COS30018
Intelligent Systems
Option B: Stock Prediction

Task B.7 – Machine Learning 4

Name: Duc Thuan Tran
Student ID: 104330455
Tutor: Dr. Ru Jia
Tutorial: Friday 2:30 – 4:30

Table of Contents

Introduction

The code base has been updated to version v0.6, with two key components: main_B7.py and data_processing_B7.py. This version incorporates several enhancements aimed at improving stock price prediction accuracy. By expanding the scope beyond traditional stock prediction techniques that rely solely on historical price data, the update addresses the influence of broader economic factors. Stock prices are affected by various indicators beyond just price trends, such as macroeconomic variables and market sentiment.
In this version, both files work together to integrate these new data points. The data_processing_B7.py handles the preprocessing of stock and macroeconomic data, while the main_B7.py coordinates model training using advanced decision-tree-based models like XGBoost and Random Forest. These models, which do not require epoch-based training, enhance the reliability of stock price predictions. This approach results in a more robust and versatile prediction tool, capable of better analyzing stock market trends for future enhancements.

Research on Potential Approaches

Several approaches were researched to extend and improve the stock price prediction model. These methods aim to enhance the accuracy and scope of the predictions by incorporating additional features, using advanced modeling techniques, and integrating external data sources. The potential approaches researched include:

1. Enhancing the Ensemble Model

To improve prediction accuracy, ensemble models that combine different algorithms were explored. Two key techniques were considered:

- **Stacking Ensemble**: This involves combining predictions from multiple models (e.g., ARIMA, SARIMA, LSTM/GRU, Random Forest) and using a meta-learner (e.g., Linear Regression or XGBoost) to determine the optimal way to blend these predictions.
- **Bagging/Boosting**: Techniques such as Bagging (Bootstrap Aggregation) and Boosting (like Gradient Boosting) were considered to further improve the performance of individual models by reducing variance and bias.

2. Multivariate Time Series Data Integration

The inclusion of multivariate data was explored as a way to capture more market dynamics:

- **Adding Macroeconomic Variables**: In addition to stock-specific data (e.g., Open, High, Low, Close, Volume), economic indicators such as inflation, GDP, and unemployment rates were considered for integration. This would provide a broader context for market trends.
- **Technical Indicators**: Key market indicators like **RSI (Relative Strength Index)**, **MACD (Moving Average Convergence Divergence)**, and **Bollinger Bands** were researched as a means of incorporating technical analysis into the model. These indicators help detect market momentum and reversals.
- **Multistep Forecasting**: A strategy to predict multiple days ahead (multistep forecasting) was examined, using both historical stock data and the newly integrated multivariate data.

3. News and Sentiment Data Integration

Market sentiment from news and social media can strongly influence stock prices. The following approaches were explored:

- **News Data Integration**: Collecting data from financial news APIs (e.g., Yahoo Finance, Reuters, Twitter) to track real-time events that may affect market sentiment.
- **Sentiment Analysis**: Natural Language Processing (NLP) techniques were considered to analyze the tone and sentiment of news articles, social media posts, and other textual data. This sentiment could be factored into the stock price prediction model to better capture market reactions.

4. Model Optimization and Training Strategies

Various optimization techniques were researched to improve model training and performance:

- **Hyperparameter Tuning**: Methods such as Grid Search or Random Search were considered for automatically adjusting model hyperparameters to find the best configuration.
- **Early Stopping and Learning Rate Scheduling**: These techniques were examined to avoid overfitting and to optimize the model's learning rate during training.
- **K-fold Cross Validation**: K-fold cross-validation was researched as a method to better evaluate the model's performance by splitting the dataset into multiple training/testing folds, ensuring more reliable results than a single train/test split.

5. Enhanced Visualization and Dashboards

Visualization improvements were explored to better present prediction results:

- **Real-time Visualization**: Integrating real-time visualization for predicted and actual stock prices using tools like Dash or Streamlit, which would allow users to interactively explore predictions by entering stock symbols.
- **Advanced Chart Types**: Expanding beyond line charts to include moving averages, candlestick charts, and heatmaps to visually represent the correlations between various features in the dataset.

6. Multi-company Prediction Expansion

The model currently focuses on predicting stock prices for a single company (CBA.AX). Expanding the model to predict for multiple companies was researched:

- **Generalized Model for Multiple Companies**: Building a generalized model that can handle predictions for different stock symbols by adding a feature that represents the stock ticker.
- **Real-time Predictions for Multiple Companies**: Developing a system for real-time prediction of stock prices for multiple companies, allowing traders to make faster, informed decisions.

7. Advanced Model Applications

Finally, more sophisticated machine learning models were considered:

- **Transformers for Time Series**: Transformer models (like BERT) were researched for their potential to handle time series data more effectively, due to their ability to process sequential data.
- **Reinforcement Learning**: Reinforcement learning was explored as a method to optimize trading strategies based on stock price predictions. This approach could help develop automated trading systems that adjust to market conditions.

