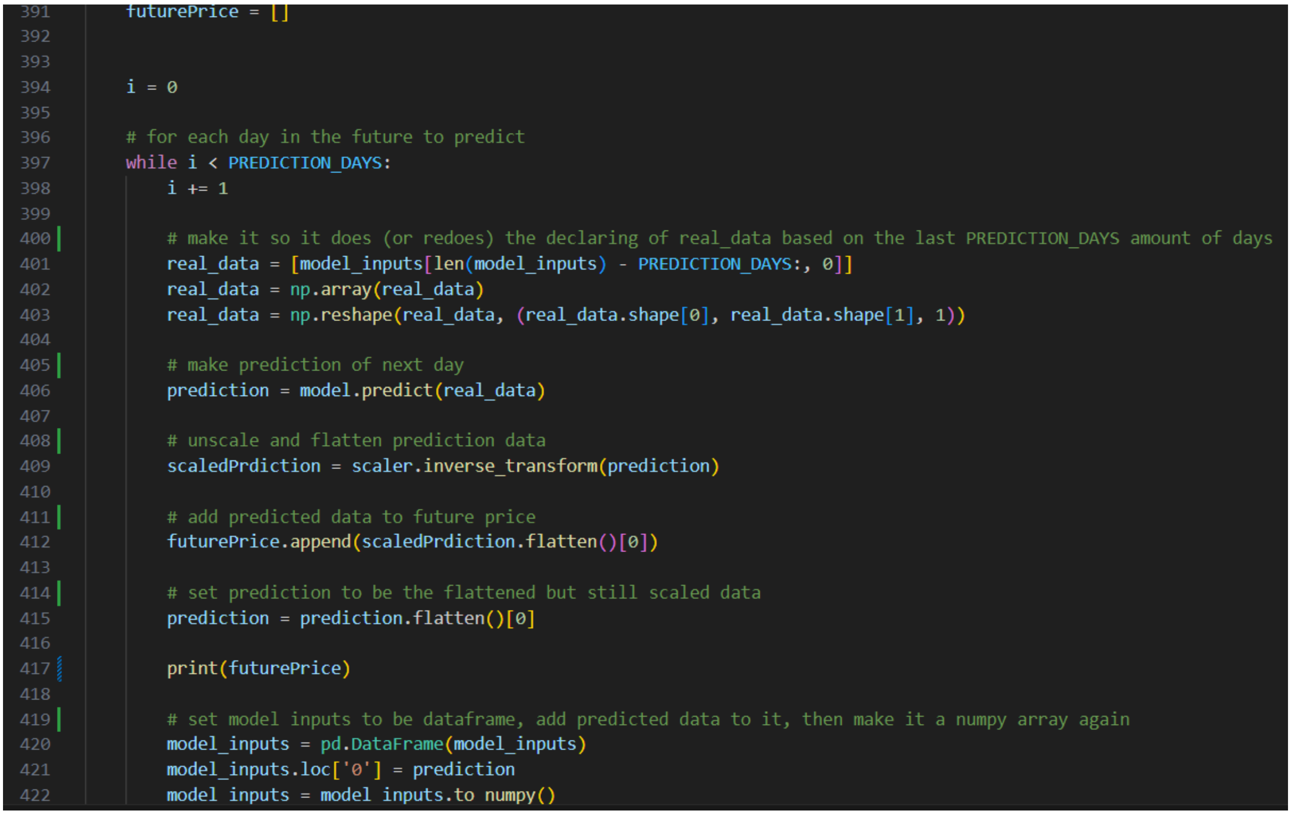
Title: Task B-5 Report

Author: Robin Findlay-Marks, s103603871

Task information:

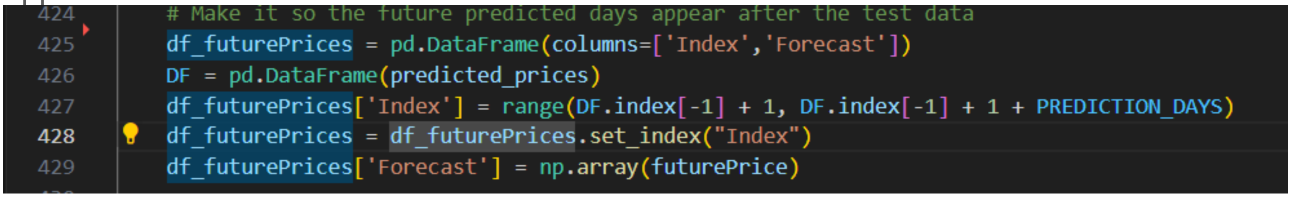
Subtask 1:

For this subtask I had to make the program go from predicting only one day into the future, to N days in the future (multistep). At first I had some difficulty figuring out how to do this. I found a bunch of tutorials of how to do this, but all of them had some issue about them which meant I couldn't use them. What I ended up doing was making a loop to go through each future day that needs to be predicted. In this loop, it would first declare the 'real_data' npArray from model_inputs. It would then make the prediction of the next future day, and add it to real_data. It then adds predicted value to futurePrice and to the end of model_inputs.



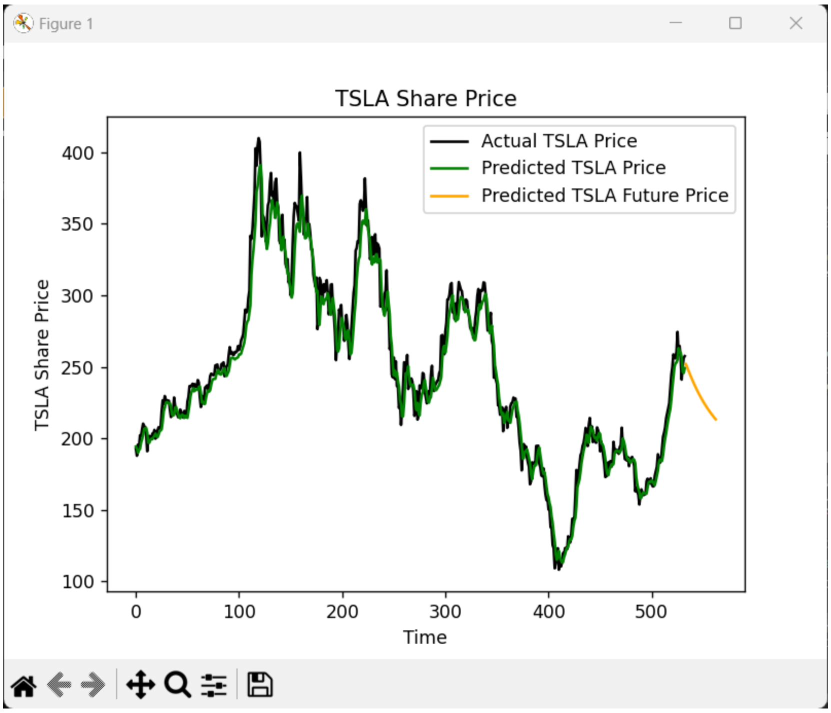
```
    futurePrice = []


    i = 0

    # for each day in the future to predict
    while i < PREDICTION_DAYS:
        i += 1

        # make it so it does (or redoes) the declaring of real_data based on the last PREDICTION_DAYS amount of days
        real_data = [model_inputs[len(model_inputs) - PREDICTION_DAYS:, 0]]
        real_data = np.array(real_data)
        real_data = np.reshape(real_data, (real_data.shape[0], real_data.shape[1], 1))

        # make prediction of next day
        prediction = model.predict(real_data)

        # unscale and flatten prediction data
        scaledPrdiction = scaler.inverse_transform(prediction)

        # add predicted data to future price
        futurePrice.append(scaledPrdiction.flatten()[0])

        # set prediction to be the flattened but still scaled data
        prediction = prediction.flatten()[0]

        print(futurePrice)

        # set model inputs to be dataframe, add predicted data to it, then make it a numpy array again
        model_inputs = pd.DataFrame(model_inputs)
        model_inputs.loc['0'] = prediction
        model_inputs = model_inputs.to_numpy()
```

