Comparison of New Classification Model Applied to Bitcoin Price Moves with Traditional Models

Coby Walmsley, Prakhar Singh, Rishabh Kumar
ISE 464 Final Project
Masters in Financial Engineering

Abstract:

While attempting to predict future asset price motion, two approaches are used broadly throughout academic papers. The ARIMA/GARCH family of approaches used by time series forecasters are often in contrast with the ML Classifier family of approaches used by machine learning experts. The inspiration for this model was to create a regression framework that can regress on the historical effects of lagged time series' on a contemporaneous y time series. This model has the added benefit of a PCA-like component analysis that eliminates multicollinearity among factors which would otherwise tend to artificially inflate regression coefficients. This regression model creates a series of orthogonal factors that have interpretable beta coefficients that can be used to predict the direction of a price motion of a future asset with a novel classifier model.

Background and Methodology:

The concept of beta as a predictor of asset prices has been widely used in finance to describe relationships between contemporaneous time series. However, beta is much less widely used as a predictive factor because its relationships are much less stable. The equation for the beta between two time series is

$$\hat{\beta} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

For two contemporaneous time series, a beta of 1.3 implies that if stock X moves up 1%, stock will move up 1.3% on average. Beta is based on the linear regression equation, but what if beta was calculated between two time series that were lagged? A beta of 1.3 between X_{t-1} and Y would imply that historically, if stock A moves up 1%, stock Y is expected to move up 1.3% *tomorrow.* But betas so strong are rare and fragile. What if we could get the betas at multiple lags of the same time series in order to calculate the full effect that all past lags have on the target variable? This is a concept similar to the Partial Autocorrelation Function from the time series forecasting field. If we simply calculate beta for each lag of a time series, a lot of the information will be repeated. When we combine these beta factors to get our final predictive factor, this repeated information will artificially amplify our predictive value. We need a way to counteract this problem.

Eliminating Multicollinearity Between Features

If we are seeking to eliminate multicollinearity, we must ensure that all of the features we input into our classifier model are orthogonal in the higher-dimensional space the vectors exist in. To begin this process, we start with an unaltered series of time series that we wish to use as features, as well as a time series that we wish to predict the next value of. The reason I have focused on direction of the move instead of magnitude of the move is that in financial engineering, if a weak signal is identified, leverage can be used to amplify that signal into a trading strategy that can be much more profitable than the actual magnitude of the move. As long as price direction can be predicted, profitability can be ensured.

We start by transforming our features and target variable from price at every time index to log return between every time index. This standardizes our features well and puts them in a form where their returns are additive. Next we want to add features to our matrix one by one and make sure each feature is completely orthogonal to the other features. If the feature is not orthogonal, we can apply the following transformation. In this example, we will be using three features even though our data uses dozens. We start by adding feature A into our matrix of features. Feature A needs no alterations (besides scaling) because there are no other vectors for it to have collinearity with yet. So we add feature B. To eliminate any possible multicollinearity, we run a linear regression to try to predict feature B with feature A. The residuals from this linear regression represent all information present in feature B that could not be captured by feature A. We take the residuals from this regression and add it to the second column of our feature matrix. For feature C, we run a multivariate linear regression that attempts to predict the values of C using feature A and our modified feature B. The residuals from this regression now can now be added to our feature matrix and we can confirm that our modified feature C shares no information with our feature A and feature B. Having transformed the features in this way, we can now calculate our beta coefficients between each time series and our final dataset.

Regularization

As the reader may surmise, as more and more features are added to our feature matrix that share information, the existing features will become increasingly able to predict the values in a new feature, making some residuals obtained by the regression matrix almost zero. As part of my model, I drop any candidate features from the model that have an average absolute value near zero. I have also applied regularization to the beta coefficients that my model calculates from each transformed feature. Since the final equation to find the classification coefficient will be to add the final value of all the transformed features multiplied by its respective beta coefficient, If the classification coefficient is above zero, this indicates a positive move while if the classification coefficient is below zero this indicates a negative move. My classifier will treat any

calculated log return higher than 5% as a predicted move up, a log return lower than -5% as a predicted move down, and ignore everything else.

