Appendix R2: Foundations

Statistics and Regression Overview

J. Alberto Espinosa

2/3/2023

Contents

Introduction

This R Markdown document contains R code examples for basic statistical analysis associated with descriptive analytics. It also contains examples of predictive modeling with simple linear regression, bi-variate regression with a dummy variables and multivariate analysis. All predictive analytics work should

start with very thorough descriptive analytics first, to become familiarized with the data at hand. The most basic descriptive analytics often performed before start building predictive models include:

Visual inspection of the data, including things like scatter plots, histograms, QQ Plots, etc. Please refer to the ITEC621_RIntro.R script.

Descriptive Statistics, including things like means, medians, standard deviations, minimum/maximum values, outliers, etc.

Covariance Analysis: covariance is an important and foundational statistical concept to help understand if two variable co-vary in one direction or another, or not. But it is not practical when variables have dissimilar scales because results will change if you change the scale of a variable (e.g., from inches to feet).

Correlation Analysis (Quantitative x Quantitative): Is the covariance of two variables, divided by the standard deviation of each of the two variables. Because we divide by the variance of each variable, all issues of scale go away and the correlation values are bound between -1 (perfectly negatively correlated) to 0 (uncorrelated or independent) to +1 (perfectly positively correlated). Because correlation is based on differences in values and variance, you can only compute correlation when both variables involved are quantitative. If one of the variables (or both) is (are) binary you can still compute the correlation statistic, but it is not as useful as ANOVA.

ANOVA or Analysis of Variance (Quantitative x Categorical). If one of the variables is quantitative and the other is categorical or even binary, it is more useful to evaluate if they co-vary with an ANOVA test. ANOVA computes the mean for the quantitative variable for each of the categories of the other (categorical) variable and evalutes if these means vary significantly across the categories. It is called Analysis of "Variance" and not analysis of "Means" because the means are evaluated to see whether they vary more within each category (not significant or independent) than across categories (significant co-variation). I illustrate this more clearly below.

Chi-Square Test of Independence (Categorical x Categorical). If you want to understand the co-variation between two categorical variables, correlation and ANOVA won't help. To evaluate this, the typical approach is to cross tabulate all the categories of one variable in rows and all the categories of the other variables as columns, with the cross-tabulated counts in the respective cells. If the two variables are independent, the proportion of counts between any two cells in a given column (or row) will be similar to the proportion of counts for the respective row (or column) totals. But if one variable has a significant influence on the other, the cell proportions will be very different than the row (or column proportions). I illustrate this more clearly below

1. Covariance and Correlation (Quantitative x Quantitative)

It is important to develop a good sense for which variables covary or not with others. Generally speaking, when building predictive models, the goal is to have predictors that are highly correlated with the outcome (i.e., dependent) variable, but are not correlated with each other (i.e., independent variables). Thus, covariance and correlation matrices provide very useful information when developing predictive models.

Let's look at the diamonds data in the ggplot package:

```
require(ggplot2) # Contains the diamonds data set
data(diamonds) # Load the data set into the work environment
```

The `cov()` and `cor()` functions in the **{stats}** package provide quick covariance and correlation, respectively. They both requre a matrix as an input, so data frames need to be first converted into a matrix. For example, let's bind 3 variable vectors into a data frame and convert it into a matrix:

```
MyDiamonds.dat <- data.frame(diamonds$price,
                             diamonds$carat,
                             diamonds$depth)

MyDiamonds.mat <- as.matrix(MyDiamonds.dat)
```

Alternatively, you can just specify the column numbers this way, all rows, columns 7, 1 and 5

```
MyDiamond.mat <- as.matrix(data.frame(diamonds[, c(7,1,5)]))
```

Before we continue, let's change the annoying scientific notation (the "scipen" keyword stands for "scientific notation penalty"). For example, if we want only very small numbers in scientific location we can use the `options()` function with `scipen=4`, so that only values with more than 4 zeros after the decimal point are displayed in scientific notation.

```
options(scipen="4")
```

Now you can compute the covariance and correlation matrices for these 3 variables

```
options(scipen = 4) # To limit the use of scientific notation

cov(MyDiamonds.mat, use="complete.obs") # Discard rows with incomplete data
```

```
##                 diamonds.price diamonds.carat diamonds.depth
## diamonds.price 15915629.42430  1742.76536427   -60.85371214
## diamonds.carat     1742.76536     0.22468666     0.01916653
## diamonds.depth      -60.85371     0.01916653     2.05240384
```

```
cov(MyDiamonds.mat, use="pairwise.complete.obs") # Discard pairs without data
```

```
##                 diamonds.price diamonds.carat diamonds.depth
## diamonds.price 15915629.42430  1742.76536427   -60.85371214
## diamonds.carat     1742.76536     0.22468666     0.01916653
## diamonds.depth      -60.85371     0.01916653     2.05240384
```

```
cor(MyDiamonds.mat, use="complete.obs") # Discard rows with incomplete data
```

```
##                 diamonds.price diamonds.carat diamonds.depth
## diamonds.price      1.0000000     0.92159130    -0.01064740
## diamonds.carat      0.9215913     1.00000000     0.02822431
## diamonds.depth     -0.0106474     0.02822431     1.00000000
```

```
cor(MyDiamonds.mat, use="pairwise.complete.obs") # Discard pairs without data
```

```
##                 diamonds.price diamonds.carat diamonds.depth
## diamonds.price      1.0000000     0.92159130    -0.01064740
## diamonds.carat      0.9215913     1.00000000     0.02822431
## diamonds.depth     -0.0106474     0.02822431     1.00000000
```

It is easy to figure out from the output above that the covariance matrix is difficult to interpret because it is affected by the scales of the variables involved. For example, the covariance of price with carats is 1742.76. Is this high or low? Hard to tell. But if we look at the correlation matrix we see that the correlation is 0.92, which is close to 1, so yes, it is very high. Again, covariance is a very important statistical value of great mathematical and modeling interest. But for business interpretations, correlation is more useful. Consequently, we will use correlation from now on for the most part.

Unfortunately the `cor()` function only gives correlation values, not **p-values**. For p-values, the `rcorr()` function in the **{Hmisc}** package does the job. It returns the number of observations plus 2 matrices -> one with **correlation** values and one with the respective **p-values**.

```
library(Hmisc) # Load the library. Note the H is upper case

rcorr(MyDiamonds.mat,
      type = "pearson") # Requires a matrix as an input
```

```
##                 diamonds.price diamonds.carat diamonds.depth
## diamonds.price           1.00           0.92          -0.01
## diamonds.carat           0.92           1.00           0.03
## diamonds.depth          -0.01           0.03           1.00
##
## n= 53940
##
##
## P
##                 diamonds.price diamonds.carat diamonds.depth
## diamonds.price                 0.0000         0.0134
## diamonds.carat  0.0000                        0.0000
## diamonds.depth  0.0134         0.0000
```

Another example with the `mtcars{database}` dataset

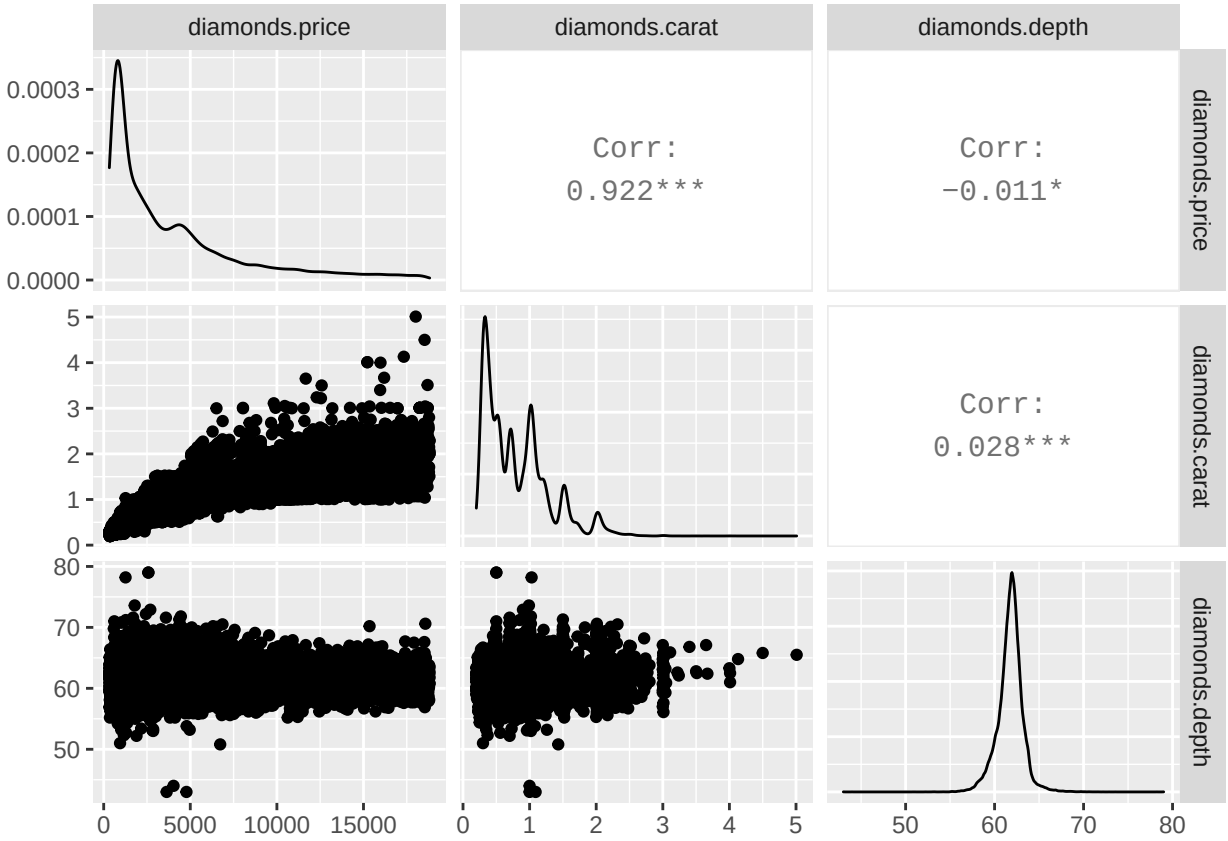
```
rcorr(as.matrix(mtcars)) # The default is Pearson correlation
```

```
##        mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
## mpg   1.00 -0.85 -0.85 -0.78  0.68 -0.87  0.42  0.66  0.60  0.48 -0.55
## cyl  -0.85  1.00  0.90  0.83 -0.70  0.78 -0.59 -0.81 -0.52 -0.49  0.53
## disp -0.85  0.90  1.00  0.79 -0.71  0.89 -0.43 -0.71 -0.59 -0.56  0.39
## hp   -0.78  0.83  0.79  1.00 -0.45  0.66 -0.71 -0.72 -0.24 -0.13  0.75
## drat  0.68 -0.70 -0.71 -0.45  1.00 -0.71  0.09  0.44  0.71  0.70 -0.09
## wt   -0.87  0.78  0.89  0.66 -0.71  1.00 -0.17 -0.55 -0.69 -0.58  0.43
## qsec  0.42 -0.59 -0.43 -0.71  0.09 -0.17  1.00  0.74 -0.23 -0.21 -0.66
## vs    0.66 -0.81 -0.71 -0.72  0.44 -0.55  0.74  1.00  0.17  0.21 -0.57
## am    0.60 -0.52 -0.59 -0.24  0.71 -0.69 -0.23  0.17  1.00  0.79  0.06
## gear  0.48 -0.49 -0.56 -0.13  0.70 -0.58 -0.21  0.21  0.79  1.00  0.27
## carb -0.55  0.53  0.39  0.75 -0.09  0.43 -0.66 -0.57  0.06  0.27  1.00
##
## n= 32
##
```

```
##
## P
##       mpg    cyl    disp   hp     drat   wt     qsec   vs     am     gear
## mpg          0.0000 0.0000 0.0000 0.0000 0.0000 0.0171 0.0000 0.0003 0.0054
## cyl  0.0000         0.0000 0.0000 0.0000 0.0000 0.0004 0.0000 0.0022 0.0042
## disp 0.0000 0.0000         0.0000 0.0000 0.0000 0.0131 0.0000 0.0004 0.0010
## hp   0.0000 0.0000 0.0000         0.0100 0.0000 0.0000 0.0000 0.1798 0.4930
## drat 0.0000 0.0000 0.0000 0.0100         0.0000 0.6196 0.0117 0.0000 0.0000
## wt   0.0000 0.0000 0.0000 0.0000 0.0000         0.3389 0.0010 0.0000 0.0005
## qsec 0.0171 0.0004 0.0131 0.0000 0.6196 0.3389         0.0000 0.2057 0.2425
## vs   0.0000 0.0000 0.0000 0.0000 0.0117 0.0010 0.0000         0.3570 0.2579
## am   0.0003 0.0022 0.0004 0.1798 0.0000 0.0000 0.2057 0.3570         0.0000
## gear 0.0054 0.0042 0.0010 0.4930 0.0000 0.0005 0.2425 0.2579 0.0000
## carb 0.0011 0.0019 0.0253 0.0000 0.6212 0.0146 0.0000 0.0007 0.7545 0.1290
##       carb
## mpg   0.0011
## cyl   0.0019
## disp  0.0253
## hp    0.0000
## drat  0.6212
## wt    0.0146
## qsec  0.0000
## vs    0.0007
## am    0.7545
## gear  0.1290
## carb
```

You can display correlations (above the diagonal), distributions (diagonal) and scatterplots (below the diagonal) all together with the `ggpairs()` function in the **{GGally}** library:

```
require(GGally) # Package with useful graphics displays

# WARNING -- the following takes a LONG TIME
ggpairs(MyDiamonds.dat) # It can be a bit slow
```

