Report3: Financial Database Manipulation

Team 5: Kaichen Bian, Longtao Chen, Zeyu He, Shuhong Cai, Siying Chen

Overview

This project involved a comprehensive analysis of multi-dimensional stock market data using SQL, with data sourced from seven tables. The primary objective was to extract meaningful insights from financial data, daily returns, and volatility, while assessing risk indicators, categorizing stock performance, and filtering stocks based on specific conditions. The report summarizes key processes such as data processing, risk assessment, stock classification, and performance statistics. SQL played a central role throughout the project, enabling efficient management, manipulation, and analysis of complex stock market data.

1.Data Preprocessing and Cleaning

The basic step was to identify and handle missing data before further analysis. BAC had missing capital expenditure (Capex) values, which we treated as NULL. This ensured that the analysis would be based on clean, accurate data, avoiding potential distortions from missing or incomplete information (Table 1).

```sql
-- Identifying stocks with missing financial data
SELECT s.Stock_ID,
       CONCAT(s.Stock_Name, ' - ', s.Stock_ID) AS stock_info,
       f.year AS year,
       IF(f.capex = 0, 'NULL', f.capex) AS capex_status
FROM stocks s
RIGHT JOIN financial_statement_v2 f ON s.Stock_ID = f.stock_id
WHERE f.capex = 0;
```

Stock_ID	stock_info	year	capex_status
BAC	Bank of America Corp - BAC	2019	NULL
BAC	Bank of America Corp - BAC	2020	NULL
BAC	Bank of America Corp - BAC	2021	NULL
BAC	Bank of America Corp - BAC	2022	NULL
BAC	Bank of America Corp - BAC	2023	NULL

table 1

2.Stock Filtering and Classification

First, we filtered stocks with daily returns between -0.025 and 0.025, which helped identify stocks with moderate returns. Additionally, stocks with either high daily returns (greater than 0.025) or very low returns (less than -0.025) were selected to highlight stocks with more extreme performances. Next, a subquery was used to calculate the average daily return for each stock. Stocks with an average daily return greater than 0.001 were filtered (Table 2 and 3 and 4).

```sql
-- Select stocks with daily returns between -0.025 and 0.025
SELECT stock_id, daily_return
FROM daily_return_and_volatility
WHERE daily_return BETWEEN -0.025 AND 0.025;

-- Select stocks with daily return greater than 0.025 or less than -0.025
SELECT stock_id, daily_return
FROM daily_return_and_volatility
WHERE daily_return > 0.025 OR daily_return < -0.025;
```

stock_id	daily_return
BAC	-0.016026
BAC	-0.000783
BAC	-0.001956
BAC	0.009797
BAC	-0.001162
BAC	0.011661
BAC	0.013062
BAC	0.006827
BAC	0.018983
BAC	0.010691
BAC	-0.007166
BAC	-0.005844
BAC	0.005532
BAC	0.017193

table 2

stock_id	daily_return
BAC	0.041531
BAC	0.071561
BAC	0.02536
BAC	-0.034064
BAC	-0.041518
BAC	0.034435
BAC	0.037838
BAC	-0.044964
BAC	-0.025621
BAC	0.046476
BAC	0.028003
BAC	-0.038784
BAC	-0.044248
BAC	-0.046897
BAC	0.029716
BAC	-0.026484

table 3

```sql
-- Use subquery in the FROM clause to fliter a stock
SELECT stock_id, avg_daily_return
FROM (
    SELECT stock_id, AVG(daily_return) AS avg_daily_return
    FROM daily_return_and_volatility
    GROUP BY stock_id
) AS subquery
HAVING avg_daily_return > 0.001;
```

stock_id	avg_daily_return
MSFT	0.0012710063643595869

table 4

Then, we applied a more specific filter by selecting stocks with daily returns greater than the market average, while further refining the results to only include stocks whose names start with the letter "M." This targeted approach allowed us to focus on a specific subset of stocks that meet both performance and naming criteria, offering a more customized view of the market for investors looking for specific opportunities (Table 5).

```sql
-- Select returns greater than average and filter by stock name starting with 'M'
SELECT stock_id, daily_return
FROM daily_return_and_volatility
WHERE daily_return > (SELECT AVG(daily_return) FROM daily_return_and_volatility)
AND stock_id IN (SELECT stock_id FROM stocks WHERE stock_name LIKE 'M%');
```

stock_id	daily_return
MSFT	0.046509
MSFT	0.001276
MSFT	0.007251
MSFT	0.014299
MSFT	0.029005
MSFT	0.003523
MSFT	0.007022
MSFT	0.014983
MSFT	0.009746
MSFT	0.009134
MSFT	0.033418
MSFT	0.028799
MSFT	0.013997

