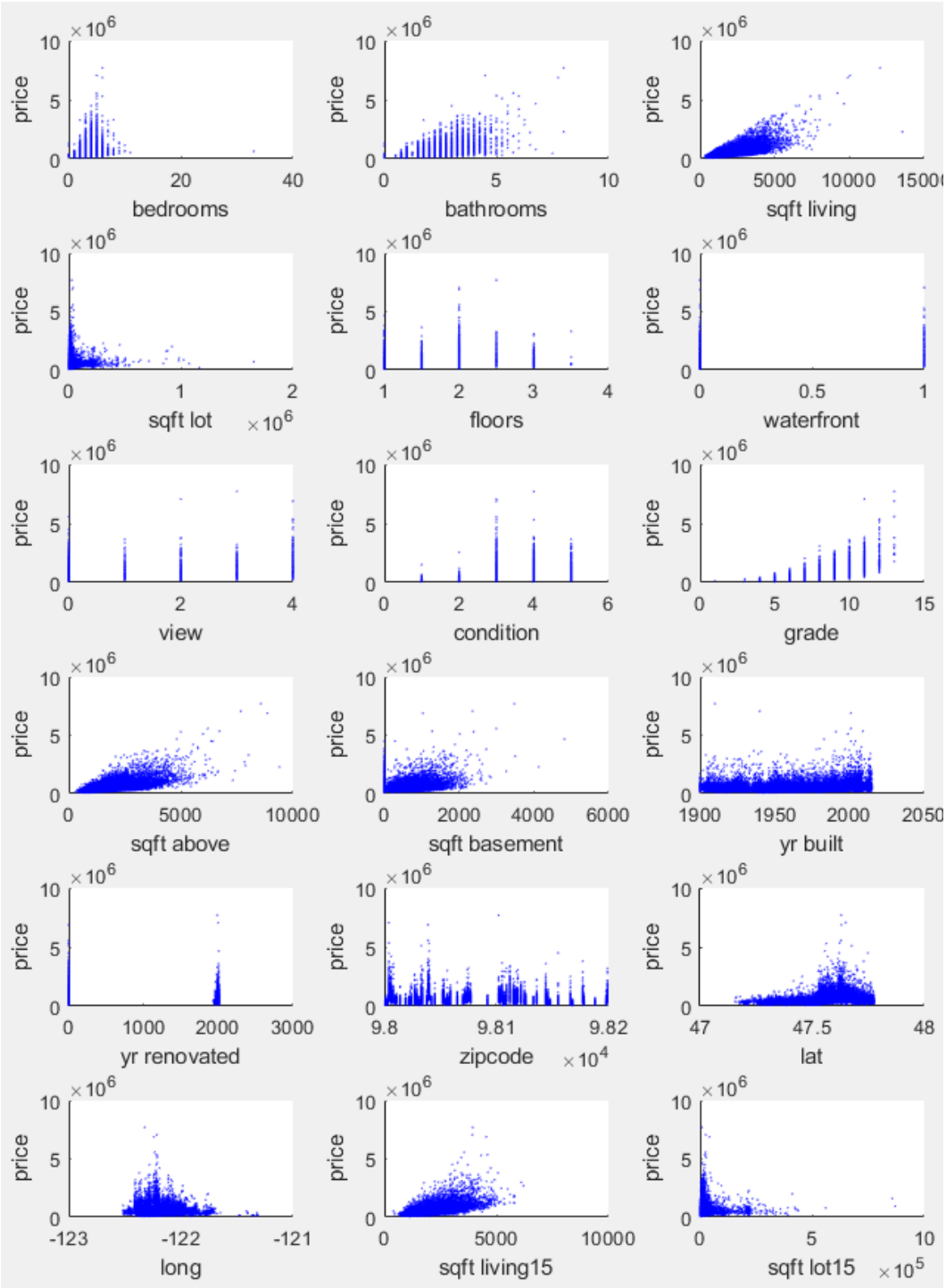
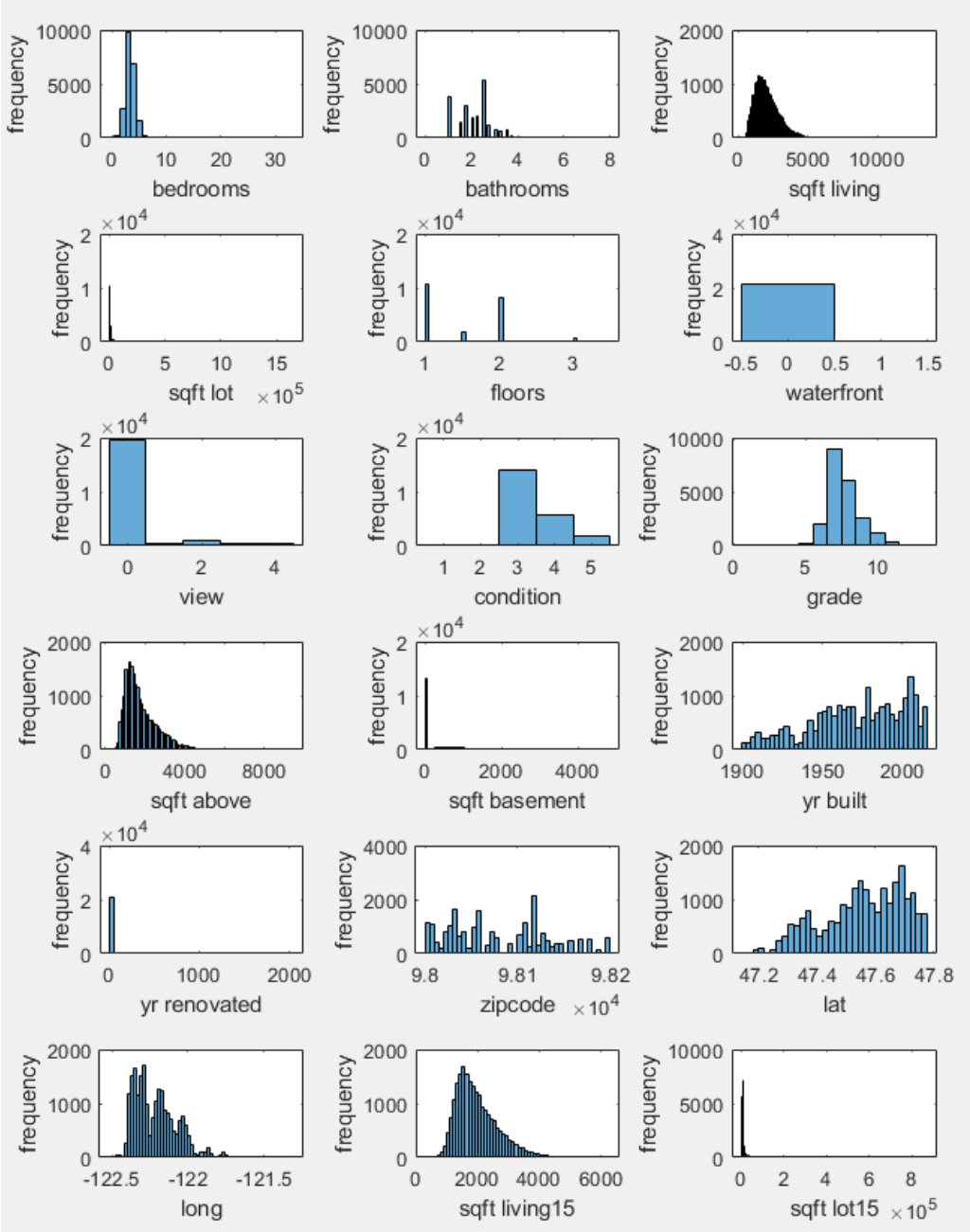
Machine Learning Übung 1

