Week 5 Attempt Summary

This week we were tasked with moving from our v3 code base to the v4 code base

In this task we create a multivariate sequence input and a multistep prediction function to take advantage of the multivariate sequence input to solve the multivariate prediction problem.

First, we define the hyperparameters needed for this multivariate sequence, which is

```
# Parameters setup
N_STEPS = 60
LOOKUP_STEP = 15
SCALE = True
SHUFFLE = True
SPLIT_BY_DATE = False
TEST_SIZE = 0.2
FEATURE_COLUMNS = ["Adj Close", "Volume", "Open", "High", "Low"]
```

In our last code even if I had the features column, only the Adj Close was used for the input sequence but in this version of the code we use all the above features to create a multivariate sequence

- Adj Close – Adjusted Closing price
- Volume
- Open – Opening price
- High – Highest Price
- Low – Lowest Price

Using all the features in our disposal makes the prediction process more sophisticated by leveraging additional information, which can help the model make more accurate predictions compared to using only a single feature (closing price)

in the load and process function where we scale the data, we scale the adj close data separately because we do the prediction on that

```
column_scaler = {}
    if scale:
        scaler = MinMaxScaler()
        df[df.columns] = scaler.fit_transform(df[df.columns])
        column_scaler = {col: scaler for col in df.columns}

        adjclose_scaler = MinMaxScaler()
        df['Adj Close'] = adjclose_scaler.fit_transform(df[['Adj
Close']])
        column_scaler['Adj Close'] = adjclose_scaler
```

In the multivariate sequence function,

```
# Create multivariate sequences
def create_multivariate_sequences(data, feature_columns, n_steps):
    X, y = [], []
    for i in range(len(data) - n_steps):
        X.append(data[feature_columns].iloc[i:i + n_steps].values)
        y.append(data['Adj Close'].iloc[i + n_steps])
    return np.array(X), np.array(y)
```

The function takes in 3 arguments – the dataset, feature list and the n_step

First, we create 2 initially arrays X and Y, then we continue into a for loop, The X array is being appended by all the features in the feature column

```
# Parameters setup
N_STEPS = 60
LOOKUP_STEP = 15
SCALE = True
SHUFFLE = True
SPLIT_BY_DATE = False
TEST_SIZE = 0.2
FEATURE_COLUMNS = ["Adj Close", "Volume", "Open", "High", "Low"]
```

And creates sequences of these features for each step in the time series.

Then Y (the target Variable)

The function uses only the 'Adj Close' as the target variable for prediction. So while the model uses multiple features for input, it's only taking this specific variable to predict the closing price for feature steps.

```
 for i in range(len(data) - n_steps):
        X.append(data[feature_columns].iloc[i:i + n_steps].values)
        y.append(data['Adj Close'].iloc[i + n_steps])
```

Then We move on to the Multistep prediction function where we combine both functions,

```
def multistep_predict(model, data, n_steps, k):
    last_sequence = data['test'][FEATURE_COLUMNS].values[-n_steps:]
    last_sequence = last_sequence.reshape((1, n_steps,
len(FEATURE_COLUMNS)))
    predictions = []

    for step in range(k):
        prediction = model.predict(last_sequence)
        prediction = prediction.reshape((1, 1, 1))

        if SCALE:
            prediction = data['column_scaler']['Adj
Close'].inverse_transform(prediction.reshape(-1, 1)).reshape(1, 1, 1)

        predictions.append(prediction[0][0][0])

        new_sequence = np.copy(last_sequence)
        new_sequence[:, -1, 0] = prediction[0, 0, 0]
        last_sequence = np.append(new_sequence[:, 1:, :],
new_sequence[:, -1:, :], axis=1)

    return predictions
```

First this function takes in 4 arguments,

Model – The trained learning model (LSTM, GRU)

Data – The data dictionary containing test data, scalers and all the features

n_step – the number of time steps the model uses as input

k – The number of steps or days into the future the model needs to predict

The function extracts the last n_step of data from the test set including all the features in the feature column array.

And after the extraction this sequence is reshaped into the expected shape by the model.

```
last_sequence = data['test'][FEATURE_COLUMNS].values[-n_steps:]
    last_sequence = last_sequence.reshape((1, n_steps,
len(FEATURE_COLUMNS)))
```

Then we iterate through for each k step with a for loop

```
for step in range(k):
        prediction = model.predict(last_sequence)
        prediction = prediction.reshape((1, 1, 1))

        if SCALE:
            prediction = data['column_scaler']['Adj
Close'].inverse_transform(prediction.reshape(-1, 1)).reshape(1, 1, 1)

        predictions.append(prediction[0][0][0])

        new_sequence = np.copy(last_sequence)
        new_sequence[:, -1, 0] = prediction[0, 0, 0]
        last_sequence = np.append(new_sequence[:, 1:, :],
new_sequence[:, -1:, :], axis=1)
```

For each future step the model predicts the next value based on the last sequence.

