

Title: Investigation of machine learning

RQ: How effective is logistic regression in classifying daily volatility based on historical price data?

Subject: Group 4—Computer science

Word count: 3657

Table of Contents

Introduction.....	2
Background Information.....	4
Volatility.....	4
The S&P 500.....	4
Beta coefficient.....	5
Logistic regression.....	5
Logistic function.....	5
Logistic model.....	7
Cost function.....	8
Gradient descent.....	11
Learning rate.....	12
Overfitting and underfitting.....	14
Experimental investigation.....	15
Experimental set-up.....	15
Data.....	15
Machine learning algorithm.....	17
Optimization of algorithm.....	18
Hyperparameter tuning.....	18
Regularization.....	18
Maximum number of iterations.....	19
Feature engineering.....	19
Logarithmic Returns.....	19
Historical Volatility.....	20
High Volatility Threshold.....	21
Z-score normalization.....	22
Dataset consideration.....	23
Running the model on different data sets.....	24
Confusion matrix.....	24
Accuracy.....	25
Volatility during crashes.....	26
High-volatility stock predictions.....	27
Low-volatility stock predictions.....	29
Conclusion.....	31
Limitations.....	32
Areas for further research.....	32
Bibliography.....	33
Appendix.....	36

Introduction

Over the past decades, there have been many more advancements within the field of artificial intelligence, which has sparked interest as these AI systems have become increasingly intelligent (Google Trends, n.d.). In 1977 Professor T. Mitchell stated that “a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

For the machine learning algorithm known as logistic regression, the task T is the ability to predict a binary outcome based on input values.

To measure performance P, a confusion matrix will be generated for each example and then the accuracy will be calculated.

The experience E is based on the dataset that the model runs on and can be categorized as supervised or unsupervised learning. In supervised learning labeled data is inputted with a corresponding output label, for the algorithm to map inputs to expected output values. While in unsupervised learning a machine learning algorithm is forced to find patterns among unlabeled data. For logistic regression, the task T requires the data to have an output label from which it is trained, therefore, making it a supervised learning algorithm (Ng, n.d.).

Through its simplicity logistic regression forms a baseline model for classification tasks and to this day is still one of the most widely used models. The reason for this is the model's dichotomous nature (Hosmer et al., 2013, p. 1). This feature makes the

algorithm useful in many different fields that deal with binary outcomes. Its applications can range from medicine, where a patient can be screened to have cancer or be cancer-free, or to finance where a transaction can be classified as fraudulent or not (Ng, n.d.).

This Extended Essay aims to answer the question “How effective is logistic regression in classifying daily volatility based on historical price data?” To find this answer, the algorithm has to be optimized and the effects discussed. Afterward, the model will be run on different datasets to see how effective this model is in separate scenarios. Finally, the results will be discussed along with the limitations of this research paper. Before that, the background information needs to be introduced.

Background Information

Volatility

Volatility represents how greatly an asset's prices swing around the mean price. The higher the volatility of a stock the more uncertain the price of it will be, that is why volatility is commonly used as a measure of risk (Hull, 2022, p. 342). The formula for volatility is given as:

$$\text{Volatility} = \sigma\sqrt{T}$$

Equation 1: Volatility formula

where:

- σ is the standard deviation of returns
- T is the specified period

For this EE I will be calculating historical volatility. "Historical volatility is based on historical prices and represents the degree of variability in the returns of an asset. This number is without a unit and is expressed as a percentage." (*Volatility*, n.d.)

The S&P 500

The S&P 500 (Standard & Poor's 500) is an index of stocks (a group of stocks), which are the 500 largest public companies in the USA by market capitalization (*S&P 500 Index*, n.d.). Since the S&P 500 comprises such a big chunk of the US market it serves as a benchmark for the overall US economy and thus its volatility will be used as a benchmark for all the stocks examined.

Beta coefficient

The beta coefficient is used in finance to denote the volatility compared to the overall market, usually to the S&P 500 which has a beta of 1.0. Stocks with betas higher than 1.0 are interpreted as more volatile, while stocks with betas lower than 1.0 are interpreted as less volatile (*What Beta Means for Investors*, n.d.).

Logistic regression

Logistic function

The logistic function is what the logistic regression is based on.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Equation 2: The logistic function

X is the input variable and e is Euler's number. The graph for this function is:

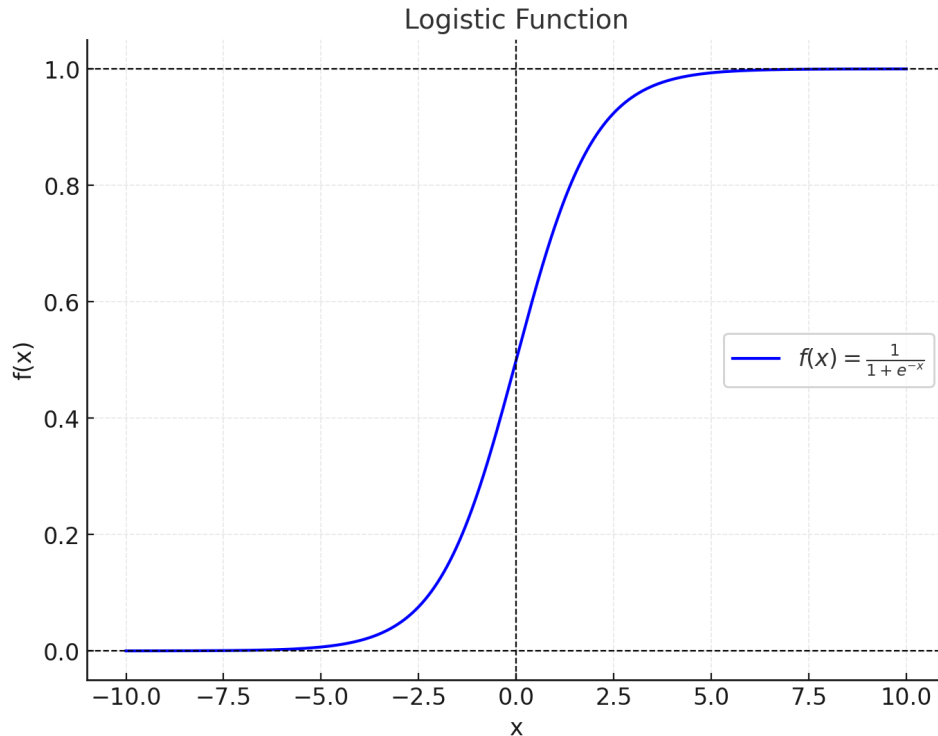


Figure 1: Graph of the logistic function (OpenAI, December 8, 2024)

In Figure 1 we can see that as x approaches positive infinity the function approaches the value of 1. When x approaches negative infinity then the function approaches the value of 0. This makes the range of the function between 0 and 1. This property is especially useful when discussing probabilities as its values also lie between 0 and 1.