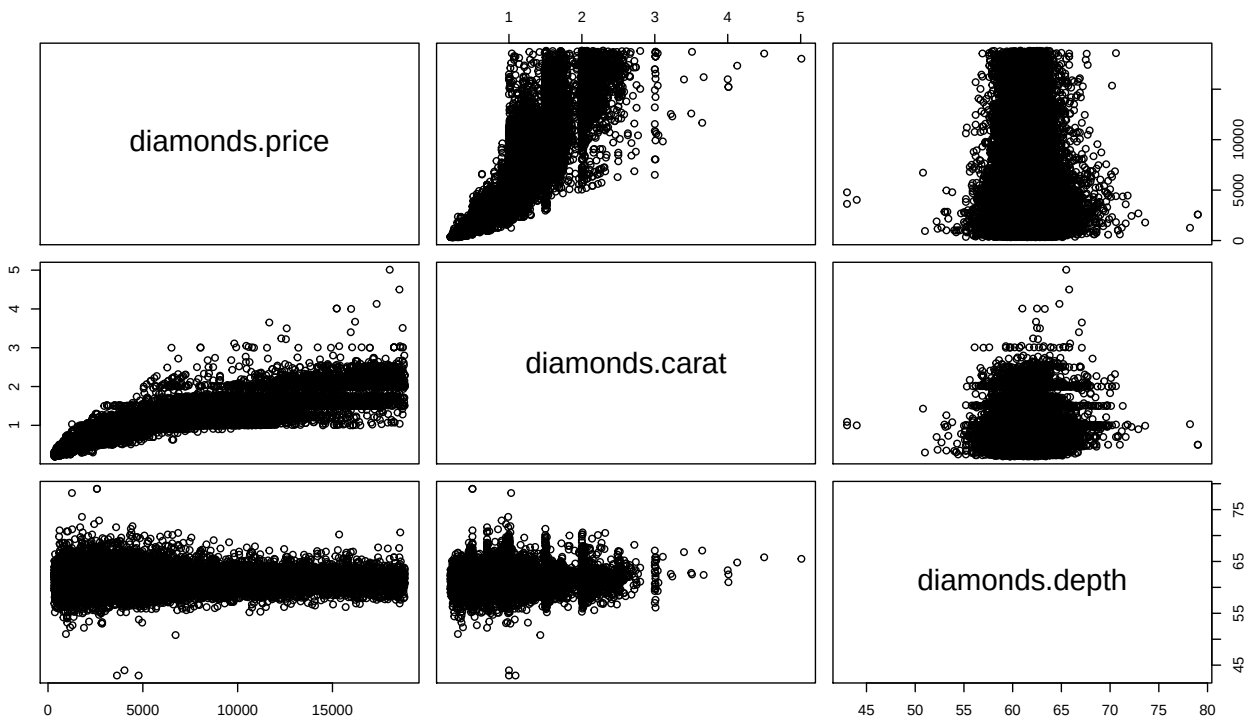


It works with categorical data too. Try this on your own. It takes a long time:

```
ggpairs(diamonds)
```

The `pairs()` function from the base **{graphics}** package works well too. It works with both, matrices and data frames:

```
pairs(MyDiamonds.dat)
```



It works with categorical data too. Try this on your own. It takes a long time:

`pairs(diamonds)`

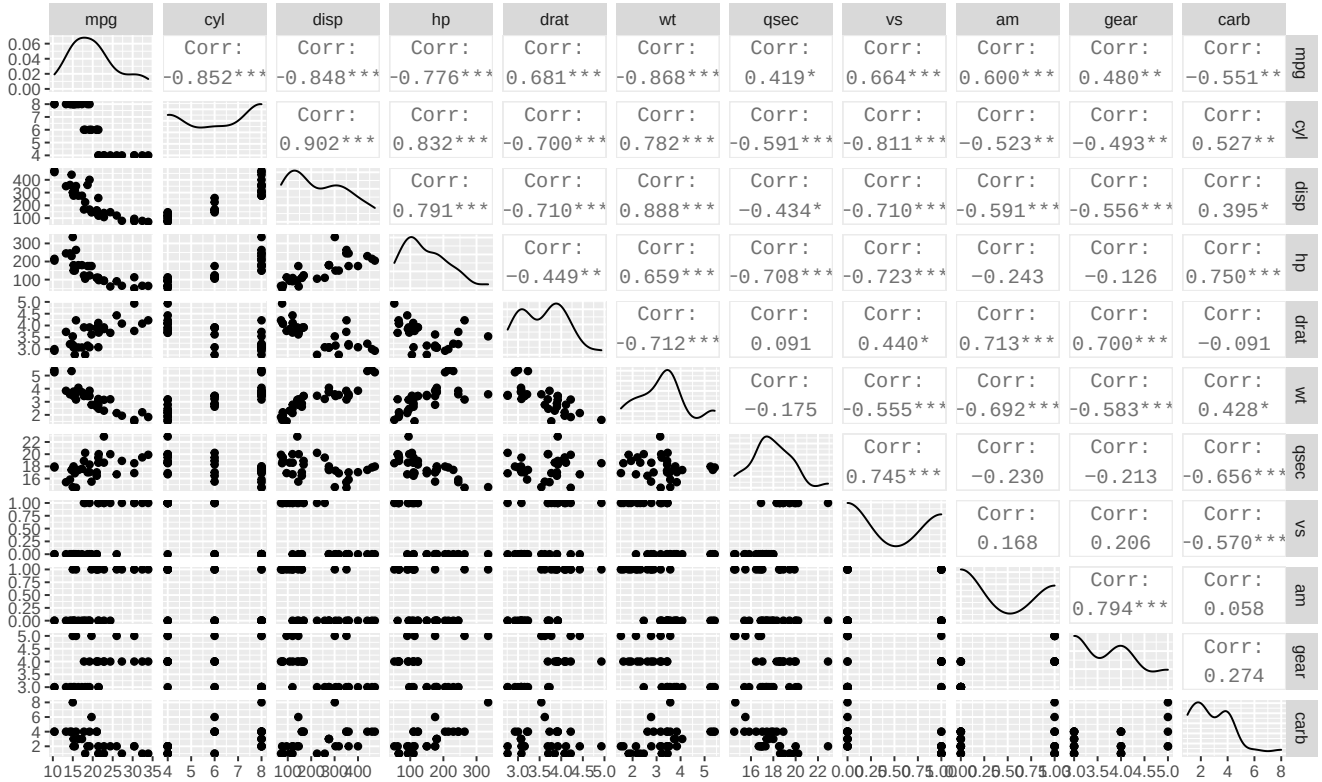
Example with the **mtcars** dataset:

```
rcorr(as.matrix(mtcars))
```

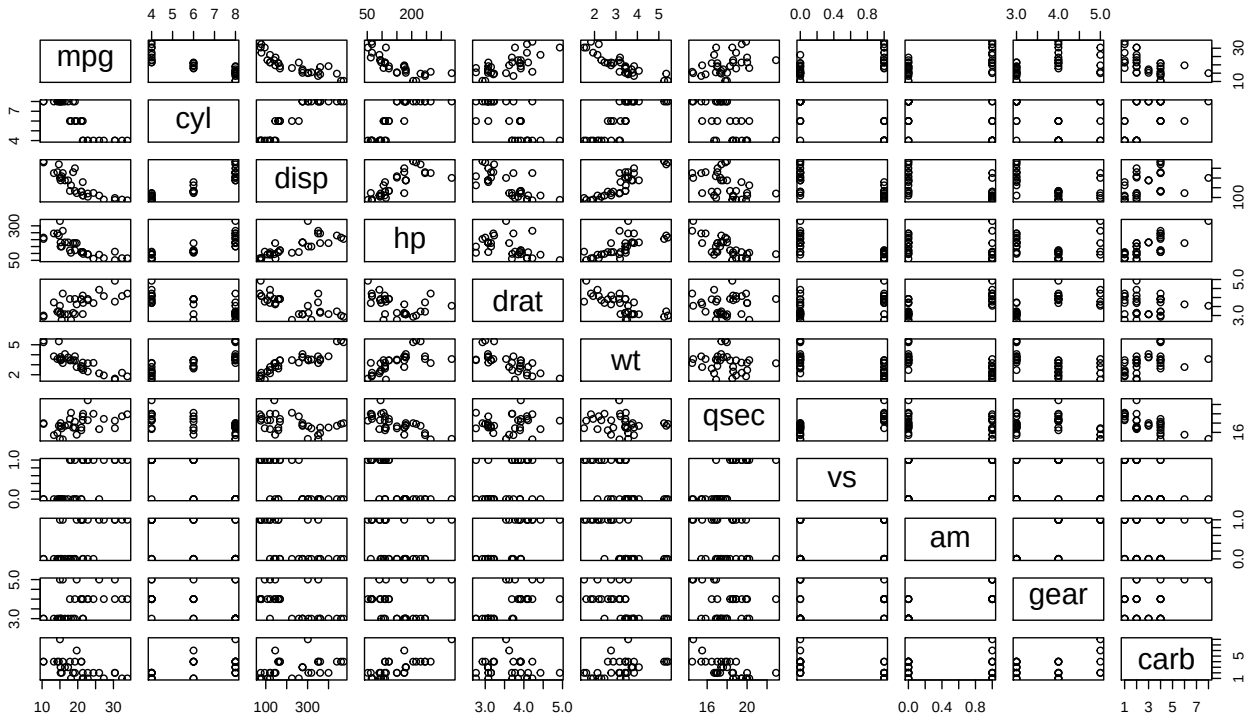
```
##        mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
## mpg   1.00 -0.85 -0.85 -0.78  0.68 -0.87  0.42  0.66  0.60  0.48 -0.55
## cyl  -0.85  1.00  0.90  0.83 -0.70  0.78 -0.59 -0.81 -0.52 -0.49  0.53
## disp -0.85  0.90  1.00  0.79 -0.71  0.89 -0.43 -0.71 -0.59 -0.56  0.39
## hp   -0.78  0.83  0.79  1.00 -0.45  0.66 -0.71 -0.72 -0.24 -0.13  0.75
## drat  0.68 -0.70 -0.71 -0.45  1.00 -0.71  0.09  0.44  0.71  0.70 -0.09
## wt   -0.87  0.78  0.89  0.66 -0.71  1.00 -0.17 -0.55 -0.69 -0.58  0.43
## qsec  0.42 -0.59 -0.43 -0.71  0.09 -0.17  1.00  0.74 -0.23 -0.21 -0.66
## vs    0.66 -0.81 -0.71 -0.72  0.44 -0.55  0.74  1.00  0.17  0.21 -0.57
## am    0.60 -0.52 -0.59 -0.24  0.71 -0.69 -0.23  0.17  1.00  0.79  0.06
## gear  0.48 -0.49 -0.56 -0.13  0.70 -0.58 -0.21  0.21  0.79  1.00  0.27
## carb -0.55  0.53  0.39  0.75 -0.09  0.43 -0.66 -0.57  0.06  0.27  1.00
##
## n= 32
##
##
## P
##      mpg    cyl    disp   hp     drat   wt     qsec   vs     am     gear
## mpg         0.0000 0.0000 0.0000 0.0000 0.0000 0.0171 0.0000 0.0003 0.0054
## cyl  0.0000        0.0000 0.0000 0.0000 0.0000 0.0004 0.0000 0.0022 0.0042
```

```
## disp 0.0000 0.0000        0.0000 0.0000 0.0000 0.0131 0.0000 0.0004 0.0010
## hp   0.0000 0.0000 0.0000        0.0100 0.0000 0.0000 0.0000 0.1798 0.4930
## drat 0.0000 0.0000 0.0000 0.0100        0.0000 0.6196 0.0117 0.0000 0.0000
## wt   0.0000 0.0000 0.0000 0.0000 0.0000        0.3389 0.0010 0.0000 0.0005
## qsec 0.0171 0.0004 0.0131 0.0000 0.6196 0.3389        0.0000 0.2057 0.2425
## vs   0.0000 0.0000 0.0000 0.0000 0.0117 0.0010 0.0000        0.3570 0.2579
## am   0.0003 0.0022 0.0004 0.1798 0.0000 0.0000 0.2057 0.3570        0.0000
## gear 0.0054 0.0042 0.0010 0.4930 0.0000 0.0005 0.2425 0.2579 0.0000
## carb 0.0011 0.0019 0.0253 0.0000 0.6212 0.0146 0.0000 0.0007 0.7545 0.1290
##      carb
## mpg  0.0011
## cyl  0.0019
## disp 0.0253
## hp   0.0000
## drat 0.6212
## wt   0.0146
## qsec 0.0000
## vs   0.0007
## am   0.7545
## gear 0.1290
## carb
```

```
ggpairs(mtcars) # Takes a bit of time
```

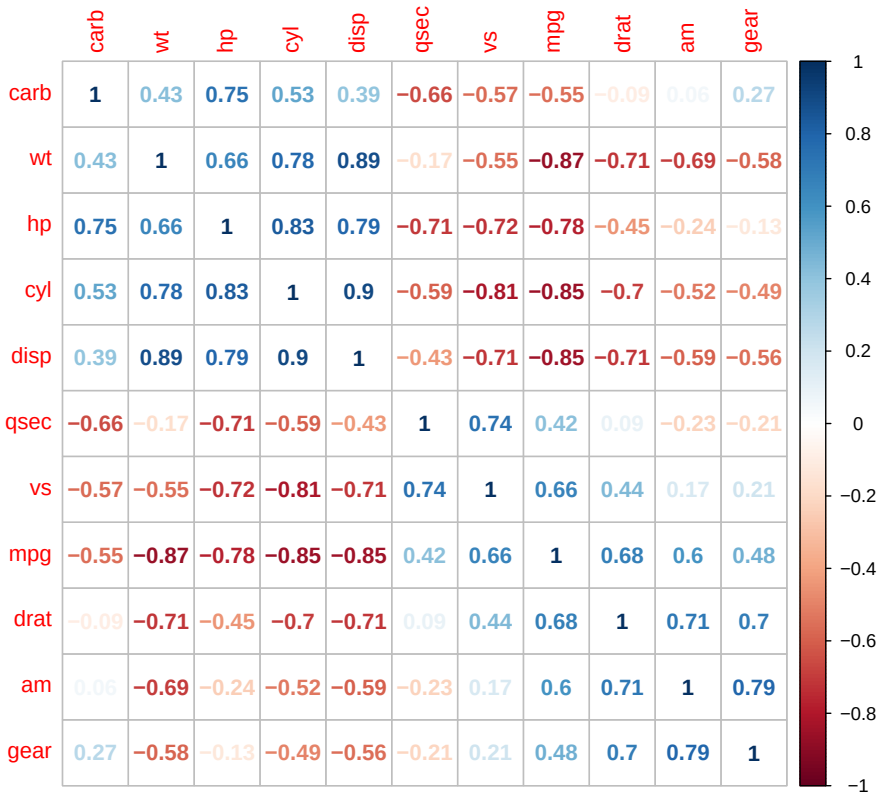



```
pairs(mtcars) # A bit quicker, but not as nice
```

