# Linear Regression

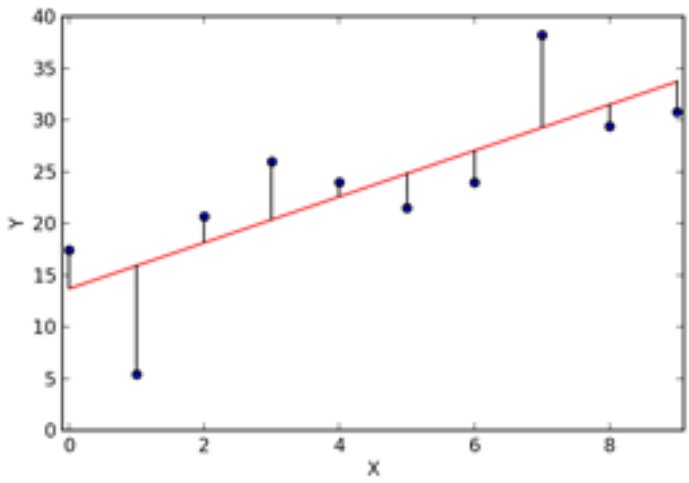
## Short Definition

A linear regression is a statistical model that analyzes the relationship between a response variable (Y) and one or more input variables (X) that are also referred to as predictor or explanatory variables. We can use this model to predict Y when X is known (note that X is a vector of values (x1, x2, ... xn)). The mathematical equation can be generalized as follows:

Y = a + BX

, where $a$ is known as the intercept, whereas $B$ are coefficients of the predictor variables. Collectively, these are referred to as regression coefficients.

Linear regression assumes that there exists a linear relationship between the response variable Y and the explanatory variables X. In case of simple linear regression (only one input variable), this means that one can fit a line that approximates the relation between Y and X.




## Boston housing dataset

Load the required R packages.

```
# load MASS, corrplot and ggplot2
library(MASS)
#install.packages('corrplot')
library(corrplot)
library(ggplot2)
```

A package can include one or multiple datasets. We will use the *Boston* dataset available from the *MASS* package. Let's start by examining the dataset.

```
# examine the structure of the Boston dataset
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 ...
```

```
# check for the presence of NA values
apply(Boston, 2, function(x) sum(is.na(x)))
```

```
##    crim      zn   indus    chas     nox      rm     age     dis     rad     tax
##       0       0       0       0       0       0       0       0       0       0
## ptratio   black   lstat    medv
##       0       0       0       0
```

We will seek to predict *medv* variable (median house value) using (some of) the other 13 variables.

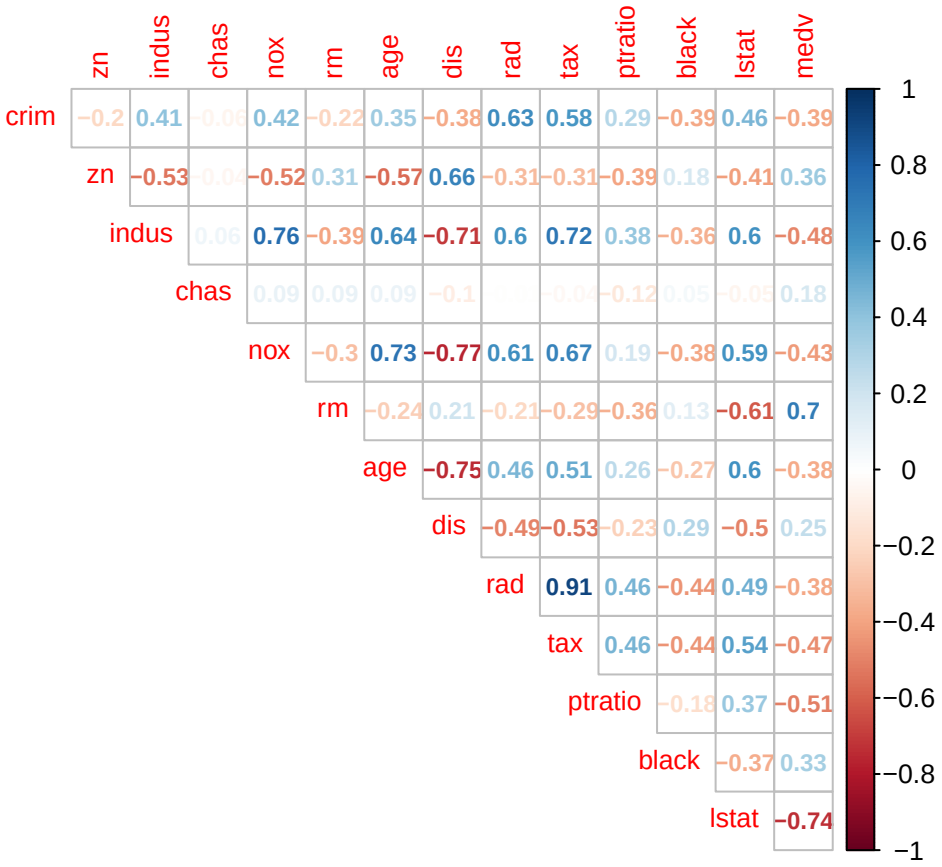
To find out more about the data set, type *?Boston*.

```
# bring out the docs for the dataset
?Boston
```

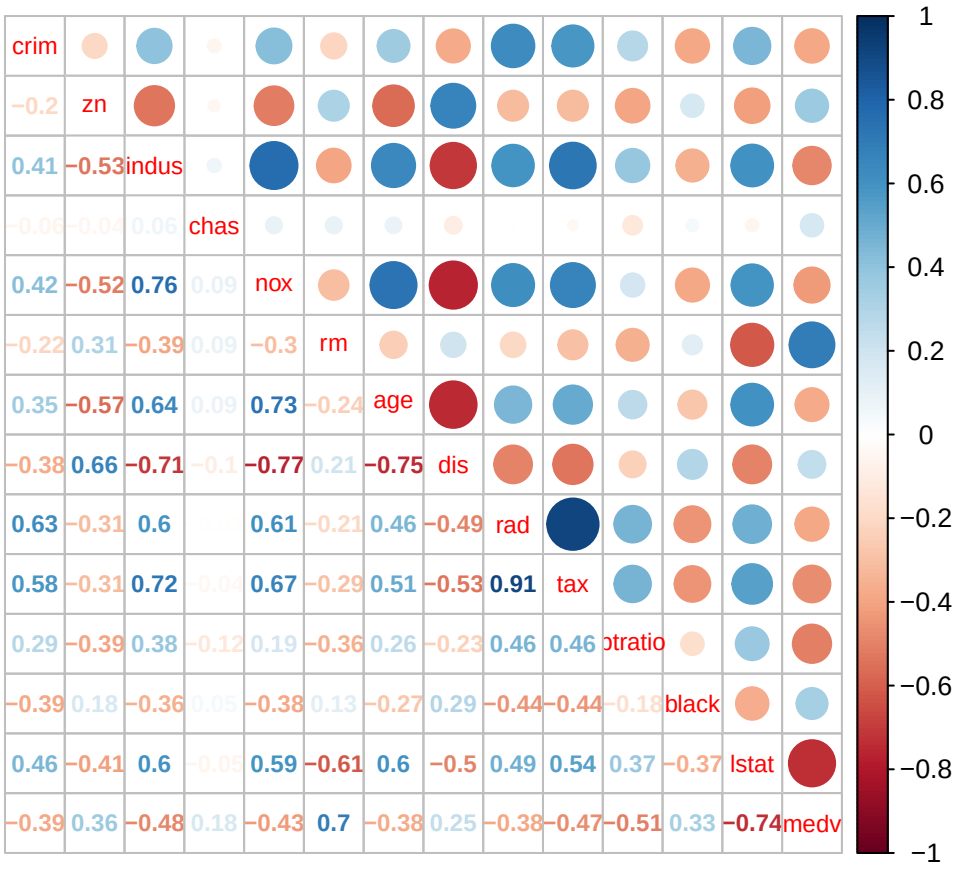
Let's start by examining which of the 13 predictors might be relevant for predicting the response variable (*medv*). One way to do that is to examine the correlation between the predictors and the response variable.