1. Visualisierung des Datasets

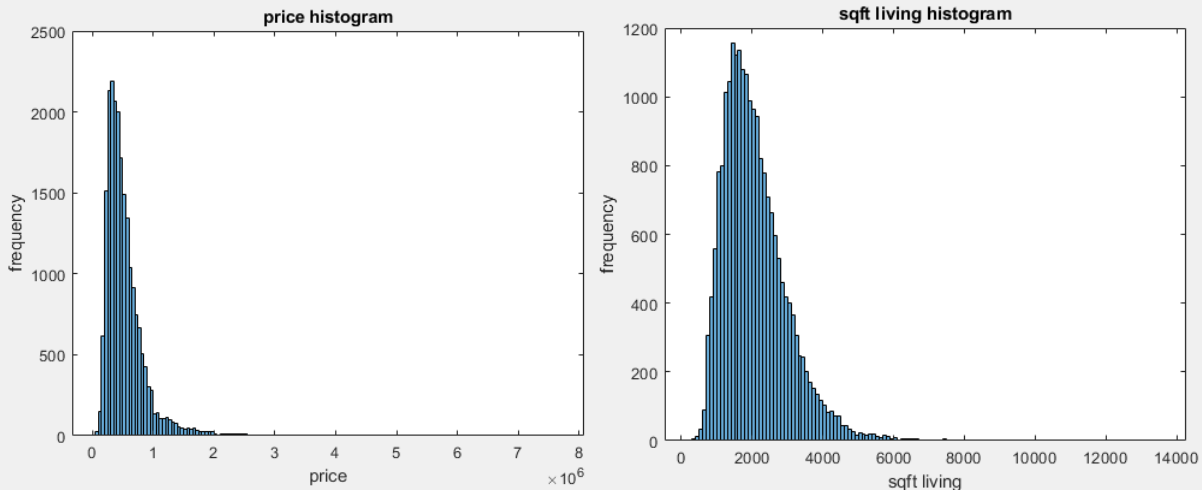
Das Dataset enthält 21’611 Datensätze von Häusern mit 18 verschiedenen Features sowie deren Preis. Nachfolgend eine Gegenüberstellung jedes einzelnen Features mit dem Preis:



Und hier die Histogramme der einzelnen Features:



Wenn wir die Histogramme des Preises sowie des Features «sqft living», welches später für die lineare Regression verwendet wird, genauer betrachten, sehen wir, dass wir es mit einem leicht schiefen Dataset zu tun haben (*skew* data). Die meisten Häuser kosten ca. eine halbe Million und sind um die 2000 sqft gross. Gegen oben gibt es weitaus mehr und extremere Ausreisser, als gegen unten.



