Bayesian regression

Sam Bashevkin[*]

2/28/2022

Background

To learn about Bayesian analyses, I highly recommend the book Statistical Rethinking by Richard McElreath. The full course for which this book was developed is recorded and available online, along with slides, code, examples, and translations of the book's code to other packages and programming languages (such as brms+tidyverse, Python, Julia): https://xcelab.net/rm/statistical-rethinking/.

Other books and resources that may be useful (and are freely available online):

1. Regression and other stories
2. Bayesian Data Analysis
3. Stan documentation
4. Stan forum

Components of a Bayesian analysis

1. Data D: observations of the process to be modeled
2. Parameters θ: unknown inputs to be estimated in a model
3. Prior $p(\theta)$: probability for each possible value of each parameter
4. Likelihood $p(D|\theta)$: Probability of your data given the estimated parameters
5. Posterior $p(\theta|D)$: probability of the estimated parameters given your data

$p(\theta|D) = p(D|\theta)p(\theta)$

Prior choices

Read the Stan prior choice recommendations for the latest recommendations on priors.

Here, you will see me using weakly informative priors with a mix of manually setting priors and using the `brms` default priors. Personally, I tend to use $Normal(0, 10)$ priors for intercepts, $Normal(0, 5)$ priors for slopes, and $Cauchy(0, 5)$ priors for variance terms. With more complex parameters (e.g., the smoothing parameters from a generalized additive model) I use the `brms` default priors.

Load data

Packages

```
library(zooper) # https://github.com/InteragencyEcologicalProgram/zooper
library(dplyr)
library(tidyr)
library(ggplot2)
library(patchwork)
```

[*]Delta Science Program, Delta Stewardship Council, sam.bashevkin@deltacouncil.ca.gov

```
library(lubridate)
library(brms)
library(tidybayes)
```

```
BL<-Zoopsynther(Data_type = "Taxa", Size_class="Meso", Taxa="Bosmina longirostris")%>%
  # Remove non-fixed stations
  filter(!Station%in%c("NZEZ2", "NZEZ6", "NZEZ2SJR", "NZEZ6SJR") &
           # Just use post-POD years to reduce computational time
           Year>=2002)%>%
  # Add survey name to station variable
  mutate(Station=paste(Source, Station),
         # log-transform salinity
         SalSurf_l = log(SalSurf),
         # Create variable for day of year
         DOY = yday(Date))%>%
  # drop rows with NAs in the key columns
  drop_na(Date, SalSurf, CPUE)%>%
  # Standardize covariates
  mutate(across(c(SalSurf_l, DOY), list(s=~(.-mean(., na.rm=T))/sd(., na.rm=T))))%>%
  select(Source, SampleID, CPUE, Date, Station,
         SalSurf, SalSurf_l, SalSurf_l_s, DOY, DOY_s)
```

```
## [1] "No orphaned taxa here!"
## [2] "NOTE: Do not use this data to make additional higher-level taxonomic summaries or any other ope
```

```
str(BL)
```

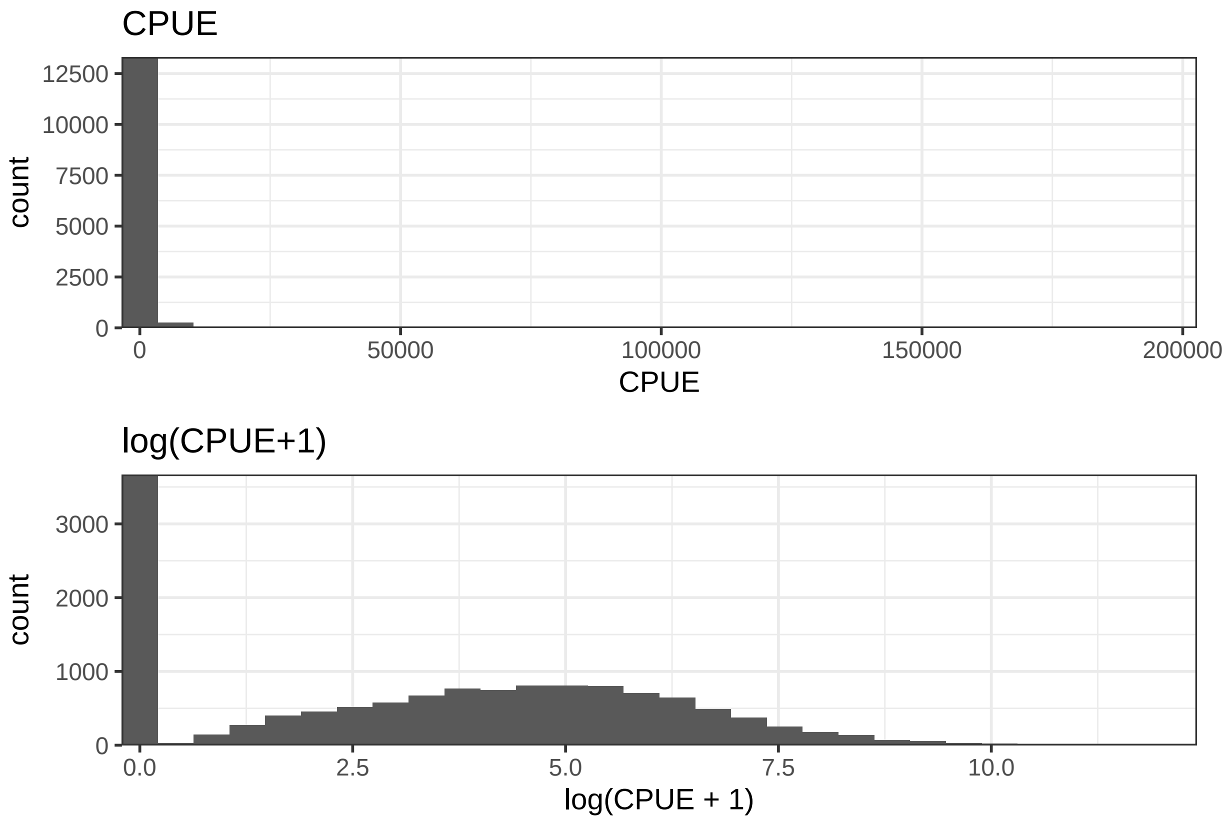
```
## tibble [13,692 x 10] (S3: tbl_df/tbl/data.frame)
##  $ Source     : chr [1:13692] "EMP" "EMP" "EMP" "EMP" ...
##  $ SampleID   : chr [1:13692] "EMP NZ086 2002-01-07" "EMP NZ092 2002-01-07" "EMP NZD16 2002-01-07" "
##  $ CPUE       : num [1:13692] 78.1 186.1 56.4 4.7 65.2 ...
##  $ Date       : POSIXct[1:13692], format: "2002-01-07" "2002-01-07" ...
##  $ Station    : chr [1:13692] "EMP NZ086" "EMP NZ092" "EMP NZD16" "EMP NZD28" ...
##  $ SalSurf    : num [1:13692] 0.0994 0.154 0.0961 0.1677 0.1361 ...
##  $ SalSurf_l  : num [1:13692] -2.31 -1.87 -2.34 -1.79 -1.99 ...
##  $ SalSurf_l_s: num [1:13692] -0.813 -0.582 -0.832 -0.538 -0.648 ...
##  $ DOY        : num [1:13692] 7 7 7 7 7 8 8 8 9 9 ...
##  $ DOY_s      : num [1:13692] -2.06 -2.06 -2.06 -2.06 -2.06 ...
```

Inspect data

Inspect distribution of response variable (CPUE)

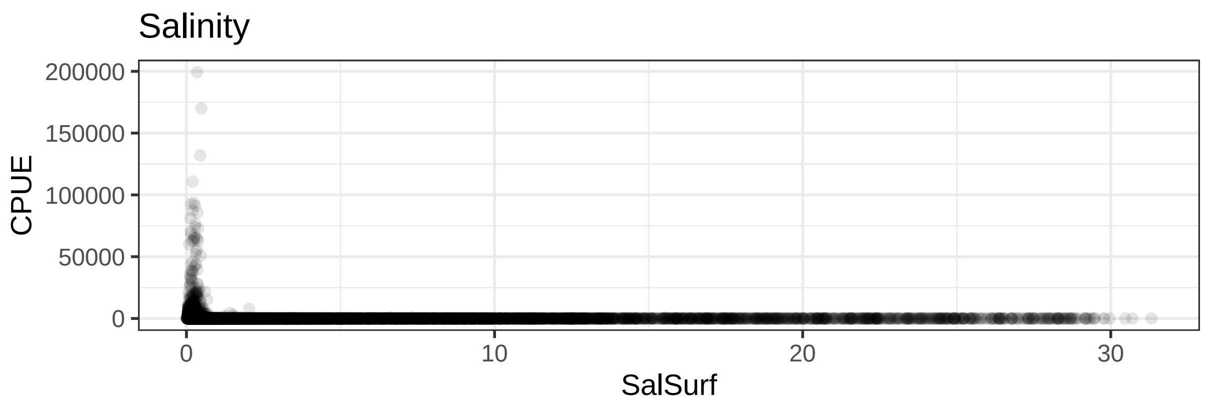
```
p1<-ggplot(BL, aes(x=CPUE))+
  geom_histogram()+
  coord_cartesian(expand = FALSE)+
  ggtitle("CPUE")+
  theme_bw()

p2<-ggplot(BL, aes(x=log(CPUE+1)))+
  geom_histogram()+
  coord_cartesian(expand = FALSE)+
  ggtitle("log(CPUE+1)")+
  theme_bw()
```

