https://github.com/brendensong/IBM-Data-Science-Professional-Certificate/wiki/4.1.Python-Basics

General Information

Welcome to this Python Project!

This mini-course is intended for you to demonstrate foundational Python skills for working with data. The completion of this course involves working on a hands-on project where you will develop a simple dashboard using Python.

This course is part of the IBM Data Science Professional Certificate and the IBM Data Analytics Professional Certificate.

PRE-REQUISITE: Python Basics for Data Science course from IBM is a pre-requisite for this project course. Please ensure that before taking this course you have either completed the Python Basics for Data Science course from IBM or have equivalent proficiency in working with Python and data.

NOTE: This course is not intended to teach you Python and does not have too much instructional content. It is intended for you to apply prior Python knowledge.

Syllabus

Bookmark this page

Course Syllabus

Module 1 - Intro to Web Scraping (optional)

- Intro to Web Scraping
- HTML for Web Scraping
- Web Scraping
- Hands-on Lab: Intro to Web Scraping Using BeautifulSoup

Module 2 - Final Project: Analyzing Stock Performance and Building a Dashboard

- Project Overview
- Stock Shares
- Hands-on Lab: Extracting Stock Data Using a Python Library
- Quiz: Extracting Stock Data Using a Python Library
- Extracting Stock Data Using Web Scraping
- Quiz: Extracting Stock Data Using Web Scraping
- Tesla and GameStop Analytical Dashboard
- Hands-on Lab: Create IBM Cloud account and Watson Studio instance
- Jupyter Notebook to Complete Your final Project
- Add notebook to Watson Studio
- Analyzing Historical Stock/Revenue Data and Building a Dashboard
- Hands-on Lab: Share your notebook from Watson Studio

Peer assignment

Grading Scheme

Bookmarked

GRADING SCHEME

This section contains information for those earning a certificate. Those auditing the course can skip this section and click next.

1. This course contains 2 Graded Quizzes and 1 Final Assignment. Your total grade at 100% is weighted as follows:
   - Each of the 2 Graded Quizzes carries an equal weight totaling 40% of your total grade.
   - Final Assignment carries a weight of 60% of your total grade.
2. The minimum passing mark for the **course** is 70%.
3. Permitted attempts are per **question**:
   - One attempt - for True/False questions
   - Two attempts - for any question other than True/False
4. There are no penalties for incorrect attempts.
5. Clicking the "**Final Check**" button when it appears, means your submission is **FINAL**. You will **NOT** be able to resubmit your answer for that question again.
6. Check your grades in the course at any time by clicking on the "Progress" tab.

Learning Objectives

Bookmark this page

LEARNING OBJECTIVES

In this course you will learn about:

- Demonstrate your skills for working with Python and Data
- Create a dashboard that shows key performance indicators from a specific data set

Course Introduction

Bookmarked

Welcome to this Python Project!

This mini-course is intended to be a follow-on to the Python for Data Science, AI and Development course for you to demonstrate basic Python skills that you have acquired in that course.

In this mini-course you will be assuming the role of a Data Scientist / Data Analyst. In this role you will be given a scenario and data to begin your Python project. During this process you will perform specific tasks such as extracting data, web scraping, visualizing data, and creating a dashboard. You will then submit your Python notebook and screenshots for your peers to review and evaluate your work.

To be successful in completing this project, you should have already completed the Python for Data Science, AI and Development course or have equivalent skills for working with Data and Python.

Intro to Web Scraping

This section of the course is intended for learners who have taken the old version of the Python for Data Science, AI & Development which did not include Web Scraping.

HTML for Webscraping

In this video we will review Hypertext Markup Language or HTML for Webscraping. Lots of useful data is available on web pages, such as real estate prices and solutions to coding questions. The website Wikipedia is a repository of the world's information.

If you have an understanding of HTML, you can use Python to extract this information. In this video, you will:

review the HTML of a basic web page; understand the Composition of an HTML Tag; understand HTML Trees;

and understand HTML Tables. Let's say you were asked to find the name and salary of players in

a National Basketball League from the following web page. The web page is comprised of HTML. It consists of text surrounded by a series of blue text elements enclosed in angle brackets called tags. The tags tells the browser how to display the content. The data we require is in this text.

The first portion contains the "DOCTYPE html" which declares this document is an HTML document. element is the root element of an HTML page, and element contains meta information about the HTML page.

Next, we have the body, this is what's displayed on the web page. This is usually the data we are interested in, we see the elements with an "h3", this means type 3 heading, makes the text larger and bold.

These tags have the names of the players, notice the data is enclosed in the elements. It starts with a h3 in brackets and ends in a slash h3 in brackets. There is also a different tag "p", this means paragraph, each p tag contains a player's salary. Let's take a closer look at the composition of an HTML tag. Here is an example of an HTML Anchor tag. It will display IBM and when you click it, it will send you to IBM.com.

We have the tag name, in this case "a". This tag defines a hyperlink, which is used to link from one page to another. It's helpful to think of each tag name as a class in Python and each individual tag as an instance.

We have the opening or start tag and we have the end tag. This has the tag name preceded by a slash.

These tags contain the content, in this case what's displayed on the web page. We have the attribute, this is composed of the Attribute Name and Attribute Value.

In this case it the url to the destination web page. Real web pages are more complex, depending on your browser you can select the HTML element, then click Inspect. The result will give you the ability to inspect the HTML. There is also other types of content such as CSS and JavaScript that we will not go over in this course.

The actual element is shown here. Each HTML document can actually be referred to as a document tree. Let's go over a simple example. Tags may contain strings and other tags. These elements are the tag's children.

We can represent this as a family tree. Each nested tag is a level in the tree. The tag HTML tag contains the head and body tag.

The Head and body tag are the descendants of the html tag. In particular they are the children of the HTML tag.

HTML tag is their parent. The head and body tag are siblings as they are on the same level. Title tag is the child of the head tag and its parent is the head tag.

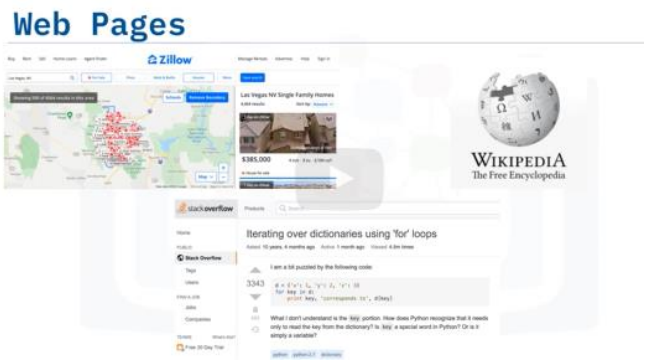
The title tag is a descendant of the HTML tag but not its child. The heading and paragraph tags are the children of the body tag; and as they are all children of the body tag they are siblings of each other.

The bold tag is a child of the heading tag, the content of the tag is also part of the tree but this can get unwieldy to draw.

Next, let's review HTML tables. To define an HTML table we have the table tag. tags, each defines a table cell.

For the first row first cell we have; for the first row second cell we have; and so on.

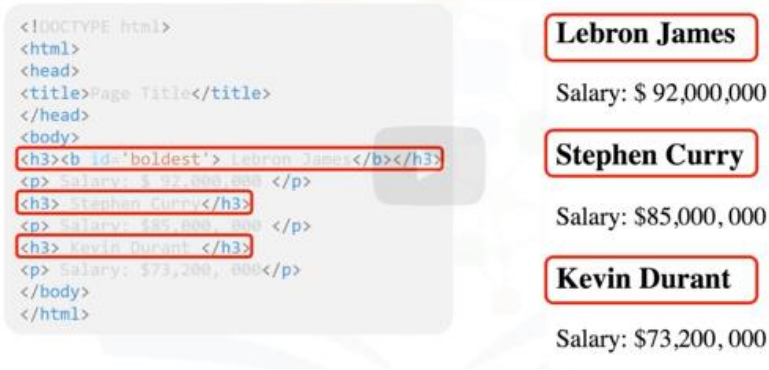
For the second row we have; and for the second row first cell we have; for the second row second cell we have; and so on. We now have some basic knowledge of HTML.



HTML Composition

The first portion contains the "DOCTYPE html" which declares this document is an HTML document.
element is the root element of an HTML page, and element contains meta information about the HTML page.

HTML Elements



Next, we have the body, this is what's displayed on the web page. This is usually the data we are interested in, we see the elements with an "h3", this means type 3 heading, makes the text larger and bold. These tags have the names of the players, notice the data is enclosed in the elements. It starts with a h3 in brackets and ends in a slash h3 in brackets. There is also a different tag "p", this means paragraph, each p tag contains a player's salary.

Composition of an HTML Tag

HTML Anchor Tag

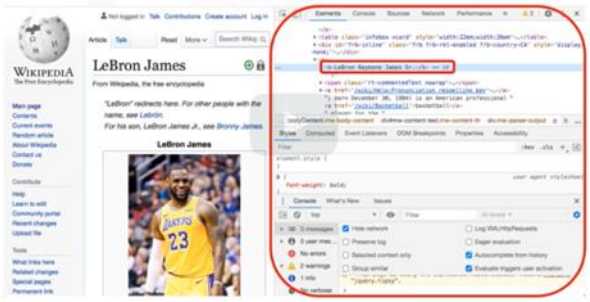
Hyperlink Tag

Opening and End Tags

Hyperlink Content

Attributes

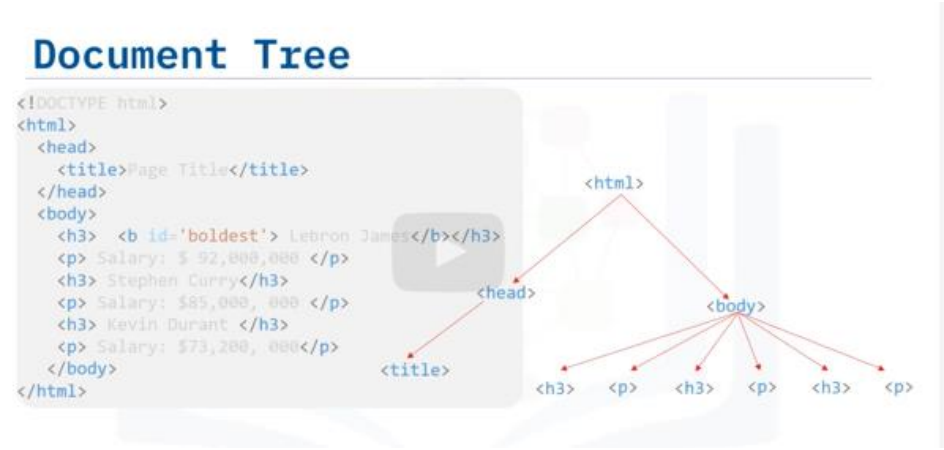
Inspect HTML



HTML Trees

Document Tree

Title tag is the child of the head tag and its parent is the head tag. The title tag is a descendant of the HTML tag but not its child.



The heading and paragraph tags are the children of the body tag; and as they are all children of the body tag they are siblings of each other. The bold tag is a child of the heading tag, the content of the tag is also part of the tree but this can get unwieldy to draw.

HTML Tables

HTML Tables

```
<table >
  <tr>
    <td>Pizza Place</td>
    <td>Orders</td>
    <td>Slices </td>
  </tr>
  <tr>
    <td>Domino Pizza</td>
    <td>10</td>
    <td>100</td>
  </tr>
  <tr>
    <td>Little Caesars</td>
    <td>12</td>
    <td >144 </td>
  </tr>
</table>
```

Pizza Place	Orders	Slices
Domino's Pizza	10	100
Little Caesars	12	144

HTML Tables

```
<table >
  <tr>
    <td>Pizza Place</td>
    <td>Orders</td>
    <td>Slices </td>
  </tr>
  <tr>
    <td>Domino Pizza</td>
    <td>10</td>
    <td>100</td>
  </tr>
  <tr>
    <td>Little Caesars</td>
    <td>12</td>
    <td >144 </td>
  </tr>
</table>
```

Pizza Place	Orders	Slices
Domino's Pizza	10	100
Little Caesars	12	144

To define an HTML table we have the table tag. tags, each defines a table cell. For the first row first cell we have; for the first row second cell we have; and so on. For the second row we have; and for the second row first cell we have; for the second row second cell we have; and so on.

