```python
# Adam Zuckerman
"""
This web scraper crawls marketwatch.com by ticker and iterates over the financial
statements.
It then downloads the statements, relevant pricing data, and organizes the data
under new directories by the industry you selected to analyze.
The financial statements are of a 5 year history. Other relevant data downloaded are
financial ratios, margin percentages, current price/volume,
a list of the company's top executives, and their biography. All tickers fed into
the web scraper are selected by the highest market cap of their industry.

For the program to work the csv file containing the lists of tickers per top 25
companies in their respective industry must be used.
CSV file at
https://github.com/AMZ58/market_watch_scraper/tree/main/csvoftickers_list
"""
import urllib.request as rq
from bs4 import BeautifulSoup
from datetime import datetime
import requests
import time
import pandas as pd
import os
import gc
import numpy as np
from pathlib import Path
from collections import Counter, OrderedDict

u = 'https://www.marketwatch.com/investing/stock/'

"Description of company's niche "
company_description = []

"Officer Names and data"
name = []
officer_title = []
bio = []
df_name = "officer_dataframe"

"single values for "
price_volume_data = []
df1_name = "price_volume_data"

ticker_used = []
"""Table Names for pickle file"""
i_t = 'income_statement'
b_t = 'balance_sheet'
c_t = 'cash_flow'
v_t = 'value_table'
e_t = 'efficiency_table'
p_t = 'profitablity_table'

"Current Time"
time_stamp = []

"Officer ID number for url"
pid_officer = []
```

```
def remove_tag(input, output):
    """ This function is used to strip the html tags from the extract text so that
the data can be processed and stored properly """
    for a in input:
        u_wrap = a.text.strip()
        output.append(u_wrap)


def find_officers_pid(s):
    """
    Variable s passes the parsed html into the function so that we can extract the
child 'a' tags from the parent 'div' tag
    PersonId found in the html 'a' tag which stores the unique id of the company
officers.
    This id is then stored globally into a array to be used by the function
find_officer_data
    """
    officer_scrape = s.find_all('div', attrs={"class": "element element--list"})
    for content in officer_scrape:
        var1 = content.find_all("a")
        for content in var1:
            id = content.get("personid")
            pid_officer.append(id)


def find_officer_data(a):
    """
    From the officers PID we then extract his/her name, biography, and title
    This is then appending to a global array is used to create the dataframes for
the excel file later
    """
    officer_data = a.find_all('div', attrs={"class": "element element--text
biography"})
    for i in officer_data:
        heading = i.find_all("h4")
        title = i.find_all("h6")
        paragraph = [a.text.strip() for a in i.find_all('p')]
        remove_tag(heading, name)
        remove_tag(title, officer_title)
        bio.append(paragraph)


def find_stock_range(r):
    """
    Here we are searching for the timestamp, current volume, high/low of todays
price, and the yearly high/low price.
    As the html tag containing the data changed between tickers I wrote a try/catch
statement to catch an error if the tag is not found and to search under the span
tag.
    The collected data is then appended to the global array to be used later to
create a pandas data frame.
    """
    intraday_data = r.find_all('div', attrs={"class": "region region--intraday"})

    for i in intraday_data:

        t_stamp = i.find('bg-quote', attrs={'field': 'date'}).text
        try:
            cur_p = i.find('h2', attrs={"class": "intraday__price"}).find('bg-
quote', attrs={"class": "value"}).text
            cur_c = i.find('table', attrs={"class": "table table--primary align--
right"}).find('td', attrs={"class": "table__cell u-semi"}).text.strip("$")
            price_volume_data.append(cur_p)
            price_volume_data.append(cur_c)
        except AttributeError as attr_error:
            print("Issue with current price tag")
            print(attr_error)
            try:
                cur_p = i.find('h2', attrs={"class":
"intraday__price"}).find('span', attrs={"class": "value"}).text
                price_volume_data.append(cur_p)
                print("Found it !")
            except AttributeError as attr_error:
                print("No dice buddy")
                print(attr_error)

        try:
            after_v = i.find('div', attrs={"class": "intraday__volume"}).find('bg-
quote',attrs={"field": "volume"}).text.strip(
                "Volume: M")
            price_volume_data.append(after_v)

        except AttributeError as attr_error:
            print("Issue with after hours volume tag")
            print(attr_error)
            try:
                after_v = i.find('mw-rangebar', attrs={"class": "element element--
range range--volume"}).find('span',attrs={"class": "primary"}).text.strip("Volume:
M")
                price_volume_data.append(after_v)
                print("Found it !")