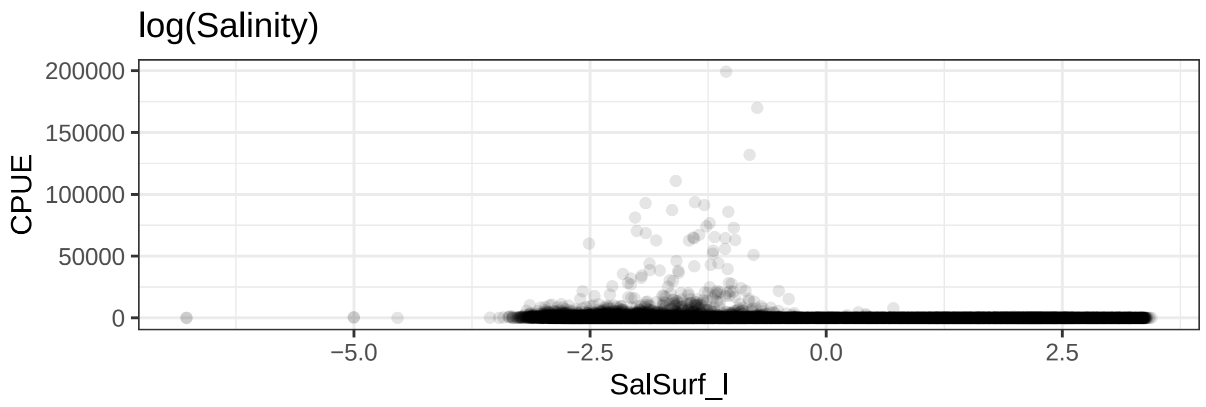
```
p1/p2
```



Why did I log-transform salinity?

```
p1<-ggplot(BL, aes(x=SalSurf, y=CPUE))+
  geom_point(alpha=0.1)+
  ggtitle("Salinity")+
  theme_bw()

p2<-ggplot(BL, aes(x=SalSurf_l, y=CPUE))+
  geom_point(alpha=0.1)+
  ggtitle("log(Salinity)")+
  theme_bw()

p1/p2
```

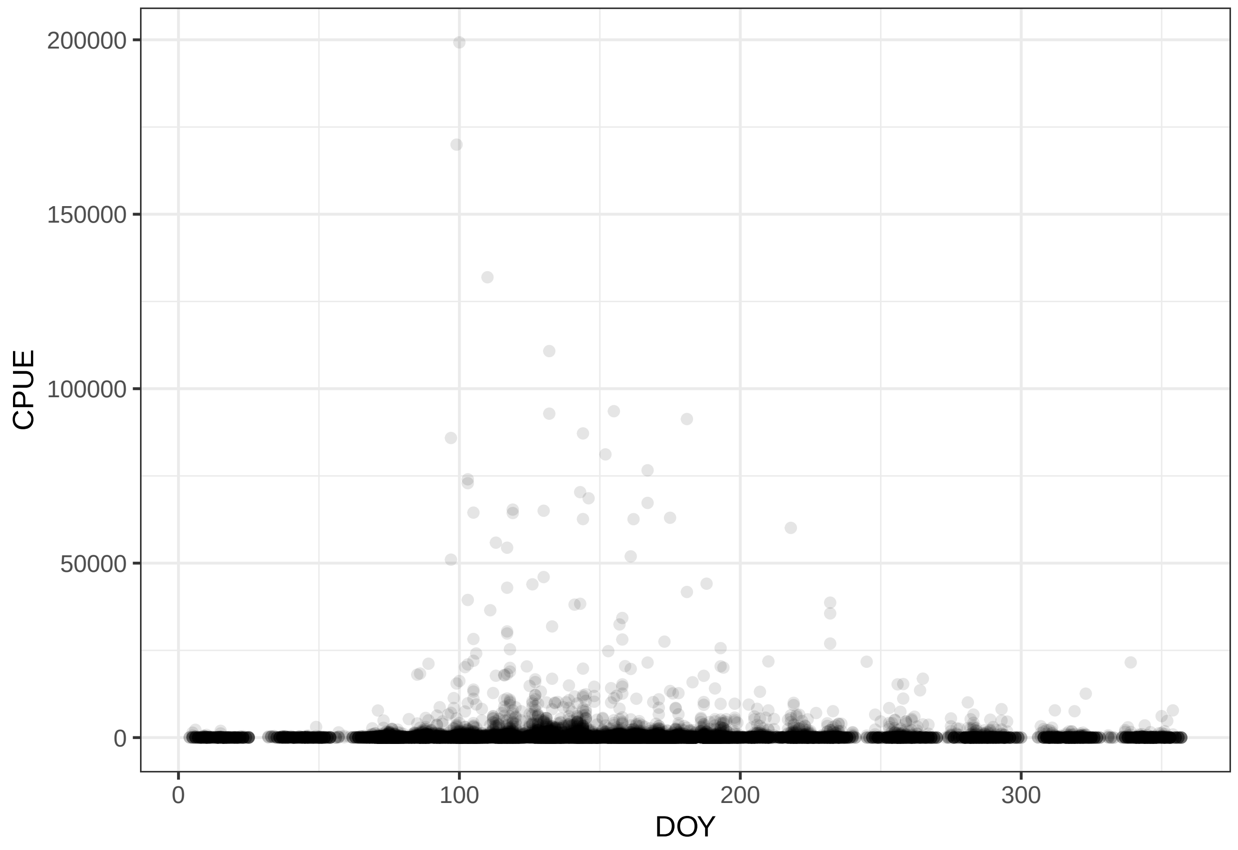




There is a much clearer unimodal relationship with the log-transformed salinity.

How do they vary seasonally?

```
ggplot(BL, aes(x=DOY, y=CPUE))+
  geom_point(alpha=0.1)+
  theme_bw()
```



Model

Set iterations

```r
iterations <- 5e3
warmup <- iterations/4
```

Super simple model

This model is purposefully bad (gaussian family when it should be hurdle_lognormal) The messages you will see are normal, but I'll suppress them for the other models

```r
m1<-brm(CPUE ~ SalSurf_l_s,
        data=BL, family=gaussian(),
        prior=prior(normal(0,10), class="Intercept")+
          prior(normal(0,5), class="b")+
          prior(cauchy(0,5), class="sigma"),
        chains=1, # Setting low to reduce computational. You should use >=3 chains
        iter = iterations, warmup = warmup,
        backend = "cmdstanr", threads = threading(5)) # Split the chain among 5 cores
```

```
## In file included from stan/lib/stan_math/stan/math/rev/functor.hpp:28,

##                  from stan/lib/stan_math/stan/math/rev.hpp:11,

##                  from stan/lib/stan_math/stan/math.hpp:19,
##                   from stan/src/stan/model/model_header.hpp:4,
##                   from C:/Users/SBASHE~1/AppData/Local/Temp/RtmpsPd3qJ/model-6db46f8228cc.hpp:3:
## stan/lib/stan_math/stan/math/rev/functor/reduce_sum.hpp: In instantiation of 'stan::math::internal::
## stan/lib/stan_math/stan/math/rev/functor/reduce_sum.hpp:249:23:   required from 'stan::math::var sta
## stan/lib/stan_math/stan/math/prim/functor/reduce_sum.hpp:207:60:   required from 'auto stan::math::r
## C:/Users/SBASHE~1/AppData/Local/Temp/RtmpsPd3qJ/model-6db46f8228cc.hpp:406:64:   required from 'stan
## C:/Users/SBASHE~1/AppData/Local/Temp/RtmpsPd3qJ/model-6db46f8228cc.hpp:663:77:   required from 'T_ f
## stan/src/stan/model/model_base_crtp.hpp:96:77:   required from 'stan::math::var stan::model::model_b
## stan/src/stan/model/model_base_crtp.hpp:93:20:   required from here
## stan/lib/stan_math/stan/math/rev/functor/reduce_sum.hpp:58:23: warning: 'stan::math::internal::reduc
##      scoped_args_tuple local_args_tuple_scope_;
##                        ^~~~~~~~~~~~~~~~~~~~~~
## stan/lib/stan_math/stan/math/rev/functor/reduce_sum.hpp:57:25: warning:   'std::tuple<const Eigen::M
##      std::tuple<Args...> args_tuple_;
##                          ^~~~~~~~~~~
## stan/lib/stan_math/stan/math/rev/functor/reduce_sum.hpp:63:5: warning:   when initialized here [-Wre
##      recursive_reducer(size_t num_vars_per_term, size_t num_vars_shared_terms,
##      ^~~~~~~~~~~~~~~~~
## stan/lib/stan_math/stan/math/rev/functor/reduce_sum.hpp: In instantiation of 'stan::math::internal::
## stan/lib/stan_math/lib/tbb_2020.3/include/tbb/parallel_reduce.h:186:27:   required from 'tbb::task* 
## stan/lib/stan_math/lib/tbb_2020.3/include/tbb/parallel_reduce.h:181:11:   required from here
## stan/lib/stan_math/stan/math/rev/functor/reduce_sum.hpp:58:23: warning: 'stan::math::internal::reduc
##      scoped_args_tuple local_args_tuple_scope_;
##                        ^~~~~~~~~~~~~~~~~~~~~~
## stan/lib/stan_math/stan/math/rev/functor/reduce_sum.hpp:57:25: warning:   'std::tuple<const Eigen::M
##      std::tuple<Args...> args_tuple_;
##                          ^~~~~~~~~~~
## stan/lib/stan_math/stan/math/rev/functor/reduce_sum.hpp:78:5: warning:   when initialized here [-Wre
##      recursive_reducer(recursive_reducer& other, tbb::split)
##      ^~~~~~~~~~~~~~~~~
## Start sampling
## Running MCMC with 1 chain, with 5 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
```

