

# Crypto Price Prediction — Phase One

**Author:** Bachler Wendu

**Project:** Machine Learning for Bitcoin Price Forecasting — Phase One

**Date:** December 3, 2025

---

## Executive summary

This report summarizes Phase One of a machine-learning-driven project to forecast Bitcoin prices. The phase focused on model selection, feature engineering, and baseline evaluation using historical OHLCV (Open, High, Low, Close, Volume) data augmented with time-based and lagged features. Models evaluated include Prophet, XGBoost, and Support Vector Regression (SVR). XGBoost produced the strongest results with a percentage error of **2.189%**, while SVR delivered a weaker baseline at **~19.911%**.

## 1. Problem statement

Forecasting cryptocurrency prices — especially Bitcoin — is challenging due to high volatility, non-stationarity, and influence from exogenous events (news, regulatory changes, macroeconomic shifts, on-chain activity, and market sentiment). Accurate short-term forecasting can support trading strategies, risk management, and research; however, models must be robust to noise and offer interpretable, actionable outputs.

**Goal (Phase One):** Evaluate a range of time-series and machine learning models on historical Bitcoin data to identify a performant approach and document feature and model design choices for iterative development.

---

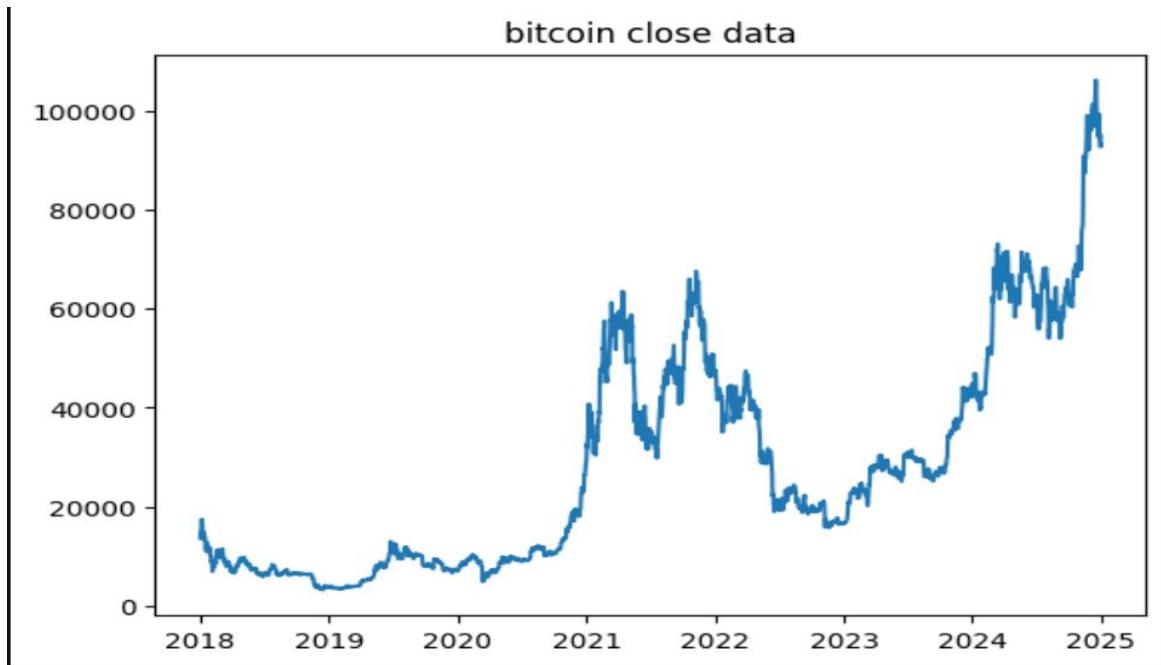
## 2. Objectives

1. Prepare and clean historical Bitcoin OHLCV data for modeling.
  2. Engineer time-based and lag features that capture short-term dependencies.
  3. Implement and tune four modeling approaches: Prophet, XGBoost, and SVR.
  4. Compare models using consistent evaluation metrics and identify the best-performing candidate.
  5. Document challenges, limitations, and a clear roadmap for Phase Two.
- 

