Homework 4

PSTAT 115, Winter 2022

Due on February 27, 2022 at 11:59 pm

Note: If you are working with a partner, please submit only one homework per group with both names and whether you are taking the course for graduate credit or not. Submit your Rmarkdown (.Rmd) and the compiled pdf on Gauchospace.

Problem 1. Frequentist Coverage of The Bayesian Posterior Interval.

In the "random facts calibration game" we explored the importance and difficulty of well-calibrated prior distributions by examining the calibration of subjective intervals. Suppose that $y_1, .., y_n$ is an IID sample from a $Normal(\mu, 1)$. We wish to estimate μ.

1a. For Bayesian inference, we will assume the prior distribution $\mu \sim Normal(0, \frac{1}{\kappa_0})$ for all parts below. Remember, from lecture that we can interpret κ_0 as the pseudo-number of prior observations with sample mean $\mu_0 = 0$. State the posterior distribution of μ given $y_1, .., y_n$. Report the lower and upper bounds of the 95% quantile-based posterior credible interval for μ, using the fact that for a normal distribution with standard eviation σ, approximately 95% of the mass is between $\pm 1.96\sigma$.

We know the Posterior Distribution is as follows: $p(\theta|y, \sigma^2) = L(\mu) * p(\mu)$

Given $Y_i and \mu$ are normal, the posterior is also normally distributed $p(\mu|y, \sigma) \propto L(\mu) * p(\mu)$

So, the posterior distribution can be justifies as $N(\mu_0, \tau_0^2)$.

In our conjugacy model the parameters $\mu_0 = \frac{\frac{1}{\tau_0^2}\mu_0 + \frac{n}{\sigma^2}\bar{y}}{\frac{1}{\tau_0^2} + \frac{n}{\sigma^2}}$, and $\tau_0^2 = \frac{1}{\frac{1}{\tau_0^2} + \frac{n}{\sigma^2}}$.

In order to find the 95% confidence interval our interval we will need to plug in values for τ_0^2 in terms of K_0 and n.

We are given a sampling variance of 1 and prior variance $1/k_0$. We also know our prior mean is 0.

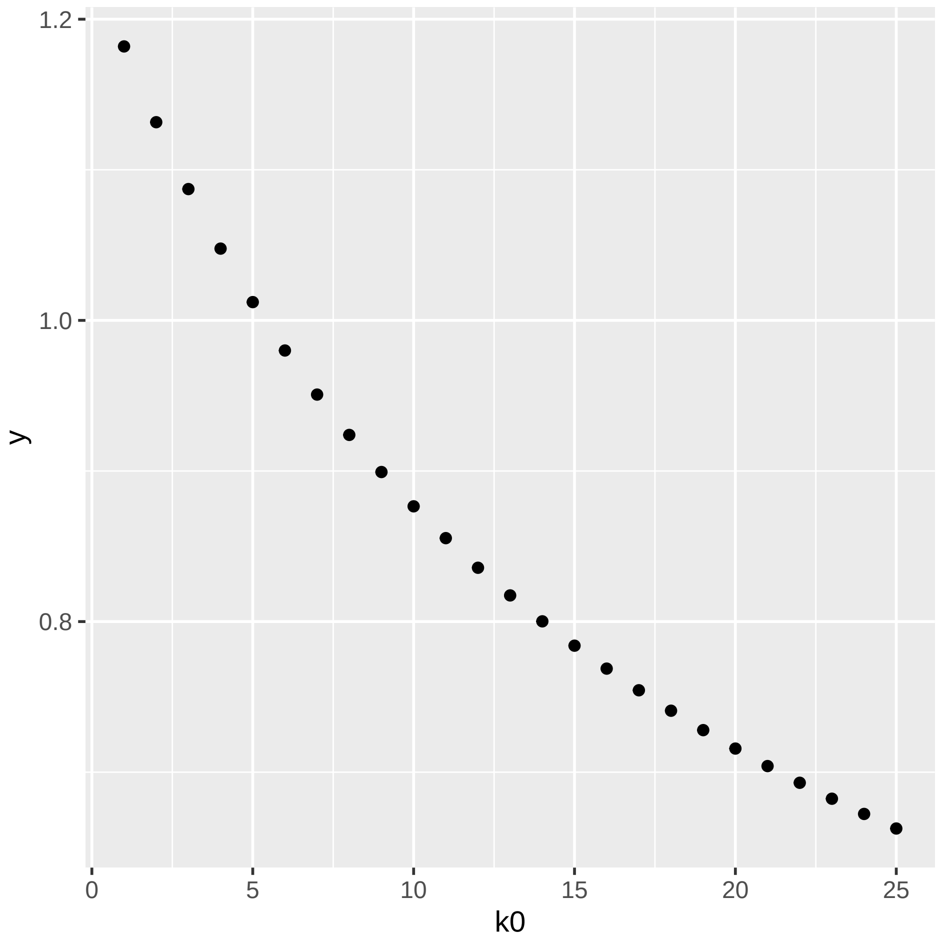
We can now establish a lower and upper bounds for the model: $1.96 * \frac{1}{\frac{1}{\tau_0^2} + \frac{n}{\sigma^2}}$