Webscraping

In this video we will cover Webscraping. After watching this video you will be able to: define web scraping; understand the role of BeautifulSoup Objects; apply the find_all method;

and webscrape a website. What would you do if you wanted to analyze hundreds of points of data to find the best players of a sports team?

Would you start manually copying and pasting information from different websites into a

spreadsheet? Spending hours trying to find the right data, and eventually giving up because the task was to overwhelming?

That's where webscraping can help. Webscraping is a process that can be used to automatically extract information from a website, and can easily be accomplished within a matter of minutes and not hours. To get started we just need a little Python code and the help of two modules named Requests and Beautiful Soup.

Let's say you were asked to find the name and salary of players in a National Basketball

League, from the following webpage. We import BeautifulSoup. We can store the webpage HTML as a string in the variable HTML.

To parse a document, pass it into the BeautifulSoup constructor. We get the Beautiful Soup object , soup, which represents the document as a nested data structure.

BeautifulSoup represents HTML as a set of Tree like objects with methods used to parse

the HTML. We will review the BeautifulSoup object Using the BeautifulSoup object, soup, we created The tag object corresponds to an HTML tag in the original document. For example,

the tag "title." Consider the tag . If there is more than one tag with the same name, the first

element with that tag is selected. In this case with Lebron James, we see the name is Enclosed in the bold attribute "b". To extract it, use the Tree representation.

Let's use the Tree representation. The variable tag-object is located here. We can access the child of the tag or navigate down the branch as follows: You can navigate up the tree by using the parent attribute.

The variable tag child is located here. We can access the parent. This is the original tag object. We can find the sibling of "tag object." We simply use the next sibling attribute.

We can find the sibling of sibling one. We simply use the next sibling attribute.

Consider the tag child object. You can access the attribute name and value as a key value pair in a dictionary as follows. You can return the content as a Navigable string, this is like a Python string that supports BeautifulSoup functionality.

Let's review the method find_all. This is a filter, you can use filters to filter based on a tag's name, it's attributes, the text of a string, or on some combination of these.

Consider the list of pizza places. Like before, create a BeautifulSoup object. But this time, name it table.

The find_all () method looks through a tag's descendants and retrieves all descendants

The result is a Python iterable just like a list, This corresponds to each row in the list-

including the table header. Each element is a tag object. Consider the first row.

For example, we can extract the first table cell. We can also iterate through each table cell.

First, we iterate through the list "table rows," via the variable row. Each element corresponds to a row in the table. We can apply the method find all to find all the table cells,

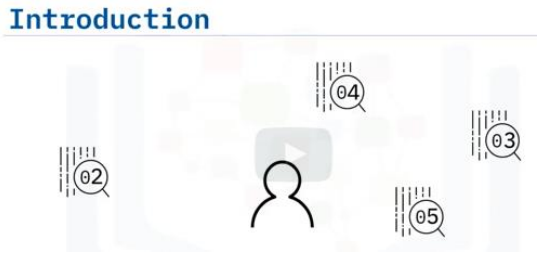
then we can iterate through the variable cells for each row. For each iteration,

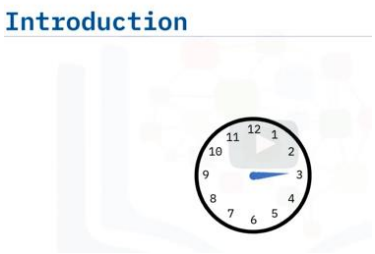
the variable cell corresponds to an element in the table for that particular row.

We continue to iterate through each element and repeat the process for each row.

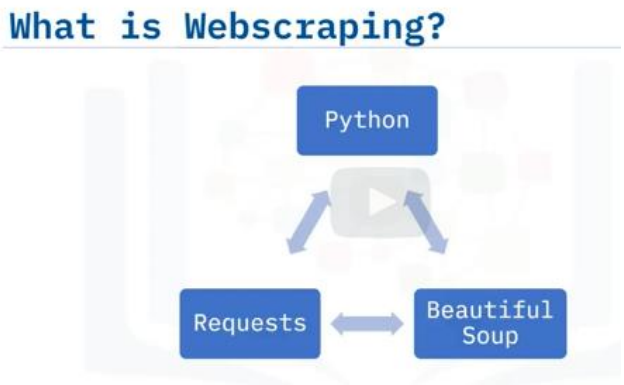
Let's see how to apply BeautifulSoup to a webpage. To scrape a webpage we also need the Requests library. The first step is to import the modules that are needed. Use the get method from the requests library to download the webpage. The input is the URL. Use the text attribute to get the text and assign it to the variable page. Then, create a BeautifulSoup object 'soup' from the variable page. It will allow you to parse through the HTML page.

You can now scrape the Page. Check out the labs for more.





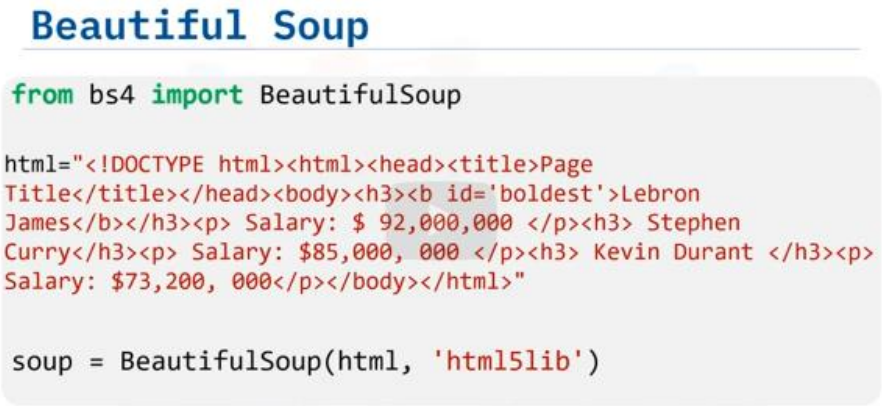
What would you do if you wanted to analyze hundreds of points of data to find the best players of a sports team?



Webscraping is a process that can be used to automatically extract information from a website, and can easily be accomplished within a matter of minutes and not hours. To get started we just need a little Python code and the help of two modules named Requests and Beautiful Soup.

Problem: Let's say you were asked to find the name and salary of players in a National Basketball League, from the following webpage.

Solution:



1. We import BeautifulSoup.

2. We can store the webpage HTML as a string in the variable HTML.
3. To parse a document, pass it into the BeautifulSoup constructor.
4. We get the Beautiful Soup object , soup, which represents the document as a nested data structure.

Beautiful Soup Objects

BeautifulSoup represents HTML as a set of Tree like objects with methods used to parse the HTML.

Using the BeautifulSoup object, soup, we created The tag object corresponds to an HTML tag in the original document. For example, the tag "title."

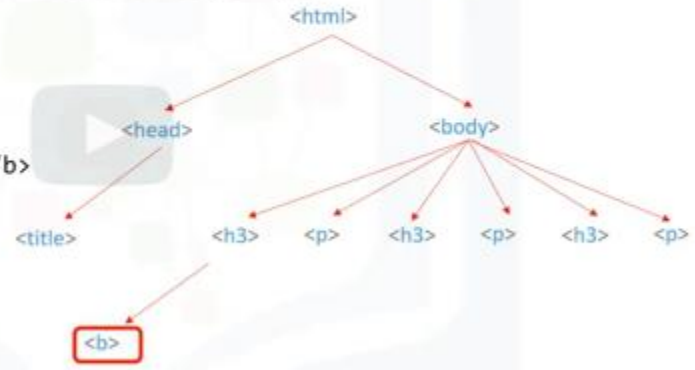
Consider the tag <h3>. If there is more than one tag with the same name, the first element with that tag is selected. In this case with Lebron James, we see the name is Enclosed in the bold attribute "b". To extract it, use the Tree representation.

HTML Tree

```
tag_object: <h3><b id="boldest">Lebron James</b></h3>

tag_child =tag_object.b
tag_child:

<b id="boldest">Lebron James</b>
```



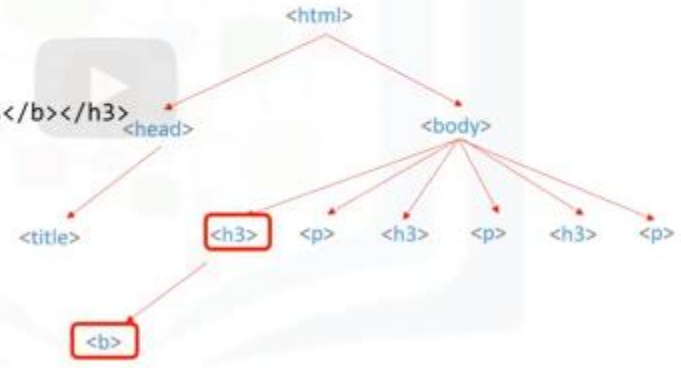
1. The variable tag-object is located here.
2. We can access the child of the tag or navigate down the branch as follows:
3. You can navigate up the tree by using the parent attribute.
4. The variable tag child is located here.

Parent attribute

```
tag_child:<b id="boldest">Lebron James</b>

parent_tag=tag_child.parent
parent_tag:

<h3><b id="boldest">Lebron James</b></h3>
```



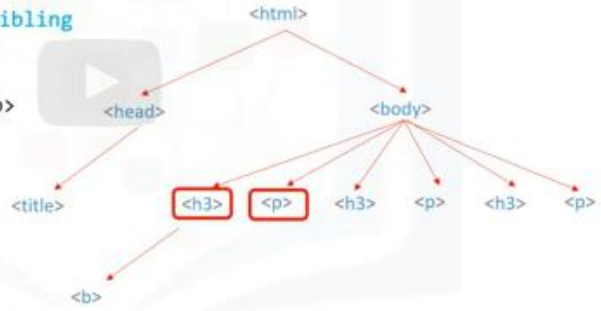
5. We can access the parent. This is the original tag object.

Next-sibling attribute

```
tag_object: <h3><b id="boldest">Lebron James</b></h3>

sibling_1= tag_object.next_sibling
sibling_1:

<p> Salary: $ 92,000,000 </p>
```



Next-sibling attribute

```
sibling_1: <p> Salary: $ 92,000,000 </p>

sibling_2=sibling_1.next_sibling
sibling_2:

<h3> Stephen Curry</h3>
```

1. We simply use the next sibling attribute.
2. We can find the sibling of sibling one.
3. We simply use the next sibling attribute.
4. Consider the tag child object.
5. You can access the attribute name and value as a key value pair in a dictionary as follows.

Navigable string

```
tag_child:<b id="boldest">Lebron James</b>

tag_child.attrs:

{'id': 'boldest'}

tag_child.string:

'Lebron James'
```

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

1. Consider the tag child object.
2. You can access the attribute name and value as a key value pair in a dictionary as follows.
3. You can return the content as a Navigable string, this is like a Python string that supports BeautifulSoup functionality.

find_all

Pizza place list

```
<table >
  <tr>
    <td>Pizza Place</td>
    <td>Orders</td>
    <td>Slices </td>
  </tr>
  <tr>
    <td>Domino Pizza</td>
    <td>10</td>
    <td>100</td>
  </tr>
  <tr>
    <td>Little Caesars</td>
    <td>12</td>
    <td >144 </td>
  </tr>
</table>
```

Pizza Place	Orders	Slices
Domino's Pizza	10	100
Little Caesars	12	144

Beautiful Soup object

```
from bs4 import BeautifulSoup

html= "<table><tr><td>Pizza Place</td><td>Orders</td>
<td>Slices </td></tr><tr><td>Domino'sPizza</td>
<td>10</td><td>100</td></tr><tr><td>Little
Caesars</td><td>12</td><td>144</td></table>"

table = BeautifulSoup(html, 'html5lib')
```

Python iterable

```
table_row=table.find_all(name='tr')


table_row:
```

Pizza Place	Orders	Slices
Domino's Pizza	10	100
Little Caesars	12	144

```
[<tr><td>Pizza Place</td><td>Orders</td><td>Slices</td></tr>,
 <tr><td>Domino's Pizza</td><td>10</td><td>100</td></tr>,
 <tr><td>Little Caesars</td><td>12</td><td>144</td></tr>,
```

Tag object

```
first_row =table_row[0]
first_row:

<tr><td>Pizza Place</td><td>Orders</td> <td>Slices </td></tr>


first_row.td :


<td>Pizza Place</td>
```

Each element is a tag object. Consider the first row.

