

# Machine Learning Course Project

## Lecturer's Project

Christophe Bontemps & Patrick Jonsson - SIAP

26 November, 2021

## Contents

### The Project

This project is an exercise for learning how to implement the Machine Learning methods we have seen on the course on a realistic data set. We follow the classical workflow that we already have proposed in our previous documents.

Our goal is to fit a *Random Forest* model with a large amount of observations and several variables. This can be quite computationally costly if we use all the variables available in the data set. Using all the variables may all lead to very complex models prone to over-fitting. To reduce the number of variables and the complexity of the model, we use a *penalization method*, and fit a *LASSO* model. As this method requires the selection of an *hyperparameter* ( $\lambda$ ), we will use a grid search to select the value providing the lowest RMSE on Cross-Validated samples.




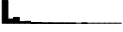
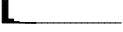

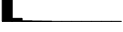


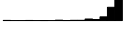
We will examine the results and select the variables to include in the *Random Forest* model that we will first estimate with a limited number of trees. Then, if we feel confident that this approach is generating promising results, we will increase the number of trees to 100.

We followed the general workflow provided in the template:

- Explore the data set
- Prepare the analysis using the right variable transformations
- Select the variables using LASSO
- Interpret the results and use them to reduce the complexity of the analysis
- Fit a Random Forest model
- Conclude the analysis

The data set can be used directly from the SIAP's server. If you need, you can also download it and use it locally.

```
# Reading DHS survey data from SIAP's website  
df <- read.csv("https://www.unsiap.or.jp/on_line/ML/M6-clean_data.csv")
```

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
price	649	0	163.7	421.8	0.0	105.0	10000.0	
latitude	18964	0	40.7	0.1	40.5	40.7	40.9	
longitude	14790	0	-74.0	0.0	-74.2	-74.0	-73.7	
accommodates	19	0	2.9	1.9	1.0	2.0	22.0	
bedrooms	12	0	1.2	0.8	0.0	1.0	21.0	
beds	17	0	1.5	1.1	0.0	1.0	16.0	
cleaning_fee	214	0	56.2	66.3	0.0	40.0	1200.0	
minimum_nights	110	0	7.9	21.6	1.0	3.0	1250.0	
availability_365	366	0	120.7	140.4	0.0	63.0	365.0	
review_scores_rating	62	0	94.2	8.0	20.0	96.0	100.0	

		N	%
host_is_superhost	False	39055	80.1
	True	9696	19.9
neighborhood	A-Zone	19604	40.2
	B-Zone	1149	2.4
	M-Zone	21778	44.7
	Q-Zone	5859	12.0
	S-Zone	361	0.7
property_type	Appartment	41746	85.6
	House	5702	11.7
	PrivateRoom	1303	2.7
cancellation_policy	flexible	15540	31.9
	moderate	11323	23.2
	strict	21888	44.9

## Data exploration

The data set is a record of renting prices for different types of housing options. It contains **48751** observations and **14** variables.

### Summary statistics

Since some variables are categorical, and other are numerical, we visualize their properties and statistical summaries separately. All variables are self-explanatory and their meaning can be understood quite easily.

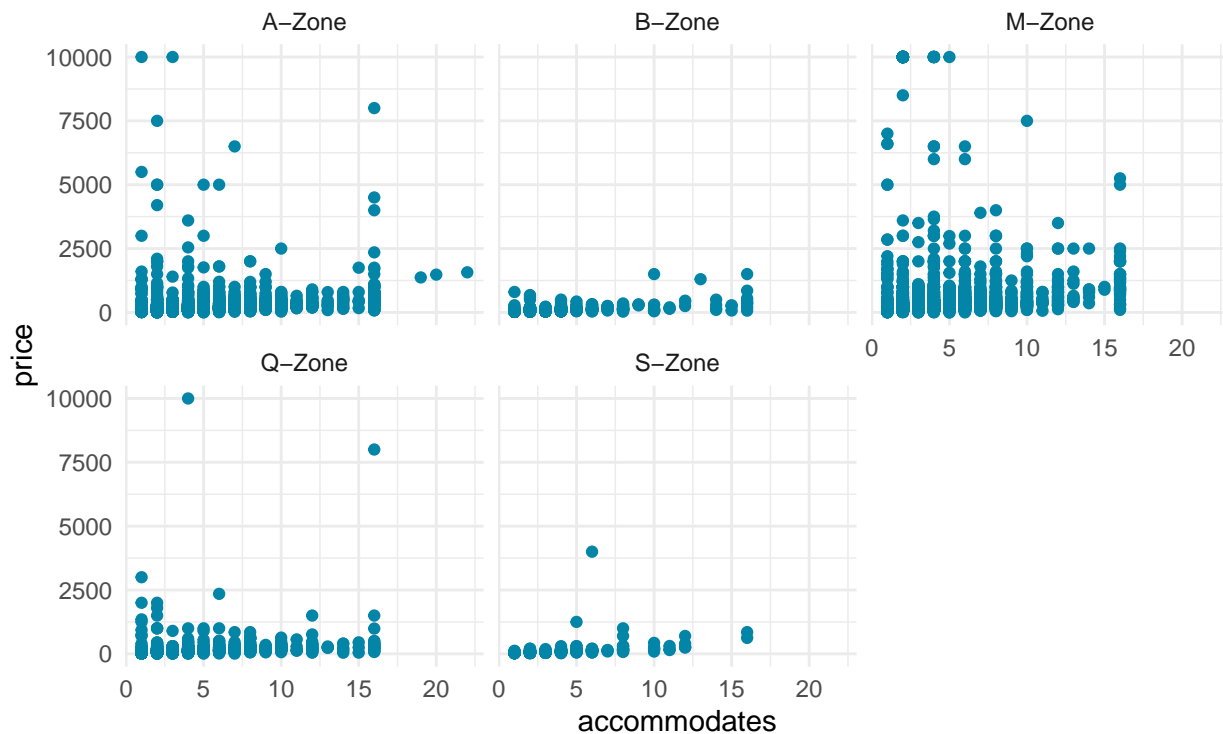
```
datasummary_skim(df, type = "numeric")
```

Numerical variables

```
datasummary_skim(df, type = "categorical" )
```

**Categorical variables** The dispersion of the price is quite skewed, with some very high prices. We may want to explore a bit and see whether the renting prices are related to some features of the good. We may produce several graphics but report only this one showing some outliers in the A, M and Q-Zones.

