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.

```
#!pip install yfinance==0.2.38
#!pip install pandas==2.2.2
#!pip install nbformat
```

```
!pip install yfinance
!pip install bs4
!pip install nbformat
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.12/dist-packages (0.2.66)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from yfinance) (2.2.2)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.12/dist-packages (from yfinance) (2.0.2)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.12/dist-packages (from yfinance) (2.32.
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.12/dist-packages (from yfinance) (
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from yfinance) (
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.12/dist-packages (from yfinance) (2025.2)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.12/dist-packages (from yfinance) (2.
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.12/dist-packages (from yfinance) (3.18.
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.12/dist-packages (from yfinance
Requirement already satisfied: curl_cffi>=0.7 in /usr/local/lib/python3.12/dist-packages (from yfinance) (0.13.
Requirement already satisfied: protobuf>=3.19.0 in /usr/local/lib/python3.12/dist-packages (from yfinance) (5.2
Requirement already satisfied: websockets>=13.0 in /usr/local/lib/python3.12/dist-packages (from yfinance) (15.
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup4>=4
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.12/dist-packages (from beauti
Requirement already satisfied: cffi>=1.12.0 in /usr/local/lib/python3.12/dist-packages (from curl_cffi>=0.7->yf
Requirement already satisfied: certifi>=2024.2.2 in /usr/local/lib/python3.12/dist-packages (from curl_cffi>=0.
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas>=
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.3.0->y
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from reques
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.31->yf
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-packages (from cffi>=1.12.0->curl_cf
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2
Collecting bs4
  Downloading bs4-0.0.2-py2.py3-none-any.whl.metadata (411 bytes)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.12/dist-packages (from bs4) (4.13.5)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup4->b
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.12/dist-packages (from beauti
Downloading bs4-0.0.2-py2.py3-none-any.whl (1.2 kB)
Installing collected packages: bs4
Successfully installed bs4-0.0.2
Requirement already satisfied: nbformat in /usr/local/lib/python3.12/dist-packages (5.10.4)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.12/dist-packages (from nbformat)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.12/dist-packages (from nbformat) (4.25
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in /usr/local/lib/python3.12/dist-packages (from nbfo
```

```
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.12/dist-packages (from nbformat) (5.7.1
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.12/dist-packages (from jsonschema>=2.6->
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.12/dist-packages (from jsonschema>
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.12/dist-packages (from jsonschema>=2.6-
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.12/dist-packages (from jupyter-core!
Requirement already satisfied: typing-extensions>=4.4.0 in /usr/local/lib/python3.12/dist-packages (from refere
```

```
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 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)
```

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", "Historic
    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), y=stock_data_specific.Close.astype("f
    fig.add_trace(go.Scatter(x=pd.to_datetime(revenue_data_specific.Date), y=revenue_data_specific.Revenue.ast
    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()
```

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`.

```
# Create a ticker object for Tesla
ticker = 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.

```
# Retrieve historical stock data for the maximum period
tesla_data = ticker.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.

```
# Reset index to move Date from index to column
tesla_data.reset_index(inplace=True)
# Display the first five rows of the DataFrame
print(tesla_data.head())
```

```
                       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
```

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`.

```
url="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0220EN-Ski
html_data = requests.get(url).text
#print(html_data[:500])
```

Parse the html data using `beautiful_soup` using parser i.e `html5lib` or `html.parser`. Make sure to use the `html_data` with the content parameter as follow `html_data.content` .

```
soup = BeautifulSoup(html_data, 'html.parser')
#print(soup.prettify()[:500])
```

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

```
# Initialize an empty DataFrame with columns "Date" and "Revenue"
tesla_revenue = pd.DataFrame(columns=['Date', 'Revenue'])

# Find all tables on the webpage
tables = soup.find_all('table')

# Loop through each table to find the relevant one
for table in tables:
    if "Tesla Quarterly Revenue" in table.get_text():
        # Isolate the body of the table
        tbody = table.find('tbody')

        if tbody:
            # Loop through each row in the table body
            for row in tbody.find_all('tr'):
                # Find all column values for each row
                col = row.find_all('td')
                if len(col) >= 2:
                    date = col[0].text.strip()
                    revenue = col[1].text.strip()

                    # Create a DataFrame for this row
                    row_df = pd.DataFrame({"Date": [date], "Revenue": [revenue]})

                    # Append the row DataFrame to the main DataFrame
                    tesla_revenue = pd.concat([tesla_revenue, row_df], ignore_index=True)

