CSC490 Assignment 2

Qixiang Fan - 1008847241
Zhuorui(Jack) Jiang - 1008917059
Zihan Zhao - 1010128274
Gabriel Lee - 1007488504

Part 1 - Aspirational Datasets

ChronoStock requires a multi-dimensional data architecture that links quantitative price movements with qualitative market context. Rather than presenting price history in isolation, we synchronize long-term stock performance with financial news and social sentiment. This structured, multi-year approach enables users to understand not only when significant movements occurred, but also what events drove them—revealing recurring patterns such as the impact of earnings releases or regulatory changes and supporting more informed, long-term decision-making.

To enable clear “cause and effect” mapping, we rely on three core datasets:

1. **Historical Stock Price Dataset** – A time-series dataset containing ticker symbol, timestamp, and price (and potentially volume). This dataset serves to identify significant price movements and key time points, which are then used to retrieve corresponding news articles and social media activity.

2. **Financial News Dataset** – A structured repository of financial news articles, where each record includes the publication date, source link, and, ideally, tagged stock identifiers. This allows us to directly associate specific news events with relevant stocks and time periods.

3. **Social Media Dataset (via API)** – A dataset of social media posts collected from one or more platforms, labeled with post date and time. Each entry should include engagement metrics—such as likes, reposts, and favorites—to compute a trending index and surface high-impact discussions relevant to specific stocks.

Together, these synchronized datasets allow ChronoStock to transform historical charts into clear, contextual narratives.

Below are the ideal schemas for these three datasets.

Table 1: Historical Stock Price Dataset Schema

Field Name	Data Type	Description

ticker_id	String	The stock ticker symbol.
timestamp	DateTime	The time interval (e.g., 1-minute or 5-minute marks).
open_price	Float	Price at the start of the interval.
high_price	Float	Highest price during the interval.
low_price	Float	Lowest price during the interval.
close_price	Float	Price at the end of the interval.
volume	Int	Number of shares traded (used to confirm "significance" of a move).

Table 2: Financial News Dataset Schema

Field Name	Data Type	Description
news_id	UUID	Unique identifier for each article.
timestamp	DateTime	Exact release time (UTC) to align with price candles.

ticker_symbols	List[String]	Array of stock tickers mentioned (e.g., ["AAPL", "TSLA"]).
headline	String	The title of the news event.
content_full	Text	Full body text for AI summarization and "NLP training".
article_url	String	Direct link for user verification and "transparency".

Table 3: Social Media Dataset Schema

Field Name	Data Type	Description
post_id	UUID	Unique identifier for the social post.
platform	String	The source platform (e.g., "X", "Reddit").
created_at	DateTime	Timestamp of the post.
associated_tickers	List[String]	Cashtags or keywords identified (e.g., "$NVDA").
post_content	Text	The text of the post/tweet for sentiment analysis.

engagement_score	Integer	Weighted sum of likes, reposts, and comments (the "Trending Index").

Part 2 - Reality Check

Table 4: Reality Check Table

Relevant Item	Description	Commentary
Tiingo	Financial news API	Provides all kinds of data, including stock history and real-time prices, and financial news. However, it only provides financial news up to the past 3 months.
Massive	Financial news API	Provides financial news since 2016. Each entry includes the text, the link, and a sentiment analysis result.
stocknewsapi	Financial news API	Provides individual ticker news, and the news can be filtered by type and sentiment. However, no free API plans are available and the API plan is quite expensive.
yfinance	An open-source Python library to download historical stock prices from Yahoo Finance.	Provides an interface for downloading price history data (Date, Close, High, Low, Open, Volumn). No rate limit constraints.
databento	API for Real-time and historical market data	Provide open, high, low, and close prices and total volume aggregated from trades at 1-second, 1-minute, 1-hour, or 1-day intervals. Unlimited API calls, but charges by usage.
alpaca	API for Real-time and historical market data	Provides real-time stock prices data (with 15-min

		delay) and historical prices over the past 7-years.
data365	Social Media Search API	Serve to extract data from social media networks, such as Facebook, Instagram, X, TikTok, etc. However, this API is very expensive (at € 300/month)
pullpush	Open-source project for querying Reddit Posts	Uses the Reddit API to query forum posts for specific subreddits across a given period. Each entry contains Date, Sore, Author, Body, and the link.

