# MSDS 451 Programming Assignment 1
# Bitcoin Data Analysis and Trading

Abhijit Joshi
MSDS-451-DL: Financial Engineering
Northwestern University
Dr. Thomas Miller
October 5, 2025

## 1. Problem Description

The MSDS-451 Programming Assignment-1 aims to predict the direction of Bitcoin (BTC-USD) spot prices (up or down) using machine learning classifiers, specifically tree-based ensemble boosting methods, such as XGBoost. The prediction is based on various lagged price features, including daily closing prices (lagged 1 to 7 days), opening, closing, high, and low price points, as well as daily trading volume.

The project also implements and evaluates a trading strategy based on the predicted direction of Bitcoin spot prices. The strategy involves buying a fixed percentage of bitcoin when the target price is predicted to go up and selling 5% of holdings when a prices go down and continue to stay down for certain timed. The performance of this strategy is compared to that of a simple buy-and-hold approach. The goal is to evaluate the efficacy of using machine learning methods to predict bitcoin prices and utilize the predicted prices to generate trading signals for a .

*Key Words*: *XGBoost, trading, indicators, moving averages, price, volatility, momentum, strategy*

## 2. Data Preparation and Pipeline

### 2.1. Data Loading

Historical Bitcoin (BTC-USD) daily data from September 1, 2014, to October 3, 2025, was downloaded using the `yfinance` library. The data include Open, High, Low, Close, Volume, Dividends, and Stock Splits. This data was then saved to a CSV file named "btc_historical_data.csv".

### 2.2. Polars for Data Manipulation

The saved CSV file was read into a Polars DataFrame. Irrelevant columns ('Dividends', 'Stock Splits') were dropped.

### 2.3. Feature Engineering

Several new features were created using Polars:

#### 2.3.1. Lagged Price Features

Lagged closing prices for one, two, and three days ('CloseLag1', 'CloseLag2', 'CloseLag3').

#### 2.3.2. High-Minus-Low (HML)

The difference between the daily high and low price ('HML') and its lags ('HMLLag1', 'HMLLag2', 'HMLLag3').

#### 2.3.3. Open-Minus-Close (OMC)

The difference between the daily open and closing price ('OMC') and its lags ('OMCLag1', 'OMCLag2', 'OMCLag3').

#### 2.3.4. Volume Lags

Lagged volume for one, two, and three days ('VolumeLag1', 'VolumeLag2', 'VolumeLag3').

#### 2.3.5. Exponential Moving Averages (EMAs)

Exponential moving averages of the lagged closing price ('CloseLag1') with half-lives of 1, 2, and 4, named 'CloseEMA2', 'CloseEMA4', and 'CloseEMA8' respectively.

#### 2.3.6. Relative Strength Index (RSI)

RSI was calculated for 'CloseLag1', 'CloseLag2', and 'CloseLag3' using a 14-period span, resulting in 'RSI_CloseLag1', 'RSI_CloseLag2', and 'RSI_CloseLag3'. Intermediate columns created during the RSI calculation were dropped.

