

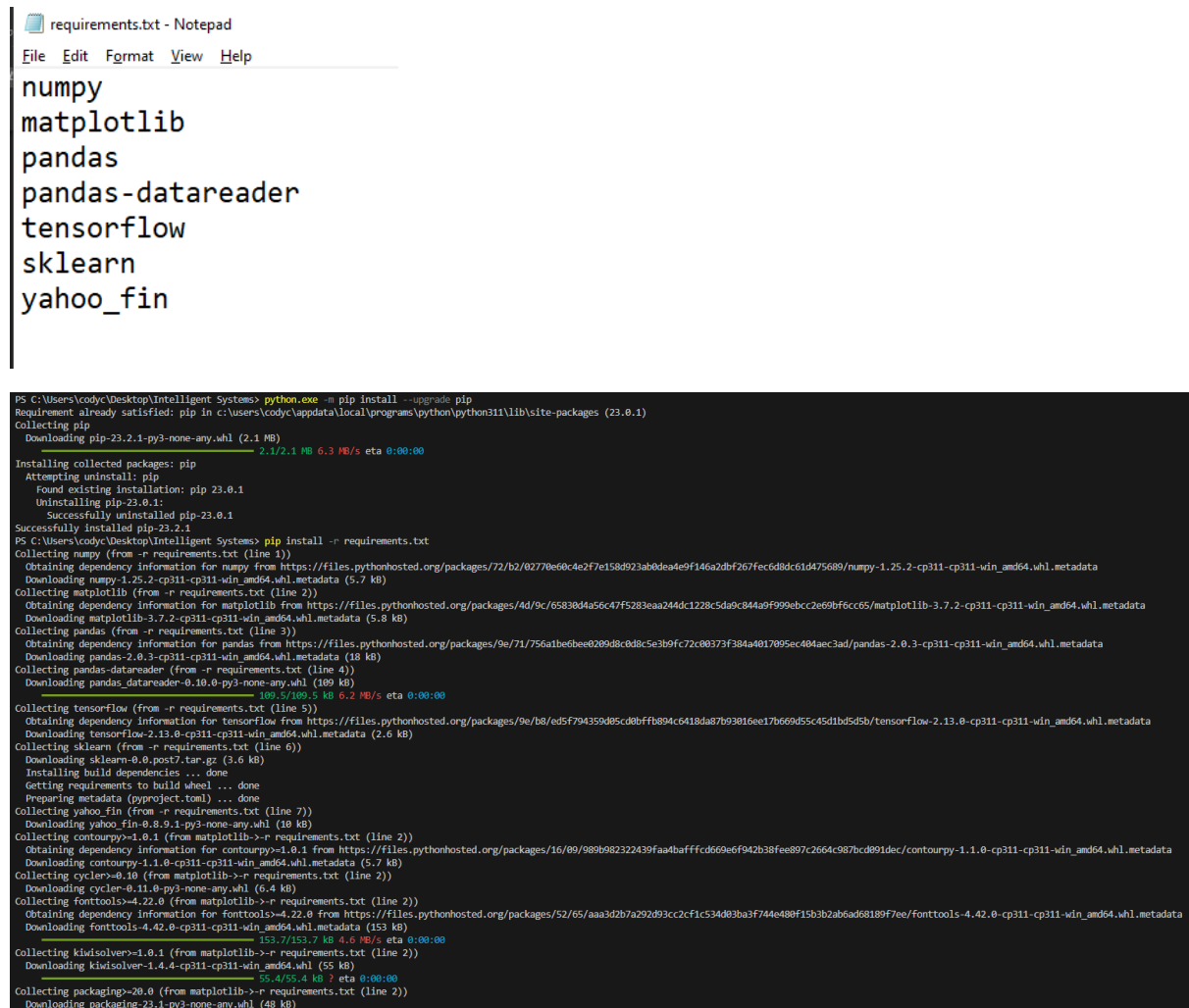
COS30018- Option B- Task 1: Setup

Cody Cronin-Sporys

103610020

The v0.1 Process:

The first step was to download the code file (v0.1) and follow along with the tutorial. I already had python setup in vs code so the first problem I came across was importing all the libraries. I read the information on requirements files, so I wrote down all the missing libraries into a text file. I also check the P1 requirements file and added those libraries as well before running pip install requirements.txt.



```
requirements.txt - Notepad
File Edit Format View Help
numpy
matplotlib
pandas
pandas-datareader
tensorflow
sklearn
yahoo_fin

PS C:\Users\cody\Desktop\Intelligent Systems> python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\cody\appdata\local\programs\python\python311\lib\site-packages (23.0.1)
Collecting pip
  Downloading pip-23.2.1-py3-none-any.whl (2.1 MB)
    2.1/2.1 MB 6.3 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.0.1
    Uninstalling pip-23.0.1:
      Successfully uninstalled pip-23.0.1
  Successfully installed pip-23.2.1
PS C:\Users\cody\Desktop\Intelligent Systems> pip install -r requirements.txt
Collecting numpy (from -r requirements.txt (line 1))
  Obtaining dependency information for numpy from https://files.pythonhosted.org/packages/72/b2/02770e60c4e2f7e15d923ab0de4e9f146a2dbf267fec6d8dc61d475689/numpy-1.25.2-cp311-cp311-win_amd64.whl.metadata
  Downloading numpy-1.25.2-cp311-cp311-win_amd64.whl.metadata (5.7 kB)
Collecting matplotlib (from -r requirements.txt (line 2))
  Obtaining dependency information for matplotlib from https://files.pythonhosted.org/packages/4d/9c/65830d4a56c47f5283aaa244dc1228c5da9c844a9f999ebcc2e69bfec65/matplotlib-3.7.2-cp311-cp311-win_amd64.whl.metadata
  Downloading matplotlib-3.7.2-cp311-cp311-win_amd64.whl.metadata (5.8 kB)
Collecting pandas (from -r requirements.txt (line 3))
  Obtaining dependency information for pandas from https://files.pythonhosted.org/packages/9e/71/756a1be6bee0209d8c8d8c5e3b9fc72c08373f384a4017095ec404aec3ad/pandas-2.0.3-cp311-cp311-win_amd64.whl.metadata
  Downloading pandas-2.0.3-cp311-cp311-win_amd64.whl.metadata (18 kB)
Collecting pandas-datareader (from -r requirements.txt (line 4))
  Downloading pandas_datareader-0.10.0-py3-none-any.whl (109 kB)
    109/37109.5 kB 6.2 MB/s eta 0:00:00
Collecting tensorflow (from -r requirements.txt (line 5))
  Obtaining dependency information for tensorflow from https://files.pythonhosted.org/packages/9e/b8/ed5f794350d85cd8bffb894c6418da87693016ee17b669d55c45d1bd5d5b/tensorflow-2.13.0-cp311-cp311-win_amd64.whl.metadata
  Downloading tensorflow-2.13.0-cp311-cp311-win_amd64.whl.metadata (2.6 kB)
Collecting sklearn (from -r requirements.txt (line 6))
  Downloading sklearn-0.0.post7.tar.gz (3.6 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting yahoo_fin (from -r requirements.txt (line 7))
  Downloading yahoo_fin-0.8.9.1-py3-none-any.whl (10 kB)
Collecting contourpy>=1.0.1 (from matplotlib->-r requirements.txt (line 2))
  Obtaining dependency information for contourpy>=1.0.1 from https://files.pythonhosted.org/packages/16/09/989b982322439faa4baffc6d69e6f942b38fee897c2664c987bc091dec/contourpy-1.1.0-cp311-cp311-win_amd64.whl.metadata
  Downloading contourpy-1.1.0-cp311-cp311-win_amd64.whl.metadata (5.7 kB)
Collecting cyclo>=0.10 (from matplotlib->-r requirements.txt (line 2))
  Downloading cyclo-0.11.0-py3-none-any.whl (6.4 kB)
Collecting fonttools>=4.22.0 (from matplotlib->-r requirements.txt (line 2))
  Obtaining dependency information for fonttools>=4.22.0 from https://files.pythonhosted.org/packages/52/65/aaa3d2b7a292d93cc2cf1c53403ba3f744e40f15b3b2ab6ad68189f7ee/fonttools-4.42.0-cp311-cp311-win_amd64.whl.metadata
  Downloading fonttools-4.42.0-cp311-cp311-win_amd64.whl.metadata (153 kB)
    153.7/153.7 kB 4.6 MB/s eta 0:00:00
Collecting kiwisolver>=1.0.1 (from matplotlib->-r requirements.txt (line 2))
  Downloading kiwisolver-1.4.4-cp311-cp311-win_amd64.whl (55 kB)
    55.4/55.4 kB ? eta 0:00:00
Collecting packaging>=20.0 (from matplotlib->-r requirements.txt (line 2))
  Downloading packaging-23.1-py3-none-any.whl (48 kB)
```

After first updating pip, the libraries installed successfully. Despite this, the code was still showing the same errors for all the libraries.

```
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
```

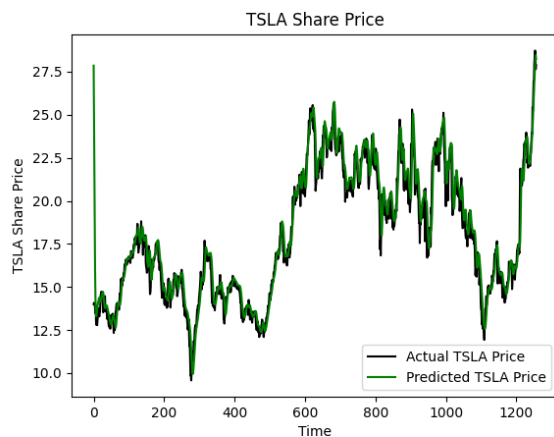
After reloading and making sure the workspace and code file were trusted, most of the issues resolved themselves. A few of the libraries (like yfinance) were still showing errors because I didn't have them added to requirements, so I updated and re-ran the requirements file.

This fixed all the imports except the bottom 2. Both "from tensorflow.keras..." were still showing issues however the code seemed to be able to run. I looked around for fixes but couldn't find any so for now I'll leave it as is and come back if it causes problems.

After running the code, it completed 25 steps of epoch in the console, then produced this graph and a prediction value.

```
38/38 [=====] - 2s 51ms/step - loss: 0.0032
Epoch 22/25
38/38 [=====] - 2s 52ms/step - loss: 0.0033
Epoch 23/25
38/38 [=====] - 2s 55ms/step - loss: 0.0036
Epoch 24/25
38/38 [=====] - 2s 55ms/step - loss: 0.0038
Epoch 25/25
38/38 [=====] - 2s 60ms/step - loss: 0.0031
40/40 [=====] - 3s 20ms/step
```

Figure 1



Navigation icons: home, back, forward, search, and save.

```
17/1 [
Prediction: [[27.55825]]
```

The graph shows predicted prices, with one day predicted price being generated from 60 days of existing data. The prices being predicted in this graph are in the past, so we already have real data to compare with, which is also shown on the graph. This allows us to check the accuracy of the prediction.

The single prediction value shown in the console is an actual future prediction on data we do not have yet.

Understanding of v0.1 code:

In the code, The first step is obtaining the raw data which is done in this code section:

```
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)
```

Next we need to manipulate the data so that it's usable for training our model. We do this through scaling and reshaping the data. We also store a limited amount of the data based on the amount of days we are predicting from.

```
scaler = MinMaxScaler(feature_range=(0, 1))
# Note that, by default, feature_range=(0, 1). Thus, if you want a different
# feature_range (min,max) then you'll need to specify it here
scaled_data = scaler.fit_transform(data[PRICE_VALUE].values.reshape(-1, 1))
```

```
for x in range(PREDICTION_DAYS, len(scaled_data)):
    x_train.append(scaled_data[x-PREDICTION_DAYS:x])
    y_train.append(scaled_data[x])
```

Next we need to build the model. In this code we are using a layered LSTM model.

```
model.add(Dropout(0.2))
# The Dropout layer randomly sets input units to
# rate (= 0.2 above) at each step during training
# prevent overfitting (one of the major problems

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

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

model.add(Dense(units=1))
```

We also setup various parameters for the model, such as the optimizer and loss.

```
model = Sequential()
```

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

Then we train the model using the data we prepared earlier.

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

Once the model has been trained, we start testing it. The first test is comparing it against existing data so we can judge it's accuracy. The first step is again downloading and setting up the data for our model.

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

# The above bug is the reason for the following line of code
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
# We need to do the above because to predict the closing price of the first
# PREDICTION_DAYS of the test period [TEST_START, TEST_END], we'll need the
# data from the training period

model_inputs = model_inputs.reshape(-1, 1)
# TO DO: Explain the above line

model_inputs = scaler.transform(model_inputs)
```

We then use this data and model to prepare a test and predict the next day's prices. The prediction also needs to have its transformations reverse so it's readable.

```
#
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)
# TO DO: Explain the above 5 lines
```

To visualise this data and the prediction we setup a plot with various labels and the two datasets (actual and predicted).

```
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()
```

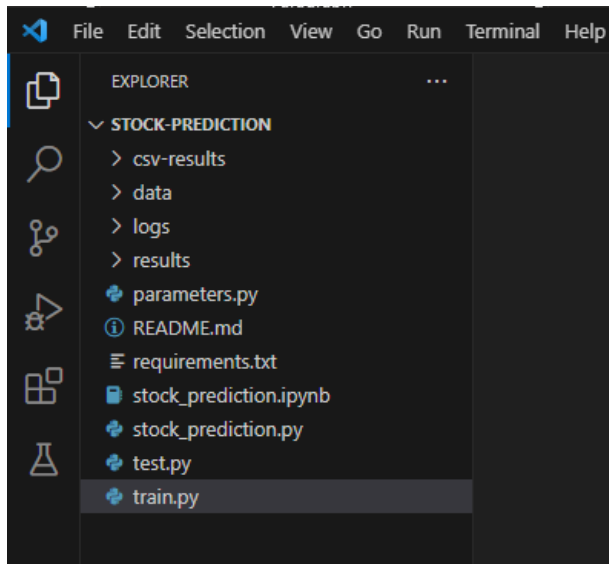
The final thing we do is a similar prediction, but this time it's not on existing data. Instead of showing the results in a graph/ plot we just print the result to console.

```
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}")
```

The P1 Process:

For the P1 code base I downloaded a zipped of the pythoncode-tutorials repo and extracted the stock-prediction folder. I then opened this folder in visual studio code. Since all the libraries were installed for the v0.1 code base there's no need to do it again here.



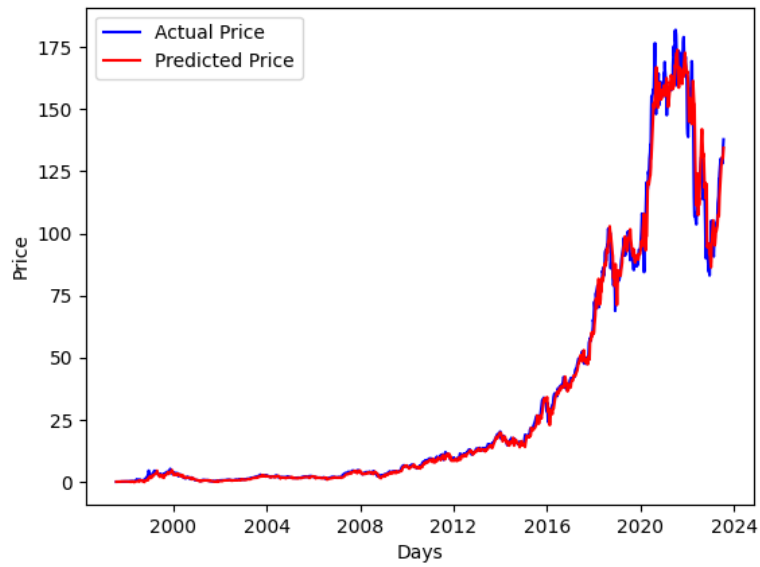
The first step is to edit the parameters.py if needed then start running the train.py code. After starting it with the default parameters I realised the batch size and number of steps was much larger than v0.1 so I reduced it down to save time, since this demonstration doesn't need super accurate results.

```
OPTIMIZER = "adam"
BATCH_SIZE = 32
EPOCHS = 25
```

```
164/164 [=====] - ETA: 0s - loss: 5.9471e-04 - mean_absolute_error: 0.0191
Epoch 20: val_loss did not improve from 0.00032
164/164 [=====] - 43s 260ms/step - loss: 5.9471e-04 - mean_absolute_error: 0.0191 - val_loss: 3.1809e-04 - val_mean_absolute_error: 0.0116
Epoch 21/25
164/164 [=====] - ETA: 0s - loss: 6.5370e-04 - mean_absolute_error: 0.0196
Epoch 21: val_loss did not improve from 0.00032
164/164 [=====] - 42s 256ms/step - loss: 6.5370e-04 - mean_absolute_error: 0.0196 - val_loss: 3.2177e-04 - val_mean_absolute_error: 0.0113
Epoch 22/25
164/164 [=====] - ETA: 0s - loss: 6.4276e-04 - mean_absolute_error: 0.0199
Epoch 22: val_loss did not improve from 0.00032
164/164 [=====] - 43s 263ms/step - loss: 6.4276e-04 - mean_absolute_error: 0.0199 - val_loss: 4.9596e-04 - val_mean_absolute_error: 0.0149
Epoch 23/25
164/164 [=====] - ETA: 0s - loss: 5.9594e-04 - mean_absolute_error: 0.0196
Epoch 23: val_loss did not improve from 0.00032
164/164 [=====] - 43s 262ms/step - loss: 5.9594e-04 - mean_absolute_error: 0.0196 - val_loss: 3.2465e-04 - val_mean_absolute_error: 0.0125
Epoch 24/25
164/164 [=====] - ETA: 0s - loss: 5.6646e-04 - mean_absolute_error: 0.0187
Epoch 24: val_loss did not improve from 0.00032
164/164 [=====] - 43s 261ms/step - loss: 5.6646e-04 - mean_absolute_error: 0.0187 - val_loss: 3.2224e-04 - val_mean_absolute_error: 0.0121
Epoch 25/25
164/164 [=====] - ETA: 0s - loss: 5.7713e-04 - mean_absolute_error: 0.0192
Epoch 25: val_loss did not improve from 0.00032
164/164 [=====] - 43s 263ms/step - loss: 5.7713e-04 - mean_absolute_error: 0.0192 - val_loss: 4.5252e-04 - val_mean_absolute_error: 0.0154
PS C:\Users\codyc\Desktop\Intelligent Systems\p1\stock-prediction>
```

Now that the training is done we can run test.py to see the results of the model. This will show a similar graph to v0.1 and also some extra information on the prediction in the console.

Figure 1



```
Future price after 15 days is 139.36$  
huber_loss loss: 0.0003173926961608231  
Mean Absolute Error: 2.2116098345270614  
Accuracy score: 0.5042016806722689  
Total buy profit: 433.24552346766006  
Total sell profit: 172.98038519918916  
Total profit: 606.2259086668493  
Profit per trade: 0.4631213969953012
```