**Multiple linear regression**

In most cases, performing a univariate linear regression will not yield a model that is useful for making accurate predictions. In this exercise, you will perform a multiple regression, which uses more than one feature.

You will use price\_log as your target and size\_log and bedrooms as your features. Each of these tensors has been defined and is available. You will also switch from using the the mean squared error loss to the mean absolute error loss: keras.losses.mae(). Finally, the predicted values are computed as follows: params[0] + feature1\*params[1] + feature2\*params[2]. Note that we've defined a vector of parameters, params, as a variable, rather than using three variables. Here, params[0] is the intercept and params[1] and params[2] are the slopes.

**Instructions**

**100 XP**

* Define a linear regression model that returns the predicted values.
* Set loss\_function() to take the parameter vector as an input.
* Use the mean absolute error loss.
* Complete the minimization operation.

# Define the linear regression model

def linear\_regression(params, feature1 = size\_log, feature2 = bedrooms):

return params[0] + feature1\*params[1] + feature2\*params[2]

# Define the loss function

def loss\_function(params, targets = price\_log, feature1 = size\_log, feature2 = bedrooms):

# Set the predicted values

predictions = linear\_regression(params, feature1, feature2)

# Use the mean absolute error loss

return keras.losses.mae(targets, predictions)

# Define the optimize operation

opt = keras.optimizers.Adam()

# Perform minimization and print trainable variables

for j in range(10):

opt.minimize(lambda: loss\_function(params), var\_list=[params])

print\_results(params)

Great job! Note that params[2] tells us how much the price will increase in percentage terms if we add one more bedroom. You could train params[2] and the other model parameters by increasing the number of times we iterate over opt.