# COS30018 - OPTION B - TASK 5: MACHINE LEARNING 2

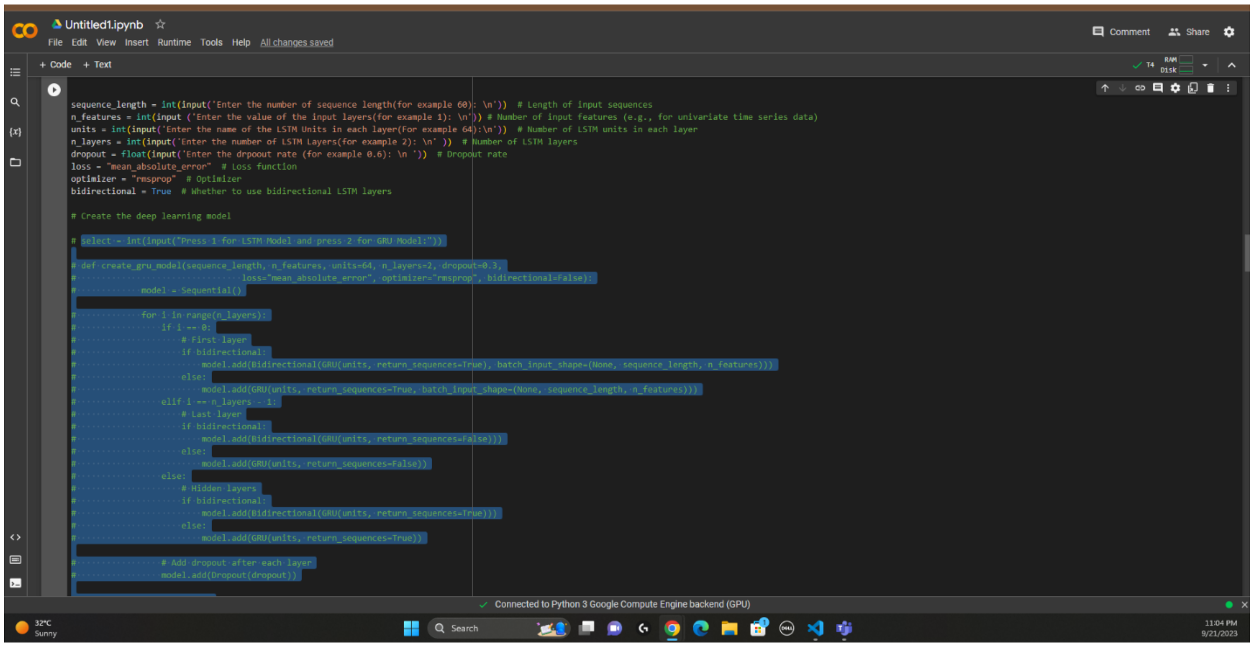
-VIVEK SAINI

In this assignment I will be talking about the models I have created  for the purpose of machine learning. Starting off with the first requirement:

*Implement a function that solve the multistep prediction problem to allow the prediction to be made for a sequence of closing prices of k days into the future*

