**` COS30018**

**Intelligent Systems**

**Option B: Stock Prediction**

**Task B.2 – Data Processing**

**Name: Duc Thuan Tran**

**Student ID: 104330455**

**Tutor: Dr. Ru Jia**

**Tutorial: Friday 2:30 – 4:30**

Table of Contents

[Introduction 3](#_Toc175302708)

[Methodology 3](#_Toc175302711)

[Code Analysis 8](#_Toc175302713)

[Model Execution and Outputs 12](#_Toc175302714)

[Execution Process 12](#_Toc175302716)

[GitHub Repository 13](#_Toc175302718)

[References 13](#_Toc175302719)

# **Introduction**

# This task focuses on enhancing the v0.1 stock prediction code by addressing its limitations in data processing. The original version only utilized the 'Close' feature and required manual date selection for training and testing datasets. Additionally, it lacked effective handling of NaN values and flexible data splitting options.

# To improve this, I developed a more versatile data processing function that supports multiple features, flexible date selection, robust NaN handling, and various data splitting methods. This work builds on the v0.1 code, integrating new functionalities for more accurate and efficient stock market predictions.

# **Methodology**

To enhance the organization and maintainability of the stock prediction project, the original source code was divided into four distinct files: **main.py, model\_operations.py, predictor.py, and data\_processing.py. Each** file serves a specific purpose in the workflow:

* **main.py**: This file orchestrates the entire stock prediction workflow. It is responsible for integrating all components, including data loading, preprocessing, model building, training, testing, and prediction. It acts as the central script that coordinates the execution of various functions from other modules.



Figure 1. main.py

* **data\_processing.py**: This module handles all tasks related to data loading, processing, and preparation. It includes functions for loading stock data, handling NaN values, splitting the data into training and testing sets, scaling features, and caching the data locally. The enhancements made in this file focus on providing flexibility and efficiency in data handling.

A screenshot of a computer program

Description automatically generated

Figure 2. data\_processing.py

* **model\_operations.py**: This file contains the functions responsible for building, training, and testing the predictive model. The architecture of the model, such as the LSTM layers and the training process, is defined here. By separating model-related operations into this module, the code becomes more modular and easier to manage.

A screenshot of a computer

Description automatically generated

Figure 3. model\_operations.py

* **predictor.py**: The predictor.py file is dedicated to making predictions using the trained model. It includes functions for generating predictions based on the most recent data and performing any necessary inverse transformations to return the predictions to their original scale. This separation ensures that the prediction logic is isolated from other parts of the code, making it easier to maintain and update.

## A screenshot of a computer program Description automatically generated

#### **Figure 4. predictor.py**

The following sections outline the approach taken to address the key requirements: data loading and processing, handling NaN values, data splitting, data caching, and feature scaling.

**1. Data Loading and Processing**

To improve the flexibility of the data processing function, a new function was developed to allow users to specify the start and end dates for the dataset. This function loads stock data for the specified period and ensures that multiple features, such as 'Close' and 'Volume', are utilized, rather than relying solely on the 'Close' price as in the original version. The ability to specify the date range ensures that the model can be trained and tested on any desired period, making the data processing pipeline adaptable to various scenarios.

