Predicting_House_Prices_Using_Linear_Regression

Laura Cline

02/08/2021

Libraries

`MASS` library is a very large collection of datasets and functions. `ISLR` includes datasets from "Intro to Statistical Learning"

```
library(MASS)
library(ISLR)
set.seed(88)
```

```
write.csv(Boston, "boston.csv")
```

Simple Linear Regression

```
names(Boston)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

```
?Boston
```

We will start by using the `lm()` function to fit a simple linear regression model, with `medv` (median value of owner-occupied homes in $1000s) as the response and `lstat` (lower status of the population as percent) as the predictor. The basic syntax is `lm(y~x, data)`, where `y` is the response, `x` is the predictor, and `data` is the dataset in which these two variables are kept.

For more detailed information, we use `summary(lm.fit)`. This gives us the p-values and standard errors for the coefficients as well as the R^2 statistic and F-statistic for the model.

```
lm.fit = lm(medv ~ lstat, data = Boston)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.168  -3.990  -1.318   2.034  24.500
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.55384    0.56263   61.41   <2e-16 ***
## lstat       -0.95005    0.03873  -24.53   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

We can use the `names()` function in order to find out what other pieces of information are stored in `lm.fit`.

```
names(lm.fit)
```

```
##  [1] "coefficients" "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"       "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
coef(lm.fit)
```

```
## (Intercept)       lstat
##  34.5538409  -0.9500494
```

The `predict()` function can be used to predict confidence intervals and prediction intervals for the prediction of `medv` for a given value of `lstat`.

```
predict(lm.fit, data.frame(lstat=c(5, 10, 15)), interval="confidence")
```

```
##        fit      lwr      upr
## 1 29.80359 29.00741 30.59978
## 2 25.05335 24.47413 25.63256
## 3 20.30310 19.73159 20.87461
```

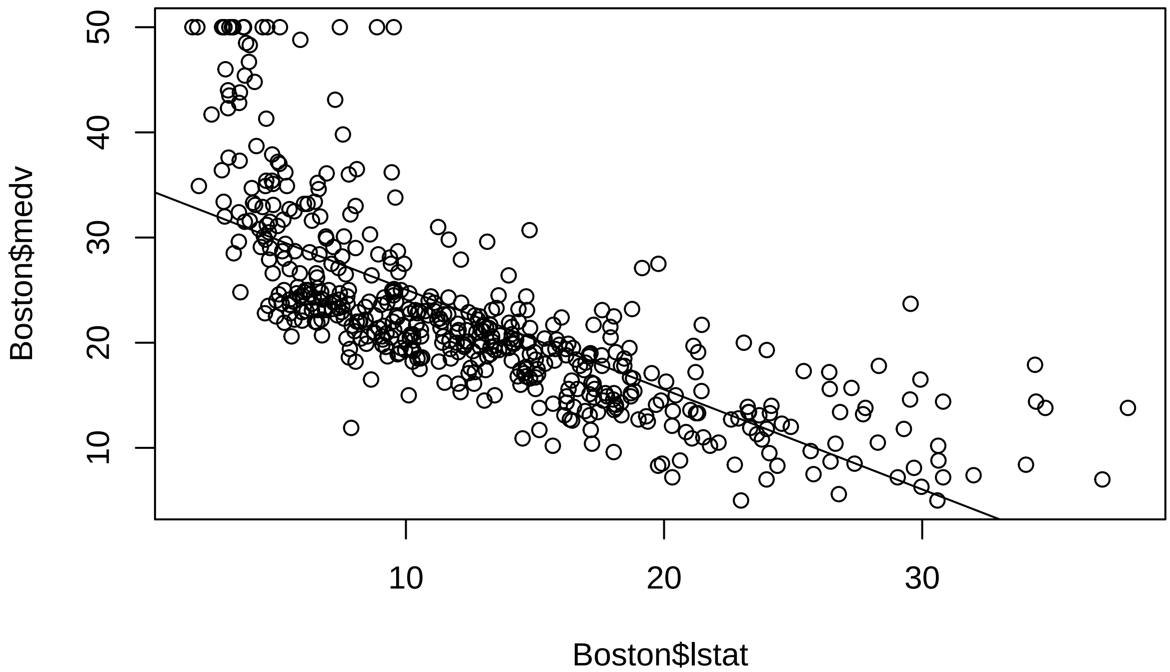
```
predict(lm.fit, data.frame(lstat=c(5,10,15)), interval = "prediction")
```

```
##        fit       lwr      upr
## 1 29.80359 17.565675 42.04151
## 2 25.05335 12.827626 37.27907
## 3 20.30310  8.077742 32.52846
```

For instance, the 95% confidence interval associated with a `lstat` value of 10 is (24.47, 25.63) and the 95% prediction interval is (12.82, 37.28). As expected, the confidence and prediction intervals are centered around the same point (a predicted value of 25.05 for `medv` when `lstat` equals 10), but the latter is substantially wider.

We will now plot `medv` and `lstat` along with the least squares regression line using the `plot()` and `abline()` functions.

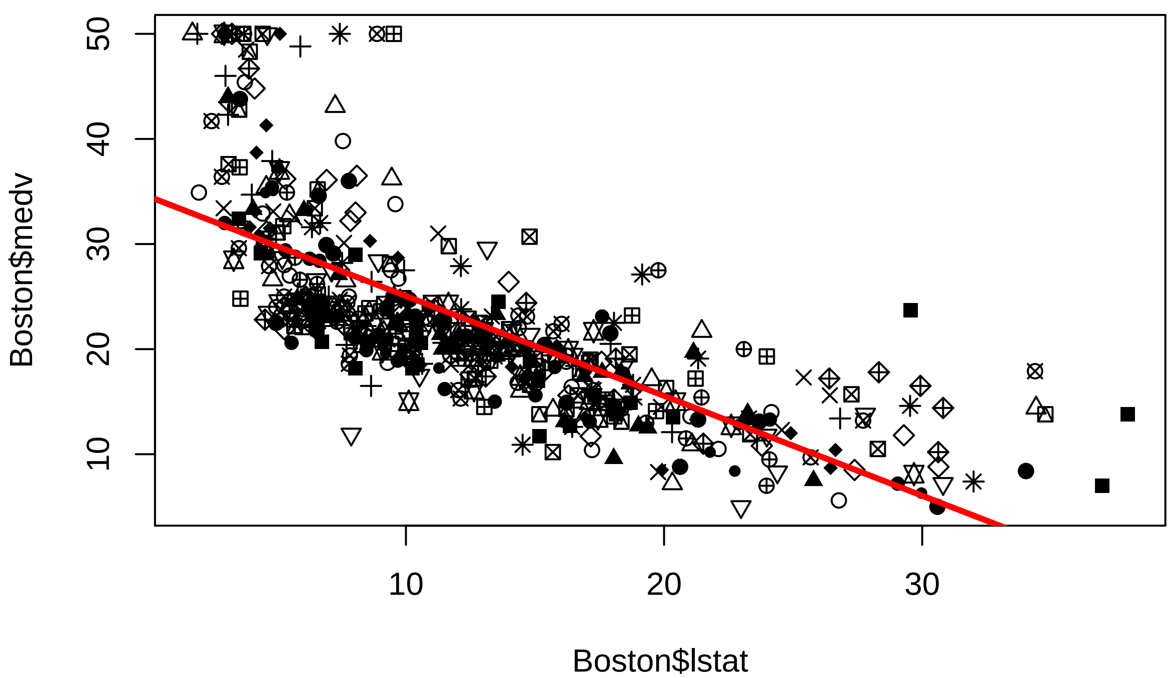
```
plot(Boston$lstat, Boston$medv)
abline(lm.fit)
```



