Sadia Akther

Stock Price Prediction

Problem Statement:

Predicting stock prices has always been a complex yet highly valuable task in the financial industry. In this project, the goal is to build a regression and classification models that can predict the next day's closing price of a stock based on historical trading data such as open, high, low, close prices, volume, and daily returns.

Stock prices are influenced by numerous factors and are inherently noisy and volatile, making accurate prediction challenging. However, even modestly successful models can provide insights into market trends and inform better decision-making for investors, analysts, and automated trading systems.

This project seeks to evaluate several classification and regression algorithms, including Random Forest, and Gradient Boosting, k-Nearest Neighbors, Extreme Gradient Boosting and Light Gradient Boosting Machine to determine which model offers the best predictive accuracy. The goal is to identify a reliable approach for forecasting short-term price movement, with a focus on minimizing error and avoiding overfitting.

Data Wrangling:

To prepare the dataset for modeling, several data wrangling steps were performed to ensure data quality and consistency. The stock data was loaded from a CSV file and initially inspected to identify data types, missing values, and summary statistics. Duplicate rows were identified and removed based on key combinations of 'Open', 'Stock', and 'Close' prices to prevent data leakage or redundancy.

The 'Date' column was converted into datetime format for future time-based analysis. Inconsistencies in stock labels were also resolved—for instance, renaming 'FB' to 'META', since Facebook is part of META 2021.

Several new features were engineered to better capture price behavior, including 'Change' (difference between closing and opening prices), 'Loss' (price drop), and 'Average' (mean of daily high and low prices). These features were created to enhance the dataset's predictive power.

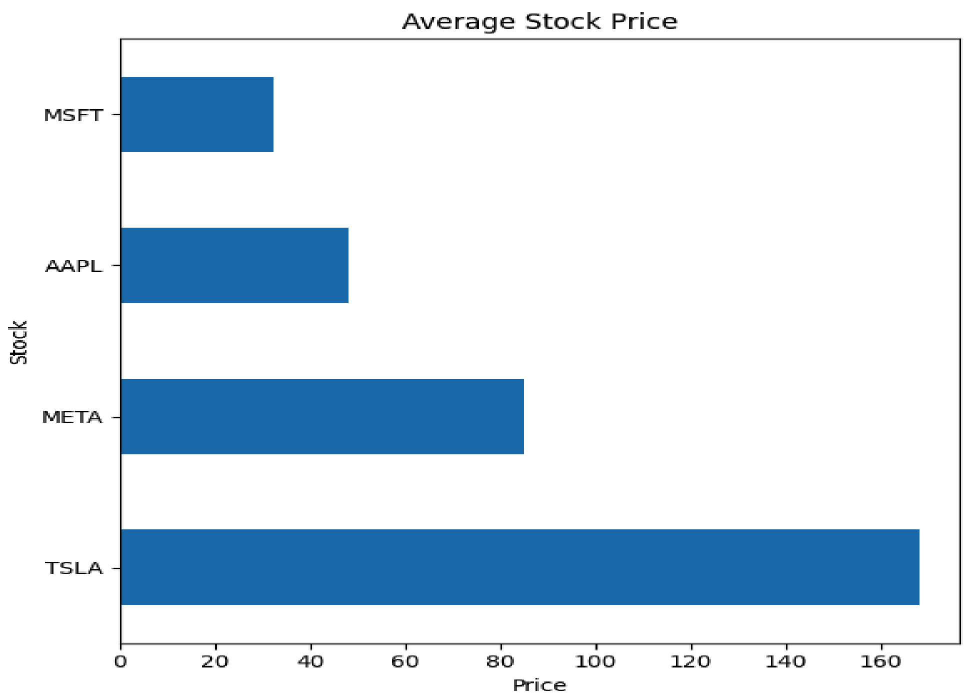

Fig 1: Average Stock Price

A bar plot was also generated to visualize and compare the average price of each stock, offering a quick glance at the trading patterns across the dataset (fig 1). By grouping the data by the 'Stock' column and calculating the mean of the 'Average' values, we gained insights into the typical trading price of each stock over the time period covered in the dataset:

1. TSLA had the highest average trading price at $168.27, indicating it was the most expensive stock in the dataset during this period.
2. META followed with an average of $85.08
3. AAPL and MSFT had lower average prices, at $48.03 and $32.46 respectively, suggesting they were the more moderately priced stocks in the dataset.

