**Linear Regression**

Two main families of algorithms in Machine Learning are Classification and Regression.

Classification: Answers questions of the fowm yes/no.

Regression: Answers questions of the form “how much”?

Fitting the line:

* Draw a random line. Check how well it fits by counting total error. Repeat.

Reminder of line formula: y = (w1)x + w2, where w1 is the slope and w2 is the y intercept.

If we increase w1 it will increase the slope of the line.

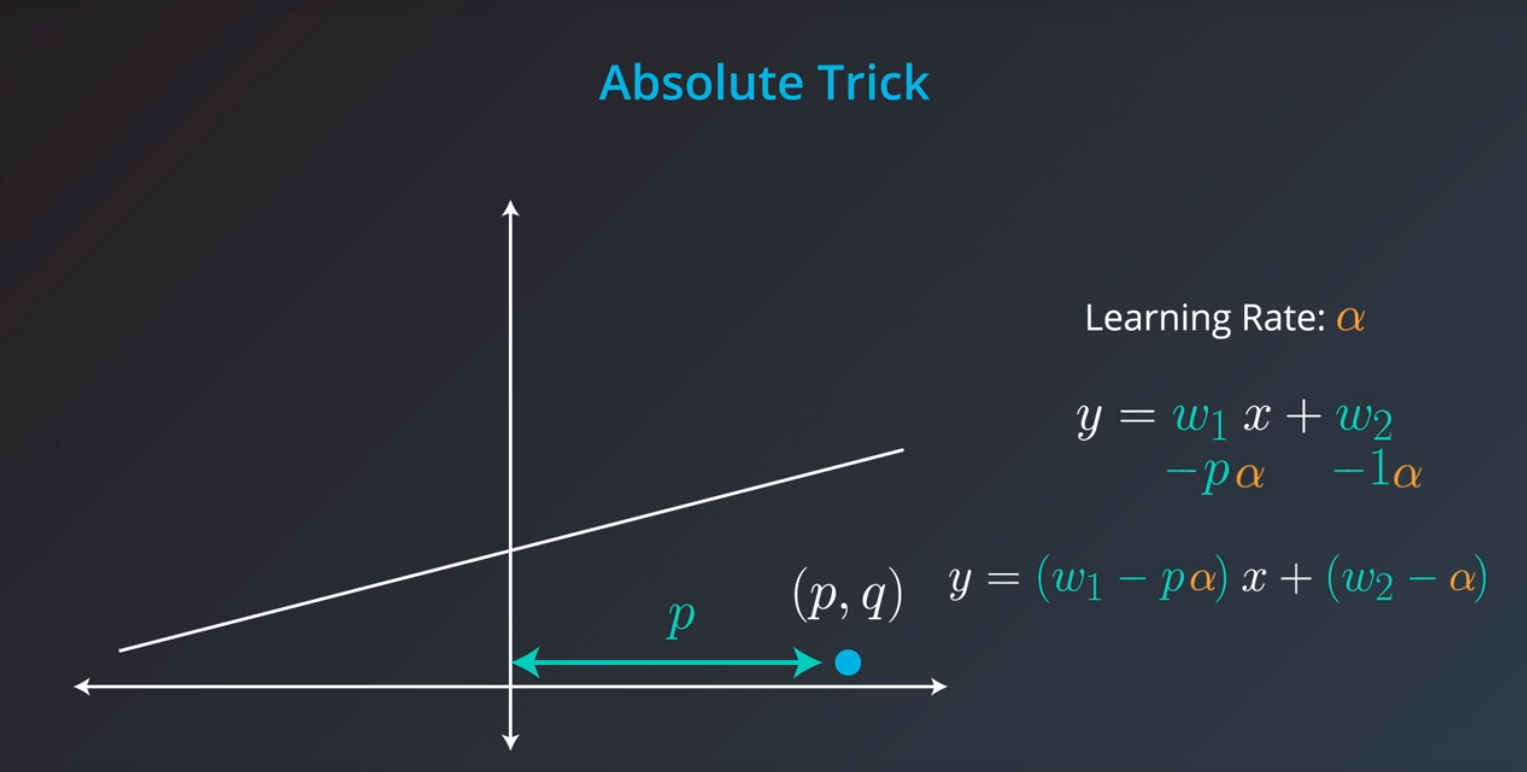
If we decrease w1 it will decrease the slope of the line.

If we increase w2 line moves up.

