Multivariate Statistics - Exercise 4

Philipp Satlawa - h0640348

27/05/2021

This document contains the answered questions of exercise 4 of the course "Multivariate Statistics".

Principal Component Analysis (PCA)

1. install and load the packages - load and summerize the data

```r
# import necessary libraries
library(ISLR)
library(cvTools)
```

```
## Loading required package: lattice

## Loading required package: robustbase
```

```r
library(pls)
```

```
##
## Attaching package: 'pls'

## The following object is masked from 'package:stats':
##
##     loadings
```

```r
# load data
data(OJ)

# explore data
help(OJ)
head(OJ)
```

```
##   Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH
## 1       CH            237       1    1.75    1.99   0.00    0.0         0
## 2       CH            239       1    1.75    1.99   0.00    0.3         0
## 3       CH            245       1    1.86    2.09   0.17    0.0         0
```

```
## 4       MM           227       1    1.69    1.69   0.00    0.0         0
## 5       CH           228       7    1.69    1.69   0.00    0.0         0
## 6       CH           230       7    1.69    1.99   0.00    0.0         0
##   SpecialMM  LoyalCH SalePriceMM SalePriceCH PriceDiff Store7 PctDiscMM
## 1         0 0.500000        1.99        1.75      0.24     No  0.000000
## 2         1 0.600000        1.69        1.75     -0.06     No  0.150754
## 3         0 0.680000        2.09        1.69      0.40     No  0.000000
## 4         0 0.400000        1.69        1.69      0.00     No  0.000000
## 5         0 0.956535        1.69        1.69      0.00    Yes  0.000000
## 6         1 0.965228        1.99        1.69      0.30    Yes  0.000000
##   PctDiscCH ListPriceDiff STORE
## 1  0.000000          0.24     1
## 2  0.000000          0.24     1
## 3  0.091398          0.23     1
## 4  0.000000          0.00     1
## 5  0.000000          0.00     0
## 6  0.000000          0.30     0
```

```r
str(OJ)
```

```
## 'data.frame':    1070 obs. of  18 variables:
##  $ Purchase      : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 1 1 1 1 1 ...
##  $ WeekofPurchase: num  237 239 245 227 228 230 232 234 235 238 ...
##  $ StoreID       : num  1 1 1 1 7 7 7 7 7 7 ...
##  $ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.99 1.99 1.99 1.99 1.99 ...
##  $ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
##  $ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
##  $ SpecialCH     : num  0 0 0 0 0 0 1 1 0 0 ...
##  $ SpecialMM     : num  0 1 0 0 0 1 1 0 0 0 ...
##  $ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
##  $ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...
##  $ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
##  $ Store7        : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 2 2 ...
##  $ PctDiscMM     : num  0 0.151 0 0 0 ...
##  $ PctDiscCH     : num  0 0 0.0914 0 0 ...
##  $ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
##  $ STORE         : num  1 1 1 1 0 0 0 0 0 0 ...
```

```r
summary(OJ)
```

```
##  Purchase WeekofPurchase     StoreID        PriceCH         PriceMM     
##  CH:653   Min.   :227.0   Min.   :1.00   Min.   :1.690   Min.   :1.690  
##  MM:417   1st Qu.:240.0   1st Qu.:2.00   1st Qu.:1.790   1st Qu.:1.990  
##           Median :257.0   Median :3.00   Median :1.860   Median :2.090  
##           Mean   :254.4   Mean   :3.96   Mean   :1.867   Mean   :2.085  
##           3rd Qu.:268.0   3rd Qu.:7.00   3rd Qu.:1.990   3rd Qu.:2.180  
##           Max.   :278.0   Max.   :7.00   Max.   :2.090   Max.   :2.290  
##      DiscCH            DiscMM          SpecialCH        SpecialMM     
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000  
##  Median :0.00000   Median :0.0000   Median :0.0000   Median :0.0000  
```

```
##  Mean   :0.05186   Mean   :0.1234   Mean   :0.1477   Mean   :0.1617
##  3rd Qu.:0.00000   3rd Qu.:0.2300   3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :0.50000   Max.   :0.8000   Max.   :1.0000   Max.   :1.0000
##     LoyalCH          SalePriceMM     SalePriceCH      PriceDiff       Store7
##  Min.   :0.000011   Min.   :1.190   Min.   :1.390   Min.   :-0.6700   No :714
##  1st Qu.:0.325257   1st Qu.:1.690   1st Qu.:1.750   1st Qu.: 0.0000   Yes:356
##  Median :0.600000   Median :2.090   Median :1.860   Median : 0.2300
##  Mean   :0.565782   Mean   :1.962   Mean   :1.816   Mean   : 0.1465
##  3rd Qu.:0.850873   3rd Qu.:2.130   3rd Qu.:1.890   3rd Qu.: 0.3200
##  Max.   :0.999947   Max.   :2.290   Max.   :2.090   Max.   : 0.6400
##    PctDiscMM        PctDiscCH       ListPriceDiff       STORE
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.000   Min.   :0.000
##  1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.140   1st Qu.:0.000
##  Median :0.0000   Median :0.00000   Median :0.240   Median :2.000
##  Mean   :0.0593   Mean   :0.02731   Mean   :0.218   Mean   :1.631
##  3rd Qu.:0.1127   3rd Qu.:0.00000   3rd Qu.:0.300   3rd Qu.:3.000
##  Max.   :0.4020   Max.   :0.25269   Max.   :0.440   Max.   :4.000
```

```r
# convert categorical variables to factors
OJ$StoreID<-as.factor(OJ$StoreID)
OJ$STORE<-as.factor(OJ$STORE)
OJ$SpecialCH<-as.factor(OJ$SpecialCH)
OJ$SpecialMM<-as.factor(OJ$SpecialMM)
```