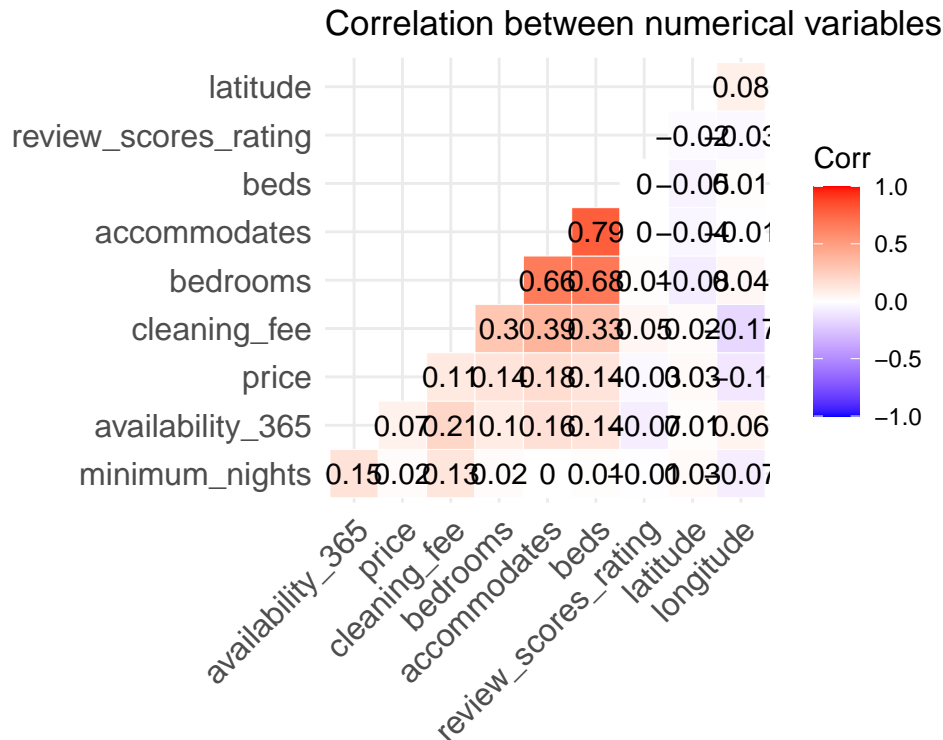
```
ggplot(df) +
  aes(x = accommodates, y = price) +
  geom_point(shape = "circle", size = 1.5, colour = SIAP.color) +
  theme_minimal() +
  facet_wrap(vars(neighborhood))
```



## Correlations

Another good practice is to see whether some variables are correlated.

```
numeric <- select_if(df, is.numeric)
# We compute the correlation matrix of the covariates
corr_coef <- cor(numeric, use = "p")
# And then plot it with nice options
ggcorrplot(corr_coef,
  type = "lower",           # lower triangle of the matrix only
  hc.order = TRUE,         # variable sorted from highest to lowest
  outline.col = "white",   # Color options
  lab = TRUE) + ggtitle("Correlation between numerical variables")
```



We see no strong correlations of any variable with the target variable *price*, based on the correlation plot. However, it seems like *cleaning fee*, *bedrooms*, and *accommodates* have the strongest correlation, but it is only in the range of 0.11-0.14. The variables *accommodates*, *beds*, and *bedrooms* are strongly correlated which is not too surprising.

A more complete exploration of the data requires some advanced graphics such as *Parallel Coordinate Plot* below where we can represent each observation (line) over 6 variables (vertical axes, standardized) and grouped by neighborhoods (colors).<sup>1</sup>