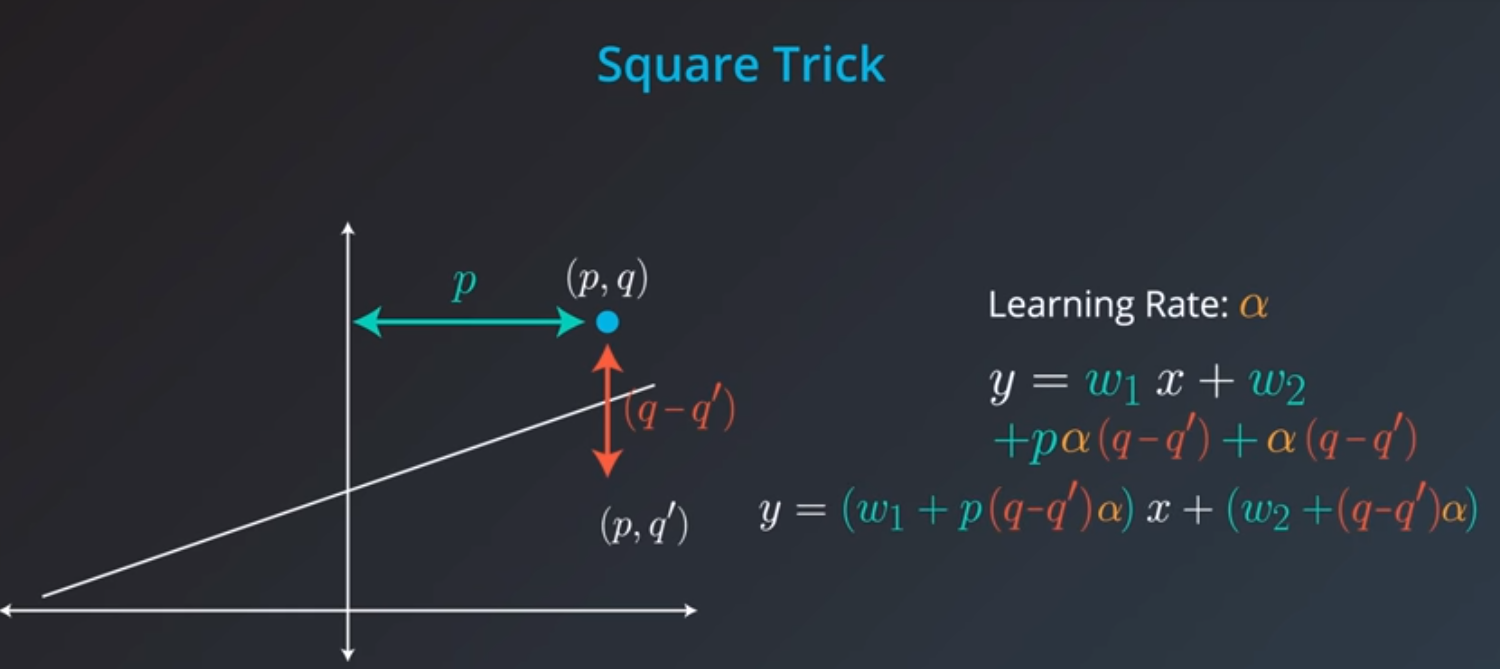
If we decrease w2 line moves down.

Absolute Trick

Trick to move a line closer to a point. Formula is as follows:

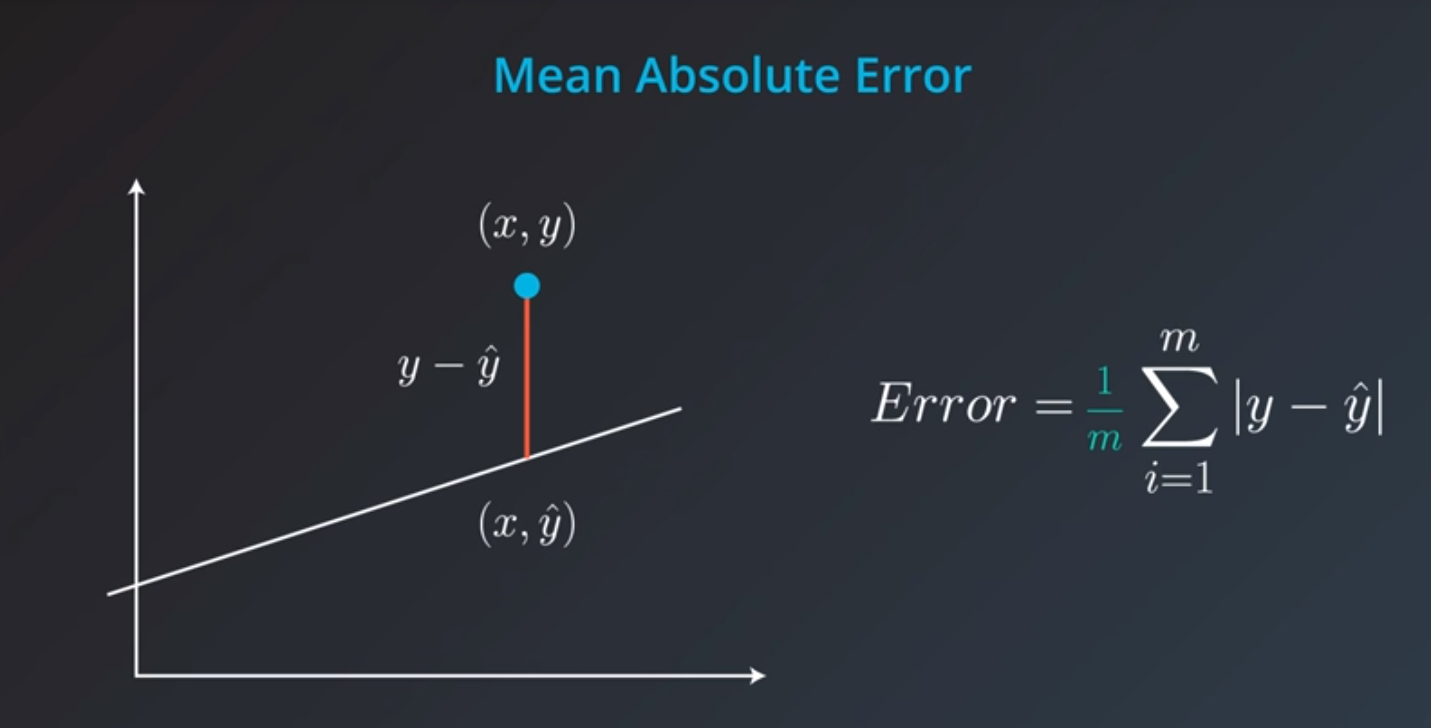


Square Trick

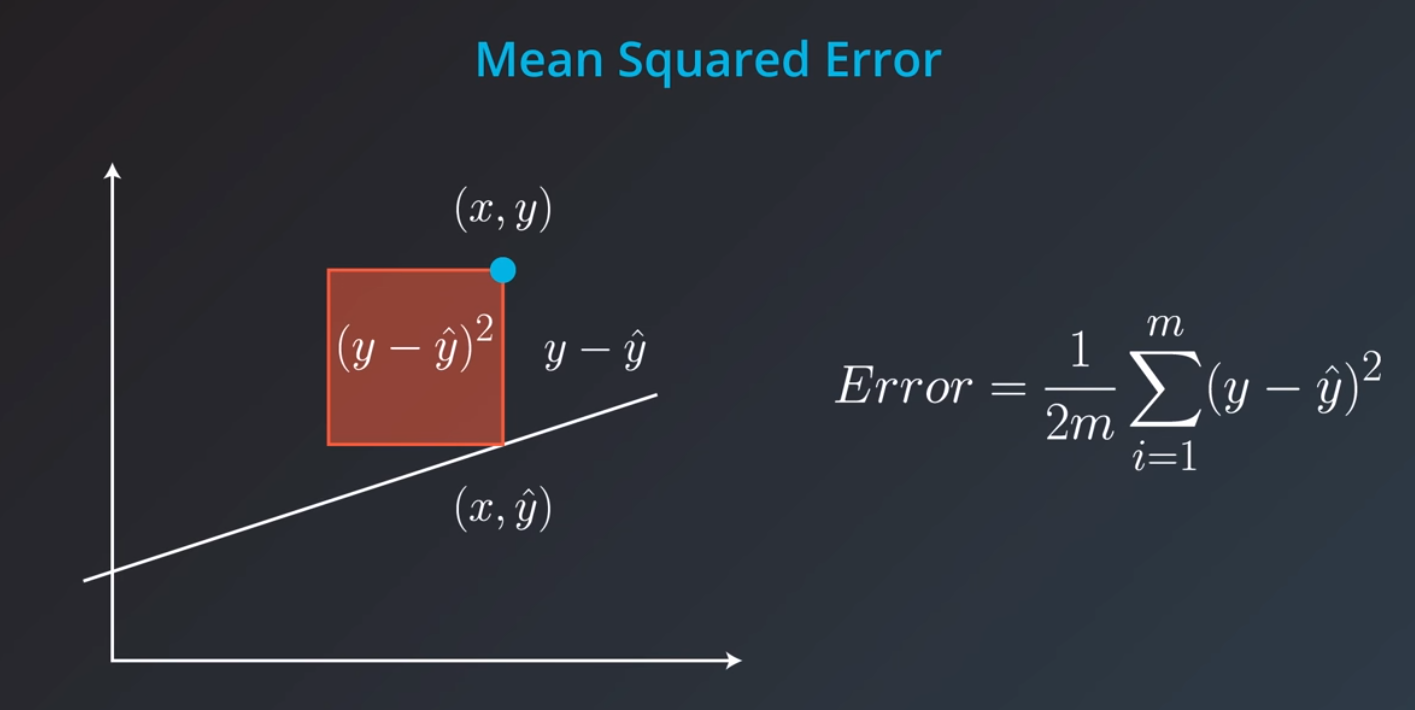


**Error Functions**

Mean Absolute Error

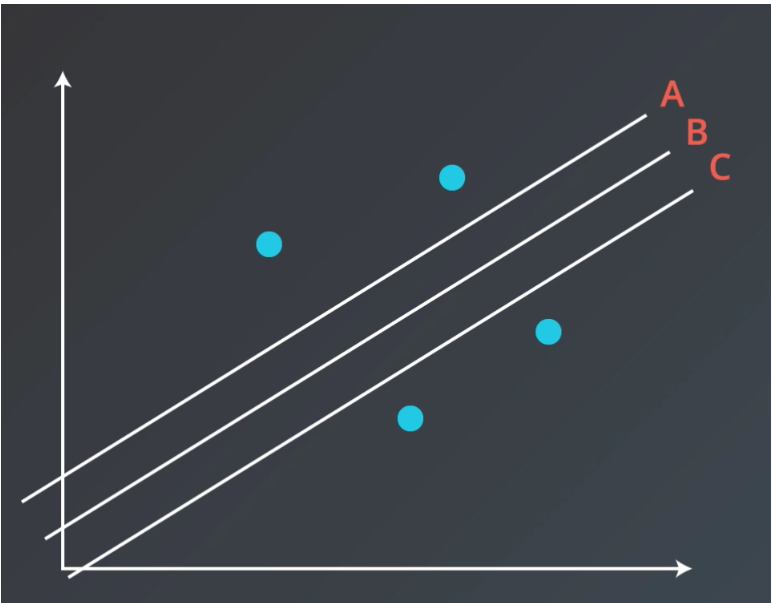


Mean Squared Error

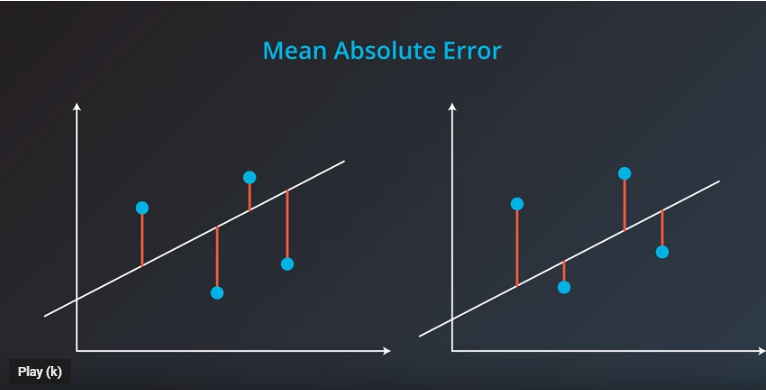


What’s better, mean absolute error or mean squared error?