To facilitate comparison of daily high and low prices across different stocks, the dataset was reshaped using the pd.melt() function. This converted the original wide format—where High and Low prices were in separate columns—into a long format with two key columns: PriceType (indicating whether the value is a High or Low) and Price (the actual value). This transformation simplifies the process of visualizing and analyzing price distributions across stocks and time. To better understand how prices vary on a daily basis, a boxplot was created using the reshaped dataset. The High and Low prices were melted into a single column using the pd.melt() function, allowing for a more flexible structure suitable for visualization.

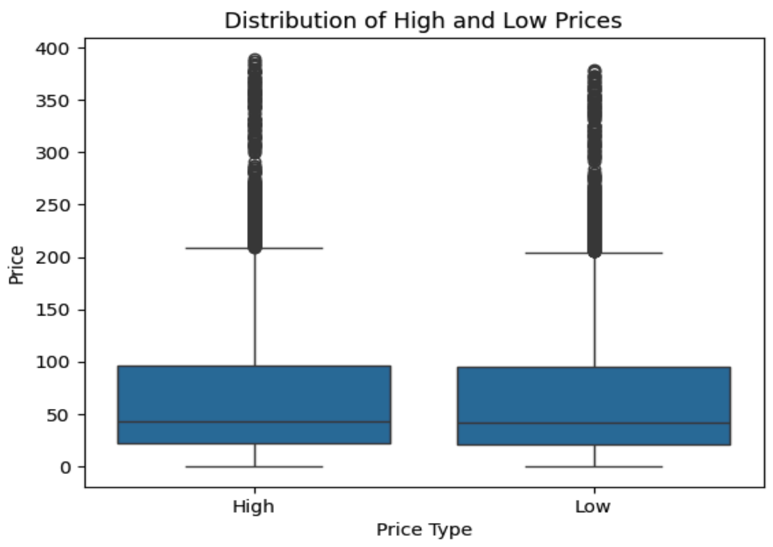

Fig 2: Distribution of high and low price

The boxplot shows that **high prices** are consistently greater than **low prices**, as expected. Further, the spread (interquartile range) and potential outliers help highlight **volatility** in daily trading. This visualization confirms that the dataset captures a realistic range of trading activity across different stock tickers. (Fig 2)

To gain deeper insights into how daily price ranges vary between different stocks, a grouped boxplot was created.

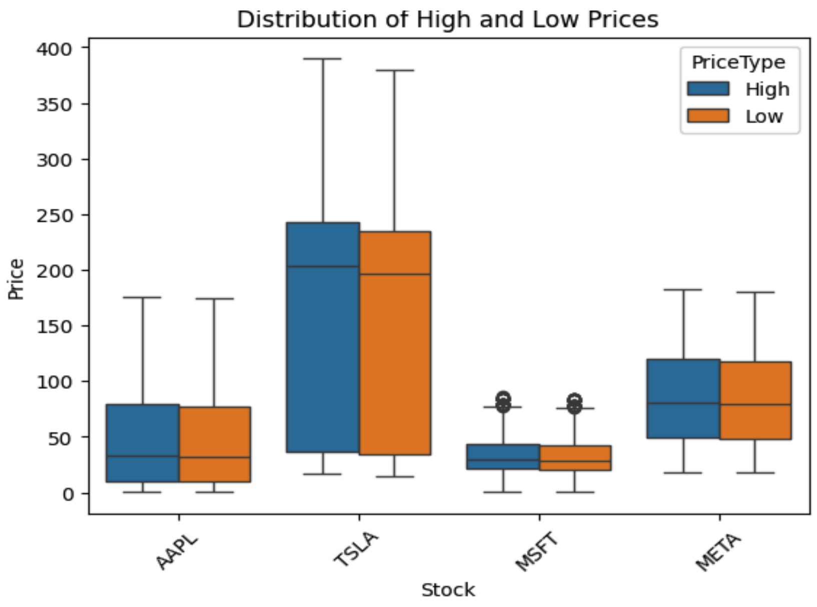

Fig 3

Each stock has two boxplots , one for High prices and one for Low prices , making it easy to compare daily trading ranges. As expected, the High prices are consistently greater than the Low prices for each stock, with a fairly stable gap between the two. This confirms the data's integrity and reflects typical intraday trading behavior — where stocks open and fluctuate, reaching a daily high and low before closing. (Fig 3)

