Final Project - Big Data Methods for Economists

Yannik Haller (12-918-645) Sophia Bieri (15-727-373)

1 5 2020

Information

These are our (Sophia Bieri & Yannik Haller) R codes used in terms of the final project in the seminar **Big Data Methods for Economists**. Explanations and discussions (i.e. our report) are also directly included within this document. Some resulting values from R-calculations are directly inserted into our answer-texts. Comment-symbols from R-Outputs (#) are removed (i.e. summaries are without hashtags for better overview).

Preparation

In a first step we need to set the appropriate working directory (this must be done for each machine individually), load all required packages and load the data.

```
# Set working directory
setwd("~/Yannik/UZH/20FS/Big Data Methods for Economists (S)/Final Project")

# Load required packages
pkg <- c("stargazer","glmnet","randomizr","corrplot","caret","readxl",
         "tidyverse","lubridate","zoo","gsubfn", "mice", "matrixStats",
         "purrr","tidyr","ggplot2","dplyr","magrittr","pre","corrplot","sparsebn",
         "sparsebnUtils","Metrics","OneR","caTools","rpart","rpart.plot","devtools",
         "splitstackshape","keras","mlbench","dplyr","magrittr","neuralnet",
         "tensorflow","Sequential","data.table")
invisible(lapply(pkg, library, character.only = TRUE))
rm(list=ls())

# Load the data
houses <- read.csv("training.csv")
```

Note that we assume that the required packages are already installed on the machine. If this is not the case, one can simply run **lapply(pkg, install.packages, character.only = FALSE)** right after defining the *pkg* variable.

Data Cleaning

Before we start cleaning the data, we first look at some initial summary statistics of our data and check for completeness, since we want to prevent system failures due to missing values.

```
# Get summary statistics
stargazer(houses, type = "text", median = T)
```

```
=================================================================================================
Statistic              N        Mean      St. Dev.      Min     Pctl(25)   Median   Pctl(75)     Max
-------------------------------------------------------------------------------------------------
id                  35,000 17,500.500 10,103.770      1      8,750.8  17,500.5  26,250.2  35,000
area                32,375   116.458     47.928      1.000    85.000    111.000   140.000   882.000
area_useable         5,647   130.263     95.234      1.000    90.000    120.000   155.000 4,667.000
balcony             17,583    1.000      0.000      1.000     1.000     1.000     1.000     1.000
basement             1,482    1.000      0.000      1.000     1.000     1.000     1.000     1.000
bath_tube              587    1.000      0.000      1.000     1.000     1.000     1.000     1.000
building_plot           10    1.000      0.000      1.000     1.000     1.000     1.000     1.000
cabletv              7,880    1.000      0.000      1.000     1.000     1.000     1.000     1.000
ceiling                 34    1.000      0.000      1.000     1.000     1.000     1.000     1.000
cheminee             4,301    1.000      0.000      1.000     1.000     1.000     1.000     1.000
elevator            16,757    1.000      0.000      1.000     1.000     1.000     1.000     1.000
first_time             670    1.000      0.000      1.000     1.000     1.000     1.000     1.000
floors              16,970    2.191      1.647     -1.000     1.000     2.000     3.000    29.000
furnished               55    1.000      0.000      1.000     1.000     1.000     1.000     1.000
kids_friendly        6,870    1.000      0.000      1.000     1.000     1.000     1.000     1.000
laundry              1,067    1.000      0.000      1.000     1.000     1.000     1.000     1.000
minergie               675    1.000      0.000      1.000     1.000     1.000     1.000     1.000
new_building         2,208    1.000      0.000      1.000     1.000     1.000     1.000     1.000
newly_built         35,000    0.253      0.435        0         0         0         1         1
oldbuilding            179    1.000      0.000      1.000     1.000     1.000     1.000     1.000
oven                    58    1.000      0.000      1.000     1.000     1.000     1.000     1.000
parking_indoor      10,640    1.000      0.000      1.000     1.000     1.000     1.000     1.000
parking_outside      8,189    1.000      0.000      1.000     1.000     1.000     1.000     1.000
playground             247    1.000      0.000      1.000     1.000     1.000     1.000     1.000
pool                   185    1.000      0.000      1.000     1.000     1.000     1.000     1.000
quiet                   60    1.000      0.000      1.000     1.000     1.000     1.000     1.000
raised_groundfloor     121    1.000      0.000      1.000     1.000     1.000     1.000     1.000
rooms               33,878    4.111      1.214      1.000     3.500     4.500     4.500    15.000
sale                35,000    1.000      0.000        1         1         1         1         1
sunny                   37    1.000      0.000      1.000     1.000     1.000     1.000     1.000
terrace                453    1.000      0.000      1.000     1.000     1.000     1.000     1.000
topstorage             215    1.000      0.000      1.000     1.000     1.000     1.000     1.000
veranda                 29    1.000      0.000      1.000     1.000     1.000     1.000     1.000
wheelchair           3,377    1.000      0.000      1.000     1.000     1.000     1.000     1.000
year                35,000 2,015.296     2.045      2,012     2,014     2,015     2,017     2,019
year_built          21,617 1,996.747    37.189   1,190.000 1,988.000 2,009.000 2,014.000 2,022.000
zipcode             35,000 4,472.052  2,746.693     1,000     1,822     3,963     6,918     9,658
-------------------------------------------------------------------------------------------------
```

```
# Get the number of missing values contained in the dataset
sum(is.na(houses))
```

```
[1] 925544
```

```
str(houses)
```

```
'data.frame':   35000 obs. of  44 variables:
 $ id                : int  1 2 3 4 5 6 7 8 9 10 ...
 $ address           : Factor w/ 15962 levels "--, 1008 Prilly",..: 10573 14621 66 832 148 11845 730
 $ area              : int  60 137 134 245 111 100 84 38 172 133 ...
 $ area_useable      : int  NA NA 142 245 NA NA NA NA NA NA ...
 $ balcony           : int  1 NA NA NA NA 1 1 1 NA NA ...
 $ basement          : int  NA NA NA NA NA NA NA NA NA NA ...
 $ bath_tube         : int  NA NA NA NA NA NA NA NA NA NA ...
 $ building_plot     : int  NA NA NA NA NA NA NA NA NA NA ...
 $ cabletv           : int  NA NA NA NA NA NA NA NA NA NA ...
 $ ceiling           : int  NA NA NA NA NA NA NA NA NA NA ...
 $ cheminee          : int  NA NA NA NA NA NA NA NA NA NA ...
 $ date              : Factor w/ 2605 levels "01.01.2013","01.01.2014",..: 2286 2139 2484 1328 2125
 $ date_available    : Factor w/ 83 levels "","01.01.2020",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ elevator          : int  1 NA NA NA NA 1 NA 1 NA NA ...
 $ first_time        : int  NA NA NA NA NA NA NA NA NA NA ...
 $ floors            : int  NA NA 2 NA NA 1 NA 1 2 NA ...
 $ furnished         : int  NA NA NA NA NA NA NA NA NA NA ...
 $ home_type         : Factor w/ 8 levels "Attika","Dachwohnung",..: 8 8 8 8 8 8 8 8 8 8 ...
 $ kids_friendly     : int  NA NA NA NA NA 1 NA NA NA NA ...
 $ laundry           : int  NA NA NA NA NA NA NA NA NA NA ...
 $ minergie          : int  NA NA NA NA NA NA NA NA NA NA ...
 $ municipality      : Factor w/ 5207 levels "  La Tour-de-TrÃªme",..: 3981 2710 334 1750 808 5111 3
 $ new_building      : int  NA NA NA NA NA NA NA NA NA NA ...
 $ newly_built       : int  1 0 0 0 0 0 0 0 0 0 ...
 $ oldbuilding       : int  NA NA NA NA NA NA NA NA NA NA ...
 $ oven              : int  NA NA NA NA NA NA NA NA NA NA ...
 $ parking_indoor    : int  1 NA 1 NA NA 1 NA NA 1 NA ...
 $ parking_outside   : int  NA NA NA NA NA NA NA NA NA NA ...
 $ playground        : int  NA NA NA NA NA NA NA NA NA NA ...
 $ pool              : int  NA NA NA NA NA NA NA NA NA NA ...
 $ price             : Factor w/ 2515 levels "1000.0CHF","10000.0CHF",..: 799 435 81 485 2152 2369 1
 $ quiet             : int  NA NA NA NA NA NA NA NA NA NA ...
 $ raised_groundfloor: int  NA NA NA NA NA NA NA NA NA NA ...
 $ rooms             : num  2.5 3.5 5.5 5.5 3.5 3.5 4.5 1.5 6.5 4.5 ...
 $ sale              : int  1 1 1 1 1 1 1 1 1 1 ...
 $ street            : Factor w/ 12222 levels "","-","--","- canevascini",..: 7280 10938 1 1 1 8499
 $ sunny             : int  NA NA NA NA NA NA NA NA NA NA ...
 $ terrace           : int  NA NA NA NA NA NA NA NA NA NA ...
 $ topstorage        : int  NA NA NA NA NA NA NA NA NA NA ...
 $ veranda           : int  NA NA NA NA NA NA NA NA NA NA ...
 $ wheelchair        : int  NA NA NA NA NA NA NA NA NA NA ...
 $ year              : int  2012 2012 2015 2018 2013 2017 2015 2012 2016 2014 ...
 $ year_built        : int  2012 NA 2004 NA NA 2014 1977 1977 NA NA ...
 $ zipcode           : int  1721 6645 1030 1965 1124 8542 7252 6612 1937 8560 ...
```

We observe that there are many missing values in our data (note that out of the numerical variables only *id*, *new_built*, *sale*, *year* and *zipcode* contain values for all of the 35000 observations). Therefore, we need to decide how to handle these missing values and assess which variables are potentially important to predict housing prices.

First, we set the *id* variable to the index of the dataset and drop it afterwards. Then, we notice that the variable *sale* contains a 1 for every observation. Thus, we decide to drop this variable due to two reasons: (1) it doesn't incorporate any variation and therefore has no explanatory power and (2) it would cause multicollinearity problems because it perfectly correlates with the constant. Furthermore, we observe that there are many binary variables which contain 1s if they are true, but missing values if it is unkown whether they are true or not. Hence, we decide to impute 0s for the missing values within those dummy variables, such that they are equal to 1 if they are known to be true and 0 otherwise.

```
# Set the id to the index
row.names(houses) <- houses$id
houses$id <- NULL

# Drop the sale variable
houses$sale <- NULL

# Impute the missing values to the dummy variables
for (i in c(1:length(houses))){
  if(is.integer(houses[,i])){
    if(sd(houses[,i], na.rm = T) == 0){
      houses[is.na(houses[,i]),i] <- as.integer(0)
    }
  }
}
```

Another feature we observe in the data is that the *floor* variable takes integer values between -1 and 29, but never contains a 0. We therefore assume that this variable tells us at which floor an appartment is located and that most of the missing values should actually be equal to 0 (which would mean that the apartment is located on the ground floor). Hence, we impute 0s for every missing value in this variable. Furthermore, since we expect a non-linear relationship between *floor* and *price*, we transform the *floor* variable into a categorical variable.

```
# Impute the missing values to the floor variable
houses$floors[is.na(houses$floors)] <- as.integer(0)

# Transform the floor variable
houses$floors <- as.factor(houses$floors)
```

In a next step, we care about the time indicating variables. Since the relationship between housing prices and time is probably non-linear and does rather depend on current market situations than on some linear time trend, we decide to consider the *date* as a categorical variable with one category for each month-year combination (i.e. we do not consider differences between different days in the same month in a given year). Furthermore, since the date variable already allows us to control for time, we decide to get rid of the *year* variable. In addition, we think that it is more reasonable to consider the age of a building, rather than the year in which it is built. Hence, we create a new variable called *age* (which is simply the difference between the *year* and the *year_built* variable) and also remove the *year_built* variable. We think that one could reasonably argue that there exists some trend between housing prices and the age of a building. Hence, we decide to treat the *age* variable as a numerical input. Moreover, we decide to impute a 0 for negative values of the *age* variable, since we think that it does not make a meaningful difference whether the house is newly built or is going to be finished in the near future.

```
# Reshape the date variable
houses$date <- as.Date(houses$date, "%d.%m.%Y")
houses$date <- as.yearmon(paste(year(houses$date), month(houses$date)), "%Y %m")
houses$date <- as.character(houses$date)
houses$date <- as.factor(houses$date)
```

```r
# Generate the age variable and remove the year and year_built variable
houses$age <- houses$year - houses$year_built
houses$age[houses$age < 0] <- 0
houses$year <- NULL
houses$year_built <- NULL
```

After we decided on how to treat the time variables, we turn to the variables which contain geographical information (i.e. *address*, *municipality*, *street* and *zipcode*). First, we argue that the *address* variable does not contain any useful information, since it is individual for every object. Thus, we decide to drop the *address* variable. Moreover, we decide to drop the *street* variable, since this variable does also contain too many different inputs to have meaningful predictive power on housing prices. However, we think that we should definitely consider different cities and villages. To do so, we have two different variables which contain the desired information: *municipality* and *zipcode*. The *municipality* variable just states the name of the city or village and the *zipcode* indicates the local authority. Since both of these variables contain potentially important information, we decide to keep both of them in the final dataset. However, since the *municipality* variable appears to contain quite inconsistently denoted entries, we decide to not consider this variable in a first step and use only the *zipcode* variable as a geographical indicator for the impuation of the missing values later on. An externally cleaned version of the *municipality* variable will then again be included after the imputation of missing values took place (we will describe how we did the cleaning in more detail within the section below the imputation part). Furthermore, we argue that it is more reasonable to consider *zipcode* fixed effects than assuming a linear relationship between the numerical values of *zipcode* and *prices*. Thus, we transform the *zipcode* variable into a categorical format. However, since both of these variables contain geographical information, one should keep in mind that the effects of a municipality and its corresponding zipcodes are probably quite similar. Due to this reason (and to safe computational capacity) we decide to apply some of our models later on with only one of these two variables.

```r
# Remove the unused geographical variables
houses$address <- NULL
houses$street <- NULL

# Transform the zipcode variable
houses$zipcode <- as.factor(houses$zipcode)
```

After we removed some carefully selcted variables and cleaned the data for all the dummies and some of the categorical variables, we now turn to a variable, which requires some deeper investigation and thoughts about how to consider it: *date_available*. First, we take a look at the distinct values this variable takes. We observe that this variable contains many different dates, two special categories (i.e. "nach Vereinbarung" & "sofort") and no information at all for some observations (which we decide to label as "keine Angabe"). By the same argument as for the *date* variable, we decide to only consider either date's month-year combination. Furthermore, since we notice that most of the dates are either located at the beginning or at the end of a month, we assign each date which is above the 15^{th} day of a month to the subsequent month. Moreover, we observe one value contained in the *date_available* variable which does not make much sense (i.e. 30.08.5292). We decide to attach this observation to the "keine Angabe" category. Finally, we consider each month-year combination and either of the three other classes as individual categories, which leads to a total of 36 different classes.

```r
# Take a look at distinct values of the date_available variable
unique(houses$date_available)
```

```
 [1]                 nach Vereinbarung 31.12.2020       15.12.2020
 [5] sofort          01.06.2021        01.08.2021       31.08.2020
 [9] 01.09.2020      30.11.2021        01.12.2019       31.07.2019
[13] 15.08.2019      30.09.2022        01.05.2021       01.11.2020
[17] 01.02.2020      30.11.2020        31.07.2020       20.12.2019
[21] 01.11.2019      01.06.2020        30.04.2021       31.03.2021
[25] 01.04.2021      31.03.2020        01.09.2019       01.10.2019
[29] 14.06.2019      01.07.2019        01.03.2021       01.01.2021
[33] 25.06.2020      01.01.2020        31.10.2019       01.10.2020
```

```
[37] 29.11.2019        01.04.2020        30.08.2020        17.10.2020
[41] 30.06.2020        01.07.2020        01.03.2020        15.07.2019
[45] 01.05.2020        01.12.2020        30.07.2019        01.02.2021
[49] 01.08.2020        15.06.2021        31.05.2020        31.12.2019
[53] 02.12.2019        15.01.2020        31.05.2021        01.11.2021
[57] 15.08.2020        31.08.2019        24.07.2019        31.08.2021
[61] 01.08.2019        15.12.2019        31.01.2020        30.09.2019
[65] 30.09.2020        02.09.2019        16.12.2019        13.12.2020
[69] 02.04.2021        28.02.2020        30.08.2019        15.06.2020
[73] 15.10.2019        28.02.2021        30.08.5292        15.04.2020
[77] 01.07.2021        05.01.2020        23.10.2020        30.09.2021
[81] 30.11.2022        30.07.2020        31.10.2020
83 Levels:  01.01.2020 01.01.2021 01.02.2020 01.02.2021 ... sofort
```

```r
# Reshape the date_available variable
houses$date_available <- as.character(houses$date_available)
houses$date_available[houses$date_available == "" |
                      houses$date_available == "30.08.5292"] <- "01.01.1990"
houses$date_available[houses$date_available == "nach Vereinbarung"] <- "01.01.1991"
houses$date_available[houses$date_available == "sofort"] <- "01.01.1992"
houses$date_available <- as.Date(houses$date_available, "%d.%m.%Y")
for(i in unique(houses$date_available)){
  dates <- houses$date_available == i
  if(day(head(houses$date_available[dates],1)) > 15 &
     month(head(houses$date_available[dates],1)) != 12){
       day(houses$date_available[dates]) <- 1
       month(houses$date_available[dates]) <- month(houses$date_available[dates]) + 1
  }
   else if(day(head(houses$date_available[dates],1)) > 15 &
           month(head(houses$date_available[dates],1)) == 12){
             month(houses$date_available[dates]) <- 1
             year(houses$date_available[dates]) <- year(houses$date_available[dates]) + 1
  }
}
houses$date_available <- as.yearmon(paste(year(houses$date_available),
                                          month(houses$date_available)), "%Y %m")
houses$date_available <- as.character(houses$date_available)
houses$date_available[houses$date_available == "Jan 1990"] <- "keine Angabe"
houses$date_available[houses$date_available == "Jan 1991"] <- "nach Vereinbarung"
houses$date_available[houses$date_available == "Jan 1992"] <- "sofort"
houses$date_available <- as.factor(houses$date_available)
```

Since we want to predict the prices for observations in the test data, we need to apply the same data cleaning procedure on the test data as we did for the independent variables of the training data. Furthermore, we decide to make use of the joint dataset (i.e. training and test data combined) later on to generate impuation values for those variables, which still contain missing values.

```r
# Load the test set
x_test <- read.csv("X_test_rawdata.csv")

# Set the id to the index
row.names(x_test) <- x_test$id
x_test$id <- NULL

# Drop the sale variable
x_test$sale <- NULL

# Impute the missing values to the dummy variables
for (i in c(1:length(x_test))){
```

```
  if(is.integer(x_test[,i])){
    if(sd(x_test[,i], na.rm = T) == 0){
      x_test[is.na(x_test[,i]),i] <- as.integer(0)
    }
  }
}

# Impute the missing values to the floor variable
x_test$floors[is.na(x_test$floors)] <- as.integer(0)

# Transform the floor variable
x_test$floors <- as.factor(x_test$floors)

# Reshape the date variable
x_test$date <- as.Date(x_test$date, "%d.%m.%Y")
x_test$date <- as.yearmon(paste(year(x_test$date), month(x_test$date)), "%Y %m")
x_test$date <- as.character(x_test$date)
x_test$date <- as.factor(x_test$date)

# Generate the age variable and remove the year and year_built variable
x_test$age <- x_test$year - x_test$year_built
x_test$age[x_test$age < 0] <- 0
x_test$year <- NULL
x_test$year_built <- NULL

# Remove the unused geographical variables
x_test$address <- NULL
x_test$street <- NULL

# Transform the zipcode variable
x_test$zipcode <- as.factor(x_test$zipcode)

# Reshape the date_available variable
x_test$date_available <- as.character(x_test$date_available)
x_test$date_available[x_test$date_available == "" |
                      x_test$date_available == "30.08.5292"] <- "01.01.1990"
x_test$date_available[x_test$date_available == "nach Vereinbarung"] <- "01.01.1991"
x_test$date_available[x_test$date_available == "sofort"] <- "01.01.1992"
x_test$date_available <- as.Date(x_test$date_available, "%d.%m.%Y")
for(i in unique(x_test$date_available)){
  dates <- x_test$date_available == i
  if(day(head(x_test$date_available[dates],1)) > 15 &
     month(head(x_test$date_available[dates],1)) != 12){
       day(x_test$date_available[dates]) <- 1
       month(x_test$date_available[dates]) <- month(x_test$date_available[dates]) + 1
  }
   else if(day(head(x_test$date_available[dates],1)) > 15 &
           month(head(x_test$date_available[dates],1)) == 12){
             month(x_test$date_available[dates]) <- 1
             year(x_test$date_available[dates]) <- year(x_test$date_available[dates]) + 1
  }
}
x_test$date_available <- as.yearmon(paste(year(x_test$date_available),
                                          month(x_test$date_available)), "%Y %m")
x_test$date_available <- as.character(x_test$date_available)
x_test$date_available[x_test$date_available == "Jan 1990"] <- "keine Angabe"
x_test$date_available[x_test$date_available == "Jan 1991"] <- "nach Vereinbarung"
```

```
x_test$date_available[x_test$date_available == "Jan 1992"] <- "sofort"
x_test$date_available <- as.factor(x_test$date_available)
```

As soon as we cleaned most of the variables in our datasets, we are now left with the variables for which we need to impute the missing values: *area*, *area_useable*, *rooms* and *age*. First, we observe that the *area* variable and *area_useable* variable often coincide. Hence, we decide to impute the values from *area* into *area_useable* for each observation, where the value for *area* is available but the one for *area_useable* is not. For the residual missing values we decide to produce impuation values using the **mice()** function. This function performs multivariate imputation by chained equations and produces multiple imputations for each variable which contains missing values. To get appropriate impuation values we decide to generate seven imputations for each missing value and then take the median of them as our final imputation value. Note that we exclude the *municipality* variable in both sets and that we do not use the *price* variable to impute the missing values, since *price* is our response variable and therefore only available in the training data.

```
# Combine the training and the test data to one big dataset
joint <- rbind(houses[-c(20,29)],x_test[-20])

# Impute the values of area into area_usable whenever available
joint$area_useable[!is.na(joint[,"area"]) & is.na(joint[,"area_useable"])] <-
  joint$area[!is.na(joint[,"area"]) & is.na(joint[,"area_useable"])]

# Run the mice function to creat imputation values
joint_imp <- mice(joint, m = 7, maxit = 0, method = "pmm", seed = 107)
```

area:

```
# Impute the generated values for the area variable
joint$area[is.na(joint[,"area"])] <- rowMedians(as.matrix(joint_imp$imp$area))
```

area_useable:

```
# Impute the generated values for the area_useable variable
joint$area_useable[is.na(joint[,"area_useable"])] <-
  rowMedians(as.matrix(joint_imp$imp$area_useable))
```

rooms:

```
# Impute the generated values for the rooms variable
joint$rooms[is.na(joint[,"rooms"])] <- rowMedians(as.matrix(joint_imp$imp$rooms))
```

age:

```
# Impute the generated values for the age variable
joint$age[is.na(joint[,"age"])] <- rowMedians(as.matrix(joint_imp$imp$age))

# Check if there are any missing values left in the data
sum(is.na(joint))
```

```
[1] 0
```

As we can see, we finally managed to produce a dataset without any missing values. We now need to split up the data again to return our original training and test datasets.

```
# Extract the training data
houses[,-c(20,29)] <- joint[c(1:nrow(houses)),]

# Extract the test data
x_test[, -20] <- joint[c((nrow(houses)+1):(nrow(joint))),]
```

Since we expect that for some models that we have in mind the large number of factors within the *zipcode* variable may exhaust the computational power of our machines later on, we decide to create a variable which categorizes the *zipcode* variable into nine different levels. To do so, we acquired a list from the

"Bundesamt für Statistik" in which each zipcode is assigned to one of nine local authority types. Hence, we could use zipcodes as a reference to create the *kategorie* variable, which contains the corresponding local authority type for each observation.

```
# Generate the kategorie variable for the training set
PLZ <- read.csv("PLZ nach Kategorie.csv")
PLZ$KAT <- as.character(PLZ$KAT)
houses$kategorie <- NA
for (i in unique(PLZ$PLZ4)){
  houses$kategorie[houses$zipcode == i] <- PLZ$KAT[PLZ$PLZ4 == i]
}

# Since the category for the zipcode 2500 (i.e. Biel) is missing, we impute it manually
houses$kategorie[is.na(houses$kategorie)] <-
  "Städtische Gemeinde einer mittelgrossen Agglomeration (12)"

# Transform the variable into a factor
houses$kategorie <- as.factor(houses$kategorie)

# Do the same for the test set
x_test$kategorie <- NA
for (i in unique(PLZ$PLZ4)){
  x_test$kategorie[x_test$zipcode == i] <- PLZ$KAT[PLZ$PLZ4 == i]
}
x_test$kategorie[is.na(x_test$kategorie)] <-
  "Städtische Gemeinde einer mittelgrossen Agglomeration (12)"
x_test$kategorie <- as.factor(x_test$kategorie)
```

As mentioned before, we noticed that some changes are necessary for the *municipality* variable. Since the original list of municipalities is very messy, we began by tidying it up (using Microsoft Excel). First, we changed all the unclear characters; this included characters such as ä, ö, ü, é and so on. Next, we got rid of unnecessary spacing and added spaces where needed. Then, we acquired a current list of municipalities. Using the zipcode as a reference we updated the original list to reflect the current statues. This included merging municipalities that no longer exist and were merged to form new municipalities. Furthermore, we acquired another list containing average square-meter prices per municipality and made use of this information to create two further variables: (1) *municipality_avg_m2_price*, which contains the average square meter price in the corresponding municipality and (2) *area_times_municipality_avg_m2_price*, which contains an interaction between the variables *area* and *municipality_avg_m2_price*. One thing to keep in mind here is the multicollinearity issue which may arise with the variables *area*, *municipality_avg_m2_price* and *area_times_municipality_avg_m2_price*.

```
# Generat the desired variables for the training set
mun_train <- read_xlsx("Data from Sophia/mun_train.xlsx")
houses$municipality <- as.factor(mun_train$municipality)
houses <- cbind(houses,mun_train[,-1])

# Generat the desired variables for the test set
mun_test <- read_xlsx("Data from Sophia/mun_test.xlsx")
x_test$municipality <- as.factor(mun_test$municipality)
x_test <- cbind(x_test,mun_test[,-1])
```

Last but not least, we should take care of the response variable (i.e. *price*). In the given dataset the inputs of our response is a categorical variable and includes the notion "CHF" of the swiss currency in the end. However, we want to produce a numerical response without any currency-indicating characters in it. We do this in the following way.

```
# Extract only the digits from the price variable
houses$price <- as.numeric(gsub("[^[:digit:].]", "",  as.character(houses$price)))
```

```
# Get summary statistics of all numerical variables from the cleaned training data
stargazer(houses, type = "text", median = T)
```