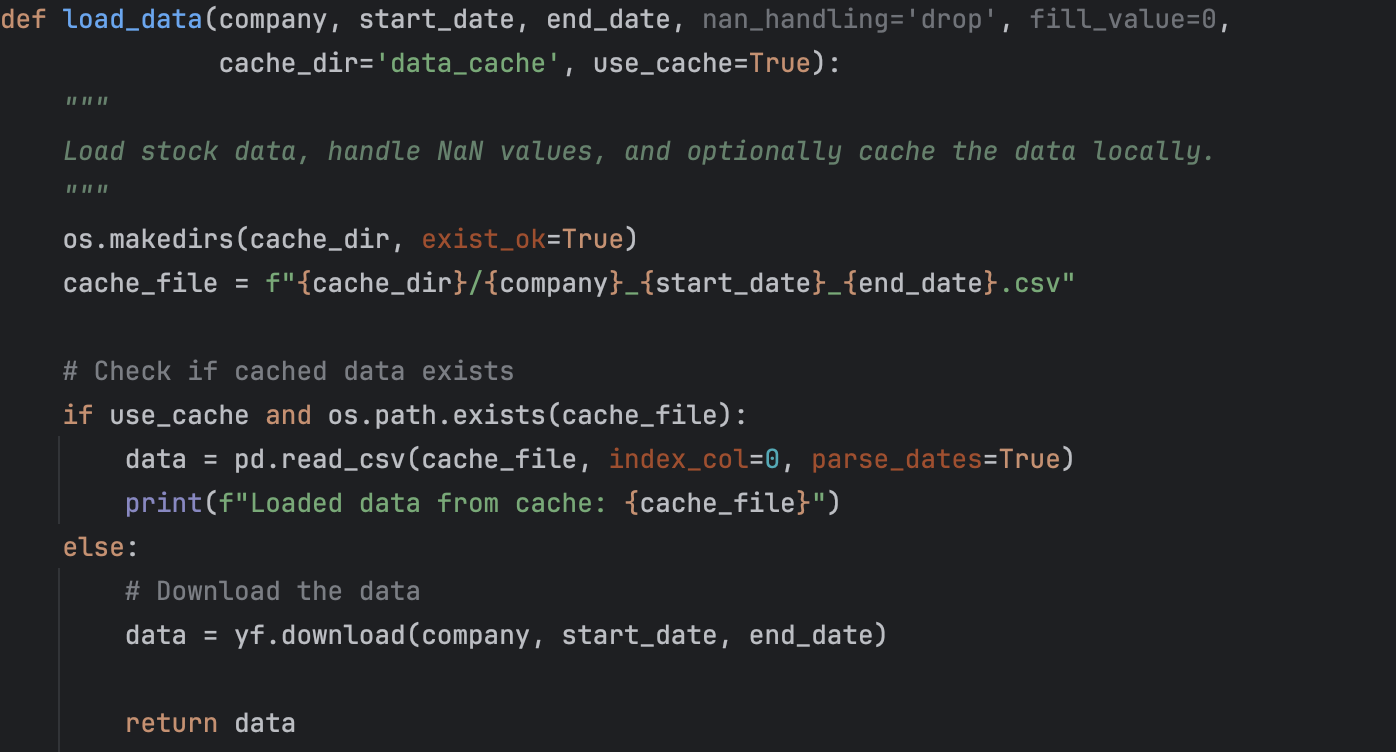


Figure 5. Data loading and processing

**2. Handling NaN Values**

Data quality is critical for accurate predictions, and missing values (NaNs) in the dataset can negatively impact model performance. To address this, the function was designed to handle NaN values using multiple methods: dropping rows with NaNs, filling them with a specific value, or using forward/backward fill techniques. This flexibility allows the user to choose the most appropriate method based on the nature of the dataset and the importance of maintaining data continuity.

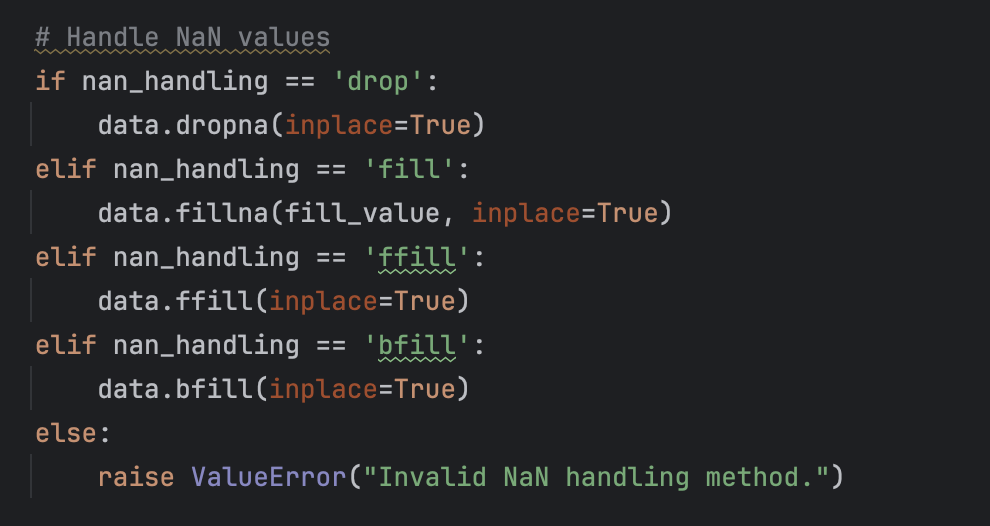


Figure 6. Handling NaN values

**3. Data Splitting**

Effective data splitting is crucial for model evaluation. The function supports various methods for dividing the dataset into training and testing subsets. Users can choose to split the data by a specified ratio (e.g., 80% training, 20% testing), by a specific date (using split\_date), or randomly. This flexibility in splitting methods ensures that the data can be appropriately divided based on the specific requirements of the analysis, whether it be time-based predictions or more generalized model training.