Exploratory Data Analysis (EDA):

To begin the analysis, Principal Component Analysis (PCA) was applied to identify the most influential patterns in the dataset and reduce dimensionality. The results showed the following explained variance ratios:

- **PC1**: 1.0000 (100.00% of total variance)
- **PC2**: 0.0000 (0.00% of total variance)
- **PC3**: 0.0000 (0.00% of total variance)
- **PC4**: 0.0000 (0.00% of total variance)

These results indicate that 100% of the variability in the dataset is captured by the first principal component (PC1) alone. This suggests that the data is essentially one-dimensional, with all other components (PC2, PC3, PC4) contributing no additional information. Therefore, PC1 alone is sufficient to represent the structure of the dataset in this case, and further components are redundant.

To evaluate the relationships between the stock prices of different stocks, a pivot table was created using 'Open' as the index and 'Close' prices as the values. A correlation matrix was then computed across all stocks, followed by a heatmap to visualize the strength of these relationships. (Fig 4)

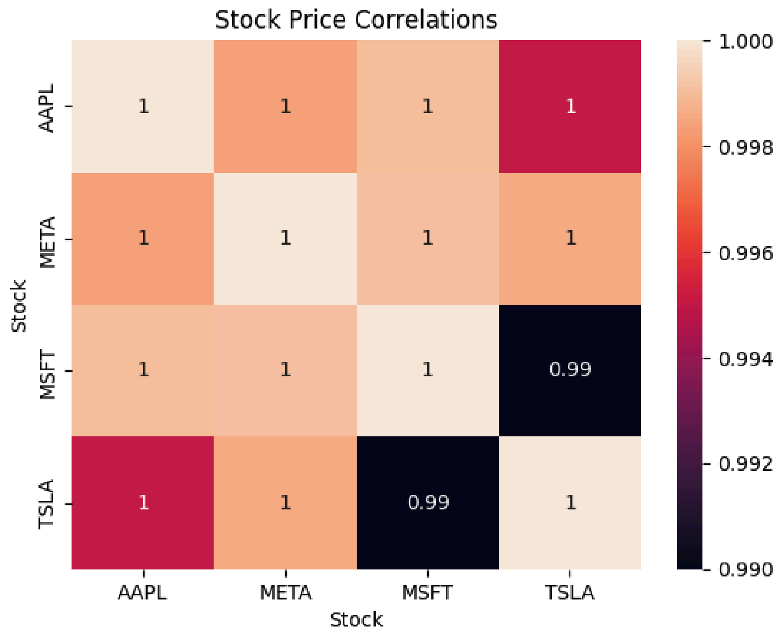

Fig 4: Stock price correlations

The results showed extremely high positive correlations among all stocks:

- AAPL and MSFT had a near-perfect correlation of 0.999, suggesting their closing prices moved very closely together.
- META also showed strong correlations with all other stocks, particularly MSFT (0.9991) and AAPL (0.9984).
- TSLA, while still highly correlated (e.g., 0.995 with AAPL), had a slightly more independent movement compared to the others.

These findings suggest that the stocks in this dataset tend to move in similar directions, possibly due to shared market influences or overlapping trading periods. Understanding these correlations is important for model development, as it indicates potential redundancy in features and the possibility of multicollinearity if multiple stock prices are used as predictors.

To assess the relevance and scale of the historical stock data used in this analysis, an API was utilized to retrieve each stock's current price and compare it to the average historical price from the dataset.

