

# Stock Price Prediction System

(Option 3)

## Weekly Report (Tasks C.1 - Setup)

COS30018 Intelligent Systems

Class: Thursday 2:30pm to 4:30pm (2 hours)

Name: Suyogya Raj Joshi

Student ID: 105098233

## Contents

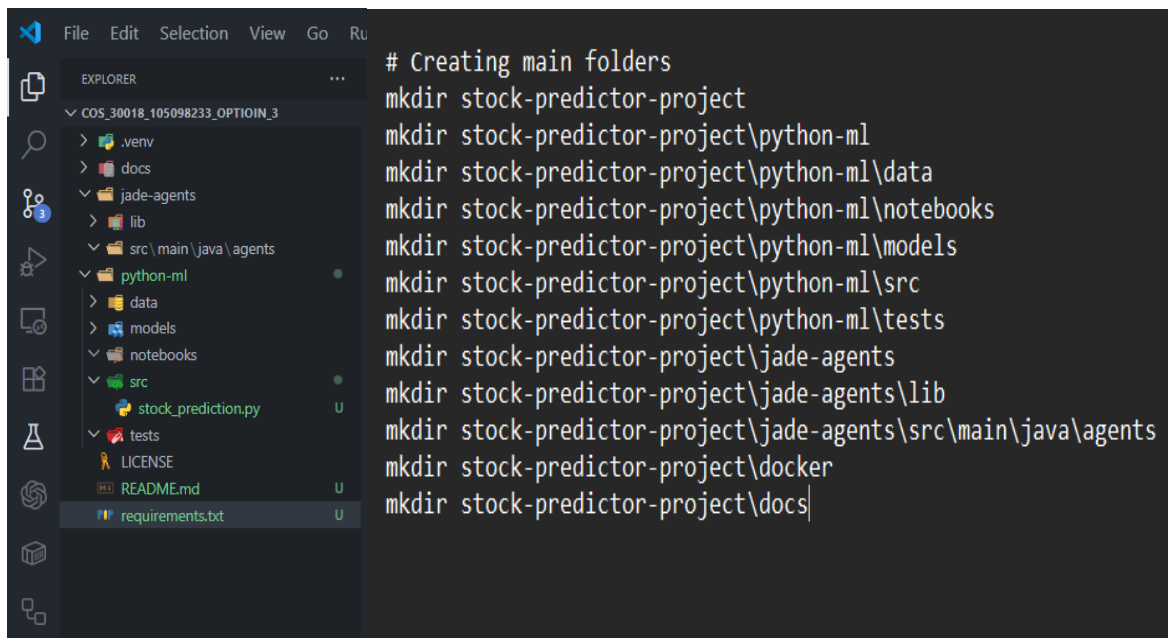
Weekly Report (Tasks C.1 - Setup).....	1
Check List of give weekly tasks: .....	3
Task 1: Setting up the environment along with the requirements file. ....	3
Task 2: Summaries of your attempts to test the provided code bases (v0.1 and P1) with screenshots. ....	4
Task 3: Summary of your understanding of the initial code base v0.1. ....	7

## Check List of given weekly tasks:

No.	Weekly Tasks	Progress
1.	Summaries of your attempt to set up your environment, including details of your requirements file	Completed
2.	Summaries of your attempts to test the provided code bases (v0.1 and P1) with screenshots.	Completed
3.	Summary of your understanding of the initial code base v0.1.	Completed

## Task 1: Setting up the environment along with the requirements file.

The first task of this week was to set up the **environment** for the project so the first thing I did was to set up the required file structure that allows me to properly manage my project. Using the given YouTube video, I created the required file structure which looks like the following:



The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar displays the project's file structure. The main editor area shows a terminal window with a list of commands used to create the directory structure.

```
# Creating main folders
mkdir stock-predictor-project
mkdir stock-predictor-project\python-ml
mkdir stock-predictor-project\python-ml\data
mkdir stock-predictor-project\python-ml\notebooks
mkdir stock-predictor-project\python-ml\models
mkdir stock-predictor-project\python-ml\src
mkdir stock-predictor-project\python-ml\tests
mkdir stock-predictor-project\jade-agents
mkdir stock-predictor-project\jade-agents\lib
mkdir stock-predictor-project\jade-agents\src\main\java\agents
mkdir stock-predictor-project\docker
mkdir stock-predictor-project\docs
```

The file structure in the sidebar includes:

- EXPLORER
  - COS\_30018\_105098233\_OPTION\_3
    - .venv
    - docs
    - jade-agents
      - lib
      - src\main\java\agents
    - python-ml
      - data
      - models
      - notebooks
      - src
        - stock\_prediction.py
      - tests
    - LICENSE
    - README.md
    - requirements.txt

