Data 3 HW 4

Sean Duan

10/8/2020

1.

A

$$a_1 = \beta_0, b_1 = \beta_1, c_1 = \beta_2, d_1 = \beta_3$$

B

$$a_2 = \beta_0 - \beta_4\xi^3, b_2 = \beta_1 + 3\beta_4\xi^2, c_2 = \beta_2 - 3\beta_4\xi, d_2 = \beta_3 + \beta_4$$

C

$$f_1(\xi) = \beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3$$

$$f_2(\xi) = (\beta_0 - \beta_4\xi^3) + (\beta_1 + 3\beta_4\xi^2)\xi + (\beta_2 - 3\beta_4\xi)\xi^2 + (\beta_3 + \beta_4)\xi^3$$

$$\beta_0 - \beta_4\xi^3 + \beta_1\xi + 3\beta_4\xi^3 + \beta_2\xi^2 - 3\beta_4\xi^3 + \beta_3\xi^3 + \beta_4\xi^3$$
$$\beta_0 + \beta_1\xi + \beta_2\xi^2 + 3\beta_4\xi^3 - 3\beta_4\xi^3 + \beta_3\xi^3 + \beta_4\xi^3 - \beta_4\xi^3$$
$$\beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3$$

D

$$f'(x) = b_1 + 2c_1x + 3d_1x^2$$
$$f_1'(\xi) = \beta_1 + 2\beta_2\xi + 3\beta_3\xi^2$$

$$f_2'(\xi) = \beta_1 + 3\beta_4\xi^2 + 2(\beta_2 - 3\beta_4\xi)\xi + 3(\beta_3 + \beta_4)\xi^2$$

$$\beta_1 + 3\beta_4\xi^2 + 2\beta_2\xi - 6\beta_4\xi^2 + 3\beta_3\xi^2 + 3\beta_4\xi^2$$

$$\beta_1 + 2\beta_2\xi + 3\beta_3\xi^2 + 3\beta_4\xi^2 + 3\beta_4\xi^2 - 6\beta_4\xi^2$$

$$\beta_1 + 2\beta_2\xi + 3\beta_3\xi^2$$

E

$$f''(x) = 2c_1 + 6d_1x$$

$$f_1''(\xi) = 2\beta_2 + 6\beta_3\xi$$

$$f_2''(\xi) = 2(\beta_2 - 3\beta_4\xi) + 6(\beta_3 + \beta_4)\xi$$

$$2\beta_2 + 6\beta_3\xi$$

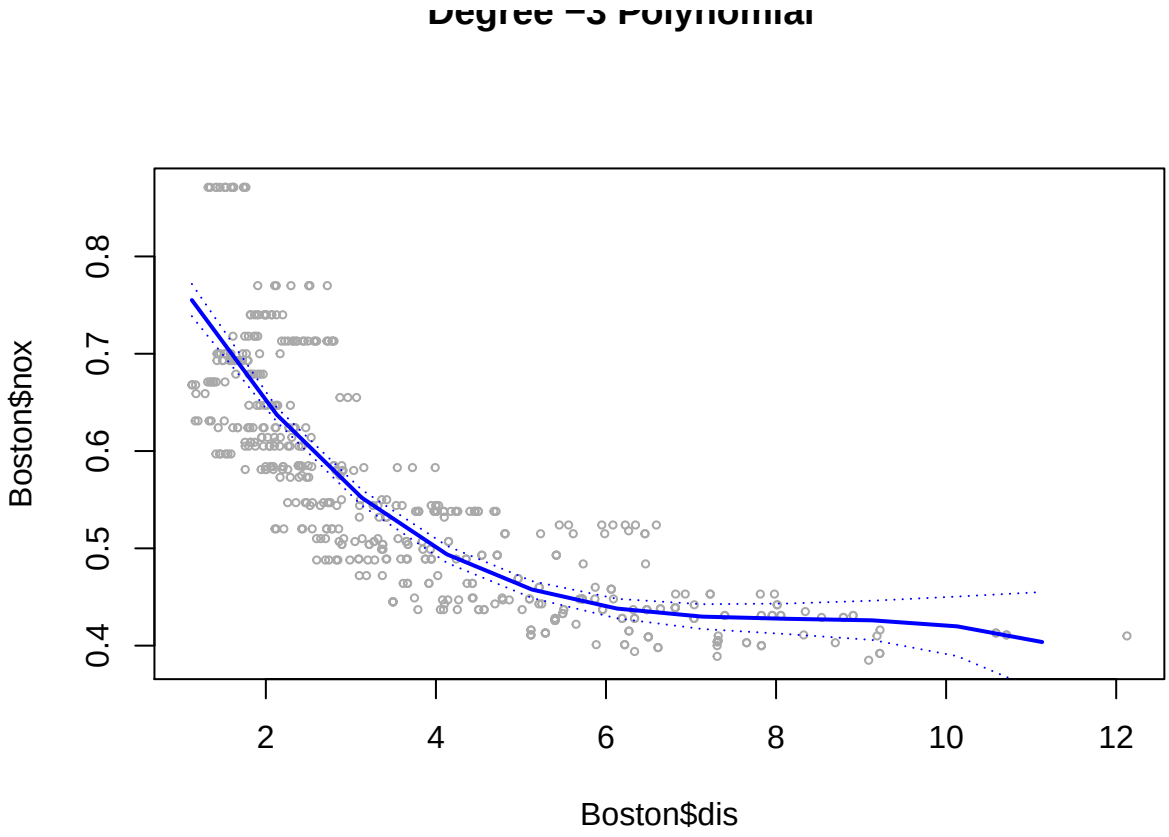
2.

A

```
data(Boston)
attach(Boston)
#polynomial reg
p2_m1<-lm(nox~poly(dis ,3) ,data=Boston)
#coef(summary (fit))
summary(p2_m1)
```

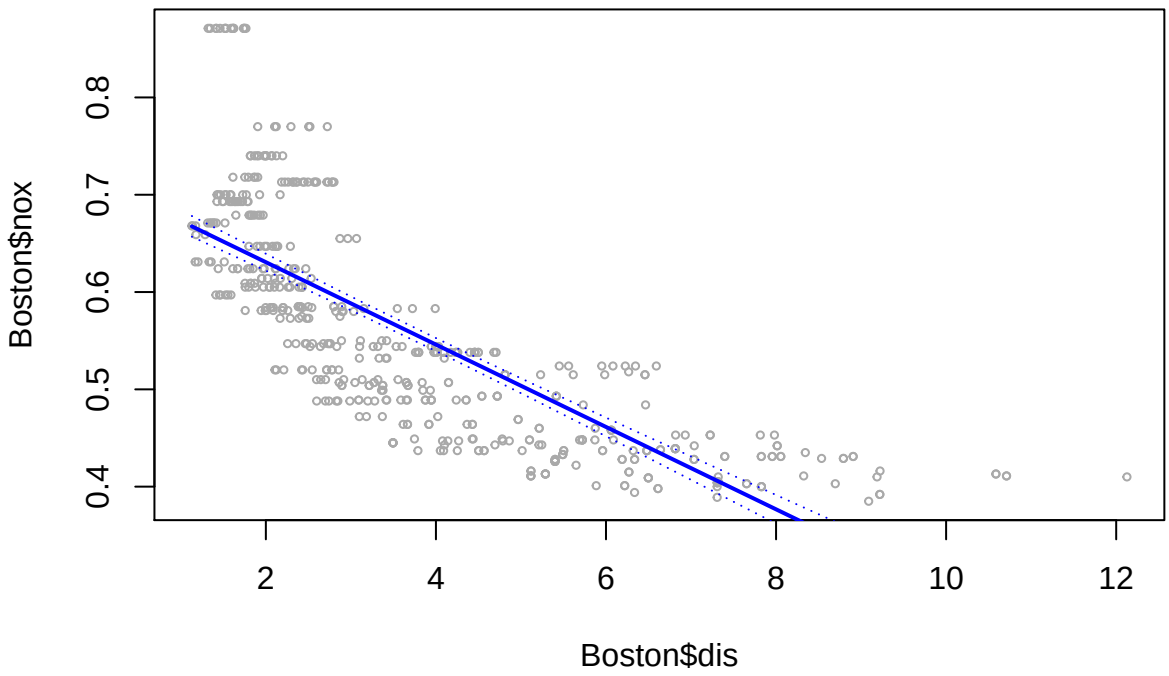
```
##
## Call:
## lm(formula = nox ~ poly(dis, 3), data = Boston)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.121130 -0.040619 -0.009738  0.023385  0.194904
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.554695   0.002759 201.021  < 2e-16 ***
## poly(dis, 3)1 -2.003096   0.062071 -32.271  < 2e-16 ***
## poly(dis, 3)2  0.856330   0.062071  13.796  < 2e-16 ***
## poly(dis, 3)3 -0.318049   0.062071  -5.124 4.27e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06207 on 502 degrees of freedom
## Multiple R-squared:  0.7148, Adjusted R-squared:  0.7131
## F-statistic: 419.3 on 3 and 502 DF,  p-value: < 2.2e-16
```

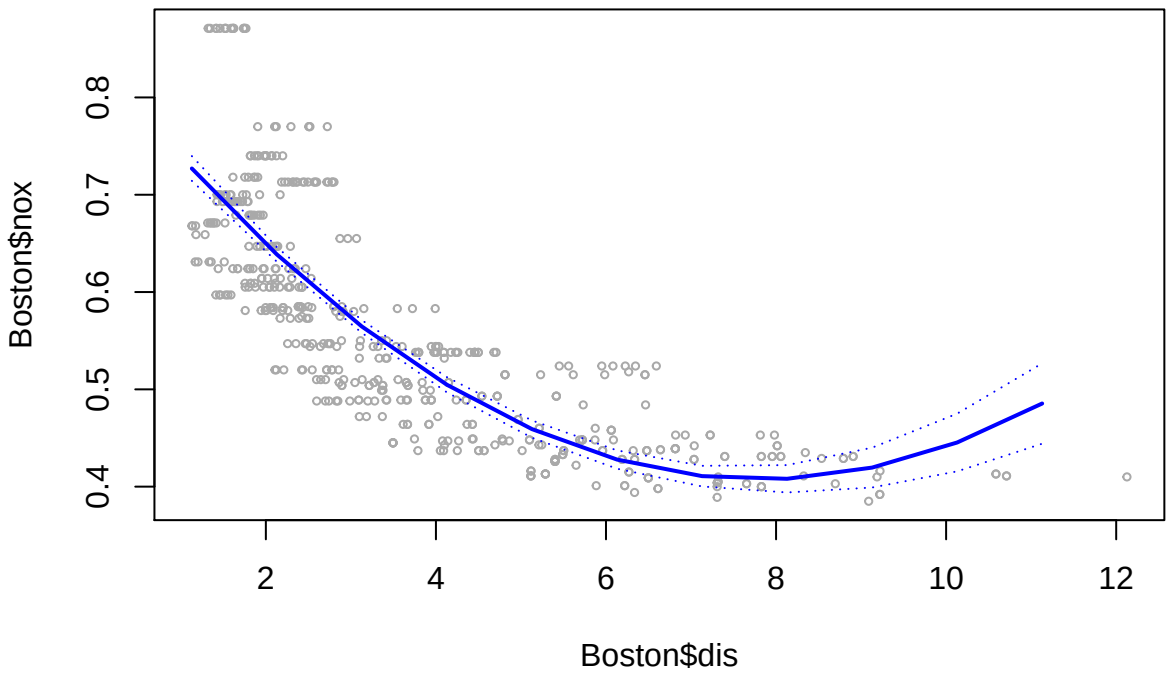
```
#code to plot, first make predictions from our range of data
dis_lims =range(Boston$dis)
dis_grid=seq(from=dis_lims [1],to=dis_lims [2])
dis_preds=predict (p2_m1 ,newdata =list(dis=dis_grid),se=TRUE)
dis_se.bands=cbind(dis_preds$fit +2* dis_preds$se.fit ,dis_preds$fit -2* dis_preds$se.fit)
#actual plotting code
plot(Boston$dis ,Boston$nox, xlim=dis_lims ,cex =.5,col=" darkgrey ")
title(" Degree -3 Polynomial ",outer=T)
lines(dis_grid ,dis_preds$fit ,lwd=2,col="blue")
matlines (dis_grid ,dis_se.bands ,lwd=1, col=" blue",lty=3)
```

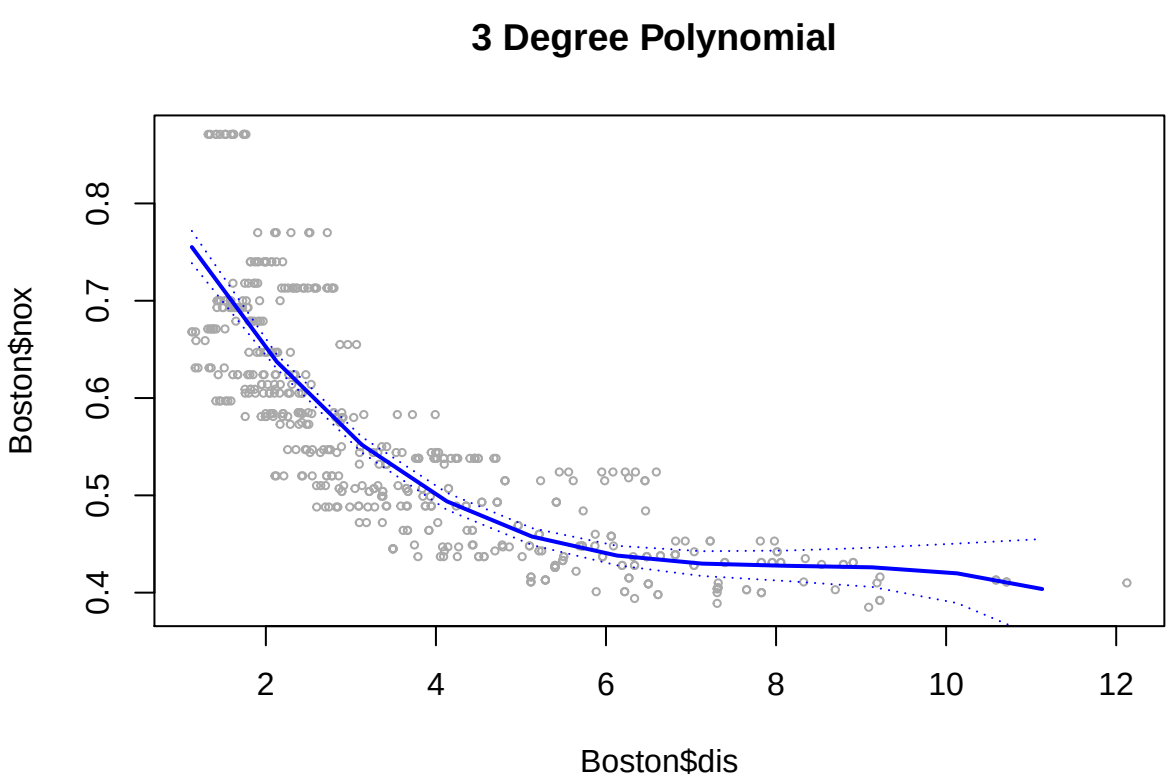


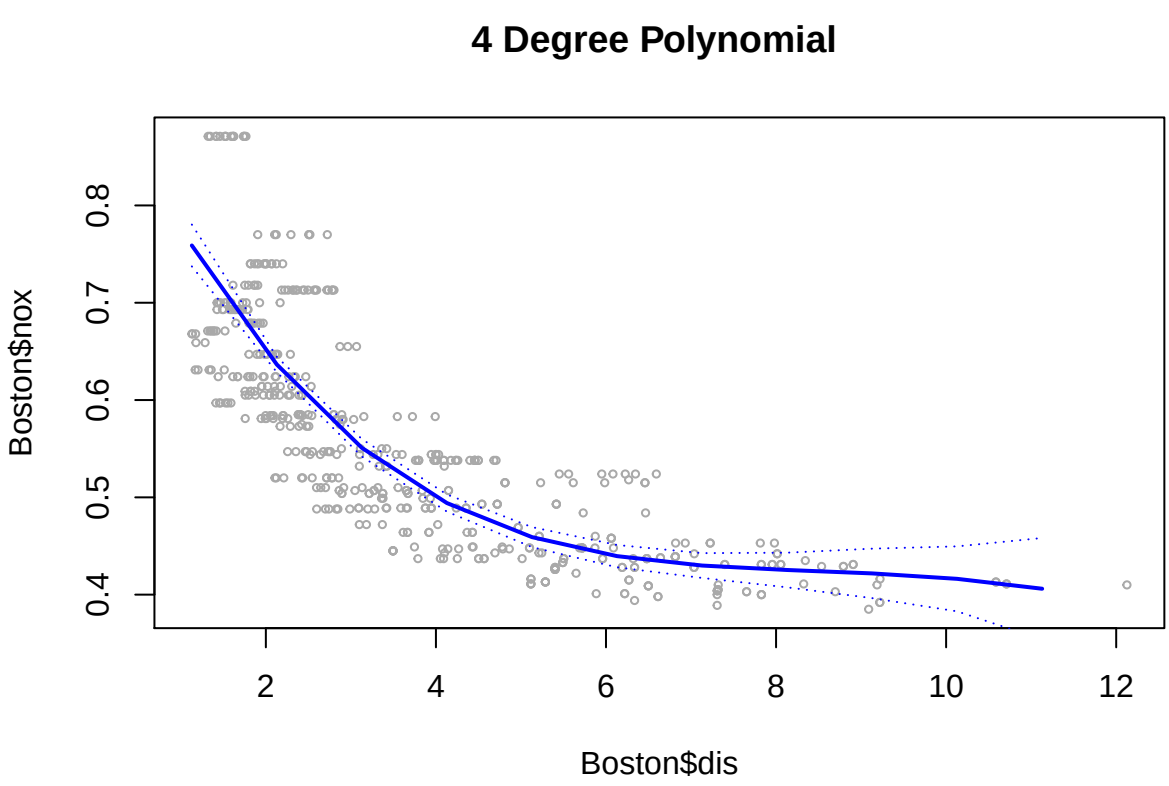
B

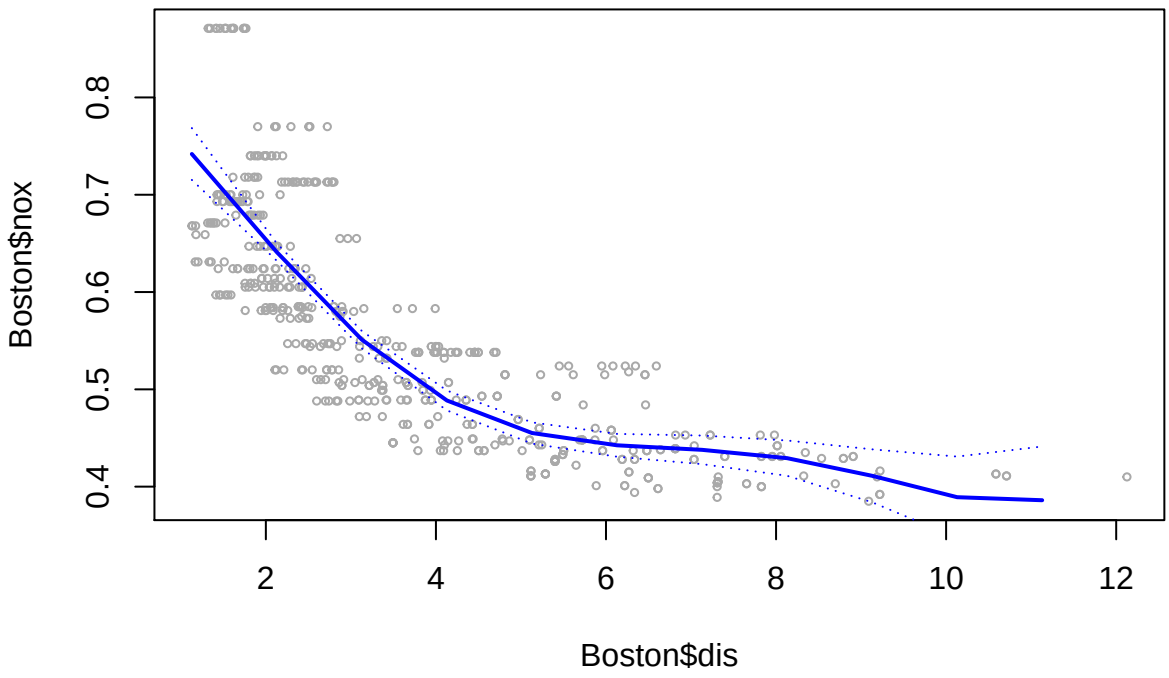
```
#make a loop to fit all the polys
testlist<-list(NA)
for(i in 1:10){
  testlist[[i]]<-lm(nox~poly(dis ,i) ,data=Boston)
}

#actual plotting code
for(i in 1:10){
  dis_preds=predict (testlist[[i]] ,newdata =list(dis=dis_grid),se=TRUE)
  dis_se.bands=cbind(dis_preds$fit +2* dis_preds$se.fit ,dis_preds$fit -2* dis_preds$se.fit)
  plot(Boston$dis ,Boston$nox, xlim=dis_lims ,cex =.5,col=" darkgrey ")
  lines(dis_grid ,dis_preds$fit ,lwd=2,col="blue")
  title(paste(i, "Degree Polynomial "))
  matlines (dis_grid ,dis_se.bands ,lwd=1, col=" blue",lty=3)
}
```