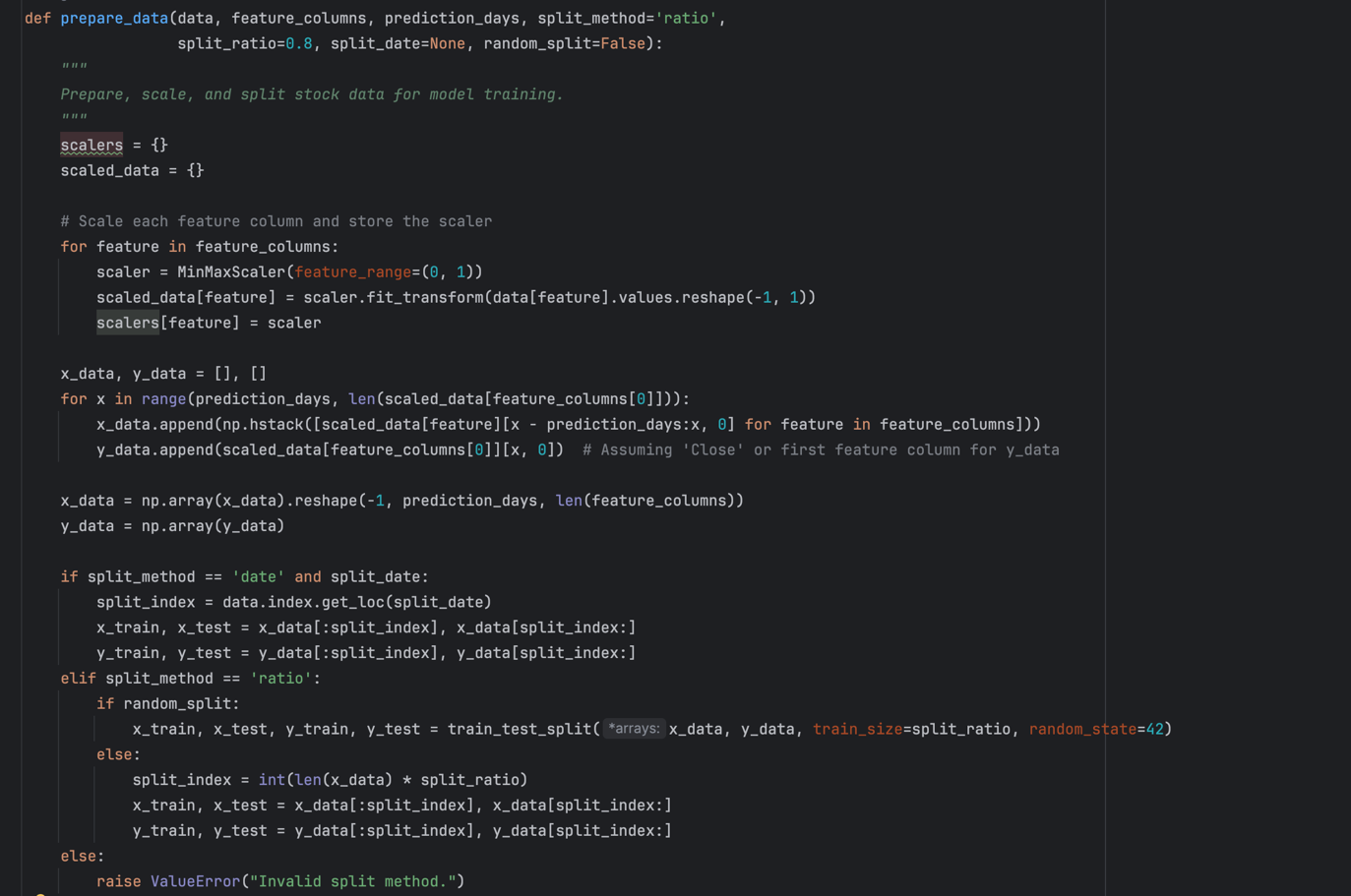


Figure 7. Data Splitting

**4. Data Caching**

To optimize the data retrieval process, especially when working with large datasets or conducting multiple experiments, a caching mechanism was implemented. The function allows the data to be saved locally after the initial download, and subsequent runs can load the data directly from the local cache if available. This reduces the need for repeated data downloads, significantly speeding up the data processing pipeline and making the workflow more efficient.

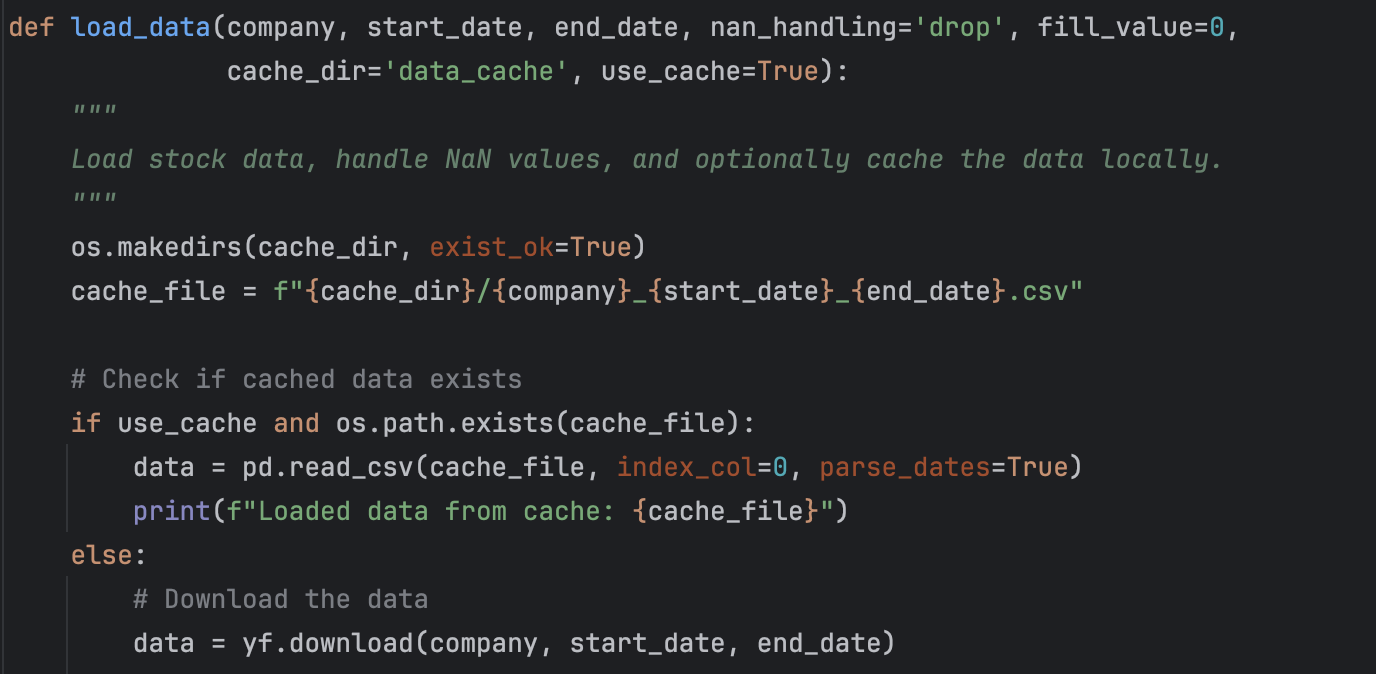


Figure 8. data catching

**5. Feature Scaling**

Scaling the feature columns is a critical step in preparing data for machine learning models, particularly for models like LSTM that are sensitive to the scale of input data. The function was extended to scale multiple features (e.g., 'Close', 'Volume') using MinMaxScaler, which normalizes the data to a specified range (typically 0 to 1). The scalers used for each feature are stored in a dictionary, allowing for easy retrieval and inverse transformation during model evaluation and prediction. This ensures that predictions can be accurately compared to actual values on the original scale.