Then do a inverse scaling to the data, as the data was scaled during preprocessing, Using the MinMaxScaler, the prediction needs to be rescaled back to it's original stock price value. As we only use 'Adj close' for prediction the rescaling will only apply to that value along.

```
if SCALE:
            prediction = data['column_scaler']['Adj
Close'].inverse_transform(prediction.reshape(-1, 1)).reshape(1, 1, 1)
```

Then We update the sequence for the next step.

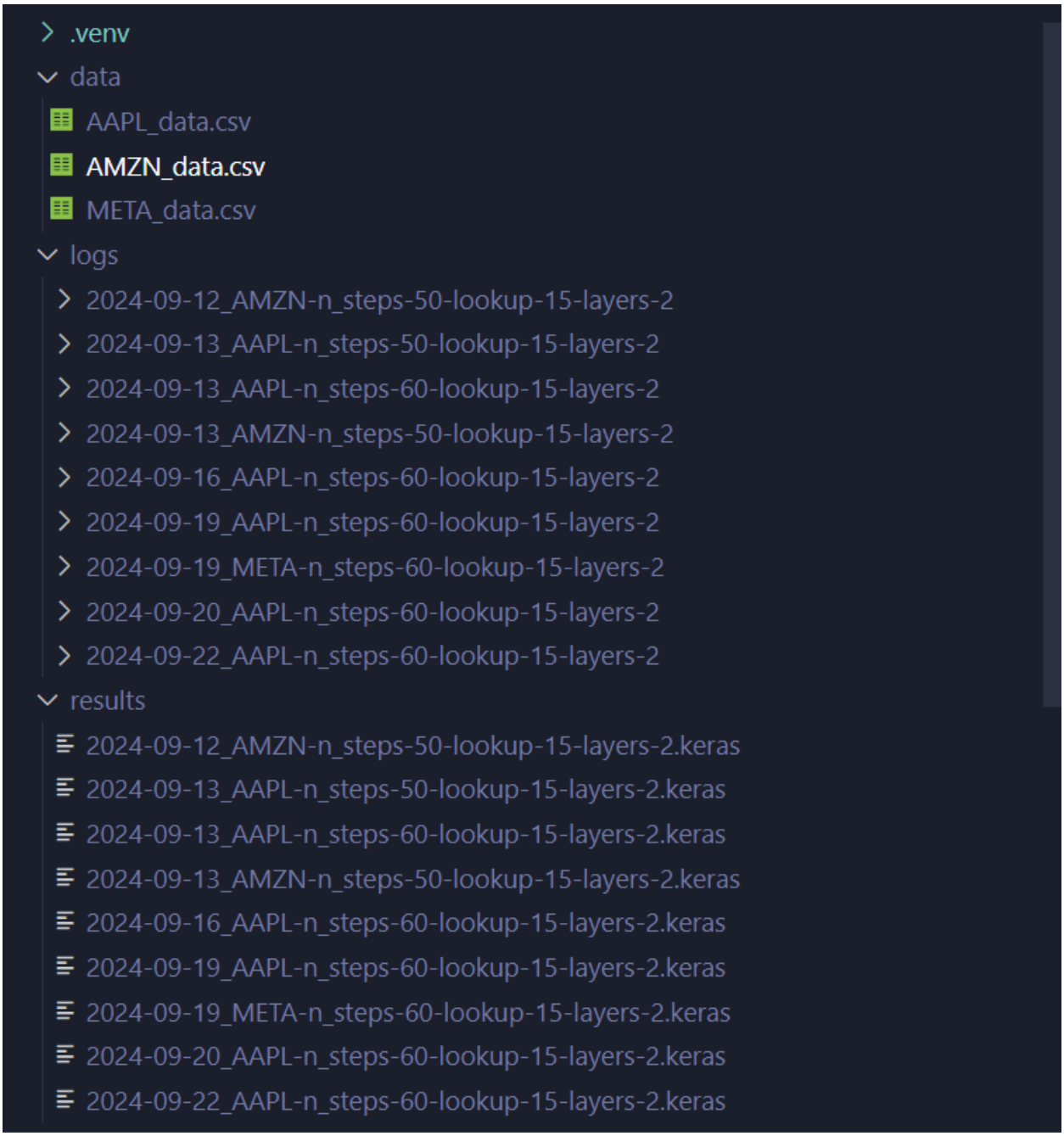
```
predictions.append(prediction[0][0][0])

        new_sequence = np.copy(last_sequence)
        new_sequence[:, -1, 0] = prediction[0, 0, 0]
        last_sequence = np.append(new_sequence[:, 1:, :],
new_sequence[:, -1:, :], axis=1)
```