Another feature that makes it very useful is its S-shape. The point at which the function reaches the value of 0.5 is where the function is at its steepest, as for the rest of the function it is smooth. This can be used for calculating a threshold, for which an observation can belong to a positive class - 1 or the negative class - 0 (Kleinbaum et al., 2010, pp. 5–7). In this EE, the threshold will represent the point at which the daily volatility will be considered high - 1 or low - 0.

Logistic model

To create the logistic model first we have to define a linear sum (Kleinbaum et al., 2010, p. 7), which will have the following parameters (Ng, n.d.):

- b - the bias term
- X - the input variable
- w - weight of the input variable X

Depending on the number of input variables k the weights will coincide with their amount, resulting in the formula:

$$x = b + w_1X_1 + w_2X_2 + \cdots + w_kX_k$$

Equation 3: Linear sum (Kleinbaum et al., 2010, p. 7)

Now we substitute the linear sum x into equation 2, to get:

$$f(x) = \frac{1}{1 + e^{-x}} = \frac{1}{1 + e^{-(b + \sum w_i X_i)}}$$

Equation 4: Logistic model equation (Kleinbaum et al., 2010, p. 7)

Cost function

The cost function can be presented as follows:

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(y'^{(i)}) + (1 - y^{(i)}) \log(1 - y'^{(i)})]$$

Equation 5: Logistic regression cost function

where:

- \vec{w} is a vector of the weights and b is the bias term
- m is the number of training examples
- y is the true label and represents the actual value of the binary outcome (either 0 or 1)
- y' is the predicted label whose value ranges from 0 to 1, given the set of features weight w and bias b (Ng, n.d.)

To aid in understanding the cost function, first, we will look at the logarithmic loss function (log loss function) which computes the prediction error for a single training example. Then we will break the function apart into 2 cases since the equation simplifies given the binary y value:

$$L(y', y) = \begin{cases} -\log(y') & \text{if } y = 1, \\ -\log(1 - y') & \text{if } y = 0. \end{cases}$$

Equation 6: Simplified loss function

From Equation 6 we can see that when the y value is equal to 1 the loss function simplifies to a single term. This can then be graphed and results in:

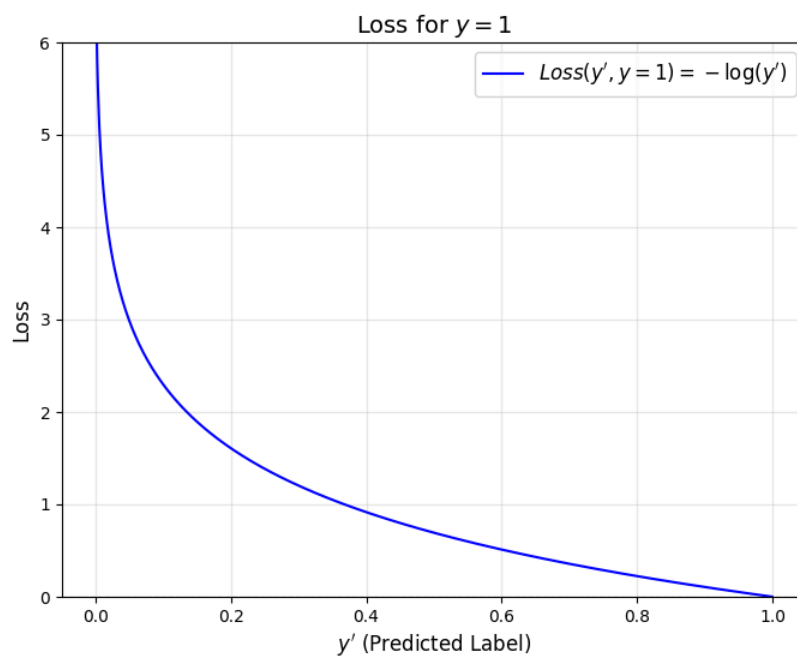


Figure 2: Graph of the loss function for $y = 1$ (OpenAI, December 8, 2024)

From Figure 2, it can be stated that:

- As y' approaches 1, the loss function approaches 0
- As y' approaches 0, the loss function approaches infinity

This process is analogical to when the true label is equal to 0 and thus the curves can be combined and create a binary cross-entropy loss graph:

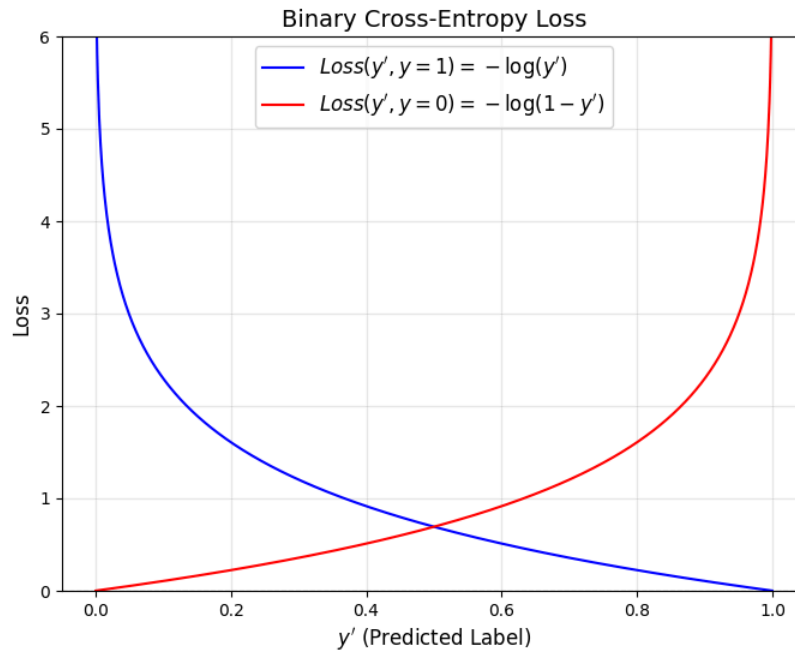


Figure 3: Binary cross-entropy graph (OpenAI, December 8, 2024)

From Figure 3 it can be seen that for each case of y , the model penalizes how far the prediction is from its true value. The use of the log function is what increases the penalty. As y' the prediction gets further away from y the true label the penalty gets higher until it reaches infinity.

Knowing this, the cost function from Equation 5 expands on this concept and applies it to all the values for which logistic regression is used, hence, the sigma operator.

The purpose of the cost function is to measure the error between the predictions and the actual labels. Therefore, if we try to minimize the results from the cost function, the y' predictions will become more and more accurate. This process of minimizing the cost is achieved with gradient descent.

Gradient descent

Gradient descent updates the weights and the bias of a model to improve its performance (Goodfellow et al., 2016). While the mathematics behind how gradient descent is implemented runs outside of the scope of this EE, it is important to have a conceptual understanding of how the algorithm works.

