```r
library(tidyverse)
library(rstan)
library(posterior)

censored_flights <- read_csv("censored_flights2.csv")
non_censored_flights <- read_csv("non_censored_flights.csv")

mc.cores <- parallel::detectCores()
```

An airline (or any other business) may want to frame data about a customer's response to an inconvenient experience - of which a missed flight is just one example - to answer different questions: do we outright lose this customer? does this customer spend less money over time? Here we format the data to answer a related question: how long until the customer makes their next booking following a made flight versus a missed flight?

We will model this time to the next booking using time-to-event or survival models. This class of model can easily accomodate censoring - when the data recording process is cut off before the modeled event happens - and a wide range of assumptions about how the event rate varies over time. Without any domain expertise to the contrary, we stick to an exponential distribution for the time to the next booking, which implies that the event rate doesn't vary over time - a customer is equally likely to make their next booking exactly 25 hours after a flight lands as 2500 hours.

We start with rudimentary simulations of event times from an exponential distribution, and slowly add elements to the model. Our goal is to verify that our modeling method works as intended by fitting a model to the simulated data with a known rate parameter before fitting it to real data where we do not know the true rate.

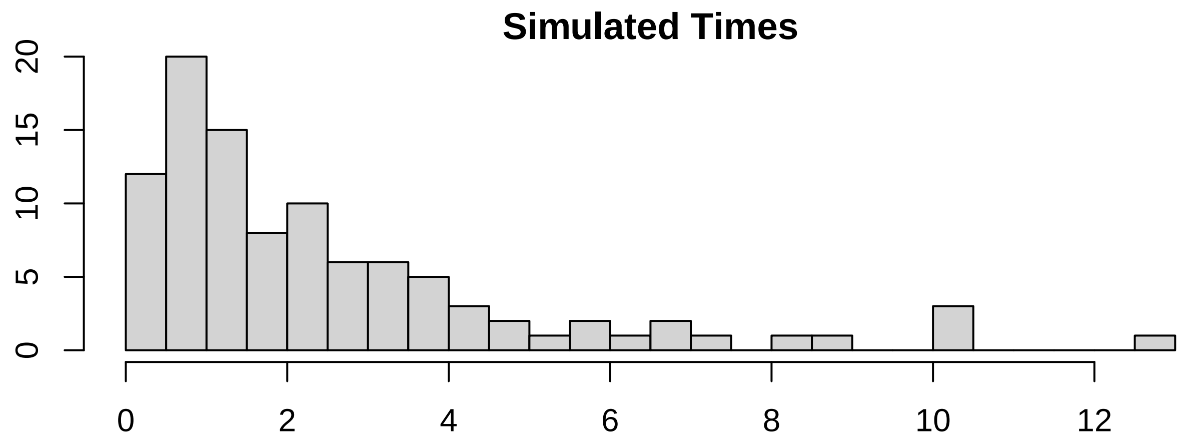
```r
exponential_rng <- stan(model_code = "
data{
  int N;
  real beta;
  }
generated quantities{
  array[N] real t = exponential_rng(rep_vector(beta,N));
}", data = list("N" = 100, "beta" = 0.5),
chains = 1, iter = 1 , warmup = 0, algorithm = "Fixed_param")
```

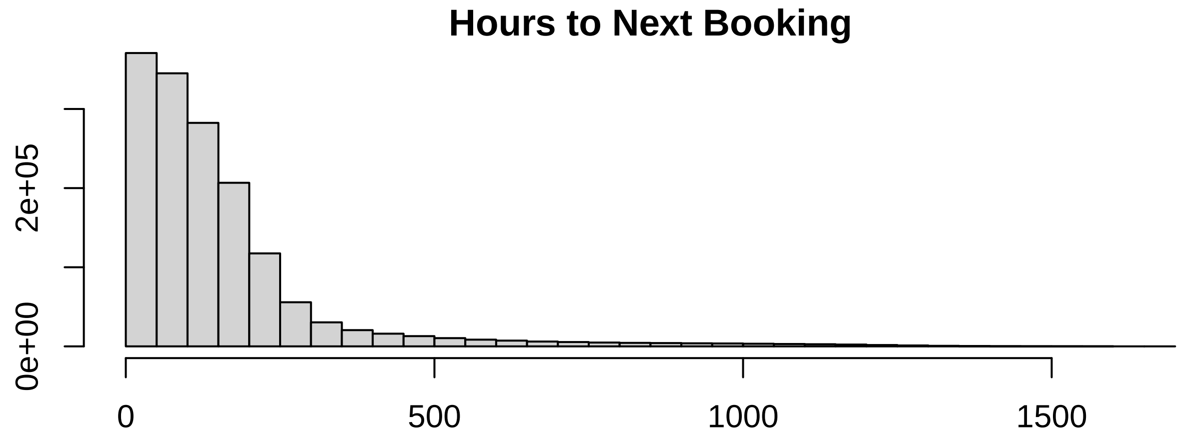
```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0 seconds (Warm-up)
## Chain 1:                0 seconds (Sampling)
## Chain 1:                0 seconds (Total)
## Chain 1:
```

```r
exponential_sim <- rstan::extract(exponential_rng)
```

Compare histograms of simulated event times compared to those observed in our data.

```r
par(mar = c(2,2,1,1))
par(mfrow=c(2,1))
hist(exponential_sim$t, main = "Simulated Times",
     xlab = "x", breaks = 30)
hist(non_censored_flights$hours_to_next_booking,
     xlab = "x", main = "Hours to Next Booking", breaks = 30)
```





