Assignment 2

Intializing the libraries and setting the seed:

```
set.seed(1006274274);
library(ggplot2)
library(MASS)
library(rcompanion)
library(lmtest)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

Question 1:

```
myTable <- read.table("vote.txt", header = TRUE, sep = "", dec = ".")

growthArr <-myTable$growth;
voteArr <- myTable$vote;


error =rnorm(length(growthArr),0,3.9);
fakeYData = vector()

for (i in 1:length(growthArr)){
  fakeYData = append(fakeYData, 46.3 + 4* growthArr[i] + error[i])
}

fakeDataModel = lm (fakeYData ~ growthArr)

summary(fakeDataModel)
```

```
##
## Call:
## lm(formula = fakeYData ~ growthArr)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.7258 -2.4867  0.3444  2.7275  5.7538
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  44.7707     1.6817  26.622 2.16e-13 ***
## growthArr     5.2615     0.7219   7.288 3.98e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.902 on 14 degrees of freedom
## Multiple R-squared:  0.7914, Adjusted R-squared:  0.7765
## F-statistic: 53.11 on 1 and 14 DF,  p-value: 3.983e-06
```

```
df = data.frame(growthArr = 0.1)

predict(fakeDataModel,newdata = df)
```

```
##        1
## 45.29687
```

```
#By default the prediction is 95%
predict(fakeDataModel, newdata = df, interval = "confidence")
```

```
##        fit      lwr      upr
## 1 45.29687 41.81493 48.77882
```

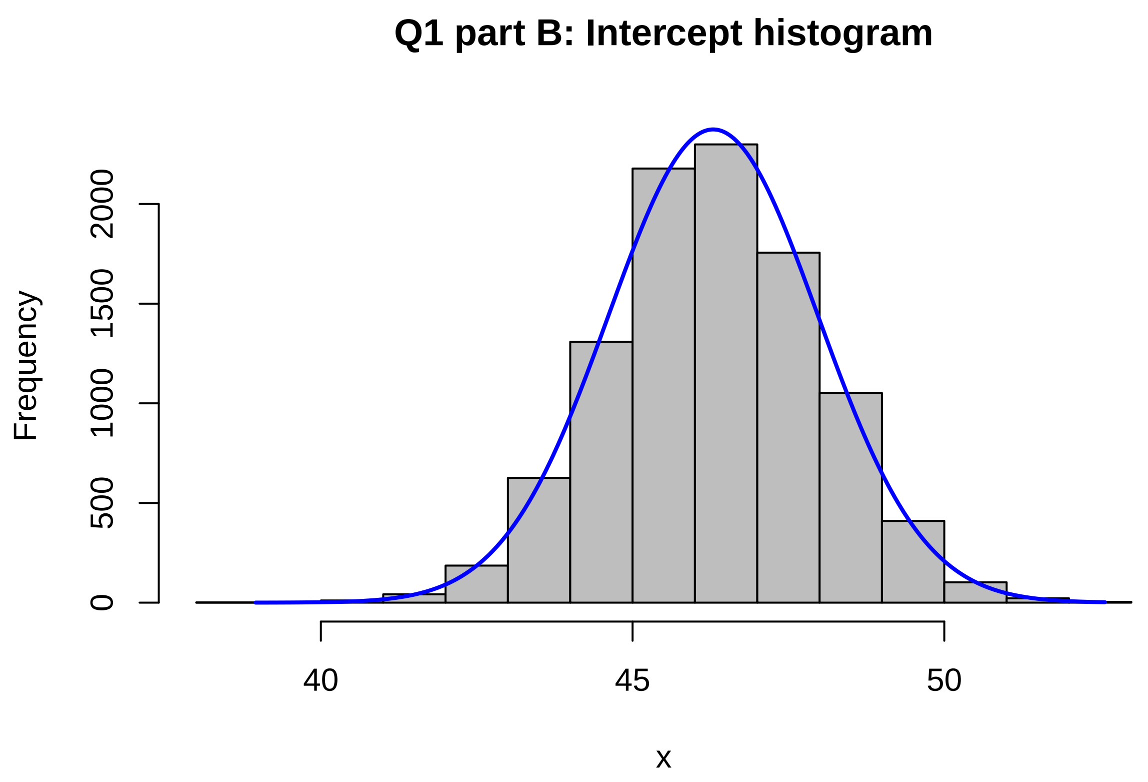
As shown in the summary, the intercept is estimated to be at 44.77, and the slope is estimated to be at 5.26.

The predicted value of $E(Y|X_0 = 0.1)$ is 45.29.

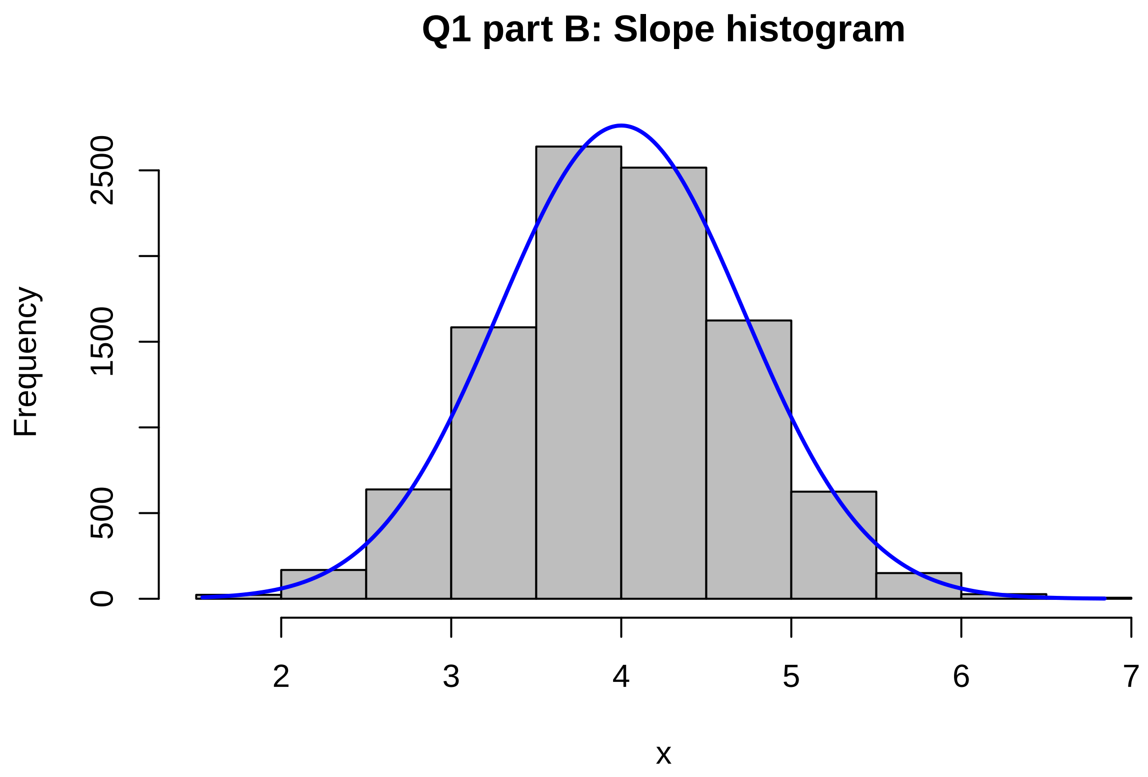
The predicted interval of $[41.81, 48.77]$ is the same as if we calculated it by hand, using the formula: predicted-value $\pm$ t-value $\times$ error.

Question 1B)

