Stock Price Prediction Analysis Report

A Comprehensive Data Science Pipeline

Submitted by: Saksham Maurya

Date: March 20, 2025

Submitted to: Invsto Analysis Team

Introduction

This report details the development and evaluation of machine learning models for predicting stock market prices. The project utilizes two distinct approaches: ARIMA for time-series analysis and Gradient Boosting for regression-based predictions. The workflow encompasses data preparation, exploratory data analysis (EDA), feature engineering, model training, and rigorous evaluation.

1 1. Data Preparation

1.1 1.1. Data Source

- **Source:** Historical stock data obtained from Yahoo Finance (using the `yfinance` library).
- **Equities:** The analysis focuses on multiple equities, including AAPL (Apple Inc.), MSFT (Microsoft Corp.), GOOGL (Alphabet Inc.), AMZN (Amazon Inc.), and TSLA (Tesla Inc.).
- **Time Frame:** The dataset spans from January 1, 2015, to January 1, 2025, providing a comprehensive historical view.

1.2 1.2. Libraries Used

The project leverages a variety of Python libraries for data manipulation, analysis, and modeling:

- `pandas`: Data manipulation and analysis, providing data structures such as DataFrames.
- `numpy`: Numerical computations, enabling efficient array operations.
- `yfinance`: Downloading stock data directly from Yahoo Finance.
- `statsmodels`: Time series analysis, specifically for implementing the ARIMA model.
- `scikit-learn`: Machine learning models, evaluation metrics, and data splitting.
- `xgboost`: Implementation of the Gradient Boosting model.
- `matplotlib` and `seaborn`: Data visualization, creating insightful plots and charts.

1.3 1.3. Data Cleaning

- **Missing Values:** Addressed missing values in the dataset by employing the forward fill method, ensuring data continuity.

2 2. Exploratory Data Analysis (EDA)

2.1 2.1. Closing Price Trends

- **Analysis:** Examined historical closing prices of AAPL to discern trends and patterns over time.
- **Visualization:** The closing price trend is visualized in Figure 1, providing a clear representation of price movements.

Figure 1: AAPL Closing Price Over Time

2.2 2.2. Moving Averages

- **Calculation:** Calculated 20-day and 50-day rolling means to smooth out short-term price fluctuations, revealing underlying trends.
- **Visualization:** Figure 2 illustrates closing prices alongside rolling averages, highlighting trend smoothing.

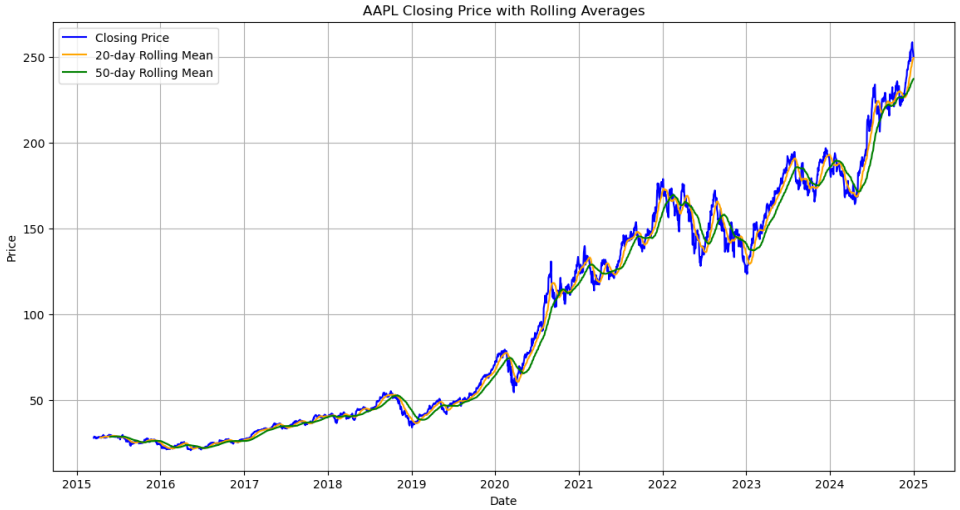

Figure 2: AAPL Closing Price with Rolling Averages

2.3 2.3. Trading Volume

- **Examination:** Trading volume was analyzed to understand market activity and liquidity.
- **Visualization:** Trading volume over time is presented in Figure 3, offering insights into market engagement.