Since we have many variables, examining a correlation matrix will not be that easy. So, it is better to plot the correlations. To that end, we'll use the *corrplot* package (explore the plotting options offered by this package here):

```
# compute the correlation matrix
corr.matrix <- cor(Boston)

# one option for plotting correlations: using colors to represent the extent of correlation
corrplot(corr.matrix, method = "number", type = "upper", diag = FALSE, number.cex=0.75, tl.cex = 0.85)
```



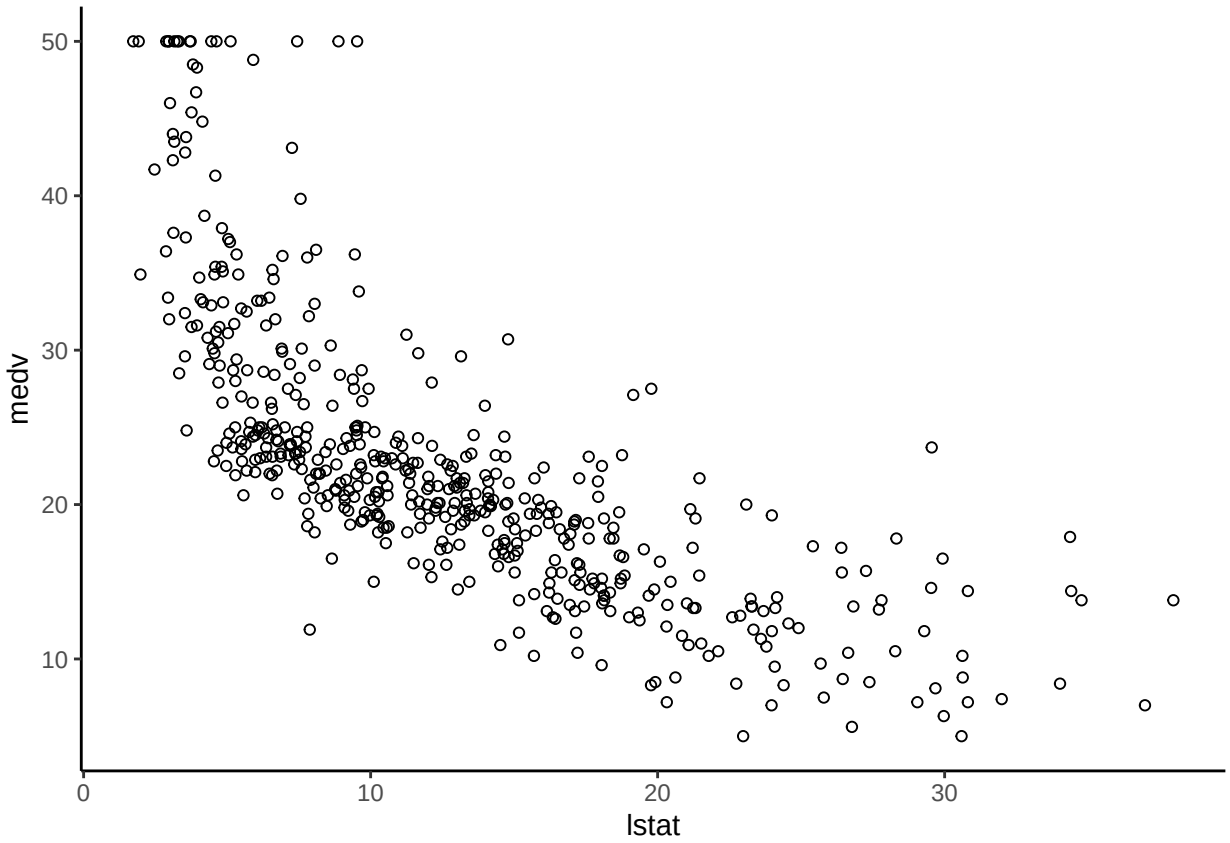

```
# another option, with both colors and exact correlation scores
corrplot.mixed(corr.matrix, tl.cex=0.75, number.cex=0.75)
```



