Scraping data from web by Python

Example 1 Get Financial Data from Yahoo Finance with Python

Source: https://www.geeksforgeeks.org/get-financial-data-from-yahoo-finance-with-python/ (https://www.geeksforgeeks.org/get-financial-data-from-yahoo-finance-with-python/)

In [5]:
```
#Installation of Yahoo Finance Module
!pip install yfinance
```

```
Collecting yfinance
  Using cached yfinance-0.2.31-py2.py3-none-any.whl (65 kB)
Requirement already satisfied: requests>=2.31 in c:\users\lenovo\anaconda3\li
b\site-packages (from yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in c:\users\lenovo\anacond
a3\lib\site-packages (from yfinance) (0.0.11)
Requirement already satisfied: numpy>=1.16.5 in c:\users\lenovo\anaconda3\lib
\site-packages (from yfinance) (1.22.4)
Requirement already satisfied: pandas>=1.3.0 in c:\users\lenovo\anaconda3\lib
\site-packages (from yfinance) (1.4.2)
Collecting frozendict>=2.3.4
  Using cached frozendict-2.3.8-cp39-cp39-win_amd64.whl (35 kB)
Requirement already satisfied: peewee>=3.16.2 in c:\users\lenovo\anaconda3\li
b\site-packages (from yfinance) (3.17.0)
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\users\lenovo\anac
onda3\lib\site-packages (from yfinance) (4.11.1)
Collecting html5lib>=1.1
  Using cached html5lib-1.1-py2.py3-none-any.whl (112 kB)
Requirement already satisfied: appdirs>=1.4.4 in c:\users\lenovo\anaconda3\li
b\site-packages (from yfinance) (1.4.4)
Requirement already satisfied: lxml>=4.9.1 in c:\users\lenovo\anaconda3\lib\s
ite-packages (from yfinance) (4.9.3)
Requirement already satisfied: pytz>=2022.5 in c:\users\lenovo\anaconda3\lib
\site-packages (from yfinance) (2023.3.post1)
Requirement already satisfied: soupsieve>1.2 in c:\users\lenovo\anaconda3\lib
\site-packages (from beautifulsoup4>=4.11.1->yfinance) (2.3.1)
Requirement already satisfied: webencodings in c:\users\lenovo\anaconda3\lib
\site-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: six>=1.9 in c:\users\lenovo\anaconda3\lib\site
-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\lenovo\anac
onda3\lib\site-packages (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\lenovo\anaconda3\lib
\site-packages (from requests>=2.31->yfinance) (3.3)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\lenovo\anaconda
3\lib\site-packages (from requests>=2.31->yfinance) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\lenovo\anaconda
3\lib\site-packages (from requests>=2.31->yfinance) (2021.10.8)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\lenovo\an
aconda3\lib\site-packages (from requests>=2.31->yfinance) (2.0.4)
Installing collected packages: html5lib, frozendict, yfinance
Successfully installed frozendict-2.3.8 html5lib-1.1 yfinance-0.2.31
```

We need to pass as an argument of Ticker i.e. the company's ticker.

Note: A stock symbol or a ticker is a unique series of letters assigned to a security for trading purposes. For example:

- For Amazon, it is "AMZN"
- For Meta, it is "META"

- For Google, it is “GOOGL”

In [7]:
```
import yfinance
print(yfinance.__version__)
```

```
0.2.31
```

In [9]:
```
import yfinance as yahooFinance

# META financial information
get_META = yahooFinance.Ticker("META")

# whole python dictionary is printed here
print(get_META.info)

get_META_info = get_META.info
for key, value in get_META_info.items():
    print(key,':', value)
```

```
{'address1': '1601 Willow Road', 'city': 'Menlo Park', 'state': 'CA', 'zi
p': '94025', 'country': 'United States', 'phone': '650 543 4800', 'websit
e': 'https://investor.fb.com', 'industry': 'Internet Content & Informatio
n', 'industryKey': 'internet-content-information', 'industryDisp': 'Intern
et Content & Information', 'sector': 'Communication Services', 'sectorKe
y': 'communication-services', 'sectorDisp': 'Communication Services', 'lon
gBusinessSummary': 'Meta Platforms, Inc. engages in the development of pro
ducts that enable people to connect and share with friends and family thro
ugh mobile devices, personal computers, virtual reality headsets, and wear
ables worldwide. It operates in two segments, Family of Apps and Reality L
abs. The Family of Apps segment offers Facebook, which enables people to s
hare, discuss, discover, and connect with interests; Instagram, a communit
y for sharing photos, videos, and private messages, as well as feed, stori
es, reels, video, live, and shops; Messenger, a messaging application for
people to connect with friends, family, communities, and businesses across
platforms and devices through text, audio, and video calls; and WhatsApp,
a messaging application that is used by people and businesses to communica
te and transact privately. The Reality Labs segment provides augmented and
virtual reality related products comprising consumer hardware, software, a
```

In [10]:
```
# META key metrics

# display Company Sector
print("Company Sector : ", get_META_info['sector'])

# display Price Earnings Ratio
print("Price Earnings Ratio : ", get_META_info['trailingPE'])

# display Company Beta
print("Company Beta : ", get_META_info['beta'])
```

```
Company Sector :  Communication Services
Price Earnings Ratio :  34.829067
Company Beta :  1.21
```

In [12]:
```
# Let us get historical stock prices for META
# covering the past few years.
# max->maximum number of daily prices available
# for Facebook.
# Valid options are 1d, 5d, 1mo, 3mo, 6mo, 1y, 2y,
# 5y, 10y and ytd.
print(get_META.history(period="max"))
```

```
                                 Open        High         Low       Close  \
Date
2012-05-18 00:00:00-04:00   42.049999   45.000000   38.000000   38.230000
2012-05-21 00:00:00-04:00   36.529999   36.660000   33.000000   34.029999
2012-05-22 00:00:00-04:00   32.610001   33.590000   30.940001   31.000000
2012-05-23 00:00:00-04:00   31.370001   32.500000   31.360001   32.000000
2012-05-24 00:00:00-04:00   32.950001   33.209999   31.770000   33.029999
...                               ...         ...         ...         ...
2023-10-19 00:00:00-04:00  319.880005  321.890015  311.750000  312.809998
2023-10-20 00:00:00-04:00  314.140015  315.299988  306.470001  308.649994
2023-10-23 00:00:00-04:00  309.500000  317.359985  307.260010  314.010010
2023-10-24 00:00:00-04:00  316.779999  318.350006  310.630005  312.549988
2023-10-25 00:00:00-04:00  310.000000  310.880005  298.839996  299.529999

                              Volume  Dividends  Stock Splits
Date
2012-05-18 00:00:00-04:00  573576400        0.0           0.0
2012-05-21 00:00:00-04:00  168192700        0.0           0.0
2012-05-22 00:00:00-04:00  101786600        0.0           0.0
2012-05-23 00:00:00-04:00   73600000        0.0           0.0
2012-05-24 00:00:00-04:00   50237200        0.0           0.0
...                              ...        ...           ...
2023-10-19 00:00:00-04:00   18709200        0.0           0.0
2023-10-20 00:00:00-04:00   22287400        0.0           0.0
2023-10-23 00:00:00-04:00   17796800        0.0           0.0
2023-10-24 00:00:00-04:00   19525500        0.0           0.0
2023-10-25 00:00:00-04:00   30447269        0.0           0.0

[2878 rows x 7 columns]
```

