University of Science and Technology of Southern Mindanao
Cagayan de Oro Campus

Developing an Automated Stock Price Prediction System using Python and scikit-learn

Members:

Enrico Villasco
John Kyle Cuarteros
Carl Vince Dominguez
Kirk Vallecera

May 22, 2025
Date

1. Overview of the project

This project aims to develop a stock price prediction system using Python and the scikit-learn machine learning library. The primary focus is to leverage historical stock market data to build a predictive model capable of forecasting future stock prices. The system is designed to support financial analysis and decision-making by identifying trends and patterns in stock price movements.

The project workflow begins with data collection from Yahoo Finance, where historical stock prices are retrieved for analysis. The data is then preprocessed to handle missing values, normalize features, and prepare it for model training. Using supervised machine learning techniques—specifically regression models from scikit-learn—the system learns from past data to make future price predictions. The performance of the model is evaluated using standard metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 score to determine the accuracy of the forecasts.

The final output is an automated system that integrates data retrieval, model training, and prediction generation in a streamlined pipeline. This project not only demonstrates the practical application of machine learning in financial markets but also highlights the importance of data-driven forecasting in modern investment strategies.

2. Objective of the project

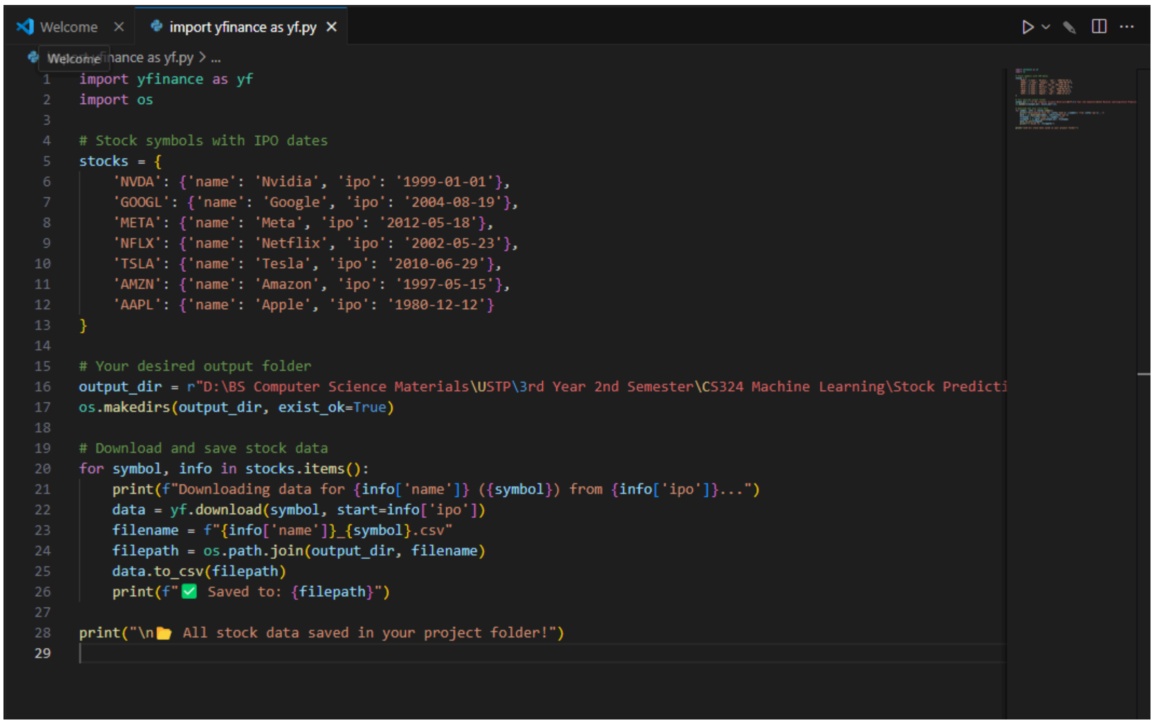
-To design and implement a machine learning-based system that predicts future stock prices using Python and scikit-learn.
-To evaluate the system's prediction accuracy by comparing model outputs with actual stock price movements.

3. Data Collection

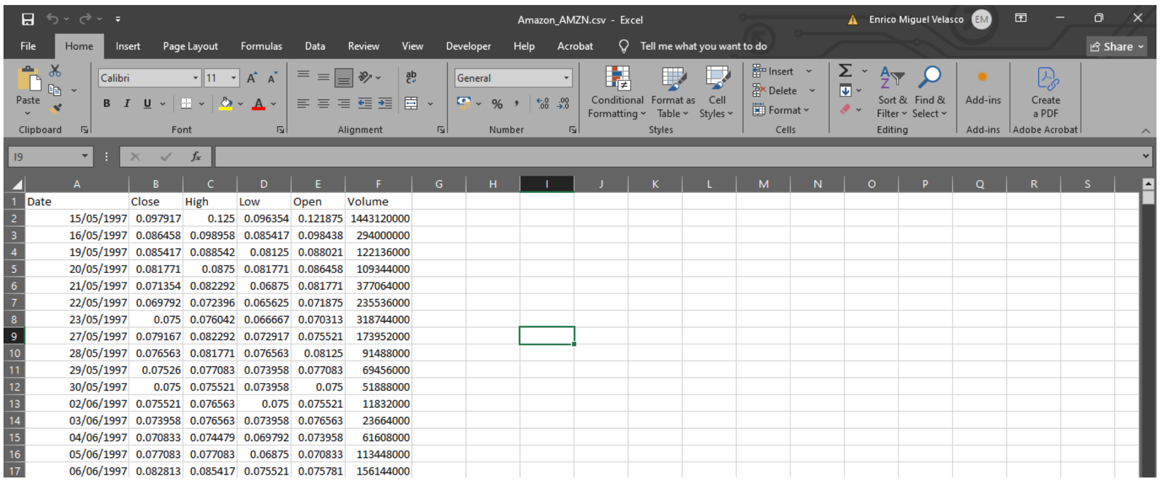
The dataset used for this project was obtained from Yahoo Finance, a widely recognized platform that provides reliable and up-to-date financial market data. Yahoo Finance allows users to search for publicly traded companies and download historical stock price data, including open, high, low, close, adjusted close prices, and trading volume.

Specifically, this project utilizes the historical stock data of the following American tech companies: Apple, Amazon, Google, Meta, Netflix, Nvidia and Tesla To collect the data programmatically, the project employed the `yfinance` Python library, which offers a convenient interface to fetch stock market data directly into a pandas

