# Detecting Flash Crash Precursors in Bitcoin Market Using Supervised and Unsupervised Learning

Computational Data Analysis, Spring 2025

Francis Lin, Jennifer Guo, Tim Phan

April 2025

## 1 Abstract

The flash crash of 2010 saw major market indices, such as the S&P 500, Dow Jones, and Nasdaq Composite, collapse by over 9% within 15 minutes, and then partially recover to 7% within the hour. The cause of this micro-crash was researched and debated for years, but we are interested in any irregular market behaviors that occurred right before the crash. [1]

Market crashes are often preceded by unusual trading activity, such as sharp spikes in volatility, volume, and price movements. Smaller market crashes, also known as "micro-crashes", are often the result of multiple factors, which can include algorithm trading, a high volume of sells, and others which are usually identified retrospectively. [3] Compared to a traditional market, bitcoin tends to experience far more extreme price swings and returns, which may indicate that crypto market crashes are driven by a different set of factors. This project aims to use machine learning-based anomaly detection to determine whether traditional indications of a micro-crash are applicable to bitcoin as well.

## 2 Data Processing

To construct a high-quality dataset for modeling market microstructure dynamics, we collected high-frequency orderbook data from two major cryptocurrency exchanges: OKX and Bybit. Both platforms provide real-time orderbook updates through public WebSocket APIs. We focused on the BTC/USDT trading pair and recorded the top 50 bids and asks for each incoming update.

### 2.1 Data Collection

We implemented an asynchronous data collector using Python's `websockets` and `asyncio` libraries. The collector subscribed to the `books5` channel on OKX and the Level-2 orderbook channel on Bybit, both of which stream incremental updates of the limit orderbook. Each message was timestamped locally upon receipt and written to disk in newline-delimited JSON format. The raw data stream included orderbook changes (deltas), heartbeat events, and subscription acknowledgements.

Table 1: Sample Raw Orderbook Data (Before Reconstruction)

| Timestamp | Side | Price | Size | # of Orders |
|---|---|---|---|---|
| 1744729672.889366 | bid | 85188.3 | 0.66892663 | 11 |
| 1744729672.889366 | bid | 85188.2 | 0.00001 | 1 |
| 1744729672.889366 | bid | 85188.0 | 0.0229 | 1 |
| 1744729672.889366 | bid | 85184.4 | 0.03363067 | 1 |
| 1744729672.889366 | bid | 85184.0 | 0.0229 | 1 |
| 1744729672.889366 | bid | 85182.5 | 0.08682071 | 1 |
| 1744729672.889366 | bid | 85182.4 | 0.1935 | 1 |
| 1744729672.889366 | bid | 85182.2 | 0.00204 | 1 |

## 2.2 Orderbook Reconstruction

Since the raw data consisted of incremental changes, we reconstructed the full orderbook at each point in time by maintaining an in-memory state of the top-level bids and asks. Upon receiving each delta, the local orderbook was updated accordingly. Each update also triggered the generation of a snapshot, representing the complete state of the orderbook at that timestamp.

To ensure consistency and avoid drift, we verified the reconstructed book against periodic full snapshots when available. The final output was a list of orderbook snapshots, each containing timestamped bid and ask ladders.