Table 5

Finally, we classified stocks based on their annual returns into three categories: high return (annual return > 20%), moderate return (annual return between 5% and 20%), and low return (annual return < 5%). (Table 6).

```sql
-- Classifying stocks based on return categories
SELECT return_category,
       COUNT(*) AS stock_count
FROM (
    SELECT stock_id,
           CASE
               WHEN yearly_return > 0.2 THEN 'High Return'
               WHEN yearly_return BETWEEN 0.05 AND 0.2 THEN 'Moderate Return'
               ELSE 'Low Return'
           END AS return_category
    FROM yearly_return_and_volatility
) AS categorized_stocks
GROUP BY return_category
HAVING COUNT(*) > 3;
```

return_category	stock_count
High Return	8
Low Return	7
Moderate Return	5

Table 6

3.Risk Analysis and Statistical Indicators

3.1 VaR Calculation

Value at risk (VaR) helps to determine the extent and probabilities of potential losses in their institutional portfolios. Since the price data aren't given, under 5th percentile, we just calculated the return rate instead of the value using two methods (Table 7):

```sql
WITH stock_counts AS (
    SELECT stock_id, COUNT(*) AS total_count
    FROM daily_return_and_volatility
    GROUP BY stock_id
),
ranked_returns AS (
    SELECT drv.stock_id, drv.daily_return,
           ROW_NUMBER() OVER (PARTITION BY drv.stock_id ORDER BY drv.daily_return) AS `rank`,
           sc.total_count
    FROM daily_return_and_volatility drv
    JOIN stock_counts sc ON drv.stock_id = sc.stock_id
),
VaR_Historical AS (
    SELECT stock_id, daily_return AS VaR_95_Historical_Method
    FROM ranked_returns
    WHERE `rank` = FLOOR(0.05 * total_count)
),

stats AS (
    SELECT stock_id,
           AVG(daily_return) AS mean_return,
           STDDEV(daily_return) AS stddev_return
    FROM daily_return_and_volatility
    GROUP BY stock_id
),
VaR_STD AS (
    SELECT stock_id,
           mean_return - 1.65 * stddev_return AS VaR_95_STD_Method
    FROM stats
)
SELECT h.stock_id, h.VaR_95_Historical_Method, s.VaR_95_STD_Method
FROM VaR_Historical h
JOIN VaR_STD s ON h.stock_id = s.stock_id;
```

stock_id	VaR_95_Historical_Method	VaR_95_STD_Method
BAC	-0.0322	-0.03660906302377129
JNJ	-0.016956	-0.020289063690511783
MSFT	-0.029172	-0.030408666823622057
WMT	-0.019059	-0.02235610006305483

Table 7

Historical Simulation: This method calculated the 5th percentile of daily returns based on historical data, representing the loss in 5% of the cases.

Normal Distribution Method: Using the mean and standard deviation of returns, this method estimated the potential loss assuming returns follow a normal distribution.

3.2 Maximum, Minimum Returns, and Volatility Analysis

For each stock, we analyzed the maximum and minimum annual returns and calculated volatility ,used some functions to combine them in different forms and selected data from MSFT as an example. These helped assess how much the stock's performance fluctuated, offering insights into potential risks associated with its market behavior (Table 8 and 9).

```sql
-- Combine yearly max and min return data for analysis using UNION
SELECT stock_id,
       MAX(yearly_return) AS return_value,
       'Max Return' AS return_type
FROM yearly_return_and_volatility
GROUP BY stock_id
UNION
SELECT stock_id,
       MIN(yearly_return) AS return_value,
       'Min Return' AS return_type
FROM yearly_return_and_volatility
GROUP BY stock_id;

-- Analyze max, min, and volatility for all stocks and filter to show only MSFT data using IN function
SELECT CONCAT(s.Stock_ID, ' - ', s.Stock_Name) AS stock_info,
       SUBSTRING(s.Stock_Name, 1, 3) AS stock_prefix,
       MAX(y.yearly_return) AS max_yearly_return,
       MIN(y.yearly_return) AS min_yearly_return,
       POWER(AVG(y.yearly_volatility), 2) AS volatility_squared,
       SQRT(POWER(AVG(y.yearly_volatility), 2)) AS volatility_sqrt
FROM yearly_return_and_volatility y
JOIN stocks s ON y.stock_ID = s.Stock_ID
WHERE s.Stock_ID IN ('MSFT')
GROUP BY s.Stock_ID, s.Stock_Name;
```

stock_id	return_value	return_type
BAC	0.4960846713697318	Max Return
JNJ	0.17402245333029764	Max Return
MSFT	0.5825973337062162	Max Return
WMT	0.2989951287451693	Max Return
BAC	-0.23822550660014652	Min Return
JNJ	-0.08578158378903378	Min Return
MSFT	-0.2802481938985969	Min Return
WMT	-0.004634708520088626	Min Return

