PERIODICALLY WEB SCRAPING PR NEWSWIRE AND YAHOO FINANCE

AIDI 1100 – Final Project

Michael Molnar
100806823

Abstract:

The aim of this project is to write a Python module that will automatically scan and parse news from the PR Newswire website every minute. For each piece of new news, the program will retrieve its title, publication date and time, and any stock symbols that it contains. For each of these stock symbols it will retrieve historical stock information from Yahoo Finance. Finally, it will plot two visualizations for each of the stock symbols found in the article – one showing the opening and closing for the previous five days, and the second showing the volume over the same time period.

The first time my module is run, it will retrieve the latest article from the news list and store its href into a global variable, keeping track of the latest processed article. For this first article, the title, publication date, and any contained stock symbols will be printed as output. If there are symbols, the stock information will be retrieved, and the plots will be generated and displayed.

Once a minute has passed the module will scan the news again. The hrefs will be extracted and compared against the current latest processed article. If there are no new articles, an output is printed advising of this. If there are new articles, for each I create a dictionary containing the details – title, href, date, and symbols mentioned. I print the summary output and display plots for any symbols that appear.

Article data for all news articles will be appended to a csv file as they are found. Stock information, for stock symbols that are mentioned, will be appended to a separate csv file. All plots generated will be displayed on the screen for 10 seconds and then stored into a folder. The plots are named according to their symbol and the current date for ease of reference.

Project Architecture:

I make use of the following libraries:

```python
"""
Periodically Web Scraping PR Newswire and Yahoo Finance
AIDI 1100 - Final Project

Michael Molnar
100806823
"""

# Import needed libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
from bs4 import BeautifulSoup
import yfinance as yf
import os
from pynput import keyboard
import time
from datetime import date

# Create to global variables
# baseurl is to simplify web requests
global baseurl
baseurl = 'https://www.prnewswire.com/'

# latest_href will keep track of the most recent processed news article
global latest_href
latest_href = ''

```

Figure 1 – Libraries and Global Variables

Besides the usual libraries I use requests and Beautiful Soup for parsing, yfinance for retrieving stock prices, os to make a folder to store plots, keyboard to allow the user to stop continuously scanning, time to allow the scan to run every minute, and date in naming the plots. I set two global variables here. The addresses to all the news articles are the PR Newswire URL followed by a unique HREF. I set a global variable, "latest_href", so that I can keep track of the latest processed article. This will be updated as new articles are found and processed. On the news articles list, the articles are within a class called "news-releases". Both the href and the article's title are available here.

Figure 2 – Location of Article Titles and HREFs

Having the href, I make another request to the actual article. Within an article I extract the publication date from the class "mb-no".

Figure 3 – Location of Article Publication Date

The stock symbols, if the article contains any, are located with a class called "ticket-symbol".

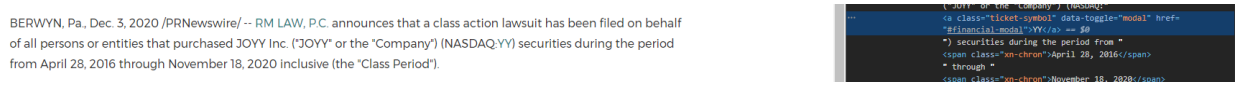


Figure 4 – Location of Stock Tickers

Opening various articles, I discovered a few things about this class. For one thing, the majority of the articles do not contain any stock symbols. In these cases this class does not exist. Secondly, some articles contain multiple stock symbol references, either to the same symbol or to multiple. To handle this I have created a loop to retrieve all instances of the "ticket-symbol" class. I then make this list a set to remove duplicates.

For each article I make a dictionary containing its title, href, date of publication, and the stock symbols it contains. I append these into a csv file so that as new articles are retrieved their details are added chronologically into the file.

For the stock symbols I use the Python module yfinance (https://pypi.org/project/yfinance/) to retrieve historical stock information by stock symbol. I append the symbol as a column to the generated Data Frame for clarity once it is written to a csv file. I retrieve the last five days' opens and closes and volume data, and I create two time series plots with these. The title of the article is printed at the top of the plot.

Retrieved stock information is appended to a separate csv file for future reference. The plots are displayed on screen and then saved into a folder.

To achieve a module that continuously runs without user interaction I make use of a keyboard listener and a while loop, and I use the time module to rescan after a certain amount of time – set for 60 seconds – has elapsed. I update the variable "latest_href" as a reference point so that upon each scan, only articles that are newer than this are processed. With this method I am not duplicating the parsing and stock retrieval for the same article more than once.

