Report

Mirco Bazzani

12/7/2020

## Importing the dataset

Let’s first take a look at the **dimensions** of the dataset - dim(training).

This would be every data analysts dream, but most of the 100 total columns mainly consist of missing values. This is why are going to reduce the dataset and only keep the ones wit a reasonable amount of missingness.

We split the data into 3 datasets:

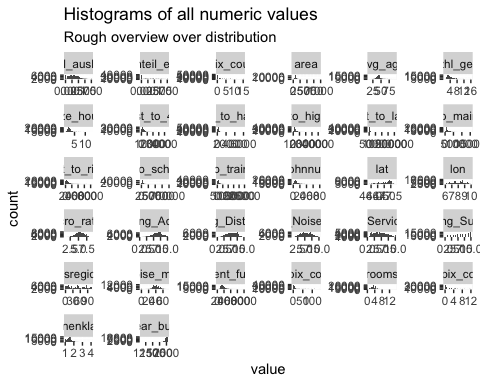
* One includes both micro ratings and descriptive variables about the surrounding area. k = 41 variables.
* One only contains the micro ratings and data about the flat. k = 22 variables.
* One without the micro ratings in order to surpass possible covariance in linear regression.

### Basic Exploration

Let’s look at the distribution of our variables. The first plot only shows numeric values

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 164691 rows containing non-finite values (stat\_bin).

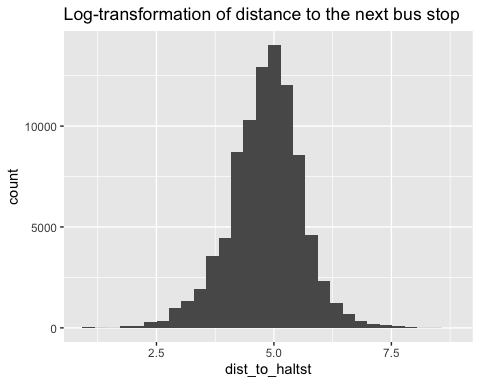


Some of the numeric variables. We can easily spot that most of them are not evenly distributet, and have either a right- or a left skew.

This could be fixed by a log-transformation. One vairable that is severely skewed is the distance to the next bus stop. Let’s log transform it:

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 646 rows containing non-finite values (stat\_bin).



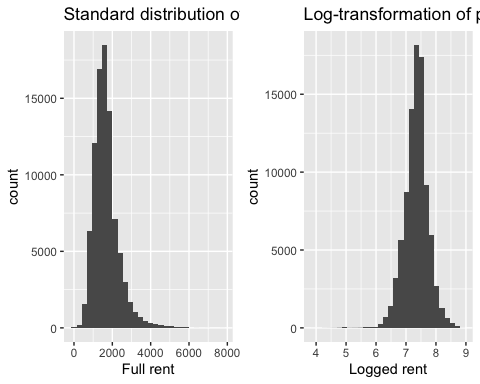
The **micro-ratings** follow an approximate normal distribution, and are therefore ok. We could use them as regressors without any hassle.

But most of the numeric values are right skewed, this includes: area “Avg\_size\_household”, “dist\_to\_haltst”, “dist\_to\_highway”, “dist\_to\_lake”, “dist\_to\_main\_stat”, “dist\_to\_river”, “dist\_to\_school\_1”, “dist\_to\_train\_stat”

A log-transformation could come in handy on order to improve the regression model.

The same counts for the dependent variable:

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



The log-distribution removed most of the swekedness in the dependent variable as well, so we’ll compare a logged regression to non-logged values afterwards.

Next, we are going to normalize our data and scaling it so that every numeric value (expect the rent) ranges from 0 to 1.

library(BBmisc)

##   
## Attaching package: 'BBmisc'

## The following objects are masked from 'package:dplyr':  
##   
## coalesce, collapse

## The following object is masked from 'package:base':  
##   
## isFALSE

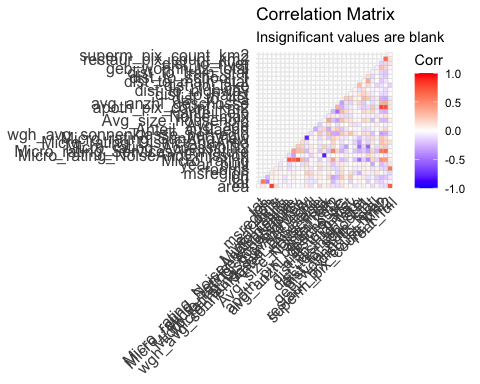
training\_log\_norm <- normalize(training\_logged[,-1], method = c("range"),range = c(0, 1))  
training\_log\_norm$rent\_full <- training\_logged$rent\_full  
  
training\_l\_norm <- normalize(training\_reduced\_large[,-1], method = c("range"),range = c(0, 1))  
training\_l\_norm$rent\_full <- training\_reduced\_large$rent\_full

Let’s check if it worked.