Additional Modifications

The reader may also notice that this model only accounts for linear relationships between an input (lagged or unlagged) variable and the target variable. But sometimes, better stock signals come from the joint relationship between two variables. To account for this, my model has an option to include, instead of just every input feature, every possible combination of multiplied input features. While this may increase accuracy, with a large number of input features, the resulting number of products increases exponentially and increases fitting time. It is also important to note that the features are multiplied together before the log return transformation is applied to keep the features standardized. Also, while I used log-differenced features to provide some scaling, since this is a regressive model it will also work without scaling. The betas are easier to compare and interpret when the features are scaled.

Hyperparameters

Our two hyperparameters for this model are h, the amount of lags per feature to include in the creation of our new feature table, and B, the size of the rolling window before our current point that will be included in our linear regressions to calculate beta. While testing smaller data sizes, we found that with t greater than 3, the beta coefficients that were calculated were near zero, indicating that for ETFs, a t parameter of 3 is sufficient. As a future improvement, we could calculate the autocorrelation function for each feature and then set a unique t parameter for each feature corresponding to the strength of the autocorrelation at lag t. For B, we did not find any improvement on smaller datasets past 30. The difficulty with setting B is that if B gets large enough, the beta coefficient will drop to zero as the regression overfits random noise. If B is too small, there is not enough data to capture the strength of a real relationship. We found that a healthy balance could be found in the range of B=30-50. For our final test, we used a B of 40.

Data and Preparation

Since the goal of this project is to predict the movements of asset prices, we choose a fairly arbitrary asset to begin. For this trial we have chosen Bitcoin as our target variable. Now we select our features. In a situation with no compute restraints, we would insert as much data as we possibly could because we know that our repeated regressions will eliminate any multicollinearity in the data. But to simplify this approach, we use a group of 73 ETFs that collectively encompass over 97% of stocks in the stock market. Each ETF is a representative asset that defines the daily movement of a particular industry or sector. The complete list of feature ETFs and their sectors is listed in the appendix. For each ETF we create a new feature for each lag as defined in the hyperparameters. Then, to account for any non-linear relationships we

square every unlagged ETF series and use a linear regression to add them to the feature matrix in the same way that the unsquared features were added. While it would be possible to multiply all unlagged time series together to capture all possible relationships between features, this would increase the number of features from 76*h to 76!. We quickly realized that this method would be infeasible with our compute capability, so we stuck to 76*h features.

Performance

After trying a series of beta windows and number of lags, we found that three lags per feature with a beta of 25 trading days, or 5 weeks worked the best. We set our number of lags first, then altered our beta window until we got the best performance on the training data. To quantify our model's performance, we used a rolling window of size B as our training data, applied our regressive transformations (optimized with linear algebra), and made a prediction for the next point in the time series. This process then repeated for the next rolling window. The rolling window approach eliminates lookahead bias and also creates a fairly large test set of data that accuracy can be displayed on. We also tuned our classifier coefficient until the model was predicting around 25% of points as moves up or down. The classifier coefficient was used as previously discussed to classify points into up, down, or neutral moves. For our best trial, we used a classifier coefficient of 3%.

Results:
While the elimination of multicollinearity as a concept is interesting and may have other applications to time series data, we were not able to get any groundbreaking accuracy levels with our model. Our accuracy for prediction of price moves up was 54% and our accuracy for moves down was 48%. While this is some small edge, it is unlikely that this is more than random noise. Our new model does not perform well, so we compare it to conventionally accepted models.

Comparison to Other Models

The results we obtained with our novel model were not successful or interesting enough to determine that it is extremely effective in predicting the motion of stock prices. But classifying stock price moves is already an extremely difficult task, so we decided to try other more conventionally accepted models using the same features to see what results we could obtain and if they were better than our novel model.

