**Data-Driven Stock Analysis: Organizing, Cleaning, and Visualizing Market Trends**

Documentation: Libraries and Their Usage for Stock Market Project

1. ***import yaml***

**Purpose**: Used for reading and writing YAML files.

1. ***import pandas as pd***

**Purpose**: Essential for data manipulation and analysis. Provides data structures like DataFrames for handling structured data.

1. ***import matplotlib.pyplot as plt***

**Purpose**: A plotting library used to create static, animated, and interactive visualizations.

1. ***import seaborn as sns***

**Purpose**: Built on top of Matplotlib, it simplifies the creation of complex visualizations and statistical plots.

1. ***import sqlalchemy***

**Purpose**: Provides a high-level interface for connecting to relational databases (like MySQL) using an Object-Relational Mapper (ORM).

1. ***import pymysql***

**Purpose**: A Python library that enables interaction with MySQL databases.

1. ***import mysql.connector***

**Purpose**: Another MySQL connector library that facilitates connecting to MySQL databases.

1. ***import os***

**Purpose**: Provides functions for interacting with the operating system (e.g., file handling, environment variables).

1. ***import csv***

**Purpose**: Provides functionality for reading and writing CSV files.

***10 from collections import defaultdict***

**Purpose**: A subclass of Python’s built-in dict that provides a default value for missing keys.

**STEP 1**

CONVERTING YAML FILE INTO CSV FILE FOR EACH TICKER:

**Input**: YAML files containing stock market data organized by Ticker.

**Output**: CSV files created for each Ticker, containing the corresponding stock market data.

**Goal**: To process YAML files, group data by Ticker, and save each group into a separate CSV file for easier analysis or visualization.

**1. Directory Traversal & YAML File Processing:**

* **os.walk(base\_path)**: Recursively walks through the directory structure starting from base\_path, searching for YAML files.
* **YAML File Parsing**: Each found YAML file is opened, and its contents are loaded using yaml.safe\_load.

>>>for root, dirs, files in os.walk(base\_path):

>>>for file in files:

>>>if file.endswith(".yaml"):

>>>file\_path = os.path.join(root, file)

>>>with open(file\_path, 'r') as f:

>>>data = yaml.safe\_load(f)

**2. Data Grouping by Ticker:**

* **Ticker Categorization**: For each entry in the YAML file (assumed to be a dictionary), the Ticker value is extracted and used as a key in the ticker\_data dictionary.
* **Data Collection**: Entries are grouped into lists under their corresponding Ticker.

>>>ticker = entry['Ticker']

>>>ticker\_data[ticker].append(entry)

**3. Output Directory Creation:**

* **Ensure Output Directory**: The output folder is created within the base\_path to store the generated CSV files.
* **os.makedirs(output\_path, exist\_ok=True)**: Ensures the output directory exists or is created.

>>>output\_path = os.path.join(base\_path, "output")

>>>os.makedirs(output\_path, exist\_ok=True)

**4. Write Data to CSV:**

* **CSV Writing**: For each Ticker, a CSV file is created. Data for each Ticker is written into the CSV file using Python's csv.DictWriter to handle the dictionary format.
* **Fieldnames**: The keys of the first dictionary in each Ticker's data are used as the CSV headers.

**>>>with open(output\_file, 'w', newline='', encoding='utf-8') as csvfile:**

**>>>writer = csv.DictWriter(csvfile, fieldnames=entries[0].keys())**

**>>>writer.writeheader()**

**>>>writer.writerows(entries)**

**5. Final Output:**

* Once all YAML files are processed, the function prints a success message and specifies the directory where the CSV files have been saved.

**>>>print(f"Data successfully transformed into CSV files at {output\_path}")**

**STEP 2**

CALCULATING TOP 10 GREEN STOCKS AND LOSS STOCKS:

**Input**: CSV files with stock data (columns: Ticker, close, date, open, volume).

**Output**:

1. A list of the top 10 green (gain) stocks based on yearly return.
2. A list of the top 10 loss (decline) stocks based on yearly return.
3. A market summary with statistics (green/red stocks, average price, average volume).

