ASML - Assignment - DSTI Autumn 2018

Alexandre Genette

July 2019

1 EXERCISE 1

From the **procespin** file, propose models between the 10 explanatory variables x_1,....,x_{10} and the response variable ln(y). Compare the different models that you can build and try to conclude.

1.1 Descriptive Statistics

First, we are going to study the explanatory variables and the target variable.

1.1.1 Load Data

Procespin file is loaded in R.

```
df <- read.table("procespin.txt", header=T)
```

1.1.2 Summary

We want ln(y) as a target variable we transform the target lny with log function then we make a summary of the data. We also remove the previous y column.

```
lny <- data.frame(log(df$y))
df <- cbind(lny, df)
names(df)[1] <- "lny"
df$y <- NULL
head(df)
summary(df)
```

```
      lny                x1             x2              x3              x4       
 Min.   :-3.5066   Min.   :1075   Min.   :15.00   Min.   : 0.00   Min.   :2.400  
 1st Qu.:-1.7148   1st Qu.:1228   1st Qu.:24.00   1st Qu.: 4.00   1st Qu.:3.700  
 Median :-0.4005   Median :1309   Median :28.00   Median : 8.00   Median :4.400  
```

```
 Mean   :-0.8133   Mean   :1315   Mean   :28.73   Mean   :11.45   Mean   :4.452
 3rd Qu.: 0.1222   3rd Qu.:1396   3rd Qu.:32.00   3rd Qu.:18.00   3rd Qu.:5.300
 Max.   : 1.0986   Max.   :1575   Max.   :46.00   Max.   :32.00   Max.   :6.500

       x5              x6              x7              x8
 Min.   : 5.80   Min.   :1.000   Min.   :1.100   Min.   : 3.600
 1st Qu.:11.50   1st Qu.:1.200   1st Qu.:1.600   1st Qu.: 5.900
 Median :15.70   Median :1.500   Median :1.700   Median : 7.200
 Mean   :15.25   Mean   :1.791   Mean   :1.658   Mean   : 7.539
 3rd Qu.:18.30   3rd Qu.:2.400   3rd Qu.:1.800   3rd Qu.: 9.100
 Max.   :21.80   Max.   :3.300   Max.   :1.900   Max.   :13.700

       x9              x10
 Min.   :1.100   Min.   :1.300
 1st Qu.:1.500   1st Qu.:1.600
 Median :2.000   Median :1.800
 Mean   :1.982   Mean   :1.752
 3rd Qu.:2.500   3rd Qu.:2.000
 Max.   :2.900   Max.   :2.000
```

1.1.3 Missing Values

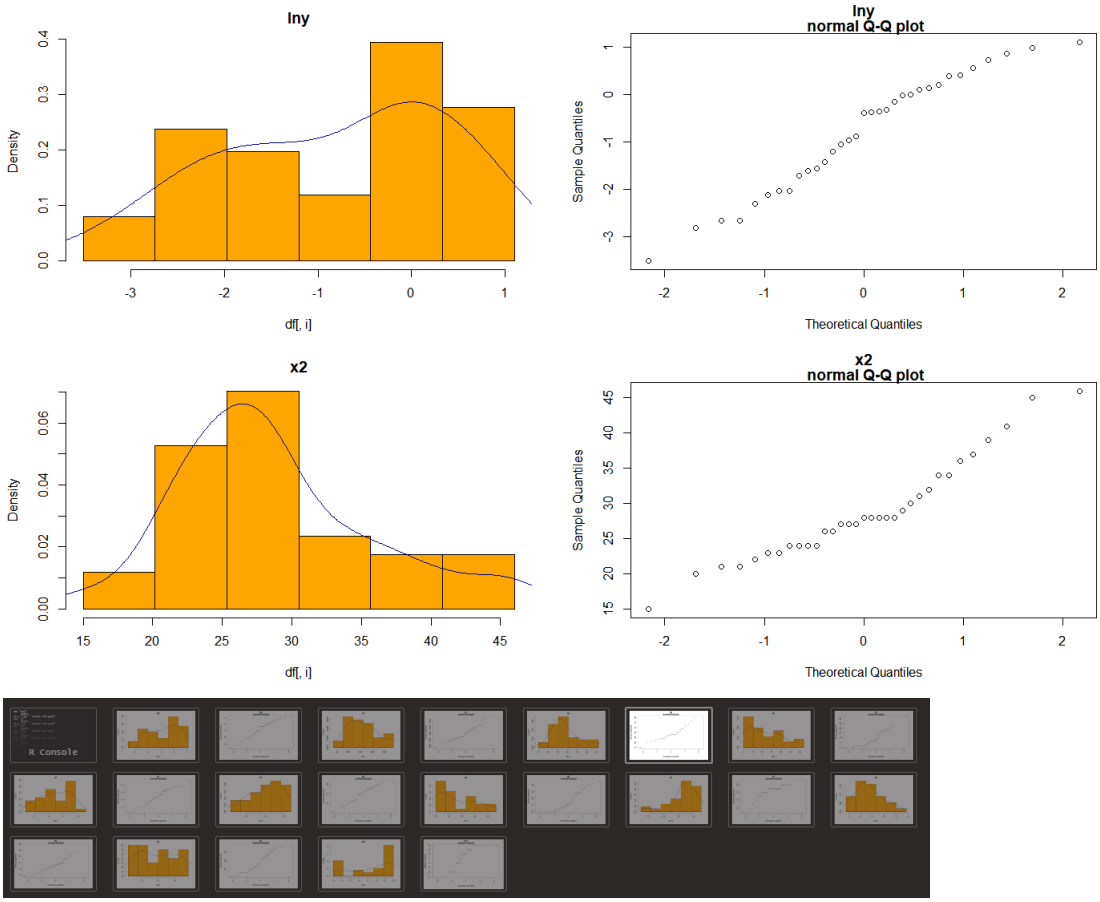
With the help of **library(mice)** we check if there are **missing values** in the dataset. We can conclude there is no missing value in the dataset.

```
library(mice)
md.pattern(df, plot = F)
 /\     /\
{  `---'  }
{  0   0  }
==>  V <==  No need for mice. This data set is completely observed.
 \  \|/  /
  `-----'
```

1.1.4 QQPlot and Histogram

We continue our **descriptive** analytics by plotting **QQPLOT and distribution** of the variables.

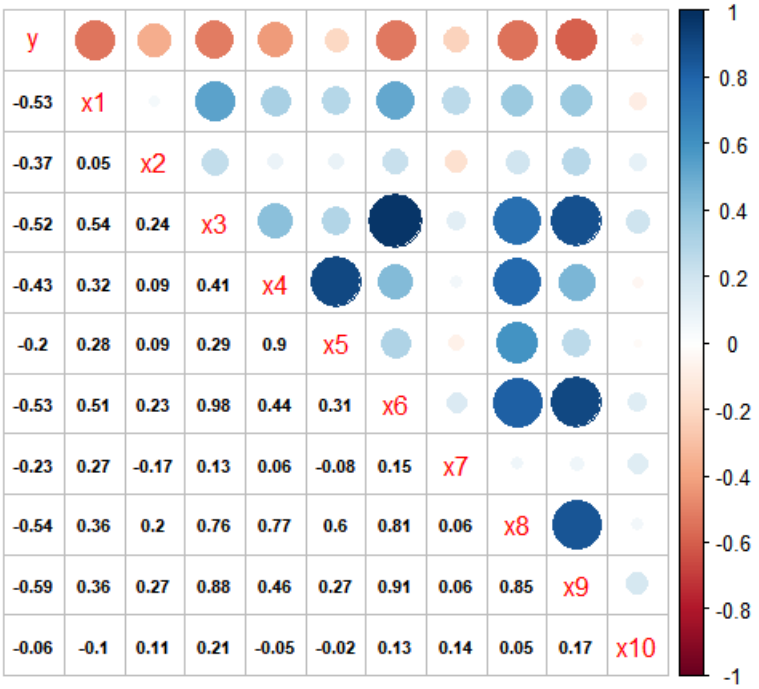
```
bins <- round(sqrt(33)) + 1
for (i in colnames(df)) {
  hist(df[,i], main = i, breaks = seq(min(df[,i]),max(df[,i]),l = bins), col="orange", freq=F) #sqrt of observati
  lines(density(df[,i], adjust=1), col="darkblue")
  charT <- as.character(i)
  print(charT)
  txt <-  cbind(charT, " normal Q-Q plot")
  print(txt)
  qqnorm(df[,i], main = txt)
}
```