If we have the following points and 3 lines to see which is the best fit:



For mean absolute error they all get approx. the same value:



As we move the line up and down (without changing slope) we are constantly increasing the value for two of our points and decreasing the value for two of our other points.

However, for Mean Squared Error our answer is B. The reason for this is that Mean Squared Error is a quadratic function and quadratic functions have a minimum for the value in the middle.

**Linear Regression in scikit-learn**

For your linear regression model, you'll be using scikit-learn's [LinearRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html" \t "_blank) class. This class provides the function [fit()](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.fit) to fit the model to your data.

>>> **from** sklearn.linear\_model **import** LinearRegression

>>> model = LinearRegression()

>>> model.fit(x\_values, y\_values)

**Example to create model and then make a prediction on data:**

*Add import statements*

*import pandas as pd*

*from sklearn.linear\_model import LinearRegression*

*# Assign the dataframe to this variable.*

*bmi\_life\_data = pd.read\_csv("bmi\_and\_life\_expectancy.csv")*

*# Make and fit the linear regression model*

*bmi\_life\_model = LinearRegression()*

*bmi\_life\_model.fit(bmi\_life\_data[['BMI']], bmi\_life\_data[['Life expectancy']])*

*# Make a prediction using the model*

*# TODO: Predict life expectancy for a BMI value of 21.07931*

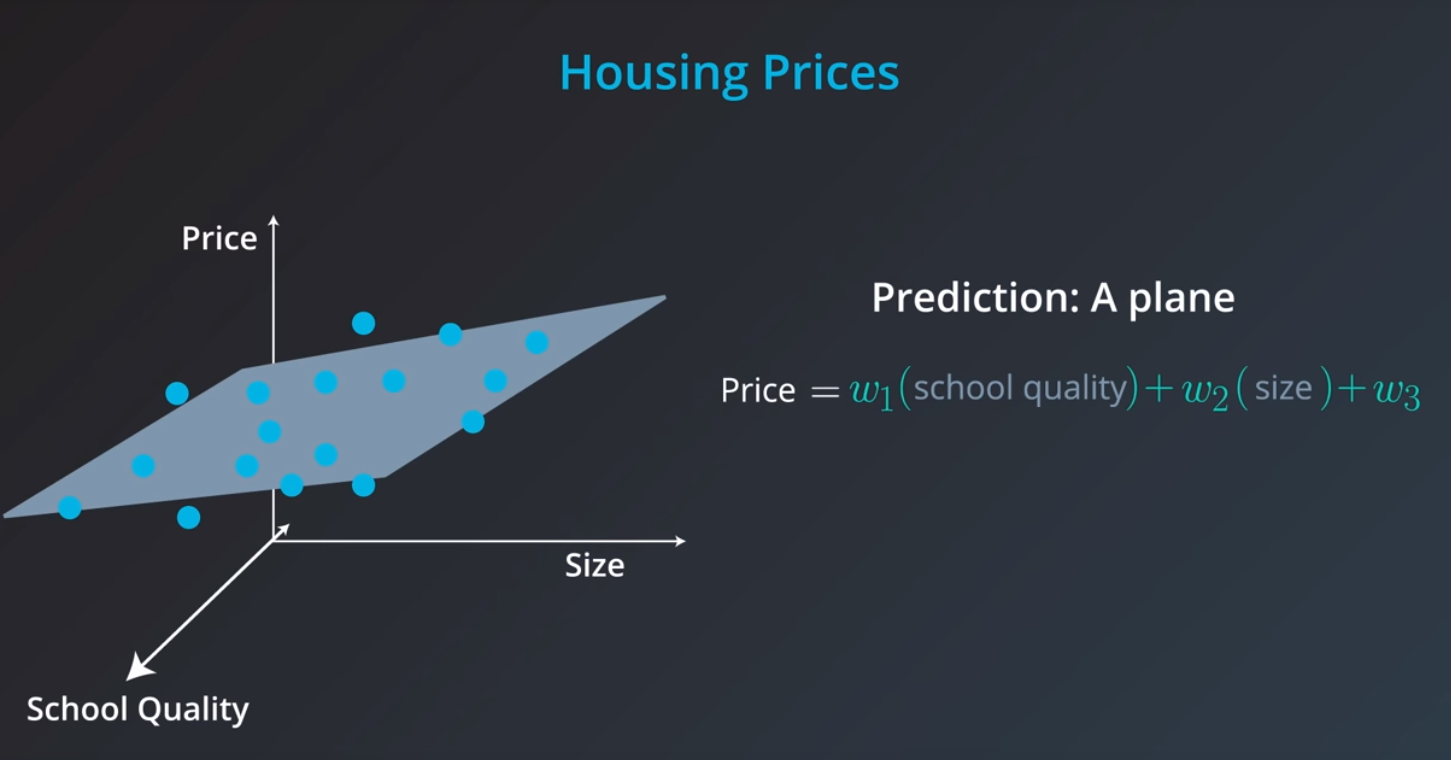
*laos\_life\_exp = bmi\_life\_model.predict(21.07931)*

where “*bmi\_and\_life\_expectancy.csv”* is LinearRegression\_BMIExample which can be found in this folder.