Project Solution and Code:

The first time the program is run I want it to retrieve the latest article. Using requests and Beautiful Soup I extract the first article from the news-release class. I use the href of this article as my stopping point for already processed news the next time the scanning takes place. A second request into the article is to retrieve the date and stock symbol information. A dictionary of information is created and printed to the screen. The function returns two things – the set of contained stock symbols and the article's title. These will be used for plotting the stock information. I append this article's information to a csv file.

```python
def initial_run():
    # On first run, create "Plots" folder if it does not exist
    # This program can be restarted at any time, so check if it already exists
    if not os.path.exists('Plots'):
        os.makedirs('Plots')

    # Request the content of PR Newswire News Release List
    page = requests.get('https://www.prnewswire.com/news-releases/news-releases-list/', timeout=1000)
    soup = BeautifulSoup(page.content, 'html.parser')

    # Find the most current news artic;e
    article = soup.find('a', attrs={'class':'news-release'})
    # Extract its hrf
    url = article.attrs['href']
    # Extract its title
    title = article.text

    # Request and parse the article
    page2 = requests.get(baseurl + url)
    soup2 = BeautifulSoup(page2.text, 'html.parser')

    # Extract the publcation date
    results = soup2.find_all('p', attrs={'class':'mb-no'})
    day = results[0].text

    # Extract all symbols from the article
    results2 = soup2.find_all('a', attrs={'class':'ticket-symbol'})
    # Create a list for if there is more than one
    symbols = []
    for symbol in results2:
        symbols.append(symbol.text)
    # Remove any publicates
    symbols = set(symbols)

    # Set symbols to None if there were none
    if not symbols:
        symbols = None
```

```python
    # Create a dictionary for the newest article
    newest_dict = {'Title' : title,
                   'URL' : url,
                   'Date' : day,
                   'Symbols' : symbols
                  }
    # Turn into a DataFrame and append to the csv file
    # The tile will be created here if it does not already exist
    article_data = pd.DataFrame(newest_dict, index=[0])
    article_data.to_csv('article_data.csv', mode='a', index=False, header=False)

    # Update the global variable latest_href as a reference point
    global latest_href
    latest_href = url

    # Print the summary to the screen
    print("Title: ", title)
    print('Date: ', day)
    print('Stock Symbols Mentioned: ', symbols)
    print('\n')

    return symbols, title
```

Figure 5 – Initial_run Function

<u>Retrieving Stock Information by Symbols</u>

As mentioned I make use of the yfinance module for this part of the task. This function takes a stock symbol as input and retrieves the historical data for it from Yahoo Finance.

```python
"""
This function takes as input a stock symbol.  It uses yfinance to retrieve the historic
stock information for it.  What returns is a data frame, and a column is added to
include the stock symbol.  The data is appended to a second csv file and then dataframe
is returned.
"""
def get_tickers(sym):

    # Use yfinance to retrieve historical stock information by symbol
    stock = yf.Ticker(sym)
    # Get the history for the past five days
    stock_hist = stock.history(period='5d')
    # A data frame is created, so create a new column containing the stock symbol
    stock_hist['Symbol'] = sym

    # Append to csv file
    stock_hist.to_csv('stock_data.csv', mode='a', header=False)

    # Return the dataframe for plotting
    return stock_hist
```

Figure 6 – Get_tickers Function

After this function is called a dataframe is returned. This is a sample of the dataframe generated for the stock symbol “MSFT”.

```
stock_hist
```

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits	Symbol
2020-11-30	64.570903	64.570903	62.862778	62.862778	5714800	0.00	0	MSFT
2020-12-01	63.820071	64.392575	63.698065	64.270569	7896100	0.00	0	MSFT
2020-12-02	64.129792	64.927541	64.092250	64.889999	1980800	0.00	0	MSFT
2020-12-03	64.949997	65.349998	64.750000	65.000000	6836200	4.25	0	MSFT
2020-12-04	65.050003	65.690002	65.040001	65.589996	1949000	0.00	0	MSFT

Figure 7 – Sample Output of get_tickers

Next comes the plotting. I am generating two subplots for each stock symbol I find – one showing the opens and closes on the same axis, and the other showing the volume, both over the previous five days of trading. I am printing the article's title at the top of the plot. This function takes in the previously created stocks data frame, the article's title, and a stock symbol. It will be called for each symbol found within each article.

```
"""
This functon takes three inputs - the data frame created by the get_tickers function,
and the symbol and article title from the dictionary created by get_details.  It creates
two subplots - one displaying opens and closes for the last five days, the other
showing volume for the same timeframe.  The article title is printed above the plot
and the symbol is used in the plot titles.  Plots are displayed on screen for 10
seconds and then saved to the Plots folder - named according to their symbol and
today's date.
"""
def make_plots(stocks_df, sym, headline):

    fig = plt.figure(figsize=(15,5))
    # The first subplot consists of the opens as closes for the stock over the last five days
    # These are plotted as lines on the same axis
    ax1 = fig.add_subplot(1,2,1)
    ax1.plot(stocks_df['Open'], color='blue', marker='o', label='Opens')
    ax1.plot(stocks_df['Close'], color='red', marker='o', label='Closes')
    ax1.set_title('"{}" Opens and Closes (last 5 days)'.format(sym))
    ax1.set_ylabel ('Price ($)', fontsize=12)
    ax1.legend(loc='best')
    plt.xticks(stocks_df.index, rotation=20)

    # The second subplot is the volume over the last five days
    ax2 = fig.add_subplot(1,2, 2)
    ax2 = plt.plot(stocks_df['Volume'], color='black', marker='o')
    ax2 = plt.ticklabel_format(style='plain', axis='y')
    ax2 = plt.title('"{}" Volumes (last 5 days)'.format(sym))
    plt.xticks(stocks_df.index, rotation=20)

    # Add the title of the article to the top of the plot
    plt.text(0.5, 0.98, 'Headline: {}'.format(headline),
    ha='center', va='top', transform=fig.transFigure, fontsize=10)

    # Instead of having the user close the plot for the code to continue,
    # I show the plot for 10 seconds and then close it automatically.
    plt.show(block=False)
    plt.pause(10)
    plt.close()

    # Get today's date for file naming - format with underscores
    today = date.today()
    today = today.strftime('%m_%d_%Y')

    # Save the plot with the stock symbol and today's date for future reference
    fig.savefig('Plots\{}_{}.png'.format(sym, today))

