Linear Regression

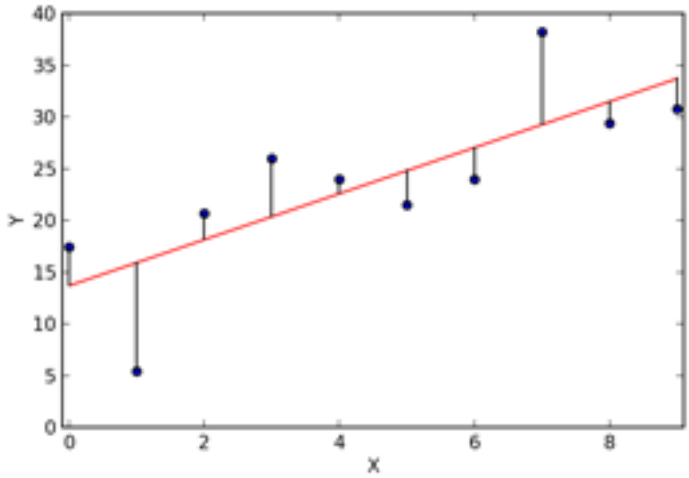
Short Definition

A linear regression is a statistical model that analyzes the relationship between a response variable (often called Y) and one or more variables (often called X or explanatory variables). We can use this model to predict the Y when only the X is known. The mathematical equation can be generalized as follows:

$Y = a + bX$

, where a is the intercept and b is the slope. Collectively, they are called regression coefficients.

Linear regression assumes that there exists a linear relationship between the response variable Y and the explanatory variables X. This means that you can fit a line between the two (or more variables).



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 that is a part of 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 ...
```

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

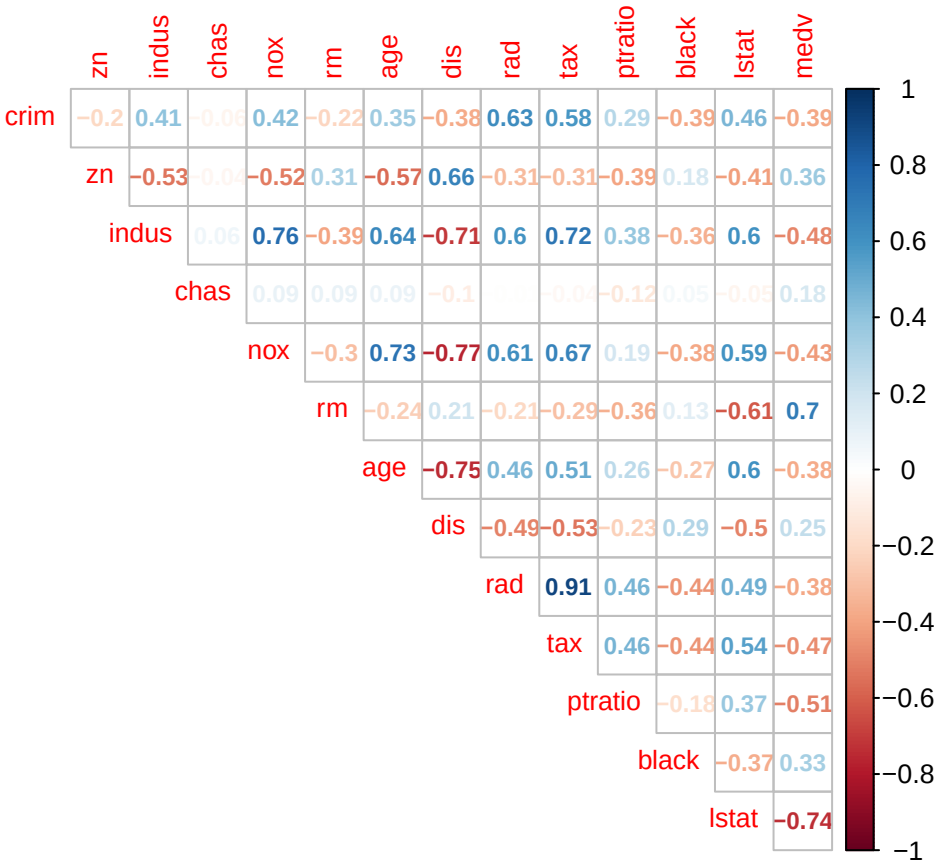
To find out more about the data set, type *?Boston*.

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

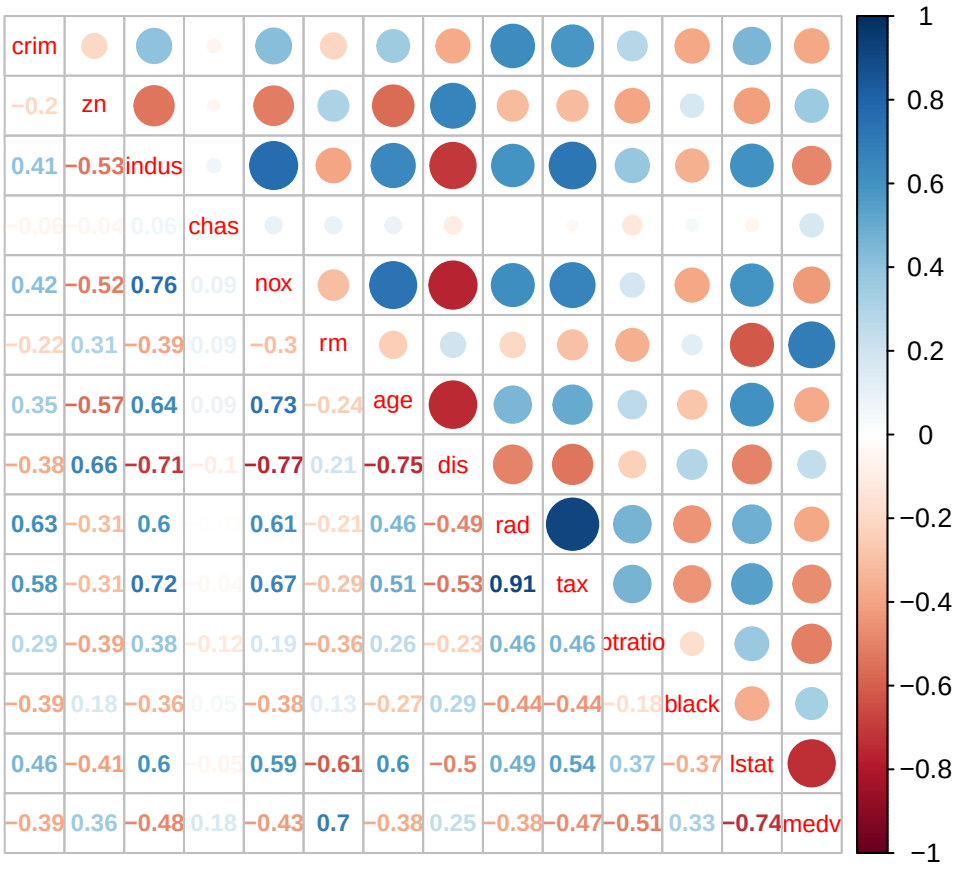
Let's start by examining which of the 13 predictors might be relevant for predicting the response variable (*medv*). One way to do that is to examine the correlation between the predictors and the response variable.

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)
```