Table 8

stock_info	stock_prefix	max_yearly_return	min_yearly_return	volatility_squared	volatility_sqrt
MSFT - Microsoft Corp	Mic	0.5825973337062162	-0.2802481938985969	0.08396912128782487	0.28977425918777683

Table 9

3.3 Calculating Sharpe Ratio Calculation

The Sharpe Ratio is a key metric for assessing the risk-adjusted return of an investment. In this analysis, we calculated the Sharpe Ratio for each stock across the years. This allowed us to measure how well stocks performed, not just in terms of returns, but also in terms of risk relative to those returns. It provided a clearer picture of which stocks delivered high returns with relatively low volatility (Table 10).

```sql
-- Calculate Sharpe ratio for each year
SELECT
    stock_id,
    MAX(CASE WHEN year = 2019 THEN (yearly_return - 0.02) / yearly_volatility END) AS sharpe_ratio_2019,
    MAX(CASE WHEN year = 2020 THEN (yearly_return - 0.02) / yearly_volatility END) AS sharpe_ratio_2020,
    MAX(CASE WHEN year = 2021 THEN (yearly_return - 0.02) / yearly_volatility END) AS sharpe_ratio_2021,
    MAX(CASE WHEN year = 2022 THEN (yearly_return - 0.02) / yearly_volatility END) AS sharpe_ratio_2022,
    MAX(CASE WHEN year = 2023 THEN (yearly_return - 0.02) / yearly_volatility END) AS sharpe_ratio_2023
FROM yearly_return_and_volatility
GROUP BY stock_id
ORDER BY stock_id;
```

stock_id	sharpe_ratio_2019	sharpe_ratio_2020	sharpe_ratio_2021	sharpe_ratio_2022	sharpe_ratio_2023
BAC	1.8291929395402435	-0.2331002371647721	1.8672466204262357	-0.7967010781773565	0.10537105587254816
JNJ	0.9382612274030887	0.292108204825824	0.6523915501880044	0.22883992565584077	-0.6406969627020439
MSFT	2.838713174920501	0.9254718491414975	2.4028741247328687	-0.8519388912759978	2.245717778785871
WMT	1.938123514208298	0.6782865852015046	-0.0018161839756319085	-0.09239290127530632	0.6969800110046483

Table 10

3.4 Rolling Average Return Calculation

To understand the fluctuation in stock performance over time, we calculated a 5-day rolling average of daily returns for each stock. Rolling averages smooth short-term fluctuations and reveal longer-term trends, helping investors observe stock performance over different periods without being overly influenced by daily volatility (Table 11).

```sql
-- Calculate rolling average return
WITH ordered_data AS (
    SELECT stock_id, daily_return,
           ROW_NUMBER() OVER (PARTITION BY stock_id ORDER BY stock_id) AS row_num
    FROM daily_return_and_volatility
)
SELECT stock_id, daily_return,
       AVG(daily_return) OVER (PARTITION BY stock_id ORDER BY row_num ROWS 4 PRECEDING) AS rolling_avg_return
FROM ordered_data;
```

stock_id	daily_return	rolling_avg_return
BAC	-0.016026	-0.016026
BAC	0.041531	0.0127525
BAC	-0.000783	0.008240666666666667
BAC	-0.001956	0.0056915
BAC	0.009797	0.0065126
BAC	-0.001162	0.0094854
BAC	0.011661	0.0035114
BAC	0.013062	0.0062804
BAC	0.006827	0.008036999999999999
BAC	0.071561	0.0203898
BAC	0.018983	0.0244188
BAC	0.010691	0.0242248
BAC	-0.007166	0.0201792
BAC	-0.005844	0.017645

