# 1. High-Level Design (HLD) Document

## 1.1 Project Overview

The **Cryptocurrency Volatility Forecaster** is a Machine Learning-based application designed to analyze historical cryptocurrency market data and predict future volatility. The system helps traders and financial analysts identify periods of market instability to manage risk and optimize portfolio allocation.

## 1.2 System Architecture

The system follows a standard **Machine Learning Pipeline Architecture**, consisting of three main layers: the Data Layer, the Logic/Processing Layer, and the Presentation Layer.

- **Data Layer**: Responsible for ingesting raw OHLCV (Open, High, Low, Close, Volume) data from CSV files.
- **Processing & Model Layer**: Handles data cleaning, feature engineering (technical indicators), model training, and inference using Scikit-Learn.
- **Presentation Layer**: A web-based User Interface (UI) built with Streamlit that allows users to select cryptocurrencies and visualize predictions.

## 1.3 Key Components

| **Component** | **Description** | **Tech Stack** |
|---|---|---|
| **Data Ingestion** | Reads raw historical data (dataset.csv). | Pandas |
| **Preprocessing Engine** | Cleans null values, sorts time-series data, and calculates log returns. | Pandas, NumPy |
| **Feature Engineering** | Computes rolling volatility, liquidity ratios, and lag features. | Pandas Rolling Window |
| **Prediction Model** | A Random Forest Regressor trained to forecast the 7-day future volatility. | Scikit-Learn |
| **Frontend UI** | Interactive dashboard for selecting assets and viewing charts. | Streamlit, Plotly |

## 1.4 Data Flow Diagram (DFD) - Level 0

1. **User** uploads dataset.csv or selects a crypto in the UI.
2. **System** processes historical prices $\rightarrow$ Calculates Volatility & Returns.
3. **Model** receives features $\rightarrow$ Predicts Next_7_Day_Volatility.
4. **UI** renders historical trends and prediction results to the **User**.

---

# 2. Low-Level Design (LLD) Document

## 2.1 Database / Data Schema Design

The system uses a flat-file database (dataset.csv). The schema for the input data is as follows:

| **Column Name** | **Data Type** | **Description** |
|---|---|---|
| date | DateTime | The record date (YYYY-MM-DD). |
| crypto_name | String | Name of the cryptocurrency (e.g., Bitcoin). |
| open | Float | Opening price of the day. |
| high | Float | Highest price of the day. |
| low | Float | Lowest price of the day. |
| close | Float | Closing price of the day (Primary feature). |
| volume | Float | Total trading volume. |
| marketCap | Float | Total market capitalization. |

## 2.2 Module Specifications

**Module 1: Data Preprocessing (`load_and_preprocess`)**

- **Input**: File path (String).
- **Logic**:
    1. Convert `date` column to datetime objects.
    2. Sort data by `crypto_name` and `date` to ensure time-series continuity.
    3. Transformation: Calculate Log Returns to stationarize price data.
       $$\text{Log Return} = \ln(\frac{\text{Close}_t}{\text{Close}_{t-1}})$$
- **Output**: Cleaned Pandas DataFrame.

**Module 2: Feature Engineering**

This module creates the input vectors for the machine learning model.

- **Input**: Cleaned DataFrame.
- **Features Created**:
    - **Rolling Volatility (7d)**: Standard deviation of log returns over a 7-day window.
        - `df['log_ret'].rolling(window=7).std()`
    - **Liquidity Ratio**: `Volume / MarketCap`.
    - **Previous Close**: Lag feature used to calculate daily changes.
- **Handling NaNs**: The first 7 rows of every crypto group are dropped because the rolling window requires historical data.

**Module 3: Model Architecture**

- **Algorithm**: Random Forest Regressor.
- **Reason for Selection**: Handles non-linear relationships well and is robust against overfitting compared to simple linear regression.
- **Hyperparameters**:
    - `n_estimators`: 100 (Number of trees).
    - `random_state`: 42 (For reproducibility).
- **Training Strategy**:
    - **Split**: Time-Series Split (Last 20% of data used for testing to prevent look-ahead bias).
    - **Target Variable**: `volatility_7d` (shifted backward during training to align past features with future volatility).

**Module 4: User Interface (`app.py`)**

- **Libraries**: Streamlit, Plotly Express.
- **Functions**:
    - `st.selectbox()`: Dropdown for user to filter by `crypto_name`.
    - `st.plotly_chart()`: Renders interactive line graphs for Close Price and Volatility.
    - `Prediction Logic`: Since the model predicts *future* volatility, the app takes the *last known* volatility and applies the model's logic (or a simulated drift for the demo) to project the trend.

## 2.3 Exception Handling

- **FileNotFoundError**: If `dataset.csv` is missing, the app displays a formatted error message via `st.error` instructing the user to upload the file.

- **Empty Data**: Checks if the filtered DataFrame is empty before attempting to plot charts to prevent crashes.

## 2.4 Deployment Strategy (Local/Colab)

- **Tunneling**: Uses cloudflared (Cloudflare Tunnel) to expose the local Streamlit port (8501) to a public URL via an encrypted tunnel.
- **Server**: Runs on a Python 3.8+ environment.