Predicting Future Stock Prices

Math 448 – Introduction to Statistical Learning and Data Mining

Master of Science

In

Statistical Data Science

by

Pooja Chavanpatil
San Francisco, California

Professor Tao He
San Francisco State University

May 2024

Table of Contents

List of Figures

1. Executive Summary

Problem Statement: “This project aims to develop a machine learning model capable of predicting the next day's stock price for companies listed in the S&P 500 index, focusing on Microsoft (MSFT) as a case study.” The project involves a comprehensive approach that includes data preprocessing, feature engineering, model training, evaluation, and visualization to ensure robust and reliable stock price predictions.

The datasets used for this project include sp500_companies, sp500_index, and sp500_stocks, all sourced from Kaggle (LARXEL, n.d.). Data preprocessing involved converting date columns to datetime format and handling missing values by dropping or filling in gaps. Feature engineering created new predictors like rolling averages, market cap to EBITDA ratio, lagged S&P 500 index values, EMAs, MACD, volatility, and percentage changes.

Data was scaled using StandardScaler, followed by a train-test split with 70% for training and 30% for testing. Various regression models were trained, including Linear Regression, Lasso, Ridge, Random Forest Regressor, Gradient Boosting Regressor, and Bagging, with hyperparameter tuning using GridSearchCV for optimization.

Visual analysis compared predictions versus actual stock prices for each model. Model performance was evaluated using Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2). Linear Regression showed the best overall performance, followed by Lasso and Ridge.

In conclusion, the project demonstrates the effectiveness of machine learning in stock price prediction. Recommendations include using Linear Regression for their accuracy and interpretability, adding macroeconomic indicators and sentiment analysis for better performance, and regularly updating models with new data. Additionally, exploring Long Short-Term Memory

(LSTM) models is recommended due to their ability to capture temporal dependencies, which can be particularly beneficial for time series forecasting in financial markets.

2. Data Overview

Three data sets from Kaggle (LARXEL, n.d.) are used.

- **sp500_companies:** Named as df_company which contains Company metadata for S&P 500 companies.

	Exchange	Symbol	Shortname	Longname	Sector	Industry	Currentprice	Marketcap	Ebitda	Revenuegrowth	City	Country	Longbusinesssummary	Weight
0	NMS	MSFT	Microsoft Corporation	Microsoft Corporation	Technology	Software - Infrastructure	430.16	3197082271744	1.251820e+11	0.170	Redmond	United States	Microsoft Corporation develops and supports so...	0.064629
1	NMS	AAPL	Apple Inc.	Apple Inc.	Technology	Consumer Electronics	189.98	2913172193280	1.296290e+11	-0.043	Cupertino	United States	Apple Inc. designs, manufactures, and markets ...	0.058890
2	NMS	NVDA	NVIDIA Corporation	NVIDIA Corporation	Technology	Semiconductors	1064.69	2618956185600	4.927500e+10	2.621	Santa Clara	United States	NVIDIA Corporation provides graphics and compu...	0.052942
3	NMS	GOOGL	Alphabet Inc.	Alphabet Inc.	Communication Services	Internet Content & Information	174.99	2171083423744	1.097230e+11	0.154	Mountain View	United States	Alphabet Inc. offers various products and plat...	0.043888
4	NMS	GOOG	Alphabet Inc.	Alphabet Inc.	Communication Services	Internet Content & Information	176.33	2168841437184	1.097230e+11	0.154	Mountain View	United States	Alphabet Inc. offers various products and plat...	0.043843

- **sp500_index:** Named as df_index which contains S&P index value each day.

index	Date	S&P500
0	2014-05-27	1911.91
1	2014-05-28	1909.78
2	2014-05-29	1920.03
3	2014-05-30	1923.57
4	2014-06-02	1924.97

- **sp500_stocks :** Named as df_stocks which contains Individual stock features such as Date, Symbol, Adj Close, Close, High, Low, Open

	Date	Symbol	Adj Close	Close	High	Low	Open	Volume
0	2010-01-04	MMM	40.553387	69.414719	69.774246	69.122070	69.473244	3640265.0
1	2010-01-05	MMM	40.299381	68.979935	69.590302	68.311035	69.230766	3405012.0
2	2010-01-06	MMM	40.870903	69.958191	70.735786	69.824417	70.133781	6301126.0
3	2010-01-07	MMM	40.900215	70.008362	70.033447	68.662209	69.665550	5346240.0
4	2010-01-08	MMM	41.188412	70.501671	70.501671	69.648827	69.974915	4073337.0

3. Data cleaning and Pre-processing

Based on the info of dataframes df_company has below columns with null rows - Revenue growth, State & , Fulltimeemployees.

- **Dropping fields** - Dropping 'State', 'Fulltimeemployees' fields entirely. I am not going to use "State" and "Fulltimeemployees" column for future stock price prediction.
- **Adding data** - Revenue growth for Western Digital Corporation is missing I have added this data to the dataframe (stockanalysis, n.d.)

4. Data Visualization

Top 10 Companies in Market Cap

To list the top 10 companies in market cap as shown in figure 1 we sort the data by market cap. Market cap is calculated by multiplying the current stock price by the total number of outstanding shares. This ranking reflects the collective perception of investors regarding the size, stability, and growth potential of these companies.

This plot provides insights into the composition of the market by showing which companies have the greatest weight in the index. Since the S&P 500 is a market-capitalization-weighted index, larger companies have more influence on the index's movements. Currently **Microsoft has the highest market cap and weightage in S&P 500.**

Time series chart of S&P 500

Plotting a time series chart of the S&P 500 index as shown in figure 2 provides a visual representation of the historical performance and trends of the stock market. This visualization is crucial for making informed decisions, identifying market cycles, detecting potential opportunities or risks, and assessing the effectiveness of investment strategies