For example, we can extract the first table cell.

Variable row

Pizza Place	Orders	Slices
Domino's Pizza	10	100
Little Caesars	12	144
Papa John's	15	166

```
for i,row in enumerate(table_rows):

   print("row", i)
   cells=row.find_all("td")

   for j,cell in enumerate(cells):

       print("column", j , "cell", cell)
```

We can also iterate through each table cell. First, we iterate through the list "table rows," via the variable row.

Elements

Pizza Place	Orders	Slices
Domino's Pizza	10	100
Little Caesars	12	144
Papa John's	15	166

```
for i,row in enumerate(table_rows):
    print("row", i)
    cells=row.find_all("td")
    for j,cell in enumerate(cells):
        print("column", j , "cell", cell)
```

Each element corresponds to a row in the table. We can apply the method find all to find all the table cells, then we can iterate through the variable cells for each row. For each iteration,
the variable cell corresponds to an element in the table for that particular row. We continue to iterate through each element and repeat the process for each row.

A Webpage Example

Overview Python Program

```
import requests
from bs4 import BeautifulSoup

page = requests.get("http://EnterWebsiteURL...").text

#Creates a BeautifulSoup object
soup = BeautifulSoup(page, "html.parser")

# Pulls all instances of <a> tag
artists = soup.find_all('a')

#Clears data of all tags
for artist in artists:
    names = artist.contents[0]
    fullLink = artist.get('href')
    print(names)
    print(fullLink)
```

1. Let's see how to apply BeautifulSoup to a webpage.
2. To scrape a webpage we also need the Requests library.

Overview Python Program

```
import requests
from bs4 import BeautifulSoup

page = requests.get("http://EnterWebsiteURL...").text

#Creates a BeautifulSoup object
soup = BeautifulSoup(page, "html.parser")

# Pulls all instances of <a> tag
artists = soup.find_all('a')

#Clears data of all tags
for artist in artists:
    names = artist.contents[0]
    fullLink = artist.get('href')
    print(names)
    print(fullLink)
```

1. The first step is to import the modules that are needed.

Overview Python Program

```
import requests
from bs4 import BeautifulSoup

page = requests.get("http://EnterWebsiteURL...").text
#Creates a BeautifulSoup object
soup = BeautifulSoup(page, "html.parser")
# Pulls all instances of <a> tag
artists = soup.find_all('a')
#Clears data of all tags
for artist in artists:
    names = artist.contents[0]
    fullLink = artist.get('href')
    print(names)
    print(fullLink)
```

2. Use the get method from the requests library to download the webpage. The input is the URL. Use the text attribute to get the text and assign it to the variable page.

Overview Python Program

```
import requests
from bs4 import BeautifulSoup

page = requests.get("http://EnterWebsiteURL...").text
#Creates a BeautifulSoup object
soup = BeautifulSoup(page, "html.parser")
# Pulls all instances of <a> tag
artists = soup.find_all('a')
#Clears data of all tags
for artist in artists:
    names = artist.contents[0]
    fullLink = artist.get('href')
    print(names)
    print(fullLink)
```

3. Then, create a BeautifulSoup object 'soup' from the variable page. It will allow you to parse through the HTML page. It will allow you to parse through the HTML page. You can now scrape the Page.

5.1.Crowdsourcing Short squeeze Dashboard

Seongjoo Brenden Song edited this page on May 25, 2021 · 4 revisions

Course Introduction

Welcome to this Python Project!

This mini-course is intended to be a follow-on to the Python for Data Science, AI and Development course for you to demonstrate basic Python skills that you have acquired in that course.

In this mini-course you will be assuming the role of a Data Scientist / Data Analyst. In this role you will be given a scenario and data to begin your Python project. During this process you will perform specific tasks such as extracting data, web scraping, visualizing data, and creating a dashboard. You will then submit your Python notebook and screenshots for your peers to review and evaluate your work.

To be successful in completing this project, you should have already completed the Python for Data Science, AI and Development course or have equivalent skills for working with Data and Python.

- Extracting Stock Data Using a Python Library
- Extracting Stock Data Using Web Scraping

https://www.coursera.org/professional-certificates/ibm-data-science

For Hands On: Intro to Webscraping Using Beautiful Soap

Pls refer to pp. 469-509 of notes Module 5 Python Basics for Data Science or file: IBM DS5 Python Basics for Data Science pp. 469-509.

Project Overview

For this project, you will assume the role of a Data Scientist / Data Analyst working for a new startup investment firm that helps customers invest their money in stocks. Your job is to extract financial data like historical share price and quarterly revenue reportings from various sources using Python libraries and webscraping on popular stocks. After collecting this data you will visualize it in a dashboard to identify patterns or trends. The stocks we will work with are Tesla, Amazon, AMD, and GameStop.

Dashboard Analytics Displayed

A dashboard often provides a view of key performance indicators in a clear way. Analyzing a data set and extracting key performance indicators will be practiced. Prompts will be used to support learning in accessing and displaying data in dashboards. Learning how to display key performance indicators on a dashboard will be included in this assignment. We will be using Plotly in this course for data visualization and is not a requirement to take this course.

Watson Studio

In the Python for Data Science, AI and Development course you utilized Skills Network Labs for hands-on labs.

For this project you will use Skills Network Labs and Watson Studio. Skills Network Labs is a sandbox environment for learning and completing labs in courses. Whereas Watson Studio, a component of IBM Cloud Pak for Data, is a suite of tools and a collaborative environment for data scientists, data analysts, AI and machine learning engineers and domain experts to develop and deploy your projects.

Review criteria

There are two hands-on labs on Extracting Stock Data and one assignment to complete. You will be judged by completing two quizzes and one peer review assignment. The quizzes will test you based on the output of the hands-on labs. In the peer review assignment you will share and take screen shots of the outcomes of your assignment.

Reading: Stock Shares

A company's stock share is a piece of the company; more precisely:

A stock (also known as equity) is a security that represents the ownership of a fraction of a corporation. This entitles the owner of the stock to a proportion of the corporation's assets and profits equal to how much stock they own. Units of stock are called "shares." [1]

An investor can buy a stock and sell it later. If the stock price increases, the investor profits, If it decreases, the investor with incur a loss. Determining the stock price is complex; it depends on the number of outstanding shares, the size of the company's future profits, and much more. People trade stocks throughout the day. The stock ticker is a report of the price of a certain stock, updated continuously throughout the trading session by the various stock market exchanges. In this lab, you will use the y-finance API to obtain the stock ticker and extract information about the stock. You will then be asked questions about your results.

Extracting Stock Data Using a Python Library

A company's stock share is a piece of the company more precisely:

A stock (also known as equity) is a security that represents the ownership of a fraction of a corporation. This entitles the owner of the stock to a proportion of the corporation's assets and profits equal to how much stock they own. Units of stock are called "shares." [1]

An investor can buy a stock and sell it later. If the stock price increases, the investor profits, If it decreases,the investor with incur a loss. Determining the stock price is complex; it depends on the number of outstanding shares, the size of the company's future profits, and much more. People trade stocks throughout the day the stock ticker is a report of the price of a certain stock, updated continuously throughout the trading session by the various stock market exchanges.

You are a data scientist working for a hedge fund; it's your job to determine any suspicious stock activity. In this lab you will extract stock data using a Python library. We will use the yfinance library, it allows us to extract data for stocks returning data in a pandas dataframe. You will use the lab to extract.

Table of Contents

In [1]:
```
!pip install yfinance
#!pip install pandas
```

```
Collecting yfinance
  Downloading https://files.pythonhosted.org/packages/a7/ee/315752b9ef281ba83c62aa7ec2e2074f85223da6e7e74efb4d3e11c0f510/yfinance-0.1.59.tar.gz
Requirement already satisfied: pandas>=0.24 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from yfinance) (1.1.5)
Requirement already satisfied: numpy>=1.15 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from yfinance) (1.19.5)
Requirement already satisfied: requests>=2.20 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from yfinance) (2.25.1)
Collecting multitasking>=0.0.7 (from yfinance)
  Downloading https://files.pythonhosted.org/packages/69/e7/e9f1661c28f7b87abfa08cb0e8f51dad2240a9f4f741f02ea839835e6d18/multitasking-0.0.9.tar.gz
Requirement already satisfied: lxml>=4.5.1 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from yfinance) (4.6.3)
Requirement already satisfied: pytz>=2017.2 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from pandas>=0.24->yfinance) (2021.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from pandas>=0.24->yfinance)
(2.8.1)
Requirement already satisfied: idna<3,>=2.5 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from requests>=2.20->yfinance) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from requests>=2.20->yfinance)
(1.26.4)
Requirement already satisfied: certifi>=2017.4.17 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from requests>=2.20->yfinance) (202
0.12.5)
Requirement already satisfied: chardet<5,>=3.0.2 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from requests>=2.20->yfinance) (4.0.
0)
Requirement already satisfied: six>=1.5 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from python-dateutil>=2.7.3->pandas>=0.24->yf
inance) (1.16.0)
Building wheels for collected packages: yfinance, multitasking
  Building wheel for yfinance (setup.py) ... done
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/f8/2a/0f/4b5a86e1d52e451757eb6bc17fd899629f0925c777741b6d04
  Building wheel for multitasking (setup.py) ... done
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/37/fa/73/d492849e319038eb4d986f5152e4b19ffb1bc0639da84d2677
Successfully built yfinance multitasking
Installing collected packages: multitasking, yfinance
Successfully installed multitasking-0.0.9 yfinance-0.1.59
```

In [2]:
```
import yfinance as yf
import pandas as pd
```

Using the yfinance Library to Extract Stock Data

Using the Ticker module we can create an object that will allow us to access functions to extract data. To do this we need to provide the ticker symbol for the stock, here the company is Apple and the ticker symbol is AAPL.

In [3]:
```
apple = yf.Ticker("AAPL")
```

Now we can access functions and variables to extract the type of data we need. You can view them and what they represent here https://aroussi.com/post/python-yahoo-finance.

Stock Info

Using the attribute info we can extract information about the stock as a Python dictionary.