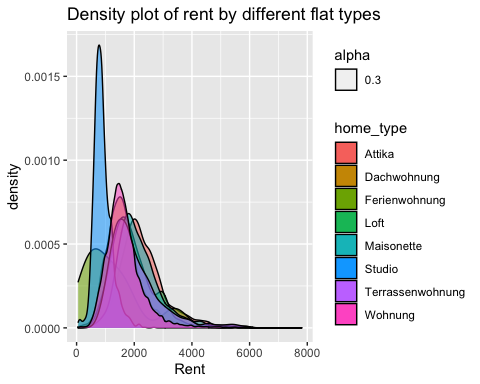
The new max in rent: max(training\_l\_norm$rent\_full) The new max in area: max(training\_l\_norm$msregion)

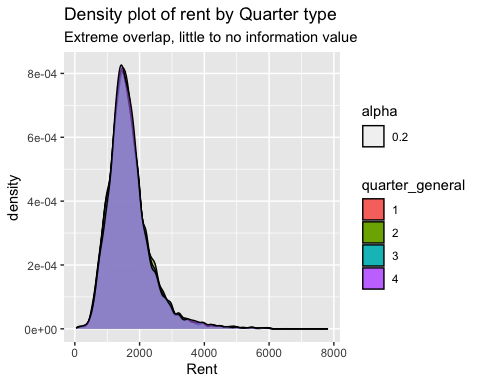
Yes, the normalization was successful.

Lastly, we should look at the correlation plot of our numerical values.

 What we see here is some minimal correlation between the regressors, but nearly no correlation between regressors and the rent.

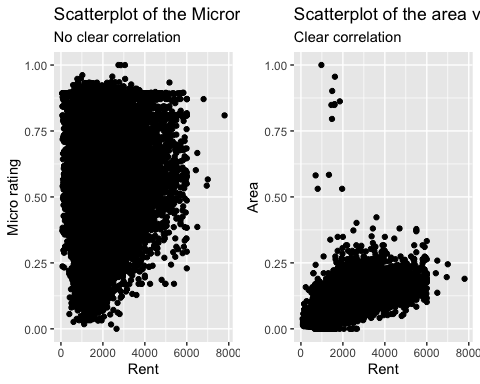
It looks better when we group the regressors by ther **flat type**; we should get good predictions by using the home type.

 Nearly no information can be gathered by using the **Quarter type** measure.



Lastly, we want to look at how the micro rating and the area correlate with our rent.

## Warning: Removed 17838 rows containing missing values (geom\_point).



What we can take away from this is that area and Home\_type need to be used as regressors.

## Linear Regresssion

The first method we want to try out is a multiple linear regression. We are going to compare the logged dataset and the non-logged without any feature engineering just to get an overview over what predicts better.

Both of the datasets are going to be cross validated, using a **80/20** split and then comparing the **Rooted Square Mean Errors**

As our model can’t handle missing values, we are first just working with, which significantly reduces our dataset, but should work as a baseline.

The remaining dimensions are:

dim(na.omit(training\_l\_norm))

Because we work with factors that have many possible outcome values like the canton and floors we are going to leave them out in this first try.

## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.  
## Convert to a vector.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_warnings()` to see where this warning was generated.

## Warning in predict.lm(linear\_fit, X\_test): prediction from a rank-deficient fit  
## may be misleading  
  
## Warning in predict.lm(linear\_fit, X\_test): prediction from a rank-deficient fit  
## may be misleading

RMSE\_logged

RMSE\_standard

450.4175

465.9124

The results are a **RMSE** or RMSE\_lm for the linear model without logged values, and a **RMSE** of RMSE\_lm\_log for the logged model.

The model fit for the logged values looks as follows:

plot(linear\_fit)

From the Normal Q-Q we can assess that the model overpredicts cheap flats and underpredicts the most expensive ones.