1.1 Normalisieren der Features

Die gegebenen Features haben teilweise enorm hohe Werte. Damit die nachfolgend verwendeten Algorithmen effizienter Laufen und potentielle Overflows vermieden werden, ist es sinnvoll, die Werte der Features zu normalisieren – das heisst, den mean auf 0 zu bringen und die Standardabweichung herunter zu skalieren. Der neue Wert der Features wird folgendermassen berechnet:

Normierte Features = (Originalwert – mean) / Standardabweichung

```
function normalizedFeatures = normalizeFeatures(features)
%NORMALIZEFEATURES Scale and features to avoid overflows and speed up
gradient descent
normalizedFeatures = zeros(size(features));
for i = 1:size(features, 2)
    normalizedFeatures(:, i) = (features(:, i) - mean(features(:, i))) /
std(features(:, i));
end
end
```

1.2 Aufsplitten der Daten in Trainings- und Testset

Das Lernmodell auf den gesamten Daten zu trainieren ist nicht sinnvoll. Ich teile die Daten deshalb zufällig auf in Trainings- und Testdaten (80%/20%).

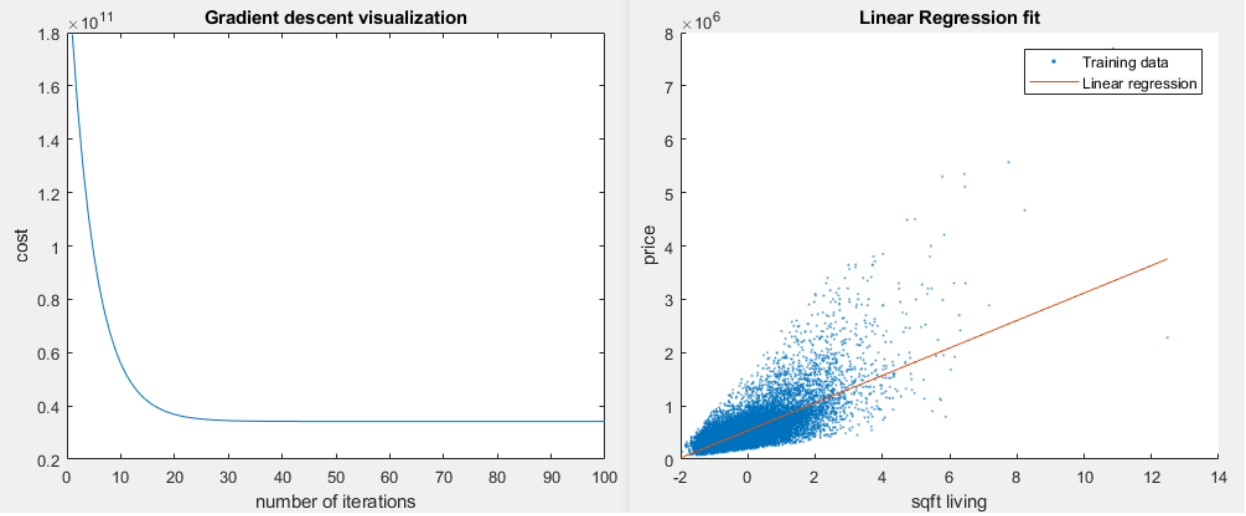
```
[Xtrain, Xtest, ytrain, ytest] = trainTestSplit(normalizedFeatures, labels,
0.8);
```

```
function [Xtrain, Xtest, ytrain, ytest] = trainTestSplit(X, y, trainRatio)
%TRAINTESTSPLIT Split features and labels into random train and test set
%(trainRatio for train part, 1-trainRatio for test part)
m = size(X, 1);
shuffledIndexes = randperm(m);
splitAt = floor(m * trainRatio);
Xtrain = X(shuffledIndexes(1:splitAt), :);
ytrain = y(shuffledIndexes(1:splitAt));
Xtest = X(shuffledIndexes(splitAt+1:end), :);
ytest = y(shuffledIndexes(splitAt+1:end));
end
```

2. Abschätzung des Preises mit linearer Regression

Der Hauspreis wird anhand des Features «sqft living» mit linearer Regression abgeschätzt. Mittels Gradient Descent und einer Learning rate alpha von 0.1 wird die Kostenfunktion minimiert und so eine optimale Gerade durch die Daten gefunden.

In der Visualisierung des Gradient Descent Algorithmus ist zu erkennen, dass die Kostenfunktion konvergiert, also ein Minimum gefunden wurde.

