1)Executive Summary:

Our project aims to build a comprehensive data warehousing solution for stock market analytics, leveraging structured data from various dimensions and a central fact table. The primary objective is to facilitate detailed historical analysis and forecasting for stock market performance, enhancing decision-making capabilities for investors and financial analysts.

Project Highlights:

- **Data Dimensions:** We have structured our data around key dimensions such as time, stocks, companies, and indices. Each dimension offers detailed attributes crucial for comprehensive analysis.
- **Fact Table:** The fact table consolidates daily stock market data, including opening, high, low, closing prices, adjusted closing prices, and volume. This allows for granular analysis and reporting at various levels of aggregation.
- **Data Integration and Quality:** Rigorous data collection, integration, and cleansing processes ensure the accuracy and reliability of the data warehouse. This enhances the trustworthiness of insights derived from the system.
- **Analytical Capabilities:** Our solution supports diverse analytical needs, from exploratory data analysis (EDA) to advanced modeling and forecasting. Users can conduct trend analysis, correlation studies, and scenario modeling to uncover actionable insights.
- **Visualization and Reporting:** Interactive dashboards and intuitive visualizations enable users to intuitively explore data trends and patterns. This facilitates effective communication of insights across stakeholders.

Future Directions: Moving forward, the project will focus on expanding data sources, integrating additional financial instruments, and enhancing predictive modeling capabilities. Continuous improvement in data governance and user interface design will ensure the system remains robust and user-friendly.

In summary, our data warehousing project offers a powerful tool for analyzing historical stock market data, empowering stakeholders with actionable insights and facilitating informed decision-making in financial markets.

2) Problem Statement

Context: In the dynamic landscape of financial markets, stakeholders constantly seek robust tools to analyze historical stock market data comprehensively. This project addresses the need for a sophisticated data warehousing solution tailored to facilitate in-depth analysis and decision-making.

Issue: The challenge lies in efficiently integrating and analyzing vast amounts of daily stock market data across multiple dimensions, including time, stocks, companies, and indices. Traditional methods often lack scalability and real-time insights required by modern investors and analysts.

Relevance: Accurate historical data analysis is crucial for predicting market trends, identifying investment opportunities, and mitigating financial risks. A reliable data warehouse can significantly enhance the ability to derive actionable insights from historical stock performance data.

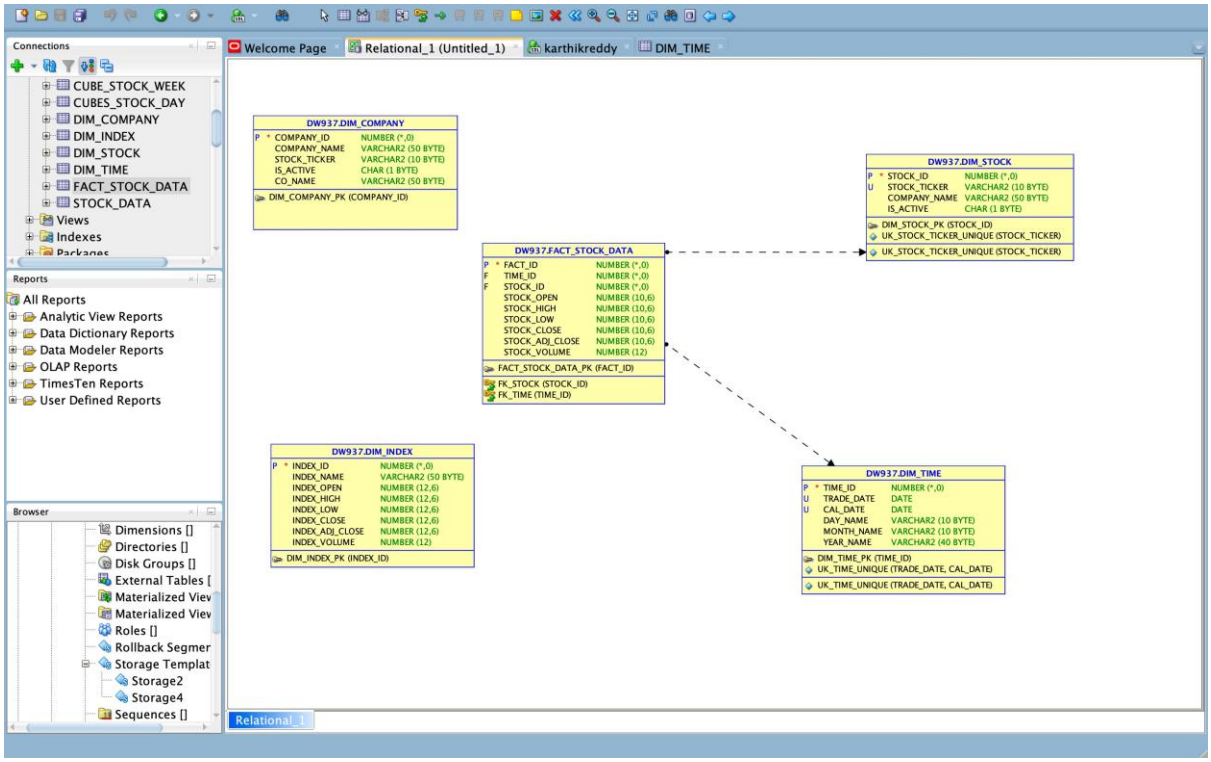
Objective: This project aims to design and implement a data warehousing solution that consolidates historical stock market data into a centralized repository. By structuring data

around key dimensions and employing advanced analytics, the objective is to empower users with intuitive tools for trend analysis, forecasting, and strategic decision-making.

Significance: Addressing this challenge not only enhances the efficiency and accuracy of financial analysis but also provides a competitive edge in navigating volatile market conditions. The project's outcome will support informed decision-making, improve investment strategies, and ultimately contribute to better financial outcomes for stakeholders.

Entity-Relationship Diagram (ERD)

The ERD below demonstrates the design of the data cubes used in the project, showcasing the dimensional modeling skills applied:



Explanation:

Dimensions: Time, Stock, Company, and Index.

Fact Table: Captures daily stock market data including prices and volumes.

Relationships: Fact table links to dimension tables through foreign keys.

"5W+H" approach:

Who: Investors, financial analysts, and stakeholders in the stock market industry are interested in this problem. They rely on historical data to make informed decisions regarding investments, risk management, and strategic planning.

What: The challenge is to effectively analyze and derive insights from extensive historical stock market data. This includes daily trading information such as opening, high, low, and closing prices, adjusted closing prices, and trading volumes across various stocks, companies, and indices.

When and Where: This problem arises daily in global financial markets where data is generated continuously, spanning different time zones and trading sessions. Timely access to

accurate historical data is critical for making decisions in real-time and for retrospective analysis.

Why: Addressing this challenge is essential to improve decision-making accuracy, enhance investment strategies, and mitigate financial risks. By leveraging comprehensive historical data, stakeholders can identify market trends, correlations, and anomalies that impact investment outcomes.

How: This challenge can be effectively solved by designing and implementing a robust data warehousing solution. This solution will consolidate diverse datasets from different dimensions (time, stocks, companies, indices) into a unified repository. Advanced analytics tools will enable users to perform detailed trend analysis, predictive modelling, and scenario planning, thus empowering them with actionable insights.

3: Literature Review

Implementing OLAP Concepts:

Dimensional Modeling:
Dimensional modeling is a design technique used in data warehousing to organize and structure data for easy querying and analysis. It typically involves creating dimension tables that describe the business entities and a central fact table that holds numerical metrics or measures associated with these entities.

1. Dimension Tables

dim_time:

- **Attributes:** TIME_ID (Primary Key), TRADE_DATE, CAL_DATE, DAY_NAME, MONTH_NAME, YEAR_NAME.
- **Purpose:** Stores time-related attributes used for analyzing stock market data by different time dimensions such as day, month, and year.

dim_stock:

- **Attributes:** STOCK_ID (Primary Key), STOCK_TICKER, COMPANY_NAME, IS_ACTIVE.
- **Purpose:** Describes details about stocks, including company information and activity status.

dim_company:

- **Attributes:** COMPANY_ID (Primary Key), COMPANY_NAME, STOCK_TICKER, IS_ACTIVE, CO_NAME.
- **Purpose:** Provides additional information about companies listed in the stock market, linked to stock ticker and company name.

dim_index:

- **Attributes:** INDEX_ID (Primary Key), INDEX_NAME, INDEX_OPEN, INDEX_HIGH, INDEX_LOW, INDEX_CLOSE, INDEX_ADJ_CLOSE, INDEX_VOLUME.
- **Purpose:** Contains data about market indices, including daily trading metrics like opening, high, low, closing prices, adjusted close, and volume.

2. Fact Table

fact_stock_data:

- **Attributes:** FACT_ID (Primary Key), TIME_ID (Foreign Key to dim_time), STOCK_ID (Foreign Key to dim_stock), STOCK_OPEN, STOCK_HIGH, STOCK_LOW, STOCK_CLOSE, STOCK_ADJ_CLOSE, STOCK_VOLUME.
- **Purpose:** Central fact table capturing detailed stock market data for analysis and reporting. It links to dimension tables (dim_time and dim_stock) through foreign key relationships and stores numeric measures such as stock prices and trading volumes.

Dimensional Modelling Benefits

- **Simplicity:** Dimensional models are straightforward and intuitive, making it easier to understand and query data.
- **Query Performance:** Optimized for query performance, as data is denormalized and aggregated, reducing the need for complex joins.
- **Flexibility:** Supports various types of analysis (e.g., by time, by company) due to its multidimensional structure.
- **Scalability:** Allows for easy expansion and integration of new dimensions or measures as business requirements evolve.

Hierarchies Definition

1. dim_time

Hierarchy: Time

- **Year > Month > Day**

Attributes:

- YEAR_NAME
- MONTH_NAME
- DAY_NAME

Purpose:

- Allows aggregation and analysis of stock market data across different time periods:
  - **Year:** Aggregate data by year to analyze annual trends.
  - **Month:** Drill down to monthly data to observe monthly fluctuations.
  - **Day:** Detailed analysis at the daily level for specific trading days.

2. dim_stock

Hierarchy: Stock

- **Company > Stock Ticker**

Attributes:

- COMPANY_NAME
- STOCK_TICKER

Purpose:

- Organizes stocks by company, facilitating analysis of performance and market activities specific to each company.
- Allows users to drill down from company-level insights to individual stock ticker details.

3. dim_company

Hierarchy: Company

- **Company Name**

Attributes:

- COMPANY_NAME

Purpose:

- Provides a single-level hierarchy for companies listed in the stock market
- Enables analysis and comparison of company-specific metrics across different dimensions (e.g., time, index).

4. dim_index

Hierarchy: Index

- **Index Name**

Attributes:

- INDEX_NAME

Purpose:

- Represents market indices for comparative analysis and benchmarking.
- Facilitates aggregation and comparison of index performance over time and across other dimensions.

Benefits of Hierarchies

- **Navigational Efficiency:** Users can navigate through data dimensions intuitively, from higher-level summaries to detailed insights.
- **Analytical Flexibility:** Supports drill-down, roll-up, and slice-and-dice operations for deeper analysis of data.
- **Aggregation Capabilities:** Enables aggregation of data at various levels of granularity, enhancing analytical capabilities.