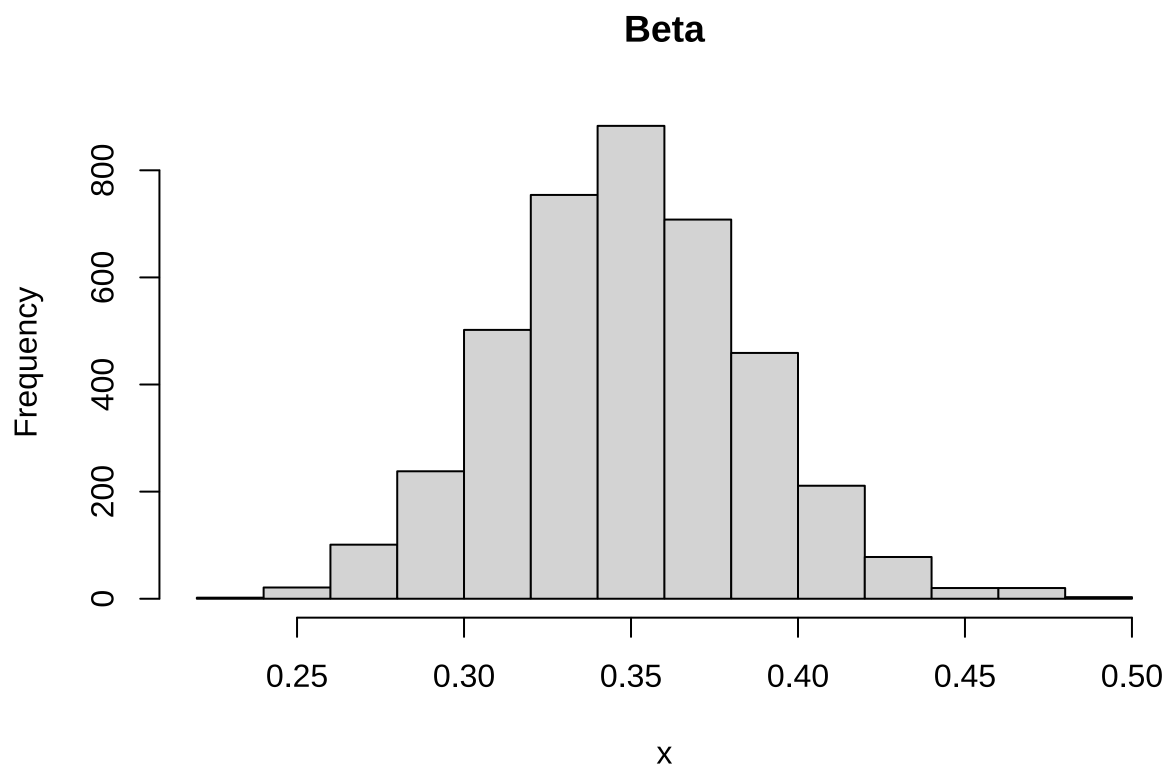
We simulated survival data with a fixed time of censoring of 2.5:

```
inv_survival_sim <- stan(model_code = "
functions{
  real inv_survival(real u, real lambda){
    return -log(u)/lambda;
  }
  real survival(real t, real lambda){
    return exp(-lambda * t);
    }
  }
data{
  int N_obs;
  real lambda;
  real t1;
  }
transformed data{
  real p1 = 1 - survival(t1, lambda);
  real p2 = 1 - p1;
  int N_rem = neg_binomial_rng(N_obs, p1/p2);
  }
generated quantities{
  array[N_obs] real obs_times;
  int cens_obs = N_rem;
  for (n in 1:N_obs){
    real u = uniform_rng(survival(t1, lambda), 1);
    obs_times[n] = inv_survival(u,lambda);
    }
  }", data = list("N_obs" = 85, "lambda" = 0.40, "t1" = 2.5), iter = 1,
warmup = 0, chains = 1, algorithm = "Fixed_param")
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0 seconds (Warm-up)
## Chain 1:                0 seconds (Sampling)
## Chain 1:                0 seconds (Total)
## Chain 1:
```

```
inverse_survival_sim_samples <- rstan::extract(inv_survival_sim)

exponential_fit <- stan(model_code = "
data{
  int N;
  int N_cens;
  vector[N] t;
  vector[N_cens] t_cens;
  }
parameters{
  real<lower=0> beta;
  }
model{
  beta ~ gamma(0.8, 0.33);
  t ~ exponential(beta);
  target += N_cens * exponential_lccdf(2.5 | beta);
  }", data = list("t" = inverse_survival_sim_samples$obs_times[1,],
                  "N" = length(inverse_survival_sim_samples$obs_times[1,]),
                  "N_cens" = as.integer(inverse_survival_sim_samples$cens_obs),
                  "t_cens" = rep(2.5, inverse_survival_sim_samples$cens_obs)),
  cores = mc.cores)
```

```
exponential_samples <- rstan::extract(exponential_fit)
hist(exponential_samples$beta, main = "Beta", xlab = "x")
```



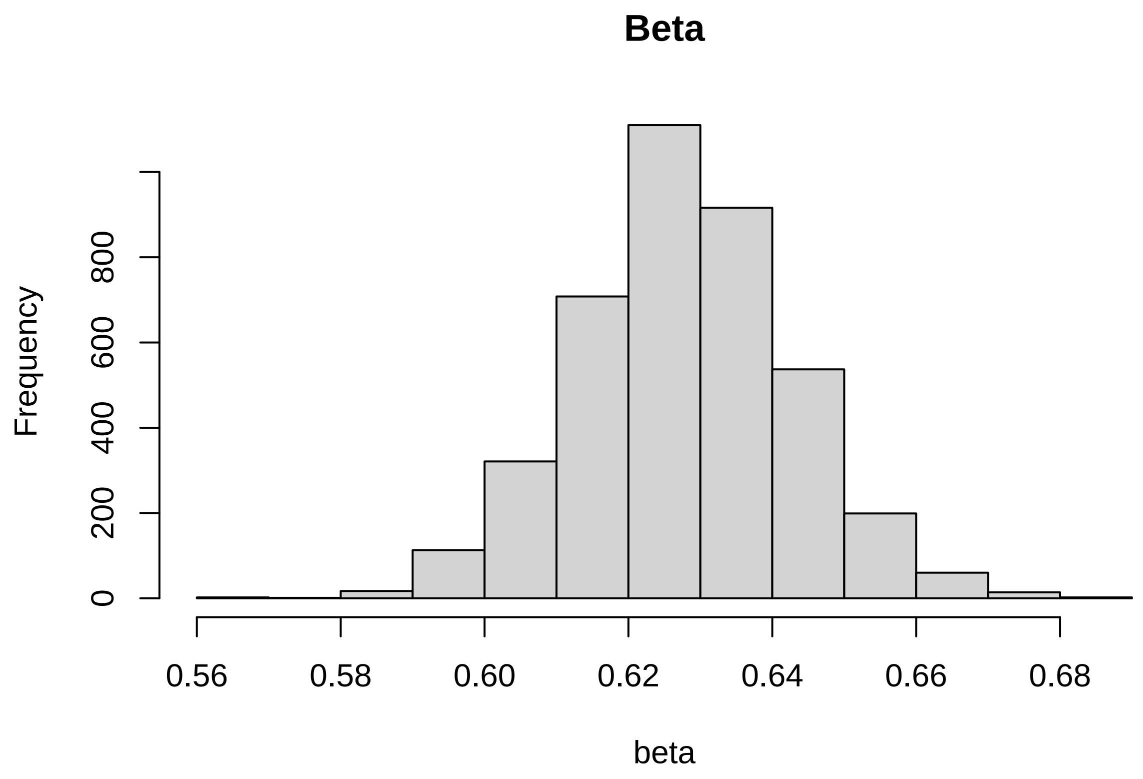
The fitted model recaptures the rate parameter with which we simulated the data neatly. Now with a variable censoring time:

```
var_inv_survival_sim <- stan(model_code =
                           "
functions{
  real inv_survival(real u, real lambda){
    return -log(u)/lambda;
  }
  real survival(real t, real lambda){
    return exp(-lambda * t);
  }
}
data{
  int N;
  real lambda;
  vector[N] noise;
}
transformed data{
  vector[N] t1;
  for (n in 1:N){
    real u = uniform_rng(0,1);
    t1[n] = inv_survival(u, lambda);
  }
  vector[N] t1_cens = 3.5 + noise;
  vector[N] cens_ind;
  int N_obs = 0;
  int N_cens = 0;
  for (n in 1:N){
    if (t1_cens[n] < t1[n]) {
      N_cens += 1;
      cens_ind[n] = 1;
    }
    else {
      N_obs += 1;
      cens_ind[n] = 0;
    }
  }
}
generated quantities{
  vector[N_obs] time_uncens;
  vector[N_cens] time_cens;
  int n_test = 1;
  int n_censo = 0;
  int n_uncenso = 0;
  while(n_test <= N){
    if (cens_ind[n_test] == 0) {
      time_uncens[n_test - n_censo] = t1[n_test];
      n_uncenso += 1;
    }
    else {
      time_cens[n_test - n_uncenso] = t1_cens[n_test];
      n_censo +=1;
    }
    n_test += 1;
  }
}
", data = list("N" = 2000, "lambda" = 0.60, "noise" = rnorm(2000,0,0.5)),
iter = 1, warmup = 0, chains = 1, algorithm = "Fixed_param")
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0 seconds (Warm-up)
## Chain 1:                0 seconds (Sampling)
## Chain 1:                0 seconds (Total)
## Chain 1:
```

```
var_inverse_survival_sim_samples <- rstan::extract(var_inv_survival_sim)


exponential_fit <- stan(model_code = "
data{
  int N;
  int N_cens;
  vector[N] t;
  vector[N_cens] t_cens;
}
parameters{
  real<lower=0> beta;
}
model{
  beta ~ gamma(0.8, 0.33);
  t ~ exponential(beta);
  for (n in 1:N_cens){
    target += exponential_lccdf(t_cens[n] | beta);
  }
}
", data = list("t" = var_inverse_survival_sim_samples$time_uncens[1,],
               "N" = length(var_inverse_survival_sim_samples$time_uncens[1,]),
               "N_cens" = as.integer(var_inverse_survival_sim_samples$n_censo),
               "t_cens" = var_inverse_survival_sim_samples$time_cens[1,]),
cores = mc.cores)

exponential_samples <- rstan::extract(exponential_fit)
```

```
hist(exponential_samples$beta, xlab = "beta",main = "Beta")
```