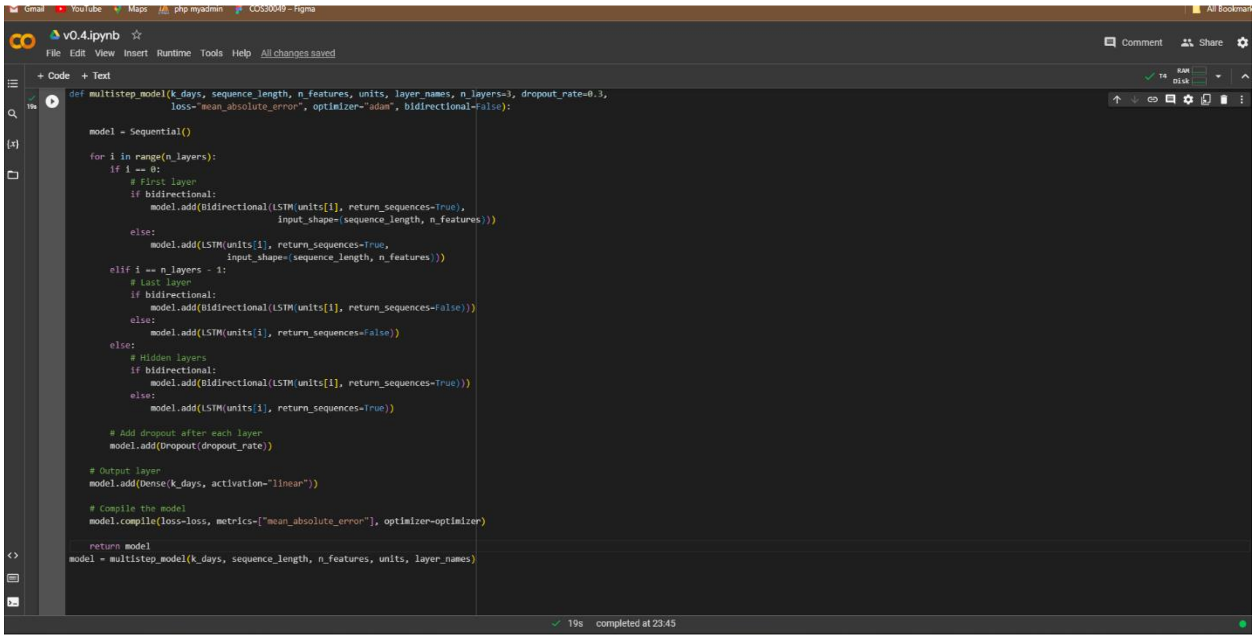
First of all I have made a few tweaks before starting this assignment. I have commented out the box plot and other graphs because it was unnecessary. I have commented out the previous model and removed the input function that takes hyperparameters into context because it also had no use and was time consuming. Instead I have hard coded certain values.

First lets talk about the multistep Prediction model:



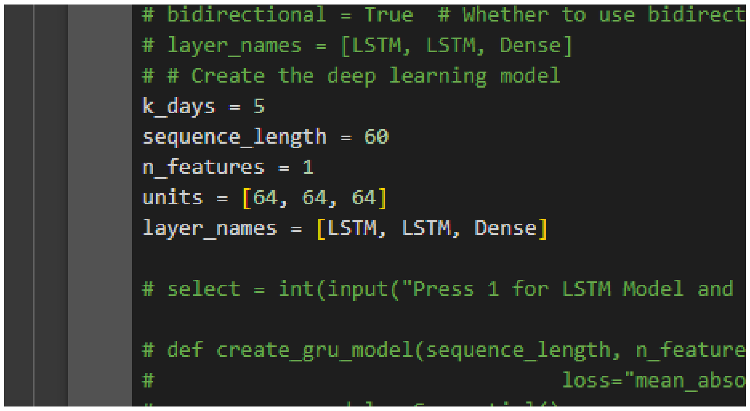
```python
sequence_length = int(input('Enter the number of sequence length(for example 60): \n'))  # Length of input sequences
n_features = int(input ('Enter the value of the input layers(for example 1): \n')) # Number of input features (e.g., for univariate time series data)
units = int(input('Enter the name of the LSTM Units in each layer(For example 64):\n'))  # Number of LSTM units in each layer
n_layers = int(input('Enter the number of LSTM Layers(for example 2): \n' ))  # Number of LSTM layers
dropout = float(input('Enter the drpoout rate (for example 0.6): \n '))  # Dropout rate
loss = "mean_absolute_error"  # Loss function
optimizer = "rmsprop"  # Optimizer
bidirectional = True  # Whether to use bidirectional LSTM layers

# Create the deep learning model

# select = int(input("Press 1 for LSTM Model and press 2 for GRU Model:"))

# def create_gru_model(sequence_length, n_features, units=64, n_layers=2, dropout=0.3,
#                                  loss="mean_absolute_error", optimizer="rmsprop", bidirectional=False):
#             model = Sequential()

#             for i in range(n_layers):
#                 if i == 0:
#                     # First layer
#                     if bidirectional:
#                         model.add(Bidirectional(GRU(units, return_sequences=True), batch_input_shape=(None, sequence_length, n_features)))
#                     else:
#                         model.add(GRU(units, return_sequences=True, batch_input_shape=(None, sequence_length, n_features)))
#                 elif i == n_layers - 1:
#                     # Last layer
#                     if bidirectional:
#                         model.add(Bidirectional(GRU(units, return_sequences=False)))
#                     else:
#                         model.add(GRU(units, return_sequences=False))
#                 else:
#                     # Hidden layers
#                     if bidirectional:
#                         model.add(Bidirectional(GRU(units, return_sequences=True)))
#                     else:
#                         model.add(GRU(units, return_sequences=True))

#                 # Add dropout after each layer
#                 model.add(Dropout(dropout))
```

Also I have switched to google colab to perform this assignment.