The dataset `OC` contains 1070 records and 18 attributes. The records represent purchases of orange juice for two brands (`CH` and `MM`). All except for two variables (`Purchase` and `Store7`) are *numerical*, however the variables `StoreID`, `STORE`, `SpecialCH` and `SpecialMM` are encoded as numerical but are in reality categorical. Hence, we change the representation of these variables from *numerical* to *factor*. Furthermore, some attribures seem to contain redundant information, such as `StoreID` and `STORE`.

2. split data into train and test set

```r
# set seed
set.seed(3333)

# create indices and shuffle randomly
idx = sample(dim(OJ)[1])

# set train split
train_split = 0.75

# split dataset in train and test
train = OJ[idx[1:(dim(OJ)[1]*train_split)],]
test = OJ[idx[((dim(OJ)[1]*train_split)+1):dim(OJ)[1]],]

# print class proportions of train and test
print(oj_prop <- table(OJ$Purchase))
```

```
##
##  CH  MM
## 653 417
```

```r
print(train_prop <- table(train$Purchase))
```

```
##
##  CH  MM
## 490 312
```

```r
print(test_prop <- table(test$Purchase))
```

```
##
##  CH  MM
## 162 105
```

```r
# calculate ratio CH/MM
print(oj_prop["CH"]/oj_prop["MM"])
```

```
##       CH
## 1.565947
```

```r
print(train_prop["CH"]/train_prop["MM"])
```

```
##       CH
## 1.570513
```

```r
print(test_prop["CH"]/test_prop["MM"])
```

```
##       CH
## 1.542857
```

The class proportions of the train-set and test-set are balanced (although not the same, the CH/MM ratio is 1.57 in train versus 1.54 in test and 1.57 in the entire data set). An imbalance in the class-proportions of the two subsets can result in a model that performs worse than trained on a balanced class-proportions dataset. The imbalance of the classes in the subsets is directly linked to the prior probability, if the classifier takes the prior probability into account and the subsets are imbalanced the trained model is biased towards one class and the prediction is worse. In the worst case, we can have a split where one class (e.g. CH) is just in the training set and the other class (e.g. MM) just in the test set. This would result in a model that cannot predict class number two (e.g. MM) since there was no data in the training set.

