Machine Learning Algorithms Documentation

Geiser, Gruen, Hefti, Kuster

08/01/2021

Data preparation Dataset - German Housing Data

Dataset of housing prices in the german market, scraped from Immo Scout24 and made available on https://www.kaggle.com/scriptsultan/german-house-prices

The dataset contains 10’318 observations with 20 variables (containing continuous, categorical variables as well as count data)

For the sake of keeping the document short, some code or/and output will not be shown in the document

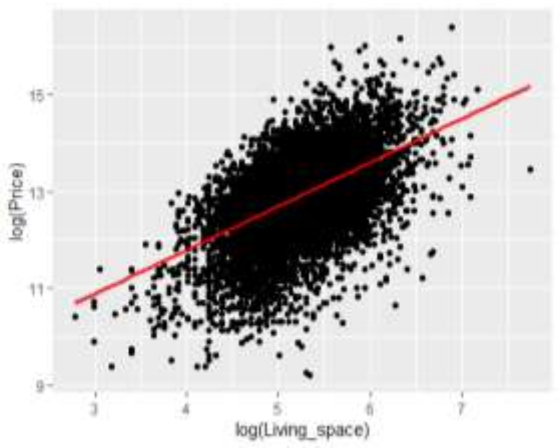
Origin data set cleaned and graphically analyzed with different plots (scatterplots, boxplots, barplots etc.), variables transformed to integers and/or count data, right skewed variables log transformed and added to the data frame.

Week 1 - Linear Models

Data Visualisation and Linear regressions

Scatterplot with regression line for log(Price) against log(Living_space)

We plot the response variable log(Price) against the predictor log(Living_Space) to get a first impression. The plot displays that there is may a positive correlation between the two variables. We further investigate this by a fitting a simple linear regression.



Fitting a Simple Linear regression of log.price against log.living and check the coefficients

```
#linear model
lm.log.price_living <- lm(log.price ~ log.living, data = data1)
summary(lm.log.price_living)
##
## Call:
## lm(formula = log.price ~ log.living, data = data1)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7836 -0.4079  0.0856  0.4683  2.7581
##
## Coefficients:
##             Estimate Std. Error t value            Pr(>|t|)
## (Intercept)  8.17522    0.07596  107.62 <0.0000000000000002 ***
## log.living   0.90280    0.01455   62.05 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7145 on 10316 degrees of freedom
## Multiple R-squared:  0.2718, Adjusted R-squared:  0.2717
## F-statistic:  3850 on 1 and 10316 DF,  p-value: < 0.00000000000000022
#estimated regression coefficients including p-values
coef(lm.log.price_living)
## (Intercept)  log.living
##   8.1752232   0.9028032
exp(coef(lm.log.price_living))
## (Intercept)  log.living
## 3551.847713    2.466508
```

Result: Very significant p-values for log.price ~ log.living. We can assume that the variable log.living has an effect on the dependent variable log.price with a positive correlation, meaning: if the log.living parameter increases in value also the Price of the property will increase.

Due to the log transformation of the variables we have to exponentiate the values before the interpretation. For the intercept exp(8.17522)= 3551.84 and the slope exp(0.9028)=2.47, looking the p-values, which are very small, we have a strong evidence that the slope of log.living is not flat and therefore the variable has an effect on the log.price variable.

Linear regression of log.price against log.living including the Type and finding the intercept for the different Types

```
##linear model
lm.log.price_living_type <- lm(data = data1, log.price ~ log.living + Type)
summary(lm.log.price_living_type)
##
## Call:
## lm(formula = log.price ~ log.living + Type, data = data1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9802 -0.3849  0.0716  0.4556  2.7356
##
## Coefficients:
##                          Estimate Std. Error t value             Pr(>|t|)
## (Intercept)               7.47225    0.09409  79.414 < 0.0000000000000002 ***
## log.living                0.99691    0.01679  59.378 < 0.0000000000000002 ***
## TypeBungalow              0.24691    0.05737   4.304    0.000016956457415 ***
## TypeCastle               -0.33356    0.31169  -1.070             0.284573
## TypeCorner house         -0.14759    0.06058  -2.436             0.014857 *
## TypeDuplex               -0.06320    0.03900  -1.620             0.105186
## TypeFarmhouse             0.26026    0.04599   5.659    0.000000015603919 ***
## TypeMid-terrace house     0.24470    0.03679   6.651    0.000000000030494 ***
## TypeMultiple dwelling     0.35983    0.05029   7.155    0.000000000000891 ***
## TypeResidential property  0.17411    0.05094   3.418             0.000634 ***
## TypeSingle dwelling       0.39721    0.04091   9.708 < 0.0000000000000002 ***
## TypeSpecial property      0.46909    0.05103   9.193 < 0.0000000000000002 ***
## TypeVilla                 0.70309    0.05077  13.847 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6912 on 10305 degrees of freedom
## Multiple R-squared:  0.3193, Adjusted R-squared:  0.3185
## F-statistic: 402.8 on 12 and 10305 DF,  p-value: < 0.00000000000000022
```

Result: Due to the small p-values of the intercept and of the different types we have a strong evidence that the different types have an effect on the log.price ~ log.living. Only Type ‘Castle’ and Type ‘Duplex’ seem not having an effect. Because of the log transformation of the independent and dependent variables we have to interpret an increase of living space by 1% with an increase of the price variable by about 0.99% (according to the slope). The coefficient of log.living is the estimated elasticity of price with respect to living space.

Linear regression of log.price against log.living including the ‘Type’ interaction

```
##linear model
lm.log.price_living_type2 <- lm(data = data1, log.price ~ log.living * Type)
##Measures of fit
formula(lm.log.price_living_type)
## log.price ~ log.living + Type
#r.squared
summary(lm.log.price_living_type)$r.squared
## [1] 0.3192727
#adj.r.squared
summary(lm.log.price_living_type)$adj.r.squared
## [1] 0.31848
formula(lm.log.price_living_type2)
## log.price ~ log.living * Type
#r.squared
summary(lm.log.price_living_type2)$r.squared
## [1] 0.3263949
#adj.r.squared
summary(lm.log.price_living_type2)$adj.r.squared
## [1] 0.3248899
```

If we compare the R^2 and adj. R^2 of the additive and the multiplicative model (for interaction), we see only a little improvement. Therefore we might have to decide to continue with the less complex model.

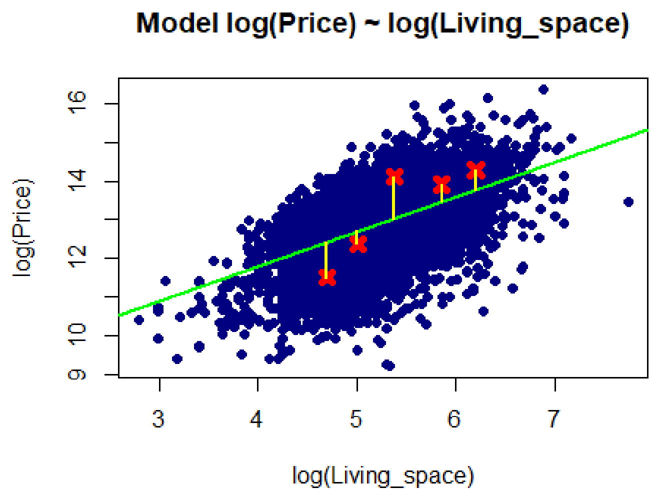
Fitted values

The function fitted() can be used to extract the predicted values for the existing observations

```
attach(data)
#lm.log.price_living
fitted.price_living <- fitted(lm.log.price_living)
#generate plot
plot(log(Price)~ log(Living_space), main = 'Model log(Price) ~ log(Living_space)', col = 'navy', pch = 16)
points(fitted.price_living ~ log(Living_space), col = 'red', pch = 16)
abline(lm.log.price_living, col = 'yellow', lwd = 2.5)
```

Residuals of model log(Price) ~ log(Living_space)

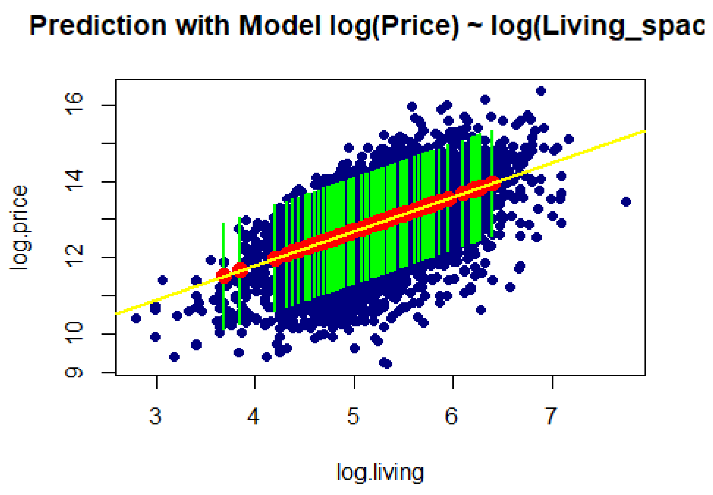
```
attach(data1)
resid.price_living <- resid(lm.log.price_living)
length(resid.price_living)
## [1] 10318
set.seed(100)
id <- sample(x = 1:10318, size = 5)
resid.price_living[id]
##       3786        503       3430       3696       4090
##  0.5106768 -0.3315001  0.4470366 -0.9261093  1.0834033
fitted.price_living[id]
##     3786      503     3430     3696     4090
## 13.77484 12.69884 13.46378 12.41883 13.03221
```



With this code we take 5 examples of the data set and plot it with the predicted values including the residuals (yellow line), which indicates the distance to the estimated regression line.