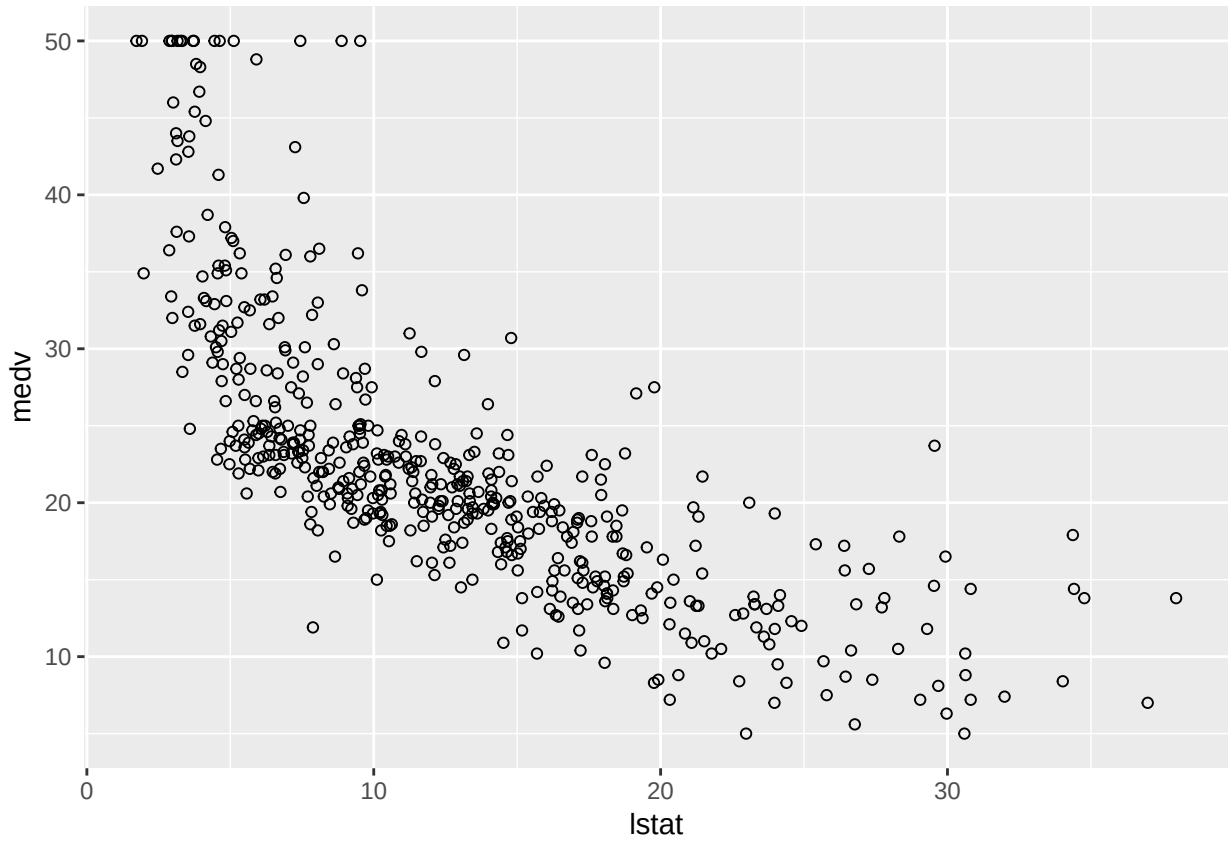

```
# another option, with both colors and exact correlation scores
corrplot.mixed(corr.matrix, tl.cex=0.75, number.cex=0.75)
```



