Topic : Neural Network-based Multiple Linear Regression Implementation

Objective for this template:

1. Introduce participants to fundamental concepts of multiple linear regression and the google colab platform
2. Use tensorflow to build a simple sequential neural network regression model that accepts multiple features.
3. Demonstrate the process of inspecting attribute relationships as well as training and evaluating the performance of the model
4. Allow participants to practice adjusting various parameters of the model to improve performance.

Designed By: *Rodolfo C. Raga Jr.*

Step 1 : load the tensorflow library and other helper libaries using the import keyword

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import io

print("Done with library declaration. Current version of Tensorflow is :", tf.__version__)
print("Select dataset to load...")
from google.colab import files
uploaded=files.upload()
```

```
Done with library declaration. Current version of Tensorflow is :
2.7.0
Select dataset to load...

<IPython.core.display.HTML object>

Saving auto-mpg.csv to auto-mpg.csv
```

Step 2 : Load data into a dataframe and visually inspect the headers and values

```
dataset_raw = pd.read_csv(io.BytesIO(uploaded['auto-mpg.csv']))
print("Done with loading data to dataframes...")

dataset_raw.head()
```

```
Done with loading data to dataframes...

    mpg  cylinders  displacement  ...  model year  origin
car name
0  18.0          8         307.0  ...          70       1  chevrolet
chevelle malibu
1  15.0          8         350.0  ...          70       1
buick skylark 320
2  18.0          8         318.0  ...          70       1
plymouth satellite
3  16.0          8         304.0  ...          70       1
amc rebel sst
4  17.0          8         302.0  ...          70       1
ford torino

[5 rows x 9 columns]
```

Step 3 : Preprocess data by removing categorical attributes, removing instances with missing values, and converting origin into a dummy variable.

```
dataset_raw.pop("car name")
#dataset_raw.isna().sum()
#dataset_raw = dataset_raw.dropna()
origin = dataset_raw.pop('origin')
dataset_raw['USA'] = (origin == 1)*1.0
dataset_raw['Europe'] = (origin == 2)*1.0
dataset_raw['Japan'] = (origin == 3)*1.0
dataset_raw.head(10)
dataset_raw.tail(10)
```

```
      mpg  cylinders  displacement  horsepower  ...  model year  USA
Europe  Japan
382  26.0          4         156.0          92  ...          82  1.0
0.0    0.0
383  22.0          6         232.0         112  ...          82  1.0
0.0    0.0
384  32.0          4         144.0          96  ...          82  0.0
0.0    1.0
385  36.0          4         135.0          84  ...          82  1.0
0.0    0.0
386  27.0          4         151.0          90  ...          82  1.0
0.0    0.0
387  27.0          4         140.0          86  ...          82  1.0
0.0    0.0
```

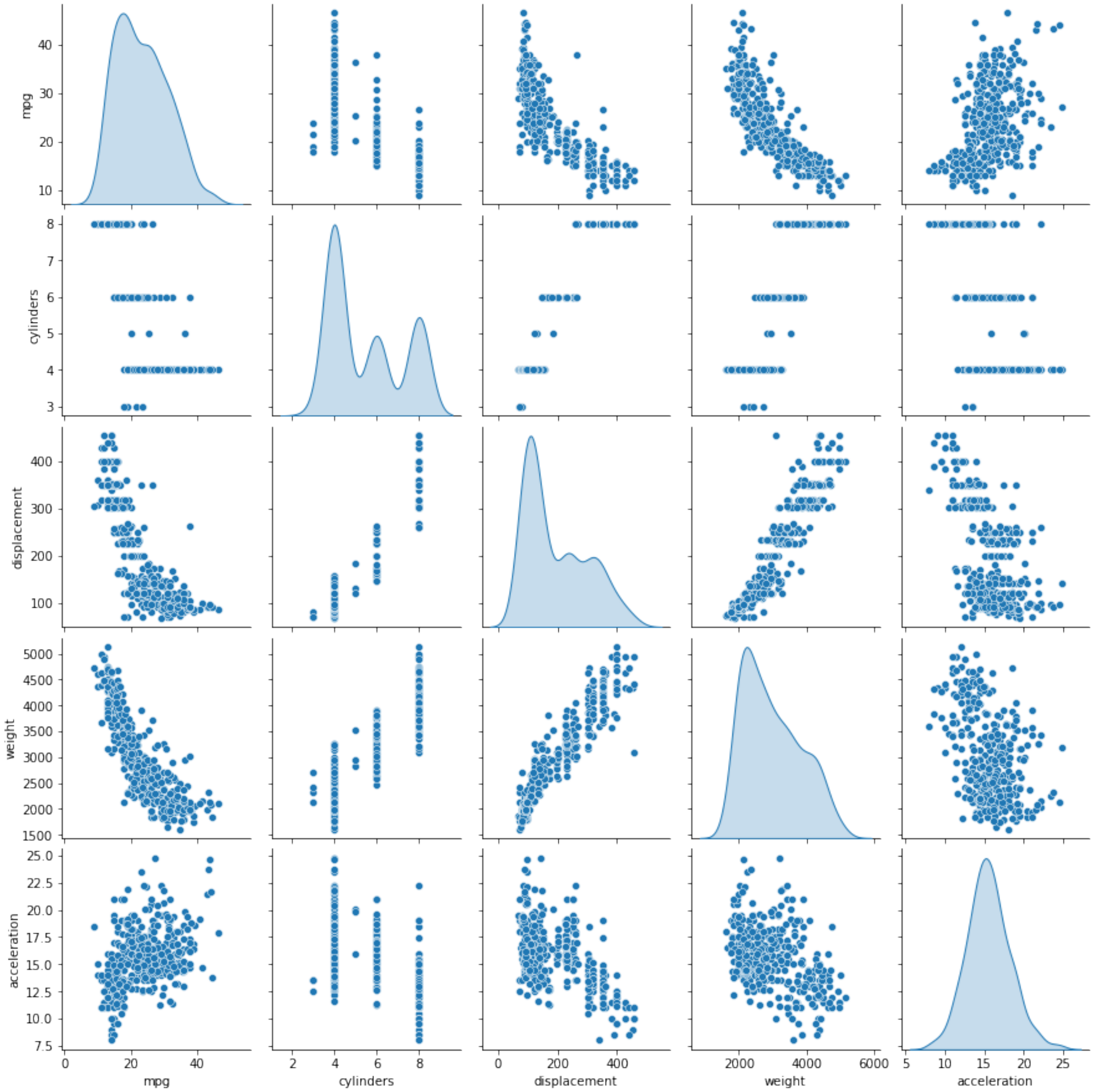
```
388  44.0          4         97.0          52  ...          82  0.0
1.0    0.0
389  32.0          4        135.0          84  ...          82  1.0
0.0    0.0
390  28.0          4        120.0          79  ...          82  1.0
0.0    0.0
391  31.0          4        119.0          82  ...          82  1.0
0.0    0.0

[10 rows x 10 columns]
```

Step 4 : Analyze the relationship between the dependent and independent attributes by visualizing the correlation using scatter charts generated using sns library.

```
sns.pairplot(dataset_raw[["mpg", "cylinders", "displacement",
"weight", "acceleration"]], diag_kind="kde")

train_stats = dataset_raw.describe()
#train_stats.pop("mpg")
#train_stats = train_stats.transpose()
#train_stats
```



Step 5 : Normalize the dataset features and inspect the new representation...

```
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledDS = scaler.fit_transform(dataset_raw)
rescaledDF = pd.DataFrame(rescaledDS)
print(rescaledDF.head())
```

```
          0    1         2         3         4         5    6    7
8    9
0  0.239362  1.0  0.617571  0.456522  0.536150  0.238095  0.0  1.0
0.0  0.0
1  0.159574  1.0  0.728682  0.646739  0.589736  0.208333  0.0  1.0
0.0  0.0
2  0.239362  1.0  0.645995  0.565217  0.516870  0.178571  0.0  1.0
0.0  0.0
3  0.186170  1.0  0.609819  0.565217  0.516019  0.238095  0.0  1.0
0.0  0.0
```

```
4  0.212766  1.0  0.604651  0.510870  0.520556  0.148810  0.0  1.0
0.0  0.0
```

Step 6 : Split the dataset into its predictor and target features. Then separate them into distinct training and testing datasets.

```
predictor_dataset = rescaledDF.iloc[:,1:10]
target_dataset = rescaledDF.iloc[:,0]

X_train, X_test, y_train, y_test =
train_test_split(predictor_dataset,target_dataset, random_state=42,
test_size=0.3)
print("Done with data separation...")

print(y_train.tail())
X_train.tail()
```

```
Done with data separation...
71     0.159574
106    0.239362
270    0.393617
348    0.555851
102    0.053191
Name: 0, dtype: float64
```

```
       1         2         3         4         5         6    7    8
9
71   1.0  0.609819  0.565217  0.646158  0.267857  0.166667  1.0  0.0
0.0
106  0.6  0.423773  0.293478  0.333428  0.416667  0.250000  1.0  0.0
0.0
270  0.2  0.214470  0.211957  0.352141  0.571429  0.666667  1.0  0.0
0.0
348  0.2  0.077519  0.103261  0.217465  0.755952  0.916667  1.0  0.0
0.0
102  1.0  0.857881  0.565217  0.959456  0.357143  0.250000  1.0  0.0
0.0
```

Step 5: We start building the neural network by using the Sequential API to define a Sequential model object named model. This type of model takes a list of layers (Input, Hidden, Output) as arguments and implicitly assumes the order of the calculation from the input layer to the output layer based on the sequence of layer definitions.

```
model = tf.keras.Sequential()
```

Step 6: Then we start defining the layers using the Dense class. Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True)

```
layer_0 = tf.keras.layers.Dense(units=40, activation="relu",
input_shape=[len(X_train.keys())])
layer_1 = tf.keras.layers.Dense(units=40, activation="relu",)
layer_2 = tf.keras.layers.Dense(units=10, activation="relu",)
layer_3 = tf.keras.layers.Dense(units=1)


#layer_0 = tf.keras.layers.Dense(units=40,
input_shape=[len(X_train.keys())])
#layer_1 = tf.keras.layers.Dense(units=40)
#layer_2 = tf.keras.layers.Dense(units=10)
#layer_3 = tf.keras.layers.Dense(units=1)
```

Step 7: After that, we start inserting the dense layers one by one into our sequential model in the order that we want data to flow through them. We can then inspect the overall architecture of the Neural network.

```
#model = tf.keras.Sequential([layer_0,layer_1,layer_2])
model.add(layer_0)
model.add(layer_1)
model.add(layer_2)
model.add(layer_3)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 40)                400       
                                                                 
 dense_1 (Dense)             (None, 40)                1640      
                                                                 
 dense_2 (Dense)             (None, 10)                410       
                                                                 
 dense_3 (Dense)             (None, 1)                 11        
                                                                 
=================================================================
Total params: 2,461
Trainable params: 2,461
Non-trainable params: 0
_________________________________________________________________
```

Step 8: Compile the built model architecture. The compilation configures the learning process by defining an optimizer, a loss function, and other useful training parameters.

```
model.compile(loss='mean_squared_error',
                optimizer=tf.keras.optimizers.Adam(),
                metrics=['mean_absolute_error', 'mean_squared_error'])
print("Done with compile")
```

```
Done with compile
```

Step 9: Train the model by using the set training data

```
trained_model = model.fit(X_train, y_train, epochs=100, verbose=1)
print("Done with model training")
model.summary()
```

```
Epoch 1/100
9/9 [==============================] - 1s 2ms/step - loss: 0.1145 -
mean_absolute_error: 0.2690 - mean_squared_error: 0.1145
Epoch 2/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0598 -
mean_absolute_error: 0.2023 - mean_squared_error: 0.0598
Epoch 3/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0275 -
mean_absolute_error: 0.1337 - mean_squared_error: 0.0275
Epoch 4/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0138 -
mean_absolute_error: 0.0881 - mean_squared_error: 0.0138
Epoch 5/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0111 -
mean_absolute_error: 0.0781 - mean_squared_error: 0.0111
Epoch 6/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0103 -
mean_absolute_error: 0.0766 - mean_squared_error: 0.0103
Epoch 7/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0095 -
mean_absolute_error: 0.0740 - mean_squared_error: 0.0095
Epoch 8/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0087 -
mean_absolute_error: 0.0706 - mean_squared_error: 0.0087
Epoch 9/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0082 -
mean_absolute_error: 0.0681 - mean_squared_error: 0.0082
Epoch 10/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0078 -
mean_absolute_error: 0.0663 - mean_squared_error: 0.0078
Epoch 11/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0076 -
mean_absolute_error: 0.0646 - mean_squared_error: 0.0076
Epoch 12/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0071 -
mean_absolute_error: 0.0627 - mean_squared_error: 0.0071
Epoch 13/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0068 -
mean_absolute_error: 0.0609 - mean_squared_error: 0.0068
Epoch 14/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0065 -
mean_absolute_error: 0.0593 - mean_squared_error: 0.0065
Epoch 15/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0062 -
mean_absolute_error: 0.0577 - mean_squared_error: 0.0062
```

```
Epoch 16/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0060 -
mean_absolute_error: 0.0567 - mean_squared_error: 0.0060
Epoch 17/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0058 -
mean_absolute_error: 0.0552 - mean_squared_error: 0.0058
Epoch 18/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0056 -
mean_absolute_error: 0.0540 - mean_squared_error: 0.0056
Epoch 19/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0055 -
mean_absolute_error: 0.0534 - mean_squared_error: 0.0055
Epoch 20/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0054 -
mean_absolute_error: 0.0529 - mean_squared_error: 0.0054
Epoch 21/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0054 -
mean_absolute_error: 0.0525 - mean_squared_error: 0.0054
Epoch 22/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0053 -
mean_absolute_error: 0.0517 - mean_squared_error: 0.0053
Epoch 23/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0052 -
mean_absolute_error: 0.0515 - mean_squared_error: 0.0052
Epoch 24/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0051 -
mean_absolute_error: 0.0513 - mean_squared_error: 0.0051
Epoch 25/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0052 -
mean_absolute_error: 0.0519 - mean_squared_error: 0.0052
Epoch 26/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0051 -
mean_absolute_error: 0.0514 - mean_squared_error: 0.0051
Epoch 27/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0050 -
mean_absolute_error: 0.0514 - mean_squared_error: 0.0050
Epoch 28/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0049 -
mean_absolute_error: 0.0504 - mean_squared_error: 0.0049
Epoch 29/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0048 -
mean_absolute_error: 0.0498 - mean_squared_error: 0.0048
Epoch 30/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0047 -
mean_absolute_error: 0.0486 - mean_squared_error: 0.0047
Epoch 31/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0047 -
mean_absolute_error: 0.0489 - mean_squared_error: 0.0047
Epoch 32/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0046 -
```

```
mean_absolute_error: 0.0484 - mean_squared_error: 0.0046
Epoch 33/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0046 -
mean_absolute_error: 0.0480 - mean_squared_error: 0.0046
Epoch 34/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0046 -
mean_absolute_error: 0.0477 - mean_squared_error: 0.0046
Epoch 35/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0045 -
mean_absolute_error: 0.0474 - mean_squared_error: 0.0045
Epoch 36/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0044 -
mean_absolute_error: 0.0469 - mean_squared_error: 0.0044
Epoch 37/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0044 -
mean_absolute_error: 0.0476 - mean_squared_error: 0.0044
Epoch 38/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0043 -
mean_absolute_error: 0.0462 - mean_squared_error: 0.0043
Epoch 39/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0043 -
mean_absolute_error: 0.0469 - mean_squared_error: 0.0043
Epoch 40/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0046 -
mean_absolute_error: 0.0498 - mean_squared_error: 0.0046
Epoch 41/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0043 -
mean_absolute_error: 0.0473 - mean_squared_error: 0.0043
Epoch 42/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0043 -
mean_absolute_error: 0.0472 - mean_squared_error: 0.0043
Epoch 43/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0043 -
mean_absolute_error: 0.0467 - mean_squared_error: 0.0043
Epoch 44/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0042 -
mean_absolute_error: 0.0458 - mean_squared_error: 0.0042
Epoch 45/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0042 -
mean_absolute_error: 0.0457 - mean_squared_error: 0.0042
Epoch 46/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0041 -
mean_absolute_error: 0.0456 - mean_squared_error: 0.0041
Epoch 47/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0041 -
mean_absolute_error: 0.0452 - mean_squared_error: 0.0041
Epoch 48/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0043 -
mean_absolute_error: 0.0476 - mean_squared_error: 0.0043
Epoch 49/100
```

