Final Assignment

May 12, 2025

Extracting and Visualizing Stock Data

Description

Extracting essential data from a dataset and displaying it is a necessary part of data science; therefore individuals can make correct decisions based on the data. In this assignment, you will extract some stock data, you will then display this data in a graph.

Table of Contents

```
<ul>
    <li>Define a Function that Makes a Graph</li>
    <li>Question 1: Use yfinance to Extract Stock Data</li>
    <li>Question 2: Use Webscraping to Extract Tesla Revenue Data</li>
    <li>Question 3: Use yfinance to Extract Stock Data</li>
    <li>Question 4: Use Webscraping to Extract GME Revenue Data</li>
    <li>Question 5: Plot Tesla Stock Graph</li>
    <li>Question 6: Plot GameStop Stock Graph</li>
</ul>
```

Estimated Time Needed: 30 min

Note:- If you are working Locally using anaconda, please uncomment the following code and execute it. Use the version as per your python version.

[38]:
```
!pip install yfinance
!pip install bs4
!pip install nbformat
!pip install --upgrade plotly
```

```
Requirement already satisfied: yfinance in /opt/conda/lib/python3.12/site-
packages (0.2.61)
Requirement already satisfied: pandas>=1.3.0 in /opt/conda/lib/python3.12/site-
packages (from yfinance) (2.2.3)
Requirement already satisfied: numpy>=1.16.5 in /opt/conda/lib/python3.12/site-
packages (from yfinance) (2.2.5)
Requirement already satisfied: requests>=2.31 in /opt/conda/lib/python3.12/site-
packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in
/opt/conda/lib/python3.12/site-packages (from yfinance) (0.0.11)
Requirement already satisfied: platformdirs>=2.0.0 in
/opt/conda/lib/python3.12/site-packages (from yfinance) (4.3.6)
```

```
Requirement already satisfied: pytz>=2022.5 in /opt/conda/lib/python3.12/site-
packages (from yfinance) (2024.2)
Requirement already satisfied: frozendict>=2.3.4 in
/opt/conda/lib/python3.12/site-packages (from yfinance) (2.4.6)
Requirement already satisfied: peewee>=3.16.2 in /opt/conda/lib/python3.12/site-
packages (from yfinance) (3.18.1)
Requirement already satisfied: beautifulsoup4>=4.11.1 in
/opt/conda/lib/python3.12/site-packages (from yfinance) (4.12.3)
Requirement already satisfied: curl_cffi>=0.7 in /opt/conda/lib/python3.12/site-
packages (from yfinance) (0.10.0)
Requirement already satisfied: protobuf>=3.19.0 in
/opt/conda/lib/python3.12/site-packages (from yfinance) (6.30.2)
Requirement already satisfied: websockets>=13.0 in
/opt/conda/lib/python3.12/site-packages (from yfinance) (15.0.1)
Requirement already satisfied: soupsieve>1.2 in /opt/conda/lib/python3.12/site-
packages (from beautifulsoup4>=4.11.1->yfinance) (2.5)
Requirement already satisfied: cffi>=1.12.0 in /opt/conda/lib/python3.12/site-
packages (from curl_cffi>=0.7->yfinance) (1.17.1)
Requirement already satisfied: certifi>=2024.2.2 in