The results revealed substantial growth across all stocks:

Stock	Old Price (Avg)	Current Price	Price Change	% Change
AAPL	$48.03	$212.40	+164.37	+342.18%
META	$85.08	$727.64	+642.56	+755.27%
MSFT	$32.46	$501.56	+469.10	+1445.02%
TSLA	$168.27	$309.81	+141.54	+84.12%

Fig 5: Historical vs. Current Stock Price Comparison

These results highlight significant long-term growth, particularly for MSFT, which increased by over 1,400%, and META, which saw gains of over 750%. Even TSLA, already priced high historically, nearly doubled in value. (Fig 5)

Pre-Processing & Training Model:

The pre-processing phase focused on preparing the data for effective model training. First, historical stock data was loaded and filtered to include only the relevant numerical features: Open, Low, High, and Volume, which were used as predictors. The Close price was selected as the target variable to forecast. To ensure all features contributed equally to the model, the input variables were standardized using StandardScaler, which transformed them to have zero mean and unit variance. This step is especially important for distance-based algorithms like k-Nearest Neighbors and helps improve the convergence of gradient-based models. The dataset was then split into training and testing sets using an 70/30 ratio without shuffling, maintaining the temporal structure of the time series. This allowed the models to be trained on past data and evaluated on future data.

Three regression models — Gradient Boosting, k-Nearest Neighbors (kNN), and Random Forest — were trained and evaluated using the test dataset. The evaluation was based on key performance metrics: R^2 (coefficient of determination), Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

Models	R^2	MAE	RMSE
Gradient boosting regression	0.78	21.91	46.98
k-Nearest Neighbors	-0.04	79.05	101.88
Random forest regression	0.77	22.53	47.97

Fig 6: Model Evaluation Results

- Gradient Boosting achieved the highest R^2 score of 0.78, indicating it explained 78% of the variance in stock closing prices on the test data. It also had the lowest MAE and RMSE, making it the best-performing model overall.
- Random Forest performed nearly as well as Gradient Boosting, with an R^2 score of 0.77 and slightly higher errors.
- k-Nearest Neighbors performed poorly, with a negative R^2 score (–0.04), indicating it failed to capture the trend in the data and performed worse than simply predicting the mean.

After initial model evaluation, I applied GridSearchCV to fine-tune hyperparameters for each model and optimize performance. GridSearchCV systematically searches through combinations of hyperparameters using cross-validation to identify the set that minimizes prediction error.

Models	R^2	MAE	RMSE
Gradient boosting regression	0.78	21.79	46.92
k-Nearest Neighbors	0.77	22.68	47.78
Random forest regression	0.77	22.41	47.80

Fig 7: Model Tuning with GridSearchCV

- Gradient Boosting remained the top-performing model even after tuning, achieving the highest R^2 (0.78) and lowest MAE and RMSE values.
- The hyperparameter tuning significantly improved kNN, which previously performed poorly before optimization.
- Random Forest also demonstrated strong results, with closely matched metrics to Gradient Boosting.

Gradient Boosting showed consistent top performance in both settings, and while the improvement after tuning was slight, it refined the model further, reducing both MAE and RMSE. Moreover, kNN experienced the most dramatic improvement. Before tuning, it had negative R^2 and high error values, meaning it failed to model the data properly. After applying GridSearchCV, kNN's performance matched that of the other ensemble models, showing how important hyperparameter tuning is for simpler algorithms. Finally, Random Forest showed modest improvement in MAE and RMSE after tuning, with stable R^2. Overall, The Gradient Boosting Regressor with GridSearchCV performed the best.

Modeling:

In this phase of the project, I transitioned from regression-based price prediction to a classification approach, with the goal of predicting whether a stock would experience a positive or negative change , represented by a binary target variable (Target). This method frames the problem as a binary classification task, offering practical value for decision-making such as whether to buy, hold, or sell a stock.

Firstly, I began by dropping irrelevant or redundant columns, including: 'Date', 'Stock', 'Average', 'OpenInt', 'Change', 'Daily Return', 'Cumulative Return', and 'Loss'. Then defined the feature matrix X and target vector y based on the binary 'Target' variable. Split the data into training (70%) and testing (30%) sets.

I first began with the **Random Forest Classifier** to establish a baseline for classification performance in predicting the direction of stock price movement. This model was chosen for its robustness, ability to handle high-dimensional data, and resistance to overfitting. The task was framed as a binary classification problem, where the goal was to predict whether the stock price would increase (1) or not increase (0) on the following day, based on historical and technical indicators.

To further improve performance, I applied GridSearchCV to the Random Forest model to explore a range of hyperparameter combinations and select the best-performing configuration using 5-fold cross-validation.