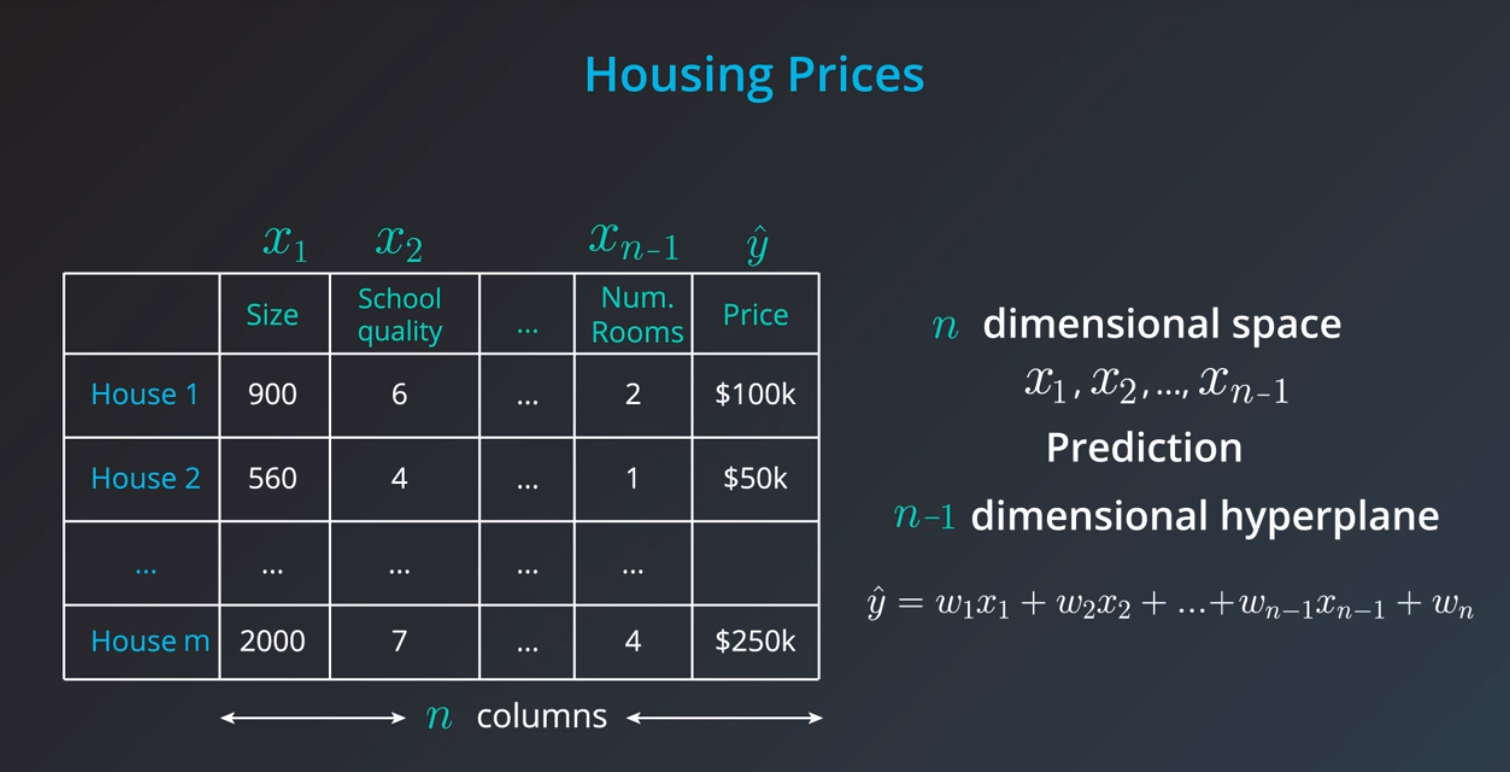
**Higher Dimensions in Linear Regression**

Examples we’ve looked at so far are on the 2-dimenional plane, we can also have linear regression examples in 3 or more dimensions.

3 dimensions:



N dimensions:



We still use the absolute and mean tricks or the absolute and mean error functions for greated then 2 dimensional items.

When coding Linear Regression with many variables we would put all the variables into the “x” variable and the value we want to predict in the “y” variable. Example below:

*from sklearn.linear\_model import LinearRegression*

*from sklearn.datasets import load\_boston*

*# Load the data from the boston house-prices dataset*

*#Note the x variable is a full table of many values*

*boston\_data = load\_boston()*

*x = boston\_data['data']*

*y = boston\_data['target']*

*# Make and fit the linear regression model*

*# TODO: Fit the model and assign it to the model variable*

*model = LinearRegression()*

*model.fit(x,y)*

*# Make a prediction using the model*

*sample\_house = [[2.29690000e-01, 0.00000000e+00, 1.05900000e+01, 0.00000000e+00, 4.89000000e-01,*

*6.32600000e+00, 5.25000000e+01, 4.35490000e+00, 4.00000000e+00, 2.77000000e+02,*

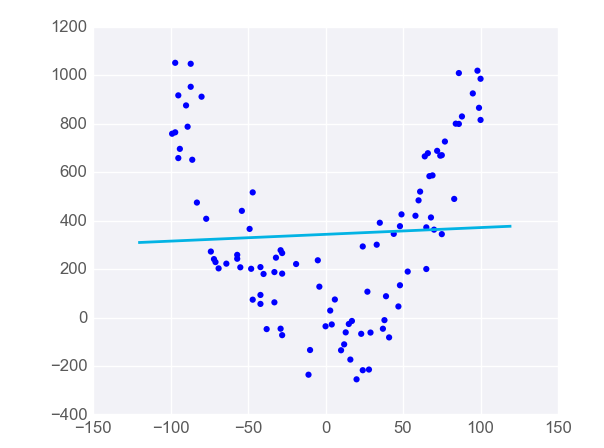
*1.86000000e+01, 3.94870000e+02, 1.09700000e+01]]*

