Introduction to Statistical Modelling in R

BCAM - Basque Center for Applied Mathematics, Applied Statistics

Dae-Jin Lee < dlee@bcamath.org >

CHAPTER 5. Basic Statistical Modelling in R

Contents

1 Statistical Modelling principles

- **Given:** a collection of variables, each variable being a vector of readings of a specific trait on the samples in an experiment.
- **Problem:** In what way does a variable Y depend on other variables $X_1, ..., X_n$ in the study.
- **Explanation:** A statistical model defines a mathematical relationship between the X_i's and Y. The model is a representation of the real Y that aims to replace it as far as possible. At least the model should capture the dependence of Y on the X_i's

1.1 Types of variables in a statistical model

The **response variable** is the one whose content we are trying to model with other variables, called the **explanatory variables**. In any given model there is one response variable (Y above) and there may be many explanatory variables (like $X_1, ...X_n$).

1.2 Identify and Characterize Variables

This is the first step in modelling:

- Which variable is the response variable;
- Which variables are the explanatory variables;
- Are the explanatory variables continuous, categorical, or a mixture of both;
- What is the nature of the response variable - is it a continuous measurement, a count, a proportion, a category, or a time-at-death?

1.2.1 Types of Variables Determine Type of Model

The explanatory variables	**Model**
All explanatory variables continuous	Regression
All explanatory variables categorical	Analysis of Variance (ANOVA)
Explanatory variables both continuous and categorical	Regression, Analysis of Covariance (ANCOVA)

The response variable	what kind of data is it?
Continuous	Normal Regression, Anova, Ancova
Proportion	Logistic regression
Counts	Log-linear models (a.k.a Poisson regression)
Binary	Binary Logistic regression
Time-at-death	Survival Analysis

2 Linear models in R

2.1 Simple linear regression

- Regression is a statistical method used to predict the value of a response variable based on the values of a set of explanatory variables.
- One very general form for the model would be

$$y = f(x_1, x_2, ..., x_p) + \epsilon,$$

where f is some unknown function and ϵ is the error in this representation. Since we usually don't have enough data to try to estimate f directly (*inverse problem*), we usually have to assume that it has some restricted form.

- Any statistical model attempts to approximate the response variable or dependent variable y as a mathematical function of the explanatory variables or regressors X (also called covariates or independent variables).
- The simplest and most common form is the **linear model (LM)**

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_z + \epsilon,$$

where β_i $i = 0, 1, 2$ are *unknown* parameters. β_0 is called the intercept term. Hence, the problem is reduced to the estimation of four values rather than the complicated infinite dimensional f.

- A simple linear model with a single exploratory variable is defined as:

$$\hat{y} = \beta_0 + \beta_1 x$$

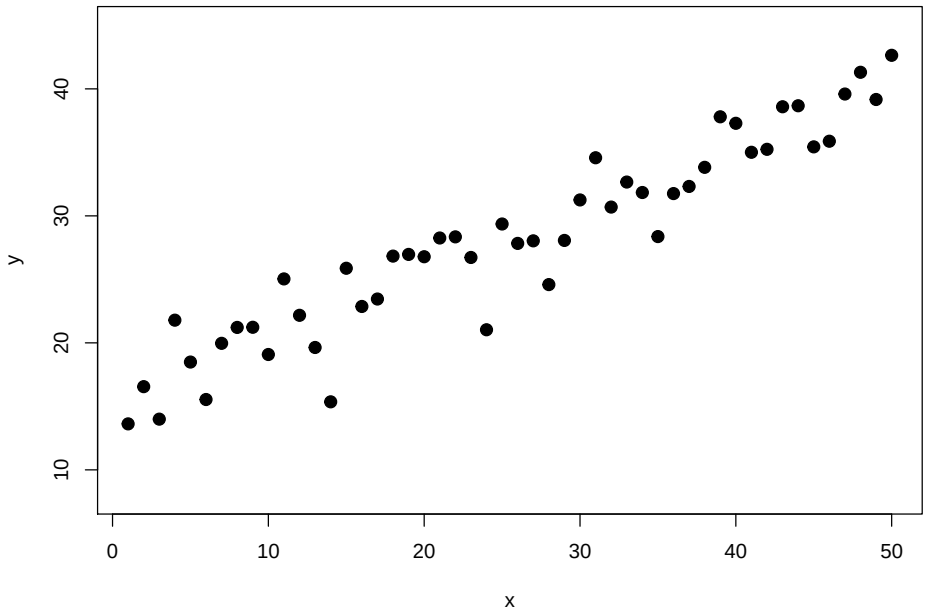
where $\hat{y}$ is the fitted values for β_0 (intercept) and β_1 (slope). Then for a given x_i we obtain a $\hat{y}_i$ that approximates y_i

Let us create a toy example (with $p = 1$):

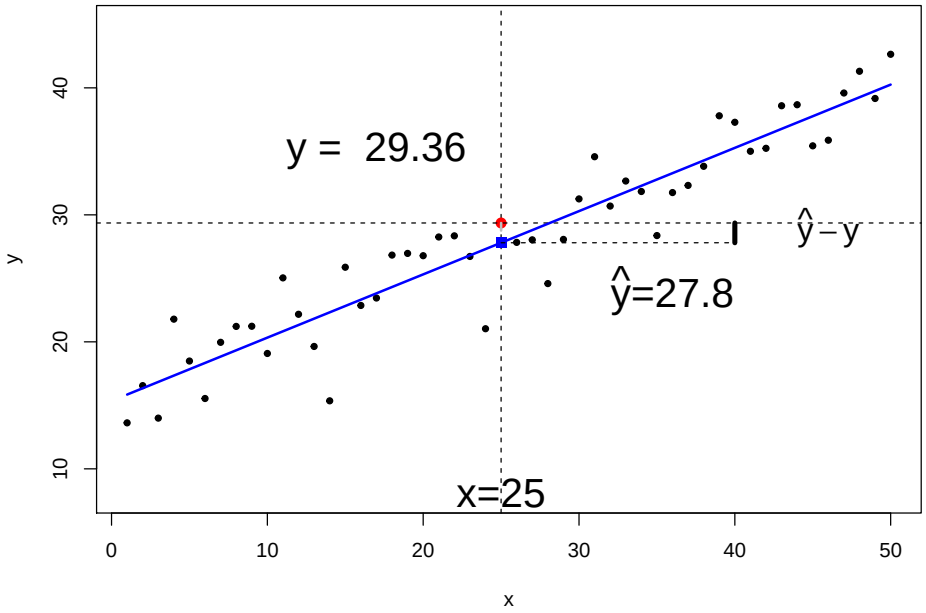
```r
set.seed(1)
n <- 50

x <- seq(1,n)
 beta0 <- 15
 beta1 <- 0.5

sigma <- 3 # standar deviation of the errors
eps <- rnorm(n,mean=0,sd=3) # generate gaussian random errors

# Generate random data
 y <- beta0 + beta1*x  +  eps
```

Plot the data

```r
plot(x,y,ylim = c(8,45), cex=1.3, xlab = "x", ylab="y",pch=19)
```



A mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the sum of the squares of the residuals of the points from the fitted line. Illustration of the least squares fit



We can directly calculate quantities of interest, i.e. the ordinary least squares solution consists of:

$$\min_{\beta_0,\beta_1} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Then $\hat{\beta}_1 = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}$ and $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$

In matrix form, with $X = [1 : x_1 : ... : x_p]$

$$\hat{\beta} = (X'X)^{-1}X'y$$

where $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1)$

2.2 Defining models in R

To complete a linear regression using R it is first necessary to understand the syntax for defining models.

- A fundamental aspect of models is the use of model formulas to specify the variables involved in the model and the possible interactions between explanatory variables included in the model.

- A model formula is input into a function that performs a linear regression or anova, for example.
- While a model formula bears some resemblance to a mathematical formula, the symbols in the "equation" mean different things than in algebra.

Syntax	Model	Comments
`y ~ x`	$y = \beta_0 + \beta_1 x$	Straight-line with an implicit intercept
`y ~ -1 + x`	$y = \beta_1 x$	Straight-line with no intercept; that is, a fit forced through (0,0)
`y ~ x + I(x^2)`	$y = \beta_0 + \beta_1 x + \beta_2 x^2$	Polynomial model; `I()` allows for mathematical symbols
`y ~ x + z`	$y = \beta_0 + \beta_1 x + \beta_2 z$	Multiple regression model
`y ~ x:z`	$y = \beta_0 + \beta_1 xz$	Model with interaction between x and z
`y ~ x*z`	$y = \beta_0 + \beta_1 x + \beta_2 z + \beta_3 xz$	Equivalent to `y~x+z+x:z`

2.3 Example: Boston Housing data

The `MASS` library contains the Boston data set, which records `medv` (median house value) for 506 neighborhoods around Boston. We will seek to predict medv using 13 predictors such as `rm` (average number of rooms per house), `age` (average age of houses), and `lstat` (percent of households with low socioeconomic status).

```
library(MASS)
data("Boston")
?Boston
```

```
# Some plots
plot(Boston$crim,Boston$medv,col=1+Boston$chas)
legend('topright', legend = levels(factor(Boston$chas)), col = 1:2, cex = 0.8, pch = 1)

plot(Boston[Boston$chas==0,c("crim","medv")],xlim=range(Boston$crim),ylim=range(Boston$medv)
points(Boston[Boston$chas==1,c("crim","medv")],col="red",pch=2)
legend("topright",c("CHAS = 0", "CHAS = 1"), col=c(4,2),pch=c(1,2))

boxplot(crim,data=Boston)
boxplot(crim ~ factor(chas), data = Boston,xlab="CHAS",ylab="crim",col=c(4,2),varwidth=TRUE)
boxplot(medv ~ factor(chas), data = Boston,xlab="CHAS",ylab="crim",col=c(4,2),varwidth=TRUE)
```

```
library(ggplot2)

qplot(crim,medv,data=Boston, colour=factor(chas))
qplot(crim,medv,data=Boston, colour=tax)


library(lattice)
xyplot(medv~crim,groups=factor(chas),auto.key = TRUE)
xyplot(medv~crim|factor(chas),auto.key = TRUE)
```

```
names(Boston)
```

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

```
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

We will start by using the `lm()` function to fit a simple linear regression model, with `medv` as the response and lstat as the predictor. The basic `lm()` 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, data=Boston)
```

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

```
lm.fit
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Coefficients:
## (Intercept)        lstat
##       34.55        -0.95
```

```
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`. 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"
```

```
lm.fit$coefficients
```

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

```
lm.fit[[1]]
```

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

```
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, level = 0.95)
```

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

Consider constructing a confidence interval for β_1 using the information provided from the summary of `lm.fit`:

```
summary(lm.fit)$coefficients
```

```
##               Estimate Std. Error   t value      Pr(>|t|)
## (Intercept) 34.5538409 0.56262735  61.41515 3.743081e-236
## lstat       -0.9500494 0.03873342 -24.52790  5.081103e-88
```

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

```
CI <- predict(object = lm.fit, newdata = data.frame(lstat = c(5, 10, 15)),
              interval = "confidence")
CI
```

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

```
PI <- predict(object = lm.fit, newdata = data.frame(lstat = c(5, 10, 15)),
              interval = "predict")
PI
```

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

NOTE:

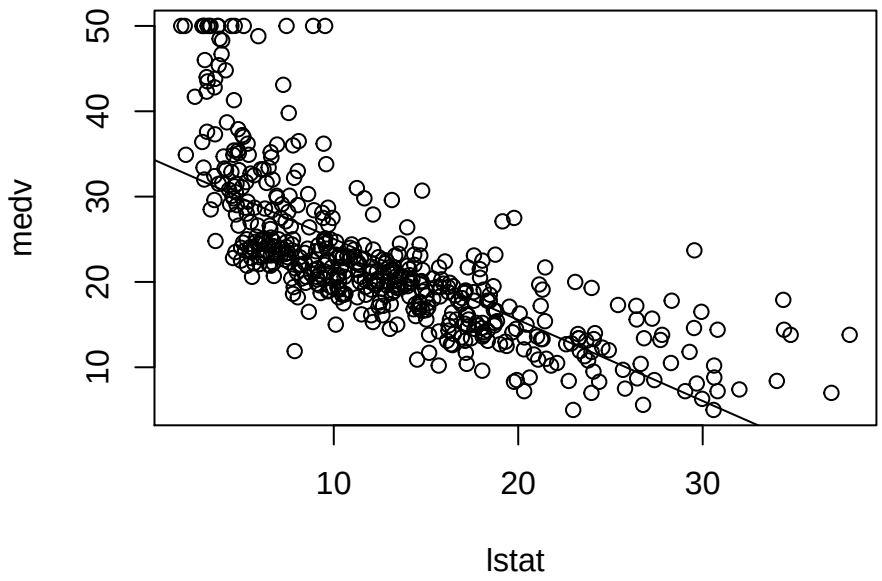
A **prediction interval** is an interval associated with a random variable yet to be observed (forecasting).

A **confidence interval** is an interval associated with a parameter and is a frequentist concept.

