STA2201H Winter 2023 Assignment 2

Due: 11:59pm ET, March 10

What to hand in: .Rmd or .qmd file and the compiled pdf, and any stan files

How to hand in: Submit files via Quercus

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.1.0
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
options(dplyr.summarise.inform = FALSE)
```

1 IQ

Suppose we are to sample n individuals from a particular town and then estimate μ, the town-specific mean IQ score, based on the sample of size n. Let Y_i denote the IQ score for the ith person in the town of interest, and assume

$$Y_1, Y_2, \ldots, Y_n | \mu, \sigma^2 \sim N\left(\mu, \sigma^2\right)$$

For this question, will assume that the onserved standard deviation of the IQ scores in the town is equal to 15, the observed mean is equal to 113 and the number of observations is equal to 10. Additionally, for Bayesian inference, the following prior will be used:

$$\mu \sim N\left(\mu_0, \sigma_{\mu 0}^2\right)$$

with $\mu_0 = 100$ and $\sigma_{\mu 0} = 15$.

a) Write down the posterior distribution of μ based on the information above. Give the Bayesian point estimate and a 95% credible interval of μ, $\hat{\mu}_{Bayes} = E(\mu|\boldsymbol{y})$.

a)

we are interested in :

$\vec{y}|\mu \sim N(\mu, 15^2)$

$\mu \sim N(100, 15^2)$

$$P(\mu|\vec{y}) = \frac{P(\vec{y}|\mu) \cdot P(\mu)}{\int P(\vec{y}|\mu')\, d\mu'}$$

$$\propto P(\vec{y}|\mu) \cdot P(\mu).$$

$$= \exp\left(-\frac{1}{2}\frac{\sum_{i=1}^{n}(y_i-\mu)^2}{15^2}\right) \cdot \exp\left(-\frac{1}{2}\frac{(\mu-100)^2}{15^2}\right).$$

$$= \exp\left(-\frac{1}{2}\frac{\Sigma(y_i^2+\mu^2)-2\mu\Sigma y_i}{15^2} - \frac{1}{2}\frac{\mu^2+10000-200\mu}{15^2}\right)$$

$\Sigma y_i = \bar{y} \cdot n = 113 \cdot 10 = 1130.$

$$\propto \exp\left(-\frac{1}{2}\frac{10\mu^2-2260\mu+\mu^2-200\mu}{15^2}\right)$$

$$= \exp\left(-\frac{1}{2}\frac{11\mu^2-2460\mu}{15^2}\right)$$

$$\propto \exp\left(-\frac{1}{2}\frac{(\mu-\frac{1230}{11})^2}{(15/\sqrt{11})^2}\right)$$

$\therefore$ as from this, we have $\mu \sim N\left(\frac{1230}{11}, \overset{\sigma^2}{15^2/11}\right)$

one can rewrite this into $\frac{\mu_0/\sigma_{\mu_0}^2 + n\bar{y}/\sigma^2}{1/\mu_{\sigma_0}^2 + n/\sigma^2}$

one can also rewrite this into $\frac{1}{1/\sigma^2_{\mu_0} + n/\sigma^2}$.

$\therefore$ Bayesian point estimate is $\hat{\mu}_{bayes} = \frac{1230}{11} \approx 111.8$

and a 95% Credible interval is

$$\left(\frac{1230}{11} - z_{97.5\%}\, 15/\sqrt{11},\ \frac{1230}{11} + z_{97.5\%}\, 15/\sqrt{11}\right)$$

$\approx (103.0, 120.7)$

We will now compare the sampling properties of the Bayes estimator to the sample mean, which is the ML estimator.

b) Suppose that (unknown to us) the true mean IQ score is μ^*. To evaluate how close an estimator is to the truth, we might want to use the mean squared error (MSE) $\text{MSE}\left[\hat{\mu}|\mu^*\right] = E\left[\left(\hat{\mu} - \mu^*\right)^2 |\mu^*\right]$. Show the MSE is equal to the variance of the estimator plus the bias of the estimator squared, i.e.

$$\text{MSE}\left[\hat{\mu}|\mu^*\right] = \text{Var}\left[\hat{\mu}|\mu^*\right] + \text{Bias}\left(\hat{\mu}|\mu^*\right)^2$$

b)

$$\mathbb{E}[(\hat{\mu}-\mu^*)^2 \mid \mu^*]$$

$$= \mathbb{E}[\hat{\mu}^2 + \mu^{*2} - 2\hat{\mu}\mu^* \mid \mu^*]$$

$$= \mathbb{E}[\hat{\mu}^2 \mid \mu^*] + \mu^{*2} - 2\mathbb{E}[\hat{\mu} \mid \mu^*] \cdot \mu^*.$$

$$= \mathbb{E}[\hat{\mu}^2 \mid \mu^*] - (\mathbb{E}[\hat{\mu} \mid \mu^*])^2 + (\mathbb{E}[\hat{\mu} \mid \mu^*])^2 - 2\mathbb{E}[\hat{\mu} \mid \mu^*] \cdot \mu^* + \mu^{*2}$$

$$= \operatorname{var}(\hat{\mu} \mid \mu^*) + (\mathbb{E}[\hat{\mu} \mid \mu^*] - \mu^*)^2$$

$$= \operatorname{Var}(\hat{\mu} \mid \mu^*) + \operatorname{Bias}(\hat{\mu} \mid \mu^*)^2. \quad \triangle.$$

c) Suppose that the true mean IQ score is 112. Calculate the bias, variance and MSE of the Bayes and ML estimates. Which estimate has a larger bias? Which estimate has a larger MSE?

c)

for Bayes:

Bias $= 111.8 - 112 = 0.2$.

Variance $= 15^2/11 \approx 20.45$,

$\therefore$ MSE $= 0.2^2 + 20.45 \approx 20.49$.

for ML.

Bias $= E[\bar{x} - \mu] = 0$ $\leftarrow$ always

variance $=$ var$(\bar{x}) = \frac{1}{10}\sigma^2 = 22.5$.

$\therefore$ MSE $= 23.5$.

ML estimator has a smaller bias but a larger MSE.

d) Write down the sampling distributions for the ML and Bayes estimates, again assuming

$\mu^* = 112$ and $\sigma = 15$. Plot the two distributions on the one graph. Summarize your understanding of the differences in bias, variance and MSE of the two estimators by describing how these differences relate to differences in the sampling distributions as plotted. To further illustrate the point, obtain the Bayes and ML MSEs for increasing sample sizes and plot the ratio (Bayes MSE)/(ML MSE) against sample size.

(d) (i)

So for ML algorith:

$Y_i \sim N(\mu^*, \sigma^2) = N(112, \sigma^2)$

$\hat{\mu}_{ML} = \bar{Y} \sim N(\mu^*, \sigma^2/n) = N(112, \sigma^2/10) = N(112, 15^2/10)$.

and for Bayes estimates, as we have proved in a)

$\hat{\mu}_{bayes} \sim N(\frac{1230}{11}, 15^2/11)$

(ii) for MSE of ML:

we have MSE $= \quad 0^2 \quad + 15^2/n$

for MSE of Bayes:

we have $MSE = (\frac{n\bar{y}+100}{n+1} - 112)^2 + \frac{15^2}{n+1}$

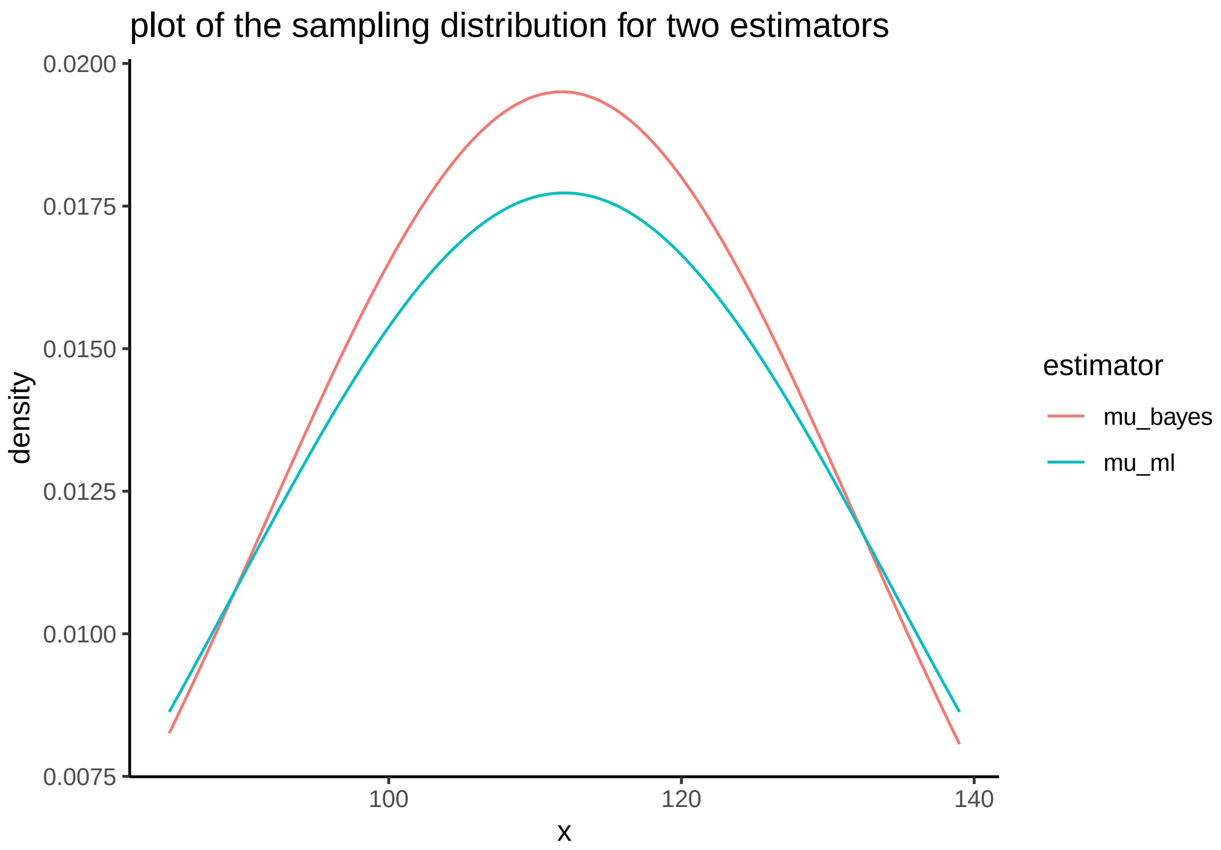
the ratio between the 2:

$$\frac{MSE\ Bayes}{MSE\ ML} = \frac{(\frac{n\bar{y}+100}{n+1} - 112)^2 + \frac{15^2}{n+1}}{0^2 + 15^2/n} \underset{\text{plug in } \bar{y} = 113.}{=} \frac{(\frac{113n+100}{n+1} - 112)^2 + \frac{15^2}{n+1}}{15^2/n}.$$

and we can plot them in R:

Let's plot the sampling distributions below:

```r
# using the 95% information we acquired from a), set X values to be between 85 to 139

x.values = seq(85,139, length = 1000)
mu_ml <- dnorm(x.values, 112, 15^2/10)
mu_bayes <- dnorm(x.values, 1230/11, 15^2/11)

d <- tibble(x = x.values, mu_ml = mu_ml, mu_bayes = mu_bayes)

d |> pivot_longer(mu_ml:mu_bayes, names_to = "estimator", values_to = 'density') |>
  ggplot(aes(x, density, color = estimator)) +
  geom_line()+
  ggtitle('plot of the sampling distribution for two estimators')+
  theme_classic()
```

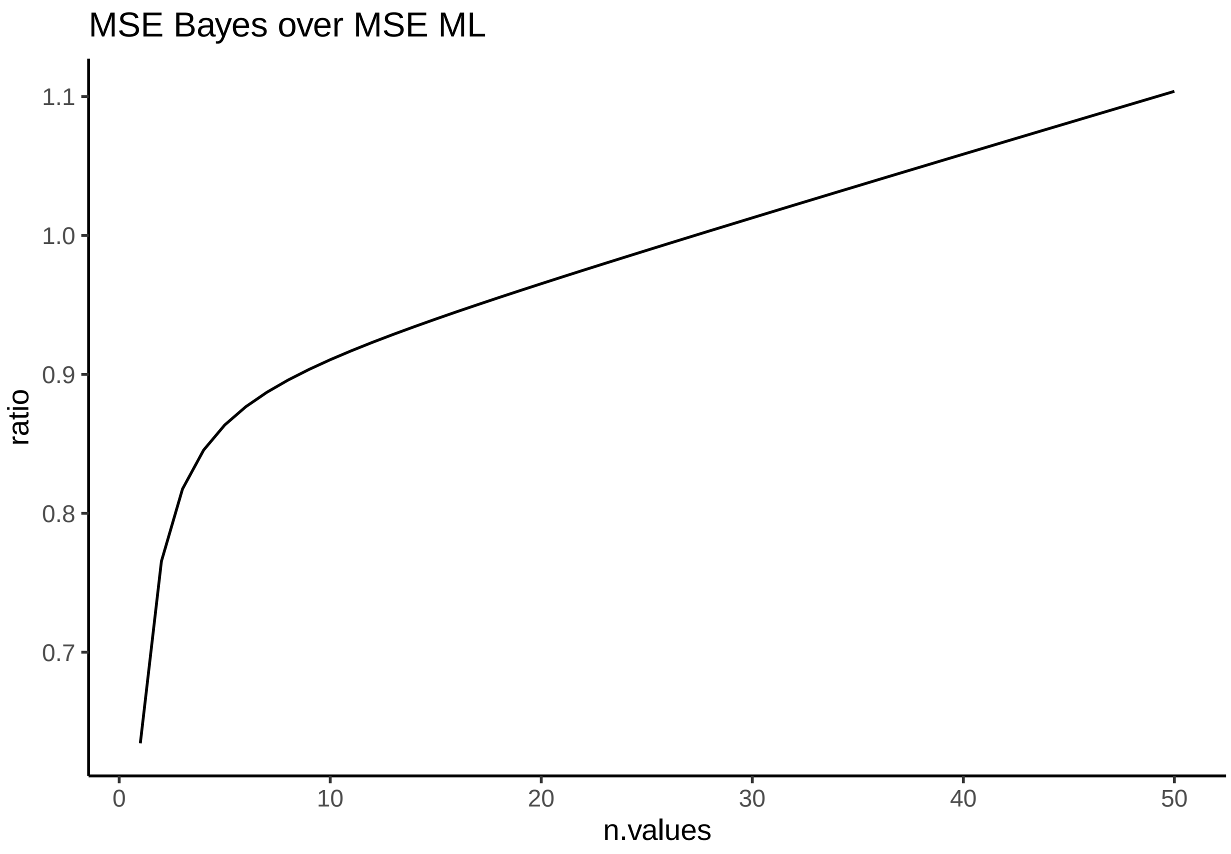


as we can see from the above plot, the mu_bayes estimator on average has a smaller variance, and a higher density near the true mean value of 112, hence contributing to both a smaller Bias and a smaller variance at mu* = 112,and hence a smaller MSE. if however, mu* changes, then it is hard to determine which estimator has a smaller MSe as the mu_ML outperforms mu_bayes at some part on the graph.

Now to further illustrate this, let's look at the ratio of MSE bayes over MSE Ml:

```r
n.values = seq(1,50, by = 1)
```

```r
MSE_ratio <- function(n){
  (15^2/(n+1)+((113*n+100)/(n+1) - 112)^2)/(15^2/n)
}

ratio <- MSE_ratio(n.values)

tibble(n.values, ratio) |>
  ggplot(aes(x= n.values, y = ratio))+
  geom_line()+
  ggtitle("MSE Bayes over MSE ML")+
  theme_classic()
```



as we can see, with a y_bar of 112 fixed, Bayes estimator significantly outperforms ML estimator when n is less than 30, but when n gets larger, such out-performance decreases and eventually becomes worse.

2 Gompertz

Gompertz hazards are of the form

$$\mu_x = \alpha e^{\beta x}$$

for $x \in [0, \infty)$ with $\alpha, \beta > 0$. It is named after Benjamin Gompertz, who suggested a similar form to capture a 'law of human mortality' in 1825.

This question uses data on deaths by age in Sweden over time. The data are in the `sweden` file in the class repo. I grabbed the data from the Human Mortality Database.

We will assume that the deaths we observe in a particular age group are Poisson distributed with a rate equal to the mortality rate multiplied by the population, i.e.

$$D_x \sim \text{Poisson}(\mu_x P_x)$$

where x refers to age. In this question we will be estimating mortality rates using the Gompertz model as described above. let's first read in the data:

```
data <- read.csv('sweden.csv')
```

a) Describe, with the aid of a couple of graphs, some key observations of how mortality above age 50 in Sweden has changed over time.

let's first see the range of the age group:

```
min(data$age)
```

```
## [1] 50
```

```
max(data$age)
```

```
## [1] 100
```

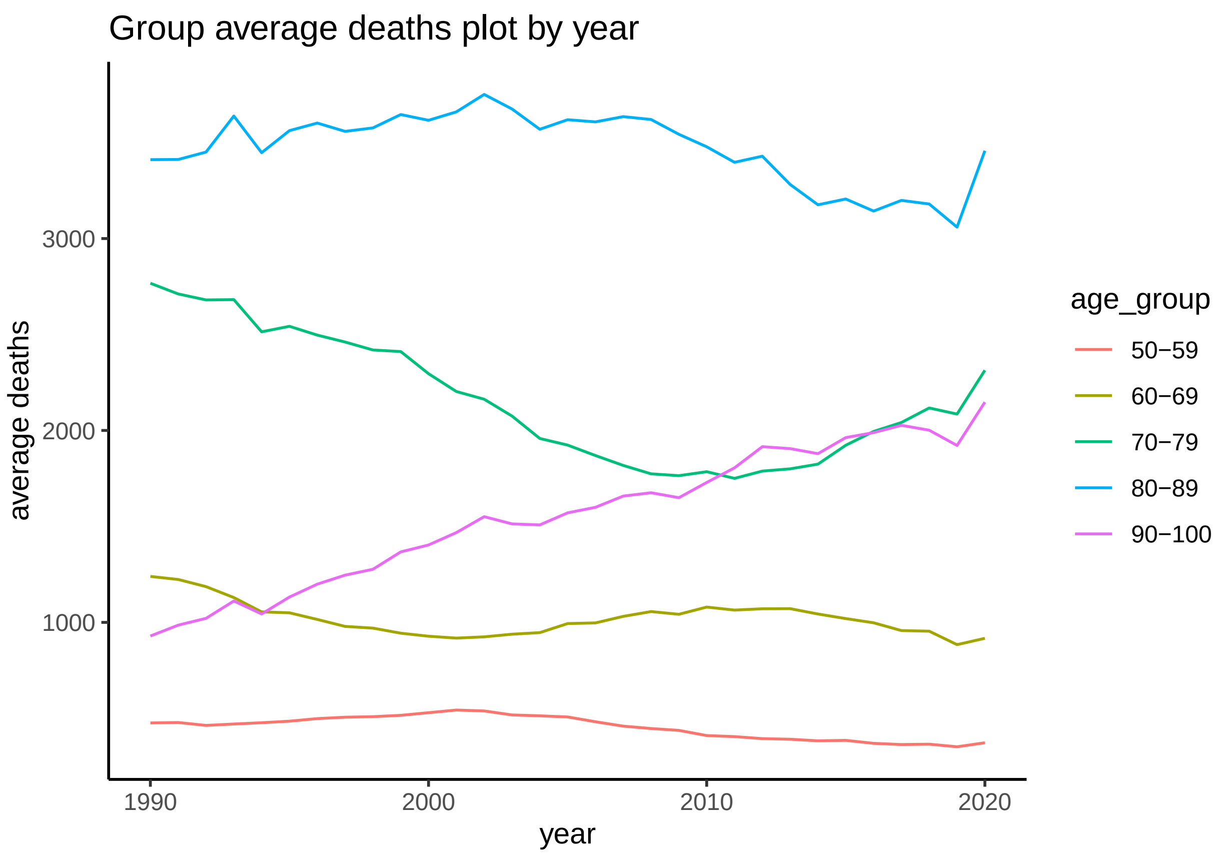
as we can see from the results above, the minimum age of mortality is 50 in the data, and maximum is 100 ages old.

Let's divide the age group into 5 groups, 50-59,60-69,70-79,80-89,90-100:

```
library(dplyr)
df <- data |>
  mutate(
    age_group = case_when(
      age >= 50 & age <= 59 ~ "50-59",
      age >= 60 & age <= 69 ~ "60-69",
      age >= 70 & age <= 79 ~ "70-79",
      age >= 80 & age <= 89 ~ "80-89",
      age >= 90 & age <= 100 ~ "90-100",
    ),
    age_group = factor(
      age_group,
      level = c("50-59", "60-69","70-79", "80-89", "90-100")
    )
  )
```

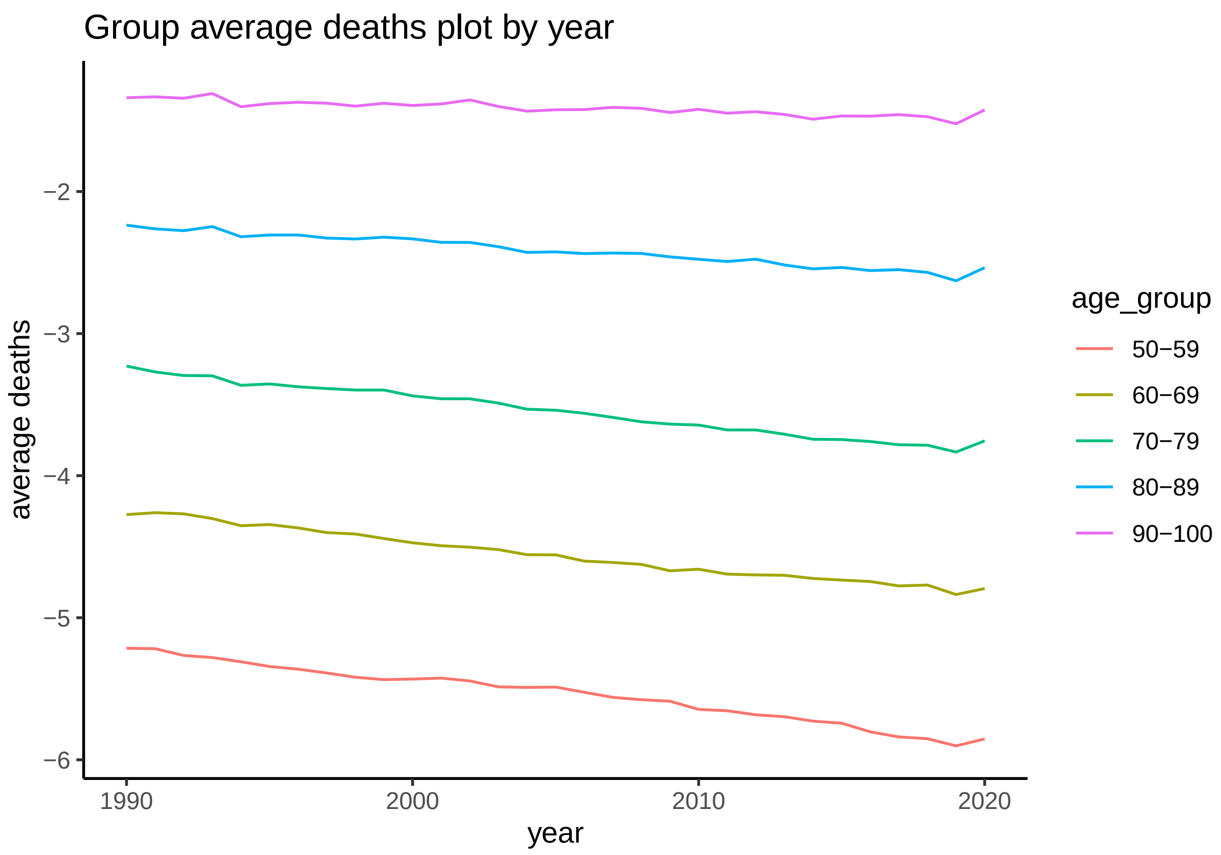
and we can make a plot of the log death number, filtered by age group, averaged on the group mean:

```
df |>
  group_by(age_group,year) |>
  summarise(group_avg_death = mean(deaths)) |>
  ggplot(aes(x = year, y = group_avg_death, color = age_group)) +
  geom_line() +
  labs(title = 'Group average deaths plot by year', y = "average deaths")+
  theme_classic()
```



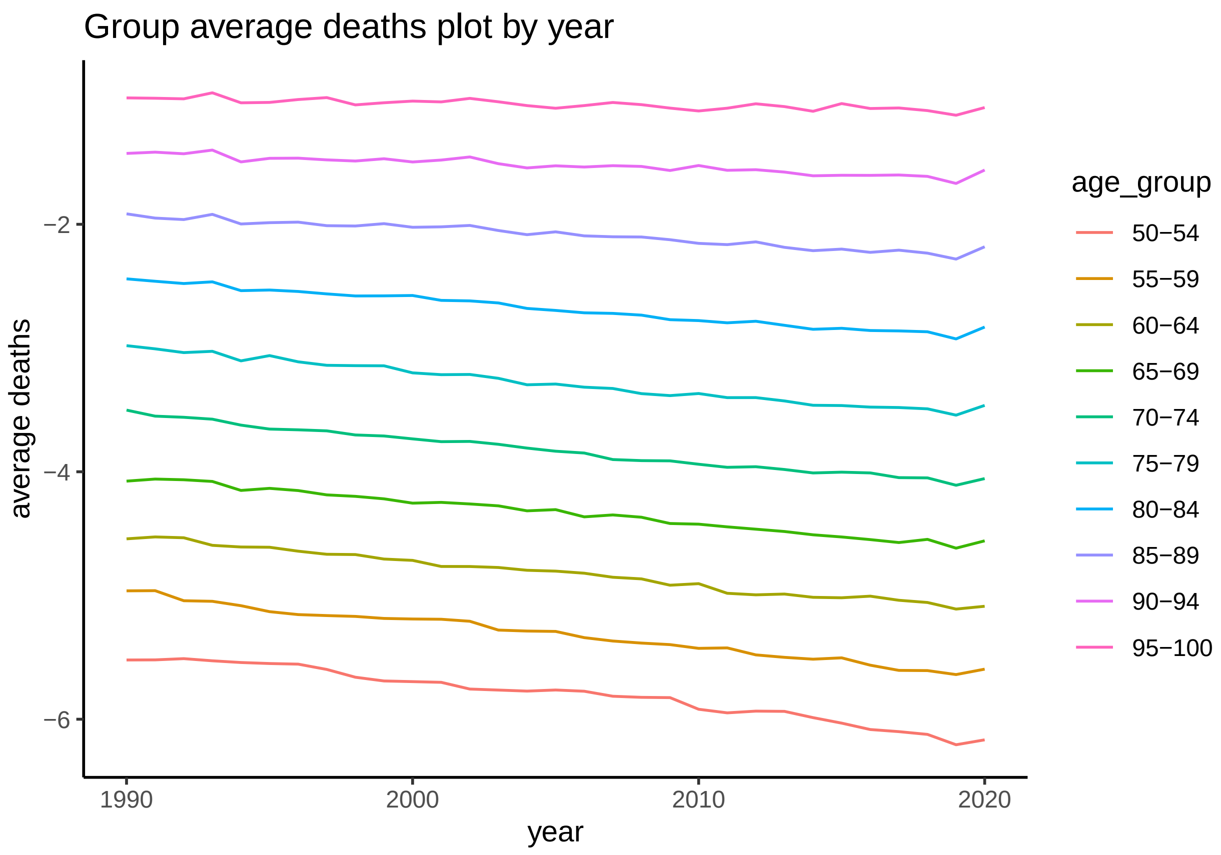
As we can see, if we separate things on the number of deaths, all the age groups are pretty layered. Although it is quite surprising that age group 80-89 has the highest deaths number, this could be quite biased because we are not comparing them at a similar scale; therefore, we can convert them into mortality rate by dividing them into group log mortality rate:

```
df |>
  group_by(age_group,year) |>
  summarise(group_avg_mortality = mean(deaths)/mean(pop)) |>
  ggplot(aes(x = year, y = log(group_avg_mortality), color = age_group)) +
  geom_line() +
  labs(title = 'Group average deaths plot by year', y = "average deaths")+
  theme_classic()
```



as we can see, the log mortality rate is very well layered with highest to lowest based the magnitude of age. this could be very helpful for us to later fit the model, to be safe, let's also look at the case where we divide them in to 10 groups instead of 5 groups, see if the result could be persistent.

```
data |>
  mutate(
    age_group = case_when(
      age >= 50 & age <= 54 ~ "50-54",
      age >= 55 & age <= 59 ~ "55-59",
      age >= 60 & age <= 64 ~ "60-64",
      age >= 65 & age <= 69 ~ "65-69",
      age >= 70 & age <= 74 ~ "70-74",
      age >= 75 & age <= 79 ~ "75-79",
      age >= 80 & age <= 84 ~ "80-84",
      age >= 85 & age <= 89 ~ "85-89",
      age >= 90 & age <= 94 ~ "90-94",
      age >= 95 & age <= 100 ~ "95-100",
    ),
    age_group = factor(
      age_group,
      level = c("50-54", "55-59", "60-64", "65-69",
                "70-74", "75-79", "80-84", "85-89",
                "90-94", "95-100")
    )
  ) |>
  group_by(age_group,year) |>
  summarise(group_avg_mortality = mean(deaths)/mean(pop)) |>
  ggplot(aes(x = year, y = log(group_avg_mortality), color = age_group)) +
  geom_line() +
  labs(title = 'Group average deaths plot by year', y = "average deaths")+
  theme_classic()
```



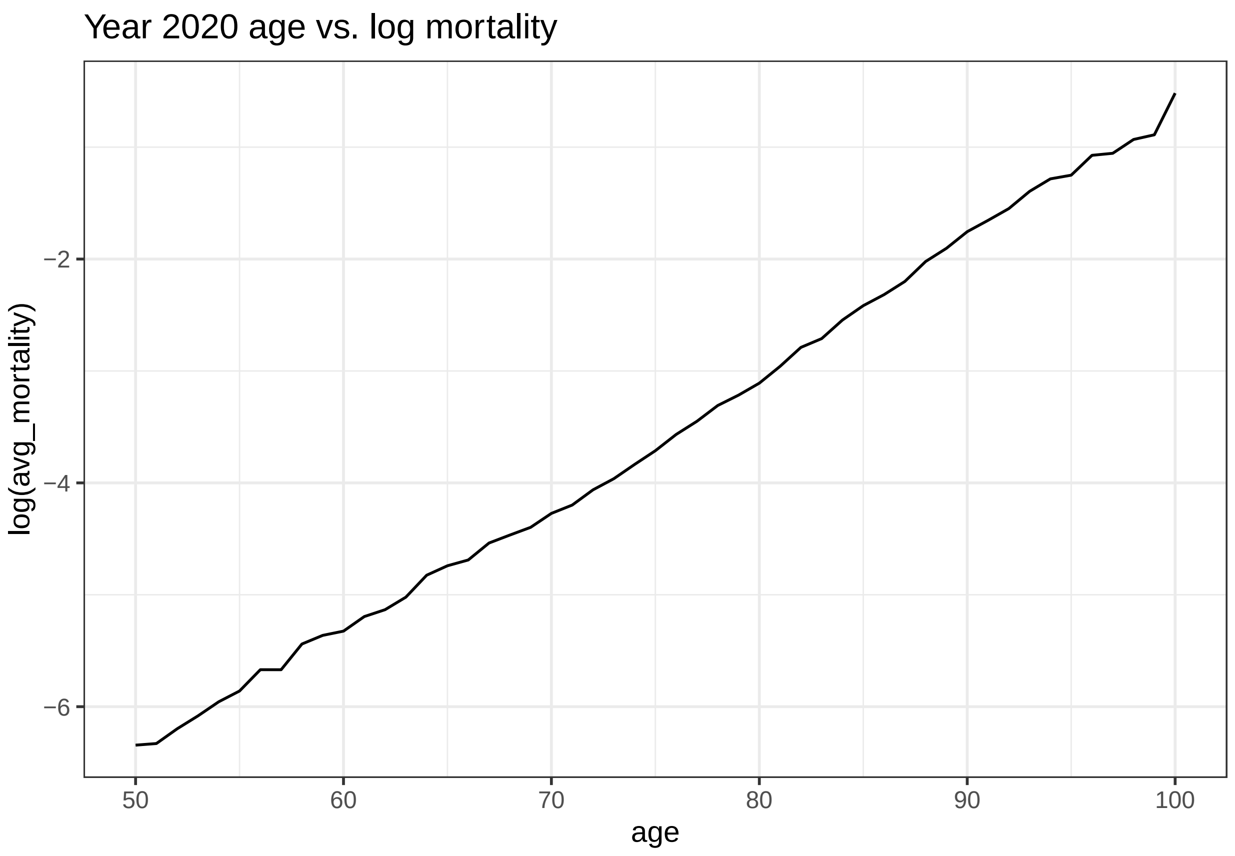
as we can see, they are still very wellly layered out, therefore suggesting that a layered mortality rate model would be useful for weakly informative prior in model fitting, and in terms of trend with respect to time, the mortality rate is in general decreasing, with smaller the age, the more significant the decrement is.

b) Carry out prior predictive checks for α and β, based on populations by age in Sweden in 2020. Summarize what you find and what you decide to be weakly informative priors for these parameters.

let's look at the mortality rate for 2020:

```
df |>
  group_by(age,year) |>
  summarise(avg_mortality = mean(deaths)/mean(pop)) |>
  filter(year == 2020) |>
  ggplot(aes(x = age, y = log(avg_mortality))) +
  geom_line() +
```

```r
  labs(title = 'Year 2020 age vs. log mortality') +
  theme_bw()
```



as we can see the log mortality is nearly a linear line, that gives us good reasoning to estimate alpha and beta:

we have $\mu_x = \alpha e^{\beta x}$

hence

$$log(\mu_x) = log(\alpha) + \beta x$$

hence fitting in a linear regression model we can get our estimate on alpha and beta:

```r
ef <- df |>
  group_by(age,year) |>
  summarise(avg_mortality = mean(deaths)/mean(pop)) |>
  filter(year == 2020)

summary(lm(log(avg_mortality)~age, data = ef))
```

```
##
## Call:
## lm(formula = log(avg_mortality) ~ age, data = ef)
##
## Residuals:
```

