CompStat/R - Paper 1

Group 2: Carlo Michaelis, Patrick Molligo, Lukas Ruff

11 May 2016

Part I

1. What are the atomic vector types in R? Explain which value they can take and give an example!

There are six *atomic* (or *basic*) vector types in R:

- **character**: Text, i.e. string variables.
- **numeric**: Real numbers, i.e. float variables.
- **integer**: Integers, i.e. values in $\mathbb{Z}$.
- **complex**: Complex numbers, i.e. a pair of values with a real and imaginary part.
- **logical**: Boolean variables, i.e. either 1 (`TRUE`) or 0 (`FALSE`).
- **raw**: A raw vector contains fixed-length sequences of bytes.

Examples

```
a <- c("blue", "red", "yellow")      ## character
b <- c(pi, exp(1), 0, 1)             ## numeric
c <- 1:10                            ## integer
d <- c(0+1i, 1+1i)                   ## complex
e <- c(TRUE, FALSE)                  ## logical
f <- raw(length = 3L)                ## raw
```

It is important to note, that a vector can only contain elements of the same type. We can check the type of an object using the `class`-function.

```
# verify types by using class function
lapply(list(a,b,c,d,e,f), class)
```

```
## [[1]]
## [1] "character"
##
## [[2]]
## [1] "numeric"
##
## [[3]]
## [1] "integer"
##
## [[4]]
## [1] "complex"
##
## [[5]]
## [1] "logical"
##
## [[6]]
## [1] "raw"
```

2. What is the difference between generic and atomic vectors?

- An *atomic vector* can only contain objects of the same class. An example would be a vector which contains only integers.
- A *generic vector* (in R represented as a `list`) can contain objects of different classes. An example would be a vector which contains characters and numbers.

3. Explain the following statement: "A data frame is a list, but not every list is a data frame."

- A `list` is an object containing collections of objects. The types of the elements of the list can be different. It is for example allowed that a `list` contains a vector of real values (doubles) and a vector of characters. The lengths of the contained vectors can be *different*.
- A `data frame` is also an object containing colletions of objects. The types of the elements of the list can also be different. But the lengths of the contained vectors have to be *the same*. We can think of a `data frame` as a table or matrix, where each row is an observation and each column a different variable. The length of each element or column are the number of rows or observations.

In conclusion, `list` and `data frame` are very similar, but the `data frame` has one more restriction (same length of all vectors). That is why a `data frame` is always a `list`, but a `list` is not always a `data frame`.

Part II

The following code will perform a simulation of 100'000'000 samples from a $\mathcal{N}(5, 10)$ distribution, i.e. a normal distribution with mean $\mu = 5$ and standard deviation $\sigma = 10$. For reproducibility, we set a seed for the random number generator. In a second step, the cumulative sums of the first 100 samples are computed in two different ways, where the function `cumsum` returns a vector where element i is the cumulative sum up to sample i. Finally, we check if the two ways of computing the cumulative sums up to sample 100 result in exactly equal vectors.

For random number generation R uses pseudo-random numbers. Starting from an initial state, called *seed state*, it will produce a deterministic sequence, which is used as random numbers. By choosing the same seed in every turn, we get the same results. To make the results of random numbers comparable, we first set the seed in a specific state, using `set.seed`.
After setting the seed, we define a vector with (pseudo-) random values. Using the `rnorm`-function we create the $1 \cdot 10^8$ normal distributed random values and save them in a vector called `largeVector`.

```r
# Set the state of the random number generator (RNG) to 1
set.seed(1)

# Perform simulation of 1e8 samples from a normal distribution with mean 5
# and standard deviation 10
largeVector <- rnorm(1e8, mean=5, sd=10)
```

The function `cumsum`, which is used in the next code block, calculates the cumulative sum of the values of the vector. It takes all elements one by one and calculates for this element the sum of all elements before, including the current element. These values will be the elements of the new vector. Consider the following example:

$$\begin{pmatrix} 1 \\ 4 \\ 3 \end{pmatrix} \xrightarrow{\texttt{cumsum}} \begin{pmatrix} 1 \\ 5 \\ 8 \end{pmatrix}$$

In case `a` it is doing `cumsum` on the whole vector `largeVector`. Afterwards it just takes the first 100 elements and saves them in vector `a`. In case `b` it first takes the 100 first elements of `largeVector` and calculates the `cumsum` afterwards, with only those 100 elements. The result is saved in vector `b`. In the end the two vectors `a` and `b` are checked for exact equality, using the `identical`-function.

```
# Compute the cumulative sums for the whole "largeVector" and subset the
# first 100 elements
a <- cumsum(largeVector)[1:100]

# Compute the cumulative sums only for the first 100 elements of
# "largeVector"
b <- cumsum(largeVector[1:100])

# Check, whether both ways of computation are exactly identical
identical(a, b)
```

```
## [1] TRUE
```

Of course, both ways of computing the cumulative sums for the first 100 samples above have the same result and hence `identical(a, b)` returns `TRUE`, but computation `a` is very inefficient compared to computation `b` since we first apply `cumsum` to the whole `largeVector`, i.e. we compute the cumulative sums for $100'000'000$ elements and then only look at the first 100 elements. Computation `b` instead only computes the cumulative sums for the subset of the first 100 elements directly.

In the following code, we stop the time for each of the two ways of computation using the `system.time`-function.

```
# Computation method a
system.time(cumsum(largeVector)[1:100])
```

```
##    user  system elapsed
##   0.853   0.766   1.718
```

```
# Computation method b
system.time(cumsum(largeVector[1:100]))
```

```
##    user  system elapsed
##       0       0       0
```

The *user* CPU time and the *system* CPU time is a technical distinction in time running the R code and time used in operating system kernel on behalf of the R code. The interesting time is the *elapsed* time, which is the sum of the *user* time and the *system* time. We can see that the first operation of taking the `cumsum` of the whole `largeVector` with its 100 million elements (and reducing the vector to 100 elements afterwards) takes a lot more CPU calculation time than taking the `cumsum` of the first 100 elements directly.

The results prove our reasoning above, the second method is much more efficient than the first one, because finally we are only interested in the `cumsum` of the first 100 elements of the vector.

Part III

In our regression analysis, we will analyze the rental prices in Munich from 2003 using the dataset "Münchner Mietspiegel 2003". The dataset contains 13 variables from 2053 apartments in Munich. The variables are the following:

- **nm**: net rent in EUR
- **nmqm**: net rent per m^2 in EUR
- **wfl**: living space in m^2
- **rooms**: number of rooms
- **bj**: year of construction
- **bez**: district

- **wohngut**: good residential area? (Y=1, N=0)
- **wohnbest**: best residential area? (Y=1, N=0)
- **ww0**: hot water supply? (Y=0, N=1)
- **zh0**: central heating? (Y=0, N=1)
- **badkach0**: tiled bathroom? (Y=0, N=1)
- **badextra**: optional extras in bathroom? (Y=1, N=0)
- **kueche**: luxury kitchen? (Y=1, N=0)

We would like to build a model to predict and explain rental prices, i.e. the dependent variable of our regression analysis will be the net rent in EUR `nm`. All other variables are potential explanatory variables for our linear regression model.