Example Queries for Building Data Cubes and Defining Aggregation Levels

1. Aggregate Data by Day

```
CREATE TABLE cube_stock_day AS
SELECT
    d.TIME_ID,
    d.TRADE_DATE,
    s.STOCK_ID,
    s.STOCK_TICKER,
    SUM(f.STOCK_OPEN) AS TOTAL_OPEN,
    SUM(f.STOCK_HIGH) AS TOTAL_HIGH,
    SUM(f.STOCK_LOW) AS TOTAL_LOW,
    SUM(f.STOCK_CLOSE) AS TOTAL_CLOSE,
    SUM(f.STOCK_ADJ_CLOSE) AS TOTAL_ADJ_CLOSE,
    SUM(f.STOCK_VOLUME) AS TOTAL_VOLUME
FROM
    fact_stock_data f
    JOIN dim_time d ON f.TIME_ID = d.TIME_ID
    JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
GROUP BY
    d.TIME_ID,
    d.TRADE_DATE,
    s.STOCK_ID,
    s.STOCK_TICKER;
```

2. Aggregate Data by Week

```
CREATE TABLE cube_stock_week AS
SELECT
    TO_CHAR(d.TRADE_DATE, 'IW') AS WEEK_NUMBER, -- Week number based on ISO week date
    TO_CHAR(d.TRADE_DATE, 'YYYY') AS YEAR,
    s.STOCK_ID,
    s.STOCK_TICKER,
    SUM(f.STOCK_OPEN) AS TOTAL_OPEN,
    SUM(f.STOCK_HIGH) AS TOTAL_HIGH,
    SUM(f.STOCK_LOW) AS TOTAL_LOW,
    SUM(f.STOCK_CLOSE) AS TOTAL_CLOSE,
    SUM(f.STOCK_ADJ_CLOSE) AS TOTAL_ADJ_CLOSE,
    SUM(f.STOCK_VOLUME) AS TOTAL_VOLUME
FROM
    fact_stock_data f
    JOIN dim_time d ON f.TIME_ID = d.TIME_ID
    JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
GROUP BY
    TO_CHAR(d.TRADE_DATE, 'IW'),
    TO_CHAR(d.TRADE_DATE, 'YYYY'),
    s.STOCK_ID,
    s.STOCK_TICKER;
```

3. Aggregate Data by Month

```
CREATE TABLE cube_stock_month AS
SELECT
    TO_CHAR(d.TRADE_DATE, 'YYYY-MM') AS MONTH,
    s.STOCK_ID,
    s.STOCK_TICKER,
    SUM(f.STOCK_OPEN) AS TOTAL_OPEN,
    SUM(f.STOCK_HIGH) AS TOTAL_HIGH,
    SUM(f.STOCK_LOW) AS TOTAL_LOW,
    SUM(f.STOCK_CLOSE) AS TOTAL_CLOSE,
    SUM(f.STOCK_ADJ_CLOSE) AS TOTAL_ADJ_CLOSE,
    SUM(f.STOCK_VOLUME) AS TOTAL_VOLUME
FROM
    fact_stock_data f
```

```
    JOIN dim_time d ON f.TIME_ID = d.TIME_ID
    JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
GROUP BY
    TO_CHAR(d.TRADE_DATE, 'YYYY-MM'),
    s.STOCK_ID,
    s.STOCK_TICKER;
```

Explanation:

- **Dimension Tables (dim_time, dim_stock):** These tables are joined with fact_stock_data to associate time-related and stock-related attributes with the aggregated measures.
- **Grouping and Aggregation:** SUM() functions aggregate numeric measures (STOCK_OPEN, STOCK_HIGH, etc.) based on the specified grouping criteria (TIME_ID, TRADE_DATE, STOCK_ID, STOCK_TICKER).
- **Time Formatting:** TO_CHAR() functions are used to format dates into week numbers ('IW' for ISO week number) and month ('YYYY-MM' for year-month format) for grouping by week and month respectively.
- **Table Creation:** Results are stored in new tables (cube_stock_day, cube_stock_week, cube_stock_month) representing different levels of aggregation.

Considerations:

- **Performance:** Ensure indexes are properly defined on join columns (TIME_ID, STOCK_ID) and where clauses to optimize query performance.
- **ETL Process:** These queries can be integrated into your ETL process to automate the creation and maintenance of data cubes as new data is loaded.
- **Query Optimization:** Monitor query performance and consider partitioning large tables or using materialized views for further optimization.

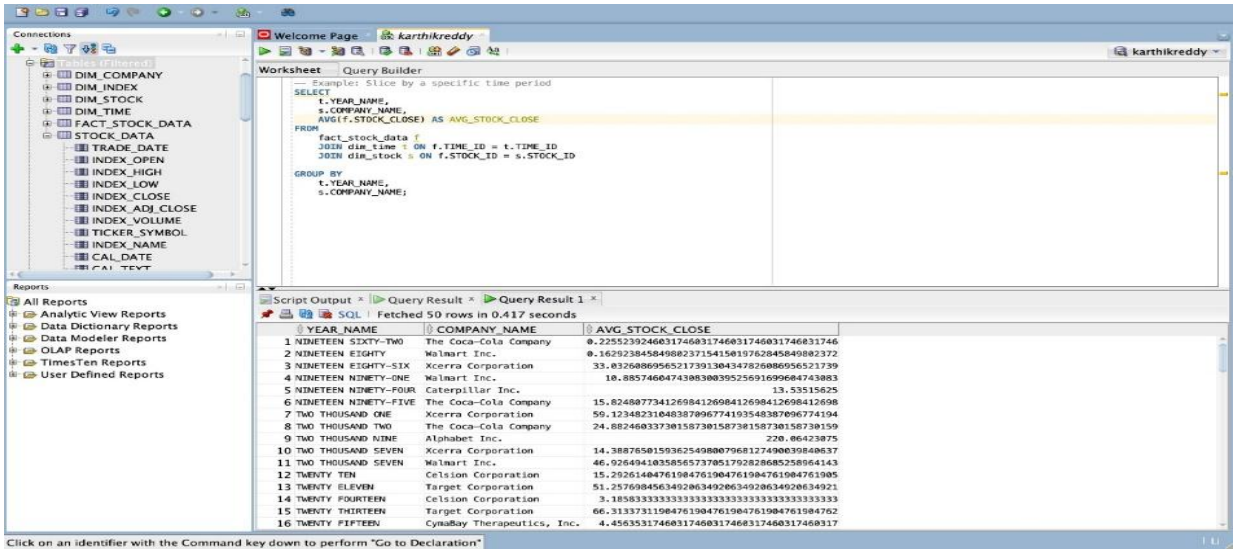
OLAP Queries Using

Here are examples of OLAP queries using to demonstrate common operations:

1. Slice-and-Dice Operation

Slice: Selecting a subset of data along one dimension.

```
-- Example: Slice by a specific time period (e.g., year 2023)
SELECT
    t.YEAR_NAME,
    s.COMPANY_NAME,
    AVG(f.STOCK_CLOSE) AS AVG_STOCK_CLOSE
FROM
    fact_stock_data f
    JOIN dim_time t ON f.TIME_ID = t.TIME_ID
    JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
WHERE
    t.YEAR_NAME = '2023'
GROUP BY
    t.YEAR_NAME,
    s.COMPANY_NAME;
```



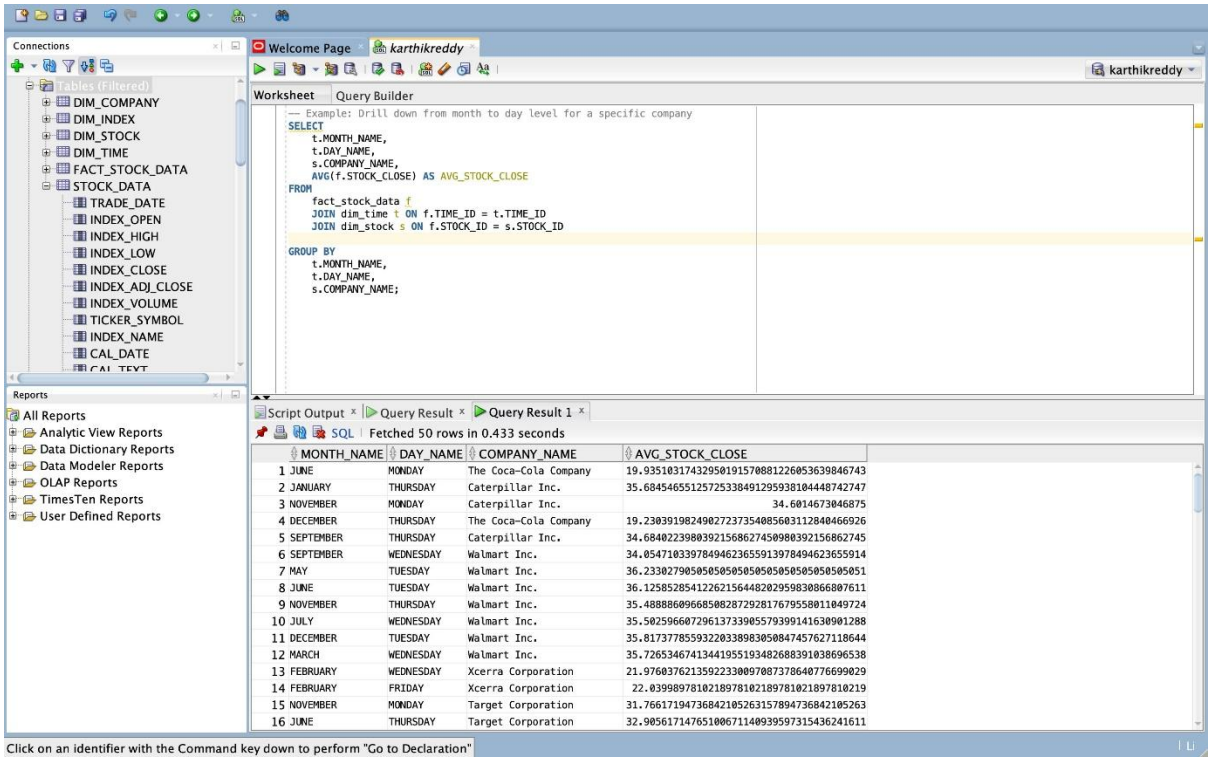
Dice: Selecting a subset of data along multiple dimensions.

```
 -- Example: Dice by year and stock ticker
SELECT
    t.YEAR_NAME,
    s.STOCK_TICKER,
    AVG(f.STOCK_CLOSE) AS AVG_STOCK_CLOSE
FROM
    fact_stock_data f
    JOIN dim_time t ON f.TIME_ID = t.TIME_ID
    JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
WHERE
    t.YEAR_NAME IN ('2022', '2023')
GROUP BY
    t.YEAR_NAME,
    s.STOCK_TICKER;
```

2. Drill-Down Operation

Drill down from higher to lower levels of granularity.