**Goal**: To analyze stock performance, identify top performers and losers, and provide an overall market summary.

**1. Reading and Processing CSV Files:**

* The function **analyze\_stocks(output\_path)** scans the specified folder (output\_path) for all CSV files.
* For each CSV, it loads the data into a Pandas DataFrame and ensures the presence of required columns: Ticker, close, date, open, and volume.
* If the required columns are found, the data is processed further.

>>>data = pd.read\_csv(file\_path)

>>>if all(col in data.columns for col in ['Ticker', 'close', 'date','open', 'volume']):

**2. Data Processing:**

* **Sorting and Date Conversion**: The stock data is sorted by the date column, and the date column is converted to datetime format.
* **Yearly Return Calculation**: The yearly return for each stock is calculated by comparing the first and last closing prices of the stock in the dataset. The formula used is:

Yearly Return=(last\_close−first\_closefirst\_close)×100\text{Yearly Return} = \left( \frac{\text{last\\_close} - \text{first\\_close}}{\text{first\\_close}} \right) \times 100Yearly Return=(first\_closelast\_close−first\_close​)×100

>>>data['date'] = pd.to\_datetime(data['date'])

>>>data = data.sort\_values(by='date')

>>>first\_close = data['close'].iloc[0]

>>>last\_close = data['close'].iloc[-1]

>>>data['Yearly Return'] = ((last\_close - first\_close) / first\_close) \* 100

**3. Data Aggregation:**

* **Combining Data**: All processed data from each CSV file is combined into a single DataFrame (full\_data) using pd.concat.
* **Top 10 Green and Loss Stocks**: The top 10 "green stocks" (highest positive yearly return) and top 10 "loss stocks" (highest negative yearly return) are identified using nlargest and nsmallest.

>>>full\_data = pd.concat(all\_data, ignore\_index=True)

>>>top\_10\_green\_stocks = full\_data.nlargest(10, 'Yearly Return')

>>>top\_10\_loss\_stocks = full\_data.nsmallest(10, 'Yearly Return')

**4. Market Summary:**

* **Market Overview**: The script calculates a market summary, which includes:
  + The count of green (positive return) and red (negative or zero return) stocks.
  + The average price and average volume of all stocks.

>>>market\_summary = {

'Green Stocks': (full\_data['Yearly Return'] > 0).sum(),

'Red Stocks': (full\_data['Yearly Return'] <= 0).sum(),

'Average Price': full\_data['Price'].mean(),

'Average Volume': full\_data['volume'].mean()

}

**5. Results:**

* The function returns three results:
  1. **Top 10 Green Stocks**: Stocks with the highest positive yearly returns.
  2. **Top 10 Loss Stocks**: Stocks with the lowest yearly returns.
  3. **Market Summary**: A dictionary containing the market statistics.

**>>>return top\_10\_green\_stocks[['Ticker', 'Yearly Return', 'Price', 'volume']], top\_10\_loss\_stocks[['Ticker', 'Yearly Return', 'Price', 'volume']], market\_summary**

**6. Execution:**

* The script runs the analysis and prints the top 10 green stocks, top 10 loss stocks, and the market summary to the console.

>>>top\_10\_green\_stocks, top\_10\_loss\_stocks, market\_summary = analyze\_stocks(output\_path)

**STEP 3**

Visualizing the volatility of each stock over the past year by calculating the standard deviation of daily returns:

**Input**: CSV files with stock data containing columns like date and close.

**Output**:

1. A CSV file (top\_10\_volatility.csv) containing the top 10 most volatile stocks based on their daily returns.
2. A bar chart visualizing the volatility of the top 10 stocks.

**Goal**: To identify and visualize the top 10 most volatile stocks over the past year, which can be useful for risk analysis and trading strategies.

**1. calculate\_volatility Function:**

* **Purpose**: This function calculates the volatility of stocks based on their daily price changes and saves the top 10 most volatile stocks to a CSV file.
* **Parameters**:
  + output\_path (str): The directory containing stock data CSV files.
  + save\_path (str): The path where the output CSV file with the top 10 volatile stocks will be saved (default is "top\_10\_volatility.csv").
* **Process**:
  + **Load Stock Data**: The function iterates through all CSV files in the specified directory (output\_path).
  + **Filter Data for Last Year**: Each stock file is filtered to include only data from the past year.
  + **Calculate Daily Returns**: The percentage change in the closing price is calculated for each stock.
  + **Volatility Calculation**: The standard deviation of daily returns is calculated to measure volatility.
  + **Top 10 Volatile Stocks**: The stocks are sorted by volatility (standard deviation) in descending order, and the top 10 are selected.
  + **Save to CSV**: The top 10 volatile stocks are saved as a CSV file.

