# Cryptocurrency Liquidity Prediction Project Documentation

## 1. Exploratory Data Analysis (EDA) Report

### 1.1 Dataset Statistics

In this section, we provide a comprehensive summary of key descriptive statistics for each numerical feature in the dataset, including measures such as mean, median, standard deviation, minimum, and maximum. Understanding these metrics helps identify skewness, outliers, and scale differences between variables.

### 1.2 Visualizations

We include time-series plots of trading volume and liquidity metrics to observe trends and seasonal patterns. A correlation heatmap is also presented to highlight relationships between features, which informs subsequent feature engineering decisions.

### 1.3 Key Insights

Key observations include periods of unusually high trading volume preceding liquidity drops, indicating potential causality. Missing data patterns were sparse and handled through interpolation without significant information loss.

# 2. High-Level Design (HLD)

## 2.1 System Overview

The system architecture consists of a data ingestion layer that retrieves raw market data, followed by preprocessing modules hosted on a scalable data pipeline. The core model training pipeline operates within a containerized environment, enabling reproducible experiments and deployments.

## 2.2 Components

- **Data Collection Service**: Scheduled jobs to fetch and store raw cryptocurrency data. - **Preprocessing Module**: Data cleaning, normalization, and feature extraction. - **Model Training Pipeline**: Automated training routines with hyperparameter tuning. - **Prediction Service Interface**: RESTful API for serving real-time liquidity forecasts.

# 3. Low-Level Design (LLD)

## 3.1 Module Breakdown

- **Preprocessing**: Functions for handling missing values, outlier detection, and normalization; structured as reusable Python classes with clear interfaces. - **Feature Engineering**: Implementation of moving averages, rolling statistics, and custom liquidity indicators. - **Model Training**: Pipeline using Scikit-Learn's `Pipeline` API, including cross-validation and grid search for hyperparameter tuning. - **Evaluation**: Custom scripts to compute RMSE, MAE, $R^2$, and to generate performance reports in HTML format.

## 3.2 Class/Function Diagrams

UML diagrams illustrate class hierarchies and function interactions for preprocessing and modeling modules. These diagrams ensure consistent implementation and facilitate onboarding of new developers.

# 4. Pipeline Architecture

## 4.1 Data Ingestion

Data is ingested via API connectors to cryptocurrency exchanges and stored in a time-series database. Automated retries and logging ensure reliability.

## 4.2 Data Processing

Processing includes data validation, cleaning, feature computation, and storage in a data warehouse for analytics.

## 4.3 Model Pipeline

Sequential execution of feature engineering, model training, validation, and artifact storage, managed by an orchestration tool such as Airflow.

## 4.4 Deployment Workflow

A Dockerized Flask application serves the liquidity prediction model. CI/CD pipelines handle testing, container builds, and deployment to cloud infrastructure.

# 5. Final Report

## 5.1 Project Summary

This project aims to forecast cryptocurrency liquidity levels to support market stability initiatives. Accurate liquidity predictions can inform trading strategies and risk management.

## 5.2 Methodology

We followed an iterative workflow: data collection, preprocessing, EDA, feature engineering, model development, and evaluation. Each phase informed decisions in subsequent stages.

## 5.3 Results

The Random Forest model achieved an RMSE of X.XX and $R^2$ of 0.XX on the test set, demonstrating robust predictive performance for liquidity forecasting.

## 5.4 Conclusions & Next Steps

Model performance indicates potential for real-time deployment. Future work includes exploring LSTM models for sequence forecasting and integrating alternative data sources.