There is some evidence for non-linearity in the relationship between `lstat` and `medv`.

The `abline` function can be used to draw any line, not just the least squares regression line. To draw a line with an intercept `a` and slope `b`, we type `abline(a,b)`. Below we experiment with some additional setttings for plotting lines and points. The `lwd=3` command causes the width of the regression line to be increased by a factor of 3; this works for the `plot()` and `lines()` functions also. We can also use the `pch` option to create different plotting symbols.

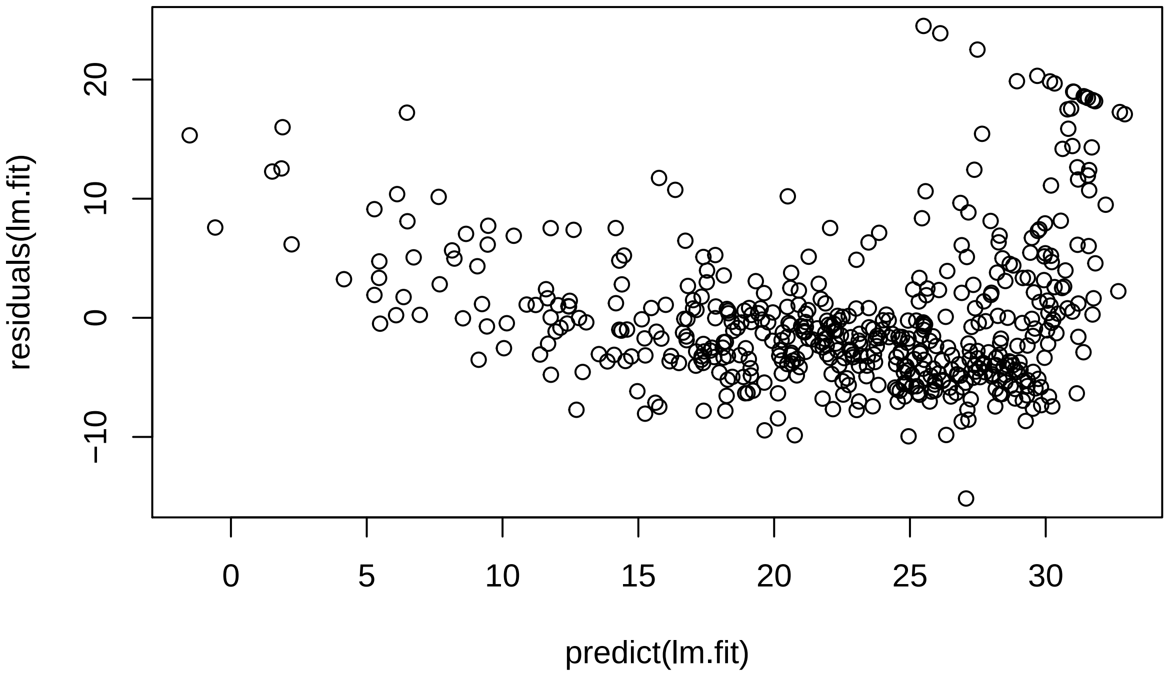
```
plot(Boston$lstat, Boston$medv, pch=1:20)
abline(lm.fit, lwd=3, col="red")
```



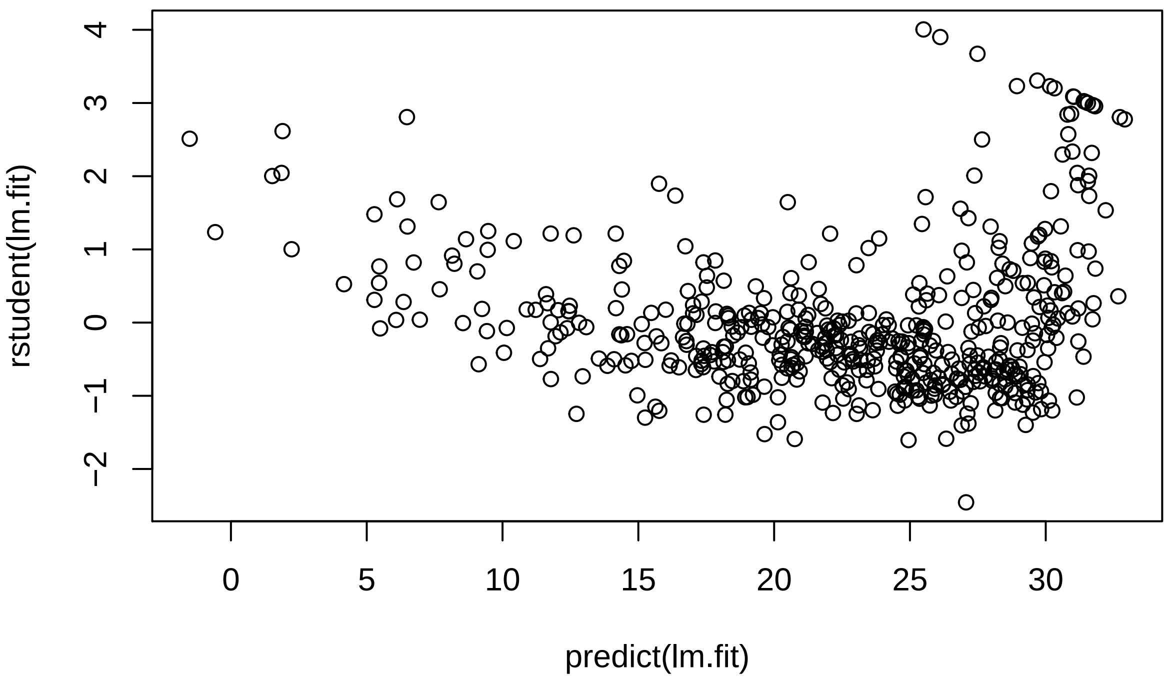
Alternatively, we can compute the residuals from the linear regression fit using the `residuals()` function.