The next step was to make this predicted data actually display on the graph. This was fairly easy to do except that it would not appear after the test data. To do this I had to make a dataframe from futurePrice and add appropriate index values so the data would appear after the test data



```
    # Make it so the future predicted days appear after the test data
    df_futurePrices = pd.DataFrame(columns=['Index','Forecast'])
    DF = pd.DataFrame(predicted_prices)
    df_futurePrices['Index'] = range(DF.index[-1] + 1, DF.index[-1] + 1 + PREDICTION_DAYS)
    df_futurePrices = df_futurePrices.set_index("Index")
    df_futurePrices['Forecast'] = np.array(futurePrice)
```

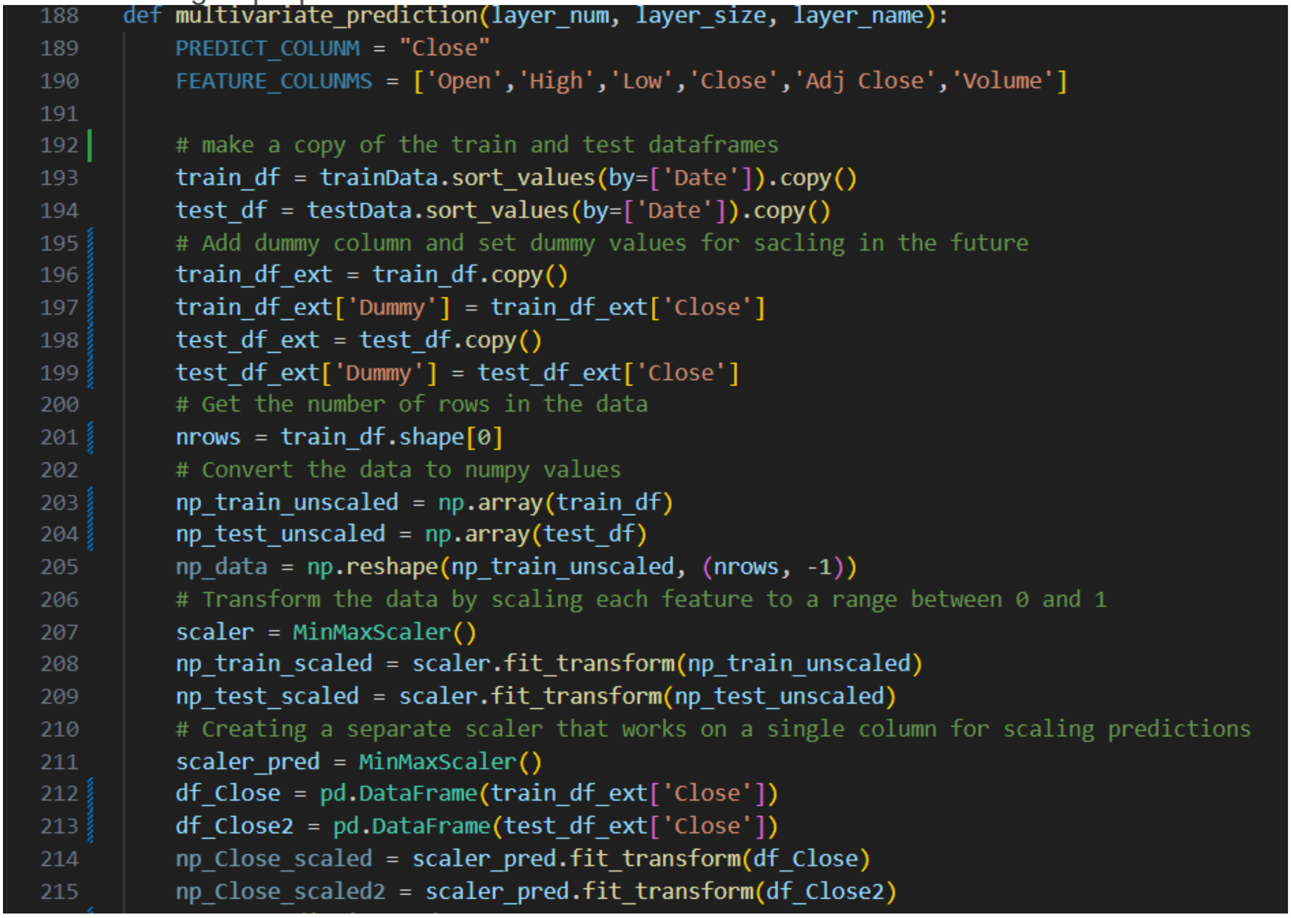
The multistep prediction data can be seen on the graph below