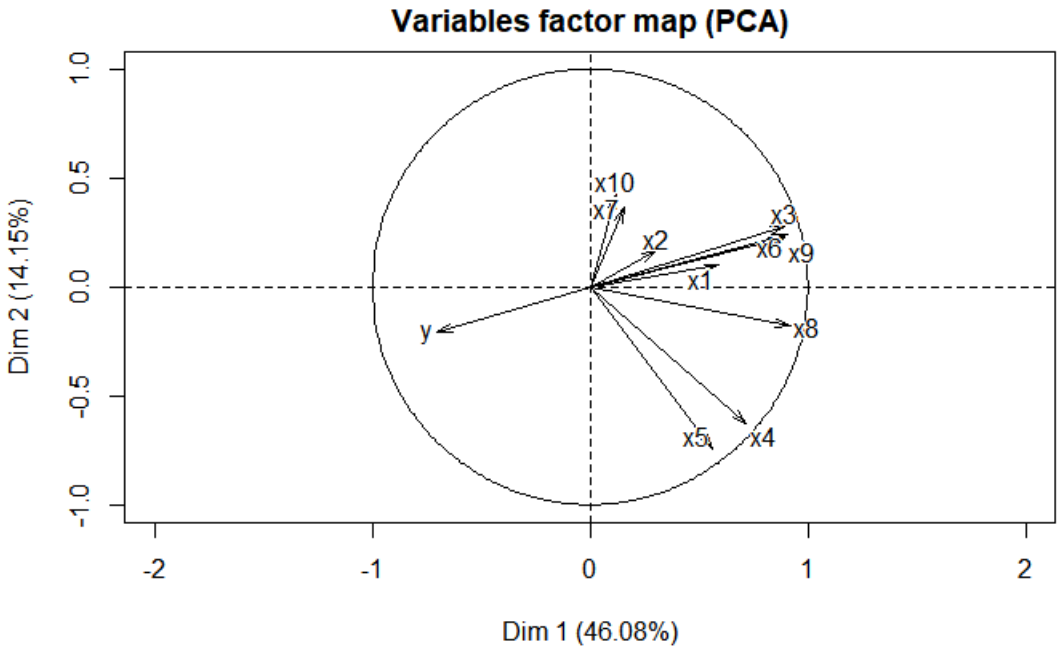


1.1.5 Variable Correlation

We are looking for the correlation between variables with the cor function, we can plot a correlation matrix and plot a PCA of the variables.

```
my.cor <- cor(df)
corrplot.mixed(my.cor, lower.col = "black", number.cex = 0.7)
c <- c()
for (i in colnames(df)) {
  test <- class(df[[i]])
  ifelse(test == "numeric" || test == "integer"  , c <- c(c, i), c)
}

a <- PCA(df, scale.unit = T, ncp = 2, graph = TRUE)
```





We see that variables x_3, x_6 and x_9 are highly correlated. x_5 and x_4 too. (PCA plot confirms the correlation matrix)

1.2 Creation of a Linear Model

A full model is first created using all the explanatory variables. We add some plot in order to have a good overview of the modelization. A Root Mean Square Error (RMSE) is calculated.

Formula of the root mean square error:
$\text{RMSE} = \sqrt{\frac{1}{N}\Sigma_{i=1}^{n}\Big(Predicted_i - Observed_i\Big)^2}$

```
model_LM <- lm(lny ~ ., data = df)
model_LM
plot(model_LM)
predict_LM <- predict(model_LM, newdata = df)
rmse_LM <- RMSE(df$lny, predict_LM)
rmse_LM
plot(predict_LM ~ df$lny)
legend <- sprintf("RMSE_LM %s", round(rmse_LM,3))
legend("topleft", legend = legend)

OUTPUT:
Coefficients:
(Intercept)           x1           x2           x3           x4           x5           x6
  10.984915    -0.004566    -0.053066     0.057882    -1.351310     0.247549    -0.286816
     x7           x8           x9          x10
 -0.156044     0.165293    -1.177886    -0.474487

 RMSE : 0.6804471
```

1.2.1 VARIABLE SELECTION

We make a variable selection with an ascending step using Akaike Information Criterium (AIC).

```
model_FULL <- lm(lny ~ ., data = df)
AIC <- step(model_FULL)
AIC

BEST MODEL SELECTED:
Call:
lm(formula = lny ~ x1 + x2 + x4 + x5, data = df)

Coefficients:
(Intercept)           x1           x2           x4           x5
   8.093589    -0.004124    -0.059002    -1.409906     0.294310
```

The best model selected is the one using only 4 variables x_1, x_2, x_4 and x_5.

1.2.2 CROSSVALIDATION

In order to be sure that the best model has been selected I am going to compare different models with crossvalidation (with library cvTools) method because we do not have a lot of observations (only 33) to split the rows into a train and a test dataset. Crossvalidation will randomly split several times the dataset to avoid overfitting. Models are created from modelLMO to modelLM5 with a different set of variables each time. An ANOVA is also calculated between different models, the lowest score will be the best model.

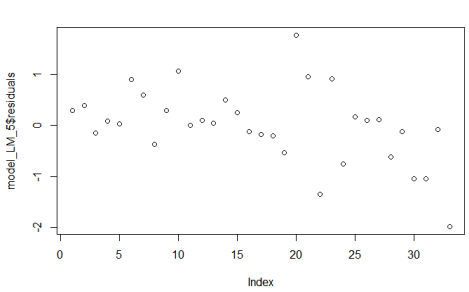
```
model_LM_0 <- lm(lny ~ ., data = df)
repCV(model_LM_0, K = 5)
model_LM_1 <- lm(lny ~ x9 + x4 + x5 + x2 + x1, data = df)
repCV(model_LM_1, K = 5)
model_LM_2 <- lm(lny ~ x9 + x4 + x5 + x2 + x1 + x3, data = df)
repCV(model_LM_2, K = 5)
model_LM_3 <- lm(lny ~ x9 + x4 + x5 + x2 + x1 + x3 + x10, data = df)
repCV(model_LM_3, K = 5)
model_LM_4 <- lm(lny ~ x3 + x1 + x8 + x4, data = df)
repCV(model_LM_4, K = 5)
model_LM_5 <- lm(lny ~ x1 + x2 + x4 + x5, data = df)
repCV(model_LM_5, K = 5)

anova(model_LM_0, model_LM_1)
anova(model_LM_0, model_LM_2)
anova(model_LM_0, model_LM_3)
anova(model_LM_0, model_LM_4)
anova(model_LM_0, model_LM_5)
anova(model_LM_1, model_LM_5)
anova(model_LM_4, model_LM_5)
fits <- list(model_LM_0 = model_LM_0, model_LM_1 = model_LM_1,model_LM_2 = model_LM_2,model_LM_3 = model_LM_3, mo
print(sapply(fits, AIC))
plot(model_LM_5)

OUPUT:
model_LM_0 model_LM_1 model_LM_2 model_LM_3 model_LM_4 model_LM_5
  92.23960   85.44240   86.07359   87.07518  100.82854   84.46463
```

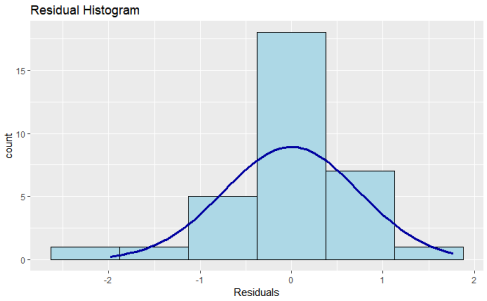
With all these results we can choose the best model as the following one:

lny $\rightarrow$ x1 + x2 + x4 + x5

We can plot the residuals of this linear model

```
library(olsrr)
plot(model_LM_5$residuals)
mean(model_LM_5$residuals)
qqnorm(model_LM_5$residuals)
ols_test_normality(model_LM_5)
ols_plot_resid_hist(model_LM_5)
```





It seems that the residuals follow a normal distribution, which is good for the modelization.

