Stat 154 Problem Set Three

Jinze Gu SID:24968967

February 28, 2014

Problem One

On the page I attached in the hand written parts.

Problem Two

Basically, I used three methods to tackle the problem, namely, linear regression, maximizing loglikelihood and k nearest neighborhood method. Please check the plot for comparing the test error of different methods.

Method One: Optimizing loglikelihood

For optimizing loglikelyhood, we need to compute the loglikelyhood function and then optimize the function in order to attain the estimate of beta. In this situation, we denote digit Two as 0 and digit Three as 1. Then we can construct the likelyhood function based on binomial distribution and do optimization. It turns out the error rate for predicting test data of TWO is 0.0404, the error rate of predicting test data THREE is 0.048, and the total error rate is which is acceptable for me, and the general error rate is about 0.044. Besides, the error rate of predicting training data set is 0 for both numbers, which is perfect and reasonable since the likelyhood under optimized data is maximized to almost 1.

Method Two: Linear regression

I used the linear regression to fit the model without origin point since the image of both zipcode 2 and zipcode 3 have no information in the origin. After I got the fitted value of Y, we use the maximum fitted value of X from group two and the minimum of fitted value of X from group three as the threshold to classify the point. The training error for 2 is 0.055 the training error for 3 is 0.0015 and the total training error is around 0.028. On the other hand, the test error for digit two is 0.034 and the test error for three is 0.006; the total test error of linear prediction error is 0.08.

Method Three: k nearest neighbors

For k nearest neighborhood method, I used the built function knn in the class package to classify the data. It turns out that the training error for knn classifier is 0 when k is 1 and increases as k increases. When, k = 1, the test error of knn classifier is better than linear classifier since the error rate of testing 2 is 0.03, the error rate of testing 3 is 0.018 and the general error rate is 0.024 when k = 1. Similarly, the test error also increases as k increases, the further information is illustrated by the graph.

Problem 3

Obviously, pointwise confidence interval is wider but the parameter wise confidence interval is more strict. The point wise confidence interval is achieved by finding the estimated sigma squared in the fit and constructing the confidence interval using t distribution. The parameter wise confidence interval is constructed by the equation of book (3.15); note that we can construct chisquared distribution for each β using the diagnal element of $X^t X$ and cholesky decomposition($M^t M$ in the code), basically, I can solve for the confidence interval by finding the 95 quantile of chisquare with degree of freedom as p + 1 and get the confidence interval of β.

Problem 4

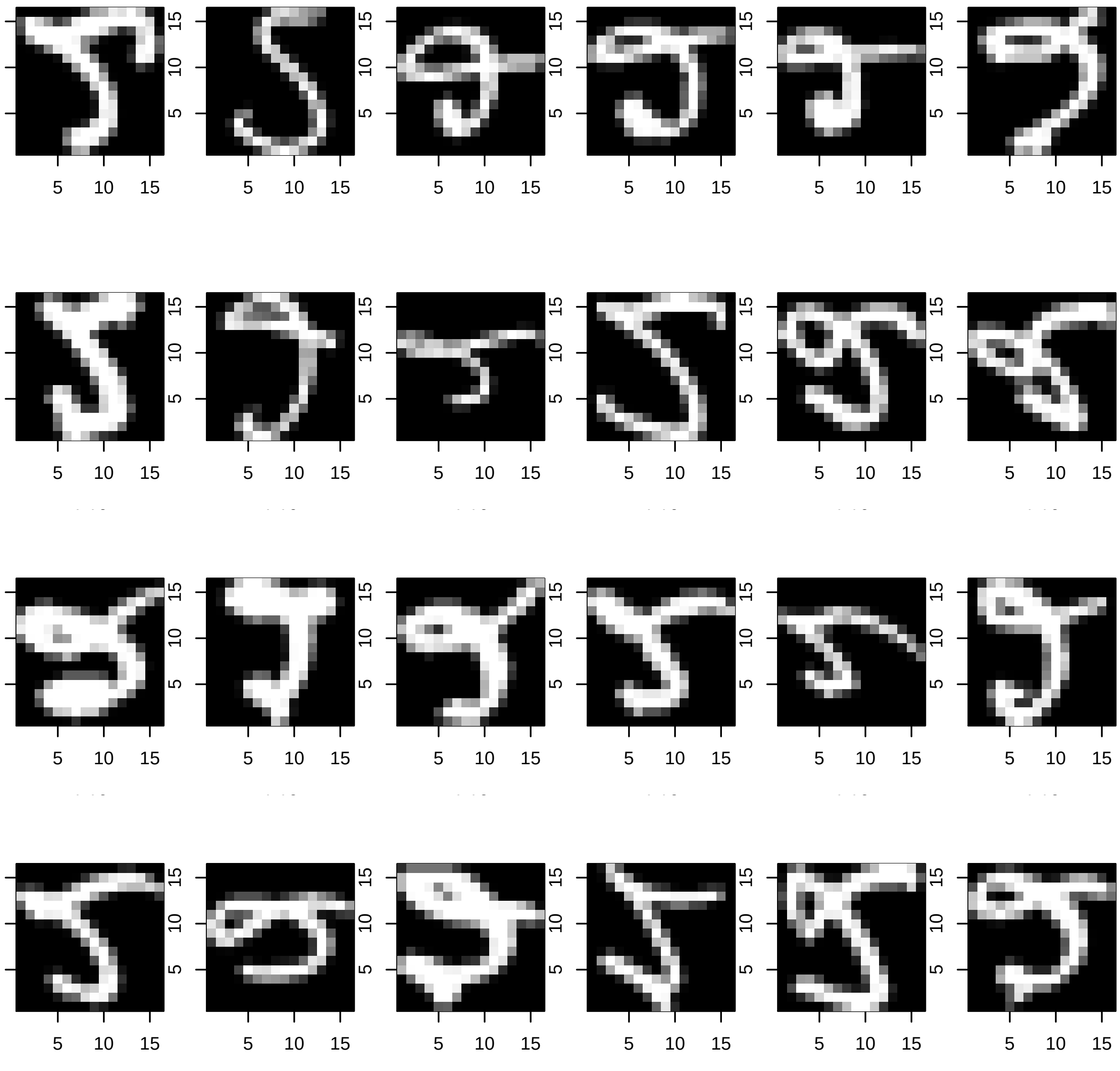
On the attached hand written page.

Problem 5

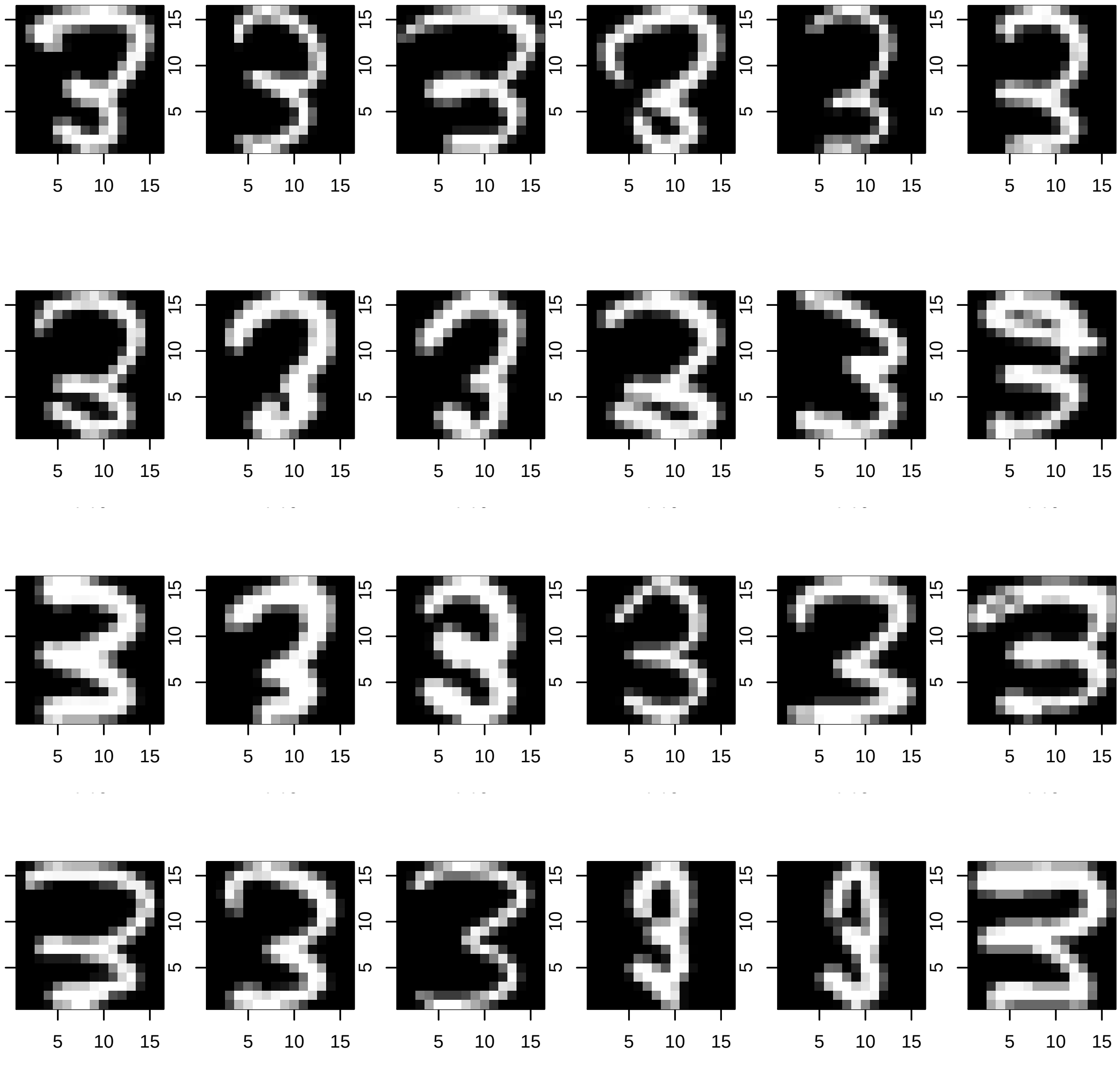
First of all, I plotted data and it turns out that I cannot really find out the linear relationship visually, so I decided to fit a naive regression model and find out the the leverage pts based on hat matrix, and it turns out that African elephant is the first leverage pt that I have discovered. Then I made the diagnostic plot of the naive fit, It turns out that human and Asian elephant are the other two leverage pts. Then I decided to remove these three points and look at the plot. Since there sort of a linear relationship, then I fitted the second linear model, but it turns out that the normality plot is not good. Thus, I decided to do transform the data in logscale and it turns out there is an obvious linear relationship based on scatter plot, and the diagnostic plot also indicates the linear fit is decent.

