

Multiple linear regression

Trevor Hastie and Robert Tibshirani

Here, I am adapting the lab associated with Chapter 3 of the textbook.

We load the `ISLR2` package, which includes the data sets associated with this book.

```
library(ISLR2)
```

The `ISLR2` library contains the Boston data set, which records `medv` (median house value) for 506 census tracts in Boston. We will seek to predict `medv` using 12 predictors such as `rmvar` (average number of rooms per house), `age` (proportion of owner-occupied units built prior to 1940) and `lstat` (percent of households with low socioeconomic status).

```
head(Boston)
```

```
##      crim  zn  indus  chas   nox    rm    age   dis   rad tax ptratio lstat medv
## 1 0.00632 18  2.31     0 0.538 6.575 65.2 4.0900    1 296  15.3  4.98 24.0
## 2 0.02731  0  7.07     0 0.469 6.421 78.9 4.9671    2 242  17.8  9.14 21.6
## 3 0.02729  0  7.07     0 0.469 7.185 61.1 4.9671    2 242  17.8  4.03 34.7
## 4 0.03237  0  2.18     0 0.458 6.998 45.8 6.0622    3 222  18.7  2.94 33.4
## 5 0.06905  0  2.18     0 0.458 7.147 54.2 6.0622    3 222  18.7  5.33 36.2
## 6 0.02985  0  2.18     0 0.458 6.430 58.7 6.0622    3 222  18.7  5.21 28.7
```

We should also investigate the variables therein and `attach` so that we don't have to use the `$` notation.

```
names(Boston)
```

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

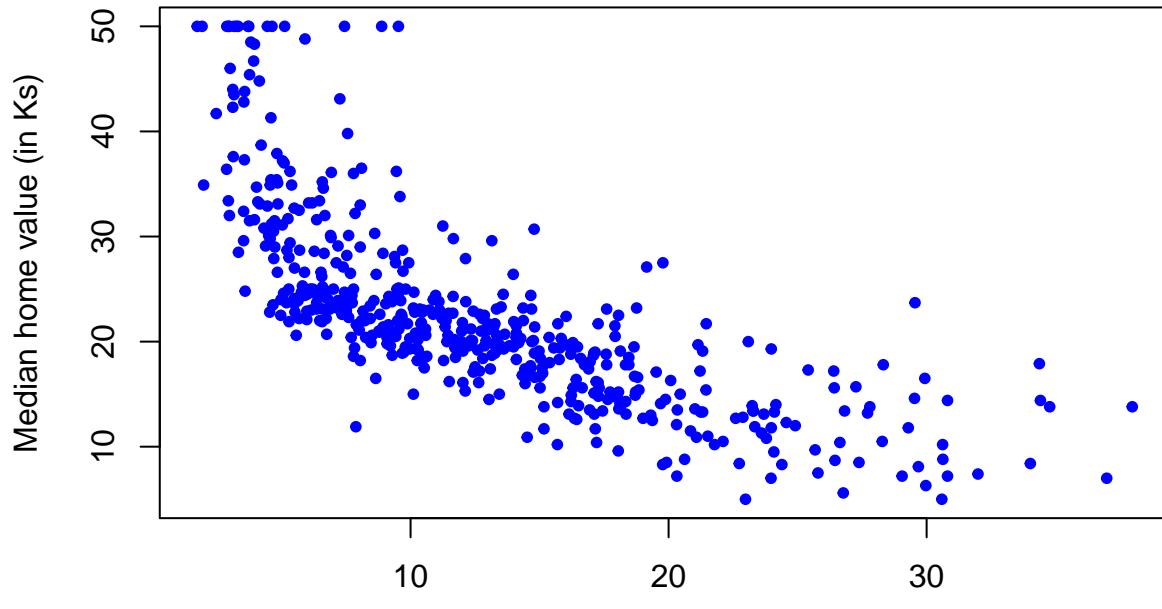
```
attach(Boston)
```

To find out more about the data set, we can type `?Boston`.