Metric	Random Forest Classifier (Default)	Random Forest Classifier (Tuned)
Accuracy	0.60	0.66
Precision	0.76	0.82
Recall	0.34	0.45
F1 Score	0.47	0.58

Fig 8: Random Forest Classifier Model Results

There is a 6% increase in accuracy (fig 8) indicating a better overall prediction capability. Precision increased, meaning the model made fewer false positive predictions. Recall improved by over 10%, showing that the model was better at identifying true positives (i.e., correctly predicting price increases). While there is a decrease in F1 Score. This demonstrates that GridSearchCV significantly enhanced model performance by fine-tuning hyperparameters, making the Random Forest classifier more robust and effective for stock movement prediction.

To further evaluate the performance of the tuned Random Forest Classifier, I plotted the Receiver Operating Characteristic (ROC) Curve using the predicted probabilities on the test set. The ROC curve visually represents the trade-off between the True Positive Rate (Recall) and the False Positive Rate at various classification thresholds. The Area Under the Curve (AUC) value was 0.77, which suggests that the model has a 77% chance of distinguishing between a positive and a negative class. (Fig 9)

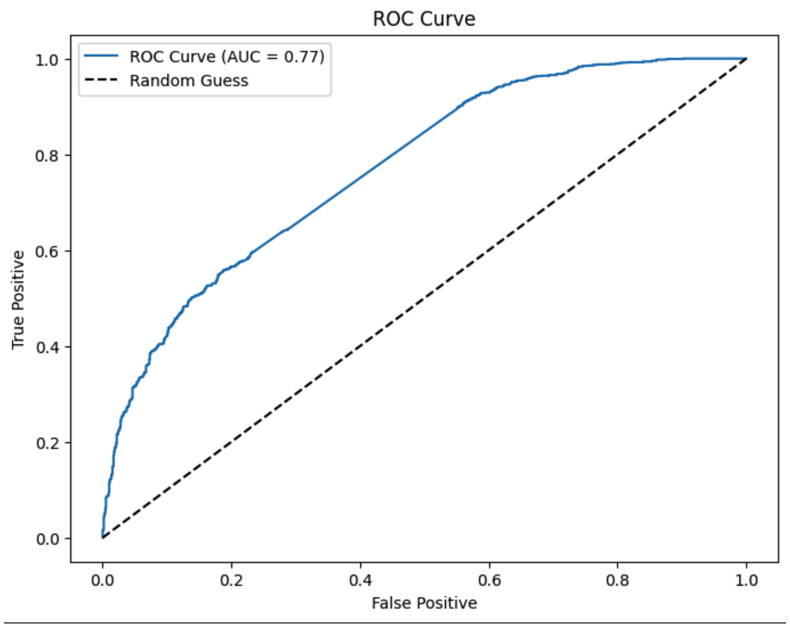

Fig 9: Random Forest Classifier ROC Curve

The second classifier I used was **XGBoost** with and without hyperparameter tuning using GridSearchCV.

Metric	XGBoost (Default)	XGBoost (Tuned)
Accuracy	0.63	0.69
Precision	0.77	0.83
Recall	0.58	0.53
F1 Score	0.66	0.64

Fig 10: XGBoost Model Results

This model performed better compared to the Random Forest Classifier, however XGBoost performed even better after applying the GridSearchCV. While tuning improved precision and accuracy, the recall and F1 score slightly dropped, suggesting that the tuned model became more cautious and avoided false positives at the cost of missing some true positives.

To further evaluate the performance of the tuned XGBoost model, I generated a Receiver Operating Characteristic (ROC) curve. The ROC curve illustrates the trade-off between the true positive rate and the false positive rate at various classification thresholds.