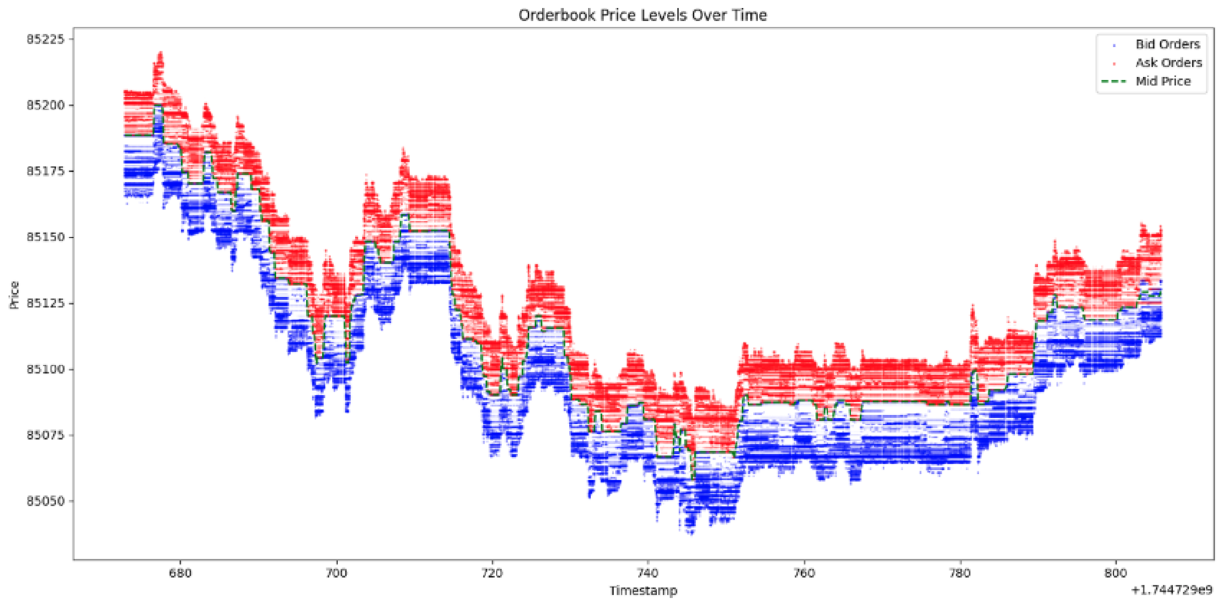

Figure 1: Orderbook Price Levels Over Time

## 2.3 Compression

Given the high frequency of updates and the depth of the orderbook, the reconstructed dataset was extremely large. To address this, we applied two levels of compression:

- **Snapshot Sampling:** Instead of recording every update, we retained snapshots at regular time intervals (e.g., every 1 minutes) to reduce redundancy while preserving meaningful dynamics.

- **Orderbook Compression:** We further transformed the full orderbook into a compact representation by extracting key features such as the top-$k$ price levels, volume imbalance, spread, cumulative volume, and slope. This step allowed us to significantly reduce dimensionality without losing crucial market signals.

The final dataset included compressed orderbook features with aligned timestamps and associated mid-price information, ready for labeling and modeling.

Table 2: Sample Compressed Orderbook Snapshot

| **Timestamp** | **Bid Volume** | **Ask Volume** | **Best Bid** | **Best Ask** | **Mid Price** |
|---|---|---|---|---|---|
| 2025-03-10 00:00:00 | 1.88e9 | 1.90e9 | 80807.5 | 80623.0 | 80715.25 |
| 2025-03-10 00:01:00 | 1.88e9 | 1.93e9 | 80866.5 | 80738.5 | 80802.5 |
| 2025-03-10 00:02:00 | 2.09e9 | 2.10e9 | 80836.5 | 80686.5 | 80761.5 |
| 2025-03-10 00:03:00 | 1.94e9 | 1.87e9 | 81091.0 | 80763.5 | 80927.25 |
| 2025-03-10 00:04:00 | 2.07e9 | 2.04e9 | 81257.5 | 81024.5 | 81141.0 |
| 2025-03-10 00:05:00 | 2.21e9 | 2.20e9 | 81232.0 | 80933.0 | 81082.5 |

## 3 Methodology

### 3.1 Features

To extract informative features from the raw order book data, we processed the lists of bids and asks at each timestamp. In the original data, the `bids` and `asks` columns contain lists of order tuples $[price, quantity]$, sorted by price—descending for bids and ascending for asks.