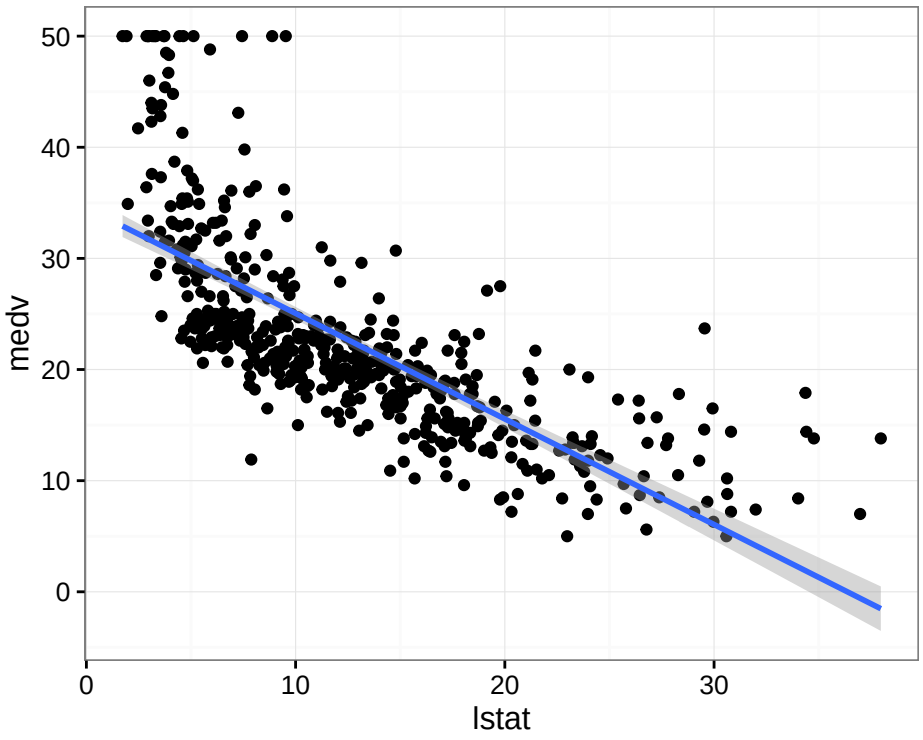
For instance, the 95% confidence interval associated with a `lstat` value of 10 is $(24.474132, 25.6325627)$ and the 95% prediction interval is $(12.8276263, 37.2790683)$. As expected, the confidence and prediction intervals are centered around the same point (a predicted value of 25.0533473 for medv when `lstat` equals 10), but the latter are substantially wider.

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

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



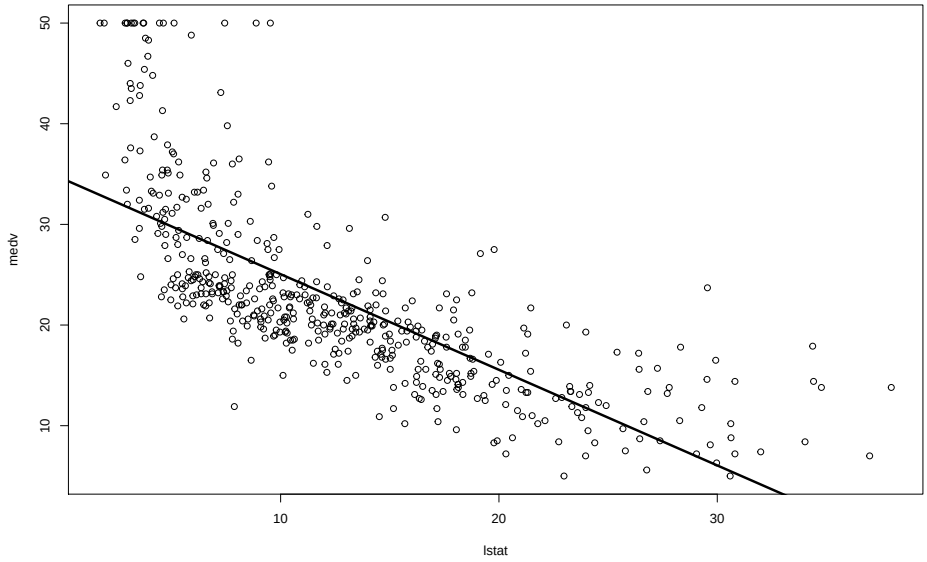

```
# Or using ggplot2
library(ggplot2)
ggplot(data = Boston, aes(x = lstat, y = medv)) +
  geom_point() +
  geom_smooth(method = "lm") +
  theme_bw()
```



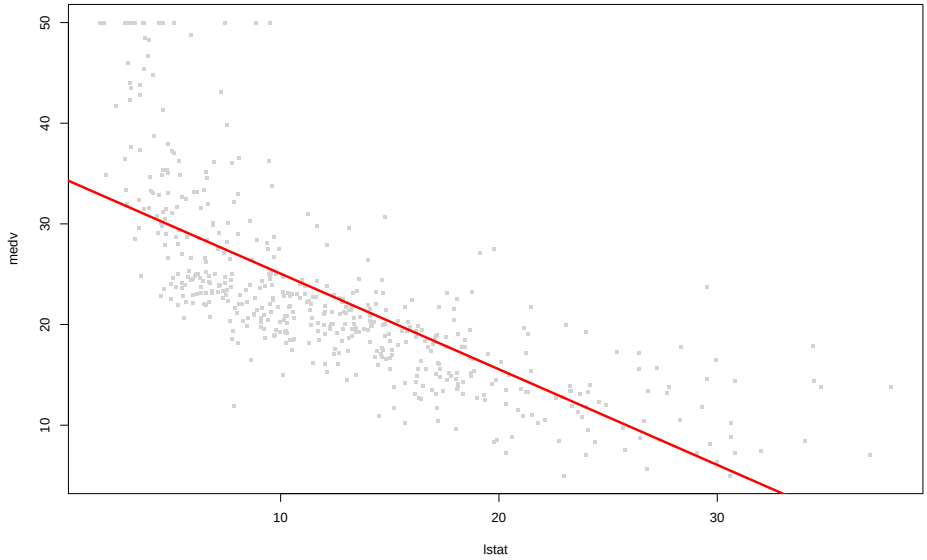
There is some evidence for non-linearity in the relationship between `lstat` and `medv`. This issue will be discussed later.

The `abline()` function can be used to draw any line, not just the least squares regression line. To draw a line with intercept `a` and slope `b`, we type `abline(a, b)`. Below we experiment with some additional settings 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(medv ~ lstat, data = Boston)
abline(lm.fit, lwd = 3)
```

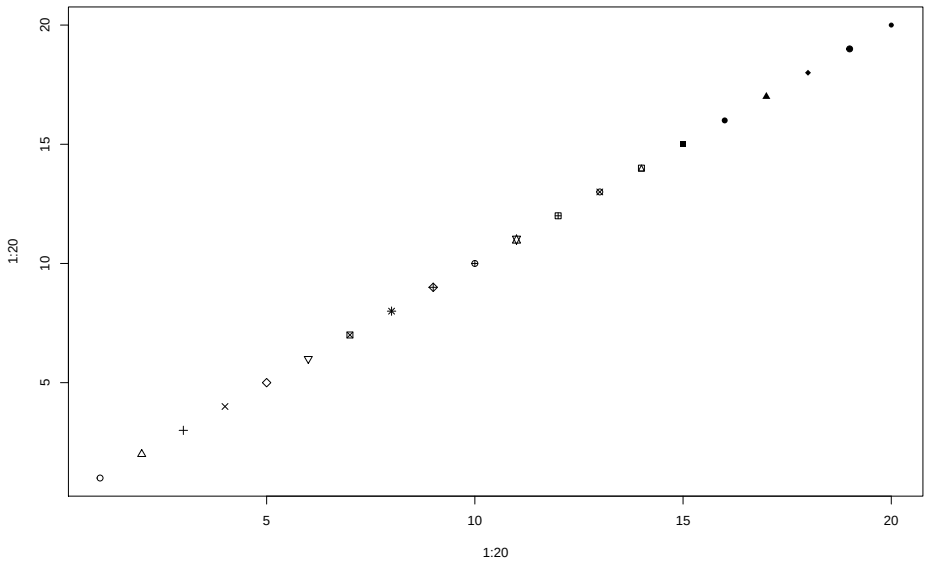



```
plot(medv ~ lstat, data = Boston,pch=15,cex=.65,col="lightgrey")
abline(lm.fit, lwd = 3, col = "red")
```



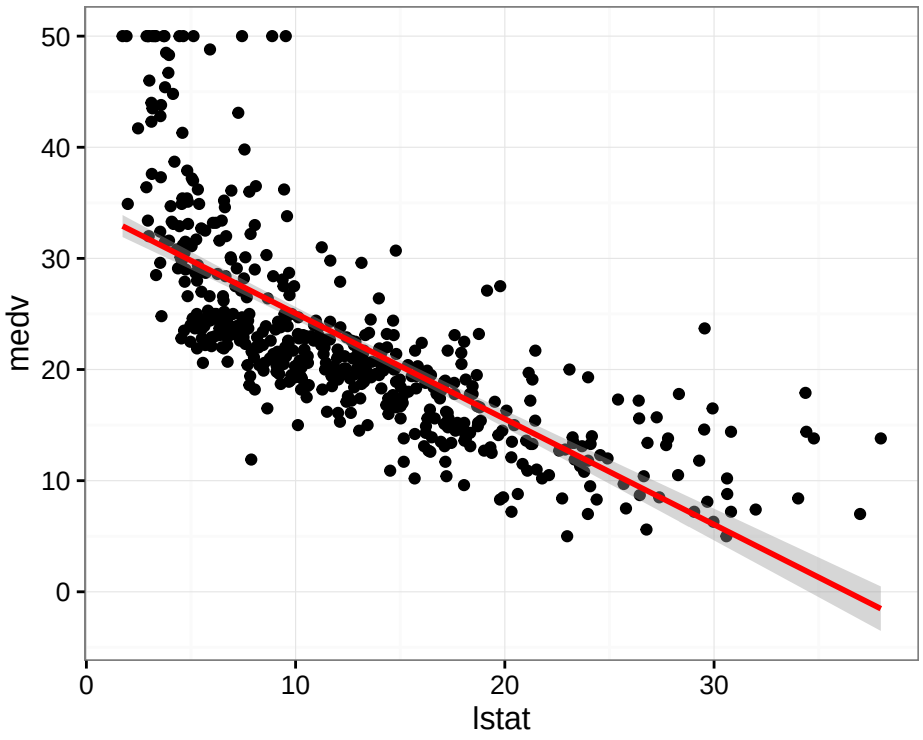
`pch` options

```
plot(1:20, 1:20, pch = 1:20)
```

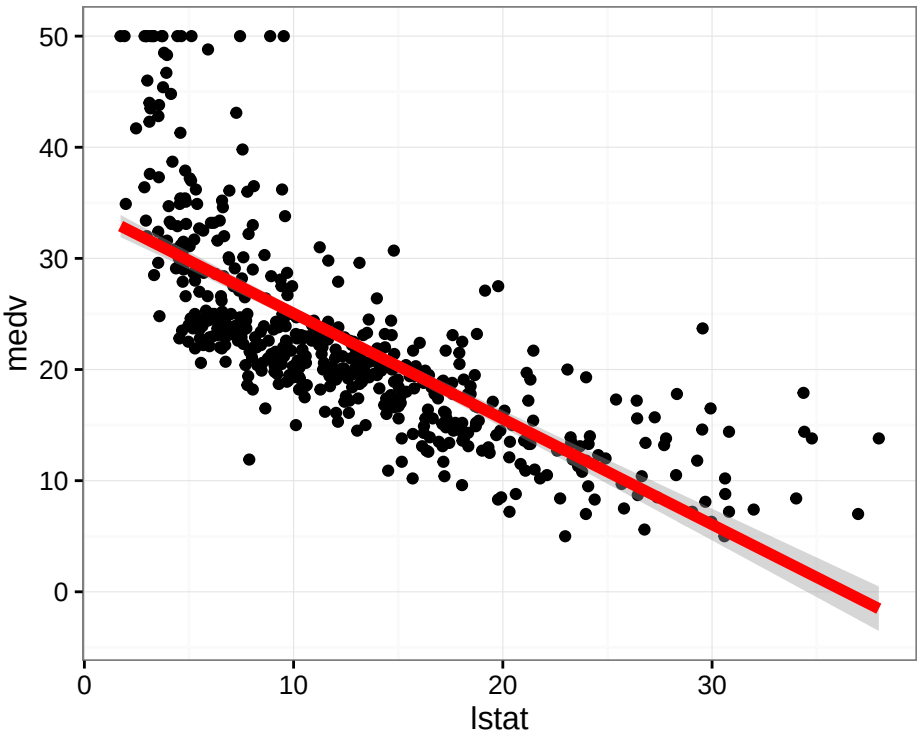


2.3.1 Using ggplot2

