# Compulsory Exercise 2: Predicting AirBnB-prices in Rome

Helle Villmones Haug  Hjalmar Jacob Vinje  Sanna Baug Warholm

20 April, 2023

**Abstract**

The purpose of the project is to use machine learning to predict the price of Airbnb nights in Rome. The data set used is the rome_weekends.csv file from Kaggle (XX source). It was analyzed by a Neutral network-model and a Random Forest-model. Both models predicted the AirBnB-prices with high accuracy. Why are findings important/novel?

## Introduction: Scope and purpose of your project

In this project, we aim to predict the total rental price of Airbnb listings in Rome during weekends, using a dataset called 'rome_weekends.csv' from Kaggle (https://www.kaggle.com/datasets/thedevastator/airbnb-prices-in-european-cities?select=rome_weekends.csv). This is a regression task, as we are predicting a continuous variable (total rental price) based on various features of the listings.

The dataset is obtained from a public platform, Kaggle, where users can upload, share, and collaborate on data science projects. The data contains information on Airbnb listings in Rome, such as room type, whether the room is shared or private, and if the host is a superhost. Our goal is to utilize this information to predict the total rental price of a given listing and evaluate the performance of two machine learning models.

The purpose of this project is two-fold:

1. To compare the performance of two different machine learning techniques: a deep learning model using the Keras library and a Random Forest model. We want to determine which model yields better prediction accuracy and whether there is a trade-off between performance and model complexity.
2. To uncover relationships between different variables and identify important predictors for the total rental price.

Our audience consists of data scientists, machine learning enthusiasts, and Airbnb stakeholders who are interested in gaining a deeper understanding of the factors affecting rental prices and improving price prediction models. The main purpose of this project is to predict the total rental price of Airbnb listings with high accuracy, while also uncovering relationships between variables that can contribute to a better understanding of the underlying factors that influence rental prices. In essence, our project focuses on both prediction and inference, with the goal of achieving high prediction accuracy and gaining insights into the determinants of rental prices.

We chose to predict AirBnB prices, but some other interesting things you could analyze with this dataset using machine learning techniques could be:

1. Recommending similar listings: Using a technique such as content-based filtering, you can recommend similar listings to users based on the features of listings they have previously booked or shown interest in. This can help users discover new listings that match their preferences..
2. Identifying superhosts: You can use the features in the dataset, such as $'host\_is\_superhost$ and $guest\_satisfaction\_overall$, to build a machine learning model that can predict whether a host is likely to be a superhost. This can help AirBnB identify hosts who are providing exceptional service and reward them accordingly.

## Descriptive data analysis/statistics

This section we describe the data, including the columns and the target. We also perform some data analysis, to get a better overview of the dataset and its feature. Lastly, we perform data pre-prosessing, and split the dataset into training- and testsets.

### The dataset: AirBnB prices on weekends in Rome

This is an overview of the columns in the dataset and their description, found on the Kaggle website.

| Column name | Description | Data type |
|---|---|---|
| realSum | The total price of the Airbnb listing | Numeric |
| room_type | The type of room being offered (e.g. private, shared, etc.) | Categorical |
| room_shared | Whether the room is shared or not | Boolean |
| room_private | Whether the room is private or not | Boolean |
| person_capacity | The maximum number of people that can stay in the room | Numeric |
| host_is_superhost | Whether the host is a superhost or not | Boolean |
| multi | Whether the listing is for multiple rooms or not | Boolean |
| biz | Whether the listing is for business purposes or not | Boolean |
| cleanliness_rating | The cleanliness rating of the listing | Numeric |
| guest_satisfaction_overall | The overall guest satisfaction rating of the listing | Numeric |
| bedrooms | The number of bedrooms in the listing | Numeric |
| dist | The distance from the city centre | Numeric |
| metro_dist | The distance from the nearest metro station | Numeric |
| lng | The longitude of the listing | Numeric |
| lat | The latitude of the listing | Numeric |

In our context, the column *realSum*, the total price of the AirBnB listing, is the target, in other words, waht we want to predict. The other columns will be used to predict the *realSum* for new listing.