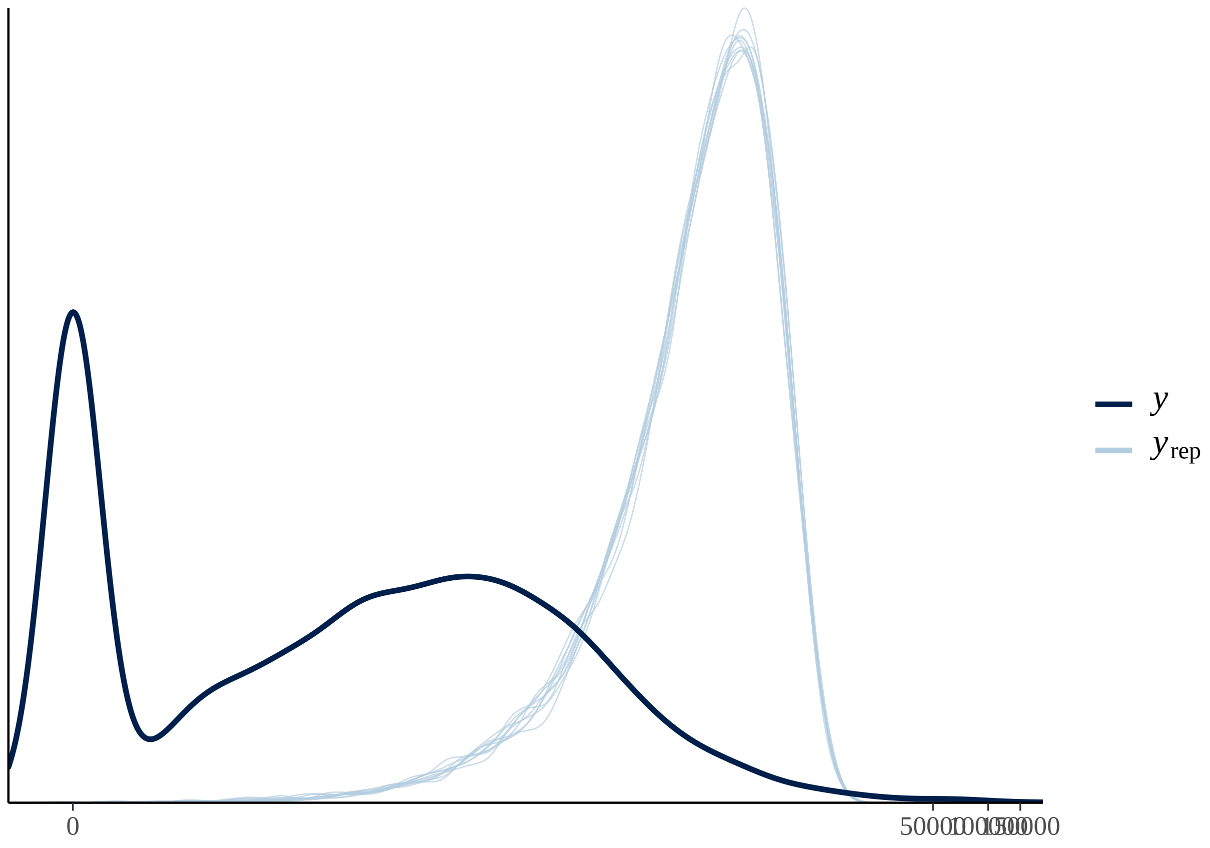
```
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 1 Iteration: 1251 / 5000 [ 25%]  (Sampling)
## Chain 1 Iteration: 1350 / 5000 [ 27%]  (Sampling)
## Chain 1 Iteration: 1450 / 5000 [ 29%]  (Sampling)
## Chain 1 Iteration: 1550 / 5000 [ 31%]  (Sampling)
## Chain 1 Iteration: 1650 / 5000 [ 33%]  (Sampling)
## Chain 1 Iteration: 1750 / 5000 [ 35%]  (Sampling)
## Chain 1 Iteration: 1850 / 5000 [ 37%]  (Sampling)
## Chain 1 Iteration: 1950 / 5000 [ 39%]  (Sampling)
## Chain 1 Iteration: 2050 / 5000 [ 41%]  (Sampling)
## Chain 1 Iteration: 2150 / 5000 [ 43%]  (Sampling)
## Chain 1 Iteration: 2250 / 5000 [ 45%]  (Sampling)
## Chain 1 Iteration: 2350 / 5000 [ 47%]  (Sampling)
## Chain 1 Iteration: 2450 / 5000 [ 49%]  (Sampling)
## Chain 1 Iteration: 2550 / 5000 [ 51%]  (Sampling)
## Chain 1 Iteration: 2650 / 5000 [ 53%]  (Sampling)
## Chain 1 Iteration: 2750 / 5000 [ 55%]  (Sampling)
## Chain 1 Iteration: 2850 / 5000 [ 57%]  (Sampling)
## Chain 1 Iteration: 2950 / 5000 [ 59%]  (Sampling)
## Chain 1 Iteration: 3050 / 5000 [ 61%]  (Sampling)
## Chain 1 Iteration: 3150 / 5000 [ 63%]  (Sampling)
## Chain 1 Iteration: 3250 / 5000 [ 65%]  (Sampling)
## Chain 1 Iteration: 3350 / 5000 [ 67%]  (Sampling)
## Chain 1 Iteration: 3450 / 5000 [ 69%]  (Sampling)
## Chain 1 Iteration: 3550 / 5000 [ 71%]  (Sampling)
## Chain 1 Iteration: 3650 / 5000 [ 73%]  (Sampling)
## Chain 1 Iteration: 3750 / 5000 [ 75%]  (Sampling)
## Chain 1 Iteration: 3850 / 5000 [ 77%]  (Sampling)
## Chain 1 Iteration: 3950 / 5000 [ 79%]  (Sampling)
## Chain 1 Iteration: 4050 / 5000 [ 81%]  (Sampling)
## Chain 1 Iteration: 4150 / 5000 [ 83%]  (Sampling)
## Chain 1 Iteration: 4250 / 5000 [ 85%]  (Sampling)
## Chain 1 Iteration: 4350 / 5000 [ 87%]  (Sampling)
## Chain 1 Iteration: 4450 / 5000 [ 89%]  (Sampling)
## Chain 1 Iteration: 4550 / 5000 [ 91%]  (Sampling)
## Chain 1 Iteration: 4650 / 5000 [ 93%]  (Sampling)
## Chain 1 Iteration: 4750 / 5000 [ 95%]  (Sampling)
## Chain 1 Iteration: 4850 / 5000 [ 97%]  (Sampling)
## Chain 1 Iteration: 4950 / 5000 [ 99%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1 finished in 4.1 seconds.
```

Explore simple model

Posterior predictive checks See this vignette for more info

```
pp_check(m1)+scale_x_continuous(trans="log1p")
```

```
## Using 10 posterior draws for ppc type 'dens_overlay' by default.

## Warning in self$trans$transform(x): NaNs produced

## Warning: Transformation introduced infinite values in continuous x-axis

## Warning: Removed 67915 rows containing non-finite values (stat_density).
```

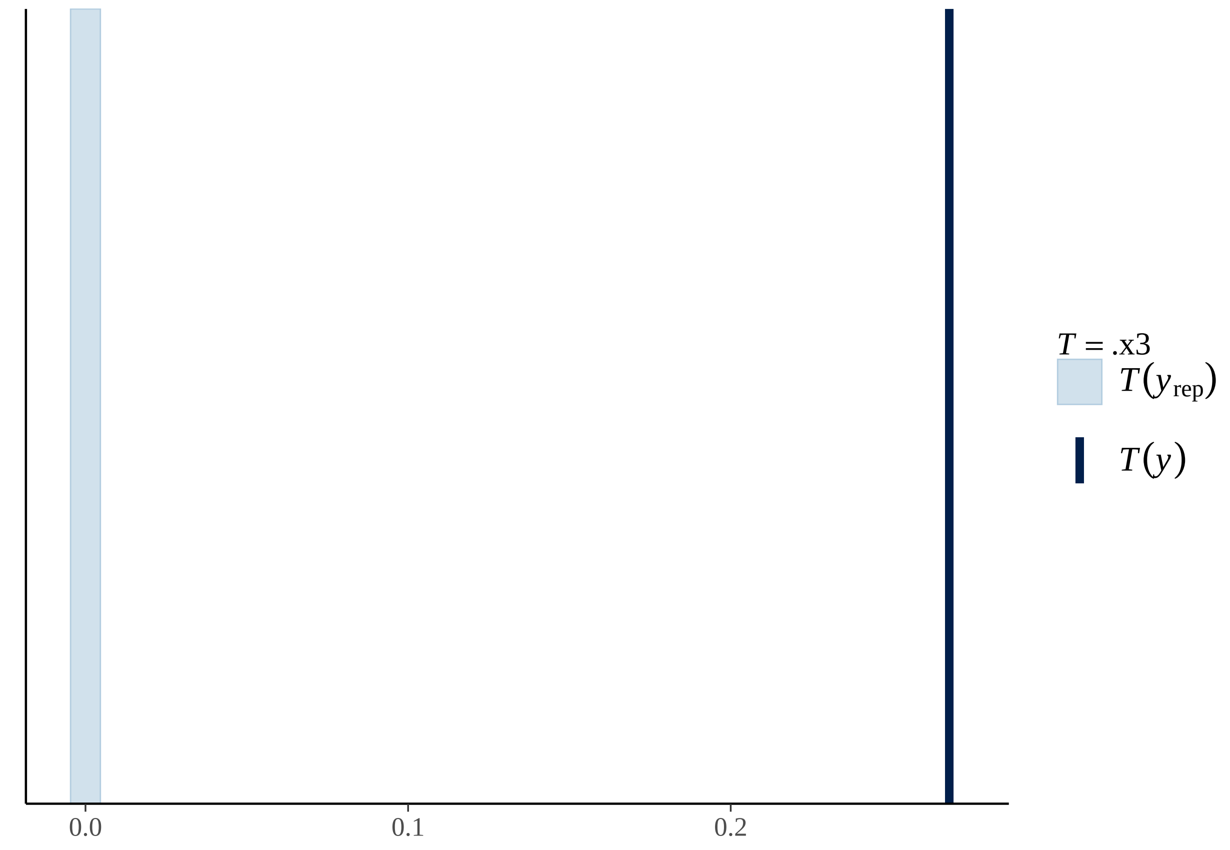


Each of the 10 light blue lines represents the density of predicted CPUE values from 1 posterior draw, while the dark line represents the density of CPUE values in the actual data. This is bad, the two density plots should be overlapping, but they look completely different