```
ggplot(data = Boston, aes(x = lstat, y = medv)) +
  geom_point() +
  geom_smooth(method = "lm", color = "red") +
  theme_bw()
```

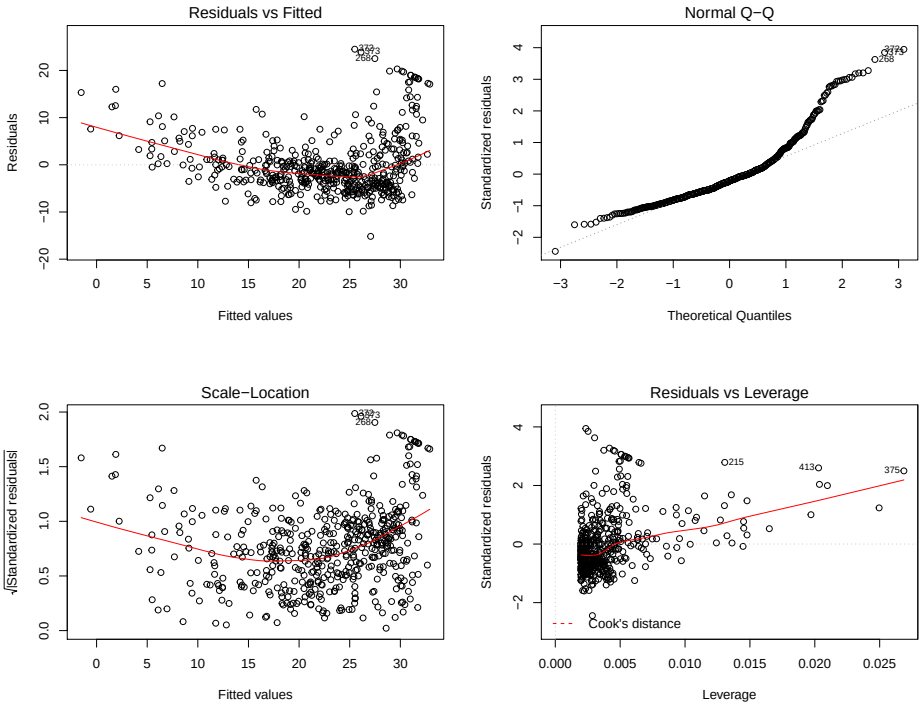



```
# thicker line
ggplot(data = Boston, aes(x = lstat, y = medv)) +
  geom_point() +
  geom_smooth(method = "lm", color = "red", size = 2) +
  theme_bw()
```



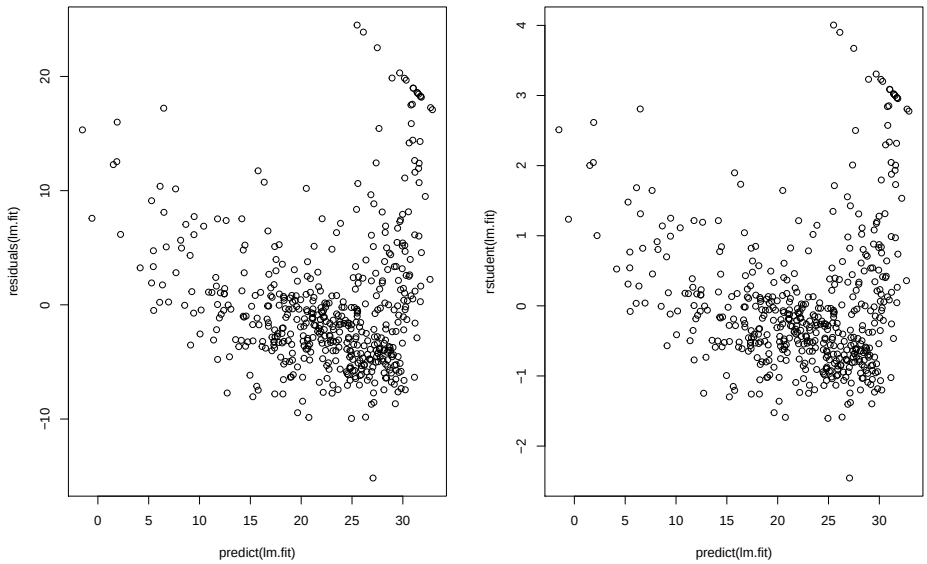
Next we examine some diagnostic plots. Four diagnostic plots are automatically produced by applying the `plot()` function directly to the output from `lm()`. In general, this command will produce one plot at a time, and hitting Enter will generate the next plot. However, it is often convenient to view all four plots together. We can achieve this by using the `par()` function, which tells `R` to split the display screen into separate panels so that multiple plots can be viewed simultaneously. For example, `par(mfrow = c(2, 2))` divides the plotting region into a 2×2 grid of panels.

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



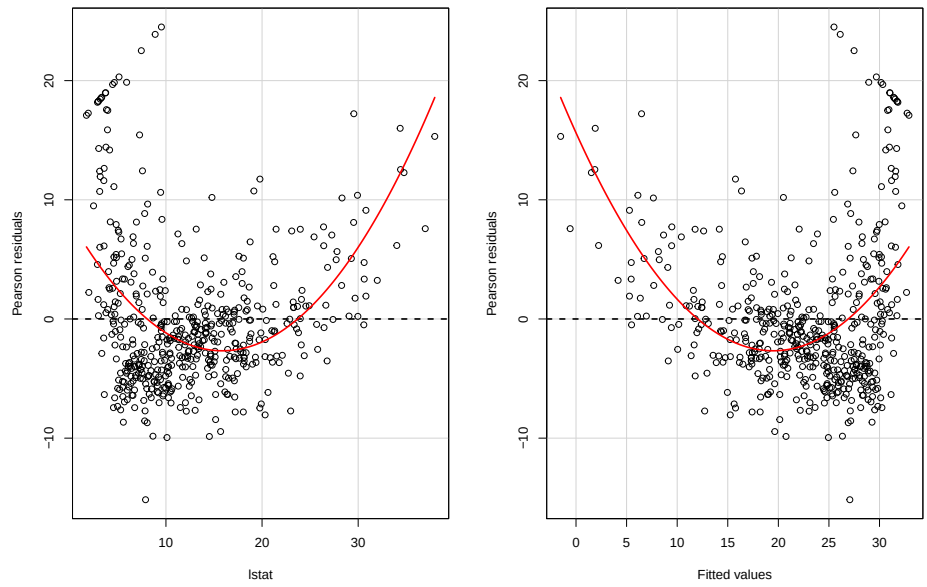
Alternatively, we can compute the residuals from a linear regression fit using the `residuals()` function. The function `rstudent()` will return the studentized residuals, and we can use this function to plot the residuals against the fitted values.

```
par(mfrow = c(1, 2))
plot(predict(lm.fit), residuals(lm.fit))
plot(predict(lm.fit), rstudent(lm.fit))
```



The library `car` has a function `residualPlots` to evaluate residuals (it computes a curvature test for each of the plots by adding a quadratic term and testing the quadratic to be zero). See `?residualPlots`

```
library(car)
residualPlots(lm.fit)
```

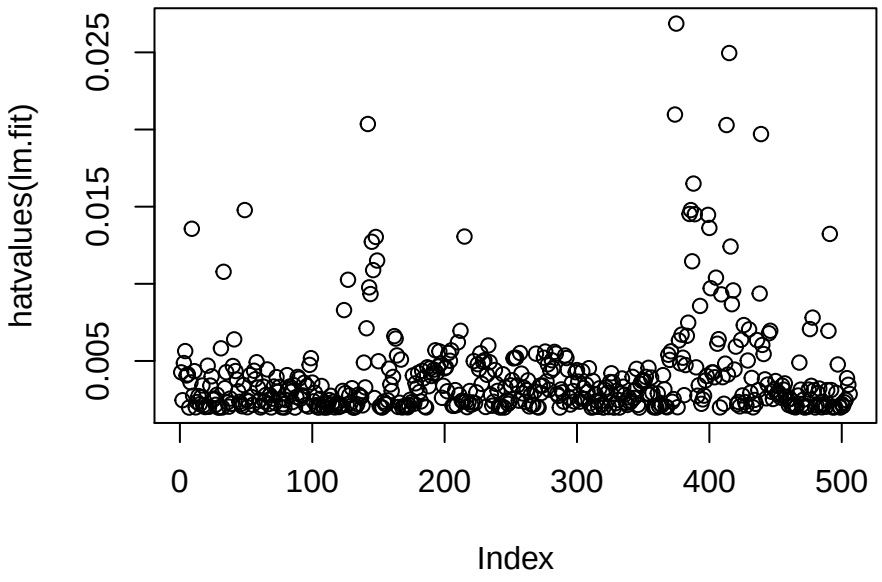



```
##             Test stat Pr(>|t|)
```

```
## lstat          11.628        0
## Tukey test     11.628        0
```

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. The function `influenceIndexPlot` from the `car` package creates four diagnostic plots including a plot of the hat-values.

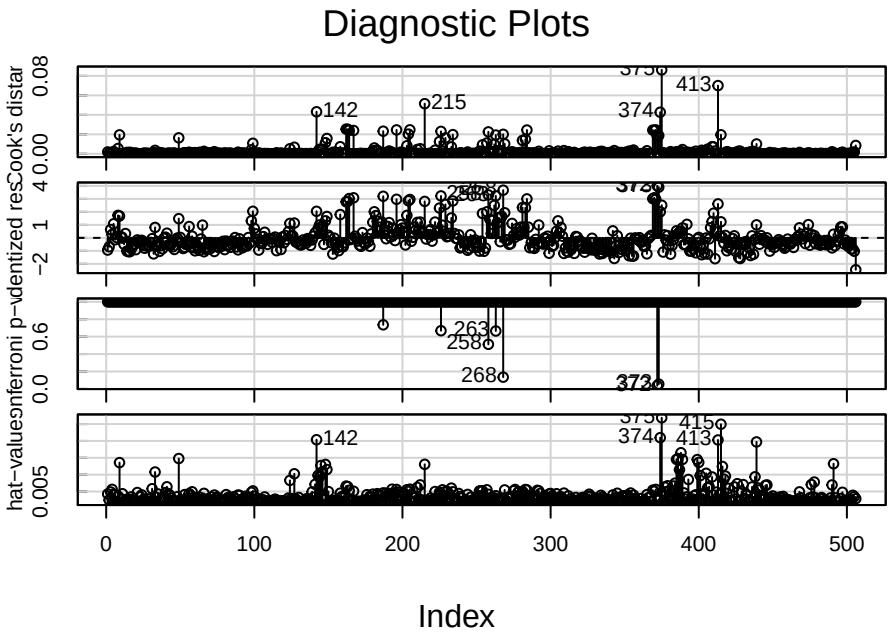
```
plot(hatvalues(lm.fit))
```




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

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

```
influenceIndexPlot(lm.fit, id.n = 5)
```



2.4 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.

```
ls.fit <- lm(medv ~ lstat + age, data = Boston)
summary(ls.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 13 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:

```
ls.fit <- lm(medv ~ ., data = Boston)
summary(ls.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 $\hat{\sigma}$.

If we would like to perform a regression using all of the variables but except one, we can remove it using `-`. 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`.

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

2.4.1 Interaction Terms

It is 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 a shorthand for `lstat + age + lstat:age`.

```
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
```

2.5 Nonlinear Transformations for 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 `I()` to raise X to the power 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
```

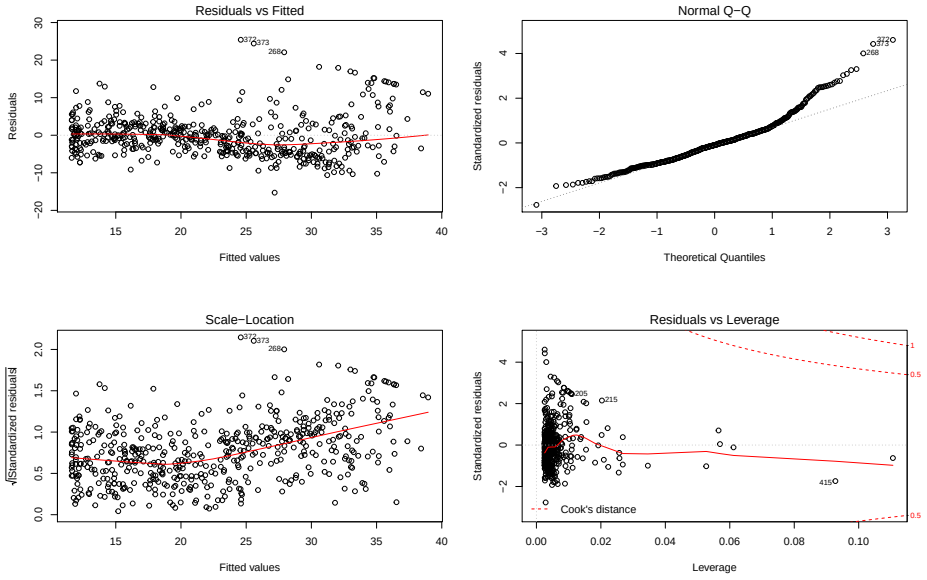
The near-zero p-value associated with the quadratic term suggests that it leads to an improved model. We use the `anova()` function to further quantify the extent to which the quadratic fit is superior to the linear fit.

```
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 (`lm.fit`) represents the linear submodel containing only one predictor, `lstat`, while Model 2 (`lm.fit2`) 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 `135.1998221` and the associated p-value is virtually zero. This provides very 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)
```




```
par(mfrow = c(1, 1))
```