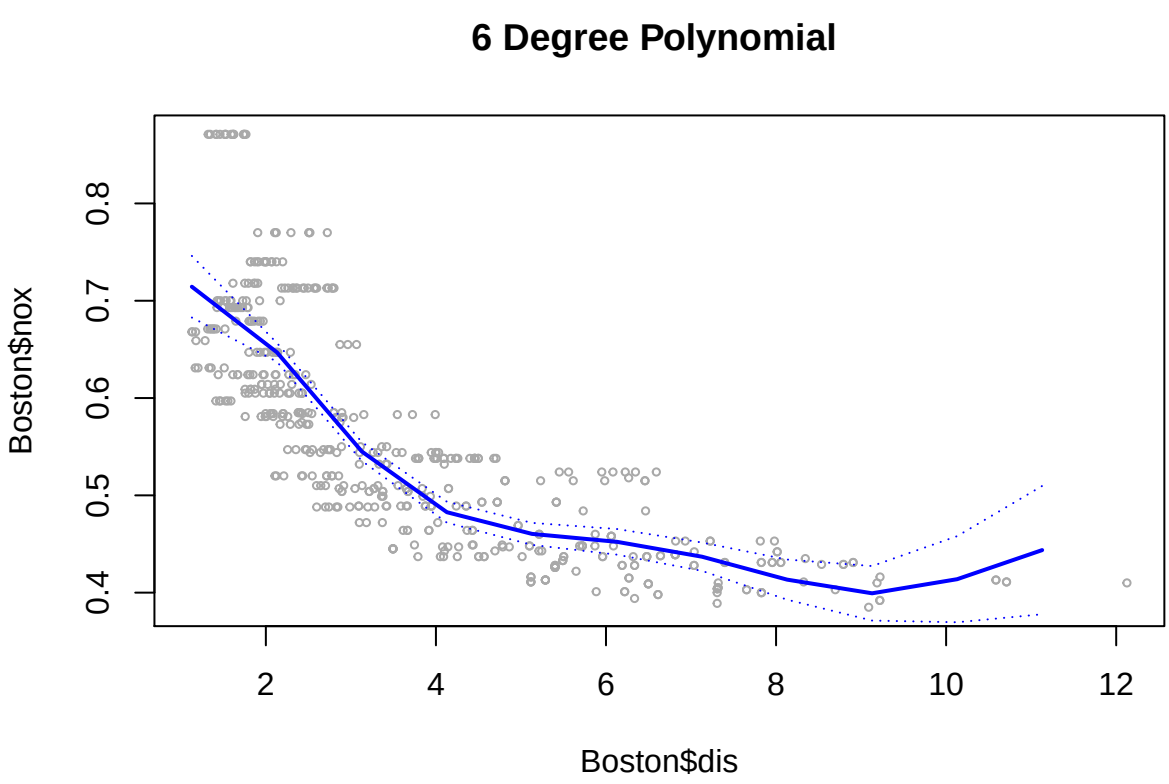

1 Degree Polynomial
Boston$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Boston$dis

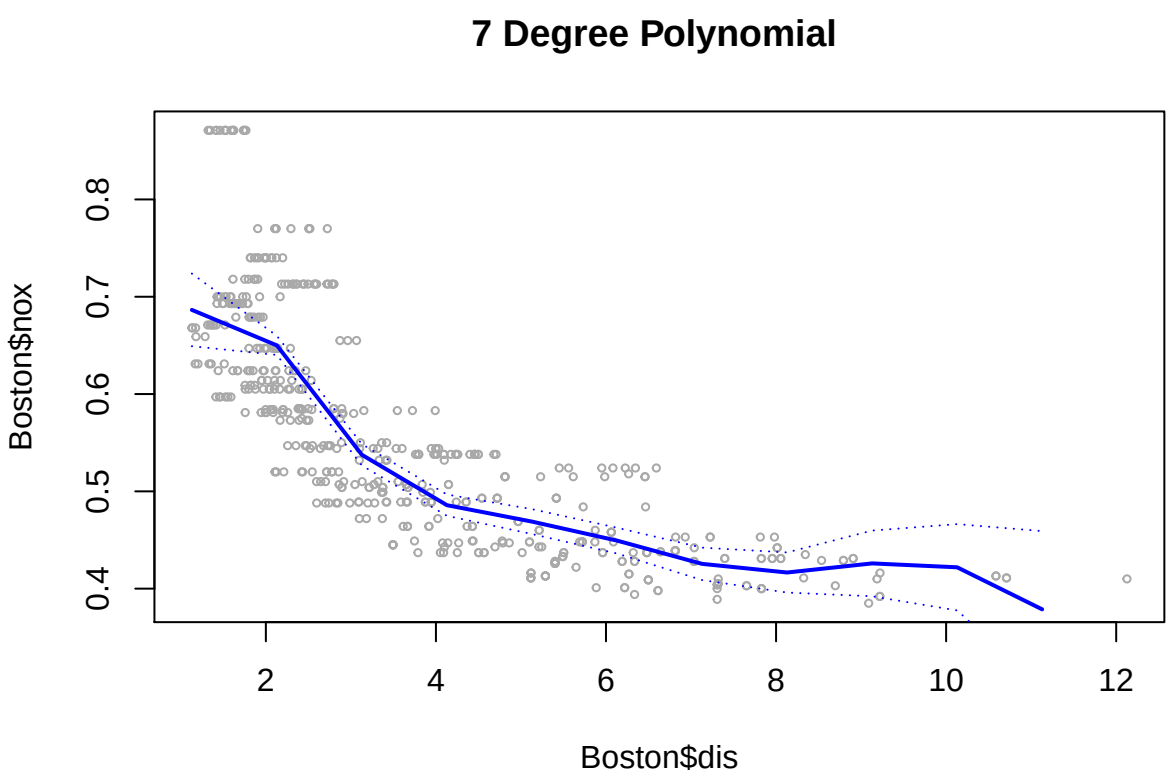

2 Degree Polynomial
Boston$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Boston$dis

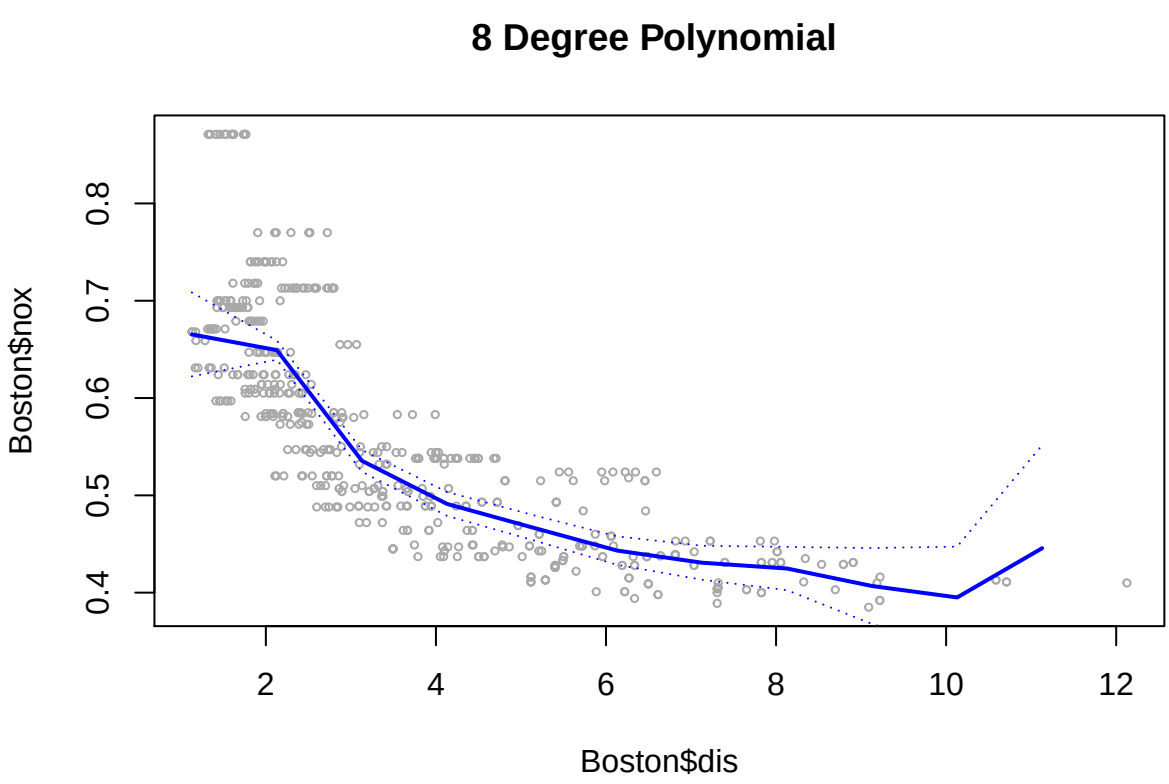

3 Degree Polynomial
Boston$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Boston$dis

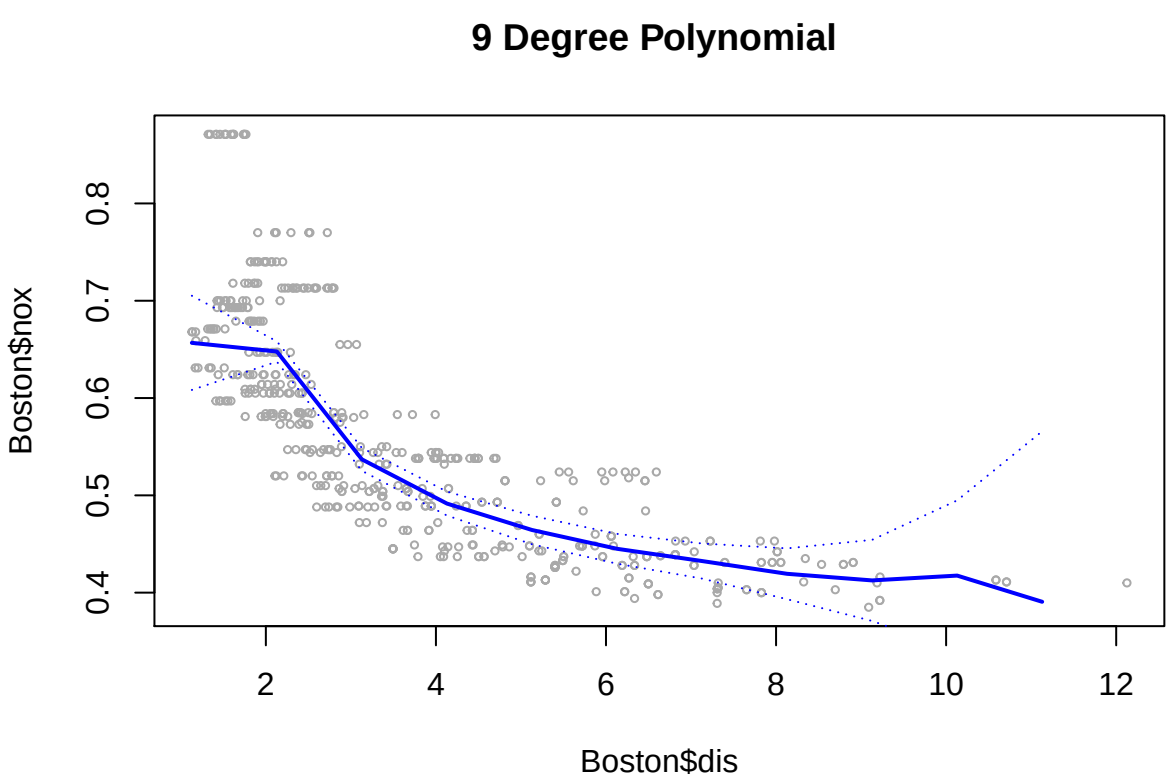

4 Degree Polynomial
Boston$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Boston$dis

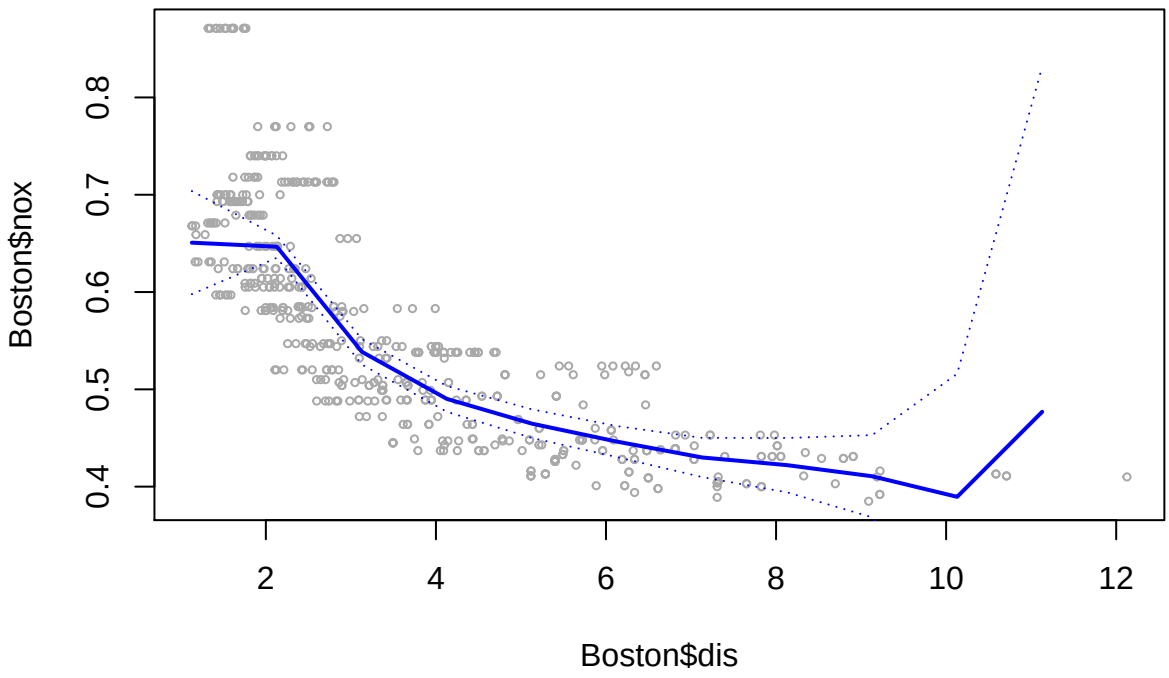

5 Degree Polynomial
Boston$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Boston$dis


6 Degree Polynomial
Boston$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Boston$dis


7 Degree Polynomial
Boston$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Boston$dis


8 Degree Polynomial
Boston$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Boston$dis


9 Degree Polynomial
Boston$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Boston$dis




```
#print the deviance (same as RSS for linear models) on all polynomial fits
for(i in 1:10){
  print(deviance(testlist[[i]]))
}
```

```
## [1] 2.768563
## [1] 2.035262
## [1] 1.934107
## [1] 1.932981
## [1] 1.91529
## [1] 1.878257
## [1] 1.849484
## [1] 1.83563
## [1] 1.833331
## [1] 1.832171
```

C

```
#setting up for CV
train=sample (506,253)
test=!train
#fitting the loop
testlist<-list(NA)
for(i in 1:10){
  testlist[[i]]<-lm(nox~poly(dis ,i) ,data=Boston, subset=train)
}
```

```
for(i in 1:10){
print(mean((Boston$nox-predict(testlist[[i]],Boston))[-train ]^2))
}
```

```
## [1] 0.005162788
## [1] 0.003800154
## [1] 0.003692687
## [1] 0.003688285
## [1] 0.003701235
## [1] 0.003679447
## [1] 0.003670083
## [1] 0.003643375
## [1] 0.003639511
## [1] 0.003638563
```

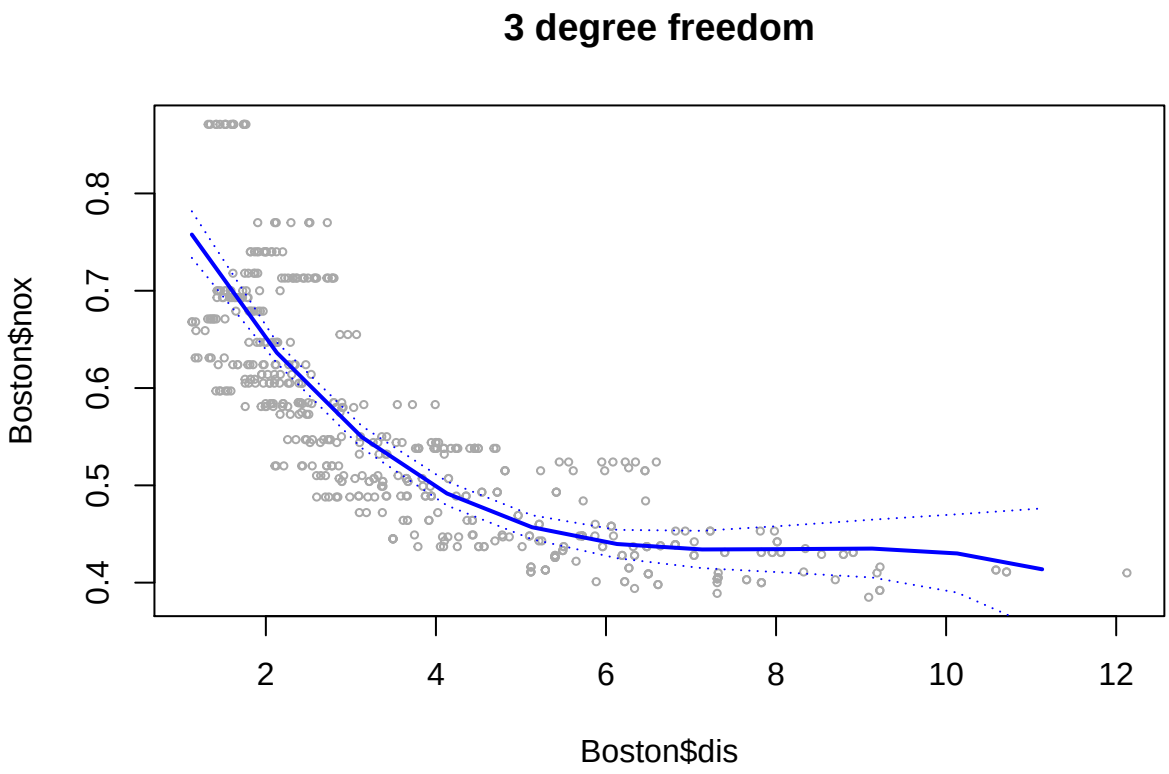
Cross validation using an randomly sampled, equivalent sized training and test set indicated that the 3rd polynomial model has the lowest MSE. Thus, 3 degrees is our optimal degree for the polynomial.