Table 11

4.Data Selection and Integration by Specific Conditions

In addition to classification, financial data was integrated to provide a more comprehensive view of the relationships between financial health and stock market performance. Various joins were used to combine data, including revenue, gross profit, and net income, with yearly returns. ‘SELECT DINTINCT’ function helped to find the latest net income, ensuring that we captured the

most up-to-date financial information for each stock (Table 12,13,14 and 15).

```sql
-- Select distinct stock IDs with their most recent year net income
SELECT DISTINCT stock_id,
       FIRST_VALUE(year) OVER (PARTITION BY stock_id ORDER BY year DESC) AS latest_year,
       FIRST_VALUE(net_income) OVER (PARTITION BY stock_id ORDER BY year DESC) AS latest_net_income
FROM financial_statement_v2;

-- Use INNER JOIN to combine return and revenue
SELECT f.stock_id, f.year, f.revenue, y.yearly_return
FROM financial_statement_v2 f
INNER JOIN yearly_return_and_volatility y
ON f.stock_id = y.stock_id AND f.year = y.year;

-- Using USING to join to combine gross_profit and revenue
SELECT f.stock_id, f.year, f.gross_profit, y.yearly_return
FROM financial_statement_v2 f
JOIN yearly_return_and_volatility y USING (stock_id, year);

-- Using LEFT JOIN to match capex with yearly_return, and treat capex = 0 as NULL
SELECT f.stock_id, f.year,
       CASE WHEN f.capex = 0 THEN NULL ELSE f.capex END AS capex,
       y.yearly_return
FROM financial_statement_v2 f
LEFT JOIN yearly_return_and_volatility y
ON f.stock_id = y.stock_id AND f.year = y.year;
```

stock_id	latest_year	latest_net_income
BAC	2023	24.87
JNJ	2023	38.02
MSFT	2023	88.14
WMT	2023	17.64

Table 12

stock_id	year	revenue	yearly_return
BAC	2019	91.24	0.44316420685928093
BAC	2020	85.53	-0.11632743773881082
BAC	2021	89.11	0.4960846713697318
BAC	2022	94.95	-0.23822550660014652
BAC	2023	98.58	0.04825268270356031
JNJ	2019	82.06	0.17402245333029764
JNJ	2020	82.58	0.1082409702855609
JNJ	2021	93.77	0.11438227837216663
JNJ	2022	85.16	0.05975543479696621
JNJ	2023	86.58	-0.08578158378903378
MSFT	2019	143.02	0.5825973337062162
MSFT	2020	168.09	0.42533102964865543
MSFT	2021	198.27	0.5247690036921517
MSFT	2022	211.92	-0.2802481938985969

Table 13

stock_id	year	gross_profit	yearly_return
BAC	2019	91.24	0.44316420685928093
BAC	2020	85.53	-0.11632743773881082
BAC	2021	89.11	0.4960846713697318
BAC	2022	94.95	-0.23822550660014652
BAC	2023	98.58	0.04825268270356031
JNJ	2019	54.5	0.17402245333029764
JNJ	2020	54.16	0.1082409702855609
JNJ	2021	55.39	0.11438227837216663
JNJ	2022	58.61	0.05975543479696621
JNJ	2023	60.11	-0.08578158378903378
MSFT	2019	96.94	0.5825973337062162
MSFT	2020	115.86	0.42533102964865543
MSFT	2021	135.62	0.5247690036921517
MSFT	2022	146.2	-0.2802481938985969

Table 14

stock_id	year	capex	yearly_return
JNJ	2020	4.18	0.1082409702855609
JNJ	2021	4.27	0.11438227837216663
JNJ	2022	4.81	0.05975543479696621
JNJ	2023	4.81	-0.08578158378903378
BAC	2019	NULL	0.44316420685928093
BAC	2020	NULL	-0.11632743773881082
BAC	2021	NULL	0.4960846713697318
BAC	2022	NULL	-0.23822550660014652
BAC	2023	NULL	0.04825268270356031
WMT	2019	10.72	0.2989951287451693
WMT	2020	10.7	0.2332171901479967
WMT	2021	10.34	0.01969698485441329
WMT	2022	13.12	-0.004634708520088626
WMT	2023	16.87	0.12881256799826502

Table 15

5.Conclusion

In conclusion, this report provided a comprehensive analysis of stock market data from multiple angles, focusing on data preprocessing, risk assessment, stock classification, and performance evaluation. Key processes such as cleaning financial data, calculating risk metrics like Value-at-Risk (VaR) and Sharpe Ratio, and analyzing daily and yearly returns were essential to understanding the financial health and market performance of various stocks. The project also included classifying stocks based on returns and identifying stocks that met specific filtering criteria, providing deeper insights into potential investment opportunities and market risks.

SQL played an indispensable role throughout this project, providing the powerful querying and manipulation tools needed to handle large and complex datasets efficiently. SQL's capabilities made it possible to filter, sort, classify, and integrate data from different tables, enabling a seamless analysis of the stock market's financial and performance metrics.Moreover, SQL's versatility in handling both structured financial data and performance indicators made it easier to derive actionable insights. These contributed to building a clearer picture of how individual stocks performed within the larger market context, helping investors make informed decisions.