In [4]:
```
apple_info=apple.info
apple_info
```

Out[4]:

```
{'zip': '95014',
 'sector': 'Technology',
 'fullTimeEmployees': 100000,
 'longBusinessSummary': 'Apple Inc. designs, manufactures, and markets smartphones, personal computers, tablets, wearables, and accessories worldwide. I
t also sells various related services. The company offers iPhone, a line of smartphones; Mac, a line of personal computers; iPad, a line of multi-purpos
e tablets; and wearables, home, and accessories comprising AirPods, Apple TV, Apple Watch, Beats products, HomePod, iPod touch, and other Apple-branded
and third-party accessories. It also provides AppleCare support services; cloud services store services; and operates various platforms, including the A
pp Store, that allow customers to discover and download applications and digital content, such as books, music, video, games, and podcasts. In addition,
the company offers various services, such as Apple Arcade, a game subscription service; Apple Music, which offers users a curated listening experience w
ith on-demand radio stations; Apple News+, a subscription news and magazine service; Apple TV+, which offers exclusive original content; Apple Card, a c
o-branded credit card; and Apple Pay, a cashless payment service, as well as licenses its intellectual property. The company serves consumers, and small
and mid-sized businesses; and the education, enterprise, and government markets. It sells and delivers third-party applications for its products through
the App Store. The company also sells its products through its retail and online stores, and direct sales force; and third-party cellular network carrie
rs, wholesalers, retailers, and resellers. Apple Inc. was founded in 1977 and is headquartered in Cupertino, California.',
 'city': 'Cupertino',
 'phone': '408-996-1010',
 'state': 'CA',
 'country': 'United States',
  'phone': '408-996-1010',
  'state': 'CA',
  'country': 'United States',
  'companyOfficers': [],
  'website': 'http://www.apple.com',
  'maxAge': 1,
  'address1': 'One Apple Park Way',
  'industry': 'Consumer Electronics',
  'previousClose': 125.43,
  'regularMarketOpen': 126.01,
  'twoHundredDayAverage': 127.30792,
  'trailingAnnualDividendYield': 0.006537511,
  'payoutRatio': 0.1834,
 'fiveYearAverageReturn': None,
 'regularMarketPrice': 127.1,
 'logo_url': 'https://logo.clearbit.com/apple.com'}
```


We can get the `'country'` using the key country

In [5]:
```
apple_info['country']
```

Out[5]:
```
'United States'
```

Extracting Share Price

A share is the single smallest part of a company's stock that you can buy, the prices of these shares fluctuate over time. Using the history() method we can get the share price of the stock over a certain period of time. Using the period parameter we can set how far back from the present to get data. The options for period are 1 day (1d), 5d, 1 month (1mo) , 3mo, 6mo, 1 year (1y), 2y, 5y, 10y, ytd, and max.

In [6]:
```
apple_share_price_data = apple.history(period="max")
```

The format that the data is returned in is a Pandas DataFrame. With the Date as the index the share Open, High, Low, Close, Volume, and Stock Splits are given for each day.

In [7]:
```
apple_share_price_data.head()
```

Out[7]:

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
1980-12-12	0.100751	0.101189	0.100751	0.100751	469033600	0.0	0.0
1980-12-15	0.095933	0.095933	0.095495	0.095495	175884800	0.0	0.0
1980-12-16	0.088923	0.088923	0.088485	0.088485	105728000	0.0	0.0
1980-12-17	0.090676	0.091114	0.090676	0.090676	86441600	0.0	0.0
1980-12-18	0.093304	0.093742	0.093304	0.093304	73449600	0.0	0.0

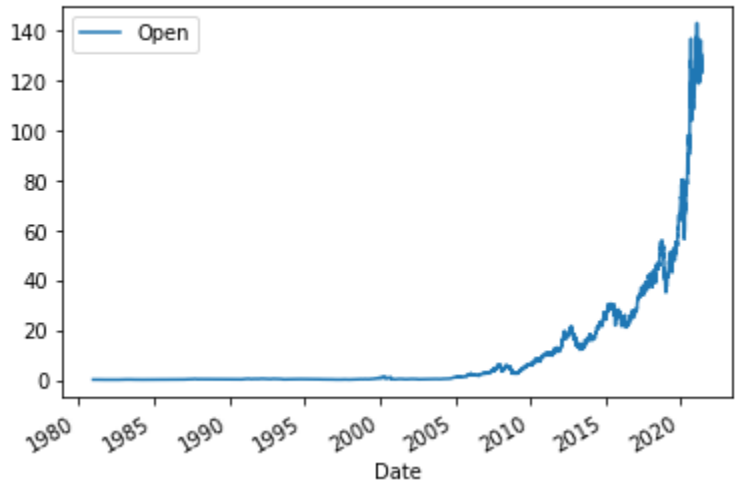
We can reset the index of the DataFrame with the reset_index function. We also set the inplace paramter to True so the change takes place to the DataFrame itself.

In [8]:
```
apple_share_price_data.reset_index(inplace=True)
```

We can plot the `Open` price against the `Date` :

In [11]:
```
apple_share_price_data.plot(x="Date", y="Open")
```

Out[11]:
```
<AxesSubplot:xlabel='Date'>
```



Extracting Dividends

Dividends are the distribution of a companys profits to shareholders. In this case they are defined as an amount of money returned per share an investor owns. Using the variable dividends we can get a dataframe of the data. The period of the data is given by the period defined in the 'history` function.

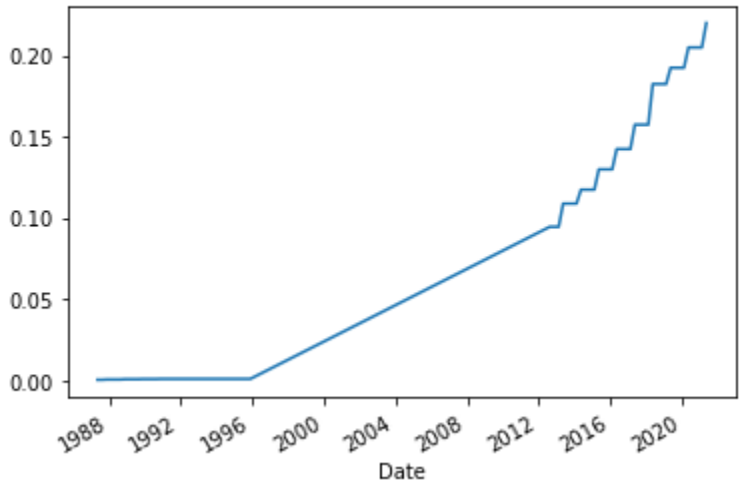
In [12]:
```
apple.dividends
```

Out[12]:
```
Date
1987-05-11    0.000536
1987-08-10    0.000536
1987-11-17    0.000714
1988-02-12    0.000714
1988-05-16    0.000714
                ...
2020-05-08    0.205000
2020-08-07    0.205000
2020-11-06    0.205000
2021-02-05    0.205000
2021-05-07    0.220000
Name: Dividends, Length: 71, dtype: float64
```

We can plot the dividends overtime:

In [13]:
```
apple.dividends.plot()
```

Out[13]:
```
<AxesSubplot:xlabel='Date'>
```



Exercise

Now using the Ticker module create an object for AMD (Advanced Micro Devices) with the ticker symbol is AMD called; name the object amd.

In [14]:
```
amd = yf.Ticker("AMD")
```

Question 1 Use the key `'country'` to find the country the stock belongs to, remember it as it will be a quiz question.

In [17]:
```
amd_info = amd.info
amd_info['country']
```

Out[17]:
```
'United States'
```

Question 2 Use the key `'sector'` to find the sector the stock belongs to, remember it as it will be a quiz question.

In [18]:
```
amd_info['sector']
```

Out[18]:
```
'Technology'
```

Question 3 Find the max of the `Volume` column of AMD using the `history` function, set the `period` to max.

In [49]:
```
amd_share_price_data = amd.history(period="max")
column = amd_share_price_data["Volume"]
amd_max_volume = column.max()
index_max = column.idxmax()
print("The max of the Volume of AMD :", amd_max_volume, '(',index_max.date(),')')
```

```
The max of the Volume of AMD : 325058400 ( 2018-08-27 )
```

In [20]:
```
amd_share_price_data.head()
```

Out[20]:

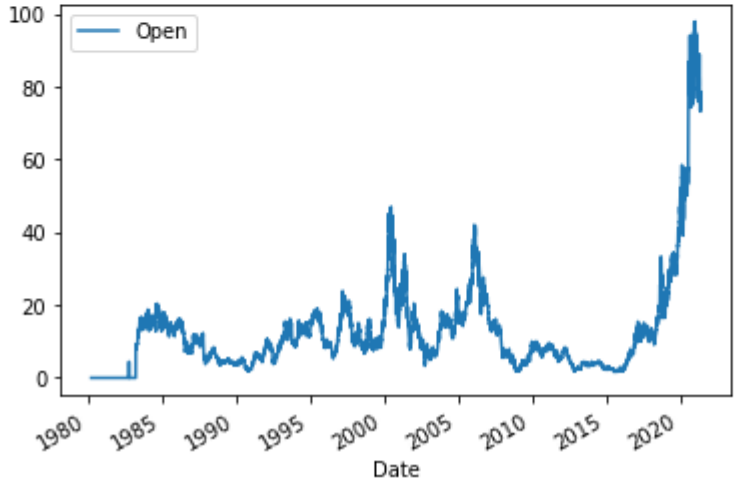
Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
1980-03-17	0.0	3.302083	3.125000	3.145833	219600	0	0.0
1980-03-18	0.0	3.125000	2.937500	3.031250	727200	0	0.0
1980-03-19	0.0	3.083333	3.020833	3.041667	295200	0	0.0
1980-03-20	0.0	3.062500	3.010417	3.010417	159600	0	0.0
1980-03-21	0.0	3.020833	2.906250	2.916667	130800	0	0.0

In [22]:

```
amd_share_price_data.reset_index(inplace=True)
amd_share_price_data.plot(x="Date", y="Open")
```

Out[22]:

```
<AxesSubplot:xlabel='Date'>
```



In [25]:
```
tesla = yf.Ticker("TSLA")
tesla_info = tesla.info
#tesla_info['country']
#tesla_info['sector']

tsla_share_price_date = tesla.history(period="max")
tsla_share_price_date.head()
```

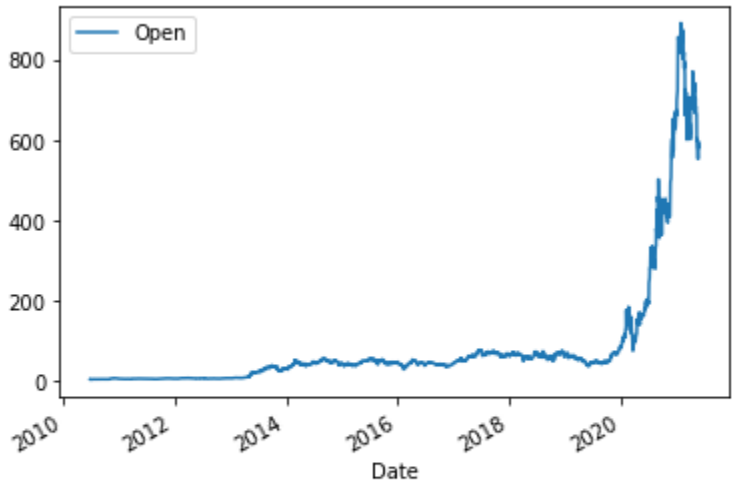
Out[25]:

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2010-06-29	3.800	5.000	3.508	4.778	93831500	0	0.0
2010-06-30	5.158	6.084	4.660	4.766	85935500	0	0.0
2010-07-01	5.000	5.184	4.054	4.392	41094000	0	0.0
2010-07-02	4.600	4.620	3.742	3.840	25699000	0	0.0
2010-07-06	4.000	4.000	3.166	3.222	34334500	0	0.0

In [26]:
```
tsla_share_price_date.reset_index(inplace=True)
tsla_share_price_date.plot(x="Date", y="Open")
```

Out[26]:
```
<AxesSubplot:xlabel='Date'>
```

Out[26]:
```
<AxesSubplot:xlabel='Date'>
```



In [32]:
```
tsla_share_price_2year = tesla.history(period='2y')
tsla_share_price_2year.reset_index(inplace=True)
tsla_share_price_2year.plot(x="Date", y="Open")
```

Graded Quiz: Extracting Stock Data Using a Python Library

Bookmarked

Graded Exam due Feb 9, 2022 09:07 +08 Completed

Question 1

1/1 point (graded)

From the lab exercise, in which country is AMD (Advanced Micro Devices) situated?

- China
- United States
- Canada

Question 2

1/1 point (graded)

In the lab exercise, to which sector does AMD (Advanced Micro Devices) belong?