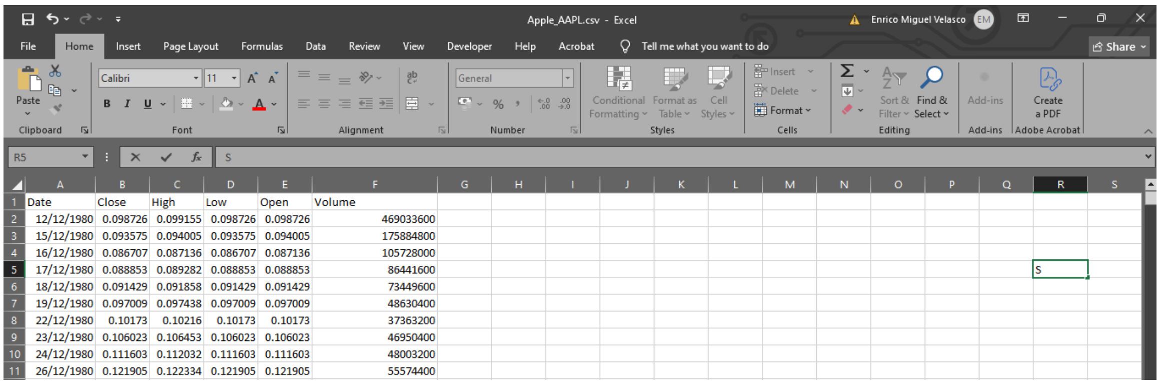
DataFrame. This method ensures consistent formatting, ease of use, and supports automation of data retrieval.



```python
import yfinance as yf
import os

# Stock symbols with IPO dates
stocks = {
    'NVDA': {'name': 'Nvidia', 'ipo': '1999-01-01'},
    'GOOGL': {'name': 'Google', 'ipo': '2004-08-19'},
    'META': {'name': 'Meta', 'ipo': '2012-05-18'},
    'NFLX': {'name': 'Netflix', 'ipo': '2002-05-23'},
    'TSLA': {'name': 'Tesla', 'ipo': '2010-06-29'},
    'AMZN': {'name': 'Amazon', 'ipo': '1997-05-15'},
    'AAPL': {'name': 'Apple', 'ipo': '1980-12-12'}
}

# Your desired output folder
output_dir = r"D:\BS Computer Science Materials\USTP\3rd Year 2nd Semester\CS324 Machine Learning\Stock Predicti
os.makedirs(output_dir, exist_ok=True)

# Download and save stock data
for symbol, info in stocks.items():
    print(f"Downloading data for {info['name']} ({symbol}) from {info['ipo']}...")
    data = yf.download(symbol, start=info['ipo'])
    filename = f"{info['name']}_{symbol}.csv"
    filepath = os.path.join(output_dir, filename)
    data.to_csv(filepath)
    print(f"✅ Saved to: {filepath}")

print("\n📁 All stock data saved in your project folder!")
```



	Date	Close	High	Low	Open	Volume
2	15/05/1997	0.097917	0.125	0.096354	0.121875	1443120000
3	16/05/1997	0.086458	0.098958	0.085417	0.098438	294000000
4	19/05/1997	0.085417	0.088542	0.08125	0.088021	122136000
5	20/05/1997	0.081771	0.0875	0.081771	0.086458	109344000
6	21/05/1997	0.071354	0.082292	0.06875	0.081771	377064000
7	22/05/1997	0.069792	0.072396	0.065625	0.071875	235536000
8	23/05/1997	0.075	0.076042	0.066667	0.070313	318744000
9	27/05/1997	0.079167	0.082292	0.072917	0.075521	173952000
10	28/05/1997	0.076563	0.081771	0.076563	0.08125	91488000
11	29/05/1997	0.07526	0.077083	0.073958	0.077083	69456000
12	30/05/1997	0.075	0.075521	0.073958	0.075	51888000
13	02/06/1997	0.075521	0.076563	0.075	0.075521	11832000
14	03/06/1997	0.073958	0.076563	0.073958	0.076563	23664000
15	04/06/1997	0.070833	0.074479	0.069792	0.073958	61608000
16	05/06/1997	0.077083	0.077083	0.06875	0.070833	113448000
17	06/06/1997	0.082813	0.085417	0.075521	0.075781	156144000



	Date	Close	High	Low	Open	Volume
2	12/12/1980	0.098726	0.099155	0.098726	0.098726	469033600
3	15/12/1980	0.093575	0.094005	0.093575	0.094005	175884800
4	16/12/1980	0.086707	0.087136	0.086707	0.087136	105728000
5	17/12/1980	0.088853	0.089282	0.088853	0.088853	86441600
6	18/12/1980	0.091429	0.091858	0.091429	0.091429	73449600
7	19/12/1980	0.097009	0.097438	0.097009	0.097009	48630400
8	22/12/1980	0.10173	0.10216	0.10173	0.10173	37363200
9	23/12/1980	0.106023	0.106453	0.106023	0.106023	46950400
10	24/12/1980	0.111603	0.112032	0.111603	0.111603	48003200
11	26/12/1980	0.121905	0.122334	0.121905	0.121905	55574400

4. Data Preprocessing

After collecting the historical stock price data from Yahoo Finance, the next crucial step involved preparing the data for model training. Preprocessing ensures that the dataset is clean, consistent, and appropriately formatted for machine learning algorithms.

Manual Data Inspection and Cleaning

4.1 Manual Inspection and Cleanup

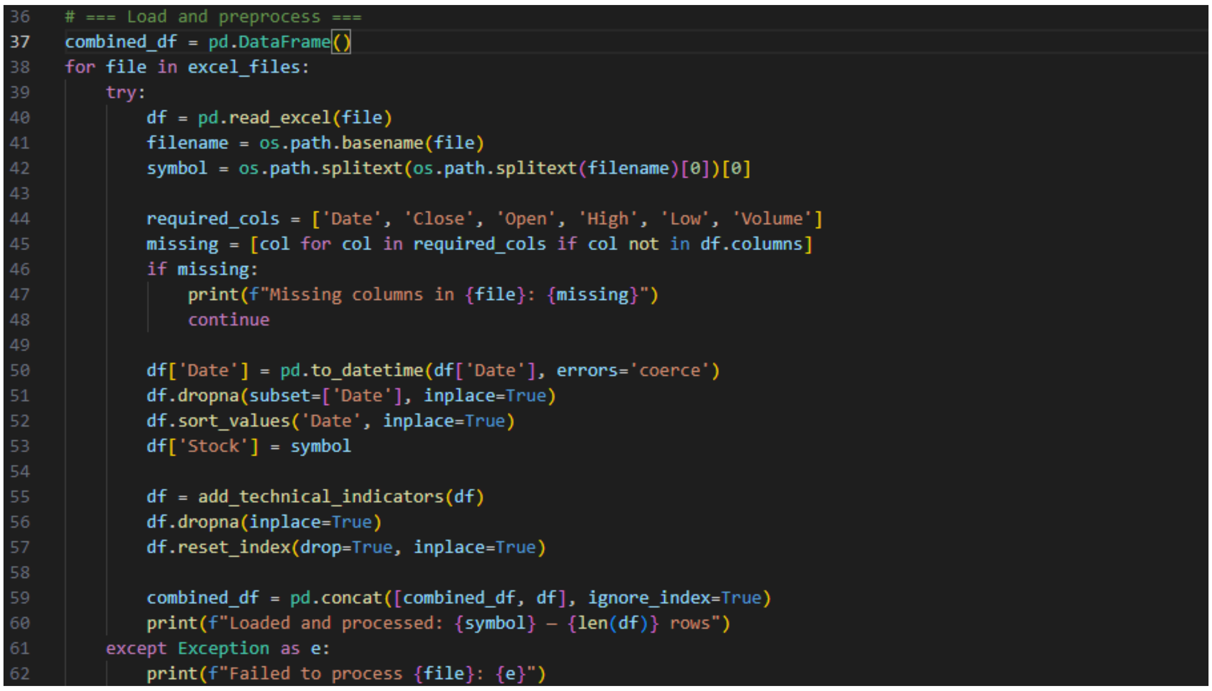
Before automated preprocessing, each Excel file downloaded from Yahoo Finance was manually checked using Microsoft Excel:

- **Header corrections** were made due to Yahoo Finance's inconsistent formatting.
- **Date columns** were visually inspected and adjusted where column widths caused values to appear hidden.
- No missing values were found during visual inspection.

