



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

- Define a Function that Makes a Graph
- Question 1: Use yfinance to Extract Stock Data
- Question 2: Use Webscraping to Extract Tesla Revenue Data
- Question 3: Use yfinance to Extract Stock Data
- Question 4: Use Webscraping to Extract GME Revenue Data
- Question 5: Plot Tesla Stock Graph
- Question 6: Plot GameStop Stock Graph

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.

```
In [3]: !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 (0.2.63)

Requirement already satisfied: pandas>=1.3.0 in /opt/conda/lib/python3.12/site-packages (from yfinance) (2.3.0)

Requirement already satisfied: numpy>=1.16.5 in /opt/conda/lib/python3.12/site-packages (from yfinance) (2.3.1)

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) (0.11.4)

Requirement already satisfied: protobuf>=3.19.0 in /opt/conda/lib/python3.12/site-packages (from yfinance) (6.31.1)

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 (6.1.2)
 Requirement already satisfied: narwhals>=1.15.1 in /opt/conda/lib/python3.12/site-packages (from plotly) (1.43.1)
 Requirement already satisfied: packaging in /opt/conda/lib/python3.12/site-packages (from plotly) (24.2)

```
In [4]: 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
```

```
In [5]: import plotly.io as pio
pio.renderers.default = "iframe"
```

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.

```
In [6]: import warnings
# Ignore all warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

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.

```
In [7]: def make_graph(stock_data, revenue_data, stock):
    fig = make_subplots(rows=2, cols=1, shared_xaxes=True, subplot_titles=(
    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
    fig.add_trace(go.Scatter(x=pd.to_datetime(revenue_data_specific.Date, infer
    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.

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`.

```
In [9]: tesla = 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.

```
In [10]: tesla_data = tesla.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.

```
In [11]: tesla_data.reset_index(inplace=True)
tesla_data.head()
```

Out [11]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2010-06-29 00:00:00-04:00	1.266667	1.666667	1.169333	1.592667	281494500	0.0	0.0
1	2010-06-30 00:00:00-04:00	1.719333	2.028000	1.553333	1.588667	257806500	0.0	0.0
2	2010-07-01 00:00:00-04:00	1.666667	1.728000	1.351333	1.464000	123282000	0.0	0.0
3	2010-07-02 00:00:00-04:00	1.533333	1.540000	1.247333	1.280000	77097000	0.0	0.0
4	2010-07-06 00:00:00-04:00	1.333333	1.333333	1.055333	1.074000	103003500	0.0	0.0

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`.

```
In [19]: url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IE
html_data = requests.get(url).text
```

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

```
In [20]: 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

► Click here if you need help locating the table

```
In [29]: # Create an Empty DataFrame
tesla_revenue = pd.DataFrame(columns=["Date", "Revenue"])

# Find the Relevant Table
tables = soup.find_all("table")

#Check for the Tesla Quarterly Revenue Table
for table in tables:
    if "Tesla Quarterly Revenue" in table.text:
        target_table = table
        break

#Iterate Through Rows in the Table Body
for row in target_table.tbody.find_all("tr"):
    cols = row.find_all("td")
    # Note:
    # The condition `if len(cols) == 2:` is used during data extraction to ensure
    # only rows with exactly two <td> elements (Date and Revenue) are processed.
    # This helps avoid errors when the table contains unexpected or malformed rows.
    # In short, it filters out structurally incomplete rows during collection.
    if len(cols) == 2:
        date = cols[0].text.strip()
        revenue = cols[1].text.strip()

    # Append Data to the DataFrame
    tesla_revenue = pd.concat([tesla_revenue, pd.DataFrame({"Date": [date], "Revenue": [revenue]})])

#Display the first few rows
tesla_revenue.head()
```

Out [29]:

	Date	Revenue
0	2022-09-30	\$21,454
1	2022-06-30	\$16,934
2	2022-03-31	\$18,756
3	2021-12-31	\$17,719
4	2021-09-30	\$13,757

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

```
In [30]: tesla_revenue["Revenue"] = tesla_revenue["Revenue"].str.replace(',', '\$', "", regex=True)
```

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

```
In [31]: ## tesla_revenue.dropna(inplace=True)
# tesla_revenue = tesla_revenue[tesla_revenue["Revenue"] != ""]
# are part of the data cleaning process after the DataFrame has been created
# They remove rows where 'Revenue' is either missing (NaN) or an empty string
# In short, it removes empty or missing values after collection
```

```
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.

```
In [32]: tesla_revenue.tail()
```

```
Out[32]:
```

	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

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`.

```
In [12]: GameStop = 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.

```
In [14]: gme_data = GameStop.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.