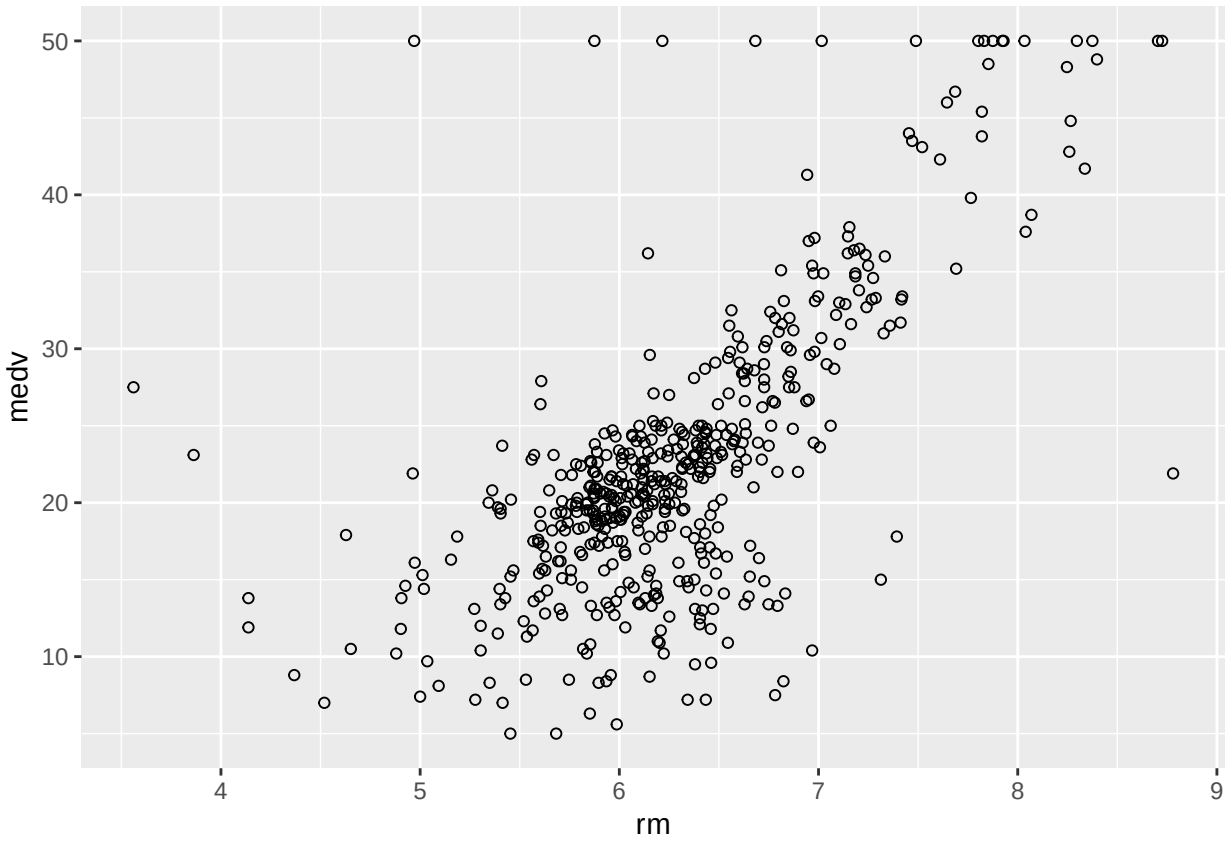
Predictors *lstat* (percent of households with low socioeconomic status) and *rm* (average number of rooms per house) have the highest correlation with the outcome variable.

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

```
#  plot *lstat* against the response variable
p1 <- ggplot(data = Boston, mapping = aes(x = lstat, y = medv)) +
  geom_point(shape = 1)
p1
```




```
#  plot *rm* against the response variable
p2 <- ggplot(data = Boston, mapping = aes(x = rm, y = medv)) +
  geom_point(shape = 1)
p2
```



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 all of the variables.

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

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

```
##
## 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
```

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

- p-values and standard errors for the coefficients,
- 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 0.95005 units.
- based on the R^2 value, this model explains 54.4% of the variability in the median house value.
- based on the F statistic and the associated p-value, there is a significant linear relationship between the predictor and the response variable.

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
##  34.5538409  -0.9500494
```

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

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

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

The **residual sum of squares (RSS)**, also known as the sum of squared residuals, is the sum of the squares of residuals (deviations predicted from actual empirical values of data). It is a measure of the discrepancy between the data and the estimation model.

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

```
## [1] 19472.38
```

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) 33.448457 35.6592247
## lstat       -1.026148 -0.8739505
```

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 create a tiny test data frame.

```r
# create a test data frame containing only lstat variable and three observations: 5, 10, 15
df.test <- data.frame(lstat=c(5, 10, 15))

# calculate the predictions with the fitted model over the test data
predict(lm1, newdata = df.test)
```

```
##        1        2        3
## 29.80359 25.05335 20.30310
```

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
predict(lm1, newdata = df.test, 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
```

Or, we can examine prediction intervals:

```r
# calculate the predictions with the fitted model over the test data, including the prediction interval
predict(lm1, newdata = df.test, interval = "predict")
```

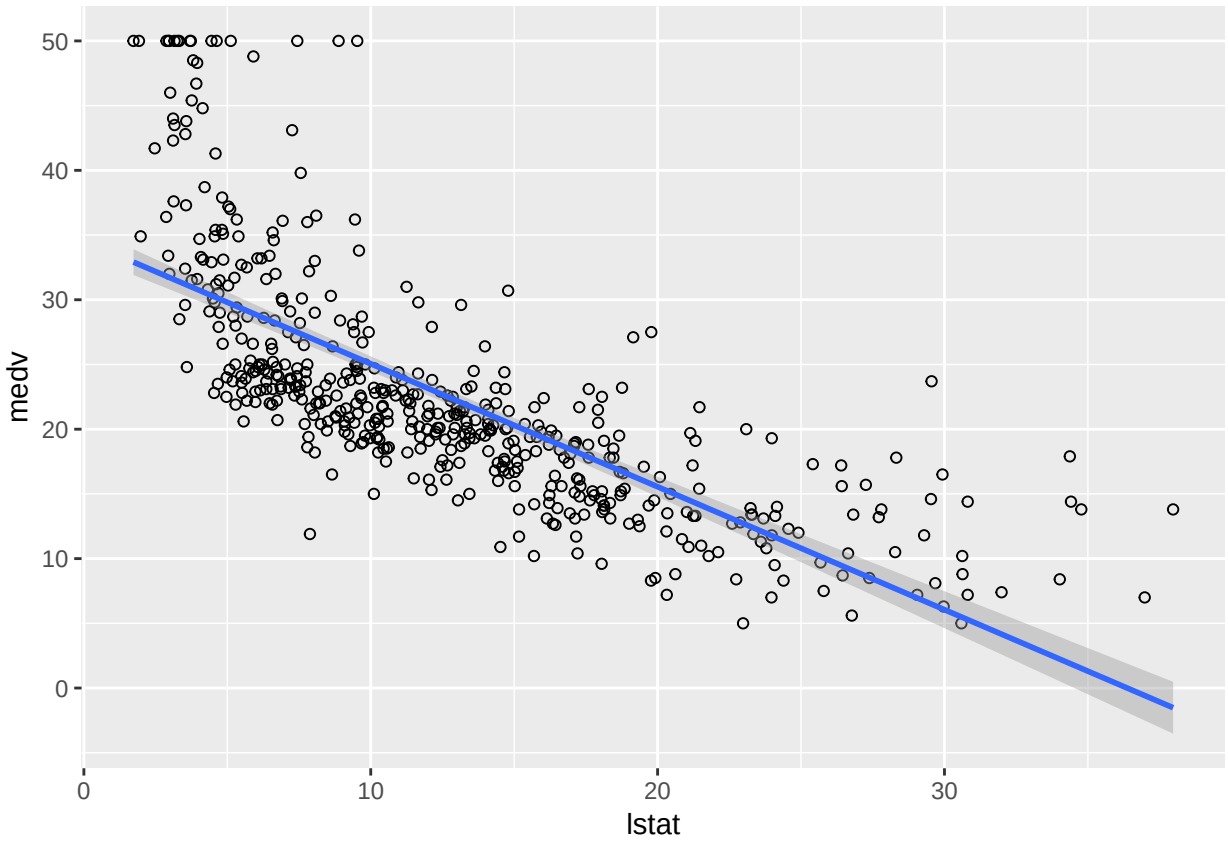
```
##        fit       lwr      upr
## 1 29.80359 17.565675 42.04151
## 2 25.05335 12.827626 37.27907
## 3 20.30310  8.077742 32.52846
```

Notice the difference between the confidence and prediction intervals - the latter is much wider, reflecting far more uncertainty in the predicted value.

Hint: recall the difference between the prediction and confidence intervals (a YouTube video explanation).

Now, we have to examine how well our model 'fits the data'. To do that, 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), and observe how well the regression line fits the data.

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



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

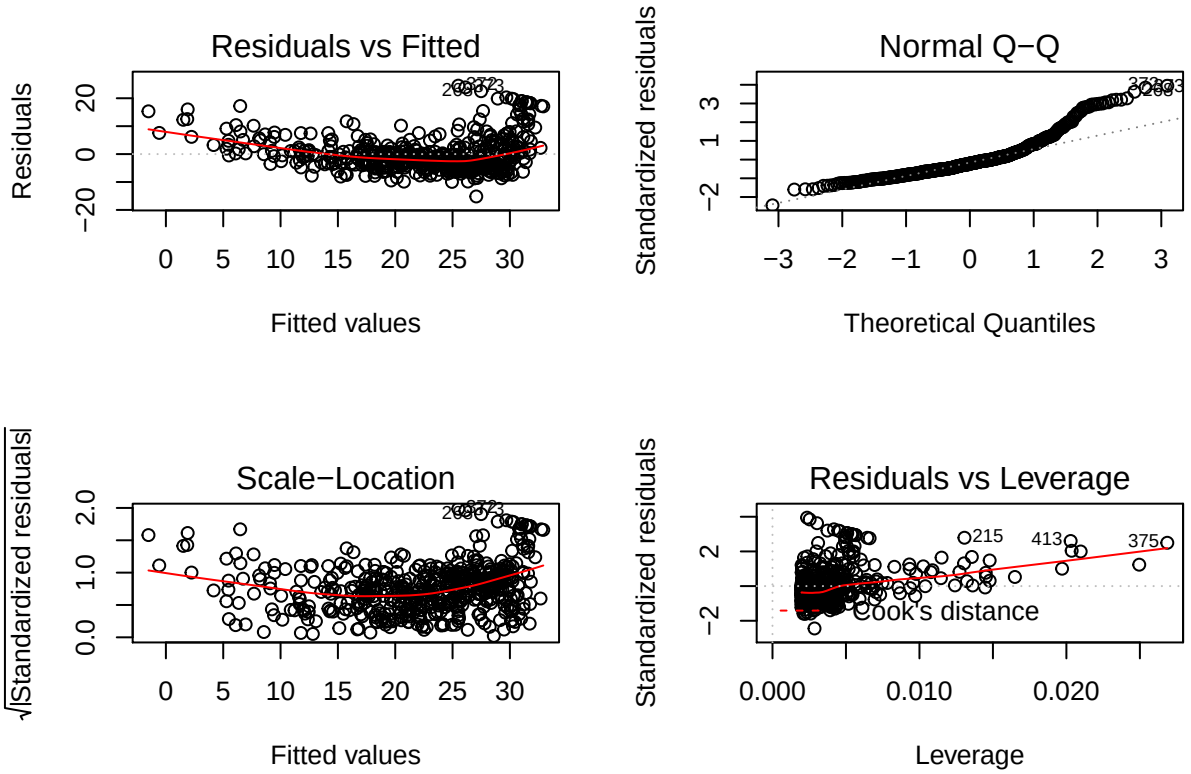
Diagnostic Plots

Next, we will use *diagnostic plots* to examine the model fitness in more detail. The diagnostic plots help in testing the assumptions of the linear regression:

1. Linear relationship between the independent and dependent variables,
2. Normallity of residuals,
3. Homoscedasticity, meaning residuals are equal across the regression line,
4. No or little multicollinearity (multicollinearity occurs when the independent variables are too highly correlated with each other).

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 separate 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)
```




