



---

## Task B1: Set Up

---

Conducted by:

### Group 1

*Team Members*

*Email*

Thanh Tam Vo

103487596@student.swin.edu.au

February 20, 2024

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Before You Can Run</b>	<b>3</b>
2.1	Hardware Requirements . . . . .	3
2.2	Python Interpreter Requirements . . . . .	3
2.3	Installation . . . . .	3
<b>3</b>	<b>V0.1</b>	<b>4</b>
3.1	Jupyter Notebook Organization for V0.1 . . . . .	4
3.2	Code Base Explanation . . . . .	5
3.2.1	Import Dependencies . . . . .	5
3.2.2	Load Data . . . . .	5
3.2.3	Preparing Data . . . . .	6
3.2.4	Build the Model . . . . .	7
<b>4</b>	<b>P1</b>	<b>8</b>

# 1 Introduction

This report aims to demonstrate how to set up before running the code base of **V0.1** and **P1**. After demonstrating how to set up the environment for the entire project, this report will show the executed result of the tutorial from *Youtube* **V0.1** and the source code **P1** from a public *Github Repo*.

The Github Repository for the entire project *Stock Price Prediction* is also provided in this report.

## 2 Before You Can Run

**Note:** Due to the limitation of resources, the setting up process in this report is only demonstrated on Windows OS.

**Link For the GitHub Repository:** Please visit this **link** to download the entire project. At this moment, only my team members and lecturer can access the Repository. The Repository will be published publicly at the end of the semester (April 12, 2024).

### 2.1 Hardware Requirements

This project requires to use Tensorflow, which is an open-source for training machine learning model, so it is highly recommended to execute the program on the computer with **x86 Architecture** instead of **ARM Architecture**. Moreover, for users who experience in Apple Silicon, please use a virtual machine or any cloud platforms (*Google Colab, etc.*) to run **V0.1** and **P1**.

### 2.2 Python Interpreter Requirements

Make sure that Python is already installed on your machine. Moreover, your Python Version should be **3.9-3.11**.

If Python is not installed in the local machine, please visit this link in order to install on your device.

### 2.3 Installation

Below is the step-by-step for the installation process, it is highly recommended to use a virtual environment for this project, if you decided to install all dependencies on your machine, you can skip the Creating Virtual Environment Step:

1. Open **PowerShell** as Administrator Privilege
2. Execute this Command Line: **Set-ExecutionPolicy RemoteSigned**
3. Execute this Command Line: **python3 -m venv myenv** to create the virtual environment
4. Execute this Command Line: **myenv\Scripts\activate** to activate the virtual environment
5. Execute this Command Line: **.\requirements.bat** to install all dependencies

From now, you are able to run **V0.1** and **P1**.

### 3 V0.1

The tutorial from the Youtube uses the `.py` extension, but the source code will be modified: instead of using `.py` file, the source code for **V0.1** is written in `.ipynb`, which is an extension for Jupyter Notebook file.

