Linear Regression

Dr Liwan Liyanage - Western Sydney University

2 July 2018

Simple Linear Regression

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)
attach(Boston )
names(Boston )
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     '
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   '
```

```
dim(Boston)
```

```
## [1] 506  14
```

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

lm() function

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.

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

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

summary(lm.fit) function

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

```r
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 ***
```

summary(lm.fit) function with no code

```
##
## 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
##
## 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
```

anova(lm.fit) function with no code

```
## Analysis of Variance Table
##
## Response: medv
##            Df Sum Sq Mean Sq F value    Pr(>F)
## lstat       1  23244 23243.9  601.62 < 2.2e-16 ***
## Residuals 504  19472    38.6
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1
```

names() function

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

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

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

lm.fit$coefficients and coef() function

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.

```r
lm.fit$coefficient
```

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

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

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

residuals

```
lm.fit$residuals
```

```
##           1           2           3           4
## -5.822595098 -4.270389786  3.974858016  1.639304221
##           6           7           8           9
## -0.904083746  0.155272588 10.739604245 10.381136279
##          11          12          13          14
## -0.125331595 -3.046685955  2.071434468 -6.306433217
##          16          17          18          19
## -6.606922853 -5.202516132 -3.116616860 -3.247763934
##          21          22          23          24
## -0.983803463 -1.814658317 -1.568916977 -1.166859727
##          26          27          28          29
## -4.968526049 -3.883609950 -3.336988046 -3.993209151
##          31          32          33          34
## -0.382725484 -7.665197306  4.972026713 -4.020435238
##          36          37          38          39
## -6.457363135 -3.713777753 -5.221908047 -0.229840926
```

confint() command

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
```

Confidence intervals for the prediction

The predict() function can be used to produce confidence intervals for the prediction of medv for a given value of lstat. The 95% confidence interval associated with a lstat value of 10 is (24.47, 25.63)

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

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

NOTE:prediction intervals

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

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

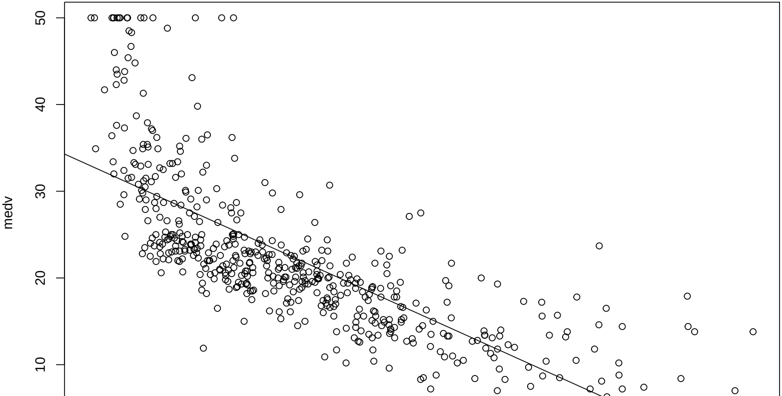
The 95% prediction interval is (12.828, 37.28). As expected, the confidence and prediction intervals are centered around the same point (a predicted value of 25.05 for medv when lstat equals 10).

Note that the latter are substantially wider.

plot() and abline() functions

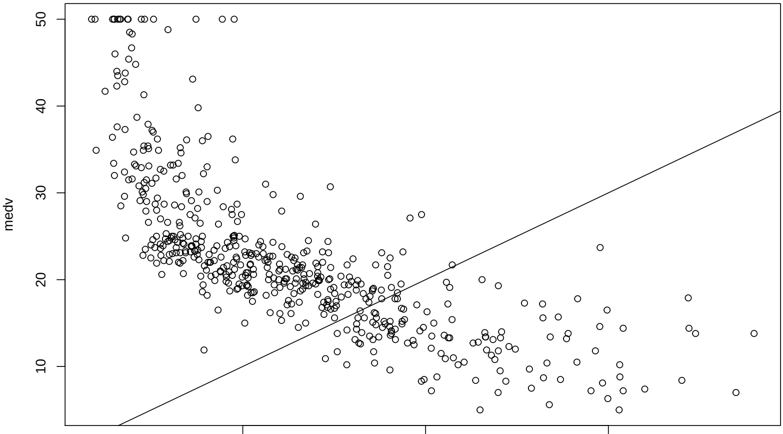
We will now plot medv and lstat along with the least squares regression line using the plot() and abline() functions.

