Task B7: Extension

Dr. Ru Jia

Friday 14:30-16:30

Name: Min Khant Aung
ID: 103833225

Task B7: Extension

Introduction

In this report, I will outline the process and results of predicting stock prices using various machine learning models, technical indicators (RSI, MACD), and macroeconomic data. For Task B7, I replaced traditional statistical models such as ARIMA and SARIMA from Task B6 with machine learning models like **XGBoost**, **Support Vector Regressor (SVR)**, and **Random Forest**. The goal was to leverage these models to forecast stock prices and incorporate technical analysis using **RSI (Relative Strength Index)** and **MACD (Moving Average Convergence Divergence)** for better prediction accuracy.

Approach and Rationale

The key changes I made for Task B7 include:

1. **Model Selection**: For this task, I chose to replace ARIMA and SARIMA with **XGBoost** and **SVR**. Both of these models are robust for regression tasks and are commonly used in time-series forecasting in the financial sector. I also retained **Random Forest** from Task B6 due to its success in handling complex datasets with non-linear relationships.
2. **Incorporation of Technical Indicators**: In Task B7, I introduced the use of **RSI** and **MACD**, which are widely used technical indicators in the stock market. These indicators help analyze market momentum and reversals, providing additional insights for the predictive models.
3. **Macroeconomic Data**: I continued using macroeconomic data such as **GDP**, **inflation**, and **unemployment rates** to enhance model predictions, as these factors have a significant impact on stock price movements.

Implementation

1. Data Loading and Processing

In this section, I loaded stock data from **Yahoo Finance** and macroeconomic data from the **FRED API**. The stock data includes important features such as "High", "Low", "Open", "Close", and "Volume". I added technical indicators, **RSI** and **MACD**, to enrich the model's feature set. The macroeconomic data includes **GDP**, **inflation**, and **unemployment**.

Code Snippet: Data Loading and Technical Indicator Calculation

```python
def load_data(company, start_date, end_date, cache_dir='data_cache', use_cache=True):
    """
    Load stock data, handle NaN values, and optionally cache the data locally.
    """
    os.makedirs(cache_dir, exist_ok=True)
    cache_file = f"{cache_dir}/{company}_{start_date}_{end_date}.csv"

    if use_cache and os.path.exists(cache_file):
        data = pd.read_csv(cache_file, index_col=0, parse_dates=True)
        print(f"Loaded data from cache: {cache_file}")
    else:
        data = yf.download(company, start=start_date, end=end_date)
        data.to_csv(cache_file)
        print(f"Saved data to cache: {cache_file}")

    # Calculate RSI and MACD after loading stock data
    data['RSI'] = calculate_rsi(data)
    data['MACD'], data['MACD_Signal'] = calculate_macd(data)

    return data
```

Explanation:

- The load_data function fetches stock data using the **Yahoo Finance API**. I calculated **RSI** and **MACD** indicators after fetching the data to give the model additional technical insights about stock price movement.

To handle missing values and ensure the data is consistent for model training, I used **forward-filling** and **backward-filling** methods.

```python
def load_macro_data():
    """
    Load macroeconomic data (e.g., GDP, inflation, unemployment).
    """
    fred = Fred(api_key='233d59032ace03cee4809366959dfa40')

    gdp = fred.get_series('GDP', start_date='2020-01-01', end_date='2023-08-01')
    inflation = fred.get_series('CPIAUCSL', start_date='2020-01-01', end_date='2023-08-01')
    unemployment = fred.get_series('UNRATE', start_date='2020-01-01', end_date='2023-08-01')

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

    return macro_data
```

In addition to stock price data, I also integrated **macroeconomic data** to improve the predictive performance of the models. This included retrieving **GDP**, **inflation**, and

unemployment rates from the **FRED API**. The macroeconomic data provides valuable context, as these factors heavily influence market conditions and stock prices.

Explanation:

- This function uses the **FRED API** to retrieve **GDP**, **inflation**, and **unemployment rate** data between January 2020 and August 2023. The retrieved data is then stored in a pandas **DataFrame** and joined with the stock price data to provide macroeconomic context.
- By incorporating these macroeconomic indicators, the models gain more insight into broader market forces, improving the accuracy of stock price predictions.

