Stock Market Data Collection System

This project collects NSE stock exchange data using external APIs, stores it in a database, and displays it on the website. The first phase of the project does not have a selection option. Instead, we dump the data on the website. We have two pipelines for collecting data, one for collecting historical data and one for collecting daily data.

Our plan is to collect NSE stock exchange data for 365 days, with intervals of 1 minute for the first 30 days, 2 minutes for the next 30 days, and 1 hour for the remaining 10 months. There are some limitations provided by the Yahoo Finance API that result in the collection interval time not being consistent. The main aim is to collect maximum available data.

# ## Yahoo Finance API

Yahoo Finance is an open source and free to use it. It is also simple, quick to learn & install, and provide the high granularity of data (1min, 2min, 5min and so on...). The key benefit is it provides the data directly pandas dataframes or series. In this module we will be using yahoo finance API to collect history data.

## ### API Details

Yahoo finance provide variety of APIs to collect different information and in different format. As we are collecting only basic information related to the stock like Adj. Close, Close, High, low etc of the selected company as per given in configuration files. (Discussion on configuration will be covered later)

**Interface**

Function - yf.download(tickers,…) -> pd.dataframe

**Key Parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Description | Default | Valid Values |
| tickers: str, list | List of tickers to download (company symbol for which you want to retrieve data |  |  |
| period: str | either set period parameter or use start and end. Don't mix with start & end | max | 1d, 5d, 1mo, 3mo, 6mo, 1y, 2y, 5y, 10y, ytd, max |
| interval: str | Intraday data cannot extend last 60 days  \* 1m - max 7 days within last 30 days  \* up to 90m - max 60 days  \* 60m, 1h - max 730 days (yes 1h is technically < 90m but this what Yahoo does) | 1d | 1m,2m,5m,15m, 30m,60m,90m,1h,1d, 5d,1wk,1mo,3mo |
| start date: str | Download start date, inclusive. | '1900-01-01' (if period=None) | 'YYYY-MM-DD' string, \_datetime, or epoch |
| end date: str | Download end date, exclusive | '1900-01-01' (if period=None) | 'YYYY-MM-DD' string, \_datetime, or epoch |

Important points to Note –

* TimeZone handling –
  + Case 1 – Fetching data from the same Stock Exchange.
    - If we fetch data only by date (start\_date & end\_date), but don't provide time zone information, the API considers the country's time zone where the Stock Exchange is located.
  + Case 3 – When we provide or add the time zone information with the start\_time and end\_time, it shows the data based on the time zone selected., end\_date, it shows data in

\*\* Note – Figure out if start and end date used timezone information. What timezone it assume if we does not share timezone information.

For more information and parameters refer - <https://github.com/ranaroussi/yfinance/wiki/Tickers>

**Return Value**

The API returns pandas dataframe.

|  |  |
| --- | --- |
| Field or Value | Description |
| Adj. Close | Adjusted Closing Price - amends a stock's closing price to reflect that stock's value after accounting for any corporate actions. |
| Close | closing price is the raw price or cash value of the last transacted price in a security before the market officially [closes](https://www.investopedia.com/terms/c/close.asp) for normal trading. |
| High | The high price refers to the highest trading price at which a stock was traded during a given period, typically a trading day. |
| Low | The low price is the lowest trading price at which a stock was traded during a specific period, such as a trading day. |
| Open | The open value, also known as the opening price, is the first trading price at which a stock is traded when the market opens for a particular trading day. |
| Volume | Volume refers to the number of shares of a particular stock that have been traded during a given period, such as a day or a specific timeframe. |
| Datetime | Date or datetime depends on the interval of collection. For day wise it is only date and for hourly collection it is datetime. |

**Considerations and Logic**

As we collect the data, we should know the size of the data that we will collect. In this project, we have divided NSE stocks data collection into two parts.