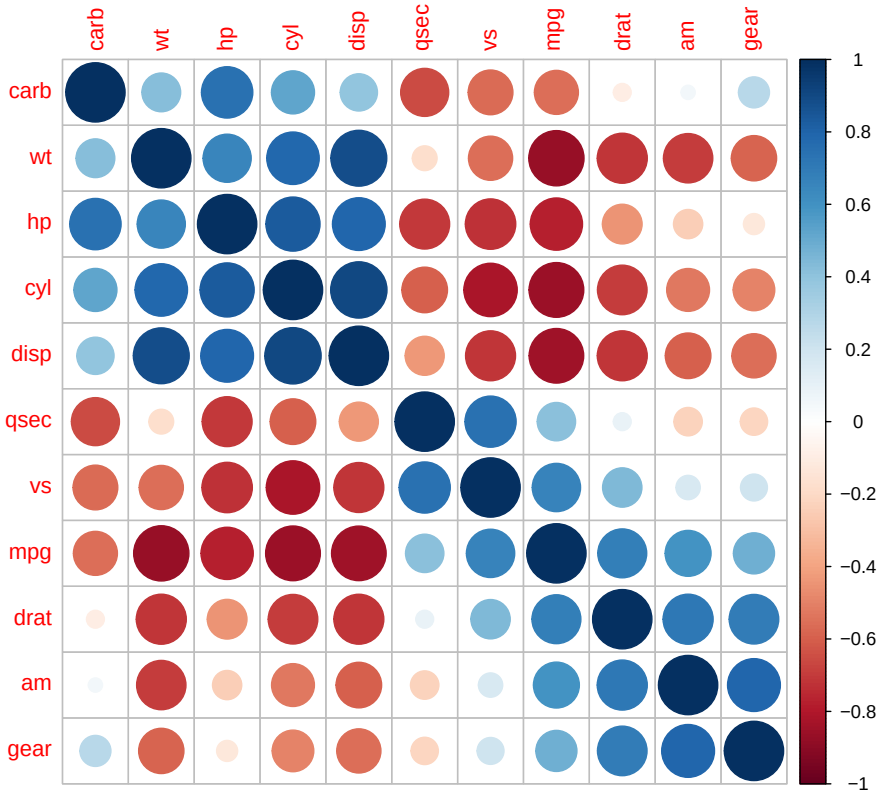


The `corrplot()` function in the **{corrplot}** library prvides nice graphics

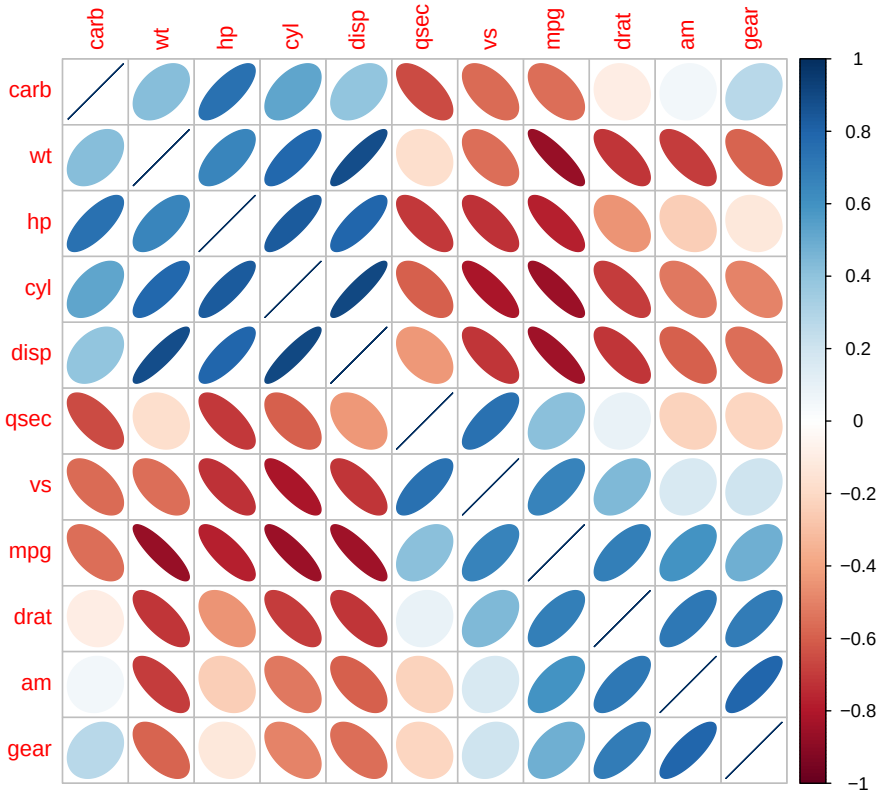
```
library(corrplot) # Library for correlation plots
mtCorr <- cor(mtcars) # First, store the correlation object

corrplot(mtCorr,
         method = "number", order = "hclust") # Show correlation
```




```
corrplot(mtCorr,
         method = "circle", order = "hclust") # Then plot it
```

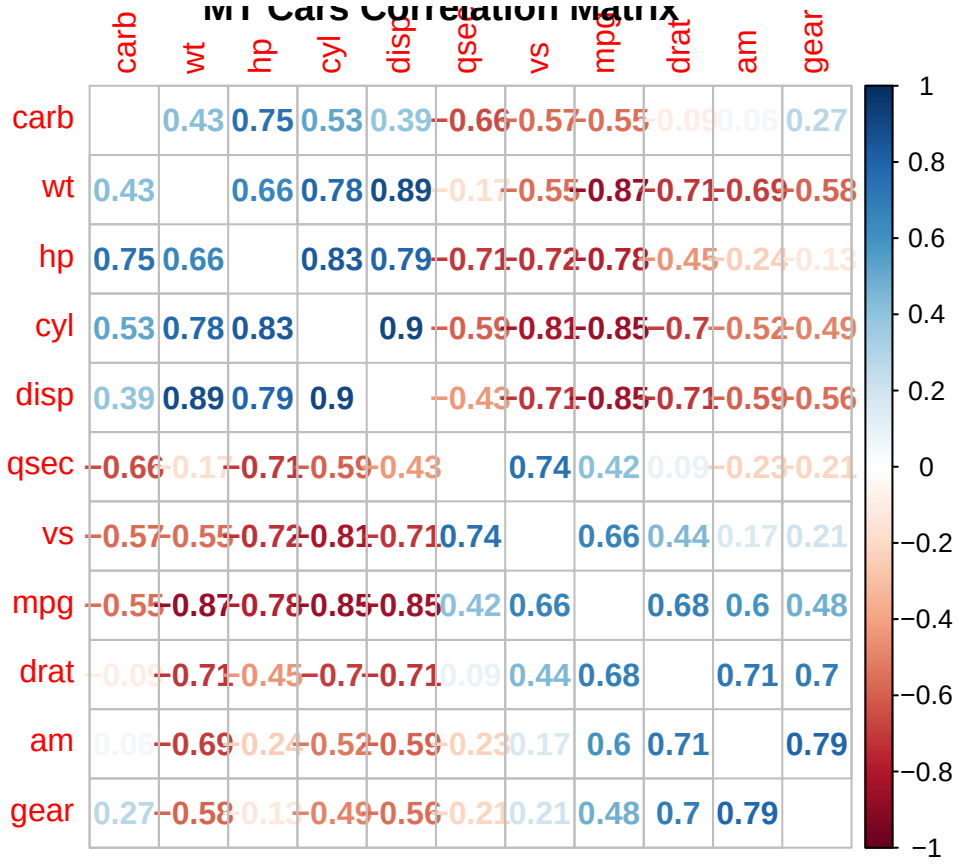


```
corrplot(mtCorr,
         method = "ellipse", order = "hclust") # Slanted left/right for +/-
```



You can order variables clustered by correlation values and omit the diagonal

```
corrplot(mtCorr,
         method = "number", order = "hclust",
         diag = F,
         title = "MT Cars Correlation Matrix")
```



Type `?corrplot()` at the console to see all the methods available

2. Analysis of Variance (ANOVA) (Quantitative x Categorical)

Analysis of Variance (ANOVA) is a test that compares the means of 2 or more groups or categories. The means of 2 groups may be different, but ANOVA tells us if this difference is significant. Generally speaking, ANOVA compares the within-category variance for both groups against the between-category variance across both groups. If the between-group variance is significantly larger than the within-group variance, then the difference in means between the two groups is significant. Otherwise it is not.

ANOVA can be used any time you need to compare means between or across groups. For example, an OLS regression reports an R-Squared (i.e., explained variance) and a p-value for the entire regression model. The p-value is based on an ANOVA test, which analyzes whether the variance in the errors or residuals around the regression line are significantly smaller that the variance of the dependent variable values, relative to its overall mean. If the p-value or ANOVA test is significant, we say that the regression model has a significant explanatory power, relative to the plain mean of provides significantly more explanation than just the mean and its variance (i.e., the "null" model, with no predictors).

There are 2 popular functions to do ANOVA tests: `aov()` and `anova()`, both available in **{stats}**.

The aov() Function

The `aov()` function is useful to compare the means of a continuous variable (e.g., price) across various categories (e.g., clarity, color). In a nutshell, it tests whether the variance across groups is larger than

within groups. The larger the F statistic, the more confidence we have that the variance across groups is significant. In the example below, the ANOVA test is significant at the p « 0.05 level.

```
aov(price ~ clarity,
    data = diamonds) # Run the ANOVA on a single factor
```

```
## Call:
##    aov(formula = price ~ clarity, data = diamonds)
##
## Terms:
##                       clarity    Residuals
## Sum of Squares   23307802882 835165332636
## Deg. of Freedom            7        53932
##
## Residual standard error: 3935.165
## Estimated effects may be unbalanced
```

```
summary(aov(price ~ clarity,
            data = diamonds)) # More detailed results
```

```
##                Df       Sum Sq    Mean Sq F value Pr(>F)
## clarity         7  23307802882 3329686126     215 <2e-16 ***
## Residuals   53932 835165332636   15485525
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Alternatively, store the ANOVA results in an object

```
MyAOV <- aov(price ~ clarity,
             data = diamonds)

summary(MyAOV) # Show the ANOVA object result summary
```

```
##                Df       Sum Sq    Mean Sq F value Pr(>F)
## clarity         7  23307802882 3329686126     215 <2e-16 ***
## Residuals   53932 835165332636   15485525
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

You can do ANOVA for a continuous variable across more than one categorical variable. In the examples below there are 2 and 3 ANOVA tests, respectively. The first shows that there is a significant differenece in price across clarity categories, but analyzed within color. In the second test is the reverse, that is there is a difference in price across color, within clarity categories. The second ANOVA test works the same way but, for example, the difference in prices is significant across clarity categories, but within the same color and cut. And so on.

```
summary(aov(price ~ clarity + color,
            data = diamonds)) # ANOVA on 2 factors
```

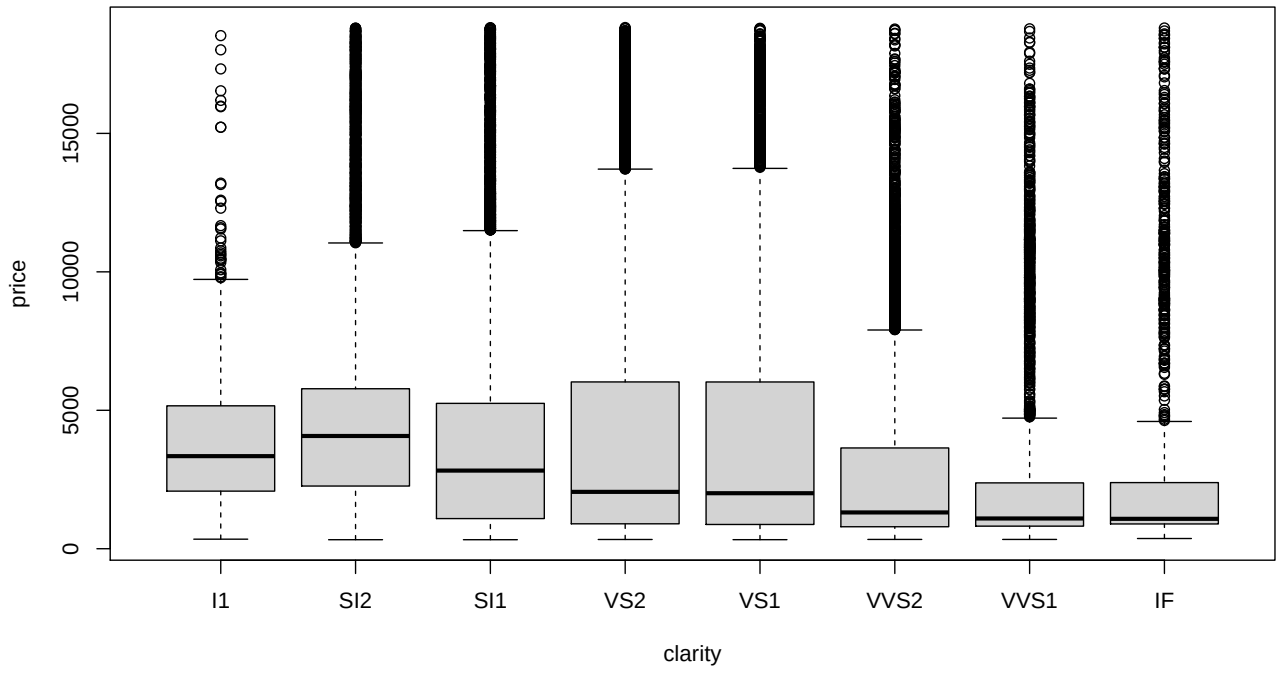
```
##                Df       Sum Sq    Mean Sq F value Pr(>F)
## clarity         7  23307802882 3329686126   222.4 <2e-16 ***
## color           6  27663814802 4610635800   307.9 <2e-16 ***
## Residuals   53926 807501517834   14974252
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(aov(price ~ clarity + color + cut,
            data = diamonds)) # On 3 factors
```