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Median	Pctl(75)	Max
area	35,000	116.174	46.387	1	87	111	137	882
area_useable	35,000	118.398	57.143	1	88	113	140	4,667
balcony	35,000	0.502	0.500	0	0	1	1	1
basement	35,000	0.042	0.201	0	0	0	0	1
bath_tube	35,000	0.017	0.128	0	0	0	0	1
building_plot	35,000	0.0003	0.017	0	0	0	0	1
cabletv	35,000	0.225	0.418	0	0	0	0	1
ceiling	35,000	0.001	0.031	0	0	0	0	1
cheminee	35,000	0.123	0.328	0	0	0	0	1
elevator	35,000	0.479	0.500	0	0	0	1	1
first_time	35,000	0.019	0.137	0	0	0	0	1
furnished	35,000	0.002	0.040	0	0	0	0	1
kids_friendly	35,000	0.196	0.397	0	0	0	0	1
laundry	35,000	0.030	0.172	0	0	0	0	1
minergie	35,000	0.019	0.138	0	0	0	0	1
new_building	35,000	0.063	0.243	0	0	0	0	1
newly_built	35,000	0.253	0.435	0	0	0	1	1
oldbuilding	35,000	0.005	0.071	0	0	0	0	1
oven	35,000	0.002	0.041	0	0	0	0	1
parking_indoor	35,000	0.304	0.460	0	0	0	1	1
parking_outside	35,000	0.234	0.423	0	0	0	0	1
playground	35,000	0.007	0.084	0	0	0	0	1
pool	35,000	0.005	0.073	0	0	0	0	1
quiet	35,000	0.002	0.041	0	0	0	0	1
raised_groundfloor	35,000	0.003	0.059	0	0	0	0	1
rooms	35,000	4.112	1.199	1.000	3.500	4.500	4.500	15.000
sunny	35,000	0.001	0.032	0	0	0	0	1
terrace	35,000	0.013	0.113	0	0	0	0	1
topstorage	35,000	0.006	0.078	0	0	0	0	1
veranda	35,000	0.001	0.029	0	0	0	0	1
wheelchair	35,000	0.096	0.295	0	0	0	0	1
age	35,000	15.216	30.117	0	1	6	21	822
municipality_avg_m2_price	35,000	215.430	49.804	109	183	200	238	397
area_times_municipal...	35,000	25,407.350	13,164.520	152	17,104.8	22,829.5	30,394	167,726
price	35,000	886,258.400	668,930.700	300	480,000	695,000	1,087,250	5,850,000

```r
# Get an overview of the classes of the variables
str(houses)
```

```
'data.frame':   35000 obs. of  42 variables:
 $ area                                : num  60 137 134 245 111 100 84 38 172 133 ...
 $ area_useable                        : num  60 137 142 245 111 100 84 38 172 133 ...
 $ balcony                             : int  1 0 0 0 0 1 1 1 0 0 ...
 $ basement                            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ bath_tube                           : int  0 0 0 0 0 0 0 0 0 0 ...
 $ building_plot                       : int  0 0 0 0 0 0 0 0 0 0 ...
 $ cabletv                             : int  0 0 0 0 0 0 0 0 0 0 ...
 $ ceiling                             : int  0 0 0 0 0 0 0 0 0 0 ...
 $ cheminee                            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ date                                : Factor w/ 93 levels "Apr 2012","Apr 2013",..: 87 16 66 53 8
 $ date_available                      : Factor w/ 37 levels "Apr 2020","Apr 2021",..: 20 20 20 20 2
 $ elevator                            : int  1 0 0 0 0 1 0 1 0 0 ...
 $ first_time                          : int  0 0 0 0 0 0 0 0 0 0 ...
 $ floors                              : Factor w/ 28 levels "-1","0","1","2",..: 2 2 4 2 2 3 2 3 4
 $ furnished                           : int  0 0 0 0 0 0 0 0 0 0 ...
 $ home_type                           : Factor w/ 8 levels "Attika","Dachwohnung",..: 8 8 8 8 8 8 8
 $ kids_friendly                       : int  0 0 0 0 0 1 0 0 0 0 ...
 $ laundry                             : int  0 0 0 0 0 0 0 0 0 0 ...
 $ minergie                            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ municipality                        : Factor w/ 1485 levels "Aadorf","Aarau",..: 903 223 262 1195
 $ new_building                        : int  0 0 0 0 0 0 0 0 0 0 ...
 $ newly_built                         : int  1 0 0 0 0 0 0 0 0 0 ...
 $ oldbuilding                         : int  0 0 0 0 0 0 0 0 0 0 ...
 $ oven                                : int  0 0 0 0 0 0 0 0 0 0 ...
 $ parking_indoor                      : int  1 0 1 0 0 1 0 0 1 0 ...
 $ parking_outside                     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ playground                          : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pool                                : int  0 0 0 0 0 0 0 0 0 0 ...
 $ quiet                               : int  0 0 0 0 0 0 0 0 0 0 ...
 $ raised_groundfloor                  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ rooms                               : num  2.5 3.5 5.5 5.5 3.5 3.5 4.5 1.5 6.5 4.5 ...
 $ sunny                               : int  0 0 0 0 0 0 0 0 0 0 ...
 $ terrace                             : int  0 0 0 0 0 0 0 0 0 0 ...
 $ topstorage                          : int  0 0 0 0 0 0 0 0 0 0 ...
 $ veranda                             : int  0 0 0 0 0 0 0 0 0 0 ...
 $ wheelchair                          : int  0 0 0 0 0 0 0 0 0 0 ...
 $ zipcode                             : Factor w/ 2473 levels "1000","1003",..: 367 1534 22 504 75
 $ age                                 : num  0 5 11 8 32 3 38 35 3 7 ...
 $ kategorie                           : Factor w/ 9 levels "Ländliche periphere Gemeinde (33)",..:
 $ municipality_avg_m2_price           : num  182 212 270 173 230 234 245 221 145 178 ...
 $ area_times_municipality_avg_m2_price: num  10915 29072 36215 42401 25555 ...
 $ price                               : num  235000 1570000 1095000 1660000 745000 ...
```

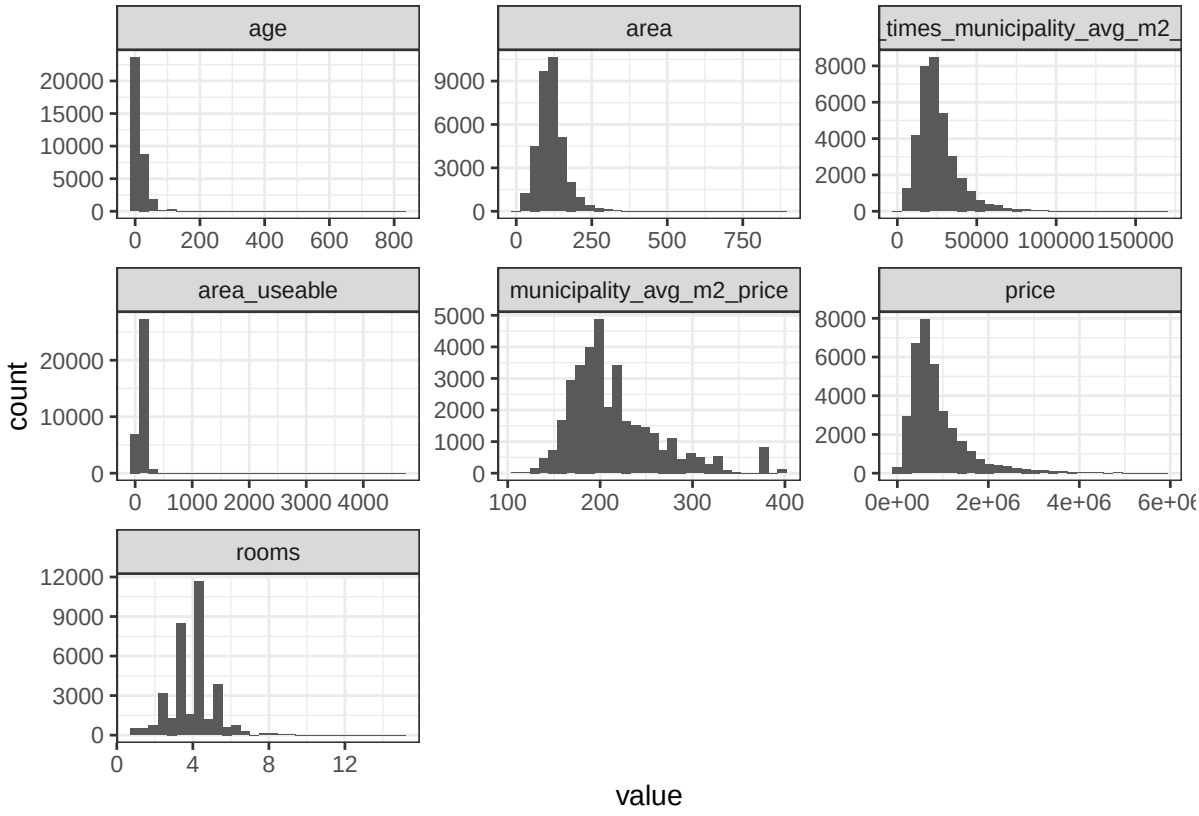
As we can see, we finally brought the data into the desired shape. Hence, we are now able to start with the estimation of our prediction models in the subsequent sections. The very last step we apply is to extract the final datasets, such that we can load them from an external source later on.

```r
# Extract the training set as an csv file
write.csv(houses,
 "D:\\Dokumente\\Yannik\\UZH\\20FS\\Big Data Methods for Economists (S)\\Final Project\\houses.csv",
  row.names = F)
# Extract the test set as an csv file
write.csv(x_test,
 "D:\\Dokumente\\Yannik\\UZH\\20FS\\Big Data Methods for Economists (S)\\Final Project\\x_test.csv",
 row.names = F)
```

Exploratory Data Analysis

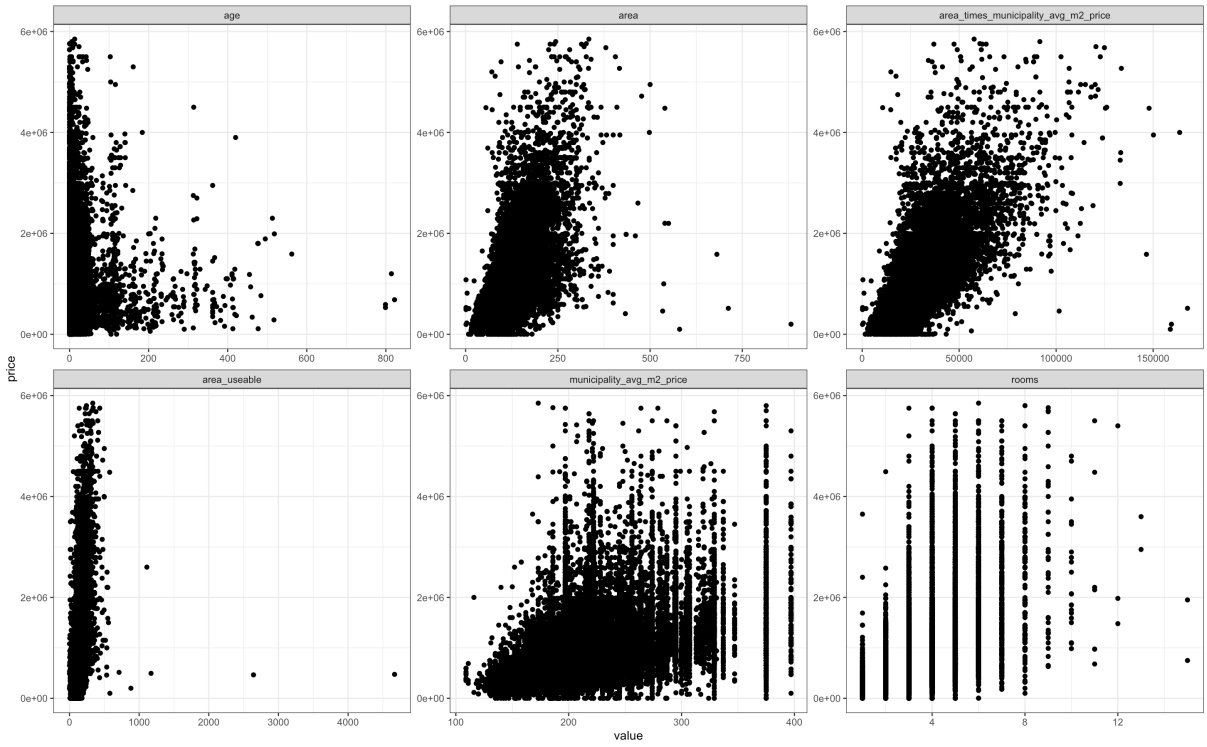
In the next step we do some exploratory data anylsis. To do so, we create a histogram for each of the numerical variables, such that the values are on the x-axis and the count is on the y-axis.

```
houses[,c(1,2,31,38,40,41,42)] %>%
  gather() %>%
  ggplot(aes(value)) +
  facet_wrap(~ key, scales = "free") +
  geom_histogram() +
  theme_bw()
```



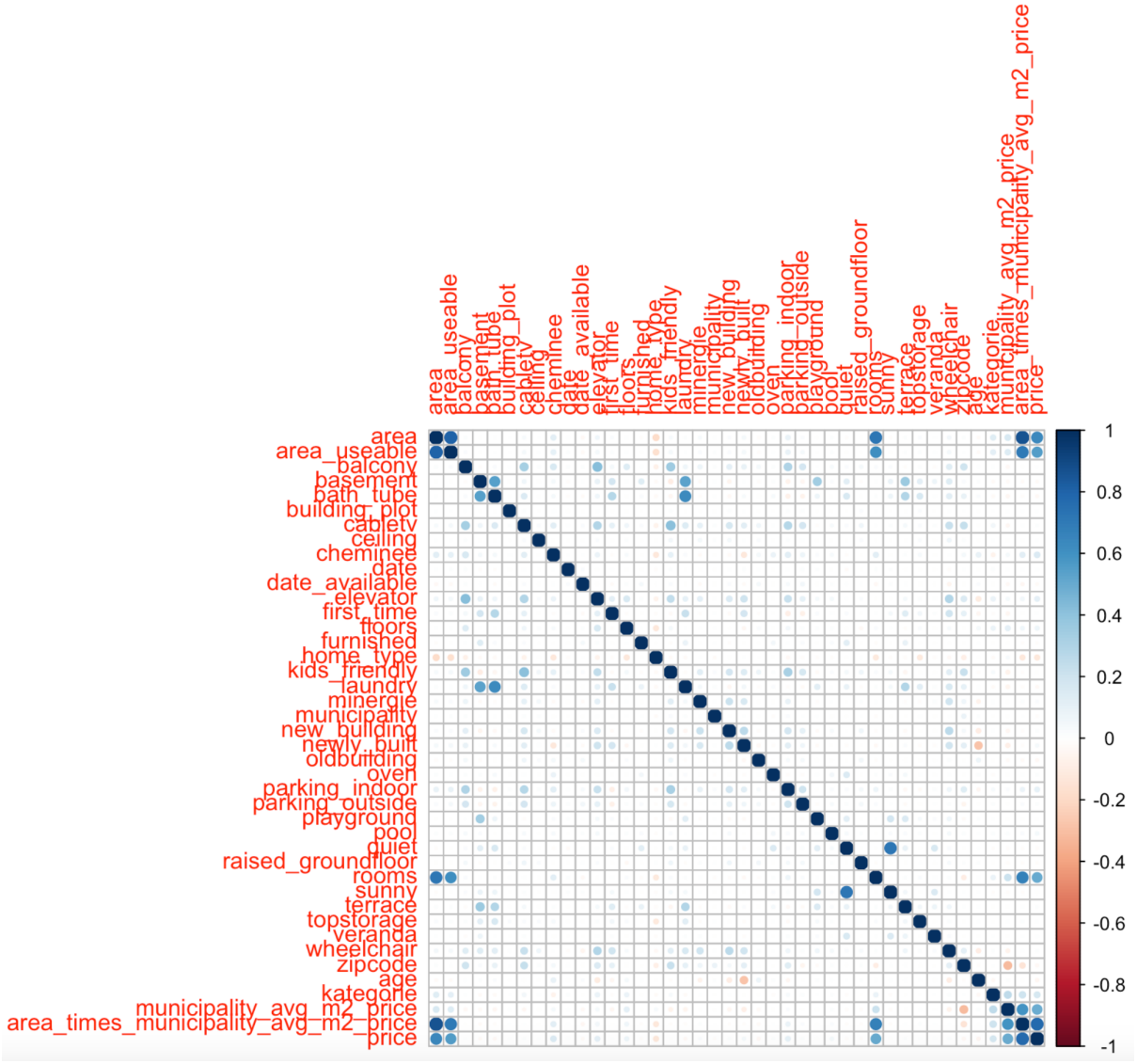
Next, we plot each of the numerical variable against the response variable (i.e. price) in order to visualize whether there is a relationship between the said variable and the response and if there is a relationship, what this relationship looks like.

```
houses[,c(1,2,31,38,40,41,42)] %>%
  gather(-price, key = "var", value = "value") %>%
  ggplot(aes(x = value, y = price)) +
  geom_point() +
  facet_wrap(~ var, scales = "free") +
  theme_bw()
```



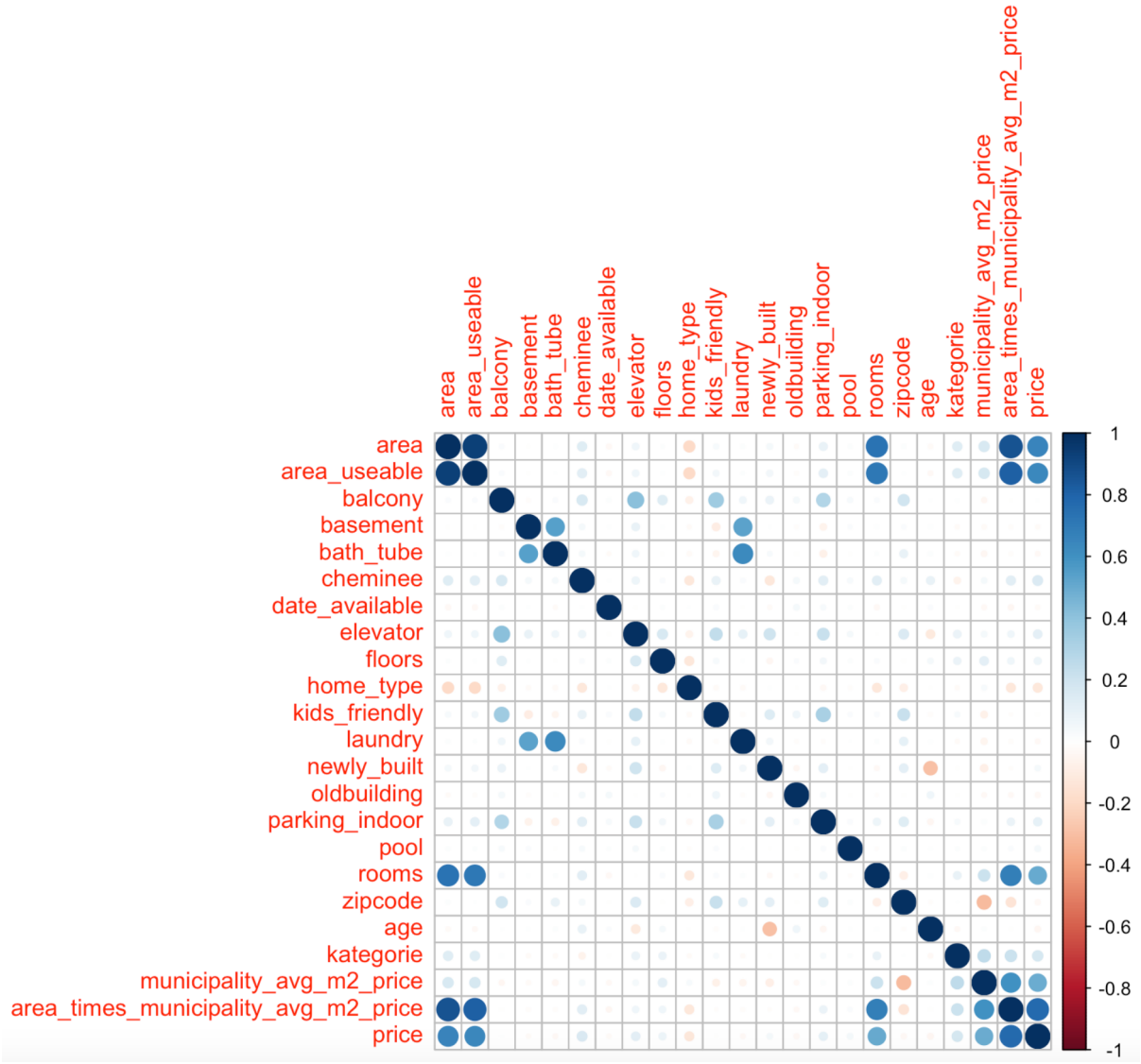
Next, we create a correlation plot to take a look at the correlations between the variables.

```
houses_numeric <- data.matrix(houses)
corrmatrix <- cor(houses_numeric,use="pairwise")
corrplot(corrmatrix, method="circle")
```



Then we create a second correlation plot. In this plot we choose only the variables, which show some sort of correlation to price. Furthermore, we remove observations with more than 30% NAs and get rid of outliers. (Note that we only count NAs among those variables, for which we did not calculate imputation values above (i.e. all dummy variables))

```
houses2 <- houses
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0

houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]

houses2 <- houses2[,c(1:5,9,11,12,14,16,17,18,22,23,25,28,31,37,38,39,40,41,42)]

houses_numeric <- data.matrix(houses2)
corrmatrix <- cor(houses_numeric,use="pairwise")
corrplot(corrmatrix, method="circle")
```


area
area_useable
balcony
basement
bath_tube
cheminee
date_available
elevator
floors
home_type
kids_friendly
laundry
newly_built
oldbuilding
parking_indoor
pool
rooms
zipcode
age
kategorie
municipality_avg_m2_price
area_times_municipality_avg_m2_price
price
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

Multiple Linear Regression

The first type of model we try is the simple Multiple Linear Regression (MLR). The advantage of this method is that it is simple and easy to interpret. The disadvantage of this model is that it may be too simple, in the sense that it only allows for linear relationships.

Model 1

In this section we create a Multiple Linear Regression Model in order to predict the price variable.

We begin by preparing and reshaping the dataset. We remove the variables kategorie and municipality, because these two variables appear to increase the MAE of the model.

```
houses2 <- houses[-39]
houses2 <- houses2[-20]
```

Next, we split the data into training and test set. We set a seed randomly. Then we split the dataset, in order to have 90% of the data in the training set and 10% of the data in the test set. Since the variable zipcode presents itself as problematic, as it may be present in the test set, but not the training set, we add a random vector that contains each zipcode once to the training set. This will lead us to take the MAE with a grain of salt.

```
set.seed(125)
split = sample.split(houses2$price, SplitRatio = 0.9)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
every_zipcode <- stratified(houses2, "zipcode", 1)
training_set <- rbind(training_set,every_zipcode)
```

We create our first MLR model using the training set.

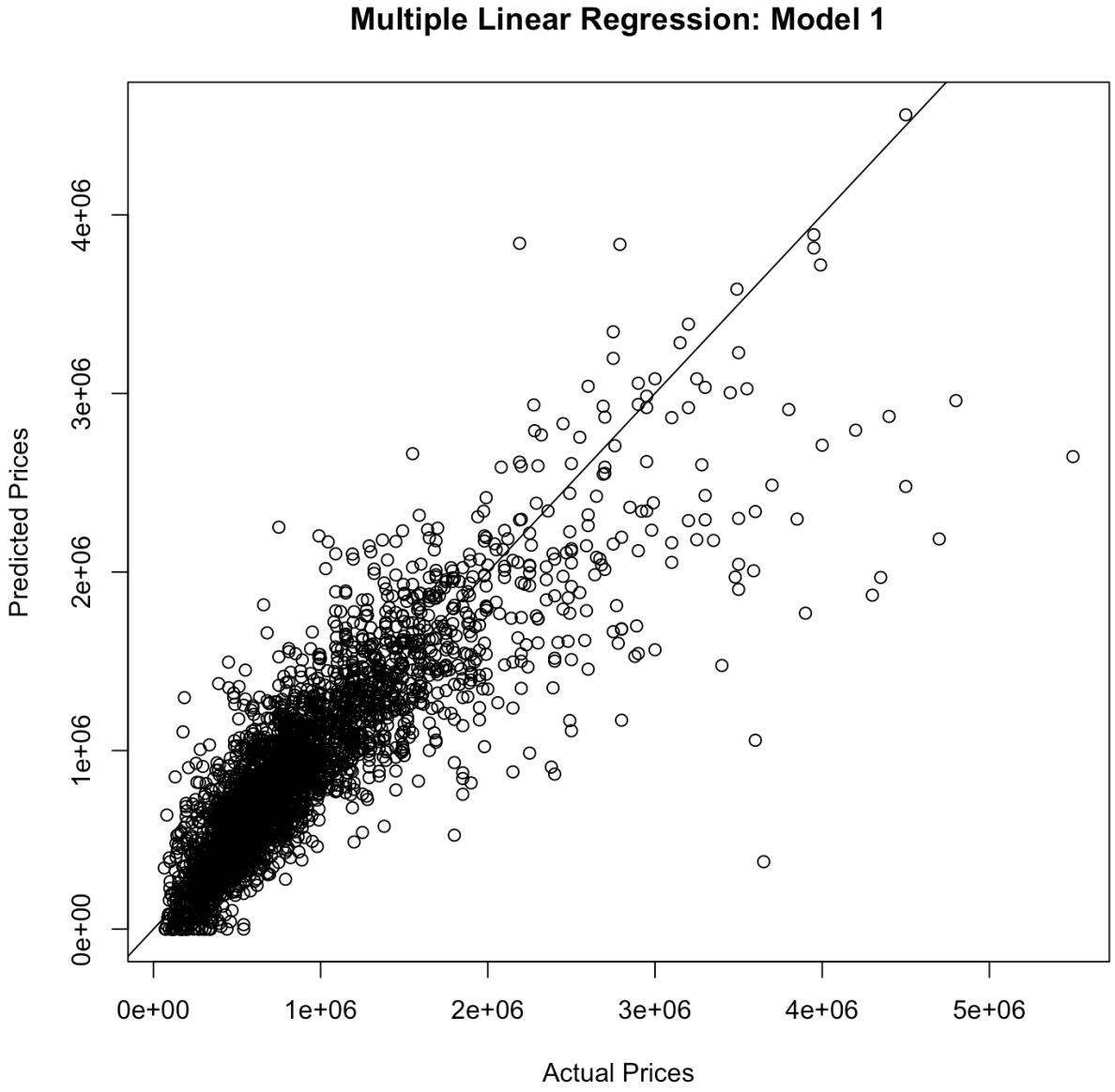
```
model = lm(formula = price ~ .,
               data = training_set)
```

We apply the model to the test set. If the predicted price is under 0, then we substitute that price with the value of 0.

```
predictions = predict(model, newdata = test_set)
predictions[predictions<0] <- 0
```

We plot and evaluate the results. To the plot we add a 45° degree line. The MAE for this model is 201'794.2. This must be taken with a grain of salt, as we used part of the test set during training.

```
plot(test_set$price,predictions, main = "Multiple Linear Regression: Model 1",
     xlab ="Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
mean((test_set$price-predictions)^2)
mae(test_set$price,predictions)
# MAE = 201794.2
```


Multiple Linear Regression: Model 1
Predicted Prices
Actual Prices
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06

Model 2

In this Model we create a Multiple Linear Regression Model in order to predict the price variable.

We reshape the data. As in the previous model, we remove the variable kategorie. Additionally, we remove observations with more than 30% NAs. We also do some outlier detection and remove some outliers, pertaining to the variables area, area_useable and rooms.

```
houses2 <- houses[-20]
houses2 <- houses2[complete.cases(houses2), ]
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
```

We split the data into training and test set. We do this in the same manner as we did in Model 1.

```
set.seed(127)
split = sample.split(houses2$price, SplitRatio = 0.9)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
every_zipcode <- stratified(houses2, "zipcode", 1)
training_set <- rbind(training_set,every_zipcode)
```

We create the model using the training set, as in Model 1.

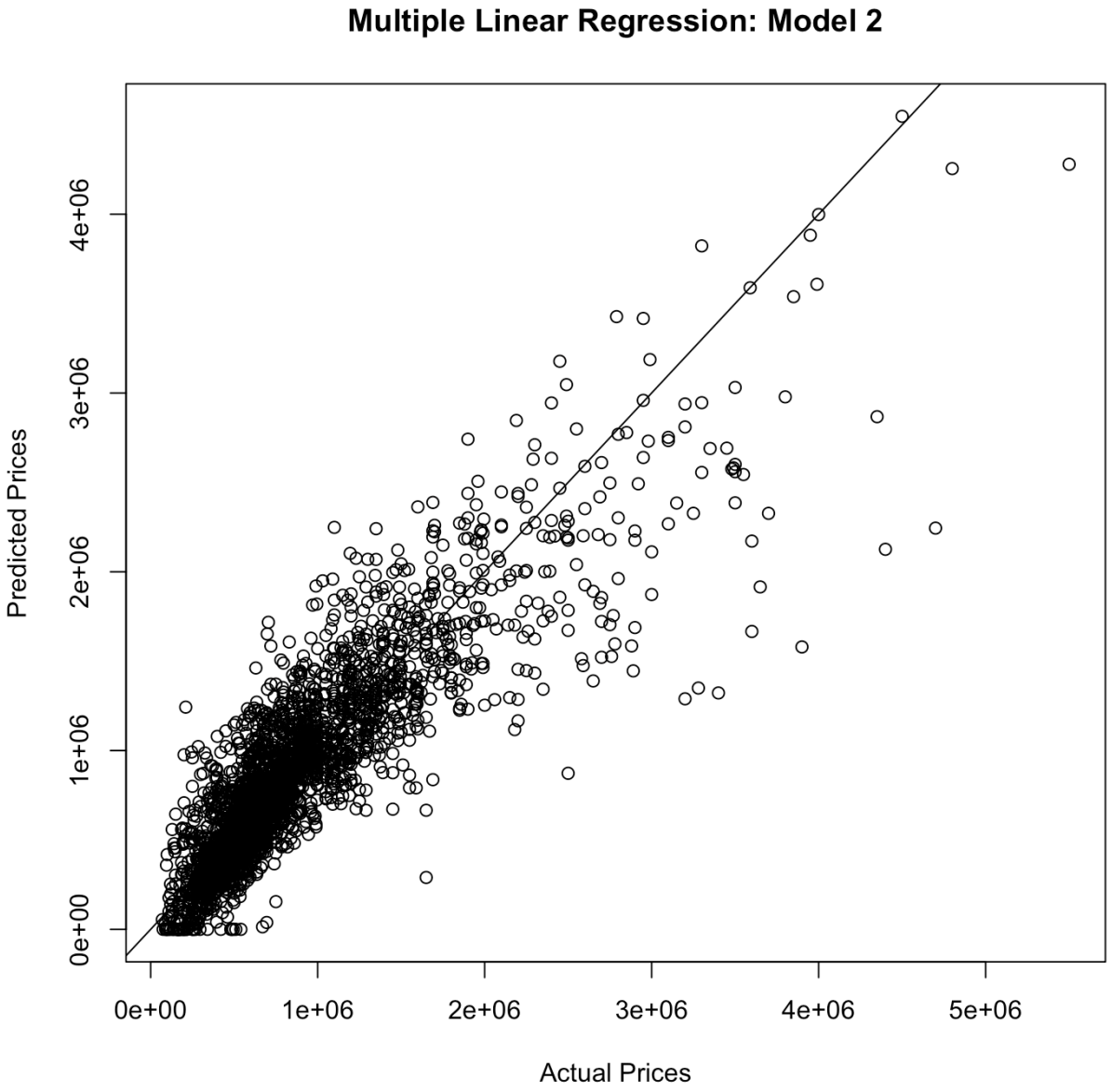
```
model = lm(formula = price ~ .,
               data = training_set)
```

We apply the model to the test set and replace negative predictions with a value of 0.

```
predictions = predict(model, newdata = test_set)
predictions[predictions<0] <- 0
```

We plot and evaluate the results. For this MLR Model 2, we get an MAE of 200'449.2

```
plot(test_set$price,predictions, main = "Multiple Linear Regression: Model 2",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
mean((test_set$price-predictions)^2)
mae(test_set$price,predictions)
# MAE = 200449.2
```


Multiple Linear Regression: Model 2
Predicted Prices
0e+00
1e+06
2e+06
3e+06
4e+06
Actual Prices
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06

Model 3

We create a third version of a MLR Model.