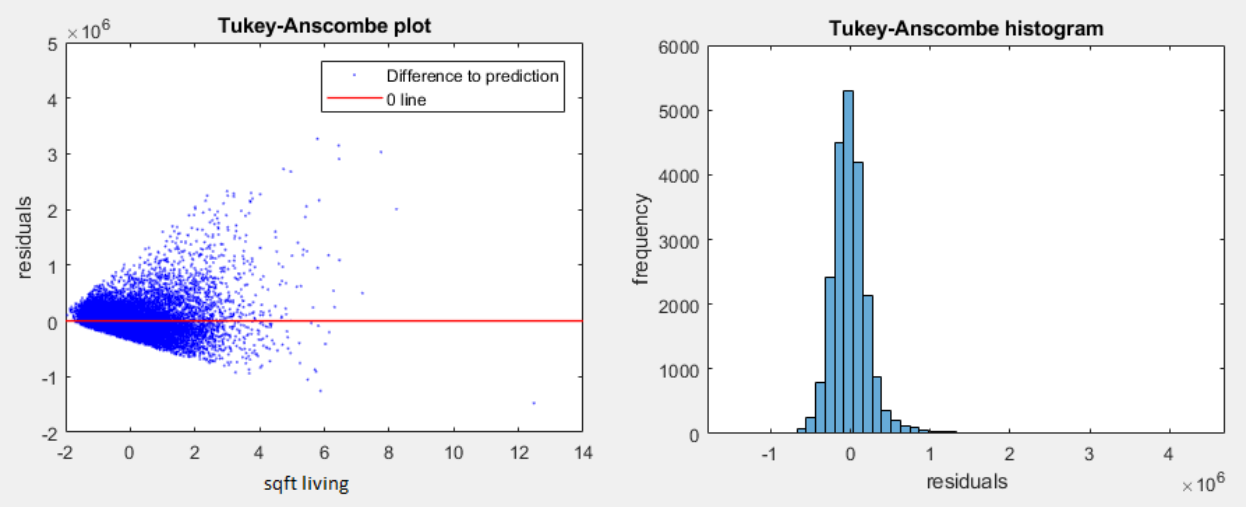


Der Tukey-Anscombe Plot im nachfolgenden Bild zeigt die Streuung der Residuen (also für jeden Wert die Differenz zwischen dem richtigen Wert und dem vorhergesagten Wert). Es ist ersichtlich, dass je höher das Feature «sqft living» wird, desto höher ist die Differenz der Vorhersage. Bei tiefen Werten erzielt das Model sehr gute Vorhersagen. Es existiert also eine nicht-lineare Beziehung der Werte!

Das ist unter anderem auf die *skewness* des Datasets zurückzuführen. Banal ausgedrückt: Es gibt sehr viele «normal» grosse Häuser, aber nur wenige richtig grosse.

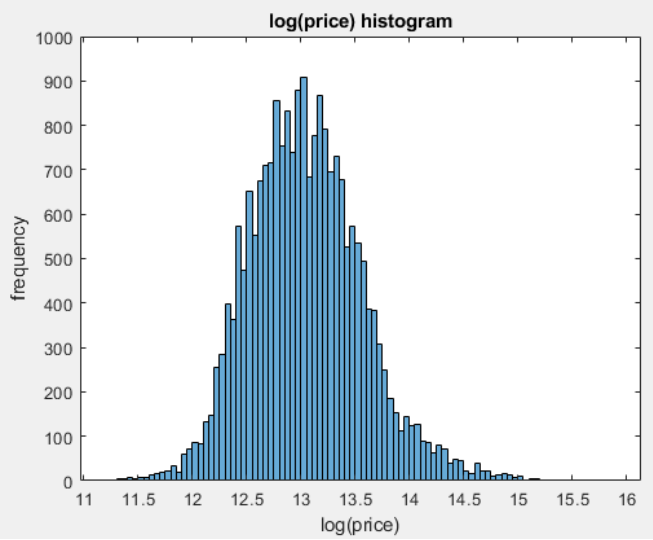
Weiter sieht man, dass die ganz grossen Häuser tendenziell zu tief eingeschätzt werden. Das Histogramm dazu zeigt, dass die Residuen annähernd normalverteilt sind. Die Residuen sind jedoch nicht symmetrisch, Abweichungen gegen oben sind viel häufiger als gegen unten. Daher ist auch der mean positiv.

```
mean=14.345890, std=261456.830212, var=68359674064.758759
```

