Stat 343 Bayes Practice with Stan

Earthquakes

This example is taken from Chihara and Hesterberg. Here's a quote from them:

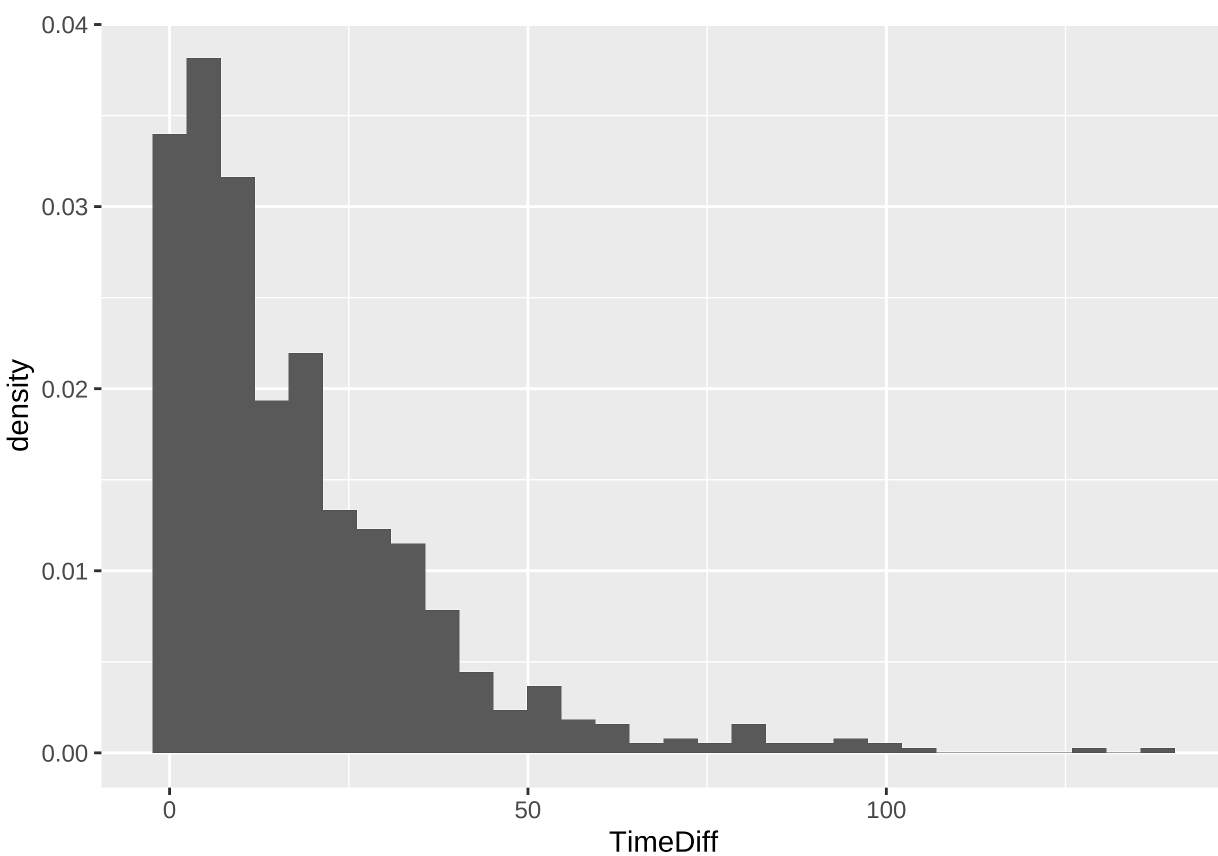
> "The Weibull distribution has been used to model the time between successive earthquakes (Hasumi et al (2009); Tiampo et al. (2008)). The data set `quakes` contains the time between earthquakes (in days) for all earthquakes of magnitude 6 or greater from 1970 through 2009 (from http://earthquake.usgs.gov/earthquakes/eqarchives/)."

The R code below reads the data in and makes an initial plot:

```
library(tidyverse)
library(rstan)
rstan_options(auto_write = TRUE)

quakes <- read_csv("http://www.evanlray.com/data/chihara_hesterberg/Quakes.csv")

ggplot(data = quakes, mapping = aes(x = TimeDiff)) +
  geom_histogram(mapping = aes(y = ..density..))
```



We have previously estimated the parameters of a Weibull model for wind speeds via Maximum Likelihood Estimation; recall that we had to do this via numerical optimization. Let's fit a Weibull distribution to the earthquake timing data, but using a Bayesian approach and MCMC this time. There is no conjugate prior for the Weibull distribution when both parameters are unknown.