The Area Under the Curve was found to be 0.79, which indicates a strong ability of the model to distinguish between the two classes. (Fig 11)

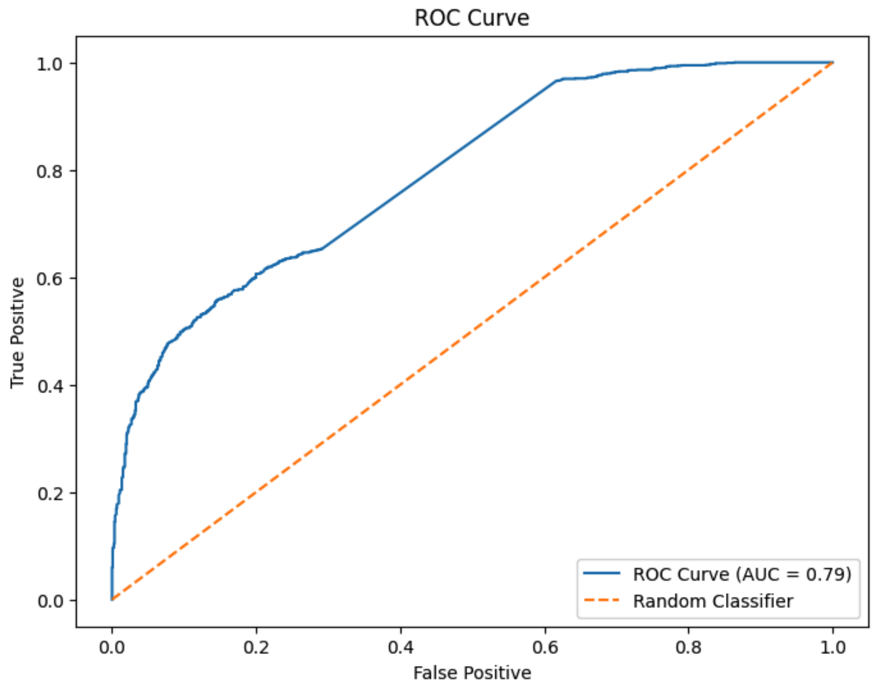

Fig 11: XGBoost ROC Curve

Fig 9: XGBoost Classifier ROC Curve

The third classifier I used was the **LGBM Classifier** with and without hyperparameter tuning using GridSearchCV.

Metric	Gradient Boosting Classifier (Default)	Gradient Boosting Classifier (Tuned)
Accuracy	0.61	0.67
Precision	0.78	0.81
Recall	0.52	0.50
F1 Score	0.63	0.61

Fig 13: LGBM Classifier Model Results

This model showed high precision, indicating it was good at correctly identifying the positive class. However, the low recall revealed it failed to detect a significant number of actual positive cases, leading to imbalanced performance. The tuned version showed notable improvements across all metrics, especially in recall and F1 score. It achieved a better balance between identifying true positives and avoiding false alarms. The AUC score of 0.79, matching that of the tuned XGBoost model, indicates strong overall discriminatory power. This means the model was effective at distinguishing between positive and negative classes over various thresholds.

The forth and final classifier I used was **Gradient Boosting Classifier** with and without hyperparameter tuning using GridSearchCV.

Metric	Gradient Boosting Classifier (Default)	Gradient Boosting Classifier (Tuned)
Accuracy	0.60	0.70
Precision	0.57	0.66
Recall	0.89	0.87
F1 Score	0.70	0.75

Fig 13: Gradient Boosting Classifier Model Results

The tuned Gradient Boosting Classifier demonstrated a notable improvement in overall performance metrics, including accuracy, precision, recall, and F1 score, making it the most effective model in this classification task. After applying GridSearchCV for hyperparameter optimization, the model became more robust and generalizable, showing reduced bias toward the majority class and better handling of class imbalance. This is particularly important in financial modeling, where predicting both upward and downward movements accurately can significantly impact decision-making.

