Fit Models Beyond One Proportion

STAT 341, Spring 2023

2023-02-03

Contents

```r
glimpse(fiji)
```

```
## Rows: 1,006
## Columns: 17
## $ household_id            <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,~
## $ town                    <chr> "Nadi", "Nadi", "Nadi", "Nadi", "Nadi", "Nadi"~
## $ time_point              <chr> "Baseline", "Baseline", "Baseline", "Baseline"~
## $ water_source            <chr> "Catchment", "Borehole", "Borehole", "River/cr~
## $ season                  <chr> "Dry", "Rainy", "Rainy", "Dry", "Dry", "Rainy"~
## $ n_adults                <dbl> 2, 2, 2, 1, 5, 4, 1, 1, 2, 1, 3, 1, 4, 2, 2, 4~
## $ n_kids                  <dbl> 0, 3, 2, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 2, 1, 2~
## $ n_total                 <dbl> 2, 5, 4, 2, 7, 5, 1, 0, 2, 0, 3, 1, 4, 4, 3, 6~
## $ household_annual_income <dbl> 16750.42273, 34132.59160, 2065.90881, 82.37176~
## $ severe_diarrhea_adults  <chr> "Absent", "Absent", "Absent", "Absent", "Absen~
## $ severe_diarrhea_kids    <chr> NA, "Absent", "Absent", "Absent", "Absent", "A~
## $ diarrhea_adults         <chr> "Absent", "Absent", "Absent", "Absent", "Absen~
## $ diarrhea_kids           <chr> NA, "Absent", "Absent", "Absent", "Absent", "A~
## $ medical_expenses_pp     <dbl> 0.0000, 0.0000, 26.4600, 0.0000, 0.0000, 0.002~
## $ water_expenses_pp       <dbl> 2.19, 5.50, 0.24, 9.45, 0.00, 0.01, 0.00, 0.00~
## $ water_expenses          <dbl> 4.38, 27.50, 0.96, 18.90, 0.00, 0.05, 0.00, 0.~
## $ medical_expenses        <dbl> 0.0000, 0.0000, 105.8400, 0.0000, 0.0000, 0.01~
```

What Came Before

We learned some *notation* to keep track of the plan for a model with more than one parameter (but still just one variable).

Now we need to fit it!

One What?

Before we go further…

Apparently we have "a model with more than one parameter (but still just one variable)."

Question 1: In the `water_expenses_pp` model, which is the variable (Hint: it will correspond to a column in your dataset) and what are the parameters (quantities you will estimate, a.k.a. get posteriors for by fitting your model)? *It's crucial to be able to clearly identify which is which as you plan a model!*

> The variable is `water_expenses_pp` .
>
> The parameters is `n_adults` .

Question 2: Which of these may go in a causal diagram: variables, parameters, priors, posteriors, likelihood? (Hint: the causal diagram for the `water_expenses_pp` model we are about to fit will be one node (dot) with *no* arrows.)

> Variables may go in a casual diagram.

Challenges

Fitting this model will not be a straightforward copy of our previous examples, because:

- We have more than one prior (and posterior distribution) to keep track of
- The data can't easily be summarized by two numbers: instead it's a list of n `water_expenses_pp` values.
- We need to think about how to compute the likelihood of each `water_expenses_pp` value *and then* combine them all together into one *joint likelihood* for the whole dataset. If the individual cases are independent, then **the joint likelihood is the product of the individual likelihoods of all the** n datapoints.
- For some conjectured parameter values, the likelihood may be a *tiny* number. If we multiply many of these together, we will quickly get a number so small R thinks it is zero! (No good.) To get around this we need some math.

Log-likelihood

We said before that the joint likelihood of a whole set of independent observations is the product of all the individual likelihoods. Letting y_i be the ith data value, and ℓ the likelihood, we can write this:

$$\mathcal{L}(y_1, y_2, y_3, \ldots y_n) = \prod_{i=1}^{n} \mathcal{L}(y_i)$$

The natural logarithm is a monotonic transformation so if likelihood A is bigger than likelihood B, then the ln(likelihood A) is bigger than ln(likelihood B). So, if we want to work with log-likelihoods instead of likelihoods, we can. We do! Why? Because…

$$ln(\mathcal{L}(y_1, y_2, y_3, \ldots y_n)) = \ell(y_1, y_2, y_3, \ldots y_n) = \sum_{i=1}^{n} \mathcal{L}(y_i)$$

Calculations of the joint likelihood will work much better if we find the **sum** of the log-likelihoods. To get back to the regular likelihood at the end we can do $e^{ln(\mathcal{L})}$ to get $\mathcal{L}$.

Similarly, notice that according to the laws of logarithms,

$$ln(\text{prior} * \mathcal{L}) = ln(\text{prior}) + ln(\mathcal{L})$$

so the un-scaled posterior (prior * likelihood) can also be computed as

$$e^{(ln(\text{prior}) + ln(\mathcal{L}))}$$

R Note

In r, `log()` computes the natural logarithm and `exp()` exponentiates.

Preliminary: Prior Predictive Check

We can simulate data based on our **priors** to get a **prior predictive distribution.** We can use this as a reality check: do the simulated data seem possible? If not, maybe we made some errors in our choices of prior distributions or other model choices (such as modeling our variable using a normal distribution).

My model is going to be for `household_annual_income` in thousands of Fijian dollars:

$$\text{annual income}_i \sim \text{Normal}(\mu, \sigma)$$
$$\mu \sim \text{Normal}(\text{mean}_1 = 50, \text{sd}_1 = 20)$$
$$\sigma \sim \text{Normal}(\text{mean}_2 = 15, \text{sd}_2 = 20)$$