So, we'll use the model

$X_i \overset{\text{iid}}{\sim} \text{Weibull}(k, \lambda)$,

where X_i is the ith observed time between consecutive earthquakes.

The Weibull distribution has two parameters, the shape parameter $k > 0$ and the scale parameter $\lambda > 0$. If $X \sim \text{Weibull}(k, \lambda)$, then it has pdf

$$f(x|k,\lambda) = \frac{kx^{k-1}}{\lambda^k} e^{-(x/\lambda)^k}$$

In R, the density function can be evaluated with the `dweibull` function, which has the following arguments:

- `x`: vector of values at which to evaluate the pdf.
- `shape`, `scale`: shape and scale parameters, the latter defaulting to 1.
- `log`: logical; if TRUE, returns the log of the pdf.

1. Set up model definition in stan

I have set up a skeleton of the stan file, included in this repository. Edit that file now to add necessary declarations and model statements for this model to the data, parameters, and model blocks. The stan function to use for the Weibull distribution is called `weibull`. Use Exponential(0.01) priors for both k and lambda. These are flat priors.

2. Perform estimation

You will need to load the rstan package, set up a list with the data for the stan model, and call `stan` to compile the model and perform sampling.

```r
fit <- stan(
  file = "earthquakes_model.stan",
  data = list(n = nrow(quakes), x = quakes$TimeDiff),
  iter = 1000,
  chains = 4,
  seed = 76732
)
```

```
##
## SAMPLING FOR MODEL 'earthquakes_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 9.8e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.98 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
```

```
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.255058 seconds (Warm-up)
## Chain 1:                0.223914 seconds (Sampling)
## Chain 1:                0.478972 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'earthquakes_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 9.4e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.94 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.258385 seconds (Warm-up)
## Chain 2:                0.260447 seconds (Sampling)
## Chain 2:                0.518832 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'earthquakes_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 9.5e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.95 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.23915 seconds (Warm-up)
## Chain 3:                0.212382 seconds (Sampling)
```

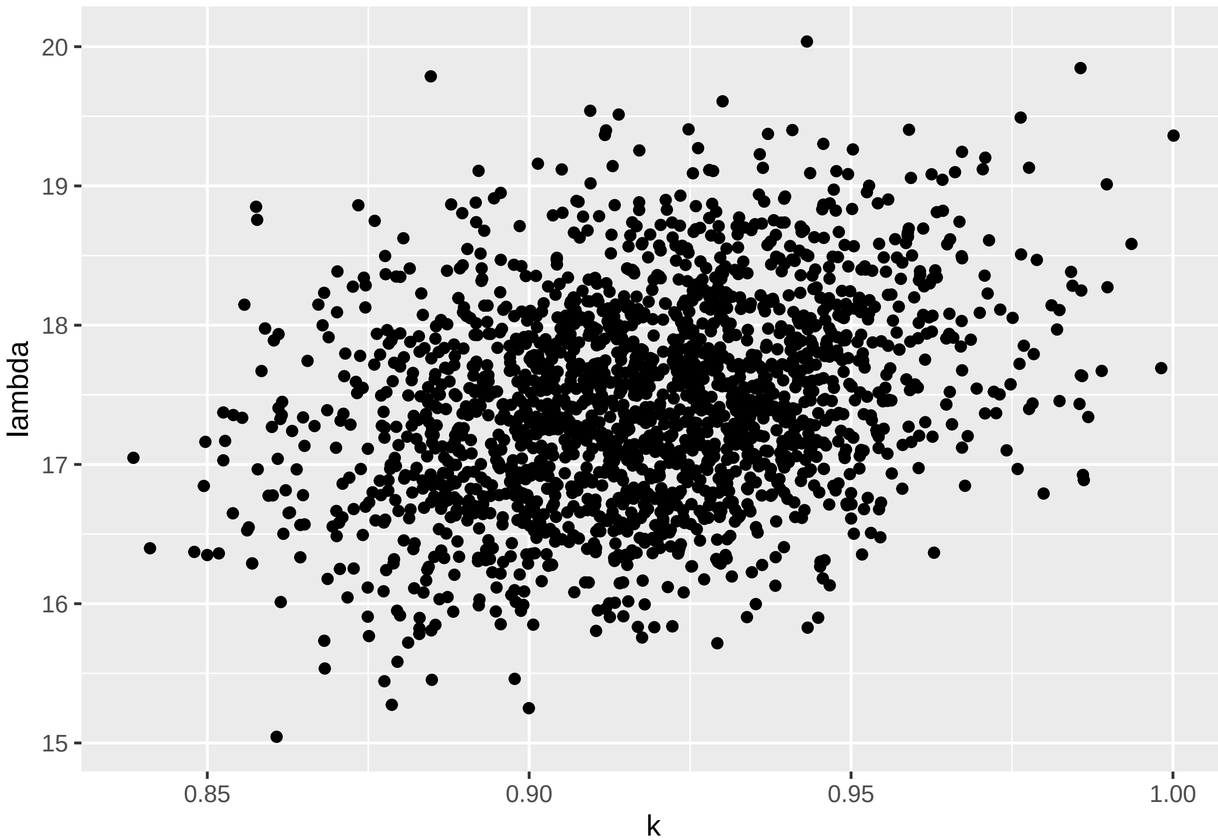
```
## Chain 3:                0.451532 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'earthquakes_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 9.4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.94 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.252741 seconds (Warm-up)
## Chain 4:                0.256392 seconds (Sampling)
## Chain 4:                0.509133 seconds (Total)
## Chain 4:
```