Appendix One: code for problem two

```
setwd("/Volumes/æIJĽèČ¡åĞžæšą/Stat 154/HW_3")
TWO <- read.csv("train2.txt", header = FALSE)
Two.digit <- as.matrix(TWO)
THREE <- read.csv("train3.txt", header = FALSE)
Three.digit <- as.matrix(THREE)
Test <- read.csv("~/Downloads/test.txt", sep = " ")
Test.two <- as.matrix(Test[which(Test[, 1] == 2), 2:257])
Test.three <- as.matrix(Test[which(Test[, 1] == 3), 2:257])
test <- rbind(Test.two, Test.three)
names(Test) <- NULL
Test <- as.matrix(Test)
colors <- c("black", "white")
cus_col <- colorRampPalette(colors = colors)
par(mfrow = c(4, 6), pty = "s", mar = c(1, 1, 1, 1))
for (i in 1:24) {
    image(1:16, 1:16, matrix(Two.digit[i, ], ncol = 16), col = cus_col(256))
}
```



```r
for (i in 1:24) {
    image(1:16, 1:16, matrix(Three.digit[i, ], nrow = 16), col = cus_col(256))
}
```



```r
# I assigned Two as 0, and Three as 1
Y <- rep(1, nrow(Three.digit) + nrow(Two.digit))
Y[1:nrow(Two.digit)] <- 0
X <- rbind(Two.digit, Three.digit)
# Method One, Optimize log_likelyhood
log_like <- function(beta) {
    first <- sum(t(Y) %*% X %*% beta)
    second <- sum(log(1 + exp(X %*% beta)))
    return(first - second)
}
optimal_1 <- optim(rep(1, 256), fn = log_like, method = "BFGS", control = list(fnscale = -1))
log_like_class <- function(pred, beta) {
    prob <- 1/(1 + exp(-pred %*% beta))
    vote <- prob
```

```r
    vote[vote > 0.5] <- 3
    vote[vote <= 0.5] <- 2
    return(data.frame(prob = prob, prediction = vote))
}
# Now I can test the effect of classfier using training data
training2 <- log_like_class(Two.digit, optimal_1$par)
sum(training2$prediction == 3)

## [1] 0

training3 <- log_like_class(Three.digit, optimal_1$par)
sum(training3$prediction == 2)

## [1] 0

# Then I can test the classifier using test data.  For digit two
prediction2 <- log_like_class(Test.two, optimal_1$par)
sum(prediction2$prediction == 3)/nrow(prediction2)

## [1] 0.0404

error_2 <- which(prediction2$prediction == 3)
par(mfrow = c(2, 4), mar = c(1, 1, 1, 1))
for (i in error_2[1:8]) {
    image(1:16, 1:16, matrix(Test.two[i, ], nrow = 16), col = cus_col(256), main = "Misleading Two")
}
# For digit three
prediction3 <- log_like_class(Test.three, optimal_1$par)
sum(prediction3$prediction == 2)/nrow(prediction3)

## [1] 0.04217

error_3 <- which(prediction3$prediction == 2)
par(mfrow = c(2, 4), pty = "s", mar = c(1, 1, 1, 1))
for (i in error_3) {
    image(1:16, 1:16, matrix(Test.three[i, ], nrow = 16), col = cus_col(256), main = "Misleading Three")
}
# The total error rate of prediction is
loglike_test_error <- (sum(prediction3$prediction == 2) + sum(prediction2$prediction == 3))/nrow(test)
loglike_test_error

## [1] 0.04121

# Method Two, fit linear regression function
fit <- lm(Y ~ X + 0)
summary(fit$fitted.values[1:731])  # The linear fitted value of digit two

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.3810 -0.0572  0.0341  0.0471  0.1360  0.5930

summary(fit$fitted.values[732:1389])  # The linear fitted value of digit three

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.299   0.845   0.964   0.948   1.050   1.320
```

```r
threshold <- 0.2993
# Then we can figure out training error and prediction error
linear_class <- function(pred, beta) {
    fit_value <- pred %*% beta
    vote <- fit_value
    vote[vote > threshold] <- 3
    vote[vote <= threshold] <- 2
    return(data.frame(prediction = vote))
}
train2_error_linear <- sum(linear_class(Two.digit, fit$coefficients) == 3)/nrow(Two.digit)
train2_error_linear
```

```
## [1] 0.05472
```

```r
train3_error_linear <- sum(linear_class(Three.digit, fit$coefficients) == 2)/nrow(Three.digit)
train3_error_linear
```

```
## [1] 0.00152
```

```r
test2_error_linear <- sum(linear_class(Test.two, fit$coefficients) == 3)/nrow(Two.digit)
test2_error_linear
```

```
## [1] 0.0342
```

```r
test3_error_linear <- sum(linear_class(Test.three, fit$coefficients) == 2)/nrow(Three.digit)
test3_error_linear
```

```
## [1] 0.006079
```

```r
test_error_linear <- (sum(linear_class(Test.three, fit$coefficients) == 2) + sum(linear_class(Test.two,
    fit$coefficients) == 3))/nrow(test)
test_error_linear
```

```
## [1] 0.07967
```

```r
# Method 3: knn classification
library(class)
twos <- as.vector(rep("Two", nrow(Two.digit)))
threes <- as.vector(rep("Three", nrow(Three.digit)))
# Training Error
knn_class_two_train <- knn(train = X, test = Two.digit, cl = c(twos, threes))
sum(knn_class_two_train == "Three")
```

```
## [1] 0
```

```r
knn_class_three_train <- knn(train = X, test = Three.digit, cl = c(twos, threes))
sum(knn_class_three_train == "Two")
```

```
## [1] 0
```

```r
knn_train_error <- function(train, test, k = 1, cl) {
    knncluster <- knn(train = train, test = test, k = k, cl = cl)
    error2 <- sum(knncluster[1:731] == "Three")
    error3 <- sum(knncluster[732:1389] == "Two")
    errorrate <- (error2 + error3)/nrow(test)
    return(errorrate)
}
```

```r
train_error_knn <- matrix(NA, nrow = 5)
train_error_knn[1] <- knn_train_error(X, X, k = 1, cl = c(twos, threes))
train_error_knn[2] <- knn_train_error(X, X, k = 3, cl = c(twos, threes))
train_error_knn[3] <- knn_train_error(X, X, k = 5, cl = c(twos, threes))
train_error_knn[4] <- knn_train_error(X, X, k = 7, cl = c(twos, threes))
train_error_knn[5] <- knn_train_error(X, X, k = 15, cl = c(twos, threes))
# Test error
knn_class_two_test <- knn(train = X, test = Test.two, cl = c(twos, threes))
sum(knn_class_two_test == "Three")/nrow(Test.two)  # Test error of predicting two
```

```
## [1] 0.0303
```

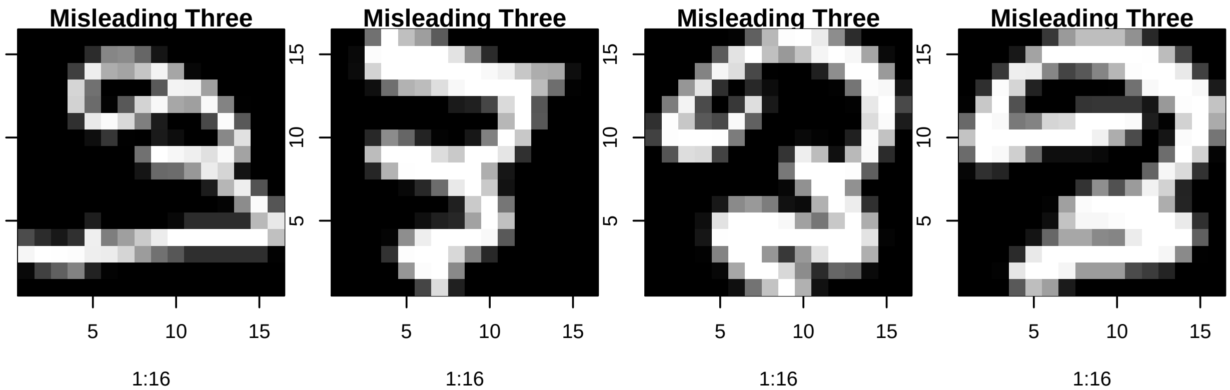
```r
knn_class_three_test <- knn(train = X, test = Test.three, cl = c(twos, threes))
sum(knn_class_three_test == "Two")/nrow(Test.three)  # Test error of predicting three
```