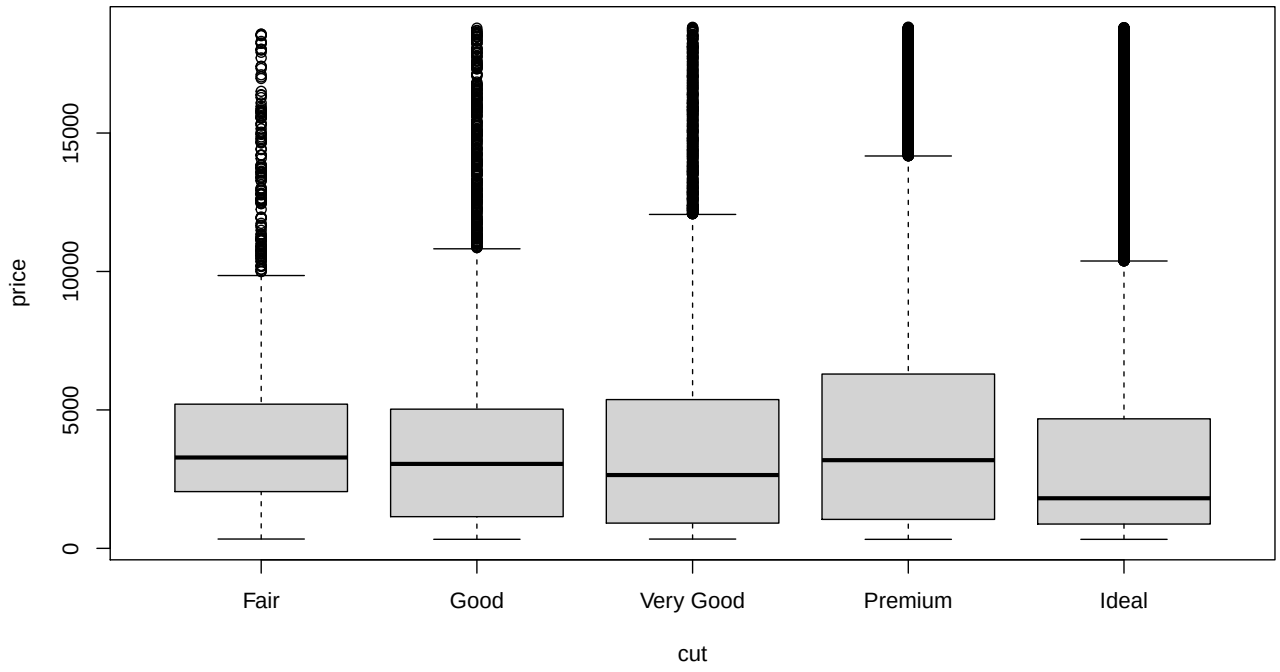
```
##                Df       Sum Sq    Mean Sq F value Pr(>F)
## clarity         7  23307802882 3329686126  223.89 <2e-16 ***
## color           6  27663814802 4610635800  310.02 <2e-16 ***
## cut             4   5574933424 1393733356   93.72 <2e-16 ***
## Residuals   53922 801926584410   14871974
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

You can also visualize the differences with boxplots. These visualizations should tell a similar story than the respective ANOVA tests, but without significance statistics, just visually:

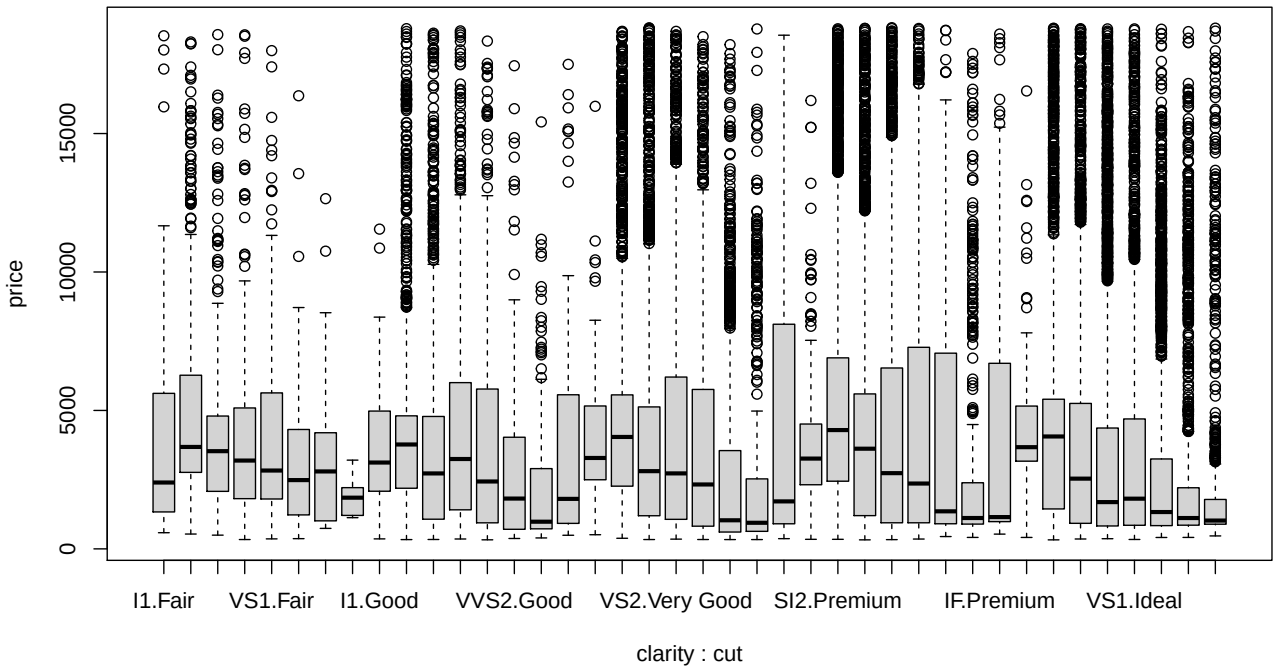
```
boxplot(price ~ clarity,
        data = diamonds)
```



```
boxplot(price ~ cut,
        data = diamonds)
```




```
boxplot(price ~ clarity + cut,
        data = diamonds)
```

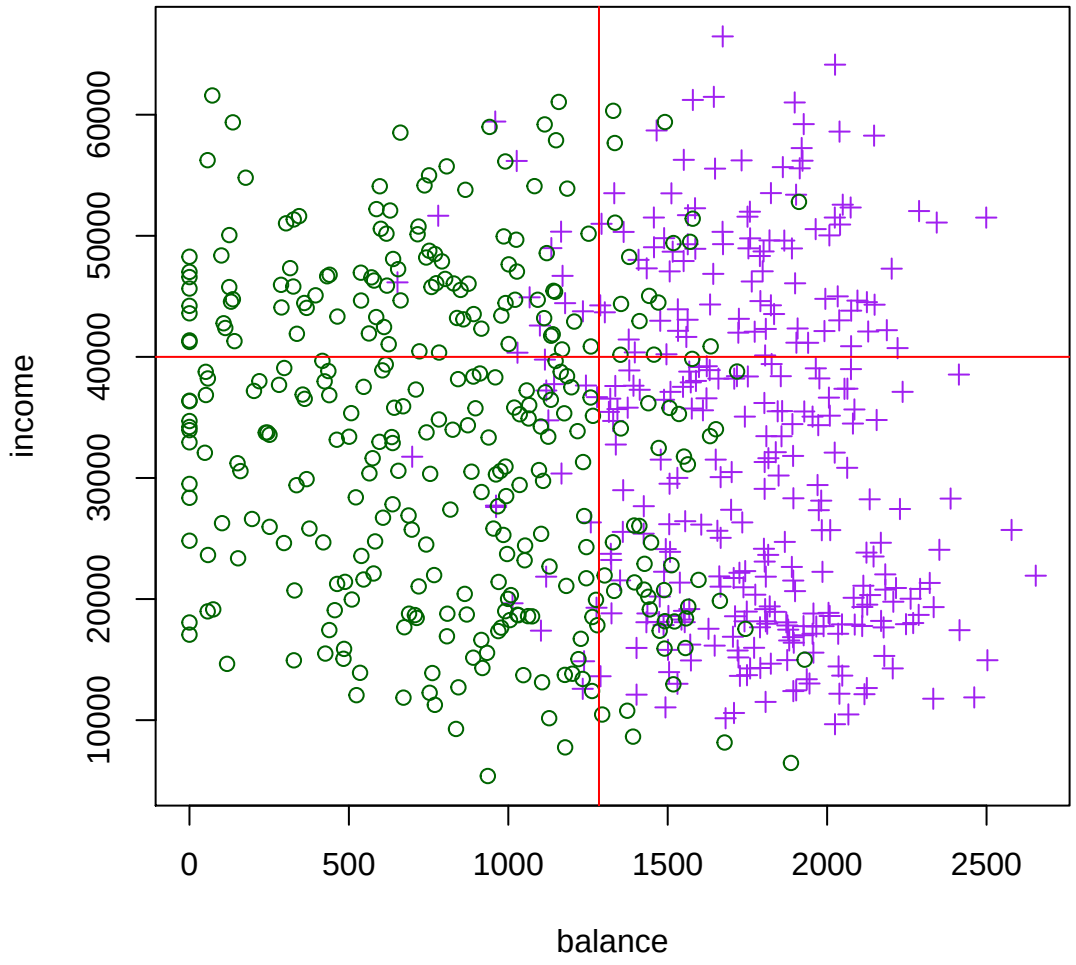


Replicating the ISLR textbook example with the credit Default dataset

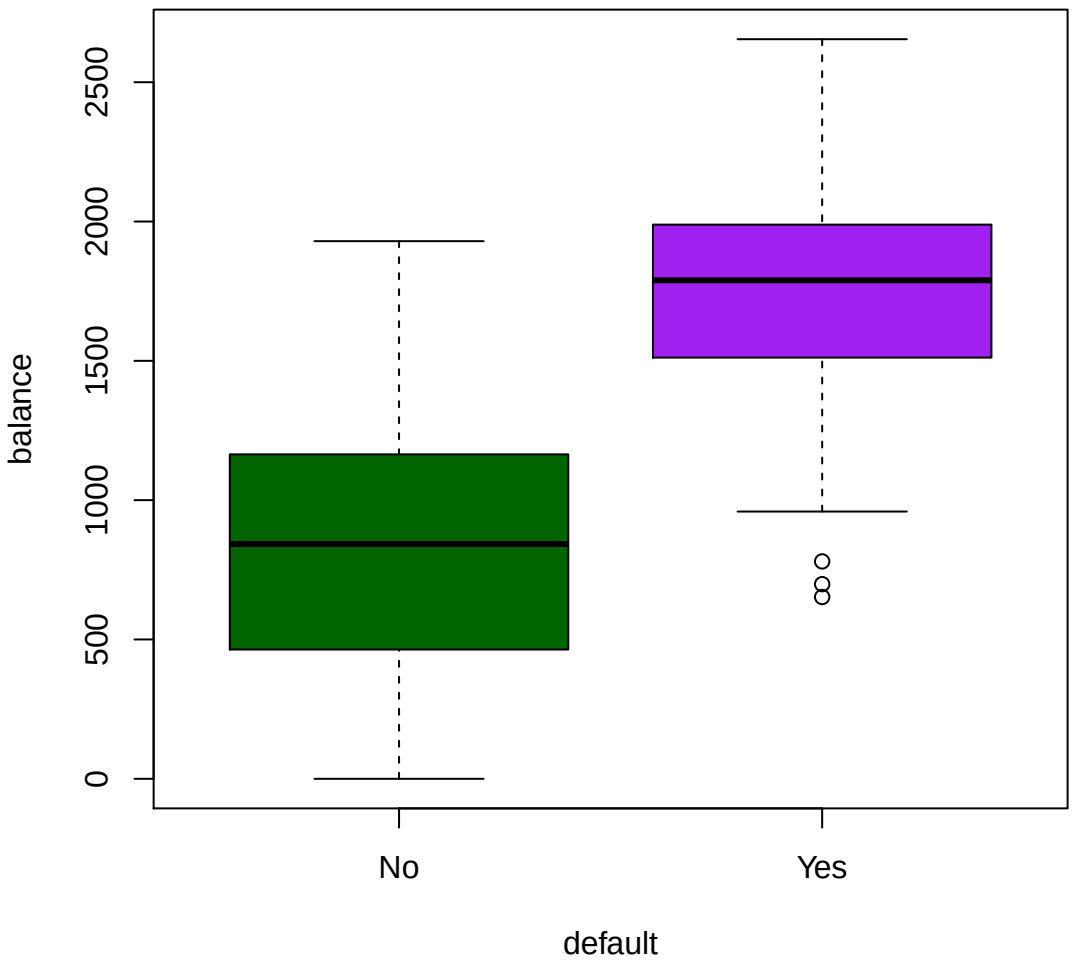
This is another illustration of ANOVA and it replicates an example provided in the ISLR book, but it also conducts an ANOVA test in addition to the plots. Similarly to the example in the ISLR book, I first extract a balanced sample from the data set. It is common in classification datasets to have unbalanced number of observations across categories. For example, medical datasets with disease diagnosis tend to have far more negatives than positives. The same is true for transaction fraud, etc. This example is no different. The **Default** dataset is synthetic and contains 10,000 observations. Only 333 of them represent loan defaults. To balance the analysis I use all 333 default records, but extract a random sample of 333 records from the no-default group.

```
library(ISLR) # Contains the Default data set

set.seed(1)

def.yes <- subset(Default,
                  default == "Yes") # Subset with all default loans

def.no <- subset(Default,
                 default == "No") # Subset with all no-default loans

def.no <- def.no[sample(nrow(def.no), 333),] # Extract 333 of them randomly

def.sub <- rbind(def.yes, def.no) # Bind the defaults and no defaults

# Scatterplot with vertical lines at the respective means
```

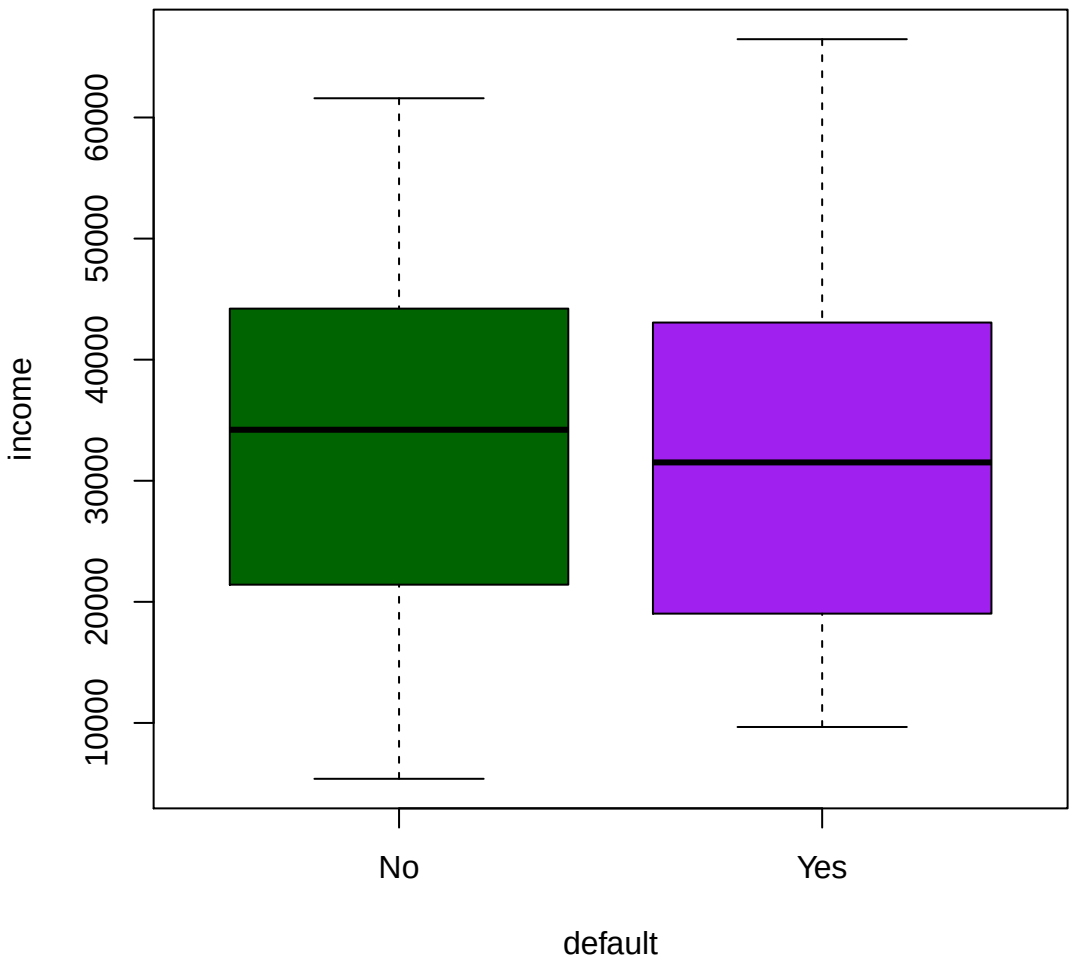
```
plot(income ~ balance,
     data = def.sub,
     col = ifelse(default == "No", "darkgreen", "purple"),
     pch = ifelse(default == "No", 1, 3))

abline(v = mean(def.sub$balance),
       col = "red")

# abline(h=mean(def.sub$income), col="red")

abline(h = 40000, col = "red")
```




```
boxplot(balance ~ default,
        data = def.sub,
        col = c("darkgreen","purple"))
```




```
boxplot(income ~ default,
        data = def.sub,
        col = c("darkgreen","purple"))
```




```
summary(aov(balance ~ default,
            data = def.sub))
```

```
##              Df    Sum Sq   Mean Sq F value Pr(>F)
## default       1 143029929 143029929   850.7 <2e-16 ***
## Residuals   664 111636896    168128
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(aov(income ~ default,
            data = def.sub))
```

```
##              Df       Sum Sq   Mean Sq F value Pr(>F)
## default       1    277297260 277297260   1.535  0.216
## Residuals   664 119976574966 180687613
```

The anova() Function

The `anova()` function is useful when comparing nested linear models. One model is nested withing another if the larger models contains all the predictors of the small model. Let's fit 3 nested regression models with the diamonds dataset and price as the outcome variable. The first model is the null model with no predictors (`~ 1` in the model indicates the intercept as the only predictor, and in this case nothing else). The second model is a simple linear regression model with a single predictors, and the last has 2 predictors.