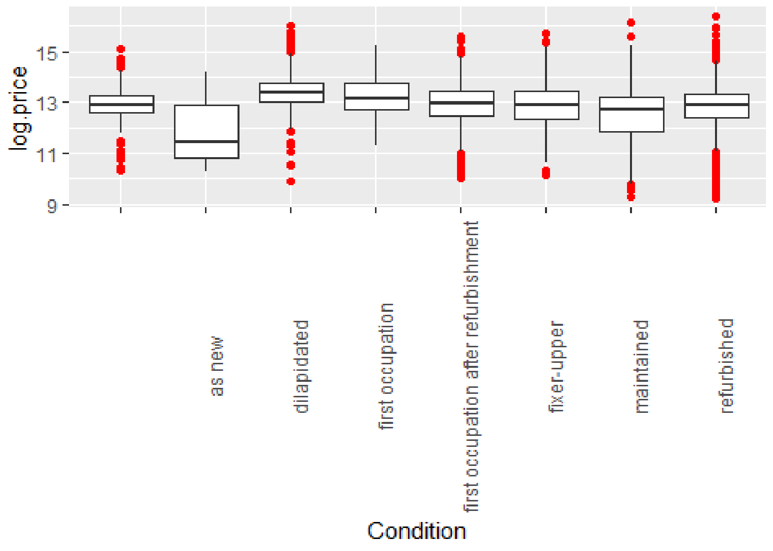
Predicting values using splitted data set 99:1 ratio

```
#split dataset
split99 <- round(nrow(data1)* 0.99)
train <- data1[1:split99,]
test <- data1[(split99 + 1):nrow(data1),]
#linear regression model
lm.train <- lm(log.price ~ log.living, data = train)
summary(lm.train)
##
## Call:
## lm(formula = log.price ~ log.living, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7888 -0.4043  0.0847  0.4633  2.7534
##
## Coefficients:
##             Estimate Std. Error t value            Pr(>|t|)
## (Intercept)  8.19143    0.07604  107.72 <0.0000000000000002 ***
## log.living   0.90074    0.01456   61.84 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.711 on 10213 degrees of freedom
## Multiple R-squared:  0.2725, Adjusted R-squared:  0.2724
## F-statistic:  3825 on 1 and 10213 DF,  p-value: < 0.00000000000000022
#predictions
pred.new.living <- predict(object = lm.train, newdata = test)
pred.new.living.CI <- predict(object = lm.train, interval = 'prediction', newdata = test)
```



In this plot we see the estimated regression line (in yellow) with the estimated values (red dots) with the corresponding confident interval (green lines). The blue dots are the original dataset points.

Testing the effect of a categorical variable and post-hoc contrasts



Model

```
lm.price_condition.1 <- lm(log.price ~ Condition, data = data1)
summary(lm.price_condition.1)
##
## Call:
## lm(formula = log.price ~ Condition, data = data1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.6043 -0.4304  0.0333  0.4884  3.6247
##
## Coefficients:
##                                         Estimate Std. Error t value
## (Intercept)                             12.95695    0.04213 307.553
## Conditionas new                         -1.01885    0.24696  -4.126
## Conditiondilapidated                     0.47108    0.04838   9.738
## Conditionfirst occupation                0.19331    0.09330   2.072
## Conditionfirst occupation after refurbishment -0.05091    0.05685  -0.895
## Conditionfixer-upper                            -0.10881    0.05398  -2.016
## Conditionmaintained                             -0.42435    0.04906  -8.649
## Conditionrefurbished                            -0.14229    0.04328  -3.288
##                                                             Pr(>|t|)
## (Intercept)                                     < 0.0000000000000002 ***
## Conditionas new                                            0.0000373 ***
## Conditiondilapidated                            < 0.0000000000000002 ***
## Conditionfirst occupation                                    0.03829 *
## Conditionfirst occupation after refurbishment                0.37055
## Conditionfixer-upper                                         0.04384 *
## Conditionmaintained                             < 0.0000000000000002 ***
## Conditionrefurbished                                         0.00101 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8071 on 10310 degrees of freedom
## Multiple R-squared:  0.07146,    Adjusted R-squared:  0.07083
## F-statistic: 113.4 on 7 and 10310 DF,  p-value: < 0.00000000000000022
aggregate(log.price ~Condition,
          FUN = mean, data = data1)
##                               Condition log.price
## 1                                        12.95695
## 2                                as new  11.93810
## 3                           dilapidated  13.42803
## 4                      first occupation  13.15025
## 5 first occupation after refurbishment  12.90604
## 6                           fixer-upper  12.84814
## 7                            maintained  12.53260
## 8                           refurbished  12.81466
#model without slope, only intercept
lm.price_condition.0 <- lm(log.price ~ 1, data = data1)
summary(lm.price_condition.0)
##
## Call:
## lm(formula = log.price ~ 1, data = data1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.6576 -0.4388  0.0287  0.5166  3.5125
##
## Coefficients:
##              Estimate Std. Error t value            Pr(>|t|)
## (Intercept) 12.867983   0.008243    1561 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8373 on 10317 degrees of freedom
```

R uses “treatment contrasts” and therefore the Intercept refers to the first in alphabetical order, here “Null”. The other coefficients represent the difference. Whilst considering the boxplots, it seems rather suprising that ‘as new’ has the lowest mean price value. Condition ‘as new’, ‘dilapidated’, ‘first occupation’, ‘maintained’ to have a strong effect on the response variable. The results are very suprising, as new condition negatively influences the price and condition dilapidated increases the price. The results obtained seem weak, represented in the adj. R-squared with 0.07083.

```
#Anova
anova(lm.price_condition.0, lm.price_condition.1)
## Analysis of Variance Table
##
## Model 1: log.price ~ 1
## Model 2: log.price ~ Condition
##   Res.Df    RSS Df Sum of Sq      F                Pr(>F)
## 1  10317 7232.5
## 2  10310 6715.7  7    516.86 113.36 < 0.00000000000000022 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

To further test the predictor a lm is built without slopes for every type and then tested by an F-Test (Anova). Surprisingly, the Model with slopes for the Type seems to perform better as indicated through a lower RSS value.

```
#post-hoc contrasts
library(multcomp)
ph.test.1 <- glht(model = lm.price_condition.1, linfct = mcp(Condition = c('refurbished - dilapidated = 0'
)))
summary(ph.test.1)
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: User-defined Contrasts
##
##
## Fit: lm(formula = log.price ~ Condition, data = data1)
##
## Linear Hypotheses:
##                                 Estimate Std. Error t value            Pr(>|t|)
## refurbished - dilapidated == 0 -0.61337    0.02576  -23.81 <0.0000000000000002
##
## refurbished - dilapidated == 0 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

The post-hoc contrasts are used to test whether ‘dilapidated’ and ‘refurbished’ differ. The outcome states that ‘dilapidated’ increases the price. This confirms also the visual analysis of the boxplots, but still is suprising as explained above.

Adding more categorical variables to the testing above

```
Year_built1 <- as.integer(data1$Year_built)
floor1 <- as.integer(data1$Floors)
typeof(Year_built1)
## [1] "integer"
lm.price_condition.2 <- update(lm.price_condition.1,. ~ . + Type + log.rooms +
                                 State + Energy_efficiency_class + Year_built1 + Furnishing_quality + floo
r1)
formula(lm.price_condition.2)
## log.price ~ Condition + Type + log.rooms + State + Energy_efficiency_class +
##     Year_built1 + Furnishing_quality + floor1
drop1(lm.price_condition.2, test = "F")
## Single term deletions
##
## Model:
## log.price ~ Condition + Type + log.rooms + State + Energy_efficiency_class +
##     Year_built1 + Furnishing_quality + floor1
##                         Df Sum of Sq    RSS     AIC  F value
## <none>                               2148.4 -8631.8
## Condition                7     17.57 2165.9 -8587.1   8.3684
## Type                    11    134.63 2283.0 -8215.5  40.7950
## log.rooms                1    152.62 2301.0 -8138.9 508.7221
## State                   15    638.74 2787.1 -6784.8 141.9381
## Energy_efficiency_class  9     33.56 2181.9 -8538.0  12.4294
## Year_built1              1     56.88 2205.2 -8445.4 189.5786
## Furnishing_quality       4    346.08 2494.4 -7562.8 288.3902
## floor1                   1     31.86 2180.2 -8527.7 106.1896
##                                     Pr(>F)
## <none>
## Condition                  0.0000000003205 ***
## Type                    < 0.00000000000000022 ***
## log.rooms               < 0.00000000000000022 ***
## State                   < 0.00000000000000022 ***
## Energy_efficiency_class < 0.00000000000000022 ***
## Year_built1             < 0.00000000000000022 ***
## Furnishing_quality      < 0.00000000000000022 ***
## floor1                  < 0.00000000000000022 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Drop1 function performes automatically or each variable presence in the model an F test. According to the results all independent variables in the model seem to have an effect.

Week 2 - Non-linearity

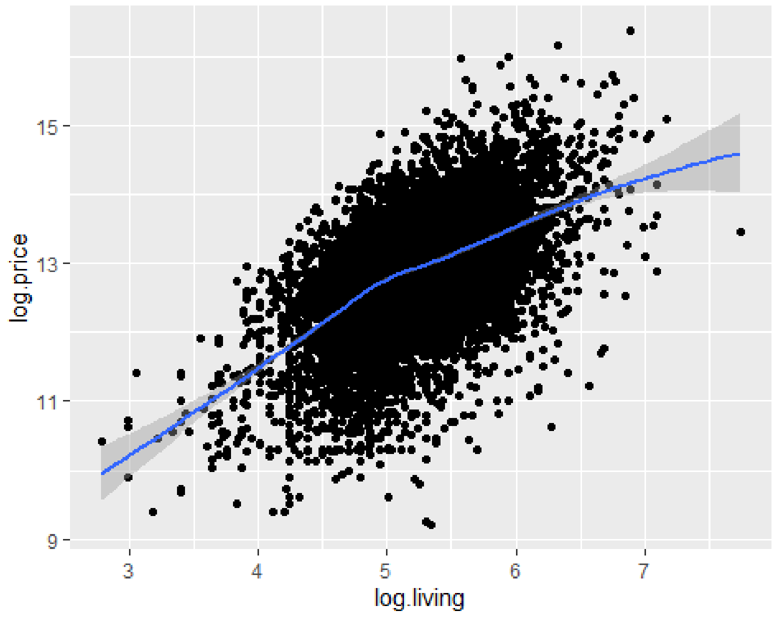
Polynomials

By including polynomials (e.g. x1 + x1^2) we can model non linear relationships with a Linear Model.

Graphical analysis

log(Price) ~ log(Living_space)

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



The graphical analysis shows a non-linear relationship for the predictor living_space.

Model with a linear effect for log.living with log.rooms