```

Figure 8 – Make_plots Function

Because this module is to run continuously on its own, until stopped by the user, without any need for user interaction, I do not want to wait for the user to have to close the plots for the code to restart running. To resolve this I have the plot show on screen for ten seconds, after which it is closed and saved to disk, and the program continues without any need for user input.

<u>On Subsequent Scans</u>

After the one minute has passed the module will scan again. The first thing it will do is print an update that it is scanning. It calls the function, "get_new_news", to extract all of the article hrefs, newest to oldest, on the articles list page of the website. It returns this list for use in the next function.

```python
"""
This function will be called on each scan.  It requests and parses the news article
list and extracts all of the hrefs.  It returns a list of these.
"""
def get_new_news():

    # After the initial run, make a new request to the article list
    to_add_page = requests.get('https://www.prnewswire.com/news-releases/news-releases-list/')
    to_add_soup = BeautifulSoup(to_add_page.content, 'html.parser')
    # Find all articles on the main page
    to_add_articles = to_add_soup.find_all('a',attrs={'class':'news-release'})
    # Get the hrefs for each and append to a list
    to_add_hrefs = []
    for article in to_add_articles:
        to_add_hrefs.append(article.attrs['href'])

    # Return this list for use in check_if_new
    return to_add_hrefs

```

Figure 9 – Get_new_news Function

Next, I call “check_if_new” with the previous list as input. Here, it checks each of these hrefs against the last processed article, “latest_href”. Since it is checking from newest to oldest, as soon as it reaches an href that matches this, it stops, because what follows will have already been processed. A message is printed to the screen at this point indicating whether there are new news articles or not, and the list of unprocessed hrefs is returned for processing.

```python
"""
This function will take as input the return of get_new_news.  It creates a new list -
to_process - by checking each of the hrefs against the last processed article.
It is processing from newest to oldest, so as soon as it reaches that, it discards the rest.
It returns the list of hrefs that have not yet been processed.
"""
def check_if_new(article_list):

    # Check to see if the articles in the list have already been processed
    to_process = []

    if article_list:
        for url in article_list:
        # If it has not, add it to list and move on
            if url != latest_href:
                to_process.append(url)
            else:
            # As soon as you reach an already parsed article, stop
                break
    if to_process:
        print('There is new news!\n')

    else:
        print('There is no new news\n')
    # Return the list of hrefs to be processed
    return to_process

```

Figure 10 – Check_if_new Function

I use this list in my next function – “get_details”. The goal of this function is to generate a dictionary for each of these hrefs. I want for each the article’s title, href, date of publication, and the stock symbols it contains. This information will be for storage in a csv file for later reference.

For each of the hrefs in this list I make a request to the page and parse it using Beautiful Soup. I extract the necessary information using the classes that I have previously identified. Again, I make sure to extract any and all stock symbols and remove any duplication. I combine the dictionaries of each new href into a list, which I call “new_data”.

```python
"""
This function takes as input the list of hrefs that have not been processed yet.
For each of these it requests and parses the article.  It extracts all of the desired
information - title, publication date, and stock symbols.  Since articles may
contain zero, one, or more stock symbols - with duplication in some cases - this
ensures that only the unique values are kept.  A dictionary is created for each href,
all of which are returned as a list.
"""
def get_details(to_process_list):
    new_data = []
    # For each article to be processed, create a dictionary
    for entry in to_process_list:
        dict_new = {}
        # Request and parse the article
        page3 = requests.get(baseurl + entry, timeout=1000)
        soup3 = BeautifulSoup(page3.text, 'html.parser')

        # The title is located in h1
        h1 = soup3.h1
        title = h1.contents[0].strip()
        dict_new['Title'] = title
        # The URL was in the element of to_process_list
        dict_new['URL'] = entry

        # The date is located in the class "mb-no"
        results3 = soup3.find_all('p', attrs={'class':'mb-no'})
        dict_new['Date'] = results3[0].text

        # Ticket symbols are located in the class "ticket-symbol"
        results4 = soup3.find_all('a', attrs={'class':'ticket-symbol'})

        # Creating a list of unique symbols in teh article
        symbols = []
        for symbol in results4:
            symbols.append(symbol.text)
        # Remove duplicates
        symbols = set(symbols)

        # If there are no symbols, set the value to None
        if not symbols:
            symbols = None

        dict_new['Symbols'] = symbols
        # Append this article's dictionary to the list
        new_data.append(dict_new)
    # Return the list of dictionaries
    return new_data
```