The algorithm for gradient descent works by repeating the given formula until convergence:

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$
$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$$

Equation 7: Formula for gradient descent

Firstly, the parameters, which are the vector of the weights w and the bias term b are set equal to random values, usually to zero, and the learning rate α is set to a certain value.

In this formula, the partial derivative a.k.a. the gradient is taken for each parameter w and b and multiplied by the learning rate α . The gradient tells us in which direction is the steepest increase in the cost function and since we want to minimize the cost we want to move the opposite way from the steepest increase, thus, we subtract the value of the gradient from the current parameters. This process is repeated and then the parameters are updated until the gradient reaches the value zero, and the parameters converge (Ng, n.d.).

Cost Function

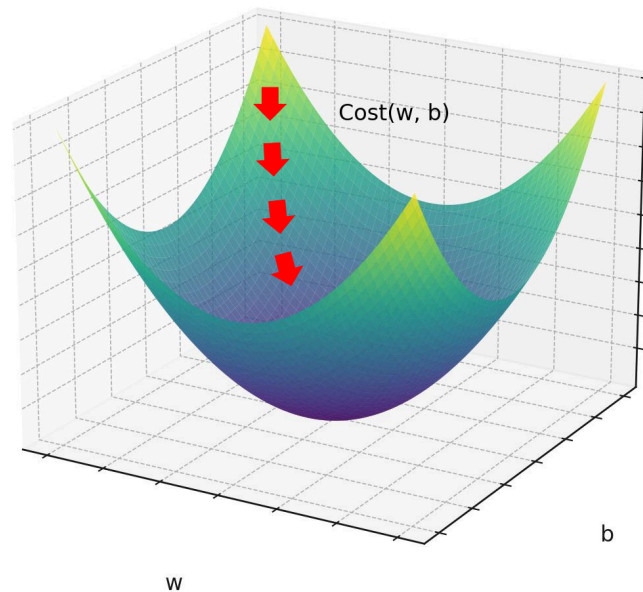


Figure 4: Graph of the cost function (OpenAI, December 8, 2024)

For logistic regression, the process of gradient descent can be seen on this graph where every step takes us closer to the global minimum since the graph is convex. When the global minimum is reached it means that the parameters have been optimized (Ng, n.d.).

Learning rate

The variable α also known as the learning rate indicates how big of a step the gradient descent takes to converge.

A small learning rate means that the steps toward convergence will be small and, therefore, it will take more steps to reach the minimum, because of extra computational cost, thus increasing the time it will take to reach it.

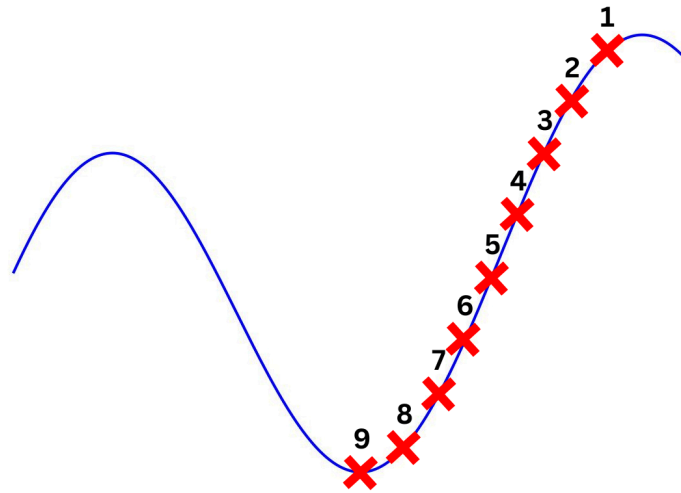


Figure 5: Graph showing the consequences of a small learning rate

A large learning rate will enable faster movement toward the minimum, but it also risks the possibility of overshooting and not reaching convergence as seen in Figure 5.

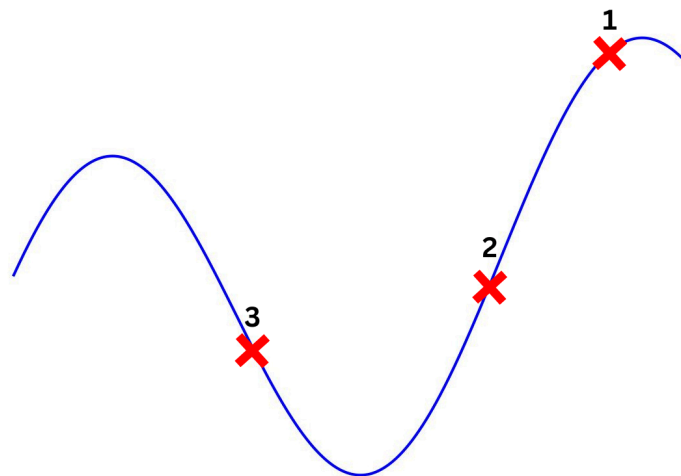


Figure 6: Graph showing the consequences of a high learning rate

An optimal learning rate is achieved when balancing the stability of the function with speed. Selecting the correct learning rate is, therefore, important to improve the performance of the machine learning algorithm (Ng, n.d.).

Overfitting and underfitting

Overfitting occurs when the model learns the data too well. It will cause the model to make inaccurate predictions. The model will perform well in training data, but when using test data it will perform poorly, meaning there will be high rates of error. The model will have high variance, meaning it focuses too much on specific data points making the model bad at generalization (Ng, n.d.).

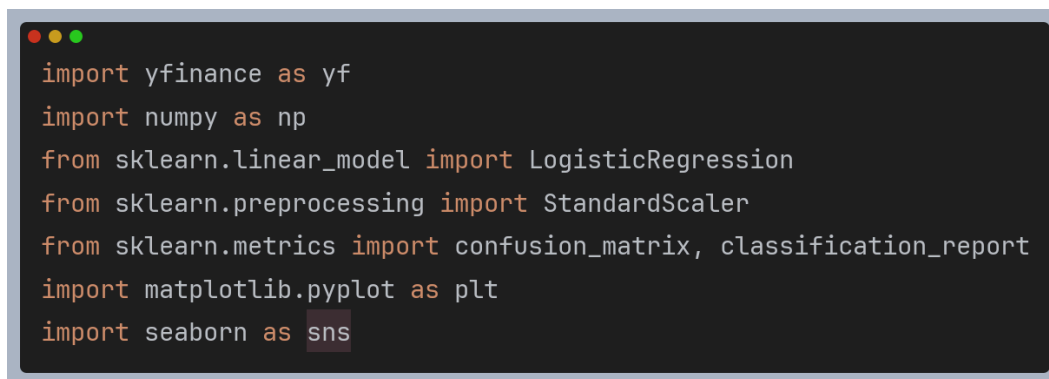
Underfitting is the opposite issue. It occurs when the model performs poorly both in training data and in test data. The model has a high bias, meaning it makes systematic errors in predictions. (Ng, n.d.). This problem usually occurs when there isn't enough training data, the data is bad quality or there are problems with the hyperparameters (Goodfellow et al., 2016).