D

```
testlist<-list(NA)
for(i in 3:6){
  testlist[[i]]<-lm(nox~bs(dis ,df = i) ,data=Boston, subset=train)
}

for(i in 3:6){
  print(mean((Boston$nox-predict(testlist[[i]],Boston))[-train ]^2))
}
```

```
## [1] 0.003692687
## [1] 0.003712239
## [1] 0.003639664
## [1] 0.003630313
```

```
#code to plot, first make predictions from our range of data
dis_lims =range(Boston$dis)
dis_grid=seq(from=dis_lims [1],to=dis_lims [2])
dis_preds=predict (testlist[[3]] ,newdata =list(dis=dis_grid),se=TRUE)
dis_se.bands=cbind(dis_preds$fit +2* dis_preds$se.fit ,dis_preds$fit -2* dis_preds$se.fit)
#actual plotting code
plot(Boston$dis ,Boston$nox, xlim=dis_lims ,cex =.5,col=" darkgrey ")
title("3 degree freedom")
lines(dis_grid ,dis_preds$fit ,lwd=2,col="blue")
matlines (dis_grid ,dis_se.bands ,lwd=1, col=" blue",lty=3)
```



The best model has 3 degrees of freedom. RSS is 0.003891.

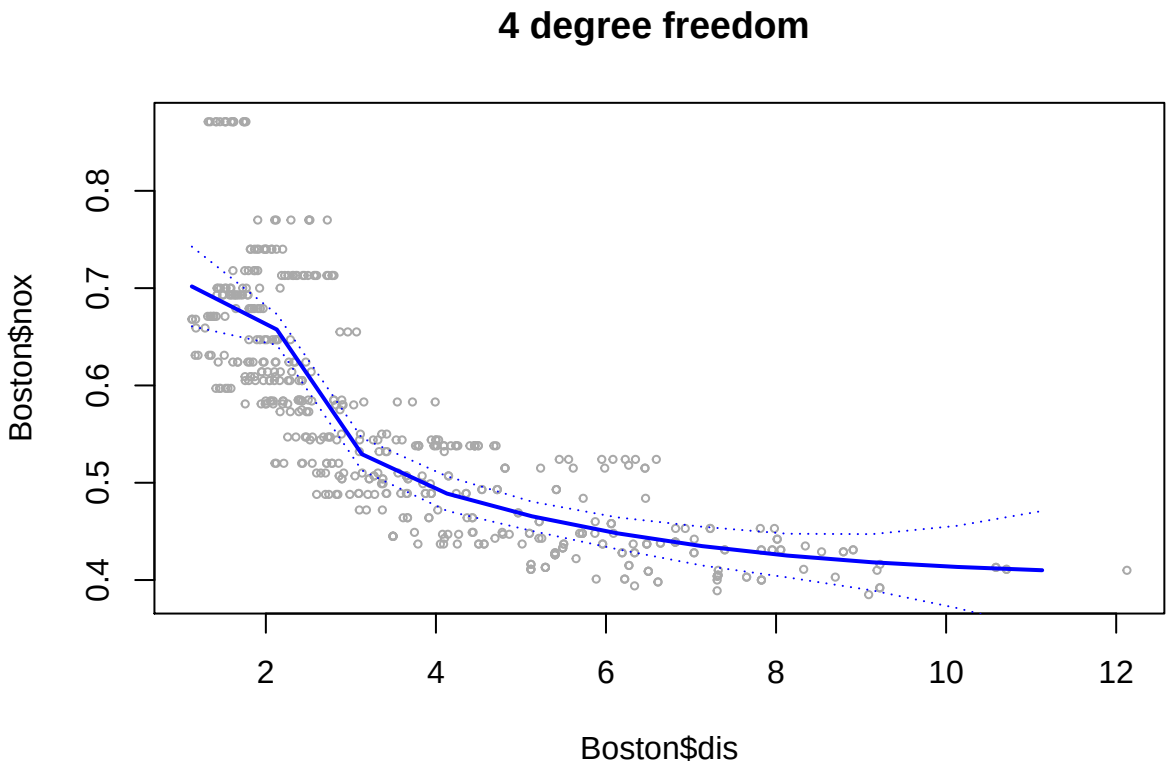
E

```
testlist<-list(NA)
for(i in 3:6){
  testlist[[i]]<-lm(nox~ns(dis ,df = i) ,data=Boston, subset=train)
}

for(i in 3:6){
  print(mean((Boston$nox-predict(testlist[[i]],Boston))[-train ]^2))
}
```

```
## [1] 0.00367774
## [1] 0.003698302
## [1] 0.003660441
## [1] 0.003670717
```

```
#5th df model best
#code to plot, first make predictions from our range of data
dis_lims =range(Boston$dis)
dis_grid=seq(from=dis_lims [1],to=dis_lims [2])
dis_preds=predict (testlist[[5]] ,newdata =list(dis=dis_grid),se=TRUE)
dis_se.bands=cbind(dis_preds$fit +2* dis_preds$se.fit ,dis_preds$fit -2* dis_preds$se.fit)
#actual plotting code
plot(Boston$dis ,Boston$nox, xlim=dis_lims ,cex =.5,col=" darkgrey ")
title(" 4 degree freedom  ")
```

```
lines(dis_grid ,dis_preds$fit ,lwd=2,col="blue")
matlines (dis_grid ,dis_se.bands ,lwd=1, col=" blue",lty=3)
```



5 degree of freedom model is best, the RSS is 0.00385.

F

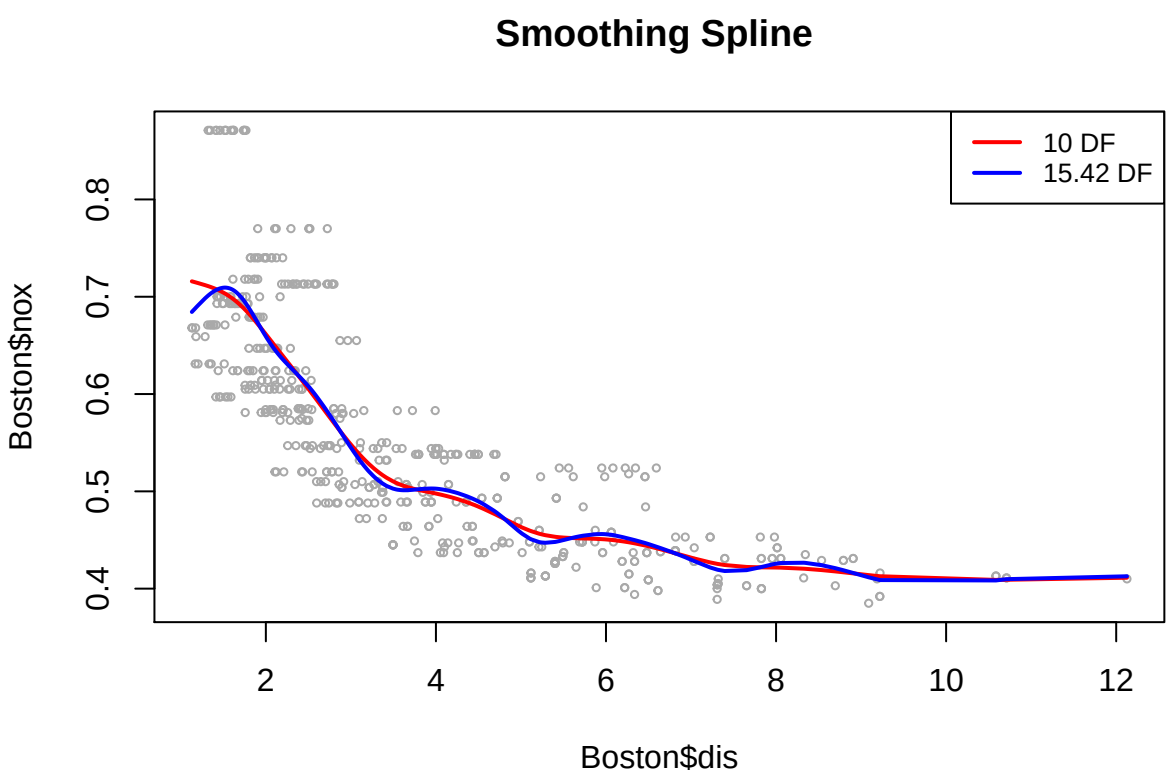
```
#fitting a smoothing spline using the smooth.spline fxn
plot(x=Boston$dis ,y=Boston$nox ,xlim=dis_lims ,cex =.5,col="darkgrey")
title("Smoothing Spline ")
fit=smooth.spline(Boston$dis ,Boston$nox ,df=10)
fit2=smooth.spline(Boston$dis ,Boston$nox ,cv=TRUE)
```

```
## Warning in smooth.spline(Boston$dis, Boston$nox, cv = TRUE): cross-validation
## with non-unique 'x' values seems doubtful
```

```
fit2$df
```

```
## [1] 15.42984
```

```
lines(fit ,col="red",lwd =2)
lines(fit2 ,col="blue",lwd=2)
legend ("topright",legend=c("10 DF" ,"15.42 DF"),col=c("red","blue"),lty=1,lwd=2, cex =.8)
```




```
fit2$lambda
```

```
## [1] 9.029534e-05
```

```
sum((fit2$y - Boston$nox[1:412])^2)
```

```
## [1] 11.81965
```

Df chosen was approximately 15.4. Lambda is 9.0295e-05, RSS is 11.819.

G

```
#kfold code from wikle
#modfied to just a plain test/train setup
rangespan<-seq(from=0.1, to=5, length.out = 20)
check<-list(NA)
testlist<-list(NA)
for(i in 1:length(rangespan)){
  testlist[[i]]<-loess(nox ~ dis, span = rangespan[[i]], data = Boston, subset = train, control = loess
  predict_list<-predict(testlist[[i]], data.frame(dis=Boston$dis[-train]))
  check[[i]]<-sum((predict_list-Boston$nox[-train])^2, na.rm = TRUE)
}
which.min(check)
```

```
## [1] 2
```

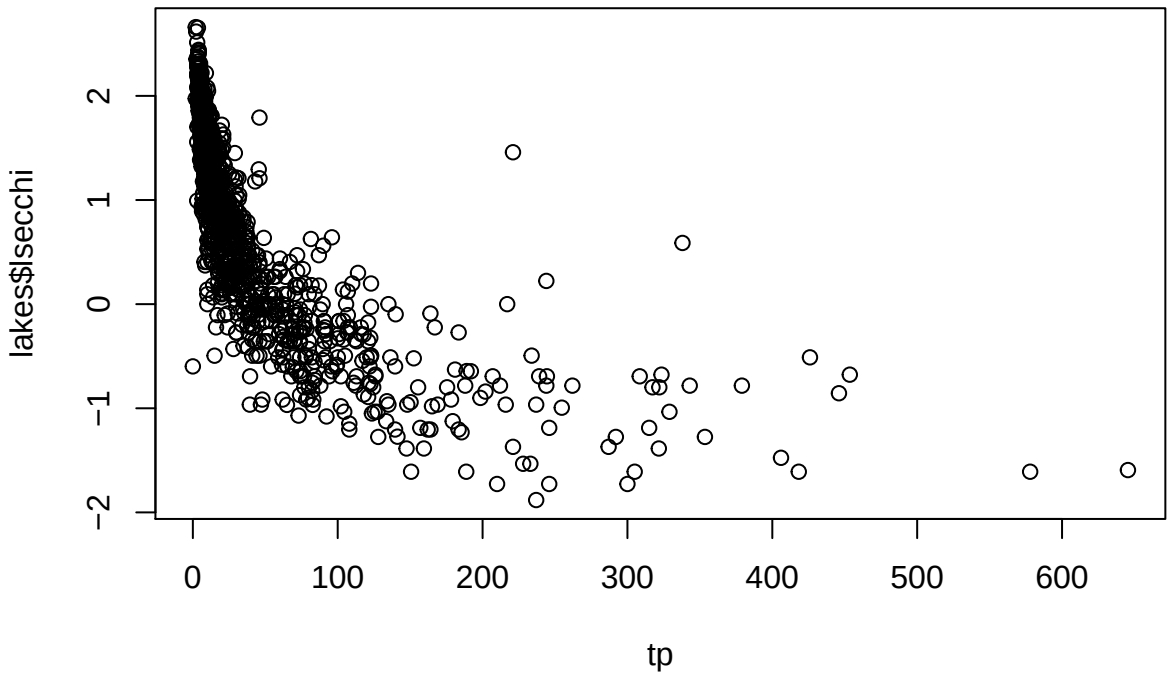
```
rangespan[which.min(check)]
```

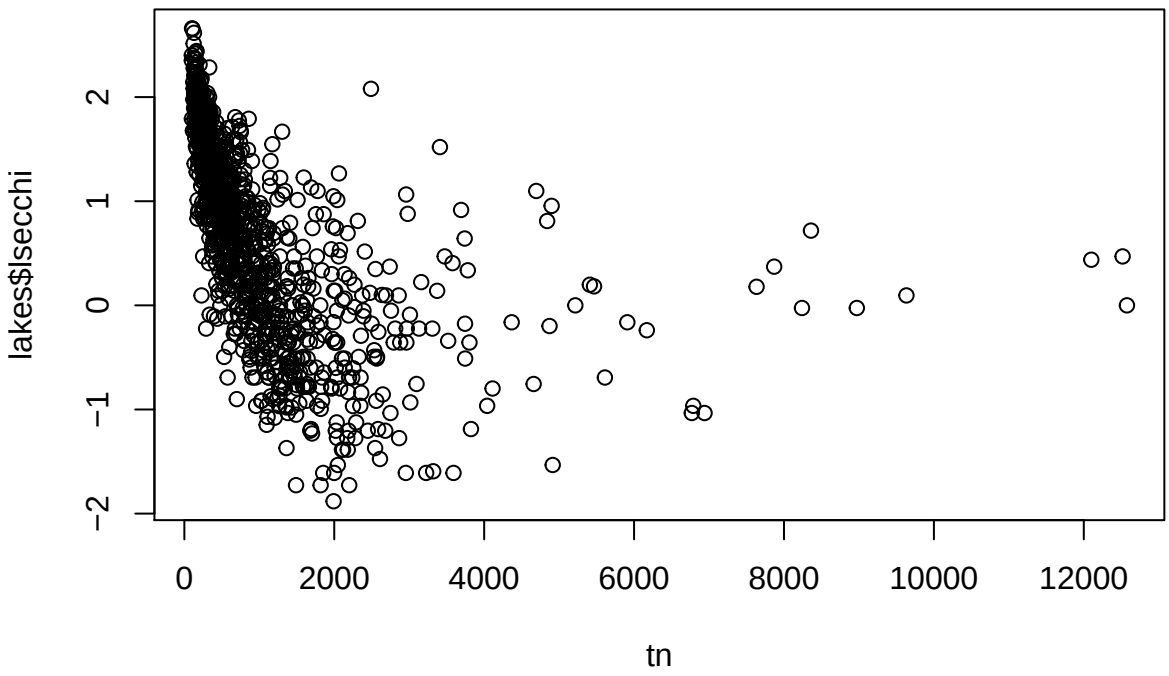
```
## [1] 0.3578947
```

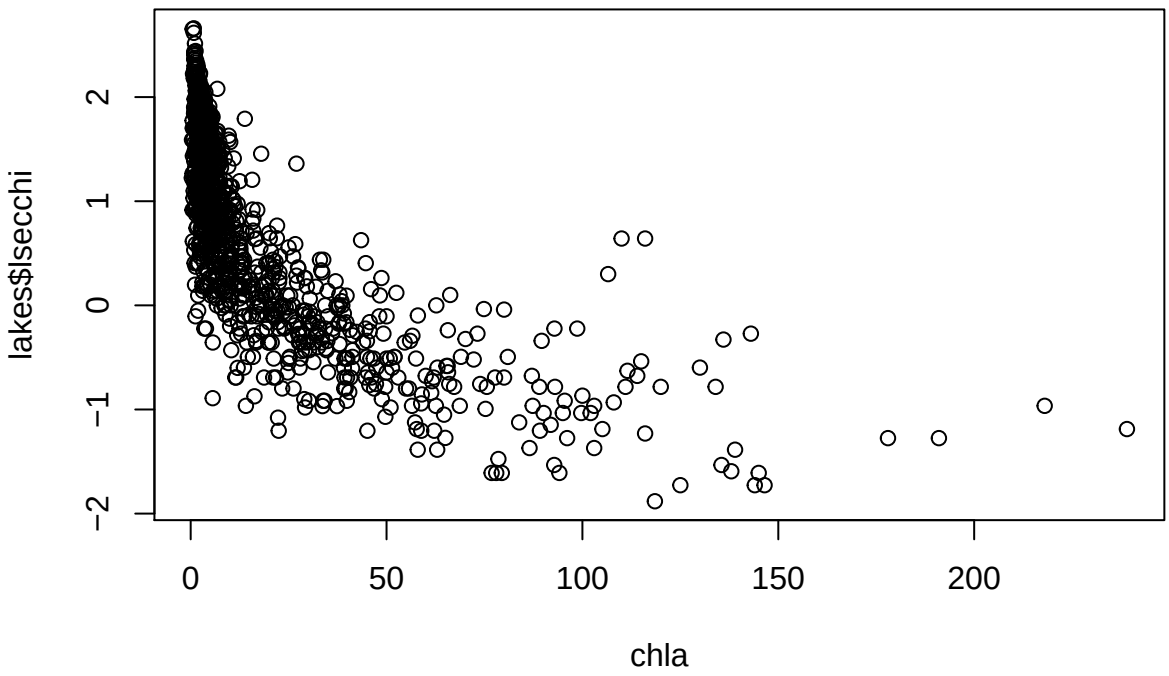
We chose the span by setting a training and testing set aside at random, then running cross validation on a grid search for values of span from .1 to 5. The span that resulted in the lowest SSE was chosen, which in this case was a span of 0.6157.

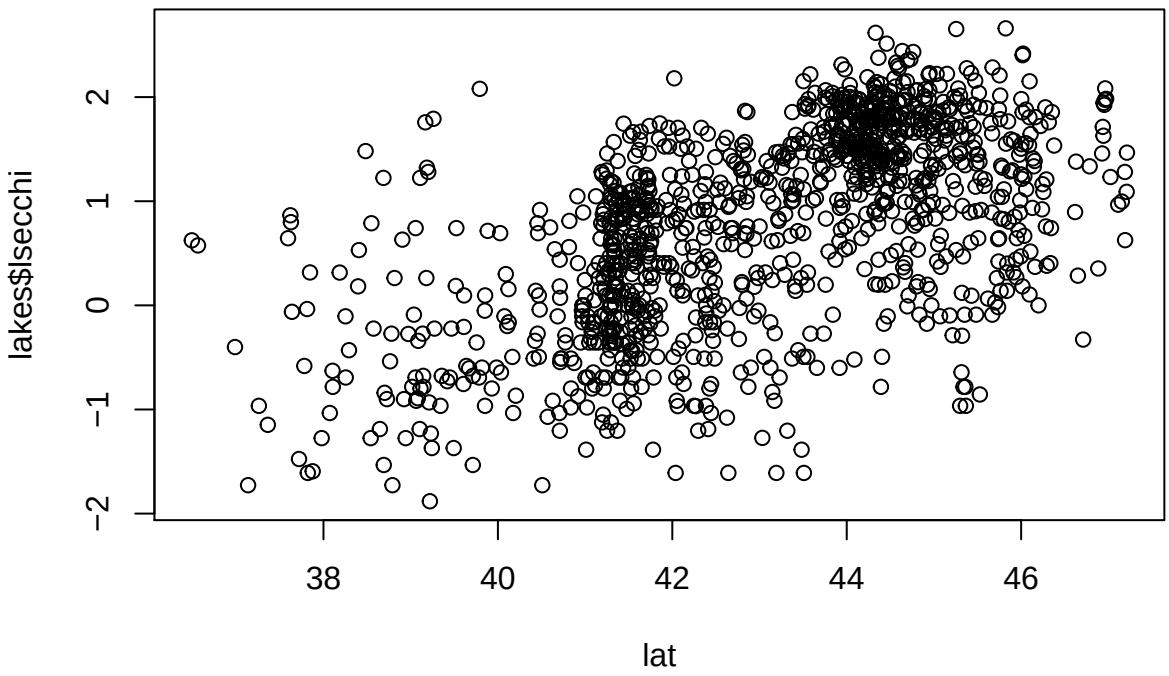
3.

