Machine Learning Classifiers for Financial Trading Strategies

Problem Definition:

This project explores the implementation of machine learning classifiers to predict stock trading signals using historical financial data. The goal is to develop and evaluate two distinct trading strategies using various ML classification algorithms. We'll collect and preprocess stock price data, engineer relevant features, and train models to generate buy or sell recommendations for securities.

- Kalpavruksha, Rohan Niranjan
- Jangareddi, Paul Rohit
- Jadhav, Poonam
- Gummadi, Sri Harshitha

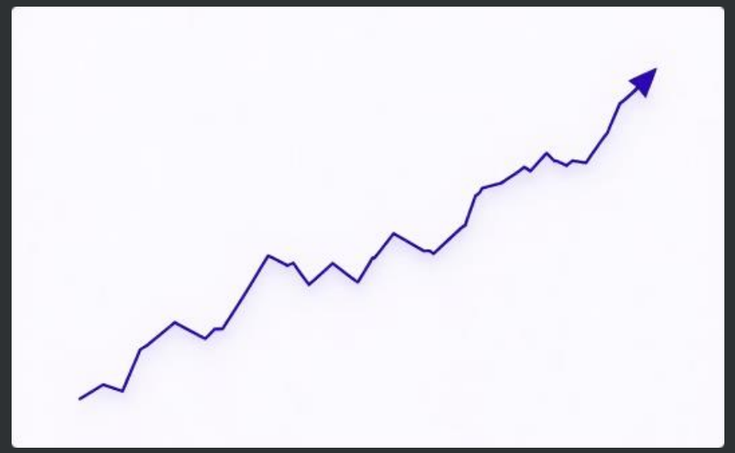
Data Collection and Preprocessing

We begin by collecting historical stock price data using the Yahoo Finance (yfinance) Python library. Our focus is on from 2015 to the end of last year. After downloading the data, we preprocess and clean it to prepare for modeling. Key steps include handling missing values, calculating necessary features like moving averages, and defining our target variables for the two trading strategies.

1. **Data Collection**
   Use yfinance to download historical stock data
2. **Data Cleaning**
   Handle missing values and outliers
3. **Feature Engineering**
   Calculate moving averages and other relevant features
4. **Target Definition**
   Create buy/sell signals for both trading strategies

Exploratory Data Analysis(EDA)

We implement various data visualization techniques to gain insights into our stock data and model performance. Key visualizations include:



Closing Price Over Time

Visualize the stock's price movement throughout the analyzed period.