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

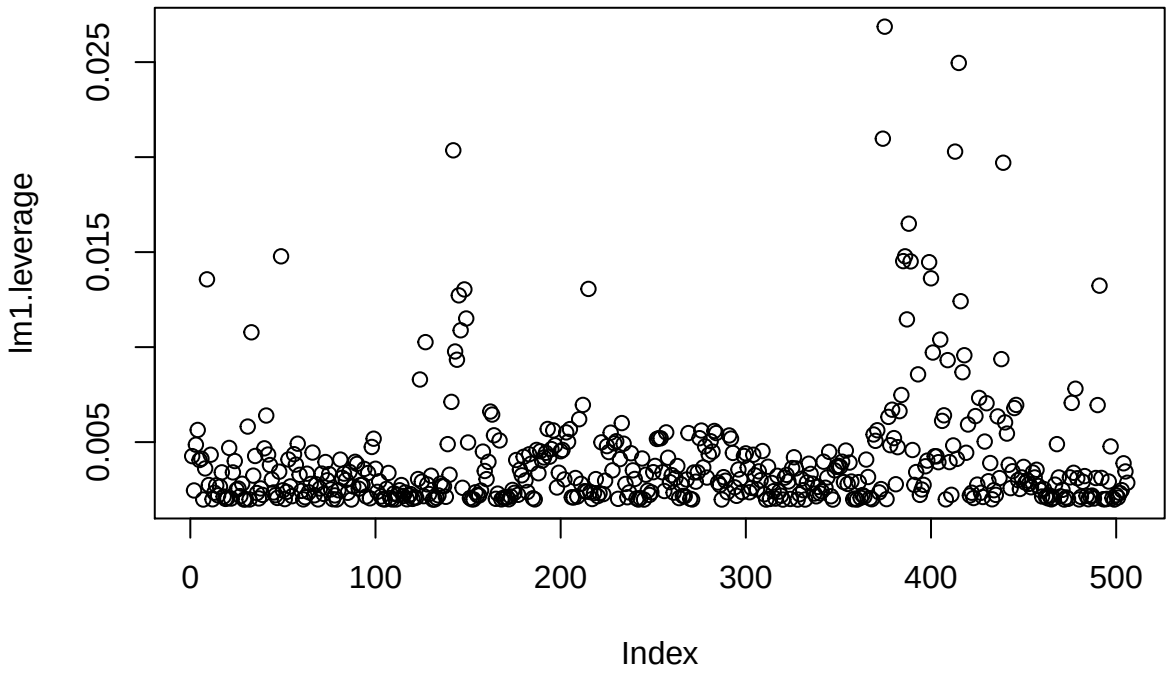
Interpretation of the plots:

- The 1st plot, **Residual vs Fitted value**, is used for checking if the linearity assumption is satisfied (Assumption 1). The pattern could show up in this plot if the model doesn't capture the non-linear relationship. If there are equally spread residuals around a horizontal line without distinct patterns, that is a good indication there are no non-linear relationships. The plot shows that there is some indication of the non-linear relationship between the predictor 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 there is a horizontal line with equally (randomly) spread points. 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; those would be the observations that have an unusually high value of the predictor variable(s). Their presence can seriously affect the estimation of the coefficients. They can be spotted at the upper right corner or at the lower right corner, meaning they are outside of the Cook's distance (they have high Cook's distance scores). In this case, there are several such observations.

