High-Level Design Report: Cryptocurrency Liquidity Prediction

1. Introduction

The High-Level Design (HLD) of the Cryptocurrency Liquidity Prediction project outlines the system's architecture, technology choices,
data flow, and integration points. The aim is to provide a top-level understanding of how the system functions from input to output.

2. System Objective

To predict the liquidity of cryptocurrencies using historical data (price, volume, market cap) by training regression models and
serving predictions via a web interface. This helps traders and institutions manage liquidity risk more effectively.

3. Architecture Overview

The system is divided into the following major components:

- Data Collection & Ingestion
- Data Preprocessing & Feature Engineering
- Model Training & Evaluation
- Model Serialization & Loading
- Web Application Interface (Flask)
- Prediction Service (REST API)

4. Technology Stack

- Python 3.x

- Libraries: Pandas, NumPy, scikit-learn, XGBoost, Flask

- Web Interface: HTML, CSS (via Flask templates)

- Version Control: Git & GitHub

- Optional: Docker for containerized deployment

5. Component Descriptions

1. Data Layer: Reads cryptocurrency datasets in CSV format.

2. Processing Layer: Cleans data, fills missing values, converts formats, and derives features.

3. Model Layer: Trains models (e.g., XGBoost, BaggingRegressor), evaluates them, and serializes the best.

4. API Layer: Exposes endpoints to trigger training (/train) and serve predictions (/predict).

5. UI Layer: Web frontend allowing users to upload data and view predictions.

6. Data Flow Summary

- Input: CSV file with historical cryptocurrency data

- Processing: Preprocessing and feature engineering

- Model: Trained with hyperparameter tuning and evaluated

- Output: Model predictions exposed via RESTful API or web interface

7. Integration Points

- Flask API: Integrates data, model, and frontend layers

- Model file (model.pkl): Loaded during prediction phase

- Templates: Connect backend output to HTML frontend

8. Assumptions and Limitations

- Assumes access to historical cryptocurrency data with sufficient granularity

- Designed for regression tasks, not classification

- Flask app is suitable for local or lightweight deployment; may require scaling for production

9. Security and Logging

- Input validation implemented for file uploads and API parameters

- Logging used to capture errors and track operations