A screen shot of a computer

Description automatically generated

Figure 9. scaling feature

# **Code Analysis**

The enhancements made to the original stock prediction code required a thorough understanding of several key functions and programming concepts. This section provides a detailed analysis of the most critical components, particularly those that required additional research and careful consideration.

**1. Data Loading and NaN Handling**

The load\_data function in data\_processing.py is central to the entire workflow. It not only retrieves stock data from Yahoo Finance but also addresses potential issues with missing values (NaNs) in the dataset.

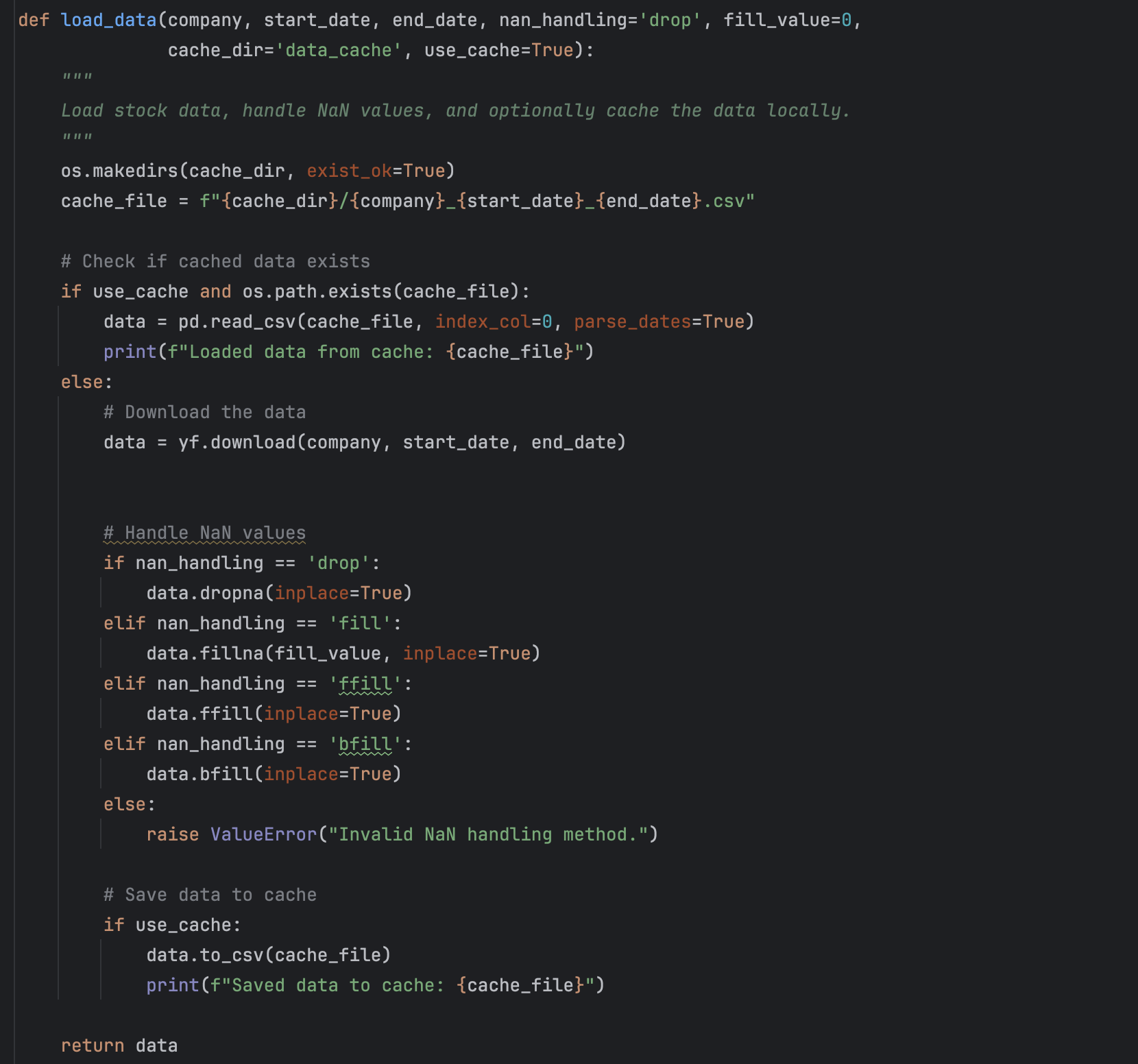


Figure 10. load\_data function

* **Data Downloading:** The function uses yfinance to download stock data for a specified date range. The use of a cache\_dir ensures that data is stored locally, reducing the need for repeated downloads.
* **NaN Handling:** The function offers multiple options for dealing with missing data, including dropping rows (dropna), filling with a specific value (fillna), and using forward or backward fill (ffill and bfill). The choice of method depends on the nature of the dataset and the specific analysis needs. This flexibility allows users to handle missing data in the way that best suits their analysis.

**2. Data Splitting**

The prepare\_data function is responsible for scaling the data and splitting it into training and testing sets. The function supports splitting by a specified ratio, by a specific date, or randomly.