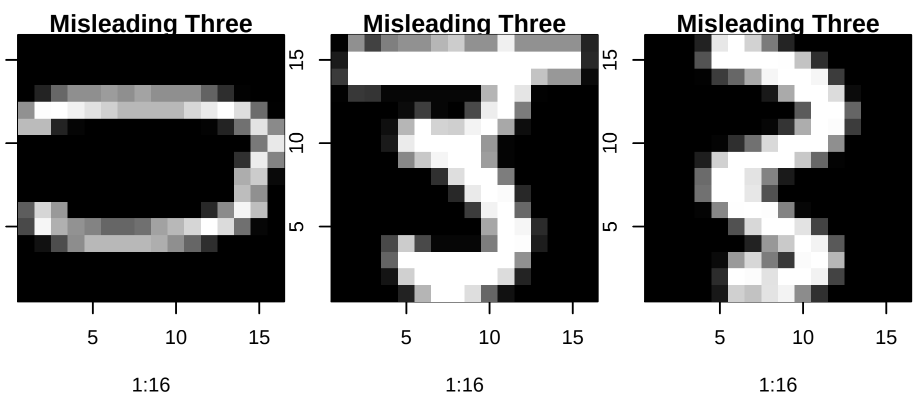
```
## [1] 0.01807
```

```r
# The total test error is
(sum(knn_class_two_test == "Three") + sum(knn_class_three_test == "Two"))/nrow(test)
```

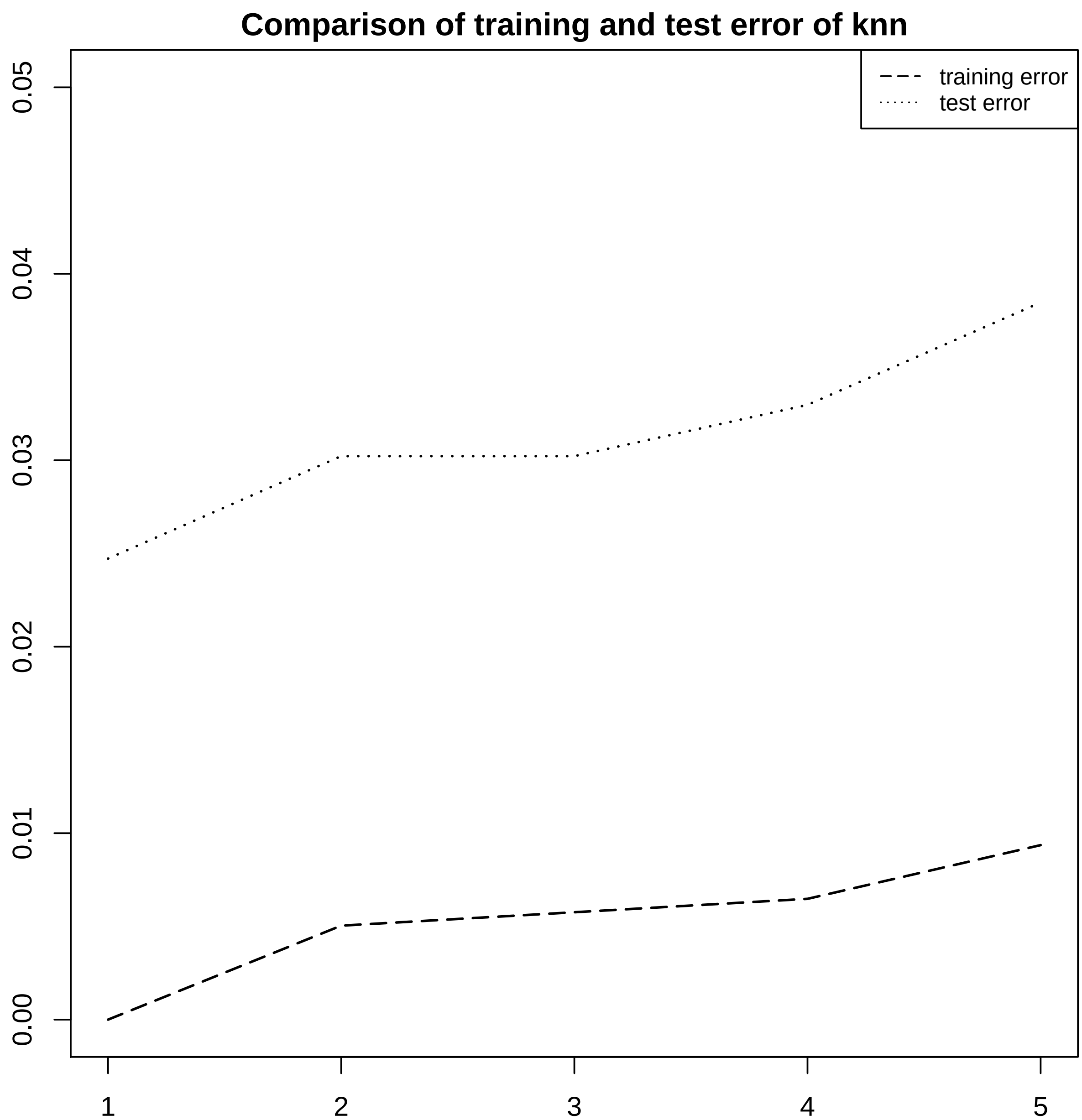
```
## [1] 0.02473
```

```r
# Now I need to make a function that output error rate
knn_class <- function(train, test, k = 1, cl) {
    knncluster <- knn(train = train, test = test, k = k, cl = cl)
    error2 <- sum(knncluster[1:198] == "Three")
    error3 <- sum(knncluster[199:364] == "Two")
    errorrate <- (error2 + error3)/nrow(test)
    return(errorrate)
}
par(mfcol = c(1, 1), mar = c(2.5, 1.5, 1.5, 0.5) + 0.1)
```

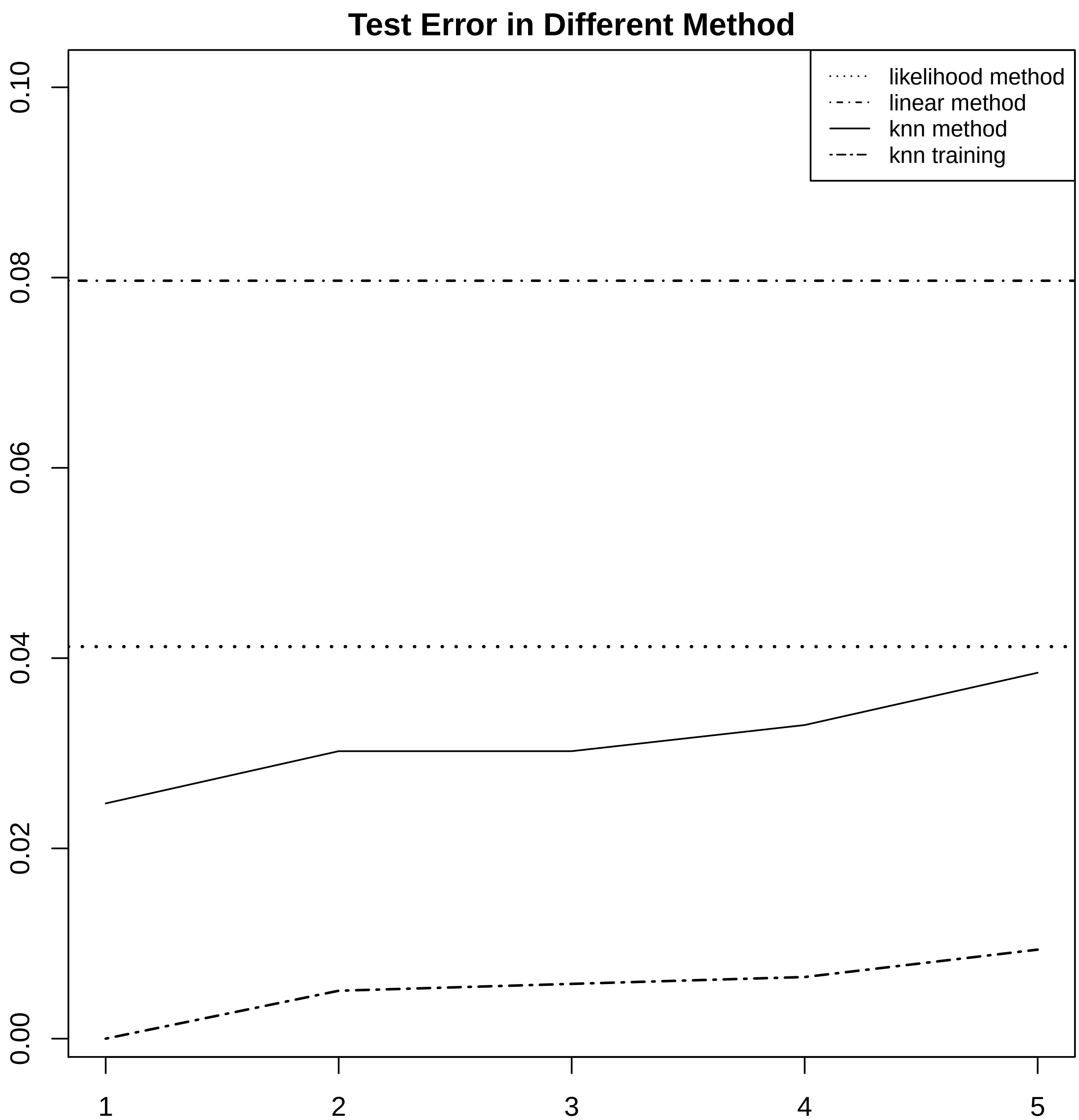





```
knn_summary <- matrix(NA, nrow = 5)
knn_summary[1] <- knn_class(X, test, k = 1, cl = c(twos, threes))
knn_summary[2] <- knn_class(X, test, k = 3, cl = c(twos, threes))
knn_summary[3] <- knn_class(X, test, k = 5, cl = c(twos, threes))
knn_summary[4] <- knn_class(X, test, k = 7, cl = c(twos, threes))
knn_summary[5] <- knn_class(X, test, k = 15, cl = c(twos, threes))
k <- c(1:5)
plot(k, knn_summary, type = "l", lty = 3, lwd = 1.5, ylim = c(0, 0.05), main = "Comparison of training a
lines(k, train_error_knn, , lty = 5, lwd = 1.5)
legend("topright", c("training error", "test error"), lty = c(5, 3), cex = 0.8)
```