We reshape the data. For this model, we remove the variable kategorie and the variable municipality. Additionally, we remove observations with more than 30% NAs. We don't however remove outliers.

```
houses2 <- houses[-39]
houses2 <- houses2[-20]

houses2 <- houses2[complete.cases(houses2), ]
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0
```

We split the data into training and test set. We do this in the same manner as we did in Model 1 and Model 2.

```
set.seed(123)
split = sample.split(houses2$price, SplitRatio = 0.9)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
every_zipcode <- stratified(houses2, "zipcode", 1)
training_set <- rbind(training_set,every_zipcode)
```

We create the model using the training set, as in Model 1 and Model 2.

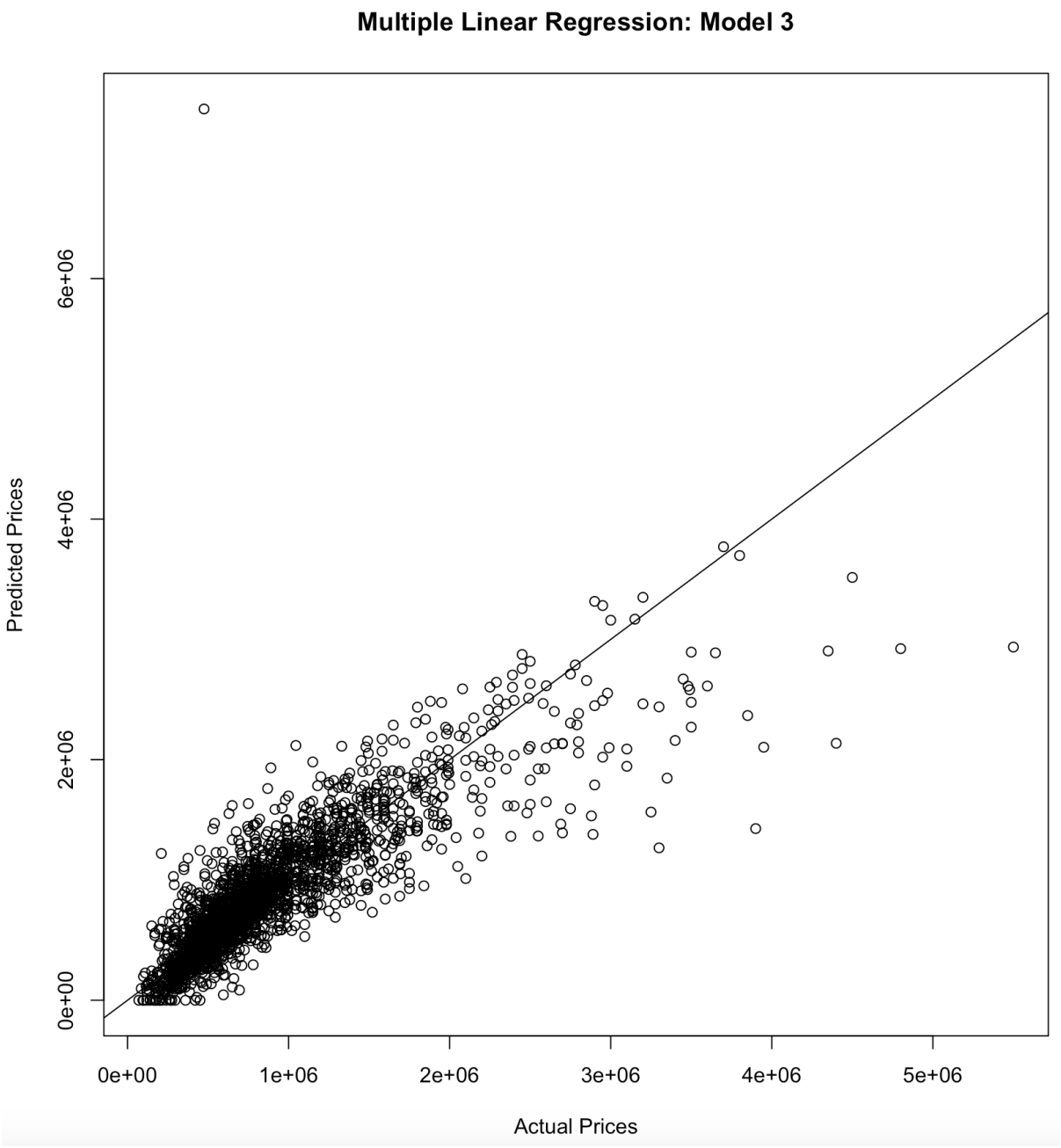
```
model = lm(formula = price ~ .,
               data = training_set)
```

We apply the model to the test set and replace negative predictions with a value of 0.

```
predictions = predict(model, newdata = test_set)
predictions[predictions<0] <- 0
```

We plot and evaluate the results. For this MLR Model 3, we get an MAE of 198354.6, leading Model 3 to be the best variant of the MLR models.

```
plot(test_set$price,predictions, main = "Multiple Linear Regression: Model 3",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
mean((test_set$price-predictions)^2)
mae(test_set$price,predictions)
# MAE = 198354.6
```


Multiple Linear Regression: Model 3
Predicted Prices
0e+00
2e+06
4e+06
6e+06
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06
Actual Prices

Ridge & Lasso Regression

In this section, we test several Ridge and Lasso Regression Models to predict prices of apartments. These models have the advantage that they trade off the ballance between variance and bias quite well. Compared to the simple MLR model, Ridge and Lasso Regression models reduce the variance of the model in turn of a slight increase of the bias. In addition, the Lasso Regression (in contrast to the Ridge Regression) has the advantage that it performs variable selection. The disandvantage of these models is that they require much compuational power if the model includes many explanatory variables or factorial variables with many categories (e.g. municipality and zipcode variable in our dataset). We evaluate each model according to three aspects (decreasing in relevance):

- Performance (evaluated using the mean average error (MAE) on the training data)
- Robustness (evaluated using the standard deviation (SD) of the MAE)
- Required computational power

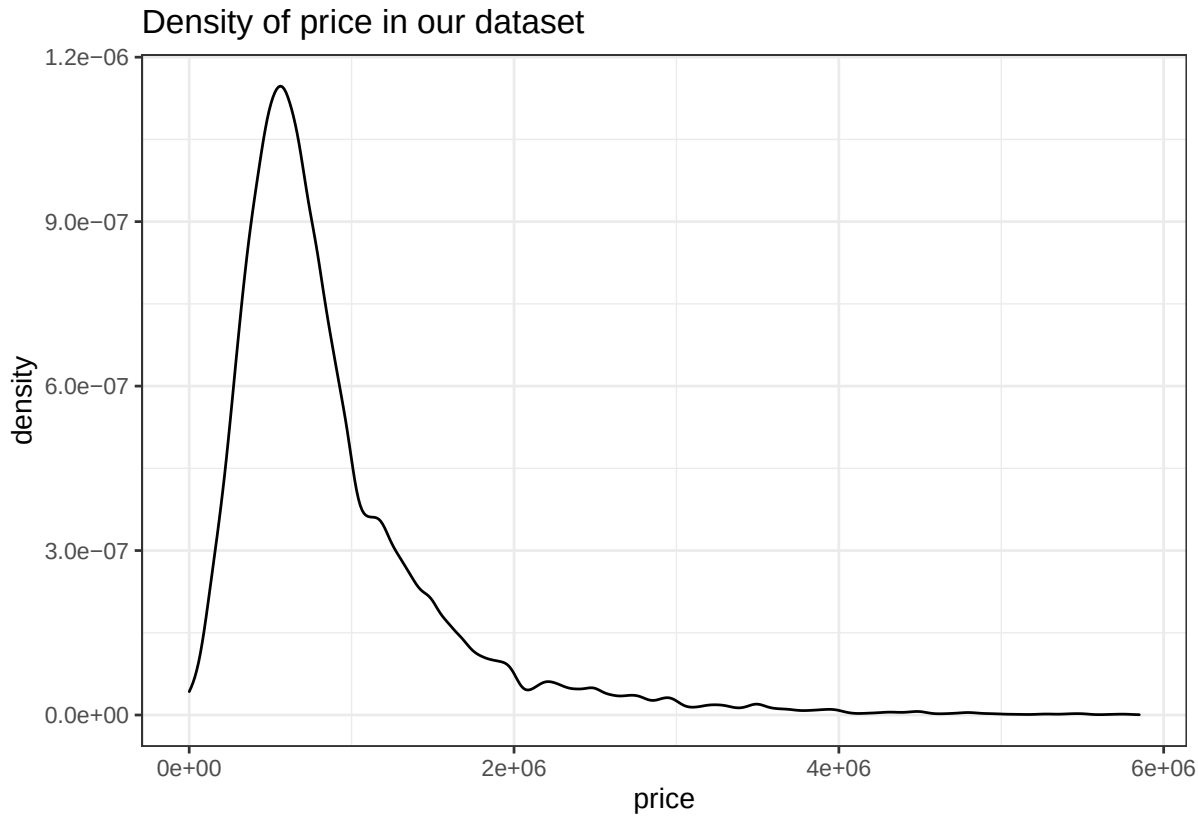
To do so, we evaluate each model 5 times and store the resulting MAE. In each of these 5 "runs", we split our houses dataset differently into a traing and a test set. Futhermore, we change the seed prior to the 7-fold crossvalidation in each run. We decided to run a 7-fold cross validation, because it appeared to lead to quite robust results.

Before we can start, we need to read in the data in a first step. We also have to reshape it, such that the variables we want to consider as factors are defined as such. This needs to be done because we read in the cleaned data from the external file we created in the end of the data cleaning section.

```
# Load the data
houses <- read.csv("houses.csv")
houses$floors <- as.factor(houses$floors)
houses$zipcode <- as.factor(houses$zipcode)
```

Furthermore, we take a look at the distribution of our prices.

```
# Create a density plot of the prices in our data to assess their distribution type
ggplot(data = houses, aes(price)) +
  geom_density() +
  theme_bw() +
  ggtitle("Density of price in our dataset")
```



We observe that the distribution of our response is most likely to be explained by a gaussian distribution. Hence, we don't have to specify this in the subsequent models, because the functions of the glmnet package we apply later on assumes a gaussian distribution for the response variable by default.

Before we start running our models, we spend some time to think about which variables are reasonable to include in the models. Since factorial variables with a large number of categories (i.e. zipcode, municipality) tend to require extreme amounts of computational power, we want to assess, which of these variables really lead to a better performing model. We do this by performing 4 different Ridge and Lasso models, from which each one relies on another type or combination of local authority indicating variables (i.e. kategorie, zipcode, municipality). In addition, we decide to exclude the factorial variable date_available, since models which include this variable appear to lead to quite similar results as models which exclude it. We do this to save computational capacity.

Model 1

In the first model we run a Ridge and a Lasso Regression using only the kategorie variable as a local authority indicating variable. We use this variable, since it contains much less different categories than the zipcode and municipality variable and therefore safes much compuational power. However, using kategorie instead of zipcodes or municipalities could lead to a worse performance, due to too stringent simplification of the local authority indicator.

We start by preparing and reshaping the data, such that it is left with only kategorie as a local authority indicating variable. Hence, we drop the variables date_available, municipality and zipcode. Next, we set up a loop to evaluate both models (Ridge and Lasso Regression model) five times. Inside of the loop we split the data into simulated training and test sets, using a splitratio of 0.9. To ensure that the split is done differently in every cycle of the loop, we set a seed which interacts with the loop variable (i.e. i). Thereafter, we transform our data into matrix form, such that the functions of the glmnet package can work with it. As soon as this is done, we are ready to perform the Ridge and Lasso Regression estimations. We start with the Ridge Regression. In a first step we determine the grid of λs for which we want to perform the Ridge Regression. Then we estimate the Ridge model for the chosen grid of λs using the simulated training set and store it into a variable. Thereafter, we identify the best performing λ out of the applied grid (i.e. the λ that leads to the smallest mean squared error) for our model by means of a 7-fold cross validation and store it into a variable. Note that preceding to performing the cross validation we again set a seed which interacts with the loop variable i, such that the 7 folds are done differently in each cycle of the loop. Thereafter, we apply the model with the best λ to the simulated test set to get the price predictions of our model. Since, negative prices do not make much sense, we replace negative predictions with a value of 0 and assess the MAE thereafter. The estimation of the Lasso Regression is then done analogously. After each cycle of the loop we store the resulting MAE of both models into a matrix to enable convenient access of the results later on. Furthermore, for both, the Ridge and the Lasso Regression model, we calculate the average and the standard deviation (SD) of the resulting MAEs after the 5 runs have been performed.

```
#### Model 1
# Create a Matrix to store the different MAEs of the models from each run
Model.1 <- matrix(NA, 6, 2)
colnames(Model.1) <- c("MAE ridge","MAE lasso")
rownames(Model.1) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average")

# Reshape the data
houses2 <- houses[-c(11,20,37)]

for (i in c(1:5)){
  # Split the training dataset into a simulated training- and test set
  set.seed(i*7)
  training_set <- sample(x = c(TRUE,FALSE), size = nrow(houses2), rep = TRUE, prob = c(0.9,0.1))
  test_set <- (!training_set)

  # Transform the data to enable the usage of glmnet functions
  y <- houses2$price
  x <- model.matrix(price ~ ., data = houses2)[,-1]

  ### Ridge
  # Create a vector containg all lambdas to estimate the ridge model
  grid_r <- exp(seq(2, -10, length = 100))

  # Estimating a ridge regression for the desired grid of lambdas
  ridge.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 0, lambda = grid_r)

  # Find the best lambda out of the grid for the ridge regression by cross validation
  set.seed(i*77)
  cv.out.ridge <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 0, lambda = grid_r)
```

```
  ridge.best.lambda <- cv.out.ridge$lambda.min

  ## Assess the performance of the 'best' ridge regression model
  # Predict prices for the created test set
  predictions.r <- predict(object = ridge.mod, s = ridge.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.r[predictions.r < 0] <- 0

  # Calculate the MAE
  Model.1[i,1] <- mae(y[test_set], predictions.r)


  ### Lasso
  # Create a vector containg all lambdas to estimate the lasso model
  grid_l <- exp(seq(8.5, 6.5, length = 100))

  # Estimating a lasso regression for the desired grid of lambdas
  lasso.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 1, lambda = grid_l)

  # Find the best lambda out of the grid for the lasso regression by cross validation
  set.seed(i*77)
  cv.out.lasso <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 1, lambda = grid_l)
  lasso.best.lambda <- cv.out.lasso$lambda.min

  ## Assess the performance of the 'best' lasso regression model
  # Predict prices for the created test set
  predictions.l <- predict(object = lasso.mod, s = lasso.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.l[predictions.l < 0] <- 0

  # Calculate the MAE
  Model.1[i,2] <- mae(y[test_set], predictions.l)

  # Calculate the average MAE
  Model.1[6,1] <- mean(Model.1[c(1:5),1], na.rm = T)
  Model.1[6,2] <- mean(Model.1[c(1:5),2], na.rm = T)
}

# Calculate the SD of the MAE
Model.1 <- rbind(Model.1, c(sd(Model.1[c(1:5),1]),sd(Model.1[c(1:5),2])))
rownames(Model.1) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average","SD")
```

We then take a look at the results.

```
# Take a look at the results
Model.1
```

```
        MAE ridge MAE lasso
1. Run   237506.9  236793.6
2. Run   243164.8  241181.8
3. Run   242513.9  240530.0
4. Run   256422.3  254158.9
5. Run   242853.4  241876.6
Average  244492.3  242908.2
SD         7061.9    6589.1
```

We observe that in this model the Ridge Regression leads to an average MAE of 244492.3 and a SD of 7061.9, while the Lasso Regression leads to an average MAE of 242908.2 and a SD of 6589.1. Hence, the Lasso Regression seems to perform better for this model.

Model 2

In the second model we run a Ridge and a Lasso Regression using only the zipcode variable as a local authority indicating variable. We start by preparing and reshaping the data, such that only the desired variables are included. Hence, we drop the variables date_available, municipality and kategorie. Next, we set up a loop to evaluate both models (Ridge and Lasso Regression model) five times. This loop is set up analogously to the one of the first model, since the estimation procedure remains exactly the same as before.

```
#### Model 2
# Create a Matrix to store the different MAEs of the models from each run
Model.2 <- matrix(NA, 6, 2)
colnames(Model.2) <- c("MAE ridge","MAE lasso")
rownames(Model.2) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average")

# Reshape the data
houses2 <- houses[-c(11,20,39)]

for (i in c(1:5)){
  # Split the training dataset into a simulated training- and test set
  set.seed(i*7)
  training_set <- sample(x = c(TRUE,FALSE), size = nrow(houses2), rep = TRUE, prob = c(0.9,0.1))
  test_set <- (!training_set)

  # Transform the data to enable the usage of glmnet functions
  y <- houses2$price
  x <- model.matrix(price ~ ., data = houses2)[,-1]

  ### Ridge
  # Create a vector containg all lambdas to estimate the ridge model
  grid_r <- exp(seq(0, -10, length = 100))

  # Estimating a ridge regression for the desired grid of lambdas
  ridge.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 0, lambda = grid_r)

  # Find the best lambda out of the grid for the ridge regression by cross validation
  set.seed(i*77)
  cv.out.ridge <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 0, lambda = grid_r)
  ridge.best.lambda <- cv.out.ridge$lambda.min

  ## Assess the performance of the 'best' ridge regression model
  # Predict prices for the created test set
  predictions.r <- predict(object = ridge.mod, s = ridge.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.r[predictions.r < 0] <- 0

  # Calculate the MAE
  Model.2[i,1] <- mae(y[test_set], predictions.r)


  ### Lasso
  # Create a vector containg all lambdas to estimate the lasso model
```

```
  grid_l <- exp(seq(8, 6, length = 100))

  # Estimating a lasso regression for the desired grid of lambdas
  lasso.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 1, lambda = grid_l)

  # Find the best lambda out of the grid for the lasso regression by cross validation
  set.seed(i*77)
  cv.out.lasso <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 1, lambda = grid_l)
  lasso.best.lambda <- cv.out.lasso$lambda.min

  ## Assess the performance of the 'best' lasso regression model
  # Predict prices for the created test set
  predictions.l <- predict(object = lasso.mod, s = lasso.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.l[predictions.l < 0] <- 0

  # Calculate the MAE
  Model.2[i,2] <- mae(y[test_set], predictions.l)

  # Calculate the average MAE
  Model.2[6,1] <- mean(Model.2[c(1:5),1], na.rm = T)
  Model.2[6,2] <- mean(Model.2[c(1:5),2], na.rm = T)
}

# Calculate the SD of the MAE
Model.2 <- rbind(Model.2, c(sd(Model.2[c(1:5),1]),sd(Model.2[c(1:5),2])))
rownames(Model.2) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average","SD")
```

We then take a look at the results.

```
Model.2
```

```
        MAE ridge MAE lasso
1. Run   211139.8  208052.8
2. Run   214062.1  211472.4
3. Run   218411.6  214315.7
4. Run   223507.5  219787.3
5. Run   215041.4  212607.6
Average  216432.5  213247.2
SD         4731.3    4314.4
```

We observe that in this model the Ridge Regression leads to an average MAE of 216432.5 and a SD of 4731.3, while the Lasso Regression leads to an average MAE of 213247.2 and a SD of 4314.4. Hence, the Lasso Regression seems to perform better for this model.

Model 3

In the third model we run a Ridge and a Lasso Regression using only the municipality variable as a local authority indicating variable. We start by preparing and reshaping the data, such that only the desired variables are included. Hence, we drop the variables date_available, kategorie and zipcode. Next, we set up a loop to evaluate both models (Ridge and Lasso Regression model) five times. This loop is set up analogously to the one of the previous model, since the estimation procedure remains exactly the same as before.

```
#### Model 3
# Create a Matrix to store the different MAEs of the models from each run
Model.3 <- matrix(NA, 6, 2)
colnames(Model.3) <- c("MAE ridge","MAE lasso")
rownames(Model.3) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average")

# Reshape the data
houses2 <- houses[-c(11,37,39)]

for (i in c(1:5)){
  # Split the training dataset into a simulated training- and test set
  set.seed(i*7)
  training_set <- sample(x = c(TRUE,FALSE), size = nrow(houses2), rep = TRUE, prob = c(0.9,0.1))
  test_set <- (!training_set)

  # Transform the data to enable the usage of glmnet functions
  y <- houses2$price
  x <- model.matrix(price ~ ., data = houses2)[,-1]

  ### Ridge
  # Create a vector containg all lambdas to estimate the ridge model
  grid_r <- exp(seq(0, -10, length = 100))

  # Estimating a ridge regression for the desired grid of lambdas
  ridge.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 0, lambda = grid_r)

  # Find the best lambda out of the grid for the ridge regression by cross validation
  set.seed(i*77)
  cv.out.ridge <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 0, lambda = grid_r)
  ridge.best.lambda <- cv.out.ridge$lambda.min

  ## Assess the performance of the 'best' ridge regression model
  # Predict prices for the created test set
  predictions.r <- predict(object = ridge.mod, s = ridge.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.r[predictions.r < 0] <- 0

  # Calculate the MAE
  Model.3[i,1] <- mae(y[test_set], predictions.r)


  ### Lasso
  # Create a vector containg all lambdas to estimate the lasso model
  grid_l <- exp(seq(8, 6, length = 100))

  # Estimating a lasso regression for the desired grid of lambdas
  lasso.mod <- glmnet(x = x[training_set,], y = y[training_set],
```

```
                      alpha = 1, lambda = grid_l)

  # Find the best lambda out of the grid for the lasso regression by cross validation
  set.seed(i*77)
  cv.out.lasso <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 1, lambda = grid_l)
  lasso.best.lambda <- cv.out.lasso$lambda.min

  ## Assess the performance of the 'best' lasso regression model
  # Predict prices for the created test set
  predictions.l <- predict(object = lasso.mod, s = lasso.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.l[predictions.l < 0] <- 0

  # Calculate the MAE
  Model.3[i,2] <- mae(y[test_set], predictions.l)

  # Calculate the average MAE
  Model.3[6,1] <- mean(Model.3[c(1:5),1], na.rm = T)
  Model.3[6,2] <- mean(Model.3[c(1:5),2], na.rm = T)
}

# Calculate the SD of the MAE
Model.3 <- rbind(Model.3, c(sd(Model.3[c(1:5),1]),sd(Model.3[c(1:5),2])))
rownames(Model.3) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average","SD")
```

We then take a look at the results.

```
Model.3
```

```
        MAE ridge MAE lasso
1. Run   217278.2  215418.9
2. Run   219964.0  217741.0
3. Run   224466.9  220955.8
4. Run   232681.3  229333.8
5. Run   222289.6  219921.8
Average  223336.0  220674.2
SD         5868.9    5288.1
```

We observe that in this model the Ridge Regression leads to an average MAE of 223336 and a SD of 5868.9, while the Lasso Regression leads to an average MAE of 220674.2 and a SD of 5288.1. Hence, the Lasso Regression seems to perform better for this model.

Model 4

In the fourth model we run a Ridge and a Lasso Regression using both, municipality and zipcode, as local authority indicating variables. We start by preparing and reshaping the data, such that only the desired variables are included. Hence, we drop the variables date_available and kategorie. Next, we set up a loop to evaluate both models (Ridge and Lasso Regression model) five times. This loop is set up analogously to the one of the previous model, since the estimation procedure remains exactly the same as before.

```
#### Model 4
# Create a Matrix to store the different MAEs of the models from each run
Model.4 <- matrix(NA, 6, 2)
colnames(Model.4) <- c("MAE ridge","MAE lasso")
rownames(Model.4) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average")

# Reshape the data
houses2 <- houses[-c(11,39)]

for (i in c(1:5)){
  # Split the training dataset into a simulated training- and test set
  set.seed(i*7)
  training_set <- sample(x = c(TRUE,FALSE), size = nrow(houses2), rep = TRUE, prob = c(0.9,0.1))
  test_set <- (!training_set)

  # Transform the data to enable the usage of glmnet functions
  y <- houses2$price
  x <- model.matrix(price ~ ., data = houses2)[,-1]

  ### Ridge
  # Create a vector containg all lambdas to estimate the ridge model
  grid_r <- exp(seq(-5, -20, length = 100))

  # Estimating a ridge regression for the desired grid of lambdas
  ridge.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 0, lambda = grid_r)

  # Find the best lambda out of the grid for the ridge regression by cross validation
  set.seed(i*77)
  cv.out.ridge <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 0, lambda = grid_r)
  ridge.best.lambda <- cv.out.ridge$lambda.min

  ## Assess the performance of the 'best' ridge regression model
  # Predict prices for the created test set
  predictions.r <- predict(object = ridge.mod, s = ridge.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.r[predictions.r < 0] <- 0

  # Calculate the MAE
  Model.4[i,1] <- mae(y[test_set], predictions.r)


  ### Lasso
  # Create a vector containg all lambdas to estimate the lasso model
  grid_l <- exp(seq(8, 6, length = 100))

  # Estimating a lasso regression for the desired grid of lambdas
  lasso.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 1, lambda = grid_l)
```

```
  # Find the best lambda out of the grid for the lasso regression by cross validation
  set.seed(i*77)
  cv.out.lasso <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 1, lambda = grid_l)
  lasso.best.lambda <- cv.out.lasso$lambda.min

  ## Assess the performance of the 'best' lasso regression model
  # Predict prices for the created test set
  predictions.l <- predict(object = lasso.mod, s = lasso.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.l[predictions.l < 0] <- 0

  # Calculate the MAE
  Model.4[i,2] <- mae(y[test_set], predictions.l)

  # Calculate the average MAE
  Model.4[6,1] <- mean(Model.4[c(1:5),1], na.rm = T)
  Model.4[6,2] <- mean(Model.4[c(1:5),2], na.rm = T)
}

# Calculate the SD of the MAE
Model.4 <- rbind(Model.4, c(sd(Model.4[c(1:5),1]),sd(Model.4[c(1:5),2])))
rownames(Model.4) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average","SD")
```

We then take a look at the results.

```
Model.4
```

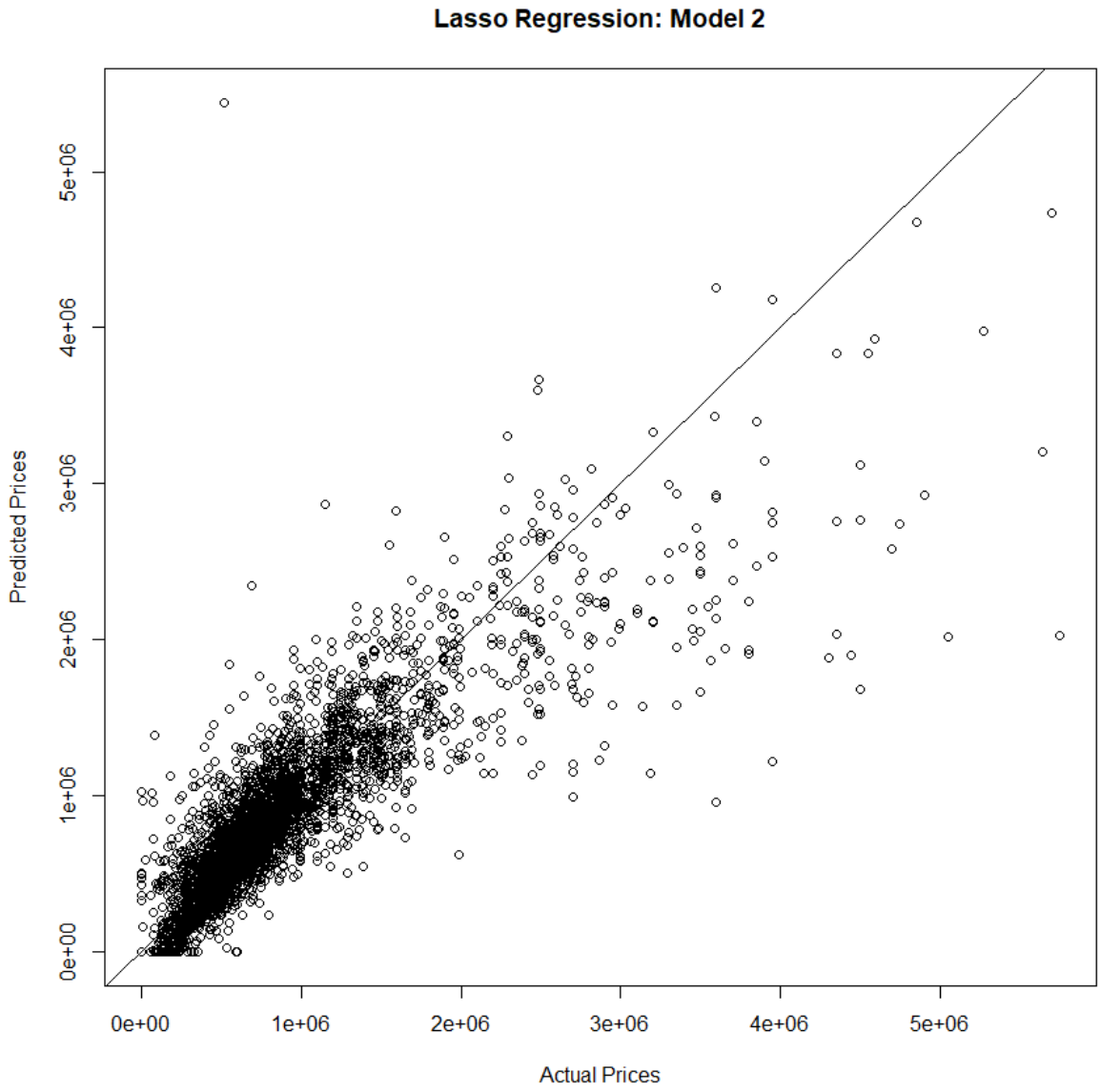
```
        MAE ridge MAE lasso
1. Run   211214.5  207646.7
2. Run   214239.2  211219.6
3. Run   218759.6  213876.7
4. Run   223563.1  219547.8
5. Run   215077.5  212283.0
Average  216570.8  212914.8
SD         4744.4    4357.4
```

We observe that in this model the Ridge Regression leads to an average MAE of 216570.8 and a SD of 4744.4, while the Lasso Regression leads to an average MAE of 212914.8 and a SD of 4357.4. Hence, the Lasso Regression seems to perform better for this model.