* **History data collection** – It is a one-time collection of stock market data on a day-by-day basis. In order to collect NSE stock exchange history data at granular levels, Yahoo Finance API download () has certain restrictions. Example - if we want to collect 1min data for each stock, we can only retrieve for the last 30 days and each call can fetch 7 days data at a time, or 1 hour data we can & anything intraday (interval <1d) only for the last 60 days.

Use Case -

* In this project we are collecting the stock information of 10 companies listed in NSE stock Exchange.
* In 7 days , NSE stock exchange open for 5 days from 09:15 to 15:30 in a day. It comes out to be 6 hours 15 mins.

Data Size –

* + - Assumption - Stock information for 10 companies ( All 10 companies should be listed in a same Stock Exchange).
      * + If we fetch data for each minute for a day ( 6 hr 15 mins) then

6 (hr) \* 60(min) + 15(mins) = 375 min

* + - * + For 5 days (in 7 days stock market opens for 5 days only), hence it will be :

375(min) \* 5(days) = 1875.

* + - * + No. of records in a single request will be 1875.
        + For 10 Companies, it will be

1874 X 60 (6 fields x 10 companies)

* + - * + The size of the dataframe will be

~ 1MB (based on the multiple test.

* + - * + For reference created the no. of companies and size chart for collection. Assumption collection interval 1 min for last 7 days.

|  |  |  |
| --- | --- | --- |
| No. of Companies | Dataframe size (kb) | Matrix Size (row X col) |
| 10 | 893 (900 kb) | (1874,60) |
| 9 | 805 | (1874, 54 |
| 8 | 717 | (1874,48) |
| 7 | 630 | (1874,42) |
| 6 | 542 | (1874,36) |
| 5 | 454 | (1874,30) |
| 1 | 102 | (1874,6) |

Note :- Based on the above data, we can say that collecting 1 company data with 1874 records the size will be approx. 100KB. And for 10 companies it will be 1MB.

Rate Limitation –

There are some limitations with Yahoo Finance API:

Using the Public API (without authentication), you are limited to 2,000 requests per hour per IP (or up to 48,000 requests a day).

Could not find any official documentation for the same. It is based on the available information. Sleep is recommended to avoid IP being blocked. May be using Sleep(300) will be good.

In our case, we will collect data for 1 year. Here are the calculations for how many requests we need to send. Calculation will help to give information about how much time will be required to collect history data for 1 year.

* For first 30 days, we will have 5 requests. ( data granularity 1 min)
* For next 30 days, we will have 1 request. (data granularity 2 min)
* For next 10 months we will have 1 request. (data granularity 1hr)

Maximum Data size in above request will be 1.5MB. **Total size will be less than 10 MB.**

Basic Logic –

To collect history data, we need to run a piece of code ( as a job or a process ) 5 to 10 times with a sleep of 3-10 min. In the first 5 calls,  start and end date or time of collection window ( 7 days window) will change. For 2 min and 1hr granularity level collection, we need only one call each. After every request, the database will be updated. Once the collection is done we can stop calling the process or job or come out of the code.

To implement we can use below to approach –

* Using systemctl timer functionality:
  + Advantages:
    1. Provides better separation of concerns, as scheduling logic is handled by the system.
    2. Ability to adjust the time interval without modifying the code.
    3. Automatic triggering of the process based on the specified timer configuration.
  + Disadvantages:

1. Requires additional setup and configuration of systemctl timers.
2. If there are multiple API calls that need to be made sequentially, separate timers may be needed for each call, which could be cumbersome.

* Handling the sleep in the code:
  + Advantages:
    1. Simple implementation, as the sleep duration is directly handled within the code.
    2. No external setup or configuration is required.
  + Disadvantages:
    1. Requires code modification and possibly recompilation or restart to change sleep duration.
    2. Relies on the application's execution to ensure accurate timing, which can be affected by factors such as system load or other code execution delays.
    3. Additional effort is needed to handle state tracking and recovery after a crash or restart.

Conclusion -

For history collection, we can use a single API call and do not anticipate complex scheduling or frequent changes in timing. Handling sleep in the code (approach 2) may be a simpler and easier approach.

