Low-Level Design Document

Cryptocurrency Liquidity Prediction Project

Contents

1 Objective

This Low-Level Design (LLD) document provides the implementation-level design of the Cryptocurrency Liquidity Prediction System. It breaks down the components, defines key classes and functions, and outlines the internal interactions within the project pipeline.

2 Scope

The LLD elaborates:

- Implementation of core modules
- Internal function signatures and logic
- Input/output formats and dependencies
- Data flow between components

3 Technology Stack

- **Language:** Python 3.11+
- **Libraries:** pandas, numpy, scikit-learn, xgboost, matplotlib, seaborn, joblib, streamlit
- **Visualization:** matplotlib, seaborn
- **Modeling:** XGBoost, RandomForest, Linear Models

4 Folder Structure

```
cryptocurrency_price_prediction/
        data/
            raw/
            processed/
        notebooks/
            01_data_preprocessing.ipynb
            02_eda.ipynb
            03_feature_engineering.ipynb
            04_model_selection.ipynb
            05_model_training.ipynb
        src/
            data_loader.py
            data_processor.py
            feature_engineer.py
            models.py
            evaluator.py
            utils.py
        models/
            final_xgboost_model.pkl
            feature_columns.pkl
        deployment/
            app.py
```

5 Module-Level Design

5.1 1. `data_loader.py`

Responsibility: Load raw CSV files.

- `load_raw_data(file_path: str)` → `pd.DataFrame`

5.2 2. `data_processor.py`

Responsibility: Handle data cleaning.

- Remove missing values
- Convert types (e.g., dates)
- Normalize column formats

`clean_data(df: pd.DataFrame)` → `pd.DataFrame`

5.3 3. `feature_engineer.py`

Responsibility: Add new features like:

- Rolling means/volatility
- Date/time parts (day, week, etc.)
- Price change percentages

`create_features(df: pd.DataFrame)` → `pd.DataFrame`

5.4 4. `models.py`

Responsibility: Train and save models.

- `train_model(X, y)` → `trained_model`
- `save_model(model, path)`

5.5 5. `evaluator.py`

Responsibility: Evaluate trained models.

- `evaluate(y_true, y_pred)` → `dict[MAE, RMSE, R`^{2}`]`

5.6 6. `app.py` (Streamlit)

Responsibility: Load model and serve predictions via GUI.

- Load `final_xgboost_model.pkl`
- Accept inputs for features
- Display predictions

6 Data Flow

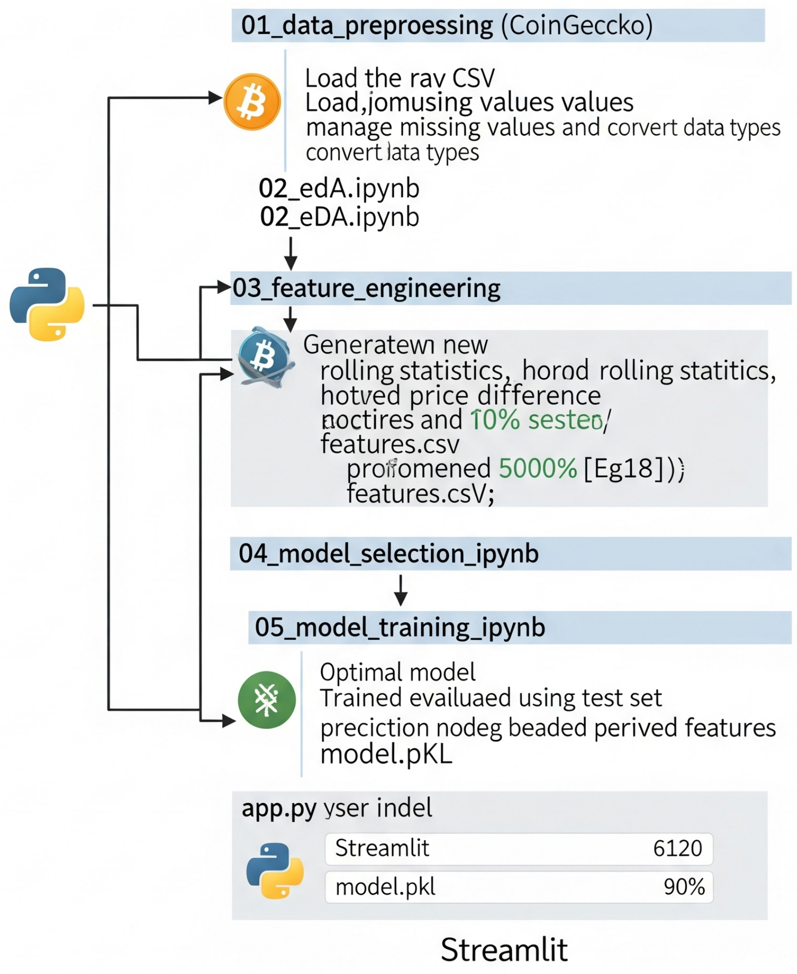

Figure: Data Flow from Raw to Prediction

7 Input/Output Formats

7.1 Inputs

- Raw data: CSV from CoinGecko API
- Cleaned data: `cleaned_crypto_price.csv`
- Feature set: `features.csv`
- User inputs (via Streamlit): numeric values for selected features

7.2 Outputs

- Trained model: `final_xgboost_model.pkl`
- Metrics: MAE, RMSE, R^2
- Prediction: Liquidity value (float)

8 Dependencies

`requirements.txt` includes:

```
pandas, numpy, xgboost, scikit-learn, matplotlib,
seaborn, joblib, streamlit
```

9 Conclusion

This LLD provides detailed insight into the system's implementation. Each module and its responsibilities are clearly defined to support scalability, maintainability, and deployment readiness.