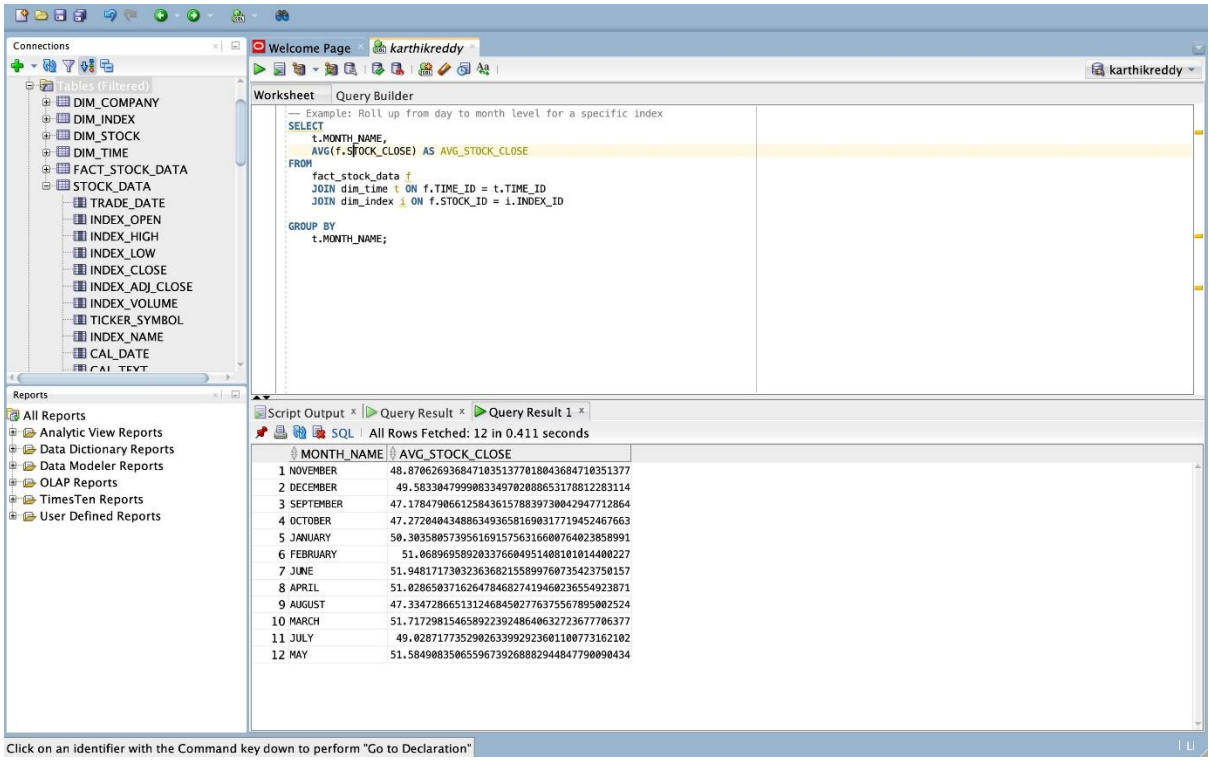
```
-- Example: Drill down from month to day level for a specific company
SELECT
    t.MONTH_NAME,
    t.DAY_NAME,
    s.COMPANY_NAME,
    AVG(f.STOCK_CLOSE) AS AVG_STOCK_CLOSE
FROM
    fact_stock_data f
    JOIN dim_time t ON f.TIME_ID = t.TIME_ID
    JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
WHERE
    t.MONTH_NAME = 'June'
    AND s.COMPANY_NAME = 'Company XYZ'
GROUP BY
    t.MONTH_NAME,
    t.DAY_NAME,
    s.COMPANY_NAME;
```



3. Roll-Up Operation

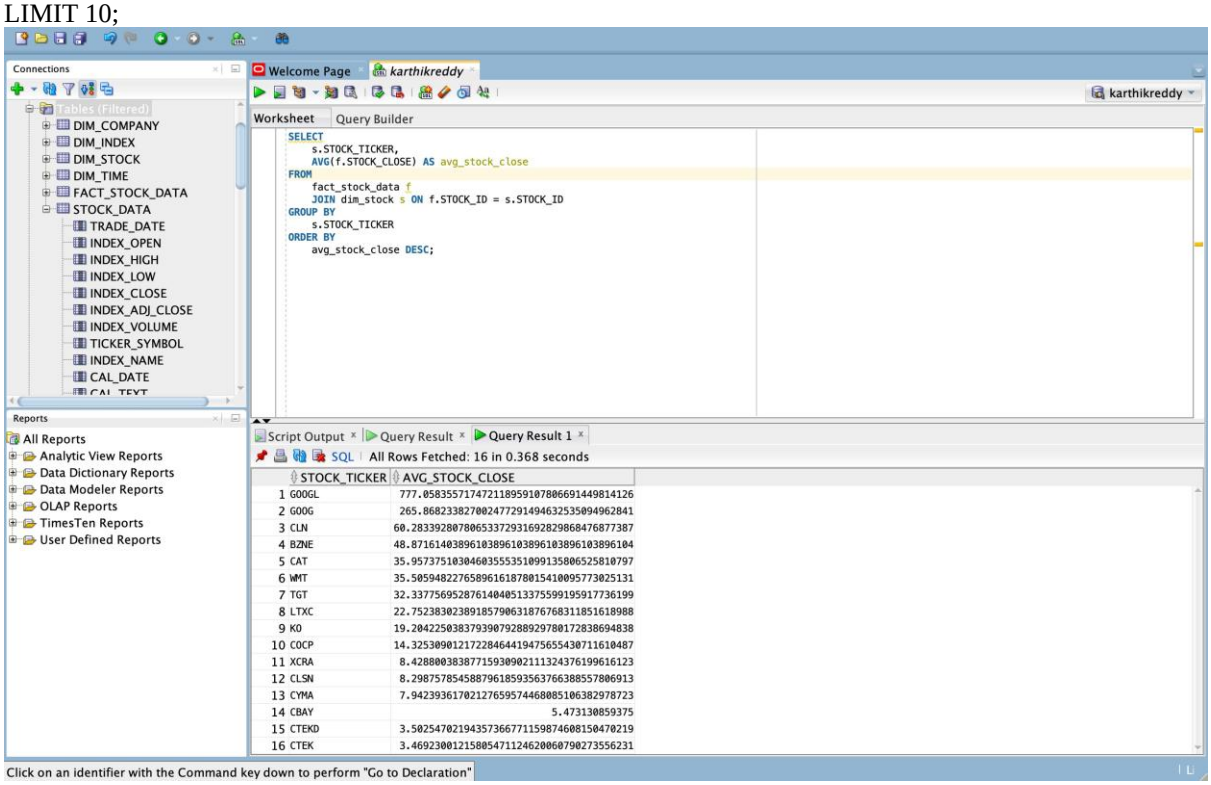
Aggregate data from lower to higher levels of granularity.

```
-- Example: Roll up from day to month level for a specific index
SELECT
   t.MONTH_NAME,
   AVG(f.STOCK_CLOSE) AS AVG_STOCK_CLOSE
FROM
   fact_stock_data f
   JOIN dim_time t ON f.TIME_ID = t.TIME_ID
   JOIN dim_index i ON f.STOCK_ID = i.INDEX_ID
WHERE
   t.YEAR_NAME = '2023'
   AND i.INDEX_NAME = 'S&P 500'
GROUP BY
   t.MONTH_NAME;
```



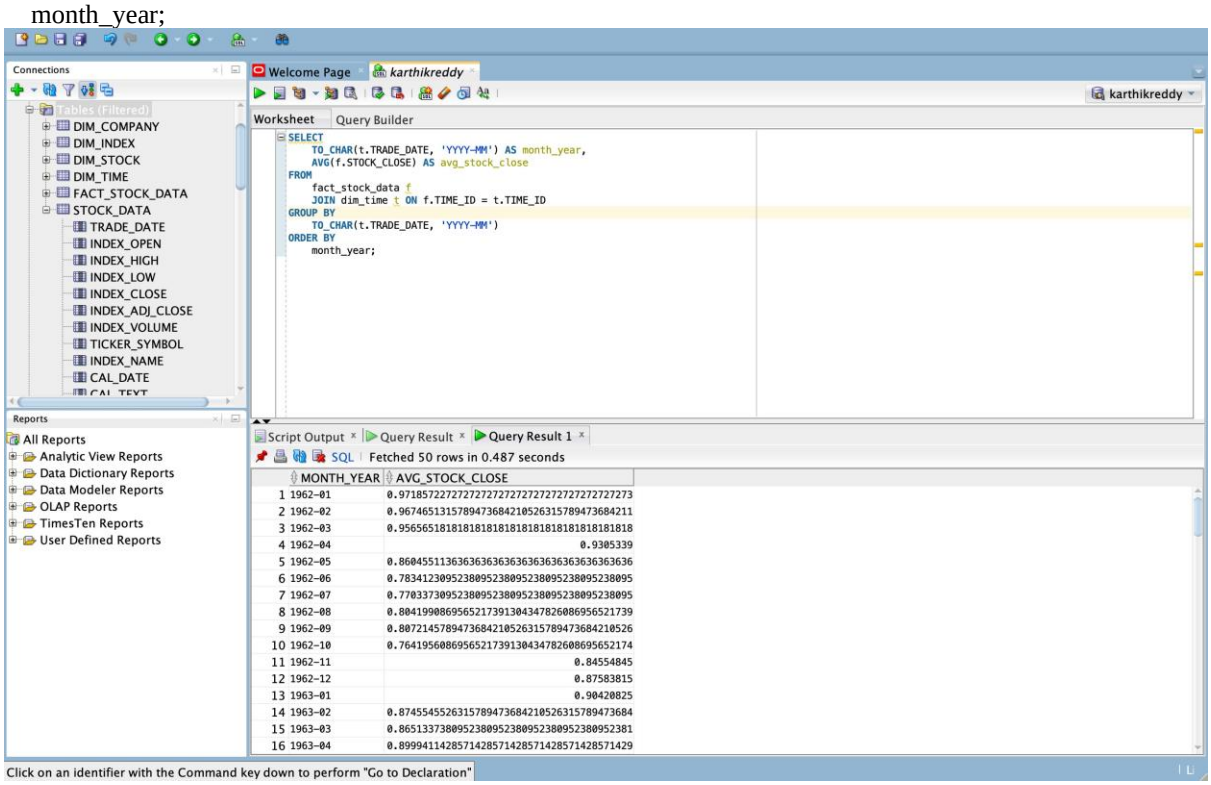
- **Top Performing Stocks by Average Closing Price:**

```
SELECT
   s.STOCK_TICKER,
   AVG(f.STOCK_CLOSE) AS avg_stock_close
FROM
   fact_stock_data f
   JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
GROUP BY
   s.STOCK_TICKER
ORDER BY
   avg_stock_close DESC
LIMIT 10;
```