* **Regular collection –** A batch process will run once per day during the week to gather NSE stock data. This will be done with granularity of an hourly or 30-minute interval depending on the user. It is possible to configure the interval (granularity of data collection). At a time, we should be able to collect data for 10 different companies listed on the same stock exchange. These users can change using a configuration file.

We will use systemctl timer functionality, to run the code every day once at a specific time. Once we collect the data, we will append it to the database and display it on the Webpage.

**Fault Tolerance or Error Recovery mechanism**

This section of the project talks about how to deal with unexpected network problems, code crashes, or any situation that may impact the flow of collection. In case of any intermediate issue occurring during history collection or regular collection of stock data we should start collection from where we stopped (last successful collection). Such situations should be handled by creating a Job Table.

Job table (SQLite Database)

|  |  |  |  |
| --- | --- | --- | --- |
| collection\_type | last\_successful\_fetch | no\_records | Description |
| History | %Y-%m-%d %H:%M:%S | Number | last\_successful\_fetch – Contain last successful write on basic\_stock\_details table. |
| Regular | %Y-%m-%d %H:%M:%S | Number | last\_successful\_fetch – Contain last successful write on basic\_stock\_details table. |

**History Data Collection** - At Job table creation, we must fill the last\_successful\_fetch field with the "current timestamp" value. After each successful write or update in the database, we should update the Job table with last\_successful\_fetch and no\_records. last\_successful\_fetch should be replaced with the start\_datetime collection and no\_records.

**Regular Data collection** - At Job table creation, we must fill the last\_successful\_fetch field with the "current timestamp - a day" value. After each successful write or update in the database, we should update the Job table with last\_successful\_fetch and no\_records. last\_successful\_fetch should be replaced with the start\_datetime collection and no\_records.

In case any exceptions or any issue occurred during the stock data collection, we can restart the service. At each start of the service, we should first check the data in the Job table. Based on the job table data, we should start collecting data. This will ensure that we do not recollect the same data again.

# ## SQLite Database

In this project we will be using SQLite as database which will be used to store the collected data.

Table – Basic stock data collection

Schema Definition

|  |  |  |
| --- | --- | --- |
| Field | Type | Corresponding Value |
| company | Text | tickers |
| timestamp | Text | Datetime |
| date | Text | Computed from datetime |
| high | Real | high |
| low | Real | low |
| close | Real | close |
| Volume | Integer | volume |
| adjusted\_close | Real | Adj close |

\*\* Explore difference between SQL queries when we store Datetime and as Text. – To be done later.

## ### Schema

basic\_stock\_data(

company TEXT,

timestamp DATETIME,

adj\_close REAL,

high REAL,

low REAL,

close REAL,

open REAL,

volume INTEGER,

PRIMARY KEY (company,timestamp)

)

# ## UI

TBA

# ## Config\_file

To help users to choose the collection interval or collection time or company name, we have provided the configuration files.

It is .ini file which store some common configuration settings as shown on below table. To parse the data from the configuration file we are using “configparser” library and uses read and get function.

|  |  |  |  |
| --- | --- | --- | --- |
| Section | Option | Default value | Range |
| stocks | name | TCS, SBI, ICICIBANK | It contains valid company name as per stock exchange market. At max it can have 10 company names. |
| collection | timezone | 5:30 | Any valid time zone value |
| collection | start\_datetime |  |  |
| collection | end\_datetime |  |  |
| collection | interval | 1h | 1m,2m,5m,15m,30m,60m,90m,1h,1d,5d,1wk,1mo,3mo |
| database | db\_path | ./database/stockdataa.db | Any valid path |

# ## Unit Test Cases

TBA

# ## Benchmark

In this project we are going to do the benchmark comparison for 2 scenarios to evaluate the performance of the code.

1. We collect the 10 companies stock data in 10 different iterations (1 company in 1 iteration), process the data and store in the Database.
2. We collect the 10 companies stock data in a single iteration, process the data and store in the Database.

# ## High level Diagram

