Machine Learning - Übung 1

Estimate the price of real estate using linear regression with different features, Module *ml*@FHNW.

```
% Initialization
clear ; close all; clc
```

Excercise 1

Load data

The data used for the upcoming tasks will be read from a given file `house_data.csv`. The description of the data and their origin is not part of this assignment. Below the first five rows will be displayed:

```
% Load data
data = importData('house_data.csv', 5);
```

```
ans = 5×19 table
```

...

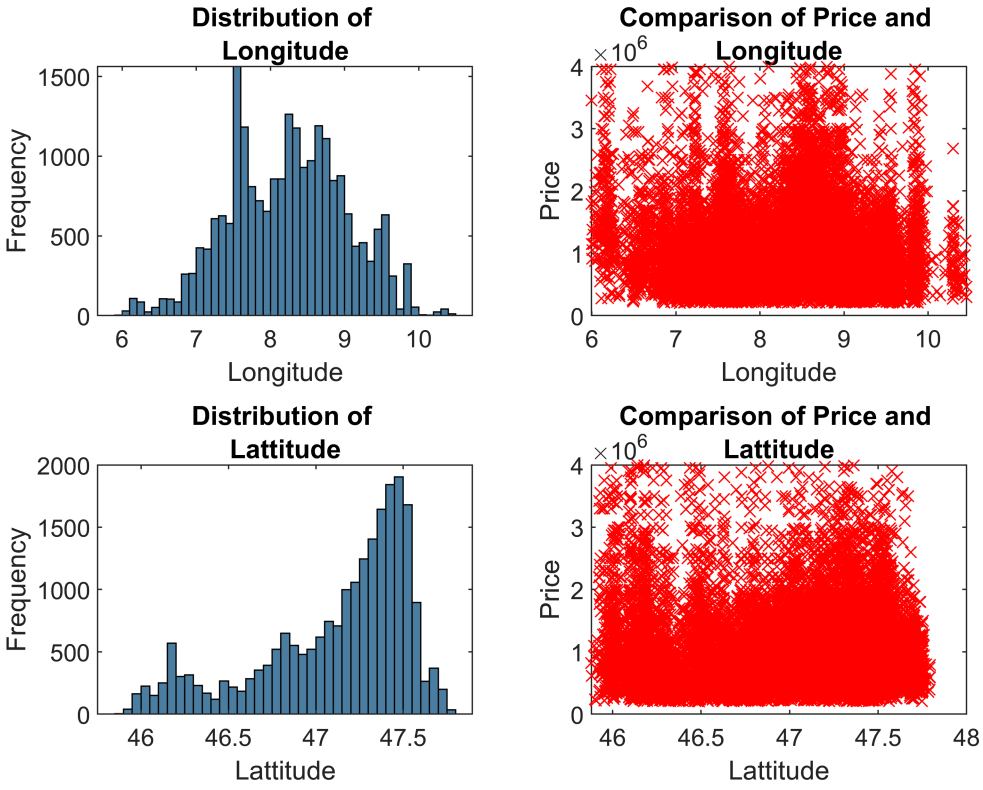
	build_year	lat	living_area	long	municipality_name	zipcode
1	1990	47.0098...	110	8.48378...	'Vitznau'	6354
2	2017	46.8721...	120	9.88018...	'Klosters-Serneus'	7250
3	2010	46.5191...	107	6.52588...	'Préverenges'	1028
4	2018	47.5213...	103	8.53697...	'Bülach'	8180
5	2007	46.5102...	95	9.85242...	'Celerina/Schlarigna'	7505

Visualize data

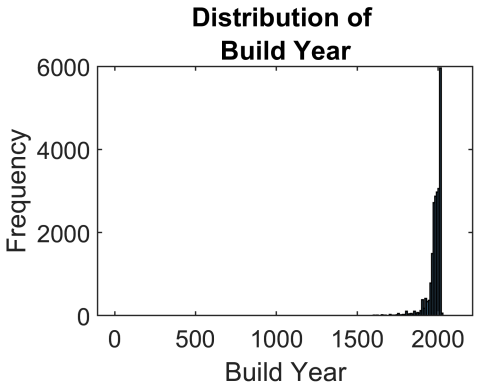
The loaded data is visualized in a next step. For all numerical features a histogram plot of the values is created to get a rough idea of the distribution of the values. In addition, a scatter plot is used to display the correlation between the respective feature and the feature `price`.

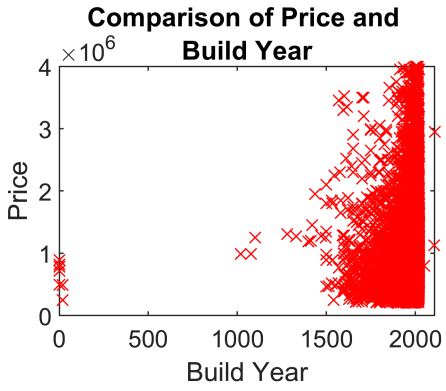
```
% Initialize variables
buildYear = table2array(data(:, {'build_year'}) );
lat = table2array(data(:, {'lat'}) );
long = table2array(data(:, {'long'}) );
livingArea = table2array(data(:, {'living_area'}) );
zipcode = table2array(data(:, {'zipcode'}) );
numRooms = table2array(data(:, {'num_rooms'}) );
price = table2array(data(:, {'price'}));
waterPercentage = table2array(data(:, {'water_percentage_1000'}) );
travelTimePrivateTransport = table2array(data(:, {'travel_time_private_transport'}) );
numbersOfBuildingsInHectare = table2array(data(:, {'number_of_buildings_in_hectare'}) );
numbersOfApartmentsInHectare = table2array(data(:, {'number_of_apartments_in_hectare'}) );
numbersOfWorkplacesInHectare = table2array(data(:, {'number_of_workplaces_in_hectare'}) );
numbersOfWorkplacesSector1InHectare = table2array(data(:, {'number_of_workplaces_sector_1_in_he
numbersOfWorkplacesSector2InHectare = table2array(data(:, {'number_of_workplaces_sector_2_in_he
numbersOfWorkplacesSector3InHectare = table2array(data(:, {'number_of_workplaces_sector_3_in_he
populationInHectare = table2array(data(:, {'population_in_hectare'}) );
```

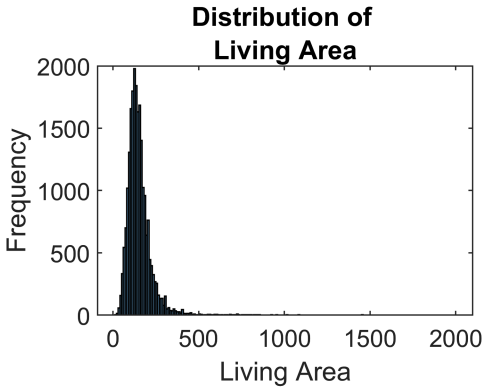
```
% Visualize data
visualizeNumericalData(long, price, 1, "Longitude");
visualizeNumericalData(lat, price, 3, "Lattitude");
```

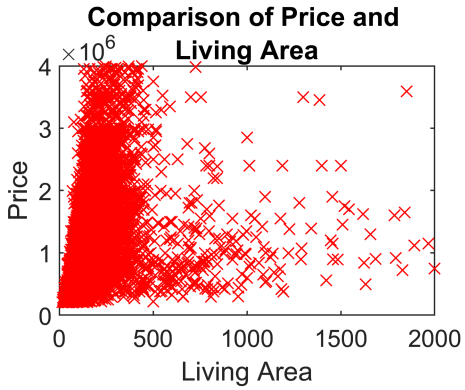



```
visualizeNumericalData(buildYear, price, 1, "Build Year");
visualizeNumericalData(livingArea, price, 3, "Living Area");
```

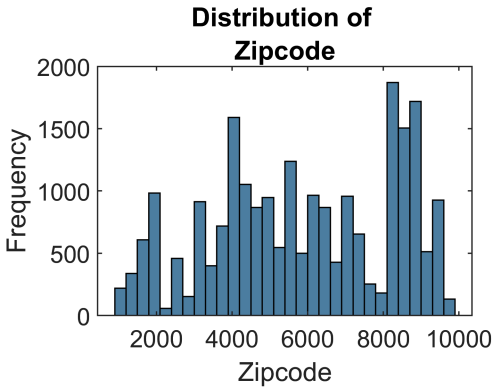


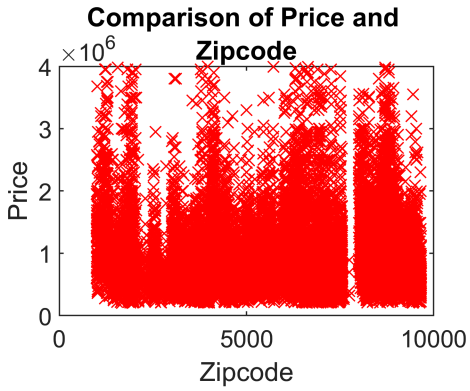


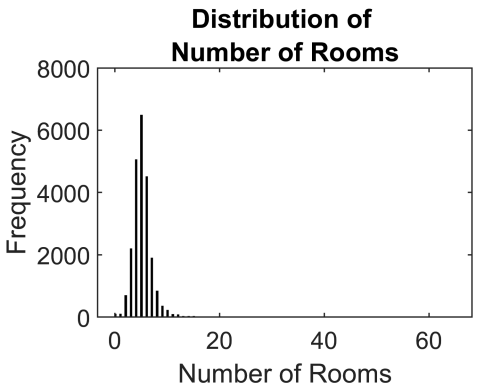


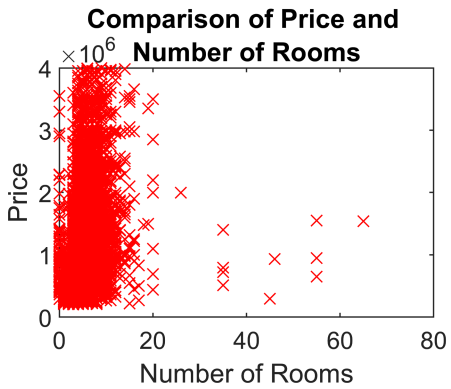



```
visualizeNumericalData(zipcode, price, 1, "Zipcode");
visualizeNumericalData(numRooms, price, 3, "Number of Rooms");
```