3. LDA

```r
# load package
require(MASS)
```

```
## Loading required package: MASS
```

```r
#LDA
lda_model_cv <- lda(Purchase ~ ., data = train, CV = TRUE)
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```r
# confusion matrix of the cross-validated LDA model
(table(prediction = lda_model_cv$class, truth = train[, "Purchase"]))
```

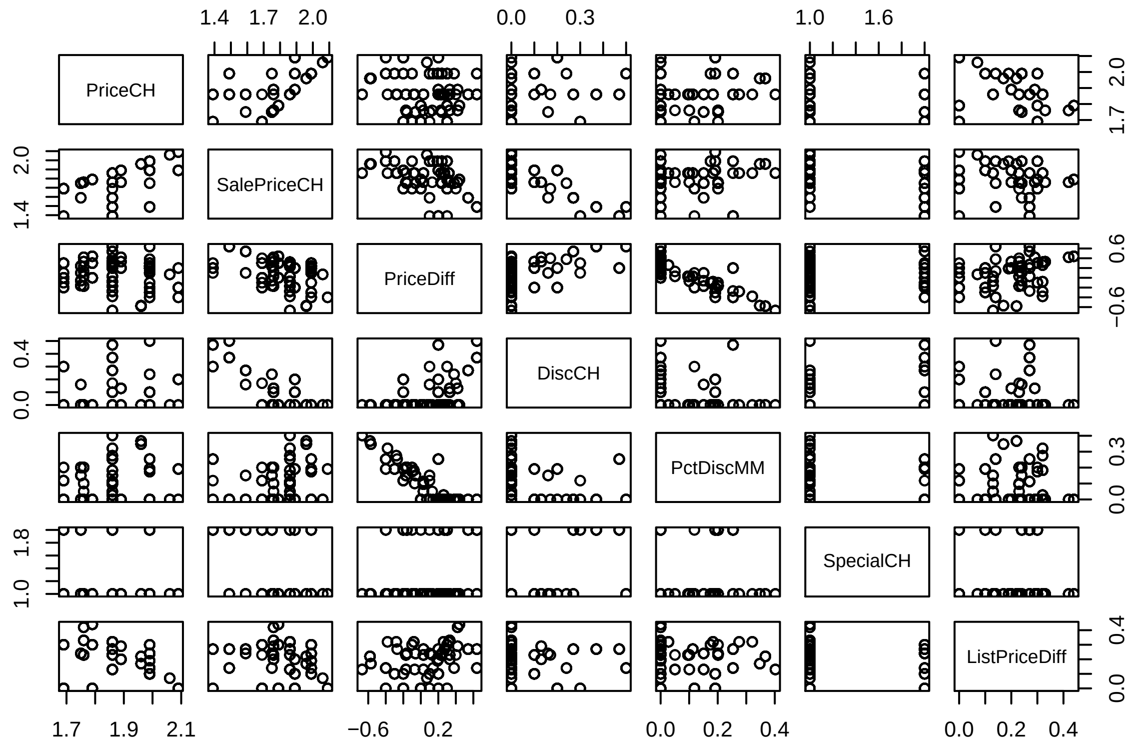
```
##           truth
## prediction  CH  MM
##         CH 425  73
##         MM  65 239
```

While training the model we encounter collinearity of some variables. This means at least two variables are strongly correlated hence contain the same information and are therefore redundant. To obtain a better model it is advisable to remove these variables.

4. select variables for predicting

```r
# plot time/location predictors
pairs(OJ[,c("StoreID", "STORE", "Store7")])
```


```r
# plot price predictors
pairs(OJ[,c("PriceCH", "SalePriceCH", "PriceDiff", "DiscCH", "PctDiscMM",
            "SpecialCH", "ListPriceDiff")])
```




```
# clean data sets
OJ_cl <- OJ[,c("Purchase", "WeekofPurchase", "StoreID", "PriceCH", "PriceMM",
               "DiscCH", "DiscMM", "LoyalCH")]
train_cl = train[,c("Purchase", "WeekofPurchase", "StoreID", "PriceCH",
                    "PriceMM", "DiscCH", "DiscMM", "LoyalCH")]
test_cl = test[,c("Purchase", "WeekofPurchase", "StoreID", "PriceCH",
                  "PriceMM", "DiscCH", "DiscMM", "LoyalCH")]
```

Time/Location Predictors

- In the subgroup *time/location* the variables `StoreID` and `STORE` are basically the same the only difference is the representation (coding).
- The variable `Store7` is contained in the variables `StoreID` and `STORE`, hence it is redundant.

