Classification&Regression problem

Khanin Sisaengsuwanchai

2022-06-07

1) In this practice, I will predict party affiliation from voting records in house_votes dataset using tree-based methods.

```
# Download the data from the website
house_votes <- read.table("https://archive.ics.uci.edu/ml/machine-learning-databases/voting-records/hous

# Change the col names to be understanable according to .names file
colnames(house_votes) <- c('ClassName','hi','waterproject', 'adoption', 'physician', 'elsalvador', 'reli
'anti', 'aid', 'mx', 'immigration', 'synfuels', 'education', 'superfund', 'crime', 'dutyfree',
'exportSF')
```

```
# Change the types of columns from characters to be categorical data
names <- 1:17 # number of columns
house_votes[,names] <- lapply(house_votes[,names] , factor)

# head(house_votes) # Show top 6 rows
summary(house_votes) # Describe the data
```

```
##       ClassName   hi      waterproject adoption physician elsalvador religious
##  democrat  :267   ?: 12   ?: 48        ?: 11    ?: 11     ?: 15      ?: 11
##  republican:168   n:236   n:192        n:171    n:247     n:208      n:152
##                   y:187   y:195        y:253    y:177     y:212      y:272
##  anti    aid     mx      immigration synfuels education superfund crime
##  ?: 14   ?: 15   ?: 22   ?:  7       ?: 21    ?: 31     ?: 25     ?: 17
##  n:182   n:178   n:206   n:212       n:264    n:233     n:201     n:170
##  y:239   y:242   y:207   y:216       y:150    y:171     y:209     y:248
##  dutyfree exportSF
##  ?: 28    ?:104
##  n:233    n: 62
##  y:174    y:269
```

```
# Note: All the columns are qualitative.
```

Partition data into train and testing dataset with 80:20 ratio

```
set.seed(7)
# Use createDataPartition from the caret lib to help us split the data randomly
trainRows <- createDataPartition(house_votes$ClassName, p=0.8, list=FALSE)
```

```
train <- house_votes[trainRows,]
test  <- house_votes[-trainRows,]
# Check # of rows for train and test after spliting
dim(train)
```

```
## [1] 349  17
```

```
dim(test)
```

```
## [1] 86 17
```

```
head(train)
```

```
##    ClassName hi waterproject adoption physician elsalvador religious anti aid
## 1 republican  n            y        n         y          y         y    n   n
## 2 republican  n            y        n         y          y         y    n   n
## 3   democrat  ?            y        y         ?          y         y    n   n
## 4   democrat  n            y        y         n          ?         y    n   n
## 5   democrat  y            y        y         n          y         y    n   n
## 6   democrat  n            y        y         n          y         y    n   n
##   mx immigration synfuels education superfund crime dutyfree exportSF
## 1  n           y        ?         y         y     y        n        y
## 2  n           n        n         y         y     y        n        ?
## 3  n           n        y         n         y     y        n        n
## 4  n           n        y         n         y     n        n        y
## 5  n           n        y         ?         y     y        y        y
## 6  n           n        n         n         y     y        y        y
```

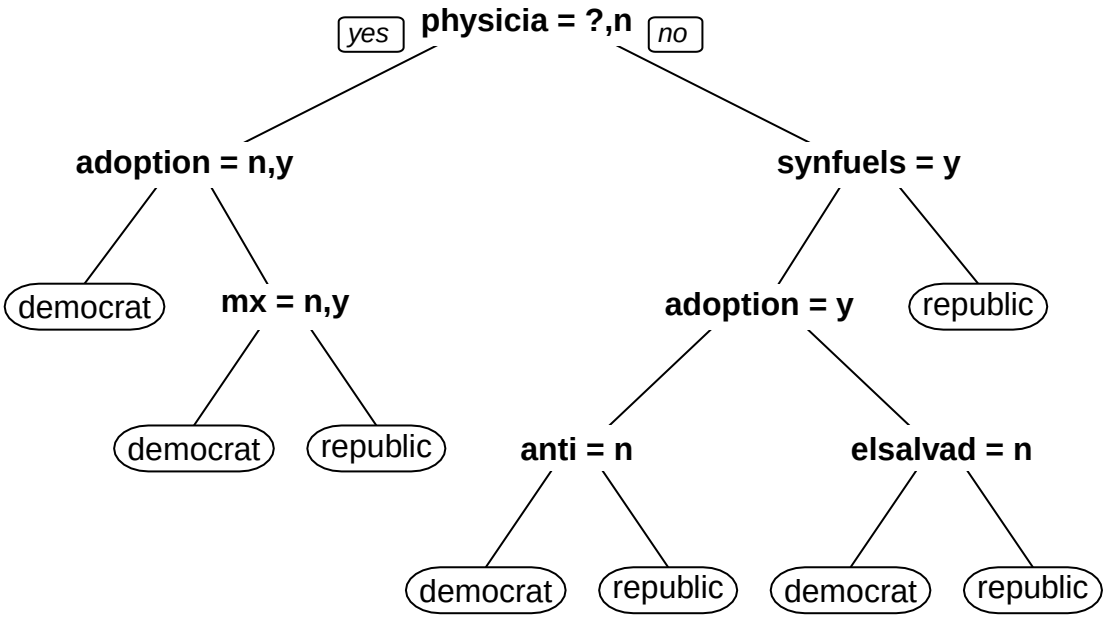
```
# I believe 349 rows and 86 rows for training and test are adequate for training and verifying the mode
```

```
set.seed(5)
votes.full_tree = rpart(ClassName ~ ., data = train, method="class", minsplit =5, minbucket = 2,
                        maxdepth = 10, cp=0)
# Fit the full tree
# Visualize the full tree
printcp(votes.full_tree)
```

```
##
## Classification tree:
## rpart(formula = ClassName ~ ., data = train, method = "class",
##     minsplit = 5, minbucket = 2, maxdepth = 10, cp = 0)
##
## Variables actually used in tree construction:
## [1] adoption   anti       elsalvador mx         physician  synfuels
##
## Root node error: 135/349 = 0.38682
##
## n= 349
##
##          CP nsplit rel error  xerror     xstd
## 1 0.8962963      0  1.000000 1.00000 0.067395
```

```
## 2 0.0111111      1  0.103704 0.10370 0.027154
## 3 0.0074074      5  0.059259 0.12593 0.029788
## 4 0.0000000      7  0.044444 0.15556 0.032908
```

```
prp(votes.full_tree)
```




```
# Assess the model via test dataset
pred.full_tree = predict(votes.full_tree, newdata = test, type = "class")
table(pred.full_tree, test$ClassName)
```

```
##
## pred.full_tree democrat republican
##     democrat         49          0
##     republican        4         33
```

```
mean(pred.full_tree != test$ClassName) # Test error rates
```

```
## [1] 0.04651163
```

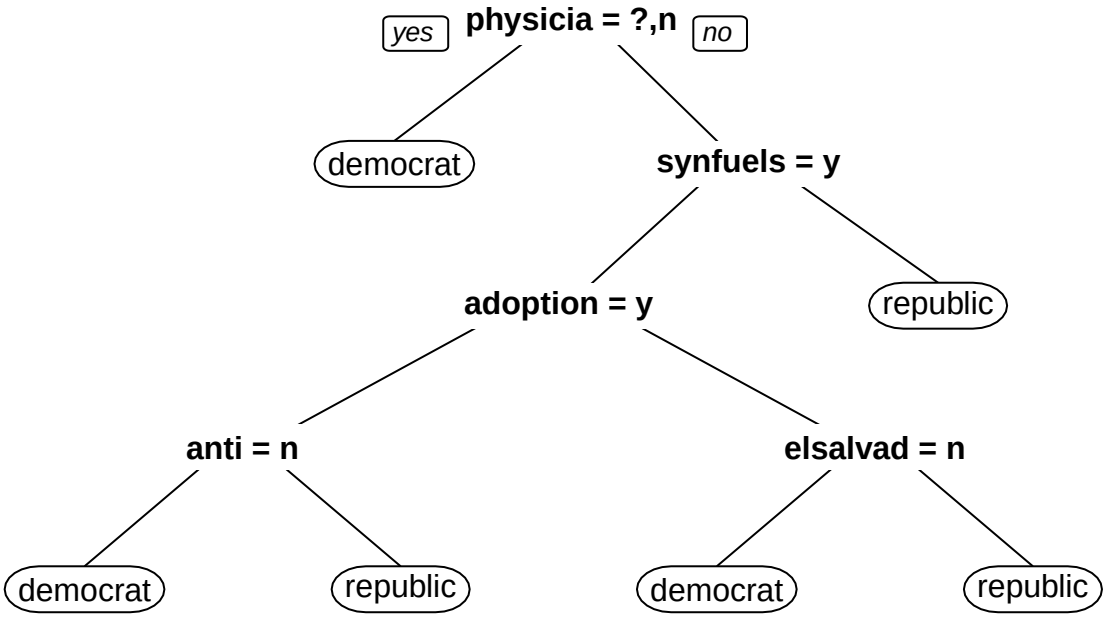
```
# According to the full tree, the model selects only physicia, adoption, mx, synfuels, adoption
# , anti, elsalvad variables as predictors because of their importance.
# This means, according to the tree model, other variables do not help explain this phenomenon with the
# same performance. The model performs really well with 4.65% test error.
```

Tree pruning

```
set.seed(5)
votes.pruned_tree<- prune(votes.full_tree,  cp=0.01)
# Specify cost complexity parameter to be 0.01
# Pruning

# Visualize the pruned tree
printcp(votes.pruned_tree)
```

```
##
## Classification tree:
## rpart(formula = ClassName ~ ., data = train, method = "class",
##     minsplit = 5, minbucket = 2, maxdepth = 10, cp = 0)
##
## Variables actually used in tree construction:
## [1] adoption   anti       elsalvador physician  synfuels
##
## Root node error: 135/349 = 0.38682
##
## n= 349
##
##         CP nsplit rel error  xerror     xstd
## 1 0.896296      0  1.000000 1.00000 0.067395
## 2 0.011111      1  0.103704 0.10370 0.027154
## 3 0.010000      5  0.059259 0.12593 0.029788
```

```
prp(votes.pruned_tree)
```




```
# Assess the model with test dataset
pred.pruned_tree = predict(votes.pruned_tree, newdata = test, type = "class")
table(pred.pruned_tree, test$ClassName)
```

```
##
## pred.pruned_tree democrat republican
##       democrat         49          0
##       republican        4         33
```

```
mean(pred.pruned_tree != test$ClassName) # Test error rates
```

```
## [1] 0.04651163
```

```
# According to the pruned tree with cost complexity parameter to be 0.01,
# the model has only 6 terminal nodes from 8.
# The model performs really well with 4.65% test error that is the same as the full tree.
```

```
# Given a new house member: n,n,n,y,y,y,y,n,n,y,n,y,n,y,n,n. What party does your tree predict for this

dim(test)
```

```
## [1] 86 17
```

```
# Add a new row
test[nrow(test) + 1,] = as.factor(c("democrat", "n", "n", "n", "y", "y", "y", "y", "n", "n", "y", "n",
                        "y", "n", "n"))
# Please ignore the first column (democrat). It is only a random variable.
# Show the added row
test[nrow(test),]
```

```
##    ClassName hi waterproject adoption physician elsalvador religious anti aid
## 87  democrat  n            n        n         y          y         y    y   n
##    mx immigration synfuels education superfund crime dutyfree exportSF
## 87  n           y        n         y         n     y        n        n
```

```
# Predict
predict(votes.pruned_tree, newdata = test[nrow(test),],
        type = "class") # Predict only a new row
```

```
##         87
## republican
## Levels: democrat republican
```

```
## According to the pruned tree model, it shows that the given voting data votes republican.
```

2) In the second practice, I will predict the number of rings on the abalone shell on abalone dataset using regression methods.

```
# Download abalone data
abalone <- read.table("https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data",

# Change the col names to be understanable according to .names file
colnames(abalone) <- c('Sex','Length','Diameter', 'Height', 'Whole_weight', 'Shucked_weight', 'Viscera_w
'Shell_weight', 'Rings')

# Change the type to be qualitative for sex
abalone$Sex = as.factor(abalone$Sex)

# Check datatype of all columns
sapply(abalone, class)
```

```
##            Sex         Length       Diameter         Height   Whole_weight
##       "factor"      "numeric"      "numeric"      "numeric"      "numeric"
## Shucked_weight Viscera_weight   Shell_weight          Rings
##      "numeric"      "numeric"      "numeric"      "integer"
```

```
head(abalone)
```

```
##   Sex Length Diameter Height Whole_weight Shucked_weight Viscera_weight
## 1   M  0.455    0.365  0.095       0.5140         0.2245         0.1010
## 2   M  0.350    0.265  0.090       0.2255         0.0995         0.0485
```

```
## 3   F  0.530    0.420  0.135       0.6770          0.2565          0.1415
## 4   M  0.440    0.365  0.125       0.5160          0.2155          0.1140
## 5   I  0.330    0.255  0.080       0.2050          0.0895          0.0395
## 6   I  0.425    0.300  0.095       0.3515          0.1410          0.0775
##   Shell_weight Rings
## 1        0.150    15
## 2        0.070     7
## 3        0.210     9
## 4        0.155    10
## 5        0.055     7
## 6        0.120     8
```

```
# Partition the data with 3133 training dataset observations and 1044 test observations
set.seed(7)
train_indices <- sample(nrow(abalone), 3133)

abalone.train <- abalone[train_indices, ] #select train
Rings.train   <- abalone$Rings[train_indices]

abalone.test <- abalone[-train_indices, ] # select test
Rings.test   <- abalone$Rings[-train_indices]

# Check the dim of the data
dim(abalone.train)
```

```
## [1] 3133    9
```

```
dim(abalone.test)
```

```
## [1] 1044    9
```