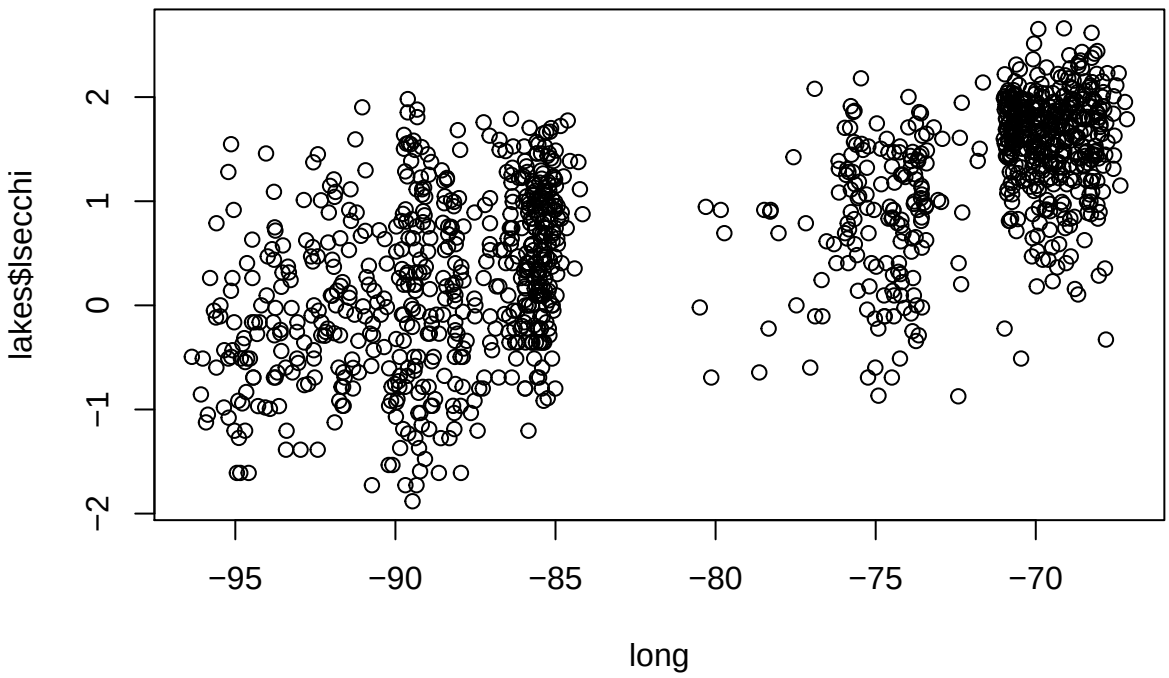
```
load("lakes_DA3.Rdata")
lakes<-lakes_DA3
lakes$lsecchi<-log(lakes$secchi)
lakes2<-lakes[,-c(4,25)]

for(i in 1:23){
  plot(y=lakes$lsecchi, x=lakes2[,i], xlab=colnames(lakes2)[i])
}
```