Comparison Analysis 1- Prakhar Singh

The foundation of the model was getting daily Bitcoin price data from Yahoo Finance. After downloading the data, I cleaned it and kept only Open, High, Low, Close, and Volume. Then I converted tomorrow's movement into a simple Bernoulli/binary label“1” if the price goes up tomorrow and “0” if the price goes

down or stays flat. This helped turn the whole problem into one direct classification task can we guess tomorrow's direction, To start with meaningful features, I used log returns, which are a smooth version of daily price movement. Then I used a 20-day window for Lower boundary and upper boundary because I wanted the model to understand where the price is sitting, not just read raw numbers. I gave it a sense of position in the range, whether the price is drifting near the top, hanging around the bottom, or somewhere in the middle.Lower boundary and upper boundary define the recent trading boundaries of Bitcoin. Lower boundary represents the floor formed by the lowest price in the last 20 days, while upper boundary is the ceiling formed by the highest price in that same window. These two levels act as natural "thresholds" in price behavior: when Bitcoin approaches either boundary, the probability of a significant move increases. In the model, the classification threshold is the usual 0.5 cutoff for predicting UP versus DOWN, but the market threshold is different. It emerges from the distance-to-Lower boundary/upper boundary and volatility. If the price is close to a boundary and volatility is high, the barrier becomes weak and a breakout becomes more likely. This interpretation directly connects to the tunneling feature To capture that, I calculated the distance from the current price to both boundaries, because when prices approach these edges, the market usually reacts. I added volatility using the 10-day standard deviation because it basically shows the market's mood. After observing how price reacts around these levels, I realized the logic is very similar to physics: a particle crosses a barrier only when it is close enough or has enough energy. Financial prices behave the same way. So I defined Lower boundary as the lowest low of the past 20 days, upper boundary as the highest high of the past 20 days, and volatility as the 10-day standard deviation. Then I combined them into a feature called "tunneling pressure," where I divide the distance to each boundary by volatility. A lower value indicates the price can break through more easily. This feature connects the distance idea with volatility and gives the model a better sense of possible breakouts or breakdowns.

Upward tunneling:

$$T_t^{(up)} = \frac{d_t^{(resistance)}}{\sigma_t}$$

Downward tunneling:

$$T_t^{(down)} = \frac{d_t^{(support)}}{\sigma_t}$$

This tunneling design is loosely inspired by quantum tunneling, where a particle's ability to cross a barrier increases when the barrier is thinner or when the particle has more energy. In financial terms, a small distance-to-barrier acts like a thin barrier, and high volatility acts like increased energy. By defining the feature as distance divided by volatility, the model receives a physics-driven signal: large values imply a low chance of breakout, while values approaching zero imply high breakout potential. This gives the model an intuitive way to recognize when Bitcoin may be preparing to break out of its current price range.

Experiments / Analysis(Models):-

For the experiments, I started with a simple model and gradually moved to more flexible ones, comparing how each one sees the market. I began with Logistic Regression on purpose because I wanted a very basic baseline. This model is asking the data: “If I draw one straight line between UP days and DOWN days, does that line even make sense” To set everything up, I scaled all the features with StandardScaler, trained the model, predicted probabilities instead of just labels, and evaluated accuracy, F1-score, and ROC–AUC. I also plotted the ROC curve to see how well it separates the two classes. The results immediately showed something obvious about Bitcoin it does not follow clean, straight-line patterns. The performance was only slightly better than random, which I expected, but it confirmed the basics: the data pipeline worked, the labels were correct, the features were being interpreted properly, and there was no overfitting.

