Stock Market Analytics Documentation

Mihail Gjorgjievski 221242
Matej Josifov 221114

1. `Analysis Service`

Purpose:

Analyzes stock price trends using moving averages and predicts future closing prices using linear regression. It also generates trading signals.

Key Features:

- Retrieves stock data from the database (`StockData`).
- Computes **Simple Moving Averages (SMA)** (5-day and 15-day) to determine trends.
- Predicts the next day's closing price using **linear regression**.
- Generates a **candlestick chart** using `plot_candlestick_chart`.
- Stores the analysis results in `StockSignal`.

Functions:

`analyze_stock(symbol: str) -> None`

- **Inputs:** Stock symbol (e.g., 'AAPL').
- **Outputs:** Prints trend, action (BUY/SELL/HOLD), and predicted price.
- **Error Handling:** Catches exceptions and prints error messages.

---

2. `Ingestion Service`

Purpose:

Fetches stock data from Yahoo Finance and stores it in the database.

Key Features:

- Retrieves **historical stock data** using `yfinance` and stores it in `StockData`.
- Fetches **real-time stock data** at 1-minute intervals.
- Scrapes the **S&P 500 stock symbols** from Wikipedia.

Functions:

`fetch_historical_data(symbol: str, period='5y', interval='1d')`

- **Inputs:** Stock symbol, time period, and interval.
- **Outputs:** Stores historical stock data.

- **Error Handling:** Skips rows with errors.

  `get_sp500_symbols() -> list[str]`

- **Outputs:** A list of S&P 500 stock symbols.

  `store_sp500_symbols() -> None`

- **Functionality:** Saves the S&P 500 stock symbols to the database.

  `fetch_all_realtime_data() -> None`

- **Functionality:** Fetches and stores **latest real-time stock data** for all tracked stocks.

---

3. `Metrics`

Purpose:

Identifies the **top gainers and losers** in stock prices based on daily percentage changes.

Key Features:

- Retrieves the most recent two stock prices per stock.
- Computes the **percentage change** in stock price.
- Identifies the **top 5 gainers and top 5 losers**.

Functions:

`get_top_movers(limit=5) -> tuple[list, list]`

- **Outputs:** Two lists: top gainers and top losers (sorted by percentage change).
- **Error Handling:** Skips stocks with missing or invalid data.

---

4. `Visualization Service`

Purpose:

Generates **interactive candlestick charts** using Plotly for stock price visualization.

Key Features:

- Retrieves stock data for a specified number of days.
- Uses **Plotly** to generate a **candlestick chart**.
- Returns the chart as an **HTML div**.

Functions:

`plot_candlestick_chart(symbol: str, days=366) -> str | None`

- **Inputs:** Stock symbol, number of days.
- **Outputs:** HTML `<div>` containing the candlestick chart.
- **Error Handling:** Returns `None` if no data is found.

---

4. Django Management Commands for Stock Data Ingestion & Analysis

1. Ingest_all_data.py

Purpose:

This command triggers the ingestion of historical stock data for all S&P 500 stocks in the database using Celery.

Implementation:

- Uses `batch_ingest_all_symbols.delay()` to queue data ingestion tasks asynchronously.

- Ensures historical data for all stocks in the system is collected.

2. Ingest_sp500.py

Purpose:

Fetches and stores the latest S&P 500 stock symbols in the database.

Implementation:

- Calls `store_sp500_symbols()` to scrape and save the latest S&P 500 symbols.

- Ensures the system tracks the correct companies.

3. Reanalyze_all.py

Purpose:

Re-analyzes all stocks in the database, updating their trading signals and chart visualizations.

Implementation:

- Iterates through all stock symbols in the database.
- Triggers `analyze_stock_task.delay(symbol)` for each stock, running the analysis asynchronously via Celery.

4. `Celery Tasks for Stock Data Ingestion & Analysis`

1. `plot_candlestick_chart_task(symbol, days=90)`

Purpose:

Generates a candlestick chart for a given stock symbol over a specified period.

Usage:

This task asynchronously creates and stores a candlestick chart, allowing visualization without blocking the main thread.

Implementation:

- Calls `plot_candlestick_chart(symbol, days)`, which generates and saves the chart.
- Returns a confirmation message indicating the visualization is complete.

---

2. `ingest_historical_task(symbol)`

Purpose:

Fetches historical stock data for a given symbol and stores it in the database.

Usage:

This task allows periodic ingestion of stock data without blocking Django's main execution flow.

Implementation:

- Calls `fetch_historical_data(symbol)`, which retrieves and saves past stock prices.
- Returns a confirmation message upon successful execution.

---

3. `batch_ingest_all_symbols()`

Purpose:

Queues historical data ingestion tasks for all stock symbols in the database.

Usage:

Instead of fetching data sequentially, this function enqueues ingestion tasks for all stocks, improving efficiency.

Implementation:

- Retrieves all stock symbols from the database.
- Iterates through the symbols and triggers `ingest_historical_task.delay(symbol)`, ensuring asynchronous execution.
- Returns a message confirming the number of tasks queued.

---

4. `ingest_realtime_all_stocks()`

Purpose:

Fetches real-time stock data for all tracked stocks in the database.

Usage:

This task runs periodically to keep stock data up-to-date.

Implementation:

- Calls `fetch_all_realtime_data()`, which fetches and stores the latest market data.
- Returns a success message upon completion.

---

5. `analyze_all_stocks()`

Purpose:

Triggers stock analysis for all symbols in the database.

Usage:

This task helps refresh technical indicators and signals periodically.

Implementation:

- Iterates through all stock symbols in the database.
- Calls `analyze_stock(stock.symbol)`, which updates technical signals.

---

6. `analyze_stock_task(symbol)`

Purpose:

Performs technical analysis on a specific stock.

Usage:

This task is triggered whenever a stock needs re-analysis (e.g., after new data ingestion).

Implementation:

- Calls `analyze_stock(symbol)`, which updates indicators and trading signals.

4. `Stocks App Views`

1. `landing_page(request)`

Purpose:

Displays the homepage where users can view available stocks and the top-performing gainers/losers.

Behavior:

- Fetches all available stock symbols from the database and orders them alphabetically.
- Retrieves top gainers and losers using `get_top_movers()`.
- If the user selects a stock symbol via a form, they are redirected to the stock's dashboard page.

Template:

- `stocks/landing.html` (Displays stock list and market movers.)

Parameters:

- `request`: HTTP request object.

Returns:

- Renders the `landing.html` template with the list of stocks and top market movers.
- If a stock is selected, redirects the user to its dashboard page.

---

2. `company_dashboard(request, symbol)`

Purpose:

Displays an individual stock's dashboard, including:

- Basic stock information
- Trading signals (trend & action)
- A candlestick chart (if available)

Behavior:

- Retrieves the `Stock` object for the given `symbol` or raises a 404 error if not found.
- Fetches the latest `StockSignal` associated with the stock.
- Extracts the candlestick chart HTML (if available).
- Determines colors for trend and action indicators.
- Renders the `stocks/dashboard.html` template with all relevant stock data.

Template:

- `stocks/dashboard.html` (Displays stock data, technical signals, and candlestick chart.)
- `request`: HTTP request object.
- `symbol` (str): Stock ticker symbol to fetch details for.

Returns:

- Renders the `dashboard.html` template with stock details, signals, and visualization.