/opt/conda/lib/python3.12/site-packages (from curl_cffi>=0.7->yfinance)
(2024.12.14)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.12/site-packages (from pandas>=1.3.0->yfinance)
(2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-
packages (from pandas>=1.3.0->yfinance) (2025.2)
Requirement already satisfied: charset_normalizer<4,>=2 in
/opt/conda/lib/python3.12/site-packages (from requests>=2.31->yfinance) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.12/site-
packages (from requests>=2.31->yfinance) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.12/site-packages (from requests>=2.31->yfinance) (2.3.0)
Requirement already satisfied: pycparser in /opt/conda/lib/python3.12/site-
packages (from cffi>=1.12.0->curl_cffi>=0.7->yfinance) (2.22)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas>=1.3.0->yfinance) (1.17.0)
Requirement already satisfied: bs4 in /opt/conda/lib/python3.12/site-packages
(0.0.2)
Requirement already satisfied: beautifulsoup4 in /opt/conda/lib/python3.12/site-
packages (from bs4) (4.12.3)
Requirement already satisfied: soupsieve>1.2 in /opt/conda/lib/python3.12/site-
packages (from beautifulsoup4->bs4) (2.5)
Requirement already satisfied: nbformat in /opt/conda/lib/python3.12/site-
packages (5.10.4)
Requirement already satisfied: fastjsonschema>=2.15 in
/opt/conda/lib/python3.12/site-packages (from nbformat) (2.21.1)
Requirement already satisfied: jsonschema>=2.6 in
/opt/conda/lib/python3.12/site-packages (from nbformat) (4.23.0)
```

```
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in
/opt/conda/lib/python3.12/site-packages (from nbformat) (5.7.2)
Requirement already satisfied: traitlets>=5.1 in /opt/conda/lib/python3.12/site-
packages (from nbformat) (5.14.3)
Requirement already satisfied: attrs>=22.2.0 in /opt/conda/lib/python3.12/site-
packages (from jsonschema>=2.6->nbformat) (25.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/opt/conda/lib/python3.12/site-packages (from jsonschema>=2.6->nbformat)
(2024.10.1)
Requirement already satisfied: referencing>=0.28.4 in
/opt/conda/lib/python3.12/site-packages (from jsonschema>=2.6->nbformat)
(0.36.2)
Requirement already satisfied: rpds-py>=0.7.1 in /opt/conda/lib/python3.12/site-
packages (from jsonschema>=2.6->nbformat) (0.22.3)
Requirement already satisfied: platformdirs>=2.5 in
/opt/conda/lib/python3.12/site-packages (from jupyter-
core!=5.0.*,>=4.12->nbformat) (4.3.6)
Requirement already satisfied: typing-extensions>=4.4.0 in
/opt/conda/lib/python3.12/site-packages (from
referencing>=0.28.4->jsonschema>=2.6->nbformat) (4.12.2)
Requirement already satisfied: plotly in /opt/conda/lib/python3.12/site-packages
(6.0.1)
Requirement already satisfied: narwhals>=1.15.1 in
/opt/conda/lib/python3.12/site-packages (from plotly) (1.39.0)
Requirement already satisfied: packaging in /opt/conda/lib/python3.12/site-
packages (from plotly) (24.2)
```

