**Task Report Cos30018 Option B**

**B.2: Data Processing 1**

**Name: Le Bao Nguyen**

**Student Id: 104169837**

1. Importing libraries:

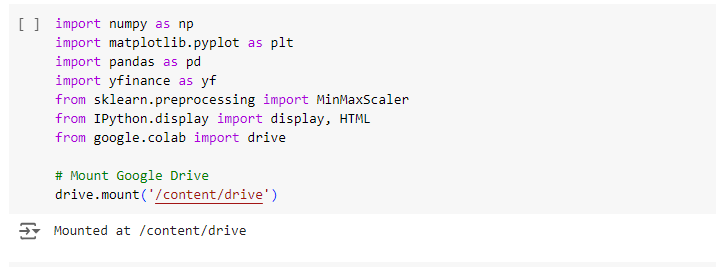


Figure 1: Importing libraries to run the code.

* The script import several libraries:
* "numpy": For effective mathematical operations, especially when working with arrays, it is imported as "np".
* "matplotlib.pyplot": In order to create plots and data visualizations, it is imported as "plt".
* "pandas": It is imported as "pd" to provide strong data analysis and modification features.
* "yfinance": Importing it as "yf" makes it simple to download Yahoo Finance stock market data.
* "MinMaxScaler" from "sklearn.preprocessing": In order to ensure that all values are scaled between 0 and 1, it is used to normalize the data, which is essential for neural network performance.
* "IPython.display": It's used to present HTML content and dataframes in Jupyter notebooks in an organized manner.
* "Drive" from "google.collab": Import drive from google collab to connect and save file to google drive.

1. Data loading and processing:

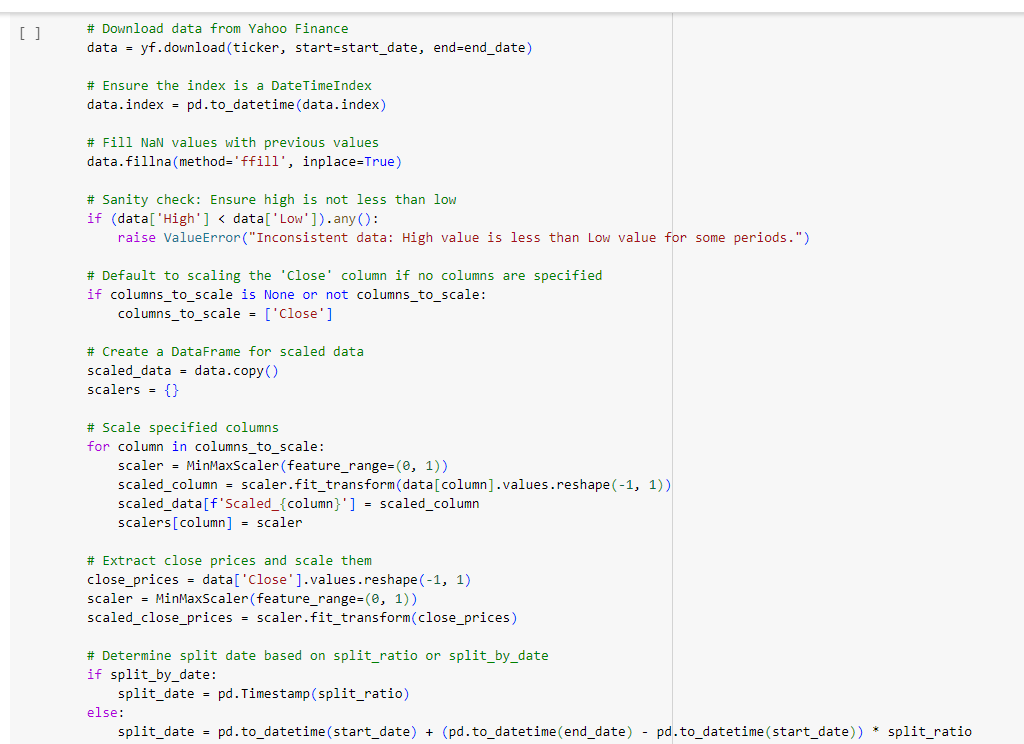


Figure 2: Loading and processing data (1).

