ASSESSMENT 1

An Analysis of Stock Price Prediction and Investment Strategy for Proctor & Gamble (P&G)

2198 words

MSc FinTech and Business Analytics

Artificial Intelligence and Machine Learning for Finance

Module Code: FNCE043W

Module Leader: Dr Yumei Yao

Deadline: 2nd April 2024

Table of Contents

1.Introduction

Procter & Gamble (P&G) is a leading multinational consumer goods company founded in 1837 in Cincinnati, Ohio. It produces a wide range of well-known household and personal care brands, including Tide detergent, Gillette razors, Pampers diapers, Crest toothpaste, and Bounty paper towels. P&G operates in over 180 countries worldwide and is known for its innovative product development, effective marketing strategies, and focus on sustainability. The company is one of the largest and most valuable consumer brands globally, with a portfolio of over 65 brands that are household names.

2.Data Collection and Feature Engineering

2.1 P&G Historical data

A data collection was performed on P&G with the ticker (PG) from Jan 2014 to Dec 2023 from Yahoo Finance resulting in 2516 rows of data which included parameters such as Open Price, Close Price, High, Low, Volume for the stock. Rows were removed if data was missing or unavailable. (Procter & Gamble, 2023)

2.2 Feature Engineering

The below mentioned features were selected for training the models:

i. **High-Low (H-L)**

H-L represents daily difference between the highest price and the lowest price of the stock which is used to signify fluctuations of the market in the day. (Yang and Zhang, 2000)

H-L = High Price – Low Price

ii. **Open-Close (O-C)**

O-C represents daily difference between the Open price and the Close price of the stock which is used to capture intraday movements of the stock. (Chan and Lien, 2003)

O-C = Open Price – Close Price

iii. **20 Day Moving Average (20 MA) and 200 Day Moving Average (200 MA)**

20 Day MA represents the short-term average price whereas 200 MA represents the long-term average price of the stock. (Dodd, 1941)

n-Day MA = $\frac{Sum\ Of\ Closing\ Prices\ of\ last\ n\ days}{n}$

iv. **20 Day Exponential Moving Average (20 EMA)**

The Exponential Moving Average (EMA) is a type of moving average that places greater emphasis on the most recent data points, rather than weighting all data points equally like the Moving Average (MA). The EMA gives an exponentially decreasing weight to older data, with the most recent prices having the highest influence on the average. This allows the EMA to be more responsive to the latest market movements compared to the MA. The 20 EMA depicts the short-term exponential moving average of the stock. (Klinker, 2010)

$$EMA_{Today} = (Value_{Today} \times (\frac{Smoothing}{1 + Days})) + EMA_{Yesterday} \times (1 - (\frac{Smoothing}{1 + Days}))$$

v. Standard Deviation of closing Price (Std_dev)

Volatility of stock price over a 5-day rolling window is depicted by the (Std_dev). (Beckers, 1981)

$$\text{Standard Deviation} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}}$$

where:

$x_i =$ Value of the i^{th} point in the data set

$\overline{x} =$ The mean value of the data set

$n =$ The number of data points in the data set

vi. Price Rise (Price_Rise)

(Price_Rise) signifies binary representation 1 when there is a rise in price otherwise 0.

3. Exploratory Data Analysis of Features

Exploratory Data Analysis (EDA) is a data analysis approach that focuses on discovering patterns, identifying anomalies, and testing hypotheses through visual and statistical methods, rather than relying solely on confirmatory techniques. EDA allows researchers to gain a deeper understanding of data characteristics and inform subsequent analysis.

The **H-L** (High-Low) feature represents the difference between the highest and lowest price of a stock during a given time, such as a trading day or week. This metric provides insight into the volatility and trading range of the stock. The **O-C** (Open-Close) feature shows the difference between the opening and closing prices, indicating the overall price movement during the period. This can reveal whether the stock closed higher or lower than it opened. The **Std_dev** (Standard Deviation) of the closing prices measures the dispersion or volatility of the stock's prices around the mean closing price. A higher standard deviation suggests greater price fluctuations and risk.

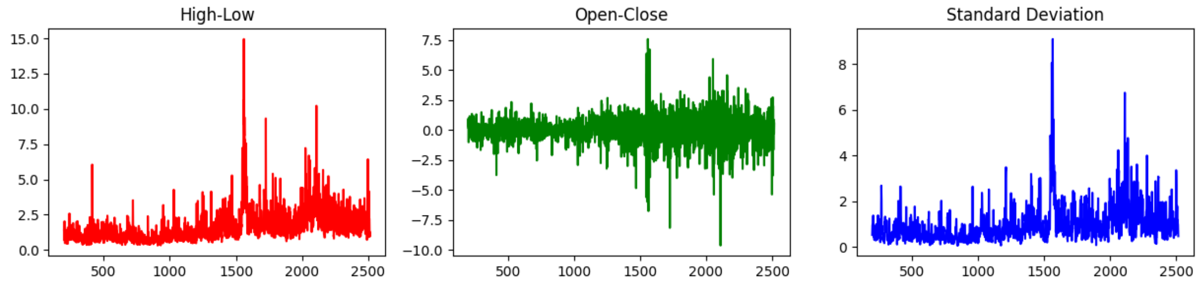


Figure 1. Plots of H-L, O-C, Std_dev

The **‘20 MA’** represents short term fluctuations in the stock whereas **‘200 MA’** signifies long term movement in the stock price and **’20 EMA’** signifies short term fluctuations with recency bias in the stock price. For both ’20 MA’ and ’20 EMA’ the graph displays quite similar movements but ’20 EMA’ has sharper highs and lows due to its recency bias and ‘200 MA’ is the smoothest as the long-term prices don’t show the noise that might be observed in the short-term features.