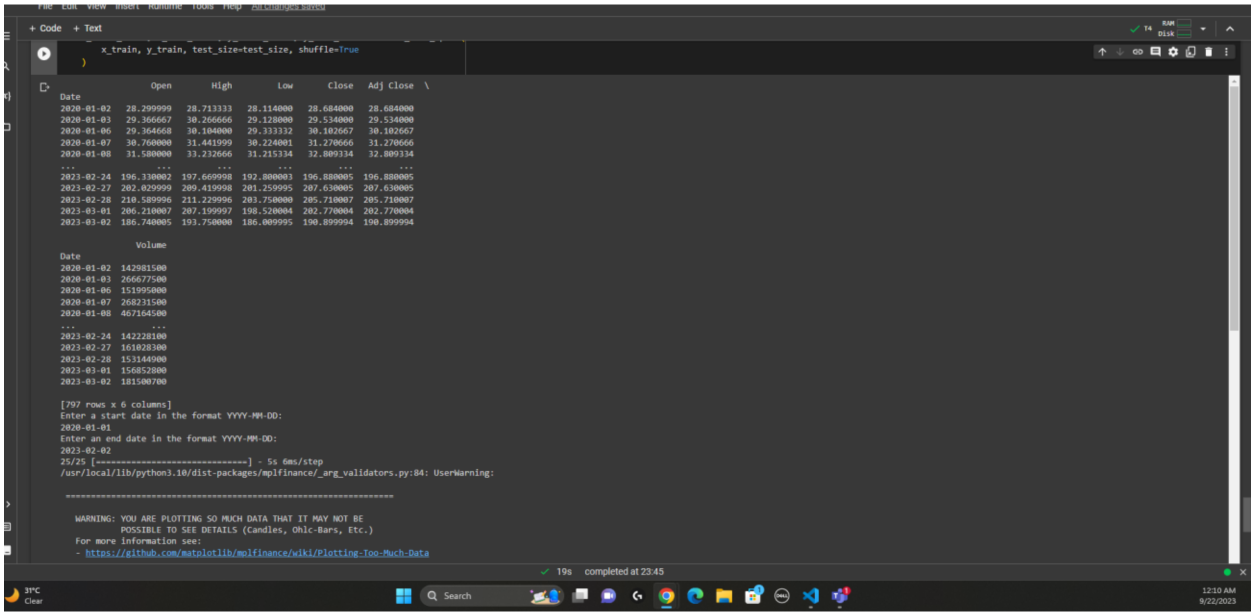
```python
def multistep_model(k_days, sequence_length, n_features, units, layer_names, n_layers=3, dropout_rate=0.3,
                    loss="mean_absolute_error", optimizer="adam", bidirectional=False):

    model = Sequential()

    for i in range(n_layers):
        if i == 0:
            # First layer
            if bidirectional:
                model.add(Bidirectional(LSTM(units[i], return_sequences=True),
                                        input_shape=(sequence_length, n_features)))
            else:
                model.add(LSTM(units[i], return_sequences=True,
                               input_shape=(sequence_length, n_features)))
        elif i == n_layers - 1:
            # Last layer
            if bidirectional:
                model.add(Bidirectional(LSTM(units[i], return_sequences=False)))
            else:
                model.add(LSTM(units[i], return_sequences=False))
        else:
            # Hidden layers
            if bidirectional:
                model.add(Bidirectional(LSTM(units[i], return_sequences=True)))
            else:
                model.add(LSTM(units[i], return_sequences=True))

        # Add dropout after each layer
        model.add(Dropout(dropout_rate))

    # Output layer
    model.add(Dense(k_days, activation="linear"))

    # Compile the model
    model.compile(loss=loss, metrics=["mean_absolute_error"], optimizer=optimizer)

    return model
model = multistep_model(k_days, sequence_length, n_features, units, layer_names)
```

In the multistep model we define certain variables like k_days, which tells us how many days we will be predicting into the future and we can set this value later in this function. Then I have set the rest of the variables which were similar to the previous LSTM model. It was pretty straightforward till this step.



```
# bidirectional = True  # Whether to use bidirect
# layer_names = [LSTM, LSTM, Dense]
# # Create the deep learning model
k_days = 5
sequence_length = 60
n_features = 1
units = [64, 64, 64]
layer_names = [LSTM, LSTM, Dense]

# select = int(input("Press 1 for LSTM Model and

# def create_gru_model(sequence_length, n_feature
#                                loss="mean_abso
```

Then I have given some values to the parameters and then passed it into the function.

