Task Report Cos30018 Option B

B.1: Set Up

Name: Le Bao Nguyen

Student Id: 104169837

I. Setting up the environment:

- We started downloading the code v0.1 and P1 from the canvas:

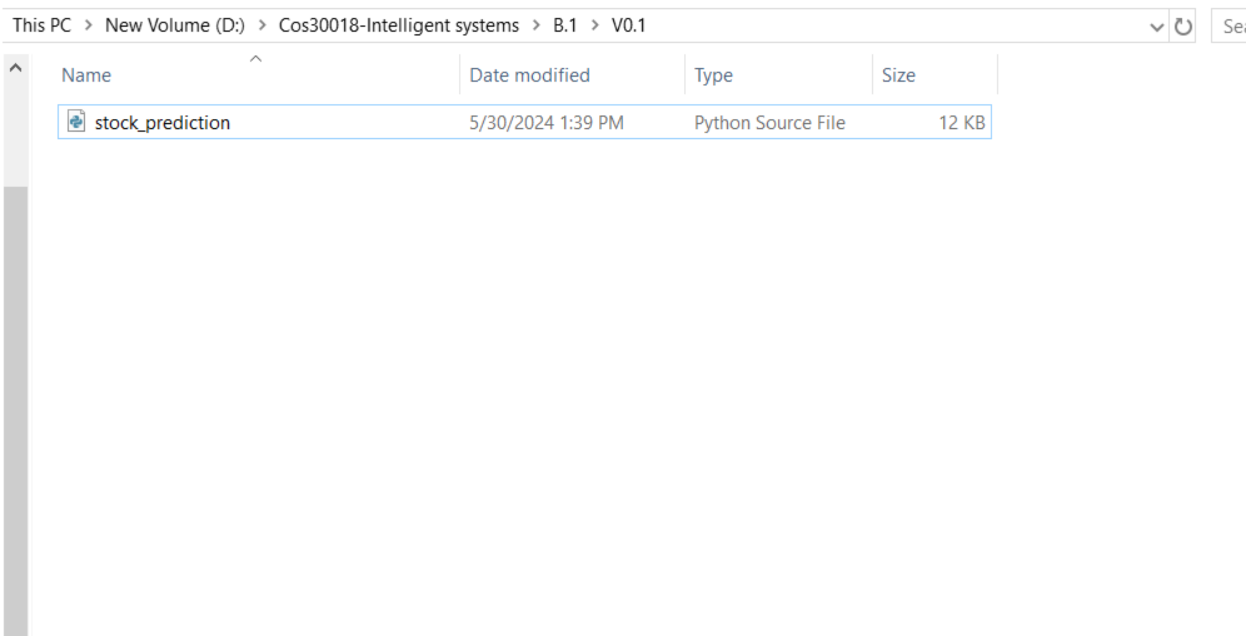


Figure 1: Downloading the v0.1 code from canvas into the computer.

