Buzz S&P500

Cheng Chen
Ruikang Liu
Puran Zhang
Wenting Zhao

Introduction and Project Goals

Nowadays we have seen plenty of stock trading/research apps/sites around the internet but there is no place for us to keep track of only stocks in SP500, the most important index of large-cap growth stocks, and dive deep into the sectors and stocks in it. Our Buzz S&P web app is designed for people who are interested to know the detailed composition of the SP500 index and the daily and historical movements of all these stocks, which should be a useful resource for entry-level investors who are interested in this topic.

The website will offer an overview of the SP500 composition and provide insights about the performance of different sectors and different stocks, you can click into each stock to review its historical performance in any time intervals in the past several years, the key metrics that demonstrate its financial performance and operational performance. There will also be some benchmarking algorithm that shows how this stock performs compared with the overall market average and the sector average.

Our group members include Cheng Chen, Ruikang Liu, Puran Zhang, and Wenting Zhao.

Technology

Our project used React as our front-end framework and Node.js as our back-end framework. We stored our data in an AWS server and used MySQL to build queries in the back end to retrieve data we need for calculation or display on the pages.
Data processing: Python
Frontend/Backend: React/Node.js
Database: AWS RDS MySQL

Data

We used Yahoo Finance API as our data source, and extracts all historical stock data of SP500 stocks for the past three years and populated them into our database in AWS.

Yahoo Finance API:
Description: API that contains stock price information and company information.
Link: https://github.com/ranaroussi/yfinance

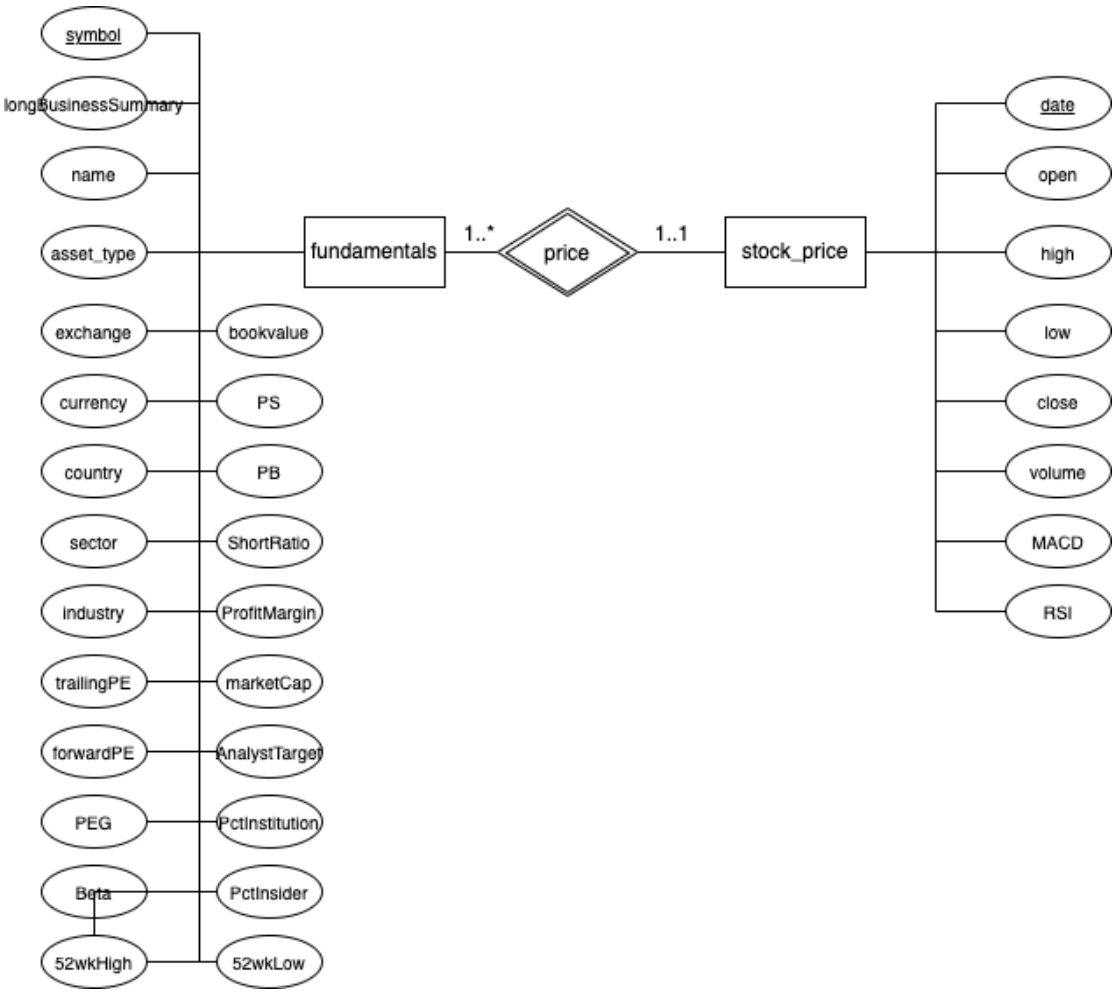
We have done a careful and detailed screening of the data. This data has a few tables that can be used across the board. However, it is not clean and contains duplicated variables. As an open-sourced version, this data governance of this tool is not well implemented. Therefore, we have to