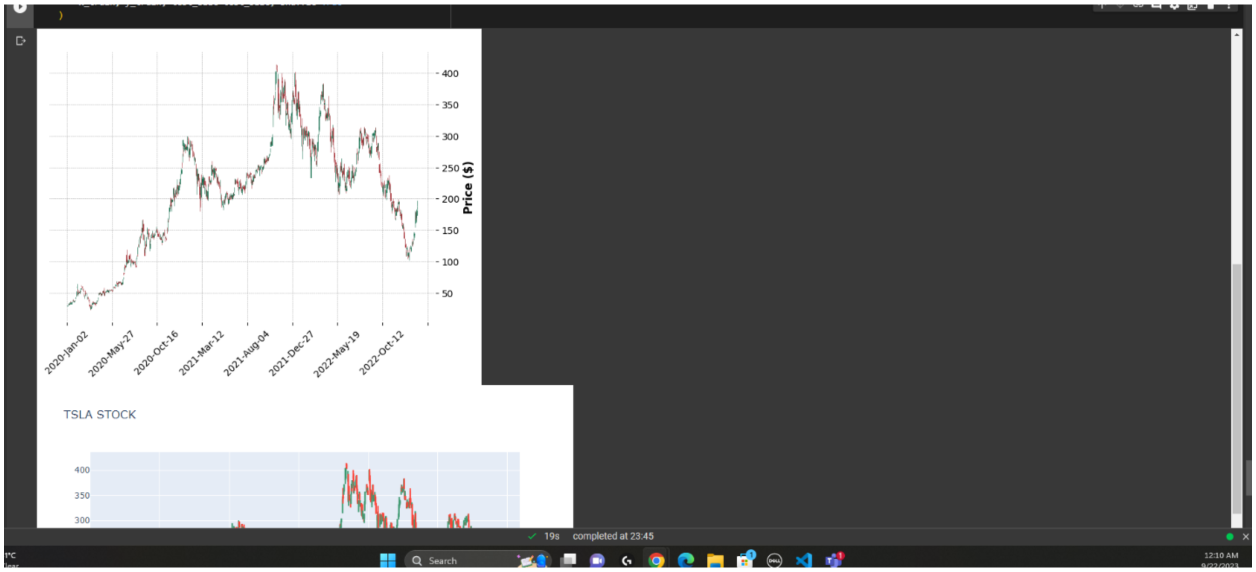


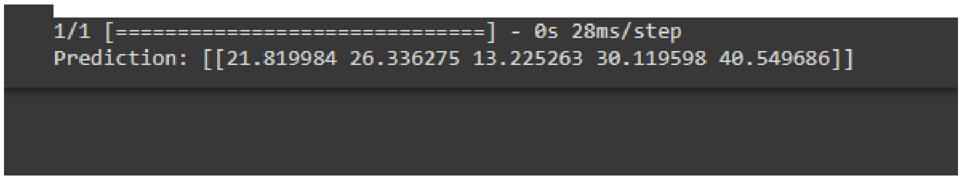
```
        x_train, y_train, test_size=test_size, shuffle=True
    )
```

```
                  Open        High         Low       Close   Adj Close  \
Date
2020-01-02   28.299999   28.713333   28.114000   28.684000   28.684000
2020-01-03   29.366667   30.266666   29.128000   29.534000   29.534000
2020-01-06   29.364668   30.104000   29.333332   30.102667   30.102667
2020-01-07   30.760000   31.441999   30.224001   31.270666   31.270666
2020-01-08   31.580000   33.232666   31.215334   32.809334   32.809334
...                ...         ...         ...         ...         ...
2023-02-24  196.330002  197.669998  192.800003  196.880005  196.880005
2023-02-27  202.029999  209.419998  201.259995  207.630005  207.630005
2023-02-28  210.589996  211.229996  203.750000  205.710007  205.710007
2023-03-01  206.210007  207.199997  198.520004  202.770004  202.770004
2023-03-02  186.740005  193.750000  186.009995  190.899994  190.899994

               Volume
Date
2020-01-02  142981500
2020-01-03  266677500
2020-01-06  151995000
2020-01-07  268231500
2020-01-08  467164500
...               ...
2023-02-24  142228100
2023-02-27  161028300
2023-02-28  153144900
2023-03-01  156852800
2023-03-02  181500700

[797 rows x 6 columns]
Enter a start date in the format YYYY-MM-DD:
2020-01-01
Enter an end date in the format YYYY-MM-DD:
2023-02-02
25/25 [==============================] - 5s 6ms/step
/usr/local/lib/python3.10/dist-packages/mplfinance/_arg_validators.py:84: UserWarning:

 ================================================================

   WARNING: YOU ARE PLOTTING SO MUCH DATA THAT IT MAY NOT BE
            POSSIBLE TO SEE DETAILS (Candles, Ohlc-Bars, Etc.)
   For more information see:
   - https://github.com/matplotlib/mplfinance/wiki/Plotting-Too-Much-Data
```