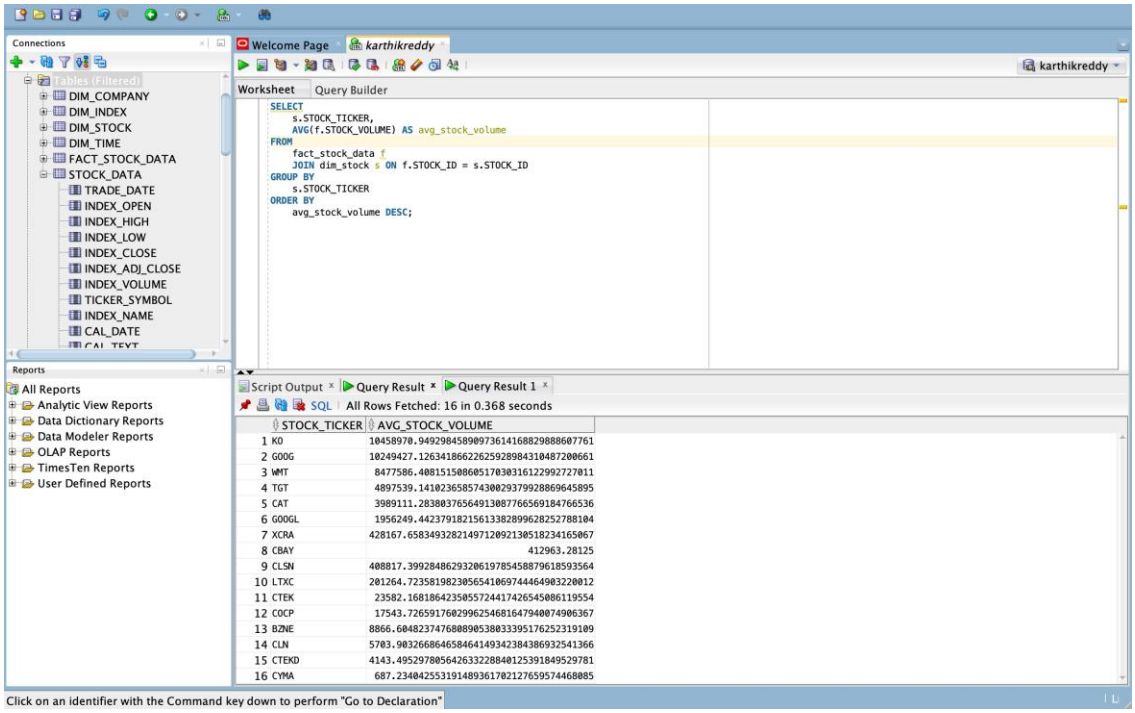
- **Monthly Average Stock Prices Over Years:**

```
SELECT
   TO_CHAR(t.TRADE_DATE, 'YYYY-MM') AS month_year,
   AVG(f.STOCK_CLOSE) AS avg_stock_close
FROM
   fact_stock_data f
   JOIN dim_time t ON f.TIME_ID = t.TIME_ID
GROUP BY
   TO_CHAR(t.TRADE_DATE, 'YYYY-MM')
ORDER BY
   month_year;
```



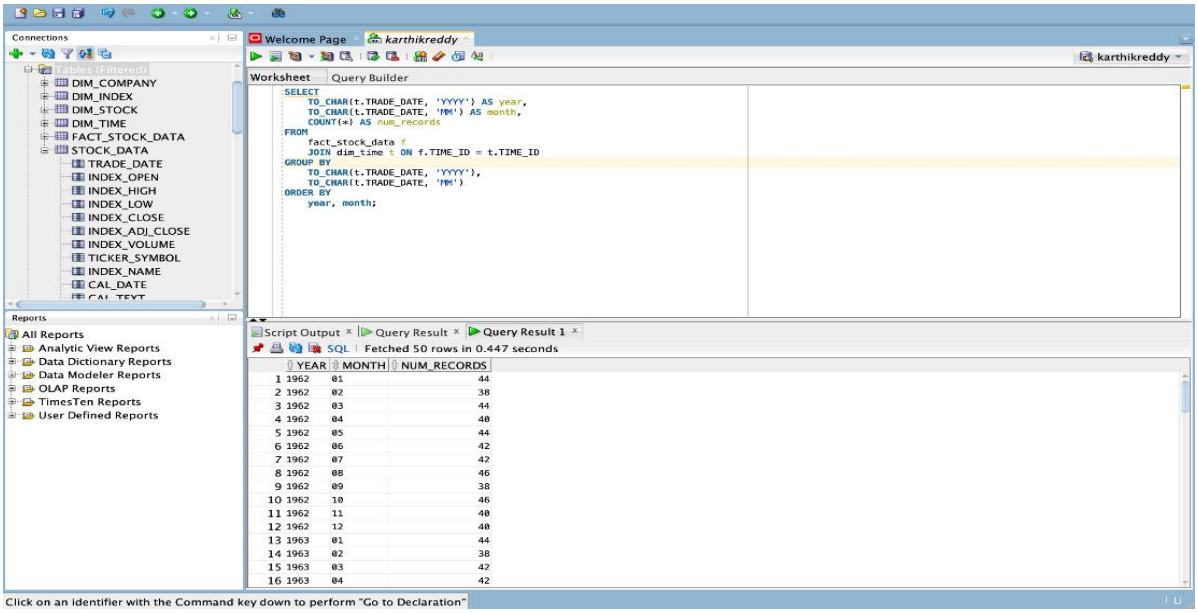
- **Average Daily Trading Volume by Stock Ticker:**

```
SELECT
   s.STOCK_TICKER,
   AVG(f.STOCK_VOLUME) AS avg_stock_volume
FROM
   fact_stock_data f
   JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
GROUP BY
   s.STOCK_TICKER
ORDER BY
   avg_stock_volume DESC;
```



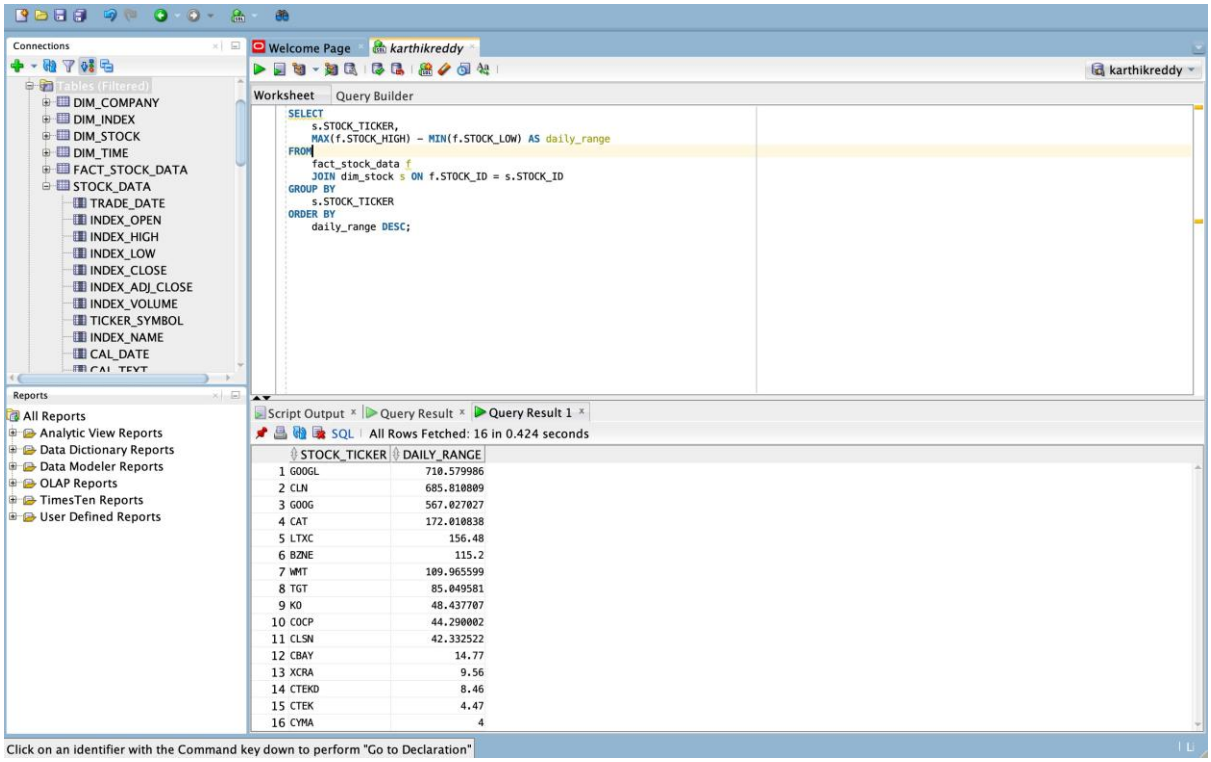
- **Count of Records by Year and Month:**

```
SELECT
   TO_CHAR(t.TRADE_DATE, 'YYYY') AS year,
   TO_CHAR(t.TRADE_DATE, 'MM') AS month,
   COUNT(*) AS num_records
FROM
   fact_stock_data f
   JOIN dim_time t ON f.TIME_ID = t.TIME_ID
GROUP BY
   TO_CHAR(t.TRADE_DATE, 'YYYY'),
   TO_CHAR(t.TRADE_DATE, 'MM')
ORDER BY
   year, month;
```



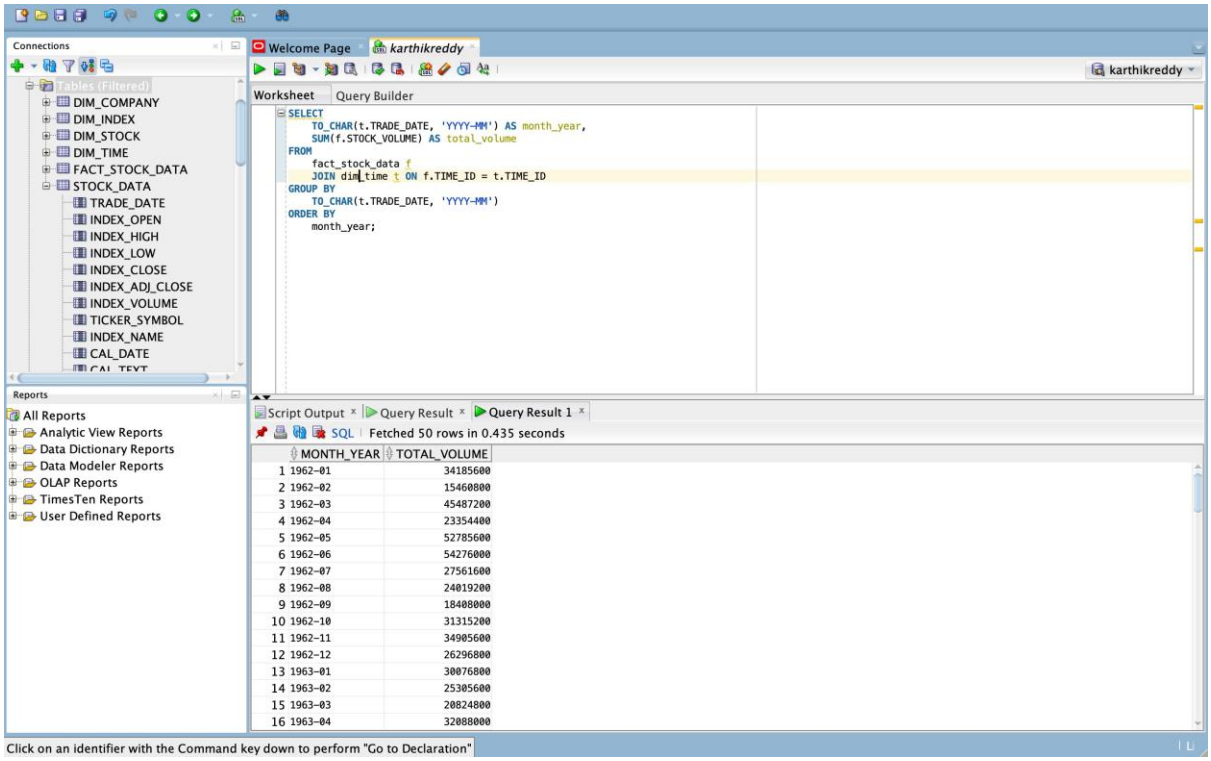
- **Daily High-Low Range by Stock Ticker:**

```
SELECT
   s.STOCK_TICKER,
   MAX(f.STOCK_HIGH) - MIN(f.STOCK_LOW) AS daily_range
FROM
   fact_stock_data f
   JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
GROUP BY
   s.STOCK_TICKER
ORDER BY
   daily_range DESC;
```



