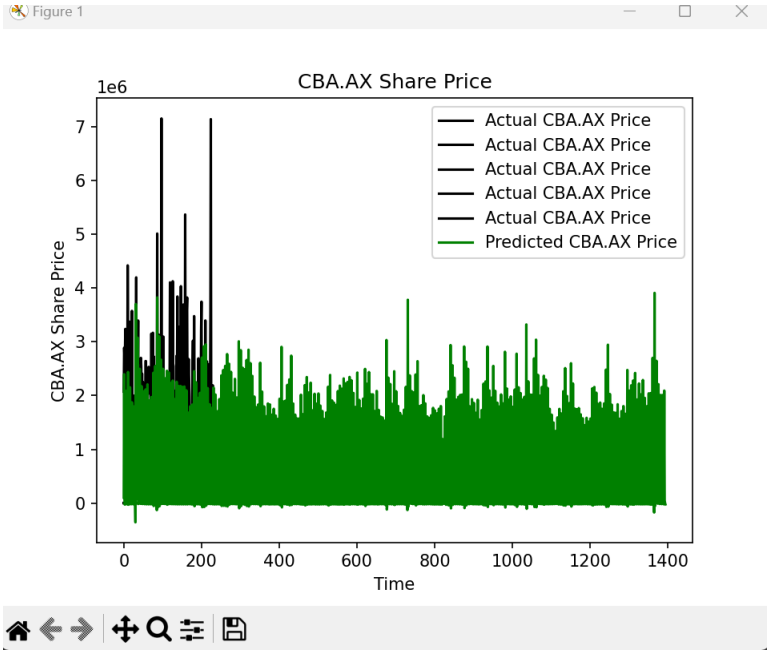
B.2

In order to create a function with the required features, I had a look at the P.1 code, and also went over that YouTube tutorial again. I noticed the function in P.1 takes a ticker value, which, on research, is a unique identifier for a company on the stock exchange, consisting of 1-5 letters.

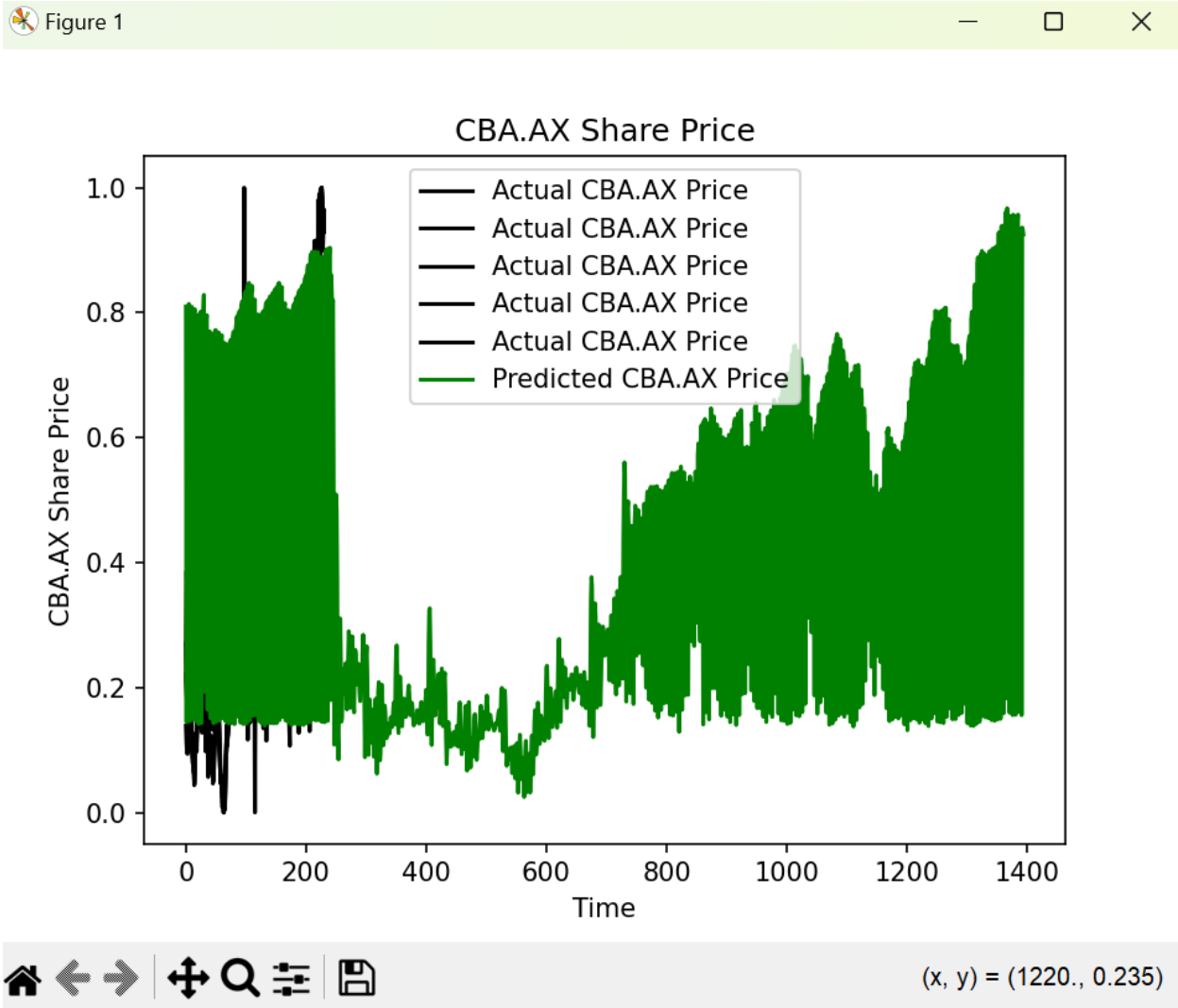
I also found that a scaler is a value between 0-10. The scaler is used to compress values higher than that to between 0-10. The ticker was not necessary for this task, but the scaling was, as well as handling NaN values, splitting the data, and saving the data.

