Task 1: Setup

Muhammad Ali, 103960437, COS30018, 16/08/2024.

Requirements / Deliverables –

- Download code bases for v0.1 and P1.
- Setup virtual environment to test code, download required files and libraries from requirements.txt.
- Debug code and make sure you can run it on your machine, train the models and get the results.
- Provide understanding of current v0.1 code base.

Setting up environment –

- Make sure to have python version 3.9 or lower installed.

Using an IDE of your choice create a new project with a virtual environment interpreter and set the python version to 3.9 or lower (important).

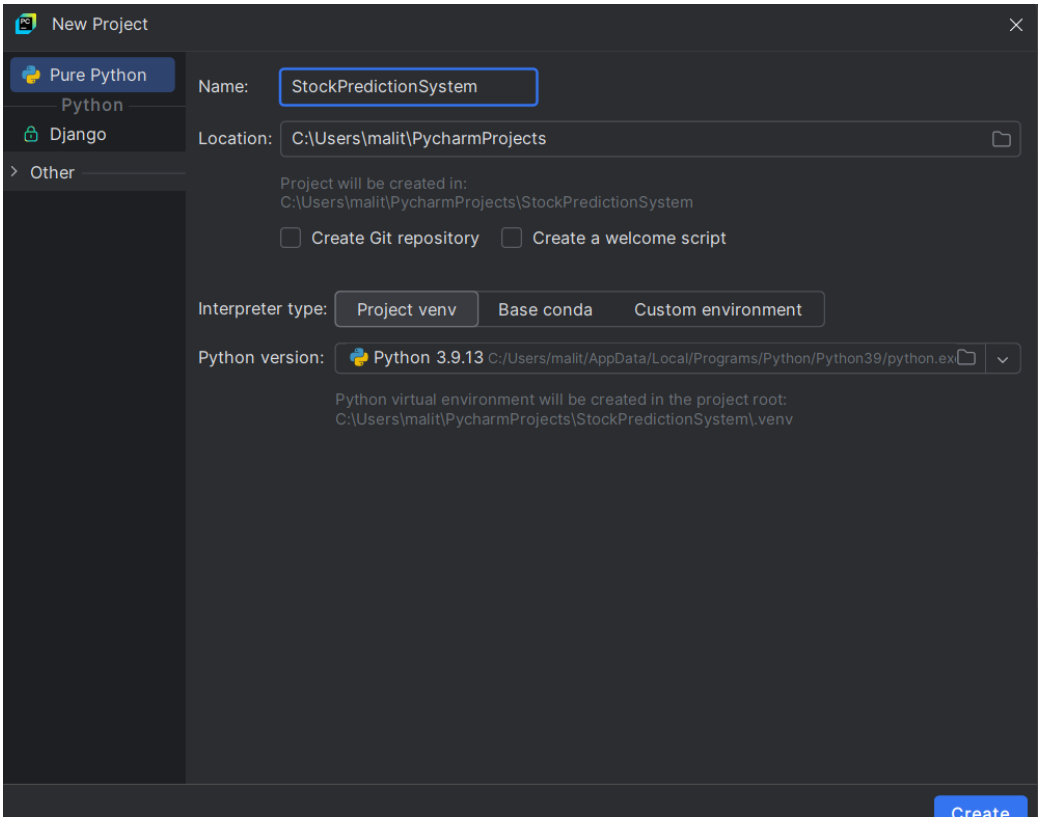
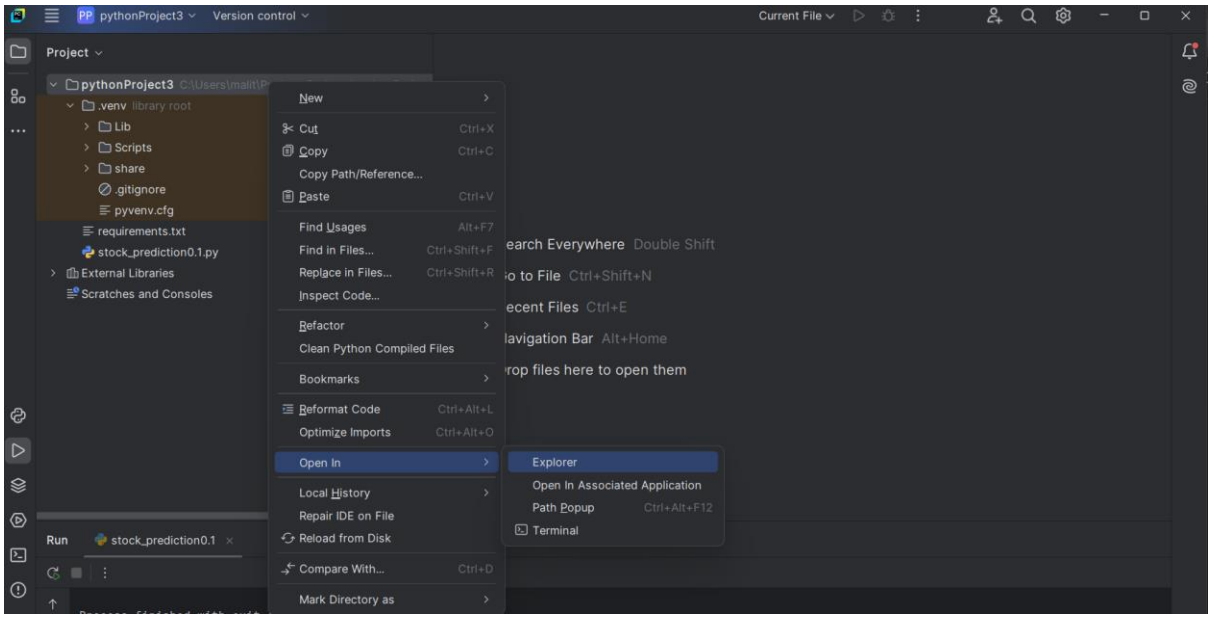


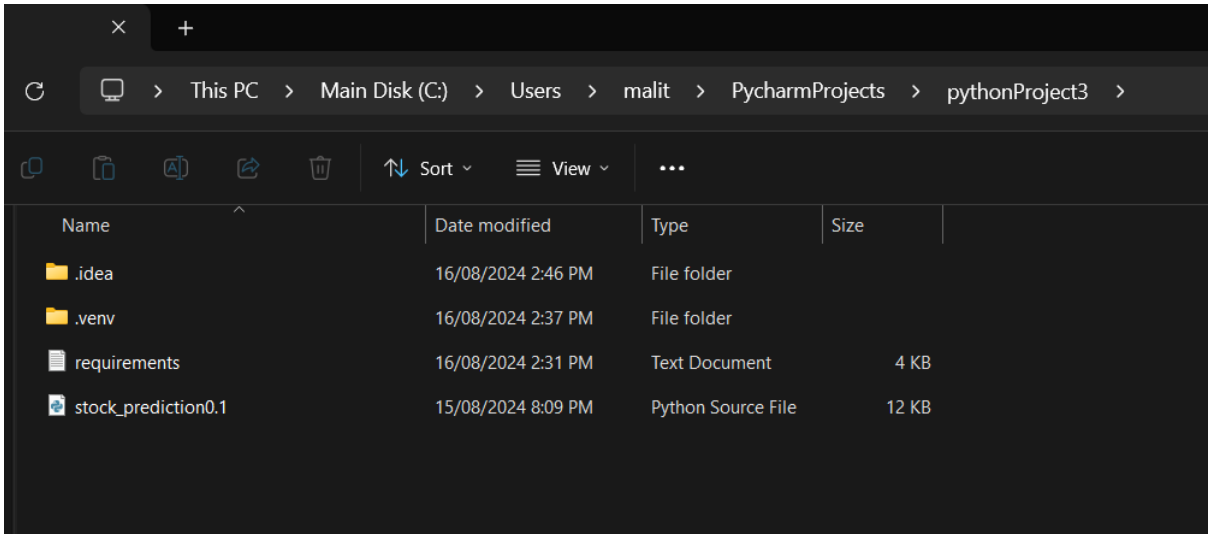
Figure 1. PyCharm Virtual Environment

Navigate to the task 1 folder in the repository and download the stock_prediction0.1.py file and requirements.txt.

In your IDE right click on your project and open it in explorer to get to the project directory.



Once in your project directory drag and drop the stock_prediction0.1.py and requirements.txt file into your project directory.



Once you've dragged the files into the project directory, go to your project terminal and make sure it's in the virtual environment:

```
(.venv) PS C:\Users\malit\PycharmProjects\pythonProject3>
```

Type the following command into the terminal:

```
(.venv) PS C:\Users\malit\PycharmProjects\pythonProject3> pip install -r requirements.txt
```

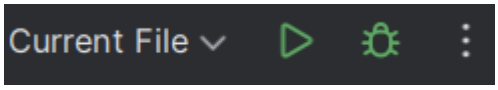
The terminal will download and install all the dependencies required to run the stock prediction file.

```
Installing collected packages: webencodings, sgmllib3k, pytz, peewee, parse, namex, multitasking, libclang, flatbuffers, appdirs, zipp, wrapt, websockets, w3lib,
 urllib3, tzdata, typing_extensions, threadpoolctl, termcolor, tensorflow-io-gcs-filesystem, tensorflow-estimator, tensorboard-data-server, soupsieve, six, pypar
sing, Pygments, pyasn1, protobuf, platformdirs, pillow, packaging, oauthlib, numpy, mdurl, MarkupSafe, lxml, kiwisolver, keras, joblib, idna, grpcio, gast, froze
ndict, fonttools, feedparser, cycler, cssselect, colorama, charset-normalizer, certifi, cachetools, absl-py, Werkzeug, tqdm, scipy, rsa, requests, python-dateuti
l, pyquery, pyee, pyasn1_modules, optree, opt-einsum, ml-dtypes, markdown-it-py, lxml_html_clean, Keras-Preprocessing, importlib_resources, importlib_metadata, h
tml5lib, h5py, google-pasta, contourpy, beautifulsoup4, astunparse, scikit-learn, rich, requests-oauthlib, pyppeteer, pandas, matplotlib, Markdown, google-auth,
fake-useragent, bs4, yfinance, requests-html, pandas-datareader, google-auth-oauthlib, yahoo-fin, tensorboard, tensorflow-intel, tensorflow
Successfully installed Keras-Preprocessing-1.1.2 Markdown-3.6 MarkupSafe-2.1.5 Pygments-2.18.0 Werkzeug-3.0.3 absl-py-2.1.0 appdirs-1.4.4 astunparse-1.6.3 beauti
fulsoup4-4.12.3 bs4-0.0.2 cachetools-5.4.0 certifi-2024.7.4 charset-normalizer-3.3.2 colorama-0.4.6 contourpy-1.2.1 cssselect-1.2.0 cycler-0.12.1 fake-useragent-
1.5.1 feedparser-6.0.11 flatbuffers-24.3.25 fonttools-4.53.1 frozendict-2.4.4 gast-0.6.0 google-auth-2.33.0 google-auth-oauthlib-1.2.1 google-pasta-0.2.0 grpcio-
1.65.4 h5py-3.11.0 html5lib-1.1 idna-3.7 importlib_metadata-8.2.0 importlib_resources-6.4.2 joblib-1.4.2 keras-2.15.0 kiwisolver-1.4.5 libclang-18.1.1 lxml-5.3.0
 lxml_html_clean-0.2.0 markdown-it-py-3.0.0 matplotlib-3.9.2 mdurl-0.1.2 ml-dtypes-0.2.0 multitasking-0.0.11 namex-0.0.8 numpy-1.26.4 oauthlib-3.2.2 opt-einsum-3
.3.0 optree-0.12.1 packaging-24.1 pandas-2.2.2 pandas-datareader-0.10.0 parse-1.20.2 peewee-3.17.6 pillow-10.4.0 platformdirs-4.2.2 protobuf-4.25.4 pyasn1-0.6.0
```

Make sure the TensorFlow version is == 2.1.5 and lower, otherwise the code will not work.

```
(.venv) PS C:\Users\malit\PycharmProjects\pythonProject3> pip show tensorflow
Name: tensorflow
Version: 2.15.0
```

Once everything is setup, the program is ready to be run. Click the ‘run’ button on your respective IDE.

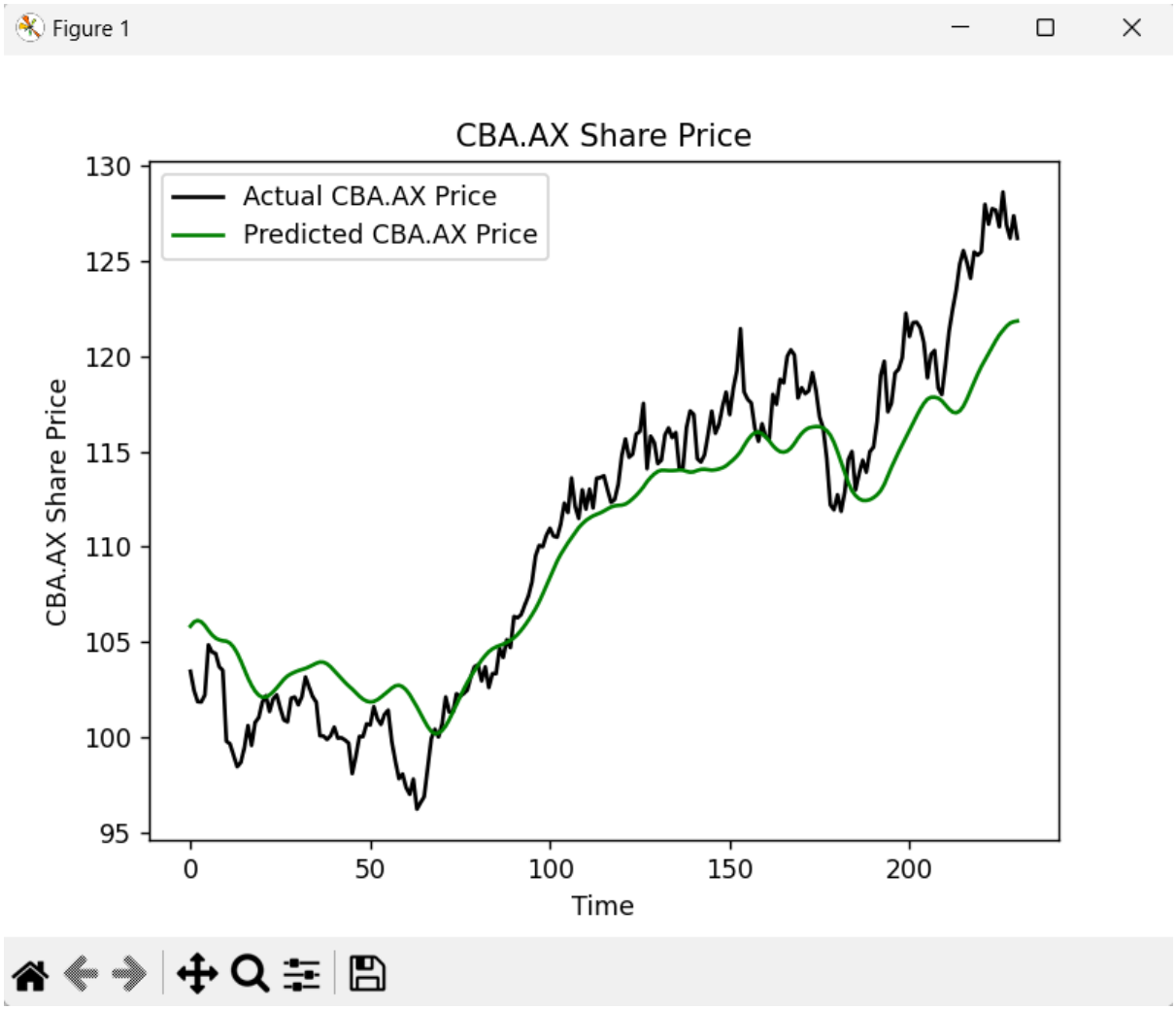


The stock prediction program will then work and go through the specified epochs to execute and provide the stock predictions.

```
Epoch 1/25
WARNING:tensorflow:From C:\Users\malit\PycharmProjects\pythonProject3\.ve

27/27 [==============================] - 4s 28ms/step - loss: 0.1023
Epoch 2/25
27/27 [==============================] - 1s 28ms/step - loss: 0.0146
Epoch 3/25
27/27 [==============================] - 1s 28ms/step - loss: 0.0106
Epoch 4/25
27/27 [==============================] - 1s 28ms/step - loss: 0.0084
Epoch 5/25
27/27 [==============================] - 1s 27ms/step - loss: 0.0086
Epoch 6/25
27/27 [==============================] - 1s 27ms/step - loss: 0.0082
Epoch 7/25
27/27 [==============================] - 1s 28ms/step - loss: 0.0085
Epoch 8/25
27/27 [==============================] - 1s 28ms/step - loss: 0.0072
Epoch 9/25
13/27 [=============>................] - ETA: 0s - loss: 0.0071
```

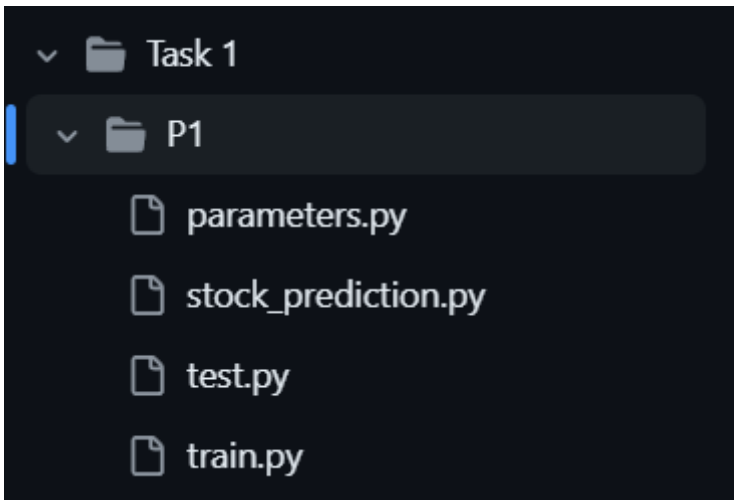
Once the program is finished, it will produce a graphic displaying the predicted value of the specified stock and its actual value using the specified data.



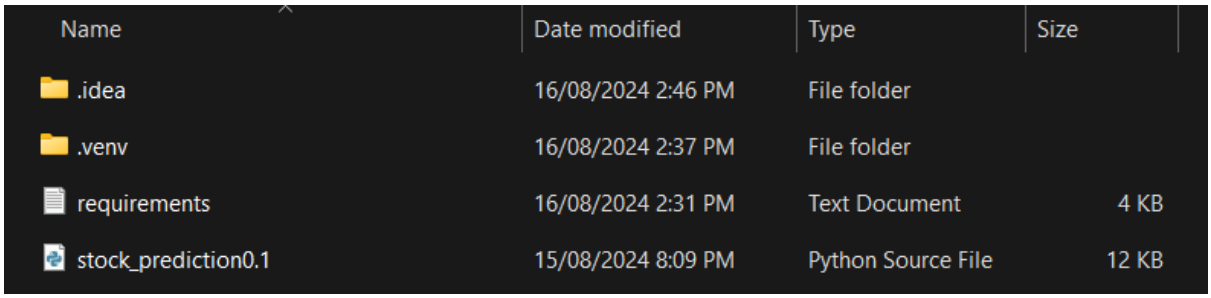
Executing the P1 code base –

Navigate to the task 1 folder in the repository and go to the **‘P1’** folder.

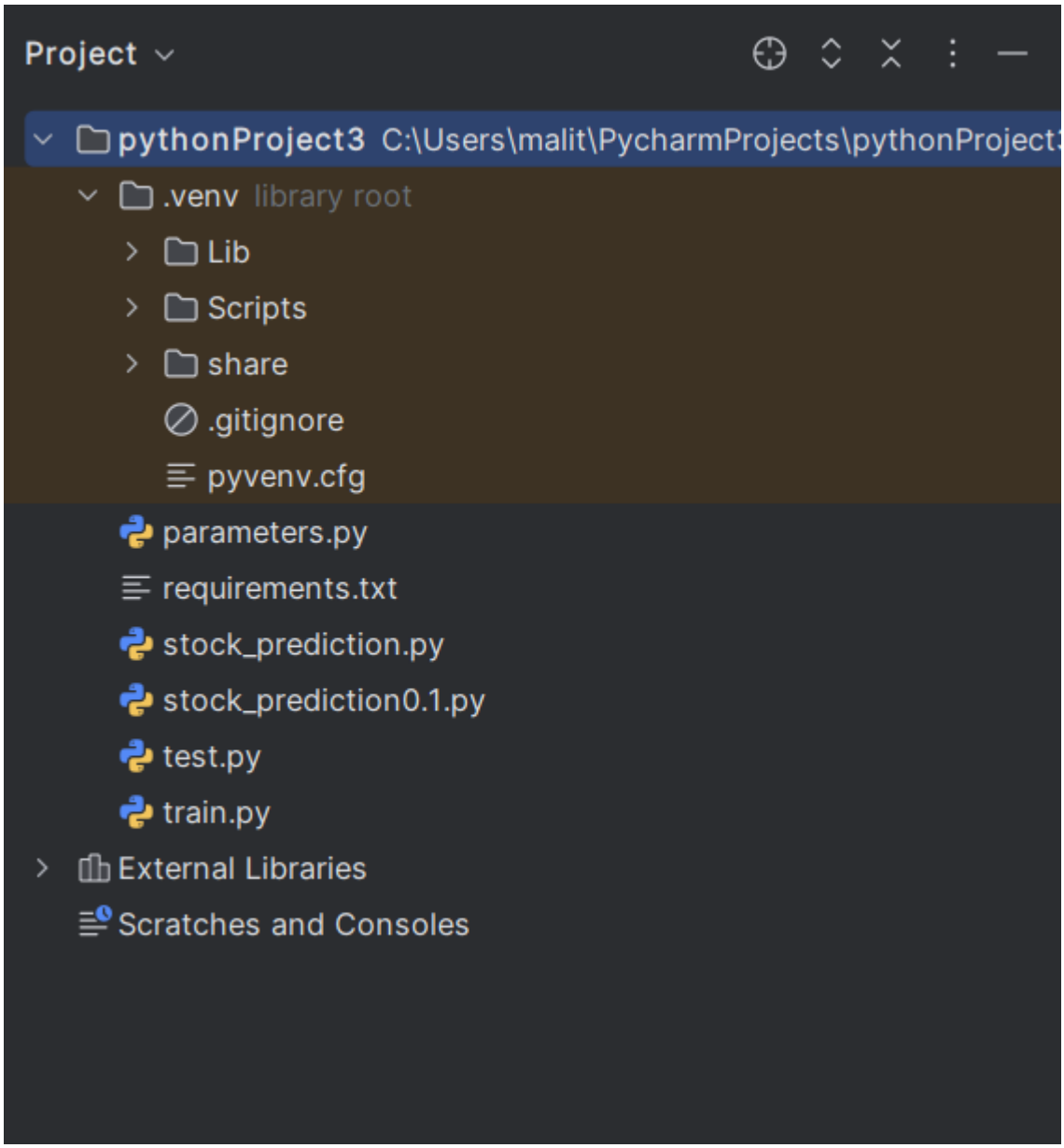
Download the provided files in the folder.



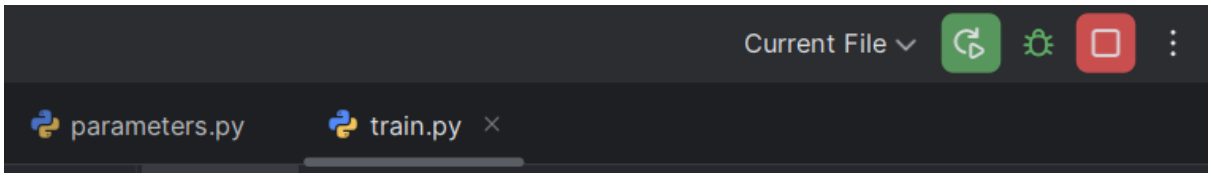
After downloading the provided files, navigate to the previously created project's directory:



Drag and drop the downloaded P1 files inside the project directory.



After the files are in the project directory, go ahead and run the train.py file to train your model.



Let the program run for the specified number of Epoch's, for our project we use 500 but we can train our model using more if needed.

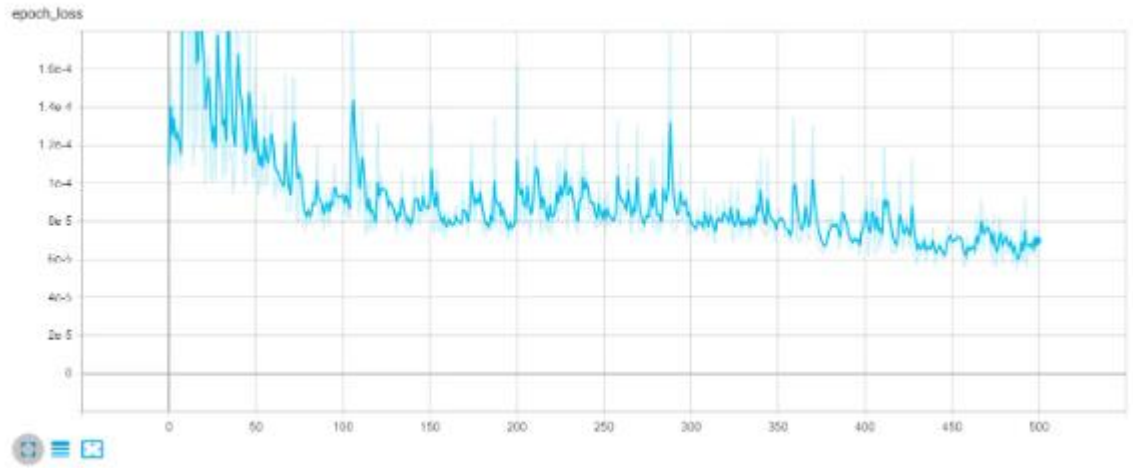
```
Epoch 1/500
```

```
85/85 [==============================] - ETA: 0s - loss: 0.0024 - mean_absolute_error: 0.0298
```

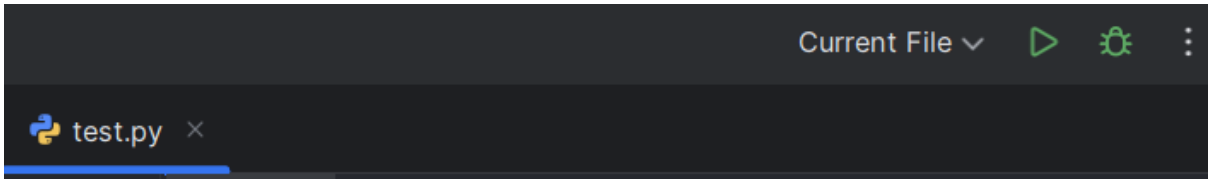
Once the model finishes training, we can inspect the training results which displays the Huber loss and validation loss of our dataset. Increasing the number of Epoch to improve results. Run the following command:

```
tensorboard --logdir="logs"
```

This command will start a local HTTP server at localhost:6006, after going to the browser, you will the following graph:



After training the model, we can evaluate it and test it on the testing dataset. Go ahead and run the test.py file in the directory:



Once the model is done evaluating the test set it provides the following statistics and figures for the specified dataset.

Mean absolute error*:* We get about 20 as error, which means, on average, the model predictions are far by over 20$ to the true prices; this will vary from ticker to another, as prices get larger, the error will increase as well. As a result, you should only compare your models using this metric when the ticker is stable (e.g., AMZN).

Buy/Sell profit: This is the profit we get if we opened trades on all the testing samples, we calculated these on get_final_df() function.

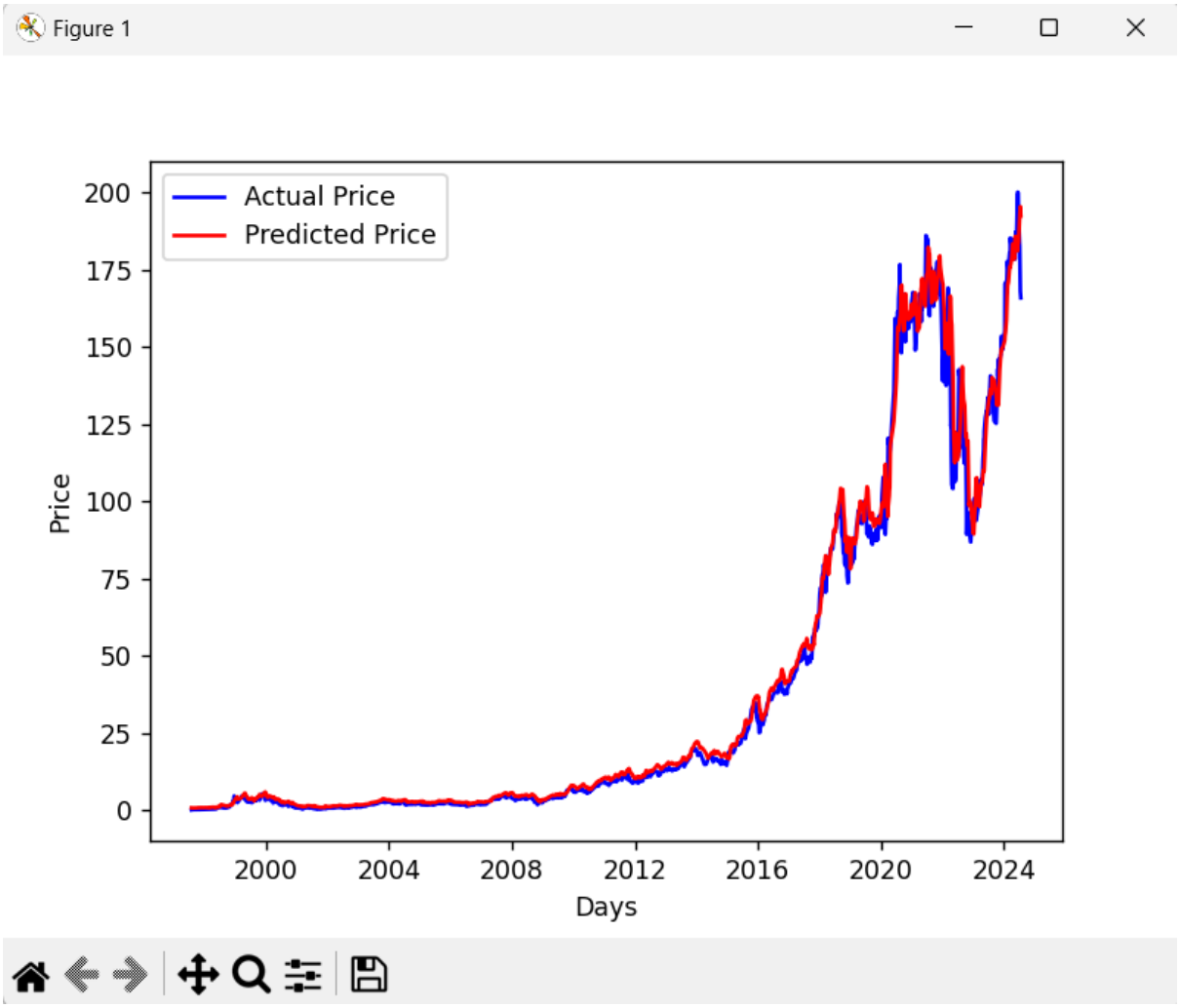
Total profit: This is simply the sum of buy and sell profits.

Profits Per Trade: The total profit divided by the total number of testing samples.

Accuracy score: This is the score of how accurate our predictions are. This calculation is based on the positive profits from all the trades from the testing samples.

```
43/43 [==============================] - 2s 26ms/step
1/1 [==============================] - 0s 22ms/step
Future price after 15 days is 169.63$
huber_loss loss: 0.0004604605201166123
Mean Absolute Error: 3.10997726240597
Accuracy score: 0.5879323031640913
Total buy profit: 606.5050662755967
Total sell profit: 55.759042739868164
Total profit: 662.2641090154649
Profit per trade: 0.487317225176942 54
```

The program will also output the graphic displaying the actual price of the stock using the data and the predicted price in the next 15 days:



Understanding of the current v0.1 code –

The program initially downloads the data for the specified stock using yahoo finance.

```
COMPANY = 'CBA.AX'

TRAIN_START = '2020-01-01'     # Start date to read
TRAIN_END = '2023-08-01'       # End date to read
```

```
import yfinance as yf

# Get the data for the stock AAPL
data = yf.download(COMPANY,TRAIN_START,TRAIN_END)
```

Next the program reshapes the x data from a 1d array to a 2d array so its usable with the

Sklearn.preprocessing.fit.transform() function.

```
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, newshape: (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
```

Next the program builds the model using a tensorflow and keras models, *Sequential* and *LSTM*, Sequential being a basic neural network, and LSTM being used to return the training results by utilising the use of an input layer with a three-dimensional sequence input. This process is referred to as a 'stacked' LSTM. The number of 'units' specifies the number of nodes in this layer, this is one of the main parameters that can affect price prediction quality.

```
model = tf.keras.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))
```

```
# 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.
```

The program utilises the 'Dropout' model to space unit inputs by setting them to '0' at an interval of '0.2' to prevent 'overfitting' which is a major problem in ML (machine learning).

```
model.add(Dropout(0.2))
```

Further utilisation of stacked LSTM.

```
model.add(LSTM(units=50, return_sequences=True))
```

Prediction of the next closing value of the stock price.

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

The program utilises another model to compile the model results by using the 'optimiser' and 'loss' parameters. There are multiple optimiser/loss options available that can **affect** prediction, optimiser= 'rmsprop' / 'sgd' / 'adadelta' /, loss = 'mean_absolute_error' / 'huber_loss'/ 'cosine_similarilty' /.

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

Now the program begins the model training using our prepared data, x_train & y_train, as well as using two other parameters to control the training process. Epochs are used to specify the iterations but must be carefully considered to avoid 'overfitting', usually more are better. Batches are used to prevent overfitting but can result in a aggregated losses/errors.

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

The program proceeds to test the model accuracy on existing data following a similar procedure to the previous training process but with

some changes and problems (for future learning).

```
actual_prices = test_data[PRICE_VALUE].values

total_dataset = pd.concat( objs: (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 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)
```

```
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,  newshape: (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)
```

This part of the program plots the data.

```
plt.plot( *args: actual_prices, color="black", label=f"Actual {COMPANY} Price")
plt.plot( *args: 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()
```

And the following provides us a prediction of the next day:

```
real_data = [model_inputs[len(model_inputs) - PREDICTION_DAYS:, 0]]
real_data = np.array(real_data)
real_data = np.reshape(real_data,  newshape: (real_data.shape[0], real_data.shape[1], 1))

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

Problems/Recommendations –

- Research stacked LSTM.
- Experiment with model parameters to improve predictions (epochs, batch_sizes, compiler parameters).
- Implement reusable data.