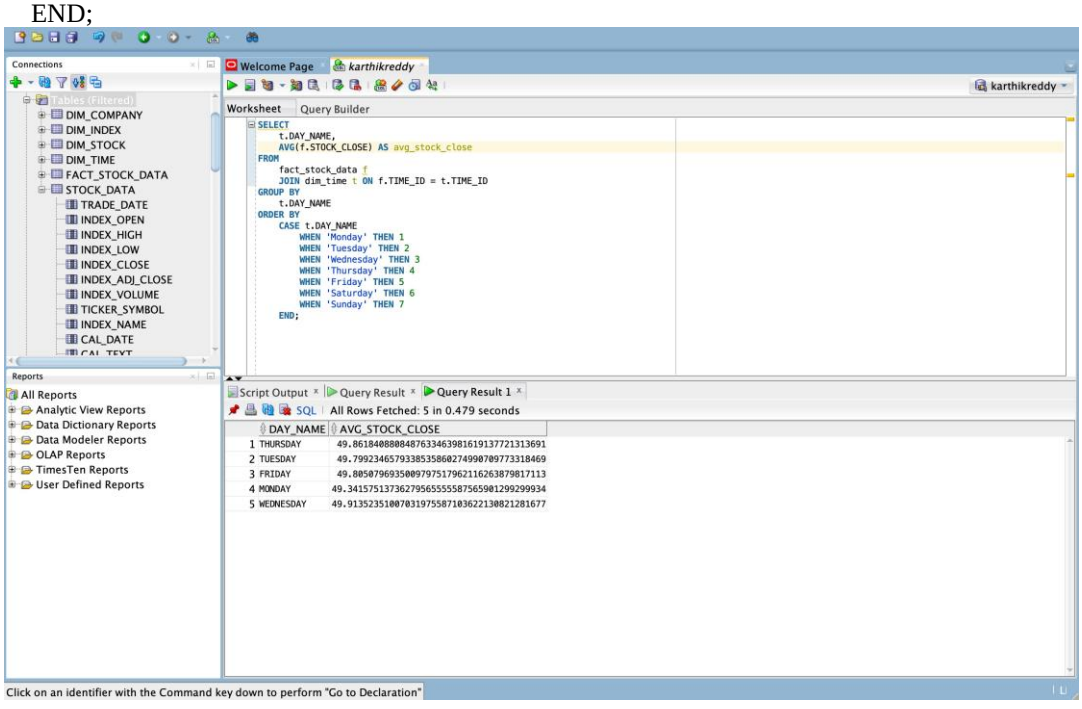
- **Monthly Total Trading Volume:**

```
SELECT
   TO_CHAR(t.TRADE_DATE, 'YYYY-MM') AS month_year,
   SUM(f.STOCK_VOLUME) AS total_volume
FROM
   fact_stock_data f
   JOIN dim_time t ON f.TIME_ID = t.TIME_ID
GROUP BY
   TO_CHAR(t.TRADE_DATE, 'YYYY-MM')
ORDER BY
   month_year;
```



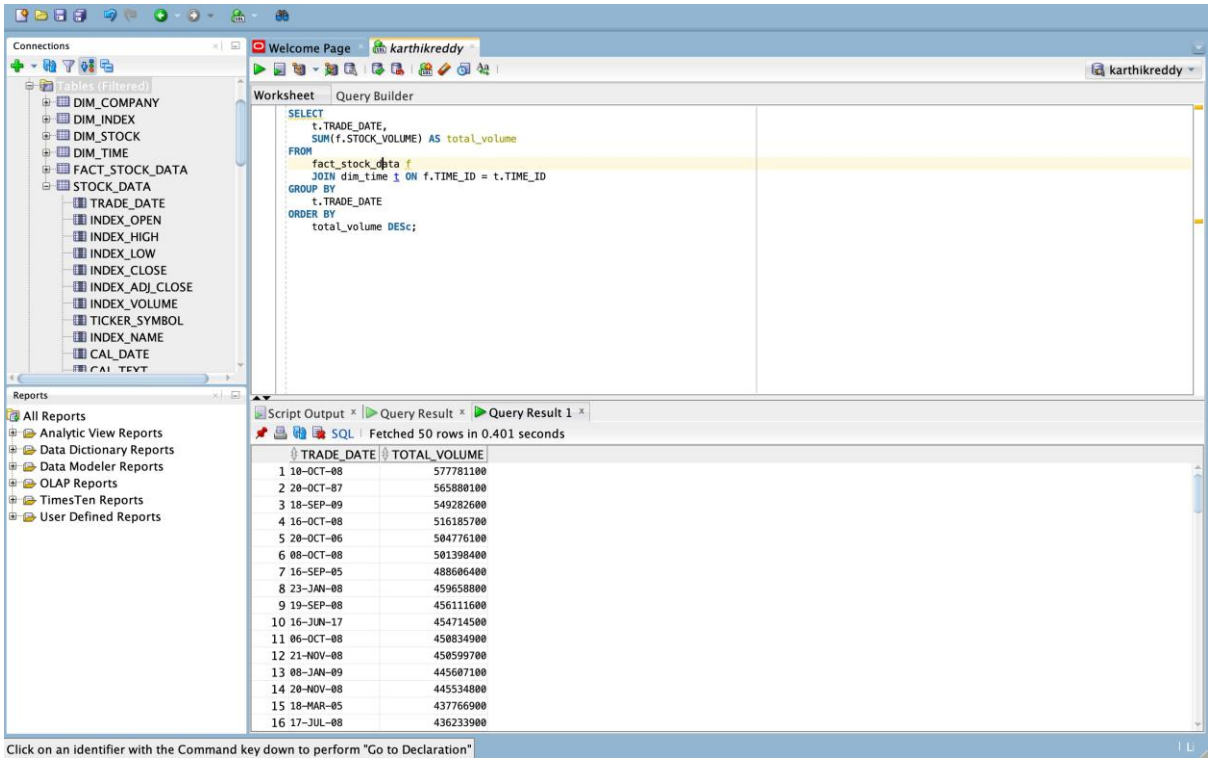
- **Average Closing Price by Day of the Week:**

```
SELECT
   t.DAY_NAME,
   AVG(f.STOCK_CLOSE) AS avg_stock_close
FROM
   fact_stock_data f
   JOIN dim_time t ON f.TIME_ID = t.TIME_ID
GROUP BY    t.DAY_NAME
ORDER BY
   CASE t.DAY_NAME
      WHEN 'Monday' THEN 1
      WHEN 'Tuesday' THEN 2
      WHEN 'Wednesday' THEN 3
      WHEN 'Thursday' THEN 4
      WHEN 'Friday' THEN 5
      WHEN 'Saturday' THEN 6
      WHEN 'Sunday' THEN 7
   END;
```



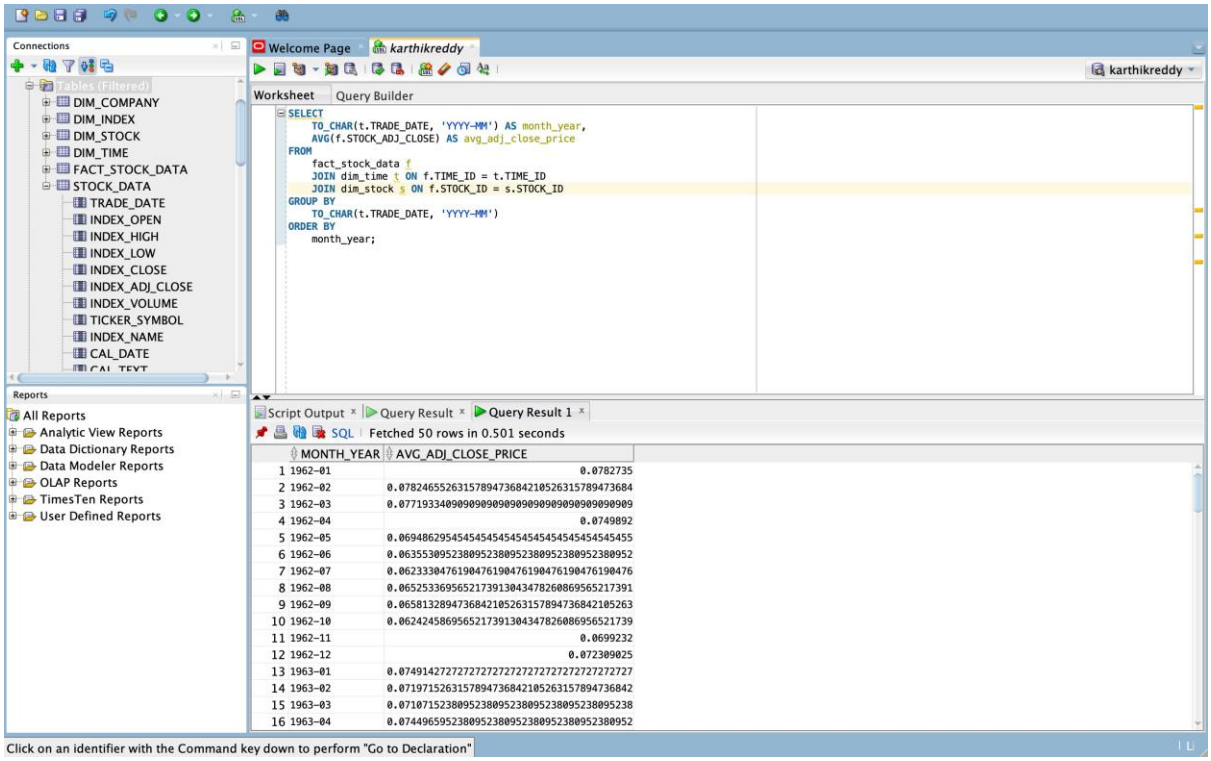
- **Highest Trading Days by Volume:**

```
SELECT
   t.TRADE_DATE,
   SUM(f.STOCK_VOLUME) AS total_volume
FROM
   fact_stock_data f
   JOIN dim_time t ON f.TIME_ID = t.TIME_ID
GROUP BY
   t.TRADE_DATE
ORDER BY
   total_volume DESC
LIMIT 10;
```



