WESTMINSTER BUSINESS SCHOOL

Machine Learning and Random Trading. A lesson from Intesa SanPaolo S.p.A.

Module Title: **Artificial Intelligence and Machine Learning in Finance Services**

Module Code: **7FNCE043W**

Module Leader: **Yao, Yumei**

Total Words Count without Tables, Appendices and References: **1954 words.**

Semester 2, 2024/2025

Riccardo Negrisoli
03/03/2025

Table of Contents

1 Company Overview

Intesa Sanpaolo S.p.A., which was formed in 1998, is headquartered in Turin (Italy). It is the backbone of the Italian banking industry since it offers a wide range of financial products and services through six business segments: Banca dei Territori, IMI Corporate and Investment Banking, International Subsidiary Banks, Asset Management, Private Banking, and Insurance. The bank's wide portfolio unequivocally includes lending and deposit operations, structured finance, corporate and investment banking, as well as full-service asset and wealth management products. Pioneering in its approach, Intesa Sanpaolo went the extra mile with digital transformation in order to deliver the best customer experience. Its digital channels, such as the Intesa Sanpaolo Mobile application and the innovative digital bank, Isybank, have secure, easy-to-use products for individuals, families, and companies.

The official Sanpaolo (2024) history page reports that the history of the Group lies in a long tradition of banking innovation and resilience dating back decades. The integration of Banca Intesa with Sanpaolo IMI in 2007 was a turning point, becoming the largest banking group in Italy while upholding its strong regional presence. This solid tradition, with continuous investments in sustainable finance and digital technology, underlies Intesa Sanpaolo's leading position in the international markets as well.

The Bank is also the primary investor in the Research Institute Collegio Carlo Alberto in Turin, Italy. Its involvement in economic and financial research positions it at the forefront of innovation in the 21st century.

2 Data Collection and Processing

The download of Intesa Sanpaolo's historical stock data, from Yahoo Finance, covers the period from January 1st 2004, to December 31st 2024 in order to have 21 full years of daily observations.

I removed missing values to ensure consistency[1]. The original columns (`Open`, `High`, `Low`, `Close`, `Volume`) are retained.

2.1 Feature Engineering and Explanatory Data Analysis

Next, several feature-engineering steps are applied, as well as Data analysis of these features.

(1) `H-L` (High minus Low) and `O-C` (Close minus Open) are calculated to capture daily price ranges and intraday shifts; Karpoff (1987) discussed the inverse relationship between stock prices

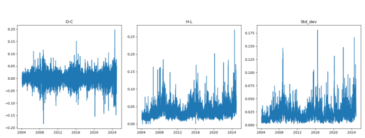


Figure 1: O-C, H-L, and Std_dev Trends Over Time.

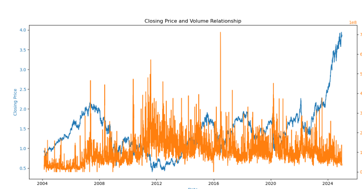


Figure 2: Closing Price (blue) and Volume (orange) Over Time.

and the volume exchanged under specific market conditions. In my analysis, this result seems to be proved (Fig 4).

[1]This step is included in the code for completeness. In practice it was not necessary due to the lack of any missing value.

(2) Three moving averages of closing price (`3day MA`, `10day MA`, `30day MA`) are created to reveal short- and medium-term trends without introducing look-ahead bias.

As we can see from figure 3, all the MA series are highly correlated. Because of this, including all of them in a machine learning model can introduce redundancy and multicollinearity. In other words, each moving average provides overlapping information about the same underlying price data. This redundancy can distort feature-importance measures and/or may deteriorate predictive performance. In addition, Intesa Sanpaolo has demonstrated strong resilience in the 21st century, showing a solid ability to increase its value after the COVID-19 pandemic.

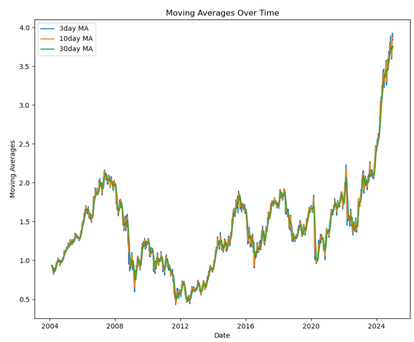


Figure 3: Moving Averages (3-day, 10-day, 30-day) Over Time.

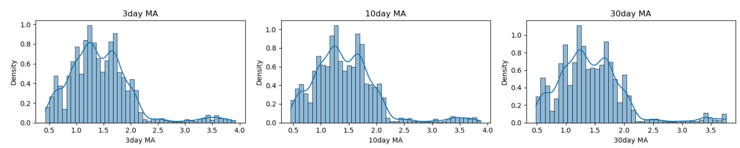


Figure 4: Histograms and Density Estimates of 3-day, 10-day, and 30-day MAs.

Figure 4 shows histograms and kernel density estimates for three moving averages applied to the same underlying time series. Because the 3-day MA reacts quickly to price changes, it yields a broader variable distribution, while the 30-day MA smooths out short-term fluctuations and produces a more peaked shape. The 10-day MA sits between these extremes, smoothing short term dynamics and capturing medium term ones. As expected, all three panels capture the same fundamental behavior.

(3) A rolling standard deviation (`Std_dev`) is derived over a five-day window to reflect short-term volatility. The code also computes daily log-returns, defined as the natural logarithm of the ratio of consecutive closing prices, to stabilize the variance and reduce non-stationarity. Before the analysis of the volatility, it is necessary to make sure computed returns are a stationary series. In figure 5 is clear that returns move around zero.

A Generalized AutoRegressive Conditional Heteroskedasticity (GARCH) model is used to model time-varying volatility and the conditional variance. It assumes that the return series is at least weakly stationary so that the conditional variance process is well-defined and stable over time. If the returns are non-stationary, the GARCH parameters and the estimated variance dynamics can become unreliable or even meaningless. As we have seen the first moment (i.e. the mean) of the returns is constant over time. (`GARCH_t_variance`) is appended to the dataset.

Fitting a Garch (1,1) model with a t-distribution allows us to capture the volatility of the stocks removing excessive sensitivity to the outliers [2]. Figure 6 shows us the clusters of volatility in our data, reflecting the volatility clustering phenomenon commonly observed in financial time series (Engle, 1982).

[2] This is clear since the t-distributions allows return to have fatter tails.

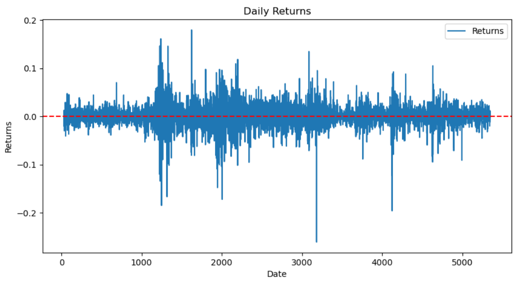


Figure 5: Daily Returns (with zero baseline in red).