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

In order to create a cubic fit, we can 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 a fifth-order polynomial fit:

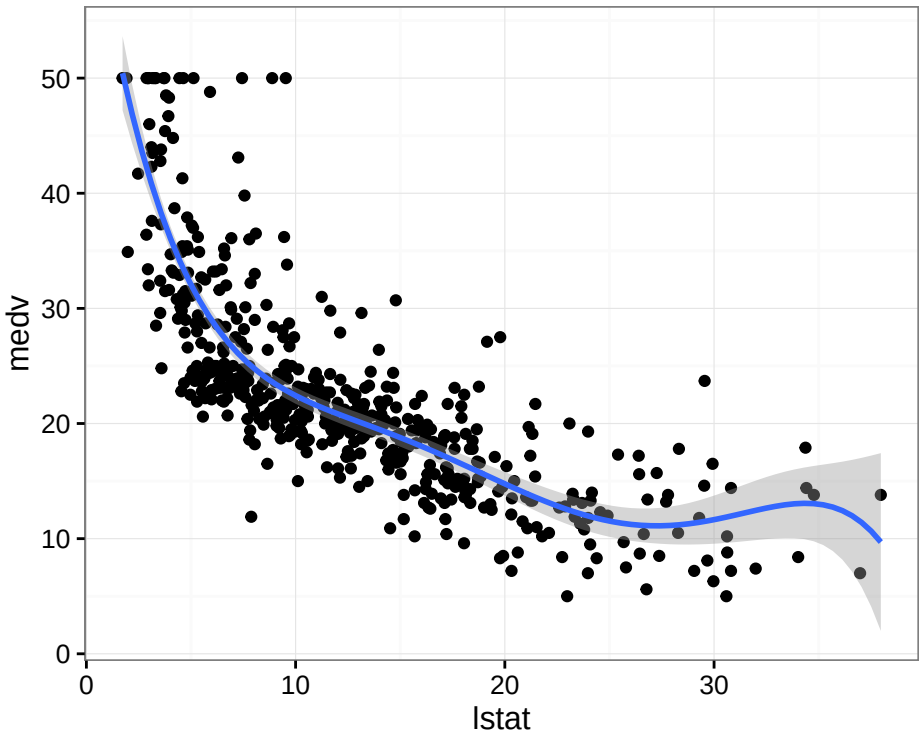
```
lm.fit5 <- lm(medv ~ poly(lstat, 5), data = Boston)
summary(lm.fit5)
```

```
##
## 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 fifth order, leads to an improvement in the model fit! However, further investigation of the data reveals that no polynomial terms beyond fifth order have significant p-values in a regression fit.

```
library(ggplot2)
ggplot(data = Boston, aes(x = lstat, y = medv)) +
  geom_point() +
  theme_bw() +
  stat_smooth(method = "lm", formula = y ~ poly(x, 5))
```



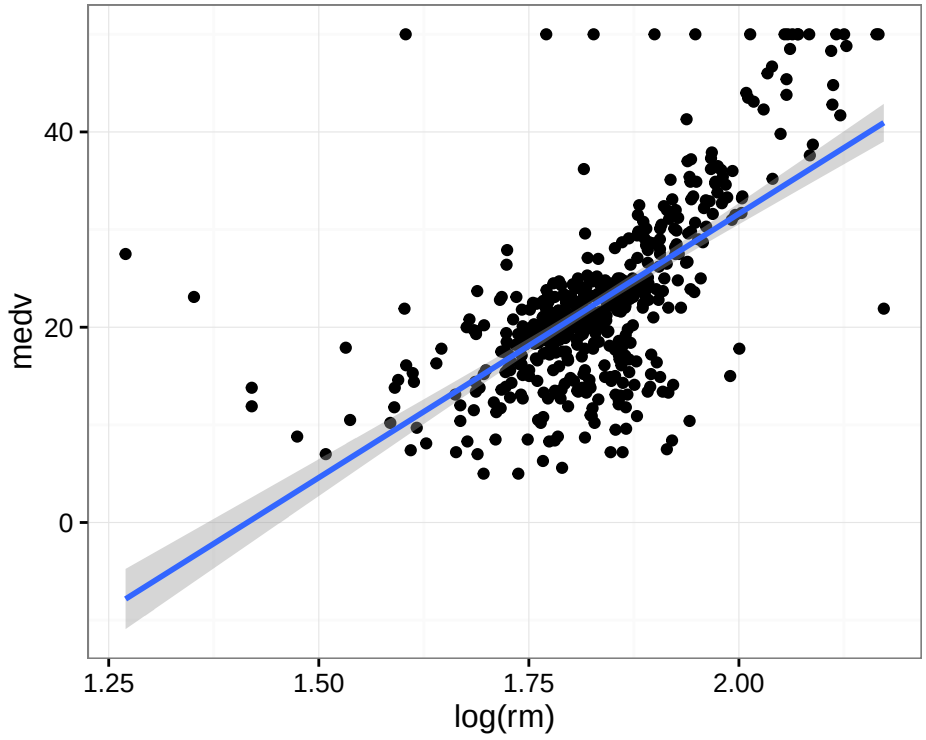
Of course, we are in no way restricted to using polynomial transformations of the predictors. Here we try a log transformation.

```
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
```

```
ggplot(data = Boston, aes(x = log(rm), y = medv)) +
  geom_point() +
  theme_bw() +
  stat_smooth(method = "lm")
```



2.5.1 Qualitative Predictors

We will now examine the `Carseats` data, which is part of the `ISLR` package. We will attempt to predict `Sales` (child car seat sales) in 400 locations based on a number of predictors.

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.3.2
```

```
data(Carseats)
names(Carseats)
```

```
##  [1] "Sales"       "CompPrice"   "Income"      "Advertising" "Population"
##  [6] "Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
## [11] "US"
```

```
?Carseats
```

The `Carseats` data includes qualitative predictors such as `Shelveloc`, an indicator of the quality of the shelving location—that is, the space within a store in which the car seat is displayed—at each location. The predictor `Shelveloc` takes on three possible values, `Bad`, `Medium`, and `Good`. Given a qualitative variable such as `Shelveloc`, `R` generates dummy variables automatically. Below we fit a multiple regression model that includes some interaction terms.

```
lm.fit <- lm(Sales ~ . + Income:Advertising + Price:Age, data = Carseats)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = Sales ~ . + Income:Advertising + Price:Age, data = Carseats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9208 -0.7503  0.0177  0.6754  3.3413
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)         6.5755654  1.0087470   6.519 2.22e-10 ***
## CompPrice           0.0929371  0.0041183  22.567  < 2e-16 ***
## Income              0.0108940  0.0026044   4.183 3.57e-05 ***
## Advertising         0.0702462  0.0226091   3.107 0.002030 **
## Population          0.0001592  0.0003679   0.433 0.665330
## Price              -0.1008064  0.0074399 -13.549  < 2e-16 ***
## ShelveLocGood       4.8486762  0.1528378  31.724  < 2e-16 ***
## ShelveLocMedium     1.9532620  0.1257682  15.531  < 2e-16 ***
## Age                -0.0579466  0.0159506  -3.633 0.000318 ***
## Education          -0.0208525  0.0196131  -1.063 0.288361
## UrbanYes            0.1401597  0.1124019   1.247 0.213171
## USYes              -0.1575571  0.1489234  -1.058 0.290729
## Income:Advertising  0.0007510  0.0002784   2.698 0.007290 **
## Price:Age           0.0001068  0.0001333   0.801 0.423812
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.011 on 386 degrees of freedom
## Multiple R-squared:  0.8761, Adjusted R-squared:  0.8719
## F-statistic:   210 on 13 and 386 DF,  p-value: < 2.2e-16
```

The `contrasts()` function returns the coding that `R` uses for the dummy variables.

```
contrasts(Carseats$ShelveLoc)
```

```
##        Good Medium
## Bad       0      0
## Good      1      0
## Medium    0      1
```

Use `?contrasts` to learn about other contrasts, and how to set them.

`R` has created a `ShelveLocGood` dummy variable that takes on a value of 1 if the shelving location is `good`, and 0 otherwise. It has also created a `ShelveLocMedium` dummy variable that equals 1 if the shelving location is `medium`, and 0 otherwise. A `bad` shelving location corresponds to a zero for each of the two dummy variables.

The fact that the coefficient for `ShelveLocGood` in the regression output is positive indicates that a good shelving location is associated with high sales (relative to a bad location). And `ShelveLocMedium` has a smaller positive coefficient, indicating that a medium shelving location leads to higher sales than a bad shelving location but lower sales than a good shelving location.

More about Linear regression at the free available book "Practical Regression and Anova using R" (Faraway, 2002)

3 Logistic regression

A logistic regression is typically used when there is one dichotomous outcome variable (such as winning or losing), and a continuous predictor variable which is related to the probability or odds of the outcome variable. It can also be used with categorical predictors, and with multiple predictors.

If we use a linear regression to model a dichotomous variable (such as Y), the resulting model may not restrict the predicted Y's within 0 and 1. In addition, other linear regression assumptions such as error normality can be violated. So instead, we modeled the log's event probabilities $\log(\frac{p}{1-p})$ or logit, where, p is the event probability.

$$z_i = \ln(\frac{p_i}{1-p_i}) = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p$$

The above equation can be modeled using `glm()` by the argument `family="binomial`. But we are more interested in the probability of

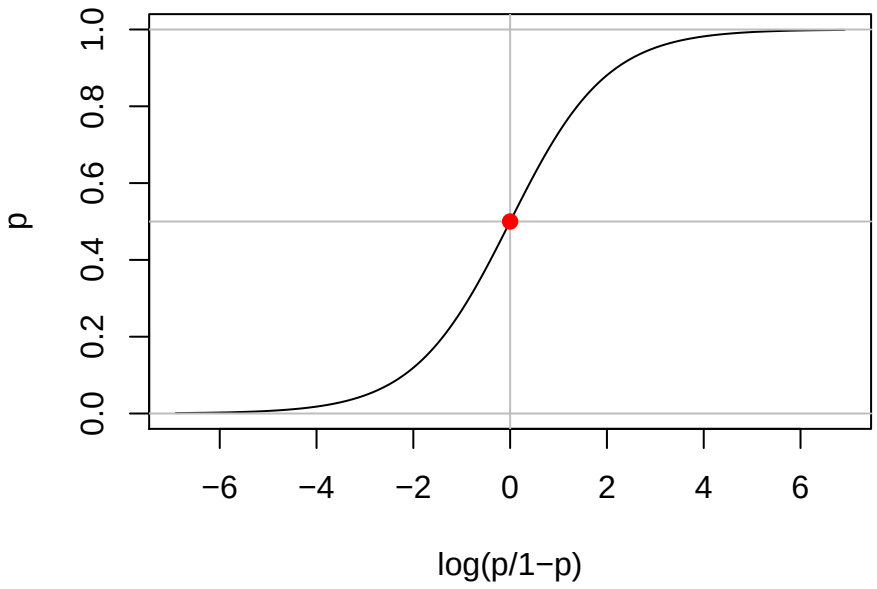
the event than the logarithmic probabilities of the event. Therefore, the predicted values from the previous model, that is, the logarithmic probabilities of the event, can be converted to event probability as follows:

$$p_i = 1 - \frac{1}{1 + \exp(z_i)}$$

This is called the *inverse-logit*, `?plogis`.

The next plot relates `p` and `logit(p)`

```
p <- seq(0,1,l=1000)
logitp <- log(p/(1-p))
plot(logitp,p,t='l',xlab="log(p/1-p)")
abline(h=c(0,0.5,1),v=0,col="grey")
points(0,0.5,pch=19,col="red")
```



3.1 Example: Predict adults salary

Let us consider the data `adult.csv`. We will try to predict the `ABOVE50k` response variable (Salary >50k) through a logistic regression based on explanatory demographic variables.

```
inputData <- read.csv("http://idaejin.github.io/bcam-courses/R/2017/data/adult.csv")
head(inputData)
```

```
##   AGE         WORKCLASS FNLWGT  EDUCATION EDUCATIONNUM       MARITALSTATUS
## 1  39         State-gov  77516  Bachelors           13       Never-married
## 2  50  Self-emp-not-inc  83311  Bachelors           13  Married-civ-spouse
## 3  38           Private 215646    HS-grad            9            Divorced
## 4  53           Private 234721       11th            7  Married-civ-spouse
## 5  28           Private 338409  Bachelors           13  Married-civ-spouse
## 6  37           Private 284582    Masters           14  Married-civ-spouse
##           OCCUPATION   RELATIONSHIP   RACE     SEX CAPITALGAIN CAPITALLOSS
## 1       Adm-clerical  Not-in-family  White    Male        2174           0
## 2    Exec-managerial        Husband  White    Male           0           0
## 3  Handlers-cleaners  Not-in-family  White    Male           0           0
## 4  Handlers-cleaners        Husband  Black    Male           0           0
## 5     Prof-specialty           Wife  Black  Female           0           0
## 6    Exec-managerial           Wife  White  Female           0           0
##   HOURSPERWEEK  NATIVECOUNTRY ABOVE50K
## 1           40  United-States        0
## 2           13  United-States        0
## 3           40  United-States        0
## 4           40  United-States        0
## 5           40           Cuba        0
## 6           40  United-States        0
```

Check Class bias

Ideally, the proportion of events and non-events in the Y variable should approximately be the same. So, lets first check the proportion of classes in the dependent variable `ABOVE50K`.

```
table(inputData$ABOVE50K)
```

```
##
##     0     1
## 24720  7841
```

Clearly, there is a class bias, a condition observed when the proportion of events is much smaller than proportion of non-events. So we must sample the observations in approximately equal proportions to get better models.