python

Copy code

>>>def calculate\_volatility(output\_path, >>>save\_path="top\_10\_volatility.csv"):

**2. visualize\_volatility Function:**

* **Purpose**: Visualizes the volatility of the top 10 most volatile stocks using a bar chart.
* **Parameters**: Takes the top\_10\_volatility DataFrame containing the stock symbols and their standard deviation (volatility).
* **Process**:
  1. The function generates a bar chart with stock symbols on the x-axis and the standard deviation (volatility) on the y-axis.
  2. The bars are colored pink with black edges, and the chart is customized for better readability.

>>>def visualize\_volatility(top\_10\_volatility):

**3. Execution:**

* **calculate\_volatility Call**: The calculate\_volatility function is executed with the given output\_path (the directory of stock CSVs) and save\_path (location to save the top 10 volatile stocks CSV).
* **visualize\_volatility Call**: After calculating the top 10 volatile stocks, the visualize\_volatility function generates a bar chart to display the results.

**>>>save\_path = "top\_10\_volatility.csv"**

**>>>top\_10\_volatility = calculate\_volatility(output\_path, save\_path)**

**>>>visualize\_volatility(top\_10\_volatility)**

**STEP 4**

Showing the cumulative return of each stock from the beginning of the year to the end:

**Input**: CSV files containing stock data (columns like date, close).

**Output**:

1. A CSV file (cumulative\_return.csv) with the top 5 performing stocks by cumulative return.
2. A line plot visualizing the cumulative returns for the top 5 performing stocks.

**Goal**: To calculate and visualize the cumulative returns of stocks over time, helping to identify and compare the top-performing stocks over a given period.

**1. calculate\_cumulative\_return Function:**

* **Purpose**: This function calculates the cumulative return for each stock, identifies the top 5 performing stocks, and saves the results in a CSV file.
* **Parameters**:
  + output\_path (str): The directory containing stock data CSV files.
  + save\_path (str): The path where the cumulative return results will be saved as a CSV file (default is "cumulative\_return.csv").
* **Process**:
  + **Load Stock Data**: The function iterates over all CSV files in the given output\_path and loads them into a DataFrame.
  + **Calculate Cumulative Return**: For each stock, daily returns are calculated using the percentage change in the close price. The cumulative return is then computed by multiplying the daily returns cumulatively over time.
  + **Store Results**: The final cumulative return for each stock is stored, along with the cumulative return trend over time for each stock.
  + **Top 5 Performing Stocks**: The stocks are sorted by their final cumulative return, and the top 5 are selected.
  + **Save Data**: The results (top 5 performing stocks) are saved to a CSV file.

>>>def calculate\_cumulative\_return(output\_path, >>>save\_path="cumulative\_return.csv"):

**2. visualize\_cumulative\_return Function:**

* **Purpose**: This function generates a line plot to visualize the cumulative return trends of the top 5 performing stocks.
* **Parameters**:
  + top\_5 (pd.DataFrame): A DataFrame containing the top 5 stocks by cumulative return.
  + return\_trends (dict): A dictionary containing cumulative return trends for each stock.
* **Process**:
  + The function plots the cumulative return over time for each of the top 5 stocks using matplotlib.
  + Each stock's cumulative return is plotted as a line on the same graph, with the x-axis representing the date and the y-axis representing the cumulative return.
  + The plot is customized with labels, a title, a legend, and grid lines for clarity.

>>>def visualize\_cumulative\_return(top\_5, return\_trends):

**3. Execution:**

* **calculate\_cumulative\_return Call**: The function is executed with the given output\_path (directory containing the stock CSV files) and save\_path (the location to save the cumulative return CSV).
* **visualize\_cumulative\_return Call**: After identifying the top 5 performing stocks, the cumulative return trends are visualized in a line plot.