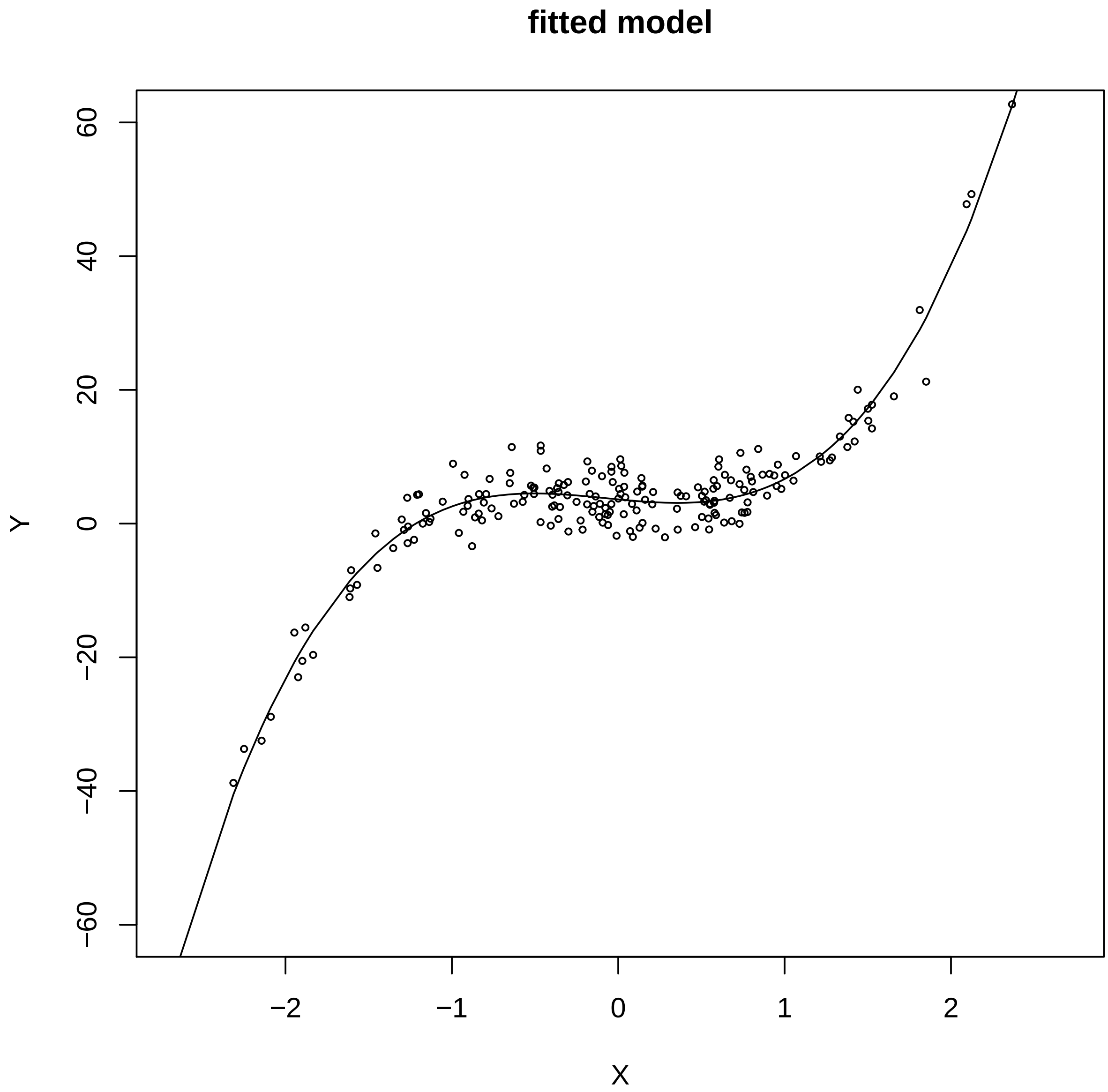



```
plot(k, knn_summary, ylim = c(0.002, 0.1), type = "l", main = "Test Error in Different Method",
    ylab = "Test Error")
abline(h = loglike_test_error, lty = 3, lwd = 2)
abline(h = test_error_linear, lty = 4, lwd = 1.5)
lines(k, train_error_knn, lty = 6, lwd = 1.5)
legend("topright", c("likelihood method", "linear method", "knn method", "knn training"), lty = c(3,
    4, 1, 6), cex = 0.8)
```