In this model we can see that it uses the data that I have defined in the very beginning of the code and loads it here according to the date range that we input.

As you can see the results of this function in the picture above, here is the charts of this function:





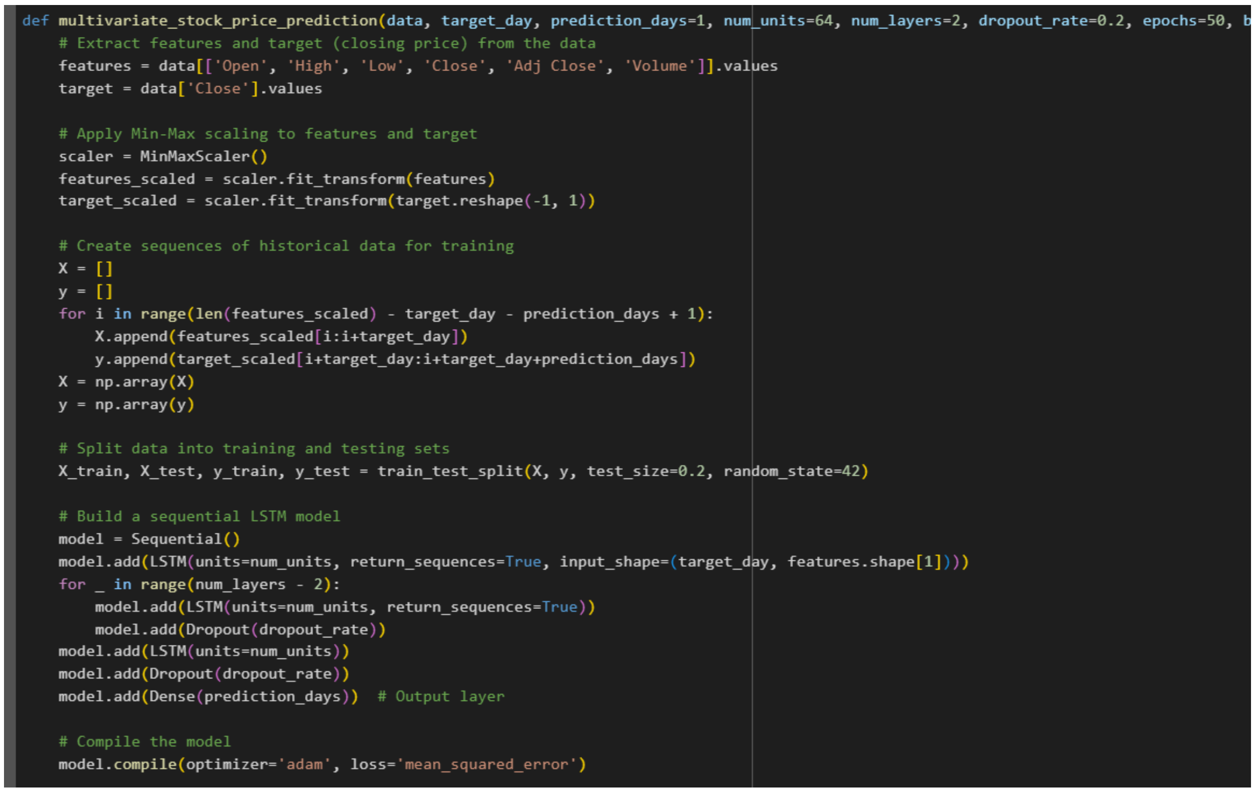
```
1/1 [==============================] - 0s 28ms/step
Prediction: [[21.819984 26.336275 13.225263 30.119598 40.549686]]
```