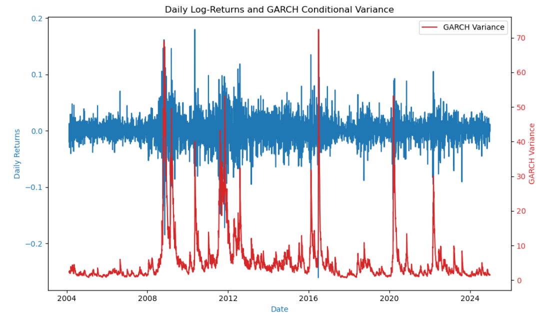


Figure 6: Daily Log-Returns (blue) and GARCH(1,1) Conditional Variance (red).

2.2 Target Variable

A binary target variable (`Price_Rise`) indicates whether the following day's closing price is higher than the current day's closing price. All rows with newly introduced `NaN` values from these rolling operations are dropped.

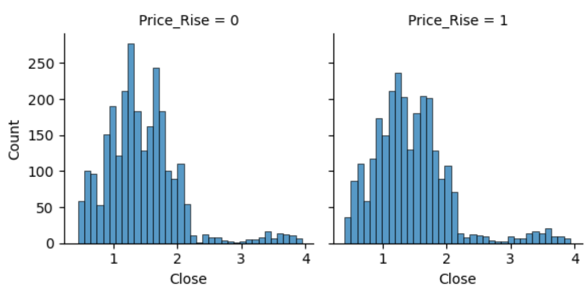


Figure 7: Distribution of Closing Prices for Rising vs. Non-Rising Days.

Figure 8: Scatter of Close vs. Std_dev, Separated by Price Rise Categories.

In the first pair of histograms (Fig 7), both distributions peak near the same "Close" value, but when the price rises (Price_Rise = 1) there is a slightly greater concentration of higher "Close" values. This suggests that larger closing prices are somewhat more likely to be associated with a price increase on the following day.

Figure 8 shows the second pair of scatter plots, where the "Std_dev" is plotted against "Close" for the two Price_Rise categories . Both clouds mostly overlap, even though the points for Price_Rise = 1 tend to have lower standard deviations at higher price levels, indicating that when prices are already high and continue to rise, the volatility may be comparatively lower (Nelson, 1991).

2.3 A final Overview of all the features

From the correlation matrix, we observe that `Open`, `High`, `Low`, and `Close` are highly correlated, as expected for price-related variables. Similarly, the moving averages (`3day MA`, `10day MA`, and `30day MA`) show strong mutual correlations and closely follow the main price indicators. In contrast, `Volume` and `Returns` show weaker correlations with price variables, suggesting that trading activity and daily return fluctuations do not always move in sync with raw prices. The binary indicator `Price_Rise` has only a modest correlation with other features, indicating that price increases on a given day depend on multiple factors rather than a single variable.

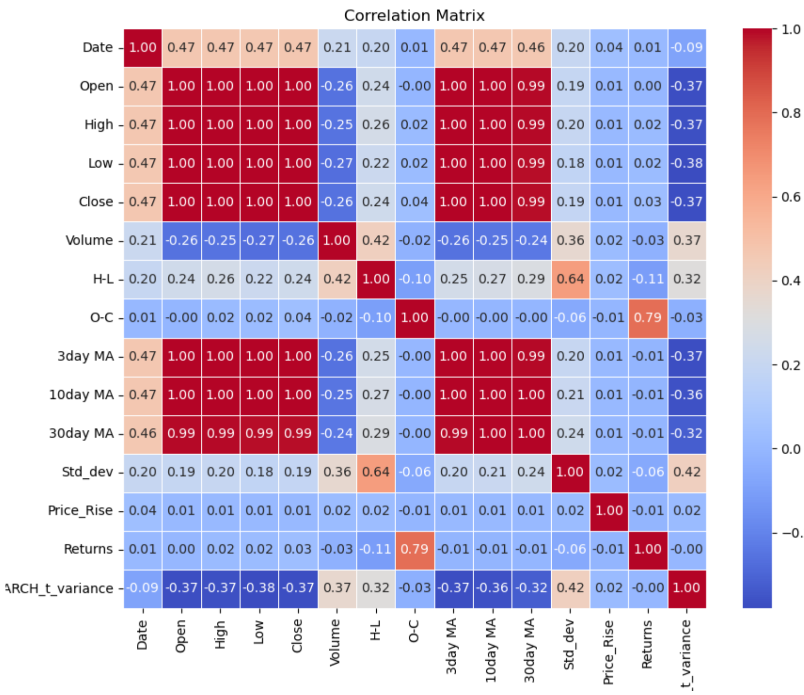


Figure 9: Correlation Matrix of All Features.

3 Machine Learning Analysis

Several classification models were trained to predict whether Intesa Sanpaolo's stock price would rise on the next trading day. Specifically, I examined Logistic Regression and Extra Trees (Gaussian Naive Bayes was also performed. See Appendix 9). All models used the same feature set: `Volume`, `H-L` (High minus Low), `O-C` (Close minus Open), `Std_dev` (5-day rolling standard deviation of closing price), and `3day MA`. After splitting the dataset into training (85%) and testing (15%)[3], `X_train` and `X_test` have been scaled using standard transformation.

The theoretical fundations of the ML models and Cross-Validation can be found in Appendix 9

4 Confusion Matrices and Classification Reports

4.1 Logistic Regression

The confusion matrix for Logistic Regression (Fig 18) shows relatively balanced true negatives (correctly predicting "Down") and true positives (correctly predicting "Up"), but also the misclassifications (false positives and false negatives) are substantial.

The classification report (Fig 19) shows that the model's overall accuracy is around 0.51, only slightly above random guessing. Class 0 is identified more effectively in terms of recall, meaning the model catches most of the actual class 0 instances but also misclassifies a fair number of them. Conversely, class 1 has higher precision but lower recall, indicating that while the model is more selective when predicting class 1, it fails to capture many true positives in that category. The macro-averaged F_1 score (0.50) confirms that the model's performance is relatively balanced between the

[3]This proportion has been checked and it appears to be the more robust

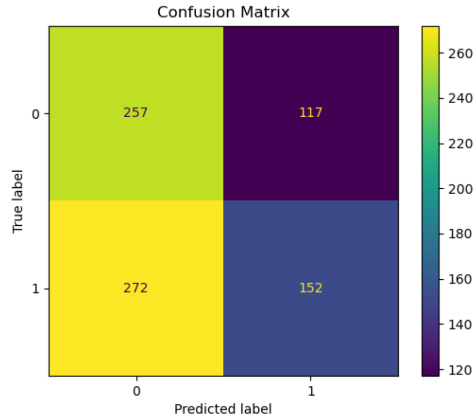


Figure 10: Confusion Matrix for Logistic Regression

Logistic Regression Classification Report:

	precision	recall	f1-score	support
0	0.49	0.69	0.57	374
1	0.57	0.36	0.44	424
accuracy			0.51	798
macro avg	0.53	0.52	0.50	798
weighted avg	0.53	0.51	0.50	798

Figure 11: Classification Report for Logistic Regression

two classes.