1.3 RMSE comparison

In order to compare the full model to the one with only 4 variables I will compare RMSE via a loop. 70 percent of the data set is randomly selected as a training set and the rest is used as validation and RMSE calculation. This is is done 100 times.

```
rmse_LM <- c()
rmse_LMBEST <- c()
rmse_FULL <- c()

for (i in 1:100) {
  set.seed(i*10+1)
  train <- sample(nrow(df), 0.7*nrow(df), replace = FALSE)
  train_Set <- df[train,]
  test_Set <- df[-train,]

  ##############################################################################
  model_LMBEST <- lm(lny ~ x4 + x5 + x2 + x1, data = train_Set)
  predict_LMBEST <- predict(model_LMBEST, newdata = test_Set[,-1])
  rmse_tmp <- RMSE(pred = predict_LMBEST, obs = test_Set$lny)
  rmse_LMBEST <- cbind(rmse_LMBEST, rmse_tmp)
  ##############################################################################
  model_FULL <- lm(lny ~ ., data = train_Set)
  predict_FULL <- predict(model_FULL, newdata = test_Set[,-1])
  rmse_tmp <- RMSE(pred = predict_FULL, obs = test_Set$lny)
  rmse_FULL <- cbind(rmse_FULL, rmse_tmp)
}
LM_best <- c(mean(rmse_LMBEST), sd(rmse_LMBEST), min(rmse_LMBEST), max(rmse_LMBEST))
LM_full <- c(mean(rmse_FULL), sd(rmse_FULL), min(rmse_FULL), max(rmse_FULL))
a <- data.frame(LM_full)
a <-cbind(a, LM_best)
rownames(a) <- c("RMSE", "SD", "MIN", "MAX")
```

```
a

OUTPUT:
          LM_full      LM_best
RMSE    1.1124775    0.8474308
SD      0.3186248    0.2111859
MIN     0.3770222    0.3613887
MAX     2.5377300    1.5519846
```

We have confirmation the model with 4 variables is better than the complete model.

1.3.1 SUMMARY

To summarize, we have the linear model with variables x1, x2, x4 and x5 and their respective coefficients :

(Intercept)	x1	x2	x4	x5
8.094	-0.004	-0.059	-1.410	0.294

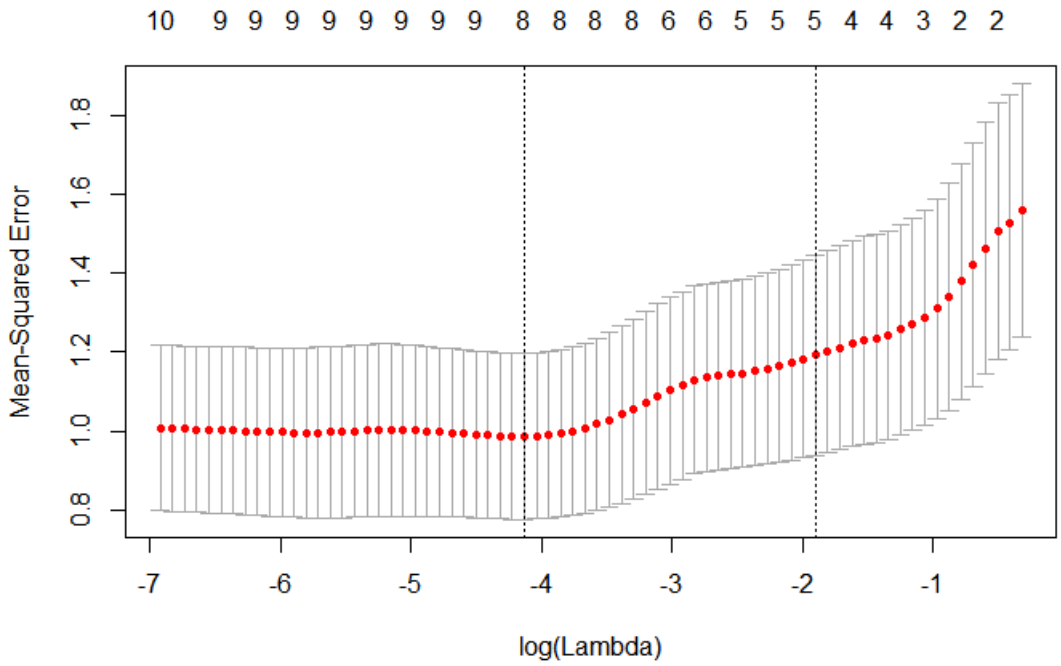
The formula is:

$$log(y) = -0.004 * x_1 - 0.059 * x_2 - 1.410 * x_4 + 0.294 * x_5 + 8.094$$

1.4 OTHER MODELS

We make a variable selection with LASSO.

```
lasso <- cv.glmnet(as.matrix(df[,2:11]), as.matrix(df$lny), alpha = 1)
coef(lasso)
plot(lasso)
```

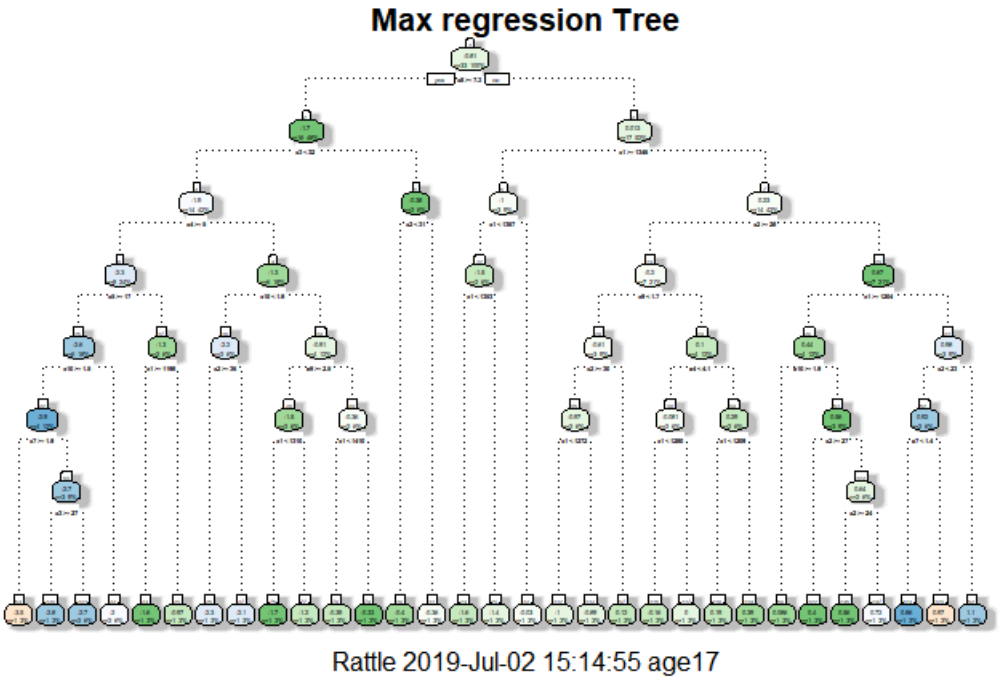


We obtain the best results keeping 8 variables.

1.4.1 DECISION TREE

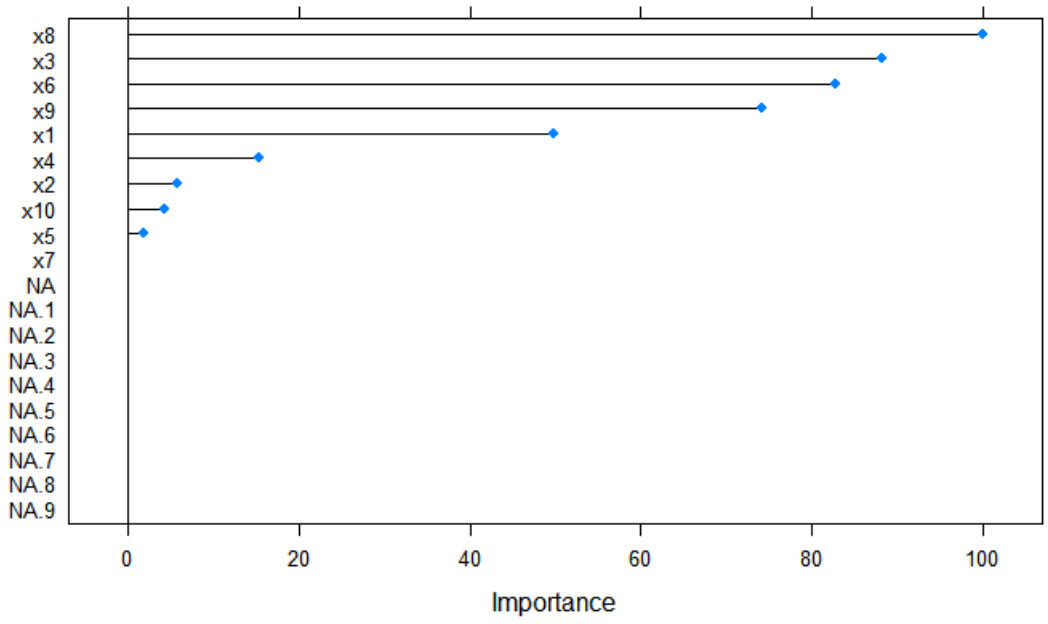
A full decision tree is built first, function rpart() from the library *rpart*.

```
mtree <- rpart(lny ~ ., data = df, cp = 0, minbucket = 1, minsplit=1, method = 'anova')
mtree
plot(mtree)
text(mtree)
fancyRpartPlot(mtree, uniform=TRUE, main="Max regression Tree",)
plotcp(mtree)
```



I use the bagging method to find the moste relevant explanatory variables.

```
bagged_dt <- train(lny ~ ., data = df,  method = "treebag",  importance = TRUE  )
# assess results
bagged_dt
# plot most important variables
plot(varImp(bagged_dt), 20)
```



We see that x_5 and x_7 have a slight influence on the model.

