**` COS30018**

**Intelligent Systems**

**Option B: Stock Prediction**

**Task B.6 – Machine Learning 2**

**Name: Duc Thuan Tran**

**Student ID: 104330455**

**Tutor: Dr. Ru Jia**

**Tutorial: Friday 2:30 – 4:30**

*Table of Contents*

[1. Introduction 3](#_Toc177908011)

[2. Model Overview 3](#_Toc177908017)

*[3. Main Overview 4](#_Toc177908022)*

[*4. Output and Hyperparameter Testing 7*](#_Toc177908023)

*5.* [*GitHub Repository 12*](#_Toc177908024)

6. [References 12](#_Toc177908025)

# **Introduction**

# The code base has been updated to **version v0.5**, which introduces significant changes, including the use of ensemble methods to combine predictions from multiple models. The primary objective is to improve forecasting accuracy by integrating models such as ARIMA (or SARIMA), a deep learning model (LSTM/GRU), and a Random Forest model.

# The task involves:

# Developing an ensemble approach that incorporates at least two models: ARIMA (or SARIMA) and the existing deep learning model (LSTM/GRU).

# Experimenting with different ensemble configurations, testing multiple models(ARIMA/SARIMA, Random Forest, LSTM/GRU), and optimizing their hyperparameters.

# Visualizing the results of each model and the combined ensemble predictions.

# **Model Overview**

## 2.1 LSTM and GRU (Included Since Version v0.3, task B4)

The LSTM and GRU models were implemented as part of the code since version v0.3. The build\_model function in the **model\_operations.py** file defines the architecture for both models. It allows flexibility in configuring the number of layers, the type of layer (LSTM or GRU), and whether the model should be bidirectional.

* 1. ARIMA and SARIMA

The ARIMA and SARIMA models are handled by the train\_arima\_model and train\_sarima\_model functions in the **model\_operations.py** file. These models were added in version v0.5 to handle linear trends and seasonality in the stock data.

* **ARIMA**: The train\_arima\_model function uses **statsmodels** to build an ARIMA model with the specified order (p, d, q). The function fits the ARIMA model to the stock price data and returns the trained model.

A screen shot of a computer code

Description automatically generated

Image 1. Train\_arima\_model function

* **SARIMA**: Similarly, the train\_sarima\_model function builds and trains a SARIMA model with both non-seasonal and seasonal components. The function returns the fitted model for prediction.

A computer screen with text and numbers

Description automatically generated

Image 2. Train\_sarima\_model function

* The predictions for both models are generated using predict\_arima and predict\_sarima functions, which forecast a specified number of future time steps:

A computer screen with white text

Description automatically generated

Image 3. Predict\_arima & samira function

2.3 Random Forest

## The **Random Forest** model is implemented in the **model\_operations.py** file under the functions train\_random\_forest\_model and test\_random\_forest\_model. Random Forest was added in version v0.5 to handle non-linear patterns in the stock data.

## **train\_random\_forest\_model**: This function builds and trains a Random Forest model using **scikit-learn’s RandomForestRegressor**. It takes training data and hyperparameters such as n\_estimators (number of trees), max\_depth, and min\_samples\_split as input and returns the trained model.

## A screen shot of a computer Description automatically generated

Image 4. Train\_random\_forest\_model

* **test\_random\_forest\_model**: This function takes the trained Random Forest model and the test data to generate predictions for the stock price.

A computer screen shot of a code

Description automatically generated

Image 5. Test\_random\_forest\_model

## **Main Overview**

In **version v0.5**, several key changes were introduced in the main file to implement the ensemble approach. The code integrates the ARIMA and SARIMA models alongside LSTM/GRU and Random Forest, combining their predictions to improve stock price forecasting. Below is an explanation of the modifications starting from the ARIMA and SARIMA models to the end of the code.

3.1 ARIMA and SARIMA Model Integration

The ARIMA and SARIMA models were trained using the train\_arima\_model and train\_sarima\_model functions, respectively. The models were applied to the "Close" price column of the stock data, and their predictions were generated for the test period:

A black background with white text

Description automatically generated

Image 6. Ariam&Samira thorugh training

* ARIMA and **SARIMA** predictions were generated using the predict\_arima and predict\_sarima function:

A black screen with white text

Description automatically generated

Image 7. Ariam&Samira thorugh prediction

Both predictions were stored as arrays to be later combined with other models' outputs.