Experimental investigation

After getting a good grasp on the topic of machine learning, we can set up the environment for the experiment and start optimizing the algorithm.

Experimental set-up

The logistics regression model will be implemented using PyCharm, an IDE (integrated development environment) for programming in Python which can be used for data science. Before coding it is necessary to import the necessary libraries, which will enable machine learning to occur.



```
import yfinance as yf
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

Figure 7: Code showing the import statements (see Appendix A)

Data

The financial data is loaded using yfinance. Yfinance provides access to Yahoo! Finance's API, which is used to retrieve historical market data. Here's an example of the yfinance library:


```

import yfinance as yf
ticker = 'INTC' # Example ticker
data = yf.download(ticker, start='2024-01-01', end='2024-01-10')
print(data)

```

Figure 8: Example of yfinance code (see Appendix D)

Date	Open	High	Low	Close	Adj Close	Volume
2024-01-02	49.200001	49.380001	47.450001	47.799999	47.168278	45905700
2024-01-03	47.099998	47.810001	46.799999	47.049999	46.428196	35858400
2024-01-04	45.720001	47.160000	45.240002	46.869999	46.250572	47797800
2024-01-05	47.029999	47.830002	46.639999	46.889999	46.270309	34332100
2024-01-08	47.070000	48.759998	46.970001	48.450001	47.809692	42135100
2024-01-09	48.009998	48.560001	47.799999	48.049999	47.414978	30097000

Figure 9: Shows data retrieved from Yfinance for the dates 2024-01-01 to 2024-01-10 (see Appendix D)

In Figure 9, it can be seen that the data includes 7 columns. Each column refers to a different feature:

- Date refers to the date that the data is from.
- Open is the price of the stock when the market opens
- High is the highest price that the stock achieved on a given day.
- Low is the lowest price that the stock achieved on that day.
- Close is the price of the stock at the time the market closes.
- Adj Close (adjusted close) refers to the closing price of the stock after accounting for any corporate actions

It is valuable to notice that the data is missing for the 6th and the 7th of January 2024, reflecting that the market is closed during the weekend. This will be important when deciding how much data will be needed to train the algorithm.

Machine learning algorithm

The logistic regression algorithm is obtained through Scikit Learn, a Python library used for machine learning and data analysis. Although the logistic regression algorithm obtained through this library doesn't use gradient descent per se, its implementation mirrors the underlying principles of it. For this reason, the learning rate also doesn't need to be set.

For the extent of this EE, pure implementation of gradient descent would not produce any significant insights as to how effective logistic regression is in classifying the daily volatility and would only add unnecessary complexity, therefore making its strict implementation redundant.

The rest of the imports Matplotlib and Seaborn are used for data visualization.

Optimization of algorithm

Hyperparameter tuning

Hyperparameters are features of the model that must be set externally. Finding their optimal values for these hyperparameters greatly improves performance (Ng, n.d.).

Regularization

In the logistic regression algorithm, regularization is implicitly added. Regularization is the process of adding a regularization term to the cost function to penalize large weights (Ng, 2004). In the case of the logistic regression that is being used in this EE the type of regularization that is used is L2. L2 regularization adds the sum of the squares of the coefficients to the cost function given by:

$$\sum_{i=1}^n w_i^2$$

Equation 8: Regularization term

By adding a square term to the cost function, the cost of having made an incorrect prediction becomes higher and, thus, discourages the use of large weights as the cost becomes exponentially higher. (Ng, 2004)

Through using regularization we can prevent overfitting as the algorithm won't overly rely on specific weights but rather it distributes the importance of weights more evenly making the algorithm more generalized.

Maximum number of iterations

The maximum number of iterations is the amount of iterations taken for the solver to converge (*Logisticregression*, n.d.). This value specifies the amount of times pseudo gradient descent runs before it reaches the global minimum. Making this number too low will make the algorithm never reach convergence, thus, choosing the right value is important. The size of the dataset is usually the predictor of what value to use.

The automatic value is 100, but since the algorithm will be using a moderately sized dataset the value of 200 will be used instead.

Feature engineering

Feature engineering is the action of preprocessing data that entails choosing features from raw data and then transforming them into formats that will be suitable for the machine learning algorithm. This step is very important as choosing the correct features will streamline the process of getting the correct prediction and will also enable the algorithm to produce outputs of higher quality (Zheng & Casari, 2018, p. 7).

Logarithmic Returns

Logarithmic returns measure the percentage change in the return of an asset over a specified period. The formula for log return is given as:

$$\text{Log Return} = \ln \left(\frac{P_t}{P_{t-1}} \right)$$

Equation 9: Equation for logarithmic return

Where:

- P_t is the price at time t
- P_{t-1} is the price at a previous time (Hudson & Gregoriou, 2015, p. 3)

The reason that log returns are especially useful for this EE is because:

1. “Logarithmic returns can be interpreted as continuously compounded returns,” (Hudson & Gregoriou, 2015, p. 5) which makes them more stable over a longer period which aligns with the long-term dataset that will be used for this analysis
2. “The logarithmic returns of the security are normally distributed” (Hudson & Gregoriou, 2015, p. 5), which will help since there won’t be any data points that are outliers
3. From the formula, it can be seen that log returns measure percentage change which is more useful for logistic regression than absolute price change since it is a more general and standardized measure
4. From the log returns, it is very easy to derive and calculate historical volatility

Historical Volatility

As a reminder, from Equation 1 the formula for volatility is:

$$\text{Volatility} = \sigma\sqrt{T}$$

Historical volatility is calculated from the aforementioned log returns by taking a rolling standard deviation, a standard deviation of a certain number of previous periods in a given column, over 30 days.

```
data['Historical Volatility'] = data['Log Return'].rolling(window=30).std() * np.sqrt(252)
data.dropna(inplace=True)
```

Figure 10: Code showing the calculation of volatility (see Appendix A)

This result is then annualized with the assumption that there are 252 trading days in a year (discounting the weekends in which markets are closed), which makes it more comparable across different and longer periods.

Dropping NaN values here means removing the first 29 rows since the historical volatility can't be calculated for the first 29 days.

High Volatility Threshold

For the threshold, which is a percentage, at which we start classifying whether the data should be classified as high volatility (positive class) or low volatility (negative class) we will use the following:

```
fixed_threshold = 0.18*1.22*normalization_factor
data['High Volatility'] = (data['Historical Volatility'] > fixed_threshold).astype(int)
```

Figure 11: Code showing the calculation of the threshold (see Appendix A)

The 18% was chosen based on the average volatility recorded by the index S&P 500 (V-Lab, n.d.). Then this value will be timed by the beta coefficient retrieved for the specific stock, in this case Intel, and then timed by the normalization factor, which was experimentally determined.

Z-score normalization

Z-score is a statistical measure that quantifies the number of standard deviations a value is from the mean. Z-score normalization is derived from this formula:

$$x' = \frac{x - \mu}{\sigma}$$

Equation 10: Z-score formula

Where:

- x is a value in the dataset,
- μ is the mean of the feature in the training data,
- σ is the standard deviation of the feature in the training data. (*Numerical Data*, n.d.)

The effects of the z-score scaling can be seen in the following graphs

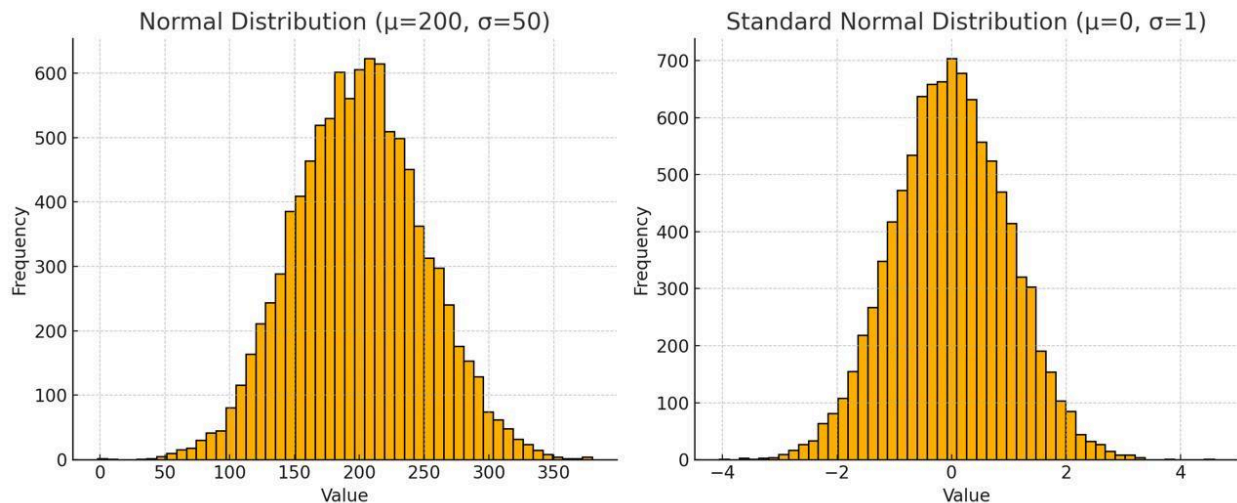


Figure 12: Graphs showing the effects of z-score scaling (OpenAI, December 8, 2024)

Normalization is one of the ways to address overfitting and also prevents larger features, for example, volume whose range is in the millions, from disproportionately affecting the model compared to smaller features like log returns, whose values stand in percentages (Ng, n.d.).

Dataset consideration

The logistic regression model doesn't need huge datasets, since it isn't an overly complex model compared to others like neural networks, etc.

For the machine learning algorithm to capture diverse conditions in the stock market like bear markets and bull markets which on average take 11 months and 4.2 years each (*Bull vs. Bear Markets*, n.d.), a dataset spanning approximately 10 years is needed.


This time frame will result in roughly 2,500 data features.

The resulting data will be then split into training and testing data with a ratio of 7 to 3 years. The split gives enough data points for sufficient training but also the 3 years will be sufficient to represent diverse market conditions.

A screenshot of a code editor window with a dark background and light-colored text. The code is: `data = yf.download(ticker, start='2014-01-01', end='2024-01-01')`. The code is underlined with a yellow wavy line. The editor has three colored window control buttons (red, yellow, green) in the top left corner.

Figure 13: Code showing the data retrieval period (see Appendix A)

The 10 years will span from the 1st of January 2014 to the 1st of January 2024.



