Predicting Short-Term Price Movements of Top 10 Cryptocurrencies Using Historical OHLCV Data

1.0 Introduction

Cryptocurrencies are digital assets that operate on decentralized blockchain networks. Their highly volatile nature makes predicting future price movement a challenging but valuable task for traders, analysts, and investors.

This project focuses on using historical daily price and volume data of the top 10 cryptocurrencies to predict whether the next-day price will go up or down. By building a classification model, we aim to identify patterns in historical data that can inform short-term trading or market insights.

1.1 Project Aim

To analyze historical price and volume data of the top 10 cryptocurrencies and build a classification model that predicts whether the price will go up or down the next day, providing insights into short-term market movement patterns.

1.2 Project Objectives

- Data Collection: Gather historical OHLCV data for 100 cryptocurrencies from Kaggle and select the top 10 coins manually based on popularity, trading volume, and data availability.
- Data Cleaning & Preprocessing: Handle missing values, convert data types, remove duplicates, and normalize features where necessary.
- Feature Engineering: Create derived metrics such as daily returns, moving averages, volatility, and volume changes to serve as model features.
- Target Label Creation: Generate a binary target variable representing next-day price movement (1 = up, 0 = down).
- Model Development: Train and evaluate classification models (e.g., Logistic Regression, Random Forest) to predict next-day price movement.
- Model Evaluation & Validation: Assess model performance using accuracy, precision, recall, F1-score, and confusion matrices.
- Documentation & Reporting: Document the methodology, analysis, and findings in a structured report for reproducibility and clarity.

2.0 Data Exploration

2.1 Prediction Type: Classification

- Target label: 1 = price goes up next day, 0 = price goes down next day
- Rationale: Predicting up/down movement is simpler and more reliable than predicting exact prices, especially given the high volatility of cryptocurrencies.

2.2 Dataset Description / Data Scope

- Source: 100 cryptocurrency datasets were downloaded from Kaggle.
- Reduction to Top 10 Coins:
  - Challenge: Analyzing all 100 coins would be computationally heavy and may dilute meaningful insights. Additionally, market capitalization data was unavailable, so automatic ranking by size or popularity wasn't possible.
  - Solution: 10 cryptocurrencies was manually selected based on:

  Historical popularity and adoption

  Trading volume and liquidity

  Early trading date

  - Top 10 Coins Selected Manually:

    1st batch- BTC, ETH, BNB, XRP, ADA, SOL, USDT, DOGE, DOT, LTC

    2nd batch- BTC, BNB, DASH, DOGE, ETH, XMR, LTC, XLM, XRP, ZCASH
  - Justification: This approach ensures that the analysis remains manageable, focused on the most relevant cryptocurrencies, and still provides meaningful patterns and insights.
- Columns: Date, Open, High, Low, Close, Volume, Currency
- Date Range: From coin's first trading day ("genesis") to August 2022 (varies per coin)

2.3 Tools and Technologies

- Python Libraries: pandas, numpy, matplotlib, seaborn, scikit-learn
- Preprocessing Tools: MinMaxScaler and One Hot Encoder
- Modeling: Logistic Regression, Random Forest, Pipeline Model, Gradient Boosting Classifier
- Environment: Visual Studio Code for workflow documentation

2.4 Data Preprocessing

- Load Data: Read all CSV files for the top 10 coins.
- Inspect Data: Check for missing values, duplicates, and date ranges.

- Convert Data Types: Ensure numeric types for price/volume; convert Date to datetime.

We did not explicitly remove outliers because large swings are a natural part of cryptocurrency markets. Instead, we cleaned obvious errors (zeros, duplicates) and used features like rolling statistics to account for volatility.

3.0 Feature Engineering & Target Creation

Feature engineering plays a crucial role in enhancing model performance, particularly when working with financial time-series data, such as cryptocurrency prices. In this project, raw market data was transformed into meaningful features that capture price behavior, market trends, and coin-specific characteristics while avoiding data leakage.

The following feature engineering variables were created:

3.1 Price-based features

- Daily Return: Daily returns were used instead of raw prices to normalize price movements across cryptocurrencies with different price scales. Returns capture relative changes in price, making them more suitable for machine learning models. Additionally, predicting price direction naturally aligns with the sign of the next-day return, allowing the classification task to be defined consistently.
- High_Low range: The High–Low range was computed as (High − Low) / Open to capture daily price volatility relative to the opening price. This feature measures the intensity of intraday price movement and provides the model with information about market uncertainty and trading activity, which are important drivers of future price direction in cryptocurrency markets.
- Open_Close range: The Open–Close range was created to measure intraday price movement by capturing the difference between opening and closing prices. It reflects market volatility and trading intensity, helping identify periods of strong buying or selling pressure that closing prices alone cannot show.
- Volatility_7: Rolling volatility windows (rolling std of returns over 7 days were created to capture short-term market risk and uncertainty, allowing the model to adapt its predictions based on recent price instability while preserving time-series integrity.