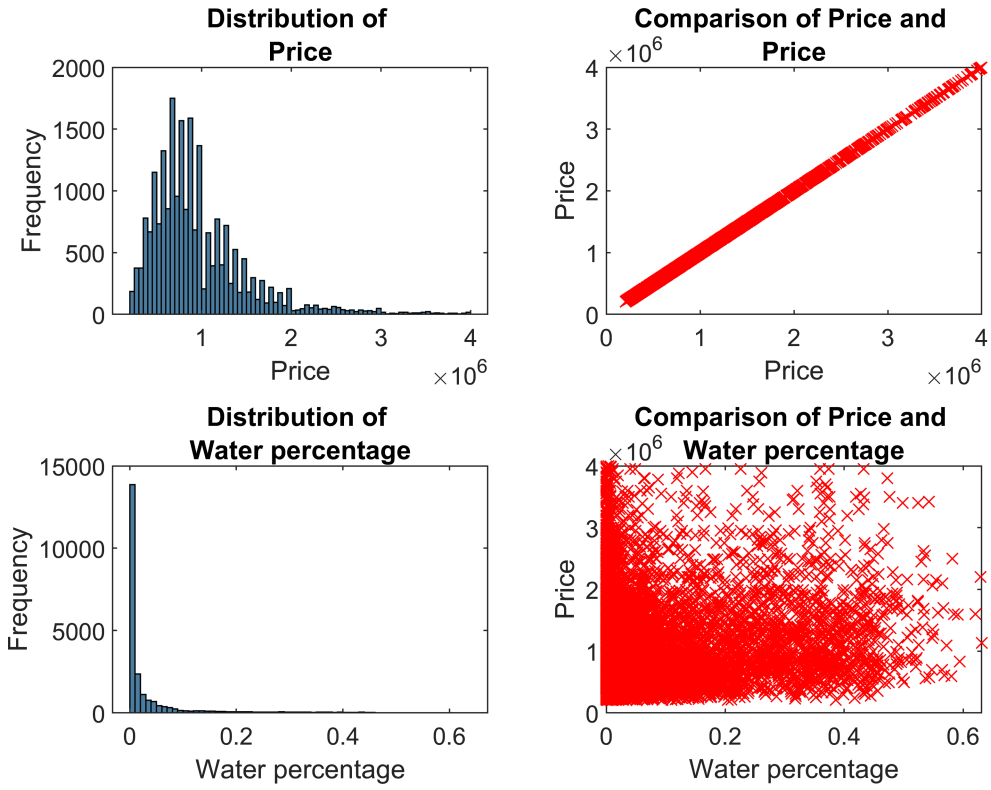




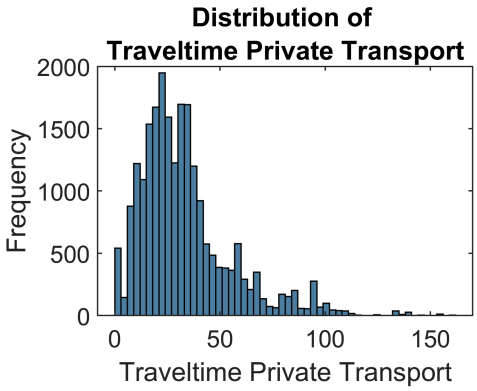


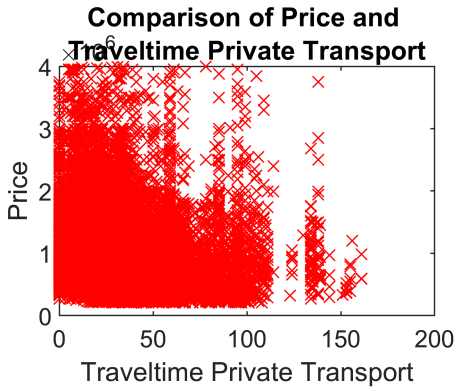


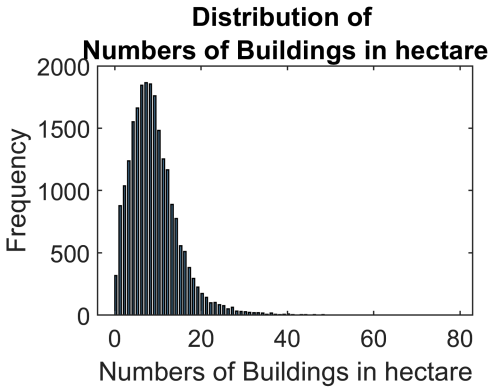
```
visualizeNumericalData(price, price, 1, "Price");
visualizeNumericalData(waterPercentage, price, 3, "Water percentage");
```

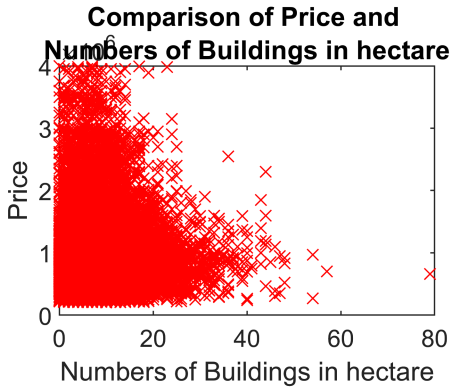



```
visualizeNumericalData(travelTimePrivateTransport, price, 1, "Traveltime Private Transport");
visualizeNumericalData(numbersOfBuildingsInHectare, price, 3, "Numbers of Buildings in hectare"
```

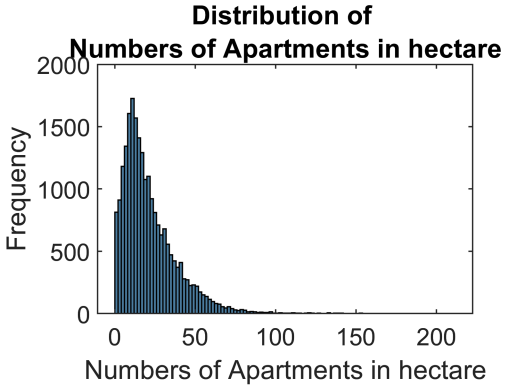


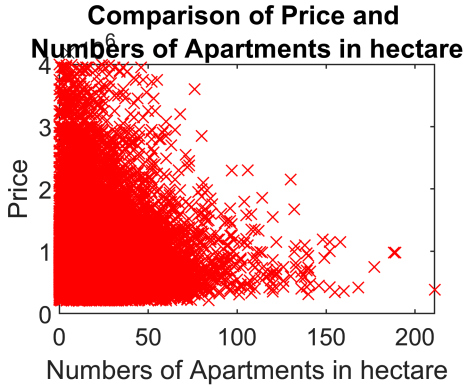


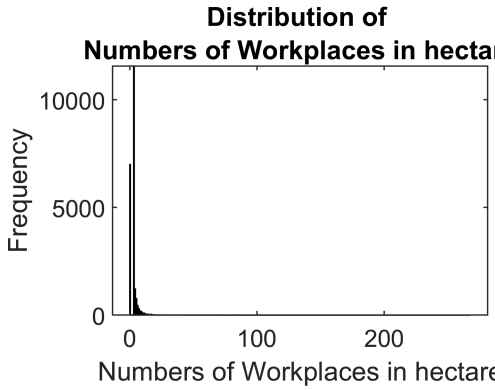


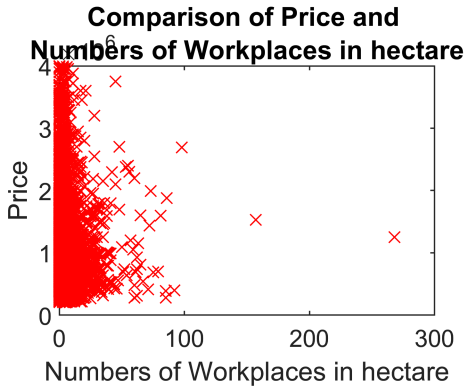



```
visualizeNumericalData(numbersOfApartmentsInHectare, price, 1, "Numbers of Apartments in hectar
visualizeNumericalData(numbersOfWorkplacesInHectare, price, 3, "Numbers of Workplaces in hectar
```