```
##       Min        1Q    Median        3Q       Max
## -0.143437 -0.064610  0.003273  0.063703  0.176211
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.228e+01  6.045e-02  -203.2   <2e-16 ***
## age          1.159e-01  7.909e-04   146.5   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08314 on 49 degrees of freedom
## Multiple R-squared:  0.9977, Adjusted R-squared:  0.9977
## F-statistic: 2.148e+04 on 1 and 49 DF,  p-value: < 2.2e-16
```

this gives us an estimate of $\beta = 0.1159, \alpha = e^{-12.28} \approx 4.643696e-06$

using this as prior information, and the scale is clearer to us, we can have weakly informative priors of :

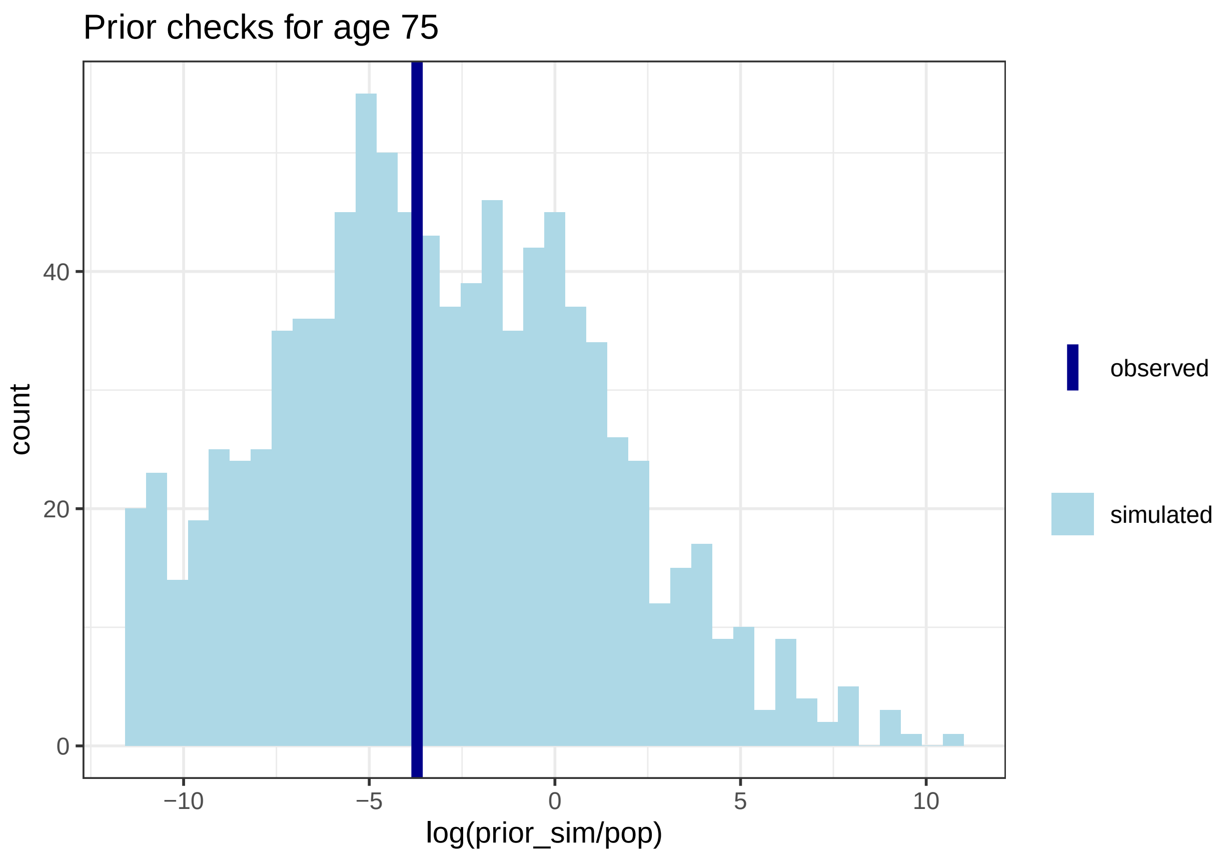
$$\beta \sim N(0.1159, 0.05) log(\alpha) \sim N(-12.28, 3)$$

following this, we can have a quick check upon the death number distribution of age 75 (median of the ages):

```r
set.seed(1024)
beta <- rnorm(1000, 0.1159, 0.05)
alpha <- exp(rnorm(1000, -12.28,3))
A <- 75
P_A <- filter(df, year == 2020, age == A)$pop
lam <- alpha*exp(beta*A)*P_A
prior_sim <- rpois(length(lam), lam) # generate the numbers
# and we can plot the histogram

tibble(death = prior_sim, pop = P_A) |>
  ggplot(aes(log(prior_sim/pop))) +
  geom_histogram(bins = 40, aes(fill = 'simulated')) +
  geom_vline(aes(xintercept = log(filter(ef, age == A)$avg_mortality), color = "observed"), lw
  scale_color_manual(name = "",
                     values = c("observed" = "darkblue"))+
  scale_fill_manual(name = "",
                     values = c("simulated" = "lightblue")) +
  labs(title = "Prior checks for age 75")+
  theme_bw()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
```

```
## Warning: Removed 49 rows containing non-finite values (`stat_bin()`).
```



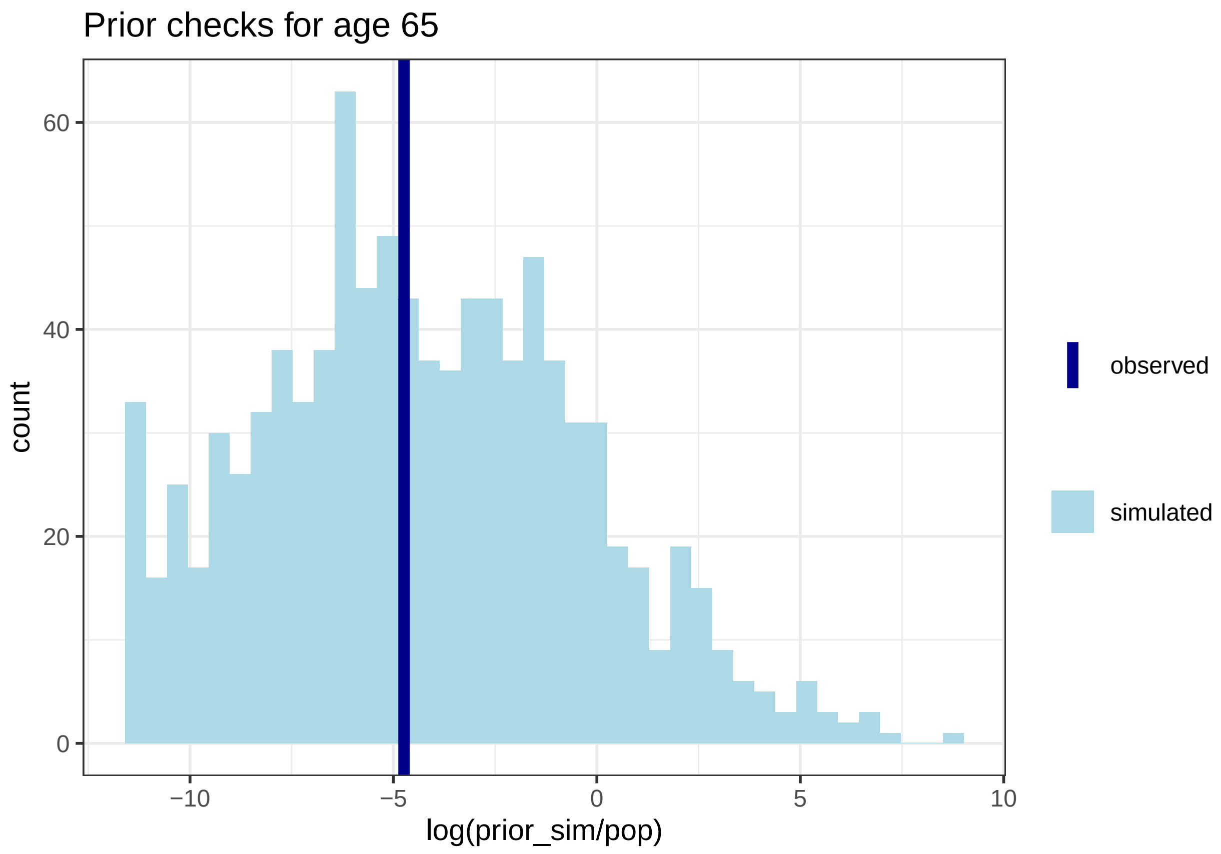
as we can see, it aligns with our distribution quite well

we can also take a look at age 65, and age 85:

for Age 65:

```r
set.seed(1024)
beta <- rnorm(1000, 0.1159, 0.05)
alpha <- exp(rnorm(1000, -12.28,3))
A <- 65
P_A <- filter(df, year == 2020, age == A)$pop
lam <- alpha*exp(beta*A)*P_A
prior_sim <- rpois(length(lam), lam) # generate the numbers
# and we can plot the histogram

tibble(death = prior_sim, pop = P_A) |>
  ggplot(aes(log(prior_sim/pop))) +
  geom_histogram(bins = 40, aes(fill = 'simulated')) +
  geom_vline(aes(xintercept = log(filter(ef, age == A)$avg_mortality), color = "observed"), lwd
  scale_color_manual(name = "",
                     values = c("observed" = "darkblue"))+
  scale_fill_manual(name = "",
                    values = c("simulated" = "lightblue")) +
  labs(title = "Prior checks for age 65")+
  theme_bw()
```

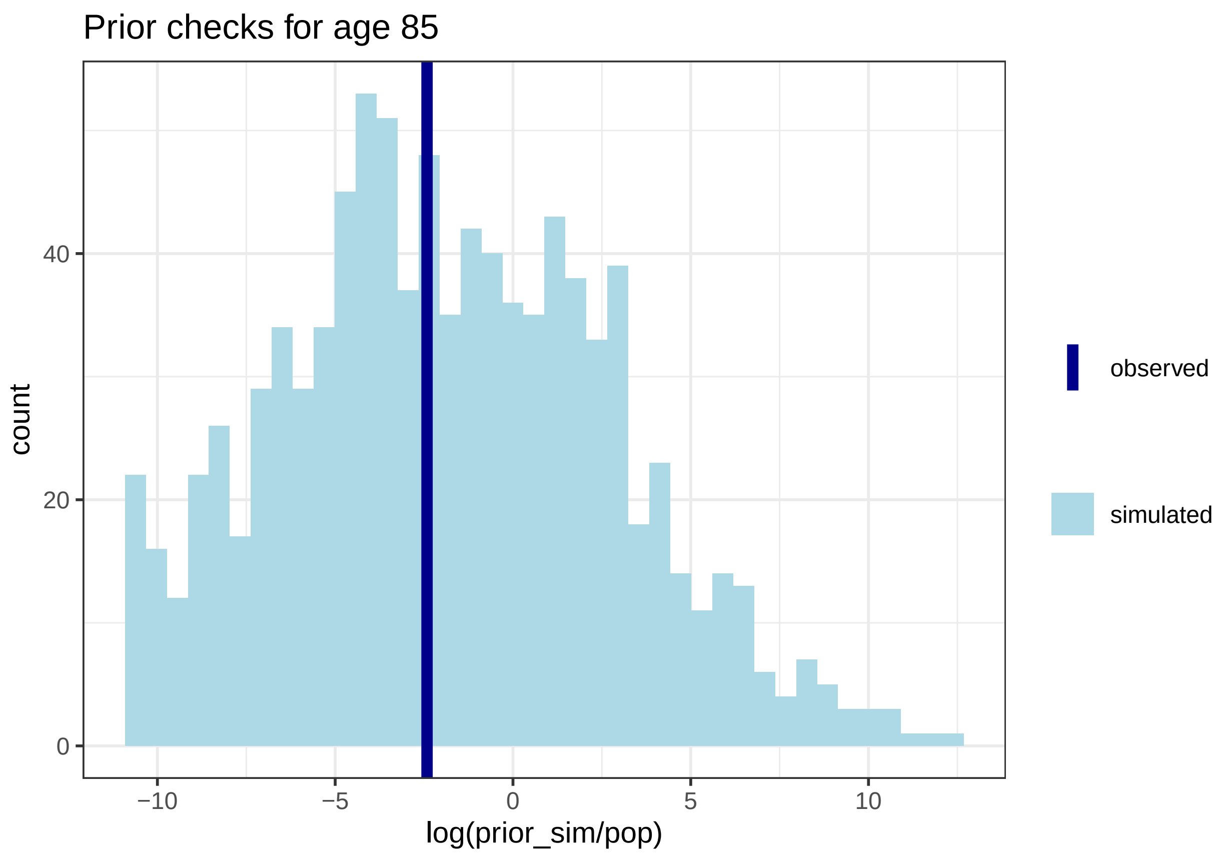
```
## Warning: Removed 53 rows containing non-finite values (`stat_bin()`).
```



and age 85:

```r
set.seed(1024)
beta <- rnorm(1000, 0.1159, 0.05)
alpha <- exp(rnorm(1000, -12.28,3))
A <- 85
P_A <- filter(df, year == 2020, age == A)$pop
lam <- alpha*exp(beta*A)*P_A
prior_sim <- rpois(length(lam), lam) # generate the numbers
# and we can plot the histogram

tibble(death = prior_sim, pop = P_A) |>
  ggplot(aes(log(prior_sim/pop))) +
  geom_histogram(bins = 40, aes(fill = 'simulated')) +
  geom_vline(aes(xintercept = log(filter(ef, age == A)$avg_mortality), color = "observed"), lw
  scale_color_manual(name = "",
                     values = c("observed" = "darkblue"))+
  scale_fill_manual(name = "",
                     values = c("simulated" = "lightblue")) +
  labs(title = "Prior checks for age 85")+
  theme_bw()
```

```
## Warning: Removed 57 rows containing non-finite values (`stat_bin()`).
```



as we can see, they all fit the model well;

c) Fit a model in Stan to estimate α and β for the year 2020. Note that it may be easier to specify the likelihood on the log scale (you can do this in Stan using the `poisson_log` function). Priors should be informed by your prior predictive checks and any other information available. Ensure that the model has converged and other diagnostics are good. Interpret your estimates for α and β.

Let's use the code in Q2_model1.stan

```
library(rstan)
```

```
## Loading required package: StanHeaders

## rstan (Version 2.21.8, 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:tidyr':
##
##     extract
```

```r
stan_data <-list(N = nrow(ef),
                 y = filter(df, year == 2020)$death,
                 x = ef$age,
                 log_p = log(filter(df, year == 2020)$pop))

mod_q2_1 <- stan(data = stan_data,
                 file = 'Q2_model1.stan',
                 iter = 2000,
                 seed = 1025)
```

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Libra
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'Q2_model1' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
```

```
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.163004 seconds (Warm-up)
## Chain 1:                0.200438 seconds (Sampling)
## Chain 1:                0.363442 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'Q2_model1' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 6e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.063953 seconds (Warm-up)
## Chain 2:                0.191443 seconds (Sampling)
## Chain 2:                0.255396 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'Q2_model1' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 8e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.05673 seconds (Warm-up)
## Chain 3:                0.201929 seconds (Sampling)
## Chain 3:                0.258659 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'Q2_model1' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 5e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.175686 seconds (Warm-up)
## Chain 4:                0.213437 seconds (Sampling)
## Chain 4:                0.389123 seconds (Total)
## Chain 4:
```

```
summary(mod_q2_1)$summary[1:2,]
```

```
##                  mean      se_mean           sd        2.5%         25%
## log_alpha -12.5955744 1.159662e-03 0.0257359540 -12.6463083 -12.6140375
## beta        0.1195835 1.399674e-05 0.0003128687   0.1189615   0.1193754
##                   50%        75%       97.5%    n_eff     Rhat
## log_alpha -12.5954780 -12.578731 -12.5434703 492.5126 1.007317
## beta        0.1195845   0.119802   0.1202004 499.6553 1.007470
```

as we can see, the model converges really nicely. and log_alpha means at an age of 0 (not possible but to interpret the model), the effect on the population of the mean of deaths number is exp(-12.5955744)

<- which is the value of exp(log(alpha)) = alpha

the beta is the effect on one year increment in age, can result in a exp(0.1195835) - 1 = 12.7% increment in the effect upon population.

d) Carry out some posterior predictive checks to assess model performance.

we can first look at the median of deaths rate between observed data and the simulated data:

```
library(bayesplot)
```

