# Assignment 3

Michal Malyska

## Question 1

### a) This is just a normal-normal conjugate distribution pair so we have:

$$p(\mu|y) \propto p(y|\mu)p(\mu) \sim N(\mu_{post}, \sigma^2_{post})$$

Where:

$$\mu_{post} = \sigma^2_{post}\left(\frac{\mu_0}{\sigma^2_0} + \frac{n\bar{Y}}{\sigma^2}\right)$$

and

$$\sigma^2_{post} = \left(\frac{1}{\sigma^2_0} + \frac{n}{\sigma^2}\right)^{-1}$$

in our case $n = 10$, $\mu_0 = 100$, $\bar{Y} = 113$, $\sigma_0 = \sigma = 15$ which gives:

$$\sigma^2_{post} = \left(\frac{1}{15^2} + \frac{10}{15^2}\right)^{-1} = 20.45$$

$$\mu_{post} = 20.45\left(\frac{100}{15^2} + \frac{1130}{15^2}\right) = 111.82$$

This will obviously also be the expected value of $\mu_{post}$ and hence the Bayesian point estimate (same with median so it doesn't matter which one we take). The credible intervals will be just the corresponding quantiles of the normal distribution:

```
sigmasq_post <- 1 / (11 / 15^2)
sigma_post <- sqrt(sigmasq_post)
print(paste0("Posterior Variance is : ", sigmasq_post))
```

```
## [1] "Posterior Variance is : 20.4545454545455"
```

```
mu_post <- sigmasq_post * (1230 / 15^2)
print(paste0("Posterior Mean is : ", mu_post))
```

```
## [1] "Posterior Mean is : 111.818181818182"
```

```
cr_interv_lower <- qnorm(p = 0.025, mean = mu_post, sd = sigma_post)
cr_interv_upper <- qnorm(p = 0.975, mean = mu_post, sd = sigma_post)

print(paste0("The credible for posterior mean is: [", cr_interv_lower, ",", cr_interv_upper , "]"))
```

```
## [1] "The credible for posterior mean is: [102.953911173642,120.682452462722]"
```

## b

$\mathbb{E}\left((\hat{\mu}-\mu^*)^2|\mu^*\right) = \mathbb{E}\left((\hat{\mu}^2-2\hat{\mu}\mu^*+\mu^{*2})|\mu^*\right) = \mathbb{E}\left(\hat{\mu}^2|\mu^*\right)+\mathbb{E}\left(-2\hat{\mu}\mu^*+\mu^{*2}|\mu^*\right) = \mathbb{V}ar(\hat{\mu}^2|\mu^*)+[\mathbb{E}(\hat{\mu}|\mu^*)]^2+\mathbb{E}\left(-2\hat{\mu}\mu^*+\mu^{*2}|\right.$

or just by noting that $\hat{\mu}|\mu^* \perp \mu^*$ from which it directly follows.

## c

Assuming $\mu^* = 112$

$Bias_B = 111.82 - 112 = 0.18$ $Bias_{MLE} = 113 - 112 = 1$

$$\mathbb{V}ar_B = \mathbb{V}ar(\mu_{post}) = \frac{n^2\sigma_0^4}{(\sigma^2+n\sigma_0^2)^2}\mathbb{V}ar(\bar{Y}) = \frac{100*15^4}{(15^2+10*15^2)^2}\mathbb{V}ar_{MLE} = 0.82644\mathbb{V}ar_{MLE}$$

$\mathbb{V}ar_{MLE} = \mathbb{V}ar(\bar{Y}) = \frac{\sigma^2}{n} = \frac{15^2}{10} = 22.5$

$MSE_B = 0.18^2 + 0.82644 * 22.5 = 18.6273$

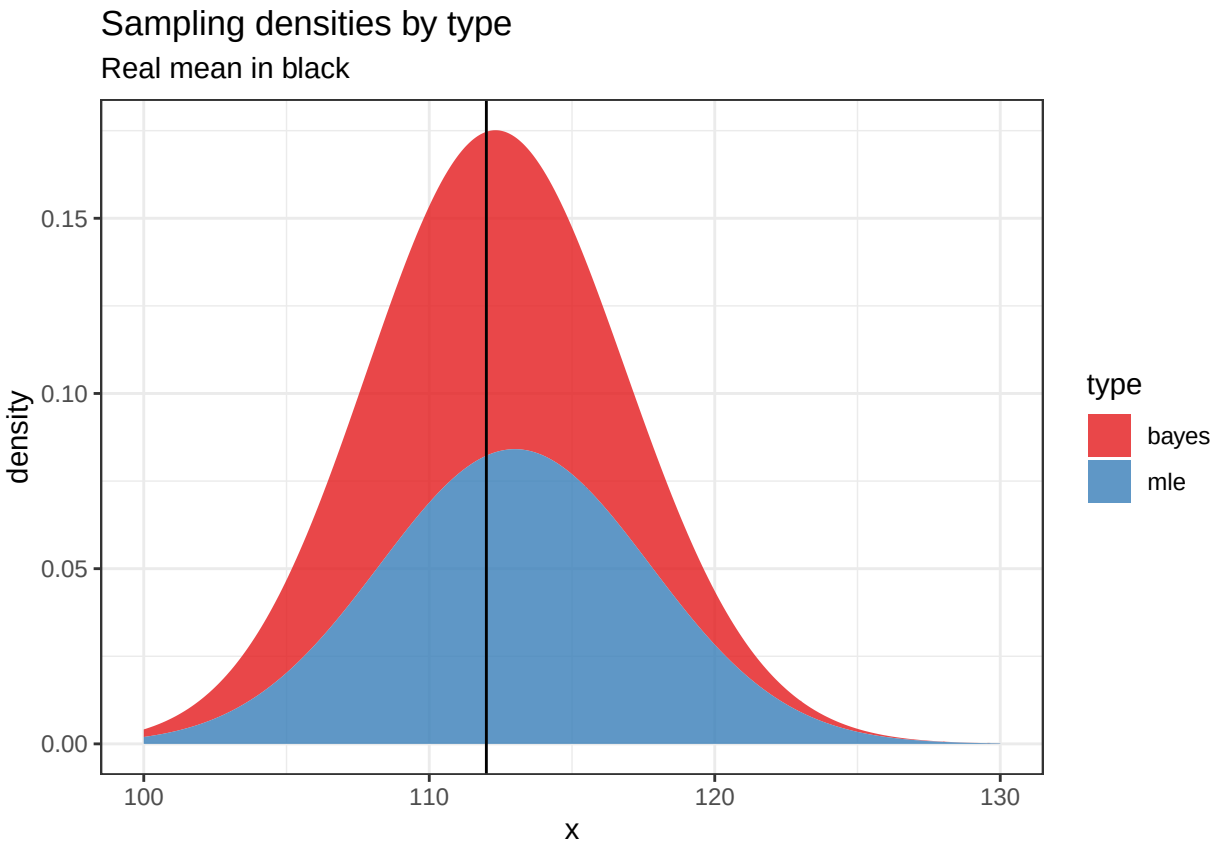
$MSE_{MLE} = 1 + 22.5 = 23.5$

MLE has a larger bias and a larger variance thus a larger MSE.

## d

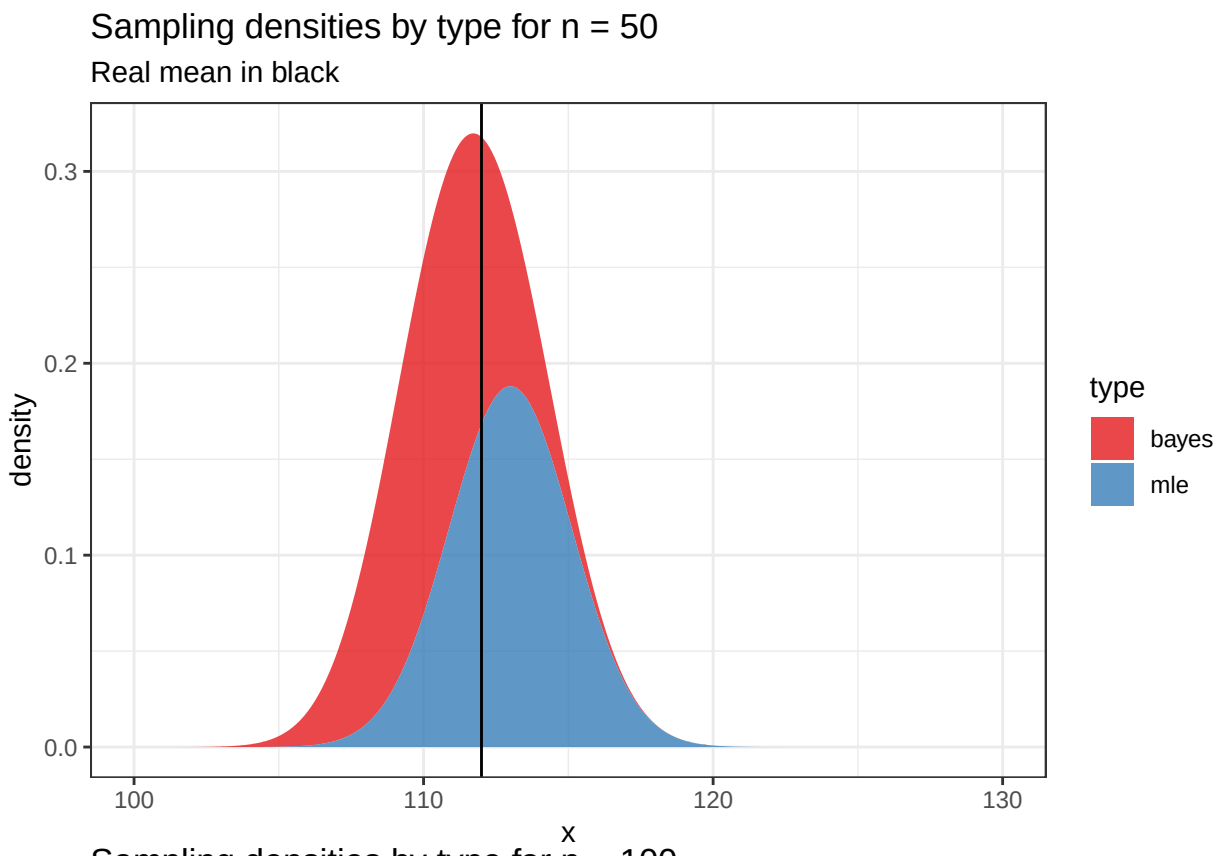
Both are normal with the means and variances as above