# Display the DataFrame after populating
```

```
print("DataFrame after populating:")
print(tesla_revenue)
```

```
DataFrame after populating:
          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
5   2021-06-30  $11,958
6   2021-03-31  $10,389
7   2020-12-31  $10,744
8   2020-09-30   $8,771
9   2020-06-30   $6,036
10  2020-03-31   $5,985
11  2019-12-31   $7,384
12  2019-09-30   $6,303
13  2019-06-30   $6,350
14  2019-03-31   $4,541
15  2018-12-31   $7,226
16  2018-09-30   $6,824
17  2018-06-30   $4,002
18  2018-03-31   $3,409
19  2017-12-31   $3,288
20  2017-09-30   $2,985
21  2017-06-30   $2,790
22  2017-03-31   $2,696
23  2016-12-31   $2,285
24  2016-09-30   $2,298
25  2016-06-30   $1,270
26  2016-03-31   $1,147
27  2015-12-31   $1,214
28  2015-09-30     $937
29  2015-06-30     $955
30  2015-03-31     $940
31  2014-12-31     $957
32  2014-09-30     $852
33  2014-06-30     $769
34  2014-03-31     $621
35  2013-12-31     $615
36  2013-09-30     $431
37  2013-06-30     $405
38  2013-03-31     $562
39  2012-12-31     $306
40  2012-09-30      $50
41  2012-06-30      $27
42  2012-03-31      $30
43  2011-12-31      $39
44  2011-09-30      $58
45  2011-06-30      $58
46  2011-03-31      $49
47  2010-12-31      $36
48  2010-09-30      $31
49  2010-06-30      $28
50  2010-03-31      $21
51  2009-12-31
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.

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

```
<>:1: SyntaxWarning: invalid escape sequence '\$'
<>:1: SyntaxWarning: invalid escape sequence '\$'
/tmp/ipython-input-1877950674.py:1: SyntaxWarning: invalid escape sequence '\$'
  tesla_revenue["Revenue"] = tesla_revenue['Revenue'].str.replace(',|\$',"", regex=True)
```

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

```
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.

```
print(tesla_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
```

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`.

```
import yfinance as yf

# Create a ticker object for GameStop using its ticker symbol 'GME'
gme = yf.Ticker('GME')

# Print the ticker object to verify
print(gme)
```

```
yfinance.Ticker object <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.

```
# Fetch historical stock data for the maximum available time period
gme_data = gme.history(period="max")

# Display the first few rows of the DataFrame to verify
print(gme_data.head())
```

```
                              Open      High       Low     Close    Volume  \
Date
2002-02-13 00:00:00-05:00  1.620129  1.693350  1.603296  1.691667  76216000
2002-02-14 00:00:00-05:00  1.712707  1.716074  1.670626  1.683250  11021600
2002-02-15 00:00:00-05:00  1.683250  1.687458  1.658002  1.674834   8389600
2002-02-19 00:00:00-05:00  1.666418  1.666418  1.578047  1.607504   7410400
2002-02-20 00:00:00-05:00  1.615920  1.662210  1.603296  1.662210   6892800

                           Dividends  Stock Splits
Date
2002-02-13 00:00:00-05:00        0.0           0.0
2002-02-14 00:00:00-05:00        0.0           0.0
2002-02-15 00:00:00-05:00        0.0           0.0
2002-02-19 00:00:00-05:00        0.0           0.0
2002-02-20 00:00:00-05:00        0.0           0.0
```

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.

```
# Reset the index of the DataFrame
gme_data.reset_index(inplace=True)

# Display the first five rows of the DataFrame
print(gme_data.head())
```

```
                       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  1.666418  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
```

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`.

```
url= "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0220EN-Sk
html_data_2 = requests.get(url).text
html_data_2[:500]
```

```
'<!DOCTYPE html>\n<!-- saved from url=(0105)https://web.archive.org/web/20200814131437/https://www.macrotrend
s.net/stocks/charts/GME/gamestop/revenue -->\n<html class=" js flexbox canvas canvastext webgl no-touch geoloc
ation postmessage websqldatabase indexeddb hashchange history draganddrop websockets rgba hsla multiplebgs bac
kgroundsize borderimage borderradius boxshadow textshadow opacity cssanimations csscolumns cssgradients cssref
```

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

```
soup = BeautifulSoup(html_data_2, 'html5lib')
print(soup.prettify()[:500])
```

```
<!DOCTYPE html>
<!-- saved from url=(0105)https://web.archive.org/web/20200814131437/https://www.macrotrends.net/stocks/charts/
<html class="js flexbox canvas canvastext webgl no-touch geolocation postmessage websqldatabase indexeddb hashc
```

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

```
import pandas as pd
from bs4 import BeautifulSoup

# Initialize an empty DataFrame
gme_revenue = pd.DataFrame(columns=["Date", "Revenue"])

# Iterate through each table found in the soup object
for table in soup.find_all('table'):
    # Check if the table header starts with "GameStop Quarterly Revenue"
    header = table.find('th')
    if header and header.get_text().startswith("GameStop Quarterly Revenue"):
        tbody = table.find("tbody")
        if tbody:
            rows = tbody.find_all("tr")
            data = []
            for row in rows:
                cols = row.find_all("td")
                if len(cols) == 2:
                    Date = cols[0].get_text(strip=True)
                    Revenue = cols[1].get_text(strip=True).replace("$", "").replace(",", "")
                    data.append({"Date": Date, "Revenue": Revenue})
            # Convert list of dictionaries to DataFrame and concatenate
            gme_revenue = pd.concat([gme_revenue, pd.DataFrame(data)], ignore_index=True)
```