[39]:
```
import yfinance as yf
import pandas as pd
import requests
from bs4 import BeautifulSoup
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

[40]:
```
import plotly.io as pio
pio.renderers.default = "iframe"
```

[42]:
```
#In Python, you can ignore warnings using the warnings module. You can use the␣
↪filterwarnings function to filter or ignore specific warning messages or␣
↪categories.
```

[43]:
```
import warnings
# Ignore all warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

[70]:
```
!pip install matplotlib
```

```
Collecting matplotlib
```

```
  Downloading matplotlib-3.10.3-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.2-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.5 kB)
Collecting cycler>=0.10 (from matplotlib)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.58.0-cp312-cp312-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (104 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.8-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (6.2 kB)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-
packages (from matplotlib) (2.2.5)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib) (24.2)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-11.2.1-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (8.9
kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Downloading pyparsing-3.2.3-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Downloading
matplotlib-3.10.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(8.6 MB)
                         8.6/8.6 MB
185.2 MB/s eta 0:00:00
Downloading
contourpy-1.3.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (323
kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.58.0-cp312-cp312-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (4.9 MB)
                         4.9/4.9 MB
144.0 MB/s eta 0:00:00
Downloading
kiwisolver-1.4.8-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.5
MB)
                         1.5/1.5 MB
93.8 MB/s eta 0:00:00
Downloading pillow-11.2.1-cp312-cp312-manylinux_2_28_x86_64.whl (4.6 MB)
                         4.6/4.6 MB
```

```
158.1 MB/s eta 0:00:00
Downloading pyparsing-3.2.3-py3-none-any.whl (111 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler,
contourpy, matplotlib
Successfully installed contourpy-1.3.2 cycler-0.12.1 fonttools-4.58.0
kiwisolver-1.4.8 matplotlib-3.10.3 pillow-11.2.1 pyparsing-3.2.3
```

