Assignment 2 ayuan45

Andy Yuan 251159206

2025-03-07

Question 1

To show that the expected training error is less than or equal to the expected test error, we will run a simulation study.

```r
set.seed(1) # set for reproducibility

Numbersim <- 1000 # number of simulations
N <- 100 # training set size
M <- 100 # test set size
p <- 5 # number of parameters

simulation <- function(N=100,M=100,p=5,Numbersim=1000){

  trainerror <- vector() # set empty vector
  testerror <- vector() # set empty vector

  for (num in 1:Numbersim) { # repeating below 1000 times
    X_train <- matrix(rnorm(N*p),nrow=N, ncol=p) # training data
    X_test <- matrix(rnorm(M*p),nrow=M, ncol=p) # test data
    beta <- rnorm(p, sd =100) # random betas
    intercept <- rnorm(N, sd=1) # random intercepts for train data
    intercept2 <- rnorm(M,sd=1) # random intercepts for test data
    y_train <- X_train %*% beta + intercept # training response
    y_test <- X_test %*% beta + intercept2 # test response

    linmod <- lm(y_train ~ ., data=data.frame(X=X_train))
    # fitting a model with training data

    pred <- predict(linmod) # training response
    prednew <- predict(linmod, newdata=data.frame(X=X_test)) # test response

    trainerror[num] <- mean((y_train-pred)**2) # append MSE of training
    testerror[num] <- mean((y_test-prednew)**2) # append MSE of test
  }
  return (paste0("Expected train error: ", mean(trainerror),
                 ". Expected test error: ", mean(testerror),
                 "."))
}

simulation(1000,500,6)
```

```
## [1] "Expected train error: 0.992734722438706. Expected test error: 1.00714556375998."
```

```
simulation(50,100,10)
```

```
## [1] "Expected train error: 0.76788589432351. Expected test error: 1.29102683480489."
```

With the given output, we see that the expected training error is less than the expected test error. When we vary our inputs (N, M, p), we will continue to see that the expected test error is greater than or equal to the expected training error. This happens because there is always additional uncertainty surrounding new data.

Question 2

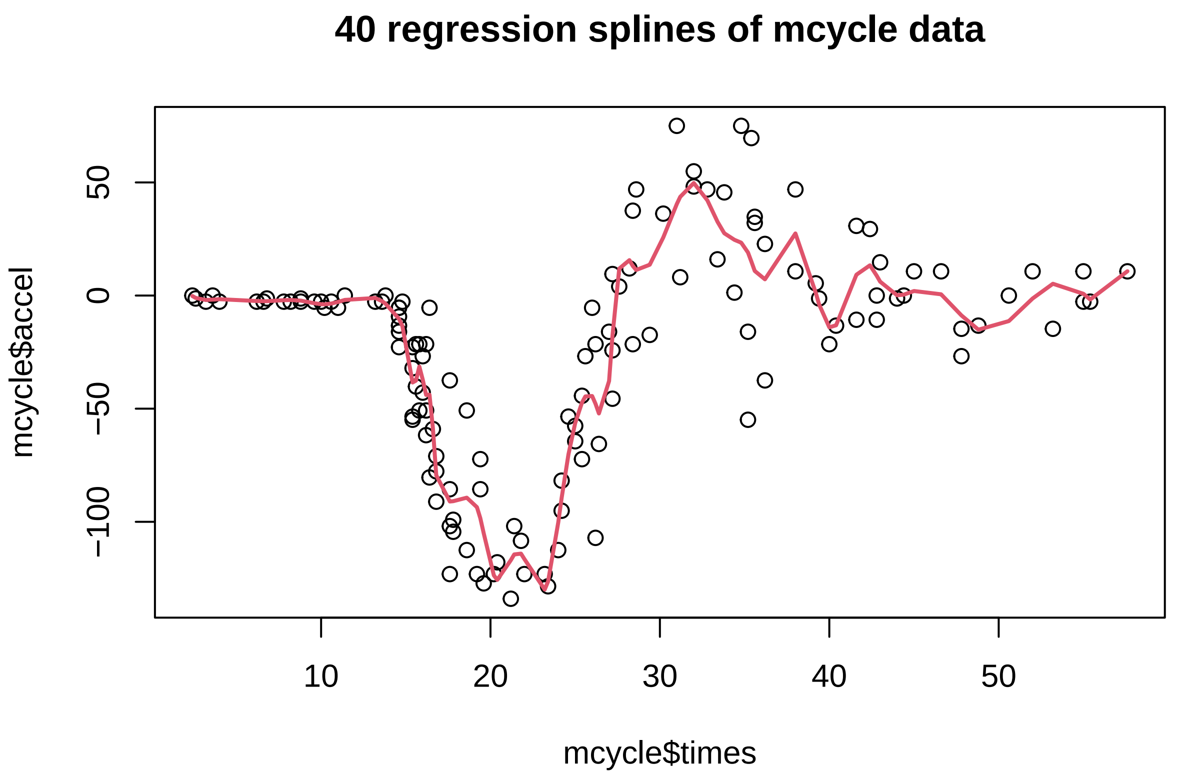
```
suppressMessages(library(MASS))
data(mcycle)
str(mcycle)
```

```
## 'data.frame':    133 obs. of  2 variables:
##  $ times: num  2.4 2.6 3.2 3.6 4 6.2 6.6 6.8 7.8 8.2 ...
##  $ accel: num  0 -1.3 -2.7 0 -2.7 -2.7 -2.7 -1.3 -2.7 -2.7 ...
```

a)

The function fit here begins to overfit the data because it looks as if it is absorbing the random error present in the data. The function is very bumpy, suggesting overfit to the data.

```
suppressMessages(library(splines))
regression_spline <- lm(accel ~ bs(times, 40), data=mcycle)
plot(mcycle$times, mcycle$accel, main = "40 regression splines of mcycle data")
lines(predict(regression_spline) ~ mcycle$times, col=2, lwd=2)
```



b)

In computing the adjusted R^2 statistic, we can pull this directly from the model.The adjusted R^2 is 0.7590244. This means that there is generally a good fit to the response using the predictors/basis functions.

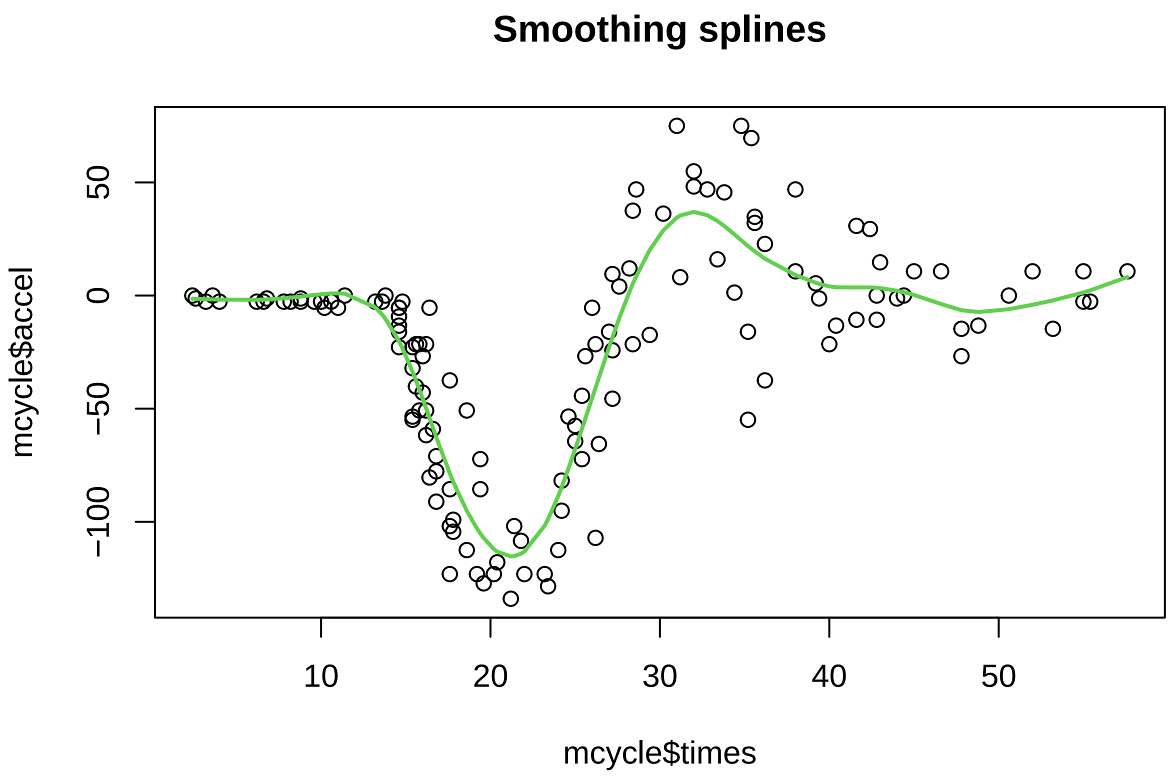
```
summary(regression_spline)$adj.r.squared
```

```
## [1] 0.7590244
```