Upper Bound: $1.96\sqrt{\frac{1}{n+k_0}}$ Lower Bound: $-1.96\sqrt{\frac{1}{n+k_0}}$

1b. Plot the length of the posterior credible interval as a function of κ_0, for $\kappa_0 = 1, 2, ..., 25$ assuming $n = 10$. Report how this prior parameter effects the length of the posterior interval and why this makes intuitive sense.

```r
k0 <- seq(1,25)
n = 10
values = c()
#Function
length_post_cred <- function(k0)
  {
    2*1.96*sqrt(1/(n+k0))
}
```

```
#Plot
interval_length <- ggplot(data.frame(k0),aes(k0)) + stat_function(fun=length_post_cred,geom= 'point', n
interval_length
```



```
. = ottr::check("tests/q1b.R")
```

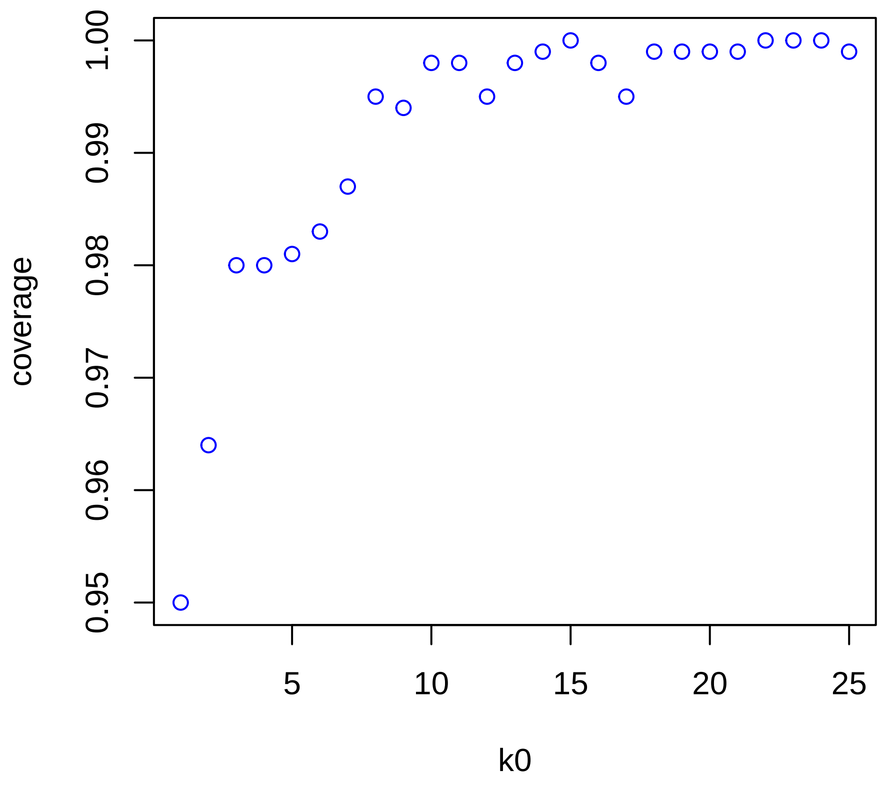
```
## Test q1b - 1 failed:
## The length of interval_length should be 25.
## Test failed
##
## Test q1b - 2 failed:
## Error in computation of int_length.
## Test failed
```