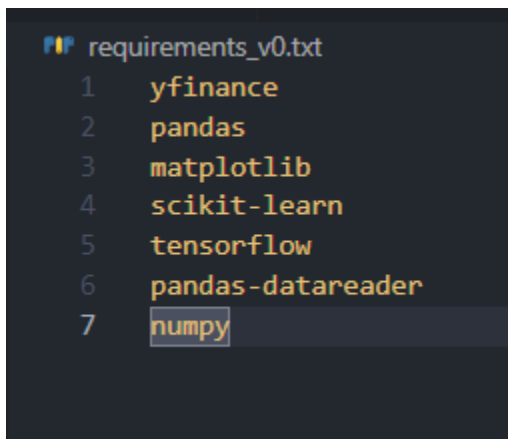
The file structure has multiple directories for storing the data, trained models, taking notes of each of week's tasks and keeping the source code of the project. I also created a specific **virtual**

**environment (venv)** which allows me to use this python environment to run the project instead of changing the configuration and adding module on the global python of my system.

The second task of the set up was to create a **requirements file** which basically is the list of all the different modules that will be used for the project. This allows me to automatically install all the modules with a single line of command instead of manually installing each module.

Following modules were installed with their application in the project:

- Yfinance: For stock prices.
- Pandas: For data analysis and manipulation tools.
- Matplotlib: For plotting the predictions and the actual stock prices.
- scikit-learn: For implementing numerous data modeling.
- Tensorflow: For building and deploying Neural networks.
- pandas-datareader: For extracting data from internet sources directly into a pandas.
- numpy: For numeric calculations on large arrays of data.

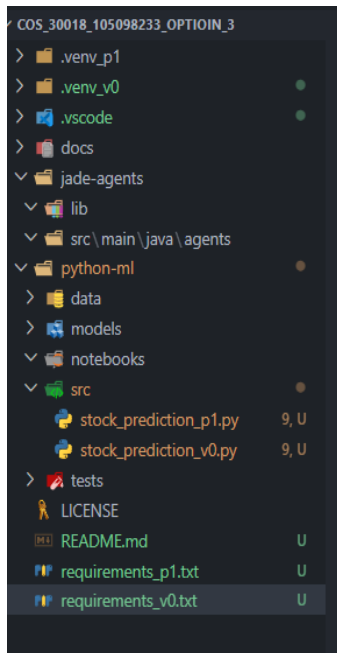


```
requirements_v0.txt
1  yfinance
2  pandas
3  matplotlib
4  scikit-learn
5  tensorflow
6  pandas-datareader
7  numpy
```

Since I will be working with given starter code, we will be needing 6 modules as for now however I can easily add more modules if I want to incorporate more modules in my project.

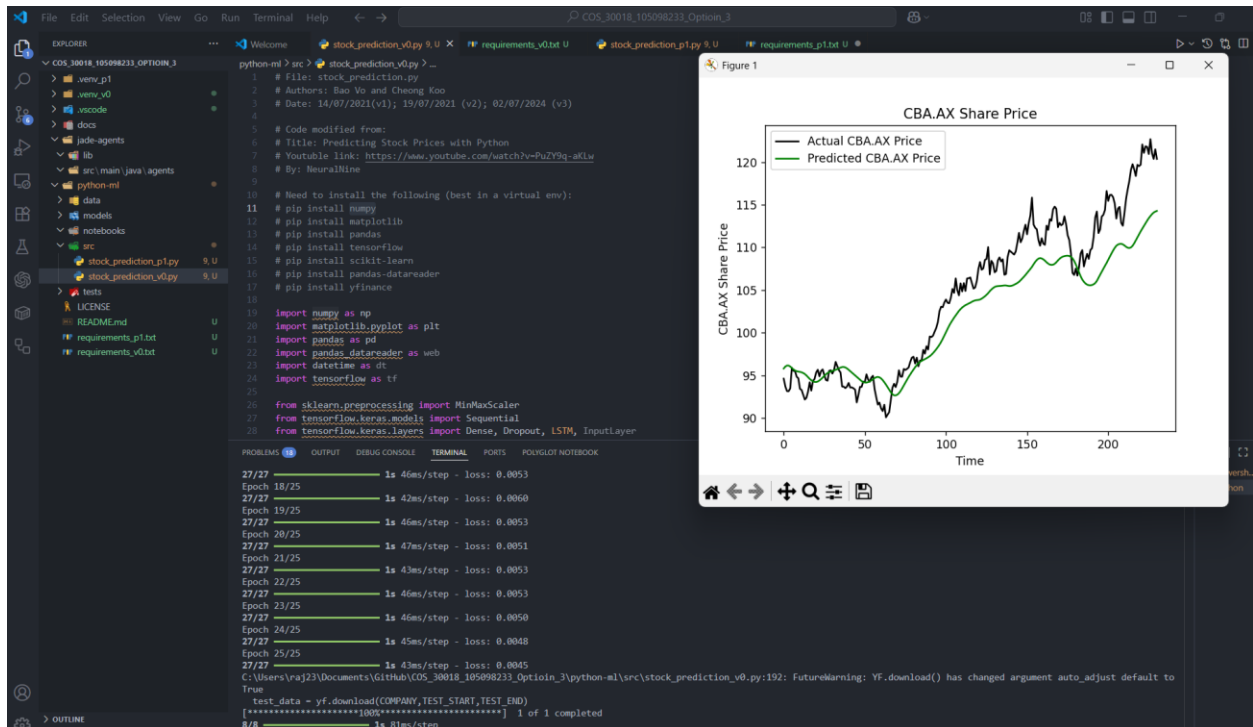
## Task 2: Summaries of your attempts to test the provided code bases (v0.1 and P1) with screenshots.

The above structure is the one that I will be following however, in the first task we were supposed to test out two code bases which presented its own challenges as there were some web socket combabilities issues. So, to get around that issue what I did was two virtual environments which allowed me to get around the previous issue.

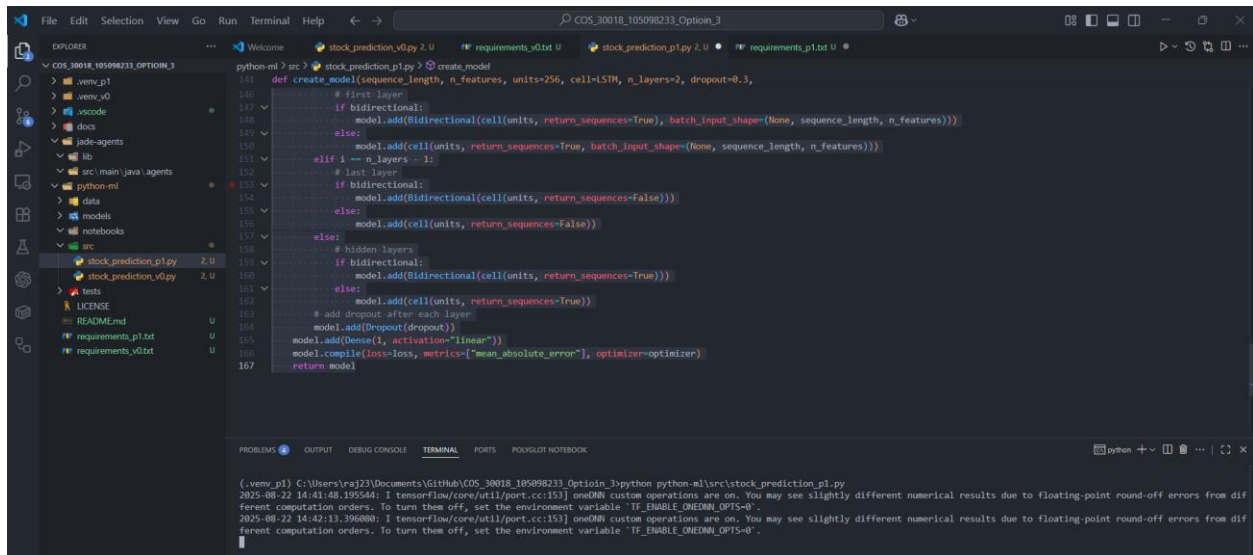


The screen shot shows two virtual environments (i.e. .venv\_v0 and .venv\_p1). I used the .venv\_v0 to run code base given as a starter code for this project and .venv\_p1 for the given GitHub code base to learn and use it improve the starter code.

The output for the starter code: Provides the prediction and actual price of stocks in a graph using matplotlib.