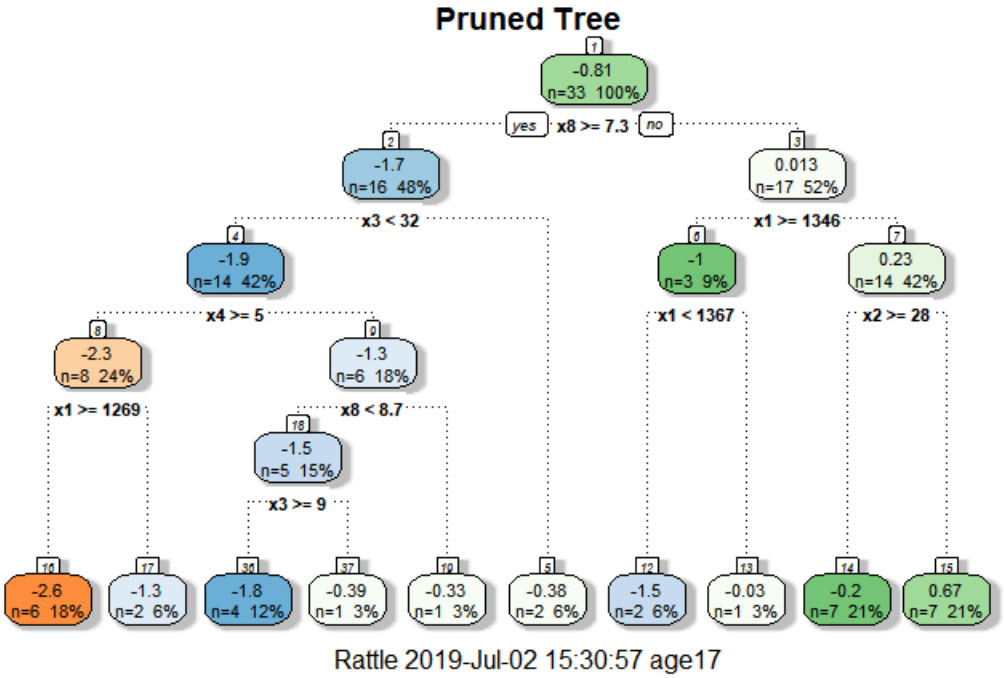
```
bagged_dt <- train(lny ~ ., data = df,  method = "treebag",  importance = TRUE  )
# assess results
bagged_dt
# plot most important variables
plot(varImp(bagged_dt), 20)
```

Using the LASSO and the BAGGING method we can keep the 8 most important explanatory variables and build a model according to these parameters.

```
mtree <- rpart(lny ~ x1 + x8 + x3 + x6 + x9 + x4 + x2 + x5, data = df, cp = 0, minbucket = 1, minsplit=1, method
mtree
plot(mtree)
text(mtree)
fancyRpartPlot(mtree, uniform=TRUE, main="Max regression Tree",)
plotcp(mtree)
```

We, now, prune the decision tree in order to reduce it. cp parameter is tweaked in order to find the best tree.

```
ptree<- prune(mtree, cp= 0.02)
fancyRpartPlot(ptree, uniform=TRUE, main="Pruned Tree")
plotcp(ptree)
```

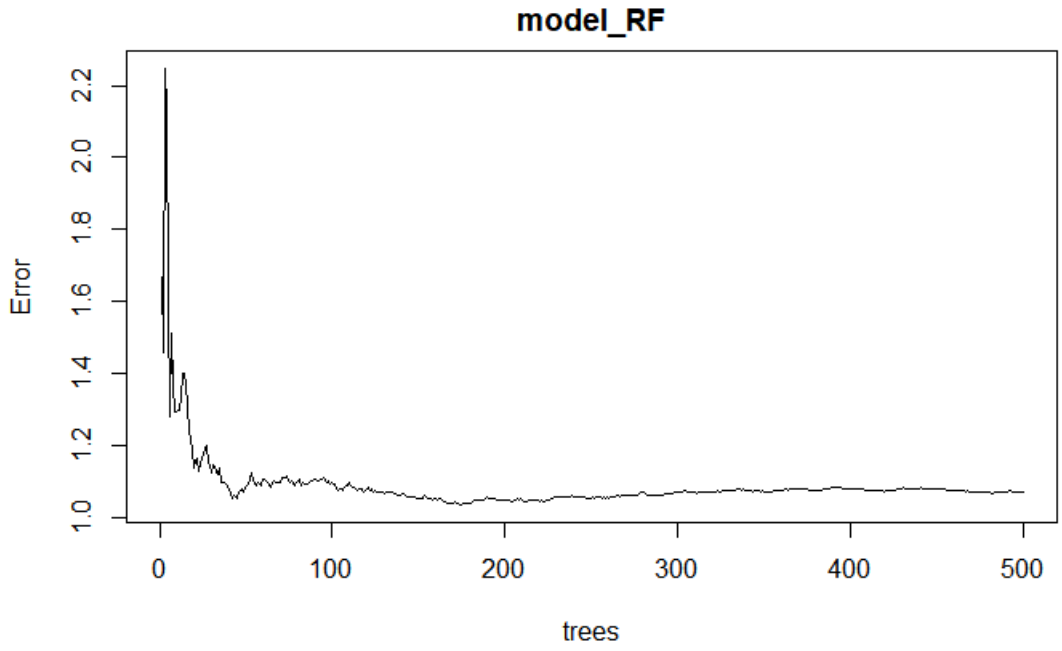


The best tree is this one.

1.4.2 Random Forest

We can do the same with the Random Forest.

```
model_RF <- randomForest(lny ~ x1 + x8 + x3 + x6 + x9 + x4 + x2 + x5, data = df)
summary(model_RF)
plot(model_RF)
```



By default 500 trees are calculated but with the graph we see that 200 iterations are enough to find the best tree.

1.4.3 CONCLUSION and Model comparison

With this dataset we have built 3 different models: One linear model, one decision tree and one random forest. We have not a lot of observations (only 33) and 10 explanatory variables. The linear model is simpler than the other two because we only keep 4 variables. I will use RMSE as metrics in order to compare the models together. I will run a loop as seen above and make a mean.

```
rmse_DT <- c()
rmse_RF <- c()
rmse_LM2 <- c()

for (i in 1:20) {
  set.seed(i*10+1)
  train <- sample(nrow(df), 0.7*nrow(df), replace = FALSE)
  train_Set <- df[train,]
  test_Set <- df[-train,]
```

```
  ##############################################################################
  model_DT <- rpart(lny ~ x1 + x8 + x3 + x6 + x9 + x4 + x2 + x10, data = train_Set, cp = 0, minbucket = 1, minspl
  prune_DT <- prune(model_DT, cp= 0.02)
  predict_DT <- predict(prune_DT, newdata = test_Set[,-1], type = 'vector')
  rmse_tmp <- RMSE(pred = predict_DT, obs = test_Set$lny)
  rmse_DT <- cbind(rmse_DT, rmse_tmp)
  ##############################################################################
  model_RF <- randomForest(lny ~ x1 + x8 + x3 + x6 + x9 + x4 + x2 + x10 , data = train_Set, ntree = 500)
  predict_RF <- predict(model_RF, newdata = test_Set[,2:11])
  rmse_tmp <- RMSE(test_Set$lny, predict_RF)
  rmse_RF <- cbind(rmse_RF, rmse_tmp)
  ##############################################################################
  model_LM2 <- lm(lny ~ x4 + x5 + x2 + x1, data = train_Set)
  predict_LM2 <- predict(model_LM2, newdata = test_Set[,-1])
  rmse_tmp <- RMSE(pred = predict_LM2, obs = test_Set$lny)
  rmse_LM2 <- cbind(rmse_LM2, rmse_tmp)

}
```

```
        DT                LM_best          RF
RMSE    1.4114581         0.8571533        1.0208152
SD      0.2167022         0.2270368        0.1976399
MIN     1.0353301         0.3613887        0.6516065
MAX     1.8724488         1.0979258        1.3621549
```

According to the RMSE(root mean square error) the LINEAR MODEL SEEMS TO BE THE BEST SO FAR.