### Loading the data

We load the data by using the function $read_csv$-function. The data set was downloaded locally and placed in a folder in the repository, and not dowNloaded directly from the internet, therefore the 'data/rome_weekends.csv'-souce

```
# Read the data
data <- read_csv("data/rome_weekends.csv", show_col_types = FALSE)
colnames(data)
```

```
##  [1] "...1"                      "realSum"
##  [3] "room_type"                 "room_shared"
##  [5] "room_private"              "person_capacity"
##  [7] "host_is_superhost"         "multi"
##  [9] "biz"                       "cleanliness_rating"
## [11] "guest_satisfaction_overall" "bedrooms"
## [13] "dist"                      "metro_dist"
## [15] "attr_index"                "attr_index_norm"
## [17] "rest_index"                "rest_index_norm"
## [19] "lng"                       "lat"
```

Note that 'attr_index' and 'rest_index' (and also 'attr_index_norm and 'rest_index_norm', the normalized versions of the columns) is not described in the table above. This is because they were not mentioned nor described on Kaggle, and therefore we don't know what they represent.

### Descriptive data analysis

Next, we conduct some analysis to get an overview over the data. We look at measures such as mean, median, range, standard deviation, and variance to describe the central tendency, variability, and distribution of a data set.

**Summary statistics** It is useful to get a quick overview of the central tendency, variability, and distribution of a data set. For example, if the prices where close to normally distributed, beside from some very high or very low prices, it could be useful to perfrom a log()-operation on the target . In addition, the summary() function can be used to check the data for any unexpected values, such as negative values for variables that should be positive or large values for variables that should be small.

```
# Compute summary statistics for the realSum column
summary(data$realSum)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   46.06  138.64  184.46  209.13  242.68 2311.74
```

```
range(data$realSum)
```

```
## [1]   46.05709 2311.73871
```

**Nan-values** It is important to remove potential NaN-values, because they can cause biased or inaccurate results in data analysis, and also affect the accuracy and reliability of statistical estimates. When missing values are present, statistical estimates such as the mean, standard deviation, or correlation coefficient may be biased or less reliable.

```
sum(is.na(data))
```

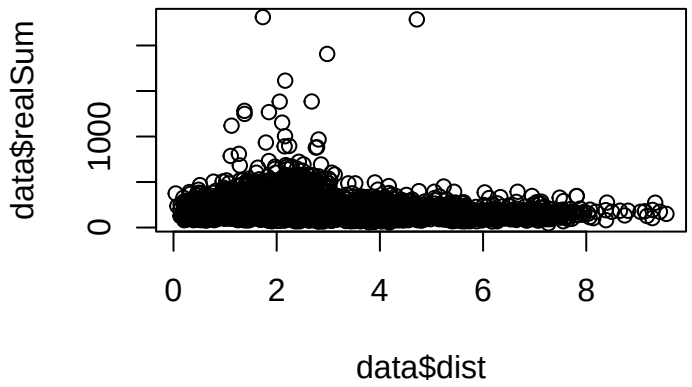
```
## [1] 0
```

As we can see, there is zero NaN-values in the data set, which means we don't need to clean the data for NaN-values. This means that the dataset we chose was of high quality. This was expected, since the dataset had a great number of upvotes on Kaggle.

**Other descriptive plots** Here we show a scatter plot, a box plot and a histogram, to show other potensial patterns in the data.

