



Intelligent System

Assignment 1 - Option B

Week 1 Report

Xuan Tuan Minh Nguyen - 103819212

<i>INTRODUCTION</i>	<i>3</i>
<i>ENVIRONMENT SETUP</i>	<i>3</i>
<i>SETTING UP AND INSTALL REQUIRED DEPENDENCIES</i>	<i>3</i>
1. <i>CREATE A NEW ENVIRONMENT USING CONDA CLI</i>	<i>3</i>
2. <i>INSTALLING REQUIRED DEPENDENCIES</i>	<i>4</i>
<i>DEPLOYING AND TESTING THE CODEBASE</i>	<i>6</i>
1. <i>Deploying and testing the v0.1 codebase</i>	<i>6</i>
<i>Setup and initialisation processes</i>	<i>6</i>
<i>Executing the codebase</i>	<i>6</i>
<i>Challenges and Solutions</i>	<i>6</i>
<i>Results</i>	<i>7</i>
2. <i>Deploying and testing the P1 codebase</i>	<i>7</i>
<i>Setup and initialisation processes</i>	<i>7</i>
<i>Executing the codebase</i>	<i>7</i>
<i>Challenges and Solutions</i>	<i>8</i>
<i>Results</i>	<i>8</i>
<i>UNDERSTANDING THE v0.1 CODEBASE</i>	<i>9</i>
1. <i>Inporting Essential Libraries</i>	<i>9</i>
2. <i>Data Loading and Preperation</i>	<i>10</i>
3. <i>Model building and Training</i>	<i>11</i>
4. <i>Testing and Prediction</i>	<i>11</i>
5. <i>Visualisation and Evaluation</i>	<i>12</i>

Introduction

This report is written in order to demonstrate Task B.1, which is aimed at setting up a development environment for the stock prediction project to securely deploy and test the given initial code base (v0.1) and an external project (P1) adopted from Github. This report illustrates the process of **setting up** a virtual environment using **Anaconda**, which is used for **managing dependencies and libraries**, **downloading and testing** the codebase from **Github**, and documenting those processes. By **diving into and understanding** these processes, readers could gain a **closer perspective** on the setting up and testing processes. In addition, there are still rooms to **identify potential improvements** and **prepare for the next steps** on the model. This report entails the procedures taken to set up the environment, execute and test the results generated from the code, and understand the given code base.

Environment Setup

Setting up and install required dependencies

In order to create a **virtual environment** for Python, which is a workspace that is **well-organised** with **necessary libraries installed** and **adaptable** for the current task, **Anaconda** is chosen as the main virtual environment distributor. **Anaconda**, along with **Pycharm** and **venv**, are the three **most recognised distributors** when it comes to creating a safe and clean environment for deploying Python (and R) code. **Anaconda** offers an **isolated, customisable workspace** with **pre-installed packages** for data scientists. It also supports **cross-platform**, seamless **package management** via **Conda**, integration with Jupyter and Intellisense, and is also a tool for creating **scalability, security, and reproducibility** code.

1. Create a new environment using Conda CLI

There are several different procedures to create an environment, but the given procedure below will **encapsulate** all of the **required steps** to create a **safe and clean environment** using **Conda**.

- Navigate to the [Github repository](#) that contains the source code for both P1 and V0.1.
- Once navigated, download the source code by clicking on **Code** → **Download ZIP** or use the following command in the CLI (Terminal):
git clone <https://github.com/cobeo2004/cos30018.git>
- Once the source code is successfully cloned (downloaded), navigate to the **Week 1/v0.1** (or **Week 1/P1**) folder and execute the file **config.sh** using the following command:

bash config.sh

- The given file **config.sh** will execute the following procedure:

- Generate an environment with a pre-defined name (you can change the name if you want to) in **Python 3.11** by using the command:
`conda create -n cos30018-env-w1-v0.1 python=3.11`
 - Activate the created environment using: `conda activate cos30018-env-w1-v0.1`.
 - Install the required packages that are defined in **requirements.txt**.
- Once the **config.sh** is executed, we can validate if the virtual environment is correctly configured by using `conda info` for the current **Conda environment** and `python --version` for checking if the **Python version** is matched with the version we defined in **config.sh**.

```

> conda info && python --version

active environment : base
active env location : /Users/cobee/miniconda3
shell level : 3
user config file : /Users/cobee/.condarc
populated config files : /Users/cobee/.condarc
conda version : 22.9.0
conda-build version : not installed
python version : 3.10.6.final.0
virtual packages :
  __unix=0
  __archspec=1=arm64

base environment : /Users/cobee/miniconda3 (writable)
conda av data dir : /Users/cobee/miniconda3/etc/conda
conda av metadata url : None
channel URLs : https://conda.anaconda.org/conda-forge/osx-arm64
               https://conda.anaconda.org/conda-forge/noarch
               https://repo.anaconda.com/pkgs/main/osx-arm64
               https://repo.anaconda.com/pkgs/main/noarch
               https://repo.anaconda.com/pkgs/r/osx-arm64
               https://repo.anaconda.com/pkgs/r/noarch
package cache : /Users/cobee/miniconda3/pkgs
                 /Users/cobee/.conda/pkgs
envs directories : /Users/cobee/miniconda3/envs
                  /Users/cobee/.conda/envs
platform : osx-arm64
user-agent : conda/22.9.0 requests/2.28.1 CPython/3.10.6 Darwin/23.2.0 OSX/14.2.1
UID:GID : 501:20
netrc file : /Users/cobee/.netrc
offline mode : False

Python 3.10.6

```

Figure 1: Results when running both *conda info* and *python --version*

2. Installing required dependencies

Once the **environment** is **successfully initialised**, we can start **installing the dependencies (libraries)** that are **required by the program**. There are multiple pathways to install dependencies in Python, but the **most popular steps** are:

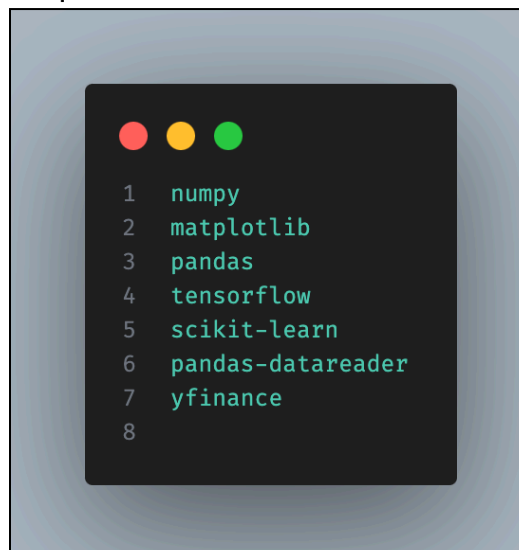
- Scan through the code to find out the required dependencies; for example, consider the file **stock_prediction.py**. We could see that there are quite a few required dependencies, such as: **numpy matplotlib pandas tensorflow scikit-learn pandas-datareader yfinance**.



```
1 # File: stock_prediction.py
2 # Authors: Bao Vo and Cheong Koo
3 # Date: 14/07/2021(v1); 19/07/2021 (v2); 02/07/2024 (v3)
4
5 # Code modified from:
6 # Title: Predicting Stock Prices with Python
7 # Youtube link: https://www.youtube.com/watch?v=PuzY9q-aKLw
8 # By: NeuralNine
9
10 # Need to install the following (best in a virtual env):
11 # pip install numpy
12 # pip install matplotlib
13 # pip install pandas
14 # pip install tensorflow
15 # pip install scikit-learn
16 # pip install pandas-datareader
17 # pip install yfinance
18
19 import numpy as np
20 import matplotlib.pyplot as plt
21 import pandas as pd
22 import pandas_datareader as web
23 import datetime as dt
24 import tensorflow as tf
25
26 from sklearn.preprocessing import MinMaxScaler
27 from tensorflow.keras.models import Sequential
28 from tensorflow.keras.layers import Dense, Dropout, LSTM, InputLayer
```

Figure 2: Required Dependencies in *stock_prediction.py*

- Once dependencies are scanned, use the following command to install the dependencies: `pip install numpy matplotlib pandas tensorflow scikit-learn pandas-datareader yfinance`.
- Another step is to list all required libraries into a *requirements.txt* file, and using the following command to install the required dependencies: `pip install -U -r requirements.txt`.



```
1 numpy
2 matplotlib
3 pandas
4 tensorflow
5 scikit-learn
6 pandas-datareader
7 yfinance
8
```

Figure 3: Example of *requirements.txt*

However, since I have configured all of the settings needed in the **config.sh** file, including the procedure to install the required dependencies, it is not essential to implement this step on your own.