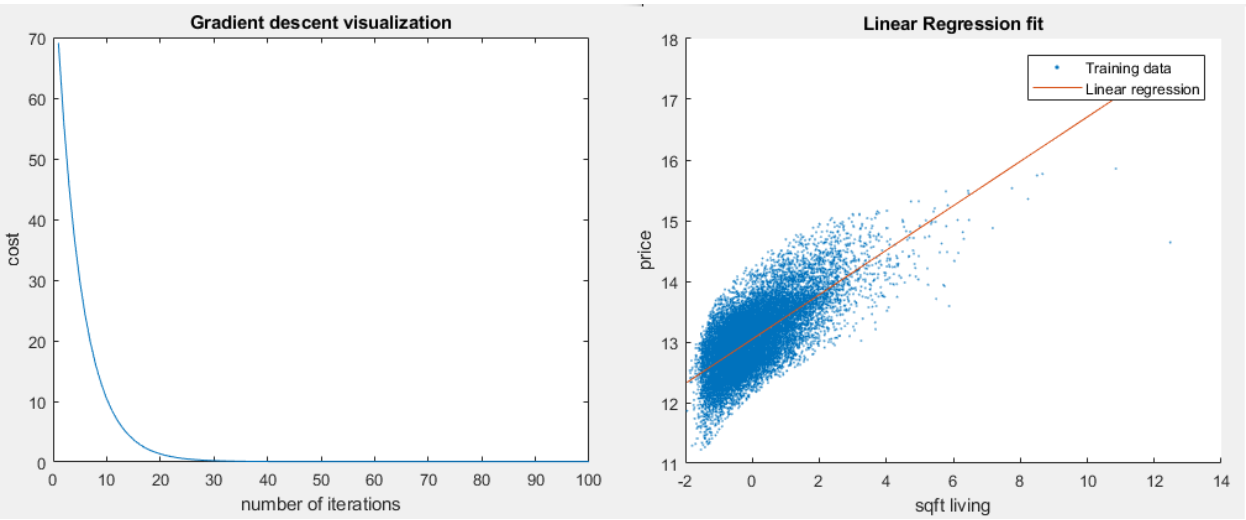


3. Abschätzung des Logarithmus des Preises mit linearer Regression

Um die Schiefheit unseres Datasets loszuwerden, führen wir eine logarithmische Transformation der Output Variable (Preis) durch. Wie dem Histogramm zu entnehmen ist, erreichen wir so eine bessere Verteilung der Output-Variable.



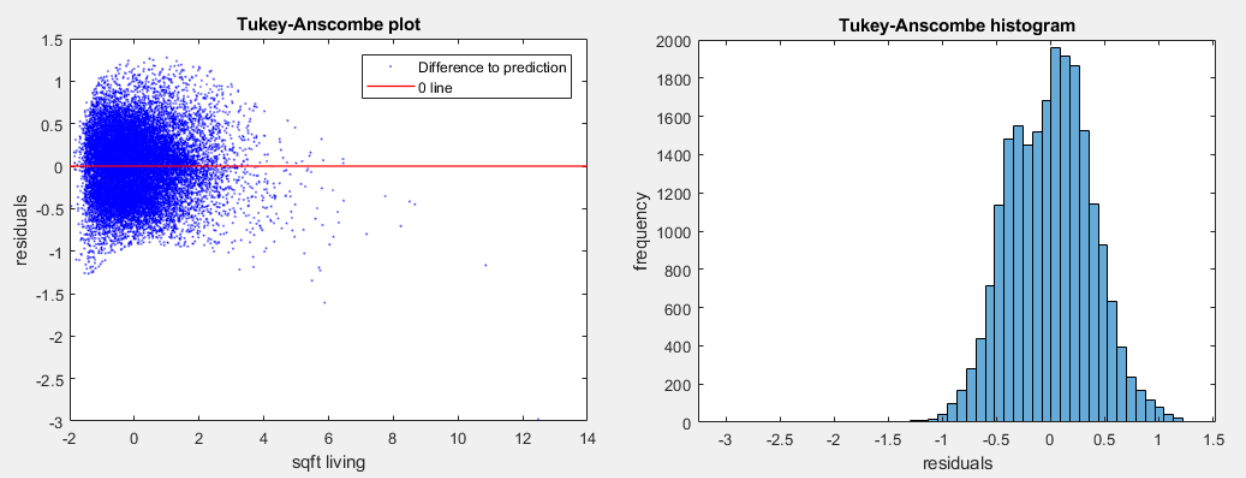
In nachfolgendem Plot ist das Model trainiert auf den Logarithmus des Preises zu sehen.



Interessant ist nun vor allem die Betrachtung der Residuen in der Grafik weiter unten, welche fast normalverteilt sind (abgesehen von einem kleinen Hick im Histogramm).

Im Gegensatz zu vorher ist die Streuung der Residuen sehr symmetrisch, die Abweichungen gegen unten und oben sind sehr ähnlich. Das wiederspiegelt sich auch im mean, welcher fast 0 ist. Das ist natürlich auf die logarithmische Transformation zurückzuführen, wodurch die Ausreisser viel weniger ins Gewicht fallen.

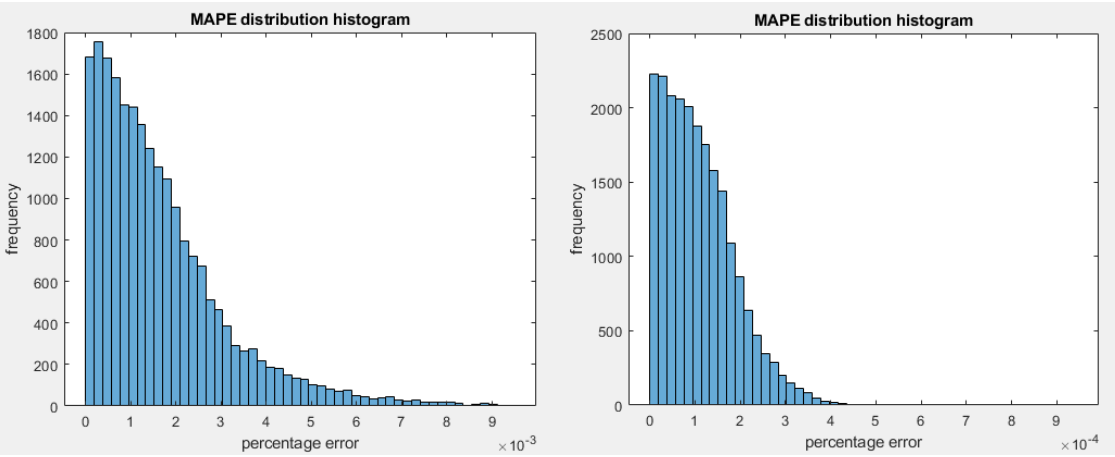
```
Residuals: mean=0.000347, std=0.378526, var=0.143282
```