```
generateFakeDataModel <- function (xVector){

  error =rnorm(length(xVector),0,3.9);
  fakeYData = vector()

  for (i in 1:length(xVector)){
    fakeYData = append(fakeYData, 46.3 + 4* xVector[i] + error[i])
  }

  return ( lm (fakeYData ~ xVector))
}

interceptVector = vector()
slopeVector = vector()

for (i in 1:10000){
  linModel <- generateFakeDataModel(growthArr);
  interceptVector = append(interceptVector, coef(linModel)["(Intercept)"])
  slopeVector = append (slopeVector, coef(linModel)["xVector"])
}

plotNormalHistogram( unlist(interceptVector), prob = FALSE,
                     main = "Q1 part B: Intercept histogram",
                     length = 1000)
```



```
plotNormalHistogram( unlist(slopeVector), prob = FALSE,
                     main = "Q1 part B: Slope histogram",
                     length = 1000)
```



```
data_frame = data.frame(growthArr = growthArr, fakeYData = fakeYData)
```

```r
cat ("The mean of the intercept vector is expected to be 46.3, and it is:", mean(interceptVector), "\n"
```

```
## The mean of the intercept vector is expected to be 46.3, and it is: 46.29328
```

```r
cat ("The mean of the slope vector is expected to be 4, and it is:", mean(slopeVector), "\n")
```

```
## The mean of the slope vector is expected to be 4, and it is: 4.000087
```

```r
# Yes, the simulated results are consistent with theoretical results.
sigmaHat = 3.9

SXX <- sum((growthArr - mean(growthArr))^2)

#Standard error of slope (beta 1 hat), should be the same as the sd of the slope vector
stanErrorBeta1 <- sigmaHat / sqrt(SXX)

cat("The standard deviation of the slope is expected to be", sd(slopeVector),
    " and it is estimated to be", stanErrorBeta1, "\n")
```

```
## The standard deviation of the slope is expected to be 0.7223497  and it is estimated to be 0.721568
```

```r
#Standard error of intercept (beta 0 hat), should be the same as the sd of the slope intercept

stanErrorBeta0 <- sqrt(1 / length(growthArr) + (mean(growthArr)^2)/SXX) * sigmaHat

cat("The standard deviation of the intercept is expected to be", sd(interceptVector),
    " and it is estimated to be", stanErrorBeta0, "\n")
```

```
## The standard deviation of the intercept is expected to be 1.680417  and it is estimated to be 1.68085
```

Yes, the results are absolutely consistent with the theoretical values, as shown above.

Part C)