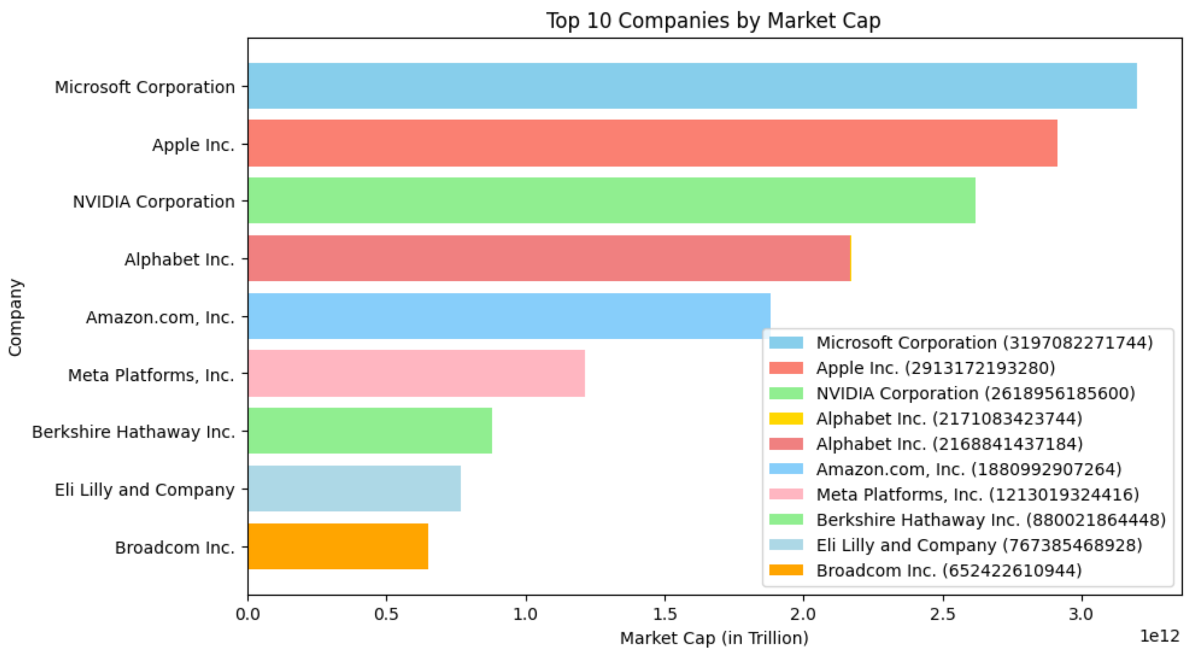


Figure 1: Top 10 Companies in Market Cap

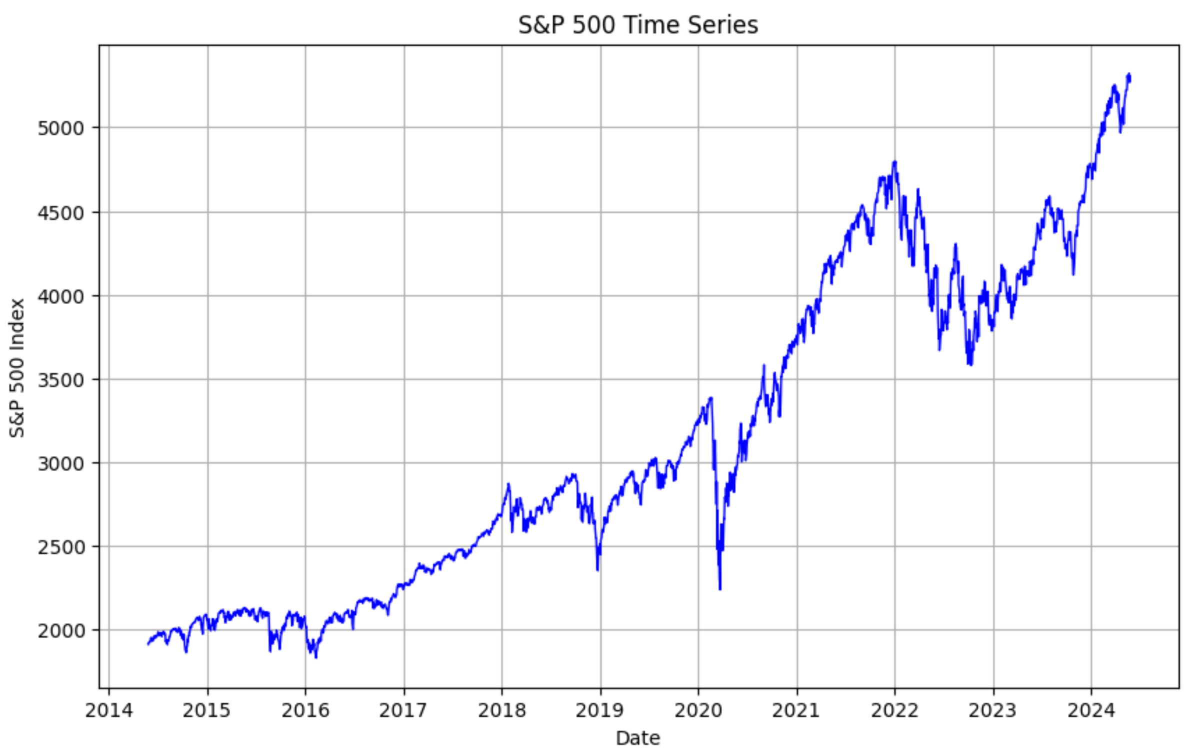


Figure 2: Time series chart of S&P 500

Correlation between S&P 500 and its top 10 companies by Market Cap

The S&P 500 is a benchmark index for the U.S. stock market, comprising 500 large-cap companies. The correlation between the S&P 500 and its top 10 companies by market capitalization shows how closely the index tracks the performance of these major players.

- **Influence of Large Companies:** The top 10 companies by market cap are often industry leaders with significant market influence. Their stock performance can drive the S&P 500 index due to their substantial weight in the index.
- **Market Indicator:** The S&P 500 is used as a benchmark for the U.S. stock market's performance. A high correlation with the top 10 companies suggests that these companies are good indicators of overall market trends.

Figure 3 shows that Google/Alphabet, Microsoft, Apple have highest correlation whereas Nvidia has the least correlation.

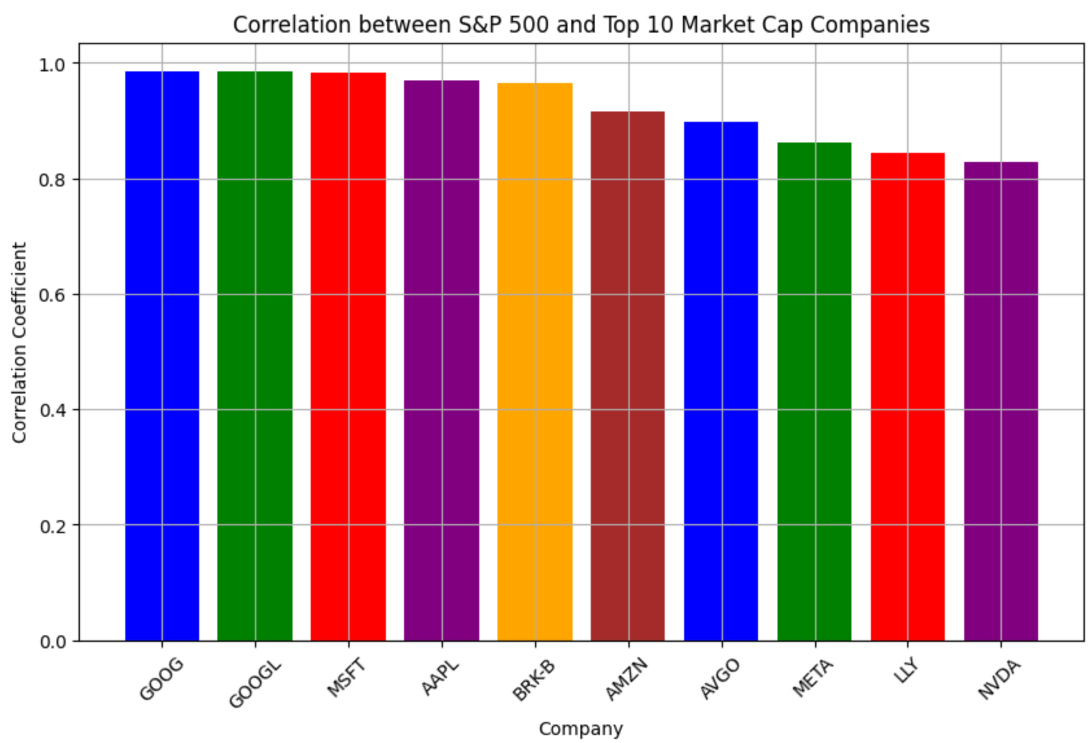
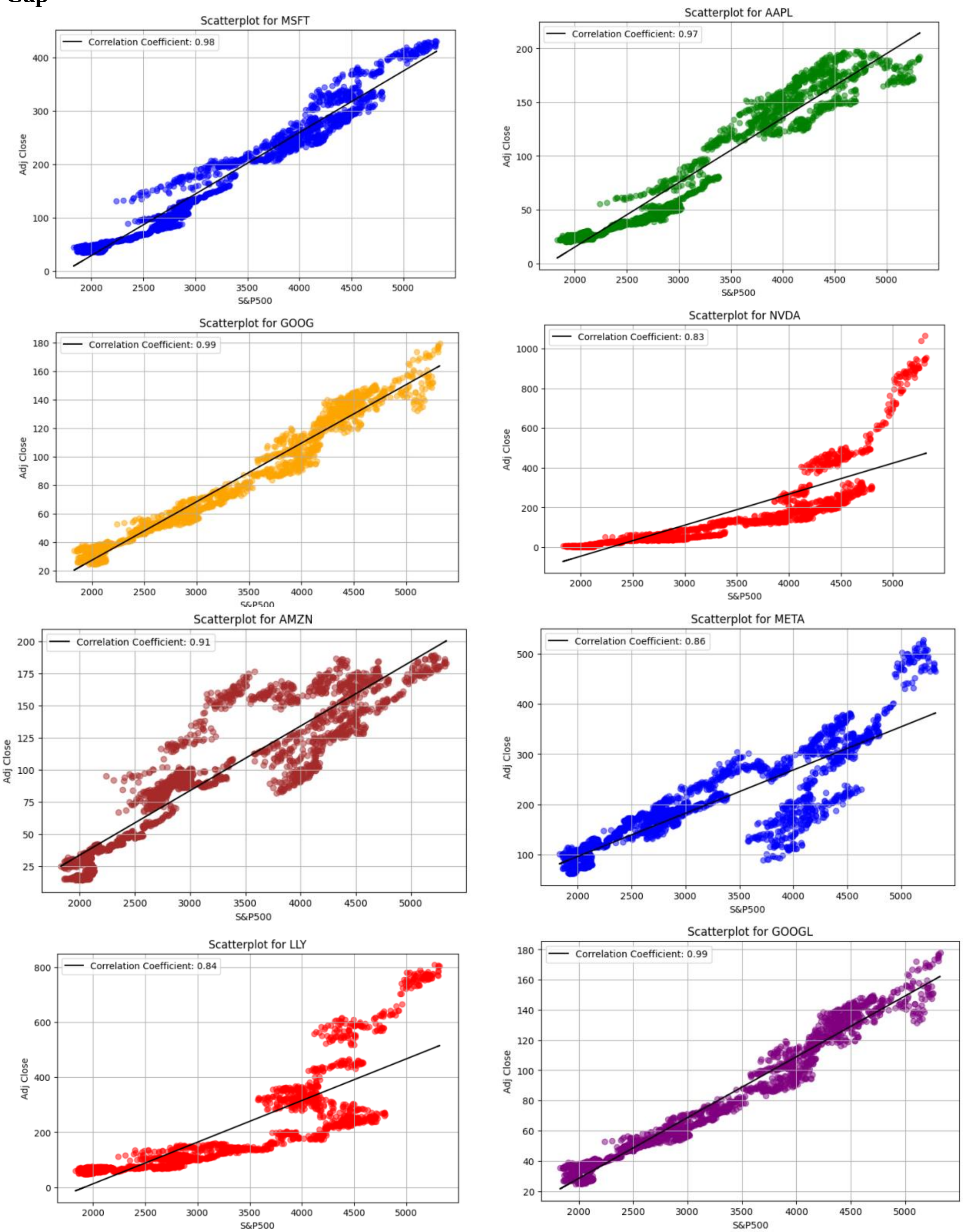