Check the proportion of zeroes in the data and predicted by the model

```
prop_zero <- function(x) mean(x == 0) # Calculates proportion of values that are zero
pp_check(m1, type="stat", stat=prop_zero)
```

```
## Using all posterior draws for ppc type 'stat' by default.

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



The light blue histogram represents the distribution of the proportion of zeroes among all the posterior draws. The dark blue line represents the actual proportion of zeroes in the data. This is also bad, the model is predicting no zeroes, while the dataset is ~1/3 zeroes

Simple model

Let's try a better model using the hurdle_lognormal distribution

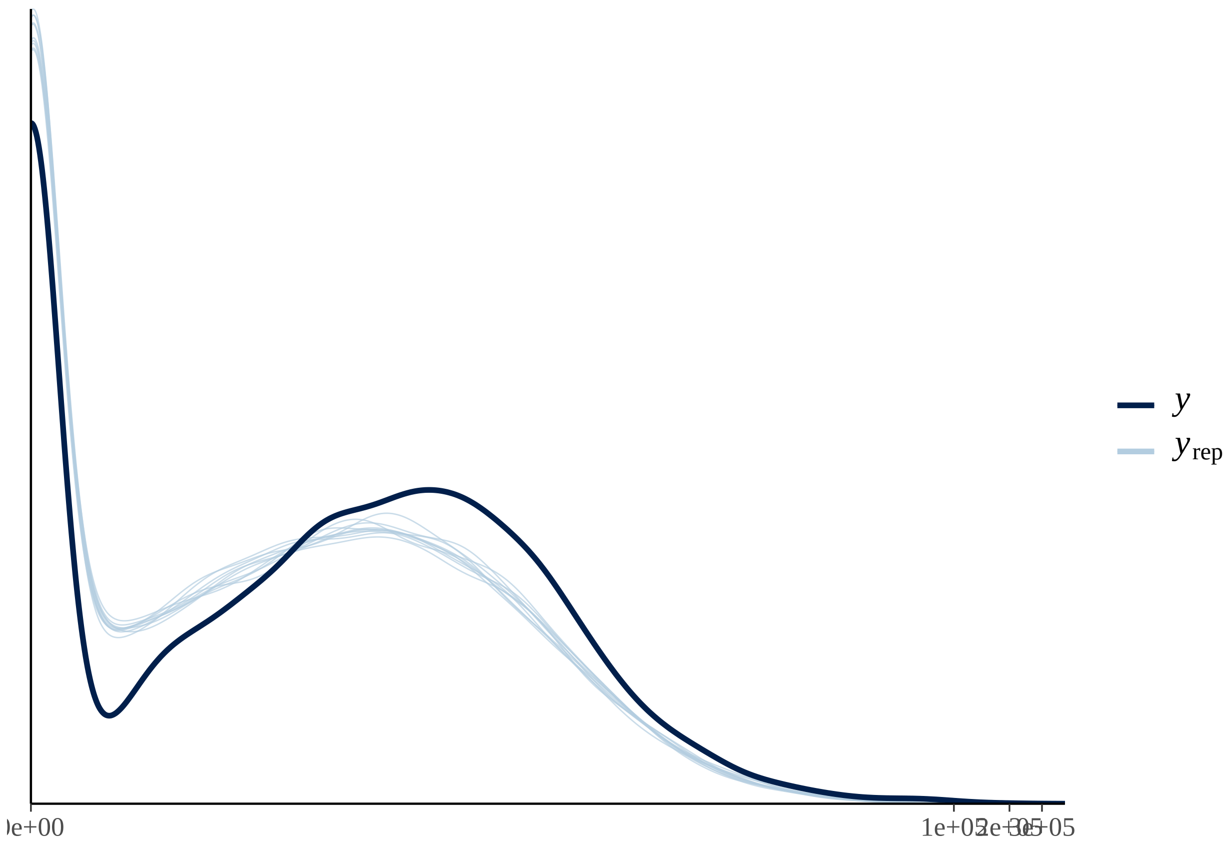
```r
m2<-brm(CPUE ~ SalSurf_l_s,
        data=BL, family=hurdle_lognormal(),
        prior=prior(normal(0,10), class="Intercept")+
          prior(normal(0,5), class="b")+
          prior(cauchy(0,5), class="sigma"),
        chains=1,
        iter = iterations, warmup = warmup,
        backend = "cmdstanr", threads = threading(5)) # Split the chain among 5 cores
```

```
## Running MCMC with 1 chain, with 5 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 1 Iteration: 1251 / 5000 [ 25%]  (Sampling)
## Chain 1 Iteration: 1350 / 5000 [ 27%]  (Sampling)
## Chain 1 Iteration: 1450 / 5000 [ 29%]  (Sampling)
## Chain 1 Iteration: 1550 / 5000 [ 31%]  (Sampling)
## Chain 1 Iteration: 1650 / 5000 [ 33%]  (Sampling)
## Chain 1 Iteration: 1750 / 5000 [ 35%]  (Sampling)
## Chain 1 Iteration: 1850 / 5000 [ 37%]  (Sampling)
## Chain 1 Iteration: 1950 / 5000 [ 39%]  (Sampling)
## Chain 1 Iteration: 2050 / 5000 [ 41%]  (Sampling)
## Chain 1 Iteration: 2150 / 5000 [ 43%]  (Sampling)
## Chain 1 Iteration: 2250 / 5000 [ 45%]  (Sampling)
## Chain 1 Iteration: 2350 / 5000 [ 47%]  (Sampling)
## Chain 1 Iteration: 2450 / 5000 [ 49%]  (Sampling)
## Chain 1 Iteration: 2550 / 5000 [ 51%]  (Sampling)
## Chain 1 Iteration: 2650 / 5000 [ 53%]  (Sampling)
## Chain 1 Iteration: 2750 / 5000 [ 55%]  (Sampling)
## Chain 1 Iteration: 2850 / 5000 [ 57%]  (Sampling)
## Chain 1 Iteration: 2950 / 5000 [ 59%]  (Sampling)
## Chain 1 Iteration: 3050 / 5000 [ 61%]  (Sampling)
## Chain 1 Iteration: 3150 / 5000 [ 63%]  (Sampling)
## Chain 1 Iteration: 3250 / 5000 [ 65%]  (Sampling)
## Chain 1 Iteration: 3350 / 5000 [ 67%]  (Sampling)
## Chain 1 Iteration: 3450 / 5000 [ 69%]  (Sampling)
## Chain 1 Iteration: 3550 / 5000 [ 71%]  (Sampling)
## Chain 1 Iteration: 3650 / 5000 [ 73%]  (Sampling)
## Chain 1 Iteration: 3750 / 5000 [ 75%]  (Sampling)
## Chain 1 Iteration: 3850 / 5000 [ 77%]  (Sampling)
## Chain 1 Iteration: 3950 / 5000 [ 79%]  (Sampling)
## Chain 1 Iteration: 4050 / 5000 [ 81%]  (Sampling)
## Chain 1 Iteration: 4150 / 5000 [ 83%]  (Sampling)
## Chain 1 Iteration: 4250 / 5000 [ 85%]  (Sampling)
## Chain 1 Iteration: 4350 / 5000 [ 87%]  (Sampling)
## Chain 1 Iteration: 4450 / 5000 [ 89%]  (Sampling)
## Chain 1 Iteration: 4550 / 5000 [ 91%]  (Sampling)
## Chain 1 Iteration: 4650 / 5000 [ 93%]  (Sampling)
## Chain 1 Iteration: 4750 / 5000 [ 95%]  (Sampling)
## Chain 1 Iteration: 4850 / 5000 [ 97%]  (Sampling)
## Chain 1 Iteration: 4950 / 5000 [ 99%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1 finished in 51.1 seconds.
```

Posterior predictive check

```r
pp_check(m2)+scale_x_continuous(trans="log1p")
```

```
## Using 10 posterior draws for ppc type 'dens_overlay' by default.
```



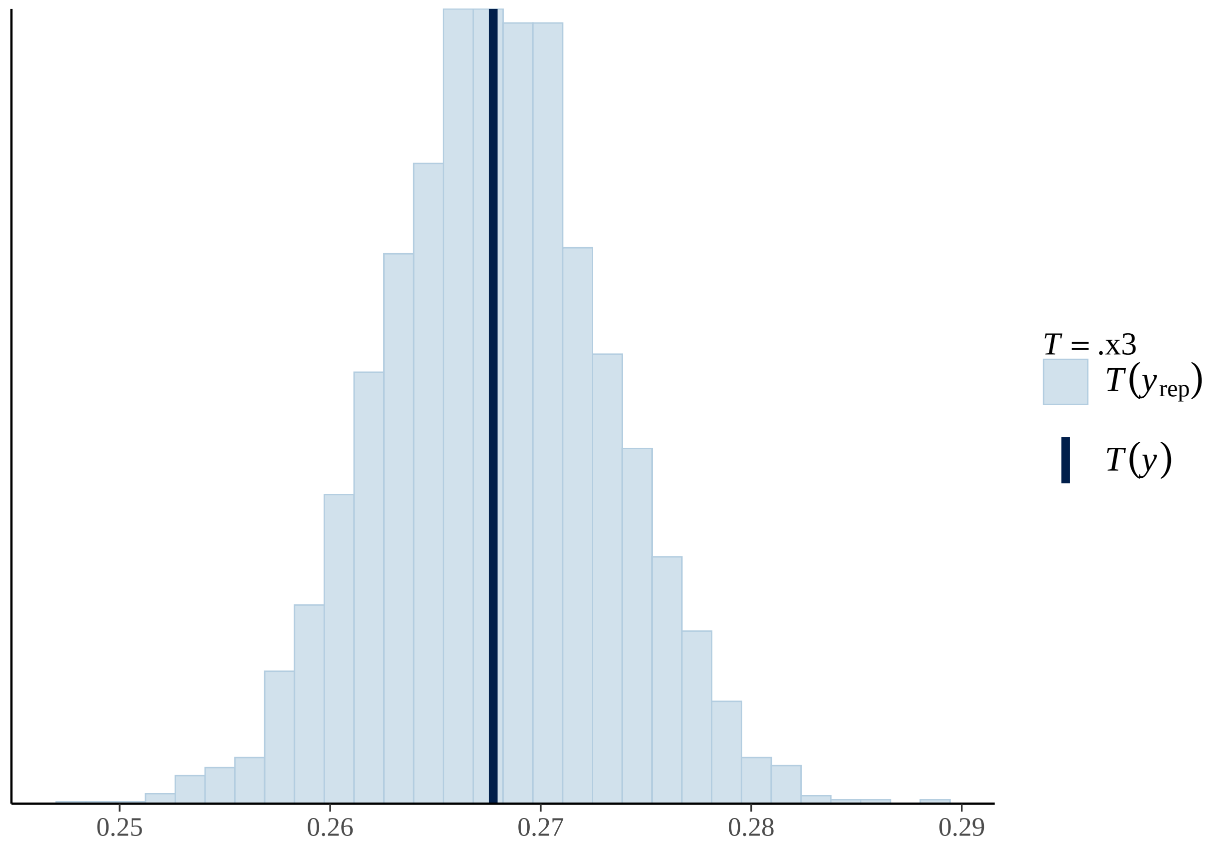
This is way better, but still not great

Check the proportion of zeroes in the data and predicted by the model