3.2 Moving averages & momentum

- Moving averages of close price (MA_7 & MA_14): A 7-day and 14-day moving average of the closing price was created to smooth short-term price fluctuations and capture short-term trends that help improve directional price prediction in volatile cryptocurrency markets.
- Moving average of volume (Vol_MA_7 & Vol_MA_14): The 7-period and 14-period moving averages of trading volume, used to smooth short-term fluctuations and highlight the underlying trend in volume.
- Price vs moving average (Close / MA_7 & Close / MA_14): The ratio of the closing price to its 7-period and 14-period moving average, used to measure how far the price is above or below its recent trend.
- Momentum (Close_t / Close_(t-7)): Momentum measures the rate and direction of price movement by comparing the current closing price to the closing price from previous periods. This ratio quantifies how much the price has changed over short and medium horizons.

3.3 Lag features

- Previous day returns(return_(t-1)): Previous day returns capture short-term price changes and serve as lagged indicators of market momentum. By including multiple lagged returns, we can account for recent trend patterns and potential autocorrelation in the time series.
- Previous day volume changes: Previous day volume changes capture short-term shifts in market activity. By including multiple lagged volume changes, we can observe patterns such as increasing participation, sudden spikes, or drops in trading activity, which often precede significant price moves.

3.4 Target Variable

- Target: The target variable was defined using next-day returns rather than same-day price movements to avoid information leakage and ensure the model predicts future price direction rather than describing already observed outcomes." To mitigate the effects of minor price fluctuations and market noise, a return threshold was implemented. A day was labeled as 1 if the next-day return exceeded +0.5%, and 0 otherwise.

3.5 Data Preparation for Training

- Handling Null Values

During feature engineering, several lag-based and moving average features were created, such as momentum, previous day returns, and moving averages. This process introduced null values, particularly in the top rows of the dataset, because these features rely on prior observations that do not exist at the start of the dataset.

To maintain data integrity and ensure the model could process all rows, these top rows with null values were removed. This step ensured that every observation included valid values for all engineered features.

- Handling Infinity Values

The computation of percentage changes in returns and volume changes sometimes produced infinity values, typically occurring when the denominator (previous day price or volume) was zero. Infinity values cannot be handled directly by most models and could lead to errors during training.

To address this:

- All infinity values were replaced with zero.
- This substitution assumes that a zero denominator effectively represents no change or negligible impact, which is reasonable for very small or missing prior values in the dataset.

4.0 Train/Validation/Test Split

4.1 Feature and Target Separation

After cleaning the dataset and handling null and infinity values, the next step was to separate the input features from the target variable for modeling purposes:

- Input Features (X): All historical OHLCV data and engineered metrics used to predict the target, excluding non-predictive identifiers.
  - Columns dropped: Date (temporal index), Currency (categorical identifier), Target (dependent variable).
- Output: Binary prediction, 1 = price up, 0 = price down. The variable to be predicted, in this case, Target, represents the outcome or class label.

4.2 Time Series Splitting

The dataset was split using a time-aware approach to avoid data leakage. An 80% cut-off date (13 June 2021) was used to separate training and testing data. Within the training period, a further 70/30 split was applied to create a validation set. This ensures that the model is always trained on past data and evaluated on future data, consistent with real-world forecasting.

The dataset is divided into three separate subsets for modeling purposes:

- Training Set: Used to fit the model and learn patterns from historical data.
- Validation Set: Used to tune hyperparameters, select models, and avoid overfitting.
- Test Set: Used to evaluate the final model's performance on unseen data.

4.3 Per-Coin Feature Scaling

To account for differences in the magnitude of features across different cryptocurrencies (coins), numerical features were scaled independently per coin. This ensures that each coin's features are normalized within its own range, avoiding distortions caused by large differences in absolute values between coins.

- Methodology
    - Identify numerical features: All integer and float columns except identifiers and categorical variables.
    - Fit Min-Max Scaler on training data per coin: For each coin in the training set, a separate Min-Max Scaler was fitted to its numerical features. This preserves the relative structure of the coin's data while scaling between 0 and 1.
    - Apply the same scaler to validation and test sets: The scaler fitted on the training set for each coin was used to transform the corresponding rows in the validation and test sets. This ensures no data leakage from future observations.

Feature scaling was performed using the Min-Max Scaler. To avoid data leakage in this time-series problem, the scaler was fitted only on the training data and subsequently applied to the validation and test sets. This ensures that future information does not influence model training.