The function `rstudent()` will return the studentized residuals, and we can use this function to plot the residuals against the fitted values.

```
plot(predict(lm.fit), residuals(lm.fit))
```



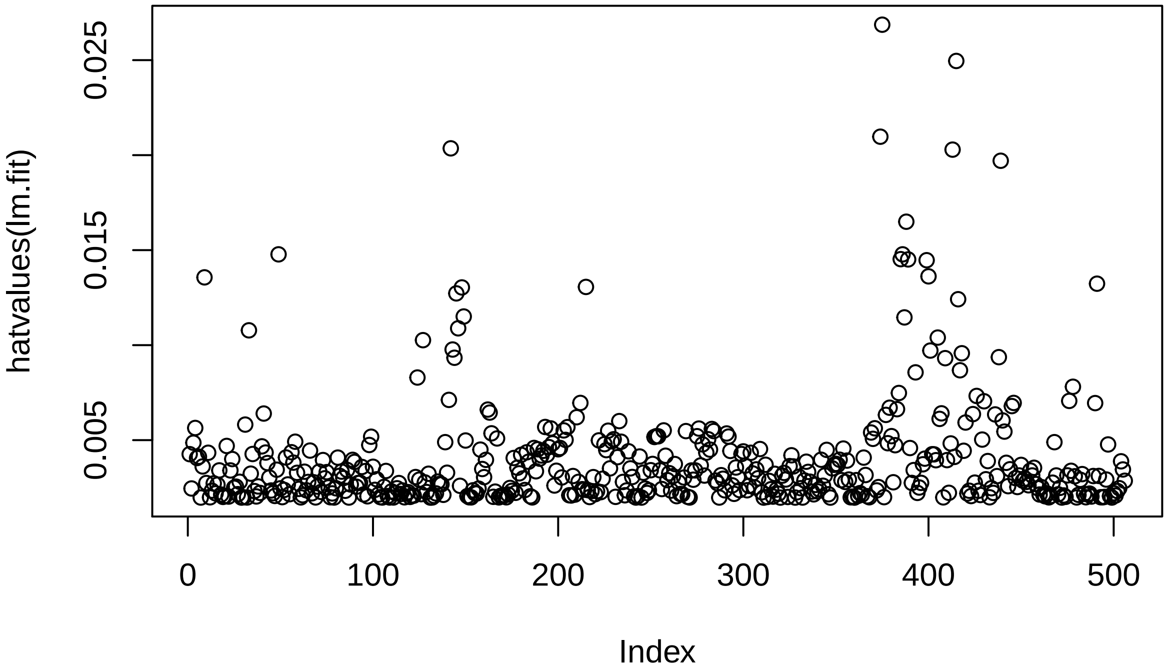

```
plot(predict(lm.fit), rstudent(lm.fit))
```



On the basis of residuals plots, there is some evidence of non-linearity. Leverage statistics can be computed for any number of predictors using the `hatvalues()` function.

The `whuch.max()` function identifies the index of the largest element of a vector. In this case, it tells us which observation has the largest leverage statistic.

```r
plot(hatvalues(lm.fit))
```




```r
which.max(hatvalues(lm.fit))
```

```
## 375
## 375
```

Multiple Linear Regression

In order to fit a multiple linear regression model using least squares, we again use the `lm()` function. The syntax `lm(y~x1+x2+x3)` is used to fit a model with three predictors, `x1`, `x2`, and `x3`. The `summary()` function now outputs the regression coefficients for all the predictors.

```r
lm.fit = lm(medv ~ lstat+age, data=Boston)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ lstat + age, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.981  -3.978  -1.283   1.968  23.158
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 33.22276    0.73085  45.458  < 2e-16 ***
## lstat       -1.03207    0.04819 -21.416  < 2e-16 ***
## age          0.03454    0.01223   2.826  0.00491 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.173 on 503 degrees of freedom
```

```
## Multiple R-squared:  0.5513, Adjusted R-squared:  0.5495
## F-statistic:   309 on 2 and 503 DF,  p-value: < 2.2e-16
```

The Boston dataset has 13 predictors. We can include all of them in the regression using the following shorthand:

```
lm.fit = lm(medv~., data = Boston)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ ., data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.595  -2.730  -0.518   1.777  26.199
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
## crim        -1.080e-01  3.286e-02  -3.287 0.001087 **
## zn           4.642e-02  1.373e-02   3.382 0.000778 ***
## indus        2.056e-02  6.150e-02   0.334 0.738288
## chas         2.687e+00  8.616e-01   3.118 0.001925 **
## nox         -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
## rm           3.810e+00  4.179e-01   9.116  < 2e-16 ***
## age          6.922e-04  1.321e-02   0.052 0.958229
## dis         -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
## rad          3.060e-01  6.635e-02   4.613 5.07e-06 ***
## tax         -1.233e-02  3.760e-03  -3.280 0.001112 **
## ptratio     -9.527e-01  1.308e-01  -7.283 1.31e-12 ***
## black        9.312e-03  2.686e-03   3.467 0.000573 ***
## lstat       -5.248e-01  5.072e-02 -10.347  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.745 on 492 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7338
## F-statistic: 108.1 on 13 and 492 DF,  p-value: < 2.2e-16
```

We can access the individual components of a summary object by name (type `?summary.lm` to see what is available). Hence `summary(lm.fit)$r.sq` gives us the R^2, and `summary(lm.fit)$sigma` gives us the RSE. The `vif()` function, part of the `car` package, can be used to compute variance inflation factors. Most VIFs are low to moderate for this data.

```
library(car)
```

```
## Loading required package: carData
```

```
vif(lm.fit)
```

```
##     crim       zn    indus     chas      nox       rm      age      dis
## 1.792192 2.298758 3.991596 1.073995 4.393720 1.933744 3.100826 3.955945
##      rad      tax  ptratio    black    lstat
## 7.484496 9.008554 1.799084 1.348521 2.941491
```

What id we would like to do regression using all of the variables but one For example, if we wanted to exclude `age`.

```r
lm.fit1 = lm(medv~.-age, data=Boston)
summary(lm.fit1)
```

```
##
## Call:
## lm(formula = medv ~ . - age, data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.6054  -2.7313  -0.5188   1.7601  26.2243
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  36.436927   5.080119   7.172 2.72e-12 ***
## crim         -0.108006   0.032832  -3.290 0.001075 **
## zn            0.046334   0.013613   3.404 0.000719 ***
## indus         0.020562   0.061433   0.335 0.737989
## chas          2.689026   0.859598   3.128 0.001863 **
## nox         -17.713540   3.679308  -4.814 1.97e-06 ***
## rm            3.814394   0.408480   9.338  < 2e-16 ***
## dis          -1.478612   0.190611  -7.757 5.03e-14 ***
## rad           0.305786   0.066089   4.627 4.75e-06 ***
## tax          -0.012329   0.003755  -3.283 0.001099 **
## ptratio      -0.952211   0.130294  -7.308 1.10e-12 ***
## black         0.009321   0.002678   3.481 0.000544 ***
## lstat        -0.523852   0.047625 -10.999  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.74 on 493 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7343
## F-statistic: 117.3 on 12 and 493 DF,  p-value: < 2.2e-16
```

Alternatively, the `update()` function can be used.

```r
lm.fit1 = update(lm.fit, ~.-age)
```

Interaction Terms

It's easy to include interaction terms in a linear model using the `lm()` function. The syntax `lstat:black` tells R to include an interaction term between `lstat` and `black`. The syntax `lstat*age` simultaneously includes `lstat`, `age` and the interaction term `lstat*age` as predictors; it is shorthand for `lstat+age+lstat:age`.

```r
summary(lm(medv~lstat*age, data=Boston))
```

```
##
## Call:
## lm(formula = medv ~ lstat * age, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.806  -4.045  -1.333   2.085  27.552
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept) 36.0885359  1.4698355  24.553  < 2e-16 ***
## lstat       -1.3921168  0.1674555  -8.313 8.78e-16 ***
## age         -0.0007209  0.0198792  -0.036   0.9711
## lstat:age    0.0041560  0.0018518   2.244   0.0252 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.149 on 502 degrees of freedom
## Multiple R-squared:  0.5557, Adjusted R-squared:  0.5531
## F-statistic: 209.3 on 3 and 502 DF,  p-value: < 2.2e-16
```

Non-Linear Transformations of the Predictors

The `lm()` function can also accommodate non-linear transformations of the predictors. For instance, given a predictor X, we can create a predictor X^2 using `I(X^2)`. The function `I()` is needed since the ^ has a special meaning in a formula; wrapping as we do allows the standard usage in R, which is to raise `X` to the power of 2. We now perform a regression of `medv` onto `lstat` and `lstat^2`.

```
lm.fit2 = lm(medv~lstat + I(lstat^2), data = Boston)
summary(lm.fit2)
```

```
##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2834  -3.8313  -0.5295   2.3095  25.4148
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.862007   0.872084   49.15   <2e-16 ***
## lstat       -2.332821   0.123803  -18.84   <2e-16 ***
## I(lstat^2)   0.043547   0.003745   11.63   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.524 on 503 degrees of freedom
## Multiple R-squared:  0.6407, Adjusted R-squared:  0.6393
## F-statistic: 448.5 on 2 and 503 DF,  p-value: < 2.2e-16
```

The near-zero p-value associated with the quadratic term suggests that it leads to an improved model. We use `anova()` function to further quantify the extent to which the quadratic fit is superior to the linear fit.

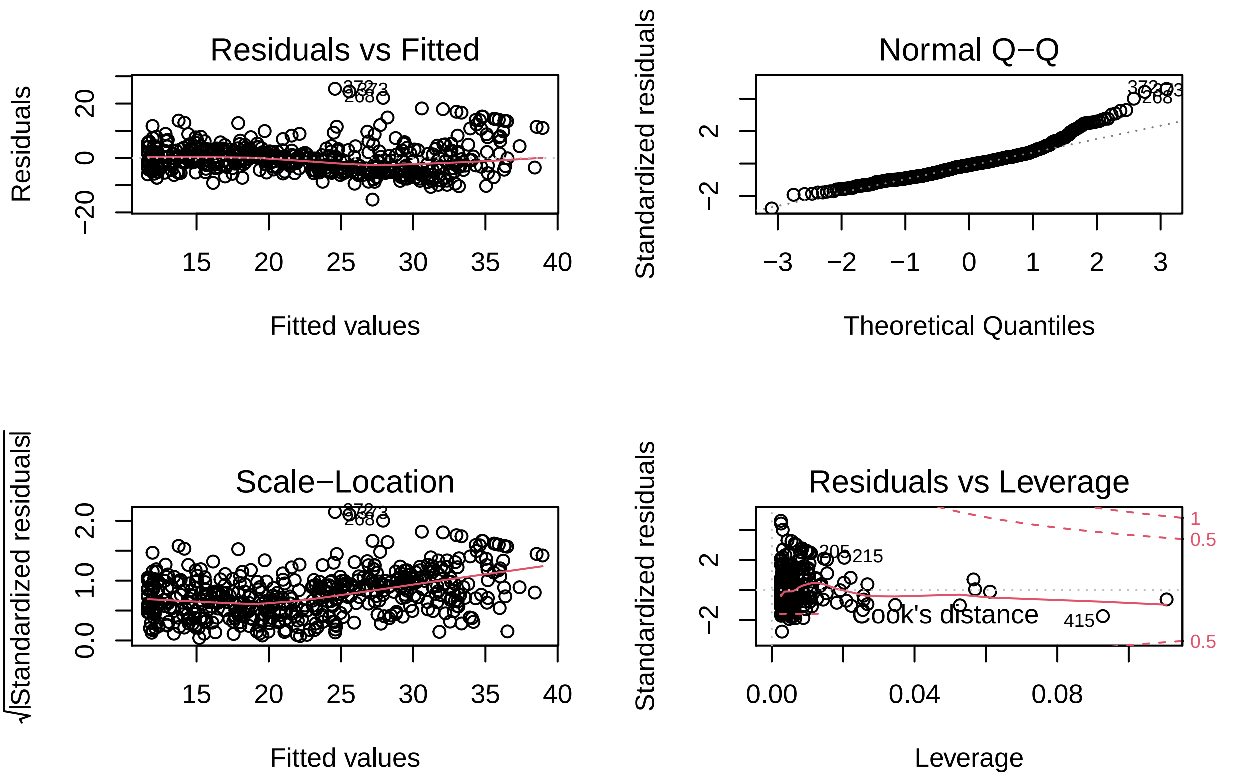
```
lm.fit = lm(medv~lstat, data = Boston)
anova(lm.fit, lm.fit2)
```

```
## Analysis of Variance Table
##
## Model 1: medv ~ lstat
## Model 2: medv ~ lstat + I(lstat^2)
##   Res.Df   RSS Df Sum of Sq     F    Pr(>F)
## 1    504 19472
## 2    503 15347  1    4125.1 135.2 < 2.2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here, Model 1 represents the linear submodel containing only one predictor `lstat`, while Model 2 corresponds to the larger quadratic model that has two predictors `lstat` and `lstat^2`. The `anova()` function performs a hypothesis test comparing the two models. The null hypothesis is that the two models fit the data equally well, and the alternative hypothesis is that the full model is superior. Here, the F-Statistic is 135.2 and the associated p-value is virtually zero. This provides clear evidence that the model containing the predictors `lstat` and `lstat^2` is far superior to the model that only contains the predictor `lstat`. This is not surprising, since earlier we saw evidence for non-linearity in the relationship between `medv` and `lstat`. If we type:

```
par(mfrow=c(2,2))
plot(lm.fit2)
```



then we see that when the `lstat^2` term is included in the model, there is a discernible pattern in the residuals.

In order to create a cubic fit, we include a predictor of the form `I(X^3)`. However, this approach can start to get cumbersome for higher-order polynomials. A better approach involves using the `poly()` function to create the polynomial within `lm()`. For example, the following command produces the fifth-order polynomial fit:

```
lm.fit5 = lm(medv~poly(lstat, 5), data=Boston)
summary(lm.fit5)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 5), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.5433  -3.1039  -0.7052   2.0844  27.1153
##
## Coefficients:
```

```
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        22.5328     0.2318  97.197  < 2e-16 ***
## poly(lstat, 5)1 -152.4595     5.2148 -29.236  < 2e-16 ***
## poly(lstat, 5)2   64.2272     5.2148  12.316  < 2e-16 ***
## poly(lstat, 5)3  -27.0511     5.2148  -5.187 3.10e-07 ***
## poly(lstat, 5)4   25.4517     5.2148   4.881 1.42e-06 ***
## poly(lstat, 5)5  -19.2524     5.2148  -3.692 0.000247 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.215 on 500 degrees of freedom
## Multiple R-squared:  0.6817, Adjusted R-squared:  0.6785
## F-statistic: 214.2 on 5 and 500 DF,  p-value: < 2.2e-16
```

This suggests that including additional polynomial terms, up to the fifth order, leads to an improvement in model fit! However, further investigation of the data reveals that no polynomial terms beyond fifth order have significant p-values in a regression fit. Of course, we are not restricted to polynomial transformations of the predictors. Here we try a log transformation:

```r
summary(lm(medv~log(rm), data = Boston))
```

```
##
## Call:
## lm(formula = medv ~ log(rm), data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.487  -2.875  -0.104   2.837  39.816
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -76.488      5.028  -15.21   <2e-16 ***
## log(rm)       54.055      2.739   19.73   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.915 on 504 degrees of freedom
## Multiple R-squared:  0.4358, Adjusted R-squared:  0.4347
## F-statistic: 389.3 on 1 and 504 DF,  p-value: < 2.2e-16
```

Predict Per Capita Crime Rate

```r
# Loop over each predictor and look for a statistically signficiant simple linear regression
crim = Boston[,1]
model_f_value = c()
model_p_Value = c()
univariable_beta_value = c()
possible_predictors = colnames(Boston)

for(pi in 1:length(possible_predictors)) {
  if(possible_predictors[pi] == 'crim') {next}
  x = Boston[,pi]
  m = lm(crim ~ x, data = Boston)
  s = summary(m)
```

```r
  model_f_value = c(model_f_value, s$fstatistic[1])
  model_p_Value = c(model_p_Value, anova(m)$'Pr(>F)'[1])
  univariable_beta_value = c(univariable_beta_value, coefficients(m)['x'])
  print(sprintf("%s %10.6f", possible_predictors[pi], coefficients(m)['x']))
}
```

```
## [1] "zn  -0.073935"
## [1] "indus   0.509776"
## [1] "chas  -1.892777"
## [1] "nox  31.248531"
## [1] "rm  -2.684051"
## [1] "age   0.107786"
## [1] "dis  -1.550902"
## [1] "rad   0.617911"
## [1] "tax   0.029742"
## [1] "ptratio   1.151983"
## [1] "black  -0.036280"
## [1] "lstat   0.548805"
## [1] "medv  -0.363160"
```

```r
# Let's look at each models F-statistics:

DF = data.frame(feature=colnames(Boston)[-1], f_values=model_f_value, p_values = model_p_Value)
DF[order(model_f_value),]
```

```
##    feature   f_values     p_values
## 3     chas   1.579364 2.094345e-01
## 1       zn  21.102782 5.506472e-06
## 5       rm  25.450204 6.346703e-07
## 10 ptratio  46.259453 2.942922e-11
## 6      age  71.619402 2.854869e-16
## 7      dis  84.887810 8.519949e-19
## 11   black  87.739763 2.487274e-19
## 13    medv  89.486115 1.173987e-19
## 2    indus  99.817037 1.450349e-21
## 4      nox 108.555329 3.751739e-23
## 12   lstat 132.035125 2.654277e-27
## 9      tax 259.190294 2.357127e-47
## 8      rad 323.935172 2.693844e-56
```

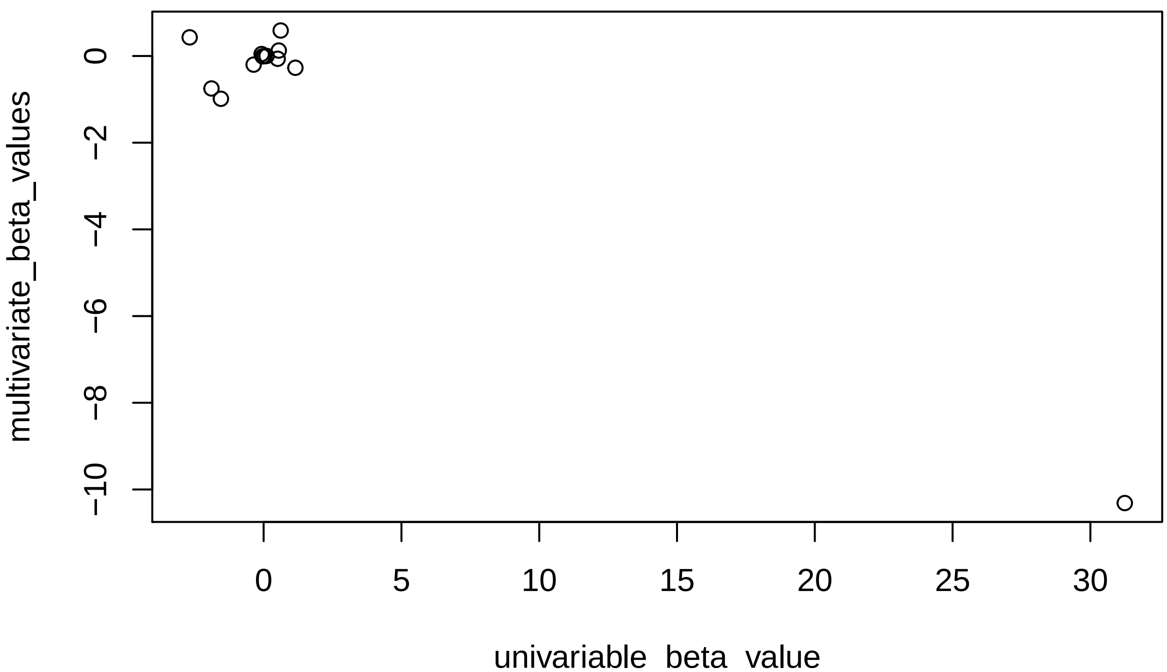
```r
# Consider a model too significant if the p-value < 0.01. Every model is signfiicant except one. This i