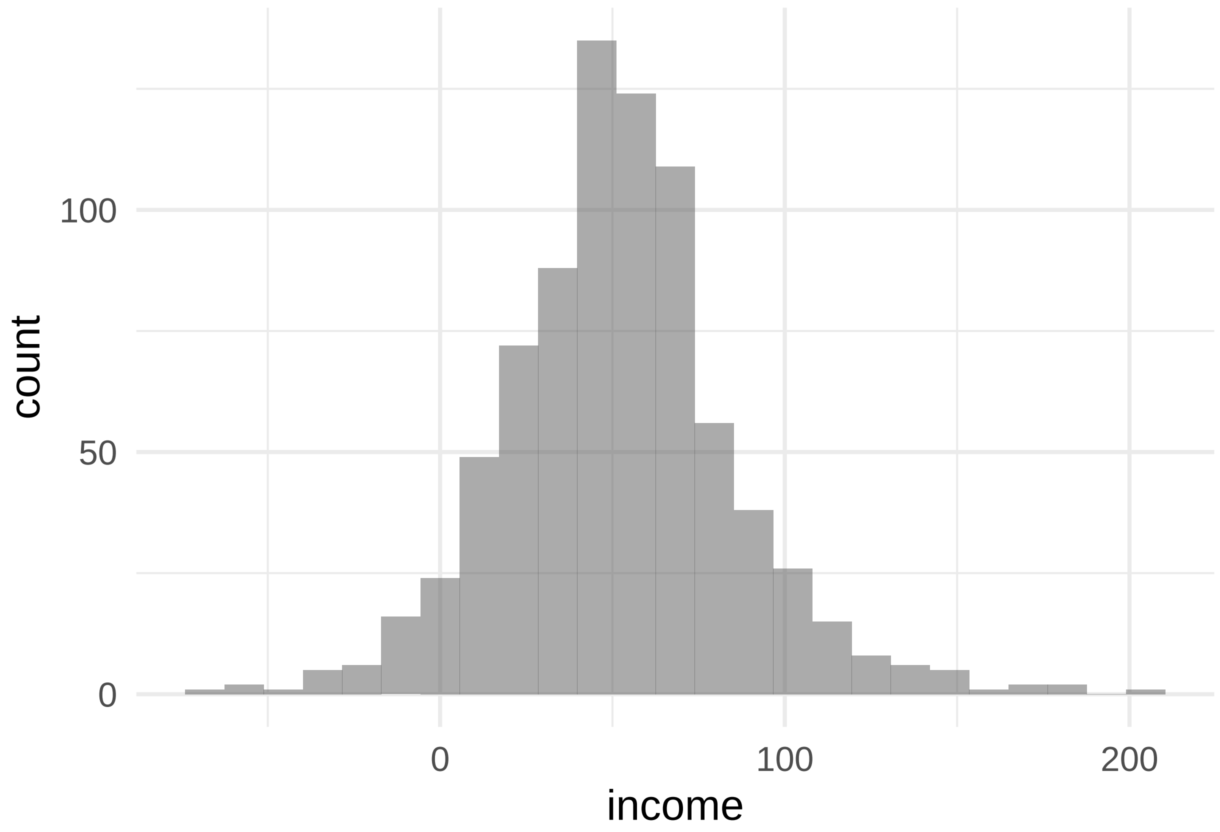
I can simulate data according to that specification:

```r
nsim <- 1000
prior_pred_dist <- tibble(
  # draw nsim values of mu based on the prior for mu ~ Norm(50, 20)
  mu = rnorm(nsim, mean = 50, sd = 20),
  # draw nsim values of sigma based on its prior
  sigma = rnorm(nsim, mean = 15, sd = 20)
) |>
  # changing to "row-wise" calculations
  # R does each calculation on one row at a time
  rowwise() |>
  mutate(
    # use the simulated mu and sigma values
    # to generate simulated incomes
    income = rnorm(1, mean = mu, sd = sigma)
      ) |>
  # go back to normal (not row-wise)
  ungroup()
```

How does it look?

```
gf_histogram(~income,
             data = prior_pred_dist)
```

```
## Warning: Removed 208 rows containing non-finite outside the scale range
## (‘stat_bin()‘).
```



Immediately, I see a problem (I knew I would - that's why I chose this example for myself!).

I generated some negative incomes, which is not possible! I also got a warning about NA incomes. . . can you see why?

```
head(prior_pred_dist, 15)
```

```
## # A tibble: 15 x 3
##       mu  sigma income
##    <dbl>  <dbl>  <dbl>
##  1  50.2   2.25   47.9
##  2  23.9  37.4    29.7
##  3  54.4  19.3    19.7
##  4  77.6  -9.02 NaN
##  5  53.0  75.5   49.7
##  6  33.4  15.9   46.0
##  7  59.7  52.1   17.0
##  8  46.6  20.5   69.4
##  9  55.0  26.5   45.9
## 10  53.1  60.2   -5.06
## 11  80.4  12.1   62.5
## 12  38.4   4.48  41.5
## 13  61.2 -23.8   NaN
## 14  43.0  22.8   25.7
## 15  29.9  55.6   12.2
```

Question 3. Why are some of the income values in my prior predictive distribution `NaN`?

> Standard deviations can't be negative.

So before fitting this model, I should go back to the drawing board!

Your turn: Description and Prior Predictive Distribution

Repeat the process demonstrated above (model description and prior predictive check) for your model for `water_expenses_pp`.

My Grid-search Model Fit

I am going to fit my (stupid unrealistic) model just to demonstrate how to set up the code. Since it already failed the prior predictive check, we should expect unreliable results – *unless* the dataset is big enough to "overrule" any prior wrong-ness…

```r
# number of conjectures to try for each parameter
n_grid = 500
#
grid_income_model <-
  # crossing() generates all possible combinations of the values in the vectors it is given, then sorts
  crossing(
    # possible values to try for mean income (spanning from definitely-too-small to definitely-too-big)
    # units are $1000s
    mu = seq(from = 5, to = 200, length.out = n_grid),
    # possible values to try for sd of income (spanning from definitely-too-small to definitely-too-big
    sigma = seq(from = 7, to = 100, length.out = n_grid)
    )

View(grid_income_model)
```

```r
grid_income_model <- grid_income_model |>
  mutate(
    # evaluate the priors for each mu and sigma in the grid
    prior_mu = dnorm(mu, mean = 50, sd = 20),
    prior_sigma = dnorm(sigma, mean = 15, sd = 20)
  )
```

```r
glimpse(grid_income_model)
```

```
## Rows: 250,000
## Columns: 4
## $ mu          <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5~
## $ sigma       <dbl> 7.000000, 7.186373, 7.372745, 7.559118, 7.745491, 7.931864~
## $ prior_mu    <dbl> 0.001586983, 0.001586983, 0.001586983, 0.001586983, 0.0015~
## $ prior_sigma <dbl> 0.01841351, 0.01848147, 0.01854807, 0.01861329, 0.01867713~
```

```r
grid_income_model <- grid_income_model |>
  # switch to operating row-wise
  rowwise() |>
  mutate(
    # compute the SUM of the LOG-likelihoods for all the rows of the dataset fiji
    logL = dnorm(
      # data. The /1000 is because it's in $1s and I want $1000s
      fiji$household_annual_income/1000,
      # conjectured parameter  values (from grid):
      mean = mu,
      sd = sigma,
      # ask R to return the ln(density) instead of density
      log = TRUE
      ) |>
      # now we have  a list of log-likelihoods as long as the fiji dataset; add them all up to get the
      sum()
    )|>
  # go back to normal (not row-wise)
  ungroup()

glimpse(grid_income_model)
```

```
## Rows: 250,000
## Columns: 5
## $ mu          <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5~
## $ sigma       <dbl> 7.000000, 7.186373, 7.372745, 7.559118, 7.745491, 7.931864~
## $ prior_mu    <dbl> 0.001586983, 0.001586983, 0.001586983, 0.001586983, 0.0015~
## $ prior_sigma <dbl> 0.01841351, 0.01848147, 0.01854807, 0.01861329, 0.01867713~
## $ logL        <dbl> -110163.01, -104697.11, -99641.76, -94956.94, -90607.40, -~
```

```r
grid_income_model <- grid_income_model |>
  mutate(
    # compute the unscaled log-posterior
    # since posterior = likelihood * prior
    # ln(posterior) = ln(likelihood) + ln(prior)
    unscaled_ln_post = logL + log(prior_mu) + log(prior_sigma)
  )
glimpse(grid_income_model)
```

```
## Rows: 250,000
## Columns: 6
## $ mu               <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,~
## $ sigma            <dbl> 7.000000, 7.186373, 7.372745, 7.559118, 7.745491, 7.9~
## $ prior_mu         <dbl> 0.001586983, 0.001586983, 0.001586983, 0.001586983, 0~
## $ prior_sigma      <dbl> 0.01841351, 0.01848147, 0.01854807, 0.01861329, 0.018~
## $ logL             <dbl> -110163.01, -104697.11, -99641.76, -94956.94, -90607.~
## $ unscaled_ln_post <dbl> -110173.45, -104707.54, -99652.19, -94967.37, -90617.~
```

Finally, we need to scale the posterior. Before when we were on the natural (non-logged) scale we did