Create Training and Test Samples

One way to address the problem of class bias is to draw the 0's and 1's for the `trainingData` (development sample) in equal proportions. In doing so, we will put rest of the `inputData` not included for training into `testData` (validation sample). As a result, the size of development sample will be smaller that validation, which is okay, because, there are large number of observations (`>10K`).

```
# Create Training Data
 input_ones <- inputData[which(inputData$ABOVE50K == 1), ]  # all 1's
input_zeros <- inputData[which(inputData$ABOVE50K == 0), ]  # all 0's

set.seed(100)  # for repeatability of samples

input_ones_training_rows <- sample(1:nrow(input_ones), 0.7*nrow(input_ones))  # 1's for tra
input_zeros_training_rows <- sample(1:nrow(input_zeros), 0.7*nrow(input_ones))  # 0's for t

# Pick as many 0's as 1's
training_ones <- input_ones[input_ones_training_rows, ]
training_zeros <- input_zeros[input_zeros_training_rows, ]
trainingData <- rbind(training_ones, training_zeros)  # row bind the 1's and 0's

# Create Test Data
 test_ones <- input_ones[-input_ones_training_rows, ]
test_zeros <- input_zeros[-input_zeros_training_rows, ]

testData <- rbind(test_ones, test_zeros)  # row bind the 1's and 0's
```

Build Logit Models and Predict

```
logitMod <- glm(ABOVE50K ~ RELATIONSHIP + AGE + CAPITALGAIN + OCCUPATION + EDUCATIONNUM, dat
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
predicted <- plogis(predict(logitMod, testData))  # predicted scores
# or
predicted <- predict(logitMod, testData, type="response")  # predicted scores
```

When we use the predict function on this model, it will predict the log(odds) of the Y variable. To convert it into prediction probability scores that is bound between 0 and 1, we use the `plogis()`.

Decide on optimal prediction probability cutoff for the model

The default cutoff prediction probability score is 0.5 or the ratio of 1's and 0's in the training data. But sometimes, tuning the probability cutoff can improve the accuracy in both the development and validation samples. The `InformationValue::optimalCutoff` function provides ways to find the optimal cutoff to improve the prediction of 1's, 0's, both 1's and 0's and o reduce the misclassification error. Lets compute the optimal score that minimizes the misclassification error for the above model.

```
library(InformationValue)
optCutOff <- optimalCutoff(testData$ABOVE50K, predicted)[1]
optCutOff
```

```
## [1] 0.89
```

Misclassification Error

Misclassification error is the percentage mismatch of predcited vs actuals, irrespective of 1's or 0's. The lower the misclassification error, the better is your model.

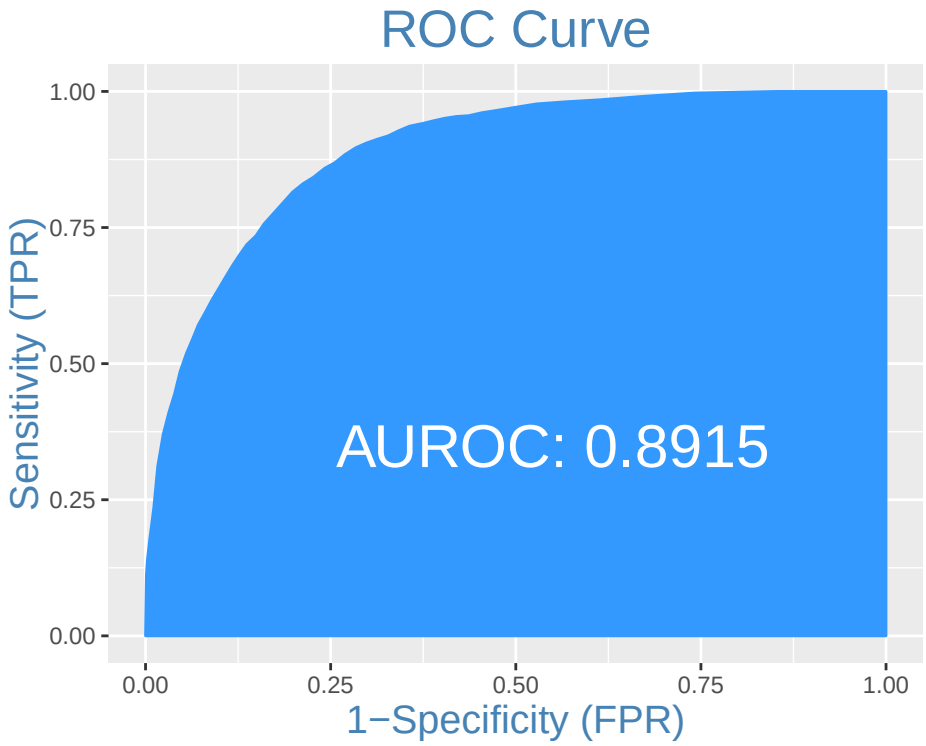
```
misClassError(testData$ABOVE50K, predicted, threshold = optCutOff)
```

```
## [1] 0.0892
```

ROC

Receiver Operating Characteristics Curve traces the percentage of true positives accurately predicted by a given logit model as the prediction probability cutoff is lowered from 1 to 0. For a good model, as the cutoff is lowered, it should mark more of actual 1's as positives and lesser of actual 0's as 1's. So for a good model, the curve should rise steeply, indicating that the TPR (Y-Axis) increases faster than the FPR (X-Axis) as the cutoff score decreases. Greater the area under the ROC curve, better the predictive ability of the model.

```
plotROC(testData$ABOVE50K, predicted)
```



Specificity and Sensitivity

Sensitivity (or `True Positive Rate`) is the percentage of 1's (actuals) correctly predicted by the model, while, **specificity** is the percentage of 0's (actuals) correctly predicted. **Specificity** can also be calculated as `1-False Positive Rate`.

$$\text{Sensitivity} = \frac{\#\text{Actual 1's and Predicted as 1's}}{\#\text{of Actual 1's}}$$

$$\text{Specificity} = \frac{\#\text{Actual 0's and Predicted as 0's}}{\#\text{of Actual 0's}}$$

```
sensitivity(testData$ABOVE50K, predicted, threshold = optCutOff)
```

```
## [1] 0.3442414
```

```
specificity(testData$ABOVE50K, predicted, threshold = optCutOff)
```

```
## [1] 0.9800853
```

The above numbers are calculated on the validation sample that was not used for training the model. So, a truth detection rate of 34.42% on test data is good.

Confusion Matrix The columns are actuals, while rows are predicteds.

```
confusionMatrix(testData$ABOVE50K, predicted, threshold = optCutOff)
```

```
##       0    1
## 0 18849 1543
## 1   383  810
```

3.2 Example: Titanic survivors data

The dataset is a collection of data about some of the passengers, and the goal is to predict the survival (either 1 if the passenger survived or 0 if they did not) based on some features such as the class of service, the sex, the age etc. As you can see, we are going to use both categorical and continuous variables.

```
VARIABLE DESCRIPTIONS:
pclass          Passenger Class
                (1 = 1st; 2 = 2nd; 3 = 3rd)
survival        Survival
                (0 = No; 1 = Yes)
name            Name
sex             Sex
age             Age
sibsp           Number of Siblings/Spouses Aboard
parch           Number of Parents/Children Aboard
ticket          Ticket Number
fare            Passenger Fare
cabin           Cabin
embarked        Port of Embarkation
                (C = Cherbourg; Q = Queenstown; S = Southampton)
boat            Lifeboat
body            Body Identification Number
home_dest       Home/Destination
```

Full description of data set

Download data here

Read `train` and `test` set

```
train <- read.csv('data/titanic_train.csv',header=TRUE,row.names=1)
 test <- read.csv('data/titanic_test.csv',header=TRUE,row.names=1)
```

3.2.0.1 Questions:

- Fit a logistic model with `pclass` as exploratory variable. What is the interpretation of the fitted model?
- Find the best possible logistic regression model based on all the available variables.

```
model <- glm(Survived ~.,family=binomial(link='logit'),data=train)
summary(model)
```

```
##
## Call:
## glm(formula = Survived ~ ., family = binomial(link = "logit"),
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6064  -0.5954  -0.4254   0.6220   2.4165
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.137627   0.594998   8.635  < 2e-16 ***
## Pclass      -1.087156   0.151168  -7.192 6.40e-13 ***
## Sexmale     -2.756819   0.212026 -13.002  < 2e-16 ***
## Age         -0.037267   0.008195  -4.547 5.43e-06 ***
## SibSp       -0.292920   0.114642  -2.555   0.0106 *
## Parch       -0.116576   0.128127  -0.910   0.3629
## Fare         0.001528   0.002353   0.649   0.5160
## EmbarkedQ   -0.002656   0.400882  -0.007   0.9947
## EmbarkedS   -0.318786   0.252960  -1.260   0.2076
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1065.39  on 799  degrees of freedom
## Residual deviance:  709.39  on 791  degrees of freedom
## AIC: 727.39
##
## Number of Fisher Scoring iterations: 5
```

```
anova(model, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
```

```
##
## Response: Survived
##
## Terms added sequentially (first to last)
##
##
##          Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                       799    1065.39
## Pclass    1   83.607       798     981.79 < 2.2e-16 ***
## Sex       1  240.014       797     741.77 < 2.2e-16 ***
## Age       1   17.495       796     724.28 2.881e-05 ***
## SibSp     1   10.842       795     713.43  0.000992 ***
## Parch     1    0.863       794     712.57  0.352873
## Fare      1    0.994       793     711.58  0.318717
## Embarked  2    2.187       791     709.39  0.334990
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
mod1 <- glm(Survived ~ as.factor(Pclass), family=binomial, data=train)
summary(mod1)
```

```
##
## Call:
## glm(formula = Survived ~ as.factor(Pclass), family = binomial,
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.3787  -0.7515  -0.7515   0.9887   1.6747
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)          0.4616     0.1474   3.131  0.00174 **
## as.factor(Pclass)2  -0.5455     0.2138  -2.551  0.01074 *
## as.factor(Pclass)3  -1.5816     0.1844  -8.575  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1065.39  on 799  degrees of freedom
## Residual deviance:  979.94  on 797  degrees of freedom
## AIC: 985.94
##
## Number of Fisher Scoring iterations: 4
```

```
anova(mod1,test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Survived
##
## Terms added sequentially (first to last)
##
##
##                   Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                                799    1065.39
## as.factor(Pclass)  2   85.452       797     979.94 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

interaction effect between passenger class and sex, as passenger class showed a much bigger difference in survival rate amongst the women compared to the men (i.e. Higher class women were much more likely to survive than lower class women, whereas first class Men were more likely to survive than 2nd or 3rd class men, but not by the same margin as amongst the women).