From the plot above we can conclude that as we get more samples and k0 increases, the variance for the estimate of MU decreases. This may suggest that our confidence in our estimate is increasing.

1c. Now we will evaluate the *frequentist coverage* of the posterior credible interval on simulated data. Generate 1000 data sets where the true value of $\mu = 0$ and $n = 10$. For each dataset, compute the posterior 95% interval endpoints (from the previous part) and see if it the interval covers the true value of $\mu = 0$. Compute the frequentist coverage as the fraction of these 1000 posterior 95% credible intervals that contain $\mu = 0$. Do this for each value of $\kappa_0 = 1, 2, ..., 25$. Plot the coverage as a function of κ_0. Store these 25 coverage values in vector called `coverage`.

```
## Fill in the vector called "coverage", which stores the fraction of intervals containing \mu = 0 for 
true_mu <- 0
n <- 10
data_count <- 1000
dataset <- matrix(0,data_count,length(k0))
coverage <- numeric(25)

for (k in k0) {
    for (data in seq(data_count)) {
        y <- rnorm(n,true_mu,1)
        post_mu <- (mean(y)*n/(k+n))
        cred_int <- qnorm(c(0.025,0.975),post_mu,sqrt(1/(k+n)))
        if(between(true_mu,cred_int[1],cred_int[2])==TRUE){
            dataset[data,k] <- 1
        }
    }
    coverage[k] <- sum(dataset[,k])/data_count
}
#plot
plot(k0,coverage,col="blue")
```



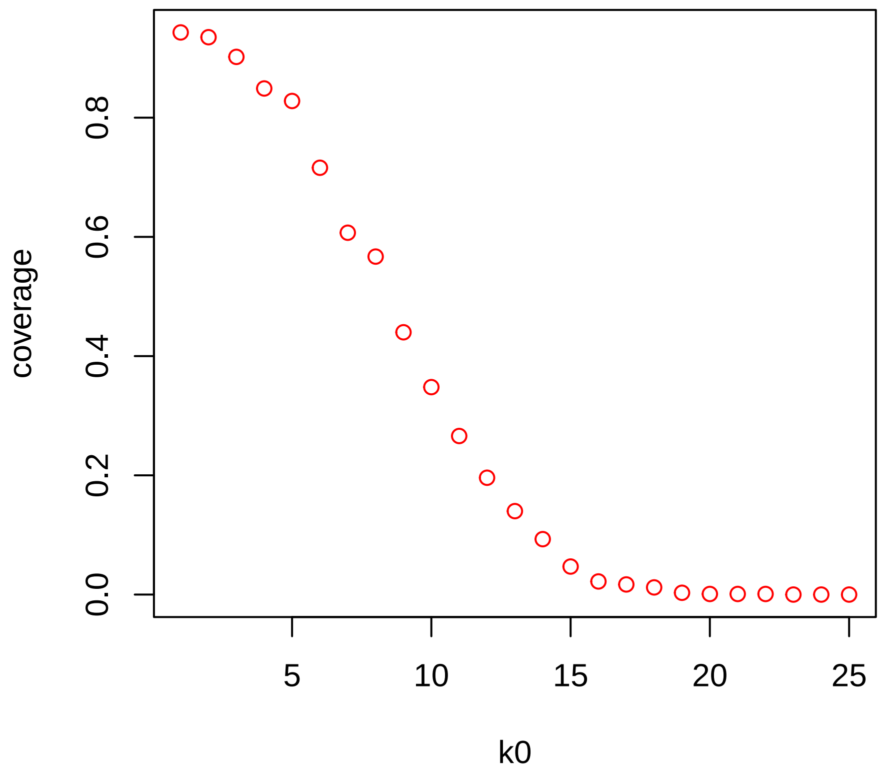
```
. = ottr::check("tests/q1c.R")
```

```
##
## All tests passed!
```