[71]:
```
import matplotlib.pyplot as plt
```

[83]:
```
!pip install html5lib
```

```
Requirement already satisfied: html5lib in /opt/conda/lib/python3.12/site-
packages (1.1)
Requirement already satisfied: six>=1.9 in /opt/conda/lib/python3.12/site-
packages (from html5lib) (1.17.0)
Requirement already satisfied: webencodings in /opt/conda/lib/python3.12/site-
packages (from html5lib) (0.5.1)
```

0.1 Define Graphing Function

In this section, we define the function `make_graph`. **You don't have to know how the function works, you should only care about the inputs. It takes a dataframe with stock data (dataframe must contain Date and Close columns), a dataframe with revenue data (dataframe must contain Date and Revenue columns), and the name of the stock.**

[44]:
```
def make_graph(stock_data, revenue_data, stock):
    fig = make_subplots(rows=2, cols=1, shared_xaxes=True,␣
 ↪subplot_titles=("Historical Share Price", "Historical Revenue"),␣
 ↪vertical_spacing = .3)
    stock_data_specific = stock_data[stock_data.Date <= '2021-06-14']
    revenue_data_specific = revenue_data[revenue_data.Date <= '2021-04-30']
    fig.add_trace(go.Scatter(x=pd.to_datetime(stock_data_specific.Date,␣
 ↪infer_datetime_format=True), y=stock_data_specific.Close.astype("float"),␣
 ↪name="Share Price"), row=1, col=1)
    fig.add_trace(go.Scatter(x=pd.to_datetime(revenue_data_specific.Date,␣
 ↪infer_datetime_format=True), y=revenue_data_specific.Revenue.
 ↪astype("float"), name="Revenue"), row=2, col=1)
    fig.update_xaxes(title_text="Date", row=1, col=1)
    fig.update_xaxes(title_text="Date", row=2, col=1)
    fig.update_yaxes(title_text="Price ($US)", row=1, col=1)
    fig.update_yaxes(title_text="Revenue ($US Millions)", row=2, col=1)
    fig.update_layout(showlegend=False,
    height=900,
    title=stock,
    xaxis_rangeslider_visible=True)
    fig.show()
    from IPython.display import display, HTML
    fig_html = fig.to_html()
    display(HTML(fig_html))
```

Use the make_graph function that we've already defined. You'll need to invoke it in questions 5 and 6 to display the graphs and create the dashboard. > **Note: You don't need to redefine the function for plotting graphs anywhere else in this notebook; just use the existing function.**

0.2 Question 1: Use yfinance to Extract Stock Data

Using the `Ticker` function enter the ticker symbol of the stock we want to extract data on to create a ticker object. The stock is Tesla and its ticker symbol is `TSLA`.

[85]:
```
import html5lib
```

[45]:
```
data = yf.Ticker("TSLA")
```

Using the ticker object and the function `history` extract stock information and save it in a dataframe named `tesla_data`. Set the `period` parameter to `"max"` so we get information for the maximum amount of time.

[46]:
```
tesla_data = data.history(period="max")
```

Reset the index using the `reset_index(inplace=True)` function on the tesla_data DataFrame and display the first five rows of the `tesla_data` dataframe using the `head` function. Take a screenshot of the results and code from the beginning of Question 1 to the results below.

[47]:
```
tesla_data.reset_index(inplace=True)
tesla_data.head()
```

[47]:
```
                       Date      Open      High       Low     Close  \
0 2010-06-29 00:00:00-04:00  1.266667  1.666667  1.169333  1.592667
1 2010-06-30 00:00:00-04:00  1.719333  2.028000  1.553333  1.588667
2 2010-07-01 00:00:00-04:00  1.666667  1.728000  1.351333  1.464000
3 2010-07-02 00:00:00-04:00  1.533333  1.540000  1.247333  1.280000
4 2010-07-06 00:00:00-04:00  1.333333  1.333333  1.055333  1.074000

      Volume  Dividends  Stock Splits
0  281494500        0.0           0.0
1  257806500        0.0           0.0
2  123282000        0.0           0.0
3   77097000        0.0           0.0
4  103003500        0.0           0.0
```

0.3 Question 2: Use Webscraping to Extract Tesla Revenue Data

Use the `requests` library to download the webpage https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/revenue.htm Save the text of the response as a variable named `html_data`.

[75]:
```python
url = " https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
 ↪IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/revenue.htm"
html_data = requests.get(url).text
```

Parse the html data using `beautiful_soup` using parser i.e `html5lib` or `html.parser`.

[76]:
```python
soup = BeautifulSoup(html_data,'html.parser')
```

Using `BeautifulSoup` or the `read_html` function extract the table with `Tesla Revenue` and store it into a dataframe named `tesla_revenue`. The dataframe should have columns `Date` and `Revenue`.

Step-by-step instructions

```
Here are the step-by-step instructions:

1. Create an Empty DataFrame
2. Find the Relevant Table
3. Check for the Tesla Quarterly Revenue Table
4. Iterate Through Rows in the Table Body
5. Extract Data from Columns
6. Append Data to the DataFrame
```

Click here if you need help locating the table

```
Below is the code to isolate the table, you will now need to loop through the rows and columns

soup.find_all("tbody")[1]

If you want to use the read_html function the table is located at index 1

We are focusing on quarterly revenue in the lab.
```

[79]:
```python
table = soup.find('table') # in html table is represented by the tag <table>
tables = soup.find_all('table')

# Step 3: Create a list to collect row data
revenue_data = []

# Step 4: Extract Tesla Revenue from second table
for row in tables[1].tbody.find_all("tr"):
    cols = row.find_all("td")
    if cols:
        date = cols[0].text.strip()
        revenue = cols[1].text.strip()
        revenue_data.append({"Date": date, "Revenue": revenue})

# Step 5: Convert the list into a DataFrame
tesla_revenue = pd.DataFrame(revenue_data)
```

```
# Step 6: Clean the Revenue column
tesla_revenue["Revenue"] = tesla_revenue["Revenue"].str.replace(',|\$', '',␣
↪regex=True)
tesla_revenue.dropna(inplace=True)
tesla_revenue = tesla_revenue[tesla_revenue["Revenue"] != ""]

# Step 7: Show last 5 rows
tesla_revenue.tail()
```

[79]:

```
          Date Revenue
48  2010-09-30      31
49  2010-06-30      28
50  2010-03-31      21
52  2009-09-30      46
53  2009-06-30      27
```

Execute the following line to remove the comma and dollar sign from the `Revenue` column.

[51]:

```
tesla_revenue["Revenue"] = tesla_revenue['Revenue'].str.replace(',|\$',"")
```

Execute the following lines to remove an null or empty strings in the Revenue column.

[52]:

```
tesla_revenue.dropna(inplace=True)

tesla_revenue = tesla_revenue[tesla_revenue['Revenue'] != ""]
```

Display the last 5 row of the `tesla_revenue` dataframe using the `tail` function. Take a screenshot of the results.

[66]:

```
tesla_revenue.tail()
```

[66]:

```
Empty DataFrame
Columns: [Date, Revenue]
Index: []
```

0.4 Question 3: Use yfinance to Extract Stock Data

Using the `Ticker` function enter the ticker symbol of the stock we want to extract data on to create a ticker object. The stock is GameStop and its ticker symbol is `GME`.

[54]:

```
gme = yf.Ticker("GME")
```

Using the ticker object and the function `history` extract stock information and save it in a dataframe named `gme_data`. Set the `period` parameter to `"max"` so we get information for the maximum amount of time.

[55]:

```
gme_data = gme.history(period = "max")
```

Reset the index using the `reset_index(inplace=True)` function on the gme_data DataFrame and display the first five rows of the `gme_data` dataframe using the `head` function. Take a screenshot of the results and code from the beginning of Question 3 to the results below.

```
[56]: gme_data.reset_index(inplace = True)
      gme_data.head()
```

```
[56]:                        Date      Open      High       Low     Close    Volume  \
      0 2002-02-13 00:00:00-05:00  1.620129  1.693350  1.603296  1.691667  76216000
      1 2002-02-14 00:00:00-05:00  1.712707  1.716074  1.670626  1.683250  11021600
      2 2002-02-15 00:00:00-05:00  1.683250  1.687458  1.658002  1.674834   8389600
      3 2002-02-19 00:00:00-05:00  1.666417  1.666417  1.578047  1.607504   7410400
      4 2002-02-20 00:00:00-05:00  1.615921  1.662210  1.603296  1.662210   6892800

         Dividends  Stock Splits
      0        0.0           0.0
      1        0.0           0.0
      2        0.0           0.0
      3        0.0           0.0
      4        0.0           0.0
```

0.5 Question 4: Use Webscraping to Extract GME Revenue Data

Use the `requests` library to download the webpage https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/stock.html. Save the text of the response as a variable named `html_data_2`.

```
[57]: url = " https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
       ↪IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/stock.html"
      html_data_2 = requests.get(url).text
```

Parse the html data using `beautiful_soup` using parser i.e `html5lib` or `html.parser`.

```
[87]: soup = BeautifulSoup(html_data_2,'html.parser')
```

```
[89]: table = soup.find('table') # in html table is represented by the tag <table>
      tables = soup.find_all('table')

      # Step 3: Create a list to collect row data
      revenue_data = []

      # Step 4: Extract Tesla Revenue from second table
      for row in tables[1].tbody.find_all("tr"):
          cols = row.find_all("td")
          if cols:
              date = cols[0].text.strip()
              revenue = cols[1].text.strip()
```

```
        revenue_data.append({"Date": date, "Revenue": revenue})

# Step 5: Convert the list into a DataFrame
gme_revenue = pd.DataFrame(revenue_data)

# Step 6: Clean the Revenue column
gme_revenue["Revenue"] = gme_revenue["Revenue"].str.replace(',|\$', '',␣
 ↪regex=True)
gme_revenue.dropna(inplace=True)
gme_revenue = gme_revenue[gme_revenue["Revenue"] != ""]

# Step 7: Show last 5 rows
tesla_revenue.tail()
```

[89]:

```
          Date Revenue
48  2010-09-30      31
49  2010-06-30      28
50  2010-03-31      21
52  2009-09-30      46
53  2009-06-30      27
```

Using `BeautifulSoup` or the `read_html` function extract the table with `GameStop Revenue` and store it into a dataframe named `gme_revenue`. The dataframe should have columns `Date` and `Revenue`. Make sure the comma and dollar sign is removed from the `Revenue` column.

Note: Use the method similar to what you did in question 2.

Click here if you need help locating the table

```
Below is the code to isolate the table, you will now need to loop through the rows and columns

soup.find_all("tbody")[1]

If you want to use the read_html function the table is located at index 1
```

[62]:

```
Gamestop_revenue = soup.find_all('tables')
```

[74]:

```
gme_revenue = pd.DataFrame(Gamestop_revenue,columns = ['Date','Revenue'])
```

Display the last five rows of the `gme_revenue` dataframe using the `tail` function. Take a screenshot of the results.

[67]:

```
gme_revenue.tail()
```

[67]:

```
Empty DataFrame
Columns: [Date, Revenue]
Index: []
```

0.6 Question 5: Plot Tesla Stock Graph

Use the `make_graph` function to graph the Tesla Stock Data, also provide a title for the graph. Note the graph will only show data upto June 2021.