In [13]:
```
import pandas as pd
# Display all rows

pd.set_option('display.max_rows', None)
# Let us get historical stock prices for META
# covering the past few years.
# max->maximum number of daily prices available
# Valid options are 1d, 5d, 1mo, 3mo, 6mo, 1y, 2y,
# 5y, 10y and ytd.
print(get_META.history(period="max"))
```

```
2013-01-29 00:00:00-05:00   32.000000   32.070000   30.709999   30.790001
2013-01-30 00:00:00-05:00   30.980000   31.490000   30.879999   31.240000
2013-01-31 00:00:00-05:00   29.150000   31.469999   28.740000   30.980000
2013-02-01 00:00:00-05:00   31.010000   31.020000   29.629999   29.730000
2013-02-04 00:00:00-05:00   29.059999   29.200001   28.010000   28.110001
2013-02-05 00:00:00-05:00   28.260000   28.959999   28.040001   28.639999
2013-02-06 00:00:00-05:00   28.740000   29.290001   28.660000   29.049999
2013-02-07 00:00:00-05:00   29.110001   29.150000   28.270000   28.650000
2013-02-08 00:00:00-05:00   28.889999   29.170000   28.510000   28.549999
2013-02-11 00:00:00-05:00   28.610001   28.680000   28.040001   28.260000
2013-02-12 00:00:00-05:00   27.670000   28.160000   27.100000   27.370001
2013-02-13 00:00:00-05:00   27.360001   28.320000   27.309999   27.910000
2013-02-14 00:00:00-05:00   28.020000   28.629999   28.010000   28.500000
2013-02-15 00:00:00-05:00   28.520000   28.750000   28.090000   28.320000
2013-02-19 00:00:00-05:00   28.230000   29.080000   28.120001   28.930000
2013-02-20 00:00:00-05:00   28.920000   29.049999   28.330000   28.459999
2013-02-21 00:00:00-05:00   28.280001   28.549999   27.150000   27.280001
2013-02-22 00:00:00-05:00   27.620001   27.629999   26.820000   27.129999
2013-02-25 00:00:00-05:00   27.160000   27.639999   27.150000   27.270000
2013-02-26 00:00:00-05:00   27.360001   27.459999   26.700001   27.389999
```

In [14]:
```
# Display META historical values for 6 months
# Valid options are 1d, 5d, 1mo, 3mo, 6mo, 1y,
# 2y, 5y, 10y and ytd.
print(get_META.history(period="6mo"))
```

```
                           Open        High         Low       Close
2023-08-31 00:00:00-04:00  295.799988  301.100006  295.660004  295.890015
2023-09-01 00:00:00-04:00  299.369995  301.739990  294.470001  296.380005
2023-09-05 00:00:00-04:00  297.019989  301.390015  295.510010  300.149994
2023-09-06 00:00:00-04:00  301.709991  303.299988  295.660004  299.170013
2023-09-07 00:00:00-04:00  298.000000  307.049988  292.220001  298.670013
2023-09-08 00:00:00-04:00  299.220001  305.250000  296.779999  297.890015
2023-09-11 00:00:00-04:00  301.410004  309.040009  301.279999  307.559998
2023-09-12 00:00:00-04:00  306.329987  308.660004  300.230011  301.660004
2023-09-13 00:00:00-04:00  302.359985  307.179993  301.320007  305.059998
2023-09-14 00:00:00-04:00  306.739990  312.869995  305.029999  311.720001
2023-09-15 00:00:00-04:00  311.609985  312.000000  298.750000  300.309998
2023-09-18 00:00:00-04:00  298.190002  303.600006  297.799988  302.549988
2023-09-19 00:00:00-04:00  302.480011  306.170013  299.809998  305.070007
2023-09-20 00:00:00-04:00  305.049988  308.059998  299.429993  299.670013
2023-09-21 00:00:00-04:00  295.700012  300.260010  293.269989  295.730011
2023-09-22 00:00:00-04:00  299.299988  305.380005  298.269989  299.079987
2023-09-25 00:00:00-04:00  295.640015  300.950012  293.700012  300.829987
2023-09-26 00:00:00-04:00  297.660004  300.299988  296.010010  298.959991
2023-09-27 00:00:00-04:00  300.450012  301.299988  286.790009  297.739990
2023-09-28 00:00:00-04:00  298.940002  306.329987  296.700012  303.959991
```

In [15]:
```
# Display META historical values for a given period of time

# in order to specify start date and
# end date we need datetime package
import datetime

# startDate , as per our convenience we can modify
startDate = datetime.datetime(2023, 6, 30)

# endDate , as per our convenience we can modify
endDate = datetime.datetime(2023, 9, 30)
get_META = yahooFinance.Ticker("META")

# pass the parameters as the taken dates for start and end
print(get_META.history(start=startDate, end=endDate))
```

```
                                 Open        High         Low       Close
\
Date
2023-06-30 00:00:00-04:00  284.760010  289.049988  284.420013  286.980011
2023-07-03 00:00:00-04:00  286.700012  289.399994  284.850006  286.019989
2023-07-05 00:00:00-04:00  287.649994  298.119995  286.359985  294.369995
2023-07-06 00:00:00-04:00  295.890015  298.119995  291.309998  291.989990
2023-07-07 00:00:00-04:00  292.179993  296.200012  288.660004  290.529999
2023-07-10 00:00:00-04:00  295.549988  298.130005  287.049988  294.100006
2023-07-11 00:00:00-04:00  293.899994  300.179993  291.899994  298.290009
2023-07-12 00:00:00-04:00  301.750000  309.450012  300.100006  309.339996
2023-07-13 00:00:00-04:00  313.619995  316.239990  310.290009  313.410004
2023-07-14 00:00:00-04:00  311.790009  314.880005  307.359985  308.869995
2023-07-17 00:00:00-04:00  307.540009  311.709991  304.709991  310.619995
2023-07-18 00:00:00-04:00  310.880005  314.200012  307.619995  312.049988
2023-07-19 00:00:00-04:00  313.029999  318.679993  310.519989  316.010010
2023-07-20 00:00:00-04:00  313.500000  315.540009  302.220001  302.519989
2023-07-21 00:00:00-04:00  304.570007  305.459991  291.200012  294.260010
2023-07-24 00:00:00-04:00  295.779999  297.519989  288.299988  291.609985
```

Save the output to a csv file

In [17]:
```
# Save the output to a csv file
meta_csv = 'meta.csv'

get_META.history(period="max").to_csv(meta_csv)
from IPython.display import FileLink

# Create a download link for the CSV file
FileLink('meta.csv')
```

Out[17]: meta.csv (meta.csv)

Assignment

Now it is your turn. Pick any company or companies, find their ticker and scrape at least 5 different data about them and save it in a csv file with your name and company name.

You may add any number of cells below by pressing the + sign or Insert menu.

I added one cell for you. Feel free to add more

In [18]:
```
# Your code goes here and below
import yfinance as yf

# Ticker symbol for Amazon.com, Inc.
ticker_symbol = "AMZN"

# Fetch financial information
amazon_info = yf.Ticker(ticker_symbol).info

# Print the information for Amazon.com, Inc.
for key, value in amazon_info.items():
    print(key, ':', value)
```

```
fiftyTwoWeekLow : 81.43
fiftyTwoWeekHigh : 145.86
priceToSalesTrailing12Months : 2.3278265
fiftyDayAverage : 132.941
twoHundredDayAverage : 116.79725
trailingAnnualDividendRate : 0.0
trailingAnnualDividendYield : 0.0
currency : USD
enterpriseValue : 1436766044160
profitMargins : 0.024300002
floatShares : 9042786330
sharesOutstanding : 10317800448
sharesShort : 79796989
sharesShortPriorMonth : 80249295
sharesShortPreviousMonthDate : 1694736000
dateShortInterest : 1697155200
sharesPercentSharesOut : 0.0077
heldPercentInsiders : 0.09718
heldPercentInstitutions : 0.60534
shortRatio : 1.45
```

