Task C.6 – Machine Learning 3

COS30018 – Intelligent Systems

Student Name: Rahul Raju
Student ID: 105143065

Introduction

This report explores the prediction of stock prices by implementing and comparing deep learning and statistical models using historical market data. An ensemble approach combining results from LSTM, ARIMA and SARIMAX is evaluated for Task C.6 to assess forecasting accuracy and robustness.

Requirements

Virtual Environment:

This project has migrated to Google Colab to leverage the convenience of cloud execution and GPU acceleration. Workflows are compatible with local Jupyter Notebooks for reproducibility.

Key Dependencies:

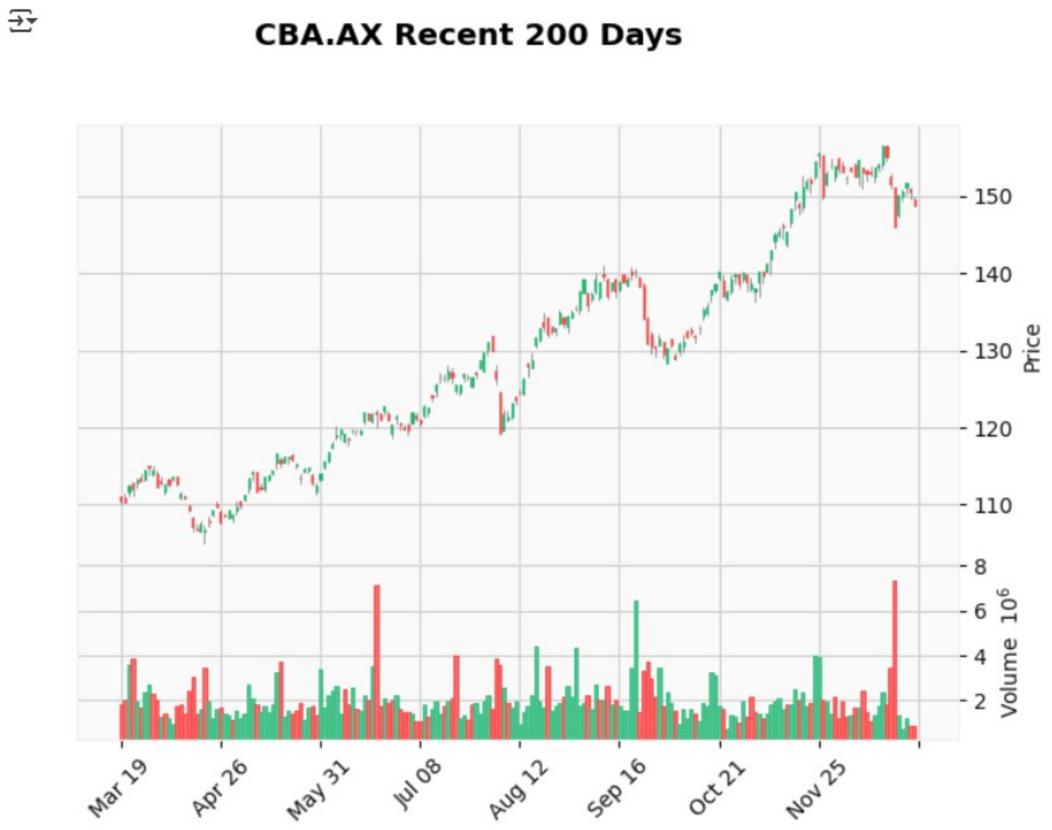
The following Python libraries and modules were used:

- numpy
- pandas
- scikit-learn
- keras / tensorflow
- matplotlib
- plotly
- statsmodels

Installation in Colab or Jupyter can be performed using **!pip install** commands for each library.

Data Preprocessing

Historical market prices for CBA.AX, specifically the Close value and trading Volume, were ingested as time series and split into training and testing sets. Feature scaling was employed using MinMaxScaler to normalise inputs for deep learning models. Sequence generation prepared the data for LSTM/ GRU by producing rolling windows of fixed length to capture temporal dependencies.