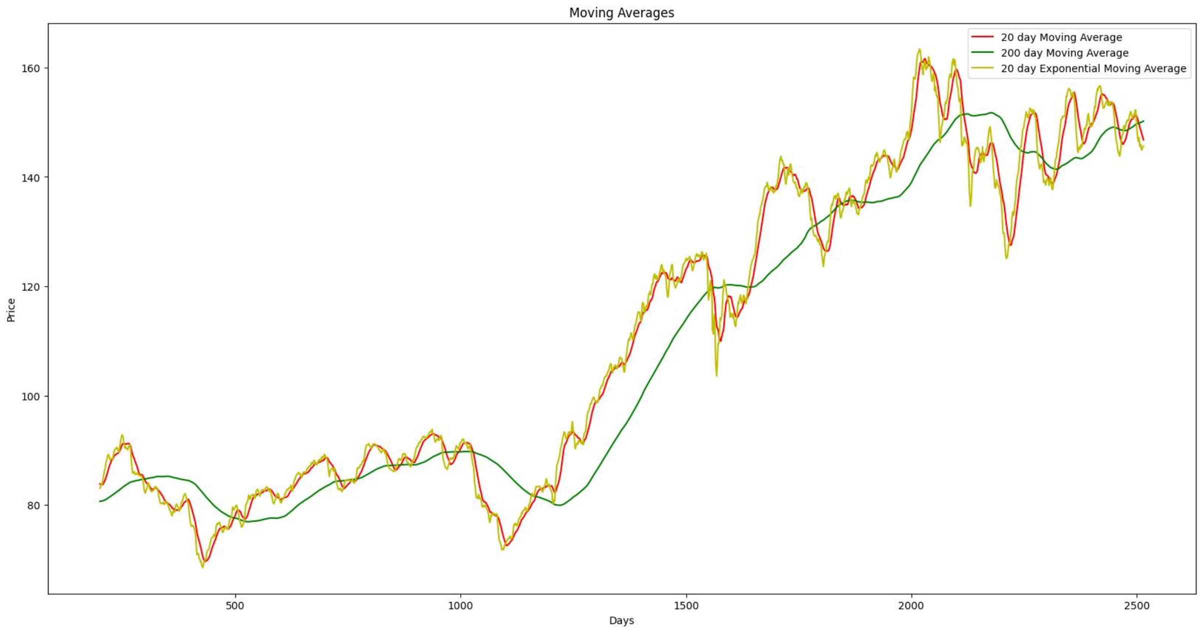


Figure 2. Plot for 20 MA, 200 MA, 20 EMA

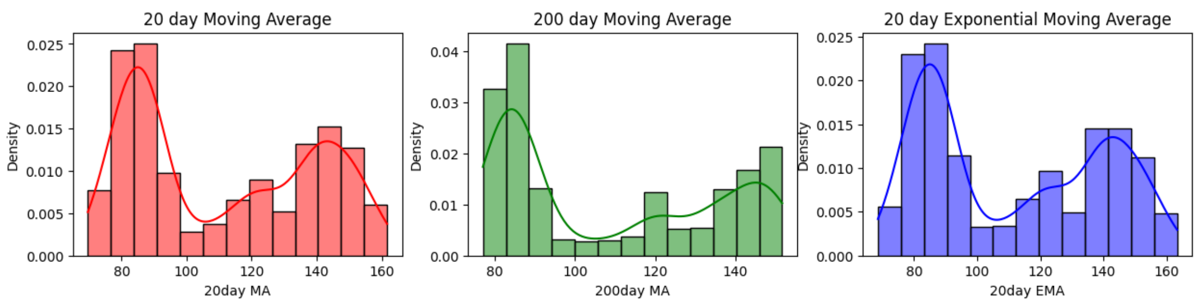


Figure 3. Histogram Plot for density of 20 MA, 200 MA, 20 EMA

The feature **"Price_Rise"** is a binary variable used in the analysis of stock price movements. It indicates whether the price of a stock has risen (Price_Rise = 1) or not risen (Price_Rise = 0) over a specific period. The bar charts in the image are used to visualize the distribution of closing prices (Close) in relation to the occurrence of a price rise.

The left bar chart shows the count of instances where there was no price rise (Price_Rise = 0), while the right bar chart shows the count of instances where there was a price rise (Price_Rise = 1). Each bar represents the frequency of closing prices within a certain range. The height of the bars indicates the count of occurrences for each range of closing prices.

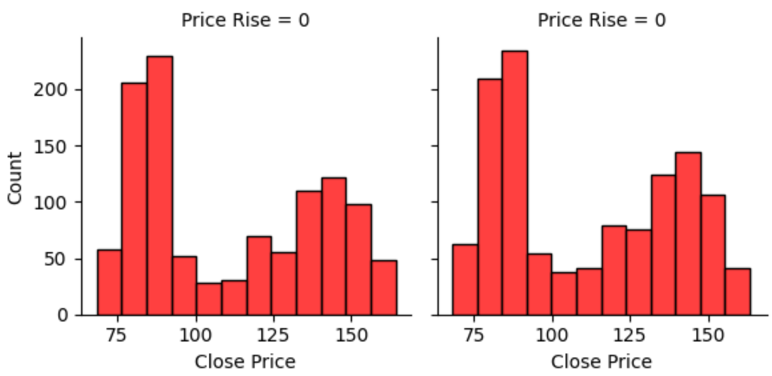


Figure 4. Rise & Fall plot and count.

The **summary statistics** provide a high-level overview of the Stock Price:

Count: The dataset contains **2,316** observations.

Mean: The average values for each metric, such as the mean open price of **$111.49**.

Standard Deviation: The spread of the data, with the open price having a standard deviation of **$28.35**.

Minimum and Maximum: The lowest and highest values observed in the dataset, e.g., the minimum open price of **$68.02** and the maximum of **$164.40**.

	open	high	low	close	H-L	O-C	20day MA	200day MA	20day EMA	Std_dev	Price_Rise
count	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000
mean	111.494227	112.326598	110.703618	111.529948	1.622979	0.035721	111.248074	108.561624	111.448503	1.012523	0.522021
std	28.347177	28.616011	28.093311	28.351195	1.148218	1.158254	28.231660	27.027643	28.308764	0.831379	0.499623
min	68.019997	68.300003	65.019997	68.059998	0.309998	-9.630005	69.643000	76.945850	68.553085	0.053197	0.000000
25%	85.122498	85.577501	84.589996	85.157503	0.880005	-0.440002	84.892374	84.395725	85.190814	0.492779	0.000000
50%	105.985001	106.875000	105.200001	106.095001	1.330002	0.070000	105.737251	93.723675	105.687053	0.788942	1.000000
75%	139.240002	140.434994	138.199997	139.264996	2.002499	0.572504	139.753625	135.741300	139.286132	1.257656	1.000000
max	164.399994	165.350006	163.399994	164.210007	14.949997	7.570000	161.622000	151.743700	163.404134	9.088996	1.000000

Figure 5. Descriptive Statistics of the Dataset

4.Machine Learning Classification

We compared the performance of Logistic Regression and Naïve Bayes Classifier models in predicting stock price movements of P&G using historical data features. The Logistic Regression Classifier outperformed Naïve Bayes, achieving higher accuracy in the binary classification task.

The study pre-processed the dataset by splitting it into features (X) and the target variable (Y) representing the 'Price_Rise' binary column. The features included 'H-L' to 'Std_dev'. The data was divided into training and testing sets (80/20) and feature scaled using StandardScaler.

4.1 Logistic Regression

Logistic regression is a statistical model used for binary classification problems. It predicts the probability of an outcome being 1 or 0, given a set of independent variables, by fitting data to the logistic function. (Stoltzfus, 2011)

Logistic Function:

$$Y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

Where:

- Y is probability of Positive output
- β_0 is the intercept.
- β_1 is the coefficient of feature X.

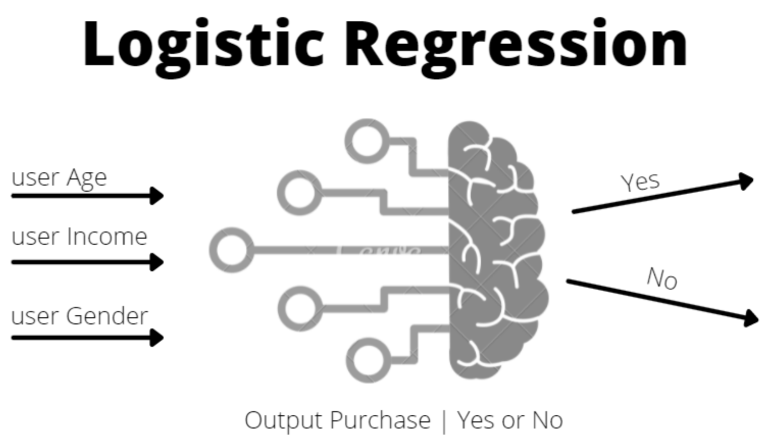


Figure 6. Logistic Regression Example

4.2 Naïve Bayes

Naive Bayes is a simple and efficient supervised machine learning algorithm that uses Bayes' theorem to classify data. It assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature, making it a "naive" algorithm. Despite this simplifying assumption, Naive Bayes often performs surprisingly well on many real-world classification tasks. (Nadeau and Sekine, 2007)

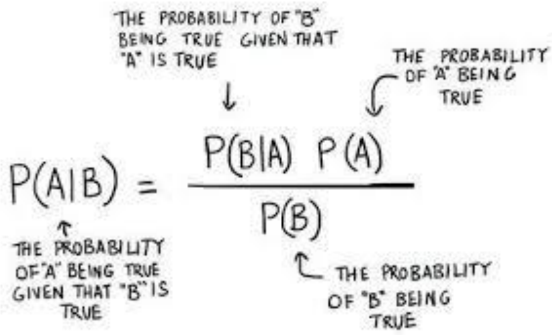


Figure 7. Baye's Theorem

5.Cross Validation on Machine Learning Methods

Cross-validation is a technique to assess a model's performance by iteratively training on a subset of the data and evaluating on the remaining subset. In k-fold cross-validation, the data is divided into k equal parts, and the model is trained and tested k times, using a different part as the test set each time. (Granholm, Noble and Käll, 2012)

$$\text{Mean Accuracy} = \frac{1}{k}\sum_{i=1}^{k} \text{Accuracy}_i$$

where Accuracy_i is the accuracy of the model on the i-th fold.

$$\text{Standard Deviation} = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(\text{Accuracy}_i - \text{Mean Accuracy})^2}$$

5.1 Cross Validation for Logistic Regression

The logistic regression model achieved an accuracy of 50.26% across the 5 folds, indicating its performance is slightly better than random guessing. The standard deviation of 0.03 suggests the model's performance is relatively consistent across different subsets of the data.