Volume Traded Over Time

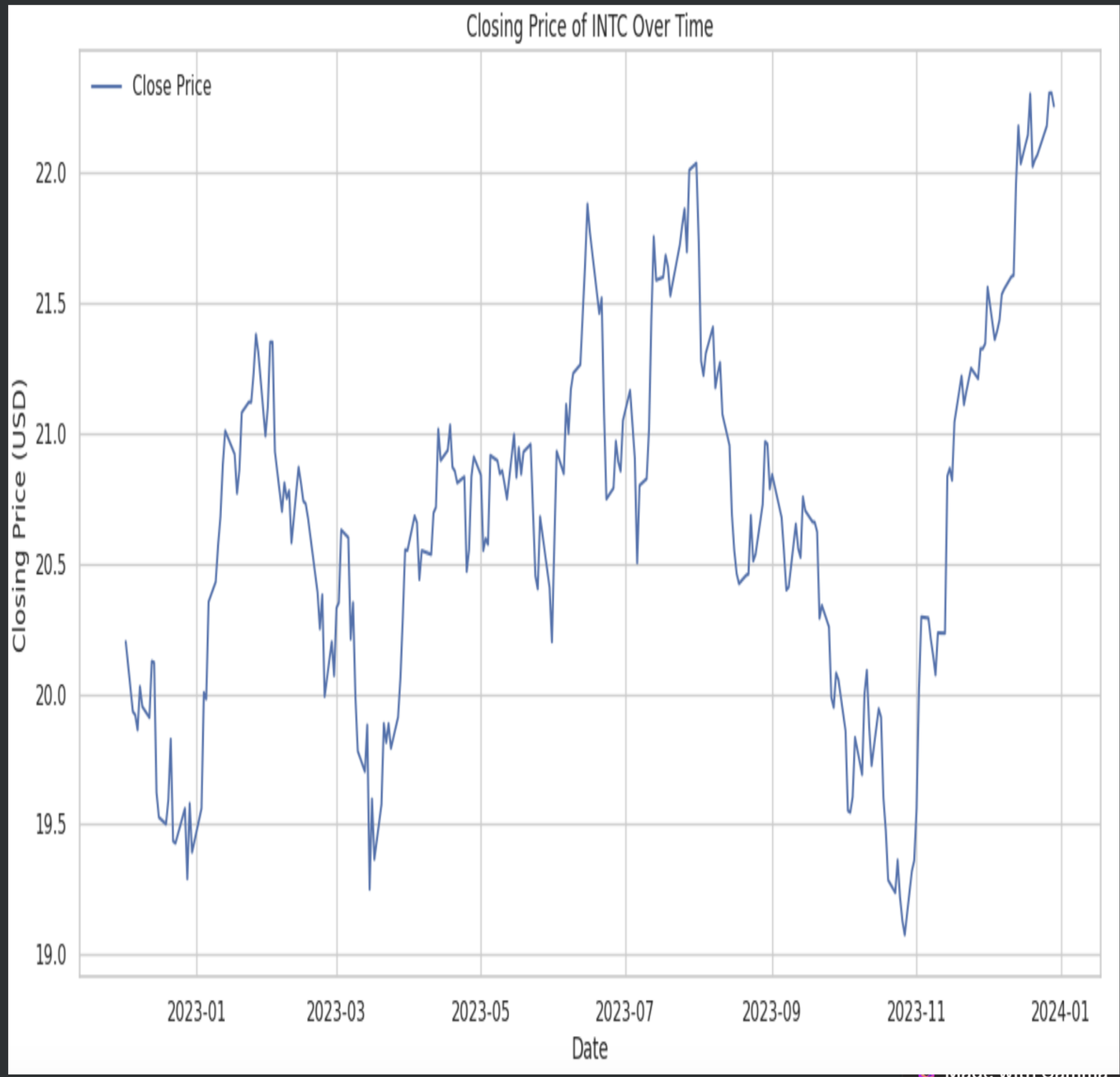
Analyze trading volume patterns and their relation to price movements.

Moving Averages

Plot 50-day and 200-day moving averages to identify trends and potential crossovers.