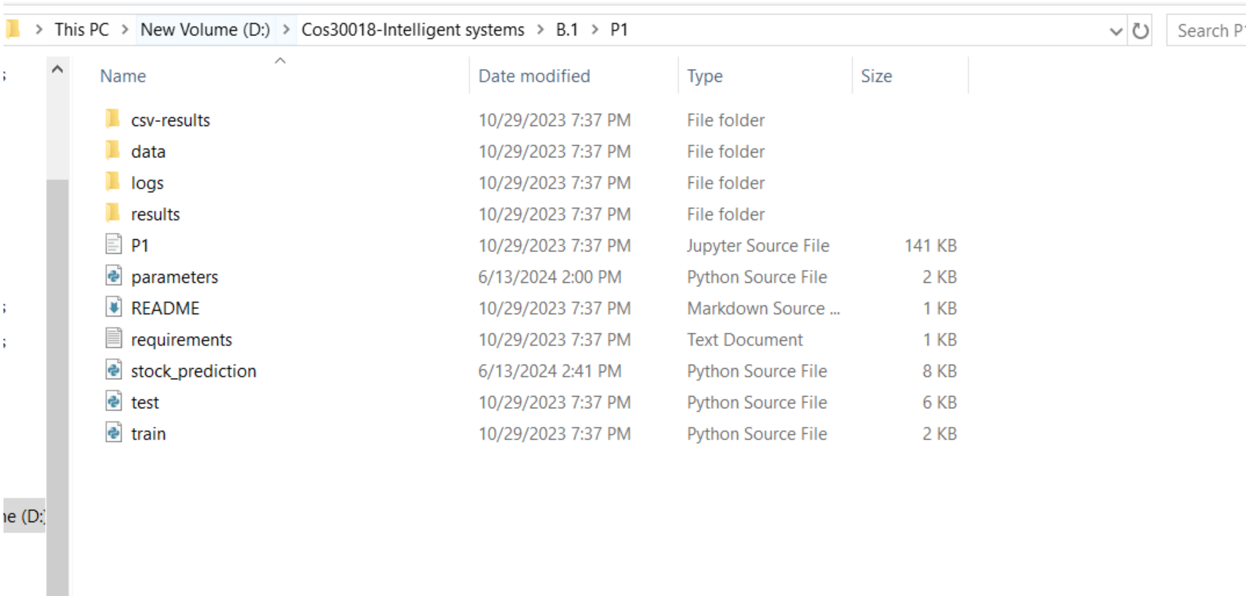


Figure 2: Downloading the P1 code from canvas into the computer.

- We upload both v0.1 and P.1 folder into the google drive to link to gg drive (we can still run the code even if it's not link to the google drive or not):

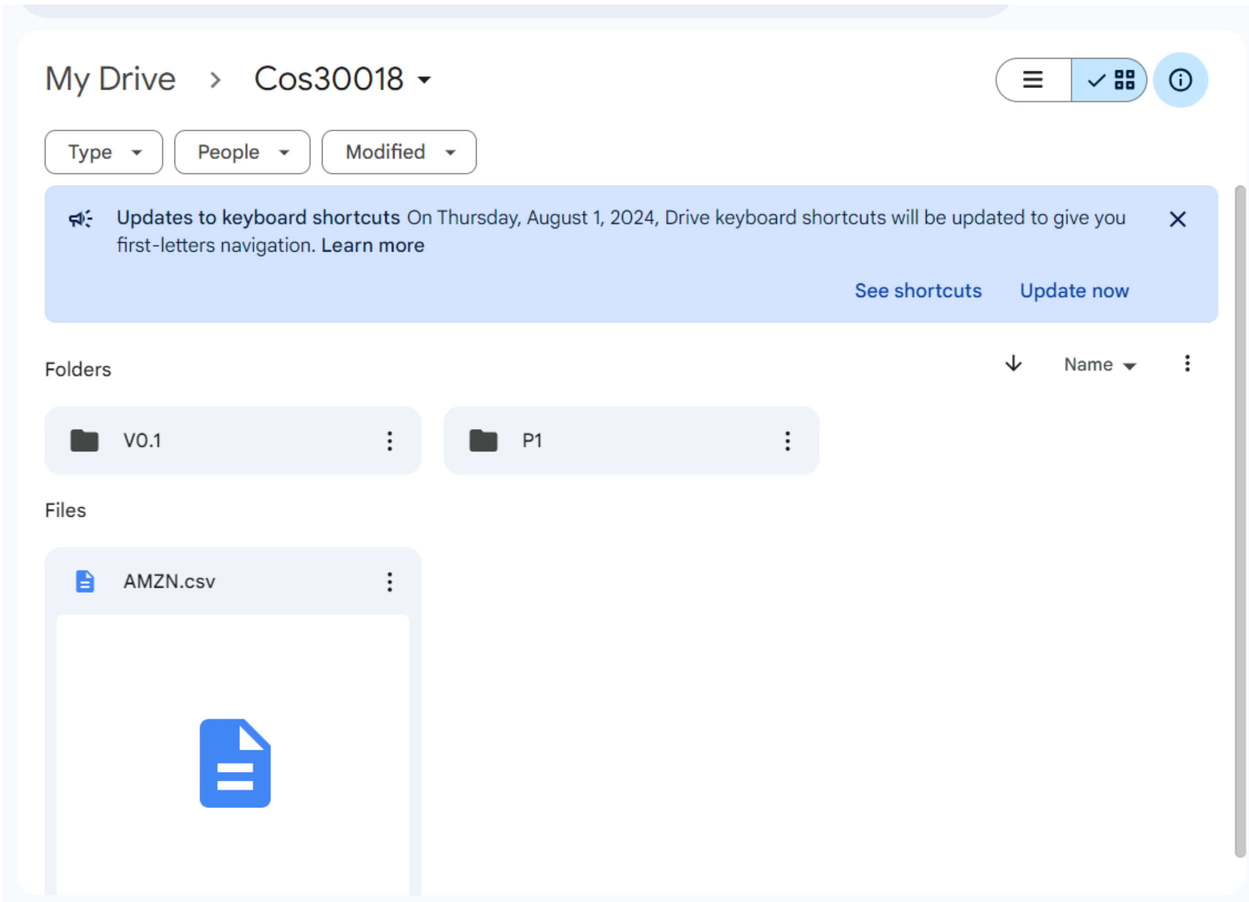


Figure 3: Uploading the v0.1 and P1 code into folder in google drive.