*# TODO: Predict housing price for the sample\_house*

*prediction = model.predict(sample\_house)*

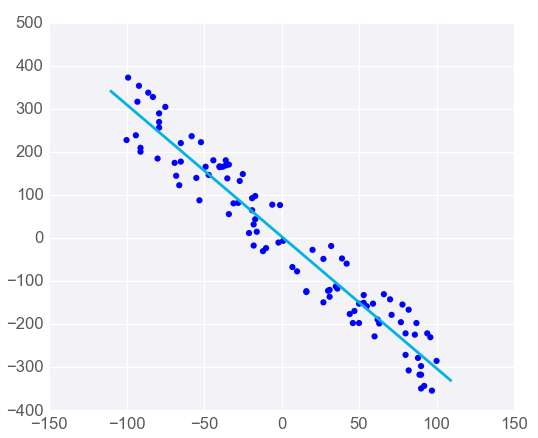
**Issues with Linear Regression**

**Linear Regression Works Best When the Data is Linear**  
Linear regression produces a straight line model from the training data. If the relationship in the training data is not really linear, you'll need to either make adjustments (transform your training data), add features (we'll come to this next), or use another kind of model.

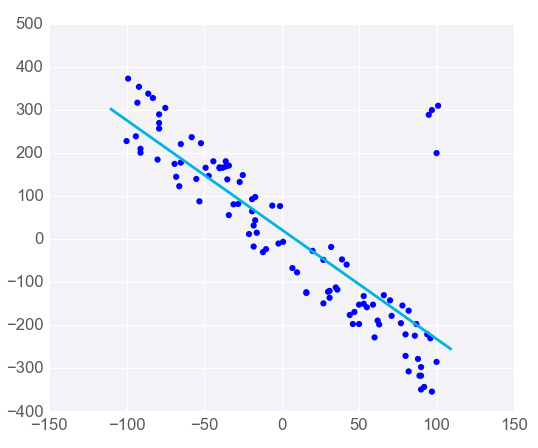


**Linear Regression is Sensitive to Outliers**  
Linear regression tries to find a 'best fit' line among the training data. If your dataset has some outlying extreme values that don't fit a general pattern, they can have a surprisingly large effect.

In this first plot, the model fits the data pretty well.



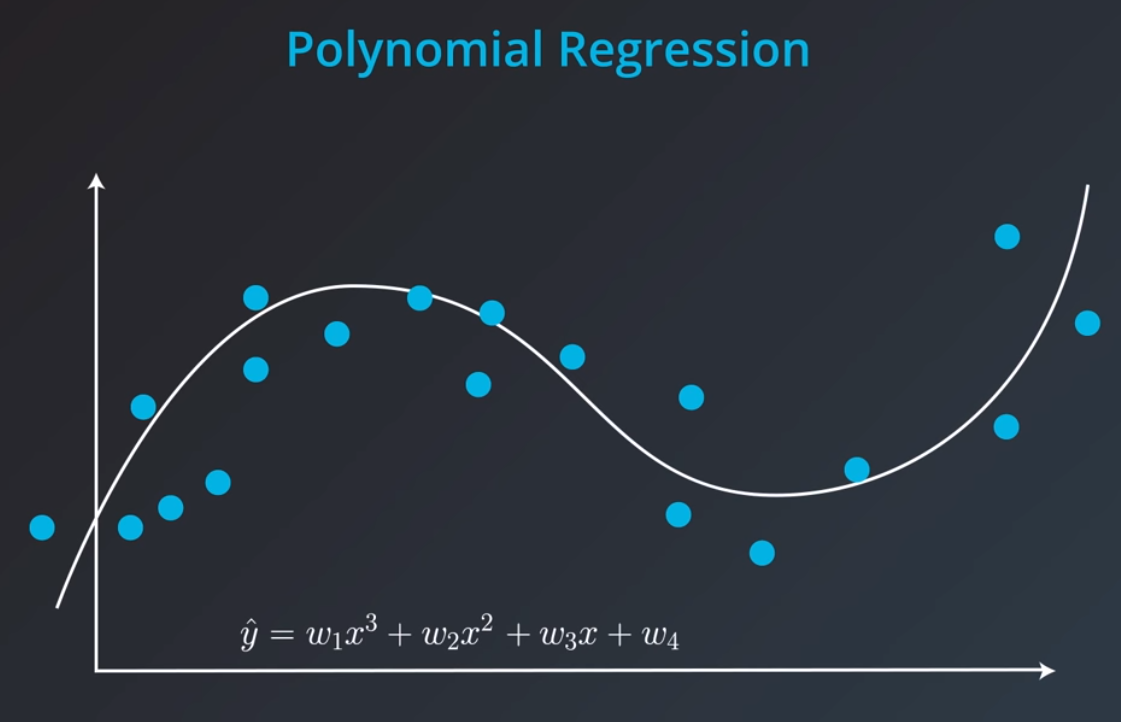
However, adding a few points that are outliers and don't fit the pattern really changes the way the model predicts.



In most circumstances, you'll want a model that fits most of the data most of the time, so watch out for outliers!