```
lm.null <- lm(price ~ 1,
              data = diamonds) # Null model

lm.small <- lm(price ~ carat,
               data = diamonds) # Small model

lm.large <- lm(price ~ carat + clarity,
               data = diamonds) # Large model

anova(lm.null, lm.small, lm.large) # Compare 3 nested models
```

```
## Analysis of Variance Table
##
## Model 1: price ~ 1
## Model 2: price ~ carat
## Model 3: price ~ carat + clarity
##   Res.Df          RSS Df    Sum of Sq        F    Pr(>F)
## 1  53939 858473135517
## 2  53938 129345695398  1 729127440120 435639.9 < 2.2e-16 ***
## 3  53931  90263944583  7  39081750815   3335.8 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The output shows 3 models and 2 ANOVA test results. The first ANOVA test compares the second model to the first in the `anova()` function. The second ANOVA test compares the thirds model to the second. You can test as many nested models as you wish in one pass. The first p-value is significant, so the small model has significantly more explanatory power than the null model. The second p-value is also significant, so the large model has more explanatory power than the small model. In other words, carats explain more variance in price than just the mean price, but adding clarity as a predictor improves the explanatory power over the small model.

3. Chi-Square Test of Independence (Categorical x Categorical)

For example, say that we want to see if diamond "cut" and "color" co-vary. In other words, we want to know if diamond cut and color are independent (i.e. one does not affect the value of the other) or if they are dependent (i.e., the value of one variable influences the value of another).

The first step is to prepare a cross table with the counts for both categories

```
attach(diamonds)
cross.table <- table (cut, color) # Store results in a cross table
cross.table # Check it out
```

```
##            color
## cut            D    E    F    G    H    I    J
##   Fair       163  224  312  314  303  175  119
##   Good       662  933  909  871  702  522  307
##   Very Good 1513 2400 2164 2299 1824 1204  678
##   Premium   1603 2337 2331 2924 2360 1428  808
##   Ideal     2834 3903 3826 4884 3115 2093  896
```

We can now look at the margin totals:

```
rowSums(cross.table) # Check the row totals
```

```
##      Fair      Good Very Good   Premium     Ideal
##      1610      4906     12082     13791     21551
```

```
cat("\n") # Blank line
```

```
colSums(cross.table) # Check the column totals
```

```
##     D     E     F     G     H     I     J
##  6775  9797  9542 11292  8304  5422  2808
```

```
cat("\n") # Blank line
```

```
sum(rowSums(cross.table)) # Compute table totals
```

```
## [1] 53940
```

```
sum(colSums(cross.table)) # Same result
```

```
## [1] 53940
```

If the row and column variables are totally independent the proportion of rowSums would be identical to the proportion of any two cells on the same two rows of the table. Similarly, the proportion colSums would be identical to the proportion of any two cells on the same two columns.

In fact, you can create a table of expected values as follows: `Exp.Cell(i,j) = rowSum(i) * colSum(j) / table.tot`. If the **observed** (actual) values in the table are close to the **expected** values, the two variables are independent. If the observed values are very different than the expected values, the two variables are dependent (i.e., they co-vary)

The `chisq.test()` function does this test for you by feeding the cross table. The null hypothesis is that the two variables are independent. A significant p-value rejects the null hypothesis suggesting that the variables are dependent. In addition, we can extract the observed an expected values from the **chisq** object:

```
chiSq.diamonds <- chisq.test(cross.table) # Conduct the test

chiSq.diamonds$observed # Check the observed values, same as the cross.table
```

```
##            color
## cut            D    E    F    G    H    I    J
##   Fair       163  224  312  314  303  175  119
##   Good       662  933  909  871  702  522  307
##   Very Good 1513 2400 2164 2299 1824 1204  678
##   Premium   1603 2337 2331 2924 2360 1428  808
##   Ideal     2834 3903 3826 4884 3115 2093  896
```

```
print(chiSq.diamonds$expected, digits = 2) # Check the expected values
```

```
##            color
## cut            D    E    F    G    H    I    J
##   Fair       202  292  285  337  248  162   84
##   Good       616  891  868 1027  755  493  255
##   Very Good 1518 2194 2137 2529 1860 1214  629
##   Premium   1732 2505 2440 2887 2123 1386  718
##   Ideal     2707 3914 3812 4512 3318 2166 1122
```

```
chiSq.diamonds # Check out Chi Square test results
```

```
##
##  Pearson's Chi-squared test
##
## data:  cross.table
## X-squared = 310.32, df = 24, p-value < 2.2e-16
```

The p-value is significant, so we reject the null hypothesis that cut and color are independent and conclude that they are dependent.

4. Simple Linear Regression

Let's work an example with the **Boston** housing dataset in the **{MASS}** package. We start by fitting a **null** model. Normally, you don't need to do this, but I do this hear as a building block. Let's load the library. You may also want to type ?Boston in the console to explore the variables in the dataset

```
library(MASS) # Contains the Boston data set
```

Fitting a Model

The `lm()` function in the **{stats}** library fits **linear models** based on the OLS regression model. The `lm()` model requires the **outcome variable**, followed by the ~ operator, followed by the **predictors**,

separated from each other with a + sign, and the data attribute to indicate the dataset to be used for fitting the model. For the null model, we specify no predictors and use **1**, which is like having a column of 1's in the dataset, which yield the intercept. As before, **medv** is the median value of homes in the respective county in the Boston area in thousands of $.

```
lm.fit.null <- lm(medv ~ 1,
                  data = Boston) # Use 1 for no predictors

summary(lm.fit.null) # The null model
```

```
##
## Call:
## lm(formula = medv ~ 1, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.533  -5.508  -1.333   2.467  27.467
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  22.5328     0.4089   55.11   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.197 on 505 degrees of freedom
```

```
# The outcome variable mean and the null model intercept are the same
mean(Boston$medv)
```

```
## [1] 22.53281
```

The null model is not very useful. We could obtain the same result just by calculating the mean of **medv**. Notice that there is no ANOVA F-test, model p-value or R-squared, because there is really nothing to report. In a simple regression model, we include a single predictor and expect to improve the explained variance of the model. The explained variance will most definitely increase when we add one predictor, but we need to figure out if the increase is significant or not.

A simple regression model has only one predictor so we don't use the + symbol yet. For example, let's add **lstat** as a predictor. This variable contains the percentage of population with low income status in the county.

```
lm.fit <- lm(medv ~ lstat,
             data = Boston) # Using the fixed data set Boston

summary(lm.fit)
```

```
##
## Call:
```

```
## lm(formula = medv ~ lstat, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.168  -3.990  -1.318   2.034  24.500
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.55384    0.56263   61.41   <2e-16 ***
## lstat       -0.95005    0.03873  -24.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

Notice that we specify the data set in the formula. This is convenient because it allows us to run the same model with a different dataste very quickly. But, alternatively, you could attach the data to the environment and you would no longer need to specify the dataset in the formula:

```
attach(Boston)
lm.fit <- lm(medv ~ lstat)
```

The lm() Object

The `lm()` function will yield an **lm** object, which we are naming **lm.fit**. This object contains properties (i.e., data) and methods (i.e., functions) to help us extract information from the model and do things like plotting, testing, etc. Let's extract some information from **lm.fit**. Ket's start by displaying the object itself.

```
lm.fit # Bare bones information
```

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

The `summary()` function always provide a summary of the most important properties stored in the model, which you would be expected to need in an analysis.

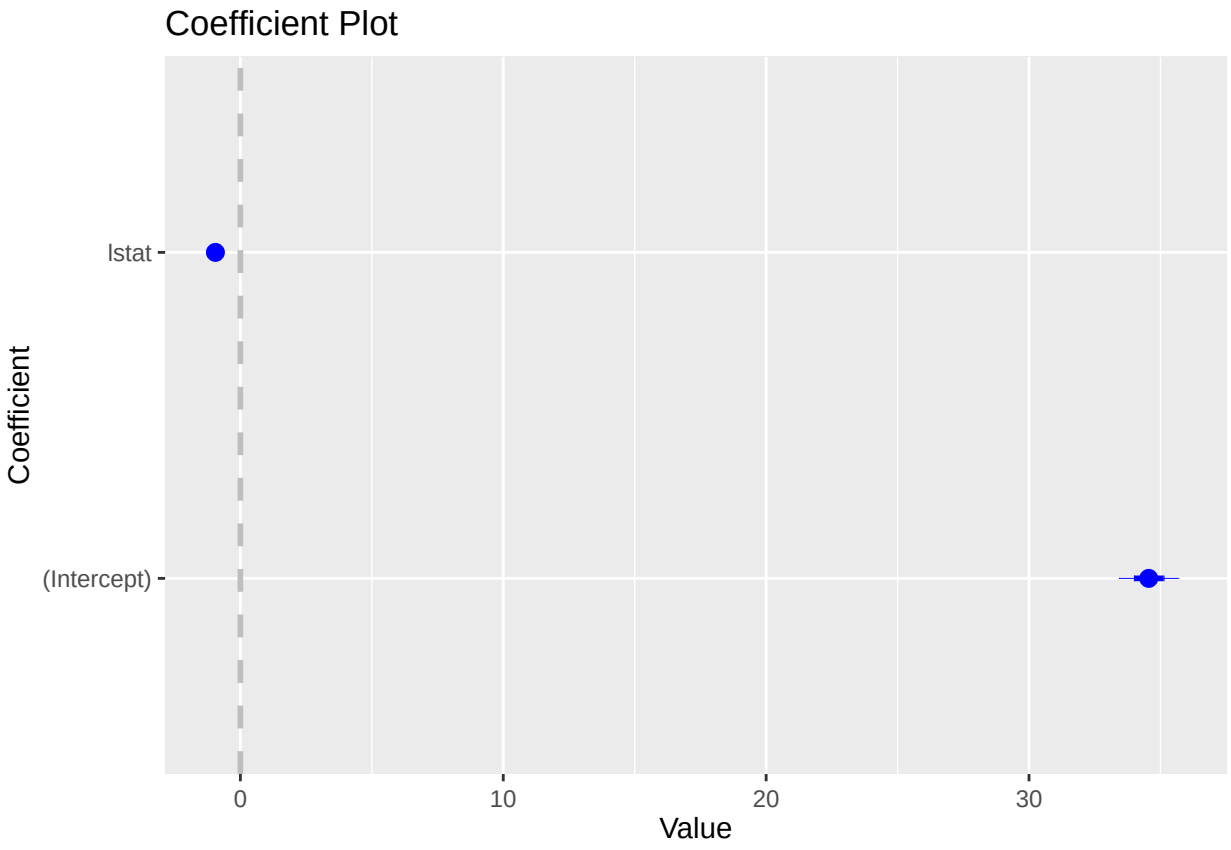
```
summary(lm.fit) # Display the summary of results
```

```
##
```

```
## 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
```

You can inspect the coefficients visually too with the `coefplot()` library and function. The dot represents the coefficient and the bands represent the confidence interval. If the confidence interval does not cross the 0 vertical line, then the coefficient is significant.

```
require(coefplot)
coefplot(lm.fit)
```



In the outup above we can see all coefficients, their respective standard errors and p-values. We can also see all key fit statistics for the regression model. If we want more specific data, we can use various functions.

For example, the `names()` function gives you a list of properties contained in the object:

```
names(lm.fit) # List all the properties of the lm.fit object
```

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

Each of these contains useful data, for example the regression coefficients:

```
lm.fit$coefficients
```

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

The regression formula:

```
lm.fit$call # Regression formula
```

```
## lm(formula = medv ~ lstat, data = Boston)
```

The first 12 residual values:

```
lm.fit$residuals[1:12] # First 12 residual values
```

```
##          1          2          3          4          5          6          7
## -5.8225951 -4.2703898  3.9748580  1.6393042  6.7099222 -0.9040837  0.1552726
##          8          9         10         11         12
## 10.7396042 10.3811363  0.5920031 -0.1253316 -3.0466860
```

The first 12 predicted values:

```
lm.fit$fitted.values[1:12] # First 12 predicted values
```

```
##         1         2         3         4         5         6         7         8
## 29.822595 25.870390 30.725142 31.760696 29.490078 29.604084 22.744727 16.360396
##         9        10        11        12
##  6.118864 18.307997 15.125332 21.946686
```

You can also use functions to extract data, for example, the coefficients:

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

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

```
confint(lm.fit) # Confidence intervals for the coefficients
```

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

To inspect the object in more detail you can use the `str()` function to display the whole structure of the object:

```
str(lm.fit) # So, for example, use:
```

```
## List of 12
##  $ coefficients : Named num [1:2] 34.55 -0.95
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "lstat"
##  $ residuals    : Named num [1:506] -5.82 -4.27 3.97 1.64 6.71 ...
##   ..- attr(*, "names")= chr [1:506] "1" "2" "3" "4" ...
##  $ effects      : Named num [1:506] -506.86 -152.46 4.43 2.12 7.13 ...
##   ..- attr(*, "names")= chr [1:506] "(Intercept)" "lstat" "" "" ...
##  $ rank         : int 2
##  $ fitted.values: Named num [1:506] 29.8 25.9 30.7 31.8 29.5 ...
##   ..- attr(*, "names")= chr [1:506] "1" "2" "3" "4" ...
##  $ assign       : int [1:2] 0 1
##  $ qr           :List of 5
##   ..$ qr   : num [1:506, 1:2] -22.4944 0.0445 0.0445 0.0445 0.0445 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:506] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:2] "(Intercept)" "lstat"
##   .. ..- attr(*, "assign")= int [1:2] 0 1
##   ..$ qraux: num [1:2] 1.04 1.02
##   ..$ pivot: int [1:2] 1 2
##   ..$ tol  : num 0.0000001
##   ..$ rank : int 2
##   ..- attr(*, "class")= chr "qr"
##  $ df.residual  : int 504
##  $ xlevels      : Named list()
##  $ call         : language lm(formula = medv ~ lstat, data = Boston)
##  $ terms        :Classes 'terms', 'formula'  language medv ~ lstat
##   .. ..- attr(*, "variables")= language list(medv, lstat)
##   .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:2] "medv" "lstat"
##   .. .. .. ..$ : chr "lstat"
```