Price Predictors

There are several attributes with relationships the subgroup describing the *price* that are containing the same information.

- `SalePriceXX` = `PriceXX` - `DiscXX`
- `DiscXX` = `PriceXX` * `PctDiscXX`
- `PriceDiff` = `SalePriceMM` - `SalePriceCH`
- `ListPriceDiff` = `PriceMM` - `PriceCH`

Since the variables can be explained through the above described relationships the two variables `PriceXX` and `DiscXX` contain all the necessary information.

To conclude, many of the variables of the original dataset contain the same information, hence the following attributes are sufficient to train a model without loosing any information `WeekofPurchase`, `StoreID`, `PriceCH`, `PriceMM`, `DiscCH`, `DiscMM`, `LoyalCH`.

5. LDA on train set

```r
#LDA
lda_model_cv <- lda(Purchase ~ ., data = train_cl, CV = TRUE)

# confusion matrix of the cross-validated LDA model
(cm_train <- table(prediction = lda_model_cv$class, truth = train_cl[, "Purchase"]))
```

```
##            truth
## prediction  CH  MM
##         CH 422  75
##         MM  68 237
```

```r
# total observations, correct and misclassified
n <- dim(train_cl)[1]
correct <- sum(diag(cm_train))
mis <- n - correct

# apparent error rate
(APER <- mis/n)
```

```
## [1] 0.1783042
```

By applying LDA on the cleaned train set we obtain an *apparent error rate* of 0.178 for the classifier.

5. LDA on test set

```r
#LDA
lda_model <- lda(Purchase ~ ., data = train_cl, CV = FALSE)

# predict test set cases
preds_test <- predict(lda_model, newdata = test_cl)

# confusion matrix of the final LDA model
(cm_test <- table(predictions = preds_test$class, truth = test_cl[, "Purchase"]))
```

```
##             truth
## predictions  CH  MM
##          CH 139  24
##          MM  23  81
```

```r
# total observations, correct and misclassified
n <- dim(test_cl)[1]
correct <- sum(diag(cm_test))
mis <- n - correct

# apparent error rate
(APER <- mis/n)
```

```
## [1] 0.17603
```

After retraining the model on the cleaned train set we apply the cleaned test set on the final model and receive an *apparent error rate* of 0.176. This result is matching with the outcome of the previously trained model `lda_model_cv`.

Regression

1. load data

```r
# load data
data(Boston)

# explore data
help(Boston)
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

```r
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 ...
```

```r
summary(Boston)
```

```
##       crim                zn             indus            chas
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000
##  1st Qu.: 0.08204   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000
##  Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.00000
##  Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.06917
##  3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000
##       nox               rm             age              dis
##  Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130
##  1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100
##  Median :0.5380   Median :6.208   Median : 77.50   Median : 3.207
##  Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795
##  3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.188
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127
##       rad              tax           ptratio          black
##  Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32
##  1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38
##  Median : 5.000   Median :330.0   Median :19.05   Median :391.44
##  Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90
##      lstat            medv
##  Min.   : 1.73   Min.   : 5.00
##  1st Qu.: 6.95   1st Qu.:17.02
##  Median :11.36   Median :21.20
##  Mean   :12.65   Mean   :22.53
##  3rd Qu.:16.95   3rd Qu.:25.00
##  Max.   :37.97   Max.   :50.00
```

```r
# missing values
which(is.na(Boston))
```

```
## integer(0)
```

```r
# convert categorical variables to factors
Boston$chas <- as.factor(Boston$chas)
Boston$rad <- as.factor(Boston$rad)
```

The dataset `Boston` contains 506 records and 14 attributes that represent the housing values in suburbs of Boston. The majority of variables are represented as continuous numbers, just the variables `chas` and `rad` are represented as discrete numbers (integers) because they are actually categorical variables. The attribute `rad` is additionally an ordinal variable.

2. split data into train and test set

```
# set seed
set.seed(3333)