Part 3 - Data-processing pipelines

3.1 Data schemas

Table 5: Stock Price Data Table

Field Name	Data Type	Description	Example
ticker	STRING	Stock symbol (Partition Key)	"AMZN"
date	DATE	Trading date	2024-01-20
open	FLOAT	Opening price of the day	155.2
high	FLOAT	Highest price of the day	157
low	FLOAT	Lowest price of the day	154.8
close	FLOAT	Closing price of the day	156.5
volume	BIGINT	Number of shares traded	45000000

Table 6: Stock News Data Table

Field Name	Data Type	Description	Example
id	STRING (UUID)	Unique identifier for the news event	"a1b2c3d4-e5f6-..."
tickers	STRING	List of related stock symbols	"AMZN|GOOGL"
title	STRING	Headline of the article	"Amazon and Google Announce Cloud Partnership"
published_utc	TIMESTAMP	UTC timestamp of publication	2026-02-18T16:30:00Z
author	STRING	Name of the reporter or agency	"Jane Doe (Reuters)"
description	STRING	Brief summary or lead paragraph	"Amazon Web Services and Google Cloud..."
keywords	STRING	Comma-separated tags or topics	"cloud, partnership, AI"
insights	JSON	AI-generated summary or sentiment data	{"sentiment": "positive", "sentiment_reasoning": "...", "ticker": "UBS"}
url	STRING	Direct link to the source article	"https://finance.yahoo.com/..."

Table 7: Stock post and comment on Reddit Data Table

Field Name	Data Type	Description	Example
type	STRING	Post or Comment	"Post"
date	TIMESTAMP	UTC timestamp of creation	2026-02-18T16:45:00Z
author	STRING	Reddit username	"cvValue"

score	INTEGER	Net upvotes (Up - Down)	25
title	STRING	Title of the thread	"Why I'm bullish on AMZN"
body	TEXT	Full content of the post	"Here is my portfolio"
permalink	STRING	Relative URL to the thread	"/r/investing/comments/..."

Table 8: Final Data Table (For frontend display)

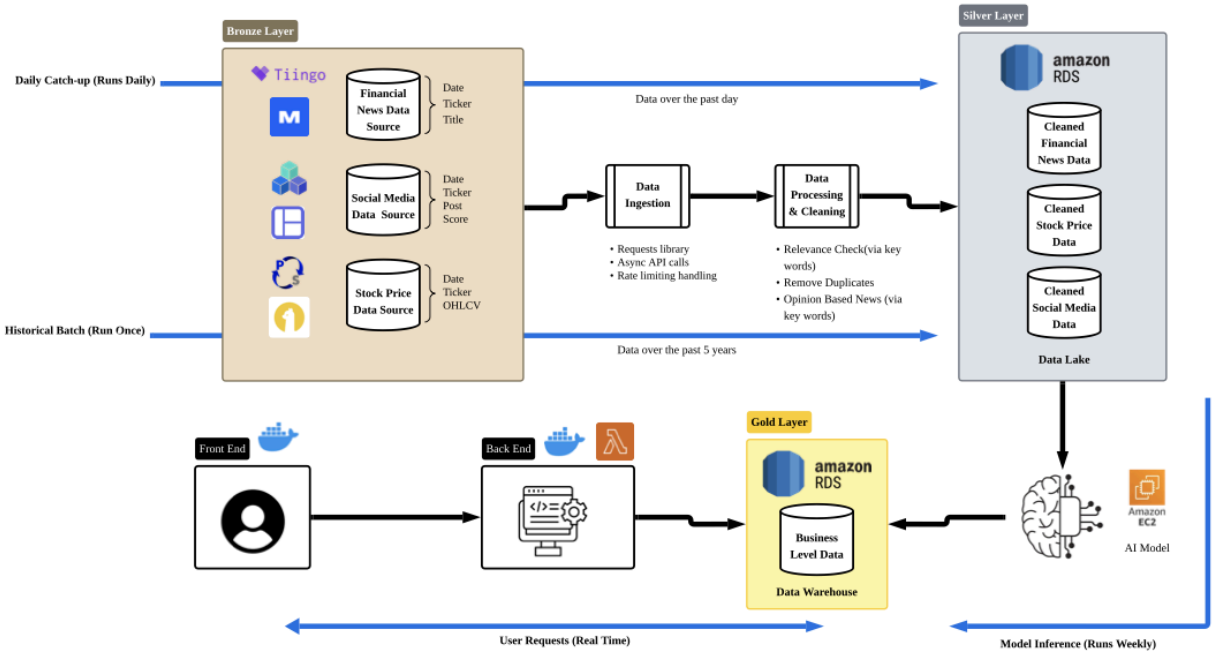
Field Name	Data Type	Description
date	TIMESTAMP	Date of the data point
price	FLOAT	Closing price
volume	INTEGER	Daily trading volume
events	ARRAY	List of significant events for this date

Table 9: Nested Events Data Table

Field Name	Data Type	Description
id	STRING (UUID)	Unique event ID
type	STRING	news/social media
headline	STRING	Short title
sentiment	FLOAT	-1.0 (Neg) to 1.0 (Pos)
relevance	FLOAT	0.0 to 1.0

3.2 Pipeline diagrams

Diagram 1: ChronoStock Data Pipeline



3.3 Pipeline Schedule & Use Cases

Table 10: Pipeline schedule & use case Table

Pipeline Type	Frequency	Description
Historical Batch	Once	Ingested the historical data to populate the initial database and train the ML models.
Daily Catch-up	Daily incremental	Scheduled to run every day at 5:00 PM. It fetches the last 24 hours of news and price data, cleans it, and appends it to the Silver layer to keep the timeline current.
Weekly inference	Weekly	Runs on the past 7 days of data to perform deeper inference tasks, such as generating weekly event summaries and re-evaluating impact scores based on the full week's price context.