**>>>save\_path = "cumulative\_return.csv"**

**>>>top\_5\_stocks, trends = calculate\_cumulative\_return(output\_path, save\_path)**

**>>>visualize\_cumulative\_return(top\_5\_stocks, trends)**

**STEP 5**

Calculating a breakdown of stock performance by sector:

**Input**: CSV files containing stock data with columns such as date and close.

**Output**:

1. A CSV file (sector\_performance.csv) containing the average yearly return for each sector.
2. A bar chart visualizing the average yearly return for each sector.

**Goal**: To analyze and compare the performance of different sectors based on their average yearly stock returns, helping investors or analysts assess sector performance.

**1. calculate\_sector\_performance Function:**

* **Purpose**: This function calculates the average yearly return for each sector and saves the results as a CSV file.
* **Parameters**:
  + output\_path (str): The directory containing the stock data CSV files.
  + save\_csv\_path (str): The path where the sector performance data will be saved as a CSV file (default is "sector\_performance.csv").
* **Process**:
  + **Sector Mapping**: A dictionary (sector\_mapping) maps each stock ticker to its respective sector (e.g., 'TCS' to 'Software', 'HDFCBANK' to 'Banking').
  + **Load Stock Data**: The function loads all stock data CSV files from the specified directory (output\_path), assigns each stock its sector using the sector\_mapping dictionary, and concatenates them into a single DataFrame.
  + **Calculate Yearly Returns**: For each stock, the function calculates the yearly return using the percentage change in the stock’s close price.
  + **Calculate Sector Average Return**: The function groups the data by Sector and calculates the average yearly return for each sector.
  + **Save to CSV**: The average yearly return for each sector is saved to a CSV file (sector\_performance.csv).

>>>def calculate\_sector\_performance(output\_path, save\_csv\_path="sector\_performance.csv"):

**2. visualize\_sector\_performance Function:**

* **Purpose**: This function visualizes the average yearly return for each sector as a bar chart.
* **Parameters**:
  + sector\_avg\_returns (pd.Series): A pandas Series containing the average yearly return for each sector.
* **Process**:
  + The function creates a bar chart using matplotlib, where the x-axis represents the sectors, and the y-axis represents the average yearly returns for each sector.
  + The bars are colored pink with black edges, and the chart is customized with labels, a title, and a rotated x-axis for readability.

>>>def visualize\_sector\_performance(sector\_avg\_returns):

**3. Execution:**

* **calculate\_sector\_performance Call**: The function is executed with the given output\_path (directory containing stock CSV files) and save\_csv\_path (path to save the sector performance data).
* **visualize\_sector\_performance Call**: After calculating the sector performance, the average yearly returns for each sector are visualized in a bar chart.

>>>sector\_avg\_returns = calculate\_sector\_performance(output\_path, save\_csv\_path="sector\_performance.csv")

>>>visualize\_sector\_performance(sector\_avg\_returns)

**STEP 6**

Visualizing the correlation between the stock prices of different companies:

**Input**: CSV files containing stock data with columns like date and close.

**Output**:

1. A CSV file (stock\_correlation.csv) containing the correlation matrix of stock returns.
2. A heatmap visualization of the correlation matrix.

**Goal**: To analyze the relationships between different stocks by calculating and visualizing how their closing prices are correlated with each other. This helps in identifying which stocks tend to move together or move in opposite directions over time.

**1. calculate\_correlation\_matrix Function:**

* **Purpose**: This function calculates the correlation matrix for the percentage change in closing prices of multiple stocks and saves it as a CSV file.
* **Parameters**:
  + output\_path (str): The directory containing the stock data CSV files.
  + save\_csv\_path (str): The path to save the correlation matrix as a CSV file (default is "stock\_correlation.csv").
* **Process**:
  + **Load Stock Data**: The function iterates over all CSV files in the specified output\_path, reads each stock's data, and extracts the close prices, setting the date column as the index.
  + **Combine Data**: The stock data is combined into a single DataFrame, where each column represents the closing prices of a different stock.
  + **Calculate Percentage Change**: The percentage change in the closing prices for each stock is calculated to standardize the data.
  + **Correlation Matrix**: The correlation matrix is computed using the .corr() method on the percentage change of the closing prices.
  + **Save to CSV**: The correlation matrix is saved as a CSV file (stock\_correlation.csv).

