Model Selection and Creation

Matt Clark

May 15, 2018

## Load all required packages

Note: rstan and rstanarm require a c++ compiler to run

[Download the C++ compiler here](https://github.com/stan-dev/rstan/wiki/Installing-RStan-on-Windows)

library(Metrics)

## Warning: package 'Metrics' was built under R version 3.4.4

library(readr)  
Data <- read\_csv("~/ParkBreak/Seasonal.csv")

## Warning: Missing column names filled in: 'X1' [1]

## Parsed with column specification:  
## cols(  
## X1 = col\_integer(),  
## Year = col\_integer(),  
## Season = col\_character(),  
## SeasVis = col\_integer(),  
## MedGas = col\_double(),  
## MedGoogle = col\_integer(),  
## GoogStag = col\_integer(),  
## Winter = col\_integer(),  
## Fall = col\_integer(),  
## Summer = col\_integer(),  
## Spring = col\_integer(),  
## Overnight = col\_integer(),  
## MedCCI = col\_double(),  
## GasStag = col\_double(),  
## CCIStag = col\_double()  
## )

View(Data)  
library(rethinking)

## Loading required package: rstan

## Loading required package: ggplot2

## Loading required package: StanHeaders

## rstan (Version 2.17.3, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling  
## options(mc.cores = parallel::detectCores()).  
## To avoid recompilation of unchanged Stan programs, we recommend calling  
## rstan\_options(auto\_write = TRUE)

## Loading required package: parallel

## rethinking (Version 1.59)

##   
## Attaching package: 'rethinking'

## The following object is masked from 'package:Metrics':  
##   
## se

library(shiny)  
library(rstanarm)

## Loading required package: Rcpp

## rstanarm (Version 2.17.3, packaged: 2018-02-17 05:11:16 UTC)

## - Do not expect the default priors to remain the same in future rstanarm versions.

## Thus, R scripts should specify priors explicitly, even if they are just the defaults.

## - For execution on a local, multicore CPU with excess RAM we recommend calling

## options(mc.cores = parallel::detectCores())

## - Plotting theme set to bayesplot::theme\_default().

##   
## Attaching package: 'rstanarm'

## The following object is masked from 'package:rethinking':  
##   
## se

## The following object is masked from 'package:Metrics':  
##   
## se

library(rstan)  
library(MASS)  
library(ggplot2)  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following object is masked from 'package:MASS':  
##   
## select

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(tidyr)

##   
## Attaching package: 'tidyr'

## The following object is masked from 'package:rstan':  
##   
## extract

First, we will create frequentist models to check for significance and do leave-one-out cross validation (LOOCV) of our models.

The first step in model creation is to standardize all of our variables so that the slopes are comparable.

stdize<-function(x) {return((x-mean(x)/(2\*sd(x))))}  
Data$stdcci<-stdize(Data$CCIStag)  
Data$stdgoog<-stdize(Data$GoogStag)

Now create the different models Note: only the significant variables are displayed here…this is to save you from running the 50+ models that were required for parameter elimination.

M1<-glm.nb(SeasVis~stdcci,data=Data)  
M2<-glm.nb(SeasVis~stdgoog,data=Data)  
M3<-glm.nb(SeasVis~Season,data=Data)  
M4<-glm.nb(SeasVis~stdcci+stdgoog,data=Data)  
M5<-glm.nb(SeasVis~stdcci+Season,data=Data)  
M6<-glm.nb(SeasVis~stdgoog+Season,data=Data)  
M7<-glm.nb(SeasVis~stdgoog+Season+stdcci,data=Data)  
M8<-glm(SeasVis~stdgoog+Season+stdcci,family="poisson",data=Data)

Let’s run the LOOCV for the negative binomial and Poisson distributions

leftout<-rep(NA,times=length(Data$SeasVis))  
for(i in 1:length(Data$SeasVis)){  
 sub\_dat<-Data[-i,]  
 m\_sub<-glm.nb(SeasVis~stdcci+stdgoog+Season,data=Data)  
 leftout[i]=predict(m\_sub,newdat=Data[i,])   
}  
  
R2 <- function (x, y) cor(x, y) ^ 2  
R2mod<-R2(exp(leftout),Data$SeasVis)  
library('Metrics')  
rmsemod<-rmse(leftout,Data$SeasVis)  
R2mod

## [1] 0.9255936

rmsemod

## [1] 471637.3

leftout<-rep(NA,times=length(Data$SeasVis))  
for(i in 1:length(Data$SeasVis)){  
 sub\_dat<-Data[-i,]  
 m\_sub<-glm(SeasVis~stdcci+stdgoog+Season,family="poisson",data=Data)  
 leftout[i]=predict(m\_sub,newdat=Data[i,])   
}  
  