```
pp_check(m2, type="stat", stat=prop_zero)
```

```
## Using all posterior draws for ppc type 'stat' by default.
```

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



This is great! The model is now capturing the correct proportion of zeroes.

Complex model

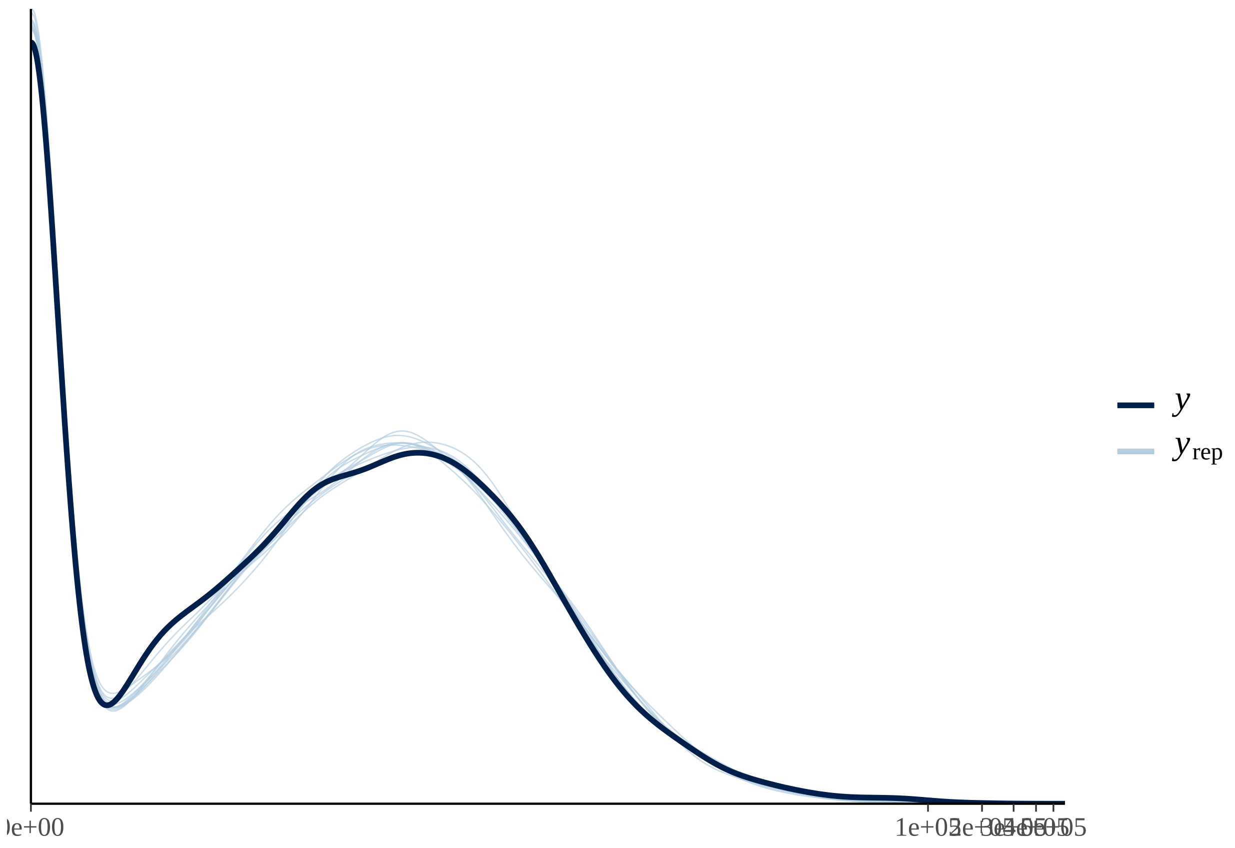
```
m3<-brm(bf(CPUE ~ t2(DOY_s, SalSurf_l_s) + (1|Station),
           hu ~ s(SalSurf_l_s, bs="cr", k=5)),
        data=BL, family=hurdle_lognormal(),
        prior=prior(normal(0,10), class="Intercept")+
          prior(normal(0,5), class="b")+
          prior(cauchy(0,5), class="sigma"),
        chains=1,
        control=list(adapt_delta=0.9),
        iter = iterations, warmup = warmup,
        backend = "cmdstanr", threads = threading(5))
```

```
## Running MCMC with 1 chain, with 5 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 1 Iteration: 1251 / 5000 [ 25%]  (Sampling)
## Chain 1 Iteration: 1350 / 5000 [ 27%]  (Sampling)
## Chain 1 Iteration: 1450 / 5000 [ 29%]  (Sampling)
## Chain 1 Iteration: 1550 / 5000 [ 31%]  (Sampling)
## Chain 1 Iteration: 1650 / 5000 [ 33%]  (Sampling)
## Chain 1 Iteration: 1750 / 5000 [ 35%]  (Sampling)
## Chain 1 Iteration: 1850 / 5000 [ 37%]  (Sampling)
## Chain 1 Iteration: 1950 / 5000 [ 39%]  (Sampling)
## Chain 1 Iteration: 2050 / 5000 [ 41%]  (Sampling)
## Chain 1 Iteration: 2150 / 5000 [ 43%]  (Sampling)
## Chain 1 Iteration: 2250 / 5000 [ 45%]  (Sampling)
## Chain 1 Iteration: 2350 / 5000 [ 47%]  (Sampling)
## Chain 1 Iteration: 2450 / 5000 [ 49%]  (Sampling)
## Chain 1 Iteration: 2550 / 5000 [ 51%]  (Sampling)
## Chain 1 Iteration: 2650 / 5000 [ 53%]  (Sampling)
## Chain 1 Iteration: 2750 / 5000 [ 55%]  (Sampling)
## Chain 1 Iteration: 2850 / 5000 [ 57%]  (Sampling)
## Chain 1 Iteration: 2950 / 5000 [ 59%]  (Sampling)
## Chain 1 Iteration: 3050 / 5000 [ 61%]  (Sampling)
## Chain 1 Iteration: 3150 / 5000 [ 63%]  (Sampling)
## Chain 1 Iteration: 3250 / 5000 [ 65%]  (Sampling)
## Chain 1 Iteration: 3350 / 5000 [ 67%]  (Sampling)
## Chain 1 Iteration: 3450 / 5000 [ 69%]  (Sampling)
## Chain 1 Iteration: 3550 / 5000 [ 71%]  (Sampling)
## Chain 1 Iteration: 3650 / 5000 [ 73%]  (Sampling)
## Chain 1 Iteration: 3750 / 5000 [ 75%]  (Sampling)
## Chain 1 Iteration: 3850 / 5000 [ 77%]  (Sampling)
## Chain 1 Iteration: 3950 / 5000 [ 79%]  (Sampling)
## Chain 1 Iteration: 4050 / 5000 [ 81%]  (Sampling)
## Chain 1 Iteration: 4150 / 5000 [ 83%]  (Sampling)
## Chain 1 Iteration: 4250 / 5000 [ 85%]  (Sampling)
## Chain 1 Iteration: 4350 / 5000 [ 87%]  (Sampling)
## Chain 1 Iteration: 4450 / 5000 [ 89%]  (Sampling)
## Chain 1 Iteration: 4550 / 5000 [ 91%]  (Sampling)
## Chain 1 Iteration: 4650 / 5000 [ 93%]  (Sampling)
## Chain 1 Iteration: 4750 / 5000 [ 95%]  (Sampling)
## Chain 1 Iteration: 4850 / 5000 [ 97%]  (Sampling)
## Chain 1 Iteration: 4950 / 5000 [ 99%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1 finished in 6231.1 seconds.
```

Posterior predictive check

```
pp_check(m3)+scale_x_continuous(trans="log1p")
```

```
## Using 10 posterior draws for ppc type 'dens_overlay' by default.
```



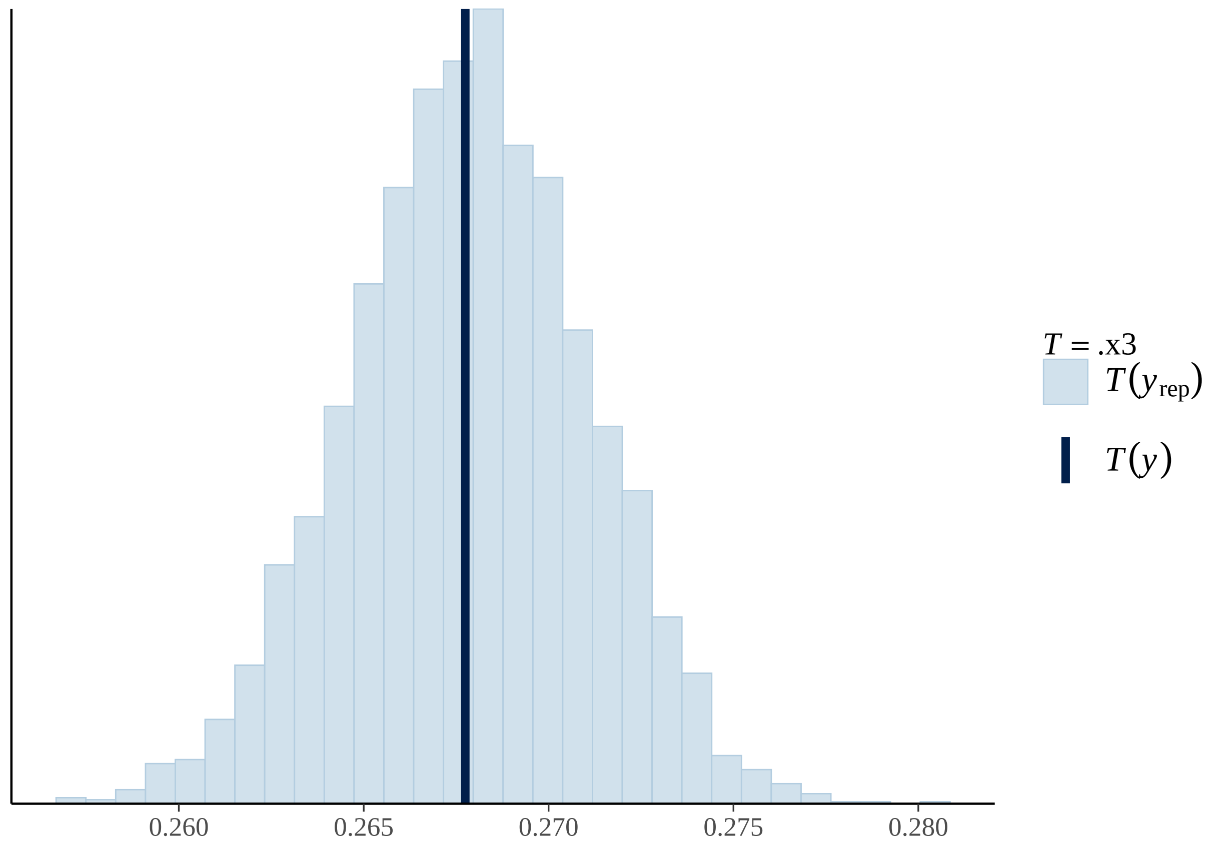
This is now great.

Check the proportion of zeroes in the data and predicted by the model