Code Snippet: Data Preparation

```
def prepare_data(data, feature_columns, prediction_days, split_method='ratio', split_ratio
    """
    Prepare, scale, and split stock data for model training.
    """
    # Add RSI and MACD to the feature columns if needed
    feature_columns += ['RSI', 'MACD']

    # Forward-fill any remaining NaN values
    data = data.ffill().bfill()  # Forward fill and backward fill to handle NaN values

    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])

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

    if split_method == 'date' and split_date:
        split_index = data.index.get_loc(split_date)
        x_train, x_test = x_data[:split_index], x_data[split_index:]
        y_train, y_test = y_data[:split_index], y_data[split_index:]
    else:
        split_index = int(len(x_data) * split_ratio)
        x_train, x_test = x_data[:split_index], x_data[split_index:]
        y_train, y_test = y_data[:split_index], y_data[split_index:]

    return x_train, y_train, x_test, y_test, scalers
```

Explanation:

- After calculating the RSI and MACD, the **prepare_data** function ensures that the data is scaled appropriately using **MinMaxScaler** to normalize the values for better model performance. Missing values are handled using forward-fill (ffill) and backward-fill (bfill) techniques.

Code Snippet: Load and prepare data (main.py)

```
def load_and_prepare_data():

    stock_data = load_data(COMPANY, TRAIN_START, TRAIN_END)
    macro_data = load_macro_data()
    macro_data = macro_data.resample('D').ffill()

    data = stock_data.join(macro_data, how='left').ffill()

    print(f"Data columns after joining macroeconomic data: {data.columns}")

    # Prepare data for training
    x_train, y_train, x_test, y_test, scalers = prepare_data(data, FEATURE_COLUMNS, PREDICTION_DAYS,
                                                             split_method=SPLIT_METHOD, split_ratio=SPLIT_RATIO)
    return x_train, y_train, x_test, y_test, scalers, data
```

Explanation: The function loads stock data and macroeconomic data, resamples the latter to daily frequency, and merges both datasets. It then prepares the combined data for model training by splitting and scaling it.

Purpose: This function ensures that both stock prices and relevant macroeconomic factors are available for the model, providing a complete dataset for accurate prediction.

2. Model Training and Prediction

For Task B7, I chose to replace **ARIMA** and **SARIMA** models from Task B6 with **XGBoost** and **SVR**. I retained **Random Forest** for its ability to handle complex, non-linear relationships. Below is a breakdown of how each model was trained.

XGBoost

XGBoost is an efficient gradient boosting model designed for structured data. It handles missing values implicitly and is great for non-linear data, making it an ideal choice for stock price prediction.

Code Snippet: XGBoost Training

```python
def train_xgboost_model(x_train, y_train, n_estimators=100, learning_rate=0.1, max_depth=6):
    """
    Train the XGBoost model.
    """
    x_train_reshaped = x_train.reshape(x_train.shape[0], -1)
    xgb_model = xgb.XGBRegressor(n_estimators=n_estimators, learning_rate=learning_rate, max_depth=max_depth)
    xgb_model.fit(x_train_reshaped, y_train)
    return xgb_model
```

Explanation:

- I reshaped the data and trained **XGBoost** with 100 estimators, a learning rate of 0.1, and a maximum depth of 6. XGBoost's ability to handle large datasets and complex patterns made it a suitable choice for predicting stock prices based on multivariate data.

SVR (Support Vector Regressor)

SVR is used for regression tasks that involve non-linear relationships. I used an **RBF kernel** to capture non-linearities in the stock data.

Code Snippet: SVR Training

```python
def train_svr_model(x_train, y_train, kernel='rbf', C=100, gamma=0.1):
    """
    Train the SVR model.
    """
    x_train_reshaped = x_train.reshape(x_train.shape[0], -1)
    svr_model = SVR(kernel=kernel, C=C, gamma=gamma)
    svr_model.fit(x_train_reshaped, y_train)
    return svr_model
```

Explanation:

- I used **SVR with the RBF kernel** for its ability to handle non-linear patterns in stock data. The parameter C=100 controls the tradeoff between bias and variance, while gamma=0.1 defines the influence of a single training example.