Data Import, Validation and Cleaning

First, we read the data into our global environment using the `load`-function and have a first look at it using `str` and `summary`:

```r
# Load data
load('miete.Rdata')
# Get a first overview
str(miete)
```

```
## 'data.frame':    2053 obs. of  13 variables:
##  $ nm      : num  741 716 528 554 698 ...
##  $ nmqm    : num  10.9 11.01 8.38 8.52 6.98 ...
##  $ wfl     : int  68 65 63 65 100 81 55 79 52 77 ...
##  $ rooms   : int  2 2 3 3 4 4 2 3 1 3 ...
##  $ bj      : num  1918 1995 1918 1983 1995 ...
##  $ bez     : Factor w/ 25 levels "1","2","3","4",..: 2 2 2 16 16 16 6 6 6 6 ...
##  $ wohngut : int  1 1 1 0 1 0 0 0 0 0 ...
##  $ wohnbest: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ww0     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zh0     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ badkach0: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ badextra: int  0 0 0 1 1 0 1 0 0 0 ...
##  $ kueche  : int  0 0 0 0 1 0 0 0 0 0 ...
```

```r
summary(miete)
```

```
##        nm               nmqm             wfl             rooms
##  Min.   :  77.31   Min.   : 1.470   Min.   : 17.0   Min.   :1.000
##  1st Qu.: 389.95   1st Qu.: 6.800   1st Qu.: 53.0   1st Qu.:2.000
##  Median : 534.30   Median : 8.470   Median : 67.0   Median :3.000
##  Mean   : 570.09   Mean   : 8.394   Mean   : 69.6   Mean   :2.598
##  3rd Qu.: 700.48   3rd Qu.:10.090   3rd Qu.: 83.0   3rd Qu.:3.000
##  Max.   :1789.55   Max.   :20.090   Max.   :185.0   Max.   :6.000
##
##        bj            bez           wohngut          wohnbest
##  Min.   :1918   9      : 177   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:1948   2      : 161   1st Qu.:0.0000   1st Qu.:0.00000
##  Median :1960   5      : 139   Median :0.0000   Median :0.00000
##  Mean   :1958   4      : 137   Mean   :0.3911   Mean   :0.02192
##  3rd Qu.:1973   3      : 132   3rd Qu.:1.0000   3rd Qu.:0.00000
##  Max.   :2001   25     : 117   Max.   :1.0000   Max.   :1.00000
##                 (Other):1190
```

```
##       ww0               zh0              badkach0        badextra
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.00000
##  Median :0.00000   Median :0.00000   Median :0.0000   Median :0.00000
##  Mean   :0.03507   Mean   :0.08524   Mean   :0.1851   Mean   :0.09303
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.0000   3rd Qu.:0.00000
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.0000   Max.   :1.00000
##
##      kueche
##  Min.   :0.00000
##  1st Qu.:0.00000
##  Median :0.00000
##  Mean   :0.07306
##  3rd Qu.:0.00000
##  Max.   :1.00000
##
```

Before we go into the variables of our data in detail, let's do a quick check on missing values using the `is.na`-function:

```r
# Check for NA's
sum(is.na(miete))
```

```
## [1] 0
```

There seem to be no missing values in our dataset.

Now, let's think about plausibility and the data types of our variables. From the five-number summary (`Min.`, `1st Qu.`, `Median`, `3rd Qu.`, `Max,`) and `Mean` values shown by `summary`, we can see that `nm`, `nmqm`, `wfl`, and `rooms` are properly formatted and within reasonable ranges. By definition of the variables, we should have that

$$\frac{\texttt{nm}}{\texttt{wfl}} = \texttt{nmqm}$$

Let's check whether this relationship holds by comparing the summary of `nmqm` with the summary of $\frac{\texttt{nm}}{\texttt{wfl}}$ and having a look at the sum of absolute errors (in relative terms):

```r
summary(miete$nmqm)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.470   6.800   8.470   8.394  10.090  20.090
```

```r
# Rebuild nmqm from nm and wfl
nmqm2 <- miete$nm / miete$wfl
summary(nmqm2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.467   6.799   8.466   8.394  10.090  20.090
```

```r
# Compute sum of absolute values and account for scale
sum(abs(miete$nmqm - nmqm2)) / sum(nmqm2)
```

```
## [1] 0.0002932959
```

There are only minor differences which are negligible and probably caused by rounding originally numeric values of `wfl` to integers. Since the year of construction, `bj`, contains values of years, we can convert it to integers using `as.integer`:

```r
miete$bj <- as.integer(miete$bj)
```

The factor variable `bez`, indicating the district where the respective flat is located, has 25 levels. Let's have a quick look on how many apartments there are per district calling the `table`-function:

```r
table(miete$bez)
```

```
##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  43 161 132 137 139  66  69  62 177  58  70  78  98  60  43 115  67  82
##  19  20  21  22  23  24  25
## 106  50  56  24  14  29 117
```

The remaining variables (`wohngut`, `wohnbest`, `ww0`, `zh0`, `badkach0`, `badextra`, `kueche`) are all binary with valid observations which we can see from the summary above, since `Min.` is 0 and `Max.` is 1 for all those variables. We choose to reformat them as factor variables with two levels, "Yes" and "No", for the purpose of convenient labeling (e.g. in plots) in our further analysis. This can be achieved by subsetting accordingly and applying the `as.factor`-function:

```r
# Y=1 and N=0 variables
miete[c(7,8,12,13)][miete[c(7,8,12,13)] == 1] <- "Yes"
miete[c(7,8,12,13)][miete[c(7,8,12,13)] == 0] <- "No"

# Y=0 and N=1 variables
miete[9:11][miete[9:11] == 1] <- "No"
miete[9:11][miete[9:11] == 0] <- "Yes"

# Convert to factor variables
miete[7:13] <- lapply(miete[7:13], as.factor)

# Remove the "0" in the names of the variables with Y=0 and N=1
names(miete)[9:11] <- c("ww", "zh", "badkach")
```

Now, we have a nice and tidy dataset and can proceed exploring our data.

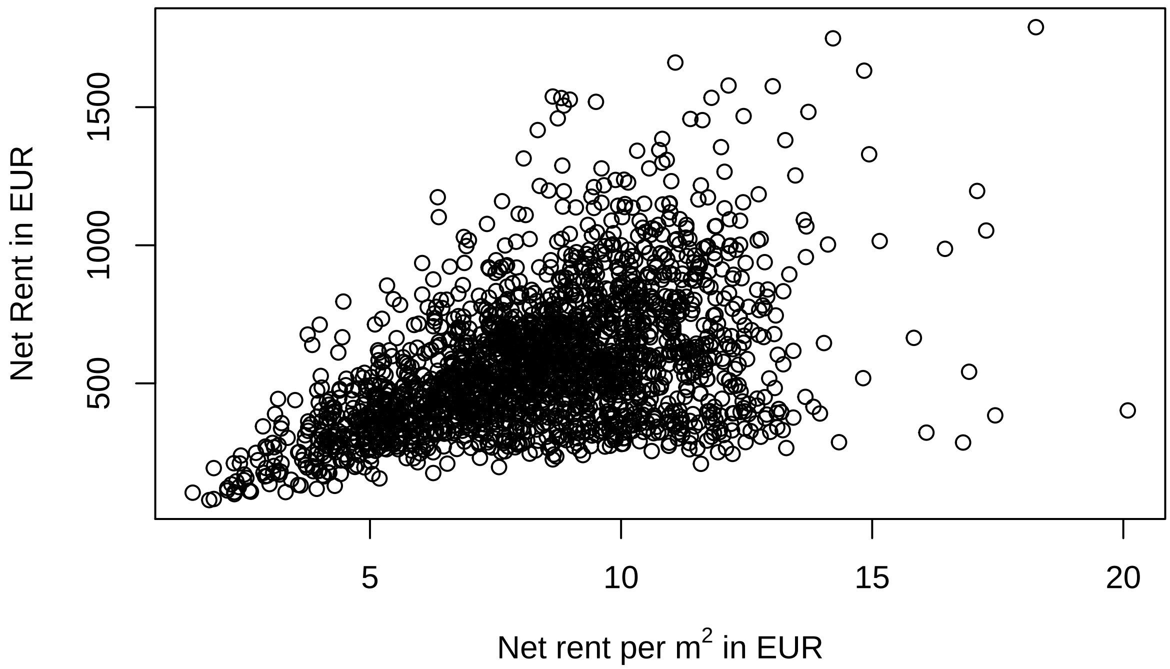
Exploratory Analysis

Before building a model, we would like to better understand our data by using different plots and methods of analysis.

The dependent variable of our model will be `nm`. Therefore, it would be nice to have a look at some scatterplots with different regressors to get a first impression on the correlation between the dependent variable and the potential regressors.