Then I moved to Random Forest. This model makes sense for this kind of problem because it handles nonlinear patterns, feature interactions, and noisy financial data much better. I kept the setup the same and again used predicted probabilities, evaluation metrics, ROC curves, probability scatter plots, and feature importance. This is where I felt the project start to make real progress. Random Forest captured more structure than Logistic Regression, especially in the Lower boundary/upper boundary distance features, volatility, and tunneling pressure. The F1-score and ROC–AUC improved. It wasn’t a huge jump, but enough to show that nonlinear models understand crypto behavior better. Even the probability scatter plot looked more meaningful, the model was no longer just guessing randomly. Working with Random Forest also showed me how sensitive financial data is and how even small feature improvements can make a difference.

Finally, I built an MLP Neural Network because I wanted a model that doesn’t follow strict preset rules. Neural networks learn patterns on their own. I built a simple two-layer network (64 and 32 neurons), trained it with Adam, set maxitex500, and monitored train-vs-test performance to avoid overfitting. Like with the other models, I focused on probability outputs, used the same evaluation metrics, checked calibration curves, and plotted probability distributions. This model behaved differently from Random Forest. The accuracy wasn’t the highest, but the ROC–AUC was the best out of all three models. The calibration curve showed the neural network wasn’t overconfident, and the probability distribution looked smoother and more meaningful. This made one thing clear: neural networks are better at ranking days by “how likely” the market is to move, even if they don’t always make the perfect UP/DOWN call. For crypto, which is extremely noisy, this ranking ability is often more useful than hard classification. This model also taught me the most about scaling, activation functions, training stability, and randomness

Discussion:-

After running all the models, the main takeaway was that Bitcoin’s daily direction is extremely noisy, but it’s not completely meaningless. Lower boundary and upper boundary features behaved exactly how they should. Volatility mattered in the moments where the market became unstable. And the tunneling feature added a small but noticeable push. All three models showed me that the market isn’t fully random it just has enough noise to hide most of the structure. Simple models break down fast, while flexible models like Random Forest and Neural Networks handle the noise a little better. The visual outputs, ROC curves, calibration curves, and probability distributions helped me understand how each model was behaving internally. The neural network’s bell-shaped probability curve especially stood out because it showed the natural confidence range of the model.

Comparison 1 Conclusion:

This project became more than just "predicting Bitcoin." It turned into a journey of connecting real market intuition, physics-inspired thinking, and machine learning techniques. The models didn't reach extremely high accuracy, but they consistently performed slightly above chance, which is already a win in something as chaotic as crypto. The whole process of cleaning data, building features, comparing models, and trying different ideas taught me more than the final numbers did. Although these engineered features worked well, the model could improve significantly with access to richer data such as order-book imbalance, funding rates, sentiment indicators, and volume-adjusted signals. Additional enhancements like hyperparameter tuning, rolling cross-validation, LSTM sequence models, and ensemble methods were beyond the scope of this project but represent clear next steps.Future versions of the tunneling feature could also incorporate deeper measures of market pressure, such as momentum bursts near boundaries or volatility clusters that behave like sudden increases in "energy," sharpening the breakout signal even further. Overall, this experiment showed me that with a bit of creativity (like the tunneling feature), machine learning can detect subtle patterns that traditional indicators might miss. And even when the market doesn't give strong signals, the learning that happens while testing and connecting ideas becomes the real takeaway for me.

Comparison Analysis 2 - Rishabh Kumar

Introduction

This project replicates and adapts the methodology presented in the research paper *Predicting Market Trends with Enhanced Technical Indicator Integration and Classification Models* by Hafid et al., which proposes an integrated technical indicator and machine learning framework for predicting cryptocurrency price direction. The authors show that combin- ing momentum-, trend-, volatility-, and volume-based technical indicators with classification models such as Logistic Regression and XGBoost produces highly accurate directional predictions.

Following the paper, we aim to classify Bitcoin (BTC-USD) daily trend direction using engineered technical indicators and a moving-average—based label. The target variable follows the rule defined in Equation (1) of the reference study:

$$\text{Signal} = \begin{cases} 1, & \text{if MA10} \geq \text{MA60} \\ 0, & \text{otherwise} \end{cases}$$

This formulation yields a stable classification target aligned with trend-following systems. The objective is to evaluate whether the methodology from the paper produces similar performance when applied to daily OHLCV data rather than 15-minute resolution.