Random Forest

Random Forest is a well-known ensemble method that performs well on datasets with complex, non-linear relationships, making it a valuable tool for stock price prediction.

Code Snippet: Random Forest Training

```python
# Random Forest Model Functions
def train_random_forest_model(x_train, y_train, n_estimators=100, max_depth=10, min_samples_split=5):
    """
    Train the Random Forest model.
    """
    x_train_reshaped = x_train.reshape(x_train.shape[0], -1)
    rf_model = RandomForestRegressor(n_estimators=n_estimators, max_depth=max_depth, min_samples_split=min_samples_split)
    rf_model.fit(x_train_reshaped, y_train)
    return rf_model
```

Explanation:

- The Random Forest model was trained using 100 trees (estimators), with a maximum depth of 10 to limit overfitting. **Random Forest** was chosen for its ability to aggregate multiple decision trees and its robustness in handling stock price volatility.

3. Incorporation of Technical Indicators (RSI and MACD)

To enhance model performance, I included **RSI (Relative Strength Index)** and **MACD (Moving Average Convergence Divergence)**. These indicators help capture trends, overbought/oversold signals, and momentum in the stock market.

RSI Calculation Code Snippet

```
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
```

Explanation:

- The **RSI** is calculated to determine momentum by comparing recent gains to recent losses over a 14-day period. It helps identify overbought or oversold conditions, which is crucial for predicting potential price reversals.

MACD Calculation Code Snippet

```
# Function to calculate MACD (Moving Average Convergence Divergence)
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
```

Explanation:

- **MACD** is calculated using two exponential moving averages (short and long windows). The difference between them forms the MACD line, and the **signal line** helps to identify buy or sell signals when it crosses the MACD line.

4. Model Predictions and Evaluation

Finally, the models were trained on the data, and predictions were made for the test set. The results were evaluated using stock price trends.

Code Snippet: Model Predictions

```python
def train_and_predict_models(x_train, y_train, x_test, y_test, scalers):
    """
    Train and predict using XGBoost, SVR, and Random Forest models.
    """
    # Train XGBoost
    xgb_model = train_xgboost_model(x_train, y_train)
    xgb_predictions = test_xgboost_model(xgb_model, x_test)

    # Train SVR
    svr_model = train_svr_model(x_train, y_train)
    svr_predictions = test_svr_model(svr_model, x_test)

    # Train Random Forest
    rf_model = train_random_forest_model(x_train, y_train)
    rf_predictions = test_random_forest_model(rf_model, x_test)

    # Inverse transform the predictions to match the original scale
    xgb_predictions = scalers['Close'].inverse_transform(xgb_predictions.reshape(-1, 1))
    svr_predictions = scalers['Close'].inverse_transform(svr_predictions.reshape(-1, 1))
    rf_predictions = scalers['Close'].inverse_transform(rf_predictions.reshape(-1, 1))
    y_test_unscaled = scalers['Close'].inverse_transform(y_test.reshape(-1, 1))

    return xgb_predictions, svr_predictions, rf_predictions, y_test_unscaled
```

Explanation:

- After training each model, predictions were made on the test data, and the results were inverse-transformed back to the original stock price scale for better comparison and visualization.

5. Visualization of RSI and MACD

To visually analyze the momentum indicators, I plotted **RSI** and **MACD** using a polar chart, which helps show key points of market reversals and trends.