4. Mean absolute percentage error

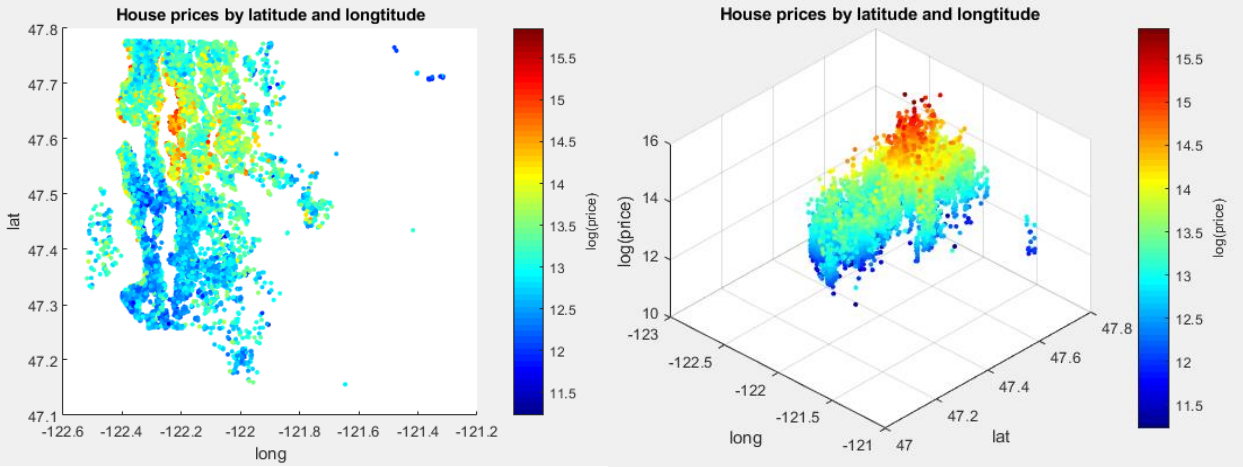
Mit dem MAPE kann das Model evaluiert werden. In den folgenden zwei Histogrammen sieht man, dass wir die meisten Vorhersagen sehr genau treffen können und je höher der Fehler, desto seltener das Vorkommnis.

Links sehen wir das Histogramm mit der untransformierten Output-Variable, rechts mit logarithmischer Transformation. Die MAPE Verteilungen sind sehr ähnlich. Daraus folgt, dass wir durch die logarithmische Transformation keine Genauigkeit unserer Vorhersage verlieren.



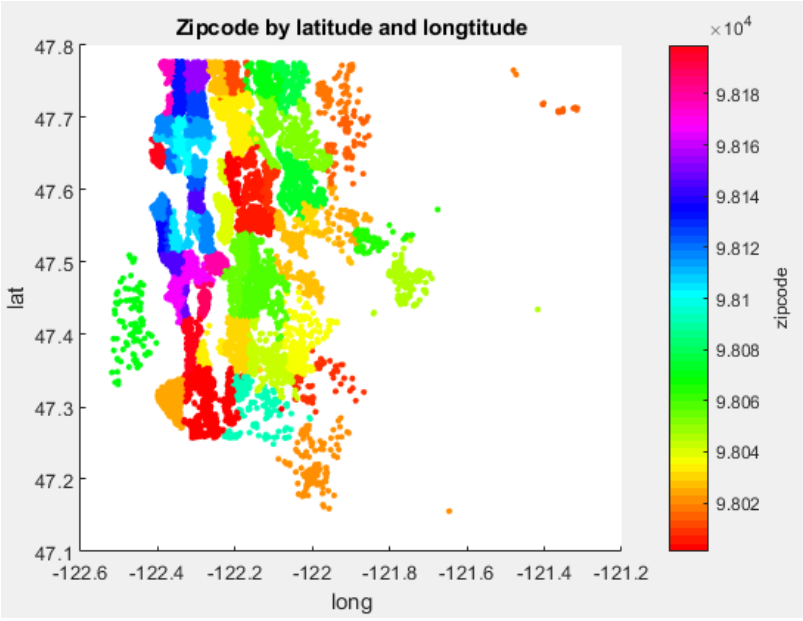
5. Hauspreis nach Längen- und Breitengrad visualisiert

Wenn wir die geografische Landkarte der Hauspreise anschauen, zeichnen sich klare Gebiete ab, in denen die Häuser billig resp. teuer sind. Im Süd-Westen der Karte sind die Häuser tendenziell billiger, während sich die teuersten Häuser mittig der Karte gegen Norden befinden.



6. Einfluss vom Zipcode auf den Hauspreis

Wenn wir die Regionen farblich gemäss Zipcode kennzeichnen und mit der Heatmap der Preise im vorherigen Abschnitt vergleichen, erschliesst sich, dass die Region mit Zipcode 98190 klar teurer ist als zum Beispiel die südlichen Regionen 98160 und 98020. Es lässt sich annehmen, dass es sich bei ersterer Region um eine Stadt, bzw. Innenstadt handelt und die anderen Regionen eher auf dem Land sind. In der Tat bestätigt sich diese Annahme, es handelt sich bei der teuersten Region nämlich um die Grossstadt Seattle.