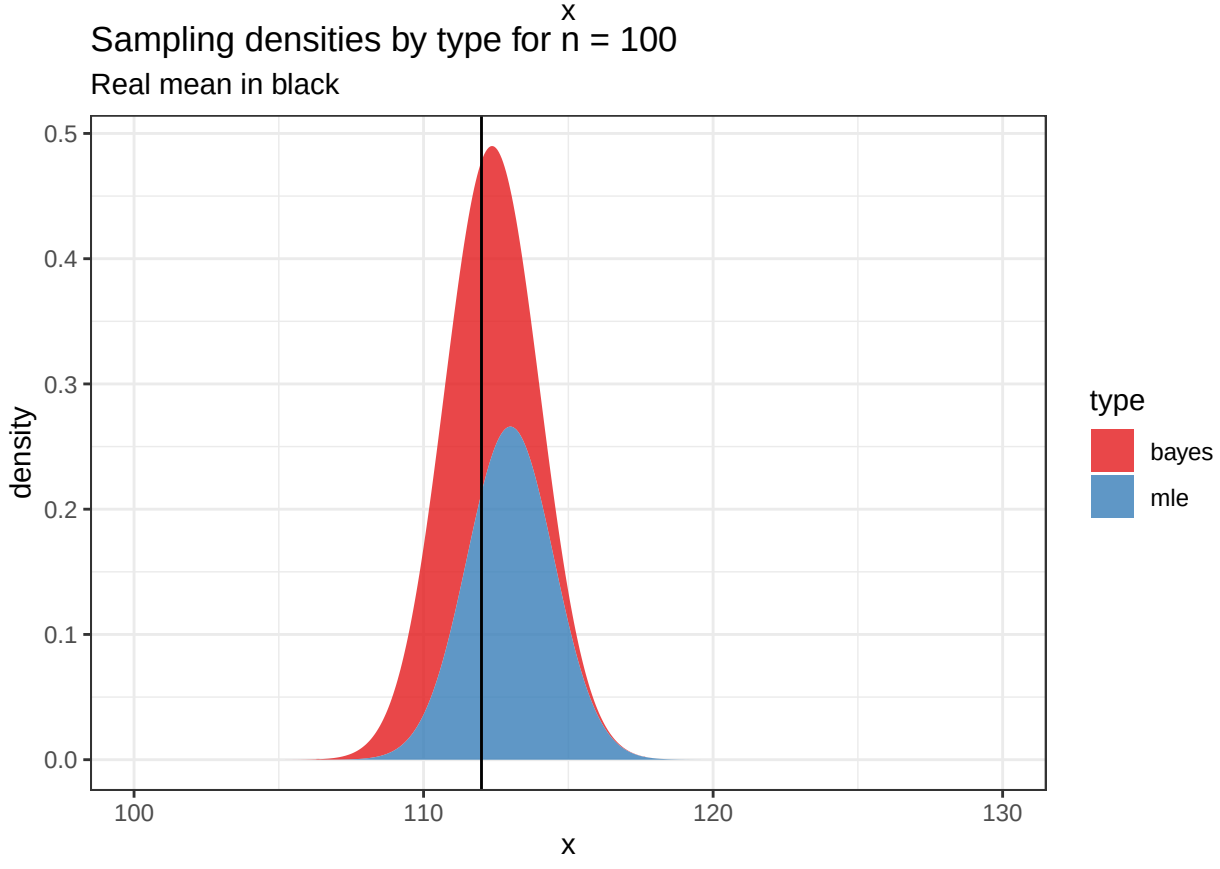
```
mle_mean <- 113
mle_var <- 22.5
bayes_mean <- mu_post
bayes_var <- (100 * 15^4) / (11 * 15^2)^2 * mle_var

x_s <- seq(from = 100, to = 130, by = 0.05)
bayes_samples <- dnorm(x_s, mean = bayes_mean, sd = sqrt(bayes_var))
mle_samples <- dnorm(x_s, mean = mle_mean, sd = sqrt(mle_var))

df_sampling_plot <- tibble(x = x_s, bayes = bayes_samples, mle = mle_samples)

df_sampling_plot %>%
    pivot_longer(cols = c("bayes","mle"), names_to = "type", values_to = "density") %>%
    ggplot(aes(x = x, y = density, fill = type)) +
    scale_fill_brewer(palette = "Set1") +
    geom_area(alpha = 0.8) +
    theme_bw() +
    geom_vline(xintercept = 112, color = "black") +
    labs(title = "Sampling densities by type",
         subtitle = "Real mean in black")
```

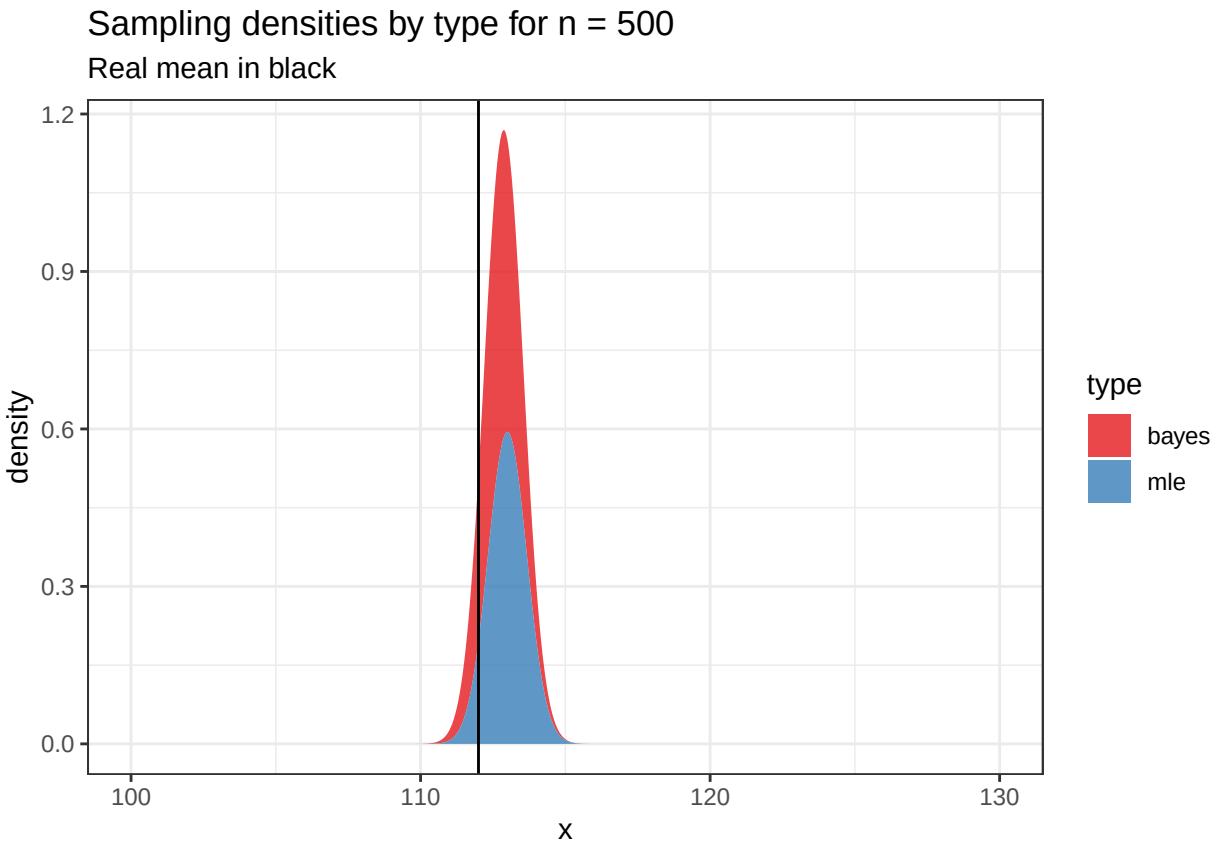



The MLE is in principle an unbiased estimator, but has higher variance. If we can put an informative prior from reliable information the Bayes estimator will have lower variance, and as in this case, could in practice have lower bias as well. All in all Bayes estimator with good prior should have a lower MSE than the ML estimator.

```
for (n in c(50, 100, 500)) {
mle_mean <- 113
mle_var <- (15^2) / n
bayes_var <- (n^2 * 15^4) / ((n + 1) * 15^2)^2 * mle_var
bayes_mean <- ((100 + 113 * n) / 15^2) * bayes_var

x_s <- seq(from = 100, to = 130, by = 0.05)
bayes_samples <- dnorm(x_s, mean = bayes_mean, sd = sqrt(bayes_var))
mle_samples <- dnorm(x_s, mean = mle_mean, sd = sqrt(mle_var))

df_sampling_plot <- tibble(x = x_s, bayes = bayes_samples, mle = mle_samples)

df_sampling_plot %>%
    pivot_longer(cols = c("bayes","mle"), names_to = "type", values_to = "density") %>%
    ggplot(aes(x = x, y = density, fill = type)) +
    scale_fill_brewer(palette = "Set1") +
    geom_area(alpha = 0.8) +
    theme_bw() +
    geom_vline(xintercept = 112, color = "black") +
    labs(title = paste0("Sampling densities by type for n = ", n),
         subtitle = "Real mean in black") -> p
print(p)
}
```


Sampling densities by type for n = 50
Real mean in black
density
0.0
0.1
0.2
0.3
100
110
120
130
x
type
bayes
mle


Sampling densities by type for n = 100
Real mean in black
density
0.0
0.1
0.2
0.3
0.4
0.5
100
110
120
130
x
type
bayes
mle




From the plots with varying n we see that both distributions get tighter around the observed data mean values, but the bayesian posterior is skewed by the prior, especially for lower n.