Again, this model recaptures the simulated rate parameter of 0.60 nicely. We move onto the simplest model for our flights data.

```
combined_data <- non_censored_flights %>% select(c(account_id,
hours_to_next_booking, missed_flight_flag)) %>% mutate(censored_id = 0) %>%
```

```r
  rename("hours" = "hours_to_next_booking") %>%
bind_rows((censored_flights %>% filter(account_id %in%
non_censored_flights$account_id) %>% mutate(hours =
as.numeric(last_flight - actual_arrival)) %>% select(c(account_id,
hours, missed_flight_flag, censored_id)))) %>% arrange(account_id,
censored_id) %>% mutate(days = hours/24) %>% select(-hours)

mean(combined_data$days)
```

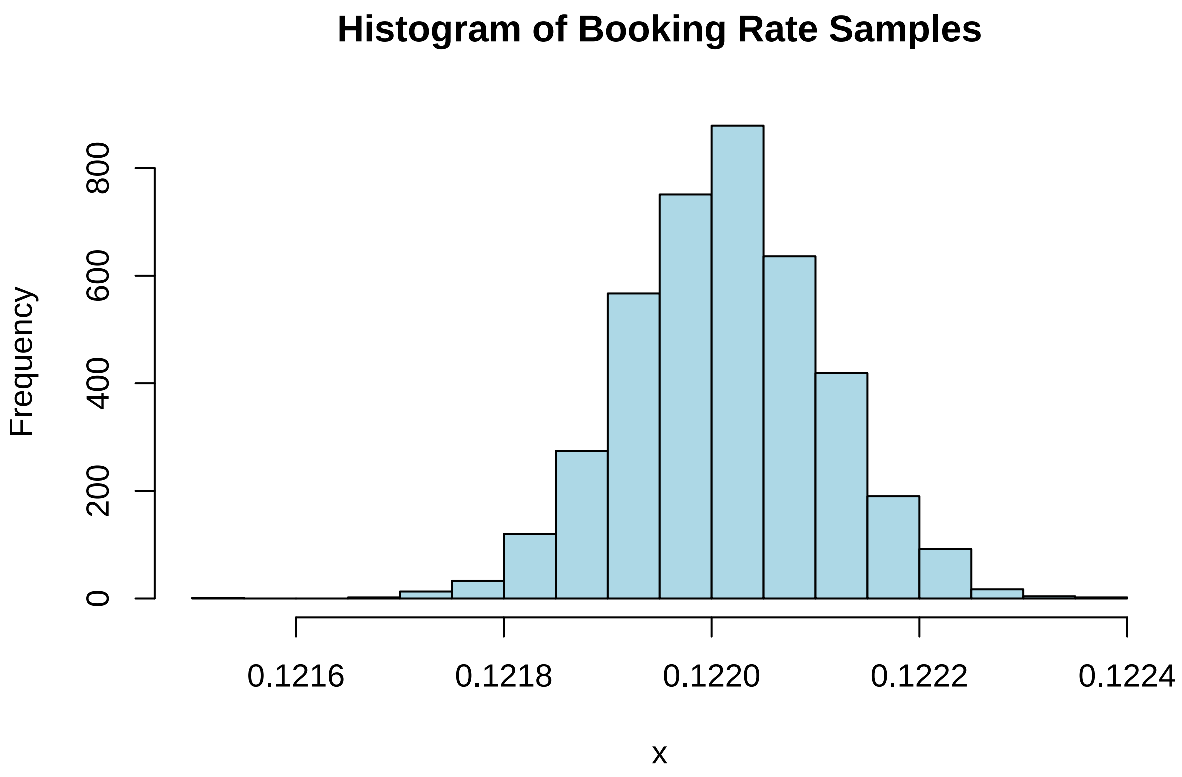
```
## [1] 7.254224
```

We used the mean here - 7.25 days to the next booking - to give us the idea of a prior for our rate parameter, which should be somewhere in the neighborhood of 1/7.25. This time between flights seems oddly low, but doesn't include any flights where a customer immediately booked a new trip for which they had missed a connection. The flights data which we call real here is actually also simulated, and this is likely an artifact of how it was created rather than an accurate representation of how customers book flights.

Now we group missed and made flights together and simply distinguish whether the next booking time was censored or not to get a parameter estimate for all data.

```r
exponential_fit_flights <- stan(model_code = "
data{
  int N;
  int N_cens;
  vector[N] t;
  vector[N_cens] t_cens;
}
parameters{
  real<lower=0> beta;
}
model{
  beta ~ gamma(0.8, 0.33);
  t ~ exponential(beta);
  for (n in 1:N_cens){
    target += exponential_lccdf(t_cens[n] | beta);
  }
}
", data = list("t" = combined_data$days[combined_data$censored_id == 0],
               "N" = as.integer(nrow(combined_data) -
                                  sum(combined_data$censored_id)),
               "N_cens" = as.integer(sum(combined_data$censored_id)),
               "t_cens" = combined_data$days[combined_data$censored_id == 1]),
cores = mc.cores)

customer_exponential_samples <- rstan::extract(exponential_fit_flights)
```

```r
hist(customer_exponential_samples$beta,
     main = "Histogram of Booking Rate Samples",xlab = "x",
     col = "lightblue")
```