Conclusions from Models 1-4

We observe that in each of these models the Lasso Regression performs better than the Ridge Regression. Furthermore, we see that the Lasso Regression of Model 4 leads to the smallest average MAE, while the Lasso Regression of Model 2 appears to be the most robust model (i.e. the model with the lowest SD from the resulting MAEs). However, performance and robustness of the Lasso Regressions from Models 2 and 4 appear to be on a very similar level, but Model 4 requires much more computational power than Model 2, as it includes both, municipalities and zipcodes. Hence, the Lasso Regression of Model 2 is our preferred model and we decide to use this one as a basis for further adjustments in the subsequent models. Furthermore, we illustrate the predictions of this model by means of a plot with the predicted prices on the y-axis and the actual prices on the x-axis. In addition, we add a 45° degree line to indicate the perfect fit.

```
# Plot the results of the lasso regression from model 2
plot(y[test_set], predictions.l, main = "Lasso Regression: Model 2",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
```



In the next steps, we want to assess whether the correlation between the variables area and area_useable as well as the one between area and area_times_municipality_avg_m2_price have a negative effect on the performance and robustness of the models.

Model 5

In the fifth model we run the same Ridge and Lasso Regression as in Model 2, except that we additionally exclude the area_times_municipality_avg_m2_price variable. We start by preparing and reshaping the data, such that only the desired variables are included. Hence, we drop the variables date_available, municipality, kategorie and area_times_municipality_avg_m2_price. Next, we set up a loop to evaluate both models (Ridge and Lasso Regression model) five times. This loop is set up analogously to the one of Model 2, since the estimation procedure is exactly the same for this model.

```
#### Model 5
# Create a Matrix to store the different MAEs of the models from each run
Model.5 <- matrix(NA, 6, 2)
colnames(Model.5) <- c("MAE ridge","MAE lasso")
rownames(Model.5) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average")

# Reshape the data
houses2 <- houses[-c(11,20,37,41)]

for (i in c(1:5)){
  # Split the training dataset into a simulated training- and test set
  set.seed(i*7)
  training_set <- sample(x = c(TRUE,FALSE), size = nrow(houses2), rep = TRUE, prob = c(0.9,0.1))
  test_set <- (!training_set)

  # Transform the data to enable the usage of glmnet functions
  y <- houses2$price
  x <- model.matrix(price ~ ., data = houses2)[,-1]

  ### Ridge
  # Create a vector containg all lambdas to estimate the ridge model
  grid_r <- exp(seq(13, 8, length = 100))

  # Estimating a ridge regression for the desired grid of lambdas
  ridge.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 0, lambda = grid_r)

  # Find the best lambda out of the grid for the ridge regression by cross validation
  set.seed(i*77)
  cv.out.ridge <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 0, lambda = grid_r)
  ridge.best.lambda <- cv.out.ridge$lambda.min

  ## Assess the performance of the 'best' ridge regression model
  # Predict prices for the created test set
  predictions.r <- predict(object = ridge.mod, s = ridge.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.r[predictions.r < 0] <- 0

  # Calculate the MAE
  Model.5[i,1] <- mae(y[test_set], predictions.r)


  ### Lasso
  # Create a vector containg all lambdas to estimate the lasso model
```

```
  grid_l <- exp(seq(8, 6, length = 100))

  # Estimating a lasso regression for the desired grid of lambdas
  lasso.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 1, lambda = grid_l)

  # Find the best lambda out of the grid for the lasso regression by cross validation
  set.seed(i*77)
  cv.out.lasso <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 1, lambda = grid_l)
  lasso.best.lambda <- cv.out.lasso$lambda.min

  ## Assess the performance of the 'best' lasso regression model
  # Predict prices for the created test set
  predictions.l <- predict(object = lasso.mod, s = lasso.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.l[predictions.l < 0] <- 0

  # Calculate the MAE
  Model.5[i,2] <- mae(y[test_set], predictions.l)

  # Calculate the average MAE
  Model.5[6,1] <- mean(Model.5[c(1:5),1], na.rm = T)
  Model.5[6,2] <- mean(Model.5[c(1:5),2], na.rm = T)
}

# Calculate the SD of the MAE
Model.5 <- rbind(Model.5, c(sd(Model.5[c(1:5),1]),sd(Model.5[c(1:5),2])))
rownames(Model.5) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average","SD")
```

We then take a look at the results.

```
Model.5
```

```
        MAE ridge MAE lasso
1. Run   253265.2  253047.3
2. Run   257378.1  256714.3
3. Run   257691.9  257031.8
4. Run   270327.5  269879.8
5. Run   258934.4  258606.3
Average  259519.4  259055.9
SD         6407.0    6384.4
```

We observe that in this model the Ridge Regression leads to an average MAE of 259519.4 and a SD of 6407, while the Lasso Regression leads to an average MAE of 259055.9 and a SD of 6384.4. Hence, excluding the area_times_municipality_avg_m2_price variable leads to a weaker model.

Model 6

In the sixth model we run the same Ridge and Lasso Regression as in Model 2, except that we additionally exclude the area_useable variable. We start by preparing and reshaping the data, such that only the desired variables are included. Hence, we drop the variables date_available, municipality, kategorie and area_useable. Next, we set up a loop to evaluate both models (Ridge and Lasso Regression model) five times. This loop is set up analogously to the one of the previous model, since the estimation procedure remains exactly the same as before.

```
#### Model 6
# Create a Matrix to store the different MAEs of the models from each run
Model.6 <- matrix(NA, 6, 2)
colnames(Model.6) <- c("MAE ridge","MAE lasso")
rownames(Model.6) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average")

# Reshape the data
houses2 <- houses[-c(2,11,20,37)]

for (i in c(1:5)){
  # Split the training dataset into a simulated training- and test set
  set.seed(i*7)
  training_set <- sample(x = c(TRUE,FALSE), size = nrow(houses2), rep = TRUE, prob = c(0.9,0.1))
  test_set <- (!training_set)

  # Transform the data to enable the usage of glmnet functions
  y <- houses2$price
  x <- model.matrix(price ~ ., data = houses2)[,-1]

  ### Ridge
  # Create a vector containg all lambdas to estimate the ridge model
  grid_r <- exp(seq(0, -15, length = 100))

  # Estimating a ridge regression for the desired grid of lambdas
  ridge.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 0, lambda = grid_r)

  # Find the best lambda out of the grid for the ridge regression by cross validation
  set.seed(i*77)
  cv.out.ridge <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 0, lambda = grid_r)
  ridge.best.lambda <- cv.out.ridge$lambda.min

  ## Assess the performance of the 'best' ridge regression model
  # Predict prices for the created test set
  predictions.r <- predict(object = ridge.mod, s = ridge.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.r[predictions.r < 0] <- 0

  # Calculate the MAE
  Model.6[i,1] <- mae(y[test_set], predictions.r)


  ### Lasso
  # Create a vector containg all lambdas to estimate the lasso model
  grid_l <- exp(seq(8, 6, length = 100))

  # Estimating a lasso regression for the desired grid of lambdas
  lasso.mod <- glmnet(x = x[training_set,], y = y[training_set],
```

```
                      alpha = 1, lambda = grid_l)

  # Find the best lambda out of the grid for the lasso regression by cross validation
  set.seed(i*77)
  cv.out.lasso <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 1, lambda = grid_l)
  lasso.best.lambda <- cv.out.lasso$lambda.min

  ## Assess the performance of the 'best' lasso regression model
  # Predict prices for the created test set
  predictions.l <- predict(object = lasso.mod, s = lasso.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.l[predictions.l < 0] <- 0

  # Calculate the MAE
  Model.6[i,2] <- mae(y[test_set], predictions.l)

  # Calculate the average MAE
  Model.6[6,1] <- mean(Model.6[c(1:5),1], na.rm = T)
  Model.6[6,2] <- mean(Model.6[c(1:5),2], na.rm = T)
}

# Calculate the SD of the MAE
Model.6 <- rbind(Model.6, c(sd(Model.6[c(1:5),1]),sd(Model.6[c(1:5),2])))
rownames(Model.6) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average","SD")
```

We then take a look at the results.

```
Model.6
```

```
        MAE ridge MAE lasso
1. Run   237640.9  236592.3
2. Run   243612.5  242191.0
3. Run   242688.4  241059.5
4. Run   256706.4  255614.6
5. Run   243133.3  242176.2
Average  244756.3  243526.7
SD         7100.2    7139.6
```

We observe that in this model the Ridge Regression leads to an average MAE of 244756.3 and a SD of 7100.2, while the Lasso Regression leads to an average MAE of 243526.7 and a SD of 7139.6. Hence, excluding the area_useable variable also leads to a weaker model.

Conclusions from Models 5&6

We observe that excluding one of the variables which potentially cause multicollinearity problems with the variable area (i.e. area_times_municipality_avg_m2_price and area_useable) increases the average MAEs and SDs for each estimated model. Hence, the Lasso Regression of Model 2 (i.e. the model which keeps area_usable and area_times_municipality_avg_m2_price and relies on zipcode as a geographical indicator) remains our preferred model thus far and we decide to still use this one as a basis for further attempts to enhance our model.

In the next steps, we want to assess whether observations with more than 30% NAs or outliers have a significantly negative effect on the performance and robustness of our models. The observations we treat as outliers are the following:

- Observations with area ≥ 400
- Observations with area_useable ≥ 1000
- Observations with rooms ≥ 12

Model 7

In the seventh model we run the same Ridge and Lasso Regression as in Model 2, except that we additionally remove outlier observations. We start by preparing and reshaping the data, such that only the desired variables and observations are included. Hence, we drop the variables date_available, municipality and kategorie and we get rid of outliers. Next, we set up a loop to evaluate both models (Ridge and Lasso Regression model) five times. This loop is set up analogously to the one of Model 2, since the estimation procedure is exactly the same for this model.

```
#### Model 7
# Create a Matrix to store the different MAEs of the models from each run
Model.7 <- matrix(NA, 6, 2)
colnames(Model.7) <- c("MAE ridge","MAE lasso")
rownames(Model.7) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average")

# Reshape the data
houses2 <- houses[-c(11,20,39)]

# Get rid of outliers
houses2 <- houses2[houses2$area < 400, ]
houses2 <- houses2[houses2$area_useable < 1000, ]
houses2 <- houses2[houses2$rooms < 12, ]

for (i in c(1:5)){
  # Split the training dataset into a simulated training- and test set
  set.seed(i*7)
  training_set <- sample(x = c(TRUE,FALSE), size = nrow(houses2), rep = TRUE, prob = c(0.9,0.1))
  test_set <- (!training_set)

  # Transform the data to enable the usage of glmnet functions
  y <- houses2$price
  x <- model.matrix(price ~ ., data = houses2)[,-1]

  ### Ridge
  # Create a vector containg all lambdas to estimate the ridge model
  grid_r <- exp(seq(0, -15, length = 100))

  # Estimating a ridge regression for the desired grid of lambdas
  ridge.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 0, lambda = grid_r)

  # Find the best lambda out of the grid for the ridge regression by cross validation
  set.seed(i*77)
  cv.out.ridge <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 0, lambda = grid_r)
  ridge.best.lambda <- cv.out.ridge$lambda.min

  ## Assess the performance of the 'best' ridge regression model
  # Predict prices for the created test set
  predictions.r <- predict(object = ridge.mod, s = ridge.best.lambda, newx =  x[test_set,])
```

```
  # Substitute negative price predictions with the value of 0.
  predictions.r[predictions.r < 0] <- 0

  # Calculate the MAE
  Model.7[i,1] <- mae(y[test_set], predictions.r)


  ### Lasso
  # Create a vector containg all lambdas to estimate the lasso model
  grid_l <- exp(seq(8, 6, length = 100))

  # Estimating a lasso regression for the desired grid of lambdas
  lasso.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 1, lambda = grid_l)

  # Find the best lambda out of the grid for the lasso regression by cross validation
  set.seed(i*77)
  cv.out.lasso <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 1, lambda = grid_l)
  lasso.best.lambda <- cv.out.lasso$lambda.min

  ## Assess the performance of the 'best' lasso regression model
  # Predict prices for the created test set
  predictions.l <- predict(object = lasso.mod, s = lasso.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.l[predictions.l < 0] <- 0

  # Calculate the MAE
  Model.7[i,2] <- mae(y[test_set], predictions.l)

  # Calculate the average MAE
  Model.7[6,1] <- mean(Model.7[c(1:5),1], na.rm = T)
  Model.7[6,2] <- mean(Model.7[c(1:5),2], na.rm = T)
}

# Calculate the SD of the MAE
Model.7 <- rbind(Model.7, c(sd(Model.7[c(1:5),1]),sd(Model.7[c(1:5),2])))
rownames(Model.7) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average","SD")
```

We then take a look at the results.

```
Model.7
```

```
        MAE ridge MAE lasso
1. Run   214668.6  211746.2
2. Run   215295.8  211648.5
3. Run   213866.0  211051.0
4. Run   216817.7  213729.6
5. Run   221410.0  218703.2
Average  216411.6  213375.7
SD         2996.3    3144.5
```

We observe that in this model the Ridge Regression leads to an average MAE of 216411.6 and a SD of 2996.3, while the Lasso Regression leads to an average MAE of 213375.7 and a SD of 3144.5. Hence, the Lasso Regression seems to perform better for this model.

Model 8

In the eighth model we run the same Ridge and Lasso Regression as in Model 2, except that we additionally remove observations with more than 30% NAs. We start by preparing and reshaping the data, such that only the desired variables and observations are included. Hence, we drop the variables date_available, municipality and kategorie and we get rid of observations with more than 30% NAs. Next, we set up a loop to evaluate both models (Ridge and Lasso Regression model) five times. This loop is set up analogously to the one of the previous model, since the estimation procedure remains exactly the same as before.

```
#### Model 8
# Create a Matrix to store the different MAEs of the models from each run
Model.8 <- matrix(NA, 6, 2)
colnames(Model.8) <- c("MAE ridge","MAE lasso")
rownames(Model.8) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average")

# Get the data
houses2 <- houses

# Get rid of rows with more than 30% NAs
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0

# Reshape the data
houses2 <- houses2[-c(11,20,39)]

for (i in c(1:5)){
  # Split the training dataset into a simulated training- and test set
  set.seed(i*7)
  training_set <- sample(x = c(TRUE,FALSE), size = nrow(houses2), rep = TRUE, prob = c(0.9,0.1))
  test_set <- (!training_set)

  # Transform the data to enable the usage of glmnet functions
  y <- houses2$price
  x <- model.matrix(price ~ ., data = houses2)[,-1]

  ### Ridge
  # Create a vector containg all lambdas to estimate the ridge model
  grid_r <- exp(seq(-10, -30, length = 100))

  # Estimating a ridge regression for the desired grid of lambdas
  ridge.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 0, lambda = grid_r)

  # Find the best lambda out of the grid for the ridge regression by cross validation
  set.seed(i*77)
  cv.out.ridge <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 0, lambda = grid_r)
  ridge.best.lambda <- cv.out.ridge$lambda.min

  ## Assess the performance of the 'best' ridge regression model
  # Predict prices for the created test set
  predictions.r <- predict(object = ridge.mod, s = ridge.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.r[predictions.r < 0] <- 0

  # Calculate the MAE
```

```
  Model.8[i,1] <- mae(y[test_set], predictions.r)


  ### Lasso
  # Create a vector containg all lambdas to estimate the lasso model
  grid_l <- exp(seq(8, 6, length = 100))

  # Estimating a lasso regression for the desired grid of lambdas
  lasso.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 1, lambda = grid_l)

  # Find the best lambda out of the grid for the lasso regression by cross validation
  set.seed(i*77)
  cv.out.lasso <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 1, lambda = grid_l)
  lasso.best.lambda <- cv.out.lasso$lambda.min

  ## Assess the performance of the 'best' lasso regression model
  # Predict prices for the created test set
  predictions.l <- predict(object = lasso.mod, s = lasso.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.l[predictions.l < 0] <- 0

  # Calculate the MAE
  Model.8[i,2] <- mae(y[test_set], predictions.l)

  # Calculate the average MAE
  Model.8[6,1] <- mean(Model.8[c(1:5),1], na.rm = T)
  Model.8[6,2] <- mean(Model.8[c(1:5),2], na.rm = T)
}

# Calculate the SD of the MAE
Model.8 <- rbind(Model.8, c(sd(Model.8[c(1:5),1]),sd(Model.8[c(1:5),2])))
rownames(Model.8) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average","SD")
```

We then take a look at the results.

```
Model.8
```

```
        MAE ridge MAE lasso
1. Run   219246.4  216452.4
2. Run   222388.9  219068.4
3. Run   220789.7  216698.6
4. Run   216179.0  213841.9
5. Run   220050.5  217403.3
Average  219730.9  216692.9
SD         2298.9    1893.0
```

We observe that in this model the Ridge Regression leads to an average MAE of 219730.9 and a SD of 2298.9, while the Lasso Regression leads to an average MAE of 216692.9 and a SD of 1893. Hence, the Lasso Regression seems to perform better for this model.

Model 9

In the ninth model we run the same Ridge and Lasso Regression as in the previous model, except that we additionally remove outlier observations. We start by preparing and reshaping the data, such that only the desired variables and observations are included. Hence, we drop the variables date_available, municipality and kategorie and we get rid of outliers as well as of observations with more than 30% NAs. Next, we set up a loop to evaluate both models (Ridge and Lasso Regression model) five times. This loop is set up analogously to the one of the previous model, since the estimation procedure remains exactly the same as before.

```
#### Model 9
# Create a Matrix to store the different MAEs of the models from each run
Model.9 <- matrix(NA, 6, 2)
colnames(Model.9) <- c("MAE ridge","MAE lasso")
rownames(Model.9) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average")

# Get the data
houses2 <- houses

# Get rid of rows with more than 30% NAs
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0

# Reshape the data
houses2 <- houses2[-c(11,20,39)]

# Get rid of outliers
houses2 <- houses2[houses2$area < 400, ]
houses2 <- houses2[houses2$area_useable < 1000, ]
houses2 <- houses2[houses2$rooms < 12, ]

for (i in c(1:5)){
  # Split the training dataset into a simulated training- and test set
  set.seed(i*7)
  training_set <- sample(x = c(TRUE,FALSE), size = nrow(houses2), rep = TRUE, prob = c(0.9,0.1))
  test_set <- (!training_set)

  # Transform the data to enable the usage of glmnet functions
  y <- houses2$price
  x <- model.matrix(price ~ ., data = houses2)[,-1]

  ### Ridge
  # Create a vector containg all lambdas to estimate the ridge model
  grid_r <- exp(seq(0, -15, length = 100))

  # Estimating a ridge regression for the desired grid of lambdas
  ridge.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 0, lambda = grid_r)

  # Find the best lambda out of the grid for the ridge regression by cross validation
  set.seed(i*77)
  cv.out.ridge <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 0, lambda = grid_r)
  ridge.best.lambda <- cv.out.ridge$lambda.min

  ## Assess the performance of the 'best' ridge regression model
  # Predict prices for the created test set
  predictions.r <- predict(object = ridge.mod, s = ridge.best.lambda, newx =  x[test_set,])
```

```
  # Substitute negative price predictions with the value of 0.
  predictions.r[predictions.r < 0] <- 0

  # Calculate the MAE
  Model.9[i,1] <- mae(y[test_set], predictions.r)


  ### Lasso
  # Create a vector containg all lambdas to estimate the lasso model
  grid_l <- exp(seq(8, 6, length = 100))

  # Estimating a lasso regression for the desired grid of lambdas
  lasso.mod <- glmnet(x = x[training_set,], y = y[training_set],
                      alpha = 1, lambda = grid_l)

  # Find the best lambda out of the grid for the lasso regression by cross validation
  set.seed(i*77)
  cv.out.lasso <- cv.glmnet(x = x[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 1, lambda = grid_l)
  lasso.best.lambda <- cv.out.lasso$lambda.min

  ## Assess the performance of the 'best' lasso regression model
  # Predict prices for the created test set
  predictions.l <- predict(object = lasso.mod, s = lasso.best.lambda, newx =  x[test_set,])

  # Substitute negative price predictions with the value of 0.
  predictions.l[predictions.l < 0] <- 0

  # Calculate the MAE
  Model.9[i,2] <- mae(y[test_set], predictions.l)

  # Calculate the average MAE
  Model.9[6,1] <- mean(Model.9[c(1:5),1], na.rm = T)
  Model.9[6,2] <- mean(Model.9[c(1:5),2], na.rm = T)
}

# Calculate the SD of the MAE
Model.9 <- rbind(Model.9, c(sd(Model.9[c(1:5),1]),sd(Model.9[c(1:5),2])))
rownames(Model.9) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average","SD")
```

We then take a look at the results.

```
Model.9
```

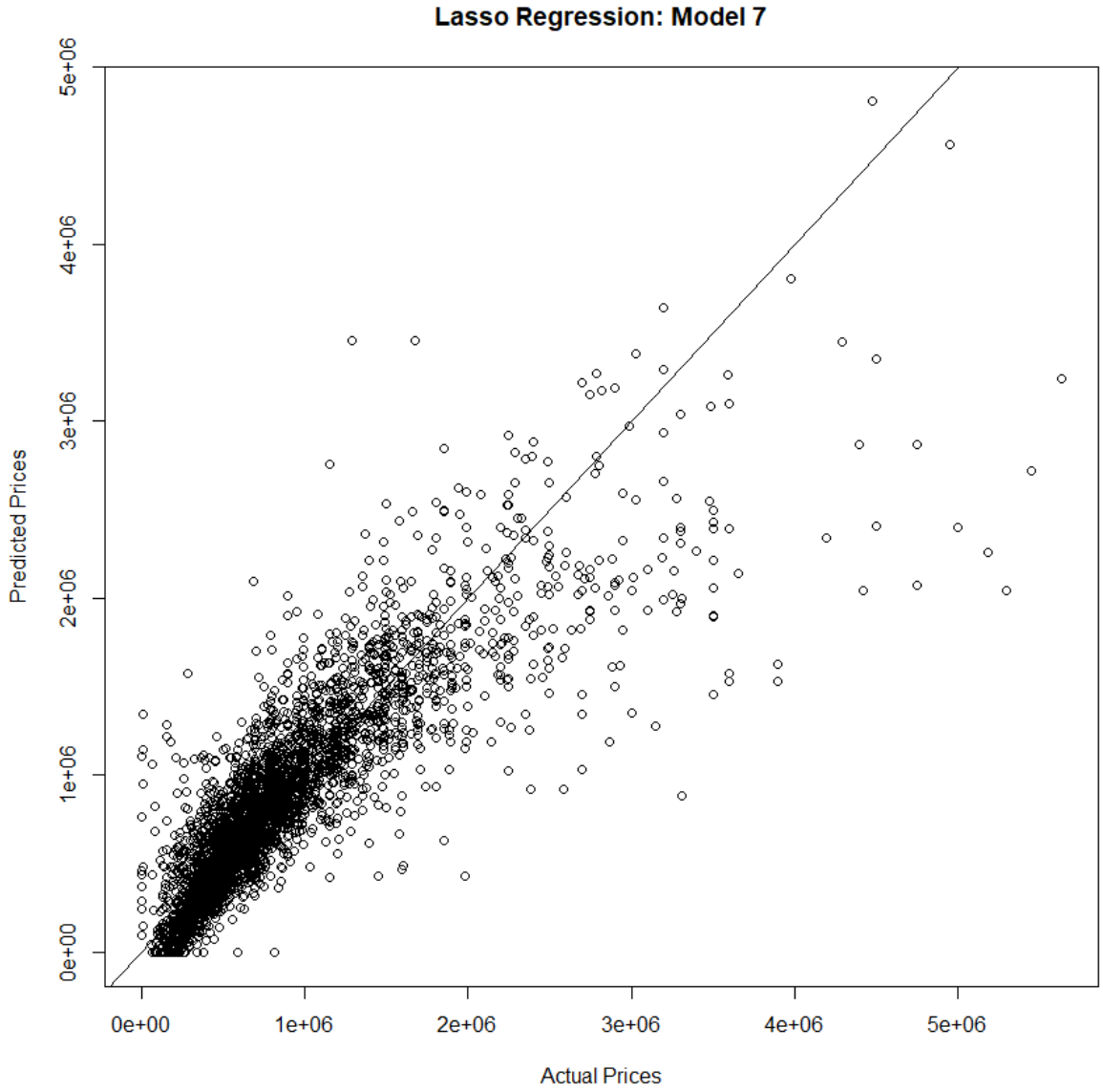
```
        MAE ridge MAE lasso
1. Run   226836.7  223585.2
2. Run   213199.5  209957.6
3. Run   219036.9  215874.0
4. Run   222080.2  219475.4
5. Run   219926.5  216831.9
Average  220216.0  217144.8
SD         4950.1    5007.0
```

We observe that in this model the Ridge Regression leads to an average MAE of 220216 and a SD of 4950.1, while the Lasso Regression leads to an average MAE of 217144.8 and a SD of 5007. Hence, the Lasso Regression seems to perform better for this model.

Conclusions from Models 7-9

We observe that in each of these models the Lasso Regression performs better than the Ridge Regression. Furthermore, we see that the Lasso Regression of Model 7 leads to the smallest average MAE, while the Lasso Regression of Model 8 appears to be the most robust model. However, in our opinion the performance gain from Model 8 to Model 7 weights heavier than the loss in robustness. Hence, the Lasso Regression of Model 7 (i.e. the model which removes outlier observations) is our preferred model thus far and we decide to use this one as a basis for further attempts to enhance our model. Furthermore, we illustrate the predictions of this model by means of a plot with the predicted prices on the y-axis and the actual prices on the x-axis. In addition, we add a 45° degree line to indicate the perfect fit.

```
# Plot the results of the lasso regression from model 7
plot(y[test_set], predictions.l, main = "Lasso Regression: Model 7",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
```



In the next and last step, we want to assess whether the factorial variable zipcode causes significant bias, due to zipcodes which are exclusively contained in the simulated test set, but not in the corresponding traing set (which is used to estimate the models).

Model 10

In the tenth model we partition the simulated test set into two subsets, such that only the second subset contains all the zipcodes, which are exclusively contained in the test set, but not in the training set. We then estimate two different models on the training set. The first model, which we refer to as the "big" model, is just the same model as model 7. The second model, which we refer to as the "small" model, is almost the same model, except that we estimate it without the zipcode variable. We then apply the big model to the first subset of the test set and the small model on the second subset. We do this because we suspect that zipcodes which are contained in the simulated test set but not in the corresponding training set (which is used to estimate the model) are a potential source of bias. Furthermore, we think that the same procedure is not necessary for other factorial variables, since none of the other included ones has a similarly high number of categories as the zipcode variable.