# Question 2

## a

The proposal is just a draw from the conditional distribution

$$J_t(\theta_j|\theta_{-j}^{s-1}) = p(\theta_j|\theta_{-j}^{s-1})$$

As for $r = 1$ this is easy to see since we are already drawing from the conditional distribution, thus the normalizing constants are the same (since they depend on all components but the one we are changing). Thus it looks something like:

$$r = \frac{p(\theta^s)}{p(\theta^{s-1})} * \frac{J_t(\theta_j^{s-1}|\theta_{-j}^s)}{J_t(\theta_j^s|\theta_{-j}^{s-1})} = \frac{p(\theta^s)}{p(\theta^{s-1})}\frac{p(\theta_j^{s-1}|\theta_{-j}^{s-1})}{p(\theta_j^s|\theta_{-j}^{s-1})} = \frac{p(\theta^s)}{p(\theta^{s-1})}\frac{p(\theta_j^{s-1})}{p(\theta_j^s)}\frac{C(\theta_{-j}^{s-1})}{C(\theta_{-j}^s)} = 1$$

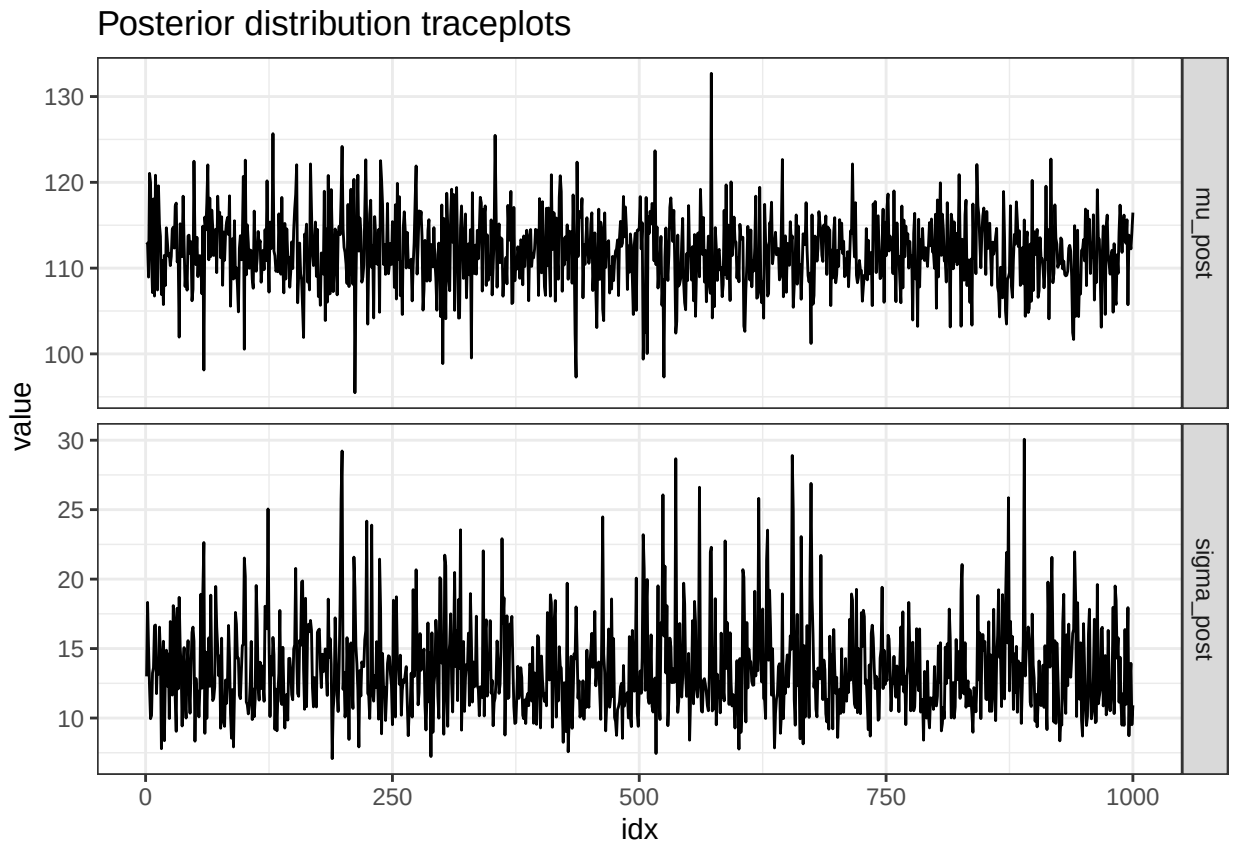
intuitively since we are in the stationary distribution and we take a step in the conditional distribution then we will tay in the stationary distribution so always accept.

## b

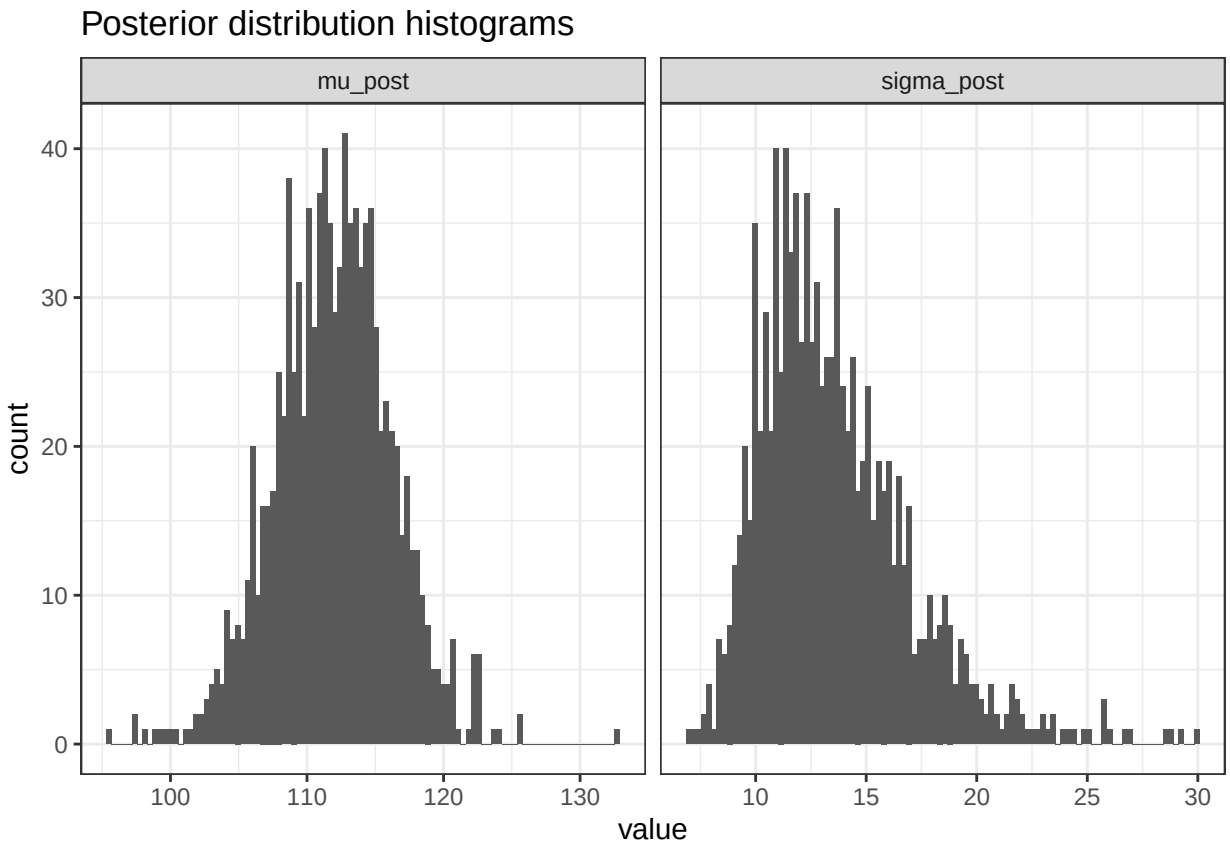
```
compute_mu_mean <- function(sigmasq) {
    top <- (100 / 15^2) + (1130 / sigmasq)
    bottom <- (1/15^2) + (10 / sigmasq)
```

```
    return(top/bottom)
}

compute_mu_var <- function(sigmasq) {
    bottom <- (1/15^2) + (10/sigmasq)
    return(1/bottom)
}

sigma_a <- 11/2

compute_sigma_b <- function(mu) {
    return(0.5 * (15 + 9 * 13^2 + 10 * (113 - mu)^2))
}

sample_mu <- function(sigmasq) {
    mu_mean <- compute_mu_mean(sigmasq)
    mu_var <- compute_mu_var(sigmasq)
    return(rnorm(1, mean = mu_mean, sd = sqrt(mu_var)))
}

sample_sigma <- function(mu) {
    sigma_b <- compute_sigma_b(mu)
    prec_sq <- rgamma(1, shape = sigma_a, rate = sigma_b)
    return(1/prec_sq)
}

sample_mean <- 113
sample_var <- 13^2

mu_post <- rep(NA, 1000)
sigmasq_post <- rep(NA, 1000)

mu_post[1] <- sample_mean
sigmasq_post[1] <- sample_var

for (i in 2:1000) {
    mu_post[i] <- sample_mu(sigmasq_post[i - 1])
    sigmasq_post[i] <- sample_sigma(mu_post[i])
}
```

```
posterior_samples <- tibble(idx = 1:1000, mu_post = mu_post, sigma_post = sqrt(sigmasq_post))

posterior_samples %>%
    pivot_longer(cols = contains("_post"), names_to = "parameter") %>%
    ggplot(aes(x = idx, y = value, facet = parameter)) +
    geom_path() +
    facet_grid(parameter ~ ., scales = "free") +
    theme_bw() +
    labs(title = "Posterior distribution traceplots")
```




```
posterior_samples %>%
    pivot_longer(cols = contains("_post"), names_to = "parameter") %>%
    ggplot(aes(x = value, facet = parameter)) +
    geom_histogram(bins = 100) +
    facet_grid(. ~ parameter, scales = "free") +
    theme_bw() +
    labs(title = "Posterior distribution histograms")
```




```
posterior_samples %>% summarise(mu_ci_lower = quantile(mu_post, 0.025),
                                mu_post_pe = mean(mu_post),
                                mu_ci_upper = quantile(mu_post, 0.975),
                                sigma_ci_lower = quantile(sigma_post, 0.025),
                                sigma_post_pe = mean(sigma_post),
                                sigma_ci_upper = quantile(sigma_post, 0.975),) %>% t() %>%
    kableExtra::kable()
```

| | |
|---|---|
| mu_ci_lower | 103.470828 |
| mu_post_pe | 111.933364 |
| mu_ci_upper | 120.687403 |
| sigma_ci_lower | 8.754188 |
| sigma_post_pe | 13.582805 |
| sigma_ci_upper | 21.928557 |