We will start by fitting a simple linear regression model, with `medv` as the response and `lstat` as the predictor. Here is the scatterplot.

```
plot(lstat, medv,
      main="Median value vs. lower status",
      xlab="Lower status in the population (%)",
      ylab="Median home value (in Ks)",
      pch=20, col="blue")
```

Median value vs. lower status



Lower status in the population (%)

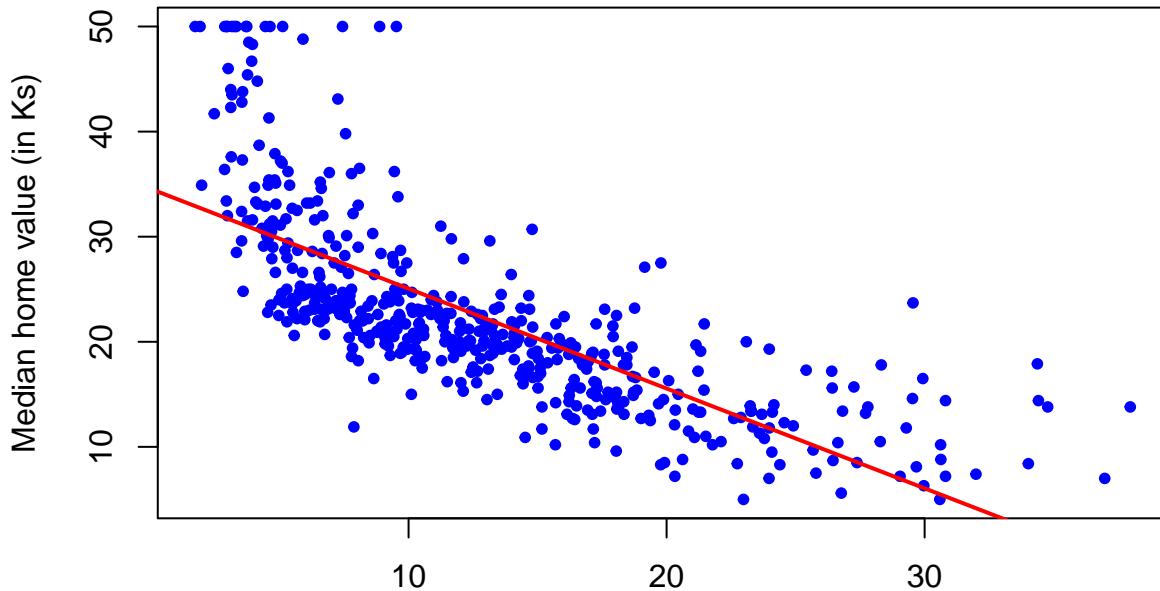
For the implementation of the linear model, the basic syntax is `lm(y ~ x, data)`, where `y` is the response, `x` is the predictor, and `data` is the data set in which these two variables are kept.

```
lm.fit <- lm(medv ~ lstat)
summary(lm.fit)

##
## Call:
## lm(formula = medv ~ lstat)
##
## 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

plot(lstat, medv,
      main="Median value vs. lower status",
      xlab="Lower status in the population (%)",
      ylab="Median home value (in Ks)",
      pch=20, col="blue")
abline(lm.fit, col="red", lwd=2)
```

Median value vs. lower status



Lower status in the population (%)

We can

use the `names()` function in order to find out what other pieces of information are stored in `lm.fit`. Although we can extract these quantities by name—e.g. `lm.fit$coefficients`—it is safer to use the extractor functions like `coef()` to access them.

```
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
```

In order to obtain a confidence interval for the coefficient estimates, we can use the `confint()` command.

```
confint(lm.fit)
```

```
##                  2.5 %      97.5 %  
## (Intercept) 33.448457 35.6592247  
## lstat        -1.026148 -0.8739505
```

The `predict()` function can be used to produce 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
```

How about the full array of confidence and prediction intervals?