We start by preparing and reshaping the data, such that only the desired variables and observations are included. Hence, we drop the variables date_available, municipality and kategorie and we get rid of outliers. Next, we set up a loop to evaluate both models (Ridge and Lasso Regression model) five times. This loop looks similar to the one of Model 7. There are only 2 crucial differences: (1) we estimate a big and a small model using the simulated training set for both, the Ridge and the Lasso Regression model and (2) we split the test set into the two desired subsets before we calculate the predictions. We then use the predictions of the big models for the first subset and the predictions of the small models for the second subset. In the end we calculate the averages and the standard deviations of the resulting MAEs analogously to the previous models.

```
#### Model 10
# Create a Matrix to store the different MAEs of the models from each run
Model.10 <- matrix(NA, 6, 2)
colnames(Model.10) <- c("MAE ridge","MAE lasso")
rownames(Model.10) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average")

# Reshape the data
houses2 <- houses[-c(11,20,39)]

# Get rid of outliers
houses2 <- houses2[houses2$area < 400, ]
houses2 <- houses2[houses2$area_useable < 1000, ]
houses2 <- houses2[houses2$rooms < 12, ]

# Remove zipcode for the small model
houses3 <- houses2[,-35]

for (i in c(1:5)){
  # Split the training dataset into a simulated training- and test set
  set.seed(i*7)
  training_set <- sample(x = c(TRUE,FALSE), size = nrow(houses2), rep = TRUE, prob = c(0.9,0.1))
  test_set <- (!training_set)

  # Split the test set into the desired subsets
  shared.zipcodes <- houses2$zipcode[test_set] %in% unique(houses2$zipcode[training_set])
  test_subset1 <- test_set
  test_subset1[test_subset1 == T][!shared.zipcodes] <- F
  test_subset2 <- test_set
  test_subset2[test_subset2 == T][shared.zipcodes] <- F

  # Transform the data to enable the usage of glmnet functions
  y <- houses2$price
```

```r
x_big   <- model.matrix(price ~ ., data = houses2)[,-1]
x_small <- model.matrix(price ~ ., data = houses3)[,-1]


####### Ridge
#### Big model
# Create a vector containg all lambdas to estimate the ridge model
grid_r <- exp(seq(-15, -30, length = 100))

# Estimating a ridge regression for the desired grid of lambdas
ridge.mod.big <- glmnet(x = x_big[training_set,], y = y[training_set],
                        alpha = 0, lambda = grid_r)

# Find the best lambda out of the grid for the ridge regression by cross validation
set.seed(i*77)
cv.out.ridge <- cv.glmnet(x = x_big[training_set,], y = y[training_set],
                          nfolds = 7, alpha = 0, lambda = grid_r)
ridge.big.best.lambda <- cv.out.ridge$lambda.min


#### Small model
# Create a vector containg all lambdas to estimate the ridge model
grid_r <- exp(seq(0, -20, length = 100))

# Estimating a ridge regression for the desired grid of lambdas
ridge.mod.small <- glmnet(x = x_small[training_set,], y = y[training_set],
                          alpha = 0, lambda = grid_r)

# Find the best lambda out of the grid for the ridge regression by cross validation
set.seed(i*77)
cv.out.ridge <- cv.glmnet(x = x_small[training_set,], y = y[training_set],
                          nfolds = 7, alpha = 0, lambda = grid_r)
ridge.small.best.lambda <- cv.out.ridge$lambda.min


#### Assess the performance of the 'best' ridge regression model
# Predict prices for the created test set
predictions.r <- c(1:sum(test_set))
predictions.r.big   <- predict(object = ridge.mod.big, s = ridge.big.best.lambda,
                               newx =  x_big[test_subset1,])
predictions.r.small <- predict(object = ridge.mod.small, s = ridge.small.best.lambda,
                               newx =  x_small[test_subset2,])
predictions.r[shared.zipcodes] <- predictions.r.big
predictions.r[!shared.zipcodes] <- predictions.r.small

# Substitute negative price predictions with the value of 0.
predictions.r[predictions.r < 0] <- 0

# Calculate the MAE
Model.10[i,1] <- mae(y[test_set], predictions.r)



####### Lasso
#### Big model
# Create a vector containg all lambdas to estimate the lasso model
grid_l <- exp(seq(8, 6, length = 100))
```

```
  # Estimating a lasso regression for the whole grid of lambdas
  lasso.mod.big <- glmnet(x = x_big[training_set,], y = y[training_set],
                          alpha = 1, lambda = grid_l)

  # Find the best lambda out of the grid for the lasso regression by cross validation
  set.seed(i*77)
  cv.out.lasso <- cv.glmnet(x = x_big[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 1, lambda = grid_l)
  lasso.big.best.lambda <- cv.out.lasso$lambda.min


  #### Small model
  # Create a vector containg all lambdas to estimate the lasso model
  grid_l <- exp(seq(8, 6, length = 100))

  # Estimating a lasso regression for the whole grid of lambdas
  lasso.mod.small <- glmnet(x = x_small[training_set,], y = y[training_set],
                            alpha = 1, lambda = grid_l)

  # Find the best lambda out of the grid for the lasso regression by cross validation
  set.seed(i*77)
  cv.out.lasso <- cv.glmnet(x = x_small[training_set,], y = y[training_set],
                            nfolds = 7, alpha = 1, lambda = grid_l)
  lasso.small.best.lambda <- cv.out.lasso$lambda.min


  #### Assess the performance of the 'best' lasso regression model
  # Predict prices for the created test set
  predictions.l <- c(1:sum(test_set))
  predictions.l.big   <- predict(object = lasso.mod.big, s = lasso.big.best.lambda,
                                 newx =  x_big[test_subset1,])
  predictions.l.small <- predict(object = lasso.mod.small, s = lasso.small.best.lambda,
                                 newx =  x_small[test_subset2,])
  predictions.l[shared.zipcodes] <- predictions.l.big
  predictions.l[!shared.zipcodes] <- predictions.l.small

  # Substitute negative price predictions with the value of 0.
  predictions.l[predictions.l < 0] <- 0

  # Calculate the MAE
  Model.10[i,2] <- mae(y[test_set], predictions.l)

  # Calculate the average MAE
  Model.10[6,1] <- mean(Model.10[c(1:5),1], na.rm = T)
  Model.10[6,2] <- mean(Model.10[c(1:5),2], na.rm = T)
}

# Calculate the SD of the MAE
Model.10 <- rbind(Model.10, c(sd(Model.10[c(1:5),1]),sd(Model.10[c(1:5),2])))
rownames(Model.10) <- c("1. Run","2. Run","3. Run","4. Run","5. Run","Average","SD")
```

We then take a look at the results.

```
Model.10
```

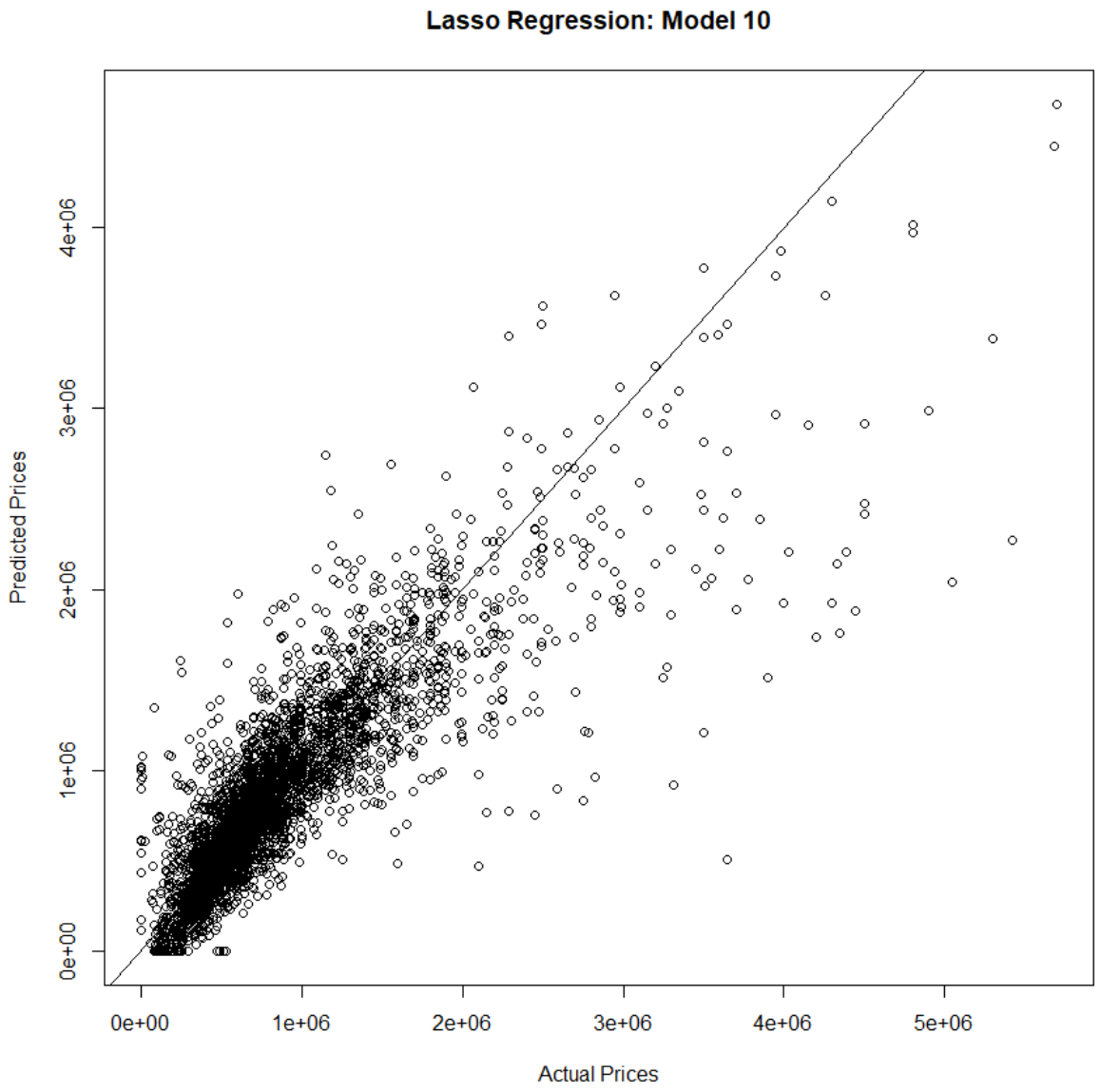
```
        MAE ridge MAE lasso
1. Run   214625.1  212029.1
2. Run   215415.6  212054.1
3. Run   213767.2  211136.4
4. Run   216664.8  213793.9
5. Run   221200.2  218866.1
Average  216334.6  213575.9
SD         2921.6    3110.2
```

We observe that in this model the Ridge Regression leads to an average MAE of 216334.6 and a SD of 2921.6, while the Lasso Regression leads to an average MAE of 213575.9 and a SD of 3110.2. Hence, the Lasso Regression seems to perform better for this model.

Conclusions Ridge & Lasso Regression

We observe that in each of the estimated models the Lasso Regression performs better than the Ridge Regression. Furthermore, we see that the Lasso Regression of Model 7 leads to a slightly smaller average MAE than Model 10, while the Lasso Regression of Model 10 appears to be the more robust model. However, in our opinion the robustness gain from Model 7 to Model 10 slightly outweighs the associated loss in performance. Hence, we conclude that the Lasso Regression of the comprehensive Model 10 is the best variant out of all Ridge and Lasso Regressions we tried. Finally, we illustrate the predictions of our final Lasso Regression model by means of a plot with the predicted prices on the y-axis and the actual prices on the x-axis. In addition, we add a 45° degree line to indicate the perfect fit.

```
# Plot the results of the lasso regression from model 10
plot(y[test_set], predictions.l, main = "Lasso Regression: Model 10",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
```


Lasso Regression: Model 10
Predicted Prices
Actual Prices
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06

Decision Tree Regression

We now test out another type of model, Decision Tree Regression models.

Model 1

In this Model 1 we test out a Decision Tree Regression.

We reshape the data. We get rid of the variable kategorie and keep the rest.

```
houses2 <- houses[-39]
```

We split the data into training and test set, with a split ratio of 90%.

```
set.seed(123)
split = sample.split(houses2$price, SplitRatio = 0.9)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
```

We create the model using the training set. We create the Decision Tree Regression Model. We set minsplit, which is the minimum number of observations required in a node to attempt a split, equal to 10. We set the number of cross-validations, xval, to 10.

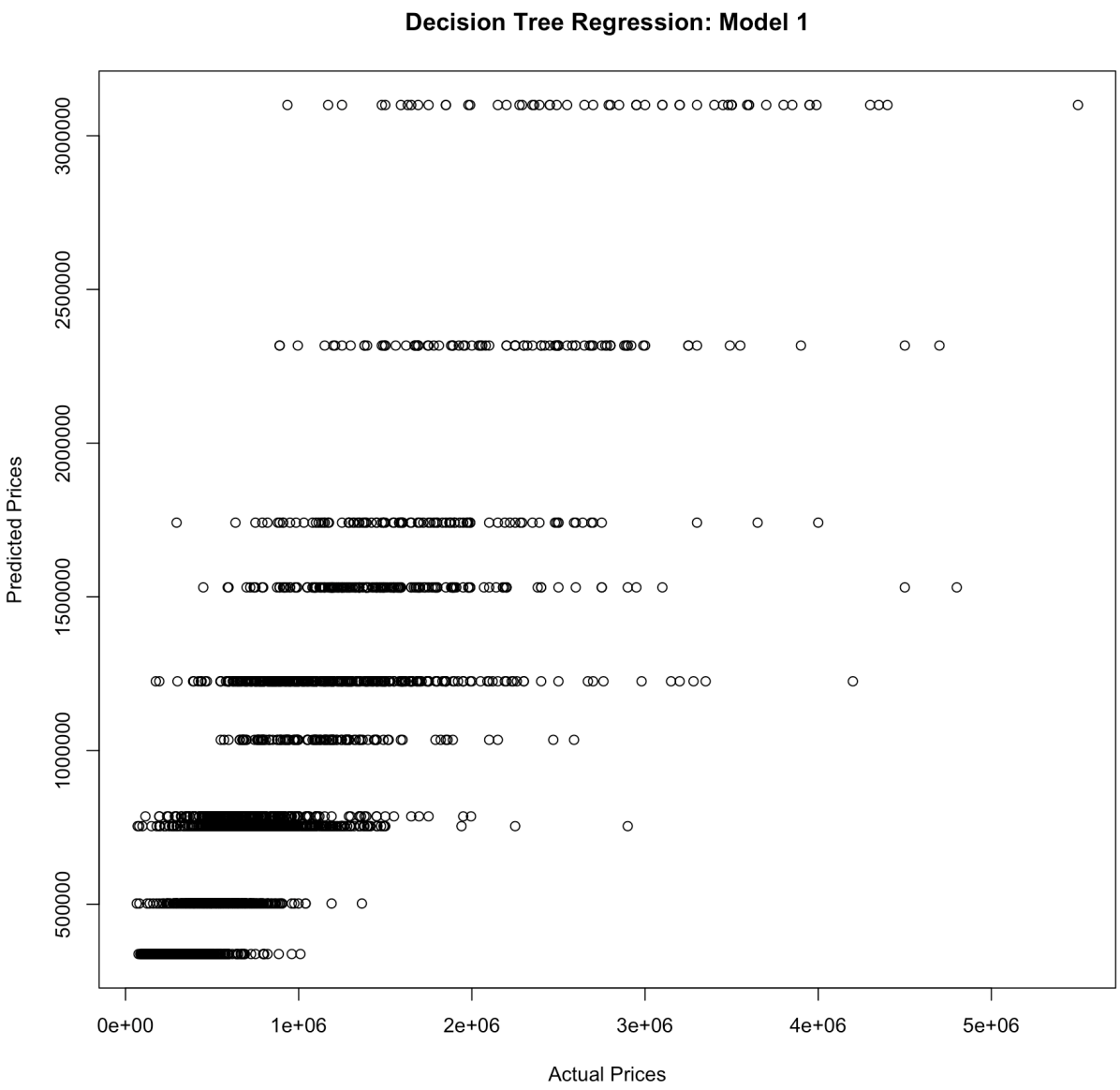
```
model = rpart(formula = price ~ .,
                  data = training_set,
                  control = rpart.control(minsplit = 10, xval = 10))
```

We apply the model to the test set.

```
predictions = predict(model, newdata = test_set)
```

We plot and evaluate the results. We get an MAE of 231'970.9

```
plot(test_set$price,predictions, main = "Decision Tree Regression: Model 1",
     xlab = "Actual Prices", ylab = "Predicted Prices")
mae(test_set$price,predictions)
#MAE = 231970.9
```


Decision Tree Regression: Model 1
Predicted Prices
Actual Prices
500000
1000000
1500000
2000000
2500000
3000000
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06

Model 2

This model is the same as Model 1, using a slightly different code. As expected, the MAE is the same as for Model 1.

We reshape the data.

```
houses2 <- houses[-39]
```

We split the data into training and test set.

```
set.seed(123)
split = sample.split(houses2$price, SplitRatio = 0.9)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
```

We create the model using the training set. This time we use the anova as the method, as this method performs best.

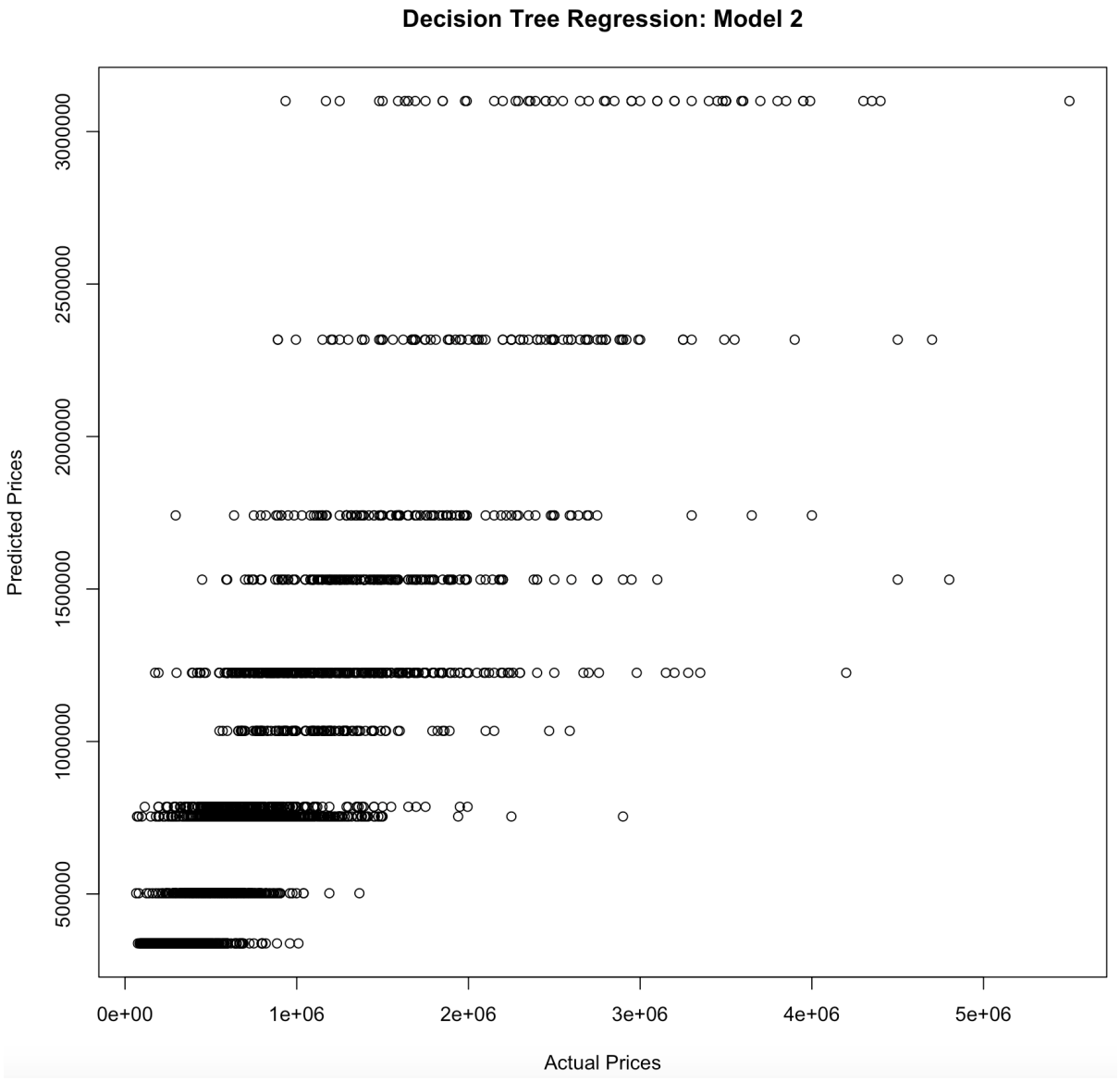
```
model = rpart(formula = price ~ .,
              data = training_set,
              method="anova")
```

We apply the model to the test set.

```
predictions = predict(model, newdata = test_set)
```

We plot and evaluate the results. We get an MAE of 231'970.9 just as before.

```
plot(test_set$price,predictions, main = "Decision Tree Regression: Model 2",
     xlab = "Actual Prices", ylab = "Predicted Prices")
mae(test_set$price,predictions)
# MAE = 231970.9
```


Decision Tree Regression: Model 2
Predicted Prices
500000
1000000
1500000
2000000
2500000
3000000
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06
Actual Prices

Model 3

We create a third model for Decision Tree Regression.

We reshape the data. We get rid of the variable kategorie and keep the rest of the variables. We get rid of rows with more than 30% NAs. We get rid of outliers.

```
houses2 <- houses[-39]
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
```

We split the data into training and test set. We set the SplitRatio equal to 0.9.

```
set.seed(123)
split = sample.split(houses2$price, SplitRatio = 0.9)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
```

We create the model using the training set. We add a minsplit of 20 and we add cp, the complexity parameter, to be equal to 0.002.

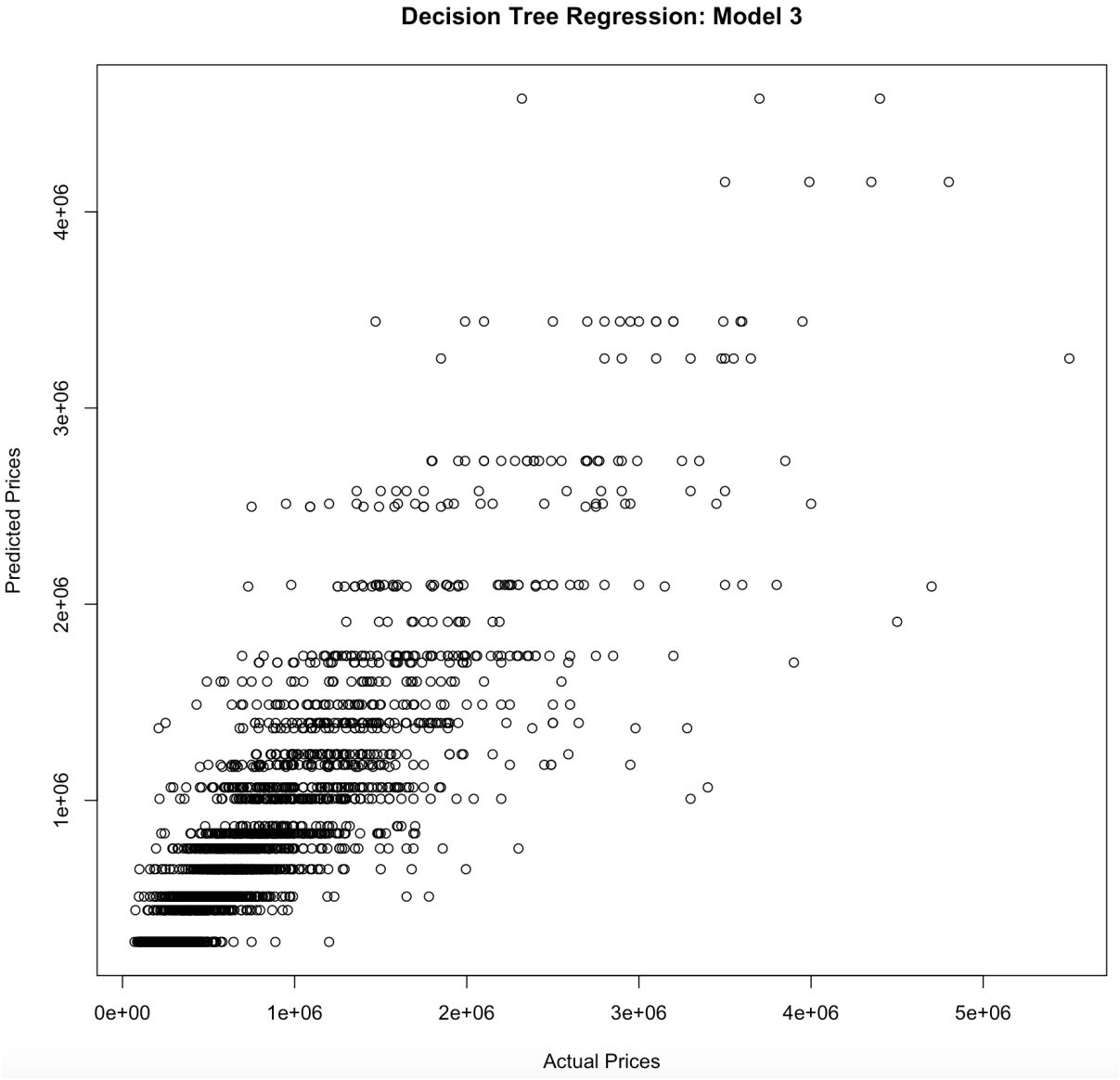
```
model = rpart(formula = price ~ .,
                  data = training_set,
                  control = rpart.control(minsplit=20, cp=0.002))
```

We apply the model to the test set.

```
predictions = predict(model, newdata = test_set)
```

We plot and evaluate the results. This new model leads us to an MAE of 214'928.8.

```
plot(test_set$price,predictions, main = "Decision Tree Regression: Model 3",
     xlab = "Actual Prices", ylab = "Predicted Prices")
mae(test_set$price,predictions)
# MAE = 214928.8
```


Decision Tree Regression: Model 3
Predicted Prices
Actual Prices
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06

Model 4

We create a fourth model for Decision Tree Regression.

We reshape the data. This time we get rid of the variable kategorie as well as rows with more than 30% NAs, however we do not remove outliers.

```
houses2 <- houses[-39]
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0
```

We split the data into training and test set. We set the SplitRatio equal to 0.9.

```
set.seed(123)
split = sample.split(houses2$price, SplitRatio = 0.9)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
```

We create the model using the training set. We add a minsplit of 20 and we add cp, the complexity parameter, to be equal to 0.002.