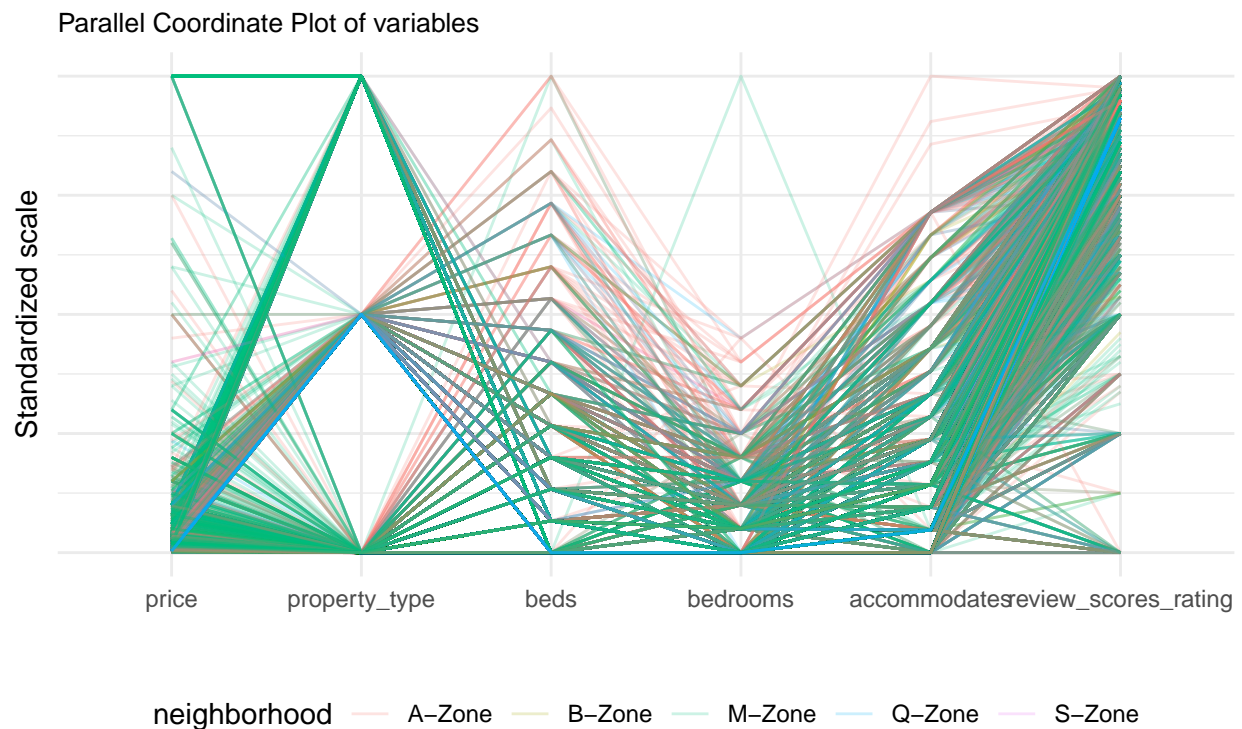
```
library(GGally)
library(dplyr)
library(viridis)
# Parallel Coordinates Plot
df %>%
  dplyr::select( price, property_type,
                 beds, bedrooms, accommodates, review_scores_rating,
                 neighborhood ) %>%
  mutate(neighborhood = as.factor(neighborhood)) %>%
  mutate(across(where(is.integer), as.numeric)) %>%
  filter(beds>0 & bedrooms >0 ) %>%
  ggparcoord(
    columns = 1:6 ,
    groupColumn = 7,
    scale = "uniminmax",
```

<sup>1</sup>This plot is used when one want to visualize data sets with a large number of variables. It uses transparent lines to highlight patterns shared by a particular group of observations. See also <https://www.data-to-viz.com/graph/parallel.html>

```

showPoints = FALSE,
title = "Parallel Coordinate Plot of variables",
alphaLines = 0.2
) +
theme_minimal()+
theme(
  legend.position= "bottom",
  plot.title = element_text(size=10),
  axis.text.y=element_blank(),
  axis.ticks.y=element_blank() ) +
xlab("")+
ylab("Standardized scale")

```



It seems that, overall, the relations between variables makes some sense as high prices correspond generally to more *beds* and *bedrooms*. They seem to be quite ordered by *property type*. We do not see obvious patterns with the neighborhood (colors) as two zones (A and M-Zones) represent 85% of the observations.

## Preparing the analysis

With **14** variables and considering the correlation between the numerical variables, it may be interesting to try to reduce the number of regressors using a **penalization method** such as the LASSO.

Since the data set is quite large we only use half the observations ( $p = 0.5$  in *createData-*

`Partition()` controls this) for *training* the models, this will reduce the time you need to train. This can of course be increased or lowered if you want to experiment on the effect this has on your models.

```
# Splits data into training and testing sets
set.seed(777)
trainIndex <- createDataPartition(df$price, p = 0.5, list = FALSE, times = 1)
```

One must also prepare our *Machine Learning* framework and prepare our *train* data set separately from the one used for *validation*

```
train_data <- df[trainIndex,]
validation_data <- df[-trainIndex,]
```

As we have seen in the course, the *LASSO* method will be affected by the scale of the variables, so we need to **scale the variables** beforehand. Of course this should apply to both our *train* and *validation* data sets.

```
# Scale the training and test data based on the training data mean and variance.
ScalingValues <- preProcess(train_data, method = c("center", "scale"))
train_data <- predict(ScalingValues, train_data)
validation_data <- predict(ScalingValues, validation_data)
```

## Variable selection

Now we can fit the a regression model with *LASSO* regularization and interpret the results, this may give us some information about which variables might be useful in the Random Forest model later. We fit the regression model using a using cross-validation to ensure it is giving robust results.

### Selecting the right value of *lambda* for the LASSO

As in many Machine Learning exercise, one must select hyperparameters to adjust the procedure and to ensure the best accuracy and performance. Here we want to select the regression model with a good balance of accuracy and a minimal number of regressors to avoid collinearity and over-fitting.

In order to select precisely the penalization parameter *lambda* in the *LASSO*, the grid search for has been defined between 0 and 0.003, in **50** regular increments.

```
# Control variables
numbers <- 5
repeats <- 20
rcvTunes <- 10 # tune number of models
seed <- 123
# repeated cross validation
rcvSeeds <- setSeeds(method = "repeatedcv",
                     numbers = numbers, repeats = repeats,
                     tunes = rcvTunes, seed = seed)
```

```

# Controls for the CV
rcvControl <- trainControl(method = "repeatedcv",
                           number = numbers, repeats = repeats,
                           seeds = rcvSeeds)

set.seed(123)
lasso_fit <- train(price ~ .,
                  data = train_data,
                  method = "glmnet",
                  tuneGrid = expand.grid(alpha = 1,
                                         # grid to search
                                         lambda = seq(from = 0,
                                                       to = 0.003,
                                                       length = 50)),
                  trControl = rcvControl)