* 1. LSTM/GRU Model Training

A **GRU** model was built using the build\_model function with 4 layers, a layer size of 100, and bidirectional configuration. This model was trained using the train\_model function, which fits the GRU model on the training data. After training, the model predicted the stock prices for the test set, and the predicted values were inverse transformed to match the original price scale:



Image 8. Build, Train, Predict GRU model

* 1. Weight Assignment and Prediction Combination

The ensemble approach combined the ARIMA, SARIMA, and LSTM/GRU predictions by assigning weights inversely proportional to the Mean Squared Error (MSE) of each model. The MSE for each model was calculated, and weights were normalized:

A screenshot of a computer program

Description automatically generated

Image 9. Calculate, Assign weight, Normalize weight

Using the assigned weights, the final predictions were calculated as a weighted average of the ARIMA, SARIMA, and LSTM/GRU model outputs:

A black background with white text

Description automatically generated

Image 10. Combine Arima, Samira and LSTM or Gru for predictions

* 1. Random Forest Model Training

The **Random Forest** model was trained separately using the train\_random\_forest\_model function. The features were flattened to fit the model, and predictions were made for the test set:

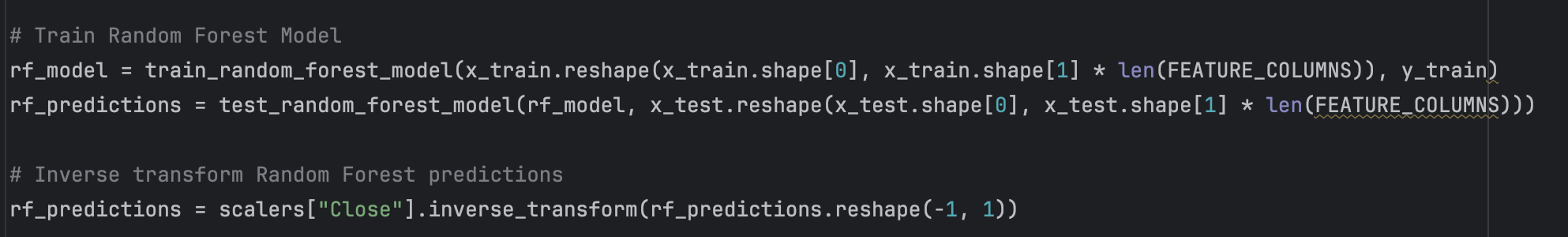


Image 11. Train and inverse transform for Random Forest

3.5 Visualization of Model Predictions

The final step was to visualize the predictions from all models (ARIMA, SARIMA, LSTM/GRU, and Random Forest) alongside the actual stock prices. This was done using **Matplotlib**, where each model’s predictions were plotted as separate lines with different styles and colors:

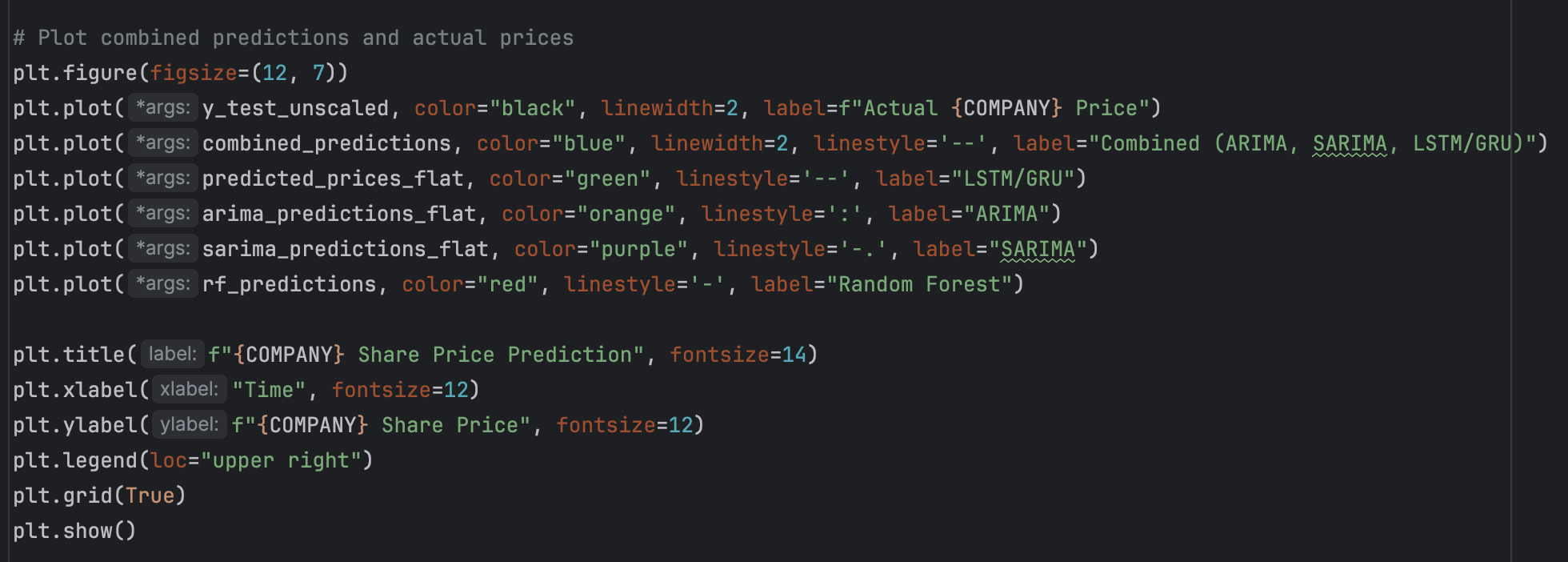


Image 12. Models Predictions Visualization with line graphs

This visualization provides a clear comparison between the actual stock prices and the predictions generated by each model, as well as the combined ensemble prediction.

## **Output and Hyperparameter Testing**

The table below represents the original hyperparameter settings used in the code for each model:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Type | Base Layers | Layer Size | Dropout Rate | Bidirectional |
| LSTM | 4 | 100 | 0.3 | True |
| GRU | 4 | 100 | 0.3 | True |
| ARIMA | Order=(6,1,0) |  |  |  |
| SARIMA | Order=(1,1,1), Seasonal\_Order=(1,1,1,12) |  |  |  |
| RandomForst | n\_estimators=100 | max\_depth=10 | min\_samples\_split=5 |  |

A graph showing different colored lines

Description automatically generated

Image 13. Share Price Prediction (Original)

4.1 Possible Hyperparameters to Modify

Below are the possible hyperparameters that can be adjusted for each model:

**LSTM and GRU:**

* **Number of Layers**: Can range from 1 to 5 or more.
* **Layer Size**: Can vary between 50, 100, 150, or higher.
* **Dropout Rate**: Adjustable between 0.1, 0.3, 0.5, etc.
* **Bidirectional**: Boolean setting, True or False.
* **Learning Rate**: Can range from 0.001 to 0.01 or more.

**ARIMA:**

**Order (p, d, q)**: Can be adjusted to different combinations of parameters:

* p = [0, 1, 2, 6]
* d = [0, 1]
* q = [0, 1, 2]

**SARIMA:**

* **Order (p, d, q)**: Same as ARIMA.
* **Seasonal Order (P, D, Q, s)**: Seasonal components can be varied:
  + P = [0, 1]
  + D = [0, 1]
  + Q = [0, 1]
  + s = Seasonal period, e.g., [12, 6]

**Random Forest:**

* **n\_estimators**: Number of trees in the forest, can be adjusted between 50, 100, 150, etc.
* **max\_depth**: Maximum depth of the tree, can range between 5, 10, 20, etc.
* **min\_samples\_split**: Minimum number of samples required to split an internal node, can range from 2 to 5 or higher.
  1. Experimenting with Different Hyperparameter Configurations

**Case 1: Reducing the Number of Layers and Increasing Dropout (LSTM/GRU)**

* **LSTM**: Reduced to 2 layers, with a smaller size of 50 units, and a dropout rate of 0.5.
* **GRU**: Similar adjustments as LSTM.
* **ARIMA**: Order changed to (2,1,0) for simplicity.
* **SARIMA**: Seasonal order changed to (0,1,1,6), reducing the seasonal period.
* **Random Forest**: Reduced n\_estimators to 50 and set max\_depth to 5.