the new_sequence is a copy of the **current** last_sequence. The new predicted value replaces the last value in the sequence for the 'Adj Close' feature. Then the updated sequence is shifted by removing the oldest time step and appending the new predicted value. This process prepares the last_sequence for the next prediction.

Results

I have tested this with 3 ticker symbols,

- AAPL
- AMZN
- META



AAPL

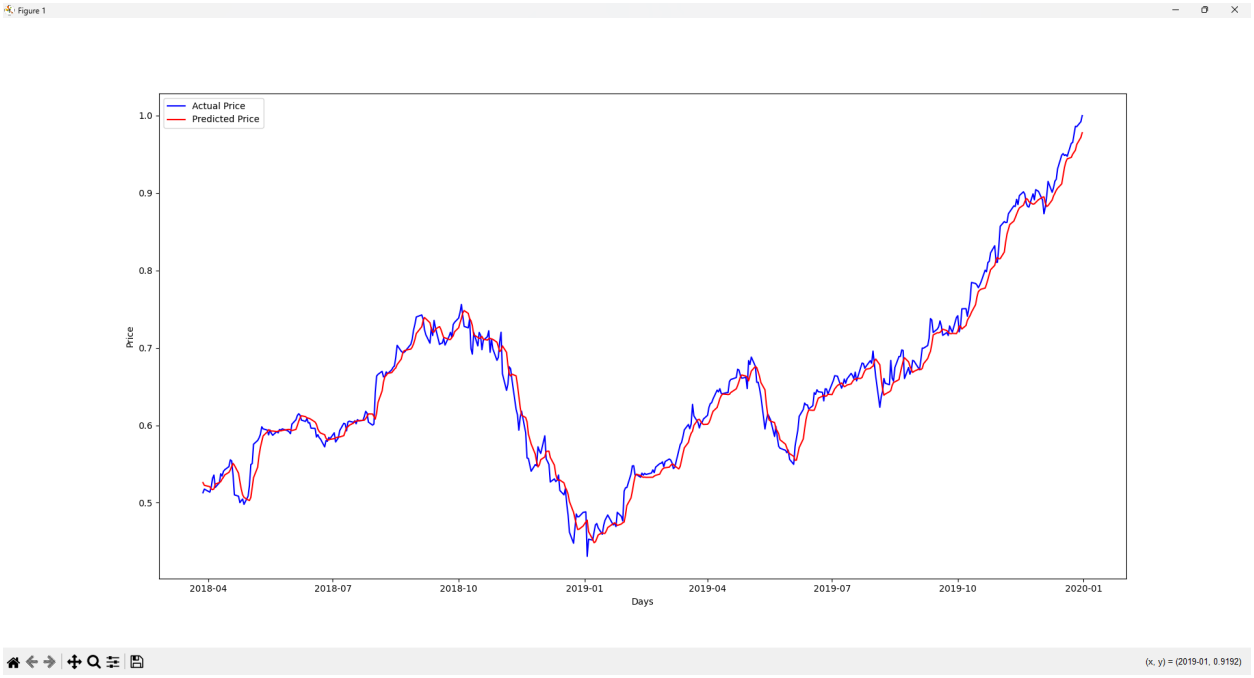


Figure 1
Actual Price
Predicted Price
Price
Days
1.0
0.9
0.8
0.7
0.6
0.5
2018-04
2018-07
2018-10
2019-01
2019-04
2019-07
2019-10
2020-01
(x, y) = (2019-01, 0.9192)

AMZN


Figure 1
Actual Price
Predicted Price
Price
Days
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
2018-04
2018-07
2018-10
2019-01
2019-04
2019-07
2019-10
2020-01
(x, y) = (2019-10, 0.7897)

META


Figure 1
Actual Price
Predicted Price
0.9
0.8
0.7
0.6
Price
2018-09
2018-11
2019-01
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
Days
(x, y) = (2019-04, 0.7521)