R2mod<-R2(exp(leftout),Data$SeasVis)  
rmsemod<-rmse(leftout,Data$SeasVis)  
R2mod

## [1] 0.9271636

rmsemod

## [1] 471637.3

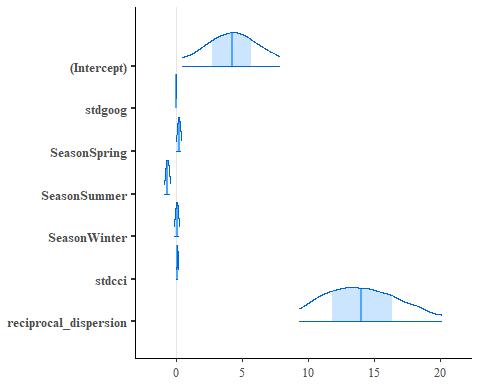
Cool, now that we know that the negative binomial model affords us the greatest out of sample predictive power, let’s run this model in a Bayesian context.

fcmod<-stan\_glm.nb(SeasVis~stdgoog+Season+stdcci,data=Data)

##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 1).  
##   
## Gradient evaluation took 0 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.529 seconds (Warm-up)  
## 0.546 seconds (Sampling)  
## 1.075 seconds (Total)  
##   
##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 2).  
##   
## Gradient evaluation took 0 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.55 seconds (Warm-up)  
## 0.407 seconds (Sampling)  
## 0.957 seconds (Total)  
##   
##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 3).  
##   
## Gradient evaluation took 0 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.522 seconds (Warm-up)  
## 0.487 seconds (Sampling)  
## 1.009 seconds (Total)  
##   
##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 4).  
##   
## Gradient evaluation took 0 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.571 seconds (Warm-up)  
## 0.482 seconds (Sampling)  
## 1.053 seconds (Total)

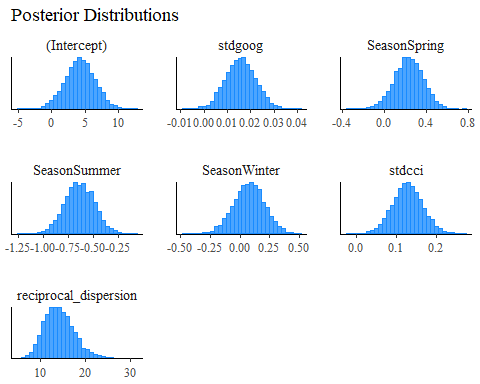
Great, now we have our Bayesian model, lets examine the posterior distributions and check out some plots.

bayesplot::color\_scheme\_set("brightblue")  
plot(fcmod, "areas",prob = 0.5, prob\_outer = 0.9)

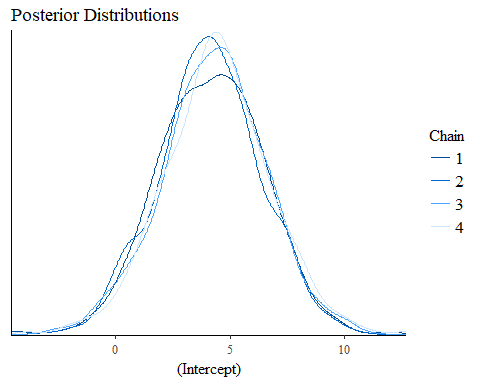


plot\_title <- ggplot2::ggtitle("Posterior Distributions")  
plot(fcmod, "hist") + plot\_title

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



plot(fcmod, "dens\_overlay", pars = "(Intercept)"  
) + plot\_title



These ^^ are useful, we can see that out chains are converging and out parameters look significant, lets look at the posterior distributions in a better way though.

sam<-as.matrix(fcmod)  
sam<-as.data.frame(sam)  
  
  
sam <- sam %>% gather(value, sam[1:8,])  
  
colnames(sam)[1] <- "Parameter"  
  
colnames(sam)[2]<-"Estimate"  
  
sam<-filter(sam, sam$Parameter!="(Intercept)")  
sam<-filter(sam, sam$Parameter!="reciprocal\_dispersion")  
  
library(ggjoy)

## Warning: package 'ggjoy' was built under R version 3.4.4

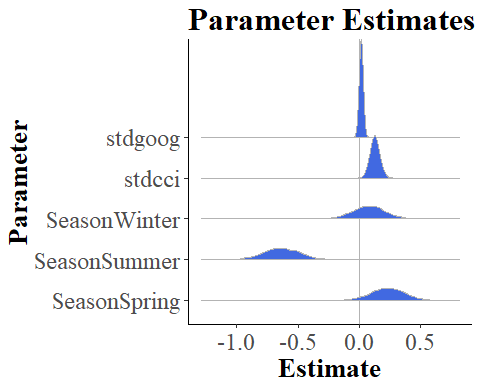
## Loading required package: ggridges

## Warning: package 'ggridges' was built under R version 3.4.4

## The ggjoy package has been deprecated. Please switch over to the  
## ggridges package, which provides the same functionality. Porting  
## guidelines can be found here:  
## https://github.com/clauswilke/ggjoy/blob/master/README.md