4.4 Categorical Encoding (One-Hot Encoding)

The dataset includes a categorical feature, Coin, representing different cryptocurrencies. Machine learning models typically require numerical inputs, so this feature was transformed using one-hot encoding.

- Methodology

- One-Hot Encode Training Data: Each unique coin in the training set was converted into a separate binary column.
  drop_first=True was applied to avoid multicollinearity (the first category is used as the baseline).
- Align Validation and Test Data: To ensure consistent feature columns across all datasets, validation and test sets were reindexed to match the training set.
- Any missing columns (coins not present in validation/test) were filled with 0, ensuring no mismatch during modeling.

5.0 Modeling Approaches

In this study, three modeling approaches were used to evaluate cryptocurrency price movement prediction. Each approach differed in coin selection, feature engineering, and model choice.

5.1 First Approach

The first approach focused on 10 cryptocurrencies, selected based on trading volume, popularity, and market relevance. Feature-engineered variables were created and rounded to two decimal places to standardize values and improve interpretability.

Data preprocessing involved scaling each coin separately to account for differences in magnitude between coins and features. Categorical variables were handled using one-hot encoding (dummy method).

For model evaluation, Logistic Regression and Random Forest Regression were applied to the prepared dataset. This approach aimed to assess predictive performance on a controlled set of high-volume, widely traded coins.

5.2 Second Approach

The second approach extended the first by incorporating additional features. Using the same set of 10 coins and the same feature variables as the First Approach, but not rounded up, an Open_Close range feature was added to capture intraday price movement.

The target variable was based on Next Close price and not Daily_Return, so it was not re-created for this approach. Preprocessing included standard scaling for numerical features and standard one-hot encoding for categorical variables.

The model used in this approach was a pipeline with a Logistic Regression and Gradient Boosting Classifier, allowing for sequential learning and better handling of complex, non-linear relationships.

This approach tested whether adding volatility-related features improved predictive accuracy over the First Approach.

5.3 Third Approach

The third approach explored a different coin selection strategy, using cryptocurrencies that had sufficient historical data to ensure the model could learn from a robust training set. Coins were selected based on their availability across trading days, emphasizing early-starting coins like Bitcoin, which provided the longest time-series data.

The same features from the First Approach were used, and preprocessing included scaling by coin and one-hot encoding. Logistic Regression and Random Forest Regression were again applied to evaluate model performance.

This approach investigated whether predictive accuracy could be improved by focusing on coins with more extensive training data, under the hypothesis that models perform better with longer historical coverage.

6.0 Baseline Models

Before evaluating the main machine learning model, baseline models were implemented to provide a reference point. Baselines represent simple, naive strategies that the ML model should outperform.

6.1 Baseline 1: Naive "No-Change" Model

Description: Predict that tomorrow's price movement will be the same as today's.

Rationale: This captures short-term persistence in price direction, which is common in high-frequency data.

6.2 Baseline 2: Always "Up" Model

Description: Predict that the price will increase every day, i.e., always predict an upward movement.

Rationale: Cryptocurrencies often exhibit long-term upward bias, so this naive assumption may achieve surprisingly high accuracy over long horizons.

These baselines provide a benchmark. The main machine learning model should outperform these trivial strategies to demonstrate that it is capturing meaningful patterns in the data rather than relying on naive heuristics.

7.0 Machine Learning Models

After establishing baseline performance, machine learning models were implemented to predict the next-day price direction based on the engineered features.

7.1 Logistic Regression

Logistic Regression is a simple linear classification algorithm that models the probability of a binary outcome (e.g., Up = 1, Down = 0) using a logistic function. It is often used as a baseline ML model due to its interpretability and was chosen as a simple linear classifier to capture the relationship between engineered features and price direction. Hyperparameter tuning was performed to optimize model performance.

- Hyperparameters tuned: To optimize the performance of the Logistic Regression model, manual hyperparameter tuning was performed using the validation set. The regularization parameter C (inverse of regularization strength) was selected because it directly controls the model's bias–variance trade-off.

  Methodology

  - A set of candidate values for C was defined: $C \in \{0.01, 0.1, 1, 10\}$
  - For each value of C:
    - The model was trained on the training set.
    - Performance was evaluated on the validation set using accuracy.
  - The value of C that produced the highest validation accuracy was selected as the optimal hyperparameter.

7.2 Random Forest Classifier

Random Forest is an ensemble of decision trees that reduces overfitting, handles feature interactions, and captures non-linear relationships and complex patterns between features and the target. It is well-suited for time-series classification with multiple features, including momentum, returns, and volume changes.

- Hyperparameters tuned: To improve predictive performance, the Random Forest model was tuned using the validation set. Two key hyperparameters were selected for tuning: maximum tree depth and number of trees, as they directly influence model complexity and generalization.