## Question 3

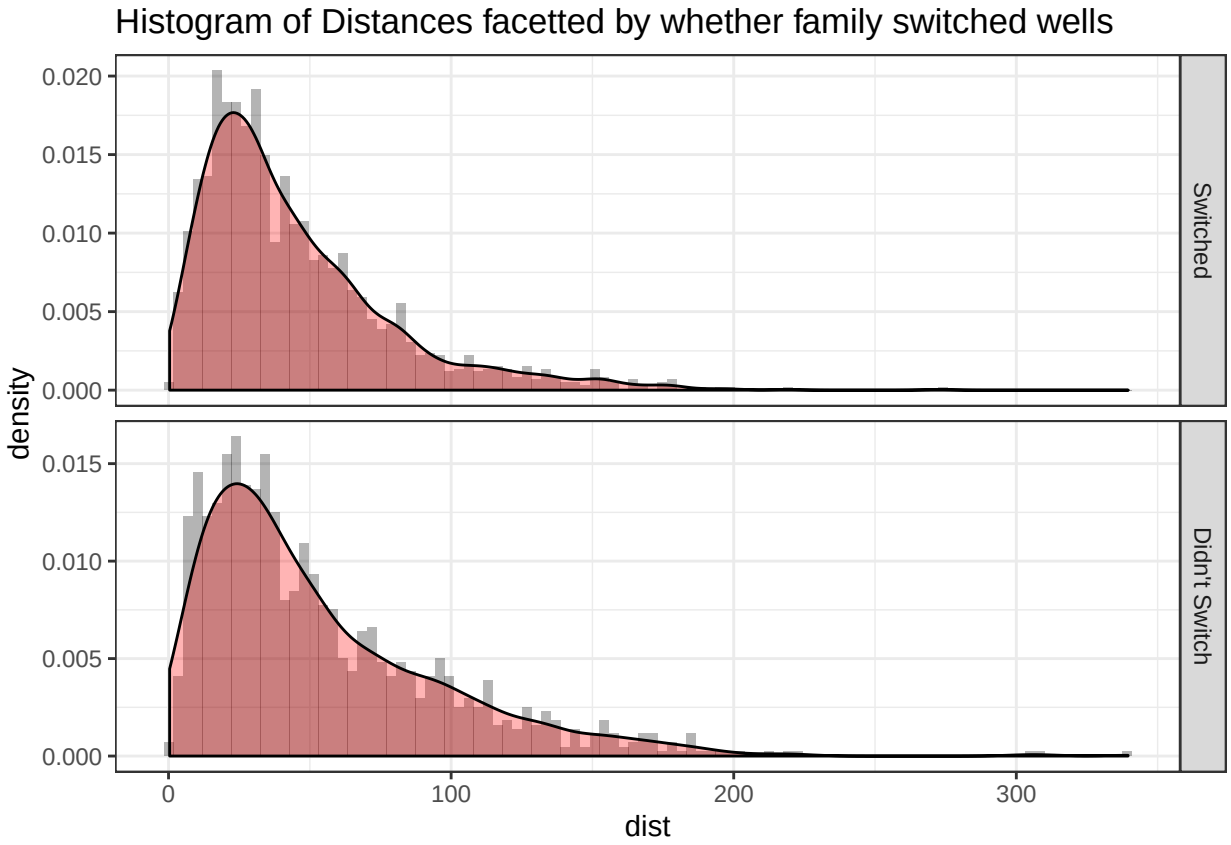
```
df <- read_delim(url("http://www.stat.columbia.edu/~gelman/arm/examples/arsenic/wells.dat"),delim = " "
```

```
## Parsed with column specification:
## cols(
##   id = col_double(),
##   switch = col_double(),
##   arsenic = col_double(),
##   dist = col_double(),
##   assoc = col_double(),
##   educ = col_double()
```

```
## )
```

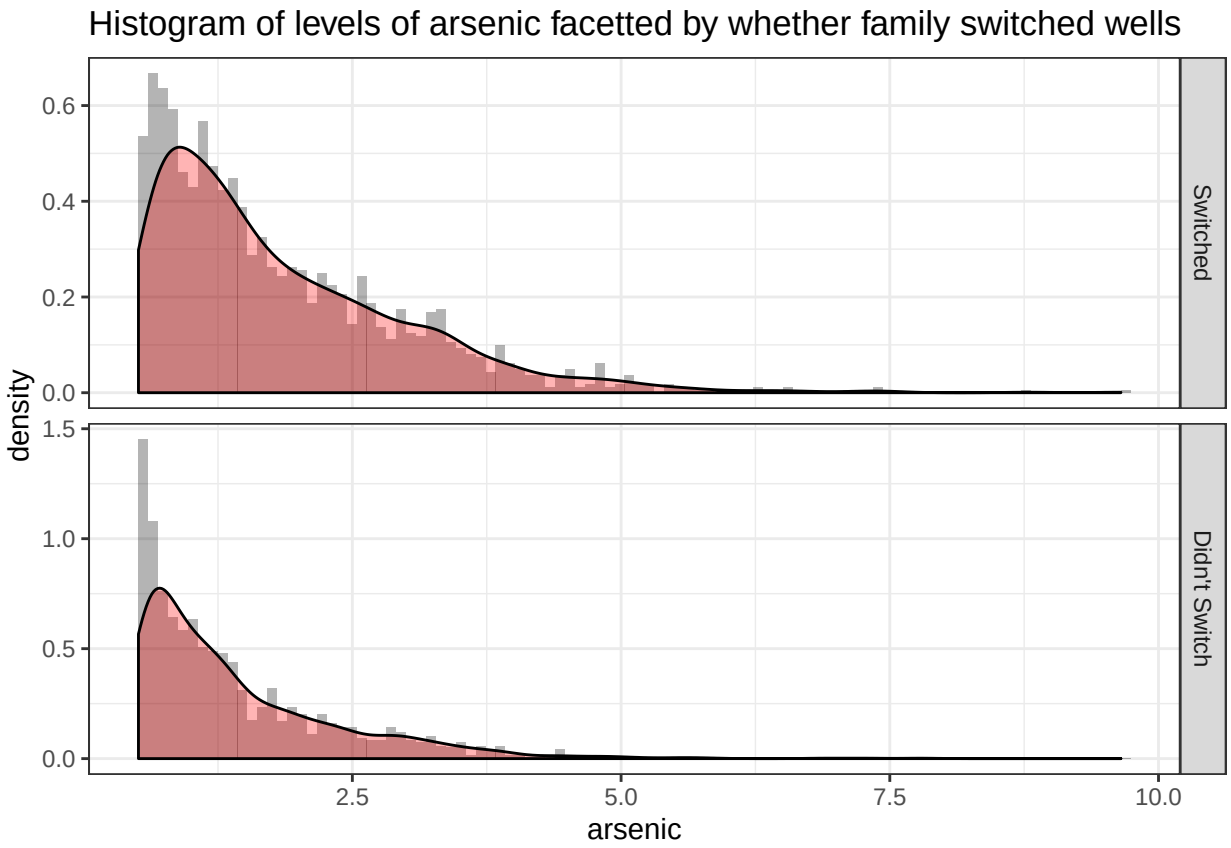
a)

```
# Distance vs Switch
df %>%
    mutate(switch = as_factor(if_else(switch == 0, "Didn't Switch", "Switched"))) %>%
    ggplot(aes(x = dist, facet = switch, ..density..)) +
    geom_histogram(bins = 100, fill = "black", alpha = 0.3) +
    geom_density(fill = "red", alpha = 0.3) +
    facet_grid(switch ~ ., scales = "free_y") +
    theme_bw() +
    labs(title = "Histogram of Distances facetted by whether family switched wells")
```