```
# Split based on a date range
train_data = data[data.index < '2021-01-01']
test_data = data[data.index >= '2021-01-01']
```

Figure 14: Code showing train-test split date (see Appendix A)

The split will occur on January 1st, 2021 making the data before this point training data and the rest test data.

Running the model on different data sets

For interpreting the results we're going to use something called a confusion matrix and then the accuracy that is derived from it.

Confusion matrix

“A confusion matrix is a table that is used to define the performance of a classification algorithm” (*Machine Learning and the Internet of Medical Things in Healthcare*, 2021, Chapter 5.5).

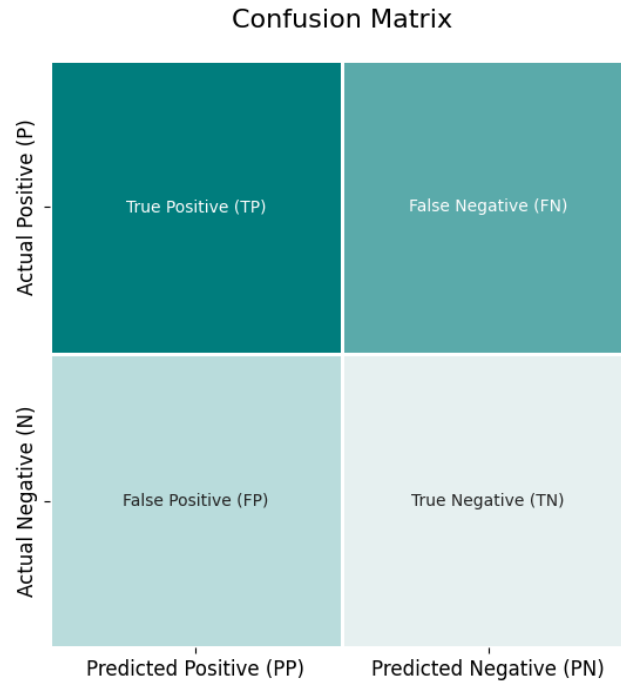


Figure 15: Example confusion matrix (OpenAI, December 9, 2024)

From Figure 15, we can see that the confusion matrix can have 4 values. These values are useful in assessing performance and calculating other measures of performance for a classification algorithm.

Accuracy

Accuracy is a measure of how well a classification algorithm performs as it calculates the ratio between the correctly classified values and the total number of values (*Machine Learning and the Internet of Medical Things in Healthcare*, 2021, Chapter 5.5).

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Equation 11: Accuracy formula

Volatility during crashes

Meta, otherwise known as Facebook has experienced headwinds in the preceding years. Its beta coefficient is 1.22 which makes it a moderately volatile stock (Yahoo Finance - Stock Market Live, Quotes, Business & Finance News, n.d.).



Figure 16: Graph of stock price for Meta (*Live Stock, Index, Futures, Forex and Bitcoin Charts on TradingView, n.d.*)

The red line denotes where the data is split from training to test data. From figure 16 it can be seen that in 2022 it experienced a very big crash. This makes it the perfect candidate to see if the logistic regression will accurately predict volatility during a crash.

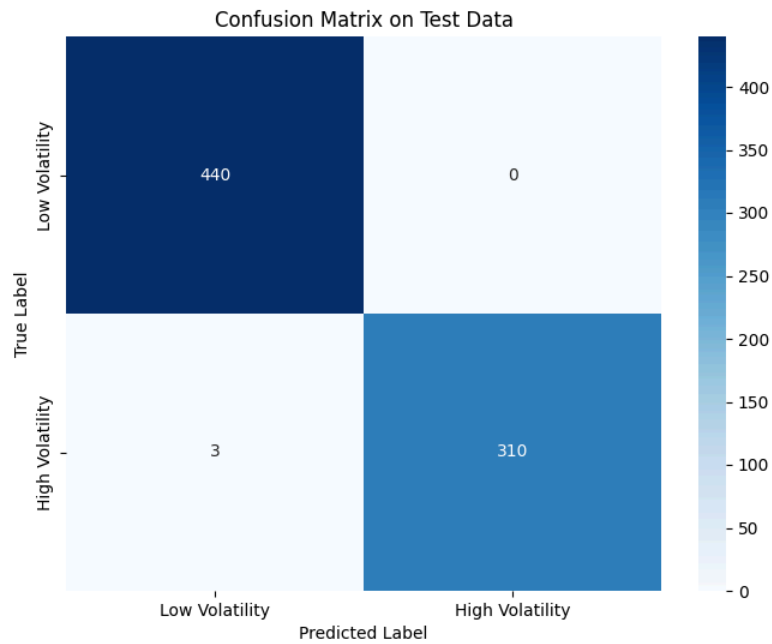


Figure 17: Confusion matrix for Meta (see Appendix A)

As seen on this confusion matrix the logistic regression was able to accurately classify the volatility during the crash, with an accuracy score of 99.6%.

High-volatility stock predictions

Next, we have Nvidia whose beta coefficient is 1.66 (Yahoo Finance - Stock Market Live, Quotes, Business & Finance News, n.d.) making it a very volatile stock.



Figure 18: Graph of stock price for Nvidia (*Live Stock, Index, Futures, Forex and Bitcoin Charts on TradingView, n.d.*)

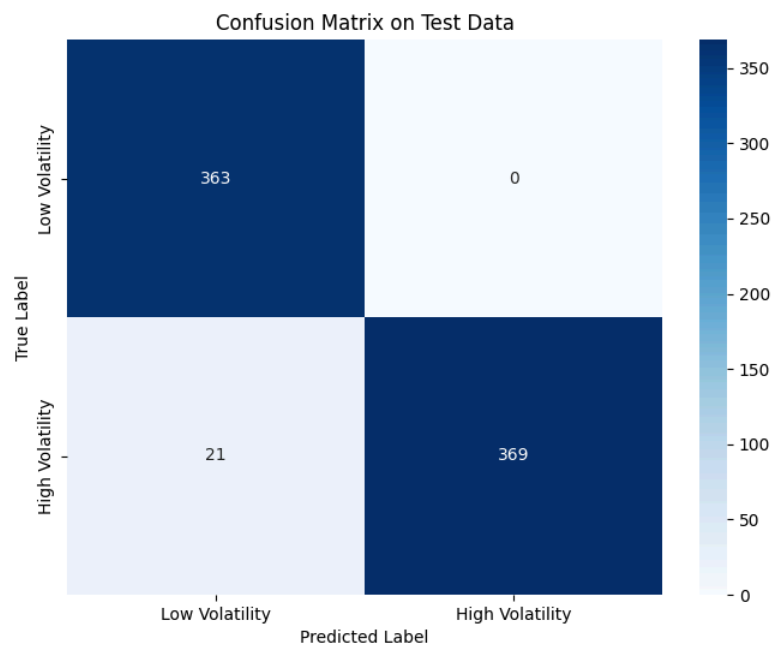


Figure 19: Confusion matrix for Nvidia (see Appendix B)

From the beta coefficient, it was already known that this stock was going to be a high volatility stock so the result of it having more positive classes than negative makes it a reasonable result. The accuracy for this prediction is 97.2%.

Low-volatility stock predictions

The last company is the Coca-Cola Company, whose beta coefficient is relatively low standing at 0.62 (Yahoo Finance - Stock Market Live, Quotes, Business & Finance News, n.d.).

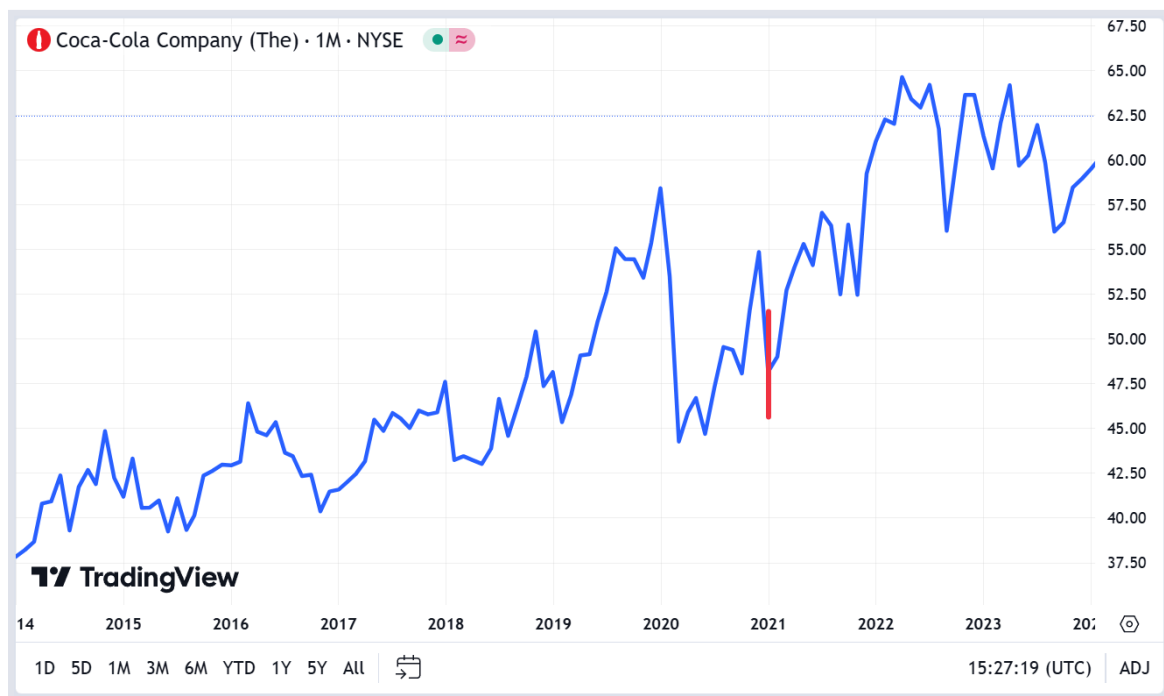


Figure 20: Graph of stock price for Coca-Cola (*Live Stock, Index, Futures, Forex and Bitcoin Charts on TradingView, n.d.*)

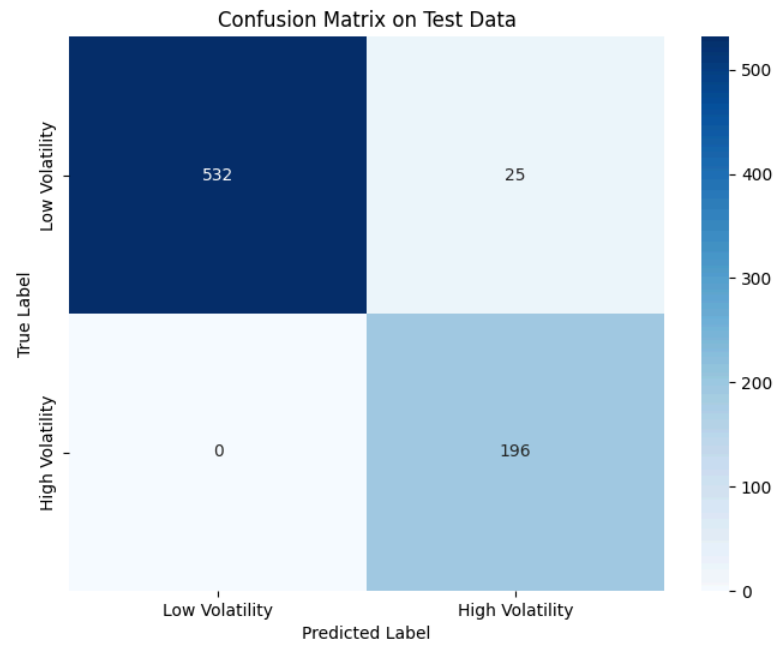


Figure 21: Confusion matrix for Coca-Cola (see Appendix C)

As expected from the low beta value, the results are negative class heavy as the stock is not volatile. The accuracy for this result is 95.4%, the lowest of all previous values.

Conclusion

For the scope of the research question the logistic regression algorithm has been proven to be successful at classifying the daily volatility based on historical price data even to a high degree with accuracy scores being above 95% in each case.

What's interesting about the findings is that for results in which classes were skewed slightly toward the negative class as seen in Figure 17, the number of inaccurate classifications declined, while when the class distribution was balanced it slightly increased as seen in Figure 19.

Generally, these findings underscore the importance of optimizing your algorithm through processes such as hyperparameter tuning, feature engineering, normalization, and choosing the correct dataset size, as without these steps taking place the outputs are bound to have a lower accuracy rate.

Hopefully, this paper provides somewhat of an easy introduction to the daunting field of machine learning and a tutorial on how logistic regression can be used to tackle classification tasks, showing in what way each step in the optimization process contributes to the overall improvement of the outputs of logistic regression. Although this paper only describes the optimization of one model, this overall process of tuning hyperparameters, etc. can be applied to many other algorithms within the machine learning sphere.