Deploying and testing the codebase

The **deploying and testing process** of the given code bases (v0.1 and P1) is important to **analyse** and ensure that the provided models are **successfully set up** and could **seamlessly deploy** within the **configured environment**. This section will **outline** the **processes and results** of **deploying and testing** these code bases.

1. Deploying and testing the v0.1 codebase

Setup and initialisation processes

- In order to execute the project, every step from the *Environment Setup* step must be completed; if you have not, please refer back to the *Environment Setup* step.
- Once the environment is activated and all dependencies are installed, we can move on to the *Executing the codebase* step.

Executing the codebase

- There are two ways to execute this project: one via **terminal** and one via **Jupyter Notebook**.
- If **terminal** is preferred, simply navigate to the **v0.1** folder and execute the following command to execute the code: `python stock_prediction.py`.
- If **Jupyter Notebook** is preferred, refer to [Installing Jupyter Notebook: A Step-by-Step Guide by Kumari Prerna](#) to learn how to setup and execute a **Jupyter Notebook**.

Challenges and Solutions

- During the initial execution, there are several issues that are related to missing dependencies and unsupported functions. However, this could be easily resolved by installing the required packages.

Results



Figure 4: V0.1 model training with 25 epochs

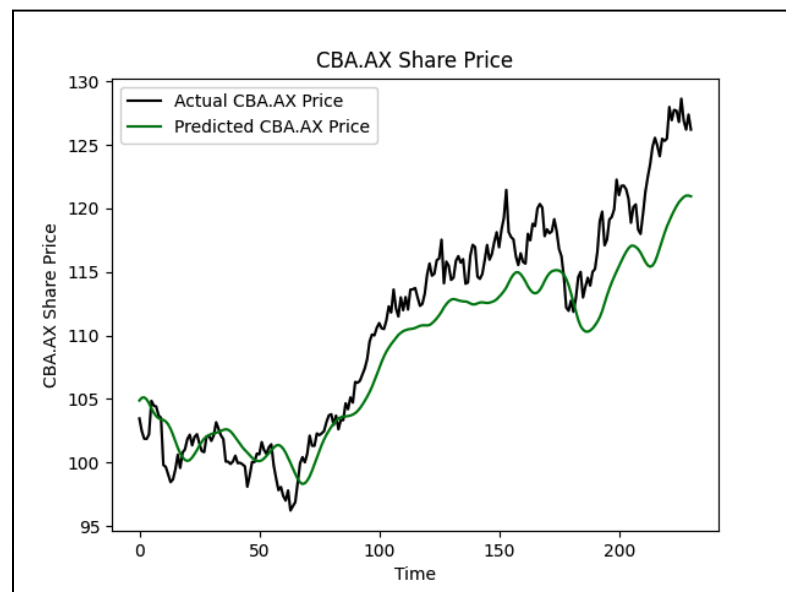


Figure 5: V0.1 stock prediction results

2. Deploying and testing the P1 codebase

Setup and initialisation processes

- In order to execute the project, every step from the *Environment Setup* step must be completed; if you have not, please refer back to the *Environment Setup* step.
- Once the environment is activated and all dependencies are installed, we can move on to *Executing the codebase*.

Executing the codebase

- There are two ways to execute this project, one via **terminal** and one via **Jupyter Notebook**.

- If **terminal** is preferred, simply navigate to the **v0.1** folder and execute the following command to execute the code: `python stock_prediction_p1.py`
- If **Jupyter Notebook** is preferred, refer to [Installing Jupyter Notebook: A Step-by-Step Guide by Kumari Prerna](#) to learn how to setup and execute a **Jupyter Notebook**.

Challenges and Solutions

- Since the given codebase was last updated a few years ago, several dependencies are deprecated and incompatible with the latest version of Python. To overcome this problem, I have made several changes to the `requirements.txt` file, including changing the library version and adding any alternative libraries that support the same functionalities.

Results

```
Epoch 1/100
/Users/cobee/miniconda3/envs/cos38819-env-wl-nl/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an 'input_shape'/'input_dim' argument to a
super().__init__(**kwargs)
85/85 0s 143ms/step - loss: 0.0066 - mean_absolute_error: 0.0502
Epoch 1: val_loss improved from inf to 0.00050, saving model to results/2024-08-16_AMZN-sh-1-sc-1-sbd-0-huber-adam-LSTM-seq-50-step-15-layers-2-units-256.weights.h5
85/85 15s 162ms/step - loss: 0.0066 - mean_absolute_error: 0.0500 - val_loss: 4.9782e-04 - val_mean_absolute_error: 0.0143
Epoch 2/100
85/85 0s 166ms/step - loss: 7.8091e-04 - mean_absolute_error: 0.0196
Epoch 2: val_loss improved from 0.00050 to 0.00047, saving model to results/2024-08-16_AMZN-sh-1-sc-1-sbd-0-huber-adam-LSTM-seq-50-step-15-layers-2-units-256.weights.h5
85/85 16s 187ms/step - loss: 7.8075e-04 - mean_absolute_error: 0.0196 - val_loss: 4.6796e-04 - val_mean_absolute_error: 0.0141
Epoch 3/100
85/85 0s 183ms/step - loss: 6.9940e-04 - mean_absolute_error: 0.0183
Epoch 3: val_loss improved from 0.00047 to 0.00043, saving model to results/2024-08-16_AMZN-sh-1-sc-1-sbd-0-huber-adam-LSTM-seq-50-step-15-layers-2-units-256.weights.h5
85/85 17s 204ms/step - loss: 6.9916e-04 - mean_absolute_error: 0.0183 - val_loss: 4.3293e-04 - val_mean_absolute_error: 0.0135
Epoch 4/100
85/85 0s 164ms/step - loss: 7.4192e-04 - mean_absolute_error: 0.0195
Epoch 4: val_loss did not improve from 0.00043
85/85 16s 184ms/step - loss: 7.4222e-04 - mean_absolute_error: 0.0195 - val_loss: 5.9813e-04 - val_mean_absolute_error: 0.0188
Epoch 5/100
85/85 0s 171ms/step - loss: 6.8029e-04 - mean_absolute_error: 0.0192
Epoch 5: val_loss did not improve from 0.00043
85/85 16s 192ms/step - loss: 6.8058e-04 - mean_absolute_error: 0.0192 - val_loss: 9.6708e-04 - val_mean_absolute_error: 0.0233
Epoch 6/100
85/85 0s 177ms/step - loss: 8.4421e-04 - mean_absolute_error: 0.0214
Epoch 6: val_loss did not improve from 0.00043
85/85 17s 196ms/step - loss: 8.4276e-04 - mean_absolute_error: 0.0213 - val_loss: 6.8323e-04 - val_mean_absolute_error: 0.0192
Epoch 7/100
85/85 0s 162ms/step - loss: 7.0262e-04 - mean_absolute_error: 0.0189
...
Epoch 100/100
85/85 0s 172ms/step - loss: 4.8834e-04 - mean_absolute_error: 0.0180
Epoch 100: val_loss did not improve from 0.00039
85/85 16s 191ms/step - loss: 4.8810e-04 - mean_absolute_error: 0.0180 - val_loss: 4.0817e-04 - val_mean_absolute_error: 0.0132
```

Figure 6: P1 Model Training with 100 epochs

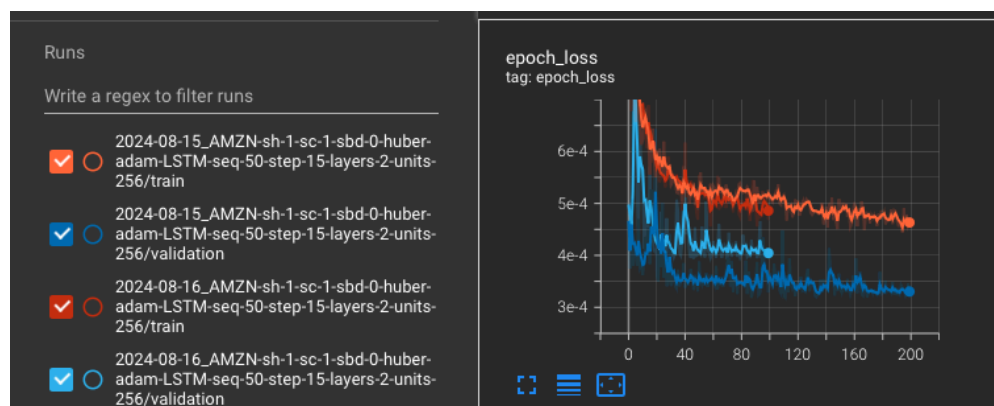


Figure 7: Epoch loss after 100 epochs


```
Future price after 15 days is 169.81$
huber loss: 0.00039213348645716906
Mean Absolute Error: 2.584657595144968
Accuracy score: 0.5459896983075792
Total buy profit: 650.449882209301
Total sell profit: 99.70385867357251
Total profit: 750.1537408828735
Profit per trade: 0.5519895076400836
```

Figure 8: Prediction result for the next 15 days

Understanding the v0.1 codebase

1. Importing Essential Libraries

In this procedure, essential libraries are imported into Python code with different functionalities in order to serve different purposes of the program, which could be listed as:

- Numerical Computing: [Numpy](#).
- Data manipulation and analysis: [Pandas](#).
- Date and time operations: [Datetime](#).
- Create an LSTM network and training process: [Tensorflow](#), [Sklearn](#), [YFinance](#).

```
# Numerical computing
import numpy as np

# Data manipulation and analysis
import pandas as pd

# Data visualization
import matplotlib.pyplot as plt

# Date and time operations
import datetime as dt

# Machine learning and deep learning
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM,
InputLayer

# Stock data retrieval
import yfinance as yf
```

Figure 9: Code section for Importing Essential Libraries

2. Data Loading and Preperation

During this process, we will load the historical data from the **Commonwealth Bank Stock (CBA.AX)** from the day range of **01-01-2020** to **01-08-2023** and prepare the loaded data for the training process:

- Load data from **Commonwealth Bank Stock (CBA.AX)** from **01-01-2020** to **01-08-2023** using **YFinance**.
- We will focus on the **closing price (Close column)** by setting up the **PRICE_VALUE** variable to **Close**.
- Using **MinMaxScaler()** algorithm provided by **Sklearn**, we will scale the data to the range of **(0, 1)**.
- Create a 60-day sequence as the input for our model (**x_train**) by setting **PREDICTION_DAYS** to 60.
- Take the 61st day's price as the target output for our model (**y_train**).
- Using **np.reshape()** function from **Numpy** to fit the input data with the LSTM model input.

```
COMPANY = 'CBA.AX'
#Train start date
TRAIN_START = '2020-01-01'
#Train end date
TRAIN_END = '2023-08-01'
#Price value to predict
PRICE_VALUE = "Close"
#Number of days to predict
PREDICTION_DAYS = 60

# Download stock data
data = yf.download(COMPANY, TRAIN_START, TRAIN_END)

# Prepare data for scaling using 'MinMaxScaler()'
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data[PRICE_VALUE].values.reshape(-1, 1))

# Prepare training data
x_train = []
y_train = []

scaled_data = scaled_data[:,0]
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 to numpy arrays
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 10: Code Section for Loading and Preparing Datasets

3. Model building and Training

Throughout this step, we will create a **sequential model** using **LSTM** as our main layer, **Dropout** layer between each **LSTM** layer to reduce our **dropout rate**. In addition, we will have a closer look at the training process of the created model with **Adam** as the optimizer and **Dense** layer as our output layer.

- Using **LSTM()** layer provided by **Tensorflow**, we create three layers (each with 50 units).
- Between each **LSTM()** layer, there is a **Dropout()** layer (provided by **Tensorflow**) with a 20% dropout rate.
- To **compile the model**, **Adam** is chosen to be the **optimizer**, and **Mean Squared Error** as the main **loss function**.
- To **train the model**, we used the **fit()** function from the **Sequential()** class, with 25 epochs and a batch size of 32.

```
# Using Sequential model
model = Sequential()

# Adding LSTM layers
model.add(LSTM(units=50, return_sequences=True,
input_shape=(x_train.shape[1], 1)))
# For each LSTM layer, we add a dropout layer
model.add(Dropout(0.2))
# Adding LSTM layers
model.add(LSTM(units=50, return_sequences=True))
# For each LSTM layer, we add a dropout layer
model.add(Dropout(0.2))
# Adding LSTM layers
model.add(LSTM(units=50))
# For each LSTM layer, we add a dropout layer
model.add(Dropout(0.2))
# Adding Dense layer
model.add(Dense(units=1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

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

Figure 11: Code Section for Building and Training Model

4. Testing and Prediction

In this section, we will load the test data from 02-08-2024 to 02-07-2024 to validate our model.

- We downloaded the test data from 02-07-2024 and 02-08-2024 using **YFinance**.
- We combined the test data with the training data to create a **Sequence** and scale using the same scaler (**MinMaxScaler()**) for training data.

- Finally, a prediction is made on the test data to get the actual price values by transforming the prediction inversively.

```
# Test start date
TEST_START = '2023-08-02'

# Test end date
TEST_END = '2024-07-02'

# Download test data
test_data = yf.download(COMPANY, TEST_START,
TEST_END)

# Actual prices
actual_prices = test_data[PRICE_VALUE].values

# Concatenate training and test data
total_dataset = pd.concat((data[PRICE_VALUE],
test_data[PRICE_VALUE]), axis=0)

# Prepare test data
model_inputs = total_dataset[len(total_dataset) -
len(test_data) - PREDICTION_DAYS:].values
# Reshape the data
model_inputs = model_inputs.reshape(-1, 1)
# Scale the data
model_inputs = scaler.transform(model_inputs)

# Make predictions
x_test = []
for x in range(PREDICTION_DAYS, len(model_inputs)):
    # Append the last PREDICTION_DAYS days to x_test
    x_test.append(model_inputs[x - PREDICTION_DAYS:x,
0])

# Convert to numpy arrays
x_test = np.array(x_test)
# Reshape the data
```

Figure 12: Code Section for Validating and Testing

5. Visualisation and Evaluation

Finally, we will draw the plot to visualize the **actual and the predicted prices** using **matplotlib** and predict the **next day price**.

```
# Plot the actual prices
plt.plot(actual_prices, color="black", label=f"Actual
{COMPANY} Price")
# Plot the predicted prices
plt.plot(predicted_prices, color="green", label=f"Predicted
{COMPANY} Price")
# Set the title
plt.title(f"{COMPANY} Share Price")
# Set the x-axis label
plt.xlabel("Time")
# Set the y-axis label
plt.ylabel(f"{COMPANY} Share Price")
# Add a legend
plt.legend()
# Show the plot
plt.show()
### Predicting the Next Day's Price
plt.show()
```

Figure 13: Code Section for Visualizing the Results using *Matplotlib*

```
# Get the last PREDICTION_DAYS days
real_data = [model_inputs[len(model_inputs) - PREDICTION_DAYS:, 0]]
# Convert to numpy array
real_data = np.array(real_data)
# Reshape the data
real_data = np.reshape(real_data, (real_data.shape[0],
real_data.shape[1], 1))

# Predict the next day's price
prediction = model.predict(real_data)
# Inverse transform the data
prediction = scaler.inverse_transform(prediction)
# Print the prediction
print(f"Prediction: {prediction}")
```

Figure 14: Code Section for Predicting Next Day's Price