```r
df = data.frame(xVector = 0.1)

total = 10000
expectedVal = 46.3 + (4 * 0.1)

valCountInConfInt = 0
for (i in 1:total){


  currPrediction = predict(generateFakeDataModel(growthArr), newdata = df, interval = "confidence")
  if (currPrediction[[2]] <= expectedVal & expectedVal <= currPrediction[[3]]){
    valCountInConfInt = valCountInConfInt + 1
  }
}

cat ("Percentage of the expected val in the generated CIs:", valCountInConfInt / total, "\n")
```

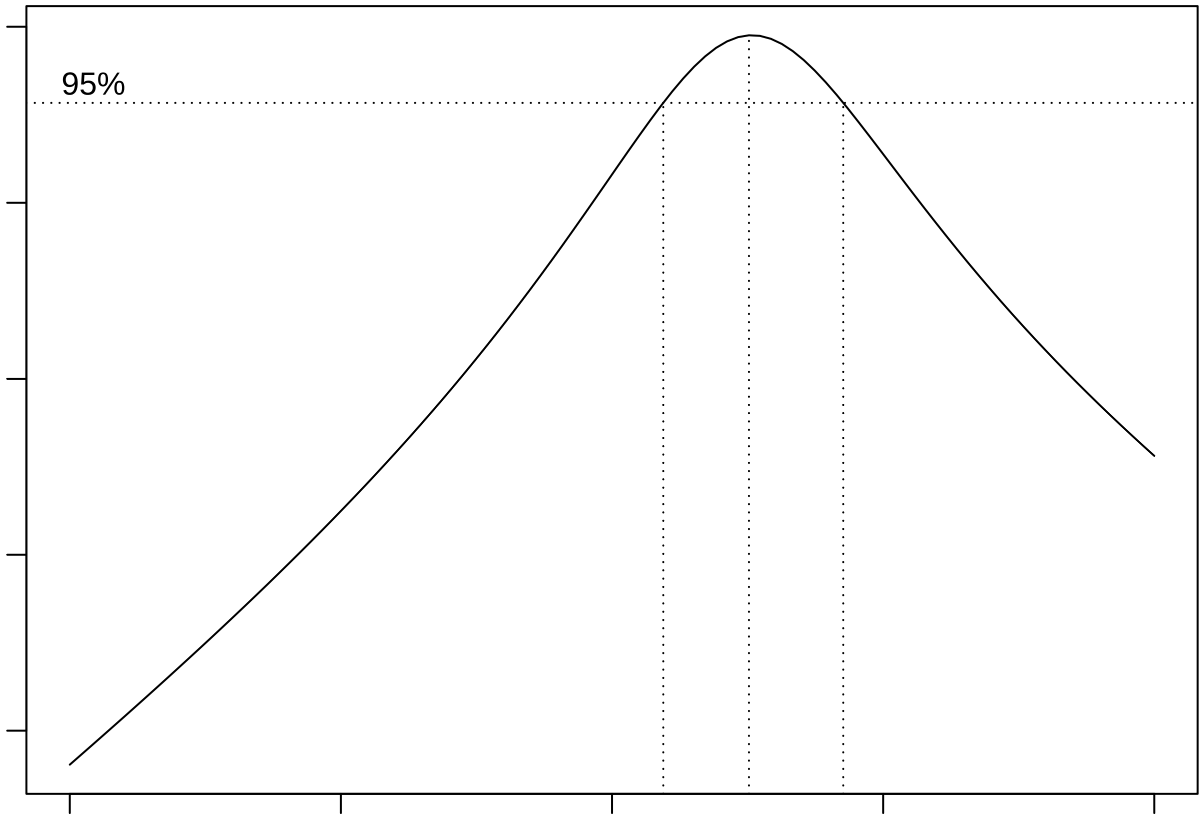
```
## Percentage of the expected val in the generated CIs: 0.9464
```

```r
# This is absolutely consistent with the predicted values.
```

Again, this is absolutely consistent with the predicted values

Question 2)

```r
x <- c(0:9)
y <- c(98, 135, 162, 178, 221, 232, 283, 300, 374, 395)
par(mar=c(1,1,1,1))
fit <- lm (y~x)
result <- boxcox(fit, lambda = seq(-2,2,0.1))

mylambda <- result$x[which.max(result$y)]

result = boxcox(y~x, lambda = seq(-2,2,0.1))
```



```r
cat("The optimal lambda given by R is: ", result$x[which.max(result$y)], "\n")
```

```
## The optimal lambda given by R is:  0.5050505
```

```r
valOf_n = length(x)

k2 = (prod(y))^(1/valOf_n)

optimal_lambda = -42
minSSE = .Machine$integer.max
for (lambda in seq(-2,2,0.1)){
  W = vector()

  for (i in 1:length(y)){

    if(lambda != 0){
      k1 = 1/(lambda * k2^(lambda-1) )
      w_i = k1 * ((y[i]^lambda) - 1)
    }else{
      w_i = k2 * (log(y[i]))
    }
    W = append(W,w_i)
  }

  tempModel = lm(W~x)
  cf <- coef(tempModel)
  intercept <- cf["(Intercept)"]
  slope <- cf["x"]
  # cat("The intercept is: ", intercept, " the slope is: ", slope, "\n")
  SSE = 0
  for(i in 1:length(x)){
    w_i_hat = intercept + slope * x[i]
    SSE = SSE + (W[i] - w_i_hat)^2
  }

  if(SSE < minSSE){
    minSSE = SSE
    optimal_lambda = lambda
  }

  W = vector()