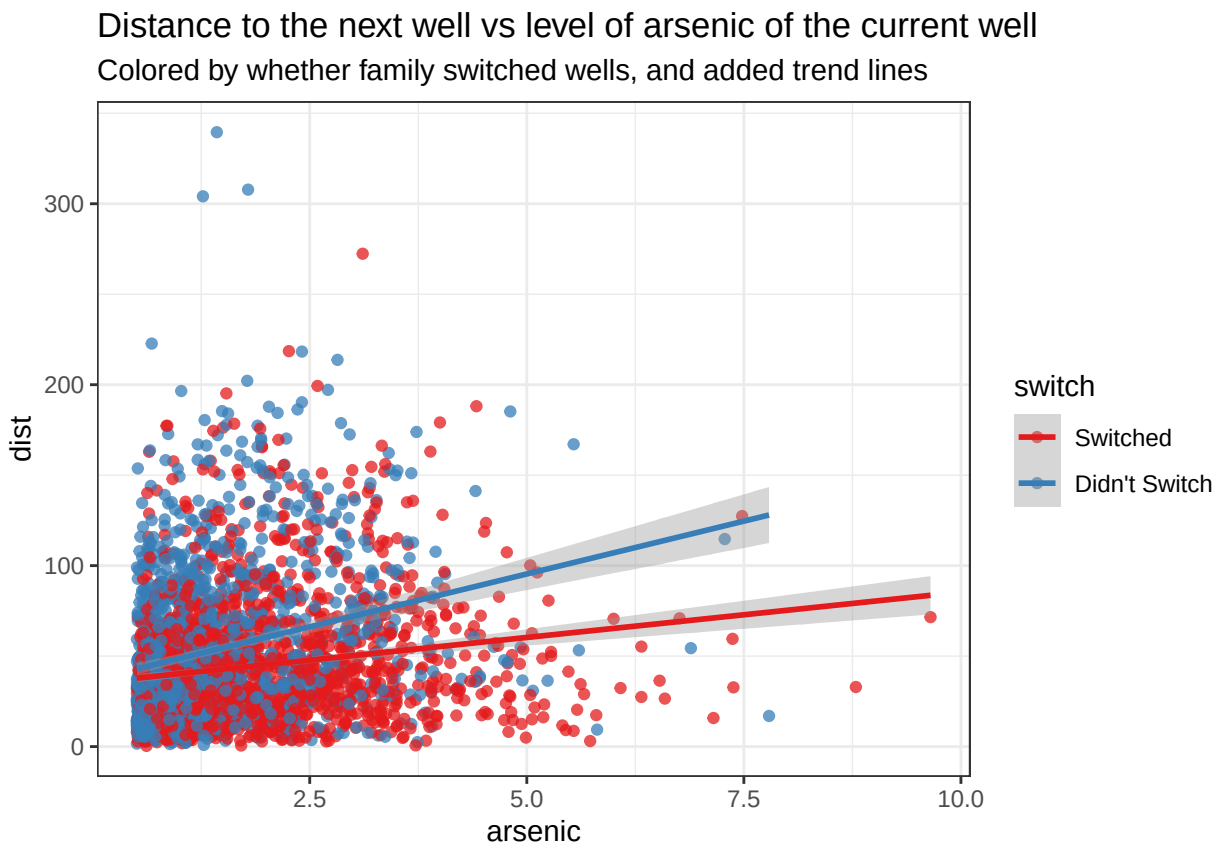

There does not seem to be that much of a difference in distanced between people who switched the wells and those who did not

```
df %>%
    mutate(switch = as_factor(if_else(switch == 0, "Didn't Switch", "Switched"))) %>%
    ggplot(aes(x = arsenic, facet = switch, ..density..)) +
    geom_histogram(bins = 100, fill = "black", alpha = 0.3) +
    geom_density(fill = "red", alpha = 0.3) +
    facet_grid(switch ~ ., scales = "free_y") +
    theme_bw() +
    labs(title = "Histogram of levels of arsenic facetted by whether family switched wells")
```



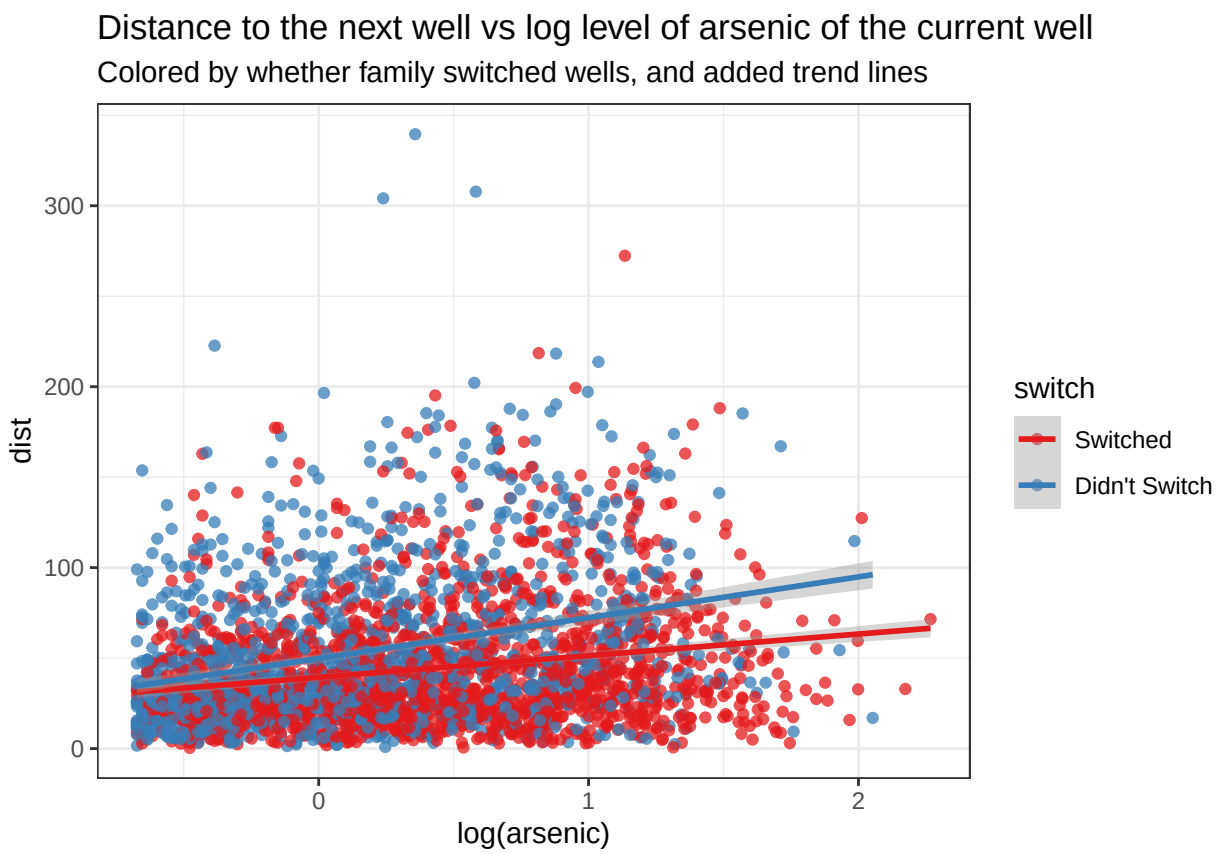

There is a very clear diffference - people with higher levels of arsenic in their wells seem to be much more willing to switch, most of the people who did not switch seem to have had a well that was close to the "safe" level of 0.5

```
df %>%
    mutate(switch = as_factor(if_else(switch == 0, "Didn't Switch", "Switched"))) %>%
    ggplot(aes(x = arsenic, y = dist, color = switch)) +
    geom_point(alpha = 0.75) +
    theme_bw() +
    geom_smooth(method = "lm") +
    scale_color_brewer(palette = "Set1") +
    labs(title = "Distance to the next well vs level of arsenic of the current well",
         subtitle = "Colored by whether family switched wells, and added trend lines")
```



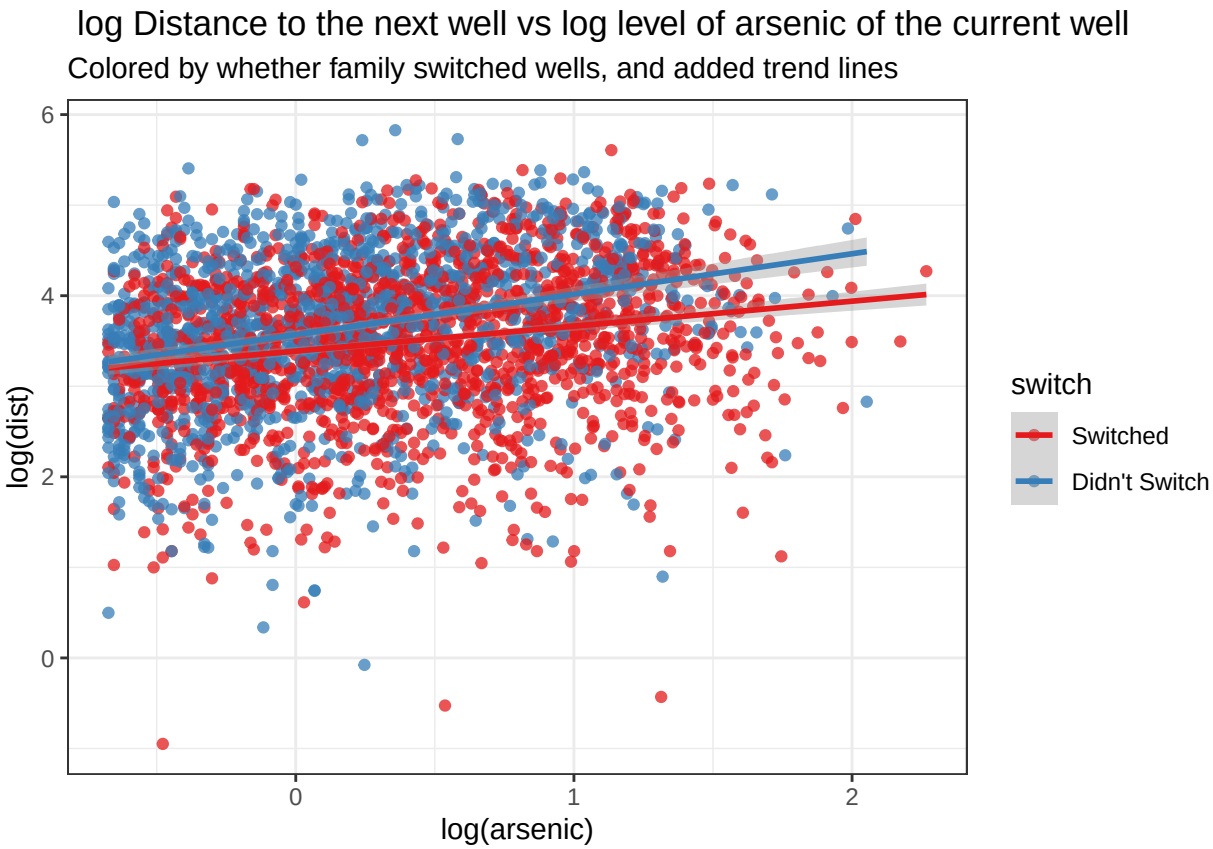

There seems to be a difference in a relationship between distance and arsenic level for people that switched - this makes sense as people with very high arsenic level might want to switch regardless of distance, while people with much lower arsenic level might not want to switch to a well that is very far. The lines clearly have quite different slopes, which would indicate this effect.

```
df %>%
    mutate(switch = as_factor(if_else(switch == 0, "Didn't Switch", "Switched"))) %>%
    ggplot(aes(x = log(arsenic), y = dist, color = switch)) +
    geom_point(alpha = 0.75) +
    theme_bw() +
    geom_smooth(method = "lm") +
    scale_color_brewer(palette = "Set1") +
    labs(title = "Distance to the next well vs log level of arsenic of the current well",
         subtitle = "Colored by whether family switched wells, and added trend lines")
```