1d. Repeat 1c but now generate data assuming the true $\mu = 1$. Again, store these 25 coverage values in vector called `coverage`.

```
## Fill in the vector called "coverage", which stores the fraction of intervals containing \mu = 1 for
mu2 <- 1
data_count <- 1000
dataset <- matrix(0,data_count,length(k0))
coverage <- numeric(25)

for (k in k0) {
    for (data in seq(data_count)) {
        y <- rnorm(n,mu2,1)
        post_mu2 <- (mean(y)*n/(k+n))
        cred_int <- qnorm(c(0.025,0.975),post_mu2,sqrt(1/(k+n)))
        if(between(mu2,cred_int[1],cred_int[2])==TRUE){
            dataset[data,k] <- 1
        }
    }
    coverage[k] <- sum(dataset[,k])/data_count
}
#plot
plot(k0,coverage,col="red")
```



```r
. = ottr::check("tests/q1d.R")
```

```
##
## All tests passed!
```

1e. Explain the differences between the coverage plots when the true $\mu = 0$ and the true $\mu = 1$. For what values of κ_0 do you see closer to nominal coverage (i.e. 95%)? For what values does your posterior interval tend to over cover (the interval covers the true value more than 95% of the time)? Undercover (the interval covers the true value less than 95% of the time)? Why does this make sense?

When $\mu = 0$, the posterior interval tends to over cover when k is greater than 2 and when k equals 1; It also tends to undercover when k is greater than 2. we see that as K_0 increases, the nominal coverage increases for $\mu = 0$ but decreases for $\mu = 1$.

Problem 2. Goal Scoring in the Women's World Cup

The Chinese Women's soccer team recently won the AFC Women's Asian Cup. Suppose you are interested in studying the World Cup performance of this soccer team. Let λ be the be the average number of goals scored by the team. We will analyze λ using the Gamma-Poisson model where data Y_i is the observed number of goals scored in the ith World Cup game, ie. we have $Y_i|\lambda \sim Pois(\lambda)$. *A priori*, we expect the rate of goal scoring to be $\lambda \sim Gamma(a, b)$. According to a sports analyst, they believe that λ follows a Gamma distribution with $a = 1$ and $b = 0.25$.

2a. Compute the theoretical posterior parameters a, b, and also the posterior mean.

```r
library(rstan)
y <- c(4, 7, 3, 2, 3) # Number of goals in each game
n <- length(y)

a = 1
b = 0.25
post_a <- a + sum(y)
post_b <- b + length(y)
post_mu <- post_a/post_b
post_mu
```

```
## [1] 3.809524
```

```r
. = ottr::check("tests/q2a.R")
```