Explore the data

```
summary(abalone)
```

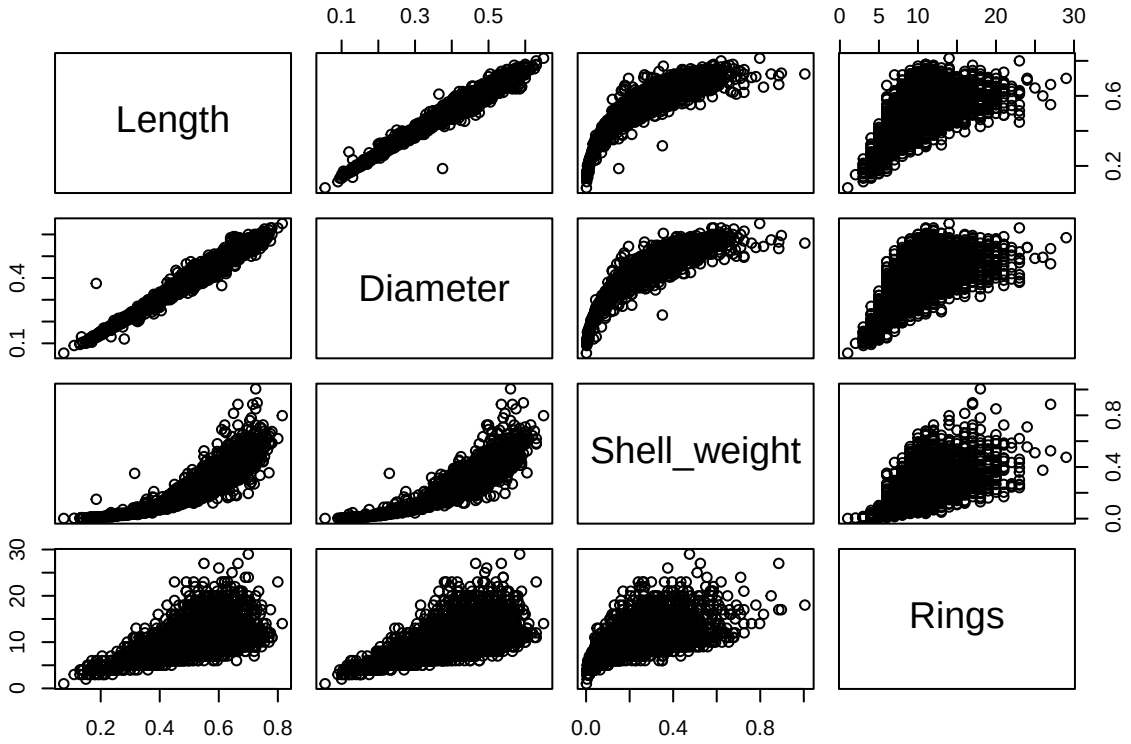
```
##  Sex          Length         Diameter          Height        Whole_weight
##  F:1307   Min.   :0.075   Min.   :0.0550   Min.   :0.0000   Min.   :0.0020
##  I:1342   1st Qu.:0.450   1st Qu.:0.3500   1st Qu.:0.1150   1st Qu.:0.4415
##  M:1528   Median :0.545   Median :0.4250   Median :0.1400   Median :0.7995
##           Mean   :0.524   Mean   :0.4079   Mean   :0.1395   Mean   :0.8287
##           3rd Qu.:0.615   3rd Qu.:0.4800   3rd Qu.:0.1650   3rd Qu.:1.1530
##           Max.   :0.815   Max.   :0.6500   Max.   :1.1300   Max.   :2.8255
##  Shucked_weight   Viscera_weight    Shell_weight        Rings
##  Min.   :0.0010   Min.   :0.0005   Min.   :0.0015   Min.   : 1.000
##  1st Qu.:0.1860   1st Qu.:0.0935   1st Qu.:0.1300   1st Qu.: 8.000
##  Median :0.3360   Median :0.1710   Median :0.2340   Median : 9.000
##  Mean   :0.3594   Mean   :0.1806   Mean   :0.2388   Mean   : 9.934
##  3rd Qu.:0.5020   3rd Qu.:0.2530   3rd Qu.:0.3290   3rd Qu.:11.000
##  Max.   :1.4880   Max.   :0.7600   Max.   :1.0050   Max.   :29.000
```

```r
# According to the sumaary function, there seem to have a lot of outliers in Height, Whole_weight,
# Shucked_weight, Viscera_weight, Shell_weight, and Rings. I will try to investigate further using corr
# and box plots.
```

```r
# Correlation plot
cor(abalone[, -1])
```

```
##                   Length  Diameter    Height Whole_weight Shucked_weight
## Length         1.0000000 0.9868116 0.8275536    0.9252612      0.8979137
## Diameter       0.9868116 1.0000000 0.8336837    0.9254521      0.8931625
## Height         0.8275536 0.8336837 1.0000000    0.8192208      0.7749723
## Whole_weight   0.9252612 0.9254521 0.8192208    1.0000000      0.9694055
## Shucked_weight 0.8979137 0.8931625 0.7749723    0.9694055      1.0000000
## Viscera_weight 0.9030177 0.8997244 0.7983193    0.9663751      0.9319613
## Shell_weight   0.8977056 0.9053298 0.8173380    0.9553554      0.8826171
## Rings          0.5567196 0.5746599 0.5574673    0.5403897      0.4208837
##                Viscera_weight Shell_weight     Rings
## Length              0.9030177    0.8977056 0.5567196
## Diameter            0.8997244    0.9053298 0.5746599
## Height              0.7983193    0.8173380 0.5574673
## Whole_weight        0.9663751    0.9553554 0.5403897
## Shucked_weight      0.9319613    0.8826171 0.4208837
## Viscera_weight      1.0000000    0.9076563 0.5038192
## Shell_weight        0.9076563    1.0000000 0.6275740
## Rings               0.5038192    0.6275740 1.0000000
```

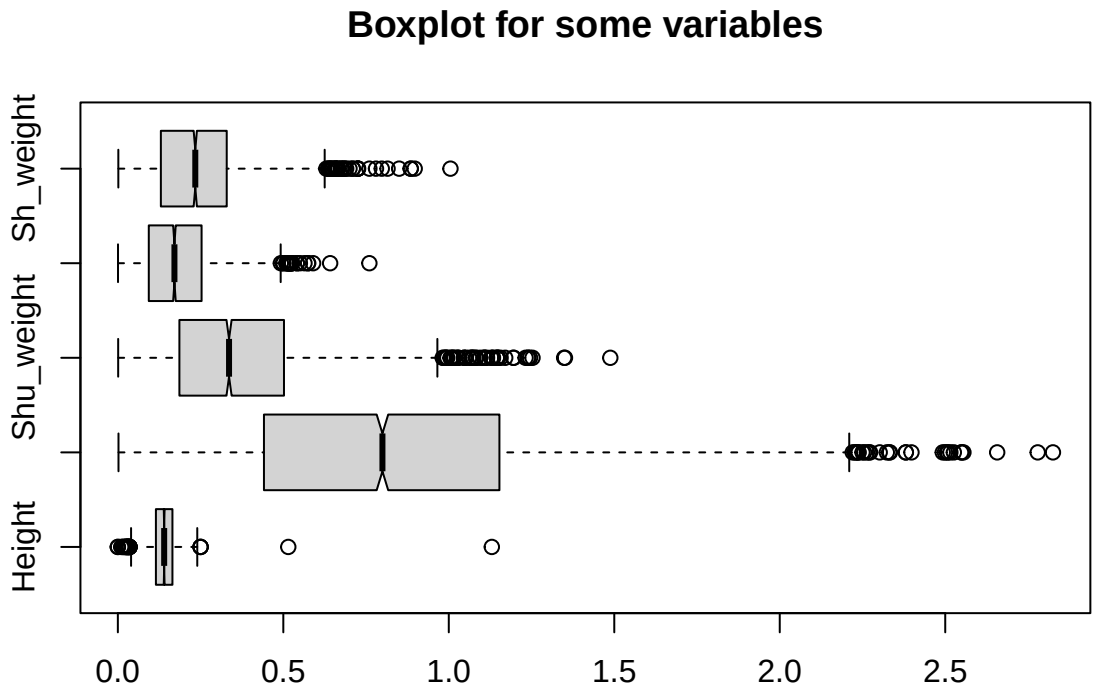
```r
#pairs(abalone)
# According to the correlation, it is more likely that predictors have linear relationship with rings,
# indicating by quite high correlation. However, many variables have high collinearity.

# Therefore, I will use scatter plot on Rings with Shell_weight, Length, and Diameter to
# verify such linear relationships.

pairs(abalone[, c(2,3,8,9)])
```



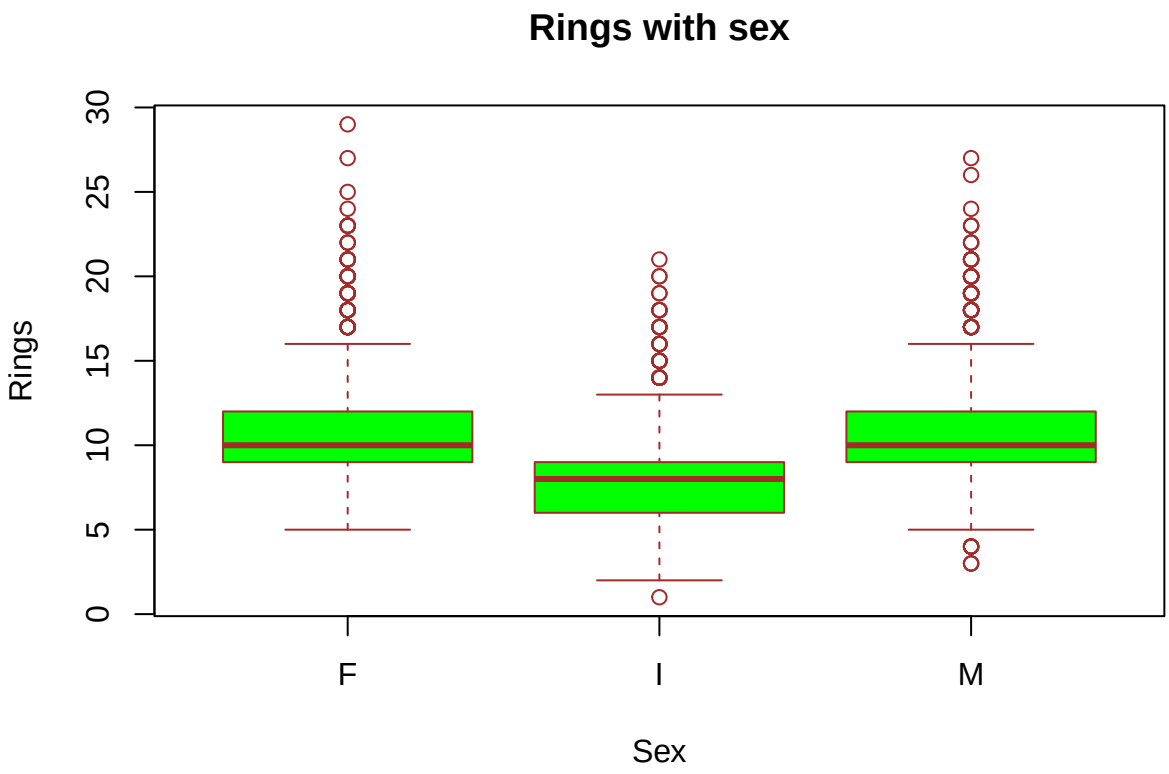

```
# As I expected, Rings with other 3 variables seem to have linear relationship, and Length
# and Diameter have a very high correlation value.
```

Boxplot

```
# Height, Whole_weight,
# Shucked_weight, Viscera_weight, Shell_weight, and Rings


boxplot(abalone$Height, abalone$Whole_weight, abalone$Shucked_weight, abalone$Viscera_weight,
        abalone$Shell_weight,
main = "Boxplot for some variables",
names = c("Height", "Who_weight", "Shu_weight", "Visc_weight", "Sh_weight"),
horizontal = TRUE,
notch = TRUE
)
```




```
# As I expected, these predictors have a lot of outliers.

boxplot(Rings~Sex,
data=abalone,
main="Rings with sex",
xlab="Sex",
ylab="Rings",
col="green",
border="brown"
)
```




```
# Sex has an affect on Rings, especially with I sex.
```

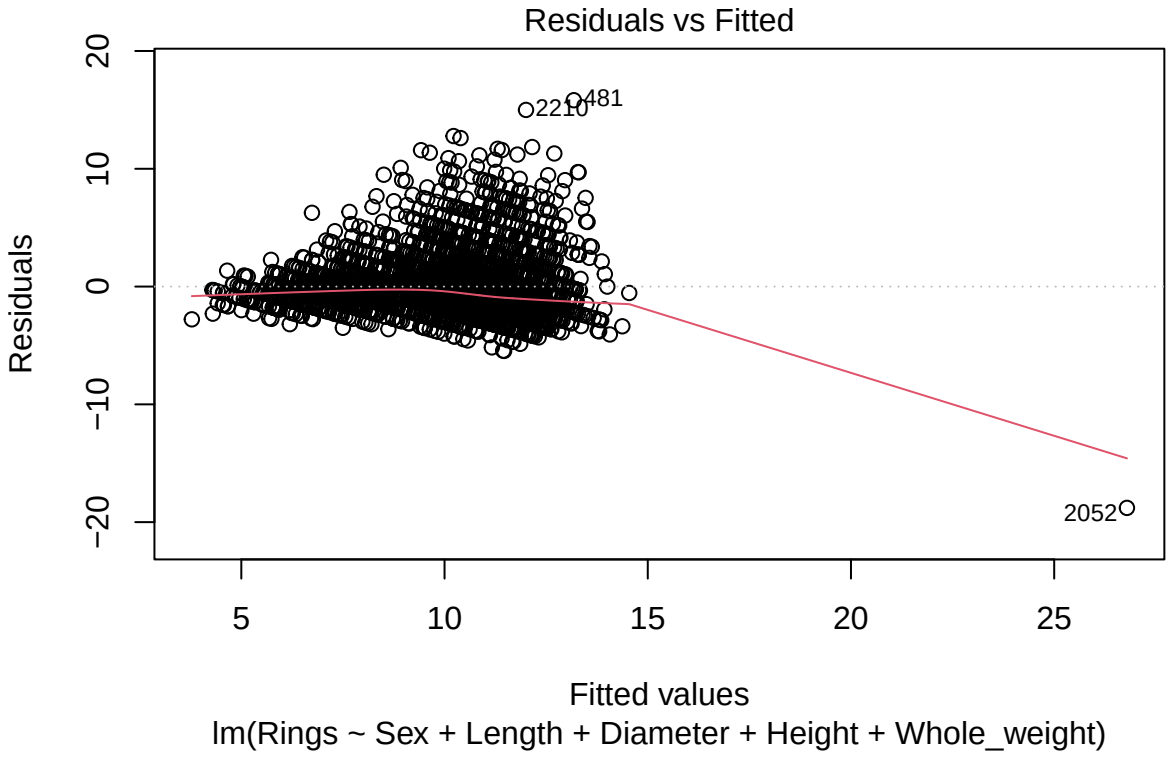
Fit linear regression with the first 5 predictors on training dataset (3,133 data-points)