4.2 Automated Data Loading and Cleaning

Using `pandas`, the project loaded and cleaned all Excel files in a loop:

- Checked for required columns (`Date`, `Close`, `Open`, `High`, `Low`, `Volume`)
- Converted `Date` columns to `datetime` format and sorted the data
- Added a stock symbol identifier (`Stock`) derived from the file name
- Applied custom technical indicators (e.g., Moving Average, RSI)
- Dropped rows with `NaN` values post-processing

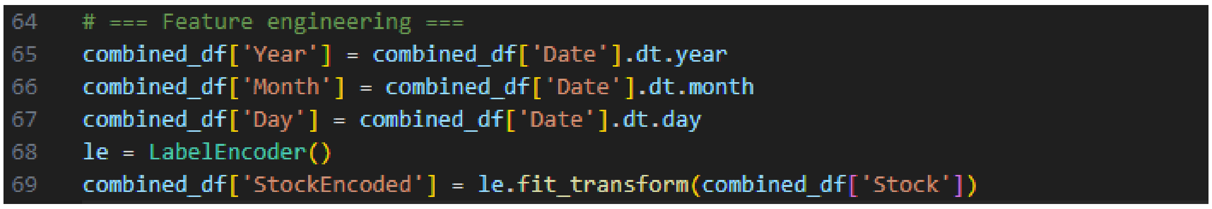


```python
# === Load and preprocess ===
combined_df = pd.DataFrame()
for file in excel_files:
    try:
        df = pd.read_excel(file)
        filename = os.path.basename(file)
        symbol = os.path.splitext(os.path.splitext(filename)[0])[0]

        required_cols = ['Date', 'Close', 'Open', 'High', 'Low', 'Volume']
        missing = [col for col in required_cols if col not in df.columns]
        if missing:
            print(f"Missing columns in {file}: {missing}")
            continue

        df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
        df.dropna(subset=['Date'], inplace=True)
        df.sort_values('Date', inplace=True)
        df['Stock'] = symbol

        df = add_technical_indicators(df)
        df.dropna(inplace=True)
        df.reset_index(drop=True, inplace=True)

        combined_df = pd.concat([combined_df, df], ignore_index=True)
        print(f"Loaded and processed: {symbol} – {len(df)} rows")
    except Exception as e:
        print(f"Failed to process {file}: {e}")
```

4.3 Feature Engineering

Several time-based and categorical features were added to enrich the dataset:

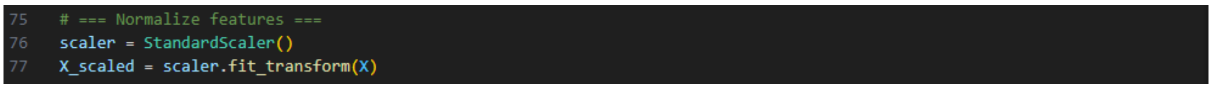
- **Extracted `Year`, `Month`, and `Day` from the `Date` column**
- **Encoded the `Stock` symbol as a numerical feature using `LabelEncoder`**



```python
# === Feature engineering ===
combined_df['Year'] = combined_df['Date'].dt.year
combined_df['Month'] = combined_df['Date'].dt.month
combined_df['Day'] = combined_df['Date'].dt.day
le = LabelEncoder()
combined_df['StockEncoded'] = le.fit_transform(combined_df['Stock'])
```

4.4 Feature Scaling

Standardization of input features was done using `StandardScaler` to normalize the scale of the data:



```python
# === Normalize features ===
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

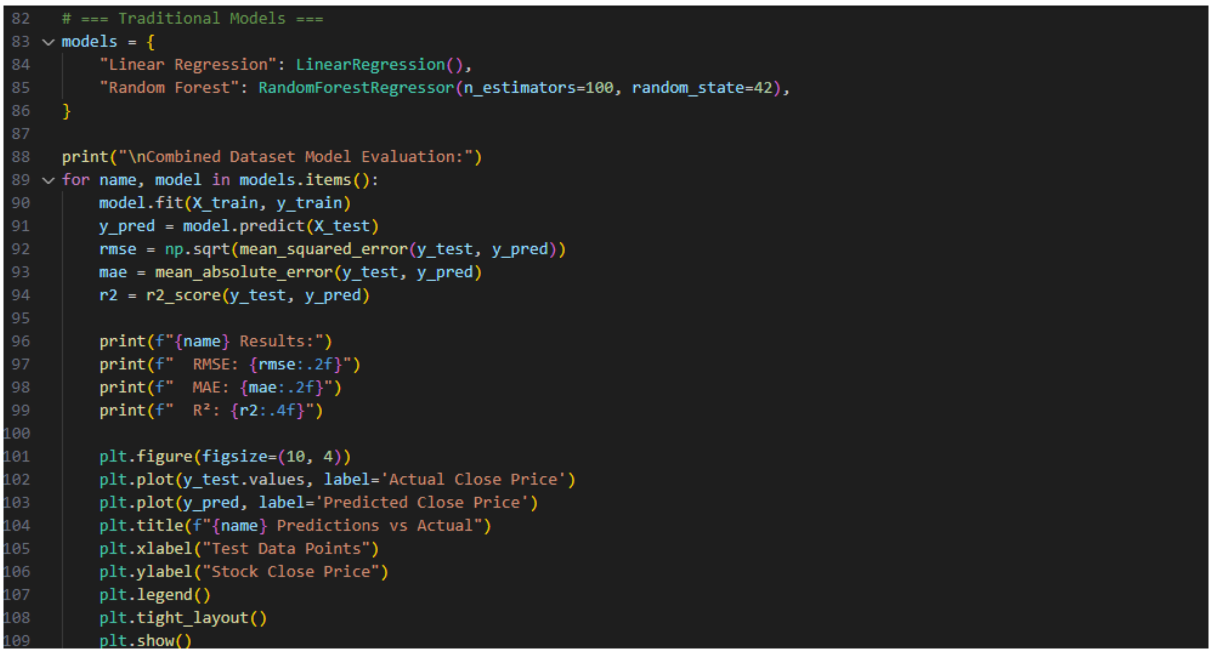
5. Modeling

In this project, three machine learning regression models were employed to predict stock prices based on historical data and engineered features:

Models Used:

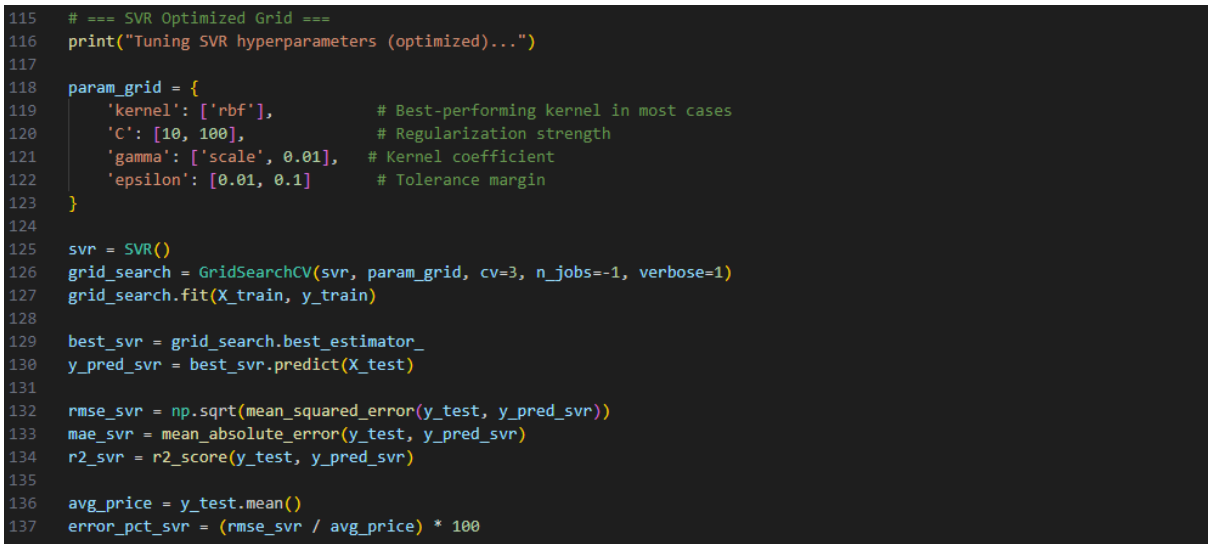
1. **Linear Regression**
   A simple yet effective baseline model that assumes a linear relationship between input features and the target variable. It is computationally efficient and provides interpretability, which makes it a good starting point for stock price prediction.

2. **Random Forest Regressor**
   An ensemble learning method based on decision trees. It handles non-linear relationships and interactions between features well, reducing the risk of overfitting through averaging multiple trees. Random Forest often performs robustly in financial datasets with complex patterns.