**Scatter plot** Scatter plots are used to display the relationship between two continuous variables.

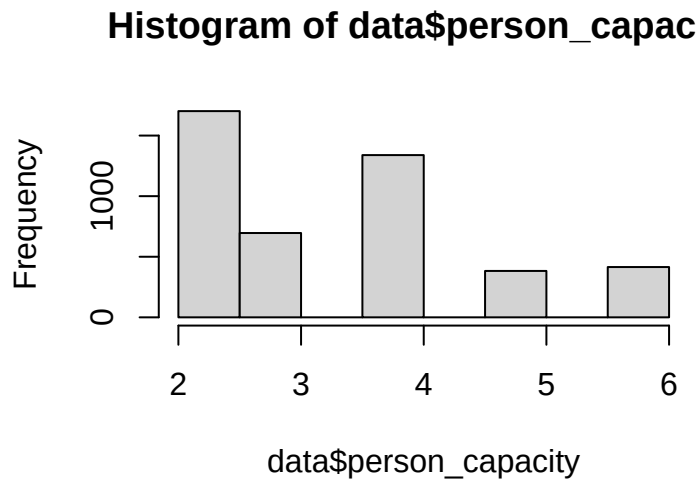
```
# Create a scatter plot of realSum vs. dist
plot(data$dist, data$realSum)
```



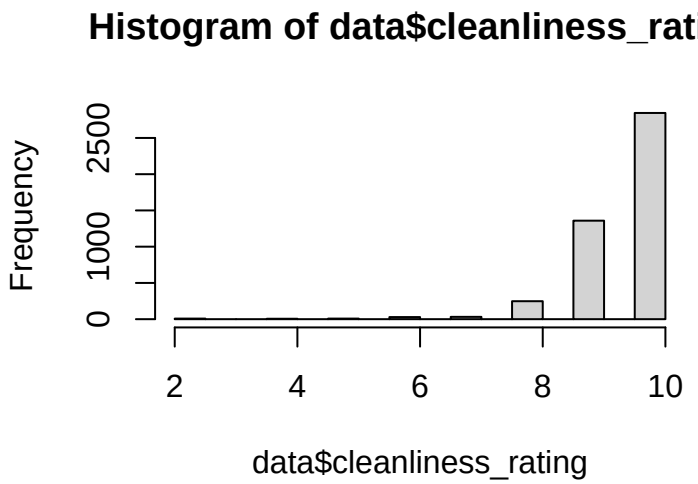

This shows a small trend where AirBnBs with higher prices tends to be located closer to the city center.

**Histogram** Histograms provide a visual representation of the distribution of a variable.

```
# Create histograms of the person_capacity and cleanliness_rating columns
hist(data$person_capacity)
```



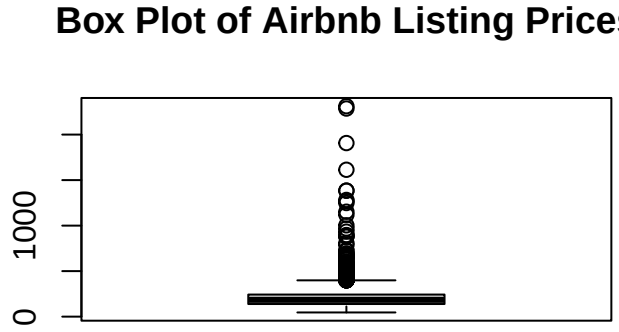

```
hist(data$cleanliness_rating)
```




The first plot shows that the most common person capacity for a AirBnB is 2, followed by 4. The second plot shows that the majority of the AirBnBs are rated as very clean.

**Box plot** Box plots are used to display the distribution of a continuous variable.

```
# Create a box plot of the realSum column
boxplot(data$realSum, main = "Box Plot of Airbnb Listing Prices")
```



This plot shows that most of the prices are in the same price range (visualized by the black straight line). Then, there are fewer and fewer AirBnBs with higher and higher prices. We don't see the same trend with lower prices.

### Data pre-processing

Data pre-processing is important in machine learning because it helps to improve the quality of the data and prepare it for modeling. It involves identifying and addressing issues such as missing values, outliers, and errors, selecting relevant features, scaling or normalizing the data, and handling categorical data.

