Project Title:

Cryptocurrency Liquidity Prediction for Market Stability

By:
Jagrati sen

Table of Contents

1. Problem Statement

Cryptocurrency markets are extremely volatile, with prices influenced by trading volume, investor sentiment, and liquidity.

Liquidity refers to how easily a cryptocurrency can be bought or sold without affecting its price. Poor liquidity can cause:

- Sudden price drops or spikes
- Higher risk for investors
- Reduced market trust

The goal of this project is to:

- Predict future **liquidity levels** using historical market data
- Provide early warnings for potential **liquidity crises**
- Help traders, exchanges, and institutions make better financial decisions

We achieve this using **machine learning models** trained on 2016–2017 cryptocurrency data, and deploy the solution using a **Streamlit web app**.

2. Dataset Overview

Source:

- Historical cryptocurrency data (2016–2017)
- Collected from open datasets (e.g., CoinGecko, Kaggle)
- File format: CSV

Key Columns:

Column Name	Description
`price`	Daily closing price of the cryptocurrency
`volume`	Daily traded volume
`market_cap`	Total market capitalization on that day

Summary:

- Number of records: ~700 (daily entries over ~2 years)
- Missing values: Present in some columns, handled in preprocessing
  Target variable: **Liquidity Ratio** (calculated as `volume / market_cap`)

This dataset was chosen for its relevance to the liquidity prediction problem, covering core market indicators over time.

3. Data Preprocessing

Before training the machine learning model, the dataset underwent several preprocessing steps to ensure quality and consistency.

Cleaning Steps:

- **Missing Values**: Removed rows with null or NaN values using `dropna()`
- **Duplicates**: Verified and confirmed there were no duplicate rows
- **Date Formatting**: Converted date column to `datetime` format (if present)

Scaling:

- Applied **MinMaxScaler** to normalize the features between 0 and 1
- This prevents features with large values (like `market_cap`) from dominating the model

Final Data:

After preprocessing:

- The dataset was clean, consistent, and ready for feature engineering
- Numeric features were scaled to a uniform range
- Outliers were visually inspected but not removed, since volatility is part of the crypto market behavior

4. Feature Engineering

To improve the predictive power of the model, we created new features based on historical patterns in the data.

Features Created:

Feature Name	Description
`price_ma_7`	7-day moving average of price – helps capture price trends
`price_volatility`	7-day rolling standard deviation of price – reflects market uncertainty
`liquidity_ratio`	volume / market_cap – shows how easily an asset can be traded

These features were chosen based on domain knowledge:

- Moving averages smooth out noise and highlight trends
- Volatility shows risk or instability
- Liquidity ratio is a key indicator of market stability

Why These Features?

These engineered features help the model learn relationships that are not obvious in raw data, improving prediction quality and robustness.

5. Model Selection & Training

The core objective was to predict a **continuous numeric value** (liquidity ratio), so we used **regression models**.

Model Used:

- **Linear Regression**
  Simple, interpretable, and efficient for continuous target prediction.

Tools & Libraries:

- `scikit-learn` for model building
- `train_test_split` to split the data into training and test sets
- `Google Colab` as the development environment

Data Splitting:

- **80%** of the data used for training
- **20%** held out for testing
- Random state fixed to ensure reproducibility

python

```
from sklearn.model_selection import train_test_split

X = df[['price_ma_7', 'price_volatility', 'liquidity_ratio']]
y = df['liquidity_ratio']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Model Training:

python

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
```

The model was trained on the processed and engineered features to learn how historical volatility and trends relate to liquidity.

6. Model Evaluation

To measure how well the model predicts liquidity, we used standard regression evaluation metrics:

Evaluation Metrics:

Metric	Description
RMSE	Root Mean Squared Error — measures average prediction error
MAE	Mean Absolute Error — average of all absolute differences
R^2 Score	Coefficient of Determination — explains how much variance is captured

Results from Linear Regression:

Metric	Value
RMSE	0.0215
MAE	0.0180
R^2 Score	0.9275

Interpretation:

- **Low RMSE & MAE** values indicate accurate predictions
- **High R^2 Score (0.92+)** means the model explains over 92% of the variance in liquidity
- The model generalizes well to unseen test data

7. High-Level Design (HLD)

This section explains the **overall architecture** and flow of the system from raw data to deployed application.

