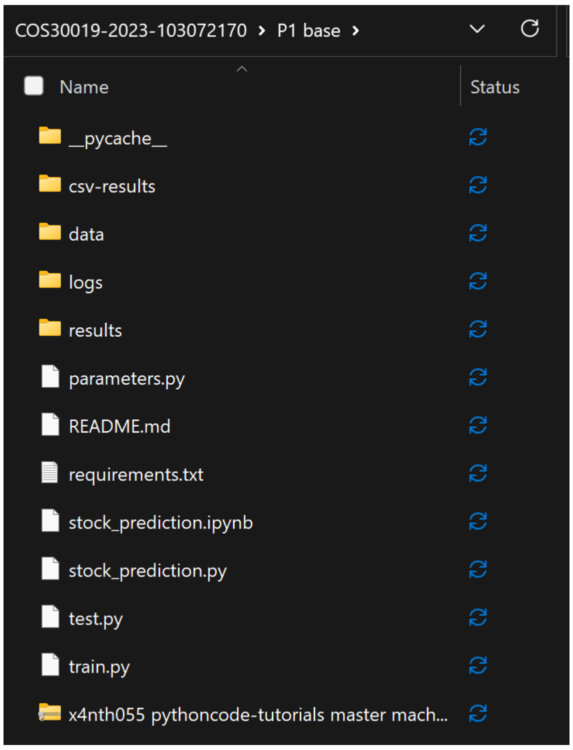
Task 1 Report

NACA HITCHMAN - 103072170

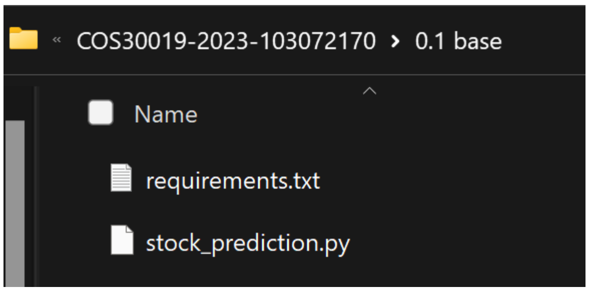
Environment Setup

I started by downloading the 0.1 and P1 code bases, storing them in their own directories.



COS30019-2023-103072170 › P1 base ›

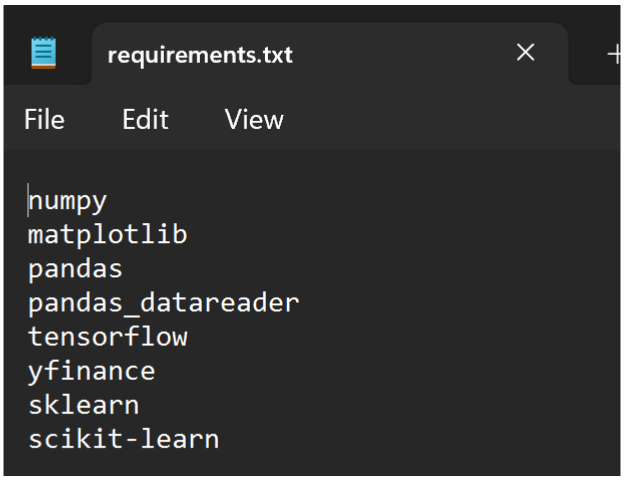
Name	Status
__pycache__	
csv-results	
data	
logs	
results	
parameters.py	
README.md	
requirements.txt	
stock_prediction.ipynb	
stock_prediction.py	
test.py	
train.py	
x4nth055 pythoncode-tutorials master mach...	