Hint

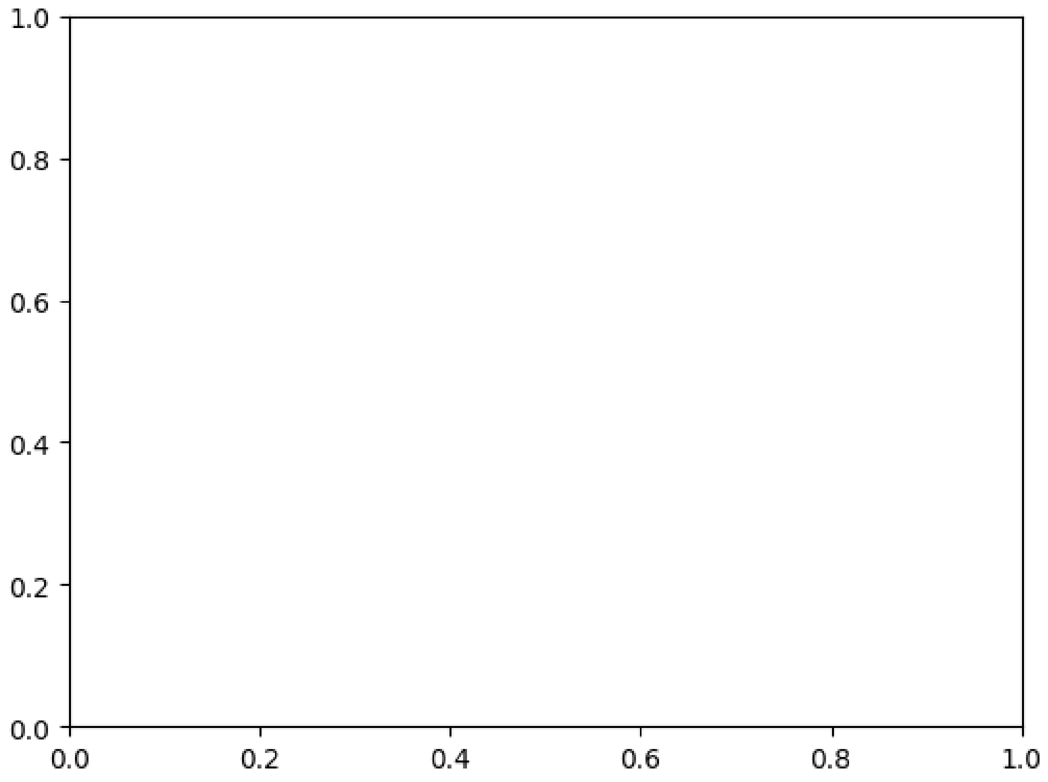
```
You just need to invoke the make_graph function with the required parameter to print the graphs
```

```
[72]: plt.plot(make_graph(tesla_data, tesla_revenue, 'Tesla'))
```

```
/tmp/ipykernel_363/109047474.py:5: UserWarning:

The argument 'infer_datetime_format' is deprecated and will be removed in a
future version. A strict version of it is now the default, see
https://pandas.pydata.org/pdeps/0004-consistent-to-datetime-parsing.html. You
can safely remove this argument.

/tmp/ipykernel_363/109047474.py:6: UserWarning:

The argument 'infer_datetime_format' is deprecated and will be removed in a
future version. A strict version of it is now the default, see
https://pandas.pydata.org/pdeps/0004-consistent-to-datetime-parsing.html. You
can safely remove this argument.


<IPython.core.display.HTML object>
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[72], line 1
----> 1 plt.plot(make_graph(tesla_data, tesla_revenue, 'Tesla'))

File /opt/conda/lib/python3.12/site-packages/matplotlib/pyplot.py:3838, in␣
 ↪plot(scalex, scaley, data, *args, **kwargs)
   3830 @_copy_docstring_and_deprecators(Axes.plot)
   3831 def plot(
   3832     *args: float | ArrayLike | str,
   (…)
   3836     **kwargs,
   3837 ) -> list[Line2D]:
-> 3838     return gca().plot(
   3839         *args,
   3840         scalex=scalex,
   3841         scaley=scaley,
   3842         **({"data": data} if data is not None else {}),
   3843         **kwargs,
   3844     )
```

```
File /opt/conda/lib/python3.12/site-packages/matplotlib/axes/_axes.py:1777, in␣
↪Axes.plot(self, scalex, scaley, data, *args, **kwargs)
   1534 """
   1535 Plot y versus x as lines and/or markers.
   1536
   (…)
   1774 (``'green'``) or hex strings (``'#008000'``).
   1775 """
   1776 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1777 lines = [*self._get_lines(self, *args, data=data, **kwargs)]
   1778 for line in lines:
   1779     self.add_line(line)

File /opt/conda/lib/python3.12/site-packages/matplotlib/axes/_base.py:297, in␣
↪_process_plot_var_args.__call__(self, axes, data, return_kwargs, *args,␣
↪**kwargs)
    295     this += args[0],
    296     args = args[1:]
--> 297 yield from self._plot_args(
    298     axes, this, kwargs, ambiguous_fmt_datakey=ambiguous_fmt_datakey,
    299     return_kwargs=return_kwargs
    300 )

File /opt/conda/lib/python3.12/site-packages/matplotlib/axes/_base.py:455, in␣
↪_process_plot_var_args._plot_args(self, axes, tup, kwargs, return_kwargs,␣
↪ambiguous_fmt_datakey)
    452 # Don't allow any None value; these would be up-converted to one
    453 # element array of None which causes problems downstream.
    454 if any(v is None for v in tup):
--> 455     raise ValueError("x, y, and format string must not be None")
    457 kw = {}
    458 for prop_name, val in zip(('linestyle', 'marker', 'color'),
    459                           (linestyle, marker, color)):

ValueError: x, y, and format string must not be None
```