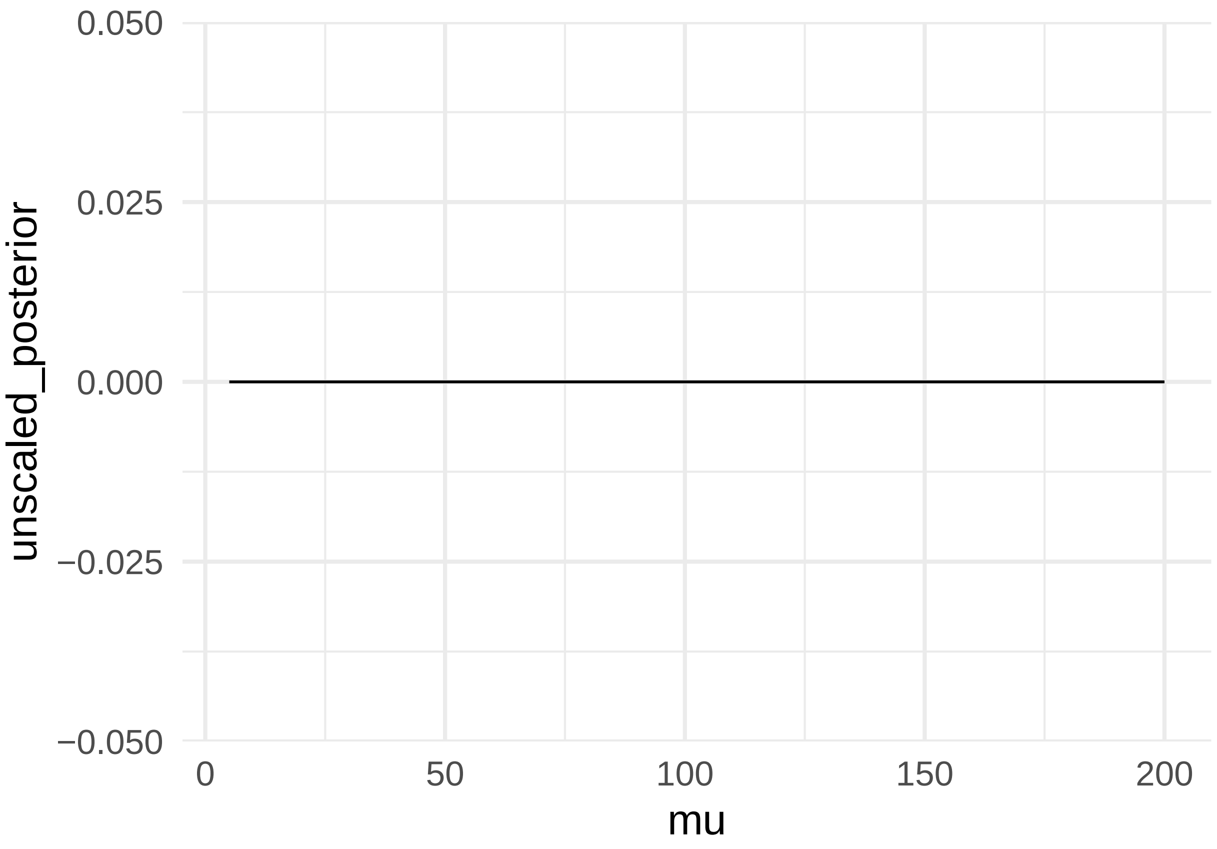
$$\text{posterior}_i = \frac{\text{unscaled posterior}_i}{\sum_{i=1}^{n} \text{unscaled posterior}_i}$$

so we may want to do something like that here: scale so all the posterior values add up to 1. But there are 2 problems with that:

- If we do it all the "probabilities" will be so small they round to zero (computational problems)
- The posterior should be a *normal* distribution so it should *integrate* to 1 not *add up* to 1!

So what if we just stick to the un-scaled posteriors but exponentiate to get back on the natural scale?

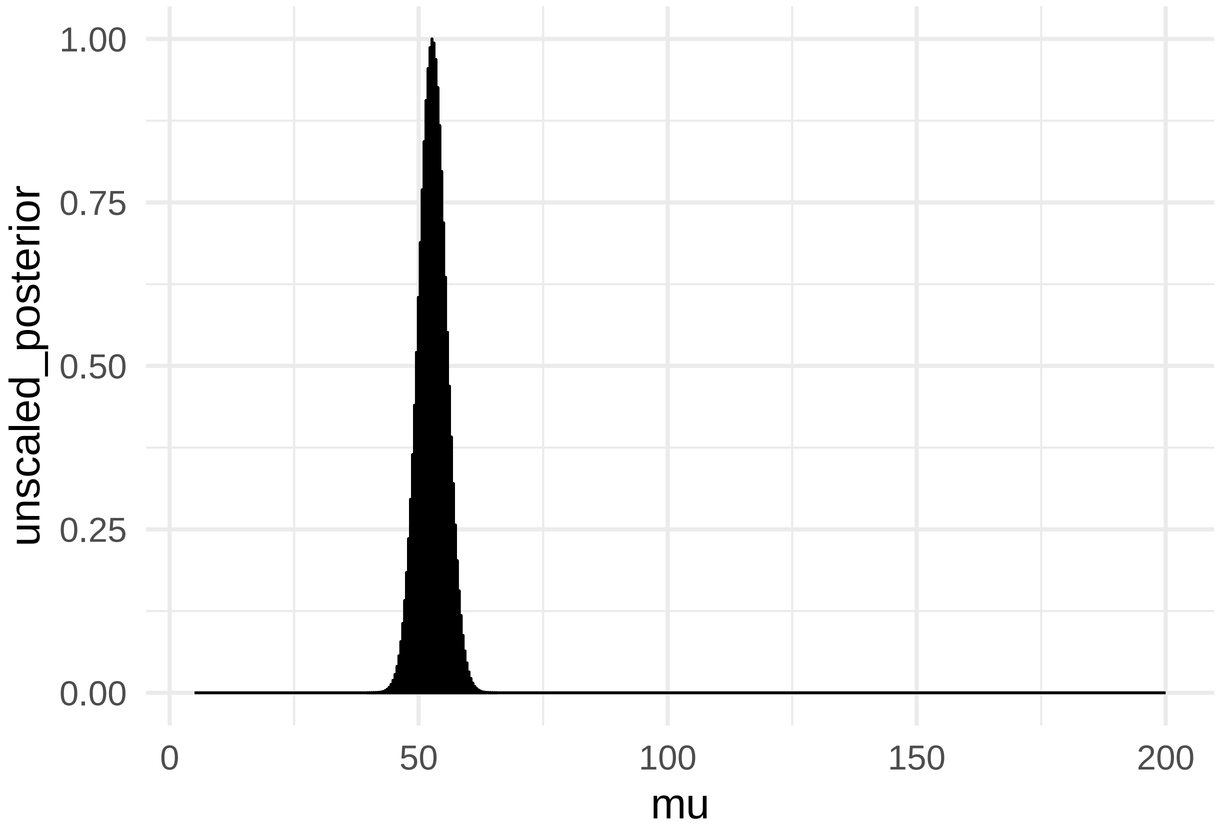
```r
grid_income_model <- grid_income_model |>
  mutate(
    unscaled_posterior = exp(unscaled_ln_post)
  )
gf_line(unscaled_posterior ~ mu,
        data  = grid_income_model)
```