4.2 Extra Trees Classifier

The Extra Trees Classifier exhibits a similar pattern. The distribution of misclassifications is again large, resulting in overall accuracy close to 50%. Despite Extra Trees being a method capable of capturing nonlinearities, the cross-validation scores confirm that it does not outperform the baseline random prediction in this particular application (see Fig 14).

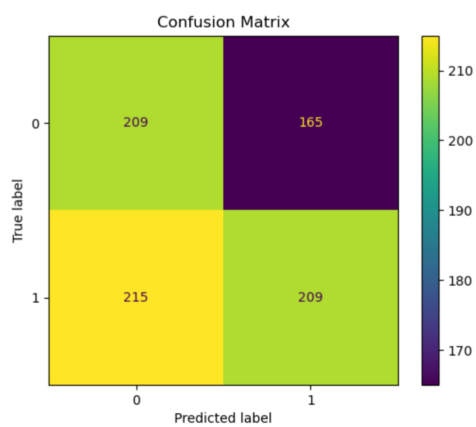


Figure 12: Confusion Matrix for Extra Trees

Extra Trees Classification Report:

	precision	recall	f1-score	support
0	0.49	0.57	0.53	374
1	0.55	0.47	0.51	424
accuracy			0.52	798
macro avg	0.52	0.52	0.52	798
weighted avg	0.52	0.52	0.52	798

Figure 13: Classification Report for Extra Trees

5 Cross-Validation and ROC Curves

Cross-validation using five folds (cv=5) for both Logistic Regression and Extra Trees confirms that mean accuracies remain around 0.50, with standard deviations indicating that performance fluctuates but does not clearly surpass random guessing.

The ROC curve illustrates a model's performance across different classification thresholds. Specifically, it plots the True Positive Rate (how many positive cases are correctly identified) against

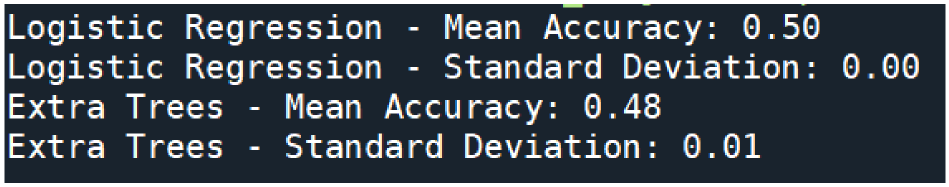

Logistic Regression - Mean Accuracy: 0.50
Logistic Regression - Standard Deviation: 0.00
Extra Trees - Mean Accuracy: 0.48
Extra Trees - Standard Deviation: 0.01

Figure 14: Cross-Validation for Logistic Regression and Extra Trees

the False Positive Rate (how many negative cases are incorrectly classified as positive). In my analysis, the ROC curves for both models exhibit Area Under the Curve (AUC) values near 0.52–0.54, highlighting minimal discriminative power. The models are more or less as good as random guessing.

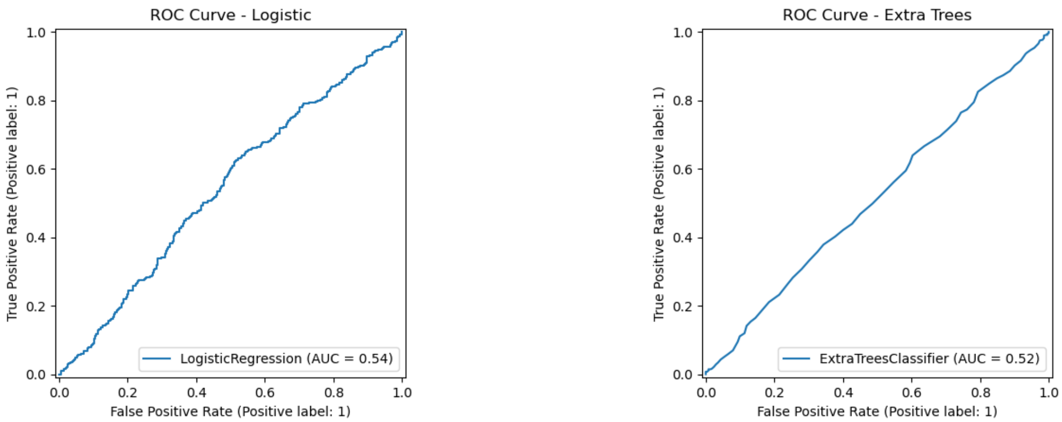


Figure 15: ROC Curves for Logistic Regression (left) and Extra Trees (right).

6 Feature Importance and Class-Conditional Means

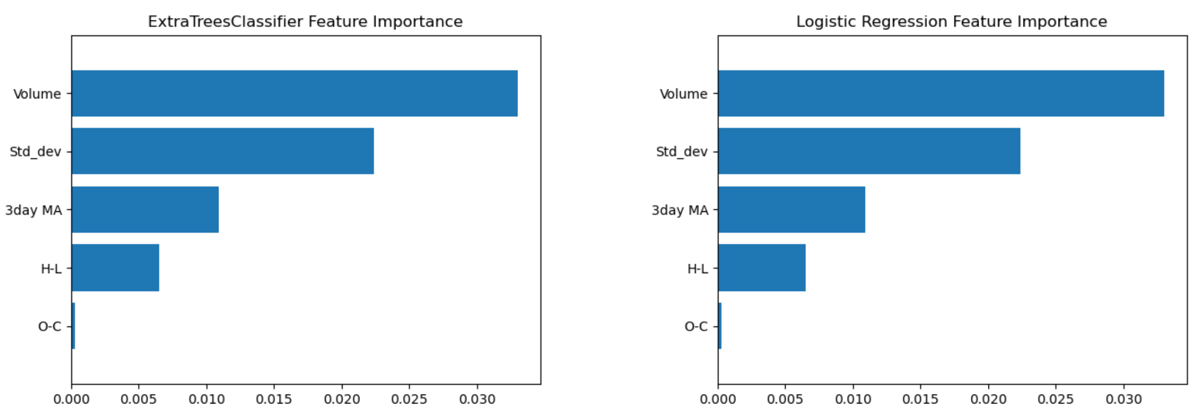


Figure 16: Feature Importance for Extra Trees (left) and Logistic Regression Coefficients (right).

Both Extra Trees and Logistic Regression highlight `Volume` as the most influential predictor for deciding whether the price will rise the next day. `Std_dev` (the short-term volatility measure) also emerges as a key factor. `3day MA` is moderately important, while `H-L` and `O-C` appear to have smaller impacts. Specifically, `O-C` was removed from the regressors after it was found to be unimportant. This step improved the overall accuracy to 53% for the Logistic Regression.

7 Market and Strategy Cumulative Returns Analysis

In figure 17 , the red line represents the cumulative market returns calculated by summing the daily log returns of the Intesa Sanpaolo stock. The blue line represents the cumulative returns of a simple long-short strategy driven by model predictions. Specifically, if the model predicts an "Up" day, we go long and capture the market's next-day return; if it predicts a "Down" day, we effectively short the market and profit when the price falls.