```
##   .. ..- attr(*, "term.labels")= chr "lstat"
##   .. ..- attr(*, "order")= int 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(medv, lstat)
##   .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. ..- attr(*, "names")= chr [1:2] "medv" "lstat"
##  $ model        :'data.frame':   506 obs. of  2 variables:
##   ..$ medv : num [1:506] 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
##   ..$ lstat: num [1:506] 4.98 9.14 4.03 2.94 5.33 ...
##   ..- attr(*, "terms")=Classes 'terms', 'formula'  language medv ~ lstat
##   .. .. ..- attr(*, "variables")= language list(medv, lstat)
##   .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:2] "medv" "lstat"
##   .. .. .. .. ..$ : chr "lstat"
##   .. .. ..- attr(*, "term.labels")= chr "lstat"
##   .. .. ..- attr(*, "order")= int 1
##   .. .. ..- attr(*, "intercept")= int 1
##   .. .. ..- attr(*, "response")= int 1
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. .. ..- attr(*, "predvars")= language list(medv, lstat)
##   .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. .. ..- attr(*, "names")= chr [1:2] "medv" "lstat"
##  - attr(*, "class")= chr "lm"
```

The summary() Object

Another interesting aspect is that when we create the `summary()` object, we can also extract data from this object, in addition to the data from **lm.fit** for example:

```
lm.sum <- summary(lm.fit) # Store the summary in an object
names(lm.sum) # Try str(lm.sum) on your own too
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

Notice all the data we can extract from the summary object. For example:

```
lm.sum$r.squared
```

```
## [1] 0.5441463
```

```
summary(lm.fit)$adj.r.squared # Adjusted R-Squared
```

```
## [1] 0.5432418
```

```
summary(lm.fit)$coefficients["lstat","Pr(>|t|)"] # p-value for lstat
```

```
## [1] 5.081103e-88
```

Notice that the `$coefficients` of **lm.sum** are different than those of **lm.fit**. The former is a data frame with coefficients, standard errors, p-values, etc. Let's say that you want to print standard errors for each coefficient. You just need to get the second column of the dataframe:

```
lm.sum$coefficients # The whole data frame
```

```
##               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
```

```
cat("\n")
```

```
lm.sum$coefficients[, 2] # Just the standard error in the second column
```

```
## (Intercept)       lstat
##  0.56262735  0.03873342
```

Now, let's do a little computational statistics and print a table with the actual 95% confidence interval, which is the coefficient +/- 2 standard errors:

```
# Coefficient - 2 * Standard error
low.coef <- lm.sum$coefficients[, 1] - 2 * lm.sum$coefficients[, 2]

# Coefficient + 2 * Standard error
hi.coef <- lm.sum$coefficients[, 1] + 2 * lm.sum$coefficients[, 2]

cbind("From" = low.coef, "To" = hi.coef)
```

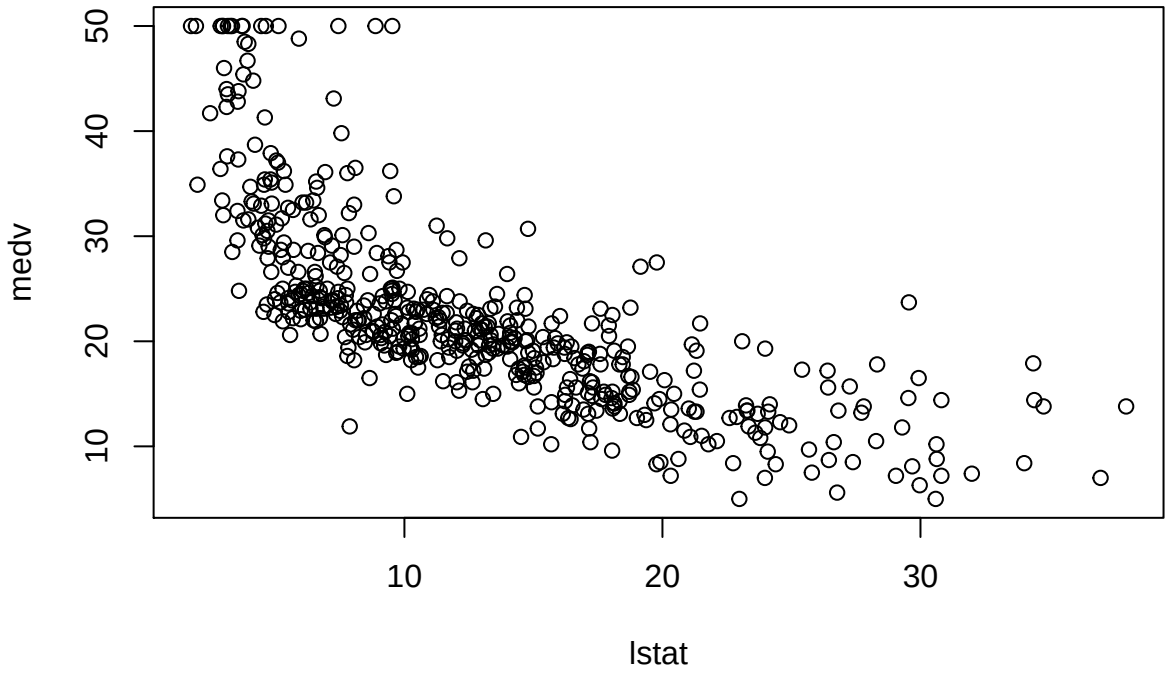
```
##                  From         To
## (Intercept) 33.428586 35.6790956
## lstat       -1.027516 -0.8725825
```

Useful Regression Plots

Residual Plots

The `plot()` function is what we call a **polymorphic** function. It sounds strange, but it means something simple. It means that the function will behave differenetly, depending what type of object we feed into it. For example, this function will simply yield a scatterplot:

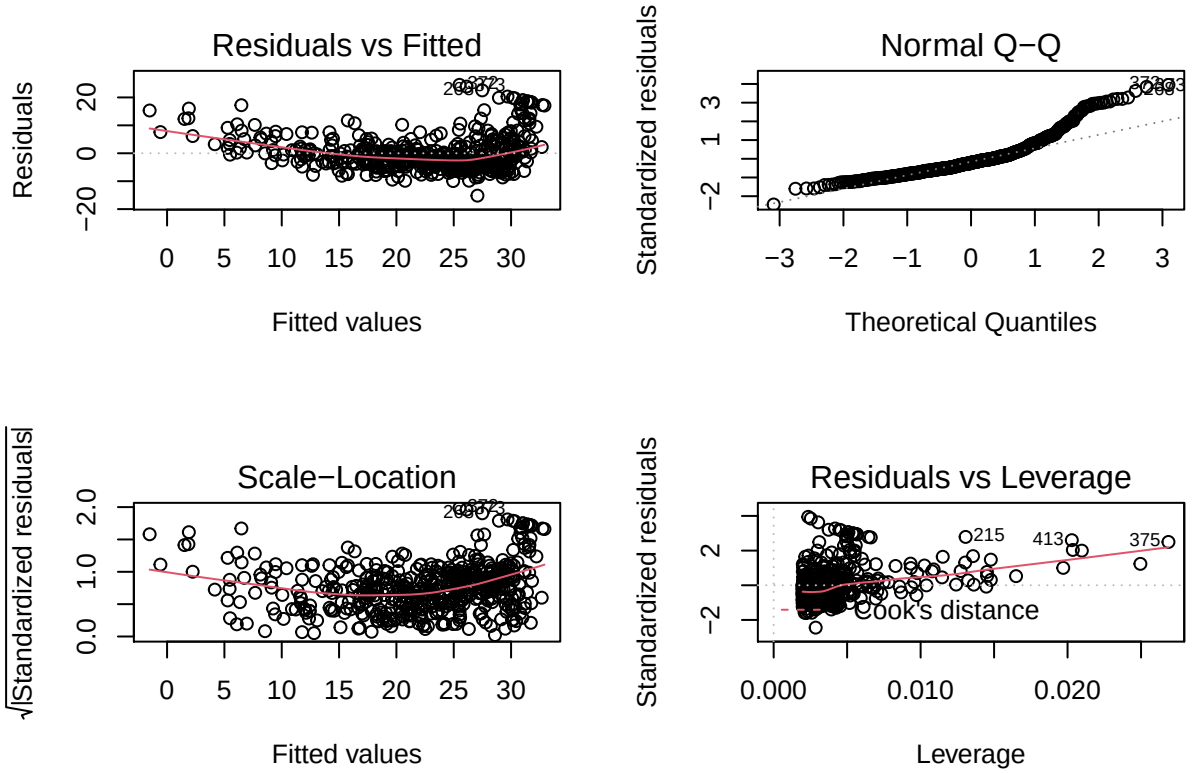
```
plot(medv ~ lstat,
     data = Boston)
```



To get general help on the `plot()` function, you can type `?plot` in the console. But since `plot()` is **polymorphic** you can get specific help for the type of object you want to plot. For example, to get help on `plot()` for `lm()` objects, type `?plot.lm`. In general, a `?function.` followed by the object type, will provide help specific to that object.

But the same function, when applied to an **lm()** object yields 4 key residual plots from the regression model. Notice that I subdivide the plot display into 4 panes with the `par()` function, otherwise I would get 4 large plots one after another. Please remember to reset the display to 1 row x 1 column:

```
par(mfrow = c(2, 2)) # Divide the output into a 2x2 frame by row

plot(lm.fit) # Display 4 important regression diagnostic plots
```



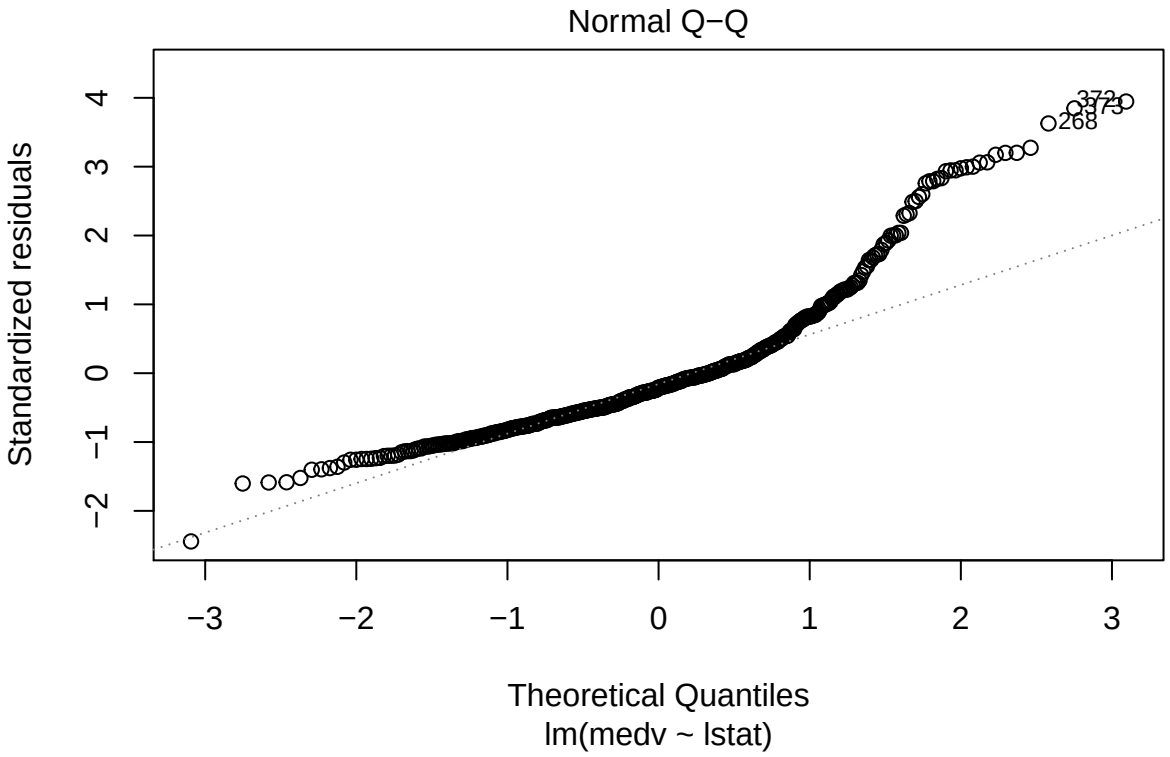

```
par(mfrow = c(1, 1)) # Restore the output window
```

Inspecting regression plots is very important in OLS regression modeling because they provide useful information to evaluate the model assumptions:

1. Residual plot (inspect dispersion of residuals)
2. Residual qqplot (inspect normality of resituals)
3. Standardized residual (square root) plot – same as (1) but residuals are divided by their standard deviations
4. shows the influence of outliers – observations that fall outside the dotted lines (Cook's distance) have too much influence on the slope of the regression line.

