Final Assignment

August 31, 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

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.

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

[ ]:
```
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
```

[ ]:
```
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.

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

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.**

[ ]:
```
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`.

[5]:
```
!pip install yfinance
!pip install pandas
```

```
import yfinance as yf
import pandas as pd

Tesla = yf.Ticker("TSLA")
print(dir(Tesla))
```

```
Collecting yfinance
  Downloading yfinance-0.2.65-py2.py3-none-any.whl.metadata (5.8 kB)
Collecting pandas>=1.3.0 (from yfinance)
  Downloading
pandas-2.3.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(91 kB)
Collecting numpy>=1.16.5 (from yfinance)
  Downloading
numpy-2.3.2-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata
(62 kB)
Requirement already satisfied: requests>=2.31 in /opt/conda/lib/python3.12/site-
packages (from yfinance) (2.32.3)
Collecting multitasking>=0.0.7 (from yfinance)
  Downloading multitasking-0.0.12.tar.gz (19 kB)
  Preparing metadata (setup.py) … done
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)
Collecting peewee>=3.16.2 (from yfinance)
  Downloading peewee-3.18.2.tar.gz (949 kB)
                           949.2/949.2 kB
50.9 MB/s eta 0:00:00
  Installing build dependencies … one
  Getting requirements to build wheel … done
  Preparing metadata (pyproject.toml) … done
Requirement already satisfied: beautifulsoup4>=4.11.1 in
/opt/conda/lib/python3.12/site-packages (from yfinance) (4.12.3)
Collecting curl_cffi>=0.7 (from yfinance)
  Downloading curl_cffi-0.13.0-cp39-abi3-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (13 kB)
Collecting protobuf>=3.19.0 (from yfinance)
  Downloading protobuf-6.32.0-cp39-abi3-manylinux2014_x86_64.whl.metadata (593
bytes)
Collecting websockets>=13.0 (from yfinance)
  Downloading websockets-15.0.1-cp312-cp312-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (6.8 kB)
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)
Collecting tzdata>=2022.7 (from pandas>=1.3.0->yfinance)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
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)
Downloading yfinance-0.2.65-py2.py3-none-any.whl (119 kB)
Downloading
curl_cffi-0.13.0-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (8.3
MB)
                         8.3/8.3 MB
155.1 MB/s eta 0:00:00
Downloading
numpy-2.3.2-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (16.6
MB)
                         16.6/16.6 MB
195.6 MB/s eta 0:00:00
Downloading
pandas-2.3.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.0
MB)
                         12.0/12.0 MB
201.2 MB/s eta 0:00:00
Downloading protobuf-6.32.0-cp39-abi3-manylinux2014_x86_64.whl (322 kB)
Downloading websockets-15.0.1-cp312-cp312-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (182 kB)
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Building wheels for collected packages: multitasking, peewee
  Building wheel for multitasking (setup.py) … one
  Created wheel for multitasking: filename=multitasking-0.0.12-py3-none-
any.whl size=15605
sha256=c7cf86c8378d21aa4b5e34a07f1962a32a631583fb064d68d255571ed676a993
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/cc/bd/6f/664d62c99327a
```

```
beef7d86489e6631cbf45b56fbf7ef1d6ef00
  Building wheel for peewee (pyproject.toml) … one
  Created wheel for peewee:
filename=peewee-3.18.2-cp312-cp312-linux_x86_64.whl size=303862
sha256=f42d2a1c192e5d3479e55dd7b36fe14a326b2ca9d5a93507611eef417f330783
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/d1/df/a9/0202b051c65b1
1c992dd6db9f2babdd2c44ec7d35d511be5d3
Successfully built multitasking peewee
Installing collected packages: peewee, multitasking, websockets, tzdata,
protobuf, numpy, pandas, curl_cffi, yfinance
Successfully installed curl_cffi-0.13.0 multitasking-0.0.12 numpy-2.3.2
pandas-2.3.2 peewee-3.18.2 protobuf-6.32.0 tzdata-2025.2 websockets-15.0.1
yfinance-0.2.65
Requirement already satisfied: pandas in /opt/conda/lib/python3.12/site-packages
(2.3.2)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-
packages (from pandas) (2.3.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-
packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_analysis',
'_data', '_download_options', '_earnings', '_earnings_dates', '_expirations',
'_fast_info', '_fetch_ticker_tz', '_financials', '_fundamentals', '_funds_data',
'_get_ticker_tz', '_holders', '_isin', '_lazy_load_price_history',
'_message_handler', '_news', '_options2df', '_price_history', '_quote',
'_shares', '_tz', '_underlying', 'actions', 'analyst_price_targets',
'balance_sheet', 'balancesheet', 'calendar', 'capital_gains', 'cash_flow',
'cashflow', 'dividends', 'earnings', 'earnings_dates', 'earnings_estimate',
'earnings_history', 'eps_revisions', 'eps_trend', 'fast_info', 'financials',
'funds_data', 'get_actions', 'get_analyst_price_targets', 'get_balance_sheet',
'get_balancesheet', 'get_calendar', 'get_capital_gains', 'get_cash_flow',
'get_cashflow', 'get_dividends', 'get_earnings', 'get_earnings_dates',
'get_earnings_estimate', 'get_earnings_history', 'get_eps_revisions',
'get_eps_trend', 'get_fast_info', 'get_financials', 'get_funds_data',
'get_growth_estimates', 'get_history_metadata', 'get_income_stmt',
'get_incomestmt', 'get_info', 'get_insider_purchases',
'get_insider_roster_holders', 'get_insider_transactions',
'get_institutional_holders', 'get_isin', 'get_major_holders',
'get_mutualfund_holders', 'get_news', 'get_recommendations',
```

```
'get_recommendations_summary', 'get_revenue_estimate', 'get_sec_filings',
'get_shares', 'get_shares_full', 'get_splits', 'get_sustainability',
'get_upgrades_downgrades', 'growth_estimates', 'history', 'history_metadata',
'income_stmt', 'incomestmt', 'info', 'insider_purchases',
'insider_roster_holders', 'insider_transactions', 'institutional_holders',
'isin', 'live', 'major_holders', 'mutualfund_holders', 'news', 'option_chain',
'options', 'quarterly_balance_sheet', 'quarterly_balancesheet',
'quarterly_cash_flow', 'quarterly_cashflow', 'quarterly_earnings',
'quarterly_financials', 'quarterly_income_stmt', 'quarterly_incomestmt',
'recommendations', 'recommendations_summary', 'revenue_estimate', 'sec_filings',
'session', 'shares', 'splits', 'sustainability', 'ticker', 'ttm_cash_flow',
'ttm_cashflow', 'ttm_financials', 'ttm_income_stmt', 'ttm_incomestmt',
'upgrades_downgrades', 'ws']
```