**2.4. Data Type Conversion and Rounding**
Volume features were cast to Float64 and rounded to zero decimal places. Other price and engineered features were rounded to three decimal places for consistency and reporting.
**2.5. Target Variable Creation**
A binary target variable ('Target') was created. It is set to 1 if the 'LogReturn' (calculated as the natural logarithm of the ratio of the current day's closing price to the previous day's closing price) is greater than 0, and 0 otherwise.
**2.6. Data Cleaning**
Rows with null values, introduced by the lagging operations and potentially the EMA/RSI calculations, were dropped using `btc.drop_nulls()`. An additional filter `btc.filter(~pl.col("RSI_CloseLag3").is_nan())` was applied to explicitly handle potential NaNs in the RSI calculation.
**2.7. Feature Selection using AIC**
Based on a preceding analysis using All Possible Classifications and the Akaike Information Criterion (AIC), a subset of six features was selected for model development. The selected features were 'CloseLag3', 'HMLLag3', 'OMCLag3', 'VolumeLag3', 'CloseEMA8', and 'RSI_CloseLag2'.
Note** - This AIC method was not used as it required 45-90 mins during training. The selected features were hand-selected based on checking correlation between features.
**2.8. Feature Standardization**
The selected subset of features was converted to a NumPy array and standardized using `StandardScaler` from `sklearn.preprocessing`. This scales the features to have zero mean and unit variance, which is often beneficial for machine learning models.
**3. Research Design**
The research design focuses on developing and evaluating a machine learning model to predict the direction of Bitcoin price movements. Given the time-dependent nature of financial data, a standard k-fold cross-validation approach, which assumes independent observations, is inappropriate. Therefore, a time series cross-validation approach using Scikit-Learn's `TimeSeriesSplit` was employed.
**3.1. Time Series Cross-Validation (`TimeSeriesSplit`)**
`TimeSeriesSplit` divides the time series data into multiple training and test sets. In each split, the training set consists of observations up to a certain point in time, and the test set consists of subsequent observations. Crucially, successive training sets are supersets of previous ones, maintaining the temporal order of the data. The `gap` parameter was set to 10. This excludes 10 samples from the end of each training set and before the start of the next test set. This gap helps to prevent data leakage from the training set into the test set, as future information (even if slightly in the future) should not be used to predict the past. A total of 5 splits (`n_splits=5`) were used, creating five pairs of training and test sets.
**3.2. Initial Model Evaluation**
An initial classification model (XGBoost Classifier with default hyperparameters except `n_estimators=1000`) was evaluated within the `TimeSeriesSplit` cross-validation framework using Scikit-Learn's `cross_validate`. The primary scoring metric used for this initial evaluation was 'accuracy'. This provides a baseline understanding of the model's performance on unseen data across different time periods.
**3.3. Hyperparameter Tuning using Randomized Search**

To find better hyperparameter settings for the XGBoost Classifier, `RandomizedSearchCV` was utilized. This method randomly samples a fixed number of hyperparameter combinations from specified distributions, which is more computationally efficient than an exhaustive grid search, especially with a large hyperparameter space.
The following hyperparameters were tuned:

1. max_depth`: Sampled from a uniform distribution between 3 and 10.
2. `min_child_weight` Sampled from a uniform distribution between 1 and 10.
3. `subsample`: Sampled from a uniform distribution between 0.5 and 1.0.
4. `learning_rate`: Sampled from a uniform distribution between 0.01 and 0.1.
5. `n_estimators`: Sampled from a uniform distribution between 100 and 1000.
6. `RandomizedSearchCV` also used the same `TimeSeriesSplit` cross-validation strategy (`cv = TimeSeriesSplit(gap=10, n_splits=5)`) to ensure that the hyperparameter tuning process respected the temporal nature of the data.
7. The search was conducted for 100 iterations (`n_iter=100`) and used 'accuracy' as the scoring metric to determine the best parameter combination.
8. The best hyperparameters found through this process were then used to define the final XGBoost classification model.

**4. Programming**

A starter notebook wss provided, which served as a jump start for developing this project. The following important Python libraries were used -