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:

```
# 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 considerably 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(Boston)
p <- 1
cutoff <- 2*(p+1)/n
length(which(lm1.leverage > cutoff))
```

```
## [1] 34
```

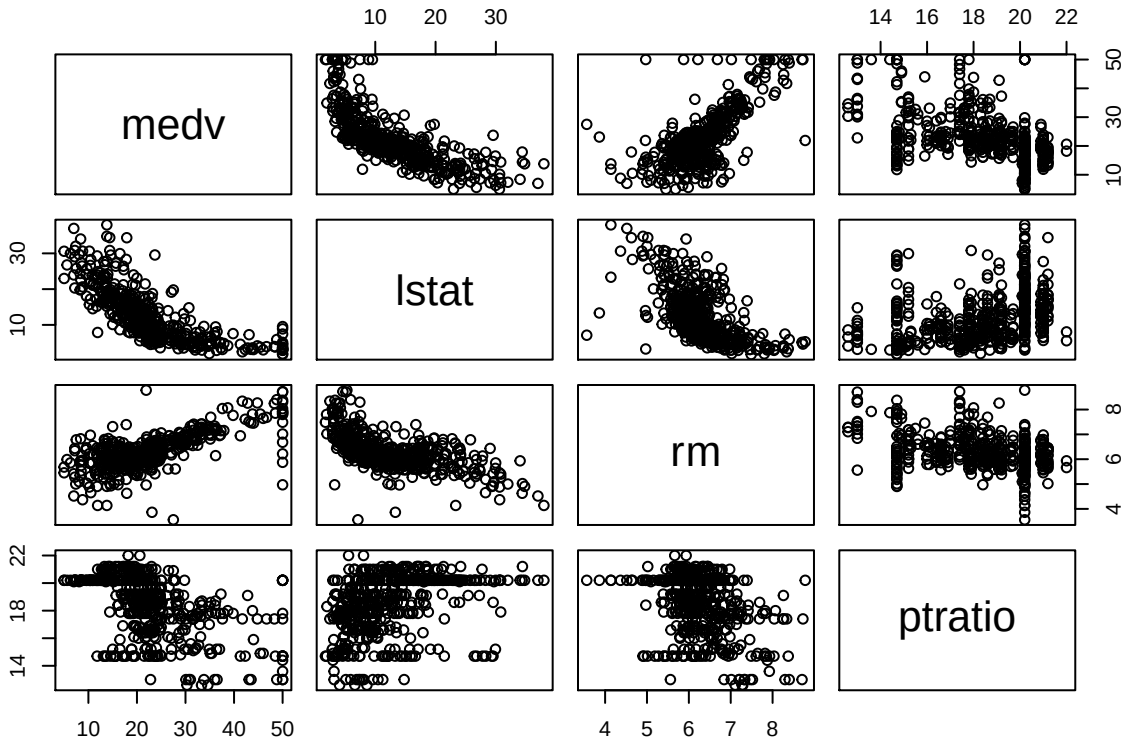
The results confirm that there are several (34) high leverage points.

Multiple Linear Regression

Let's now extend our model by including some other 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) variable into our model.

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

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



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

To be able to properly test our model (not use fictitious data points as we did in the case of simple linear regression), 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

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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.00   16.95   21.20   22.74   25.00   50.00
```

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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.00   17.05   21.00   21.68   24.65   50.00
```

Now, build a model using the training data set.

```
# 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
## -14.8219  -3.0757  -0.8036   1.7893  29.7479
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 18.11824    4.33535   4.179 3.59e-05 ***
## lstat       -0.56496    0.04778 -11.824  < 2e-16 ***
## rm           4.62379    0.45996  10.053  < 2e-16 ***
## ptratio     -0.94082    0.13192  -7.132 4.63e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.181 on 403 degrees of freedom
## Multiple R-squared:  0.6935, Adjusted R-squared:  0.6912
## F-statistic: 303.9 on 3 and 403 DF,  p-value: < 2.2e-16
```

From the summary, we can see that:

- R-squared has increased considerably, from 0.544 to 0.694 even though we have built it with a smaller dataset (407 observations, instead of 506 observations),
- 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.