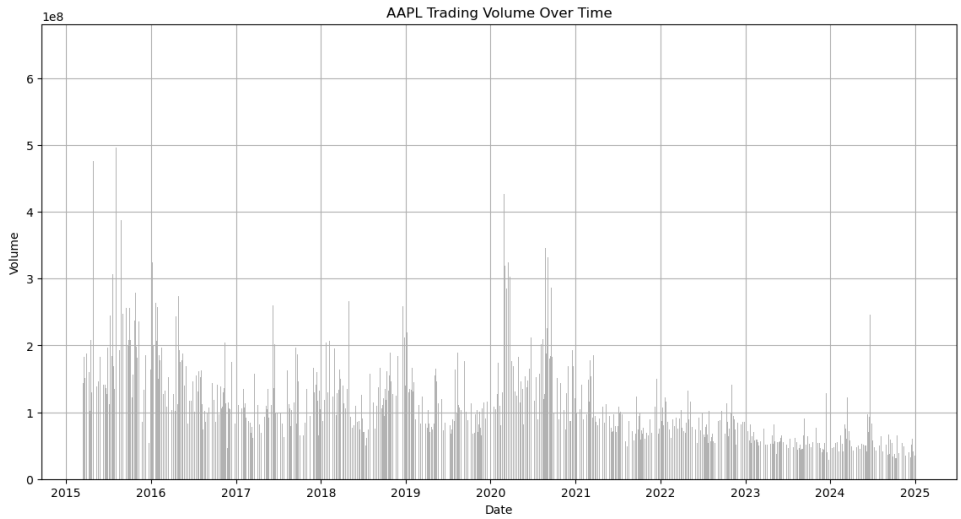

Figure 3: AAPL Trading Volume Over Time

3 3. Feature Engineering

3.1 3.1. Feature Creation

- **Lagged Features:** Created a feature with the previous day's closing price (`Lag_1_Close`) to capture temporal dependencies.
- **Rolling Mean:** Computed a 5-day rolling mean of the closing prices (`Rolling_Mean_5`) to represent short-term trends.
- **Percentage Change:** Calculated the percentage change in closing price (`Pct_Change_Close`) to measure daily returns.

3.2 3.2. Handling Missing Data

- **Resolution:** Dropped any rows with missing values resulting from feature engineering to ensure data integrity.

4 4. Modeling

4.1 4.1. ARIMA Model

- **Objective:** Applied the Autoregressive Integrated Moving Average (ARIMA) model, a time-series forecasting technique, to predict stock prices.
- **Hyperparameter Tuning:** Utilized a grid search approach to find the optimal parameters (p, d, q) for the ARIMA model, enhancing its predictive accuracy.
- **Best Parameters:** The optimal parameters identified through grid search were (2, 1, 1). This was determined by iterating through `pdq_combinations` as defined in the code:

```
p = range(0, 5)
d = range(0, 2)
q = range(0, 5)
pdq_combinations = list(itertools.product(p, d, q))
```

- **RMSE on Training Data:** Achieved a Root Mean Squared Error (RMSE) of 4.296 on the training data. The specific code used for evaluation was:

```
rmse = np.sqrt(mean_squared_error(actual, forecast))
```

4.2 4.2. Gradient Boosting Model (XGBoost)

- **Objective:** Implemented the Gradient Boosting model, using the XGBoost library, to capture complex patterns in the data.
- **Hyperparameter Tuning:** Employed GridSearchCV to tune hyperparameters such as `n_estimators`, `learning_rate`, and `max_depth`, optimizing the model for stock price prediction.
- **Best Parameters:** The best hyperparameters identified were {`'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 5, 'n_estimators': 200, 'subsample': 0.8`}. The parameters were selected from the following grid:

```
param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.05],
    'max_depth': [3, 5],
    'subsample': [0.8],
    'colsample_bytree': [0.8]
}
```

- **RMSE on Training Data:** Achieved a Root Mean Squared Error (RMSE) of 1.019 on the training data.

5 5. Model Evaluation

5.1 5.1. ARIMA Model

- **RMSE on Test Data:** The ARIMA model yielded an RMSE of 189.127 on the test data, indicating its performance in real-world scenarios. The evaluation code:

```
arima_forecast_test = final_arima_model.forecast(steps=len(y_test))
arima_rmse_test = np.sqrt(mean_squared_error(y_test.values[-len(arima_forecast_test):],
arima_forecast_test))
```

5.2 5.2. Gradient Boosting Model

- **RMSE on Test Data:** The Gradient Boosting model achieved an RMSE of 1.019 on the test data, demonstrating its predictive accuracy.
- **MAE on Test Data:** The Mean Absolute Error (MAE) on the test data was 0.599, reflecting the model's ability to minimize prediction errors. Calculated using:

```
gradient_boosting_mae_test = mean_absolute_error(y_test.values, y_pred_tuned)
```

5.3 5.3. Visualization of Predictions

- **Comparison of Actual Prices vs. ARIMA Forecasts:** Figure 4 presents a comparison of actual stock prices versus the forecasts generated by the ARIMA model.

