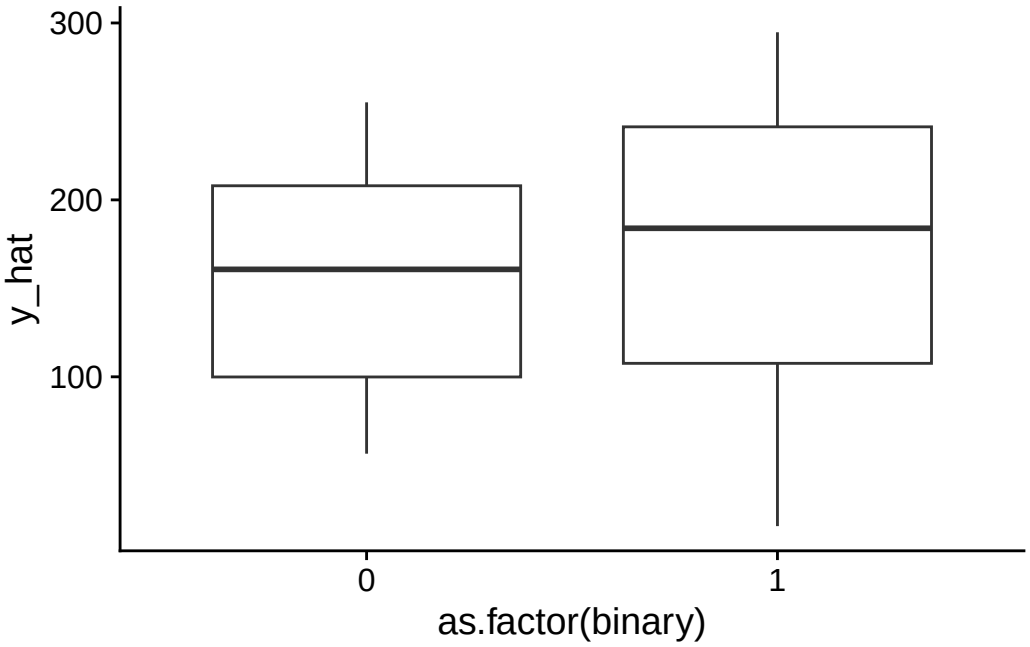
This does a pretty good job of estimating alpha & beta!

```
fake <- sim_data(formula = a + b1 * x + b2 * binary + b3 * x * binary)
mod2 <- stan_glm(y_hat ~ x * binary, data = fake, refresh = 0)
coef(mod2)
```

```
(Intercept)           x      binary    x:binary
 53.8088818   1.9672596 -20.7925959   0.9051055
```

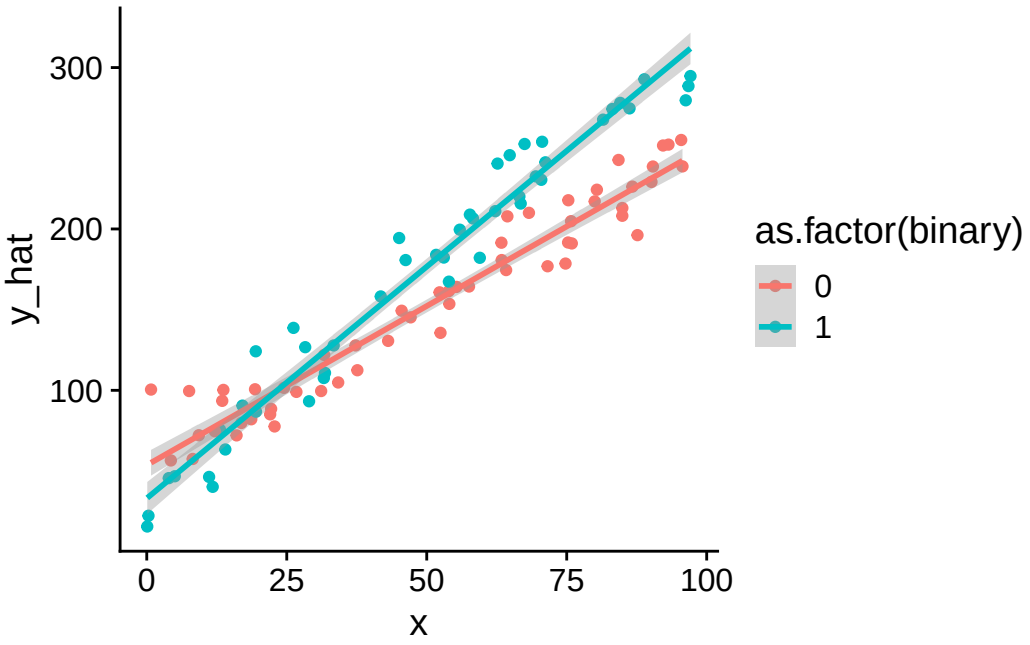
Pretty decent at returning simulated parameter values here, but added complexity of model generating the data may make it harder to accurately estimate simulated parameter values?

```
ggplot(data = fake, aes(x = as.factor(binary), y = y_hat )) +
  geom_boxplot()
```




```
ggplot(data = fake, aes(x = x, y = y_hat, color = as.factor(binary))) +
  geom_point() +
  geom_smooth(method = "lm")
```