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



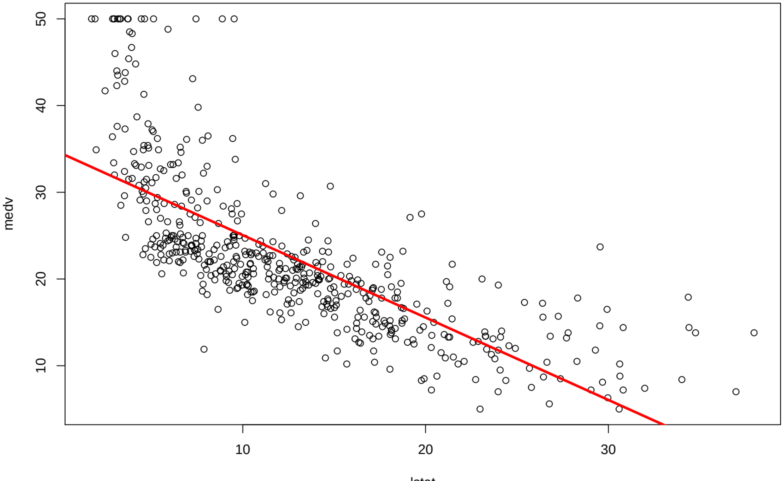
Below we experiment with some additional settings for plotting lines and points

```
plot(lstat ,medv)
abline(a=0,b=1)
```



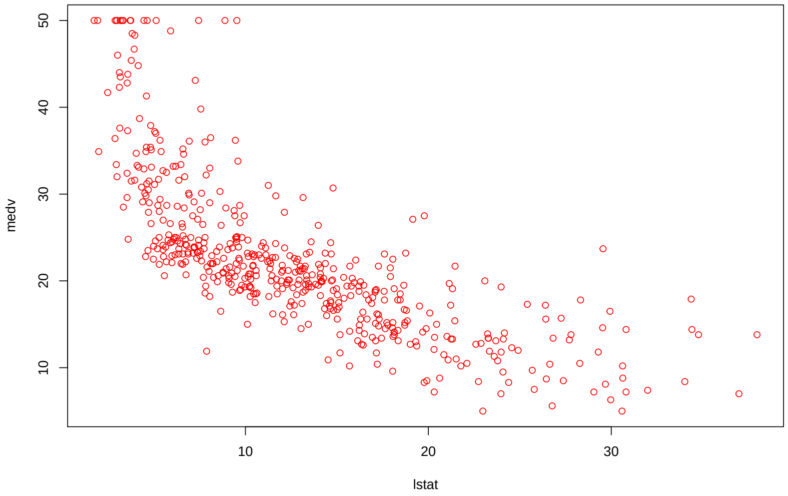
Colour command

```
plot(lstat ,medv)
abline (lm.fit ,lwd =3, col ="red ")
```