```python
# === Traditional Models ===
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=100, random_state=42),
}

print("\nCombined Dataset Model Evaluation:")
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"{name} Results:")
    print(f"  RMSE: {rmse:.2f}")
    print(f"  MAE: {mae:.2f}")
    print(f"  R²: {r2:.4f}")

    plt.figure(figsize=(10, 4))
    plt.plot(y_test.values, label='Actual Close Price')
    plt.plot(y_pred, label='Predicted Close Price')
    plt.title(f"{name} Predictions vs Actual")
    plt.xlabel("Test Data Points")
    plt.ylabel("Stock Close Price")
    plt.legend()
    plt.tight_layout()
    plt.show()
```

3. **Support Vector Regression (SVR)**
   SVR is a kernel-based model that can capture non-linear trends by mapping input features into higher-dimensional spaces. This model was included to explore potentially improved accuracy over the linear and tree-based models.



```python
# === SVR Optimized Grid ===
print("Tuning SVR hyperparameters (optimized)...")

param_grid = {
    'kernel': ['rbf'],           # Best-performing kernel in most cases
    'C': [10, 100],              # Regularization strength
    'gamma': ['scale', 0.01],   # Kernel coefficient
    'epsilon': [0.01, 0.1]       # Tolerance margin
}

svr = SVR()
grid_search = GridSearchCV(svr, param_grid, cv=3, n_jobs=-1, verbose=1)
grid_search.fit(X_train, y_train)

best_svr = grid_search.best_estimator_
y_pred_svr = best_svr.predict(X_test)

rmse_svr = np.sqrt(mean_squared_error(y_test, y_pred_svr))
mae_svr = mean_absolute_error(y_test, y_pred_svr)
r2_svr = r2_score(y_test, y_pred_svr)

avg_price = y_test.mean()
error_pct_svr = (rmse_svr / avg_price) * 100
```

Why These Models?

- **Diversity in approach:** Combining linear, ensemble tree-based, and kernel-based models allowed comparison of different learning mechanisms on the dataset.

- **Performance and interpretability:** Linear Regression provides a baseline and interpretability, while Random Forest and SVR can model complex data patterns.

- **Suitability for regression:** All three are well-established for regression tasks and are available in the scikit-learn library.

Computational Considerations

Training the SVR model proved to be the most resource-intensive step. The development laptop used for this project has:

- Intel Core i5 10th generation CPU with 8 cores

- NVIDIA MX130 GPU (entry-level)

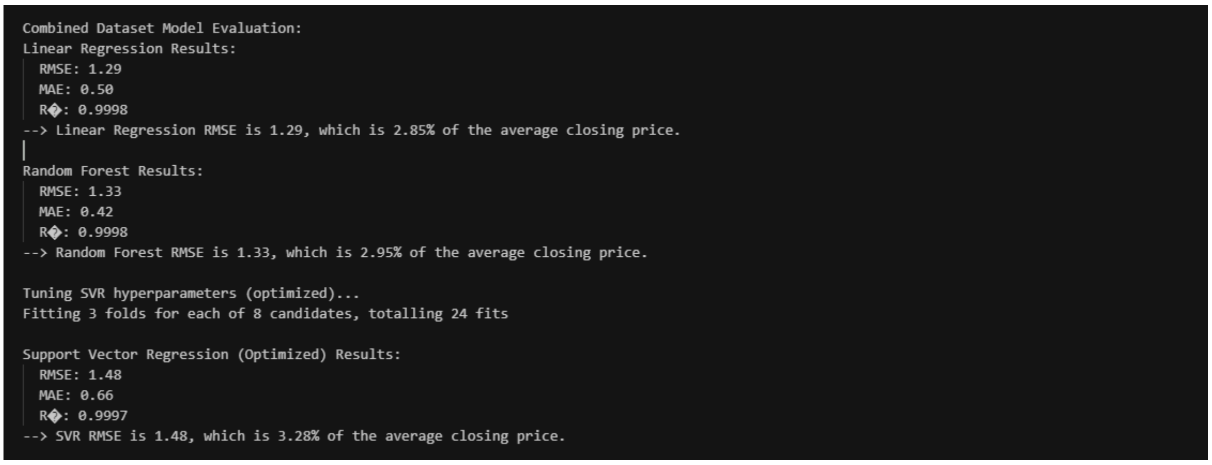
Due to these hardware limitations, SVR training and hyperparameter tuning required significant time compared to the other models. Despite this, SVR produced competitive results, justifying its inclusion.

6. Evaluation

The evaluation of the stock price prediction models was performed using three standard regression metrics:

- **Root Mean Squared Error (RMSE)**
- **Mean Absolute Error (MAE)**
- **R² Score (Coefficient of Determination)**

These metrics help assess how accurately the models predicted stock prices compared to the actual values. The models were trained and tested on a combined dataset consisting of the historical stock prices of Amazon, Apple, Google, Meta, Netflix, Nvidia, and Tesla, resulting in a diverse and comprehensive input space.



```
Combined Dataset Model Evaluation:
Linear Regression Results:
  RMSE: 1.29
  MAE: 0.50
  R�: 0.9998
--> Linear Regression RMSE is 1.29, which is 2.85% of the average closing price.

Random Forest Results:
  RMSE: 1.33
  MAE: 0.42
  R�: 0.9998
--> Random Forest RMSE is 1.33, which is 2.95% of the average closing price.