In [19]:
```
# Ticker symbol for Apple Inc.
apple_ticker_symbol = "AAPL"

# Fetch financial information for Apple Inc.
apple_info = yf.Ticker(apple_ticker_symbol).info

# Display Company Sector
print("Company Sector: ", apple_info['sector'])

# Display Price Earnings Ratio (P/E Ratio)b
print("Price Earnings Ratio (P/E Ratio): ", apple_info['trailingPE'])

# Display Company Beta
print("Company Beta: ", apple_info['beta'])
```

```
Company Sector:  Technology
Price Earnings Ratio (P/E Ratio):  28.65997
Company Beta:  1.308
```

In [21]:
```
# Fetch and print historical stock prices for Apple Inc. covering the maximum
apple_historical_prices = yf.Ticker(apple_ticker_symbol).history(period="max")
print(apple_historical_prices)
```

```
                                 Open        High         Low       Close
\
Date
1980-12-12 00:00:00-05:00    0.099450    0.099882    0.099450    0.099450
1980-12-15 00:00:00-05:00    0.094694    0.094694    0.094261    0.094261
1980-12-16 00:00:00-05:00    0.087775    0.087775    0.087343    0.087343
1980-12-17 00:00:00-05:00    0.089504    0.089937    0.089504    0.089504
1980-12-18 00:00:00-05:00    0.092099    0.092532    0.092099    0.092099
1980-12-19 00:00:00-05:00    0.097720    0.098152    0.097720    0.097720
1980-12-22 00:00:00-05:00    0.102476    0.102909    0.102476    0.102476
1980-12-23 00:00:00-05:00    0.106800    0.107233    0.106800    0.106800
1980-12-24 00:00:00-05:00    0.112421    0.112854    0.112421    0.112421
1980-12-26 00:00:00-05:00    0.122799    0.123231    0.122799    0.122799
1980-12-29 00:00:00-05:00    0.124528    0.124961    0.124528    0.124528
1980-12-30 00:00:00-05:00    0.121934    0.121934    0.121502    0.121502
1980-12-31 00:00:00-05:00    0.118475    0.118475    0.118043    0.118043
1981-01-02 00:00:00-05:00    0.119340    0.120205    0.119340    0.119340
1981-01-05 00:00:00-05:00    0.117178    0.117178    0.116746    0.116746
1981-01-06 00:00:00-05:00    0.111989    0.111989    0.111556    0.111556
```

In [23]:
```
import pandas as pd
# Display all rows

pd.set_option('display.max_rows', None)
apple_historical_prices = yf.Ticker(apple_ticker_symbol).history(period="max")
print(apple_historical_prices)
```

```
                                 Open        High         Low       Close
\
Date
1980-12-12 00:00:00-05:00    0.099450    0.099882    0.099450    0.099450
1980-12-15 00:00:00-05:00    0.094694    0.094694    0.094261    0.094261
1980-12-16 00:00:00-05:00    0.087775    0.087775    0.087343    0.087343
1980-12-17 00:00:00-05:00    0.089504    0.089937    0.089504    0.089504
1980-12-18 00:00:00-05:00    0.092099    0.092532    0.092099    0.092099
1980-12-19 00:00:00-05:00    0.097720    0.098152    0.097720    0.097720
1980-12-22 00:00:00-05:00    0.102476    0.102909    0.102476    0.102476
1980-12-23 00:00:00-05:00    0.106800    0.107233    0.106800    0.106800
1980-12-24 00:00:00-05:00    0.112421    0.112854    0.112421    0.112421
1980-12-26 00:00:00-05:00    0.122799    0.123231    0.122799    0.122799
1980-12-29 00:00:00-05:00    0.124528    0.124961    0.124528    0.124528
1980-12-30 00:00:00-05:00    0.121934    0.121934    0.121502    0.121502
1980-12-31 00:00:00-05:00    0.118475    0.118475    0.118043    0.118043
1981-01-02 00:00:00-05:00    0.119340    0.120204    0.119340    0.119340
1981-01-05 00:00:00-05:00    0.117178    0.117178    0.116746    0.116746
1981-01-06 00:00:00-05:00    0.111989    0.111989    0.111556    0.111556
```

In [26]:
```
apple_historical_prices = yf.Ticker(apple_ticker_symbol).history(period="6mo")
print(apple_historical_prices)
```

```
                                 Open        High         Low       Close
\
Date
2023-04-26 00:00:00-04:00  162.615176  164.829121  162.355890  163.313263
2023-04-27 00:00:00-04:00  164.739375  168.100177  164.739375  167.950592
2023-04-28 00:00:00-04:00  168.030376  169.386667  167.422040  169.217117
2023-05-01 00:00:00-04:00  168.818213  169.985020  168.179960  169.127365
2023-05-02 00:00:00-04:00  169.625990  169.885291  167.082943  168.080215
2023-05-03 00:00:00-04:00  169.037605  170.453730  166.703992  166.993195
2023-05-04 00:00:00-04:00  164.440184  166.584313  163.861764  165.337723
2023-05-05 00:00:00-04:00  170.513566  173.824516  170.294165  173.096512
2023-05-08 00:00:00-04:00  172.009466  173.375739  171.640480  173.026688
2023-05-09 00:00:00-04:00  172.577929  173.066583  171.131888  171.301422
2023-05-10 00:00:00-04:00  172.548012  173.555252  171.431057  173.086533
2023-05-11 00:00:00-04:00  173.375750  174.113721  171.700325  173.276016
2023-05-12 00:00:00-04:00  173.385852  173.825261  170.769390  172.337280
2023-05-15 00:00:00-04:00  172.926493  172.976429  171.238770  171.837967
2023-05-16 00:00:00-04:00  171.758073  172.906516  171.568327  171.837967
2023-05-17 00:00:00-04:00  171.478449  172.696789  170.190180  172.457123
```

In [28]:
```
# Display apple historical values for a given period of time

# in order to specify start date and
# end date we need datetime package
import datetime

# startDate , as per our convenience we can modify
startDate = datetime.datetime(2023, 6, 30)

# endDate , as per our convenience we can modify
endDate = datetime.datetime(2023, 9, 30)


# Fetch and print historical stock prices for Apple Inc. within the specified
apple_historical_prices = yf.Ticker(apple_ticker_symbol).history(start=startDa
print(apple_historical_prices)
```

```
                                 Open        High         Low       Close
\
Date
2023-06-30 00:00:00-04:00  191.371579  194.217727  191.002068  193.708420
2023-07-03 00:00:00-04:00  193.518682  193.618553  191.501402  192.200470
2023-07-05 00:00:00-04:00  191.311658  192.719744  190.362927  191.071976
2023-07-06 00:00:00-04:00  189.583986  191.761054  188.944850  191.551331
2023-07-07 00:00:00-04:00  191.151878  192.410173  189.983458  190.422852
2023-07-10 00:00:00-04:00  189.004769  189.733796  186.787762  188.355652
2023-07-11 00:00:00-04:00  188.904916  189.044726  186.348370  187.826370
2023-07-12 00:00:00-04:00  189.424194  191.441474  188.215834  189.514084
2023-07-13 00:00:00-04:00  190.243096  190.932168  189.524066  190.283035
2023-07-14 00:00:00-04:00  189.973460  190.922176  189.374278  190.432846
2023-07-17 00:00:00-04:00  191.641201  194.057950  191.551326  193.728394
2023-07-18 00:00:00-04:00  193.089258  194.067932  192.160504  193.468735
2023-07-19 00:00:00-04:00  192.839596  197.962667  192.390191  194.836899
2023-07-20 00:00:00-04:00  194.826901  196.205045  192.240397  192.869553
2023-07-21 00:00:00-04:00  193.838258  194.707080  190.972118  191.681168
2023-07-24 00:00:00-04:00  193.149180  194.647157  191.990741  192.490067
```

In [30]:
```
import yfinance as yf
from IPython.display import FileLink



# Fetch historical stock prices for Apple Inc. covering the maximum available
apple_historical_prices = yf.Ticker(apple_ticker_symbol).history(period="max")

# Define the CSV file name
sreekethana_apple_csv = 'sreekethana_apple.csv'

# Save the historical stock prices to a CSV file
apple_historical_prices.to_csv(apple_csv)

# Create a download link for the CSV file
FileLink(sreekethana_apple_csv)
```

Out[30]: sreekethana_apple.csv (sreekethana_apple.csv)

Example 2 Web Scraping Financial News Using Python

Source: https://www.geeksforgeeks.org/web-scraping-financial-news-using-python/ (https://www.geeksforgeeks.org/web-scraping-financial-news-using-python/)

Request: This module has several built-in methods to make HTTP requests to specified URI using GET, POST, PUT, PATCH, or HEAD requests. An HTTP request is meant to either retrieve data from a specified URI or push data to a server.