```
newdata<-data.frame(lstat=seq(min(lstat),max(lstat),0.1))
```

```
conf<-predict(lm.fit,newdata,interval="confidence")  
conf
```

```
##          fit      lwr      upr  
## 1 32.91025550 31.917443200 33.9030678  
## 2 32.81525056 31.828800866 33.8017003  
## 3 32.72024563 31.740140752 33.7003505  
## 4 32.62524069 31.651462508 33.5990189  
## 5 32.53023576 31.562765779 33.4977057  
## 6 32.43523082 31.474050201 33.3964114  
## 7 32.34022589 31.385315402 33.2951364  
## 8 32.24522095 31.296561001 33.1938809  
## 9 32.15021601 31.207786608 33.0926454  
## 10 32.05521108 31.118991823 32.9914303  
## 11 31.96020614 31.030176238 32.8902360  
## 12 31.86520121 30.941339436 32.7890630  
## 13 31.77019627 30.852480987 32.6879116  
## 14 31.67519134 30.763600455 32.5867822  
## 15 31.58018640 30.674697391 32.4856754  
## 16 31.48518147 30.585771336 32.3845916  
## 17 31.39017653 30.496821822 32.2835312  
## 18 31.29517160 30.407848368 32.1824948  
## 19 31.20016666 30.318850482 32.0814828  
## 20 31.10516173 30.229827662 31.9804958  
## 21 31.01015679 30.140779392 31.8795342  
## 22 30.91515185 30.051705147 31.7785986  
## 23 30.82014692 29.962604387 31.6776895  
## 24 30.72514198 29.873476561 31.5768074  
## 25 30.63013705 29.784321104 31.4759530  
## 26 30.53513211 29.695137439 31.3751268  
## 27 30.44012718 29.605924976 31.2743294  
## 28 30.34512224 29.516683111 31.1735614  
## 29 30.25011731 29.427411226 31.0728234  
## 30 30.15511237 29.338108688 30.9721161  
## 31 30.06010744 29.248774852 30.8714400  
## 32 29.96510250 29.159409056 30.7707959  
## 33 29.87009757 29.070010626 30.6701845  
## [ reached 'max' / getOption("max.print") -- omitted 330 rows ]
```

```
pred<-predict(lm.fit,newdata,interval="prediction")  
pred
```

```
##          fit      lwr      upr  
## 1 32.91025550 20.65797246 45.16254
```