>>>def calculate\_correlation\_matrix(output\_path, save\_csv\_path="stock\_correlation.csv"):

**2. visualize\_correlation\_matrix Function:**

* **Purpose**: This function visualizes the correlation matrix as a heatmap.
* **Parameters**:
  + correlation\_matrix (pd.DataFrame): A pandas DataFrame containing the correlation matrix of stock prices.
* **Process**:
  + **Heatmap Plot**: The correlation matrix is visualized using a heatmap, where each cell represents the correlation between two stocks.
  + **Customization**: The heatmap is customized with labels, color scales, and annotations to make it more informative. The coolwarm color palette is used to highlight positive and negative correlations.

>>>def visualize\_correlation\_matrix(correlation\_matrix):

**3. Execution:**

* **calculate\_correlation\_matrix Call**: The function is executed with the given output\_path (directory containing stock CSV files) and save\_csv\_path (path to save the correlation matrix data).
* **visualize\_correlation\_matrix Call**: After calculating the correlation matrix, the matrix is visualized as a heatmap.

**>>>correlation\_csv\_path = "stock\_correlation.csv"**

**>>>correlation\_matrix = calculate\_correlation\_matrix(output\_path, save\_csv\_path=correlation\_csv\_path)**

**>>>visualize\_correlation\_matrix(correlation\_matrix)**

**STEP 7**

Calculating monthly breakdowns of the top-performing and worst-performing stocks:

**Input**: CSV files containing stock data with columns like date and close.

**Output**:

1. A CSV file (gainers\_and\_losers.csv) containing the top 5 gainers and losers for each month, with columns: Year Month, Type (Gainer or Loser), Ticker, and Return (%).
2. Bar charts visualizing the top gainers and losers for each month.

**Goal**: To identify and visualize the **top-performing** and **worst-performing** stocks for each month based on their monthly return percentages, which can help in tracking stock performance trends.

**1. top\_gainers\_and\_losers\_monthwise Function:**

* **Purpose**: The function calculates the **top 5 gainers** and **top 5 losers** for each month based on **monthly returns** of stocks and saves the results to a CSV file.
* **Parameters**:
  + output\_path (str): The directory containing the stock data CSV files.
  + save\_csv\_path (str): The path to save the top gainers and losers data as a CSV file (default is "gainers\_and\_losers.csv").
* **Process**:
  + **Load and Process Stock Data**: Reads all CSV files in the output\_path, assigns each file a ticker (based on the filename), and converts the date column to a Date column.
  + **Calculate Monthly Return**: For each stock, calculates the **monthly return** by comparing the closing price (close) at the beginning and end of the month.
  + **Identify Top Gainers and Losers**:
    - Groups data by YearMonth (year and month), and for each month, finds the **top 5 gainers** and **top 5 losers** based on the highest and lowest monthly return percentages.
  + **Save Results to CSV**: The results, including YearMonth, Type (Gainer or Loser), Ticker, and Return (%), are saved to a CSV file (gainers\_and\_losers.csv).
* **Return**: The function returns a dictionary (top\_gainers\_losers) with the top gainers and losers for each month, structured by YearMonth.

>>>def top\_gainers\_and\_losers\_monthwise(output\_path, save\_csv\_path="gainers\_and\_losers.csv"):

**2. visualize\_top\_gainers\_and\_losers Function:**

* **Purpose**: This function visualizes the top 5 gainers and losers for each month using **bar charts**.
* **Parameters**:
  + top\_gainers\_losers (dict): A dictionary containing the top gainers and losers for each month, as returned by the top\_gainers\_and\_losers\_monthwise function.
* **Process**:
  + **Create Bar Charts**: For each month, two bar charts are created:
    - One for the **top 5 gainers**.
    - One for the **top 5 losers**.
  + **Customization**: Each chart is customized with labels, titles, and color coding (green for gainers and red for losers).
  + **Display the Charts**: The charts are displayed using plt.show() for visual comparison.

>>>def visualize\_top\_gainers\_and\_losers(top\_gainers\_losers):

**3. Execution:**

* **top\_gainers\_and\_losers\_monthwise Call**: The function is executed with the directory output\_path (containing stock CSV files), and the top gainers/losers data is saved to a CSV file (gainers\_and\_losers.csv).
* **visualize\_top\_gainers\_and\_losers Call**: After calculating the top gainers and losers, the function visualizes the results using bar charts.