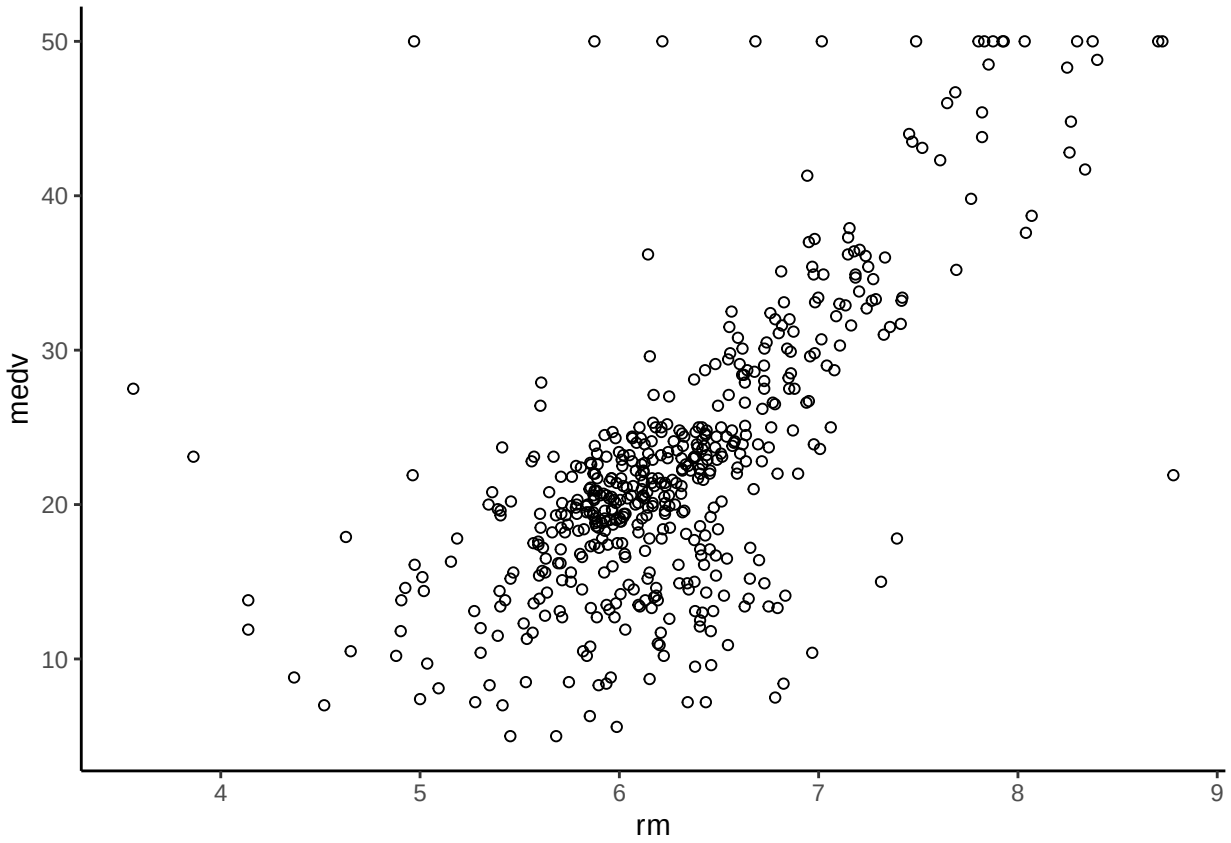

Predictors *lstat* (percent of households with low socioeconomic status) and *rm* (average number of rooms per house) have the highest correlation with the outcome variable.

To examine this further, we can plot variables *lstat* and *rm* against the response variable.

```
#  plot *lstat* against the response variable
ggplot(data = Boston, mapping = aes(x = lstat, y = medv)) +
  geom_point(shape = 1) +
  theme_classic()
```




```
#  plot *rm* against the response variable
ggplot(data = Boston, mapping = aes(x = rm, y = medv)) +
  geom_point(shape = 1) +
  theme_classic()
```




To be able to properly test our model, we need to split our dataset into:

1. **training data** that will be used to build a model,
2. **test data** to be used to evaluate/test the predictive power of our model.

Typically, 80% of observations are used for training and the rest for testing.

When splitting the dataset, we need to assure that observations are randomly assigned to the training and testing data sets. In addition, we should assure that the outcome variable has the same distribution in the train and test sets. This can be easily done using the *createDataPartition()* function from the *caret* package.

```
# install.packages('caret')
library(caret)

# assure the replicability of the results by setting the seed
set.seed(123)

# generate indices of the observations to be selected for the training set
train.indices <- createDataPartition(Boston$medv, p = 0.80, list = FALSE)
# select observations at the positions defined by the train.indices vector
train.boston <- Boston[train.indices,]
# select observations at the positions that are NOT in the train.indices vector
test.boston <- Boston[-train.indices,]
```

Check that the outcome variable (*medv*) has the same distribution in the training and test sets

```r
# print the summary of the outcome variable on both train and test sets
summary(train.boston$medv)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.00   16.95   21.10   22.51   25.00   50.00
```

```r
summary(test.boston$medv)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.00   17.05   21.20   22.62   24.70   50.00
```

## Simple Linear Regression

Let's start by building a simple linear regression model, with *medv* as the response and *lstat* as the predictor. We create this simple model just to showcase the basic functions related to linear regression. In the section *Multiple Linear Regression* we will create a more realistic model that includes multiple variables.

```r
# build an lm model with a formula: medv ~ lstat
lm1 <- lm(medv ~ lstat, data = train.boston)

# print the model summary
summary(lm1)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = train.boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.218  -4.011  -1.123   2.025  24.459
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  34.6527     0.6230   55.62   <2e-16 ***
## lstat        -0.9561     0.0428  -22.34   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.144 on 405 degrees of freedom
## Multiple R-squared:  0.5521, Adjusted R-squared:  0.551
## F-statistic: 499.2 on 1 and 405 DF,  p-value: < 2.2e-16
```

As we see, the *summary()* function gives us (among other things) the following:

- estimated values for the coefficients and their p-values; the latter indicate if a coefficent is significantly different than zero, that is, if the associated input variable is a significant predictor of the response variable

- R-squared ($R^2$) statistic,
- F-statistic for the model.

In particular, we can conclude the following:

- based on the coefficient of the *lstat* variable, with each unit increase in *lstat*, that is, with a percentage increase in the households with low socioeconomic status, median house value decreases by 0.9561 units; since outcome variable is expressed in 1000s USD, that further means that a percentage increase in the low-income households leads to a decrease of 956USD in the median house value
- based on the $R^2$ value, this model explains 55.21% of the variability in the median house value.
- based on the F statistic and the associated p-value, there is a significant relationship between the predictor and the response variable. Note that the F-test is a test of the overall significance of a regression model and it indicates whether the model provides a better fit to the data than a model with no independent variables.

To find out what other pieces of information are stored in the fitted model (that is, the *lm1* object), we can use the *names()* function.

```
# print all attributes stored in the fitted model
names(lm1)
```

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

So, for instance, to get the coefficients of the model:

```
# print the coefficients
lm1$coefficients
```

```
## (Intercept)       lstat
##  34.6527243  -0.9561329
```

Note, there is also the *coef()* function that returns the coefficients:

```
# print the coefficients with the coef() f.
coef(lm1)
```

```
## (Intercept)       lstat
##  34.6527243  -0.9561329
```

The **residual sum of squares (RSS)** is the sum of the squares of residuals (residuals are differences between the predicted and true values of the outcome variable). It measures the amount of variability that is left unexplained after performing the regression.

```
# compute the RSS
lm1_rss <- sum(lm1$residuals^2)
lm1_rss
```

```
## [1] 15285.91
```

Recall that the obtained coefficient values are just estimates (of the real coefficient values) obtained using one particular sample from the target population. If some other sample was taken, these estimates might have been somewhat different. So, we usually compute the **95 confidence interval** for the coefficients to get an interval of values within which we can expect, in 95% of cases (i.e. 95% of examined samples), that the 'true' value for the coefficients will be.

