

DRS

C011007 강지호 C144018 조명진

Agenda

0. Simple Information

1. Current Progress

2. Challenges

3. End-of-Semester Plan

Direction

Pattern-Based Stock Search Model:
Draw a Chart → Find Matching Stocks

- ✗ Not a numbers-based investing tool
- ✓ A new stock exploration method centered on chart shapes
- ✗ Not a service only for experienced investors
- ✓ A new investing experience that anyone can use intuitively and visually

Goal

Final Goal: Build and Deploy a Real-Time Graph-Based Stock Recommendation Engine

- Store and maintain segment-based vectors for 3,000–4,000 stocks
- Compare user-drawn sketches with precomputed vectors using FastDTW in real time
- Return Top-K similar stocks within 3 seconds
- Provide a web UI where users can draw patterns and view matching stocks on charts

Approach

- Large-scale automated stock price ingestion
 - NASDAQ API
 - : real-time automatic ticker collection
 - Yfinance
 - : automated OHLCV price data ingestion
- Graph-segment-based pattern comparison
- Shape similarity computed using FastDTW
- Data normalization, missing-value handling, and incremental updates
- FastAPI-based backend + Canvas UI integration

System Design

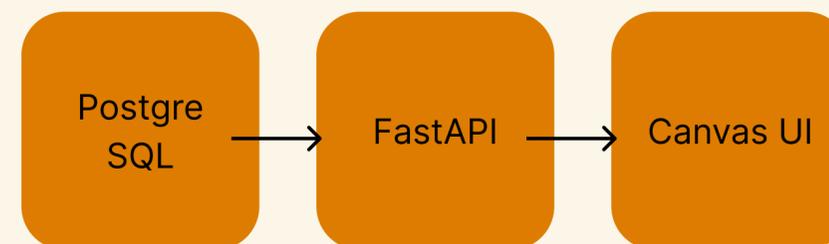
Store price and vector data for 1,000 tickers in PostgreSQL (pgvector)

➡ FastAPI server generates a sketch vector from user input

➡ Query pre-stored ticker vectors from the database

➡ Perform Top-K similarity search using FastDTW / Cosine similarity

➡ Return results to the Canvas UI in real time



Evaluation

1. Accuracy Evaluation

- Input partial segments of the same stock
 - Measure the ratio of cases where the correct stock appears in the recommendations

- Evaluate performance using traditional chart pattern examples as a comparison baseline

2. Performance Evaluation

- Measure FastDTW search speed (target: under 3 seconds)
- Run scaling tests to observe how response time changes as the dataset grows

3. Service Evaluation

- Conduct UI testing for usability and clarity
- Collect feedback from real user sketches and interaction patterns

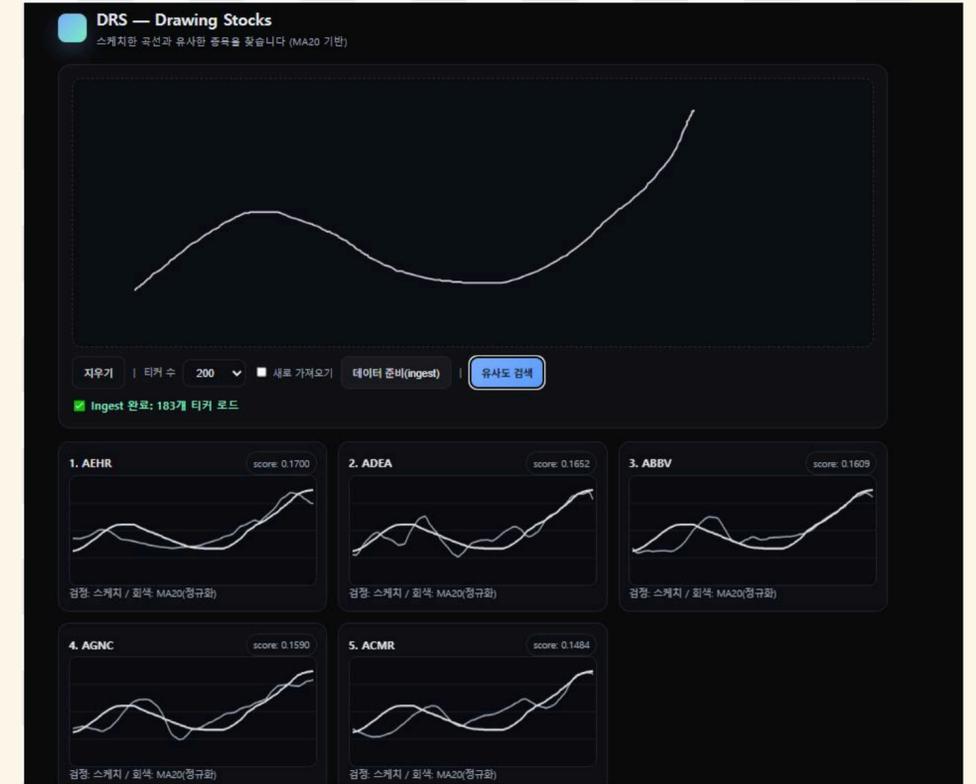
The screenshot shows a database query interface with two tables. The top table lists stock profiles with columns: id, ticker, ma_type, segment_start, segment_end, and vector. The bottom table lists stock profiles with columns: ticker, trade_date, open, high, low, close, and adj_close.