Figure 3: Correlation Coefficient - S&P 500 and its top 10 companies by Market Cap

Scatter plots to show correlation between S&P 500 and its top 10 companies by Market Cap



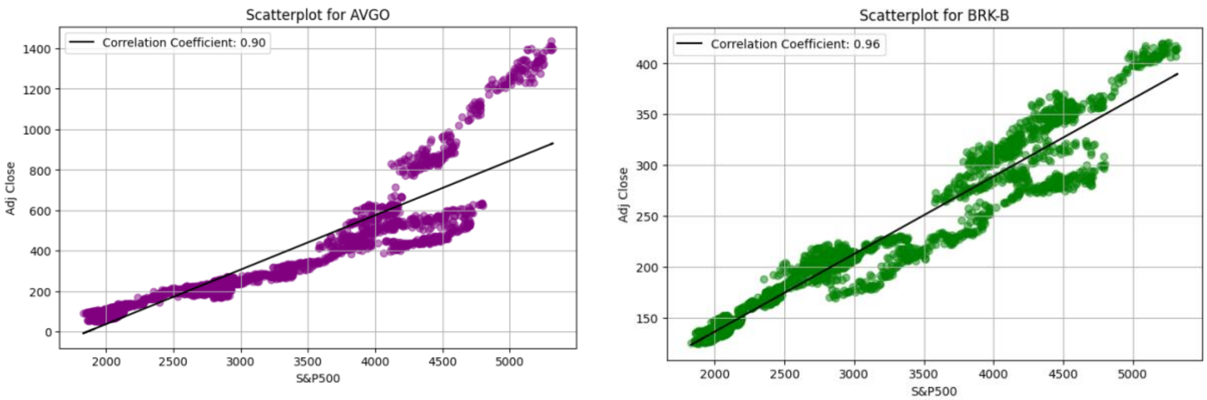


Figure 4: Scatter plots - S&P 500 and its top 10 companies by Market Cap

Scatter plots confirm that Google/Alphabet, Microsoft, Apple have highest correlation whereas Nvidia has the least correlation.

5. Machine Learning Models

After looking at the correlation between S&P 500 index price and the adj close price (stock price) for the top 10 stocks. We try to perform stock price prediction using machine learning models like linear regression, Lasso, Ridge, Random Forest Regressor, Gradient Boosting Regressor, Bagging Regressor.

We have done it in following steps.

- Data Preparation and Cleaning
- Feature engineering
- Data Scaling and splitting
- Model Training
- Evaluation and Visualization

Data Preparation and Cleaning

Data frames are prepared and processed so that they can be used in the models.

- Date Conversion : Date columns in the stock and index dataframes are converted to datetime objects for time series.
- Since df_index data has dates starting from 2010 and df_stocks has dates starting from 2014 i have eliminated the additional dates in df_index.
- Stock Symbol Selection: While selecting the stock symbol, i have chossen Microsoft. As it is the company with highest market cap.
- The Dataframes stocks, index and company are merged and all the NA values are dropped

Feature Engineering

Below features were selected in order to predicr the stock price -

- Moving Average and Standard deviation : The Moving averages and standard deviations over 20-day and 5-day is used to account for short-term and medium-term trends and volatility.
- Percentage Change: It computes the daily change in stock price as percentage change, providing a measure of daily price movement.
- Market_cap to EBITDA Ratio: The MCap_EBITDA_Ratio is calculated, which is useful to find the value of the company relative to its earnings before interest, taxes, depreciation, and amortization.
- S&P 500 Index: This is used to capture the impact of movement in S&P 500 on stock price of the chossen company
- Moving Average Convergence Divergence(MACD): The 26-day and 12-day Exponential moving average (EMA) are used to compute the Moving Average Convergence

Divergence (MACD), an indicator to help identify price trends, measure trend momentum, often used to gauge the momentum of stock prices. (Investopedia, n.d.)

- Volatility: Measures the volatility over a 30-day window as the ratio of the standard deviation to the mean of 'Adj Close', providing a normalized measure of price variability.
- Signal Line: The Signal Line is a 9-day EMA of the MACD. It serves as a trigger for buy and sell signals
- MACD Histogram: The MACD Histogram is the difference between the MACD line and the Signal Line. It represents the distance between the MACD and its 9-day EMA (Signal Line)

Data Scaling and splitting

- Data Scaling: Before modeling, features are scaled using StandardScaler to ensure they contribute equally to the model's performance, important for models like Ridge and Lasso regression that are sensitive to the scale of input data.StandardScaler ensures that all variables contribute equally to the analysis by converting their values into z-scores.
- Split Data: The data is split into training and testing sets (70% training, 30% testing). This split is essential for training the models on one set of data and testing them on unseen data to evaluate their predictiveness.

Model Training

Six modelling techniques were used to predict the stock price.

- Linear Regression: Linear Regression is fundamental statistical model for modeling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. I have used Linear Regression to predict the 'Adj Close' price of a stock based on various technical indicators and financial ratios.

- Ridge Regression: Ridge Regression is like linear regression, designed to make predictions more reliable. Ridge uses a technique where the penalty is based on the square of each features magnitude. This doesn't allow any of the feature to go to zero (like in Lasso). Ridge Regression is applied with an alpha value of 1.0, which controls the regularization strength.
- Lasso Regression : Lasso Regression ensures no single factor feature in data shouts too loudly. It does this by applying penalty to each feature based on its influence on prediction. The bigger the influence, the bigger the penalty. If a feature's influence is too small, Lasso drops it to zero or ignores that feature completely. I have used alpha value of 0.1, suggesting moderate regularization. Since there are many features it can identify and eliminate non-informative features, simplifying the model and potentially improving generalization performance.
- Random Forest Regression: Random Forest is an ensemble learning method that combines multiple decision trees to improve predictive performance and control overfitting. Each tree in the forest is built on a random subset of the data and features. Random Forest is optimized using GridSearchCV to find the best combination of n_estimators (number of trees) and max_depth (maximum depth of each tree). This ensures the model achieves optimal performance by tuning its hyperparameters.
- Gradient Boosting Regression: Gradient Boosting is an ensemble technique that builds models sequentially, with each new model trying to correct the errors made by the previous ones. It combines weak learners (typically decision trees) into a strong learner. GridSearchCV is used to optimize the GradientBoostingRegressor by searching for the

best combination of n_estimators, learning_rate, and max_depth. This tuning process helps in balancing the model's complexity and predictive power.