2b. Create a new Stan file by selecting "Stan file" and name it `women_cup.stan`, use Rstan to report and estimate the posterior mean of the scoring rate by computing the sample average of all Monte Carlo samples of λ.

```r
soccer_model <- stan_model("women_cup.stan")
```

```
## Warning in readLines(file, warn = TRUE): incomplete final line found on '/home/
## jovyan/winter22/homeworks/homework4/women_cup.stan'

## Trying to compile a simple C file

## Running /opt/conda/lib/R/bin/R CMD SHLIB foo.c
## x86_64-conda-linux-gnu-cc -I"/opt/conda/lib/R/include" -DNDEBUG   -I"/opt/conda/lib/R/library/Rcpp/i
## In file included from /opt/conda/lib/R/library/RcppEigen/include/Eigen/Core:88,
##                  from /opt/conda/lib/R/library/RcppEigen/include/Eigen/Dense:1,
##                  from /opt/conda/lib/R/library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:
##                  from <command-line>:
## /opt/conda/lib/R/library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type n
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /opt/conda/lib/R/library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /opt/conda/lib/R/library/RcppEigen/include/Eigen/Dense:1,
##                  from /opt/conda/lib/R/library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:
##                  from <command-line>:
## /opt/conda/lib/R/library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or d
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/opt/conda/lib/R/etc/Makeconf:170: foo.o] Error 1
```

```r
stan_fit1 <- rstan::sampling(soccer_model,
                             data = list(N=n,y=y),
                             refresh = 0)
samples1 <- rstan::extract(stan_fit1)

lambda_samples <- samples1$lambda

post_mean <- mean(lambda_samples) #help
post_mean
```

```
## [1] 3.836673
```

```
. = ottr::check("tests/q2b.R")
```

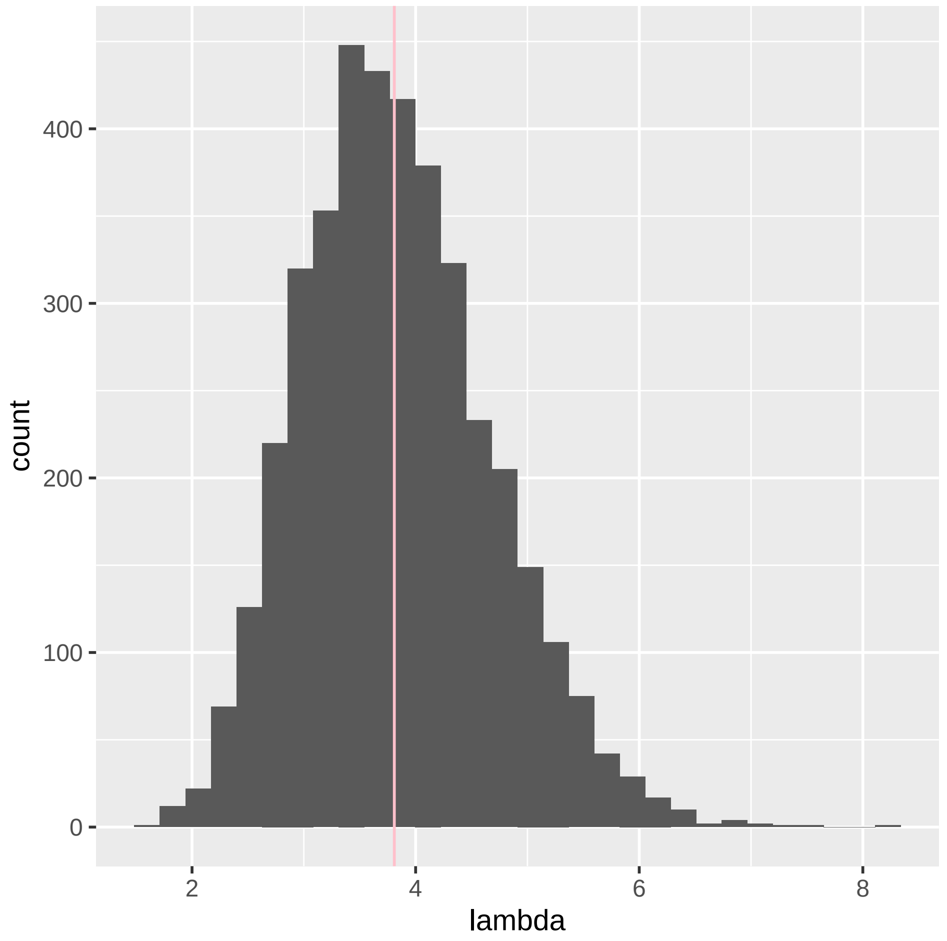
```
##
## All tests passed!
```

2c. Create a histogram of the Monte Carlo samples of λ and add a line showing the theoretical posterior of density of λ. Do the Monte Carlo samples coincide with the theoretical density?

```
soccer.df <- data.frame(
    lambda = as.numeric(lambda_samples)
    )

ggplot(soccer.df, aes(x=lambda)) + geom_histogram() + geom_vline(xintercept = post_mu, col = "pink")
```

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



Yes the Monte Carlo sample coincides with the theoretical density.