- Agriculture
- Technology
- Electronics

Submit You have used 1 of 2 attempts

Question 3

1/1 point (graded)

Question 3 In the lab exercise, what is the Volume of AMD traded on the first day (first row)?

219600

219600

Submit You have used 2 of 2 attempts

Correct (1/1 point)

Extracting Stock Data Using a Web Scraping

Not all stock data is available via API in this assignment; you will use web-scraping to obtain financial data. You will be quizzed on your results.
Using beautiful soup we will extract historical share data from a web-page.

Table of Contents

Estimated Time Needed: **30 min**

In [1]:
```
#!pip install pandas
#!pip install requests
!pip install bs4
#!pip install plotly
```

```
Collecting bs4
  Downloading https://files.pythonhosted.org/packages/10/ed/7e8b97591f6f456174139ec089c769f89a94a1a4025fe967691de971f314/bs4-0.0.1.tar.gz
Collecting beautifulsoup4 (from bs4)
  Downloading https://files.pythonhosted.org/packages/d1/41/e6495bd7d3781cee623ce23ea6ac73282a373088fcd0ddc809a047b18eae/beautifulsoup4-4.9.3-py3-none-a
ny.whl (115kB)
    |████████████████████████████████| 122kB 18.5MB/s eta 0:00:01
Collecting soupsieve>1.2; python_version >= "3.0" (from beautifulsoup4->bs4)
  Downloading https://files.pythonhosted.org/packages/36/69/d82d04022f02733bf9a72bc3b96332d360c0c5307096d76f6bb7489f7e57/soupsieve-2.2.1-py3-none-any.wh
l
Building wheels for collected packages: bs4
  Building wheel for bs4 (setup.py) ... done
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/a0/b0/b2/4f80b9456b87abedbc0bf2d52235414c3467d8889be38dd472
Successfully built bs4
Installing collected packages: soupsieve, beautifulsoup4, bs4
Successfully installed beautifulsoup4-4.9.3 bs4-0.0.1 soupsieve-2.2.1
```

In [3]:
```
import pandas as pd
import requests
from bs4 import BeautifulSoup
```

Using Webscraping to Extract Stock Data

Use the requests library to download the webpage https://finance.yahoo.com/quote/AMZN/history?period1=1451606400&period2=1612137600&interval=1mo&filter=history&frequency=1mo&includeAdjustedClose=true. Save the text of the response as a variable named html_data.

In [4]:
```
url = "https://finance.yahoo.com/quote/AMZN/history?period1=1451606400&period2=1612137600&interval=1mo&filter=history&frequency=1mo&includeAdjustedClos
html_data = requests.get(url).text
```

Parse the html data using `beautiful_soup`.

In [5]:
```
soup = BeautifulSoup(html_data, "html5lib")
```

Question 1 what is the content of the title attribute:

In [6]:
```
soup.find_all('title')
```

Out[6]:
```
[<title>Amazon.com, Inc. (AMZN) Stock Historical Prices &amp;amp; Data - Yahoo Finance</title>]
```

Using beautiful soup extract the table with historical share prices and store it into a dataframe named amazon_data. The dataframe should have columns Date, Open, High, Low, Close, Adj Close, and Volume. Fill in each variable with the correct data from the list col.

Hint: Print the col list to see what data to use

In [15]:
```
amazon_data = pd.DataFrame(columns=["Date", "Open", "High", "Low", "Close", "Volume"])

for row in soup.find("tbody").find_all("tr"):
    col = row.find_all("td")
    date =col[0].text
    Open = col[1].text
    high = col[2].text
    low = col[3].text
    close = col[4].text
    adj_close = col[5].text
    volume = col[6].text

    amazon_data = amazon_data.append({"Date":date, "Open":Open, "High":high, "Low":low, "Close":close, "Adj Close":adj_close, "Volume":volume}, ignore_
```

Print out the first five rows of the amazon_data dataframe you created.

In [16]:
```
amazon_data.head()
```

Out[16]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	Jan 01, 2021	3,270.00	3,363.89	3,086.00	3,206.20	71,528,900	3,206.20
1	Dec 01, 2020	3,188.50	3,350.65	3,072.82	3,256.93	77,556,200	3,256.93
2	Nov 01, 2020	3,061.74	3,366.80	2,950.12	3,168.04	90,810,500	3,168.04
3	Oct 01, 2020	3,208.00	3,496.24	3,019.00	3,036.15	116,226,100	3,036.15
4	Sep 01, 2020	3,489.58	3,552.25	2,871.00	3,148.73	115,899,300	3,148.73

Question 2 What is the name of the columns of the dataframe

In [26]:
```
for col in amazon_data.columns:
    print(col)
```

```
Date
Open
High
Low
Close
Volume
Adj Close
```

Question 3 What is the `Open` of `Jun 01, 2019` of the dataframe?

In [23]:
```
index = amazon_data[amazon_data['Date']=='Jun 01, 2019'].index.values
amazon_data.loc[index, 'Open']
```

Out[23]:
```
19    1,760.01
Name: Open, dtype: object
```

5.2.Peer graded Assignment Analyzing Historical Stock Revenue Data and Building a Dashboard

Seongjoo Brenden Song edited this page on May 26, 2021 · 5 revisions

> Project Notebook

> IBM Cloud Pak for Data Link

```
!pip install yfinance
#!pip install pandas
#!pip install requests
!pip install bs4
#!pip install 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

#Define Graphing Function
def make_graph(stock_data, revenue_data, stock):
    fig = make_subplots(rows=2, cols=1, shared_xaxes=True, subplot_titles=("Historical Share Price", "Historical Rever
    fig.add_trace(go.Scatter(x=pd.to_datetime(stock_data.Date, infer_datetime_format=True), y=stock_data.Close.astype(
    fig.add_trace(go.Scatter(x=pd.to_datetime(revenue_data.Date, infer_datetime_format=True), y=revenue_data.Revenue.a
    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()
```

Question 1: Use yfinance to Extract Stock Data

1. 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`.
2. 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.
3. **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.

```
#1
tesla = yf.Ticker("TSLA")

#2
tesla_data = tesla.history(period="max")

#3
tesla_data.reset_index(inplace=True)
tesla_data.head()
```

Question 2: Use Webscraping to Extract Tesla Revenue Data

1. Use the `requests` library to download the webpage https://www.macrotrends.net/stocks/charts/TSLA/tesla/revenue. Save the text of the response as a variable named `html_data`.
2. Parse the html data using `beautiful_soup`.
3. Using beautiful soup extract the table with `Tesla Quarterly Revenue` and store it into a dataframe named `tesla_revenue`. The dataframe should have columns `Date` and `Revenue`. Make sure the comma and dollar sign is removed from the `Revenue` column.
4. Remove the rows in the dataframe that are empty strings or are NaN in the Revenue column. Print the entire `tesla_revenue` DataFrame to see if you have any.
5. Display the last 5 row of the `tesla_revenue` dataframe using the `tail` function.

```
#1
tesla_url = "https://www.macrotrends.net/stocks/charts/TSLA/tesla/revenue"
tesla_html_data = requests.get(tesla_url).text

#2
tesla_soup = BeautifulSoup(tesla_html_data, "html5lib")

#3
tesla_tables = tesla_soup.find_all('table')

for index,table in enumerate(tesla_tables):
    if ("Tesla Quarterly Revenue" in str(table)):
        tesla_table_index = index

tesla_revenue = pd.DataFrame(columns=["Date", "Revenue"])

for row in tesla_tables[tesla_table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if (col !=[]):
        date = col[0].text
        revenue = col[1].text.replace("$", "").replace(",", "")
        tesla_revenue = tesla_revenue.append({"Date" : date, "Revenue" : revenue}, ignore_index=True)

#4
tesla_revenue = tesla_revenue[tesla_revenue['Revenue'] != ""]
tesla_revenue

#5
tesla_revenue.tail()
```

Question 3: Use yfinance to Extract Stock Data

1. 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`.
2. 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.
3. **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.

```
#1
gamestop = yf.Ticker("GME")

#2
gme_data = gamestop.history(period="max")

#3
gme_data.reset_index(inplace=True)
gme_data.head()
```

Question 4: Use Webscraping to Extract GME Revenue Data

1. Use the requests library to download the webpage https://www.macrotrends.net/stocks/charts/GME/gamestop/revenue. Save the text of the response as a variable named html_data.
2. Parse the html data using beautiful_soup.
3. Using beautiful soup extract the table with GameStop Quarterly 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 using a method similar to what you did in Question 2.
4. Display the last five rows of the gme_revenue dataframe using the tail function.

```
#1
gme_url = "https://www.macrotrends.net/stocks/charts/GME/gamestop/revenue"
gme_html_data = requests.get(gme_url).text

#2
gme_soup = BeautifulSoup(gme_html_data, "html5lib")

#3
gme_tables = gme_soup.find_all('table')

for index,table in enumerate(gme_tables):
    if ("GameStop Quarterly Revenue" in str(table)):
        gme_table_index = index

gme_revenue = pd.DataFrame(columns=["Date", "Revenue"])

for row in gme_tables[gme_table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if (col !=[]):
        date = col[0].text
        revenue = col[1].text.replace("$", "").replace(",", "")
        gme_revenue = gme_revenue.append({"Date" : date, "Revenue" : revenue}, ignore_index=True)

#4
gme_revenue.tail()
```

Question 5: Plot Stock Graphs

1. Use the make_graph function to graph the Tesla Stock Data, also provide a title for the graph.
2. Use the make_graph function to graph the GameStop Stock Data, also provide a title for the graph.

```
#1
make_graph(tesla_data, tesla_revenue, 'Tesla')

#2
make_graph(gme_data, gme_revenue, 'GameStop')
```

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**

In [1]:

```
!pip install yfinance
#!pip install pandas
#!pip install requests
!pip install bs4
#!pip install plotly
```

```
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/dhcrypto.py:16: CryptographyDeprecationWarning: int_from_bytes is deprecated,
use int.from_bytes instead
  from cryptography.utils import int_from_bytes
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/util.py:25: CryptographyDeprecationWarning: int_from_bytes is deprecated, use
int.from_bytes instead
  from cryptography.utils import int_from_bytes
Collecting yfinance
  Downloading yfinance-0.1.59.tar.gz (25 kB)
Requirement already satisfied: pandas>=0.24 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from yfinance) (1.0.5)
Requirement already satisfied: numpy>=1.15 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from yfinance) (1.18.5)
Requirement already satisfied: requests>=2.20 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from yfinance) (2.24.0)
Collecting multitasking>=0.0.7
  Downloading multitasking-0.0.9.tar.gz (8.1 kB)
Requirement already satisfied: lxml>=4.5.1 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from yfinance) (4.5.1)
Requirement already satisfied: python-dateutil>=2.6.1 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from pandas>=0.24->yfinance) (2.8.
1)
Requirement already satisfied: pytz>=2017.2 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from pandas>=0.24->yfinance) (2020.1)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from requests>=2.20->yfinance) (2.9)
Requirement already satisfied: chardet<4,>=3.0.2 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from requests>=2.20->yfinance) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from requests>=2.20->yfinance) (2020.1
2.5)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from requests>=2.
20->yfinance) (1.25.9)
Requirement already satisfied: six>=1.5 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from python-dateutil>=2.6.1->pandas>=0.24->yfina
nce) (1.15.0)
Building wheels for collected packages: yfinance, multitasking
  Building wheel for yfinance (setup.py) ... done
  Created wheel for yfinance: filename=yfinance-0.1.59-py2.py3-none-any.whl size=23442 sha256=ecfa067b9ba596625794612fb83b8667ea8f3c622683ecbca9a3f3889f
0e8b71
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/26/af/8b/fac1b47dffef567f945641cdc9b67bb25fae5725d462a8cf81
  Building wheel for multitasking (setup.py) ... done
  Created wheel for multitasking: filename=multitasking-0.0.9-py3-none-any.whl size=8366 sha256=46405ab147baa417656ad8e61a5ec62381550137d5a8c788196c0b29
cee8d519

  Stored in directory: /tmp/wsuser/.cache/pip/wheels/26/af/8b/fac1b47dffef567f945641cdc9b67bb25fae5725d462a8cf81
  Building wheel for multitasking (setup.py) ... done
  Created wheel for multitasking: filename=multitasking-0.0.9-py3-none-any.whl size=8366 sha256=46405ab147baa417656ad8e61a5ec62381550137d5a8c788196c0b29
cee8d519
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/ae/25/47/4d68431a7ec1b6c4b5233365934b74c1d4e665bf5f968d363a
Successfully built yfinance multitasking
Installing collected packages: multitasking, yfinance
Successfully installed multitasking-0.0.9 yfinance-0.1.59
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/dhcrypto.py:16: CryptographyDeprecationWarning: int_from_bytes is deprecated,
use int.from_bytes instead
  from cryptography.utils import int_from_bytes
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/util.py:25: CryptographyDeprecationWarning: int_from_bytes is deprecated, use
int.from_bytes instead
  from cryptography.utils import int_from_bytes
Collecting bs4
  Downloading bs4-0.0.1.tar.gz (1.1 kB)
Requirement already satisfied: beautifulsoup4 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from bs4) (4.9.1)
Requirement already satisfied: soupsieve>1.2 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from beautifulsoup4->bs4) (2.0.1)
Building wheels for collected packages: bs4
  Building wheel for bs4 (setup.py) ... done
  Created wheel for bs4: filename=bs4-0.0.1-py3-none-any.whl size=1272 sha256=e436f71a5c5957561a23ec9181722a2772f2995d5fc7a4b447ae4f18eacacd64
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/0a/9e/ba/20e5bbc1afef3a491f0b3bb74d508f99403aabe76eda2167ca
Successfully built bs4
Installing collected packages: bs4
Successfully installed bs4-0.0.1
```