# create indices and shuffle randomly
idx = sample(dim(Boston)[1])

# set train split
train_split = 2/3

# split dataset in train and test
train = Boston[idx[1:(dim(Boston)[1]*train_split)],]
test = Boston[idx[((dim(Boston)[1]*train_split)+1):dim(Boston)[1]],]

# print dimensions of train and test
dim(train)
```

```
## [1] 337  14
```

```
dim(test)
```

```
## [1] 168  14
```

```
# print class proportions of train set
print((mean(train$medv)))
```

```
## [1] 22.39496
```

```
print((sd(train$medv)))
```

```
## [1] 8.848013
```

```
# print class proportions of test set
print((mean(test$medv)))
```

```
## [1] 22.82321
```

```
print((sd(test$medv)))
```

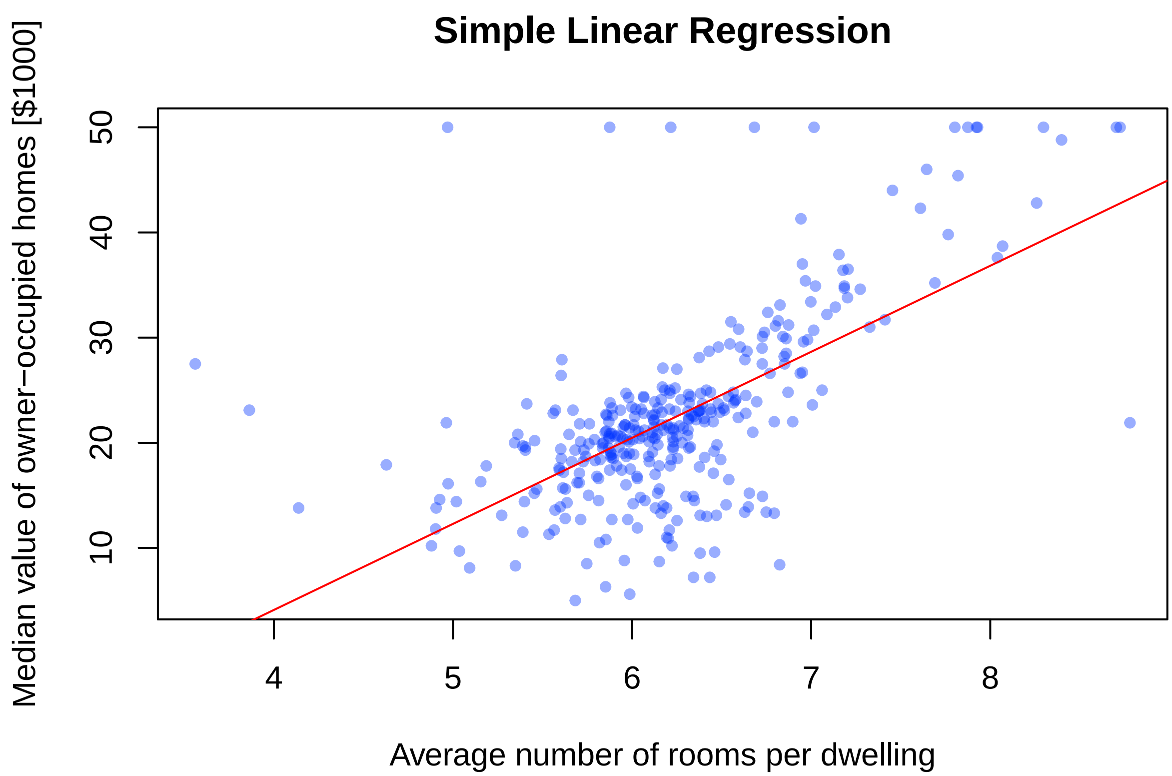
```
## [1] 9.905463
```

By comparing the mean of the target variable `medv` in both the train and test set we get similar numbers 22.4 and 22.8 respectively, which indicates a good split between the train and test set (at least of the target variable). The higher standard deviation in the test set also makes sense due to the smaller number of records contained in the test set the standard deviation is expected to be higher compared to the train set. The model we want to create for predicting the house prices shall be as simple as possible, hence flexible for adapting to unknown data, however as complex as necessary due to the fact that we do not have the future data and the best guess is looking into the data of the past. Too complex models might fit the training data perfectly however they often perform poorly on unseen data (overfitting) therefore we hold out a part of our data (test set) to be able to validate the model.

3. simple regression model