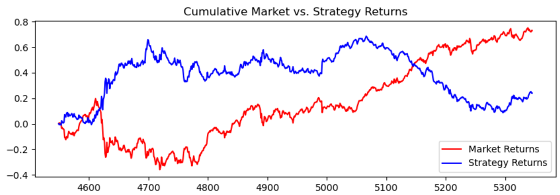


Figure 17: Cumulative Market Returns vs. Strategy Returns (Logistic Regression).

While the strategy can sometimes outperform the market, it remains quite unstable overall. This highlights how challenging it is to build a consistently profitable trading approach based solely on daily price predictions, especially when using a limited set of features and basic models. Without deeper market knowledge or more advanced techniques, short-term forecasts may not be reliable enough to ensure steady profits, particularly when factoring in real-world risks and trading costs.

8 Final Results and Literature Review

Recent studies question whether machine learning (ML) is always the best choice for predicting bank returns. For example, Makridakis et al. (2018) found that many ML models don't consistently perform better than traditional statistical methods and often struggle with problems like overfitting and limited data. Similarly, Bluwstein et al. (2021) showed that while non-linear ML models can slightly improve banking crisis predictions, their complexity often cancels out those benefits. Lastly, Antonopoulou et al. (2023) pointed out that financial markets are so unpredictable that even advanced ML models can sometimes perform no better than random guessing. These findings suggest that ML's advantages in financial forecasting are not guaranteed and should be carefully compared to simpler models.

9 Conclusion

Machine learning models struggle to reliably predict the next-day price direction of Intesa Sanpaolo's stock. The classification reports and cross-validation results consistently show accuracy staying close to 50%, with AUC values ranging from 0.52 to 0.54. While some features like `Volume`, `Std_dev`, and `3day MA` seem to have more influence, the unpredictable nature of short-term price changes and possible shifts in financial data patterns limit the models' effectiveness. This aligns with a well-known challenge in quantitative finance: predicting daily price movements is extremely difficult, and even small improvements beyond random guessing often require more complex features, larger datasets or advanced modeling techniques. These results emphasize the need to carefully test models on unseen data and be cautious about overfitting or reading too much into slight accuracy gains in noisy financial markets.

References

Antonopoulou, H. et al. (2023), ‘Utilizing machine learning to reassess the predictability of bank stocks’, *Emerging Science Journal* **7**(3), 724–736.

Bluwstein, K. et al. (2021), Credit growth, the yield curve and financial crisis prediction: Evidence from a machine learning approach, Technical Report 2614, European Central Bank.

Boschetti, A. and Massaron, L. (2018), *Python data science essentials: a practitioner’s guide covering essential data science principles, tools, and techniques*, 3rd edn, Packt Publishing, Birmingham, UK.

Cox, D. R. (1958), ‘The regression analysis of binary sequences’, *Journal of the Royal Statistical Society. Series B (Methodological)* **20**(2), 215–242.

Dixon, M., Halperin, I. and Bilokon, P. (2020), *Machine learning in finance: from theory to practice*, Springer, Switzerland.

Engle, R. F. (1982), ‘Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation’, *Econometrica* **50**(4), 987–1007.

Geurts, P., Ernst, D. and Wehenkel, L. (2006), ‘Extremely randomized trees’, *Machine Learning* **63**(1), 3–42.

Karpoff, S. (1987), ‘The relation between price changes and trading volume: A survey’, *Journal of Financial Economics* **17**(1), 153–206.

Makridakis, S., Spiliotis, E. and Assimakopoulos, V. (2018), ‘Statistical and machine learning forecasting methods: Concerns and ways forward’, *PLOS ONE* **13**(3), e0194889.

Nelson, D. B. (1991), ‘Conditional heteroskedasticity in asset returns: A new approach’, *Econometrica* **59**(2), 347–370.

Orlando, G., Chironna, G. and Penikas, H. (2022), A default prediction (pd) model for italian banks: An empirical analysis with econometric and machine learning approaches, *in* ‘Proceedings of the 3rd International Conference on Modern Management based on Big Data (MMBD)’.

Sanpaolo, G. I. (2024), ‘Storia’, `https://group.intesasanpaolo.com/it/chi-siamo/storia`. Accessed: 3 March 2025.

Appendix A

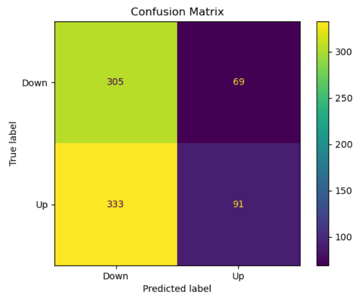
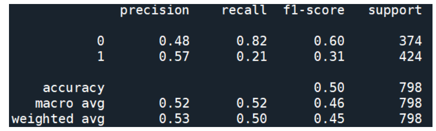


Figure 18: Confusion Matrix for Naive Bayes



	precision	recall	f1-score	support
0	0.48	0.82	0.60	374
1	0.57	0.21	0.31	424
accuracy			0.50	798
macro avg	0.52	0.52	0.46	798
weighted avg	0.53	0.50	0.45	798

Figure 19: Classification Report for Naive Bayes

Gaussian Naive Bayes also struggles to exceed 50% accuracy. These results further validate the difficulty of predicting daily direction of price movement.

In this report, class 0 has a high recall (0.82) but a low precision (0.48), indicating the model correctly identifies most class-0 cases but also misclassifies many class-1 instances as class 0. For class 1, the precision is somewhat better (0.57) but recall is very low (0.21), meaning many true positives for class 1 are missed. Overall accuracy sits at 0.50, close to random guessing, and the F_1 scores reflect that neither class is predicted reliably.

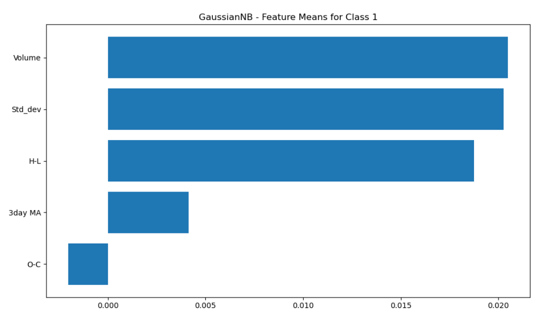


Figure 20: GaussianNB Class-Conditional Means for "Up" Class (Class 1).

Since Gaussian Naive Bayes does not directly provide feature importance via coefficients, the class-conditional means (`theta_`) for the "Up" class (Class 1) are used as a proxy. A larger mean in a specific feature suggests that if a day is classified as "Up," that feature typically has a higher average value. Here, `Volume` and `Std_dev` again rank highest, indicating that "Up" days tend to be associated with higher trading volume and greater recent volatility. `H-L` (daily range) also shows a noticeable mean, implying that upward movements often coincide with a wider trading range.