  Methodology

  - Hyperparameters tuned:
    - max_depth: {3, 5, 10}

        - n_estimators: {100, 200}
    - For each combination of tree depth and number of trees:
        - The model was trained on the training set.
        - Validation accuracy was computed using the validation set.
        - The combination that achieved the highest validation accuracy was selected as the optimal configuration.

7.3 Model Training Using a Pipeline

A machine learning pipeline was used to combine data preprocessing and model training into a single workflow. The preprocessing step handles feature transformations such as encoding and scaling, while the Logistic Regression classifier learns the relationship between the input features and the target variable.

Logistic Regression was selected for its interpretability and efficiency in binary classification tasks. The class_weight="balanced" parameter addresses class imbalance by giving more weight to the minority class, and max_iter=2000 ensures the model converges during training. Using a pipeline prevents data leakage and ensures consistent preprocessing across training, validation, and test datasets.

7.4 Gradient Boosting Classifier Model

A Gradient Boosting model was implemented using a pipeline that integrates data preprocessing with a Histogram-based Gradient Boosting classifier. This model builds an ensemble of decision trees sequentially, where each new tree corrects the errors of the previous ones, allowing it to capture complex and non-linear relationships in the data.

The model was configured with a maximum tree depth of 6 and a learning rate of 0.05 to balance model complexity and generalization. After training on the training dataset, the model's performance was evaluated on the test set using accuracy and a classification report, providing insight into both overall performance and class-specific prediction quality.

8.0 Evaluation & Results

8.1 Evaluation Metrics

The trained models were evaluated using standard classification metrics to assess predictive performance and generalization ability. The following metrics were used:

- Accuracy: Measures the overall proportion of correct predictions made by the model across all classes.

- Precision: Measures how many of the instances predicted as positive are actually positive, indicating the reliability of positive predictions.
- Recall: Measures how many of the actual positive instances were correctly identified, reflecting the model's ability to capture true positives.
- F1-Score: The harmonic mean of precision and recall, used to balance both metrics when class distribution is uneven.
- Confusion Matrix: A table that summarizes prediction results by showing true positives, true negatives, false positives, and false negatives, helping to visualize and analyze classification errors.
- Directional Accuracy: Measures how often the model correctly predicts the direction of movement (e.g., price going up or down), regardless of the exact magnitude. It is used to evaluate the model's ability to capture market trends and make correct directional decisions, which is especially important in financial forecasting and trading contexts.
- Classification Report: Provides a detailed evaluation of model performance by reporting precision, recall, F1-score, and support for each class. It is used to assess how well the model distinguishes between classes, identify class imbalances, and understand the types of errors the model makes beyond overall accuracy.

8.2 Results

8.2.1 Baseline Model Performance

- Naive Baseline Accuracy

This baseline serves as a reference point for determining whether our predictive model adds meaningful value beyond a trivial solution.

- First Approach

Naive baseline accuracy: 0.716 (≈71.6%)

Interpretation: Approximately 71.6% of the dataset belongs to the majority class, meaning a model that predicts this class for all examples would achieve 71.6% accuracy without learning any patterns.

- Second Approach

Naive baseline accuracy: 0.7208 (≈72.1%)

Interpretation: Approximately 72.1% of the dataset belongs to the majority class, meaning a model that predicts this class for all examples would achieve 72.1% accuracy without learning any patterns.

- Third Approach

Naive baseline accuracy: 0.7548 (≈ 75.5%)

Interpretation: Approximately 75.5% of the dataset belongs to the majority class, meaning a model that predicts this class for all examples would achieve 75.5% accuracy without learning any patterns.

- Always-Up Baseline Accuracy

The always-up baseline predicts that the target always goes "up" (positive class) for all observations.

- First Approach

Always-Up Baseline Accuracy: 0.4034 (≈ 40.3%)

Interpretation: An accuracy of 40.3% indicates that this single class occurs in about 40% of the dataset.

- Second Approach

Always-Up Baseline Accuracy: 0.4888 (≈ 49%)

Interpretation: This result indicates that upward movements occur in roughly 49% of the dataset, showing that the data is relatively balanced in terms of direction.

- Third Approach

Always-Up Baseline Accuracy: 0.4655 (≈ 46.6%)

Interpretation: An accuracy of 46.6% means the "up" class represents about 46–47% of the dataset. This confirms the dataset is not balanced, but also not extremely skewed.