```

## 2 32.81525056 20.56348145 45.06702
## 3 32.72024563 20.46898573 44.97151
## 4 32.62524069 20.37448531 44.87600
## 5 32.53023576 20.27998018 44.78049
## 6 32.43523082 20.18547035 44.68499
## 7 32.34022589 20.09095581 44.58950
## 8 32.24522095 19.99643656 44.49401
## 9 32.15021601 19.90191260 44.39852
## 10 32.05521108 19.80738393 44.30304
## 11 31.96020614 19.71285055 44.20756
## 12 31.86520121 19.61831246 44.11209
## 13 31.77019627 19.52376966 44.01662
## 14 31.67519134 19.42922215 43.92116
## 15 31.58018640 19.33466993 43.82570
## 16 31.48518147 19.24011299 43.73025
## 17 31.39017653 19.14555134 43.63480
## 18 31.29517160 19.05098498 43.53936
## 19 31.20016666 18.95641390 43.44392
## 20 31.10516173 18.86183811 43.34849
## 21 31.01015679 18.76725760 43.25306
## 22 30.91515185 18.67267238 43.15763
## 23 30.82014692 18.57808244 43.06221
## 24 30.72514198 18.48348779 42.96680
## 25 30.63013705 18.38888842 42.87139
## 26 30.53513211 18.29428433 42.77598
## 27 30.44012718 18.19967552 42.68058
## 28 30.34512224 18.10506200 42.58518
## 29 30.25011731 18.01044375 42.48979
## 30 30.15511237 17.91582079 42.39440
## 31 30.06010744 17.82119311 42.29902
## 32 29.96510250 17.72656070 42.20364
## 33 29.87009757 17.63192358 42.10827
## [ reached 'max' / getOption("max.print") -- omitted 330 rows ]
all.ints<-cbind(conf,pred[, -1])
all.ints
```

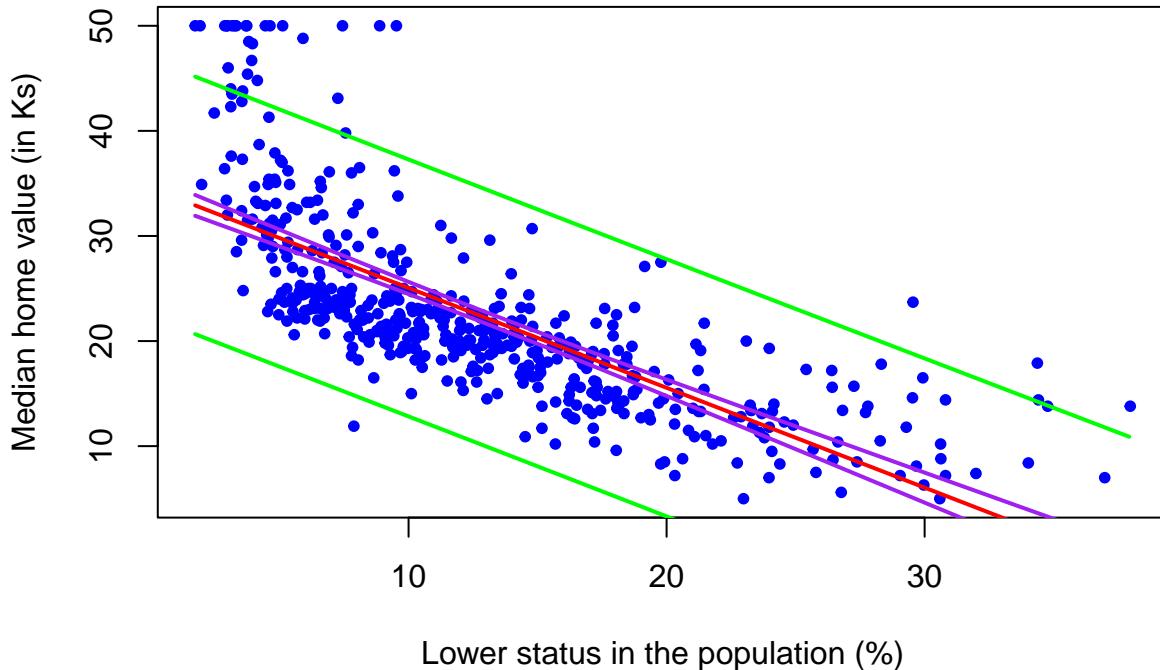
	fit	lwr	upr	lwr	upr
## 1	32.91025550	31.917443200	33.9030678	20.65797246	45.16254
## 2	32.81525056	31.828800866	33.8017003	20.56348145	45.06702
## 3	32.72024563	31.740140752	33.7003505	20.46898573	44.97151
## 4	32.62524069	31.651462508	33.5990189	20.37448531	44.87600
## 5	32.53023576	31.562765779	33.4977057	20.27998018	44.78049
## 6	32.43523082	31.474050201	33.3964114	20.18547035	44.68499
## 7	32.34022589	31.385315402	33.2951364	20.09095581	44.58950
## 8	32.24522095	31.296561001	33.1938809	19.99643656	44.49401
## 9	32.15021601	31.207786608	33.0926454	19.90191260	44.39852
## 10	32.05521108	31.118991823	32.9914303	19.80738393	44.30304
## 11	31.96020614	31.030176238	32.8902360	19.71285055	44.20756
## 12	31.86520121	30.941339436	32.7890630	19.61831246	44.11209
## 13	31.77019627	30.852480987	32.6879116	19.52376966	44.01662
## 14	31.67519134	30.763600455	32.5867822	19.42922215	43.92116
## 15	31.58018640	30.674697391	32.4856754	19.33466993	43.82570
## 16	31.48518147	30.585771336	32.3845916	19.24011299	43.73025
## 17	31.39017653	30.496821822	32.2835312	19.14555134	43.63480