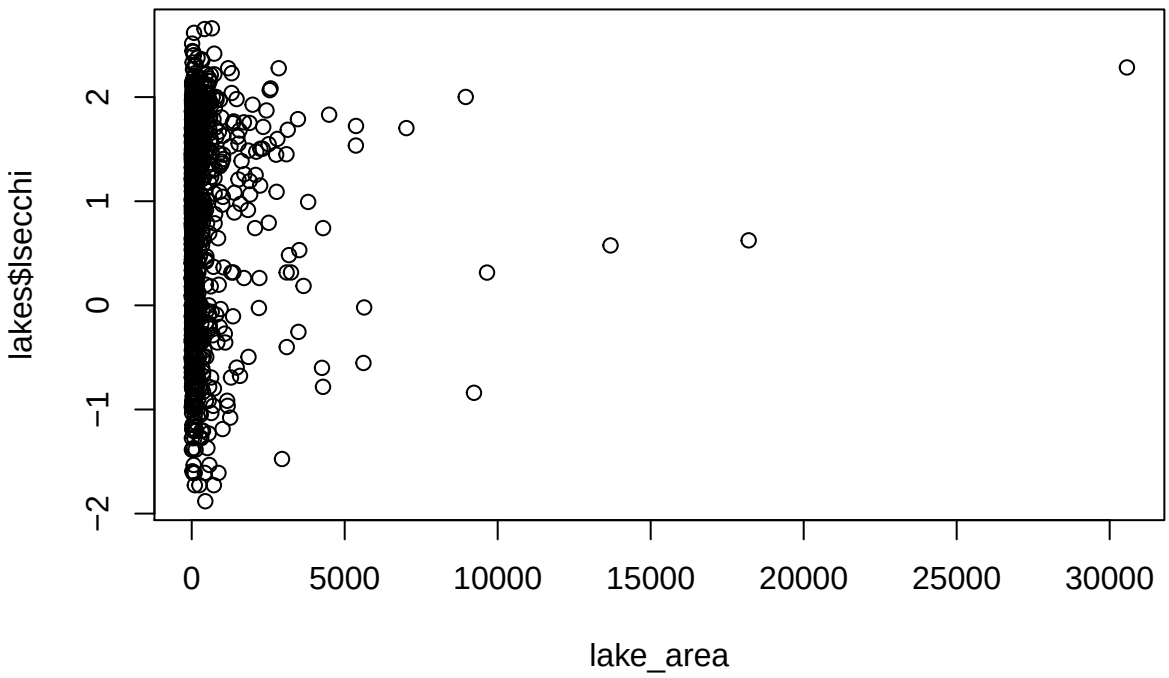


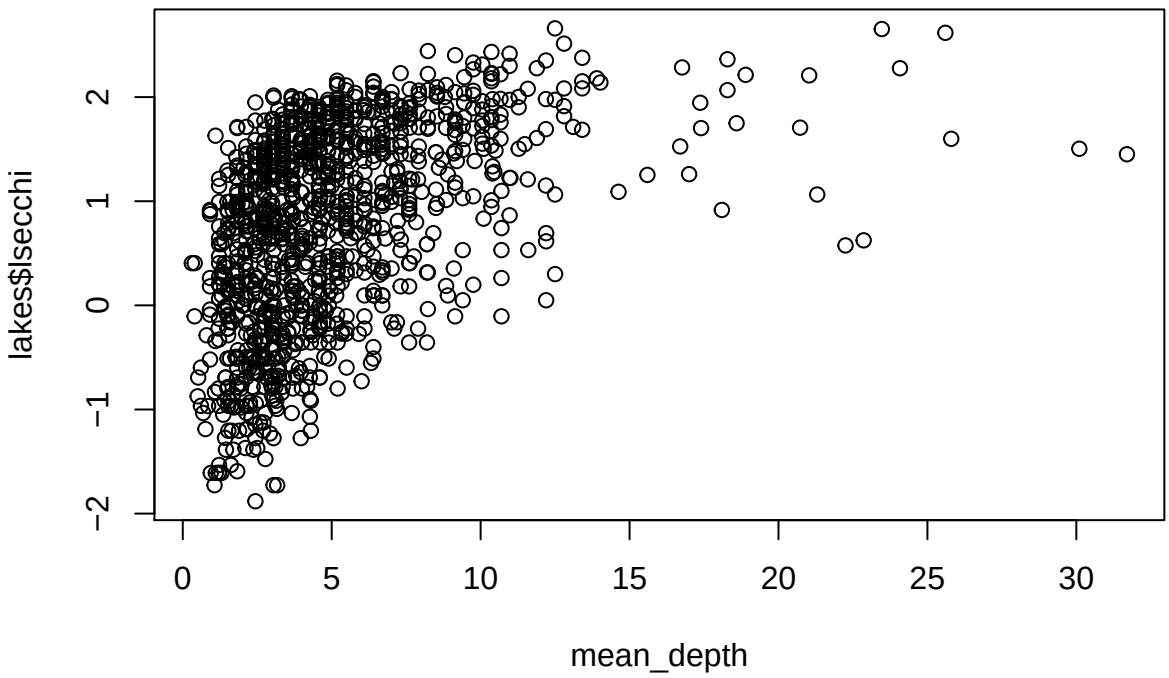

lakes$lsecchi
2
1
0
−1
−2
0
2000
4000
6000
8000
10000
12000
tn

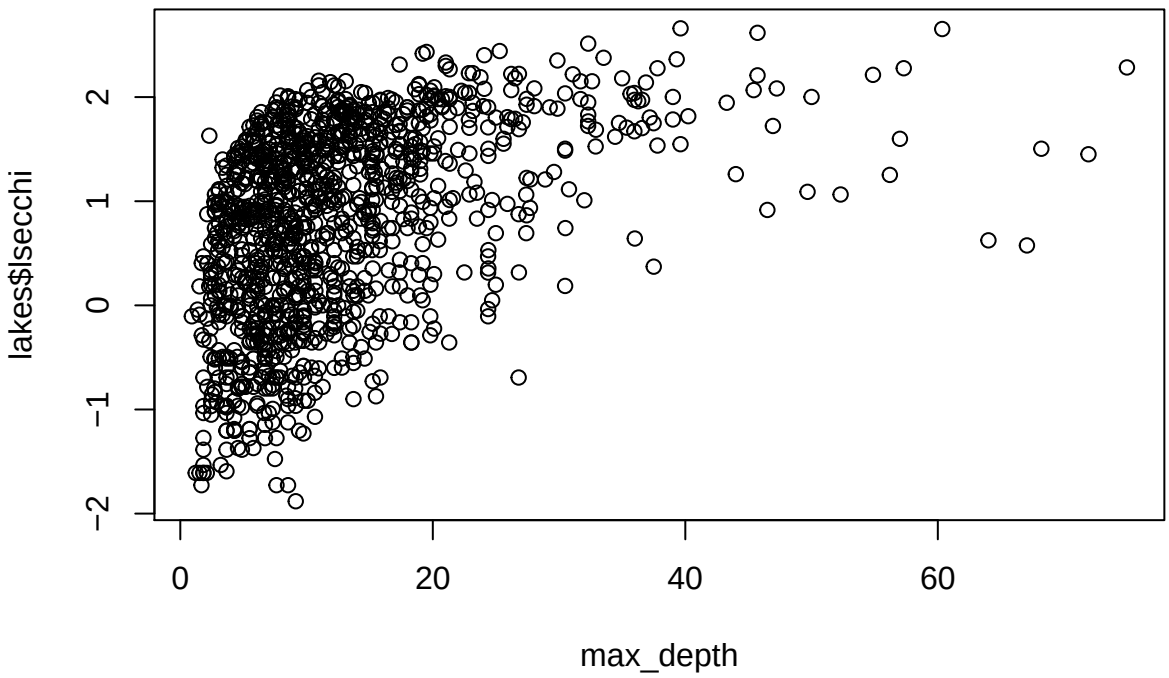

lakes$lsecchi
2
1
0
−1
−2
0
50
100
150
200
chla

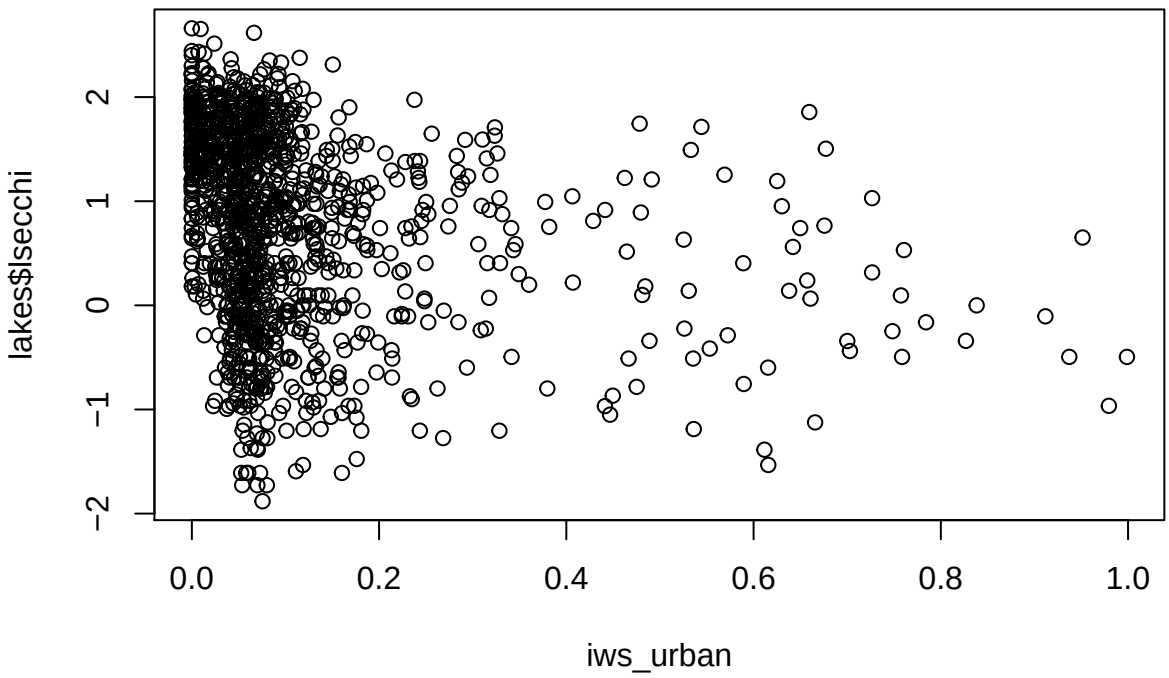

lakes$lsecchi
2
1
0
−1
−2
38
40
42
44
46
lat

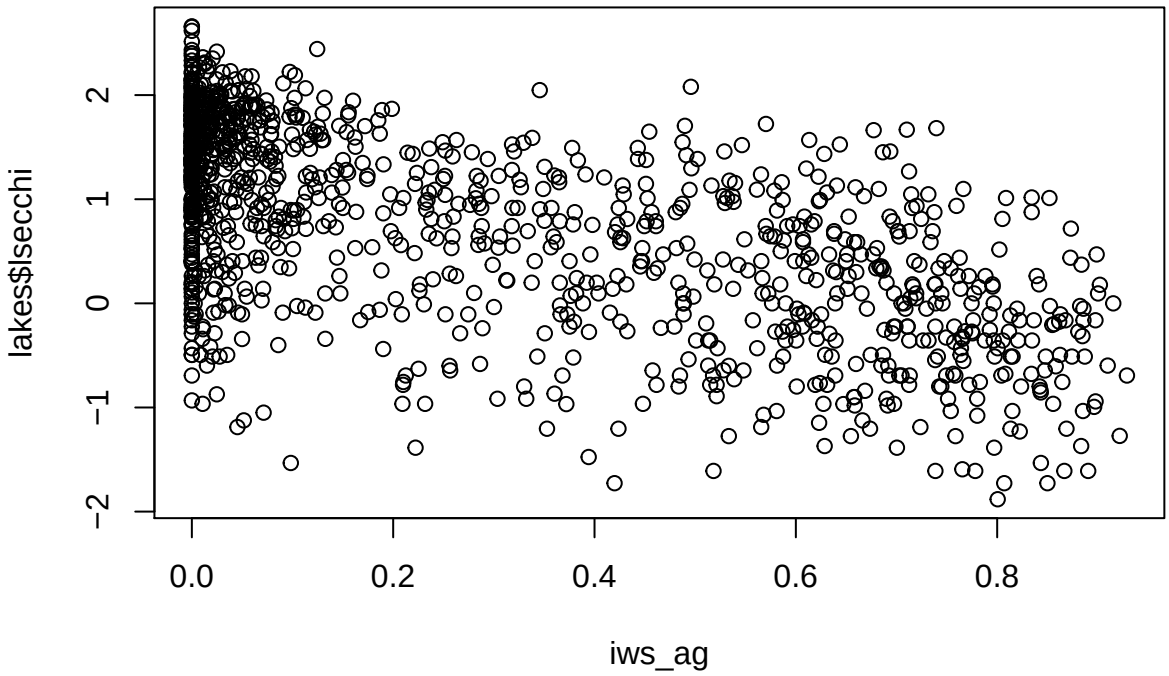

lakes$lsecchi
2
1
0
−1
−2
−95
−90
−85
−80
−75
−70
long

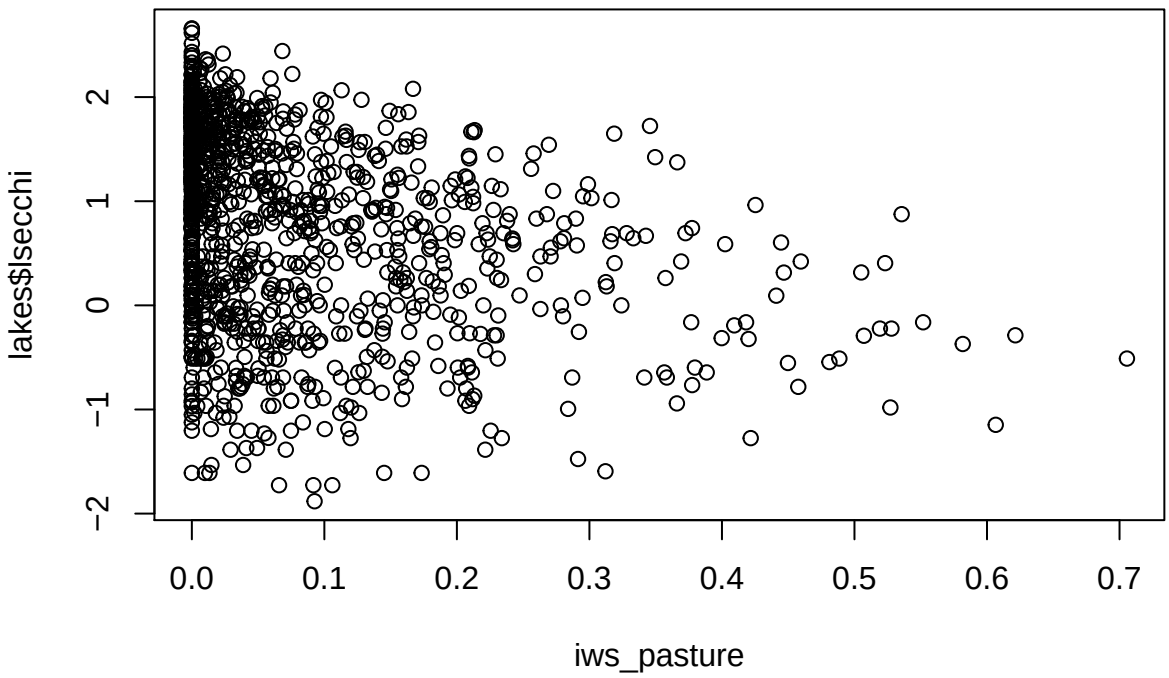

lakes$lsecchi
2
1
0
−1
−2
0
5000
10000
15000
20000
25000
30000
lake_area

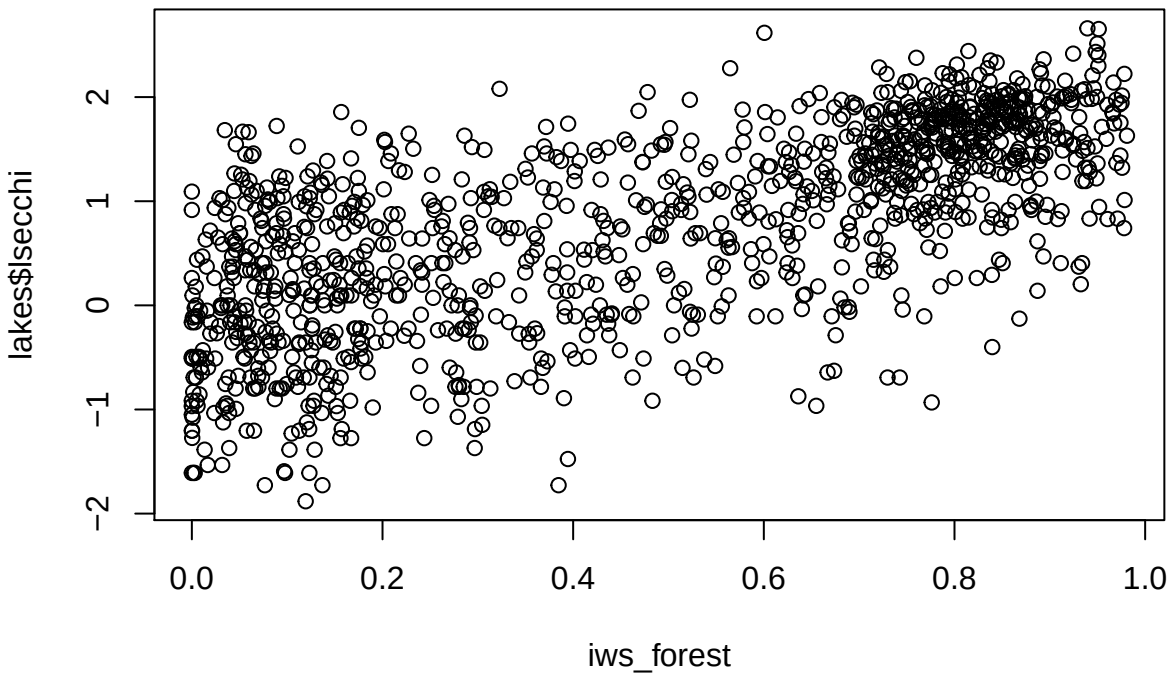

lakes$lsecchi
2
1
0
−1
−2
0
5
10
15
20
25
30
mean_depth

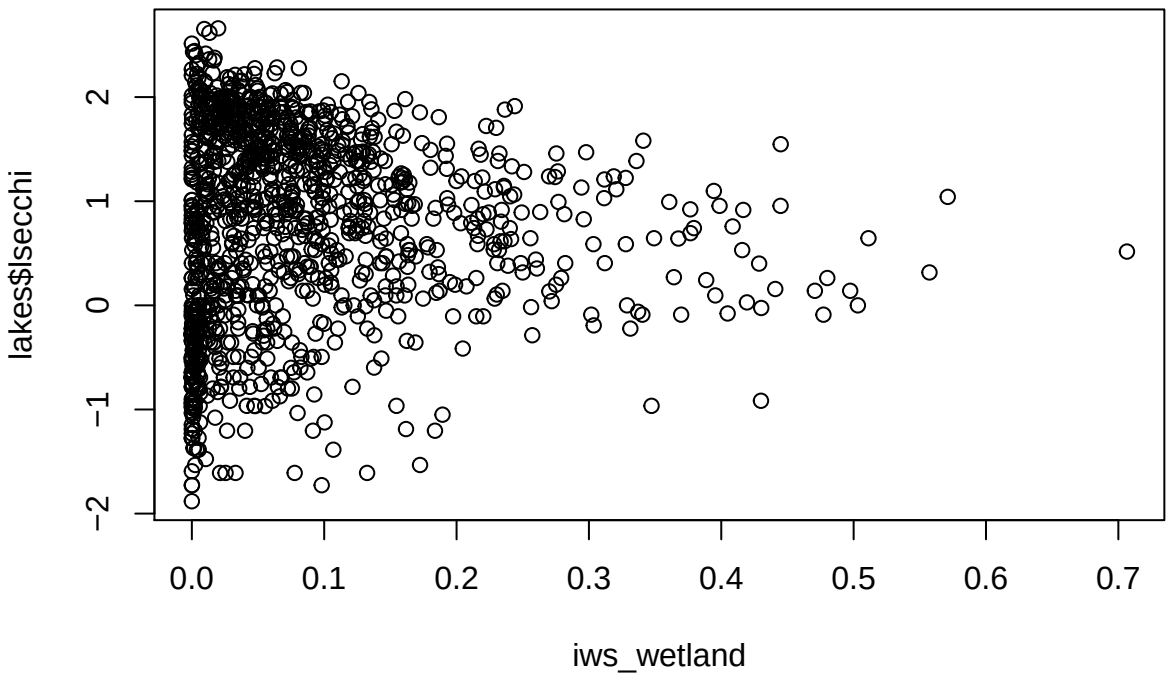

lakes$lsecchi
2
1
0
−1
−2
0
20
40
60
max_depth

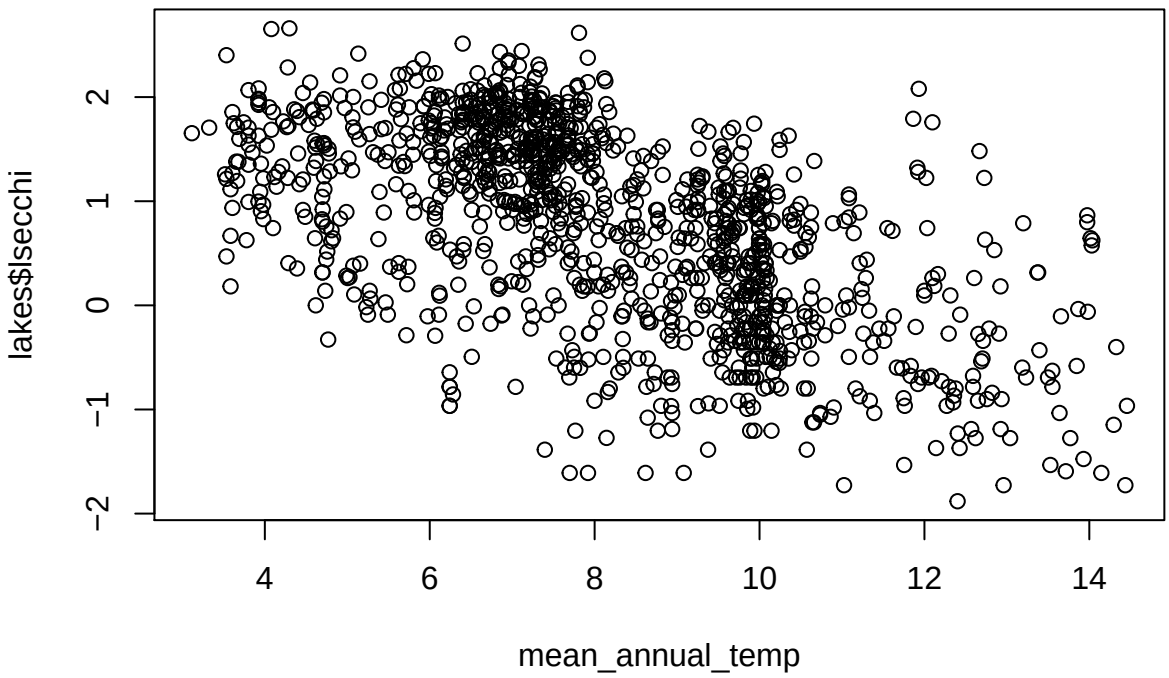

lakes$lsecchi
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0
iws_urban

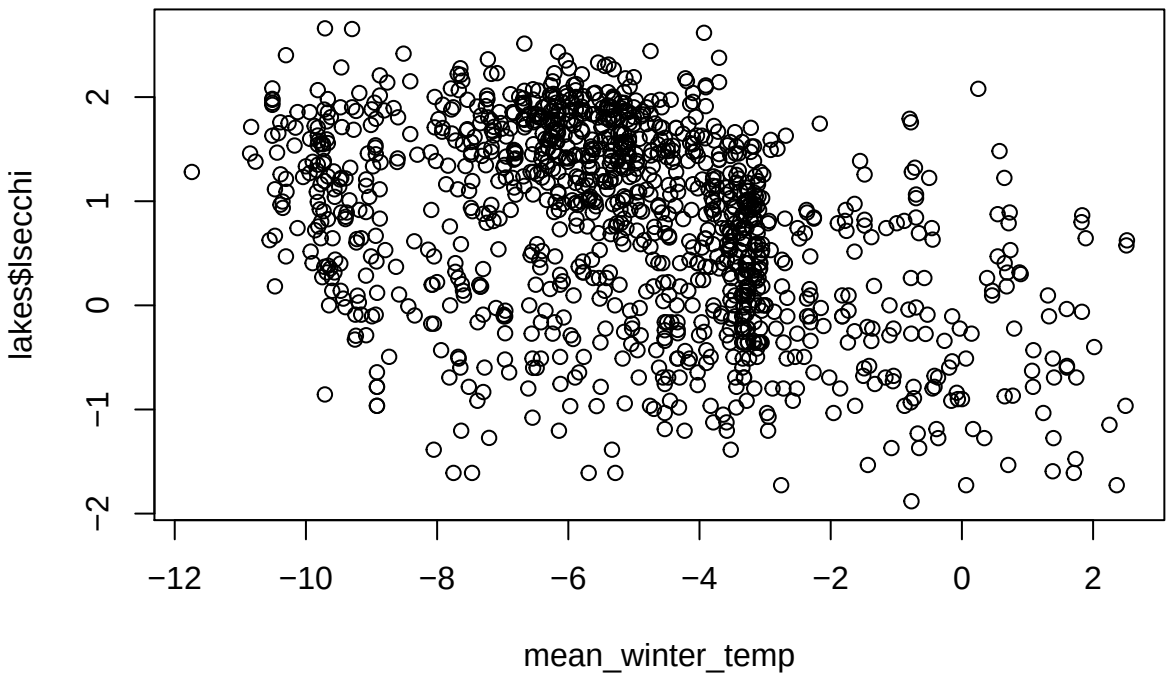

lakes$lsecchi
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
iws_ag

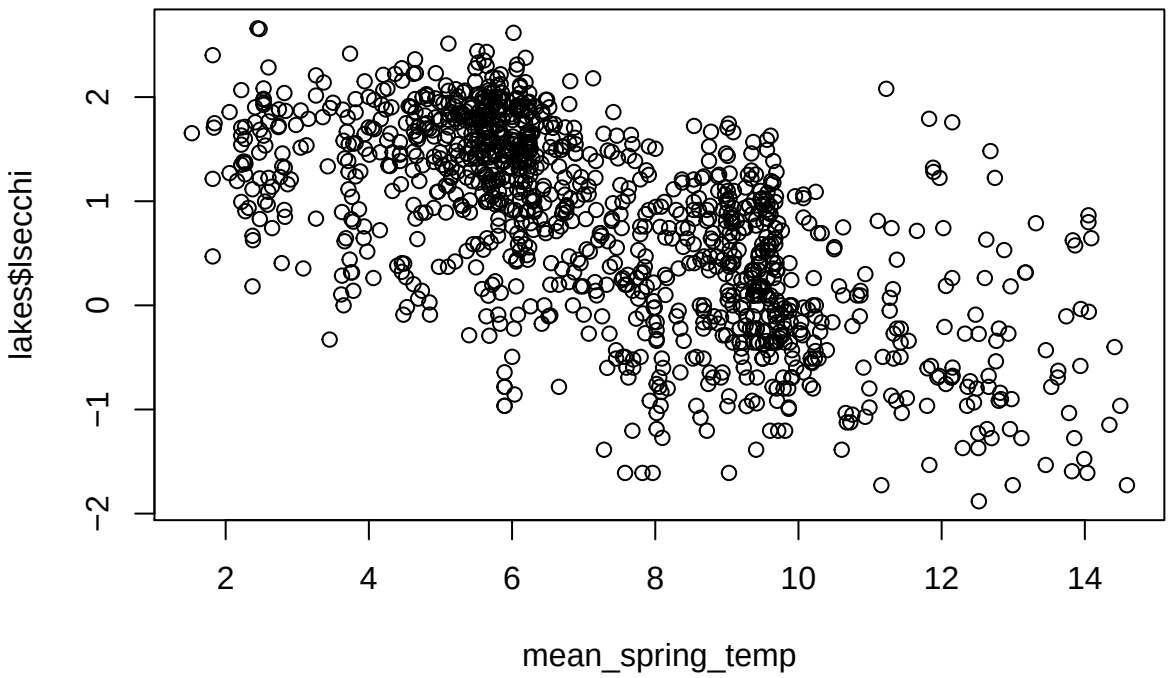

lakes$lsecchi
2
1
0
−1
−2
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
iws_pasture

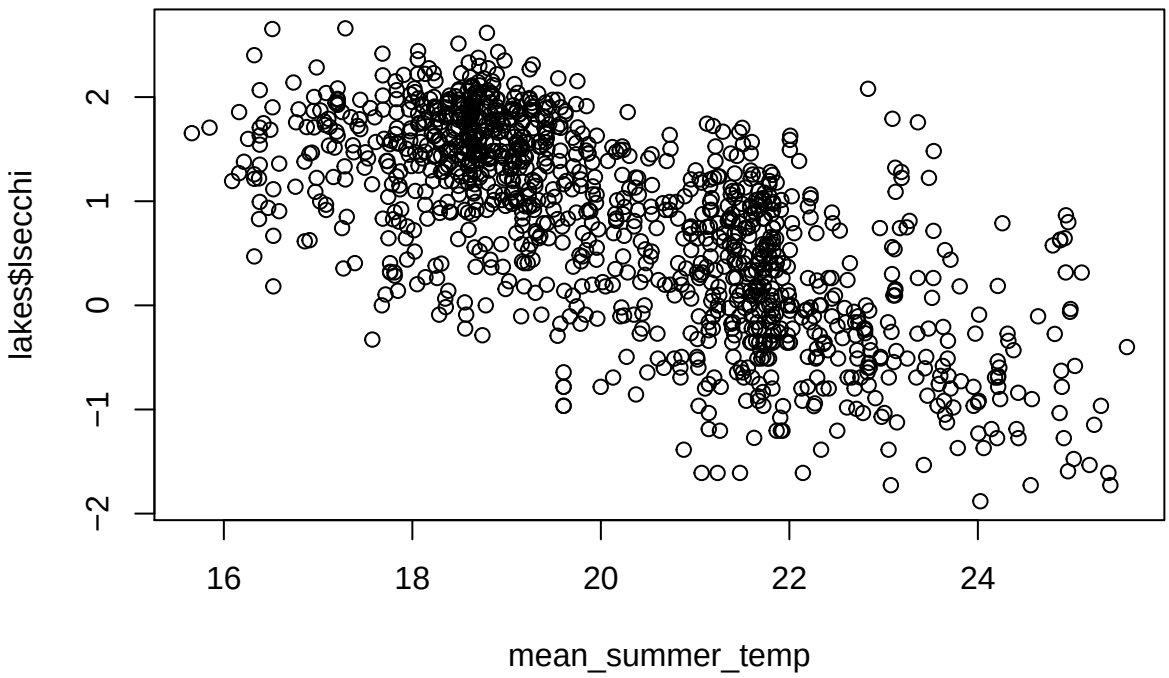

lakes$lsecchi
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0
iws_forest

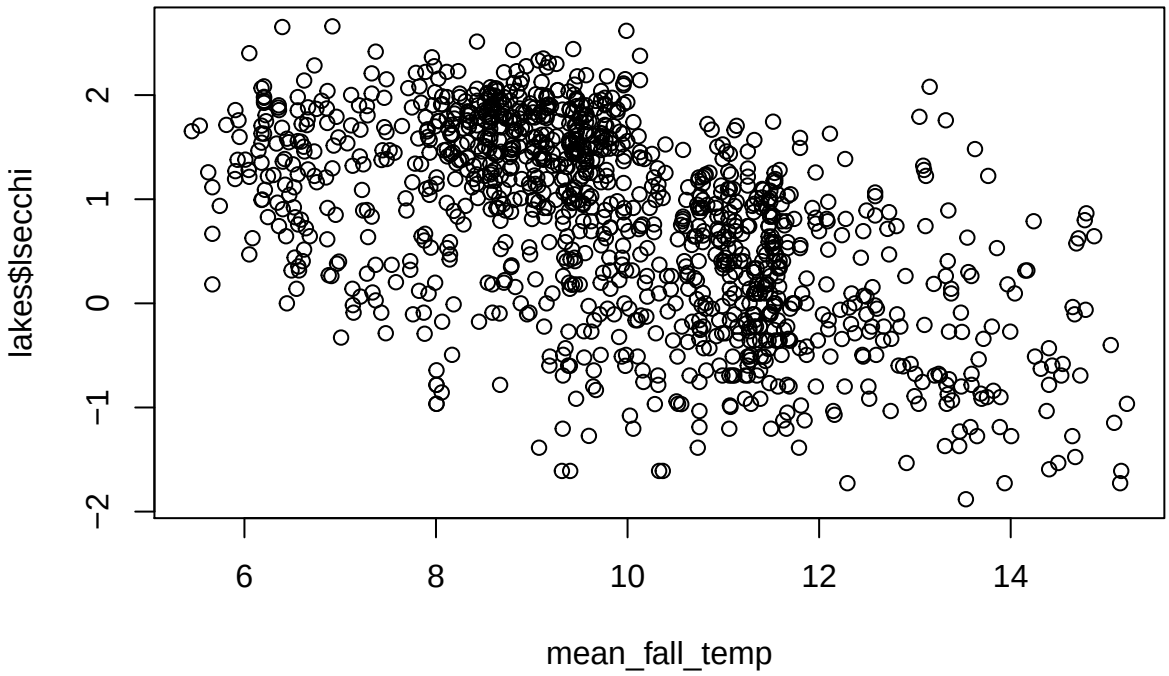

lakes$lsecchi
2
1
0
−1
−2
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
iws_wetland