If you are only interested in one of the plots, say the second one, you use `which = 2`:

```
plot(lm.fit, which = 2)
```

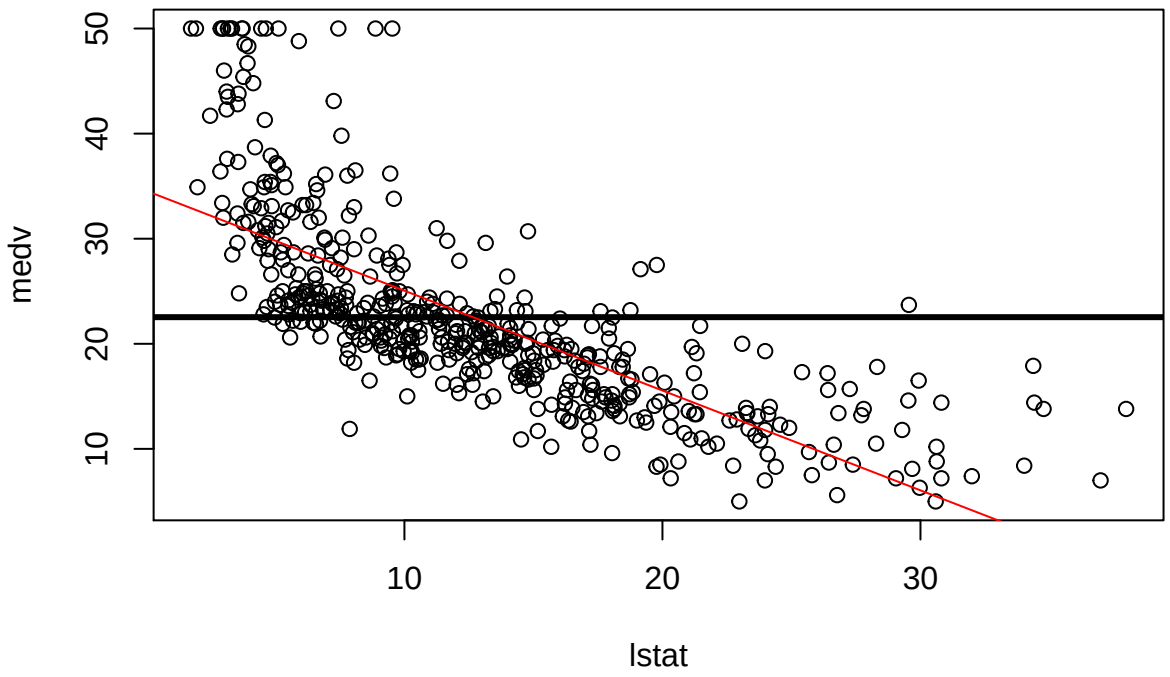


You can also inspect if the residuals are normally distributed with a histogram:

```
hist(lm.fit$residuals)
```

Plot the Data and Regression Line

```
plot(medv ~ lstat,
     data = Boston) # Plot the (x,y) data

abline(a = mean(Boston$medv),
       b = 0,
       lwd = 3) # Line at the mean of medv

abline(lm.fit, col = "red") # Draw a red regression line through the plot
```



You can change the color of the line (e.g., `col = "red"`), the thickness (e.g., `lwd=3`), the type (e.g., `lty="dotted"`), dot size (e.g., `pch=20`), or symbol to use instead of a dot (e.g., `pch="+"`). Try these on your own. But note that you can only visualize regression lines with a single predictor.

Making Predictions With the Fitted Model

The `predict()` function displays predicted values from a model for all data points. I'm just displaying the first 12 belos:

```
predict(lm.fit)[1:12]
```

```
##         1         2         3         4         5         6         7         8
## 29.822595 25.870390 30.725142 31.760696 29.490078 29.604084 22.744727 16.360396
##         9        10        11        12
##  6.118864 18.307997 15.125332 21.946686
```

If you also want to display the lower and upper bounds of the respective 95% confidence intervals for these predictions, use (note that we use a comma at the end to get all 3 columns of the `predict()` object:

```
predict(lm.fit,
        interval = "confidence")[1:12,]
```

```
##          fit       lwr       upr
## 1  29.822595 29.025299 30.619891
```

```
## 2  25.870390 25.265246 26.475534
## 3  30.725142 29.873477 31.576807
## 4  31.760696 30.843594 32.677798
## 5  29.490078 28.712077 30.268079
## 6  29.604084 28.819515 30.388652
## 7  22.744727 22.201573 23.287882
## 8  16.360396 15.626114 17.094677
## 9   6.118864  4.696433  7.541295
## 10 18.307997 17.668272 18.947722
## 11 15.125332 14.321106 15.929557
## 12 21.946686 21.401770 22.491601
```

IF you want predicted values with confidence intervals for 3 new data points for lstat of, say 5, 10 and 15, you have 2 opetions:

```
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
```

```
cat("\n")
```

```
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
```

`interval = "confidence"` provides the range of values representing a 95% probability (by default) that the model will predict values within that interval. If the interval does not include 0, then the coefficients are significant at the $\mathbf{p < 0.05}$ level. This is your confidence interval of what the model is likely to predict.

To get other confidence levels, e.g. 99%, use the `level =` attribute:

```
predict(lm.fit,
        data.frame(lstat = (c(5,10,15))),
        interval = "confidence",
        level = 0.99)
```

```
##        fit      lwr      upr
## 1 29.80359 28.75578 30.85141
## 2 25.05335 24.29107 25.81562
## 3 20.30310 19.55096 21.05524
```

`interval = "prediction"` works a little different. It provides the 95% probability that an **actual prediction** will fall in the interval range. This is your confidence that an actual prediction will fall in that range. Generally, prediction intervals are larger that confidence intervals because they need to account for the variance in the outcome variable.

Technical note on fitting and predicting models. For now, we are using the full data set (e.g., Boston) to fit the model and to make predictions. Later on, we will split the data set into training (e.g., Boston[train,]) and test (e.g., Boston[-train,]) subsamples, and then fit or train the model with the training subsample and test it with the test subsample. This is called **cross-validation** which is central to **machine learning**.

Quick Simple Regression Examples

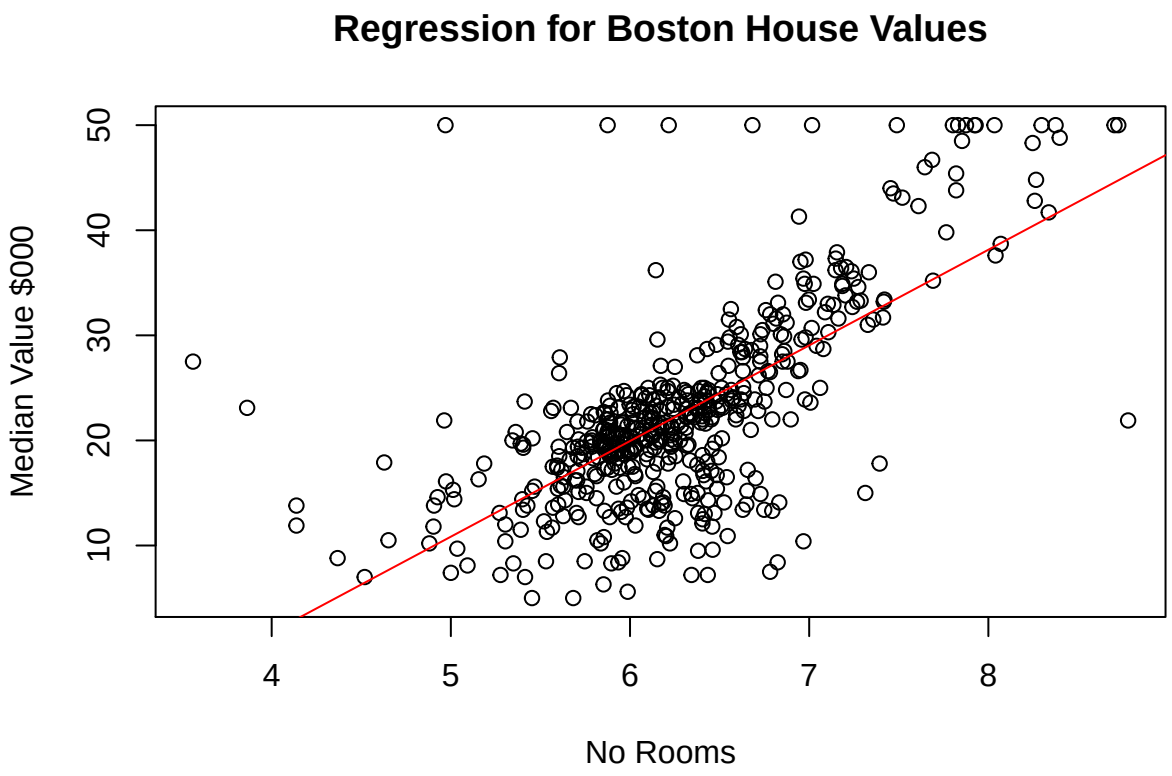
```
library(MASS) # Contains the Boston dataset

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

summary(lm.fit)
```

Boston Housing Dataset

```
##
## Call:
## lm(formula = medv ~ rm, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.346  -2.547   0.090   2.986  39.433
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -34.671      2.650  -13.08   <2e-16 ***
## rm             9.102      0.419   21.72   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.616 on 504 degrees of freedom
## Multiple R-squared:  0.4835, Adjusted R-squared:  0.4825
## F-statistic: 471.8 on 1 and 504 DF,  p-value: < 2.2e-16
```

Let's do some plotting:

```
plot(medv ~ rm,
     data = Boston,
     main = "Regression for Boston House Values",
     xlab = "No Rooms", ylab = "Median Value $000")

abline(lm.fit, col = "red" )
```




```
library(ISLR) # Contains the Hitter dataset

Hitters <- na.omit(Hitters) # Need to remove several omitted values

lm.fit <- lm(Salary ~ HmRun,
             data = Hitters) # Using home runs to predict salaries

summary(lm.fit) # Home runs has a positive significant effect
```

Hitters Baseball Player Salary Dataset

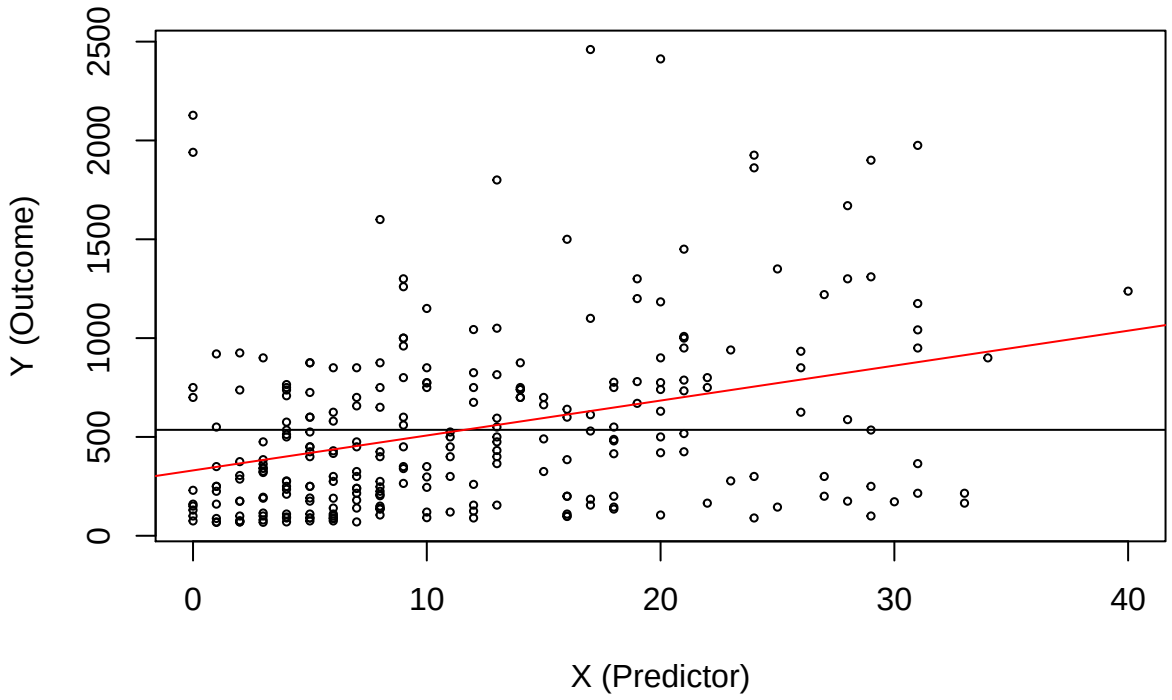
```
##
## Call:
## lm(formula = Salary ~ HmRun, data = Hitters)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -748.73 -275.49  -79.27  184.72 1829.00
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  330.594     43.551   7.591 5.64e-13 ***
## HmRun         17.671      2.995   5.900 1.13e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 424.6 on 261 degrees of freedom
## Multiple R-squared:  0.1177, Adjusted R-squared:  0.1143
## F-statistic: 34.81 on 1 and 261 DF,  p-value: 0.00000001125
```

The resulst show that home runs have a significant effect on players' salaries. For each additional home run, on average, a player's salary goes up by \$17, 671.

Now, let's plot the regression, first the data points

```r
plot(Salary ~ HmRun,
     data = Hitters,
     cex = 0.5,
     xlab = "X (Predictor)",
     ylab = "Y (Outcome)")

abline(a = mean(Hitters$Salary),
       b = 0) # Line at the mean of medv

abline(lm.fit, col = "red") # Draw the regression line on the plot
```



5. Bi-Variate Regression with a Dummy Variable

In the **Carseats** dataset in the **{ISLR}** library, **Price** is a continuous variable and **US** is a binary variable (`Yes=1`, `No=0`). You can inspect the dataset by typing `?Carseats` on the console.

```
library(ISLR) # Contains the "Carseats" data set

lm.fit <- lm(Sales ~ Price + US,
             data = Carseats)

summary(lm.fit)
```

```
##
## Call:
## lm(formula = Sales ~ Price + US, data = Carseats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.9269 -1.6286 -0.0574  1.5766  7.0515
##
## Coefficients:
##             Estimate Std. Error t value   Pr(>|t|)
## (Intercept) 13.03079    0.63098  20.652    < 2e-16 ***
```

```
## Price       -0.05448    0.00523 -10.416    < 2e-16 ***
## USYes        1.19964    0.25846   4.641 0.00000471 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.469 on 397 degrees of freedom
## Multiple R-squared:  0.2393, Adjusted R-squared:  0.2354
## F-statistic: 62.43 on 2 and 397 DF,  p-value: < 2.2e-16
```

The resulst suggest that, on average and holding car seat origin constant, for every $1 increase in price, the total sales in the aveage location goes down by 54 units. Similarly, on average and holding price constant, a US car seat sells 1,199 more units than a non-US car seat.

Technical note: The binary variable **US** has two possible categorical (i.e., factor) values, **Yes** or **No**. When you model a categorical variable like this, R quantifies the model by creating two dummy variables by combining the variable name along with the respective categorical variables. In this case, it creates one variable called **USNo** with a value of 1 if the car seat is not from the US, and 0 otherwise. It also creates another variable called **USYes** with a value of 1 if the car seat is from the US, and 0 otherwise. Because USNo and USYes are perfectly linearly related (i.e., if one is 1 the other one has to be 0), it drops the first one alphabetically (i.e., USNo) from the model, which is why we see a predictor named **USYes**, not **US**. The intercept captures the effect of USNo (i.e., USYes=0) when all predictors are 0. And the coefficient for USYes captures the difference in sales for US car seats, compared to non-US car seats.