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



Step 2 - How does N affect parm estimates?

Use 20% of data

```
fake_20 <- fake %>% slice_sample(n = 20)
mod3 <- stan_glm(y_hat ~ x * binary, data = fake_20, refresh = 0)
coef(mod3)
```

```
(Intercept)           x      binary    x:binary
 37.7780326   2.1361155  -3.1383024   0.6531599
```

Not surprisingly this does worse job of returning parameter estimates

Examine parm estimates with sample size N=1:100

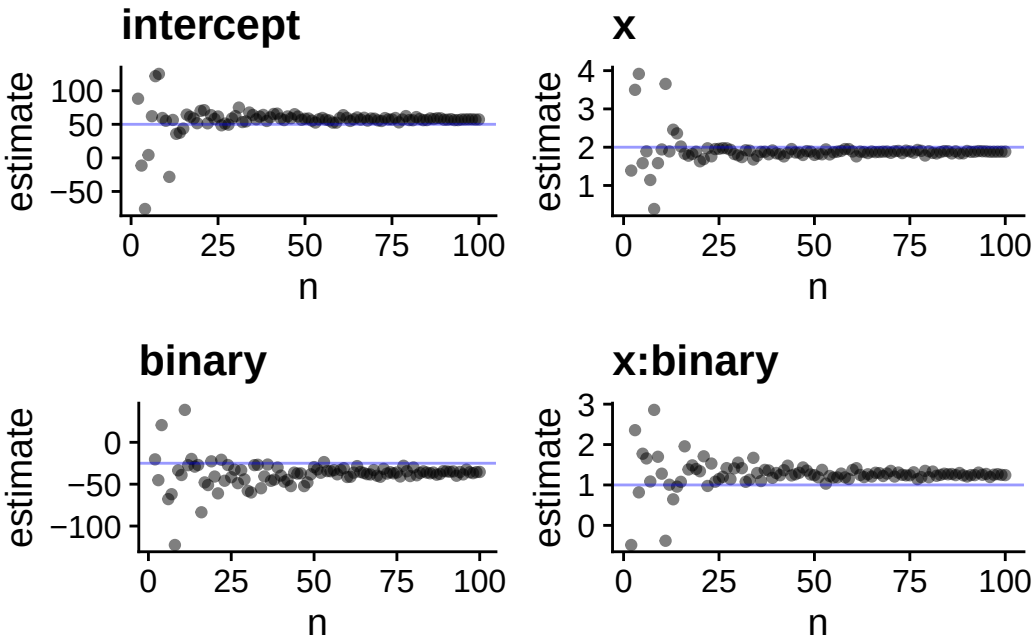
```
# Create a safely-wrapped stan_glm function
safe_stan_glm <- safely(function(data) stan_glm(y_hat ~ x * binary, data = data, refresh = 0)

# Simulate models for different sample sizes
n_values <- setNames(1:100, 1:100)
parm_vals <- map(n_values, \(n) {
  fake_n <- fake %>% slice_sample(n = n)
  fit_stan <- safe_stan_glm(fake_n)
  if (!is.null(fit_stan$result)) {
    broom.mixed::tidy(fit_stan$result)  # Extract and tidy successful fits
  } else {
    # Placeholder for failed fits
    tibble(term = NA, estimate = NA, std.error = NA, statistic = NA, p.value = NA)
  }
})

# Combine results into a single data frame
parm_vals_df <- bind_rows(parm_vals, .id = "n") %>%
  mutate(n = as.numeric(n))
```

```
terms <- unique(parm_vals_df$term)
# Remove NA & sigma (only estimated from 1 model for some reason)
terms <- terms %>%
```

```
  na.omit() %>%
  setdiff("sigma")
terms
```

```
[1] "(Intercept)" "x"           "binary"      "x:binary"
```

```
labels <- c("intercept", "x", "binary", "x:binary")
true_vals <- c(a = 50, b1 = 2, b2 = -25,  b3 = 1)

plots_parms <- pmap(list(terms, true_vals, labels), \(parm, truth, labs){
  parm_vals_df %>% filter(term == parm) %>%
    ggplot(aes(x = n, y = estimate)) +
    geom_point(alpha = .5) +
    labs(title = labs) +
    geom_hline(yintercept = truth, color = "blue", alpha = .4) #+
    #geom_errorbar(aes(ymin = estimate - std.error, ymax = estimate + std.error),
                  #width = 0.2, , alpha = .5)
})

ggarrange(plots_parms[[1]], plots_parms[[2]], plots_parms[[3]], plots_parms[[4]])
```