**>>>top\_gainers\_losers = top\_gainers\_and\_losers\_monthwise(output\_path, save\_csv\_path="gainers\_and\_losers.csv")**

**>>>visualize\_top\_gainers\_and\_losers(top\_gainers\_losers)**

**STEP 8**

Storing the five CSV file to MySQL:

The next step is to store multiple CSV files into a MySQL database. Each CSV file is saved into its corresponding table in the database, creating the tables if they do not already exist.

**Input**: A list of CSV files and corresponding MySQL table names, along with MySQL connection parameters.

**Output**: Data from each CSV file is inserted into the respective table in the MySQL database. If a table doesn't exist, it is created automatically.

**Goal**: To efficiently upload data from multiple CSV files into a MySQL database, ensuring the tables are created if they do not exist and the data is inserted correctly.

**store\_csv\_to\_mysql Function:**

* **Purpose**: This function takes a list of CSV file paths and their corresponding table names, then stores the contents of each CSV file into the specified MySQL database tables.
* **Parameters**:
  + csv\_files (list): A list of file paths to the CSV files that need to be uploaded to the database.
  + table\_names (list): A list of table names in MySQL where each corresponding CSV file will be stored. The number of table names should match the number of CSV files.
  + host (str): The hostname or IP address of the MySQL server.
  + user (str): MySQL username for authentication.
  + password (str): MySQL password for authentication.
  + database (str): The name of the MySQL database where the tables will be created and data will be inserted.
* **Process**:
  + **Validation**: The function checks that the number of CSV files matches the number of table names. If not, it raises a ValueError.
  + **Database Connection**: It establishes a connection to the MySQL database using the provided credentials (host, user, password, database).
  + **Processing Each CSV**:
    - For each CSV file, it reads the data into a pandas DataFrame.
    - It generates a CREATE TABLE query for each table based on the column names in the CSV file and creates the table if it does not already exist.
    - It constructs an INSERT INTO query and inserts each row from the DataFrame into the corresponding table.
  + **Commit and Close**: After all CSV data is inserted into the tables, the function commits the changes to the database and closes the connection.
* **Example Usage**: The function is used to upload five CSV files (top\_10\_volatility.csv, cumulative\_return.csv, etc.) into five corresponding tables (table1, table2, etc.) in the stockmarket database.

>>>csv\_files = [

"top\_10\_volatility.csv", "cumulative\_return.csv", "sector\_performance.csv", "stock\_correlation.csv", "gainers\_and\_losers.csv"

]

>>>table\_names = [

"table1", "table2", "table3", "table4", "table5"

]

>>>store\_csv\_to\_mysql(

csv\_files=csv\_files,

table\_names=table\_names,

host="localhost",

user="root",

password="root",

database="stockmarket"

)

**STEP 9**

Streamlit Dashboard for Displaying MySQL Data:

Creating a simple Streamlit web application that connects to a MySQL database, fetches data from it, and displays it to the user. It allows users to explore the tables in a database and view data from the selected table.

1. **fetch\_data\_from\_mysql(query, host, user, password, database)**:
   * **Purpose**: Connects to the specified MySQL database, executes a given SQL query, and returns the results as a pandas DataFrame.
   * **Parameters**:
     + query (str): The SQL query to be executed on the MySQL database.
     + host, user, password, database (str): MySQL connection parameters (defaults are provided).
   * **Returns**: A pandas DataFrame containing the query results.
   * **Steps**:
     + Establishes a connection to the MySQL database using mysql.connector.
     + Executes the SQL query and fetches the results.
     + Converts the results into a pandas DataFrame with appropriate column names.
     + Closes the MySQL connection before returning the DataFrame.
2. **Streamlit UI**:
   * **Purpose**: Creates an interactive dashboard where users can select a table from the database and view the data.
   * **Steps**:
     + The title and header display "Investors' Dashboard" and the section for available tables.
     + The SHOW TABLES query is executed to list all the tables in the MySQL database (stockmarket by default). This list is shown to the user.
     + A dropdown (selectbox) allows the user to choose a table to view.
     + Once a table is selected, the first 20 rows of the selected table are displayed using the SELECT \* FROM <table> LIMIT 20 SQL query.