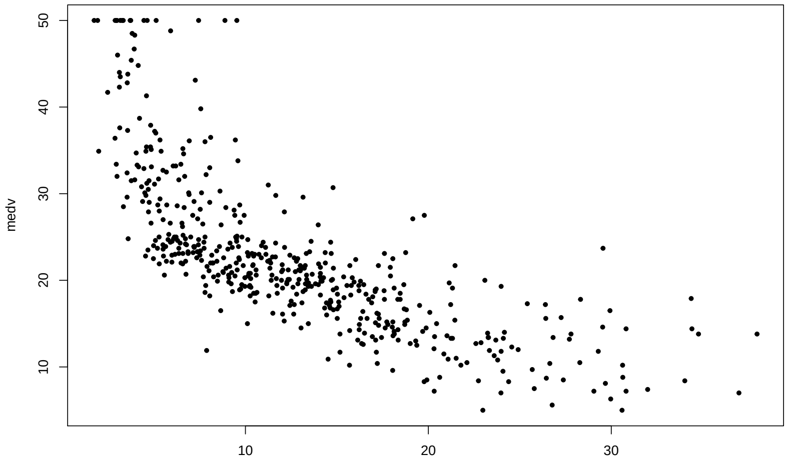
Colouring the observations in the pot

```
plot(lstat ,medv ,col ="red ")
```



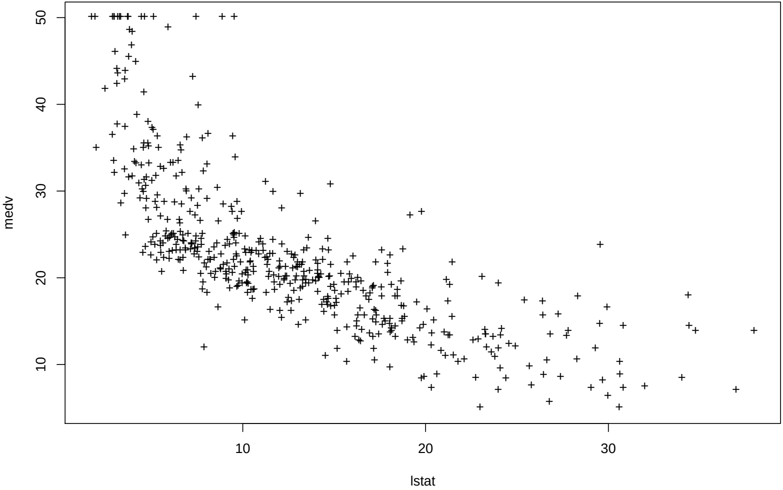
We can also use the pch option to create different plotting symbols

```
plot(lstat ,medv ,pch =20)
```



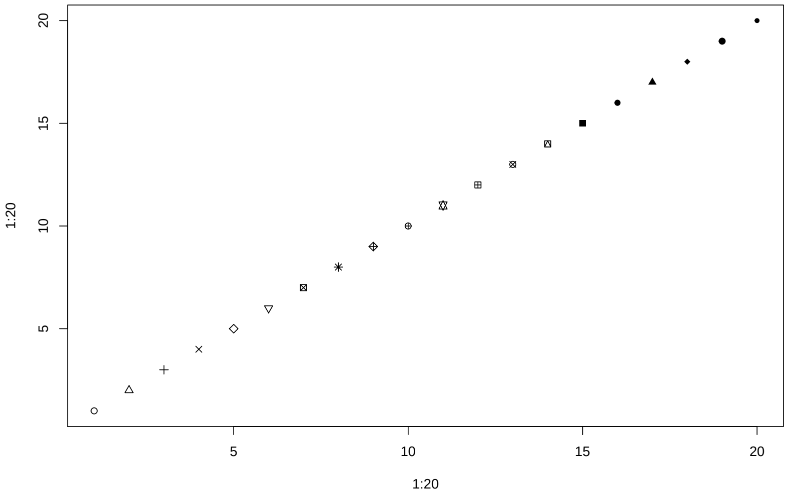
Alternate ploting symbols

```
plot(lstat ,medv ,pch ="+")
```



