Name:

Student ID:

FINANCIAL DATA ANALYSIS

PROJECT REPORT

**Introduction**

Our Python application visualizes stock price analysis for a given date range and stock market. We use the yfinance (Yahoo Finance) library to import stock data and download it as a CSV file. To read the downloaded CSV file and manage the data, we use the Pandas library. Additionally, we use the NumPy library for various operations. After processing the data, we utilize the Matplotlib library to visualize the results.

**Data Collection and Preprocessing**

As previously mentioned, we used yfinance to fetch and download stock data to a csv file. After downloading the csv file,

**Read the csv file into a Pandas Dataframe.**

**A computer screen shot of text

Description automatically generated**

* Reads the CSV file into a Pandas DataFrame called data.
* Converts the 'Date' column to a datetime format to ensure proper date handling.
* Sets the 'Date' column as the DataFrame index for easy time-series analysis.

**Exception Handling: File and Parsing Errors**

**A computer screen shot of text

Description automatically generated**

* If the file is not found, a message is printed, and the function returns **None**
* If the file is empty, a message is printed, and the function returns **None**
* If the file cannot be parsed (e.g., incorrect format), a message is printed, and the function returns **None**.
* Catches any other errors, prints the error message, and returns **None**

**Filling Missing Values**

**A black background with colorful text

Description automatically generated with medium confidence**

* Fills missing values by propagating the last valid value forward (forward fill).
* Fills remaining missing values by propagating the next valid value backward (backward fill).

**Ensuring 'Close' Column is Numeric**

**A screen shot of a computer code

Description automatically generated**

* Converts the 'Close' column to numeric values. If conversion fails, invalid entries are replaced with “NaN”.
* Drops rows where the 'Close' value is “NaN” after conversion.

**Extracting Close Prices**

**A screen shot of a computer code

Description automatically generated**

* Converts the 'Close' column into a NumPy array for numerical calculations.

**Analysis and Findings**

After the data collection and preprocessing, application calculates Daily Returns, Moving Averages and Rolling Volatility and visualize those data in a graph using Matplotlib.

**Calculating Daily Returns, Moving Averages and Rolling Volatility**

**A screen shot of a computer

Description automatically generated**

* Computes the difference between consecutive elements in the “close\_prices” array and calculates the percentage change (daily returns).
* Inserts “NaN” at the beginning to align the daily returns with the original DataFrame's length.

A black background with white text

Description automatically generated

* In function “moving \_averages”, applies a convolution operation to compute the rolling sum for the specified window size.
* After that it divides by the window size to calculate the average.

A screen shot of a computer code

Description automatically generated

* “sma\_50” computes the 50-day simple moving average.
* **“**sma\_200**”** computes the 200-day simple moving average.
* Creates an array of “NaN” values for the initial 49 rows (because moving averages require a full window).
* Combines “NaN” values with the computed moving average to align with the DataFrame length.

A computer screen shot of text

Description automatically generated

* In “rolling\_std” function, iterates through values in sliding windows of size window.
* **“**np.std” computes the standard deviation for each window.
* Volatility computes the 20-day rolling standard deviation of daily returns.
* **“**np.full(20, np.nan)” adds “NaN” values for the initial rows to align with the DataFrame length.

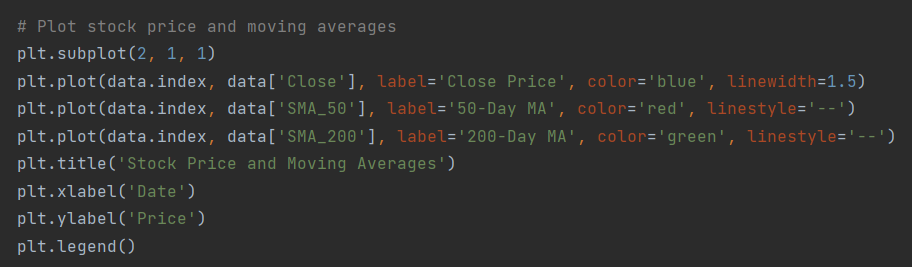
**Creating the Figure**

**A close up of text

Description automatically generated**

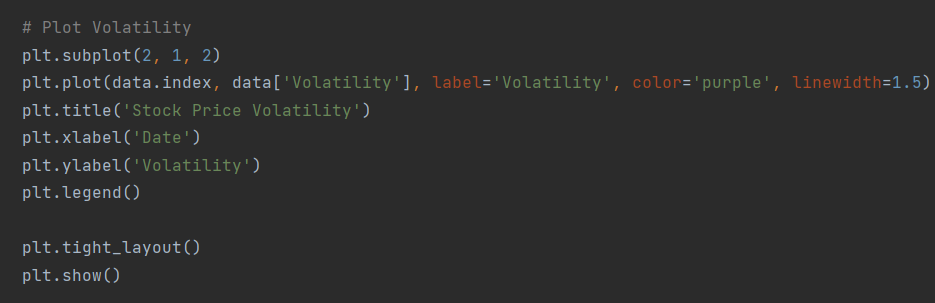
* Creates a new figure for plotting with a specified size of 14 inches wide by 10 inches tall.