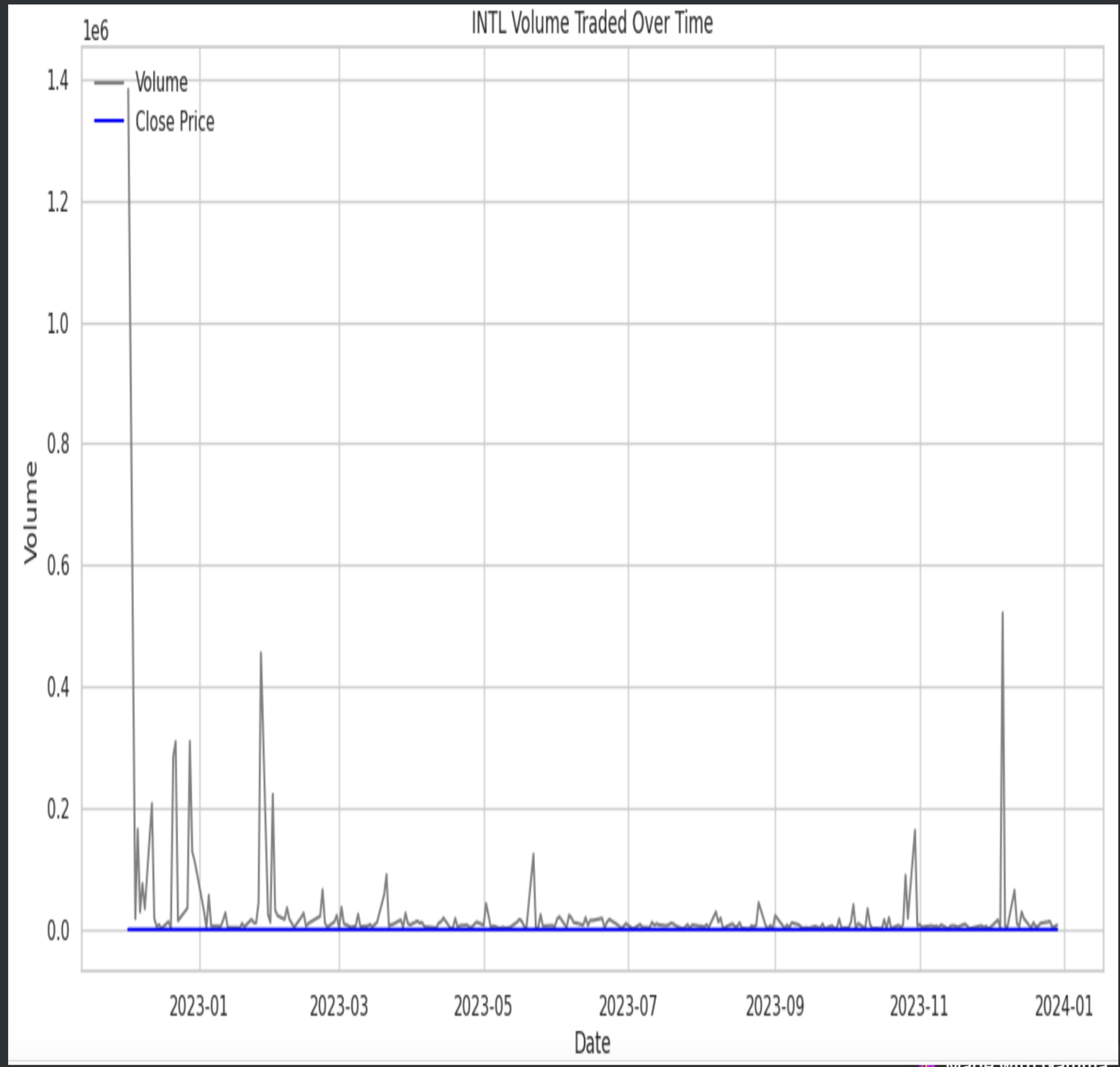
Closing Price Over Time

- Intel's stock price showed high volatility in 2023, with a notable drop mid-year. It rebounded strongly towards the end of the year, closing near yearly highs.