Figure 11 – Get_details Function

If new articles have been found on this scan I now do two things. First, I convert the dictionaries into a Pandas Data Fame and append it to the previously used csv file for article data. Since there is the possibility of there being more than one article retrieved within a one minute span, I first reverse the dataframe before appending to the file. This ensures that the order remains chronological – the file will list articles from oldest to newest. Next, I take the href of the latest article and set the variable, “latest_href”, with this value for future scans.

```python
"""
This function is called if the scan found new articles to be processed.  It takes the
list of dictionaries and creates a data frame of them.  Before appending to the csv
file it reverses the data frame, ensuring that the csv is written continuously from
oldest to newest.  Finally it resets latest_href to be the newest of these
unprocessed articles.
"""
def store_articles(article_dicts):
    # Take the list of dictionaries and convert to dataframe
    new_articles = pd.DataFrame(article_dicts)
    # Reverse so that proper order is maintained in the csv
    new_articles = new_articles[::-1]
    # Append to "article_data.csv"
    new_articles.to_csv('article_data.csv', mode='a', index=False, header=False)

    # Set the new latest href to be the newest unprocessed article
    global latest_href
    latest_href = new_articles['URL'][0]

    # Return the dataframe
    return new_articles
```

Figure 12 – Store_article Function

Next, for each of these unprocessed articles I use the dictionary of each to first print the information to the screen. If there are stock symbols, for each I use the previously made functions to first extract the stock price and volume information from Yahoo Finance, and then I use that to create the plots. This is done for each symbol found in each of the new articles.

Putting Everything Together

Upon starting of the program I print a welcome message and inform the user that they can stop the program by pressing "ESC" at any time. I then call my "initial_run" function and display the output.

```python
if __name__ == "__main__":
    # Upon running the module a welcome is printed to the user
    print('................')
    print('Hello!')
    print('I will scan PR Newswire every minute and show you new stock information!')
    print("Plots will be saved with stock symbols and today's date")
    # Inform the user how to stop the program
    print('Press "ESC" at any time to quit')
    print('................')
    print('\nHere is the latest article:\n')

    # Run the initial scan
    syms, title = initial_run()
    if not syms:
        pass
    # If the latest article contains stock symbols, get history and plot each
    else:
        for sym in syms:
            stock_history = get_tickers(sym)
            make_plots(stock_history, sym, title)
```

Figure 13 – Beginning the Module

Periodic Automatic Scanning

To accomplish this part of the task I make use of a keyboard listener and the time. I first create the variable "break_program", and set it to false. The periodic scanning will continue for however long this is false. I then define a function, "on_press", that uses Python's Keyboard. If the "ESC" key is pressed, a message will be printed to the screen and "break_program" will be set to true, thus ending the program.

```python
    # Enter the main body of the program
    # Set break_program to allow it to run continuously
    break_program = False

    """
    This function resolves allowing the user to stop the scan.
    If the user pressers the "ESC" key, a message is printed and break_program is
    set to true, ending the program.  All other key presses are ignored.
    """
    def on_press(key):
        global break_program
        if key == keyboard.Key.esc:
            print ('Shutting down....')
            print('Bye!')
            break_program = True
            return False
```

Figure 14 – On_press Function

I use this function and the keyboard listener and first get the starting time. On each iteration through the while loop I get the current time and see if 60 seconds has elapsed between it and the starting time. If not, I continue. If the 60 seconds has passed I first print a message that I am now scanning, and then start calling my functions – retrieving the hrefs, checking which are unprocessed, getting the details. If there are new articles, I print the information to the screen, and then retrieve and plot stock information if there are stocks listed. At the end of this I reset the starting time for the next iteration of the loop. As described, the program will repeat every 60 seconds provided the user does not press the “ESC” key.

```python
    # Set a keyboard listener using the above function
    with keyboard.Listener(on_press=on_press) as listener:
        # Get the starting time
        start = time.time()
        # Enter the main loop
        while break_program == False:
            # Since the program will loop continuously, add a tiny sleep here to
            # prevent the CPU from consuing resources
            time.sleep(0.25)
            # Check the current time
            current = time.time()
            # If one minute has passed since the last scan, it is time to scan again
            if current > (start + 60):
                print('Scanning....')

                # For each scan, first get the articles
                new_articles = get_new_news()
                # Then check for which are unprocessed
                unprocessed = check_if_new(new_articles)
                # Then get the details of these
                new_details = get_details(unprocessed)
                # If there are new articles, store the details to the csv file
                if new_details:
                    store_articles(new_details)

                # For each of the unprocessed articles, print the details
                for detail in new_details:
                    title = detail['Title']
                    day = detail['Date']
                    syms = detail['Symbols']
                    print("Title: ",title)
                    print('Date: ', day)
                    print('Stock Symbols Mentioned: ', syms)
                    print('\n')
                    # Check if the article contained any stock symbols
                    if not syms:
                        pass
                    else:
                        # If it did, for each stock symbol get the info and make the plots
                        for sym in syms:
                            stock_history = get_tickers(sym)
                            make_plots(stock_history, sym, title)

                # Lastly, reset the start time at the completion of this scan
                start = time.time()
            # If one minute has not passed since the last scan, do nothing
            else:
                continue

        listener.join()
```