Model Architectures

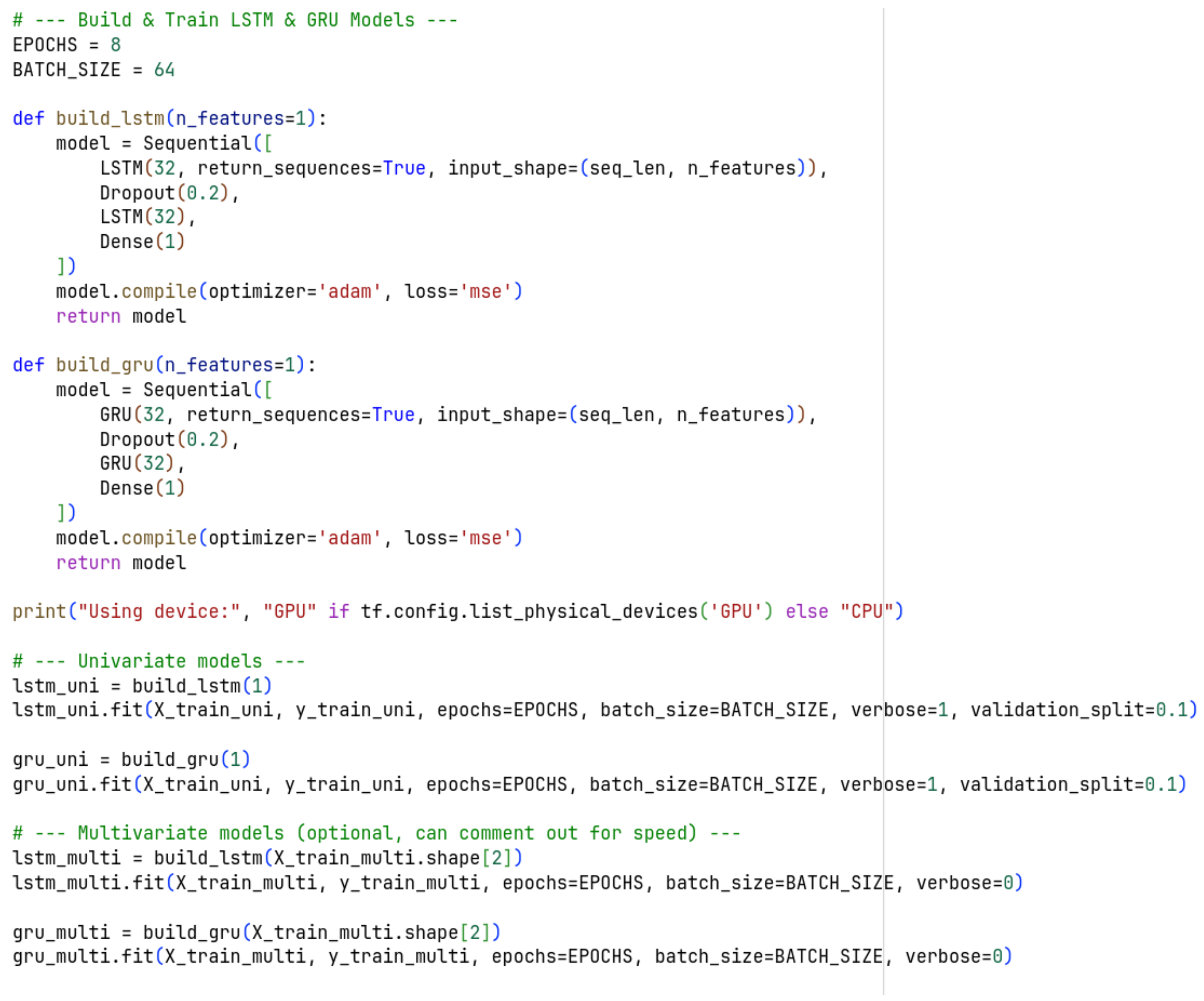
Deep Learning: LSTM and GRU

Both LSTM and GRU models were constructed using Keras' Sequential API with the following design:

- Two stacked recurrent layers (LSTM or GRU, 32 units each)
- Dropout between layers (rate 0.2)
- Fully-connected output layer

Separate models were trained for univariate (Close only) and multivariate (OHLCV) scenarios, with model inputs reshaped according to sequence length and feature dimension. Training was performed for eight epochs per run, and validation was used to monitor mean squared error.

1. **Univariate LSTM and GRU:** Trained on X_train_uni and y_train_uni to forecast using a single feature. Training includes a validation_split=0.1, meaning 10% of the training data is used for validation during training.
2. **Multivariate LSTM and GRU:** Trained on X_train_multi and y_train_multi to forecast using multiple features (eg, price, volume, etc). The n_features parameter is dynamically set to the number of features in the training data. For speed, verbose output is disabled (verbose=0).



```python
# --- Build & Train LSTM & GRU Models ---
EPOCHS = 8
BATCH_SIZE = 64

def build_lstm(n_features=1):
    model = Sequential([
        LSTM(32, return_sequences=True, input_shape=(seq_len, n_features)),
        Dropout(0.2),
        LSTM(32),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

def build_gru(n_features=1):
    model = Sequential([
        GRU(32, return_sequences=True, input_shape=(seq_len, n_features)),
        Dropout(0.2),
        GRU(32),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

print("Using device:", "GPU" if tf.config.list_physical_devices('GPU') else "CPU")

# --- Univariate models ---
lstm_uni = build_lstm(1)
lstm_uni.fit(X_train_uni, y_train_uni, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=1, validation_split=0.1)

gru_uni = build_gru(1)
gru_uni.fit(X_train_uni, y_train_uni, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=1, validation_split=0.1)

# --- Multivariate models (optional, can comment out for speed) ---
lstm_multi = build_lstm(X_train_multi.shape[2])
lstm_multi.fit(X_train_multi, y_train_multi, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=0)

gru_multi = build_gru(X_train_multi.shape[2])
gru_multi.fit(X_train_multi, y_train_multi, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=0)
```

Deep Model Predictions and RMSE

Function: *inverse_scale(data, scaler)*

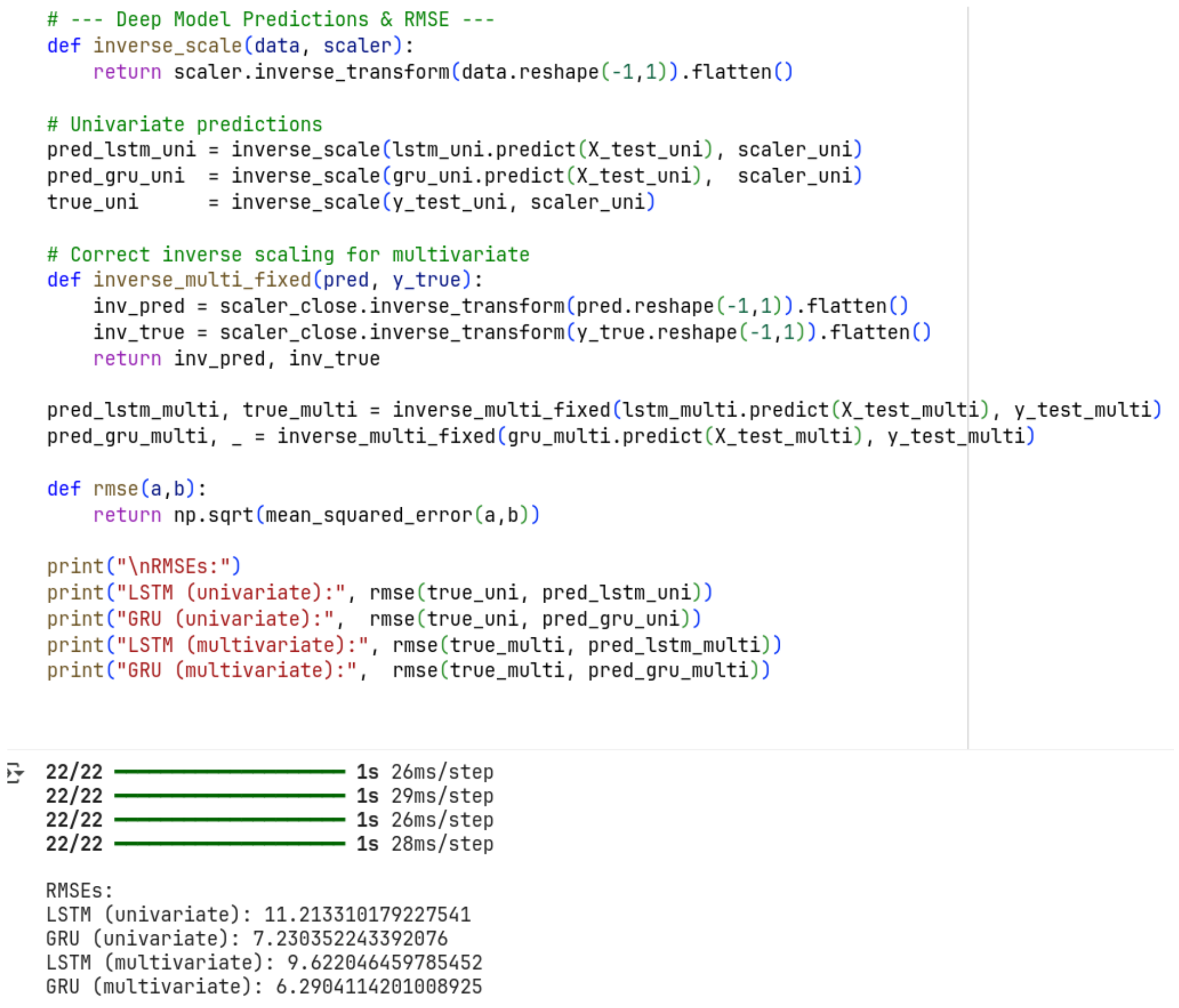
Reverses the scaling applied to the data, transforming model predictions and target values back to their original, interpretable units (eg, actual stock price). Returns a 1D NumPy array of data in its original scale.