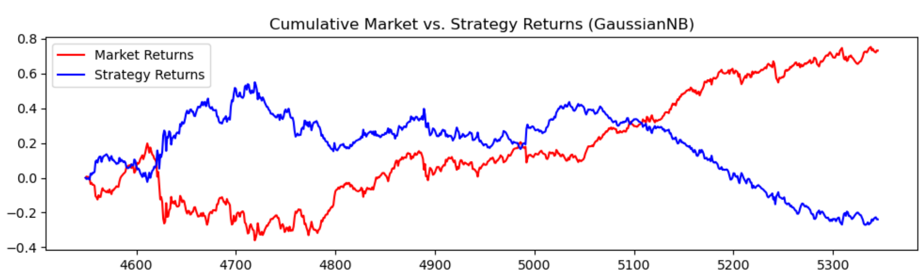


Figure 21: Cumulative Market Returns vs. Strategy Returns (GaussianNB).

Also this model is not able to beat cumulative market returns.

Appendix B

In Figure 22, the confusion matrix shows that the model very well predicts the positive label, missing most negative cases. Despite this, the ROC curve in Figure 23 yields an AUC of 0.79, suggesting that adjusting the classification threshold could improve separation between the two volatility regimes. Figure 24 indicates a nearly balanced count of correct vs. incorrect predictions, consistent with around 50% accuracy. Finally, Figure 25 reveals that *Volume* dominates the feature importance, overshadowing other inputs in driving the model's volatility-regime predictions. This feature is a pure novelty of this proposed research.

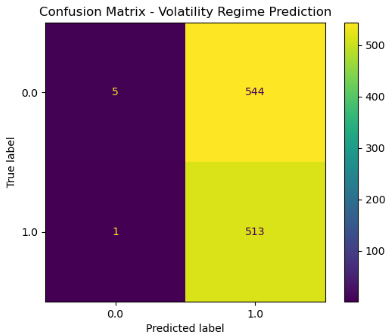


Figure 22: Confusion Matrix — Volatility Regime Prediction

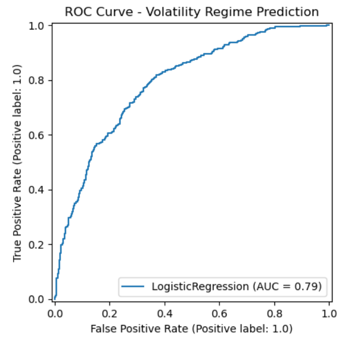


Figure 23: ROC Curve — Volatility Regime Prediction

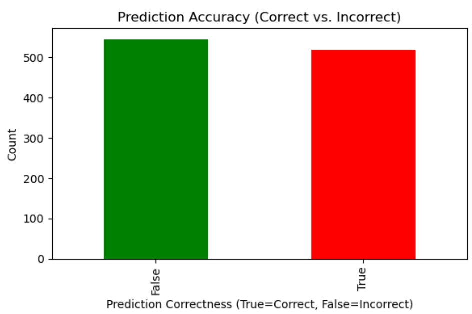


Figure 24: Prediction Accuracy (Correct vs. Incorrect)

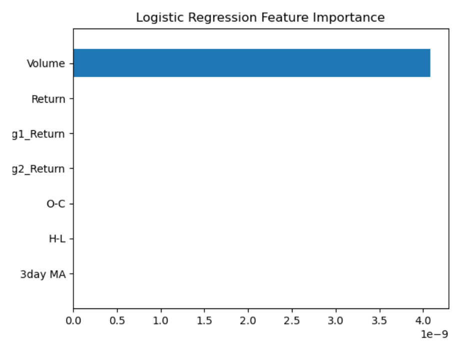


Figure 25: Logistic Regression Feature Importance

Appendix C

Logistic Regression

The logistic function (Cox, 1958) transforms a linear combination of input features into a probability between 0 and 1. Specifically, if x is a vector of predictors and β is the vector of coefficients, then the probability p of a positive class is given by

$$p(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)}},$$

where β_0 is the intercept and $\beta_1, \ldots, \beta_n$ are the model parameters. This formulation naturally handles binary outcomes by mapping any real-valued input to a value in $(0, 1)$. Training proceeds by maximizing the likelihood of the observed data under this model.

Extra Trees

Extra Trees method (Geurts et al., 2006) creates multiple decision trees to make predictions. Each tree is built using the original data, but instead of choosing the best split points, it picks them randomly. This randomness, along with selecting features randomly, helps prevent overfitting and makes the model more stable. In the end, all the trees work together by voting on the final result, either by averaging their predictions for numerical values (regression) or by choosing the most common answer (classification).

Cross Validation Analysis

Cross-validation is a technique used to test the performance of a model on new data. In k-fold cross-validation, the data is divided into k smaller sets. The model is trained on k-1 sets and tested on the remaining one. This is done k times, with a different test fold used every time. The performance is found by taking an average across all the folds, based on metrics like accuracy, precision, or recall. Cross-validation, by testing the model on multiple pieces of the data, avoids overfitting and gives a better indication of how the model will perform on new data.

Appendix D

The entire project is available at my GitHub repository

For completeness and clarity, I report the code in this document:

```
# Riccardo Negrisoli
# AI and ML



# To correctly run the code with no errors PLEASE
# use the command : pip install arch





from IPython import get_ipython
get_ipython().magic('reset -sf')
get_ipython().magic('clear')
#%%
#pip install yfinance pandas openpyxl
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import yfinance as yf
import pandas as pd
from arch import arch_model
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score, roc_curve, classification_report
from sklearn.metrics import confusion_matrix, make_scorer, roc_auc_score
from sklearn.metrics import RocCurveDisplay, ConfusionMatrixDisplay


from yahooquery import Ticker
import os

#%%  Section 0
## Relevant Information and preliminary analysis of Intesa
# FETCH FUNDAMENTAL DATA FOR INTESA SAN PAOLO

# Create a Ticker object for Intesa San Paolo (ticker "ISP.MI")
ISP = Ticker("ISP.MI")

# Retrieve and print the ESG scores
```

```python
print("ESG Scores:")
print(ISP.esg_scores)

# Retrieve and print key statistics
print("\nKey Statistics:")
print(ISP.key_stats)

# Retrieve and print the summary profile
print("\nSummary Profile:")
print(ISP.summary_profile)

# Retrieve and print institutional ownership details
print("\nInstitutional Ownership:")
print(ISP.institution_ownership)

# Retrieve and print fund ownership details
print("\nFund Ownership:")
print(ISP.fund_ownership)

# Retrieve and print the quarterly balance sheet by specifying frequency="q"
print("\nQuarterly Balance Sheet:")
print(ISP.balance_sheet(frequency="q"))

# Retrieve and print the cash flow statement
print("\nCash Flow:")
print(ISP.cash_flow())

# Retrieve and print the income statement
print("\nIncome Statement:")
print(ISP.income_statement())

#%% IMPORTING DATA

# Setting my working directory
os.chdir(r"C:\Users\Utente\Desktop\WESTMINSTER\AI and ML\Intesa_Project")
folder_path = r"C:\Users\Utente\Desktop\WESTMINSTER\AI and ML\Intesa_Project\
    Figures"

# Download Intesa Sanpaolo data
ticker = "ISP.MI" ### Intesa San Paolo S.p.A. is traded on the Milan Stock
    Exchange with the ticker "ISP.MI"
start_date = "2004-01-01"
end_date = "2024-12-31"

ISPdata = yf.download(ticker, start=start_date, end=end_date)
ISPdata.dropna(inplace=True)

# If columns is a MultiIndex (e.g., top level is "ISP.MI"), drop that level
if isinstance(ISPdata.columns, pd.MultiIndex):
    ISPdata.columns = ISPdata.columns.droplevel(1)

# Move the date index into a normal column
ISPdata.reset_index(inplace=True)

# Keep only the columns you want
ISPdata = ISPdata[['Date', 'Open', 'High', 'Low', 'Close', 'Volume']]