8.2.2 Machine Learning Model Performance

8.2.2.1 First Approach

- Logistic Regression
  - Validation Set

Validation Accuracy: 0.579 (≈57.9%)

Interpretation:

- The model correctly predicts the target approximately 57.9% of the time on the validation set.
- Compared to the naive baseline of 71.6%, this indicates that Logistic Regression currently underperforms and does not capture the patterns in the data effectively.
- The lower accuracy may be due to class imbalance or limited feature information.

- Hyperparameter Tuning

The Logistic Regression model was trained with different values of the regularization parameter C, and validation accuracy was measured for each setting:

C	Validation Accuracy
0.01	0.5770
0.1	0.5821
1	0.5789
10	0.5808

Best C: 0.1 (Validation Accuracy ≈ 58.2%)

Interpretation:

- The model performs similarly across different C values, with minor differences in validation accuracy.
- C=0.1 provides the best performance, balancing regularization and fit to the data.
- Despite tuning, the validation accuracy (~58%) is still lower than the naive baseline (71.6%), suggesting that Logistic Regression may struggle with the current features or class imbalance.

- Test Set

Test Accuracy: 0.5934

Interpretation:

- The Logistic Regression model achieves ≈59.3% accuracy on the test set, showing slight improvement over validation accuracy.
- Despite tuning, this accuracy is below the naive baseline of 71.6%, indicating that the model does not fully capture patterns in the dataset.
- The relatively low performance may be due to class imbalance, limited or uninformative features, or the linear nature of Logistic Regression.

- Model Evaluation

Metrics

Metric	Value
Accuracy	0.5934
Precision	0.4896
Recall	0.1897
F1 Score	0.2735

Directional Accuracy	0.5934

Confusion Matrix

	Predicted Down	**Predicted Up**
Actual Down	2067	319
Actual Up	1307	306

Interpretation

- Accuracy: The model correctly predicts the target approximately 59% of the time. This is below the naive baseline of 71.6%, suggesting the model struggles to outperform a simple majority-class prediction.
- Precision: Of all predicted "up" cases, roughly 49% were correct.
- Recall: The model identifies only ~19% of actual "up" cases, indicating poor detection of the minority class.
- F1 Score: Combines precision and recall; a low score reflects the difficulty in predicting minority class instances.
- Directional Accuracy: Matches overall accuracy, showing that the model is slightly better than random at predicting movement direction.
- Confusion Matrix Insights: The model predicts the majority class ("down") well (2067 correct), but fails to detect most minority class instances (1307 misclassified "up" cases). This confirms that class imbalance heavily affects performance.

- Random Forest Classifier
  - Validation Set

Validation Accuracy: 0.578 (≈57.9%)

Interpretation:

- The Random Forest model achieves roughly 57.9% accuracy on the validation set.
- This performance is comparable to the Logistic Regression model and below the naive baseline of 71.6%.

- Despite being a non-linear model, Random Forest does not yet outperform the simple majority-class prediction, suggesting:
  The current features may not capture strong predictive signals.
  Class imbalance may be affecting performance.

- Hyperparameter Tuning

Validation accuracy for each configuration is shown below:

Max Depth	Trees	Validation Accuracy
3	100	0.5754
3	200	0.5754
5	100	0.5777
5	200	0.5777
10	100	0.5787
10	200	0.5814

Best configuration: Depth = 10, Trees = 200

Best Validation Accuracy: 0.5814 (≈58.1%)

Interpretation

- Increasing depth slightly improves performance, as deeper trees capture more complex patterns.
- Doubling the number of trees marginally improves accuracy, indicating the model benefits slightly from more estimators.
- The best accuracy (58.1%) is still below the naive baseline (71.6%), suggesting:
  Features may not contain enough predictive information.
  Class imbalance in the dataset affects minority class prediction.

- Test Set

Test Accuracy: 0.5981 (≈59.8%)

Interpretation:

- Increasing tree depth and number of trees slightly improved model performance.
- The Random Forest achieves ≈59.8% accuracy on the test set, slightly better than the best validation accuracy.

- Despite being a non-linear ensemble model, it does not outperform the naive baseline (71.6%), suggesting:
    - Features may not fully capture the predictive signal.
    - Class imbalance may hinder performance on minority classes.

- Model Evaluation

Metrics

Metric	Value
Accuracy	0.5981
Precision	0.5625
Recall	0.0167
F1 Score	0.0325
Directional Accuracy	0.5981

Confusion Matrix

	Predicted Down	Predicted Up
Actual Down	2365	21
Actual Up	1586	27

Interpretation:

- Accuracy: The model correctly predicts the target about 59.8% of the time. Slight improvement over Logistic Regression, but still below the naive baseline (71.6%).
- Precision: Of all predicted "up" cases, about 56% are correct.
- Recall: The model identifies only ~1.7% of actual "up" cases, showing extremely poor detection of the minority class.
- F1 Score: A low value reflects the model's difficulty in predicting minority-class instances.
- Directional Accuracy: Same as overall accuracy; the model is slightly better than random at predicting movement direction.
- Confusion Matrix Insights: The model predicts the majority class ("down") very well (2365 correct) but fails to detect most minority-class instances (1586

misclassified "up" cases). Confirms that class imbalance heavily affects performance.