```
lm.living.1 <- lm(log.price ~ log.living + log.rooms)
summary(lm.living.1)
##
## Call:
## lm(formula = log.price ~ log.living + log.rooms)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9154 -0.3918  0.0788  0.4617  2.6939
##
## Coefficients:
##             Estimate Std. Error t value            Pr(>|t|)
## (Intercept)  7.41210    0.08773   84.49 <0.0000000000000002 ***
## log.living   1.19671    0.02268   52.76 <0.0000000000000002 ***
## log.rooms   -0.41705    0.02492  -16.74 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7051 on 10315 degrees of freedom
## Multiple R-squared:  0.291,  Adjusted R-squared:  0.2909
## F-statistic:  2117 on 2 and 10315 DF,  p-value: < 0.00000000000000022
```

Both predictors show a strong effect on the response variable price. Now we want to build a model with a quadratic effect for living space.

Model with a cuadratic effect for log.living

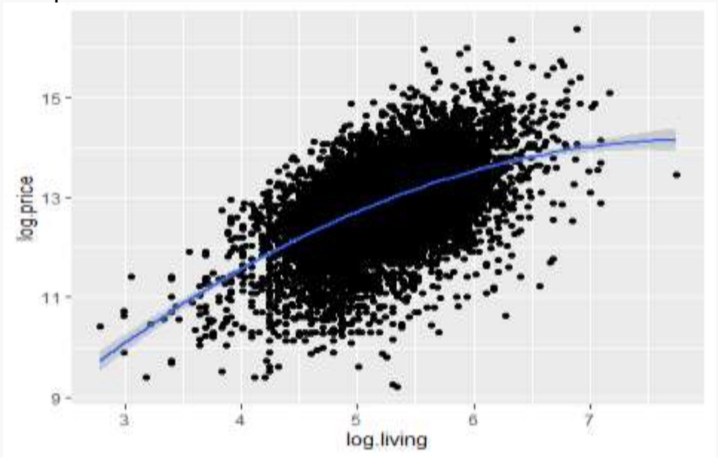
```
#model with a quadratic poly
lm.living.2 <- lm(log.price ~ log.rooms + poly(log.living, degree = 2))
summary(lm.living.2)
```

Now that we have added the quadratic effect (with degree = 2). The two models can be compared by an F-Test.

Compare the two models by an F-Test

```
#test in quadratic
anova(lm.living.1, lm.living.2)
## Analysis of Variance Table
##
## Model 1: log.price ~ log.living + log.rooms
## Model 2: log.price ~ log.rooms + poly(log.living, degree = 2)
##   Res.Df    RSS Df Sum of Sq      F                Pr(>F)
## 1  10315 5127.7
## 2  10314 5067.1  1    60.652 123.46 < 0.00000000000000022 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The second model with a quadratic effect for living space has a better performance, as the RSS is lower th an the one from the model without a quadratic effect. Additionally, the p-value indicates that the second m odel performs better.



The quadratic fit seems to model the non-linear relationship of living space quite well. Nevertheless, we try to fit a cubic poly.

Model with a cubic poly

```
#model with a cubic poly
lm.living.3 <- lm(log.price ~ log.rooms + poly(log.living, degree = 3))
summary(lm.living.3)
anova(lm.living.2, lm.living.3)
## Analysis of Variance Table
##
## Model 1: log.price ~ log.rooms + poly(log.living, degree = 2)
## Model 2: log.price ~ log.rooms + poly(log.living, degree = 3)
##   Res.Df    RSS Df  Sum of Sq      F Pr(>F)
## 1  10314 5067.1
## 2  10313 5067.1  1 0.00072045 0.0015 0.9695
```

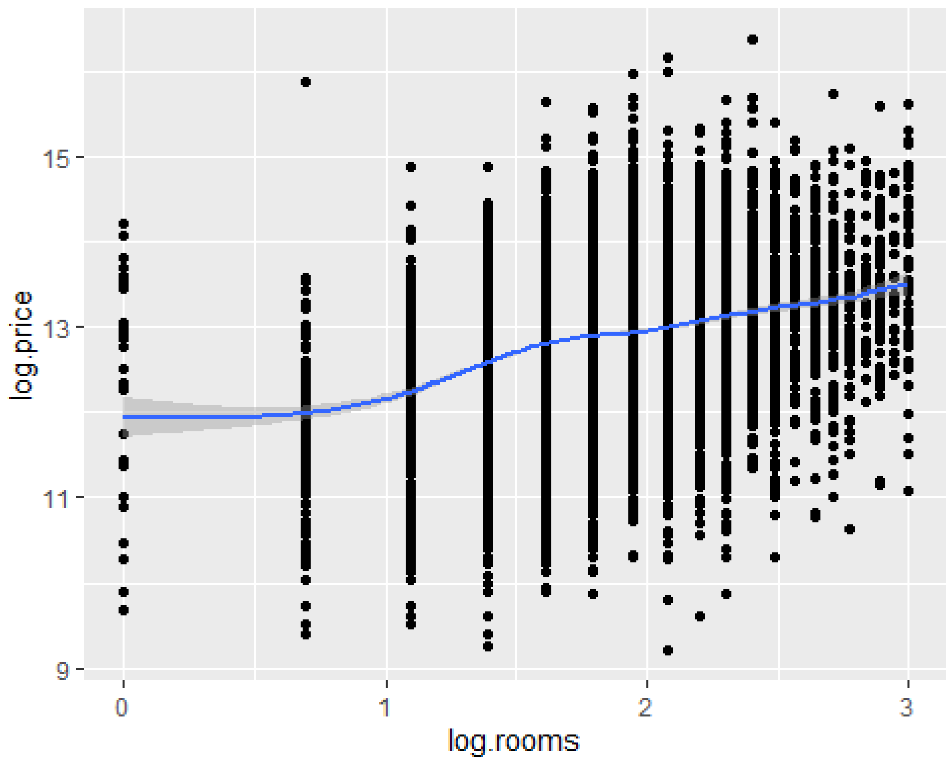
As expected, the cubic term does not fit better, because the RSS is equal to the quadratic model, so we prefer the less complex model with degree 2.

Generalised Additive Models - GAMs

Graphical analysis

log(Price) ~ log(Rooms)

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

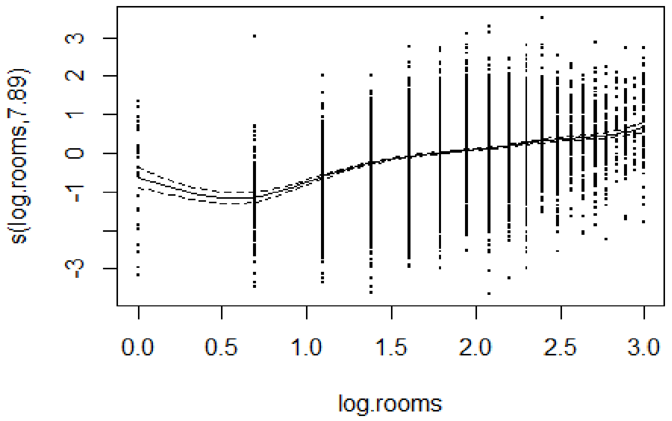


The graphical analysis shows a non-linear relationship for the predictor log.rooms. As from eyeballing it seems to not be a quadratic or cubic effect a GAM will be applied (as it chooses the degree of complexity automatically).

GAMs for log(Price) ~ log(Rooms)

```
attach(data1)
gam.log.price.log.rooms <- gam(log.price ~ s(log.rooms))
summary(gam.log.price.log.rooms)
##
## Family: gaussian
## Link function: identity
##
## Formula:
## log.price ~ s(log.rooms)
##
## Parametric coefficients:
##              Estimate Std. Error t value            Pr(>|t|)
## (Intercept) 12.867983   0.007783    1653 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                edf Ref.df     F             p-value
## s(log.rooms) 7.888  8.623 145.8 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

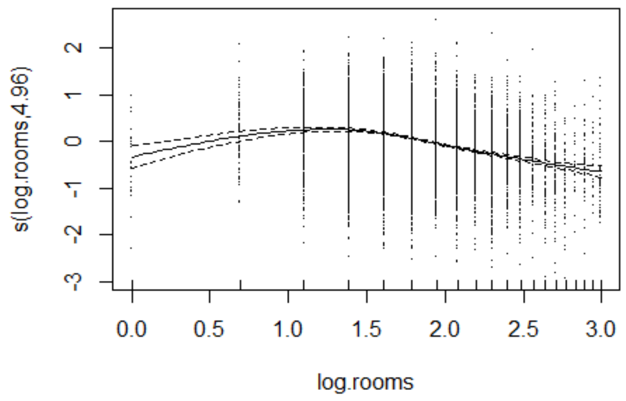
```
## R-sq.(adj) =  0.108   Deviance explained = 10.9%
## GCV = 0.62562  Scale est. = 0.62509   n = 10318
```



The GAM-model has an edf of 7.888 (so degree = almost 8). This shows as that the GAM function choose a polynomial of 7.888

GAMs for log(Price) ~ log(Living_space) + s(log.rooms) + s(log(Garages))

```
gam.log.price.log.living <- gam(log.price ~ log.living + s(log.rooms) + s(log(Garages)))
summary(gam.log.price.log.living)
##
## Family: gaussian
## Link function: identity
##
## Formula:
## log.price ~ log.living + s(log.rooms) + s(log(Garages))
##
## Parametric coefficients:
##             Estimate Std. Error t value            Pr(>|t|)
## (Intercept)  6.46447    0.13313   48.56 <0.0000000000000002 ***
## log.living   1.24132    0.02547   48.74 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                   edf Ref.df      F             p-value
## s(log.rooms)    4.964  6.084 67.976 < 0.0000000000000002 ***
## s(log(Garages)) 8.215  8.681  3.852           0.0000691 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.305   Deviance explained = 30.6%
## GCV = 0.42626  Scale est. = 0.4255    n = 8437
```



The GAM-model has an edf of 4.964 (so degree = almost 5) for log.rooms and an edf of 8.215 for log(Garages). Both independent variables seem to have a strong effect.

Week 3 - Generalised Linear Models

GLM - Possion Model

Count Data

With the GLM function and the family “possion” we could generalize the Linear model.