}

cat("The optimal lambda that we've calculated is:", optimal_lambda, "\n")
```

```
## The optimal lambda that we've calculated is: 0.5
```

This is pretty much the same result. I can get more precise values if I changed the sequence to iterate by a smaller value. Currently, it's iterating by 0.1, and so 0.5 is the most accurate value it will generate.

Question 3)

Part a)

```
kidiqTable = read.table("kidiq.csv",header = TRUE,sep = ",", dec = ".")

xAxis = kidiqTable$mom.iq
yAxis = kidiqTable$kid.score
modelForIQandScore = lm (yAxis ~ xAxis)


#Small p-value means reject null, so variance is constant

 predict(modelForIQandScore, newdata = data.frame(xAxis = 110), interval = "confidence")
```

```
##        fit      lwr     upr
## 1 92.89698 90.82506 94.9689
```

```
# Mom.work is probably how much money mom makes. 4= the most, 1=the least
n = length(kidiqTable[[1]])

t_val = qt(p=0.025, df=432, lower.tail = FALSE)
```

```
# summary(modelForIQandScore)
sumOfSquareDev =   sum( (xAxis - mean(xAxis) )^2 )
MSE = mean(modelForIQandScore$residuals^2)
stanError = sqrt(MSE * (1/n + ((110 - mean(xAxis))^2 / sumOfSquareDev)))

upperIntVal = 92.79 + t_val * stanError
cat ("Upper interval is: ", upperIntVal, "\n")
```

```
## Upper interval is:  94.85714
```

```
lowerIntVal = 92.79 - t_val * stanError
cat ("Lower interval is: ", lowerIntVal, "\n")
```

```
## Lower interval is:  90.72286
```

```
# Both values match the R values generated by predict.
```

Again, the manually computed values (using R) are pretty much identical to the ones generated by the predict() function. Only slight rounding errors cause the small differences.

Part B)

```
t_val_partB = qt(p=0.995, df=432, lower.tail = TRUE)

stanError_partB = sqrt(MSE * (1 + 1/n + ((110-mean(xAxis))^2 / sumOfSquareDev)))

upperIntVal_partB = 92.79 + t_val_partB * stanError_partB
cat ("Upper interval is: ", upperIntVal_partB, "\n")
```

```
## Upper interval is:  140.0186
```

```
lowerIntVal_partB = 92.79 - t_val_partB * stanError_partB
cat ("Lower interval is: ", lowerIntVal_partB, "\n")
```

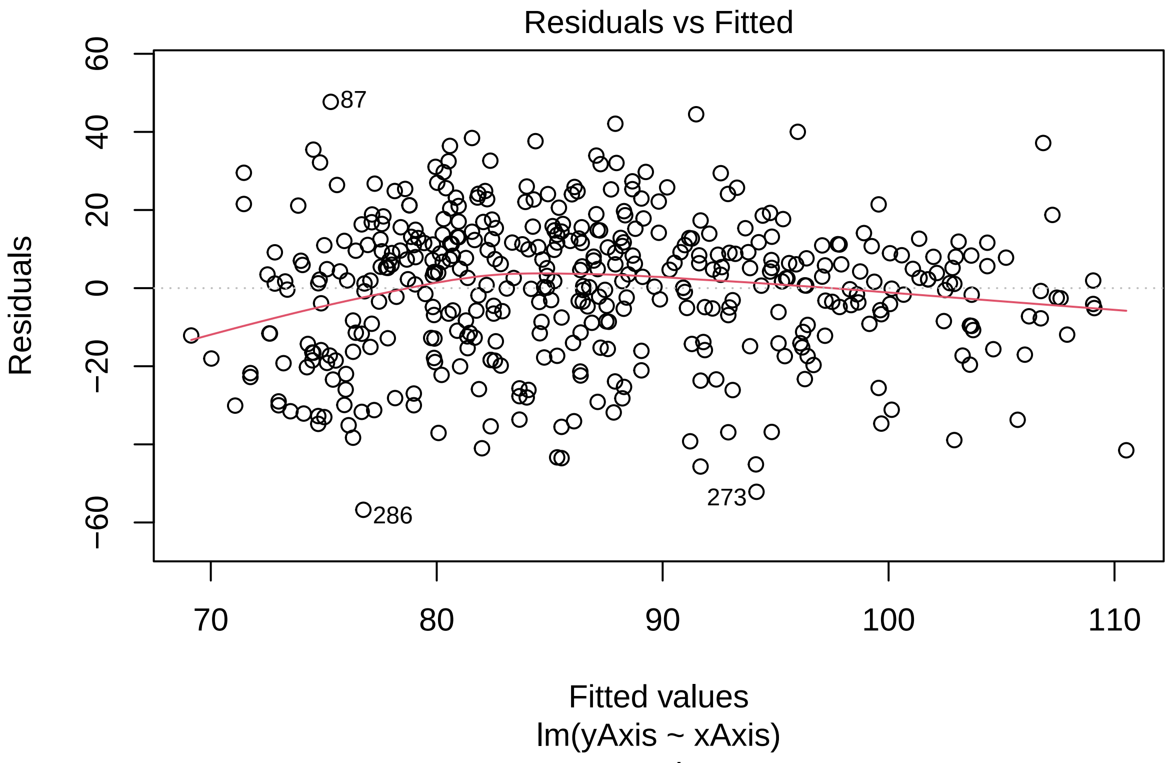
```
## Lower interval is:  45.5614
```