```

## 18 31.29517160 30.407848368 32.1824948 19.05098498 43.53936
## 19 31.20016666 30.318850482 32.0814828 18.95641390 43.44392
## 20 31.10516173 30.229827662 31.9804958 18.86183811 43.34849
## [ reached 'max' / getOption("max.print") -- omitted 343 rows ]
plot(lstat, medv,
      main="Median value vs. lower status",
      xlab="Lower status in the population (%)",
      ylab="Median home value (in Ks)",
      pch=20, col="blue")
matplot(newdata, all.ints, type="l", lty=1, lwd=2, col=c("red", "purple", "purple", "green", "green"), add=TRUE)

```

Median value vs. lower status



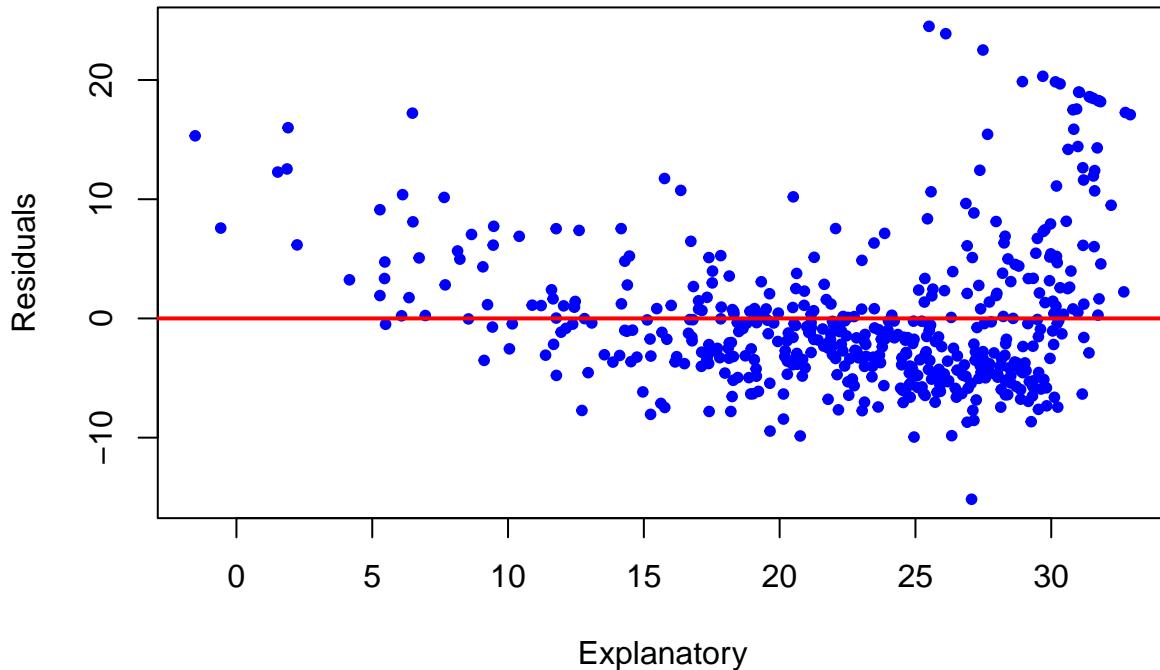
We can compute the residuals from a linear regression fit using the `residuals()` function. The function `rstudent()` will return the studentized residuals, and we can use this function to plot the residuals against the fitted values.