```
mod2 <- glm(Survived ~ Pclass + Sex  + Age + SibSp, family = binomial(logit), data = train)
summary(mod2)
```

```
##
## Call:
## glm(formula = Survived ~ Pclass + Sex + Age + SibSp, family = binomial(logit),
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6595  -0.6125  -0.4247   0.6149   2.4302
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.05604    0.50130  10.086  < 2e-16 ***
## Pclass      -1.14391    0.12585  -9.089  < 2e-16 ***
## Sexmale     -2.75564    0.20471 -13.461  < 2e-16 ***
## Age         -0.03725    0.00812  -4.588 4.48e-06 ***
## SibSp       -0.33075    0.10892  -3.037  0.00239 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1065.39  on 799  degrees of freedom
## Residual deviance:  713.43  on 795  degrees of freedom
## AIC: 723.43
##
## Number of Fisher Scoring iterations: 5
```

```
anova(mod2,test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Survived
##
## Terms added sequentially (first to last)
##
##
##        Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                     799    1065.39
## Pclass  1   83.607       798     981.79 < 2.2e-16 ***
## Sex     1  240.014       797     741.77 < 2.2e-16 ***
## Age     1   17.495       796     724.28 2.881e-05 ***
## SibSp   1   10.842       795     713.43  0.000992 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
mod3 <-  glm(Survived ~ Pclass + Sex + Pclass:Sex + Age + SibSp, family = binomial(logit), d
summary(mod3)
```

```
##
## Call:
## glm(formula = Survived ~ Pclass + Sex + Pclass:Sex + Age + SibSp,
##     family = binomial(logit), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1993  -0.6265  -0.4770   0.4485   2.3093
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)     7.606411   0.960804   7.917 2.44e-15 ***
```

```
## Pclass         -2.108360   0.316024  -6.672 2.53e-11 ***
## Sexmale        -5.887480   0.920417  -6.397 1.59e-10 ***
## Age            -0.038063   0.008498  -4.479 7.50e-06 ***
## SibSp          -0.310269   0.109370  -2.837 0.004556 **
## Pclass:Sexmale  1.254202   0.338241   3.708 0.000209 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1065.39  on 799  degrees of freedom
## Residual deviance:  695.66  on 794  degrees of freedom
## AIC: 707.66
##
## Number of Fisher Scoring iterations: 6
```

```
anova(mod3,test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Survived
##
## Terms added sequentially (first to last)
##
##
##            Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                         799    1065.39
## Pclass      1   83.607       798     981.79 < 2.2e-16 ***
## Sex         1  240.014       797     741.77 < 2.2e-16 ***
## Age         1   17.495       796     724.28 2.881e-05 ***
## SibSp       1   10.842       795     713.43  0.000992 ***
## Pclass:Sex  1   17.779       794     695.66 2.481e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the steps above, we briefly evaluated the fitting of the model, now we would like to see how the model is doing when predicting y on a new set of data. By setting the parameter `type='response'`, `R` will output probabilities in the form of $P(y = 1|X)$. Our decision boundary will be 0.5. If $P(y = 1|X) > 0.5$ then $y = 1$ otherwise $y = 0$. Note that for some applications different decision boundaries could be a better option.

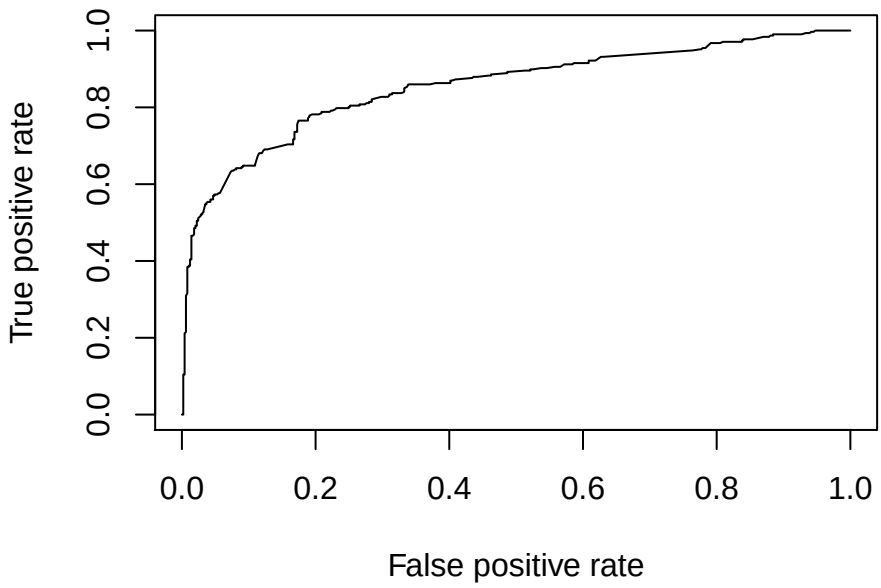
```
fitted.results <- predict(mod3,newdata=test,type='response')
fitted.results <- ifelse(fitted.results > 0.5,1,0)

misClasificError <- mean(fitted.results != test$Survived)
print(paste('Accuracy',1-misClasificError))
```

```
## [1] "Accuracy 0.8075"
```

The 0.8075 accuracy on the test set is quite a good result. However, keep in mind that this result is somewhat dependent on the manual split of the data that I made earlier, therefore if you wish for a more precise score, you would be better off running some kind of cross validation such as k-fold cross validation.

Evaluate predictive performance

```
library(ROCR)
p <- predict(mod3, newdata=subset(test,select=c(2,3,4,5,6,7,8)), type="response")
pr <- prediction(p, test$Survived)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```




```
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 0.8543155
```

4 Multivariate Analysis

4.1 Principal Components Analysis

Principal component analysis (PCA) is a technique used to emphasize variation and bring out strong patterns in a dataset. It's often used to make data easy to explore and visualize. We are going to carry out a major principal component analysis of the results obtained in the women's heptathlon competition at the Olympic Games in Seoul (1988).

```
library(HSAUR2)
```

```
## Loading required package: tools
```

```
data("heptathlon")
head(heptathlon)
```

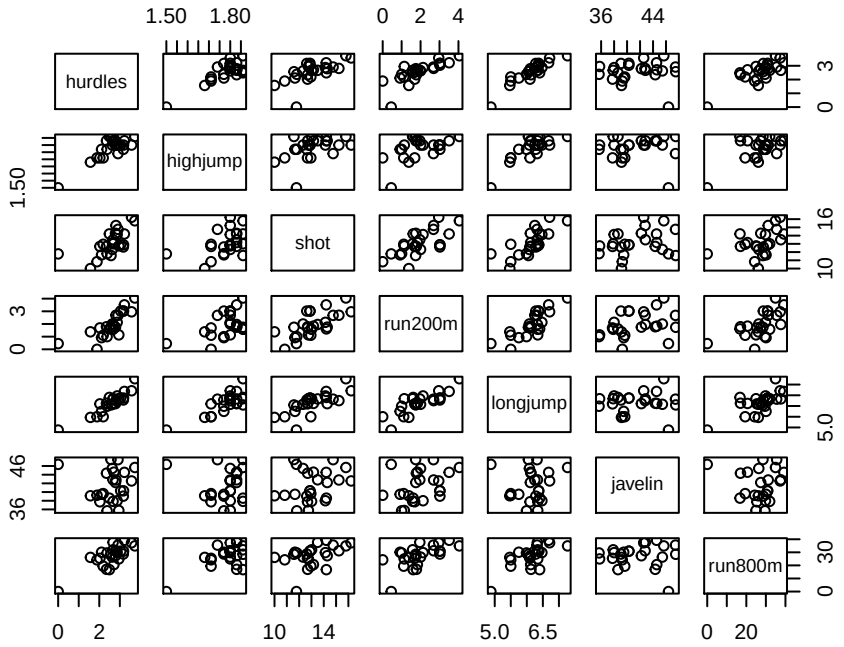
```
##                      hurdles highjump  shot run200m longjump javelin
## Joyner-Kersee (USA)    12.69     1.86 15.80   22.56     7.27   45.66
## John (GDR)             12.85     1.80 16.23   23.65     6.71   42.56
## Behmer (GDR)           13.20     1.83 14.20   23.10     6.68   44.54
## Sablovskaite (URS)     13.61     1.80 15.23   23.92     6.25   42.78
## Choubenkova (URS)      13.51     1.74 14.76   23.93     6.32   47.46
## Schulz (GDR)           13.75     1.83 13.50   24.65     6.33   42.82
##                      run800m score
## Joyner-Kersee (USA)   128.51  7291
## John (GDR)            126.12  6897
## Behmer (GDR)          124.20  6858
## Sablovskaite (URS)    132.24  6540
## Choubenkova (URS)     127.90  6540
## Schulz (GDR)          125.79  6411
```

We re-coded the tests relating to 3 races `hurdles`, `run200m` and `run800m`, subtracting the highest value in each race, each of the 35 athletes' times.

```
heptathlon$hurdles <- max(heptathlon$hurdles) - heptathlon$hurdles
heptathlon$run200m <- max(heptathlon$run200m) - heptathlon$run200m
heptathlon$run800m <- max(heptathlon$run800m) - heptathlon$run800m
```

Dispersion diagram

```
score <- which(colnames(heptathlon) == "score")
plot(heptathlon[,-score])
```



Correlation Matrix

```
round(cor(heptathlon[,-score]),3)
```

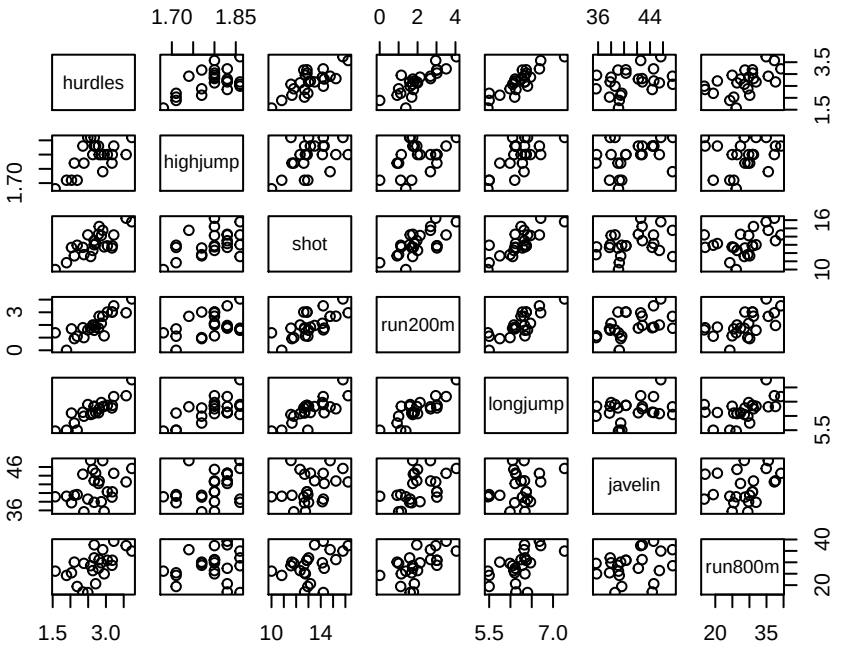
```
##          hurdles highjump  shot run200m longjump javelin run800m
## hurdles    1.000    0.811 0.651   0.774    0.912   0.008   0.779
## highjump   0.811    1.000 0.441   0.488    0.782   0.002   0.591
## shot       0.651    0.441 1.000   0.683    0.743   0.269   0.420
## run200m    0.774    0.488 0.683   1.000    0.817   0.333   0.617
## longjump   0.912    0.782 0.743   0.817    1.000   0.067   0.700
## javelin    0.008    0.002 0.269   0.333    0.067   1.000  -0.020
## run800m    0.779    0.591 0.420   0.617    0.700  -0.020   1.000
```

Result matrix confirms that the vast majority of correlations between the tests are positive, with a high correlation between the long jump (`longjump`) and the 100m hurdles (`hurdles`). Some less like the high jump (`highjump`) and the shot (`shot`) and the javelin (`javelin`) which has a close to zero correlation with the rest of the tests.

A possible explanation for this result may be that training for the other 6 tests does not add too much to the javelin test, which is a more technical test.

It can be observed that there is an atypical value in almost all tests that corresponds to an athlete' `Launa (PNG)` from Papua New Guinea - we will eliminate this observation to see if the correlation matrix is significantly different:

```
heptathlon <- heptathlon[-which(rownames(heptathlon)=="Launa (PNG)"),]
plot(heptathlon[,-score])
```



Eliminating the Papua New Guinea athlete, correlations change substantially and in the matrix scatter diagram, no extreme values are observed.

```
round(cor(heptathlon[,-score]),3)
```

```
##          hurdles highjump  shot run200m longjump javelin run800m
## hurdles    1.000    0.582 0.767   0.830    0.889   0.332   0.559
## highjump   0.582    1.000 0.465   0.391    0.663   0.348   0.152
## shot       0.767    0.465 1.000   0.669    0.784   0.343   0.408
## run200m    0.830    0.391 0.669   1.000    0.811   0.471   0.573
## longjump   0.889    0.663 0.784   0.811    1.000   0.287   0.523
## javelin    0.332    0.348 0.343   0.471    0.287   1.000   0.256
## run800m    0.559    0.152 0.408   0.573    0.523   0.256   1.000
```

To make the PCA, we will start from the correlation matrix, since the 7 tests are measured in different scales (meters, seconds). This procedure is called normalized PCA (`scale=TRUE' in the`prcomp‘ function).