Varyig ploting symbols

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

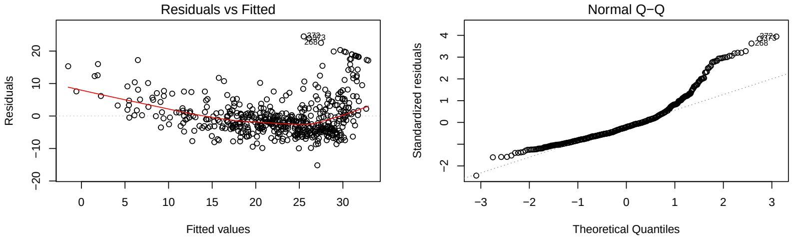


Next we examine some diagnostic plots

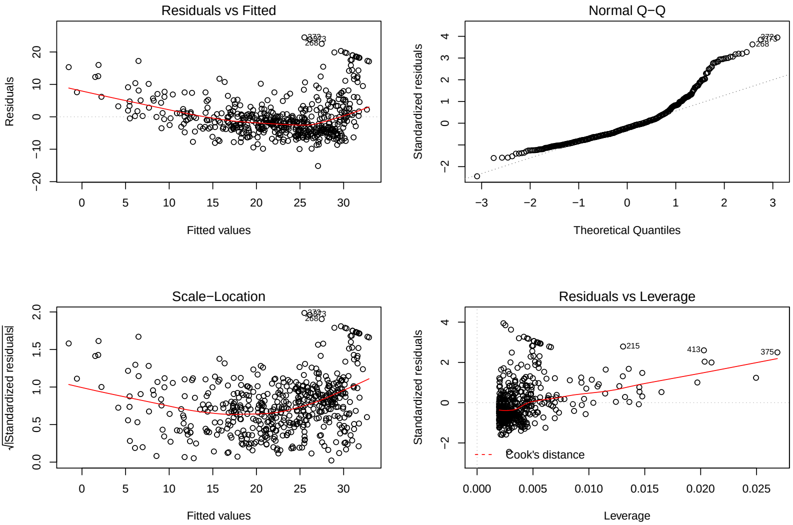
It is convenient to view all four automatic plots together. We can achieve this by using the par() function, which tells R to split the display screen into separate panels so that multiple plots can be viewed simultaneously.

For example, par(mfrow=c(2,2)) divides the plotting region into a 2 $\times$ 2 grid of panels.

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



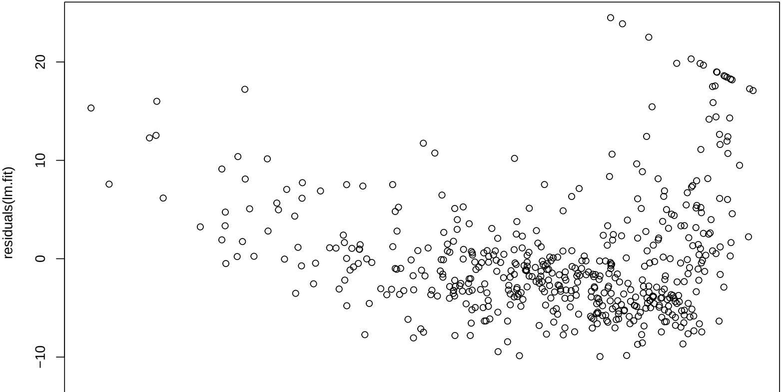
Plots without code



Plot the residuals against the fitted values

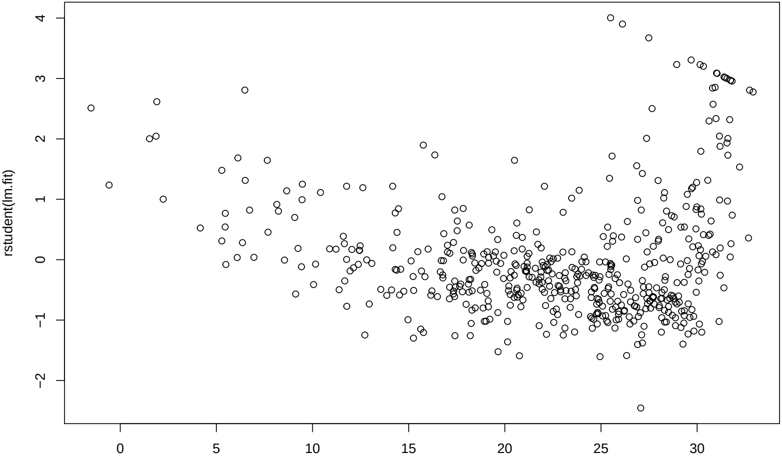
Alternatively, we can compute the residuals from a linear regression fit using the residuals() function and we can use this function to plot the residuals against the fitted values.