8.2.2.2 Second Approach

- Pipeline Model
  - Validation Set

Validation Accuracy: 0.529 (≈52.9%)

Confusion Matrix

	Predicted 0	**Predicted 1**
Actual 0	1052	1224
Actual 1	1034	1482

Classification Report

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.50	0.46	0.48	2276
1	0.55	0.59	0.57	2516
Macro avg	0.53	0.53	0.52	4792
Weighted avg	0.53	0.53	0.53	4792

Interpretation

- Accuracy (~52.9%) is lower than both Logistic Regression (~59.3%) and Random Forest (~59.8%), and well below the naive baseline (71.6%).
- Class-wise performance:
  Class 0 (majority class) is slightly under-predicted (precision 0.50, recall 0.46).
  Class 1 (minority class) is slightly better predicted (precision 0.55, recall 0.59).
- F1 Scores (0.48–0.57) indicate that the Pipeline model achieves a more balanced performance across classes, unlike Logistic Regression or Random Forest, which heavily favor the majority class.

Summary

- The Pipeline model achieves balanced predictions across classes but at the cost of overall lower accuracy.

- This shows a trade-off between accuracy and class balance, which may be preferable if minority-class detection is important.
- Further improvements could involve feature selection, hyperparameter tuning, or adjusting class weights within the pipeline.

- Test Set

Test Accuracy: 0.538 (≈53.8%)

Confusion Matrix

	Predicted 0	**Predicted 1**
Actual 0	1022	1022
Actual 1	827	1128

Classification Report

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.55	0.50	0.53	2044
1	0.52	0.58	0.55	1955
Macro avg	0.54	0.54	0.54	3999
Weighted avg	0.54	0.54	0.54	3999

Interpretation

- Overall Accuracy (~53.8%): Slightly improved from the validation accuracy (~52.9%). Still below the naive baseline (71.6%), but the model is more balanced across classes.
- Class-wise Performance

  Class 0: Precision 0.55, Recall 0.50 → moderate performance.

  Class 1: Precision 0.52, Recall 0.58 → slightly better recall than Class 0.

  The Pipeline achieves a more balanced trade-off between classes compared to Logistic Regression and Random Forest, which heavily favored the majority class.
- F1 Scores: Shows that the model handles both classes more evenly, even if overall accuracy is lower.

- Confusion Matrix Insights: Predictions are distributed more evenly across both classes, reducing extreme bias toward the majority class.

Summary

- The Pipeline model prioritizes balanced prediction over raw accuracy.
- While overall test accuracy (~53.8%) is lower than that of other models, the Pipeline handles class imbalance better, achieving comparable precision and recall for both classes.
- Further improvements could include feature engineering, hyperparameter tuning, or ensemble techniques to raise both accuracy and minority-class performance.

- Gradient Boosting Classifier
  - Test Set

Test Accuracy: 0.557 (≈55.7%)

Classification Report

Class	Precision	Recall	F1-Score	Support
0	0.56	0.65	0.60	2044
1	0.56	0.46	0.50	1955
Macro avg	0.56	0.56	0.55	3999
Weighted avg	0.56	0.56	0.55	3999

Interpretation

- Accuracy (~55.7%): Better than Logistic Regression (~59.3% val / 59.8% RF test), Pipeline (~53.8%), and slightly better than Random Forest test (~59.8%); shows moderate overall performance.
- Class-wise Performance

  Class 0: Recall 0.65 → most majority-class instances are detected correctly.

  Class 1: Recall 0.46 → still struggles with minority-class detection, but slightly better balance than Random Forest.
- F1 Scores (0.50–0.60): Reflects a reasonable trade-off between precision and recall for both classes, more balanced than previous models.

Summary

- Gradient Boosting achieves moderate overall accuracy (~55.7%) while improving minority-class balance compared to Logistic Regression and Random Forest.
- This indicates that ensemble boosting methods capture some patterns better, though class imbalance still affects performance.
- Further improvements could involve hyperparameter tuning, feature engineering, and class weighting to enhance recall for the minority class.

8.2.2.2 Third Approach

- Logistic Regression –
  - Validation Set

Validation Accuracy: 0.5350 (≈ 53.5%)

Interpretation

- The model correctly predicts about 53.5% of validation samples.
- This is higher than the always-up baseline (~46.6%), indicating the model is learning some signal.
- However, it performs well below the naive baseline (~75.5%), highlighting strong class imbalance and the limitation of accuracy as a sole metric.