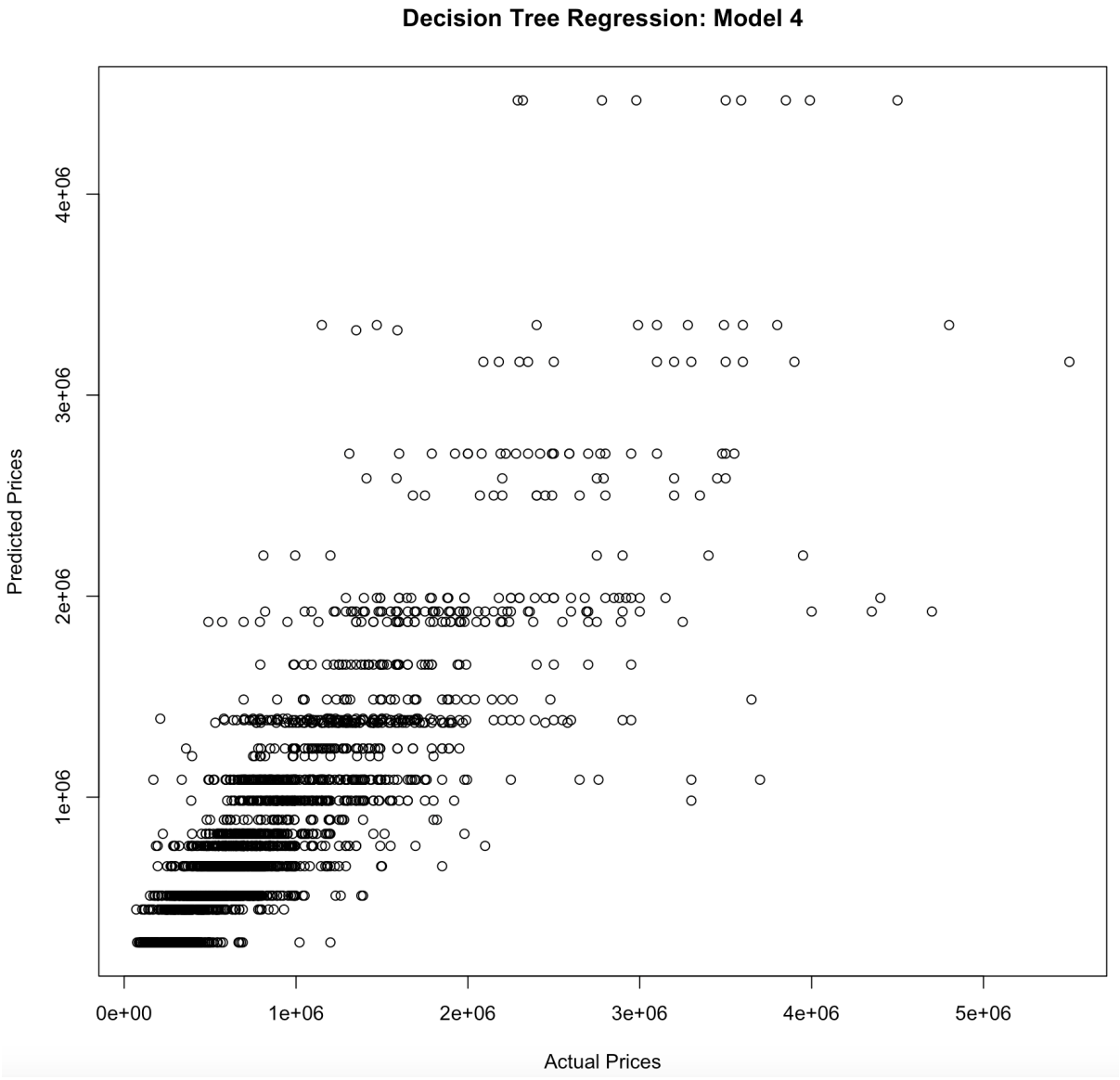
```
model = rpart(formula = price ~ .,
                  data = training_set,
                  control = rpart.control(minsplit=20, cp=0.002))
```

We apply the model to the test set.

```
predictions = predict(model, newdata = test_set)
```

We plot and evaluate the results. This new model leads us to an MAE of 213215.5, leading Model 4 to be the best variant of the Decision Tree Regression models.

```
plot(test_set$price,predictions, main = "Decision Tree Regression: Model 4",
     xlab = "Actual Prices", ylab = "Predicted Prices")
mae(test_set$price,predictions)
# MAE = 213215.5
```


Decision Tree Regression: Model 4
Predicted Prices
Actual Prices
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06

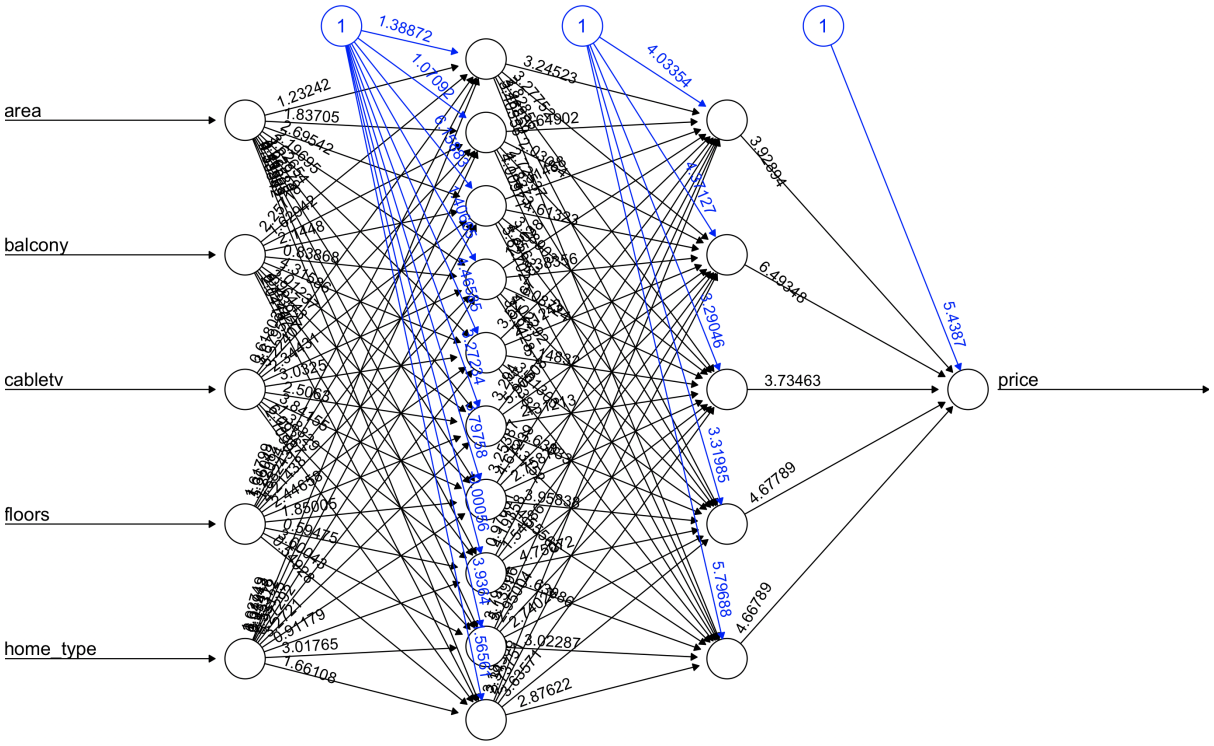
Artificial Neural Network

We now turn to more complex models, namely deep learning models, more specifically Artificial Neural Networks (ANNs). ANNs require some caution as they represent a black box in many ways.

Visualization

Before we build our first ANN, we visualize what it could look like. Because we use many variables and many neurons and layers, the visualization could get cluttered, so instead we just pick five variables as predictors and two hidden layers with 10 and 5 neurons respectively.

```
houses2 <- houses
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
str(houses2)

n <- neuralnet(price ~ area + balcony + cabletv + floors + home_type,
               data = houses2,
               hidden = c(10,5),
               linear.output = F,
               lifesign = 'full',
               rep = 1)
plot(n,
     show.weights = F,
     information = F,)
```



Model 1

We reshape the data. We create a first model such that all variables are in the model. Next, we must set all variables to be numeric, as ANNs require all numeric variables. We do this and check if they are in fact all numeric. Next, we create a matrix out of the data.

```
houses2 <- houses
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
```

```
str(houses2)
houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL
```

We split the data into training and test set. For our ANN we need not just to split the data into a training and test set, but we need a training target and a test target.

```
set.seed(123)
ind <- sample(2, nrow(houses2), replace = T, prob = c(0.9,0.1))
training_set <- houses2[ind==1,1:41]
test_set <- houses2[ind==2,1:41]
trainingtarget <- houses2[ind==1,42]
testtarget <- houses2[ind==2,42]
```

We create the model using the training set. We use a Python extension for our model, more specifically the keras sequential model, as well as tensorflow backend. We create our model with three hidden layers, with 100, 50 and 20 neurons. The last layer is the output layer. We do not need to specify the input layer, as the input shape is given. As an activation function, to activate the neurons we use the relu (rectified linear unit) function. To compile the model we must specify the loss measure, which we specify to be MSE. As the optimizer we use the adam (adaptive learning rate optimization algorithm) optimizer as it seems to work best in this setting. Other optimizers tested were SGD, RMSprop and so on. The metrics we choose to display are MAE. We fit the model to the training set in order to predict the training target. We iterate over 20 epochs and use a batch size of 32. While the model is iterating through the observations, it will validate and update the weights for backpropagation. The validation split is chosen to be 0.2.

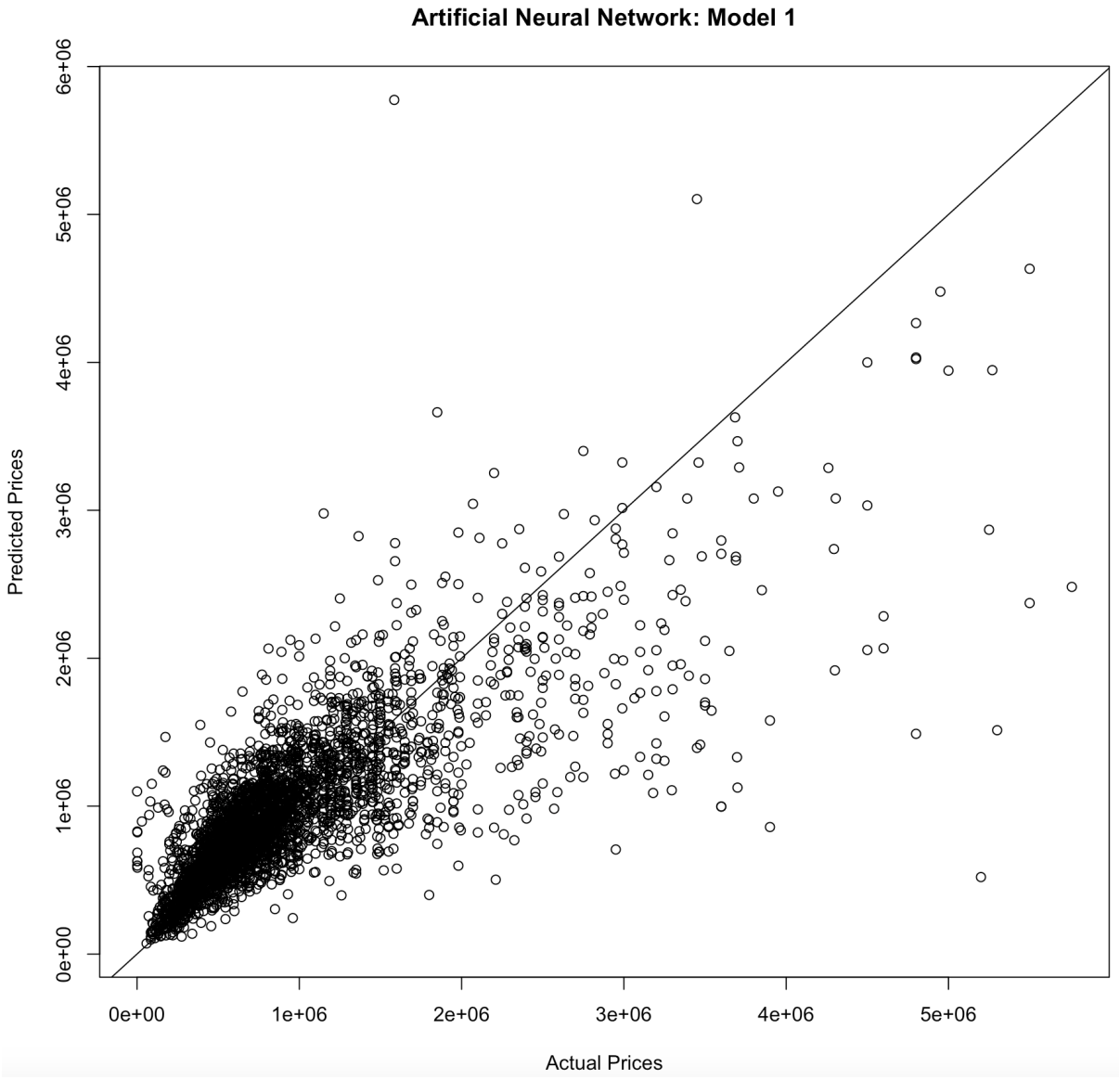
```
model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 50, activation = 'relu') %>%
        layer_dense(units = 20, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mse',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 20,
            batch_size = 32,
            validation_split = 0.2)
```

We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. For this first model we get an MAE of 264'346.2.

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 1",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 264346.2
```


Artificial Neural Network: Model 1
Predicted Prices
Actual Prices
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06
6e+06

Model 2

We reshape the data. For Model 2 we get rid of the variable municipality as it has increases the MAE of the model. For an ANN it is required that all variables are in numeric form. For this reason all variables must be set to numeric. Next a the dataset is transformed into matrix form for the model.

```
houses2 <- houses[-20]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
str(houses2)
houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL
```

We split the data into training and test set, as we did in Model 1.

```
set.seed(123)
ind <- sample(2, nrow(houses2), replace = T, prob = c(0.9,0.1))
training_set <- houses2[ind==1,1:40]
test_set <- houses2[ind==2,1:40]
trainingtarget <- houses2[ind==1,41]
testtarget <- houses2[ind==2,41]
```

We create the model using the training set. This part we leave as it was in Model 1.

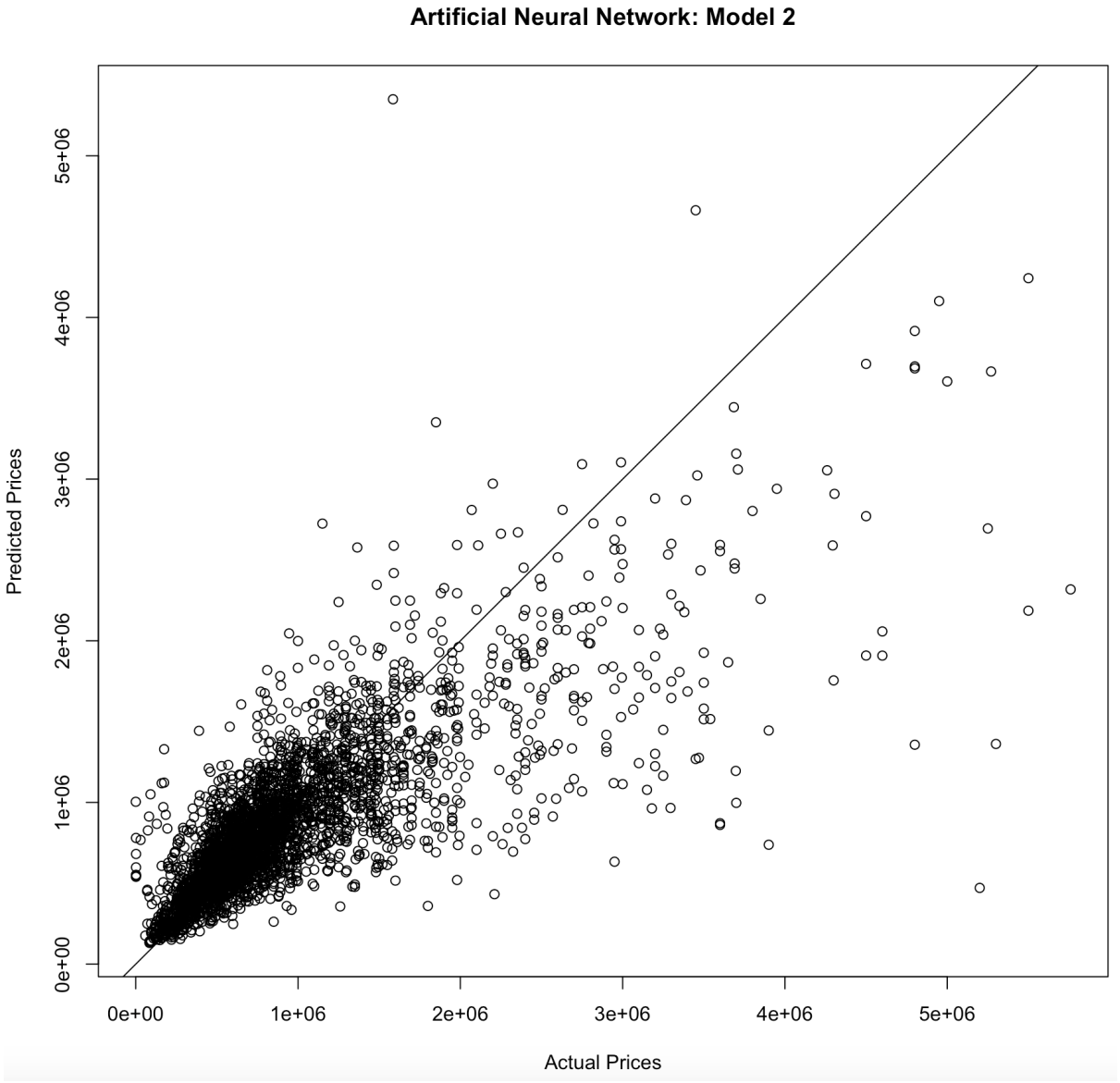
```
model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(40)) %>%
        layer_dense(units = 50, activation = 'relu') %>%
        layer_dense(units = 20, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mse',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 20,
            batch_size = 32,
            validation_split = 0.2)
```

We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We observe that the MAE has dropped to 256'506.3.

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 2",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 256506.3
```


Artificial Neural Network: Model 2
Predicted Prices
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06
Actual Prices
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06

Model 3

We reshape the data. For Model 3 we decide to get rid of all observations containing more than 30% NAs. We again transform the data to be numeric and in matrix form.

```
houses2 <- houses
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL
```

We split the data into training and test set. This time we use a split ratio of 80%

```
set.seed(123)
ind <- sample(2, nrow(houses2), replace = T, prob = c(0.8,0.2))
training_set <- houses2[ind==1,1:41]
test_set<- houses2[ind==2,1:41]
trainingtarget <- houses2[ind==1,42]
testtarget <- houses2[ind==2,42]
```

We create the model using the training set. This model is the same as in Model 1 and Model 2.

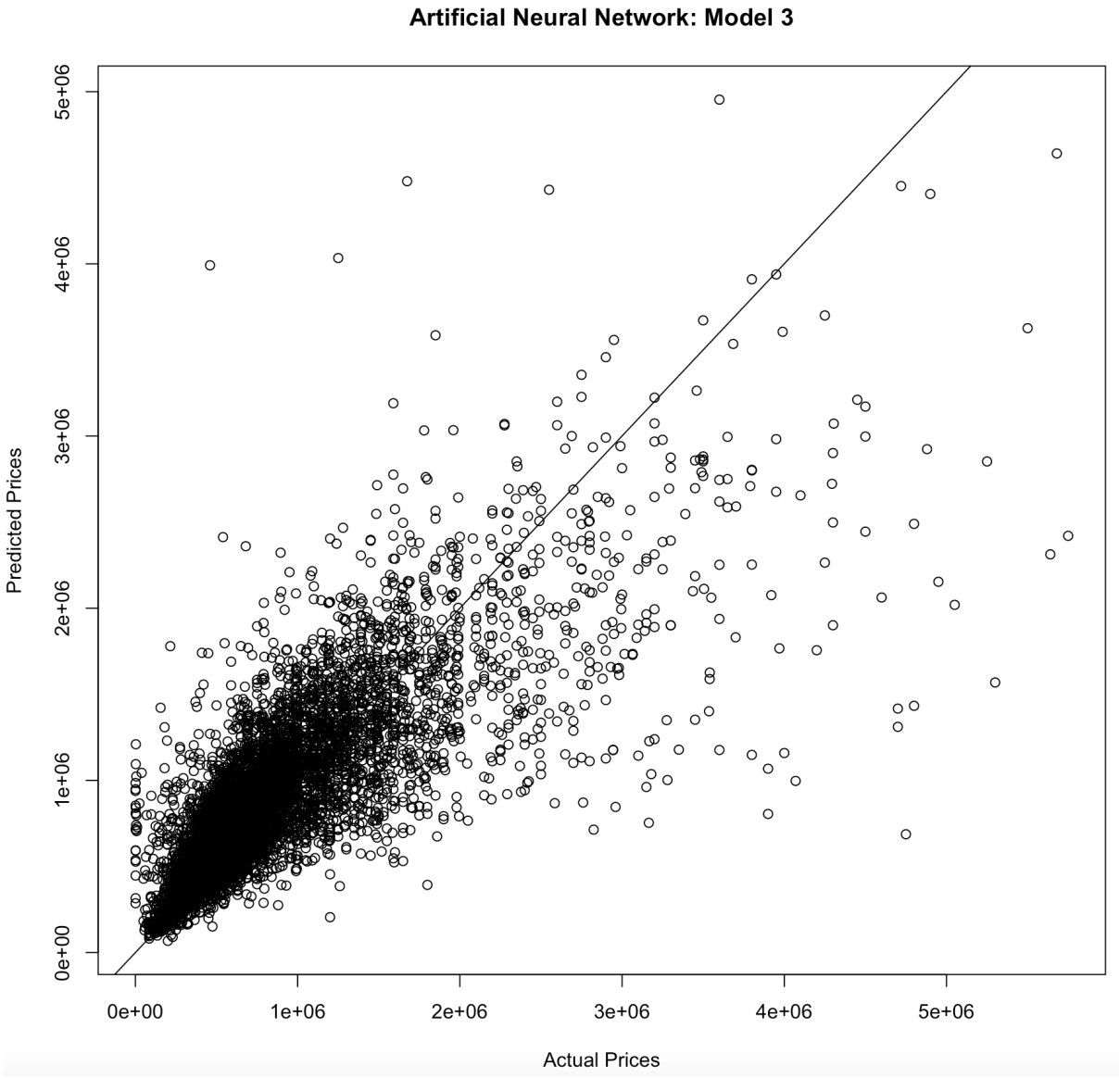
```
model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 50, activation = 'relu') %>%
        layer_dense(units = 20, activation = 'relu') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mse',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 20,
            batch_size = 32,
            validation_split = 0.2)
```

We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We see a further improvement in MAE, as it drops to 248'424.2.

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 3",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 248424.2
```


Artificial Neural Network: Model 3
Predicted Prices
Actual Prices
0e+00
1e+06
2e+06
3e+06
4e+06
5e+06

Model 4

We reshape the data. We keep this part as in Model 3.

```
houses2 <- houses
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL
```

We split the data into training and test set, as in Model 3.

```
set.seed(123)
ind <- sample(2, nrow(houses2), replace = T, prob = c(0.8,0.2))
training_set <- houses2[ind==1,1:41]
test_set<- houses2[ind==2,1:41]
trainingtarget <- houses2[ind==1,42]
testtarget <- houses2[ind==2,42]
```

We create the model using the training set. What is new in this model is that we for the first time apply feature scaling. Namely, we scale the training and test set using the mean and standard deviation of respective columns. In addition, this time we add neurons on to the second hidden layer. We increase the neurons from 50 to 100. In addition to this, we change the activation function of the last two hidden layers to be linear. This especially makes sense for the last layer as we are predicting a price.

```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 20, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mse',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 20,
            batch_size = 32,
            validation_split = 0.2)
```

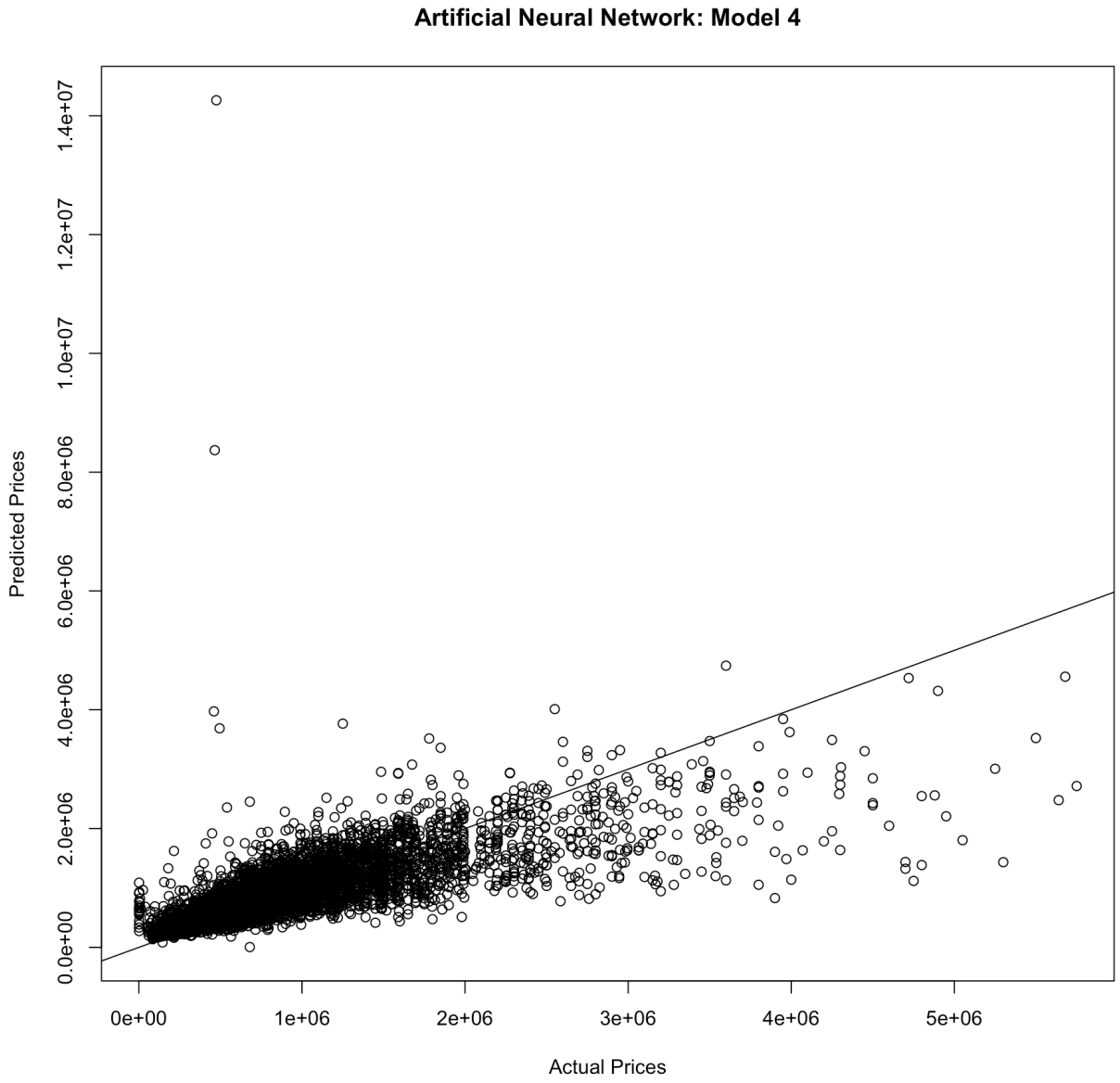
We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We manage to get the MAE down to 245'032.

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 4",
     xlab = "Actual Prices", ylab = "Predicted Prices")
```

```
abline(coef = c(0,1))
# MAE = 245032
```



Model 5

We reshape the data. For Model 5 we don't just get rid of rows with more than 30% NA, but we also remove outliers.

```
houses2 <- houses
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL
```

We split the data into training and test set, as in Model 3 and Model 4.

```
set.seed(123)
ind <- sample(2, nrow(houses2), replace = T, prob = c(0.8,0.2))
training_set <- houses2[ind==1,1:41]
test_set <- houses2[ind==2,1:41]
trainingtarget <- houses2[ind==1,42]
testtarget <- houses2[ind==2,42]
```

We create the model using the training set. For this model we keep the scaling and we add a fourth hidden layer, with another 100 neurons.

```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 20, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mse',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 20,
            batch_size = 32,
            validation_split = 0.2)
```

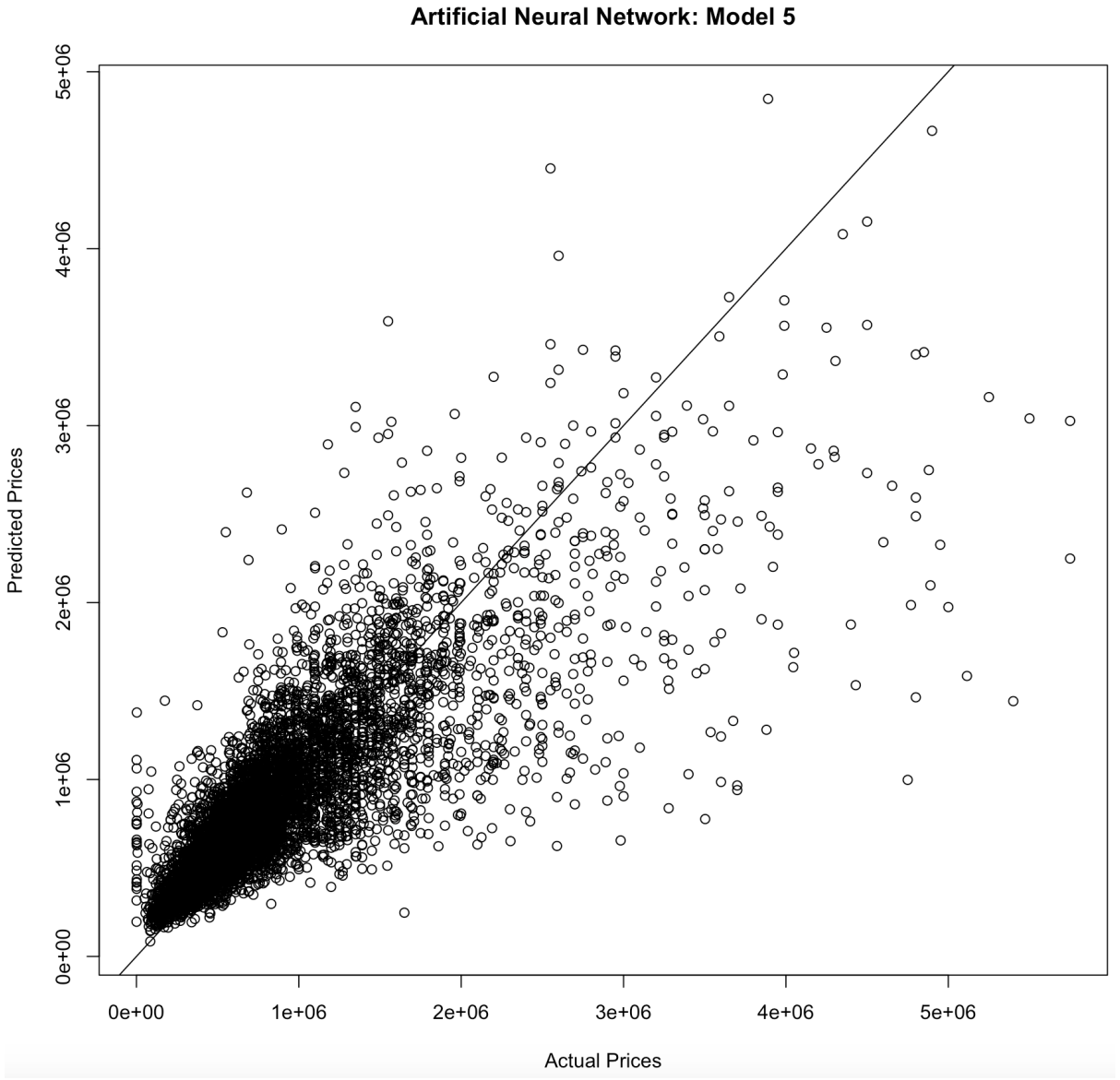
We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We manage to get the MAE down to 240'088.6.