Limitations

While the results of running the algorithm are positive, this high level of accuracy is pretty unexpected still. First of all, this algorithm needs to be run on more datasets to see if this result is replicable. Secondly, the nature of volatility, which is that it reverts to its mean, may provide an answer as to why this problem can be classified with such a high degree of accuracy. Thirdly, the input variables for this logistic regression are highly correlated, with values like open and close price being part of a continuous daily trading cycle. The fact that other values like logarithmic returns are derived features further adds to the correlation, thus, reducing the reliability of the algorithm. Lastly, the volatility threshold, while calculated through the use of the average volatility, the beta coefficient, and a normalization factor, is still a rather arbitrary boundary.

Areas for further research

Further research might include running this logistic regression algorithm on more datasets and exploring extreme cases to see if the algorithm still works. Tackling the limitations that were mentioned, is a worthwhile pursuit, for example, examining the high volatility threshold and how it impacts the class distribution can provide interesting research. Additionally, changing the features that will be put into the algorithm, possibly removing and adding some other features outside of market data might result in even more accurate results, alongside changing the hyperparameters.

Bibliography

Bull vs. Bear markets: What's the difference? (n.d.). Investopedia. Retrieved

December 9, 2024, from

<https://www.investopedia.com/insights/digging-deeper-bull-and-bear-markets/>

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Google Trends. (n.d.). Google Trends: AI. Retrieved December 4, 2024, from

<https://trends.google.com/trends/explore?date=all&geo=IL&q=AI&hl=en>

Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied logistic regression* (Third edition). Wiley.

Hudson, R. S., & Gregoriou, A. (2015). Calculating and comparing security returns is harder than you think: A comparison between logarithmic and simple returns.

International Review of Financial Analysis, 38, 151–162.

<https://doi.org/10.1016/j.irfa.2014.10.008>

Hull, J. (2022). *Options, futures, and other derivatives* (Eleventh edition, global edition). Pearson4060 1 Online-Ressource (880 Seiten).

Kleinbaum, D. G., Klein, M., & Pryor, E. R. (2010). *Logistic regression: A self-learning text* (3rd ed). Springer.

Lightning. Classification. Sagaclassifier—Lightning 0. 6. 3. Dev0 documentation.

(n.d.). Retrieved December 9, 2024, from

<https://contrib.scikit-learn.org/lightning/generated/lightning.classification.SAGAClassifier.html>

Live stock, index, futures, Forex and Bitcoin charts on TradingView. (n.d.).

TradingView. Retrieved December 9, 2024, from

<https://www.tradingview.com/chart/>

Logisticregression. (n.d.). Scikit-Learn. Retrieved December 10, 2024, from

https://scikit-learn/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Machine learning and the internet of medical things in healthcare. (2021). Academic press.

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

Ng, A. (n.d.). Supervised Machine Learning: regression and classification.

Coursera. <https://www.coursera.org/learn/machine-learning/>

Ng, A. Y. (2004). Feature selection, L_1 vs. L_2 regularization, and rotational invariance. *Twenty-First International Conference on Machine Learning - ICML '04*, 78. <https://doi.org/10.1145/1015330.1015435>

Numerical data: Normalization | machine learning. (n.d.). Google for Developers.

Retrieved October 26, 2024, from

<https://developers.google.com/machine-learning/crash-course/numerical-data/normalization>

OpenAI. (2024). ChatGPT (GPT-4-turbo) [Large language model].

<https://chat.openai.com/chat>

S&p 500 index: What it's for and why it's important in investing. (n.d.). Investopedia.

Retrieved December 8, 2024, from

<https://www.investopedia.com/terms/s/sp500.asp>

V-lab: S&p 500 index garch volatility analysis. (n.d.). V-Lab. Retrieved December 7, 2024, from <https://vlab.stern.nyu.edu/volatility/VOL.SPX%3AIND-R.GARCH>

Volatility: Meaning in finance and how it works with stocks. (n.d.). Investopedia.

Retrieved December 7, 2024, from

<https://www.investopedia.com/terms/v/volatility.asp>

What beta means for investors. (n.d.). Investopedia. Retrieved December 8, 2024,

from <https://www.investopedia.com/terms/b/beta.asp>

Yahoo finance—Stock market live, quotes, business & finance news. (n.d.). Yahoo

Finance. Retrieved December 8, 2024, from <https://finance.yahoo.com/>

Zheng, A., & Casari, A. (2018). *Feature engineering for machine learning: Principles and techniques for data scientists.* O'Reilly.

Appendix

Appendix A