```
pp_check(m3, type="stat", stat=prop_zero)
```

```
## Using all posterior draws for ppc type 'stat' by default.
```

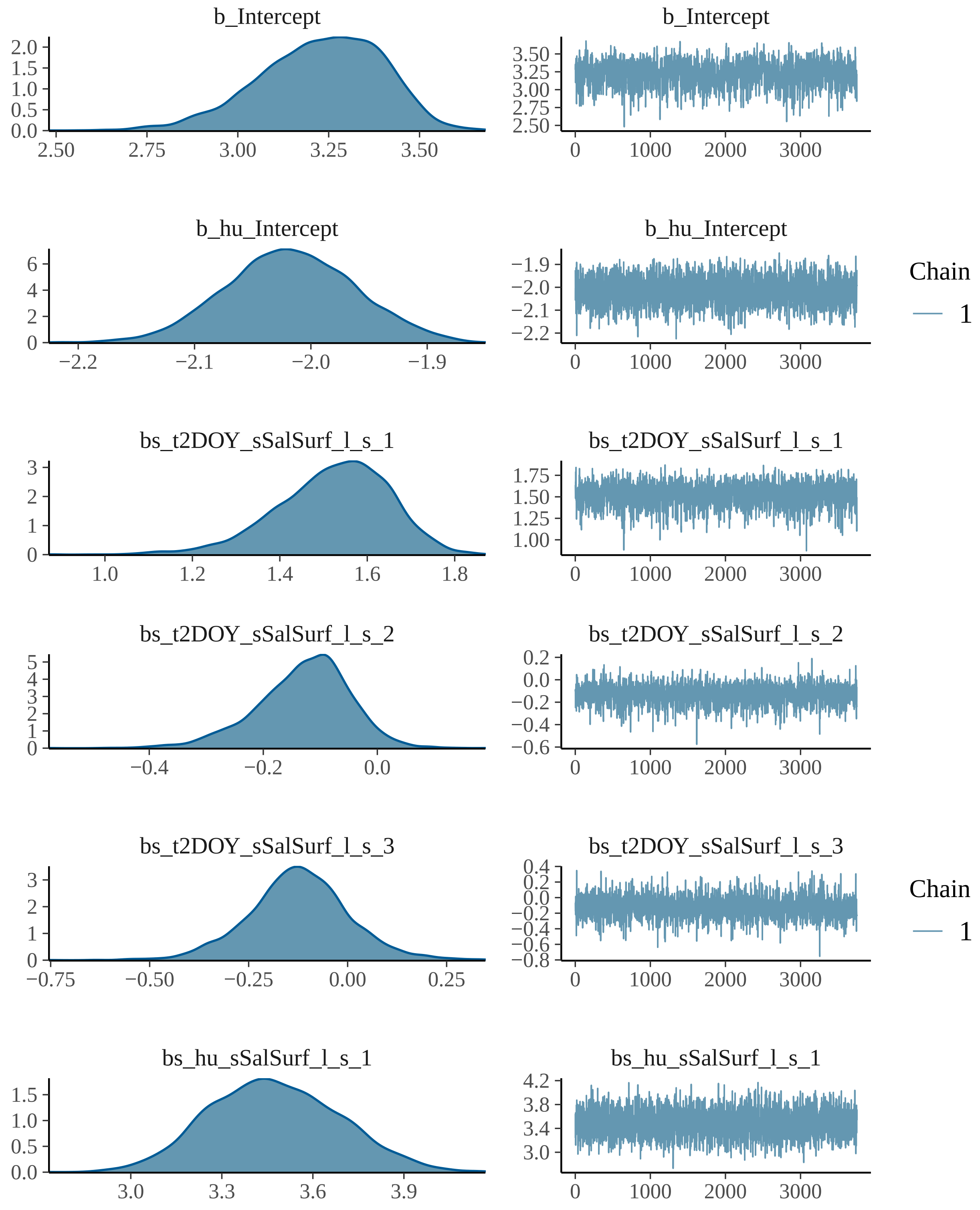
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

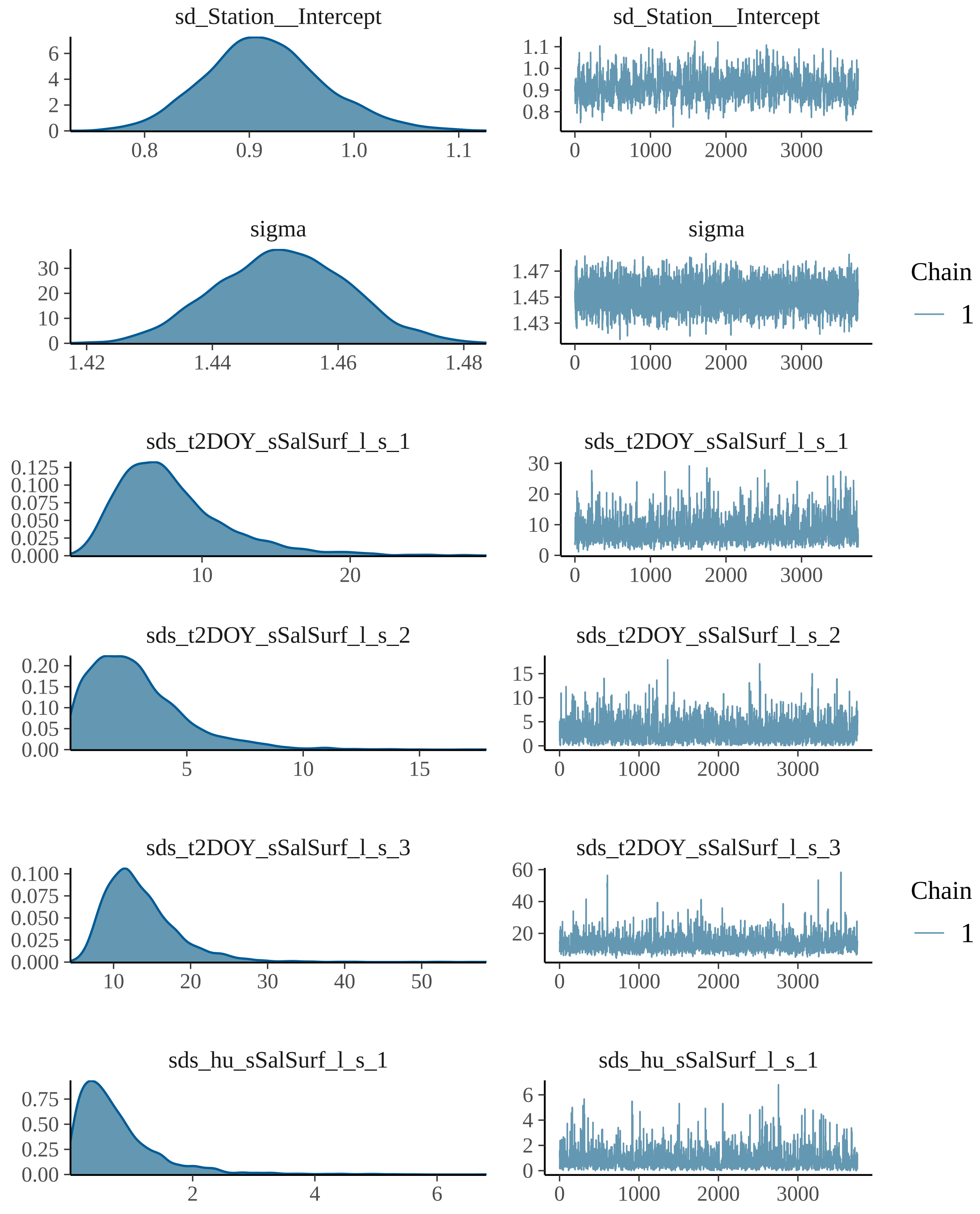


This is still great, practically the same as the prior model.

Since the model is much improved, let's check the chains as well. You want this to look like a fuzzy caterpillar, not a snake.

```
plot(m3, ask=F, N=3)
```


b_Intercept
b_Intercept
b_hu_Intercept
b_hu_Intercept
Chain
1
bs_t2DOY_sSalSurf_l_s_1
bs_t2DOY_sSalSurf_l_s_1
bs_t2DOY_sSalSurf_l_s_2
bs_t2DOY_sSalSurf_l_s_2
bs_t2DOY_sSalSurf_l_s_3
bs_t2DOY_sSalSurf_l_s_3
Chain
1
bs_hu_sSalSurf_l_s_1
bs_hu_sSalSurf_l_s_1
0
1000
2000
3000



Looks good!

Let's look at a summary of the model outputs

```r
summary(m3)
```

```
## Warning: There were 4 divergent transitions after warmup. Increasing adapt_delta
## above may help. See http://mc-stan.org/misc/warnings.html#divergent-transitions-
## after-warmup

##  Family: hurdle_lognormal
##   Links: mu = identity; sigma = identity; hu = logit
## Formula: CPUE ~ t2(DOY_s, SalSurf_l_s) + (1 | Station)
##          hu ~ s(SalSurf_l_s, bs = "cr", k = 5)
##    Data: BL (Number of observations: 13692)
##   Draws: 1 chains, each with iter = 3750; warmup = 0; thin = 1;
##          total post-warmup draws = 3750
##
## Smooth Terms:
##                            Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
## sds(t2DOY_sSalSurf_l_s_1)     7.86      3.97     2.84    17.75 1.00      685
## sds(t2DOY_sSalSurf_l_s_2)     2.88      2.12     0.16     8.14 1.00      813
## sds(t2DOY_sSalSurf_l_s_3)    13.65      4.84     7.12    25.72 1.00      709
## sds(hu_sSalSurf_l_s_1)        0.74      0.71     0.03     2.64 1.00      500
##                            Tail_ESS
## sds(t2DOY_sSalSurf_l_s_1)     1268
## sds(t2DOY_sSalSurf_l_s_2)      935
## sds(t2DOY_sSalSurf_l_s_3)     1274
## sds(hu_sSalSurf_l_s_1)         469
##
## Group-Level Effects:
## ~Station (Number of levels: 211)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.92      0.06     0.82     1.04 1.00      433      869
##
## Population-Level Effects:
##                       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
## Intercept                 3.22      0.18     2.84     3.52 1.00      456
## hu_Intercept             -2.02      0.06    -2.13    -1.91 1.00     1965
## t2DOY_sSalSurf_l_s_1      1.52      0.13     1.21     1.73 1.00      826
## t2DOY_sSalSurf_l_s_2     -0.14      0.08    -0.31     0.01 1.00      942
## t2DOY_sSalSurf_l_s_3     -0.13      0.12    -0.39     0.12 1.00     1076
## hu_sSalSurf_l_s_1         3.48      0.22     3.07     3.94 1.00     1099
##                       Tail_ESS
## Intercept                 1051
## hu_Intercept              2498
## t2DOY_sSalSurf_l_s_1      1394
## t2DOY_sSalSurf_l_s_2      1566
## t2DOY_sSalSurf_l_s_3      1656
## hu_sSalSurf_l_s_1         1667
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     1.45      0.01     1.43     1.47 1.00     5002     2962
##
## Draws were sampled using sample(hmc). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

The effective sample sizes ("ESS") should be >100 per chain, and Rhat should be <1.05, so we're good on both those metrics. However, there was a warning about divergent transitions, so the `adapt_delta` parameter should be increased further for a real analysis.