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
```

```r
plot(testtarget,predictions, main = "Artificial Neural Network: Model 5",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 240088.6
```



Model 6

We reshape the data, as we did in Model 5. This time, we scale the data before splitting it into training and test set. For the scaling we try min max scaling instead of scaling by mean and standard deviation.

```
houses2 <- houses
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL

max = apply(houses2 , 2 , max)
min = apply(houses2, 2 , min)
houses2 = as.data.frame(scale(houses2, center = min, scale = max - min))

houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL
```

We split the data into training and test set.

```
set.seed(123)
ind <- sample(2, nrow(houses2), replace = T, prob = c(0.8,0.2))
training_set <- houses2[ind==1,1:41]
test_set <- houses2[ind==2,1:41]
trainingtarget <- houses2[ind==1,42]
testtarget <- houses2[ind==2,42]
```

We create the model using the training set. The setup of the model stays the same.

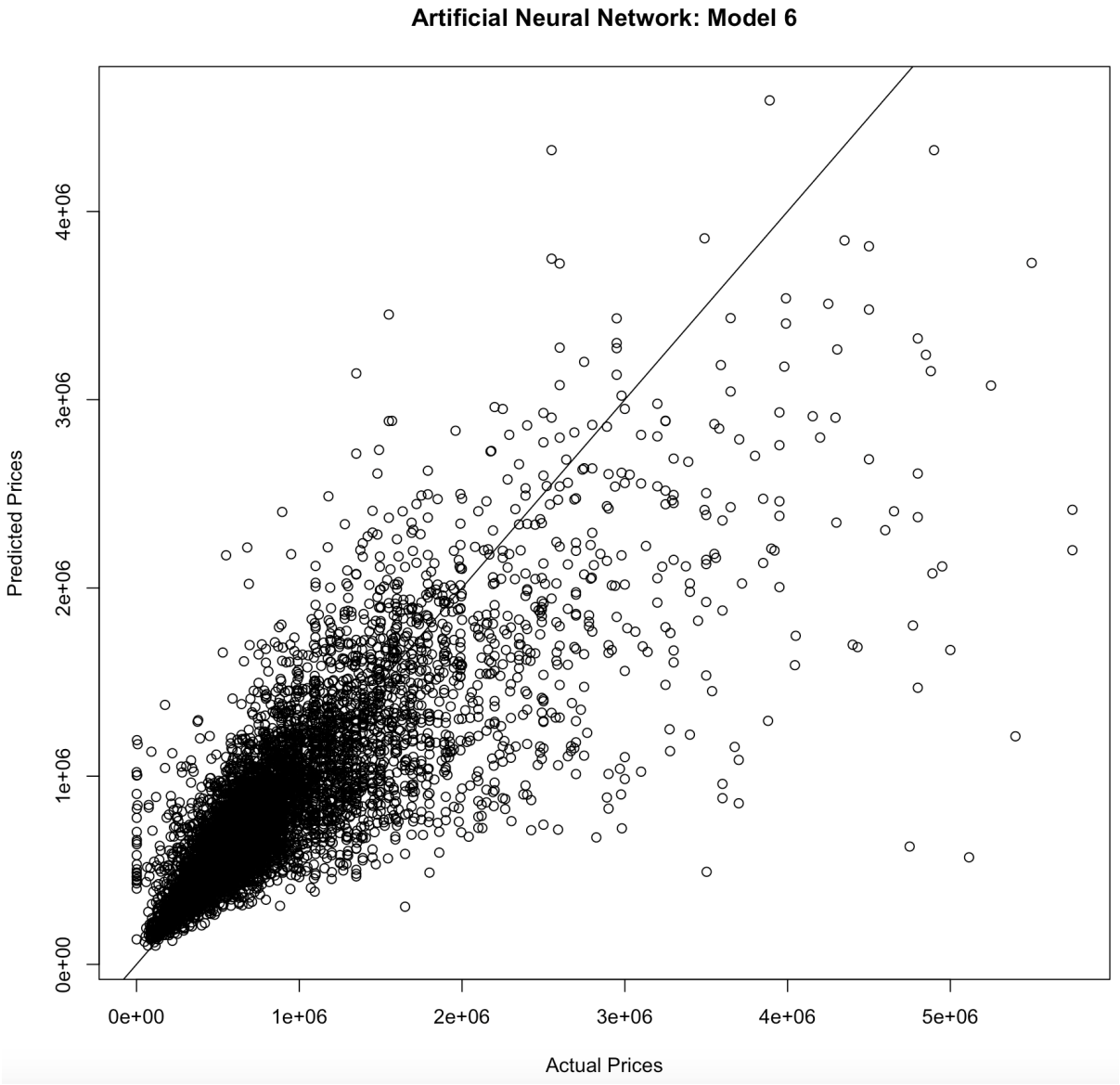
```
model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 20, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mse',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 20,
            batch_size = 32,
            validation_split = 0.2)
```

We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We manage to get an MAE of 228'286.2.

```r
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 6",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
testtarget <- testtarget * (5850000 - 300) + 300
predictions <- predictions * (5850000 - 300) + 300
# MAE = 228286.2
```



Model 7

We reshape the data. We keep this part as in previous models.

```
houses2 <- houses
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL
```

We split the data into training and test set, as in previous models.

```
set.seed(123)
ind <- sample(2, nrow(houses2), replace = T, prob = c(0.9,0.1))
training_set <- houses2[ind==1,1:41]
test_set <- houses2[ind==2,1:41]
trainingtarget <- houses2[ind==1,42]
testtarget <- houses2[ind==2,42]
```

We create the model using the training set. We go back to scaling with mean and standard deviation. We increase the number of neurons in the fourth layer to 100.

```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mse',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 20,
            batch_size = 32,
            validation_split = 0.2)
```

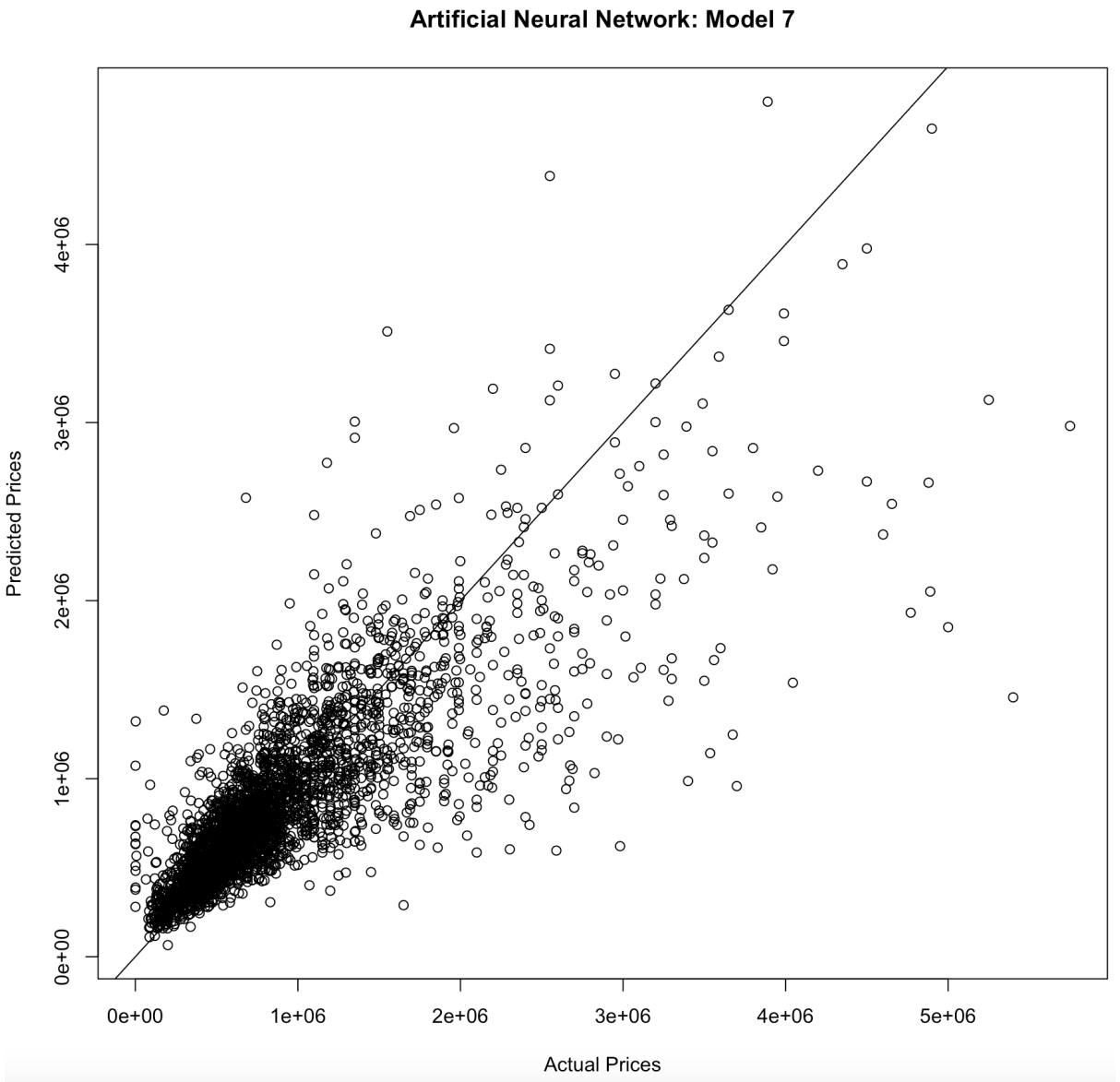
We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We get an MAE of 227'533.2.

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 7",
```

```
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 227533.2
```



Model 8

We reshape the data, as in previous models.

```
houses2 <- houses
houses2[houses2 == 0] <- NA
houses2 <- houses2[which(rowMeans(!is.na(houses2)) > 0.3), ]
houses2[is.na(houses2)] <- 0
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL
```

We split the data into training and test set, as in previous models.

```
set.seed(123)
ind <- sample(2, nrow(houses2), replace = T, prob = c(0.9,0.1))
training_set <- houses2[ind==1,1:41]
test_set <- houses2[ind==2,1:41]
trainingtarget <- houses2[ind==1,42]
testtarget <- houses2[ind==2,42]
```

We create the model using the training set. We scale as before. We add on two more layers, adding up to six layers in total. The two new layers each bring 100 neurons with them. We lower the batch size to 25. We change the loss to MAE, instead of MSE.

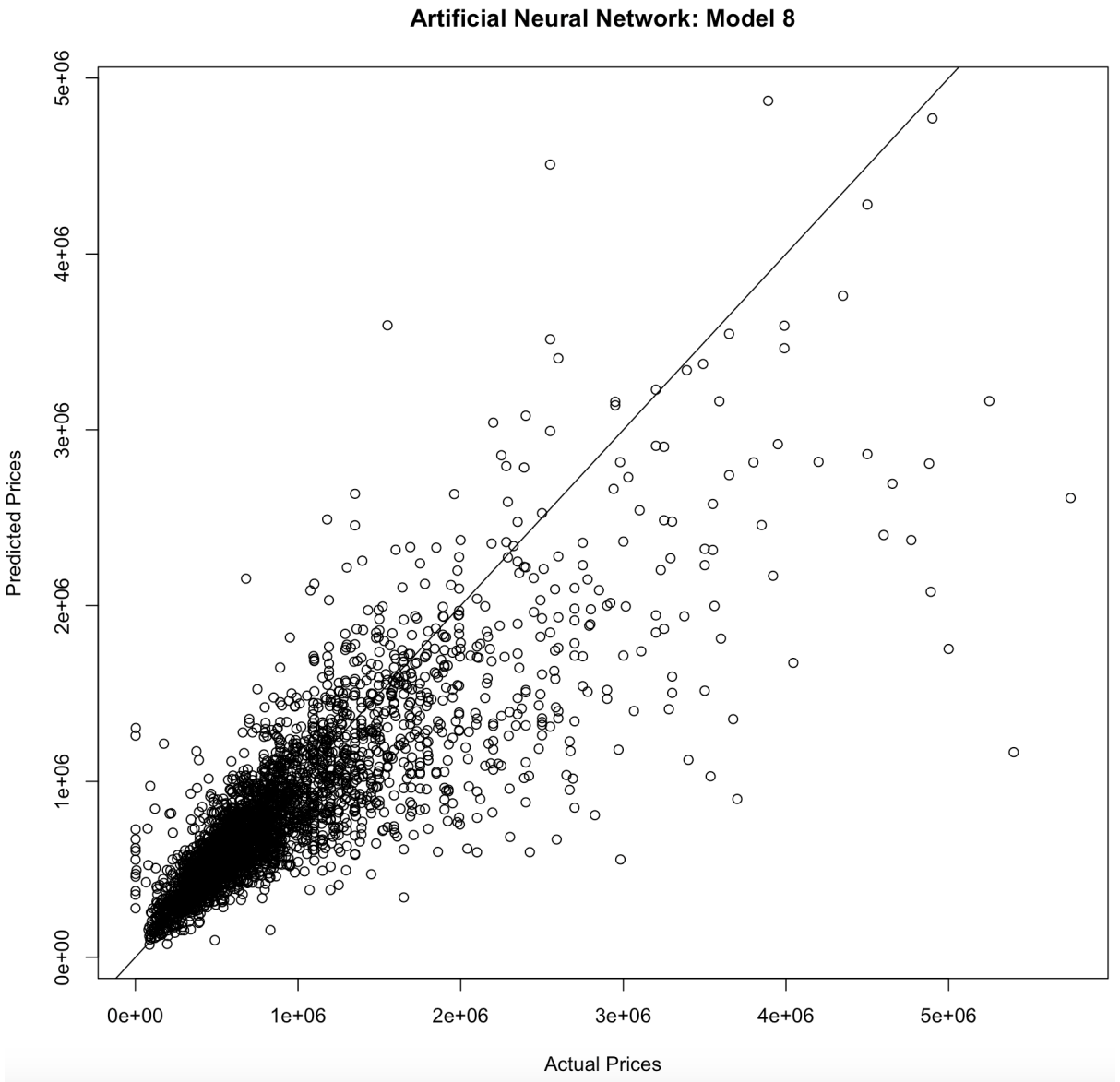
```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mae',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 20,
            batch_size = 25,
            validation_split = 0.2)
```

We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We achieve an MAE of 215'258.7.

```r
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 8",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 215258.7
```



Model 9

We reshape the data. This time we get rid of outliers, but not rows with more than 30% NAs.

```
houses2 <- houses
houses2 <- houses2[complete.cases(houses2), ]
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL
```

We split the data into training and test set.

```
set.seed(123)
ind <- sample(2, nrow(houses2), replace = T, prob = c(0.9,0.1))
training_set <- houses2[ind==1,1:41]
test_set <- houses2[ind==2,1:41]
trainingtarget <- houses2[ind==1,42]
testtarget <- houses2[ind==2,42]
```

We create the model using the training set. We increase the number of epochs to 30 and the batch size to 32 again.

```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mae',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 30,
            batch_size = 32,
            validation_split = 0.2)
```

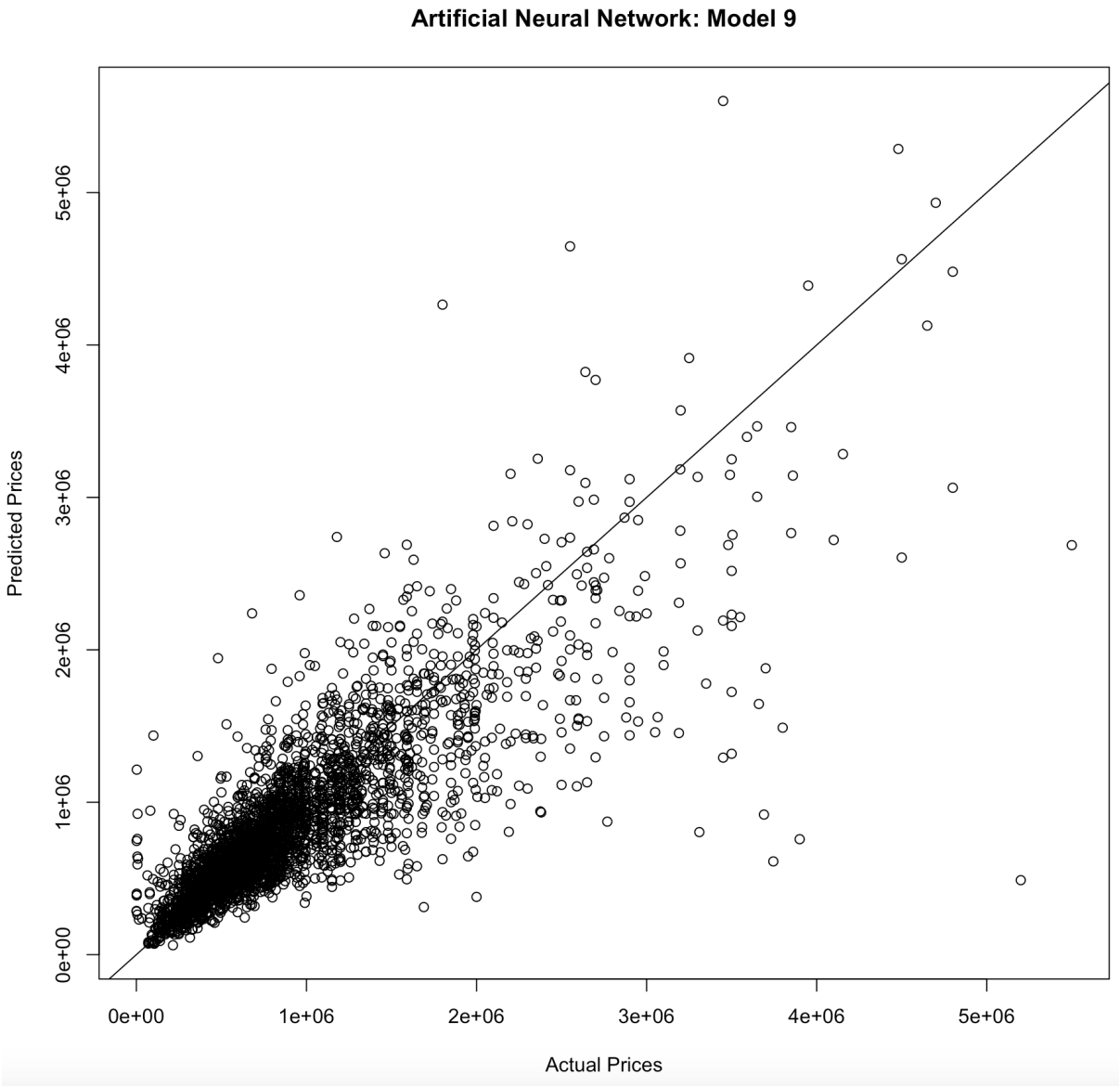
We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We get an MAE of 211'882.8

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 9",
```

```
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 211882.8
```



Model 10

We reshape the data as in Model 9.

```
houses2 <- houses
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL
```

We split the data into training and test set.

```
set.seed(123)
ind <- sample(2, nrow(houses2), replace = T, prob = c(0.8,0.2))
training_set <- houses2[ind==1,1:41]
test_set <- houses2[ind==2,1:41]
trainingtarget <- houses2[ind==1,42]
testtarget <- houses2[ind==2,42]
```

We create the model using the training set. For this model we decide to add a dropout rate to two layers to avoid overfitting. This dropout rate tells us what percentage of observations are dropped.

```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dropout(rate = 0.2) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dropout(rate = 0.2) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mae',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 30,
            batch_size = 32,
            validation_split = 0.2)
```

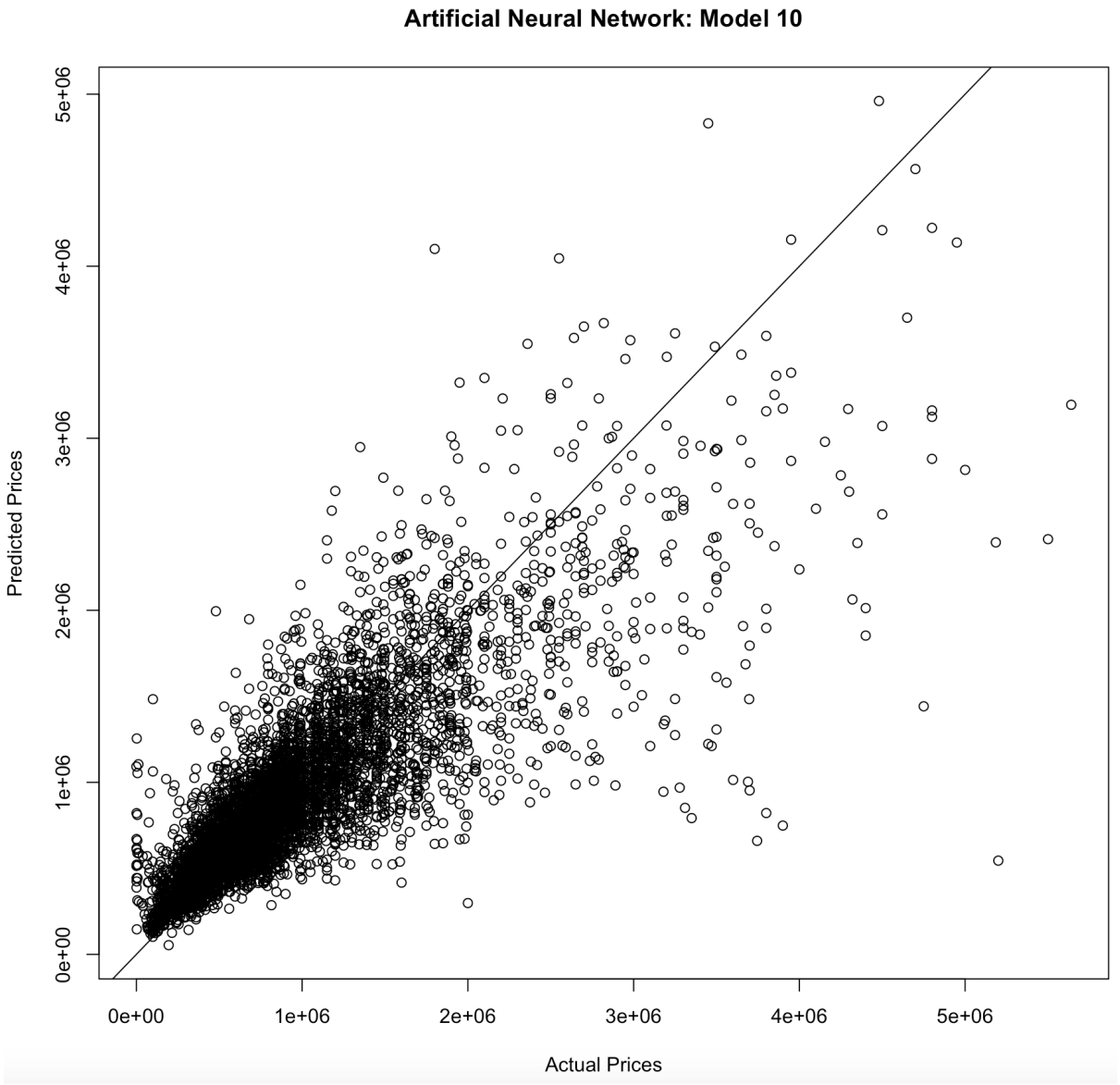
We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We get an MAE of 216'133.3

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
```

```r
plot(testtarget,predictions, main = "Artificial Neural Network: Model 10",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 216133.3
```



Model 11

We reshape the data, as before.

```
houses2 <- houses
houses2 <- houses2[complete.cases(houses2), ]
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
houses2 <- as.matrix(houses2)
dimnames(houses2) <- NULL
```

We split the data into training and test set.

```
set.seed(123)
ind <- sample(2, nrow(houses2), replace = T, prob = c(0.9,0.1))
training_set <- houses2[ind==1,1:41]
test_set <- houses2[ind==2,1:41]
trainingtarget <- houses2[ind==1,42]
testtarget <- houses2[ind==2,42]
```

We create the model using the training set. We get rid of dropout at one layer. We decrease the batch size to 25. We increase the validation split to 25%.

```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dropout(rate = 0.2) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mae',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 30,
            batch_size = 25,
            validation_split = 0.25)
```

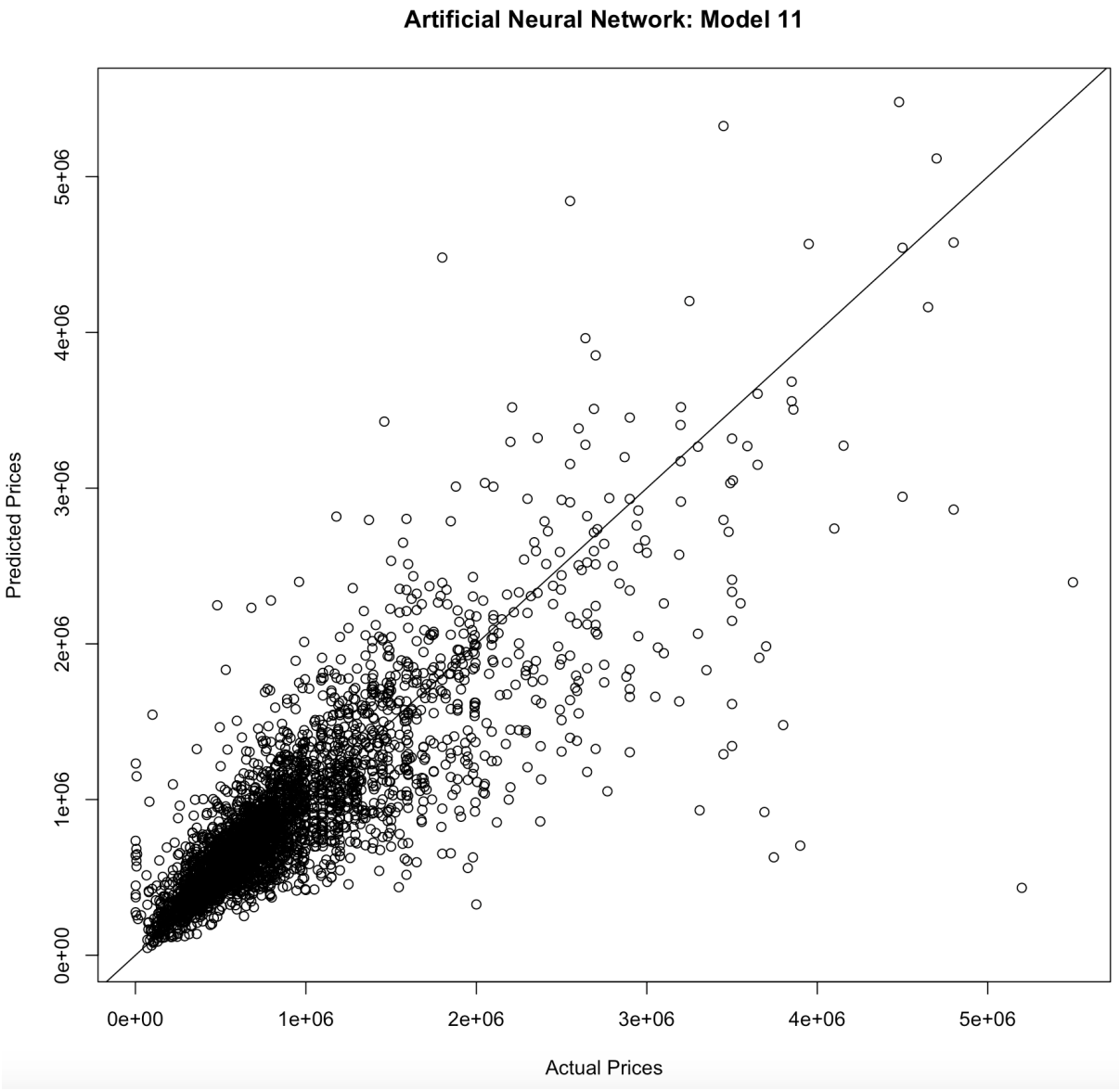
We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We get an MAE of 211'512.2

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
```

```r
plot(testtarget,predictions, main = "Artificial Neural Network: Model 11",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 211512.2
```



Model 12

We reshape the data, as before.

```
houses2 <- houses
houses2 <- houses2[complete.cases(houses2), ]
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
```

We split the data into training and test set.