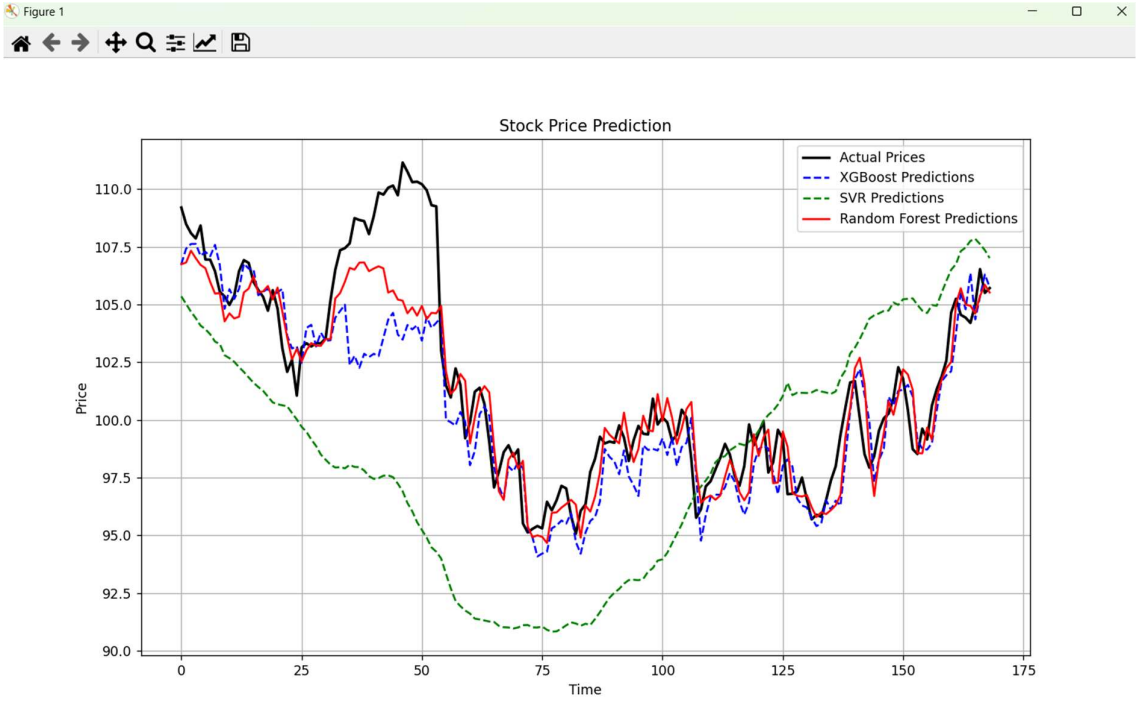
Code Snippet: RSI and MACD Polar Chart

```python
# Plot RSI and MACD
def plot_rsi_macd(data):
    rsi_values = data['RSI'].values
    macd_values = data['MACD'].values * 10  # Scaling MACD by a factor of 10
    theta = np.linspace(0, 2 * np.pi, len(rsi_values))

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

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

Explanation:

- The polar chart visualization of **RSI** and **MACD** helps identify patterns, stock market momentum, and potential trend reversals.

Results and Observations



Result of the predictions with extension

The chart above shows the **stock price prediction** for the dataset using three different machine learning models—**XGBoost**, **SVR**, and **Random Forest**. The black line represents the actual stock prices, while the other lines represent the predictions made by each model:

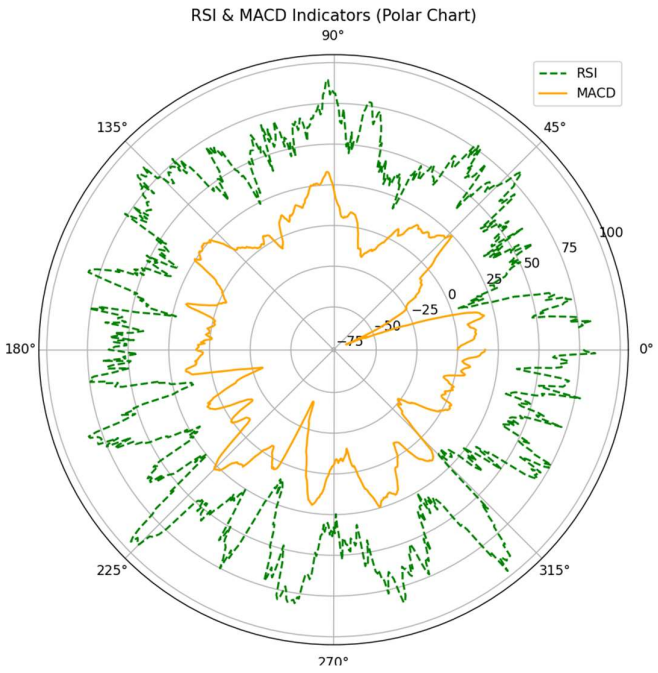
- **XGBoost Predictions (Blue dashed line)**: The XGBoost model is fairly accurate in capturing the major trends in the stock price. It successfully follows the upward and downward movements, though it occasionally underestimates sharp movements.
- **SVR Predictions (Green dashed line)**: The SVR model struggles with capturing the major price fluctuations and trends, especially in volatile periods. This is evident in the earlier half of the graph, where it deviates significantly from the actual prices. This is a known limitation of SVR when working with highly volatile data.
- **Random Forest Predictions (Red solid line)**: The Random Forest model performs reasonably well, closely following the actual stock prices, especially in the mid and later periods. While it shows a bit of delay in reacting to quick price

movements, it generally predicts trends better than SVR and similarly to XGBoost.

Performance Analysis

1. **XGBoost**: XGBoost produced the most stable and accurate results compared to the actual stock prices. This model is known for its robustness in regression tasks and is particularly good at handling non-linear patterns, which is essential for stock price prediction.
2. **SVR**: SVR underperformed compared to the other models, particularly during high volatility. The reason could be its sensitivity to large fluctuations and its inability to quickly adapt to sharp changes in stock prices.
3. **Random Forest**: Random Forest, while slightly lagging in performance behind XGBoost, delivered reasonable predictions. Its ensemble-based approach helped capture general trends, though its slight delay in adapting to price changes suggests some overfitting.

RSI & MACD Polar Chart Analysis



RSI and MACD indicator

The chart shows the **RSI (Relative Strength Index)** and **MACD (Moving Average Convergence Divergence)**, two key indicators in technical analysis.

