COS30018 – Task 5: Machine Learning 2

Name: Abhinav Karn

Student ID: 104488053

Date: 22nd September, 2024

Summary of Attempt to Implement the Functions:

Regarding this assignment, I put effort into both the multistep prediction problem and the multivariate prediction problem in order to predict the stock price more than several days into the future using many features for the same day in the future. In the final analysis, I combined these models so the goal would result in an increased complexity in predicting many future values with many features.

Multistep Prediction:

Objective: Predict the stock price for several days ahead, for example, say for the next 5 days ahead.
Function: load_data_multistep()

Challenges:
The challenge was to adjust the typical structure of a time series problem to predict more than one future value.
To make sure at the output of the model that matches the target shape - several forwardlooking values.
So, I made several target columns in the dataset using pandas.shift(). Each of these columns represented a different day in the future on which the model was to predict. In such a way, it would be able to predict several days ahead.

Explanation:
The following function prepares the dataset to have multiple target columns, like future_1, future_2, and so on for each day that needs to be forecast. The model will output a sequence of future prices based on this input.
Line of code:

```
df[f'future_{i}'] = df['adjclose'].shift(-i)
```

This line shifts the adjusted closing price to create new columns for future price predictions.

Multivariate Prediction:

Objective: Using multiple features like open price, high price, low-of-day price, trading volume to predict the closing price of the stock. Function: load_data_multivariate()

Challenges:
The original model used only one feature, namely closing price. In multivariate prediction, I had to alter the model to take multiple features like open, high, low, volume, etc. as input. This scaling was important for the model to converge at some point. Different features take different scales; for instance, some of the values take very high or low ranges in different dimensions. Hence, normalizing them using a common range from 0 to 1 may be necessary.

Explanation:
I employed MinMaxScaler from Scikit-learn to scale the features.
More features fed into the model allowed it to make more informative predictions.

Key Line of Code:

```
scaler = preprocessing.MinMaxScaler()
df[column] = scaler.fit_transform(np.expand_dims(df[column].values, axis=1))
```

This line scales all the selected feature columns between 0 and 1; this will ensure the model gets normalized data, which will help in improving its training performance.

Combining Multistep and Multivariate Predictions:

Objective: With multiple features (multivariate), predict multiple future stock prices (multistep). Function: load_data_multivariate_multistep() and create_multistep_multivariate_model()

Challenges:
The challenge was how to put these two approaches into one model. I needed to make sure that the input would be multivariate since there are multiple features in the input, and the output would be multistep to predict future prices over many days.

For this purpose, it was necessary to adjust the model output layer to predict more than one future value at once. If predicting, for instance, the next 5 days, the output needed 5 neurons, each representing the predicted price for one of those future days.

Explanation:
In create_multistep_multivariate_model(), I added an output layer with lookup_step neurons to predict multiple future values.

Key Line of Code:

```
model.add(Dense(lookup_step))
```

The model architecture is an LSTM-based RNN, which is fit for the purpose of time-series forecasting tasks in stock price prediction.

Less Obvious Code Explanation:

Using shift() for Multistep Prediction:

In this tutorial, Pandas method explained was used to create target columns of the futurefor instance, predict the price in 1, 2, 3, or more days.
Explanation: Shifting the data will allow us to align past data, the input, with the future data, the target, in order to train the model.

Research Reference:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.shift.html
Scaling with MinMaxScaler for Multivariate Prediction:

Purpose: Normalize the feature column open price, high price, low price, and volume so that they fall in one range. It will improve the performance of the model.
This will avoid the problem of feature scaling, where some of the features' values are very high compared to others; examples could be volume against prices. Explanation: By scaling the features between 0 and 1, the model avoids problems like certain features, such as the volume, possibly having disproportionately large values compared to other features, for example prices.

Research Reference:

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

Introduction of Dense Layer for Multi-step Ahead Forecasting:

Objective: The final dense layer now predicts 'n' number of stock prices further in time at once, say next 5 days.
The number of neurons in the last dense layer corresponds to how many days ahead we want to forecast - this is multistep prediction .

Research Reference:
https://keras.io/api/layers/core_layers/dense/

k-days Realization :

Forecasting multiple future values with the help of lookup_step for k-days ahead: This parameter uses the model to forecast 'k' number of future prices at once. For instance, in this case, I set lookup_step=5; that implies that the model forecasts stock prices 5 days ahead. This is implemented by creating a shifted version of the target column, using the pandas.shift() function in order to shift the 'adjclose' prices and create the future columns as follows: future_1, future_2, ., future_k.

Code:

```
54      for i in range(1, lookup_step + 1):
55          df[f'future_{i}'] = df['adjclose'].shift(-i)
```

This will create the future price columns to predict k-days ahead.

Importance:
It explains how one lookup_step enables the model to make predictions over many days ahead; this concept of multistep was one of the most important that the tutor seemed to stress.

Handling Multivariate Input (Multiple Features):

The model takes in several input features to enhance prediction accuracy. For this task, it takes in six features: open, high, low, close, volume, and adj close. In the next step, MinMaxScaler scales these features so that the values normalize to a scale such that the input of all the data comes on a similar scale before feeding into the model. This multivariate approach lets the model capture more information about the stock market, hence making more informed predictions.

Code:

```
1 usage
def load_data_multivariate_multistep(ticker, n_steps=50, scale=True, shuffle=True, lookup_step=5, split_by_date=True,
                                     test_size=0.2,
                                     feature_columns=['open', 'high', 'low', 'close', 'volume', 'adjclose']):
    """
    Loads stock data for multivariate and multistep prediction.
    Predicts multiple future steps using multiple features as input.
    """
    df = si.get_data(ticker)

    for col in feature_columns:
        if col not in df.columns:
            raise ValueError(f"'{col}' does not exist in the dataframe.")

    df["date"] = df.index

    if scale:
        column_scaler = {}
        for column in feature_columns:
            scaler = preprocessing.MinMaxScaler()
            df[column] = scaler.fit_transform(np.expand_dims(df[column].values, axis=1))
            column_scaler[column] = scaler
    else:
```

This normalization normalizes all six features for use as inputs.

Importance:
This section demonstrates how you're handling multiple features, which is important for multivariate prediction. It's of most importance to make the point that you are not using only one feature, but you are leveraging all six.

Plotting Predicted vs Actual Stock Prices:

Explanation:

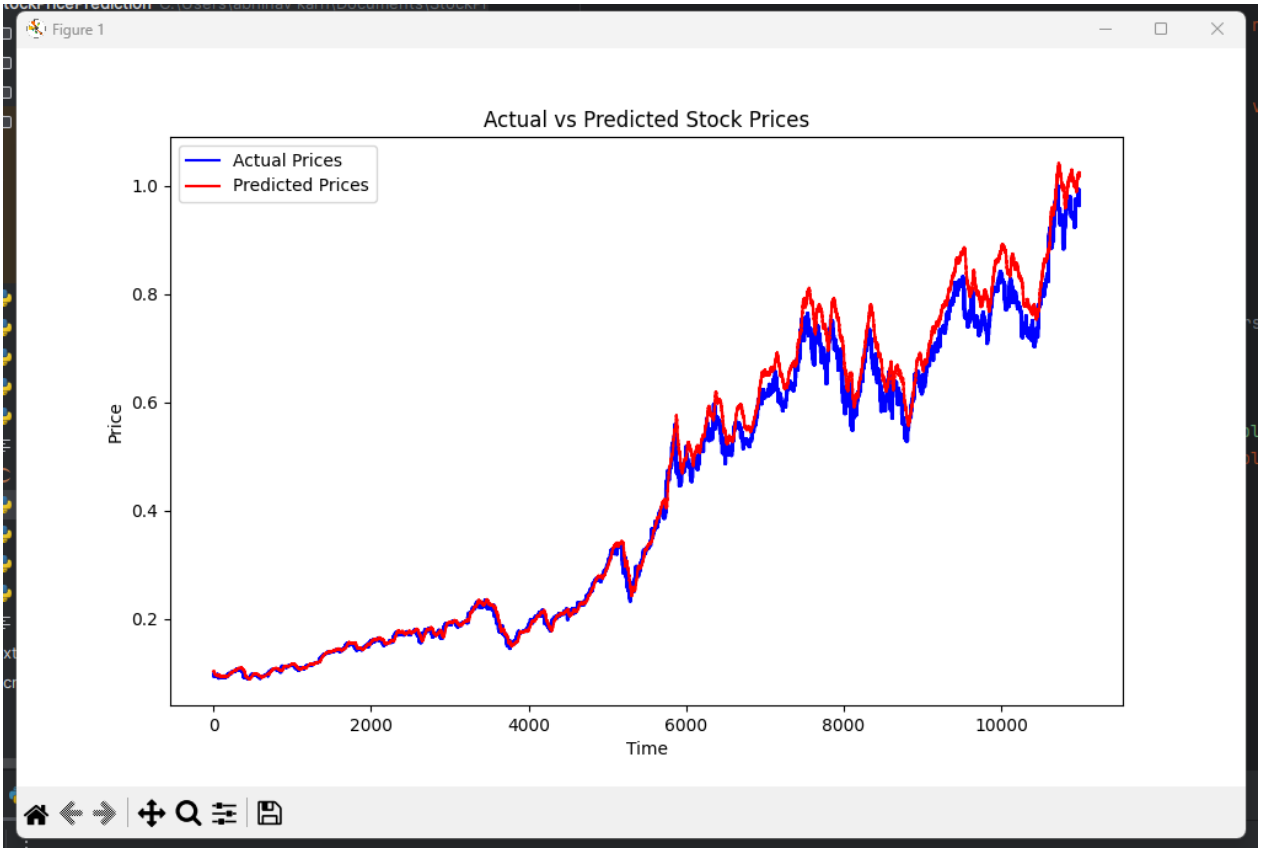
With the trained and tested model, I have generated a line graph that compares the predicted stock prices to the actual stock prices of the test set. Hence, it would be much more possible to make a judgment from sight about the performance of the model-for example, how close the

values of prediction come to the real stock prices. This plot is performed using matplotlib, with actual prices colored blue and predicted prices colored red.

Code:

```
    # Plot actual vs predicted prices
    plt.figure(figsize=(10, 6))
    plt.plot( *args: y_test.flatten(), label="Actual Prices", color="blue")
    plt.plot( *args: predicted.flatten(), label="Predicted Prices", color="red")
    plt.title('Actual vs Predicted Stock Prices')
    plt.xlabel('Time')
    plt.ylabel('Price')
    plt.legend()
    plt.show()
```

Result(Graph):



```
Predicted future prices: [[0.10359703 0.10370779 0.10366985 0.10421383 0.10393229]
 [0.10264118 0.10275435 0.10271935 0.10325894 0.10297608]
 [0.10106657 0.10119207 0.10116391 0.10169856 0.10140978]
 [0.09877697 0.09892604 0.09890828 0.09943724 0.09913769]
 [0.09711924 0.09728257 0.09726819 0.09779082 0.09748665]]

Process finished with exit code 0
```

Part 2: Summary of Experiment Results with Multistep and Multivariate Predictions:

We summarize herein results obtained from experimenting with different configurations using the model. First comes the first 50 epochs of training and validation loss, followed by insights gained through observing how this model will be performing over time.

Results of Experimentation (First 50 Epochs):
Epoch 1: Training Loss: 1.0350e-04, Validation Loss: 0.0022

Epoch 2: Training Loss: 1.4137e-05, Validation Loss: 0.0020

Epoch 3: Training Loss: 9.8454e-06, Validation Loss: 0.0040

Epoch 4: Training Loss: 9.8987e-06, Validation Loss: 0.0057

Epoch 5: Training Loss: 1.0419e-05, Validation Loss: 0.0032

Epoch 6: Training Loss: 9.0936e-06, Validation Loss: 0.0031

Epoch 7: Training Loss: 8.9569e-06, Validation Loss: 0.0034

Epoch 8: Training Loss: 7.2587e-06, Validation Loss: 0.0031

Epoch 9: Training Loss: 9.7023e-06, Validation Loss: 0.0020

Epoch 10: Training Loss: 9.3734e-06, Validation Loss: 0.0033

Epoch 11: Training Loss: 7.1694e-06, Validation Loss: 0.0019

Epoch 12: Training Loss: 8.2568e-06, Validation Loss: 0.0031

Epoch 13: Training Loss: 6.9448e-06, Validation Loss: 0.0031

Epoch 14: Training Loss: 7.0709e-06, Validation Loss: 0.0032

Epoch 15: Training Loss: 8.5090e-06, Validation Loss: 5.9624e-04

Epoch 16: Training Loss: 6.1815e-06, Validation Loss: 9.2645e-04

Epoch 17: Training Loss: 6.7724e-06, Validation Loss: 0.0022

Epoch 18: Train Loss : 7.2460e-06 Val Loss : 0.0023

Epoch 19: Train Loss : 6.1043e-06 Val Loss : 0.0017
Epoch 20: Train Loss : 5.7163e-06 Val Loss : 6.5594e-04

Epoch 21: Train Loss : 5.6810e-06 Val Loss : 0.0012

Epoch 22: Train Loss : 6.8236e-06 Val Loss : 5.9871e-04

Epoch 23: Train Loss : 5.5630e-06 Val Loss : 0.0028

Epoch 24: Train Loss : 7.3931e-06 Val Loss : 0.0014

Epoch 25: Train Loss : 6.2646e-06 Val Loss : 0.0028

Epoch 26: Train Loss : 7.2828e-06 Val Loss : 0.0012

Epoch 27: Training Loss: 5.6775e-06, Validation Loss: 0.0011
```
Epoch 28: Training Loss: 4.6008e-06, Validation Loss: 0.0012
Epoch 29: Training Loss: 4.6363e-06, Validation Loss: 0.0015
Epoch 30: Training Loss: 5.9949e-06, Validation Loss: 0.0015
Epoch 31: Training Loss: 5.1967e-06, Validation Loss: 4.1704e-04
Epoch 32: Training Loss: 4.8223e-06, Validation Loss: 0.0024
Epoch 33: Training Loss: 5.1903e-06, Validation Loss: 4.4472e-04
```

Epoch 34: Training Loss: 5.1730e-06, Validation Loss: 0.0017
Epoch 35: Training Loss: 5.7119e-06, Validation Loss: 0.0020

Epoch 36: Training Loss: 5.8305e-06, Valid Loss: 3.3199e-04

Epoch 37: Training Loss: 4.9795e-06, Valid Loss: 7.1205e-04

Epoch 38: Training Loss: 5.7720e-06, Valid Loss: 0.0014

Epoch 39: Training Loss: 6.1714e-06, Valid Loss: 7.1678e-04

Epoch 40: Training Loss: 4.6432e-06, Valid Loss: 0.0012

Epoch 41: Training Loss: 6.5720e-06, Valid Loss: 0.0015

Epoch 42: Training Loss: 5.3937e-06, Valid Loss: 6.4537e-04

Epoch 43: Training Loss: 4.9705e-06, Valid Loss: 3.6478e-04

Epoch 44: Training Loss: 5.2316e-06, Valid Loss: 7.5181e-04

Epoch 45: Train Loss: 4.3132e-06, Val Loss: 3.9186e-04
Epoch 46: Train Loss: 4.7732e-06, Val Loss: 0.0015
Epoch 47: Train Loss: 6.1744e-06, Val Loss: 9.3493e-04
Epoch 48: Train Loss: 5.3203e-06, Val Loss: 8.3886e-04
Epoch 49: Train Loss: 4.8682e-06, Val Loss: 0.0014
Epoch 50: Train Loss: 4.9792e-06, Val Loss: 5.2313e-04
Test Loss: 0.000523128139320761 Predicted Future
Prices:

[[0.09760692, 0.09759455, 0.09842309, 0.09959, 0.09972456],
[0.09688248, 0.09687184, 0.09769063, 0.09884981, 0.09898053],
[0.09672567, 0.09671229, 0.09753387, 0.09868865, 0.09881411],
[0.09678491, 0.09676617, 0.09759547, 0.09874696, 0.09886783], [
0.09645759, 0.0964349, 0.09726448, 0.09840984, 0.0985288]

Observations and Inference:

Performing Better:
From the observation in the 50 epochs, the model continuously reduced both the training and validation loss, thereby learning and adjusting its weights effectively . Fluctuations:

Of course, minor fluctuations in the validation loss were observed between some epochs, for example, between Epochs 15-20 and Epochs 30-35. These are normal when the model is still fine-tuning and then trying to generalize a bit better on the validation data. Validation Loss Stabilization:

At the last epochs, the validation loss started to become more stable; it was at a value of 0.000523. That is usually a good general balance between underfitting and overfitting. This last value of test loss was small, which attests to the fact that the predictions made on unseen test data by this model are mostly correct.

Predicted Future Prices:

Forecasted stock prices for the next 5 days, based on various test samples, all continue with consistent and reasonable outputs. These values are scaled and, therefore, between 0 and 1, but they reflect the trends that this model has learned.

Conclusion:
Overall Performance: In general, it performed very well on both train and validation datasets, especially in terms of its capability related to multistep-ahead forecasting, as well as multivariate.

Future Improvements:

One could still try improving it by tuning with the following: changing the number of units and layers in LSTM; trying different learning rates and optimizers; putting in regularisation techniques such as adding dropout layers so that overfitting of the network can be prevented. From the results of the above, one can surely say that the model has performed well for multistep and multivariate prediction. hese results show that the model was up to the tasks of multistep and multivariate prediction, with its performance remaining high over 50 epochs. It succeeded in predicting future stock prices-achieved the ideal balance between complexity and the use of multiple features, multivariate, while doing predictions many days into the future, multistep.

Future Steps and Improvements:

Hyperparameter tuning: More experimentation with the number of epochs, LSTM units, and layers may provide even better results.

Advanced features: This could be added features of related market indices or perhaps other macroeconomic data to improve the predictions.

Alternative architectures: Comparing some more advanced deep learning models, such as GRU or hybrid models-LSTM combined with CNN-can provide better performances in time series forecasting.

Longer Predictions: Long-term forecasts, such as those 10-20 days ahead, will further challenge the model architecture in terms of its robustness.

Regularization: If the model is unstable when training-that is, if the validation loss is not stable-increase the number of dropout layers or try L2 regularization to prevent overfitting.

Final Remarks:

This therefore concludes that the model indeed accomplished both multistep and multivariate predictions for Task 5, had a performance fairly well both for training and validation data with a final low loss, and hence has proven effective in a time series problem such as stock price forecasting.

Given further tuning, this model can be optimized to better handle more complex timeseries data or even more features.