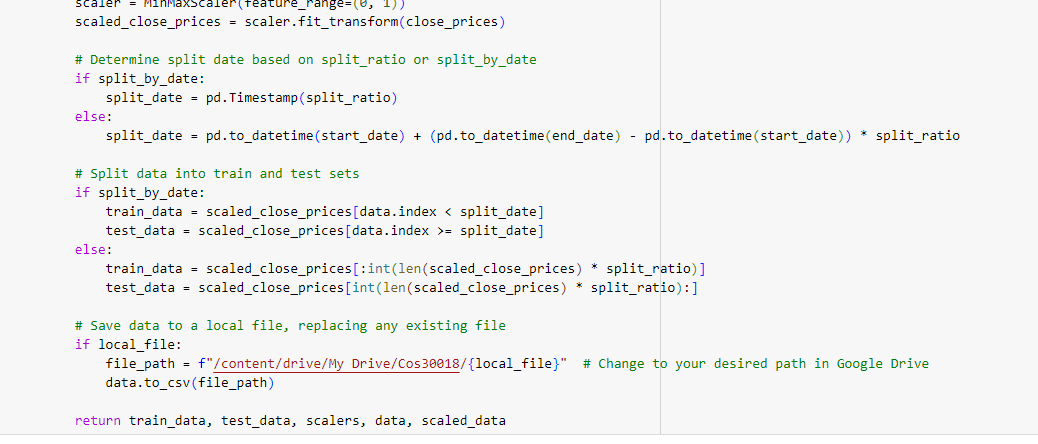


Figure 3: Loading and processing data (2).

* For the specified ticker and date range, the "load\_and\_process\_data" function is intended to retrieve and preprocess stock market data. First, use "yfinance" to download the data from Yahoo Finance. The downloaded data is saved to a CSV file if a "local\_file" path is provided.
* Any missing values (NaNs) in the dataset are filled using the forward-fill approach, which replaces the last valid observation for missing values, in order to maintain data integrity.
* The "Close" prices are then extracted by the function, which then reshapes them into a column vector and uses "MinMaxScaler" to normalize them to a range of 0 to 1. The neural network's performance depends on this scaling. Also, it can scale to a specified column instead of scaling all. If there is no column to specified column is scaled, it auto scales the close column.
* A specified ratio or split date are then used for splitting the data into training and testing sets. Splitting data based on a certain date is applied if "split\_by\_date" is set to "True"; if not, the ratio is calculated.

1. Displaying data in a custom table:

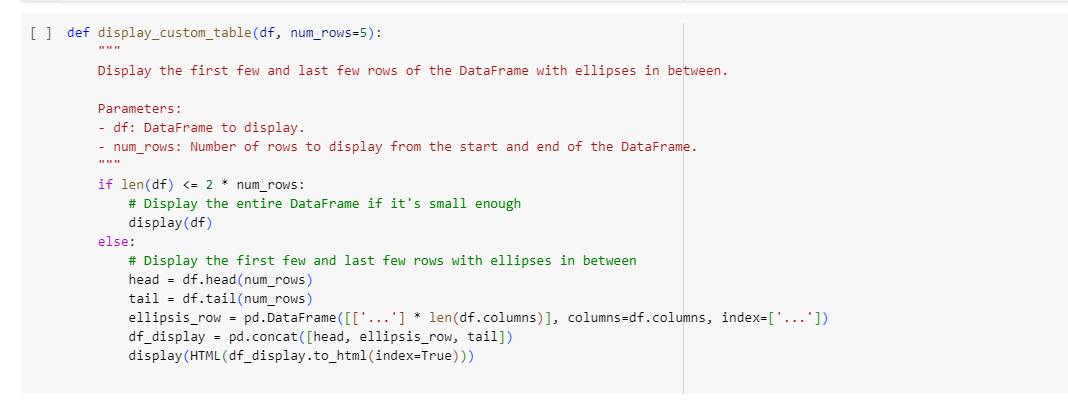


Figure 4: Displaying the data from csv file.

* The "display\_custom\_table" function aims to display a DataFrame's contents in an organized manner. It shows only the first and final few rows, with ellipses splitting them to show that the middle part of the DataFrame is hidden. This method is especially helpful for handling big datasets since it provides an overview of the start and finish of the data without overloaded the user with details.
* The function decides whether to display the DataFrame in entirety or in sections, with an ellipsis row placed between the head and tail, according on how long it is. The HTML rendering capabilities of "IPython.display'" are then used to display the resulting DataFrame.

1. Main script run:



Figure 5: The script to run the code and change the date.

* The process of predicting stock prices is managed by the main script. The stock ticker (AMZN), the data collecting date range, and the location to a local CSV file for data storage are set first.
* We can add specify columns to scale the value.
* To download and preprocess the stock data and split it into training and testing sets, the "load\_and\_process\_data" function is invoked. The original stock data is then shown succinctly using the "display\_custom\_table" function

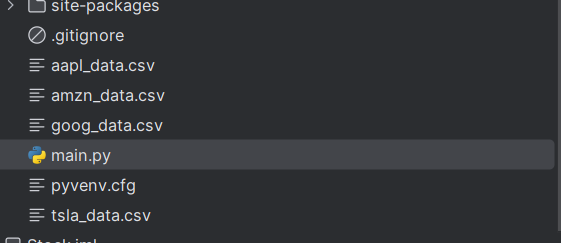


Figure 6: The csv file is loading and saving after running the code (if file\_path = local\_file).