Function: *inverse_multi_fixed(pred, y_true)*

A specialised function to correctly inverse-scale predictions and true values from the multivariate models. This is necessary because the multivariate dataset was likely scaled feature-wise, and this function ensures only the 'close' price feature is inverse-transformed correctly. Returns a tuple containing two 1D NumPy arrays: (inv_pred, inv_true).

Function: *rmse(a, b)*

Calculates the Root Mean Square Error (RMSE), a standard metric to evaluate forecasting accuracy. It measures the standard deviation of prediction errors. It returns a single float value representing the RMSE. A lower RMSE indicates a better model fit.



```python
# --- Deep Model Predictions & RMSE ---
def inverse_scale(data, scaler):
    return scaler.inverse_transform(data.reshape(-1,1)).flatten()

# Univariate predictions
pred_lstm_uni = inverse_scale(lstm_uni.predict(X_test_uni), scaler_uni)
pred_gru_uni  = inverse_scale(gru_uni.predict(X_test_uni),  scaler_uni)
true_uni      = inverse_scale(y_test_uni, scaler_uni)

# Correct inverse scaling for multivariate
def inverse_multi_fixed(pred, y_true):
    inv_pred = scaler_close.inverse_transform(pred.reshape(-1,1)).flatten()
    inv_true = scaler_close.inverse_transform(y_true.reshape(-1,1)).flatten()
    return inv_pred, inv_true

pred_lstm_multi, true_multi = inverse_multi_fixed(lstm_multi.predict(X_test_multi), y_test_multi)
pred_gru_multi, _ = inverse_multi_fixed(gru_multi.predict(X_test_multi), y_test_multi)

def rmse(a,b):
    return np.sqrt(mean_squared_error(a,b))

print("\nRMSEs:")
print("LSTM (univariate):", rmse(true_uni, pred_lstm_uni))
print("GRU (univariate):",  rmse(true_uni, pred_gru_uni))
print("LSTM (multivariate):", rmse(true_multi, pred_lstm_multi))
print("GRU (multivariate):",  rmse(true_multi, pred_gru_multi))
```

```
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step

RMSEs:
LSTM (univariate): 11.213310179227541
GRU (univariate): 7.230352243392076
LSTM (multivariate): 9.622046459785452
GRU (multivariate): 6.2904114201008925
```

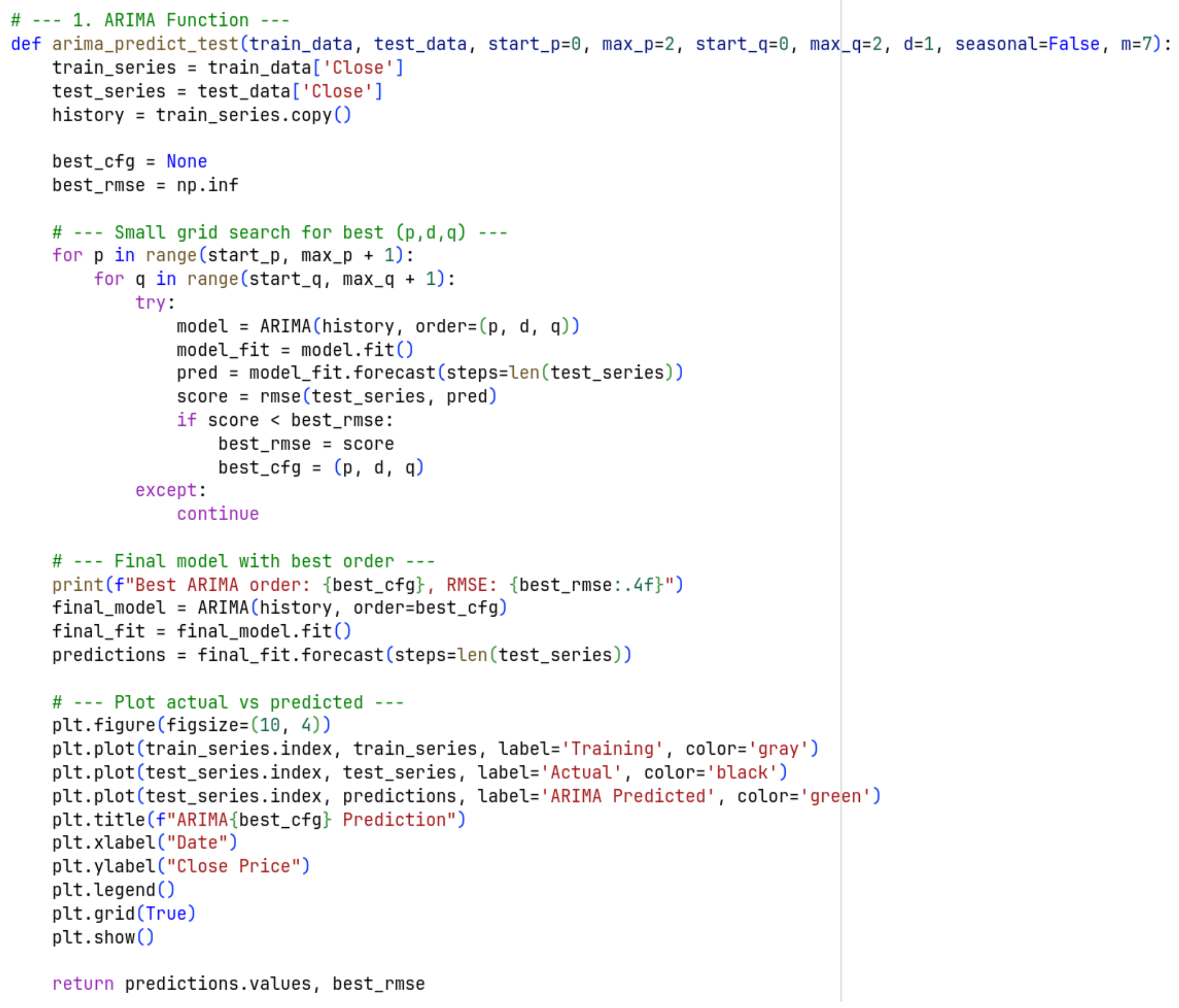
Statistical Models: ARIMA and SARIMAX

The ARIMA and SARIMAX models were set up using statsmodels for time series forecasting. Hyper-parameters were optimised via small grid search. For SARIMAX, Volume was incorporated as an exogenous feature to enrich the regression.