I'm really sorry about the wait for this task, but I was confused about the results for my program, and I was not doing well (ASD related, plus had the flu). I thought there was an issue with the scaling, but I'm not exactly sure what the issue is now. Looking over the task document, I believe I have achieved the requirements of the task, regardless. If there is an issue I will resolve it in the next task.

<u>This is the initial result before scaling the columns (and not date splitting)</u>



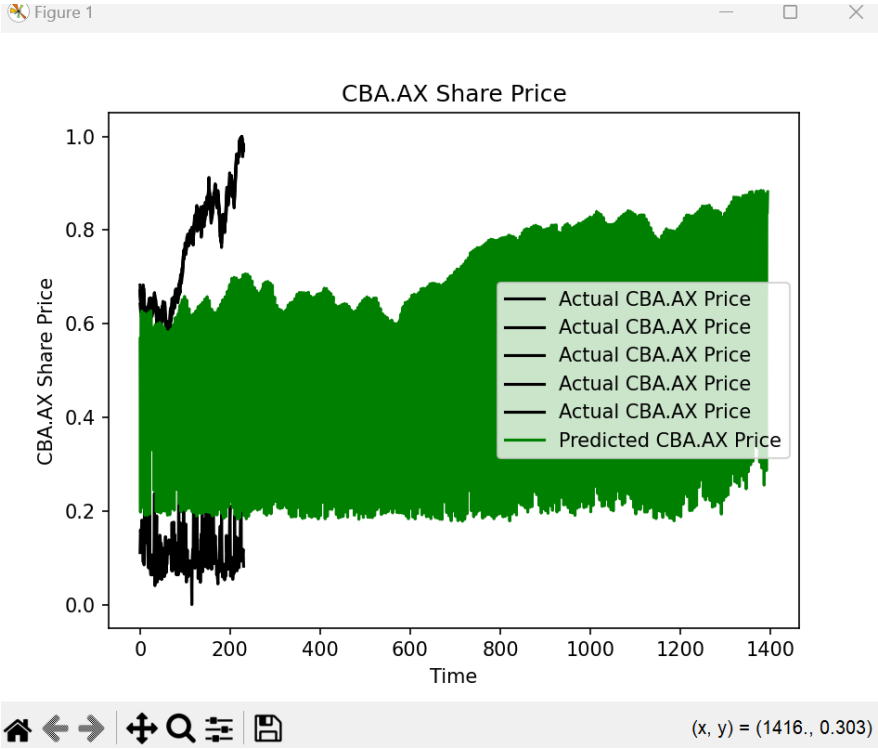
And after scaling the columns



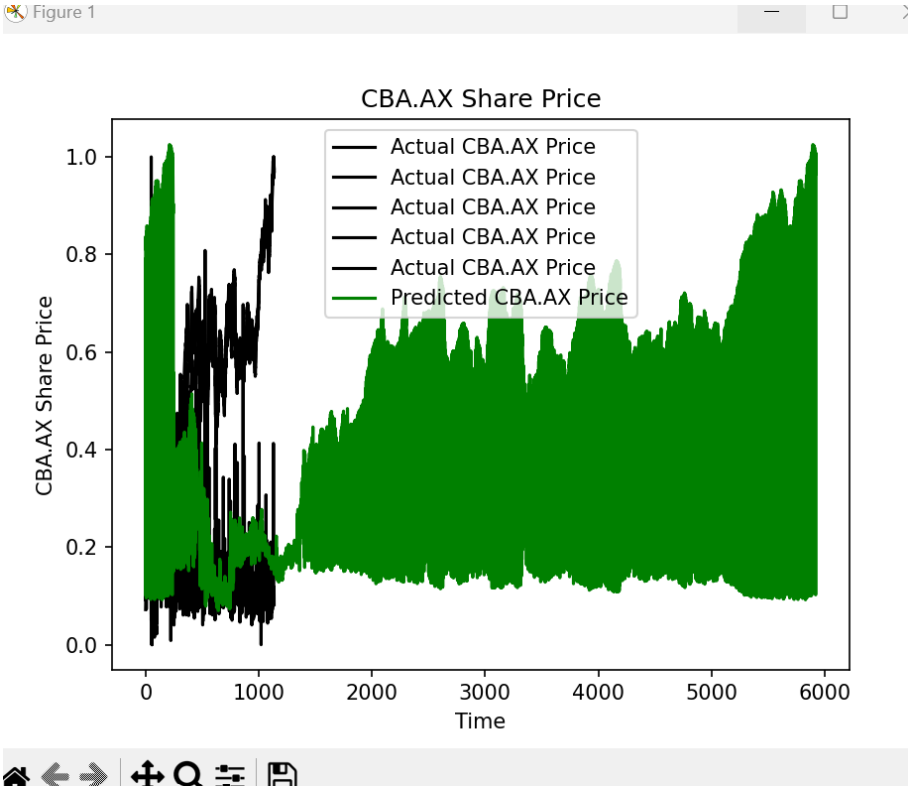

```
'column_scaler': {'Adj Close': MinMaxScaler(), 'Volume': MinMaxScaler(), 'Open': MinMaxScaler(), 'High': MinMaxScaler(), 'Low': MinMaxScaler()}}
```

As you can see in the terminal output, MinMaxScaler has been applied to the columns. The issue that I couldn't understand is the chaotic nature of the green on the graph. There is a lot of shading in of green, so I assumed there was something wrong, but if there is, I'm yet to figure out how to fix it. I stored the scalers to a dataframe and returned it at the end of the function.

Splitting by date



Splitting randomly



Code: located in the venv/scripts folder, file '*stock_prediction2.py*'

Load data function:

For this task, it was my understanding that I needed to make a function that loads data for the Yahoo yfinance dataframe, with parameters that let me pass the start and end dates for the dataframe. So, I put the initialisation of the dataframe inside the function, and passed the date parameters into the dataframe declaration, so that when the function is called elsewhere, different dates can be specified for the data loading.

```
def shuffle_in_unison(a, b):
    # shuffle two arrays in the same way
    state = np.random.get_state()
    np.random.shuffle(a)
    np.random.set_state(state)
    np.random.shuffle(b)

def load_data(tr_start_date, tr_end_date, te_start_date, te_end_date, n_steps=50, lookup_step=1, shuffle=True, split_by_date=False,
              test_size=0.2, feature_columns=['Adj Close', 'Volume', 'Open', 'High', 'Low']):