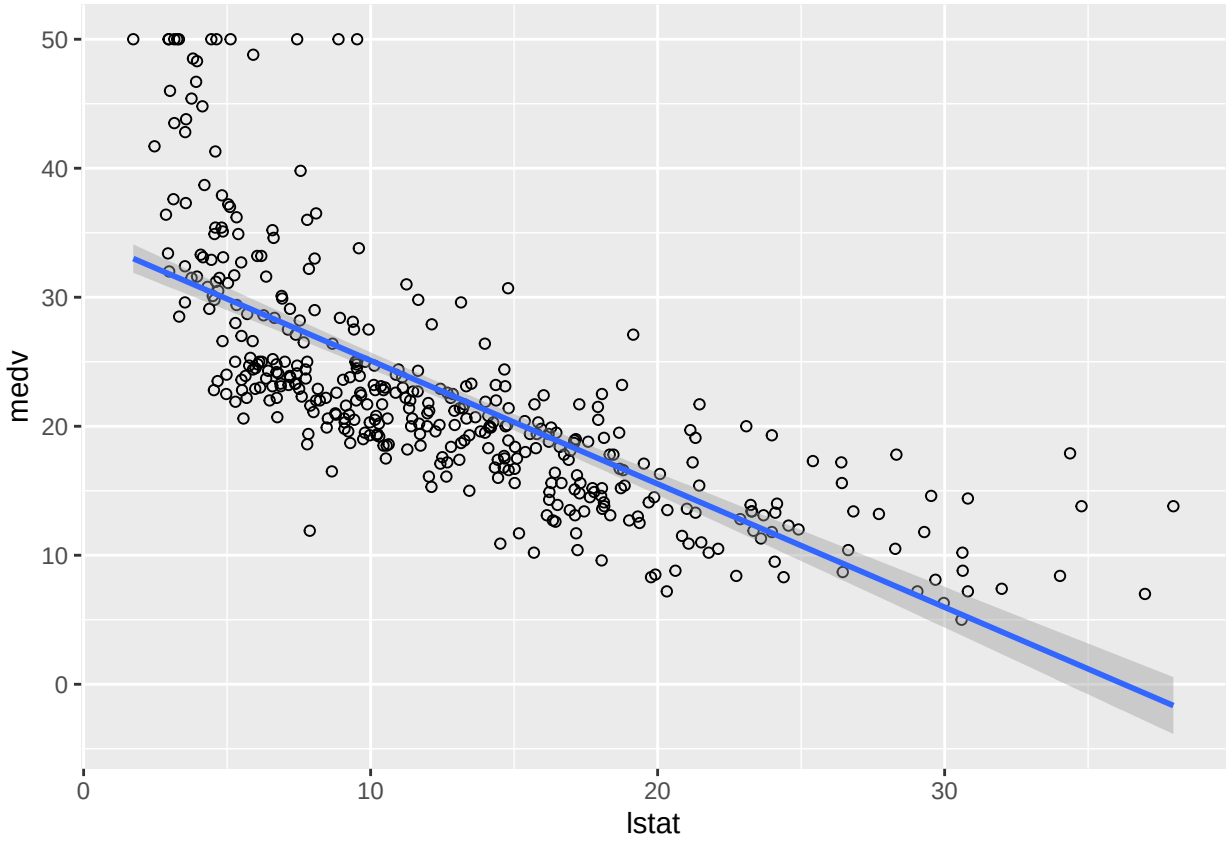
```r
# compute 95% confidence interval
confint(lm1, level = 0.95)
```

```
##                 2.5 %     97.5 %
## (Intercept) 33.428058 35.8773910
## lstat       -1.040262 -0.8720035
```

To visually examine how well our model 'fits the data', we will first plot the data points for the *lstat* and *medv* variables. Next, we will add to the plot the regression line (blue coloured), with the confidence intervals (the gray area around the line).

```r
# plot the data points and the regression line
ggplot(data = train.boston, mapping = aes(x = lstat, y = medv)) +
  geom_point(shape = 1) +
  geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




The plot indicates that there is some non-linearity in the relationship between *lstat* and *medv*.

## Making predictions

Now that we have a model, we can predict the value of *medv* based on the given *lstat* values. To do that, we will use the test data set.

```r
medv_pred <- predict(lm1, newdata = test.boston)
head(medv_pred, 10)
```

```
##         3         6         9        11        14        15        31        32
## 30.799509 29.671272  6.035668 15.099807 26.755067 24.842801 13.044121 22.184752
##        36        41
## 25.397358 32.759581
```

We can also include the confidence interval for the predictions:

```r
# calculate the predictions with the fitted model over the test data, including the confidence interval
medv_pred <- predict(lm1, newdata = test.boston, interval = "confidence")
head(medv_pred, 10)
```

```
##          fit       lwr       upr
## 3  30.799509 29.855947 31.743070
## 6  29.671272 28.802159 30.540385
## 9   6.035668  4.467307  7.604028
## 11 15.099807 14.214615 15.985000
## 14 26.755067 26.049478 27.460655
## 15 24.842801 24.209961 25.475641
## 31 13.044121 12.018367 14.069876
## 32 22.184752 21.585421 22.784082
## 36 25.397358 24.747055 26.047661
## 41 32.759581 31.677166 33.841997
```

Or, we can examine prediction intervals:

```r
# calculate the predictions with the fitted model over the test data, including the prediction interval
medv_pred <- predict(lm1, newdata = test.boston, interval = "predict")
head(medv_pred, 10)
```

```
##          fit        lwr      upr
## 3  30.799509 18.6855113 42.91351
## 6  29.671272 17.5628459 41.77970
## 9   6.035668 -6.1429358 18.21427
## 11 15.099807  2.9902162 27.20940
## 14 26.755067 14.6572784 38.85286
## 15 24.842801 12.7490376 36.93656
## 31 13.044121  0.9234449 25.16480
## 32 22.184752 10.0926954 34.27681
## 36 25.397358 13.3026683 37.49205
## 41 32.759581 20.6339781 44.88518
```

Notice the difference between the confidence and prediction intervals - the latter is much wider, reflecting far more uncertainty in the predicted value. This can be explained as follows:

- **confidence intervals** capture uncertainty due to the error we make when estimating values of the regression coefficients (reducible error);
- **prediction intervals** capture uncertainty that is due to both reducible error and irreducible error, the latter originating in the fact that our linear model is just an approximation of the true relationship between input and output variables (there is that random error in the model that we cannot avoid).

To better understand the difference between the prediction and confidence intervals, you can check this (YouTube video explanation).

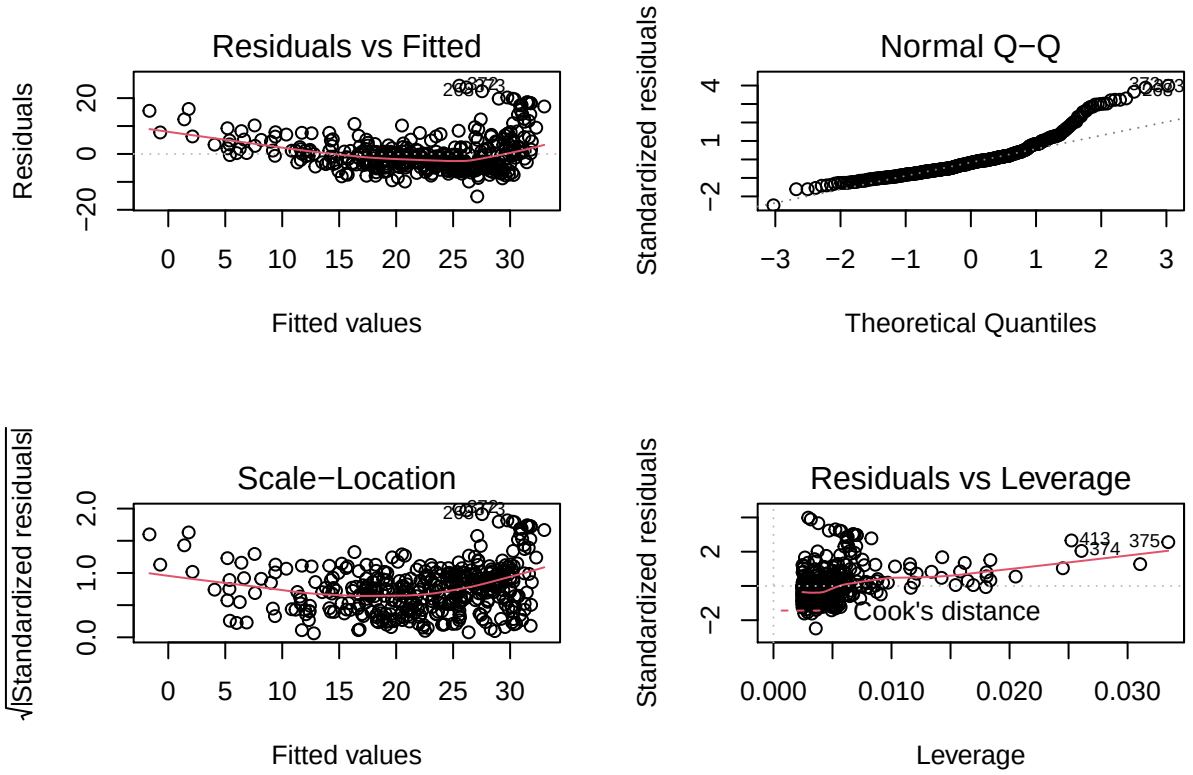
## Diagnostic Plots

Next, we will use *diagnostic plots* to examine how well our model satisfies the key assumptions of linear regression:

1. Linear relationship between the dependent and independent variables,
2. Normallity of residuals,
3. Homoscedasticity, meaning that the variance of residuals remains more-or-less constant across the regression line,
4. No high leverage points (observations that have unusually high / low predictor values are considered to have high leverage).

Four diagnostic plots are automatically produced by passing the output from *lm()* function (e.g. our *lm1*) to the *plot()* function. This 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 a grid of panels so that multiple plots can be viewed simultaneously.