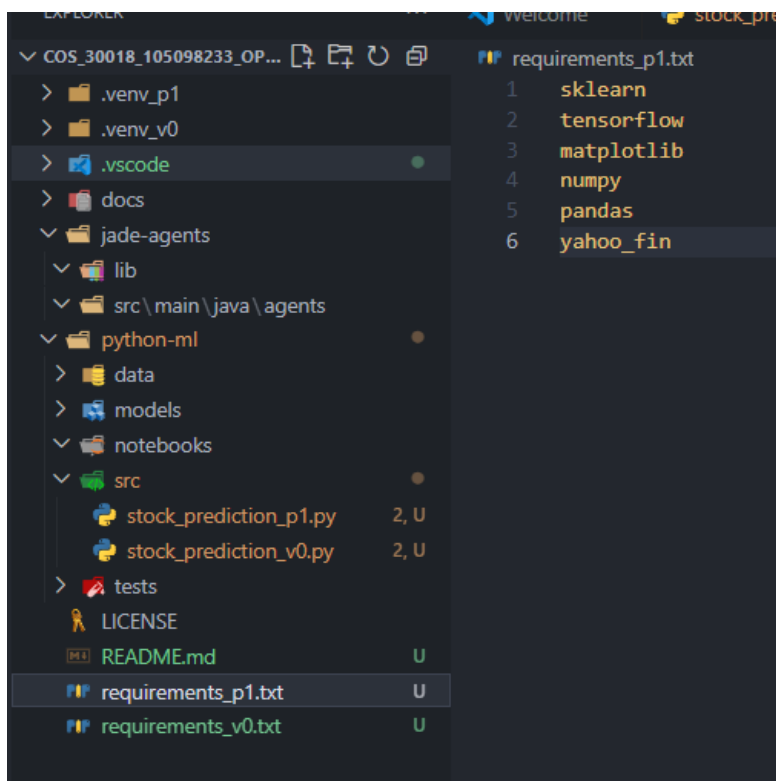
The output for the GitHub code base: it only **defines functions** and sets random seeds. Running the code doesn't produce any output as it doesn't call functions. To get meaningful output, we must alter the code base so here the only thing that I can do is to learn implementation of certain modules and techniques to improve my own initial code base.



```
def create_model(sequence_length, n_features, units=256, cell=LSTM, n_layers=2, dropout=0.3):
    # first layer
    if bidirectional:
        model.add(Bidirectional([cell(units, return_sequences=True), cell(units, return_sequences=True)]))
    else:
        model.add(cell(units, return_sequences=True, batch_input_shape=(None, sequence_length, n_features)))
    # hidden layers
    elif 1 < n_layers < 2:
        # last layer
        if bidirectional:
            model.add(Bidirectional([cell(units, return_sequences=False), cell(units, return_sequences=False)]))
        else:
            model.add(cell(units, return_sequences=False))
    # hidden layers
    elif n_layers > 2:
        # last layer
        if bidirectional:
            model.add(Bidirectional([cell(units, return_sequences=True), cell(units, return_sequences=True)]))
        else:
            model.add(cell(units, return_sequences=True))
    # add dropout after each layer
    model.add(Dropout(dropout))
    model.add(Dense(1, activation="linear"))
    model.compile(loss=loss, metrics=["mean_absolute_error"], optimizer=optimizer)
    return model
```

```
(.venv_ml) C:\Users\raj23\Documents\GitHub\COS_30018_105098233_Option3>python python-ml\src\stock_prediction_v0.py
2025-08-22 14:41:48.195544: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-08-22 14:42:13.396802: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
```

The GitHub code base also uses few different modules hence the requirements file is different and each of those modules will be installed in its own virtual environment.



```
requirements_p1.txt
1 sklearn
2 tensorflow
3 matplotlib
4 numpy
5 pandas
6 yahoo_fin
```

## Task 3: Summary of your understanding of the initial code base v0.1.

The code base major focus is on the usage of an AI model known as an LSTM Neural Network, which uses data from the previous stock market to predict the future. The code base has made use of data from the last 60 days. The code also focuses just on the stock's closing price; the most straightforward equivalent would be to analyze the stock price from the previous 60 days and forecast the closing price for day 61.

According to my current understanding, the code base can be divided into four pieces. Parts would include:

- **Data collection:** The code retrieves stock prices from Yahoo Finance's API for the past 60 days.
- **Preparing the data:** The fetched data cannot be directly used to train the AI model like the neural network used in the code. Therefore, it must be prepared and scaled. For example, the lowest price is considered zero, the highest price is considered one, and all other prices fall between 0 and 1.
- **Building the AI model:** Using the LSTM model with sequential layers, we create an AI model capable of learning from past trends and making future predictions.
- **Model Training and Testing:** After creating a model, it is necessary to train it. For example, we can train the model using data from 2012 to 2020 and then use it to generate predictions in 2025. The same thing was done in the code, and the results were displayed in a graph, with the black line representing the actual trend of real stock prices and the green line representing the AI model's predictions for stock prices.

The initial code displayed in the video has fatal faults because the predictions were incorrect and didn't precisely represent different stock values. The model's most fatal flaw was that it predicted that tomorrow's price would be like that of today, which means that it cannot predict sudden price drops.