5.2 Cross Validation for Naïve Bayes

The Naïve Bayes Classifier achieved an average accuracy of 51.3% across the cross-validation folds, suggesting its performance is slightly better than random guessing and better than logistic regression. The standard deviation of 0.02 indicates some variability in the model's performance across different subsets of the data, potentially signalling sensitivity to the training data.

6.Evaluation of Machine Learning Methods

6.1 Classification Report

The classification report is a comprehensive evaluation of a machine learning model's performance. It provides key metrics such as precision, recall, F1-score, and support for each class predicted by the model, allowing for a detailed assessment of the model's strengths and weaknesses across different classes. (A.Jabbar Alkubaisi, Kamaruddin and Husni, 2018)

The logistic regression model for the given data achieved an accuracy of 52%, which is slightly above random guessing performance. However, the model demonstrated higher precision and recall for the positive class (class 1, representing price rises), suggesting that it is better at identifying instances of price rises compared to the negative class (class 0, representing non-price rises).

	precision	recall	f1-score	support
0	0.52	0.10	0.16	226
1	0.52	0.92	0.66	238
accuracy			0.52	464
macro avg	0.52	0.51	0.41	464
weighted avg	0.52	0.52	0.42	464

The Naive bayes classification model has an accuracy of 53% which is better than that of logistic regression and the recall and precision is high for the Price Rise class (Price Rise=1).

	precision	recall	f1-score	support
0	0.59	0.12	0.20	226
1	0.52	0.92	0.67	238
accuracy			0.53	464
macro avg	0.56	0.52	0.43	464
weighted avg	0.55	0.53	0.44	464

After evaluating these classifiers, we can conclude with the 20% split test data the Naïve bayes model gives us slightly higher accuracy and lower standard deviation than the logistic regression.

6.2 Confusion Matrix

The confusion matrices compare the performance of Naive Bayes and Logistic Regression models. Naive Bayes correctly predicted 27 instances of class 0 and 219 of class 1, with 19 false negatives and 199 false positives. Logistic Regression predicted 22 instances of class 0 and 218 of class 1 correctly, with 20 false negatives and 204 false positives. Both models are better at predicting class 1, but Logistic Regression has fewer true positives and more false positives than Naive Bayes, indicating a trade-off between sensitivity and precision between the two models. The colour scale reflects the quantity of predictions in each category. (Parker, 2001)

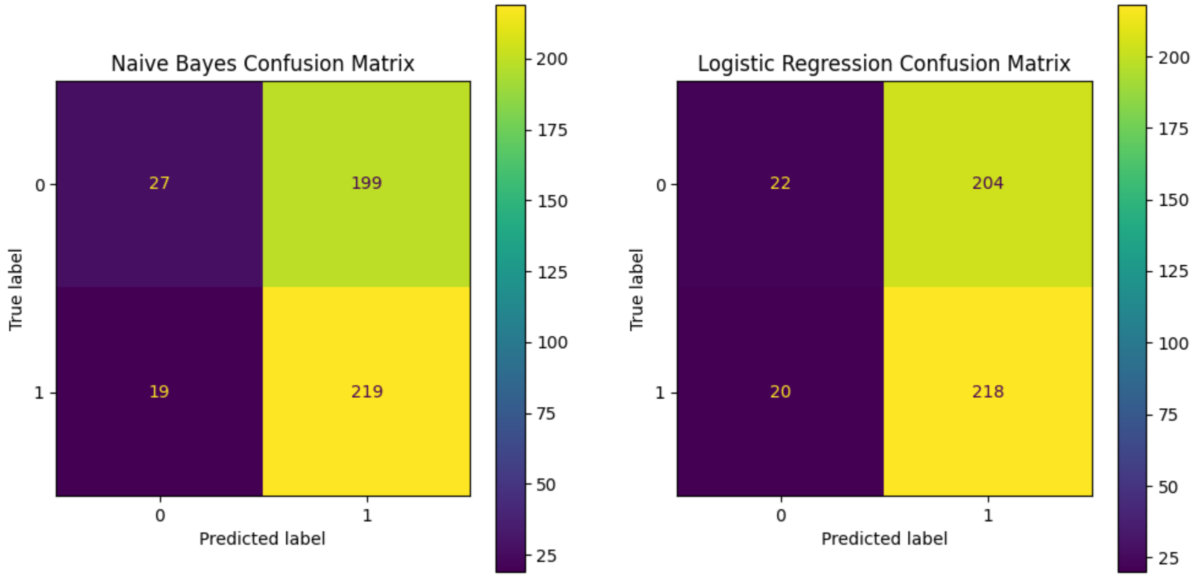


Figure 8. Confusion Matrix for Naïve bayes and Logistic regression

6.3 Roc Curve

The ROC curves in the image indicate that both the Naive Bayes and Logistic Regression classifiers have an AUC (Area Under the Curve) of 0.50, which suggests that these models perform no better than random chance at distinguishing between the positive and negative classes. The diagonal line of the curves implies that for any increase in the True Positive Rate, there is an equal increase in the False Positive Rate, indicating a lack of discrimination between the classes. (Hong and Lee, 2011)

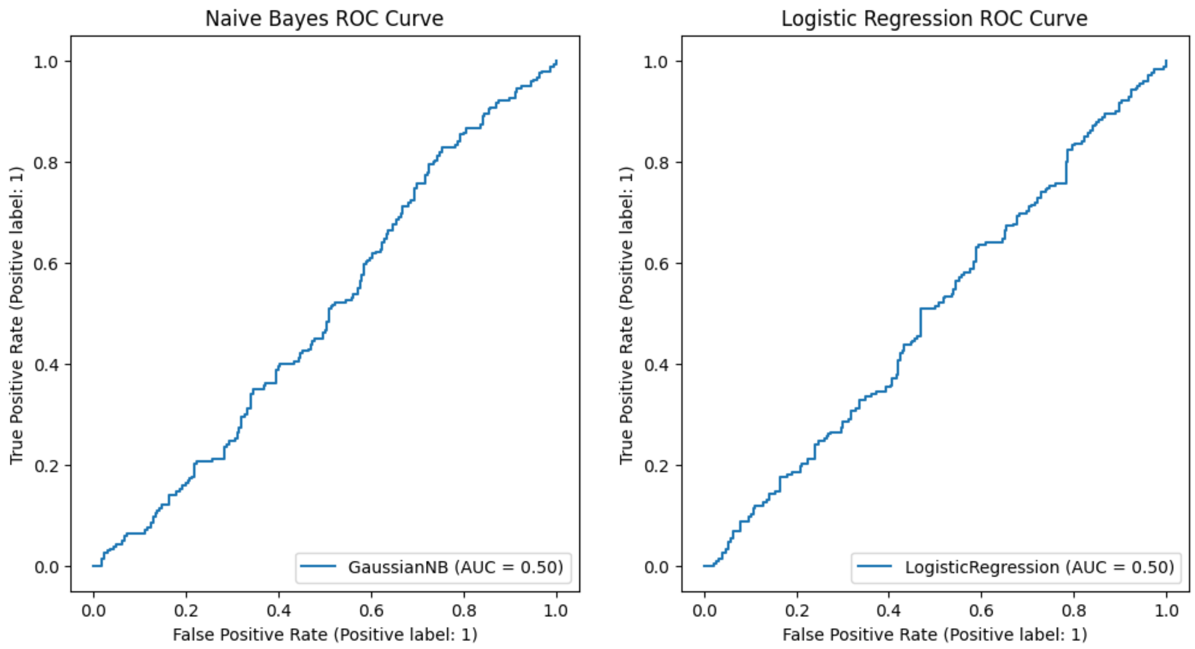


Figure 9. ROC Curve for Naïve bayes and Logistic regression

7. Market Return and Strategy Returns

Calculating the Market Returns and Strategy returns involves integrating Predicted values of Y and handling the Nan Values generated through the Models both Logistic Regression and Naïve Bayes.