2 EXERCISE 2

For this exercise I have chosen a Dataset from kaggle. This dataset contains features and price of houses sold in Ames, Iowa between 2006 and 2010. There are 79 different explanatory variables, some categorical(zoning...) others numerical(surface...). The training set contains 1460 rows that we can easily divide as a train set and a test set. The target is a continuous variable so, as model, we need a **regressor** not a classifier.

The purpose of this exercise is to build the best model that could predict the best selling price of a house.

The dataset can be downloaded here : https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

DESCRIPTION OF THE EXPLANATORY VARIABLES:

SalePrice - the property's sale price in dollars. This is the target variable.
MSSubClass: The building class
MSZoning: The general zoning classification
LotFrontage: Linear feet of street connected to property
LotArea: Lot size in square feet
Street: Type of road access
Alley: Type of alley access
LotShape: General shape of property
LandContour: Flatness of the property
Utilities: Type of utilities available
LotConfig: Lot configuration
LandSlope: Slope of property
Neighborhood: Physical locations within Ames city limits
Condition1: Proximity to main road or railroad
Condition2: Proximity to main road or railroad (if a second is present)
BldgType: Type of dwelling
HouseStyle: Style of dwelling
OverallQual: Overall material and finish quality
OverallCond: Overall condition rating
YearBuilt: Original construction date
YearRemodAdd: Remodel date
RoofStyle: Type of roof
RoofMatl: Roof material
Exterior1st: Exterior covering on house
Exterior2nd: Exterior covering on house (if more than one material)
MasVnrType: Masonry veneer type
MasVnrArea: Masonry veneer area in square feet
ExterQual: Exterior material quality
ExterCond: Present condition of the material on the exterior
Foundation: Type of foundation
BsmtQual: Height of the basement
BsmtCond: General condition of the basement
BsmtExposure: Walkout or garden level basement walls
BsmtFinType1: Quality of basement finished area
BsmtFinSF1: Type 1 finished square feet
BsmtFinType2: Quality of second finished area (if present)
BsmtFinSF2: Type 2 finished square feet
BsmtUnfSF: Unfinished square feet of basement area
TotalBsmtSF: Total square feet of basement area
Heating: Type of heating
HeatingQC: Heating quality and condition
CentralAir: Central air conditioning
Electrical: Electrical system
1stFlrSF: First Floor square feet
2ndFlrSF: Second floor square feet
LowQualFinSF: Low quality finished square feet (all floors)
GrLivArea: Above grade (ground) living area square feet
BsmtFullBath: Basement full bathrooms
BsmtHalfBath: Basement half bathrooms
FullBath: Full bathrooms above grade
HalfBath: Half baths above grade
Bedroom: Number of bedrooms above basement level
Kitchen: Number of kitchens
KitchenQual: Kitchen quality
TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
Functional: Home functionality rating
Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality
GarageType: Garage location
GarageYrBlt: Year garage was built
GarageFinish: Interior finish of the garage
GarageCars: Size of garage in car capacity
GarageArea: Size of garage in square feet
GarageQual: Garage quality
GarageCond: Garage condition
PavedDrive: Paved driveway
WoodDeckSF: Wood deck area in square feet
OpenPorchSF: Open porch area in square feet
EnclosedPorch: Enclosed porch area in square feet
3SsnPorch: Three season porch area in square feet
ScreenPorch: Screen porch area in square feet
PoolArea: Pool area in square feet
PoolQC: Pool quality
Fence: Fence quality
MiscFeature: Miscellaneous feature not covered in other categories
MiscVal: Value of miscellaneous feature
MoSold: Month Sold
YrSold: Year Sold
SaleType: Type of sale
SaleCondition: Condition of sale

2.1 Preprocessing Data

2.1.1 Loading and Cleaning Data

First, the dataset is loaded and a summary is done in order to characterize more precisely all the explanatory variables and the target (Price of the sold house).

```
df <- read.csv2("train.csv", header = T, sep = ",")
```

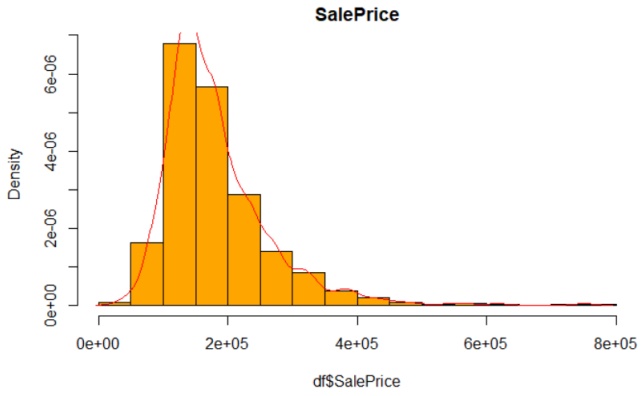
We remove the column "Id" because there is obviously no relationship between the price and the Id of the transaction.

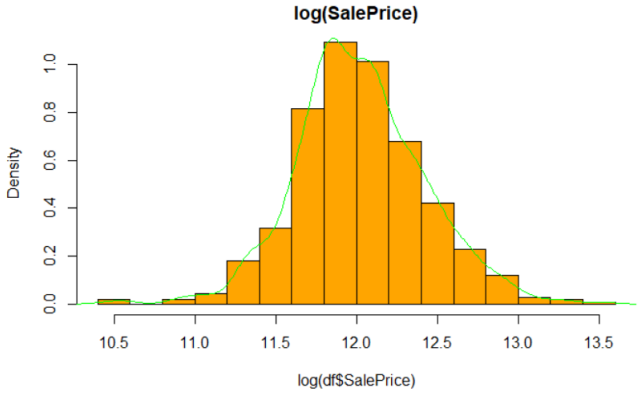
We observe numerical variables and categorical variables, on the other hand, some numerical variables can be treated as categorical variables, such as Year or Month of the Sale, some Quality Scale (OverallCond) or Fireplaces... We convert these features into categorical variables.

```
df$MoSold <- factor(df$MoSold)
df$YrSold <- factor(df$YrSold)
df$MSSubClass <- factor(df$MSSubClass)
df$OverallCond <- factor(df$OverallCond)
df$OverallQual <- factor(df$OverallQual)
df$GarageCars <- factor(df$GarageCars)
```

```
df$Fireplaces <- factor(df$Fireplaces)
df$FullBath <- factor(df$FullBath)
df$HalfBath <- factor(df$HalfBath)
```

After plotting the Price distribution we see the target has not a normal distribution. I apply a natural log transformation to the target.





We also notice that the variable Utilities is completely unbalanced with 2 rows and 1458 rows for 2 categories. This feature is eliminated.

After checking for missing values I decide to make imputation. For the categorical variables NA's are imputed as "None", for example missing values for the variable GarageType are houses that do not have a garage.

For the numerical variables, the missing values are replaced by the median of the serie.

```
for (i in c("Alley", "MasVnrType", "BsmtQual", "BsmtCond", "BsmtExposure", "BsmtFinType2", "FireplaceQu",
"GarageQual","GarageCond", "PoolQC","Fence","MiscFeature", "GarageFinish", "GarageType",  "BsmtFinType1", "Electr
"Functional", "MSZoning","SaleType", "KitchenQual", "Condition2", "Condition1", "Exterior2nd")) {
  df[,i] <- fct_explicit_na(df[,i], "None")
 }
df[,"Exterior1st"] <- fct_explicit_na(df[,"Exterior1st"], "VinylSd")

df$YearBltCat <- car::recode(df$YearBuilt, "1800:1899=1;  1900:1930=2; 1931:1945=3; 1946:1960=4; 1961:1975=5;
1976:1990=6; 1991:2000=7;else=8")
summary(df$YearBltCat)
df$YearBltCat <- as.factor(df$YearBltCat)
df$YearBuilt <- NULL

df$YearRemoCat <- car::recode(df$YearRemodAdd, "1800:1899=1;  1900:1930=2; 1931:1945=3; 1946:1960=4;
1961:1975=5;1976:1990=6; 1991:2000=7;else=8")
summary(df$YearRemoCat)
df$YearRemoCat <- as.factor(df$YearRemoCat)
df$YearRemodAdd <- NULL


df$YearGrgBltCat <- car::recode(df$GarageYrBlt, "1800:1899=1;  1900:1930=2; 1931:1945=3; 1946:1960=4;
1961:1975=5;1976:1990=6; 1991:2000=7;else=8")
summary(df$YearGrgBltCat)
df$YearGrgBltCat <- as.factor(df$YearGrgBltCat)
df$GarageYrBlt <- NULL
df$MasVnrArea[is.na(df$MasVnrArea)] <- median(df$MasVnrArea, na.rm = TRUE)
df$LotFrontage[is.na(df$LotFrontage)] <- median(df$LotFrontage, na.rm = TRUE)
df$BsmtFinSF1[is.na(df$BsmtFinSF1)] <- median(df$BsmtFinSF1, na.rm = TRUE)
df$BsmtFinSF2[is.na(df$BsmtFinSF2)] <- median(df$BsmtFinSF2, na.rm = TRUE)
df$BsmtUnfSF[is.na(df$BsmtUnfSF)] <- median(df$BsmtUnfSF, na.rm = TRUE)
df$TotalBsmtSF[is.na(df$TotalBsmtSF)] <-median(df$TotalBsmtSF, na.rm = TRUE)
df$BsmtFullBath[is.na(df$BsmtFullBath)] <- median(df$BsmtFullBath, na.rm = TRUE)
df$BsmtHalfBath[is.na(df$BsmtHalfBath)] <- median(df$BsmtHalfBath, na.rm = TRUE)
df$GarageCars[is.na(df$GarageCars)] <- 0
df$GarageArea[is.na(df$GarageArea)] <- 0
```