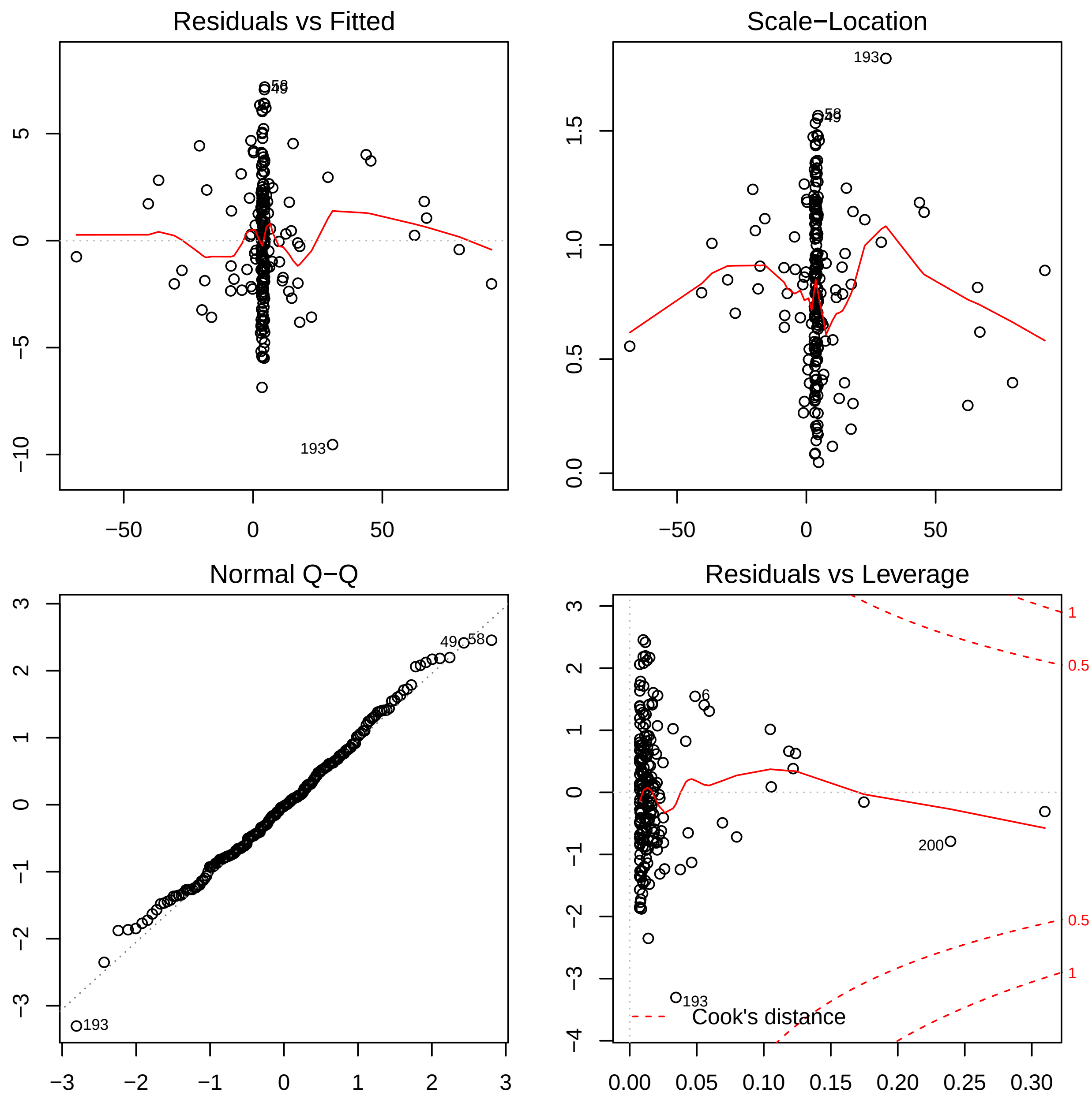


Appendix Two: code for problem three

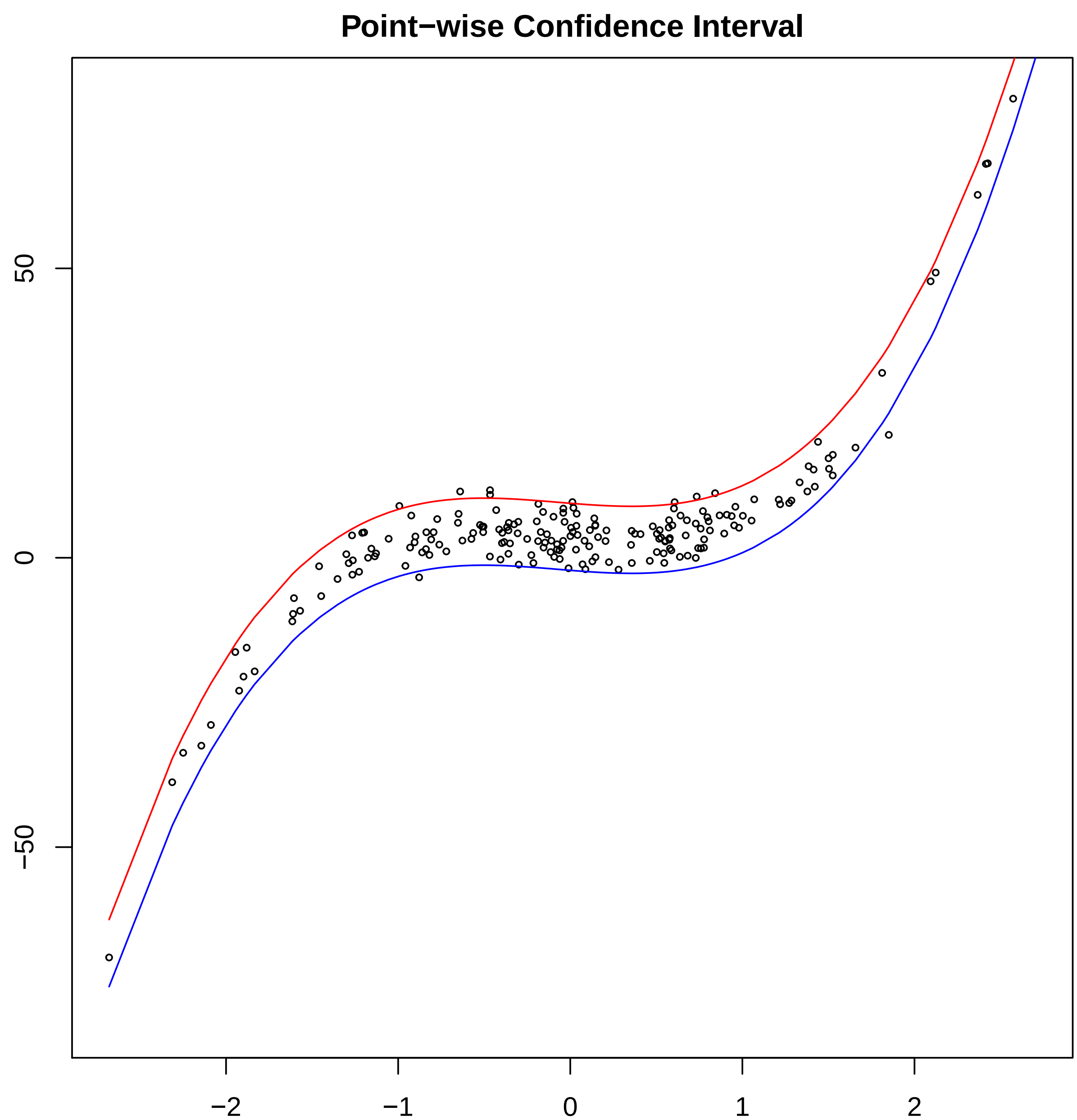
```
set.seed(250)
par(mfcol = c(1, 1))
X <- sort(matrix(rnorm(200), nrow = 200))
Beta <- sample(runif(20, min = -5, max = 5), 4)
f_x <- cbind(rep(1, 200), X, X^2, X^3)
Y <- f_x %*% Beta + rnorm(200, sd = 3)
fit <- lm(Y ~ X + I(X^2) + I(X^3))
plot(X, Y, ylim = c(-60, 60), cex = 0.5, main = "fitted model")
lines(X, f_x %*% fit$coefficients)
```



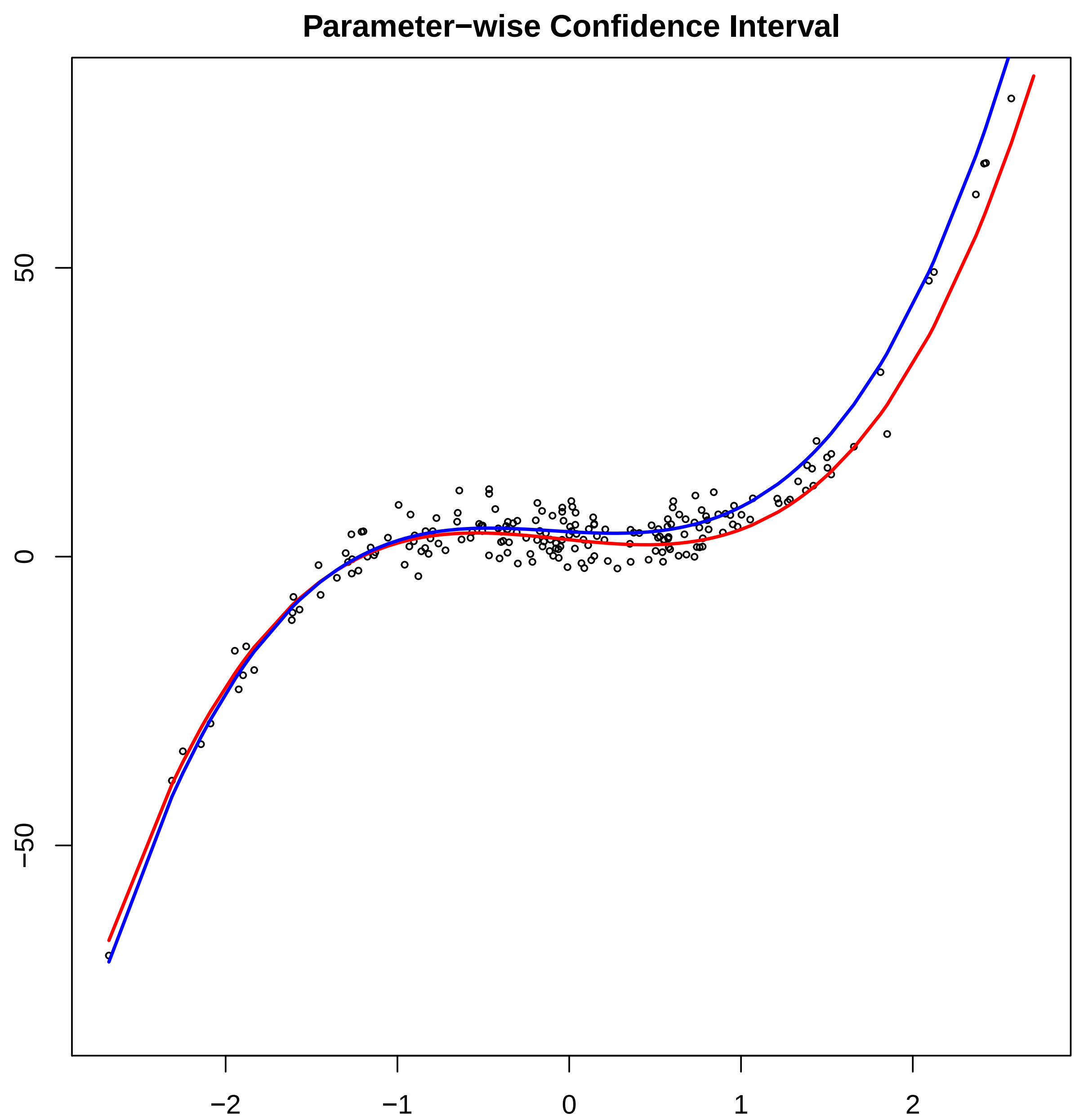

```r
par(mfcol = c(2, 2), mar = c(2, 2, 2, 2))
plot(fit)
```