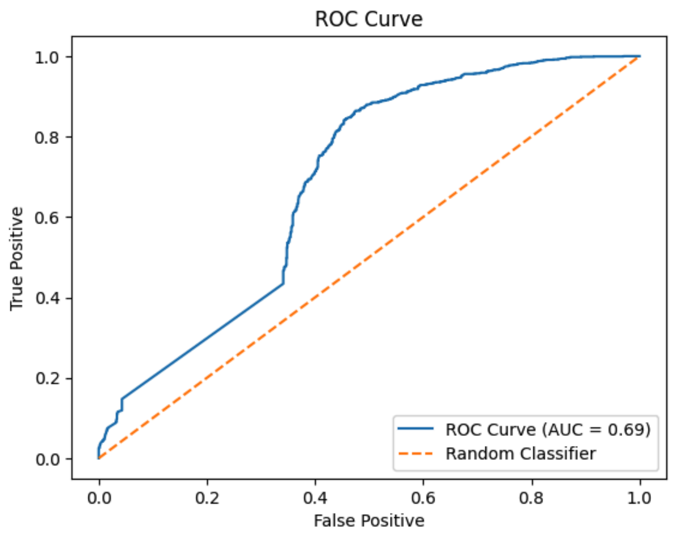

Fig 14: LGBM ROC Curve

When compared to LGBM, XGBoost and Random Forest, the tuned Gradient Boosting Classifier achieved the highest F1 score, indicating better balance between precision and recall. Although the Area Under the ROC Curve (AUC) was 0.69, slightly below the XGBoost, Random Forest, and LGBM model, the Gradient Boosting model offered more consistent classification performance across both classes. Its stable performance and reduced overfitting make it a strong candidate for real-world stock price movement prediction. (Fig 14)

Overall, The tuned Gradient Boosting Classifier is the most reliable model for this task. It achieved the best balance between recall and F1 score, which are critical metrics for stock market classification, especially in minimizing false negatives.

To further evaluate the performance of the tuned Gradient Boosting Classifier, a Precision-Recall (PR) Curve was generated. This curve provides insight into the balance between precision (how many predicted positives are actual positives) and recall (how many actual positives are correctly predicted), especially important in imbalanced classification tasks.

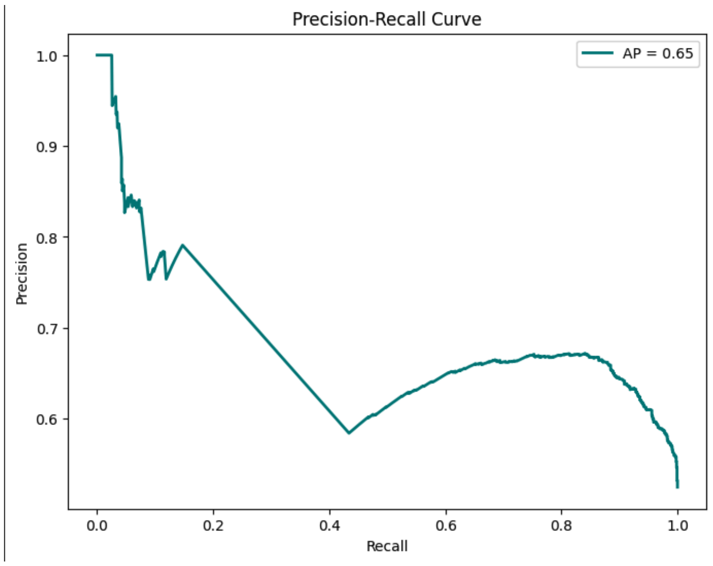

Fig 15: Precision-Recall Curve

The Average Precision (AP) is 0.65, indicating a good balance between precision and recall, reinforcing that the tuned Gradient Boosting model is not only accurate but also effective at handling the trade-off between false positives and false negatives.The PR curve (fig 15) shows consistent performance across various thresholds, which complements the model's high F1 score and strong AUC, confirming its robustness.

Final Takeaways:

Applying GridSearchCV to optimize hyperparameters led to noticeable improvements across most models. Tuned models consistently outperformed their default counterparts in terms of accuracy, precision, recall, and F1 score.

The tuned Gradient Boosting Classifier achieved the highest F1 score (0.75), strong recall (0.87), and competitive precision (0.66). It demonstrated excellent capability in identifying positive class instances while minimizing false negatives. This makes it particularly valuable in real-world forecasting where missing an upward trend can be costly.

Future Research:

For future research, several directions can be explored to further enhance the performance and applicability of the stock price prediction model. One key area is feature engineering, incorporating more complex financial indicators, macroeconomic variables, and sentiment analysis from news or social media could provide deeper insights into market behavior. Additionally, leveraging time-series-based models such as Long Short-Term Memory (LSTM) may help capture temporal dependencies in stock movements more effectively. Ensemble strategies, such as stacking multiple models, could also be investigated to combine the strengths of different algorithms and improve robustness.