```
glm.rooms <- glm(Rooms ~ Type, family = "poisson", data = data)
summary(glm.rooms)
##
## Call:
## glm(formula = Rooms ~ Type, family = "poisson", data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6463  -0.6320  -0.0781   0.3702   4.4559
##
## Coefficients:
##                           Estimate Std. Error z value             Pr(>|z|)
## (Intercept)                2.02917    0.01848 109.821 < 0.0000000000000002 ***
## TypeBungalow               0.17457    0.02838   6.152       0.000000000766 ***
## TypeCastle                 0.55104    0.12447   4.427       0.000009550749 ***
## TypeCorner house           0.01343    0.03162   0.425                0.671
## TypeDuplex                 0.26787    0.01981  13.522 < 0.0000000000000002 ***
## TypeFarmhouse             -0.43417    0.02638 -16.456 < 0.0000000000000002 ***
## TypeMid-terrace house     -0.22653    0.01948 -11.626 < 0.0000000000000002 ***
## TypeMultiple dwelling     -0.38499    0.02917 -13.197 < 0.0000000000000002 ***
## TypeResidential property  0.04744    0.02637   1.799                0.072 .
## TypeSingle dwelling       -0.38432    0.02252 -17.066 < 0.0000000000000002 ***
## TypeSpecial property      -0.47841    0.03056 -15.655 < 0.0000000000000002 ***
## TypeVilla                  0.14646    0.02522   5.806       0.000000006387 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 13951.1  on 10317  degrees of freedom
## Residual deviance:  9350.9  on 10306  degrees of freedom
## AIC: 47557
##
## Number of Fisher Scoring iterations: 4
```

```
#exponentiate coefficients
exp(coef(glm.rooms))
##             (Intercept)             TypeBungalow               TypeCastle
##               7.6077922                1.1907296                1.7350632
##         TypeCorner house               TypeDuplex            TypeFarmhouse
##               1.0135214                1.3071727                0.6477992
##     TypeMid-terrace house  TypeMultiple dwelling TypeResidential property
##               0.7972979                0.6804577                1.0485829
##      TypeSingle dwelling     TypeSpecial property                TypeVilla
##               0.6809147                0.6197703                1.1577330
```

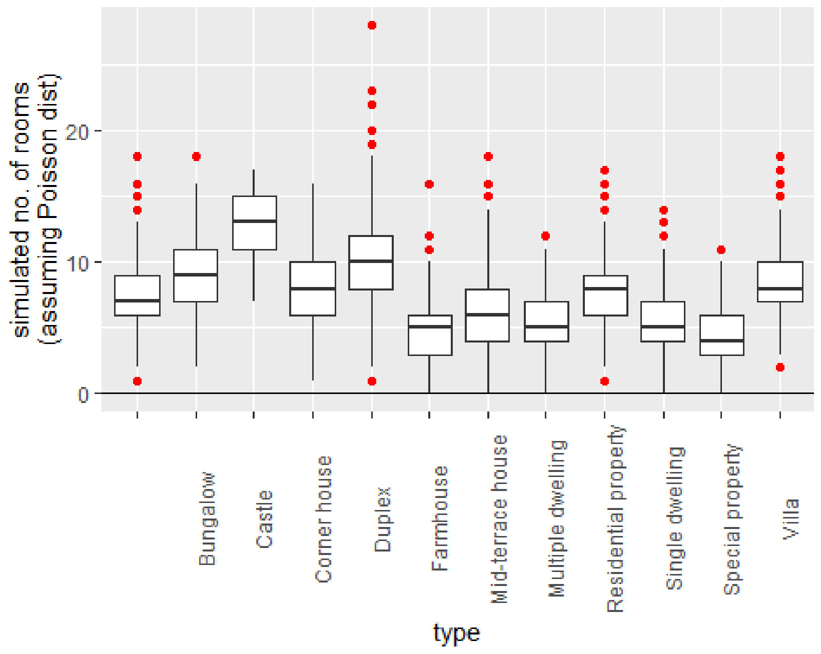
Before the Interpretation of the coefficients of a Poisson model the inverse, the exponential, is needed due to use of the link function (log() function). The Interpretation of the coefficients are: A house with no Type information has around 7.6 rooms and e.g. Villa has ca. 15.77% more rooms (ca. 8.8 rooms)

```
#to double check and convince ourselves as single house (type) is checked and predicted
library(tidyverse)
#which(data$Type == "")
#data[99,]
fitted.room <- fitted(glm.rooms)[99]
fitted.room
##       99
## 7.607792
specific.room <- data[99,]
specific.room$Type <- "Villa"
#specific.room
predict(glm.rooms, type = "response",newdata = specific.room)
##       99
## 8.807792
fitted.room * exp(coef(glm.rooms)["TypeVilla"])
##       99
## 8.807792
```

This house with no specific Type, according to the model, is expected to have 7.607 Rooms. If we check the number of rooms after setting the Type to e.g. Villa the model expect 8.807 rooms and this is exactly the same number of rooms we get for the fitted room times the exponential estimated for Villa.

Data simulation from the glm count data model

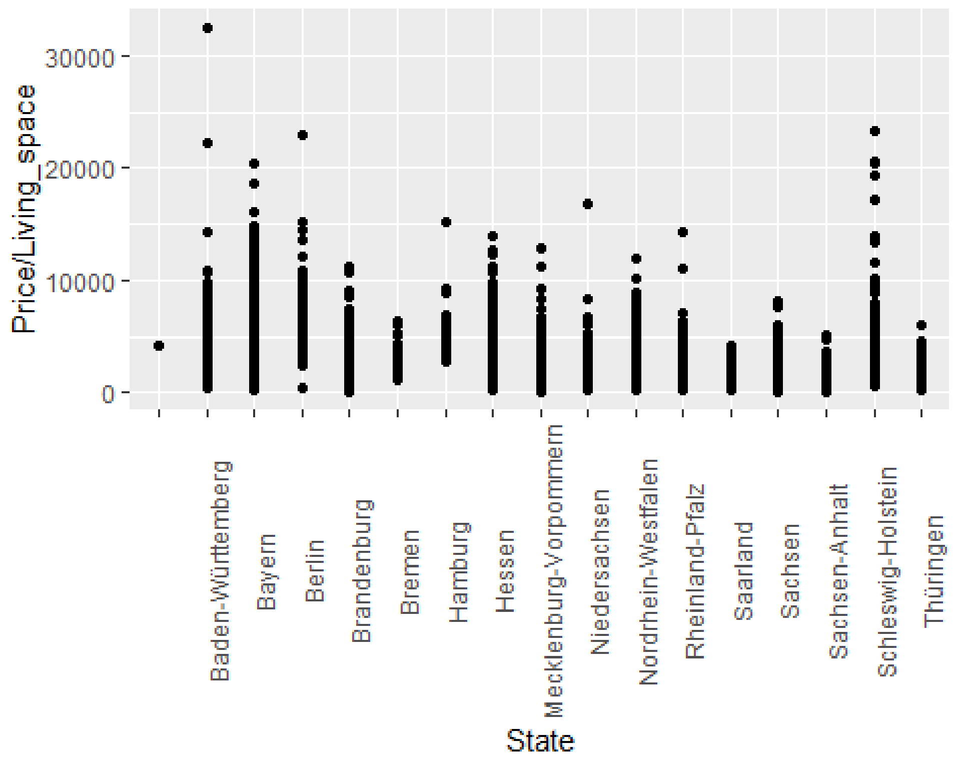
Visualization of the glm count data model



The results of the simulation seem to agree with the observed data (no negative values, similar variation, only integer values).

GLM with binomial data factor variable

```
glm.sq.price <- glm(cbind(Price, Living_space)~ State,
                    family = "binomial",
                    data = data)
## Warning in eval(family$initialize): non-integer counts in a binomial glm!
summary(glm.sq.price)
##
## Call:
## glm(formula = cbind(Price, Living_space) ~ State, family = "binomial",
##     data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -59.255   -6.366   -1.195    3.473   61.109
##
## Coefficients:
##                              Estimate Std. Error z value             Pr(>|z|)
## (Intercept)                   8.34931    0.09536  87.558 < 0.0000000000000002
## StateBaden-Württemberg       -0.26983    0.09538  -2.829             0.004667
## StateBayern                  -0.11439    0.09538  -1.199             0.230381
## StateBerlin                   0.22109    0.09549   2.315             0.020597
## StateBrandenburg             -0.34169    0.09542  -3.581             0.000342
## StateBremen                  -0.45021    0.09601  -4.689 0.000002744267682783
## StateHamburg                  0.20992    0.09607   2.185             0.028878
## StateHessen                  -0.35562    0.09538  -3.728             0.000193
## StateMecklenburg-Vorpommern -0.77292    0.09544  -8.099 0.000000000000000555
## StateNiedersachsen           -0.78534    0.09538  -8.234 < 0.0000000000000002
## StateNordrhein-Westfalen     -0.52038    0.09537  -5.456 0.000000048613961012
## StateRheinland-Pfalz         -0.77836    0.09538  -8.160 0.000000000000000334
## StateSaarland                -1.13274    0.09552 -11.859 < 0.0000000000000002
## StateSachsen                 -1.08280    0.09541 -11.349 < 0.0000000000000002
## StateSachsen-Anhalt          -1.36482    0.09544 -14.300 < 0.0000000000000002
## StateSchleswig-Holstein      -0.36652    0.09541  -3.842             0.000122
## StateThüringen               -1.20943    0.09555 -12.657 < 0.0000000000000002
##
## (Intercept)                 ***
## StateBaden-Württemberg      **
## StateBayern
## StateBerlin                 *
## StateBrandenburg            ***
## StateBremen                 ***
## StateHamburg                *
## StateHessen                 ***
## StateMecklenburg-Vorpommern ***
## StateNiedersachsen          ***
## StateNordrhein-Westfalen    ***
## StateRheinland-Pfalz        ***
## StateSaarland               ***
## StateSachsen                ***
## StateSachsen-Anhalt         ***
## StateSchleswig-Holstein     ***
## StateThüringen              ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1039137  on 10317  degrees of freedom
## Residual deviance:  812246  on 10301  degrees of freedom
## AIC: 884878
##
## Number of Fisher Scoring iterations: 5
#exp(coef(glm.sq.price))
## `geom_smooth()` using formula 'y ~ x'
```



If we compare the Residual deviance and the corresponding degrees of freedom in the summary output we would expect in an truly Poisson distributed data that the residual deviance and the degrees of freedom would be approximately the same value. Therefore, it could be over dispersed here and we use the "quasi-binomial" family.

GLM with quasi-binomial data factor variable

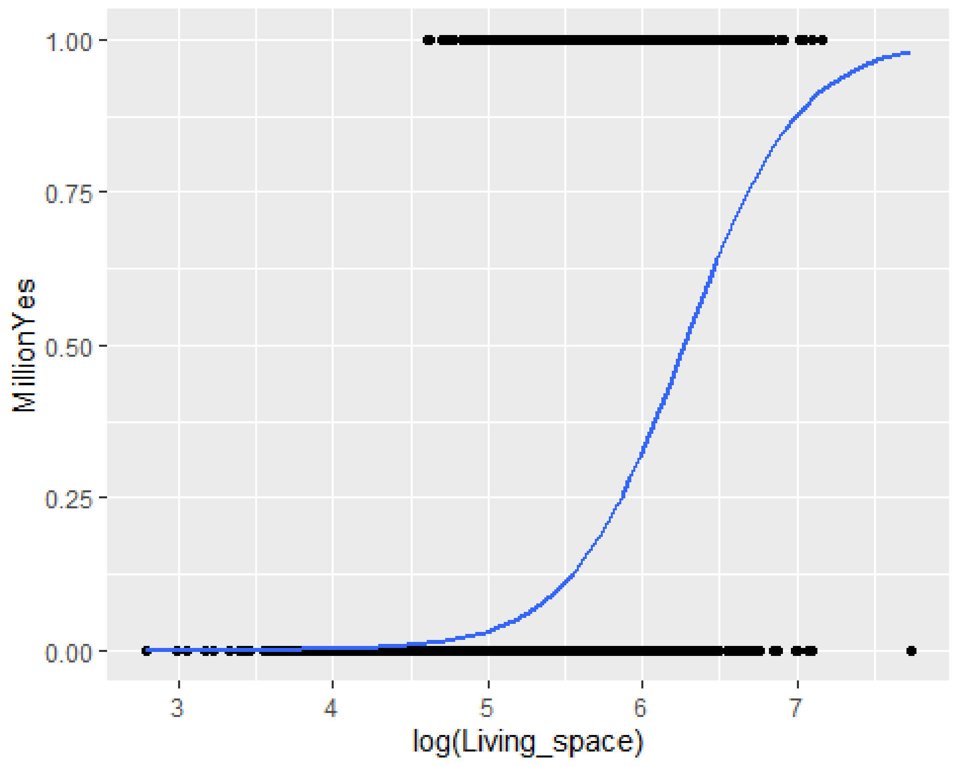
```
glm.sq.price <- glm(cbind(Price, Living_space)~ State,
                    family = "quasibinomial",
                    data = data)
summary(glm.sq.price)
##
## Call:
## glm(formula = cbind(Price, Living_space) ~ State, family = "quasibinomial",
##     data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -59.255   -6.366   -1.195    3.473   61.109
##
## Coefficients:
##                              Estimate Std. Error t value            Pr(>|t|)
## (Intercept)                    8.3493     0.9998   8.351 <0.0000000000000002 ***
## StateBaden-Württemberg        -0.2698     1.0000  -0.270               0.787
## StateBayern                   -0.1144     1.0000  -0.114               0.909
## StateBerlin                    0.2211     1.0012   0.221               0.825
## StateBrandenburg              -0.3417     1.0004  -0.342               0.733
## StateBremen                   -0.4502     1.0066  -0.447               0.655
## StateHamburg                   0.2099     1.0072   0.208               0.835
## StateHessen                   -0.3556     1.0000  -0.356               0.722
## StateMecklenburg-Vorpommern   -0.7729     1.0006  -0.772               0.440
## StateNiedersachsen            -0.7853     1.0000  -0.785               0.432
## StateNordrhein-Westfalen      -0.5204     0.9999  -0.520               0.603
## StateRheinland-Pfalz          -0.7784     1.0000  -0.778               0.436
## StateSaarland                 -1.1327     1.0015  -1.131               0.258
## StateSachsen                  -1.0828     1.0003  -1.083               0.279
## StateSachsen-Anhalt           -1.3648     1.0006  -1.364               0.173
## StateSchleswig-Holstein       -0.3665     1.0003  -0.366               0.714
## StateThüringen                -1.2094     1.0018  -1.207               0.227
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasibinomial family taken to be 109.9222)
##
##     Null deviance: 1039137  on 10317  degrees of freedom
## Residual deviance:  812246  on 10301  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 5
```

The dispersion parameter ist now 109.92, This implies the variance increases faster than linearly. Anyway in this case there is no evidence that the State have an impact on the response variable.

GLM - Binary Model

Let's fit a logistic regression model and add fit to the graph. For that purpose a binary variable has been created from the price variable. Is the Price over one Million Euro or not.