The next step we are going to take is just inserting those columns that do not contain any missing values into the model in order to keep the full set of rows.

set.seed(123)  
  
no\_na <- training\_l\_norm %>%  
 select\_if(~ !any(is.na(.)))  
  
index <- createDataPartition(no\_na$rent\_full, p=0.8, list=FALSE)  
  
X <- no\_na  
X\_train <- X[index,]  
X\_test <- X[-index, 1:25]  
Y\_test<- X[-index,26]  
  
linear\_fit <- lm(formula = rent\_full ~., data = X\_train)  
  
linear\_predict <- predict(linear\_fit,X\_test)

## Warning in predict.lm(linear\_fit, X\_test): prediction from a rank-deficient fit  
## may be misleading

predictions\_1 <- data.frame(linear\_predict,Y\_test$rent\_full)  
  
RMSE\_lm\_nona <- RMSE(predictions\_1$linear\_predict,predictions\_1$Y\_test.rent\_full)  
  
#####  
  
no\_na\_log <- training\_log\_norm %>%  
 select\_if(~ !any(is.na(.)))  
  
index <- createDataPartition(no\_na\_log$rent\_full, p=0.8, list=FALSE)  
  
  
X <- no\_na\_log  
X\_train <- X[index,]  
X\_test <- X[-index, 1:25]  
Y\_test<- X[-index,26]  
  
linear\_fit <- lm(formula = rent\_full ~., data = X\_train)  
  
linear\_predict <- predict(linear\_fit,X\_test)

## Warning in predict.lm(linear\_fit, X\_test): prediction from a rank-deficient fit  
## may be misleading

predictions\_1 <- data.frame(exp(linear\_predict),exp(Y\_test$rent\_full))  
  
RMSE\_lm\_nona\_log <- RMSE(predictions\_1$exp.linear\_predict.,predictions\_1$exp.Y\_test.rent\_full.)  
  
kable(data.frame("RMSE\_logged" = RMSE\_lm\_nona\_log, "RMSE\_standard" = RMSE\_lm\_nona))

RMSE\_logged

RMSE\_standard

590.1544

575.0965

The dimensions here are dim(no\_na). But as we see from the new RSMEs, this approach kicked out most of the important columns, which left us with a significant decrease of prediction power.

In this scenario, the standard values predict better, but this is due to model fluctuations; both the logged and the standard values are sometimes better, sometimes worse.

### Imputation

Our base dataset consists of way to many missing values. This why we are going to impute those NAs using Multiple Imputation

set.seed(123)  
  
index <- createDataPartition(complete\_set$rent\_full, p=0.8, list=FALSE)  
  
X <- complete\_set  
X\_train <- X[index,]  
X\_test <- X[-index, c(1:38,40:41)]  
Y\_test<- X[-index,39]  
  
linear\_fit <- lm(formula = log(rent\_full) ~ . + rooms \* area, data = X\_train)  
  
summary(linear\_fit)