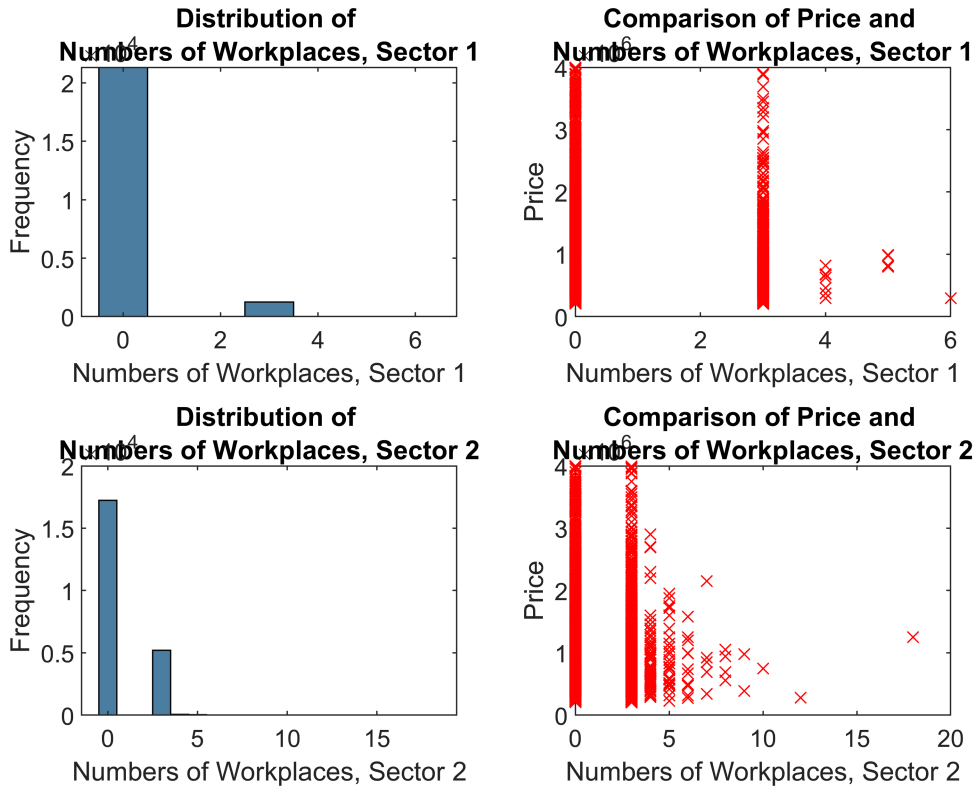




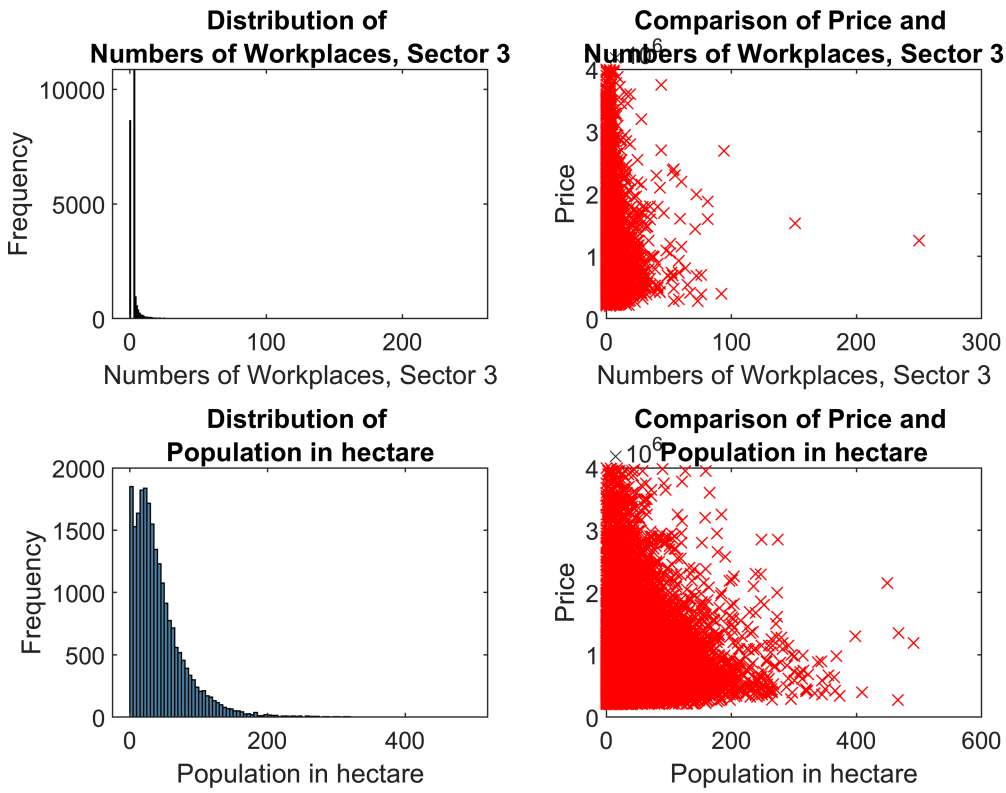




```
visualizeNumericalData(numbersOfWorkplacesSector1InHectare, price, 1, "Numbers of Workplaces, S
visualizeNumericalData(numbersOfWorkplacesSector2InHectare, price, 3, "Numbers of Workplaces, S
```

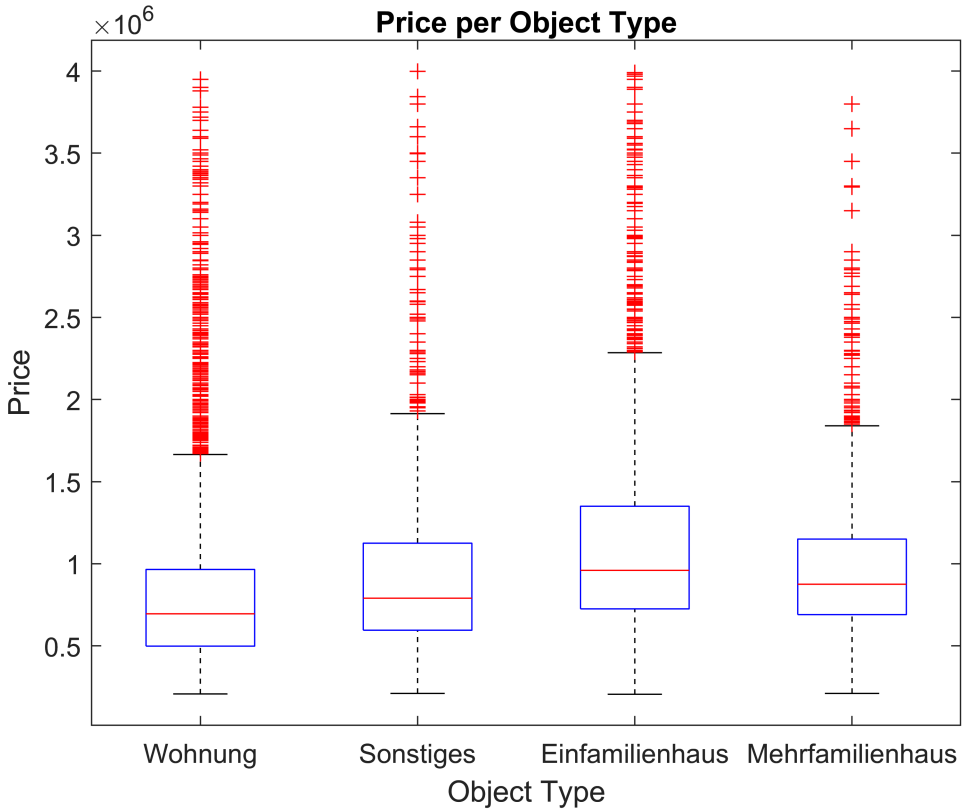



```
visualizeNumericalData(numbersOfWorkplacesSector3InHectare, price, 1, "Numbers of Workplaces, S
visualizeNumericalData(populationInHectare, price, 3, "Population in hectare");
```



The only categorical feature `object type` is represented by a boxplot. In the boxplot we can see the distribution of the achieved prices per object type.

```
objectTypeName = table2array(data(:, {'object_type_name'}) );   % X refers to the object type
price = table2array(data(:, {'price'}));                        % y refers to the price
m = length(price);                                              % m refers to the number of tra

% Visualize categorical data
visualizeCategoricalData(objectTypeName, price, 'Object Type');
```



Excercise 2

Normalize feature `living area`

To standardize the range of the feature, mean normalization is applied to the feature `living area`. The normalization is done according to https://en.wikipedia.org/wiki/Feature_scaling#Mean_normalization.

```
% Normalize feature
livingAreaNorm = featureNormalize(livingArea);
```