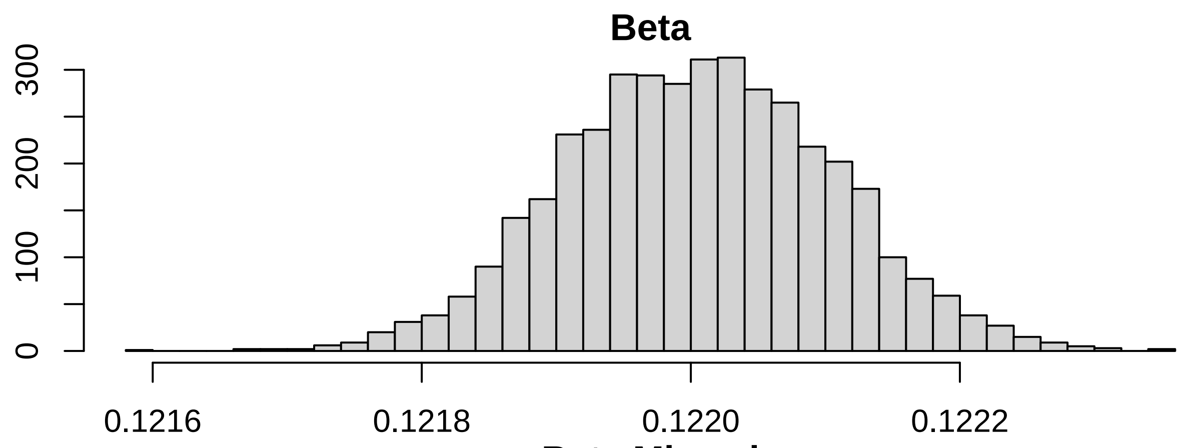
The parameter estimates concentrate heavily around 0.122, which implies a mean time of 8.2 days to the next flight. The censored events lower the rate, which shows why we have to model the data in this way rather than use an empirical mean to answer how long customers take to make another booking.

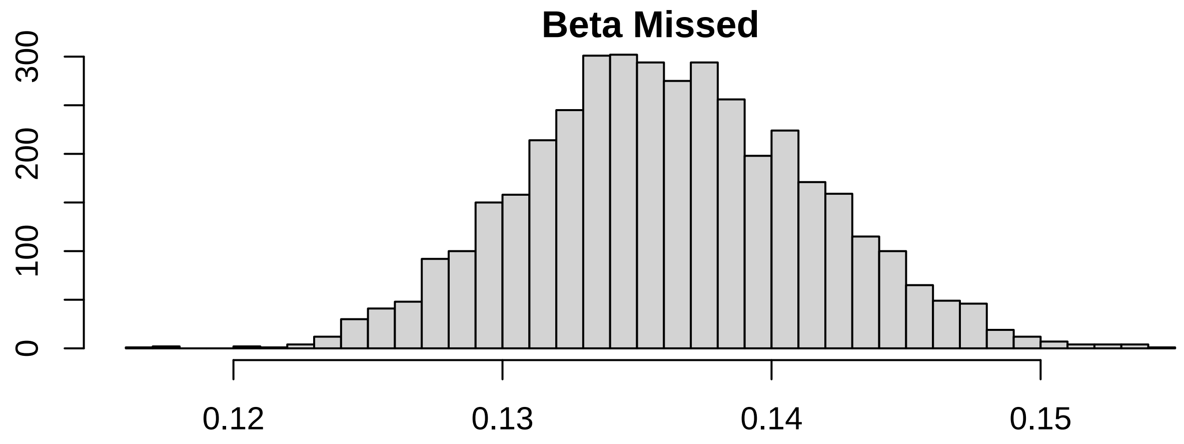
Now we do the same model, with the addition of separating made and missed flights.

```
exponential_fit_flights_missed <- stan(model_code = "
data{
  int N;
  int N_missed;
  int N_cens;
  int N_missed_cens;
  vector[N] t;
  vector[N_missed] t_missed;
  vector[N_cens] t_cens;
  vector[N_missed_cens] t_missed_cens;
  }
parameters{
  real<lower=0> beta;
  real<lower=0> beta_missed;
  }
model{
  beta ~ gamma(0.8, 0.33);
  beta_missed ~ gamma(0.8, 0.33);
  t ~ exponential(beta);
  t_missed ~ exponential(beta_missed);
  for (n in 1:N_cens){
```

```r
    target += exponential_lccdf(t_cens[n] | beta);
  }
  for (n in 1:N_missed_cens){
    target += exponential_lccdf(t_cens[n] | beta_missed);
  }
}", data = list(
  "t" = combined_data$days[combined_data$censored_id == 0 &
                             combined_data$missed_flight_flag == 0],
  "t_cens" = combined_data$days[combined_data$censored_id == 1 &
                                  combined_data$missed_flight_flag == 0],
  "t_missed" = combined_data$days[combined_data$censored_id == 0 &
                                    combined_data$missed_flight_flag == 1],
  "t_missed_cens" = combined_data$days[combined_data$censored_id == 1
                                       &combined_data$missed_flight_flag == 1],
  "N" = nrow(combined_data[combined_data$censored_id == 0 &
                             combined_data$missed_flight_flag == 0,]),
  "N_cens" = nrow(combined_data[combined_data$censored_id == 1 &
                                  combined_data$missed_flight_flag == 0,]),
  "N_missed" = nrow(combined_data[combined_data$censored_id == 0 &
                                    combined_data$missed_flight_flag == 1,]),
  "N_missed_cens" = nrow(combined_data[combined_data$censored_id == 1 &
                                      combined_data$missed_flight_flag == 1,])),
cores = mc.cores)

customer_exponential_missed_samples <-
  rstan::extract(exponential_fit_flights_missed)
```

```r
par(mar = c(2,2,1,1))
par(mfrow=c(2,1))
hist(customer_exponential_missed_samples$beta, main = "Beta",
     xlab = "x", breaks = 30)
hist(customer_exponential_missed_samples$beta_missed,
     xlab = "x", main = "Beta Missed", breaks = 30)
```





Unfortunately this analysis is confounded by the fact that missed flights happen to accounts with a higher propensity to book (not that this increased rate includes rebooking a missed flight). This can be seen by calculating the average total number of flights in this short booking window for accounts that missed a flight compared to all other accounts.

```
print(combined_data %>% group_by(account_id) %>%
        filter(sum(missed_flight_flag) == 1) %>%
        summarize(n = n()) %>% ungroup() %>% summarize(mean = mean(n)))
```

```
## # A tibble: 1 x 1
##    mean
##   <dbl>
## 1  4.18
```

Around 4.2 flights per account where at least one flight was missed.

```
print(combined_data %>% group_by(account_id) %>%
        filter(sum(missed_flight_flag) == 0) %>%
        summarize(n = n()) %>% ungroup() %>% summarize(mean = mean(n)))
```

```
## # A tibble: 1 x 1
##    mean
##   <dbl>
## 1  2.92
```

Around 2.9 flights per account where at least one flight was missed.