```

```python
# Export to Excel without the index
FileName = "Intesa_stock_data.xlsx"
ISPdata.to_excel(FileName, index=False)

print(f"Stock data saved to {FileName}")


#%% Section 2 : Feature Engeneering

# Create new features based on the stock prices
ISPdata['H-L'] = ISPdata['High'] - ISPdata['Low']
ISPdata['O-C'] = ISPdata['Close'] - ISPdata['Open']

# Calculate moving averages for the Close price using a 1-day lag to avoid
    lookahead bias
ISPdata['3day MA'] = ISPdata['Close'].shift(1).rolling(window=3).mean()
ISPdata['10day MA'] = ISPdata['Close'].shift(1).rolling(window=10).mean()
ISPdata['30day MA'] = ISPdata['Close'].shift(1).rolling(window=30).mean()

# Calculate the rolling standard deviation of the Close price over a 5-day
    window
ISPdata['Std_dev'] = ISPdata['Close'].rolling(window=5).std()

# Create a binary target variable: 1 if the next day's Close is higher than
    today's Close, else 0
ISPdata['Price_Rise'] = np.where(ISPdata['Close'].shift(-1) > ISPdata['Close'],
    1, 0)

# Drop any rows with missing values from shifting/rolling operations
ISPdata = ISPdata.dropna()

# Display the first few rows of the enhanced dataset
print(ISPdata.head())
ISPdata.index.name = "Date"
ISPdata.to_excel(FileName, index=False)
print(f"Stock data saved to {FileName}")


#%% Extra section 2.1: ANALYSIS OF THE VARIANCE

# Calculate returns (ensure the series is stationary)
returns = np.log(ISPdata['Close'] / ISPdata['Close'].shift(1)).dropna()
#Checking for stationarity
plt.figure(figsize=(10, 5))
plt.plot(returns.index, returns, label='Returns')
plt.axhline(0, color='red', linestyle='--')
plt.title('Daily Returns')
plt.xlabel('Date')
plt.ylabel('Returns')
plt.legend()
plt.savefig(os.path.join(folder_path, "fig0.ClosingVol.Relationship.png"))
plt.show()

ISPdata['Returns']= returns

# Fit a GARCH(1,1) model with t-distribution
model = arch_model(returns, vol='Garch', p=1, q=1, dist='t', rescale=True)
```

```
fit = model.fit(disp='off')

# Compute conditional variance (volatility squared)
ISPdata = ISPdata.iloc[1:]  # Align with returns dropna
ISPdata['GARCH_t_variance'] = fit.conditional_volatility ** 2

print(fit.summary())
ISPdata.to_excel(FileName, index=False)
print(f"Stock␣data␣saved␣to␣{FileName}")
fig, ax1 = plt.subplots(figsize=(10, 6))

# Create a figure and an axis (ax1) for the first plot
fig, ax1 = plt.subplots(figsize=(10, 6))

# Plot Daily Returns on the left y-axis
color1 = 'tab:blue'
ax1.set_xlabel('Date', color=color1)
ax1.set_ylabel('Daily␣Returns', color=color1)
ax1.plot(ISPdata['Date'], ISPdata['Returns'], color=color1, label='Daily␣Returns')
ax1.tick_params(axis='y', labelcolor=color1)

# Create a second axis (ax2) sharing the same x-axis
ax2 = ax1.twinx()

# Plot GARCH Variance on the right y-axis
color2 = 'tab:red'
ax2.set_ylabel('GARCH␣Variance', color=color2)
ax2.plot(ISPdata['Date'], ISPdata['GARCH_t_variance'], color=color2, label='GARCH␣Variance')
ax2.tick_params(axis='y', labelcolor=color2)

plt.title('Daily␣Log-Returns␣and␣GARCH␣Conditional␣Variance')
fig.tight_layout()
plt.legend()
plt.savefig(os.path.join(folder_path, "fig20.Garch&Returns.png"))
plt.show()




#%%  PLOTS

# Figure 1: Plot for Closing Price and Volume
fig, ax1 = plt.subplots(figsize=(12, 6))
color = 'tab:blue'
ax1.set_xlabel('Date', color=color)
ax1.set_ylabel('Closing␣Price', color=color)
ax1.plot(ISPdata['Date'], ISPdata['Close'], label='Closing␣Price', color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()
color = 'tab:orange'
ax2.set_ylabel('Volume', color=color)
ax2.plot(ISPdata['Date'], ISPdata['Volume'], label='Volume', color=color)
ax2.tick_params(axis='y', labelcolor=color)