Potential Approaches Chosen for Implementation

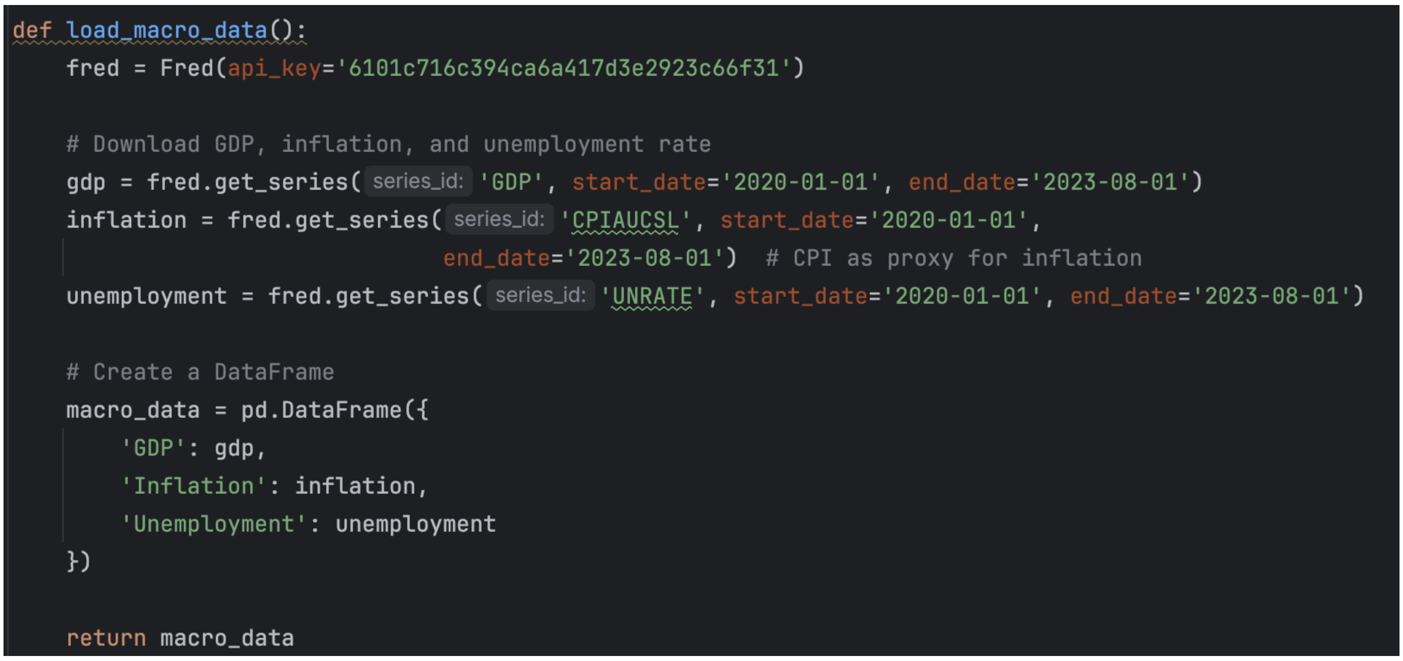
The approach chosen for implementation in this project is **Multivariate Time Series Data Integration**. This approach was selected because of its potential to significantly enhance the predictive capabilities of the model by incorporating additional variables beyond traditional stock price data. The goal was to provide the model with more context about market trends and economic conditions, leading to more accurate and reliable predictions.

Development of Multivariate Time Series Data Integration

In this implementation, multiple enhancements were made to the model, focusing on the integration of both technical and macroeconomic indicators:

1. Macroeconomic Variables

- Integration: Data on macroeconomic factors such as GDP, inflation rate, and unemployment was integrated into the stock price dataset using the Fred API. These variables provide critical economic context that influences stock market behavior and, by extension, stock prices.
- Resampling: The macroeconomic data was resampled to a daily frequency to ensure it aligns with the stock price data. Missing values were handled using forward-filling (ffill), ensuring that all data points remained continuous for training purposes.
- Purpose: Incorporating these macroeconomic variables allows the model to factor in external economic conditions that might affect stock price movements, leading to a more informed and accurate prediction.