2.1.2 Splitting into Train and Test set

The train set will serve for the training of the model while the test set will serve for the validation of the model. The split chosen is 80/20. The seed chosen is 1234.

```
set.seed(1234)
train <- sample(nrow(df), 0.8*nrow(df), replace = FALSE)
train_Set <- df[train,]
test_Set <- df[-train,]
```

2.1.3 Feature Scaling

Because we have a lot of categorical variables I will get a lot of dummy variables (factor equals to 0 or 1) I then apply a feature scaling to numerical values in order to scale my data set. This may improve the model especially when there is a factor of 10,100 or 1000 between numerical explanatory variables. For the scaling of the test set we use the min and max argument of the train set, it is not allowed to stack the two datasets and apply a scaling to the whole dataset.

The scaling method I choose is the following:

$$\text{ScaledValue}(x_i|feature) = \frac{min(feature) - x_i}{min(feature) - max(feature)}$$

For all numerical values I obtain scaled values between 0 and 1.

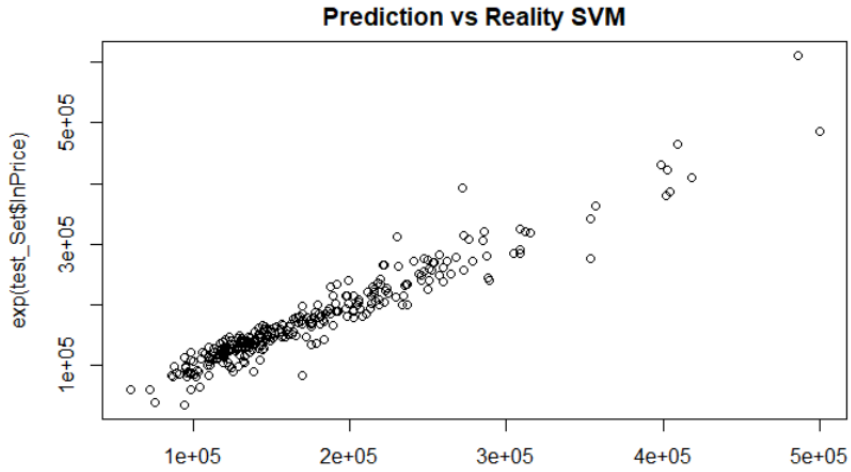
```
#FEATURE SCALING
scaleFct <- function(x,y) { (min(x) - y) / (min(x) - max(x))}
scale1 <- train_Set
scale2 <- test_Set

for (i in colnames(test_Set[1:78]))  {

  if (class(test_Set[,i]) != "factor") {
    train_Set[i] <- scaleFct(scale1[i], scale1[i])
    test_Set[i] <- scaleFct(scale1[i], scale2[i])
  }
}
```

Now we have cleaned, filtered and transformed our data in the best format we are ready for the model.

2.2 Support Vector Machine Model

Support-vector Machine algorithm (SVM) is a popular machine learning tool identified by Vladimir Vapnik al. in 1992. It is implemented with the library(svm) and the function svm in R. The prediction is then compared with RMSE (roo mean square error) to the exponential target (because we had a previous log transformation on the SalePrice.

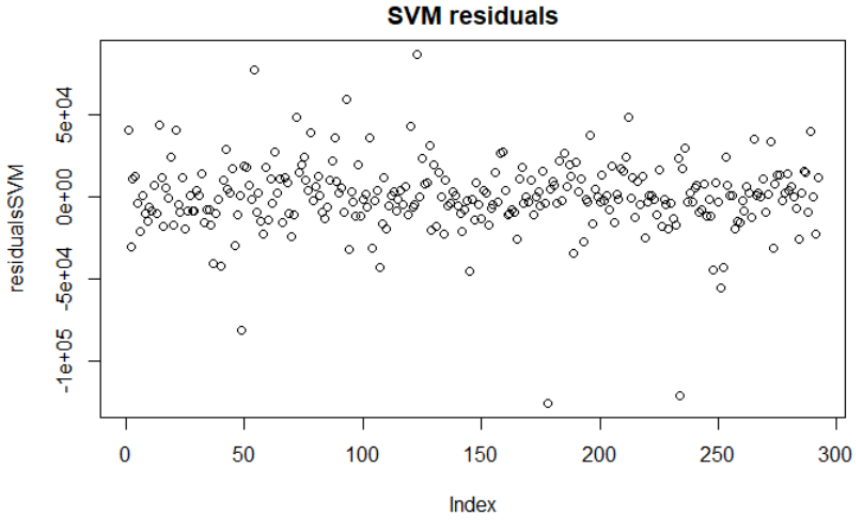
RMSE = 21458.6

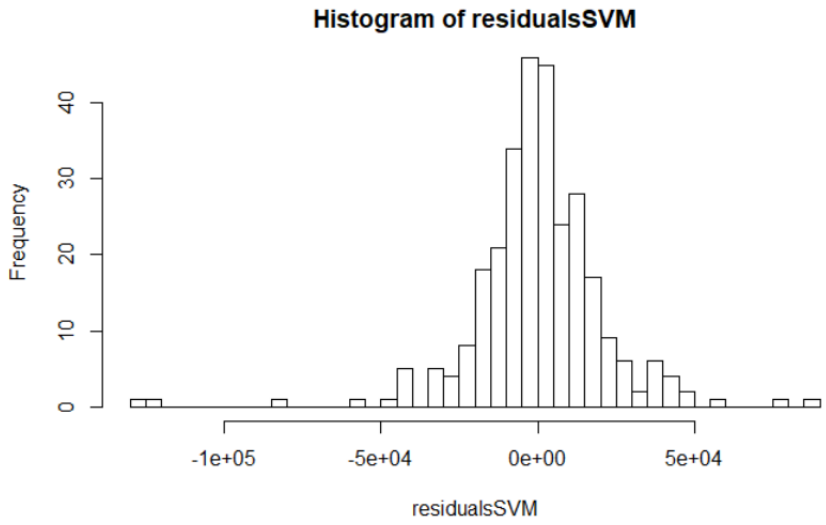


The linearity is excellent meaning that our model is quite accurate for predicting sold price.

```
library(modelr)
model <- svm(lnPrice ~ .,  data = train_Set, cost = 3)
predict <- predict(model, newdata = test_Set)
RMSE(exp(predict), exp(test_Set$lnPrice))
rmse(model, train_Set)
plot(exp(predict), exp(test_Set$lnPrice))
```

2.2.1 Analysis of the residuals





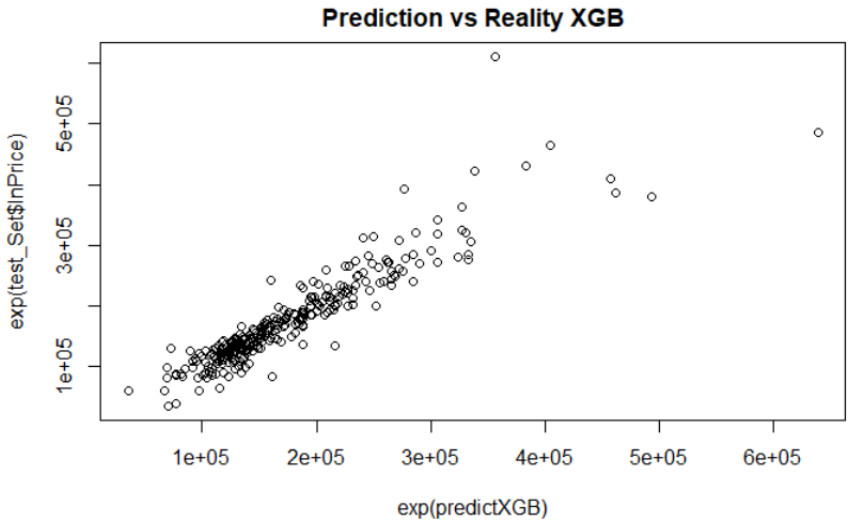
These graphs can teach us a lot, because even if the noise has a normal distribution and seems to be well distributed around 0 there exist a few outliers that are not well predicted by the model. We are not allowed to get rid of them we need more information about those 2 points.