3.4 Future Work & Next Steps

With the historical backfill and daily catch-up pipelines now operational, our engineering focus will shift to optimizing latency and improving the semantic depth of our data.

1. Low-Latency Streaming (Real-Time)

- Current Limitation: The system currently relies on a daily batch job at 5:00 PM. This means users cannot see the impact of news events (like an earnings call at 4:05 PM) until the following day.
- Proposed Implementation: We aim to migrate from a purely batch architecture to a real-time event-driven architecture using Apache Kafka or AWS Kinesis.
- News webhooks will push events directly into a message queue.
- A stream processor (e.g., Spark Structured Streaming) will tokenize and clean the data in sub-minute latency before pushing it to the frontend via WebSockets.

2. Advanced NLP & Entity Linking

- Current Limitation: The pipeline uses keyword matching (e.g., searching for "Amazon" or "AMZN"). This method is brittle; it may miss articles that mention "AWS" without explicitly mentioning "Amazon," or falsely include articles about the Amazon rainforest.
- Proposed Implementation: We will integrate a Named Entity Recognition (NER) step using libraries like spaCy or a fine-tuned BERT model.
- This will allow us to detect "Amazon" as an Organization specifically, reducing false positives.
- We also plan to implement a "Sentiment Scoring" transformation step (using FinBERT) to tag each event with a sentiment polarity score (-1.0 to +1.0), enabling users to filter news by "Negative Impact" or "Positive Impact."

3. Data Quality & Dead Letter Queues (DLQ)

- Current Limitation: If a row fails validation (e.g., malformed date or missing source), it is currently dropped or logged to stdout, making it difficult to debug intermittent API failures.
- Proposed Implementation: We will implement a "Dead Letter Queue" for failed ingestion records.
- Malformed data will be automatically routed to a separate storage bucket (e.g. s3://chronostock-error/) rather than being discarded.
- This allows engineers to manually inspect, fix, and replay the failed events without stopping the production pipeline.

4. Data Lake Refactor (S3 + Parquet)

- Current Limitation:The Silver Layer is currently stored in Amazon RDS, which is not optimized for large-scale batch scanning or model training as data volume grows.
- We will implement an S3-based data lake architecture. Raw API responses will be stored in Amazon S3 as immutable JSON objects (Bronze Layer). Cleaned and feature-ready data will then be transformed and written to S3 in partitioned Parquet format (Silver Layer). RDS will be retained strictly as the Gold Serving Layer for low-latency user queries.
- This enables efficient columnar reads, scalable batch inference, and large-model analytics without overloading transactional databases.

Part 4 - Infrastructure as Code Implementation

To ensure that the data pipelines and serving layers are straightforward and maintainable, the infrastructure is implemented using Terraform. This declarative approach allows us to manage our data and compute resources efficiently while maintaining a clear version-controlled audit trail of changes.

4.1 Provisioned Infrastructure and Deployment Strategy

The Terraform configuration provisions the following AWS resources:

- An EC2 instance to run the data pipeline
- An RDS PostgreSQL instance for persistent data storage
- Security Groups to control inbound/outbound access
- An IAM role attached to EC2 for secure AWS access
- An Elastic IP associated with the EC2 instance
- Docker-based deployment with automated daily scheduling via cron

The entire infrastructure can be recreated using:

terraform init

terraform apply -var-file="dev.tfvars"
or
terraform apply -var-file="prod.tfvars"

4.2 Implementation of Multiple Environments

To safely promote changes without risking the production timeline, the infrastructure supports multiple isolated environments (`dev` and `prod`). Example variable files (`dev.tfvars.example` and `prod.tfvars.example`) have been provided in the repository to support different deployment environments.

4.3 Justification for Using a Bootstrap Script

The EC2 instance provisioning steps are handled via a user_data bootstrap script (user_data.sh). This approach is chosen because these steps involve operating system-level configuration and application runtime setup, which are not directly managed by Terraform resources. Terraform is used to define and provision infrastructure components such as EC2, RDS, networking, while the bootstrap script ensures the application environment is automatically configured upon instance launch.

Part 5 - Disaster Recovery Demonstration

The disaster recovery process is demonstrated in the following recorded video:

Video: Part5.mp4

https://drive.google.com/file/d/1bayzdfF2Cp1ZIVthAIhULXT9h3FnklWF/view?usp=sharing