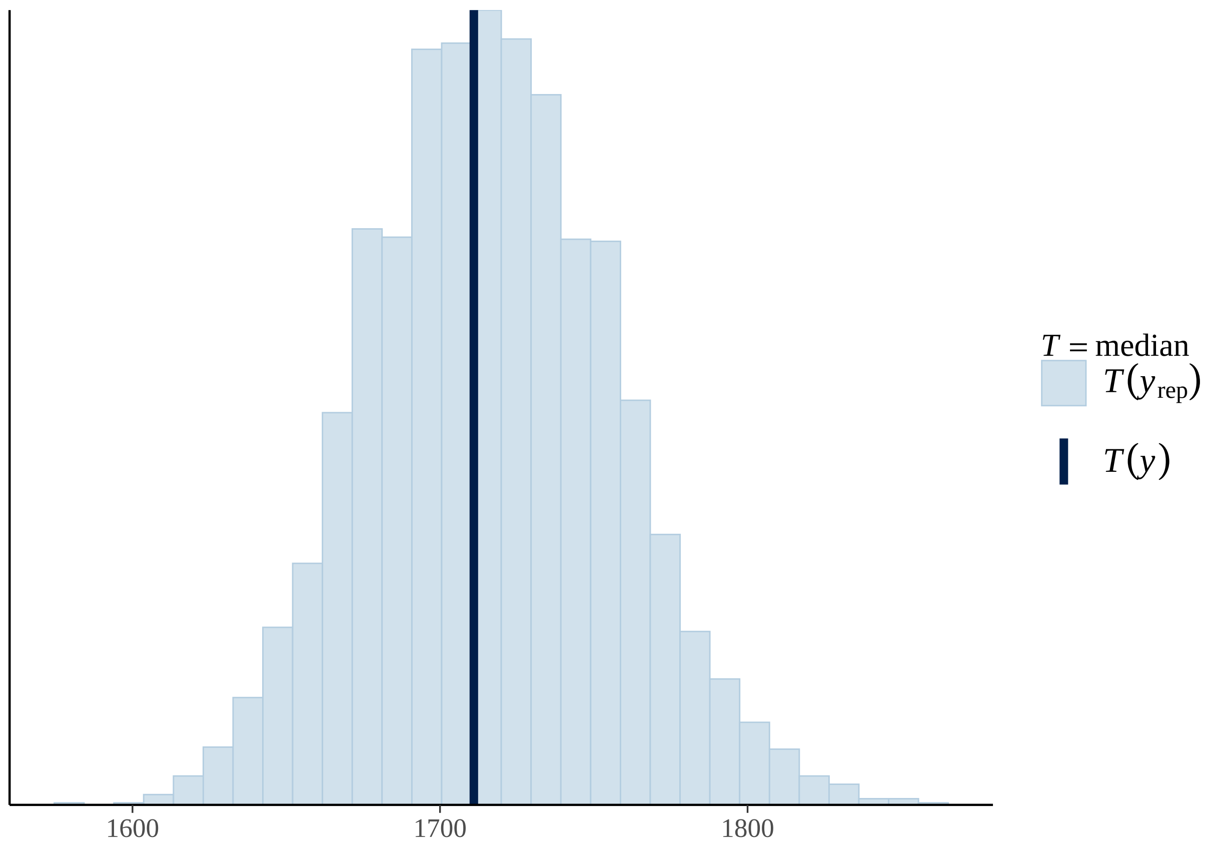
```
## This is bayesplot version 1.10.0

## - 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
```

```
library(loo)
```

```
## This is loo version 2.5.1

## - 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 '

##
## Attaching package: 'loo'

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

```
library(tidybayes)
y_rep <- extract(mod_q2_1)[["y_hat"]]
d = filter(df, year == 2020)$deaths
ppc_stat(d, y_rep, stat = 'median')
```

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



as we can see, the median aligns with the simulated data very well.

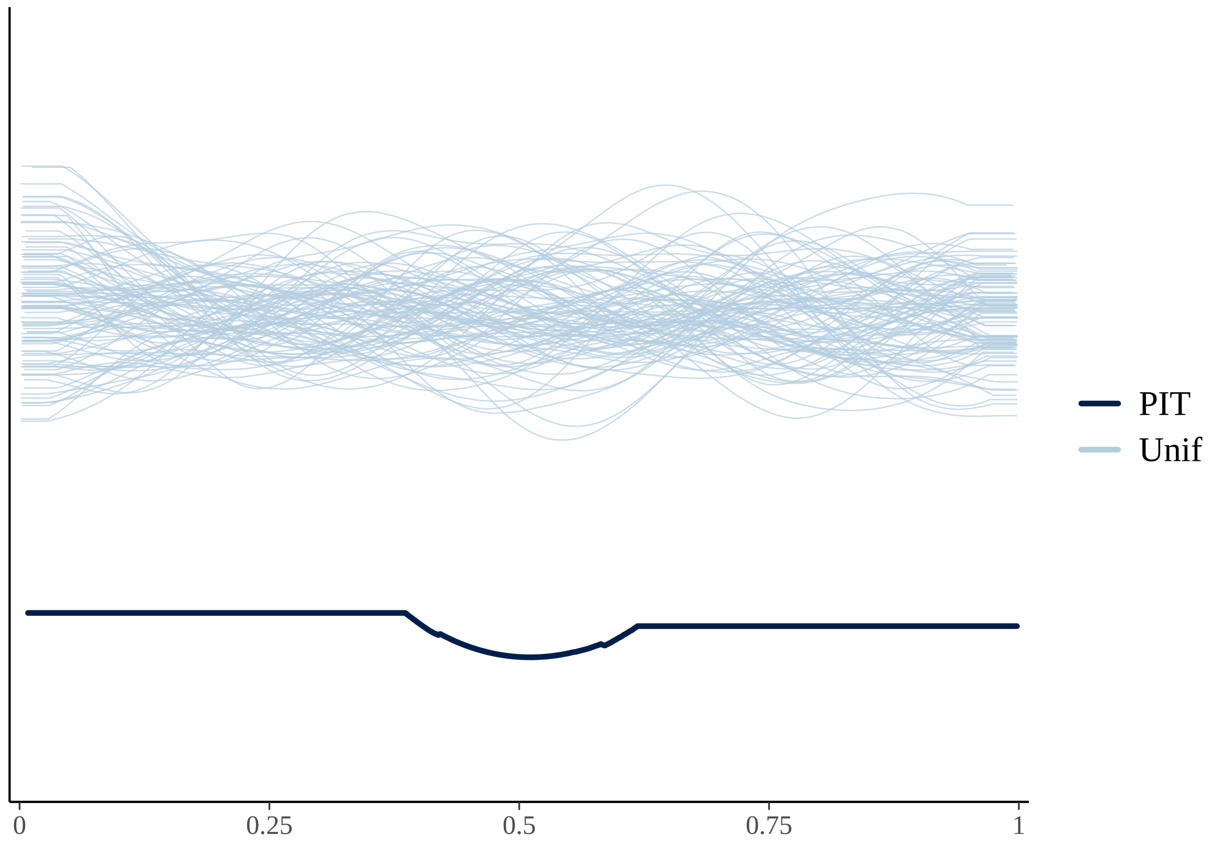
furthermore, We can also compare the LOO-PIT of the model to standard uniforms:

```
loglik1 <- extract(mod_q2_1)[["log_prob"]]
loo1 <- loo(loglik1, 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.
```

```
ppc_loo_pit_overlay(yrep = y_rep, y = filter(df, year == 2020)$deaths, lw = weights(loo1$psis_o
```

```
## NOTE: The kernel density estimate assumes continuous observations and is not optimal for dis
```



but such plot is not working very well in the case, looking into the documentation, the PSIS check will not work very well on discrete cases, so let's come up with something else:

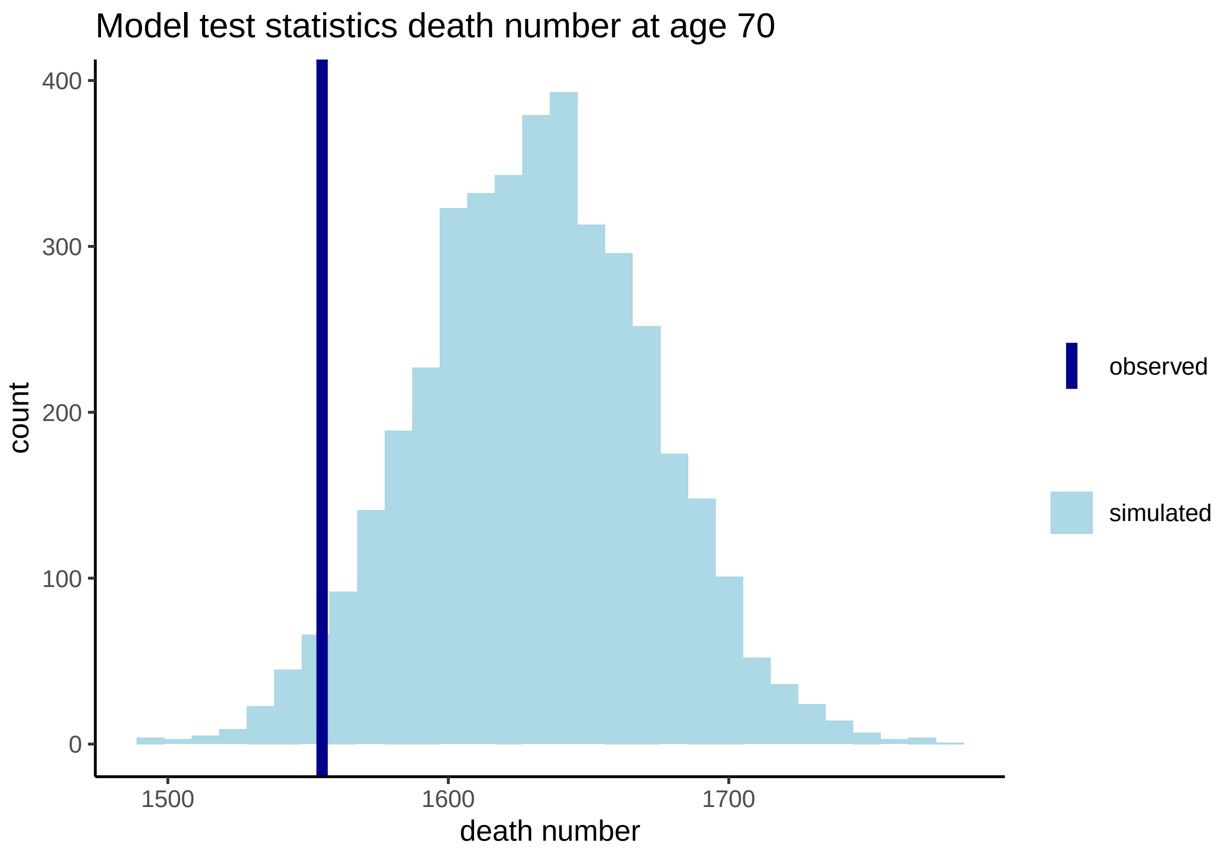
we will compare the death number at age 70:

```r
A = 70
stat_mod <- c()
stat_y <- sum(filter(df, year == 2020, age == A)$deaths)
for (i in 1:nrow(y_rep)){
  stat_mod[i] <- y_rep[i, A-50+1]
}
```

and plot them into one graph we can see:

```r
stat_mod |>
  as.tibble() |>
  ggplot(aes(value)) +
  geom_histogram(aes(fill = 'simulated'))+
  geom_vline(aes(xintercept = stat_y, color = "observed"), lwd = 2)+
  ggtitle('Model test statistics death number at age 70')+
  labs(x='death number') +
  scale_color_manual(name = "",
                     values = c("observed" = "darkblue"))+
  scale_fill_manual(name = "",
                     values = c("simulated" = "lightblue")) +
  # the manual change of color idea is adapted from professor's blog post
  theme_classic()
```

```
## Warning: `as.tibble()` was deprecated in tibble 2.0.0.
## i Please use `as_tibble()` instead.
## i The signature and semantics have changed, see `?as_tibble`.

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

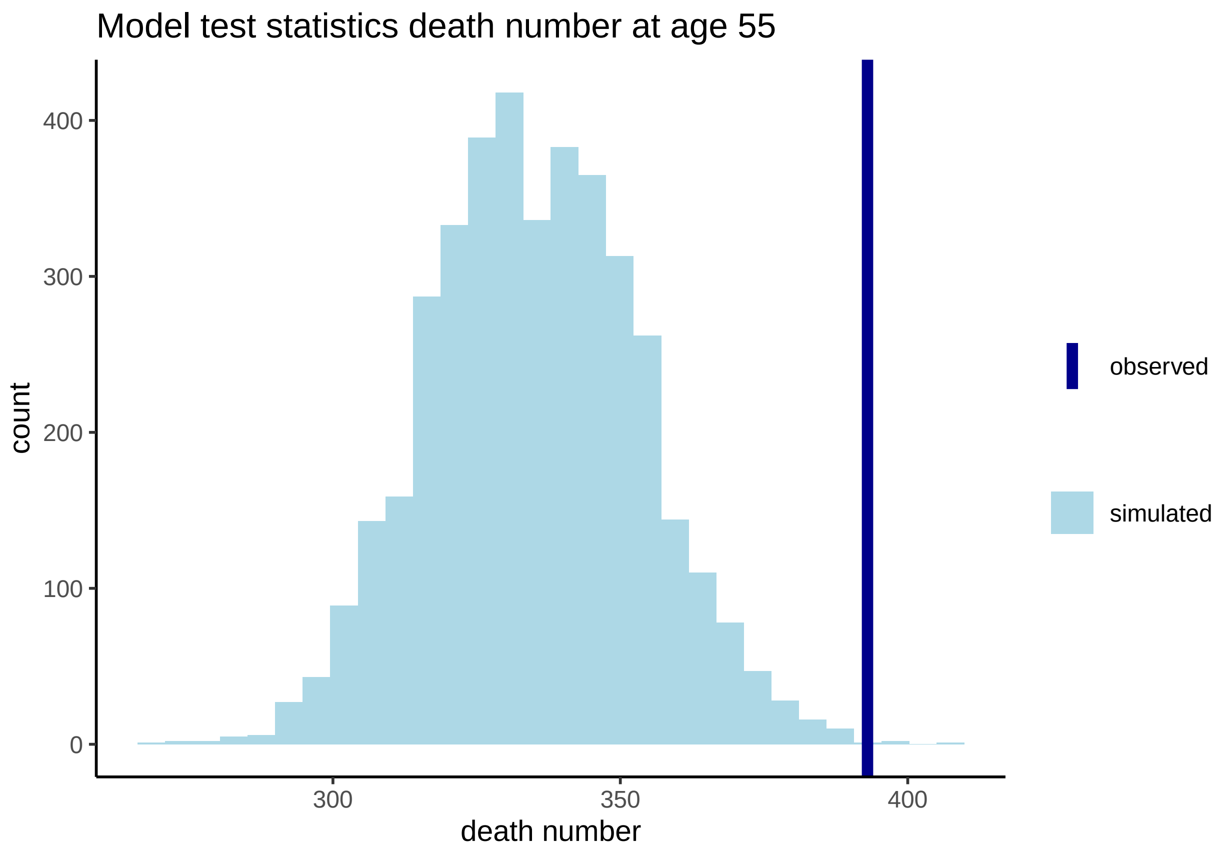


can also look at age 55, and age 80:

we will compare the death number at age 70:

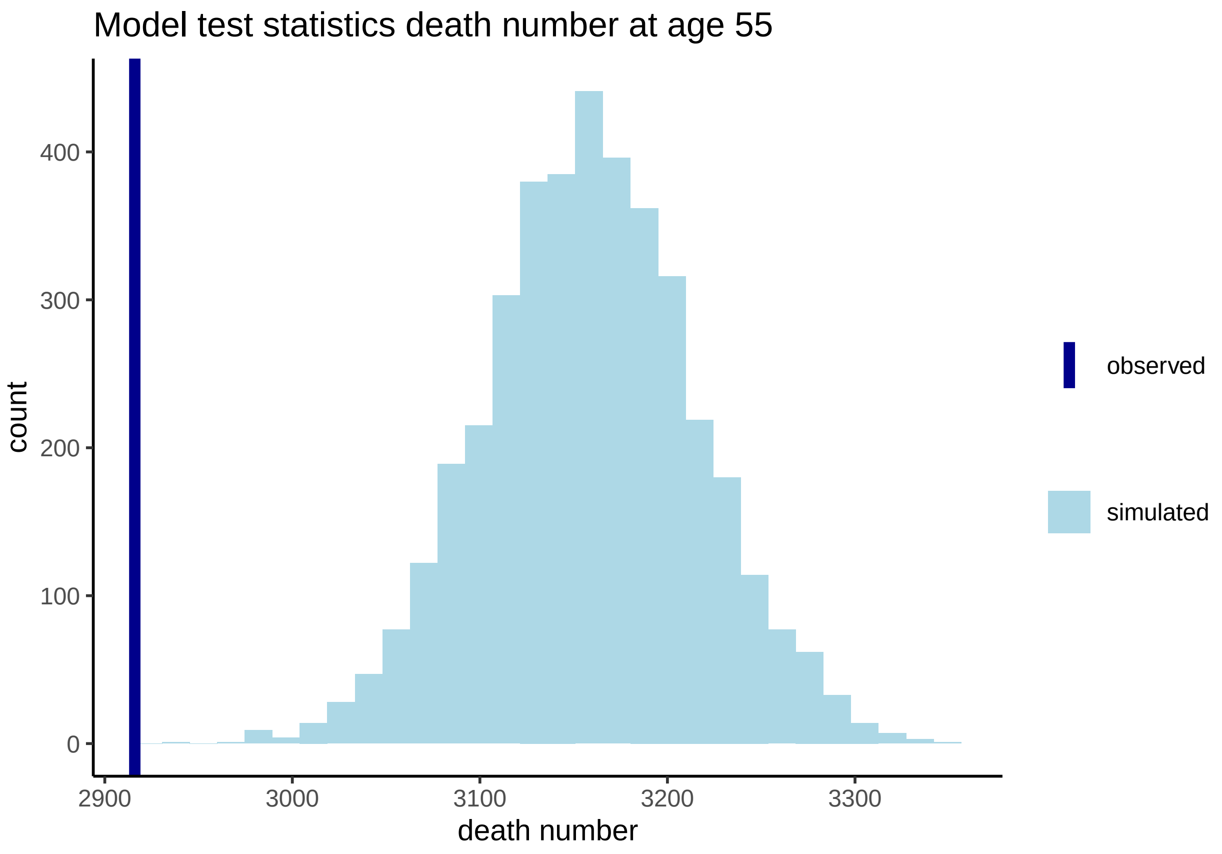
```
A = 55
stat_mod <- c()
stat_y <- sum(filter(df, year == 2020, age == A)$deaths)
for (i in 1:nrow(y_rep)){
  stat_mod[i] <- y_rep[i, A-50+1]
}

stat_mod |>
  as.tibble() |>
  ggplot(aes(value)) +
  geom_histogram(aes(fill = 'simulated'))+
  geom_vline(aes(xintercept = stat_y, color = "observed"), lwd = 2)+
  ggtitle('Model test statistics death number at age 55')+
  labs(x='death number') +
  scale_color_manual(name = "",
                     values = c("observed" = "darkblue"))+
  scale_fill_manual(name = "",
                    values = c("simulated" = "lightblue")) +
  # the manual change of color idea is adapted from professor's blog post
  theme_classic()
```

