

## **Project Blueprint**

### **Team 4:**

**Dataset:** Historical Bitcoin 1-Minute Price Data (Batch)

**API:** CoinCap Live Cryptocurrency Price API (Streaming)

**Date:** November 23

### **1.1. Business Problem**

Cryptocurrency markets move extremely fast, and price volatility is one of the biggest risks for traders, exchanges, and automated trading systems. Low-latency price prediction helps traders react to rapid price changes, exchanges set better liquidity thresholds, algorithmic trading models adjust risk, and risk managers who monitor volatility in real time.

We intend to propose a solution to this issue by utilizing historical data in order to gain a better understanding of Bitcoin price movements and real-time streaming metrics. Our question would be framed the following way: Can we use historical Bitcoin price movements and real-time streaming metrics to predict short-term BTC price direction and volatility?

### **1.2. Data Sources**

#### **A. Batch Dataset (Kaggle)**

**Dataset:** [btccusd\\_1-min\\_data.csv](#) ( $\approx 7.3$ M rows)

**Fields:** Timestamp, Open, High, Low, Close, Volume

**Timeframe:** 2012 – Present

**Storage:**

- Raw CSV → **GCS bucket**
- Clean partitioned table → **BigQuery**

Batch dataset provides:

- Long-term OHLCV structure

- Lagged features
  - Volatility indicators
  - Moving averages
  - Trend features
  - Training data for BQML
- 

## **B. Streaming Data (Public API)**

**API:** CoinCap Live Prices (no auth needed)

Endpoint:

<https://api.coincap.io/v2/assets/bitcoin>

Streaming provides real-time:

- Current BTC price
- 24h volume
- 24h change %
- Market cap

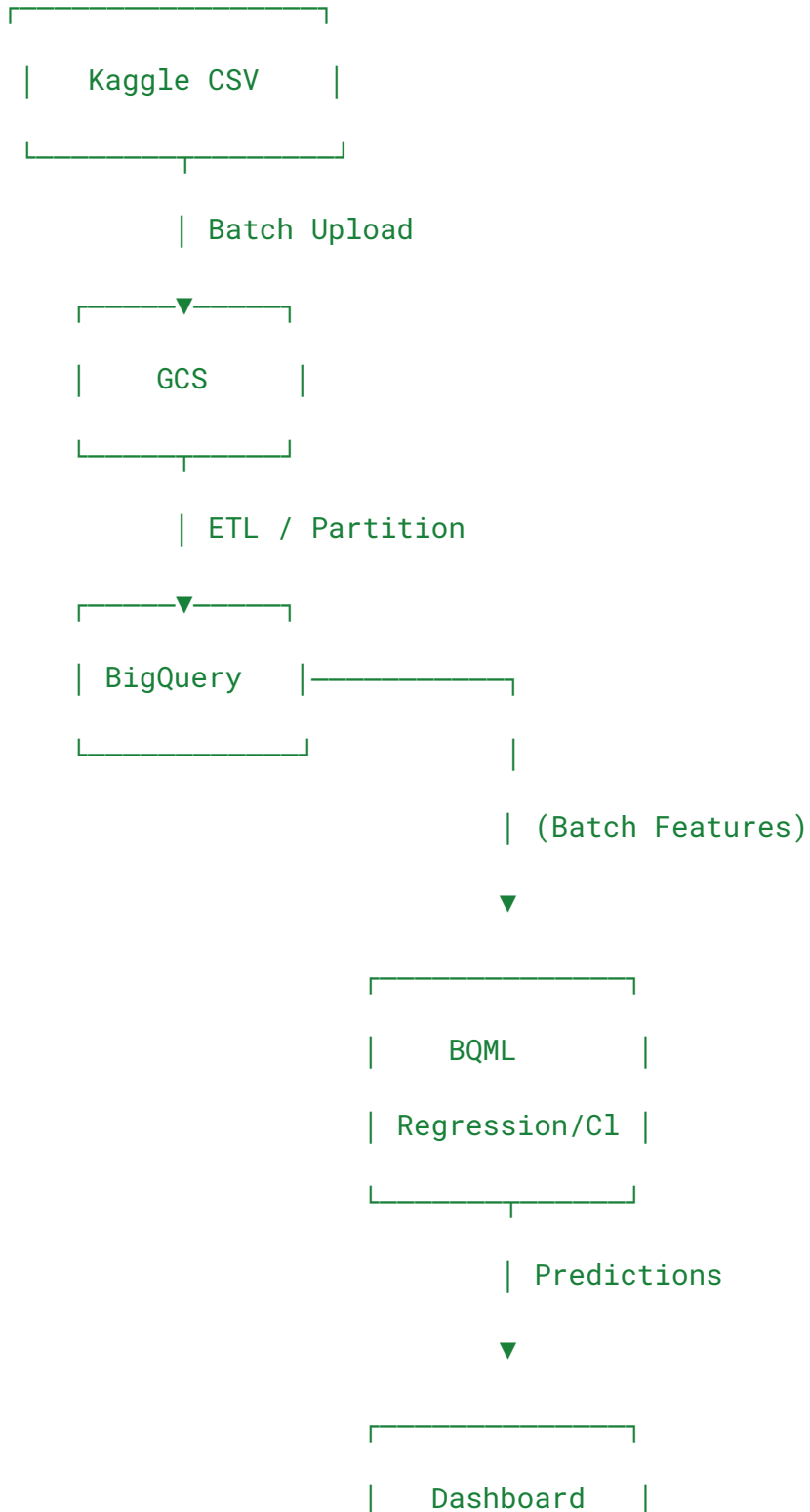
These metrics refresh every few seconds, perfect for a streaming pipeline.