Subtask 2:
For this subtask I had to make it so the program could use multiple sets of the time series data from the downloaded pandas stock data (multivariate). To do this I found a tutorial online that showed me how to do that which is referenced at the bottom of the document. It uses the same system as the old prediction code, however it changes some things to allow for the use of several inputs. This is mainly done by using a ‘sliding window’ technique where a ‘window’ is moved over the different time series data where a sequence of the different time series data are added to the input for the prediction model.

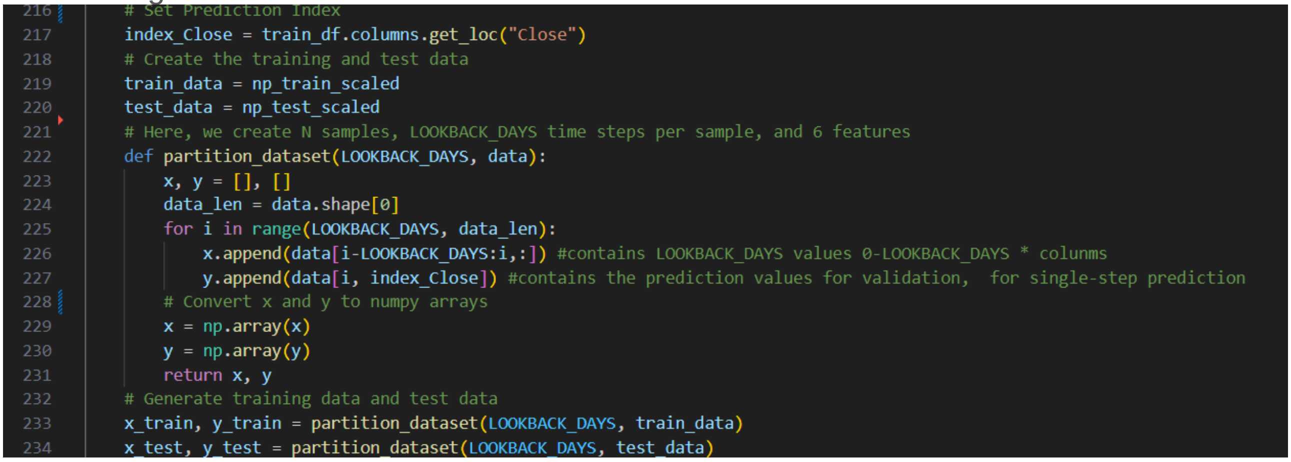
The code for this can be seen below

Lots of scaling to prepare the data



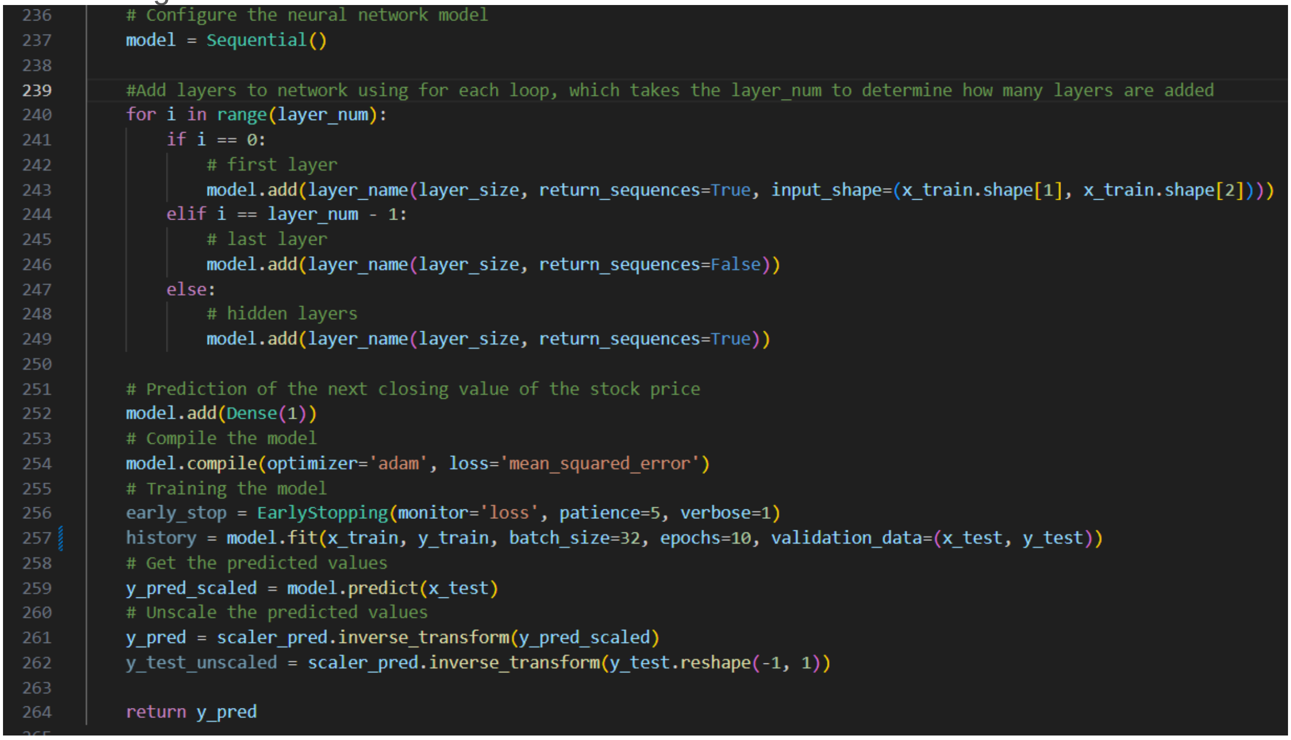
```python
def multivariate_prediction(layer_num, layer_size, layer_name):
    PREDICT_COLUNM = "Close"
    FEATURE_COLUNMS = ['Open','High','Low','Close','Adj Close','Volume']

    # make a copy of the train and test dataframes
    train_df = trainData.sort_values(by=['Date']).copy()
    test_df = testData.sort_values(by=['Date']).copy()
    # Add dummy column and set dummy values for sacling in the future
    train_df_ext = train_df.copy()
    train_df_ext['Dummy'] = train_df_ext['Close']
    test_df_ext = test_df.copy()
    test_df_ext['Dummy'] = test_df_ext['Close']
    # Get the number of rows in the data
    nrows = train_df.shape[0]
    # Convert the data to numpy values
    np_train_unscaled = np.array(train_df)
    np_test_unscaled = np.array(test_df)
    np_data = np.reshape(np_train_unscaled, (nrows, -1))
    # Transform the data by scaling each feature to a range between 0 and 1
    scaler = MinMaxScaler()
    np_train_scaled = scaler.fit_transform(np_train_unscaled)
    np_test_scaled = scaler.fit_transform(np_test_unscaled)
    # Creating a separate scaler that works on a single column for scaling predictions
    scaler_pred = MinMaxScaler()
    df_Close = pd.DataFrame(train_df_ext['Close'])
    df_Close2 = pd.DataFrame(test_df_ext['Close'])
    np_Close_scaled = scaler_pred.fit_transform(df_Close)
    np_Close_scaled2 = scaler_pred.fit_transform(df_Close2)
```

The ‘sliding window’