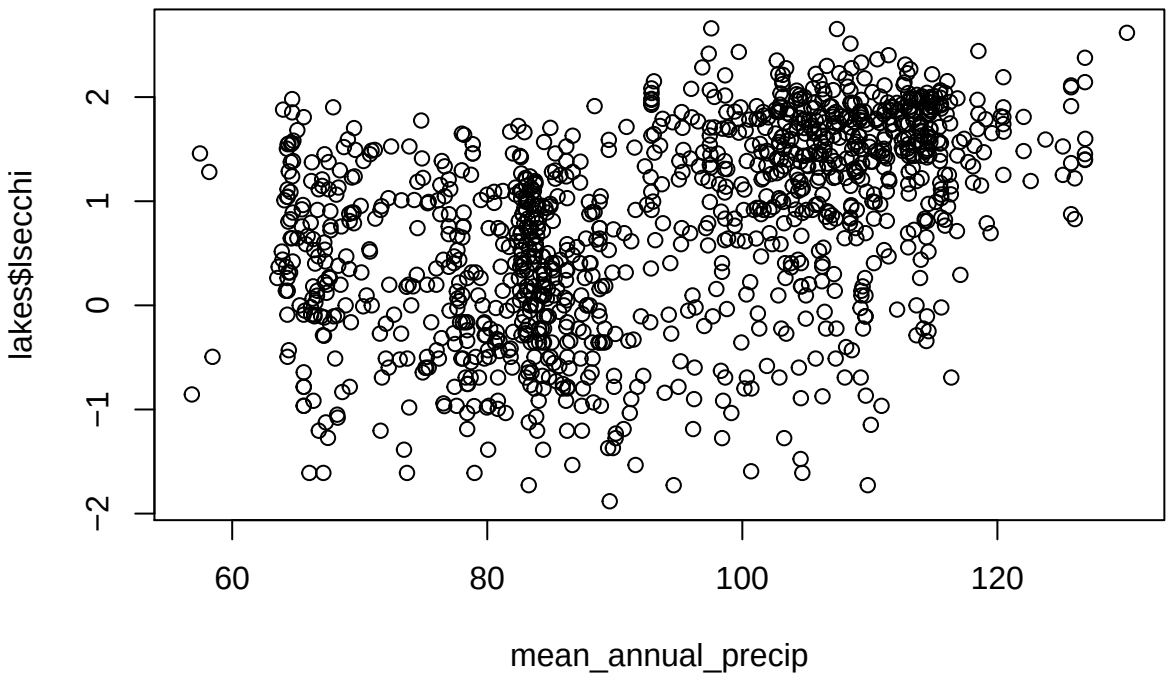

lakes$lsecchi
2
1
0
−1
−2
4
6
8
10
12
14
mean_annual_temp

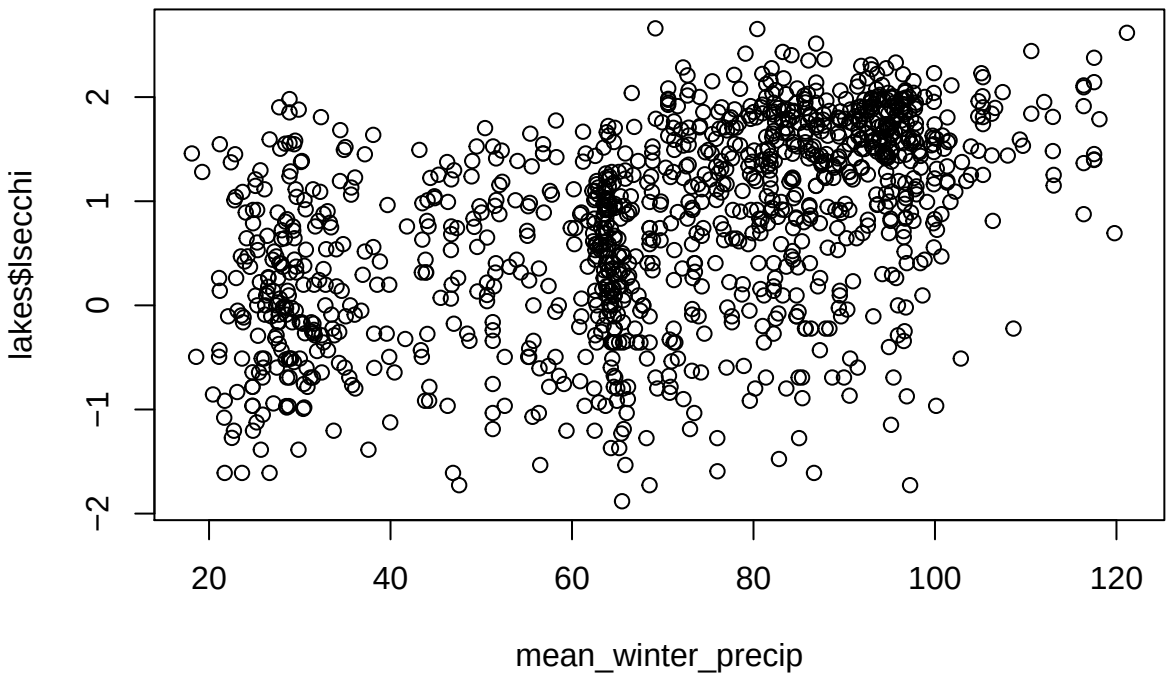

lakes$lsecchi
2
1
0
−1
−2
−12
−10
−8
−6
−4
−2
0
2
mean_winter_temp

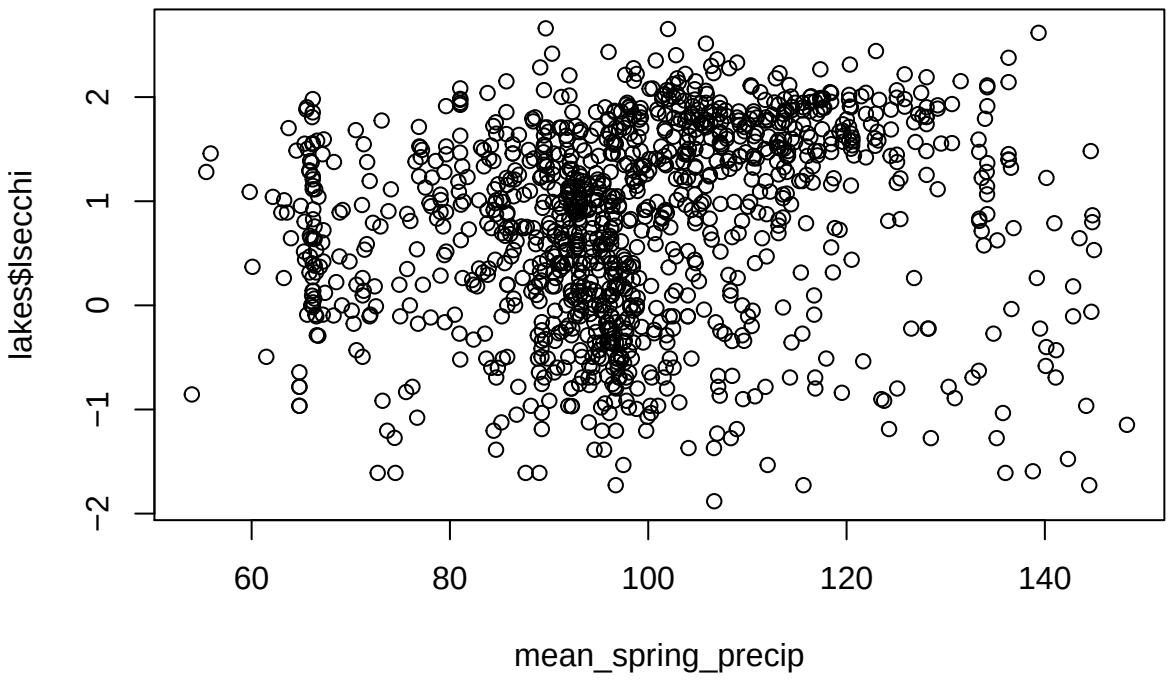

lakes$lsecchi
2
1
0
−1
−2
2
4
6
8
10
12
14
mean_spring_temp

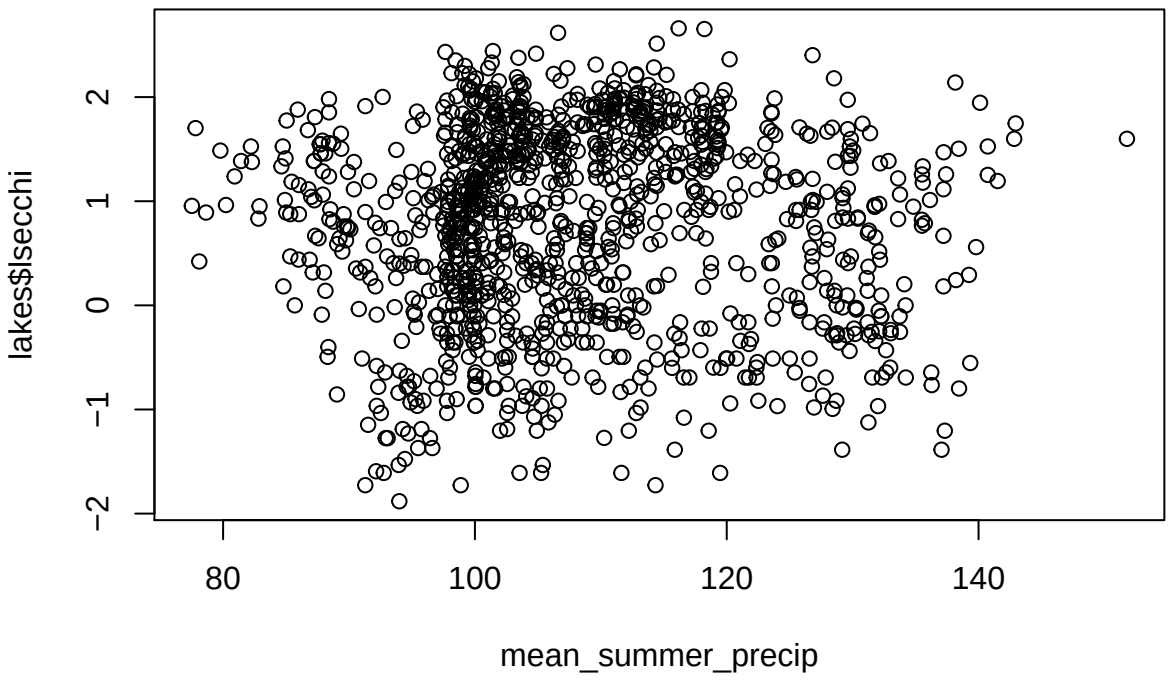

lakes$lsecchi
2
1
0
−1
−2
16
18
20
22
24
mean_summer_temp

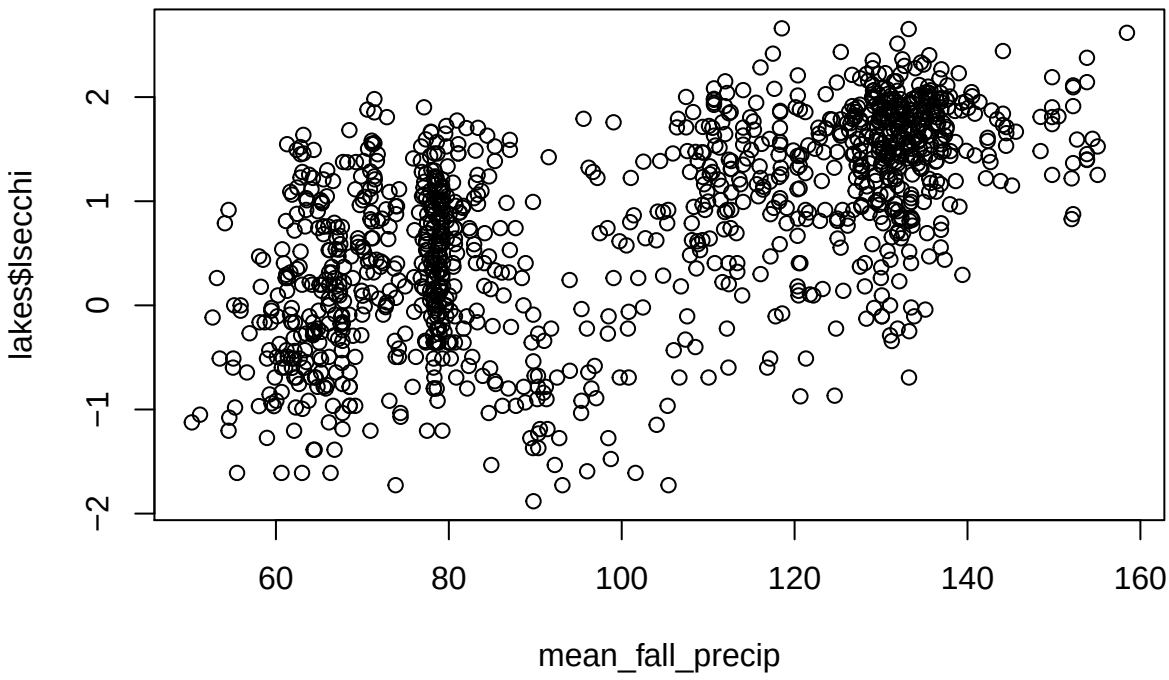

lakes$lsecchi
2
1
0
−1
−2
6
8
10
12
14
mean_fall_temp


lakes$lsecchi
−2
−1
0
1
2
60
80
100
120
mean_annual_precip


lakes$lsecchi
2
1
0
−1
−2
20
40
60
80
100
120
mean_winter_precip


lakes$lsecchi
2
1
0
−1
−2
60
80
100
120
140
mean_spring_precip


lakes$lsecchi
2
1
0
−1
−2
80
100
120
140
mean_summer_precip




```
vars <- c("chla","mean_depth", "tn", "tp", "mean_annual_temp", "mean_spring_temp", "mean_summer_temp",
          "mean_fall_temp", "mean_fall_precip", "iws_wetland")
comb.vars <- expand.grid(vars, vars, stringsAsFactors = FALSE)
comb.vars <- comb.vars[comb.vars[,1] != comb.vars[,2],]
i.vars <- apply(comb.vars, 1, paste, collapse = "+")
var_list<-comb.vars[!duplicated(t(apply(comb.vars, 1, sort))), ]
```

The first step I took in determining our two variables for GAM was doing exploratory data analysis. I began this exploratory data analysis by starting by creating plots of all our variables against log(secchi). Looking at this, I began to shrink the decision space by eliminating all plotted variables that did not visually indicate that they seemed to be related to log(secchi). After this step, I was left with 10 variables: chla, mean depth, tn, tp, mean_annual_temp, mean_spring_temp, mean_summer_temp, mean_fall_temp, mean_fall_precip, and iws_wetland.

The next step was to look at every two variable combination of those 10 variables (total 45 combinations), and use multiple different methods to see which could fit them best.

Logistic Regression

Below is code and output for cross validation error on our logistic regression model.

```
cv_error<-list(NA)
modelfits<-list(NA)
for(i in 1:length(i.vars)) {
  modelformula <- paste("lsecchi ~", i.vars[i])
  modelfits[[i]] <- glm(as.formula(modelformula), family = "gaussian", data = lakes)
  cv_error[[i]]=cv.glm(lakes,modelfits[[i]] ,K=5)$delta[1]
```

```
}
which.min(cv_error)
```

```
## [1] 8
```

```
cv_error[8]
```

```
## [[1]]
## [1] 0.2850753
```

```
best_model<-modelfits[[8]]
summary(best_model)
```

```
##
## Call:
## glm(formula = as.formula(modelformula), family = "gaussian",
##     data = lakes)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.04286  -0.34597   0.02528   0.35465   2.60290
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -0.3095818  0.0634903  -4.876 1.23e-06 ***
## mean_fall_precip  0.0141596  0.0005718  24.763  < 2e-16 ***
## chla             -0.0199308  0.0006023 -33.092  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.2845683)
##
##     Null deviance: 1041.42  on 1187  degrees of freedom
## Residual deviance:  337.21  on 1185  degrees of freedom
## AIC: 1883.3
##
## Number of Fisher Scoring iterations: 2
```

Regression Splines

Below is our code and output for cross validation error on our regression splines.

```
cv_error<-list(NA)
modelfits<-list(NA)

for(i in 1:nrow(var_list)) {
  modelformula<-paste("lsecchi~","bs(",var_list[[i,1]],",df=3)+bs(",var_list[[i,2]],",df=3)")
  modelfits[[i]]<-glm(as.formula(modelformula), family = "gaussian",data=lakes)
  cv_error[[i]]<-cv.glm(lakes, modelfits[[i]], K=5)$delta[1]
}
which.min(cv_error)
```

```
## [1] 27
```

```
cv_error[18]
```

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

```
## [1] 0.2757938
```

```
best_model<-modelfits[[18]]
summary(best_model)
```

```
##
## Call:
## glm(formula = as.formula(modelformula), family = "gaussian",
##     data = lakes)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.67139  -0.25001   0.01408   0.27487   1.97949
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.83230    0.02094  87.489  < 2e-16 ***
## bs(tp, df = 3)1 -5.36961    0.20416 -26.301  < 2e-16 ***
## bs(tp, df = 3)2  1.87872    0.32640   5.756 1.10e-08 ***
## bs(tp, df = 3)3 -3.46226    0.35594  -9.727  < 2e-16 ***
## bs(tn, df = 3)1 -3.20724    0.20069 -15.981  < 2e-16 ***
## bs(tn, df = 3)2  2.28829    0.33355   6.860 1.11e-11 ***
## bs(tn, df = 3)3 -0.87973    0.25583  -3.439 0.000605 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.1715565)
##
##     Null deviance: 1041.42  on 1187  degrees of freedom
## Residual deviance:  202.61  on 1181  degrees of freedom
## AIC: 1286.1
##
## Number of Fisher Scoring iterations: 2
```

Natural Splines

Below is our code and output for cross validation error on our natural splines.

```
cv_error<-list(NA)
modelfits<-list(NA)

for(i in 1:nrow(var_list)) {
  modelformula<-paste("lsecchi~","ns(",var_list[[i,1]],",df=3)+ns(",var_list[[i,2]],",df=3)")
  modelfits[[i]]<-glm(as.formula(modelformula), family = "gaussian",data=lakes)
  cv_error[[i]]<-cv.glm(lakes, modelfits[[i]], K=5)$delta[1]
}
which.min(cv_error)
```

```
## [1] 3
```

```
cv_error[3]
```

```
## [[1]]
## [1] 0.1334536
```

```
best_model<-modelfits[[3]]
summary(best_model)
```

```
##
## Call:
## glm(formula = as.formula(modelformula), family = "gaussian",
##     data = lakes)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.84444  -0.21230   0.03307   0.22916   1.48661
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)         2.60052    0.04056  64.123  < 2e-16 ***
## ns(tp, df = 3)1    -2.51294    0.13606 -18.469  < 2e-16 ***
## ns(tp, df = 3)2    -3.92503    0.15009 -26.151  < 2e-16 ***
## ns(tp, df = 3)3    -1.39141    0.21209  -6.561    8e-11 ***
## ns(chla, df = 3)1 -1.77659    0.11932 -14.890  < 2e-16 ***
## ns(chla, df = 3)2 -1.60509    0.13637 -11.770  < 2e-16 ***
## ns(chla, df = 3)3 -0.62263    0.20074  -3.102  0.00197 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.1326292)
##
##     Null deviance: 1041.42  on 1187  degrees of freedom
## Residual deviance:  156.64  on 1181  degrees of freedom
## AIC: 980.38
##
## Number of Fisher Scoring iterations: 2
```

Smoothing Splines

Below is our code and output for smoothing splines

```
errorlist<-list(NA)
bestmodel<-list(NA)
i=1
#code for our own k-fold cv
K=5
folds = sample(1:K,nrow(lakes),replace=T)
modelfits<-list(NA)
errorlist<-list(NA)
bestmodel<-list(NA)
moderror<-list(NA)
for(k in 1:K){
  CV.train = lakes[folds != k,]
  CV.test = lakes[folds == k,]
  CV.ts_y = CV.test$lsecchi
  for(i in 1:nrow(var_list)) {
    modelformula<-paste("lsecchi~","s(",var_list[[i,1]],",4)+s(",var_list[[i,2]],",4)")
    modelfits[[i]]<-gam(as.formula(modelformula), data=CV.train)
    errorlist[[i]]<-sum((CV.ts_y-predict(modelfits[[i]], newdata=CV.test))^2)
      }
  moderror[[k]]<-errorlist[[which.min(errorlist)]]/nrow(CV.test)
  bestmodel[[k]]<-which.min(errorlist)
```

```
}
bestmodel
```

```
## [[1]]
## [1] 27
##
## [[2]]
## [1] 11
##
## [[3]]
## [1] 2
##
## [[4]]
## [1] 2
##
## [[5]]
## [1] 2
```