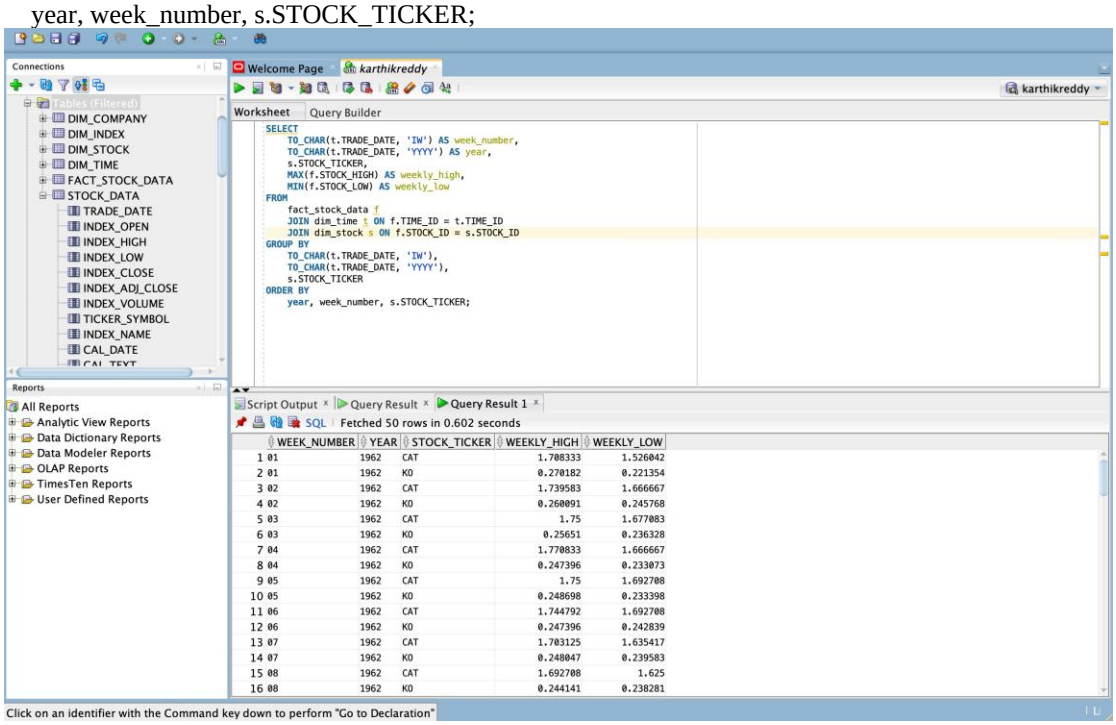
- **Monthly Average Adjusted Closing Price for a Specific Stock:**

```
SELECT
   TO_CHAR(t.TRADE_DATE, 'YYYY-MM') AS month_year,
   AVG(f.STOCK_ADJ_CLOSE) AS avg_adj_close_price
FROM
   fact_stock_data f
   JOIN dim_time t ON f.TIME_ID = t.TIME_ID
   JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
WHERE
   s.STOCK_TICKER = 'AAPL'  -- Example: Replace with specific stock ticker
GROUP BY
   TO_CHAR(t.TRADE_DATE, 'YYYY-MM')
ORDER BY
   month_year;
```



- **Weekly High and Low Prices for All Stocks:**

```
SELECT
   TO_CHAR(t.TRADE_DATE, 'IW') AS week_number,
   TO_CHAR(t.TRADE_DATE, 'YYYY') AS year,
   s.STOCK_TICKER,
   MAX(f.STOCK_HIGH) AS weekly_high,
   MIN(f.STOCK_LOW) AS weekly_low
FROM
   fact_stock_data f
   JOIN dim_time t ON f.TIME_ID = t.TIME_ID
   JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
GROUP BY
   TO_CHAR(t.TRADE_DATE, 'IW'),
   TO_CHAR(t.TRADE_DATE, 'YYYY'),
   s.STOCK_TICKER
ORDER BY
   year, week_number, s.STOCK_TICKER;
```



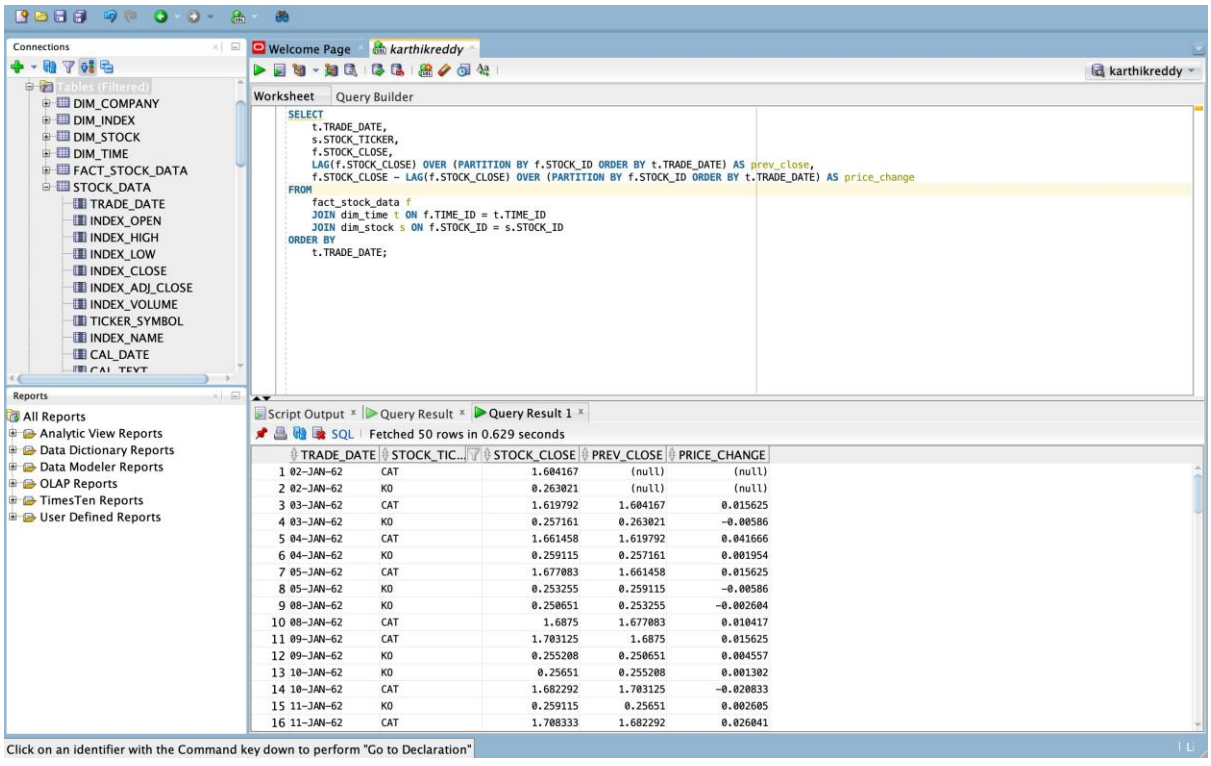
1. LAG and LEAD Functions

- **LAG:** Retrieves data from a previous row in the result set.
- **LEAD:** Retrieves data from a subsequent row in the result set.

Example: Calculate the Change in Stock Price Over Time

```
SELECT
    t.TRADE_DATE,
    s.STOCK_TICKER,
    f.STOCK_CLOSE,
    LAG(f.STOCK_CLOSE) OVER (PARTITION BY f.STOCK_ID ORDER BY t.TRADE_DATE) AS
prev_close,
    f.STOCK_CLOSE - LAG(f.STOCK_CLOSE) OVER (PARTITION BY f.STOCK_ID ORDER BY
t.TRADE_DATE) AS price_change
FROM
    fact_stock_data f
    JOIN dim_time t ON f.TIME_ID = t.TIME_ID
    JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
ORDER BY
    t.TRADE_DATE;
```

This query calculates the change in stock price (price_change) compared to the previous trading day (LAG(f.STOCK_CLOSE)).



2. NTILE Function

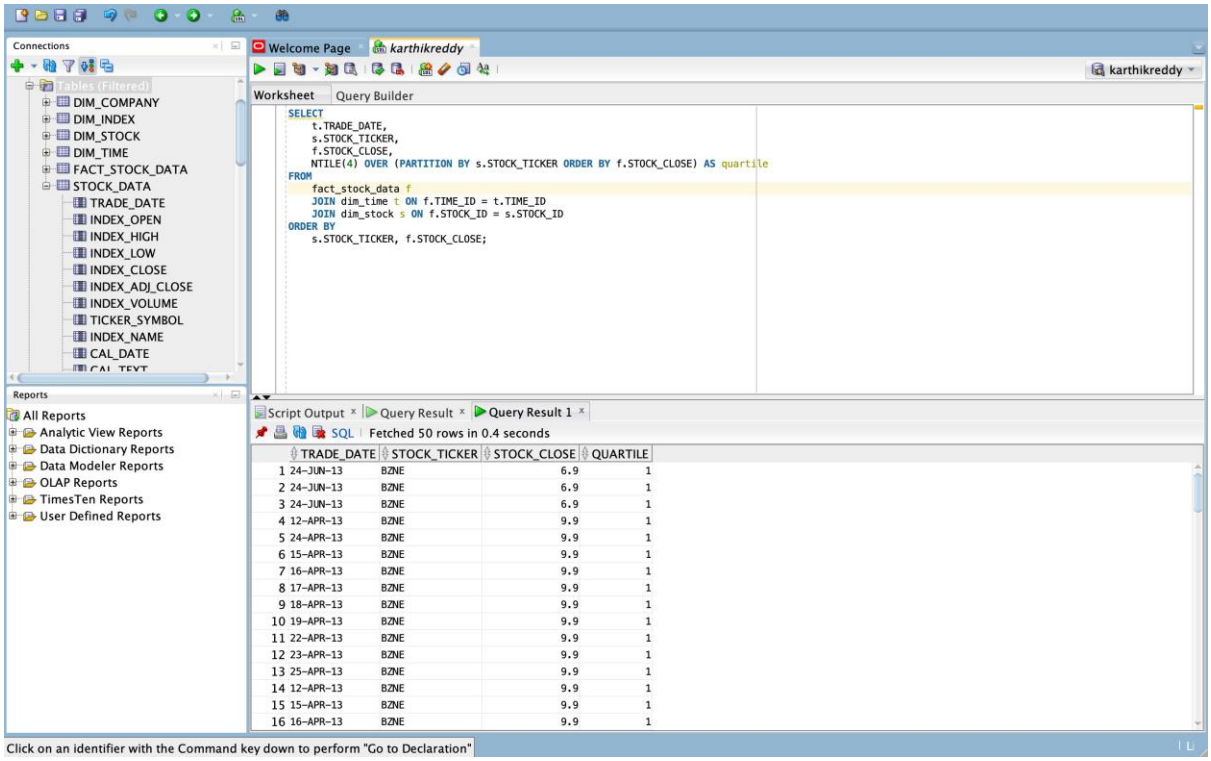
- **NTILE:** Divides the result set into a specified number of groups or "buckets".

Example: Assign Quartiles Based on Stock Closing Price

```
SELECT
    t.TRADE_DATE,
    s.STOCK_TICKER,
    f.STOCK_CLOSE,
```

```
    NTILE(4) OVER (PARTITION BY s.STOCK_TICKER ORDER BY f.STOCK_CLOSE) AS quartile
FROM
    fact_stock_data f
    JOIN dim_time t ON f.TIME_ID = t.TIME_ID
    JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
ORDER BY
    s.STOCK_TICKER, f.STOCK_CLOSE;
```

In this query, NTILE(4) partitions the data by STOCK_TICKER and assigns quartiles based on the STOCK_CLOSE values.