- ARIMA: parameters p, d, q were swept for best RMSE on validation
- SARIMAX: seasonal effects with relevant (P, D, Q, S)

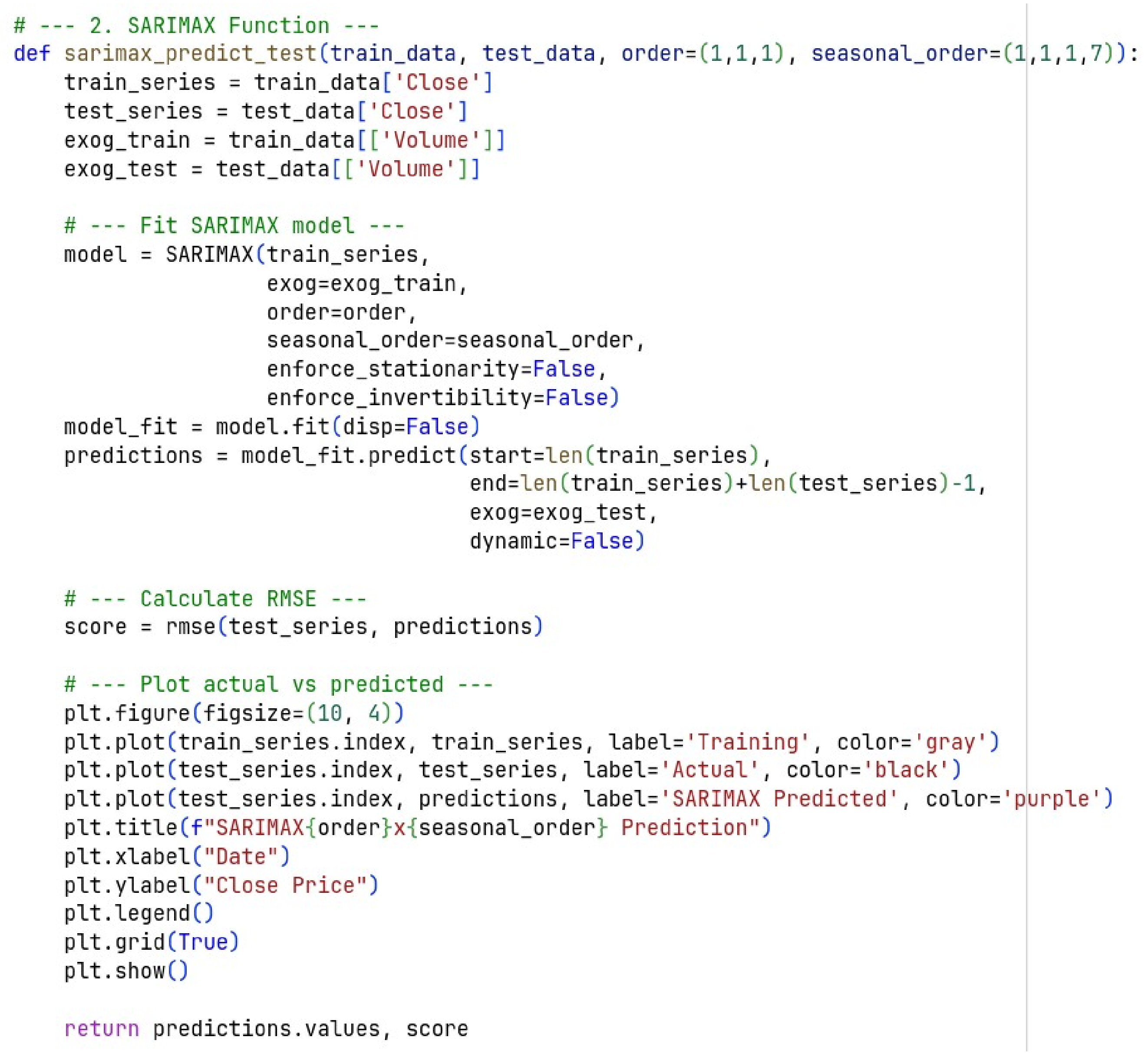
Function: ***arima_predict_test()***

Performs grid search over p and q parameters to find optimal ARIMA configuration. It returns predictions and RMSE for the best-performing model which includes visualisations of training, actual, and predicted values.



```python
# --- 1. ARIMA Function ---
def arima_predict_test(train_data, test_data, start_p=0, max_p=2, start_q=0, max_q=2, d=1, seasonal=False, m=7):
    train_series = train_data['Close']
    test_series = test_data['Close']
    history = train_series.copy()

    best_cfg = None
    best_rmse = np.inf

    # --- Small grid search for best (p,d,q) ---
    for p in range(start_p, max_p + 1):
        for q in range(start_q, max_q + 1):
            try:
                model = ARIMA(history, order=(p, d, q))
                model_fit = model.fit()
                pred = model_fit.forecast(steps=len(test_series))
                score = rmse(test_series, pred)
                if score < best_rmse:
                    best_rmse = score
                    best_cfg = (p, d, q)
            except:
                continue

    # --- Final model with best order ---
    print(f"Best ARIMA order: {best_cfg}, RMSE: {best_rmse:.4f}")
    final_model = ARIMA(history, order=best_cfg)
    final_fit = final_model.fit()
    predictions = final_fit.forecast(steps=len(test_series))

    # --- Plot actual vs predicted ---
    plt.figure(figsize=(10, 4))
    plt.plot(train_series.index, train_series, label='Training', color='gray')
    plt.plot(test_series.index, test_series, label='Actual', color='black')
    plt.plot(test_series.index, predictions, label='ARIMA Predicted', color='green')
    plt.title(f"ARIMA{best_cfg} Prediction")
    plt.xlabel("Date")
    plt.ylabel("Close Price")
    plt.legend()
    plt.grid(True)
    plt.show()

    return predictions.values, best_rmse
```

Function: *sarimax_predict_test()*
Fits Seasonal ARIMA with exogenous variables using predefined parameters and incorporates Volume as external regressor. It returns predictions and RMSE with comparative visualisations.