```

plot(predict(lm.fit), residuals(lm.fit),
      main="Do residuals depend on the explanatory?",
      xlab="Explanatory",
      ylab="Residuals",
      pch=20, col="blue")
abline(0,0, col="red", lwd=2)

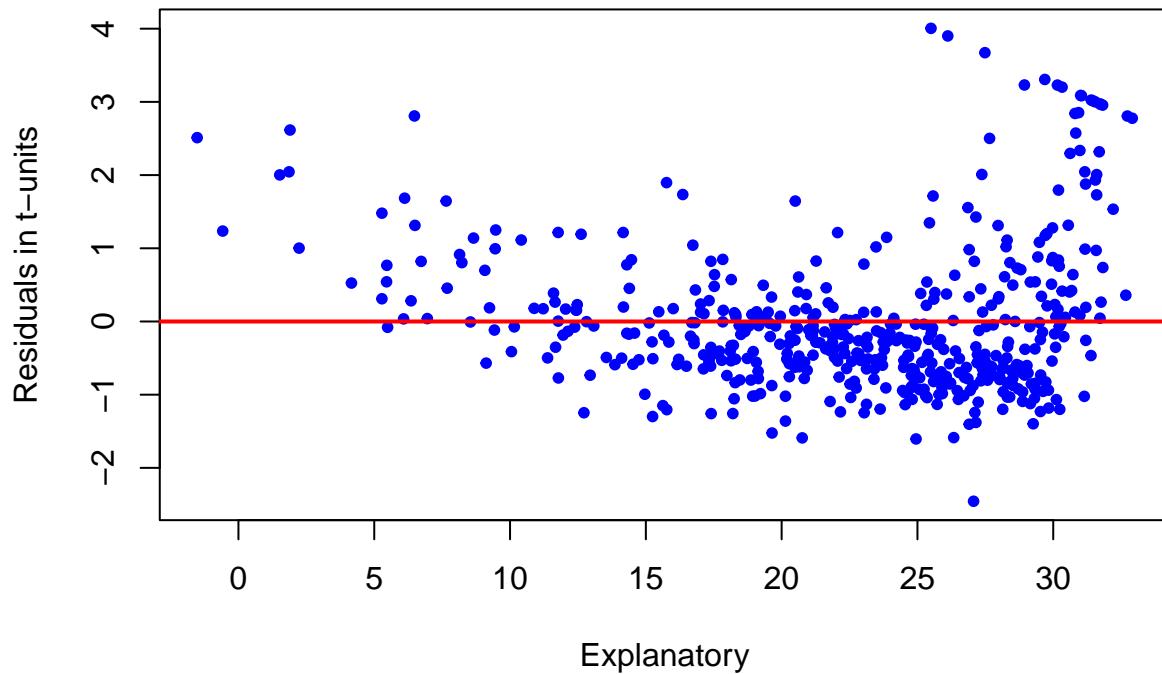
```

Do residuals depend on the explanatory?