6. Multiple Linear Regression

Just about everything discussed up to this applies to multivariate regressions. We just need to refine the wording in the interpretation a bit. Suppose that, as before, we want to predict median values of houses in Boston counties. But this time we anticipate that **lstat** may not be the only predictor, but that crime rate (**crim**), proportion of homes build prior to 1940 and proximity to the Charles River (**Chas**, 0 or 1) may have an affect on home values. The model syntax is the same as for simple linear models, but now we have more than one predictor, separated with the + operator

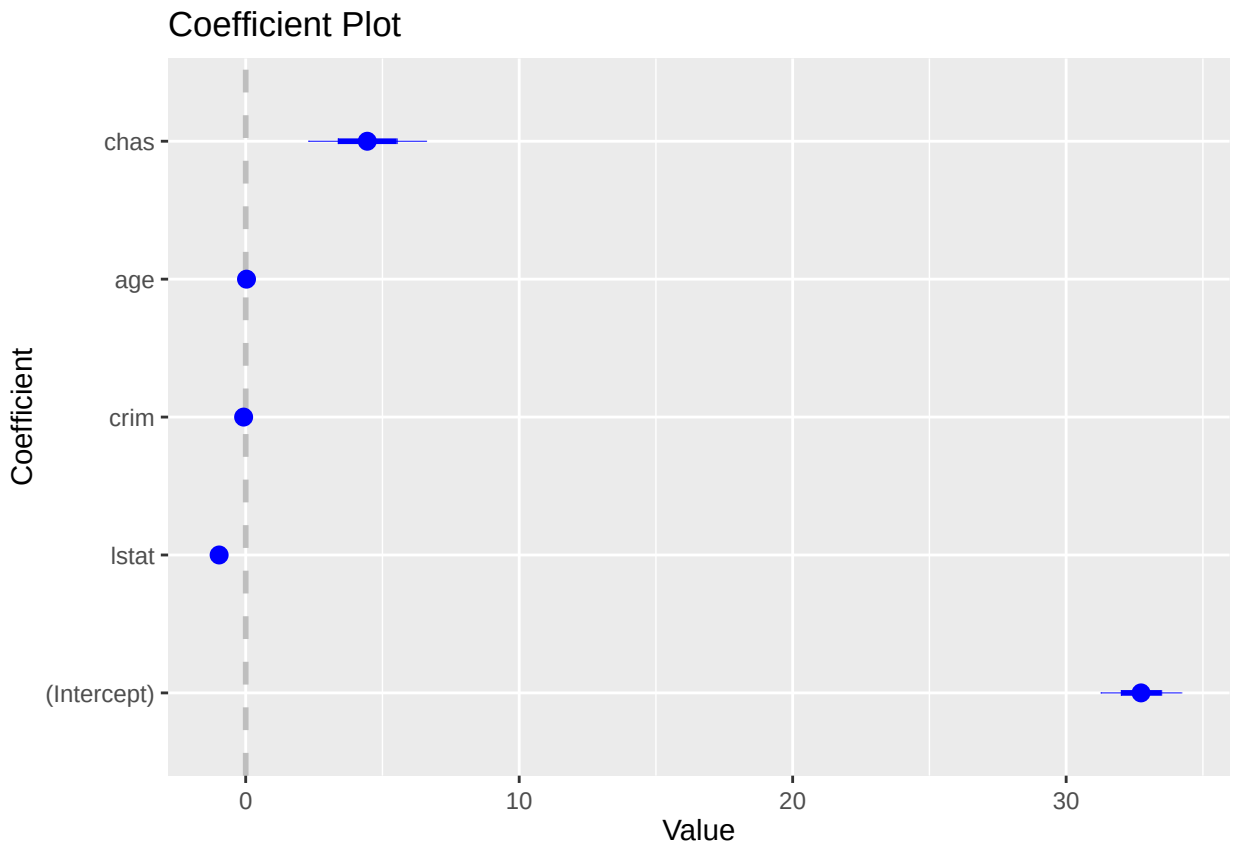
```
lm.fit <- lm(medv ~ lstat + crim + age + chas,
             data = Boston)

summary(lm.fit) # Display regression output summary
```

```
##
## Call:
## lm(formula = medv ~ lstat + crim + age + chas, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.594  -3.834  -1.319   1.932  24.224
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 32.73813    0.73635  44.460   < 2e-16 ***
## lstat       -0.97132    0.05026 -19.326   < 2e-16 ***
```

```
## crim        -0.07492    0.03543  -2.115    0.0350 *
## age          0.02987    0.01220   2.448    0.0147 *
## chas         4.44525    1.07516   4.135 0.0000417 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.051 on 501 degrees of freedom
## Multiple R-squared:  0.5706, Adjusted R-squared:  0.5672
## F-statistic: 166.4 on 4 and 501 DF,  p-value: < 2.2e-16
```

You can inspect the coefficients visually too:

```
library(coefplot)
coefplot(lm.fit)
```



To run a multiple regression on all available variables use the dot **"."**. Also note that we are including the parameter `correlation=T` to display a correlation matrix along with the regression output.

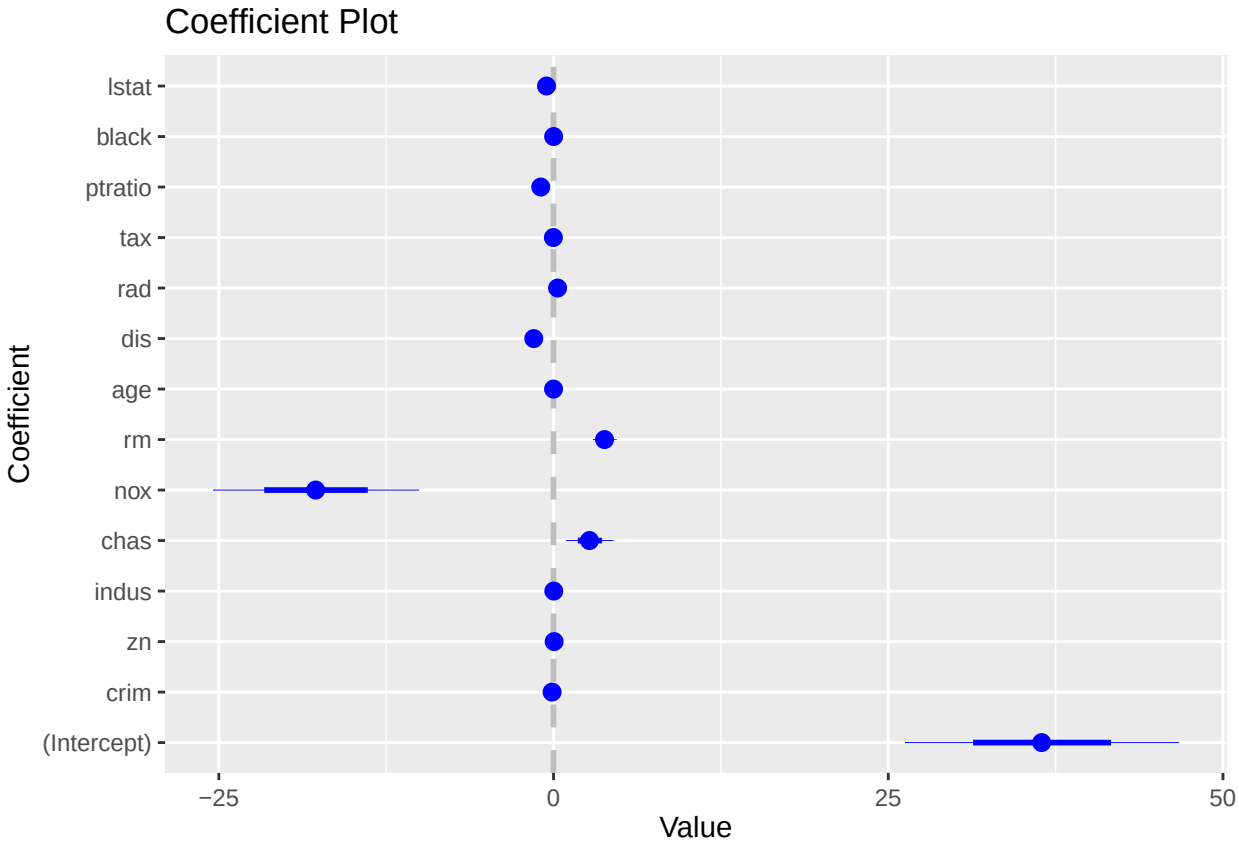
```
lm.fit <- lm(medv ~ .,
             data = Boston)

summary(lm.fit, correlation = T)
```

```
##
```

```
## 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)  36.4594884   5.1034588   7.144 3.28e-12 ***
## crim         -0.1080114   0.0328650  -3.287 0.001087 **
## zn            0.0464205   0.0137275   3.382 0.000778 ***
## indus         0.0205586   0.0614957   0.334 0.738288
## chas          2.6867338   0.8615798   3.118 0.001925 **
## nox         -17.7666112   3.8197437  -4.651 4.25e-06 ***
## rm            3.8098652   0.4179253   9.116  < 2e-16 ***
## age           0.0006922   0.0132098   0.052 0.958229
## dis          -1.4755668   0.1994547  -7.398 6.01e-13 ***
## rad           0.3060495   0.0663464   4.613 5.07e-06 ***
## tax          -0.0123346   0.0037605  -3.280 0.001112 **
## ptratio      -0.9527472   0.1308268  -7.283 1.31e-12 ***
## black         0.0093117   0.0026860   3.467 0.000573 ***
## lstat        -0.5247584   0.0507153 -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
##
## Correlation of Coefficients:
##         (Intercept) crim  zn    indus chas  nox   rm    age   dis   rad   tax
## crim    -0.06
## zn      -0.02       -0.09
## indus    0.06        0.04  0.11
## chas    -0.03        0.05 -0.01 -0.10
## nox     -0.55        0.06  0.04 -0.26 -0.03
## rm      -0.71        0.03 -0.16  0.09 -0.03  0.09
## age      0.08        0.00  0.12  0.00 -0.05 -0.27 -0.21
## dis     -0.36        0.12 -0.40  0.21  0.02  0.28  0.13  0.29
## rad      0.28       -0.27  0.11  0.28 -0.11 -0.14 -0.16  0.08  0.02
## tax     -0.11        0.01 -0.22 -0.44  0.12 -0.07  0.07 -0.03 -0.02 -0.79
## ptratio -0.60        0.02  0.31 -0.13  0.09  0.33  0.16 -0.08 -0.09 -0.19 -0.08
## black   -0.29        0.12 -0.01  0.03 -0.05  0.07  0.10 -0.06  0.02  0.07  0.03
## lstat   -0.30       -0.15 -0.05 -0.08  0.06 -0.07  0.53 -0.34 -0.04 -0.04  0.02
##         ptratio black
## crim
## zn
## indus
## chas
```

```
## nox
## rm
## age
## dis
## rad
## tax
## ptratio
## black   -0.03
## lstat   -0.05    0.16
```

```
coefplot(lm.fit)
```



The respective p-values show that predictors of interest like **chas**, **lstat**, etc. are significant. The effect of **lstat** can now be interpreted as follows: on average and **holding all other variables constant**, when the population of lower income status increases by one percentage point, the median value of houses are $524 lower. The one thing different in this interpretation is the term **holding all other variables constant**, which basically says that the effect of **lstat** is indpendent of the other predictors because we are **controlling for** them.

Naturally, we can access the properties inside **lm.fit** and extract useful information, such as coefficients and confidence intervals:

```
coef(lm.fit) # Display only the regression coefficients
```

```
##    (Intercept)           crim             zn          indus           chas
```

```
##  36.4594883851  -0.1080113578   0.0464204584   0.0205586264   2.6867338193
##            nox             rm            age            dis            rad
## -17.7666112283   3.8098652068   0.0006922246  -1.4755668456   0.3060494790
##            tax        ptratio          black          lstat
##  -0.0123345939  -0.9527472317   0.0093116833  -0.5247583779
```

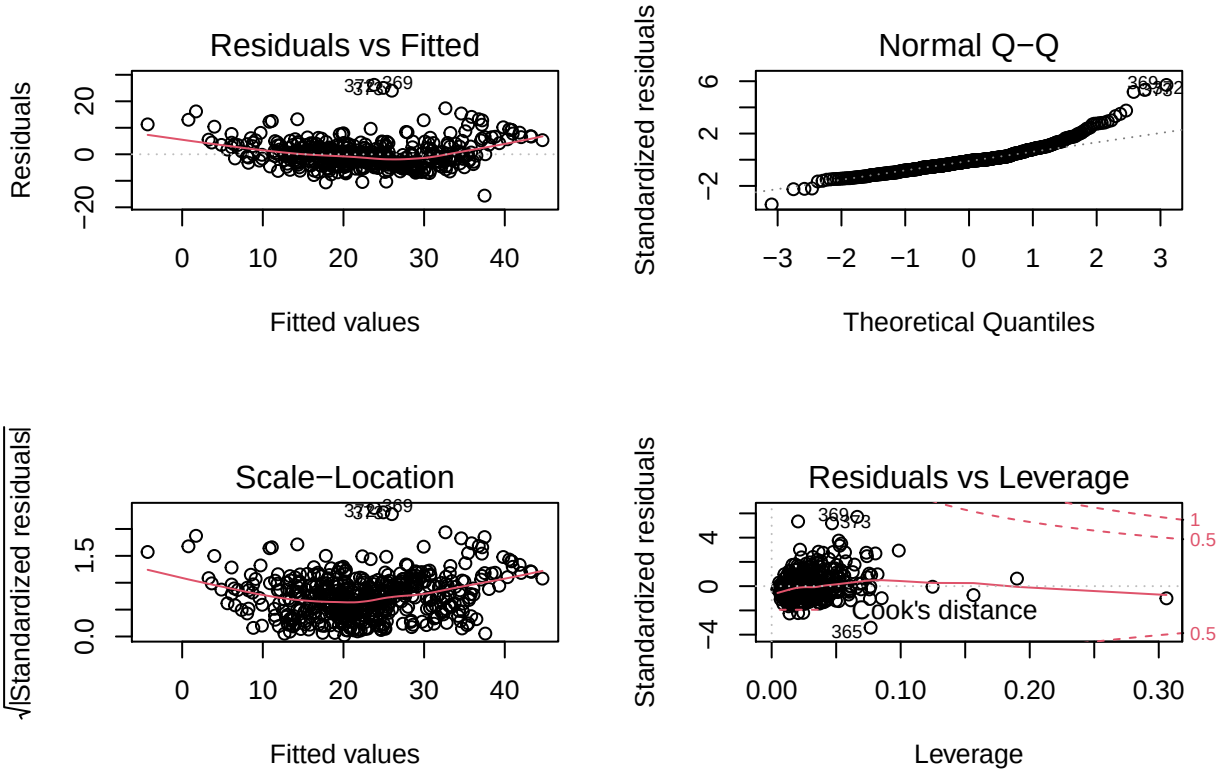
```
cat("\n")
```

```
confint(lm.fit) # Display only the confidence intervals
```

```
##                    2.5 %        97.5 %
## (Intercept)  26.432226009  46.486750761
## crim         -0.172584412  -0.043438304
## zn            0.019448778   0.073392139
## indus        -0.100267941   0.141385193
## chas          0.993904193   4.379563446
## nox         -25.271633564 -10.261588893
## rm            2.988726773   4.631003640
## age          -0.025262320   0.026646769
## dis          -1.867454981  -1.083678710
## rad           0.175692169   0.436406789
## tax          -0.019723286  -0.004945902
## ptratio      -1.209795296  -0.695699168
## black         0.004034306   0.014589060
## lstat        -0.624403622  -0.425113133
```

We can also inspect the residual plots:

```
par(mfrow = c(2, 2))

plot(lm.fit)
```




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