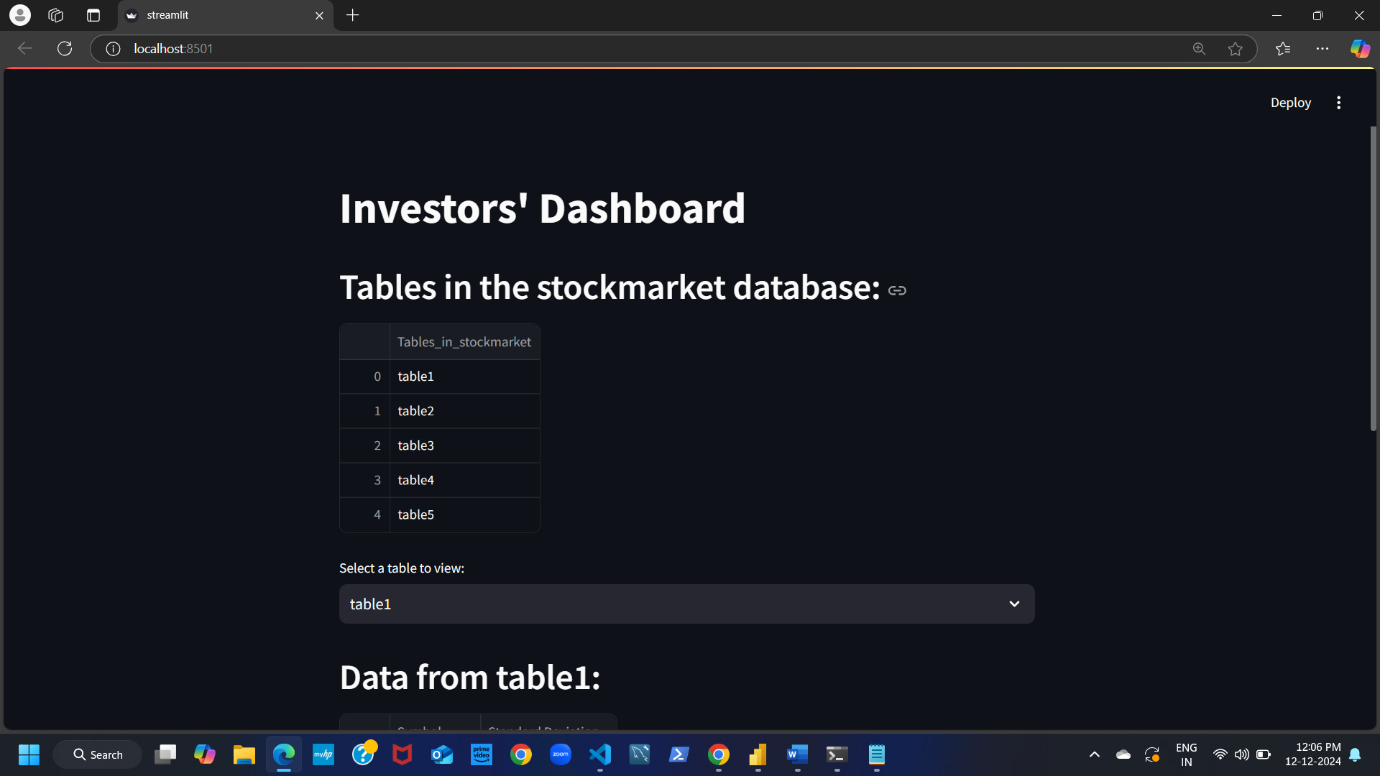
**Key Streamlit Elements:**

* **st.title()**: Displays the title of the dashboard.
* **st.header()**: Displays headers for different sections (e.g., database tables, selected table data).
* **st.write()**: Displays the list of tables in the database.
* **st.selectbox()**: Dropdown list for the user to choose a table.
* **st.dataframe()**: Displays the fetched data from the selected table in a tabular format.

**Example Usage:**

The application will:

1. Show a list of tables in the stockmarket database.
2. Allow users to select one of those tables.
3. Display the first 20 rows of data from the selected table.



**STEP 10**

Importing data into Power BI:

Power BI Desktop is a powerful data analysis and visualization tool that allows users to import data from various sources, including CSV (Comma-Separated Values) files.

**Create Visualizations**

* Drag and drop fields from the **Fields** to create charts, tables, and other visualizations.