```
?prcomp
heptathlon_pca <- prcomp(heptathlon[,-score],scale=TRUE)
head(heptathlon_pca,5)
```

```
## $sdev
## [1] 2.0793370 0.9481532 0.9109016 0.6831967 0.5461888 0.3374549 0.2620420
##
## $rotation
##                  PC1         PC2        PC3         PC4         PC5
## hurdles  -0.4503876  0.05772161 -0.1739345  0.04840598 -0.19889364
## highjump -0.3145115 -0.65133162 -0.2088272 -0.55694554  0.07076358
## shot     -0.4024884 -0.02202088 -0.1534709  0.54826705  0.67166466
## run200m  -0.4270860  0.18502783  0.1301287  0.23095946 -0.61781764
## longjump -0.4509639 -0.02492486 -0.2697589 -0.01468275 -0.12151793
## javelin  -0.2423079 -0.32572229  0.8806995  0.06024757  0.07874396
## run800m  -0.3029068  0.65650503  0.1930020 -0.57418128  0.31880178
##                  PC6         PC7
## hurdles   0.84665086 -0.06961672
## highjump -0.09007544  0.33155910
## shot     -0.09886359  0.22904298
## run200m  -0.33279359  0.46971934
## longjump -0.38294411 -0.74940781
## javelin   0.07193437 -0.21108138
## run800m  -0.05217664  0.07718616
##
## $center
##   hurdles  highjump      shot   run200m  longjump   javelin   run800m
##  2.687500  1.793750 13.173333  2.023750  6.205417 41.278333 28.516667
##
## $scale
##    hurdles   highjump       shot    run200m   longjump    javelin
## 0.51456398 0.05232112 1.49714995 0.93676972 0.40165938 3.46870690
##    run800m
## 6.14724800
##
## $x
##                               PC1         PC2          PC3          PC4
## Joyner-Kersee (USA) -4.757530189 -0.13986143 -0.006040526  0.293416339
## John (GDR)          -3.147943402  0.94859029 -0.243919842  0.549171385
## Behmer (GDR)        -2.926184760  0.69534239  0.622293440 -0.554744912
## Sablovskaite (URS)  -1.288135516  0.17900713  0.250632380  0.637174187
## Choubenkova (URS)   -1.503450994  0.96177329  1.780588549  0.784035325
## Schulz (GDR)        -0.958467101  0.35121643  0.413086366 -1.113546938
## Fleming (AUS)       -0.953445060  0.49982537 -0.265135015 -0.140202490
## Greiner (USA)       -0.633239267  0.37592917 -1.140338594  0.142558348
```

```
## Lajbnerova (CZE)    -0.381571974 -0.71213459 -0.068395353  0.087212735
## Bouraga (URS)       -0.522322004  0.77688861 -0.481071429  0.283745698
## Wijnsma (HOL)       -0.217701500 -0.23369645 -1.154221444 -1.260128609
## Dimitrova (BUL)     -1.075984276  0.51552998 -0.312458252 -0.127032432
## Scheider (SWI)       0.003014986 -1.44688825  1.582739069 -1.254415325
## Braun (FRG)          0.109183759 -1.63595645  0.469577294  0.362580442
## Ruotsalainen (FIN)   0.208868056 -0.68866173  1.152140223 -0.112914470
## Yuping (CHN)         0.232507119 -1.95999641 -1.541230813  0.598325122
## Hagger (GB)          0.659520046 -0.08775813 -1.796509771 -0.182375000
## Brown (USA)          0.756854602 -2.04292201  0.451506018  0.476926314
## Mulliner (GB)        1.880932819  0.91530324 -0.359311801  0.799619094
## Hautenauve (BEL)     1.828170404  0.72629699 -1.048640439 -0.711793161
## Kytola (FIN)         2.118203163  0.39921397  0.190158154 -0.788445056
## Geremias (BRA)       2.770706272  0.03463584  0.170274969  1.385562494
## Hui-Ing (TAI)        3.901166920  1.20175472  0.943677497 -0.002429122
## Jeong-Mi (KOR)       3.896847898  0.36656804  0.390599321 -0.152299968
##                             PC5         PC6         PC7
## Joyner-Kersee (USA) -0.36183307 -0.27050283 -0.47587527
## John (GDR)           0.75364464  0.37770017 -0.05172711
## Behmer (GDR)        -0.19035037 -0.25780287  0.11054960
## Sablovskaite (URS)   0.60362153 -0.21575716  0.53075152
## Choubenkova (URS)    0.58969949  0.08014332 -0.30081842
## Schulz (GDR)         0.71483887 -0.25436956  0.03838796
## Fleming (AUS)       -0.86581530  0.03691813  0.23005943
## Greiner (USA)        0.20807431 -0.14236240 -0.06374657
## Lajbnerova (CZE)     0.67727618  0.25014881  0.35555639
## Bouraga (URS)       -1.18784299  0.39881271  0.19712215
## Wijnsma (HOL)        0.37497195 -0.20267731  0.17459647
## Dimitrova (BUL)     -0.91992929  0.26727067  0.21111846
## Scheider (SWI)      -0.20526249  0.17597425 -0.03915701
## Braun (FRG)         -0.14712208  0.26134199 -0.01334416
## Ruotsalainen (FIN)  -0.31539746  0.18351622 -0.14127555
## Yuping (CHN)         0.17451428 -0.50175724  0.04999374
## Hagger (GB)         -0.05104049  0.55058471 -0.46388534
## Brown (USA)         -0.38154294 -0.26606429 -0.11099445
## Mulliner (GB)       -0.06942955 -0.73259727 -0.31281502
## Hautenauve (BEL)     0.14092347  0.06933542 -0.07548638
## Kytola (FIN)         0.41815113 -0.03363651  0.12143219
## Geremias (BRA)       0.28541366  0.38083979  0.34574480
## Hui-Ing (TAI)       -0.67080776 -0.52756760  0.09436975
## Jeong-Mi (KOR)       0.42524426  0.37250885 -0.41055719
```

```
a1 <- heptathlon_pca$rotation[,1]
```

summary

```
summary(heptathlon_pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     2.0793 0.9482 0.9109 0.68320 0.54619 0.33745
## Proportion of Variance 0.6177 0.1284 0.1185 0.06668 0.04262 0.01627
## Cumulative Proportion  0.6177 0.7461 0.8646 0.93131 0.97392 0.99019
##                           PC7
## Standard deviation     0.26204
## Proportion of Variance 0.00981
## Cumulative Proportion  1.00000
```

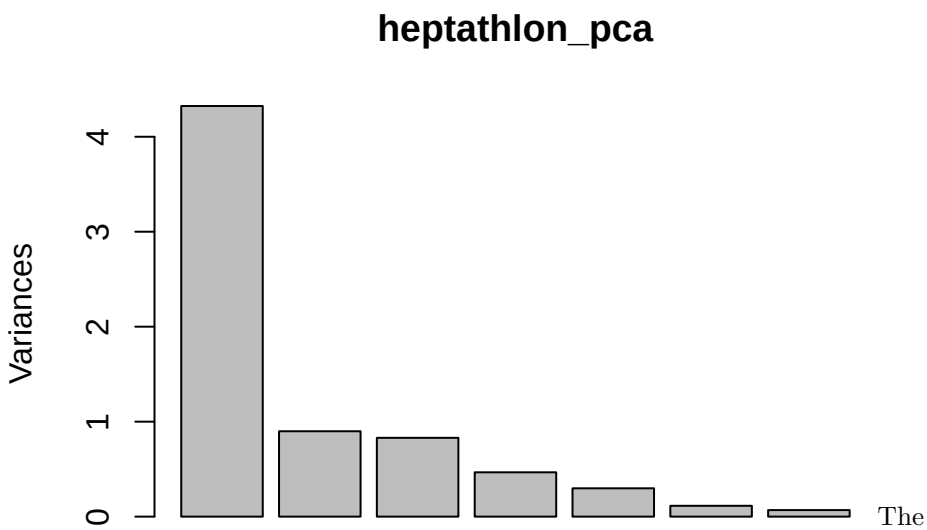
Check the explained variability

```
head(heptathlon_pca$x)
```

```
##                            PC1        PC2          PC3        PC4
## Joyner-Kersee (USA) -4.7575302 -0.1398614 -0.006040526  0.2934163
## John (GDR)          -3.1479434  0.9485903 -0.243919842  0.5491714
## Behmer (GDR)        -2.9261848  0.6953424  0.622293440 -0.5547449
## Sablovskaite (URS)  -1.2881355  0.1790071  0.250632380  0.6371742
## Choubenkova (URS)   -1.5034510  0.9617733  1.780588549  0.7840353
## Schulz (GDR)        -0.9584671  0.3512164  0.413086366 -1.1135469
##                            PC5         PC6         PC7
## Joyner-Kersee (USA) -0.3618331 -0.27050283 -0.47587527
## John (GDR)           0.7536446  0.37770017 -0.05172711
## Behmer (GDR)        -0.1903504 -0.25780287  0.11054960
## Sablovskaite (URS)   0.6036215 -0.21575716  0.53075152
## Choubenkova (URS)    0.5896995  0.08014332 -0.30081842
## Schulz (GDR)         0.7148389 -0.25436956  0.03838796
```

Graphically, it is observed that the first major component is the most dominant.

```
plot(heptathlon_pca)
```



The *biplot* is a graphical representation of multivariate data. Just as a scatter plot shows the combined distribution of two variables, a **biplot** represents three or more variables.

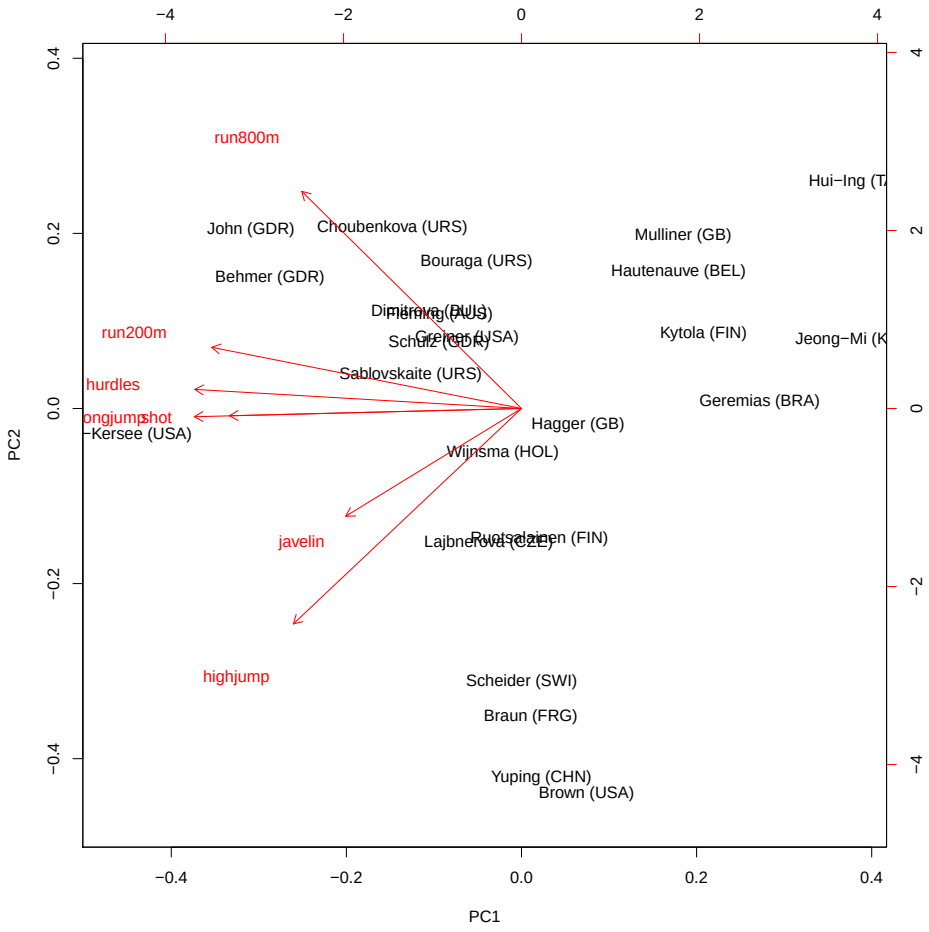
If we order from major to minor the variable `score` we have the three gold, silver and bronze medals.

```
head(heptathlon[order(heptathlon$score,decreasing = TRUE),],3)
```

```
##                     hurdles highjump  shot run200m longjump javelin
## Joyner-Kersee (USA)    3.73     1.86 15.80    4.05     7.27   45.66
## John (GDR)             3.57     1.80 16.23    2.96     6.71   42.56
## Behmer (GDR)           3.22     1.83 14.20    3.51     6.68   44.54
##                     run800m score
## Joyner-Kersee (USA)   34.92  7291
## John (GDR)            37.31  6897
## Behmer (GDR)          39.23  6858
```

The biplot, shows us the athletes projected on their first 2 main components, but also the arrows give us information on the variances and covariances of the variables (addresses of maximum variability).

```
biplot(heptathlon_pca)
```



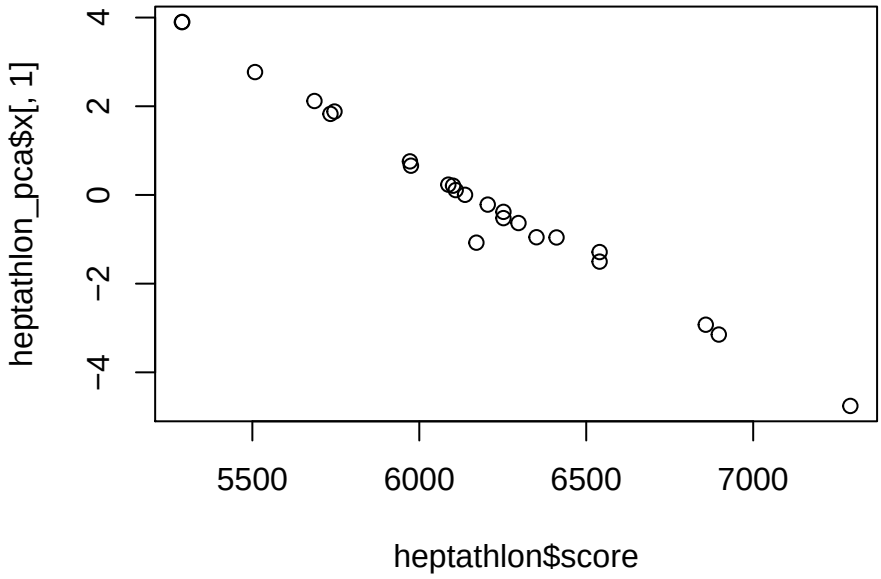
For example, the winner of `Joyner-Kersee (USA)` accumulates higher scores in the `longjump`,`hurdlesp` and `run200m` tests.

Podemos analizar la correlación entre la variable `score` y la PC1. Lo cual indica que la correlación es muy negativa y muy fuerte con el `score`.