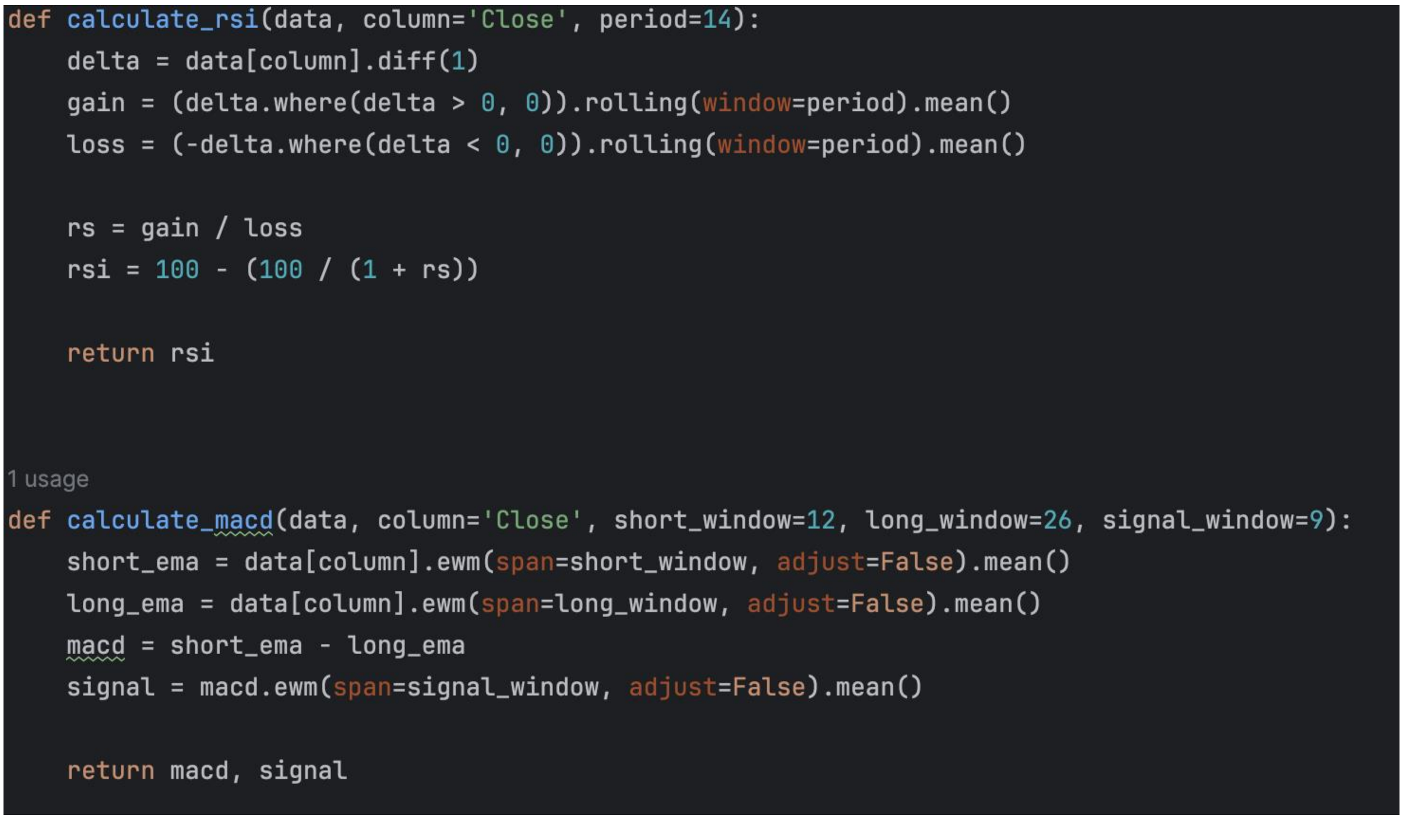


```python
def load_macro_data():
    fred = Fred(api_key='6101c716c394ca6a417d3e2923c66f31')

    # Download GDP, inflation, and unemployment rate
    gdp = fred.get_series( series_id: 'GDP', start_date='2020-01-01', end_date='2023-08-01')
    inflation = fred.get_series( series_id: 'CPIAUCSL', start_date='2020-01-01',
                                 end_date='2023-08-01')  # CPI as proxy for inflation
    unemployment = fred.get_series( series_id: 'UNRATE', start_date='2020-01-01', end_date='2023-08-01')

    # Create a DataFrame
    macro_data = pd.DataFrame({
        'GDP': gdp,
        'Inflation': inflation,
        'Unemployment': unemployment
    })

    return macro_data
```

Image 1. Load_marco_data

2. **Technical Indicators**

- RSI and MACD Calculation: Two popular technical indicators were added to the dataset:
    - RSI (Relative Strength Index): Helps assess whether a stock is overbought or oversold, signaling potential price reversals.
    - MACD (Moving Average Convergence Divergence): Tracks the momentum of stock prices and provides buy/sell signals based on trend reversals.
- Why important: RSI and MACD offer valuable insights into market momentum and price trends, which are not captured by raw stock price data alone. These indicators help the model understand market conditions better and predict future movements more accurately.