id	ticker	ma_type	segment_start	segment_end	vector
822	822 AAGG	MA20	2025-01-16	2025-05-27	(0.87815744, 0.0)
823	823 AAGG	MA20	2025-01-17	2025-05-28	(0.76833266, 0.0)
824	824 AAGG	MA20	2025-01-21	2025-05-29	(0.59920955, 0.0)
825	825 AAGG	MA20	2025-01-22	2025-05-30	(0.5288554, 0.4)
826	826 AAGG	MA20	2025-01-23	2025-06-02	(0.4372804, 0.4)
827	827 AAGG	MA20	2025-01-24	2025-06-03	(0.40303, 0.342)
828	828 AAGG	MA20	2025-01-27	2025-06-04	(0.35172752, 0.0)
829	829 AAGG	MA20	2025-01-28	2025-06-05	(0.27034977, 0.0)
830	830 AAGG	MA20	2025-01-29	2025-06-06	(0.12786455, 0.0)
831	831 AAGG	MA20	2025-01-30	2025-06-09	(0.1691551, 0.2)
832	832 AAGG	MA20	2025-01-31	2025-06-10	(0.25139004, 0.0)
833	833 AAGG	MA20	2025-02-03	2025-06-11	(0.38882896, 0.0)
834	834 AAGG	MA20	2025-02-04	2025-06-12	(0.4972565, 0.6)
835	835 AAGG	MA20	2025-02-05	2025-06-13	(0.6880775, 0.8)
836	836 AAGG	MA20	2025-02-06	2025-06-16	(0.8502929, 0.9)

```

1 def dtw_distance(a: np.ndarray, b: np.ndarray) -> float:
2     try:
3         d, _ = fastdtw(a, b)
4         result = float(d)
5         return result if np.isfinite(result) else 0.0
6     except Exception as e:
7         logger.warning(f"DTW calculation failed: {e}")
8         return 0.0
9
10 def cosine_sim(a: np.ndarray, b: np.ndarray) -> float:
11     try:
12         # NaN 체크
13         if np.any(np.isnan(a)) or np.any(np.isnan(b)):
14             logger.warning("NaN detected in cosine_sim input")
15             return 0.0
16
17         norm_a = norm(a)
18         norm_b = norm(b)
19
20         # Zero vector 체크
21         if norm_a < 1e-10 or norm_b < 1e-10:
22             return 0.0
23
24         result = float(np.dot(a, b) / (norm_a * norm_b))
25         return result if np.isfinite(result) else 0.0
26     except Exception as e:
27         logger.warning(f"Cosine similarity calculation failed: {e}")
28         return 0.0

```



DataBase

Store 1,000 stock profiles, daily price data, and graph vector representations

Server

Compute similarity using FastDTW and Cosine functions

Connect the database with the local server

Data-related

1. API Rate Limits & Request Failures
 - Adjust batch size
 - Add retry logic with backoff and sleep intervals
 - Reprocess failed tickers later
2. Data Quality Issues
 - Remove 0 / NaN values and filter out invalid tickers
 - Automatically exclude stocks with no trading activity for the past N days
 - Run data validation queries/scripts regularly

System-related

1. Automation / Scheduling Issues
 - Use a test mode like python
ingest_prices.py --dry-run
 - Automatically print summary logs after each run
2. Performance Bottlenecks
 - Apply index tuning for faster queries
 - Cache frequently used segments and preprocessed results

Similarity-related

1. Ambiguity in Defining Ground Truth
 - Use traditional chart patterns
+ a manually labeled stock set as the evaluation baseline
2. User Sketches Being Too Rough or Inconsistent
 - Normalize vertical scale and horizontal length
 - Anchor the starting point at 0 and resample the sketch to a fixed length

End-of-Semester Plan

1. Implement a similarity matching system between approximately 3,300 NASDAQ stocks and the user's input data (based on a 20-day window).
2. Automated stock data updating
3. Target accuracy:
90%+ using traditional chart-pattern benchmarks
 - Ability to adjust the search window beyond the default 20-day period
 - Search functionality based on traditional chart patterns