```

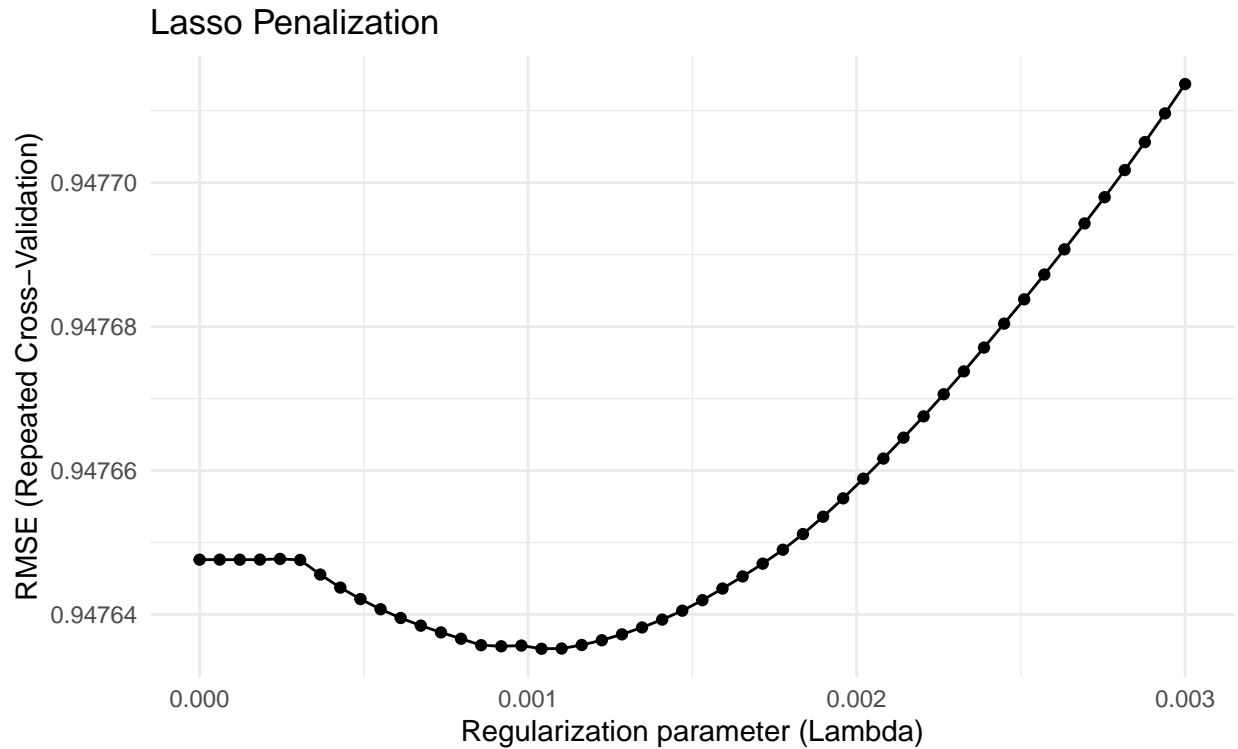
It is always a good practice to visualize the outcome of the grid search to check whether the value of the hyperparameter (*lambda* here) affects the outcome and if there exists a value that is optimal.

```

ggplot(lasso_fit) +
  ggtitle("Lasso Penalization") +
  labs(x = "Regularization parameter (Lambda)") +
  theme_minimal()

```

$\lambda$ (exp)
0.001



```
cbind(lasso_fit$bestTune$lambda) %>%
  kable(digits=3, col.names = c("lambda (exp)")) %>%
  kable_styling()
```

We have a nice U-shape curve.

It appears that the best value for  $\lambda$  is here **0.00104**. (This result is automatically updated when you *knit* your notebook.)

Since our goal is to feed a *Random Forest* model, we can use the previous results and identify the variables and features selected by the LASSO procedure (when used with the right  $\lambda$ ). This can also be done by visualizing the feature importance of the *LASSO* regression:

We first focused on the most important features with VIF being greater than 3 and refine the analysis later.

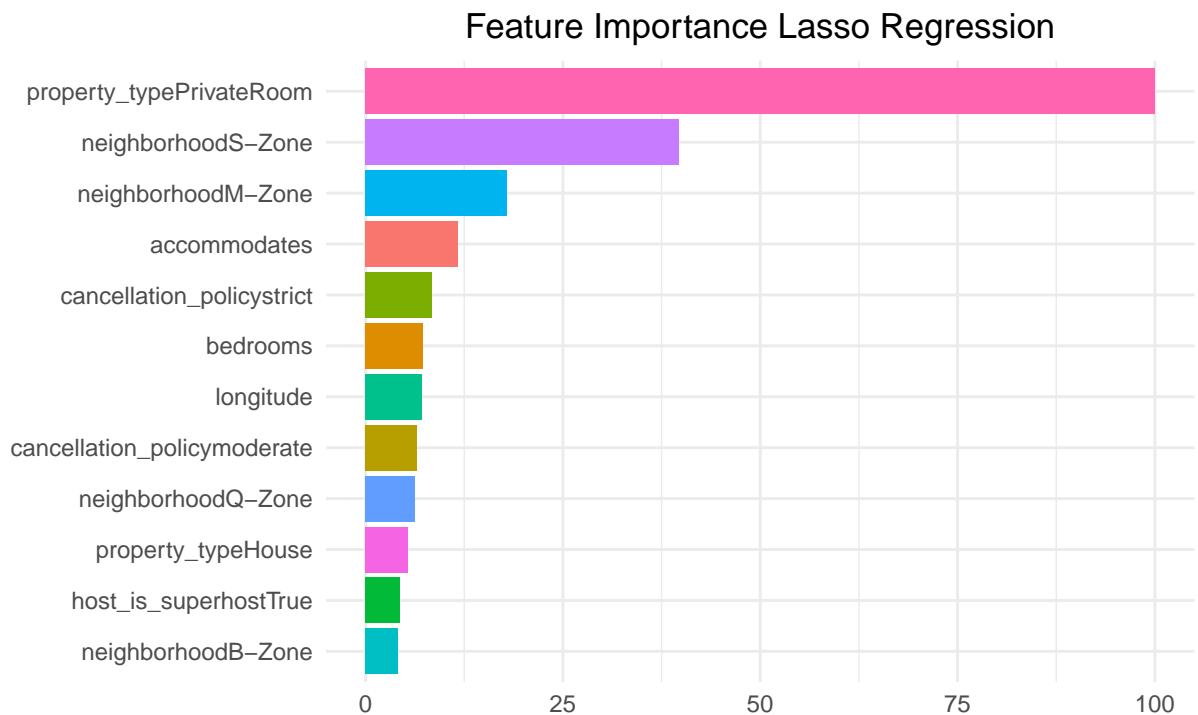
```
theme_models <- theme_minimal()+ theme(plot.title = element_text(hjust = 0.5),
  legend.position = "none")
```

```
lasso_varImp <- data.frame(variables = row.names(varImp(lasso_fit)$importance), varImp(1
```

```
# Below we set that we only show feature importance with a value larger than 3
# You can lower this if you want to see more variables, or increase it if you want to
```



```
threshold = 3
lasso_varImp <- lasso_varImp[lasso_varImp$Overall >= threshold,]
ggplot(data = lasso_varImp, mapping = aes(x=reorder(variables, Overall),
                                             y=Overall,
                                             fill=variables)) +
  coord_flip() + geom_bar(stat = "identity", position = "dodge") +
  theme_models +
  labs(x = "", y = "") +
  ggtitle("Feature Importance Lasso Regression")
```