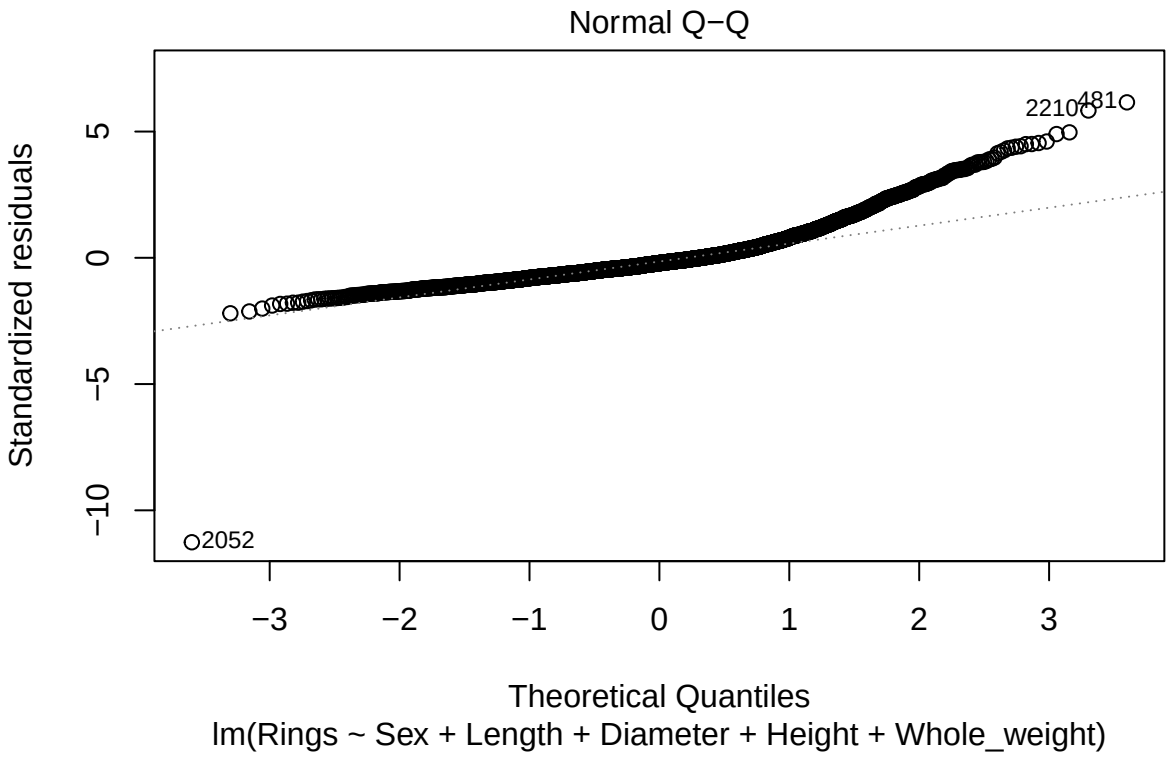
```
lm.fit1 = lm(Rings ~ Sex+Length+Diameter+Height+Whole_weight, data = abalone.train)
summary(lm.fit1)
```

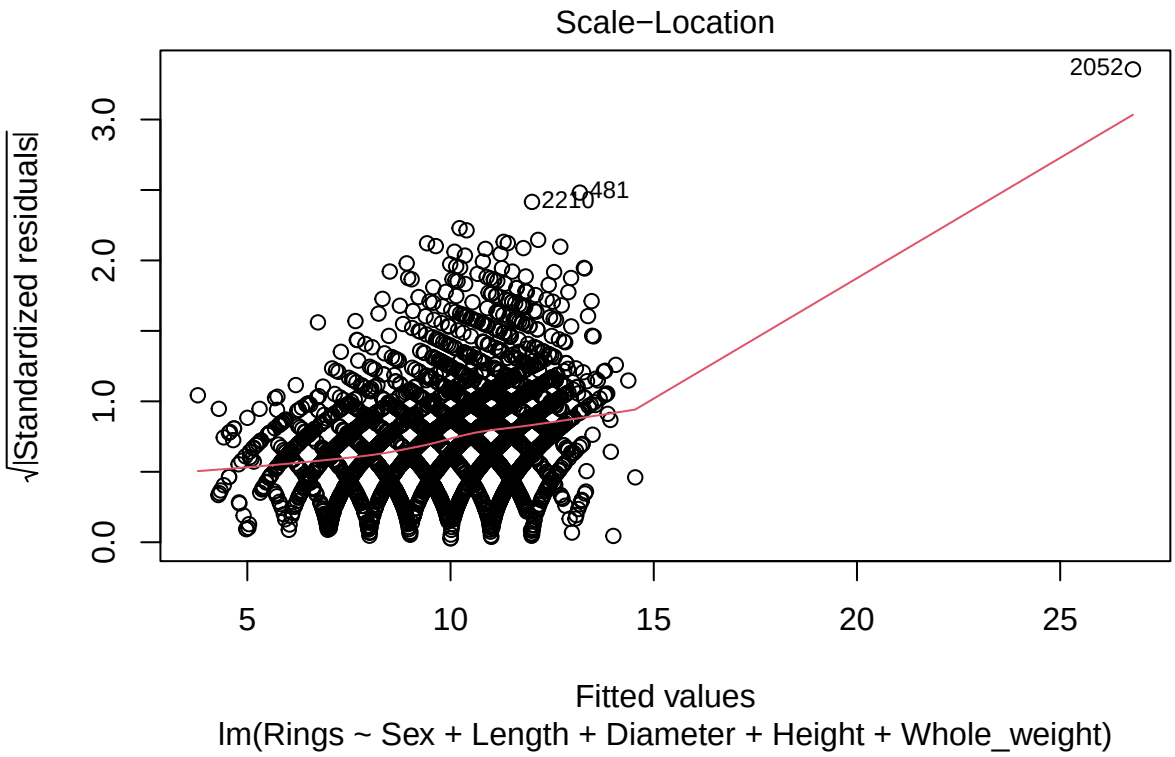
```
##
## Call:
## lm(formula = Rings ~ Sex + Length + Diameter + Height + Whole_weight,
##     data = abalone.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.7900  -1.6125  -0.5372   0.8534  15.8191
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.2699     0.3871  11.031  < 2e-16 ***
## SexI          -1.1355     0.1381  -8.221 2.91e-16 ***
## SexM          -0.1764     0.1128  -1.564    0.118
## Length        -9.3988     2.3405  -4.016 6.07e-05 ***
```

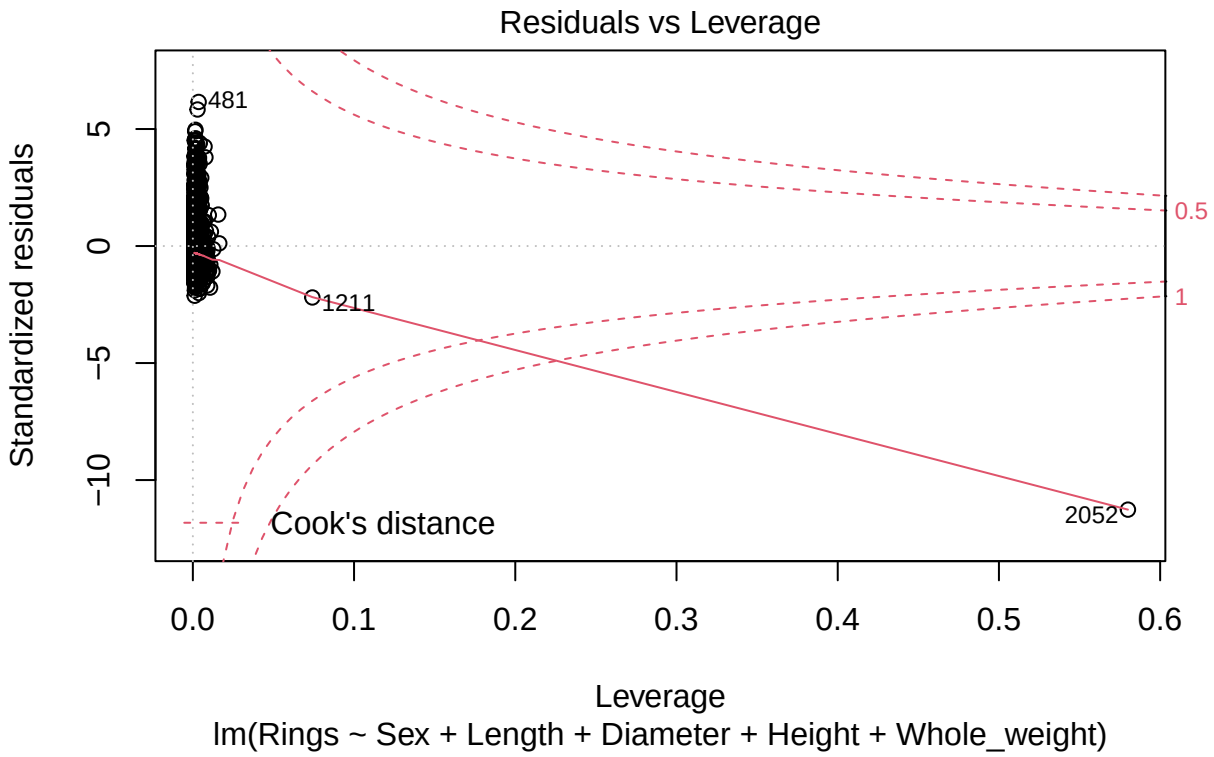
```
## Diameter      21.5130     2.8705   7.495 8.61e-14 ***
## Height        17.0264     1.9475   8.743  < 2e-16 ***
## Whole_weight  -0.1355     0.2577  -0.526    0.599
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.574 on 3126 degrees of freedom
## Multiple R-squared:  0.3614, Adjusted R-squared:  0.3602
## F-statistic: 294.9 on 6 and 3126 DF,  p-value: < 2.2e-16
```

```
plot(lm.fit1)
```




Normal Q−Q
Standardized residuals
5
0
−5
−10
2210
481
2052
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(Rings ~ Sex + Length + Diameter + Height + Whole_weight)


Scale−Location
√|Standardized residuals|
0.0
1.0
2.0
3.0
5
10
15
20
25
2052
2210
481
Fitted values
lm(Rings ~ Sex + Length + Diameter + Height + Whole_weight)




```
# According to the linear model, specifically for Sex variable, I use Female as a base line
# to handle the nominal variable. It's coefficient is added into the intercept of the model.

# Only about 36% of variance can be explained by this model, indicated by R-squared.
# Finally, Whole_weight has a high p-value and thus we might remove this variable when we do
# feature selection.
```