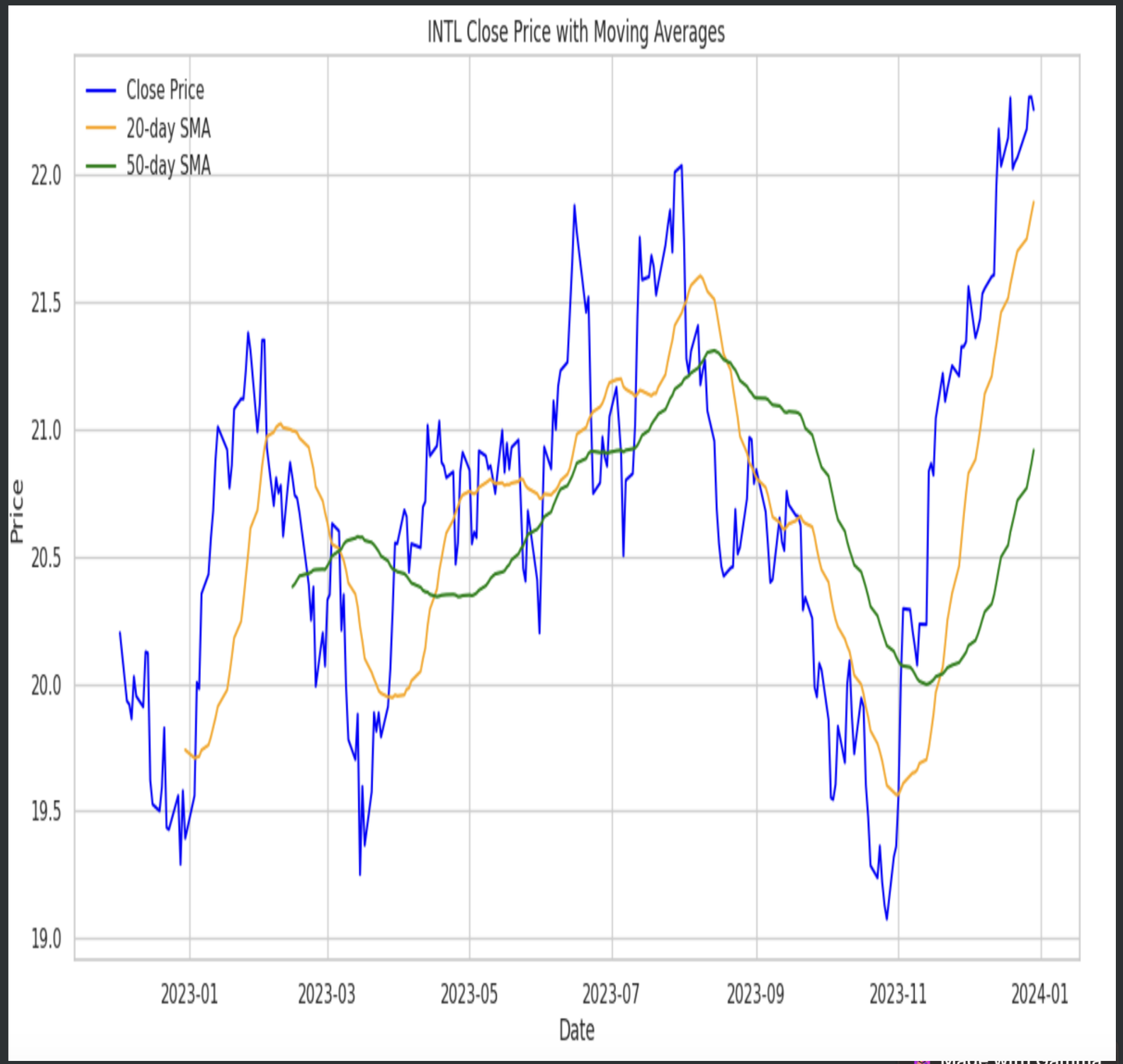
Volume Traded Over Time

- The chart shows that INTC's trading volume experienced occasional spikes, particularly at the start and towards the end of the year. These high-volume periods could indicate increased trading activity, likely due to significant events or news affecting the stock.



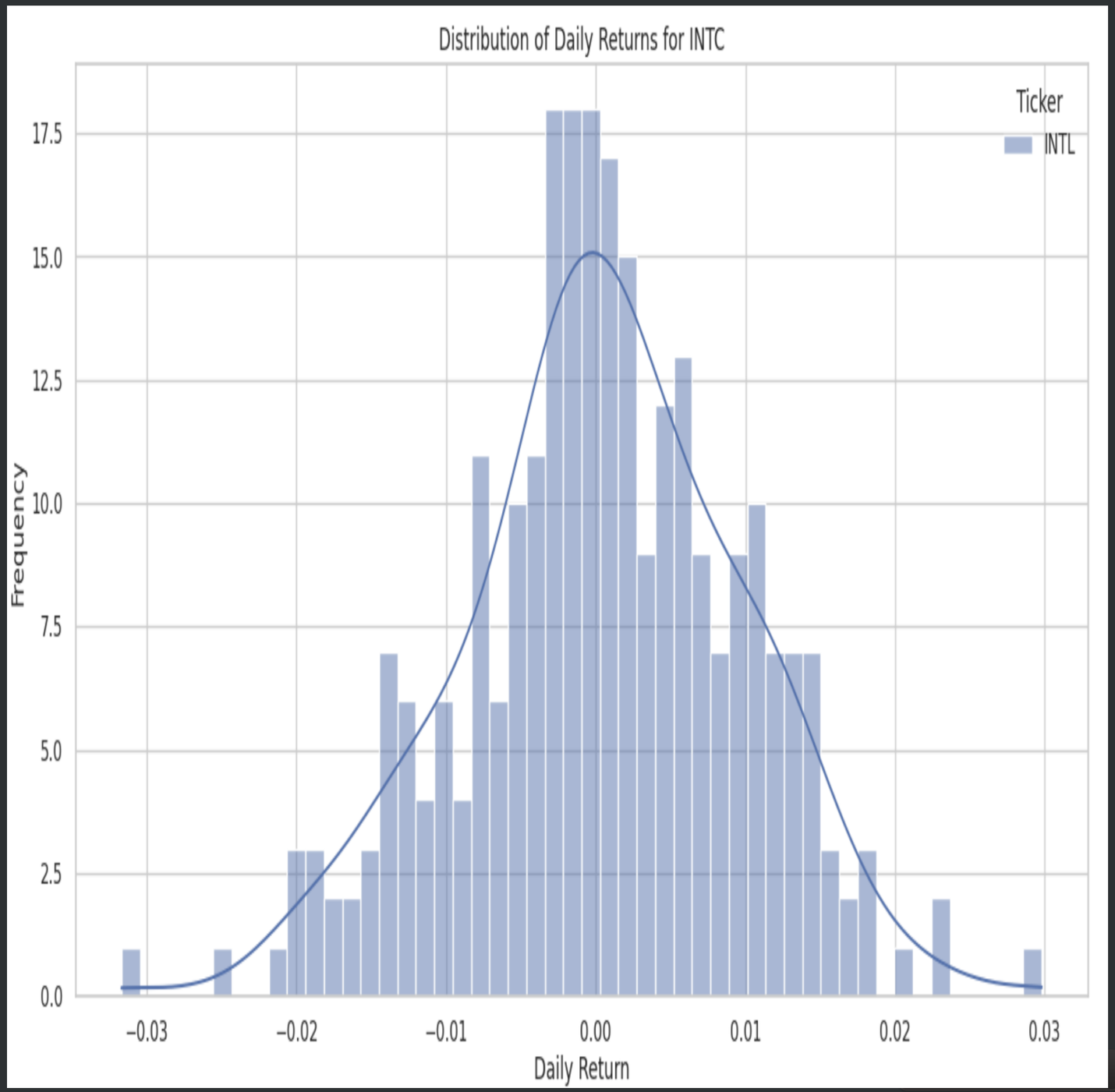
Moving Averages (50-day and 200-day)