Net rent per m^2 (`nmqm`) is the net rent (`nm`) per living space (`wfl`) as we have already seen above. Therefore, it is not appropriate to use `nmqm` as an explanatory variable because we would use rent pricing information to explain rent pricing information. Since we have living space `wfl` as a separate variable, `nmqm` is of no additional explanatory value. Let's verify our reasoning with a scatterplot using the `plot`-function, where we expect `nm` to be highly positively correlated with `nmqm`:

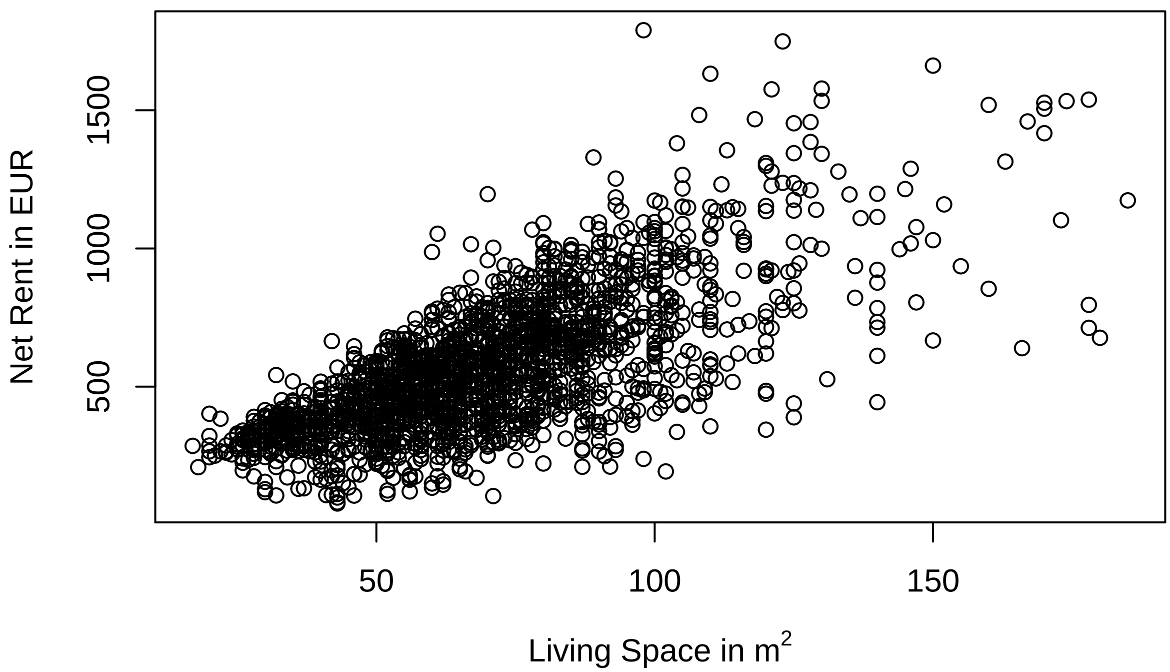
```r
plot(miete$nmqm,
     miete$nm,
     xlab = expression(paste("Net rent per m"^"2", " in EUR")),
     ylab = "Net Rent in EUR")
```



As expected, we can see a strong positive correlation.

Next, we will have a look at living space `wfl`. Naturally, one would assume prices to be higher for larger spaces. Let's have a look:

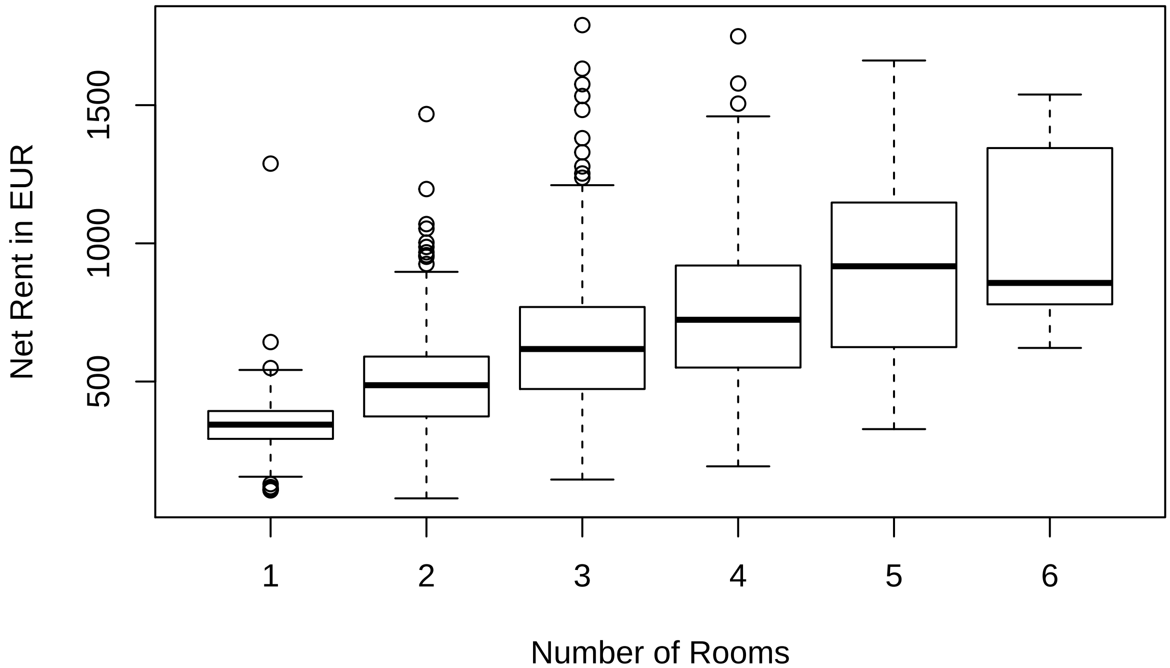
```
plot(miete$wfl,
     miete$nm,
     xlab = expression(paste("Living Space in m"^"2")),
     ylab = "Net Rent in EUR")
```



Indeed, there seems to be a positive correlation and therefore we expect living space to be a significant regressor later in our model.

A further potential regressor are the number of rooms (`rooms`) available in a flat. Number of rooms ranges from one to six rooms at most. Therefore, a `boxplot` is suitable to get a first impression on how net rent varies by number of rooms:

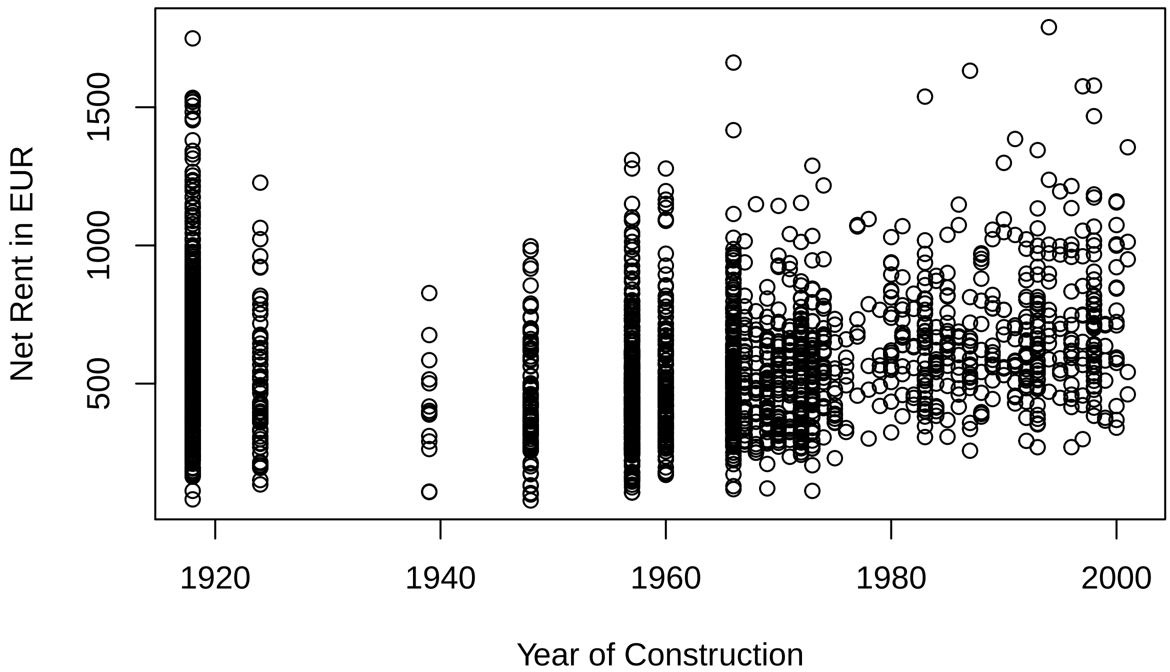
```
boxplot(nm ~ as.factor(rooms), data = miete,
        xlab = "Number of Rooms",
        ylab = "Net Rent in EUR")
```