#### 3.1 Jupyter Notebook Organization for V0.1

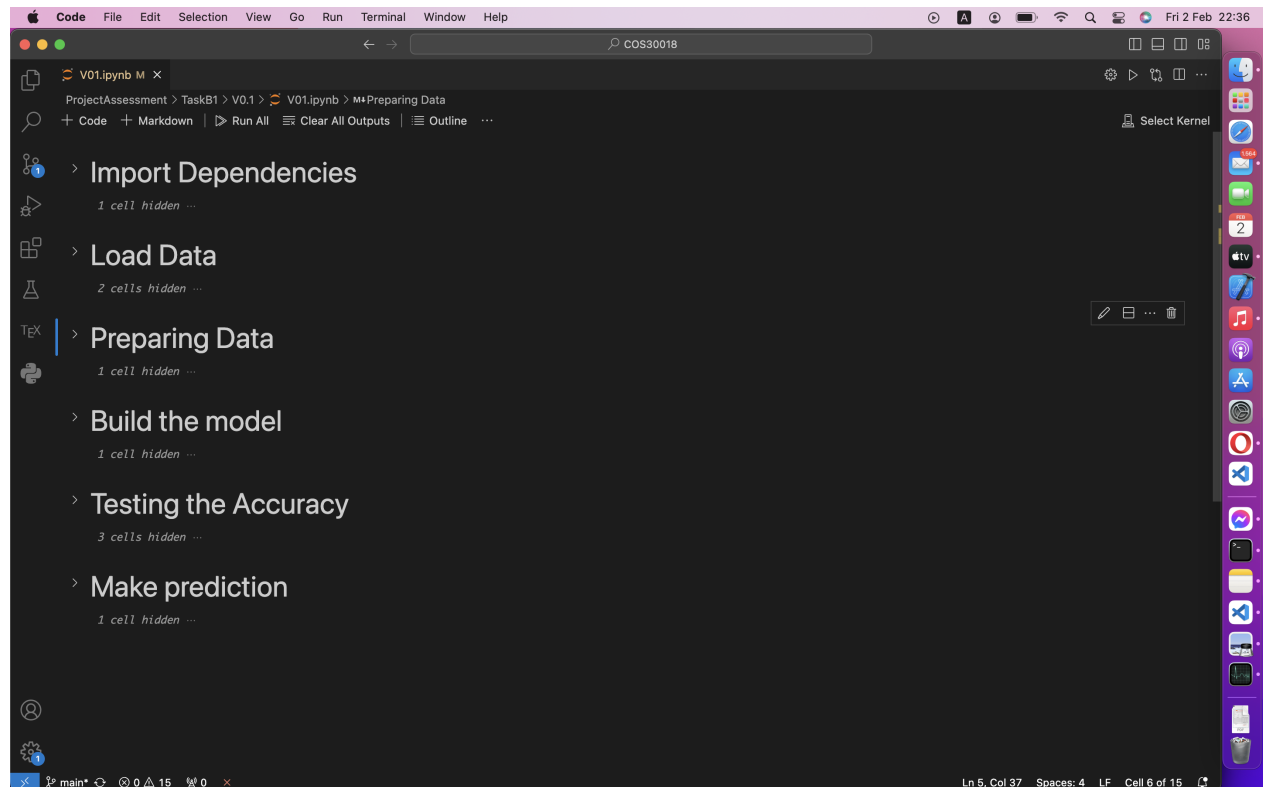


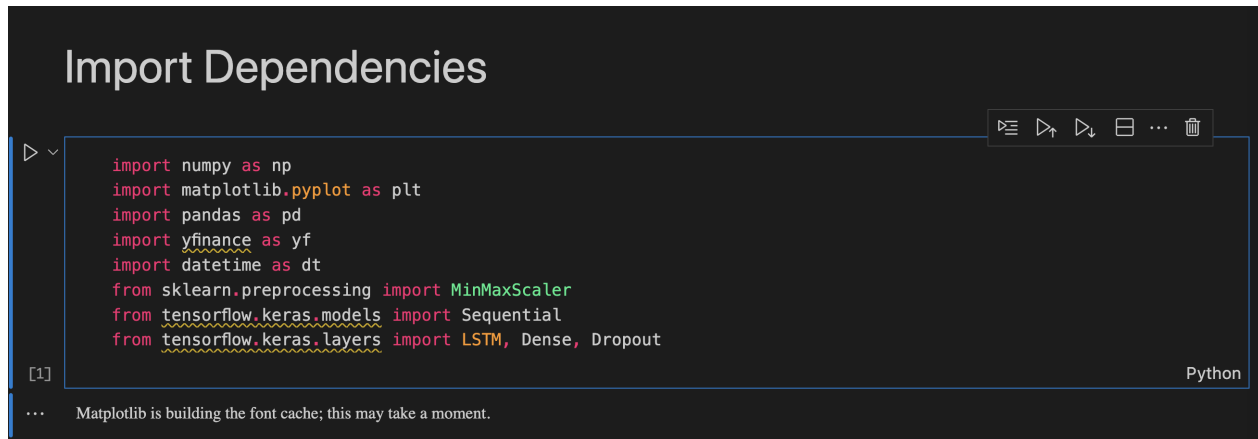
Figure 1: Structure Overview for **V01.ipynb**

The Figure 1 above is the overview for the **V01.ipynb** file. It is easy to observe that in each section, the cell will serve a specific responsibility:

- **Import Dependencies:** Importing all required packages to train the model
- **Load Data:** Downloading Stock Price dataset in order to train the model
- **Preparing Data:** Normalize the data, which is an important stage in training machine learning model
- **Build the model:** Configuring our model with specific hyper-parameter
- **Testing the Accuracy:** Testing the accuracy of the model based on the Test Dataset
- **Make prediction:** Making the prediction for the stock price based on the trained model

## 3.2 Code Base Explanation

### 3.2.1 Import Dependencies



```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import yfinance as yf
import datetime as dt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

```

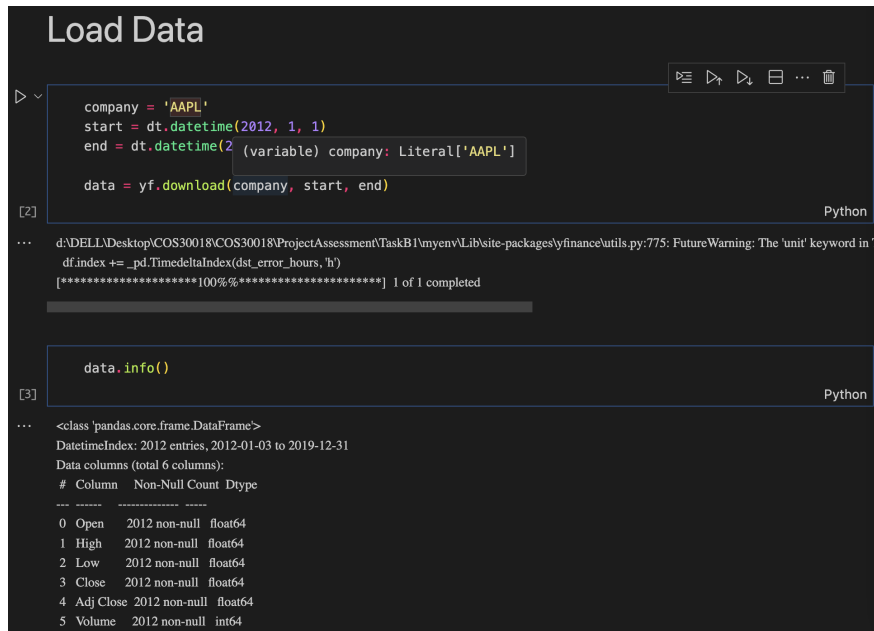
[1] Python

... Matplotlib is building the font cache; this may take a moment.

Figure 2: Importing required packages

The Figure 2 above shows the cell serving import packages responsibility. It is obvious so this part will not be discussed further.

### 3.2.2 Load Data



```

company = 'AAPL'
start = dt.datetime(2012, 1, 1)
end = dt.datetime(2020, 1, 1)

data = yf.download(company, start, end)

```

[2] Python

... d:\DELL\Desktop\COS30018\COS30018\ProjectAssessment\TaskB1\myenv\Lib\site-packages\yfinance\utils.py:775: FutureWarning: The 'unit' keyword in 'df.index += \_pd.TimedeltaIndex(dst\_error\_hours, 'h')' is deprecated and will be removed in a future version. Please use 'unit=' instead.

```

data.info()

```

[3] Python

```

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2012 entries, 2012-01-03 to 2019-12-31
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Open        2012 non-null    float64
1   High        2012 non-null    float64
2   Low         2012 non-null    float64
3   Close       2012 non-null    float64
4   Adj Close   2012 non-null    float64
5   Volume      2012 non-null    int64

```

Figure 3: Loading Data for Training

The Figure 3 above shows 2 hidden cells for the Data Loading Stage. Our dataset is a Pandas Dataframe for the Stock Price of *Apple Inc.* This dataframe contains 2012 records of Apple Inc. stock price from 2012 to 2020, in which the index for each record is a date.

### 3.2.3 Preparing Data

```

Preparing Data

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(data['Close'].values.reshape(-1, 1))

prediction_days = 60

x_train = []
y_train = []

for x in range(prediction_days, len(scaled_data)):
    x_train.append(scaled_data[x - prediction_days:x, 0])
    y_train.append(scaled_data[x, 0])

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))

```

[4]

Figure 4: Preparing Data for Training

The **MinMaxScaler** class was imported from the famous library **sklearn** to scale our data (*Close price*) such that all records are in the range  $[0, 1]$ .

The **prediction\_days** variable indicates the number of looking back days to perform the prediction. In Figure 4, we will observe the last 60 days to perform the *Close* price prediction for our Stock Ticket. Next, we prepare two arrays for training model: **x\_train** and **y\_train**. In **y\_train**, the first element is the *Close Price* (of AAPL) at the 60<sup>th</sup> date among 2012 records in the total dataset. On other hands, the first element in **x\_train** is an array of 60 *Close Price* records before the 60<sup>th</sup> date, more specifically, the first element is an array containing *Close Price* from the 1<sup>st</sup> date to the 59<sup>th</sup> date. You can look at Figure 5 for more details.

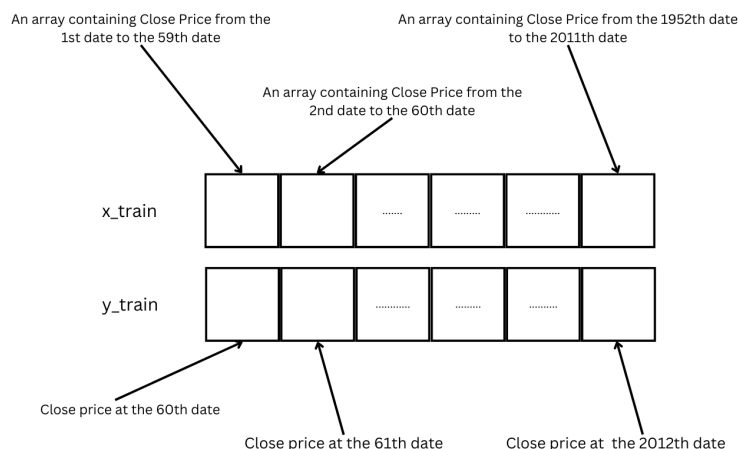
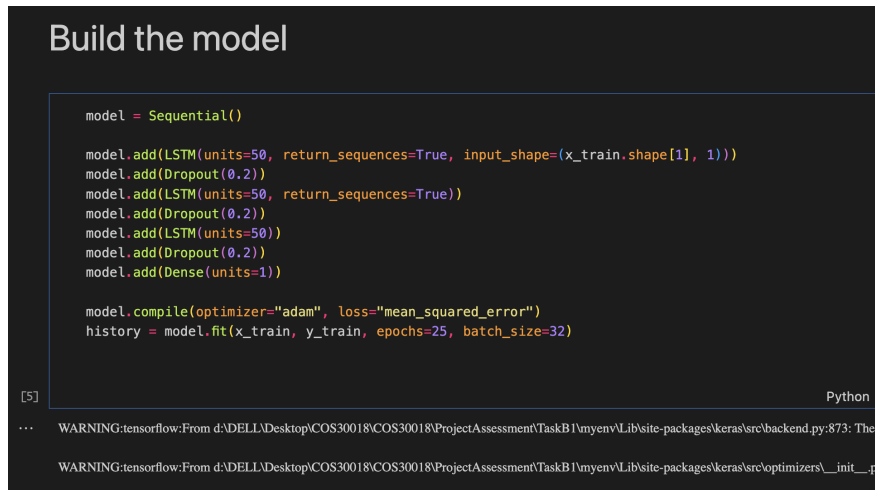


Figure 5: **x\_train** and **y\_train** arrays

Next, we will convert them into *np.array* for manipulation. Until now, **x\_train** is matrix with a shape of (1952,60) and **y\_train** is a vector with a shape of (1952,1). Then, we will reshape **x\_train** into a cube, so its final shape is (1952,60,1).

### 3.2.4 Build the Model



```

model = Sequential()

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

```

[5] Python

... WARNING:tensorflow:From d:\DELL\Desktop\COS30018\COS30018\ProjectAssessment\TaskB1\myenv\Lib\site-packages\keras\src\backend.py:873: The

WARNING:tensorflow:From d:\DELL\Desktop\COS30018\COS30018\ProjectAssessment\TaskB1\myenv\Lib\site-packages\keras\src\optimizers\\_init\_.py

Figure 6: Model Configuration

The Figure 6 above is the configuration of the Model, since we have to deal with Sequential Dataset, choosing the LSTM model and adding Dropout to reduce overfitting would be suitable. In order to find the perfect parameters efficiently, Mean Square Error is the metric for the loss function and Adam Optimizer Algorithm is applied. After 25 epochs of training (about 30 minutes), the Figure 7 below is our result:

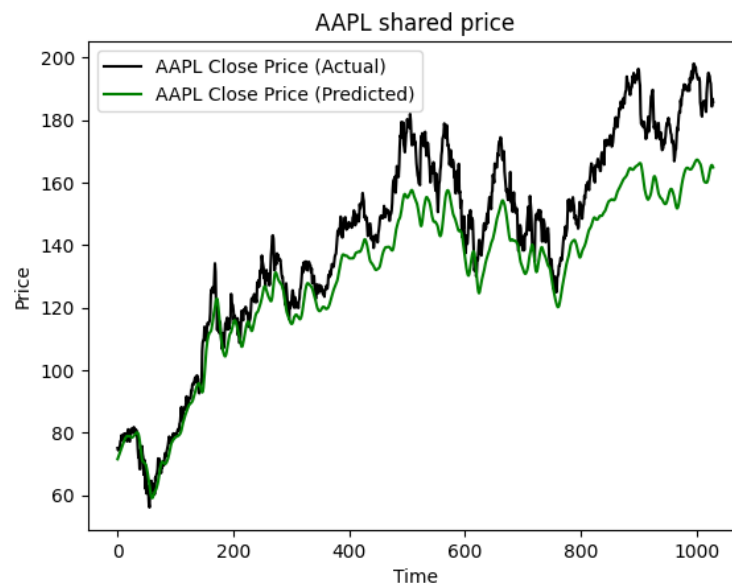


Figure 7: Result for the Training Process



## 4 P1

In **P1**, the model will predict the price of Amazon based on the last 15 days. Two Figures below are our attempt to test the code base.

```

42/42 [=====] - 3s 45ms/step
1/1 [=====] - 0s 40ms/step
Future price after 15 days is 164.91%
huber_loss loss: 0.800275625177891925
Mean Absolute Error: 2.1862286325089667
Accuracy score: 0.5657166416781865
Total buy profit: 676.3958988799334
Total sell profit: 181.12765620648864
Total profit: 777.522755086422
Profit per trade: 0.8128586409943193

  open      high      low  ... true_adjclose_15  buy_profit  sell_profit
2023-11-30  144.759995  146.929993  144.330002  ...      153.839996      0.0      -7.750000
2023-12-07  146.149994  147.919998  145.339996  ...      151.940002      0.0      -5.859998
2023-12-08  145.479996  147.839996  145.309994  ...      149.929993      0.0      -2.509995
2023-12-12  145.520004  147.500000  145.300003  ...      144.570007      0.0      2.909988
2023-12-15  148.380005  150.570007  147.880005  ...      151.369995      0.0      -1.399994
2024-01-08  146.740005  149.309994  146.149994  ...      159.000000      0.0      -9.209994
2024-01-09  148.330002  151.710007  148.210007  ...      155.199997      0.0      -3.830002
2024-01-10  152.059998  154.419998  151.880005  ...      159.279999      0.0      -5.550003
2024-01-16  153.939999  154.900005  152.349994  ...      169.149994      0.0     -15.589998
2024-01-26  158.419998  160.720001  157.910004  ...      169.389995      0.0     -10.389999

```

Figure 8: The Console Output when predicting the stock price

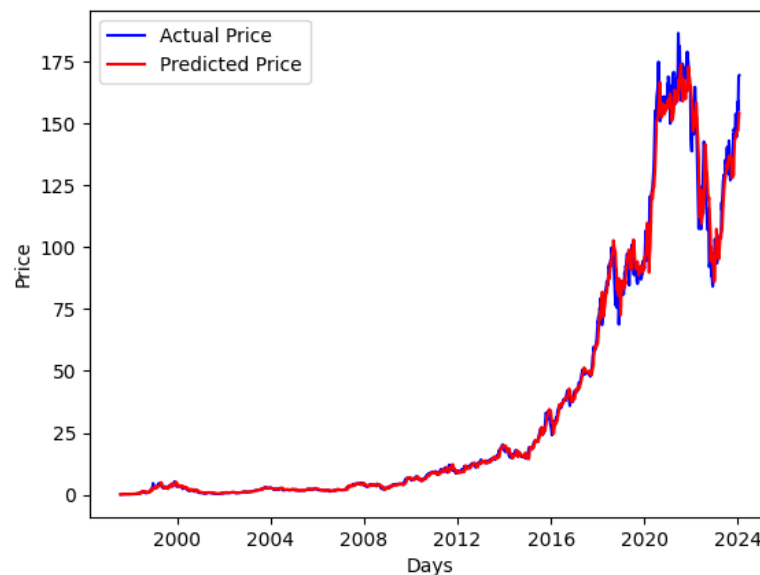


Figure 9: Stock Prediction of Amazon Ticker