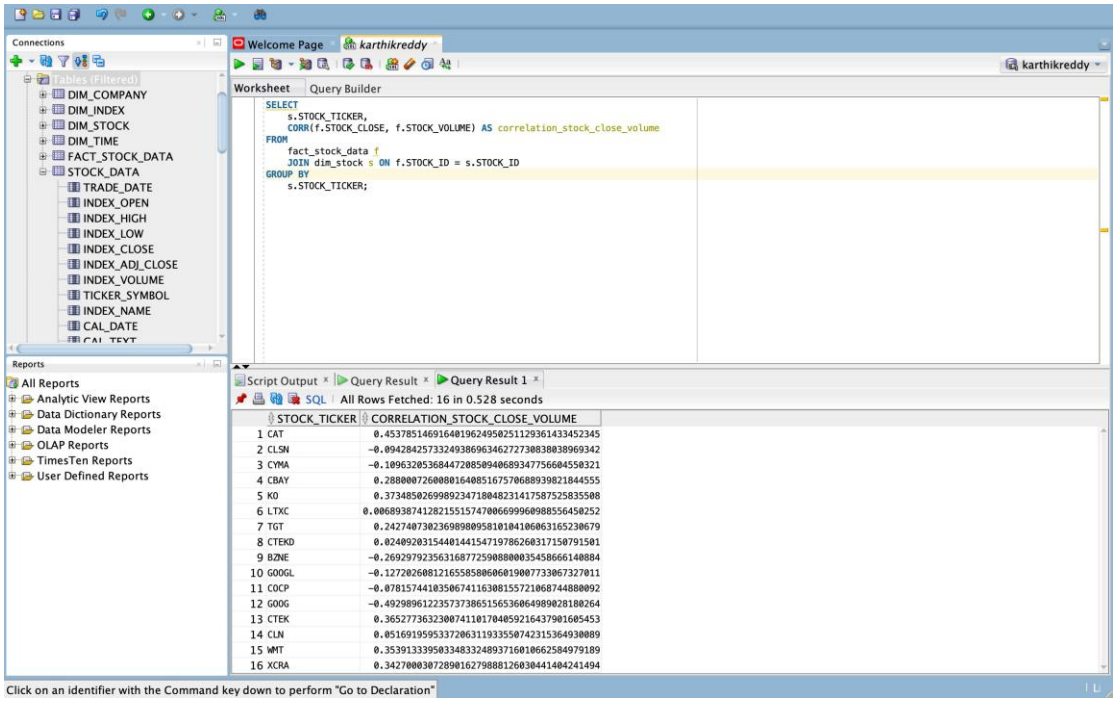
3. CORR Function

- **CORR:** Calculates the correlation coefficient between two numeric columns.

Example: Calculate Correlation Between Stock Prices and Volumes

```
SELECT
    s.STOCK_TICKER,
    CORR(f.STOCK_CLOSE, f.STOCK_VOLUME) AS correlation_stock_close_volume
FROM
    fact_stock_data f
    JOIN dim_stock s ON f.STOCK_ID = s.STOCK_ID
GROUP BY
    s.STOCK_TICKER;
```

This query computes the correlation coefficient (correlation_stock_close_volume) between STOCK_CLOSE and STOCK_VOLUME for each stock ticker.

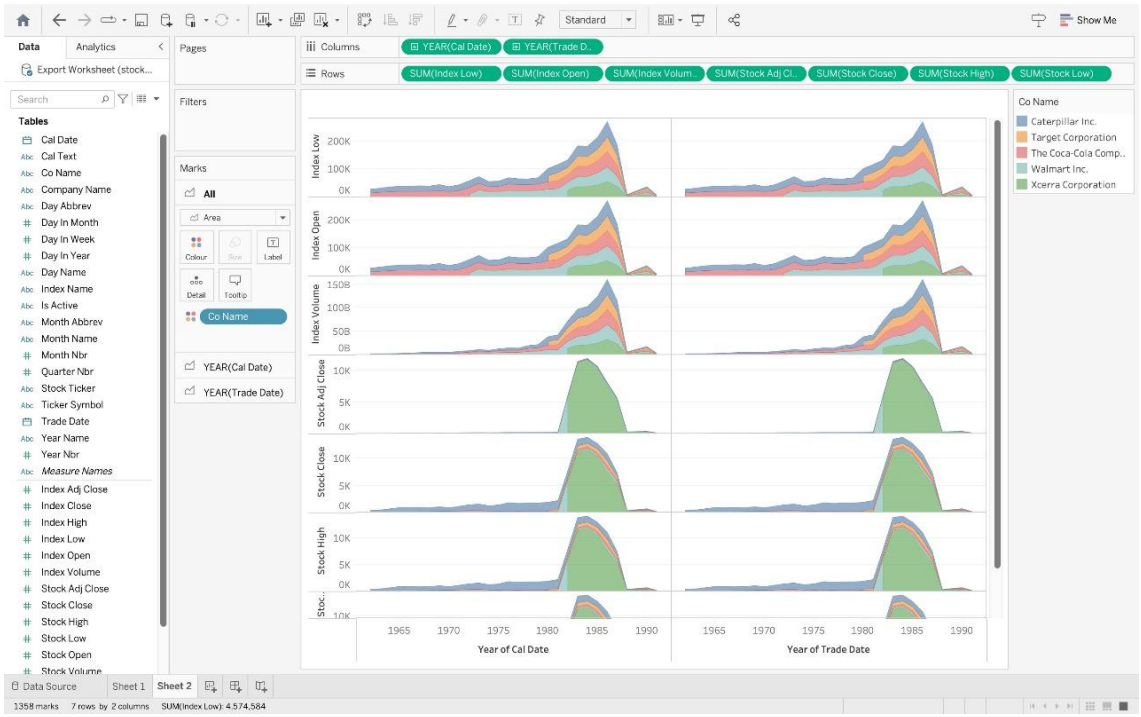


Applications

- **Price Change Analysis:** Use LAG/LEAD to analyze trends and changes over time, such as daily price changes or sequential movements in stock data.
- **Segmentation and Ranking:** NTILE can help categorize data into groups based on specified criteria, useful for quartiles, percentiles, or other segmentations.
- **Correlation Analysis:** CORR facilitates understanding relationships between variables, aiding in identifying factors influencing stock performance.

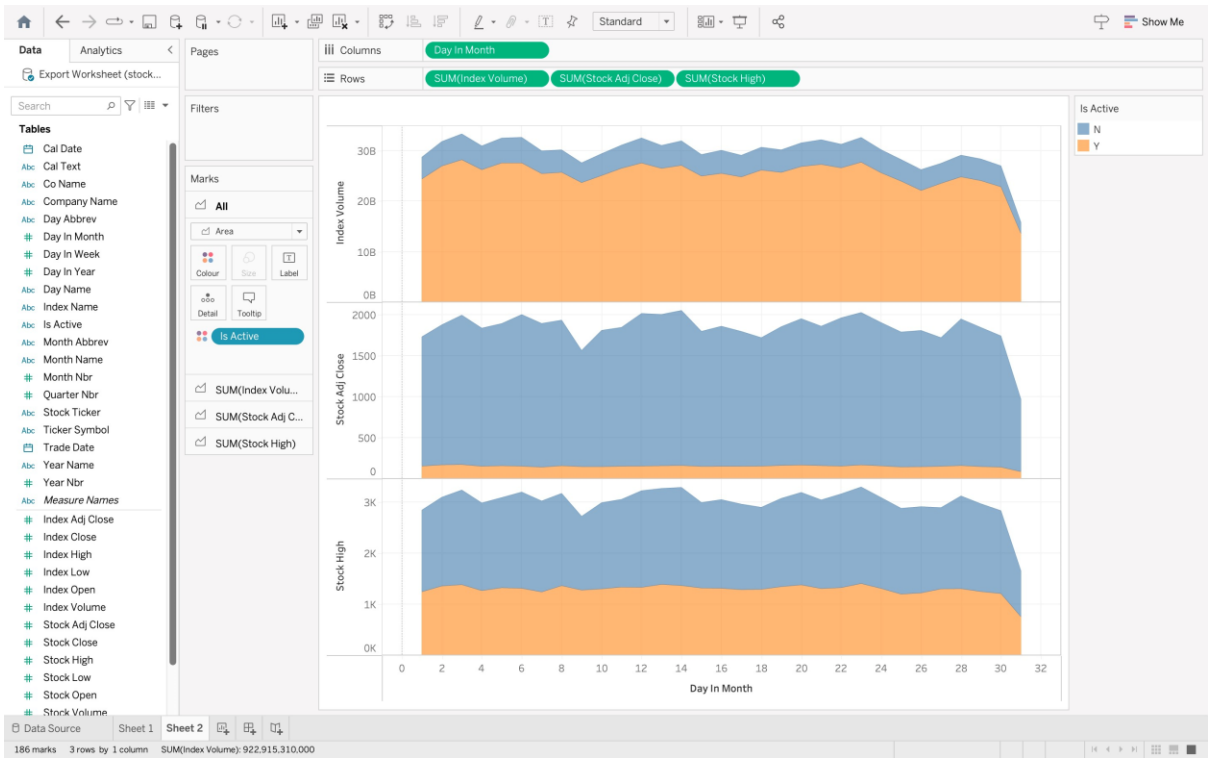
Data Visualization

a) Stock Performance Over Time:



This visualization showcases the trends in various stock metrics, including Index Low, Index Open, Index Volume, Stock Adjusted Close, Stock Close, Stock High, and Stock Low.

b) Stock Market Performance by Day in Month

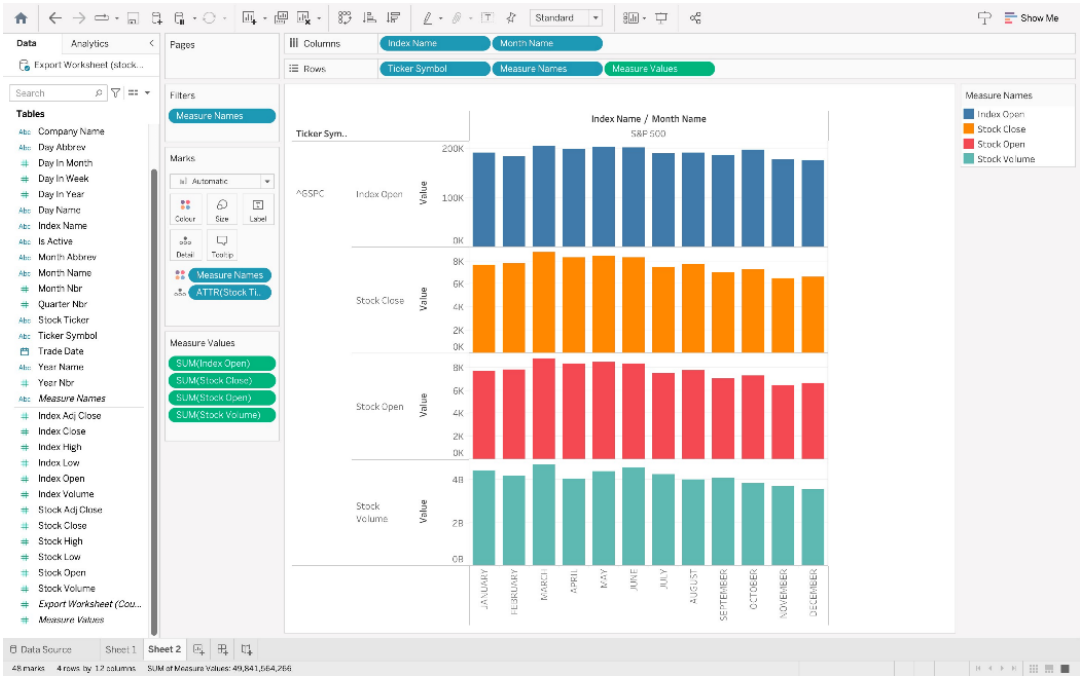


- To further analyze the stock market data, an area chart was created using Tableau to visualize the volume, adjusted closing prices, and high prices of stocks over the days in a month. This visualization helps in understanding the daily performance and activity levels in the stock market.