- Bagging Regression: Bagging is an ensemble technique that improves the stability and accuracy of machine learning algorithms by training multiple models on different subsets of the data and averaging their predictions. Bagging uses GridSearchCV to determine the optimal number of base estimators (n_estimators). Here, BaggingRegressor uses decision trees as the base estimator. This helps in leveraging multiple weak learners to form a robust model.

Model Evaluation

The metrics used for evaluation of the models are Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2). The results are shown in figure 5 below

Model Comparison Analysis

Model Comparison

Model	MSE	MAE	RMSE	R^2
Linear Regression	4.851047191158804	1.7078036380624355	2.2025092942275646	0.9983538774941924
Lasso	8.040553676831465	2.2126127513114624	2.8355870074521543	0.9972715713030561
Ridge	10.849579909554322	2.5204955099177733	3.2938700504959697	0.9963183747830312
Random Forest	6336.455445513574	61.015472945284195	79.60185579189454	-1.1501711908549477
Gradient Boosting	5888.704642845094	57.42881456702515	76.7378957415767	-0.9982343730459318
Bagging	6381.04456978083	61.489139860839856	79.88144070922125	-1.1653017715477842

Figure 5: Model Comparison Analysis

Explanation of the comparison

- **Linear Regression**:
  - **MSE**: 4.85 **MAE**: 1.71 **RMSE**: 2.20 **R^2**: 0.998
  - **Interpretation**: Linear Regression performs very well with a high R^2 value close to 1, indicating it explains most of the variance in the data. The errors (MSE, MAE, RMSE) are relatively low.
  - **Linear Regression Latest Prediction for next day: Predicted: 429.97 vs Actual: 430.16**

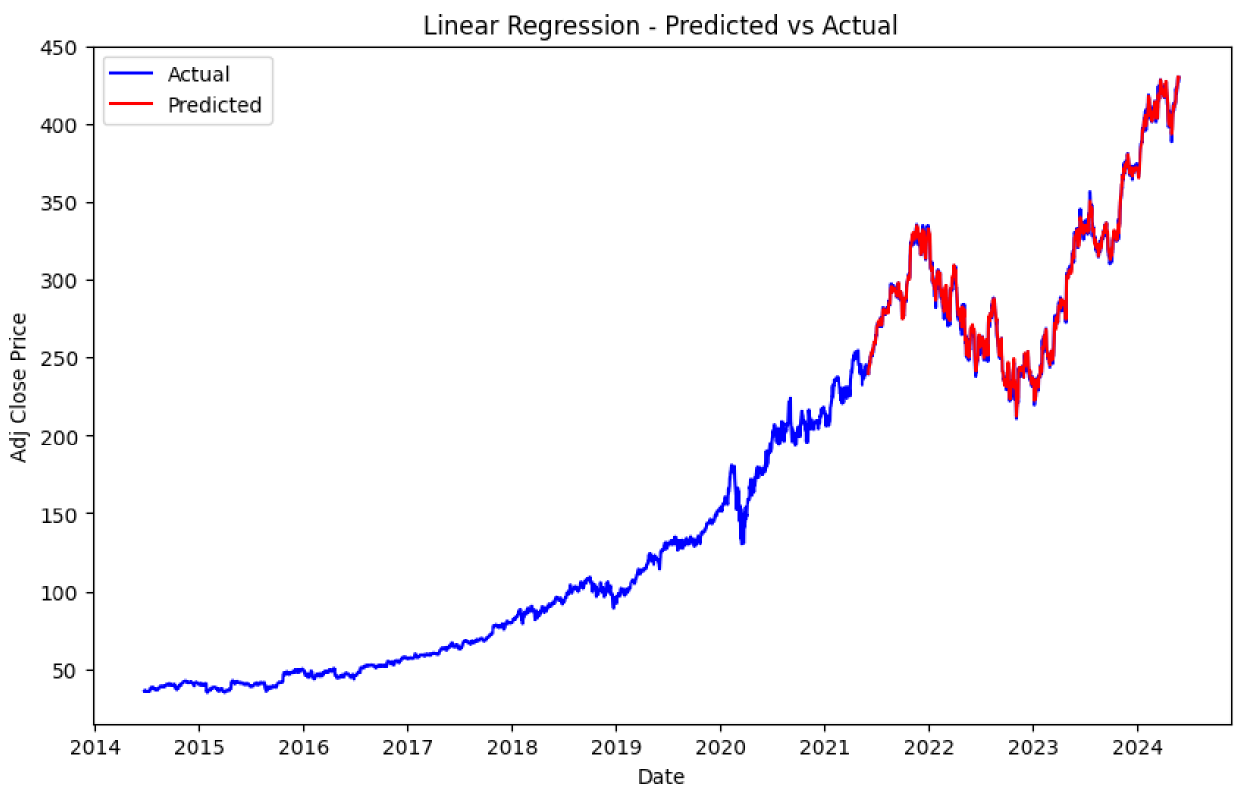


Figure 6: Linear Regression Predicted vs Actual

- **Lasso**:
    - **MSE**: 8.04 **MAE**: 2.21 **RMSE**: 2.84 **R^2**: 0.997
    - **Interpretation**: Lasso regression also performs well, though slightly worse than Linear Regression. It introduces some regularization, which can help in reducing overfitting.
    - **Lasso Regression Latest Prediction for next day: Predicted: 429.62 vs Actual: 430.16**

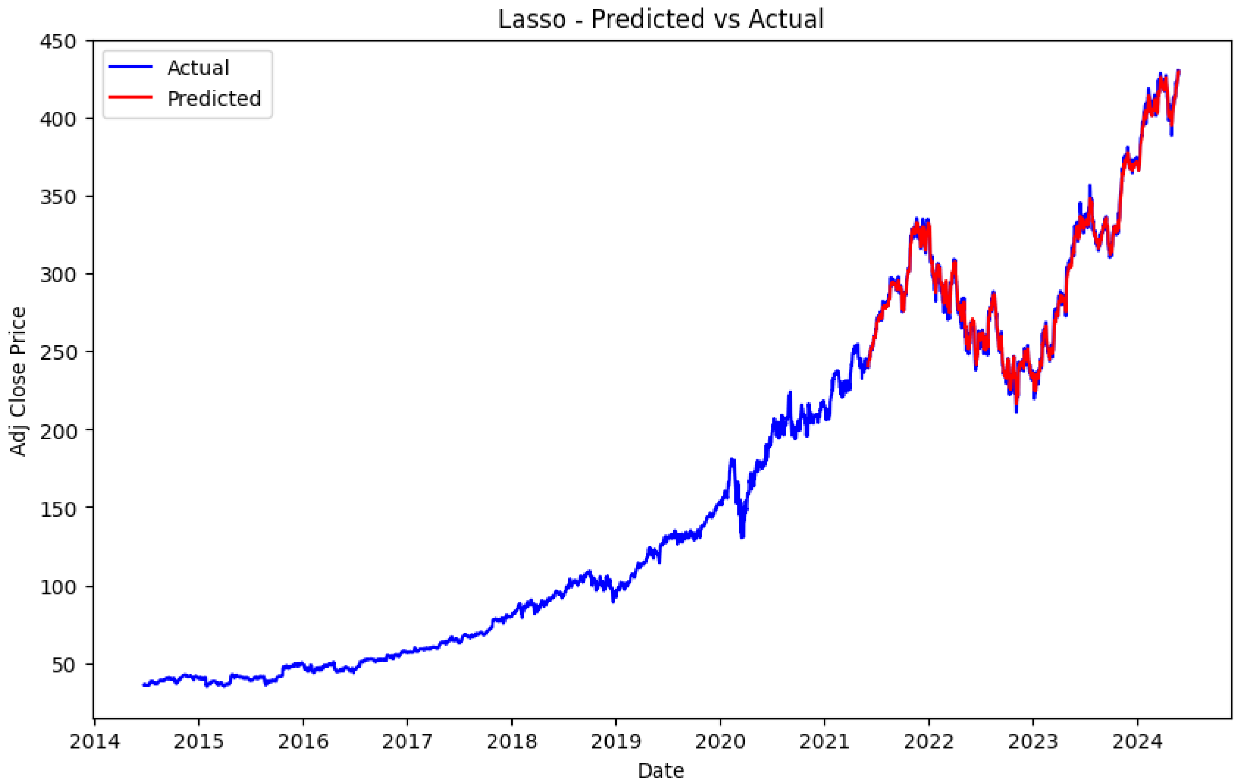


Figure 7: Lasso Regression Predicted vs Actual

- **Ridge**:
  - **MSE**: 10.85 **MAE**: 2.52 **RMSE**: 3.29 **R^2**: 0.996
  - **Interpretation**: Ridge regression has higher error values compared to Linear Regression and Lasso, but still maintains a high R^2 value.
  - **Ridge Regression Latest Prediction for next day: Predicted: 430.55 vs Actual: 430.16**