Moving on to the next step,

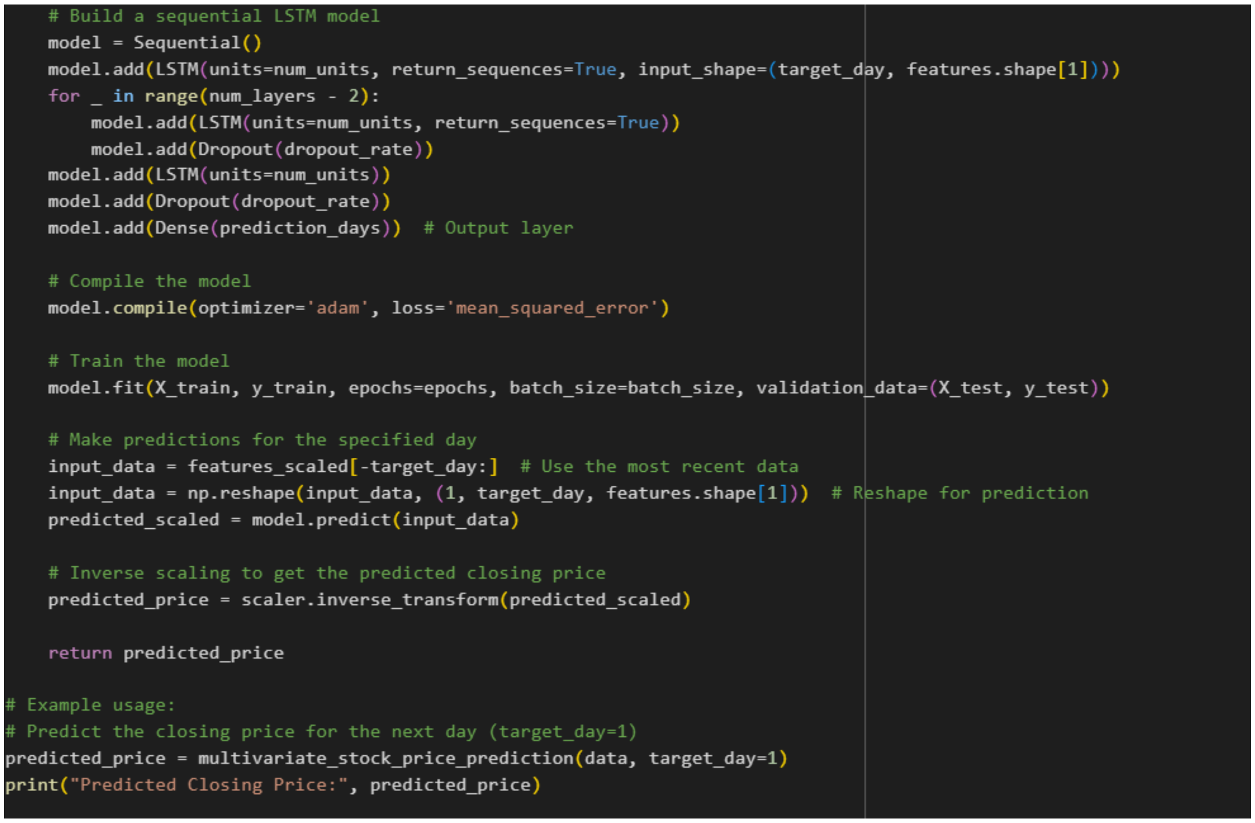
*Implement a function that solve the simple multivariate prediction problem to that takes into account the other features for the same company (including opening price, highest price, lowest price, closing price, adjusted closing price, trading volume) as the input for predicting the closing price of the company for a specified day in the future.*

Here things get more complicated, this function took too long to make but in the end I figured out a way. Firstly, I was trying to make some changes in the existing model by changing the parameters and Creating an array including all the Features. And then passing into the function but it did not work because I was doing something wrong with the reshaping of this function.

After many tries I commented the whole thing out and started from beginning to make this function work.