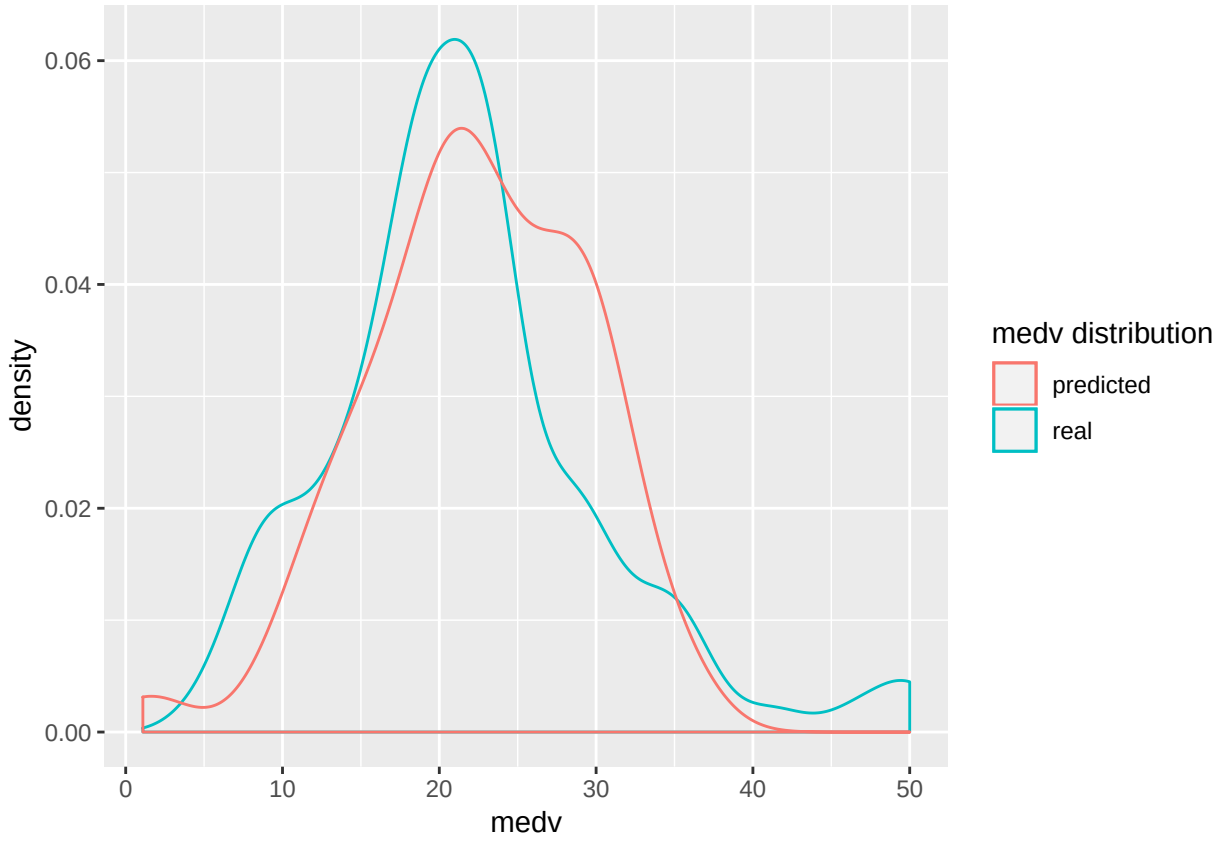
Let's make predictions using this model on the test data set that we have created.

```
# 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        5       11       12       14       15
## 32.31678 30.55989 21.75026 24.10511 21.20138 20.75116
```

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() +
  geom_density(data = test.boston.lm2, mapping = aes(x=medv, color = 'real')) +
  geom_density(data = test.boston.lm2, mapping = aes(x=pred, color = 'predicted')) +
  scale_colour_discrete(name ="medv distribution")
```



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] 0.6076704
```

R^2 on the test is lower than the one obtained on the training set, which is expected.

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

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

```
## [1] 5.432056
```

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

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

```
## [1] 21.68384
```

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

```
## [1] 0.2505117
```

So, it's not a small error, it's about 25% of the mean value.

Let's now build another model using all available predictors:

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

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

```
##
## Call:
## lm(formula = medv ~ ., data = train.boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.1772  -2.6987  -0.5194   1.7225  26.0486
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.759e+01  5.609e+00   6.702 7.17e-11 ***
## crim        -9.610e-02  4.024e-02  -2.388  0.01741 *
## zn           4.993e-02  1.521e-02   3.283  0.00112 **
## indus       -5.789e-03  6.745e-02  -0.086  0.93166
## chas         2.292e+00  1.019e+00   2.250  0.02501 *
## nox         -1.723e+01  4.244e+00  -4.059 5.95e-05 ***
## rm           3.784e+00  4.537e-01   8.341 1.26e-15 ***
## age          8.387e-04  1.450e-02   0.058  0.95391
## dis         -1.620e+00  2.217e-01  -7.310 1.50e-12 ***
## rad          3.031e-01  7.434e-02   4.078 5.51e-05 ***
## tax         -1.316e-02  4.144e-03  -3.176  0.00161 **
## ptratio     -9.582e-01  1.473e-01  -6.505 2.37e-10 ***
## black        9.723e-03  2.993e-03   3.249  0.00126 **
## lstat       -5.297e-01  5.691e-02  -9.308  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.692 on 393 degrees of freedom
## Multiple R-squared:  0.7549, Adjusted R-squared:  0.7468
## F-statistic:  93.1 on 13 and 393 DF,  p-value: < 2.2e-16
```

Note that even though we now have 13 predictors, we haven't much improved the R^2 value: in the model with 3 predictors (lm2), it was 0.693 and now it is 0.755. 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 a number of other significant variables.