```

```python
plt.title('Closing Price and Volume Relationship')
fig.tight_layout()
plt.savefig(os.path.join(folder_path, "fig1.ClosingVol.Relationship.png"))
plt.show()

# Figure 2: Plot for 3day, 10day, 30day MAs
plt.figure(figsize=(10, 8))
plt.plot(ISPdata['Date'], ISPdata['3day MA'], label='3day MA')
plt.plot(ISPdata['Date'], ISPdata['10day MA'], label='10day MA')
plt.plot(ISPdata['Date'], ISPdata['30day MA'], label='30day MA')
plt.title('Moving Averages Over Time')
plt.xlabel('Date')
plt.ylabel('Moving Averages')
plt.legend()
plt.savefig(os.path.join(folder_path, "fig2.MA.png"))
plt.show()

# Figure 3: Plots for O-C, H-L, and Std_dev
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].plot(ISPdata['Date'], ISPdata['O-C'])
axes[0].set_title('O-C')
axes[1].plot(ISPdata['Date'], ISPdata['H-L'])
axes[1].set_title('H-L')
axes[2].plot(ISPdata['Date'], ISPdata['Std_dev'])
axes[2].set_title('Std_dev')

plt.tight_layout()
plt.savefig(os.path.join(folder_path, "fig3.Comparisons.png"))
plt.show()

# Figure 4: 3day, 10day, 30day MA histogram & density
fig, axes = plt.subplots(1, 3, figsize=(15, 3))
sns.histplot(data=ISPdata, x='3day MA', kde=True, stat='density', ax=axes[0])
axes[0].set_title('3day MA')

sns.histplot(data=ISPdata, x='10day MA', kde=True, stat='density', ax=axes[1])
axes[1].set_title('10day MA')

sns.histplot(data=ISPdata, x='30day MA', kde=True, stat='density', ax=axes[2])
axes[2].set_title('30day MA')

plt.tight_layout()
plt.savefig(os.path.join(folder_path, "fig4.Hist&Density.png"))
plt.show()

# Figure 5: Price Rise & Fall Count Plot + Scatter
chart = sns.FacetGrid(ISPdata, col='Price_Rise')
chart.map(sns.histplot, 'Close')
plt.savefig(os.path.join(folder_path, "figure5_hist.png"))
plt.show()

chart = sns.FacetGrid(ISPdata, col='Price_Rise')
chart.map(plt.scatter, 'Close', 'Std_dev')
plt.savefig(os.path.join(folder_path, "figure5_scatter.png"))
plt.show()

#Fig 6: Corr Matrix
```

```
corr_matrix = ISPdata.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title("Correlation Matrix")
plt.savefig(os.path.join(folder_path, "fig6.CorrMatrix.png"))
plt.show()



#%%
ISPdata.info()
ISPdata.describe()
ISP.summary_profile

ISP.institution_ownership

ISP.fund_ownership
ISP.history()

#%%  MACHINE LEARNING

feature_cols = ['Volume', 'H-L', 'O-C','Std_dev', '3day MA']
target_col = 'Price_Rise'

X = ISPdata[feature_cols]

Y = ISPdata[target_col]

Y



X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15,
    random_state=240)
X_train

# Standardize features by removing the mean and scaling to unit variance.
# StandardScaler transfers data to standard normally distributed data: Gaussian
    with zero mean and unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

regr = LinearRegression()
regr.fit(X_train, Y_train)
Y_pred = regr.predict(X_test)
print ("MAE", mean_absolute_error(Y_test, Y_pred))

#%%
#  Logistic Regression

model_logistic = LogisticRegression(random_state=101)
model_logistic.fit(X_train, Y_train)

# Predict on the test set
Y_pred_logistic = model_logistic.predict(X_test)
print("Logistic Regression Classification Report:")
```

```
print(classification_report(Y_test, Y_pred_logistic))


# Extra Trees
model_extra_trees = ExtraTreesClassifier(random_state=101)
model_extra_trees.fit(X_train, Y_train)

# Predict on the test set
Y_pred_extra = model_extra_trees.predict(X_test)
print("Extra Trees Classification Report:")
print(classification_report(Y_test, Y_pred_extra))

 ## LOGISTIC REGRESSION PERFORMS BETTER!!!


#%%  #N-fold Cross Validation

#Cross Validation for Logistic Regression
accuracy_scores_log = cross_val_score(model_logistic, X, Y, cv=5, scoring='accuracy')
print(f"Logistic Regression - Mean Accuracy: {accuracy_scores_log.mean():.2f}")
print(f"Logistic Regression - Standard Deviation: {accuracy_scores_log.std():.2f}")

#Cross Validation for Extra Trees
accuracy_scores_et = cross_val_score(model_extra_trees, X, Y, cv=5, scoring='accuracy')
print(f"Extra Trees - Mean Accuracy: {accuracy_scores_et.mean():.2f}")
print(f"Extra Trees - Standard Deviation: {accuracy_scores_et.std():.2f}")

## WE ARE PERFORMING NO BETTER THAN RANDOM GUESSING



# ----------------------------------------
#%% Prediction of Price Rise Using Logistic Regression on X_test Data

# Re-train or re-use model_extra_trees
model_logistic = LogisticRegression(random_state=101)
model_logistic.fit(X_train, Y_train)

Y_pred_logistic_test = model_logistic.predict(X_test)
classification_rep = classification_report(Y_test, Y_pred_logistic_test)
print("Final Logistic Regression Classification Report on X_test:")
print(classification_rep)


#%% Confusion MAtrix for logistic
matrix = ConfusionMatrixDisplay.from_estimator(model_logistic, X_test, Y_test)
plt.title('Confusion Matrix')
plt.savefig(os.path.join(folder_path, "fig6.ConfMatrix.png"))
plt.show()


#%% Confusion MAtrix for Extra Trees
matrix = ConfusionMatrixDisplay.from_estimator(model_extra_trees, X_test, Y_test)
```

```
plt.title('Confusion Matrix')
plt.savefig(os.path.join(folder_path, "fig7.ConfMatrix2.png"))
plt.show()

#%%   ROC Curve for Logistic Regression
log_disp = RocCurveDisplay.from_estimator(model_logistic, X_test, Y_test)
plt.title("ROC Curve - Logistic")
plt.savefig(os.path.join(folder_path, "roc_curve.Logistic.png"))
plt.show()



#%%  BAYESIAN NAIVE


Gauss_Model = GaussianNB()
Gauss_Model.fit(X_train, Y_train)
Y_pred = Gauss_Model.predict(X_test)
target_names=["Down", "Up"]

print (classification_report(Y_test, Y_pred))

# Evaluate the model by means of a Confusion Matrix

matrix = ConfusionMatrixDisplay.from_estimator(Gauss_Model, X_test, Y_test,
    display_labels=target_names)
plt.title('Confusion Matrix')
plt.savefig(os.path.join(folder_path, "fig8.ConfMatrix3.png"))
plt.show()

#%%  FEATURE IMPORTANCE  LOGISTIC
classifier = LogisticRegression(random_state=101)
classifier.fit(X_train, Y_train)

# Now retrieve the coefficients from the fitted classifier
importance = classifier.coef_[0]



feature_names=X.columns


indices = np.argsort(importance)
range1 = range(len(importance[indices]))
plt.figure()
plt.title("Logistic Regression Feature Importance")
plt.barh(range1,importance[indices])
plt.yticks(range1, feature_names[indices])
plt.ylim([-1, len(range1)])
plt.savefig(os.path.join(folder_path,"fig9.Logistic.FeatureImportance.png"))
plt.show()

#%% MARKET AND RETURN STRATEGIES WITH LOGISTIC

#  Data Preprocessing
ISPdata['Y_pred_logistic_test'] = np.nan
ISPdata.iloc[len(ISPdata) - len(Y_pred_logistic_test):, -1] =
```

```
    Y_pred_logistic_test  # Fill the last rows with predictions
trade_ISPdata = ISPdata.dropna()  # Drop rows without a prediction

# Computation of Market Returns
trade_ISPdata['Tomorrows Returns'] = 0.