Predict price using gradient descent

In the next step gradient descent is used, to predict the price using the given and normalized feature `living area`.

```
X = [ones(m, 1), livingAreaNorm];    % Add a column of ones to living area, which refers X
y = price;                           % y refers to price
theta = zeros(2, 1);                 % Initialize fitting parameters
iterations = 3000;
alpha = 0.01;

% Running gradientDescent
[theta, J_history] = gradientDescent(X, y, theta, alpha, iterations);

% Print theta to screen
fprintf('Theta found by gradient descent:\n');
```

Theta found by gradient descent:

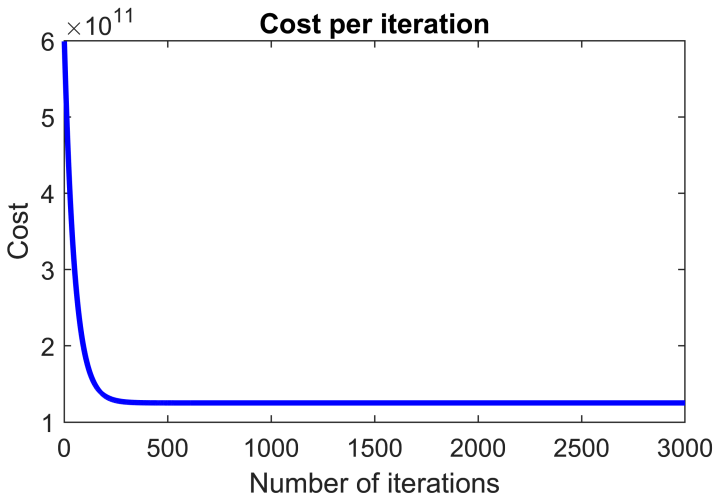
```
fprintf('%f\n', theta);
```

```
960568.080018
210590.105394
```

Visualize costs per iteration

To ensure that the gradient descent procedure has been implemented correctly, the development of costs per iteration is plotted:

```
% Visualize costs
set(groot,'defaultfigureposition',[600 250 400 250]) % Set default image size
plotCosts(J_history)
```

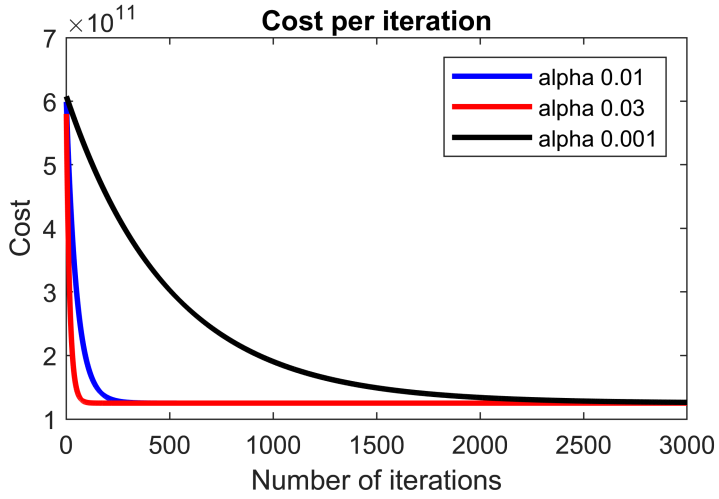


Vary alpha

To show the effect of using different parameters, the value of the learing rate `alpha` is changed in the following executions. The larger `the` alpha is chosen, the faster the costs settle down. However, it must be taken care that no underfitting occurs.

```
X = [ones(m, 1), livingAreaNorm];    % Add a column of ones to living area, which refers X
y = price;                           % y refers to price
iterations = 3000;

% Set alpha to 0.03
theta003 = zeros(2, 1);     % Reset fitting parameters
[theta003, J_history003] = gradientDescent(X, y, theta003, 0.03, iterations);
hold on;                    % Add line to existing plot
plot(1:numel(J_history003), J_history003, 'r', 'LineWidth', 2);

% Set alpha to 0.001
theta0001 = zeros(2, 1);    % Reset fitting parameters
[theta0001, J_history0001] = gradientDescent(X, y, theta0001, 0.001, iterations);
hold on;                    % Add line to existing plot
plot(1:numel(J_history0001), J_history0001, 'k', 'LineWidth', 2);
```

```
% Add legends to plot
legend('alpha 0.01', 'alpha 0.03', 'alpha 0.001')
```



Predict price using normal equiation

To verify gradient descent results normal quidation is used, to predict the price using the given and normalized feature `living area`.

```
X = [ones(m, 1), livingAreaNorm];    % Add a column of ones to living area, which refers X
y = price;                           % y refers to price

% Calculate the parameters from the normal equation
thetaNormalEqn = normalEqn(X, y);

% Display normal equation's result
fprintf('Theta computed from the normal equation: \n');
```

```
Theta computed from the normal equation:
```

```
fprintf(' %f \n', thetaNormalEqn);
```

```
 960568.080018
 210590.105394
```

```
fprintf('\n');
```

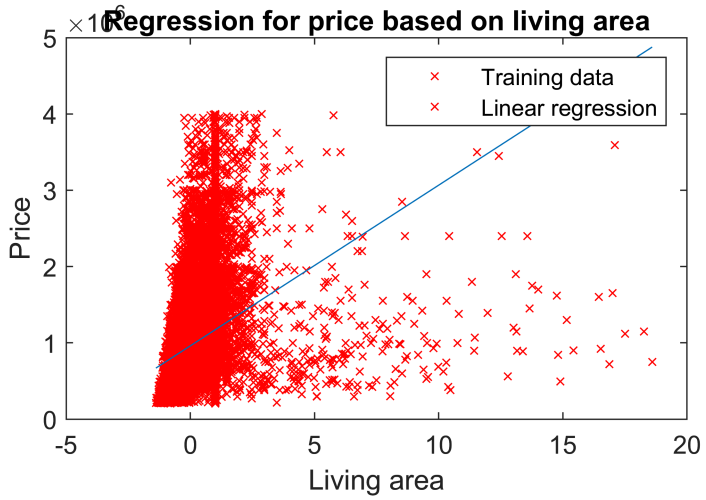
While the learning rate and number of iterations must be determined for the gradient descent procedure, this is omitted for the calculation using normal equation. If a large number of features are examined, the calculation using normal equiation can become more expensive. The results of both gradient descent and normal equation are consistent. In both cases we get `960568.080018` and `210590.105394` as optimal elements for theta.

Visualize found regression

In the following plot, the found regression is added to our former plot, where we displayed data points for `price` on the y axis and `living area on the x axis`. Already in this plot it can be seen that the linear regression

fits only conditionally to the data. Especially when predicting the price of objects with a large living space, the predictions seem inaccurate.

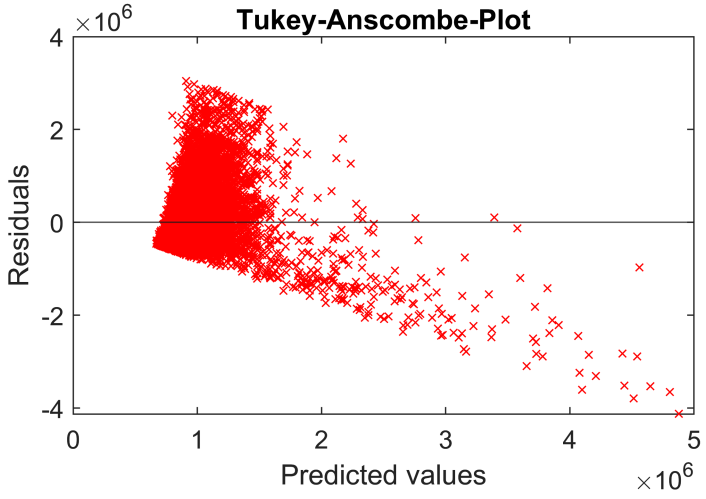
```
X = [ones(m, 1), livingAreaNorm];    % Add a column of ones to living area, which refers X
y = price;                           % y refers to price

% Plot Regression
plotRegression(X, y, theta);
```



Visualize residuals

Residuals are deviations from the expected value. In a tukey-anscombe-plot those residuals can be visualized. The tukey-anscombe plot is further described under http://stat.ethz.ch/~stahel/courses/regression/reg-resanal.pdf. In the plot a trend can be seen that the predicted values are less accurate the bigger the values are. In addition, the forecasts for large values tend to be too low.

```
% Tukey-Anscombe-Plot
[mu, sigma] = plotResiduals(X, theta, y);
```



The high deviations of the model to the effective prices already assumed in the plot are underlined by the values obtained for mean value and standard deviation:

```
% Display mean and sigma
fprintf(' Mean: %f \n', mu);
```

```
Mean: 0.000000
```