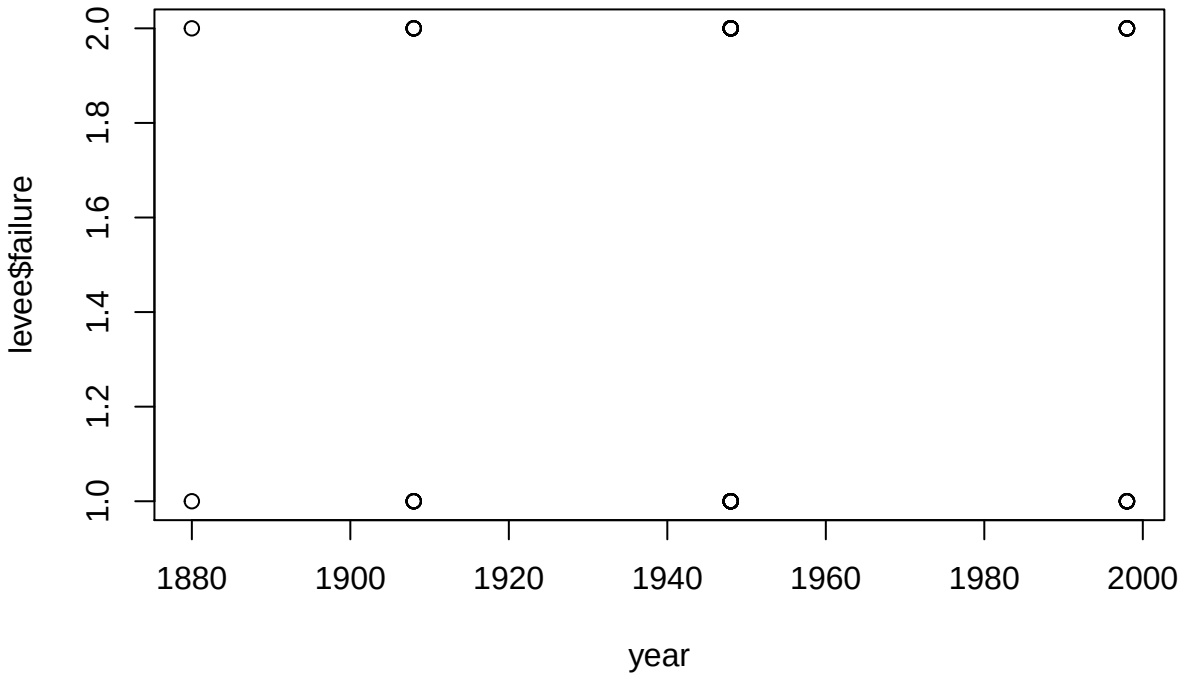
```
moderror
```

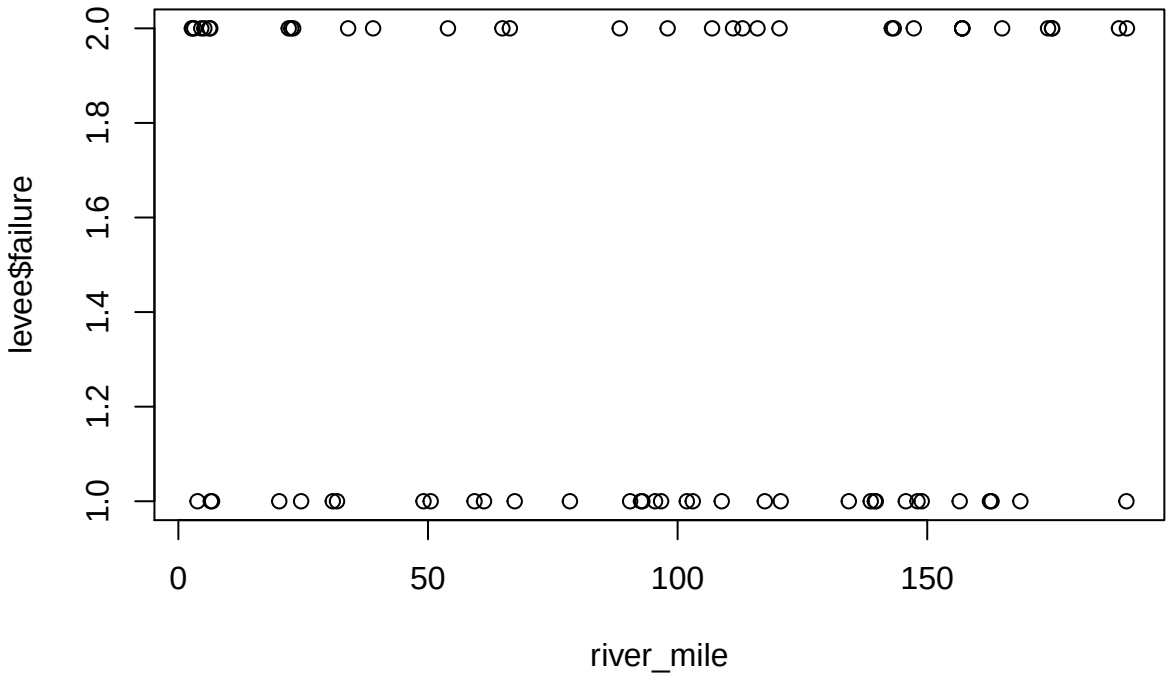
```
## [[1]]
## [1] 0.1586299
##
## [[2]]
## [1] 0.1401898
##
## [[3]]
## [1] 0.1356859
##
## [[4]]
## [1] 0.1530831
##
## [[5]]
## [1] 0.1322459
```

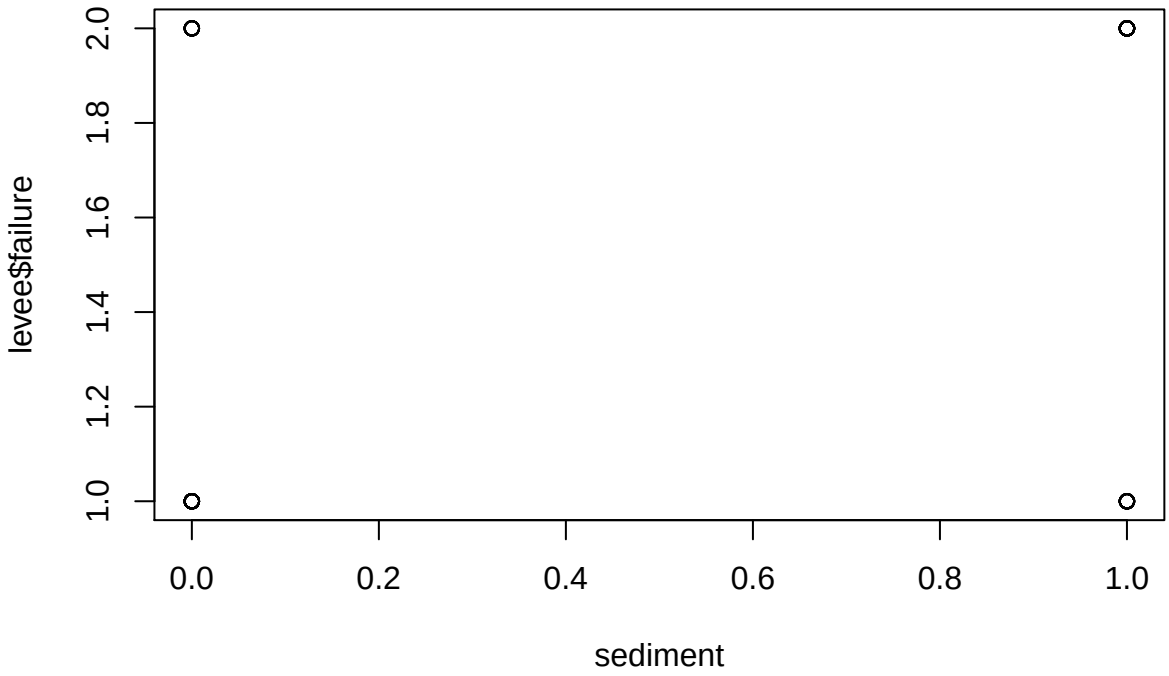
After looking at our two variable combinations using multiple methods, it seemed like the variables that appeared multiple times as the 'best' model was mean fall depth and mean temp. The best 5-fold cross validation error that we were able to achieve was 0.12926, which seemed superior to our error on HW 3.

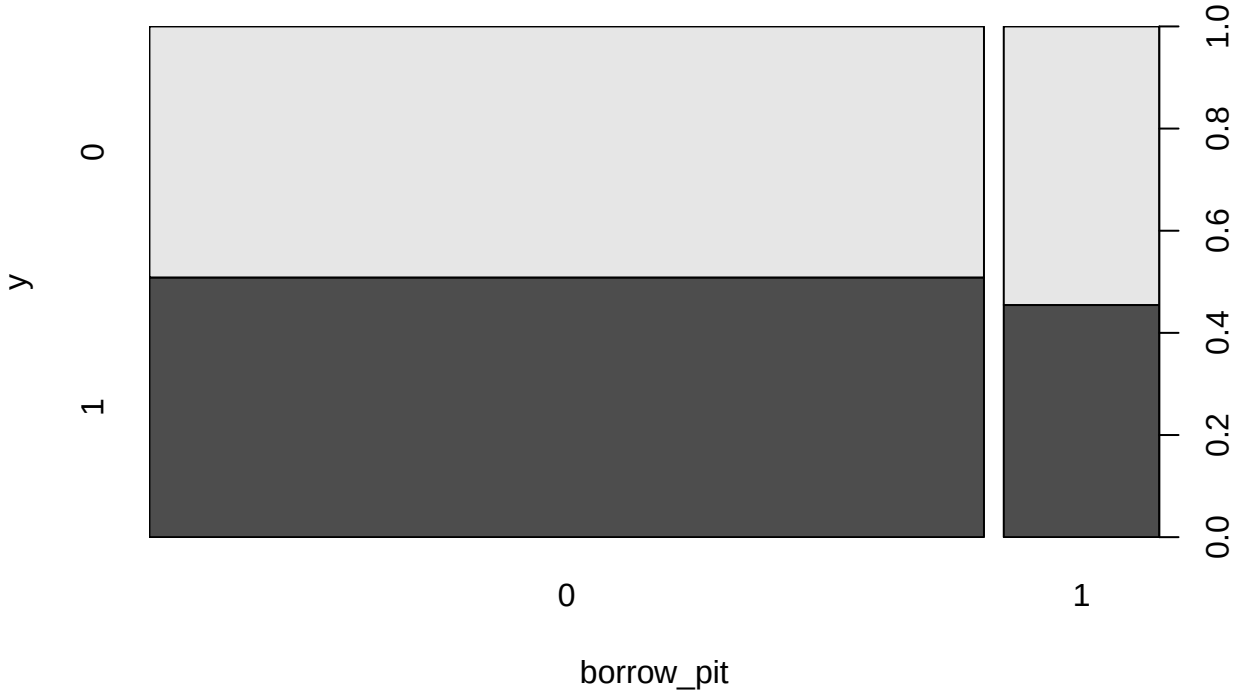
4.

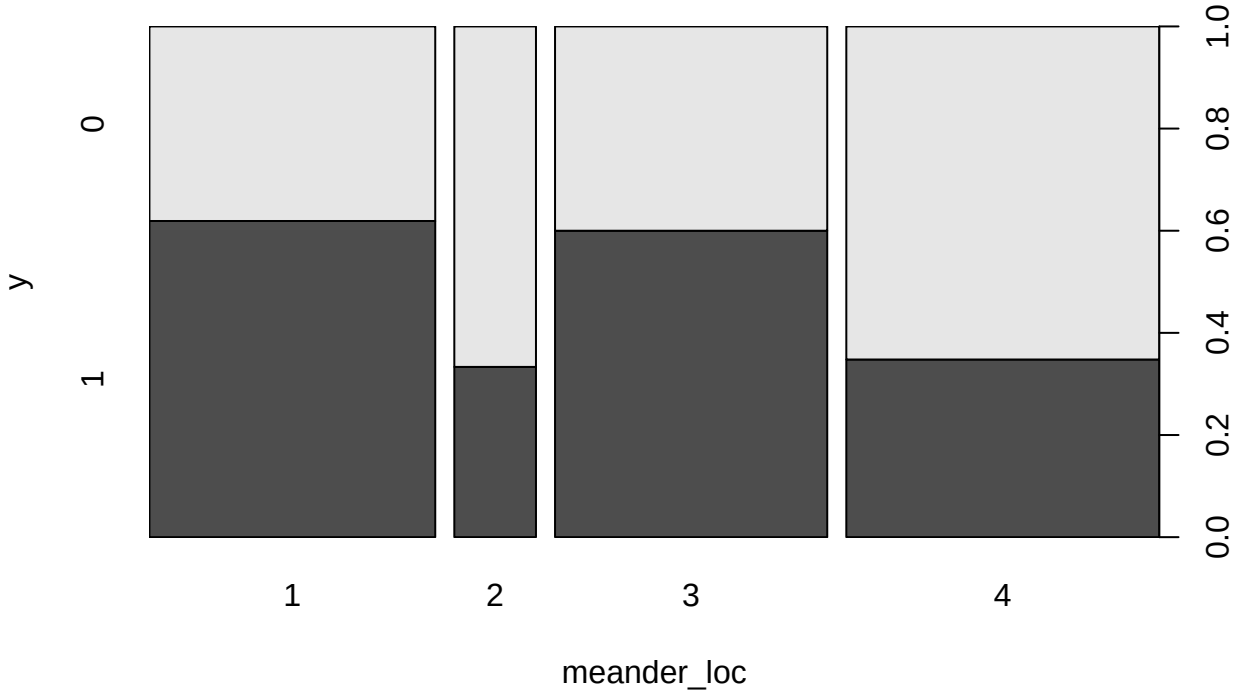
```
levee<-read.delim("mmr_levee.txt", sep = "", header = FALSE)
colnames(levee)<-c("failure","year","river_mile","sediment","borrow_pit","meander_loc","channel_width",
levee$failure<-as.factor(levee$failure)
levee$meander_loc<-as.factor(levee$meander_loc)
levee$land_type<-as.factor(levee$land_type)
levee$borrow_pit<-as.factor(levee$borrow_pit)

for(i in 2:14){
  plot(y=levee$failure, x=levee[,i], xlab=colnames(levee)[i])
}
```