```
## `geom_smooth()` using formula 'y ~ x'
```



As expected higher log(Living_space) leads to a higher probability that the property price is higher than a million.

Week 4 - Support Vector Machines

load data and filter rows with variable 'Type? == 'Multiple dwelling' or 'Villa'

```
svm.data <- fread('german_housing_cleaned.csv',header =T, encoding='UTF-8')
house <- svm.data %>% filter(Type == "Multiple dwelling" | Type == "Villa")

#create new variable 'is.multi' with 'TRUE' 'FALSE' to use for the SVM model
house[,is.multi := as.factor(Type == 'Multiple dwelling')]
```

Split dataset

```
split80 <- round(nrow(house)* 0.80)
train <- house[1:split80,]
test <- house[(split80 + 1):nrow(house),]
```

Fit SVM and do a cross validation with cost 0.25, 0.5 and 1.00

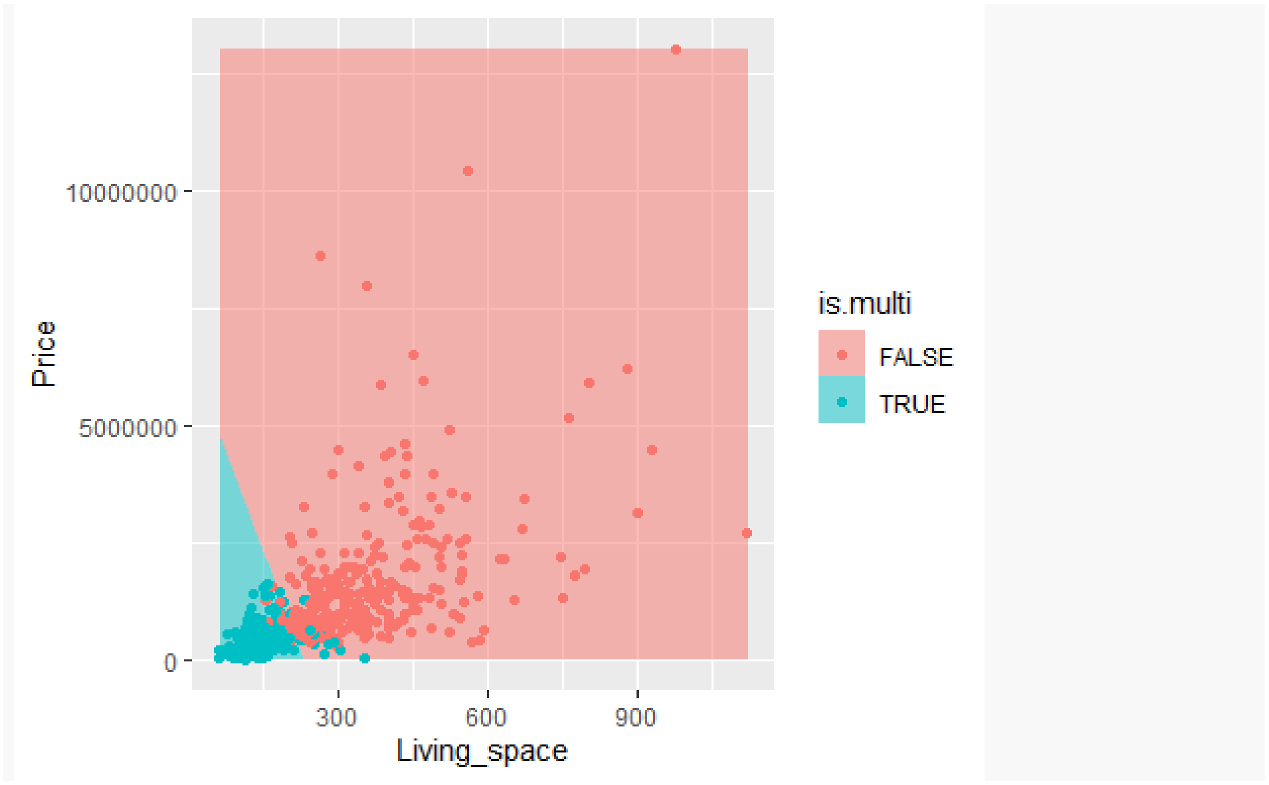
```
set.seed(1)
model <- train(is.multi ~ Price + Living_space,
               data = train, method = 'svmLinear2', trControl = trainControl(method = 'cv'))
print(model)
## Support Vector Machines with Linear Kernel
##
## 611 samples
##   2 predictor
##   2 classes: 'FALSE', 'TRUE'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 550, 550, 550, 549, 551, 550, ...
## Resampling results across tuning parameters:
##
##   cost  Accuracy   Kappa
##   0.25  0.9296651  0.8593149
##   0.50  0.9329438  0.8658320
##   1.00  0.9313044  0.8625365
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cost = 0.5.
```

With the method = ‘svmLinear2’ the model will be tested with different cost values. As the model with cost = 0.5 has the highest accuracy (0.9329385) the prediction will be made with these parameters.

Prediction with SVM model with cost = 0.5

For further prediction we will create a new dataframe with all possible combinations of points (Price in 10.000er steps, sqm in 1.00sqm steps)

```
house1 <- expand.grid(Price = seq(min(house$Price),max(house$Price), 10000),Living_space = seq(min(house$L
iving_space), max(house$Living_space),1))