Beautiful Soup: Beautiful Soup is a web scraping framework for Python. Web scraping is the process of extracting data from the website using automated tools to make the process faster.

In [13]:
```
!pip install requests
!pip install bs4
```

```
Requirement already satisfied: requests in c:\users\shiva\.conda\lib\site-pac
kages (2.28.1)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\shiva\.co
nda\lib\site-packages (from requests) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\shiva\.conda
\lib\site-packages (from requests) (1.26.14)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\shiva\.conda\li
b\site-packages (from requests) (2022.12.7)
Requirement already satisfied: idna<4,>=2.5 in c:\users\shiva\.conda\lib\site
-packages (from requests) (3.4)
Requirement already satisfied: bs4 in c:\users\shiva\.conda\lib\site-packages
(0.0.1)
Requirement already satisfied: beautifulsoup4 in c:\users\shiva\.conda\lib\si
te-packages (from bs4) (4.11.1)
Requirement already satisfied: soupsieve>1.2 in c:\users\shiva\.conda\lib\sit
e-packages (from beautifulsoup4->bs4) (2.3.2.post1)
```

Step 1: Import all the required libraries.

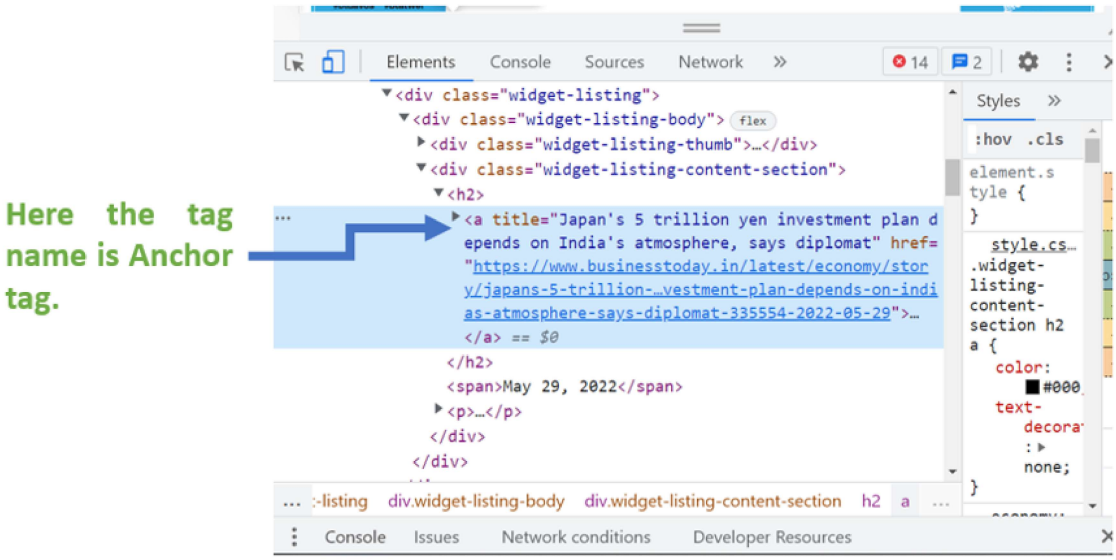
In [14]:
```
from bs4 import BeautifulSoup as BS
import requests as req
```

Step 2: Find the best website for finance news to get daily updates seamlessly.

https://www.businesstoday.in/latest/economy (https://www.businesstoday.in/latest/economy)

Step 3: Inspect the tag in which news content is stored with the help of inspecting the HTML code.

Step 4: Now we will check the tag name and use that name in our code, i.e. Here, an anchor tag is used so we will use 'a' in our code.



Step 6: To Fetch the news-related material we need only “bs4.element.NavigableString” class.

Step 7: Set the limit of the news character length.

In [15]:
```
url = "https://www.businesstoday.in/latest/economy"

webpage = req.get(url)


trav = BS(webpage.content, "html.parser")


for link in trav.find_all('a'):
    print(type(link.string), " ", link.string)
```

```
<class 'bs4.element.NavigableString'>   Business Today
<class 'bs4.element.NavigableString'>   BT Bazaar
<class 'bs4.element.NavigableString'>   India Today
<class 'bs4.element.NavigableString'>   Northeast
<class 'bs4.element.NavigableString'>   Web3Cafe
<class 'bs4.element.NavigableString'>   DailyO
<class 'bs4.element.NavigableString'>   India Today Gaming
<class 'bs4.element.NavigableString'>   Cosmopolitan
<class 'bs4.element.NavigableString'>   Harper's Bazaar
<class 'bs4.element.NavigableString'>   Brides Today
<class 'bs4.element.NavigableString'>   Ishq FM
<class 'bs4.element.NavigableString'>   Aaj Tak
<class 'bs4.element.NavigableString'>   GNTTV
<class 'bs4.element.NavigableString'>   iChowk
<class 'bs4.element.NavigableString'>   Kisan Tak
<class 'bs4.element.NavigableString'>   Lallantop
<class 'bs4.element.NavigableString'>   Malyalam
<class 'bs4.element.NavigableString'>   Bangla
<class 'bs4.element.NavigableString'>   Sports Tak
```

In [16]:
```
# All code

# IMPORT ALL THE REQUIRED LIBRARIES
from bs4 import BeautifulSoup as BS
import requests as req

url = "https://www.businesstoday.in/latest/economy"

webpage = req.get(url)
trav = BS(webpage.content, "html.parser")
news_no = 1
for link in trav.find_all('a'):
    if(str(type(link.string)) == "<class 'bs4.element.NavigableString'>"
       and len(link.string) > 5):
        print(str(news_no)+".", link.string)
        news_no += 1
```

```
1. Business Today
2. BT Bazaar
3. India Today
4. Northeast
5. Web3Cafe
6. DailyO
7. India Today Gaming
8. Cosmopolitan
9. Harper's Bazaar
10. Brides Today
11. Ishq FM
12. Aaj Tak
13. iChowk
14. Kisan Tak
15. Lallantop
16. Malyalam
17. Bangla
18. Sports Tak
19. Crime Tak
```

Wordcloud of news

In [5]:
```
# install packages
!pip install wordcloud
```

```
Collecting wordcloud
  Downloading wordcloud-1.9.2-cp310-cp310-win_amd64.whl (152 kB)
     ---------------------------------------- 152.1/152.1 kB 1.8 MB/s eta 0:00:
00
Requirement already satisfied: numpy>=1.6.1 in c:\users\shiva\.conda\lib\site
-packages (from wordcloud) (1.23.5)
Requirement already satisfied: matplotlib in c:\users\shiva\.conda\lib\site-p
ackages (from wordcloud) (3.7.0)
Requirement already satisfied: pillow in c:\users\shiva\.conda\lib\site-packa
ges (from wordcloud) (9.4.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\shiva\.conda\lib
\site-packages (from matplotlib->wordcloud) (1.4.4)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\shiva\.conda
\lib\site-packages (from matplotlib->wordcloud) (2.8.2)
Requirement already satisfied: cycler>=0.10 in c:\users\shiva\.conda\lib\site
-packages (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\shiva\.conda\lib
\site-packages (from matplotlib->wordcloud) (1.0.5)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\shiva\.conda\lib
\site-packages (from matplotlib->wordcloud) (4.25.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\shiva\.conda\lib
\site-packages (from matplotlib->wordcloud) (3.0.9)
Requirement already satisfied: packaging>=20.0 in c:\users\shiva\.conda\lib\s
ite-packages (from matplotlib->wordcloud) (22.0)
Requirement already satisfied: six>=1.5 in c:\users\shiva\.conda\lib\site-pac
kages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.16.0)
Installing collected packages: wordcloud
Successfully installed wordcloud-1.9.2
```