In [2]:

```
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
```

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 [3]:
```
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)
    fig.add_trace(go.Scatter(x=pd.to_datetime(stock_data.Date, infer_datetime_format=True), y=stock_data.Close.astype("float"), name="Share Price"), ro
    fig.add_trace(go.Scatter(x=pd.to_datetime(revenue_data.Date, infer_datetime_format=True), y=revenue_data.Revenue.astype("float"), name="Revenue"),
    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()
```

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 [4]:
```
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 [5]:
```
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 [7]:
```
tesla_data.reset_index(inplace=True)
tesla_data.head()
```

Out[7]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2010-06-29	3.800	5.000	3.508	4.778	93831500	0	0.0
1	2010-06-30	5.158	6.084	4.660	4.766	85935500	0	0.0
2	2010-07-01	5.000	5.184	4.054	4.392	41094000	0	0.0
3	2010-07-02	4.600	4.620	3.742	3.840	25699000	0	0.0
4	2010-07-06	4.000	4.000	3.166	3.222	34334500	0	0.0

Question 2: Use Webscraping to Extract Tesla Revenue Data

Use the requests library to download the webpage https://www.macrotrends.net/stocks/charts/TSLA/tesla/revenue. Save the text of the response as a variable named html_data.

In [68]:
```
tesla_url = "https://www.macrotrends.net/stocks/charts/TSLA/tesla/revenue"
tesla_html_data = requests.get(tesla_url).text
```

Parse the html data using `beautiful_soup`.

In [69]:
```
tesla_soup = BeautifulSoup(tesla_html_data, "html5lib")
```

Using beautiful soup extract the table with Tesla Quarterly Revenue and store it into a dataframe named tesla_revenue. The dataframe should have columns Date and Revenue. Make sure the comma and dollar sign is removed from the Revenue column.

In [74]:
```
tesla_tables = tesla_soup.find_all('table')

for index,table in enumerate(tesla_tables):
    if ("Tesla Quarterly Revenue" in str(table)):
        tesla_table_index = index

tesla_revenue = pd.DataFrame(columns=["Date", "Revenue"])

for row in tesla_tables[tesla_table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if (col !=[]):
        date = col[0].text
        revenue = col[1].text.replace("$", "").replace(",", "")
        tesla_revenue = tesla_revenue.append({"Date" : date, "Revenue" : revenue}, ignore_index=True)
```

Click here if you need help removing the dollar sign and comma

Remove the rows in the dataframe that are empty strings or are NaN in the Revenue column. Print the entire tesla_revenue DataFrame to see if you have any.

Click here if you need help removing the dollar sign and comma ``` If you parsed the HTML table by row and column you can use the replace function on the string revenue = col[1].text.replace("
$$",,).replace(,,,)If youusetheread_html functionyoucanusethereplacefunctiononthestringrepresentationofthecolumntesla_revenue[Revenue] = tesla_rev$$
", "").str.replace(",", "") ```

Remove the rows in the dataframe that are empty strings or are NaN in the Revenue column. Print the entire tesla_revenue DataFrame to see if you have any.

In [75]:
```
tesla_revenue = tesla_revenue[tesla_revenue['Revenue'] != ""]
tesla_revenue
```

Out[75]:

	Date	Revenue
0	2021-03-31	10389
1	2020-12-31	10744
2	2020-09-30	8771
3	2020-06-30	6036
4	2020-03-31	5985
5	2019-12-31	7384
6	2019-09-30	6303
7	2019-06-30	6350
8	2019-03-31	4541
9	2018-12-31	7226
10	2018-09-30	6824
11	2018-06-30	4002
12	2018-03-31	3409
13	2017-12-31	3288
14	2017-09-30	2985
15	2017-06-30	2790
16	2017-03-31	2696
17	2016-12-31	2285
18	2016-09-30	2298
19	2016-06-30	1270
20	2016-03-31	1147
21	2015-12-31	1214
22	2015-09-30	937
23	2015-06-30	955
24	2015-03-31	940
25	2014-12-31	957
25	2014-12-31	957
26	2014-09-30	852
27	2014-06-30	769
28	2014-03-31	621
29	2013-12-31	615
30	2013-09-30	431
31	2013-06-30	405
32	2013-03-31	562
33	2012-12-31	306
34	2012-09-30	50
35	2012-06-30	27
36	2012-03-31	30
37	2011-12-31	39
38	2011-09-30	58
39	2011-06-30	58
40	2011-03-31	49
41	2010-12-31	36
42	2010-09-30	31
43	2010-06-30	28
44	2010-03-31	21
46	2009-09-30	46
47	2009-06-30	27

Click here if you need help removing the Nan or empty strings ``` If you have NaN in the Revenue column tesla_revenue.dropna(inplace=True) If you have emtpty string in the Revenue column 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 [76]: tesla_revenue.tail()
```

Out[76]:

	Date	Revenue
42	2010-09-30	31
43	2010-06-30	28
44	2010-03-31	21
46	2009-09-30	46
47	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 [19]: 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 [20]: 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 [22]: gme_data.reset_index(inplace=True)
         gme_data.head()
```

Out[22]:

	index	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	0	2002-02-13	6.480513	6.773399	6.413183	6.766666	19054000	0.0	0.0
1	1	2002-02-14	6.850831	6.864296	6.682506	6.733003	2755400	0.0	0.0
2	2	2002-02-15	6.733001	6.749833	6.632006	6.699336	2097400	0.0	0.0
3	3	2002-02-19	6.665671	6.665671	6.312189	6.430017	1852600	0.0	0.0
4	4	2002-02-20	6.463681	6.648838	6.413183	6.648838	1723200	0.0	0.0

Question 4: Use Webscraping to Extract GME Revenue Data

Use the requests library to download the webpage https://www.macrotrends.net/stocks/charts/GME/gamestop/revenue. Save the text of the response as a variable named html_data.

In [77]:
```
gme_url = "https://www.macrotrends.net/stocks/charts/GME/gamestop/revenue"
gme_html_data = requests.get(gme_url).text
```

Parse the html data using `beautiful_soup`.

In [78]:
```
gme_soup = BeautifulSoup(gme_html_data, "html5lib")
```

Using beautiful soup extract the table with GameStop Quarterly 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 using a method similar to what you did in Question 2.

In [79]:
```
gme_tables = gme_soup.find_all('table')

for index,table in enumerate(gme_tables):
    if ("GameStop Quarterly Revenue" in str(table)):
        gme_table_index = index

gme_revenue = pd.DataFrame(columns=["Date", "Revenue"])

for row in gme_tables[gme_table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if (col !=[]):
        date = col[0].text
        revenue = col[1].text.replace("$", "").replace(",", "")
        gme_revenue = gme_revenue.append({"Date" : date, "Revenue" : revenue}, ignore_index=True)
```

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

In [80]:
```
gme_revenue.tail()
```

Out[80]:

	Date	Revenue
60	2006-01-31	1667
61	2005-10-31	534
62	2005-07-31	416
63	2005-04-30	475
64	2005-01-31	709

Question 5: Plot Tesla Stock Graph

Use the make_graph function to graph the Tesla Stock Data, also provide a title for the graph. The structure to call the make_graph function is make_graph(tesla_data, tesla_revenue, 'Tesla')

In [81]:
```
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').

In [82]:
```
make_graph(gme_data, gme_revenue, 'GameStop')
```

Final Assignment

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**

In [1]:
```
!pip install yfinance
#!pip install pandas
#!pip install requests
!pip install bs4
#!pip install plotly
```

```
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/dhcrypto.py:16: CryptographyDeprecationWarning: int_f
rom_bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/util.py:25: CryptographyDeprecationWarning: int_from_
bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
Collecting yfinance
  Downloading yfinance-0.1.59.tar.gz (25 kB)
Requirement already satisfied: pandas>=0.24 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from yfinance) (1.
0.5)
Requirement already satisfied: numpy>=1.15 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from yfinance) (1.1
8.5)
Requirement already satisfied: requests>=2.20 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from yfinance)
(2.24.0)
Collecting multitasking>=0.0.7
  Downloading multitasking-0.0.9.tar.gz (8.1 kB)
Requirement already satisfied: lxml>=4.5.1 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from yfinance) (4.5.
1)
Requirement already satisfied: python-dateutil>=2.6.1 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from pand
as>=0.24->yfinance) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from pandas>=0.24->
yfinance) (2020.1)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from requests>=2.20
->yfinance) (2.9)
Requirement already satisfied: chardet<4,>=3.0.2 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from requests>
=2.20->yfinance) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from requests
>=2.20->yfinance) (2020.12.5)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-pa
ckages (from requests>=2.20->yfinance) (1.25.9)
Requirement already satisfied: six>=1.5 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from python-dateutil>=
2.6.1->pandas>=0.24->yfinance) (1.15.0)
Building wheels for collected packages: yfinance, multitasking
  Building wheel for yfinance (setup.py) ... done
  Created wheel for multitasking: filename=multitasking-0.0.9-py3-none-any.whl size=8366 sha256=46405ab147baa417656ad8e61a5ec62
381550137d5a8c788196c0b29cee8d519
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/ae/25/47/4d68431a7ec1b6c4b5233365934b74c1d4e665bf5f968d363a
Successfully built yfinance multitasking
Installing collected packages: multitasking, yfinance
Successfully installed multitasking-0.0.9 yfinance-0.1.59
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/dhcrypto.py:16: CryptographyDeprecationWarning: int_f
rom_bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/util.py:25: CryptographyDeprecationWarning: int_from_
bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
Collecting bs4
  Downloading bs4-0.0.1.tar.gz (1.1 kB)
Requirement already satisfied: beautifulsoup4 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from bs4) (4.9.1)
Requirement already satisfied: soupsieve>1.2 in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (from beautifulsoup
4->bs4) (2.0.1)
Building wheels for collected packages: bs4
  Building wheel for bs4 (setup.py) ... done
  Created wheel for bs4: filename=bs4-0.0.1-py3-none-any.whl size=1272 sha256=e436f71a5c5957561a23ec9181722a2772f2995d5fc7a4b44
7ae4f18eacacd64
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/0a/9e/ba/20e5bbc1afef3a491f0b3bb74d508f99403aabe76eda2167ca
Successfully built bs4
Installing collected packages: bs4
Successfully installed bs4-0.0.1
```

In [2]:
```
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
```

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 [3]:
```
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"), vert
ical_spacing = .3)
    fig.add_trace(go.Scatter(x=pd.to_datetime(stock_data.Date, infer_datetime_format=True), y=stock_data.Close.astype("float"),
name="Share Price"), row=1, col=1)
    fig.add_trace(go.Scatter(x=pd.to_datetime(revenue_data.Date, infer_datetime_format=True), y=revenue_data.Revenue.astype("flo
at"), 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()
```

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 [4]:
```
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 [5]:
```
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 [7]:
```
tesla_data.reset_index(inplace=True)
tesla_data.head()
```

Out[7]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2010-06-29	3.800	5.000	3.508	4.778	93831500	0	0.0
1	2010-06-30	5.158	6.084	4.660	4.766	85935500	0	0.0
2	2010-07-01	5.000	5.184	4.054	4.392	41094000	0	0.0
3	2010-07-02	4.600	4.620	3.742	3.840	25699000	0	0.0
4	2010-07-06	4.000	4.000	3.166	3.222	34334500	0	0.0

Question 2: Use Webscraping to Extract Tesla Revenue Data¶

Use the requests library to download the webpage https://www.macrotrends.net/stocks/charts/TSLA/tesla/revenue. Save the text of the response as a variable named html_data.

In [68]:
```
tesla_url = "https://www.macrotrends.net/stocks/charts/TSLA/tesla/revenue"
tesla_html_data = requests.get(tesla_url).text
```

Parse the html data using `beautiful_soup`.

In [69]:
```
tesla_soup = BeautifulSoup(tesla_html_data, "html5lib")
```

Using beautiful soup extract the table with Tesla Quarterly Revenue and store it into a dataframe named tesla_revenue. The dataframe should have columns Date and Revenue. Make sure the comma and dollar sign is removed from the Revenue column.

In [74]:
```
tesla_tables = tesla_soup.find_all('table')