**Polynomial Regression**

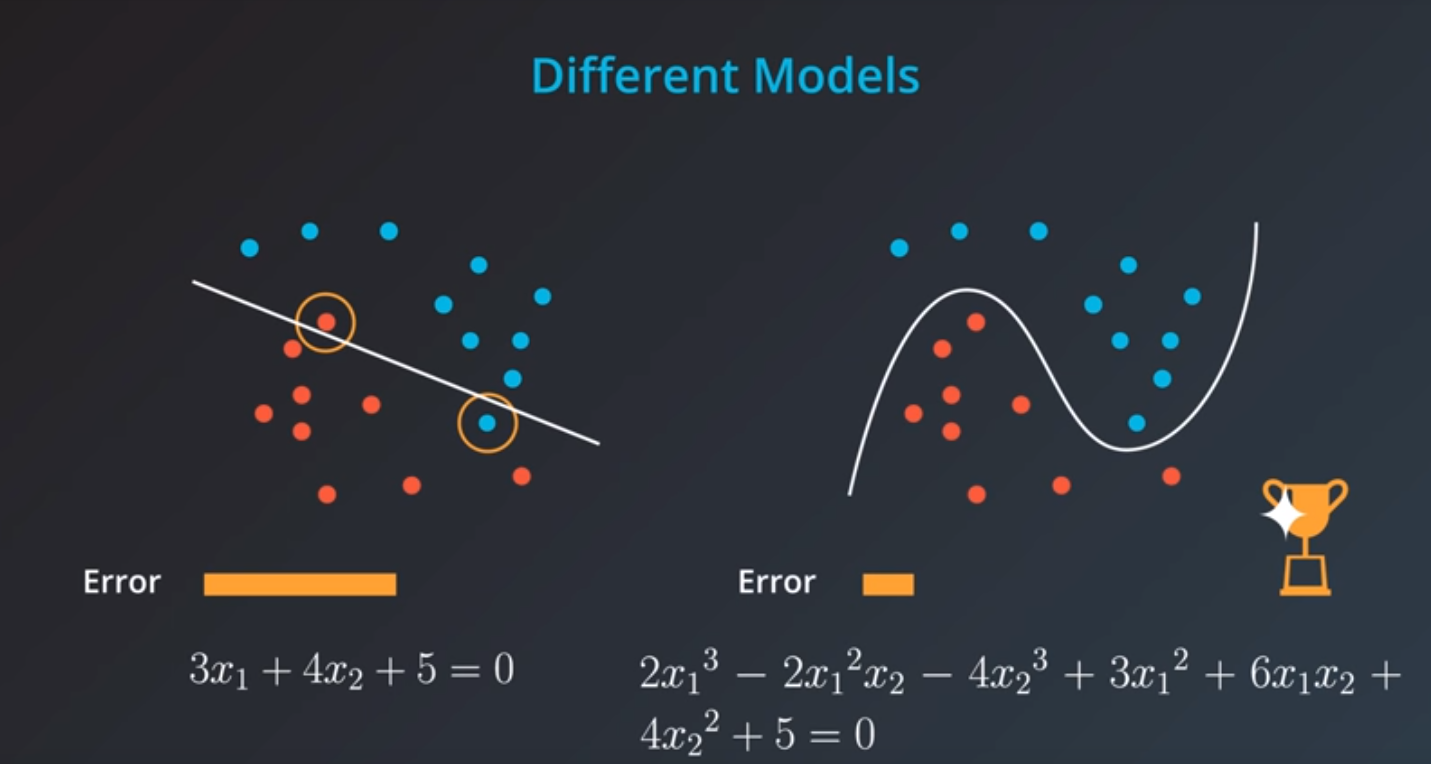
If we have a curved line (non linear) we can still solves using mean absolute or squared error similar to what we’ve done in linear regression. This is known as polynomial regression.