Extract the results

Now that we have a good model, we can look more closely at the model and the results!

Set up a dataset with a range of covariates to explore the model results

```
newdata<-expand_grid(SalSurf=round(quantile(BL$SalSurf, seq(0.05, 0.95, by=0.05)), 4),
                     DOY=seq(min(BL$DOY), max(BL$DOY), length.out=60))%>%
  # standardize to match model inputs
  mutate(DOY=round(DOY),
         SalSurf_l_s=(log(SalSurf)-mean(BL$SalSurf_l))/sd(BL$SalSurf_l),
         DOY_s=(DOY-mean(DOY))/sd(DOY))
```

Create model predictions to visualize

```
# We're using the fitted function here to get the predicted mean response values
# Using the predict function would return simulated response values
pred<-fitted(m3, newdata=newdata, re_formula=NA)

newdata_pred<-newdata%>%
  mutate(prediction=pred[,"Estimate"], # Add predicted values
         l95=pred[,"Q2.5"], # Add lower 95% credible intervals
         u95=pred[,"Q97.5"]) # Add upper 95% credible intervals
```

Where do the 95% credible intervals come from? They are just the 95% quantiles of the posterior distribution. We could calculate the same metric this way

```
pred_full<-fitted(m3, newdata=newdata, re_formula=NA, summary=FALSE)

str(pred_full)
```

```
##  num [1:3750, 1:1140] 176 189 206 237 203 ...
```

So this full set of predictions is a matrix with rows representing the 3750 posterior draws (one for each model iteration, note that `iterations-warmup`=3750) and columns representing the 1140 rows in the `newdata` file.

Calculate credible intervals with quantiles

```
pred_full_sum<-matrix(nrow=nrow(newdata), ncol=3, dimnames = list(NULL, c("Estimate",
↪  "Q2.5", "Q97.5")))

pred_full_sum[,"Estimate"]<-apply(pred_full, MARGIN=2, mean)
pred_full_sum[,"Q2.5"]<-apply(pred_full, MARGIN=2, quantile, probs=0.025)
pred_full_sum[,"Q97.5"]<-apply(pred_full, MARGIN=2, quantile, probs=0.975)

str(pred_full_sum)
```

```
##  num [1:1140, 1:3] 202 237 276 320 369 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:3] "Estimate" "Q2.5" "Q97.5"
```

Now how does this compare to the first way we calculated the credible intervals?

```r
# Estimate
all(near(pred[,"Estimate"], pred_full_sum[,"Estimate"]))
```

```
## [1] TRUE
```

```r
# Lower 95% quantile
all(near(pred[,"Q2.5"], pred_full_sum[,"Q2.5"]))
```

```
## [1] TRUE
```

```r
# Upper 95% quantile
all(near(pred[,"Q97.5"], pred_full_sum[,"Q97.5"]))
```

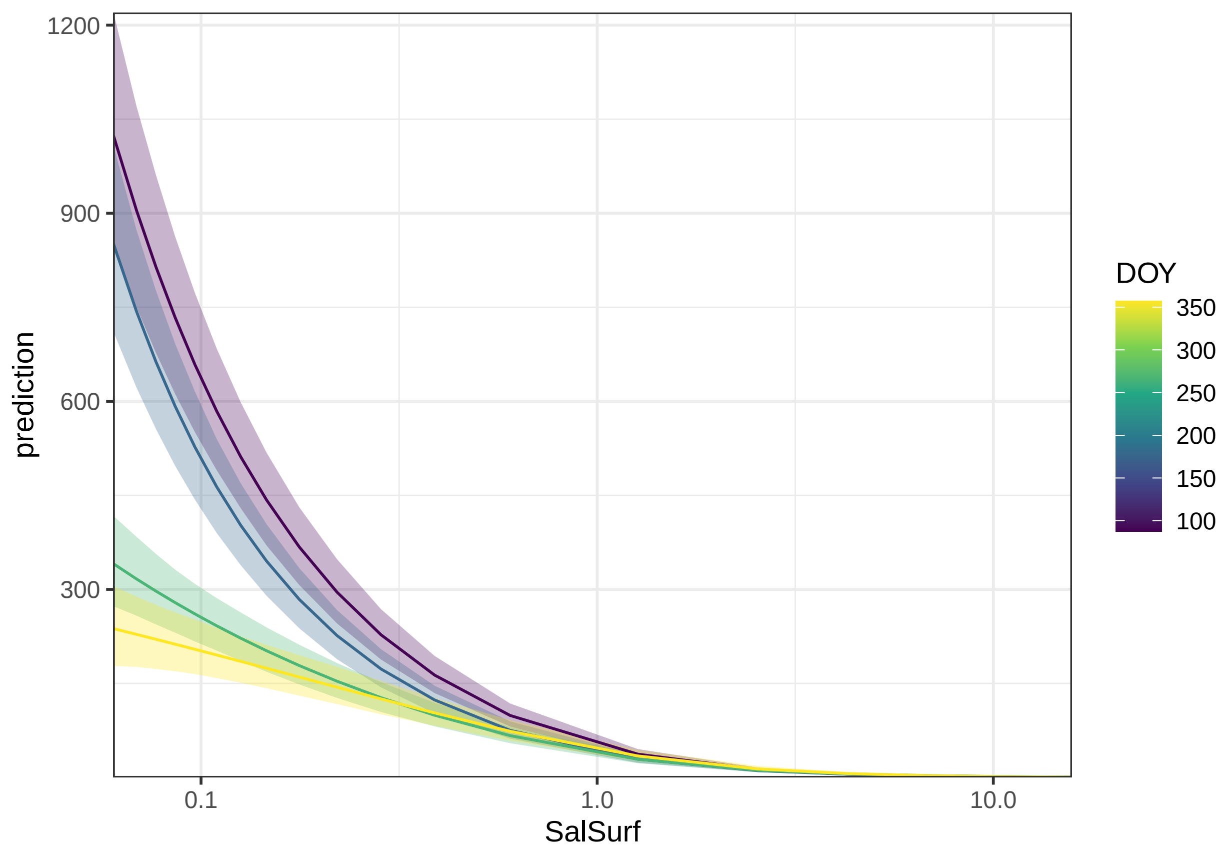
```
## [1] TRUE
```

Success! They are all identical

Visualize the results

CPUE by salinity (with salinity on a log scale)

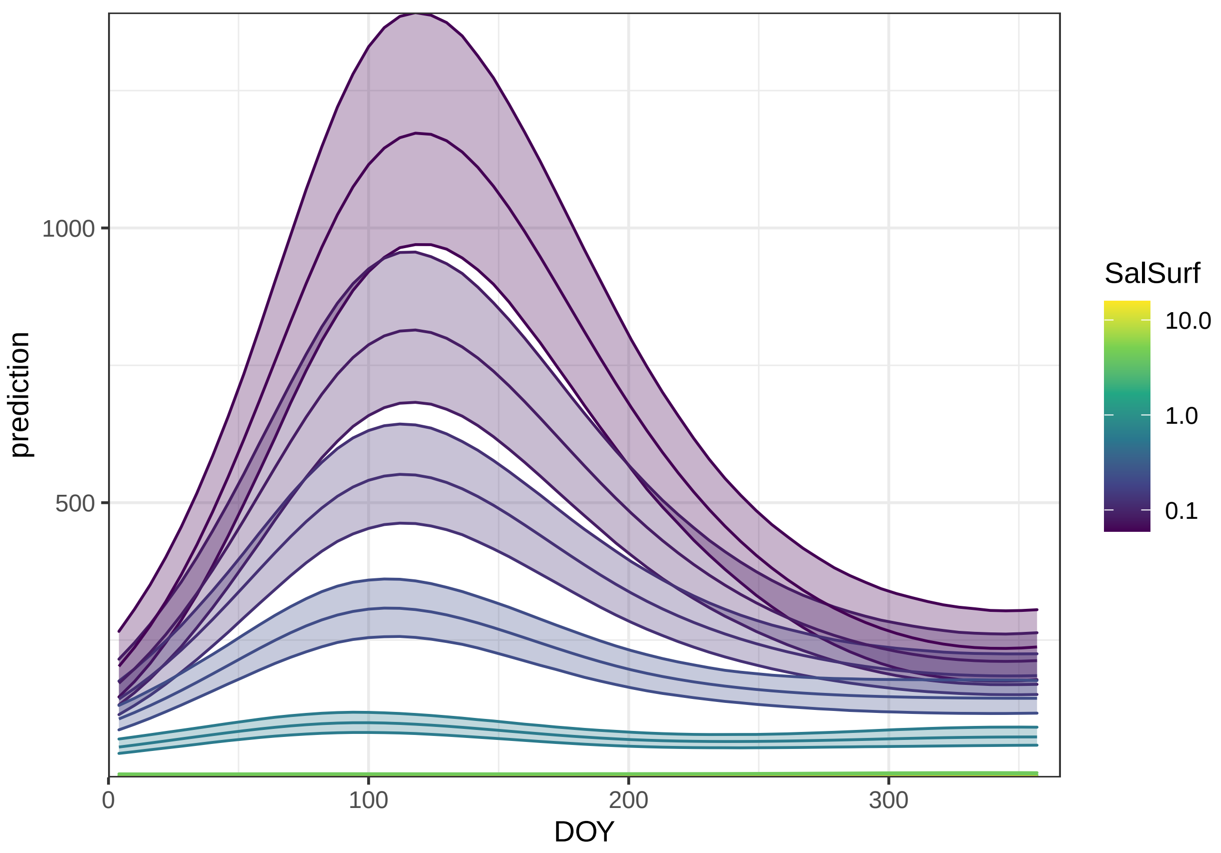
```r
ggplot(filter(newdata_pred, DOY%in%unique(DOY)[1:4*15]),
       aes(x=SalSurf, y=prediction, ymin=l95, ymax=u95, fill=DOY, color=DOY, group=DOY))+
  geom_ribbon(alpha=0.3, color=NA)+
  geom_line()+
  scale_fill_viridis_c(aesthetics = c("color", "fill"))+
  scale_x_continuous(trans="log", breaks=scales::breaks_log())+
  coord_cartesian(expand = FALSE)+
  theme_bw()
```



As you might expect for a Cladoceran (freshwater taxa), abundance increases with lower salinity.