house1$is.multi <- predict(model, newdata = house1)
```



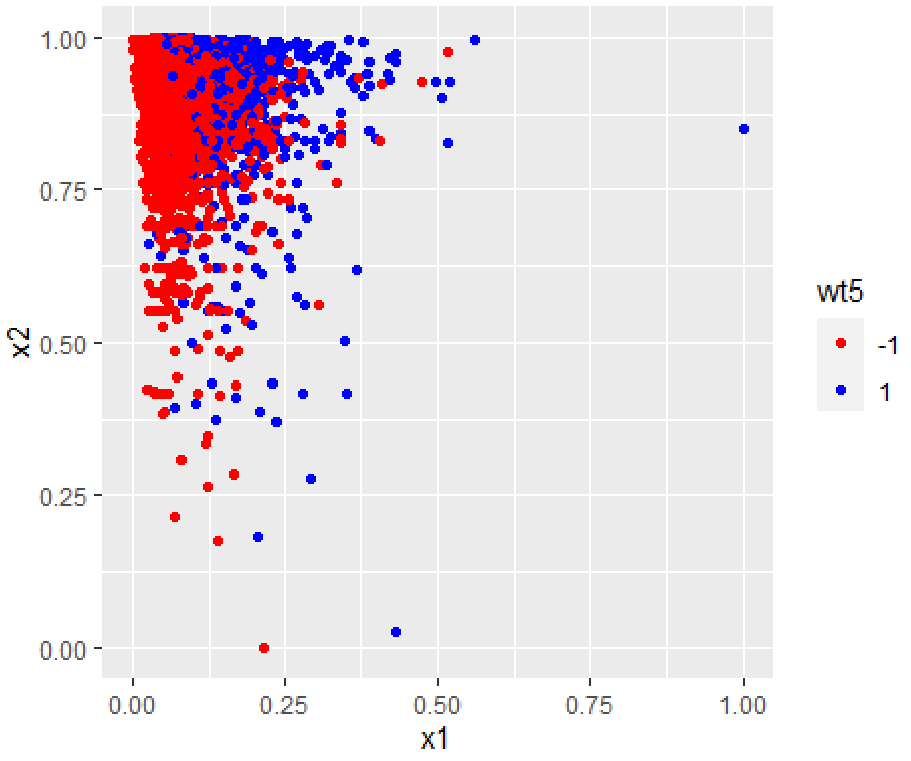
As the plot clearly shows there is a line -1 and 1 classification. In color turqoise we see all the objects that fall into category ‘Multiple dwelling’ and the rose-colored area with the dots are of the Type ‘Villa’. The dots in the other color then the background are mismatched points that fall in the cost of 0.5.

SVM with kernel = ‘radial’

Creating another binary variable from the price variable for a classification with an SVM which use Radial Base Function. Is the Price above 600’000 Euro or not.

Scaling for a nicer plot

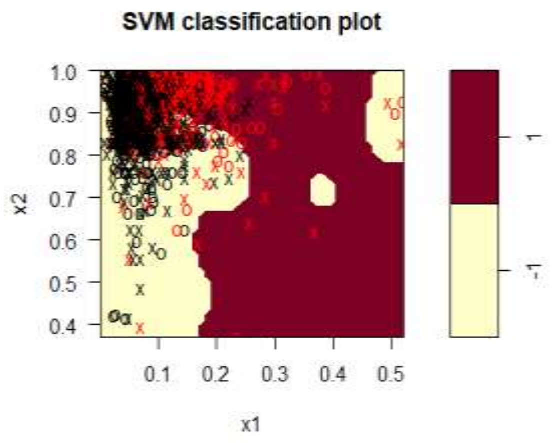
Plot x2 vs. x1, colored by y



Print average accuracy and standard deviation

```
accuracy <- rep(NA, 5)
set.seed(2)
# Calculate accuracies for 10 training/test partitions
for (i in 1:5){
    testdf[, "train"] <- ifelse(runif(nrow(testdf)) < 0.8, 1, 0)
    trainset <- testdf[testdf$train == 1, ]
    testset <- testdf[testdf$train == 0, ]
    trainColNum <- grep("train", names(trainset))
    trainset <- trainset[, -trainColNum]
    testset <- testset[, -trainColNum]
    svm_model <- svm(data.wt5 ~ ., data = trainset, type = "C-classification", kernel = "radial")
    pred_test <- predict(svm_model, testset)
    accuracy[i] <- mean(pred_test == testset$data.wt5)
}
# Print average accuracy and standard deviation
mean(accuracy)
## [1] 0.7770167
sd(accuracy)
## [1] 0.004420142
```

The accuracy mean is 0.7770 with a standard deviation of 0.0044, what will tell us that approx. 77.7% of the data will be classified correctly.



As we can see in this plot we get a nonlinear classification because we use a radial basis function kernel in our SVM code

Week 5 - Neural Networks

ANN - neuralnet package

Preparing data for neuralnet

```
library(neuralnet)
```

Prepare for Training

```
set.seed(123)
indices <- createDataPartition(df.house$data.Price, p = 0.8, list = FALSE)
train <- df.house %>% slice(indices)
test <- df.house %>% slice(-indices)
boxplot(train$data.Price, test$data.Price, df.house %>% sample_frac(0.2) %>% pull(data.Price))
max <- apply(df.house, 2, max)
min <- apply(df.house, 2, min)
df.house_scaled <- as.data.frame(scale(df.house, center = min, scale = max - min))
train_scaled <- df.house_scaled %>% slice(indices)
test_scaled <- df.house_scaled %>% slice(-indices)
```

Fit the Network

```
n <- names(train_scaled)
f <- as.formula(paste("data.Price ~", paste(n[!n %in% "data.Price"], collapse = " + ")))
nn <- neuralnet(f,data=train_scaled,hidden=c(5,3),linear.output=T)
plot(nn)
pred_scaled <- compute(nn, test_scaled %>% select(-data.Price))

pred <- pred_scaled$net.result * (max(df.house$data.Price) - min(df.house$data.Price)) + min(df.house$data
.Price)
#pred
```

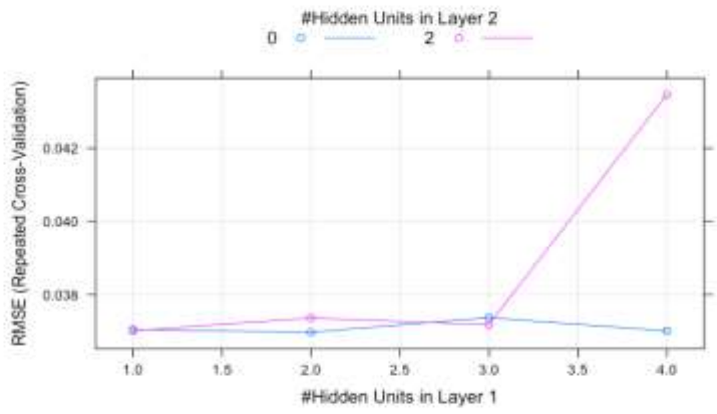
And calculate the RMSE

```
sqrt(mean((test$data.Price - pred)^2))
```

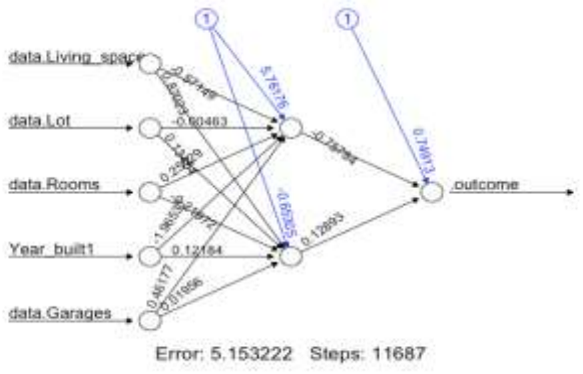
ANN - caret package

```
str(df.house)

set.seed(42)
tuGrid <- expand.grid(.layer1=c(1:4), .layer2=c(0,2), .layer3=c(0))

trCtrl <- trainControl(
  method = 'repeatedcv',
```

```r
  number = 5,
  repeats = 5,
  returnResamp = 'final'
)

models <- train(
  x = df.house %>% select(-data.Price),
  y = df.house_scaled %>% pull(data.Price),
  method = 'neuralnet', metric = 'RMSE',
  linear.output = TRUE,
  #be careful, does only work on x!
  preProcess = c('center', 'scale'),
  tuneGrid = tuGrid,
  trControl = trCtrl
)
plot(models)
plot(models$finalModel)
```



Plot neuralnet models



Neuralnet final model

For the interpretation “neuralnet” from caret package is used. Thus because, all the parameter combinations listed in expand.grid() were used to create and test a model. For the test the repeated cross-validation (5-Fold cross-validation repeated 5 times) were used to identify the final model with the best cross validation performance. The final model has 2 Neurons in layer and 0 in layer 2 (no layer 2).

Week 6 - Agent-based Modelling and Approximate Bayesian Computation

Here we define a Agent-based model that represents a real-world situation. After that we generate synthetic data with different parameters in a for loop. Agents (40-100), Pubs(1-20). The generated data will be saved in the file summary_stat.csv which will be used for the ABC in the next part.

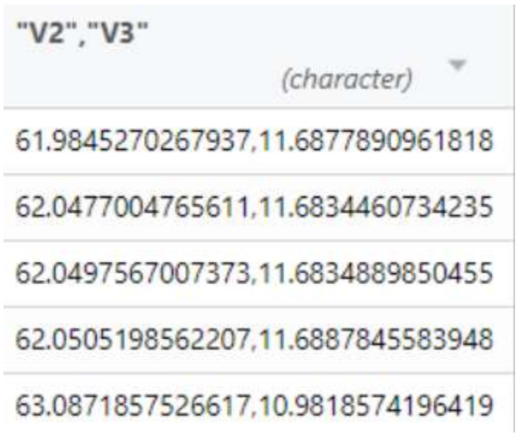
```
rm(list = ls(all = TRUE)) # clean the memory
install.packages("devtools")
devtools::install_github("PredictiveEcology/NetLogoR")

#libraries
library(NetLogoR)
library(stringr)
library(ggplot2)
library(minpack.lm)



for (i in 40:100){
  for (j in 1:20) {
    ### 1. DEFINE THE SPACE AND AGENTS ###

    # simulations parameters
    simtime<-100 # duration time of the simulation
    number_agents<-i
    number_pubs<-j # number of pubs (or homes. whatever) in the space named "world"
    gridSize_x<-10 # number of patches in the grid where moving agents move around
    gridSize_y<-10
    displacement_normal<-0.1 # speed of moving agents
    displacement_pub<-0.01 # if in the pub, agents move slower and spend more time there
    plot_data_out<-numeric() # initialize variable to store data to be plotted later on

    # world set up, this is about the static patches
    w1 <- createWorld(minPxcor = 0, maxPxcor = gridSize_x-1, minPycor = 0, maxPycor = gridSize_y-1) # worl
d defined by patches with coordinates Pxcor & Pycor
    x_pub<-randomPxcor(w1,number_pubs) # random pub location on the world grid
    y_pub<-randomPycor(w1,number_pubs)
    w1 <- NLset(world = w1, agents = patches(w1), val = 0) # initialize all the patches to their internal
state value = 0...
    w1 <- NLset(world = w1, agents = patch(w1, x_pub, y_pub), val = 1) # ...except for the pubs with a val
ue set to 1

    # agents set up, this is about the moving objects (traditionally named turtles)
    t1 <- createTurtles(n = number_agents, coords = randomXYcor(w1, n = number_agents), breed="S", color="
black") # all agents are set to the state (breed) S=susceptible, colored black
    t1 <- NLset(turtles = t1, agents = turtle(t1, who = 0), var = "breed", val = "I") # agent 0 is set to
I=infected (patient 0 that contaminates the others)
    t1 <- NLset(turtles = t1, agents = turtle(t1, who = 0), var = "color", val = "red") # ... and coloured
red
    t1 <- turtlesOwn(turtles = t1, tVar = "displacement", tVal = displacement_normal) # all initially move
with standard speed (normal displacement)

    plot(w1, axes = 0, legend = FALSE, par(bty = 'n')) # initialize graphics by displaying world patches
    points(t1, col = of(agents = t1, var = "color"), pch = 20) # initialize graphics by displaying agents