c)

This fitted curve is smoother than the one in part a, which means that it is not overfitting to the data as much. Since it is not absorbing randomness of the data, then we should see a higher r-squared value which denotes a better fit to the response.

```
# automatically chooses a lambda value using cross-validation
smoothmod <- smooth.spline(mcycle$times, mcycle$accel)
plot(mcycle$times, mcycle$accel, main = "Smoothing splines")
lines(smoothmod, col=3, lwd = 2)
```



d)

The adjusted R^2 for the smoothing splines is higher than the one from the 40 basis functions, coming in at 0.7800681. This suggests that the 40 basis functions do a worse job of explaining the variability in the response variable. An important thing to note is that the first regression splines was limited to 40 basis functions, but since the smoothing splines explicitly has a penalty for too many parameters/basis functions, it fit a better model with less basis functions. However, this cross-validated fit was only 3 percentage points greater than the model with 40 basis functions. A better comparison would be to use new test data to evaluate the generalizability of the models.

```r
yhat <- predict(smoothmod, x=mcycle$times)$y
RSS <- sum((yhat-mcycle$accel)**2)
TSS <- sum((mcycle$accel - mean(mcycle$accel))**2)
(R2a = 1- (RSS/(length(mcycle$accel)-smoothmod$df))/(TSS/(length(mcycle$accel)-1)))
```

```
## [1] 0.7800681
```

Question 3

```r
suppressMessages(library(faraway))
data(aatemp); str(aatemp)
```

```
## 'data.frame':    115 obs. of  2 variables:
##  $ year: int  1854 1855 1871 1881 1882 1883 1884 1885 1890 1891 ...
##  $ temp: num  49.1 46.5 48.8 48 47.3 ...
```

```r
min(aatemp$year); max(aatemp$year)
```

```
## [1] 1854

## [1] 2000
```

a)

The slope of the line is 0.0122373 with a standard error of 0.0037683. At a 95% confidence, the critical value should be 1.96.

- The null hypothesis is that the slope is 0. Rejecting the null would mean going outside of the range [-1.96,1.96] set by the critical value.
- With the test statistic being the z-value, calculated as $(slope - 0)/\sigma$, the calculated z-value is $0.0122373/0.0037683 = 3.247433$.

- We can reject the null hypothesis that the slope is 0, concluding that it is different from 0.

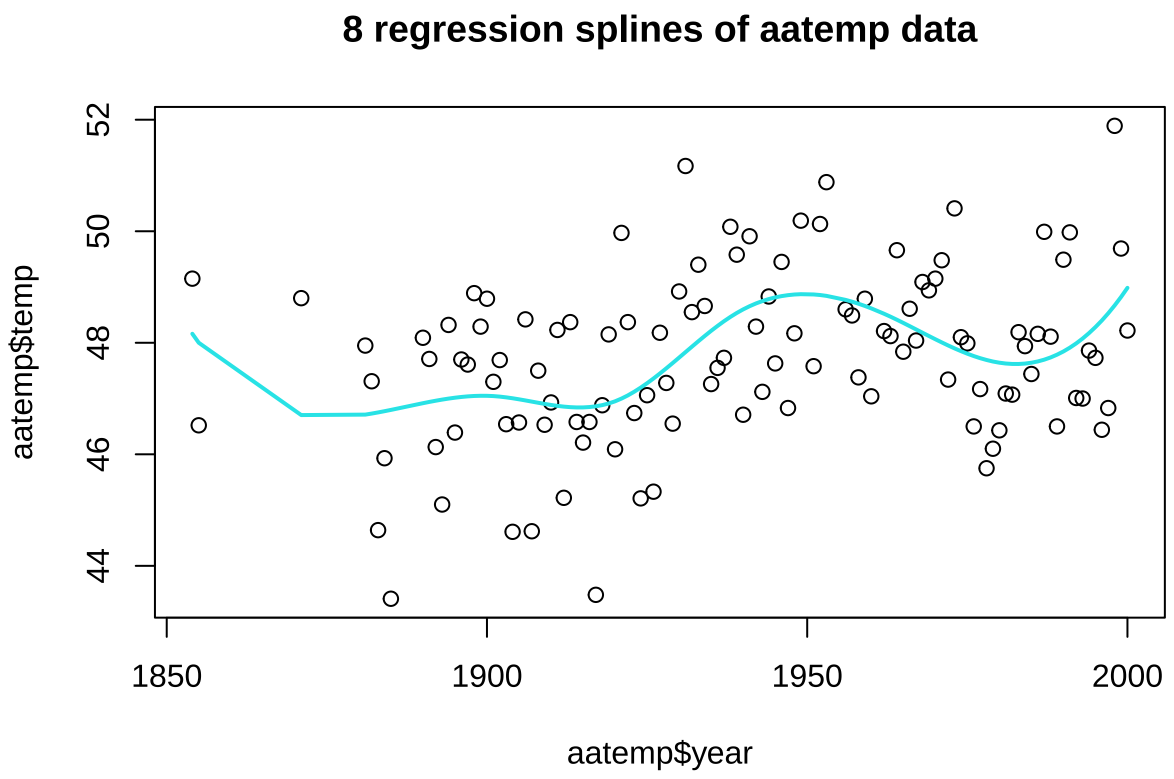
The main drawback of this linear fit is the very low R^2 value of 0.09. This suggests that the covariates do not explain the variability of the response well.

```r
linmod <- lm(temp ~ year, data = aatemp)
sumary(linmod)
```

```
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept) 24.0055103  7.3107806  3.2836 0.001365
## year         0.0122373  0.0037683  3.2474 0.001533
##
## n = 115, p = 2, Residual SE = 1.46626, R-Squared = 0.09
```