ggplot(sam, aes(x=Estimate, y=Parameter, height=..density..)) +  
 geom\_vline(xintercept = 0, col = "grey70") +  
 geom\_density\_ridges2(col = "grey70", fill = "royalblue", scale = 2.4) +  
 ggtitle("Parameter Estimates") +  
 theme(plot.title = element\_text(color="black", size=24, face="bold"))+  
 theme(axis.title.x = element\_text(size=20,face="bold"))+  
 theme(axis.title.y = element\_text(size=20,face="bold"))+  
 theme(axis.text = element\_text(size=18))+  
 theme(axis.text = element\_text(size=18))

## Picking joint bandwidth of 0.0157



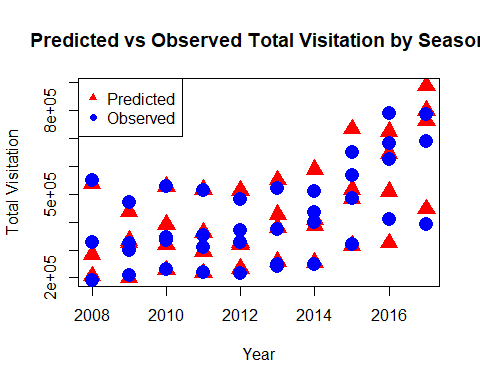
Cool, now lets look at our R^2 from our LOOCV and look at our predicted vs observed values over time.

We need to make a LOOCV data set for overnight visitation first

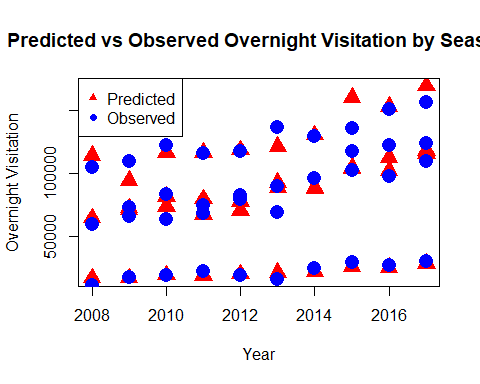
fcmodON<-glm.nb(Overnight~stdgoog+Season+stdcci,data=Data)  
leftout2<-rep(NA,times=length(Data$SeasVis))  
for(i in 1:length(Data$SeasVis)){  
 sub\_dat<-Data[-i,]  
 m\_sub<-glm.nb(Overnight~stdcci+stdgoog+Season,data=Data)  
 leftout2[i]=predict(m\_sub,newdat=Data[i,])   
}  
  
R2mod2<-R2(exp(leftout2),Data$Overnight)

Note: if your “leftout” is still set to a Poisson distribution, so back and set it to negative binomial

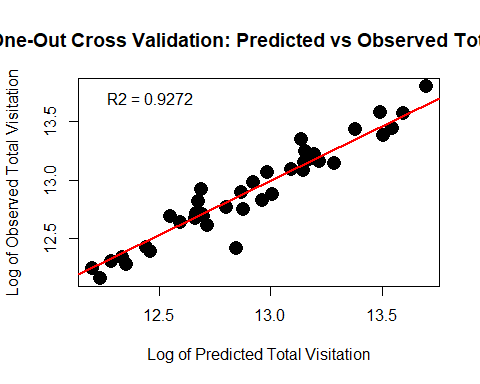
par(mfrow=c(1,1))  
  
plot(exp(leftout)~Data$Year,col="red", cex=2,pch=17, ylab="Total Visitation",xlab="Year",  
 main="Predicted vs Observed Total Visitation by Season")  
legend("topleft", pch=c(17,16), col=c("red", "blue"), c("Predicted", "Observed"))  
points(SeasVis~Year,data=Data,cex=2, pch=16, col="blue")



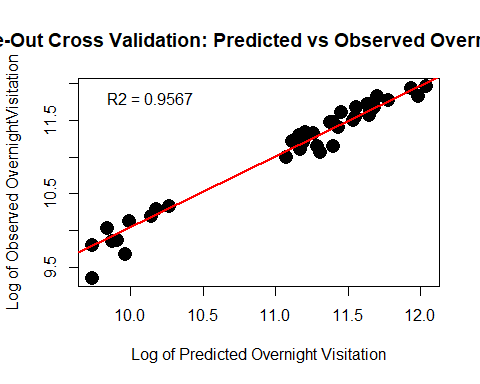
plot(exp(leftout2)~Data$Year,col="red", cex=2,pch=17, ylab="Overnight Visitation",xlab="Year",  
 main="Predicted vs Observed Overnight Visitation by Season")  
legend("topleft", pch=c(17,16), col=c("red", "blue"), c("Predicted", "Observed"))  
points(Overnight~Year,data=Data,cex=2, pch=16, col="blue")