- Hyperparameter Tuning (C)

The Logistic Regression model was trained with different values of the regularization parameter C, and validation accuracy was measured for each setting:

C	Validation Accuracy
0.01	0.53499
0.1	0.53499
1	0.53499
10	0.53499

Best C: 0.01 (tie, selected by first occurrence)

Interpretation

- Changing the regularization strength (C) had no effect on validation accuracy. This suggests:

  The model is insensitive to regularization in this range, or

The predictive signal is weak / dominated by class imbalance, or

Features are already heavily constrained (e.g., one-hot encoding, limited variance).

Conclusion

- Hyperparameter tuning for C did not improve performance.
- Further gains are more likely from feature engineering, class weighting, or non-linear models, rather than adjusting regularization strength.

- Test Set

Test Accuracy: 0.5345 (≈ 53.4%)

Interpretation

- Test accuracy is very close to validation accuracy (≈53.5%), indicating good generalization and no obvious overfitting.
- Performance is above the always-up baseline (~46.6%), showing the model learns some structure.
- However, it remains well below the naive baseline (~75.5%), reinforcing that accuracy alone is misleading due to class imbalance.

Conclusion

- Logistic Regression provides stable but limited performance.
- The model captures weak linear signals, and meaningful improvement likely requires richer features or more expressive models rather than further tuning of C.

- Model Evaluation

Test Performance

Metric	**Value**
Accuracy	0.5345
Precision	0.0
Recall	0.0
F1 Score	0.0
Directional Accuracy	0.5345

Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	504	0
Actual 1	439	0

Interpretation

- The model predicts only class 0 and completely misses class 1.
- Accuracy (~53.4%) is misleading, as it only reflects the proportion of class 0 in the test set.
- Precision, recall, and F1-score are all 0, confirming the model fails to identify the positive class.
- This behavior indicates strong class imbalance and the model's inability to capture meaningful patterns.

○ Random Forest

- Validation Set

Validation Accuracy: 0.5350 (≈ 53.5%)

Interpretation

- The model correctly predicts about 53.5% of validation samples.
- This is slightly above the always-up baseline (~46.6%), indicating the model captures some structure in the data.
- However, performance is well below the naive baseline (~75.5%), highlighting strong class imbalance and limited predictive features.
- Random Forest shows minimal learning, performing similarly to Logistic Regression.
- The low accuracy suggests that the current features or dataset are insufficient to train a non-linear ensemble effectively.

- Hyperparameter Tuning

Max Depth	Trees	Validation Accuracy
3	100	0.53499
3	200	0.53499
5	100	0.53499
5	200	0.53499

10	100	0.53499
10	200	0.53499

Interpretation

- Changing tree depth or number of trees had no effect on performance.
- The model is likely limited by class imbalance or low predictive signal in the features.
- Random Forest, in this setup, behaves almost identically to Logistic Regression.

- Test Performance

Test Accuracy: 0.5345 (≈ 53.4%)

Interpretation

- Test accuracy is very close to validation accuracy (≈53.5%), indicating no overfitting.
- The model performs only slightly better than the always-up baseline (~46.6%), suggesting minimal learning.
- Performance is well below the naive baseline (~75.5%), showing that class imbalance and limited predictive features dominate the model behavior.

- Model Evaluation

Metrics

Metric	**Value**
Accuracy	0.5345
Precision	0.0
Recall	0.0
F1 Score	0.0
Directional Accuracy	0.5345

Confusion Matrix

	Predicted 0	**Predicted 1**
Actual 0	504	0
Actual 1	439	0

Interpretation

- The model predicts only class 0 for all test samples.
- All 439 positive samples were missed, leading to 0 precision, 0 recall, and 0 F1-score for the positive class.
- Accuracy (~53.4%) is misleading, as it comes entirely from correctly predicting the majority class.
- This behavior is identical to Logistic Regression in this setup, indicating minimal learning and dominance of class imbalance.

Conclusion

- Random Forest fails to detect the minority class and acts effectively as a majority-class predictor.
- Hyperparameter changes (tree depth, number of trees) did not improve performance.

9.0 Discussion & Limitations

9.1 Discussion

Among the three modeling approaches, the First Approach produced the best overall results. Selecting the top 10 cryptocurrencies based on trading volume, popularity, and market relevance ensured that the dataset contained sufficient and consistent historical data, which improved model learning and stability. Feature engineering in this approach was straightforward and effective, and scaling features per coin helped account for differences in price magnitude across assets.

The use of Logistic Regression and Random Forest models in the first approach provided reliable performance, particularly in predicting price direction. These models benefited from the structured feature set and the availability of adequate training data for each coin, resulting in better generalization compared to the other approaches.