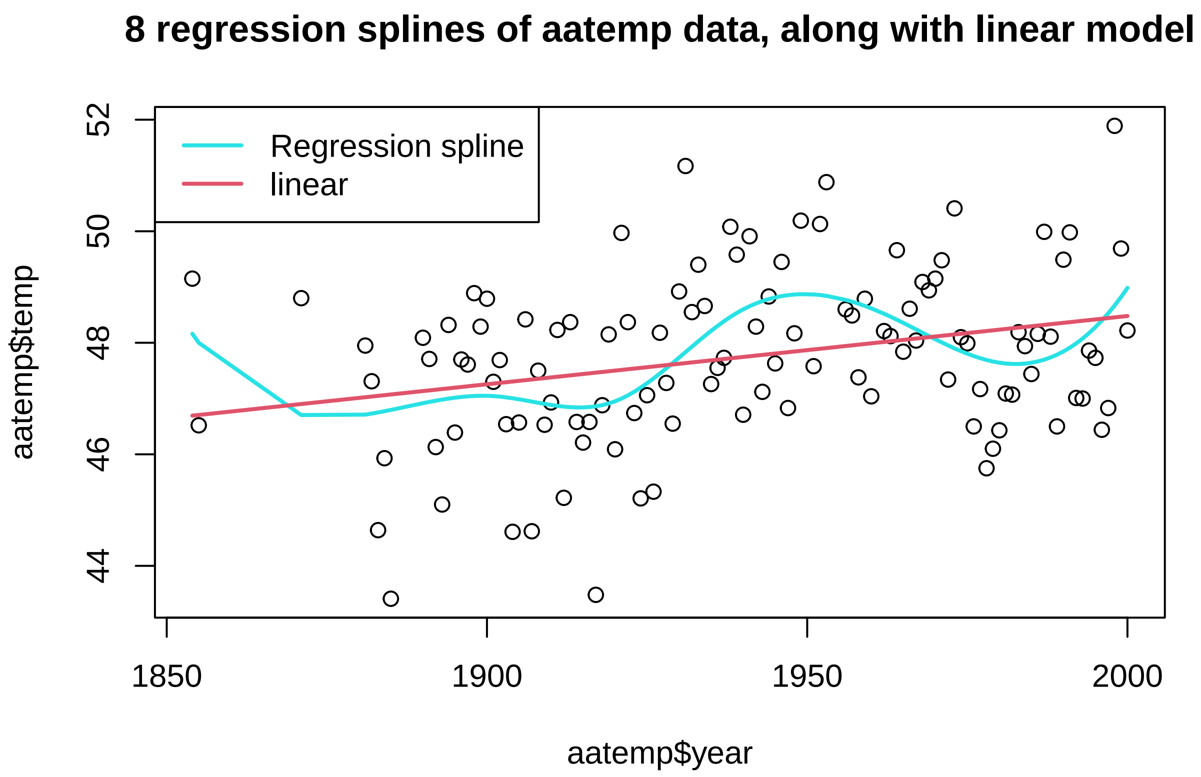
b)

```r
suppressMessages(library(splines))
regression_spline <- lm(temp ~ bs(year, 8), data=aatemp)
plot(aatemp$year, aatemp$temp, main = "8 regression splines of aatemp data")
lines(predict(regression_spline) ~ aatemp$year, col=5, lwd=2)
```



c)

```
plot(aatemp$year, aatemp$temp,
     main = "8 regression splines of aatemp data, along with linear model")
lines(predict(regression_spline) ~ aatemp$year, col=5, lwd=2)
lines(predict(linmod)~ aatemp$year, col=2, lwd =2)
legend('topleft', c('Regression spline', 'linear'), lty=c(1,1), col=c(5,2), lwd=c(2,2))
```



d)

$$F = \frac{(SSE_{Mod1} - SSE_{Mod2})/(p-q)}{SSE_{Mod2}/(n-p)} \sim Fp - q, n - p$$

```r
anova(linmod, regression_spline)
```

```
## Analysis of Variance Table
##
## Model 1: temp ~ year
## Model 2: temp ~ bs(year, 8)
##   Res.Df    RSS Df Sum of Sq      F  Pr(>F)
## 1    113 242.94
## 2    106 207.68  7    35.257 2.5707 0.01736 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The null hypothesis is that the linear model fits the data best. Since our F-statistic has been calculated to have a p-value of under 0.05, at a 95% confidence, we can conclude that the regression splines is better fit to the response data.

We can also explore metrics like r-squared to assess the goodness of fit of the regression splines. We can also explore the performance to new data, which can show if the greater fit is due to overfitting or not. The r-squared, calculated below, is better than the linear model at 0.22. This is an improvement, but still does not capture the response effectively.

On a side note, the data varies above and below the mean quite a lot, which explains why the models we have fit above do not adequately explain the variability in the response.

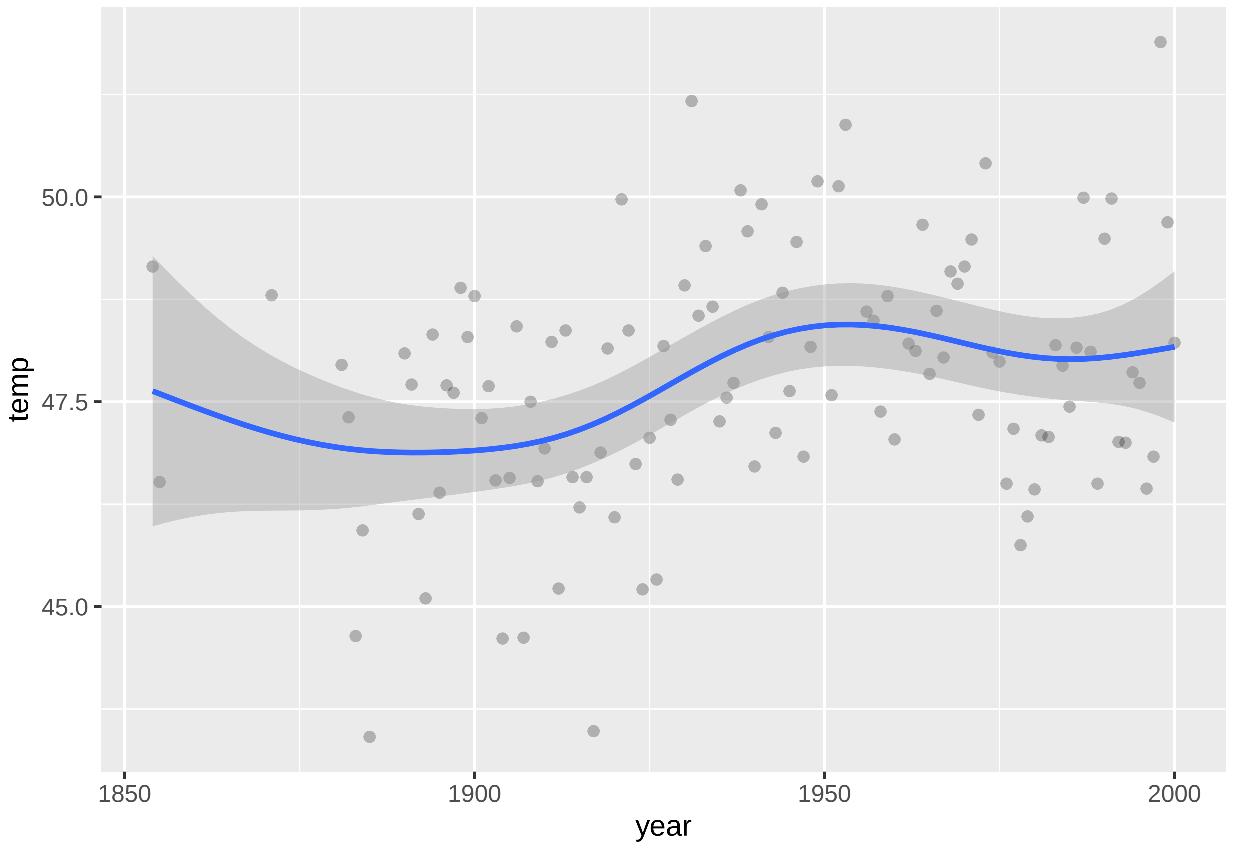
```r
sumary(regression_spline)
```

```
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    48.16111    1.02472 46.9994   <2e-16
## bs(year, 8)1 -2.60718    2.29186 -1.1376   0.2579
## bs(year, 8)2 -0.35825    1.44222 -0.2484   0.8043
## bs(year, 8)3 -1.81239    1.26304 -1.4349   0.1542
## bs(year, 8)4  1.00872    1.23799  0.8148   0.4170
## bs(year, 8)5  0.63616    1.34390  0.4734   0.6369
## bs(year, 8)6 -0.95826    1.40941 -0.6799   0.4980
## bs(year, 8)7 -0.25047    1.47333 -0.1700   0.8653
## bs(year, 8)8  0.82392    1.36778  0.6024   0.5482
##
## n = 115, p = 9, Residual SE = 1.39974, R-Squared = 0.22
```

e)

I would say that there is a pattern of temperatures in that there are peaks and troughs throughout the annual mean temperatures between 1854 and 2000. However, the peaks don't necessarily go towards the same value, nor do the troughs. There is a slight shift upwards of the entire data, where the peaks and troughs are both higher in later years.