2d. Use the Monte Carlo samples from Stan to compute the mean of predicative posterior distribution to estimate the distribution of expected goals scored for next game played by the Chinese women's soccer team.

```
y <- c(4, 7, 3, 2, 3) # Number of goals in each game
n <- length(y)
```

```
#pred_mean <-
```

```
#. = ottr::check("tests/q2d.R")
```

Problem 3. Bayesian inference for the normal distribution in Stan.

Create a new Stan file and name it `IQ_model.stan`. We will make some basic modifications to the template example in the default Stan file for this problem. Consider the IQ example used from class. Scoring on IQ tests is designed to yield a N(100, 15) distribution for the general population. We observe IQ scores for a sample of n individuals from a particular town, $y_1, \dots y_n \sim N(\mu, \sigma^2)$. Our goal is to estimate the population mean in the town. Assume the $p(\mu, \sigma) = p(\mu \mid \sigma)p(\sigma)$, where $p(\mu \mid \sigma)$ is $N(\mu_0, \sigma/\sqrt{\kappa_0})$ and $p(\sigma)$ is Gamma(a, b). Before you administer the IQ test you believe the town is no different than the rest of the population, so you assume a prior mean for μ of $\mu_0 = 100$, but you aren't to sure about this a priori and so you set $\kappa_0 = 1$ (the effective number of pseudo-observations). Similarly, a priori you assume σ has a mean of 15 (to match the intended standard deviation of the IQ test) and so you decide on setting $a = 15$ and $b = 1$ (remember, the mean of a Gamma is a/b). Assume the following IQ scores are observed:

```
y <- c(70, 85, 111, 111, 115, 120, 123)
n <- length(y)
```

```
#
```

3a. Make a scatter plot of the posterior distribution of the mean, μ, and the precision, $1/\sigma^2$. Put μ on the x-axis and $1/\sigma^2$ on the y-axis. What is the posterior relationship between μ and $1/\sigma^2$? Why does this make sense? *Hint:* review the lecture notes.

```
normal_stan_model <- stan_model("IQ_model.stan")
```

```
## Warning in readLines(file, warn = TRUE): incomplete final line found on '/home/
## jovyan/winter22/homeworks/homework4/IQ_model.stan'

## Trying to compile a simple C file

## Running /opt/conda/lib/R/bin/R CMD SHLIB foo.c
## x86_64-conda-linux-gnu-cc -I"/opt/conda/lib/R/include" -DNDEBUG   -I"/opt/conda/lib/R/library/Rcpp/i
## In file included from /opt/conda/lib/R/library/RcppEigen/include/Eigen/Core:88,
##                  from /opt/conda/lib/R/library/RcppEigen/include/Eigen/Dense:1,
##                  from /opt/conda/lib/R/library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:
##                  from <command-line>:
## /opt/conda/lib/R/library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type n
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /opt/conda/lib/R/library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /opt/conda/lib/R/library/RcppEigen/include/Eigen/Dense:1,
##                  from /opt/conda/lib/R/library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:
##                  from <command-line>:
## /opt/conda/lib/R/library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or d
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/opt/conda/lib/R/etc/Makeconf:170: foo.o] Error 1
```

```
k0 <- 1
mu0 <- 100
stan_fit2 <-rstan::sampling(normal_stan_model, data = list(N=n,y=y, mu0=mu0, k0=k0, a=a, b=b))
```

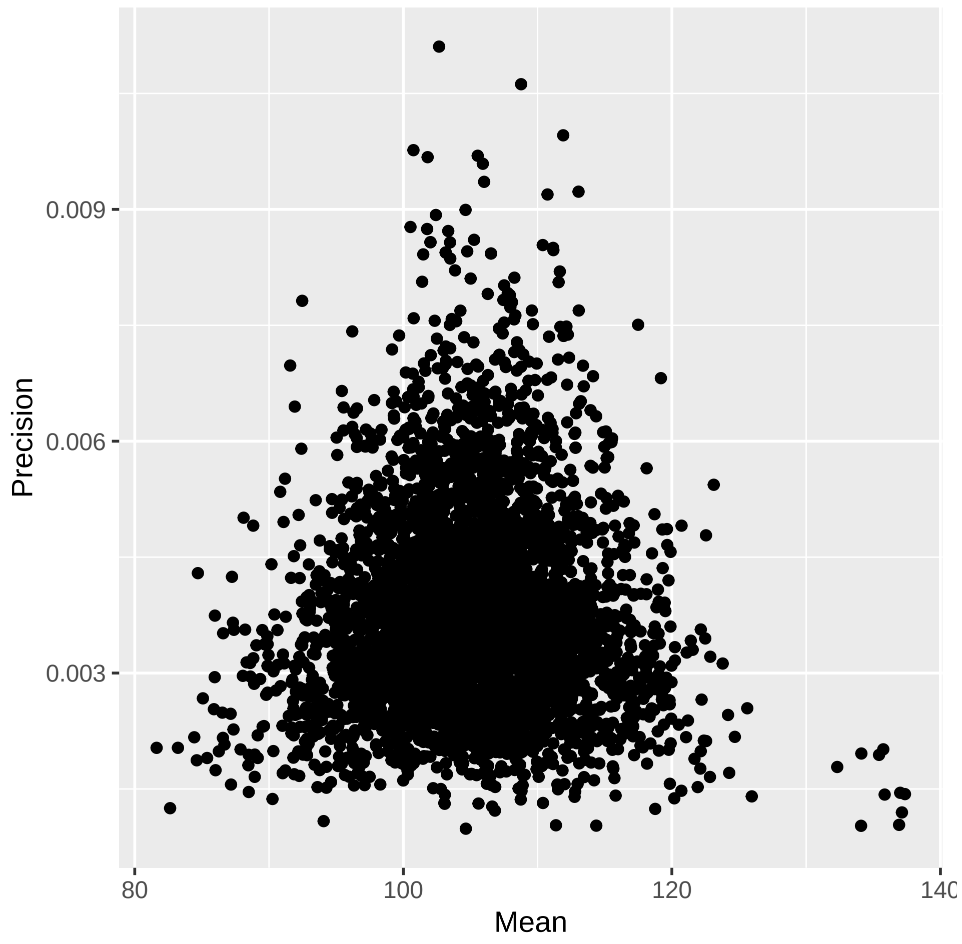
```
##
## SAMPLING FOR MODEL 'IQ_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.3 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    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: 0.01729 seconds (Warm-up)
## Chain 1:                0.011712 seconds (Sampling)
## Chain 1:                0.029002 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'IQ_model' 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: 0.017204 seconds (Warm-up)
## Chain 2:                0.009593 seconds (Sampling)
## Chain 2:                0.026797 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'IQ_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 6e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.06 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: 0.021013 seconds (Warm-up)
## Chain 3:                0.012188 seconds (Sampling)
## Chain 3:                0.033201 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'IQ_model' 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: 0.018937 seconds (Warm-up)
## Chain 4:                0.012732 seconds (Sampling)
## Chain 4:                0.031669 seconds (Total)
## Chain 4:
```

```r
samples2 <-rstan::extract(stan_fit2)

mu_samples <- samples2$mu
sigma_samples <- samples2$sigma
precision_samples <- 1/((samples2$sigma)^2)

## Make the plot
```

```r
tibble(Mean=mu_samples,Precision=1/sigma_samples^2) %>%
    ggplot()+geom_point(aes(x=Mean,y=Precision))
```



```r
. = ottr::check("tests/q3a.R")
```

```
##
## All tests passed!
```

Based on the data plot, we see that when there is a high precision, there is a low posterior variability is μ. When there is a low precision, there is high uncertainty about μ. This makes sense because σ^2 is a measure of spread.