evaluate the variables and do some cross-check with data from other sites so that we can confirm the availability as well as sanity. Initially, we are using AlphaVantage which is more commercially and as a result, it is better in terms of quality. We choose to move away from it because of the high cost of such usage.

We also calculated some fields (technical indicators) directly in our python script as it will complicate the processes if we have to find the data source. It is a better approach as we can control the logic and not create additional dependencies on the data quality and data availability. It is also more maintainable in the long run.

Database

We built two tables, one for stock price metrics, which includes daily prices for all SP500 stocks over the past three years, and another one for stock fundamental data, which includes company info and financial metrics, etc. The price table contains more than 1,200,000 instances and the fundamental contains around 500 instances. ER Diagram is as below (attributes of each table are a subset version):



Description of System Architecture

Below are descriptions of our Buzz S&P500 web pages with information about what the respective pages accomplish:

Home Page:
Our home page provides some up-to-date summary information of stock markets as a snapshot. Showing S&P500, Dow, Nasdaq these three major market indices at the top of the page brings a quick overview of the market to our investors. Next, it displays the stock price change trend overtime via a line graph, where the user could select different date ranges by dragging the slider below the chart. Also, when the user moves their mouse to a point in the graph, it will immediately show the close price for that trading day. Additionally, there is a Sectors chart at the bottom to share the percentages of both 1-day and 10-day price change at various sectors.

Stock Page:
Our stock detail page presents a profile of a certain stock over a specific period with price/volume trendlines. We also provide a brief introduction of the underlying company with several key metrics of fundamentals, which contribute to the company's value or worth as a business. Investors could also take a look at the price details over time in the chart at the bottom with customized sort criteria.

SearchStocks Page:
Our SearchStocks page as a stock screener provides a tool that allows investors and traders to quickly sort through over five hundred available stocks according to the investor's own criteria. By selecting a set of parameters, this page makes investors be able to find those stocks which fit their own methodologies and benefit both longer-term and short-term traders. Furthermore, it will direct investors to the stock detail page by clicking a certain ticker name in the chart.

MarketView Page:
Our MarketView page presents users with a quick market view of the S&P 500 stocks market on the latest trading day. We will see the top 5 gainers, the top 5 losers, the top 5 surprises of stocks with the highest max dropdown (amplitude), the top 5 trending stocks with the highest turnover, and the progress bars to show how many gainers and losers today. Furthermore, we also get a market view depending on the sectors and industries. We will see the top 5 winners of sectors, the top 5 winners of industries, the top winner stocks in the best-performing sector, the top winner stocks in the best-performing industry, and the progress bars to show the statistic of winners and losers in each block so that users could find the alpha (out-performing stocks).