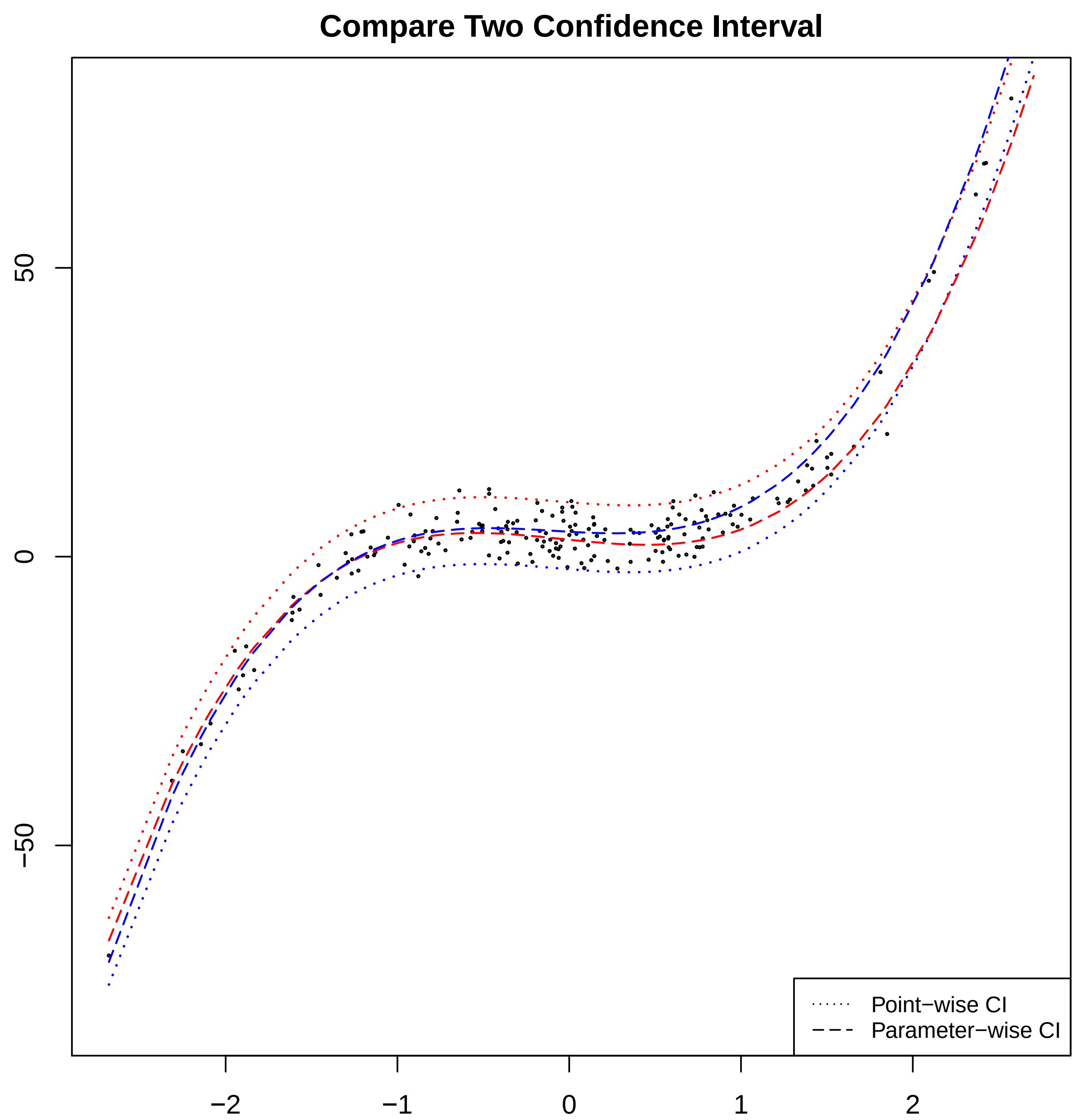

```
sigma_2 <- (summary(fit)$sigma)^2
# First I need to draw the confidence band that is constructed based on each point.
sd <- summary(fit)$sigma
par(mfcol = c(1, 1))
upper_bound <- fit$fitted.values + sd * qt(0.975, df = 200 - 4 - 1, lower.tail = TRUE)
lower_bound <- fit$fitted.values + sd * qt(0.025, df = 200 - 4 - 1, lower.tail = TRUE)
plot(X, Y, ylim = c(-80, 80), main = "Point-wise Confidence Interval", cex = 0.5)
lines(X, upper_bound, col = "red")
lines(X, lower_bound, col = "blue")
```




```
# Now I generate confidence interval based on the 3.15 equation
M <- t(f_x) %*% f_x
beta_upper <- fit$coefficients + sqrt(qchisq(0.95, df = 5) * sigma_2/diag(M))
beta_lower <- fit$coefficients - sqrt(qchisq(0.95, df = 5) * sigma_2/diag(M))
Y_hat_lower <- f_x %*% beta_upper
Y_hat_upper <- f_x %*% beta_lower
plot(X, Y, ylim = c(-80, 80), cex = 0.5, main = "Parameter-wise Confidence Interval")
lines(X, Y_hat_upper, col = "red", lwd = 2)
lines(X, Y_hat_lower, col = "blue", lwd = 2)
```

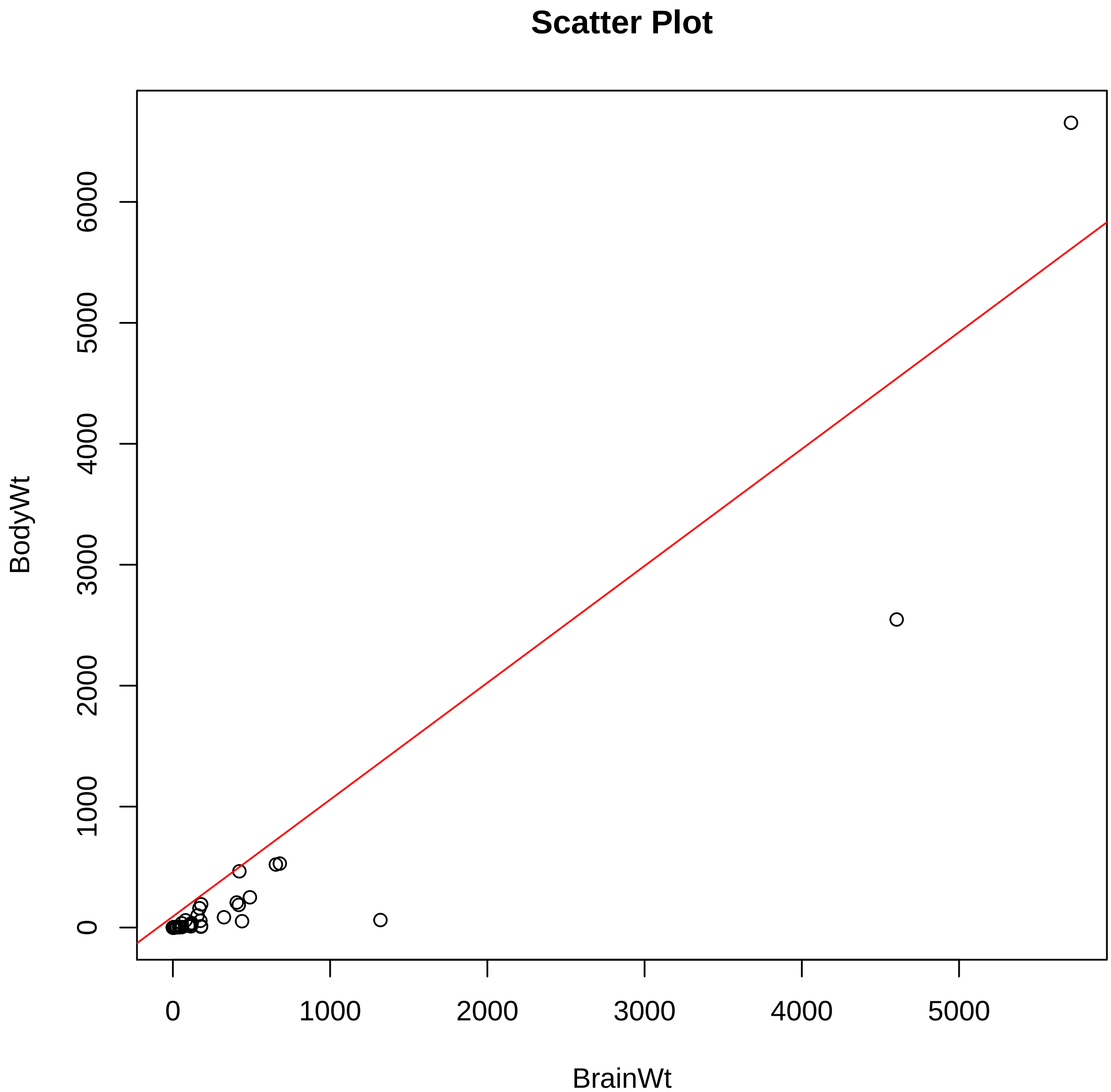



```r
# Now plot them together
plot(X, Y, ylim = c(-80, 80), cex = 0.2, main = "Compare Two Confidence Interval")
lines(X, upper_bound, col = "red", lty = 3, lwd = 1.5)
lines(X, lower_bound, col = "blue", lty = 3, lwd = 1.5)
lines(X, Y_hat_upper, col = "red", lty = 5, lwd = 1.2)
lines(X, Y_hat_lower, col = "blue", lty = 5, lwd = 1.2)
legend("bottomright", c("Point-wise CI", "Parameter-wise CI"), lty = c(3, 5), cex = 0.8)
```