Methodology

Daily BTC-USD OHLCV data (2019–2025) is retrieved from Yahoo Finance and cleaned for indicator generation. Technical indicators are computed exactly as outlined in Sections III(A)–III(D) of Hafid et al., including RSI(14,30), Momentum, ROC, MACD, EMA(10,30,200), Bollinger Bands, ATR(14), CCI(20), Stochastic Oscillator variants, OBV, Chaikin Money Flow, and the Accumulation/Distribution Line. Indicators follow the mathematical definitions provided in the study.

Consistent with the research methodology, we apply chi-square (X^2) feature selection as defined in Equation (6) of the paper to identify the most statistically relevant predictors. Two models are trained: Logistic Regression as a baseline and XGBoost as the primary classifier. We preserve chronological ordering using an 80%/20% time-series split.

Evaluation follows the paper's design, using accuracy, precision, recall, F1-score, ROC-AUC, and a confusion matrix. Additionally, a long-only backtest based on predicted signals is implemented as an extension beyond the original study.

Experiments / Analysis

Model performance closely matches the results reported by Hafid et al. XGBoost achieves approximately 91.9% accuracy and a ROC-AUC of 0.978, while Logistic Regression achieves 91.9% accuracy and 0.979 ROC-AUC. These values align with the research paper's findings, which report accuracy around 92.4% and AUC near 0.982 for XGBoost.

Feature selection and model-based importance rankings identify RSI, MACD, momentum indicators, and Stochastic Oscillator components as highly influential, consistent with Figure 6 (feature ranking) in the reference study. PCA results show strong redundancy among indicators, supporting the use of X^2 selection.

The backtest demonstrates trend-following behavior: the ML-driven strategy performs well in sideways or mildly trending markets but lags during aggressive bull runs due to the smoothing effect of the MA60 component. This limitation is also acknowledged in the discussion of the reference study.

Discussion

Our findings reinforce the conclusions of Hafid et al.: technical indicators contain substantial predictive information regarding moving-average–based trend states, and both Logistic Regression and XGBoost are capable of learning these relationships effectively. The similarity in accuracy and ROC-AUC across models suggests that the indicator–label relationship is nearly linear, explaining why Logistic Regression performs comparably to XGBoost.

Limitations include delayed response to sharp market reversals, reliance on historical price–derived labels, and reduced performance during strong momentum surges. The use of daily rather than intraday data likely smooths some local patterns that may have been captured in the original study.

Conclusion

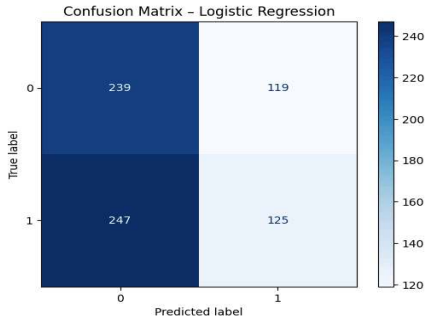
This project successfully reproduces the methodology and reported performance of Hafid et al. using daily BTC-USD data. The results demonstrate consistent classification accuracy, meaningful indicator importance patterns, and economically interpretable trend signals. Future extensions include experimenting with forward-return labels, walk-forward validation, and adapting the system to higher-frequency data consistent with the original study's context Through both comparison models, we were able to outperform the novel model with more traditional approaches..