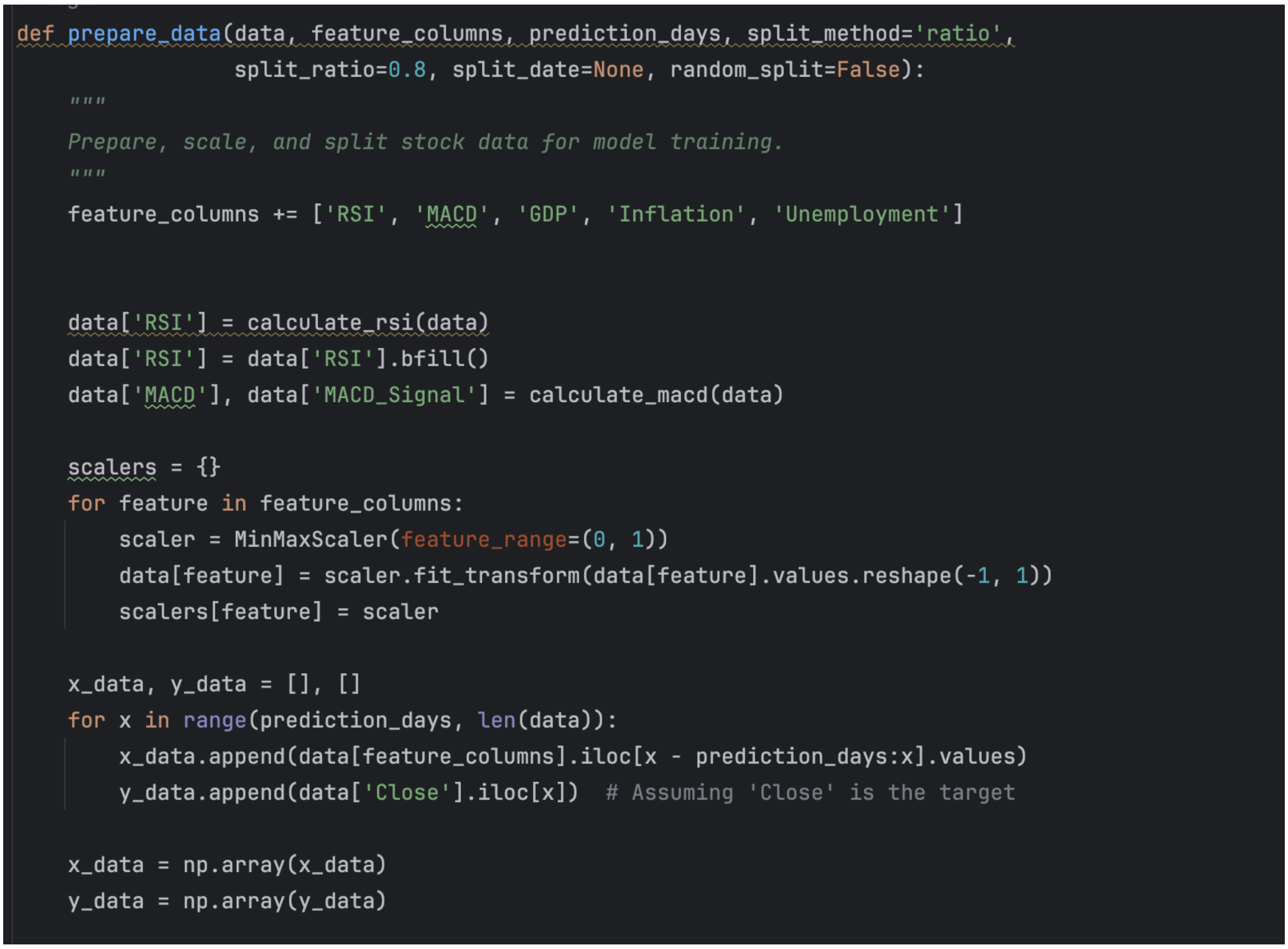


```python
def calculate_rsi(data, column='Close', period=14):
    delta = data[column].diff(1)
    gain = (delta.where(delta > 0, 0)).rolling(window=period).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=period).mean()

    rs = gain / loss
    rsi = 100 - (100 / (1 + rs))

    return rsi


1 usage
def calculate_macd(data, column='Close', short_window=12, long_window=26, signal_window=9):
    short_ema = data[column].ewm(span=short_window, adjust=False).mean()
    long_ema = data[column].ewm(span=long_window, adjust=False).mean()
    macd = short_ema - long_ema
    signal = macd.ewm(span=signal_window, adjust=False).mean()

    return macd, signal
```

Image 2. Caculate_rsi and caculate_masd function

3. **Data Preparation**

- Scaling and Normalization: After the integration of macroeconomic variables and technical indicators, the dataset was normalized using the MinMaxScaler to ensure that all variables were scaled to the same range. This step is essential for machine learning models as it ensures that no single feature dominates due to differences in magnitude.
- Train-Test Split: The dataset was split into training and testing sets based on a predefined ratio. This split allows the model to be trained on historical data and then evaluated on unseen data to ensure generalization.
- Why it matters: Proper data preparation ensures that the model can effectively learn patterns in the data and is robust enough to make accurate predictions on future data.