We find that some of the categorical variables appear to be informative, mainly *property type* and *neighborhood*, it also appears like the variables with the strongest correlations with our target variable still are somewhat useful in the model. We can also look at the model out-of-sample *Root Mean Squared Error* (RMSE), and compare this later to the results of the *Random Forest*:

```
lasso_preds <- predict(lasso_fit, validation_data)
Lasso.perf <- rmse(actual = validation_data$price, predicted = lasso_preds)
```

## Fitting a *Random Forest* model

We are now ready to give it a try and fit the **Random Forest**, but for this we need to specify some variables. Based on the variable importance in the *LASSO* regression and which

variables had the highest correlations with **price**, a good start is to use the model:

```
price = property_type+neighborhood+accommodates+bedrooms+cancellation_policy+cleaning_fee
```

as this contains what seems to be the most important variables, and it will not be too complex to fit.

```
# Modify this formula (keep the syntax!) to choose your regressors
```

```
MyRegressors <- "property_type + neighborhood + accommodates +  
bedrooms +cancellation_policy + cleaning_fee"
```

After a first experiment with a restricted number of trees (set to 10)we now run the model with **100** trees as we have seen that this may be crucial in the performances of a *Random Forest* model.

Estimating that model with 100 tree takes time!

```
# Training this model may take some time!!!
```

```
Ntree <-100 # The number of trees affect greatly the duration of the process
```

```
#Formula using the regressors' list defined above
```

```
MyFormula <- as.formula(paste('price ~', MyRegressors))
```

```
# Change the formula below:
```

```
rf_fit <- train(MyFormula,  
               data = train_data,  
               method = "rf",  
               ntree = Ntree)
```

```
rf_fit
```

```
## Random Forest
```

```
##
```

```
## 24376 samples
```

```
##      6 predictor
```

```
##
```

```
## No pre-processing
```

```
## Resampling: Bootstrapped (25 reps)
```

```
## Summary of sample sizes: 24376, 24376, 24376, 24376, 24376, 24376, ...
```

```
## Resampling results across tuning parameters:
```

```
##
```

##	mtry	RMSE	Rsquared	MAE
##	2	0.9324490	0.1501057	0.2064559
##	6	0.9251543	0.1561805	0.2083030
##	11	0.9371773	0.1451030	0.2150847

```
##
```

```
## RMSE was used to select the optimal model using the smallest value.
```

```
## The final value used for the model was mtry = 6.
```

```
rf_preds <- predict(rf_fit, validation_data)
rmse(actual = validation_data$price, predicted = rf_preds)
```

```
## [1] 0.9826546
```

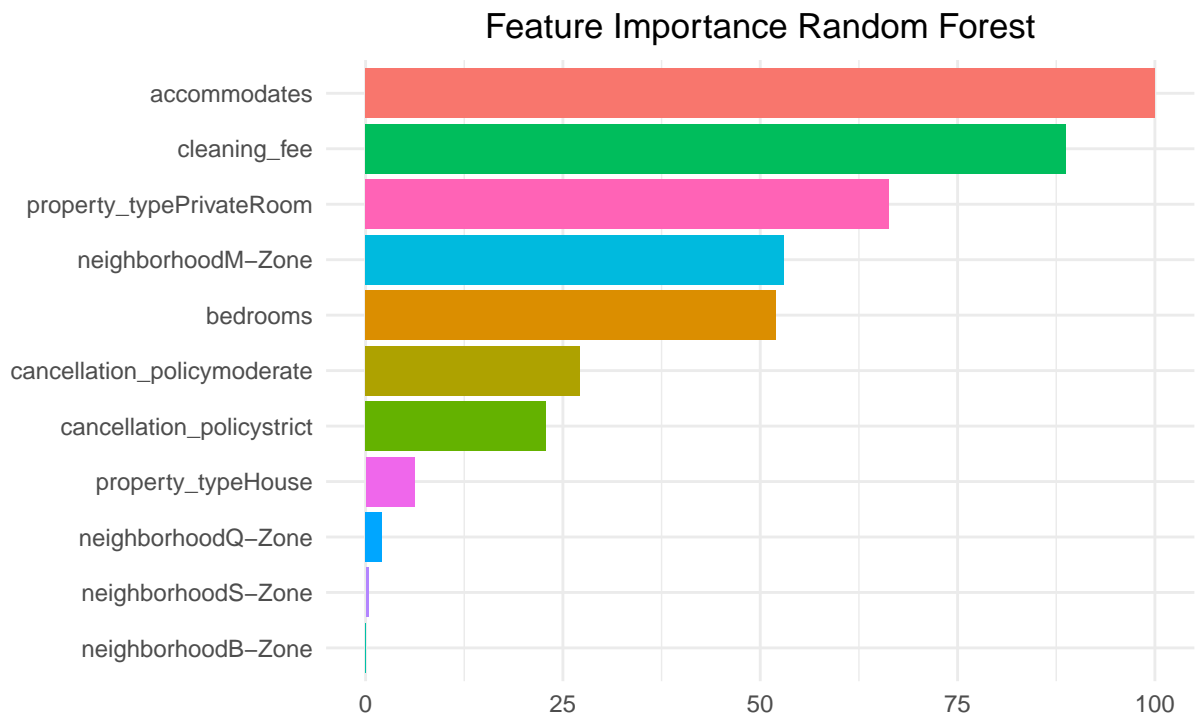
With this model we find some slight improvement over the LASSO in terms of the RMSE.