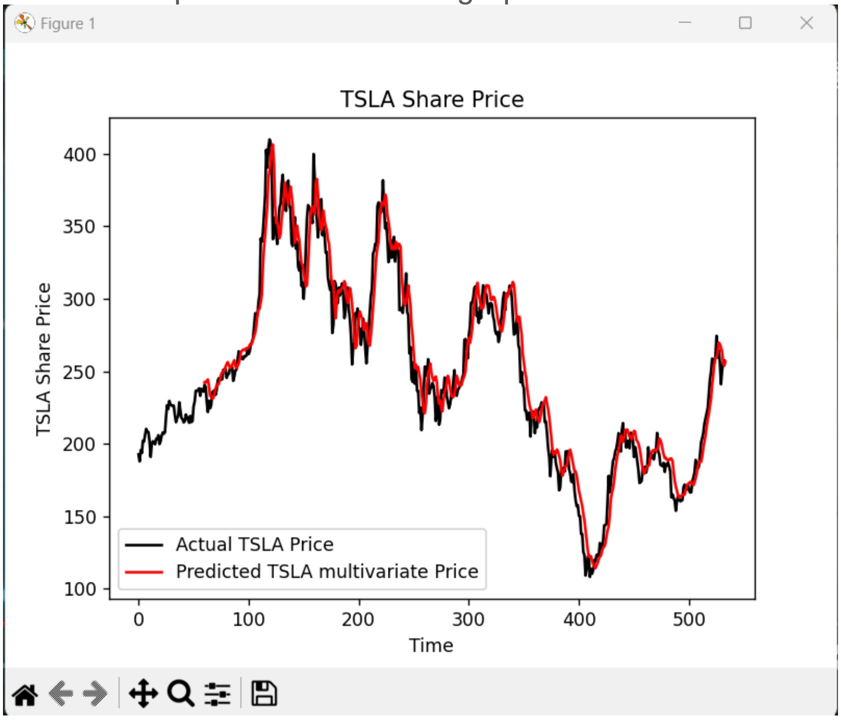
```python
    # Set Prediction Index
    index_Close = train_df.columns.get_loc("Close")
    # Create the training and test data
    train_data = np_train_scaled
    test_data = np_test_scaled
    # Here, we create N samples, LOOKBACK_DAYS time steps per sample, and 6 features
    def partition_dataset(LOOKBACK_DAYS, data):
        x, y = [], []
        data_len = data.shape[0]
        for i in range(LOOKBACK_DAYS, data_len):
            x.append(data[i-LOOKBACK_DAYS:i,:]) #contains LOOKBACK_DAYS values 0-LOOKBACK_DAYS * columns
            y.append(data[i, index_Close]) #contains the prediction values for validation,  for single-step prediction
        # Convert x and y to numpy arrays
        x = np.array(x)
        y = np.array(y)
        return x, y
    # Generate training data and test data
    x_train, y_train = partition_dataset(LOOKBACK_DAYS, train_data)
    x_test, y_test = partition_dataset(LOOKBACK_DAYS, test_data)
```