By n = 100, the model does (roughly) equally well with all parameter estimates. However the required sample size for the parameter estimates to converge upon their true value varies greatly between the continuous predictor variable x , and binary predictor (or the interaction).

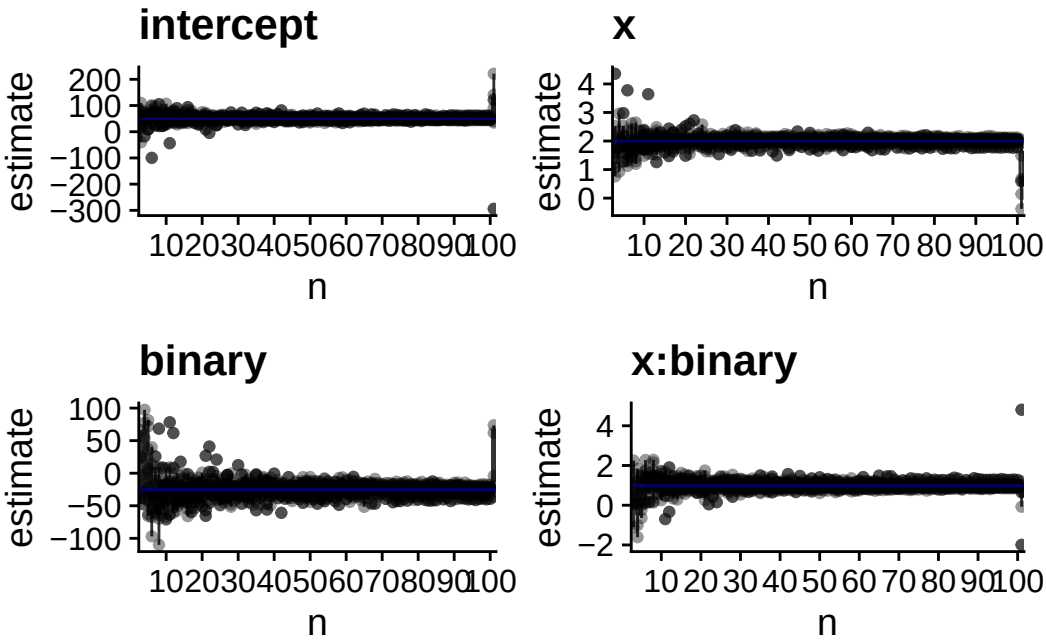
- x: The continuous predictor benefits from a large, well-distributed dataset and a moderate effect size, leading to faster convergence.
- Binary: The binary predictor provides less information due to its discrete nature, so its estimate converges more slowly.
- Interaction: The interaction term suffers from both reduced effective sample size (due to the binary split) and signal dilution, making it the slowest to converge.

Now let's take 10 draws per value of N

```
draws <- setNames(1:10, 1:10)
fake_10_draws_l <- map(draws, ~sim_data(formula = a + b1 * x + b2 * binary + b3 * x * binary)

parm_vals2 <- map(fake_10_draws_l, \(draw_df){ #1:10
  map(n_values, \(n) { #1:100
  fake_n <- draw_df %>% slice_sample(n = n)
  fit_stan <- safe_stan_glm(fake_n)
  if(!is.null(fit_stan$result)) {
    broom.mixed::tidy(fit_stan$result)  # Extract and tidy successful fits
  } else {
    # Placeholder for failed fits
    tibble(term = NA, estimate = NA, std.error = NA, statistic = NA, p.value = NA)
  }
  })
})
```

```
# We have a nested list -- flatten & extract draw number and sample size n
parm_vals_df2 <- parm_vals2 %>% list_flatten() %>%
  list_rbind(names_to = "draw_n") %>%
  mutate(draw = str_split_i(draw_n, "_", 1),
         n = str_split_i(draw_n, "_", 2)) %>%
  select(-draw_n)

plots_parms2 <- pmap(list(terms, true_vals, labels), \(parm, truth, labs){
  parm_vals_df2 %>% filter(term == parm) %>%
    ggplot(aes(x = as_factor(n), y = estimate)) +
    geom_boxplot(alpha = .6) +
    geom_point(alpha = .4) +
    labs(title = labs, x = "n") +
    geom_hline(yintercept = truth, color = "blue", alpha = .4) +
    scale_x_discrete(breaks = seq(10, 100, 10))
```

```
})

ggarrange(plots_parms2[[1]], plots_parms2[[2]], plots_parms2[[3]], plots_parms2[[4]])
```