In [17]:
```
# Start with loading all necessary libraries
import numpy as np
import pandas as pd
from os import path
import matplotlib.pyplot as plt
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

Store all news text in a single string

In [18]:
```
# IMPORT ALL THE REQUIRED LIBRARIES
from bs4 import BeautifulSoup as BS
import requests as req

url = "https://www.businesstoday.in/latest/economy"

webpage = req.get(url)
trav = BS(webpage.content, "html.parser")
news_text = ''
for link in trav.find_all('a'):
    if(str(type(link.string)) == "<class 'bs4.element.NavigableString'>" and l
        news_text += link.string + '\n'

# Create and generate a word cloud image:
wordcloud = WordCloud().generate(news_text)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Changing optional word cloud arguments

Now, change some optional arguments of the word cloud like max_font_size, max_word, and background_color.

In [19]:
```
# lower max_font_size, change the maximum number of word and lighten the backg
wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="white
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

We can use the process_text() and words_ methods to display the word count and relative counts from the text respectively.

Source: https://towardsdatascience.com/generate-meaningful-word-clouds-in-python-5b85f5668eeb (https://towardsdatascience.com/generate-meaningful-word-clouds-in-python-5b85f5668eeb)

In [20]:
```
# create a dictionary of word frequencies
text_dictionary = wordcloud.process_text(news_text.upper())
# sort the dictionary
word_freq={k: v for k, v in sorted(text_dictionary.items(),reverse=True, key=l

# use words_ to print relative word frequencies
rel_freq=wordcloud.words_

# print top 5 results
print(list(word_freq.items())[:5])
print(list(rel_freq.items())[:5])

# print all results
word_pd = pd.DataFrame(word_freq.items())
word_pd.to_csv('news_words.csv')
word_pd
```

```
[('TODAY', 14), ('BT', 9), ('MARKET', 6), ('STOCK', 6), ('TAK', 5)]
[('Today', 1.0), ('BT', 0.6428571428571429), ('Market', 0.42857142857142855),
('Stock', 0.42857142857142855), ('Tak', 0.35714285714285715)]
```

Out[20]:

	0	1
0	TODAY	14
1	BT	9
2	MARKET	6
3	STOCK	6
4	TAK	5
...	...	...
267	CONDITIONS	1
268	PARTNERS	1
269	PRESS	1
270	ADD	1
271	SCREEN	1

272 rows × 2 columns

Assignment

Now it is your turn. Pick any news agency or a web page that has text or titles similar to the one above. Scrape text data from that web site. Generate wordcloud and word frequencies. Then save the words and their frequencies in a csv file with your name and the title of the webpage.

You may add any number of cells below by pressing the + sign or Insert menu.

I added one cell for you. Feel free to add more

In [38]:
```
# Import the required libraries
import requests as req
from bs4 import BeautifulSoup as BS
url = "https://www.halloween.com/"
webpage = req.get(url)
trav = BS(webpage.content, "html.parser")
for link in trav.find_all('a'):
 print(type(link.string), " ", link.string)
```

```
<class 'bs4.element.NavigableString'>   View All Accesssories
<class 'bs4.element.NavigableString'>   Accessory Kits
<class 'bs4.element.NavigableString'>   Beards / Mustaches
<class 'bs4.element.NavigableString'>   Boots / Shoes
<class 'bs4.element.NavigableString'>   Capes / Cloaks
<class 'bs4.element.NavigableString'>   Crowns / Tiaras
<class 'bs4.element.NavigableString'>   Fake Blood
<class 'bs4.element.NavigableString'>   Fangs & Teeth
<class 'bs4.element.NavigableString'>   Glasses
<class 'bs4.element.NavigableString'>   Gloves
<class 'bs4.element.NavigableString'>   Hats
<class 'bs4.element.NavigableString'>   Headbands
<class 'bs4.element.NavigableString'>   Helmets
<class 'bs4.element.NavigableString'>   Jewelry
<class 'bs4.element.NavigableString'>   Makeup
<class 'bs4.element.NavigableString'>   Masks
<class 'bs4.element.NavigableString'>   Noses / Snouts
<class 'bs4.element.NavigableString'>   Petticoats
<class 'bs4.element.NavigableString'>   Prosthetics
<class 'bs4.element.NavigableString'>   Purses
```

In [39]:
```
# IMPORT ALL THE REQUIRED LIBRARIES
from bs4 import BeautifulSoup as BS
import requests as req

url = "https://www.halloween.com/"

webpage = req.get(url)
trav = BS(webpage.content, "html.parser")
news_no = 1
for link in trav.find_all('a'):
        if(str(type(link.string)) == "<class 'bs4.element.NavigableString'>"
           and len(link.string) > 5):
            print(str(news_no)+".", link.string)
            news_no += 1
```

```
1. Skip to main content
2. View All Adult Costumes
3. View All Adult Costumes
4. Men's Costumes
5. Women's Costumes
6. View All Plus Size Costumes
7. Men's Plus Size Costumes
8. Women's Plus Size Costumes
9. View All Sexy Costumes
10. Sexy Plus Size Costumes
11. View All Kids Costumes
12. View All Kids Costumes
13. Baby Costumes
14. Boy Costumes
15. Girl Costumes
16. Teen Costumes
17. Toddler Costumes
18. View All Halloween Ideas
19. Alice In Wonderland
```

In [40]:
```
# Send an HTTP request and parse the HTML content
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
```

In [41]:
```
# Extract text data from the webpage
news_text = ""
for link in soup.find_all('a'):
    if link.string:
        news_text += link.string + '\n'
```

In [42]:
```
# Generate a word cloud
wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="white
wordcloud.generate(news_text)
# Display the word cloud
import matplotlib.pyplot as plt
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

In [36]:
```
# Count word frequencies
words = news_text.split()
word_freq = {}
for word in words:
    if word in word_freq:
        word_freq[word] += 1
    else:
        word_freq[word] = 1
```

In [37]:
```
# Save word frequencies to a CSV file
df = pd.DataFrame(list(word_freq.items()), columns=["Word", "Frequency"])
df.to_csv("word_frequencies.csv", index=False)

print("shivaniwordfrequency.csv")
```

```
shivaniwordfrequency.csv
```

Example 3 Scrape Top 100 Movie List From IMDB

Source: https://www.stratascratch.com/blog/4-data-collection-libraries-in-python-that-you-should-know/ (https://www.stratascratch.com/blog/4-data-collection-libraries-in-python-that-you-should-know/)

There are 50 records in each page so collect them separately

In [43]:
```
# First 50 movies
import requests
from bs4 import BeautifulSoup

movie_names = []
ratings = []

# Set the URL to scrape first 50
url = "https://www.imdb.com/search/title/?groups=top_100&sort=user_rating,desc

# from 51 to 100
# url = "https://www.imdb.com/search/title/?groups=top_100&sort=user_rating,de

page = requests.get(url)

# Parse the HTML content of the page
soup = BeautifulSoup(page.text, 'html.parser')

# Find all the <div> elements with the class "lister-item mode-advanced"
movies = soup.find_all('div', class_='lister-item mode-advanced')

# For each movie, extract the title and rating
for movie in movies:

    title = movie.h3.a.text
    movie_names.append(title)

    rating = movie.strong.text
    ratings.append(rating)

movie_dic = {"Movie Name" : movie_names, "Rating" : ratings}
df = pd.DataFrame(movie_dic)
df
```