```
# split the plotting area into 4 cells
par(mfrow=c(2,2))

# print the diagnostic plots
plot(lm1)
```



```r
# reset the plotting area
par(mfrow=c(1,1))
```

Interpretation of the plots:

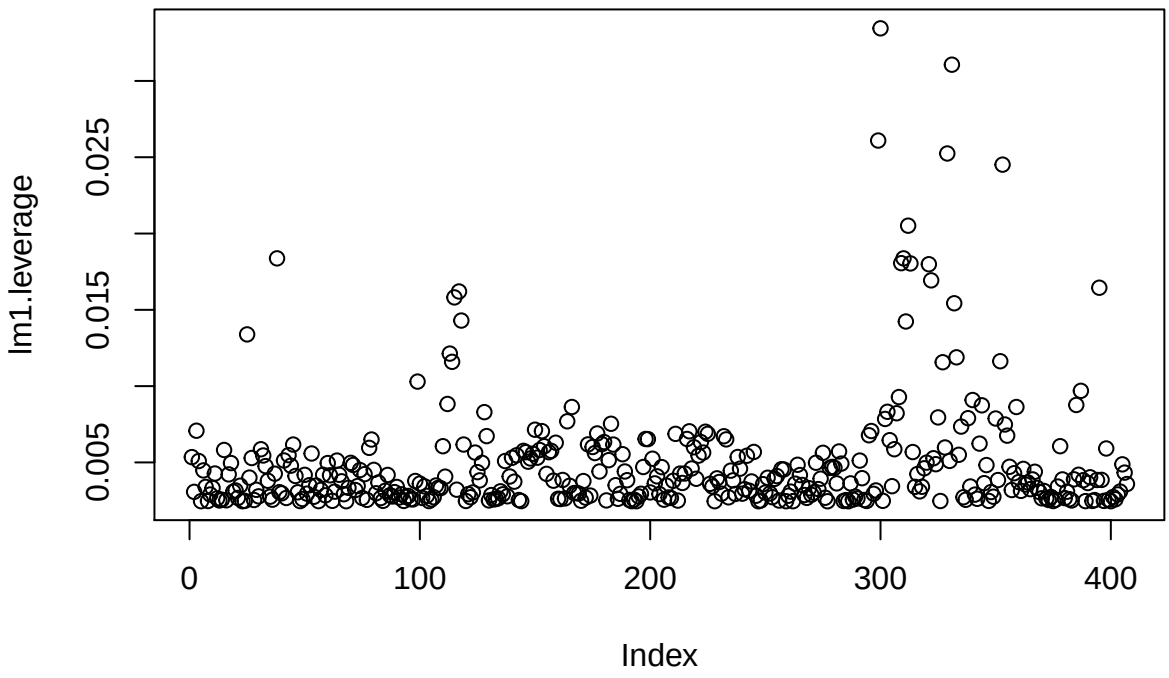
- The 1st plot, **Residual vs Fitted value**, is used for checking if the linearity assumption is satisfied (Assumption 1). A pattern could show up in this plot if there is a non-linear relationship between the dependent and independent variables. If residuals are equally spread around the horizontal line without distinct patterns, that is an indication that there are no non-linear relationships. In this case, the plot indicates a non-linear relationship between the predictors and the response variable.
- The 2nd plot, **Normal Q-Q plot**, tells us if residuals are normally distributed (Assumption 2). The residuals should be lined well on the straight dashed line. In this case, we see a considerable deviation from the diagonal, and therefore, from the normal distribution.
- The 3rd plot, **Scale-Location**, is used for checking the assumption of the equal variance of residuals, homoscedasticity (Assumption 3). It's good if the data points seem to be randomly spread above and below the horizontal line. In this case, the variance of the residuals tends to differ. So, the assumption is not fulfiled.
- The 4th plot, **Residuals vs Leverage**, is used for spotting the presence of high leverage points (Assumption 4). Their presence can seriously affect the estimation of the regression coefficients. They can be typically spotted in the corners, that is, beyond the dashed line that indicates the Cook's distance (they have high Cook's distance scores). In this case, there are several such observations.

Taken all together, the four plots indicate that our linear model, and thus its predictions, are not trustworthy.

For a nice explanation of the diagnostic plots, check the article Understanding Diagnostic Plots for Linear Regression Analysis.

If we want to examine leverage points in more detail, we can compute the leverage statistic using the *hatvalues()* function:

```r
# compute the leverage statistic
lm1.leverage <- hatvalues(lm1)
plot(lm1.leverage)
```




The plot suggests that there are several observations with high leverage values.

We can check this further by examining the value of leverage statistic for the observations. Leverage statistics is always between 1/n and 1 (n is the number of observations); observations with leverage statistic above 2*(p+1)/n (p is the number of predictors) are often considered as high leverage points.

Let's check this for our data:

```
# calculate the number of high leverage points
n <- nrow(train.boston)
p <- 1
cutoff <- 2*(p+1)/n
length(which(lm1.leverage > cutoff))
```

```
## [1] 25
```

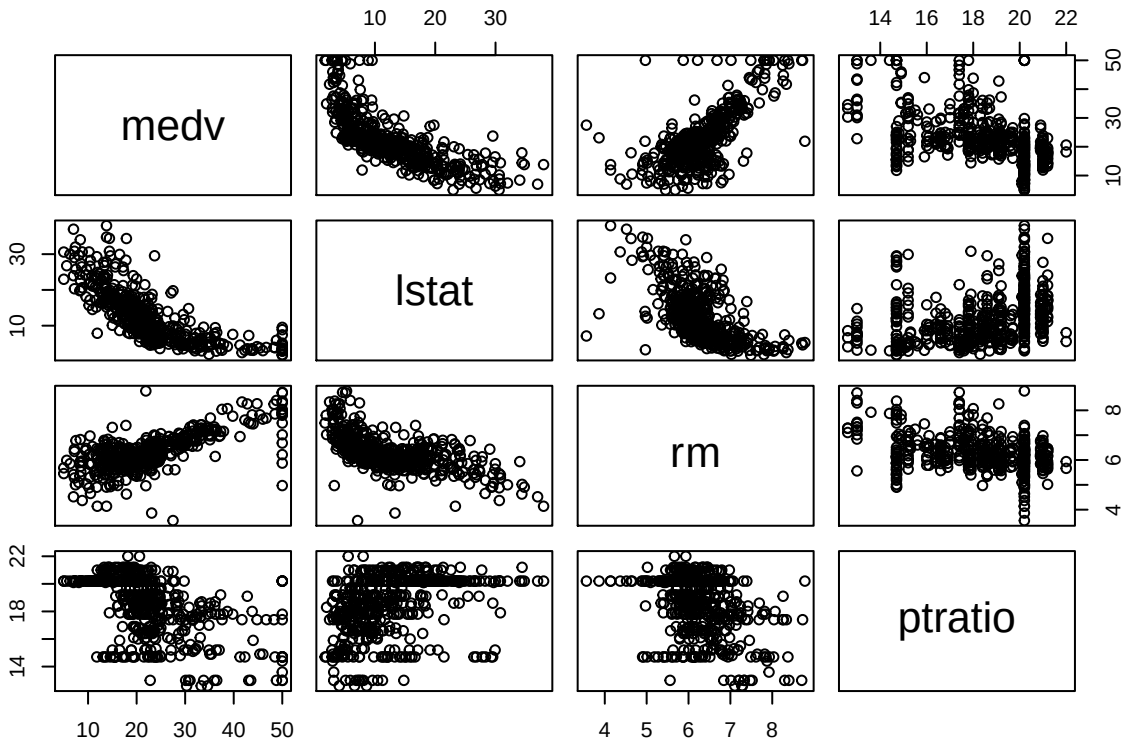
The results confirm that there are several (25) high leverage points.

## Multiple Linear Regression

Let's now extend our model by including more predictor variables that have a high correlation with the response variable. Based on the correlation plot, we can include *rm* (average number of rooms per house) and *ptratio* (pupil-teacher ratio by town) variables into our model.

Scatterplot matrices are useful for examining the presence of a linear relationship between several pairs of variables.

```r
# generate the scatterplots for variables medv, lstat, rm, ptratio
pairs(~medv + lstat + rm + ptratio, data = Boston)
```




Far from proper linear relation, but let's see what the model will look like.

```r
# build an lm model with a train dataset using the formula: medv ~ lstat + rm + ptratio
lm2 <- lm(medv ~ lstat + rm + ptratio, data = train.boston)

# print the model summary
summary(lm2)
```

```
##
## Call:
## lm(formula = medv ~ lstat + rm + ptratio, data = train.boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.6699  -3.0772  -0.8071   1.7431  29.0256
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 21.12353    4.24287   4.979 9.51e-07 ***
## lstat       -0.58411    0.04783 -12.211  < 2e-16 ***
## rm           4.14205    0.45846   9.035  < 2e-16 ***
## ptratio     -0.93222    0.13239  -7.041 8.27e-12 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.244 on 403 degrees of freedom
## Multiple R-squared:  0.6753, Adjusted R-squared:  0.6729
## F-statistic: 279.4 on 3 and 403 DF,  p-value: < 2.2e-16
```

From the summary, we can see that:

- R-squared has increased considerably, from 0.552 to 0.675,
- all 3 predictors are highly significant.

**TASK 1**: Interpret the estimated coefficients (see how it was done for the simple linear regression).

**TASK 2**: Use diagnostic plots to examine how well the model adheres to the assumptions.

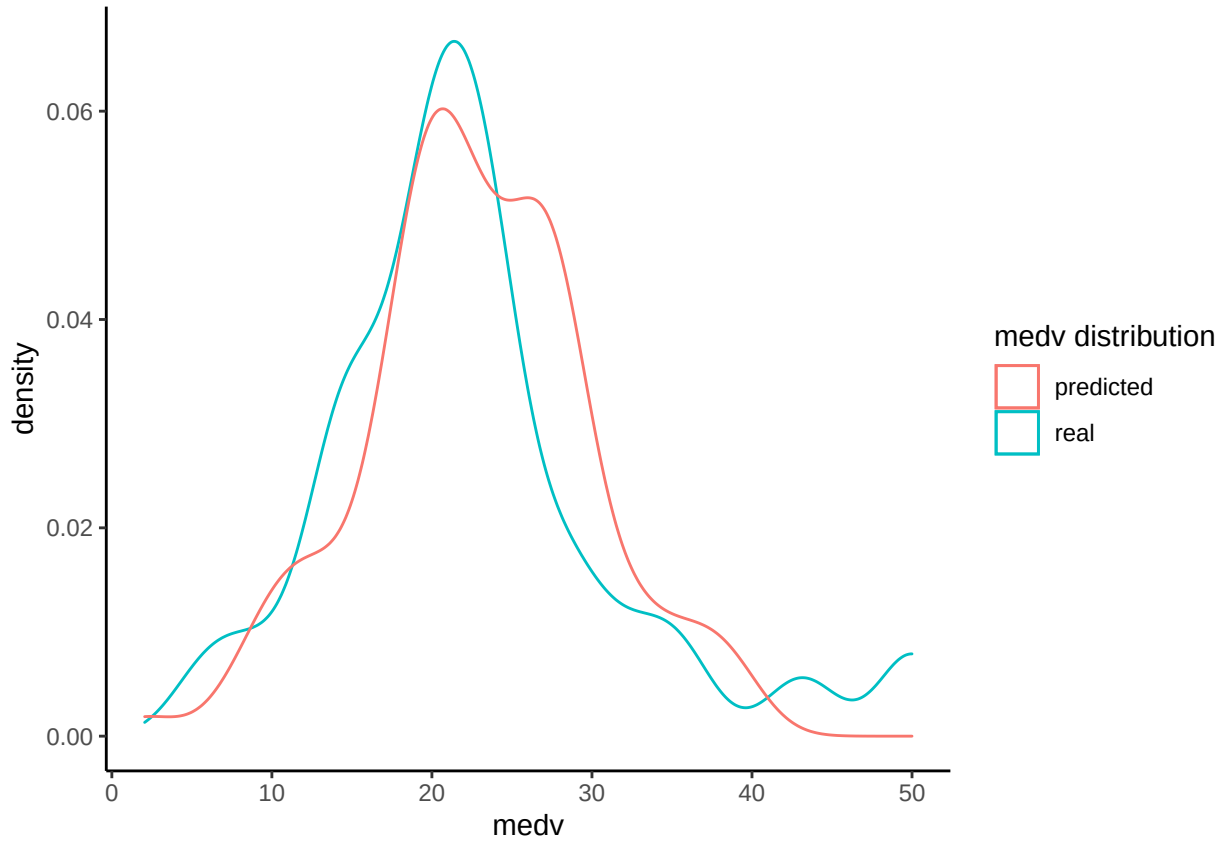
Let's make predictions using this model on the test data set.

```
# calculate the predictions with the lm2 model over the test data
lm2.predict <- predict(lm2, newdata = test.boston)

# print out a few predictions
head(lm2.predict)
```

```
##        3        6        9       11       14       15
## 31.93665 27.28116 12.79533 21.42263 21.36320 20.80387
```

To examine the predicted against the real values of the response variable (*medv*), we can plot their distributions one against the other.

