Stock Market Data Collection System

# ## Synopsis

This project gathers NSE stock exchange data using external APIs, stores it in a database, and displays it on the website. We have two pipelines for collecting data, one for collecting historical data (executed manually at the beginning of the application) and one for gathering daily data (It is based on systemd - service and timer units).

Our plan is to gather NSE stock exchange data regularly (Daily Data Collection) as well as one-time data collection (Historical Data Collection). During the history data collection, we will collect 730 days of data. The first 30 days of data will be collected at a 1 minute interval, the next 30 days at a 2 minute interval, and the rest at a 1 hour interval. This is because 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 the maximum.

# ## 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 – User doesn’t provide time zone information
    - If we fetch data only by date (start\_date & end\_date), but user don't provide time zone information, the API considers the country's time zone where the Stock Exchange is located.
  + Case 2 – User provided time zone information
    - We collect the data based on the date and time with time zone information provided by the user.

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. |

**Size Chart based on collection per call –**

Table captured the size information for each fetch.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of Symbols | Size (kb) (Dataframe) | No. records Shape(row X col) | time range | time interval |
| 25 | 4000 | 3200 | 670 days | 1 hr |
| 25 | 3000 | 2255 | 30 days | 2 min |
| 25 | 2500 | 1874 | 7 days | 1 min |
| 10 | 893 (900 kb) | (1874,60) | 7 days | 1 min |
| 9 | 805 | (1874, 54 | 7 days | 1 min |
| 8 | 717 | (1874,48) | 7 days | 1 min |
| 7 | 630 | (1874,42) | 7 days | 1 min |
| 6 | 542 | (1874,36) | 7 days | 1 min |
| 5 | 454 | (1874,30) | 7 days | 1 min |
| 1 | 102 | (1874,6) | 7 days | 1 min |

**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, for history collection we can collect with only 7 requests. Hence rate limitation will not impact us.

# ## Project Blueprint: Understanding the Fundamental Logic and Flow

**This sections talks about the basic flow and logic of the project. It all covers some technologies used in the project.**

In this project we are using Yahoo Finance API to collect the basic information of NSE stock exchange. Once we fetch the data, we store the data in the SQLite database. Same data we populate on the Webpage. We also provide some filtering options on Website. For Web development we have used Flask (web framework) and HTML to design UI interface. We have provided REST based API for UI to communicate with the Web server.

**History Collection**

History collection is a one-time collection which will be run manually at the start of the project. As part of the history data collection process, we will collect 730 days of data for different granularity levels. Only 7 requests are needed to fetch the maximum amount of stock data (25 companies available on the NSE Exchange). The first 30 days of data will be collected at a 1 minute interval, the next 30 days at a 2 minute interval, and the rest at a 1 hour interval. This is because Yahoo Finance API limitations cause inconsistent collection interval times.

We have used different technologies and packages like Python, Yahoo Finance, Pandas Dataframe, SQLite database, configparser, datetime module and logging. A predefined list has been created with the number of days and intervals to calculate the start, end date and intervals (granularity of data). These parameters are then used to call yf.download() to collect stock data. Once the data is fetched, we store it in the SQLite database.

**Regular collection**

The approach described involves a regular collection process to collect stock data at a specified interval, once a day at 18:00 IST. The process is divided into two parts: the "daily\_job\_collection" service and the execution of the job created by this service.

The use of systemd to start the "daily\_data\_collection" service at a scheduled interval is a common practice for managing and automating processes in a Linux environment. When the "daily\_data\_collection" script is triggered, it creates a job and stores it in the Job table. The Job table is used to track information related to the job, such as the job ID, collection start time, successful fetch time, job status, and the number of records.

Storing the job details in a table allows for easy retrieval and management of the information. By capturing the collection start time, the script ensures that data collection begins at the specified time. The successful fetch time can be updated once data collection is completed successfully. The job status field provides information about the current state of the job, whether it is pending, in progress, or completed. Finally, the number of records logged indicates the progress and quantity of data collected.

To implement regular collection, we are using Python, Systemd, SQLite.

Job table (SQLite Database)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| job\_id | collection\_start\_date | successful\_fetch\_time | job\_status | | num\_records |
| string | %Y-%m-%d %H:%M:%S | %Y-%m-%d %H:%M:%S | | string | number |
| string | %Y-%m-%d %H:%M:%S | %Y-%m-%d %H:%M:%S | | string | number |

1. job\_id – Job\_id is a string data type having incremental values with prefex as H or D for history or daily collection type.
2. collection\_start\_time – date when data collection should start.
3. successful\_fetch\_time – Time logged when the job is successfully completed.
4. job\_status – pending or done. When a job is logged the status will be in the pending state. When a job is successfully completed the status will be done.
5. num\_recordsNo. of records logged.

**Webpage –**

<TBA>

# ## 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