Tuning SVR hyperparameters (optimized)...
Fitting 3 folds for each of 8 candidates, totalling 24 fits

Support Vector Regression (Optimized) Results:
  RMSE: 1.48
  MAE: 0.66
  R�: 0.9997
--> SVR RMSE is 1.48, which is 3.28% of the average closing price.
```

Combined Dataset Evaluation Summary

Model	**RMSE**	**MAE**	**R² Score**	**% of Avg Closing Price**
Linear Regression	1.29	0.50	0.9998	2.85%
Random Forest Regressor	1.33	0.42	0.9998	2.95%
Support Vector Regressor (SVR)	1.48	0.66	0.9997	3.28%

Evaluation Metrics Explained:

1. **RMSE (Root Mean Squared Error)**
   This measures the average magnitude of the prediction error, with larger errors having a disproportionately higher impact due to squaring.

   ○ **Lower RMSE = Better accuracy**

   ○ In our case, the RMSE values ranged from **1.29 to 1.48**, which is very low considering that this is less than 3.3% of the average stock price in the dataset.

   ○ This suggests the models are very precise in estimating future closing prices.

2. **MAE (Mean Absolute Error)**
   This measures the average absolute difference between predicted and actual values. Unlike RMSE, it treats all errors equally.

   ○ **Lower MAE = More consistent accuracy**

   ○ The MAE values we observed ranged from **0.42 to 0.66**, indicating that on average, predictions were less than a dollar off from the actual prices.