for index,table in enumerate(tesla_tables):
    if ("Tesla Quarterly Revenue" in str(table)):
        tesla_table_index = index

tesla_revenue = pd.DataFrame(columns=["Date", "Revenue"])

for row in tesla_tables[tesla_table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if (col !=[]):
        date = col[0].text
        revenue = col[1].text.replace("$", "").replace(",", "")
        tesla_revenue = tesla_revenue.append({"Date" : date, "Revenue" : revenue}, ignore_index=True)
```

Click here if you need help removing the dollar sign and comma

Remove the rows in the dataframe that are empty strings or are NaN in the Revenue column. Print the entire tesla_revenue DataFrame to see if you have any.

```
In [75]: tesla_revenue = tesla_revenue[tesla_revenue['Revenue'] != ""]
         tesla_revenue
```

Out[75]:

	Date	Revenue
0	2021-03-31	10389
1	2020-12-31	10744
2	2020-09-30	8771
3	2020-06-30	6036
4	2020-03-31	5985
5	2019-12-31	7384
6	2019-09-30	6303
7	2019-06-30	6350
8	2019-03-31	4541
9	2018-12-31	7226
10	2018-09-30	6824
11	2018-06-30	4002
12	2018-03-31	3409
13	2017-12-31	3288
14	2017-09-30	2985
15	2017-06-30	2790
16	2017-03-31	2696
17	2016-12-31	2285
18	2016-09-30	2298
19	2016-06-30	1270
20	2016-03-31	1147
21	2015-12-31	1214
22	2015-09-30	937
23	2015-06-30	955
24	2015-03-31	940
25	2014-12-31	957
26	2014-09-30	852
27	2014-06-30	769
28	2014-03-31	621
29	2013-12-31	615
30	2013-09-30	431
31	2013-06-30	405
32	2013-03-31	562
33	2012-12-31	306
34	2012-09-30	50
35	2012-06-30	27
36	2012-03-31	30
37	2011-12-31	39
38	2011-09-30	58
39	2011-06-30	58
40	2011-03-31	49
41	2010-12-31	36
42	2010-09-30	31
43	2010-06-30	28
43	2010-06-30	28
44	2010-03-31	21
46	2009-09-30	46
47	2009-06-30	27

Click here if you need help removing the Nan or empty strings

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

```
In [76]: tesla_revenue.tail()
```

Out[76]:

	Date	Revenue
42	2010-09-30	31
43	2010-06-30	28
44	2010-03-31	21
46	2009-09-30	46
47	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 [19]: 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 [20]: 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 [22]:
```
gme_data.reset_index(inplace=True)
gme_data.head()
```

Out[22]:

	index	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	0	2002-02-13	6.480513	6.773399	6.413183	6.766666	19054000	0.0	0.0
1	1	2002-02-14	6.850831	6.864296	6.682506	6.733003	2755400	0.0	0.0
2	2	2002-02-15	6.733001	6.749833	6.632006	6.699336	2097400	0.0	0.0
3	3	2002-02-19	6.665671	6.665671	6.312189	6.430017	1852600	0.0	0.0
4	4	2002-02-20	6.463681	6.648838	6.413183	6.648838	1723200	0.0	0.0

Question 4: Use Webscraping to Extract GME Revenue Data

Use the requests library to download the webpage https://www.macrotrends.net/stocks/charts/GME/gamestop/revenue. Save the text of the response as a variable named html_data.

In [77]:
```
gme_url = "https://www.macrotrends.net/stocks/charts/GME/gamestop/revenue"
gme_html_data = requests.get(gme_url).text
```

Parse the html data using `beautiful_soup`.

In [78]:
```
gme_soup = BeautifulSoup(gme_html_data, "html5lib")
```

Using beautiful soup extract the table with GameStop Quarterly 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 using a method similar to what you did in Question 2.

In [79]:
```
gme_tables = gme_soup.find_all('table')

for index,table in enumerate(gme_tables):
    if ("GameStop Quarterly Revenue" in str(table)):
        gme_table_index = index

gme_revenue = pd.DataFrame(columns=["Date", "Revenue"])

for row in gme_tables[gme_table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if (col !=[]):
        date = col[0].text
        revenue = col[1].text.replace("$", "").replace(",", "")
        gme_revenue = gme_revenue.append({"Date" : date, "Revenue" : revenue}, ignore_index=True)
```

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

```
In [80]: gme_revenue.tail()
```

Out[80]:

	Date	Revenue
60	2006-01-31	1667
61	2005-10-31	534
62	2005-07-31	416
63	2005-04-30	475
64	2005-01-31	709

Question 5: Plot Tesla Stock Graph¶

Use the make_graph function to graph the Tesla Stock Data, also provide a title for the graph. The structure to call the make_graph function is make_graph(tesla_data, tesla_revenue, 'Tesla')

```
In [81]: 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').

```
In [82]: make_graph(gme_data, gme_revenue, 'GameStop')
```

https://github.com/navassherif98/IBM_Data_Science_Professional_Certification/tree/master/5.Python%20Project%20for%20Data%20Science/Week%201%20Crowdsourcing%20Short%20squeeze%20Dashboard

https://technorj.com/python-for-data-science-and-ai-course/

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

In [2]:
```
!pip install yfinance
#!pip install pandas
#!pip install requests
!pip install bs4
#!pip install plotly
```

```
Requirement already satisfied: yfinance in c:\users\admin\anaconda3\lib\site-packages (0.1.63)
Requirement already satisfied: requests>=2.20 in c:\users\admin\anaconda3\lib\site-packages (from yfinance) (2.24.0)
Requirement already satisfied: numpy>=1.15 in c:\users\admin\anaconda3\lib\site-packages (from yfinance) (1.18.5)
Requirement already satisfied: multitasking>=0.0.7 in c:\users\admin\anaconda3\lib\site-packages (from yfinance) (0.0.9)
Requirement already satisfied: pandas>=0.24 in c:\users\admin\anaconda3\lib\site-packages (from yfinance) (1.0.5)
Requirement already satisfied: lxml>=4.5.1 in c:\users\admin\anaconda3\lib\site-packages (from yfinance) (4.5.2)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\users\admin\anaconda3\lib\site-packages (from requests>=2.20->yfinance) (1.
25.9)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\admin\anaconda3\lib\site-packages (from requests>=2.20->yfinance) (2020.6.20)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\users\admin\anaconda3\lib\site-packages (from requests>=2.20->yfinance) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in c:\users\admin\anaconda3\lib\site-packages (from requests>=2.20->yfinance) (2.10)
Requirement already satisfied: python-dateutil>=2.6.1 in c:\users\admin\anaconda3\lib\site-packages (from pandas>=0.24->yfinance) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in c:\users\admin\anaconda3\lib\site-packages (from pandas>=0.24->yfinance) (2020.1)
Requirement already satisfied: six>=1.5 in c:\users\admin\anaconda3\lib\site-packages (from python-dateutil>=2.6.1->pandas>=0.24->yfinance) (1.15.0)
Requirement already satisfied: bs4 in c:\users\admin\anaconda3\lib\site-packages (0.0.1)
Requirement already satisfied: beautifulsoup4 in c:\users\admin\anaconda3\lib\site-packages (from bs4) (4.9.1)
Requirement already satisfied: soupsieve>1.2 in c:\users\admin\anaconda3\lib\site-packages (from beautifulsoup4->bs4) (2.0.1)
```

In [3]:
```
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
```

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 [4]:
```
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=
    fig.add_trace(go.Scatter(x=pd.to_datetime(revenue_data_specific.Date, infer_datetime_format=True), y=revenue_data_specific.Revenue.astype("float"),
    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()
```

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 [5]:
```
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 [6]:
```
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 [7]:
```
tesla_data.reset_index(inplace=True)
tesla_data.head(5)
```

Out[7]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2010-06-29	3.800	5.000	3.508	4.778	93831500	0	0.0
1	2010-06-30	5.158	6.084	4.660	4.766	85935500	0	0.0
2	2010-07-01	5.000	5.184	4.054	4.392	41094000	0	0.0
3	2010-07-02	4.600	4.620	3.742	3.840	25699000	0	0.0
4	2010-07-06	4.000	4.000	3.166	3.222	34334500	0	0.0

Question 2: Use Webscraping to Extract Tesla Revenue Data

Use the requests library to download the webpage https://www.macrotrends.net/stocks/charts/TSLA/tesla/revenue. Save the text of the response as a variable named html_data.

In [68]:
```
url = "https://www.macrotrends.net/stocks/charts/TSLA/tesla/revenue"
html_data = requests.get(url).text
```

Parse the html data using beautiful_soup.

In [69]:
```
soup = BeautifulSoup(html_data)
```

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

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 like in the previous lab soup.find_all("tbody")[1] If you want to use the read_html function the table is located at index 1 ```

In [70]:
```
data = []
for table in soup.find_all("table"):

    if any(["Tesla Quarterly Revenue".lower() in th.text.lower() for th in table.find_all("th")]):
        for row in table.find("tbody").find_all("tr"):
            date_col, rev_col = [col for col in row.find_all("td")]
            data.append({
                "Date": date_col.text,
                "Revenue": rev_col.text
            })

tesla_revenue = pd.DataFrame(data)
```

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

In [71]:
```
tesla_revenue["Revenue"] = tesla_revenue['Revenue'].str.replace(',|\$',"")
```

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

In [72]:
```
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 [73]:
```
tesla_revenue.tail(5)
```

Out[73]:

	Date	Revenue
43	2010-09-30	31
44	2010-06-30	28
45	2010-03-31	21
47	2009-09-30	46
48	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 [74]:
```
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.

In [75]:
```
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.

In [76]:
```
gme_data.reset_index(inplace=True)
gme_data.head()
```

Out[76]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2002-02-13	6.480513	6.773399	6.413183	6.766666	19054000	0.0	0.0
1	2002-02-14	6.850831	6.864296	6.682506	6.733003	2755400	0.0	0.0
2	2002-02-15	6.733001	6.749833	6.632006	6.699336	2097400	0.0	0.0
3	2002-02-19	6.665671	6.665671	6.312189	6.430017	1852600	0.0	0.0
4	2002-02-20	6.463681	6.648838	6.413183	6.648838	1723200	0.0	0.0

Question 4: Use Webscraping to Extract GME Revenue Data

Use the requests library to download the webpage https://www.macrotrends.net/stocks/charts/GME/gamestop/revenue. Save the text of the response as a variable named html_data.

In [77]:
```
url = "https://www.macrotrends.net/stocks/charts/GME/gamestop/revenue"
html_data = requests.get(url).text
```

Parse the html data using beautiful_soup.

In [78]:
```
soup = BeautifulSoup(html_data)
```

Using BeautifulSoup or the read_html function extract the table with GameStop Quarterly 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 using a 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 like in the previous lab soup.find_all("tbody")[1] If you want to use the read_html function the table is located at index 1 ```

In [79]:
```
data = []
for table in soup.find_all("table"):

    if any(["GameStop Quarterly Revenue".lower() in th.text.lower() for th in table.find_all("th")]):
        for row in table.find("tbody").find_all("tr"):
            date_col, rev_col = [col for col in row.find_all("td")]
            data.append({
                "Date": date_col.text,
                "Revenue": rev_col.text.replace("$", "").replace(",", "")
            })

gme_revenue = pd.DataFrame(data)
```

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

In [80]:
```
gme_revenue.tail()
```

Out[80]:

	Date	Revenue
61	2006-01-31	1667
62	2005-10-31	534
63	2005-07-31	416
64	2005-04-30	475
65	2005-01-31	709

Question 5: Plot Tesla Stock Graph

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

In [81]:
```
make_graph(tesla_data, tesla_revenue, 'Telsa (revenue vs. price comparison)')
```

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.

In [82]:
```
make_graph(gme_data, gme_revenue, 'GameStop (revenue vs. price comparison)')
```

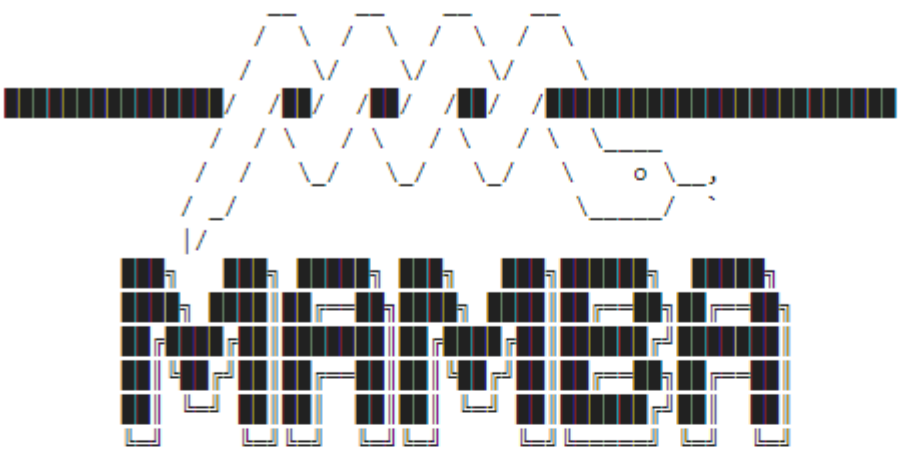
Extracting Stock Data Using a Web Scraping (Jan 2022)

Not all stock data is available via API in this assignment; you will use web-scraping to obtain financial data. You will be quizzed on your results.
Using beautiful soup we will extract historical share data from a web-page.

Table of Contents

Estimated Time Needed: **30 min**

[1]:
```
#!pip install pandas==1.3.3
#!pip install requests==2.26.0
!mamba install bs4==4.10.0 -y
!mamba install html5lib==1.1 -y
!pip install lxml==4.6.4
#!pip install plotly==5.3.1
```



```
mamba (0.15.3) supported by @QuantStack

GitHub:  https://github.com/mamba-org/mamba
Twitter: https://twitter.com/QuantStack
```

```
Looking for: ['bs4==4.10.0']

pkgs/main/noarch         [<=>                ] (00m:00s)
pkgs/main/noarch         [=>                 ] (00m:00s) 364 KB / ?? (1.17 MB/s)
pkgs/main/noarch         [=>                 ] (00m:00s) 364 KB / ?? (1.17 MB/s)
pkgs/r/linux-64          [<=>                ] (00m:00s)
pkgs/main/noarch         [=>                 ] (00m:00s) 364 KB / ?? (1.17 MB/s)
pkgs/r/linux-64          [=>                 ] (00m:00s) 364 KB / ?? (1.17 MB/s)
pkgs/main/noarch         [=>                 ] (00m:00s) 364 KB / ?? (1.17 MB/s)
pkgs/r/linux-64          [=>                 ] (00m:00s) 364 KB / ?? (1.17 MB/s)
pkgs/r/noarch            [<=>                ] (00m:00s)
pkgs/main/noarch         [=>                 ] (00m:00s) 364 KB / ?? (1.17 MB/s)
```

[2]:
```
import pandas as pd
import requests
from bs4 import BeautifulSoup
```

Using Webscraping to Extract Stock Data Example

First we must use the request library to downlaod the webpage, and extract the text. We will extract Netflix stock data https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0220EN-SkillsNetwork/labs/project/netflix_data_webpage.html.

```
[6]: netflix_data = pd.DataFrame(columns=["Date", "Open", "High", "Low", "Close", "Volume"])

# First we isolate the body of the table which contains all the information
# Then we loop through each row and find all the column values for each row
for row in soup.find("tbody").find_all('tr'):
    col = row.find_all("td")
    date = col[0].text
    Open = col[1].text
    high = col[2].text
    low = col[3].text
    close = col[4].text
    adj_close = col[5].text
    volume = col[6].text

------
    # Finally we append the data of each row to the table
    netflix_data = netflix_data.append({"Date":date, "Open":Open, "High":high, "Low":low, "Close":close, "Adj Close":adj_close, "Volume":volume}, ignore_index=True)
```

We can now print out the dataframe

We can now print out the dataframe

```
[6]: netflix_data.head()
```

[6]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	Jun 01, 2021	504.01	536.13	482.14	528.21	78,560,600	528.21
1	May 01, 2021	512.65	518.95	478.54	502.81	66,927,600	502.81
2	Apr 01, 2021	529.93	563.56	499.00	513.47	111,573,300	513.47
3	Mar 01, 2021	545.57	556.99	492.85	521.66	90,183,900	521.66
4	Feb 01, 2021	536.79	566.65	518.28	538.85	61,902,300	538.85

We can also use the pandas `read_html` function using the url

```
[7]: read_html_pandas_data = pd.read_html(url)
```

Or we can convert the BeautifulSoup object to a string

```
[8]: read_html_pandas_data = pd.read_html(str(soup))
```

Beacause there is only one table on the page, we just take the first table in the list returned

```
[9]: netflix_dataframe = read_html_pandas_data[0]

     netflix_dataframe.head()
```

[9]:

	Date	Open	High	Low	Close*	Adj Close**	Volume
0	Jun 01, 2021	504.01	536.13	482.14	528.21	528.21	78560600
1	May 01, 2021	512.65	518.95	478.54	502.81	502.81	66927600
2	Apr 01, 2021	529.93	563.56	499.00	513.47	513.47	111573300
3	Mar 01, 2021	545.57	556.99	492.85	521.66	521.66	90183900
4	Feb 01, 2021	536.79	566.65	518.28	538.85	538.85	61902300

Using Webscraping to Extract Stock Data Exercise

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

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

Parse the html data using `beautiful_soup`.

```
[20]: soup = BeautifulSoup(html_data, "html5lib")
```

Question 1 What is the content of the title attribute:

```
[30]: soup.find_all('title')
```

```
[30]: [<title>Amazon.com, Inc. (AMZN) Stock Historical Prices &amp;amp; Data - Yahoo Finance</title>]
```

Using beautiful soup extract the table with historical share prices and store it into a dataframe named amazon_data. The dataframe should have columns Date, Open, High, Low, Close, Adj Close, and Volume. Fill in each variable with the correct data from the list col.

```
[8]: amazon_data = pd.DataFrame(columns=["Date", "Open", "High", "Low", "Close", "Volume"])

for row in soup.find("tbody").find_all("tr"):
    col = row.find_all("td")
    date =col[0].text
    Open = col[1].text
    high = col[2].text
    low = col[3].text
    close = col[4].text
    adj_close = col[5].text
    volume = col[6].text
------
    amazon_data = amazon_data.append({"Date":date, "Open":Open, "High":high, "Low":low, "Close":close, "Adj Close":adj_close, "Volume":volume}, ignore_index=True)
```

Print out the first five rows of the `amazon_data` dataframe you created.

```
[9]: amazon_data.head()
```

[9]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	Jun 01, 2021	504.01	536.13	482.14	528.21	78,560,600	528.21
1	May 01, 2021	512.65	518.95	478.54	502.81	66,927,600	502.81
2	Apr 01, 2021	529.93	563.56	499.00	513.47	111,573,300	513.47
3	Mar 01, 2021	545.57	556.99	492.85	521.66	90,183,900	521.66
4	Feb 01, 2021	536.79	566.65	518.28	538.85	61,902,300	538.85

Question 2 What is the name of the columns of the dataframe

[10]:
```
for col in amazon_data.columns:
    print (col)
```

```
Date
Open
High
Low
Close
Volume
Adj Close
```

Question 3 What is the `Open` of the last row of the amazon_data dataframe?

[11]:
```
index = amazon_data[amazon_data['Date']=='Feb 01, 2021'].index.values
amazon_data.loc[index, 'Open']
```

[11]:
```
index = amazon_data[amazon_data['Date']=='Feb 01, 2021'].index.values
amazon_data.loc[index, 'Open']
```

[11]:
```
4    536.79
Name: Open, dtype: object
```

Quiz: Extracting Stock Data Using a Web Scraping

Bookmarked

Graded Exam due Feb 9, 2022 09:07 +08

Question 1

1/1 point (graded)

In the lab exercise, what is the content of the title attribute from the object soup?

- (●) < title > Amazon.com, Inc. (AMZN) Stock Historical Prices & Data - Yahoo Finance < /title >
- ( ) (AMZN) Stock Historical Prices & Data - Yahoo Finance
- ( ) < b class="Hidden" > Yahoo Finance < /b >

✔

Submit You have used 1 of 2 attempts Reset

✔ Correct (1/1 point)

Question 2

1/1 point (graded)

In the lab exercise, what are the correct names of the columns of the dataframe?

- (●) 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'
- ( ) 'Date', 'Open', 'High', 'Low'
- ( ) 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close', 'max','min'

✔

Submit You have used 1 of 2 attempts

✔ Correct (1/1 point)

Question 3

1/1 point (graded)

In the lab exercise What is the Open of Jun 01, 2019 in the dataframe?

- (●) 1,760.01
- ( ) 1,935.20
- ( ) 1,672.00

✔

Submit You have used 2 of 2 attempts

✔ Correct (1/1 point)

Reading: Tesla and GameStop Analytical Dashboard

Determining the price of a stock is complex; It depends on many things like revenue, profits, losses. company news. public perception, future business, and much more. An essential factor is the company's growth of profit/revenue. If the company reports good profits/revenue, the stock price should increase and if it reports bad profits/revenues the stock price will decrease.

Tesla and GameStop have become very popular stocks, with GameStop being one of the fastest rising and popular stocks of the year. Both have seen tremendous gains and for different reasons. In the course project, we will extract revenue and stock price data for Tesla and GameStop and build a dashboard to compare the price of the stock vs the revenue. This dashboard will not only provide information about the revenue and stock price but will allow us to see if there is a correlation between the two.