    ### 2. RUN THE SIMULATION TIME LOOP ###

    for (time in 1:simtime) { # start the simulation time loop

      t1 <- fd(turtles = t1, dist=t1$displacement, world = w1, torus = TRUE, out = FALSE) # each timestep
move each agent forward with the fd() function, by a distance
      t1 <- right(turtles = t1, angle = sample(-20:20, 1, replace = F)) # each timestep agents can randoml
y turn 20 deg right of 20 left (-20)

      plot(w1, axes = 0, legend = FALSE, par(bty = 'n')) # update graphics
      points(t1, col = of(agents = t1, var = "color"), pch = 20) # update graphics

      meet<-turtlesOn(world = w1, turtles = t1, agents = t1[of(agents = t1, var = "breed")=="I"]) # contac
t if multiple agents are on the same patch with function turtlesOn()
      t1 <- NLset(turtles = t1, agents = meet, var = "breed", val = "I") # get the state of the infected a
gent
      t1 <- NLset(turtles = t1, agents = meet, var = "color", val = "red") # and change its colour

      # agents that enter a pub spend more time there (have a lower displacement value)
      pub <- turtlesOn(world = w1, turtles = t1, agents = patch(w1, x_pub, y_pub)) # check if agent is on
a pub patch with function turtlesOn()
      # # if enters the pub
      t1 <- NLset(turtles = t1, agents = turtle(t1, who = pub$who), var = "displacement", val = displaceme
nt_pub)
      # # if exits the pub
      t1 <- NLset(turtles = t1, agents = turtle(t1, who = t1[-(pub$who+1)]$who), var = "displacement", val
= displacement_normal)


      Sys.sleep(0.1) # give some time to the computer to update all the thing graphically

      # store time-course data for plotting in the end
      contaminated_counter<-sum(str_count(t1$color, "red"))
      tmp_data<-c(time,contaminated_counter)
      plot_data_out<-rbind(plot_data_out, tmp_data) # store in a matrix

    }



    ### 3. PLOTTING AND FITTING SIMULATED DATA ###

    # perform non-linear curve fitting of the data
    df<-as.data.frame(plot_data_out)
    names(df)<-c("time","contaminated_counter")
    x  <- df$time
    y  <- df$contaminated_counter

    # give initial guesses and fit with 4-parameters logistic equation (fits well S-shaped generic curves)
    model <- nlsLM(y ~ d + (a-d) / (1 + (x/c)^b) ,start = list(a = 3, b = 4, c = 600, d = 1000)) # initial
guesses are set (arbitrarily) to 3,4,600,1000

    # make a line with the fitting model that goes through the data
    fit_x <- data.frame(x = seq(min(x),max(x),len = 100))
    fit_y <- predict(model, newdata = fit_x)
    fit_df <- as.data.frame(cbind(fit_x,fit_y))
    names(fit_df)<-c("x","y")
    fitted_function <- data.frame(x = seq(min(x),max(x),len = 100))
    lines(fitted_function$x,predict(model,fitted_function = fitted_function))

    # store summary statistics in a vector to be appended after each iteration to the output file
    # # put in the filename all the parameters used to set the simulation run
    simulation_run_name <- paste0("sim_",number_agents,"_",number_pubs)
    varied_params <- c(number_agents,number_pubs)
    summary_stat <- c( simulation_run_name, varied_params, as.vector(model$m$getPars()) )
    # save summary statistics of all the performed simulations in file with:
    # # Simulation ID
    # # parameters used for that simulation
    # # outcome of the curve (described by the fitting parameters)
    write.table(as.data.frame(t(summary_stat)), "./summary_stat.csv", sep = ",", col.names = FALSE, row.na
mes=FALSE, append = TRUE) # append to pile up the different runs in a single file

    # graphical representation of simulation data and fitting
    #ggplot(data = df, mapping = aes(y = y, x = x)) +
    #  ggtitle(paste0("Simulation ID: \t", simulation_run_name,
    #                 "\nSimulation Params: agents=",number_agents,"; pubs=", number_pubs,
    #                 "\nCurve Fit Params: \t",toString(round(model$m$getPars(),2)))) +
    #  xlab("time") +
    #  ylab("Agents contaminated") +
    #  geom_point(data=df, aes(y=y, x=x), colour="black") +
    #  geom_line(data = fit_df, colour="red")
    }}
```

Approximate Bayesian Computation (ABC) Code

In a next step we take some data observed in the real world and our Data Simulation from part one, which is saved in the summary_stat.csv. Then we want to find with Approximate Bayesian Computation (ABC) the parameters, which generated similar data in Part one with ABM to the data observed in the real world.

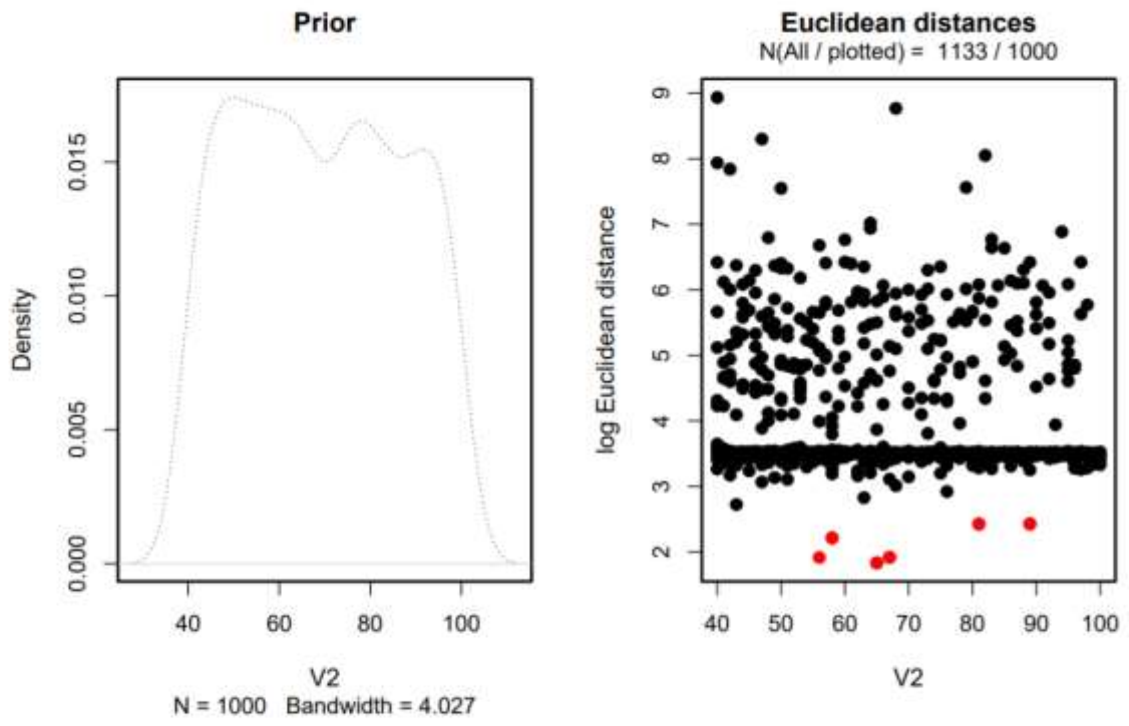
```
set.seed(2)
rm(list = ls(all = TRUE))
#install.packages("abc")
library(abc)

sum_stat <- read.csv(file='summary_stat.csv', header=FALSE)
head(sum_stat)

# import simulation and observed data
obs_data <- c(3,2,1143,1655)
sim_param <- sum_stat[,2:3]
sim_data <- sum_stat[,4:7]

# Run ABC
res <- abc(target=obs_data,
           param=sim_param,
           sumstat=sim_data,
           tol=0.005,
           transf=c("log"),
           method="neuralnet")