Appendix

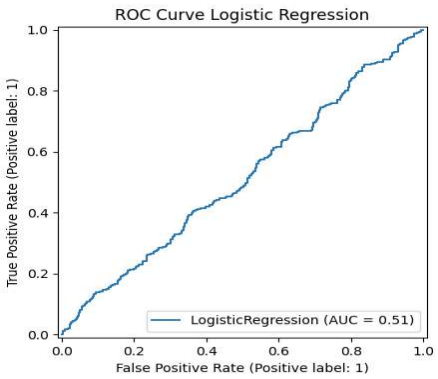
Comparison Analysis 1

Confusion Matrix: Logistic Regression

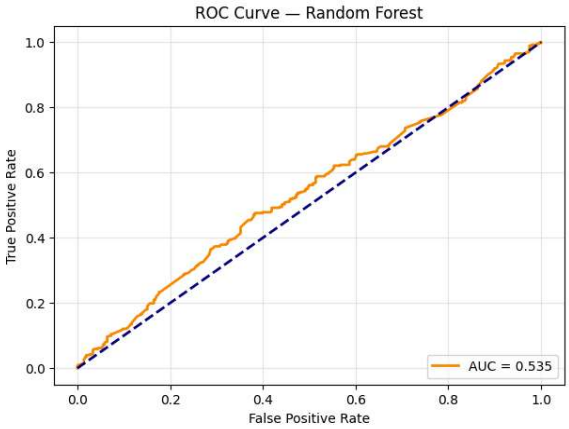
Displays correct vs incorrect classifications, showing how often the model mislabels UP and DOWN days.



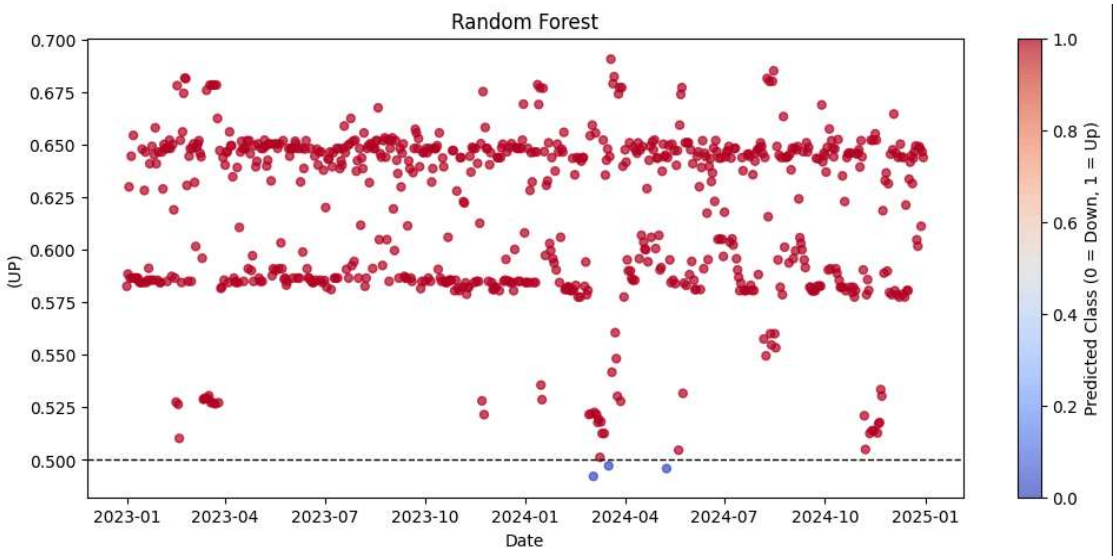
ROC Curve Logistic regression



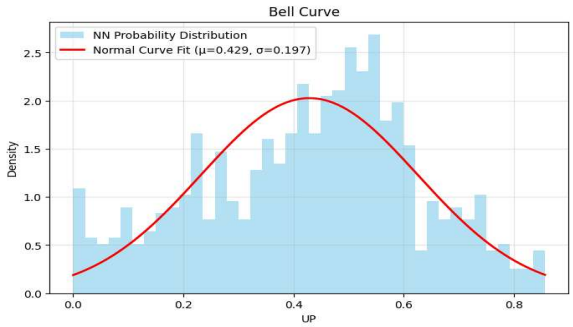
Random Forest:



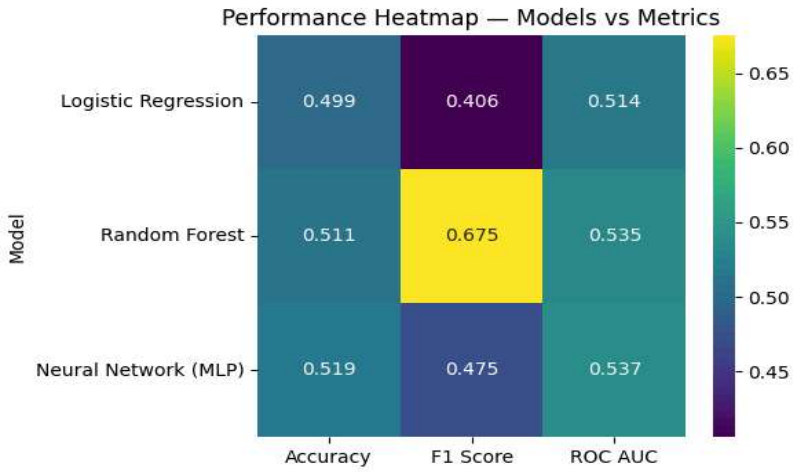
Random Forest Predicted Probabilities Over Time: the model's confidence changes across market regimes.



Distribution of Neural Network Predicted Probabilities: The Neural Network is used to predict whether Bitcoin will go UP the next day.

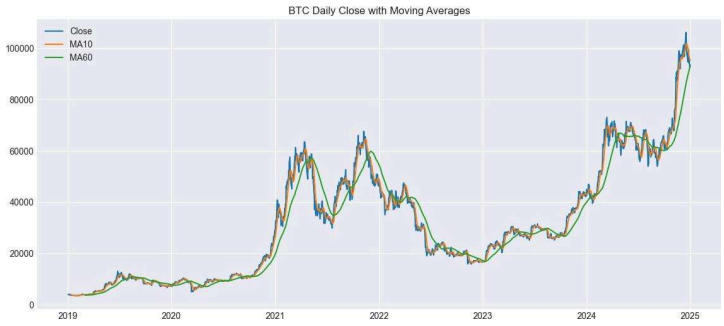


Model Comparison: The heatmap shows that all models are slightly above random guessing, which aligns with the noisy nature of short-term crypto movement prediction**.**