Out[43]:

	Movie Name	Rating
0	The Shawshank Redemption	9.3
1	The Godfather	9.2
2	The Dark Knight	9.0
3	Schindler's List	9.0
4	The Lord of the Rings: The Return of the King	9.0
5	12 Angry Men	9.0
6	The Godfather Part II	9.0
7	Pulp Fiction	8.9
8	Fight Club	8.8
9	Inception	8.8
10	The Lord of the Rings: The Fellowship of the Ring	8.8
11	Forrest Gump	8.8
12	The Lord of the Rings: The Two Towers	8.8
13	The Good, the Bad and the Ugly	8.8
14	Spider-Man: Across the Spider-Verse	8.7
15	Interstellar	8.7
16	Goodfellas	8.7
17	The Matrix	8.7
18	One Flew Over the Cuckoo's Nest	8.7
19	Star Wars: Episode V - The Empire Strikes Back	8.7
20	The Silence of the Lambs	8.6
21	Se7en	8.6
22	The Green Mile	8.6
23	Saving Private Ryan	8.6
24	Star Wars: Episode IV - A New Hope	8.6
25	Spirited Away	8.6
26	Terminator 2: Judgment Day	8.6
27	City of God	8.6
28	Life Is Beautiful	8.6
29	It's a Wonderful Life	8.6
30	Seven Samurai	8.6
31	Harakiri	8.6
32	Oppenheimer	8.5
33	Gladiator	8.5
34	The Departed	8.5
35	Alien	8.5

	Movie Name	Rating
36	Parasite	8.5
37	The Prestige	8.5
38	Psycho	8.5
39	Whiplash	8.5
40	Back to the Future	8.5
41	Django Unchained	8.5
42	Léon: The Professional	8.5
43	The Lion King	8.5
44	The Usual Suspects	8.5
45	The Intouchables	8.5
46	American History X	8.5
47	The Pianist	8.5
48	Casablanca	8.5
49	Once Upon a Time in the West	8.5

In [44]:
```
# All 100 movies
import requests
from bs4 import BeautifulSoup

movie_names = []
ratings = []

# Set the URL to scrape first 50
url_1_50 = "https://www.imdb.com/search/title/?groups=top_100&sort=user_rating

# from 51 to 100
url_51_100 = "https://www.imdb.com/search/title/?groups=top_100&sort=user_rati

page = requests.get(url_1_50)

# Parse the HTML content of the page
soup = BeautifulSoup(page.text, 'html.parser')

# Find all the <div> elements with the class "lister-item mode-advanced"
movies = soup.find_all('div', class_='lister-item mode-advanced')

# For each movie, extract the title and rating
for movie in movies:

    title = movie.h3.a.text
    movie_names.append(title)

    rating = movie.strong.text
    ratings.append(rating)


page = requests.get(url_51_100)

# Parse the HTML content of the page
soup = BeautifulSoup(page.text, 'html.parser')

# Find all the <div> elements with the class "lister-item mode-advanced"
movies = soup.find_all('div', class_='lister-item mode-advanced')

# For each movie, extract the title and rating
for movie in movies:

    title = movie.h3.a.text
    movie_names.append(title)

    rating = movie.strong.text
    ratings.append(rating)

movie_dic = {"Movie Name" : movie_names, "Rating" : ratings}
df = pd.DataFrame(movie_dic)
df
```

Out[44]:

	Movie Name	Rating
0	The Shawshank Redemption	9.3
1	The Godfather	9.2
2	The Dark Knight	9.0
3	Schindler's List	9.0
4	The Lord of the Rings: The Return of the King	9.0
5	12 Angry Men	9.0
6	The Godfather Part II	9.0
7	Pulp Fiction	8.9
8	Fight Club	8.8
9	Inception	8.8
10	The Lord of the Rings: The Fellowship of the Ring	8.8
11	Forrest Gump	8.8

In [45]:
```
# Save the movies and ratings in a csv file
df.to_csv('movies_ratings_100.csv')
```

Scrape a given number of movies and ratings

In [46]:
```
# Determine the number of movies by a variable
# Not all values can be used
import requests
from bs4 import BeautifulSoup

movie_names = []
ratings = []
how_many = 1000

# Set the URL to scrape first 50
first_url = "https://www.imdb.com/search/title/?groups=top_" + str(how_many)+"
page_no = 1
next_page = ""

while page_no < how_many:
    current_page = first_url + next_page
    print(current_page)
    page = requests.get(current_page)

    # Parse the HTML content of the page
    soup = BeautifulSoup(page.text, 'html.parser')

    # Find all the <div> elements with the class "lister-item mode-advanced"
    movies = soup.find_all('div', class_='lister-item mode-advanced')

    # For each movie, extract the title and rating
    for movie in movies:

        title = movie.h3.a.text
        movie_names.append(title)

        rating = movie.strong.text
        ratings.append(rating)

    page_no += 50
    next_page = "&start=" + str(page_no) + "&ref_=adv_nxt"


movie_dic = {"Movie Name" : movie_names, "Rating" : ratings}
df = pd.DataFrame(movie_dic)
df
```

```
tart=101&ref_=adv_nxt (https://www.imdb.com/search/title/?groups=top_1000&
sort=user_rating,desc&start=101&ref_=adv_nxt)
https://www.imdb.com/search/title/?groups=top_1000&sort=user_rating,desc&s
tart=151&ref_=adv_nxt (https://www.imdb.com/search/title/?groups=top_1000&
sort=user_rating,desc&start=151&ref_=adv_nxt)
https://www.imdb.com/search/title/?groups=top_1000&sort=user_rating,desc&s
tart=201&ref_=adv_nxt (https://www.imdb.com/search/title/?groups=top_1000&
sort=user_rating,desc&start=201&ref_=adv_nxt)
https://www.imdb.com/search/title/?groups=top_1000&sort=user_rating,desc&s
tart=251&ref_=adv_nxt (https://www.imdb.com/search/title/?groups=top_1000&
sort=user_rating,desc&start=251&ref_=adv_nxt)
https://www.imdb.com/search/title/?groups=top_1000&sort=user_rating,desc&s
tart=301&ref_=adv_nxt (https://www.imdb.com/search/title/?groups=top_1000&
sort=user_rating,desc&start=301&ref_=adv_nxt)
https://www.imdb.com/search/title/?groups=top_1000&sort=user_rating,desc&s
tart=351&ref_=adv_nxt (https://www.imdb.com/search/title/?groups=top_1000&
sort=user_rating,desc&start=351&ref_=adv_nxt)
https://www.imdb.com/search/title/?groups=top_1000&sort=user_rating,desc&s
tart=401&ref_=adv_nxt (https://www.imdb.com/search/title/?groups=top_1000&
sort=user_rating,desc&start=401&ref_=adv_nxt)
```

In [47]:
```
# Save the movies and ratings in a csv file
df.to_csv('movies_ratings_1000.csv')
```

Assignment

Now it is your turn. Pick any public web page that has ratings similar to the one above. Scrape text data from that web site. Generate a dictionary with the ratings. Then save the items and their ratings in a csv file with your name and the title of the webpage.

You may add any number of cells below by pressing the + sign or Insert menu.

