1. Title Page

- **Project:** Cryptocurrency Liquidity Prediction
- **Report:** Low-Level Design (LLD)
- **Prepared by:** Arju Rewatkar
- **Date:** [Insert Date]

---

2. Introduction

The Low-Level Design (LLD) document provides a detailed description of each module in the Cryptocurrency Liquidity Prediction project, including data preprocessing, feature engineering, model training, evaluation, and deployment. It ensures developers have a clear implementation plan.

---

3. Module-wise Detailed Design

3.1 Data Preprocessing Module

- **Script:** `src/data_preprocessing.py`
- **Responsibilities:**
  - Handle missing values and duplicates
  - Normalize numerical features
  - Generate processed CSV files
- **Inputs:** `data/raw/crypto_raw.csv`
- **Outputs:** `data/processed/crypto_processed.csv`
- **Functions:**
  - `load_data()` – Load raw CSV

    - `clean_data()` – Remove missing/duplicate rows
    - `normalize_features()` – Scale features
    - `save_processed_data()` – Save cleaned data

3.2 Feature Engineering Module

- **Script:** `src/feature_engineering.py`
- **Responsibilities:**
    - Generate new features for prediction:
        - Liquidity Index (Volume / Market Cap)
        - Daily Return (% change in Close price)
        - Rolling Volatility
- **Inputs:** `data/processed/crypto_processed.csv`
- **Outputs:** `data/processed/crypto_features.csv`
- **Functions:**
    - `calculate_liquidity_index()`
    - `calculate_daily_return()`
    - `calculate_rolling_volatility()`

3.3 Modeling Module

- **Script:** `src/models.py`, `src/train.py`
- **Responsibilities:**
    - Train ML models (Random Forest, XGBoost, LSTM)

    - Save trained models for prediction
- **Inputs:** `data/processed/crypto_features.csv`
- **Outputs:** `models/` folder containing trained model files
- **Functions:**
    - `train_model()` – Train selected ML algorithm
    - `save_model()` – Save trained model
    - `load_model()` – Load model for evaluation/deployment

3.4 Evaluation Module

- **Script:** `src/evaluate.py`
- **Responsibilities:**
    - Evaluate model performance
    - Generate plots and metrics (RMSE, MAE, R²)
- **Inputs:** Trained model + test data
- **Outputs:** Evaluation report + plots in `reports/figures/`
- **Functions:**
    - `predict()` – Make predictions on test set
    - `calculate_metrics()` – Compute evaluation metrics
    - `plot_results()` – Visualize predictions vs actual

3.5 Deployment Module

- **Script:** `deployment/app.py`

- **Responsibilities:**
    - Deploy trained model for end-user predictions
    - Provide web interface using Streamlit/Flask

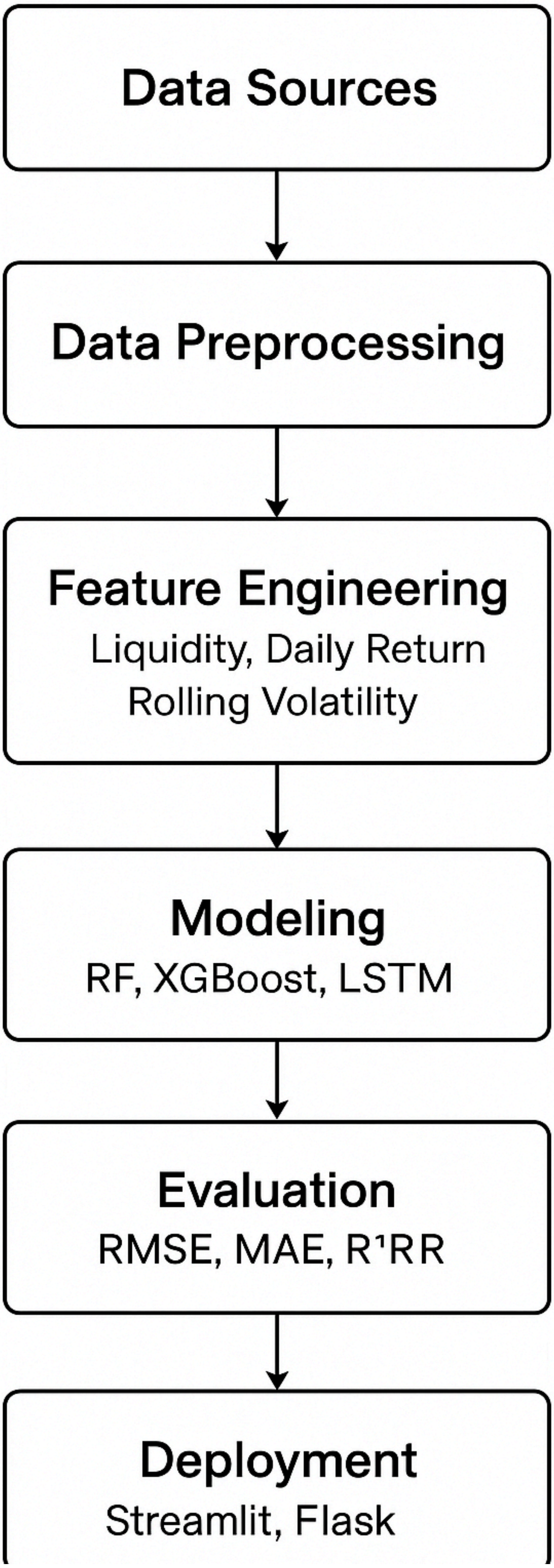
Folder Structure:

```
deployment/
├── app.py
├── templates/       # HTML files if Flask used
└── static/          # CSS, JS, images
```

-
- **Functions:**
    - `load_model()` – Load trained model
    - `predict_input()` – Predict liquidity based on user input
    - `render_ui()` – Display results in web app

---

4. Data Flow Diagram (Optional)


Data Sources
Data Preprocessing
Feature Engineering
Liquidity, Daily Return
Rolling Volatility
Modeling
RF, XGBoost, LSTM
Evaluation
RMSE, MAE, R^{1}RR
Deployment
Streamlit, Flask

5. File & Folder Structure Reference

```
Crypto-Liquidity-Prediction/
├── src/
│   ├── data_preprocessing.py
│   ├── feature_engineering.py
│   ├── models.py
│   ├── train.py
│   └── evaluate.py
├── deployment/
│   ├── app.py
│   ├── templates/
│   └── static/
├── data/
│   ├── raw/
│   └── processed/
└── reports/
    ├── figures/
```

6. Conclusion

The LLD document ensures developers can implement each module clearly, understand dependencies, and maintain a smooth workflow from raw data to deployed prediction.