```r
suppressMessages(library(mgcv))
suppressMessages(library(ggplot2))
ggplot(aatemp, aes(x=year,y=temp)) +
  geom_point(alpha=0.25) + geom_smooth(method="gam", formula=y ~ s(x))
```



Question 4

```
suppressMessages(library(faraway))
data(prostate)
str(prostate)
```

```
## 'data.frame':    97 obs. of  9 variables:
##  $ lcavol : num  -0.58 -0.994 -0.511 -1.204 0.751 ...
##  $ lweight: num  2.77 3.32 2.69 3.28 3.43 ...
##  $ age    : int  50 58 74 58 62 50 64 58 47 63 ...
##  $ lbph   : num  -1.39 -1.39 -1.39 -1.39 -1.39 ...
##  $ svi    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ lcp    : num  -1.39 -1.39 -1.39 -1.39 -1.39 ...
##  $ gleason: int  6 6 7 6 6 6 6 6 6 6 ...
##  $ pgg45  : int  0 0 20 0 0 0 0 0 0 0 ...
##  $ lpsa   : num  -0.431 -0.163 -0.163 -0.163 0.372 ...
```

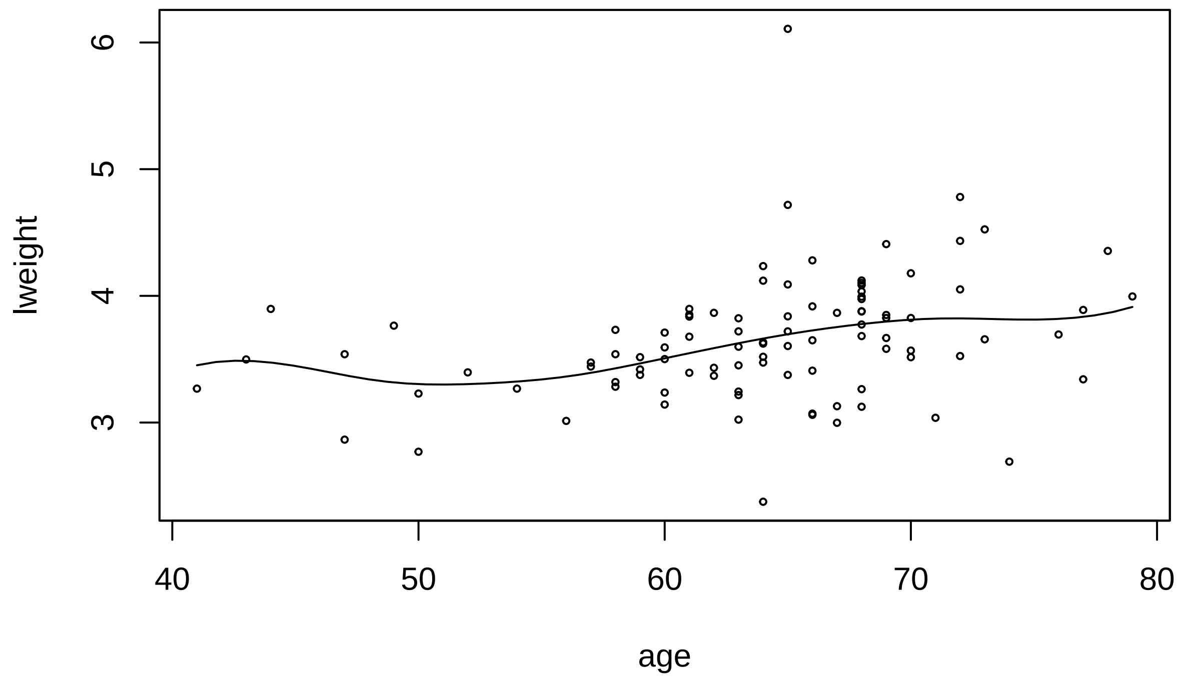
a)

Using a normal kernel function, the outlier shifts the entire graph upwards towards it. However, since the outlier does not have high leverage, I would not consider it influential. The fit of the data does look linear, but if a linear model was fit, the outlier point would have a larger effect on the line than it has on the kernel fit.

```
suppressMessages(library(sm))
```

```
## Warning: package 'sm' was built under R version 4.4.3
```

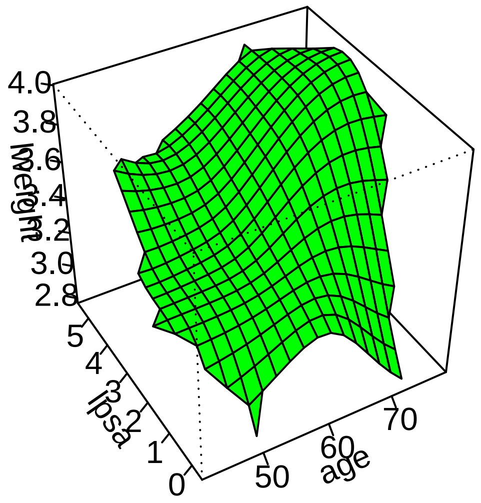
```
with(prostate,sm.regression(age, lweight, h=h.select(age,lweight)))
```



b)

With an additional predictor, we can see an additional effect on the fit on the response variable. The general shape of age on lweight is still present, but it appears lpsa interacts with age in the plot. The two variables jointly effect the response.

```r
age <- prostate$age
lpsa <- prostate$lpsa
lweight <- prostate$lweight
sm.regression(cbind(age,lpsa), lweight,
              h=h.select(cbind(age,lpsa),lweight))
```



Question 5

```r
suppressMessages(library(ISLR))
data(OJ)
str(OJ)
```

```
## 'data.frame':    1070 obs. of  18 variables:
##  $ Purchase      : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 1 1 1 1 1 ...
##  $ WeekofPurchase: num  237 239 245 227 228 230 232 234 235 238 ...
##  $ StoreID       : num  1 1 1 1 7 7 7 7 7 7 ...
##  $ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.99 1.99 1.99 1.99 1.99 ...
##  $ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
##  $ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
##  $ SpecialCH     : num  0 0 0 0 0 0 1 1 0 0 ...
##  $ SpecialMM     : num  0 1 0 0 0 1 1 0 0 0 ...
##  $ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
##  $ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...
##  $ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
##  $ Store7        : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 2 2 ...
##  $ PctDiscMM     : num  0 0.151 0 0 0 ...
##  $ PctDiscCH     : num  0 0 0.0914 0 0 ...
##  $ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
##  $ STORE         : num  1 1 1 1 0 0 0 0 0 0 ...
```

```
set.seed(123)
```

a)

```
data = OJ
len_x = dim(data)[1]
set.seed(123)
index_ran = sample(1:len_x, len_x)
train_size = 800
train = data[index_ran[1:train_size],]
test = data[index_ran[(train_size+1):len_x],]

x_train = train
x_test = test
```

b)

There are 8 nodes to the tree, there were 2 variables used in construction, and the training error rate is 0.165.