Similar to the plot above just trying to see if perhaps there will be more clear of a trend with looking at log arsenic level, and there doesn't seem to be that much more compared to the dist vs arsenic. The slopes are ver clearly different, but there are too many high distance observations for switched for even lower arsenic levels to make the difference in slopes extremely pronounced.

```
df %>%
    mutate(switch = as_factor(if_else(switch == 0, "Didn't Switch", "Switched"))) %>%
    ggplot(aes(x = log(arsenic), y = log(dist), color = switch)) +
    geom_point(alpha = 0.75) +
    theme_bw() +
    geom_smooth(method = "lm") +
    scale_color_brewer(palette = "Set1") +
    labs(title = " log Distance to the next well vs log level of arsenic of the current well",
         subtitle = "Colored by whether family switched wells, and added trend lines")
```




Putting both on the log scale also does not show much change compared to the previoius ones.

## b)

```
# Standardize the variables:
scale2 <- function(x, na.rm = FALSE) (x - mean(x, na.rm = na.rm))


df_stan <- df %>% mutate(d = scale2(dist),
                         a = scale2(arsenic),
                         da = d * a,
                         loga = scale2(log(arsenic)),
                         dloga = d*loga,
                         y = switch)
N = nrow(df_stan)

# Make Stan Data format
stan_data <- list(
    N = N,
    a = df_stan$a,
    d = df_stan$d,
    da = df_stan$da,
    loga = df_stan$loga,
    dloga = df_stan$dloga,
    y = df_stan$y
)
```

```r
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## rstan (Version 2.19.2, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
##
## Attaching package: 'rstan'
```

```
## The following object is masked from 'package:arm':
##
##     traceplot
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```r
library(tidybayes)
library(bayesplot)
```

```
## This is bayesplot version 1.7.1
```

```
## - Online documentation and vignettes at mc-stan.org/bayesplot
```

```
## - bayesplot theme set to bayesplot::theme_default()
```

```
##    * Does _not_ affect other ggplot2 plots
```

```
##    * See ?bayesplot_theme_set for details on theme setting
```

```r
library(loo)
```

```
## This is loo version 2.2.0
```

```
## - Online documentation and vignettes at mc-stan.org/loo
```

```
## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' ar
```

```
##
## Attaching package: 'loo'
```

```
## The following object is masked from 'package:rstan':
##
##     loo
```

```r
# fit model 1
model1 <- stan(file = here("Assignments/Assignment 3/model1.stan"),
               data = stan_data,
               chains = 4,
               iter = 1000)
```

```
##
## SAMPLING FOR MODEL 'model1' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.001599 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 15.99 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
```

```
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 14.7683 seconds (Warm-up)
## Chain 1:                7.29629 seconds (Sampling)
## Chain 1:                22.0646 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'model1' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000791 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 7.91 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 16.2144 seconds (Warm-up)
## Chain 2:                6.24246 seconds (Sampling)
## Chain 2:                22.4568 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'model1' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000716 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 7.16 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
```

```
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 14.0025 seconds (Warm-up)
## Chain 3:                7.13923 seconds (Sampling)
## Chain 3:                21.1418 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'model1' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000731 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 7.31 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 14.6991 seconds (Warm-up)
## Chain 4:                7.79775 seconds (Sampling)
## Chain 4:                22.4969 seconds (Total)
## Chain 4:
```

```
# fit model 2
model2 <- stan(file = here("Assignments/Assignment 3/model2.stan"),
               data = stan_data,
               chains = 4,
               iter = 1000)
```

```
##
## SAMPLING FOR MODEL 'model2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.002002 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 20.02 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
```

```
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 16.0647 seconds (Warm-up)
## Chain 1:                6.42826 seconds (Sampling)
## Chain 1:                22.493 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'model2' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000954 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 9.54 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 12.9362 seconds (Warm-up)
## Chain 2:                6.03082 seconds (Sampling)
## Chain 2:                18.967 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'model2' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.00306 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 30.6 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
```

```
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 13.408 seconds (Warm-up)
## Chain 3:                7.5425 seconds (Sampling)
## Chain 3:                20.9505 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'model2' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000785 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 7.85 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 13.5272 seconds (Warm-up)
## Chain 4:                7.01656 seconds (Sampling)
## Chain 4:                20.5437 seconds (Total)
## Chain 4:
```

coefficient interpretations:

$\beta_0$ : The intercept - The log odds of a family switching the well if their distance and arsenic levels are average $\beta_1$ : Coefficient for distance - The change in log odds as the distance changes from the average $\beta_2$ : Coefficient for arsenic level - The change in log odds as the arsenic level changes from the average $\beta_3$ : Coefficient for the interaction - The change in the change in log odds with distance with arsenic level from the average

```
summary(model1)$summary[c("beta0", "beta1", "beta2", "beta3"),]
```

```
##               mean      se_mean          sd         2.5%          25%
## beta0  0.350491503 1.499397e-03 0.040632624  0.273793617  0.321943312
## beta1  0.468520181 1.504381e-03 0.041917110  0.392857808  0.438457977
## beta2 -0.008743189 2.253775e-05 0.001054593 -0.010863252 -0.009432061
## beta3 -0.001763914 2.122341e-05 0.001023617 -0.003775387 -0.002447118
##                50%          75%         97.5%     n_eff      Rhat
## beta0  0.349905646  0.376786076  0.4365990831  734.3726 1.0048188
## beta1  0.467128442  0.496169910  0.5548809570  776.3669 1.0011077
## beta2 -0.008750573 -0.008014206 -0.0066416098 2189.5189 1.0020612
```

```
## beta3 -0.001724609 -0.001056791  0.0001416386 2326.1857 0.9992552
```

```
summary(model2)$summary[c("beta0", "beta1", "beta2", "beta3"),]
```

```
##               mean      se_mean          sd         2.5%          25%
## beta0  0.342148752 1.356568e-03 0.039986905  0.265041843  0.315384419
## beta1  0.873986667 2.471168e-03 0.067060944  0.740998431  0.828243422
## beta2 -0.009483441 2.458148e-05 0.001092615 -0.011639206 -0.010213329
## beta3 -0.002336567 4.200586e-05 0.001828784 -0.005919769 -0.003558964
##                50%          75%        97.5%     n_eff     Rhat
## beta0  0.341720552  0.368643293  0.423938739  868.8648 1.006371
## beta1  0.873836558  0.918381623  1.013390525  736.4356 1.006114
## beta2 -0.009496074 -0.008748191 -0.007330368 1975.6881 1.000770
## beta3 -0.002307237 -0.001130053  0.001125652 1895.4178 1.000904
```

Rhats and n_eff look pretty good so I won't make the chains longer.