But can we do better ? with an other algorithm for example

2.3 Extrem Gradient Boosting (XGB)

This is an other algorithm very popular for the recent past years in machine learning. In R it does accept only matrix input, so we need to convert our data frame into a matrix. library : xgboost function : xgboost()

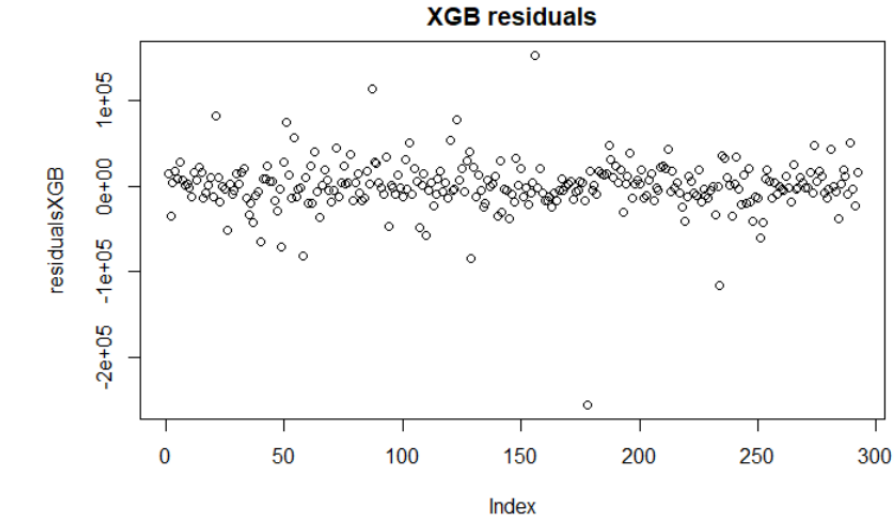
RMSE = 30198.76 Which is worst than SVM

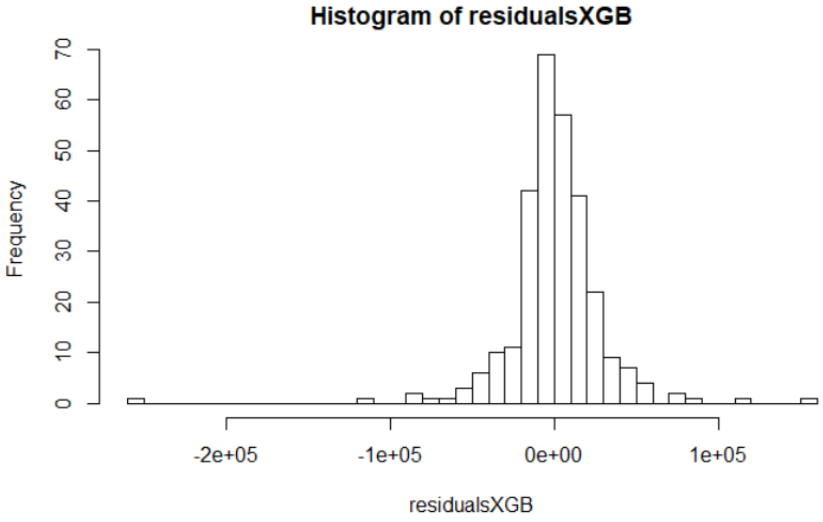



```
library(modelr)
model <- svm(lnPrice ~ .,  data = train_Set, cost = 3)
predict <- predict(model, newdata = test_Set)
RMSE(exp(predict), exp(test_Set$lnPrice))
rmse(model, train_Set)
plot(exp(predict), exp(test_Set$lnPrice))
```

2.3.1 Analysis of the residuals

We still have outliers that lower the performance of the algorithm unfortunately. But we really cannot get rid of them without a good explanation.

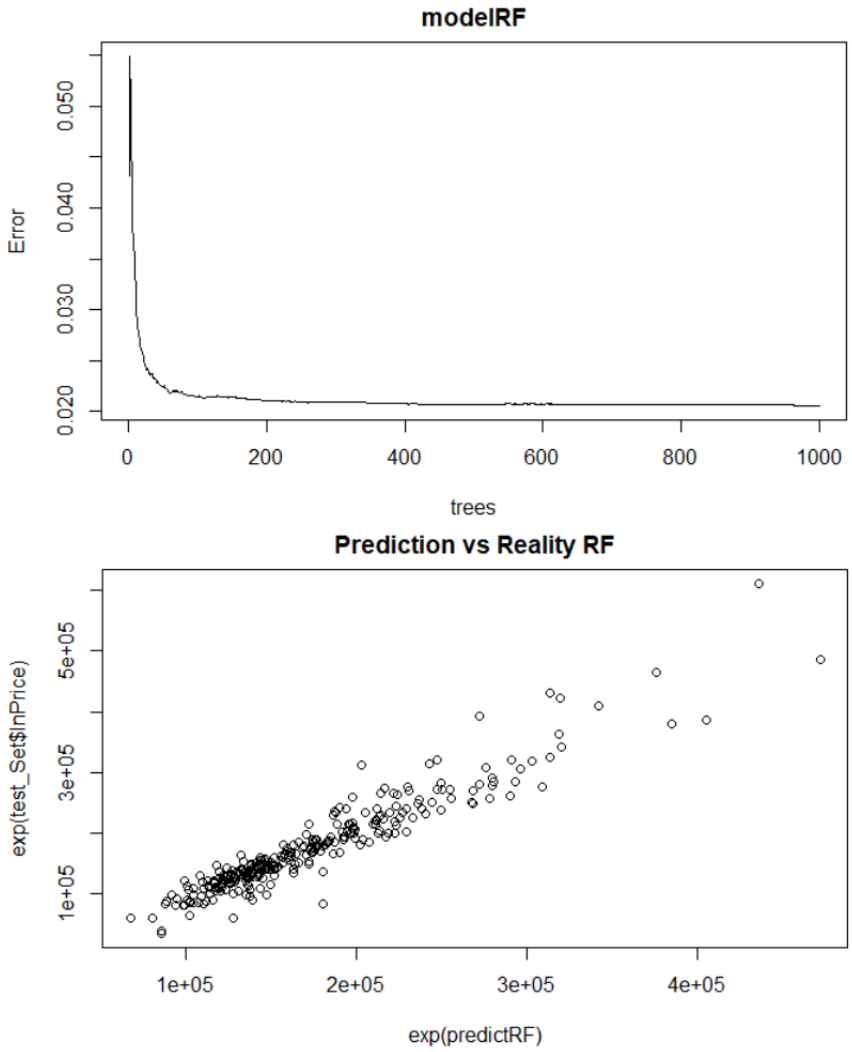




2.4 Random Forest

RMSE = 27188.43

```
library(randomForest)

modelRF <- randomForest(lnPrice ~ .,  data = train_Set, cost = 3, ntree = 500)
plot(modelRF)
predictRF <- predict(modelRF, newdata = test_Set)
RMSE(exp(predictRF), exp(test_Set$lnPrice))
rmse(modelRF, train_Set)
plot(exp(predictRF), exp(test_Set$lnPrice), main = 'Prediction vs Reality')
```

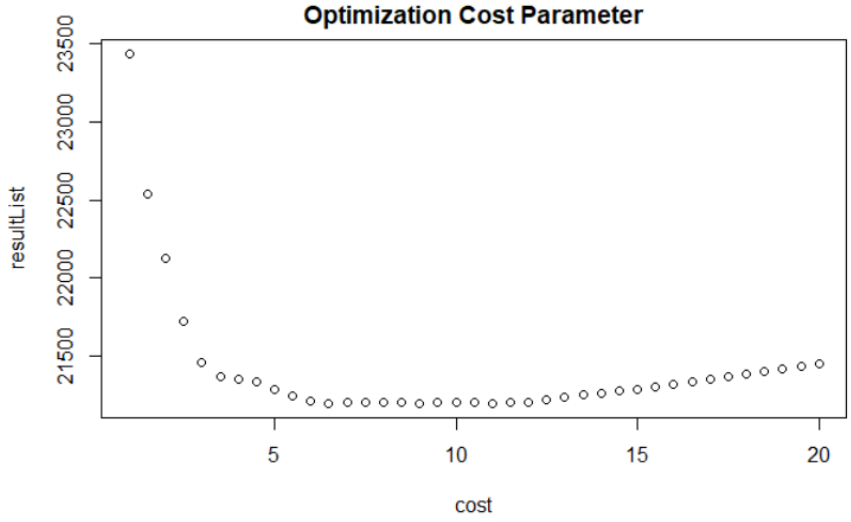


CONCLUSION : Between SVM, XGB and RF the best model is SVM so far, so now can you improve it ?