### **Use in ML:**

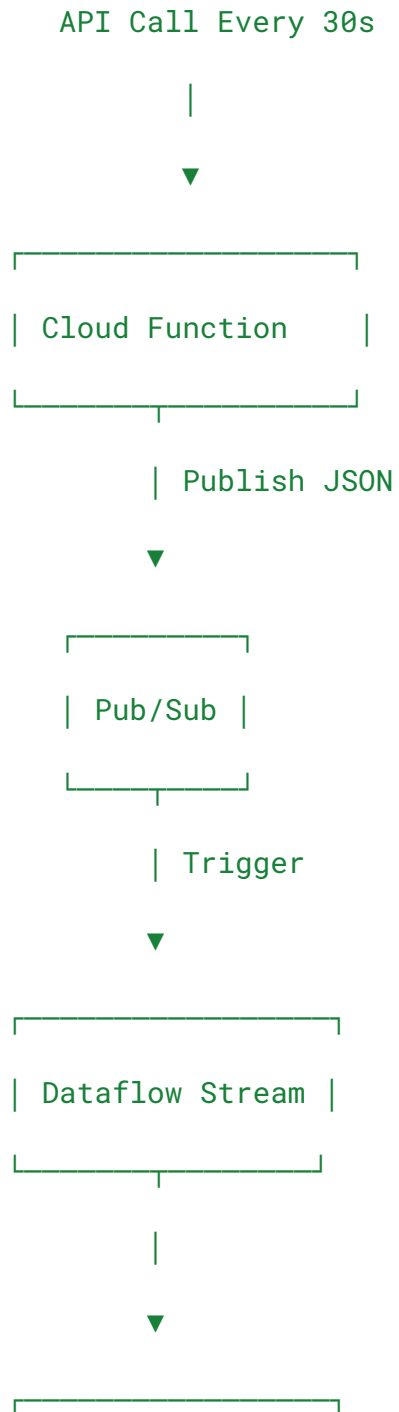
- Join streaming features with batch features
  - Predict next 1-minute movement (up/down)
  - Predict volatility clusters
-

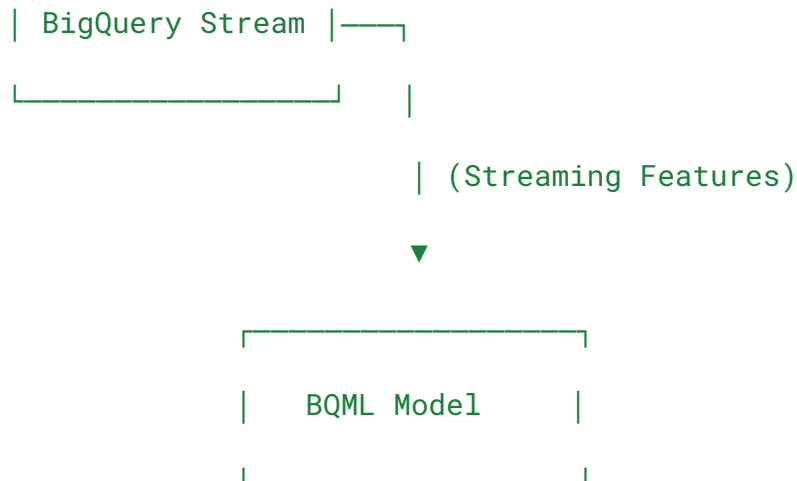
## 2.1 Architecture Overview

### Ingestion Pipeline



## Streaming Pipeline





---

## 2.2. Detailed Technical Plan

### 2.2.1 Batch Ingestion

- Upload the raw CSV to **GCS**
- Create a curated BigQuery table:
  - Partitioned by **DATE(Timestamp)**
  - Clustered by **Close, Volume**
- Apply transformations:
  - Convert timestamp → DATETIME
  - Create returns, volatility, lag features
  - Remove duplicates and zero-volume rows

### Data Quality Check

- Check for missing timestamps

- Check for negative prices
  - Ensure monotonic timestamp order
- 

## 2.2.2 Streaming Ingestion

### Cloud Function (2nd Gen)

- Python function hits [api.coincap.io](https://api.coincap.io)
- Normalizes JSON into:

timestamp

price\_usd

percent\_change\_24h

volume\_24h

market\_cap

- Publishes to **Pub/Sub** topic

### Dataflow Template

- Pub/Sub → JSON → BigQuery Streaming Table
- Table partitioned by ingestion\_time

### Validation

- Query BQ with:

```
SELECT * FROM streaming_table
```

```
ORDER BY ingestion_time DESC  
  
LIMIT 10
```

Should show near real-time rows.