```r
# linear model
lm_simp <- lm(medv ~ rm, data = train)

# scatterplot with regression line
plot(train[c("rm", "medv")], col = rgb(0,0.2,1,0.4), pch = 20,
      main = "Simple Linear Regression",
      xlab = "Average number of rooms per dwelling",
      ylab = "Median value of owner-occupied homes [$1000]")
abline(lm_simp, lwd = 1, col = "red")
```



After calculating the simple regression model, we can see that there is a correlation between the two attributes `medv` and `rm`. Nonetheless, there are many data points that cannot be explained by just the linear relationship between `medv` and `rm`. In particular there are some outliers, that are high value homes with a small average number of rooms per dwelling and cannot be explained by this simple regression model.

4. RMSE

```
# predict train set
preds_simp <- predict(lm_simp)
# calculate RMSE on train set
sqrt(mean((preds_simp - train[, "medv"])^2))
```

```
## [1] 6.825318
```

```
# use cross validation on train set
cvFit(lm_simp, data = train, y = train$medv, K = 10, seed = 3333)
```

```
## 10-fold CV results:
##       CV
## 6.874905
```

The results show a slightly higher RMSE 6.87 calculated on a 10-fold cross validated train set compared to the RMSE of 6.83 obtained on the train set without cross validation.

5. Partial Least Squares (PLS)

```
# make this example reproducible
set.seed(3333)
# PLS on train set with 10-fold CV
pls_model <- plsr(medv ~ ., data = train, ncomp = 10, validation = "CV")
# print summary
summary(pls_model)
```

```
## Data:    X dimension: 337 20
##  Y dimension: 337 1
## Fit method: kernelpls
## Number of components considered: 10
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           8.861    8.110    7.992    7.742    6.951    6.120    6.011
## adjCV        8.861    8.109    7.992    7.739    6.950    6.113    6.010
##        7 comps  8 comps  9 comps  10 comps
## CV       5.750    5.669    5.513     5.371
## adjCV    5.744    5.660    5.502     5.358
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       79.95    95.37    98.94    99.47    99.85    99.93    99.96    99.99
```

```
## medv     16.35    19.17    24.86    40.57    53.93    56.40    60.28    61.60
##        9 comps  10 comps
## X      100.00    100.00
## medv    64.42     66.68
```

From the results of the PLS without scaling the data, we can deduct that the accuracy of the model increases with the number of components used. Considering the steady increases we would use at lest 7 components that can explain 60.3 % of `medv`s variance. Certainly, it is also possible to to use 10 components to reduce the RMSE by ruffly 0.6 and get a model that can explain 66.7% of `medv`s variance.

6. prediction error on the test data

```r
# Performance on test set for 7-component PLS model
preds_pls_test <- predict(pls_model, newdata = test, ncomp = 7)

# calculate test RMSE
sqrt(mean((preds_pls_test - test[, "medv"])^2))
```

```
## [1] 5.245013
```

The PLS model with 7 components achieves a RMSE of 5.25 on the test data.

7. scale the data and apply Partial Least Squares (PLS)

```r
# make this example reproducible
set.seed(3333)
# PLS on train set with 10-fold CV
pls_model_scaled <- plsr(medv ~ ., data = train, ncomp = 10, validation = "CV",
                         scale = TRUE)
summary(pls_model_scaled)
```

```
## Data:    X dimension: 337 20
##  Y dimension: 337 1
## Fit method: kernelpls
## Number of components considered: 10
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           8.861    6.658    5.581    5.388    5.326    5.289    5.286
## adjCV        8.861    6.657    5.569    5.373    5.308    5.268    5.263
##        7 comps  8 comps  9 comps  10 comps
## CV       5.278    5.262    5.267     5.244
## adjCV    5.255    5.240    5.243     5.222
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       29.08    38.71    44.99    49.92    54.09    59.32    62.18    65.75
## medv    44.51    63.06    66.69    68.27    69.31    69.60    69.79    69.87
```