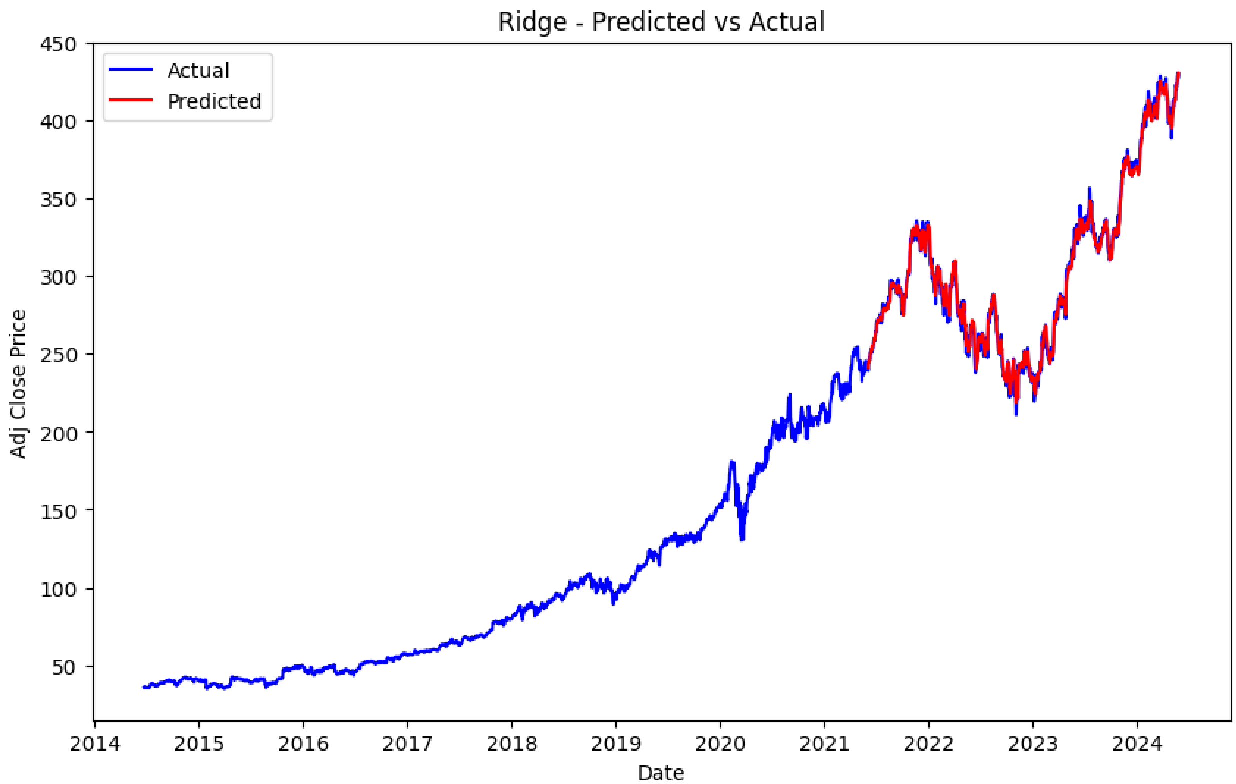


Figure 8: Ridge Regression Predicted vs Actual

- **Random Forest**:
  - **MSE**: 6336.46 **MAE**: 61.02 **RMSE**: 79.60 **R^2**: -1.15
  - **Interpretation**: Random Forest performs poorly in this case, with very high error values and a negative R^2, indicating it is not a good fit for the data.

- **Random Forest Latest Prediction for next day: Predicted: 250.80 vs Actual: 430.16**

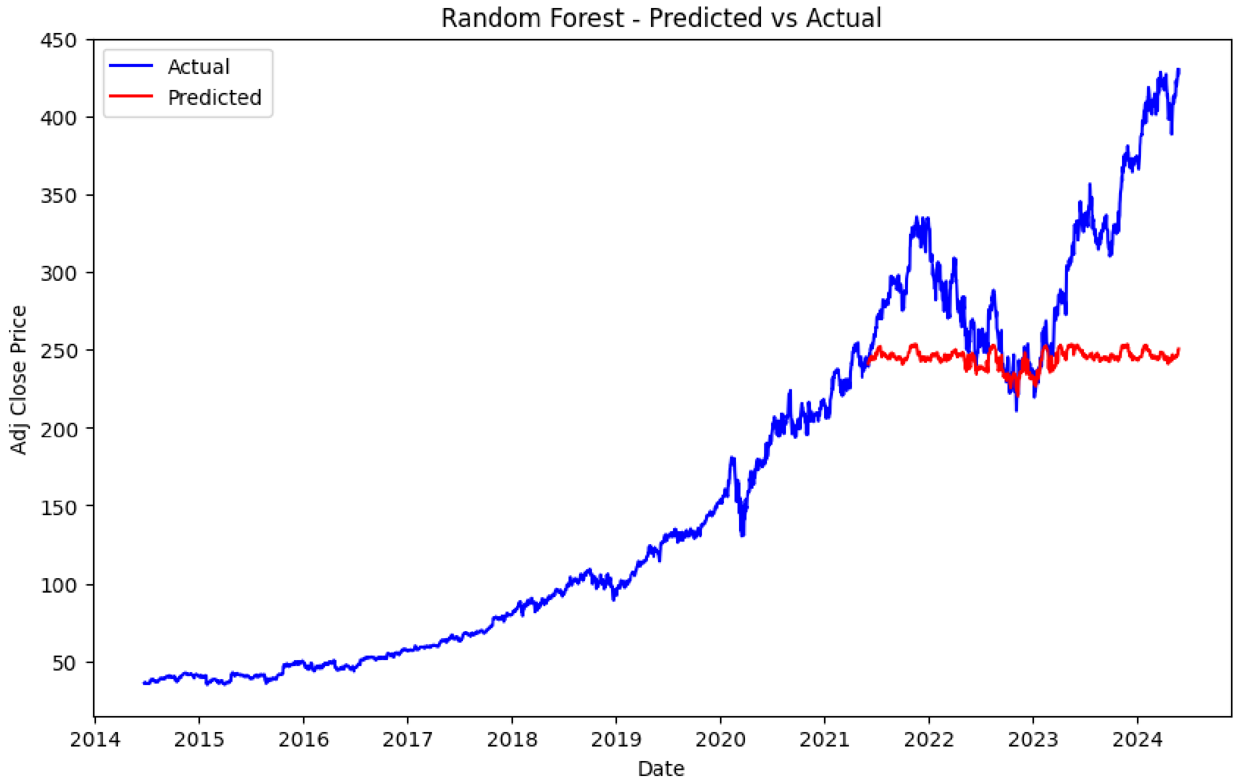


Figure 9: Random Regression Predicted vs Actual

- **Gradient Boosting**:
  - **MSE**: 5888.70 **MAE**: 57.43 **RMSE**: 76.74 **R^2**: -0.99
  - **Interpretation**: Gradient Boosting also performs poorly, similar to Random Forest, with high errors and a negative R^2.
  - **Gradient Boosting Latest Prediction for next day: Predicted: 253.80 vs Actual: 430.16**