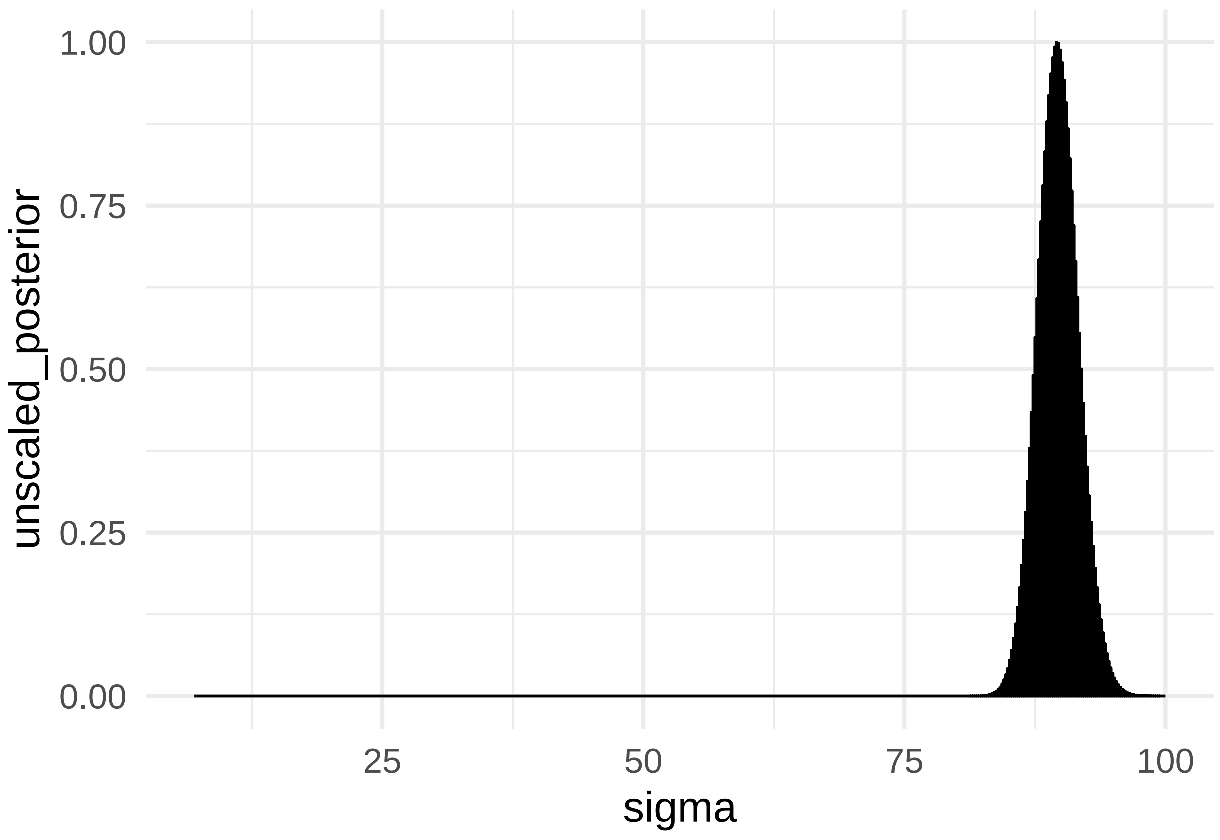
Uh, oh - this is rounding error, not a result. All our unscaled-log-posterior values are so small that when we convert to the non-log scale they all round (incorrectly) to zero.

We only really care about the relative values anyway, so we need to standardize the posterior values somehow such that they won't all round to 0 on the natural scale. Your book's solution is to divide by the maximum un-scaled posterior value. (On the log scale, this is subtracting the max.)

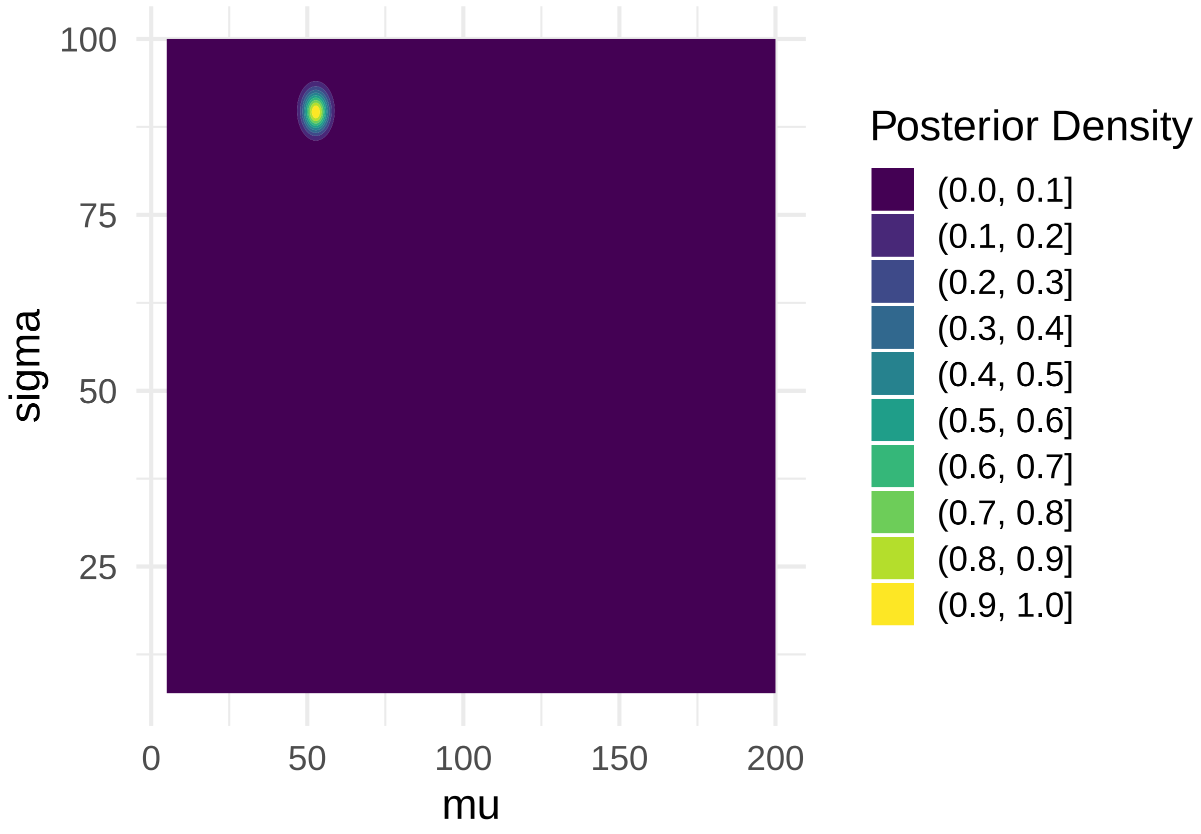
```r
grid_income_model <- grid_income_model |>
  mutate(
    unscaled_posterior = exp(unscaled_ln_post -
                               max(unscaled_ln_post)
                             )
  )
# check out the results...plot the posterior
# each part one by one, then together
gf_line(unscaled_posterior ~ mu,
        data  = grid_income_model)
```



```r
gf_line(unscaled_posterior ~ sigma,
        data  = grid_income_model)
```



```r
gf_contour_filled(unscaled_posterior ~ mu + sigma,
                  data = grid_income_model) |>
  # this is just to change the legend title
  gf_theme(scale_fill_viridis_d("Posterior Density"))
```