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



```
A = 80
stat_mod <- c()
stat_y <- sum(filter(df, year == 2020, age == A)$deaths)
for (i in 1:nrow(y_rep)){
  stat_mod[i] <- y_rep[i, A-50+1]
}

stat_mod |>
  as.tibble() |>
  ggplot(aes(value)) +
  geom_histogram(aes(fill = 'simulated'))+
  geom_vline(aes(xintercept = stat_y, color = "observed"), lwd = 2)+
  ggtitle('Model test statistics death number at age 55')+
  labs(x='death number') +
  scale_color_manual(name = "",
                      values = c("observed" = "darkblue"))+
  scale_fill_manual(name = "",
                      values = c("simulated" = "lightblue")) +
  # the manual change of color idea is adapted from professor's blog post
  theme_classic()
```

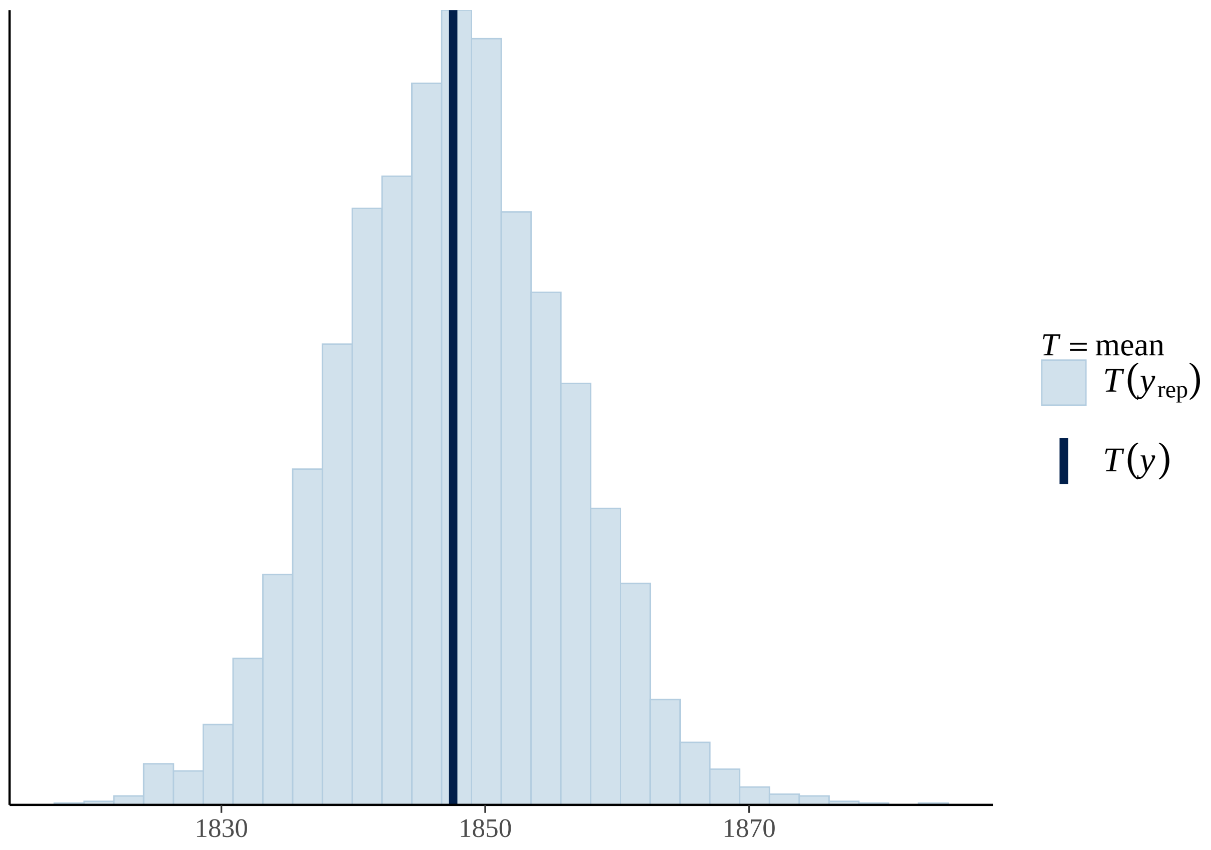
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



as we can see, it is also not showing a very good result in the sense that this statistisc is very time-dependent, but our simulation is not too off. Let's do one more PPC of mean values:

```
ppc_stat(filter(df, year == 2020)$deaths, y_rep, stat = 'mean')
```

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



as shown in the graph, using mean values to PPC gave us good results.

e) Now extend your model to estimate α and β in every year over the interval 1990-2020. Plot the resulting point estimates and 95% credible intervals for your estimates of α and β over time. Comment briefly on what you observe.

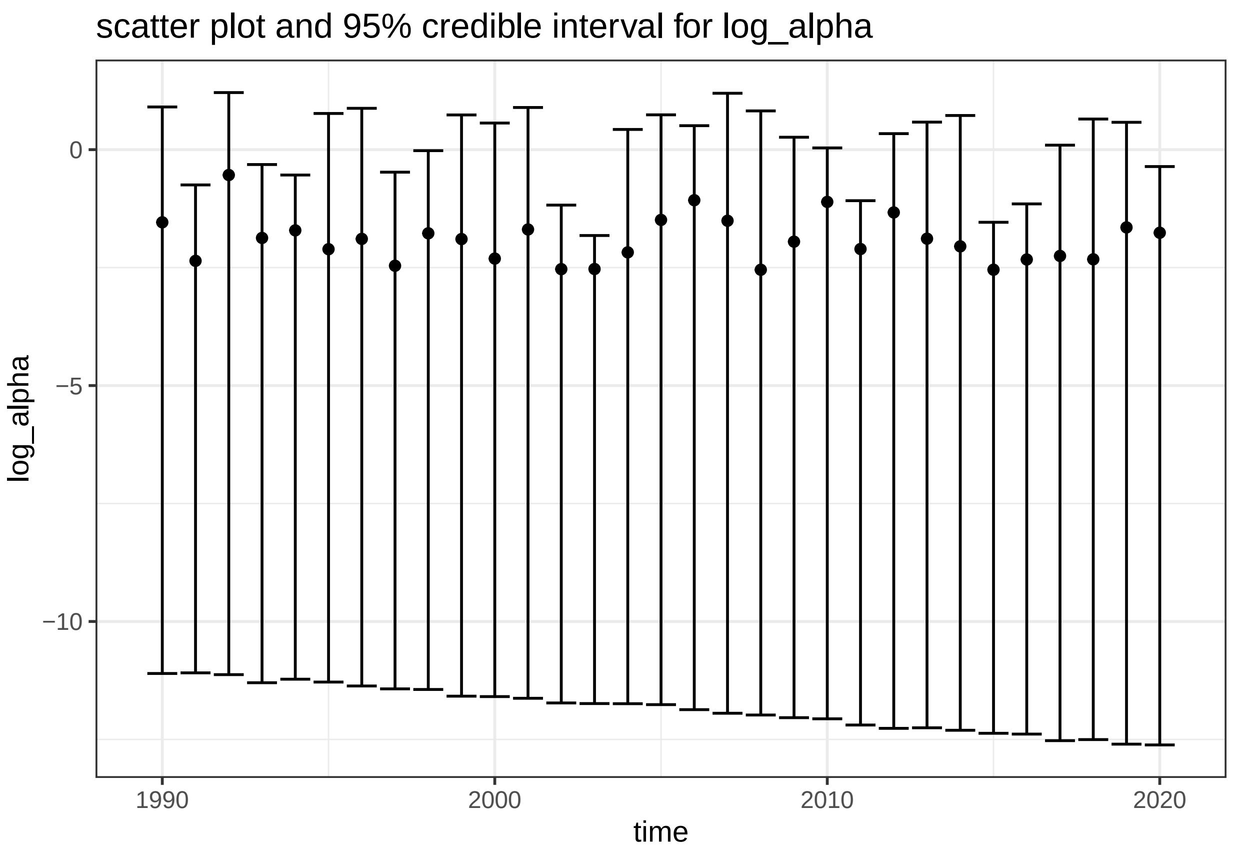
Let's put the code at Q2_model2.stan, and run the results below:

```
d <- df |>
  select(age,year,deaths) |>
  pivot_wider(names_from = 'year', values_from = 'deaths') |>
  as.matrix()

p <- df |>
  select(age,year,pop) |>
  pivot_wider(names_from = 'year', values_from = 'pop') |>
  as.matrix()

stan_data <-list(N = nrow(ef),
                 Time = nrow(unique(select(df,year))),
                 y = d[,-1],
                 x = ef$age,
                 log_p = log(p[,-1]))

mod_q2_2 <- stan(data = stan_data,
                 file = 'Q2_model2.stan',
                 iter = 2000,
                 seed = 1025)
```

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Libra
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/I
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/I
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/I
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'Q2_model2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000163 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.63 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 5.65204 seconds (Warm-up)
```

```
## Chain 1:                3.66948 seconds (Sampling)
## Chain 1:                9.32152 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'Q2_model2' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000105 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 1.05 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 5.86299 seconds (Warm-up)
## Chain 2:                2.63751 seconds (Sampling)
## Chain 2:                8.5005 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'Q2_model2' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000106 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 1.06 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 4.23871 seconds (Warm-up)
## Chain 3:                4.34183 seconds (Sampling)
## Chain 3:                8.58054 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'Q2_model2' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000107 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 1.07 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 22.9522 seconds (Warm-up)
## Chain 4:                61.2468 seconds (Sampling)
## Chain 4:                84.199 seconds (Total)
## Chain 4:

## Warning: There were 616 transitions after warmup that exceeded the maximum treedepth. Increa
## https://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded

## Warning: There were 4 chains where the estimated Bayesian Fraction of Missing Information wa
## https://mc-stan.org/misc/warnings.html#bfmi-low

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

## Warning: The largest R-hat is 4.15, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat

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

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

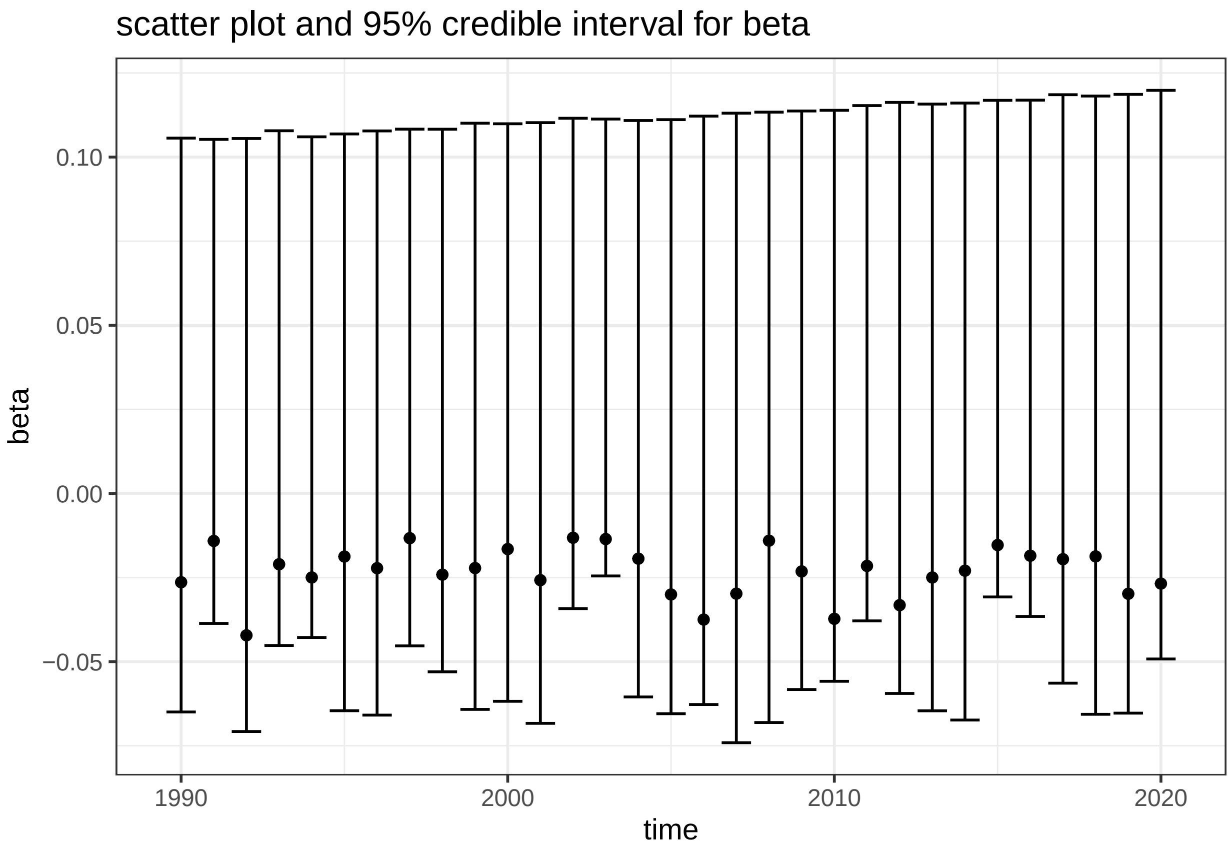
```
# let's draw for alpha 1 first:
library(dplyr)
mod_q2_2 |>
  gather_draws(log_alpha[i]) |>
  median_qi() |>
  rename(median_mod = .value,
         lower_mod = .lower,
         upper_mod = .upper) |>
  dplyr::select(i, median_mod:upper_mod) |>
  ggplot(aes(1990:2020, median_mod)) +
  geom_point() +
  geom_errorbar(aes(ymin = lower_mod, ymax = upper_mod)) +
  labs(title = 'scatter plot and 95% credible interval for log_alpha', x = 'time', y = 'log_al
  theme_bw()
```



as we can see from the graph, the estimate for alpha ranging from year 1990 to 2020 does not change by much, but the variability is very significant across many years.

now for beta:

```
mod_q2_2 |>
  gather_draws(beta[i]) |>
  median_qi() |>
  rename(median_mod = .value,
         lower_mod = .lower,
        upper_mod = .upper) |>
  dplyr::select(i, median_mod:upper_mod) |>
  ggplot(aes(1990:2020, median_mod)) +
  geom_point() +
  geom_errorbar(aes(ymin = lower_mod, ymax = upper_mod)) +
  labs(title = 'scatter plot and 95% credible interval for beta', x = 'time', y = 'beta') +
  theme_bw()
```



as we can see from the graph, the beta also flucatuate around a mean value around -0.25 from 1990 to 2020, but the variablity is quite large across samples.

f) Life expectancy at age x is defined as

$$\int_x^{\omega} e^{-\mu_a} da$$

where ω is the oldest age group (you may assume this is age 100). Life expectancy is the expected number of years of life left at age x. The integral can be approximated by summing over discrete age groups. Based on your estimates in the previous question, estimate life expectancy at age 40 (note starting age!) for every year from 1990-2020. Plot your resulting point estimates and 95% credible intervals over time and comment briefly.

well this question is to sum:

$$\int_x^{\omega} e^{-\mu_a} da = \sum_{x=40}^{100} e^{-\alpha_t exp(\beta_t x)}$$

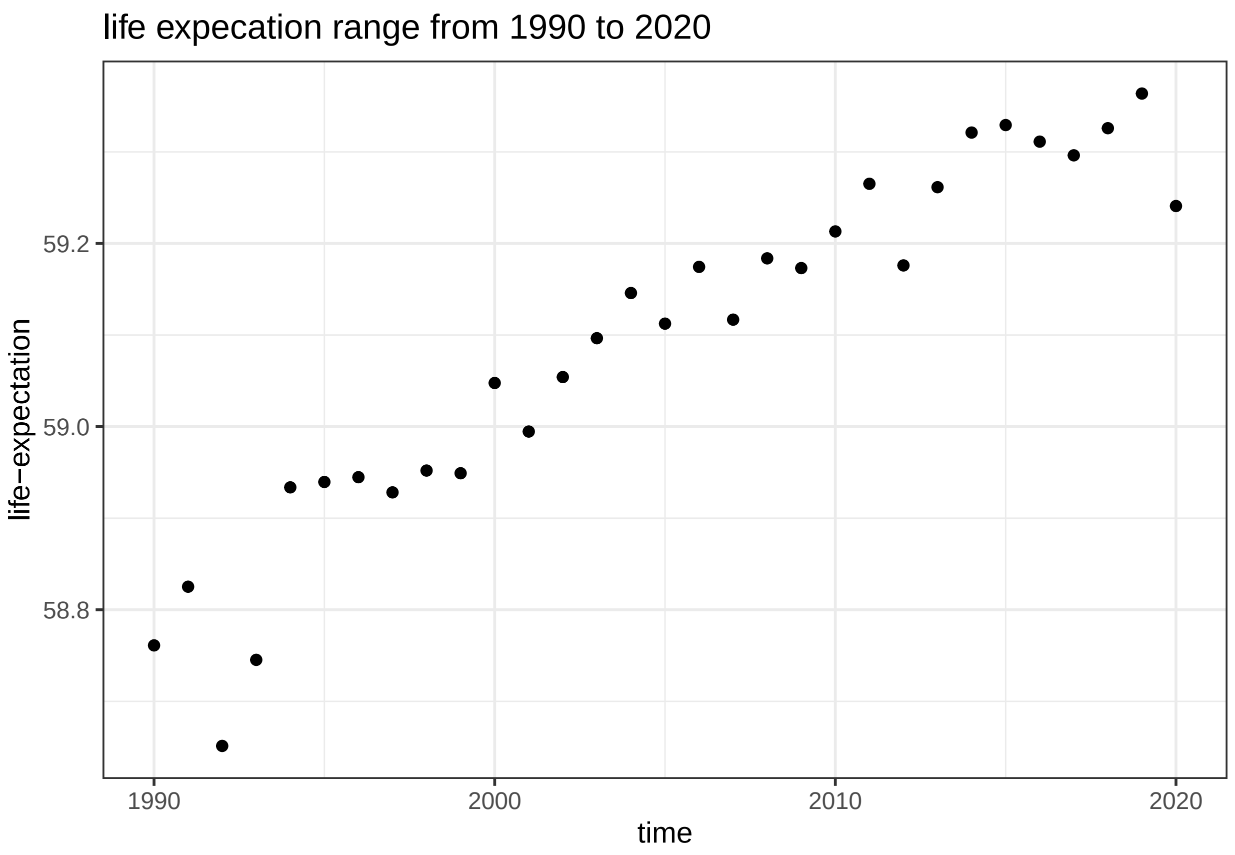
let's define a helper function for such:

```
life_expec <- function(a, b){
  ex = 0
  for (x in 40:100){
    ex = ex + exp(-a*exp(b*x))
  }
  return(ex)
}
```

therefore we can calculate the life expectation using the data we acquired:

```
alpha <- mod_q2_2 |>
  gather_draws(log_alpha[i]) |>
  median_qi() |>
  rename(median_alpha = .value,
         lower_alpha = .lower,
         upper_alpha = .upper) |>
  dplyr::select(i, median_alpha:upper_alpha)

beta <- mod_q2_2 |>
  gather_draws(beta[i]) |>
  median_qi() |>
  rename(median_beta = .value,
         lower_beta = .lower,
         upper_beta = .upper) |>
  dplyr::select(i, median_beta:upper_beta)
```

```
lower_ex <- c()
median_ex <- c()
upper_ex <- c()

for (i in 1:31){
  lower_ex[i] = life_expec(exp(alpha$lower_alpha[i]), beta$lower_beta[i])
  median_ex[i] = life_expec(exp(alpha$median_alpha[i]), beta$median_beta[i])
  upper_ex[i] = life_expec(exp(alpha$upper_alpha[i]), beta$upper_beta[i])
}

tibble(lower_ex, median_ex, upper_ex) |>
  ggplot(aes(1990:2020, median_ex)) +
  geom_point() +
  #geom_errorbar(aes(ymin = lower_ex, ymax = upper_ex)) +
  labs(title = 'life expecation range from 1990 to 2020', x = 'time', y = 'life-expectation')
  theme_bw()
```



as we can see, the life expecatation slightly increases over the years. I didn't plot the error bar here but one can un-comment my geom_errorbar(aes(ymin = lower_ex, ymax = upper_ex)) to display it, it is just not showing any pattern with the error bar.

3 Wells

This question uses data looking at the decision of households in Bangladesh to switch drinking water wells in response to their well being marked as unsafe or not. A full description from the Gelman Hill text book (page 87):

"Many of the wells used for drinking water in Bangladesh and other South Asian countries are contaminated with natural arsenic, affecting an estimated 100 million people. Arsenic is a cumulative poison, and exposure increases the risk of cancer and other diseases, with risks estimated to be proportional to exposure. Any locality can include wells with a range of arsenic levels. The bad news is that even if your neighbor's well is safe, it does not mean that yours is safe. However, the corresponding good news is that, if your well has a high arsenic level, you can probably find a safe well nearby to get your water from—if you are willing to walk the distance and your neighbor is willing to share. [In an area of Bangladesh, a research team] measured all the wells and labeled them with their arsenic level as well as a characterization as"safe" (below 0.5 in units of hundreds of micrograms per liter, the Bangladesh standard for arsenic in drinking water) or "unsafe" (above 0.5). People with unsafe wells were encouraged to switch to nearby private or community wells or to new wells of their own construction. A few years later, the researchers returned to find out who had switched wells."

The outcome of interest is whether or not household i switched wells:

$$y_i = \begin{cases} 1 & \text{if household } i \text{ switched to a new well} \\ 0 & \text{if household } i \text{ continued using its own well.} \end{cases}$$

The data we are using for this question are here: http://www.stat.columbia.edu/~gelman/arm/examples/arsenic/wells.dat and you can load them in directly using `read_table`.

The variables of interest for this questions are

- `switch`, which is y_i above
- `arsenic`, the level of arsenic of the respondent's well
- `dist`, the distance (in metres) of the closest known safe well

let's first read in the data

```r
ds <- read.table("http://www.stat.columbia.edu/~gelman/arm/examples/arsenic/wells.dat")
```

a) Do an exploratory data analysis illustrating the relationship between well-switching, distance and arsenic. Think about different ways of effectively illustrating the relationships given the binary outcome. As usual, a good EDA includes well-thought-out descriptions and analysis of any graphs and tables provided, well-labelled axes, titles etc.

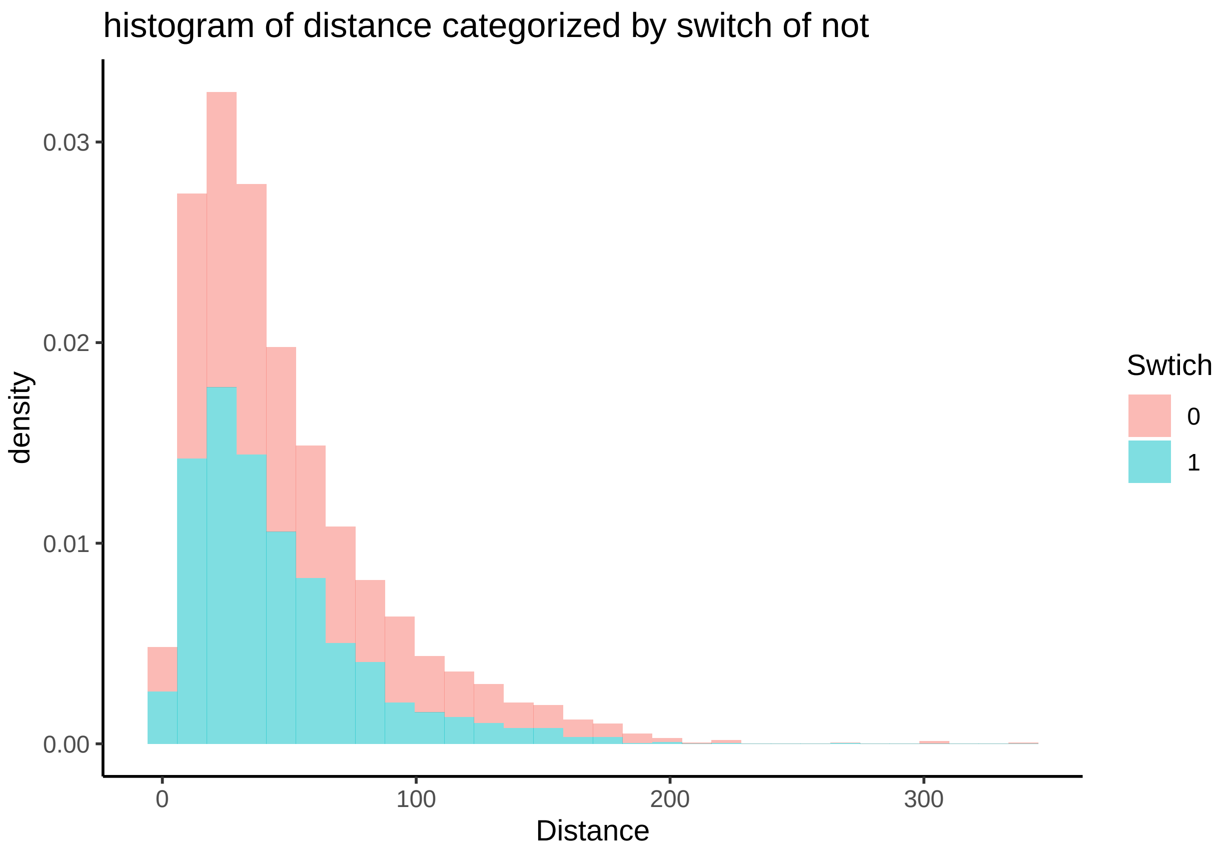
Assume $y_i \sim Bern(p_i)$, where p_i refers to the probability of switching. Consider two candidate models.

let's first plot the hitogram of the distance and arsenic under switch or not:

for distance:

```
ds |> ggplot() +
  geom_histogram(aes(x = dist, y = ..density.., fill = as.factor(switch)), alpha = 0.5, bins =
  labs(title = 'histogram of distance categorized by switch of not', x = 'Distance', fill = 'Sw
  theme_classic()
```

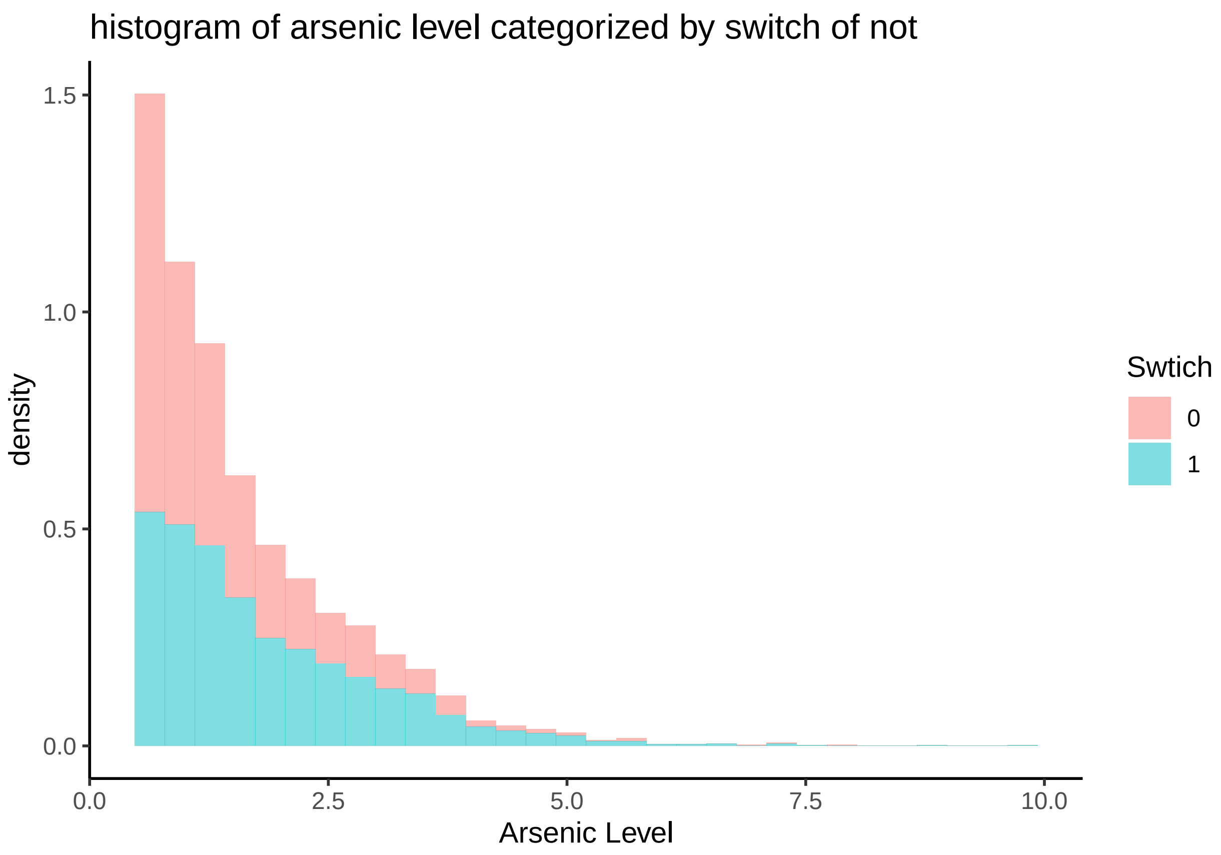
```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
```



```
as we can see, on average, household uses their own well more frequent to household switched t
```

Similarly, for arsenic level:

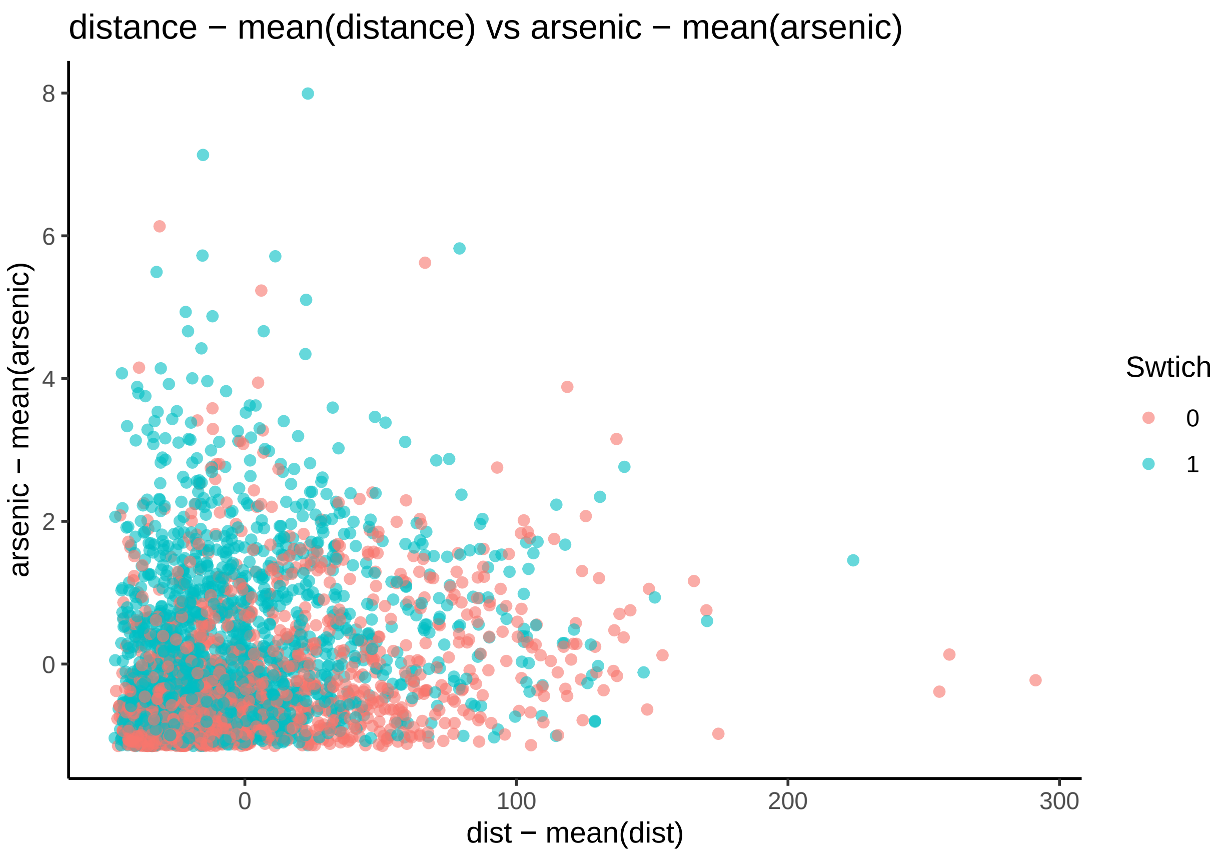
```
ds |> ggplot() +
  geom_histogram(aes(x = arsenic, y = ..density.., fill = as.factor(switch)), alpha = 0.5, bin
  labs(title = 'histogram of arsenic level categorized by switch of not', x = 'Arsenic Level',
  theme_classic()
```



And a similar conclusion can be reached, that is, a larger proportion of people tend to stay with their own well, but for people who switch to another well, their arsenic level tend to be high, At a very low level of arsenic level, household tends to stay with their own well.

Now we can create a scatter plot between distance and arsenic level, categorized by their switch or not.

```
ds |>
  ggplot(aes(dist - mean(dist), arsenic - mean(arsenic), color = as.factor(switch))) +
  geom_point(alpha = 0.6) +
  labs(title = 'distance - mean(distance) vs arsenic - mean(arsenic)',  color = 'Swtich')+
  theme_classic()
```



As we can see from the plot, on average for people who switch to a new well, they tend to have a higher arsenic level, and a lower distance.

- Model 1:

$$\text{logit}\left(p_i\right) = \beta_0 + \beta_1 \cdot \left(d_i - \bar{d}\right) + \beta_2 \cdot \left(a_i - \bar{a}\right) + \beta_3 \cdot \left(d_i - \bar{d}\right)\left(a_i - \bar{a}\right)$$

- Model 2:

$$\begin{aligned}\text{logit}\left(p_i\right) =& \beta_0 + \beta_1 \cdot \left(d_i - \bar{d}\right) + \beta_2 \cdot \left(\log\left(a_i\right) - \overline{\log(a)}\right) \\ &+ \beta_3 \cdot \left(d_i - \bar{d}\right)\left(\log\left(a_i\right) - \overline{\log(a)}\right)\end{aligned}$$

where d_i is distance and a_i is arsenic level.

b) Fit both of these models using Stan. Put $N(0,1)$ priors on all the βs. You should generate pointwise log likelihood estimates (to be used in later questions), and also samples from the posterior predictive distribution (unless you'd prefer to do it in R later on). For model 1, interpret each coefficient.

let's write the following stan code, one in Q3_model1.stan, one in Q3_model2.stan

```r
library(rstan)

dist <- ds$dist
arsenic <- ds$arsenic
```

```
p <- ds$switch
N <- nrow(ds)

stan_data <- list(N =N, dist = dist, arsenic = arsenic, p = p)
mod1 <- stan(data = stan_data,
             file = 'Q3_model1.stan',
             iter = 500,
             seed = 245)
```

```
## Trying to compile a simple C file
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Libra
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'Q3_model1' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000377 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 3.77 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 1: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 1: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 1: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 1: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 1: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 1: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 1: Iteration: 300 / 500 [ 60%]  (Sampling)
```

```
## Chain 1: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 1: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 1: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 1: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 6.48947 seconds (Warm-up)
## Chain 1:                5.23491 seconds (Sampling)
## Chain 1:                11.7244 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'Q3_model1' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000264 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 2.64 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 2: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 2: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 2: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 2: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 2: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 2: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 2: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 2: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 2: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 2: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 2: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 6.43648 seconds (Warm-up)
## Chain 2:                5.31248 seconds (Sampling)
## Chain 2:                11.749 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'Q3_model1' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000262 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 2.62 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 3: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 3: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 3: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 3: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 3: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 3: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 3: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 3: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 3: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 3: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 3: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 5.92389 seconds (Warm-up)
## Chain 3:                5.15265 seconds (Sampling)
## Chain 3:                11.0765 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'Q3_model1' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.00026 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 2.6 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 4: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 4: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 4: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 4: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 4: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 4: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 4: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 4: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 4: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 4: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 4: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 6.82052 seconds (Warm-up)
## Chain 4:                5.22524 seconds (Sampling)
## Chain 4:                12.0458 seconds (Total)
## Chain 4:
```

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

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

now we can extract the model summary for such from mod1:

```
summary(mod1)$summary[1:4,]
```

```
##                mean      se_mean          sd         2.5%           25%
## beta0  0.354474498 1.955991e-03 0.040219779  0.276442792  0.327840712
## beta1 -0.008799839 3.373059e-05 0.001049229 -0.010773325 -0.009500269
## beta2  0.473591785 2.312732e-03 0.042489203  0.392679192  0.446235222
## beta3 -0.001808328 3.123953e-05 0.001058264 -0.003852275 -0.002545436
##                50%          75%         97.5%     n_eff      Rhat
## beta0  0.355333858  0.381448824  0.4304817313  422.8105 1.0057904
## beta1 -0.008804910 -0.008127314 -0.0066944566  967.5939 1.0009892
## beta2  0.471957582  0.502167783  0.5641656514  337.5256 1.0091619
## beta3 -0.001804379 -0.001115086  0.0002108327 1147.5699 0.9966347
```

with the above information, we can say that:

beta0 : holding other dist-mean(dist), and arsenic- mean(arsenic) 0, on average people is exp(0.350292367) -1 = 42% more likely to switch to a new well beta1, beta3 : holding arsenic - mean(arsenic) constant, with one unit increase in dist-mean(dist), people are 1- exp(-0.008789761-0.001783209) = 1.05% less likely to switch to a new well. beta2, beta3 : holding dist - mean(dist) constant, with one unit increase in arsenic-mean(arsenic), people are exp(0.470130182 - -0.001783209) -1 = 60% more likely to switch to a new well.

and we save the log-likelihood and p_hat

```
log_lik_mod1 <- extract(mod1)[['log_lik']]
p_hat_mod1 <- extract(mod1)[['p_hat']]
```

now let us run the stan code for model 2, and look at the summary:

```
mod2 <- stan(data = stan_data,
             file = 'Q3_model2.stan',
             iter = 500,
             seed = 245)
```

```
## Trying to compile a simple C file
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Libra
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/S
## In file included from /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/R
## /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/RcppEigen/include/Eigen
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'Q3_model2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000435 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 4.35 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 1: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 1: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 1: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 1: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 1: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 1: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 1: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 1: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 1: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 1: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 1: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 21.9076 seconds (Warm-up)
## Chain 1:                20.558 seconds (Sampling)
## Chain 1:                42.4656 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'Q3_model2' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000323 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 3.23 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 2: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 2: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 2: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 2: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 2: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 2: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 2: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 2: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 2: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 2: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 2: Iteration: 500 / 500 [100%]  (Sampling)
```

```
## Chain 2:
## Chain 2:  Elapsed Time: 22.087 seconds (Warm-up)
## Chain 2:                20.6381 seconds (Sampling)
## Chain 2:                42.7251 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'Q3_model2' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000333 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 3.33 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 3: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 3: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 3: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 3: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 3: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 3: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 3: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 3: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 3: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 3: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 3: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 22.1462 seconds (Warm-up)
## Chain 3:                20.6613 seconds (Sampling)
## Chain 3:                42.8076 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'Q3_model2' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000328 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 3.28 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 500 [  0%]  (Warmup)
## Chain 4: Iteration:  50 / 500 [ 10%]  (Warmup)
## Chain 4: Iteration: 100 / 500 [ 20%]  (Warmup)
## Chain 4: Iteration: 150 / 500 [ 30%]  (Warmup)
## Chain 4: Iteration: 200 / 500 [ 40%]  (Warmup)
## Chain 4: Iteration: 250 / 500 [ 50%]  (Warmup)
## Chain 4: Iteration: 251 / 500 [ 50%]  (Sampling)
## Chain 4: Iteration: 300 / 500 [ 60%]  (Sampling)
## Chain 4: Iteration: 350 / 500 [ 70%]  (Sampling)
## Chain 4: Iteration: 400 / 500 [ 80%]  (Sampling)
## Chain 4: Iteration: 450 / 500 [ 90%]  (Sampling)
## Chain 4: Iteration: 500 / 500 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 21.9778 seconds (Warm-up)
## Chain 4:                20.702 seconds (Sampling)
## Chain 4:                42.6798 seconds (Total)
## Chain 4:
```

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

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

```r
summary(mod2)$summary[1:4,]
```

```
##               mean      se_mean          sd         2.5%         25%
## beta0  0.339751015 1.942283e-03 0.040988494  0.262335193  0.31102008
## beta1 -0.009512179 3.389190e-05 0.001100547 -0.011722015 -0.01024998
## beta2  0.871427013 3.453582e-03 0.070695728  0.724029766  0.82545402
## beta3 -0.002161153 6.887645e-05 0.001987597 -0.006303569 -0.00342848
##                50%           75%        97.5%     n_eff      Rhat
## beta0  0.340087429  0.3681391603  0.417906433  445.3474 1.0032182
## beta1 -0.009495342 -0.0087507189 -0.007478782 1054.4481 0.9990313
## beta2  0.871873556  0.9202296049  1.007585533  419.0316 1.0002737
## beta3 -0.002095869 -0.0008463575  0.001741325  832.7515 1.0017249
```

and we can extract the results from mod2:

```r
log_lik_mod2 <- extract(mod2)[['log_lik']]
p_hat_mod2 <- extract(mod2)[['p_hat']]
```

c) Let $t(\boldsymbol{y}) = \sum_{i=1}^{n} 1\left(y_i = 1, a_i < 0.82\right) / \sum_{i=1}^{n} 1\left(a_i < 0.82\right)$ i.e. the proportion of households that switch with arsenic level less than 0.82. Calculate $t(\boldsymbol{y}^{\boldsymbol{rep}})$ for each replicated dataset for each model, plot the resulting histogram for each model and compare to the observed value of $t(\boldsymbol{y})$. Calculate $P\left(t\left(\boldsymbol{y}^{rep}\right) < t(\boldsymbol{y})\right)$ for each model. Interpret your findings.

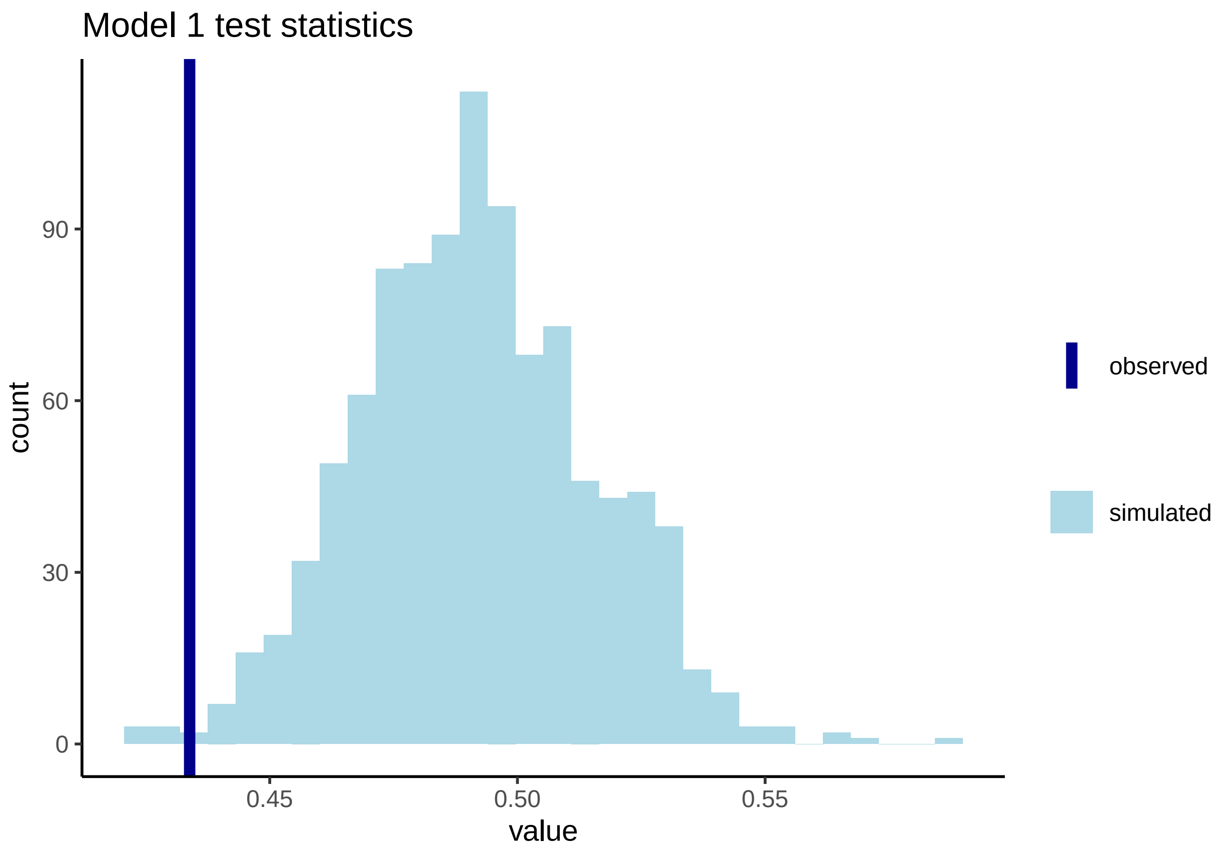
let's find the test statistics:

```r
stat_mod1 <- c()
stat_mod2 <- c()

for (i in 1:nrow(p_hat_mod1)){
  stat_y <- sum(ds$switch ==1 & ds$arsenic<0.82)/sum(ds$arsenic<0.82)
  stat_mod1[i] <- sum(p_hat_mod1[i,] == 1 & ds$arsenic<0.82)/sum(ds$arsenic<0.82)
  stat_mod2[i] <- sum(p_hat_mod2[i,] == 1 & ds$arsenic<0.82)/sum(ds$arsenic<0.82)
}
```

now let's plot the histogram, first for model 1:

```r
#Model 1:

stat_mod1 |>
  as_tibble() |>
  ggplot(aes(value)) +
  geom_histogram(aes(fill = 'simulated'))+
  geom_vline(aes(xintercept = stat_y, color = "observed"), lwd = 2)+
  ggtitle('Model 1 test statistics')+
  scale_color_manual(name = "",
                     values = c("observed" = "darkblue"))+
  scale_fill_manual(name = "",
                     values = c("simulated" = "lightblue")) +
  # the manual change of color idea is adapted from professor's blog post
  theme_classic()
```

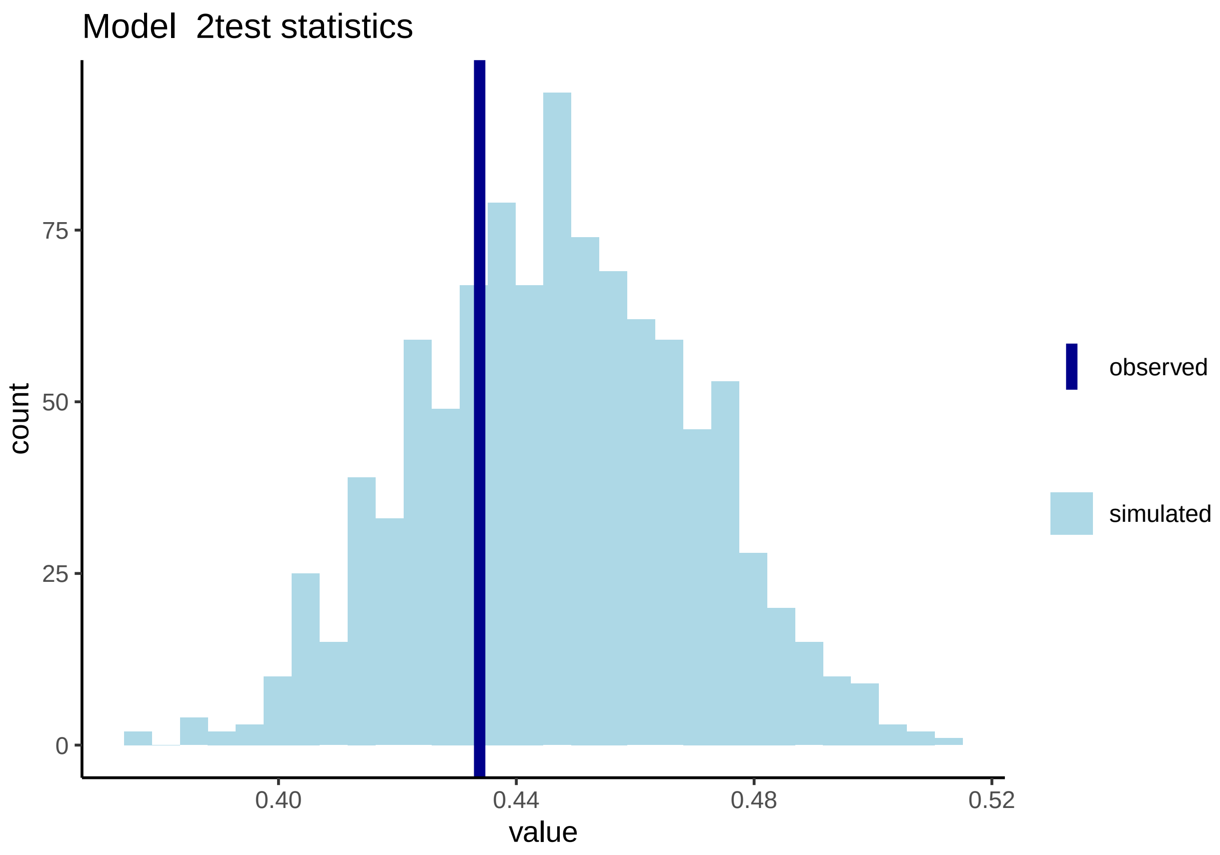
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



now for model 2:

```r
#Model 2:

stat_mod2 |>
  as_tibble() |>
  ggplot(aes(value)) +
  geom_histogram(aes(fill = 'simulated'))+
  geom_vline(aes(xintercept = stat_y, color = "observed"), lwd = 2)+
  ggtitle('Model  2test statistics ')+
  scale_color_manual(name = "",
                     values = c("observed" = "darkblue"))+
  scale_fill_manual(name = "",
                     values = c("simulated" = "lightblue")) +

  # the manual change of color idea is adapted from professor's blog post
  theme_classic()
```

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



To calculate the proportion of the time:

```
# for model 1:

paste('The probability for t(y_rep) <t(y) for model 1 is: ', sum(stat_mod1 < stat_y)/nrow(p_ha
```

```
## [1] "The probability for t(y_rep) <t(y) for model 1 is:  0.006"
```

```
# for model 2:

paste('The probability for t(y_rep) <t(y) for model 2 is: ', sum(stat_mod2 < stat_y)/nrow(p_ha
```

```
## [1] "The probability for t(y_rep) <t(y) for model 2 is:  0.284"
```

as we can see, under a test statistics we used in c), model 2 has a better performance of 28.4% of the statistics we acquired from model 2 is less than the observed statistics, while for model 1 there is only 0.6% chance for its statistics to be less than the observed statistics.

Therefore at this point of time, model 2 hass a better performance.

d) Use the `loo` package to get estimates of the expected log pointwise predictive density for each point, $ELPD_i$. Based on $\sum_i ELPD_i$, which model is preferred?

let's get the ELPD_i here:

```
library(loo)
loo1 <- loo(log_lik_mod1, 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_mod2, 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.
```

and we can store the computed elpd_i in each elpd_i_mod1 and elpd_i_mod2:

```
elpd_i_mod1 <- loo1$pointwise[,1]
elpd_i_mod2 <- loo2$pointwise[,1]
```

compare the sum of elpid_i, we can use loo_compare function:

```
loo_compare(loo1, loo2)
```

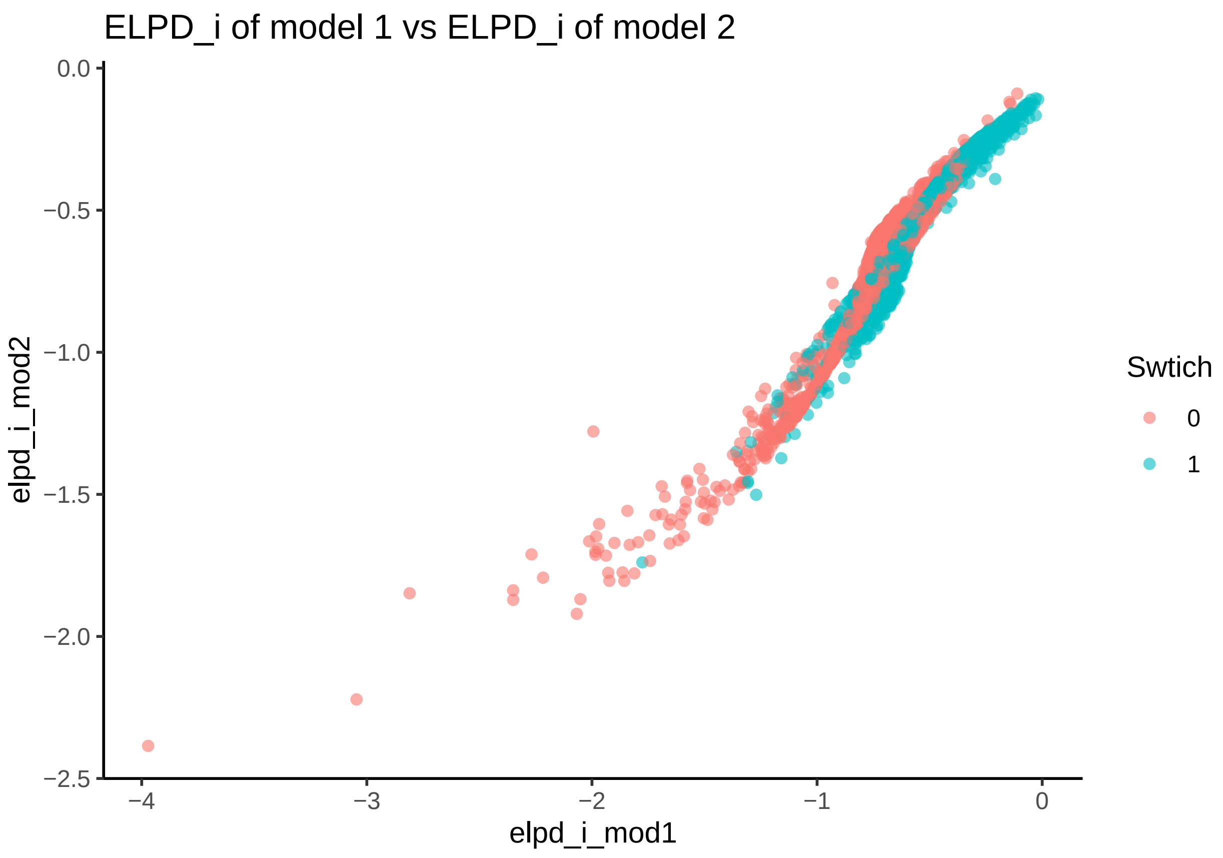
```
##        elpd_diff se_diff
## model2   0.0       0.0  
## model1 -15.4       4.4  
```

Therefore by comparing the sum, model 2 is preferred.

e) Create a scatter plot of the $ELPD_i$'s for Model 2 versus the $ELPD_i$'s for Model 1. Create another scatter plot of the difference in $ELPD_i$'s between the models versus log arsenic. In both cases, color the dots based on the value of y_i. Interpret both plots.

now let us do the first scatter plot elpd_i_mod1 versus elpid_i_mod2

```
tibble(ds$switch,elpd_i_mod1,elpd_i_mod2) |>
  ggplot(aes(x = elpd_i_mod1, y = elpd_i_mod2, color = as.factor(ds$switch))) +
  geom_point(alpha = 0.6) +
  labs(title = 'ELPD_i of model 1 vs ELPD_i of model 2',  color = 'Swtich')+
  theme_classic()
```

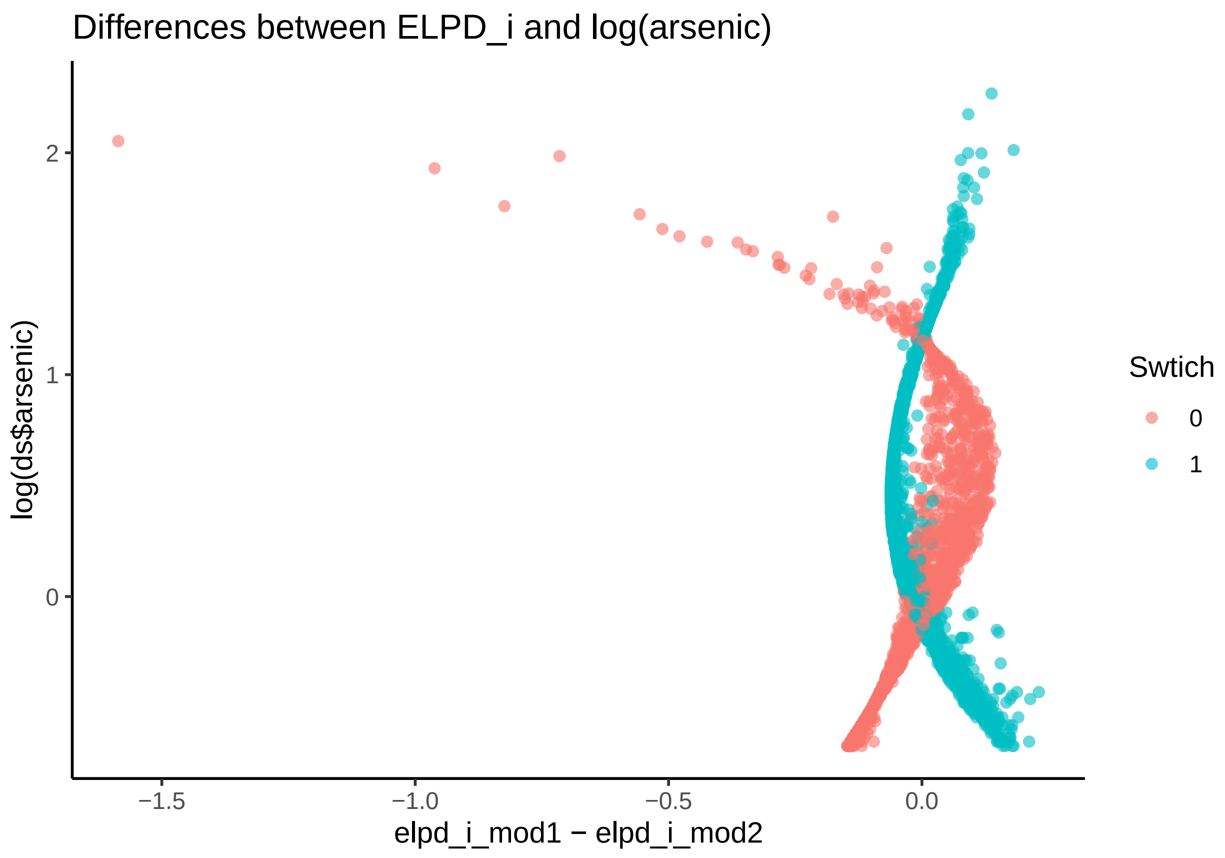


as we can see from the graph, the expected log pointwise predictive density, for when yi = 0 is more evenly distributed from elpd_i_mod1 from -4 to 0, and elpd_i_mod2 from -2.5 to 0, but for yi =1 , it is more centred at elpd_i_mod1 from -2 to 0, and elpd_i_mod2 from -1.5 to 0.

It is also observed that model 2's predictive density is on a smaller range.

Now for the second part, we plot the difference between ELPD_i and the log arsenic:

```
tibble(ds$switch,elpd_i_mod1-elpd_i_mod2, log(ds$arsenic)) |>
  ggplot(aes(x = elpd_i_mod1-elpd_i_mod2, y = log(ds$arsenic), color = as.factor(ds$switch)))
  geom_point(alpha = 0.6) +
  labs(title = 'Differences between ELPD_i and log(arsenic) ',  color = 'Swtich')+
  theme_classic()
```



as we can see, that most of the differences is around 0, with yi = 1 bending leftward, and yi = 0 bending rightward, and extreme values of the differnce is often observed when yi = 0 and the log(arsenic) is high.

f) Given the outcome in this case is discrete, we can directly interpret the $ELPD_i$s. In particular, what is $\exp(ELPD_i)$?

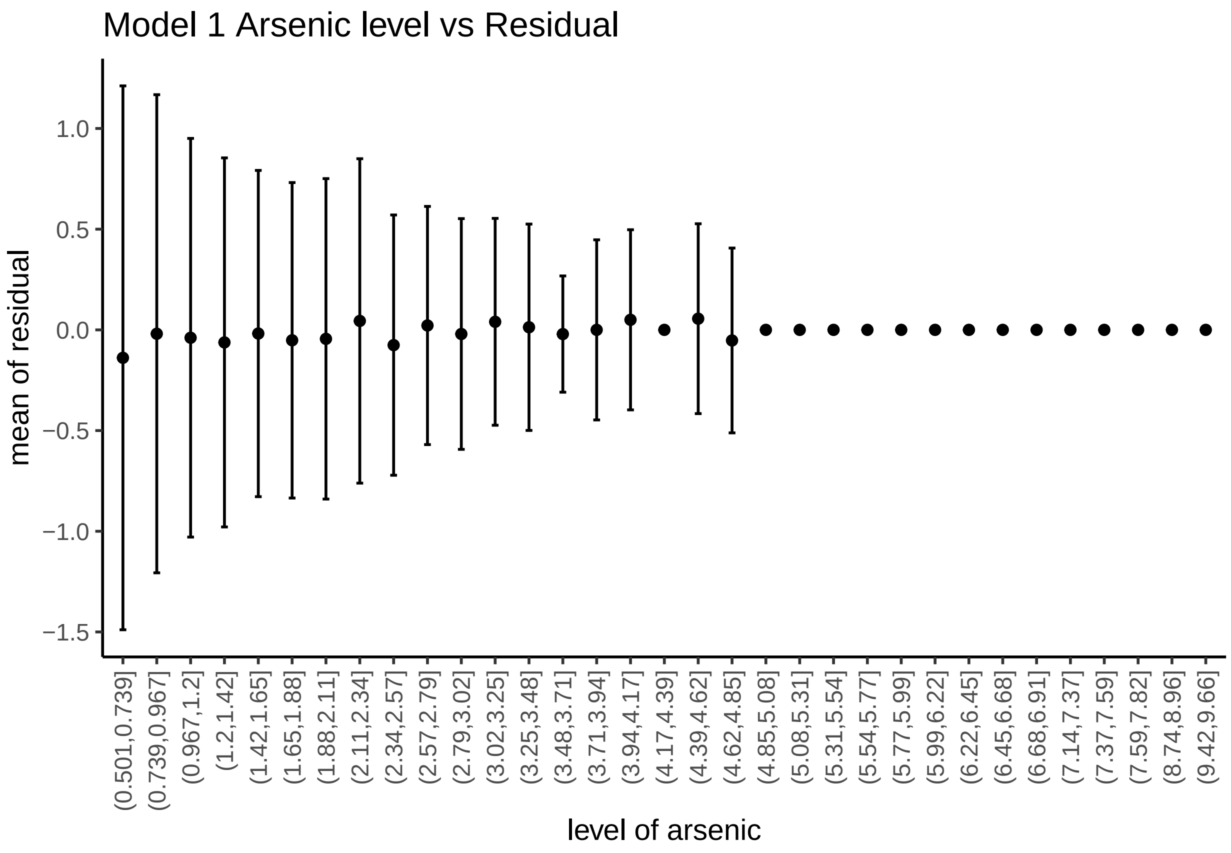
it is the pointwise predictive density at yi given the model is fitted on all the data but yi.

I hope the above explanation was correct…but the general idea is that it is that we fit the model without yi, and use the model to predict the value of yi, so the value is the probability of yi = 1.

g) For each model recode the $ELPD_i$'s to get $\hat{y}_i = E\left(Y_i|\boldsymbol{y}_{-i}\right)$. Create a binned residual plot, looking at the average residual $y_i - \hat{y}_i$ by arsenic for Model 1 and by log(arsenic) for Model 2. Split the data such that there are 40 bins. On your plots, the average residual should be shown with a dot for each bin. In addition, add in a line to represent +/- 2 standard errors for each bin. Interpret the plots for both models.

Let's do the plot for model 1 first:

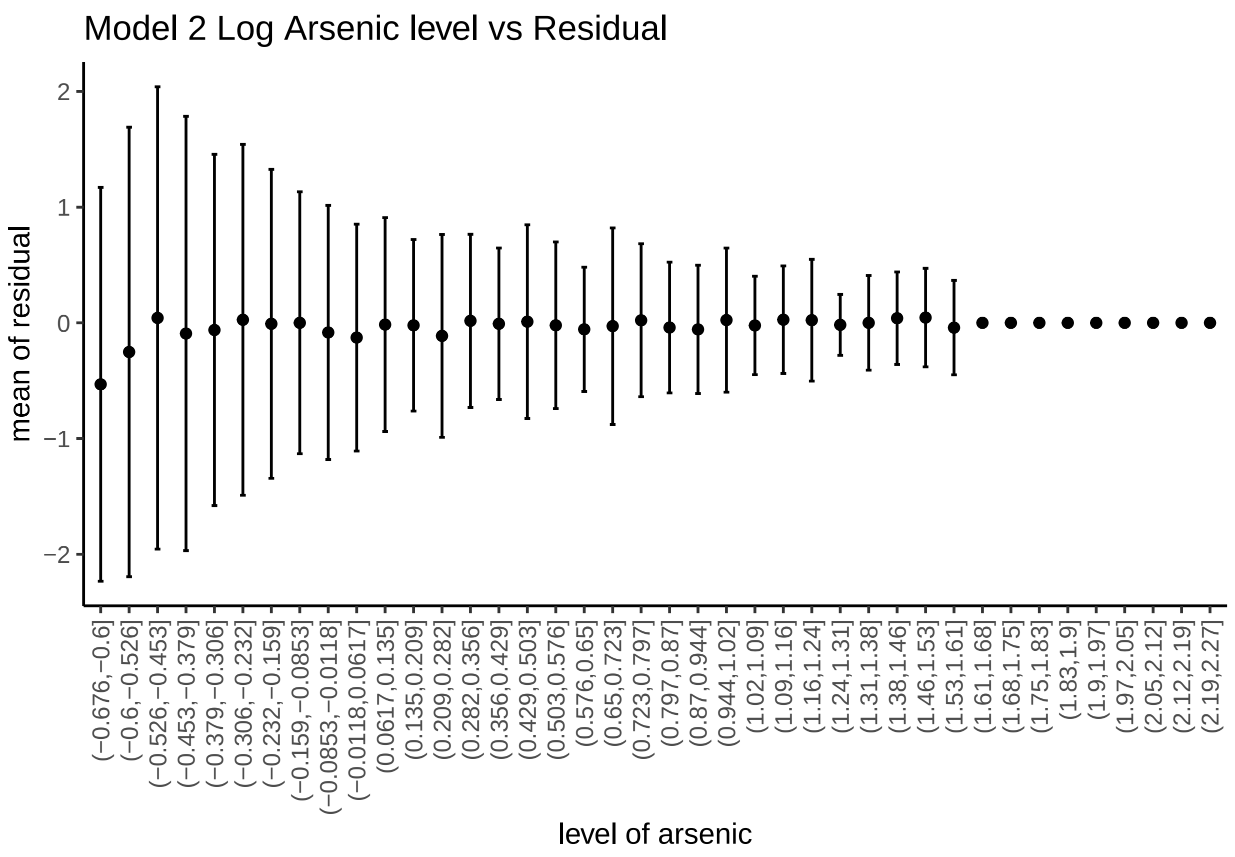
```
y_hat1 <- ifelse(exp(elpd_i_mod1)<=0.5, 0,1)
res <- ds$switch - y_hat1
bins <- cut(ds$arsenic, breaks = 40)

tibble(res, arsenic = ds$arsenic, bins) |>
  group_by(bins) |>
  summarise(mean = mean(res), sd = sd(res)) |>
  ggplot(aes(x = bins, y = mean)) +
  geom_point()+
  geom_errorbar( aes(ymin = mean-2*sd, ymax = mean+2*sd),width = 0.2)+
  labs(title = 'Model 1 Arsenic level vs Residual', y = 'mean of residual', x ='level of arseni
  theme_classic()+
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



as we can see from the above plot, the Model 1's predict less accurately when the arsenic is in the level of 0.501 to 4.85, and predict very accurately for the region of arsenic level greater than 4.85. Here a mean of 0 means all the residuals in that group is 0.

now let's plot for model 2:

```
y_hat2 <- ifelse(exp(elpd_i_mod2)<=0.5, 0,1)
res <- ds$switch - y_hat2
bins <- cut(log(ds$arsenic), breaks = 40)

tibble(res, log_arsenic = log(ds$arsenic), bins) |>
  group_by(bins) |>
  summarise(mean = mean(res), sd = sd(res)) |>
  ggplot(aes(x = bins, y = mean)) +
  geom_point()+
  geom_errorbar( aes(ymin = mean-2*sd, ymax = mean+2*sd),width = 0.2)+
  labs(title = 'Model 2 Log Arsenic level vs Residual', y = 'mean of residual', x ='level of a
  theme_classic()+
theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



as we can see from model 2,on average it has a larger prediction residual compare to that of model 1, and it makes accurate predictions for less bins compare to model1, but this could be because of the way we classify the bins, as well as the ways we classify y_hat. But in general here, model 1 did a better job in terms of prediction, while model2's prediction in is better when log of arsenic level is low.