```
suppressMessages(library(tree))
```

```
## Warning: package 'tree' was built under R version 4.4.3
```

```
tree.OJ <- tree(Purchase~., data=x_train)
summary(tree.OJ)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = x_train)
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7625 = 603.9 / 792
## Misclassification error rate: 0.165 = 132 / 800
```

c)

For the terminal node 11, starting at the top, LoyalCH is under 0.5036, so we go to the left. Then, since LoyalCH is over 0.276142, we go to the right. Then, since PriceDiff is greater than 0.05, then we categorize as CH. Buying CH orange juice in the end is when someone is moderately loyal to CH (under 50% of the time) and when MM is more expensive than CH by more than 5 cents. Otherwise, someone who is less loyal to CH will get minute maid most of the time.

```
tree.OJ
```

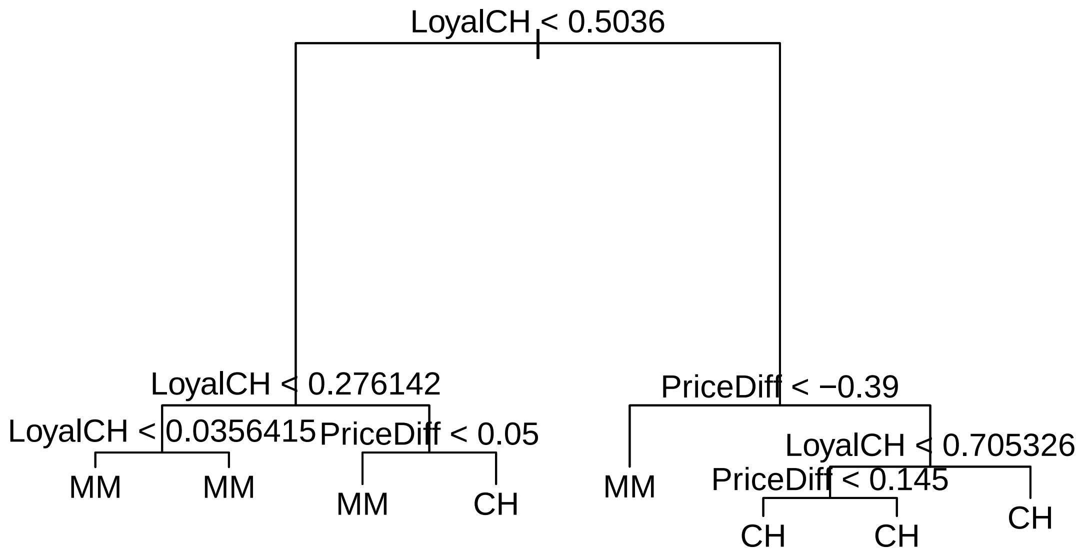
```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1071.00 CH ( 0.60875 0.39125 )
##    2) LoyalCH < 0.5036 350  415.10 MM ( 0.28000 0.72000 )
##      4) LoyalCH < 0.276142 170  131.00 MM ( 0.12941 0.87059 )
##        8) LoyalCH < 0.0356415 56   10.03 MM ( 0.01786 0.98214 ) *
##        9) LoyalCH > 0.0356415 114  108.90 MM ( 0.18421 0.81579 ) *
##      5) LoyalCH > 0.276142 180  245.20 MM ( 0.42222 0.57778 )
##       10) PriceDiff < 0.05 74   74.61 MM ( 0.20270 0.79730 ) *
##       11) PriceDiff > 0.05 106  144.50 CH ( 0.57547 0.42453 ) *
##    3) LoyalCH > 0.5036 450  357.10 CH ( 0.86444 0.13556 )
##      6) PriceDiff < -0.39 27   32.82 MM ( 0.29630 0.70370 ) *
##      7) PriceDiff > -0.39 423  273.70 CH ( 0.90071 0.09929 )
##       14) LoyalCH < 0.705326 130  135.50 CH ( 0.78462 0.21538 )
##         28) PriceDiff < 0.145 43   58.47 CH ( 0.58140 0.41860 ) *
##         29) PriceDiff > 0.145 87   62.07 CH ( 0.88506 0.11494 ) *
##       15) LoyalCH > 0.705326 293  112.50 CH ( 0.95222 0.04778 ) *
```

d)

The tree sorts observations into MM if LoyalCH is under 0.5036 AND PriceDiff is under 0.05. The only time it sorts into MM when LoyalCH is over 0.5036 is when PriceDiff is under -0.39, which is also under 0.05. Otherwise, it sorts into CH.

For someone who is loyal to CH, they will only buy MM if it is cheaper than CH by more than 39 cents. For someone who is less loyal to CH, they will only buy CH if it is cheaper by more than 5 cents and they have a moderate loyalty to CH (~30%-50%).

```
plot(tree.OJ)
text(tree.OJ,pretty=0)
```



e)

```
pred <- predict(tree.OJ, newdata=x_test, type="class")
xtabs(~pred + x_test$Purchase)
```

```
##     x_test$Purchase
## pred  CH  MM
##   CH 150  34
##   MM  16  70
```

```
(34+16)/(150+34+16+70) # test error rate
```

```
## [1] 0.1851852
```

f)

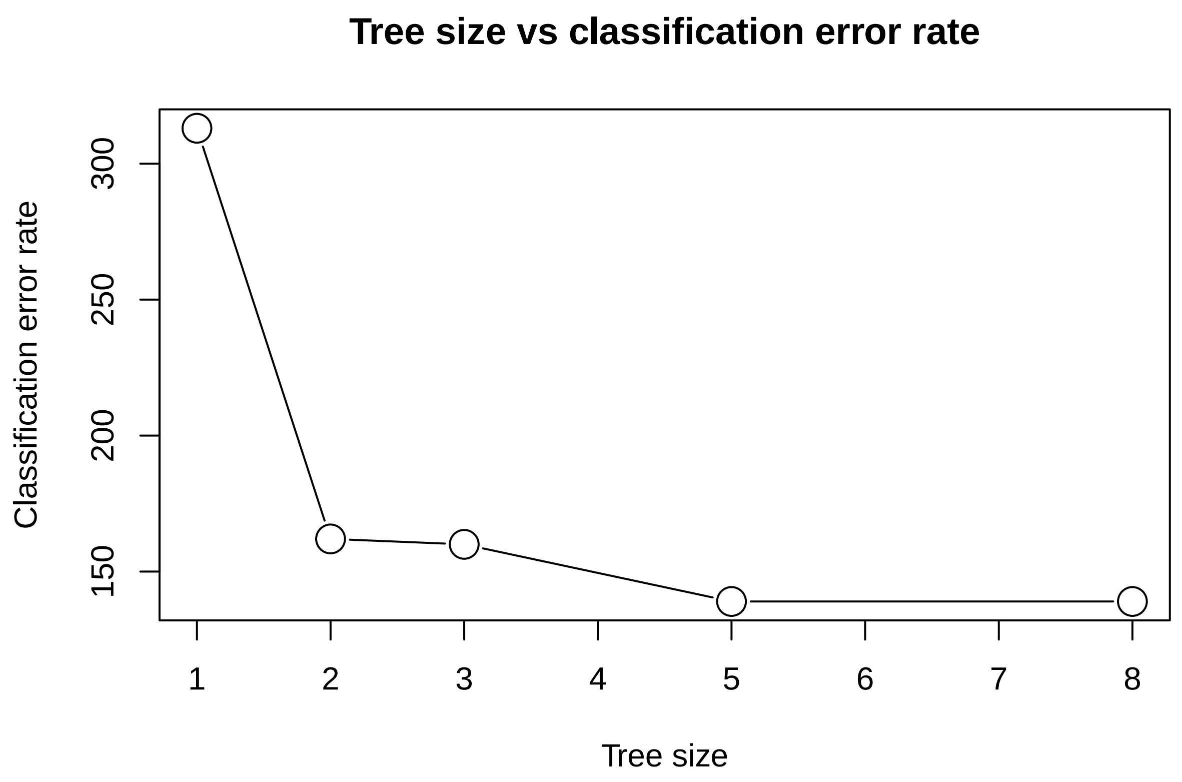
The optimal tree size is with 5 nodes or 8 nodes. However, since we already have a tree with 8 nodes, it would be best to choose 5 nodes here.