It is now interesting to see if the feature importance of the two different approaches are similar or not:

```
theme_models <- theme_minimal()+ theme(plot.title = element_text(hjust = 0.5),
                                       legend.position = "none")

rf_varImp <- data.frame(variables = row.names(varImp(rf_fit)$importance), varImp(rf_fit))

ggplot(data = rf_varImp, mapping = aes(x=reorder(variables, Overall),
                                       y=Overall,
                                       fill=variables)) +
  coord_flip() + geom_bar(stat = "identity", position = "dodge") +
  theme_models +
  labs(x = "", y = "") +
  ggtitle("Feature Importance Random Forest")
```



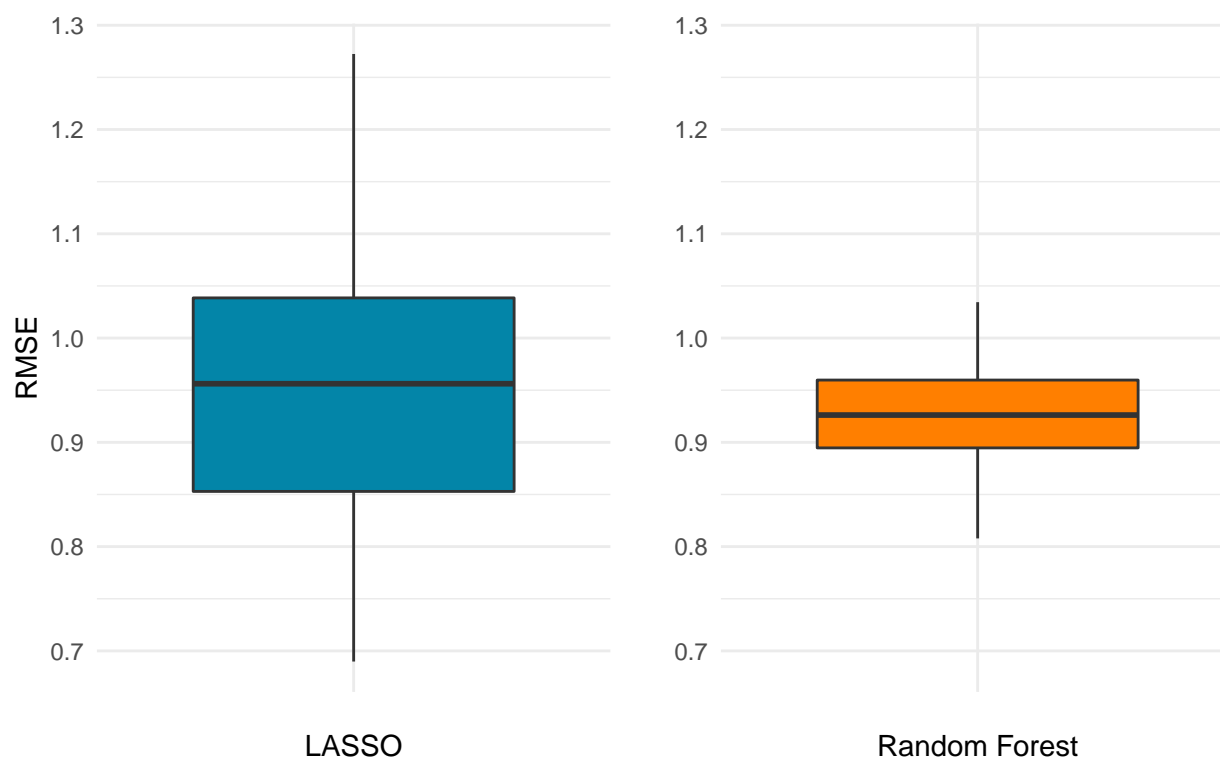
Comparing the result of the variable importance plot from the *LASSO* with the one from *Random Forest* we see some differences. While *property type* and *neighborhood* is still important, it is now less important than how many the unit accommodates, as well as the cleaning fee. We can also see that it is mainly just the impact of the *M-Zone* which is the

important feature in the *neighborhood variable*, where previously it was *S-Zone*.