1. yfinance: This library was used for initial data acquisition, specifically to download historical daily price and volume data for Bitcoin (BTC-USD) from Yahoo Finance.
2. Polars: Polars, a high-performance DataFrame library, was extensively used for data manipulation and feature engineering.
3. Scikit-learn: Scikit-learn provided essential tools for the machine learning workflow, including:
    a. Preprocessing: `StandardScaler` was used to standardize the selected features, ensuring they have zero mean and unit variance, which is beneficial for many models.
    b. Cross-Validation: `TimeSeriesSplit` was implemented to create appropriate training and test sets for the time-series data, respecting the temporal order and preventing data leakage with a defined `gap`.
    c. Model Evaluation: Functions and classes such as `cross_validate`, `accuracy_score`, `classification_report`, `roc_curve`, `roc_auc_score`, `RocCurveDisplay`, `ConfusionMatrixDisplay`, and `confusion_matrix` were utilized to assess the performance of the classification models.
    d. ** Hyperparameter Tuning:** `RandomizedSearchCV` was employed to efficiently search for optimal hyperparameters within the XGBoost model using the time series cross-validation framework.
4. XGBoost: The core classification model used is the XGBoost Classifier (`XGBClassifier`) from the XGBoost library. XGBoost is a powerful gradient boosting framework known for its performance. It was used to build the predictive model for the direction of Bitcoin price movements.
5. Matplotlib and Seaborn: These libraries were used for data visualization, specifically to plot the ROC curve and the confusion matrix, which visually represent the model's

performance. Matplotlib was also used in the backtesting visualization to plot the closing price along with the buy and sell signals.

4.1. Code Implementations

1. **Feature Selection (`getAIC` and `powerset`):** A custom function `getAIC` was implemented to calculate the Akaike Information Criterion (AIC) for a given set of features using Logistic Regression. This function helps evaluate the trade-off between the goodness of fit and the complexity of a model. The `powerset` function from `itertools` was used to generate all possible combinations of features to find the subset with the lowest AIC. However, the execution of the full powerset was interrupted. Based on a review of the top AIC models, a subset of features was manually selected.
2. **Cross-Validation and Evaluation (`evaluate`):** The `TimeSeriesSplit` object was initialized to define the cross-validation strategy. A custom `evaluate` function was created to perform cross-validation using `cross_validate` and report evaluation metrics, such as mean accuracy and its standard deviation across the folds. This function allowed for a systematic assessment of the model's performance on different time periods.
3. **Hyperparameter Tuning (`RandomizedSearchCV`):** The `RandomizedSearchCV` object was configured with the XGBoost Classifier, the hyperparameter distributions to sample from, the number of iterations, and the `TimeSeriesSplit` cross-validation strategy. The `fit` method was called on the training data to find the best hyperparameter combination based on the specified scoring metric ('accuracy').
4. **Final Model Training and Evaluation:** An XGBoost Classifier (`finalModel`) was instantiated with the hyperparameters identified from the randomized search (or manually chosen based on the search results). This model was then trained on the full standardized dataset (`X`, `y`). After training, predictions (`ypred`) were generated on the training data itself to evaluate the model's fit and performance. Evaluation metrics including the confusion matrix, classification report (precision, recall, f1-score), and the ROC curve were generated and displayed using functions from `sklearn.metrics` and `matplotlib`.
5. **Backtesting Strategies (`backtest_strategy` functions):** Two separate `backtest_strategy` functions were implemented to simulate trading strategies based on the model's predicted 'Target' variable.
    a. The first function simulated a simple buy strategy where a fixed `buy_amount` (or 5% of current capital, whichever is greater) is invested when the `Target` changes from 0 to 1.
    b. The second function implemented a momentum-based strategy with specific buy and sell rules: sell 5% of holdings if `Target` is 0 for three consecutive days, and buy 10% of capital if `Target` is 1 following three straight days where `Target` was 0.
    c. Both functions tracked the capital and Bitcoin holdings over time and calculated the final portfolio value, which was then compared to a simple buy-and-hold strategy.

**5. Exposition**

**5.1. Key Findings**

1. Model Evaluation: The final XGBoost classification model, trained on the selected and standardized features, achieved a high overall accuracy on the full dataset. The confusion matrix shows that out of 1905 negative return days, the model correctly predicted 1683

(True Negatives), and out of 2125 positive return days, it correctly predicted 1972 (True Positives). The misclassifications include 222 False Positives and 153 False Negatives.
2. The classification report provides a detailed breakdown of precision, recall, and f1-score for each class. For the '0' class (Negative Return), the precision is 0.92, recall is 0.88, and f1-score is 0.90. For the '1' class (Positive Return), the precision is 0.90, recall is 0.93, and f1-score is 0.91. The overall micro and macro averages for precision, recall, and f1-score are all around 0.91, indicating strong performance on the training data.
3. Backtesting Strategy 1 (Simple Buy): The simple buy strategy, where a fixed amount (or 5% of capital) is invested on each buy signal, resulted in a final portfolio value of $4,108,172.17. This significantly outperformed the simple buy-and-hold strategy, which yielded $3,025,950.44 over the same period. This suggests that even a simple strategy based on the model's buy signals could enhance returns compared to a passive buy-and-hold, especially in a strongly trending market like Bitcoin's historical performance.
4. Backtesting Strategy 2 (Momentum-Based Buy/Sell): The momentum-based systematic buy-and-sell strategy, which involves selling a portion of holdings after consecutive predicted negative returns and buying after a predicted shift to positive momentum, yielded a final portfolio value of $1,732,348.87. This strategy performed worse than both the simple buy strategy and the simple buy and hold strategy over the evaluated period. This indicates that the specific sell rule (selling 5% after three consecutive predicted down days) and buy rule (buying 10% after a specific momentum shift) in this strategy were not as effective in capturing the overall upward trend of Bitcoin as the simpler strategies.

**5.2. Insights**

1. The high evaluation metrics (accuracy, precision, recall, f1-score) on the full dataset suggest that the XGBoost model with the selected features is highly capable of classifying the direction of Bitcoin price movements. However, it's important to note that these metrics are based on the full dataset, including the data used for training and hyperparameter tuning. The cross-validation results with an average accuracy of around 0.508 +/- 0.013 suggest that the model's performance on unseen data during the cross-validation folds was significantly lower than the performance on the full dataset. This discrepancy highlights a potential risk of overfitting the model to the training data, especially after further tuning and increasing estimators as mentioned in the summary.
2. For Bitcoin's historical performance, a simple buy and hold strategy proved to be highly profitable due to the significant upward trend. The simple buy strategy based on the model's signals further amplified these gains, likely by timing entries to capture upward movements. This aligns with the insight that buy and hold works well in a strong bull market.
3. The momentum-based buy/sell strategy's underperformance compared to the simple buy strategies suggests that the chosen selling rule might have caused the strategy to exit positions prematurely in a strongly trending market, missing out on significant upward price movements. The specific buy signal condition also might not have been frequent or well-timed enough to fully capitalize on the available opportunities.

**5.3. Next Steps**

1. Refine Trading Strategies:

    a. Optimize Strategy Parameters: The percentages for buying (10%) and selling (5%) in the momentum-based strategy could be optimized through backtesting over different periods or using techniques like walk-forward optimization.
    b. Alternative Sell Rules: Explore different sell rules, perhaps based on profit targets, stop-loss levels, or different combinations of predicted negative returns, to avoid premature exits in trending markets.
    c. Incorporate Transaction Costs: A more realistic backtest should include transaction costs (e.g., trading fees) and potential slippage, which can significantly impact profitability.
    d. Position Sizing: Implement more sophisticated position sizing techniques rather than fixed percentages of capital or holdings.
2. Explore Additional Features: Investigate the predictive power of other potential features, such as sentiment analysis from news or social media, on-chain data (e.g., transaction volume, active addresses), or macroeconomic indicators.
3. Alternative Models: Experiment with other classification models or ensemble techniques to see if they offer improved performance or different trade-offs.
4. Regression for Return Magnitude: While this project focused on the direction, predicting the magnitude of returns using regression models could inform strategies that size positions based on the expected return.
5. Consider Drawdown and Risk Metrics: Evaluate the trading strategies not just on final capital but also on risk metrics like maximum drawdown, Sharpe ratio, or Sortino ratio, which provide a more complete picture of the strategy's performance and risk profile.