Appendix Three: code for problem five

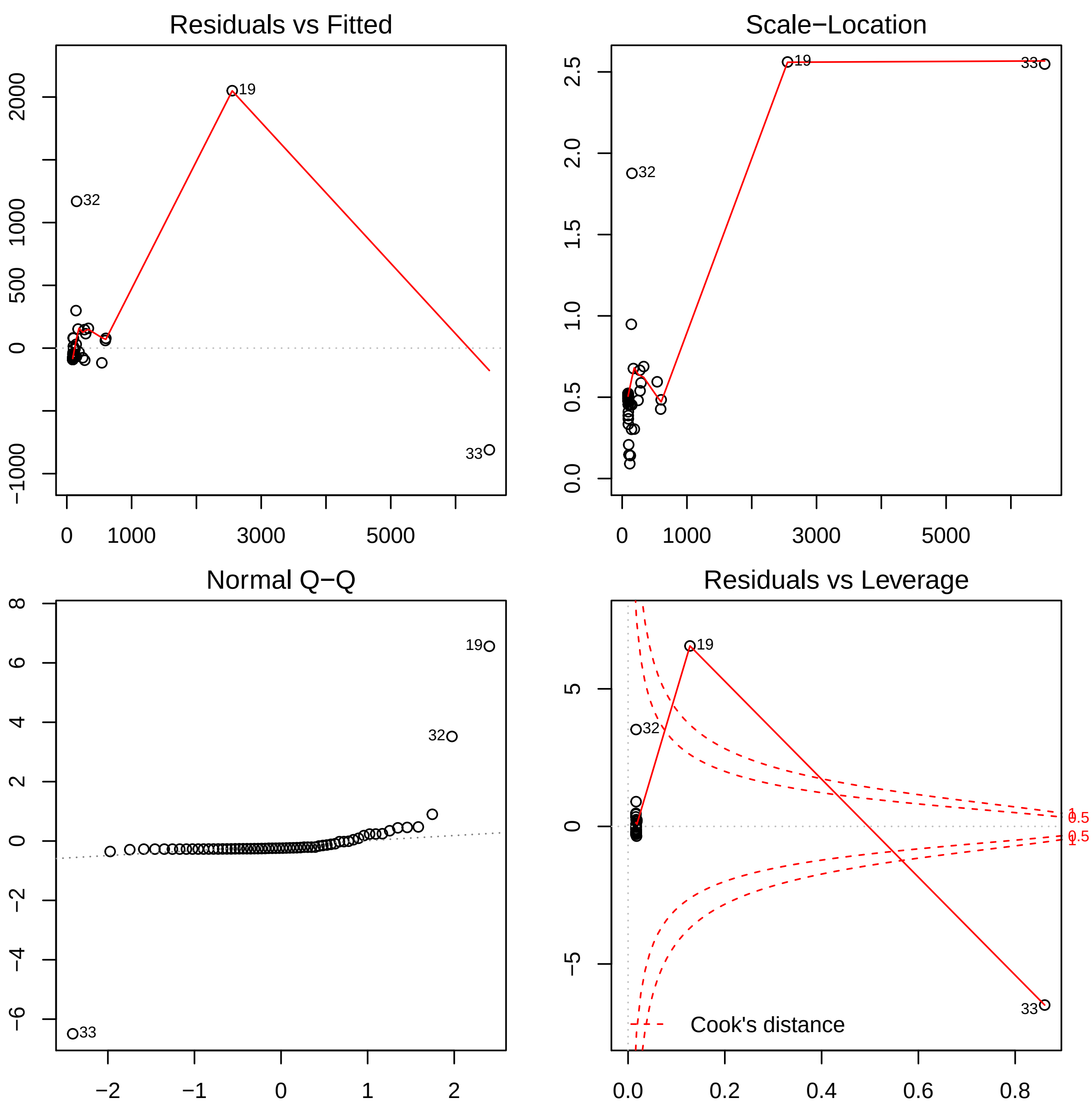
```
setwd("/Volumes/æIJĽèČ¡åĞžæšą/Stat 154/HW_3")
brain <- read.csv("brain.csv")
Brain <- brain$BrainWt
Body <- brain$BodyWt
# First fit
par(mfcol = c(1, 1))
fit_1 <- lm(Brain ~ Body)
plot(brain[, 2:3], main = "Scatter Plot", xlab = "BrainWt")
abline(fit_1, col = "Red")
```




```
summary(fit_1)

##
## Call:
## lm(formula = Brain ~ Body)
##
## Residuals:
##     Min      1Q Median      3Q     Max
## -810.1  -88.5  -79.6  -13.0 2050.5
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  91.0086    43.5557    2.09    0.041 *
## Body          0.9665     0.0477   20.28   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 335 on 60 degrees of freedom
## Multiple R-squared:  0.873,Adjusted R-squared:  0.871
## F-statistic:  411 on 1 and 60 DF,  p-value: <2e-16
```

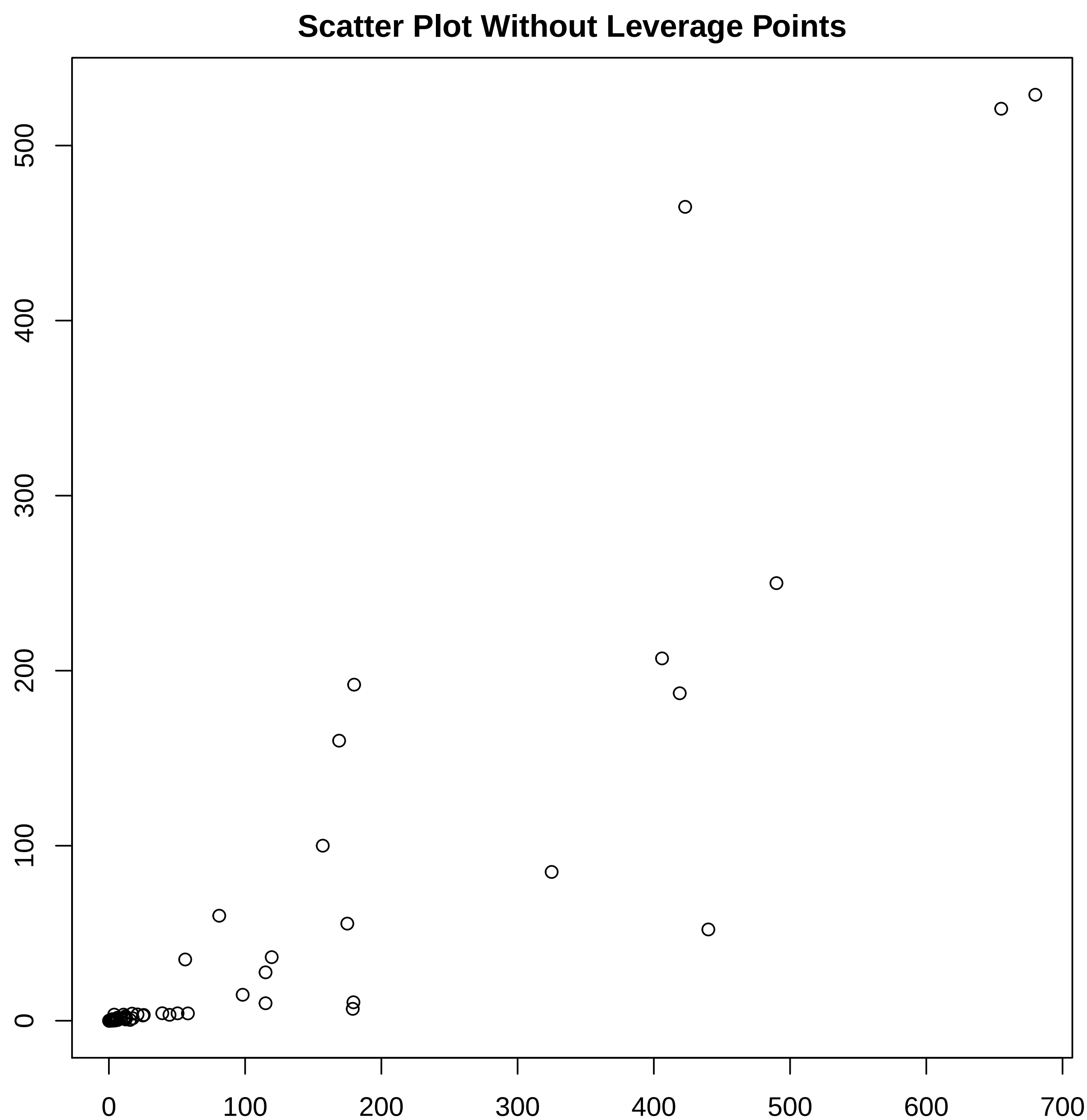
```
par(mfcol = c(2, 2), mar = c(2, 2, 2, 2))
plot(fit_1)
```




```
which(influence(fit_1)$hat == max(influence(fit_1)$hat))
```

```
## 33
## 33
```