```
# combine the test set with the predictions
test.boston.lm2 <- cbind(test.boston, pred = lm2.predict)

# plot actual (medv) vs. predicted values
ggplot(data = test.boston.lm2) +
  geom_density(mapping = aes(x=medv, color = 'real')) +
  geom_density(mapping = aes(x=pred, color = 'predicted')) +
  scale_colour_discrete(name ="medv distribution") +
  theme_classic()
```




The distributions look similar, but we still fail to fully approximate the real values.

To evaluate the predictive power of the model, we'll compute R-squared on the test data. Recall that R-squared is computed as $1 - \frac{RSS}{TSS}$, where $TSS$ is the total sum of squares.

```
# calculate RSS
lm2.test.RSS <- sum((lm2.predict - test.boston$medv)^2)

# calculate TSS
lm.test.TSS <- sum((mean(train.boston$medv) - test.boston$medv)^2)

# calculate R-squared on the test data
lm2.test.R2 <- 1 - lm2.test.RSS/lm.test.TSS
lm2.test.R2
```

```
## [1] 0.6890174
```

$R^2$ on the test is very similar to the one obtained on the training set, which confirms the stability of our model.

Let's also compute **Root Mean Squared Error (RMSE)** to see how much error we are making with the predictions. Recall: $RMSE = \sqrt{\frac{RSS}{n}}$.

```
# calculate RMSE
lm2.test.RMSE <- sqrt(lm2.test.RSS/nrow(test.boston))
lm2.test.RMSE
```

```
## [1] 5.19483
```

To get a perspective of how large this error is, let's check the mean value of the response variable on the test set.

```
# compare medv mean to the RMSE
mean(test.boston$medv)
```

```
## [1] 22.62424
```

```
lm2.test.RMSE/mean(test.boston$medv)
```

```
## [1] 0.2296134
```

So, it's not a small error, it's about 23% of the mean value.

Let's now build another model using all available variables except the *chas* variable since it is essentially a binary indicator variable and not a number (Note: variables such as *chas* can be included in regression models; however, they require different treatment than numeric variables and thus will be omitted here).

```
# build an lm model with the training set using all of the variables except chas
lm3 <- lm(medv ~ . - chas, data = train.boston)
# note the use of '.' to mean all variables and the use of '-' to exclude the chas variable

# print the model summary
summary(lm3)
```

```
##
## Call:
## lm(formula = medv ~ . - chas, data = train.boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.9992  -2.8700  -0.5507   1.7738  25.9178
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  38.390646   5.648449   6.797 3.97e-11 ***
## crim         -0.099039   0.039343  -2.517 0.012220 *
## zn            0.040367   0.016082   2.510 0.012473 *
## indus         0.006096   0.069584   0.088 0.930239
## nox         -17.017693   4.369094  -3.895 0.000115 ***
## rm            3.531753   0.454061   7.778 6.51e-14 ***
## age           0.011020   0.015599   0.706 0.480347
## dis          -1.407492   0.231948  -6.068 3.04e-09 ***
## rad           0.351924   0.077130   4.563 6.75e-06 ***
## tax          -0.013823   0.004329  -3.193 0.001520 **
## ptratio      -0.996400   0.150522  -6.620 1.18e-10 ***
## black         0.010150   0.002950   3.440 0.000644 ***
## lstat        -0.565604   0.059531  -9.501  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 4.831 on 394 degrees of freedom
## Multiple R-squared:  0.7306, Adjusted R-squared:  0.7224
## F-statistic: 89.04 on 12 and 394 DF,  p-value: < 2.2e-16
```

Note that even though we now have 12 predictors, we haven't much improved the $R^2$ value: in the model with 3 predictors (lm2), it was 0.675 and now it is 0.731. In addition, it should be recalled that $R^2$ increases with the increase in the number of predictors, no matter how good/useful they are.

The 3 predictors from the previous model are still highly significant. Plus, there are several other significant variables.

Considering the number of variables in the model, we should check for **multicolinearity**. *Collinearity* refers to a situation when 2 or more predictors are highly related to one another. A simple way to detect collinearity is to look at the correlation matrix of the predictors. However, it is possible for collinearity to exist between 3 or more variables even if no pair of variables has a particularly high correlation - this is known as *multicollinearity*. One of the assumptions of multiple linear regression is the *absence of multicolinearity*.

One way to check for multicollinearity is to compute the **Variance Inflation Factor (VIF)**:

```
# load the 'car' package since it contains a function for computing VIF
library(car)
```

```
# calculate VIF
vif(lm3)
```

```
##     crim       zn    indus      nox       rm      age      dis      rad
## 1.839134 2.322005 3.906240 4.447346 1.910230 3.210559 4.061814 8.050857
##      tax  ptratio    black    lstat
## 9.481643 1.834001 1.361989 3.129977
```

As a rule of thumb, variables having $\sqrt{vif} > 2$ are problematic.

```
# calculate square root of the VIF and sort the computed values to facilitate inspection
sort(sqrt(vif(lm3)))
```

```
##    black  ptratio     crim       rm       zn    lstat      age    indus
## 1.167043 1.354253 1.356147 1.382111 1.523813 1.769174 1.791803 1.976421
##      dis      nox      rad      tax
## 2.015394 2.108873 2.837403 3.079228
```

We can see that variables *tax*, *rad*, *nox*, and *dis* are probably causing multicollinearity. If we go back to the correlation plot, we'll see that they are, indeed, highly correlated. The *indus* variable is also suspicious since its VIF value is at the very threshold of what is considered acceptable.

We will proceed by excluding variables with the highest VIF value one by one; after excluding each such variable, we will create a new model and check VIF values of the variables in the model. This iterative process will stop once no variable has high VIF value (that is square root of VIF >= 2).

So, we will start by creating a model (lm4) without the *tax* variable, as the variable with the highest VIF value:

```
# build an lm model with the training set using all of the variables except chas and tax
lm4 <- lm(medv ~ . - (chas + tax), data = train.boston)
```

Check the VIF scores again

```
sort(sqrt(vif(lm4)))
```

```
##    black  ptratio     crim       rm       zn      rad    lstat      age
## 1.166126 1.349089 1.356122 1.377518 1.483529 1.722878 1.769121 1.791119
##    indus      dis      nox
## 1.837470 2.015370 2.100705
```

Next, we will exclude *nox* and build a new model (lm5):

```
lm5 <- lm(medv ~ . - (chas + tax + nox), data = train.boston)
```

```
sort(sqrt(vif(lm5)))
```

```
##    black  ptratio     crim       rm       zn      rad    indus      age
## 1.163233 1.276204 1.354019 1.372611 1.482560 1.623895 1.727998 1.729090
##    lstat      dis
## 1.766900 1.942154
```

Now, it's better. Variable *dis* is the edge case, but let's keep it for now and examine the model:

```
summary(lm5)
```

```
##
## Call:
## lm(formula = medv ~ . - (chas + tax + nox), data = train.boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.8054  -2.7733  -0.7237   1.7604  26.1031
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 23.120409   4.784258   4.833 1.93e-06 ***
## crim        -0.089101   0.040539  -2.198  0.02853 *
## zn           0.031008   0.016148   1.920  0.05555 .
## indus       -0.167954   0.062786  -2.675  0.00778 **
## rm           3.809838   0.465381   8.186 3.73e-15 ***
## age         -0.007411   0.015535  -0.477  0.63361
## dis         -1.151434   0.230678  -4.992 8.99e-07 ***
## rad          0.090640   0.045557   1.990  0.04732 *
## ptratio     -0.834520   0.146390  -5.701 2.34e-08 ***
## black        0.011393   0.003035   3.754  0.00020 ***
## lstat       -0.579576   0.061359  -9.446  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.985 on 396 degrees of freedom
## Multiple R-squared:  0.7116, Adjusted R-squared:  0.7043
## F-statistic: 97.71 on 10 and 396 DF,  p-value: < 2.2e-16
```

Note the following in the model's summary: variable *dis* turned out to be highly significant even though it has low correlation with the outcome variable (0.25 - check the correlation plot). Likewise, the *indus* varible that previously proved insignificant, now has become significant. Finally, note that *indus* and *dis* are highly mutually correlated (0.71 - check the correlation plot). This is a clear example of collinearity and that the VIF value of 1.942 for the *dis* variable should have been interpreted as a sign to exclude *dis*; we'll do that now:

```
lm6 <- lm(medv ~ . - (chas + tax + nox + dis), data = train.boston)

summary(lm6)
```

```
##
## Call:
## lm(formula = medv ~ . - (chas + tax + nox + dis), data = train.boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.9534  -3.0362  -0.6671   1.8840  27.1514
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.820797   4.803416   3.710 0.000237 ***
## crim        -0.068517   0.041526  -1.650 0.099737 .
## zn          -0.005297   0.014845  -0.357 0.721401
## indus       -0.054382   0.060255  -0.903 0.367322
## rm           4.003132   0.477532   8.383 9.07e-16 ***
## age          0.023898   0.014635   1.633 0.103270
## rad          0.118685   0.046551   2.550 0.011160 *
## ptratio     -1.020618   0.145764  -7.002 1.08e-11 ***
## black        0.011381   0.003125   3.642 0.000306 ***
## lstat       -0.593742   0.063112  -9.408  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.133 on 397 degrees of freedom
## Multiple R-squared:  0.6935, Adjusted R-squared:  0.6865
## F-statistic: 99.79 on 9 and 397 DF,  p-value: < 2.2e-16
```

Now, we have a model 'cleaned' of multicollinearity.

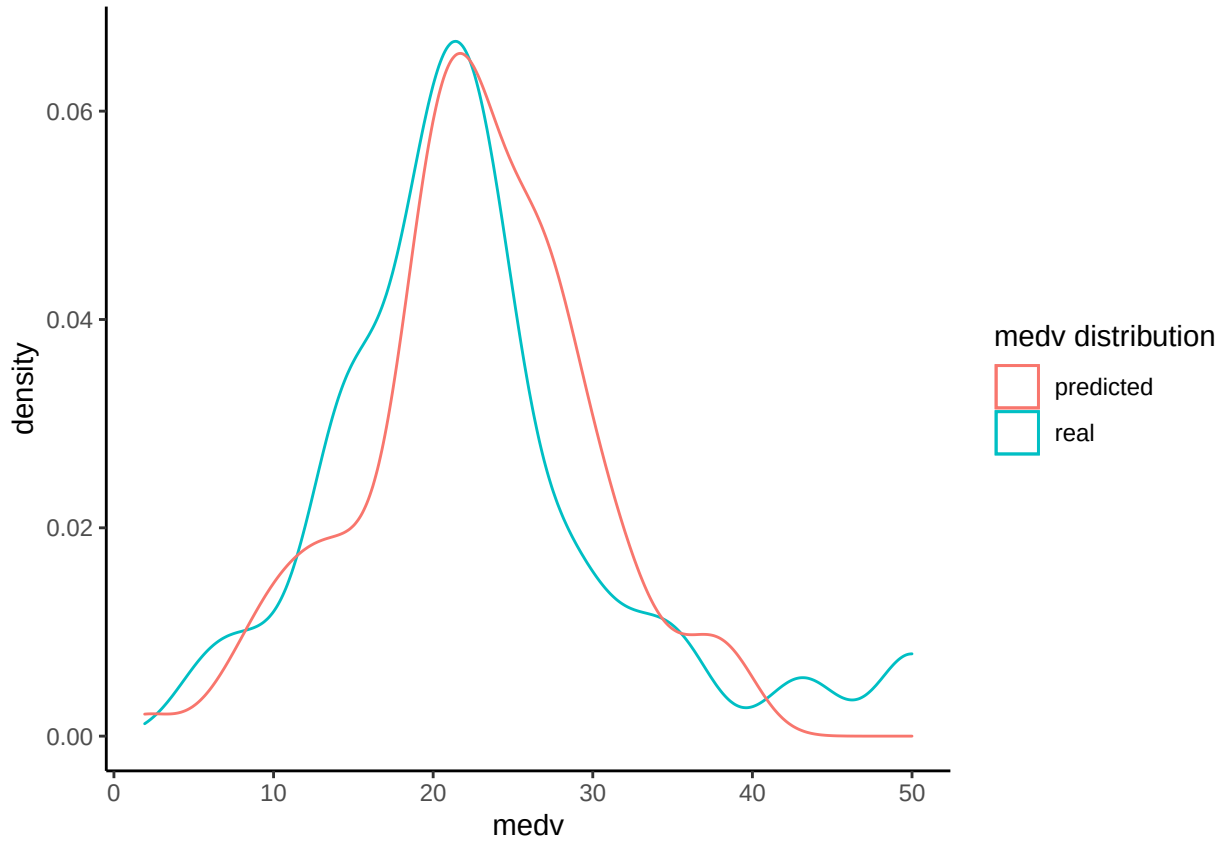
Let's do the prediction using the new model:

```
lm6.predict <- predict(lm6, newdata = test.boston)
```

Plot the distribution of predictions against the real values of the response variable (*medv*).

```
# combine the test set with the predictions
test.boston.lm6 <- cbind(test.boston, pred = lm6.predict)

# plot actual (medv) vs. predicted values
ggplot(data = test.boston.lm6) +
  geom_density(mapping = aes(x=medv, color = 'real')) +
  geom_density(mapping = aes(x=pred, color = 'predicted')) +
```

```
  scale_colour_discrete(name ="medv distribution") +
  theme_classic()
```




The plot suggests that the latest model is somewhat better fit for the data.

As before, we'll compute $R^2$ on the test data.

```
# calculate RSS
lm6.test.RSS <- sum((lm6.predict - test.boston$medv)^2)

# calculate R-squared on the test data
lm6.test.R2 <- 1 - lm6.test.RSS/lm.test.TSS
lm6.test.R2
```

```
## [1] 0.6910706
```

We got almost the same $R^2$ as for the lm2 model.

We can also compute RMSE:

```
# calculate RMSE
lm6.test.RMSE <- sqrt(lm6.test.RSS/nrow(test.boston))
lm6.test.RMSE
```

```
## [1] 5.177652
```

We've got almost the same RMSE as for lm2. Since lm2 is simpler model (less variables), it is the preferred one (recall that simpler model generalise better).

**TASK**: use diagnostic plots to examine how well this model adheres to the assumptions.