```
predict(modelForIQandScore, newdata = data.frame(xAxis = 110), interval = "confidence", level = 0.99)
```

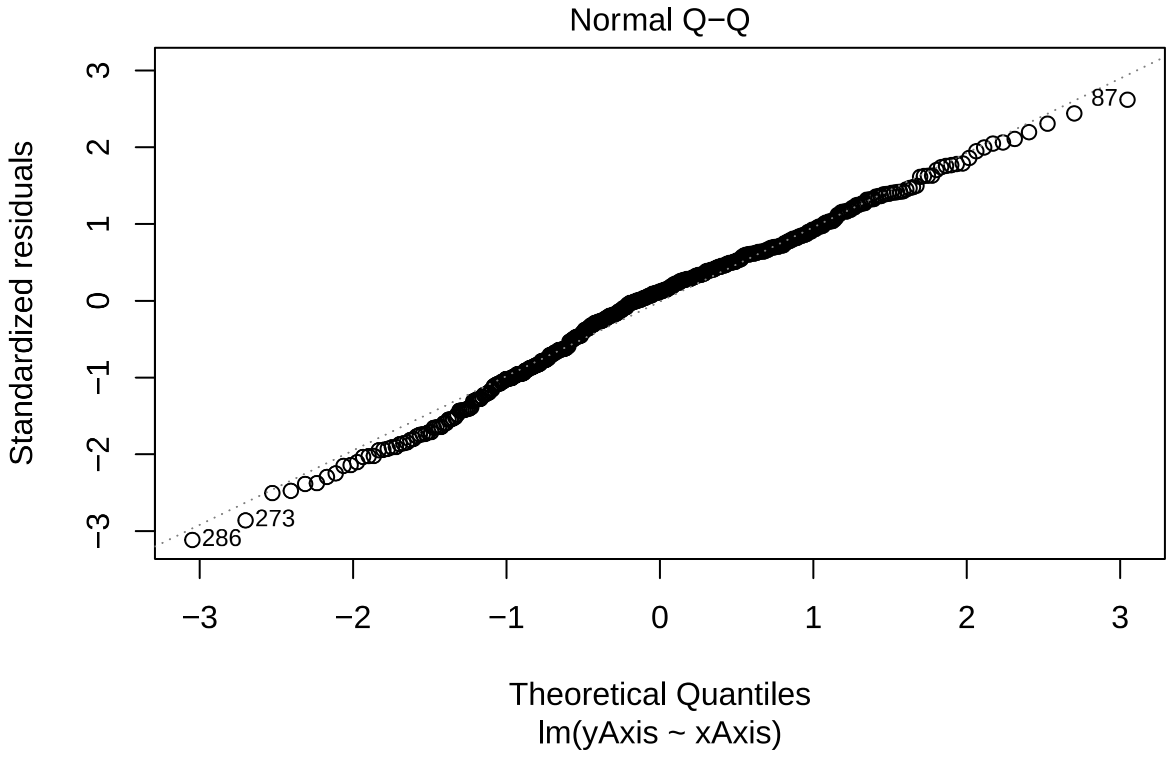
```
##        fit     lwr      upr
## 1 92.89698 90.1696 95.62436
```

Part c)

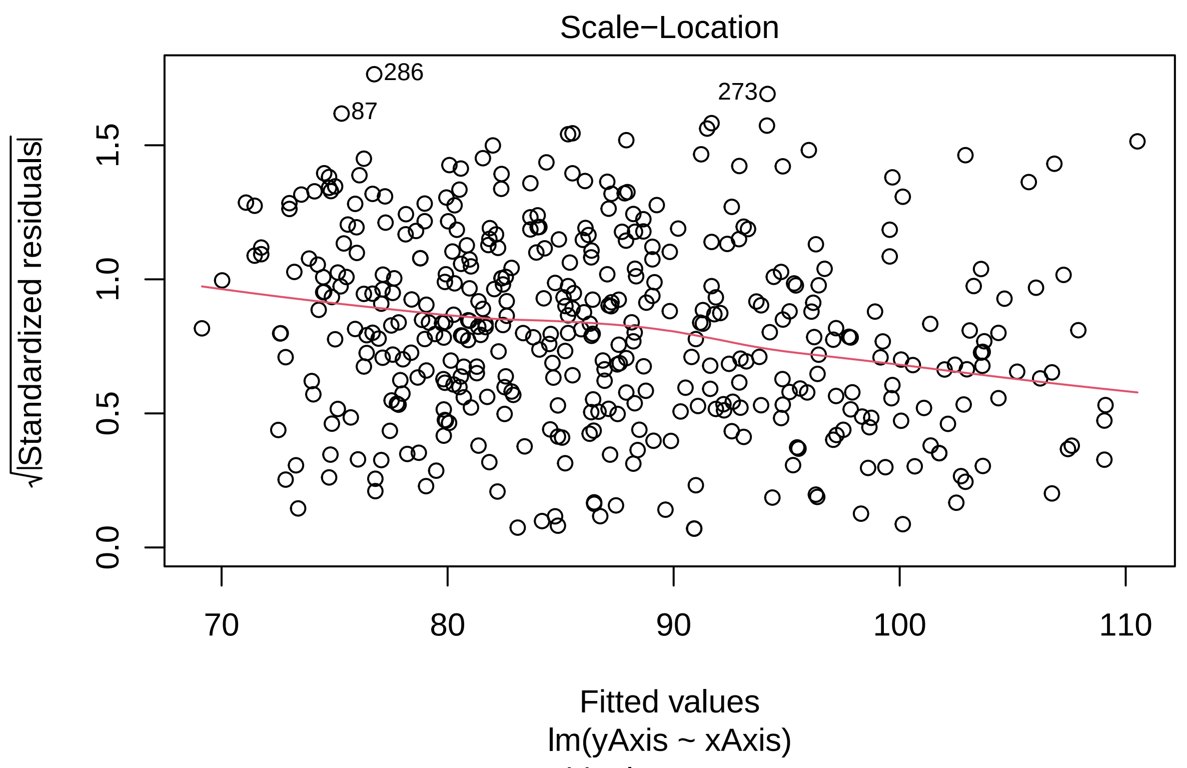
```
plot(modelForIQandScore)
```

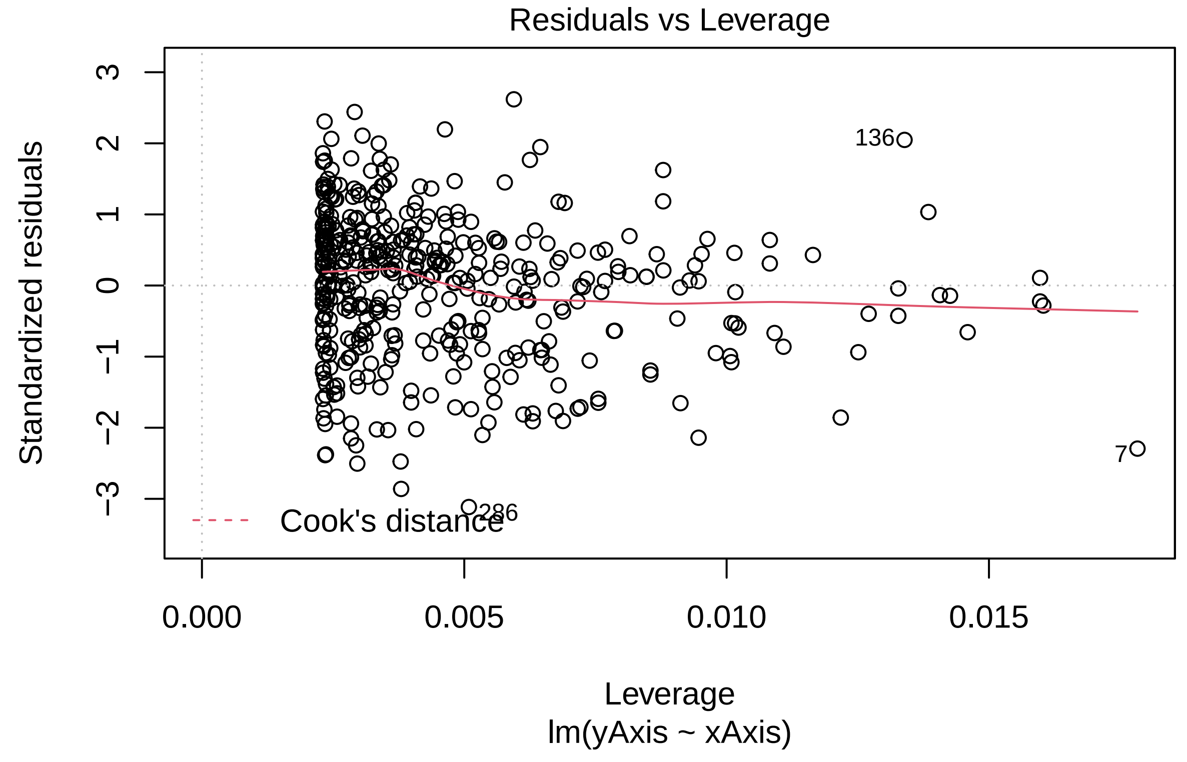


Residuals vs Fitted
Residuals
60
40
20
0
−20
−60
70
80
90
100
110
87
286
273
Fitted values
lm(yAxis ~ xAxis)

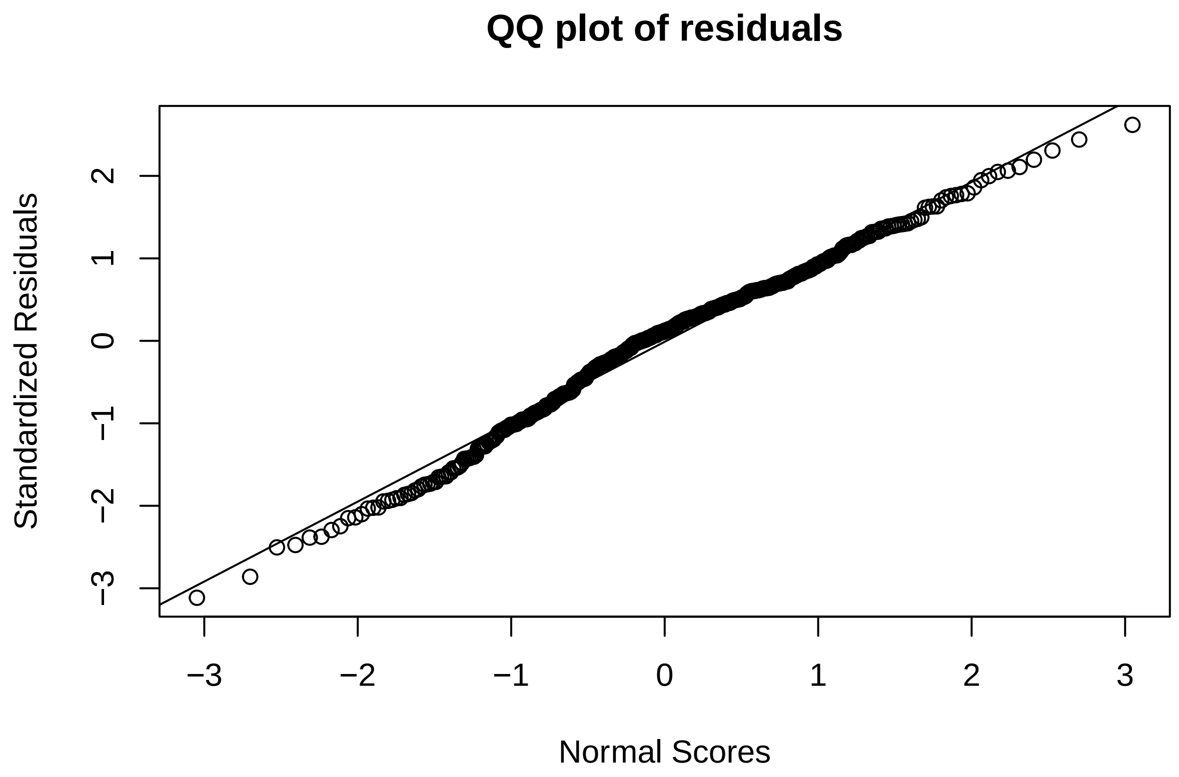


Normal Q−Q
Standardized residuals
3
2
1
0
−1
−2
−3
87
273
286
Theoretical Quantiles
lm(yAxis ~ xAxis)