Queries

For the stock detail page:

The most sophisticated query is the Stock_Outperformance Query:

```
WITH target AS (
        SELECT p.Symbol, f.Sector, p.Change,p.Volume, AVG(p.Change)
OVER() AS avgChange, AVG(p.Volume) OVER() AS avgVolume
        FROM Price p JOIN Fundamentals f ON p.Symbol = f.Symbol
        WHERE Date = (
```

```
            SELECT MAX(Date)
            FROM Price
        )
    ),
    compute AS(
        SELECT f.Sector, AVG(p.Change) AS sectorChange,
AVG(p.Volume) AS sectorVolume
        FROM target p JOIN Fundamentals f ON f.Symbol = p.Symbol
        group by f.Sector
    )
    SELECT t.Change, t.Volume, t.Change - cp.sectorChange AS
beatBySector, t.Volume - cp.sectorVolume AS beatVBySector,
t.Change-t.avgChange AS beatByAll, t.Volume-t.avgVolume AS
beatVByAll
    FROM target t JOIN compute cp ON t.Sector = cp.Sector
WHERE t.Symbol = '${symbol}';
```

This query is used to calculate for each stock, how did it perform in price movement and inactive trading volume compared with the average of the overall market and average of stocks in the same sector it belongs to. With this query, we show these metrics in the benchmarking section in the stock detail page when a user checks in to a specific stock.

For the landing page:

The most advanced query is the sector performance table.

```
 WITH a AS (
     SELECT symbol, date, p.close, p.change,
            ROW_NUMBER() over (PARTITION BY symbol ORDER BY date
desc) as rk
       FROM Price AS p
      WHERE DATEDIFF((SELECT MAX(Date) FROM Price), date) <= 20
 ), b AS (
     SELECT a1.symbol,
          a1.change AS d1_change,
          (a1.close-a2.close)/a2.close AS d10_change
       FROM a AS a1
      INNER JOIN a AS a2
         ON a1.symbol = a2.symbol
        AND a1.rk = 1
        AND a2.rk = 11
)
SELECT f.sector,
      CONCAT(ROUND(100 * AVG(b.d1_change), 3), '%') AS d1_change,
      CONCAT(ROUND(100 * AVG(b.d10_change), 3), '%') AS d10_change
 FROM Fundamentals AS f
INNER join b
   ON f.symbol = b.symbol
GROUP BY f.sector;
```

This query is to compute the one-trading-day and ten-trading-days performance of each sector. This is used as an overview on the landing page in order to show the trend based on sector. For example, usually, when oil or gas prices go up, it is very likely that the energy sector will see an increase gradually. Also, when money tends to go to tech companies in recent days, that will show that the technology sector is comparatively more bullish. Similarly, when the fed rate goes up, the financial services sector might see a gradual jump too.

For the search page:

The most complicated query is the search result table:

```
WITH temp AS (
SELECT p.Symbol, p.Close AS Price, p.Change, p.Volume
FROM Price AS p
WHERE date = (SELECT MAX(Date) FROM Price)
), MC AS (
SELECT m.symbol,
CASE
WHEN m.marketCap between 0 and 2000000000 THEN 'SmallCap'
WHEN m.marketCap between 2000000000 and 10000000000 THEN 'MidCap'
WHEN m.marketCap between 10000000000 and 100000000000 THEN
'LargeCap'
ELSE 'MegaCap'
END AS size
FROM Fundamentals AS m
)
SELECT row_number() OVER (ORDER BY f.symbol) NO,
f.symbol AS Ticker, shortName AS Company, sector AS Sector,
industry AS Industry, country AS Country, size AS Size,
ROUND(f.marketCap, 0) AS MarketCap,
ROUND (trailingPE, 2) AS PE, ROUND (Price, 2) AS Price,
CONCAT (ROUND (100.00 * temp.Change, 2), '%') AS 'Change',
ROUND (temp.Volume, 0) AS Volume
FROM Fundamentals AS f
INNER JOIN temp ON temp.Symbol = f.symbol INNER JOIN MC ON
MC.Symbol = f.symbol
ORDER BY f.symbol;
```

This query is a subquery of filtering certain stocks with investors' own criteria. Since there is one field about market capital in the original dataset, I add a temporary table with "case…when…" by grouping stocks into 4 categories based on their market capital size. Therefore, our users could pick the size they like in the drop-down list instead of manually entering a specific number for the market capital field.

For the MarketView page (sectorpage):

One of the typical queries is the condition_sector query as follows:

```
WITH target AS (
   SELECT p.Symbol, p.Change
   FROM Price AS p
   WHERE Date = (
        SELECT MAX(Date)
        FROM Price
   )
),
compute AS (
   SELECT SUBSTRING(fund.sector, 1, 14) AS Sector,
CONCAT(ROUND(SUM(target.Change*fund.marketcap*100)/SUM(fund.marketc
ap), 2),'%') AS SectorChange,

IF(ROUND(SUM(target.Change*fund.marketcap*100)/SUM(fund.marketcap),
2)>0,1, 0 ) AS positive,

IF(ROUND(SUM(target.Change*fund.marketcap*100)/SUM(fund.marketcap),
2)<0,1, 0 ) AS negative

   FROM Fundamentals AS fund
   INNER JOIN target
   ON fund.symbol = target.Symbol
   GROUP BY fund.sector
   )

SELECT SUM(positive) AS positive, SUM(negative) AS negative,
COUNT(*) AS total
FROM compute
```

The condition_sector query is to find how many sectors are gainers, how many sectors are losers on the latest trading day in the stock market. To construct the query, we calculate the overall market capital change for each sector and mark the sector with positive or negative signals to indicate the condition of the sector, then summarize the condition. With the result of the query and Progress bar in React, we are able to visualize the condition of sectors to help users quickly view the entire market and gain information about sector rotations.

Another typical query on the MarketView page (sector page) is the first_industry query as follows:

```
WITH target AS (
   SELECT p.Symbol, p.Close, p.Change
   FROM Price AS p
   WHERE Date = (
        SELECT MAX(Date)
        FROM Price
   )
)
SELECT target.Symbol AS Symbol, ROUND(target.Close, 2) AS Close,
CONCAT(ROUND(target.Change*100, 2),'%') AS vary
FROM Fundamentals AS fund
INNER JOIN target
ON fund.symbol = target.Symbol
WHERE fund.industry = (
   SELECT fund.industry
   FROM Fundamentals AS fund
```

```
    INNER JOIN target
    ON fund.symbol = target.Symbol
    GROUP BY fund.industry
    ORDER BY
ROUND(SUM(target.Change*fund.marketcap*100)/SUM(fund.marketcap), 2)
DESC
    LIMIT 1
)
ORDER BY target.Change DESC
LIMIT 5
```

The first_industry query is to show the top stocks in the best performing industry on the latest trading day. To construct the query, we find the best-performing industry by calculating the overall market capital change for each industry, then we sort the stocks in the best-performing industry to show the top-performing stocks. With the result of the query and card in React, we are able to show the top stocks in the best-performing industry to help users find alphas in the hot industry.

Performance Evaluation

On the stock detail page, for the query of stock performance benchmarking, we originally split the calculation into multiple queries, for example, there was one query designed for stock performance in price movement vs market average, and another one for stock performance in price movement vs sector average, and similar two queries for volume change comparison. Each of these four queries has a complex structure of joins and temporary tables, which in turn makes the stock page loading very slowly, on average around 10 seconds to display all metrics because these four metrics require four queries to run sequentially. Then we noticed that we can actually combine these queries into one because for the comparisons of volume and price change, we can actually apply one query with just two more attributes to get the calculation for both at once. And for the comparisons of the sector and the overall market, we can still use a temporary table to do the aggregation for the sector but for the overall market, we can simply use OVER() to perform the aggregation in the original table, thus saving our time to run another query on the same page. In this way, we combined the original four queries into one, and the stock page now can be loaded in less than 2 seconds on average.

In the example of a 1-day and 10-day query, we do some optimization to eliminate the intermediate data that is used. Since stock doesn't simply trade on calendar days, we cannot simply do the calculation to get 1 day and 10 days directly through the date. We need to order the date information and get the row number of each so that we can compare. Note that there is an optimization that we do in the common table expression of "a", so that it will only do the sorting to a reduced amount of data (because we filter on the date in the where clause since regardless of what, it is guaranteed that any 10 trading days is within 20 calendar days). Therefore, by doing this the sorting cost will be significantly reduced. This small change alone helps reduce 50% of the total query time, from 7s 500 ms to 1s 600 ms on average. We are just using 2,500 trading days, which is a little more than 10 years of historical data. However, if we want to store the entire history, and a significant proportion of stock would have more than 40 years of history, this would mean an even larger query time improvement, considering the sorting is nlog(n) operation.

In the stock search page, we firstly used * in the SELECT statement when displaying the result table since we wanted to show stock information as much as possible to users. However, the result table was too wide to be shown within one-page width without cutting some digits. By specifically adding the names of columns in the query, not only do we generate a nicer table with key fields for investors

but also dramatically save the time of getting response and achieve a better query performance. The runtime decreases to 3 seconds from 5 seconds on average.

For the condition_sector query and first_industry query, it requires us to compute the change of each sector and industry. For the original data acquired from the API, each stock's capital change should be (today_close - yesterday_close)*(float_shares + outstanding_shares)/market_capital. As you know, today_close and yesterday_close are in Price Table whereas float_shares, outstanding_shares, market_capital belong to the fundamental table. If we try to do the computation in this way, we would always join the 2-day data of the Price Table with the Fundamental table for the above queries and another query. So we create the change variable to indicate percentage price change in the Price table in creating the database, then each stock's capital change would just be the change variable, as the total volume keeps the same. In this case, we just need to inner join one-day data of the Price Table with the Fundamental Table so that we save 50% Join processing time ((date, symbol) is primary key) and some time of formula computation. As both condition_sector query and first_industry query require sequential query running, the optimization here helps condition_sector query save 28% average time from 5s 126ms to 3s 713ms and help first_industry query save 23% average time from 7s 54ms to 5s 444ms.

Technical challenges

When we are building the stock price table and charts, one of the key obstacles is the formatting of the data, since most of our original data is in float when we display the result directly on the page, it does not look professional because the numbers are either large without comma-separated, or percentage shown as too many decimal places. But when if we change data format in the source or back end, it will cause a problem that the front-end functions may not be working, for example, sorting in the table, display of charts, etc. Therefore, we figured out that we should do the formatting in the front end, we researched a lot and finally found a solution to use the toLocaleString() method to format our data in the front end so that we can display our data and charts in any format we need for the scenario without conflicting with the sorting and chart functions.

One technical challenge on the data is that we spend lots of time collecting the data. In the beginning, we are trying to use a stock API provider is called AlphaVantage. However, the free tier is only limited to 500 requests per day and 5 requests per minute. Think about it, if we do this with a single IP, it means 100 minutes for each type of API for all S&P 500 stocks. And we just need to use 5-6 types of API. This is a great limitation for us as we need to get different data (meaning different API calls) for each of the 500 stocks. Since we are a group of four, technically we can call 4 APIs per stock. If we don't try to abuse that system, which we should not do as well, we can ask our friends' help too. But this would create a lot of dependencies and inflexibility for the project, and it is not what it should like for sure.

We try this way in the beginning, sometimes we didn't manage the pace well so it will return the empty results and we have to implement new functions, just to detect and rerun the data fetching job. We cannot simply rerun everything because we have a limitation on the speed as well as the total quota. Even like that, our IP was blocked by the system. So later on, even though we have built lots of abnormal value detection and retry functionalities, we decide to turn to open-source packages for help. That significantly reduces the workload. The downside is that we have no guarantee on the data quality because it could be considered as a free version that is not supervised by some profitable institution, and we have to do lots of screening on the data quality because it has many data variables that are overlapping with each and some of them have poor quality.

When we work on the stock search page, we must consider a tradeoff between the volume of information within one page and user-friendly characteristics. If we include more rows with more stocks within one page, users could click “next page” less time to browse all target stocks. However, it will make it the user harder to see the criteria they apply on the top of the page. We tried some ways like showing only 5 rows per page or fixing the criteria parts, but the page still looked clumsy. In this dilemma, we come up with an idea by adding a “Top” button at the right lower corner of the page. This "scroll to top" button will become visible when the user starts to scroll the page. Besides on the stock search page, we apply this button to all our 4 web pages. It does improve user experience and make our webpage look more professional.

As we tried to provide a quick market view for users, we must pick the limited useful features among a large number of meaningful features. Thus, after referring to famous stock websites like YahooFinance, Fidelity Investment, and so on, we picked up 14 features that do great help for users. As forms occupied large space and did not provide a good visualization, we finally used 8 cards with progress bars to display 14 meaningful features to provide a clear, intuitive, meaningful visualization.