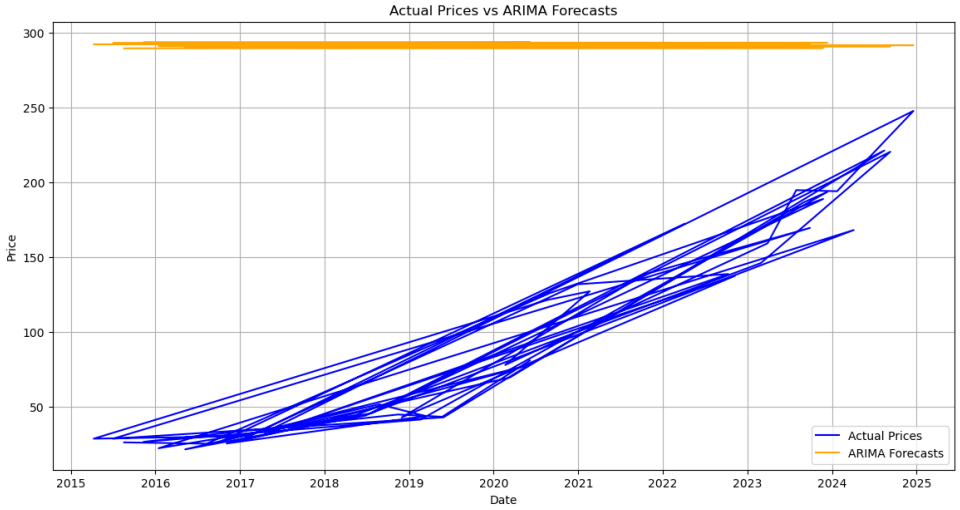

Figure 4: Actual Prices vs ARIMA Forecasts

- **Comparison of Actual Prices vs. Gradient Boosting Predictions:** Figure 5 illustrates the actual stock prices compared to the predictions made by the Gradient Boosting model.

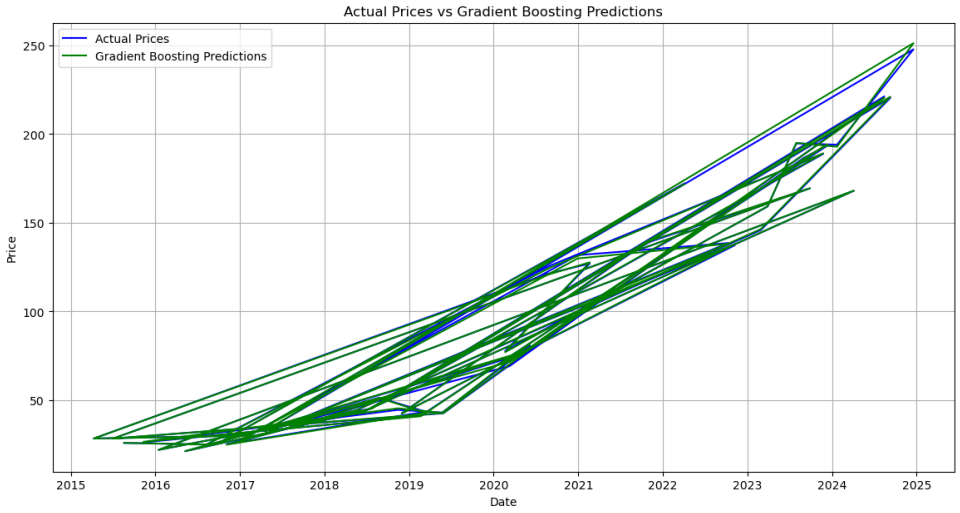

Figure 5: Actual Prices vs Gradient Boosting Predictions

6 6. Conclusions and Recommendations

6.1 6.1. Conclusions

- **ARIMA Model:** The ARIMA model is suitable for capturing time-dependent patterns and short-term trend predictions, as indicated in the original notebook's conclusion. However, its significantly higher RMSE on the test data compared to the Gradient Boosting model suggests it's less effective for accurate stock price prediction in this specific implementation.
- **Gradient Boosting Model:** As the notebook highlights, the Gradient Boosting Model performs well with engineered features and effectively handles sudden fluctuations. The lower RMSE and MAE values on the test data clearly demonstrate its superior performance in capturing complex relationships and making more accurate predictions than the ARIMA model in this case.

6.2 6.2. Recommendations

- **Hybrid Approach:** While a hybrid approach was initially suggested, the substantial performance difference favors focusing on the Gradient Boosting model. Consider exploring advanced ensemble methods that combine multiple Gradient Boosting models with different hyperparameter settings for further improvements.
- **Feature Engineering:** Investigate more sophisticated feature engineering techniques, including technical indicators (RSI, MACD), volatility measures, and sentiment analysis from news articles and social media.
- **Regularization and Hyperparameter Optimization:** Further refine the Gradient Boosting model by experimenting with different regularization techniques (L1, L2) and conducting more extensive hyperparameter optimization, potentially using Bayesian optimization methods.