A graph showing the price of a stock market

Description automatically generated

Image 14. Share Price Prediction (Case 1)

**Case 2: Increasing Model Complexity (LSTM/GRU)**

* **LSTM**: Increased to 5 layers with 150 units per layer and a dropout rate of 0.2.
* **GRU**: Same adjustments as LSTM.
* **ARIMA**: Changed to (6,1,2) to handle more autocorrelations.
* **SARIMA**: Order changed to (1,1,1), Seasonal Order (1,1,0,12).
* **Random Forest**: Increased n\_estimators to 150 and max\_depth to 20.

A graph on a screen

Description automatically generated

Image 15. Share Price Prediction (Case 2)

**Case 3: Adjusting Seasonal and Depth Parameters (SARIMA, Random Forest)**

* **LSTM/GRU**: Kept the default settings.
* **ARIMA**: Adjusted to (1,1,1).
* **SARIMA**: Seasonal order set to (1,1,1,12).
* **Random Forest**: Set min\_samples\_split to 10, increased n\_estimators to 200.

A graph showing different colored lines

Description automatically generated

Image 16. Share Price Prediction (Case 3)

**Case 4: Simplifying Models (Reducing Complexity)**

* **LSTM/GRU**: Reduced to 1 layer, with 50 units, dropout 0.1, no bidirectional setting.
* **ARIMA**: Set to (1,1,1).
* **SARIMA**: Seasonal order reduced to (0,1,0,6).
* **Random Forest**: Reduced n\_estimators to 50 and max\_depth to 5.