plot(leftout,log(Data$SeasVis),pch=16,cex=2,main="Leave-One-Out Cross Validation: Predicted vs Observed Total Visitation",  
 xlab="Log of Predicted Total Visitation",ylab="Log of Observed Total Visitation")  
abline(lm(leftout~log(Data$SeasVis)),col="red",lwd=2)  
legend("topleft", bty="n", legend=paste("R2 =", format(R2mod, digits=4)))



plot(leftout2,log(Data$Overnight),pch=16,cex=2,main="Leave-One-Out Cross Validation: Predicted vs Observed Overnight Visitation",  
 xlab="Log of Predicted Overnight Visitation",ylab="Log of Observed OvernightVisitation")  
abline(lm(leftout2~log(Data$Overnight)),col="red",lwd=2)  
legend("topleft", bty="n", legend=paste("R2 =", format(R2mod2, digits=4)))



Last thing we will do is see how many of our data points fall within the 50% credibility interval that we use for the forecasting app. The first example is Total visitation, the second is Overnight visitation

n1<-25  
n2<-75  
  
fcmod<-stan\_glm.nb(SeasVis~GoogStag+CCIStag+Season,data=Data)

##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 1).  
##   
## Gradient evaluation took 0 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.526 seconds (Warm-up)  
## 0.522 seconds (Sampling)  
## 1.048 seconds (Total)  
##   
##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 2).  
##   
## Gradient evaluation took 0 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.567 seconds (Warm-up)  
## 0.466 seconds (Sampling)  
## 1.033 seconds (Total)  
##   
##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 3).  
##   
## Gradient evaluation took 0 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.59 seconds (Warm-up)  
## 0.499 seconds (Sampling)  
## 1.089 seconds (Total)  
##   
##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 4).  
##   
## Gradient evaluation took 0 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.526 seconds (Warm-up)  
## 0.528 seconds (Sampling)  
## 1.054 seconds (Total)

leftout<-rep(NA,times=length(Data$SeasVis))  
for(i in 1:length(Data$SeasVis)){  
 df <- data.frame(CCIStag=Data$CCIStag[i],  
 GoogStag=Data$GoogStag[i],  
 Season=Data$Season[i])  
 mod<-(posterior\_predict(  
 fcmod,df,draws=4000))  
 upbound<-min(mod[mod > quantile(mod,prob=1-n1/100),])  
 lwrbound<-max(mod[mod < quantile(mod,prob=1-n2/100),])  
 inside<-sum(Data$SeasVis[i]>lwrbound & Data$SeasVis[i] < upbound)  
   
 leftout[i]= inside  
   
}  
totinside<-sum(leftout)  
percinside<-totinside/39  
percinside

## [1] 0.9230769

fcmodON<-stan\_glm.nb(Overnight~GoogStag+Season+CCIStag,data=Data)

##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 1).  
##   
## Gradient evaluation took 0 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.532 seconds (Warm-up)  
## 0.53 seconds (Sampling)  
## 1.062 seconds (Total)  
##   
##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 2).  
##   
## Gradient evaluation took 0 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.561 seconds (Warm-up)  
## 0.495 seconds (Sampling)  
## 1.056 seconds (Total)  
##   
##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 3).  
##   
## Gradient evaluation took 0 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.531 seconds (Warm-up)  
## 0.493 seconds (Sampling)  
## 1.024 seconds (Total)  
##   
##   
## SAMPLING FOR MODEL 'count' NOW (CHAIN 4).  
##   
## Gradient evaluation took 0.001 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 10 seconds.  
## Adjust your expectations accordingly!  
##   
##   
## Iteration: 1 / 2000 [ 0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##   
## Elapsed Time: 0.582 seconds (Warm-up)  
## 0.519 seconds (Sampling)  
## 1.101 seconds (Total)

leftout<-rep(NA,times=length(Data$SeasVis))  
for(i in 1:length(Data$SeasVis)){  
 df <- data.frame(CCIStag=Data$CCIStag[i],  
 GoogStag=Data$GoogStag[i],  
 Season=Data$Season[i])  
 mod<-(posterior\_predict(  
 fcmodON,df,draws=4000))  
 upbound<-min(mod[mod > quantile(mod,prob=1-n1/100),])  
 lwrbound<-max(mod[mod < quantile(mod,prob=1-n2/100),])  
 inside<-sum(Data$Overnight[i]>lwrbound & Data$Overnight[i] < upbound)  
   
 leftout[i]= inside  
   
}  
totinside<-sum(leftout)  
percinside<-totinside/39  
percinside

## [1] 0.8717949