##   
## Call:  
## lm(formula = log(rent\_full) ~ . + rooms \* area, data = X\_train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -5.3210 -0.1090 0.0056 0.1216 1.3447   
##   
## Coefficients: (1 not defined because of singularities)  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 6.763e+00 3.281e-02 206.118 < 2e-16 \*\*\*  
## area 7.068e+00 5.942e-02 118.961 < 2e-16 \*\*\*  
## lat -5.320e-01 1.623e-02 -32.771 < 2e-16 \*\*\*  
## lon -3.202e-01 2.140e-02 -14.960 < 2e-16 \*\*\*  
## balconyTRUE 1.410e-02 2.118e-03 6.660 2.76e-11 \*\*\*  
## cabletvTRUE -5.239e-03 2.085e-03 -2.513 0.011987 \*   
## elevatorTRUE 7.891e-02 2.146e-03 36.768 < 2e-16 \*\*\*  
## floors2 3.105e-02 9.852e-03 3.152 0.001622 \*\*   
## floors3 2.572e-02 9.863e-03 2.608 0.009106 \*\*   
## floors4 3.057e-02 9.952e-03 3.071 0.002131 \*\*   
## floors5 4.369e-02 1.021e-02 4.280 1.87e-05 \*\*\*  
## floors6 5.558e-02 1.079e-02 5.153 2.57e-07 \*\*\*  
## floors7 2.849e-02 1.188e-02 2.399 0.016452 \*   
## floors8 3.709e-02 1.343e-02 2.761 0.005764 \*\*   
## floors9 4.708e-02 1.721e-02 2.736 0.006222 \*\*   
## floors10 7.933e-03 1.951e-02 0.407 0.684295   
## floors11 -4.902e-02 2.368e-02 -2.070 0.038459 \*   
## floors12 -8.768e-03 2.729e-02 -0.321 0.747945   
## floors13 7.880e-02 2.717e-02 2.900 0.003738 \*\*   
## floors14 3.449e-02 3.111e-02 1.108 0.267672   
## floors15 9.517e-02 4.287e-02 2.220 0.026414 \*   
## floors16 1.596e-01 5.451e-02 2.928 0.003413 \*\*   
## floors17 6.445e-02 5.052e-02 1.276 0.202079   
## floors18 2.545e-02 5.300e-02 0.480 0.631048   
## floors19 1.050e-01 7.868e-02 1.335 0.182016   
## floors20 3.305e-01 7.424e-02 4.451 8.54e-06 \*\*\*  
## floors21 2.907e-01 9.050e-02 3.212 0.001318 \*\*   
## floors22 1.546e-01 8.392e-02 1.842 0.065444 .   
## floors23 2.351e-02 7.869e-02 0.299 0.765090   
## floors24 1.966e-01 1.562e-01 1.259 0.208059   
## floors25 4.444e-01 1.278e-01 3.477 0.000508 \*\*\*  
## floors26 4.203e-01 2.207e-01 1.905 0.056794 .   
## floors27 4.215e-01 2.206e-01 1.910 0.056088 .   
## msregion -2.075e-01 1.456e-02 -14.255 < 2e-16 \*\*\*  
## parking\_indoorTRUE 1.814e-02 2.294e-03 7.905 2.73e-15 \*\*\*  
## parking\_outsideTRUE -2.018e-02 2.122e-03 -9.509 < 2e-16 \*\*\*  
## quarter\_general2 -1.394e-04 2.138e-03 -0.065 0.948013   
## quarter\_general3 1.814e-03 2.173e-03 0.835 0.403848   
## quarter\_general4 6.863e-03 2.683e-03 2.558 0.010541 \*   
## rooms 2.641e+00 2.365e-02 111.653 < 2e-16 \*\*\*  
## year\_built 2.683e-01 1.427e-02 18.805 < 2e-16 \*\*\*  
## Micro\_rating -3.704e+00 2.259e-01 -16.393 < 2e-16 \*\*\*  
## Micro\_rating\_NoiseAndEmission 3.941e-01 2.491e-02 15.824 < 2e-16 \*\*\*  
## Micro\_rating\_ServicesAndNature 1.109e+00 6.435e-02 17.242 < 2e-16 \*\*\*  
## Micro\_rating\_Accessibility 1.152e+00 7.611e-02 15.138 < 2e-16 \*\*\*  
## Micro\_rating\_DistrictAndArea 1.863e+00 1.041e-01 17.905 < 2e-16 \*\*\*  
## Micro\_rating\_SunAndView NA NA NA NA   
## wgh\_avg\_sonnenklasse\_per\_egid -2.958e-01 2.053e-02 -14.407 < 2e-16 \*\*\*  
## Anteil\_auslaend -1.944e-02 5.295e-03 -3.671 0.000242 \*\*\*  
## Avg\_age 5.475e-02 1.086e-02 5.041 4.64e-07 \*\*\*  