- After that, we open the gg collab and start to link to our gg drive that contain folders and direct to the folder locations:

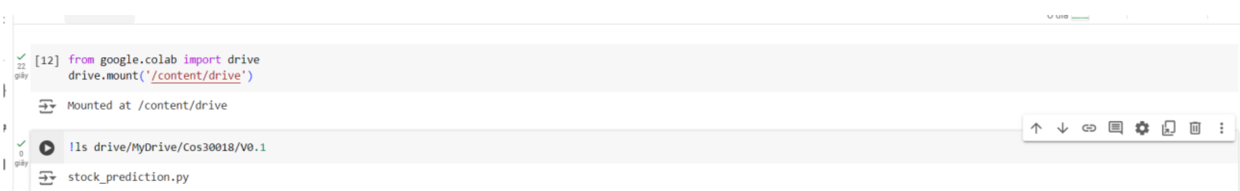


Figure 4: Linking the v0.1 and P1 code from google drive to google collab.

- We import the libraries or extensions that we need to run the code and use "!pip install ...."install the libraries if they are still not available in the gg collab:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
import datetime as dt
import tensorflow as tf
import yfinance as yf

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, InputLayer
```

```
!pip install numpy pandas scikit-learn matplotlib tensorflow keras yfinance
```

Figure 5: Importing and installing libraries.

- After completing import the resources, we start to implement code into the gg collab to run it but we implement separately to test out if the code run or not:

```
[2] DATA_SOURCE = "yahoo"
    COMPANY = "TSLA"

    # start = '2012-01-01', end='2017-01-01'
    TRAIN_START = '2015-01-01'
    TRAIN_END = '2020-01-01'

    data =  yf.download(COMPANY, start=TRAIN_START, end=TRAIN_END, progress=False)
```

```
[3] PRICE_VALUE = "Close"

    scaler = MinMaxScaler(feature_range=(0, 1))

    scaled_data = scaler.fit_transform(data[PRICE_VALUE].values.reshape(-1, 1))
```

Figure 6: Implementing the code separately to check if it works or not (1).

```
[4] # Number of days to look back to base the prediction
    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)
    # Now, x_train is a 2D array(p,q) where p = len(scaled_data) - PREDICTION_DAYS
    # and q = PREDICTION_DAYS; while y_train is a 1D array(p)

    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    # We now reshape x_train into a 3D array(p, q, 1); Note that x_train
    # is an array of p inputs with each input being a 2D array
```

```
[5] model = Sequential() # Basic neural network
    # See: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential
    # for some useful examples

    model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
```

Figure 7: Implementing the code separately to check if it works or not (2).

```python
model.add(Dropout(0.2))
# The Dropout layer randomly sets input units to 0 with a frequency of
# rate (= 0.2 above) at each step during training time, which helps
# prevent overfitting (one of the major problems of ML).

model.add(LSTM(units=50, return_sequences=True))
# More on Stacked LSTM:
# https://machinelearningmastery.com/stacked-long-short-term-memory-networks/

model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))

model.add(Dense(units=1))
# Prediction of the next closing value of the stock price

# We compile the model by specify the parameters for the model
# See lecture Week 6 (COS30018)
model.compile(optimizer='adam', loss='mean_squared_error')
```

Figure 8: Implementing the code separately to check if it works or not (3).

```
[7] model.fit(x_train, y_train, epochs=25, batch_size=32)

Epoch 1/25
38/38 [==============================] - 9s 104ms/step - loss: 0.0253
Epoch 2/25
38/38 [==============================] - 3s 82ms/step - loss: 0.0082
Epoch 3/25
38/38 [==============================] - 3s 71ms/step - loss: 0.0076
Epoch 4/25
38/38 [==============================] - 3s 71ms/step - loss: 0.0063
Epoch 5/25
38/38 [==============================] - 3s 77ms/step - loss: 0.0065
Epoch 6/25
38/38 [==============================] - 4s 106ms/step - loss: 0.0057
Epoch 7/25
38/38 [==============================] - 4s 115ms/step - loss: 0.0062
Epoch 8/25
38/38 [==============================] - 4s 91ms/step - loss: 0.0054
Epoch 9/25
38/38 [==============================] - 4s 94ms/step - loss: 0.0054
Epoch 10/25
38/38 [==============================] - 3s 85ms/step - loss: 0.0054
Epoch 11/25
38/38 [==============================] - 3s 72ms/step - loss: 0.0050
Epoch 12/25
38/38 [==============================] - 3s 72ms/step - loss: 0.0047
Epoch 13/25
```

✓ 0 giây hoàn thành lúc 23:45

Figure 9: Implementing the code separately to check if it works or not (4).

II. Testing the code v0.1 and P1:

- We run the v0.1 code and train through 25 epochs before creating the charts to show the accurate and predicted data of TSLA stock. After creating the chart, the model predicts the stock price for the next day after the test period.
- The Sequential API from Keras is used to build the model. To avoid overfitting, it is composed of three LSTM layers, each of which is followed by a Dropout layer.

```
model.fit(x_train, y_train, epochs=25, batch_size=32)

Epoch 1/25
38/38 [==============================] - 9s 104ms/step - loss: 0.0253
Epoch 2/25
38/38 [==============================] - 3s 82ms/step - loss: 0.0082
Epoch 3/25
38/38 [==============================] - 3s 71ms/step - loss: 0.0076
Epoch 4/25
38/38 [==============================] - 3s 71ms/step - loss: 0.0063
Epoch 5/25
38/38 [==============================] - 3s 77ms/step - loss: 0.0065
Epoch 6/25
38/38 [==============================] - 4s 106ms/step - loss: 0.0057
Epoch 7/25
38/38 [==============================] - 4s 115ms/step - loss: 0.0062
Epoch 8/25
38/38 [==============================] - 4s 91ms/step - loss: 0.0054
Epoch 9/25
38/38 [==============================] - 4s 94ms/step - loss: 0.0054
Epoch 10/25
38/38 [==============================] - 3s 85ms/step - loss: 0.0054
Epoch 11/25
38/38 [==============================] - 3s 72ms/step - loss: 0.0050
Epoch 12/25
38/38 [==============================] - 3s 72ms/step - loss: 0.0047
Epoch 13/25
38/38 [==============================] - 3s 77ms/step - loss: 0.0044
Epoch 14/25
38/38 [==============================] - 4s 106ms/step - loss: 0.0045
Epoch 15/25
38/38 [==============================] - 3s 74ms/step - loss: 0.0046
Epoch 16/25
38/38 [==============================] - 3s 72ms/step - loss: 0.0038
Epoch 17/25
38/38 [==============================] - 3s 73ms/step - loss: 0.0042
Epoch 18/25
38/38 [==============================] - 4s 103ms/step - loss: 0.0047
Epoch 19/25
38/38 [==============================] - 3s 76ms/step - loss: 0.0043
Epoch 20/25
38/38 [==============================] - 3s 72ms/step - loss: 0.0038
Epoch 21/25
38/38 [==============================] - 3s 73ms/step - loss: 0.0037
Epoch 22/25
38/38 [==============================] - 3s 85ms/step - loss: 0.0038
Epoch 23/25
38/38 [==============================] - 4s 96ms/step - loss: 0.0037
Epoch 24/25
38/38 [==============================] - 3s 73ms/step - loss: 0.0033
Epoch 25/25
38/38 [==============================] - 3s 73ms/step - loss: 0.0034
<keras.src.callbacks.History at 0x7e64b84c68c0>
```

Figure 10: Training the model through 25 epochs before printing the chart.

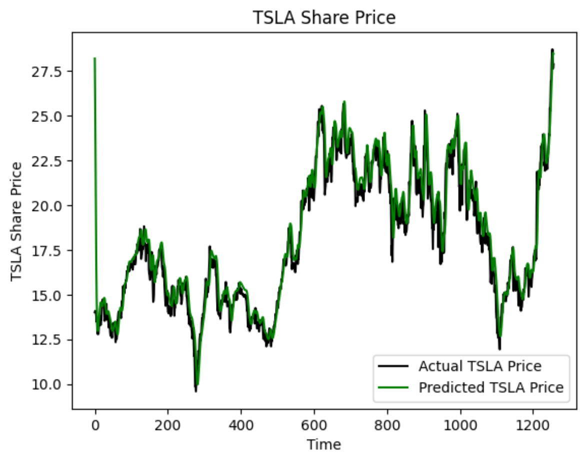


Figure 11: The prediction chart after training.

```
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))

prediction = model.predict(real_data)
prediction = scaler.inverse_transform(prediction)
print(f"Prediction: {prediction}")
```

```
1/1 [==============================] - 0s 62ms/step
Prediction: [[28.212034]]
```

Figure 12: Printing the prediction for next day.

- When we run the P1 code, it has to go through 500 epochs to train which take a long time to do it so we decide to reduce it to 15 epochs by editing the epochs in the parameter.py.

```
TEST_SIZE = 0.2
# features to use
FEATURE_COLUMNS = ["adjclose", "volume", "open", "high", "low"]
# date now
date_now = time.strftime("%Y-%m-%d")

### model parameters

N_LAYERS = 2
# LSTM cell
CELL = LSTM
# 256 LSTM neurons
UNITS = 256
# 40% dropout
DROPOUT = 0.4
# whether to use bidirectional RNNs
BIDIRECTIONAL = False

### training parameters

# mean absolute error loss
# LOSS = "mae"
# huber loss
LOSS = "huber_loss"
OPTIMIZER = "adam"
BATCH_SIZE = 64
EPOCHS = 500

# Amazon stock market
ticker = "AMZN"
ticker_data_filename = os.path.join("data", f"{ticker}_{date_now}.csv")
# model name to save, making it as unique as possible based on parameters
model_name = f"{date_now}_{ticker}-{shuffle_str}-{scale_str}-{split_by_date_str}-\
{LOSS}-{OPTIMIZER}-{CELL.__name__}-seq-{N_STEPS}-step-{LOOKUP_STEP}-layers-{N_LAYERS}-units-{UNITS}"
if BIDIRECTIONAL:
    model_name += "-b"
```

Figure 13: We need to replace the epochs from 500 to 15 for faster train.

```python
    os.mkdir("data")

# load the data
data = load_data(ticker, N_STEPS, scale=SCALE, split_by_date=SPLIT_BY_DATE,
                 shuffle=SHUFFLE, lookup_step=LOOKUP_STEP, test_size=TEST_SIZE,
                 feature_columns=FEATURE_COLUMNS)

# save the dataframe
data["df"].to_csv(ticker_data_filename)

# construct the model
model = create_model(N_STEPS, len(FEATURE_COLUMNS), loss=LOSS, units=UNITS, cell=CELL, n_layers=N_LAYERS,
                    dropout=DROPOUT, optimizer=OPTIMIZER, bidirectional=BIDIRECTIONAL)

# some tensorflow callbacks
checkpointer = ModelCheckpoint(os.path.join("results", model_name + ".h5"), save_weights_only=True, save_best_only=True, verbose=1)
tensorboard = TensorBoard(log_dir=os.path.join("logs", model_name))
# train the model and save the weights whenever we see
# a new optimal model using ModelCheckpoint
history = model.fit(data["X_train"], data["y_train"],
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    validation_data=(data["X_test"], data["y_test"]),
                    callbacks=[checkpointer, tensorboard],
                    verbose=1)
```

```
Epoch 1/500
85/85 [==============================] - ETA: 0s - loss: 0.0033 - mean_absolute_error: 0.0316
Epoch 1: val_loss improved from inf to 0.00033, saving model to results/2024-06-18_AMZN-sh-1-sc-1-sbd-0-huber_loss-adam-LSTM-seq-50-step-15-layers-2-units-256.h5
85/85 [==============================] - 55s 564ms/step - loss: 0.0033 - mean_absolute_error: 0.0316 - val_loss: 3.2943e-04 - val_mean_absolute_error: 0.0122
Epoch 2/500
18/85 [=====>........................] - ETA: 36s - loss: 8.0592e-04 - mean_absolute_error: 0.0200
```

Figure 14: The training duration when we set to 500 epochs.

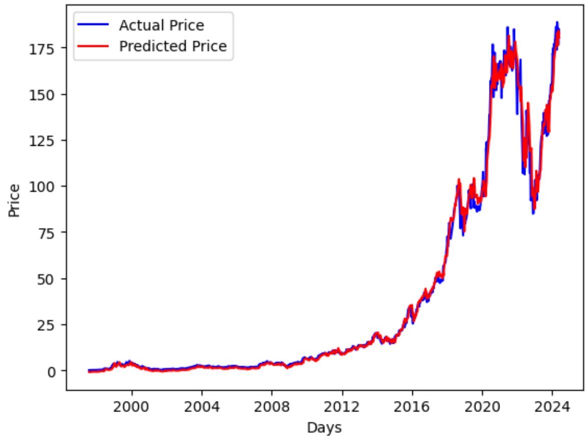
- After we set the epochs to 15 and train the model, we print the chart and results by running the test.py.

```
                    verbose=1)
```

```
Epoch 1/15
85/85 [==============================] - ETA: 0s - loss: 0.0022 - mean_absolute_error: 0.0294
Epoch 1: val_loss improved from inf to 0.00041, saving model to results/2024-06-18_AMZN-sh-1-sc-1-sbd-0-huber_loss-adam-LSTM-seq-50-step-15-layers-2-units-256.h5
85/85 [==============================] - 53s 566ms/step - loss: 0.0022 - mean_absolute_error: 0.0294 - val_loss: 4.0662e-04 - val_mean_absolute_error: 0.0136
Epoch 2/15
85/85 [==============================] - ETA: 0s - loss: 8.0580e-04 - mean_absolute_error: 0.0202
Epoch 2: val_loss did not improve from 0.00041
85/85 [==============================] - 47s 556ms/step - loss: 8.0580e-04 - mean_absolute_error: 0.0202 - val_loss: 4.8817e-04 - val_mean_absolute_error: 0.0162
Epoch 3/15
85/85 [==============================] - ETA: 0s - loss: 7.7261e-04 - mean_absolute_error: 0.0194
Epoch 3: val_loss improved from 0.00041 to 0.00038, saving model to results/2024-06-18_AMZN-sh-1-sc-1-sbd-0-huber_loss-adam-LSTM-seq-50-step-15-layers-2-units-256.h5
85/85 [==============================] - 54s 635ms/step - loss: 7.7261e-04 - mean_absolute_error: 0.0194 - val_loss: 3.7964e-04 - val_mean_absolute_error: 0.0127
Epoch 4/15
85/85 [==============================] - ETA: 0s - loss: 7.8805e-04 - mean_absolute_error: 0.0198
Epoch 4: val_loss did not improve from 0.00038
85/85 [==============================] - 47s 559ms/step - loss: 7.8805e-04 - mean_absolute_error: 0.0198 - val_loss: 7.4042e-04 - val_mean_absolute_error: 0.0207
Epoch 5/15
85/85 [==============================] - ETA: 0s - loss: 7.7583e-04 - mean_absolute_error: 0.0195
Epoch 5: val_loss did not improve from 0.00038
85/85 [==============================] - 47s 556ms/step - loss: 7.7583e-04 - mean_absolute_error: 0.0195 - val_loss: 4.4636e-04 - val_mean_absolute_error: 0.0140
Epoch 6/15
85/85 [==============================] - ETA: 0s - loss: 8.0781e-04 - mean_absolute_error: 0.0202
Epoch 6: val_loss did not improve from 0.00038
85/85 [==============================] - 47s 553ms/step - loss: 8.0781e-04 - mean_absolute_error: 0.0202 - val_loss: 4.3228e-04 - val_mean_absolute_error: 0.0141
Epoch 7/15
85/85 [==============================] - ETA: 0s - loss: 7.2338e-04 - mean_absolute_error: 0.0193
Epoch 7: val_loss did not improve from 0.00038
85/85 [==============================] - 52s 611ms/step - loss: 7.2338e-04 - mean_absolute_error: 0.0193 - val_loss: 3.8263e-04 - val_mean_absolute_error: 0.0129
Epoch 8/15
85/85 [==============================] - ETA: 0s - loss: 7.3848e-04 - mean_absolute_error: 0.0191
Epoch 8: val_loss did not improve from 0.00038
85/85 [==============================] - 50s 595ms/step - loss: 7.3848e-04 - mean_absolute_error: 0.0191 - val_loss: 3.8806e-04 - val_mean_absolute_error: 0.0148
Epoch 9/15
85/85 [==============================] - ETA: 0s - loss: 6.5371e-04 - mean_absolute_error: 0.0183
Epoch 9: val_loss did not improve from 0.00038
85/85 [==============================] - 50s 585ms/step - loss: 6.5371e-04 - mean_absolute_error: 0.0183 - val_loss: 4.0242e-04 - val_mean_absolute_error: 0.0134
Epoch 10/15
85/85 [==============================] - ETA: 0s - loss: 7.4229e-04 - mean_absolute_error: 0.0190
Epoch 10: val_loss did not improve from 0.00038
85/85 [==============================] - 50s 584ms/step - loss: 7.4229e-04 - mean_absolute_error: 0.0190 - val_loss: 5.0126e-04 - val_mean_absolute_error: 0.0175
Epoch 11/15
85/85 [==============================] - ETA: 0s - loss: 6.7105e-04 - mean_absolute_error: 0.0188
Epoch 11: val_loss improved from 0.00038 to 0.00037, saving model to results/2024-06-18_AMZN-sh-1-sc-1-sbd-0-huber_loss-adam-LSTM-seq-50-step-15-layers-2-units-256.h5
85/85 [==============================] - 50s 587ms/step - loss: 6.7105e-04 - mean_absolute_error: 0.0188 - val_loss: 3.7237e-04 - val_mean_absolute_error: 0.0133
Epoch 12/15
85/85 [==============================] - ETA: 0s - loss: 6.6790e-04 - mean_absolute_error: 0.0186
Epoch 12: val_loss did not improve from 0.00037
85/85 [==============================] - 48s 571ms/step - loss: 6.6790e-04 - mean_absolute_error: 0.0186 - val_loss: 4.6430e-04 - val_mean_absolute_error: 0.0140
Epoch 13/15
85/85 [==============================] - ETA: 0s - loss: 6.6271e-04 - mean_absolute_error: 0.0185
Epoch 13: val_loss did not improve from 0.00037
85/85 [==============================] - 50s 595ms/step - loss: 6.6271e-04 - mean_absolute_error: 0.0185 - val_loss: 3.8967e-04 - val_mean_absolute_error: 0.0130
Epoch 14/15
85/85 [==============================] - ETA: 0s - loss: 6.3052e-04 - mean_absolute_error: 0.0180
Epoch 14: val_loss did not improve from 0.00037
85/85 [==============================] - 56s 655ms/step - loss: 6.3052e-04 - mean_absolute_error: 0.0180 - val_loss: 3.7513e-04 - val_mean_absolute_error: 0.0130
Epoch 15/15
85/85 [==============================] - ETA: 0s - loss: 6.0415e-04 - mean_absolute_error: 0.0177
```

Figure 15: The training duration when we set to 15 epochs.

```
43/43 [==============================] - 5s 91ms/step
1/1 [==============================] - 0s 37ms/step
Future price after 15 days is 181.66$
huber_loss loss: 0.0003250316367484629
Mean Absolute Error: 2.527136243503515
Accuracy score: 0.5077720207253886
Total buy profit: 699.8878648281097
Total sell profit: -38.04809225350619
Total profit: 661.8397725746036
Profit per trade: 0.48988880279393304
```




```
                 open        high         low       close    adjclose  \
2024-03-28  180.169998  181.699997  179.259995  180.380005  180.380005
2024-04-04  184.000000  185.100006  180.000000  180.000000  180.000000
2024-04-05  182.380005  186.270004  181.970001  185.070007  185.070007
2024-04-08  186.899994  187.289993  184.809998  185.190002  185.190002
2024-04-15  187.429993  188.690002  183.000000  183.619995  183.619995
2024-04-29  182.750000  183.529999  179.389999  180.960007  180.960007
```

Figure 16: The prediction chart when after train model set to 15 epochs.

III. The understanding of the code base v0.1:

1. Import libraries.

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
import datetime as dt
import tensorflow as tf
import yfinance as yf

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, InputLayer
```

Figure 17: Importing the libaries.

- Before we start running the code, we need to implement the libraries. These are the libraries required for date-time modification (datetime), machine learning (tensorflow, keras), data visualization (matplotlib), and data management (numpy, pandas, pandas_datareader). The purpose of yfinance is to get historical stock data.

2. Set data source and company.

```
DATA_SOURCE = "yahoo"
COMPANY = "TSLA"
```

Figure 18: Setting the data resources from yahoo and get the TESLA data.

- To retrieve the stock data for Tesla, “DATA_SOURCE” is set to "yahoo" (using Yahoo Finance), and “COMPANY” is set to "TSLA" (Tesla).

3. Define training period and fetch data.

```
TRAIN_START = '2015-01-01'
TRAIN_END = '2020-01-01'

data =  yf.download(COMPANY, start=TRAIN_START, end=TRAIN_END, progress=False)
```

Figure 19: Setting the time to train and fetch the TESLA data.

- The data source, the ticker symbol of the business whose stock values are being examined, and the date ranges for the training and testing datasets are all defined in this section using constants. The stock data will be retrieved from Yahoo Finance, as indicated by the setting of “DATA_SOURCE” to "yahoo".

4. Prepare the data.

```
[ ] PRICE_VALUE = "Close"

    scaler = MinMaxScaler(feature_range=(0, 1))

    scaled_data = scaler.fit_transform(data[PRICE_VALUE].values.reshape(-1, 1))
```

Figure 20: Scaling the data.

- For predicted purposes, the stock's closing price is chosen. Using “MinMaxScaler”, the data is scaled to a range of 0 to 1, which enhances the performance of the model.

5. Prepare training data.

```
# Number of days to look back to base the prediction
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])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

Figure 21: Setting the training data part.

- The model forecasts the price for the following day using data spanning 60 days. Two arrays are created from the data: “x_train” (input) and “y_train” (output). As required by the LSTM model, the “x_train” array is reshaped to be three-dimensional.

6. Build the model.

```
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)
```

Figure 22: Building the model after training.

- The Sequential API from Keras is used to build the model. To avoid overfitting, it is composed of three LSTM layers, each of which is followed by a Dropout layer. The anticipated stock price is produced by the last Dense layer. The Mean Squared Error loss function and the Adam optimizer are used to construct the model. After that, it is trained for 25 epochs with a batch size of 32 using the training data.

7. Load and prepare test data.

```
# Load the test data
TEST_START = '2020-01-02'
TEST_END = '2022-12-31'

test_data = yf.download(COMPANY, start=TRAIN_START, end=TRAIN_END, progress=False)

test_data = test_data[1:]

actual_prices = test_data[PRICE_VALUE].values

total_dataset = pd.concat((data[PRICE_VALUE], test_data[PRICE_VALUE]), axis=0)

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

Figure 23: Loading the test data.

- Similar to how the training data is retrieved and prepared, as well is the test data. For the duration of the test, “actual_prices” contains the actual stock prices. The scaled values needed for prediction are stored in the “model_inputs” array.

8. Prepare test inputs and predict.

```
x_test = []
for x in range(PREDICTION_DAYS, len(model_inputs)):
    x_test.append(model_inputs[x - PREDICTION_DAYS:x, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
# TO DO: Explain the above 5 lines

predicted_prices = model.predict(x_test)
predicted_prices = scaler.inverse_transform(predicted_prices)
```

Figure 24: Preparing the test input and begin to predict.

- In order to create input sequences for the LSTM model to make predictions, the test data is prepared in the same way as the training data. These sequences are initially stored in an empty list called “x_test”. For the test data, the stock prices are predicted by the trained LSTM model. Predicted prices are generated in a series based on the test sequences that have been developed. The “MinMaxScaler”'s inverse transform function is then used to scale these forecasted prices back to their initial range.

9. Visualize the results.

```
plt.plot(actual_prices, color="black", label=f"Actual {COMPANY} Price")
plt.plot(predicted_prices, color="green", label=f"Predicted {COMPANY} Price")
plt.title(f"{COMPANY} Share Price")
plt.xlabel("Time")
plt.ylabel(f"{COMPANY} Share Price")
plt.legend()
plt.show()
```

Figure 25: Printing the result and prediction chart.

- The script uses matplotlib.pyplot to plot the actual and predicted stock prices in order to evaluate the effectiveness of the model. Plotting the expected costs is done in green, and plotting the actual prices is done in black. The x- and y-axes of the plot are labeled "TSLA Share Price," in accordance with its title.

10. Predict the next day.

Time

```
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))

prediction = model.predict(real_data)
prediction = scaler.inverse_transform(prediction)
print(f"Prediction: {prediction}")
```

```
1/1 [==============================] - 0s 62ms/step
Prediction: [[28.212034]]
```

Figure 26: Printing the prediction for the next day.

- Finally, following the test period, the model predicts the stock price for the following day. This illustrates how the model may forecast future events using the most recent data.