---

### 3. Machine Learning Plan (BQML)

Our approach uses BQML to build two models: a classification model to predict whether Bitcoin's price will move up or down in the next minute, and a regression model to estimate the actual next-minute close price. We'll engineer short-term lag, volatility, and rolling window features that capture immediate market momentum, making the models responsive to real-time changes. We will then evaluate performance using ML.EVALUATE, analyze feature impact with ML.EXPLAIN\_PREDICT, and tune the classification threshold to optimize directional accuracy.

#### Goal 1 — Classification

**Predict if BTC price will go UP or DOWN in the next minute.**

Label (`up_next_min`):

```
IF (lead(Close, 1) OVER (...) > Close) THEN 1 ELSE 0
```

Features:

- lag\_close\_1
- lag\_close\_5
- volatility\_1min
- volatility\_5min
- rolling\_mean\_5
- rolling\_volume\_5
- streaming\_current\_price

- streaming\_24h\_change

Model:

```
CREATE MODEL btc_price_classifier  
  
OPTIONS(model_type='logistic_reg') AS  
  
SELECT ...
```

---

## Goal 2 — Regression

**Predict the next minute close price.**

Model:

```
CREATE MODEL btc_price_regressor  
  
OPTIONS(model_type='linear_reg') AS  
  
SELECT ...
```

---

## Required Outputs

- **ML.EVALUATE**
  - **ML.EXPLAIN\_PREDICT**
  - **Threshold tuning for classification**
- 

## 4. Dashboard KPIs (Looker Studio or Plotly)

KPIs:

1. **Live BTC Price** (streaming)
2. **Price Change (1 min, 5 min, 1 hr)**
3. **Volatility Index** (rolling std)
4. **Trading Volume Trends**
5. **Prediction vs Actual Price Direction**
6. **Candlestick chart** (OHLC)

Include at least 1 **time-series chart** driven by the streaming table.

---

### 5.1. Challenge: Construct Your Own Prompt

- Prompt: Play devil's advocate for our proposed sentiment analysis pipeline. What is the most prominent risk or potential point of failure in our design? Provide a snapshot of the top three risk and a brief summary of what can be done to mitigate them before we start building?

#### Top 3 Risks and Mitigation Strategies

##### 1. Data Ingestion: API Downtime

This is the most critical risk, as the model relies on **near real-time data** from the CoinCap Live Prices API for short-term predictions.

- **Risk:** The CoinCap Live Prices API may experience downtime or data corruption, leading to **stale or absent real-time features** (like `streaming_current_price` and `streaming_24h_change`) for the BQML model.
- **Strategy:** Implement logic in the Cloud Function to **cache the last successfully fetched data point** locally or in temporary storage and use this cached data if the API call fails.

##### 2. Operational & Cost Management

This risk involves the overhead and potential financial liability of running a high-frequency, two-part pipeline.

- **Risk:** High-frequency Cloud Function execution (every 30 seconds) and the resulting **high-frequency writes** to BigQuery can lead to elevated cloud costs and potential Cloud Function timeouts or throttling.

- **Strategy: Reduce the Cloud Function frequency** from every 30 seconds to **every 1 minute**. This drastically reduces the write volume and execution cost while maintaining a high-frequency stream. Additionally, use **partition pruning** for cost-effective querying of the batch BigQuery table.

### 3. Data Quality & Model Integrity

The quality of the historical batch data is fundamental for training reliable features like lagged returns and volatility indicators.

- **Risk:** The batch historical data may contain quality issues such as **missing timestamps, duplicate rows, or zero-volume rows**, which would corrupt the derived features (`lag_close_1`, `volatility_1min`, etc.) and degrade the BQML model's performance.
  - **Strategy: Enforce the planned data quality checks** during the ETL process, specifically by scripting steps to **remove duplicates and zero-volume rows**, and validating for **missing timestamps** and **monotonic order** before the data is written to the clean, partitioned BigQuery table.
- 

## 5.2. Risk, Ethics, and Governance

- Crypto data is public and non-sensitive
- No PII
- No privacy concerns
- Main risks:
  - API downtime
  - Cloud Function timeout
  - High-frequency writes cost
- Mitigation:
  - Cache data if API fails
  - Reduce function frequency to 1 min

- Use partition pruning
- 

## 6. Roles & Responsibilities (Team)

### Role suggestions:

- **Data Engineer:** GCS, BQ ingestion, Dataflow
  - **ML Engineer:** BQML models, feature engineering
  - **Streaming Lead:** Cloud Function + Pub/Sub
  - **Dashboard Lead:** Looker Studio KPIs
  - **Documentation Lead:** Blueprint, governance, ops runbook
- 

## 7. Deliverables

### Team

- Architecture README
- Governance doc
- 8–10 slides
- Live pipeline demo

### Individual

- Notebook (`Final_Name_analysis.ipynb`)
- DIVE + prompts
- Plotly figure
- Contribution markdown