COS30019-2023-103072170 › 0.1 base

Name
requirements.txt
stock_prediction.py

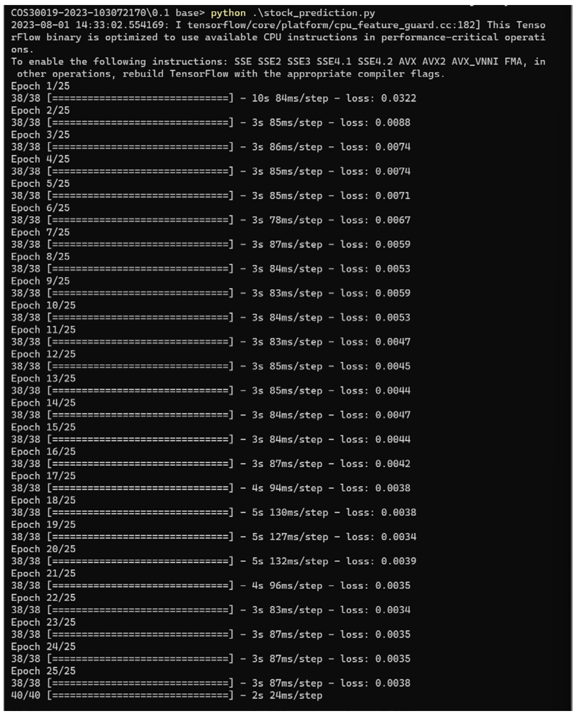
I then viewed the code of and ran the 0.1 stock_prediction.py file to determine the dependencies that would need to be in the requirements.txt.



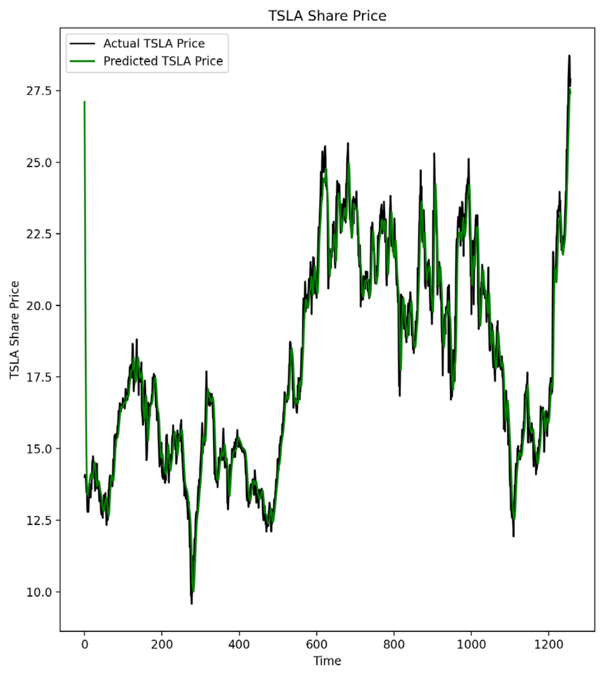
```
numpy
matplotlib
pandas
pandas_datareader
tensorflow
yfinance
sklearn
scikit-learn
```

Testing Code Bases

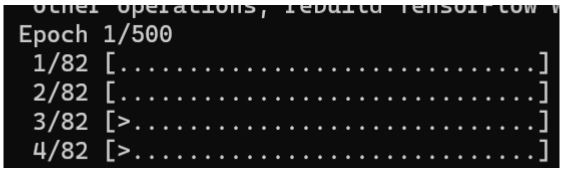
Running the 0.1 code base was relatively quick, going through only 25 epochs before opening the chart showing the actual and predicted TSLA stock price.



```
COS30019-2023-103072170\0.1 base> python .\stock_prediction.py
2023-08-01 14:33:02.554169: I tensorflow/core/platform/cpu_feature_guard.cc:182] This Tenso
rFlow binary is optimized to use available CPU instructions in performance-critical operati
ons.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in
 other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/25
38/38 [==============================] - 10s 84ms/step - loss: 0.0322
Epoch 2/25
38/38 [==============================] - 3s 85ms/step - loss: 0.0088
Epoch 3/25
38/38 [==============================] - 3s 86ms/step - loss: 0.0074
Epoch 4/25
38/38 [==============================] - 3s 85ms/step - loss: 0.0074
Epoch 5/25
38/38 [==============================] - 3s 85ms/step - loss: 0.0071
Epoch 6/25
38/38 [==============================] - 3s 78ms/step - loss: 0.0067
Epoch 7/25
38/38 [==============================] - 3s 87ms/step - loss: 0.0059
Epoch 8/25
38/38 [==============================] - 3s 84ms/step - loss: 0.0053
Epoch 9/25
38/38 [==============================] - 3s 83ms/step - loss: 0.0059
Epoch 10/25
38/38 [==============================] - 3s 84ms/step - loss: 0.0053
Epoch 11/25
38/38 [==============================] - 3s 83ms/step - loss: 0.0047
Epoch 12/25
38/38 [==============================] - 3s 85ms/step - loss: 0.0045
Epoch 13/25
38/38 [==============================] - 3s 85ms/step - loss: 0.0044
Epoch 14/25
38/38 [==============================] - 3s 84ms/step - loss: 0.0047
Epoch 15/25
38/38 [==============================] - 3s 84ms/step - loss: 0.0044
Epoch 16/25
38/38 [==============================] - 3s 87ms/step - loss: 0.0042
Epoch 17/25
38/38 [==============================] - 4s 94ms/step - loss: 0.0038
Epoch 18/25
38/38 [==============================] - 5s 130ms/step - loss: 0.0038
Epoch 19/25
38/38 [==============================] - 5s 127ms/step - loss: 0.0034
Epoch 20/25
38/38 [==============================] - 5s 132ms/step - loss: 0.0039
Epoch 21/25
38/38 [==============================] - 4s 96ms/step - loss: 0.0035
Epoch 22/25
38/38 [==============================] - 3s 83ms/step - loss: 0.0034
Epoch 23/25
38/38 [==============================] - 3s 87ms/step - loss: 0.0035
Epoch 24/25
38/38 [==============================] - 3s 87ms/step - loss: 0.0035
Epoch 25/25
38/38 [==============================] - 3s 87ms/step - loss: 0.0038
40/40 [==============================] - 2s 24ms/step
```

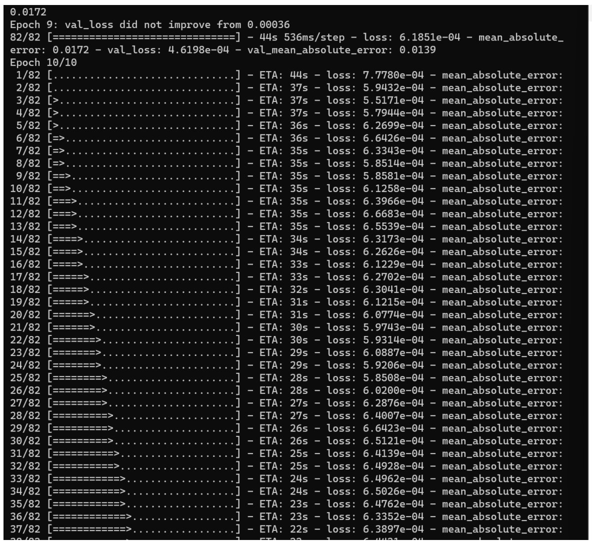


The initial run of the P1 code base, however, was set to iterate through 500 epochs, which would take significantly too long for the purposes of this task.

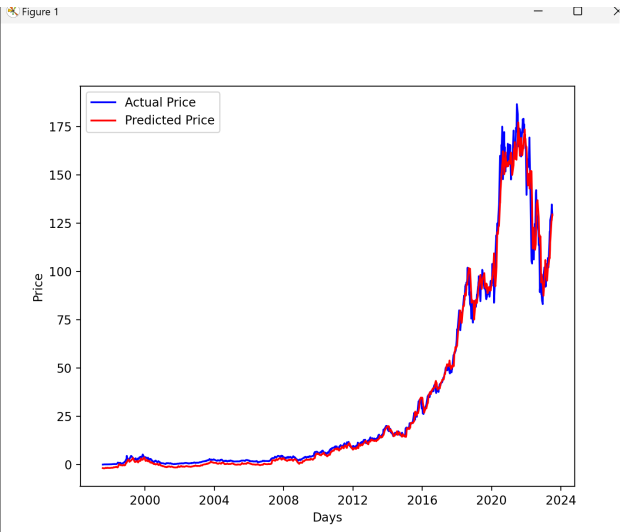


```
 other operations, rebuild TensorFlow w
Epoch 1/500
 1/82 [..............................]
 2/82 [..............................]
 3/82 [>.............................]
 4/82 [>.............................]
```

Thus, for this task I edited the parameters.py file to have the training only run through 10 epochs. I then ran the test.py file to build and view the results.



```
0.0172
Epoch 9: val_loss did not improve from 0.00036
82/82 [==============================] - 44s 536ms/step - loss: 6.1851e-04 - mean_absolute_
error: 0.0172 - val_loss: 4.6198e-04 - val_mean_absolute_error: 0.0139
Epoch 10/10
 1/82 [..............................] - ETA: 44s - loss: 7.7780e-04 - mean_absolute_error:
 2/82 [..............................] - ETA: 37s - loss: 5.9432e-04 - mean_absolute_error:
 3/82 [>.............................] - ETA: 37s - loss: 5.5171e-04 - mean_absolute_error:
 4/82 [>.............................] - ETA: 37s - loss: 5.7944e-04 - mean_absolute_error:
 5/82 [>.............................] - ETA: 36s - loss: 6.2699e-04 - mean_absolute_error:
 6/82 [=>............................] - ETA: 36s - loss: 6.6426e-04 - mean_absolute_error:
 7/82 [=>............................] - ETA: 35s - loss: 6.3343e-04 - mean_absolute_error:
 8/82 [=>............................] - ETA: 35s - loss: 5.8514e-04 - mean_absolute_error:
 9/82 [==>...........................] - ETA: 35s - loss: 5.8581e-04 - mean_absolute_error:
10/82 [==>...........................] - ETA: 35s - loss: 6.1258e-04 - mean_absolute_error:
11/82 [===>..........................] - ETA: 35s - loss: 6.3966e-04 - mean_absolute_error:
12/82 [===>..........................] - ETA: 35s - loss: 6.6683e-04 - mean_absolute_error:
13/82 [===>..........................] - ETA: 35s - loss: 6.5539e-04 - mean_absolute_error:
14/82 [====>.........................] - ETA: 34s - loss: 6.3173e-04 - mean_absolute_error:
15/82 [====>.........................] - ETA: 34s - loss: 6.2626e-04 - mean_absolute_error:
16/82 [====>.........................] - ETA: 33s - loss: 6.1229e-04 - mean_absolute_error:
17/82 [=====>........................] - ETA: 33s - loss: 6.2702e-04 - mean_absolute_error:
18/82 [=====>........................] - ETA: 32s - loss: 6.3041e-04 - mean_absolute_error:
19/82 [=====>........................] - ETA: 31s - loss: 6.1215e-04 - mean_absolute_error:
20/82 [======>.......................] - ETA: 31s - loss: 6.0774e-04 - mean_absolute_error:
21/82 [======>.......................] - ETA: 30s - loss: 5.9743e-04 - mean_absolute_error:
22/82 [=======>......................] - ETA: 30s - loss: 5.9314e-04 - mean_absolute_error:
23/82 [=======>......................] - ETA: 29s - loss: 6.0887e-04 - mean_absolute_error:
24/82 [=======>......................] - ETA: 29s - loss: 5.9206e-04 - mean_absolute_error:
25/82 [========>.....................] - ETA: 28s - loss: 5.8508e-04 - mean_absolute_error:
26/82 [========>.....................] - ETA: 28s - loss: 6.0200e-04 - mean_absolute_error:
27/82 [========>.....................] - ETA: 27s - loss: 6.2876e-04 - mean_absolute_error:
28/82 [=========>....................] - ETA: 27s - loss: 6.4007e-04 - mean_absolute_error:
29/82 [=========>....................] - ETA: 26s - loss: 6.6423e-04 - mean_absolute_error:
30/82 [=========>....................] - ETA: 26s - loss: 6.5121e-04 - mean_absolute_error:
31/82 [==========>...................] - ETA: 25s - loss: 6.4139e-04 - mean_absolute_error:
32/82 [==========>...................] - ETA: 25s - loss: 6.4928e-04 - mean_absolute_error:
33/82 [===========>..................] - ETA: 24s - loss: 6.4962e-04 - mean_absolute_error:
34/82 [===========>..................] - ETA: 24s - loss: 6.5026e-04 - mean_absolute_error:
35/82 [===========>..................] - ETA: 23s - loss: 6.4762e-04 - mean_absolute_error:
36/82 [============>.................] - ETA: 23s - loss: 6.3352e-04 - mean_absolute_error:
37/82 [============>.................] - ETA: 22s - loss: 6.3897e-04 - mean_absolute_error:
```



Understanding the 0.1 Code Base

After running both the 0.1 and P1 code bases, I investigated the python code of the 0.1 code base to attempt to understand how it works.

As is typical, it begins by importing the required dependencies and libraries to allow the system and machine learning to function.

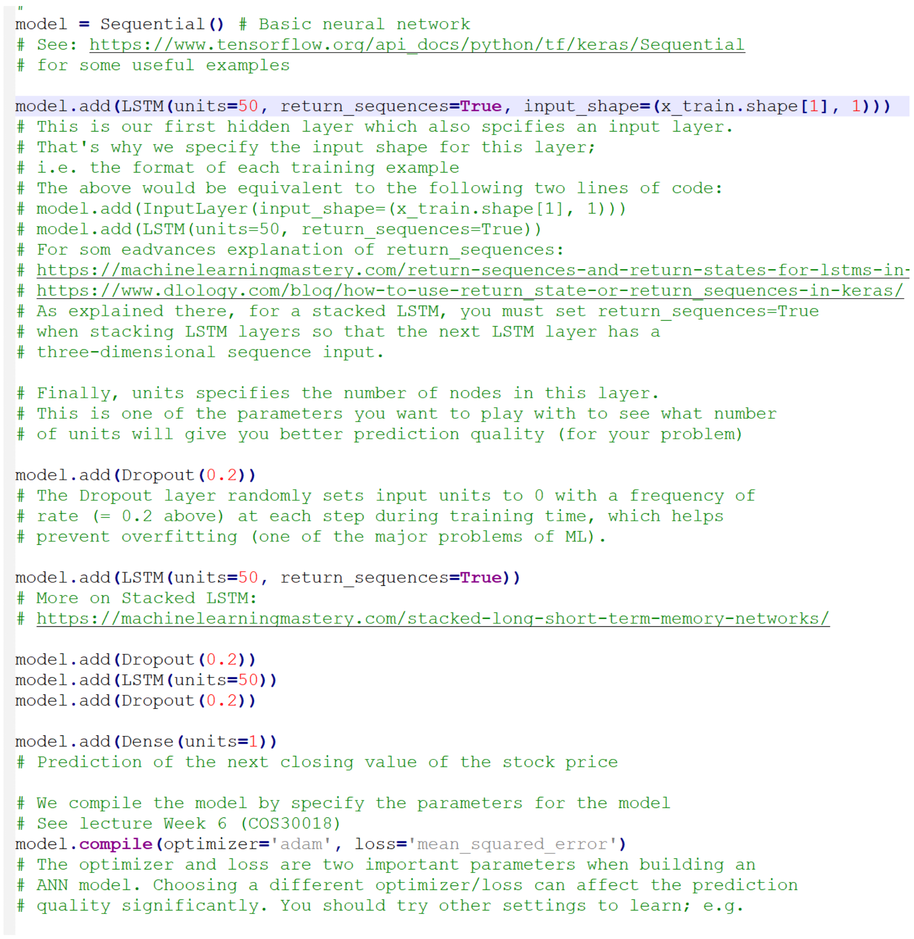
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
import datetime as dt
import tensorflow as tf
import yfinance as yf

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

Some global constants are then set including the source of the data, the stock name being predicted, and the dates to train the data on, before downloading the required data from the yfinance source.

```
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)
# yf.download(COMPANY, start = TRAIN_START, end=TRAIN_END)
```

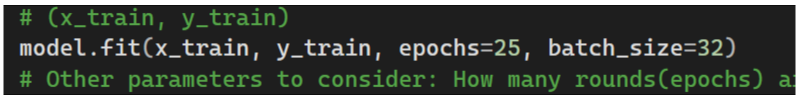
Following this, the data is prepared for usage, and the neural network model is created and its various properties and settings are defined. The model then gets compiled.

```
PRICE_VALUE = "Close"

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))
# Flatten and normalise the data
# First, we reshape a 1D array(n) to 2D array(n,1)
# We have to do that because sklearn.preprocessing.fit_transform()
# requires a 2D array
# Here n == len(scaled_data)
# Then, we scale the whole array to the range (0,1)
# The parameter -1 allows (np.)reshape to figure out the array size n automati
# values.reshape(-1, 1)
# https://stackoverflow.com/questions/18691084/what-does-1-mean-in-numpy-resha
# When reshaping an array, the new shape must contain the same number of eleme
# as the old shape, meaning the products of the two shapes' dimensions must be
# When using a -1, the dimension corresponding to the -1 will be the product o
# the dimensions of the original array divided by the product of the dimension
# given to reshape so as to maintain the same number of elements.

# 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

#------------------------------------------------------------------------------
```



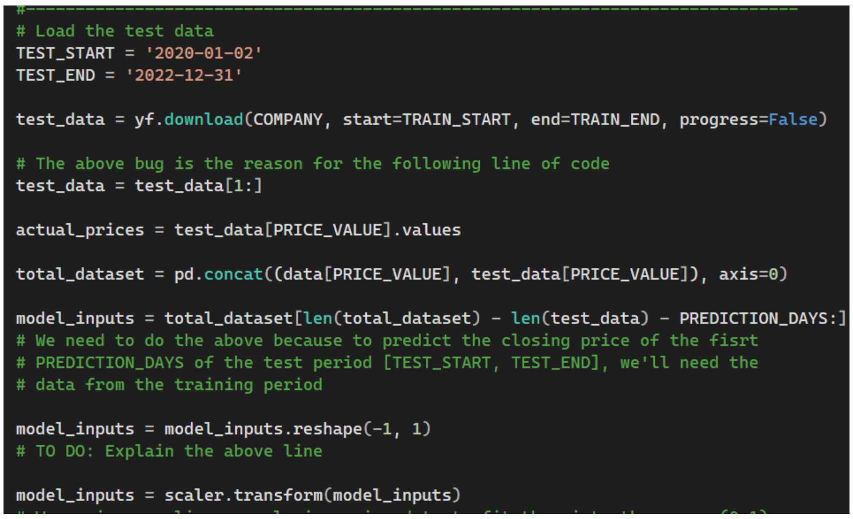
```
"
model = Sequential() # Basic neural network
# See: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential
# for some useful examples

model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
# This is our first hidden layer which also spcifies an input layer.
# That's why we specify the input shape for this layer;
# i.e. the format of each training example
# The above would be equivalent to the following two lines of code:
# model.add(InputLayer(input_shape=(x_train.shape[1], 1)))
# model.add(LSTM(units=50, return_sequences=True))
# For som eadvances explanation of return_sequences:
# https://machinelearningmastery.com/return-sequences-and-return-states-for-lstms-in-
# https://www.dlology.com/blog/how-to-use-return_state-or-return_sequences-in-keras/
# As explained there, for a stacked LSTM, you must set return_sequences=True
# when stacking LSTM layers so that the next LSTM layer has a
# three-dimensional sequence input.

# Finally, units specifies the number of nodes in this layer.
# This is one of the parameters you want to play with to see what number
# of units will give you better prediction quality (for your problem)

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')
# The optimizer and loss are two important parameters when building an
# ANN model. Choosing a different optimizer/loss can affect the prediction
# quality significantly. You should try other settings to learn; e.g.
```

The model is then trained



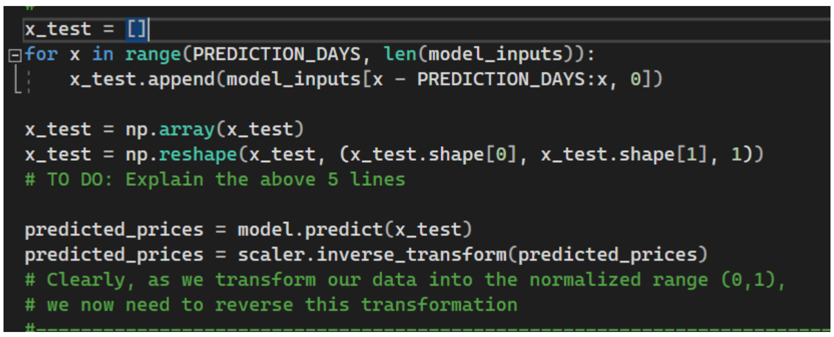
```
# (x_train, y_train)
model.fit(x_train, y_train, epochs=25, batch_size=32)
# Other parameters to consider: How many rounds(epochs) a
```

More recent test data is downloaded and formatted for usage to check the accuracy of the model



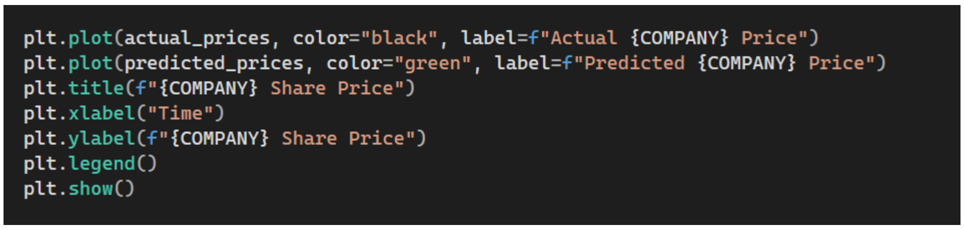
```
#------------------------------------------------------------------------------
# Load the test data
TEST_START = '2020-01-02'
TEST_END = '2022-12-31'

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

# 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:]
# We need to do the above because to predict the closing price of the fisrt
# 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)
```

The relevant amount of days' worth of data is converted to an array, before having the prediction run on it.



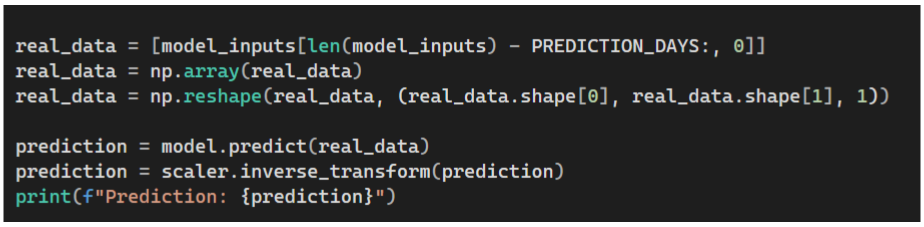
```python
x_test = []
for x in range(PREDICTION_DAYS, len(model_inputs)):
    x_test.append(model_inputs[x - PREDICTION_DAYS:x, 0])

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)
# Clearly, as we transform our data into the normalized range (0,1),
# we now need to reverse this transformation
#----------------------------------------------------------------------------
```

The prediction and actual prices are rendered onto a graph



```python
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()
```

Lastly, the next days’ price is also predicted and printed



```python
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}")
```