3. Plot results

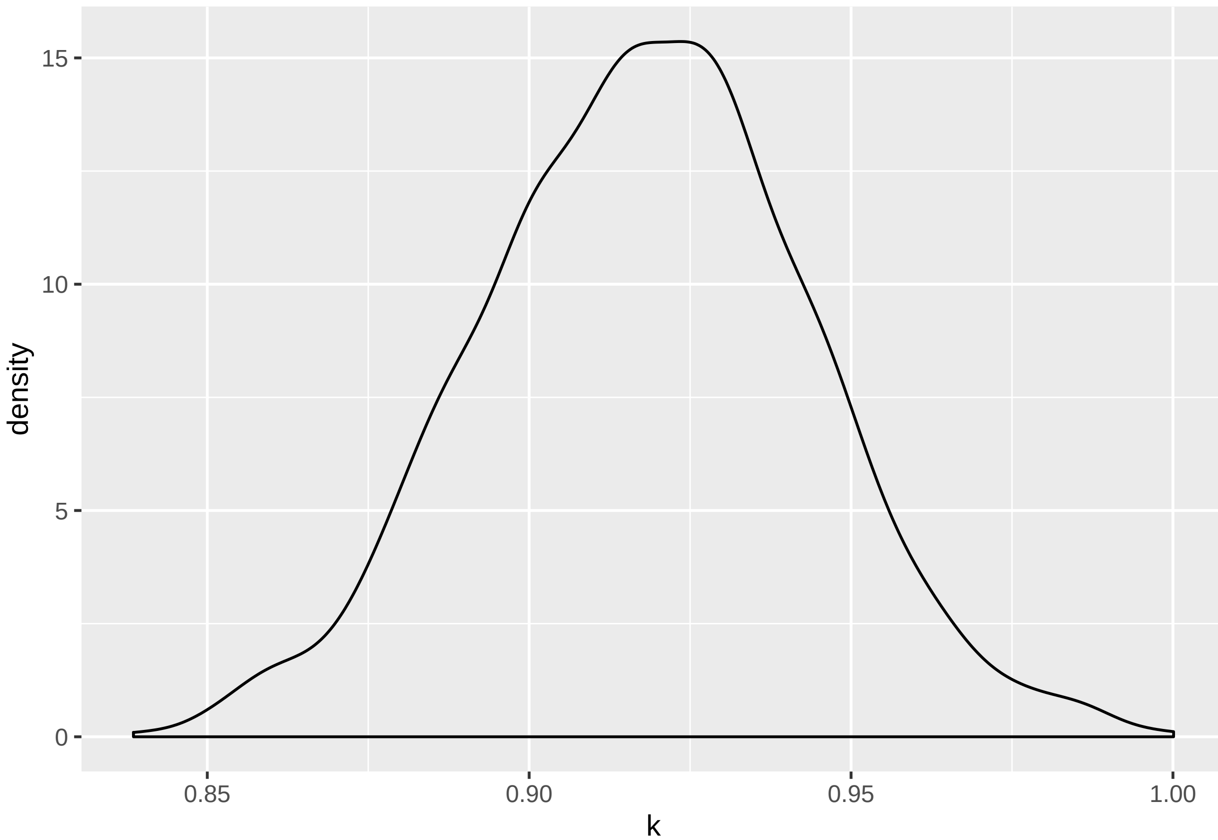
Make some exploratory plots of the results. It would be nice to have:

- a scatterplot of the posterior samples, showing both parameters for each sample from the posterior
- histograms or density plots summarizing the marginal posterior distribution for each model parameter.

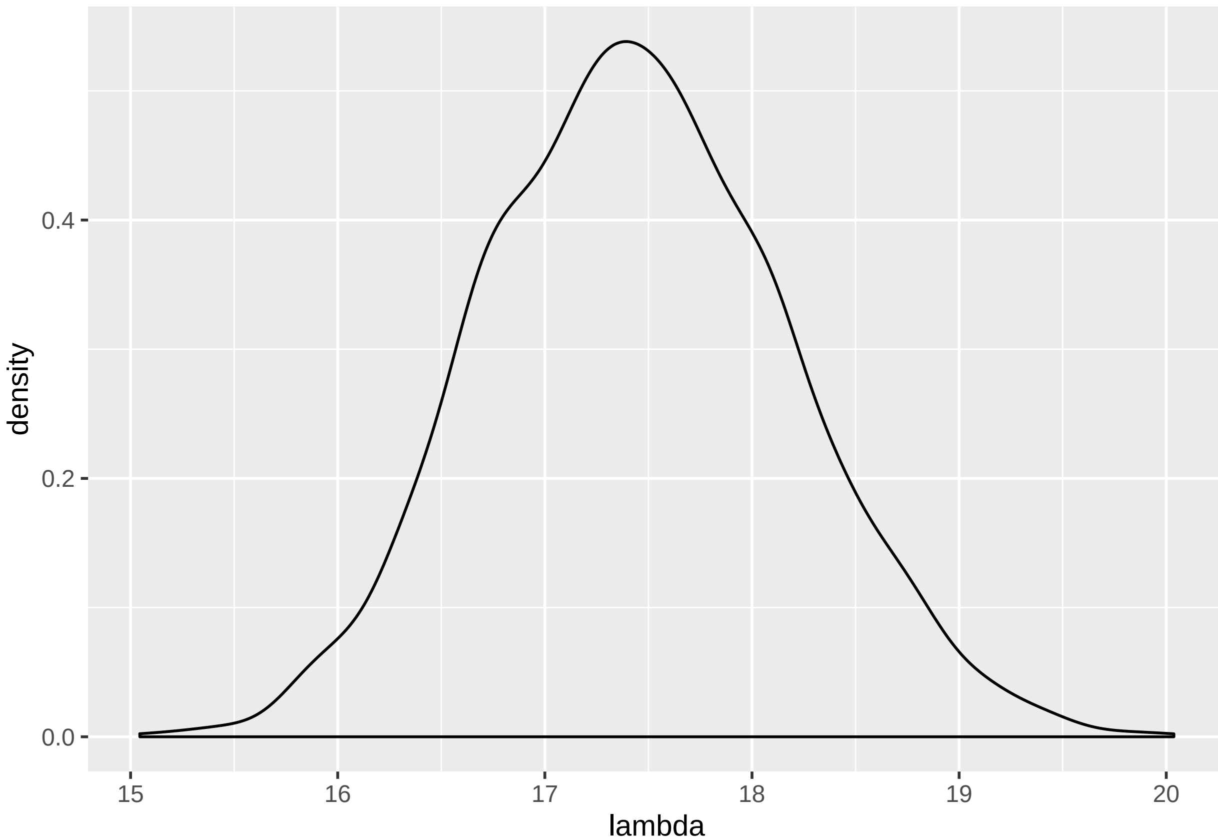
```r
posterior_samples <- as.data.frame(fit)
ggplot(data = posterior_samples, mapping = aes(x = k, y = lambda)) +
  geom_point()
```




```r
ggplot(data = posterior_samples, mapping = aes(x = k)) +
  geom_density()
```




```
ggplot(data = posterior_samples, mapping = aes(x = lambda)) +
  geom_density()
```