# Fit all of the predictors:
gm = lm(crim ~ ., data = Boston)
summary(gm)
```

```
##
## Call:
## lm(formula = crim ~ ., data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -9.924 -2.120 -0.353  1.019 75.051
##
## Coefficients:
```

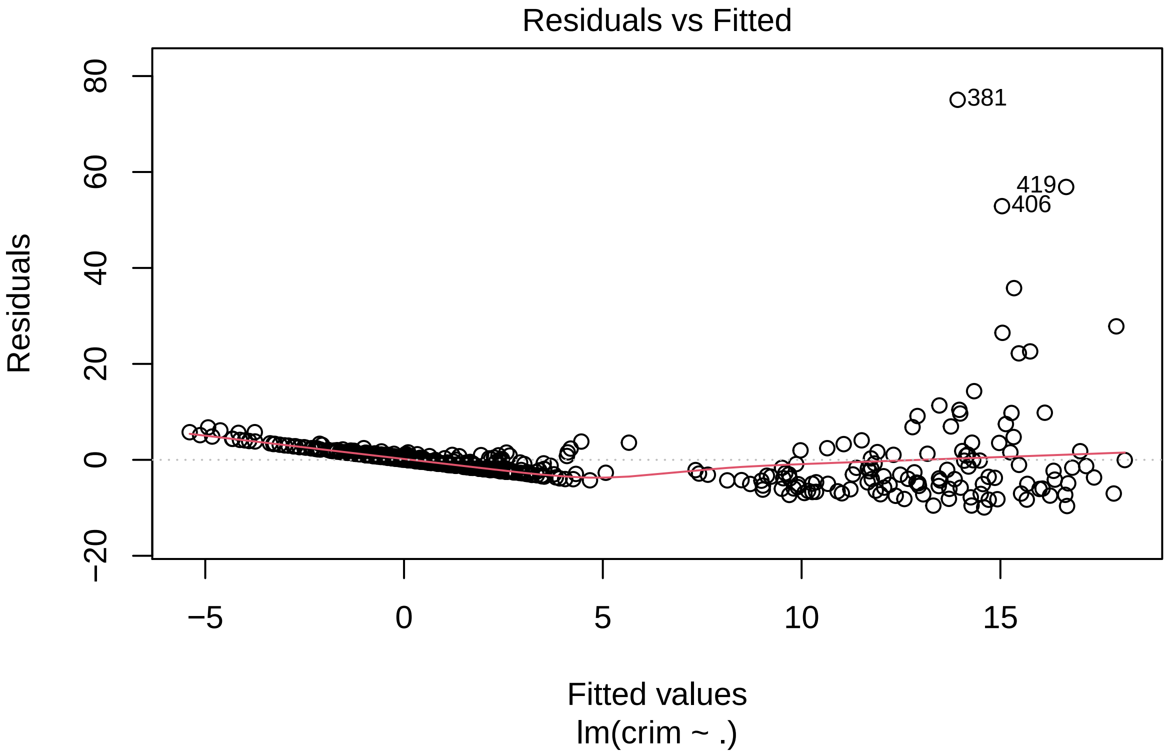
```
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)  17.033228   7.234903   2.354 0.018949 *
## zn            0.044855   0.018734   2.394 0.017025 *
## indus        -0.063855   0.083407  -0.766 0.444294
## chas         -0.749134   1.180147  -0.635 0.525867
## nox         -10.313535   5.275536  -1.955 0.051152 .
## rm            0.430131   0.612830   0.702 0.483089
## age           0.001452   0.017925   0.081 0.935488
## dis          -0.987176   0.281817  -3.503 0.000502 ***
## rad           0.588209   0.088049   6.680 6.46e-11 ***
## tax          -0.003780   0.005156  -0.733 0.463793
## ptratio      -0.271081   0.186450  -1.454 0.146611
## black        -0.007538   0.003673  -2.052 0.040702 *
## lstat         0.126211   0.075725   1.667 0.096208 .
## medv         -0.198887   0.060516  -3.287 0.001087 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.439 on 492 degrees of freedom
## Multiple R-squared:  0.454,  Adjusted R-squared:  0.4396
## F-statistic: 31.47 on 13 and 492 DF,  p-value: < 2.2e-16
```

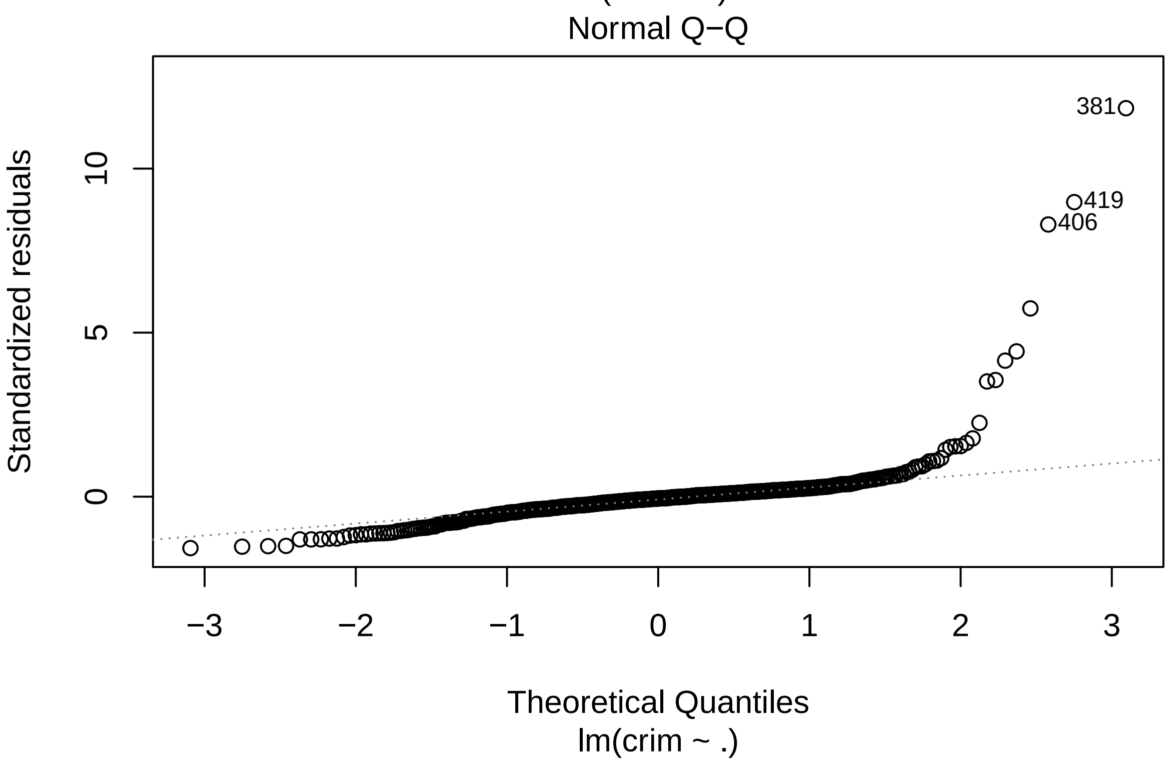
```
multivariate_beta_values = coefficients(gm)
multivariate_beta_values = multivariate_beta_values[possible_predictors[-1]]
# Order the coefficients in the same order as the univariate coefficients
```

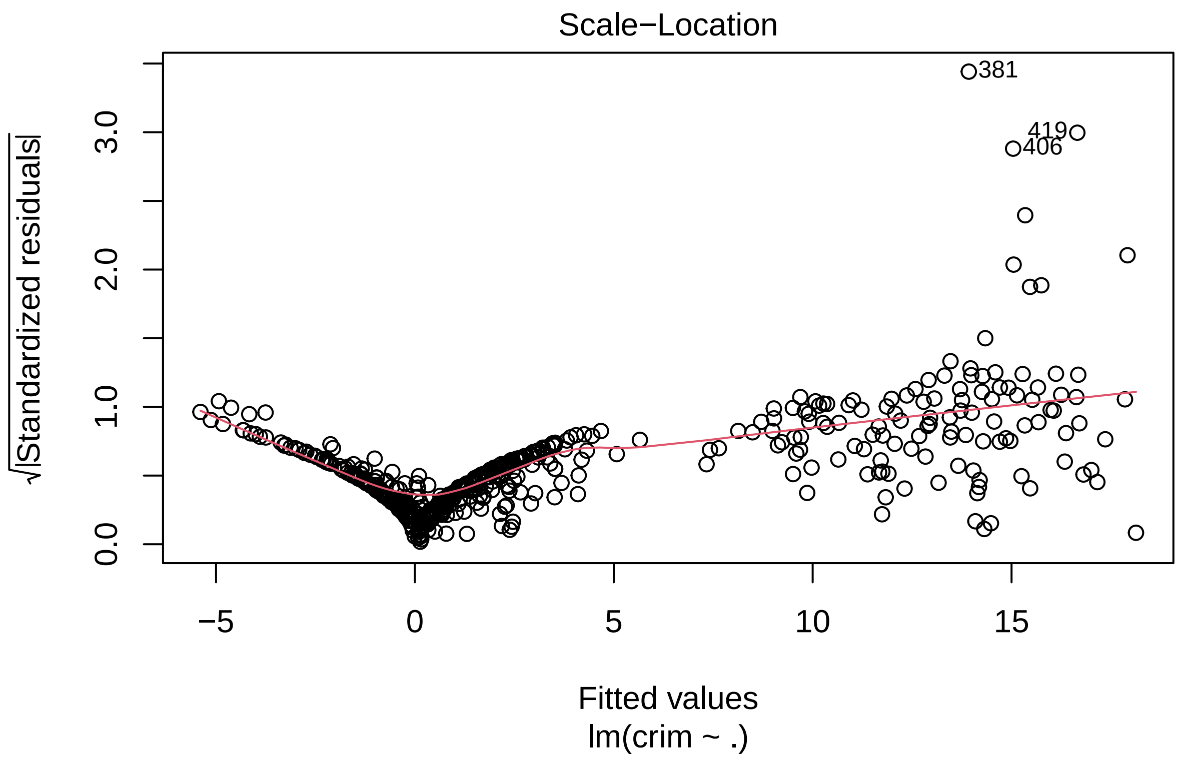
```
# Plot these 2 coefficients
plot(univariable_beta_value, multivariate_beta_values)
```

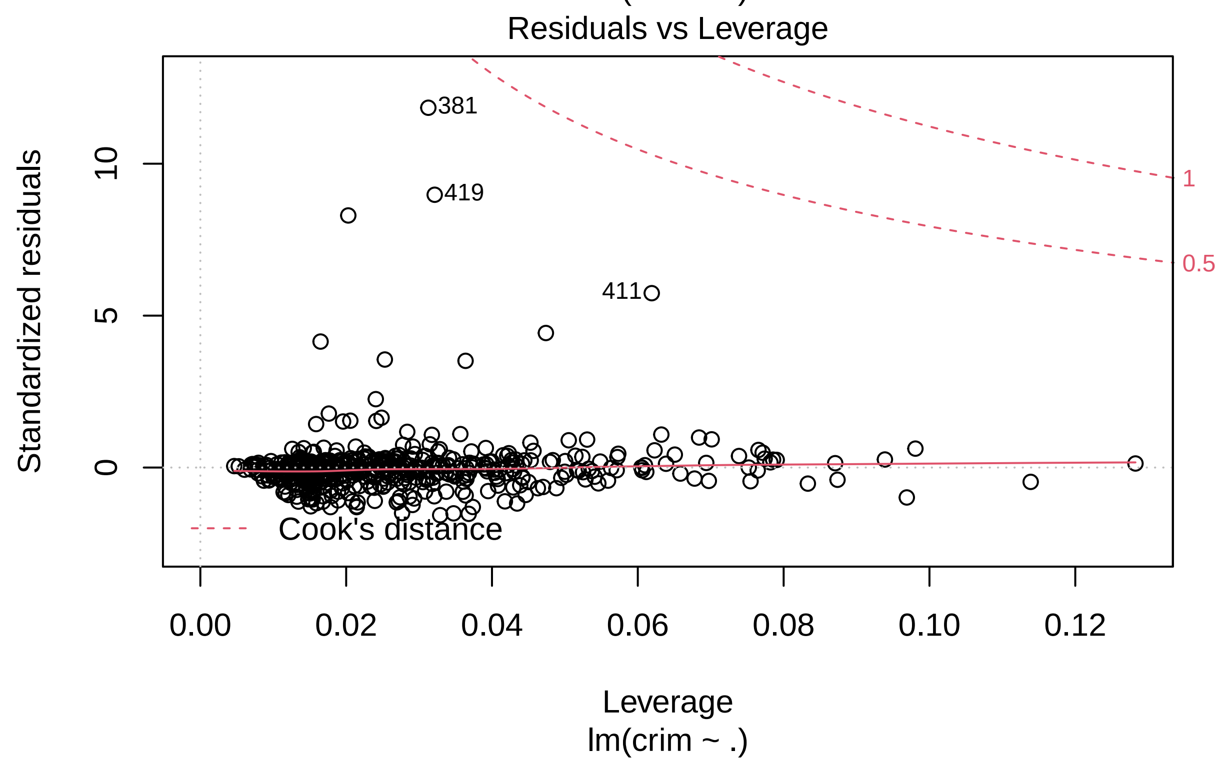