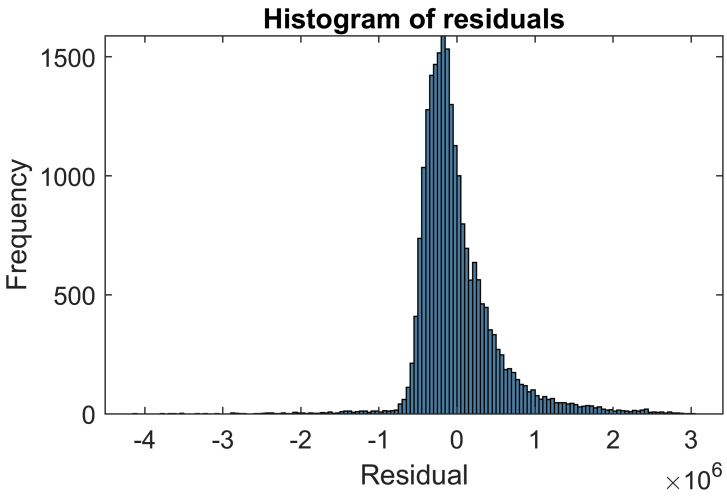
```
fprintf(' Standard deviation: %f \n', sigma);
```

```
Standard deviation: 500349.786811
```

```
fprintf('\n');
```

Furthermore the distribution of the residuals can be visualized in a histogram.

```
% Plot histrogram
plotHistogram(X, theta, y)
```



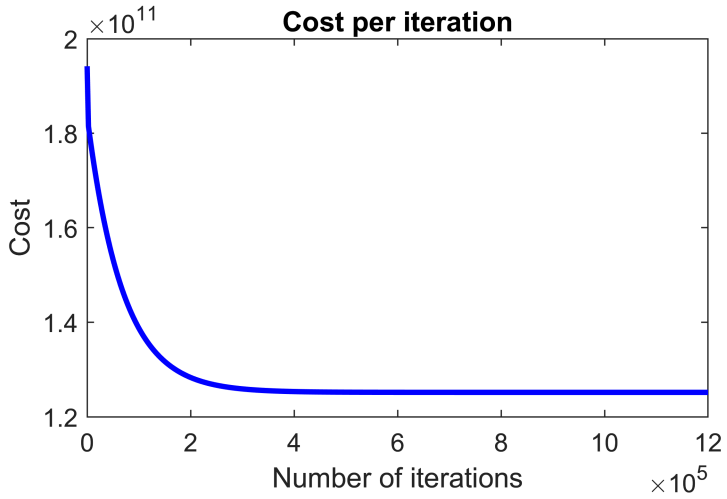
The histogram shows the frequencies of the respective residuals. Also in this diagram it can be seen that the differences are mostly in a range from -1'000'000 to 1'000'000. Our assumption of a linear model seems to be inaccurate.

Predict price using non-normalized feature *living area*

In the following the linear regression is calculated with the non-normalized feature `living area`. For the gradient descent procedure to work, the number of iterations must be very high and the `alpha` very low. Due to the high number of iterations and the low learning rate, the complete process takes a relatively long time. The following example shows a run using 1'200'000 iterations and a learning rate of 0.000025:

```
X = [ones(m, 1), livingArea];   % Add a column of ones to x
thetaNonNorm = zeros(2, 1);     % Initialize fitting parameters
iterations = 1200000;
alpha = 0.000025;               % Use small alpha

% Run gradient descent without normalization
[thetaNonNorm, J_history] = gradientDescent(X, y, thetaNonNorm, alpha, iterations);

% Visualize costs
```

```
plotCosts(J_history)
```



The result can be checked again with normal equiation. Various attempts show that the values obtained are only approximately the same.

```
X = [ones(m, 1), livingArea];    % Add a column of ones to living area, which refers X
y = price;                       % y refers to price

% Calculate the parameters from the normal equation
thetaNormalEqnNonNorm = normalEqn(X, y);

% Print theta to screen
fprintf('Theta found by gradient descent:\n');
```

```
Theta found by gradient descent:
```

```
fprintf('%f\n', thetaNonNorm);
```

```
632584.058974
2123.445110
```

```
% Display normal equation's result
fprintf('Theta computed from the normal equations: \n');
```

```
Theta computed from the normal equations:
```

```
fprintf(' %f \n', thetaNormalEqnNonNorm);
```

```
 632683.237456
 2122.990475
```

Excercise 3

Predict price using gradient descent

In the next step gradient descent is used, to predict the price using the given and normalized feature `living area`. Instead of linear, price is assumpted to be a logarithmic-function and therefore transformed to *log(price)*.

```
X = [ones(m, 1), livingAreaNorm];    % Add a column of ones to living area, which refers X
```

```
y = log(price);                      % y refers to price
thetaLog = zeros(2, 1);              % Initialize fitting parameters
iterations = 3000;
alpha = 0.01;

% Run gradientDescent
[thetaLog, J_historyLog] = gradientDescent(X, y, thetaLog, alpha, iterations);

% Print theta to screen
fprintf('Theta found by gradient descent:\n');
```

```
Theta found by gradient descent:
```

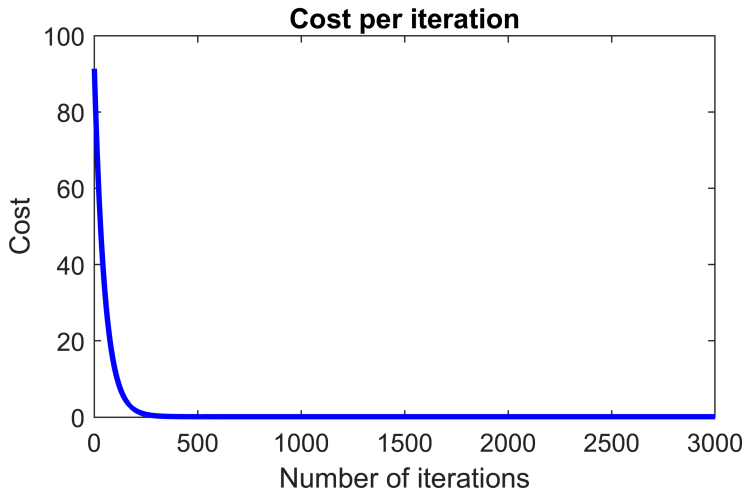
```
fprintf('%f\n', thetaLog);
```

```
13.641726
0.201981
```

Visualize costs per iteration

To ensure that the gradient descent procedure has been implemented correctly, the development of costs per iteration is plotted. In comparison to the previous model, it can be seen that the model is more quickly stabilising at a certain value.

```
% Visualize costs
plotCosts(J_historyLog)
```

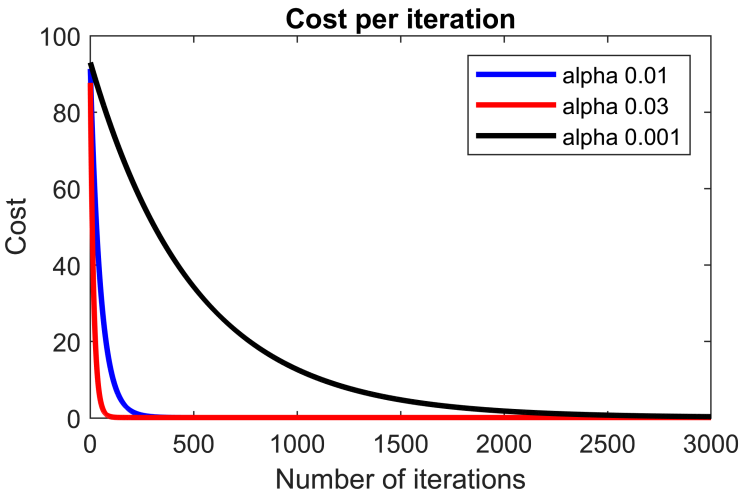


Vary alpha

To show the effect of using a different parameters, the value of alpha ist changed in the following executions:

```
X = [ones(m, 1), livingAreaNorm];    % Add a column of ones to living area, which refers X
y = log(price);                      % y refers to price
iterations = 3000;

% Set alpha to 0.03
theta003 = zeros(2, 1);     % Reset fitting parameters
[theta003, J_history003] = gradientDescent(X, y, theta003, 0.03, iterations);
hold on;                    % Add line to existing plot
```

```
plot(1:numel(J_history003), J_history003, 'r', 'LineWidth', 2);

% Set alpha to 0.001
theta0001 = zeros(2, 1);    % Reset fitting parameters
[theta0001, J_history0001] = gradientDescent(X, y, theta0001, 0.001, iterations);
hold on;                    % Add line to existing plot
plot(1:numel(J_history0001), J_history0001, 'k', 'LineWidth', 2);

% Add legends to plot
legend('alpha 0.01', 'alpha 0.03', 'alpha 0.001')
```



Predict price using normal equiation

To verify gradient descent results normal quidation is used, to predict the price using the given and normalized feature `living area`. Again we receive the same results for gradient descent and normal equiation.

```
X = [ones(m, 1), livingAreaNorm];    % Add a column of ones to living area, which refers X
y = log(price);                      % y refers to price

% Calculate the parameters from the normal equation
thetaNormalEqnLog = normalEqn(X, y);

% Display normal equation's result
fprintf('Theta computed from the normal equations: \n');
```

```
Theta computed from the normal equations:
```

```
fprintf(' %f \n', thetaNormalEqnLog);
```