From the boxplot, we can observe higher net rents for flats with more rooms (although from 5 to 6 rooms there doesn't seem to be a significant difference). But we have to be careful with our conclusion. Since more rooms most likely mean larger living space (or the other way round), this positive relationship in the plot could already be explained by `wfl`. For example, if people generally prefer more open rooms for some fixed living space, i.e. fewer rooms per space, and are willing to pay more for this kind of architecture, then there could even be a reducing effect of more rooms on renting prices, when pure living space has already explained a higher renting price level.

For the effect of the year of construction (`bj`) on net rents, we do not have a clear intuition, since very old but renovated buildings could also be of high value. Let's look at the scatterplot:

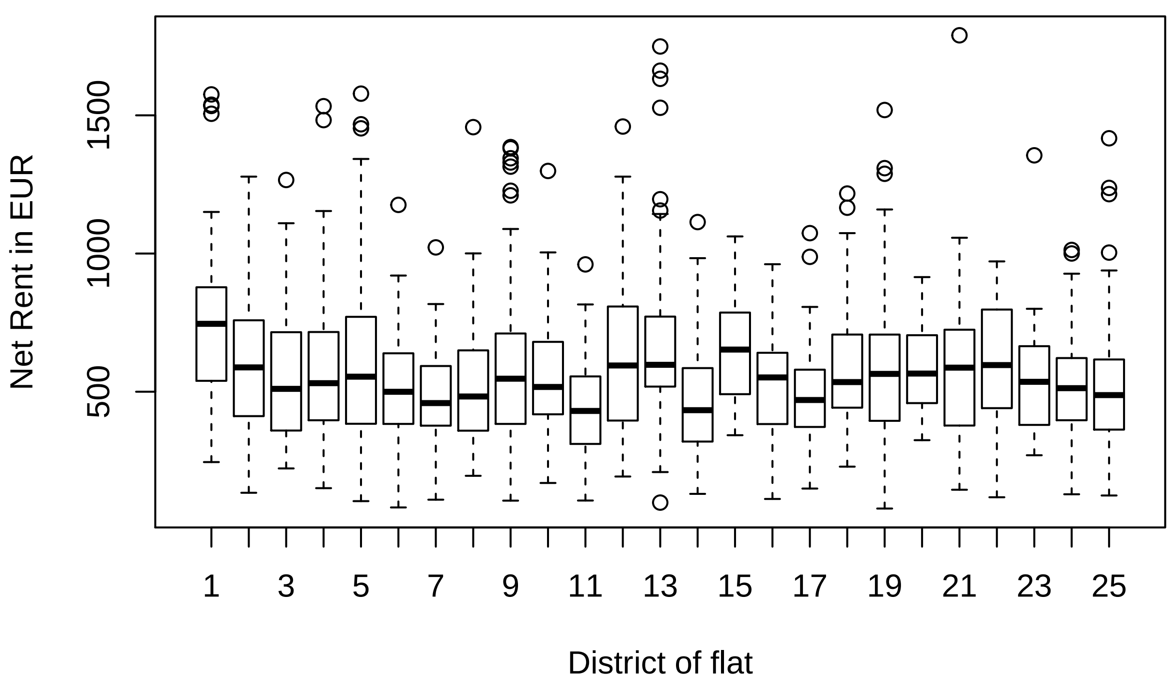
```
plot(miete$bj,
     miete$nm,
     xlab = "Year of Construction",
     ylab = "Net Rent in EUR")
```



From the plot, there seems to be a slightly positive relationship.

Another candidate for providing explanatory value on rent levels is the district, where the respective property is located (`bez`). Generally, one would expect higher rental levels in districts close to the center of Munich. Overall, the observations in our dataset are located in 25 different districts. A complete list of all districts of Munich can for example be found at en.wikipedia.org/wiki/Boroughs_of_Munich. Munich has 25 districts in total, i.e. the dataset contains flats from all districts. Let's consider a boxplot:

```
boxplot(nm ~ bez, data = miete,
        xlab = "District of flat",
        ylab = "Net Rent in EUR")
```



From the boxplot we can see, that rental prices in district 1 are relatively high. This is the “Altstadt-Lehel”-district, which is the center of Munich where the “Marienplatz” is also located. Since we will incorporate the factor variable `bez` as a dummy-variables in our linear regression, the “Altstadt-Lehel”-district will be our reference-district (i.e. zero encoded dummy). Hence, we expect from looking at the boxplot, that different districts will have a decreasing effect on rental prices when compared to the benchmark “Altstadt-Lehel”. For example, lower rental prices could be expected in district 11 (“Milbertshofen-Am Hart”) or district 14 (“Berg am Laim”).

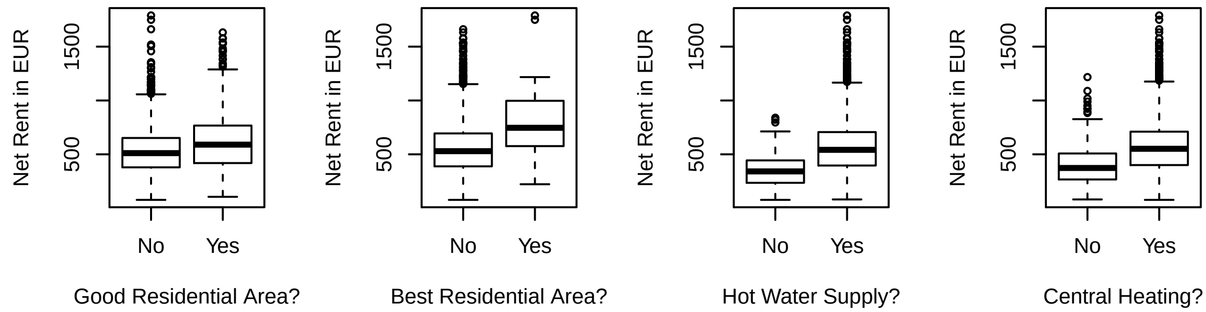
To complete our exploratory analysis, let’s consider a further plot, showing the boxplots of all binary variables:

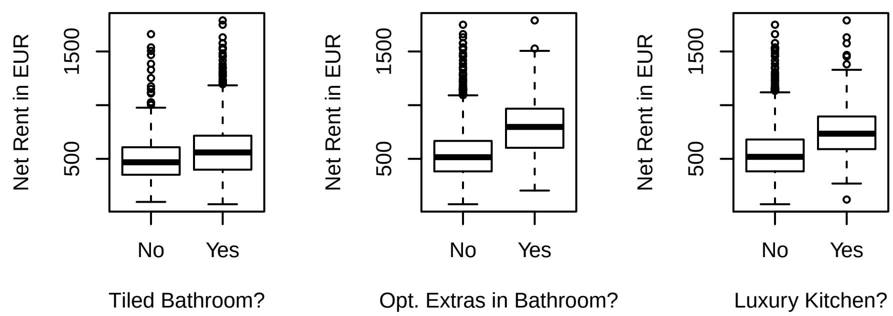
```r
# Prepare for multiple base plots
par(mfrow = c(2,4))

# Labels
nmLabel <- "Net Rent in EUR"
BinLabel <- c("Good Residential Area?",
            "Best Residential Area?",
            "Hot Water Supply?",
            "Central Heating?",
            "Tiled Bathroom?",
            "Opt. Extras in Bathroom?",
            "Luxury Kitchen?")

# Plot
for (i in 7:13){
    boxplot(formula(paste("nm ~ ", names(miete)[i])),
            data = miete,
            xlab = BinLabel[i-6],
            ylab = nmLabel)
}

# Reset to single base plot
par(mfrow = c(1,1))
```





From the boxplots it seems that each extra (i.e. an answer of "Yes" to each one of the questions) has an increasing effect on rental prices, since there are positive distribution shifts visible in every boxplot.