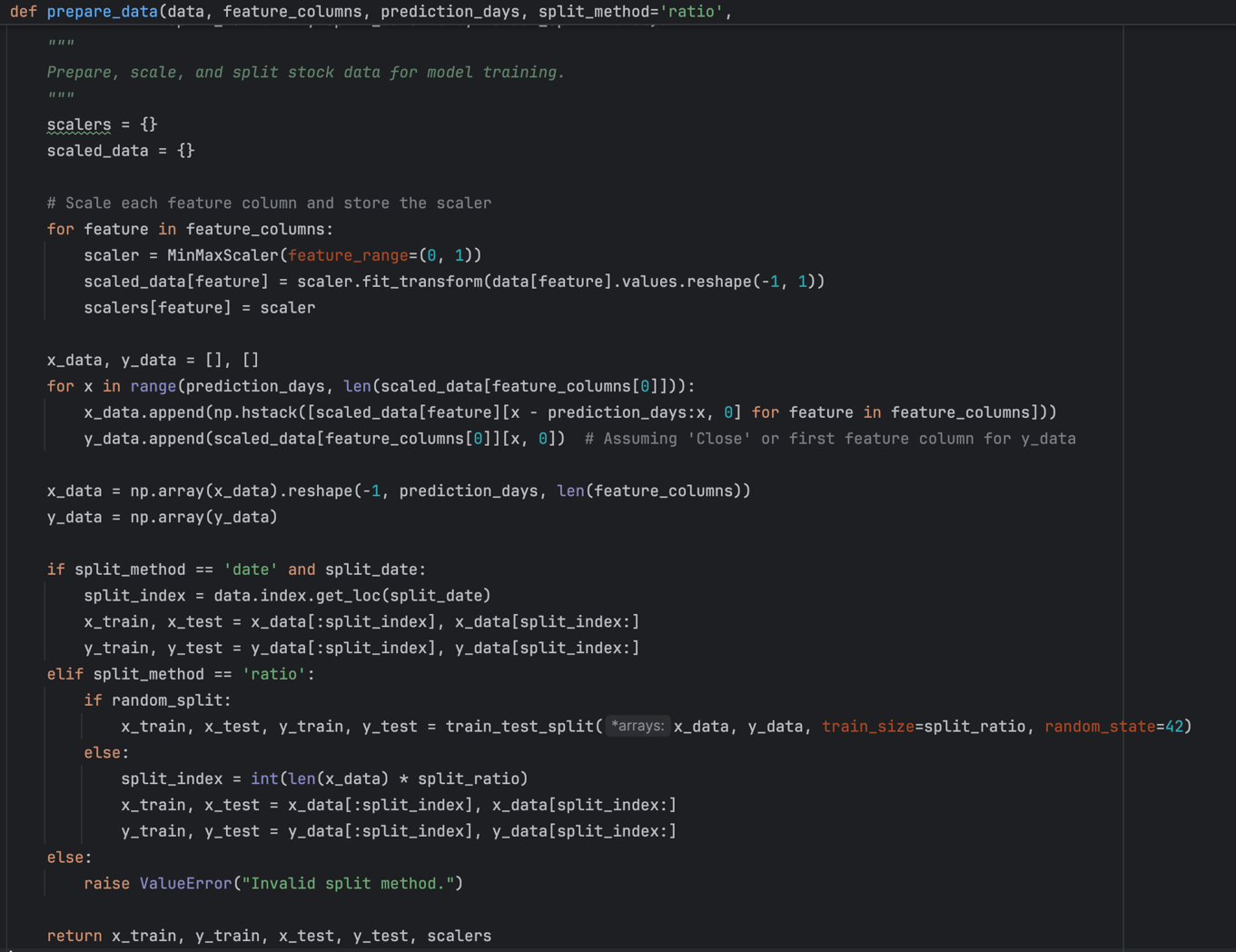


Figure 11. prepare\_data function

* **Feature Scaling:** The function scales each feature using MinMaxScaler, which normalizes the data to a range between 0 and 1. This step is crucial for models like LSTM, which are sensitive to the scale of input data.
* **Data Splitting:** The function supports three methods of data splitting:
* **Ratio-Based Split:** Divides the data into training and testing sets based on a specified ratio.
* **Date-Based Split:** Uses a specific date (split\_date) to separate the training and testing periods.
* **Random Split:** Provides an option to randomly shuffle and split the data, ensuring that the model is not biased by the order of the data.

#### **3. Model Building and Training**

The model\_operations.py file contains the logic for building, training, and evaluating the predictive model. The choice of architecture and training parameters is crucial for model performance.

A computer screen shot of a program code

Description automatically generated

Figure 12. build\_model function

* **LSTM Layers:** The model consists of three stacked LSTM layers, each with 50 units. Stacking multiple LSTM layers allows the model to capture complex temporal dependencies in the data, which is essential for accurate stock price predictions.
* **Dropout Layers:** Dropout is used after each LSTM layer to prevent overfitting by randomly setting a fraction of the input units to zero during training. This helps the model generalize better to unseen data.
* **Output Layer:** A Dense layer with a single unit is used for the final output, which predicts the stock price for the next day.