System Overview:

```
Raw Dataset (CSV)
        ↓
Data Preprocessing
        ↓
Feature Engineering
        ↓
Model Training (Linear Regression)
        ↓
Model Evaluation (RMSE, MAE, R²)
        ↓
Model Saved (model.pkl)
        ↓
Streamlit App (User Inputs)
        ↓
Live Prediction Output
```

Tools Used:

- **Google Colab** – for data cleaning, feature engineering, model training
- **Scikit-learn** – for regression modeling
- **Pickle** – to save the trained model
- **Streamlit** – for web-based UI and deployment
- **GitHub + Streamlit Cloud** – to host and run the app online

8. Low-Level Design (LLD)

This section details the specific components, inputs, outputs, and logic inside the system.

Components:

File	Role
`app.py`	Streamlit script to run the web app
`model.pkl`	Serialized trained model using Pickle
`requirements.txt`	Python packages needed to run the app

Input Features from User:

1. 7-day Price Moving Average (`price_ma_7`)
2. Price Volatility (`price_volatility`)
3. Previous Liquidity Ratio (`liquidity_ratio`)

Processing Logic:

- Inputs are collected via Streamlit's `number_input`
- The model is loaded using `pickle.load()`
- Prediction is made using `model.predict()`
- Result is displayed via `st.success()`

User Flow:

```
User opens the app →
Enters input values →
Clicks "Predict Liquidity" →
App shows the predicted value
```

9. Pipeline Architecture

This section shows the **step-by-step data flow** from the raw dataset to the deployed web app.

Data Flow Pipeline:

```
Raw CSV Dataset
      ↓
Data Cleaning (drop missing values)
      ↓
Feature Engineering (MA, volatility, liquidity ratio)
      ↓
Scaling (MinMaxScaler)
      ↓
Train/Test Split (80/20)
      ↓
Model Training (Linear Regression)
      ↓
Model Evaluation (RMSE, MAE, R²)
      ↓
Save model using Pickle (model.pkl)
      ↓
Deploy app using Streamlit
      ↓
User Inputs → Model Prediction → Output on Web App
```

Technologies Used at Each Stage:

Stage	Tool/Library
Data handling	pandas
Feature creation	pandas (rolling)
Scaling & training	scikit-learn
Model saving	pickle
Frontend (UI)	Streamlit
Hosting	Streamlit Cloud

10. Streamlit App Deployment

This project was deployed as an **interactive web application** using **Streamlit Cloud**. The app allows users to input market data and receive a **predicted liquidity ratio** instantly.

App Features:

- Easy-to-use interface with input sliders and buttons
- Accepts 3 user inputs:
  - `price_ma_7` (7-day price moving average)
  - `price_volatility` (rolling price volatility)
  - `liquidity_ratio` (previous liquidity)
- Predicts next liquidity ratio using a trained model
- Displays results in real time

Deployment Stack:

Task	Tool Used
App Framework	Streamlit
Model Serialization	Pickle
Code Hosting	GitHub
Deployment	Streamlit Cloud

🔗 App Link: [APP]

11. Conclusion

This project successfully demonstrates how machine learning can be applied to predict **cryptocurrency liquidity** using historical data. Liquidity is one of the most important indicators of market stability, and this app helps visualize and forecast it effectively.

Achievements:

- Cleaned and preprocessed real-world financial data
- Created meaningful features such as moving averages and volatility
- Trained a regression model with strong performance ($R^2 \approx 0.92$)
- Deployed a working Streamlit app for real-time prediction
- Delivered a complete ML pipeline from raw data to web app

Key Learnings:

- The importance of feature engineering in financial modeling
- How volatility and trading patterns affect liquidity
- How to deploy machine learning apps using Streamlit and GitHub
- The value of user-friendly interfaces in model communication

12. References

- Public Crypto Data: CoinGecko / Kaggle datasets
- NumPy & Pandas Official Docs