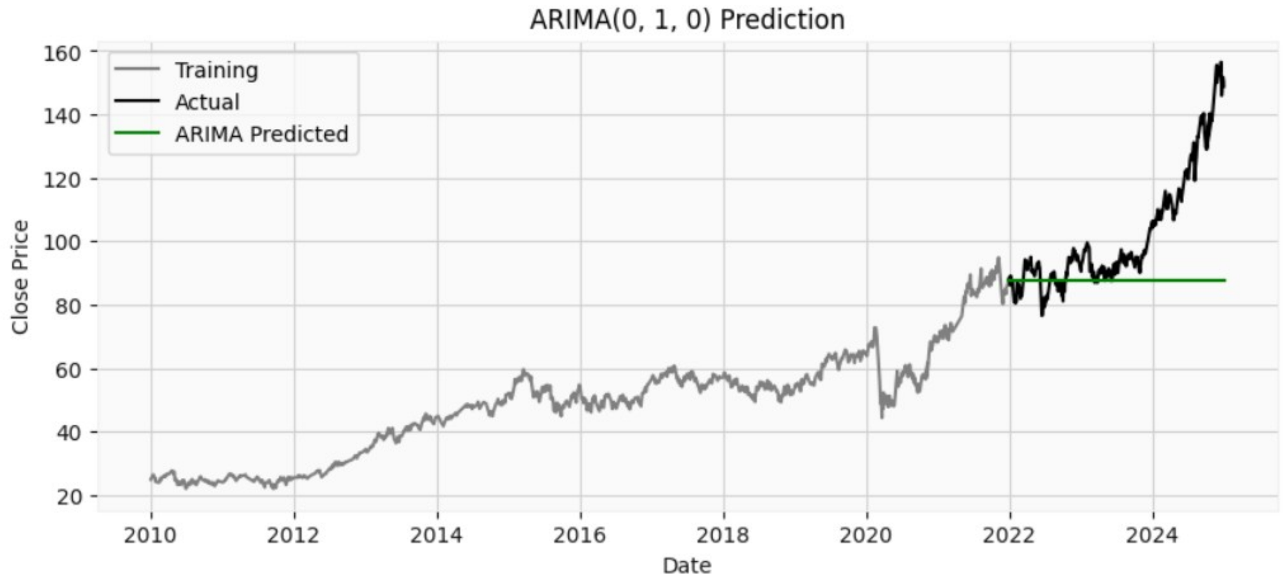


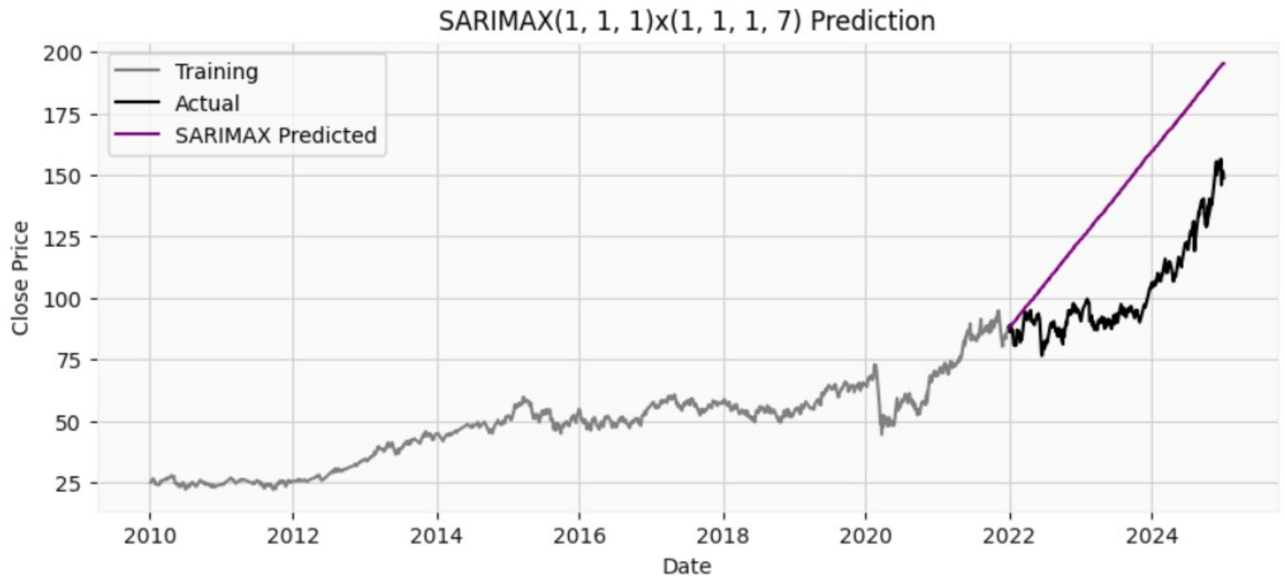
```python
# --- 2. SARIMAX Function ---
def sarimax_predict_test(train_data, test_data, order=(1,1,1), seasonal_order=(1,1,1,7)):
    train_series = train_data['Close']
    test_series = test_data['Close']
    exog_train = train_data[['Volume']]
    exog_test = test_data[['Volume']]

    # --- Fit SARIMAX model ---
    model = SARIMAX(train_series,
                    exog=exog_train,
                    order=order,
                    seasonal_order=seasonal_order,
                    enforce_stationarity=False,
                    enforce_invertibility=False)
    model_fit = model.fit(disp=False)
    predictions = model_fit.predict(start=len(train_series),
                                    end=len(train_series)+len(test_series)-1,
                                    exog=exog_test,
                                    dynamic=False)

    # --- Calculate RMSE ---
    score = rmse(test_series, predictions)

    # --- Plot actual vs predicted ---
    plt.figure(figsize=(10, 4))
    plt.plot(train_series.index, train_series, label='Training', color='gray')
    plt.plot(test_series.index, test_series, label='Actual', color='black')
    plt.plot(test_series.index, predictions, label='SARIMAX Predicted', color='purple')
    plt.title(f"SARIMAX{order}x{seasonal_order} Prediction")
    plt.xlabel("Date")
    plt.ylabel("Close Price")
    plt.legend()
    plt.grid(True)
    plt.show()

    return predictions.values, score
```