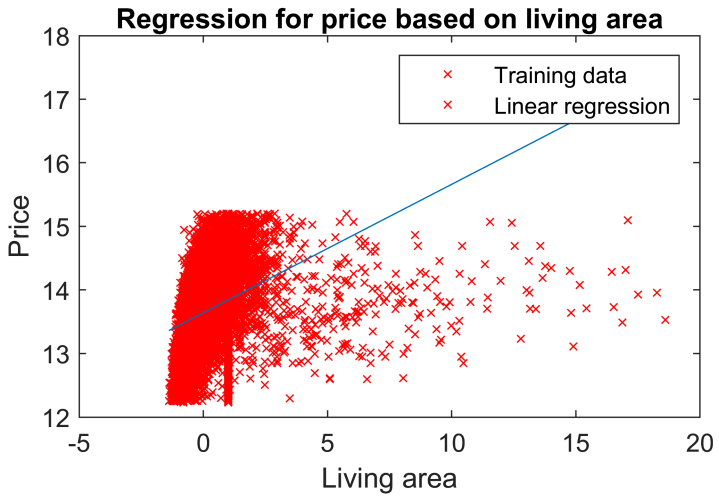
```
 13.641726
 0.201981
```

```
fprintf('\n');
```

Visualize found regression

In the following plot, the found regression is added to our former plot, where we displayed data points for `log(price)` and `living area`. In general, plots show that price developments often behave approximately logarithmically and that the model therefore delivers better results. In the next plot, these findings are still difficult to see, but at the latest the calculation of the standard deviation provides corresponding results.

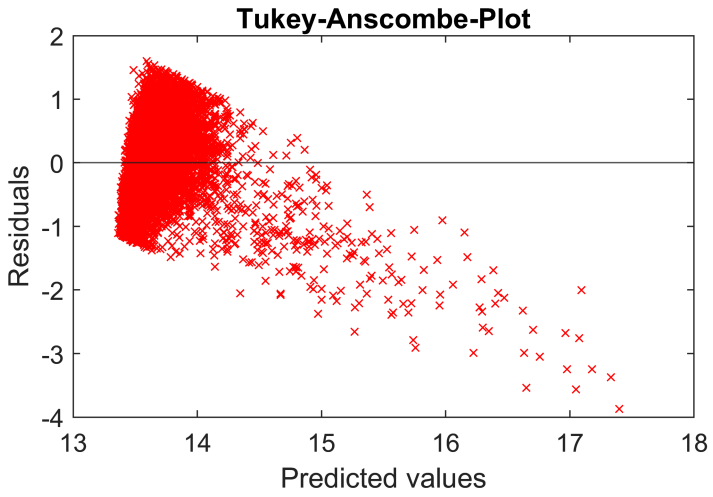
```
X = [ones(m, 1), livingAreaNorm];    % Add a column of ones to living area, which refers X
y = log(price);                      % y refers to price

% Plot Regression
plotRegression(X, y, thetaLog);
```



Visualize residuals

Residuals are deviations from the expected value. In a tukey-ansombe-plot those residuals can be visualized.

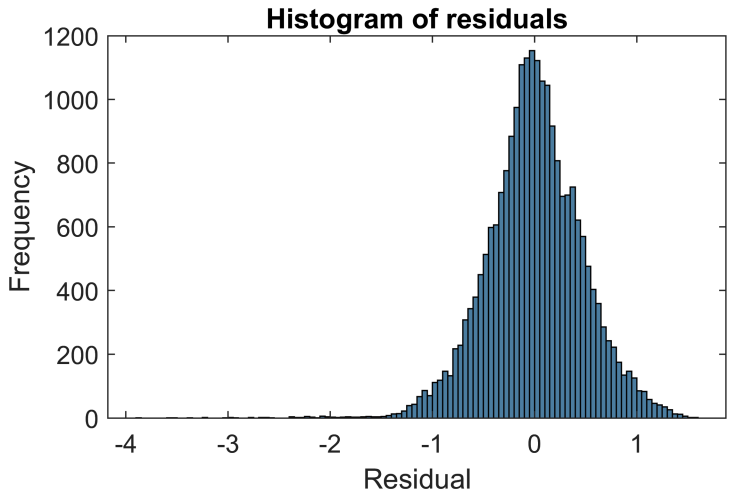
```
% Tukey-Anscombe-Plot
[mu, sigma] = plotResiduals(X, thetaLog, y);
```



Furthermore the distribution of the residuals can be visualized in a histogram. The deviations on the x-axis allow the conclusion that the model can now predict values more accurately.

```
% Plot histrogram
```

```
plotHistogram(X, thetaLog, y)
```



The above assumptions are confirmed by a significantly lower standard deviation (compared with *~500'349* using `y = price`).

```
% Display mean and sigma
fprintf(' Mean: %f \n', mu);
```

```
Mean: 0.000000
```

```
fprintf(' Standard deviation: %f \n', sigma);
```

```
Standard deviation: 0.469710
```

```
fprintf('\n');
```

Excercise 4

Calculate mean absolute percentage error

In the following section the mean absolute percentage error (MAPE) for `y = price` is calculated and the distribution of de percentage error displayed in a histogram. The MAPE of the first linear model is relatively high. This coincides with the high standard deviation of its residuals.

```
X = [ones(m, 1), livingAreaNorm];       % Add a column of ones to x
y = price;                              % y refers to price

% Compute MAPE
mape = computeMape(X,theta,y);

fprintf('Computed MAPE from y = price: \n');
```

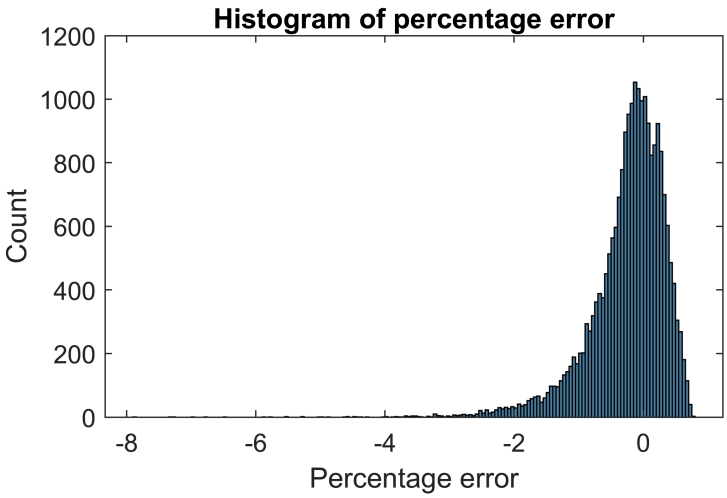
```
Computed MAPE from y = price:
```

```
fprintf(' %f \n', mape);
```

```
0.438663
```

```
fprintf('\n');
```

```
% Plot percentage error
plotPercentageError(X,theta,y)
```



The same can be done for `y = log(price)`. The comparison shows that the model makes much more accurate predictions and the percentage error is much lower.

```
X = [ones(m, 1), livingAreaNorm];       % Add a column of ones to x
y = log(price);                         % y refers to price

% Compute MAPE
mape = computeMape(X,thetaLog,y);

fprintf('Computed MAPE from y = log(price): \n');
```

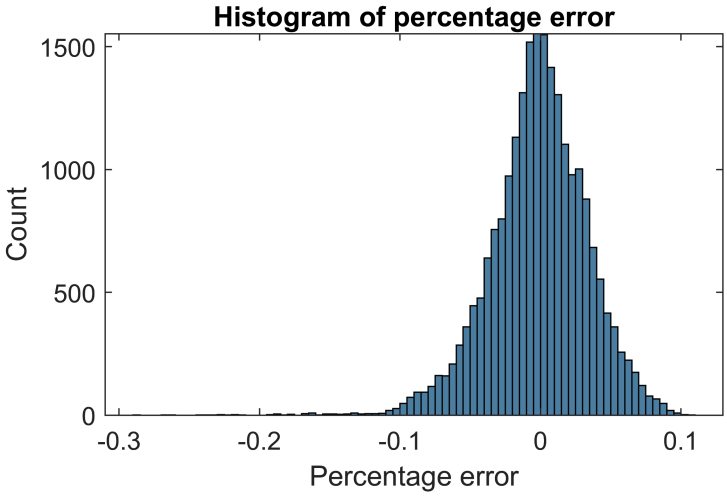
```
Computed MAPE from y = log(price):
```

```
fprintf(' %f \n', mape);
```

```
 0.026266
```

```
fprintf('\n');
```

```
% Plot percentage error
plotPercentageError(X,thetaLog,y)
```

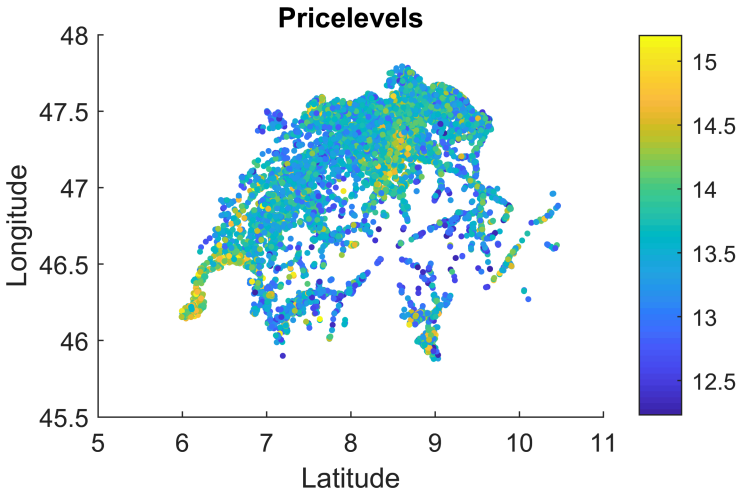


Excercise 5

Visualize price in relation to longitude/latitude

The output variable `y = log(price)` can be visualized on a map using longitude and latitude values. In the plot the yellow data points represent the most expensive objects. These buildings are located mainly in the regions of Geneva and Zurich.

```
X = [lat, long];    % X refers to the latitude and longitude
y = log(price);     % y refers to the log of price

% Visualize price on map
plotMap(X, y, "Pricelevels")
```

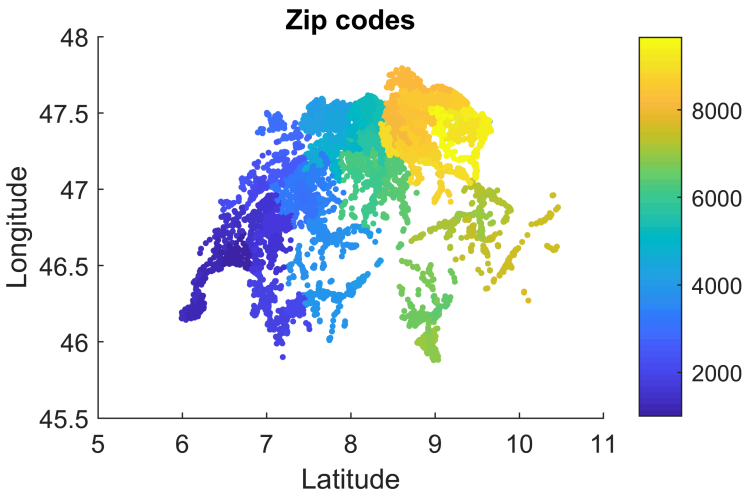


Excercise 6

Visualize zip code in relation to longitude/latitude

A similar visualization can be done using `y = zip`. The plot shows the distribution of the zip code ranges in Switzerland.

```
X = [lat, long];    % X refers to the latitude and longitude
y = zipcode;        % y refers to zip codes

% Visualize zip codes on a map
plotMap(X, y, "Zip codes")
```



Excercise 7

Predict price using additional one-hot-encoded feature `zipcode`

For the following prediction we additionally use the first two digits of the feature `zipcode` to predict an objects price. This should reduce outliers in high-price areas and allow to predict prices more accurately overall.

```
zipCodeBin = oneHotEncode(floor((zipcode/100)), 100);     % X refers to the binned zip code
X = [ones(m, 1), livingAreaNorm, zipCodeBin];             % Add a column of ones to x

y = log(price);                                           % y refers to the log of price

% Calculate the parameters from the normal equation
thetaNormalEqnAdd = normalEqn(X, y);

% Display normal equation's result
fprintf('Theta computed from the normal equations (partly shown): \n');
```

```
Theta computed from the normal equations (partly shown):
```

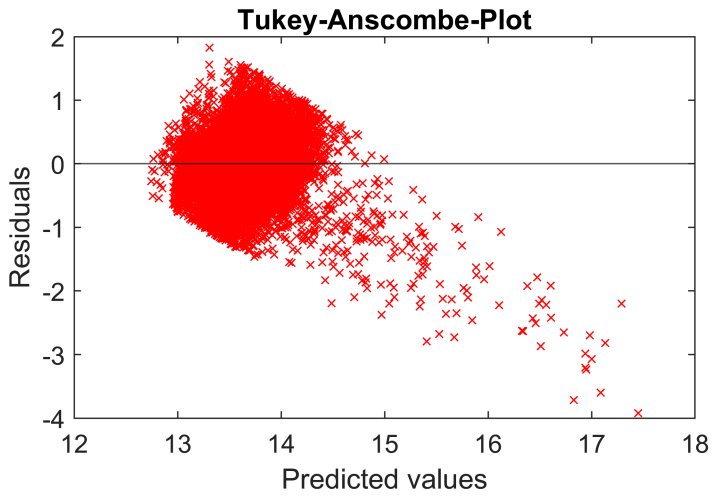
```
fprintf(' %f \n', thetaNormalEqnAdd(1:5,1));
```

```
 13.418721
 0.208330
 0.000000
 -0.000000
 -0.000000
```

```
fprintf('\n');
```

In the tukey-anscombe plot only tiny differences are visible:

```
% Tukey-Anscombe-Plot
[mu, sigma] = plotResiduals(X, thetaNormalEqnAdd, y);
```



However, the standard deviations show that the predicted values could be improved further (compared to a std. deviation of `0.469710`):

```
% Display mean and sigma
fprintf(' Mean: %f \n', mu);
```

```
Mean: -0.000000
```

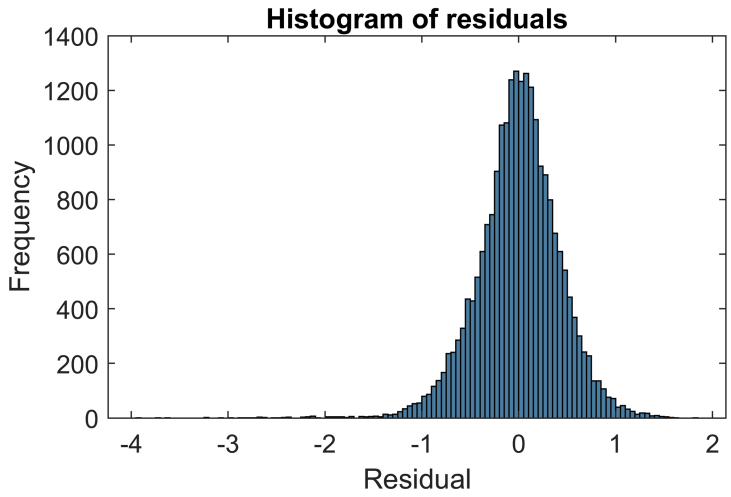
```
fprintf(' Standard deviation: %f \n', sigma);
```

```
Standard deviation: 0.425935
```

```
fprintf('\n');
```

The improved model is also reflected in the histogram. The deviations from the mean value are again smaller.

```
% Plot histogram of residuals
plotHistogram(X, thetaNormalEqnAdd, y);
```



Excercise 8

Predict price using further additional features

Based on the previous findings, we consider other features to determine the price.

```
zipCodeBin = oneHotEncode(floor((zipcode/100)), 100); % X refers to the binned zip code
X = [ones(m, 1), livingAreaNorm, buildYear, numRooms, populationInHectare, zipCodeBin]; % Add a
X = double(X);

y = log(price); % y refers to the log of price
thetaAdd = zeros(105, 1);

% Calculate the parameters from the normal equation
thetaNormalEqnAdd2 = normalEqn(X, y);

% Display normal equation's result
fprintf('Theta computed from the normal equations(partly shown): \n');
```

```
Theta computed from the normal equations(partly shown):
```

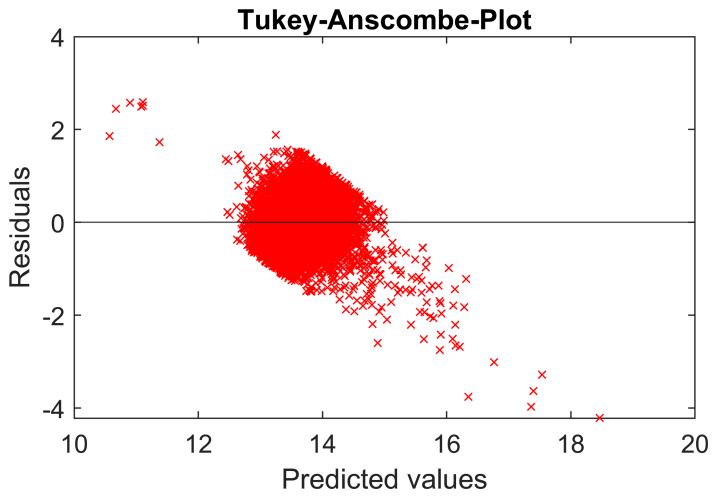
```
fprintf(' %f \n', thetaNormalEqnAdd2(1:5,1));
```

```
 10.841215
 0.137625
 0.001143
 0.076836
 -0.001264
```

```
fprintf('\n');
```

The tukey-anscombe plot shows that the model could be improved significantly:

```
% Tukey-Anscombe-Plot
[mu, sigma] = plotResiduals(X, thetaNormalEqnAdd2, y);
```



With the help of the additional features the standard deviation is even smaller.

```
% Display mean and sigma
fprintf(' Mean: %f \n', mu);
```

```
Mean: 0.000000
```

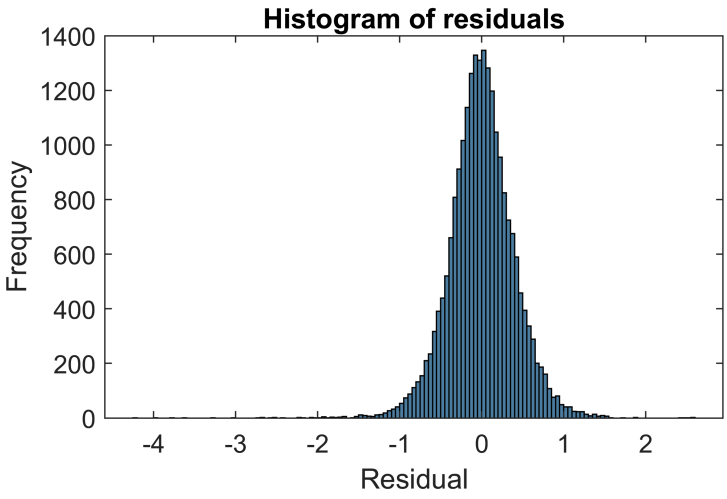
```
fprintf(' Standard deviation: %f \n', sigma);
```

```
Standard deviation: 0.399064
```

```
fprintf('\n');
```

The histogram of the residuals of this model (so far the most accurate) looks as follows:

```
% Plot histogram of residuals
plotHistogram(X, thetaNormalEqnAdd2, y);
```



Excercise 9

Predict price using mape instead of residual-sum-of-squares

Instead of the residual sum of squares (RSS), the mean abosulte percentage error (MAPE) can also be used as a cost function. The formula of the mape is defined as follows:

$$\text{MAPE} := \frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{y}_i - y_i|}{y_i}$$

In order to perform the gradient descent procedure with the mean absolute percentage error, the resulting costs must be minimized using the derivatives. The derivatives of the MAPE function are as follows:

$$\frac{\partial \text{MAPE}}{\partial \hat{y}_i} = \begin{cases} -\frac{1}{N}, & \text{if } \hat{y}_i < y_i \\ \text{undefined}, & \text{if } \hat{y}_i = y_i \\ \frac{1}{N}, & \text{if } \hat{y}_i > y_i \end{cases}$$

Within the gradient descent function, the code part in which the costs are calculated must be exchanged. In the function `gradientDescentMape` this adjustment has been made and the MAPE is used instead of RSS.

```
m = length(y);
X = [ones(m, 1), livingAreaNorm];       % Add a column of ones to x
y = log(price);                         % y refers to the log of price

thetaMape = zeros(2, 1);               % Initialize fitting parameters
iterations = 150000;
alpha = 0.0001;

% Rund gradientDescent
[thetaMape, J_historyMape] = gradientDescentMape(X, y, thetaMape, alpha, iterations);

% Print theta to screen
fprintf('Theta found by gradient descent:\n');
```

```
Theta found by gradient descent:
```

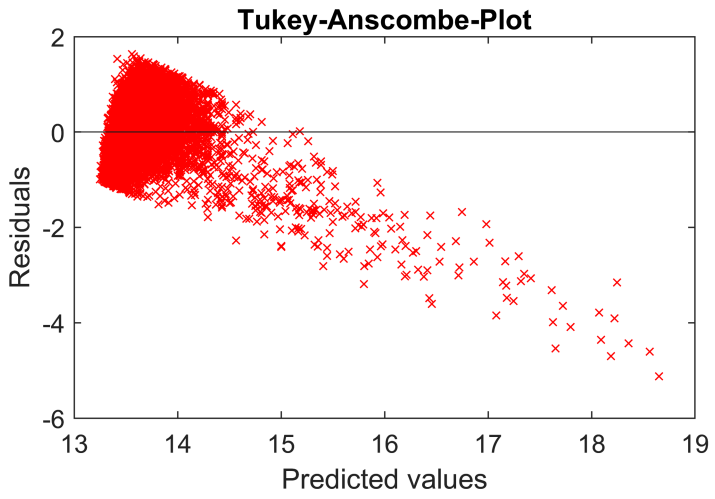
```
fprintf('%f\n', thetaMape);
```

```
13.624230
0.270425
```

```
fprintf('\n');
```

The obtained theta almost matches the results from the previous exercises and also the plot with the residuals looks plausible.

```
% Tukey-Anscombe-Plot
[mu, sigma] = plotResiduals(X, thetaMape, y);
```




```
% Display mean and sigma
fprintf(' Mean: %f \n', mu);
```

```
Mean: 0.017496
```

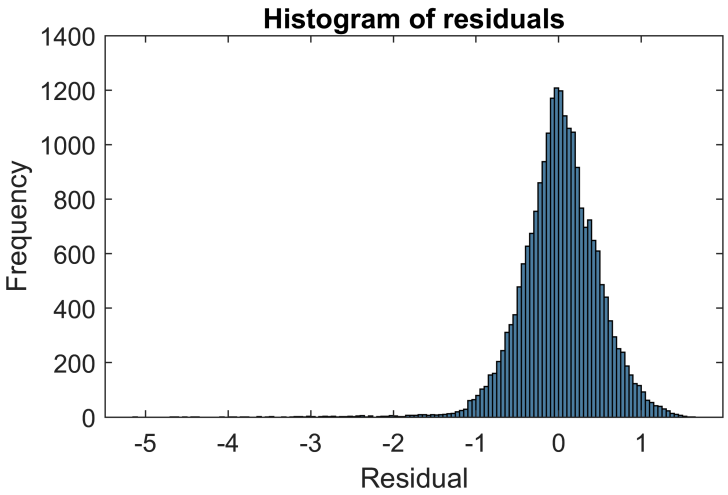
```
fprintf(' Standard deviation: %f \n', sigma);
```

```
Standard deviation: 0.474671
```

```
fprintf('\n');
```

The obtained standard deviation of ~0.5 is also reflected in the diagramm:

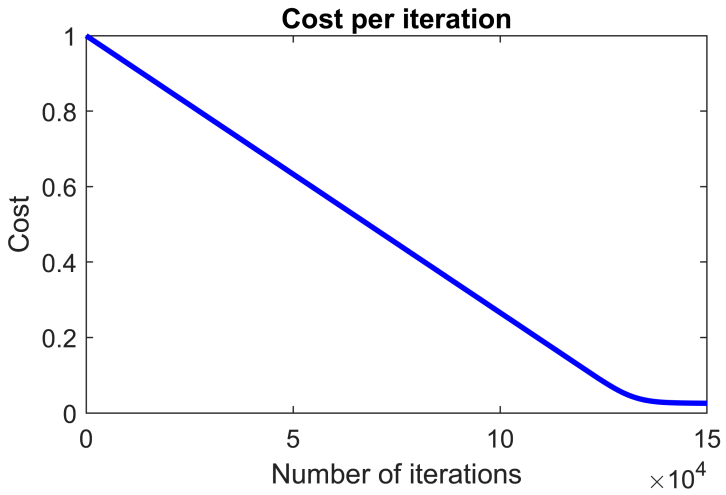
```
% Plot histogram of residuals
plotHistogram(X, thetaMape, y);
```



Visualize costs per iteration

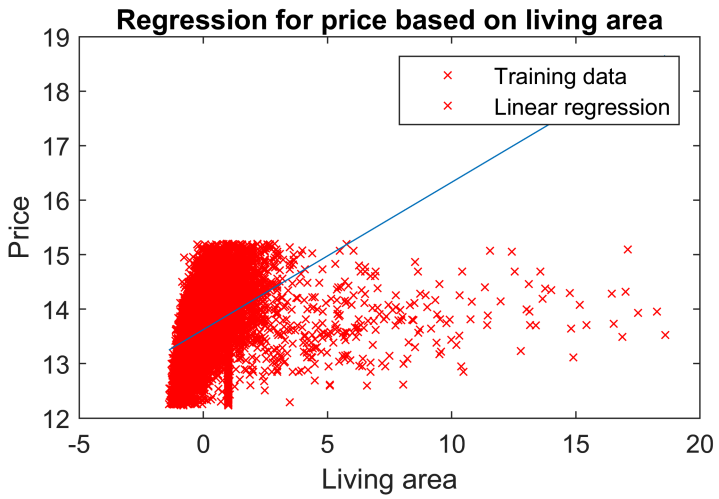
To ensure that the gradient descent procedure has been implemented correctly, the development of costs per iteration is plotted. The development of costs behaves as expected: By correcting the values using the derivations described above, costs can be minimized. The plot shows that the costs settle at a minimum.

```
% Visualize costs
plotCosts(J_historyMape)
```



In the next plot, the regression line is displayed together with the data points from the training data set. The plot shows that the function obtained corresponds roughly to the model in which RSS was used.

```
% Plot Regression
plotRegression(X, y, thetaMape);
```



If MAPE is used as a cost function, but the model is then evaluated with RSS, it must be noted that two different aspects were considered. Accordingly, the predictions could be less accurate.