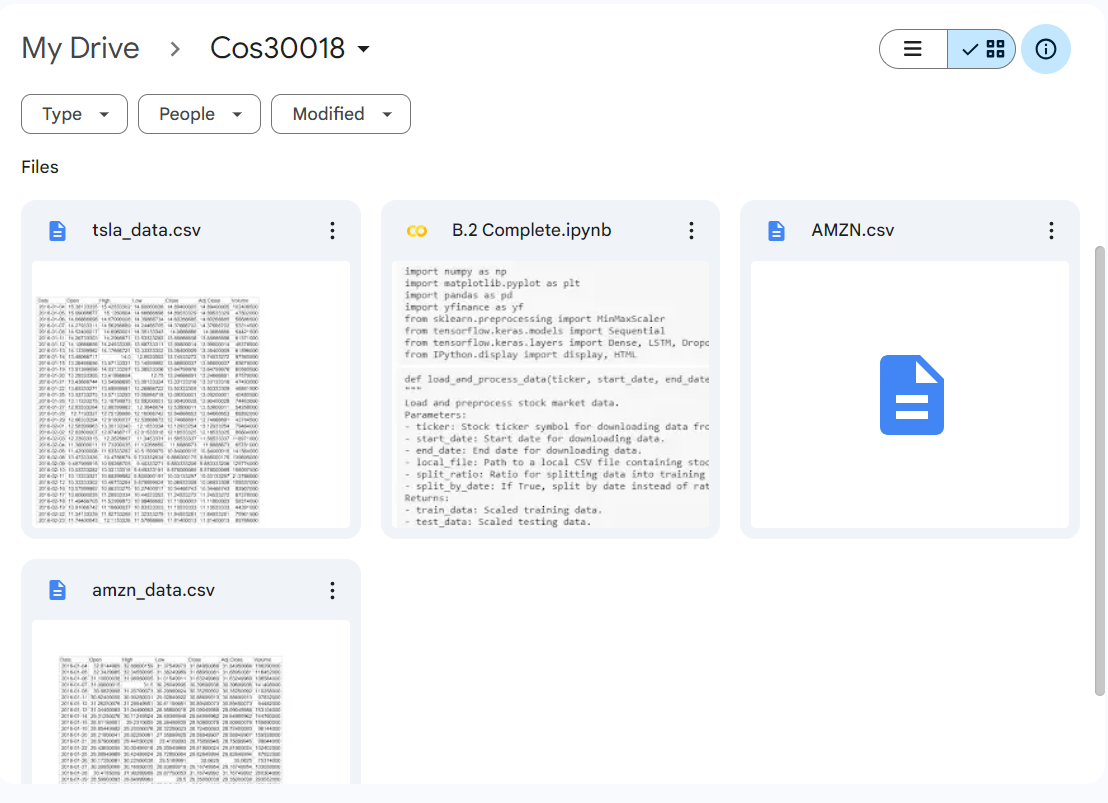


Figure 7: The csv file is loading and saving after running the code (if file\_path = f"/content/drive/My Drive/Cos30018/{local\_file}").

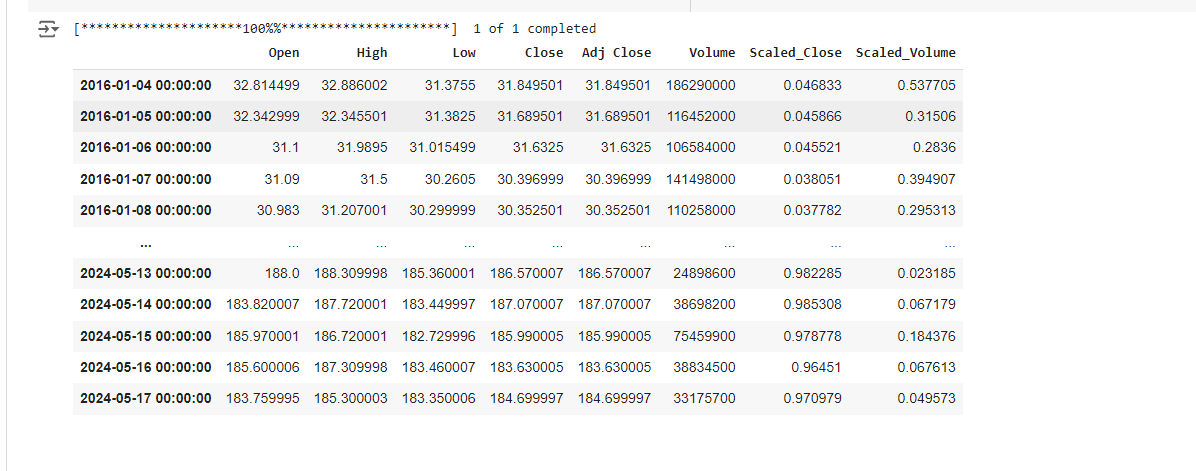


Figure 8: The loading table from csv file.