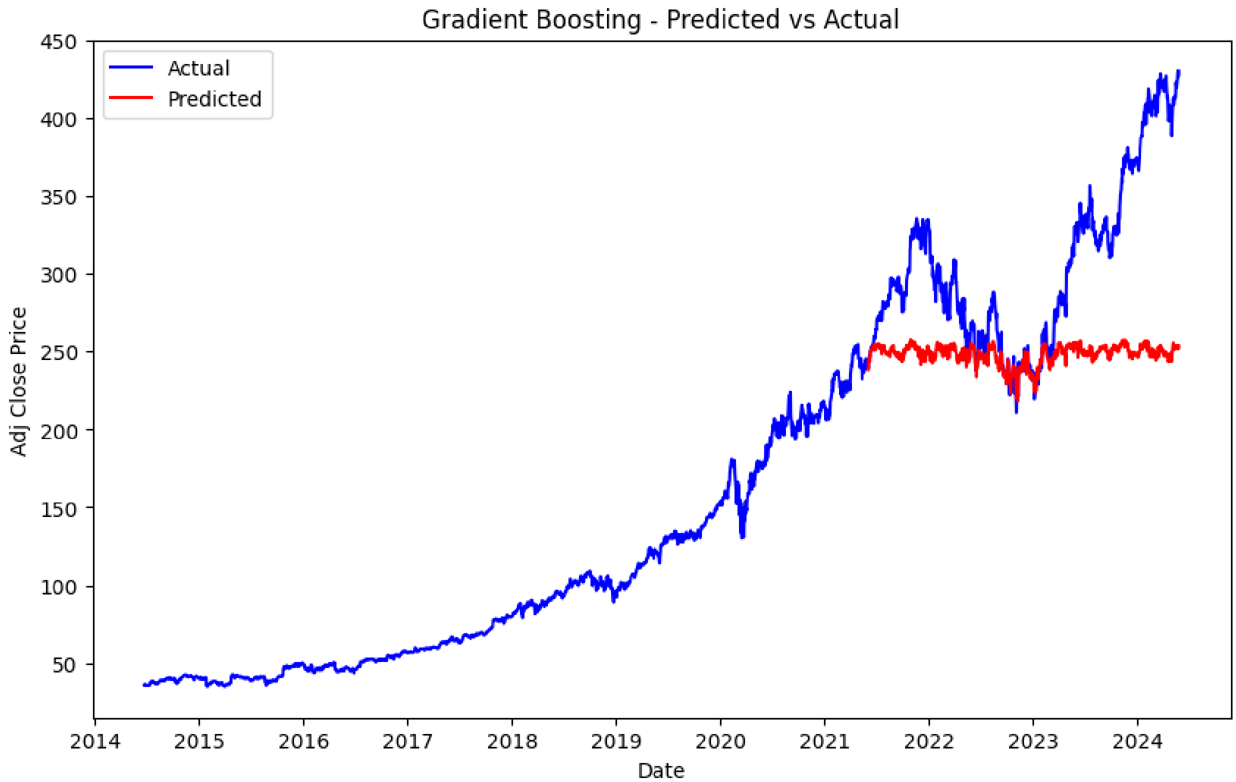


Figure 10: Gradient Boosting Predicted vs Actual

- **Bagging**:
  - **MSE**: 6381.04 **MAE**: 61.49 **RMSE**: 79.88 **R^2**: -1.17
  - **Interpretation**: Bagging shows poor performance with very high error values and a negative R^2.
  - **Bagging Latest Prediction for next day: Predicted: 250.76 vs Actual: 430.16**

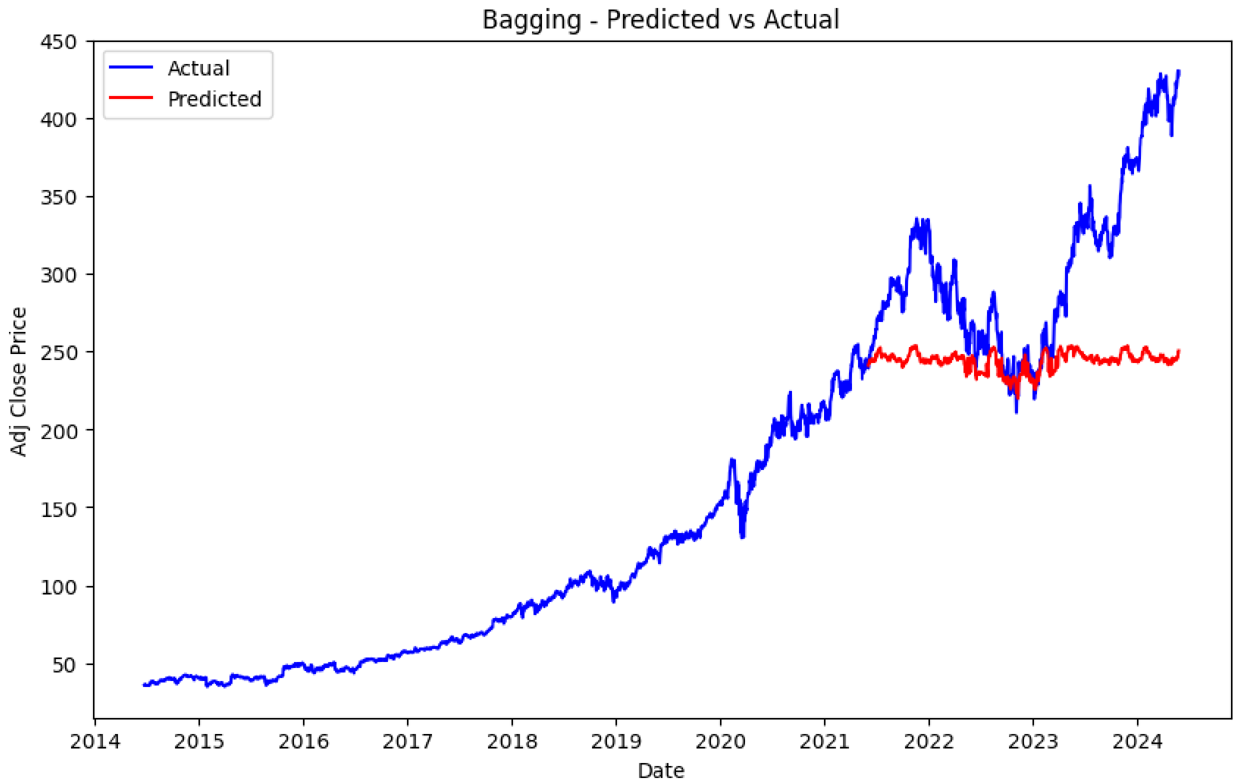


Figure 11: Bagging Predicted vs Actual

Summary

- **Linear Regression, Lasso, and Ridge**: These models perform well, with Linear Regression being the best, followed by Lasso and Ridge. Their high R^2 values and low error metrics indicate good model fit and prediction accuracy.
- **Random Forest, Gradient Boosting, and Bagging**: These methods perform poorly to predict stock price, indicated by high error metrics and negative R^2 values. This suggests that these models are not well-suited for this dataset, potentially due to overfitting or the nature of the data.