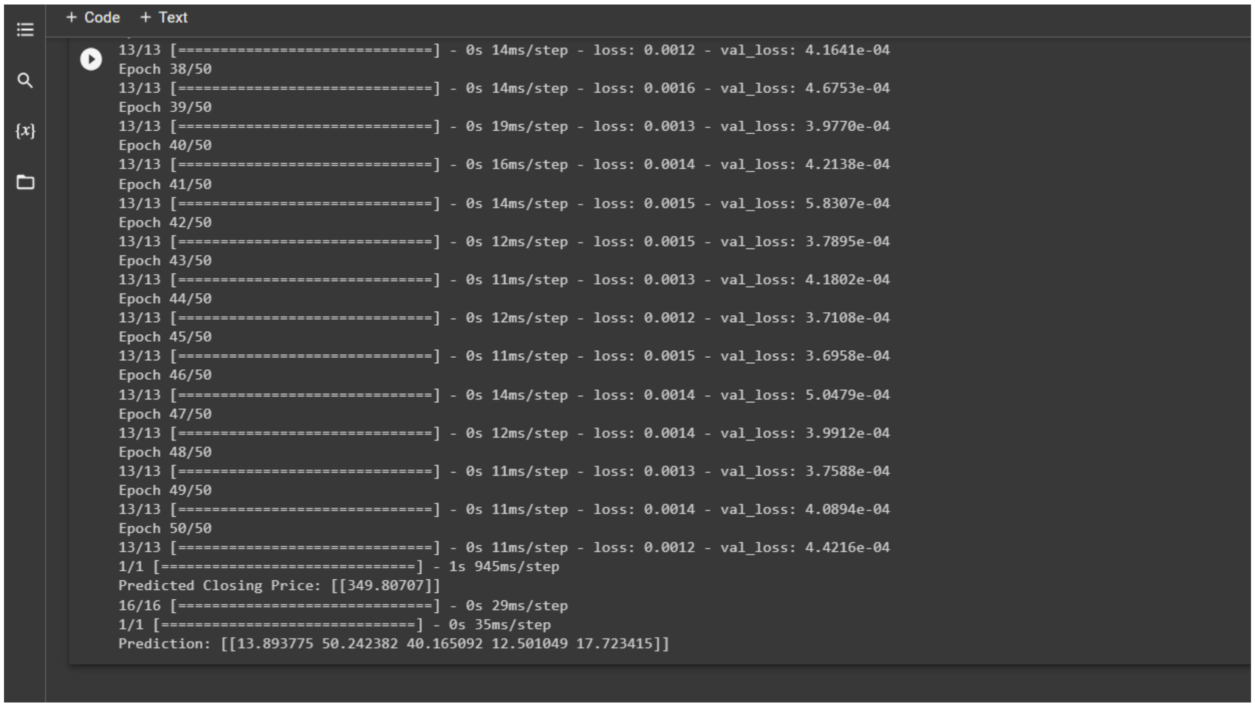
```python
def multivariate_stock_price_prediction(data, target_day, prediction_days=1, num_units=64, num_layers=2, dropout_rate=0.2, epochs=50, b
    # Extract features and target (closing price) from the data
    features = data[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']].values
    target = data['Close'].values

    # Apply Min-Max scaling to features and target
    scaler = MinMaxScaler()
    features_scaled = scaler.fit_transform(features)
    target_scaled = scaler.fit_transform(target.reshape(-1, 1))

    # Create sequences of historical data for training
    X = []
    y = []
    for i in range(len(features_scaled) - target_day - prediction_days + 1):
        X.append(features_scaled[i:i+target_day])
        y.append(target_scaled[i+target_day:i+target_day+prediction_days])
    X = np.array(X)
    y = np.array(y)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Build a sequential LSTM model
    model = Sequential()
    model.add(LSTM(units=num_units, return_sequences=True, input_shape=(target_day, features.shape[1])))
    for _ in range(num_layers - 2):
        model.add(LSTM(units=num_units, return_sequences=True))
        model.add(Dropout(dropout_rate))
    model.add(LSTM(units=num_units))
    model.add(Dropout(dropout_rate))
    model.add(Dense(prediction_days))  # Output layer

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')
```



```python
    # Build a sequential LSTM model
    model = Sequential()
    model.add(LSTM(units=num_units, return_sequences=True, input_shape=(target_day, features.shape[1])))
    for _ in range(num_layers - 2):
        model.add(LSTM(units=num_units, return_sequences=True))
        model.add(Dropout(dropout_rate))
    model.add(LSTM(units=num_units))
    model.add(Dropout(dropout_rate))
    model.add(Dense(prediction_days))  # Output layer

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')
```

```python
    # Train the model
    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, y_test))

    # Make predictions for the specified day
    input_data = features_scaled[-target_day:]  # Use the most recent data
    input_data = np.reshape(input_data, (1, target_day, features.shape[1]))  # Reshape for prediction
    predicted_scaled = model.predict(input_data)

    # Inverse scaling to get the predicted closing price
    predicted_price = scaler.inverse_transform(predicted_scaled)

    return predicted_price

# Example usage:
# Predict the closing price for the next day (target_day=1)
predicted_price = multivariate_stock_price_prediction(data, target_day=1)
print("Predicted Closing Price:", predicted_price)
```