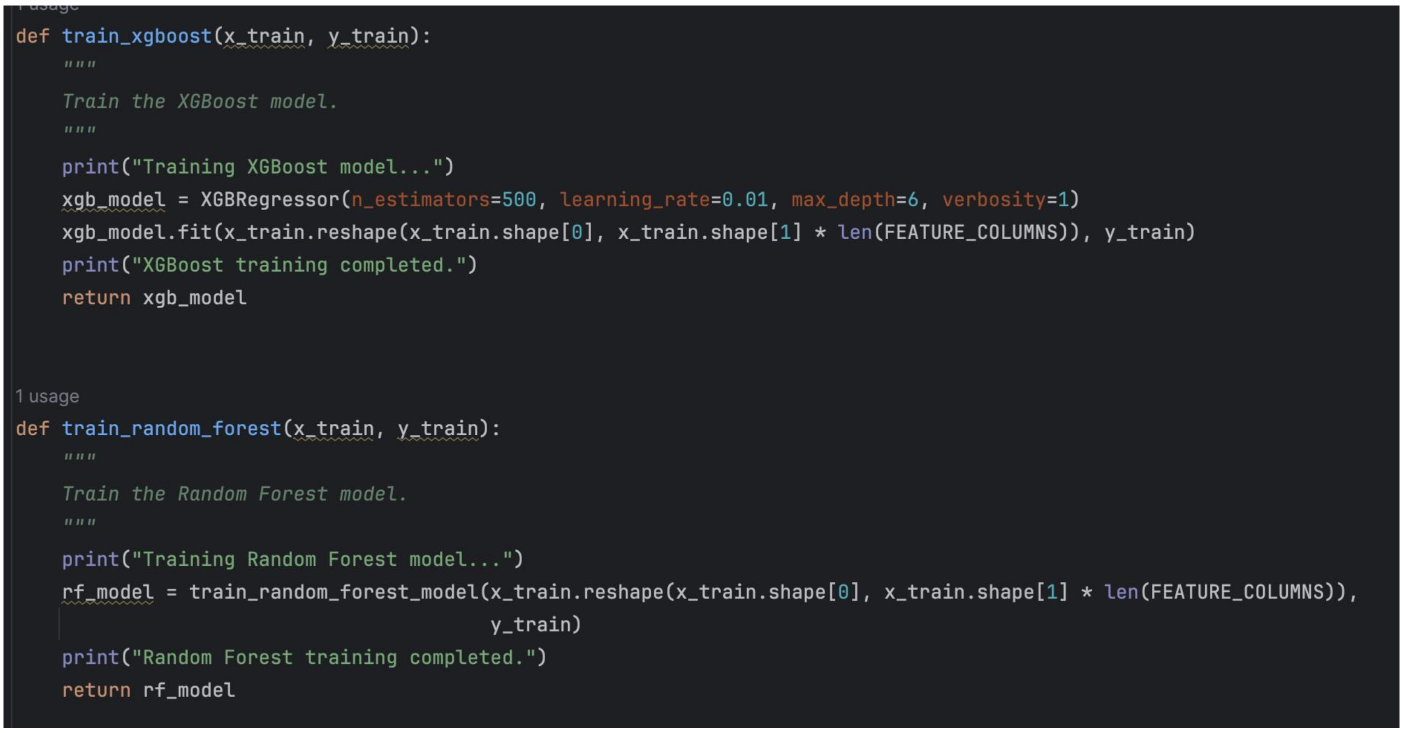


```python
def prepare_data(data, feature_columns, prediction_days, split_method='ratio',
                 split_ratio=0.8, split_date=None, random_split=False):
    """
    Prepare, scale, and split stock data for model training.
    """
    feature_columns += ['RSI', 'MACD', 'GDP', 'Inflation', 'Unemployment']


    data['RSI'] = calculate_rsi(data)
    data['RSI'] = data['RSI'].bfill()
    data['MACD'], data['MACD_Signal'] = calculate_macd(data)

    scalers = {}
    for feature in feature_columns:
        scaler = MinMaxScaler(feature_range=(0, 1))
        data[feature] = scaler.fit_transform(data[feature].values.reshape(-1, 1))
        scalers[feature] = scaler

    x_data, y_data = [], []
    for x in range(prediction_days, len(data)):
        x_data.append(data[feature_columns].iloc[x - prediction_days:x].values)
        y_data.append(data['Close'].iloc[x])  # Assuming 'Close' is the target

    x_data = np.array(x_data)
    y_data = np.array(y_data)
```

Image 3. Prepare_data function (developed)

4. **Model Integration**

- Machine Learning Models: Two models were selected for integration—XGBoost and Random Forest. These models were trained on the enriched dataset, which included not only stock price data but also the newly added macroeconomic and technical indicators.
- Why this is important: By training the models on a more comprehensive dataset, they were able to account for both internal and external factors that influence stock prices. The inclusion of technical indicators provided the models with insights into price momentum and market trends, while macroeconomic data introduced the broader economic context.
- Evaluation: The models were evaluated based on their predictive accuracy on unseen test data, with the goal of improving the accuracy of stock price predictions.



```
def train_xgboost(x_train, y_train):
    """
    Train the XGBoost model.
    """
    print("Training XGBoost model...")
    xgb_model = XGBRegressor(n_estimators=500, learning_rate=0.01, max_depth=6, verbosity=1)
    xgb_model.fit(x_train.reshape(x_train.shape[0], x_train.shape[1] * len(FEATURE_COLUMNS)), y_train)
    print("XGBoost training completed.")
    return xgb_model


1 usage
def train_random_forest(x_train, y_train):
    """
    Train the Random Forest model.
    """
    print("Training Random Forest model...")
    rf_model = train_random_forest_model(x_train.reshape(x_train.shape[0], x_train.shape[1] * len(FEATURE_COLUMNS)),
                                         y_train)
    print("Random Forest training completed.")
    return rf_model
```