We can now also visualize the overall performances of the two models by comparing the distribution of the RMSE over the Cross-Validated samples.

```
# This graphics allows the comparison of results (on the SAME scale)  
# even if number of CV samples are different
```

```
lasso <- as.data.frame(lasso_fit$resample$RMSE)  
rf <- as.data.frame(rf_fit$resample$RMSE)  
names(lasso) <- "RMSE"  
names(rf) <- "RMSE"  
  
y.min <- min(lasso, rf)  
y.max <- max(lasso, rf)  
  
p.lasso <- ggplot(lasso) +  
  aes(x = "", y = RMSE) +  
  geom_boxplot(fill = SIAP.color) +  
  labs(x = "LASSO", y = "RMSE") +  
  coord_cartesian(ylim = c(y.min, y.max)) +  
  theme_minimal()  
  
p.rf <- ggplot(rf) +  
  aes(x = "", y = RMSE) +  
  geom_boxplot(fill = orange.color) +  
  coord_cartesian(ylim = c(y.min, y.max)) +  
  labs(x = "Random Forest", y = "") +  
  theme_minimal()  
  
grid.newpage()  
grid.draw(cbind(ggplotGrob(p.lasso), ggplotGrob(p.rf), size = "last"))
```



In terms of performance, the median value for the RMSE using the *Random Forest* is **0.9262** which is slightly lower than **0.9562** for the regression model using *LASSO* despite the fact that the *Random Forest* model only using a small subset of the variables available. We also observe that the *Random Forest* model has a much lower variance since the dispersion of the errors - measured by the values of the RMSEs on the validation sets - is lower magnitude compared to the ones for the *LASSO*.

We are confident that the **Random Forest** model would be the best suited for predicting a price on a new data set!

## We went a bit further

We let the algorithm sort out the variables selection itself

A more common approach in machine learning where the algorithm has a natural way of selecting variables is to let the algorithm sort out the variable selection itself. In large data sets this can be very difficult to train as the computational complexity grows with your data and choice of hyperparameters.

We fixed here the `.mtry` hyperparameter, limiting the maximum number of variables considered at each node to **6**. Not fixing this parameter, and then letting the algorithm examine all variables all the time, could lead to a very high computation time and this would probably not be efficient.

```
# Training this model may take some time.
```

```
# Change the formula below:
```

```
rf_fit_opt <- train(price ~ .,
  data = train_data,
  method = "rf",
  ntree = 100,
  tuneGrid = expand.grid(.mtry = 6))
```

```
rf_fit_opt
```

```
## Random Forest
##
## 24376 samples
## 13 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 24376, 24376, 24376, 24376, 24376, 24376, ...
## Resampling results:
##
## RMSE          Rsquared    MAE
## 0.8027999 0.3386982 0.1807132
##
## Tuning parameter 'mtry' was held constant at a value of 6
```

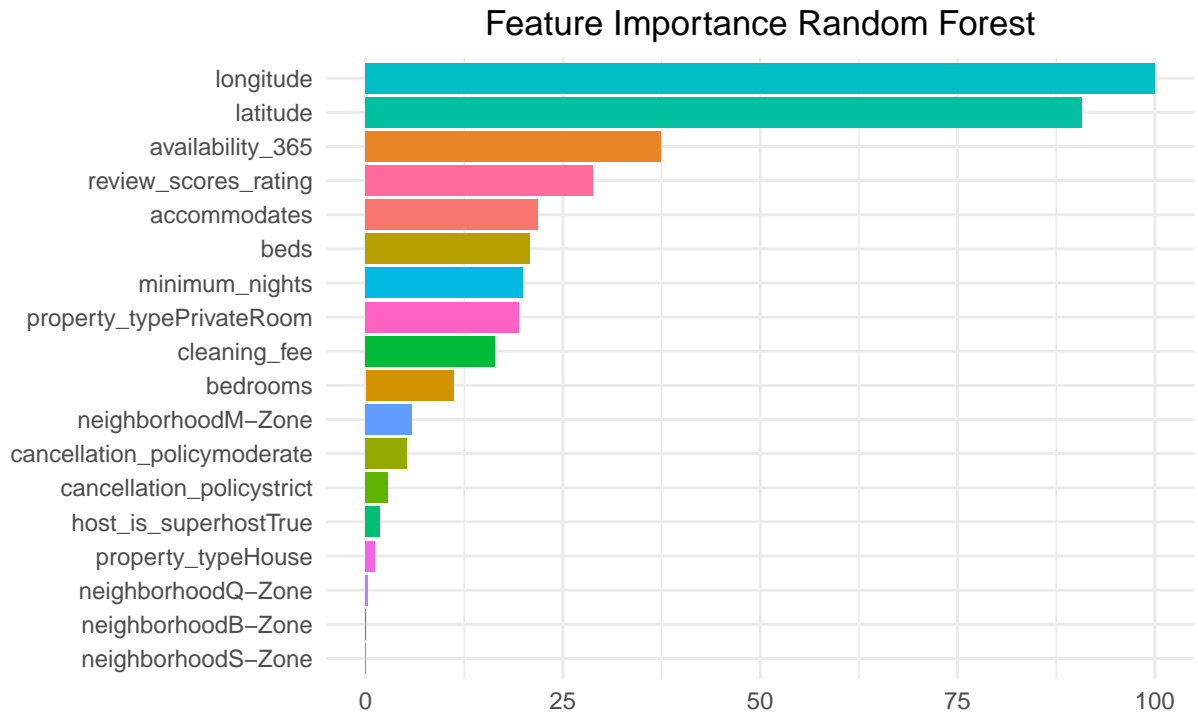
```
rf_opt_preds <- predict(rf_fit_opt, validation_data)
rmse(actual = validation_data$price, predicted = rf_opt_preds)
```

```
## [1] 0.8638984
```

```
theme_models <- theme_minimal()+ theme(plot.title = element_text(hjust = 0.5),
  legend.position = "none")
```

```
rf_opt_varImp <- data.frame(variables = row.names(varImp(rf_fit_opt)$importance), varImp
```

```
ggplot(data = rf_opt_varImp, mapping = aes(x=reorder(variables, Overall),
  y=Overall,
  fill=variables)) +
  coord_flip() + geom_bar(stat = "identity", position = "dodge") +
  theme_models +
  labs(x = "", y = "") +
  ggtitle("Feature Importance Random Forest")
```