    #load from yfinance
    if os.path.exists('data.csv'):
        #load data from file if it exists
        df = pd.read_csv('data.csv')
        print("it exists")
    else:
        #dataframe starts at the furthest date (train start), through to the most recent date (test end), for splitting purposes
        df = yf.download(COMPANY, start=tr_start_date, end=te_end_date)
    result = {}
    result['df'] = df.copy()

    for col in feature_columns: #test if column exists in dataframe
        assert col in df.columns, f"'{col}' does not exist in the dataframe."

    #scale feature columns
    column_scaler = {}
    for column in feature_columns:
        scaler = MinMaxScaler(feature_range=(0, 1))
        df[column] = scaler.fit_transform(np.expand_dims(df[column].values, axis=1))
        column_scaler[column] = scaler
    result["column_scaler"] = column_scaler
    scaler_data = pd.DataFrame(column_scaler, index=[0])

    #for column in scaler_data.columns:
    #    scaler_data[column] = column_scaler[column].transform(scaler_data[column])

    # add date as a column
    if "Date" not in df.columns:
        df["Date"] = df.index
```

```
last_sequence = np.array(df[feature_columns].tail(lookup_step))
sequence_data = []
sequences = deque(maxlen=n_steps)

for entry, target in zip(df[feature_columns + ["Date"]].values, df['Low'].values):
    sequences.append(entry)
    if len(sequences) == n_steps:
        sequence_data.append([np.array(sequences), target])

# get the last sequence by appending the last `n_step` sequence with `lookup_step` sequence
# for instance, if n_steps=50 and lookup_step=10, last_sequence should be of 60 (that is 50+10) length
# this last_sequence will be used to predict future stock prices that are not available in the dataset
last_sequence = list([s[:len(feature_columns)] for s in sequences]) + list(last_sequence)
last_sequence = np.array(last_sequence).astype(np.float32)
# add to result
result['last_sequence'] = last_sequence
#split data
X, y = [], []
for seq, target in sequence_data:
    X.append(seq)
    y.append(target)
X = np.array(X)
y = np.array(y)
```

For splitting the data, I followed the general layout from the P1 code but changed what they labelled as "split_by_date" as it didn't look like it was splitting by date to me, but samples rather.

When it came to splitting the data randomly, I used the train_test_split function, but there was an error (I believe regarding n_samples=0), so I implemented some extra code that was in P1 to see if that was related, and it fixed the issue. It seems it had something to do with time series prediction. The code involves the "sequence" shown above.

```
#by date
if split_by_date:
    df.set_index( keys: 'Date', inplace=True)
    train_data_split = df.loc[tr_start_date : tr_end_date]
    test_data_split = df.loc[te_start_date : te_end_date]
    result["X_train"] = train_data_split.values
    result["y_train"] = train_data_split.values
    result["X_test"] = test_data_split.values
    result["y_test"] = test_data_split.values
else:
#split randomly
    train_samples = int((1 - test_size) * len(X))
    train_data_split = df
    test_data_split = df
    result["X_train"] = X[:train_samples]
    result["y_train"] = y[:train_samples]
    result["X_test"] = X[train_samples:]
    result["y_test"] = y[train_samples:]
    result["X_train"], result["X_test"], result["y_train"], result["y_test"] = train_test_split( *arrays: X, y,
                                                    test_size=test_size, train_size=0.6, random_state=42)
```

For the NaN issue, I used the dropna() function. ffill() and bfill() were an option but the P1 code used dropna() so I went with what they did.

```
    #handle NaN values
    df.dropna(inplace=True)

    #save data
    df.to_csv( path_or_buf: 'data.csv',index=False)
    return scaler_data, train_data_split, test_data_split

#year-month-day
TRAIN_START_DATE = '2020-01-01'
TRAIN_END_DATE = '2023-08-01'
TEST_START_DATE = '2023-08-02'
TEST_END_DATE = '2024-07-02'

scaler_data, train_data_split, test_data_split = load_data(
    tr_start_date = TRAIN_START_DATE,
    tr_end_date = TRAIN_END_DATE,
    te_start_date = TEST_START_DATE,
    te_end_date = TEST_END_DATE
)

PRICE_VALUE = ['Adj Close', 'Volume', 'Open', 'High', 'Low']
```

At the end of the function, I saved the dataframe to a csv file using *df.to_csv('data.csv', index=False)*. On the first run, this would create the data.csv file, so I went back to the top and created an if condition to set the dataframe to *df.read_csv('data.csv')*, if it exists, else, df would be set to the usual *yf.download(COMPANY, start=tr_start_date, end=te_end_date)*.

I noticed that the columns were case sensitive, so I had to put a capital before each feature_column name.

Alterations to already existing code, so that it uses the returns splitting data

```
#above here add functions
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(train_data_split[PRICE_VALUE].values.reshape(-1, 1))
PREDICTION_DAYS = 60 # Original
# To store the training data
x_train = []
y_train = []
#scaled_data = scaled_data[:,0] # Turn the 2D array back to a 1D array
# Prepare the data
for x in range(PREDICTION_DAYS, len(scaled_data)):
    x_train.append(scaled_data[x-PREDICTION_DAYS:x])
    y_train.append(scaled_data[x])

# Convert them into an array
x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, newshape: (x_train.shape[0], x_train.shape[1], 1))
#------------------------------------------------------------------------------
model = Sequential() # Basic neural network
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, epochs=25, batch_size=32)

#------------------------------------------------------------------------------
# Test the model accuracy on existing data
#------------------------------------------------------------------------------


actual_prices = test_data_split[PRICE_VALUE].values

total_dataset = pd.concat( objs: (train_data_split[PRICE_VALUE], test_data_split[PRICE_VALUE]), axis=0)

model_inputs = total_dataset[len(total_dataset) - len(test_data_split) - PREDICTION_DAYS:].values
model_inputs = model_inputs.reshape(-1, 1)
model_inputs = scaler.transform(model_inputs)
```

```
print(f"Prediction: {prediction}")
print("Train Data:")
print(train_data_split)
print("\nTest Data:")
print(test_data_split)
```