0.7 Question 6: Plot GameStop Stock Graph

Use the `make_graph` function to graph the GameStop Stock Data, also provide a title for the graph. The structure to call the `make_graph` function is `make_graph(gme_data, gme_revenue, 'GameStop')`. Note the graph will only show data upto June 2021.

Hint

```
You just need to invoke the make_graph function with the required parameter to print the graphs
```

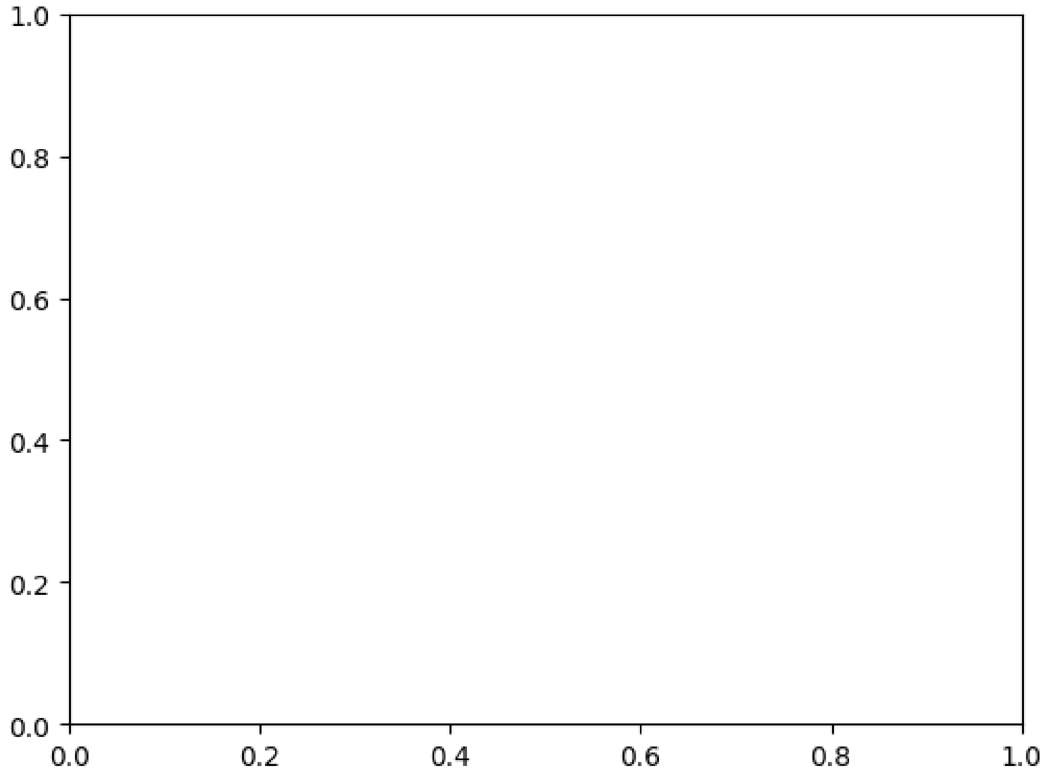
[73]:
```
plt.plot(make_graph(gme_data, gme_revenue, 'GameStop'))
```

```
/tmp/ipykernel_363/109047474.py:5: UserWarning:

The argument 'infer_datetime_format' is deprecated and will be removed in a
future version. A strict version of it is now the default, see
https://pandas.pydata.org/pdeps/0004-consistent-to-datetime-parsing.html. You
can safely remove this argument.

/tmp/ipykernel_363/109047474.py:6: UserWarning:

The argument 'infer_datetime_format' is deprecated and will be removed in a
future version. A strict version of it is now the default, see
```

```
https://pandas.pydata.org/pdeps/0004-consistent-to-datetime-parsing.html. You
can safely remove this argument.
```

```
<IPython.core.display.HTML object>
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[73], line 1
----> 1 plt.plot(make_graph(gme_data, gme_revenue, 'GameStop'))

File /opt/conda/lib/python3.12/site-packages/matplotlib/pyplot.py:3838, in␣
 ↪plot(scalex, scaley, data, *args, **kwargs)
   3830 @_copy_docstring_and_deprecators(Axes.plot)
   3831 def plot(
   3832     *args: float | ArrayLike | str,
   (…)
   3836     **kwargs,
   3837 ) -> list[Line2D]:
-> 3838     return gca().plot(
   3839         *args,
   3840         scalex=scalex,
   3841         scaley=scaley,
   3842         **({"data": data} if data is not None else {}),
   3843         **kwargs,
   3844     )

File /opt/conda/lib/python3.12/site-packages/matplotlib/axes/_axes.py:1777, in␣
 ↪Axes.plot(self, scalex, scaley, data, *args, **kwargs)
   1534 """
   1535 Plot y versus x as lines and/or markers.
   1536
   (…)
   1774 (``'green'``) or hex strings (``'#008000'``).
   1775 """
   1776 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1777 lines = [*self._get_lines(self, *args, data=data, **kwargs)]
   1778 for line in lines:
   1779     self.add_line(line)

File /opt/conda/lib/python3.12/site-packages/matplotlib/axes/_base.py:297, in␣
 ↪_process_plot_var_args.__call__(self, axes, data, return_kwargs, *args,␣
 ↪**kwargs)
    295     this += args[0],
    296     args = args[1:]
--> 297 yield from self._plot_args(
    298     axes, this, kwargs, ambiguous_fmt_datakey=ambiguous_fmt_datakey,
    299     return_kwargs=return_kwargs
    300 )

File /opt/conda/lib/python3.12/site-packages/matplotlib/axes/_base.py:455, in␣
 ↪_process_plot_var_args._plot_args(self, axes, tup, kwargs, return_kwargs,␣
 ↪ambiguous_fmt_datakey)
    452 # Don't allow any None value; these would be up-converted to one
    453 # element array of None which causes problems downstream.
    454 if any(v is None for v in tup):
--> 455     raise ValueError("x, y, and format string must not be None")
    457 kw = {}
    458 for prop_name, val in zip(('linestyle', 'marker', 'color'),
    459                           (linestyle, marker, color)):

ValueError: x, y, and format string must not be None
```



About the Authors:

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Azim Hirjani

0.8 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-02-28	1.2	Lakshmi Holla	Changed the URL of GameStop
2020-11-10	1.1	Malika Singla	Deleted the Optional part
2020-08-27	1.0	Malika Singla	Added lab to GitLab

##