The methods we are using require numerical variables, therefore we convert the categorical- and numeric values we will use.

```
# Convert non-numeric variables into numeric variables
data$room_type <- as.numeric(factor(data$room_type))
data$room_shared <- as.numeric(data$room_shared)
data$room_private <- as.numeric(data$room_private)
data$host_is_superhost <- as.numeric(data$host_is_superhost)
```

Next, we need to split the dataset into a training- and a test set, for traning and testing the two models we will use:

```
# Split the dataset into training and test sets
set.seed(42)
sample_size <- floor(0.8 * nrow(data))
train_index <- sample(seq_len(nrow(data)), size = sample_size)
train <- data[train_index, ]
test <- data[-train_index, ]
```

## Methods

• Explain briefly how each method works, what its strengths and weaknesses are, both in general but also in the light of your project (how suitable is the method in your case?). 3 • Describe which hyperparameters are optimized for the methods (e.g., a shrinkage factor is a hyperparameter in Lasso). • Describe clearly how you evaluate the performance of the different models and methods (accuracy, MSE, misclassification error, CV error,. . . ). Explain how each metric is calculated, and why it is a useful measure of model performance. • (optional) Consider and describe potential limitations of the methods and the chosen evaluation metrics.

### Method 1 A deep learning model using ReLU

The ReLU activation function introduces non-linearity but it does so using linear segments.

The method is implemented with the following features:

- ReLU activation function
- The model weights are updated 50 times on the entire data set
- The batch size of the training data is 32
- 20% of the data is used for training, the rest for validation
- MSE is the loss function used to measure the difference between the predicted and actual values
- MAE is used to measure the absolute difference between predicted and actual values
- The model has 4 hidden layers with 60, 100, 20 and 20 units

The Keras library functions have features such as an optimizer ("Adam optimizer") and X, which in many cases contribute to improved results.

```
# Set up the model architecture
model <- keras_model_sequential() %>%
    layer_dense(units = 60, activation = "relu", input_shape = ncol(train) - 1) %>%
    layer_dense(units = 100, activation = "relu") %>%
    layer_dense(units = 20, activation = "relu") %>%
    layer_dense(units = 20, activation = "relu") %>%
    layer_dense(units = 1)

# Compile the model
model %>%
    compile(loss = "mean_squared_error", optimizer = optimizer_adam(learning_rate = 1e-04),
        metrics = c("mean_absolute_error"))

# Train the model
history <- model %>%
    fit(as.matrix(train %>%
        select(-realSum)), as.matrix(train[["realSum"]]), epochs = 50, batch_size = 32,
        validation_split = 0.2, verbose = 1)

# Evaluate the model on the test set
model %>%
    evaluate(as.matrix(test %>%
        select(-realSum)), as.matrix(test[["realSum"]]))
```

```
##                loss mean_absolute_error
##          6209.90625            59.31735
```

```
# Make predictions using the trained model
predictions <- model %>%
```

```
    predict(as.matrix(test %>%
        select(-realSum)))
```

### Method 2 Random Forest model

Random forest is a method that constructs several decision trees and then aggregates their predictions. It doesn't make any assumptions about the underlying distribution of the data, but works by iteratively partitioning the feature space and fitting decision trees to each partition. Each decision tree makes a prediction, and the final prediction is the average or the mode of the predictions of all the trees.

```
# Train the random forest model
rf_model <- randomForest(realSum ~ ., data = train, ntree = 800, mtry = 4, importance = TRUE)

# Make predictions using the trained random forest model
predictions_rf <- predict(rf_model, newdata = test)
```

It is interesting to see which attributes is the most important ones in the classification, this we can do with the varImp()-function for Random Forest.

```
varImp(rf_model, scale = TRUE)
```

```
##                              Overall
## ...1                        4.840278
## room_type                  12.393053
## room_shared                 7.082236
## room_private               11.945738
## person_capacity            14.100942
## host_is_superhost           5.788405
## multi                       4.310304
## biz                         3.359930
## cleanliness_rating          1.466912
## guest_satisfaction_overall  3.680870
## bedrooms                   14.769628
## dist                       10.046156
## metro_dist                  9.183405
## attr_index                 13.314266
## attr_index_norm            13.477607
## rest_index                 11.341268
## rest_index_norm            11.082192
## lng                        11.724951
## lat                        18.544589
```

As we can see, it's *lat* that is the most important feature, followed by the number of *bedrooms* and *person_capacity*.

## Results and interpretation

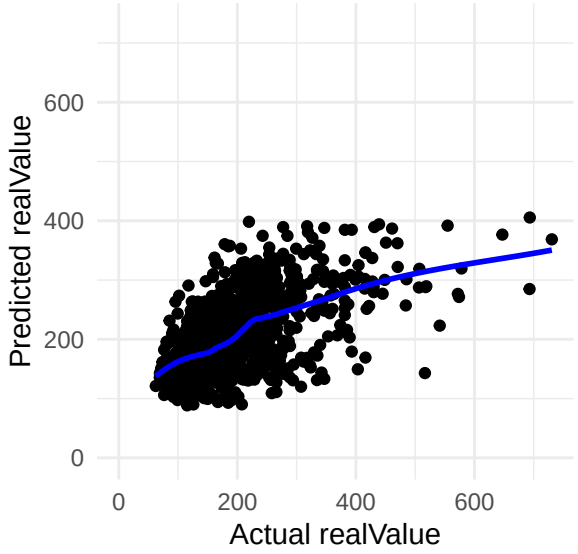
We expected the random forest to be slightly worse than the ReLU based deep learning model. The results we got are XX.

### Results for model with deep learning

```
# Add predictions to the test dataset
test$predicted_realSum <- predictions

# Find the maximum value of realSum and predicted_realSum
max_val <- max(max(test$realSum), max(test$predicted_realSum))

# Create a scatter plot of predicted realValue vs actual realValue
ggplot(test, aes(x = realSum, y = predicted_realSum)) + geom_point() + geom_smooth(method = "loess",
    span = 0.5, se = FALSE, color = "blue") + xlab("Actual realValue") + ylab("Predicted realValue") +
    theme_minimal() + xlim(0, max_val) + ylim(0, max_val) + coord_fixed()
```



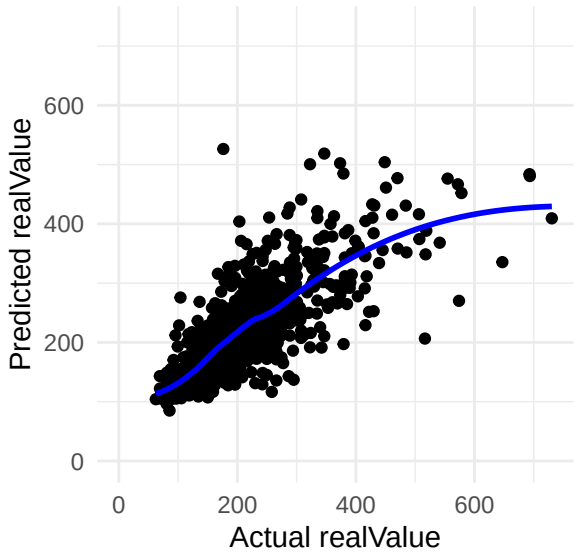

### Results for model with Random Forest

```
# Calculate the mean squared error
mae <- mean(abs(test$realSum - predictions_rf))
print(paste("Mean Absolute Error:", mae))
```

```
## [1] "Mean Absolute Error: 41.3787351412344"
```

```
# Add the predicted realValue to the test dataset
test$predicted_realSum <- predictions_rf

# Create a scatter plot of predicted realValue vs actual realValue
ggplot(test, aes(x = realSum, y = predicted_realSum)) + geom_point() + geom_smooth(method = "loess",
    span = 0.5, se = FALSE, color = "blue") + xlab("Actual realValue") + ylab("Predicted realValue") +
    theme_minimal() + xlim(0, max_val) + ylim(0, max_val) + coord_fixed()
```


Predicted realValue
600
400
200
0
0
200
400
600
Actual realValue

## Summary