Configurations

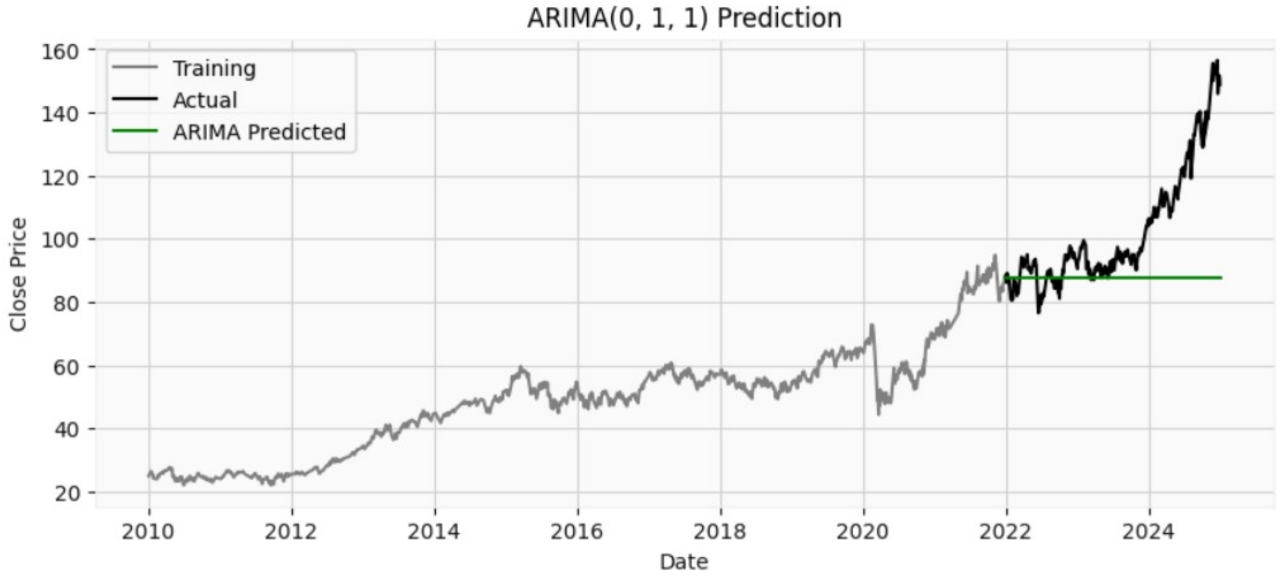
The code tests multiple parameter configurations for both models, comparing performance via RMSE. ARIMA uses an automated grid search while SARIMAX employs fixed seasonal patters with exogenous market data. All configurations are evaluated and visualised to identify the most effective time series forecasting approach.

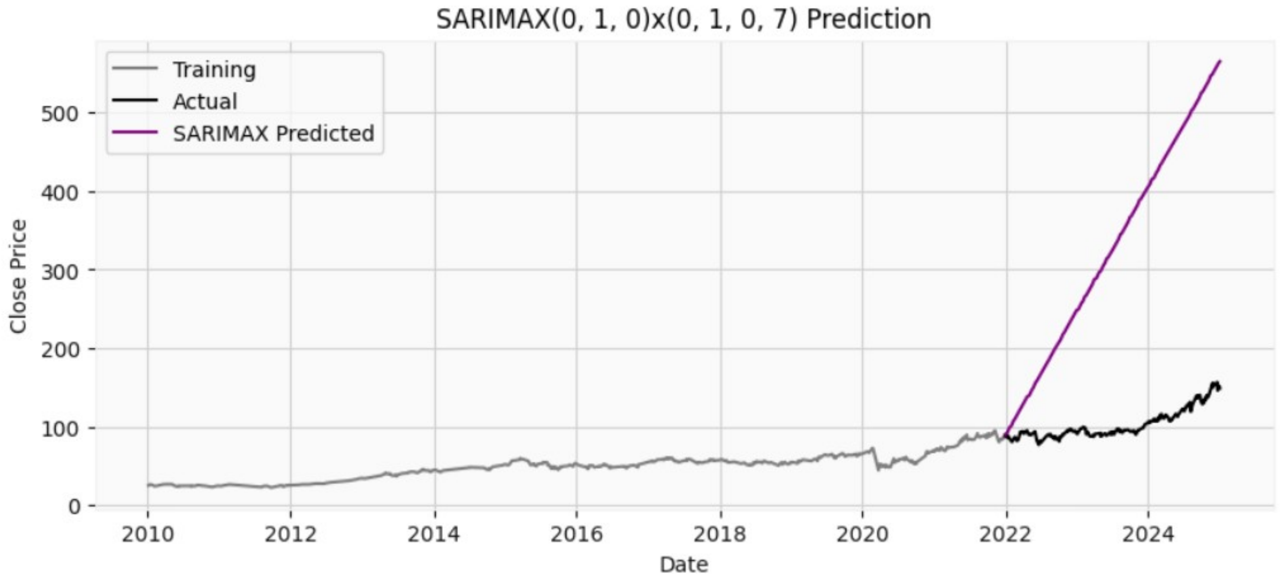
```
Running ARIMA/SARIMAX with multiple configurations...
Best ARIMA order: (0, 1, 0), RMSE: 23.6500
```