7. Lineare Regression zusätzlich mit onehot-codiertem Zipcode

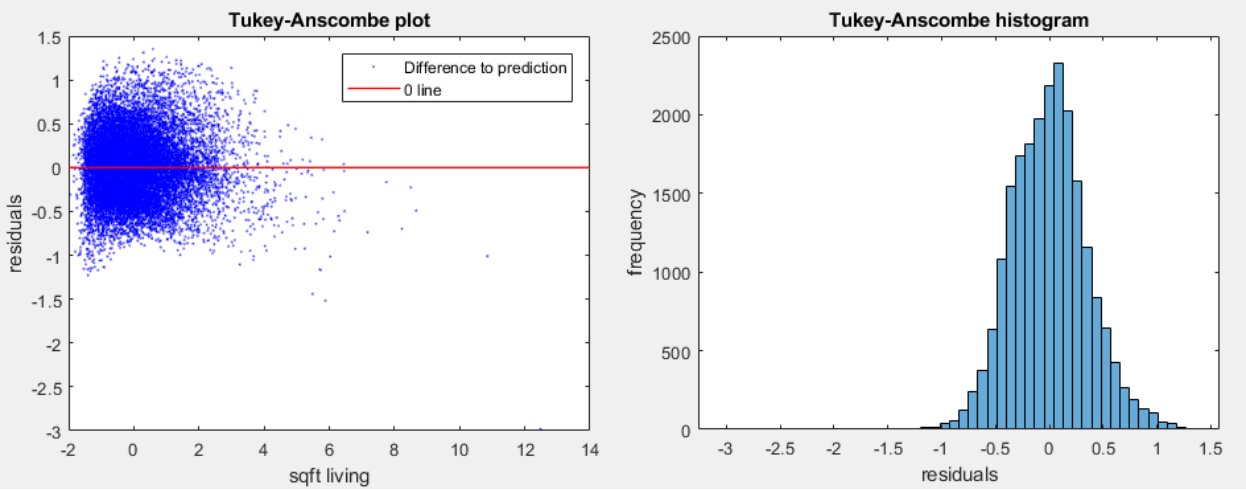
Beim Zipcode ist der numerische Wert im Prinzip irrelevant, Distanzmetriken zwischen Zipcodes sind nicht wirklich aussagekräftig, es handelt sich um ein rein kategorisches Feature. Aus diesem Grund verwenden wir Onehot-encoding.

Das Model arbeitet jetzt also in einem Raum mit viel mehr Dimensionen als vorher (genau genommen 72 – eine bias-Dimension, eine für sqft_living und eine für alle 70 Zipcodes). Das hat keinen Einfluss auf die Funktionsweise der verwendeten Algorithmen, erschwert aber eine Visualisierung der Regression.

Der mean hat sich im Vergleich zu vorher etwa verdoppelt (wobei zu bemerken ist, dass er immer noch sehr klein ist, fast 0, und diese Verdoppelung daher zu verachten ist), die Standardabweichung und Varianz haben sich jedoch leicht verkleinert.

Das Bedeutet, dass jetzt, wo mehr Features im Lern-Algorithmus verwendet werden, die Streuung der Residuen weiter verringert werden konnte.

```
Residuals: mean=0.000653, std=0.349016, var=0.121812
```