I added one cell for you. Feel free to add more

This code link of IMDB movies in 2023

below code scrape first page

In [69]:
```
import requests
from bs4 import BeautifulSoup
import pandas as pd

movie_names = []
ratings = []

# Set the URL to scrape the IMDb list (first page)
url = "https://www.imdb.com/list/ls562300956/?sort=list_order,asc&st_dt=&mode=

# Send an HTTP GET request to the URL
page = requests.get(url)

# Parse the HTML content of the page
soup = BeautifulSoup(page.text, 'html.parser')

# Find all the movie items on the page
movies = soup.find_all('div', class_='lister-item')

# For each movie, extract the title and rating
for movie in movies:
    title = movie.find('h3', class_='lister-item-header').a.text
    rating_element = movie.find('span', class_='ipl-rating-star__rating')
    rating = rating_element.text if rating_element else "N/A"
    movie_names.append(title)
    ratings.append(rating)

# Create a DataFrame
movie_data = {"Movie Name": movie_names, "Rating": ratings}
df = pd.DataFrame(movie_data)

# Print the DataFrame
print(df)
```

	Movie Name	Rating
0	Are You There God? It's Me, Margaret.	7.4
1	Evil Dead Rise	6.6
2	The Super Mario Bros. Movie	7.1
3	The Covenant	7.5
4	Tetris	7.4
5	A Good Person	7
6	Flamin' Hot	6.9
7	Infinity Pool	6.1
8	Champions	6.8
9	Ant-Man and the Wasp: Quantumania	6.1
10	Dungeons & Dragons: Honor Among Thieves	7.3
11	BlackBerry	7.4
12	Renfield	6.4
13	Somewhere in Queens	6.8
14	Extraction II	7
15	Sisu	6.9
16	Knock at the Cabin	6.1
17	Big George Foreman	6.6
18	Jesus Revolution	7.1
19	Missing	7.1
20	Beau Is Afraid	6.8
21	Ponniyin Selvan: Part Two	7.3
22	Stan Lee	6.9
23	Air	7.4
24	John Wick: Chapter 4	7.8
25	Somebody I Used to Know	5.7
26	A Man Called Otto	7.5
27	Shazam! Fury of the Gods	6
28	The Pope's Exorcist	6.1
29	Scream VI	6.5
30	M3GAN	6.4
31	Plane	6.5
32	Creed III	6.8
33	Operation Fortune: Ruse de Guerre	6.3
34	You People	5.5
35	Magic Mike's Last Dance	5.2
36	To Catch a Killer	6.6
37	True Spirit	6.8
38	Sharper	6.7
39	Boston Strangler	6.5
40	Pinball: The Man Who Saved the Game	6.9
41	Luther: The Fallen Sun	6.4
42	The Last Kingdom: Seven Kings Must Die	6.9
43	Polite Society	6.7
44	Fast X	5.8
45	Daliland	5.9
46	Chevalier	6.5
47	Kandahar	6
48	Ghosted	5.8
49	Rye Lane	7.2
50	How to Blow Up a Pipeline	7
51	Reality	6.7
52	Nefarious	6.4
53	What's Love Got to Do with It?	6.3
54	The Artifice Girl	6.6
55	Wildflower	6.7

56	Linoleum	6.5
57	Cocaine Bear	5.9
58	Murder Mystery 2	5.7
59	80 for Brady	5.8
60	Mafia Mamma	5.4
61	Your Place or Mine	5.7
62	Paint	4.9
63	Blue Jean	7
64	We Have a Ghost	6.1
65	The Mother	5.6
66	Hypnotic	5.5
67	Legion of Super-Heroes	5.8
68	Batman: The Doom That Came to Gotham	6.1
69	The Magician's Elephant	6.5
70	Teen Wolf: The Movie	5.6
71	When You Finish Saving the World	5.6
72	Dog Gone	6
73	Inside	5.5
74	Mummies	5.9
75	Prom Pact	6.3
76	Chupa	5.5
77	Cocaine Bear: The True Story	5.6
78	Mighty Morphin Power Rangers: Once & Always	5.7
79	Justice League x RWBY: Super Heroes and Huntsm...	5.3
80	Sweetwater	6
81	Love Again	5.9
82	Spinning Gold	6
83	Ride On	6.3
84	About My Father	5.7
85	On Sacred Ground	5.6
86	On a Wing and a Prayer	5.5
87	Robots	5.5
88	The Wedding Veil Journey	7.2
89	The Wedding Veil Inspiration	7
90	The Wedding Veil Expectations	6.7
91	At Midnight	5.6
92	The Magic Flute	5.6
93	65	5.4
94	Shotgun Wedding	5.4
95	Marlowe	5.3
96	Seriously Red	5.6
97	Blood	5.4
98	The Old Way	5.5
99	Jung_E	5.5

2nd page scrape

In [71]:
```
import requests
from bs4 import BeautifulSoup
import pandas as pd

movie_names = []
ratings = []

# Set the URL to scrape the second page of the IMDb list
url = "https://www.imdb.com/list/ls562300956/?sort=list_order,asc&st_dt=&mode=

# Send an HTTP GET request to the URL
page = requests.get(url)

# Parse the HTML content of the page
soup = BeautifulSoup(page.text, 'html.parser')

# Find all the movie items on the page
movies = soup.find_all('div', class_='lister-item')

# For each movie, extract the title and rating
for movie in movies:
    title = movie.find('h3', class_='lister-item-header').a.text
    rating_element = movie.find('span', class_='ipl-rating-star__rating')
    rating = rating_element.text if rating_element else "N/A"
    movie_names.append(title)
    ratings.append(rating)

# Create a DataFrame
movie_data = {"Movie Name": movie_names, "Rating": ratings}
df = pd.DataFrame(movie_data)

# Print the DataFrame
print(df)
```

	Movie Name	Rating
0	Die Hart	5.3
1	Space Oddity	5.4
2	Dead Shot	5.4
3	Transfusion	5
4	One Day as a Lion	5.2
5	Clock	5
6	Invitation to a Murder	5.1
7	Last Sentinel	5
8	Simulant	5.1
9	Crater	5.3
10	Book Club: The Next Chapter	5.6
11	Maybe I Do	4.9
12	Unwelcome	5.3
13	There's Something Wrong with the Children	5
14	The Price We Pay	4.9
15	Baby Ruby	4.8
16	The Donor Party	4.6
17	The Tutor	4.8
18	Peter Pan & Wendy	4.4
19	One Ranger	5.2
20	White Men Can't Jump	5.1
21	Fool's Paradise	4.7
22	House Party	4.4
23	97 Minutes	3.5
24	The Best Man	3.8
25	The Black Demon	3.7
26	Night Train	3.7
27	Winnie the Pooh: Blood and Honey	2.9
28	Snow Falls	2.9
29	The Machine	5.8
30	Maggie Moore(s)	6.1
31	The Wrath of Becky	6.1
32	Maximum Truth	4.6
33	You Hurt My Feelings	6.6
34	The Civil Dead	6.2
35	Prisoner's Daughter	6.3
36	Asteroid City	6.6
37	The Blackening	6
38	Carmen	5.9
39	Guardians of the Galaxy Vol. 3	7.9
40	The Out-Laws	5.4
41	The Starling Girl	6.7
42	Transformers: Rise of the Beasts	6.1
43	Bird Box: Barcelona	5.3
44	God Is a Bullet	5.6
45	The Flood	3.2
46	Ruby Gillman, Teenage Kraken	5.7
47	The Flash	6.7
48	They Cloned Tyrone	6.6
49	About him & her	5.7
50	The Beanie Bubble	6.3
51	Hidden Strike	5.3
52	Insidious: The Red Door	5.5
53	Joy Ride	6.5
54	Justice League: Warworld	5.3
55	Little Bone Lodge	6.1