```
set.seed(123)
split = sample.split(houses2$price, SplitRatio = 0.8)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
```

We create the model using the training set. This time we apply our feature scaling to the training target and test target as well.

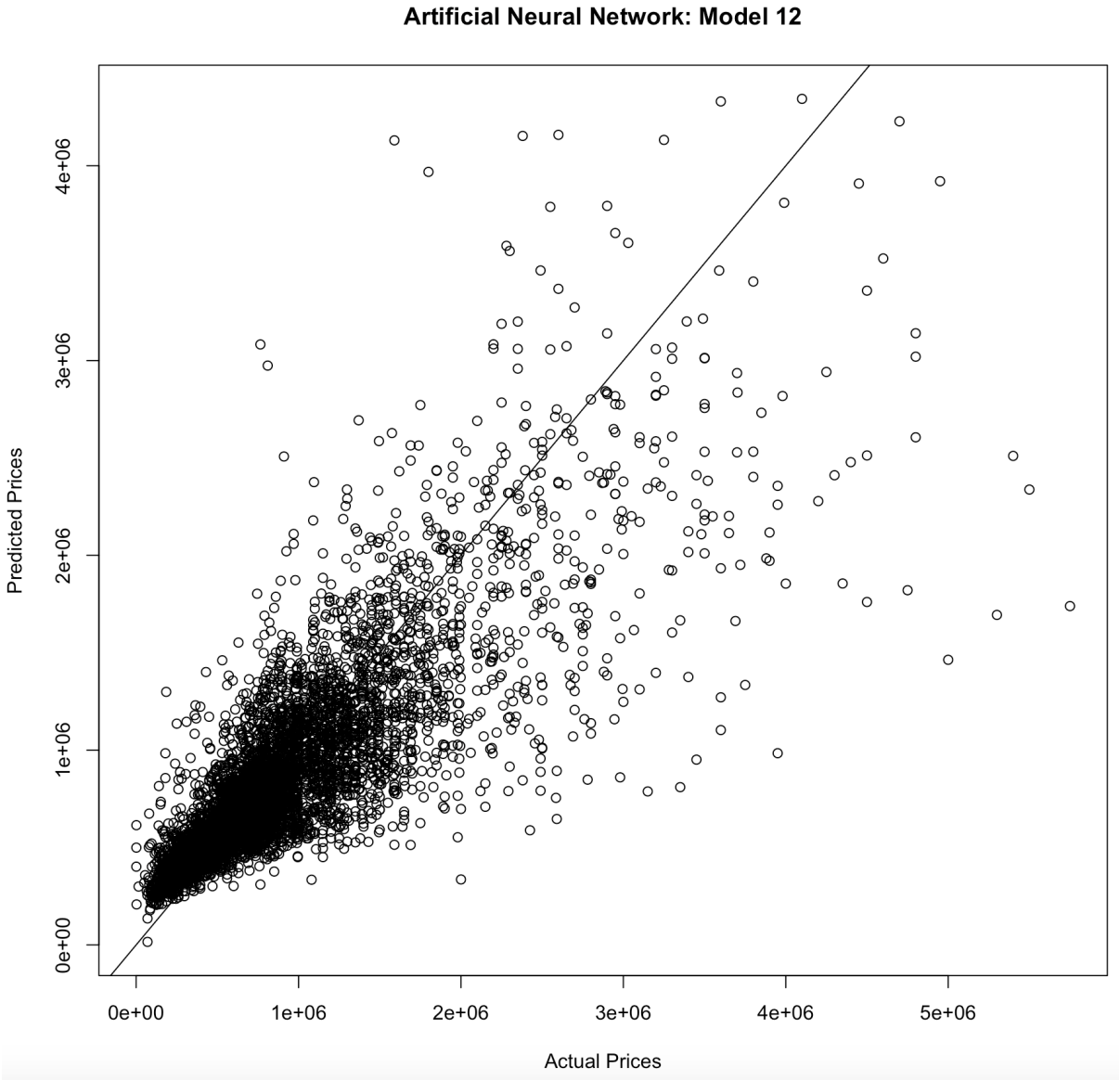
```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

training_set <- as.matrix(training_set)
dimnames(training_set) <- NULL
test_set <- as.matrix(test_set)
dimnames(test_set) <- NULL

trainingtarget <- training_set[,42]
training_set <- training_set[,1:41]
testtarget <- test_set[,42]
test_set <- test_set[,1:41]

model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dropout(rate = 0.2) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mae',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 30,
            batch_size = 25,
            validation_split = 0.25)
```

We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. Since we applied feature scaling to the targets, we must scale the values back in order to get prices. We get an MAE of 217'816.9.

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
testtarget <- testtarget * s['price'] + m['price']
predictions <- predictions * s['price'] + m['price']
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 12",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 217816.9
```



Cross validation of model 12

We reshape the data.

```
houses2 <- houses
houses2 <- houses2[complete.cases(houses2), ]
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
```

We split the data into training and test set.

```
set.seed(123)
split = sample.split(houses2$price, SplitRatio = 0.8)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
```

We create the model using the training set and evaluate it using the test set. We use k-fold cross validation.

```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

training_set <- as.matrix(training_set)
dimnames(training_set) <- NULL
test_set <- as.matrix(test_set)
dimnames(test_set) <- NULL

folds = createFolds(training_set[,42], k = 10)
cv = lapply(folds, function(x) {
  training_fold = training_set[-x, ]
  test_fold = training_set[x, ]

  trainingtarget_fold <- training_fold[,42]
  training_fold <- training_fold[,1:41]
  testtarget_fold <- test_fold[,42]
  test_fold <- test_fold[,1:41]

  model <- keras_model_sequential()
  model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dropout(rate = 0.2) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 1)

  model %>% compile(loss = 'mae',
                    optimizer = 'adam',
                    metrics = 'mae')

  mymodel <- model %>%
        fit(training_fold,
            trainingtarget_fold,
```

```
            epochs = 30,
            batch_size = 25,
            validation_split = 0.25)

  model %>% evaluate(test_fold, testtarget_fold)
  predictions <- model %>% predict(test_fold)

  testtarget_fold <- testtarget_fold * 6.700680e+05 + 8.863608e+05
  predictions <- predictions * 6.700680e+05 + 8.863608e+05
  MAE = mae(testtarget_fold,predictions)

  return(MAE)

  })

MAE = mean(as.numeric(cv))
# MAE = 227211.2
```

Model 13

We reshape the data, as before.

```
houses2 <- houses
houses2 <- houses2[complete.cases(houses2), ]
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
```

We split the data into training and test set.

```
set.seed(123)
split = sample.split(houses2$price, SplitRatio = 0.8)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
```

We create the model using the training set. This time we increase the number of neurons in the first layer to 300. We get rid of the dropout.

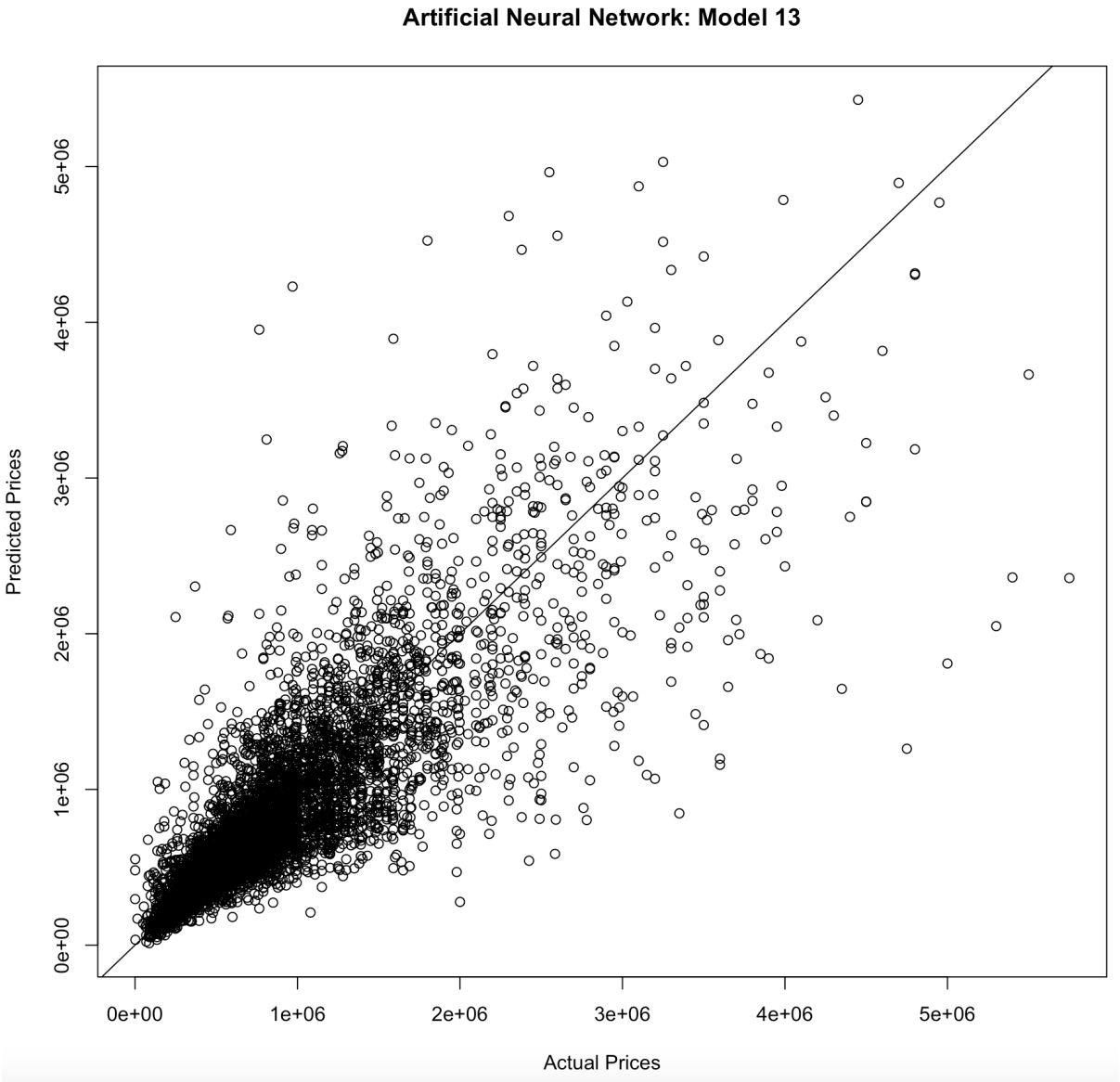
```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

training_set <- as.matrix(training_set)
dimnames(training_set) <- NULL
test_set <- as.matrix(test_set)
dimnames(test_set) <- NULL

trainingtarget <- training_set[,42]
training_set <- training_set[,1:41]
testtarget <- test_set[,42]
test_set <- test_set[,1:41]

model <- keras_model_sequential()
model %>%
        layer_dense(units = 300, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mae',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 30,
            batch_size = 25,
            validation_split = 0.25)
```

We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We get an MAE of 212'374.6

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
testtarget <- testtarget * s['price'] + m['price']
predictions <- predictions * s['price'] + m['price']
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 13",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 212374.6
```



Model 14

We reshape the data, as before.

```
houses2 <- houses
houses2 <- houses2[complete.cases(houses2), ]
houses2 <- houses2[which(houses2$area < 400), ]
houses2 <- houses2[which(houses2$area_useable < 1000), ]
houses2 <- houses2[which(houses2$rooms < 12), ]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
```

We split the data into training and test set. We decrease the split ratio to 80%.

```
set.seed(123)
split = sample.split(houses2$price, SplitRatio = 0.8)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
```

We create the model using the training set. We change the number of neurons in the first layer back to 100.

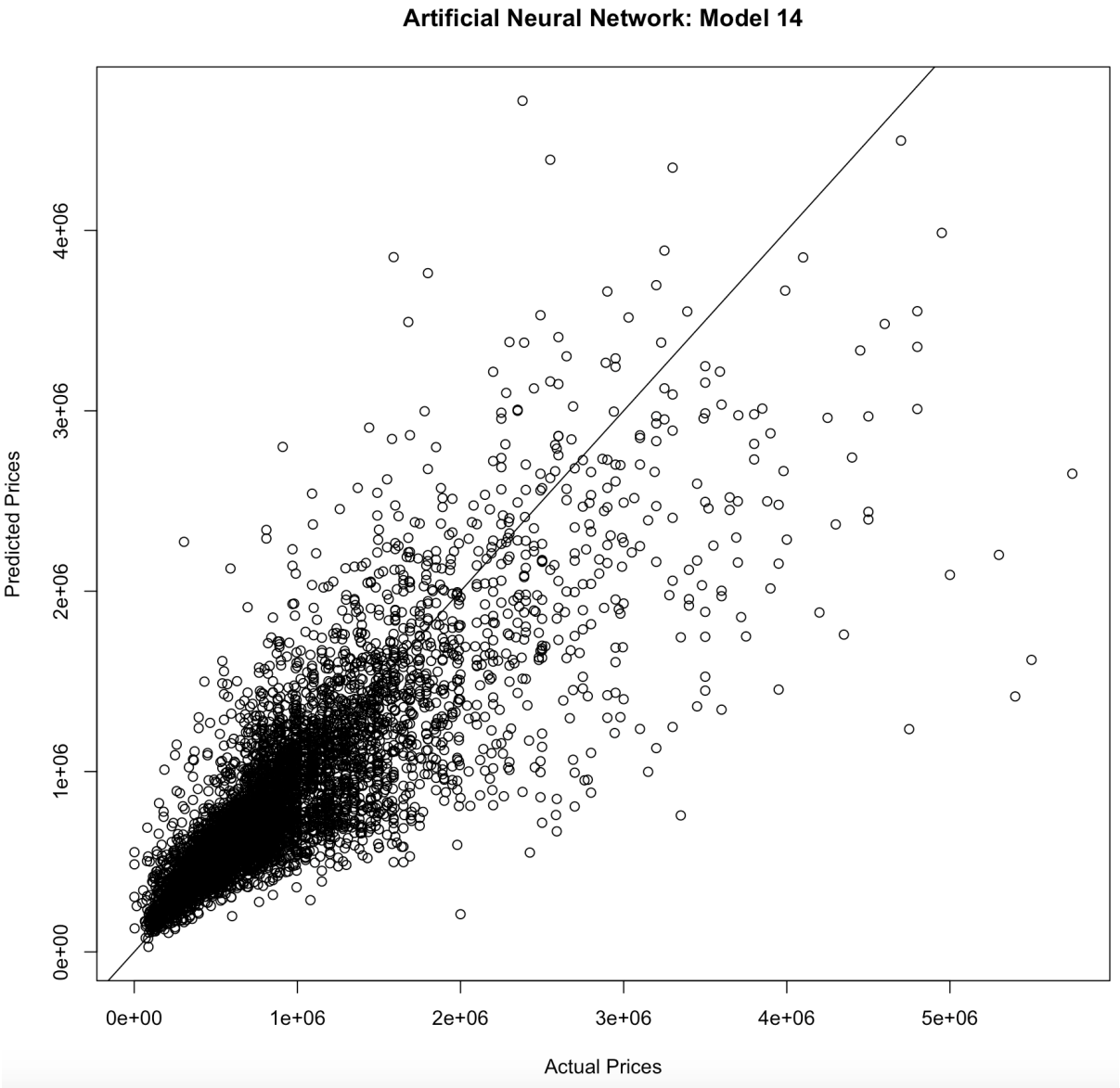
```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

training_set <- as.matrix(training_set)
dimnames(training_set) <- NULL
test_set <- as.matrix(test_set)
dimnames(test_set) <- NULL

trainingtarget <- training_set[,42]
training_set <- training_set[,1:41]
testtarget <- test_set[,42]
test_set <- test_set[,1:41]

model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(41)) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mae',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 30,
            batch_size = 25,
            validation_split = 0.25)
```

We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We get an MAE of 212'244.5

```
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
testtarget <- testtarget * s['price'] + m['price']
predictions <- predictions * s['price'] + m['price']
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 14",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 212244.5
```



Model 15

We reshape the data. This time we keep our entire dataset, except for kategorie.

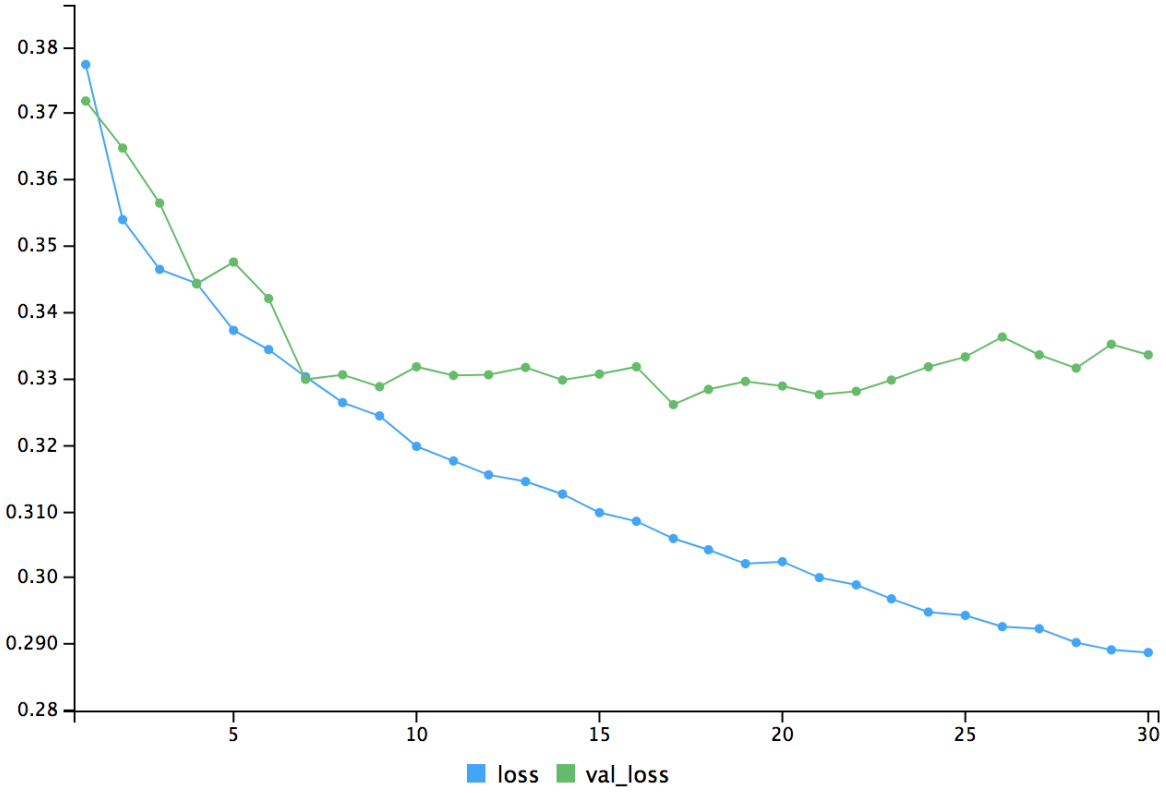
```
houses2 <- houses
houses2 <- houses2[-39]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)
```

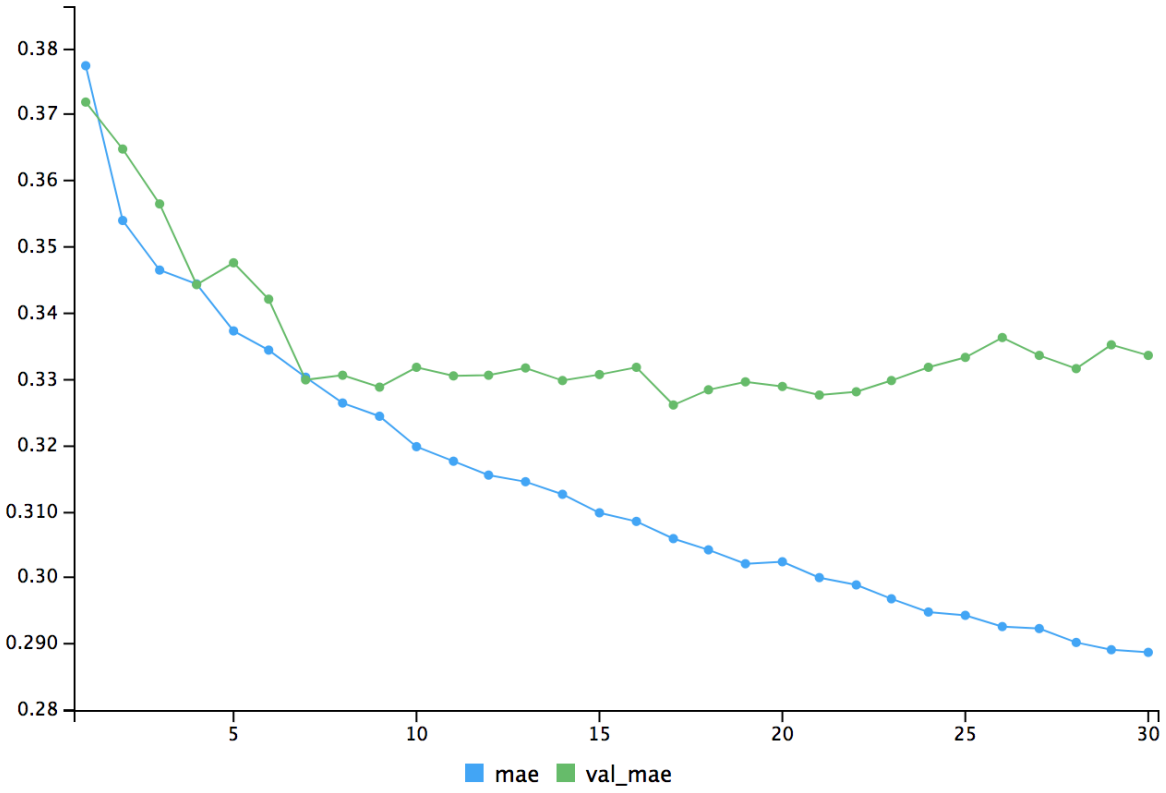
We split the data into training and test set.

```
set.seed(123)
split = sample.split(houses2$price, SplitRatio = 0.9)
training_set = subset(houses2, split == TRUE)
test_set = subset(houses2, split == FALSE)
```

We create the model using the training set.

```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

training_set <- as.matrix(training_set)
dimnames(training_set) <- NULL
test_set <- as.matrix(test_set)
dimnames(test_set) <- NULL

trainingtarget <- training_set[,41]
training_set <- training_set[,1:40]
testtarget <- test_set[,41]
test_set <- test_set[,1:40]

model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(40)) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mae',
                  optimizer = 'adam',
                  metrics = 'mae')

mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 30,
            batch_size = 25,
            validation_split = 0.25)
```



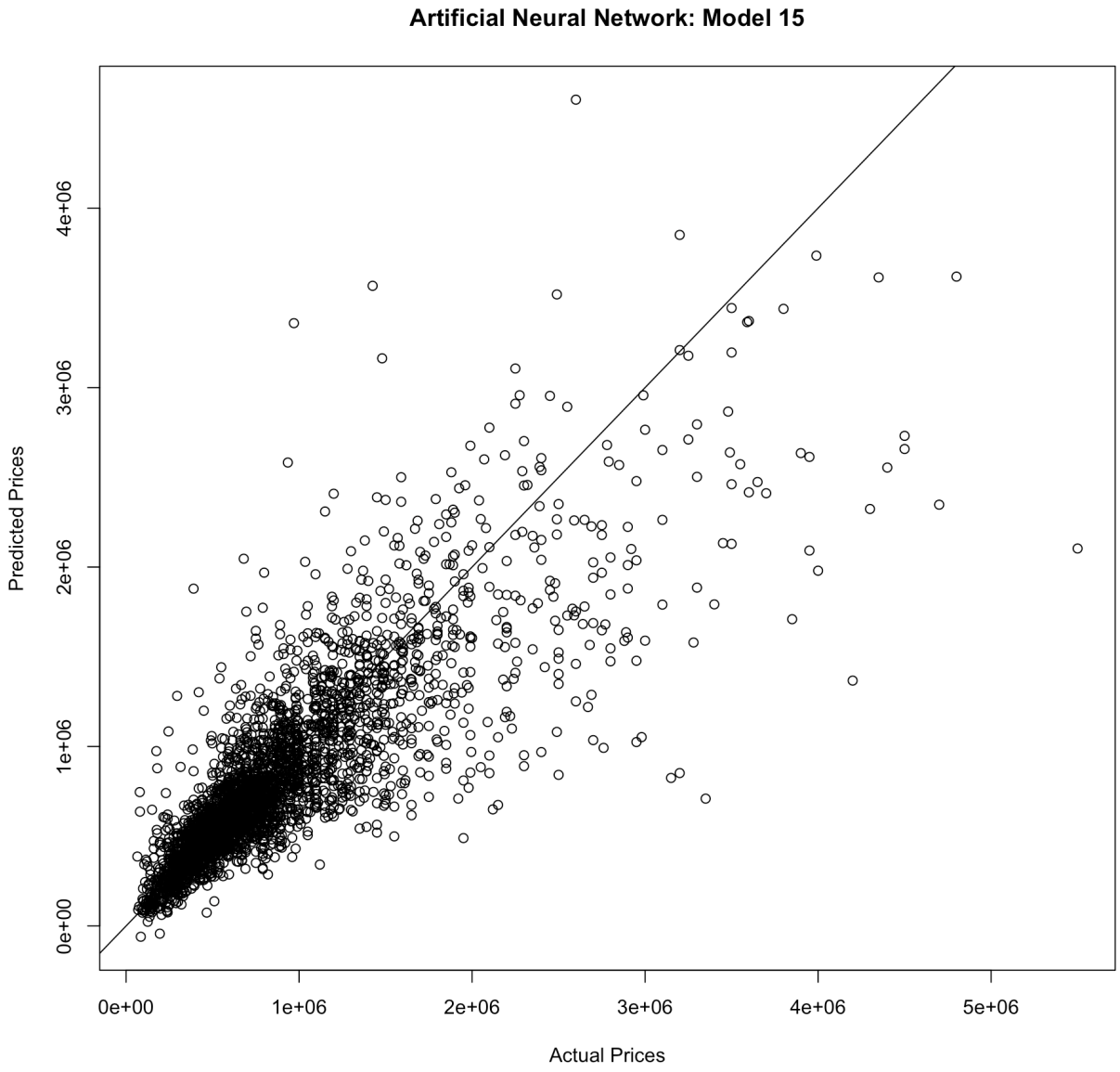


We apply the model to the test set.

```
model %>% evaluate(test_set, testtarget)
predictions <- model %>% predict(test_set)
```

We plot and evaluate the results. We get an MAE of 202'733.6, leading Model 15 to be the best ANN.

```r
mean((testtarget-predictions)^2)
mae(testtarget,predictions)
testtarget <- testtarget * s['price'] + m['price']
predictions <- predictions * s['price'] + m['price']
mae(testtarget,predictions)
plot(testtarget,predictions, main = "Artificial Neural Network: Model 15",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(coef = c(0,1))
# MAE = 202733.6
```



Our Model

For our final model to predict the actual test set prices, we decide to use Model 15 of the ANN models. There are various reasons for our choice. Firstly, the model seems to perform very well in terms of test MAE. Secondly, the ANN model transforms all values to numeric values, eliminating all issues of new factor levels being present in the test set. Thirdly, ANNs allow for non linear relationships. For all these reasons we decide to apply Model 15 of the ANN models as our final model.

We begin by reshaping the data. We use the entire dataset, except for kategorie. We apply the same reshaping to both the training as well as the test set.

```
houses2 <- houses
houses2 <- houses2[-39]
houses2 %<>% mutate_if(is.factor, as.numeric)
houses2 %<>% mutate_if(is.integer, as.numeric)

x_test2 <- x_test
x_test2 <- x_test2[-39]
x_test2 %<>% mutate_if(is.factor, as.numeric)
x_test2 %<>% mutate_if(is.integer, as.numeric)
```

Next, we define the training and test set.

```
training_set <- houses2
test_set <- x_test2
```

Then we create our model. We begin by scaling the data using mean and standard deviation. Then we put our data in matrix form. Next, we split into training set, training target, test set and test target. Then we build our model, which consists of six layers, each carrying 100 neurons. We use both relu as well as linear activation functions. In compiling the model we apply MAE for the loss function, the adam optimizer and we set MAE as our metric. We fit the model using the training set and training target. For this we iterate over 30 epochs with a batch size of 25 and validation split of 0.25.

```
m = colMeans(training_set)
s <- apply(training_set, 2, sd)
training_set <- scale(training_set, center = m, scale = s)
test_set <- scale(test_set, center = m, scale = s)

training_set <- as.matrix(training_set)
dimnames(training_set) <- NULL
test_set <- as.matrix(test_set)
dimnames(test_set) <- NULL

trainingtarget <- training_set[,41]
training_set <- training_set[,1:40]
testtarget <- test_set[,41]
test_set <- test_set[,1:40]

model <- keras_model_sequential()
model %>%
        layer_dense(units = 100, activation = 'relu', input_shape = c(40)) %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 100, activation = 'relu') %>%
        layer_dense(units = 100, activation = 'linear') %>%
        layer_dense(units = 1)

model %>% compile(loss = 'mae',
                  optimizer = 'adam',
                  metrics = 'mae')
```

```
mymodel <- model %>%
        fit(training_set,
            trainingtarget,
            epochs = 30,
            batch_size = 25,
            validation_split = 0.25)
```

Finally, we apply the model to the test set to get our predictions, which we then scale back.

```
predictions <- model %>% predict(test_set)
predictions <- predictions * s['price'] + m['price']
Y_test <- data.frame("id" = c(1:15000), "price" = predictions)
write.csv(Y_test, file = "Y_test.csv", row.names=FALSE)
```