**Plotting Stock Price and Moving Averages**

****

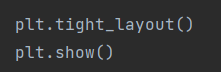
* Sets up a subplot layout with 2 rows and 1 column, activating the first subplot for plotting.
* Represents the x-axis values (dates).
* Plots the stock's closing prices on the y-axis in **blue** with a solid line and line width of 1.5.
* Plots the 50-day simple moving average in **red** with a dashed line
* Plots the 200-day simple moving average in **green** with a dashed line.
* Sets the title of the first plot as "Stock Price and Moving Averages."
* Labels the x-axis as "Date."
* Labels the y-axis as "Price."
* Displays a legend to identify the different lines in the plot.

**Plotting Volatility**

****

* Activates the second subplot in the 2-row layout.
* Represents the x-axis values (dates).
* Plots the stock's volatility on the y-axis in **purple** with a line width of 1.5.
* Sets the title of the second plot as "Stock Price Volatility."
* Labels the x-axis as "Date."
* Labels the y-axis as "Volatility."
* Displays a legend to identify the volatility line.

**Adjusting Layout and Displaying the Plot**

****

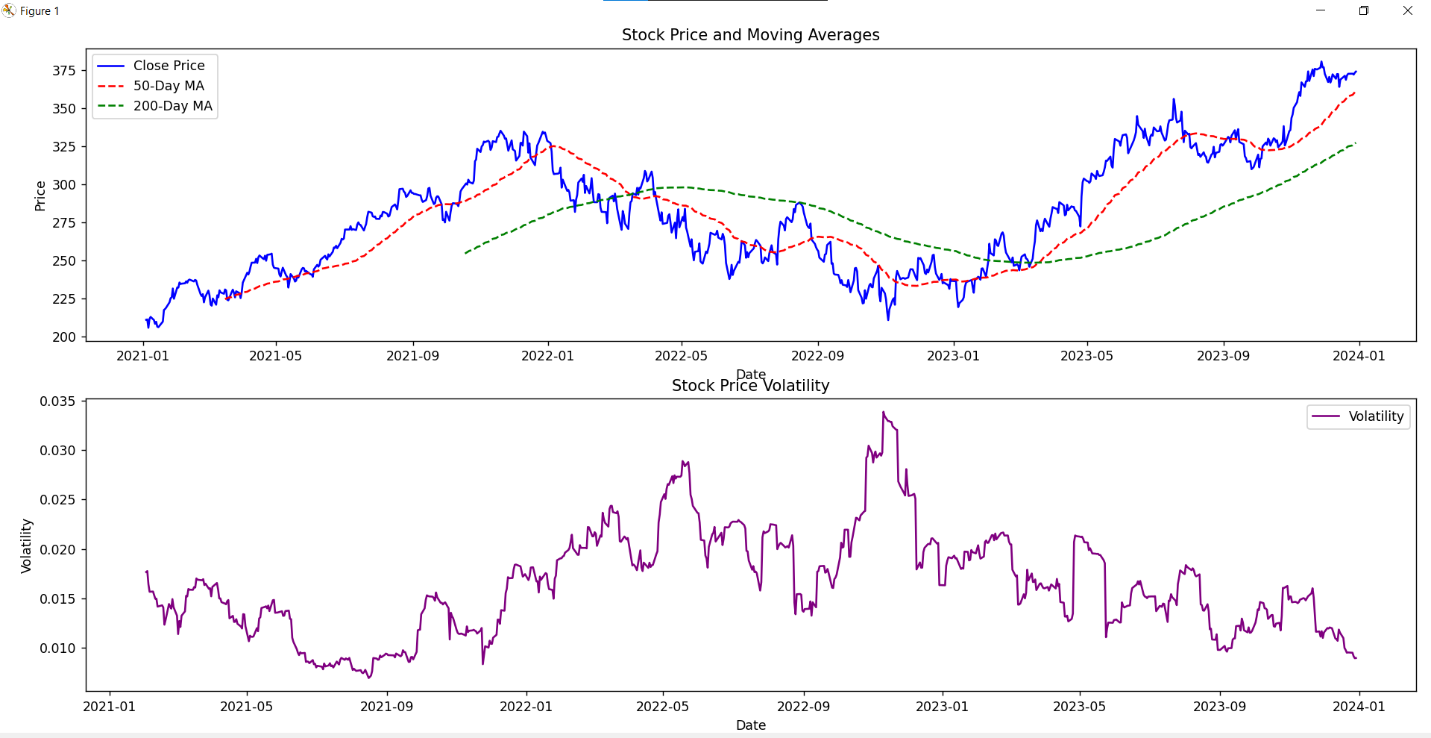
* Automatically adjusts subplot parameters to minimize overlap between elements.
* Displays the final combined figure with the two subplots

**Conclusion**

After all the process is done, the application provides the information below.

* Stock Price and Moving Averages
* Stock Price Volatility

Users can enter the start date, end date and ticker symbol as necessary.

**Sample output**