Figure 15 – The Main Body of the Program

Sample Output

Upon running the module (here a stock symbol was found in the first article):

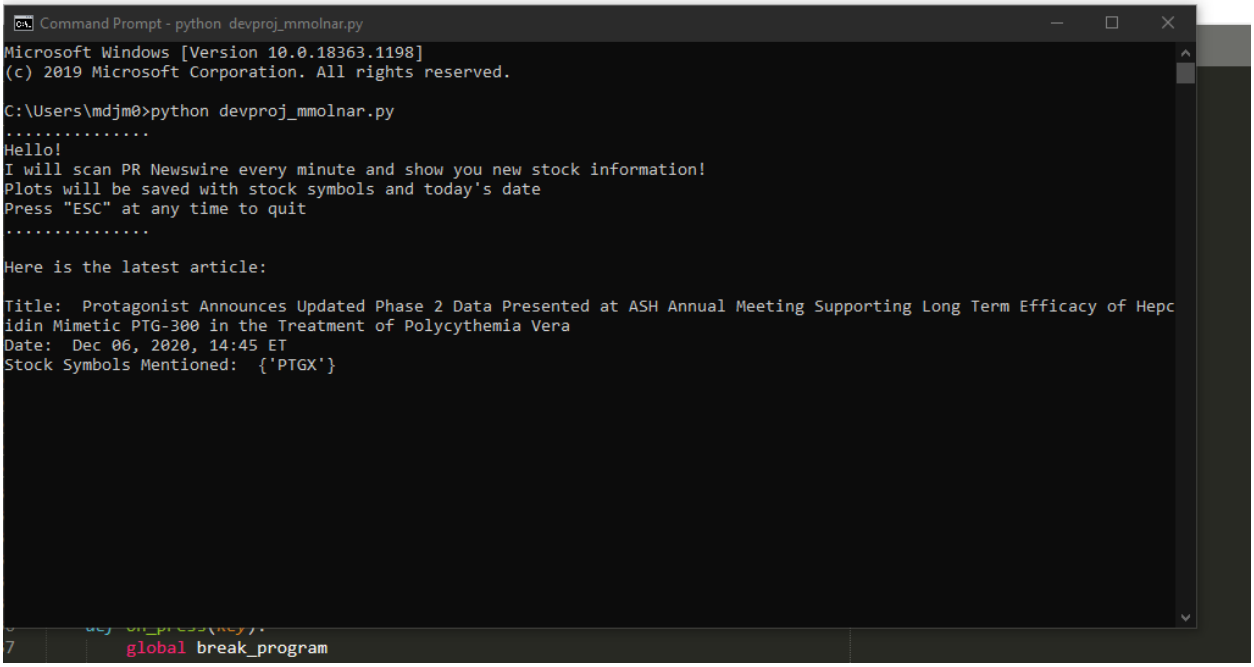


Figure 16 – Output Upon Running devproj_mmolnar.py

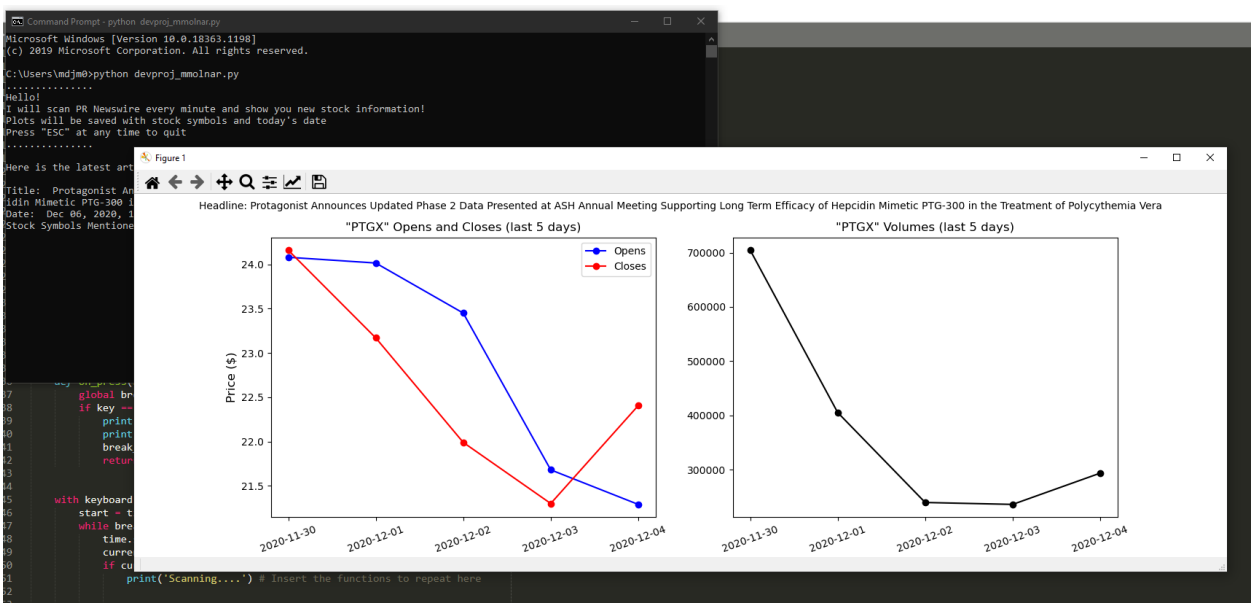


Figure 17 – Output When a Stock Symbol is Found

After one minute:

```
Command Prompt - python devproj_mmolnar.py
Microsoft Windows [Version 10.0.18363.1198]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\mdjm0>python devproj_mmolnar.py
...............
Hello!
I will scan PR Newswire every minute and show you new stock information!
Plots will be saved with stock symbols and today's date
Press "ESC" at any time to quit
...............

Here is the latest article:

Title:  Protagonist Announces Updated Phase 2 Data Presented at ASH Annual Meeting Supporting Long Term Efficacy of Hepc
idin Mimetic PTG-300 in the Treatment of Polycythemia Vera
Date:  Dec 06, 2020, 14:45 ET
Stock Symbols Mentioned:  {'PTGX'}


Scanning....
There is no new news
```

Figure 18 – It Scans Again After One Minute

After several minutes:

```
Command Prompt - python devproj_mmolnar.py
Microsoft Windows [Version 10.0.18363.1198]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\mdjm0>python devproj_mmolnar.py
...............
Hello!
I will scan PR Newswire every minute and show you new stock information!
Plots will be saved with stock symbols and today's date
Press "ESC" at any time to quit
...............

Here is the latest article:

Title:  Protagonist Announces Updated Phase 2 Data Presented at ASH Annual Meeting Supporting Long Term Efficacy of Hepc
idin Mimetic PTG-300 in the Treatment of Polycythemia Vera
Date:  Dec 06, 2020, 14:45 ET
Stock Symbols Mentioned:  {'PTGX'}


Scanning....
There is no new news

Scanning....
There is no new news

Scanning....
There is no new news

Scanning....
There is no new news

Scanning....
There is no new news

Scanning....
There is no new news

Scanning....
There is no new news

Scanning....
There is no new news

Scanning....
There is no new news

Scanning....
There is no new news

Scanning....
There is no new news

Scanning....
There is no new news
```

Figure 19 – Every Minute it Scans for New Articles

Upon finding an unprocessed article:

```
Scanning....
There is no new news

Scanning....
There is new news!

Title:  Fortlake Selects SS&C to Support Fixed Income Operations
Date:  Dec 06, 2020, 17:00 ET
Stock Symbols Mentioned:  {'SSNC'}


Scanning....
There is no new news
```

Figure 20 – Output When a New Article is Found

Upon pressing “ESC”:

```
Scanning....
There is new news!

Title:  Fortlake Selects SS&C to Support Fixed Income Operations
Date:  Dec 06, 2020, 17:00 ET
Stock Symbols Mentioned:  {'SSNC'}


Scanning....
There is no new news

Scanning....
There is no new news

Shutting down....
Bye!

C:\Users\mdjm0>
```

Figure 21 – Output When “ESC” is Pressed

The plots folder:

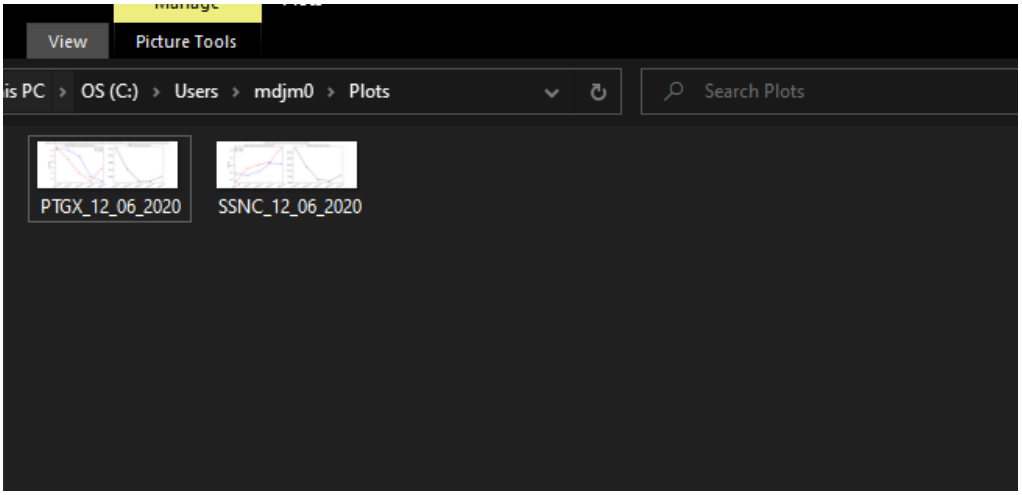


Figure 22 – The Plots Folder that All Plots are Saved To

A saved plot:

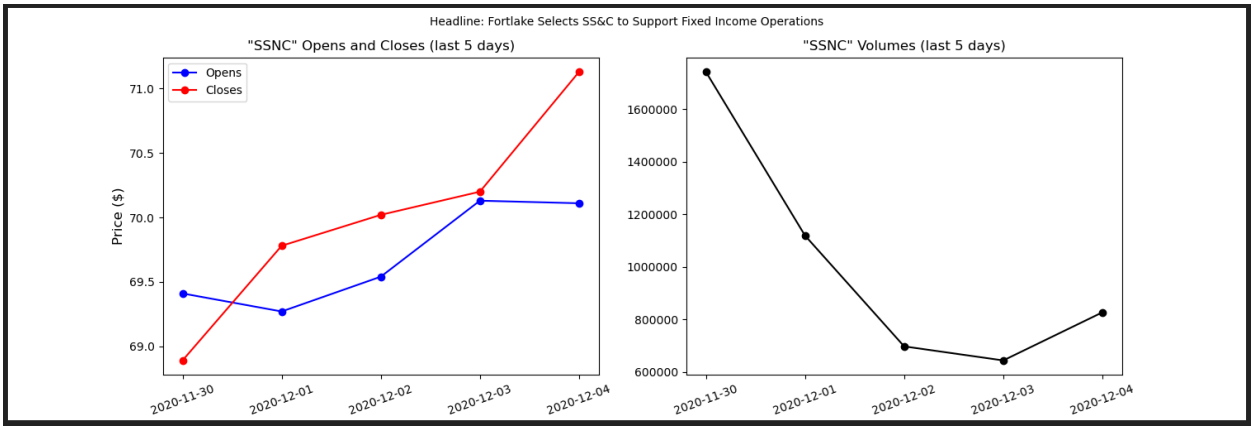


Figure 23 – The Saved Plot for the Second Found Article

Article_data.csv:

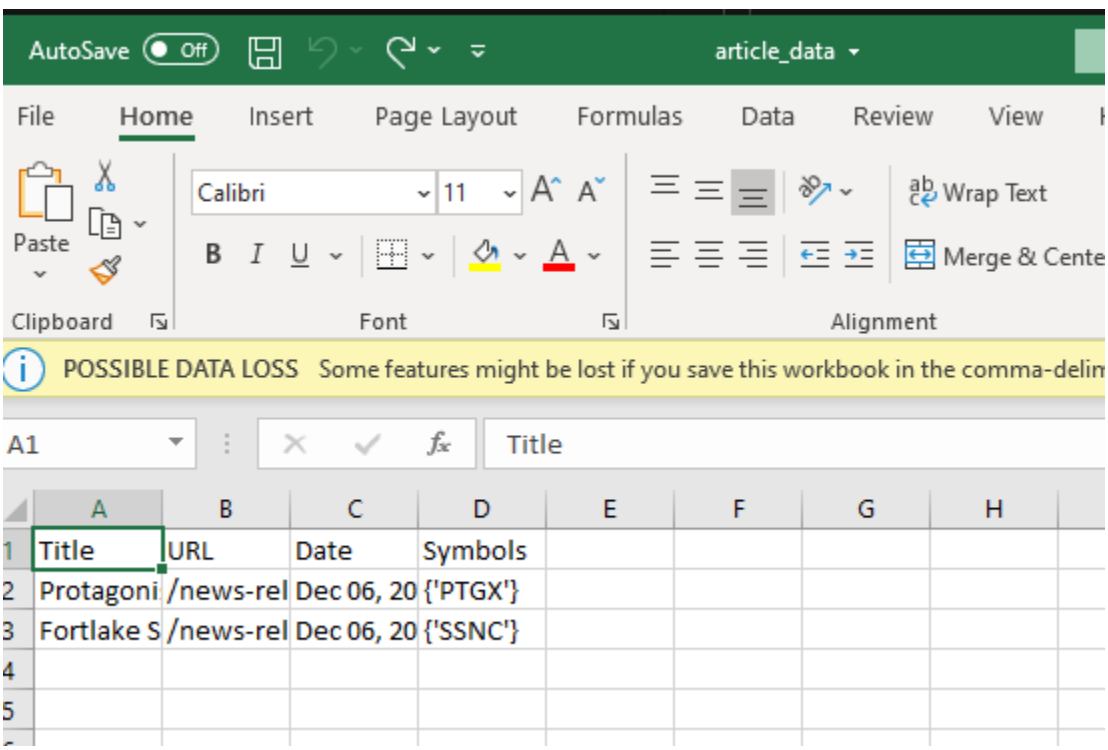
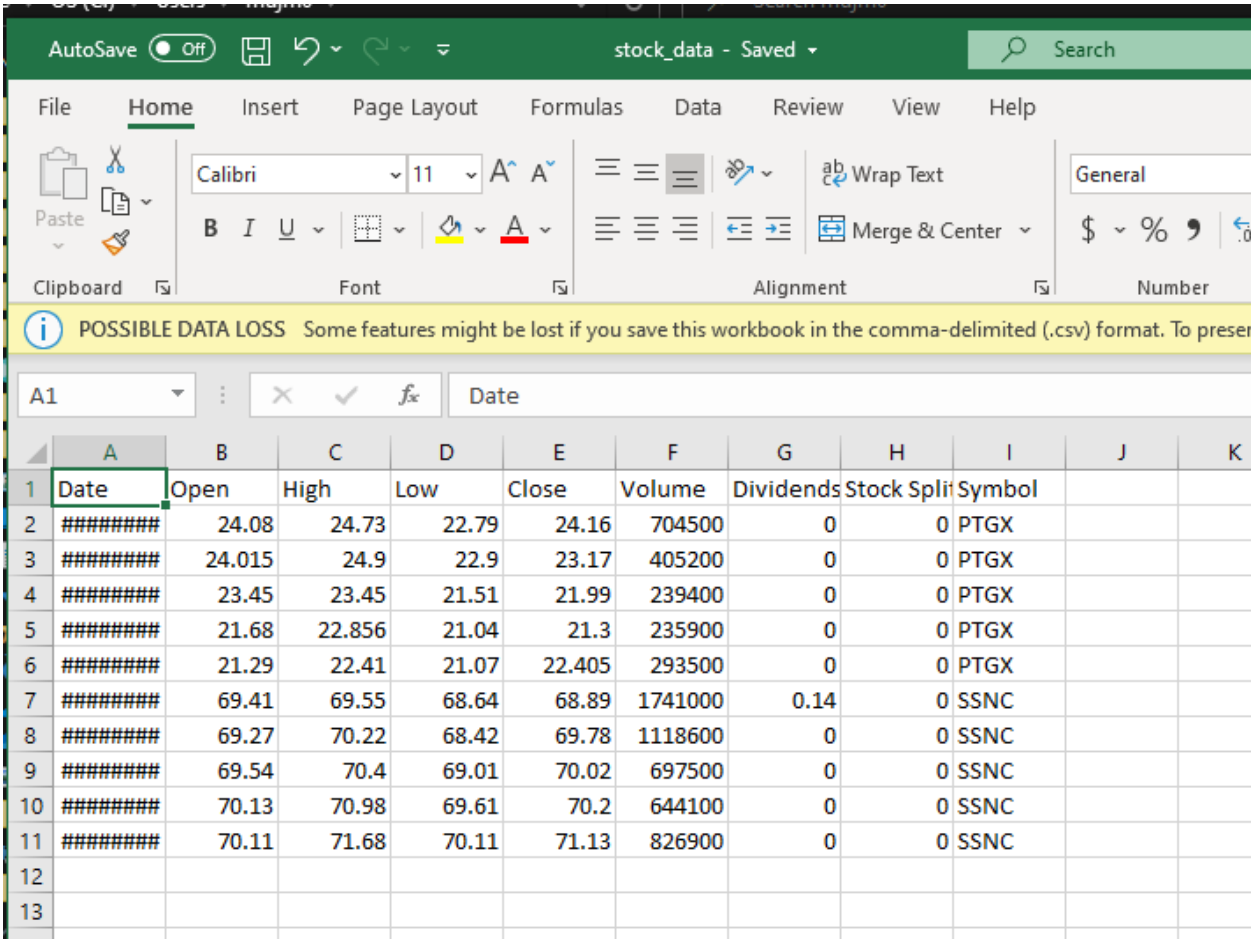