## 3. Dataset and preprocessing

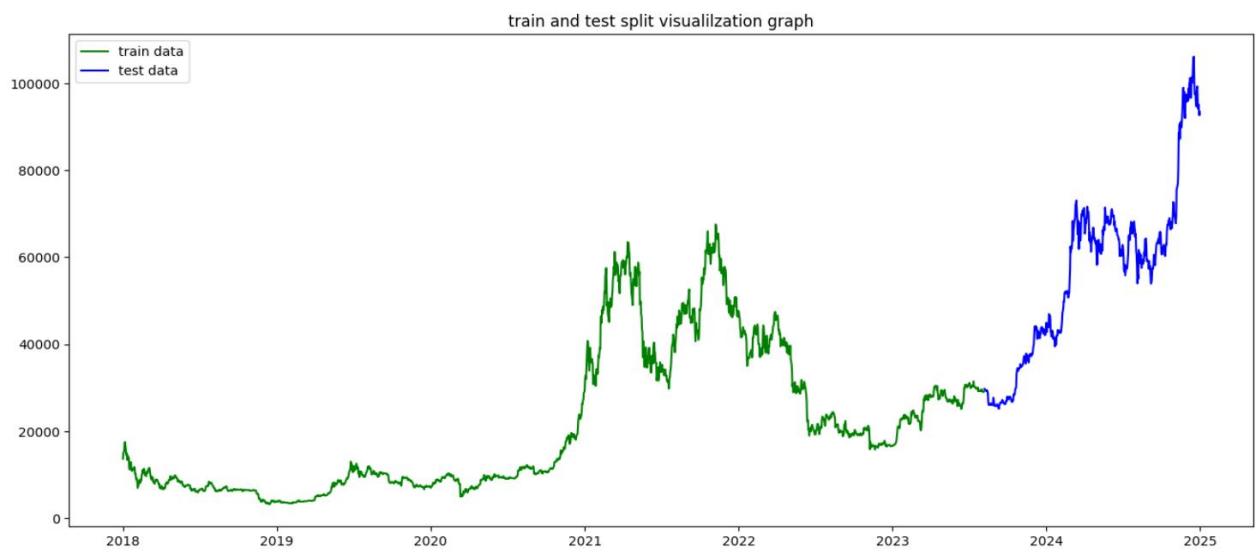
**Data source:** Yahoo Finance (BTC-USD), with the following range:  
start = 2018-01-01, end = 2025-01-01.

### Visualization placeholders

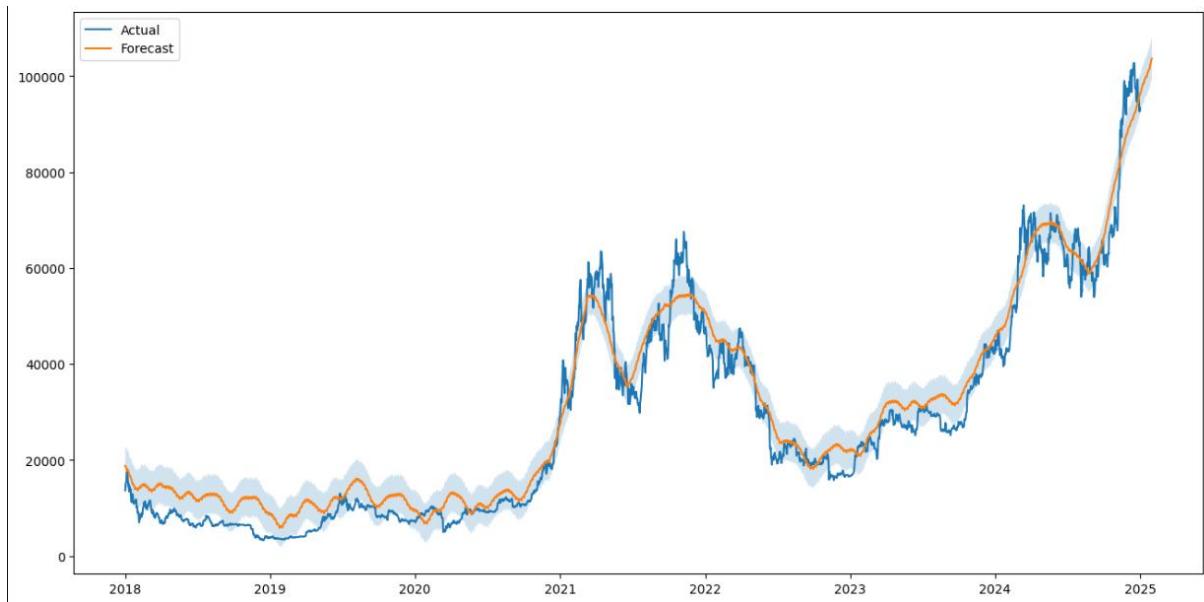


- **Figure 1:** Bitcoin Closing Price (2018–2025)

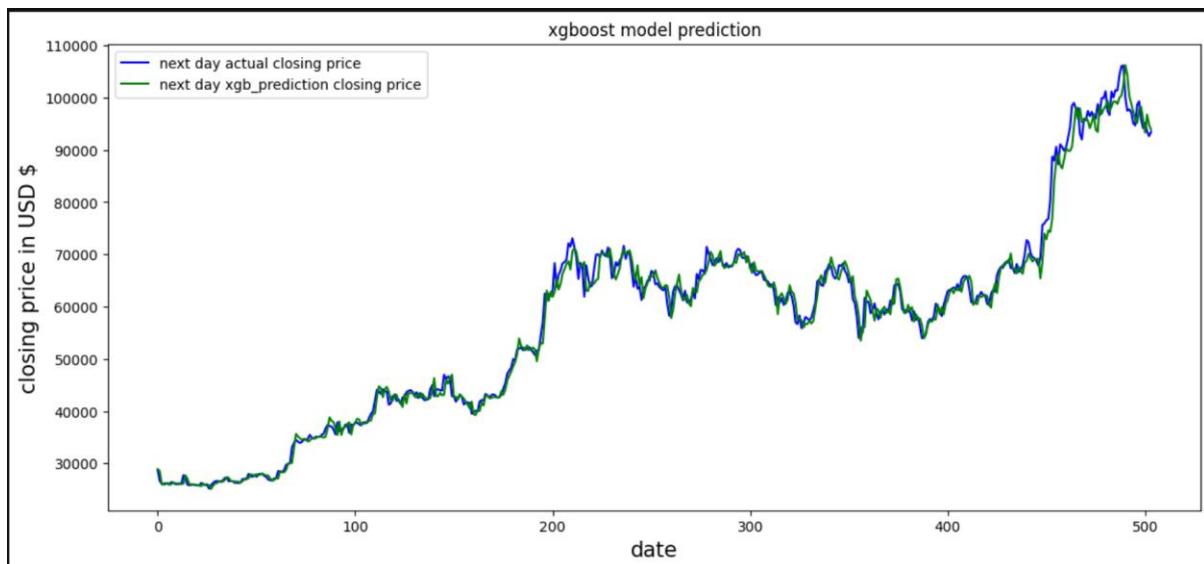
➡ Train/Test Split(80% training data,20% test data)



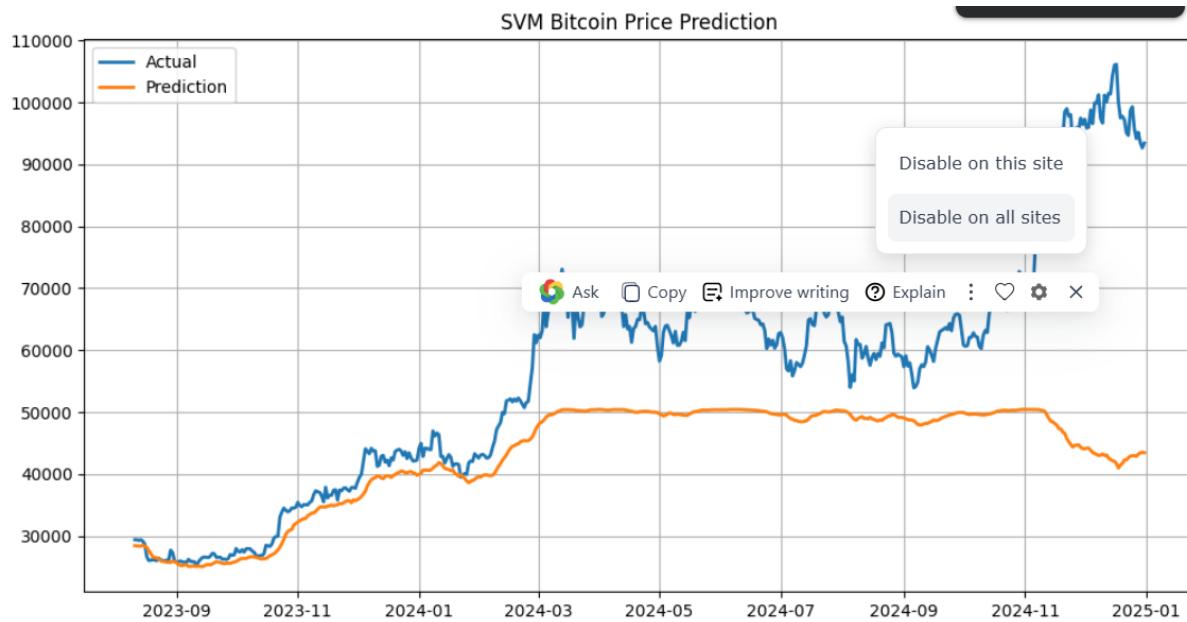
- **Figure 2:** Train/Test Split Visualization



- **Figure 3:** Prophet Predictions vs Actual



- **Figure 4:** XGBoost Predictions vs Actual



- **Figure 5:** SVM Predictions vs Actual

A comparison table has been added in the Results section.

### Data source & timeframe

Historical Bitcoin OHLCV data covering the study window (user-provided range).

### Preprocessing steps:

- Time index normalization and timezone consistency.
- Handling missing values via forward/backward fill where appropriate (small gaps) and dropping or flagging larger gaps.
- Outlier detection and optional clipping for extreme price spikes caused by data errors (not market moves).
- Generating returns and log-returns as candidate target variables.
- Scaling features when required by model type (e.g., SVR).

## 4. Feature engineering

Feature engineering is a critical component of this project. Although all models benefit from engineered features, **XGBoost particularly relies on a rich, well-designed feature set** because it learns patterns from tabular data rather than assuming statistical structures like Prophet.

Below is the comprehensive set of features used in Phase One:

### 4.1 Core market features

- **Close** – daily closing price

- **High** – intraday high
- **Low** – intraday low
- **Open** – intraday opening price
- **Volume** – traded volume

These features provide the fundamental market structure.

## 4.2 Anomaly detection features

- **z\_anom** – z-score based anomaly flag
- **iso\_anom** – anomaly score using Isolation Forest
- **is\_anomaly** – binary indicator for anomaly-like behavior

These help the model understand unusual market movements that may distort prediction.

## 4.3 Lagged return features

Lag features allow models like XGBoost to learn temporal dependencies:

- **return\_lag\_1**
- **return\_lag\_2**
- **return\_lag\_3**
- **return\_lag\_7**
- **return\_lag\_14**
- **return\_lag\_21**
- **return\_lag\_30**

These represent momentum across short, medium, and long horizons.

## 4.4 Rolling window features

- **Close\_roll\_7** – 7-day moving average of the closing price
- **Vol\_roll\_7** – 7-day moving average of volume

These capture smoothed market behavior.

## 4.5 Time-based features

Extracted from the date index:

- **date** (timestamp)
- **hour**
- **dayofweek**
- **quarter**
- **month**
- **year**
- **dayofyear**
- **dayofmonth**

These allow the model to learn seasonal or periodic behavior.

## 4.6 Target variables

- **target** – true next-period value being predicted
- **xgb\_prediction** – stored predictions from the XGBoost model (used for comparing actual vs predicted)

## Why this matters for XGBoost

XGBoost benefits the most from large, diverse feature sets because:

- It handles nonlinear interactions extremely well.
- It is less sensitive to scaling.
- It automatically selects the most useful features.
- It thrives when given engineered temporal and statistical patterns.

Thus, XGBoost becomes the strongest model due to this rich feature engineering pipeline.

Feature engineering focused on capturing temporal patterns and short-term momentum:

### Core features used:

- Raw OHLCV columns (Close, Open, High, Low, Volume).
- Lag features: Close(t-1), Close(t-2), ... (several lags to capture autoregressive behavior).
- Rolling statistics: rolling mean, rolling standard deviation, rolling min/max over windows (e.g., 3, 7, 14 periods).
- Time features: day of year, week of year, day of month, hour of day (if intraday), day of week, month, quarter.
- Volatility proxies: intraday range (High–Low), volume-related features (rolling volume mean, volume ratio).

These features were used across the models with model-specific transformations (e.g., differencing for ARIMA; no heavy scaling for tree-based models like XGBoost; standardized inputs for SVR).

---

## 5. Modeling approaches

### 5.1 Prophet

- Used for decomposable trend+seasonality modeling.
- Helpful for capturing weekly/seasonal structures if present; less effective for rapid regime shifts or noisy minute-level data.

### 5.2 XGBoost (best performer)

- Gradient-boosted trees trained on lag features, rolling statistics, and time features.
- Handled nonlinearities, interactions, and noisy signals better than linear/statistical models.
- Best observed performance: **2.189% percentage error** (user-reported).

### 5.3 Support Vector Regression (SVR)

- Trained on engineered features with scaling.
  - Performed significantly worse (~19.911% percentage error) indicating sensitivity to hyperparameters and possibly inadequate feature scaling or kernel choice for the task.
- 

## 6. Experimental setup & evaluation

**Train/validation split:** historical split with the 20% held out as the test/validation window.

**Evaluation metrics:** percentage error (user-provided), mean absolute error (MAE), root mean square error (RMSE), and visual inspection of forecasts versus ground truth.

**Cross-validation:** time-series-aware validation (walk-forward) was recommended but limited by the short test window; Phase Two should implement a robust walk-forward scheme.

---

## 7. Results summary

### Model Performance Comparison Table (to be filled)

model	RMSE	MAPE
Prophet	<b>3605.8183</b>	<b>7.563</b>
XGBoost	<b>1934.587</b>	<b>2.189</b>
SVM	<b>19937.72</b>	<b>19.34</b>

### Key findings:

- XGBoost is the strongest candidate with a very low percentage error (2.2224%).

- SVR underperformed (19.911%), suggesting that kernel selection, scaling, or feature set require rework.
- ARIMA and Prophet provided useful baselines but were not competitive with XGBoost on the engineered feature set.

**Visuals:** The report includes placeholders for the following plots (to be inserted with the user's images):

- Actual vs predicted (per model) on the test window.
  - Residual analysis plots (error distribution, autocorrelation of residuals).
  - Feature importance from XGBoost (gain/cover) to demonstrate which features drove predictions.
- 

## 8. Discussion & interpretation

- **Why XGBoost worked well:** Tree-based ensembles capture nonlinear interactions between lagged price terms, volume, and engineered time features. They are also robust to feature scaling and moderate noise.
  - **Limitations:**
    - Short test window (two weeks) limits confidence in performance generalization. The metric could be sensitive to a benign or volatile test period.
    - Bitcoin price is influenced by off-chain signals (news, sentiment, macro events) that are not included — this constrains predictive power.
    - Overfitting risk if too many lag/rolling features were used without appropriate cross-validation.
  - **Operational concerns:** model retraining frequency, compute time for feature generation, and latency if deploying for near-real-time use.
- 

## 9. Roadmap & Phase Two plan

Immediate next steps (Phase Two):

1. **Extend evaluation window** — use multiple non-overlapping test windows and an expansive walk-forward cross-validation to validate robustness.
2. **Add exogenous signals** — sentiment (Twitter/Reddit), major news events, on-chain metrics (active addresses, transaction volume), and funding rates from exchanges.
3. **Hyperparameter search & model tuning** — automated search (Bayesian optimization or randomized search) for XGBoost and SVR; tune tree depth, learning rate, regularization, and number of estimators.
4. **Model ensembles & stacking** — combine XGBoost with statistical models and possibly neural sequence models to reduce error and increase robustness.
5. **Multistep forecasting** — train for horizon-aware predictions (1-step, multi-step), and experiment with direct vs recursive forecasting strategies.

6. **Feature selection & regularization** — prune redundant lags and windows, use SHAP or permutation importance for explainability.
7. **Operationalization** — build a retraining pipeline, monitoring for concept drift, create backtesting framework for trading-strategy evaluation.

**Longer-term directions:**

- Incorporate intraday, order-book, and derivatives/funding rate data.
  - Explore transformer-based sequence models or temporal convolutional networks for complex temporal dependencies.
  - Add an investment-decision layer (risk constraints, position sizing) if the model will inform trading.
- 

## 10. Conclusion

Phase One established a clear winner in XGBoost under the experiment constraints and showed that careful feature engineering (lag and time features) meaningfully improves predictive performance on noisy Bitcoin data. However, the result is conditional on the short validation window and the absence of exogenous market signals. The proposed Phase Two roadmap prioritizes robustness, feature expansion, and operational readiness to convert this prototype into a production-capable forecasting system.

---

## 11. References & appendix

**References : kaggle notebook time series forcasting**