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.

[7]:
```
data = Tesla.history(period = "max")
print(data)
```

```
                                 Open        High         Low       Close  \
Date
2010-06-29 00:00:00-04:00    1.266667    1.666667    1.169333    1.592667
2010-06-30 00:00:00-04:00    1.719333    2.028000    1.553333    1.588667
2010-07-01 00:00:00-04:00    1.666667    1.728000    1.351333    1.464000
2010-07-02 00:00:00-04:00    1.533333    1.540000    1.247333    1.280000
2010-07-06 00:00:00-04:00    1.333333    1.333333    1.055333    1.074000
…                                 …           …           …           …
2025-08-25 00:00:00-04:00  338.899994  349.529999  335.029999  346.600006
2025-08-26 00:00:00-04:00  344.929993  351.899994  343.720001  351.670013
2025-08-27 00:00:00-04:00  351.940002  355.390015  349.160004  349.600006
2025-08-28 00:00:00-04:00  350.910004  353.549988  340.260010  345.980011
2025-08-29 00:00:00-04:00  347.230011  348.750000  331.700012  333.869995

                              Volume  Dividends  Stock Splits
Date
2010-06-29 00:00:00-04:00  281494500        0.0           0.0
2010-06-30 00:00:00-04:00  257806500        0.0           0.0
2010-07-01 00:00:00-04:00  123282000        0.0           0.0
2010-07-02 00:00:00-04:00   77097000        0.0           0.0
2010-07-06 00:00:00-04:00  103003500        0.0           0.0
…                                …          …             …
2025-08-25 00:00:00-04:00   86670000        0.0           0.0
2025-08-26 00:00:00-04:00   76651600        0.0           0.0
2025-08-27 00:00:00-04:00   65519000        0.0           0.0
2025-08-28 00:00:00-04:00   67903200        0.0           0.0
2025-08-29 00:00:00-04:00   80949700        0.0           0.0
```

[3817 rows x 7 columns]

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.

[13]:
```
data.reset_index(inplace=True)
data.head()
```

[13]:

	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

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`.

[16]:
```
import requests
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
 ↪IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/revenue.htm"
response = requests.get(url)
html_data = response.text
```

Cell In[15], line 2 url = https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/revenue.htm”response = requests.get(url) ^ SyntaxError: unterminated string literal (detected at line 2)

[6]:
```
!pip install lxml
import requests
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
 ↪IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/revenue.htm"
response = requests.get(url)
html_data = response.text
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_data,"lxml")
```

```
print(soup.title.text)
```

```
Requirement already satisfied: lxml in /opt/conda/lib/python3.12/site-packages
(6.0.1)
Tesla Revenue 2010-2022 | TSLA | MacroTrends
```

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.

[27]:
```
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
 ↪IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/revenue.htm"
response = requests.get(url)
html_data = response.text

soup = BeautifulSoup(html_data, "html.parser")
tables = soup.find_all("table")
tesla_table = tables[1]
rows = tesla_table.find_all("tr")
data = []
for r in rows:
    cells = [c.get_text(strip=True) for c in r.find_all(["th", "td"])]
    if cells:
        data.append(cells)

tesla_revenue = pd.DataFrame(data[1:], columns=["Date", "Revenue"]).dropna().
 ↪reset_index(drop=True)
```

```
tesla_revenue.head()
```

[27]:

	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.

[32]:
```
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
 ↪IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/revenue.htm"
response = requests.get(url)
html_data = response.text

soup = BeautifulSoup(html_data, "html.parser")
tables = soup.find_all("table")
tesla_table = tables[1]
rows = tesla_table.find_all("tr")
data = []
for r in rows:
    cells = [c.get_text(strip=True) for c in r.find_all(["th", "td"])]
    if cells:
        data.append(cells)

tesla_revenue = pd.DataFrame(data[1:], columns=["Date", "Revenue"]).dropna().
 ↪reset_index(drop=True)

tesla_revenue.head()
tesla_revenue["Revenue"] = tesla_revenue["Revenue"].str.replace(r',|\$', "",␣
 ↪regex=True)
```

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

[29]:
```
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.

[30]:
```
tesla_revenue.tail()
```

[30]:

	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
```

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`.