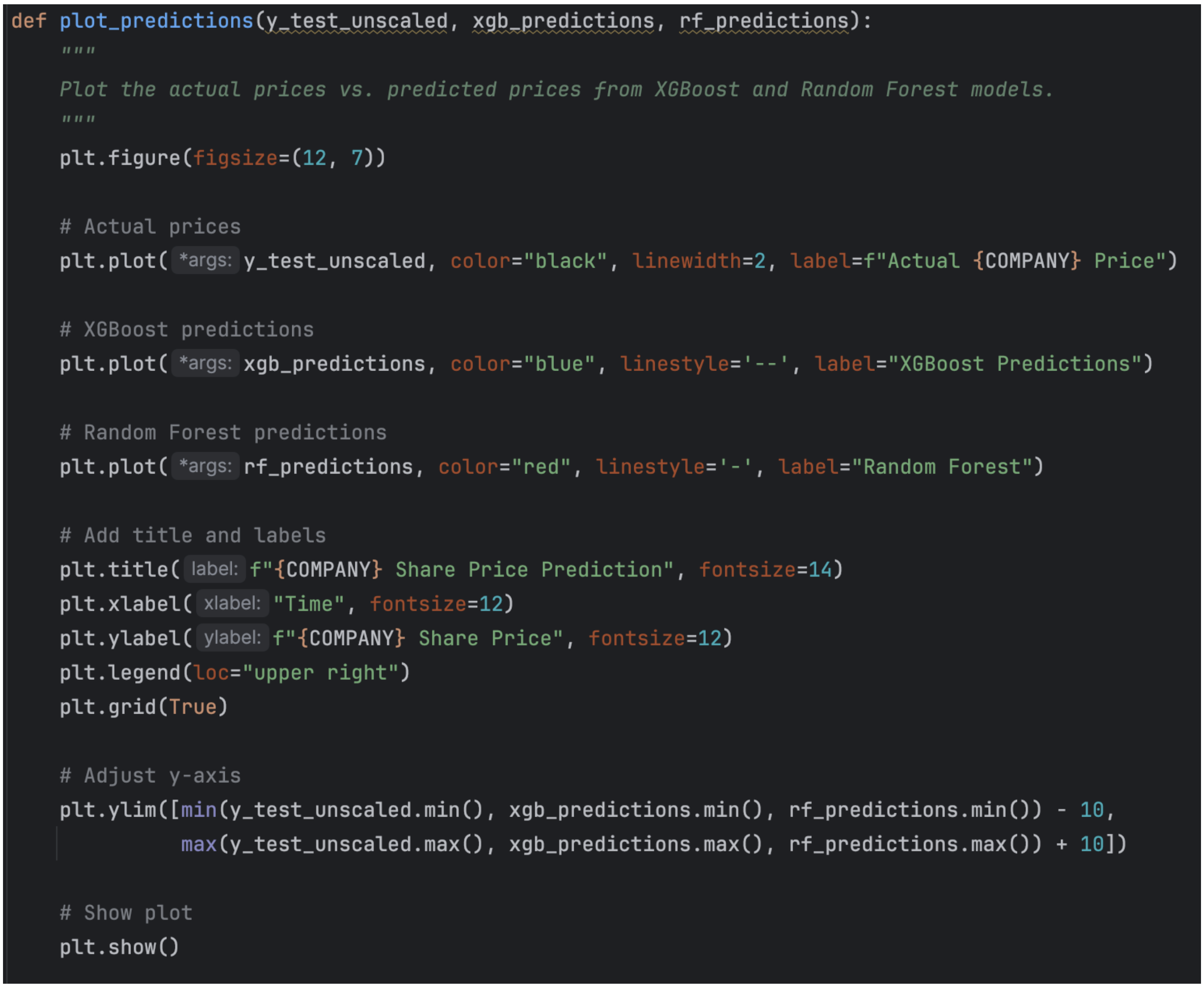
Image 4. train_xgboost and train_random_forest function

Data Visualization

1. Stock Price and Prediction Plot

The stock price and predictions generated by **XGBoost** and **Random Forest** models were visualized in a **line chart**. This allowed seeing how closely the models' predictions matched the actual prices over time.

- **Purpose**: This visualization makes it easy to compare how the models are performing by directly contrasting their predictions with actual stock prices.
- **Code Explanation**:
  - The actual stock prices are displayed in **black**.
  - The **XGBoost** predictions are shown with a **dashed blue line**.
  - The **Random Forest** predictions are shown with a **solid red line**.



```python
def plot_predictions(y_test_unscaled, xgb_predictions, rf_predictions):
    """
    Plot the actual prices vs. predicted prices from XGBoost and Random Forest models.
    """
    plt.figure(figsize=(12, 7))

    # Actual prices
    plt.plot( *args: y_test_unscaled, color="black", linewidth=2, label=f"Actual {COMPANY} Price")

    # XGBoost predictions
    plt.plot( *args: xgb_predictions, color="blue", linestyle='--', label="XGBoost Predictions")

    # Random Forest predictions
    plt.plot( *args: rf_predictions, color="red", linestyle='-', label="Random Forest")

    # Add title and labels
    plt.title( label: f"{COMPANY} Share Price Prediction", fontsize=14)
    plt.xlabel( xlabel: "Time", fontsize=12)
    plt.ylabel( ylabel: f"{COMPANY} Share Price", fontsize=12)
    plt.legend(loc="upper right")
    plt.grid(True)

    # Adjust y-axis
    plt.ylim([min(y_test_unscaled.min(), xgb_predictions.min(), rf_predictions.min()) - 10,
              max(y_test_unscaled.max(), xgb_predictions.max(), rf_predictions.max()) + 10])

    # Show plot
    plt.show()
```