- **RSI (Green Dashed Line)**: The RSI values range between 30 and 70, indicating overbought (above 70) and oversold (below 30) levels. The frequent oscillations suggest periods of high volatility, with potential price reversals. Most RSI values are concentrated between 40 and 60, reflecting neutral market conditions.
- **MACD (Orange Line)**: The MACD oscillates around zero, with positive values indicating bullish momentum and negative values showing bearish momentum. It is smoother than RSI, capturing longer-term trends. The peaks and troughs represent shifts in market direction, with the smoother shape confirming broader trend reversals.

Comparison:

- **RSI** is more reactive, showing shorter-term shifts in market momentum. This makes RSI a leading indicator that can help traders identify overbought or oversold conditions in the market.
- **MACD**, being more of a lagging indicator, is useful for confirming trends. Its smoother trajectory provides confirmation of larger market moves and helps to validate RSI signals.

Conclusion

This report demonstrated the use of **XGBoost**, **SVR**, and **Random Forest** models for stock price prediction, enhanced by the inclusion of technical indicators like **RSI** and **MACD** using **FRED API key**. Among the models, **XGBoost** provided the most accurate predictions, followed by **Random Forest**, while **SVR** struggled with the volatility in the data.

The combination of machine learning models and technical indicators helped to capture both short-term and long-term market trends. **RSI** and **MACD** offered additional insights into stock price momentum and potential reversals, complementing the predictive capabilities of the models. Overall, the approach successfully demonstrated the value of integrating technical analysis with machine learning for stock price forecasting.

References

1. **Chen, T., & Guestrin, C.** (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://doi.org/10.1145/2939672.2939785
2. **Cortes, C., & Vapnik, V.** (1995). Support-vector networks. *Machine Learning*, 20(3), 273-297. https://doi.org/10.1007/BF00994018
3. **Breiman, L.** (2001). Random Forests. *Machine Learning*, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324
4. **Wilder, J. W.** (1978). *New Concepts in Technical Trading Systems*. Trend Research.
5. **Appel, G.** (2005). *Technical Analysis: Power Tools for Active Investors*. FT Press.
6. **Murphy, J. J.** (1999). *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. New York Institute of Finance.
7. **Pedregosa, F., et al.** (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830. https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html
8. **Yahoo Finance API Documentation**. (n.d.). Retrieved from https://www.yfinance.org
9. **FRED Economic Data API**. (n.d.). Federal Reserve Economic Data (FRED). Retrieved from https://fred.stlouisfed.org