```
9/9 [==============================] - 0s 3ms/step - loss: 0.0043 -
mean_absolute_error: 0.0471 - mean_squared_error: 0.0043
Epoch 50/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0040 -
mean_absolute_error: 0.0449 - mean_squared_error: 0.0040
Epoch 51/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0041 -
mean_absolute_error: 0.0451 - mean_squared_error: 0.0041
Epoch 52/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0041 -
mean_absolute_error: 0.0454 - mean_squared_error: 0.0041
Epoch 53/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0046 -
mean_absolute_error: 0.0516 - mean_squared_error: 0.0046
Epoch 54/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0040 -
mean_absolute_error: 0.0454 - mean_squared_error: 0.0040
Epoch 55/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0040 -
mean_absolute_error: 0.0449 - mean_squared_error: 0.0040
Epoch 56/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0039 -
mean_absolute_error: 0.0443 - mean_squared_error: 0.0039
Epoch 57/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0039 -
mean_absolute_error: 0.0437 - mean_squared_error: 0.0039
Epoch 58/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0039 -
mean_absolute_error: 0.0452 - mean_squared_error: 0.0039
Epoch 59/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0040 -
mean_absolute_error: 0.0453 - mean_squared_error: 0.0040
Epoch 60/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0039 -
mean_absolute_error: 0.0446 - mean_squared_error: 0.0039
Epoch 61/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0039 -
mean_absolute_error: 0.0456 - mean_squared_error: 0.0039
Epoch 62/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0039 -
mean_absolute_error: 0.0440 - mean_squared_error: 0.0039
Epoch 63/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0038 -
mean_absolute_error: 0.0438 - mean_squared_error: 0.0038
Epoch 64/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0037 -
mean_absolute_error: 0.0433 - mean_squared_error: 0.0037
Epoch 65/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0037 -
mean_absolute_error: 0.0429 - mean_squared_error: 0.0037
```

```
Epoch 66/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0038 -
mean_absolute_error: 0.0451 - mean_squared_error: 0.0038
Epoch 67/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0039 -
mean_absolute_error: 0.0451 - mean_squared_error: 0.0039
Epoch 68/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0041 -
mean_absolute_error: 0.0465 - mean_squared_error: 0.0041
Epoch 69/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0038 -
mean_absolute_error: 0.0449 - mean_squared_error: 0.0038
Epoch 70/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0037 -
mean_absolute_error: 0.0431 - mean_squared_error: 0.0037
Epoch 71/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0038 -
mean_absolute_error: 0.0440 - mean_squared_error: 0.0038
Epoch 72/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0037 -
mean_absolute_error: 0.0426 - mean_squared_error: 0.0037
Epoch 73/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0037 -
mean_absolute_error: 0.0440 - mean_squared_error: 0.0037
Epoch 74/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0037 -
mean_absolute_error: 0.0429 - mean_squared_error: 0.0037
Epoch 75/100
9/9 [==============================] - 0s 2ms/step - loss: 0.0036 -
mean_absolute_error: 0.0422 - mean_squared_error: 0.0036
Epoch 76/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0036 -
mean_absolute_error: 0.0421 - mean_squared_error: 0.0036
Epoch 77/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0036 -
mean_absolute_error: 0.0421 - mean_squared_error: 0.0036
Epoch 78/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0037 -
mean_absolute_error: 0.0428 - mean_squared_error: 0.0037
Epoch 79/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0036 -
mean_absolute_error: 0.0425 - mean_squared_error: 0.0036
Epoch 80/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0035 -
mean_absolute_error: 0.0419 - mean_squared_error: 0.0035
Epoch 81/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0035 -
mean_absolute_error: 0.0420 - mean_squared_error: 0.0035
Epoch 82/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0036 -
```

```
mean_absolute_error: 0.0418 - mean_squared_error: 0.0036
Epoch 83/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0036 -
mean_absolute_error: 0.0423 - mean_squared_error: 0.0036
Epoch 84/100
9/9 [==============================] - 0s 4ms/step - loss: 0.0034 -
mean_absolute_error: 0.0410 - mean_squared_error: 0.0034
Epoch 85/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0037 -
mean_absolute_error: 0.0442 - mean_squared_error: 0.0037
Epoch 86/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0037 -
mean_absolute_error: 0.0437 - mean_squared_error: 0.0037
Epoch 87/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0037 -
mean_absolute_error: 0.0431 - mean_squared_error: 0.0037
Epoch 88/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0035 -
mean_absolute_error: 0.0424 - mean_squared_error: 0.0035
Epoch 89/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0035 -
mean_absolute_error: 0.0417 - mean_squared_error: 0.0035
Epoch 90/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0035 -
mean_absolute_error: 0.0421 - mean_squared_error: 0.0035
Epoch 91/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0037 -
mean_absolute_error: 0.0430 - mean_squared_error: 0.0037
Epoch 92/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0034 -
mean_absolute_error: 0.0415 - mean_squared_error: 0.0034
Epoch 93/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0035 -
mean_absolute_error: 0.0409 - mean_squared_error: 0.0035
Epoch 94/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0036 -
mean_absolute_error: 0.0429 - mean_squared_error: 0.0036
Epoch 95/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0035 -
mean_absolute_error: 0.0417 - mean_squared_error: 0.0035
Epoch 96/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0034 -
mean_absolute_error: 0.0408 - mean_squared_error: 0.0034
Epoch 97/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0034 -
mean_absolute_error: 0.0414 - mean_squared_error: 0.0034
Epoch 98/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0034 -
mean_absolute_error: 0.0410 - mean_squared_error: 0.0034
Epoch 99/100
```

```
9/9 [==============================] - 0s 3ms/step - loss: 0.0034 -
mean_absolute_error: 0.0409 - mean_squared_error: 0.0034
Epoch 100/100
9/9 [==============================] - 0s 3ms/step - loss: 0.0034 -
mean_absolute_error: 0.0409 - mean_squared_error: 0.0034
Done with model training
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 40)                400

 dense_1 (Dense)             (None, 40)                1640

 dense_2 (Dense)             (None, 10)                410

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 2,461
Trainable params: 2,461
Non-trainable params: 0
_________________________________________________________________
```

Step 10: Test the performance of the model using the testing dataset

```
y_pred = model.predict(X_test)
print(y_pred.size)
print(y_test.size)
print('Actual Values')
print(y_test.values.reshape(1,-1))

print('Predicted Values')
print(y_pred.reshape(1,-1))

score=r2_score(y_test,y_pred)
print("Overall score: {}".format(score*100))
```

```
118
118
Actual Values
[[0.45212766 0.33510638 0.72074468 0.45212766 0.4787234  0.50531915
  0.10638298 0.45212766 0.26595745 0.53191489 0.15957447 0.26595745
  0.21010638 0.53191489 0.19148936 0.50531915 0.29255319 0.62234043
  0.4893617  0.69148936 0.34574468 0.93085106 0.66755319 0.2393617
  0.45212766 0.45212766 0.31914894 0.18617021 0.45212766 0.34574468
  0.15957447 0.27659574 0.2712766  0.60106383 0.10638298 0.7712766
  0.13297872 0.42553191 0.10638298 0.02659574 0.10638298 0.39893617
  0.71808511 0.45212766 0.13297872 0.10638298 0.26595745 0.75265957
  0.31914894 0.53191489 0.13297872 0.39893617 0.50531915 0.71808511
  0.34574468 0.18617021 0.29255319 0.29255319 0.42553191 0.42553191
```

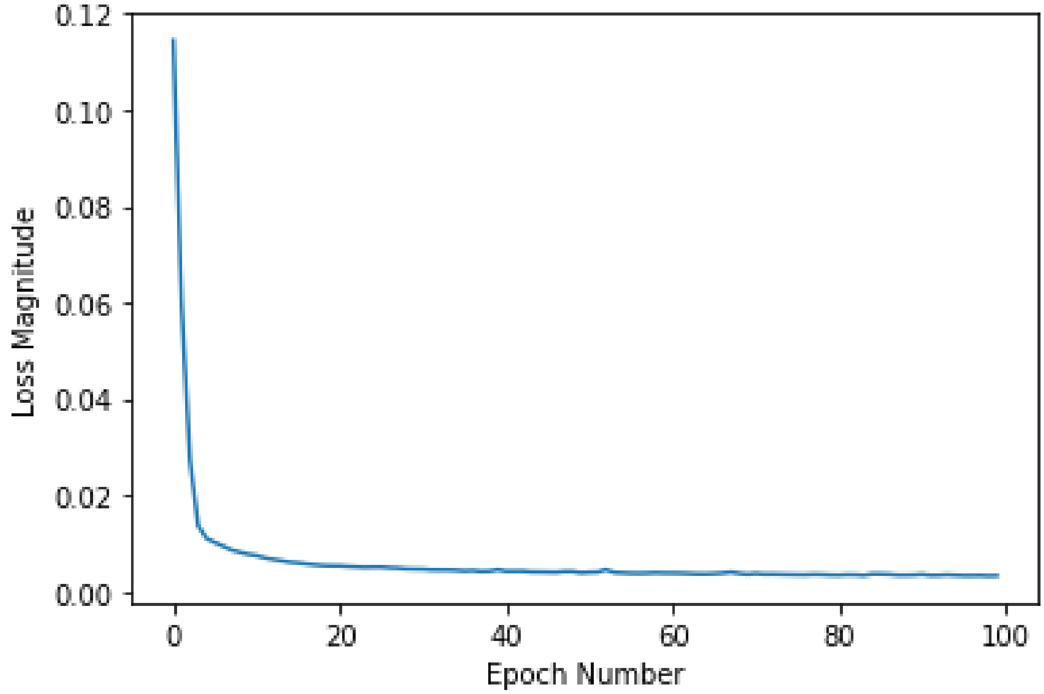
```
  0.07978723 0.35904255 0.47340426 0.37234043 0.32180851 0.42553191
  0.42819149 0.53191489 0.28989362 0.15957447 0.34574468 0.10638298
  0.34574468 0.2393617  0.2287234  0.50531915 0.15957447 0.18617021
  0.4787234  0.25531915 0.42553191 0.2393617  0.15957447 0.79787234
  0.17287234 0.55319149 0.42553191 0.33244681 0.21276596 0.18617021
  0.3537234  0.57712766 0.55851064 0.58510638 0.2712766  0.19946809
  0.2393617  0.10638298 0.49202128 0.2393617  0.05319149 0.21276596
  0.13297872 0.45212766 0.67021277 0.67553191 0.69414894 0.29255319
  0.34574468 0.24468085 0.61170213 0.60904255 0.31914894 0.34574468
  0.75       0.6356383  0.27659574 0.15957447]]
Predicted Values
[[0.45185477 0.31447068 0.70359635 0.38632882 0.52516973 0.53350514
  0.10617682 0.55659497 0.2785139  0.57102954 0.17591478 0.40484783
  0.19762732 0.49693403 0.1742858  0.51266676 0.2903688  0.65917915
  0.526367   0.5458922  0.30015907 0.99499565 0.7476126  0.21293078
  0.5407005  0.42373666 0.32915878 0.22921729 0.55704534 0.40795314
  0.14705047 0.31033397 0.2798928  0.62557936 0.12191872 0.78542984
  0.11988422 0.4180373  0.1101426  0.04750061 0.1500561  0.48822305
  0.78329265 0.45738372 0.148339   0.11336804 0.22344297 0.6715518
  0.33778328 0.57244253 0.12104269 0.40730923 0.3629153  0.7184501
  0.43352053 0.16855295 0.22570895 0.28650147 0.4102977  0.41954646
  0.10590013 0.27552703 0.40990886 0.33449316 0.44502592 0.46465394
  0.46038768 0.5446213  0.29448807 0.1411425  0.30939743 0.10457183
  0.46844804 0.4007345  0.40251794 0.44118285 0.13854586 0.18872681
  0.4848871  0.24460995 0.4429333  0.3460624  0.12927012 0.71615887
  0.15401335 0.76691705 0.4275907  0.32127678 0.24031755 0.20306452
  0.4189331  0.63458717 0.62477785 0.5770402  0.2487469  0.1691779
  0.3255543  0.096951   0.4372753  0.302514   0.10707144 0.1129647
  0.16108936 0.50292194 0.64584756 0.6898084  0.7884861  0.39974877
  0.33676916 0.25021785 0.79215634 0.67526364 0.3816135  0.37981635
  0.71767473 0.5603364  0.20924012 0.25778356]]
Overall score: 89.36987823437983
```

Other things we can do:

1. Analyze training statistics

```
plt.xlabel('Epoch Number')
plt.ylabel("Loss Magnitude")
plt.plot(trained_model.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7fc7b10fe650>]
```



Build the deployment interface for the prediction model below: