DATA - 200

San Jose State University

Stock Analysis

Lab 2

Background

The project will provide students with experience creating applications in Python. Students will use object-oriented techniques to develop a stock tracking application. The application will have both console and GUI (Graphical User Interfaces). By processing the historical stock data, profit/loss reports can be generated. An embedded database management system will allow users to save and retrieve stock data. The system will use the Python libraries to create charts and get historical stock data from web sites. Students will learn how to use the GitHub platform for their project.

In this part of the project, you will implement the ability to get stock price history from a web site, or by importing a .csv file.

Objectives – Lab 2

1. Use the beautifulsoup and selenium libraries to scrape historical price data from the Yahoo! Finance web site.
2. Use the csv library to import data from Yahoo! Finance.

Steps

Always test as you go!

So far, our program has use manual data entry for stock prices. This is tedious and time consuming for users. Fortunately, historical stock prices are readily available on the web. For this part of the project, we will implement two ways to get price and volume data into our program.

Web Scraping

The first method uses a technique called **web scraping**. When you visit a web site, HTML code is sent to your browser. Your browser renders this in a user-friendly format and you typically don't see the actual HTML code. However, the code is available and can be used to get raw data into your program. With web scraping, we look for the specific HTML tags that contain the data of interest so that we can load it into our program.

We will be getting our data from Yahoo! Finance. You can see an example of historical data for Microsoft (MSFT) at https://finance.yahoo.com/quote/MSFT/history?p=MSFT. You can use your browser's developer tools to view the HTML code.

To work with the HTML data, we will be using the BeautifulSoup library (**bs4**). This library lets us search for and access data located within specific tags. We will also be using the **selenium** library. Selenium lets you control the browser from your Python code. In many cases, you can scrape web data with just beautifulsoup. In some cases, as with Yahoo! Finance, the site depends heavily on JavaScript for displaying the data. For complex sites, you may need to use selenium to control a real browser which will properly execute and render all the javascript, html, and css code. Selenium controls your browser and can work with most modern browsers. We will be using the Chrome driver, however you can modify the code to work with other browser drivers.

Note that for real-world applications, web scraping should generally be used as a last result. Even minor changes to the HTML code or layout can break your algorithm. When possible use API (application programming interface) calls, or file imports.

CSV File Import

The second technique we will use to get historical price and volume data into our system is importing a **CSV** (Comma Separated Value) file that will be downloaded from Yahoo! Finance. CSV files are simple plain text files. Each line in the file has one record. Each field in the record is separated by commas. This is a common file format which can be exported or imported by most applications. It is a common format for exchanging data between otherwise incompatible systems. To process the .csv files, we will use the **csv** library.

Remember, we are REPLACING the stub code, so you should delete the stub and use your own code. Delete the following stub code in all functions that you will be implementing.

```
messagebox.showinfo("Under Construction","This Module Not Yet Implemented")
```

1. Implement Web Scraping

We will be working in the **stock_GUI.py** file for this part of the project so load this file into your IDE.

The import statements have been included for you. Remember you will need to have installed the bs4 and selenium libraries in project.

1.1 Download Browser Driver and Set Path Variable

The **selenium** library will be used to control your browser which will fetch the historical price data page from Yahoo! Finance for the stocks in your portfolio.

The instructions assume you are using the Chrome browser. If you want to use a different browser, you will have to download the appropriate browser.

1.1.1 Check your browser version.

In Chrome click on the menu (three dots near the top-right) and select Help, then About Google Chrome. Note the browser version.

Visit https://sites.google.com/a/chromium.org/chromedriver/downloads

1.1.2 Download driver and copy to project folder.

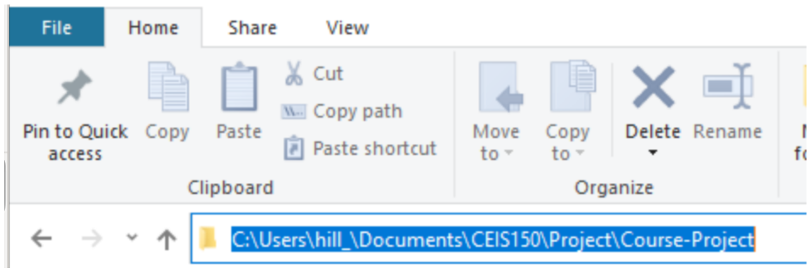
Select the driver that most closely matches your version of Chrome. At the time of this writing, version 81.x was the current version while version 83.x was the testing version. At the time of this writing, the most recent 81.x version of the driver was 81.0.4044.138. The driver may not exactly match your version of Chrome but if you have the most recent version for your branch (ex. 81) it should work.

If you want to use another browser, see the following link for supported browsers and links to the appropriate drivers. https://www.selenium.dev/documentation/en/webdriver/driver_requirements/

- Download the appropriate driver.
- Unzip the driver.
- Copy the unzipped driver file to your project folder.

1.1.3 Set your Path variable.

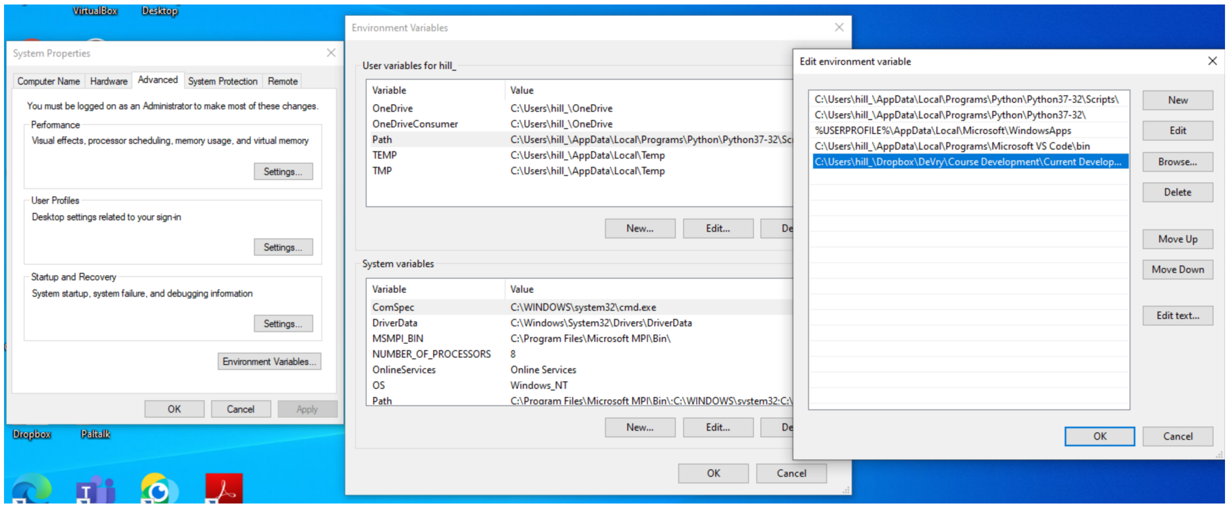
- In Windows explorer, navigate to your project folder and copy the full path.



C:\Users\hill_\Documents\CEIS150\Project\Course-Project

- Click on the Windows Start menu and type “path”. Select **Edit the system environment variables**.

- Click on the **Environment Variables**… button.



You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your sign-in

Startup and Recovery
System startup, system failure, and debugging information

User variables for hill_

Variable	Value
OneDrive	C:\Users\hill_\OneDrive
OneDriveConsumer	C:\Users\hill_\OneDrive
Path	C:\Users\hill_\AppData\Local\Programs\Python\Python37-32\Sc
TEMP	C:\Users\hill_\AppData\Local\Temp
TMP	C:\Users\hill_\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
MSMPI_BIN	C:\Program Files\Microsoft MPI\Bin\
NUMBER_OF_PROCESSORS	8
OnlineServices	Online Services
OS	Windows_NT
Path	C:\Program Files\Microsoft MPI\Bin\;C:\WINDOWS\system32;C:\

```
C:\Users\hill_\AppData\Local\Programs\Python\Python37-32\Scripts\
C:\Users\hill_\AppData\Local\Programs\Python\Python37-32\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\hill_\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\hill_\Dropbox\DeVry\Course Development\Current Develop...
```

- In the User variables area, select the **Path** variable and click the **Edit**… button.
- Click the **New** button to add a path.
- **Paste the path** to your project folder and click **OK**.
- You can then exit out of the Environment Variables and System Properties windows.

Linux users can use the following command in the terminal to temporarily set the PATH variable which will remain set until you log off. Replace {full path to project folder} with the full path to your folder.

```
PATH=$PATH:"{full path to project folder}"
```

For example: `PATH=$PATH:"/home/bob/lab2"`

1.2 Implement retrieve_stock_web() in stock_data.py

The **retrieve_stock_web()** function is located in the **stock_data.py** file. It can be used by either the console or GUI versions of the program.

retrieve_stock_web(dateStart,dateEnd,stock_list)

Code Description

Part 1 – Use Selenium to control the browser to fetch the page.

The function accepts the date range for the history we want to retrieve (startDate, endDate) as well as the list of stocks in our portfolio stock_list.

The dates are converted into integers that match the date format used by Yahoo! Finance. These starting and ending date integers will be used to build the URL for the request.

We set recordCount to 0. We will use this variable to keep track of the number of records returned.

We then use a for loop to process each stock in the portfolio.

For each stock we set stockSymbol to the symbol for the stock.

Then we build a string called url that will contain the url of the page we want to process. The url will include the stockSymbol, dateFrom, and dateTo.

We next set some options for the driver to enable the page to load in a format that is easier to scrape.

In our try block, we create a driver object, passing the options. Then We set a wait time for the page to load.

We load the page using the get method, passing in the url we created. Selenium will load the page in Chrome. If the process fails, we raise a warning so that the user will know to check the Path variable.

Next we pass the HTML source code to BeautifulSoup to create our soup object.

Part 2 – Use BeautfiulSoup to locate and extract the relevant data.

Since the stock price history is located inside a <table> tag, we look for that item.

Next we create a collection of rows called dataRows for each row int the table (<tr> tag).

For each row in the table we look for the data (<td> tag) and add the data to the list rowList.

A normal record contains 7 columns so we check the length of rowList. The data may also contain rows for dividends, stock splits, or other events. For our purposes we will ignore these special cases, however, an actual stock tracking program should include these.

We create a new DailyData object passing in the date (rowList[0]), the closing price (rowList[5]) and volume (rowList[6]). The table contains other information such as the high and low price for the day which we will ignore for our purposes.

Finally we add the daily_data object to our stock and increment the record count.

Note, code may be shown with line breaks due to space limitations but each statement should be entered on its own line.

```
dateFrom = str(int(time.mktime(time.strptime(dateStart,"%m/%d/%y"))))
dateTo = str(int(time.mktime(time.strptime(dateEnd,"%m/%d/%y"))))
recordCount = 0
```

```
for stock in stock_list:
    stockSymbol = stock.symbol
    url = 
"https://finance.yahoo.com/quote/"+stockSymbol+"/history?period1="+dateFrom+"&perio
d2="+dateTo+"&interval=1d&filter=history&frequency=1d"
    # Note this code assumes the use of the Chrome browser.
    # You will have to modify if you are using a different browser.
    options = webdriver.ChromeOptions()
    options.add_experimental_option('excludeSwitches',['enable-logging'])

options.add_experimental_option("prefs",{'profile.managed_default_content_settings.
javascript': 2})
    try:
        driver = webdriver.Chrome(options=options)
        driver.implicitly_wait(60)
        driver.get(url)
    except:
        raise RuntimeWarning("Chrome Driver Not Found")

    soup = BeautifulSoup(driver.page_source,"html.parser")
```

1.3 Implement scrape_web_data() in stock_GUI.py

Now that we have the code to gather data from the web and insert it into our portfolio, we can add it to the GUI. The user will use the **Web** menu and select **Scrape Data from Yahoo! Finance**. This calls the scrape_web_data() method.

```
self.webmenu.add_command(label = "Scrape Data from Yahoo! Finance...",
command=self.scrape_web_data)
```

scrape_web_data(self)

Code Description

Since the retrieve_stock_web() function that we just implemented requires the start and end date, we first need to gather than information from the user. We will use the **simpledialog** class and call the **askstring** method which will return a string entered by the user (in this case the date).

Next we call the retrieve_stock_web() method that we just created, passing the dates and our stock portfolio list. If the process fails, we display an error message.

We then call display_stock_data() to update our history and report tabs.

Finally, we display a message to the user that the data has been retrieved.

```
dateFrom = simpledialog.askstring("Starting Date","Enter Starting Date (m/d/yy)")
dateTo = simpledialog.askstring("Ending Date","Enter Ending Date (m/d/yy")
try:
    stock_data.retrieve_stock_web(dateFrom,dateTo,self.stock_list)
except:
    messagebox.showerror("Cannot Get Data from Web","Check Path for Chrome Driver")
    return
self.display_stock_data()
```

```
messagebox.showinfo("Get Data From Web","Data Retrieved")
```

1.4 Test the Code

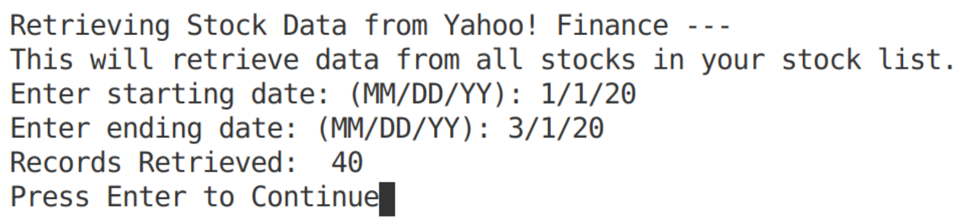
Now that you have implemented the back end functionality as well as the user interface, you can test the code.

- Run the **stock_GUI.py** file.
- Make sure you have data loaded either from the database or through manual data entry of stocks. Make sure that all the stocks in your portfolio have a valid ticker symbol.
- Select **Scrape Data from Yahoo! Finance** from the **Web** menu.
- Enter the starting and ending dates in m/d/yy format. Note that you can only retrieve about 60 days worth of stock history. If you want more history, you can repeat the process to load additional dates.
- You should see your browser pop up and open the Yahoo! Finance history page for each stock in your portfolio. This may take a few moments depending on the speed of your connection and computer.
- After all the data has loaded, check your History and Report tabs. You should see the stock price and volume history there.
- Create a chart. You should see the history on your chart.

1.5 Implement retrieve_from_web()

Now that you can get stock information from the web using the GUI, it should be easy to implement the feature in the console version of the program. The **retrieve_from_web()** function is in the **stock_console.py** file.

Below is an example of how the user interface might look. You can implement the code yourself.



```
Retrieving Stock Data from Yahoo! Finance ---
This will retrieve data from all stocks in your stock list.
Enter starting date: (MM/DD/YY): 1/1/20
Enter ending date: (MM/DD/YY): 3/1/20
Records Retrieved:  40
Press Enter to Continue
```

1.6 Test the Code

Test your console interface by running stock_console.py.

- Run the **stock_console.py** file.

- Make sure you have data loaded either from the database or through manual data entry of stocks. Make sure that all the stocks in your portfolio have a valid ticker symbol.
- Select **Manage Data** (option 5), then **Retrieve Data from Web** (option 3).
- Enter the starting and ending dates in m/d/yy format. Note that you can only retrieve about 60 days worth of stock history. If you want more history, you can repeat the process to load additional dates.
- You should see your browser pop up and open the Yahoo! Finance history page for each stock in your portfolio. This may take a few moments depending on the speed of your connection and computer.
- After all the data has loaded, go back to the menu, and run **Show Report** (option 3). You should see the stock price and volume history there.
- Create a chart. You should see the history on your chart.

2. Implement CSV File Import

Python can work with many different file types. The CSV (Comma Separated Value) format is a simple text file format that is commonly used to exchange data between different systems. In .csv files, each row represents one record. Each record is divided into fields separated by commas. We will use the **csv** library to implement the import functionality.

2.1 Implement import_stock_web_csv()

The **import_stock_web_csv()** method is found in the **stock_data.py** file. It receives our stock portfolio (stock_list), the ticker symbol (symbol), and filename of the .csv file (filename) as arguments.

import_stock_web_csv(stock_list,symbol,filename)

Code Description

First we find the correct stock in stock_list by looping through the list and comparing the symbol with the symbol passed into the function.

Once we have found the stock, we use the **with** statement to open the file. The with structure will automatically close the file when we are finished reading in all the records.
Next we create a csv.reader object specifying "," as the delimiter.

We do an initial read using next(datareader). This skips past the column header information in the file so we can start reading in the actual historical data.

For each record in the file, we create a new DailyData object using the date (row[0]), closing price (row[4]), and volume (row[6]). Remember the index values represent the columns in each row.

We then add the row of daily history to our stock.

2.2 Implement importCSV_web_data()

Now that we’ve implemented the csv reader, we need to update **importCSV_web_data()** in **stock_GUI.py**.

Code Description

First we need to find out which stock is selected in the list box. We use the get method of stockList and save the results in symbol.

Next we need to know what file to import. We use the filedialog class askopenfilename method. We save the full path in filename.

We test to make sure the filename is not blank.

Next we call import_stock_web_csv() passing the stock list, symbol, and filename.

We then call display_stock_data() to update the History and Report tabs.

Finally, we display a message box showing that the file was imported.

```
symbol = self.stockList.get(self.stockList.curselection())
filename = filedialog.askopenfilename(title="Select " + symbol + " File to
Import",filetypes=[('Yahoo Finance! CSV','*.csv')])
if filename != "":
    stock_data.import_stock_web_csv(self.stock_list,symbol,filename)
    self.display_stock_data()
    messagebox.showinfo("Import Complete",symbol + "Import Complete")
```

2.3 Test Code

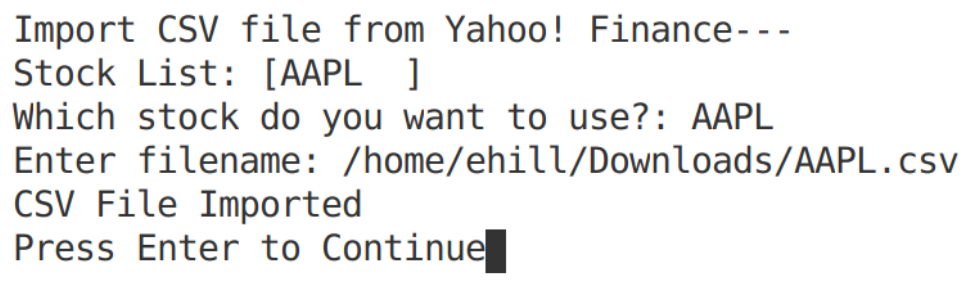
Test your GUI CSV Import code.

- To test your code, enter a new stock into your program. It should have a valid ticker symbol, and have no preexisting data.
- Go to https://finance.yahoo.com/ and enter the stock symbol.
- Click on the **Historical Data** tab.
- Click the **Download** link. (Be sure you remember where you save the file!)
- In your program, **click on the stock** you will be importing data for in the stock list.
- Click on the **Web** menu and select **Import CSV From Yahoo! Finance**.
- Browse to the folder where you saved the .csv file, and **select the file**.
- Click **Open** to open the file and import the data.
- Verify the data was imported by reviewing the History and Report tabs.

2.4 Implement import_csv()

Now that you can get stock information from the web using a CSV file import, it should be easy to implement the feature in the console version of the program. The **import_csv() function** is in the **stock_console.py** file.

Below is an example of how the user interface might look. You can implement the code yourself.



```
Import CSV file from Yahoo! Finance---
Stock List: [AAPL  ]
Which stock do you want to use?: AAPL
Enter filename: /home/ehill/Downloads/AAPL.csv
CSV File Imported
Press Enter to Continue
```

2.5 Test Code

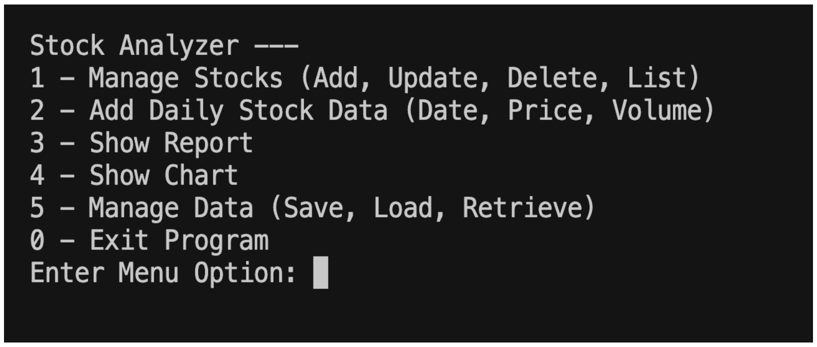
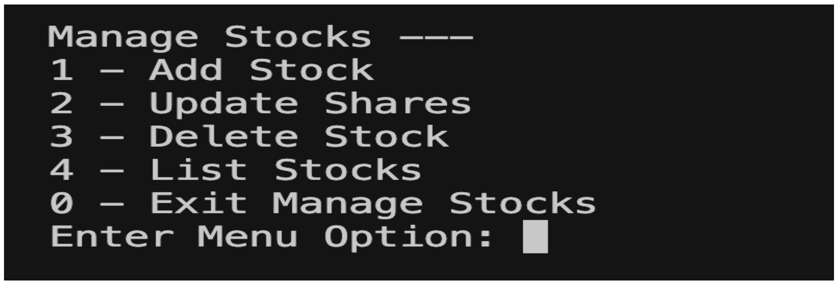
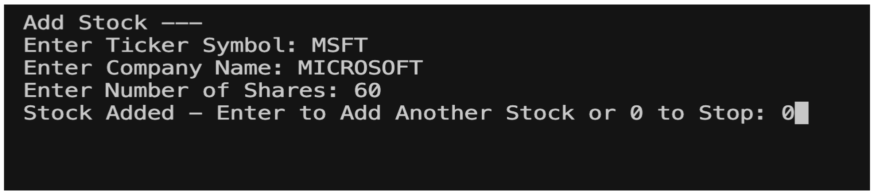
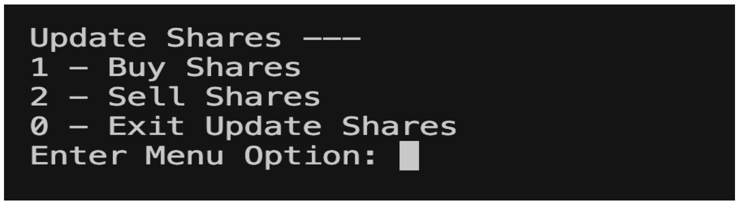
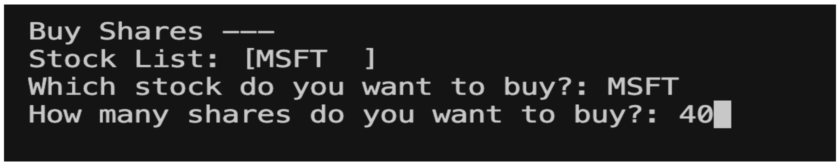
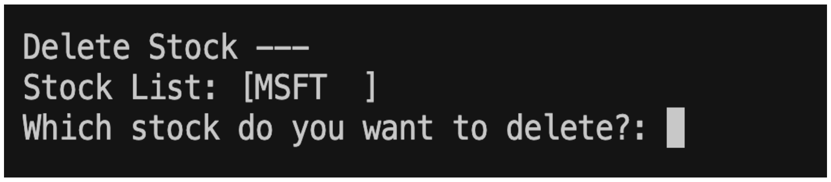
Test your console CSV Import code.

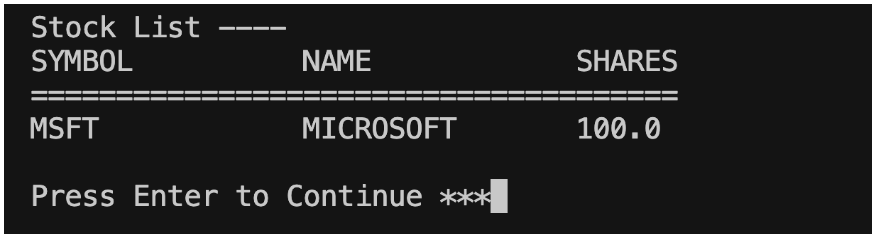
- Run s**tock_console.py.**
- To test your code, enter a new stock into your program. It should have a valid ticker symbol, and have no preexisting data.
- Go to https://finance.yahoo.com/ and enter the stock symbol.
- Click on the **Historical Data** tab.
- Click the **Download** link. (Be sure you remember where you save the file!)
- In your program go to **Manage Data** (option 5).
- Select **Import from CSV File** (option 4).
- Enter the **symbol** and **filename** with full path.
- Go back to the main menu and verify the data was imported by selecting **Show Report** (option 3).

2.6 Sample Output

Console Output

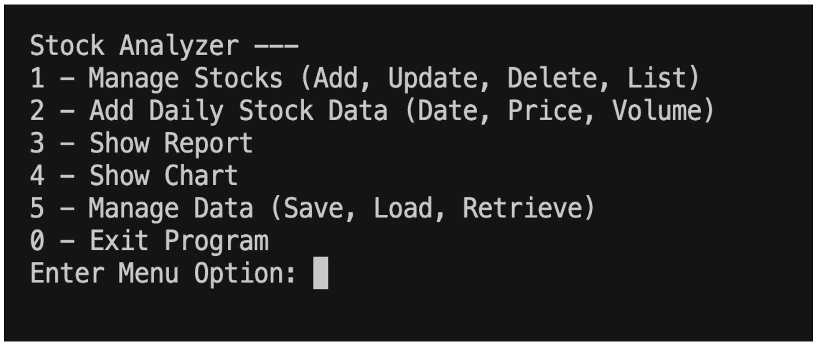
Add/manage Stocks

Sno	Description	Output
1	1st screen	 ``Stock Analyzer --- / 1 - Manage Stocks (Add, Update, Delete, List) / 2 - Add Daily Stock Data (Date, Price, Volume) / 3 - Show Report / 4 - Show Chart / 5 - Manage Data (Save, Load, Retrieve) / 0 - Exit Program / Enter Menu Option:``
2	Choose 1 – Manage Stocks	 ``Manage Stocks --- / 1 - Add Stock / 2 - Update Shares / 3 - Delete Stock / 4 - List Stocks / 0 - Exit Manage Stocks / Enter Menu Option:``
3	Add Stock	 ``Add Stock --- / Enter Ticker Symbol: MSFT / Enter Company Name: MICROSOFT / Enter Number of Shares: 60 / Stock Added - Enter to Add Another Stock or 0 to Stop: 0``
4	Choose 2 Update Shares In this you buy and sell the shares	 ``Update Shares --- / 1 - Buy Shares / 2 - Sell Shares / 0 - Exit Update Shares / Enter Menu Option:``  ``Buy Shares --- / Stock List: [MSFT ] / Which stock do you want to buy?: MSFT / How many shares do you want to buy?: 40``
5	Delete the stock	 ``Delete Stock --- / Stock List: [MSFT ] / Which stock do you want to delete?:``

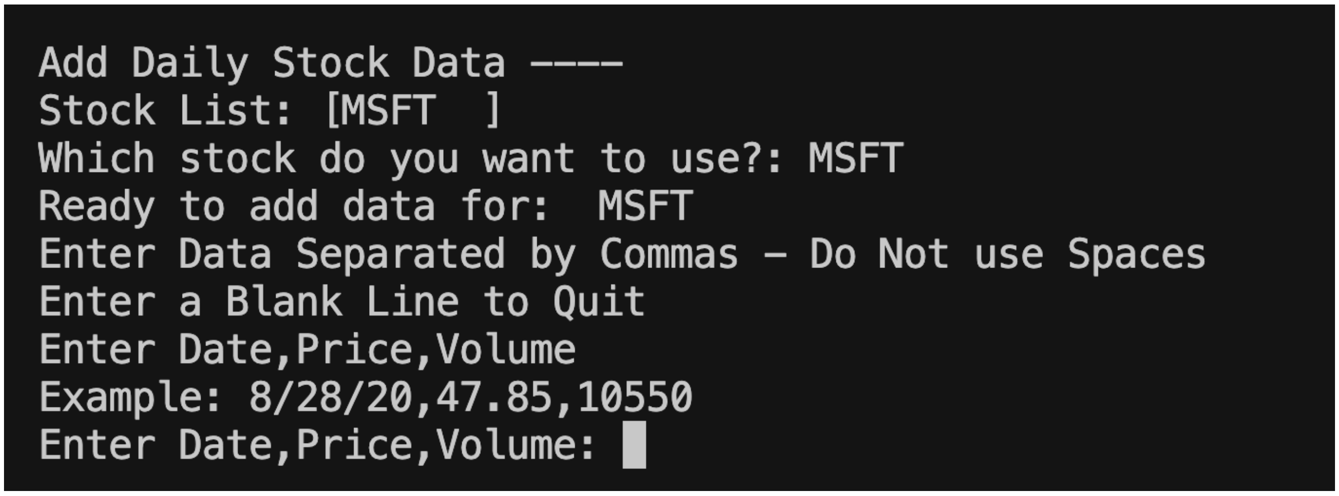
6	**List the stocks**	 `Stock List ----` `SYMBOL NAME SHARES` `MSFT MICROSOFT 100.0` `Press Enter to Continue ***`

```
Stock List ----
SYMBOL              NAME               SHARES
=========================================
MSFT                MICROSOFT          100.0

Press Enter to Continue ***
```

Add Daily Stock Data (Date, Price, Volume)

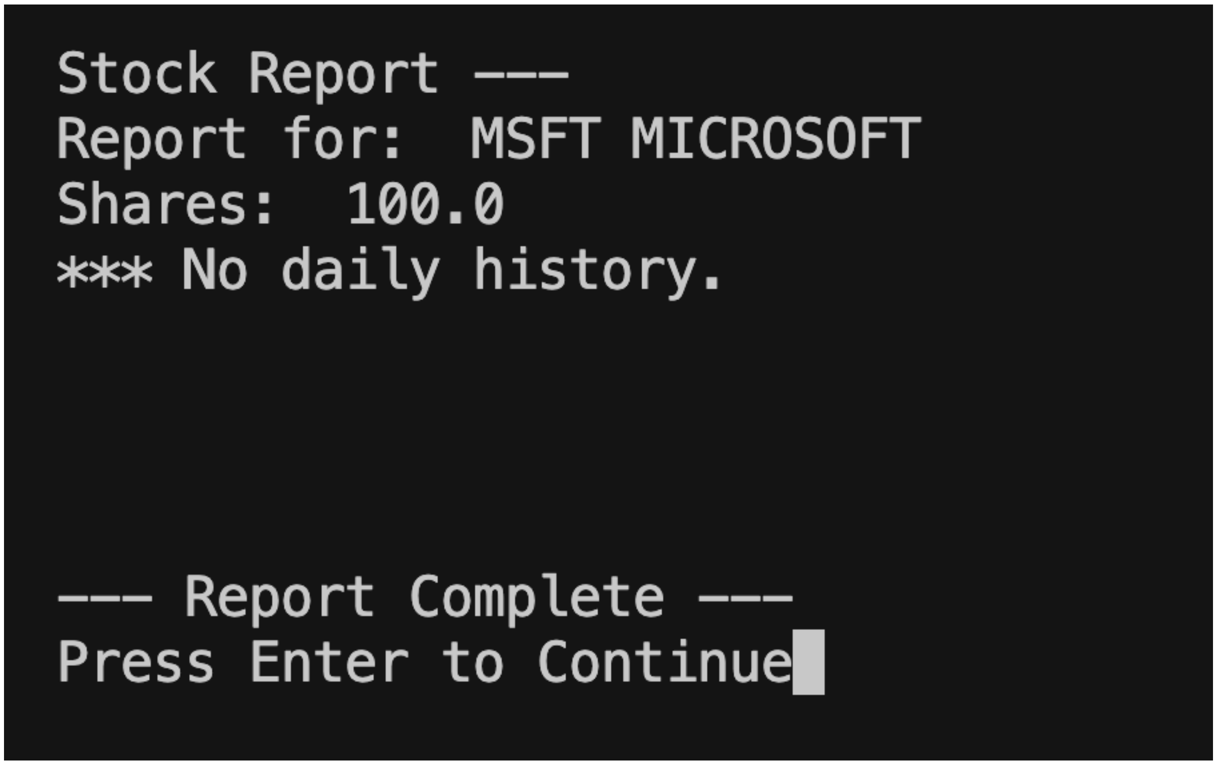


```
Stock Analyzer ---
1 - Manage Stocks (Add, Update, Delete, List)
2 - Add Daily Stock Data (Date, Price, Volume)
3 - Show Report
4 - Show Chart
5 - Manage Data (Save, Load, Retrieve)
0 - Exit Program
Enter Menu Option: 
```



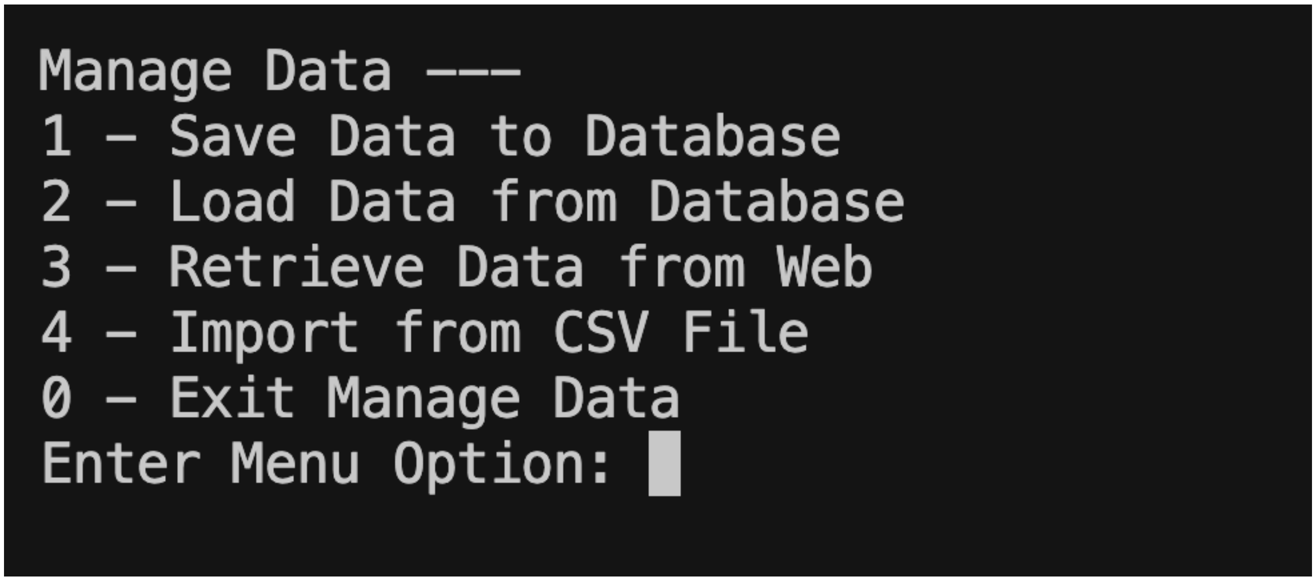
```
Add Daily Stock Data ----
Stock List: [MSFT  ]
Which stock do you want to use?: MSFT
Ready to add data for:  MSFT
Enter Data Separated by Commas - Do Not use Spaces
Enter a Blank Line to Quit
Enter Date,Price,Volume
Example: 8/28/20,47.85,10550
Enter Date,Price,Volume: 
```

Show Report



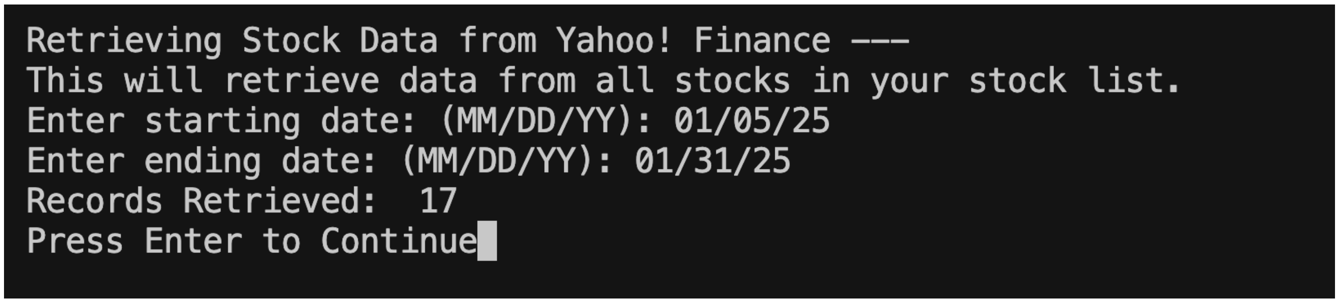
```
Stock Report ---
Report for:  MSFT MICROSOFT
Shares:  100.0
*** No daily history.




--- Report Complete ---
Press Enter to Continue
```

5 - Manage Data (Save, Load, Retrieve)



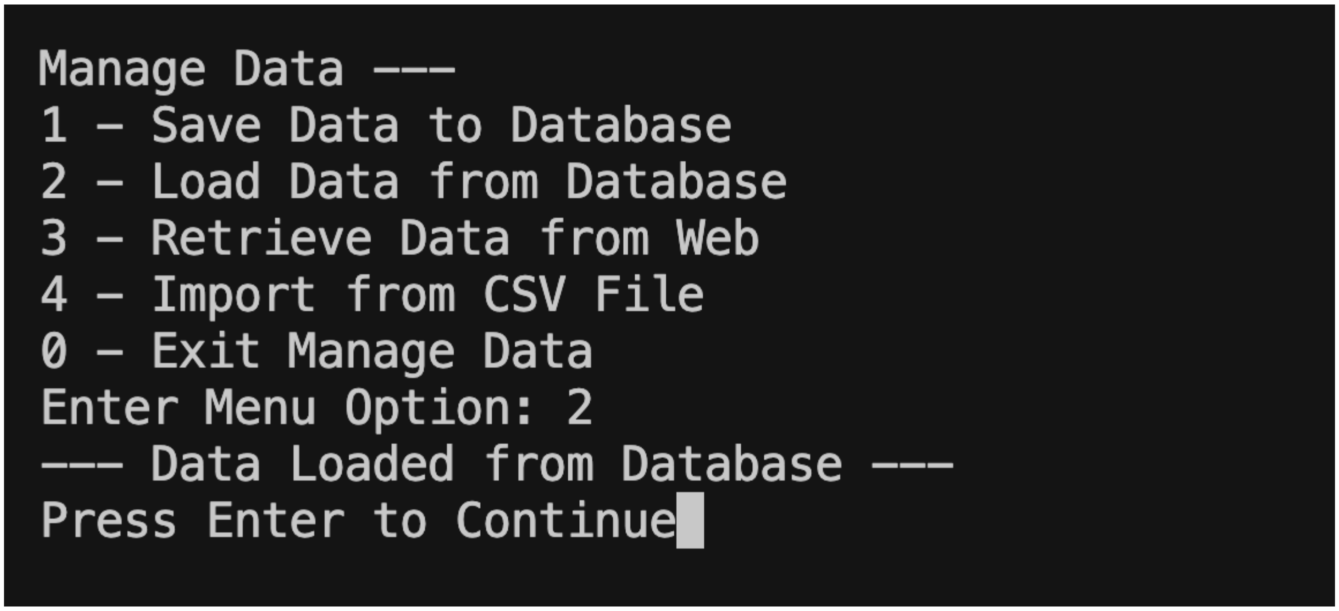
```
Manage Data ---
1 - Save Data to Database
2 - Load Data from Database
3 - Retrieve Data from Web
4 - Import from CSV File
0 - Exit Manage Data
Enter Menu Option: 
```

Choose 3 -? Retrieve Data From Web



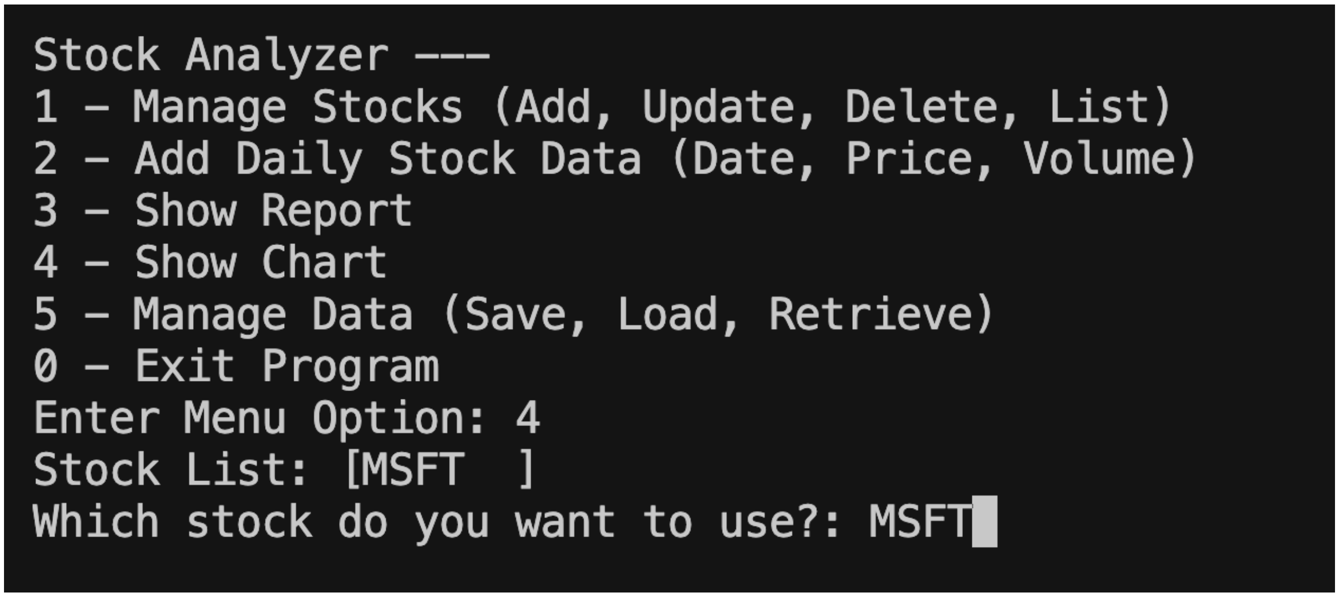
```
Retrieving Stock Data from Yahoo! Finance ---
This will retrieve data from all stocks in your stock list.
Enter starting date: (MM/DD/YY): 01/05/25
Enter ending date: (MM/DD/YY): 01/31/25
Records Retrieved:  17
Press Enter to Continue
```

Save to Database and then load



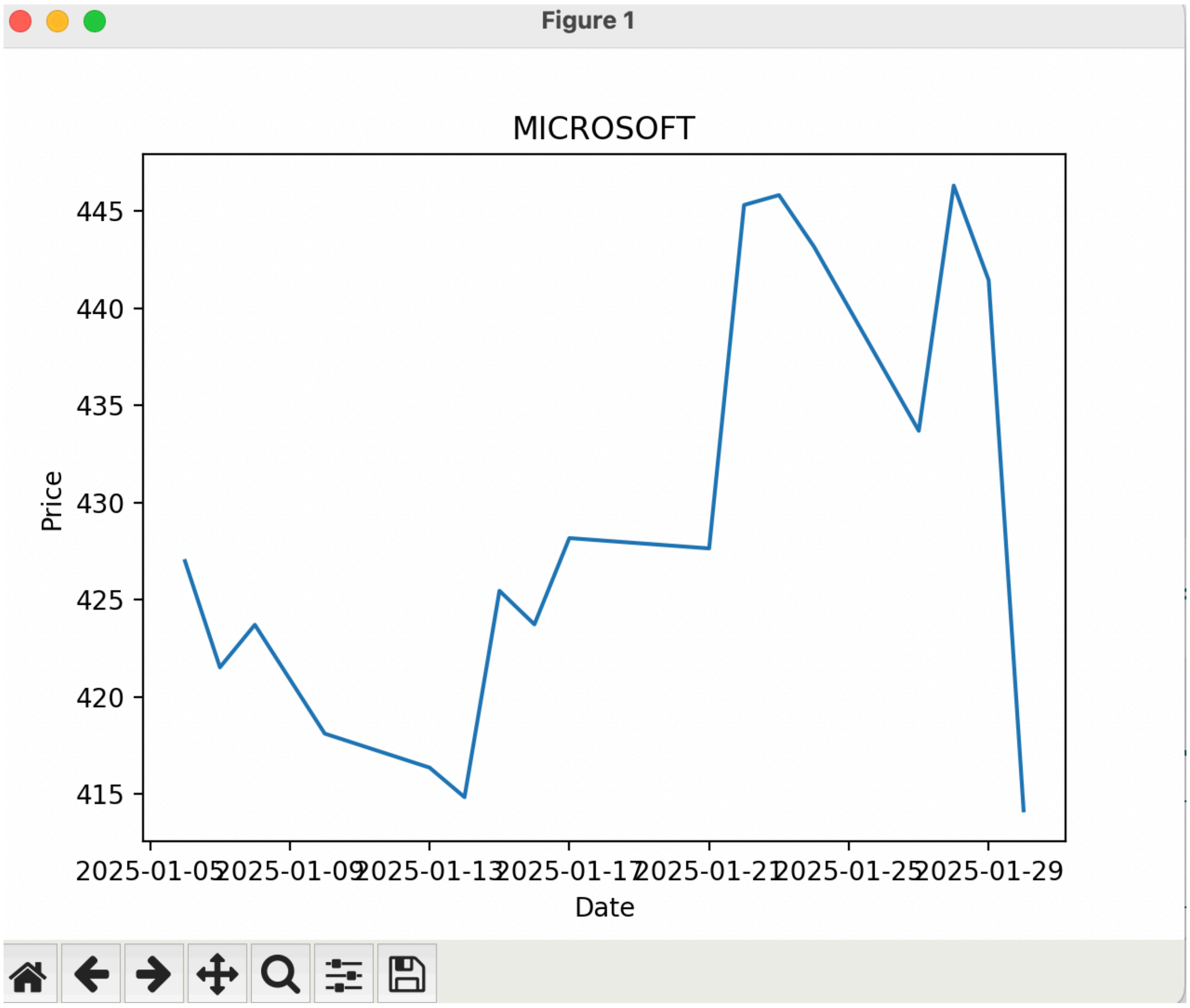
```
Manage Data ---
1 - Save Data to Database
2 - Load Data from Database
3 - Retrieve Data from Web
4 - Import from CSV File
0 - Exit Manage Data
Enter Menu Option: 2
--- Data Loaded from Database ---
Press Enter to Continue
```

Show Chart



```
Stock Analyzer ---
1 - Manage Stocks (Add, Update, Delete, List)
2 - Add Daily Stock Data (Date, Price, Volume)
3 - Show Report
4 - Show Chart
5 - Manage Data (Save, Load, Retrieve)
0 - Exit Program
Enter Menu Option: 4
Stock List: [MSFT  ]
Which stock do you want to use?: MSFT
```

It should graph:



Similarly you can show graph based on your daily transactions or the load the data from CSV.