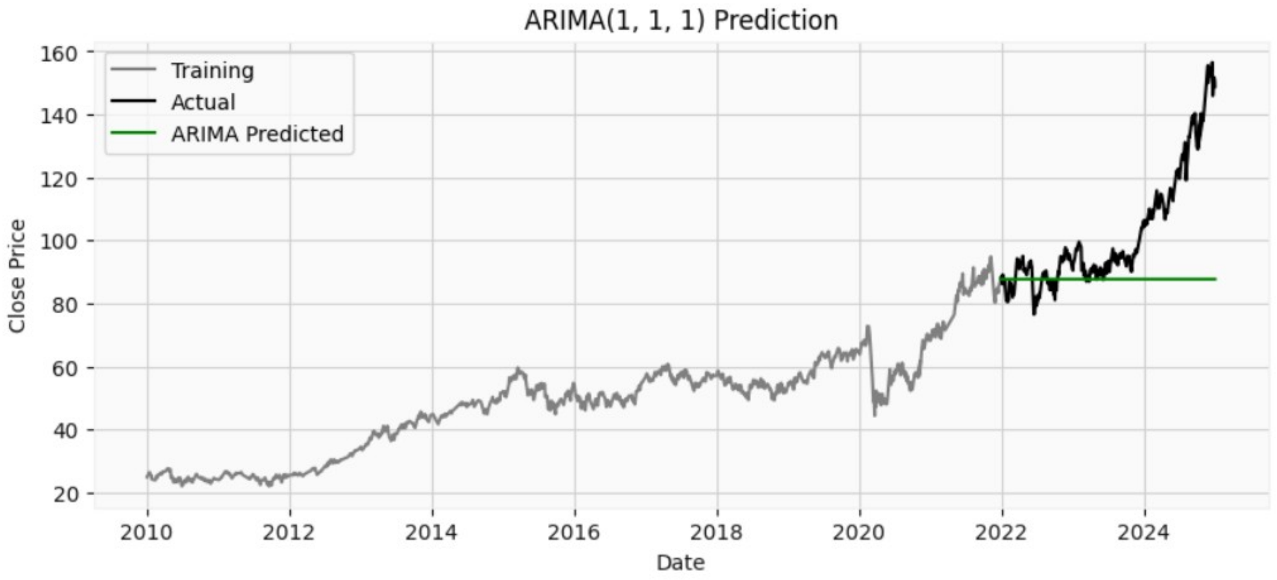


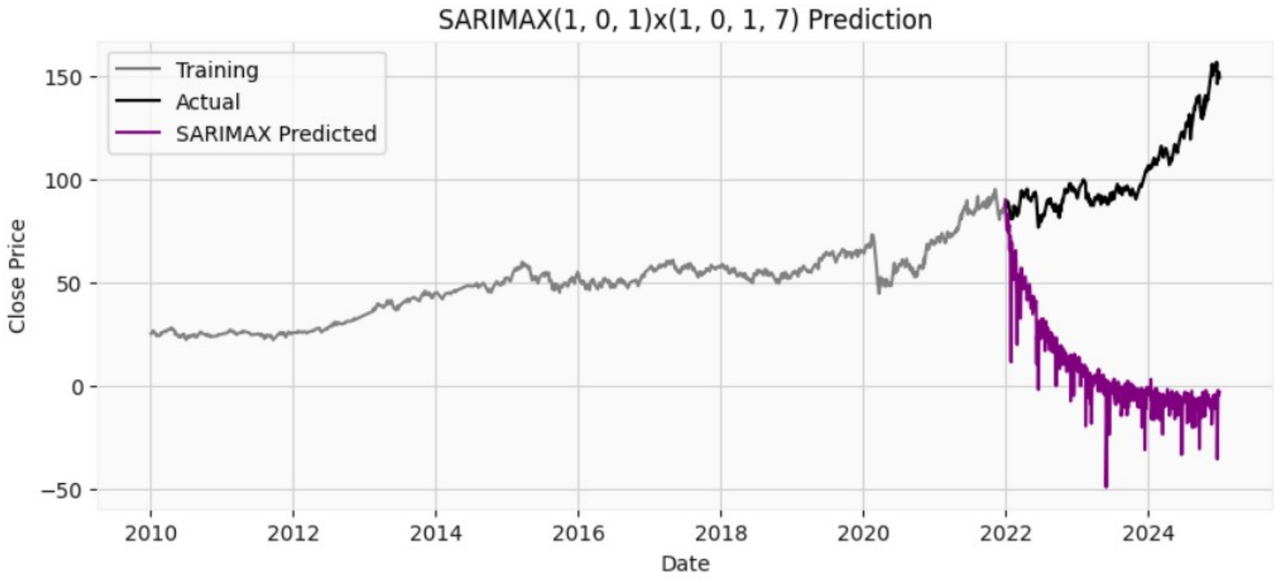
```
Testing Config ARIMA(0, 1, 1) / SARIMAX((0, 1, 0), (0, 1, 0, 7))
Best ARIMA order: (0, 1, 1), RMSE: 23.6920
```