## Avg\_size\_household 6.734e-02 1.072e-02 6.282 3.36e-10 \*\*\*  
## Noise\_max 3.282e-02 1.208e-02 2.716 0.006604 \*\*   
## anteil\_efh 1.730e-03 3.876e-03 0.446 0.655340   
## apoth\_pix\_count\_km2 2.425e-02 1.058e-02 2.292 0.021890 \*   
## avg\_anzhl\_geschosse 5.976e-02 1.115e-02 5.358 8.43e-08 \*\*\*  
## dist\_to\_4G -1.236e-01 1.238e-02 -9.983 < 2e-16 \*\*\*  
## dist\_to\_haltst -1.843e-01 3.232e-02 -5.703 1.18e-08 \*\*\*  
## dist\_to\_highway -1.702e-01 1.245e-02 -13.671 < 2e-16 \*\*\*  
## dist\_to\_lake -1.722e-01 2.549e-02 -6.755 1.44e-11 \*\*\*  
## dist\_to\_main\_stat 1.416e-01 1.329e-02 10.651 < 2e-16 \*\*\*  
## dist\_to\_school\_1 -2.616e-02 2.348e-02 -1.114 0.265182   
## dist\_to\_train\_stat -1.346e-01 2.399e-02 -5.610 2.03e-08 \*\*\*  
## geb\_wohnnutz\_total 1.036e-01 1.196e-02 8.660 < 2e-16 \*\*\*  
## dist\_to\_river 8.467e-02 7.845e-03 10.792 < 2e-16 \*\*\*  
## restaur\_pix\_count\_km2 1.043e-01 1.038e-02 10.051 < 2e-16 \*\*\*  
## superm\_pix\_count\_km2 -3.516e-02 8.678e-03 -4.052 5.09e-05 \*\*\*  
## KTKZAI 3.933e-03 2.802e-02 0.140 0.888381   
## KTKZAR -9.534e-02 1.306e-02 -7.302 2.86e-13 \*\*\*  
## KTKZBE -3.046e-01 1.006e-02 -30.275 < 2e-16 \*\*\*  
## KTKZBL 2.454e-02 7.197e-03 3.410 0.000651 \*\*\*  
## KTKZBS 1.556e-02 7.564e-03 2.057 0.039693 \*   
## KTKZFR -3.187e-01 1.001e-02 -31.824 < 2e-16 \*\*\*  
## KTKZGE -1.265e-01 1.631e-02 -7.757 8.84e-15 \*\*\*  
## KTKZGL -2.891e-01 1.699e-02 -17.017 < 2e-16 \*\*\*  
## KTKZGR -5.817e-02 1.100e-02 -5.289 1.23e-07 \*\*\*  
## KTKZJU -2.894e-01 1.230e-02 -23.533 < 2e-16 \*\*\*  
## KTKZLU -1.580e-01 7.971e-03 -19.819 < 2e-16 \*\*\*  
## KTKZNE -3.185e-01 1.058e-02 -30.102 < 2e-16 \*\*\*  
## KTKZNW -1.273e-01 1.613e-02 -7.890 3.06e-15 \*\*\*  
## KTKZOW -1.874e-01 2.064e-02 -9.079 < 2e-16 \*\*\*  
## KTKZSG -6.856e-02 7.737e-03 -8.861 < 2e-16 \*\*\*  
## KTKZSH -4.244e-02 9.007e-03 -4.712 2.46e-06 \*\*\*  
## KTKZSO -2.258e-01 7.557e-03 -29.886 < 2e-16 \*\*\*  
## KTKZSZ -3.385e-02 9.768e-03 -3.466 0.000530 \*\*\*  
## KTKZTG 3.764e-02 7.377e-03 5.102 3.38e-07 \*\*\*  
## KTKZTI -3.269e-01 1.351e-02 -24.187 < 2e-16 \*\*\*  
## KTKZUR -3.314e-01 1.937e-02 -17.111 < 2e-16 \*\*\*  
## KTKZVD -1.575e-01 1.154e-02 -13.655 < 2e-16 \*\*\*  
## KTKZVS -4.193e-01 1.267e-02 -33.097 < 2e-16 \*\*\*  
## KTKZZG 1.788e-01 1.008e-02 17.741 < 2e-16 \*\*\*  
## KTKZZH 8.612e-02 1.000e-02 8.612 < 2e-16 \*\*\*  
## home\_typeDachwohnung -1.288e-01 6.856e-03 -18.780 < 2e-16 \*\*\*  
## home\_typeFerienwohnung -2.501e-01 9.017e-02 -2.774 0.005541 \*\*   
## home\_typeLoft -3.378e-02 1.265e-02 -2.670 0.007593 \*\*   
## home\_typeMaisonette -9.955e-02 7.014e-03 -14.194 < 2e-16 \*\*\*  
## home\_typeStudio -2.136e-01 8.261e-03 -25.856 < 2e-16 \*\*\*  
## home\_typeTerrassenwohnung -4.105e-02 9.997e-03 -4.106 4.02e-05 \*\*\*  
## home\_typeWohnung -1.403e-01 5.528e-03 -25.378 < 2e-16 \*\*\*  
## area:rooms -1.332e+01 2.143e-01 -62.186 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.2202 on 71904 degrees of freedom  
## Multiple R-squared: 0.6872, Adjusted R-squared: 0.6867   
## F-statistic: 1628 on 97 and 71904 DF, p-value: < 2.2e-16