```
plot(predict(lm.fit), rstudent(lm.fit),
  main="Do residuals depend on the explanatory?",
  xlab="Explanatory",
  ylab="Residuals in t-units",
  pch=20, col="blue")
abline(0,0, col="red", lwd=2)
```

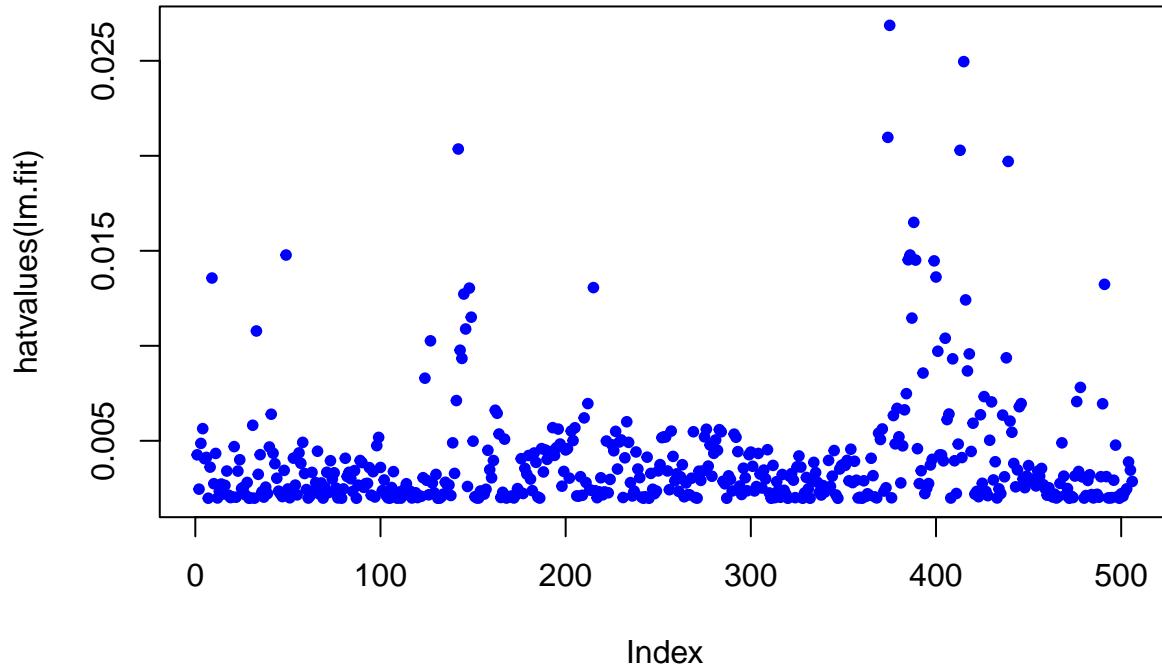
Do residuals depend on the explanatory?



On the basis of the residual plots, there is some evidence of non-linearity.

Leverage statistics can be computed for any number of predictors using the `hatvalues()` function.

```
plot(hatvalues(lm.fit),  
     pch=20, col="blue")
```



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

```
## 375
```

```
## 375
```

The `which.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.

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.

```
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 data set contains 12 variables, and so it would be cumbersome to have to type all of these in order to perform a regression using all of the predictors. Instead, we can use the following short-hand:

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

##
## Call:
## lm(formula = medv ~ ., data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q      Max 
## -15.1304  -2.7673  -0.5814   1.9414  26.2526 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 41.617270  4.936039  8.431 3.79e-16 ***
## crim        -0.121389  0.033000 -3.678 0.000261 *** 
## zn          0.046963  0.013879  3.384 0.000772 *** 
## indus       0.013468  0.062145  0.217 0.828520  
## chas        2.839993  0.870007  3.264 0.001173 **  
## nox        -18.758022 3.851355 -4.870 1.50e-06 *** 
## rm          3.658119  0.420246  8.705 < 2e-16 ***
```

```

## age          0.003611  0.013329  0.271 0.786595
## dis         -1.490754  0.201623 -7.394 6.17e-13 ***
## rad          0.289405  0.066908  4.325 1.84e-05 ***
## tax         -0.012682  0.003801 -3.337 0.000912 ***
## ptratio      -0.937533  0.132206 -7.091 4.63e-12 ***
## lstat        -0.552019  0.050659 -10.897 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.798 on 493 degrees of freedom
## Multiple R-squared:  0.7343, Adjusted R-squared:  0.7278
## F-statistic: 113.5 on 12 and 493 DF,  p-value: < 2.2e-16

```

What if we would like to perform a regression using all of the variables but one? For example, in the above regression output, `age` has a high p -value. So we may wish to run a regression excluding this predictor. The following syntax results in a regression using all predictors except `age`.