The second approach, which introduced additional features such as the Open–Close range and employed a Gradient Boosting model, showed potential in capturing non-linear relationships. However, the added complexity did not consistently translate into improved performance, suggesting that feature richness alone does not guarantee better results when the underlying signal is weak or noisy.

The third approach explored the impact of coin selection based on historical availability. While early-starting coins such as Bitcoin provided longer time series, limiting the coin set reduced data diversity and constrained the model's ability to generalize across different market behaviors.

Baseline models were effective in revealing the class distribution of the dataset and provided useful reference points for evaluating machine learning models.

The trained models showed stable performance, with validation and test accuracies being very similar, indicating no overfitting.

Gradient Boosting achieved a relatively better balance between precision and recall compared to Logistic Regression and Random Forest, showing that ensemble methods can extract slightly more information from the data.

- What Doesn't Work
  - Logistic Regression and Random Forest failed to predict the minority class, resulting in zero precision, recall, and F1-score despite moderate accuracy.
  - Hyperparameter tuning improves performance, suggesting that model adjustments alone are insufficient.
  - Accuracy proved to be a misleading metric due to class imbalance, as high accuracy did not translate into meaningful predictions.
  - The models struggled to capture directional movement, indicating weak predictive signals in the features.

9.2 Limitations

- Data source is only historical OHLCV
  - Limitation: The dataset has only price & volume.
  - Why it matters: Crypto prices are heavily influenced by news, regulations, macroeconomics, social media sentiment, on-chain activity, etc. Your model is blind to all that, so it can only capture technical patterns, not fundamental drivers.
- No guarantee that past patterns repeat (concept drift)
  - Limitation: Model assumes relationships learned from historical data remain stable.
  - Why: Crypto markets evolve fast - new regulations, exchanges, stablecoins, big crashes (e.g. FTX), etc. Patterns that worked in the past may completely break. This is called concept drift; it makes long-term predictive performance unstable.

- Top 10 coins only
  - Limitation: The dataset includes only the largest coins.
  - Why: Results may not generalise to: Smaller, illiquid coins and newly launched tokens with different behaviours.
- Possible survivorship bias
  - Limitation: Depending on how the dataset was built, it might include only coins that remained in the top 10.
  - Why: Coins that failed or dropped out may be missing the data is skewed toward winners, making the market look more predictable and stable than it actually is.
- Single-market / single-source biases
  - Limitation: Historical prices may come from one or a few exchanges.
  - Why: Real crypto markets involve many exchanges with different liquidity and spreads. The model might not capture: Arbitrage effects, Slippage, Order book depth.
- No transaction costs or execution risk in your evaluation
  - Limitation: When simulating trading strategies, these are usually ignored: Trading fees, Bid-ask spread, Latency & slippage.
  - Why: Even if the model shows good paper profits, once you subtract real-world costs, the strategy might not be profitable.
- Risk of overfitting
  - Limitation: With many engineered features and relatively few years of data, complex models (e.g. deep nets, random forests) can easily memorise patterns.
  - Why: The overfitting model performs very well on training/validation data but poorly on unseen future periods.
- Academic/educational scope only
  - Limitation: This project is for learning and experimentation, not a production trading system.
  - Why: Missing critical components: live data feeds, rigorous risk management, stress tests, regulatory/compliance constraints, etc.

10.0 Conclusion & Future Work

10.1 Conclusion

This study evaluated multiple baseline and machine learning approaches for directional prediction. While some models achieved moderate accuracy, deeper analysis revealed that accuracy alone was insufficient due to class imbalance. Logistic Regression and Random Forest models consistently predicted the majority class, resulting in poor precision, recall, and F1-scores for the minority class.

Among the tested approaches, ensemble-based methods—particularly Gradient Boosting—performed better, offering a more balanced trade-off between precision and recall. Baseline models were useful for contextual understanding, but they did not provide meaningful predictive power. Overall, the results indicate that data characteristics and feature quality had a greater impact on performance than model complexity.

10.2 Future Work

To improve predictive performance, future work should focus on:

- Addressing class imbalance: Apply techniques such as class weighting, oversampling (e.g., SMOTE), or under sampling.
- Enhanced feature engineering: Incorporate technical indicators, rolling statistics, and volatility-based features to strengthen predictive signals.
- Advanced models: Explore more sophisticated algorithms such as XGBoost, LightGBM, or neural networks.
- Temporal validation: Use time-aware cross-validation to better reflect real-world prediction scenarios.

Final Remark

This work demonstrates that model performance is strongly constrained by data quality and structure, and meaningful improvements require better features and imbalance handling rather than simply increasing model complexity.