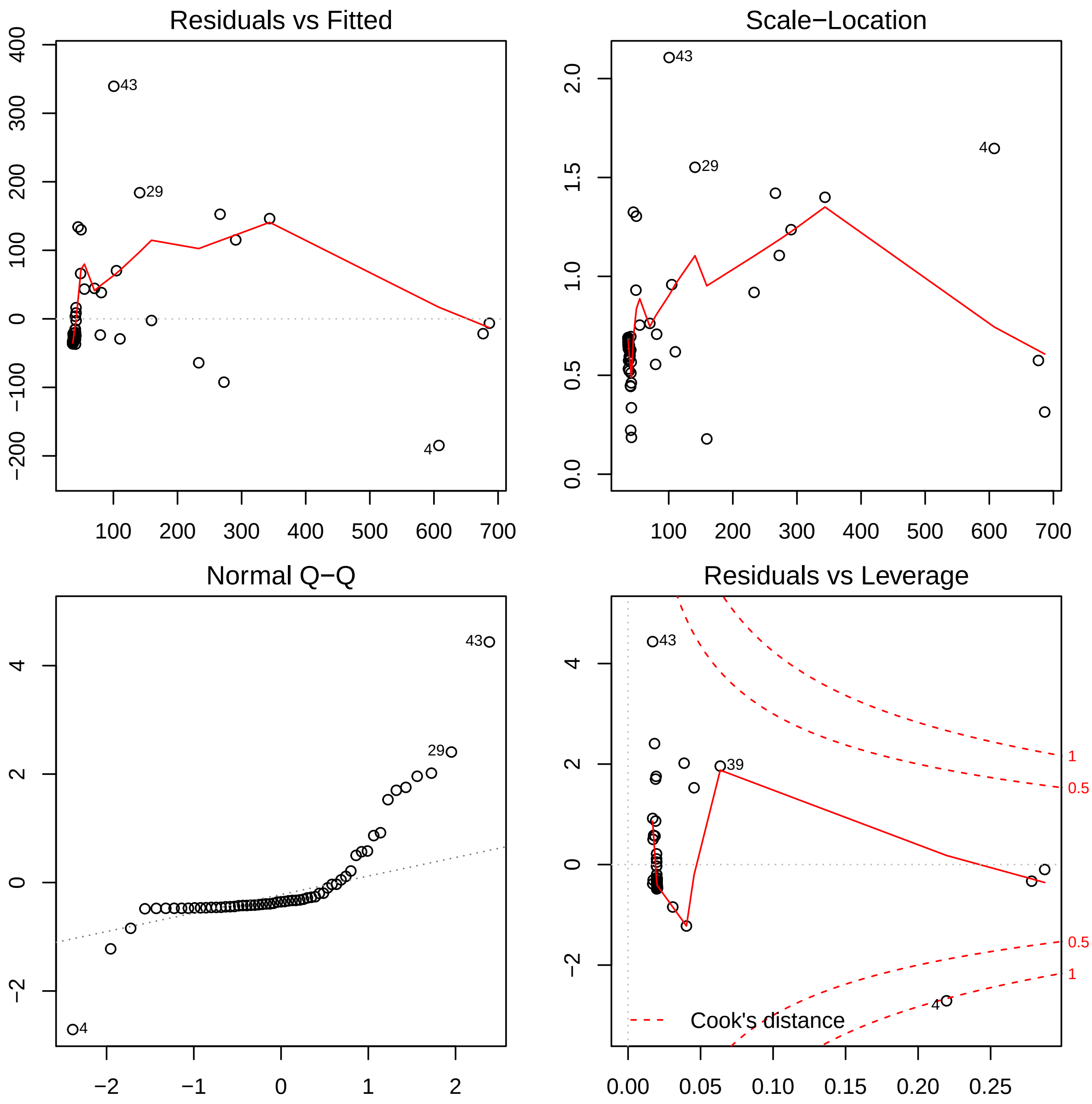
```r
brain[33, ]

##           Case_names BrainWt BodyWt
## 33 African_elephant    5712   6654

brain[19, ]

##         Case_names BrainWt BodyWt
## 19 Asian_elephant    4603   2547

brain[32, ]

##    Case_names BrainWt BodyWt
## 32      Human    1320     62

# Plot the points without the leverage points.
no_leverage <- rbind(brain[1:18, ], brain[20:31, ], brain[34:62, ])
par(mfcol = c(1, 1))
plot(no_leverage[, 2:3], main = "Scatter Plot Without Leverage Points")
```




```
fit_2 <- lm(no_leverage[, 2] ~ no_leverage[, 3])
summary(fit_2)

##
## Call:
## lm(formula = no_leverage[, 2] ~ no_leverage[, 3])
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -184.8  -34.5  -27.2    0.7  339.4
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       36.5728    10.9509    3.34   0.0015 **
```

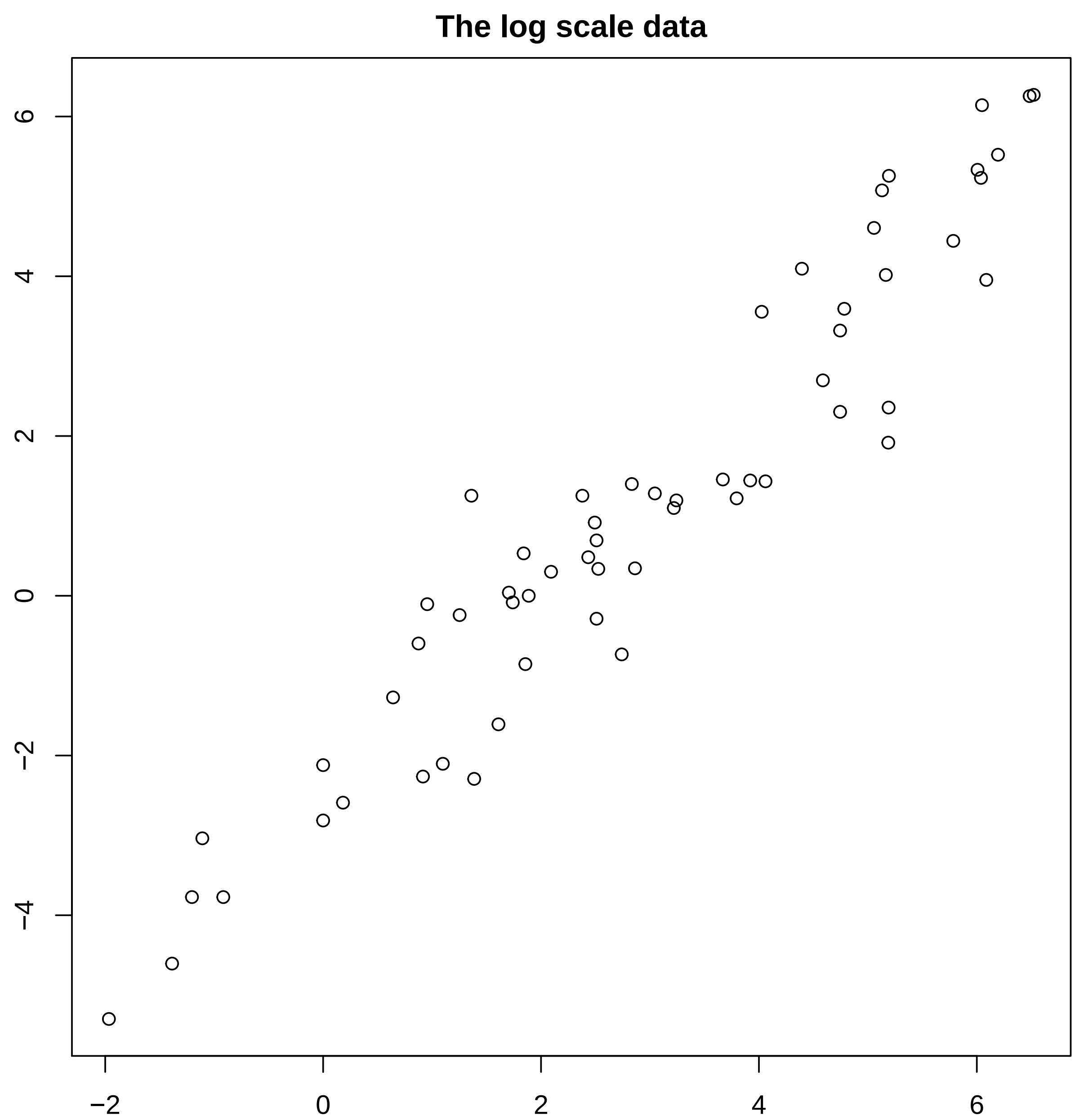
```
## no_leverage[, 3]    1.2285      0.0841    14.61    <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 77.1 on 57 degrees of freedom
## Multiple R-squared:  0.789,Adjusted R-squared:  0.786
## F-statistic:  213 on 1 and 57 DF,  p-value: <2e-16
```

```
par(mfcol = c(2, 2), mar = c(2, 2, 2, 2))
plot(fit_2)
```




```
# Then I decided to take the log-scale of data and do the simple regression
fit_3 <- lm(log(no_leverage[, 2]) ~ log(no_leverage[, 3]))
par(mfcol = c(1, 1))
```

```r
plot(log(no_leverage[, 2]), log(no_leverage[, 3]), main = "The log scale data", ylab = "BrainWt",
    xlab = "BodyWt")
```




```r
summary(fit_3)
```

```
##
## Call:
## lm(formula = log(no_leverage[, 2]) ~ log(no_leverage[, 3]))
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -1.674 -0.490 -0.035  0.475  1.664
##
## Coefficients:
```

```
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)              2.1139     0.0914    23.1   <2e-16 ***
## log(no_leverage[, 3])    0.7353     0.0299    24.6   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.659 on 57 degrees of freedom
## Multiple R-squared:  0.914,Adjusted R-squared:  0.912
## F-statistic:  603 on 1 and 57 DF,  p-value: <2e-16

par(mfcol = c(2, 2), mar = c(2, 2, 2, 2))
plot(fit_3)
```