The features we extracted aim to capture the instantaneous state of the order book and its potential implications for market dynamics. Specifically, we considered the following types of features:

- **Best Bid and Offer (BBO)**: These represent the highest price a buyer is willing to pay (from the first element of the `bids` list) and the lowest price a seller is willing to accept (from the first element of the `asks` list). They indicate the most competitive prices for immediate execution. We extracted the best bid price $P_{bid}^{best}$ and the best ask price $P_{ask}^{best}$.

- **Bid-Ask Spread**: This is the difference between the best ask price and the best bid price, calculated as $S = P_{ask}^{best} - P_{bid}^{best}$. The bid-ask spread is a widely used measure of market liquidity; a narrower spread typically suggests higher liquidity and lower transaction costs.

- **Best Bid and Ask Size**: These are the quantities of orders available at the best bid and best ask prices, respectively. We extracted the best bid size $Q_{bid}^{best}$ and the best ask size $Q_{ask}^{best}$. These quantities reflect the immediate buying and selling interest at the most competitive prices.

- **Top N Bid and Ask Prices**: Beyond the best prices, we also extracted the prices of the top $N$ levels of bids and asks. Specifically, we used the top 5 bid prices $P_{bid}^1, P_{bid}^2, ..., P_{bid}^5$ and the top 5 ask prices $P_{ask}^1, P_{ask}^2, ..., P_{ask}^5$. These price levels capture the shape and depth of the order book. As demonstrated by Tran et al. [1], while the best level contains most of the predictive information, including additional layers—particularly up to the fourth level—can yield a 2-3% improvement in forecasting performance, suggesting that deeper levels provide complementary insights.

These extracted features transform the raw order book data into a numerical format suitable for machine learning models to learn and predict the `label` column. By capturing the best quotes, the spread, and the structure of the top levels of the order book, we aim to provide sufficient information for the models to forecast future market movements or the associated labels.

## 3.2 Model Selection

To evaluate the effectiveness of our features extracted from the reconstructed and compressed orderbook, we compared the performance of several classification models using standard metrics: precision, recall, F1-score, and accuracy. Below are the results for each model tested.

Table 3: Classification Report Summary for All Models in Tick Scale

| Model | Precision | Recall | F1-Score | Accuracy |
|---|---|---|---|---|
| **Logistic Regression** | 0.10 | 0.32 | 0.15 | 0.32 |
| **Random Forest** | 0.90 | 0.90 | 0.90 | 0.90 |
| **Gradient Boosting** | 0.84 | 0.84 | 0.84 | 0.84 |
| **AdaBoost** | 0.66 | 0.65 | 0.65 | 0.65 |

From the table above, we observe that:

- **Random Forest** achieved the best overall performance across all metrics, with 90% accuracy and consistent high precision/recall values.
- **Gradient Boosting** also performed well, with slightly lower but still strong scores across the board.
- **AdaBoost** offered decent performance, but with a slight tradeoff between classes.
- **Logistic Regression** underperformed significantly in this multiclass scenario, capturing only the dominant class (1.0) and ignoring the others.

Based on these results, we selected the **Random Forest Classifier** as our final model due to its superior accuracy and balanced classification performance.

To better understand how the model makes predictions, we visualized the decision rules of the Random Forest model using one of its decision trees. Figure 2 shows the structure of the tree and the feature thresholds used at each split.

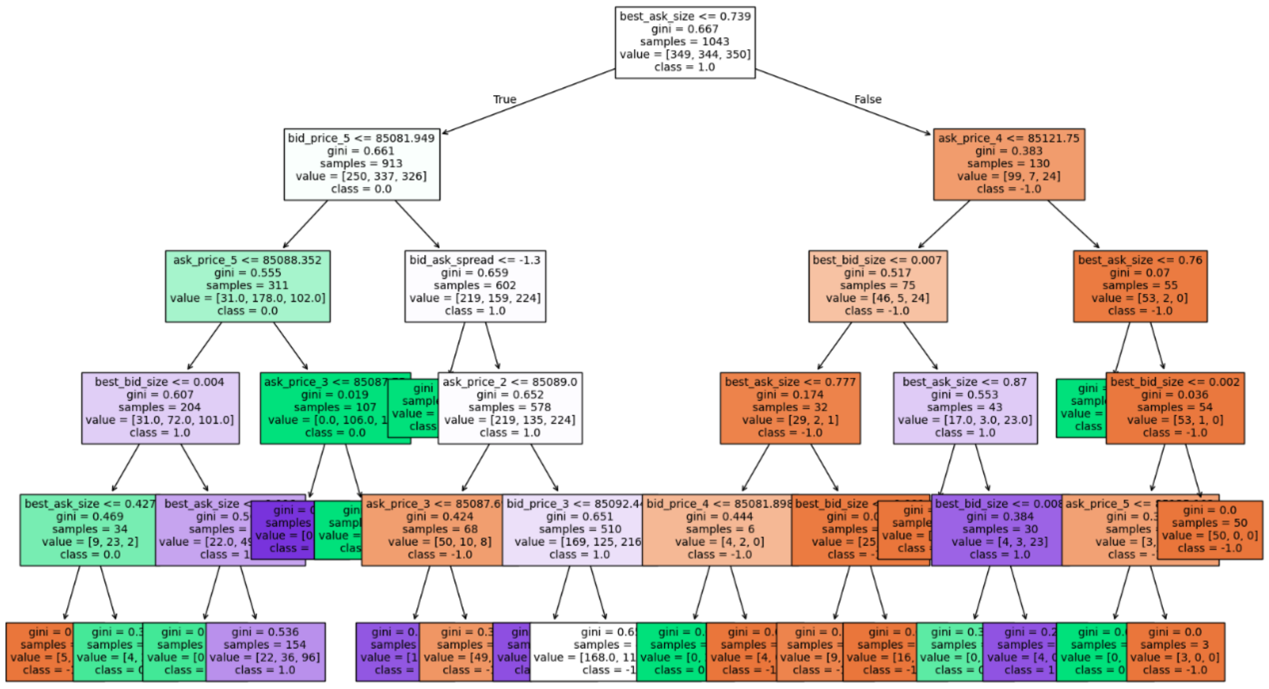

Figure 2: Decision Tree from Random Forest Classifier (1 of 100 trees). Node color reflects majority class.

This visualization allows us to interpret which features contribute the most to the model's decision making. For instance, the root split is based on `best_ask_size`, indicating that large ask liquidity tends to signal an upward or neutral move in price. Other informative splits include differences between bid/ask prices and size imbalances, such as `bid_price_5` and `bid_ask_spread`.

This insight confirms the value of features extracted from the orderbook structure and helps us validate the trading intuition behind price prediction based on microstructure signals.

## 4 Models and Results on Final Minute Data

Following the success of the Random Forest Classifier on the one hour of tick data, the model was used as a basis to create the model to be utilized on one week of extracted orderbook data compressed and aggregated by minute.

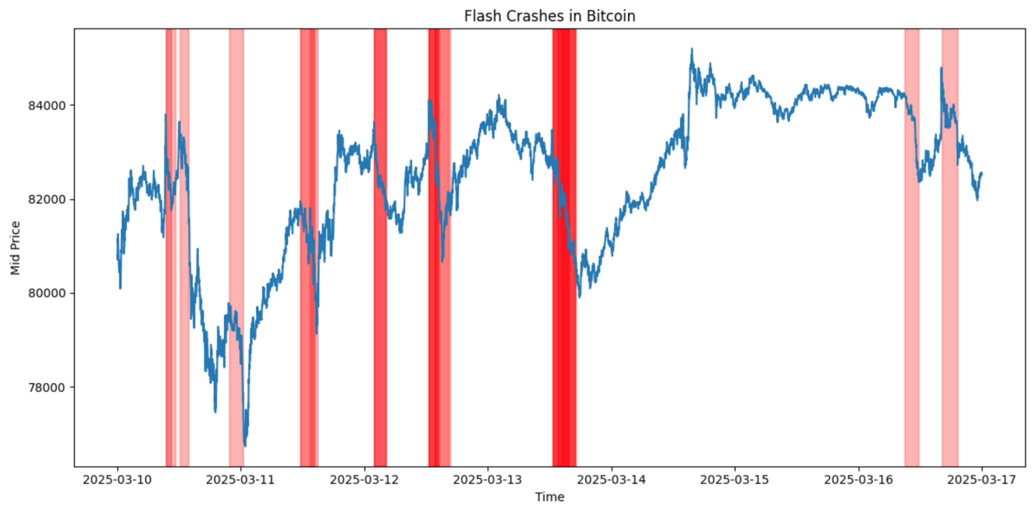

Figure 3: Flash Crashes Highlighted on One Week of Minute Data

## 4.1 Temporal Crash Window Random Forest Predictor

The Random Forest classifier was implemented using sklearn.ensemble.RandomForestClassifier with a focus on predicting whether a given 3-hour window was a pre-crash period. The model used 100 trees and class weighting to account for the heavy imbalance between normal and pre-crash windows. A chronological train-test split was applied, reserving the final 20% of the dataset for evaluation. Input features were derived from minute-level order book data and aggregated into 3-hour blocks. These features included bid-ask spread, order-flow imbalance, the number of ask orders, and several lagged variables—such as the number of ask orders and the imbalance at 1, 5, and 15 minutes prior. We also included rolling volatility of the imbalance over a 5-minute window to capture localized variance patterns that often precede market stress.

The model aimed to capture the gradual buildup of market pressure that might not be evident in any singular moment, but becomes clearer when observed over time. By combining static and temporal features, the Random Forest was able to learn patterns that are indicative of deteriorating liquidity or imbalance buildup. This approach, while supervised, allowed us to directly associate feature patterns with crash-labeled windows and served as a strong baseline for comparing with unsupervised techniques.

## 4.2 Autoencoder

The autoencoder was implemented using sklearn MLPRegressor as an unsupervised anomaly detection tool. It was trained to reconstruct the same 3-hour aggregated features used by the Random Forest model. The architecture included a single hidden layer of size 5, followed by a linear output layer that matched the original input size. This design created a bottleneck representation that forced the network to compress and learn the most important patterns of normal market behavior. We used ReLU activation functions, an alpha (L2 regularization term) of 1e-5, a learning rate of 0.01, and a batch size of 128. Early stopping was enabled to prevent overfitting, and the training data was normalized using a StandardScaler to stabilize learning dynamics.

After training on non-crash data, the autoencoder was evaluated on the test set by calculating the mean squared reconstruction error for each sample. A threshold for anomaly detection was set at the 99th percentile of training MSE values. Any test window with an error above this threshold was flagged as an anomaly—i.e., a potential pre-crash window. This method provided a completely label-free way to surface unusual patterns, making it particularly useful for detecting flash crash scenarios that don't follow previously observed paths. The autoencoder ultimately served as a valuable comparison to the Random Forest model, offering a different lens through which to view market instability.

## 4.3 Results

The results summarized in Table 4 compare the performance of the autoencoder and Random Forest models at the minute scale. Both models achieved the same F1-score of 0.74, indicating a balanced trade-off between precision and recall. The autoencoder demonstrated slightly higher precision (0.71 vs. 0.67), suggesting it was better at reducing false positives—i.e., it was more conservative in flagging market anomalies. On the other hand, the Random Forest model achieved a marginally higher recall (0.82 vs. 0.80), meaning it was slightly better at identifying all true instances of pre-crash conditions, though at the cost of more false positives.

In terms of overall accuracy, the Random Forest slightly outperformed the autoencoder (0.82 vs. 0.80), though the difference is minor. These results indicate that while both models are comparable in performance, they bring different strengths to the task: the autoencoder excels in filtering out noise and highlighting only the most extreme deviations from normal market behavior, whereas the Random Forest model is more aggressive in detecting a broader set of potential pre-crash patterns. This comparison validates the use of unsupervised learning in market anomaly detection and suggests that ensemble or hybrid approaches may further improve early warning systems for flash crashes.

Table 4: Classification Report Summary for All Models in Minute Scale

| Model | Precision | Recall | F1-Score | Accuracy |
|---|---|---|---|---|
| **Autoencoder** | 0.71 | 0.80 | 0.74 | 0.80 |
| **Random Forest** | 0.67 | 0.82 | 0.74 | 0.82 |

## 5 Conclusions

### 5.1 Further Improvement Directions

While our project faced challenges in achieving high predictive performance, this outcome also highlights important insights into the nature of the Bitcoin market and the direction of future research. We outline several key areas for further improvements and considerations:

1. **Regulatory**
   Unlike traditional markets, Bitcoin trading operates in a largely unregulated environment. This leads to higher volatility, lack of circuit breakers, and fewer institutional participants, all of which complicate the modeling of price dynamics. Incorporating regulatory proxies (e.g., KYC exchanges vs. decentralized platforms) could offer deeper market segmentation.

2. **Difference in Market Nature**
   Traditional financial models often assume market closure, centralized liquidity, and institutional order flow. Peters [4], the author of the fractal market hypothesis, states that in traditional markets, there is a certain time period the market takes before responding to important information or events. Investor reaction to the new information is limited by the market nature, such as market trading hours, and daily upwards and downwards limits.

   This does not hold in the context of the bitcoin market. The Bitcoin market exhibits unique features such as 24/7 trading, high retail participation, and fragmented liquidity across multiple exchanges. These characteristics result in faster reaction times and noisier orderbook signals.

3. **Other Non-Model Aspects**
   Our current setup primarily focuses on orderbook-based features. However, Bitcoin prices are also heavily influenced by social sentiment (e.g., Twitter, Reddit), on-chain metrics (e.g., wallet activity, miner flows), macroeconomic signals (e.g., Fed announcements), and many more [2]. Incorporating these auxiliary signals via feature engineering or multimodal models may enhance predictive performance.

4. **Time Duration**
   Initially, our models were trained using high-frequency tick-level data, where every orderbook update was recorded. Under this setup, especially with Decision Tree classifiers, we observed promising performance, suggesting that rapid microstructure changes held predictive power for short-term price direction.

However, when we shifted to minute-level aggregated data for computational efficiency and to simulate more realistic trading strategies, the model performance significantly dropped. This suggests that valuable signals may be lost when compressing the data into longer time intervals. Future work could explore hybrid methods that retain important features of tick data (e.g., volatility bursts or volume imbalances) while maintaining computational tractability.

## Author Contributions

Guo was responsible for contextual and academic research as well as data methodology. Lin was responsible for feature engineering and modeling of the hour-of-tick data. Phan was responsible for modeling the final minute data and conducting evaluations. We met regularly to ensure project alignment and collaboration, during which we reached our conclusions.

## References

[1] A. N. Akansu. The flash crash: A review. *Journal of Capital Markets Studies*, 1(1):89–100, 2017.

[2] Zhen Ge and Caixia Zhou. Bitcoin price trends and influencing factors. In *2019 International Conference on Humanities, Management Engineering and Education Technology (HMEET 2019)*, pages 122–128, 2019.

[3] Andrei A. Kirilenko, Albert S. Kyle, Mehrdad Samadi, and Tugkan Tuzun. The flash crash: High-frequency trading in an electronic market. *The Journal of Finance*, 72(3):967–998, April 2017.

[4] Edgar E. Peters. *Fractal Market Analysis: Applying Chaos Theory to Investment & Economics*. Wiley Finance Edition. Wiley, Hoboken, NJ, 1994.