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

```
print(gme_revenue.head())
```

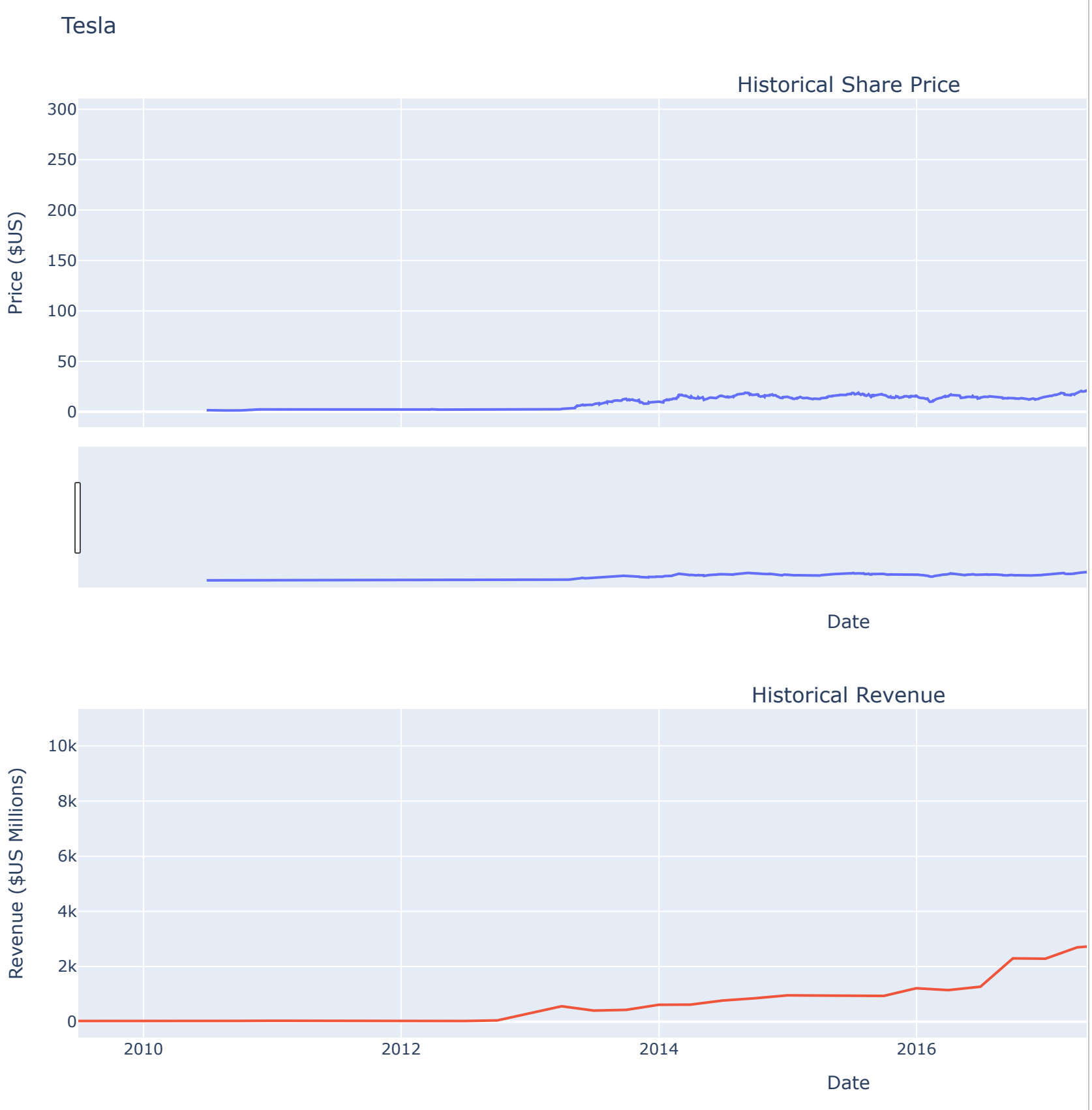
```
         Date Revenue
0  2020-04-30    1021
1  2020-01-31    2194
2  2019-10-31    1439
3  2019-07-31    1286
4  2019-04-30    1548
```

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

```
make_graph(tesla_data, tesla_revenue, 'Tesla')
```



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

```
make_graph(gme_data, gme_revenue, 'GameStop')
```


```
print("Stock Data Columns:", gme_data.columns)
print("Revenue Data Columns:", gme_revenue.columns)
```

```
Stock Data Columns: Index(['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Dividends',
       'Stock Splits'],
      dtype='object')
Revenue Data Columns: Index(['Date', 'Revenue'], dtype='object')
```

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.