Market returns have been calculated through the 'Tomorrows Returns' column in the trade dataset represents the logarithmic returns of the current day, calculated as the natural logarithm of the current closing price divided by the previous day's closing price. This column is shifted by one position to align tomorrow's returns with today's prices.

Market Return = $\frac{Closing\ Price\ today - Closing\ Price\ yesterday}{Closing\ Price\ yesterday}$

The **strategy returns** are determined by taking long positions for correctly predicted 'Y' values and short positions for incorrectly predicted 'Y' values. A 'Strategy Returns' column is initialized to 0 for floating-point values and serves as the repository for the 'Tomorrows Returns' values. If the 'Y_logreg_pred' or 'Y_nb_pred' indicates a long position, the 'Tomorrows Returns' values are stored directly in the 'Strategy Returns' column. Conversely, if the 'Y_logreg_pred' or 'Y_nb_pred' indicates a short position, the negative of the 'Tomorrows Returns' values are stored in the 'Strategy Returns' column. Where 'Y_logreg_pred' is predicted value of price rise generated by the logistic regression model and 'Y_nb_pred' is the predicted value of price rise from the naïve bayes model.

8.Cumulative Returns Analysis

Cumulative returns depict the efficacy of the of various models used in shaping various trading strategies. Cumulative returns were calculated for market returns as well as both the strategy returns of logistic regression and naïve bayes using the cumsum() method.

"Cumulative Returns," compares the performance of different investment strategies over time, measured in days. Three strategies are plotted for P&G stock: Market Returns (red line), Logistic Regression Returns (green line), and Naive Bayes Returns (blue line). The graph shows fluctuations in the performance of each strategy, with the Market Returns underperforming compared to the Logistic Regression and Naive Bayes strategies, which appear to follow a similar trend with slight variations. The time frame observed spans from around day 2100 to day 2500. The graph is likely used to analyse the effectiveness of predictive models in finance, such as Logistic Regression and Naive Bayes, against the actual market performance.

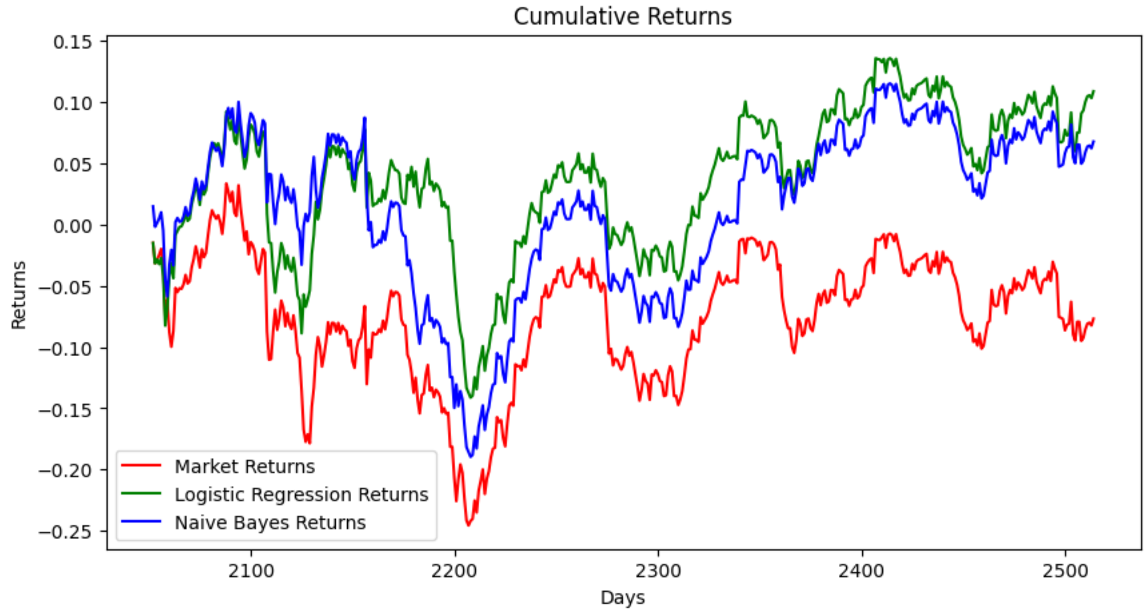


Figure 10. Cumulative Returns for Market, Naïve bayes and Logistic regression

9. Interpretation

The cumulative returns analysis provides insights into the performance of different investment strategies for Procter & Gamble (P&G) stock. The key findings are:

1. Market Returns (Red Line): The market returns, representing the actual P&G stock performance, show a relatively flat or underperforming trend compared to the other strategies over the observed period.
2. Logistic Regression Returns (Green Line): The Logistic Regression strategy outperforms the market returns, with a steadily increasing cumulative returns

trend, suggesting the model's ability to identify and capitalize on price rise patterns.

3. Naive Bayes Returns (Blue Line): Like Logistic Regression, the Naive Bayes strategy also outperforms the market, with a closely aligned cumulative returns trend, indicating both models captured similar price movement patterns.

The analysis suggests the predictive models, Logistic Regression and Naive Bayes, generated higher cumulative returns than the market where logistic regression led the way in cumulative returns whereas Naïve Bayes model had a better accuracy, implying their success in identifying and profiting from price rise patterns in the P&G stock.

10. Conclusion

The analysis evaluated the performance of Logistic Regression and Naive Bayes models in predicting stock price movements for Procter & Gamble (P&G). The Naive Bayes model achieved slightly higher average accuracy of 51.3% compared to Logistic Regression at 50.26% across cross-validation folds, suggesting better predictive capabilities.

The cumulative returns analysis showed both predictive models outperformed the market returns, indicating their ability to identify and capitalize on price rise patterns in the P&G stock. However, the models' limited predictive power, only marginally better than random guessing, and the lack of consideration for real-world constraints highlight the challenges of using machine learning for accurate stock price forecasting.

The findings provide valuable insights into the complexities involved in developing effective stock price prediction models and the need for further research to improve their practical applicability.

11. References

A.Jabbar Alkubaisi, G.A., Kamaruddin, S.S. and Husni, H. (2018). Stock Market Classification Model Using Sentiment Analysis on Twitter Based on Hybrid Naive Bayes Classifiers. *Computer and Information Science*, 11(1), p.52. doi:https://doi.org/10.5539/cis.v11n1p52.

Beckers, S. (1981). Standard deviations implied in option prices as predictors of future stock price variability. *Journal of Banking & Finance*, 5(3), pp.363–381. doi:https://doi.org/10.1016/0378-4266(81)90032-7.

Chan, L. and Lien, D. (2003). Using high, low, open, and closing prices to estimate the effects of cash settlement on futures prices. *International Review of Financial Analysis*, 12(1), pp.35–47. doi:https://doi.org/10.1016/s1057-5219(02)00125-4.