```

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

##
## Call:
## lm(formula = medv ~ . - age, data = Boston)
##
## Residuals:
##    Min     1Q   Median     3Q    Max 
## -15.1851 -2.7330 -0.6116  1.8555 26.3838
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 41.525128  4.919684  8.441 3.52e-16 ***
## crim        -0.121426  0.032969 -3.683 0.000256 *** 
## zn          0.046512  0.013766  3.379 0.000785 *** 
## indus       0.013451  0.062086  0.217 0.828577    
## chas        2.852773  0.867912  3.287 0.001085 **  
## nox        -18.485070 3.713714 -4.978 8.91e-07 *** 
## rm          3.681070  0.411230  8.951 < 2e-16 *** 
## dis        -1.506777  0.192570 -7.825 3.12e-14 *** 
## rad         0.287940  0.066627  4.322 1.87e-05 *** 
## tax        -0.012653  0.003796 -3.333 0.000923 *** 
## ptratio     -0.934649  0.131653 -7.099 4.39e-12 *** 
## lstat       -0.547409  0.047669 -11.483 < 2e-16 *** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.794 on 494 degrees of freedom
## Multiple R-squared:  0.7343, Adjusted R-squared:  0.7284
## F-statistic: 124.1 on 11 and 494 DF,  p-value: < 2.2e-16

```

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

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

Take away the next explanatory with a sizeable p -value, i.e., `indus`.

```

lm.fit2 <- lm(medv ~ . - age - indus, data = Boston)
summary(lm.fit2)

```

```

## 
## Call:
## lm(formula = medv ~ . - age - indus, data = Boston)
## 
## Residuals:
##      Min      1Q Median      3Q     Max 
## -15.1814 -2.7625 -0.6243  1.8448 26.3920 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 41.451747  4.903283  8.454 3.18e-16 ***
## crim        -0.121665  0.032919 -3.696 0.000244 *** 
## zn          0.046191  0.013673  3.378 0.000787 *** 
## chas        2.871873  0.862591  3.329 0.000935 *** 
## nox       -18.262427  3.565247 -5.122 4.33e-07 *** 
## rm          3.672957  0.409127  8.978 < 2e-16 *** 
## dis        -1.515951  0.187675 -8.078 5.08e-15 *** 
## rad         0.283932  0.063945  4.440 1.11e-05 *** 
## tax        -0.012292  0.003407 -3.608 0.000340 *** 
## ptratio     -0.930961  0.130423 -7.138 3.39e-12 *** 
## lstat      -0.546509  0.047442 -11.519 < 2e-16 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 4.789 on 495 degrees of freedom 
## Multiple R-squared:  0.7342, Adjusted R-squared:  0.7289 
## F-statistic: 136.8 on 10 and 495 DF,  p-value: < 2.2e-16

```

The above approach is called backward selection. How would you design forward selection?

Interaction Terms

It is easy to include interaction terms in a linear model using the `lm()` function. The syntax `lstat:age` tells R to include an interaction term between `lstat` and `age`.

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

```

## 
## Call:
## lm(formula = medv ~ lstat:age, data = Boston)
## 
## Residuals:
##      Min      1Q Median      3Q     Max 
## -13.347 -4.372 -1.534  1.914 27.193 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 30.1588631  0.4828240  62.46   <2e-16 ***
## lstat:age   -0.0077146  0.0003799 -20.31   <2e-16 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 6.827 on 504 degrees of freedom 
## Multiple R-squared:  0.4501, Adjusted R-squared:  0.449 
## F-statistic: 412.4 on 1 and 504 DF,  p-value: < 2.2e-16

```

```

summary(lm(medv ~ lstat + age + lstat:age, data = Boston))

##
## Call:
## lm(formula = medv ~ lstat + age + 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
summary(lm(medv ~ lstat + lstat:age, data = Boston))

```

```

##
## Call:
## lm(formula = medv ~ lstat + lstat:age, data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max 
## -15.815 -4.039 -1.335  2.086 27.491 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 36.041514   0.691334 52.133 < 2e-16 ***
## lstat       -1.388161   0.126911 -10.938 < 2e-16 ***
## lstat:age    0.004103   0.001133   3.621 0.000324 *** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.142 on 503 degrees of freedom
## Multiple R-squared:  0.5557, Adjusted R-squared:  0.554  
## F-statistic: 314.6 on 2 and 503 DF,  p-value: < 2.2e-16

```

The syntax `lstat * age` simultaneously includes `lstat`, `age`, and the interaction term `lstat×age` as predictors; it is a shorthand for `lstat + age + lstat:age`. We can also pass in transformed versions of the predictors.

```

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
```