```
cv.oj <- cv.tree(tree.OJ, FUN = prune.misclass)
cv.oj$size[which(cv.oj$dev == min(cv.oj$dev))]
```

```
## [1] 8 5
```

g)

```
plot(cv.oj$size, cv.oj$dev,type='b', cex=2,
     ylab="Classification error rate", xlab = "Tree size",
     main = "Tree size vs classification error rate")
```

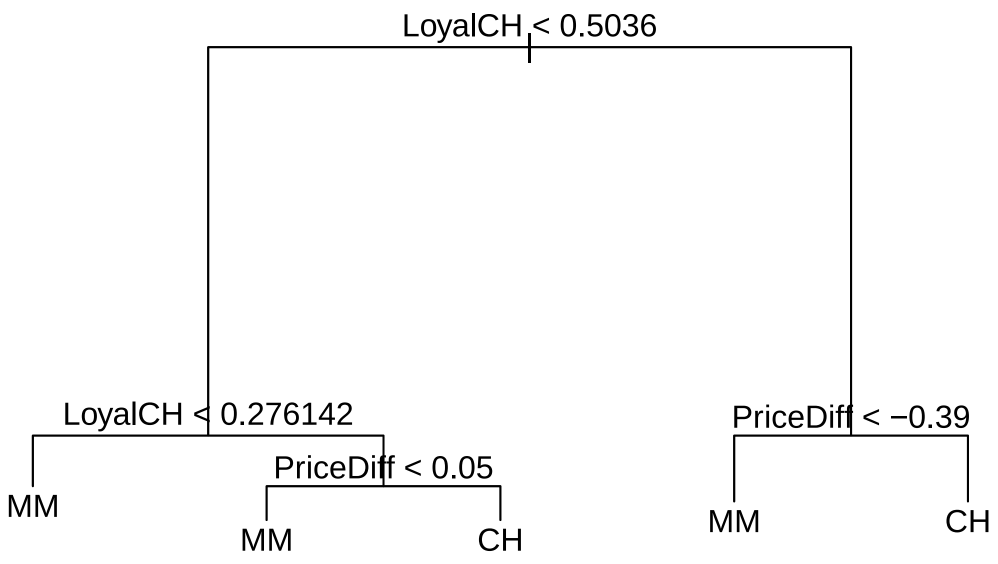


h)

The optimal tree size is 5 or 8, but since our original tree had 8 nodes, we should choose the tree with 5 nodes.

i)

```r
tree.p <- prune.misclass(tree.OJ, best=5)
plot(tree.p)
text(tree.p)
```



j)

The training error rate of both trees is the same because of the behavior of the first tree where one branch would eventually lead to the same classification, but chose to continue splitting. This is seen on the right side after the PriceDiff < -0.39 split and on the left side after the LoyalCH < 0.276142 split.

```r
summary(tree.p)
```

```
##
## Classification tree:
## snip.tree(tree = tree.OJ, nodes = c(4L, 7L))
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  5
## Residual mean deviance:  0.826 = 656.6 / 795
## Misclassification error rate: 0.165 = 132 / 800
```

```r
summary(tree.OJ)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = x_train)
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7625 = 603.9 / 792
## Misclassification error rate: 0.165 = 132 / 800
```

k)

The test error rates are thus the same as well because the conclusions are the same across both pruned and unpruned trees. Again, the structure of the trees did not change, only the number of terminal nodes.

```
pred <- predict(tree.OJ, newdata=x_test, type="class")
xtabs(~pred + x_test$Purchase)
```

```
##     x_test$Purchase
## pred  CH  MM
##   CH 150  34
##   MM  16  70
```

```
(34+16)/(150+34+16+70) # test error rate for unpruned
```

```
## [1] 0.1851852
```

```
pred2 <- predict(tree.p, newdata=x_test, type="class")
xtabs(~pred2 + x_test$Purchase)
```

```
##      x_test$Purchase
## pred2  CH  MM
##    CH 150  34
##    MM  16  70
```

```
(34+16)/(150+34+16+70) # test error rate for pruned
```

```
## [1] 0.1851852
```

Question 6

```
set.seed(1)
suppressMessages(library(Ecdat))
```

```
## Warning: package 'Ecdat' was built under R version 4.4.3
```

```
## Warning: package 'Ecfun' was built under R version 4.4.3
```

```
Housing$price <- log(Housing$price)
trainH <- sample(1:nrow(Housing),nrow(Housing)/2)
str(Housing[trainH,])
```

```
## 'data.frame':    273 obs. of  12 variables:
##  $ price   : num  11.7 11.4 11 10.9 11 ...
##  $ lotsize : num  4560 8372 2145 5450 4632 ...
##  $ bedrooms: num  3 3 3 2 4 3 4 3 2 3 ...
##  $ bathrms : num  2 1 1 1 1 1 2 1 1 1 ...
##  $ stories : num  2 3 3 1 2 1 2 1 1 1 ...
##  $ driveway: Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ recroom : Factor w/ 2 levels "no","yes": 2 1 1 1 1 1 2 1 1 1 ...
##  $ fullbase: Factor w/ 2 levels "no","yes": 2 1 1 1 1 2 2 1 1 1 ...
##  $ gashw   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ airco   : Factor w/ 2 levels "no","yes": 2 2 1 1 2 1 1 2 1 1 ...
##  $ garagepl: num  1 2 1 0 0 0 1 2 1 1 ...
##  $ prefarea: Factor w/ 2 levels "no","yes": 1 1 2 1 1 1 1 1 1 1 ...
```

```
housing.test <- Housing[-trainH,"price"]
```

Linear regression

```
linmod <- lm(price~.,data=Housing[trainH,])
yhat <- predict(linmod, newdata=Housing[-trainH,])
mean((yhat - housing.test)**2) # test MSE = 0.04133829
```

```
## [1] 0.04133829
```

Boosting

```
suppressMessages(library(gbm))
```

```
## Warning: package 'gbm' was built under R version 4.4.3
```

```
set.seed(1)
boost.housing <- gbm(price~., data=Housing[trainH,],
                     distribution="gaussian", n.trees=5000,
                     interaction.depth=4)
yhat.boost <- predict(boost.housing,
                      newdata=Housing[-trainH,],n.trees=5000)
mean((yhat.boost-housing.test)**2) # test MSE = 0.08305532
```

```
## [1] 0.08305532
```

Bagging

```
suppressMessages(library(randomForest))
```

```
## Warning: package 'randomForest' was built under R version 4.4.3
```

```
set.seed(1)
bag.housing <- randomForest(price~., data=Housing[trainH,],
                            mtry=11, importance=TRUE)
bag.housing
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = Housing[trainH, ], mtry = 11,      importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 11
##
##           Mean of squared residuals: 0.06449588
##                     % Var explained: 56.81
```