3. **R^2 Score (Coefficient of Determination)**
   This measures how well the model explains the variability of the target variable (stock price).

   ○ **R^2 close to 1 = The model explains nearly all variability**

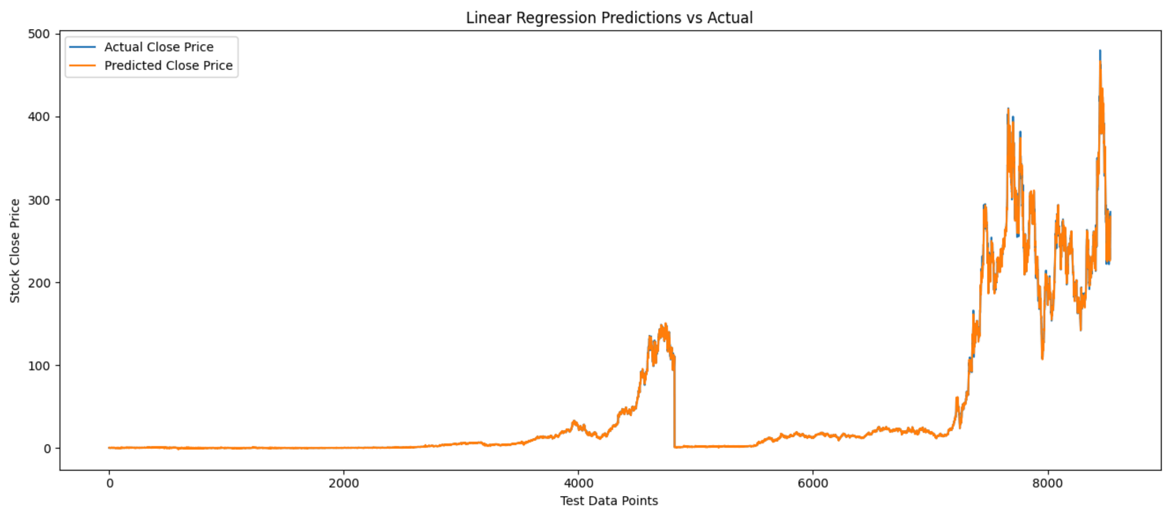
   ○ All models had an R^2 score of **0.9997 or higher**, which means nearly **99.97%** of the variation in stock prices is explained by the model—an excellent fit.

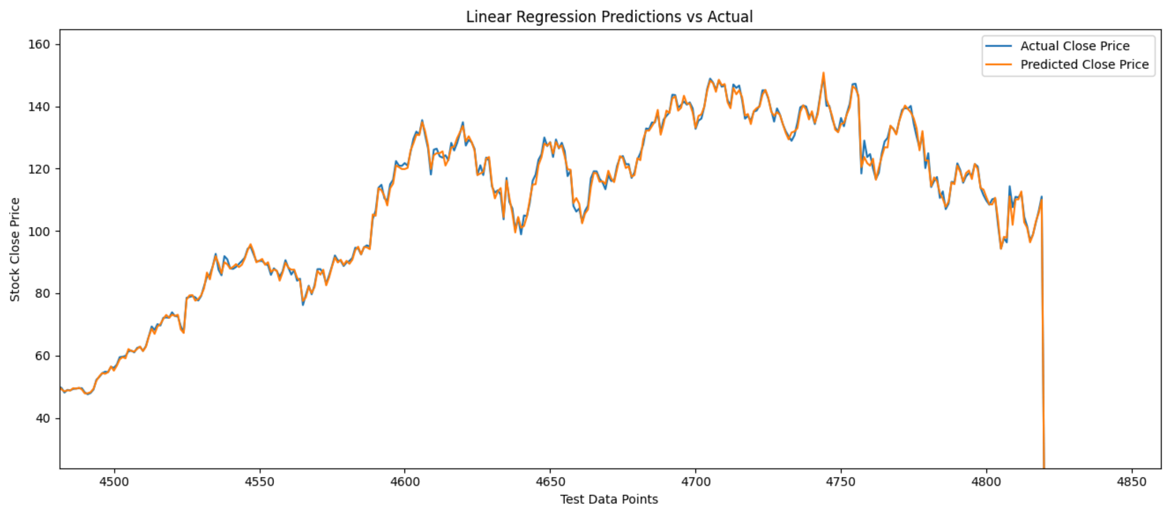
Interpretation

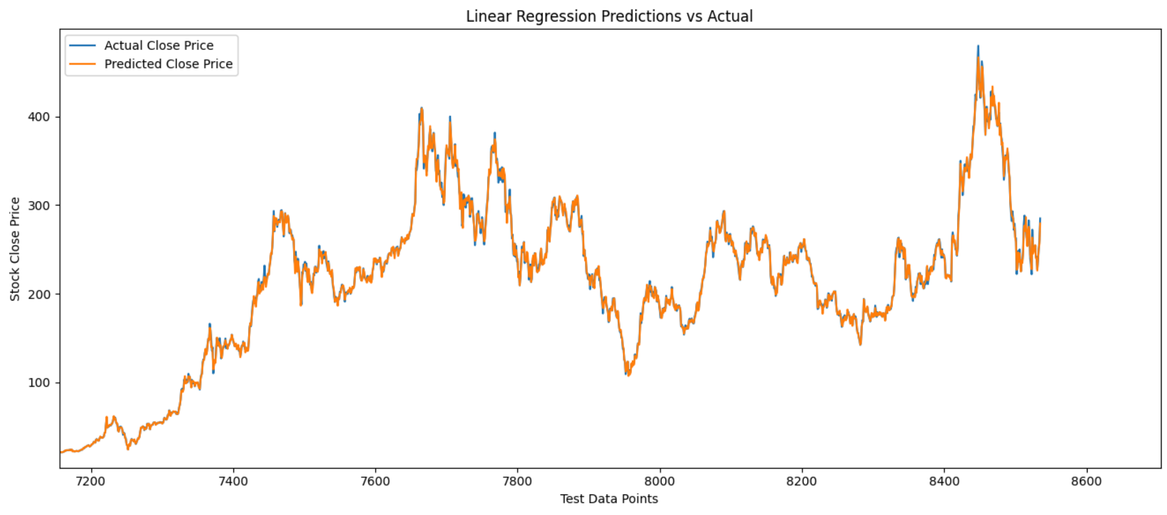
- All three models performed **extremely well**, achieving near-perfect R² scores and very low RMSE and MAE values.
- The **Linear Regression** model was the fastest to train and still very accurate, making it a reliable baseline.
- The **Random Forest** model offered slightly better MAE but took more time and memory due to its ensemble nature.
- The **SVR model**, while slightly slower (especially on a mid-tier laptop), still yielded competitive results with a low error margin.

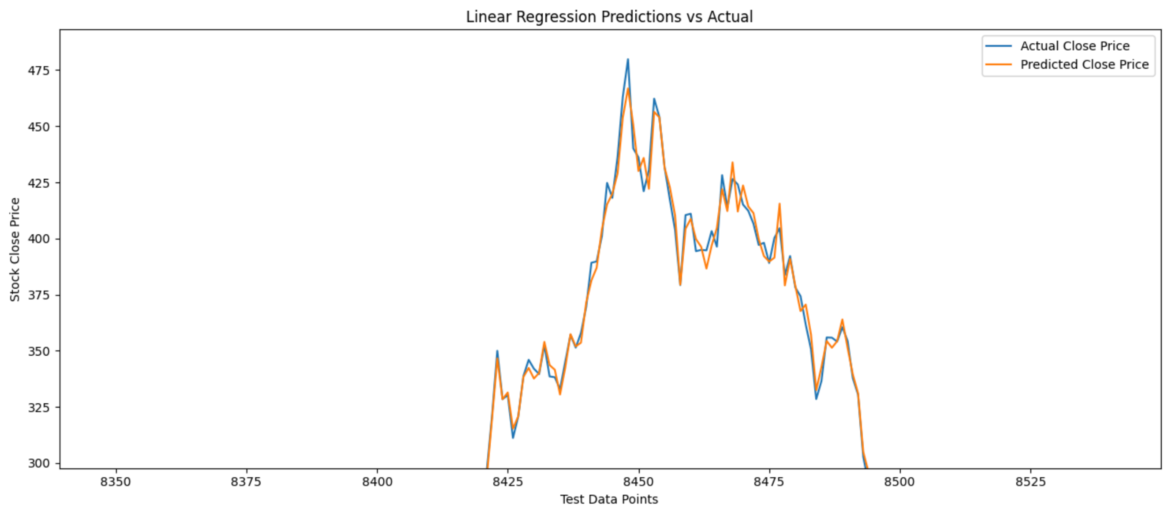
These evaluations show that the system is capable of learning complex patterns in stock price data and can make accurate predictions for unseen data. The model's performance also validates the effectiveness of the data preprocessing and feature engineering steps taken earlier in the pipeline.

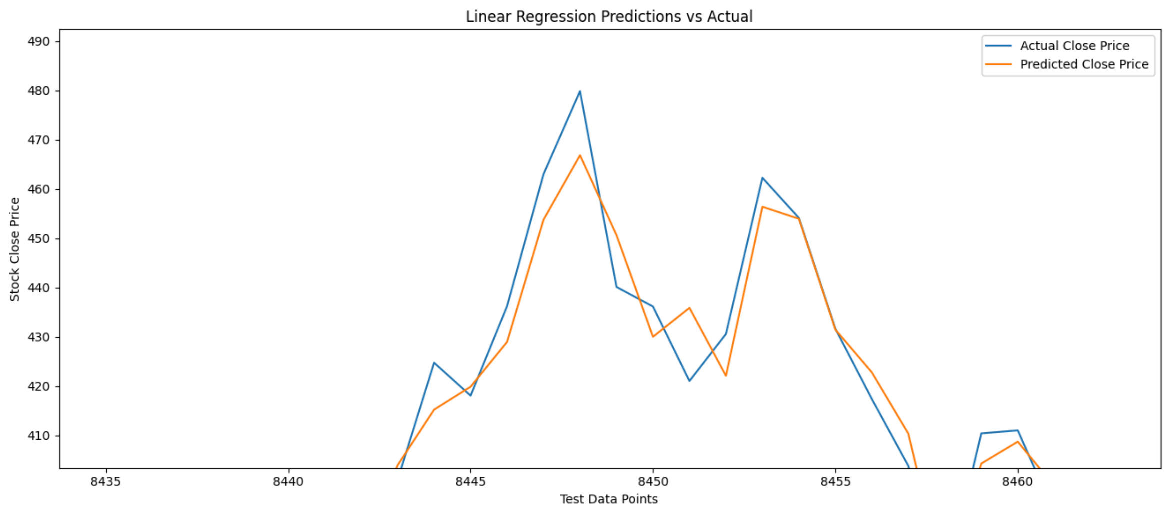