linear\_predict <- predict(linear\_fit,X\_test)

## Warning in predict.lm(linear\_fit, X\_test): prediction from a rank-deficient fit  
## may be misleading

predictions\_1 <- data.frame(exp(linear\_predict),Y\_test)  
  
RMSE\_lm\_imp <- RMSE(predictions\_1$exp.linear\_predict.,predictions\_1$Y\_test)

The imputation was absolutely horrible, with a RMSE of RMSE\_lm\_imp. So imputing the whole dataset does not do the trick ; let’s try something different.

Why not choose those columns that don’t have missing values like the Micro Ratings and only impute significant rows like the area and Rooms.

This version already delivered better results with a RMSE of RMSE\_lm\_imp and Regression Coefficients of summary(linear\_fit).

I think this is the optimum we can get out of linear regression models.

## Ridge and Lasso

Building on the imputed dataset we gathered in the last computation, we teach a Ridge and Lasso model.

You see the parameters of the trained models below; the method used was a resampling method that searches for the best lambda-value.

kable(RMSE)

Ridge\_RMSE

Lasso\_RMSE

Linear\_RMSE

434.6313

434.6313

434.6826

kable(R2)

Ridge\_R2

Lasso\_R2

Linear\_R2

0.5778019

0.5778019

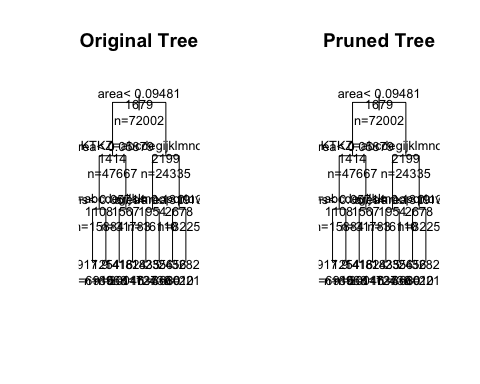
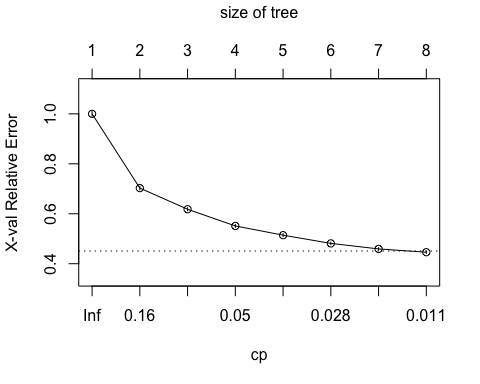
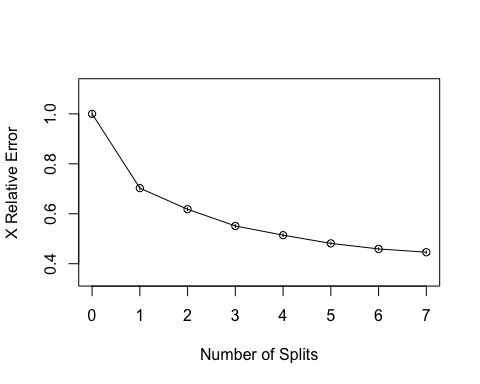
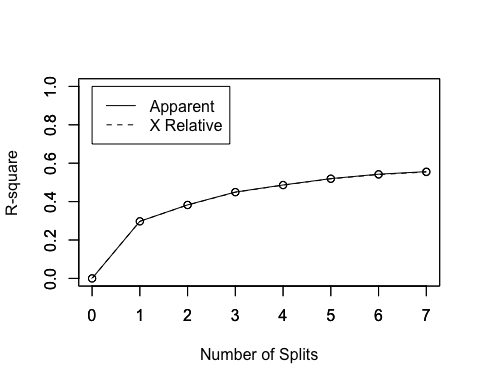
0.5778067

All of the three models come to similar results using the imputed dataset.

## Random Trees

We build a basic regression tree, try to prune it and then fit a prediction model.