Simulate data where each account has its own rate parameter. Use a normal observational model for the simulated data (y).

```
test_missing <- combined_data %>% group_by(account_id) %>%
filter(sum(missed_flight_flag) >= 1)
test_missing <- test_missing %>%
left_join((test_missing %>% distinct(account_id) %>% ungroup() %>%
mutate( index = 1:n_distinct(account_id))), by = "account_id")

simu_hier <- stan(model_code = "
data{
  int <lower=1> N;
  int <lower=1> K;
  array[N] int ind_index;
  real <lower=0> sigma;
  }
transformed data{
  real mu = 4.5;
  real tau = 3.5;
  array[K] real theta = normal_rng(rep_vector(mu, K), tau);
  }
generated quantities{
  array[K] real theta_act = theta;
  array[N] real y;
  for (n in 1:N){
    y[n] =  normal_rng(theta[ind_index[n]], sigma);
    }
  } ", data = list("N" =nrow(test_missing), "K" = max(test_missing$index),
                   "ind_index" = as.integer(test_missing$index),"sigma" = 1),
      chains = 1, iter = 1, warmup = 0, algorithm ="Fixed_param")
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0 seconds (Warm-up)
## Chain 1:                0 seconds (Sampling)
## Chain 1:                0 seconds (Total)
## Chain 1:
```

```
simu_data <- rstan::extract(simu_hier)
```

Fit model to simulated data

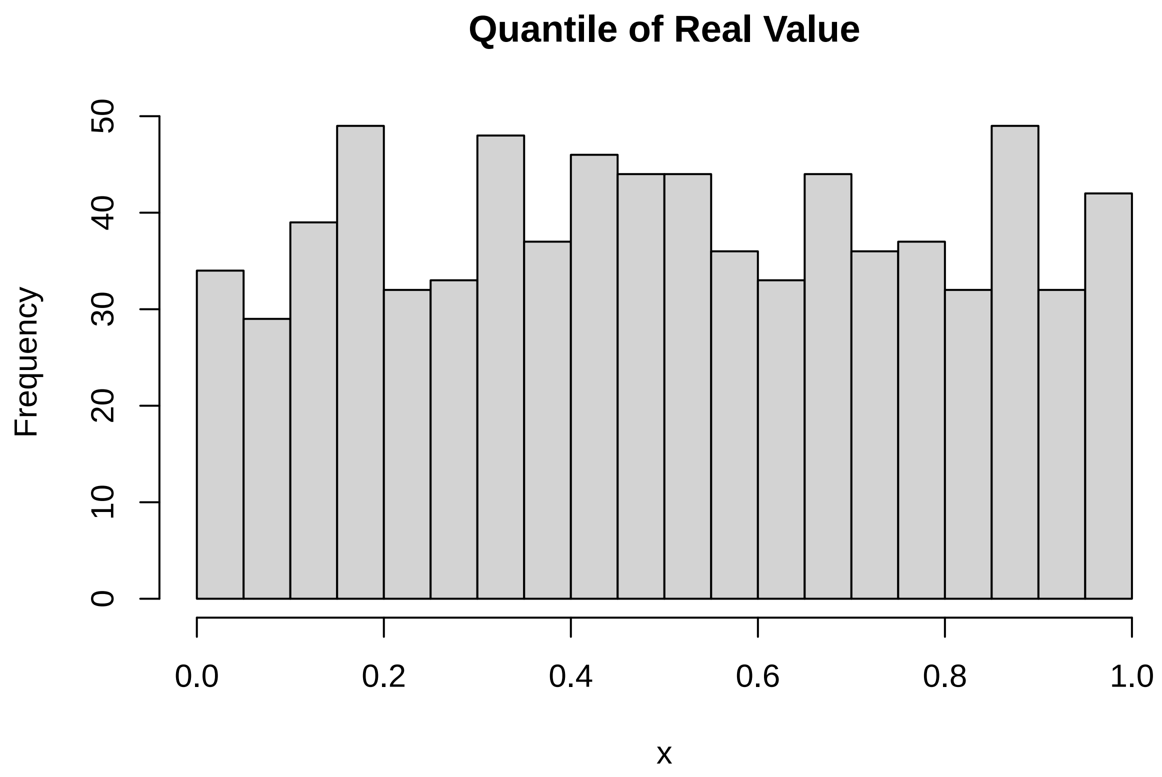
```
hier_fit <- stan(model_code = "
data{
  int N;
  int K;
  real sigma;
  array[N] int ind_index;
  array[N] real y;
}
parameters{
  real tau;
  real mu;
  vector[K] theta;
}
model{
  mu ~ normal(0,5);
  tau ~ normal(0,5);
  theta ~ normal(mu,tau);
  for (n in 1:N){
      y[n] ~ normal(theta[ind_index[n]], sigma);
  }
}
", data = list("N" = nrow(test_missing), "K" = max(test_missing$index),
               "ind_index" = as.integer(test_missing$index), "sigma" = 1,
               "y" = simu_data$y[1,]), cores = mc.cores)

hier_cp_samples <- rstan::extract(hier_fit)
```

Use a custom summary check of how well the fit is working. Average disparity is the average difference between the true value and each of 4000 parameter samples.

```
summary_check <- bind_cols("true_value" = simu_data$theta_act[1,],
"quantile_of_real_value" = sapply(seq_along(simu_data$theta_act[1,]),
function(x) ecdf(hier_cp_samples$theta[,x])  (simu_data$theta_act[1,x])),
"average_disparity" = sapply(seq_along(simu_data$theta_act),  function(x)
  (sum(hier_cp_samples$theta[,x] - simu_data$theta_act[1,x])/4000)),
t(apply(hier_cp_samples$theta, 2,
        function(x) quantile(x, probs = c(seq(0.1,0.9,0.1))))))
```

One of the parameters with this seed lies above all parameter samples (quantile_of_real_value equal to 1.00) but the fit works as expected

```
hist(summary_check$quantile_of_real_value,
     main = "Quantile of Real Value", breaks = 30, xlab = "x")
```



```
mean(abs(summary_check$average_disparity))/mean(abs(summary_check$true_value))
```

```
## [1] 0.08766774
```

The average disparity is around 0.1 of the true value, which is an acceptable amount of error here.

Now simulate survival times with an exponential distribution and a rate that more closely mirrors the booking rate in the flights data.