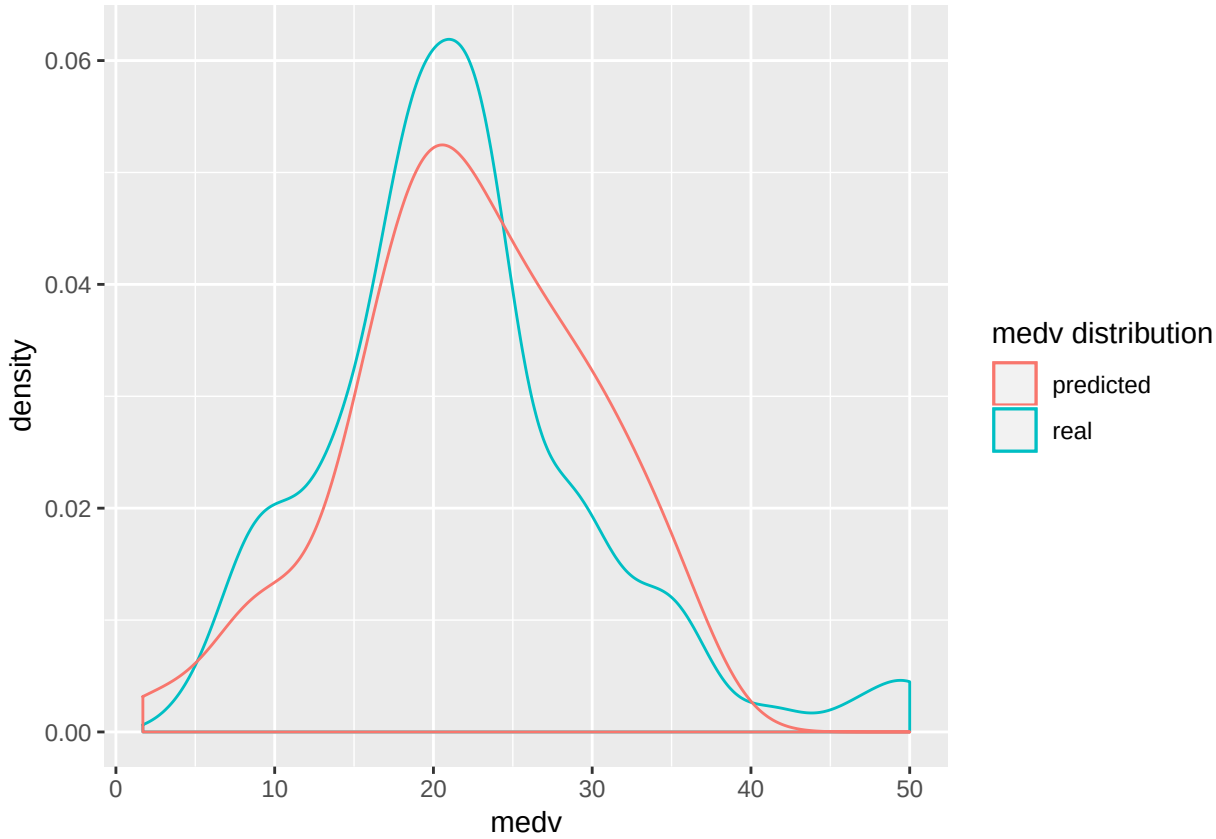
Let's do the prediction using the new model:

```
# calculate the predictions with the lm3 model over the test data
lm3.predict <- predict(lm3, newdata = test.boston)

# print out a few predictions
head(lm3.predict)
```

```
##        3        5       11       12       14       15
## 30.70615 28.05079 18.88585 21.53429 19.68122 19.43022
```

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

```
# combine the test set with the predictions
test.boston.lm3 <- cbind(test.boston, pred = lm3.predict)

# plot actual (medv) vs. predicted values
ggplot() +
  geom_density(data = test.boston.lm3, mapping = aes(x=medv, color = 'real')) +
  geom_density(data = test.boston.lm3, mapping = aes(x=pred, color = 'predicted')) +
  scale_colour_discrete(name ="medv distribution")
```



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

```
# calculate RSS
lm3.test.RSS <- sum((lm3.predict - test.boston$medv)^2)

# calculate R-squared on the test data
lm3.test.R2 <- 1 - lm3.test.RSS/lm.test.TSS
lm3.test.R2
```

```
## [1] 0.6685588
```

Again, we got lower R^2 than on the train set.

We can also compute RMSE:

```
# calculate RMSE
lm3.test.RMSE <- sqrt(lm3.test.RSS/nrow(test.boston))
lm3.test.RMSE
```

```
## [1] 4.992775
```

It is lower (therefore, better) than with the previous model.

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

Considering the number of variables in the model, we should check for *multicolinearity* (Assumption 4). To do that, we'll compute the **variance inflation factor (VIF)**:

```
# load the 'car' package
library(car)
```

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

```
##     crim       zn    indus     chas      nox       rm      age      dis
## 1.865531 2.364859 3.901322 1.064429 4.471619 2.010665 3.018555 3.961686
##      rad      tax  ptratio    black    lstat
## 7.799919 9.163102 1.907071 1.311933 2.967784
```

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

```
# calculate square root of the VIF
sqrt(vif(lm3))
```

```
##     crim       zn    indus     chas      nox       rm      age      dis
## 1.365844 1.537810 1.975177 1.031712 2.114620 1.417979 1.737399 1.990398
##      rad      tax  ptratio    black    lstat
## 2.792833 3.027062 1.380967 1.145396 1.722726
```

So, *tax* and *rad* variables exhibit multicollinearity. If we go back to the correlation plot, we'll see that they are, indeed, highly correlated (0.91).

There are also a few other suspicious variables: *indus*, *nox*, and *dis*.

TASK: Create a new model (lm4) by excluding either *tax* or *rad* variable. Compare the new model with lm3.