Image 5. Plot_prediction function

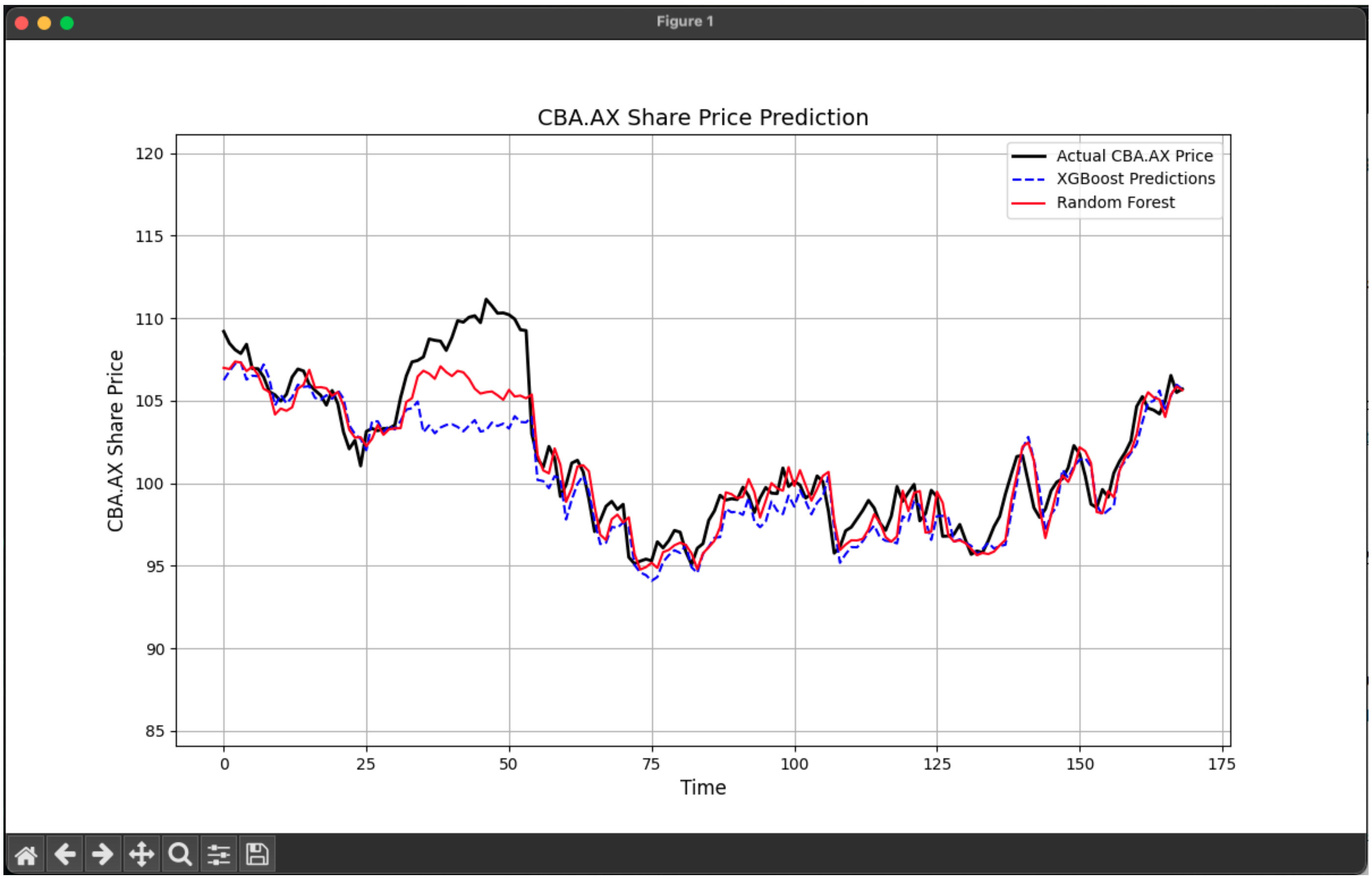
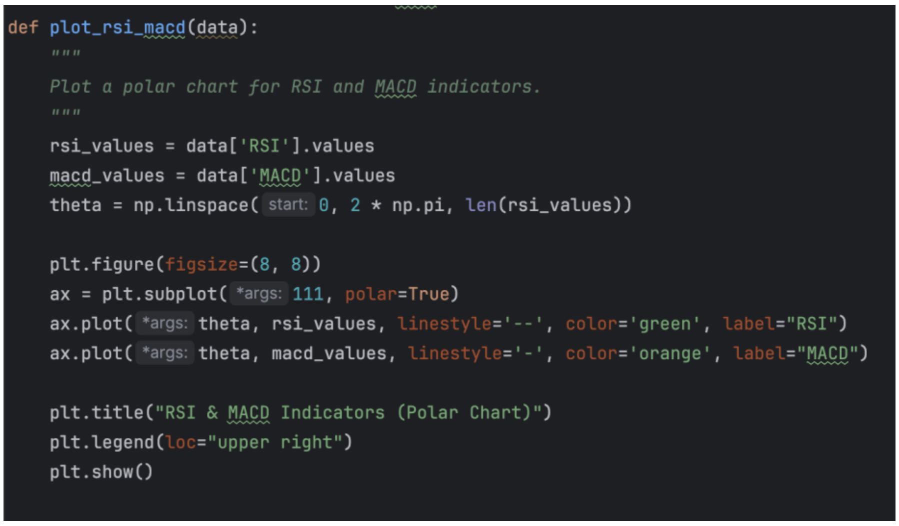