```
# Look for the possibility of including a non-linear term
plot(gm)
```


Residuals vs Fitted
Residuals
80
60
40
20
0
−20
381
419
406
−5
0
5
10
15
Fitted values
lm(crim ~ .)


Normal Q−Q
Standardized residuals
10
5
0
381
419
406
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(crim ~ .)





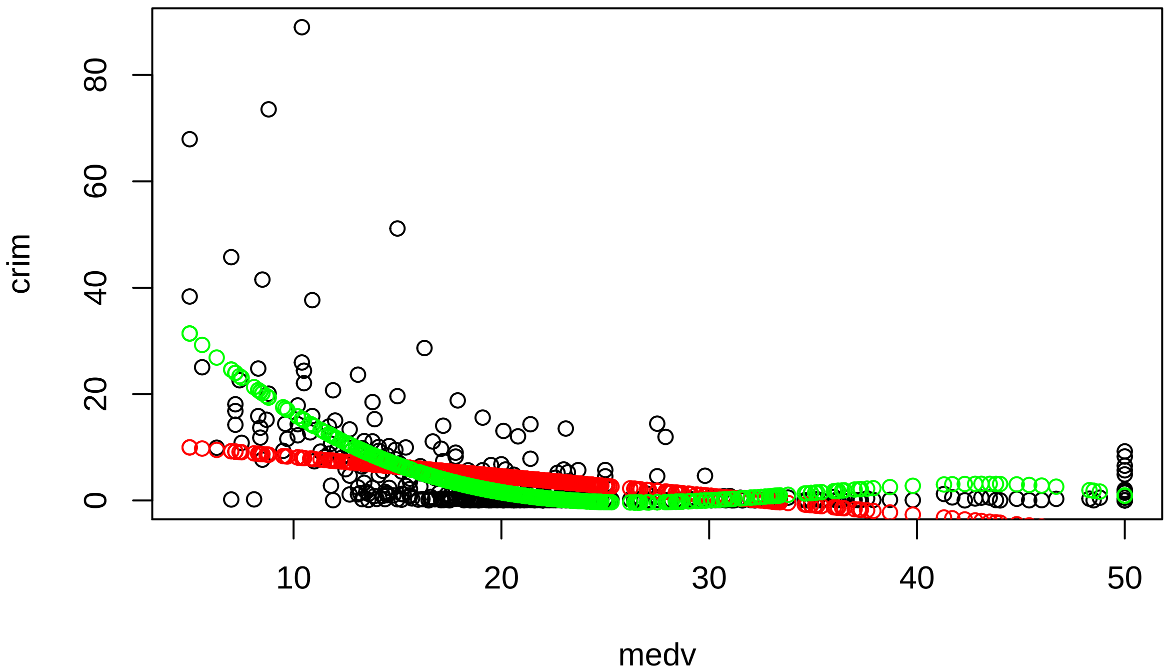

```
# See if we should fit a non-linear model of one variable (compare the ANOVA values):
crim = Boston[,1]
anova_f_value = c()
possible_predictors = colnames(Boston)

for (pi in 1:length(possible_predictors)) {
```

```r
  if(possible_predictors[pi] == 'crim'){next}
  x = Boston[,pi]
  if(possible_predictors[pi] == 'chas'){
    F = NA
  }else{
    m_1 = lm(crim ~ x)
    m_3 = lm(crim ~ poly(x,3))
    F = anova(m_1, m_3)$F[2]
  }
  anova_f_value = c(anova_f_value, F)
}
```

```r
DF = data.frame(feature=colnames(Boston)[-1], f_values=anova_f_value)
DF[order(anova_f_value),]
```

```
##    feature    f_values
## 11   black   0.4622222
## 12   lstat   3.3190437
## 8      rad   3.6732699
## 1       zn   4.8118205
## 5       rm   5.3088168
## 10 ptratio   8.4155300
## 9      tax  11.6400227
## 6      age  15.1400633
## 2    indus  31.9869602
## 4      nox  42.7581707
## 7      dis  46.4603654
## 13    medv 116.6340058
## 3     chas          NA
```

```r
# Let's look at how the non-linear model compared to the linear model
m_1 = lm(crim ~ medv, data=Boston)
m_3 = lm(crim ~ poly(medv,3), data = Boston)

plot(crim ~ medv, data=Boston)
lines(Boston$medv, predict(m_1), col='red', type='p')
lines(Boston$medv, predict(m_3), col='green', type='p')
```



crim
0
20
40
60
80
10
20
30
40
50
medv