[31]:
```
import yfinance as yf
GameStop = yf.Ticker("GME")
print(dir(GameStop))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_analysis',
'_data', '_download_options', '_earnings', '_earnings_dates', '_expirations',
'_fast_info', '_fetch_ticker_tz', '_financials', '_fundamentals', '_funds_data',
'_get_ticker_tz', '_holders', '_isin', '_lazy_load_price_history',
'_message_handler', '_news', '_options2df', '_price_history', '_quote',
'_shares', '_tz', '_underlying', 'actions', 'analyst_price_targets',
'balance_sheet', 'balancesheet', 'calendar', 'capital_gains', 'cash_flow',
'cashflow', 'dividends', 'earnings', 'earnings_dates', 'earnings_estimate',
'earnings_history', 'eps_revisions', 'eps_trend', 'fast_info', 'financials',
'funds_data', 'get_actions', 'get_analyst_price_targets', 'get_balance_sheet',
'get_balancesheet', 'get_calendar', 'get_capital_gains', 'get_cash_flow',
'get_cashflow', 'get_dividends', 'get_earnings', 'get_earnings_dates',
'get_earnings_estimate', 'get_earnings_history', 'get_eps_revisions',
'get_eps_trend', 'get_fast_info', 'get_financials', 'get_funds_data',
'get_growth_estimates', 'get_history_metadata', 'get_income_stmt',
'get_incomestmt', 'get_info', 'get_insider_purchases',
'get_insider_roster_holders', 'get_insider_transactions',
'get_institutional_holders', 'get_isin', 'get_major_holders',
'get_mutualfund_holders', 'get_news', 'get_recommendations',
'get_recommendations_summary', 'get_revenue_estimate', 'get_sec_filings',
'get_shares', 'get_shares_full', 'get_splits', 'get_sustainability',
'get_upgrades_downgrades', 'growth_estimates', 'history', 'history_metadata',
'income_stmt', 'incomestmt', 'info', 'insider_purchases',
'insider_roster_holders', 'insider_transactions', 'institutional_holders',
'isin', 'live', 'major_holders', 'mutualfund_holders', 'news', 'option_chain',
'options', 'quarterly_balance_sheet', 'quarterly_balancesheet',
'quarterly_cash_flow', 'quarterly_cashflow', 'quarterly_earnings',
'quarterly_financials', 'quarterly_income_stmt', 'quarterly_incomestmt',
'recommendations', 'recommendations_summary', 'revenue_estimate', 'sec_filings',
'session', 'shares', 'splits', 'sustainability', 'ticker', 'ttm_cash_flow',
'ttm_cashflow', 'ttm_financials', 'ttm_income_stmt', 'ttm_incomestmt',
'upgrades_downgrades', 'ws']
```

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.

[35]:
```
gme_data = GameStop.history(period="max")
print(gme_data)
```

```
                                 Open       High        Low      Close  \
Date
2002-02-13 00:00:00-05:00    1.620129   1.693350   1.603296   1.691667
2002-02-14 00:00:00-05:00    1.712707   1.716074   1.670626   1.683250
2002-02-15 00:00:00-05:00    1.683251   1.687459   1.658002   1.674834
2002-02-19 00:00:00-05:00    1.666418   1.666418   1.578047   1.607504
2002-02-20 00:00:00-05:00    1.615920   1.662210   1.603296   1.662210
…                                 …          …          …          …
2025-08-25 00:00:00-04:00   22.860001  22.900000  22.559999  22.680000
2025-08-26 00:00:00-04:00   22.670000  22.799999  22.270000  22.299999
2025-08-27 00:00:00-04:00   22.459999  22.580000  22.270000  22.500000
2025-08-28 00:00:00-04:00   22.559999  22.900000  22.370001  22.790001
2025-08-29 00:00:00-04:00   22.719999  22.820000  22.410000  22.410000

                              Volume  Dividends  Stock Splits
Date
2002-02-13 00:00:00-05:00  76216000        0.0           0.0
2002-02-14 00:00:00-05:00  11021600        0.0           0.0
2002-02-15 00:00:00-05:00   8389600        0.0           0.0
2002-02-19 00:00:00-05:00   7410400        0.0           0.0
2002-02-20 00:00:00-05:00   6892800        0.0           0.0
…                               …          …           …
2025-08-25 00:00:00-04:00   3801000        0.0           0.0
2025-08-26 00:00:00-04:00   7867600        0.0           0.0
2025-08-27 00:00:00-04:00   6892000        0.0           0.0
2025-08-28 00:00:00-04:00   6482300        0.0           0.0
2025-08-29 00:00:00-04:00   5357700        0.0           0.0

[5925 rows x 7 columns]
```

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.

