HIGH-LEVEL DESIGN (HLD) & LOW-LEVEL DESIGN (LLD) DOCUMENT

OVERVIEW

The High-Level Design explains **what the system does** and **how major components interact**, without going into implementation details. The Low-Level Design explains **how each component is implemented**, including logic, data flow, and functions.

HLD – COMPONENT DESCRIPTION

1 DATA SOURCE

- Historical cryptocurrency data (daily OHLC, volume, market cap)
- Stored as CSV and loaded into the system

2 DATA PREPROCESSING LAYER

- Handles missing values
- Ensures chronological order
- Validates numerical consistency

3 FEATURE ENGINEERING LAYER

- Converts raw data into meaningful indicators
- Creates volatility and liquidity features

4 EDA LAYER

- Visual analysis to understand trends and relationships
- Helps validate feature relevance

5 MACHINE LEARNING LAYER

- Trains regression model to predict volatility
- Evaluates performance using RMSE, MAE, R^2

6 DEPLOYMENT LAYER

- Flask API exposes model for prediction
- ngrok enables access from Google Colab

HLD – Technology Stack

Layer	Technology
Data Handling	Pandas, NumPy
Visualization	Matplotlib, Seaborn
ML Model	XGBoost
Scaling	Scikit-learn
API	Flask
Deployment	Google Colab + ngrok

High-Level Design (HLD) Summary

The High-Level Design defines the overall architecture of the cryptocurrency volatility prediction system and describes how major components interact. The system follows a modular architecture that separates data preprocessing, feature engineering, exploratory analysis, machine learning, and deployment into independent layers.

This design ensures scalability, maintainability, and reproducibility. Time-ordered data processing prevents information leakage, while modular components allow easy modification or extension of features and models. The trained machine learning model is exposed through a Flask API, enabling seamless integration with external applications.

Overall, the HLD provides a structured and reliable framework that bridges data analysis and real-world deployment for cryptocurrency volatility prediction.

Low-Level Design (LLD)

Module-Wise Breakdown

1. Data Loading Module

- Loads CSV data
- Converts date columns
- Sorts data by cryptocurrency and date

---

2. Data Preprocessing Module

- Forward fills missing OHLC values
- Handles volume and market cap inconsistencies
- Removes duplicate and invalid records

---

3. Feature Engineering Module

- Computes log returns
- Calculates rolling volatility (target variable)
- Generates liquidity ratios and moving averages

---

4. EDA Module

- Generates statistical summaries
- Produces visualizations for trends and relationships

---

5. Model Training Module

- Applies feature scaling
- Performs time-based train-test split
- Trains XGBoost regression model

---

6. Model Evaluation Module

- Evaluates performance using RMSE, MAE, and R^2

7. Deployment Module

- Flask API receives input features
- Validates request data
- Returns predicted volatility in JSON format

LLD Summary

The Low-Level Design (LLD) describes the internal working and implementation details of the cryptocurrency volatility prediction system. It explains how raw market data is processed, transformed, and used for prediction at a functional level.

The system begins by loading cryptocurrency data from CSV files, parsing date fields, and sorting records chronologically for each asset. The preprocessing module handles missing values, removes invalid records, and ensures numerical consistency to maintain time-series integrity.

Feature engineering transforms raw data into predictive indicators, including log returns, high–low price spread, rolling volatility, liquidity ratios, volume-based averages, and moving averages. Rolling and grouped calculations ensure that only historical information is used.

The model training module applies feature scaling, performs a time-based train–test split, and trains an XGBoost regression model. Model performance is evaluated using RMSE, MAE, and R^2 metrics.

Finally, the deployment module exposes the trained model through a Flask API, which validates input data, applies the same preprocessing steps, and returns predicted volatility values in JSON format. This modular design ensures consistency, reliability, and ease of integration.