plot(res, param=sim_param)
write.table(res$adj.values,"out_abc.csv", sep=",", row.name=FALSE)
```

We get one plot output for the parameter number of agents and one plot output for the parameter number of pubs with prior and posterior distributions. We also get a table with the best parameter combinations(out_abc.csv). We repeated this code many times and we got always different best parameter combinations. But mostly the best parameters for number of agents was between 60 and 70 with numbers of pubs between 10 and 13 as we can see in the file out_abc.csv

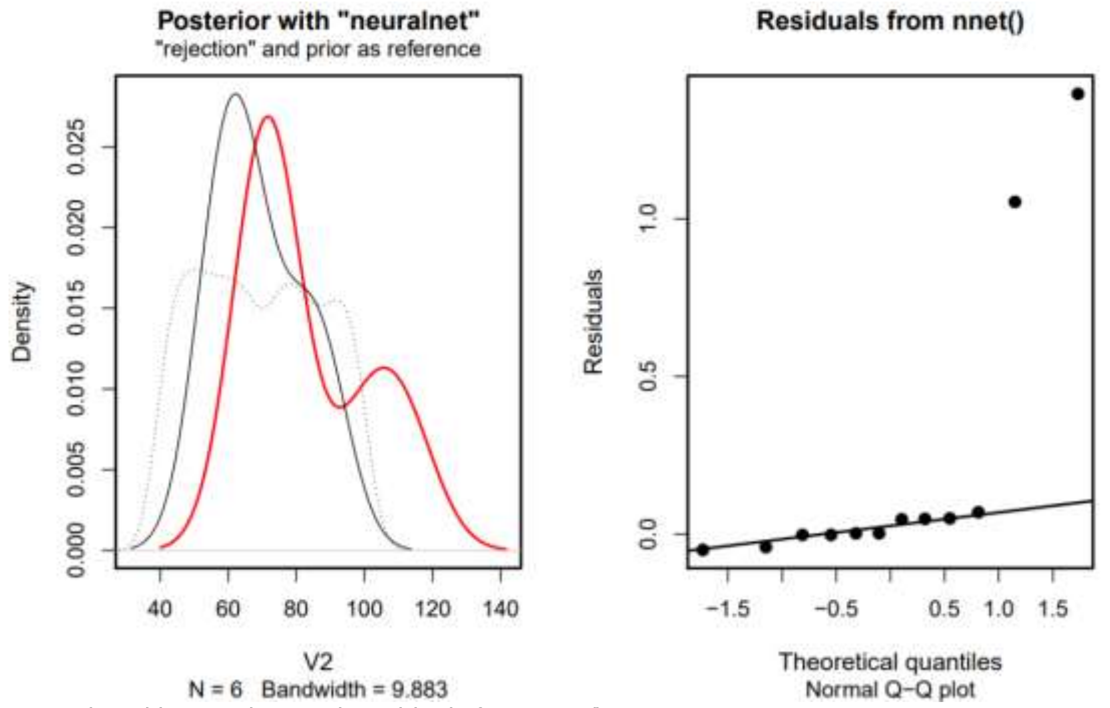


"V2","V3" (character)
61.9845270267937,11.6877890961818
62.0477004765611,11.6834460734235
62.0497567007373,11.6834889850455
62.0505198562207,11.6887845583948
63.0871857526617,10.9818574196419

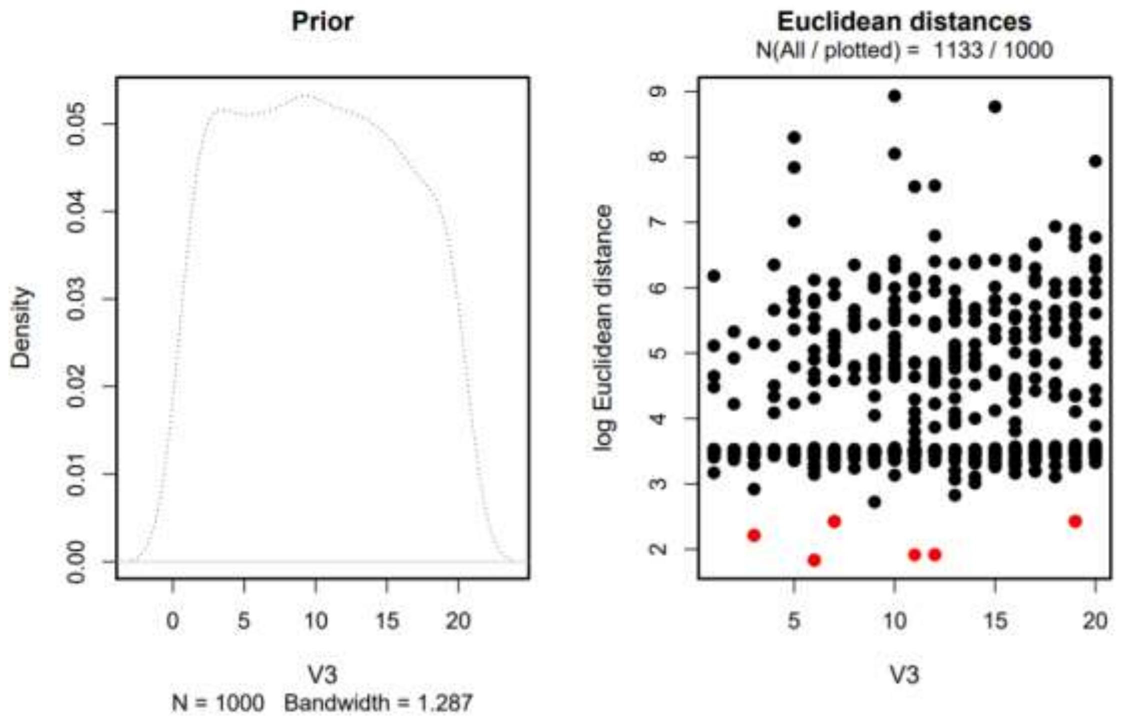
ABC_Output



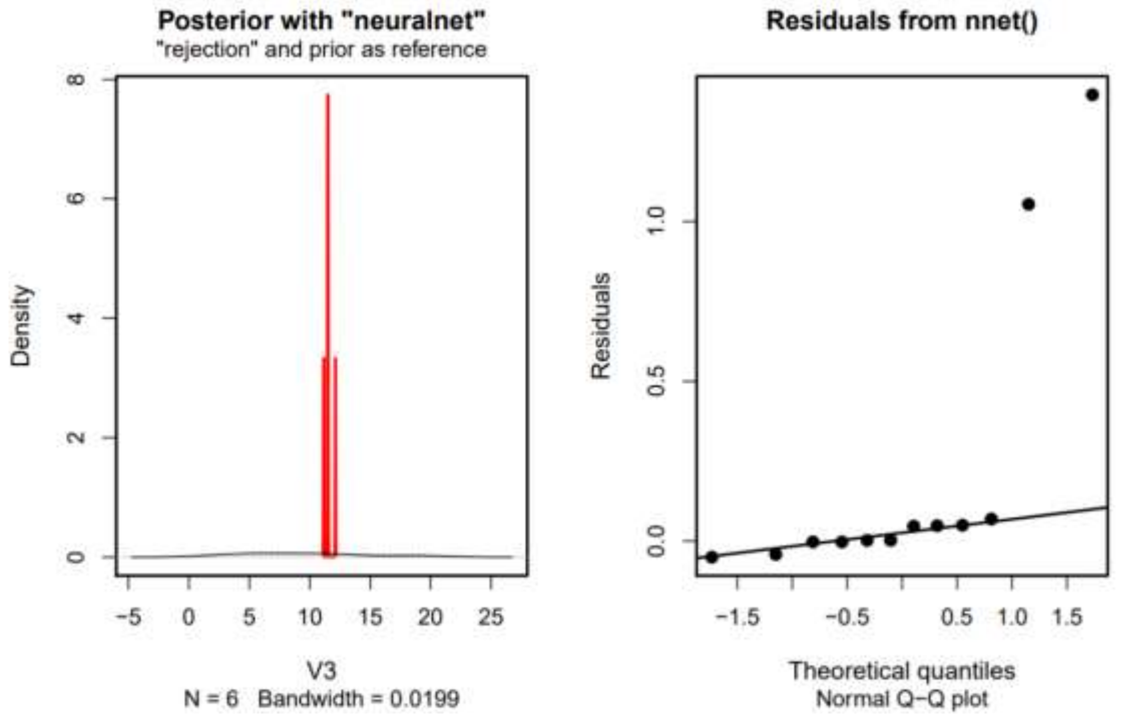
Prior and Euclidean distances_Agents



Posterior with neuralnet and Residuals from nnet()_Agents



Prior and Euclidean distances_Barplots



Posterior with neuralnet and Residuals from nnet()_Barplots

Conclusion

Week 1

Through applying a simple linear regression to our data, we found out that the living space has a strong influence on the price of house. This also makes sense very much, if you think of the real life, where the price always depends much on the square meters of a residence. Further, the analysis showed that by adding the Type of a house (e.g. Villa) the model can be slightly improved leading us to the conclusion that also the Type has an effect on the price.
Lastly, by an F-Test it could be confirmed that a model with levels for the condition of the house performs better and hence, influences the price of the house. The post-hoc contrast then showed that the condition refurbished and dilapidated differ in price (as also visible in the boxplots). However, in real life this seems to make no sense that a dilapidated house is priced higher.

Week 2

The graphical inspection of the predictor living space with the response variable price displayed that there might is a quadratic fit for living space. Therefore, a quadratic term for living space has been modelled. Then an F-Test confirmed the expected outcome that a model with a quadratic term better fits the data. Further, it was found out that adding a cubic term for living space does not improve the model but adds more complexity to it. Hence, the quadratic model seems to have the best fit.
Secondly, the predictor rooms also follows a non-linear relationship. The visual analysis showed that there is might a more complex relationship than before, which can't be modelled by a quadratic or cubic term. Therefore, the GAM-model has been chosen in order to find out the right degree of complexity. The GAM-model states that the best degree of complexity is 8. Whilst comparing with the initial plot these seems to be a valid result.

Week 3

In the third section a GLM-model was fitted to the count data rooms as response variable and the type as predictor. The outcome states that a house with no type has 7.6 rooms and the villa has 8.8 rooms. The simulation of the fitted GLM-model confirmed this outcome. Furthermore, a GLM-model with binomial data factor variable (i.e. cbind(Price ~ Living_space), State) has been compiled. The result showed an overdispersion, indicated by the difference in residual deviance and the degrees of freedom. Consequently, a quasibinomial has been built to try to handle this overdispersion, as the variance increases faster than linearly. Additionally, the p-value showed that there is no evidence that the predictor State influences the price. Moreover, a binary model with the response variable house prices over 1 Mio has been created with the predictor living space. Not surprisingly, the result showed that with increasing living space the probability for the house to be priced over 1 million gets higher.

Week 4

With the help of a SVM, we aimed to classify the houses into multiple dwelling and villa. The analysis shows that this is best modelled with a cost parameter of 0.5, as it has a prediction accuracy of 0.93. Thus, the classification works quite good, which is also visible in the plot of the analysis.
In a second SVM, it was aimed to classify a binary variable. Therefore, again the binary variable house price over 1 million was used with living space and year built as predictors. As the model does follow a non-linear relationship the radial kernel has been chosen. The result of the SVM shows an accuracy of 0.77, which tells us that 77.7% of data is classified correctly.

Week 5

In this section a neural network has been built to predict the price for a house considering all predictors of the dataset. In a first step, two hidden layers have been chosen with 1 hidden layer with 5 neurons and 1 hidden layer with 3 neurons. To evaluate the quality of this model the RMSE has been considered. As the RMSE shows a very high value (nearly 500'000) it is assumed that the neural network does not fit the data very well. Therefore, in a second step, the different combinations of the hidden layers have been evaluated against each other by using the caret package (cross-validation). The result showed that the combination of 2 neurons in the hidden layer 1 and 0 neurons in the hidden layer 2 delivers the best result.

Week 6

Regarding the topic agent-based modelling we performed a computational experiment, where we simulated with ABM all the combinations of pubs and agents to receive a simulated dataset. In a next step, the simulated data is compared with the actual observations by using the ABC-method. This allows us to select the parameters with the best fit for our observations.

Challenges

During the execution of the group work we encountered several challenges. First of all, we decided to use Github as a shared folder for our codes, which allowed us to simultaneously work on the project. However, the usage of Github with Rstudio requires some knowledge and technical skills, which is where we had difficulties especially in the beginning (e.g. merging files correctly). During the execution of the project we found out pretty fast that the models in practice do not deliver such nice results in theory and sometimes were difficult to interpret. Nevertheless, through this we had very useful group discussions and got more familiar with all the models. Lastly, the compilation of neural networks and agent-based modelling does not only require theoretical knowledge it as well requires some technical skills in Rstudio. In the beginning, we therefore encountered some difficulties to execute the codes.