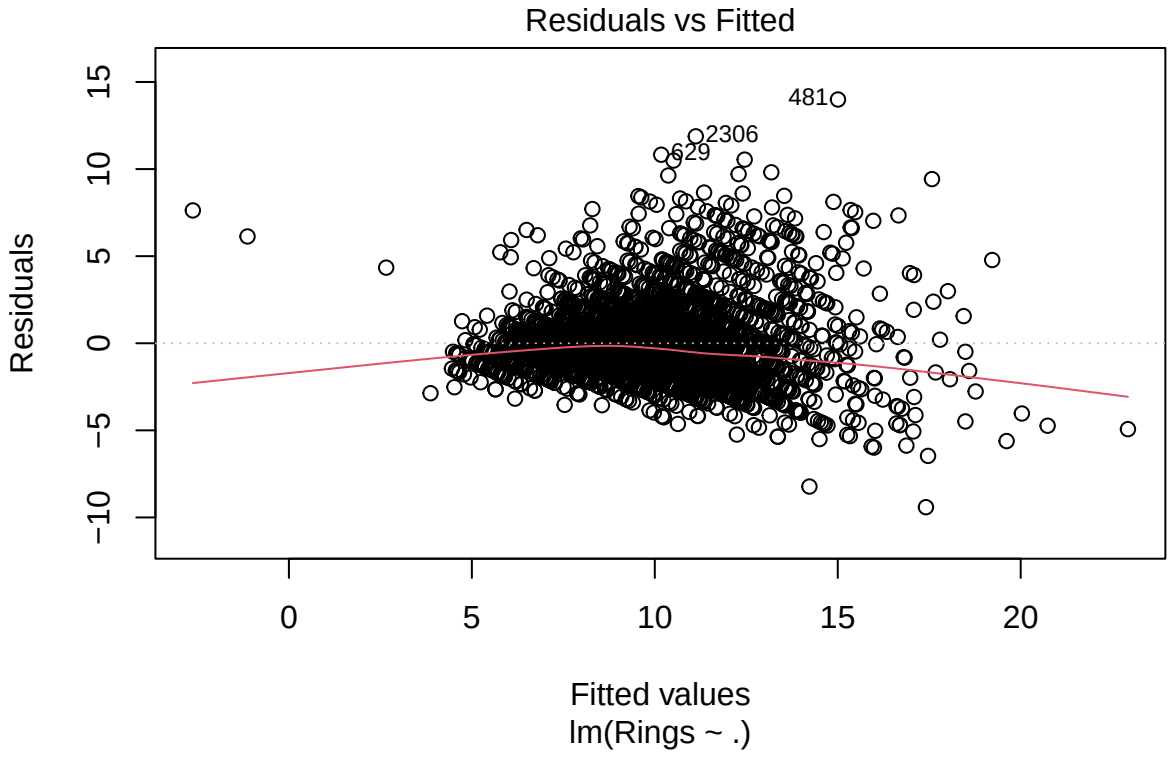
Fit the linear model on all 8 variables

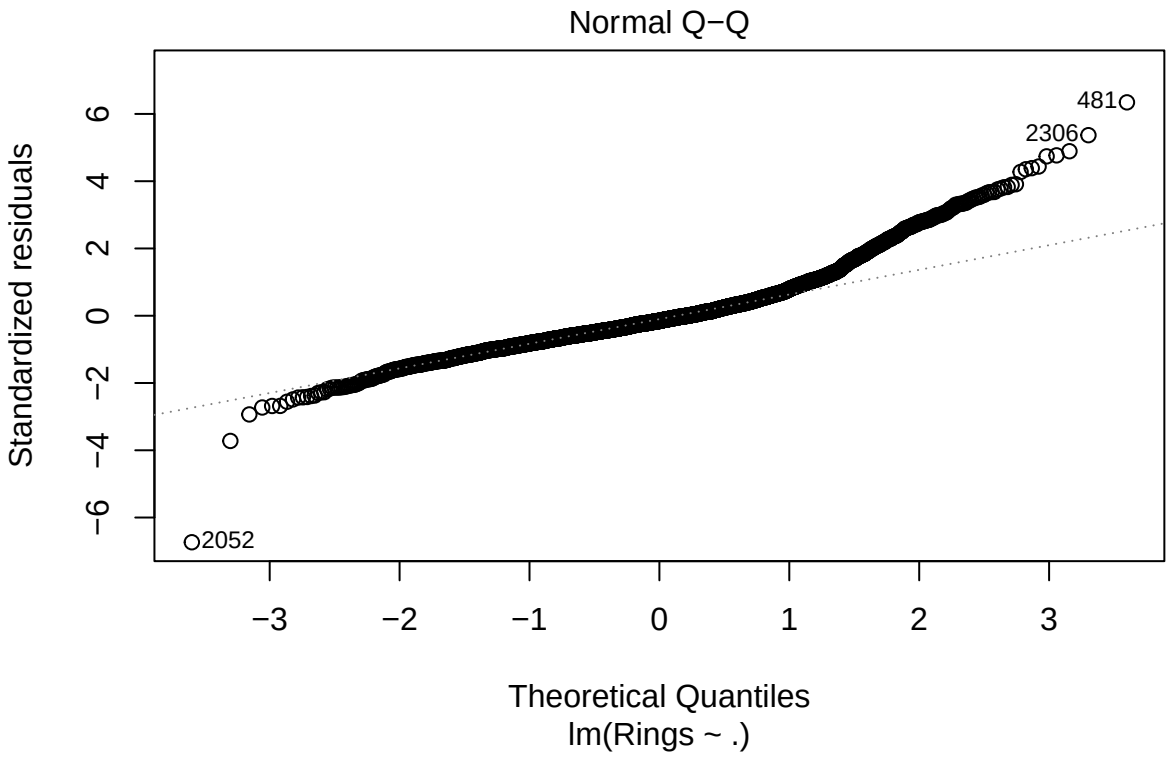
```
lm.fit2 = lm(Rings ~ ., data = abalone.train)
summary(lm.fit2)
```

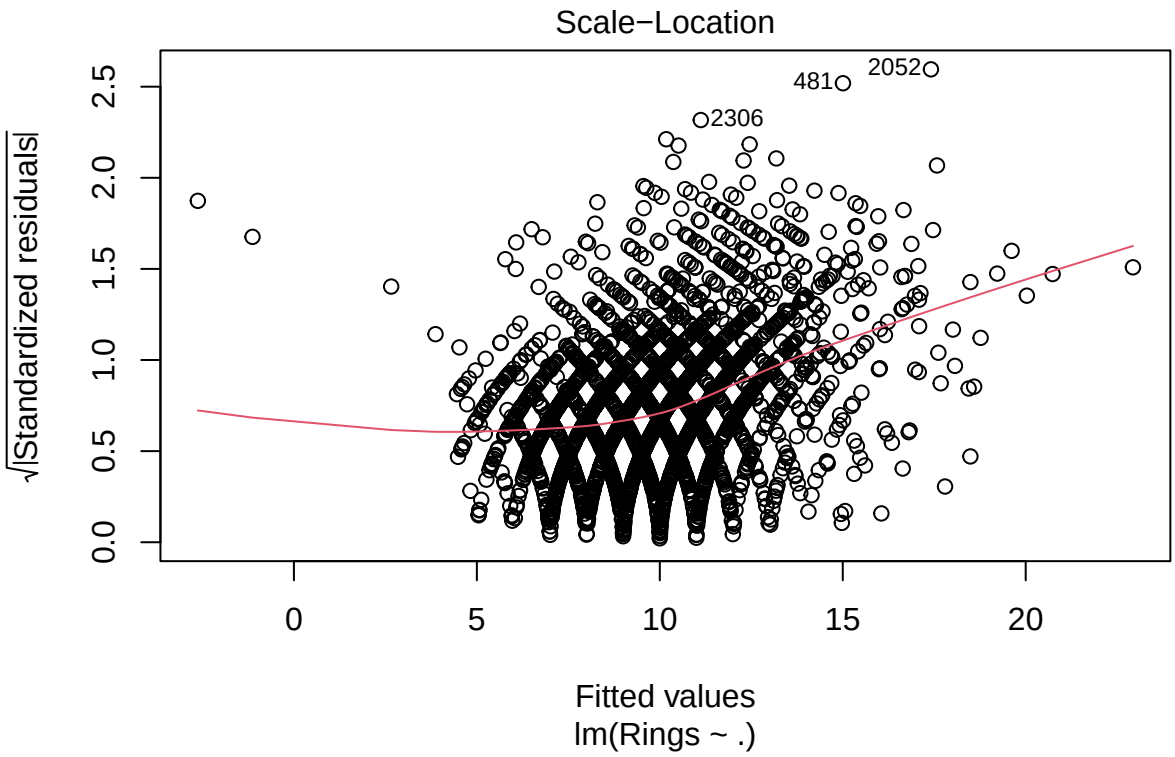
```
##
## Call:
## lm(formula = Rings ~ ., data = abalone.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.4107 -1.3145 -0.3424  0.8716 13.9939
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      4.078177   0.337450  12.085  < 2e-16 ***
```

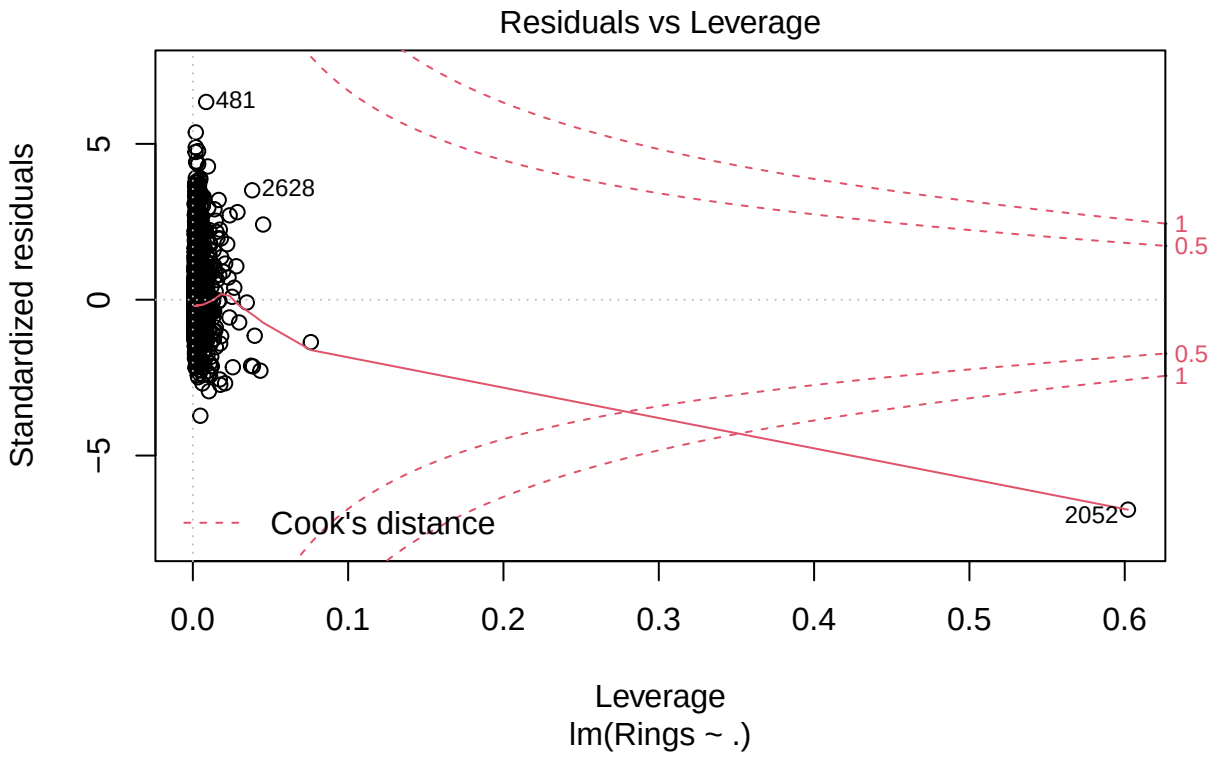
```
## SexI            -0.888952   0.119829  -7.418 1.52e-13 ***
## SexM             0.009134   0.097407   0.094    0.925
## Length          -0.325700   2.039552  -0.160    0.873
## Diameter        10.956844   2.502866   4.378 1.24e-05 ***
## Height           9.624771   1.696631   5.673 1.53e-08 ***
## Whole_weight     8.675730   0.820346  10.576  < 2e-16 ***
## Shucked_weight -19.500625   0.932037 -20.923  < 2e-16 ***
## Viscera_weight -10.022217   1.469600  -6.820 1.09e-11 ***
## Shell_weight     8.976809   1.288014   6.969 3.86e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.215 on 3123 degrees of freedom
## Multiple R-squared:  0.5276, Adjusted R-squared:  0.5262
## F-statistic: 387.5 on 9 and 3123 DF,  p-value: < 2.2e-16
```

```
plot(lm.fit2)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(Rings ~ .)
481
2306
2052
−6
−4
−2
0
2
4
6
−3
−2
−1
0
1
2
3


Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
2.5
481
2052
2306
0
5
10
15
20
Fitted values
lm(Rings ~ .)




```
# After using all 8 variables, the model's performance increases significantly.
# The adjusted R-squared, indicating how much variance the model can explain,
# increases from approximately 36% to 53%, and RSE reduces from about 2.57 to 2.215.