```
import yfinance as yf
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the dataset and perform feature engineering
ticker = 'META' # Example ticker
data = yf.download(ticker, start='2014-01-01', end='2024-01-01')

# Feature Engineering
data['Log Return'] = np.log(data['Close'] / data['Close'].shift(1))
data['Historical Volatility'] = data['Log Return'].rolling(window=30).std() * np.sqrt(252)
print(data['Historical Volatility'])
data.dropna(inplace=True)

# Experimentally determined factor
normalization_factor = 1.7

# Fixed Volatility Threshold
fixed_threshold = 0.18 * 1.22 * normalization_factor
data['High Volatility'] = (data['Historical Volatility'] >
fixed_threshold).astype(int)

# Prepare features and target
features = data[['Open', 'High', 'Low', 'Close', 'Volume', 'Log Return', 'Historical Volatility']]
target = data['High Volatility']

# Step 2: Train-test split and normalization

# Split based on a date range
train_data = data[data.index < '2021-01-01']
test_data = data[data.index >= '2021-01-01']

# Separate features and target
```

```

X_train = train_data[['Open', 'High', 'Low', 'Close', 'Volume', 'Log
Return', 'Historical Volatility']]
y_train = train_data['High Volatility']
X_test = test_data[['Open', 'High', 'Low', 'Close', 'Volume', 'Log
Return', 'Historical Volatility']]
y_test = test_data['High Volatility']

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train Logistic Regression Model
logistic_model = LogisticRegression(solver='saga', max_iter=200)
logistic_model.fit(X_train, y_train)

# Step 3: Evaluate the model using the test data
test_predictions = logistic_model.predict(X_test)

# Confusion Matrix
test_cm = confusion_matrix(y_test, test_predictions)
print("Confusion Matrix on Test Data:\n", test_cm)

# Step 4: Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(test_cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Low Volatility', 'High Volatility'],
            yticklabels=['Low Volatility', 'High Volatility'])
plt.title('Confusion Matrix on Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
(OpenAI, December 6, 2024)

```

Appendix B

```

import yfinance as yf
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the dataset and perform feature engineering

```

```

ticker = 'NVDA' # Example ticker
data = yf.download(ticker, start='2014-01-01', end='2024-01-01')

# Feature Engineering
data['Log Return'] = np.log(data['Close'] / data['Close'].shift(1))
data['Historical Volatility'] = data['Log
Return'].rolling(window=30).std() * np.sqrt(252)
print(data['Historical Volatility'])
data.dropna(inplace=True)

# Experimentally determined factor
normalization_factor = 1.7

# Fixed Volatility Threshold
fixed_threshold = 0.18 * 1.66 * normalization_factor
data['High Volatility'] = (data['Historical Volatility'] >
fixed_threshold).astype(int)

# Prepare features and target
features = data[['Open', 'High', 'Low', 'Close', 'Volume', 'Log
Return', 'Historical Volatility']]
target = data['High Volatility']

# Step 2: Train-test split and normalization

# Split based on a date range
train_data = data[data.index < '2021-01-01']
test_data = data[data.index >= '2021-01-01']

# Separate features and target
X_train = train_data[['Open', 'High', 'Low', 'Close', 'Volume', 'Log
Return', 'Historical Volatility']]
y_train = train_data['High Volatility']
X_test = test_data[['Open', 'High', 'Low', 'Close', 'Volume', 'Log
Return', 'Historical Volatility']]
y_test = test_data['High Volatility']

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train Logistic Regression Model
logistic_model = LogisticRegression(solver='saga', max_iter=200)
logistic_model.fit(X_train, y_train)

```

```

# Step 3: Evaluate the model using the test data
test_predictions = logistic_model.predict(X_test)

# Confusion Matrix
test_cm = confusion_matrix(y_test, test_predictions)
print("Confusion Matrix on Test Data:\n", test_cm)

# Step 4: Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(test_cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Low Volatility', 'High Volatility'],
            yticklabels=['Low Volatility', 'High Volatility'])
plt.title('Confusion Matrix on Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
(OpenAI, December 6, 2024)

```

Appendix C

```

import yfinance as yf
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the dataset and perform feature engineering
ticker = 'KO' # Example ticker
data = yf.download(ticker, start='2014-01-01', end='2024-01-01')

# Feature Engineering
data['Log Return'] = np.log(data['Close'] / data['Close'].shift(1))
data['Historical Volatility'] = data['Log Return'].rolling(window=30).std() * np.sqrt(252)
print(data['Historical Volatility'])
data.dropna(inplace=True)

# Experimentally determined factor
normalization_factor = 1.7

# Fixed Volatility Threshold
fixed_threshold = 0.18 * 0.62 * normalization_factor

```



```

data['High Volatility'] = (data['Historical Volatility'] >
fixed_threshold).astype(int)

# Prepare features and target
features = data[['Open', 'High', 'Low', 'Close', 'Volume', 'Log
Return', 'Historical Volatility']]
target = data['High Volatility']

# Step 2: Train-test split and normalization

# Split based on a date range
train_data = data[data.index < '2021-01-01']
test_data = data[data.index >= '2021-01-01']

# Separate features and target
X_train = train_data[['Open', 'High', 'Low', 'Close', 'Volume', 'Log
Return', 'Historical Volatility']]
y_train = train_data['High Volatility']
X_test = test_data[['Open', 'High', 'Low', 'Close', 'Volume', 'Log
Return', 'Historical Volatility']]
y_test = test_data['High Volatility']

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train Logistic Regression Model
logistic_model = LogisticRegression(solver='saga', max_iter=200)
logistic_model.fit(X_train, y_train)

# Step 3: Evaluate the model using the test data
test_predictions = logistic_model.predict(X_test)

# Confusion Matrix
test_cm = confusion_matrix(y_test, test_predictions)
print("Confusion Matrix on Test Data:\n", test_cm)

# Step 4: Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(test_cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Low Volatility', 'High Volatility'],
            yticklabels=['Low Volatility', 'High Volatility'])
plt.title('Confusion Matrix on Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

```

```
plt.show()
```

(OpenAI, December 6, 2024)

Appendix D

```
import yfinance as yf
```

```
ticker = 'INTC' # Example ticker
```

```
data = yf.download(ticker, start='2024-01-01', end='2024-01-10')
```

```
print(data)
```

(OpenAI, December 6, 2024)