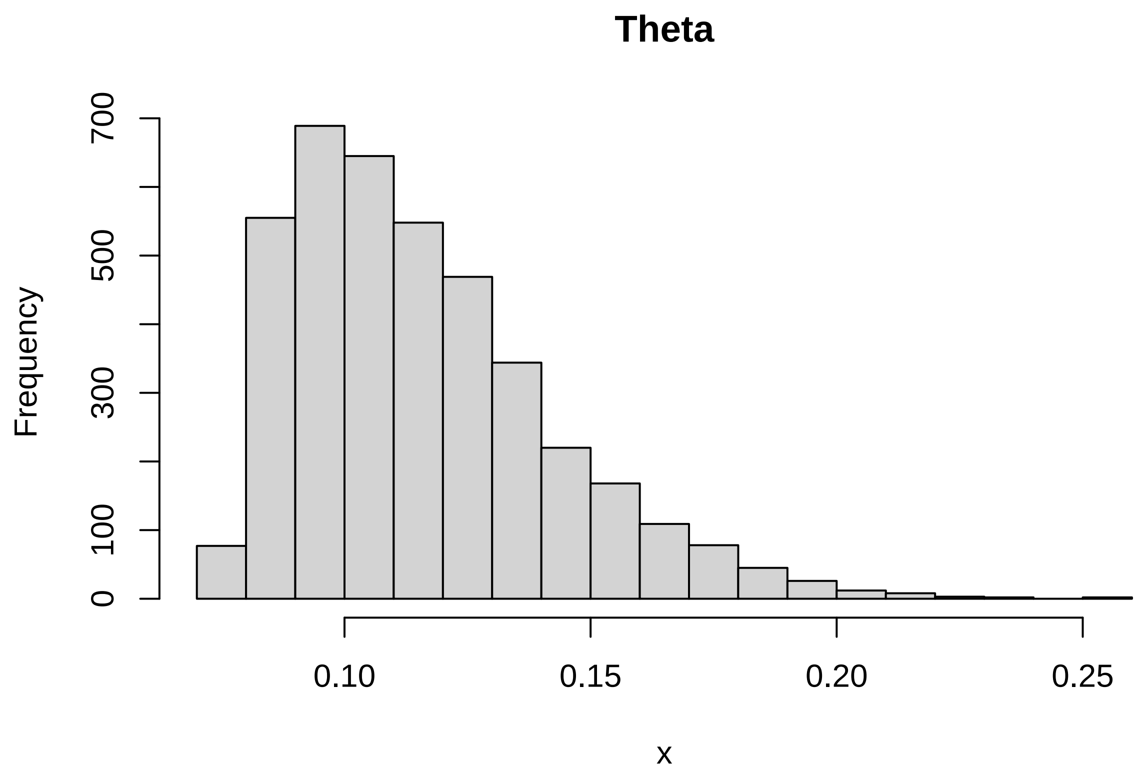
```
simu_survival_hier <- stan(model_code = "
data{
  int <lower=1> N;
  int <lower=1> K;
  array[N] int ind_index;
  }
transformed data{
  real<lower=0> mu = 0.12;
  real tau = 0.03;
  array[K] real <lower=0> theta = normal_rng(rep_vector(mu, K), tau);
  }
generated quantities{
  array[K] real theta_act = theta;
  array[N] real t;
  for (n in 1:N){
    t[n] = exponential_rng(theta[ind_index[n]]);
    }
  } ", data = list("N" = nrow(test_missing), "K" = max(test_missing$index),
                   "ind_index" = as.integer(test_missing$index)),
chains = 1, iter = 1, warmup = 0, algorithm = "Fixed_param", seed = 24)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0 seconds (Warm-up)
## Chain 1:                0 seconds (Sampling)
## Chain 1:                0 seconds (Total)
## Chain 1:
```

```
survival_hier_data <- rstan::extract(simu_survival_hier)
```

The prior on tau strongly implies we expect that the true value should lie between 0.01 and 0.05. The prior on mu implies we expect that the true value should lie between 0.05 and 0.15. Use a non-centered parameterization for the theta parameters, which works far more reliably when the amount of data for each parameter is small (1 to 5 data points each in this case)

```
survival_hier_fit <- stan(model_code = "
data{
  int <lower=1> N;
  int <lower=1> K;
  array[N] int ind_index;
  array[N] real t;
}
parameters{
  real<lower=0> mu;
  real<lower=0> tau;
  vector <lower=0> [K] eta;
}
transformed parameters{
  vector <lower=0> [K] theta = mu + eta*tau;
}
model{
  mu ~ inv_gamma(18.5,1.5);
  eta ~ normal(0,1);
  tau ~ gamma(8.9, 346);
  t ~ exponential(theta[ind_index]);
}
", data = list("N" = nrow(test_missing), "K" = max(test_missing$index),
               "ind_index" = as.integer(test_missing$index),
               "t" = survival_hier_data$t[1,]), cores = mc.cores)


survival_hier_samples <- rstan::extract(survival_hier_fit)
hist(survival_hier_samples$theta[,1], main = "Theta", xlab = "x")
```

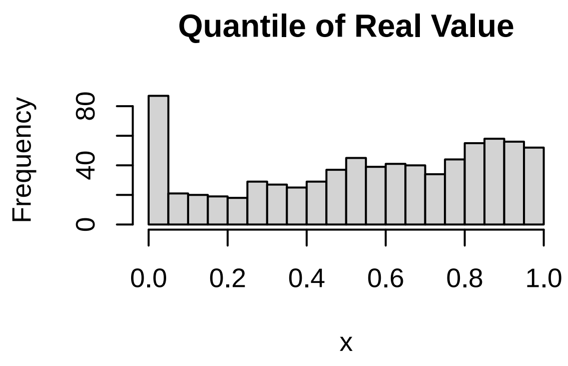


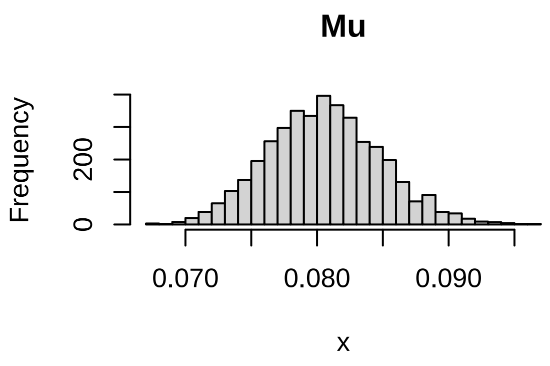
```
survival_summary_check <-
  bind_cols("true_value" = survival_hier_data$theta_act[1,],
"bulk_ESS" =  sapply(seq_along(survival_hier_samples$theta[1,]),
                                 function(x)
ess_bulk(survival_hier_samples$theta[,x])),
"quantile_of_real_value" =
  sapply(seq_along(survival_hier_data$theta_act[1,]), function(x)
            ecdf(survival_hier_samples$theta[,x])
                 (survival_hier_data$theta_act[1,x])),
"average_disparity" = sapply(seq_along(survival_hier_data$theta_act),
    function(x) (sum(survival_hier_samples$theta[,x] -
                         survival_hier_data$theta_act[1,x])/4000)),
    t(apply(survival_hier_samples$theta, 2, function(x)
      quantile(x, probs = c(seq(0.1,0.9,0.1))))))
```

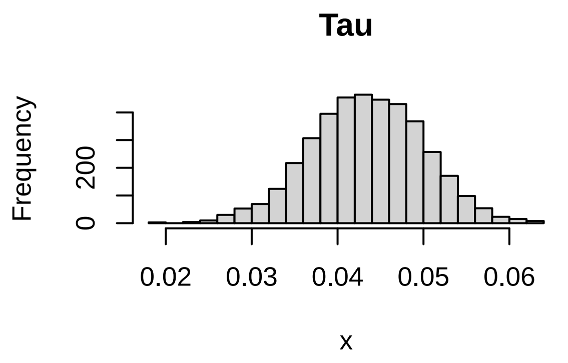
Unfortunately, no matter how the priors or the simulated data are tweaked, the fitted samples for mu are consistently lower than the data value, and the fitted samples for tau are consistently much higher than the data value. With a small number of parameters with a small number of data points each, the population mean is poorly informed and the model compensates by allowing too much flexibility between each theta parameter. True theta values pile up at the extremes of the quantile range and many fall below all sampled values:

```
par(mfrow=c(2,2))
hist(survival_summary_check$quantile_of_real_value,
     main = "Quantile of Real Value", xlab = "x", breaks = 30)
```

```r
hist(survival_hier_samples$mu, breaks = 30, main = "Mu", xlab = "x")
hist(survival_hier_samples$tau, breaks = 30, main = "Tau", xlab = "x")
```