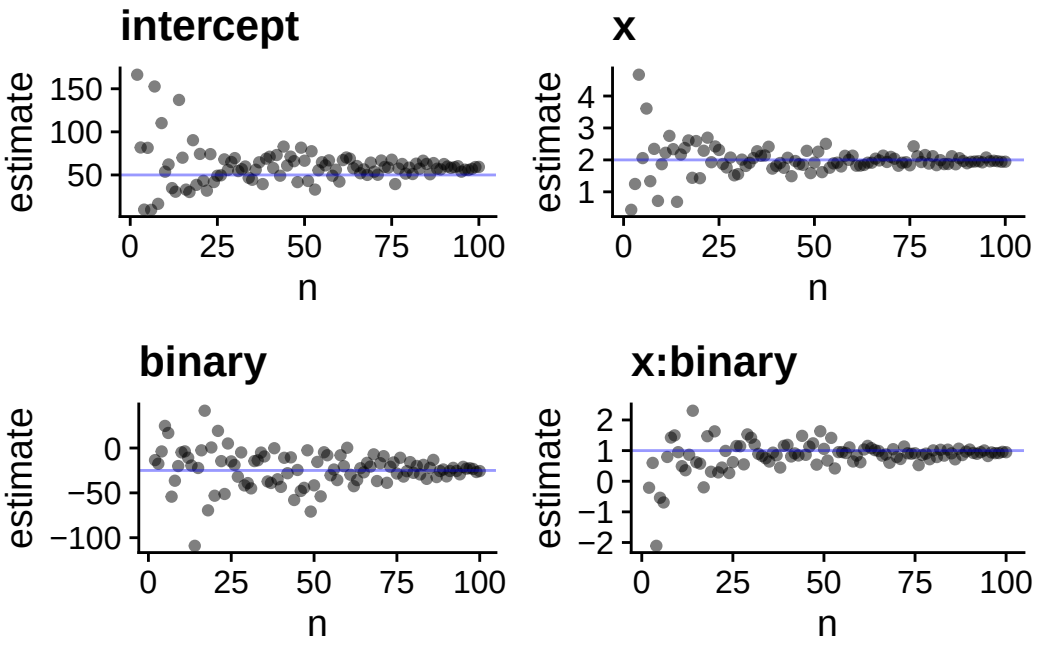


Step 3 - Change error term by 50%

```
fake_noisy <- sim_data(sigma = 30, formula = a + b1 * x + b2 * binary + b3 * x * binary)

parm_vals_noisy <- map(n_values, \(n) {
  fake_n <- fake_noisy %>% slice_sample(n = n)
  fit_stan <- safe_stan_glm(fake_n)
  if (!is.null(fit_stan$result)) {
    broom.mixed::tidy(fit_stan$result)  # Extract and tidy successful fits
  } else {
    # Placeholder for failed fits
    tibble(term = NA, estimate = NA, std.error = NA, statistic = NA, p.value = NA)
  }
})

# Combine results into a single data frame
```

```
parm_vals_noisy <- bind_rows(parm_vals_noisy, .id = "n") %>%
  mutate(n = as.numeric(n))
```

```
plots_parms_noisy <- pmap(list(terms, true_vals, labels), \(parm, truth, labs){
  parm_vals_noisy %>% filter(term == parm) %>%
    ggplot(aes(x = n, y = estimate)) +
    geom_point(alpha = .5) +
    labs(title = labs) +
    geom_hline(yintercept = truth, color = "blue", alpha = .4) #+
    #geom_errorbar(aes(ymin = estimate - std.error, ymax = estimate + std.error),
                  #width = 0.2, , alpha = .5)
})

ggarrange(plots_parms_noisy[[1]], plots_parms_noisy[[2]],
          plots_parms_noisy[[3]], plots_parms_noisy[[4]])
```



Increasing the variability in the dataset (doubling sigma) causes the spread of the estimates to increase greatly, and models requires a larger sample size (n) to begin to approximate the true parameter estimate. However, these estimates do not reach their true values with n less than or equal to 100.

Step 4 - Review datasets

I would like to work with the carnivore tooth dataset.