Figure 24 – Saved Article Data

Stock_data.csv:



	A	B	C	D	E	F	G	H	I
1	Date	Open	High	Low	Close	Volume	Dividends	Stock Split	Symbol
2	########	24.08	24.73	22.79	24.16	704500	0	0	PTGX
3	########	24.015	24.9	22.9	23.17	405200	0	0	PTGX
4	########	23.45	23.45	21.51	21.99	239400	0	0	PTGX
5	########	21.68	22.856	21.04	21.3	235900	0	0	PTGX
6	########	21.29	22.41	21.07	22.405	293500	0	0	PTGX
7	########	69.41	69.55	68.64	68.89	1741000	0.14	0	SSNC
8	########	69.27	70.22	68.42	69.78	1118600	0	0	SSNC
9	########	69.54	70.4	69.01	70.02	697500	0	0	SSNC
10	########	70.13	70.98	69.61	70.2	644100	0	0	SSNC
11	########	70.11	71.68	70.11	71.13	826900	0	0	SSNC
12									
13									

Figure 25 – Saved Stock Data

Reading with Pandas:

```
In [1]: import pandas as pd
        stock_data = pd.read_csv('stock_data.csv')

In [2]: stock_data
```

Out[2]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits	Symbol
0	2020-11-30	24.080000	24.730000	22.790001	24.160000	704500	0.00	0	PTGX
1	2020-12-01	24.014999	24.900000	22.900000	23.170000	405200	0.00	0	PTGX
2	2020-12-02	23.450001	23.450001	21.510000	21.990000	239400	0.00	0	PTGX
3	2020-12-03	21.680000	22.856001	21.040001	21.299999	235900	0.00	0	PTGX
4	2020-12-04	21.290001	22.410000	21.070000	22.405001	293500	0.00	0	PTGX
5	2020-11-30	69.410004	69.550003	68.639999	68.889999	1741000	0.14	0	SSNC
6	2020-12-01	69.269997	70.220001	68.419998	69.779999	1118600	0.00	0	SSNC
7	2020-12-02	69.540001	70.400002	69.010002	70.019997	697500	0.00	0	SSNC
8	2020-12-03	70.129997	70.980003	69.610001	70.199997	644100	0.00	0	SSNC
9	2020-12-04	70.110001	71.680000	70.110001	71.129997	826900	0.00	0	SSNC

Figure 26 – The Stock Data Read by Pandas