In this function I have created the variables, defined them, made the features array and passed on to take all those as inputs and then building the model to output in close. I have also made new data for training and testing of the data. I faced some issues here, my function wont work because I was repeatedly getting this error where it says the expected input of the numpy array is not correct as it should be. Spent some time here an there searching for it and ended up just making a new sequence that collects previous data and reshaped it accordingly. Hence I get these results.



```
13/13 [==============================] - 0s 14ms/step - loss: 0.0012 - val_loss: 4.1641e-04
Epoch 38/50
13/13 [==============================] - 0s 14ms/step - loss: 0.0016 - val_loss: 4.6753e-04
Epoch 39/50
13/13 [==============================] - 0s 19ms/step - loss: 0.0013 - val_loss: 3.9770e-04
Epoch 40/50
13/13 [==============================] - 0s 16ms/step - loss: 0.0014 - val_loss: 4.2138e-04
Epoch 41/50
13/13 [==============================] - 0s 14ms/step - loss: 0.0015 - val_loss: 5.8307e-04
Epoch 42/50
13/13 [==============================] - 0s 12ms/step - loss: 0.0015 - val_loss: 3.7895e-04
Epoch 43/50
13/13 [==============================] - 0s 11ms/step - loss: 0.0013 - val_loss: 4.1802e-04
Epoch 44/50
13/13 [==============================] - 0s 12ms/step - loss: 0.0012 - val_loss: 3.7108e-04
Epoch 45/50
13/13 [==============================] - 0s 11ms/step - loss: 0.0015 - val_loss: 3.6958e-04
Epoch 46/50
13/13 [==============================] - 0s 14ms/step - loss: 0.0014 - val_loss: 5.0479e-04
Epoch 47/50
13/13 [==============================] - 0s 12ms/step - loss: 0.0014 - val_loss: 3.9912e-04
Epoch 48/50
13/13 [==============================] - 0s 11ms/step - loss: 0.0013 - val_loss: 3.7588e-04
Epoch 49/50
13/13 [==============================] - 0s 11ms/step - loss: 0.0014 - val_loss: 4.0894e-04
Epoch 50/50
13/13 [==============================] - 0s 11ms/step - loss: 0.0012 - val_loss: 4.4216e-04
1/1 [==============================] - 1s 945ms/step
Predicted Closing Price: [[349.80707]]
16/16 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 35ms/step
Prediction: [[13.893775 50.242382 40.165092 12.501049 17.723415]]
```

Changed the epochs and batch size to just check it was working by changing these parameters.

And at last we move on to the last step.

*Combine the above two functions to solve the multivariate, multistep prediction problem.*

I went for a similar approach for this and commented the previous version out. In this combined function, we can specify both the target_day (the day for which we want to predict the closing price) and the prediction_days (the number of consecutive days to predict). The function will return an array of predicted closing prices for those days.

The model architecture remains the same, but the target variable y is now an array of sequences representing the closing prices for multiple days into the future. The output layer of the model has been adjusted accordingly.

## CONCLUSION:

In this project I have used various models to predict different types of data and have learned morer how these learning and predicting models work.

References: https://stackoverflow.com/

https://www.relataly.com/stock-market-prediction-using-multivariate-time-series-in-python/1815/

https://vannguyen-8073.medium.com/using-lstm-multivariate-and-univariate-in-order-to-predict-stock-market-39e8f6551c93