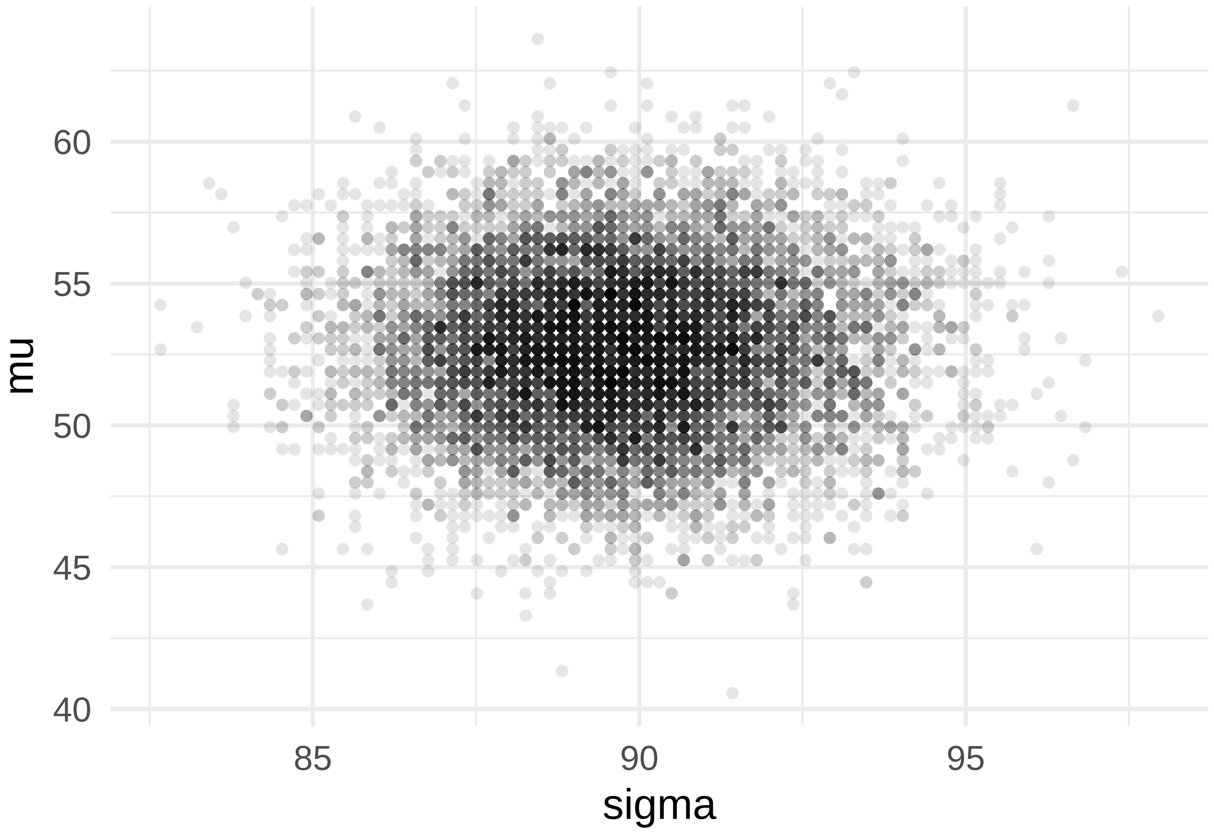
Question 4. Can you explain in words what we learn (about annual household incomes in Fiji) from the three plots of the posterior that were provided?

> The average household income in Fiji is around 63 with high certainty.
>
> The posterior density shows th standard deviation is high, around 80.

Posterior Sample

Draw a sample from my model's posterior.

```r
npsamp <- 10000
grid_post_sample <- grid_income_model |>
  slice_sample(n = npsamp,
               replace = TRUE,
               weight_by = unscaled_posterior)

gf_point(mu ~ sigma, # show both parameters
         data  = grid_post_sample,
         alpha = 0.1 # make points semi-transparent
         )
```



Your Turn!

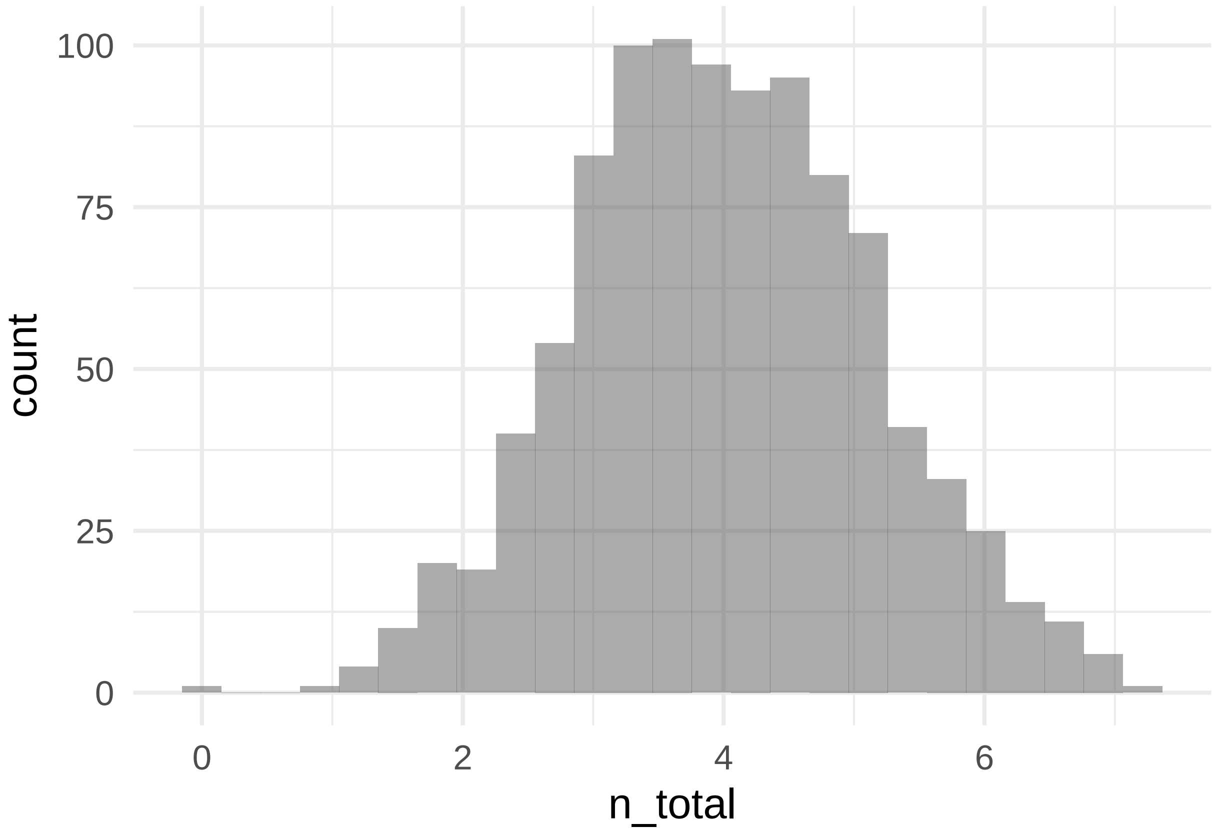
Repeat the process shown above to fit the model via grid search and then obtain and graph a posterior sample, but for *your* model of `water_expenses_pp`.