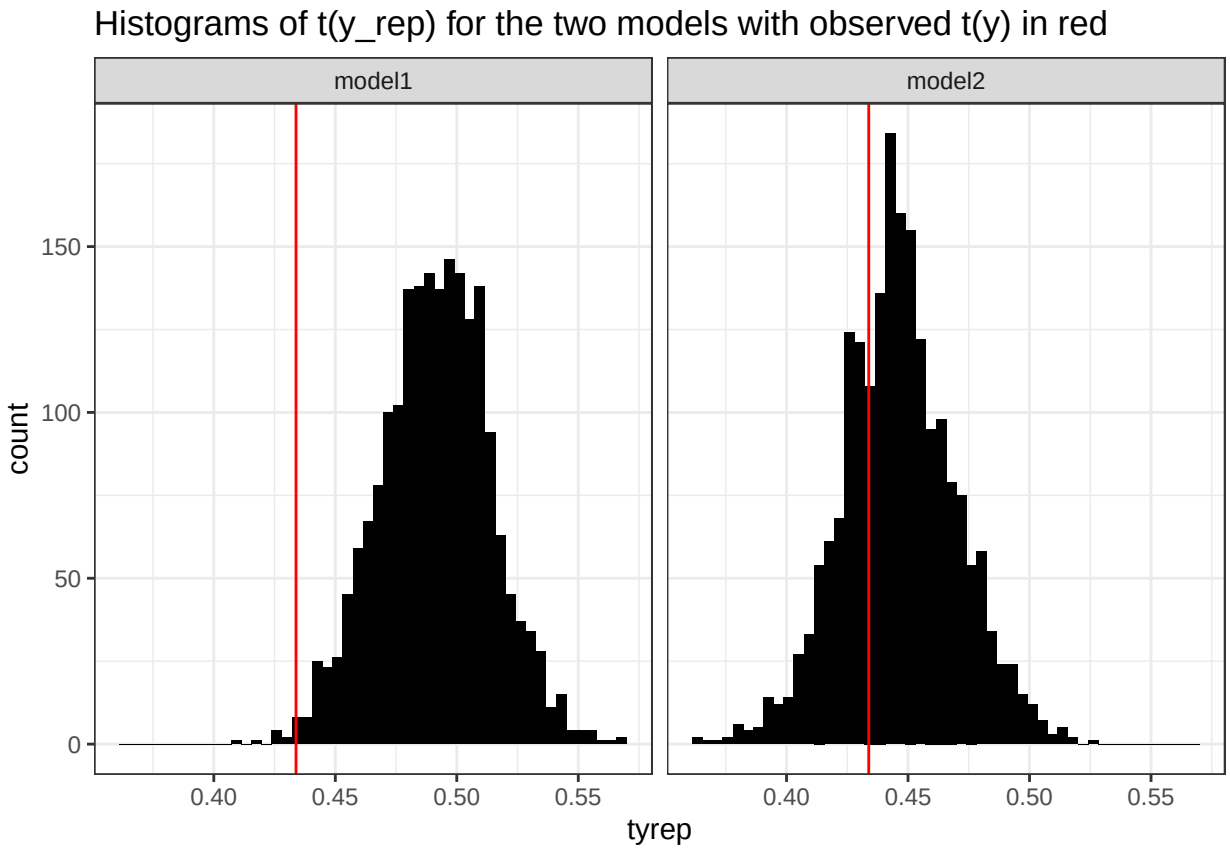
**c**

```
ty_id <- df %>% filter(arsenic < 0.82) %>% pull(id)

# Compute t(y)
ty <- df %>% slice(ty_id) %>% summarise(mean(switch)) %>% pull()

yrep1 <- extract(model1)[["y_rep"]] %>% t() %>% as_tibble()
```

```
## Warning: `as_tibble.matrix()` requires a matrix with column names or a `.name_repair` argument. Using
## This warning is displayed once per session.
```

```
yrep2 <- extract(model2)[["y_rep"]] %>% t() %>% as_tibble()

ty_rep1 <- yrep1 %>% slice(ty_id) %>% summarise_all(mean) %>% t()
ty_rep1 <- ty_rep1[,1] %>% as_vector() %>% unname()

ty_rep2 <- yrep2 %>% slice(ty_id) %>% summarise_all(mean) %>% t()
ty_rep2 <- ty_rep2[,1] %>% as_vector() %>% unname()
```

```
df_tyreps <- tibble(model1 = ty_rep1, model2 = ty_rep2)

df_tyreps %>%
    pivot_longer(cols = everything(), names_to = "model", values_to = "tyrep") %>%
    mutate(model = as_factor(model)) %>%
    ggplot(aes(x = tyrep)) +
    geom_histogram(bins = 50, fill = "black") +
    geom_vline(xintercept = ty, color = "red") +
    facet_wrap(.~model) +
    theme_bw() +
    labs(title = "Histograms of t(y_rep) for the two models with observed t(y) in red")
```




```
df_tyreps %>%
    pivot_longer(cols = everything(), names_to = "model", values_to = "tyrep") %>%
    mutate(model = as_factor(model)) %>%
    group_by(model) %>%
    summarise(prob = mean(tyrep < ty)) %>%
    kableExtra::kable()
```

| model | prob |
|---|---|
| model1 | 0.0040 |
| model2 | 0.2905 |

For both models the probabilities are low, but for model 1 it is significantly lower than for model 2. This means that both models vastly overestimate the proportion of people with wells with arsenic levels under 0.82 that will switch.

## d

```
log_lik_1 <- extract_log_lik(model1)
log_lik_2 <- extract_log_lik(model2)


loo1 <- loo(log_lik_1, save_psis = TRUE)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

```
loo2 <- loo(log_lik_2, save_psis = TRUE)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

```
comp <- loo_compare(loo1, loo2)
print(comp)
```

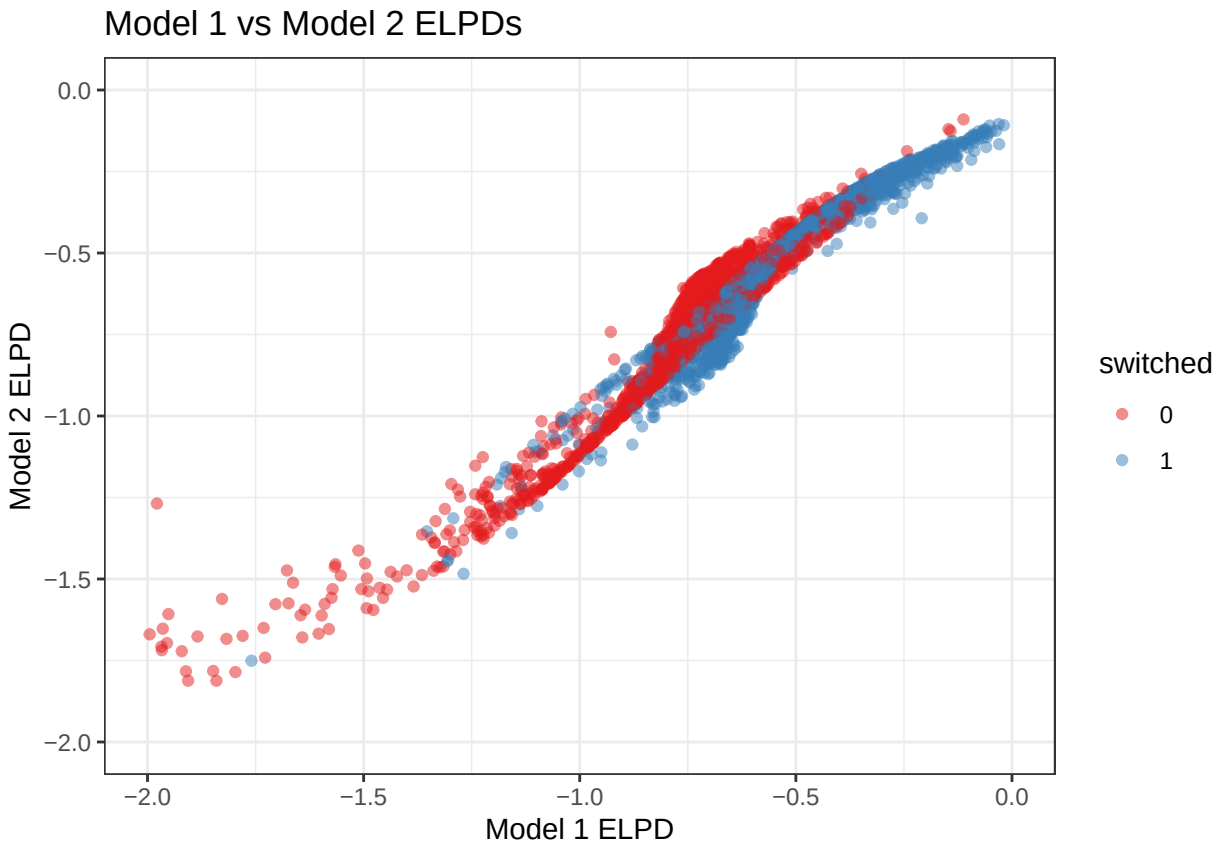
```
##        elpd_diff se_diff
## model2   0.0       0.0
## model1 -15.7       4.4
```

We see that the second model is better (higher elpd).

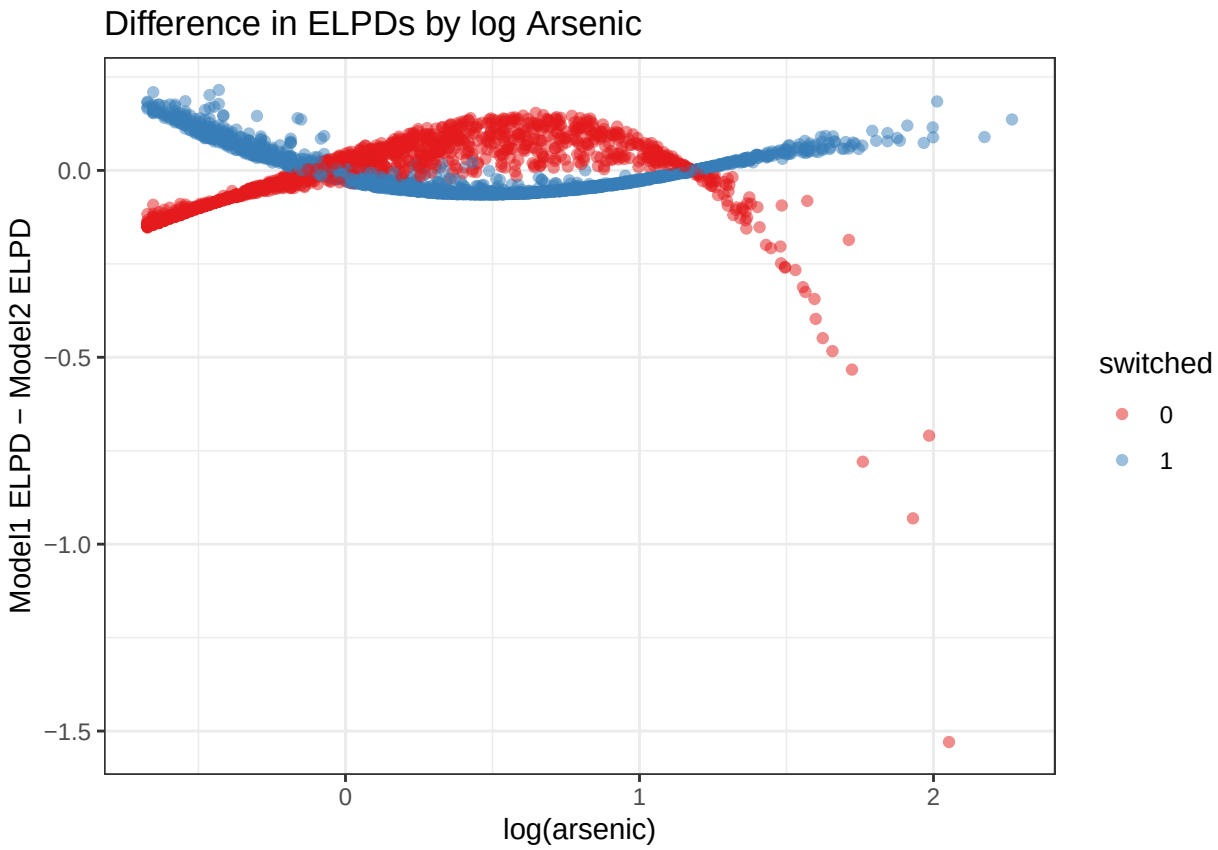
## e)

```
df_elpd_plots <- tibble(model1 = loo1$pointwise[,1], model2 = loo2$pointwise[,1], loga = log(df$arsenic

df_elpd_plots <- df_elpd_plots %>%
    mutate(diff = model1 - model2)

df_elpd_plots %>%
    ggplot(aes(x = model1, y = model2, color = switched)) +
    geom_point(alpha = 0.5) +
    scale_color_brewer(palette = "Set1") +
    theme_bw() +
    scale_x_continuous(name = "Model 1 ELPD", limits = c(-2,0)) +
    scale_y_continuous(name = "Model 2 ELPD", limits = c(-2,0)) +
    labs(title = "Model 1 vs Model 2 ELPDs")
```

```
## Warning: Removed 9 rows containing missing values (geom_point).
```




```
df_elpd_plots %>%
    ggplot(aes(x = loga, y = diff, color = switched)) +
    geom_point(alpha = 0.5) +
    scale_color_brewer(palette = "Set1") +
    theme_bw() +
    scale_x_continuous(name = "log(arsenic)") +
    scale_y_continuous(name = "Model1 ELPD - Model2 ELPD") +
    labs(title = "Difference in ELPDs by log Arsenic")
```