```
##       9 comps  10 comps
## X       68.04     71.48
## medv    69.98     70.03
```

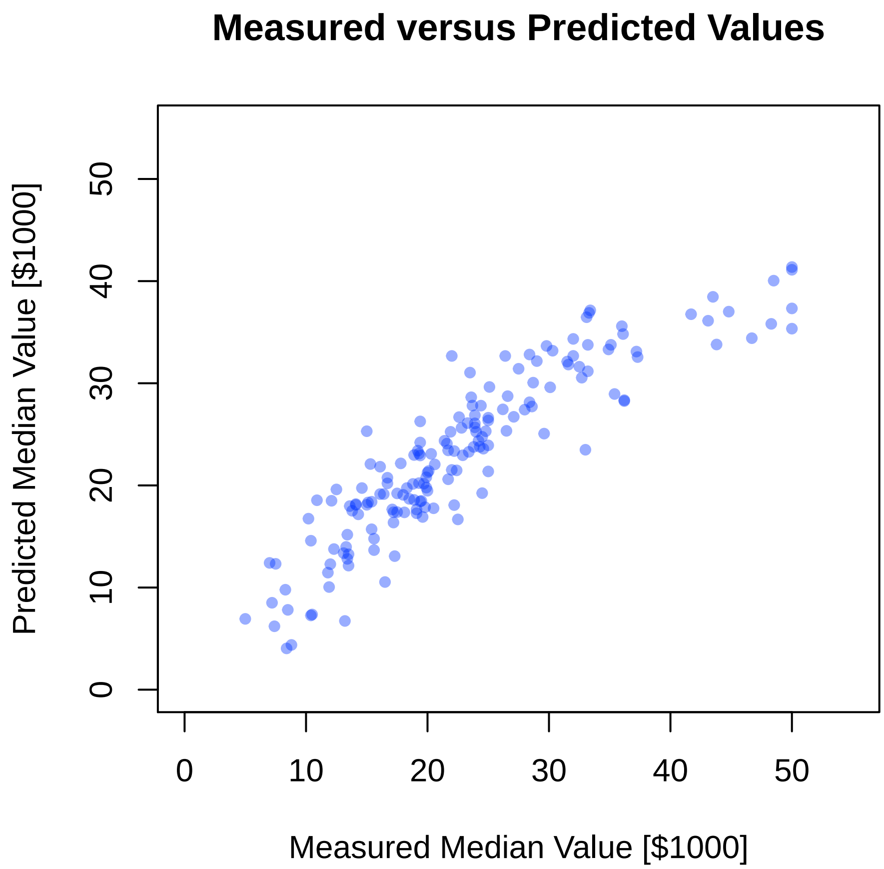
```r
# Performance on test set for scaled 4-component PLS model
preds_pls_scaled_test <- predict(pls_model_scaled, newdata = test, ncomp = 5)
# calculate test RMSE
sqrt(mean((preds_pls_scaled_test - test[, "medv"])^2))
```

```
## [1] 4.373463
```

Training the model on scaled data increased the models performance and reduced the number of essential components to 4. With 4 components we are able to explain 68.3 % of `medv`s variance and obtain a RMSE 5.31 on the train set. If we now compare the non-scaled PLS-model with the scaled PLS-model, we can can clearly see the superiority of the scaled PLS-model, where we can obtain with 4 components a RMSE of 4.37 on the test set, compared to a RMSE of 5.25 the test set of the non-scaled PLS-model.

8. Visualize performance

```r
# Plot results
plot(test[, "medv"], preds_pls_scaled_test, col = rgb(0,0.2,1,0.4), pch = 20,
     xlim = c(0,55), ylim = c(0,55),
     main = "Measured versus Predicted Values",
     xlab = "Measured Median Value [$1000]", ylab = "Predicted Median Value [$1000]")
```




```r
#abline(lm_simp, lwd = 1, col = "red")
```

In the plot we can see the model performing quite well until the median home value of $30000. Homes with a measured median home value above $30000 are being undervalued by the model. Thereby undervaluing the homes with the highest measured median home value (above $40000) the most. However, for most cases the model predicts good estimates and can be deployed with the constraint that more valuable homes should be measured.