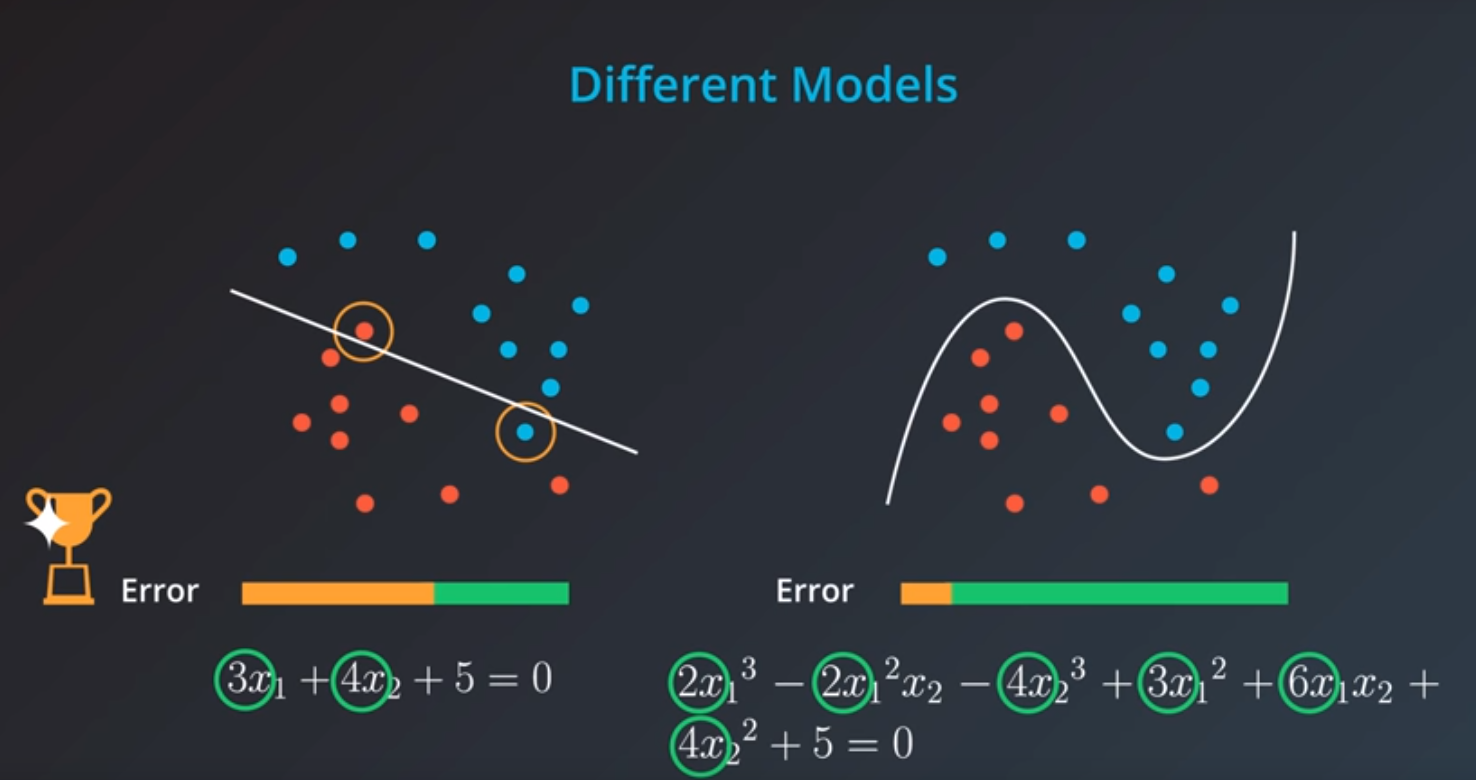
**Regularisation**

Regularisation works for both classification and regression. If we train a model only based off of error we would be likely to overfit a model. We can use a technique called regularisation to help us do this.

If we had two models with the following equations below, we can see that the one on the left has larger error but might generalise better and the one on the right has a small error but may overfit. We will try to regularise these two models.



If we add the coefficients to the error we’d see that the model on the left would have less error:



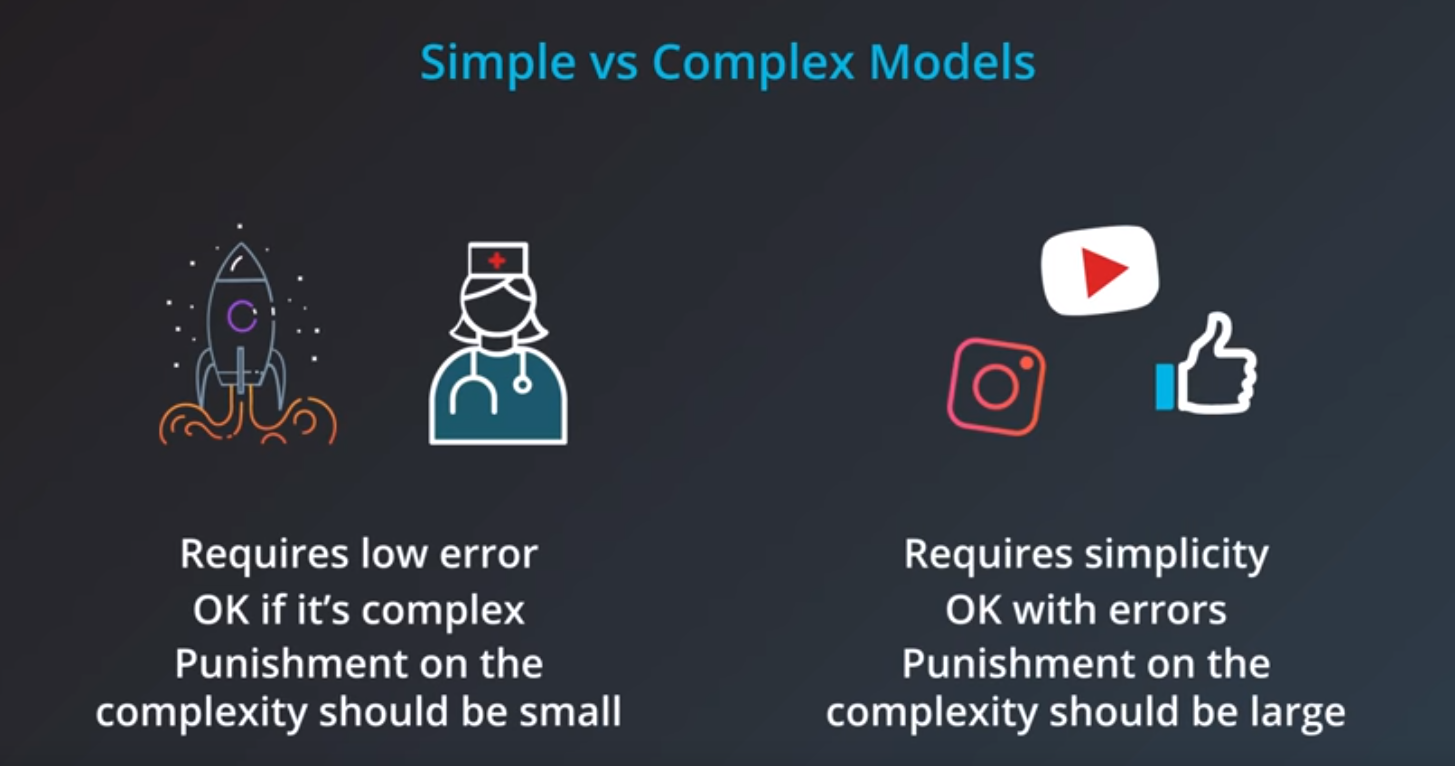
There are two techniques for doing this:

L1 Regularisation and L2 Regularisation.

L1 Regularisation would take the absolute value of the coefficients and add them to the error.

L2 Regularisation would add the squares of the coefficients and add them to the error.

Whether to regularise or not should depend on what kind of model we want.



We can use a parameter lambda to decide how much we should punish complexity. We multiply this lambda by our coefficients part of the error. We’ll use a small lambda if we are ok with high complexity and use a large lambda if we want a more general model.