```
yhat.bag <- predict(bag.housing, newdata=Housing[-trainH,])
mean((yhat.bag-housing.test)**2) # test MSE = 0.04739514
```

```
## [1] 0.04739514
```

Random forest

```
set.seed(1)
rf.housing <- randomForest(price~.,data=Housing[trainH,],
                           importance=TRUE)
rf.housing
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = Housing[trainH, ], importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 0.05819628
##                     % Var explained: 61.02
```

```
yhat.rf <- predict(rf.housing, newdata=Housing[-trainH,])
mean((yhat.rf - housing.test)**2) # test MSE = 0.04318891
```

```
## [1] 0.04318891
```

Interpretation

The best performing nonparametric model was the random forest model with a test MSE of 0.04318891. In contrast, the linear regression had a test MSE of 0.04133829, which is only slightly better than the random forest model. The worst performing model was the boosting model, where the test MSE was 0.08305532. The bagging model was similar to the random forest with a test MSE of 0.04739514.

In the end, the model that won out was the linear regression, so in the real world, there may not have been a use for the random forest model since it is more computationally heavy.

Question 7

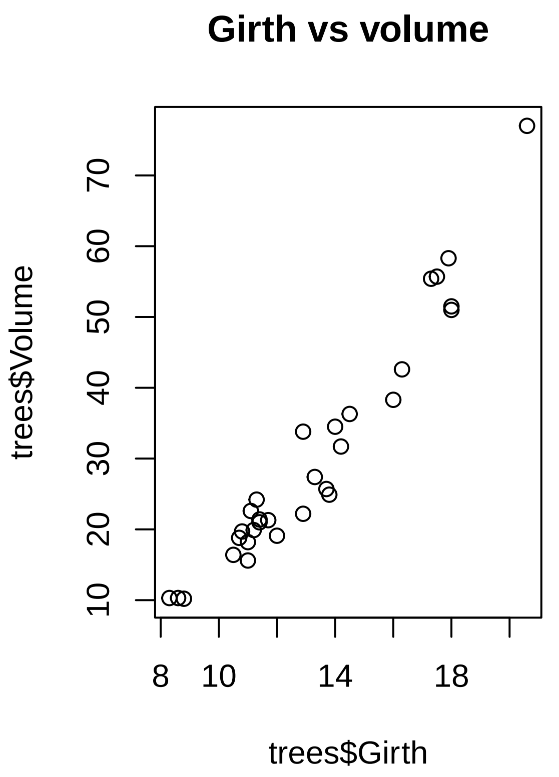
```
suppressMessages(library(faraway))
head(trees,3)
```

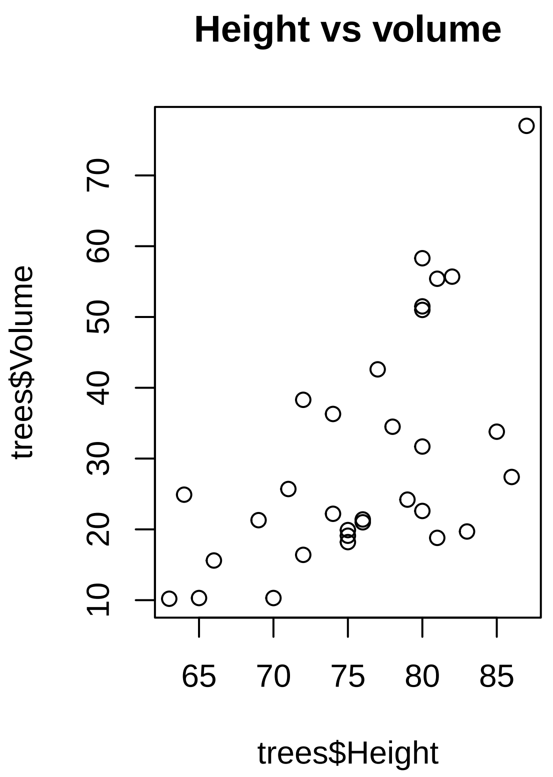
```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
```

a)

From the plots below, the relationship between the two predictors and the response appears generally linear. Girth has more of a linear relationship than height though.

```
par(mfrow=c(1,2))
plot(trees$Girth, trees$Volume, main = "Girth vs volume")
plot(trees$Height, trees$Volume, main = "Height vs volume")
```





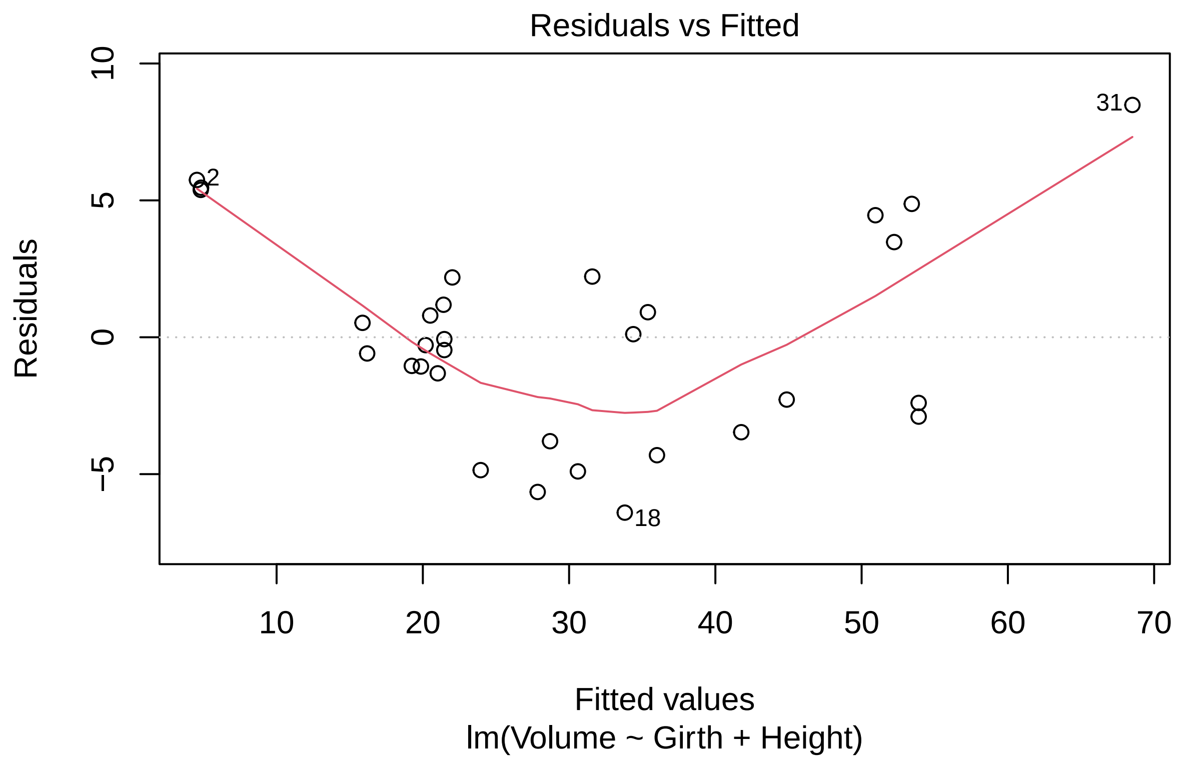
b)

There appears to be a clustering of the points on the residuals vs fitted values that is creating an uneven plot. On the ends of the fitted values, the residuals become larger. The R-squared is 0.95.