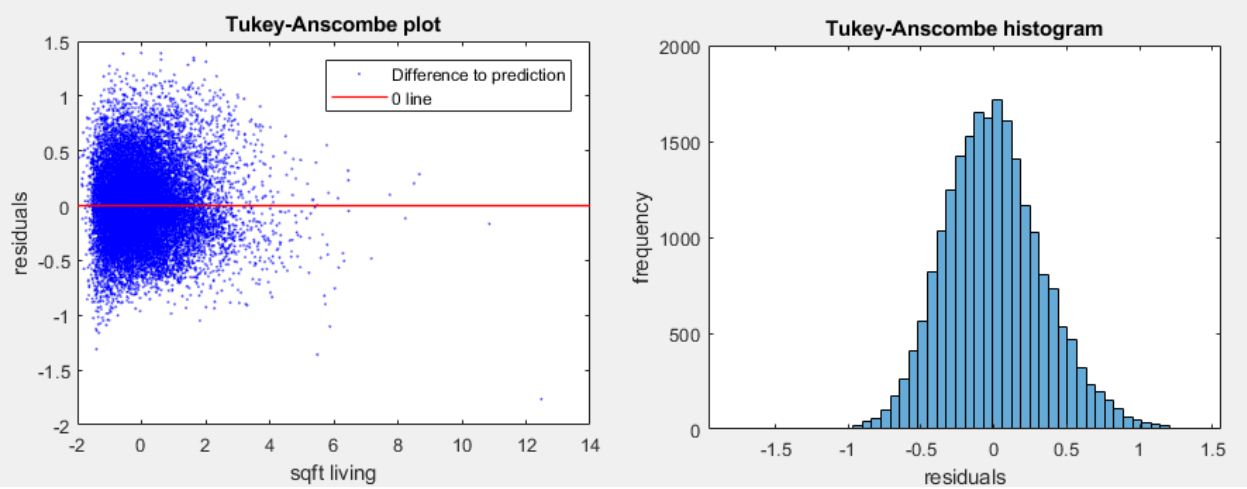


8. Noch mehr Features

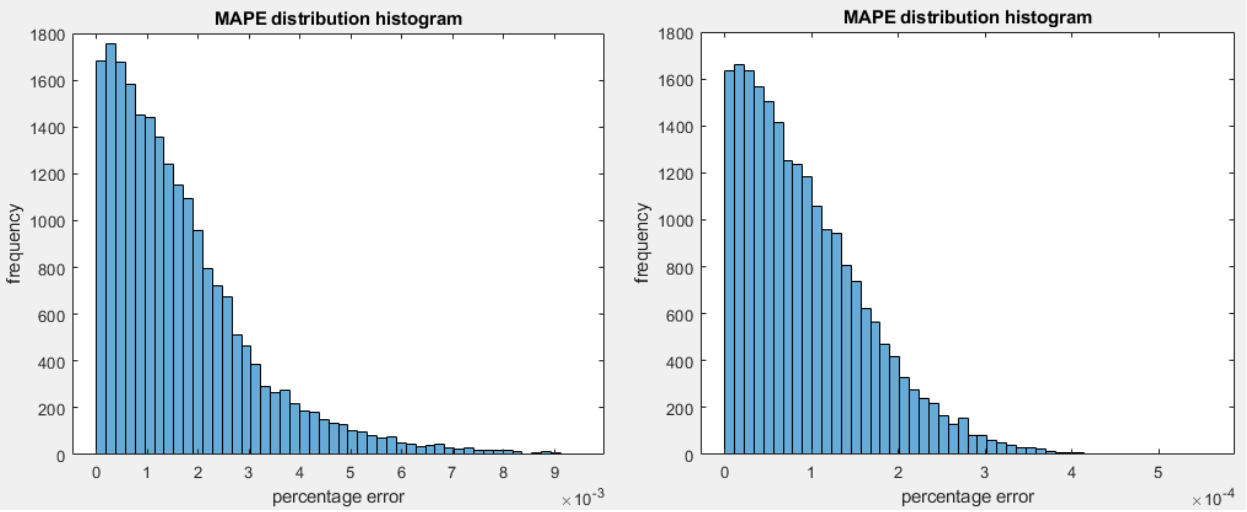
Nun verwenden wir zusätzlich noch Anzahl Bade- und Schlafzimmer, Bewertung des Hauses sowie das Baujahr für das Training des Modells.

Wir können genau das gleiche beobachten wie vorher: Der mean hat sich erneut ungefähr (unwesentlich) verdoppelt während die Standardabweichung und Varianz – und somit die Streuung der Residuen - kleiner wurden. Auch das Histogramm kommt somit noch mehr an eine Normalverteilung heran, wie in den folgenden Grafiken zu sehen ist.

```
Residuals: mean=0.001285, std=0.338334, var=0.114470
```



Auch nach erneuter Evaluation des Modells anhand des MAPE Histogramms ist eine Verbesserung feststellbar, es passieren weniger (grosse) Fehler bei der Vorhersage. Links vorher, rechts nachher.



9. Änderung der Kostenfunktion

Um anstatt der konventionellen Kostenfunktion (Summe der quadrierten Fehler, dividiert durch 2 mal die Anzahl Trainingsdatensätze), muss lediglich die Funktion im Gradient-Descent Algorithmus eingesetzt werden (gelb markiert).

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
%GRADIENTDESCENT Performs gradient descent to learn theta
%   theta = GRADIENTDESCENT(X, y, theta, alpha, num_iters) updates theta by
%   taking num_iters gradient steps with learning rate alpha
m = length(y);
J_history = zeros(num_iters, 1);
for iter = 1:num_iters
    h = theta' * X';
    theta = theta - (alpha * 1/m * ((h' - y)' * X))';
    J_history(iter) = ~~cost~~mape(X, y, theta);
end
end


function J = cost(X, y, theta)
%COST Compute cost for linear regression
%   J = COST(X, y, theta) computes the cost of using theta as the
%   parameter for linear regression to fit the data points in X and y
m = length(y);
h = X * theta;
J = 1/(2*m) * sum((h - y).^2);
end


function [m, dist] = mape(X, y, theta)
%MAPE mean absolute percentage error
actual = y;
predicted = X * theta;
dist = 100 / length(actual) * abs((actual - predicted) ./ actual);
m = sum(dist);
end
```

Wenn man das macht sollte man sich aber im Klaren sein, dass die Fehler nicht mehr quadratisch gewichtet werden und somit Ausreisser viel weniger ins Gewicht fallen, was natürlich (wahrscheinlich unerwünschte) Auswirkungen auf das trainierte Modell hat.

Bei Training mit RSS und Evaluation des Modells mit MAPE kann es Probleme geben für 0 Werte, da MAPE ja durch die Output-Variable dividiert. Zusätzlich resultieren positive Abweichungen potentiell in einem Wert grösser 100%, wobei negative Werte immer kleiner gleich 100% sein werden, was suboptimal ist, da negative und positive Abweichungen so ungleich gewichtet werden.