Model Specification

In a first step, let's create a naive linear regression model using all available regressors. The function to fit linear models in R is `lm` (*linear model*). The first argument of `lm` is the regression formula. In this naive case, we would like to do a regression of the rent in EUR (`nm`) on all other variables (we can use the `.` to include all variables). Left of the tilde is the dependent variable, right of the tilde the regressors. In the second argument we set our dataset. To get a nice summary of the linear model, we can use the `summary`-function:

```r
# Fitting the naive linear regression
lmNaive <- lm(nm ~ ., data = miete)
summary(lmNaive)
```

```
##
## Call:
## lm(formula = nm ~ ., data = miete)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -511.19  -19.26    7.26   27.60  328.92
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -933.61145  140.00413  -6.668 3.33e-11 ***
## nmqm          65.33069    0.71104  91.880  < 2e-16 ***
## wfl            8.29758    0.11435  72.563  < 2e-16 ***
## rooms          2.52110    2.82157   0.894  0.37169
## bj             0.16275    0.07243   2.247  0.02475 *
```

```
## bez2          16.40563   11.36719   1.443  0.14911
## bez3          19.32616   11.70560   1.651  0.09889 .
## bez4          13.71455   11.57493   1.185  0.23622
## bez5          13.97488   11.53420   1.212  0.22581
## bez6          19.08483   13.22698   1.443  0.14921
## bez7          15.46011   13.22919   1.169  0.24269
## bez8          25.69554   13.43483   1.913  0.05594 .
## bez9          24.60532   11.30832   2.176  0.02968 *
## bez10         16.46769   13.67104   1.205  0.22851
## bez11         27.77102   13.31002   2.086  0.03706 *
## bez12         19.89723   12.55427   1.585  0.11315
## bez13         24.86314   12.36727   2.010  0.04452 *
## bez14         26.41531   13.58446   1.945  0.05197 .
## bez15         21.80586   14.57358   1.496  0.13474
## bez16         24.94937   12.21742   2.042  0.04127 *
## bez17         22.44613   13.25114   1.694  0.09044 .
## bez18         14.62044   12.74381   1.147  0.25141
## bez19         25.02123   12.25559   2.042  0.04132 *
## bez20         15.82627   13.92585   1.136  0.25590
## bez21         30.21343   13.55179   2.229  0.02589 *
## bez22         27.75977   17.20238   1.614  0.10675
## bez23         20.26342   20.57135   0.985  0.32473
## bez24         29.69837   16.27283   1.825  0.06814 .
## bez25         26.62443   12.09441   2.201  0.02782 *
## wohngutYes    -3.52914    3.73042  -0.946  0.34424
## wohnbestYes   27.20112   10.44388   2.605  0.00927 **
## wwYes         45.99457    9.42129   4.882 1.13e-06 ***
## zhYes        -11.53547    6.44480  -1.790  0.07362 .
## badkachYes    -4.52387    3.84480  -1.177  0.23949
## badextraYes    7.25510    5.31839   1.364  0.17267
## kuecheYes     27.29273    5.84519   4.669 3.22e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 65.64 on 2017 degrees of freedom
## Multiple R-squared:  0.9297, Adjusted R-squared:  0.9285
## F-statistic:   762 on 35 and 2017 DF,  p-value: < 2.2e-16
```

We can see in the output that `nmqm` is a significant regressor with a very low p-value. Besides, the number of rooms (`rooms`), good residential area (`wohngut`), central heating (`zh`), a tiled bathroom (`badkach`) and optional extras in the bathroom (`badextra`) are not significant at the 5%-level in this naive model. Although we have significant regressors, a very high adjusted R^2 of 0.9285 and a very low p-value of the F-statistic, this model is fundamentally misspecified. As mentioned before, the variable `nmqm` is a transformation of our dependent variable `nm`. Therefore, if we include `nmqm` as a regressor, we would use (part of) the dependent variable to explain and estimate itself. In consequence, `nmqm` is neither appropriate for inference nor for prediction (one would have to know the price of an apartment in advance, before estimating the price). As a result, we will omit the variable `nmqm` in our model.

Fitting the Regression Model and Identification of relevant Regressors

Let's fit a linear regression model omitting `nmqm`. This can be achieved by `-nmqm` in the regression formula. Again, we take a look at the model using the `summary`-function:

```r
# Fitting the regression model omitting nmqm
lm1 <- lm(nm ~ .-nmqm, data = miete)
summary(lm1)
```

```
##
## Call:
## lm(formula = nm ~ . - nmqm, data = miete)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -602.82  -81.81   -4.47   85.74  737.55
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3075.4506   314.2815  -9.786  < 2e-16 ***
## wfl             7.4464     0.2595  28.699  < 2e-16 ***
## rooms         -21.0300     6.3970  -3.287 0.001028 **
## bj              1.5102     0.1615   9.352  < 2e-16 ***
## bez2          -29.3484    25.8535  -1.135 0.256434
## bez3          -31.0684    26.6195  -1.167 0.243296
## bez4          -50.6816    26.3029  -1.927 0.054140 .
## bez5          -32.8202    26.2329  -1.251 0.211041
## bez6          -59.3667    30.0495  -1.976 0.048333 *
## bez7         -110.3126    29.9557  -3.683 0.000237 ***
## bez8          -49.0576    30.5293  -1.607 0.108233
## bez9          -52.3311    25.6736  -2.038 0.041648 *
## bez10         -82.2557    31.0270  -2.651 0.008086 **
## bez11        -104.2274    30.1243  -3.460 0.000552 ***
## bez12         -35.4803    28.5479  -1.243 0.214072
## bez13         -51.9167    28.0907  -1.848 0.064722 .
## bez14        -112.1143    30.7351  -3.648 0.000271 ***
## bez15         -80.5210    33.0809  -2.434 0.015017 *
## bez16        -120.3236    27.5800  -4.363 1.35e-05 ***
## bez17         -89.3554    30.0398  -2.975 0.002969 **
## bez18         -59.1197    28.9547  -2.042 0.041301 *
## bez19         -87.0827    27.7622  -3.137 0.001733 **
## bez20         -96.8369    31.5802  -3.066 0.002195 **
## bez21         -77.3390    30.7364  -2.516 0.011940 *
## bez22        -122.2267    38.9859  -3.135 0.001742 **
## bez23        -117.4035    46.7080  -2.514 0.012029 *
## bez24        -111.2573    36.8814  -3.017 0.002588 **
## bez25         -97.3885    27.3619  -3.559 0.000380 ***
## wohngutYes     30.5535     8.4505   3.616 0.000307 ***
## wohnbestYes   127.6679    23.6457   5.399 7.48e-08 ***
## wwYes         168.6749    21.2318   7.944 3.21e-15 ***
## zhYes          74.1718    14.5176   5.109 3.54e-07 ***
## badkachYes     40.0843     8.6829   4.616 4.15e-06 ***
## badextraYes    55.9316    12.0475   4.643 3.66e-06 ***
## kuecheYes     113.5398    13.1343   8.644  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 149.4 on 2018 degrees of freedom
## Multiple R-squared:  0.6354, Adjusted R-squared:  0.6293
```

```
## F-statistic: 103.4 on 34 and 2018 DF,  p-value: < 2.2e-16
```