```
treemod <- lm(Volume ~ Girth + Height, data=trees)
sumary(treemod)
```

```
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -57.98766    8.63823 -6.7129  2.75e-07
## Girth         4.70816    0.26426 17.8161 < 2.2e-16
## Height        0.33925    0.13015  2.6066   0.01449
##
## n = 31, p = 3, Residual SE = 3.88183, R-Squared = 0.95
```

```
plot(treemod, which=1)
```



c)

```
scaledtrees <- scale(trees)
```

The r-squared is greater than the one from the linear model. There is a 3.29675% difference between the two r-squared values.

```
suppressMessages(library(nnet))
bestrss <- 10000
set.seed(1)
for(i in 1:100){
  nnmdl <- nnet(Volume ~ Girth + Height, scaledtrees, size=2, linout=T, trace=F)
  if(nnmdl$value < bestrss){
    bestnn <- nnmdl
    bestrss <- nnmdl$value
    }}
# calculate the r-squared value
1-bestnn$value/sum((scaledtrees[,1]-mean(scaledtrees[,1]))**2)
```

```
## [1] 0.9829675
```

d)

```
scaledtrees <- data.frame(scaledtrees)
newtrees <- data.frame(Height=seq(-2,2,length=length(scaledtrees$Height)),
                       Girth=rep(0,length(scaledtrees$Girth)))
```

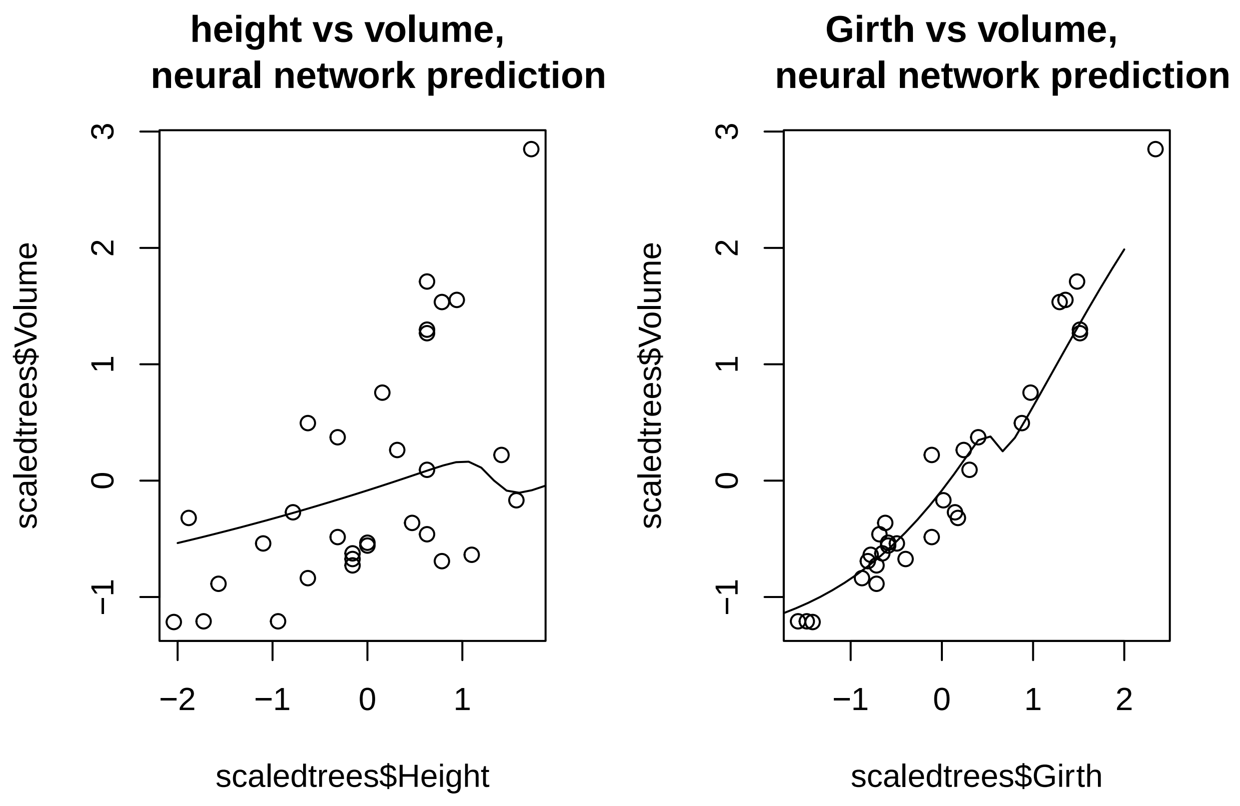
```r
newtrees2 <- data.frame(Girth=seq(-2,2,length=length(scaledtrees$Girth)),
                        Height=rep(0,length(scaledtrees$Height)))
```

The shape of the curve with height performs okay to predict the data, but falters at the right side of the plot. The fit increases where it should and does its best to take into account the data's behaviour.

The shape of the curve with girth performs well to not overfit the cluster of points around the mean and follows a smooth line up towards the rest of the data.

Both curves are smooth and continuous, which shows that the neural network fit to the data well.

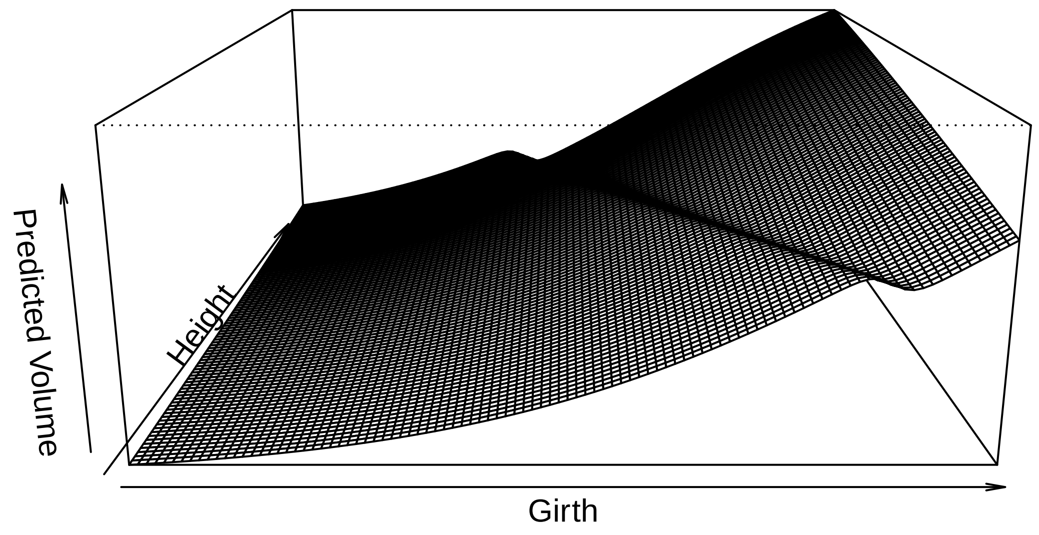
```r
par(mfrow=c(1,2))
pred <- predict(bestnn, newdata=newtrees)
plot(scaledtrees$Height, scaledtrees$Volume, main="height vs volume,
     neural network prediction")
lines(newtrees$Height, pred)

pred2 <- predict(bestnn, newdata=newtrees2)
plot(scaledtrees$Girth, scaledtrees$Volume, main="Girth vs volume,
     neural network prediction")
lines(newtrees2$Girth, pred2)
```



e)

```r
girth <- seq(-2,2,length=100)
height <- seq(-2,2,length=100)
perspplotting <- expand.grid(Girth=girth, Height=height)
```

```r
predpersp <- predict(bestnn, newdata=persppplotting)
predicted_volume <- matrix(predpersp, nrow=length(girth), ncol = length(height))
persp(girth,height, predicted_volume, xlab = "Girth", ylab = "Height",
      zlab = "Predicted Volume", expand = 0.4)
```


Predicted Volume
Height
Girth