CPUE by seasonality (DOY)

```
ggplot(filter(newdata_pred, SalSurf%in%unique(SalSurf)[seq(1,19, by=3)]),
       aes(x=DOY, y=prediction, ymin=l95, ymax=u95,
           fill=SalSurf, color=SalSurf, group=SalSurf))+
  geom_ribbon(alpha=0.3)+
  geom_line()+
  scale_fill_viridis_c(aesthetics = c("color", "fill"),
                       trans="log", breaks=scales::breaks_log())+
  coord_cartesian(expand = FALSE, xlim=c(0, 366))+
  theme_bw()
```



It looks like abundance peaks around the end of April, and there doesn't seem to be an interaction of DOY with Salinity.

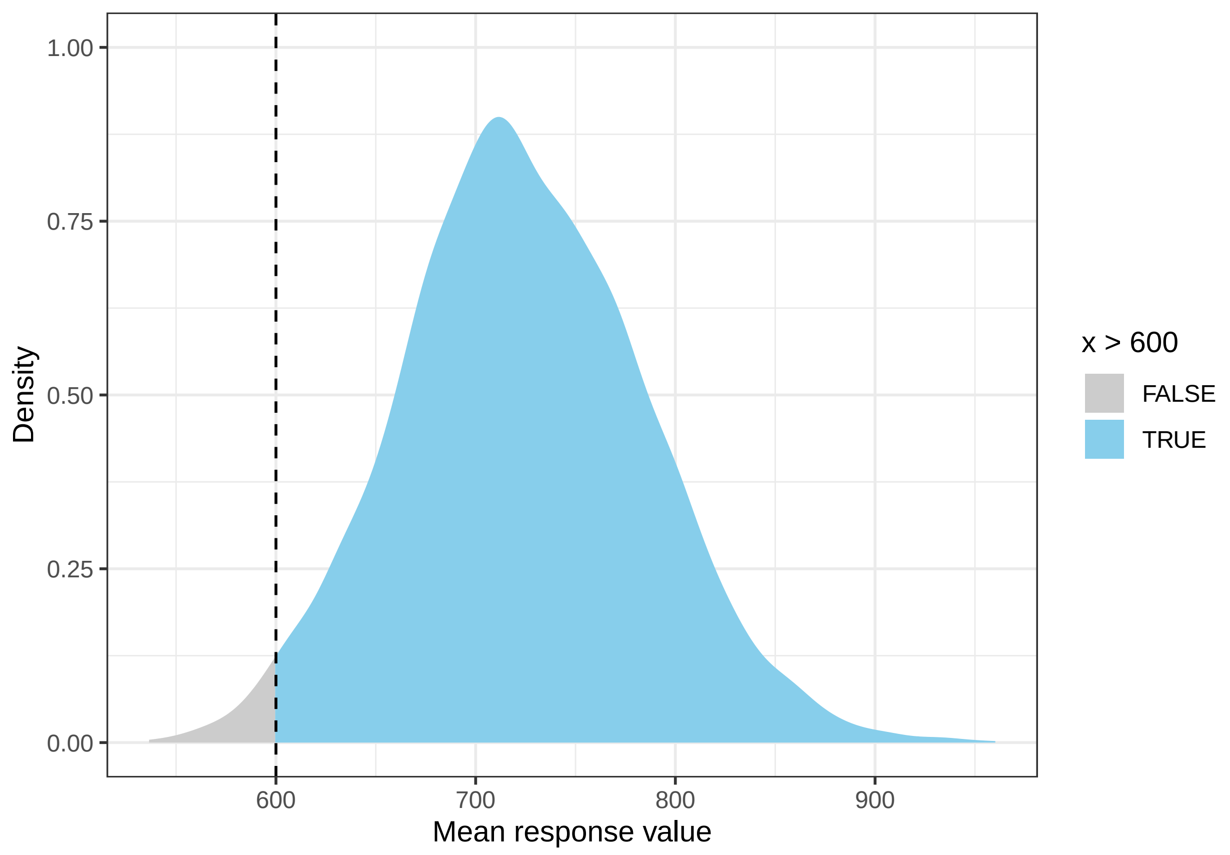
Exploring the posterior

Let's dive deeper into the model to take advantage of our Bayesian analysis and full posterior distribution

Earlier we explored the mean predicted values with 95% credible intervals, but what does the full posterior actually look like? We'll use the `tidybayes` package, which is an excellent way to explore and understand your Bayesian model. To simplify things, We'll explore the posterior for 1 scenario: DOY 118 and Salinity 0.0965, guided by the question: how often is CPUE > 600?

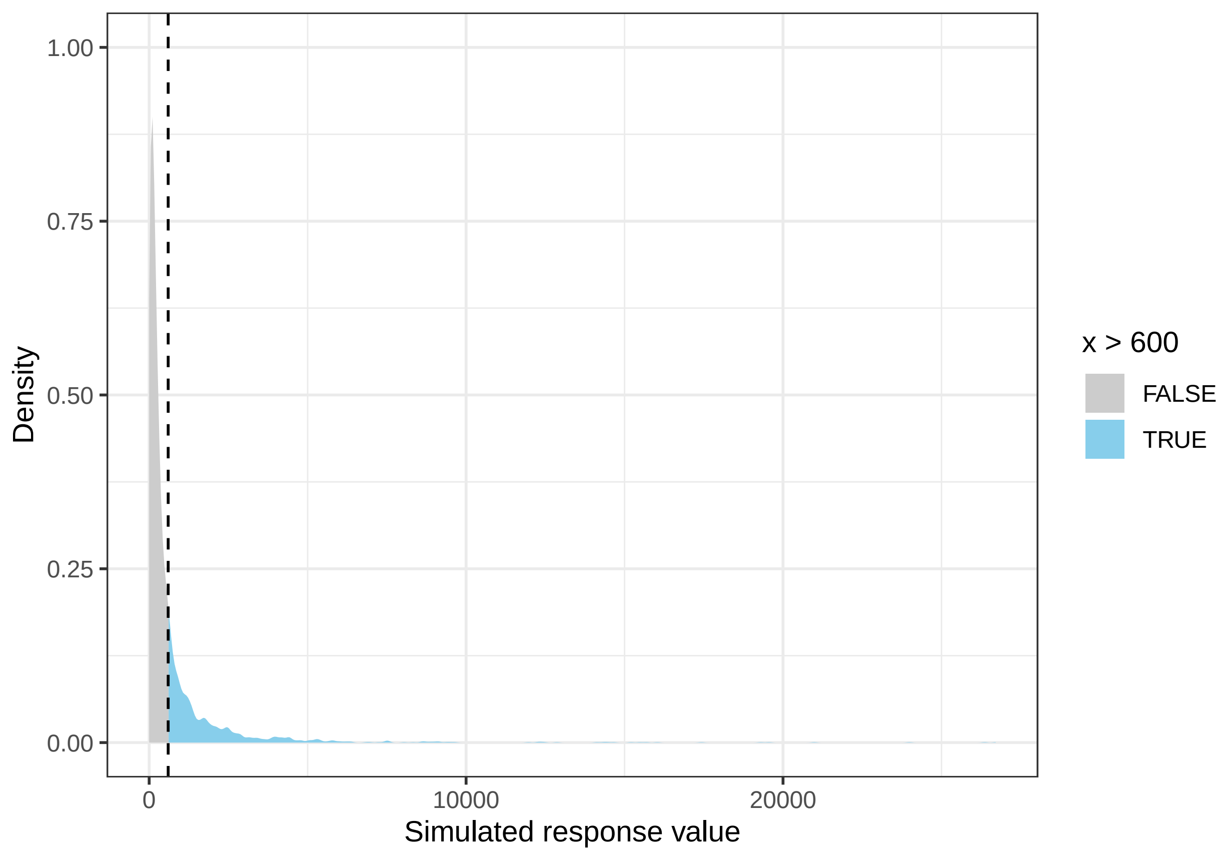
Here is the distribution of mean response values from the posterior

```
all_epreds<-filter(newdata, SalSurf==0.0965 & DOY==118)%>%
  add_epred_draws(m3, re_formula = NA)

ggplot(all_epreds, aes(x=.epred, fill=stat(x>600)))+
  stat_slab()+
  geom_vline(xintercept = 600, linetype = "dashed")+
  xlab("Mean response value")+
  ylab("Density")+
  scale_fill_manual(values = c("gray80", "skyblue"))+
  theme_bw()
```



Here is the distribution of simulated response values from the posterior (e.g., simulating an actual zooplankton sample, so it incorporates the model-family variance)

```
all_preds<-filter(newdata, SalSurf==0.0965 & DOY==118)%>%
  add_predicted_draws(m3, re_formula = NA)

ggplot(all_preds, aes(x=.prediction, fill=stat(x>600)))+
  stat_slab()+
  geom_vline(xintercept = 600, linetype = "dashed")+
  scale_fill_manual(values = c("gray80", "skyblue"))+
  xlab("Simulated response value")+
  ylab("Density")+
  theme_bw()
```



Wow, this looks way different, why might that be? Well, the mean response values in the prior plot represent the mean expected value. It would not be 0 for this combination of covariates, which regularly results in positive catches, and is much less variable since it represents the mean expectation. The latter plot includes actual simulations of data points, which are often 0 despite the favorable conditions, but when non-zero, they can be very large numbers. The large numbers contribute to dragging the mean response value far from 0, as you see in the former plot. You can see this convergence by comparing the mean values of the predictions from each method, which are very close with the mean of the mean response values at 724 and the mean of the simulated response values at 707