High-Level Design (HLD) Document

The HLD provides an architectural overview, defining the main components and their interactions in the machine learning system.

1. System Goal

To predict the **Cryptocurrency Liquidity Index** using historical market data, enabling proactive detection of potential liquidity crises and informing risk management strategies.

2. Architectural Diagram

The system follows a standard batch-processing Machine Learning Pipeline architecture.

3. Key Components and Technologies

Component	Description	Technologies
Data Source	Raw historical cryptocurrency market data (price, volume, market cap).	CSV files (coin_gecko_*.csv)
Data Ingestion	Module for loading, merging, and initial cleaning of the raw data files.	**Pandas**
Data Processor	Handles data quality checks, missing value imputation, and feature scaling.	**Pandas, NumPy, Scikit-learn (MinMaxScaler)**
Feature Engineer	Creates the target **Liquidity Index** and other derivative features (e.g., Volume/Price Ratio).	**Pandas, NumPy**
ML Model	The core prediction engine selected for non-linear regression and robustness.	**Scikit-learn (RandomForestRegressor)**
Model Evaluation	Measures model performance against key business and statistical metrics.	**Scikit-learn (RMSE, MAE, R^2)**
Persistence Layer	Stores the trained model and the feature scaler for later use/deployment.	**Joblib (.pkl files)**

4. Data Flow

1. **Ingestion:** Raw .csv files are read and merged into a single DataFrame.
2. **Preprocessing:** Data is cleaned (median imputation) and prepared for feature creation.
3. **Feature Engineering:** The **Liquidity_Index** target is calculated.
4. **Scaling & Split:** All numerical features are scaled, and the dataset is split into training (80%) and testing (20%) sets.
5. **Training:** The **RandomForestRegressor** is trained on the scaled training data.
6. **Evaluation:** The model's predictions on the test set are evaluated using defined metrics.
7. **Output:** The final model and scaler are saved to disk using joblib.

Low-Level Design (LLD) Document

The LLD details the implementation of each core module, including function specifications, inputs, outputs, and algorithms.

1. Project Directory Structure

```
├── cryptocurrency_liquidity_prediction.py  # Main ML Pipeline script
├── coin_gecko_2022-03-17.csv           # Data File 1
├── coin_gecko_2022-03-16.csv           # Data File 2
├── liquidity_predictor.pkl             # Output: Trained Model
└── data_scaler.pkl                     # Output: Trained Scaler
```

2. Module Specifications (Function Detail)

A. load_and_merge_data(file_paths)

- **Purpose:** Reads raw market data from specified CSV files and concatenates them.
- **Algorithm:** Iterative read, `pd.concat`, and sorting by `coin` and `date`.
- **Inputs:** `file_paths` (List of strings, e.g., `['file1.csv', 'file2.csv']`).
- **Outputs:** `df_combined` (Pandas DataFrame).

B. preprocess_data(df)

- **Purpose:** Cleans the DataFrame by handling non-numeric columns and imputing missing values.
- **Algorithm:**
    1. Drop categorical columns (`coin`, `symbol`, `date`).
    2. Use the **Median** to fill all remaining `NaN` values across all numerical columns.
- **Inputs:** `df` (Raw Pandas DataFrame).
- **Outputs:** `df_cleaned` (Pandas DataFrame with only numerical, imputed features).

C. feature_engineering(df)

- **Purpose:** Creates the predictive target variable and a highly correlated feature.
- **Algorithm:**
    - Target Creation (Liquidity_Index):

      $$\text{Liquidity Index} = \frac{\text{24h\_volume} / \text{mkt\_cap}}{|\text{24h change}| + \epsilon}$$

    - **Feature Creation (Vol_Price_Ratio):** $\text{Vol\_Price\_Ratio} = \frac{\text{24h\_volume}}{\text{price} + \epsilon}$
- **Inputs:** `df` (Cleaned DataFrame).
- **Outputs:** `df_featured` (DataFrame including the Liquidity_Index target).

D. prepare_for_training(df)

- **Purpose:** Scales features and splits the data into training and testing sets.
- **Algorithm:**

1. Initialize **MinMaxScaler**.
2. Fit and transform features (X).
3. Use **train_test_split** with test_size=0.2 and shuffle=False (to maintain time order).

- **Inputs:** df (Featured DataFrame).
- **Outputs:** X_train, X_test, y_train, y_test, scaler (fitted MinMaxScaler).

E. train_and_evaluate_model(X_train, X_test, y_train, y_test)

- **Purpose:** Trains the chosen model and assesses its performance.
- **Algorithm:**
  1. **Model:** Instantiate **RandomForestRegressor** (n_estimators=100, max_depth=10).
  2. **Training:** Call model.fit(X_train, y_train).
  3. **Prediction:** Call model.predict(X_test).
  4. **Evaluation Metrics:** Calculate **RMSE**, **MAE**, and **R^2 score**.
- **Inputs:** Train/Test splits (X and y).
- **Outputs:** model (Trained RandomForestRegressor object).

3. Model Selection Justification

Approach	Model	Justification
Algorithm	**RandomForestRegressor**	Non-linear model capable of capturing complex interactions between price, volume, and market cap. Robust against high data skewness and less sensitive to feature collinearity compared to linear models.
Scaling	**MinMaxScaler**	Essential for normalizing the vast differences in magnitude between features (e.g., price vs. mkt_cap) to ensure fair contribution during model training.