```
import numpy as np
import pandas as pd
import math
import os
from matplotlib import pyplot as plt
import tensorflow as tf
from tensorflow import feature_column
from tensorflow.keras import layers
from tensorflow.keras import regularizers
from sklearn.model_selection import train_test_split


data = pd.read_csv('/content/data.csv') #import housedata
data = data.reindex(np.random.permutation(data.index)) #shuffling the data


#reducing range of data and data skew through log transformation
data["price"] = np.log((data.price + 1)) #the one is added to work around the 0s in the dataset.
#adding one has little signifiance as most price values are in the hundreds of thousands



print(data.corr()) #using the correlationc coefficient (r) between price and other features to determine which features to train the mode
#Any feature with |r| > 0.1 was used
```

```
                     price  bedrooms  bathrooms  sqft_living  sqft_lot    floors  \
price             1.000000  0.070052   0.135704     0.182691  0.027662  0.115600
bedrooms          0.070052  1.000000   0.545920     0.594884  0.068819  0.177895
bathrooms         0.135704  0.545920   1.000000     0.761154  0.107837  0.486428
sqft_living       0.182691  0.594884   0.761154     1.000000  0.210538  0.344850
sqft_lot          0.027662  0.068819   0.107837     0.210538  1.000000  0.003750
floors            0.115600  0.177895   0.486428     0.344850  0.003750  1.000000
waterfront       -0.011130 -0.003483   0.076232     0.117616  0.017241  0.022024
view              0.049657  0.111028   0.211960     0.311009  0.073907  0.031211
condition        -0.009580  0.025080  -0.119994    -0.062826  0.000558 -0.275013
sqft_above        0.161910  0.484705   0.689918     0.876443  0.216455  0.522814
sqft_basement     0.078372  0.334165   0.298020     0.447206  0.034842 -0.255510
yr_built          0.024780  0.142461   0.463498     0.287775  0.050706  0.467481
yr_renovated     -0.023865 -0.061082  -0.215886    -0.122817 -0.022730 -0.233996

                  waterfront      view  condition  sqft_above  sqft_basement  \
price              -0.011130  0.049657  -0.009580    0.161910       0.078372
bedrooms           -0.003483  0.111028   0.025080    0.484705       0.334165
bathrooms           0.076232  0.211960  -0.119994    0.689918       0.298020
sqft_living         0.117616  0.311009  -0.062826    0.876443       0.447206
sqft_lot            0.017241  0.073907   0.000558    0.216455       0.034842
floors              0.022024  0.031211  -0.275013    0.522814      -0.255510
waterfront          1.000000  0.360935   0.000352    0.078911       0.097501
view                0.360935  1.000000   0.063077    0.174327       0.321602
condition           0.000352  0.063077   1.000000   -0.178196       0.200632
sqft_above          0.078911  0.174327  -0.178196    1.000000      -0.038723
sqft_basement       0.097501  0.321602   0.200632   -0.038723       1.000000
yr_built           -0.023563 -0.064465  -0.399698    0.408535      -0.161675
yr_renovated        0.008625  0.022967  -0.186818   -0.160426       0.043125

                  yr_built  yr_renovated
price             0.024780     -0.023865
bedrooms          0.142461     -0.061082
bathrooms         0.463498     -0.215886
sqft_living       0.287775     -0.122817
sqft_lot          0.050706     -0.022730
floors            0.467481     -0.233996
waterfront       -0.023563      0.008625
view             -0.064465      0.022967
condition        -0.399698     -0.186818
sqft_above        0.408535     -0.160426
sqft_basement    -0.161675      0.043125
yr_built          1.000000     -0.321342
yr_renovated     -0.321342      1.000000
```

```
#splitting the data into training and test batches
train = data.iloc[:3910]
test = data.iloc[3910:,:]


#creating a list and a layer to hold all of the features that will be used
featureColumns = []

bedrooms = tf.feature_column.numeric_column("bedrooms")
featureColumns.append(bedrooms)

bathrooms = tf.feature_column.numeric_column("bathrooms")
featureColumns.append(bathrooms)

sqft_living = tf.feature_column.numeric_column("sqft_living")
```

```
featureColumns.append(sqft_living)

floors = tf.feature_column.numeric_column("floors")
featureColumns.append(floors)

waterfront = tf.feature_column.numeric_column("waterfront")
featureColumns.append(waterfront)

view = tf.feature_column.numeric_column("view")
featureColumns.append(view)

condition = tf.feature_column.numeric_column("condition")
featureColumns.append(condition)

featureLayer = tf.keras.layers.DenseFeatures(featureColumns)

#defining the buildModel function
def buildModel(my_learningRate, my_featureLayer):
    model = tf.keras.models.Sequential() #creating a sequential model
    model.add(my_featureLayer) #adding the feature layer
    model.add(tf.keras.layers.Dense(units=60,activation='relu', name="1st_Hidden_Layer",    #adding the first hidden layer. 30 nodes
                                    kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
                                    bias_regularizer=regularizers.L2(1e-4),
                                    activity_regularizer=regularizers.L2(1e-5)))
    model.add(tf.keras.layers.Dense(units=30, activation='relu', name="2nd_Hidden_Layer",    #adding the second hidden layer. 30 nodes
                                    kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
                                    bias_regularizer=regularizers.L2(1e-4),
                                    activity_regularizer=regularizers.L2(1e-5)))
    model.add(tf.keras.layers.Dense(units=1, name='Output_Layer',                           #adding the output layer
                                    kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
                                    bias_regularizer=regularizers.L2(1e-4),
                                    activity_regularizer=regularizers.L2(1e-5)))
    model.compile(optimizer=tf.keras.optimizers.Adam(lr=my_learningRate), #compiling the model and using mean squared error for loss
                loss="mean_squared_error",
                metrics=[tf.keras.metrics.MeanSquaredError()])
    return model


#defining the trainModel function
def trainModel(model, dataset, epochs, label_name, batch_size=None, validationSplit=0.2):

  features = {name:np.array(value) for name, value in dataset.items()} #splitting the dataset into features and label
  label = np.array(features.pop(label_name))
  history = model.fit(x=features, y=label, batch_size=batch_size,
                      epochs=epochs, shuffle=True, validation_split=validationSplit)
  epochs = history.epoch
  hist = pd.DataFrame(history.history) #getting the mse at each epoch
  mse = hist["mean_squared_error"]

  return epochs, mse


#defining the function to plot the graph
#this graph with plot loss (mean squared error) vs. epoch
def plot_the_loss_curve(epochs, mse):
  plt.figure()
  plt.xlabel("Epoch")
  plt.ylabel("Mean Squared Error")
  plt.plot(epochs, mse, label="Loss")
  plt.legend()
  plt.ylim([mse.min()*0.95, mse.max() * 1.05])
  plt.show()


#defining hyperparameters
learning_rate = 0.011
epochs = 250
batch_size = 100
validation_split = .2

#label name
label_name = "price"


myModel = buildModel(learning_rate, featureLayer)

#training the model on the train dataframe
epochs, rmse = trainModel(myModel, train, epochs, label_name, batch_size)
plot_the_loss_curve(epochs, rmse)

#testing the model against the test dataframe
testFeatures = {name:np.array(value) for name, value in test.items()}
```

```
testLabel = np.array(testFeatures.pop(label_name))
myModel.evaluate(x = testFeatures, y = testLabel, batch_size=batch_size)
```