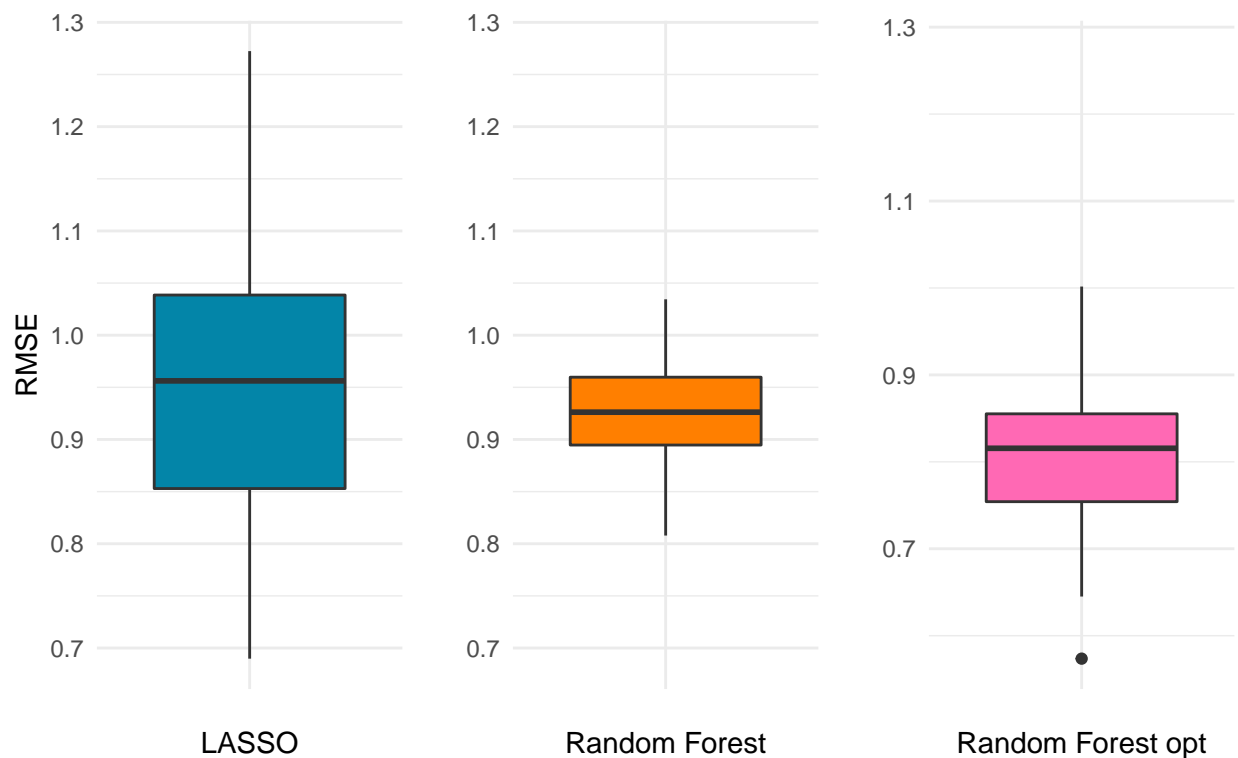
The feature importance graphic come here as **a surprise** since the variables that seem to have the greatest importance are new and were not previously selected. Both *latitude* and *longitude*, but also *availability365*, are now very important to predict prices, when we let the model choose the variables (up to 6) at each node. This actually makes sense since the location of any renting place is certainly an important component of the renting price. The *availability365* may be a proxy to determine long-term (and thus cheaper) prices. The reviews from previous customers seem to have a great effect as well as they probably reflect the prices quite efficiently.

```
rf.opt <- as.data.frame(rf_fit_opt$resample$RMSE)
names(rf.opt) <- "RMSE"
y.min <- min(lasso, rf, rf.opt)
y.max <- max(lasso, rf, rf.opt)

p.rf.opt <- ggplot(rf.opt) +
  aes(x = "", y = RMSE) +
  geom_boxplot(fill = pink.color) +
  coord_cartesian(ylim = c(y.min, y.max))+
  labs( x = "Random Forest opt", y = "")+
  theme_minimal()

grid.newpage()
grid.draw(cbind(ggplotGrob(p.lasso),
```

```
ggplotGrob(p.rf) ,
ggplotGrob(p.rf.opt) ,
size = "last"))
```



## Conclusion

In this exercise, it seems that the *Random Forest* model is performing much better when all parameters are optimized and when the model has enough trees to learn and “*bagg*”. Increasing the number of trees to 100 lead to a great increase in performance but the greatest improvement came from letting the algorithm choose (within a limit of 6) which variables to use at each decision node. This automatic procedure is here very efficient since this model is better than the ones using our pre-selected set of variables.

Letting the model choose the variables used to split in each tree, even if we limited the maximum number, led also to a better understanding of which variables we should consider when trying to predict a price. This came here as a surprise since the variables found to be the most important were not the ones selected by the *LASSO* procedure.

This confirms that, in Machine Learning, one should always try different methods and compare their performances. This provides also a multifaceted view of the same problem using different methods, different representations and different outcomes.



## Corresponding functions if you use Python for this project

- *pandas* and *numpy* offer great functions for handling your data.
- *Sklearns* ensemble library has the function *ensemble.RandomForestClassifier()*, which can be used to train the Random Forest model. It also has gradient boosting functions like *ensemble.GradientBoostingClassifier()* and *ensemble.GradientBoostingRegressor()*.
- the library *imbalanced-learn* has tools for oversampling, such as the SMOTE algorithm.
- *matplotlib* offers good visualizations for your results, including feature importance plots.