Image 6. Actual price, Xgboost prediction, randomforest plot

2. RSI and MACD Indicators Visualization
To further enhance the analysis, visualizing the **RSI** and **MACD** indicators in a **polar chart** and **area chart**. These technical indicators provide insights into market momentum, which is useful for understanding trends and potential reversals in stock prices.

RSI and MACD Polar Chart

- **Purpose**: The polar chart provides a unique visualization to represent the cyclical nature of the stock market's momentum. It allows for an intuitive representation of how these two indicators fluctuate over time in relation to each other.
- **Code Explanation**:
    - **RSI** values are plotted as a **green line** in a polar format.
    - **MACD** values are plotted as an **orange line**, helping to compare the two indicators.



```
def plot_rsi_macd(data):
    """
    Plot a polar chart for RSI and MACD indicators.
    """
    rsi_values = data['RSI'].values
    macd_values = data['MACD'].values
    theta = np.linspace( start: 0, 2 * np.pi, len(rsi_values))

    plt.figure(figsize=(8, 8))
    ax = plt.subplot( *args: 111, polar=True)
    ax.plot( *args: theta, rsi_values, linestyle='--', color='green', label="RSI")
    ax.plot( *args: theta, macd_values, linestyle='-', color='orange', label="MACD")

    plt.title("RSI & MACD Indicators (Polar Chart)")
    plt.legend(loc="upper right")
    plt.show()
```

Image 7. Plot_rsi_macd function

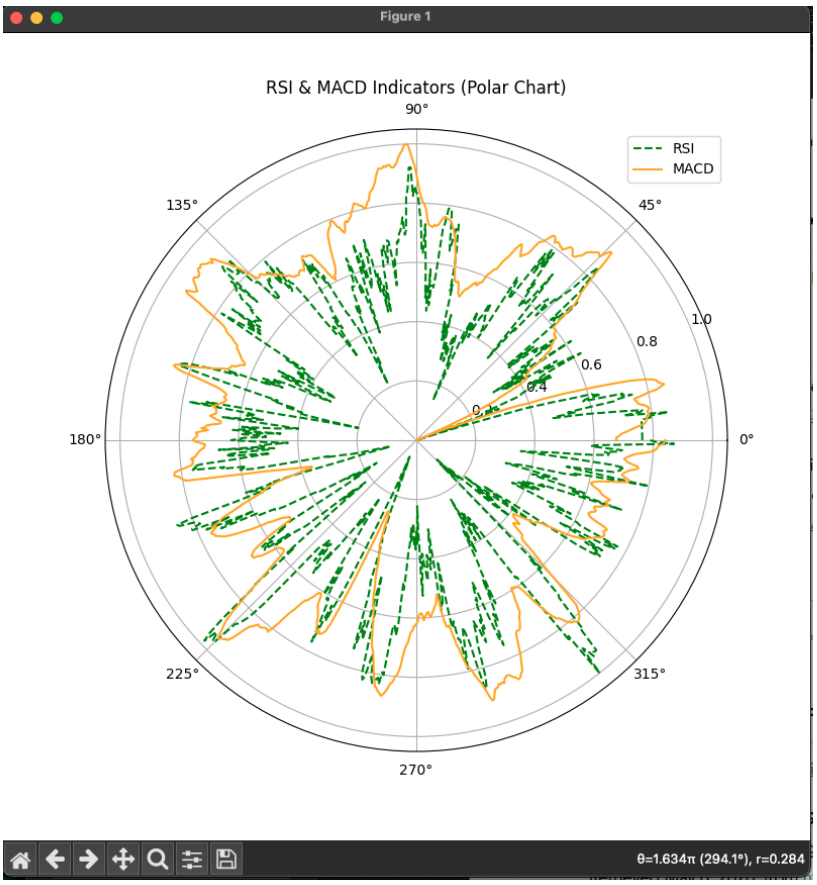


Image 8. RSI vs MACD Indicators plot

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. This repository contains all necessary files for the project and is available for the tutor to review the work.

References

https://github.com/jason887/Using-Deep-Learning-Neural-Networks-and-Candlestick-Chart-Representation-to-Predict-Stock-Market

https://www.datacamp.com/tutorial/xgboost-in-python
https://fredaccount.stlouisfed.org/apikey
https://www.sciencedirect.com/science/article/abs/pii/S0959804922002349