56	The Little Mermaid	7.2
57	Sympathy for the Devil	5.5
58	The Unlikely Pilgrimage of Harold Fry	6.8
59	The Venture Bros.: Radiant Is the Blood of the...	8
60	War Pony	6.9
61	River Wild	5.4
62	The Childe	6.9
63	Devil's Peak	5.1
64	The Collective	3.1
65	Corner Office	6
66	The Passenger	6.3
67	What Comes Around	5.2
68	The Three Musketeers - Part I: D'Artagnan	6.7
69	Spider-Man: Across the Spider-Verse	8.7
70	See You on Venus	5.8
71	No Hard Feelings	6.4
72	Double Life	5
73	The Adults	5.8
74	Brother	7
75	Cobweb	5.9
76	Heart of Stone	5.7
77	Match Me If You Can	5.6
78	Red, White & Royal Blue	7
79	Wonderwell	4.5
80	Elemental	7
81	Miguel Wants to Fight	5.7
82	Past Lives	8
83	Vacation Friends 2	5.3
84	Puppy Love	6.3
85	The Boogeyman	5.9
86	Indiana Jones and the Dial of Destiny	6.7
87	The Pod Generation	5.5
88	The Last Voyage of the Demeter	6.1
89	Meg 2: The Trench	5.1
90	The Monkey King	5.8
91	Seneca	5
92	Teenage Mutant Ninja Turtles: Mutant Mayhem	7.3
93	Strays	6.3
94	The Portable Door	6.1
95	Barbie	7
96	Landscape with Invisible Hand	6
97	Shortcomings	6.5
98	Talk to Me	7.2
99	Theater Camp	7.1

In [72]:
```
import requests
from bs4 import BeautifulSoup
import pandas as pd

movie_names = []
ratings = []

# Set the URL to scrape the IMDb list
url = "https://www.imdb.com/list/ls562300956/?sort=list_order,asc&st_dt=&mode=

# Send an HTTP GET request to the URL
page = requests.get(url)

# Parse the HTML content of the page
soup = BeautifulSoup(page.text, 'html.parser')

# Find all the movie items on the page
movies = soup.find_all('div', class_='lister-item')

# For each movie, extract the title and rating
for movie in movies:
    title = movie.find('h3', class_='lister-item-header').a.text
    # IMDb has likely changed its HTML structure for ratings; you'll need to i
    # Assuming the rating is within a span with class 'ipl-rating-star__rating
    rating_element = movie.find('span', class_='ipl-rating-star__rating')
    rating = rating_element.text if rating_element else "N/A"
    movie_names.append(title)
    ratings.append(rating)

# Pagination: Loop through pages
for page in range(1, 5):  # Change the range to the number of pages you want t
    next_page_url = f"https://www.imdb.com/list/ls562300956/?sort=list_order,a
    next_page = requests.get(next_page_url)
    next_soup = BeautifulSoup(next_page.text, 'html.parser')
    next_movies = next_soup.find_all('div', class_='lister-item')

    for movie in next_movies:
        title = movie.find('h3', class_='lister-item-header').a.text
        rating_element = movie.find('span', class_='ipl-rating-star__rating')
        rating = rating_element.text if rating_element else "N/A"
        movie_names.append(title)
        ratings.append(rating)

# Create a DataFrame
movie_data = {"Movie Name": movie_names, "Rating": ratings}
df = pd.DataFrame(movie_data)

# Print the DataFrame
df.head(200)
```

Out[72]:

	Movie Name	Rating
0	Are You There God? It's Me, Margaret.	7.4
1	Evil Dead Rise	6.6
2	The Super Mario Bros. Movie	7.1
3	The Covenant	7.5
4	Tetris	7.4
5	A Good Person	7
6	Flamin' Hot	6.9
7	Infinity Pool	6.1
8	Champions	6.8
9	Ant-Man and the Wasp: Quantumania	6.1
10	Dungeons & Dragons: Honor Among Thieves	7.3
11	BlackBerry	7.4

In [74]:
```
import requests
from bs4 import BeautifulSoup
import pandas as pd

movie_names = []
ratings = []
how_many = 1000

# Set the URL to scrape first 50
first_url = "https://www.imdb.com/list/ls562300956/?sort=list_order,asc&st_dt=
page_no = 1
next_page = ""

while page_no < how_many:
    current_page = first_url + next_page
    print(current_page)
    page = requests.get(current_page)

    # Parse the HTML content of the page
    soup = BeautifulSoup(page.text, 'html.parser')

    # Find all the <div> elements with the class "lister-item mode-advanced"
    movies = soup.find_all('div', class_='lister-item')

    # For each movie, extract the title and rating
    for movie in movies:
        title = movie.find('h3', class_='lister-item-header').a.text
        rating = movie.find('span', class_='ipl-rating-star__rating').text
        movie_names.append(title)
        ratings.append(rating)

    page_no += 50
    next_page = f"&page={page_no}"

movie_dic = {"Movie Name": movie_names, "Rating": ratings}
df = pd.DataFrame(movie_dic)
df
```

59	80 for Brady	5.8
60	Mafia Mamma	5.4
61	Your Place or Mine	5.7
62	Paint	4.9
63	Blue Jean	7
64	We Have a Ghost	6.1
65	The Mother	5.6
66	Hypnotic	5.5
67	Legion of Super-Heroes	5.8
68	Batman: The Doom That Came to Gotham	6.1
69	The Magician's Elephant	6.5
70	Teen Wolf: The Movie	5.6
71	When You Finish Saving the World	5.6

In [76]:
```
# Save the movies and ratings in a csv file
df.to_csv('sreekethana_imbd_2023.csv')
# Create a download link for the CSV file
from IPython.display import FileLink

FileLink('sreekethana_imbd_2023.csv')
```

Out[76]: sreekethana_imbd_2023.csv (sreekethana_imbd_2023.csv)

Other website

In [13]:
```
import requests
from bs4 import BeautifulSoup
movie_names = []
ratings = []
url = "import requests"
from bs4 import BeautifulSoup
movie_names = []
ratings = []
url = "https://editorial.rottentomatoes.com/guide/best-movies-of-2023/"
# Send an HTTP GET request to the URL
response = requests.get(url)
# Parse the HTML content of the page
soup = BeautifulSoup(response.text, 'html.parser')
# Find the elements that contain movie data
movie_elements = soup.find_all("div", class_="article_movie_title")
# Extract movie names and ratings
for movie_element in movie_elements:
    title = movie_element.a.text.strip()
    rating = movie_element.find("span", class_="tMeterScore").text.strip()
    movie_names.append(title)
    ratings.append(rating)
# Print the movie names and ratings
movie_dic = {"Movie Name" : movie_names, "Letterboxd_rating" : ratings}
df = pd.DataFrame(movies)
df
# Send an HTTP GET request to the URL
response = requests.get(url)
# Parse the HTML content of the page
soup = BeautifulSoup(response.text, 'html.parser')
# Find the elements that contain movie data
movie_elements = soup.find_all("div", class_="article_movie_title")
# Extract movie names and ratings
for movie_element in movie_elements:
 title = movie_element.a.text.strip()
 rating = movie_element.find("span", class_="tMeterScore").text.strip()
 movie_names.append(title)
 ratings.append(rating)
# Print the movie names and ratings
movies= {"Movie Name" : movie_names, "Letterboxd_rating" : ratings}
df = pd.DataFrame(movies)
print(df)
```

```
                              Movie Name Letterboxd_rating
0                       The First Slam Dunk             100%
1    Are You There God? It's Me, Margaret.              99%
2                               Past Lives              98%
3                               BlackBerry              98%
4                                 Rye Lane              98%
..                                     ...              ...
251                            Moving On              75%
252                            Elemental              74%
253                          The Offering             74%
254                   Strange Way of Life             74%
255                         Medusa Deluxe              72%

[256 rows x 2 columns]
```

In [14]:
```
# Save the movies and ratings in a csv file
df.to_csv('sreekethana_rotten_2023.csv')
# Create a download link for the CSV file
from IPython.display import FileLink

FileLink('sreekethana_rotten_2023.csv')
```

Out[14]: sreekethana_rotten_2023.csv (sreekethana_rotten_2023.csv)

In [ ]: