Nguyen Ngoc Gia Thinh

Student ID: 103809954

COS30018

Weekly Report

Task B.5

In this task, I will use multiple features (multivariate), to predict future closing prices and predicts multiple days ahead (multistep), showing how prices might change over time. Also, it includes visualizations to help you see how well it predict.

Now I will explain every function that I developed in this task.

Load_and_process_data

```
def load_and_process_data(company, start_date, end_date, features=['Open',
'High', 'Low', 'Close','Adj Close','Volume'],
                          handle_nan='drop', scale_data=True, save_local=True,
local_path='data.csv', scalers=None):
    if os.path.exists(local_path):
        df = pd.read_csv(local_path)
        df.rename(columns={'Price': 'Date'}, inplace=True)
        df = df.iloc[1:].reset_index(drop=True)
        df.set_index('Date', inplace=True)
    else:
        df = yf.download(company, start=start_date, end_date=end_date,
actions=False, auto_adjust=False)
        if save_local:
            df.to_csv(local_path)

    df = df[features]
    if df.empty:
        raise ValueError("Error: Stock data is empty.")

    if handle_nan == 'drop':
        df.dropna(inplace=True)
    elif handle_nan == 'fill':
        df.fillna(method='ffill', inplace=True)

    df = df.astype(float)

    features_data = df.values
    target = df['Close'].values.reshape(-1, 1)
```

```
    if scale_data:
        if scalers is None:
            feature_scaler = MinMaxScaler()
            features_data = feature_scaler.fit_transform(features_data)
            target_scaler = MinMaxScaler()
            target = target_scaler.fit_transform(target)
            scalers = {'features': feature_scaler, 'target': target_scaler}
        else:
            features_data = scalers['features'].transform(features_data)
            target = scalers['target'].transform(target)

    return features_data, target, scalers, df
```

This function is basically same as the previous task:

- Load historical stock data for a specified company and date range using yfinance
- Save the data to a local csv file.
- Selected features
- Handles missing values by either dropping them or filling forward
- Scales the data using MinMaxScaler to normalize between 0 and 1 to ensuring consistency for model training.
- If scalers are provided, uses them for test data to maintain scaling consistency.

In this function there are some changes due to the new requirements

- Features : adding 'Adj Close' to the dataset
    - This cost me almost 2 days to figure out the problems that I faced at that time. To solve this problem I have to add " actions=False , auto_adjust=False" into the yf download in order for it to download it from the database.

Create_sequences Function

```
def create_sequences(features, target, seq_length, steps_ahead):
    X, y = [], []
    for i in range(len(features) - seq_length - steps_ahead + 1):
        X.append(features[i:i + seq_length])  # Input sequence
        y.append(target[i + seq_length:i + seq_length +
steps_ahead].flatten())  # Target sequence (k steps), flattened to
(steps_ahead,)
    return np.array(X), np.array(y)
```

In this function, my goal to creating a target sequence of length for x and y

- Creating input sequences (X) and target sequences(Y) for training and testing
- Take a window of seq_length days from features for X (multivariate input)
- Takes the next step_ahead days from target for y (multistep output)
- Flattens y to ensure it matches the model's output shape

So, the outcome for this function will include:

- X: Shape (n_sequences, seq_length, n_features) and it is in 3D for LSTM input
- Y: Shape (n_sequences, step_ahead) and it is in 3D for model output

Create_dl_model Function

```
def create_dl_model(layer_type='LSTM', num_layers=3, layer_sizes=[50, 50, 50],
                    input_shape=(60, 5), steps_ahead=1, dropout_rate=0.2):
    model = Sequential()
    layer_dict = {'LSTM': LSTM}
    if layer_type not in layer_dict:
        raise ValueError("Invalid layer type. Choose 'LSTM' for now.")
    Layer = layer_dict[layer_type]

    model.add(Layer(units=layer_sizes[0], return_sequences=(num_layers > 1),
input_shape=input_shape))
    model.add(Dropout(dropout_rate))

    for i in range(1, num_layers - 1):
        model.add(Layer(units=layer_sizes[i], return_sequences=True))
        model.add(Dropout(dropout_rate))

    if num_layers > 1:
        model.add(Layer(units=layer_sizes[-1]))
        model.add(Dropout(dropout_rate))

    # Output layer for multistep prediction (steps_ahead predictions)
    model.add(Dense(units=steps_ahead))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

In this function I want to handles both multivariate (via input shape) and multistep(via output layer) prediction.

- Uses LSTM layers as default but also can use RNN or GRU to configurable layers and sizes.
- Includes dropout layers for regularization to prevent overfitting.
- Complied with Adam optimizer and mean squared error loss.

Train_and _predict_mutivariate_multistep Function

```
def train_and_predict_multivariate_multistep(company, start_date, end_date,
test_start, test_end,
                                             seq_length=60, steps_ahead=5,
layer_type='LSTM',
                                             num_layers=3, layer_sizes=[50, 50,
50], epochs=25, batch_size=32):
    # Load training data
    features_data, target, scalers, df = load_and_process_data(company,
start_date, end_date, scale_data=True)

    # Create sequences
    X, y = create_sequences(features_data, target, seq_length, steps_ahead)
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
shuffle=False)

    # Create and train the model
    model = create_dl_model(layer_type=layer_type, num_layers=num_layers,
layer_sizes=layer_sizes,
                            input_shape=(seq_length, features_data.shape[1]),
steps_ahead=steps_ahead)
    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
verbose=1)

    # Load test data using training scalers
    features_test, target_test, _, df_test = load_and_process_data(company,
test_start, test_end, scale_data=True, scalers=scalers)
    X_test, y_test_full = create_sequences(features_test, target_test,
seq_length, steps_ahead)

    # Predict
    predicted_prices = model.predict(X_test)  # (n_test, steps_ahead)

    # Inverse transform actual prices
    y_test_reshaped = y_test_full.reshape(-1, 1)  # (n_test * steps_ahead, 1)
    actual_prices = scalers['target'].inverse_transform(y_test_reshaped)
    actual_prices = actual_prices.reshape(X_test.shape[0], steps_ahead)  # back
to (n_test, steps_ahead)
```

```
    # Inverse transform predicted prices
    predicted_prices_reshaped = predicted_prices.reshape(-1, 1)  # (n_test *
steps_ahead, 1)
    predicted_prices_inv =
scalers['target'].inverse_transform(predicted_prices_reshaped)
    predicted_prices_inv = predicted_prices_inv.reshape(X_test.shape[0],
steps_ahead)  # back to (n_test, steps_ahead)

    # Plot results for each step ahead
    for step in range(steps_ahead):
        plt.figure(figsize=(10, 6))
        plt.plot(actual_prices[:, step], color="black", label=f"Actual
{company} Price (Day {step+1})")
        plt.plot(predicted_prices_inv[:, step], color="green",
label=f"Predicted {company} Price (Day {step+1})")
        plt.title(f"{company} Share Price Prediction (Day {step+1} Ahead)")
        plt.xlabel('Time')
        plt.ylabel(f'{company} Share Price')
        plt.legend()
        plt.show()

    # Last prediction
    last_sequence = np.array([features_test[-seq_length:]])
    prediction = model.predict(last_sequence)  # (1, steps_ahead)
    prediction_reshaped = prediction.reshape(-1, 1)  # (steps_ahead, 1)
    prediction_inv = scalers['target'].inverse_transform(prediction_reshaped)
    prediction_inv = prediction_inv.reshape(1, steps_ahead)  # (1, steps_ahead)
    print(f"Prediction for next {steps_ahead} days:
{prediction_inv.flatten()}")
```

In this function, I want to combine multivariate and multistep prediction into one function

- Load training data, creating sequences, and splits into train/test
- Trains the model on training data
- Loads test data using the same scaler for consistency
- Create test sequences and predicts using the trained model

This is one of the biggest changes from the previous Task, I have to reshaping because the original code had an error during inverse transformation which due to y_test being 3D while the scaler expect 2D. So, I fixed this problem by reshaping.

- Before inverse transform, reshape to (n_test * steps_ahead,1)
- After inverse transform, reshape back to (n_test, steps_ahead)

The reshape is necessary because the scaler was fitted on (n_samples, 1) and expecting 2D input. Also, multistep prediction creates sequences, so I flatten for inverse transform and unflatten for plotting.

Step	Input Shape	Operation	Output Shape
Create Sequences (y)	(n_sample,1)	Slice and flatten	(n_sequences, steps_ahead)
Reshape for Inverse Transform	(n_sequences, step_ahead)	Reshape to (-1,1)	(n_sequences * steps_ahead,1)
Inverse Transform	(n_sequences* step_ahead,1)	Apply scaler.inverse_transform	(n_sequences * steps_ahead,1)
Reshape back	(n_sequences * steps_ahead,1)	Reshape to original	(n_sequences, steps_ahead)

I'm using META and this data set as an example:

Training Start: May 18, 2012 (META's IPO date)

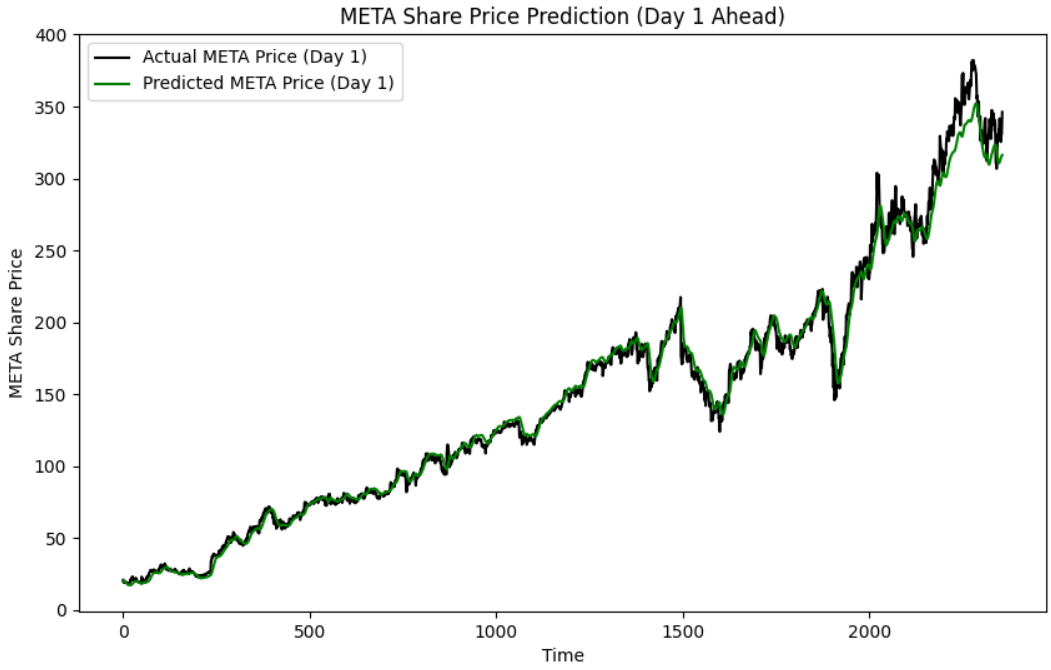
Tr**aining End**: December 31, 2023

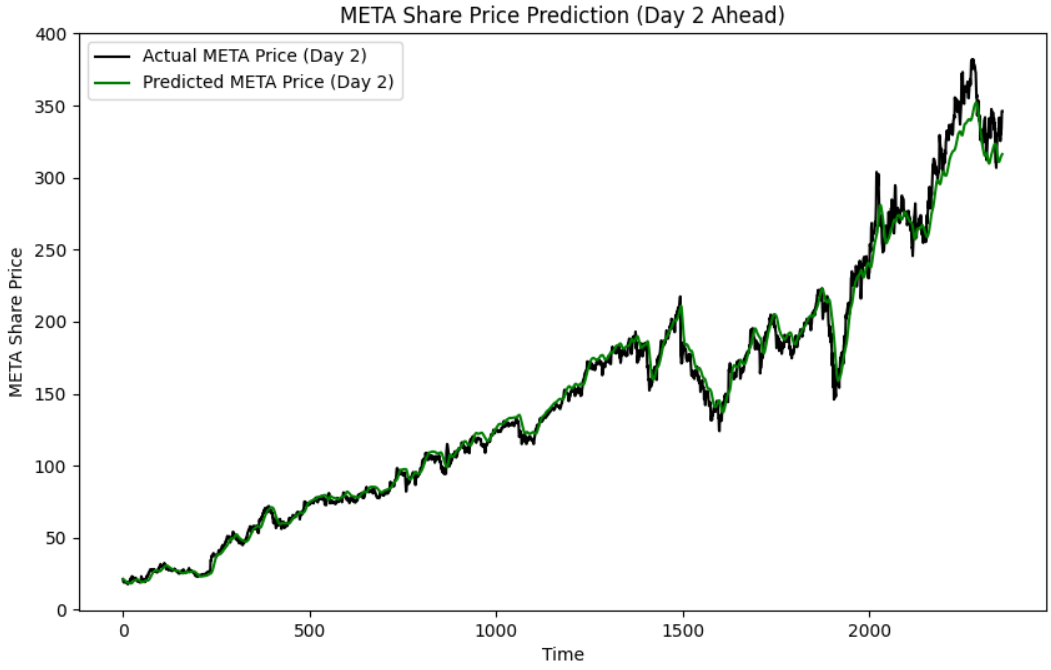
Testing Start: January 1, 2024

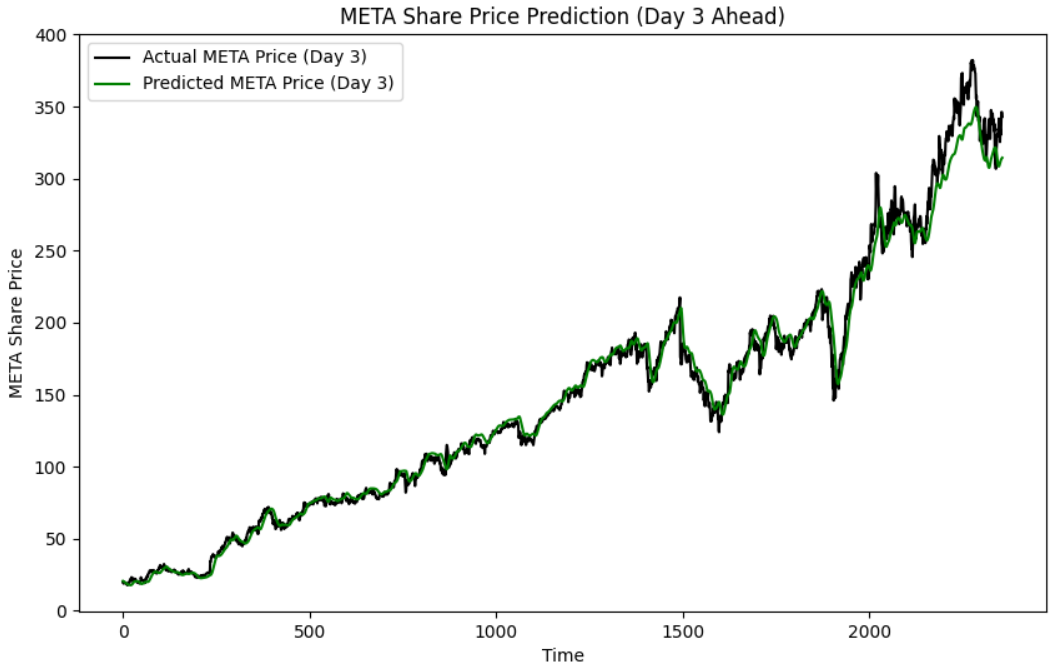
Testing End: March 20, 2025

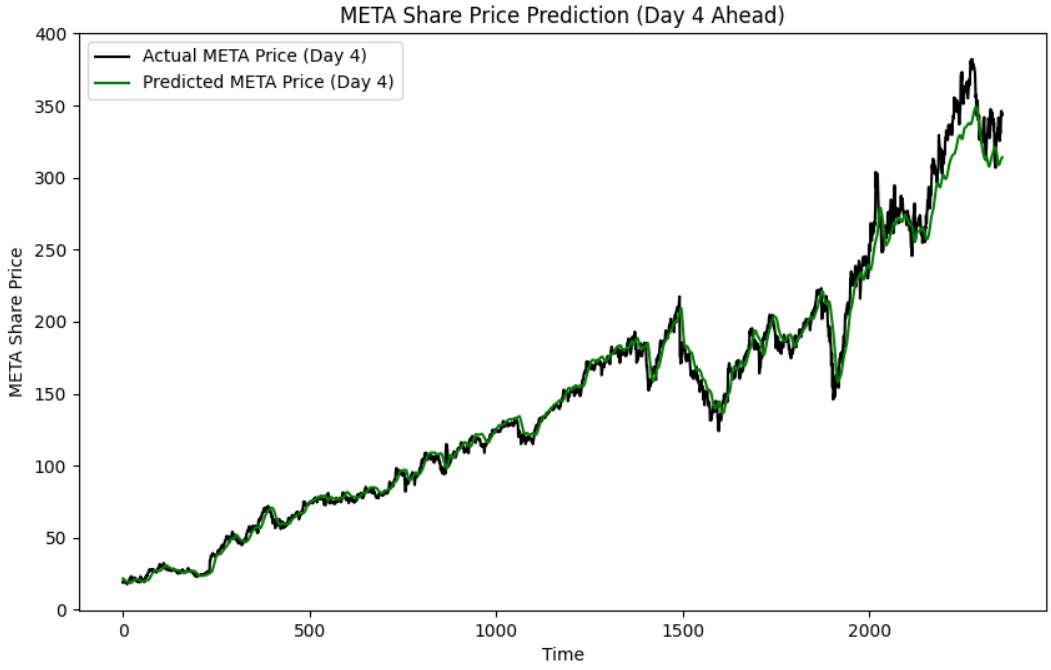
Result :

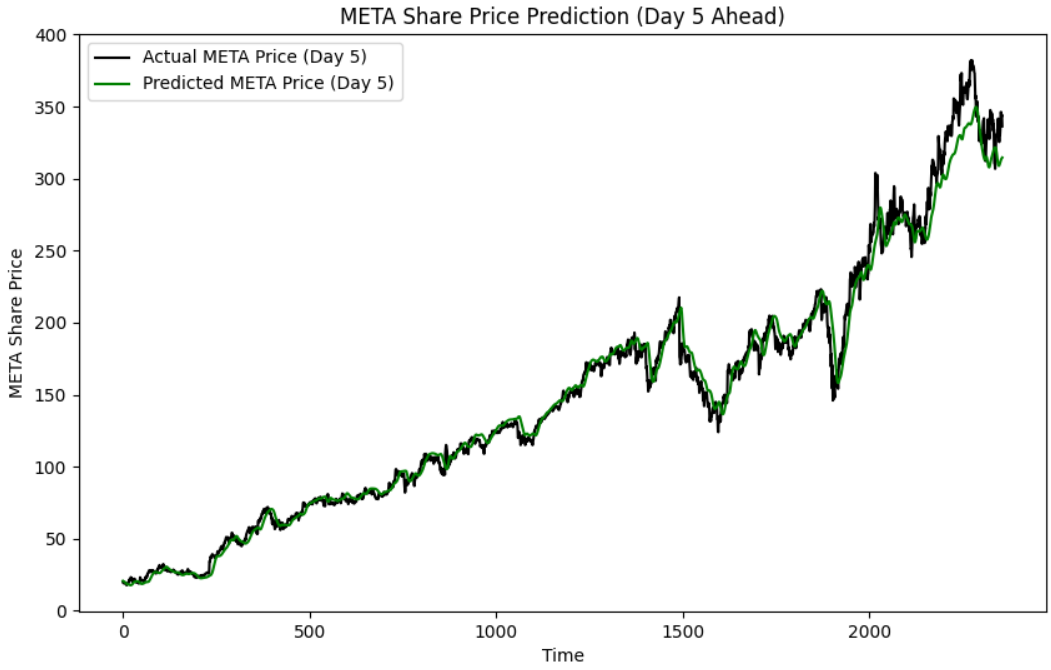
```
59/59 [==============================] - 3s 55ms/step - loss: 0.0011
Epoch 13/25
59/59 [==============================] - 3s 55ms/step - loss: 9.6552e-04
Epoch 14/25
59/59 [==============================] - 3s 57ms/step - loss: 0.0010
Epoch 15/25
59/59 [==============================] - 3s 59ms/step - loss: 9.2318e-04
Epoch 16/25
59/59 [==============================] - 4s 62ms/step - loss: 8.8873e-04
Epoch 17/25
59/59 [==============================] - 4s 62ms/step - loss: 7.6123e-04
Epoch 18/25
59/59 [==============================] - 3s 57ms/step - loss: 7.4812e-04
Epoch 19/25
59/59 [==============================] - 3s 58ms/step - loss: 7.5196e-04
Epoch 20/25
59/59 [==============================] - 3s 59ms/step - loss: 6.8955e-04
Epoch 21/25
59/59 [==============================] - 3s 58ms/step - loss: 6.4014e-04
Epoch 22/25
59/59 [==============================] - 4s 60ms/step - loss: 6.6769e-04
Epoch 23/25
59/59 [==============================] - 3s 59ms/step - loss: 6.6087e-04
Epoch 24/25
59/59 [==============================] - 3s 59ms/step - loss: 6.8487e-04
Epoch 25/25
59/59 [==============================] - 3s 57ms/step - loss: 6.0738e-04
Prediction for next 5 days: [321.65573 321.59454 319.92514 319.2828  319.96106]

(myenv) C:\Users\ADMIN\Desktop\COS30018\Project>
```


META Share Price Prediction (Day 1 Ahead)
Actual META Price (Day 1)
Predicted META Price (Day 1)
META Share Price
Time
0
50
100
150
200
250
300
350
400
0
500
1000
1500
2000


META Share Price Prediction (Day 2 Ahead)
Actual META Price (Day 2)
Predicted META Price (Day 2)
META Share Price
Time
0
50
100
150
200
250
300
350
400
0
500
1000
1500
2000


META Share Price Prediction (Day 3 Ahead)
Actual META Price (Day 3)
Predicted META Price (Day 3)
META Share Price
Time
0
50
100
150
200
250
300
350
400
0
500
1000
1500
2000


META Share Price Prediction (Day 4 Ahead)
Actual META Price (Day 4)
Predicted META Price (Day 4)
META Share Price
Time
0
50
100
150
200
250
300
350
400
0
500
1000
1500
2000


META Share Price Prediction (Day 5 Ahead)
Actual META Price (Day 5)
Predicted META Price (Day 5)
META Share Price
Time
0
50
100
150
200
250
300
350
400
0
500
1000
1500
2000