# Note: I use adjusted R-squared because the normal R-squared will always keep increasing
# when we add more predictors, but adjusted R-squared does not due to penalty terms.
```

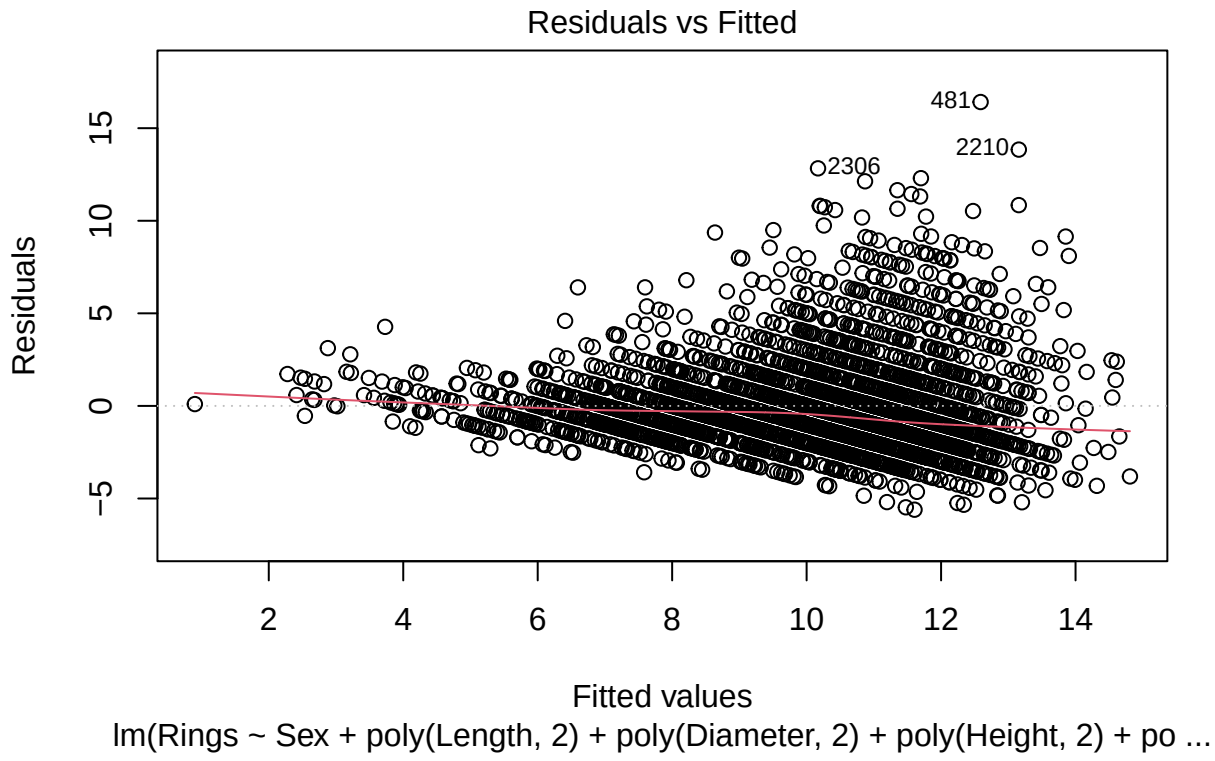
Quadratic regression models is the first 5 and first 8 predictors

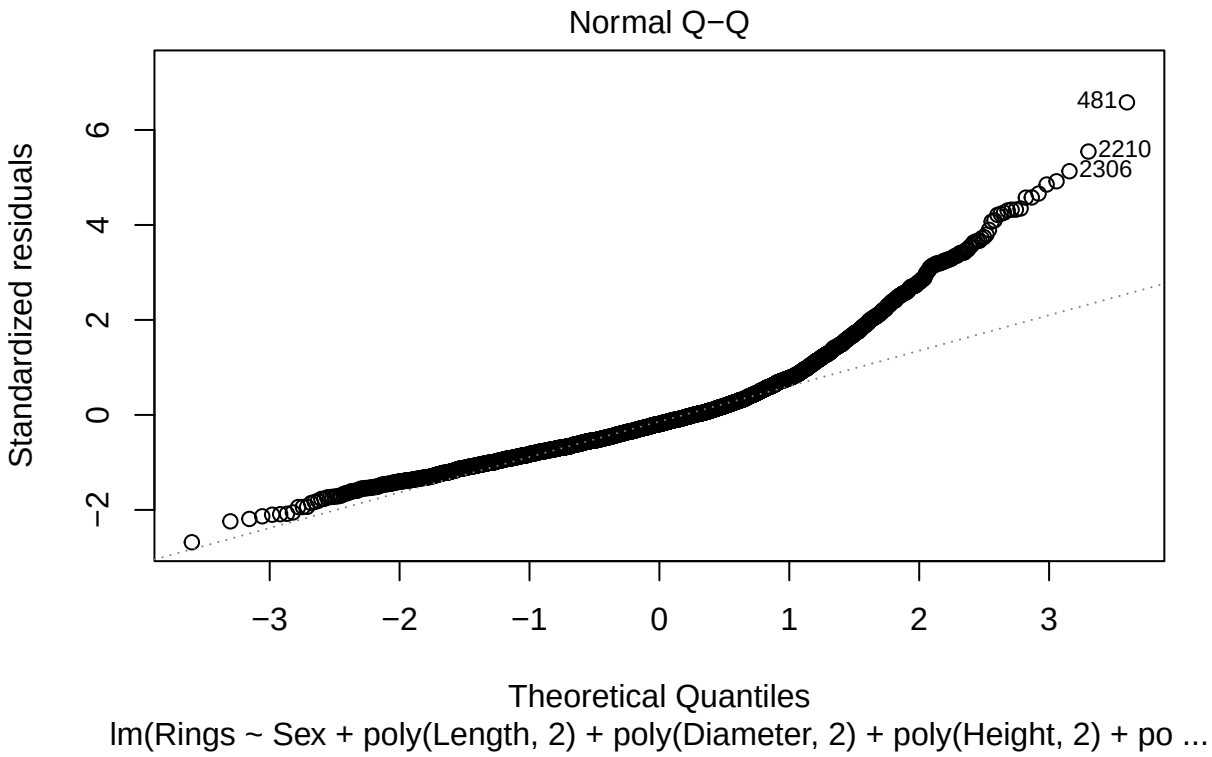
```
# Fit a quadratic model of the first 5 predictors
fit.q5 = lm(Rings ~ Sex+ poly(Length, 2)+ poly(Diameter, 2)+ poly(Height, 2)
            + poly(Whole_weight, 2), data = abalone.train)
summary(fit.q5)
```

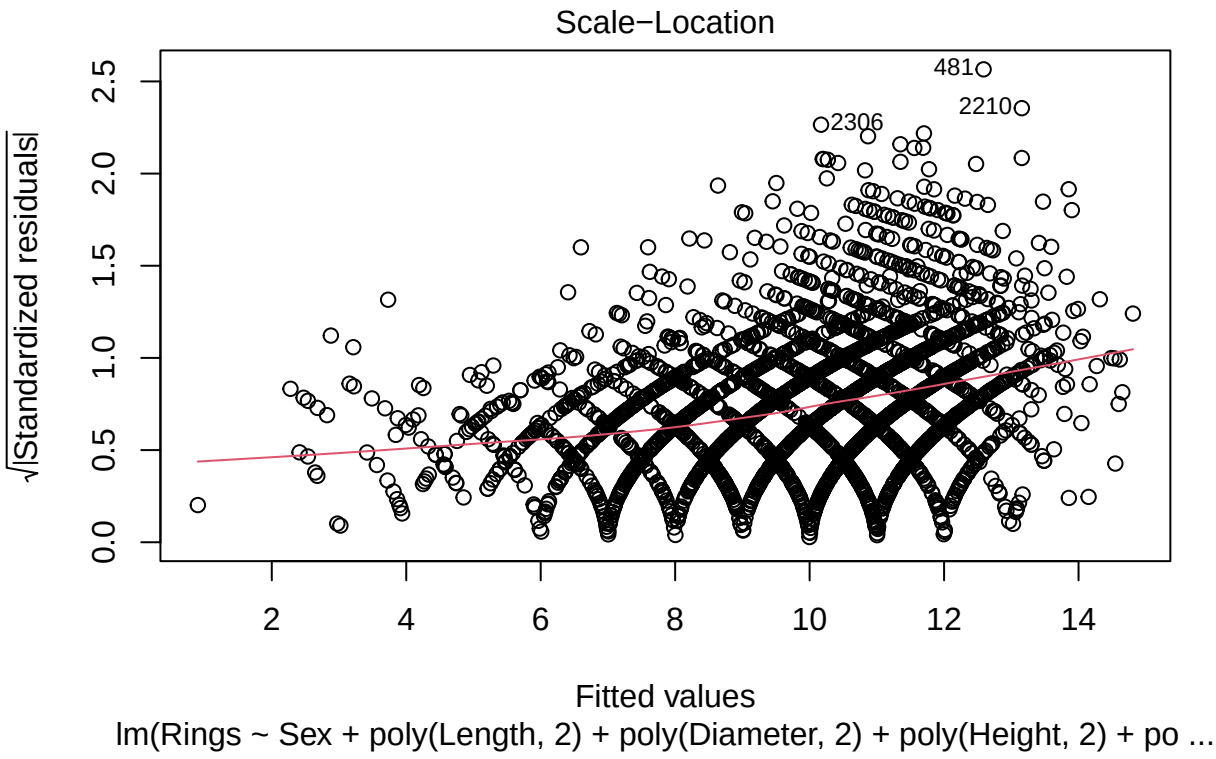
```
##
## Call:
## lm(formula = Rings ~ Sex + poly(Length, 2) + poly(Diameter, 2) +
##     poly(Height, 2) + poly(Whole_weight, 2), data = abalone.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.6049 -1.6082 -0.4756  0.9072 16.4121
##
```

```
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)              10.33911    0.08592 120.335  < 2e-16 ***
## SexI                     -0.98100    0.13784  -7.117 1.36e-12 ***
## SexM                     -0.20148    0.10976  -1.836 0.066504 .
## poly(Length, 2)1        -97.36569   17.81100  -5.467 4.95e-08 ***
## poly(Length, 2)2        -30.73307    9.94519  -3.090 0.002018 **
## poly(Diameter, 2)1       67.41762   17.56378   3.838 0.000126 ***
## poly(Diameter, 2)2        4.47099   10.04284   0.445 0.656212
## poly(Height, 2)1         71.28864    6.22393  11.454  < 2e-16 ***
## poly(Height, 2)2        -35.51807    3.71768  -9.554  < 2e-16 ***
## poly(Whole_weight, 2)1   48.25931   12.93309   3.731 0.000194 ***
## poly(Whole_weight, 2)2   -1.72511    4.23381  -0.407 0.683697
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.503 on 3122 degrees of freedom
## Multiple R-squared:  0.3972, Adjusted R-squared:  0.3953
## F-statistic: 205.7 on 10 and 3122 DF,  p-value: < 2.2e-16
```

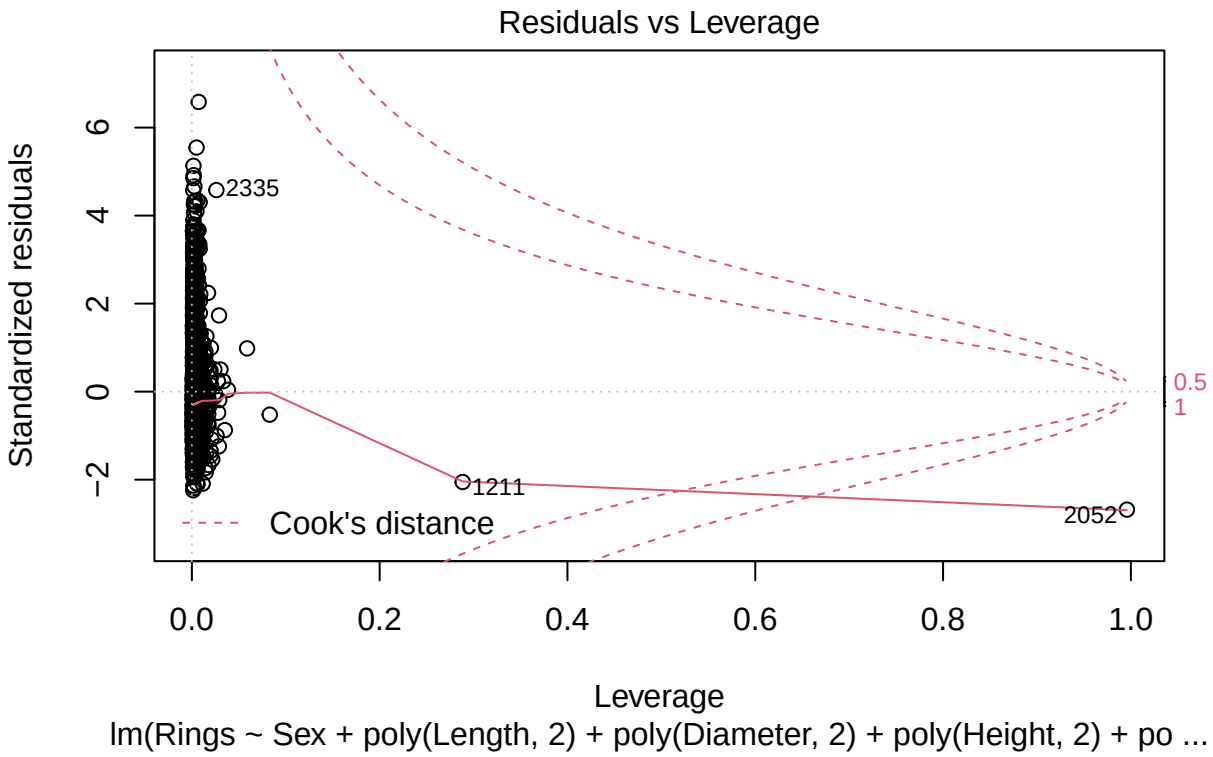
```
plot(fit.q5)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(Rings ~ Sex + poly(Length, 2) + poly(Diameter, 2) + poly(Height, 2) + po ...
481
2210
2306
−2
0
2
4
6
−3
−2
−1
0
1
2
3




```
## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced

## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced
```