trade_ISPdata['Tomorrows Returns'] = np.log(trade_ISPdata['Close'] / trade_ISPdata['Close'].shift(1))
trade_ISPdata['Tomorrows Returns'] = trade_ISPdata['Tomorrows Returns'].shift(-1)

# Strategy Returns based on Y_pred
trade_ISPdata['Strategy Returns'] = 0.
trade_ISPdata['Strategy Returns'] = np.where(
     trade_ISPdata['Y_pred_logistic_test'] == True,
     trade_ISPdata['Tomorrows Returns'],
     -trade_ISPdata['Tomorrows Returns']
)


# ######Cumulative Market and Strategy Returns

#Computation of cumulative market and strategy returns
trade_ISPdata['Cumulative Market Returns'] = np.cumsum(trade_ISPdata['Tomorrows Returns'])
trade_ISPdata['Cumulative Strategy Returns'] = np.cumsum(trade_ISPdata['Strategy Returns'])

#Plot of cumulative market and strategy returns based on Y_pred
plt.figure(figsize=(10, 3))
plt.plot(trade_ISPdata['Cumulative Market Returns'], color='red', label='Market Returns')
plt.plot(trade_ISPdata['Cumulative Strategy Returns'], color='blue', label='Strategy Returns')
plt.title("Cumulative Market vs. Strategy Returns")
plt.legend()
plt.savefig(os.path.join(folder_path,"fig10.MktStrategy.png"))
plt.show()



#%% FEATURES IMPORTANCE AND MARKET STRATEGY RETURNS USING NAIVE BAYES

# FEATURE "IMPORTANCE" with GaussianNB
# ================================
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, Y_train)

# Naive Bayes does not provide coefficients; as a proxy we use the class conditional means.
# (This only makes sense if your task is binary. Here we assume class 1 is the "positive" class.)
importance = nb_classifier.theta_[1]
feature_names = X.columns

# Sort the "importance" values for plotting
indices = np.argsort(importance)
range1 = range(len(importance))
```

```

plt.figure(figsize=(10, 6))
plt.title("GaussianNB - Feature Means for Class 1")
plt.barh(range1, importance[indices])
plt.yticks(range1, feature_names[indices])
plt.xlabel("Mean Value")
plt.tight_layout()
plt.savefig(os.path.join(folder_path, "fig9.GaussianNB_FeatureMeans.png"))
plt.show()

# ================================
# MARKET AND RETURN STRATEGIES WITH GAUSSIAN NAIVE BAYES
# ================================

# Get predictions on the test set from the NB model
Y_pred_nb = nb_classifier.predict(X_test)
print("GaussianNB Classification Report on X_test:")
print(classification_report(Y_test, Y_pred_nb))

# Data Preprocessing: Add NB predictions to ISPdata.
# Here we assume that ISPdata originally has all the data in order and that the predictions
# correspond to the last len(Y_pred_nb) rows.
ISPdata['Y_pred_nb'] = np.nan
ISPdata.iloc[len(ISPdata) - len(Y_pred_nb):, -1] = Y_pred_nb  # Fill last rows with predictions
trade_ISPdata = ISPdata.dropna()  # Drop rows without a prediction

# Compute Market Returns as log returns
trade_ISPdata['Tomorrows Returns'] = np.log(trade_ISPdata['Close'] / trade_ISPdata['Close'].shift(1))
trade_ISPdata['Tomorrows Returns'] = trade_ISPdata['Tomorrows Returns'].shift(-1)

# Compute Strategy Returns based on the NB predictions:
# If Y_pred_nb is True (or 1), take the market return; else, take the negative.
trade_ISPdata['Strategy Returns'] = np.where(
    trade_ISPdata['Y_pred_nb'] == True,  # adjust if your positive class is represented differently
    trade_ISPdata['Tomorrows Returns'],
    -trade_ISPdata['Tomorrows Returns']
)

# Compute cumulative returns
trade_ISPdata['Cumulative Market Returns'] = np.cumsum(trade_ISPdata['Tomorrows Returns'])
trade_ISPdata['Cumulative Strategy Returns'] = np.cumsum(trade_ISPdata['Strategy Returns'])

# Plot the cumulative returns
plt.figure(figsize=(10, 3))
plt.plot(trade_ISPdata['Cumulative Market Returns'], color='red', label='Market Returns')
plt.plot(trade_ISPdata['Cumulative Strategy Returns'], color='blue', label='Strategy Returns')
plt.title("Cumulative Market vs. Strategy Returns (GaussianNB)")
plt.legend()
```

```python
plt.tight_layout()
plt.savefig(os.path.join(folder_path, "fig10.MktStrategy_GaussianNB.png"))
plt.show()

#%%   ANALYSIS OF VARIANCE WITH ML

# Use the median of GARCH_t_variance as the threshold
vol_threshold = ISPdata['GARCH_t_variance'].median()
ISPdata['HighVolatility'] = (ISPdata['GARCH_t_variance'] > vol_threshold).astype(int)

ISPdata['Lag1_Return'] = ISPdata['Return'].shift(1)
ISPdata['Lag2_Return'] = ISPdata['Return'].shift(2)
ISPdata['HighVolatility_Tomorrow'] = ISPdata['HighVolatility'].shift(-1)

# Drop rows with missing values caused by pct_change and shifting
data_ml = ISPdata.dropna(subset=['Return', 'Lag1_Return', 'Lag2_Return', 'Volume', 'HighVolatility_Tomorrow'])

# Define Feature Matrix X and Target Vector Y
featuresNew = ['Return', 'Lag1_Return', 'Lag2_Return', 'Volume','H-L','O-C', '3day MA']
targetNew = 'HighVolatility_Tomorrow'
X = data_ml[featuresNew]
Y = data_ml[targetNew]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=101)

# Train a Logistic Regression Model for Volatility Regime Prediction
modelGarch = LogisticRegression(random_state=101)
modelGarch.fit(X_train, Y_train)

# Predict on test set
Y_pred = modelGarch.predict(X_test)
correct_predictions = (Y_pred == Y_test)
print("Classification Report:")
print(classification_report(Y_test, Y_pred))
#  Plot and Save the Confusion Matrix
plt.figure(figsize=(6, 6))
cm_disp = ConfusionMatrixDisplay.from_estimator(modelGarch, X_test, Y_test)
plt.title("Confusion Matrix - Volatility Regime Prediction")
plt.savefig(os.path.join(folder_path, "Volatility_Regime_ConfusionMatrix.png"))
plt.show()

#  Plot the ROC Curve

plt.figure(figsize=(6, 6))
roc_disp = RocCurveDisplay.from_estimator(modelGarch, X_test, Y_test)
plt.title("ROC Curve - Volatility Regime Prediction")
plt.savefig(os.path.join(folder_path, "Volatility_Regime_ROC_Curve.png"))
plt.show()


modelGarch.fit(X_train, Y_train)
importance = modelGarch.coef_[0]
feature_names=X.columns
```

```python
indices = np.argsort(importance)
range1 = range(len(importance[indices]))
plt.figure()
plt.title("Logistic Regression Feature Importance")
plt.barh(range1,importance[indices])
plt.yticks(range1, feature_names[indices])
plt.ylim([-1, len(range1)])
plt.savefig(os.path.join(folder_path,"fig30.Variance.FeatureImportance.png"))
plt.show()

# Count the number of correct and incorrect predictions.
accuracy_counts = correct_predictions.value_counts()
print("Prediction Correctness Counts:\n", accuracy_counts)


plt.figure(figsize=(6,4))
accuracy_counts.plot(kind='bar', color=['green', 'red'])
plt.title("Prediction Accuracy (Correct vs. Incorrect)")
plt.xlabel("Prediction Correctness (True=Correct, False=Incorrect)")
plt.ylabel("Count")
plt.tight_layout()
plt.savefig(os.path.join(folder_path, "Volatility_Tomorrow_Prediction_Accuracy.png"))
plt.show()
```

Listing 1: Complete Python Code