7. Results



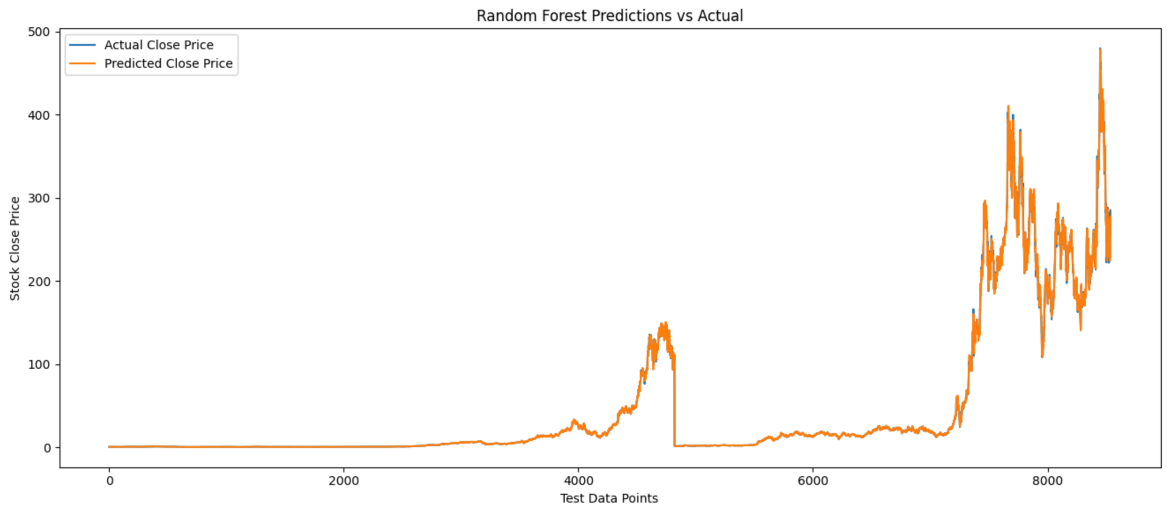

Linear Regression Predictions vs Actual
Actual Close Price
Predicted Close Price
Stock Close Price
Test Data Points
160
140
120
100
80
60
40
4500
4550
4600
4650
4700
4750
4800
4850

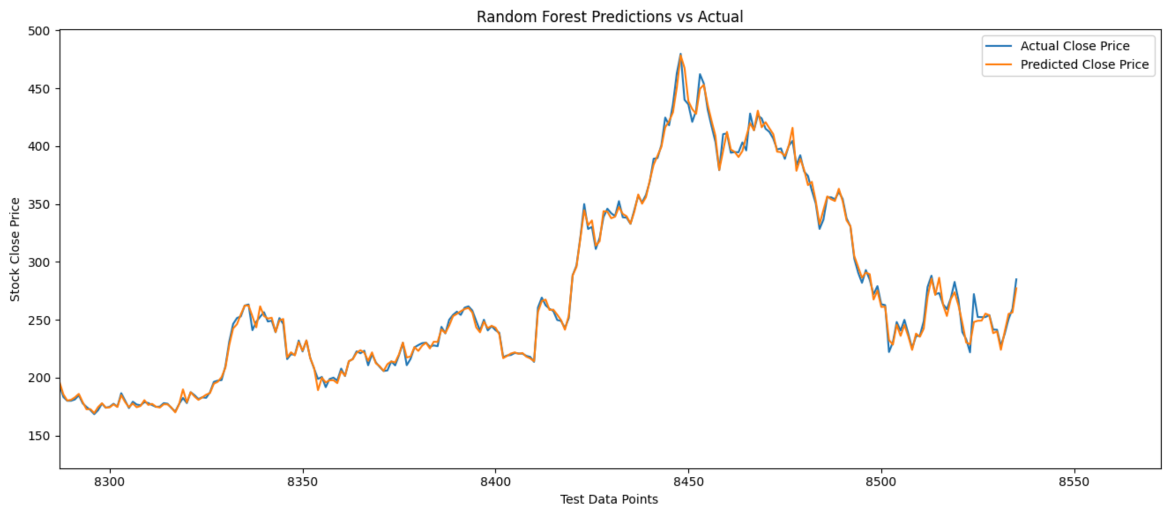

Linear Regression Predictions vs Actual
Actual Close Price
Predicted Close Price
Stock Close Price
Test Data Points
400
300
200
100
7200
7400
7600
7800
8000
8200
8400
8600

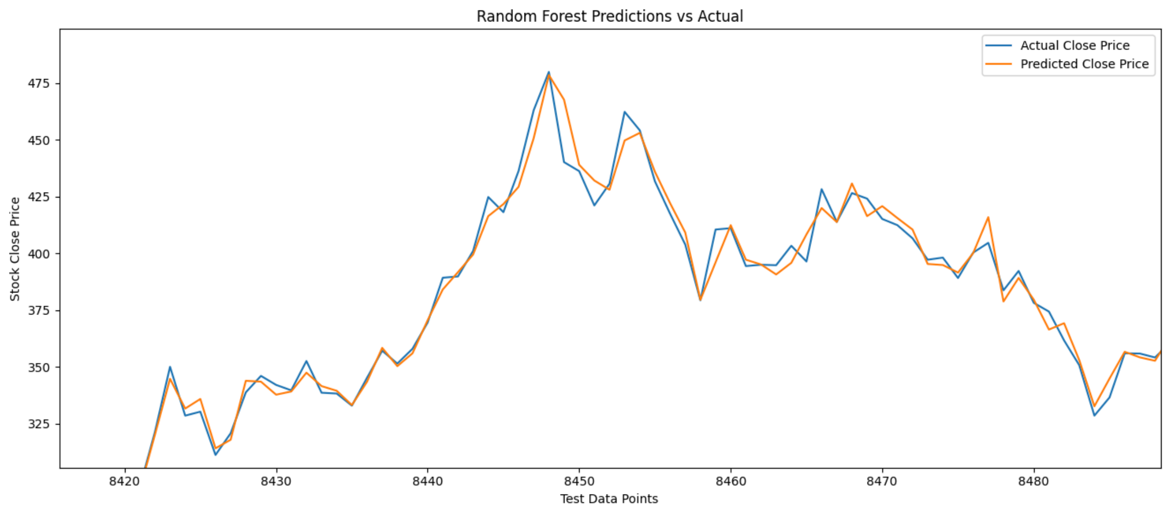

Linear Regression Predictions vs Actual
Actual Close Price
Predicted Close Price
Stock Close Price
Test Data Points
475
450
425
400
375
350
325
300
8350
8375
8400
8425
8450
8475
8500
8525

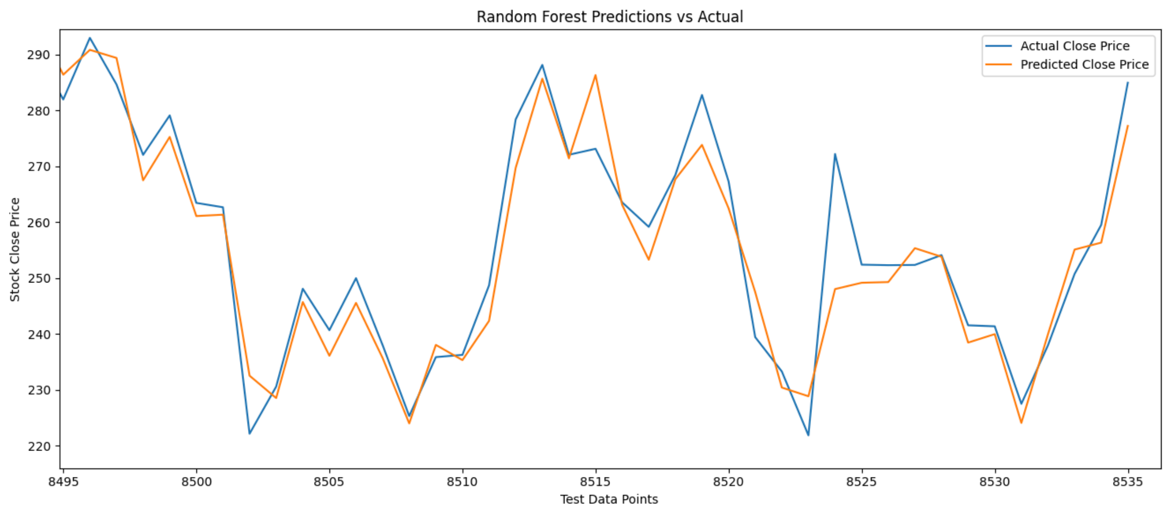


Graphs for the Linear Regression model suggest that the predicted values almost mirror the actual values of the combined stock data, which suggest that it is almost accurate with up to almost 99.9x% accuracy, which suggest that Linear Regression is well suited to predict stock prices.

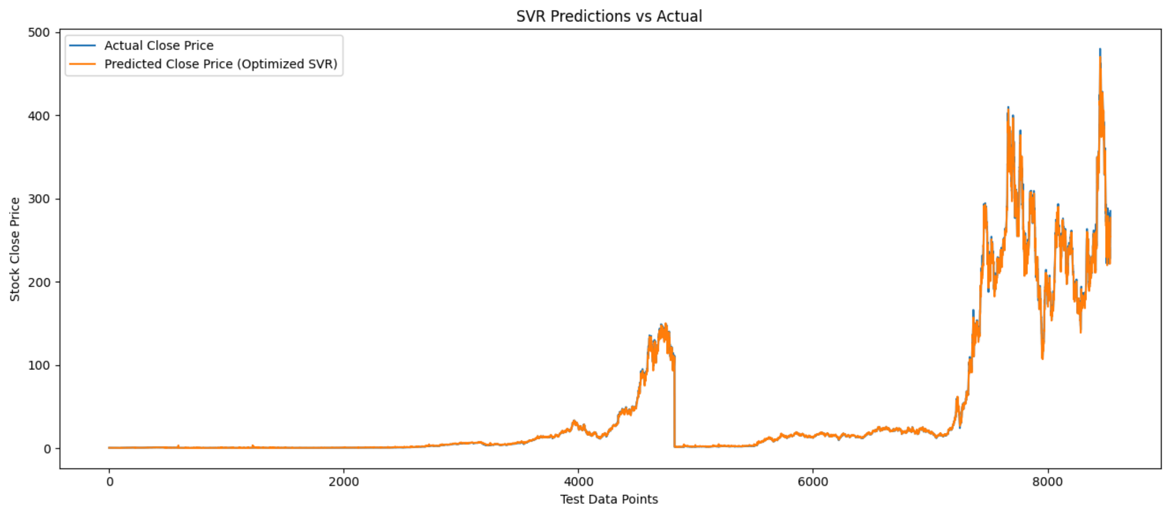


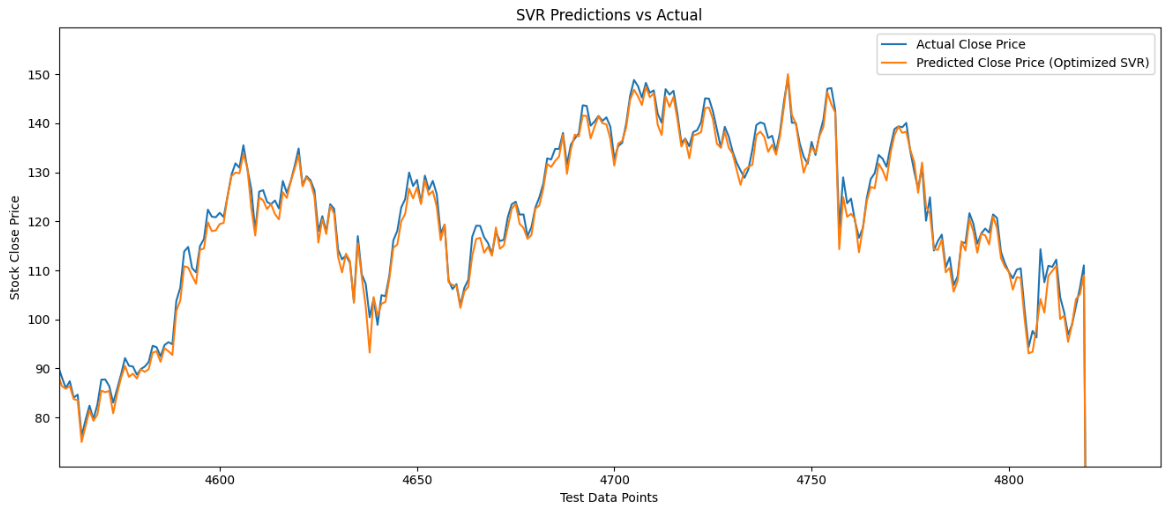

Random Forest Predictions vs Actual
Actual Close Price
Predicted Close Price
Stock Close Price
Test Data Points
500
450
400
350
300
250
200
150
8300
8350
8400
8450
8500
8550

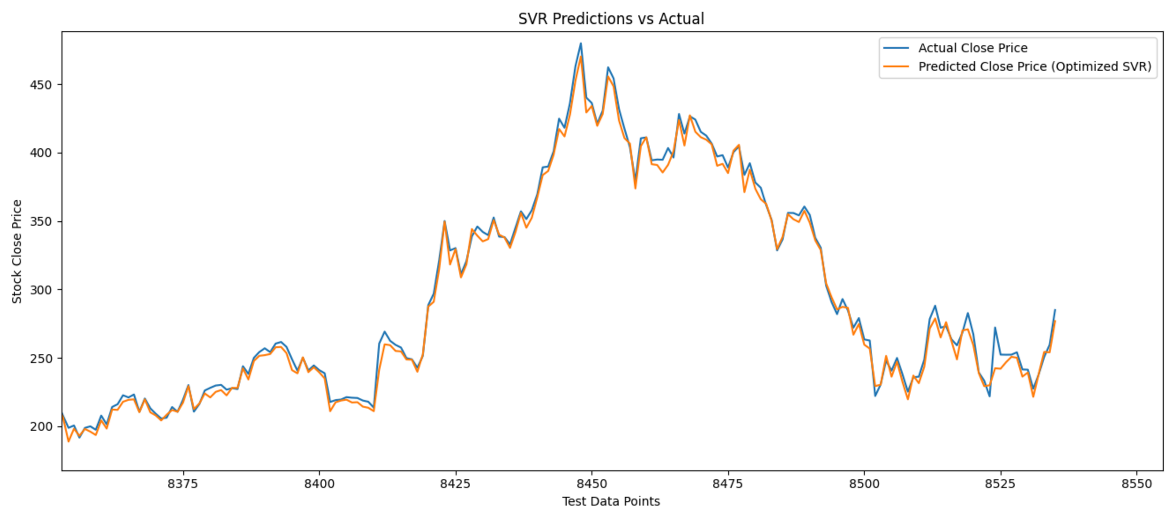

Random Forest Predictions vs Actual
Actual Close Price
Predicted Close Price
Stock Close Price
Test Data Points
475
450
425
400
375
350
325
8420
8430
8440
8450
8460
8470
8480

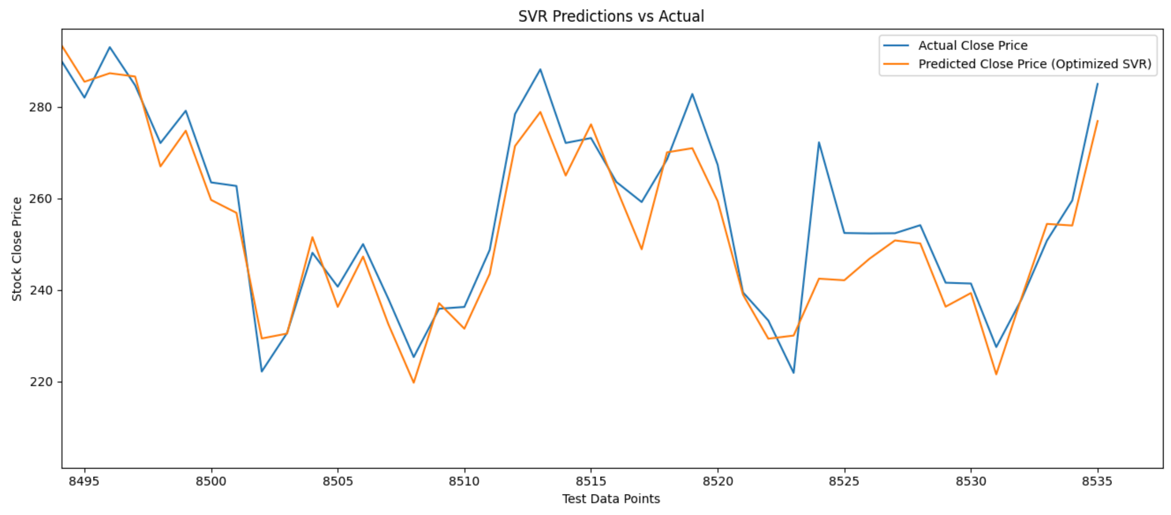

Random Forest Predictions vs Actual
Actual Close Price
Predicted Close Price
Stock Close Price
Test Data Points
290
280
270
260
250
240
230
220
8495
8500
8505
8510
8515
8520
8525
8530
8535

Random forest graphs from the same program also suggest that the Random Forest model is well-fitted to be used in predicting stock prices, however, since the project is not currently using live data due to access restrictions from Yahoo Finance (needing a subscription before live data can be accessed), if ever implemented for investing or algorithmic trading purposes, alternative sources for live data must be found.









SVR prediction vs. actual closing stock prices suggest that the Support Vector Regression model is slightly inaccurate compared to the Linear Regression and Random Forest models. SVRs are derived from Support Vector Machines, which are normally used for classification, but are used to find relationships between features and target within a certain margin of error (e). SVRs are robust, in that they tend to ignore outlier variables, useful in algorithmically trading stocks, where prices are often changing. Despite the slow hardware used to model the data, the SVR model still demonstrated strong results.