**LAB 6 ( LSTM VS RNN)**

*Use the same 2 datasets used for RNN experiment  
short sequences and long sequences  
  
Apply LSTM - many to one   
vary config of LSTM  
 write results of both RNN and LSTM in 2 tables for 2 datasets*

1. **Dataset Overview**: We’ll assume the two datasets you used for your RNN experiments are already prepared — one for short sequences and one for long sequences. The sequences will be used as inputs to the LSTM model.
2. **LSTM Model Setup**: A Many-to-One configuration will be used. In this setup, the entire sequence is processed, and the model will output a single result (e.g., a classification or regression value) at the last time step.
3. **Vary LSTM Configurations**: You can vary several hyperparameters:
   * Number of LSTM layers (e.g., 1 layer vs. 2 layers)
   * Number of units in each LSTM layer (e.g., 50, 100, 200)
   * Dropout (e.g., 0.2, 0.5)
   * Optimizer (e.g., Adam, RMSprop)
   * Batch size (e.g., 32, 64)
   * Learning rate
4. **Train and Evaluate**:
   * For each configuration, train the LSTM model on both the short and long sequence datasets.
   * Evaluate performance metrics such as accuracy (for classification tasks) or loss (for regression tasks).
5. **Results Table Format**:
   * Create two tables: one for the short sequence dataset and another for the long sequence dataset.
   * Compare the results for both RNN and LSTM models.

CODE:

In [ ]:

*#RNN ----------->Google\_Stock\_Price\_Train.csv(dataset)*

In [ ]:

*# Importing the libraries*

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

**import** warnings

warnings.filterwarnings('ignore')

In [2]:

*# Importing the training set*

dataset\_train **=** pd.read\_csv('Google\_Stock\_Price\_Train.csv')

In [3]:

dataset\_train.head()

Out[3]:

|  | **Date** | **Open** | **High** | **Low** | **Close** | **Volume** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 1/3/2012 | 325.25 | 332.83 | 324.97 | 663.59 | 7,380,500 |
| **1** | 1/4/2012 | 331.27 | 333.87 | 329.08 | 666.45 | 5,749,400 |
| **2** | 1/5/2012 | 329.83 | 330.75 | 326.89 | 657.21 | 6,590,300 |
| **3** | 1/6/2012 | 328.34 | 328.77 | 323.68 | 648.24 | 5,405,900 |
| **4** | 1/9/2012 | 322.04 | 322.29 | 309.46 | 620.76 | 11,688,800 |

In [4]:

train **=** dataset\_train.loc[:, ["Open"]].values

train

Out[4]:

array([[325.25],

[331.27],

[329.83],

...,

[793.7 ],

[783.33],

[782.75]])

In [5]:

*# Feature Scaling*

**from** sklearn.preprocessing **import** MinMaxScaler

scaler **=** MinMaxScaler(feature\_range **=** (0, 1))

train\_scaled **=** scaler.fit\_transform(train)

train\_scaled

Out[5]:

array([[0.08581368],

[0.09701243],

[0.09433366],

...,

[0.95725128],

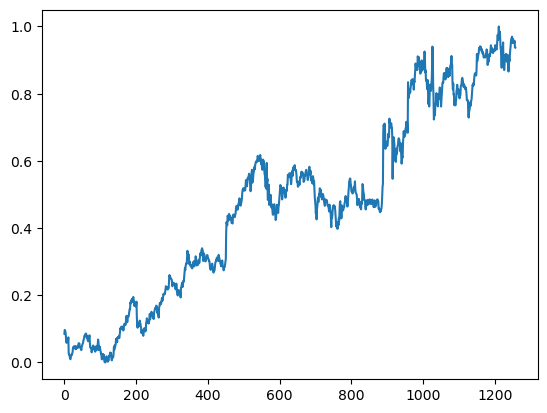
[0.93796041],

[0.93688146]])

In [6]:

plt.plot(train\_scaled)

plt.show()



In [8]:

*# Creating a data structure with 50 timesteps and 1 output*

X\_train **=** []

y\_train **=** []

timesteps **=** 50

**for** i **in** range(timesteps, 1258):

X\_train.append(train\_scaled[i**-**timesteps:i, 0])

y\_train.append(train\_scaled[i, 0])

X\_train, y\_train **=** np.array(X\_train), np.array(y\_train)

In [9]:

*# Reshaping*

X\_train **=** np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

X\_train

Out[9]:

array([[[0.08581368],

[0.09701243],

[0.09433366],

...,

[0.03675869],

[0.04486941],

[0.05065481]],

[[0.09701243],

[0.09433366],

[0.09156187],

...,

[0.04486941],

[0.05065481],

[0.05214302]],

[[0.09433366],

[0.09156187],

[0.07984225],

...,

[0.05065481],

[0.05214302],

[0.05612397]],

...,

[[0.9313937 ],

[0.94636878],

[0.96569685],

...,

[0.95475854],

[0.95204256],

[0.95163331]],

[[0.94636878],

[0.96569685],

[0.97510976],

...,

[0.95204256],

[0.95163331],

[0.95725128]],

[[0.96569685],

[0.97510976],

[0.95966962],

...,

[0.95163331],

[0.95725128],

[0.93796041]]])

In [10]:

y\_train

Out[10]:

array([0.05214302, 0.05612397, 0.05818885, ..., 0.95725128, 0.93796041,

0.93688146])

In [17]:

*# Importing the Keras libraries and packages*

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense

**from** keras.layers **import** SimpleRNN

**from** keras.layers **import** Dropout

​

*# Initialising the RNN*

regressor **=** Sequential()

​

*# Adding the first RNN layer and some Dropout regularisation*

regressor.add(SimpleRNN(units **=** 50,activation**=**'tanh', return\_sequences **=** **True**, input\_shape **=** (X\_train.shape[1], 1)))

regressor.add(Dropout(0.2))

​

*# Adding a second RNN layer and some Dropout regularisation*

regressor.add(SimpleRNN(units **=** 50,activation**=**'tanh', return\_sequences **=** **True**))

regressor.add(Dropout(0.2))

​

*# Adding a third RNN layer and some Dropout regularisation*

regressor.add(SimpleRNN(units **=** 50,activation**=**'tanh', return\_sequences **=** **True**))

regressor.add(Dropout(0.2))

​

*# Adding a fourth RNN layer and some Dropout regularisation*

regressor.add(SimpleRNN(units **=** 50))

regressor.add(Dropout(0.2))

​

*# Adding the output layer*

regressor.add(Dense(units **=** 1))

​

*# Compiling the RNN*

regressor.compile(optimizer **=** 'adam', loss **=** 'mean\_squared\_error')

​

*# Fitting the RNN to the Training set*

regressor.fit(X\_train, y\_train, epochs **=** 100, batch\_size **=** 32)

Epoch 1/100

38/38 [==============================] - 5s 31ms/step - loss: 0.4601

Epoch 2/100

38/38 [==============================] - 1s 32ms/step - loss: 0.3077

Epoch 3/100

38/38 [==============================] - 1s 32ms/step - loss: 0.2229

Epoch 4/100

38/38 [==============================] - 1s 33ms/step - loss: 0.1764

Epoch 5/100

38/38 [==============================] - 1s 33ms/step - loss: 0.1365

Epoch 6/100

38/38 [==============================] - 1s 33ms/step - loss: 0.1017

Epoch 7/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0822

Epoch 8/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0723

Epoch 9/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0611

Epoch 10/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0531

Epoch 11/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0461

Epoch 12/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0422

Epoch 13/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0358

Epoch 14/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0303

Epoch 15/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0295

Epoch 16/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0275

Epoch 17/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0239

Epoch 18/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0221

Epoch 19/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0210

Epoch 20/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0211

Epoch 21/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0165

Epoch 22/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0161

Epoch 23/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0152

Epoch 24/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0138

Epoch 25/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0143

Epoch 26/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0129

Epoch 27/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0115

Epoch 28/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0105

Epoch 29/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0107

Epoch 30/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0098

Epoch 31/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0089

Epoch 32/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0089

Epoch 33/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0085

Epoch 34/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0080

Epoch 35/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0084

Epoch 36/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0074

Epoch 37/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0073

Epoch 38/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0067

Epoch 39/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0066

Epoch 40/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0062

Epoch 41/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0059

Epoch 42/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0065

Epoch 43/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0054

Epoch 44/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0051

Epoch 45/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0049

Epoch 46/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0054

Epoch 47/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0056

Epoch 48/100

38/38 [==============================] - 2s 40ms/step - loss: 0.0054

Epoch 49/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0050

Epoch 50/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0047

Epoch 51/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0049

Epoch 52/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0046

Epoch 53/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0043

Epoch 54/100

38/38 [==============================] - 1s 38ms/step - loss: 0.0043

Epoch 55/100

38/38 [==============================] - 1s 36ms/step - loss: 0.0037

Epoch 56/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0039

Epoch 57/100

38/38 [==============================] - 1s 39ms/step - loss: 0.0042

Epoch 58/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0037

Epoch 59/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0039

Epoch 60/100

38/38 [==============================] - 1s 37ms/step - loss: 0.0039

Epoch 61/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0037

Epoch 62/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0039

Epoch 63/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0039

Epoch 64/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0037

Epoch 65/100

38/38 [==============================] - 1s 36ms/step - loss: 0.0034

Epoch 66/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0036

Epoch 67/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0033

Epoch 68/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0033

Epoch 69/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0030

Epoch 70/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0032

Epoch 71/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0033

Epoch 72/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0031

Epoch 73/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0031

Epoch 74/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0032

Epoch 75/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0031

Epoch 76/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0030

Epoch 77/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0029

Epoch 78/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0028

Epoch 79/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0030

Epoch 80/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0027

Epoch 81/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0029

Epoch 82/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0028

Epoch 83/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0029

Epoch 84/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0025

Epoch 85/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0024

Epoch 86/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0026

Epoch 87/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0025

Epoch 88/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0026

Epoch 89/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0024

Epoch 90/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0028

Epoch 91/100

38/38 [==============================] - 1s 34ms/step - loss: 0.0023

Epoch 92/100

38/38 [==============================] - 1s 35ms/step - loss: 0.0021

Epoch 93/100

38/38 [==============================] - 1s 36ms/step - loss: 0.0021

Epoch 94/100

38/38 [==============================] - 1s 37ms/step - loss: 0.0023

Epoch 95/100

38/38 [==============================] - 1s 39ms/step - loss: 0.0023

Epoch 96/100

38/38 [==============================] - 1s 37ms/step - loss: 0.0023

Epoch 97/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0022

Epoch 98/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0021

Epoch 99/100

38/38 [==============================] - 1s 33ms/step - loss: 0.0023

Epoch 100/100

38/38 [==============================] - 1s 32ms/step - loss: 0.0022

Out[17]:

<keras.src.callbacks.History at 0x11eb9c8c850>

In [18]:

*# Getting the real stock price of 2017*

dataset\_test **=** pd.read\_csv('Google\_Stock\_Price\_Test.csv')

dataset\_test.head()

Out[18]:

|  | **Date** | **Open** | **High** | **Low** | **Close** | **Volume** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 1/3/2017 | 778.81 | 789.63 | 775.80 | 786.14 | 1,657,300 |
| **1** | 1/4/2017 | 788.36 | 791.34 | 783.16 | 786.90 | 1,073,000 |
| **2** | 1/5/2017 | 786.08 | 794.48 | 785.02 | 794.02 | 1,335,200 |
| **3** | 1/6/2017 | 795.26 | 807.90 | 792.20 | 806.15 | 1,640,200 |
| **4** | 1/9/2017 | 806.40 | 809.97 | 802.83 | 806.65 | 1,272,400 |

In [19]:

real\_stock\_price **=** dataset\_test.loc[:, ["Open"]].values

real\_stock\_price

Out[19]:

array([[778.81],

[788.36],

[786.08],

[795.26],

[806.4 ],

[807.86],

[805. ],

[807.14],

[807.48],

[807.08],

[805.81],

[805.12],

[806.91],

[807.25],

[822.3 ],

[829.62],

[837.81],

[834.71],

[814.66],

[796.86]])

In [20]:

*# Getting the predicted stock price of 2017*

dataset\_total **=** pd.concat((dataset\_train['Open'], dataset\_test['Open']), axis **=** 0)

inputs **=** dataset\_total[len(dataset\_total) **-** len(dataset\_test) **-** timesteps:].values.reshape(**-**1,1)

inputs **=** scaler.transform(inputs) *# min max scaler*

inputs

Out[20]:

array([[0.97510976],

[0.95966962],

[0.97808617],

[1. ],

[0.98076494],

[0.97083116],

[0.98450406],

[0.96054394],

[0.9371419 ],

[0.92841729],

[0.90804747],

[0.8771858 ],

[0.92153434],

[0.93809063],

[0.93165414],

[0.95254483],

[0.88812412],

[0.88637547],

[0.87032145],

[0.88563137],

[0.90743359],

[0.91571173],

[0.89941588],

[0.91805566],

[0.9089404 ],

[0.9024853 ],

[0.89456061],

[0.91600938],

[0.9132934 ],

[0.88979835],

[0.86589404],

[0.89030062],

[0.90335962],

[0.89642086],

[0.91777662],

[0.93176576],

[0.94114145],

[0.95762334],

[0.96413424],

[0.96402262],

[0.96971501],

[0.95077759],

[0.96294367],

[0.96123223],

[0.95475854],

[0.95204256],

[0.95163331],

[0.95725128],

[0.93796041],

[0.93688146],

[0.92955205],

[0.94731751],

[0.94307612],

[0.96015329],

[0.98087655],

[0.98359253],

[0.97827219],

[0.98225314],

[0.98288563],

[0.98214153],

[0.979779 ],

[0.97849542],

[0.98182528],

[0.98245777],

[1.01045465],

[1.02407173],

[1.03930724],

[1.03354044],

[0.99624228],

[0.9631297 ]])

In [21]:

X\_test **=** []

**for** i **in** range(timesteps, 70):

X\_test.append(inputs[i**-**timesteps:i, 0])

X\_test **=** np.array(X\_test)

X\_test **=** np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

predicted\_stock\_price **=** regressor.predict(X\_test)

predicted\_stock\_price **=** scaler.inverse\_transform(predicted\_stock\_price)

​

*# Visualising the results*

plt.plot(real\_stock\_price, color **=** 'red', label **=** 'Real Google Stock Price')

plt.plot(predicted\_stock\_price, color **=** 'blue', label **=** 'Predicted Google Stock Price')

plt.title('Google Stock Price Prediction')

plt.xlabel('Time')

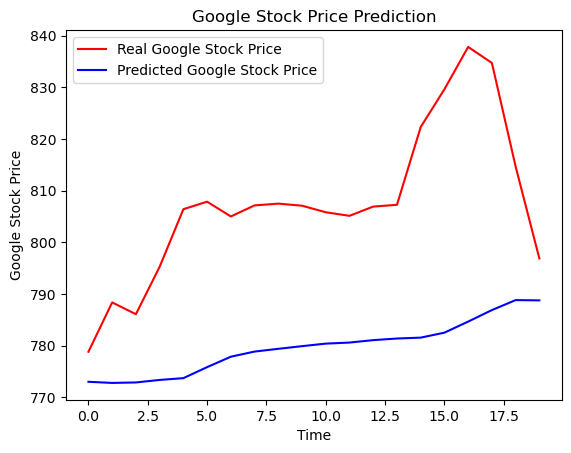
plt.ylabel('Google Stock Price')

plt.legend()

plt.show()

*# epoch = 250 daha güzel sonuç veriyor.*

1/1 [==============================] - 1s 510ms/step



In [ ]:

*#LSTM ---> international-airline-passengers.csv (dataset)*

In [22]:

**import** numpy

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** math

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense

**from** keras.layers **import** LSTM

**from** sklearn.preprocessing **import** MinMaxScaler

**from** sklearn.metrics **import** mean\_squared\_error

In [23]:

data **=** pd.read\_csv('international-airline-passengers.csv',skipfooter**=**5) *# The last 5 data was contaminated*

data.head()

Out[23]:

|  | **Month** | **International airline passengers: monthly totals in thousands. Jan 49 ? Dec 60** |
| --- | --- | --- |
| **0** | 1949-01 | 112 |
| **1** | 1949-02 | 118 |
| **2** | 1949-03 | 132 |
| **3** | 1949-04 | 129 |
| **4** | 1949-05 | 121 |

In [24]:

dataset **=** data.iloc[:,1].values

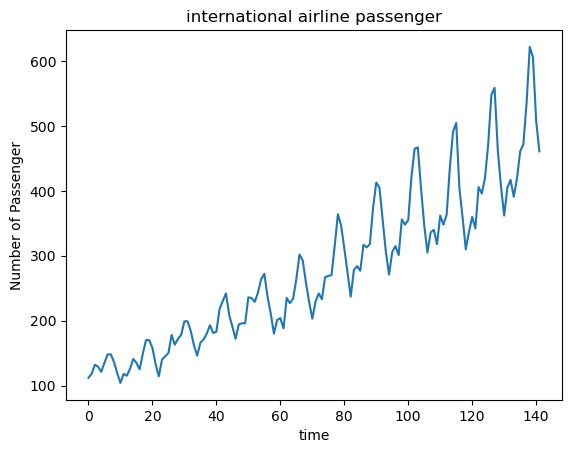
plt.plot(dataset)

plt.xlabel("time")

plt.ylabel("Number of Passenger")

plt.title("international airline passenger")

plt.show()



In [25]:

dataset **=** dataset.reshape(**-**1,1)

dataset **=** dataset.astype("float32")

dataset.shape

Out[25]:

(142, 1)

In [26]:

*# scaling*

scaler **=** MinMaxScaler(feature\_range**=**(0, 1))

dataset **=** scaler.fit\_transform(dataset)

In [27]:

train\_size **=** int(len(dataset) **\*** 0.50)

test\_size **=** len(dataset) **-** train\_size

train **=** dataset[0:train\_size,:]

test **=** dataset[train\_size:len(dataset),:]

print("train size: {}, test size: {} ".format(len(train), len(test)))

train size: 71, test size: 71

In [28]:

time\_stemp **=** 10

dataX **=** []

dataY **=** []

**for** i **in** range(len(train)**-**time\_stemp**-**1):

a **=** train[i:(i**+**time\_stemp), 0]

dataX.append(a)

dataY.append(train[i **+** time\_stemp, 0])

trainX **=** numpy.array(dataX)

trainY **=** numpy.array(dataY)

In [29]:

dataX **=** []

dataY **=** []

**for** i **in** range(len(test)**-**time\_stemp**-**1):

a **=** test[i:(i**+**time\_stemp), 0]

dataX.append(a)

dataY.append(test[i **+** time\_stemp, 0])

testX **=** numpy.array(dataX)

testY **=** numpy.array(dataY)

In [30]:

trainX **=** numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX **=** numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

In [31]:

*# model*

model **=** Sequential()

model.add(LSTM(10, input\_shape**=**(1, time\_stemp))) *# 10 lstm neuron(block)*

model.add(Dense(1))

model.compile(loss**=**'mean\_squared\_error', optimizer**=**'adam')

model.fit(trainX, trainY, epochs**=**50, batch\_size**=**1)

Epoch 1/50

60/60 [==============================] - 3s 4ms/step - loss: 0.0080

Epoch 2/50

60/60 [==============================] - 0s 4ms/step - loss: 0.0034

Epoch 3/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0033

Epoch 4/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0031

Epoch 5/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0030

Epoch 6/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0029

Epoch 7/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0029

Epoch 8/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0027

Epoch 9/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0025

Epoch 10/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0024

Epoch 11/50

60/60 [==============================] - 0s 4ms/step - loss: 0.0023

Epoch 12/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0022

Epoch 13/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0022

Epoch 14/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0021

Epoch 15/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0020

Epoch 16/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0020

Epoch 17/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0020

Epoch 18/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0018

Epoch 19/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0018

Epoch 20/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0018

Epoch 21/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0017

Epoch 22/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0017

Epoch 23/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0016

Epoch 24/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0017

Epoch 25/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0016

Epoch 26/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0015

Epoch 27/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0015

Epoch 28/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0015

Epoch 29/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0014

Epoch 30/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0015

Epoch 31/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0014

Epoch 32/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0014

Epoch 33/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0014

Epoch 34/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0013

Epoch 35/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0013

Epoch 36/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0013

Epoch 37/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0013

Epoch 38/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0013

Epoch 39/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0013

Epoch 40/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0013

Epoch 41/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0012

Epoch 42/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0012

Epoch 43/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0012

Epoch 44/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0012

Epoch 45/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0012

Epoch 46/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0013

Epoch 47/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0012

Epoch 48/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0011

Epoch 49/50

60/60 [==============================] - 0s 3ms/step - loss: 0.0013

Epoch 50/50

60/60 [==============================] - 0s 2ms/step - loss: 0.0012

Out[31]:

<keras.src.callbacks.History at 0x11ebe463650>

In [32]:

trainPredict **=** model.predict(trainX)

testPredict **=** model.predict(testX)

*# invert predictions*

trainPredict **=** scaler.inverse\_transform(trainPredict)

trainY **=** scaler.inverse\_transform([trainY])

testPredict **=** scaler.inverse\_transform(testPredict)

testY **=** scaler.inverse\_transform([testY])

*# calculate root mean squared error*

trainScore **=** math.sqrt(mean\_squared\_error(trainY[0], trainPredict[:,0]))

print('Train Score: %.2f RMSE' **%** (trainScore))

testScore **=** math.sqrt(mean\_squared\_error(testY[0], testPredict[:,0]))

print('Test Score: %.2f RMSE' **%** (testScore))

2/2 [==============================] - 1s 4ms/step

2/2 [==============================] - 0s 4ms/step

Train Score: 16.94 RMSE

Test Score: 43.58 RMSE

In [33]:

*# shifting train*

trainPredictPlot **=** numpy.empty\_like(dataset)

trainPredictPlot[:, :] **=** numpy.nan

trainPredictPlot[time\_stemp:len(trainPredict)**+**time\_stemp, :] **=** trainPredict

*# shifting test predictions for plotting*

testPredictPlot **=** numpy.empty\_like(dataset)

testPredictPlot[:, :] **=** numpy.nan

testPredictPlot[len(trainPredict)**+**(time\_stemp**\***2)**+**1:len(dataset)**-**1, :] **=** testPredict

*# plot baseline and predictions*

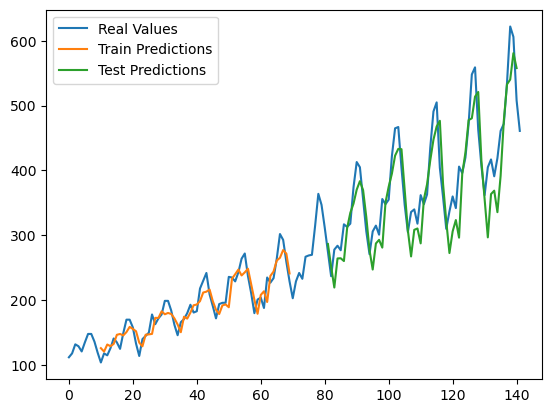
plt.plot(scaler.inverse\_transform(dataset), label **=** "Real Values")

plt.plot(trainPredictPlot, label **=** "Train Predictions")

plt.plot(testPredictPlot, label **=** "Test Predictions")

plt.legend()

plt.show()



In [ ]:

​