```
stanResiduals = rstandard(modelForIQandScore)



qqnorm(stanResiduals,
       ylab="Standardized Residuals", xlab="Normal Scores", main="QQ plot of residuals")
qqline(stanResiduals)
```



This looks pretty accurate to what we would expect.

Part d)

```
shapiro.test(modelForIQandScore$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  modelForIQandScore$residuals
## W = 0.98885, p-value = 0.00217
```

The p-value is 0.00217, which is < 0.05 (our alpha level), therefore we reject the null hypothesis that the data is normally distributed.

Part e)

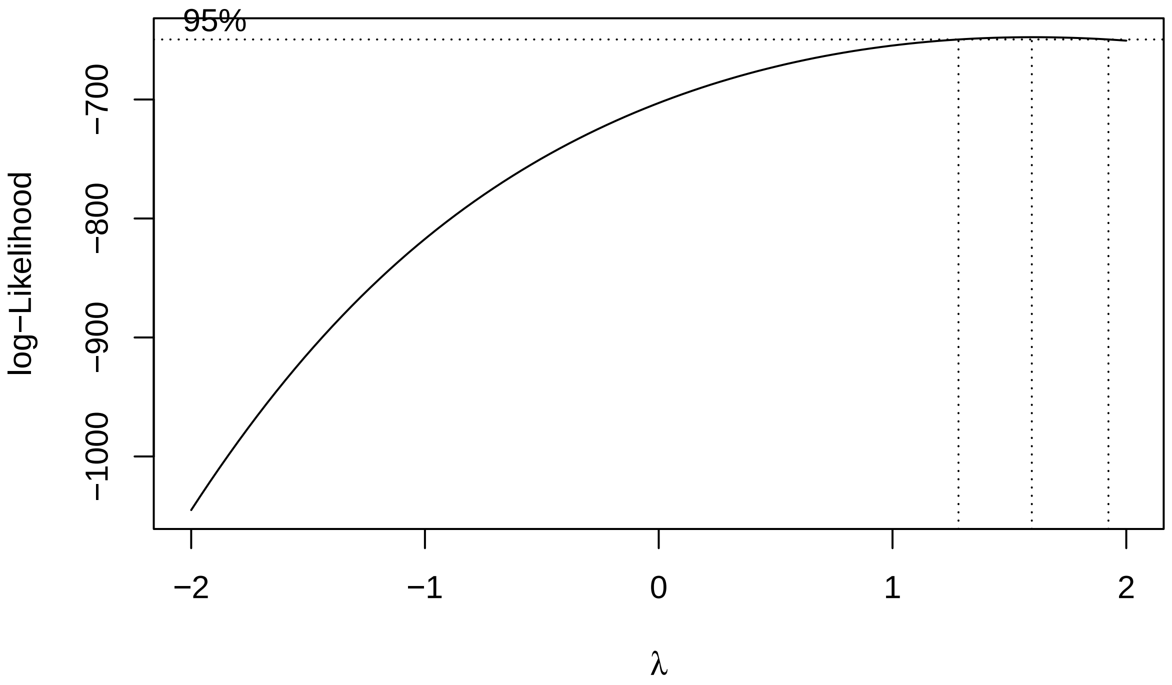
```
bptest(modelForIQandScore)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  modelForIQandScore
## BP = 6.0401, df = 1, p-value = 0.01398
```

For the breusch-pagan test, we test the errors in the regression. Null hypothesis is that all the error variances are the same Alternate hypothesis is that they're not the same. Since the p value is $0.013 < 0.05$, we reject the null hypothesis.

Part f) Since the errors are not normally distributed, nor have constant variance, we should apply a boxcox transformation to make it more normal

```
bc <- boxcox(yAxis ~ xAxis)
```



```
lambda = bc$x[which.max(bc$y)]

#fit new linear regression model using the Box-Cox transformation
new_model <- lm((yAxis^lambda) ~ xAxis)

print("The shapiro-wilk test on the boxcox transformed model")
```

```
## [1] "The shapiro-wilk test on the boxcox transformed model"
```

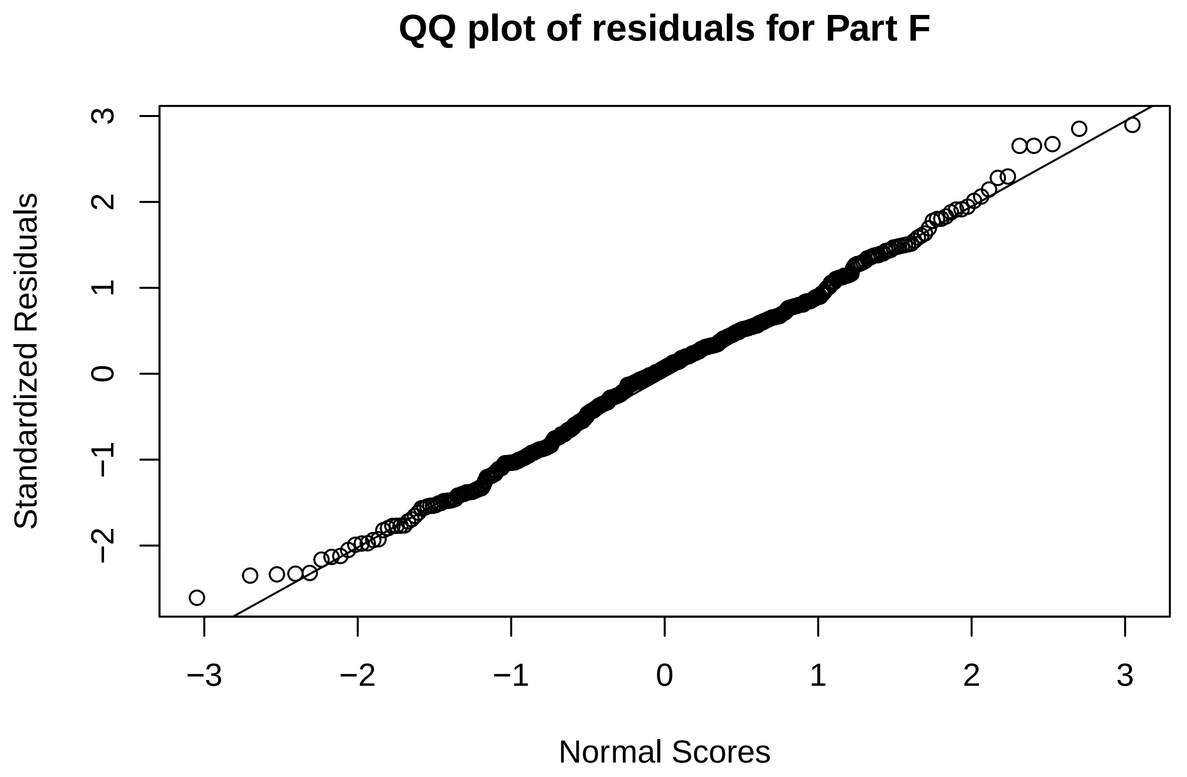
```
shapiro.test(new_model$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  new_model$residuals
## W = 0.99484, p-value = 0.1556
```

```
bptest(new_model)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  new_model
## BP = 0.75176, df = 1, p-value = 0.3859
```

```
# The new p-value for the new_model is 0.3859, which is more than the alpha level, which means we fail

stanResidualsPartF = rstandard(new_model)

qqnorm(stanResidualsPartF,
       ylab="Standardized Residuals", xlab="Normal Scores", main="QQ plot of residuals for Part F")
qqline(stanResidualsPartF)
```



Now we've repeated the same tests, and we've failed to reject the null hypothesis in both times. Therefore, now the errors are likely (not necessarily) to be normally distributed, as well as likely having the same underlying variance.