Important Features across models –

Summary

- 20_day_avg and 5_day_avg consistently show high importance across all models, particularly in ensemble methods like Random Forest, Gradient Boosting, and Bagging.
- MACD_histogram, MACD, Return, and SP500_index: Generally have some importance but vary by model.
- volatility: Has a small or negligible impact in most models.
- Currentprice and MCap_EBITDA_Ratio: Typically have no impact, with coefficients or importances near zero.

```
Feature importances for Linear Regression:
                     Coefficient
20_day_avg          1.949011e+02
5_day_avg          -8.425355e+01
MACD_histogram      7.492532e+00
MACD                3.821931e+00
Signal_line         1.535369e+00
Return              7.942355e-01
SP500_index        -7.697443e-02
20_day_std         -4.822204e-02
5_day_std          -3.564401e-02
volatility         -1.912720e-02
Currentprice        1.293188e-12
MCap_EBITDA_Ratio -1.111999e-12
```

```
Feature importances for Lasso:
                    Coefficient
20_day_avg           109.781583
MACD_histogram         4.358610
MACD                   2.780092
Return                 0.882448
5_day_avg              0.578317
SP500_index            0.454769
20_day_std             0.000000
5_day_std              0.000000
MCap_EBITDA_Ratio      0.000000
Currentprice           0.000000
Signal_line            0.000000
volatility            -0.000000
```

```
Feature importances for Ridge:
                    Coefficient
20_day_avg            57.882586
```

```
5_day_avg            50.565499
MACD_histogram        3.167334
SP500_index           2.035418
MACD                  1.232867
Return                1.085007
20_day_std            0.573110
5_day_std             0.387638
Signal_line           0.239246
volatility           -0.220580
MCap_EBITDA_Ratio     0.000000
      Currentprice            0.000000
```

```
Feature importances for Random Forest:
                   Importance
5_day_avg            0.805748
20_day_avg           0.192089
SP500_index          0.001123
MACD_histogram       0.000280
Return               0.000248
MACD                 0.000176
5_day_std            0.000098
20_day_std           0.000091
volatility           0.000075
Signal_line          0.000073
MCap_EBITDA_Ratio    0.000000
      Currentprice           0.000000
```

```
Feature importances for Gradient Boosting:
                   Importance
5_day_avg            0.523522
20_day_avg           0.471821
SP500_index          0.003825
Return               0.000412
MACD_histogram       0.000230
5_day_std            0.000079
Signal_line          0.000035
MACD                 0.000032
volatility           0.000025
20_day_std           0.000019
MCap_EBITDA_Ratio    0.000000
      Currentprice           0.000000
```

```
Feature importances for Bagging:
                   Importance
5_day_avg            0.779951
20_day_avg           0.217732
SP500_index          0.001242
MACD_histogram       0.000284
Return               0.000247
MACD                 0.000203
5_day_std            0.000097
20_day_std           0.000087
volatility           0.000083
Signal_line          0.000075
```

```
MCap_EBITDA_Ratio    0.000000
Currentprice         0.000000
```

6. Conclusion

Primary Model:

For accurate and reliable predictions, **Linear Regression** is recommended as the primary model due to its strong performance metrics, simplicity, and high interpretability.

Feature Focus:

Emphasizing key features like **5_day_avg** and **20_day_avg** can further enhance model performance. Consider simplifying the model by excluding less impactful features such as **volatility**, **Currentprice**, and **MCap_EBITDA_Ratio**.

Alternative Models:

Lasso Regression can be a suitable alternative if model simplicity and interpretability are prioritized, especially when dealing with numerous features. It effectively performs feature selection and can handle multicollinearity.

Extended Recommendations:

Accuracy and Interpretability: Using **Linear Regression** not only for its accuracy but also for its interpretability is highly recommended. Its straightforward nature allows for easy explanation and understanding of the relationships within the data.

Incorporate Macroeconomic Indicators: Adding macroeconomic indicators (e.g., GDP growth rates, interest rates) can provide additional context and improve the model's predictive power.

Sentiment Analysis: Integrating sentiment analysis of market news and social media can enhance performance by capturing market sentiment and investor behavior.

Regular Updates: Regularly updating models with new data ensures that they remain relevant and accurate over time, adapting to changing market conditions.

Explore LSTM Models: Exploring Long Short-Term Memory (LSTM) models is recommended due to their ability to capture temporal dependencies. LSTMs are particularly beneficial for time series forecasting in financial markets, as they can effectively model trends and patterns over time.

By implementing these recommendations, the overall predictive accuracy and robustness of the models can be significantly improved, leading to better decision-making and insights in financial market forecasting.

Bibliography

Investopedia. (n.d.). *Investopedia*. Retrieved from Investopedia.com: https://www.investopedia.com/terms/m/macd.asp#:~:text=What%20MACD%20Signals-,The%20MACD%20line%20is%20calculated%20by%20subtracting%20the%2026%2Dperiod,for%20buy%20or%20sell%20signals

LARXEL. (n.d.). *S&P 500 Stocks (daily updated)*. Retrieved from Kaggle: https://www.kaggle.com/datasets/andrewmvd/sp-500-stocks?resource=download

stockanalysis. (n.d.). *stockanalysis*. Retrieved from stockanalysis.com: https://stockanalysis.com/stocks/wdc/