3b. You are interested in whether the mean IQ in the town is greater than the mean IQ in the overall population. Use Stan to find the posterior probability that μ is greater than 100.

```r
y <- c(70, 85, 111, 111, 115, 120, 123)
n <- length(y)

mean(mu_samples>100)
```

```
## [1] 0.79125
```

The posterior probability of μ being greater than 100 is about 80%.

3c. You notice that two of the seven scores are significantly lower than the other five. You think that the normal distribution may not be the most appropriate model, in particular because you believe some people in this town are likely have extreme low and extreme high scores. One solution to this is to use a model that is more robust to these kinds of outliers. The Student's t distribution and the Laplace distribution are two so called "heavy-tailed distribution" which have higher probabilities of outliers (i.e. observations further from the mean). Heavy-tailed distributions are useful in modeling because they are more robust to outliers. Fit the model assuming now that the IQ scores in the town have a Laplace distribution, that is $y_1, \ldots, y_n \sim Laplace(\mu, \sigma)$. Create a copy of the previous stan file, and name it `IQ_laplace_model.stan1`. *Hint:* In the Stan file you can replace `normal` with `double_exponential` in the model section, another name for the Laplace distribution. Like the normal distribution it has two arguments, μ and σ. Keep the same prior distribution, $p(\mu, \sigma)$ as used in the normal model. Under the Laplace model, what is the posterior probability that the median IQ in the town is greater than 100? How does this compare to the probability under the normal model? Why does this make sense?

```
sm_t <- stan_model("IQ_laplace_model.stan")
```

```
## Warning in readLines(file, warn = TRUE): incomplete final line found on '/home/
## jovyan/winter22/homeworks/homework4/IQ_laplace_model.stan'

## Trying to compile a simple C file

## Running /opt/conda/lib/R/bin/R CMD SHLIB foo.c
## x86_64-conda-linux-gnu-cc -I"/opt/conda/lib/R/include" -DNDEBUG   -I"/opt/conda/lib/R/library/Rcpp/i
## In file included from /opt/conda/lib/R/library/RcppEigen/include/Eigen/Core:88,
##                  from /opt/conda/lib/R/library/RcppEigen/include/Eigen/Dense:1,
##                  from /opt/conda/lib/R/library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:
##                  from <command-line>:
## /opt/conda/lib/R/library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type n
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /opt/conda/lib/R/library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /opt/conda/lib/R/library/RcppEigen/include/Eigen/Dense:1,
##                  from /opt/conda/lib/R/library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:
##                  from <command-line>:
## /opt/conda/lib/R/library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or d
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/opt/conda/lib/R/etc/Makeconf:170: foo.o] Error 1
```

```
k0 <- 1
mu0 <- 100
stan_fit3 <-rstan::sampling(sm_t, data = list(N=n,y=y, mu0=mu0, k0=k0, a=a, b=b), refresh=0)
samples3 <-rstan::extract(stan_fit3)

mu_samples <- samples3$mu
sigma_samples <- samples3$sigma

post_prob_laplace <- mean(mu_samples>100) #HELP
post_prob_laplace
```

```
## [1] 0.92925
```

```
. = ottr::check("tests/q3c.R")
```

```
##
## All tests passed!
```

The posterior probability that the median IQ is greater than 100 is about 95%. This is much higher compared to the probability under the normal distribution. The biggest difference in the Laplace model is a conjugate model with the prior also following a normal distribution.