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



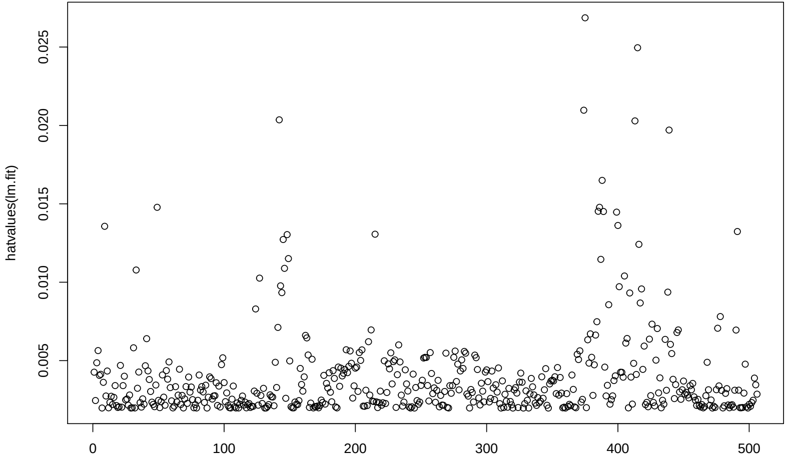
The function rstudent() will return the studentized residuals,

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



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

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



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.

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

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

Predicted values

```r
predict(lm.fit)
```

```
##         1         2         3         4         5
## 29.8225951 25.8703898 30.7251420 31.7606958 29.4900778 2
##         7         8         9        10        11
## 22.7447274 16.3603958  6.1188637 18.3079969 15.1253316 2
##        13        14        15        16        17
## 19.6285655 26.7064332 24.8063345 26.5069229 28.3025161 2
##        19        20        21        22        23
## 23.4477639 23.8372842 14.5838035 21.4146583 16.7689170 1
##        25        26        27        28        29
## 19.0680364 18.8685260 20.4836100 18.1369880 22.3932092 2
##        31        32        33        34        35
## 13.0827255 22.1651973  8.2279733 17.1204352 15.2298370 2
##        37        38        39        40        41
## 23.7137778 26.2219080 24.9298409 30.4496277 32.6727432 2
##        43        44        45        46        47
## 29.0340541 27.4854737 25.4808696 24.8538370 21.1106425 1
```

Residulas

```
Residuals = (medv-predict(lm.fit))
Residuals
```

```
##            1            2            3            4
##  -5.822595098 -4.270389786  3.974858016  1.639304221
##            6            7            8            9
##  -0.904083746  0.155272588 10.739604245 10.381136279
##           11           12           13           14
##  -0.125331595 -3.046685955  2.071434468 -6.306433217
##           16           17           18           19
##  -6.606922853 -5.202516132 -3.116616860 -3.247763934
##           21           22           23           24
##  -0.983803463 -1.814658317 -1.568916977 -1.166859727
##           26           27           28           29
##  -4.968526049 -3.883609950 -3.336988046 -3.993209151
##           31           32           33           34
##  -0.382725484 -7.665197306  4.972026713 -4.020435238
##           36           37           38           39
```

Mean Square Error

```
MSE = mean((medv-predict(lm.fit))^2)
MSE
```

```
## [1] 38.48297
```