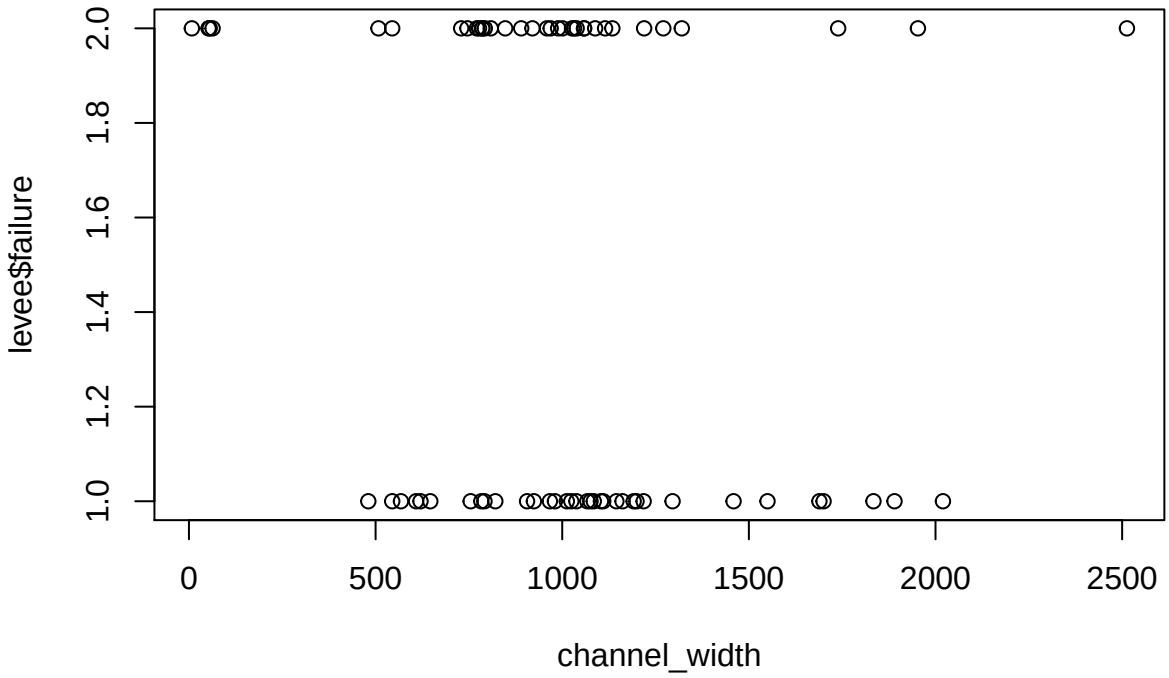

levee$failure
1.0
1.2
1.4
1.6
1.8
2.0
1880
1900
1920
1940
1960
1980
2000
year

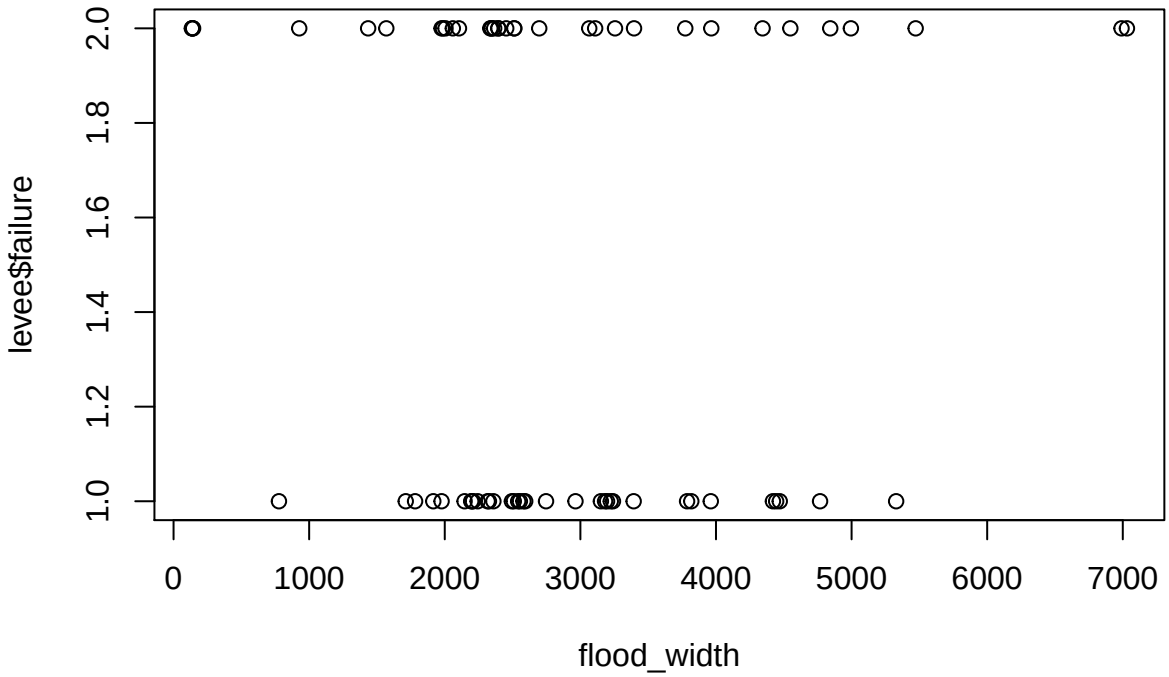

levee$failure
1.0
1.2
1.4
1.6
1.8
2.0
0
50
100
150
river_mile

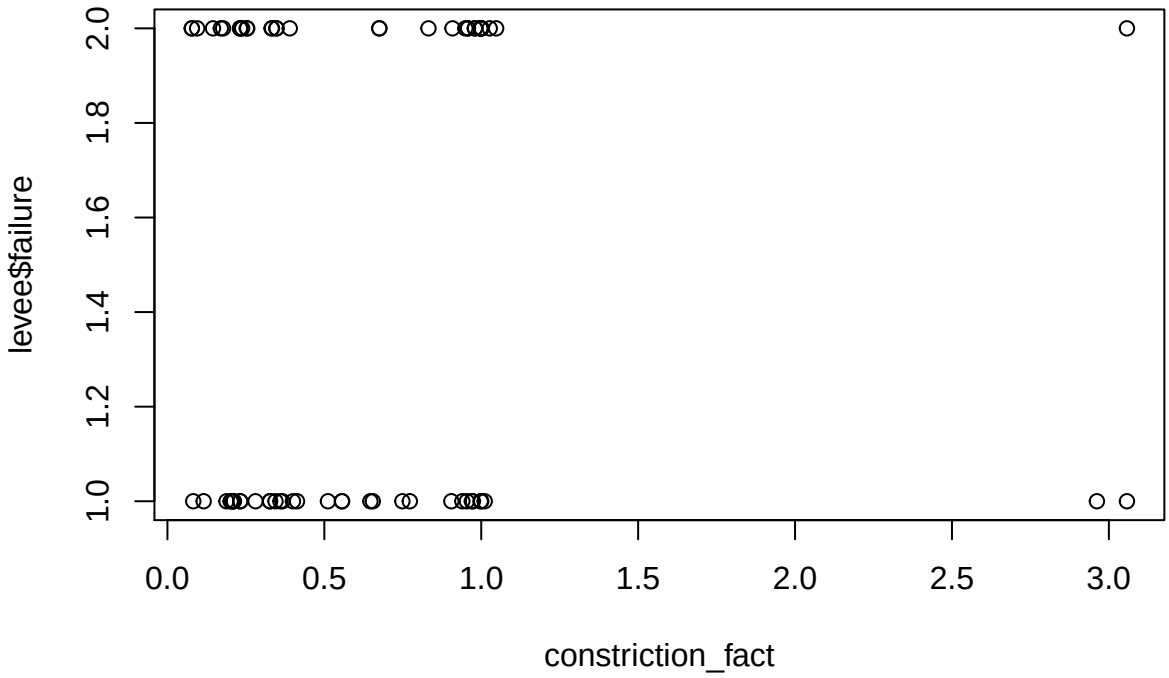

levee$failure
1.0
1.2
1.4
1.6
1.8
2.0
0.0
0.2
0.4
0.6
0.8
1.0
sediment

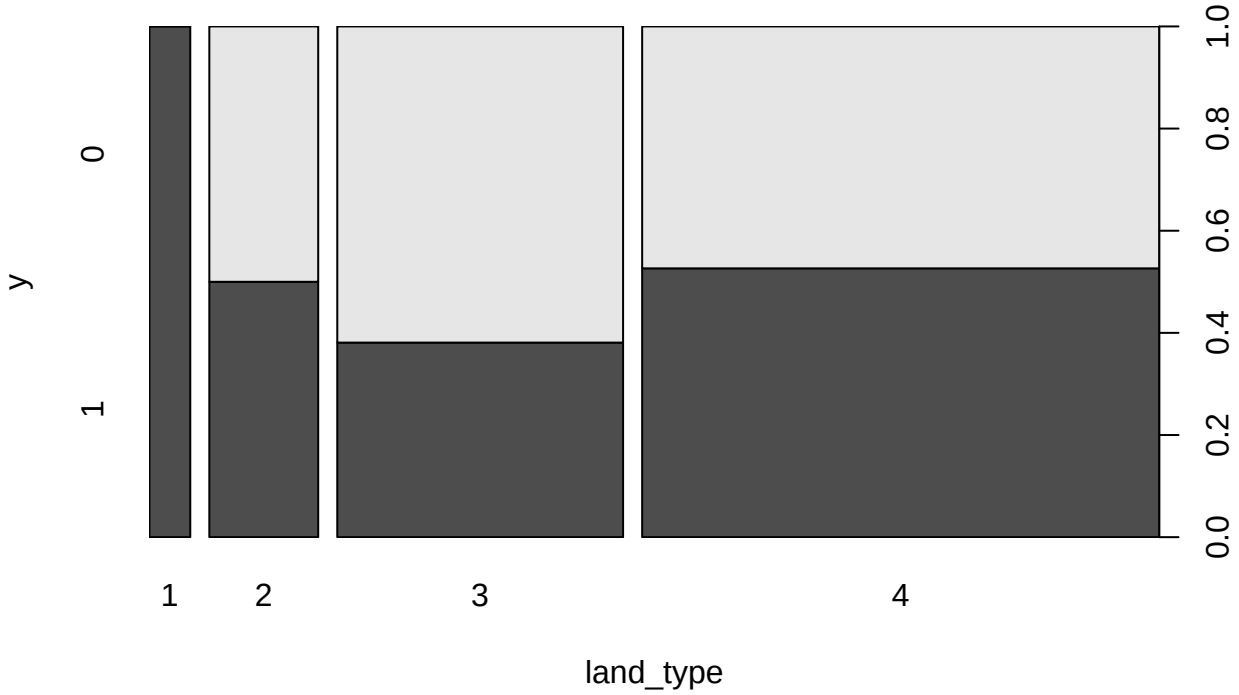

y
0
1
0.0
0.2
0.4
0.6
0.8
1.0
0
1
borrow_pit

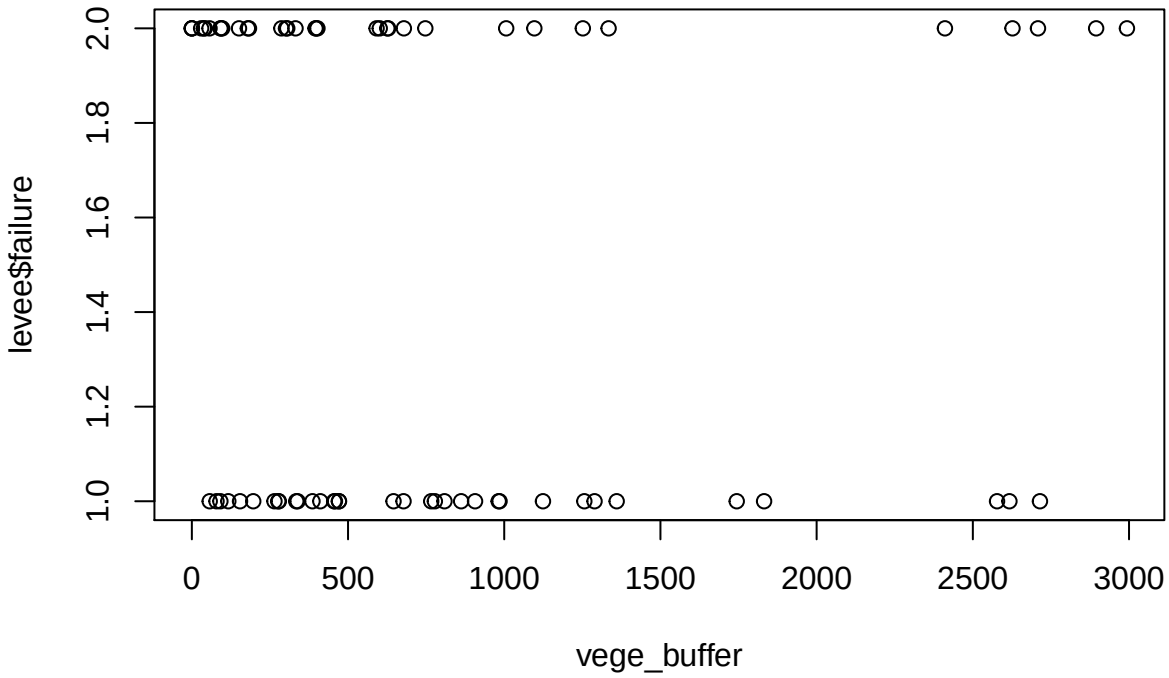

y
0
1
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
4
meander_loc

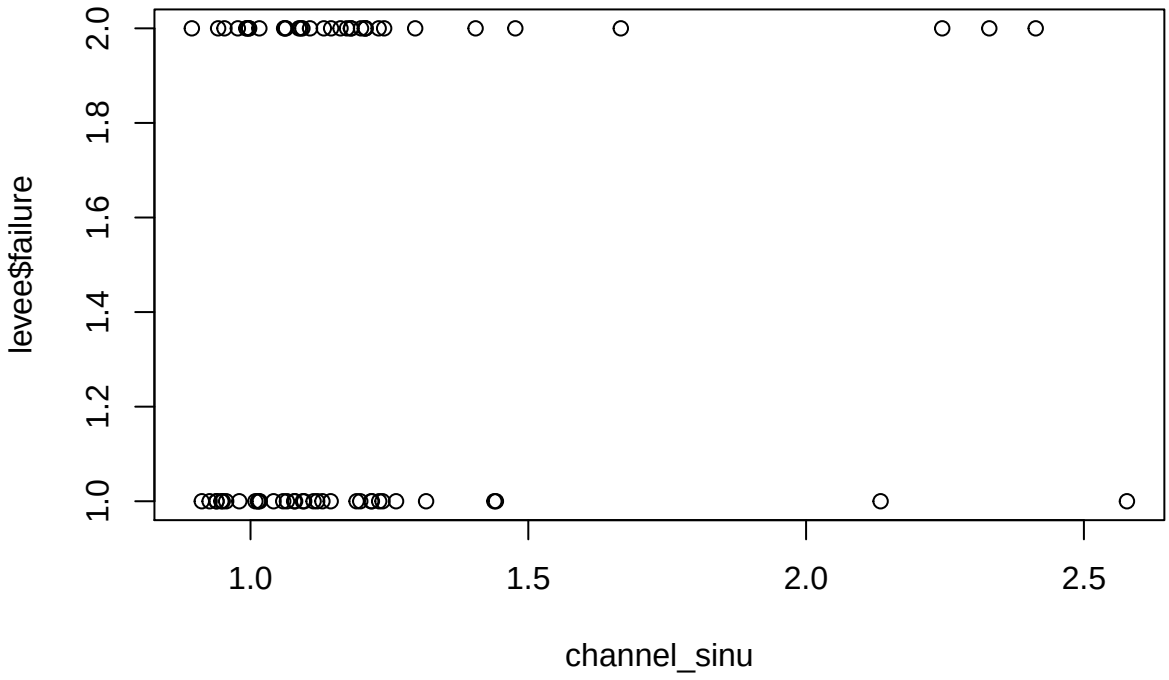

levee$failure
1.0
1.2
1.4
1.6
1.8
2.0
0
500
1000
1500
2000
2500
channel_width

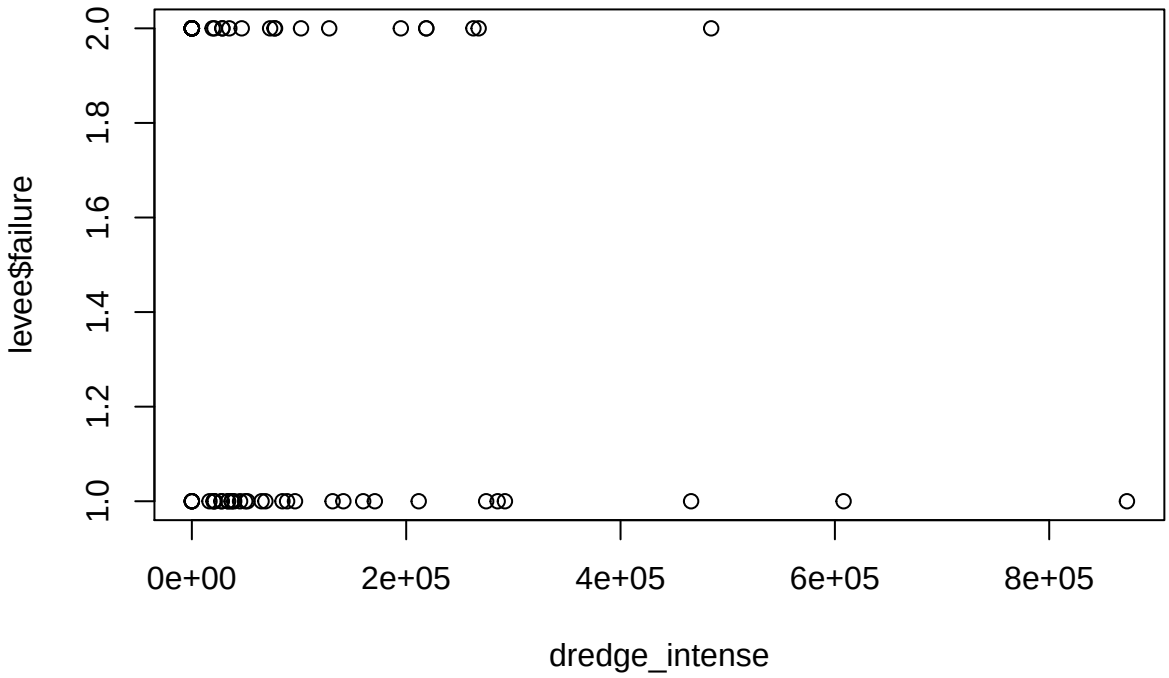

levee$failure
1.0
1.2
1.4
1.6
1.8
2.0
0
1000
2000
3000
4000
5000
6000
7000
flood_width

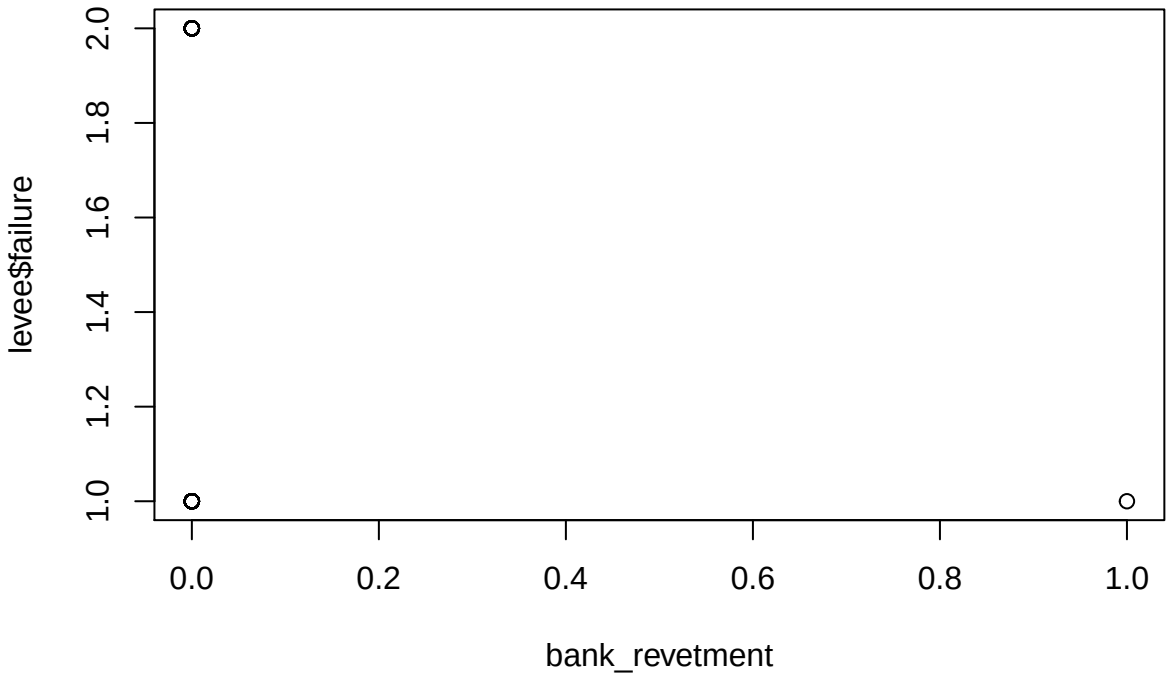

levee$failure
1.0
1.2
1.4
1.6
1.8
2.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
constriction_fact


y
0
1
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
4
land_type


levee$failure
1.0
1.2
1.4
1.6
1.8
2.0
0
500
1000
1500
2000
2500
3000
vege_buffer


levee$failure
1.0
1.2
1.4
1.6
1.8
2.0
1.0
1.5
2.0
2.5
channel_sinu


levee$failure
1.0
1.2
1.4
1.6
1.8
2.0
0e+00
2e+05
4e+05
6e+05
8e+05
dredge_intense




```
train=sample(70,35)
levee_test<-levee[-train,]
```

The first step I took in determining our variables for GAM was doing exploratory data analysis. I began this exploratory data analysis by starting by creating plots of all our variables against levee failure. Looking at this, I began to shrink the decision space by eliminating all plotted variables that did not visually indicate that they seemed to be related to levee failure. After this step, I was left with 2 variables, land type and meander location.

Since this was a classification problem, I wanted to see this two variable gam using logistic regression, lda, and qda.

GLM

Below is the model summary and confusion matrix/error rate for our GLM model on these two variables.

```
cost <- function(r, pi = 0) mean(abs(r-pi) > 0.5)
modelfits<- glm(failure ~ meander_loc + land_type, family = "binomial", data = levee, subset=train)
glm.probs=predict(modelfits,newdata=levee_test,type="response")
glm.pred=rep(0 ,35)
glm.pred[glm.probs >.5]=1

table(glm.pred,levee_test$failure)
```

```
##
## glm.pred  0  1
##        0 11 13
##        1  4  7
```

```r
mean(glm.pred==levee_test$failure)
```

```
## [1] 0.5142857
```

```r
summary(modelfits)
```

```
##
## Call:
## glm(formula = failure ~ meander_loc + land_type, family = "binomial",
##     data = levee, subset = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9422  -1.0205  -0.2846   0.8755   1.6584
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    17.74848 2780.83378   0.006    0.995
## meander_loc2   -0.35695    1.43714  -0.248    0.804
## meander_loc3    2.05769    1.31727   1.562    0.118
## meander_loc4   -0.04478    1.05247  -0.043    0.966
## land_type2    -17.74848 2780.83414  -0.006    0.995
## land_type3    -20.88992 2780.83410  -0.008    0.994
## land_type4    -18.08467 2780.83376  -0.007    0.995
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 47.804  on 34  degrees of freedom
## Residual deviance: 35.229  on 28  degrees of freedom
## AIC: 49.229
##
## Number of Fisher Scoring iterations: 16
```

LDA

Below is the model summary and confusion matrix/error rate for our QDA model on these two variables.

```r
lda.fit<-lda(failure~meander_loc + land_type, data = levee, subset = train)

lda.pred=predict(lda.fit , levee_test, type = "response")
lda.class=lda.pred$class
table(lda.class ,levee_test$failure)
```

```
##
## lda.class  0  1
##         0 11 13
##         1  4  7
```

```r
mean(lda.class==levee_test$failure)
```

```
## [1] 0.5142857
```

Our LDA and GLM analysis had the same confusion matrix! Additionally, attempting to fit our QDA, our model was unable to run, thus I examined if there were other methods that would provide a better choice for variable

Variable Selection Part 2

I then decided to use best subset selection with BIC as the criterion to find if I had missed something important that I had not considered.

```
regfit.full=regsubsets(failure~.,data=levee, nvmax=13)
reg.summary<-summary(regfit.full)

reg.summary$bic
```

```
##  [1] -2.0431096 -1.3066992 -0.1596221  2.7436193  5.8294542  8.7250862
##  [7] 10.9591759 14.3889170 18.1523848 22.1253617 26.0818605 30.1549509
## [13] 34.1377451
```

The best subset regression using BIC as the criteria actually indicated that a 1 variable model would be superior, specifically a model using Sediment as it's only predictor.

Thus, I wished to see if the previous methods (GLM, LDA, QDA) would provide a better error rate with sediment than the other two variable model.

Logistic Reg 2

```
cost <- function(r, pi = 0) mean(abs(r-pi) > 0.5)
modelfits<- glm(failure ~ sediment, family = "binomial", data = levee, subset=train)
glm.probs=predict(modelfits,newdata=levee_test,type="response")
glm.pred=rep(0 ,35)
glm.pred[glm.probs >.5]=1
table(glm.pred,levee_test$failure)
```

```
##
## glm.pred  0  1
##        0  8  4
##        1  7 16
```

```
mean(glm.pred==levee_test$failure)
```

```
## [1] 0.6857143
```

LDA 2

```
lda.fit<-lda(failure~ sediment, data = levee, subset = train)
lda.pred=predict(lda.fit , levee_test, type = "response")
lda.class=lda.pred$class
table(lda.class ,levee_test$failure)
```

```
##
## lda.class  0  1
##         0  8  4
##         1  7 16
```

```
mean(lda.class==levee_test$failure)
```

```
## [1] 0.6857143
```

QDA 2

```
qda.fit<-qda(failure~sediment, data = levee, subset = train)
qda.pred=predict(qda.fit , levee_test, type = "response")
qda.class=qda.pred$class
table(qda.class ,levee_test$failure)
```

```
##
## qda.class  0  1
##         0  8  4
##         1  7 16
```

```
mean(qda.class==levee_test$failure)
```

```
## [1] 0.6857143
```

Looking at our confusion matrix results, it seems like our highest success rate was using the 2 variable model consisting of land type and meander location. Sadly, a 1 variable model using sediment did not outperform the 2 variable model, looking at our training vs testing cross validation method.