The making of the model

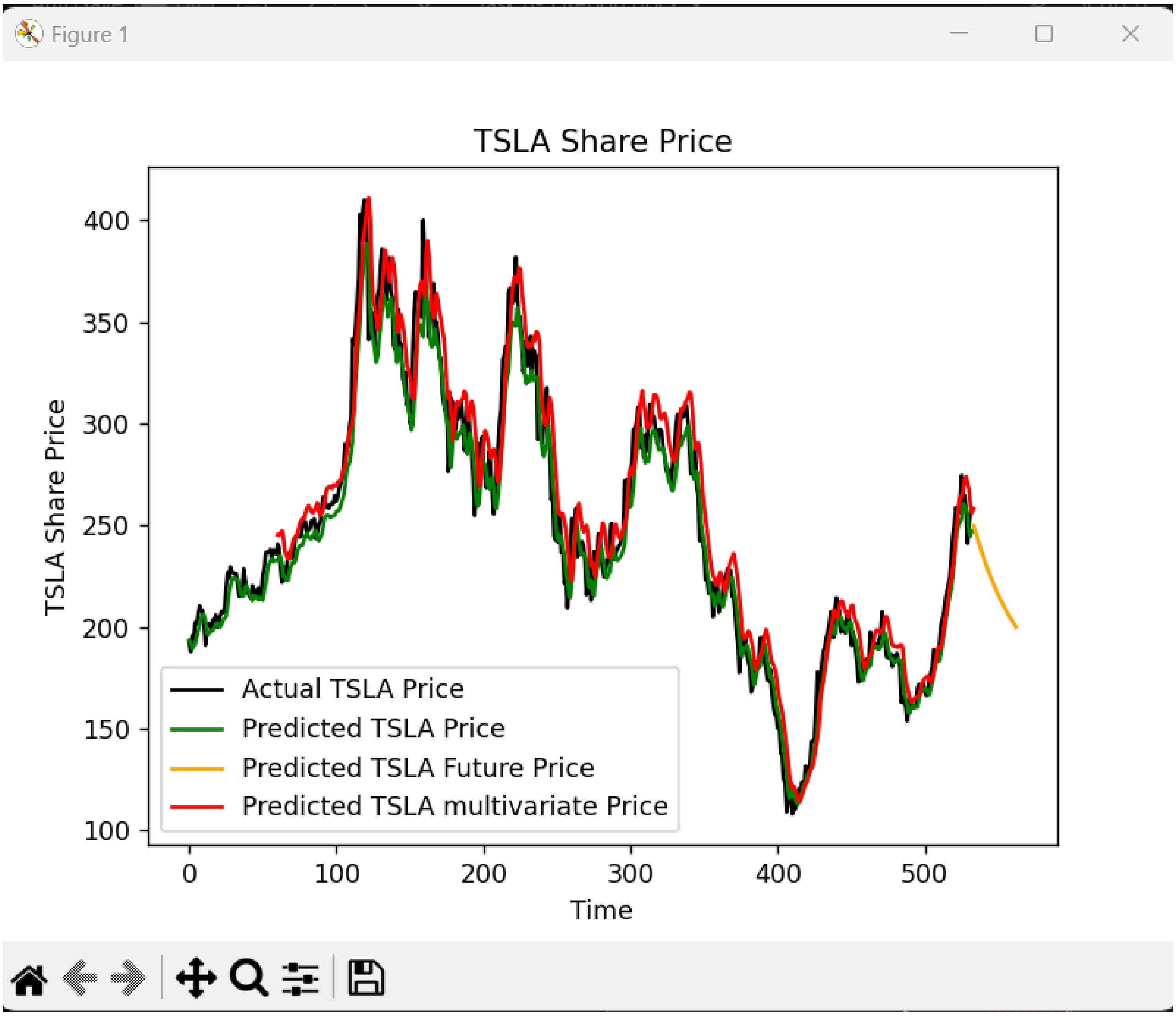


```
    # Configure the neural network model
    model = Sequential()

    #Add layers to network using for each loop, which takes the layer_num to determine how many layers are added
    for i in range(layer_num):
        if i == 0:
            # first layer
            model.add(layer_name(layer_size, return_sequences=True, input_shape=(x_train.shape[1], x_train.shape[2])))
        elif i == layer_num - 1:
            # last layer
            model.add(layer_name(layer_size, return_sequences=False))
        else:
            # hidden layers
            model.add(layer_name(layer_size, return_sequences=True))

    # Prediction of the next closing value of the stock price
    model.add(Dense(1))
    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')
    # Training the model
    early_stop = EarlyStopping(monitor='loss', patience=5, verbose=1)
    history = model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_test, y_test))
    # Get the predicted values
    y_pred_scaled = model.predict(x_test)
    # Unscale the predicted values
    y_pred = scaler_pred.inverse_transform(y_pred_scaled)
    y_test_unscaled = scaler_pred.inverse_transform(y_test.reshape(-1, 1))

    return y_pred
```

Multivatiate predicted data on a graph:



Subtask 3:
The final subtask was to make it so both the prediction data from subtask 1 and subtask 2 displayed on the say graph. This was very easy to do as all I had to do was input the multivariate data on to the graph and it displayed.



References:

Follonier, F 2020, *Stock Market Prediction using Multivariate Time Series and Recurrent Neural Networks in Python*, relataly, viewed 21/09/2023, <https://www.relataly.com/stock-market-prediction-using-multivariate-time-series-in-python/1815/>.