interpretations:

I cut off 10 points for the first plot to make it much more readable, it seems that there is some mild pattern for both models - for non-switched both have on average lower ELPD and for switched both have higher. There is some convolved pattern around where the two start mixing but nothing I would be comfortable talking about since it looks like a blob.

For the second plot it's clear that for low and high values of log arsenic model 1 performs better for switched and worse for non-switched, and in the middle the opposite - model 2 is better for switched and worse for non-switched.

## f

$elpd_{loo} = log(p(y_i|y_{-i}))$

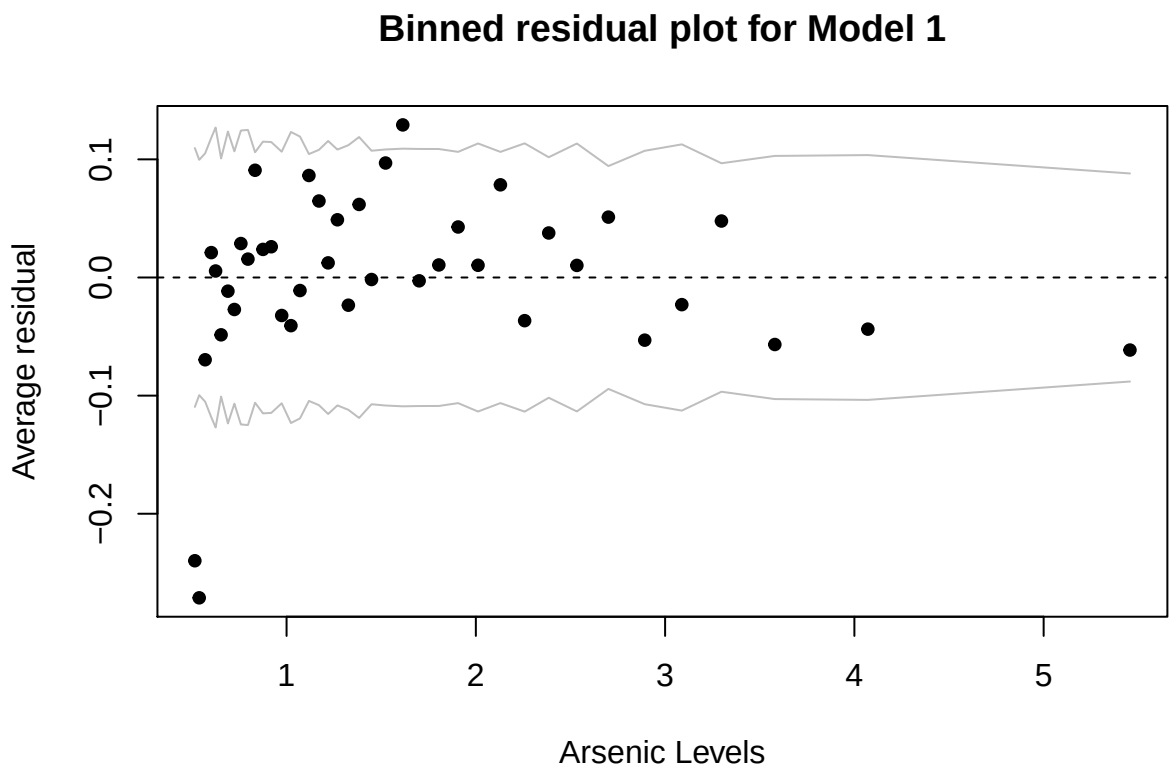
$exp(elpd_{loo}) = p(y_i|y_{-i})$

For a single $elpd_{loo}$ it seems to be the probabilty that the model prediction is correct. So it's the probability of the model predicting $y = 0$ when it there was no well switch, and predicting 1 when there was.

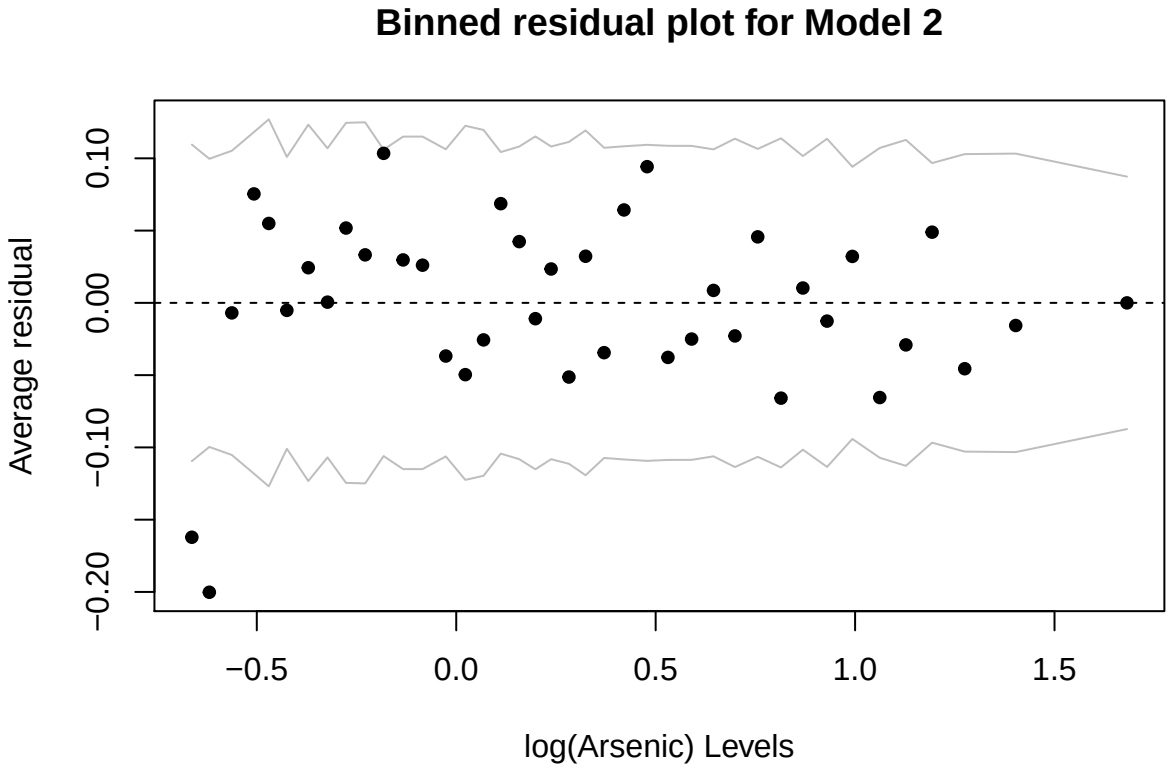
## g

```
# Model 1

df_elpd_mod1 <- df_elpd_plots %>%
    dplyr::select(-model2) %>%
    mutate(model1 = case_when(
        switched == 1 ~ exp(model1),
        switched != 1 ~ 1-exp(model1)),
```

```
        switched = as.numeric(as.character(switched)),
        resid = switched - model1,
        a = exp(loga))

arm::binnedplot(x = df_elpd_mod1$a,
                y = df_elpd_mod1$resid,
                nclass = 40,
                xlab = "Arsenic Levels",
                main = "Binned residual plot for Model 1")
```




```
# Model 2
df_elpd_mod2 <- df_elpd_plots %>%
    dplyr::select(-model1) %>%
    mutate(model2 = case_when(
        switched == 1 ~ exp(model2),
        switched != 1 ~ 1-exp(model2)),
        switched = as.numeric(as.character(switched)),
        resid = switched - model2)

arm::binnedplot(x = df_elpd_mod2$loga,
                y = df_elpd_mod2$resid,
                nclass = 40,
                xlab = "log(Arsenic) Levels",
                main = "Binned residual plot for Model 2")
```




Both binned residual plots look very similar and don't show much in terms of trend except for the two observations with very low arsenic levels that have much more negative residuals than the rest. The other residuals seem to behave quite well for both models: centered around zero, no obvious patterns etc. Overall, pretty good.