```
In [16]: gme_data.reset_index(inplace=True)
gme_data.head()
```

Out [16]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2002-02-13 00:00:00-05:00	1.620128	1.693350	1.603296	1.691666	76216000	0.0	0.0
1	2002-02-14 00:00:00-05:00	1.712707	1.716074	1.670626	1.683251	11021600	0.0	0.0
2	2002-02-15 00:00:00-05:00	1.683250	1.687458	1.658002	1.674834	8389600	0.0	0.0
3	2002-02-19 00:00:00-05:00	1.666418	1.666418	1.578047	1.607504	7410400	0.0	0.0
4	2002-02-20 00:00:00-05:00	1.615921	1.662210	1.603296	1.662210	6892800	0.0	0.0

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`.

```
In [34]: url2 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/stock.html"
html_data_2 = requests.get(url2).text
```

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

```
In [35]: soup = BeautifulSoup(html_data_2, 'html.parser')
```

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](#)

```
In [39]: gme_revenue = pd.DataFrame(columns=["Date", "Revenue"])

for row in soup.find("tbody").find_all('tr'):
    cols = row.find_all("td")
    if len(cols) == 2:
        date = cols[0].text.strip()
        revenue = cols[1].text.strip()
        gme_revenue = pd.concat([gme_revenue, pd.DataFrame({"Date": [date], "Revenue": [revenue]})])

#remove the comma and dollar sign
gme_revenue["Revenue"] = gme_revenue["Revenue"].str.replace(',|\$', '', regex=True)

#remove an null or empty strings
gme_revenue.dropna(inplace=True)

gme_revenue = gme_revenue[gme_revenue["Revenue"] != ""]

##Display the first few rows
gme_revenue.head()
```

```
Out[39]:
```

	Date	Revenue
0	2021	53823
1	2020	31536
2	2019	24578
3	2018	21461
4	2017	11759

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

```
In [40]: gme_revenue.tail()
```

```
Out[40]:
```

	Date	Revenue
8	2013	2013
9	2012	413
10	2011	204
11	2010	117
12	2009	112

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

```
In [42]: make_graph(tesla_data, tesla_revenue, 'Tesla')
```

```
/tmp/ipykernel_299/109047474.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_299/109047474.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.
```





Tesla

His



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

```
In [43]: make_graph(gme_data, gme_revenue, 'GameStop')
```

```
/tmp/ipykernel_299/109047474.py:5: UserWarning:
```

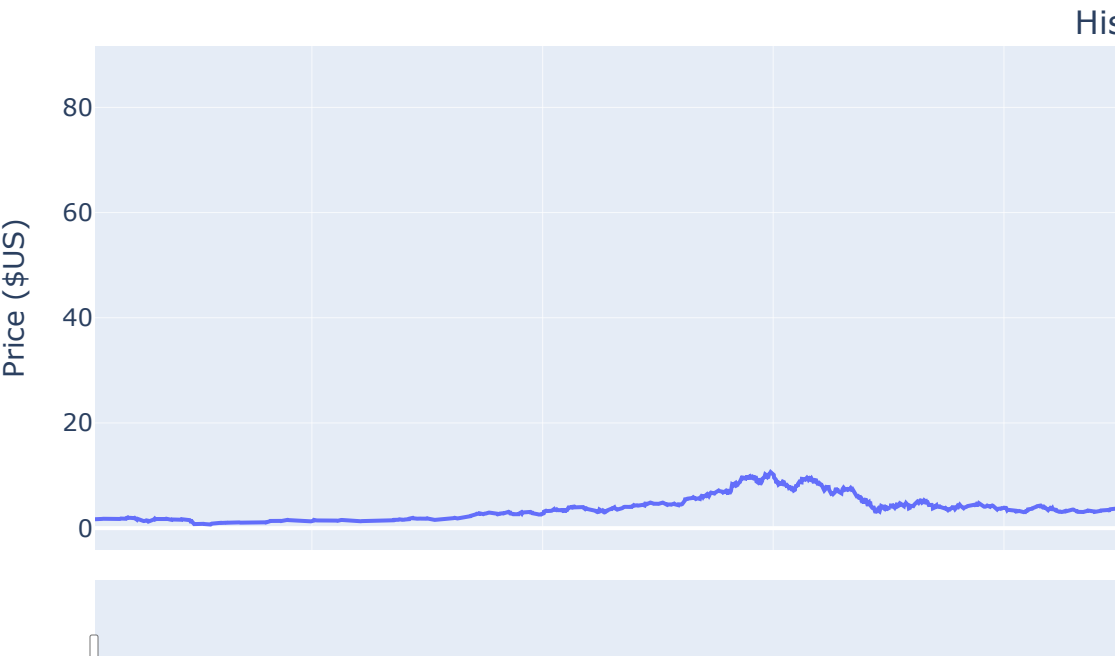
```
The argument 'infer_datetime_format' is deprecated and will be removed in a future version. A strict version of it is now the default, see https://pandas.pydata.org/pdeps/0004-consistent-to-datetime-parsing.html. You can safely remove this argument.
```

```
/tmp/ipykernel_299/109047474.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.
```



GameStop



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

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

© IBM Corporation 2020. All rights reserved.