Deep Learning: Assignment-9

Training a LSTM Model to predict stock prices of COAL India Stock price

Submitted By: Mrinal Bhan

DSAI 211020428

1.Download/search the Stock market dataset from https://www.kaggle.com/datasets/rohanrao/nifty50-stock-market-data. You have given the flexibility to use any other stock dataset if found suitable.

```
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime as dt

```

```
data = pd.read_csv('/kaggle/input/nifty50-stock-market-data/COALINDIA.csv')
```

```
data.head(10)
```

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliv
0	2010-11-04	COALINDIA	EQ	245.00	291.00	344.90	291.00	342.00	342.55	327.29	479716245	1.570040e+16	NaN	187584905	0
1	2010-11-05	COALINDIA	EQ	342.55	343.00	356.50	343.00	348.30	349.85	349.78	31927173	1.116747e+15	NaN	10894509	0
2	2010-11-08	COALINDIA	EQ	349.85	351.80	355.90	329.50	331.40	330.75	335.19	46932779	1.573118e+15	NaN	16651623	0
3	2010-11-09	COALINDIA	EQ	330.75	330.15	333.40	325.00	325.40	326.05	327.75	23741956	7.781383e+14	NaN	12977359	0
4	2010-11-10	COALINDIA	EQ	326.05	325.40	327.80	320.05	321.30	322.80	323.78	21057129	6.817982e+14	NaN	6280335	0
5	2010-11-11	COALINDIA	EQ	322.80	323.00	336.95	321.85	329.60	328.55	331.43	26548372	8.799018e+14	NaN	9365609	0
6	2010-11-12	COALINDIA	EQ	328.55	330.00	332.95	318.00	320.55	320.15	325.77	15004107	4.887872e+14	NaN	5797721	0
7	2010-11-15	COALINDIA	EQ	320.15	321.20	322.90	315.05	318.10	317.20	319.39	9917395	3.167544e+14	NaN	4812011	0
8	2010-11-16	COALINDIA	EQ	317.20	320.00	323.95	316.10	317.50	318.05	319.21	12100114	3.862460e+14	NaN	6620732	0
9	2010-11-18	COALINDIA	EQ	318.05	319.60	328.95	310.00	327.30	326.90	319.09	23983896	7.652941e+14	NaN	9152722	0

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2598 entries, 0 to 2597
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Date                2598 non-null   object 
 1   Symbol              2598 non-null   object 
 2   Series              2598 non-null   object 
 3   Prev Close          2598 non-null   float64
 4   Open                2598 non-null   float64
 5   High                2598 non-null   float64
 6   Low                 2598 non-null   float64
 7   Last                2598 non-null   float64
 8   Close               2598 non-null   float64
 9   VWAP                2598 non-null   float64
 10  Volume              2598 non-null   int64  
 11  Turnover            2598 non-null   float64
 12  Trades              2456 non-null   float64
 13  Deliverable Volume  2598 non-null   int64  
 14  %Deliverble         2598 non-null   float64
dtypes: float64(10), int64(2), object(3)
memory usage: 304.6+ KB
```

2. Apply any two Preprocessing technique to the above datasets.

```
data.isnull().sum()
```

```
Date                   0
Symbol                 0
Series                 0
Prev Close             0
Open                   0
High                   0
Low                    0
Last                   0
Close                  0
```

```
VWAP                   0
Volume                 0
Turnover               0
Trades               142
Deliverable Volume     0
%Deliverble            0
dtype: int64
```

```
data = data.drop(['Symbol', 'Series', 'Prev Close', 'Last', 'VWAP', 'Turnover', 'Trades', 'Deliverable Volume', '%Deliverble'], axis = 1)
```

```
data.head(10)
```

	Date	Open	High	Low	Close	Volume
0	2010-11-04	291.00	344.90	291.00	342.55	479716245
1	2010-11-05	343.00	356.50	343.00	349.85	31927173
2	2010-11-08	351.80	355.90	329.50	330.75	46932779
3	2010-11-09	330.15	333.40	325.00	326.05	23741956
4	2010-11-10	325.40	327.80	320.05	322.80	21057129
5	2010-11-11	323.00	336.95	321.85	328.55	26548372
6	2010-11-12	330.00	332.95	318.00	320.15	15004107
7	2010-11-15	321.20	322.90	315.05	317.20	9917395
8	2010-11-16	320.00	323.95	316.10	318.05	12100114
9	2010-11-18	319.60	328.95	310.00	326.90	23983896

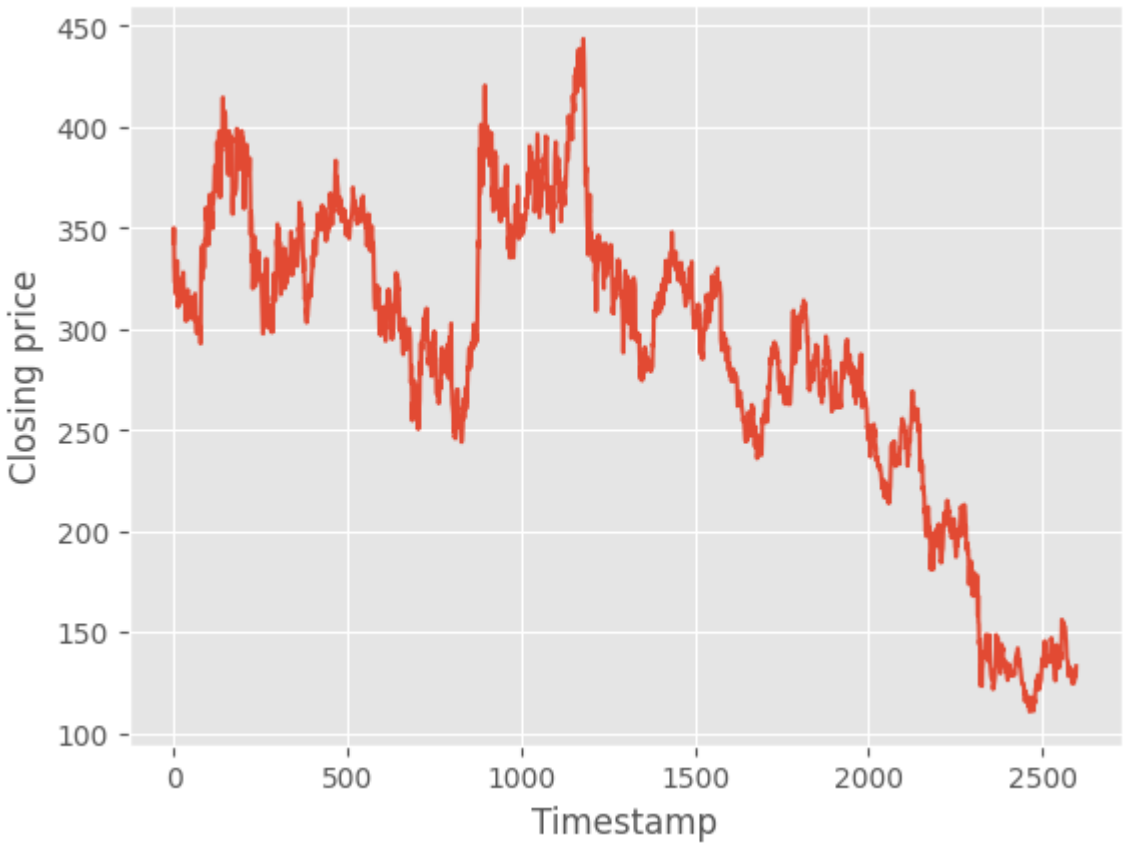
```
data.isnull().sum()
```

```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

```
plt.plot(data['Close'], label="Close price")
plt.xlabel("Timestamp")
plt.ylabel("Closing price")
df = data
print(df)
```

```
            Date    Open    High     Low   Close     Volume
0     2010-11-04  291.00  344.90  291.00  342.55  479716245
1     2010-11-05  343.00  356.50  343.00  349.85   31927173
2     2010-11-08  351.80  355.90  329.50  330.75   46932779
3     2010-11-09  330.15  333.40  325.00  326.05   23741956
4     2010-11-10  325.40  327.80  320.05  322.80   21057129
...          ...     ...     ...     ...     ...        ...
2593  2021-04-26  127.75  128.00  126.50  126.95    4967884
2594  2021-04-27  127.00  127.90  126.60  127.50    3982954
2595  2021-04-28  128.00  129.45  127.50  128.50    6206074
2596  2021-04-29  129.75  130.05  127.65  128.05    8345584
2597  2021-04-30  127.40  134.60  127.00  133.05   27396950

[2598 rows x 6 columns]
```



```
data.reset_index(drop=True, inplace=True)
data
```

	Date	Open	High	Low	Close	Volume
0	2010-11-04	291.00	344.90	291.00	342.55	479716245
1	2010-11-05	343.00	356.50	343.00	349.85	31927173
2	2010-11-08	351.80	355.90	329.50	330.75	46932779
3	2010-11-09	330.15	333.40	325.00	326.05	23741956
4	2010-11-10	325.40	327.80	320.05	322.80	21057129
...	...	...	...	...	...	...
2593	2021-04-26	127.75	128.00	126.50	126.95	4967884
2594	2021-04-27	127.00	127.90	126.60	127.50	3982954
2595	2021-04-28	128.00	129.45	127.50	128.50	6206074
2596	2021-04-29	129.75	130.05	127.65	128.05	8345584
2597	2021-04-30	127.40	134.60	127.00	133.05	27396950

2598 rows × 6 columns

```
from sklearn.model_selection import train_test_split
y = data['Close']
df = data
X = df.drop(['Close'], axis = 1)
X
```

	Date	Open	High	Low	Volume
0	2010-11-04	291.00	344.90	291.00	479716245
1	2010-11-05	343.00	356.50	343.00	31927173
2	2010-11-08	351.80	355.90	329.50	46932779
3	2010-11-09	330.15	333.40	325.00	23741956
4	2010-11-10	325.40	327.80	320.05	21057129
...	...	...	...	...	...
2593	2021-04-26	127.75	128.00	126.50	4967884
2594	2021-04-27	127.00	127.90	126.60	3982954
2595	2021-04-28	128.00	129.45	127.50	6206074
2596	2021-04-29	129.75	130.05	127.65	8345584
2597	2021-04-30	127.40	134.60	127.00	27396950

2598 rows × 5 columns

```
from sklearn.model_selection import train_test_split

X = []
Y = []
window_size=100
for i in range(1 , len(data) - window_size -1 , 1):
    first = data.iloc[i,2]
    temp = []
    temp2 = []
    for j in range(window_size):
        temp.append((data.iloc[i + j, 2] - first) / first)
    temp2.append((data.iloc[i + window_size, 2] - first) / first)
    X.append(np.array(temp).reshape(100, 1))
    Y.append(np.array(temp2).reshape(1, 1))

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, shuffle=False)

train_X = np.array(x_train)
test_X = np.array(x_test)
train_Y = np.array(y_train)
test_Y = np.array(y_test)

train_X = train_X.reshape(train_X.shape[0],1,100,1)
test_X = test_X.reshape(test_X.shape[0],1,100,1)

print(len(train_X))
print(len(test_X))
```

```
1996
```

500

3. Apply the LSTM architecture to the above dataset.

```
import tensorflow as tf
from tensorflow.keras.layers import Conv1D, LSTM, Dense, Dropout, Bidirectional, TimeDistributed
from tensorflow.keras.layers import MaxPooling1D, Flatten
from tensorflow.keras.regularizers import L1, L2
from tensorflow.keras.metrics import Accuracy
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam

def lstm_model(lr, af):
    model = tf.keras.Sequential()

    # CNN layers
    model.add(TimeDistributed(Conv1D(64, kernel_size=3, activation='relu', input_shape=(None, 100, 1))))
    model.add(TimeDistributed(MaxPooling1D(2)))
    model.add(TimeDistributed(Conv1D(128, kernel_size=3, activation='relu')))
    model.add(TimeDistributed(MaxPooling1D(2)))
    model.add(TimeDistributed(Conv1D(64, kernel_size=3, activation='relu')))
    model.add(TimeDistributed(MaxPooling1D(2)))
    model.add(TimeDistributed(Flatten()))
    # model.add(Dense(5, kernel_regularizer=L2(0.01)))

    # LSTM layers
    model.add(Bidirectional(LSTM(100, activation = af, return_sequences=True)))
    model.add(Dropout(0.5))
    model.add(Bidirectional(LSTM(100, activation = af, return_sequences=False)))
    model.add(Dropout(0.5))

    #Final layers
    model.add(Dense(1, activation='linear'))
    model.compile(optimizer=Adam(learning_rate=lr), loss='mse', metrics=['mse', 'mae'])

    history = model.fit(train_X, train_Y, validation_data=(test_X,test_Y), epochs=10,batch_size=40, verbose=1, shuffle =False)
    return model
```

4 . Demonstrate the results with various learning rates.

5. Demonstrate the results with various Activation functions.

```
lr = [0.0001, 0.001, 0.01, 0.1]
af = ['relu', 'tanh', 'linear', 'sigmoid']
```

```
print('Model 1 with learning rate of 0.0001')
model1 = lstm_model(lr[0], af[1])
print('Model 2 with learning rate of 0.001\n')
model2 = lstm_model(lr[1], af[1])
print('Model 3 with learning rate of 0.01\n')
model3 = lstm_model(lr[2], af[1])
print('Model 4 with learning rate of 0.1\n')
model4 = lstm_model(lr[3], af[1])
```

```
Model 1 with learning rate of 0.0001
Epoch 1/10
50/50 [==============================] - 13s 61ms/step - loss: 0.0208 - mse: 0.0208 - mae: 0.1118 - val_loss: 0.0317 - val_mse: 0.0317 - v
Epoch 2/10
50/50 [==============================] - 1s 30ms/step - loss: 0.0178 - mse: 0.0178 - mae: 0.1014 - val_loss: 0.0193 - val_mse: 0.0193 - va
Epoch 3/10
50/50 [==============================] - 1s 30ms/step - loss: 0.0115 - mse: 0.0115 - mae: 0.0783 - val_loss: 0.0108 - val_mse: 0.0108 - va
Epoch 4/10
50/50 [==============================] - 2s 32ms/step - loss: 0.0083 - mse: 0.0083 - mae: 0.0683 - val_loss: 0.0096 - val_mse: 0.0096 - va
Epoch 5/10
50/50 [==============================] - 1s 28ms/step - loss: 0.0071 - mse: 0.0071 - mae: 0.0635 - val_loss: 0.0085 - val_mse: 0.0085 - va
Epoch 6/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0060 - mse: 0.0060 - mae: 0.0583 - val_loss: 0.0076 - val_mse: 0.0076 - va
Epoch 7/10
50/50 [==============================] - 1s 28ms/step - loss: 0.0052 - mse: 0.0052 - mae: 0.0542 - val_loss: 0.0068 - val_mse: 0.0068 - va
Epoch 8/10
50/50 [==============================] - 1s 29ms/step - loss: 0.0050 - mse: 0.0050 - mae: 0.0525 - val_loss: 0.0063 - val_mse: 0.0063 - va
Epoch 9/10
50/50 [==============================] - 1s 28ms/step - loss: 0.0046 - mse: 0.0046 - mae: 0.0508 - val_loss: 0.0061 - val_mse: 0.0061 - va
Epoch 10/10
50/50 [==============================] - 1s 28ms/step - loss: 0.0044 - mse: 0.0044 - mae: 0.0494 - val_loss: 0.0059 - val_mse: 0.0059 - va
Model 2 with learning rate of 0.001

Epoch 1/10
50/50 [==============================] - 13s 62ms/step - loss: 0.0163 - mse: 0.0163 - mae: 0.0941 - val_loss: 0.0099 - val_mse: 0.0099 - v
Epoch 2/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0066 - mse: 0.0066 - mae: 0.0617 - val_loss: 0.0074 - val_mse: 0.0074 - va
Epoch 3/10
50/50 [==============================] - 1s 29ms/step - loss: 0.0069 - mse: 0.0069 - mae: 0.0605 - val_loss: 0.0077 - val_mse: 0.0077 - va
Epoch 4/10
50/50 [==============================] - 1s 29ms/step - loss: 0.0057 - mse: 0.0057 - mae: 0.0592 - val_loss: 0.0111 - val_mse: 0.0111 - v
```

```
Epoch 5/10
50/50 [==============================] - 2s 32ms/step - loss: 0.0066 - mse: 0.0066 - mae: 0.0621 - val_loss: 0.0101 - val_mse: 0.0101 - v
Epoch 6/10
50/50 [==============================] - 1s 30ms/step - loss: 0.0047 - mse: 0.0047 - mae: 0.0525 - val_loss: 0.0055 - val_mse: 0.0055 - v
Epoch 7/10
50/50 [==============================] - 1s 29ms/step - loss: 0.0042 - mse: 0.0042 - mae: 0.0488 - val_loss: 0.0063 - val_mse: 0.0063 - v
Epoch 8/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0039 - mse: 0.0039 - mae: 0.0475 - val_loss: 0.0068 - val_mse: 0.0068 - v
Epoch 9/10
50/50 [==============================] - 1s 29ms/step - loss: 0.0038 - mse: 0.0038 - mae: 0.0473 - val_loss: 0.0067 - val_mse: 0.0067 - v
Epoch 10/10
50/50 [==============================] - 1s 28ms/step - loss: 0.0041 - mse: 0.0041 - mae: 0.0481 - val_loss: 0.0064 - val_mse: 0.0064 - v
Model 3 with learning rate of 0.01

Epoch 1/10
50/50 [==============================] - 13s 61ms/step - loss: 0.0841 - mse: 0.0841 - mae: 0.1593 - val_loss: 0.0269 - val_mse: 0.0269 -
Epoch 2/10
50/50 [==============================] - 1s 28ms/step - loss: 0.0244 - mse: 0.0244 - mae: 0.1204 - val_loss: 0.0276 - val_mse: 0.0276 - v
Epoch 3/10
50/50 [==============================] - 1s 28ms/step - loss: 0.0238 - mse: 0.0238 - mae: 0.1190 - val_loss: 0.0278 - val_mse: 0.0278 - v
Epoch 4/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0235 - mse: 0.0235 - mae: 0.1183 - val_loss: 0.0285 - val_mse: 0.0285 - v
Epoch 5/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0232 - mse: 0.0232 - mae: 0.1175 - val_loss: 0.0291 - val_mse: 0.0291 - v
Epoch 6/10
50/50 [==============================] - 1s 29ms/step - loss: 0.0230 - mse: 0.0230 - mae: 0.1169 - val_loss: 0.0297 - val_mse: 0.0297 - v
Epoch 7/10
```

```
print('Model 5 with activation function - ReLU')
model5 = lstm_model(lr[1], af[0])
print('\nModel 6 with activation function - Tanh')
model6 = lstm_model(lr[1], af[1])
print('\nModel 7 with activation function - linear')
model7 = lstm_model(lr[1], af[2])
print('\nModel 8 with activation function - sigmoid')
model8 = lstm_model(lr[1], af[3])
```

```
Model 5 with activation function - ReLU
Epoch 1/10
50/50 [==============================] - 9s 44ms/step - loss: 0.0202 - mse: 0.0202 - mae: 0.1099 - val_loss: 0.0198 - val_mse: 0.0198 - v
Epoch 2/10
50/50 [==============================] - 1s 28ms/step - loss: 0.0103 - mse: 0.0103 - mae: 0.0765 - val_loss: 0.0136 - val_mse: 0.0136 - v
Epoch 3/10
50/50 [==============================] - 1s 26ms/step - loss: 0.0070 - mse: 0.0070 - mae: 0.0645 - val_loss: 0.0130 - val_mse: 0.0130 - v
Epoch 4/10
50/50 [==============================] - 1s 26ms/step - loss: 0.0070 - mse: 0.0070 - mae: 0.0628 - val_loss: 0.0090 - val_mse: 0.0090 - v
Epoch 5/10
50/50 [==============================] - 1s 26ms/step - loss: 0.0049 - mse: 0.0049 - mae: 0.0541 - val_loss: 0.0084 - val_mse: 0.0084 - v
Epoch 6/10
50/50 [==============================] - 1s 25ms/step - loss: 0.0048 - mse: 0.0048 - mae: 0.0532 - val_loss: 0.0081 - val_mse: 0.0081 - v
Epoch 7/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0049 - mse: 0.0049 - mae: 0.0530 - val_loss: 0.0091 - val_mse: 0.0091 - v
Epoch 8/10
50/50 [==============================] - 1s 28ms/step - loss: 0.0042 - mse: 0.0042 - mae: 0.0501 - val_loss: 0.0075 - val_mse: 0.0075 - v
Epoch 9/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0047 - mse: 0.0047 - mae: 0.0521 - val_loss: 0.0064 - val_mse: 0.0064 - v
Epoch 10/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0037 - mse: 0.0037 - mae: 0.0467 - val_loss: 0.0067 - val_mse: 0.0067 - v

Model 6 with activation function - Tanh
Epoch 1/10
50/50 [==============================] - 13s 64ms/step - loss: 0.0170 - mse: 0.0170 - mae: 0.0965 - val_loss: 0.0111 - val_mse: 0.0111 -
Epoch 2/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0069 - mse: 0.0069 - mae: 0.0625 - val_loss: 0.0070 - val_mse: 0.0070 - v
Epoch 3/10
50/50 [==============================] - 1s 26ms/step - loss: 0.0062 - mse: 0.0062 - mae: 0.0577 - val_loss: 0.0079 - val_mse: 0.0079 - v
Epoch 4/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0062 - mse: 0.0062 - mae: 0.0610 - val_loss: 0.0091 - val_mse: 0.0091 - v
Epoch 5/10
50/50 [==============================] - 1s 26ms/step - loss: 0.0068 - mse: 0.0068 - mae: 0.0619 - val_loss: 0.0051 - val_mse: 0.0051 - v
Epoch 6/10
50/50 [==============================] - 1s 26ms/step - loss: 0.0044 - mse: 0.0044 - mae: 0.0510 - val_loss: 0.0055 - val_mse: 0.0055 - v
Epoch 7/10
50/50 [==============================] - 1s 26ms/step - loss: 0.0038 - mse: 0.0038 - mae: 0.0475 - val_loss: 0.0052 - val_mse: 0.0052 - v
Epoch 8/10
50/50 [==============================] - 1s 26ms/step - loss: 0.0041 - mse: 0.0041 - mae: 0.0488 - val_loss: 0.0055 - val_mse: 0.0055 - v
Epoch 9/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0035 - mse: 0.0035 - mae: 0.0453 - val_loss: 0.0060 - val_mse: 0.0060 - v
Epoch 10/10
50/50 [==============================] - 1s 26ms/step - loss: 0.0036 - mse: 0.0036 - mae: 0.0460 - val_loss: 0.0058 - val_mse: 0.0058 - v

Model 7 with activation function - linear
Epoch 1/10
50/50 [==============================] - 10s 57ms/step - loss: 0.0170 - mse: 0.0170 - mae: 0.0966 - val_loss: 0.0104 - val_mse: 0.0104 -
Epoch 2/10
50/50 [==============================] - 1s 29ms/step - loss: 0.0071 - mse: 0.0071 - mae: 0.0620 - val_loss: 0.0073 - val_mse: 0.0073 - v
Epoch 3/10
50/50 [==============================] - 2s 32ms/step - loss: 0.0057 - mse: 0.0057 - mae: 0.0558 - val_loss: 0.0065 - val_mse: 0.0065 - v
Epoch 4/10
50/50 [==============================] - 1s 28ms/step - loss: 0.0051 - mse: 0.0051 - mae: 0.0563 - val_loss: 0.0096 - val_mse: 0.0096 - v
```

```
Epoch 5/10
50/50 [==============================] - 1s 27ms/step - loss: 0.0070 - mse: 0.0070 - mae: 0.0633 - val_loss: 0.0070 - val_mse: 0.0070 - v
Epoch 6/10
50/50 [==============================] - 1s 26ms/step - loss: 0.0059 - mse: 0.0059 - mae: 0.0592 - val_loss: 0.0072 - val_mse: 0.0072 - v
Epoch 7/10
```

6. Demonstrate the results with all possible evaluation criteria.

```
res_lr = []
res_lr.append(model1.evaluate(test_X, test_Y))
res_lr.append(model2.evaluate(test_X, test_Y))
res_lr.append(model3.evaluate(test_X, test_Y))
res_lr.append(model4.evaluate(test_X, test_Y))
modelNums = ['\n-------Model1-------', '\n-------Model2-------', '\n-------Model3-------', '\n-------Model4-------']
para = ['Test Loss : ', 'MSE : ', 'MAE : ']
for i in range(4):
    print(modelNums[i])
    for j in range(3):
        print(para[j], res_lr[i][j])
```

```
16/16 [==============================] - 0s 8ms/step - loss: 0.0059 - mse: 0.0059 - mae: 0.0622
16/16 [==============================] - 0s 9ms/step - loss: 0.0064 - mse: 0.0064 - mae: 0.0648
16/16 [==============================] - 0s 8ms/step - loss: 0.0302 - mse: 0.0302 - mae: 0.1453
16/16 [==============================] - 0s 7ms/step - loss: 0.0887 - mse: 0.0887 - mae: 0.2605

-------Model1-------
Test Loss :  0.00589575432240963
MSE :  0.00589575432240963
MAE :  0.06219031661748886

-------Model2-------
Test Loss :  0.006415408104658127
MSE :  0.006415408104658127
MAE :  0.06482697278261185

-------Model3-------
Test Loss :  0.030163953080773354
MSE :  0.030163953080773354
MAE :  0.14534984529018402

-------Model4-------
Test Loss :  0.08865439891815186
MSE :  0.08865439891815186
MAE :  0.26054394245147705
```

```
res_af = []
res_af.append(model5.evaluate(test_X, test_Y))
res_af.append(model6.evaluate(test_X, test_Y))
res_af.append(model7.evaluate(test_X, test_Y))
res_af.append(model8.evaluate(test_X, test_Y))
modelNums = ['\n-------Model5(ReLU)-------', '\n-------Model6(Tanh)-------', '\n-------Model7(Linear)-------', '\n-------Model8(Sigmoid)--
para = ['Test Loss : ', 'MSE : ', 'MAE : ']
for i in range(4):
    print(modelNums[i])
    for j in range(3):
        print(para[j], res_af[i][j])
```

```
16/16 [==============================] - 0s 7ms/step - loss: 0.0067 - mse: 0.0067 - mae: 0.0670
16/16 [==============================] - 0s 8ms/step - loss: 0.0058 - mse: 0.0058 - mae: 0.0610
16/16 [==============================] - 0s 7ms/step - loss: 0.0074 - mse: 0.0074 - mae: 0.0703
16/16 [==============================] - 0s 7ms/step - loss: 0.0119 - mse: 0.0119 - mae: 0.0888

-------Model5(ReLU)-------
Test Loss :  0.006718510761857033
MSE :  0.006718510761857033
MAE :  0.06697747856378555

-------Model6(Tanh)-------
Test Loss :  0.005775176454335451
MSE :  0.005775176454335451
MAE :  0.061000458896160126

-------Model7(Linear)-------
Test Loss :  0.0073815868236124516
MSE :  0.0073815868236124516
MAE :  0.07033152133226395

-------Model8(Sigmoid)-------
Test Loss :  0.011858091689646244
MSE :  0.011858091689646244
MAE :  0.08875854313373566
```

8. Predict the stock market value based on your implemented model.

```
from sklearn.metrics import explained_variance_score, mean_poisson_deviance, mean_gamma_deviance
from sklearn.metrics import r2_score
```

```
from sklearn.metrics import max_error

yhat_probs = model2.predict(test_X, verbose=0)
yhat_probs = yhat_probs[:, 0]
print('Model with learning rate = 0.001 and activation function - Tanh\n')
var = explained_variance_score(test_Y.reshape(-1,1), yhat_probs)
print('Variance: %f' % var)

r2 = r2_score(test_Y.reshape(-1,1), yhat_probs)
print('R2 Score: %f' % var)

var2 = max_error(test_Y.reshape(-1,1), yhat_probs)
print('Max Error: %f' % var2)
```

```
Model with learning rate = 0.001 and activation function - Tanh

Variance: 0.818686
R2 Score: 0.818686
Max Error: 0.246054
```

9. Demonstrate the learning results in all possible visualizing ways.

```
predicted  = model6.predict(test_X)
test_label = test_Y.reshape(-1,1)
predicted = np.array(predicted[:,0]).reshape(-1,1)
len_t = len(train_X)
for j in range(len_t , len_t + len(test_X)):
    temp = data.iloc[j,3]
    test_label[j - len_t] = test_label[j - len_t] * temp + temp
    predicted[j - len_t] = predicted[j - len_t] * temp + temp
plt.plot(predicted, color = 'green', label = 'Predicted  Stock Price')
plt.plot(test_label, color = 'red', label = 'Real Stock Price')
plt.title(' Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel(' Stock Price')
plt.legend()
plt.show()
```

```
16/16 [==============================] - 2s 7ms/step
```