**4. Prediction and Inverse Transformation**

The predictor.py file is focused on making predictions using the trained model. It includes functions for generating predictions and transforming them back to the original scale.

A screen shot of a computer

Description automatically generated

Figure 13. predict\_next\_day function

* **Prediction Generation:** The function takes the last sequence of input data, uses the trained model to predict the next day's stock price, and then applies an inverse transformation to convert the prediction back to its original scale.
* **Inverse Transformation:** The inverse transformation is necessary because the data was scaled during preprocessing. Without this step, the predicted prices would remain in the normalized range (0-1), making them difficult to interpret.

#### **5. Inclusion of New Variables in main.py**

The main.py file now includes several new variables that enhance the flexibility and control over the data processing and model training process.

A screenshot of a computer program

Description automatically generated

Figure 14. variables in main.py

## **Model Execution and Outputs**

## This section demonstrates how the enhanced stock prediction model runs and the outputs it produces. The goal is to provide a clear overview of the model's performance and its ability to predict stock prices based on the given input data.

## **Execution Process**

## The model was executed on the CBA.AX stock data, covering a training period from January 2020 to August 2023 and a testing period from August 2023 to July 2024. The training process involved 25 epochs, during which the model's loss steadily decreased, indicating effective learning.



Figure 15. Training process

**Visual Output**

The following plot illustrates the model’s predictions against the actual stock prices for the testing period:

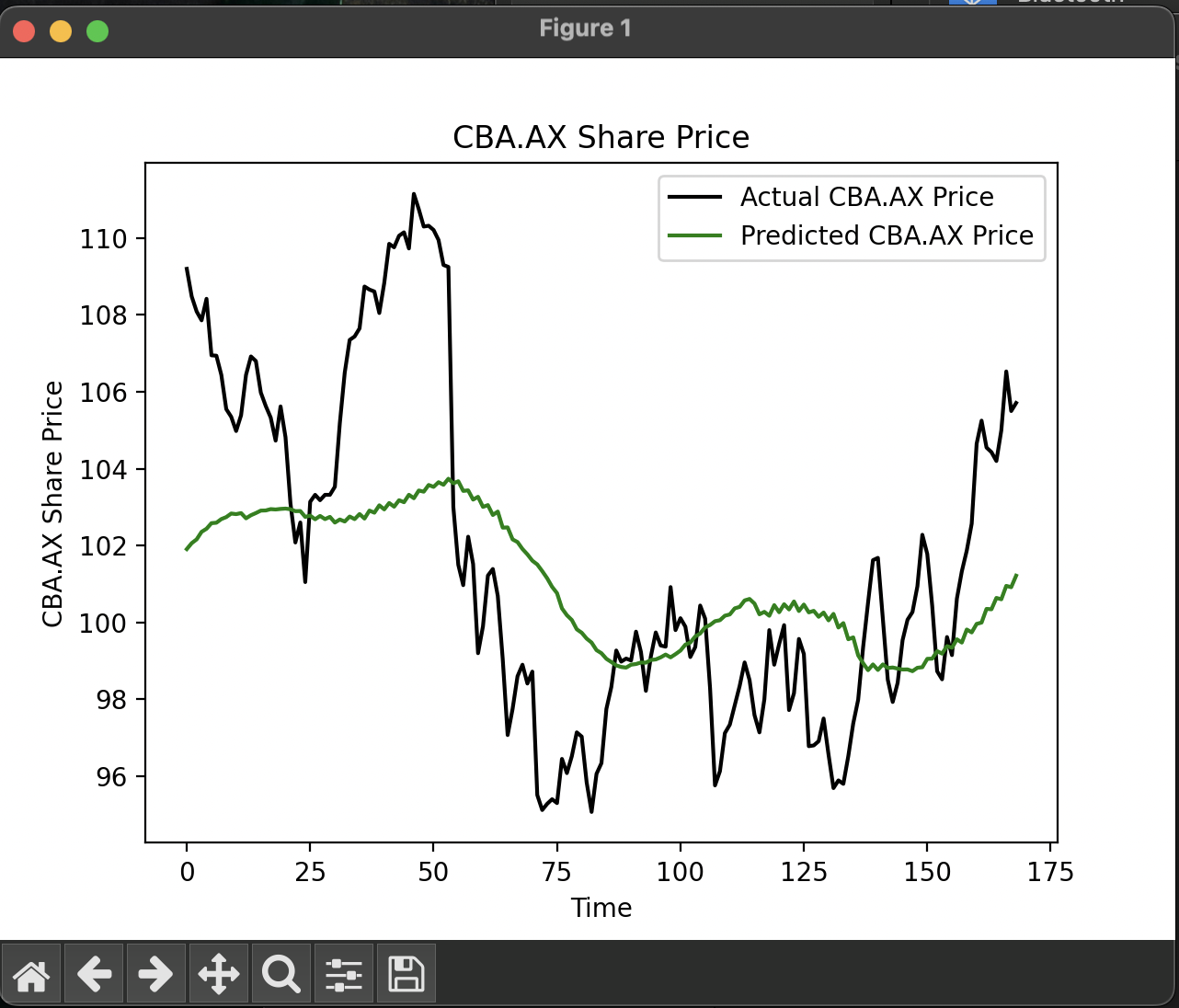


Figure 16. stock prediction result

* **Black Line:** Represents the actual CBA.AX share prices.
* **Green Line:** Represents the model's predicted prices.

This output shows that the model captures the general trend of the stock prices, although some discrepancies are observed during periods of high volatility. These discrepancies highlight areas where further tuning and enhancements could improve prediction accuracy.

## **GitHub Repository**

The project's code base is hosted on GitHub for version control and review. Everyone can access the repository via the following link: [GitHub Repository](https://github.com/ThuanDanchoi/COS30018/tree/main). This repository contains all necessary files for the project and is available for the tutor to review the work.

# **References**

* Simplilearn. (n.d.). *Stock Price Prediction Using Machine Learning*. Retrieved from <https://www.simplilearn.com/tutorials/machine-learning-tutorial/stock-price-prediction-using-machine-learning>
* GeeksforGeeks. (n.d.). *Stock Price Prediction using Machine Learning in Python*. Retrieved from <https://www.geeksforgeeks.org/stock-price-prediction-using-machine-learning-in-python/>
* NeuralNine. (2022, October 1). *Stock Price Prediction using LSTM in Python*. [Video]. YouTube. Retrieved from <https://www.youtube.com/watch?v=CeTR_-ALdRw&list=PL7yh-TELLS1G9mmnBN3ZSY8hYgJ5kBOg-&index=4>
* GeeksforGeeks. (n.d.). *Best Python Libraries for Machine Learning*. Retrieved from <https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/>

# 