- The area chart of stock market performance by day in the month is a valuable tool for visualizing and analyzing daily market activities. It aids investors and analysts in understanding trading patterns, market stability, and the level of participation, thereby supporting informed decision-making in stock investments.

c) Monthly Performance of S&P 500

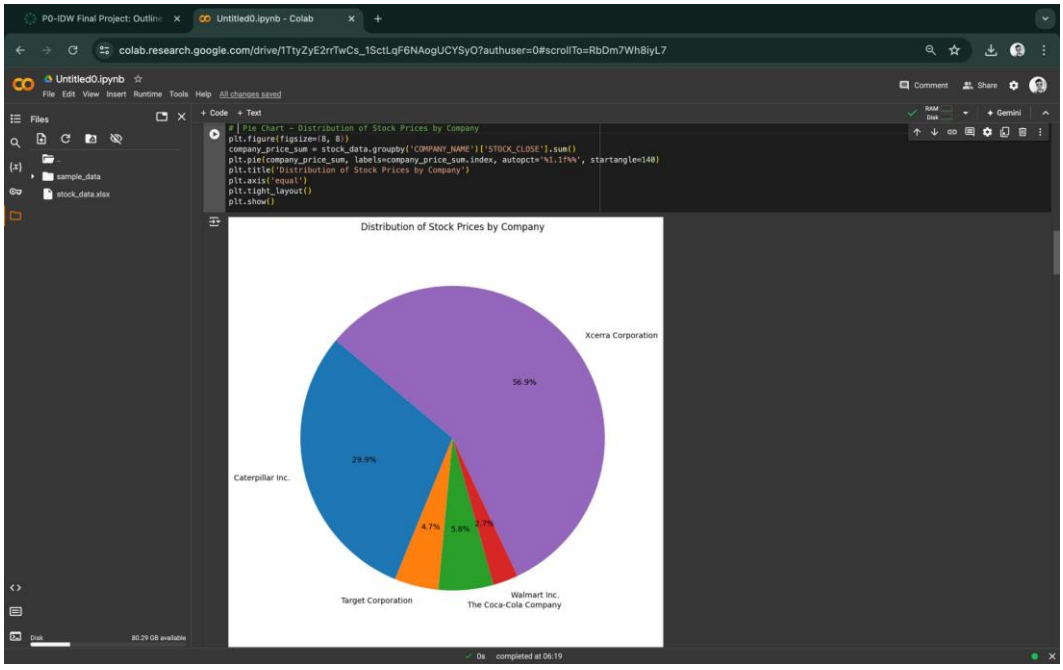
- To analyze the monthly performance of the S&P 500 index, a bar chart was created using Tableau. This visualization helps in comparing different measures such as index open, stock close, stock open, and stock volume across the months.
- This chart is useful for investors and analysts in understanding the stability of opening and closing prices, as well as the fluctuations in trading volumes, thereby supporting better financial decisions and strategies.



d) Distribution of Stock Prices by Company

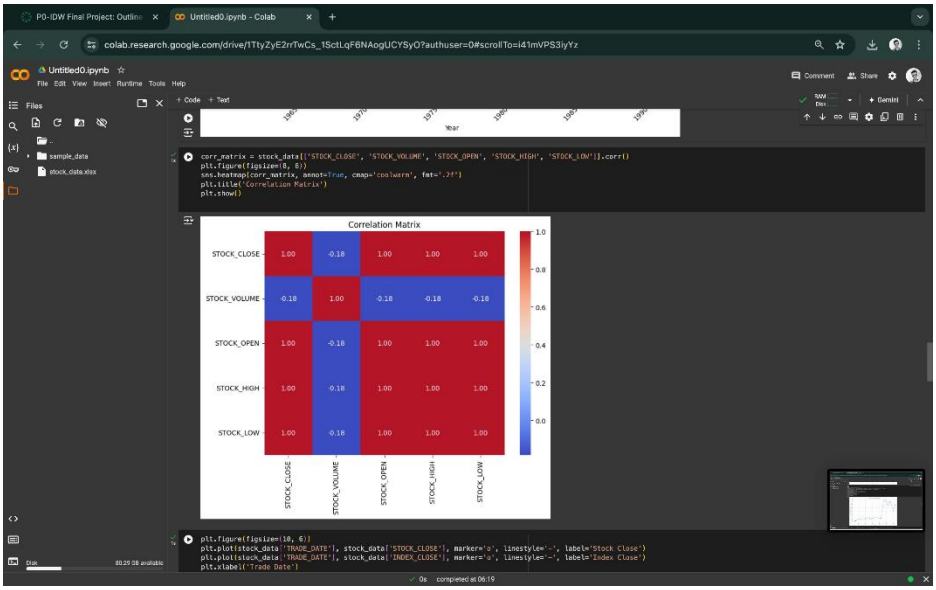
- To understand the distribution of stock prices among different companies, a pie chart was created using Python. This visualization helps in identifying the proportion of stock prices contributed by each company in the dataset.

The pie chart illustrates the distribution of stock prices across various companies, providing a clear visual representation of each company's share in the total stock prices.



e) Correlation Matrix of Stock Variables

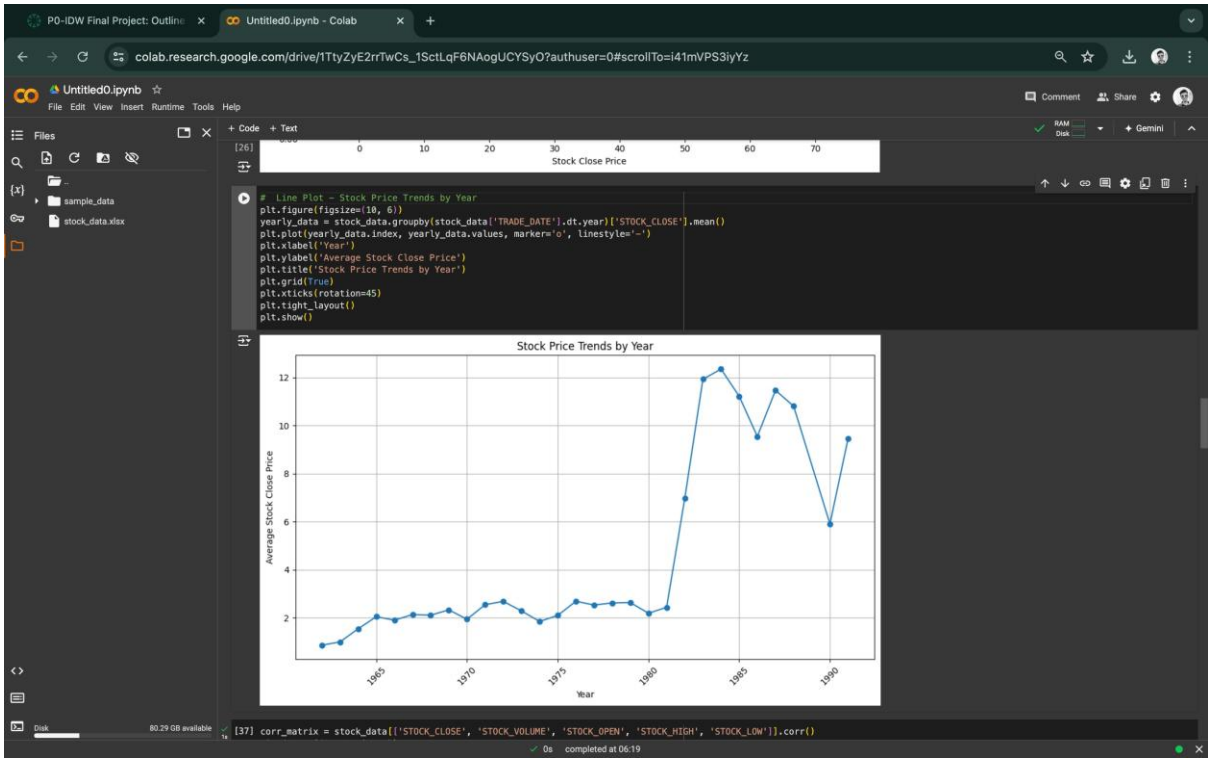
- To analyse the relationships between different stock variables, a correlation matrix was created using Python. This visualization helps in understanding how various stock metrics are correlated with each other.



- The heatmap illustrates the correlation matrix of stock close prices, volumes, open prices, high prices, and low prices, providing a visual representation of the strength and direction of correlations.

f) Stock Price Trends by Year

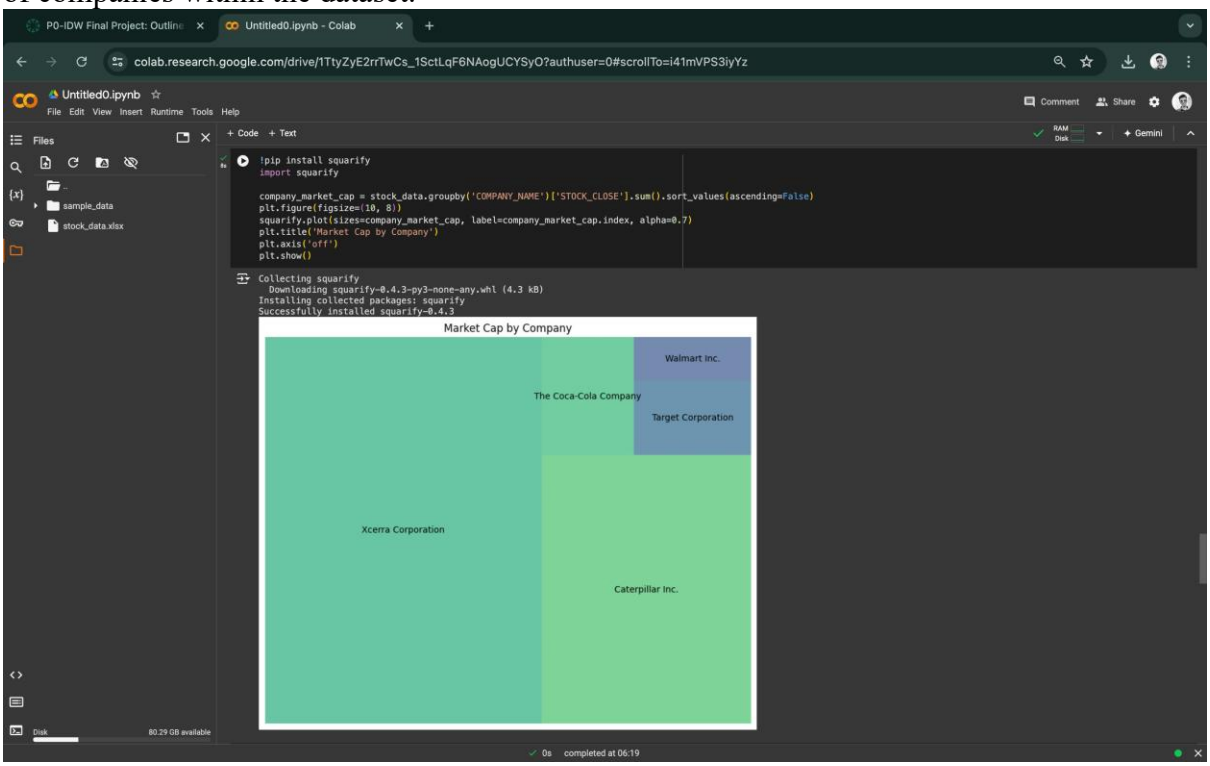
- To analyze the trends in stock prices over the years, a line plot was created using Python. This visualization helps in understanding the changes in average stock close prices over a span of several years.



- The line plot illustrates the average stock close prices by year, providing a visual representation of how stock prices have trended over time.

g) Market Cap by Company

- To visualize the market capitalization of different companies, a treemap was created using Python. This visualization helps in understanding the relative market cap sizes of companies within the dataset.



- The treemap illustrates the market capitalization of various companies, providing a clear and proportional visual representation of each company's market cap.

References:

1. https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/
2. https://www.vertabelo.com/blog/a-beginners-guide-to-data-modeling/
3. https://chartio.com/learn/data-visualization/what-is-data-visualization/