##   
## Regression tree:  
## rpart(formula = rent\_full ~ ., data = X\_train\_tree, method = "anova")  
##   
## Variables actually used in tree construction:  
## [1] area KTKZ rooms  
##   
## Root node error: 3.3387e+10/72002 = 463692  
##   
## n= 72002   
##   
## CP nsplit rel error xerror xstd  
## 1 0.297313 0 1.00000 1.00002 0.0099029  
## 2 0.085528 1 0.70269 0.70274 0.0072257  
## 3 0.066836 2 0.61716 0.61802 0.0061745  
## 4 0.036913 3 0.55032 0.55119 0.0060384  
## 5 0.033072 4 0.51341 0.51431 0.0058879  
## 6 0.023125 5 0.48034 0.48134 0.0052588  
## 7 0.013212 6 0.45721 0.45920 0.0048157  
## 8 0.010000 7 0.44400 0.44599 0.0048046



## [1] 451.185

The RMSE is in the ranks of our Ridge and Lasso-Models wit RMSE(output\_i$Y\_test\_tree,output\_i$predictions\_tree\_i).

The best model until now was our Linear Regression based on the imputed dataset.

Let’s use some more sofisticated regression trees, ctrees, as regression trees that are based on information Gain (the standard trees I’ve used so far) don’t do the trick. I’ll swap to **Conditional Inference Trees**.

library(party)

## Loading required package: grid

##   
## Attaching package: 'grid'

## The following object is masked from 'package:BBmisc':  
##   
## explode

## Loading required package: mvtnorm

## Loading required package: modeltools

## Loading required package: stats4

## Loading required package: strucchange

## Loading required package: zoo

##   
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':  
##   
## as.Date, as.Date.numeric

## Loading required package: sandwich

##   
## Attaching package: 'strucchange'

## The following object is masked from 'package:stringr':  
##   
## boundary

fit\_ctree <- ctree(formula = rent\_full ~ ., data = X\_train\_tree)  
  
prediction\_ctree <- predict(fit\_ctree, X\_test\_tree)  
  
output\_ctree <- data.frame(prediction\_ctree, Y\_test\_tree)  
  
RMSE(output\_ctree$rent\_full,output\_ctree$Y\_test\_tree)

## [1] 374.5985

The conditional trees work as follows:

*“ctree, according to its authors (see chl’s comments) avoids the following variable selection bias of rpart (and related methods): They tend to select variables that have many possible splits or many missing values. Unlike the others, ctree uses a significance test procedure in order to select variables instead of selecting the variable that maximizes an information measure (e.g. Gini coefficient).”*

As we can see, with a RMSE of RMSE(output\_ctree$rent\_full,output\_ctree$Y\_test\_tree) they are better at predicting the housing prices than all of my other models.

## Random Forest

library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:gridExtra':  
##   
## combine

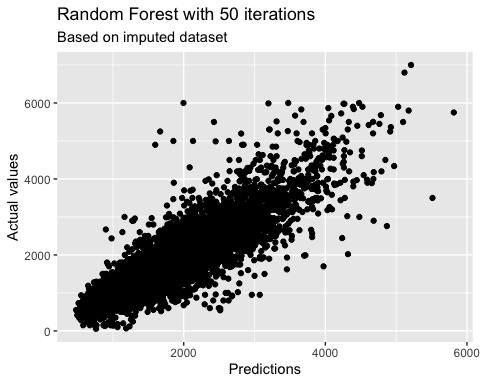
## The following object is masked from 'package:dplyr':  
##   
## combine

## The following object is masked from 'package:ggplot2':  
##   
## margin

X <- complete\_set\_rl  
X\_train <- X[index,c(1:25,27:28)]  
X\_test <- X[-index, c(1:25,27:28)]  
Y\_train <- X[index,26]  
Y\_test<- X[-index,26]  
  
fit\_randomf <- randomForest(x=X\_train, y = Y\_train, ntree = 50)  
  
prediction\_randomf <- predict(fit\_randomf, X\_test)  
  
output\_randomf <- data.frame(prediction\_randomf, Y\_test)  
  
RMSE(output\_randomf$prediction\_randomf,output\_randomf$Y\_test)

## [1] 309.4756

ggplot(output\_randomf, aes(prediction\_randomf,Y\_test)) +  
 geom\_point() +  
 labs(title = "Random Forest with 50 iterations",  
 subtitle = "Based on imputed dataset") +  
 xlab("Predictions") +  
 ylab("Actual values")



With only a RMSE of RMSE(output\_randomf$prediction\_randomf,output\_randomf$Y\_test), the random forest of n = 50 trees got by far the best result! The output values are nearly linearilly distributed in comparison to the actual values.