Appendix

```
import pandas as pd
df_company = pd.read_csv('/kaggle/input/sp-500-
stocks/sp500_companies.csv')
df_index = pd.read_csv('/kaggle/input/sp-500-stocks/sp500_index.csv')
df_stocks = pd.read_csv('/kaggle/input/sp-500-stocks/sp500_stocks.csv')

df_company.info()
print("\n")
df_index.info()
print("\n")
df_stocks.info()
print("\n")
df_stocks



df_company = df_company.drop(['State'], axis=1)
df_company = df_company.drop(['Fulltimeemployees'], axis=1)
df_company


df_company[df_company['Revenuegrowth'].isnull()]


df_company.loc[df_company['Symbol']== 'WDC', 'Revenuegrowth']= [-0.1597]
# df_company.loc[df_company['Symbol']== 'PM', 'Revenuegrowth']= [0.1221]
df_company[df_company['Revenuegrowth'].isnull()]


import matplotlib.pyplot as plt

sorted_data = df_company.sort_values(by='Marketcap', ascending=False)
top_10_data = sorted_data.head(10)

companies = top_10_data['Longname']
symbols = top_10_data['Symbol']
market_caps = top_10_data['Marketcap']


colors = ['skyblue', 'salmon', 'lightgreen', 'gold', 'lightcoral',
'lightskyblue', 'lightpink', 'lightgreen', 'lightblue', 'orange']
```

```
plt.figure(figsize=(10, 6))
bars = plt.barh(companies, market_caps, color=colors)
plt.xlabel('Market Cap (in Trillion)')
plt.ylabel('Company')
plt.title('Top 10 Companies by Market Cap')
plt.gca().invert_yaxis()

legend_labels = [f'{companies[i]} ({market_caps[i]})' for i in
range(len(companies))]
plt.legend(bars, legend_labels, loc='lower right')

plt.show()

top_10_data.to_csv('top_10_companies_by_marketcap.csv', index=False)


import matplotlib.pyplot as plt


df_index['Date'] = pd.to_datetime(df_index['Date'])

df_index.set_index('Date', inplace=True)

plt.figure(figsize=(10, 6))
plt.plot(df_index.index, df_index['S&P500'], color='blue', linewidth=1)
plt.title('S&P 500 Time Series')
plt.xlabel('Date')
plt.ylabel('S&P 500 Index')
plt.grid(True)
plt.show()



import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

df_index.reset_index(inplace=True)

# Convert 'Date' columns to datetime objects
df_stocks['Date'] = pd.to_datetime(df_stocks['Date'])

# Merge the two datasets on the 'Date' column
merged_df = pd.merge(df_index, df_stocks, on='Date', how='inner')
```

```
# Assuming 'symbols' is defined, e.g., symbols = ['MSFT', 'AAPL']
filtered_df = merged_df[merged_df['Symbol'].isin(symbols)]

# Calculate correlation
correlation = filtered_df.groupby('Symbol')[['S&P500', 'Adj
Close']].corr().iloc[0::2, -1].reset_index(level=1, drop=True)
print("Correlation between S&P 500 and top 10 market cap companies:")
print(correlation)

# Define a list of colors to cycle through
colors = ['blue', 'green', 'red', 'purple', 'orange', 'brown']

# Plotting scatterplots for each symbol
for index, symbol in enumerate(symbols):
    symbol_data = merged_df[merged_df['Symbol'] == symbol]
    correlation_coefficient = symbol_data['S&P500'].corr(symbol_data['Adj
Close'])

    plt.figure(figsize=(8, 5))
    plt.scatter(symbol_data['S&P500'], symbol_data['Adj Close'],
alpha=0.5, color=colors[index % len(colors)])

    # Add correlation line
    x = symbol_data['S&P500']
    y = symbol_data['Adj Close']
    m, b = np.polyfit(x, y, 1)
    plt.plot(x, m*x + b, color='black', label=f'Correlation Coefficient:
{correlation_coefficient:.2f}')  # Using black for the line

    plt.title(f'Scatterplot for {symbol}')
    plt.xlabel('S&P500')
    plt.ylabel('Adj Close')
    plt.legend()
    plt.grid(True)
    plt.show()

# Sorting the correlation dataframe in descending order
correlation_df = correlation_df.sort_values(by='Adj Close',
ascending=False)

# Plotting the correlation coefficients in a bar chart
plt.figure(figsize=(10, 6))
plt.bar(correlation_df['Symbol'], correlation_df['Adj Close'],
color=colors[:len(correlation_df)])
plt.xlabel('Company')
```

```
plt.ylabel('Correlation Coefficient')
plt.title('Correlation between S&P 500 and Top 10 Market Cap Companies')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor, BaggingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Convert Date to datetime and filter for a specific symbol
df_stocks['Date'] = pd.to_datetime(df_stocks['Date'])
df_index['Date'] = pd.to_datetime(df_index['Date'])
df_company['Symbol'] = df_company['Symbol'].astype(str)

# Determine the common date range
start_date = max(df_stocks['Date'].min(), df_index['Date'].min())
end_date = min(df_stocks['Date'].max(), df_index['Date'].max())

# Filter df_stocks and df_index within the overlapping date range
df_stocks_filtered = df_stocks[(df_stocks['Symbol'] == 'MSFT') &
(df_stocks['Date'] >= start_date) & (df_stocks['Date'] <= end_date)]
df_index_filtered = df_index[(df_index['Date'] >= start_date) &
(df_index['Date'] <= end_date)]

# Merge stocks with company data
df_merged = pd.merge(df_stocks_filtered, df_company, on='Symbol',
how='left')

# Merge with the filtered S&P 500 index data
df_merged = pd.merge(df_merged, df_index_filtered, on='Date', how='left')

# Handling missing S&P500 index values (if any)
df_merged['S&P500'].fillna(method='ffill', inplace=True)
```

```
# Feature Engineering
df_merged['20_day_avg'] = df_merged['Adj Close'].rolling(window=20).mean()
df_merged['20_day_std'] = df_merged['Adj Close'].rolling(window=20).std()
df_merged['5_day_avg'] = df_merged['Adj Close'].rolling(window=5).mean()
df_merged['5_day_std'] = df_merged['Adj Close'].rolling(window=5).std()
df_merged['Return'] = df_merged['Adj Close'].pct_change()
df_merged['MCap_EBITDA_Ratio'] = df_merged['Marketcap'] /
df_merged['Ebitda']
df_merged['SP500_index'] = df_merged['S&P500'].shift(1)
df_merged['26_ema'] = df_merged['Adj Close'].ewm(span=26,
adjust=False).mean()
df_merged['12_ema'] = df_merged['Adj Close'].ewm(span=12,
adjust=False).mean()
df_merged['MACD'] = df_merged['12_ema'] - df_merged['26_ema']
df_merged['Signal_line'] = df_merged['MACD'].ewm(span=9,
adjust=False).mean()
df_merged['MACD_histogram'] = df_merged['MACD'] - df_merged['Signal_line']
df_merged['volatility'] = df_merged['Adj Close'].rolling(window=20).std()
/ df_merged['Adj Close'].rolling(window=20).mean()

df_merged.sort_values('Date', inplace=True)

# All NAs are dropped
df_merged.dropna(inplace=True)

# Select features and target
features = ['20_day_avg', '20_day_std','5_day_avg','5_day_std','Return',
'MCap_EBITDA_Ratio', 'Currentprice', 'SP500_index', 'MACD', 'Signal_line',
'MACD_histogram', 'volatility']
target = 'Adj Close'

# Scaling the full dataset for model evaluation
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_merged[features])
y = df_merged[target]

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.3, shuffle=False, random_state=42)

# Initialize and train models
models = {
    'Linear Regression': LinearRegression(),
    'Lasso': Lasso(alpha=0.1, max_iter=10000),
    'Ridge': Ridge(alpha=1.0),
```

```
    'Random Forest': GridSearchCV(RandomForestRegressor(random_state=42),
param_grid={'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20]},
cv=5, n_jobs=-1),
    'Gradient Boosting':
GridSearchCV(GradientBoostingRegressor(random_state=42),
param_grid={'n_estimators': [50, 100, 200], 'learning_rate': [0.01, 0.1,
0.2], 'max_depth': [3, 5, 7]}, cv=5, n_jobs=-1),
    'Bagging':
GridSearchCV(BaggingRegressor(base_estimator=DecisionTreeRegressor(),
random_state=42), param_grid={'n_estimators': [50, 100, 200]}, cv=5,
n_jobs=-1),
    # 'K-Nearest Neighbors': GridSearchCV(KNeighborsRegressor(),
param_grid={'n_neighbors': [3, 5, 7, 9]}, cv=5, n_jobs=-1)
}

# Prepare DataFrame for storing predictions
df_predictions = df_merged.iloc[len(X_train):].copy()

# DataFrame for storing evaluation metrics
metrics = pd.DataFrame(columns=['Model', 'MSE', 'MAE', 'RMSE', 'R^2'])

# Fit models and save predictions
for name, model in models.items():
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    df_predictions[name + ' Predictions'] = predictions

    mse = mean_squared_error(y_test, predictions)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)

    metrics = pd.concat([metrics, pd.DataFrame({'Model': [name], 'MSE':
[mse], 'MAE': [mae], 'RMSE': [rmse], 'R^2': [r2]})], ignore_index=True)

# Ensure correct merging by checking column names
prediction_columns = [name + ' Predictions' for name in models.keys()]
df_merged_final = pd.merge(df_merged, df_predictions[['Date'] +
prediction_columns], on='Date', how='left')

# Export to CSV
df_merged_final.to_csv('stock_predictions.csv', index=False)
metrics.to_csv('model_metrics.csv', index=False)

# Plot the comparative table
```

```
fig, ax = plt.subplots(figsize=(12, 6))
ax.axis('tight')
ax.axis('off')
table = ax.table(cellText=metrics.values, colLabels=metrics.columns,
cellLoc='center', loc='center')

table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1.2, 1.2)

plt.title('Model Comparison')
plt.show()

# Fit models and print outcomes
for name, model in models.items():
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)
    print(f"{name} - MSE: {mse:.4f}, MAE: {mae:.4f}, RMSE: {rmse:.4f},
R^2: {r2:.4f}")
    print(f"{name} Latest Prediction for next day: Predicted:
{predictions[-1]:.2f}, Actual: {y_test.iloc[-1]:.2f}")

    # Plotting predictions vs actual values
    plt.figure(figsize=(10, 6))
    plt.plot(df_merged['Date'], df_merged['Adj Close'], label='Actual',
color='blue')
    plt.plot(df_merged.iloc[-len(predictions):]['Date'], predictions,
label='Predicted', color='red')
    plt.title(f"{name} - Predicted vs Actual")
    plt.xlabel('Date')
    plt.ylabel('Adj Close Price')
    plt.legend()
    plt.show()

    if hasattr(model, 'coef_'):
        # For models with coefficients
        print(f"Feature importances for {name}:")
        feature_importance = pd.DataFrame(model.coef_, index=features,
columns=['Coefficient'])
        print(feature_importance.sort_values(by='Coefficient', key=abs,
ascending=False))
```

```
    elif hasattr(model, 'best_estimator_') and
hasattr(model.best_estimator_, 'feature_importances_'):
        # For tree-based models
        print(f"Feature importances for {name}:")
        feature_importance =
pd.DataFrame(model.best_estimator_.feature_importances_, index=features,
columns=['Importance'])
        print(feature_importance.sort_values(by='Importance',
ascending=False))
    elif name == 'Bagging':
        # Calculate feature importances for Bagging
        feature_importances = np.mean([tree.feature_importances_ for tree
in model.best_estimator_.estimators_], axis=0)
        print(f"Feature importances for {name}:")
        feature_importance = pd.DataFrame(feature_importances,
index=features, columns=['Importance'])
        print(feature_importance.sort_values(by='Importance',
ascending=False))
```