The population mean should be better informed with more data, so despite this setback we move on to include accounts where no flight was missed.

```r
combined_data <- combined_data %>%
  left_join((combined_data %>% distinct(account_id) %>%
               ungroup() %>% mutate(index = 1:n_distinct(account_id))),
                                                 by = "account_id")
```

Because the model with a parameter for each account would be too big to fit in the memory of a standard computer, we take a selection of account ids starting with all that missed at least one flight plus a random subset of all other accounts.

```r
combined_data2 <- combined_data %>% group_by(account_id) %>%
filter(sum(missed_flight_flag) >= 1)
combined_data3 <- combined_data %>% anti_join(combined_data2)
```

```
## Joining with ‘by = join_by(account_id, missed_flight_flag, censored_id, days,
## index)‘
```

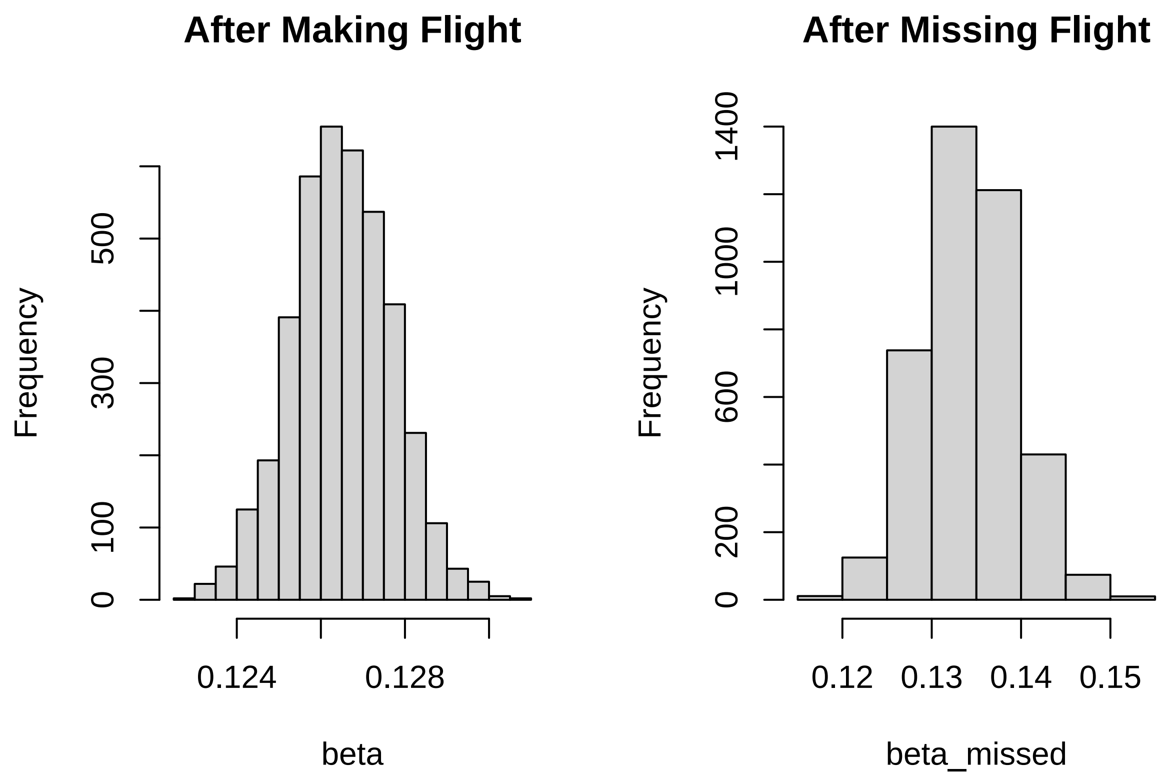
```r
thinning_indices <- sample.int(591654,5917)
combined_data_thinned <- combined_data2 %>%
bind_rows((combined_data3 %>% filter(index %in% thinning_indices)))
combined_data_thinned <- combined_data_thinned %>% select(-index) %>%
left_join((combined_data_thinned %>% distinct(account_id) %>%
ungroup() %>% mutate( index = 1:n_distinct(account_id))), by =
"account_id")

customer_missed_cens_data <-
  list("t" = combined_data_thinned$days[combined_data_thinned$censored_id == 0
        & combined_data_thinned$missed_flight_flag == 0],
"t_cens" = combined_data_thinned$days[combined_data_thinned$censored_id == 1 &
combined_data_thinned$missed_flight_flag == 0],
"t_missed" = combined_data_thinned$days[combined_data_thinned$censored_id == 0
& combined_data_thinned$missed_flight_flag == 1],
"t_missed_cens" =
  combined_data_thinned$days[combined_data_thinned$censored_id == 1 &
                             combined_data_thinned$missed_flight_flag == 1],
"N" = nrow(combined_data_thinned[combined_data_thinned$censored_id == 0 &
  combined_data_thinned$missed_flight_flag == 0,]),
"N_cens" = nrow(combined_data_thinned[combined_data_thinned$censored_id == 1 &
combined_data_thinned$missed_flight_flag == 0,]),
"N_missed" = nrow(combined_data_thinned[combined_data_thinned$censored_id == 0 &
combined_data_thinned$missed_flight_flag == 1,]),
"N_missed_cens" = nrow(combined_data_thinned[combined_data_thinned$censored_id
== 1 & combined_data_thinned$missed_flight_flag == 1,]),
"ind_index" =
  as.integer(combined_data_thinned$index[combined_data_thinned$censored_id == 0
& combined_data_thinned$missed_flight_flag == 0]),
"ind_index_cens" =
  as.integer(combined_data_thinned$index
             [combined_data_thinned$censored_id == 1
              & combined_data_thinned$missed_flight_flag == 0]),
"ind_index_missed" =
  as.integer(combined_data_thinned$index
             [combined_data_thinned$censored_id == 0
& combined_data_thinned$missed_flight_flag == 1]),
"ind_index_missed_cens" =
  as.integer(combined_data_thinned$index
             [combined_data_thinned$censored_id == 1
& combined_data_thinned$missed_flight_flag == 1]),
"K" = as.integer(max(combined_data_thinned$index)),
"K2" =
  as.integer(max(combined_data_thinned$index
                 [combined_data_thinned$missed_flight_flag == 1])))

exponential_hier_fit_flights_missed <- stan(model_code = "
  data{
    int K;
    int K2;
    int N;
    int N_missed;
    int N_cens;
    int N_missed_cens;
    array[N] int ind_index;
    array[N_missed] int ind_index_missed;
    array[N_cens] int ind_index_cens;
    array[N_missed_cens] int ind_index_missed_cens;
    vector[N] t;
    vector[N_missed] t_missed;
    vector[N_cens] t_cens;
    vector[N_missed_cens] t_missed_cens;
  }
  parameters{
    real beta;
    real beta_missed;
    real<lower=0> tau;
    vector[K] eta;
  }
  transformed parameters{
    vector [K2] eta2 = eta[1:K2];
    vector [K] theta_trans = exp(beta + tau*eta);
    vector [K2] theta_trans_missed = exp(beta_missed + tau*eta2);
  }
  model{
    tau ~ normal(0.4,0.1);
    eta ~ normal(0,1);
    beta ~ normal(-2.1,0.1);
    beta_missed ~ normal(-2.1,0.1);
    for (n in 1:N){
      t[n] ~ exponential(theta_trans[ind_index[n]]);
    }
    for (n in 1:N_missed){
      t_missed[n] ~
      exponential(theta_trans_missed[ind_index_missed[n]]);
    }
    for (n in 1:N_cens){
      target += exponential_lccdf(t_cens[n] |
      theta_trans[ind_index_cens[n]]);
    }
    for (n in 1:N_missed_cens){
      target += exponential_lccdf(t_missed_cens[n] |
      theta_trans_missed[ind_index_missed_cens[n]]);
    }
  } ", data = customer_missed_cens_data, cores = mc.cores, seed = 24)

exponential_hier_fit_flights_missed_ncp_samples <-
rstan::extract(exponential_hier_fit_flights_missed)
```