2.5 Improve SVM model

2.5.1 Cost parameter

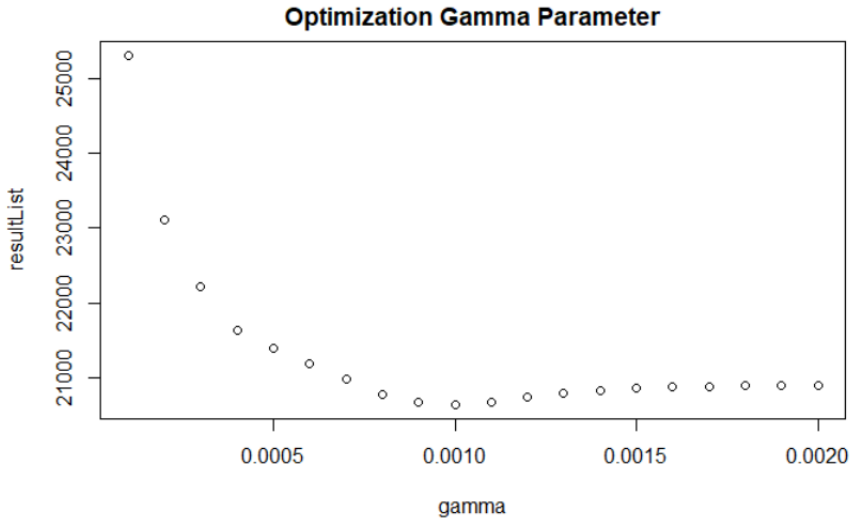
With a simple loop we look for the best "cost" parameter for the svm function. Result is 11




```
#OPTIMIZATION OF THE SVM
cost <- seq(1,20, by=0.5)
resultList <- c()
for (i in cost) {
  modelOpt <- svm(lnPrice ~ .,  data = train_Set, cost = i)
  predictOpt <- predict(modelOpt, newdata = test_Set)
  print(i)
  result <- RMSE(exp(predictOpt), exp(test_Set$lnPrice))
  print(result)
  resultList <- cbind(resultList, result)
}
plot(x = cost, y = resultList, main = 'Optimization Cost Parameter')
```

2.5.2 Gamma parameter

Now I tweak the gamma parameter: By default the value is 1/dimension here it is 0.012 We get 0.0010 as best parameter.




```
#OPTIMIZATION OF THE SVM
gamma <- seq(0.0001,0.002, by=0.0001)
resultList <- c()
for (i in gamma) {
  modelOpt <- svm(lnPrice ~ .,  data = train_Set, cost = 11, gamma = i)
  predictOpt <- predict(modelOpt, newdata = test_Set)
  print(i)
  result <- RMSE(exp(predictOpt), exp(test_Set$lnPrice))
  print(result)
  resultList <- cbind(resultList, result)
}
plot(x = gamma, y = resultList, main = 'Optimization Gamma Parameter')
```

2.6 FINAL MODEL

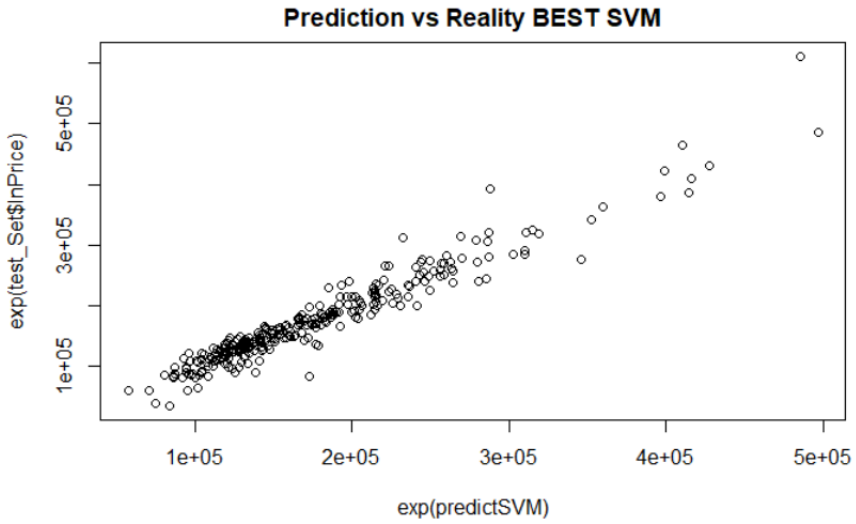
Finally I chose the SVM model with the following parameters:

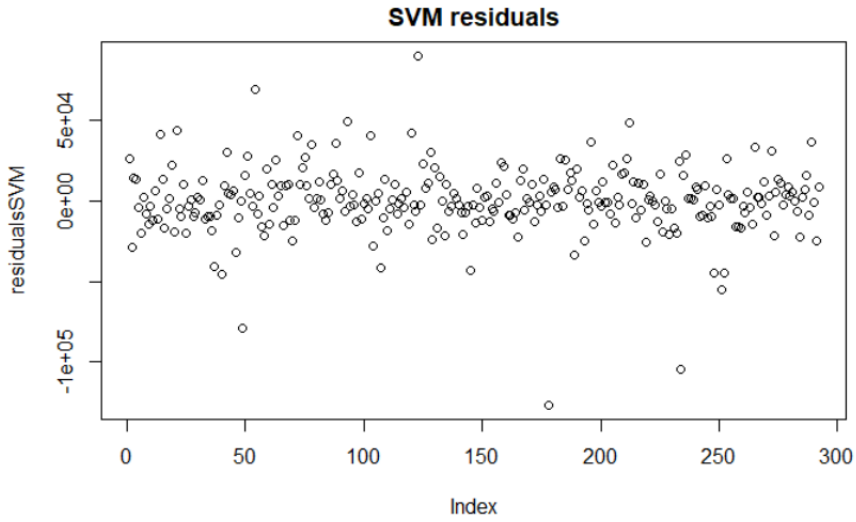
gamma = 0.0010

cost = 11

We run the model with our train set and test set:

We obtain a RMSE of 20644.95 which is better than the 21458 of the first model.





2.7 CONCLUSION EX 2

I obtained a solid model for predicting House Pricing using Support Vector Machine function of R.

The data were not very clean but with a few tricks I obtain a nice model.

3 SUMMARY OF MTGAUE

Neurons are connected with other neurons but in order to measure the neuronal activity we can measure the "spikes" with electrodes. the issue comes from the neurons that are functionally connected to other neurons and how to determine it. MTGAUE method is based on the Unitary Event procedure but go further than this.

The main issue is to determine the dependency/Independency that exists between neurons. When neurons have spikes close in time we can assume they are linked. What we try to determine here is whether this link is only a coincidence or not. We can model it as Bernoulli process. The notion of coincidence is defined by:

$$X = \sum_{i=1}^{n} \sum_{k=1}^{n} \mathbf{1}_{|k-i| \leq d} \mathbf{1}_{H_i^1=1} \mathbf{1}_{H_k^2=1}$$

or more generally:

$$X = \int_{W^2} \mathbf{1}_{|x-y| \leq \delta} N_1(dx) N_2(dy).$$

In order to define independence between neuron N1 and N2 and if N1/N2 are Poisson homogeneous processes, we can build a test based on theorem 1 and theorem 2 such that:

Theorem 1:

$$m_0 := E(X) = \lambda_1 \lambda_2 \left[2\delta T - \delta^2\right]$$

Theorem 2:

$$\hat{\lambda}_j := \frac{1}{MT} \sum_{m=1}^{M} N_j^{(m)}(W).$$

$H_0 : N1/N2 independent$
$H_1 : N1/N2 dependent$

False Discovery Rate (FDR) is performed under Benjamini-Hochberg process.

This new process MTGAUE has been validated experimentally.