[40]:
```
gme_data.reset_index(inplace=True)
gme_data.head()
```

[40]:

	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	1.683251	1.687459	1.658002	1.674834	8389600

```
3 2002-02-19 00:00:00-05:00  1.666418  1.666418  1.578047  1.607504   7410400
4 2002-02-20 00:00:00-05:00  1.615920  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`.

[42]:
```
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
 ↪IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/stock.html"
df = pd.read_html(url)
```

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

[46]:
```
from bs4 import BeautifulSoup
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
 ↪IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/stock.html"
response = requests.get(url)

soup= BeautifulSoup(response.text,"html.parser")
print(soup.title.text)
```

```
GameStop Revenue 2006-2020 | GME | MacroTrends
```

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
```

[28]:
```
table = soup.find_all("table")
gme_revenue = table[1]
rows = gme_revenue.find_all("tr")
data = []
for row in rows:
    col = [c.get_text(strip=True)  for c in row.find_all(["td","th"])]
    if col:
        data.append(col)

gme_revenue = pd.DataFrame(data[1:], columns=["Date", "Revenue"])
gme_revenue["Revenue"] = gme_revenue["Revenue"].replace({'\$':'', ',':''},␣
 ↪regex=True)
print(gme_revenue.head())
```

```
         Date Revenue
0  2022-09-30   21454
1  2022-06-30   16934
2  2022-03-31   18756
3  2021-12-31   17719
4  2021-09-30   13757
```

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

[51]:
```
gme_revenue.tail()
```

[51]:

	Date	Revenue
57	2006-01-31	1667
58	2005-10-31	534
59	2005-07-31	416
60	2005-04-30	475
61	2005-01-31	709

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.

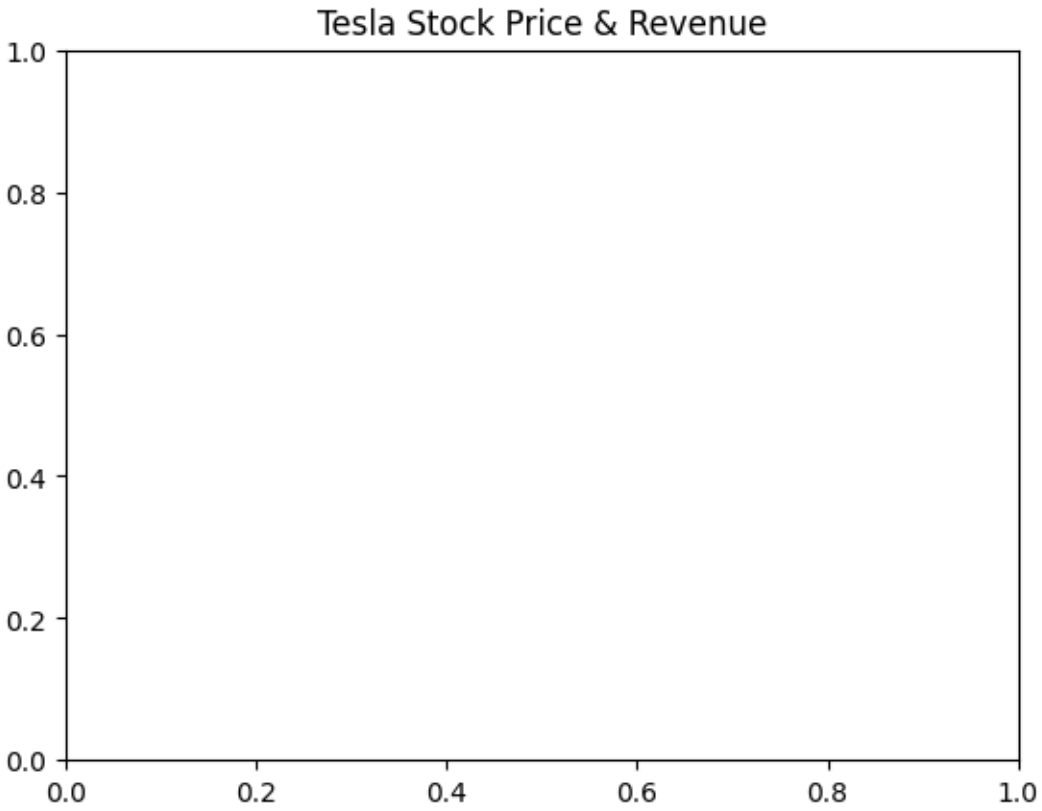
Hint

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

[19]:
```
plt.title("Tesla Stock Price & Revenue")
make_graph(tesla_data, tesla_revenue, "Tesla")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[19], line 2
      1 plt.title("Tesla Stock Price & Revenue")
```

```
----> 2 make_graph(tesla_data, tesla_revenue, "Tesla")

NameError: name 'tesla_data' is not defined
```



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

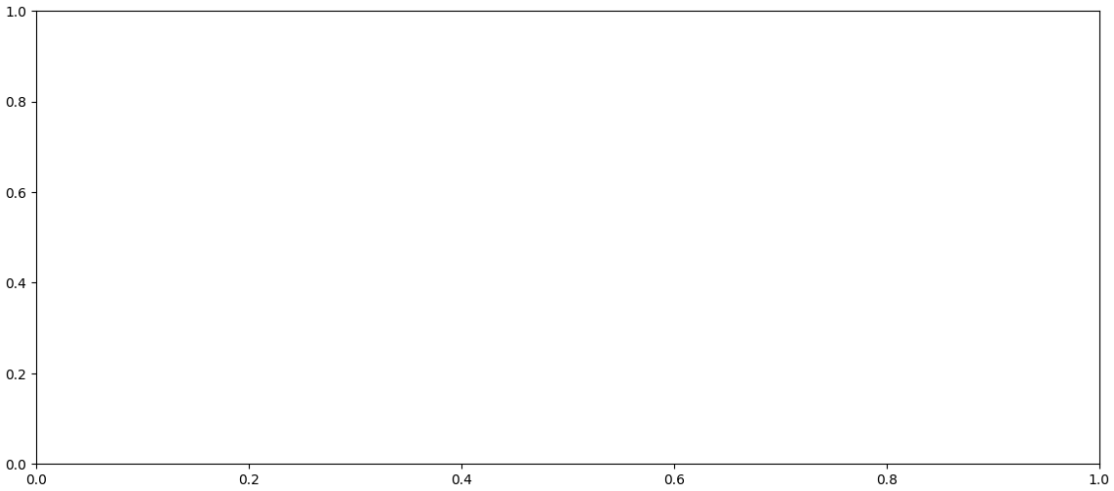
You just need to invoke the make_graph function with the required parameter to print the graphs

[24]:
```
make_graph(gme_data, gme_revenue, 'GameStop')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File /opt/conda/lib/python3.12/site-packages/pandas/core/indexes/base.py:3812,␣
 ↪in Index.get_loc(self, key)
   3811 try:
```

```
-> 3812     return self._engine.get_loc(casted_key)
   3813 except KeyError as err:

File pandas/_libs/index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7096, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'Date'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[24], line 1
----> 1 make_graph(gme_data, gme_revenue, 'GameStop')

Cell In[13], line 7, in make_graph(stock_data, revenue_data, stock_name)
      4 fig, ax1 = plt.subplots(figsize=(14,6))
      6 # Plot stock price (blue line)
----> 7 ax1.plot(stock_data['Date'], stock_data['Close'], 'b-')
      8 ax1.set_xlabel('Date')
      9 ax1.set_ylabel('Stock Price', color='b')

File /opt/conda/lib/python3.12/site-packages/pandas/core/frame.py:4107, in␣
 ↪DataFrame.__getitem__(self, key)
   4105 if self.columns.nlevels > 1:
   4106     return self._getitem_multilevel(key)
-> 4107 indexer = self.columns.get_loc(key)
   4108 if is_integer(indexer):
   4109     indexer = [indexer]

File /opt/conda/lib/python3.12/site-packages/pandas/core/indexes/base.py:3819,␣
 ↪in Index.get_loc(self, key)
   3814     if isinstance(casted_key, slice) or (
   3815         isinstance(casted_key, abc.Iterable)
   3816         and any(isinstance(x, slice) for x in casted_key)
   3817     ):
   3818         raise InvalidIndexError(key)
-> 3819     raise KeyError(key) from err
   3820 except TypeError:
   3821     # If we have a listlike key, _check_indexing_error will raise
   3822     #  InvalidIndexError. Otherwise we fall through and re-raise
   3823     #  the TypeError.
```

```
    3824     self._check_indexing_error(key)

KeyError: 'Date'
```



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

##