In the summary we can see, that *all* regressors (except some district-dummies) are significant at the 99%-level, which is indicated by two asterisks (`**`) or more. To judge the relevance of the factor variable `bez`, we can consider all factor levels at once by means of a global F-Test. To perform a global F-Test on each regressor, we use the `anova`-function:

```r
# F-Tests on variables
anova(lm1)
```

```
## Analysis of Variance Table
##
## Response: nm
##             Df   Sum Sq  Mean Sq   F value    Pr(>F)
## wfl          1 61866515 61866515 2770.4212 < 2.2e-16 ***
## rooms        1  1073786  1073786   48.0848 5.475e-12 ***
## bj           1  4663020  4663020  208.8130 < 2.2e-16 ***
## bez         24  2665912   111080    4.9742 3.877e-14 ***
## wohngut      1   313731   313731   14.0491 0.0001831 ***
## wohnbest     1   852106   852106   38.1578 7.870e-10 ***
## ww           1  3594484  3594484  160.9633 < 2.2e-16 ***
## zh           1   728476   728476   32.6216 1.286e-08 ***
## badkach      1   529139   529139   23.6952 1.216e-06 ***
## badextra     1   588528   588528   26.3547 3.114e-07 ***
## kueche       1  1668742  1668742   74.7273 < 2.2e-16 ***
## Residuals 2018 45064133    22331
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The F-Test confirms that `bez` is significant (at a greater than 99.9%-level) as well as all the other regressors. Hence, we already have identified all relevant regressors.

In general, R provides a very useful procedure to identify relevant regressors automatically. Imagine having a very large number of potential regressors, then identifying all relevant regressors step by step manually can become very tedious. Fortunately, there is a function called `step` which chooses a model, i.e. selects relevant regressors, by the Akaike information criterion (AIC) in a stepwise algorithm automatically. Since all regressors in our model are already significant, the `step`-function should not be able to improve the model, that is finding a new model with a lower AIC by including and excluding regressors:

```r
# Chosse a model by AIC in a stepwise algorithm
lm2 <- step(lm1)
```

```
## Start:  AIC=20592.9
## nm ~ (nmqm + wfl + rooms + bj + bez + wohngut + wohnbest + ww +
##     zh + badkach + badextra + kueche) - nmqm
##
##            Df Sum of Sq      RSS   AIC
## <none>                  45064133 20593
## - rooms     1    241345 45305478 20602
## - wohngut   1    291922 45356055 20604
## - bez      24   1469508 46533641 20611
## - badkach   1    475912 45540045 20612
## - badextra  1    481314 45545446 20613
## - zh        1    582905 45647038 20617
## - wohnbest  1    650984 45715117 20620
## - ww        1   1409406 46473538 20654
## - kueche    1   1668742 46732875 20666
```

```
## - bj        1   1953167 47017300 20678
## - wfl       1  18392030 63456163 21294
```

```r
summary(lm2)
```

```
##
## Call:
## lm(formula = nm ~ (nmqm + wfl + rooms + bj + bez + wohngut +
##     wohnbest + ww + zh + badkach + badextra + kueche) - nmqm,
##     data = miete)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -602.82  -81.81   -4.47   85.74  737.55
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3075.4506   314.2815  -9.786  < 2e-16 ***
## wfl             7.4464     0.2595  28.699  < 2e-16 ***
## rooms         -21.0300     6.3970  -3.287 0.001028 **
## bj              1.5102     0.1615   9.352  < 2e-16 ***
## bez2          -29.3484    25.8535  -1.135 0.256434
## bez3          -31.0684    26.6195  -1.167 0.243296
## bez4          -50.6816    26.3029  -1.927 0.054140 .
## bez5          -32.8202    26.2329  -1.251 0.211041
## bez6          -59.3667    30.0495  -1.976 0.048333 *
## bez7         -110.3126    29.9557  -3.683 0.000237 ***
## bez8          -49.0576    30.5293  -1.607 0.108233
## bez9          -52.3311    25.6736  -2.038 0.041648 *
## bez10         -82.2557    31.0270  -2.651 0.008086 **
## bez11        -104.2274    30.1243  -3.460 0.000552 ***
## bez12         -35.4803    28.5479  -1.243 0.214072
## bez13         -51.9167    28.0907  -1.848 0.064722 .
## bez14        -112.1143    30.7351  -3.648 0.000271 ***
## bez15         -80.5210    33.0809  -2.434 0.015017 *
## bez16        -120.3236    27.5800  -4.363 1.35e-05 ***
## bez17         -89.3554    30.0398  -2.975 0.002969 **
## bez18         -59.1197    28.9547  -2.042 0.041301 *
## bez19         -87.0827    27.7622  -3.137 0.001733 **
## bez20         -96.8369    31.5802  -3.066 0.002195 **
## bez21         -77.3390    30.7364  -2.516 0.011940 *
## bez22        -122.2267    38.9859  -3.135 0.001742 **
## bez23        -117.4035    46.7080  -2.514 0.012029 *
## bez24        -111.2573    36.8814  -3.017 0.002588 **
## bez25         -97.3885    27.3619  -3.559 0.000380 ***
## wohngutYes     30.5535     8.4505   3.616 0.000307 ***
## wohnbestYes   127.6679    23.6457   5.399 7.48e-08 ***
## wwYes         168.6749    21.2318   7.944 3.21e-15 ***
## zhYes          74.1718    14.5176   5.109 3.54e-07 ***
## badkachYes     40.0843     8.6829   4.616 4.15e-06 ***
## badextraYes    55.9316    12.0475   4.643 3.66e-06 ***
## kuecheYes     113.5398    13.1343   8.644  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 149.4 on 2018 degrees of freedom
## Multiple R-squared:  0.6354, Adjusted R-squared:  0.6293
## F-statistic: 103.4 on 34 and 2018 DF,  p-value: < 2.2e-16
```

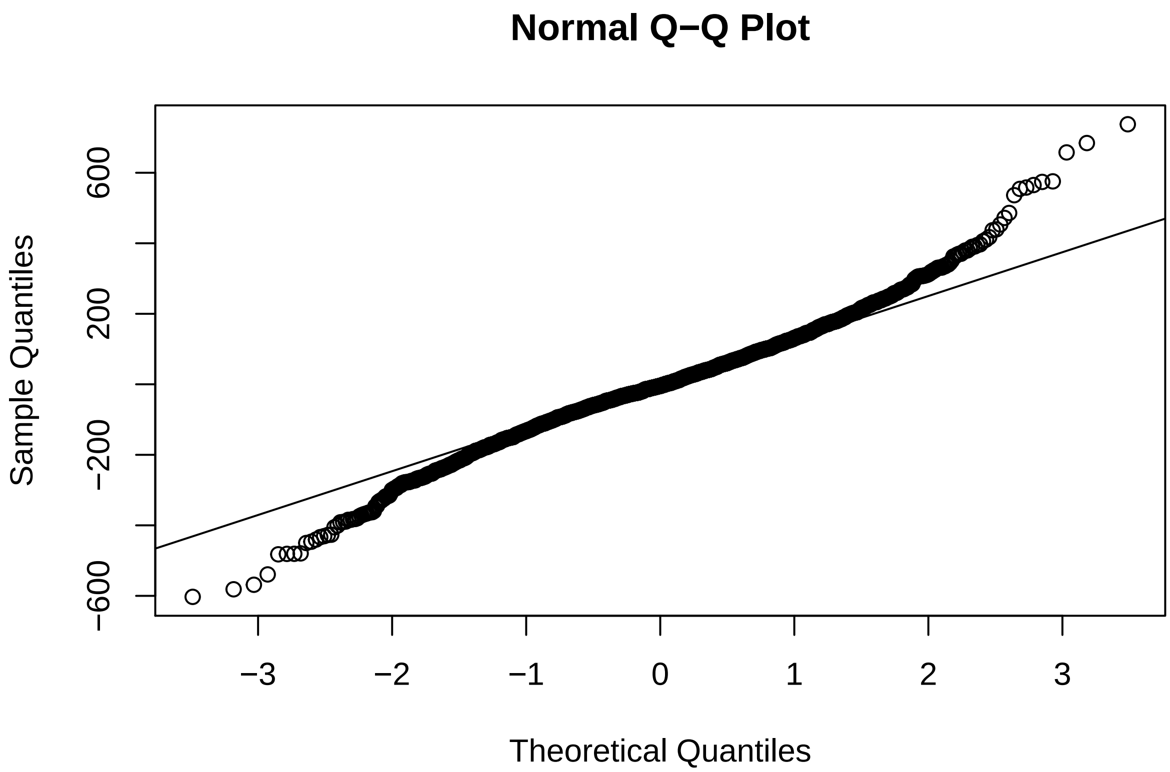
As expected, our model remains unchanged when applying the `step`-function.

Discussion of Model Fit

As we have already seen, all regressors in our model are significant. Our model has a R^2 of 0.6354 and an adjusted R^2 of 0.6293.

To further evaluate the goodness of fit of our model, we check the distributional assumption of the linear model that errors are normally distributed with mean zero. We do this via a QQ-plot. In R, this can be done with `qqnorm` and `qqline`:

```r
# Build a Q-Q Plot
qqnorm(lm1$residuals)
qqline(lm1$residuals)
```



We can see that the residuals follow a symmetric distribution with mean zero which is similar to the normal but has fatter tails since the sample quantiles are smaller for negative and greater for positive values. To additionally test the normality assumption of the errors, we can perform a Shapiro-Wilk test:

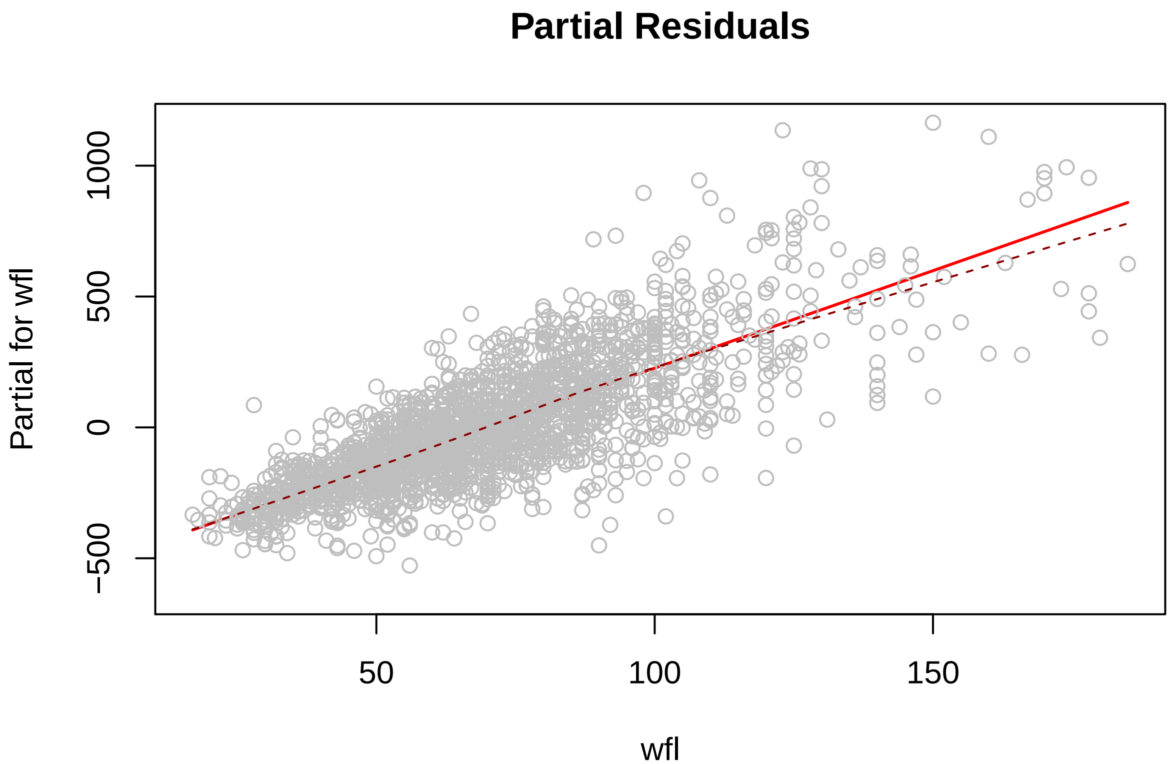
```r
# Perform Shapiro-Wilk test
shapiro.test(lm1$residuals)
```

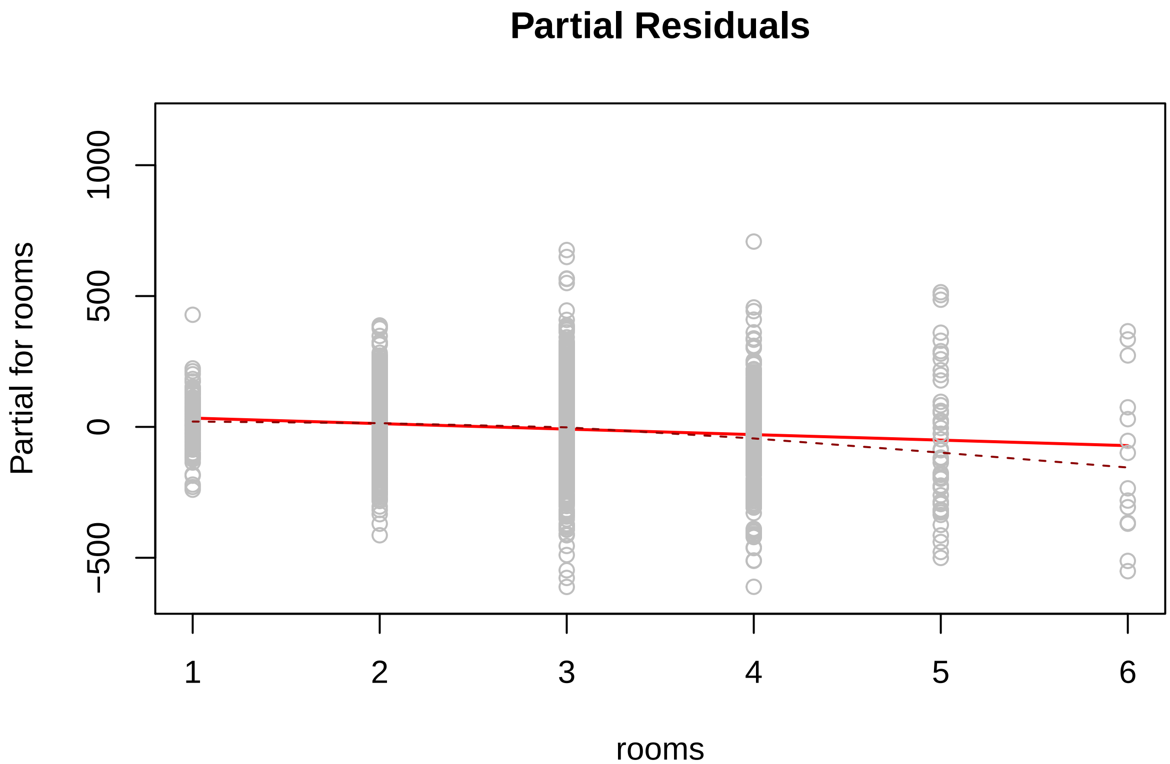
```
##
##  Shapiro-Wilk normality test
##
## data:  lm1$residuals
## W = 0.98289, p-value = 5.588e-15
```

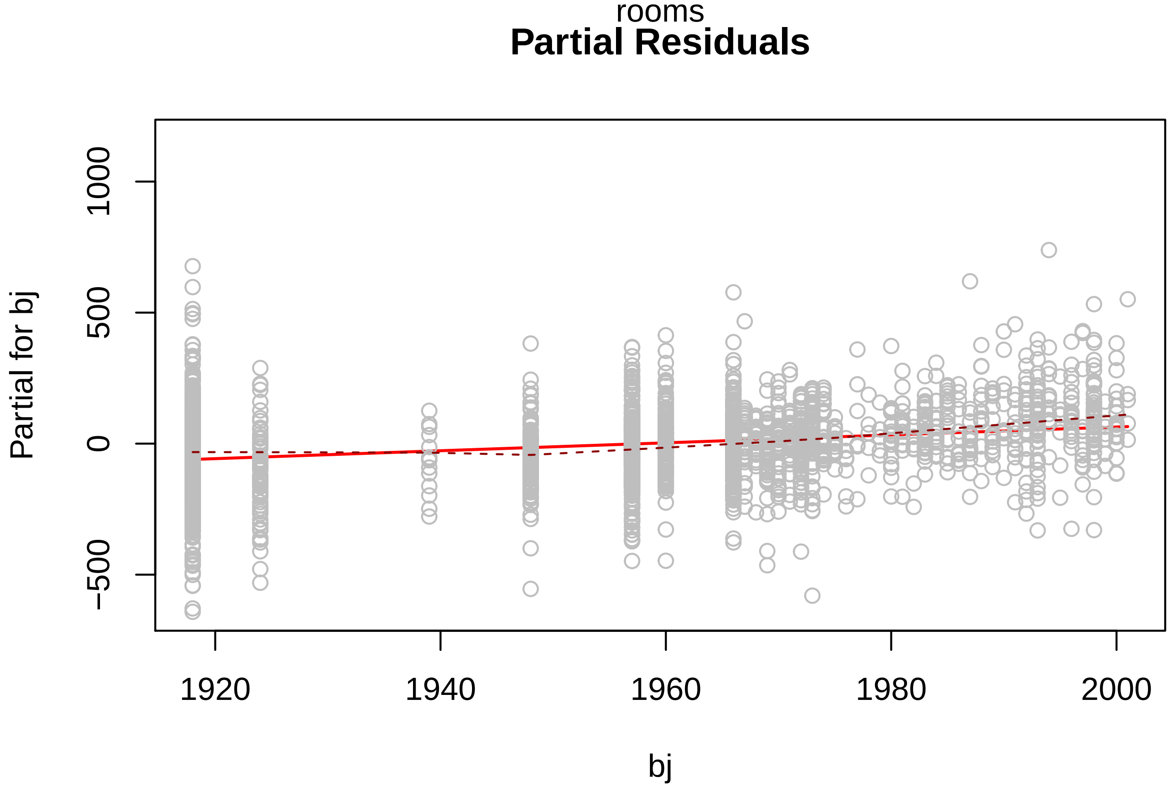
Hence, we must reject the null hypothesis of normality with a p-value of $5.5880102 \times 10^{-15}$, i.e. there is strong evidence, that the assumption of normally distributed errors is false.

Additionally, although we are considering a linear regression model, we have to think and account for non-linear effects. We can check for non-linear relationships between the dependent variable and individual (non-categorical) regressors by looking at the partial residuals of the individual regressors. R also has a function implemented to conveniently create plots of the partial residuals. We can create them using `termplot`:

```
# Plot partial residuals of wfl, rooms and bj using termplot
termplot(model = lm1,
         partial.resid = TRUE,
         terms = c("wfl", "rooms", "bj"),
         main = "Partial Residuals",
         smooth = panel.smooth)
```





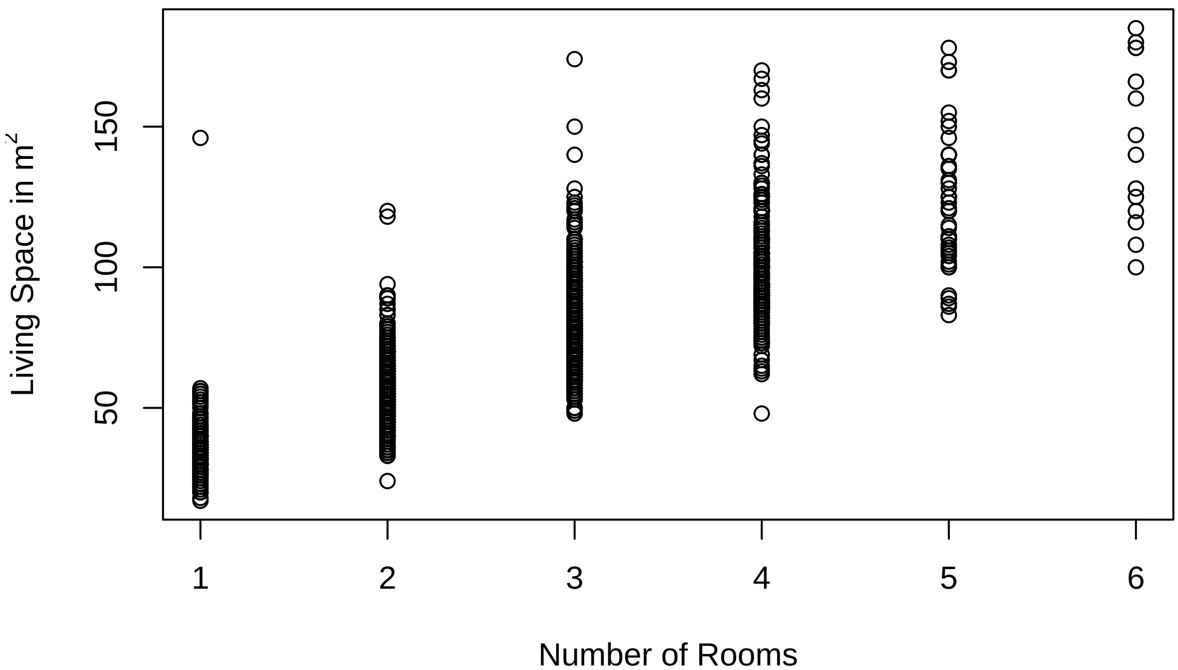


From the plots, we can see that there does not seem to be a clear non-linear effect or relationship.

Model Interpretation

To conclude our analysis, we would like to give an interpretation of our model. The net rent increases by 7.45 EUR per m^2 of living space. Although net rent and the number of rooms are positively correlated, as we have seen in our exploratory analysis, the coefficient of `rooms` is negative, that is net rent decreases by 21.03 EUR per additional room. This statistical phenomena is called the Simpson's paradox and in our case is explained by the positive correlation between the number of rooms and living space:

```r
plot(miete$rooms,
     miete$wfl,
     xlab = "Number of Rooms",
     ylab = expression(paste("Living Space in m"^"2")))
```



Hence, after controlling for living space, rental prices decrease with additional number of rooms, i.e. tenants could have a preference for fewer, larger rooms. The coefficient for the year of construction is 1.51 EUR. Therefore, a 50 year old difference in year of construction would make a difference of about 75 EUR in our model, where the older apartment would be the cheaper one. Considering the effect of different districts, we have to interpret the statistically significant coefficients in relation to the reference district ("Altstadt-Lehel"). For example, our model predicts that renting an apartment in "Sendling-Westpark" (district 7) will be 110.31 EUR cheaper compared to "Altstadt-Lehel". Other predicted savings compared to the center of Munich are 104.23 EUR in "Milbertshofen-Am Hart" (district 11) and 120.32 EUR in "Ramersdorf-Perlach" (district 16). Finally, we have statistically significant rental price increasing effects for all additional extras. The highest increase in net rent of 168.67 EUR is for hot water supply. Living in an apartment with a tiled bathroom that has fancy extras will cost an additional 96.01 EUR according to our model.

In the end of our interpretation, we would like to remind that the interpretation has to be taken with a grain of salt, since, as we have seen above, the assumption of normally distributed errors is very likely violated for our data.