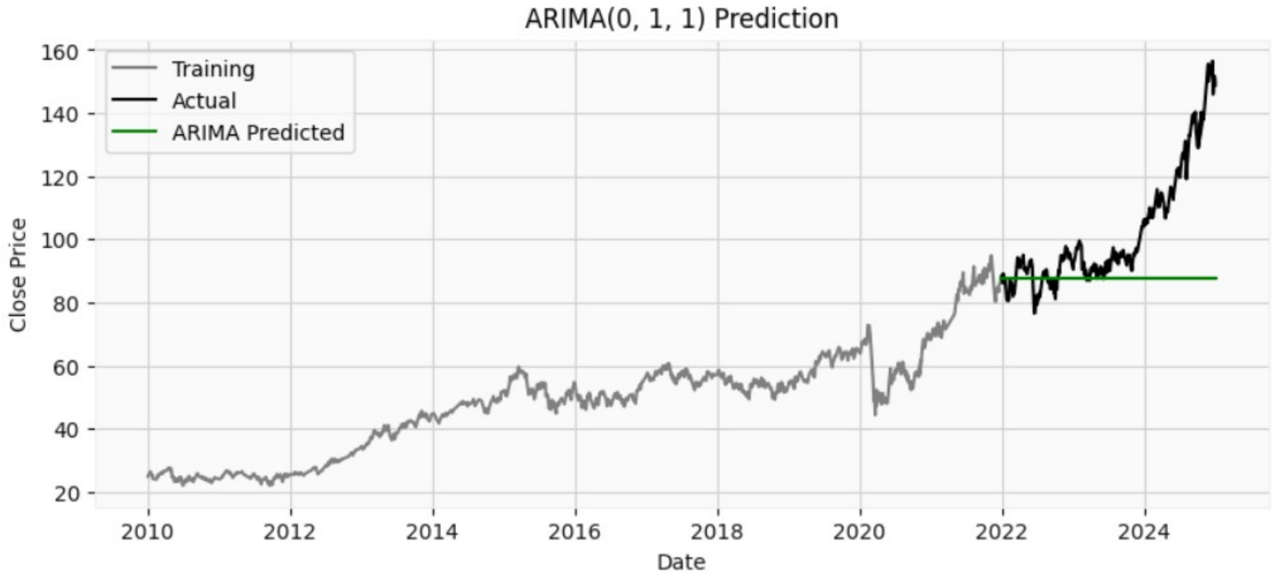


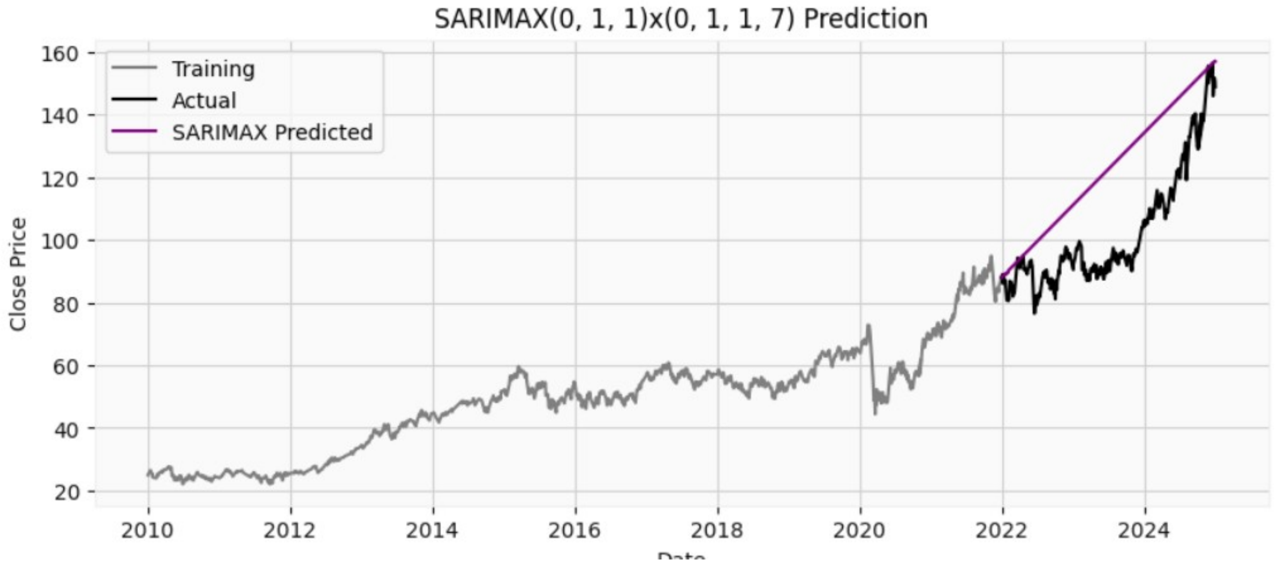
```
Testing Config ARIMA(1, 0, 1) / SARIMAX((1, 0, 1), (1, 0, 1, 7))
Best ARIMA order: (1, 1, 1), RMSE: 23.6876
```





```
Testing Config ARIMA(0, 1, 1) / SARIMAX((0, 1, 1), (0, 1, 1, 7))
Best ARIMA order: (0, 1, 1), RMSE: 23.6920
```





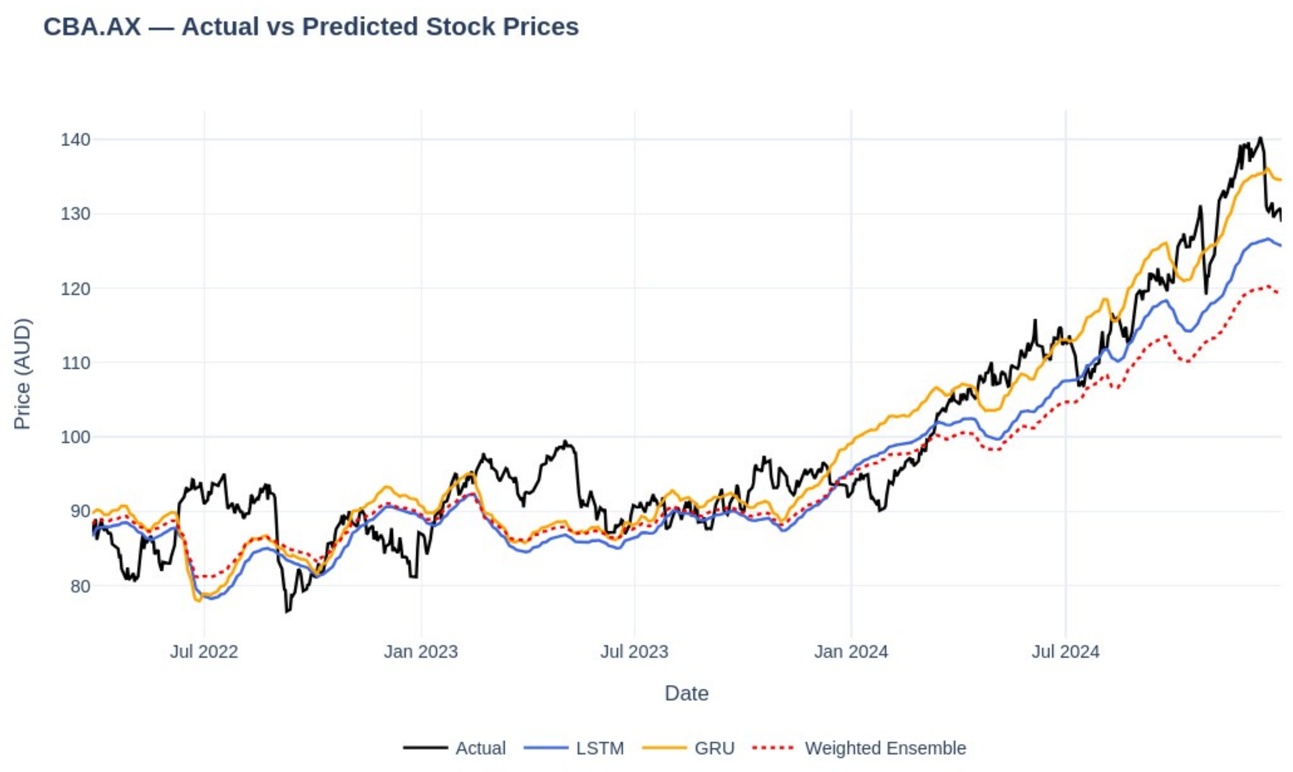
Results

Accuracy was measured using the RMSE between predicted and true test set Close values for each model. Additionally, ensemble predictions were assessed both by simple mean and by weighted mean, in which each model's RMSE determined its influence on the aggregate forecast.

The results support the hypothesis that combining model forecasts through average or weighted ensemble methods leads to improved predictive performance, especially when models are diverse in type and training. The GRU consistently performed best among the neural models for this time series, perhaps due to its architecture being less susceptible to overfitting in shorter sequence lengths. ARIMA and SARIMAX, while providing interpretable results, were less accurate than deep learning alternatives for this particular dataset.

Overall, the ARIMA predictions results in similar results whereas the SARIMAX was more accurate on some configurations.

I had also included visualisations using Plotly and Matplotlib to compare predictions versus actual prices across time for each model and for the ensemble:


CBA.AX — Actual vs Predicted Stock Prices
Price (AUD)
Jul 2022
Jan 2023
Jul 2023
Jan 2024
Jul 2024
Date
Actual
LSTM
GRU
Weighted Ensemble