4. Find posterior means and credible intervals

Obtain approximate posterior means and 95% posterior credible intervals for each model parameter.

```
mean(posterior_samples$k)
```

```
## [1] 0.9180287
```

```
quantile(x = posterior_samples$k, probs = c(0.025, 0.975))
```

```
##      2.5%     97.5%
## 0.8672404 0.9672584
```

```
mean(posterior_samples$lambda)
```

```
## [1] 17.44566
```

```
quantile(x = posterior_samples$lambda, probs = c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 16.04767 18.89953
```

5. What is your effective sample size for each parameter?

```
summary(fit)
```

```
## $summary
##                 mean      se_mean         sd        2.5%         25%
## k          0.9180287 0.0007178973 0.02553343   0.8672404   0.9006312
## lambda    17.4456556 0.0216929660 0.73263035  16.0476651  16.9245164
```

```
## lp__   -3126.9348027 0.0355336607 1.02866781 -3129.6610443 -3127.3155023
##                 50%           75%         97.5%     n_eff     Rhat
## k          0.918112     0.9345939     0.9672584 1265.0090 1.000977
## lambda    17.428222    17.9373936    18.8995307 1140.5962 1.000444
## lp__   -3126.611522 -3126.1940533 -3125.9284142  838.0509 1.002658
##
## $c_summary
## , , chains = chain:1
##
##          stats
## parameter          mean         sd          2.5%           25%
##    k          0.9190099 0.02661511     0.8640947     0.9018404
##    lambda    17.4491838 0.76019424    15.9287878    17.0001058
##    lp__   -3127.0268324 1.13491153 -3130.0880010 -3127.5264659
##          stats
## parameter           50%           75%         97.5%
##    k          0.9198414     0.9360511     0.9701625
##    lambda    17.4297300    17.9151820    19.1026335
##    lp__   -3126.6699023 -3126.1696387 -3125.9278616
##
## , , chains = chain:2
##
##          stats
## parameter          mean         sd          2.5%           25%
##    k          0.9200479 0.02554909     0.8721555     0.9025818
##    lambda    17.4921587 0.67455437    16.3355528    16.9840017
##    lp__   -3126.8534694 0.94728986 -3129.3348530 -3127.1922136
##          stats
## parameter           50%           75%         97.5%
##    k          0.9181348     0.9346563     0.9739702
##    lambda    17.4882531    17.9665949    18.8134749
##    lp__   -3126.5788051 -3126.1811354 -3125.9366992
##
## , , chains = chain:3
##
##          stats
## parameter         mean         sd          2.5%           25%
##    k          0.917284 0.02502292     0.8646562     0.9006217
##    lambda    17.387816 0.74718966    16.0856964    16.8134163
##    lp__   -3126.925796 1.02020809 -3129.5916187 -3127.2766073
##          stats
## parameter           50%           75%         97.5%
##    k          0.9172099     0.9345709     0.9651703
##    lambda    17.3721475    17.8951231    18.9252980
##    lp__   -3126.6158709 -3126.1975776 -3125.9271665
##
## , , chains = chain:4
##
##          stats
## parameter         mean         sd          2.5%           25%
##    k          0.915773 0.02477443     0.8697214     0.8981679
##    lambda    17.453464 0.74391668    16.0745417    16.9198775
##    lp__   -3126.933113 0.99879213 -3129.4954447 -3127.3064666
##          stats
```

```
## parameter            50%          75%        97.5%
##    k           0.9166499    0.9326271    0.9604575
##    lambda     17.4285686   17.9432932   18.9406793
##    lp__    -3126.6140544 -3126.2438775 -3125.9220582
```