```
cor(heptathlon$score, heptathlon_pca$x[,1])
```

```
## [1] -0.9931168
```

```
plot(heptathlon$score, heptathlon_pca$x[,1])
```



prcomp object

```
class(heptathlon_pca)
```

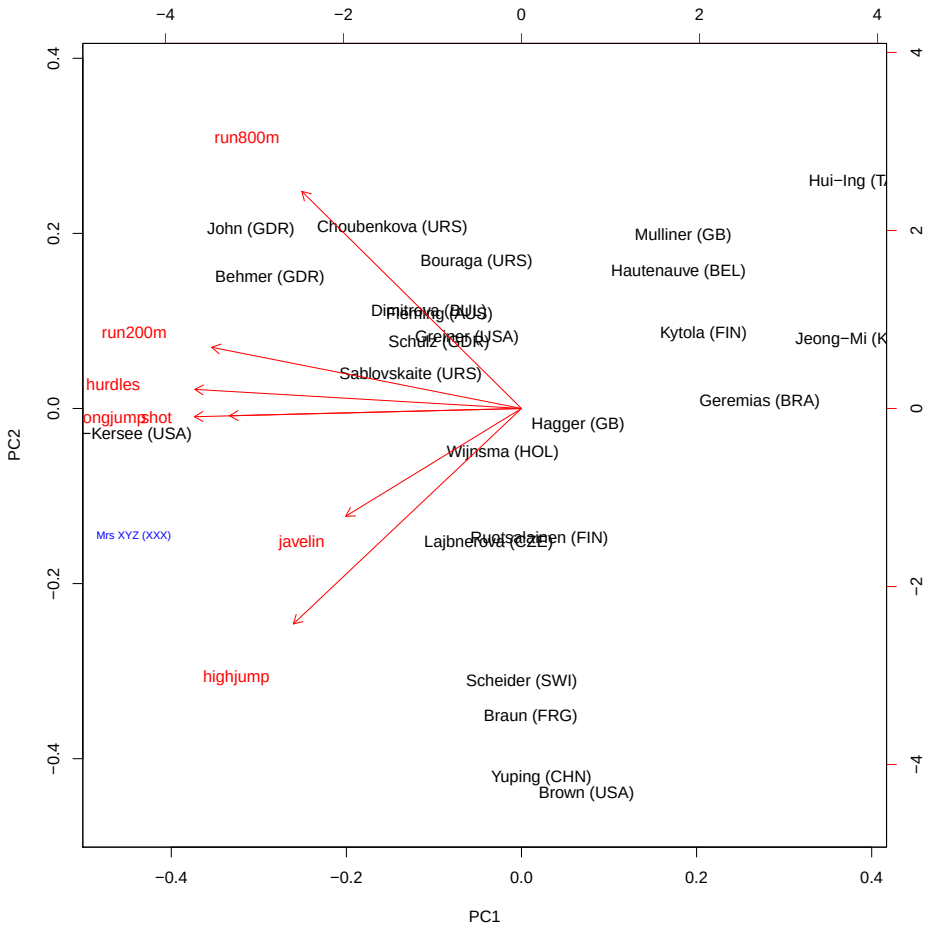
```
## [1] "prcomp"
```

we can use predict, e.g.:

```
new.athlete <- as.data.frame(t(as.vector(c(3.5,2,13,5,7,41,33))))
colnames(new.athlete) <- c("hurdles","highjump",
                           "shot","run200m","longjump",
                           "javelin","run800m")
rownames(new.athlete) <- "Mrs XYZ (XXX)"
pp<-predict(heptathlon_pca,newdata = new.athlete)
pp
```

```
##                      PC1       PC2       PC3       PC4       PC5        PC6
## Mrs XYZ (XXX) -4.354879 -1.430366 -1.130194 -1.901377 -2.089963 -0.8654821
##                     PC7
## Mrs XYZ (XXX) 1.253643
```

```
biplot(heptathlon_pca)
text(pp[,1],pp[,2],"Mrs XYZ (XXX)",col="blue",cex=.65)
```



4.2 K-means

K-means Clustering is an unsupervised learning algorithm that tries to cluster data based on their similarity. Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to find patterns in the data. In k-means clustering, we have to specify the number of clusters we want the data to be grouped into. The algorithm randomly assigns each observation to a cluster, and finds the centroid of each cluster. Then, the algorithm iterates through two steps:

1. Reassign data points to the cluster whose centroid is closest.
2. Calculate new centroid of each cluster.

These two steps are repeated till the within cluster variation cannot be reduced

any further. The within cluster variation is calculated as the sum of the euclidean distance between the data points and their respective cluster centroids.

Example:

```
library(rattle)
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
data(wine)
head(wine)
```

```
##   Type Alcohol Malic  Ash Alcalinity Magnesium Phenols Flavanoids
## 1    1   14.23  1.71 2.43       15.6       127    2.80       3.06
## 2    1   13.20  1.78 2.14       11.2       100    2.65       2.76
## 3    1   13.16  2.36 2.67       18.6       101    2.80       3.24
## 4    1   14.37  1.95 2.50       16.8       113    3.85       3.49
## 5    1   13.24  2.59 2.87       21.0       118    2.80       2.69
## 6    1   14.20  1.76 2.45       15.2       112    3.27       3.39
##   Nonflavanoids Proanthocyanins Color  Hue Dilution Proline
## 1          0.28            2.29  5.64 1.04     3.92    1065
## 2          0.26            1.28  4.38 1.05     3.40    1050
## 3          0.30            2.81  5.68 1.03     3.17    1185
## 4          0.24            2.18  7.80 0.86     3.45    1480
## 5          0.39            1.82  4.32 1.04     2.93     735
## 6          0.34            1.97  6.75 1.05     2.85    1450
```

It is always recommended to standardize the variables

```
wine.stand <- scale(wine[-1])  # To standarize the variables
# K-Means
k.means.fit <- kmeans(wine.stand, 3) # k = 3
attributes(k.means.fit)
```

```
## $names
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
##
## $class
## [1] "kmeans"
```

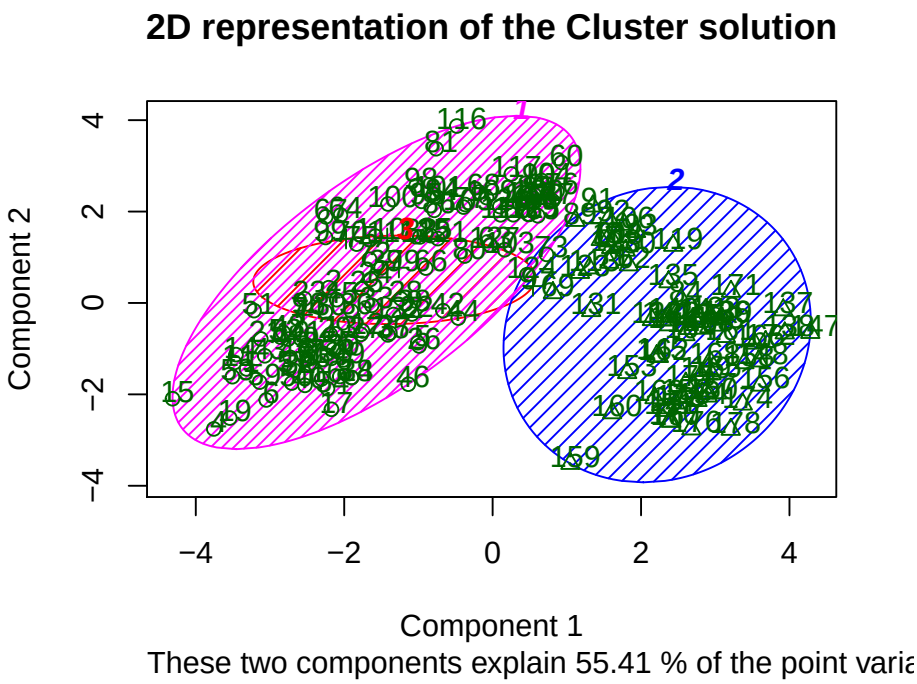
A fundamental question is how to determine the value of parameter k. If we consider the percentage of variance explained as a function of the number of groups:

One must choose a number of groups so that adding another group does not give much better modeling of the data. More precisely, if the percentage of variance explained by the clusters is traced according to the number of bunches, the first clusters will add a lot of information (they explain a lot of variance), but at some point the marginal gain will decrease, giving an angle in the graph. The number of bunches is chosen at this point, hence the "elbow criterion".

```
wssplot <- function(data, nc=15, seed=1234){
  wss <- (nrow(data)-1)*sum(apply(data,2,var))
  for (i in 2:nc){
    set.seed(seed)
    wss[i] <- sum(kmeans(data, centers=i)$withinss)}
  plot(1:nc, wss, type="b", xlab="Number of Clusters",
       ylab="Within groups sum of squares")}

wssplot(wine.stand, nc=6)
```

`library(cluster)` allows for representing the data in two dimensions

```
library(cluster)
clusplot(wine.stand, k.means.fit$cluster,
         main='2D representation of the Cluster solution',
         color=TRUE, shade=TRUE,
         labels=2, lines=0)
```



Knowing that there are three types of `wine$Type` wines, we can calculate the confusion matrix.

```
table(wine$Type)
```

```
##
##  1  2  3
## 59 71 48
```

```
table(wine[,1],k.means.fit$cluster)
```

```
##
##       1  2  3
##    1 59  0  0
##    2 48 17  6
##    3  0 48  0
```

Hierarchical cluster

Hierarchical methods use a distance matrix as an input for the grouping algorithm. The choice of an appropriate metric will influence the form of the clusters, as

some elements may be close together according to a distance and further apart according to another.

```
d <- dist(wine.stand, method = "euclidean") # Euclidean distance matrix.
```

Ward's minimum variance criterion minimizes the total within-cluster variance

```
H.fit <- hclust(d, method="ward")
```

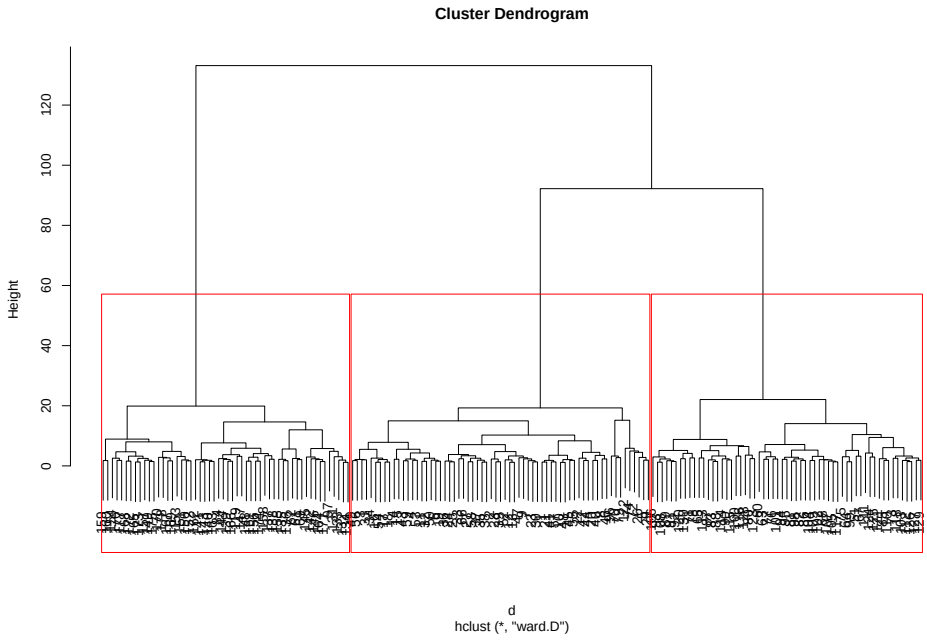
```
## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
```

```
class(H.fit)
```

```
## [1] "hclust"
```

The option `plot` returns a `hclust` showing the dendogram:

```
plot(H.fit) # display dendogram
groups <- cutree(H.fit, k=3) # cut tree into 5 clusters
# draw dendogram with red borders around the 5 clusters
rect.hclust(H.fit, k=3, border="red")
```



```r
table(wine[,1],groups)
```

```
##    groups
##      1  2  3
##   1 58  1  0
##   2  7 58  6
##   3  0  0 48
```