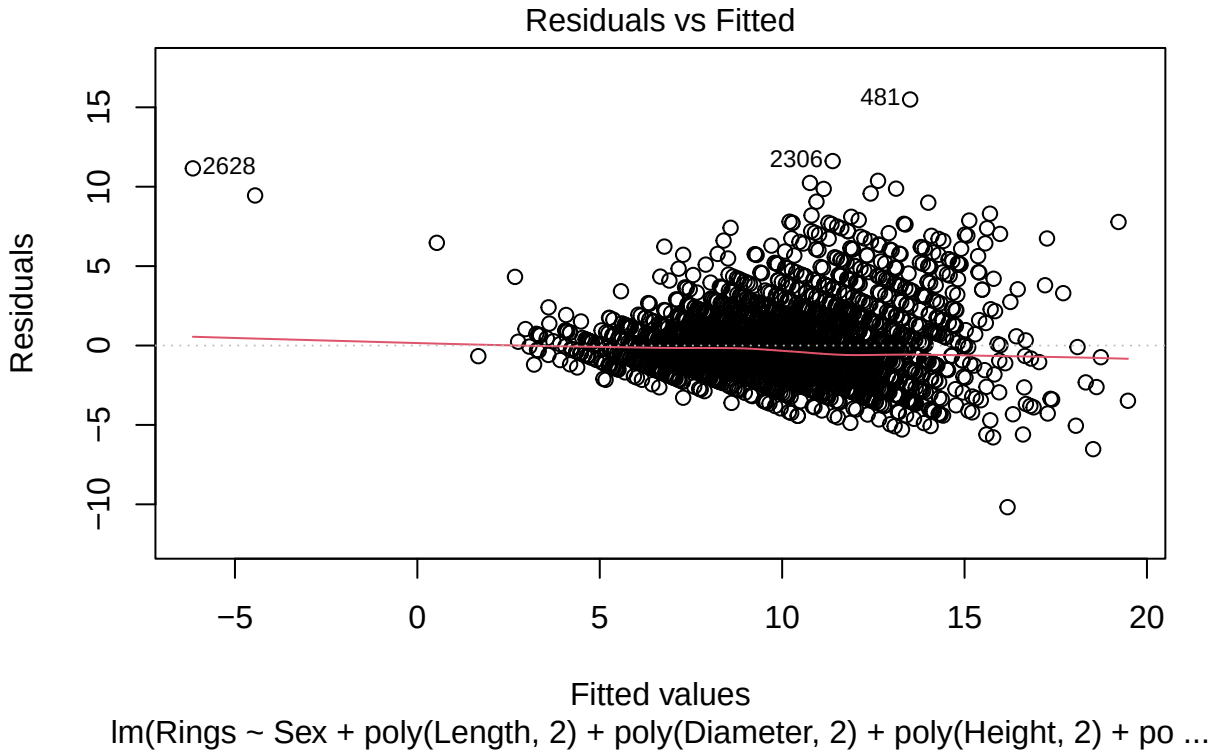


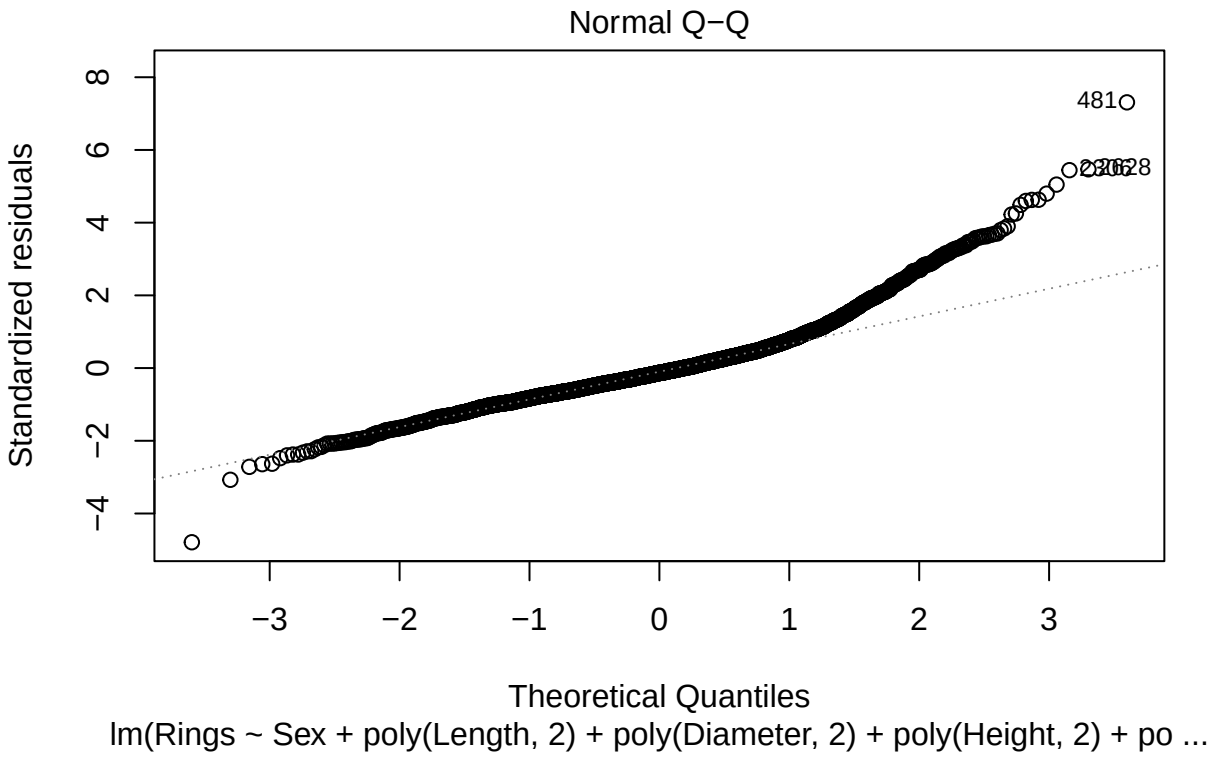

```
# Fit a quadratic model of the all 8 predictors
fit.q8 = lm(Rings ~ Sex+ poly(Length, 2)+ poly(Diameter, 2)+ poly(Height, 2)
            + poly(Whole_weight, 2) + poly(Shucked_weight, 2) + poly(Viscera_weight, 2)
            + poly(Shell_weight, 2), data = abalone.train)
summary(fit.q8)
```

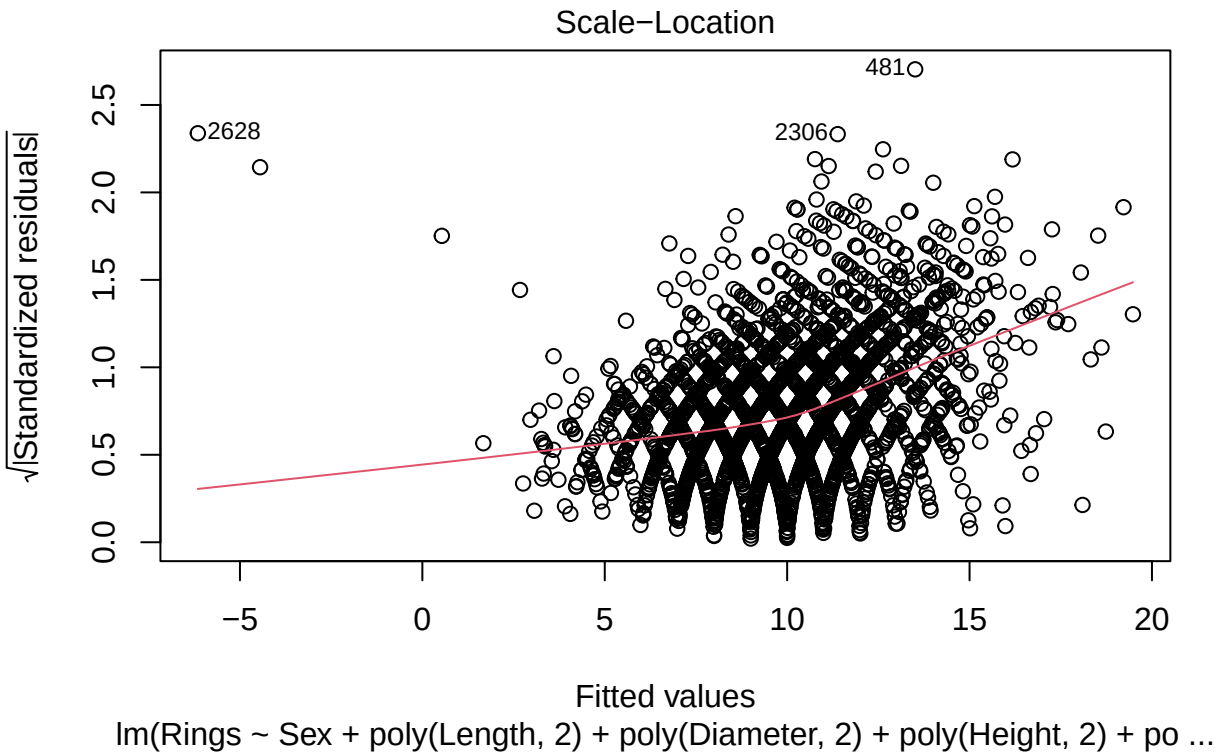
```
##
## Call:
## lm(formula = Rings ~ Sex + poly(Length, 2) + poly(Diameter, 2) +
##     poly(Height, 2) + poly(Whole_weight, 2) + poly(Shucked_weight,
##     2) + poly(Viscera_weight, 2) + poly(Shell_weight, 2), data = abalone.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.1790  -1.3074  -0.2847   0.8741  15.4912
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                10.17927    0.07386 137.810  < 2e-16 ***
## SexI                       -0.68269    0.11934  -5.720 1.16e-08 ***
## SexM                       -0.02982    0.09405  -0.317 0.751239
## poly(Length, 2)1          -39.98296   15.37496  -2.601 0.009352 **
## poly(Length, 2)2          -22.82260    8.56736  -2.664 0.007764 **
## poly(Diameter, 2)1         24.34919   15.20529   1.601 0.109398
## poly(Diameter, 2)2         -5.33962    8.62766  -0.619 0.536030
## poly(Height, 2)1           26.57977    5.58127   4.762 2.00e-06 ***
```

```
## poly(Height, 2)2          -12.20382    3.29281  -3.706 0.000214 ***
## poly(Whole_weight, 2)1    337.85814   25.84282  13.074  < 2e-16 ***
## poly(Whole_weight, 2)2    -62.95714   11.94432  -5.271 1.45e-07 ***
## poly(Shucked_weight, 2)1 -275.08108   12.84180 -21.421  < 2e-16 ***
## poly(Shucked_weight, 2)2   55.20620    6.87561   8.029 1.38e-15 ***
## poly(Viscera_weight, 2)1  -73.12085   10.58603  -6.907 5.96e-12 ***
## poly(Viscera_weight, 2)2   17.58045    5.79854   3.032 0.002450 **
## poly(Shell_weight, 2)1     90.07812   11.91839   7.558 5.35e-14 ***
## poly(Shell_weight, 2)2     -9.35980    5.47135  -1.711 0.087237 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.136 on 3116 degrees of freedom
## Multiple R-squared:  0.5616, Adjusted R-squared:  0.5594
## F-statistic: 249.5 on 16 and 3116 DF,  p-value: < 2.2e-16
```

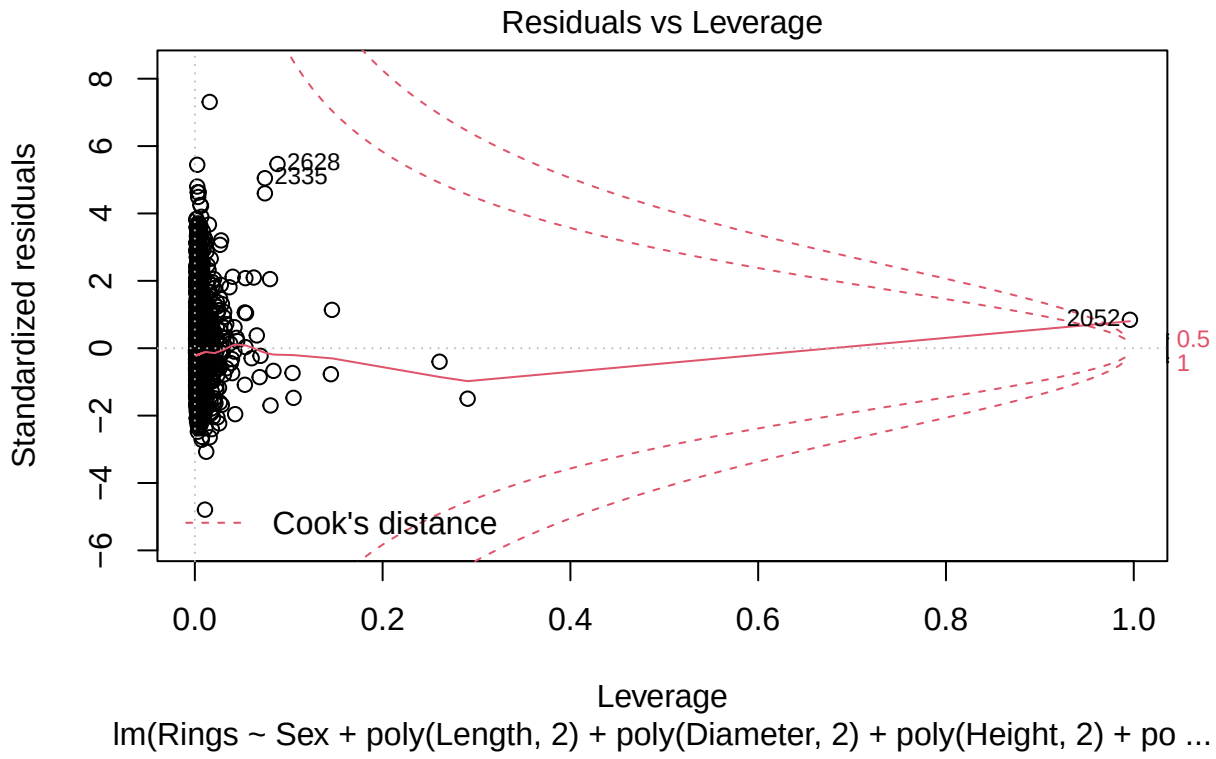
```
plot(fit.q8)
```




Normal Q−Q
Standardized residuals
8
6
4
2
0
−2
−4
481
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(Rings ~ Sex + poly(Length, 2) + poly(Diameter, 2) + poly(Height, 2) + po ...




```
## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced

## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced
```




```
# Again, to handle the nominal variable, I use Female as a base line for dummy variables.
# It's coefficient is added into the intercept of the models.

# After adding the polynomial terms in both model, adjusted R-squared are increased and
# RSEs are decreased.

# Note: I cannot make the Sex variable becomes polynomial because it is a qualitative data.
```

Ridge regression

```
# Select 784 observations from training dataset (Validation dataset)

set.seed(7)
valid_indices <- sample(nrow(abalone.train), 784)

abalone.valid  <- abalone.train[valid_indices, ] #select valid observations
abalone.train2 <- abalone.train[-valid_indices, ] #select the new train observations
Rings.valid    <- abalone.train$Rings[valid_indices]

# Check the dim of the data
dim(abalone.valid)
```

```
## [1] 784   9
```

```r
dim(abalone.train2)
```

```
## [1] 2349    9
```

```r
length(Rings.valid)
```

```
## [1] 784
```

```r
# Generate a new training dataset for ridge regression
x <- model.matrix(Rings ~ ., abalone.train)[, -1]
y <- abalone.train$Rings

# Split train and test randomly
set.seed(1)
train2.indice <- sample(nrow(abalone.train), 2349)

x.train <- x[train2.indice,]
y.train <- y[train2.indice]
x.valid <- x[-train2.indice,]
y.valid <- y[-train2.indice]

dim(x.train)
```

```
## [1] 2349    9
```

```r
length(y.train)
```

```
## [1] 2349
```

```r
dim(x.valid)
```

```
## [1] 784   9
```

```r
length(y.valid)
```

```
## [1] 784
```

```r
# Fit the ridge regression model
grid <- 10^seq(10, -2, length = 100) # Create a set of lambdas
ridge.mod <- glmnet(x.train, y.train, alpha = 0, lambda = grid)
dim(coef(ridge.mod)) # 10 variables and 100 lambda
```

```
## [1]  10 100
```

```r
# Run a for loop to choose the best lambda
lambda_records = data.frame(lambda = numeric(0), MSE = numeric(0))

# Use the optimal lambda and calculate MSE
```

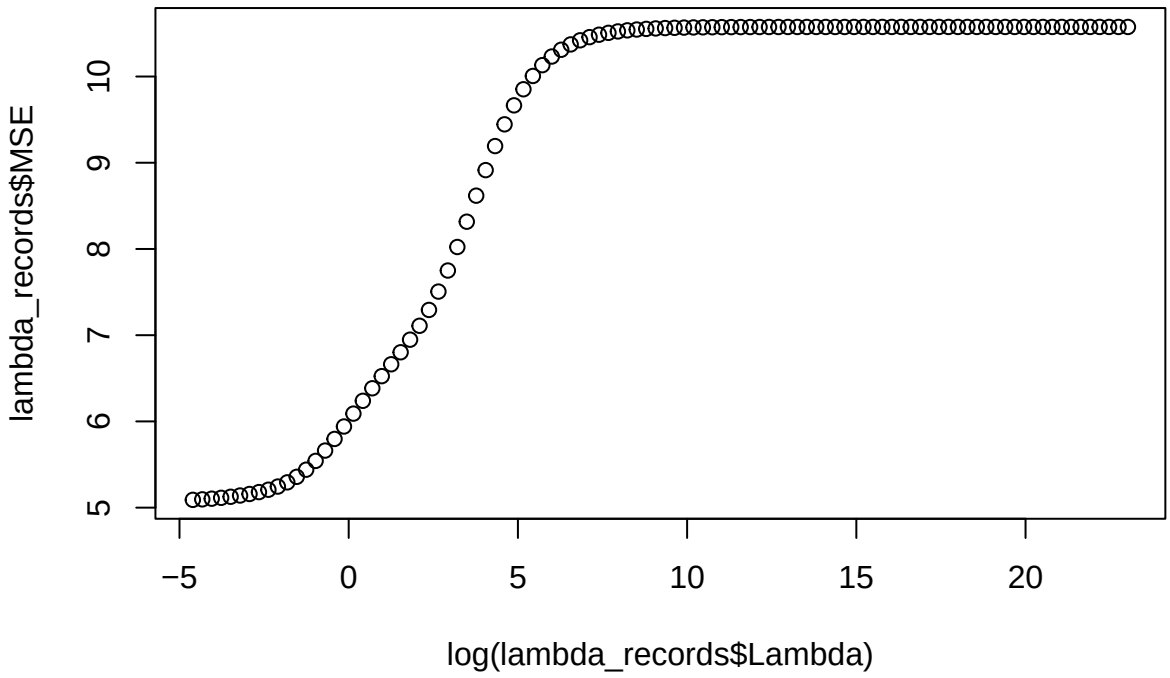
```
errorCal.ridge = function(ridge_model, vali, y_vali, lambda_val){
  ridge.pred <- predict(ridge_model, s = lambda_val, newx = vali)
  # Bundle chosen lambda and MSE
  #print(mean((ridge.pred - y_vali)^2))
  MSE_by_lamb = data.frame("Lambda" = lambda_val, "MSE" = mean((ridge.pred - y_vali)^2))
  # print(MSE_by_lamb)
  return (MSE_by_lamb)
}

for (lamb in grid){
  lambda_records = rbind(lambda_records, errorCal.ridge(ridge.mod, x.valid, y.valid, lamb))
}

lambda_records
```

```
##           Lambda       MSE
## 1   1.000000e+10 10.574791
## 2   7.564633e+09 10.574791
## 3   5.722368e+09 10.574791
## 4   4.328761e+09 10.574791
## 5   3.274549e+09 10.574791
## 6   2.477076e+09 10.574791
## 7   1.873817e+09 10.574791
## 8   1.417474e+09 10.574791
## 9   1.072267e+09 10.574791
## 10  8.111308e+08 10.574791
## 11  6.135907e+08 10.574791
## 12  4.641589e+08 10.574791
## 13  3.511192e+08 10.574791
## 14  2.656088e+08 10.574791
## 15  2.009233e+08 10.574791
## 16  1.519911e+08 10.574790
## 17  1.149757e+08 10.574790
## 18  8.697490e+07 10.574790
## 19  6.579332e+07 10.574789
## 20  4.977024e+07 10.574788
## 21  3.764936e+07 10.574787
## 22  2.848036e+07 10.574786
## 23  2.154435e+07 10.574785
## 24  1.629751e+07 10.574782
## 25  1.232847e+07 10.574779
## 26  9.326033e+06 10.574775
## 27  7.054802e+06 10.574770
## 28  5.336699e+06 10.574763
## 29  4.037017e+06 10.574754
## 30  3.053856e+06 10.574743
## 31  2.310130e+06 10.574727
## 32  1.747528e+06 10.574706
## 33  1.321941e+06 10.574678
## 34  1.000000e+06 10.574642
## 35  7.564633e+05 10.574594
## 36  5.722368e+05 10.574530
## 37  4.328761e+05 10.574446
```

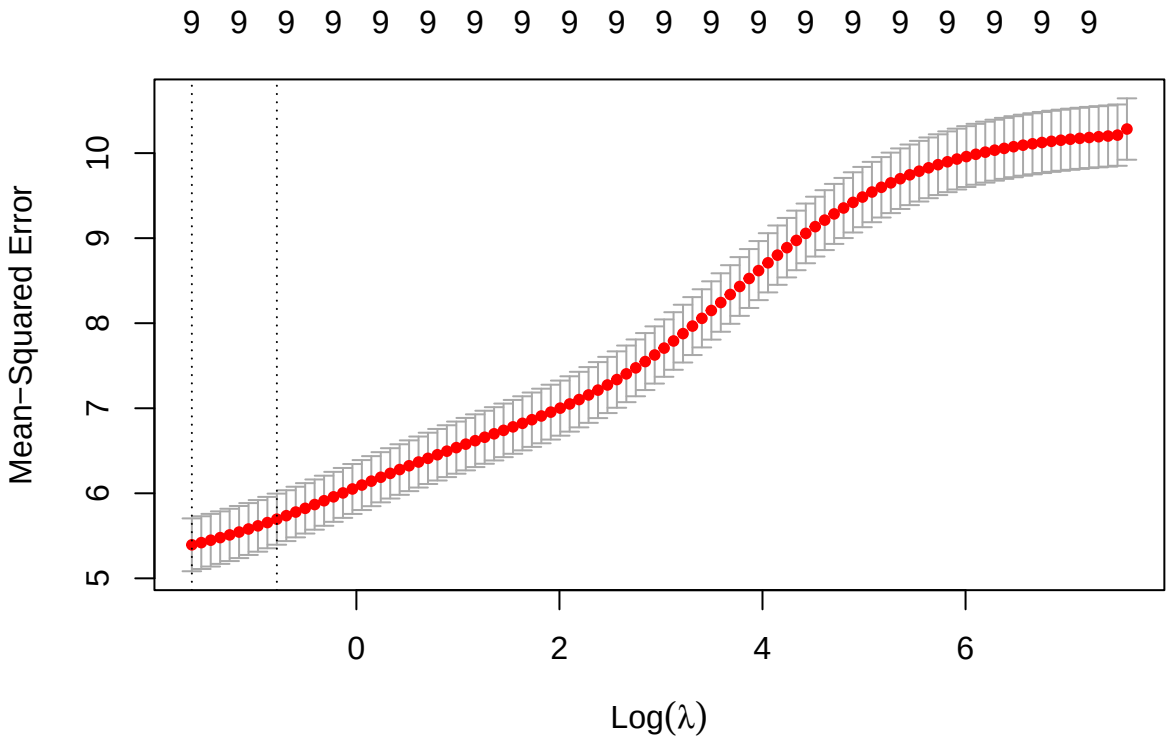
```
## 38  3.274549e+05 10.574335
## 39  2.477076e+05 10.574189
## 40  1.873817e+05 10.573995
## 41  1.417474e+05 10.573738
## 42  1.072267e+05 10.573399
## 43  8.111308e+04 10.572951
## 44  6.135907e+04 10.572359
## 45  4.641589e+04 10.571576
## 46  3.511192e+04 10.570542
## 47  2.656088e+04 10.569175
## 48  2.009233e+04 10.567369
## 49  1.519911e+04 10.564985
## 50  1.149757e+04 10.561835
## 51  8.697490e+03 10.557678
## 52  6.579332e+03 10.552193
## 53  4.977024e+03 10.544960
## 54  3.764936e+03 10.535431
## 55  2.848036e+03 10.522888
## 56  2.154435e+03 10.506402
## 57  1.629751e+03 10.484775
## 58  1.232847e+03 10.456675
## 59  9.326033e+02 10.419896
## 60  7.054802e+02 10.372171
## 61  5.336699e+02 10.310601
## 62  4.037017e+02 10.231765
## 63  3.053856e+02 10.131794
## 64  2.310130e+02 10.006586
## 65  1.747528e+02  9.852223
## 66  1.321941e+02  9.665634
## 67  1.000000e+02  9.445528
## 68  7.564633e+01  9.193407
## 69  5.722368e+01  8.914486
## 70  4.328761e+01  8.617944
## 71  3.274549e+01  8.316157
## 72  2.477076e+01  8.022788
## 73  1.873817e+01  7.749891
## 74  1.417474e+01  7.505653
## 75  1.072267e+01  7.292882
## 76  8.111308e+00  7.109153
## 77  6.135907e+00  6.948143
## 78  4.641589e+00  6.801729
## 79  3.511192e+00  6.662580
## 80  2.656088e+00  6.524832
## 81  2.009233e+00  6.384325
## 82  1.519911e+00  6.238956
## 83  1.149757e+00  6.090252
## 84  8.697490e-01  5.940994
## 85  6.579332e-01  5.796657
## 86  4.977024e-01  5.661869
## 87  3.764936e-01  5.542009
## 88  2.848036e-01  5.439996
## 89  2.154435e-01  5.357309
## 90  1.629751e-01  5.293056
## 91  1.232847e-01  5.244394
```

```
## 92  9.326033e-02  5.207818
## 93  7.054802e-02  5.180038
## 94  5.336699e-02  5.158265
## 95  4.037017e-02  5.140631
## 96  3.053856e-02  5.125970
## 97  2.310130e-02  5.113804
## 98  1.747528e-02  5.103856
## 99  1.321941e-02  5.096080
## 100 1.000000e-02  5.090278
```

```r
# Plot the MSE
plot(x=log(lambda_records$Lambda), y=lambda_records$MSE, type="b")
```




```r
# Since it is hard to read the best lamb by eyes, I will use a build-in cv function
# to help choose the best lambda.
set.seed(1)
cv.out <- cv.glmnet(x.train, y.train, alpha = 0)
plot(cv.out)
```




```
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 0.1977713
```

```
# The best lambda is 0.1977713.

# Use the optimal lambda and calculate MSE
ridge.pred <- predict(ridge.mod, s = bestlam, newx = x.valid)
mean((ridge.pred - y.valid)^2)
```

```
## [1] 5.334234
```

```
# The MSE with the best lambda is 5.33.
```

Let R select a model over the 3133 data point, using its "stepwise" automatic model selection model.

```
# I am going to use forward stepwise selection.
regfit.fwd <- regsubsets(Rings ~ ., data = abalone.train, nvmax = 9, method = "forward")
regfit.fwd_sum <- summary(regfit.fwd)
regfit.fwd_sum
```

```
## Subset selection object
## Call: regsubsets.formula(Rings ~ ., data = abalone.train, nvmax = 9,
##     method = "forward")
## 9 Variables  (and intercept)
##                Forced in Forced out
## SexI               FALSE      FALSE
## SexM               FALSE      FALSE
## Length             FALSE      FALSE
## Diameter           FALSE      FALSE
## Height             FALSE      FALSE
## Whole_weight       FALSE      FALSE
## Shucked_weight     FALSE      FALSE
## Viscera_weight     FALSE      FALSE
## Shell_weight       FALSE      FALSE
## 1 subsets of each size up to 9
## Selection Algorithm: forward
##          SexI SexM Length Diameter Height Whole_weight Shucked_weight
## 1  ( 1 ) " "  " "  " "    " "      " "    " "          " "
## 2  ( 1 ) " "  " "  " "    " "      " "    " "          "*"
## 3  ( 1 ) " "  " "  " "    "*"      " "    " "          "*"
## 4  ( 1 ) "*"  " "  " "    "*"      " "    " "          "*"
## 5  ( 1 ) "*"  " "  " "    "*"      " "    "*"          "*"
## 6  ( 1 ) "*"  " "  " "    "*"      " "    "*"          "*"
## 7  ( 1 ) "*"  " "  " "    "*"      "*"    "*"          "*"
## 8  ( 1 ) "*"  " "  "*"    "*"      "*"    "*"          "*"
## 9  ( 1 ) "*"  "*"  "*"    "*"      "*"    "*"          "*"
##          Viscera_weight Shell_weight
## 1  ( 1 ) " "            "*"
## 2  ( 1 ) " "            "*"
## 3  ( 1 ) " "            "*"
## 4  ( 1 ) " "            "*"
## 5  ( 1 ) " "            "*"
## 6  ( 1 ) "*"            "*"
## 7  ( 1 ) "*"            "*"
## 8  ( 1 ) "*"            "*"
## 9  ( 1 ) "*"            "*"
```

```
par(mfrow = c(2, 2))
plot(regfit.fwd_sum$rss, xlab = "Number of Variables",
     ylab = "RSS", type = "l")
plot(regfit.fwd_sum$adjr2, xlab = "Number of Variables",
     ylab = "Adjusted RSq", type = "l")

max_adjr = which.max(regfit.fwd_sum$adjr2)
#identify the location of the maximum point of a vector based on adjusted r-squared
max_adjr
```

```
## [1] 7
```

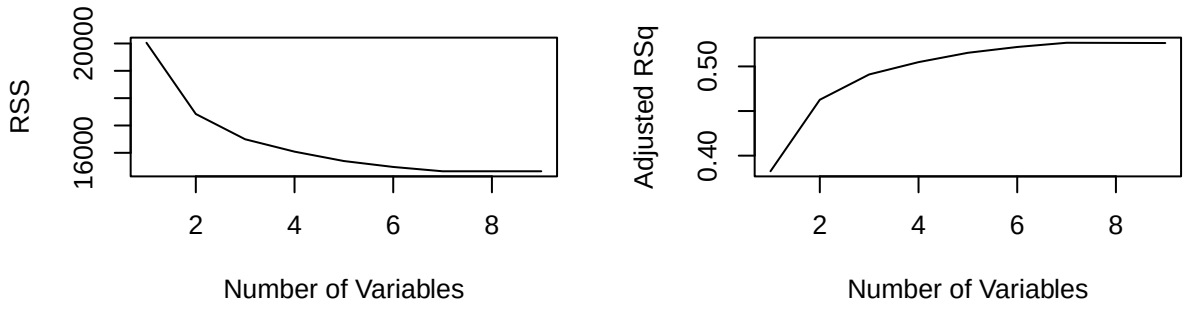
```
# According to these plots, the forward stepwise chooses the number of variables to be 7 because
# it is the first point where RSS is the lowest and adjusted R-squared is highest.

# Show intercept of the selected model
```

```
coefi <- coef(regfit.fwd, max_adjr)
coefi
```

```
##    (Intercept)           SexI       Diameter         Height   Whole_weight
##       4.067379      -0.895686      10.597527       9.610926       8.677487
## Shucked_weight Viscera_weight   Shell_weight
##     -19.507677     -10.052318       8.980017
```

```
# Test the select model with training observations
train.mat <- model.matrix(Rings ~ ., data = abalone.train)
# generate test dataset to be the same format with the model
pred.fwd_train <- train.mat[, names(coefi)] %*% coefi
MSE.fwd_train = mean((Rings.train - pred.fwd_train)^2) # Calculate MSE
MSE.fwd_train
```

```
## [1] 4.89148
```



Use an improved model

```
# According to correlation, these data are not fully linear (with correlation values
# of predictors and y are less than 0.6).
# Therefore, I believe tree-based methods could further improve the accuracy.

# I will use a boosting model with gbm because it is the best model among the tree-based family.
# Train the boosting model with shrinkage
```

```
boost.abalone <- gbm(Rings ~ ., data = abalone.train, distribution = "gaussian",
                     n.trees = 5000, interaction.depth = 4, shrinkage = 0.01,
                     verbose = F)
# Shrinkage is the learning rate parameter and this model depends on slow learning concept.


yhat.boost_train <- predict(boost.abalone,
                       newdata = abalone.train, n.trees = 5000)
MSE.boost_train <- mean((yhat.boost_train - Rings.train)^2) # MSE of training data
MSE.boost_train
```

```
## [1] 2.628041
```

Compare and summarize models' performance 7 models in total

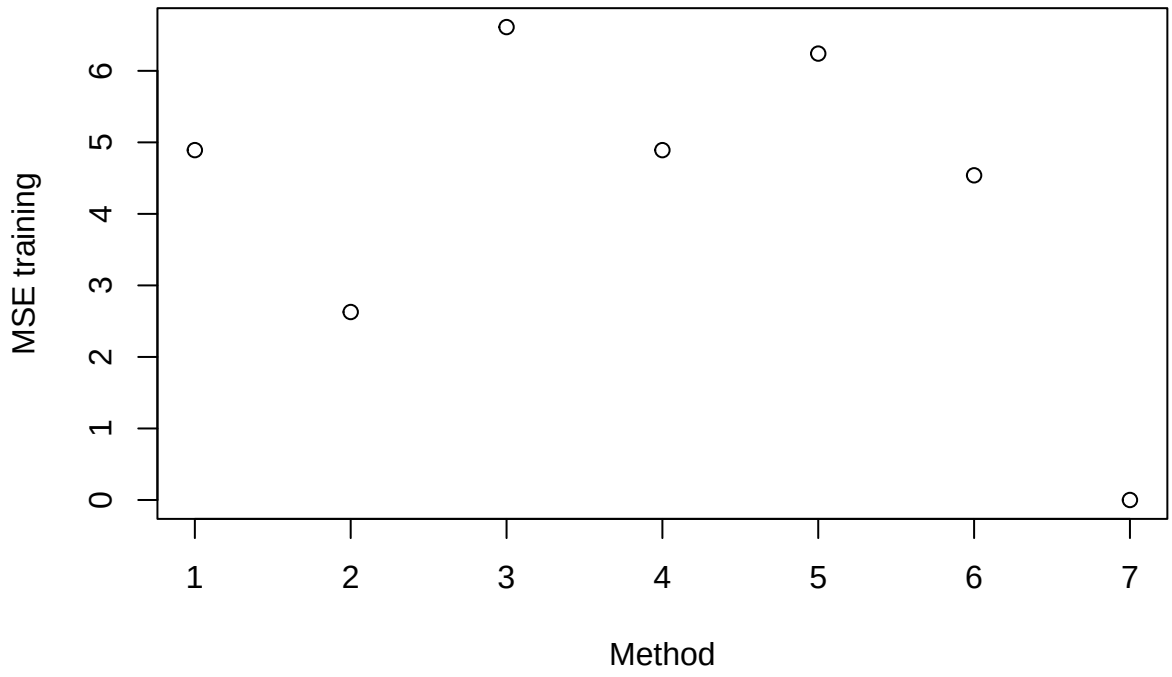
```
df.sum_acc = data.frame("method"=as.factor(0), "MSE_train"=as.numeric(0),
                        "MSE_test"=as.numeric(0))

# 1. lm1
# Calculate training errors (3133 observations)
pred.lm_fit1 <- predict(lm.fit1)
MSE.lm_fit1_train <- mean((Rings.train - pred.lm_fit1)^2)
# Calculate test errors (1044 observations)
pred.lm_fit1 <- predict(lm.fit1, newdata = abalone.test)
MSE.lm_fit1_test <- mean((Rings.test - pred.lm_fit1)^2)

# Store the MSE
df.sum_acc = data.frame("method"="lm.fit1", "MSE_train"=MSE.lm_fit1_train,
                        "MSE_test"=MSE.lm_fit1_test)

# 2. lm2
# Calculate training errors (3133 observations)
pred.lm_fit2 <- predict(lm.fit2)
MSE.lm_fit2_train <- mean((Rings.train - pred.lm_fit2)^2)
# Calculate test errors (1044 observations)
pred.lm_fit2 <- predict(lm.fit2, newdata = abalone.test)
MSE.lm_fit2_test <- mean((Rings.test - pred.lm_fit2)^2)

# Store the MSE
df.sum_acc[nrow(df.sum_acc) + 1,] <- c("lm.fit2", MSE.lm_fit2_train, MSE.lm_fit2_test)

# 3. q5
# Calculate training errors (3133 observations)
pred.q5 <- predict(fit.q5)
MSE.q5_train <- mean((Rings.train - pred.q5)^2)
# Calculate test errors (1044 observations)
pred.q5 <- predict(fit.q5, newdata = abalone.test)
MSE.q5_test <- mean((Rings.test - pred.q5)^2)

# Store the MSE
df.sum_acc[nrow(df.sum_acc) + 1,] <- c("q5", MSE.q5_train, MSE.q5_test)
```

```r
# 4. q8
# Calculate training errors (3133 observations)
pred.q8 <- predict(fit.q8)
MSE.q8_train <- mean((Rings.train - pred.q8)^2)
# Calculate test errors (1044 observations)
pred.q8 <- predict(fit.q8, newdata = abalone.test)
MSE.q8_test <- mean((Rings.test - pred.q8)^2)

# Store the MSE
df.sum_acc[nrow(df.sum_acc) + 1,] <- c("q8", MSE.q8_train, MSE.q8_test)


# 5. Ridge regression
# I will not calculate the training errors of ridge regression because
# this model is trained on 2349 observations, but other model are used 3133 observations.
# Comparing performance on training data with this model is not to compare apple to apple.

# Calculate test errors (1044 observations)
test.mat <- model.matrix(Rings ~ ., data = abalone.test)
ridge.pred_test <- predict(ridge.mod, s = bestlam, newx = test.mat[, -1])
MSE.ridge_test = mean((ridge.pred_test - Rings.test)^2)
# Use the optimal lambda and calculate MSE

# Store the MSE
df.sum_acc[nrow(df.sum_acc) + 1,] <- c("ridge", 0, MSE.ridge_test)
# Note: I put the MSE of training the ridge model as a dummy. You can ignore this value.

# 6. Forward stepwise model
# Test the select model with test observations
# Generate test dataset to be the same format with the model
pred.fwd_test <- test.mat[, names(coefi)] %*% coefi
MSE.fwd_test = mean((Rings.test - pred.fwd_test)^2) # Calculate MSE

# Store the MSE
df.sum_acc[nrow(df.sum_acc) + 1,] <- c("fwd_stepwise", MSE.fwd_train, MSE.fwd_test)
# Note: I already calculated MSE.fwd_train in g)


# 7. GBM
yhat.boost_test <- predict(boost.abalone,
                   newdata = abalone.test, n.trees = 5000)
MSE.boost_test <- mean((yhat.boost_test - Rings.test)^2) # This is the best performance so far.

# Store the MSE
df.sum_acc[nrow(df.sum_acc) + 1,] <- c("GBM", MSE.boost_train, MSE.boost_test)
# Note: I already calculated MSE.boost_train in h)

df.sum_acc
```

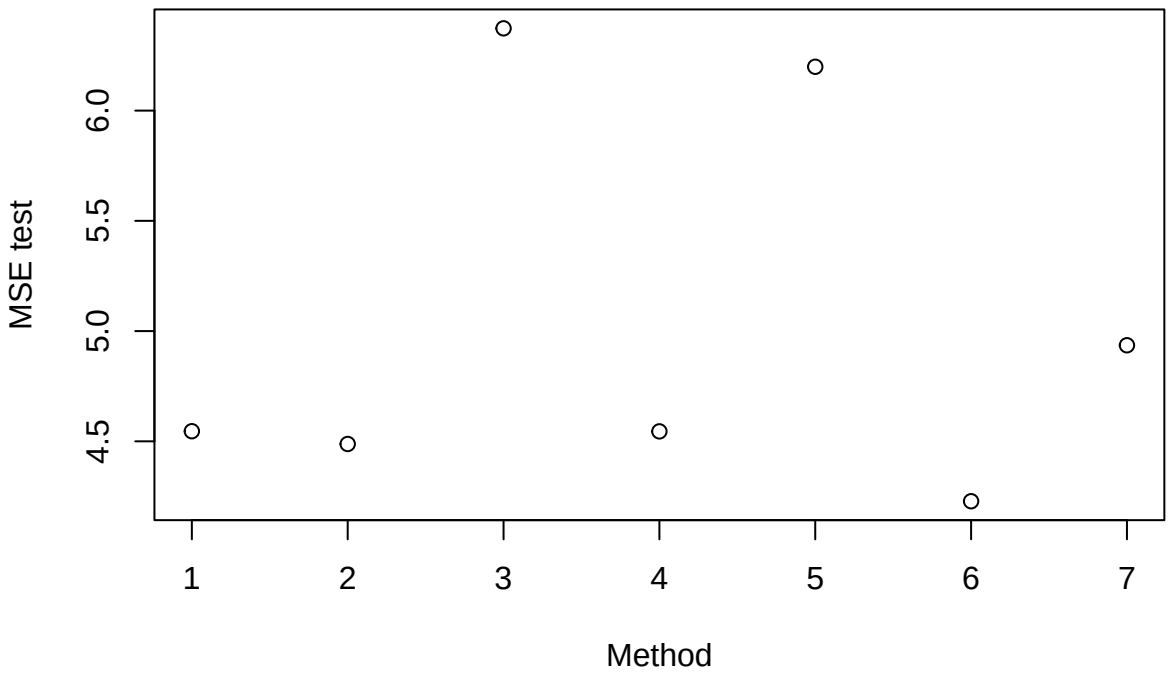
```
##         method        MSE_train         MSE_test
## 1      lm.fit1 6.61181075430578 6.37301094316447
## 2      lm.fit2 4.89142665686238 4.54516011352482
## 3           q5 6.24122391266507 6.19882914275479
```

```
## 4           q8 4.53915991709535  4.2283466968326
## 5        ridge                0 4.93572065616582
## 6 fwd_stepwise 4.89147971457224 4.54593950717788
## 7          GBM  2.6280406968077  4.4879879169426
```

```
df.sum_acc$method = as.factor(df.sum_acc$method)
plot(df.sum_acc$method, df.sum_acc$MSE_train, xlab = "Method", ylab = "MSE training")
```




```
plot(df.sum_acc$method, df.sum_acc$MSE_test, xlab = "Method", ylab = "MSE test") # plot
```




```
# According to the plot GBM performs best in MSE training but q8 has the lowest in MSE test.
```