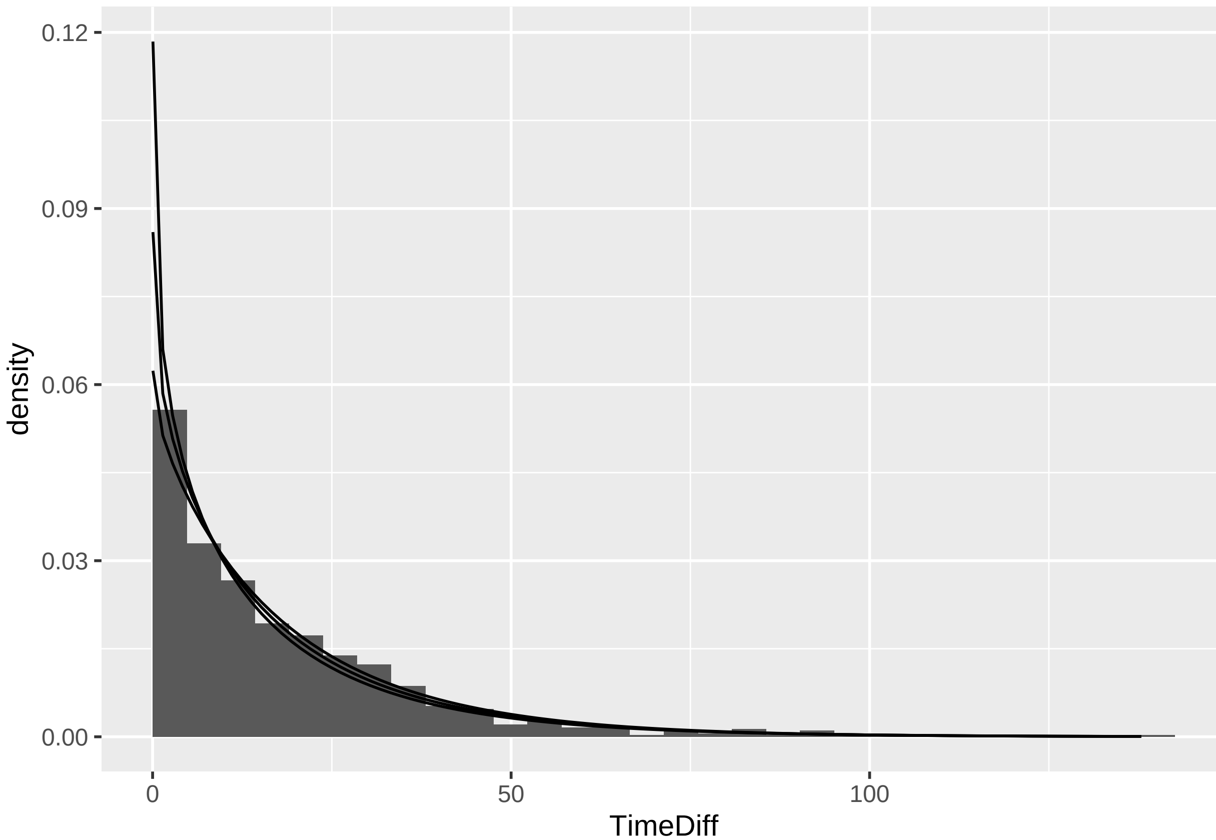
The effective sample sizes are about 1265 for k and 1141 for lambda.

Because of dependence in Markov chain sampling, we don't really have 2000 independent samples from the posterior.

6. Add three new layers to the data plot below: 1) a Weibull density using the posterior mean parameter values; 2) a Weibull density using the parameter values at the lower endpoints of the 95% credible intervals; and 3) a Weibull density using the parameter values at the upper endpoints of the 95% credible intervals.

```r
ggplot(data = quakes, mapping = aes(x = TimeDiff)) +
  geom_histogram(mapping = aes(y = ..density..), boundary = 0) +
  stat_function(
    fun = dweibull,
    args = list(shape = mean(posterior_samples$k),
                scale = mean(posterior_samples$lambda))) +
  stat_function(
    fun = dweibull,
    args = list(shape = quantile(x = posterior_samples$k, probs = 0.025),
                scale = quantile(x = posterior_samples$lambda, probs = 0.025))) +
  stat_function(
    fun = dweibull,
    args = list(shape = quantile(x = posterior_samples$k, probs = 0.975),
                scale = quantile(x = posterior_samples$lambda, probs = 0.975)))
```

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



0.12
0.09
0.06
0.03
0.00
density
0
50
100
TimeDiff