Our parameters of interest are primarily beta_missed and beta, the intercepts for the rate to the next booking for when a flight is missed compared to all other flights. We are less interested in the parameters for individual accounts, especially with so few flights.

In the real world, we might expect that customers would be dissatisfied by the experience of missing a flight (or other inconveniences they might face in their trips, none of which we can intuit from this dataset), but since this is not real data, it was probably not simulated with that structure in mind, and we don't expect much difference between beta and beta_missed. This code could easily be adapted to real data, where we would expect some difference.

Fortunately, unlike the previous simulated example, our mean parameters fall in a much more plausible range, similar to the rate of ~.122 we found in the basic model.

```
par(mfrow = c(1,2))
hist(exp(exponential_hier_fit_flights_missed_ncp_samples$beta),
     main = "After Making Flight", xlab = "beta")
hist(exp(exponential_hier_fit_flights_missed_ncp_samples$beta_missed),
     main = "After Missing Flight", xlab = "beta_missed")
```



Both rates are higher than .122, which may come down to the randomly chosen parameter indices.

1 divided by the rate parameter of the exponential distribution gives the mean time to event with that rate parameter.

```
mean(1/exp(exponential_hier_fit_flights_missed_ncp_samples$beta))
```

```
## [1] 7.905574
```

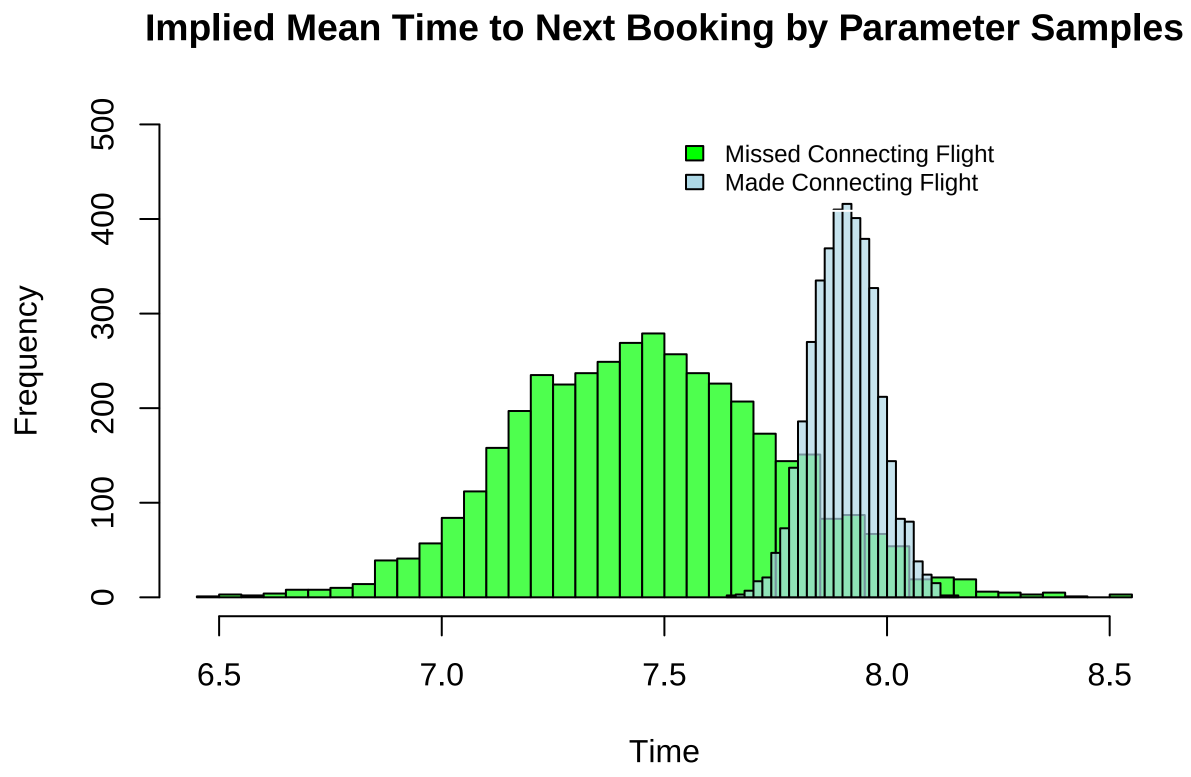
```
mean(1/exp(exponential_hier_fit_flights_missed_ncp_samples$beta_missed))
```

```
## [1] 7.464568
```

So according to our model, we expect customers to book flights more quickly following a missed flight, even once we account for individual accounts and censoring, contrary to our expectation.

Overlaying the samples on the same plot shows how much more uncertainty there is in the beta_missed parameter than in the beta parameter - the breadth along the x-axis of the histogram of posterior samples is much wider - as expected with a far larger dataset of made flights than missed flights.

```r
hist(1/exp(exponential_hier_fit_flights_missed_ncp_samples$beta_missed),
     breaks = 30, ylim = c(0,500), col = alpha("green", 0.7),
     main = "Implied Mean Time to Next Booking by Parameter Samples",
     xlab = "Time")
hist(1/exp(exponential_hier_fit_flights_missed_ncp_samples$beta),
     breaks = 30,
     main = "Implied Mean Time to Next Booking by Parameter Samples",
     col = alpha("lightblue", 0.7), add = T)
legend(7.5,500, legend =
         c("Missed Connecting Flight", "Made Connecting Flight"), fill =
         c("green", "lightblue"), box.col = "white", cex = 0.75)
```


Implied Mean Time to Next Booking by Parameter Samples
🟩 Missed Connecting Flight
🟦 Made Connecting Flight
Frequency
Time