- The chart shows Intel's (INTC) stock price with 50-day and 200-day Simple Moving Averages (SMAs). The 50-day SMA crossing above the 200-day SMA in late 2023 suggests a bullish trend, indicating potential upward momentum in the stock price.



Distribution of Daily Returns

- The returns appear to follow a normal distribution centered around zero, indicating that positive and negative returns are roughly balanced. Additionally, most returns lie within the range of -0.02 to 0.02, suggesting low volatility in daily returns.



Implementation in Python

We implement our analysis in a Jupyter Notebook, organizing our code into clear sections for data preprocessing, feature engineering, model training, and evaluation. Here's a sample structure:

Data Preparation

- Import libraries
- Download data
- Clean and preprocess
- Engineer features
- Define target variables

Model Training

- Split data
- Initialize classifiers
- Train models
- Generate predictions

Evaluation

- Calculate metrics
- Visualize results
- Compare strategies
- Interpret findings

Trading Strategy 1: Next Day Price Comparison

Our first trading strategy compares the next trading day's close price to today's close price. If tomorrow's close is higher, we generate a buy signal (+1). Otherwise, we generate a sell signal (-1). This strategy aims to capture short-term price movements. We implement this using NumPy's functionality:

Signal Generation

```
y = np.where(df['Stock_Price'].shift(-1) > df['Stock_Price'], 1, -1)
```

Buy Signal (+1)

Next day's close > Today's close

Sell Signal (-1)

Next day's close ≤ Today's close

Trading Strategy 2: Moving Average Crossover

The second strategy utilizes the golden cross and death cross concepts based on the 50-day and 200-day moving averages. A golden cross (buy signal) occurs when the 50-day MA crosses above the 200-day MA, indicating a bullish trend. Conversely, a death cross (sell signal) happens when the 50-day MA crosses below the 200-day MA, suggesting a bearish trend.

1

Calculate MAs

Compute 50-day and 200-day moving averages

2

Identify Crossovers

Detect when the 50-day MA crosses the 200-day MA

3

Generate Signals

Assign +1 for golden cross, -1 for death cross

Model Training and Evaluation Process

For each classifier and trading strategy, we follow these steps:

1. **Data Split**

   Divide the dataset into 80% training and 20% testing sets

2. **Model Fitting**

   Train the classifier on the training data using default parameters

3. **Prediction**

   Generate predictions on the test set

4. **Evaluation**

   Assess model performance using appropriate metrics

We use scikit-learn for KNN, Random Forest and Gradient Boosting.

Feature Engineering and Data Preprocessing:

```
from sklearn.model_selection import train_test_split

# For Strategy 1
X1 = df_cleaned[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']]
y1 = df_cleaned['Target_Strategy_1']

# Split the data into training and testing sets for Strategy 1
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2, random_state=42)

# For Strategy 2
X2 = df_cleaned[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']]
y2 = df_cleaned['Target_Strategy_2']

# Split the data into training and testing sets for Strategy 2
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2, random_state=42)

(X1_train.shape, X1_test.shape, y1_train.shape, y1_test.shape), (X2_train.shape, X2_test.shape, y2_train.shape, y2_test.shape)
```

```
(((56, 6), (15, 6), (56,), (15,)), ((56, 6), (15, 6), (56,), (15,)))
```

Model Selection

We implement three different machine learning classifiers to predict buy/sell signals based on our engineered features:

K-Nearest Neighbors (KNN)

A non-parametric method that classifies based on the majority class of its k nearest neighbors.

Random Forest Classifier (RF)

An ensemble learning method that constructs multiple decision trees and outputs the mode of their predictions.

Gradient Boosting Classifier (GB)

An ensemble technique that builds trees sequentially, with each tree correcting errors from the previous ones.

KNN

7.APPLY KNN

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Apply KNN for Strategy 1
knn1 = KNeighborsClassifier()
knn1.fit(X1_train, y1_train)
y1_pred_knn = knn1.predict(X1_test)
accuracy1_knn = accuracy_score(y1_test, y1_pred_kn

# Apply KNN for Strategy 2
knn2 = KNeighborsClassifier()
knn2.fit(X2_train, y2_train)
y2_pred_knn = knn2.predict(X2_test)
accuracy2_knn = accuracy_score(y2_test, y2_pred_kn

(accuracy1_knn, accuracy2_knn)
```

```
(0.4, 0.6666666666666666)
```

```
K-Nearest Neighbors (KNN) Model Evaluation
==========================================
Accuracy: 0.6066

Confusion Matrix:
[[22 10]
 [14 15]]

Classification Report:
              precision    recall  f1-score   support

          -1       0.61      0.69      0.65        32
           1       0.60      0.52      0.56        29

    accuracy                           0.61        61
   macro avg       0.61      0.60      0.60        61
weighted avg       0.61      0.61      0.60        61
```

Strategy 1 Accuracy: 40% Strategy 2 Accuracy: 66.67%

Random Forest

8. APPLY RANDOM FOREST CLASSIFIER

```
from sklearn.ensemble import RandomForestClassifier

# Apply Random Forest for Strategy 1
rf1 = RandomForestClassifier(random_state=42)
rf1.fit(X1_train, y1_train)
y1_pred_rf = rf1.predict(X1_test)
accuracy1_rf = accuracy_score(y1_test, y1_pred_rf)

# Apply Random Forest for Strategy 2
rf2 = RandomForestClassifier(random_state=42)
rf2.fit(X2_train, y2_train)
y2_pred_rf = rf2.predict(X2_test)
accuracy2_rf = accuracy_score(y2_test, y2_pred_rf)

(accuracy1_rf, accuracy2_rf)
```

```
(0.7333333333333333, 0.6666666666666666)
```

Strategy 1 Accuracy: 73.34% Strategy 2 Accuracy: 66.67%

```
Random Forest Model Evaluation on Dataset 1
==========================================
Accuracy: 0.5370

Confusion Matrix:
[[10 16]
 [ 9 19]]

Classification Report:
-1: {'precision': 0.5263157894736842, 'recall': 0.38461538461538464, 'f1-score': 0.4444444444444444, 'support': 26.0}
1: {'precision': 0.5428571428571428, 'recall': 0.6785714285714286, 'f1-score': 0.6031746031746031, 'support': 28.0}
accuracy: 0.5370370370370371
macro avg: {'precision': 0.5345864661654135, 'recall': 0.5315934065934066, 'f1-score': 0.5238095238095237, 'support': 54.0}
weighted avg: {'precision': 0.5348927875243665, 'recall': 0.5370370370370371, 'f1-score': 0.5267489711934156, 'support': 54.0}


Random Forest Model Evaluation on Dataset 2
==========================================
Accuracy: 0.5738

Confusion Matrix:
[[27  5]
 [21  8]]

Classification Report:
-1: {'precision': 0.5625, 'recall': 0.84375, 'f1-score': 0.675, 'support': 32.0}
1: {'precision': 0.6153846153846154, 'recall': 0.27586206896551724, 'f1-score': 0.38095238095238093, 'support': 29.0}
accuracy: 0.5737704918032787
macro avg: {'precision': 0.5889423076923077, 'recall': 0.5598060344827587, 'f1-score': 0.5279761904761905, 'support': 61.0}
weighted avg: {'precision': 0.587641866330391, 'recall': 0.5737704918032787, 'f1-score': 0.5352068696330992, 'support': 61.0}
```

Gradient Boosting

```
Class Balance after SMOTE:
 Target_Strategy_2
-1    151
 1    151
Name: count, dtype: int64
Model Accuracy: 0.7868852459016393
```

- **Accurancy of Startergy 1: 53.73%, Accuracy of Stratergy 2 is 78.68%**

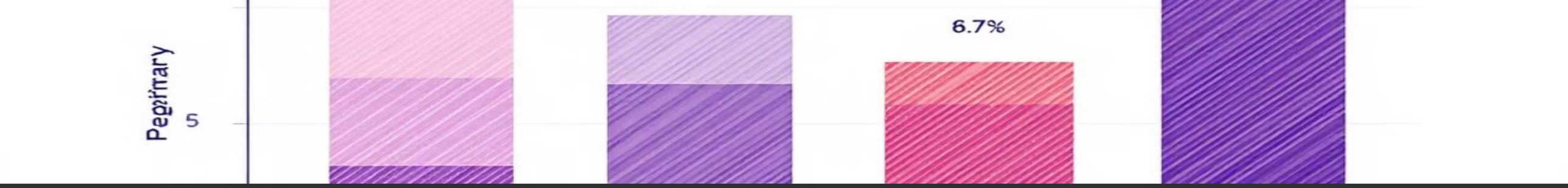
```
Gradient Boosting Model Evaluation
==================================
Accuracy: 0.7869

Confusion Matrix:
[[26  6]
 [ 7 22]]

Classification Report:
              precision    recall  f1-score   support

    Decrease       0.79      0.81      0.80        32
    Increase       0.79      0.76      0.77        29

    accuracy                           0.79        61
   macro avg       0.79      0.79      0.79        61
weighted avg       0.79      0.79      0.79        61
```



Model Evaluation Conclusion

For Strategy 1, Random Forest is the recommended model due to its strong accuracy. For Strategy 2, XGBoost is the best choice, providing the highest accuracy across all models. These findings suggest that different strategies benefit from different model types, likely due to variations in how each model interprets the feature patterns and interactions within each strategy.

Best Performing Model: Our analysis revealed that the Gradient Boosting Classifier (GB) emerged as the most effective model, delivering the highest accuracy in predicting the future movement of stock prices based on the implemented trading strategies. This underscores the GB model's robustness and its aptitude for capturing complex patterns in financial time series data.

Least Performing Model: The K-Nearest Neighbors (KNN) model, while valuable for its simplicity and ease of interpretation, lagged in performance compared to its counterparts. This highlights the challenges KNN faces in navigating the noisy and non-linear nature of stock market data.

Model Evaluation: Through precision, recall, and F1-score metrics, we gained deeper insights into each model's predictive performance. These evaluations were instrumental in understanding the trade-offs between sensitivity and specificity among the models.

Insights Gained

- Historical stock data is loaded for analysis (1995–2024).
- Two trading strategies are explored:
  Strategy 1: Predicts next day's stock price movement using closing price.
  Strategy 2: Uses moving averages (50-day and 200-day) to define trends.

Problems Encountered

- Missing values were present but addressed through data cleaning.
- Large data spans may introduce computational inefficiencies or risks of outdated patterns.
- Potential overfitting due to reliance on specific indicators like moving averages.

Future Improvements

To enhance our project, we can explore model tuning by adjusting hyperparameters for selected classifiers. This process can potentially improve model performance and provide insights into the sensitivity of our models to different parameter settings. Additional areas for improvement include:

Feature Selection

Identify most impactful features

Hyperparameter Tuning

Optimize model parameters

Time Series CV

Use time-based cross-validation

Backtesting

Simulate trading performance