Dodd, E.L. (1941). The Problem of Assigning a Length to the Cycle to be Found in a Simple Moving Average and in a Double Moving Average of Chance Data. *Econometrica*, 9(1), p.25. doi:https://doi.org/10.2307/1907172.

Granholm, V., Noble, W. and Käll, L. (2012). A cross-validation scheme for machine learning algorithms in shotgun proteomics. *BMC Bioinformatics*, 13(Suppl 16), p.S3. doi:https://doi.org/10.1186/1471-2105-13-s16-s3.

Hong, C.-S. and Lee, W.-Y. (2011). ROC Curve Fitting with Normal Mixtures. *Korean Journal of Applied Statistics*, 24(2), pp.269–278. doi:https://doi.org/10.5351/kjas.2011.24.2.269.

Klinker, F. (2010). Exponential moving average versus moving exponential average. *Mathematische Semesterberichte*, 58(1), pp.97–107. doi:https://doi.org/10.1007/s00591-010-0080-8.

Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticæ Investigationes. International Journal of Linguistics and Language Resources*, 30(1), pp.3–26. doi:https://doi.org/10.1075/li.30.1.03nad.

Parker, J.R. (2001). Rank and response combination from confusion matrix data. *Information Fusion*, 2(2), pp.113–120. doi:https://doi.org/10.1016/s1566-2535(01)00030-6.

Procter & Gamble (2023). *P&G History*. [online] us.pg.com. Available at: https://us.pg.com/pg-history/ [Accessed 2 Apr. 2024].

Stoltzfus, J.C. (2011). Logistic Regression: A Brief Primer. *Academic Emergency Medicine*, 18(10), pp.1099–1104. doi:https://doi.org/10.1111/j.1553-2712.2011.01185.x.

Yang, D. and Zhang, Q. (2000). Drift Independent Volatility Estimation Based on High, Low, Open, and Close Prices. *The Journal of Business*, 73(3), pp.477–492. doi:https://doi.org/10.1086/209650.

12. Appendix

Github Link for the code:
https://github.com/conquerorpulkit/AIML/blob/main/AI_CW1.ipynb

Data Collection

```
from yahooquery import Ticker
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.metrics import classification_report,ConfusionMatrixDisplay,RocCurveDisplay, mean_absolute_error
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.naive_bayes import GaussianNB
import seaborn as sns
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
pd.options.mode.chained_assignment = None
```

[1] ✓ 2.2s

```
nvda = Ticker('PG')
df=nvda.history(start='2014-01-01', end='2023-12-31', period='max')
df
```

[2] ✓ 4.1s

		open	high	low	close	volume	adjclose	dividends
symbol	date							
PG	2014-01-02	81.330002	81.360001	80.320000	80.540001	6981700	60.028904	0.0
	2014-01-03	80.760002	80.849998	80.190002	80.449997	6925600	59.961800	0.0
	2014-01-06	80.610001	80.980003	80.300003	80.639999	7208200	60.103405	0.0
	2014-01-07	80.709999	81.580002	80.620003	81.419998	7158200	60.684772	0.0
	2014-01-08	80.970001	81.150002	80.050003	80.239998	13458800	59.805271	0.0
	...	...	...	...	...	...	...	...
	2023-12-22	144.500000	145.630005	144.289993	145.279999	4412800	144.368240	0.0
	2023-12-26	145.089996	146.169998	144.970001	145.940002	3634900	145.024109	0.0
	2023-12-27	145.649994	146.309998	145.360001	146.059998	4569400	145.143356	0.0
	2023-12-28	146.000000	146.009995	145.039993	145.729996	5023000	144.815414	0.0
	2023-12-29	146.000000	146.960007	145.729996	146.539993	5300900	145.620331	0.0

2516 rows × 7 columns

Feature Engineering

```
dataset = df.dropna()
dataset = dataset[['open', 'high', 'low', 'close']]

dataset['H-L'] = dataset['high'] - dataset['low']
dataset['O-C'] = dataset['close'] - dataset['open']
dataset['20day MA'] = dataset['close'].shift(1).rolling(window = 20).mean()
dataset['200day MA'] = dataset['close'].shift(1).rolling(window = 200).mean()
dataset['20day EMA'] = dataset['close'].shift(1).ewm(span=5, adjust=True).mean()


dataset['Std_dev']= dataset['close'].rolling(5).std()

dataset['Price_Rise'] = np.where(dataset['close'].shift(-1) > dataset['close'], 1, 0)
dataset = dataset.dropna()
dataset.head()
```

✓ 0.0s

	open	high	low	close	H-L	O-C	20day MA	200day MA	20day EMA	Std_dev	Price_Rise
200	82.949997	83.480003	82.029999	83.269997	1.450005	0.320000	83.870001	80.65920	83.056301	0.515637	1
201	83.250000	84.330002	82.830002	84.180000	1.500000	0.930000	83.810001	80.67285	83.127533	0.718451	1
202	84.470001	84.639999	83.660004	84.610001	0.979996	0.139999	83.778501	80.69150	83.478356	0.951973	0
203	84.059998	84.589996	83.919998	84.230003	0.669998	0.170006	83.787001	80.71135	83.855571	0.955999	0
204	84.260002	84.330002	82.300003	83.230003	2.029999	-1.029999	83.736501	80.72540	83.980381	0.619904	1

EDA

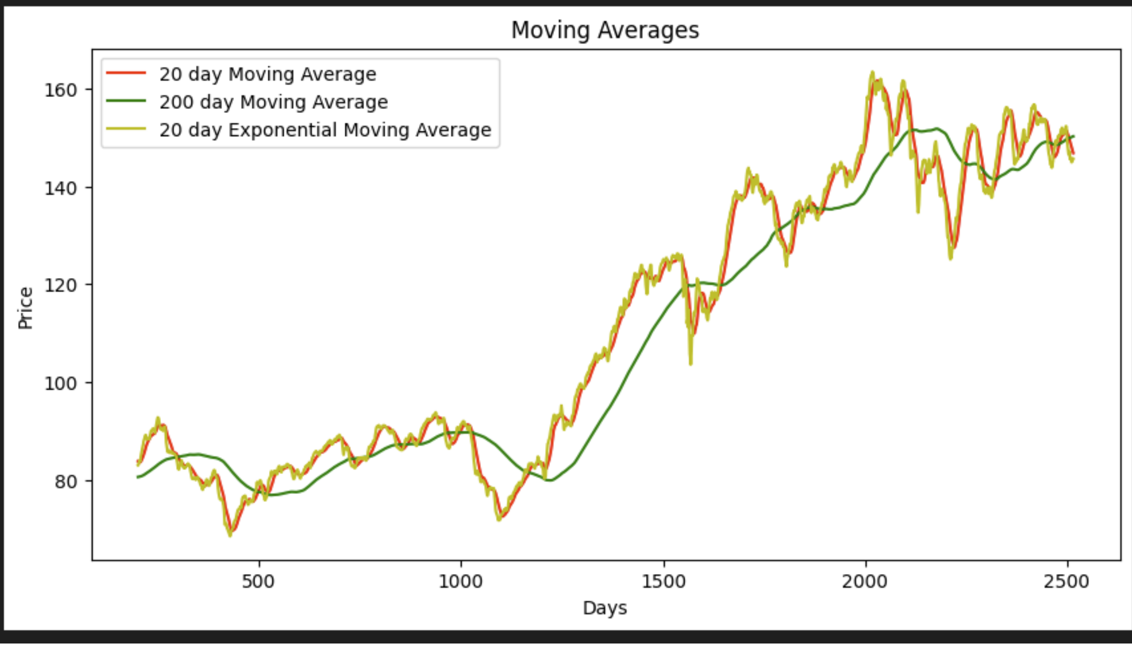
```
dataset.describe()
```

[6] ✓ 0.0s

	open	high	low	close	H-L	O-C	20day MA	200day MA	20day EMA	Std_dev	Price_Rise
count	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000	2316.000000
mean	111.494227	112.326598	110.703618	111.529948	1.622979	0.035721	111.248074	108.561624	111.448503	1.012523	0.522021
std	28.347177	28.616011	28.093311	28.351195	1.148218	1.158254	28.231660	27.027643	28.308764	0.831379	0.499623
min	68.019997	68.300003	65.019997	68.059998	0.309998	-9.630005	69.643000	76.945850	68.553085	0.053197	0.000000
25%	85.122498	85.577501	84.589996	85.157503	0.880005	-0.440002	84.892374	84.395725	85.190814	0.492779	0.000000
50%	105.985001	106.875000	105.200001	106.095001	1.330002	0.070000	105.737251	93.723675	105.687053	0.788942	1.000000
75%	139.240002	140.434994	138.199997	139.264996	2.002499	0.572504	139.753625	135.741300	139.286132	1.257656	1.000000
max	164.399994	165.350006	163.399994	164.210007	14.949997	7.570000	161.622000	151.743700	163.404134	9.088996	1.000000

```python
plt.figure(figsize=(10,5))
plt.plot(dataset['20day MA'], label='20 day Moving Average', color='r')
plt.plot(dataset['200day MA'], label='200 day Moving Average', color='g')
plt.plot(dataset['20day EMA'], label='20 day Exponential Moving Average', color='y')
plt.legend()
plt.title('Moving Averages')
plt.xlabel('Days')
plt.ylabel('Price')
plt.show()
```

✓ 0.1s



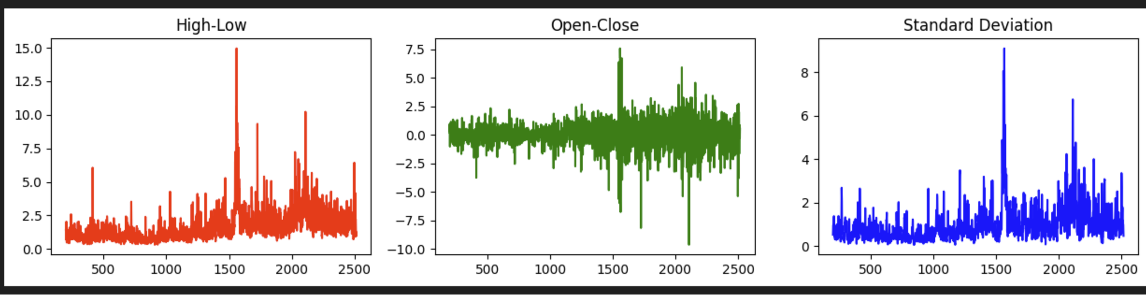

```python
fig, axes = plt.subplots(1, 3, figsize=(15, 3))
axes[0].plot(dataset['H-L'], label='High-Low', color='r')
axes[0].set_title('High-Low')

axes[1].plot(dataset['O-C'], label='Open-Close', color='g')
axes[1].set_title('Open-Close')

axes[2].plot(dataset['Std_dev'], label='Standard Deviation', color='b')
axes[2].set_title('Standard Deviation')
```

✓ 0.3s

```
Text(0.5, 1.0, 'Standard Deviation')
```



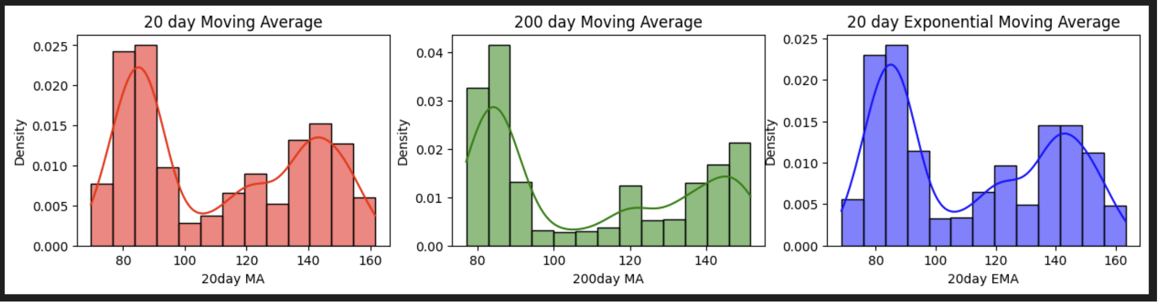
```
fig, axes = plt.subplots(1, 3, figsize=(15, 3))
sns.histplot(dataset,x="20day MA", kde=True, stat="density", ax=axes[0], color='r')
axes[0].set_title('20 day Moving Average')

sns.histplot(dataset,x="200day MA", kde=True, stat="density", ax=axes[1], color='g')
axes[1].set_title('200 day Moving Average')

sns.histplot(dataset,x="20day EMA", kde=True, stat="density", ax=axes[2], color='b')
axes[2].set_title('20 day Exponential Moving Average')
```

[9] ✓ 0.4s

```
Text(0.5, 1.0, '20 day Exponential Moving Average')
```

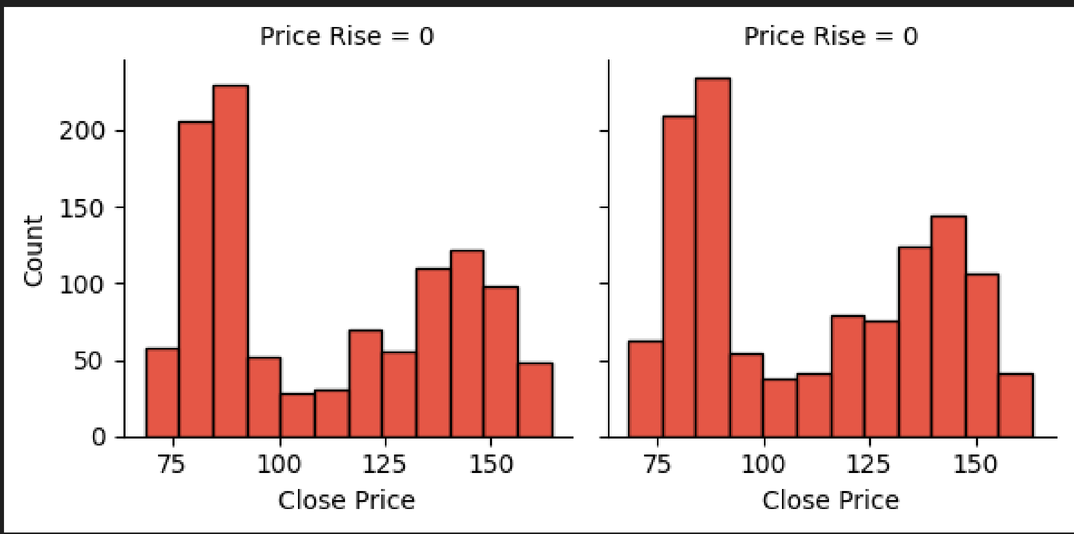


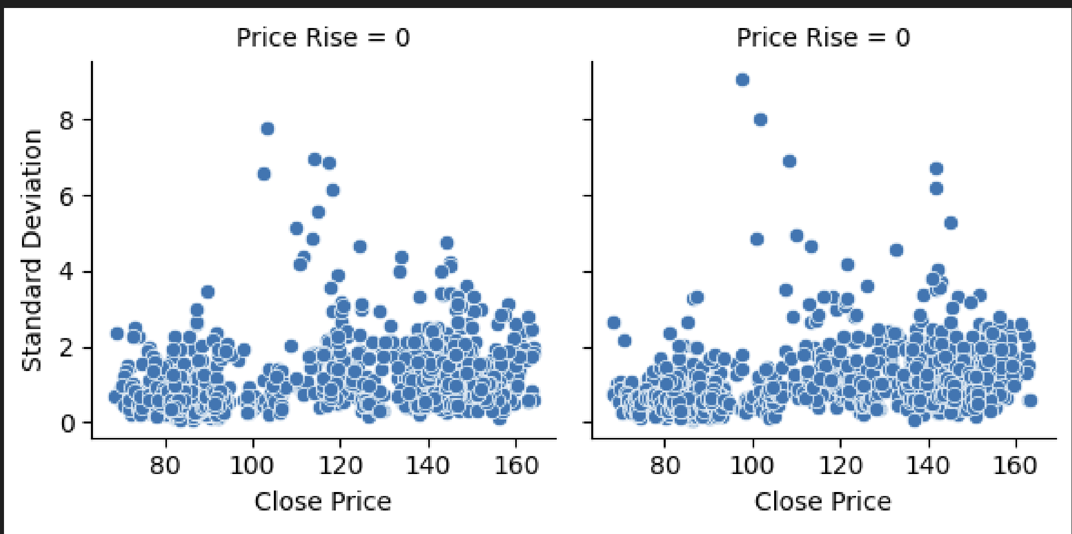
```python
chart = sns.FacetGrid(dataset, col='Price_Rise')
chart.map(sns.histplot, 'close', color='r')
chart.set_axis_labels('Close Price', 'Count')
chart.set_titles('Price Rise = 0', 'Price Rise = 1')

chart = sns.FacetGrid(dataset, col='Price_Rise')
chart.map(sns.scatterplot, 'close', 'Std_dev')
chart.set_axis_labels('Close Price', 'Standard Deviation')
chart.set_titles('Price Rise = 0', 'Price Rise = 1')
```

[10] ✓ 0.6s

```
<seaborn.axisgrid.FacetGrid at 0x286e28f53d0>
```





Machine Learning Classification methods

Pre-processing

```
X = dataset.iloc[:, 4:-1]
y = dataset.iloc[:, -1]
X
```

[11] ✓ 0.0s

	H-L	O-C	20day MA	200day MA	20day EMA	Std_dev
200	1.450005	0.320000	83.870001	80.659200	83.056301	0.515637
201	1.500000	0.930000	83.810001	80.672850	83.127533	0.718451
202	0.979996	0.139999	83.778501	80.691500	83.478356	0.951973
203	0.669998	0.170006	83.787001	80.711350	83.855571	0.955999
204	2.029999	-1.029999	83.736501	80.725400	83.980381	0.619904
...	...	...	...	...	...	...
2511	1.340012	0.779999	147.940501	150.037051	144.908225	1.052031
2512	1.199997	0.850006	147.635500	150.080601	145.032149	1.000485
2513	0.949997	0.410004	147.370500	150.124351	145.334767	0.972214
2514	0.970001	-0.270004	147.059000	150.163951	145.576511	0.730673
2515	1.230011	0.539993	146.789000	150.193351	145.627672	0.460867

2316 rows × 6 columns

```
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
X_test
```

[12] ✓ 0.0s

	H-L	O-C	20day MA	200day MA	20day EMA	Std_dev
2052	6.690002	5.910004	158.918501	146.119700	155.717993	3.124855
2053	3.369995	-0.399994	158.874001	146.233200	156.558664	2.584771
2054	3.059998	-1.000000	158.643501	146.324050	156.335776	2.527843
2055	2.479996	0.829987	158.286501	146.400550	155.327183	2.514934
2056	2.089996	0.589996	157.989001	146.480850	154.814786	1.996869
...	...	...	...	...	...	...
2511	1.340012	0.779999	147.940501	150.037051	144.908225	1.052031
2512	1.199997	0.850006	147.635500	150.080601	145.032149	1.000485
2513	0.949997	0.410004	147.370500	150.124351	145.334767	0.972214
2514	0.970001	-0.270004	147.059000	150.163951	145.576511	0.730673
2515	1.230011	0.539993	146.789000	150.193351	145.627672	0.460867

464 rows × 6 columns

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[13] ✓ 0.0s

Logistic Regression

LOGISTIC REGRESSION

```
logreg = LogisticRegression(max_iter=1000000)
logreg.fit(X_train, Y_train)
Y_logreg_pred = logreg.predict(X_test)
print (classification_report(Y_test, Y_logreg_pred))
```

[14] ✓ 0.0s

```
              precision    recall  f1-score   support

           0       0.52      0.10      0.16       226
           1       0.52      0.92      0.66       238

    accuracy                           0.52       464
   macro avg       0.52      0.51      0.41       464
weighted avg       0.52      0.52      0.42       464
```

Naïve Bayes

NAIVE BAYES

```python
nb = GaussianNB()
nb.fit(X_train, Y_train)
Y_nb_pred = nb.predict(X_test)
print (classification_report(Y_test, Y_nb_pred))
```

[23] ✓ 0.0s

```
              precision    recall  f1-score   support

           0       0.59      0.12      0.20       226
           1       0.52      0.92      0.67       238

    accuracy                           0.53       464
   macro avg       0.56      0.52      0.43       464
weighted avg       0.55      0.53      0.44       464
```

Confusion Matrix

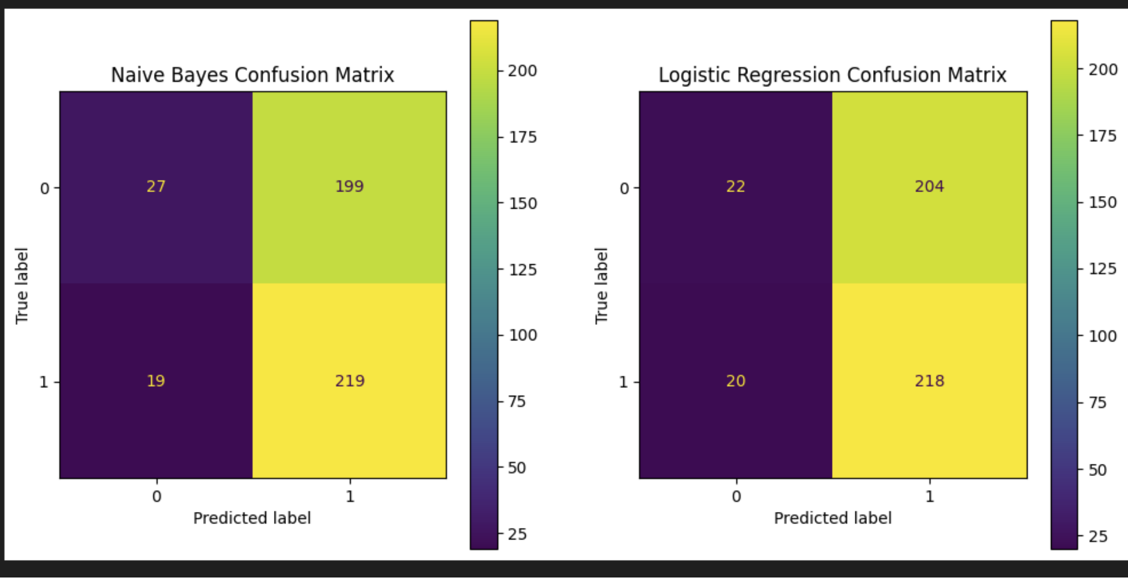
```python
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Confusion matrix for nb
nb_matrix = ConfusionMatrixDisplay.from_estimator(nb, X_test, Y_test, ax=axes[0])
axes[0].set_title('Naive Bayes Confusion Matrix')

# Confusion matrix for logreg
logreg_matrix = ConfusionMatrixDisplay.from_estimator(logreg, X_test, Y_test, ax=axes[1])
axes[1].set_title('Logistic Regression Confusion Matrix')

plt.show()
```

[24] ✓ 0.2s



ROC Curve

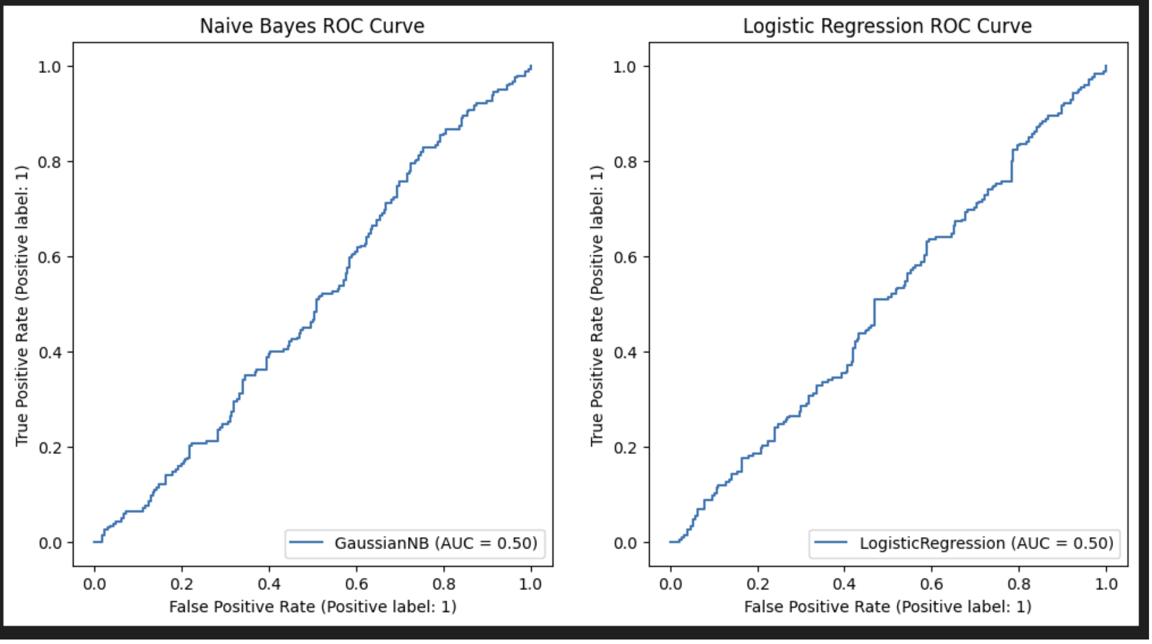
```
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# ROC curve for nb
nb_disp = RocCurveDisplay.from_estimator(nb, X_test, Y_test, ax=axes[0])
axes[0].set_title('Naive Bayes ROC Curve')

# ROC curve for logreg
log_disp = RocCurveDisplay.from_estimator(logreg, X_test, Y_test, ax=axes[1])
axes[1].set_title('Logistic Regression ROC Curve')

plt.show()
```

[25] ✓ 0.2s



Cross Validation

```
# Cross-validation
log_reg_scores = cross_val_score(logreg, X_train, Y_train, cv=5)
extra_trees_scores = cross_val_score(classifier, X_train, Y_train, cv=5)
nb_scores = cross_val_score(nb, X_train, Y_train, cv=5)

# Print cross-validation results
print("Logistic Regression mean accuracy:", log_reg_scores.mean())
print("Logistic Regression std deviation:", log_reg_scores.std())
print("Naive Bayes mean accuracy:", nb_scores.mean())
print("Naive Bayes std deviation:", nb_scores.std())
```

[34] ✓ 1.5s

```
Logistic Regression mean accuracy: 0.502688132876812
Logistic Regression std deviation: 0.03448922487446926
Naive Bayes mean accuracy: 0.5135207984264587
Naive Bayes std deviation: 0.026693822516964234
```

Market and Strategy Returns

Pre-processing

```
dataset['Y_logreg_pred'] = np.NaN
dataset.iloc[(len(dataset) - len(Y_logreg_pred)):,-1] = Y_logreg_pred
dataset['Y_nb_pred'] = np.NaN
dataset.iloc[(len(dataset) - len(Y_nb_pred)):,-1] = Y_nb_pred
trade_dataset = dataset.dropna()
trade_dataset
```

[27] ✓ 0.0s

	open	high	low	close	H-L	O-C	20day MA	200day MA	20day EMA	Std_dev	Price_Rise	Y_logreg_pred	Y_nb_pred
2052	152.330002	158.940002	152.250000	158.240005	6.690002	5.910004	158.918501	146.119700	155.717993	3.124855	0	1.0	0.0
2053	156.289993	157.190002	153.820007	155.889999	3.369995	-0.399994	158.874001	146.233200	156.558664	2.584771	0	1.0	1.0
2054	154.309998	155.399994	152.339996	153.309998	3.059998	-1.000000	158.643501	146.324050	156.335776	2.527843	1	1.0	1.0
2055	152.960007	155.080002	152.600006	153.789993	2.479996	0.829987	158.286501	146.400550	155.327183	2.514934	1	0.0	1.0
2056	153.770004	155.860001	153.770004	154.360001	2.089996	0.589996	157.989001	146.480850	154.814786	1.996869	1	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2511	144.500000	145.630005	144.289993	145.279999	1.340012	0.779999	147.940501	150.037051	144.908225	1.052031	1	1.0	1.0
2512	145.089996	146.169998	144.970001	145.940002	1.199997	0.850006	147.635500	150.080601	145.032149	1.000485	1	1.0	1.0
2513	145.649994	146.309998	145.360001	146.059998	0.949997	0.410004	147.370500	150.124351	145.334767	0.972214	0	1.0	1.0
2514	146.000000	146.009995	145.039993	145.729996	0.970001	-0.270004	147.059000	150.163951	145.576511	0.730673	1	1.0	1.0
2515	146.000000	146.960007	145.729996	146.539993	1.230011	0.539993	146.789000	150.193351	145.627672	0.460867	0	1.0	1.0

464 rows × 13 columns

Market Returns Calculation

```
trade_dataset['Tomorrows Returns'] = 0.
trade_dataset['Tomorrows Returns'] = np.log(trade_dataset['close']/trade_dataset['close'].shift(1))
trade_dataset['Tomorrows Returns'] = trade_dataset['Tomorrows Returns'].shift(-1)
```

[28] ✓ 0.0s

Strategy Returns of Logistic Regression

```
trade_dataset['Strategy Logistic Returns'] = 0.
trade_dataset['Strategy Logistic Returns'] = np.where(trade_dataset['Y_logreg_pred'] == True, trade_dataset['Tomorrows Returns'], - trade_dataset['Tomorrows Returns'])
```

[29] ✓ 0.0s

Strategy Returns of Naive Bayes

```
trade_dataset['Strategy Naive Bayes Returns'] = 0.
trade_dataset['Strategy Naive Bayes Returns'] = np.where(trade_dataset['Y_nb_pred'] == True, trade_dataset['Tomorrows Returns'], - trade_dataset['Tomorrows Returns'])
```

[31] ✓ 0.0s

Cumulative Returns

```
trade_dataset['Cumulative Market Returns'] = np.cumsum(trade_dataset['Tomorrows Returns'])
trade_dataset['Cumulative Strategy Logistic Returns'] = np.cumsum(trade_dataset['Strategy Logistic Returns'])
trade_dataset['Cumulative Strategy Naive Bayes Returns'] = np.cumsum(trade_dataset['Strategy Naive Bayes Returns'])
```

[32] ✓ 0.0s

```python
plt.figure(figsize=(10,5))
plt.plot(trade_dataset['Cumulative Market Returns'], color='r', label='Market Returns')
plt.plot(trade_dataset['Cumulative Strategy Logistic Returns'], color='g', label='Logistic Regression Returns')
plt.plot(trade_dataset['Cumulative Strategy Naive Bayes Returns'], color='b', label='Naive Bayes Returns')
plt.title('Cumulative Returns')
plt.xlabel('Days')
plt.ylabel('Returns')
plt.legend()
plt.show()
```

[34] ✓ 0.1s