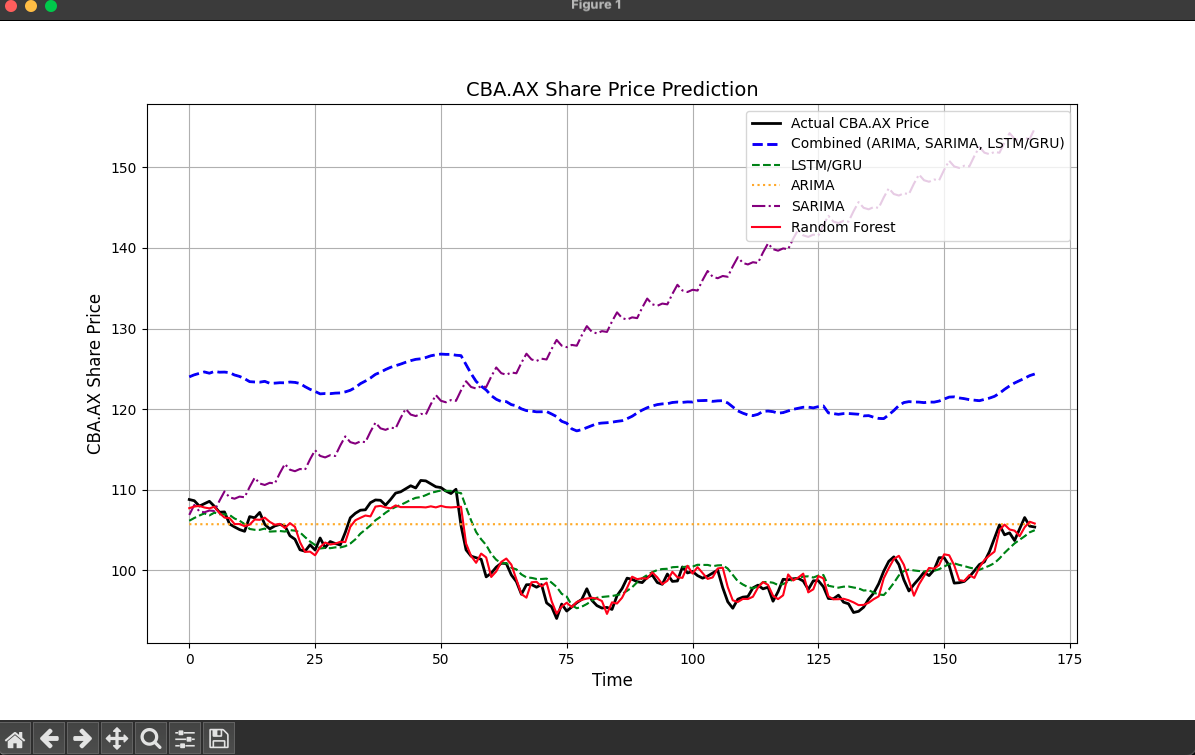


Image 17. Share Price Prediction (Case 4)

## 

* 1. Comparison of Model Predictions Across Different Hyperparameter Cases

**Original Configuration:**

In the original configuration, the combined ensemble prediction (blue dashed line) seems to overestimate the stock price significantly after around 50 time steps. The individual predictions from ARIMA (orange dotted line), SARIMA (purple dash-dot line), LSTM/GRU (green dashed line), and Random Forest (red solid line) mostly stay closer to the actual stock price (black solid line).

* **Strength**: The Random Forest and LSTM/GRU models track the actual price better than the ARIMA and SARIMA models.
* **Limitation**: The combined ensemble prediction diverges from the actual prices, showing that the weights might not be well-balanced between the models.

**Case 1 (Second Image):**

In Case 1, where the number of layers in the LSTM/GRU models was reduced and the dropout rate was increased, the combined prediction still overestimates the price. However, it stays relatively closer to the actual prices, compared to the original configuration.

* **Strength**: The combined ensemble prediction is more aligned with the actual prices compared to the original configuration.
* **Limitation**: The SARIMA and ARIMA models still fail to capture the true dynamics of the stock price, particularly in periods of volatility.

**Case 2 (Third Image):**

In Case 2, with increased model complexity (more layers and units in LSTM/GRU), the individual model predictions (particularly LSTM/GRU and SARIMA) start to diverge further from the actual price, especially after 50 time steps.

* **Strength**: The combined model still slightly improves over the individual model performances.
* **Limitation**: The increased model complexity didn’t improve the predictive power; instead, it led to more overfitting, as seen in the SARIMA and LSTM/GRU lines.

**Case 3 (Fourth Image):**

In Case 3, focusing on adjusting the seasonal and depth parameters for SARIMA and Random Forest, the combined prediction shows the least divergence from the actual stock price. The Random Forest prediction closely follows the actual price for most of the time.

* **Strength**: This case shows a better balance between the individual model contributions, with Random Forest providing a stable prediction close to the actual stock price.
* **Limitation**: The SARIMA model still tends to drift away from the actual price with a strong upward trend, which is unrealistic in some periods.

**Case 4 (Fifth Image):**

In Case 4, where model complexity was reduced, the combined prediction shows moderate overestimation but follows a similar pattern as in Case 1. The Random Forest and LSTM/GRU models perform better here with less complexity.

* **Strength**: Reducing the complexity allowed for better performance by the ensemble without overfitting.
* **Limitation**: The ensemble prediction is still overestimating the stock price but is an improvement from Case 2.

**Overall Observations:**

* The **original configuration** overestimated the stock price in the long term, while **Case 1** offered some improvements by reducing the model complexity.
* **Case 2**, which increased complexity, resulted in more divergence from the actual prices, indicating overfitting.
* **Case 3** provided the best overall balance, especially with the Random Forest model closely matching the actual price, while the combined model performed reasonably well.
* **Case 4** was like Case 1 but slightly better in terms of ensemble prediction alignment with the actual price.

## **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**

* https://www.youtube.com/watch?v=UuBigNaO\_18NeuralNine. (2022, October 1). *Stock Price Prediction using LSTM in Python*. [Video]. YouTube. Retrieved from
* <https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/>
* <https://commtelnetworks.com/exploring-the-depths-unraveling-the-intricacies-of-machine-learning-and-deep-learning/>
* <https://www.tensorflow.org/api_docs/python/tf/keras/Sequential>

# 