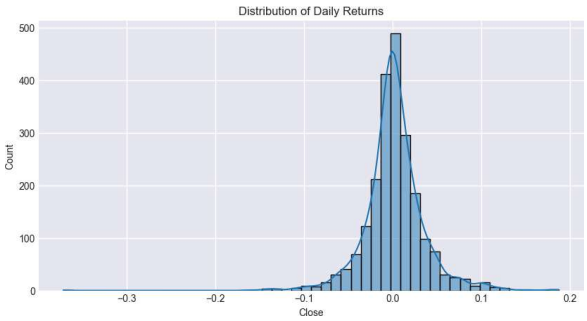


Comparison Analysis 2

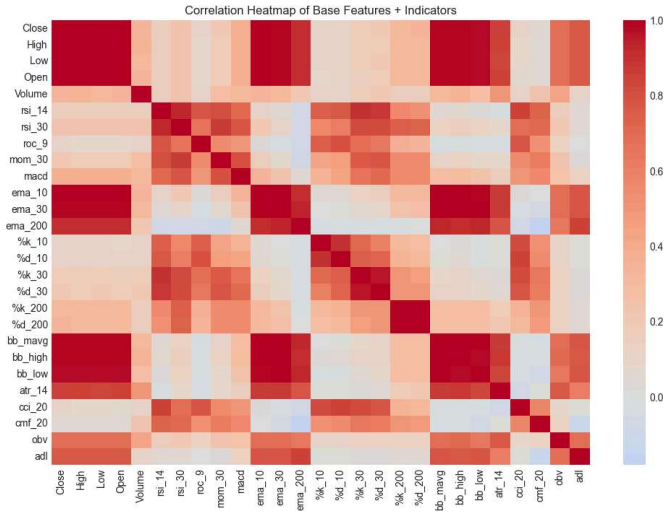
BTC Closing Price with Moving Averages



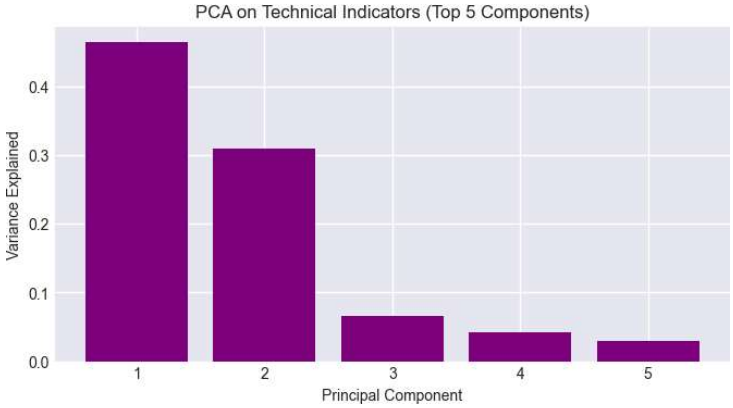
Distribution of Daily Returns



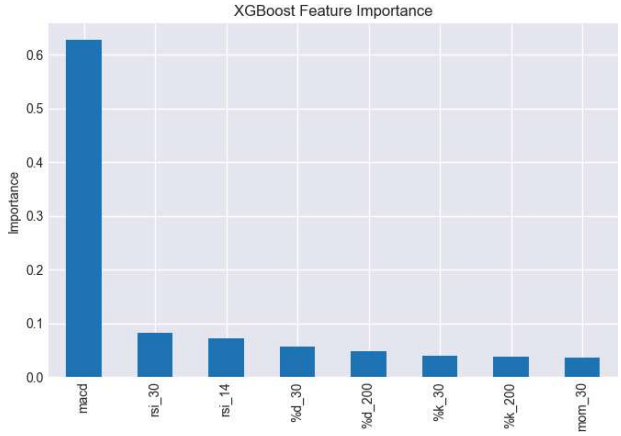
Correlation Heatmap of Technical Indicators



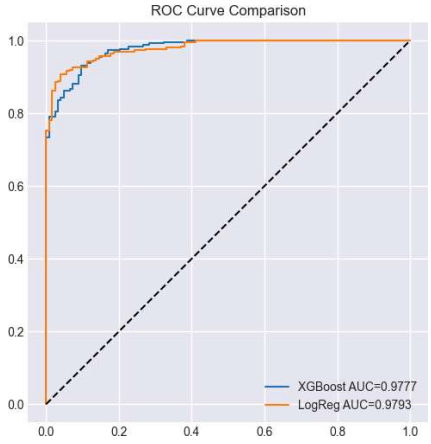
PCA Variance Explained



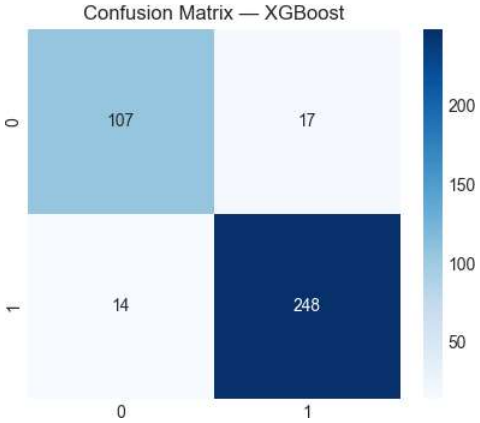
XGBoost Feature Importance Ranking



ROC Curve Comparison (XGBoost vs Logistic Regression)



Confusion Matrix for XGBoost Predictions



Backtest: Model Strategy vs Buy & Hold