n_total

$$\mu \sim \text{Normal}(\text{mean} = 4, \text{sd} = 1)$$

$$\sigma \sim \text{Unif}(\text{min} = 0, \text{max} = 1)$$

```r
nsim <- 1000
prior_pred_dist <- tibble(
  # draw nsim values of mu based on the prior for mu ~ Norm(4, 1)
  mu = rnorm(nsim, mean = 4, sd = 1),
  # draw nsim values of sigma based on its prior
  sigma = runif(nsim, min = 0, max = 1)
) |>
  # changing to "row-wise" calculations
  # R does each calculation on one row at a time
  rowwise() |>
  mutate(
    # use the simulated mu and sigma values
    # to generate simulated n_total
    n_total = rnorm(1, mean = mu, sd = sigma)
  ) |>
  # go back to normal (not row-wise)
  ungroup()

# Plot the prior predictive distribution
gf_histogram(~n_total,
             data = prior_pred_dist)
```



```
# number of conjectures to try for each parameter
n_grid = 500
#
grid_n_total_model <-
  # crossing() generates all possible combinations of the values in the vectors it is given, then sorts
  crossing(
    # possible values to try for mean n_total (spanning from definitely-too-small to definitely-too-big
    mu = seq(from = 0, to = 10, length.out = n_grid),
    # possible values to try for sd of n_total (spanning from definitely-too-small to definitely-too-bi
    sigma = seq(from = 0.1, to = 2, length.out = n_grid)
  )

View(grid_n_total_model)
```

```
grid_n_total_model <- grid_n_total_model |>
  mutate(
    # evaluate the priors for each mu and sigma in the grid
    prior_mu = dnorm(mu, mean = 4, sd = 1),
    prior_sigma = dunif(sigma, min = 0, max = 1)
  )

glimpse(grid_n_total_model)
```

```
## Rows: 250,000
## Columns: 4
## $ mu          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ sigma       <dbl> 0.1000000, 0.1038076, 0.1076152, 0.1114228, 0.1152305, 0.1~
## $ prior_mu    <dbl> 0.0001338302, 0.0001338302, 0.0001338302, 0.0001338302, 0.~
## $ prior_sigma <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
```

```
grid_n_total_model <- grid_n_total_model |>
  # switch to operating row-wise
  rowwise() |>
  mutate(
    # compute the SUM of the LOG-likelihoods for all the rows of the dataset fiji
    logL = dnorm(
      # data
      fiji$n_total,
      # conjectured parameter values (from grid):
      mean = mu,
      sd = sigma,
      # ask R to return the ln(density) instead of density
      log = TRUE
      ) |>
      # now we have a list of log-likelihoods as long as the fiji dataset; add them all up to get the j
      sum()
    )|>
  # go back to normal (not row-wise)
  ungroup()

glimpse(grid_n_total_model)
```

```
## Rows: 250,000
## Columns: 5
## $ mu          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ sigma       <dbl> 0.1000000, 0.1038076, 0.1076152, 0.1114228, 0.1152305, 0.1~
## $ prior_mu    <dbl> 0.0001338302, 0.0001338302, 0.0001338302, 0.0001338302, 0.~
## $ prior_sigma <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ logL        <dbl> -915308.1, -849330.7, -790234.4, -737094.8, -689137.7, -64~
```

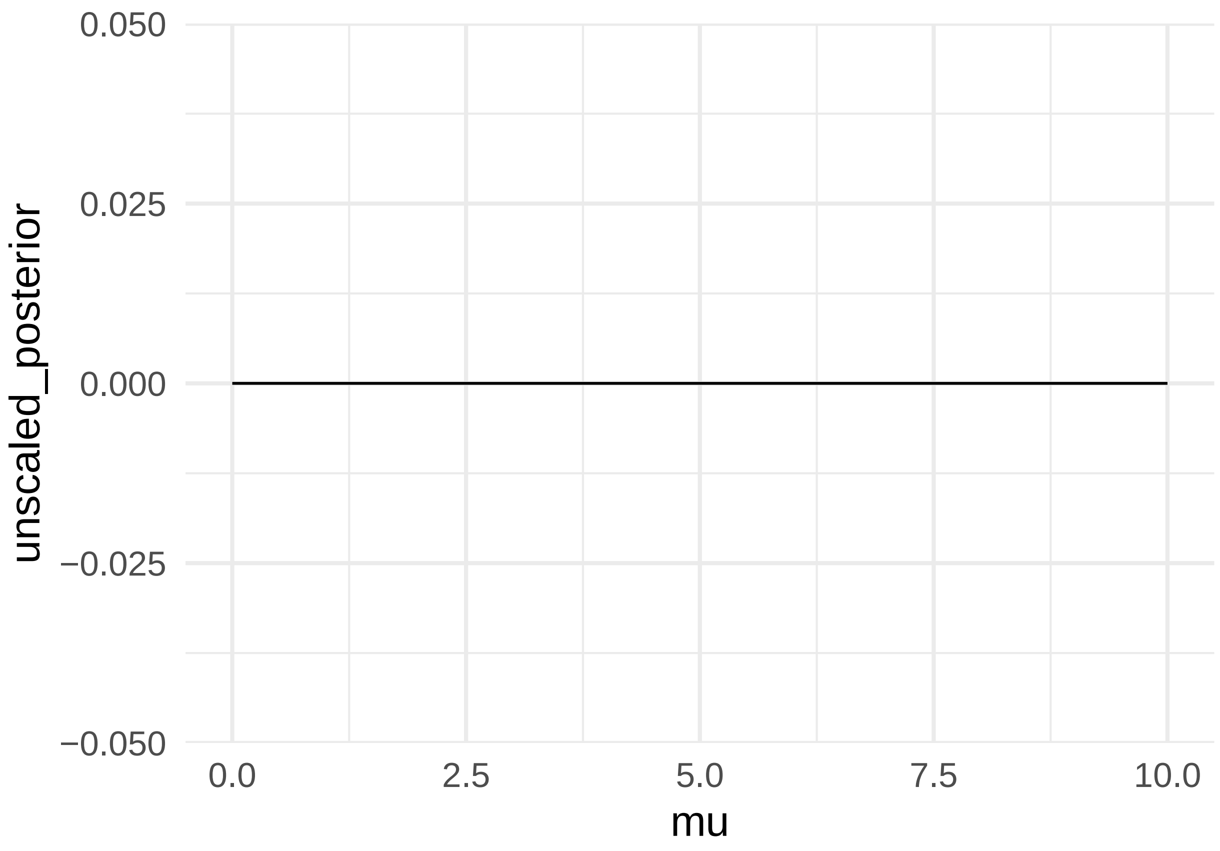
```
grid_n_total_model <- grid_n_total_model |>
  mutate(
    # compute the unscaled log-posterior
    # since posterior = likelihood * prior
    # ln(posterior) = ln(likelihood) + ln(prior)
    unscaled_ln_post = logL + log(prior_mu) + log(prior_sigma)
  )
glimpse(grid_n_total_model)
```

```
## Rows: 250,000
## Columns: 6
## $ mu                <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ sigma             <dbl> 0.1000000, 0.1038076, 0.1076152, 0.1114228, 0.1152305~
## $ prior_mu          <dbl> 0.0001338302, 0.0001338302, 0.0001338302, 0.000133830~
## $ prior_sigma       <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
## $ logL              <dbl> -915308.1, -849330.7, -790234.4, -737094.8, -689137.7~
## $ unscaled_ln_post <dbl> -915317.0, -849339.6, -790243.4, -737103.8, -689146.6~
```

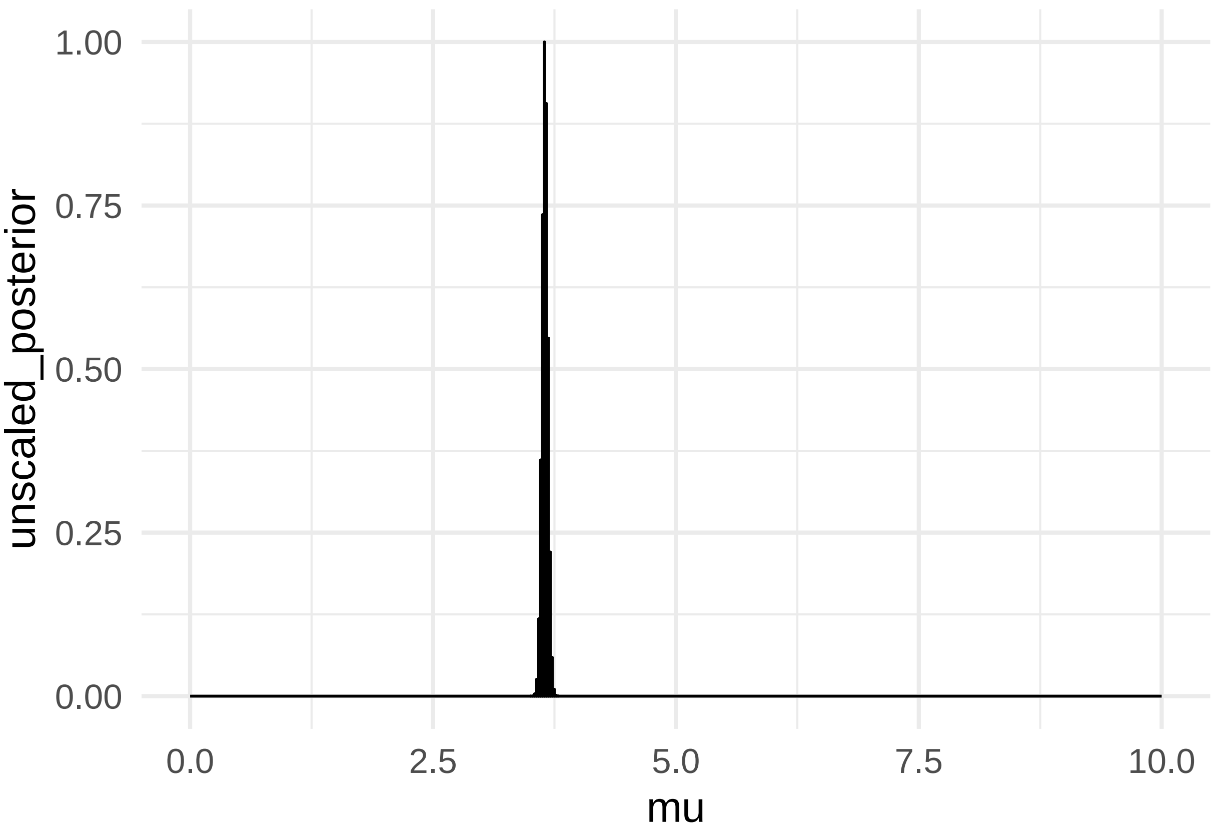
```
grid_n_total_model <- grid_n_total_model |>
  mutate(
    # compute the unscaled log-posterior
    # since posterior = likelihood * prior
    # ln(posterior) = ln(likelihood) + ln(prior)
    unscaled_ln_post = logL + log(prior_mu) + log(prior_sigma)
  )
glimpse(grid_n_total_model)
```

```
## Rows: 250,000
## Columns: 6
## $ mu                <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ sigma             <dbl> 0.1000000, 0.1038076, 0.1076152, 0.1114228, 0.1152305~
## $ prior_mu          <dbl> 0.0001338302, 0.0001338302, 0.0001338302, 0.000133830~
## $ prior_sigma       <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
## $ logL              <dbl> -915308.1, -849330.7, -790234.4, -737094.8, -689137.7~
## $ unscaled_ln_post <dbl> -915317.0, -849339.6, -790243.4, -737103.8, -689146.6~
```

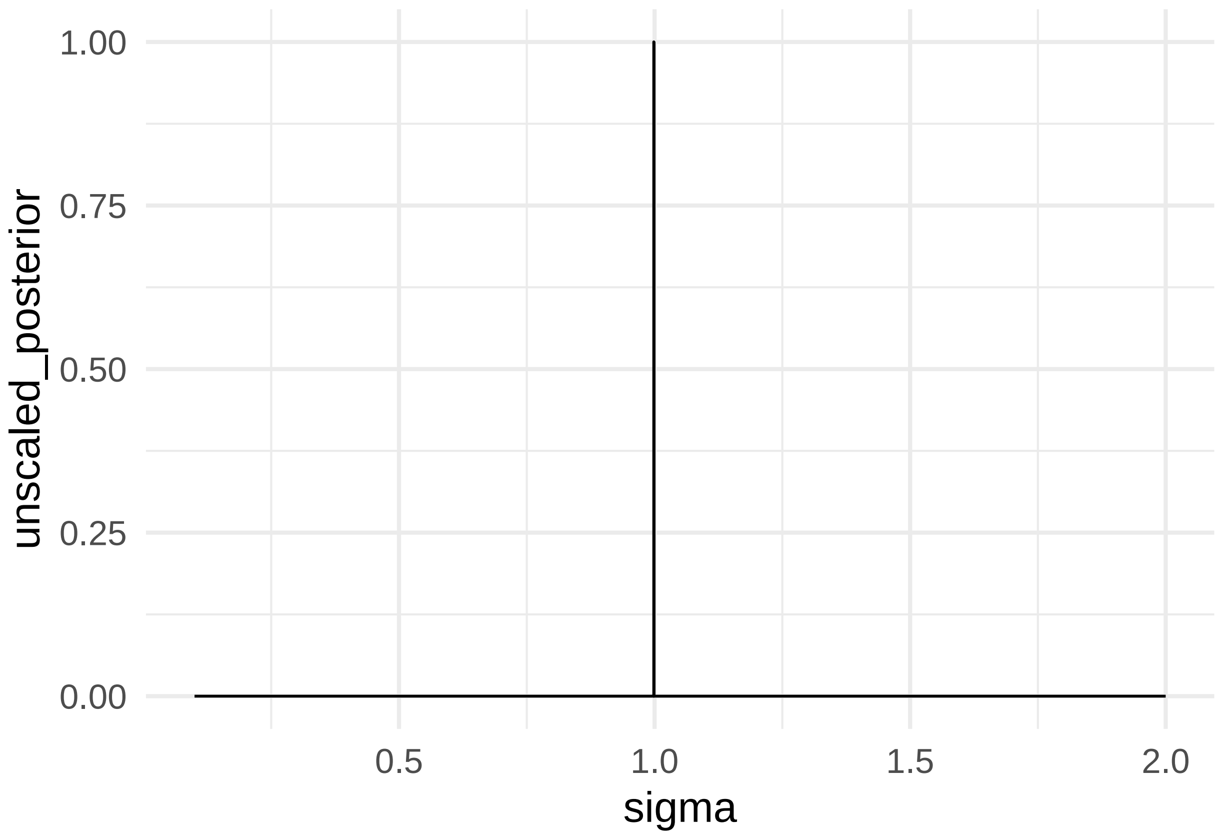
```
grid_n_total_model <- grid_n_total_model |>
  mutate(
    unscaled_posterior = exp(unscaled_ln_post)
  )
gf_line(unscaled_posterior ~ mu,
        data = grid_n_total_model)
```



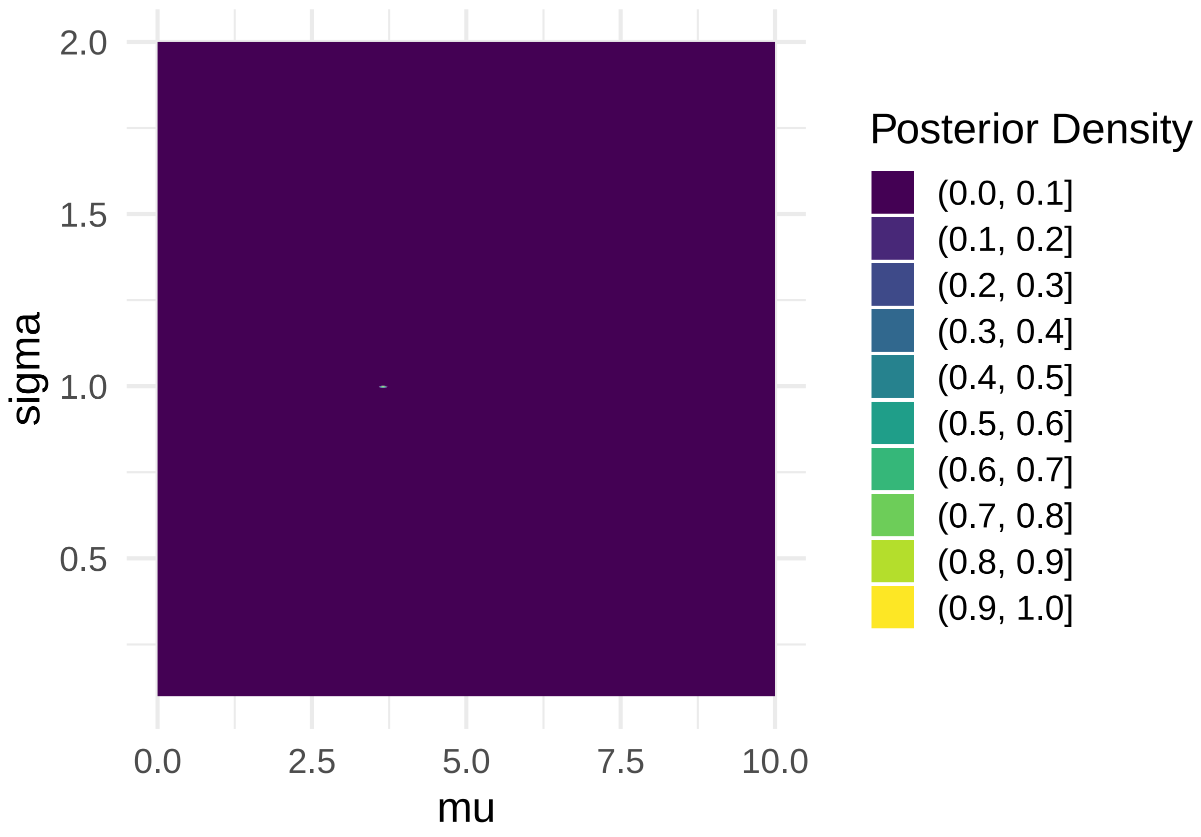
```r
grid_n_total_model <- grid_n_total_model |>
  mutate(
    unscaled_posterior = exp(unscaled_ln_post - max(unscaled_ln_post))
  ) |>
  filter(is.finite(unscaled_posterior))

# check out the results...plot the posterior
# each part one by one, then together
gf_line(unscaled_posterior ~ mu,
        data  = grid_n_total_model)
```



```r
gf_line(unscaled_posterior ~ sigma,
        data  = grid_n_total_model)
```

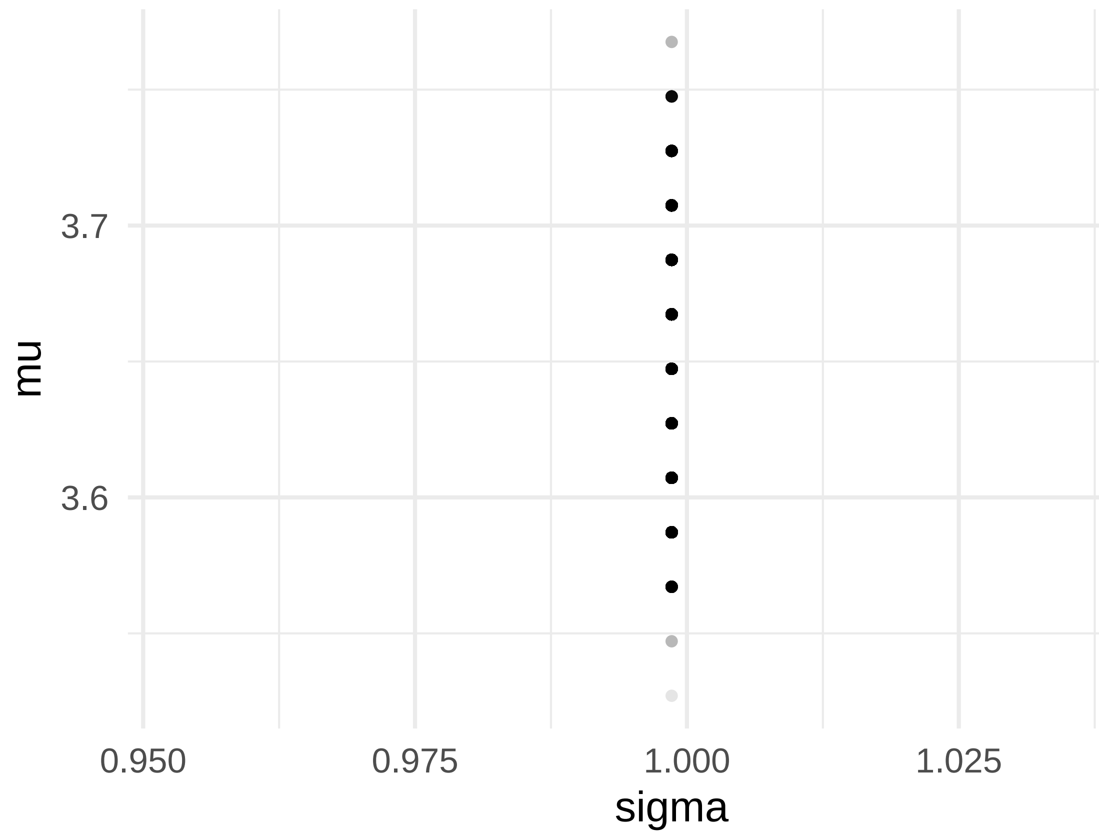


```r
gf_contour_filled(unscaled_posterior ~ mu + sigma,
                  data = grid_n_total_model) |>
  # this is just to change the legend title
  gf_theme(scale_fill_viridis_d("Posterior Density"))
```



```r
npsamp <- 10000
grid_post_sample <- grid_n_total_model |>
  slice_sample(n = npsamp,
               replace = TRUE,
               weight_by = unscaled_posterior)

gf_point(mu ~ sigma, # show both parameters
         data  = grid_post_sample,
         alpha = 0.1 # make points semi-transparent
         )
```

Question 5. Describe, show or sketch the posterior you end up with. (Since the dataset is pretty large, we should all agree pretty closely in our conclusions even if we have somewhat dif-



ferent prior choices. . .)

> The average number of people in a household is around 3.65, with a standard deviation centered around 1.

What's next

We have run into the limitations of grid search (computationally, and in terms of code complexity) and will bid it farewell oh-so-soon. How can we use `quap()` to fit these models? *Feel free to give it a try - we will do it together next week!*