            except AttributeError as attr_error:
                print("No dice buddy")
                print(attr_error)

        cur_v = i.find('mw-rangebar', attrs={"class": "element element--range range-
-volume"}).find('span', attrs={"class": "primary"}).text.strip("Volume: M")
        day_l = i.find('mw-rangebar', attrs={"class": "element element--range range-
-daily"}).get("range-low")
        day_h = i.find('mw-rangebar', attrs={"class": "element element--range range-
-daily"}).get("range-high")
        year_l = i.find('mw-rangebar', attrs={"class": "element element--range
range--yearly"}).get("range-low")
        year_h = i.find('mw-rangebar', attrs={"class": "element element--range
range--yearly"}).get("range-high")

        time_stamp.append(t_stamp)
        string = time_stamp[0]
        print("Marketwatch last updated data on : " + string)
        price_volume_data.append(cur_v)
        price_volume_data.append(day_h)
        price_volume_data.append(day_l)
        price_volume_data.append(year_h[0:6])
        price_volume_data.append(year_l[0:6])
```

```python
def find_statements(a, b, c, tick, path, path_excel):
    """Here we extract the financial statements and pass the statements into the
dataframe creating function."""

    income_table = a.find_all('div', class_='overflow--table')
    balance_table = b.find_all('div', class_='overflow--table')
    cash_table = c.find_all('div', class_='overflow--table')

    html_table_to_pandas_pickle(income_table, i_t, tick, path, path_excel)
    html_table_to_pandas_pickle(balance_table, b_t, tick, path, path_excel)
    html_table_to_pandas_pickle(cash_table, c_t, tick, path, path_excel)


def find_valuation_data(a, tick, path, path_excel):
    """
    This function searches the parsed html data and extracts the description of the
company, a valuation table, efficiency table, and profitablity table.
    The data is then passed into the dataframe creating function.
    """
    table = a.find_all('div', attrs={"class": "column column--primary"})

    for i in table:

        description = i.find_all('div', attrs={"class": "element element--
description description__long"})
        value_t = i.find_all("table", attrs={"aria-label": "VALUATION data table"})
        efficiency_t = i.find_all("table", attrs={"aria-label": "EFFICIENCY data
table"})
        profitability_t = i.find_all("table", attrs={"aria-label": "PROFITABILITY
data table"})
        html_table_to_pandas_pickle(value_t, v_t, tick, path, path_excel)
        html_table_to_pandas_pickle(efficiency_t, e_t, tick, path, path_excel)
        html_table_to_pandas_pickle(profitability_t, p_t, tick, path, path_excel)

        for i in description:
            descrip = i.find('p').text
            company_description.append(descrip)


def html_table_to_pandas_pickle(input, output, ticker, path, path_excel):
    """This function takes the processed html data and then formats the data by
ensuring proper column names.
       The Pandas dataframe library is then accessed to create a dataframe for the
extracted html and ensures
       that null values are dropped and replaces by '-'.

       This function returns both a pickle file and a excel file. The purpose of
returning a pickle file was to have
       a serialized data source to run computations on.
    """
    for i in input:
        table = pd.read_html(str(i))[0]
        table.rename(columns={'Item Item': 'Item'}, inplace=True)
        df = table.iloc[:, 0].str.split().apply(lambda x: "
".join(OrderedDict.fromkeys(x).keys()))
        table.iloc[:, 0] = df
        table = table.replace('-', np.nan)
        df1 = pd.DataFrame(table).dropna(how="all", thresh=2)
        df1.to_pickle(path + output + "_" + ticker)
        df1.to_excel(path_excel + "\\" + output + "_" + ticker + ".xlsx")




def organize_data(path):
    """
    Current price data and officer data existed on the main page of the ticker so
they were one off tables.
    This data is then organized into its respective key-value pair.
    """

    officer_data_frame = pd.DataFrame(list(zip(name, officer_title, bio)),
columns=['Name',

'Title',

'Biography'],

dtype="string")

    print(price_volume_data)
    try:
        price_volume_frame = pd.DataFrame(list(zip(price_volume_data)),
index=['Current Price',

'Close Price',

'After Hours Volume (In Millions)',

'Current Volume (In Millions)',

'Daily High',

'Daily Low',

'Yearly High',

'Yearly Low', ])
    except ValueError as wrong_length:
        print(wrong_length)
        if len(price_volume_data) > 8:
            price_volume_frame = pd.DataFrame(list(zip(price_volume_data[8:16])),
index=['Current Price',

'Close Price',

'After Hours Volume (In Millions)',

'Current Volume (In Millions)',

'Daily High',

'Daily Low',

'Yearly High',
'Yearly Low', ])

    print(officer_data_frame)

    officer_data_frame.to_pickle(path + df_name)
    price_volume_frame.to_pickle(path + df1_name)

    officer_data_frame.to_excel(path + "excel_format\\" + df_name + ".xlsx")
    price_volume_frame.to_excel(path + "excel_format\\" + df1_name + ".xlsx")





def market_scrape(ticker, path):

    #A browser header is used so that the website doesnt deny repeated requests to
download html
    headers = {'Connection': 'keep-alive',
               'Expires': '-1',
               'Upgrade-Insecure-Requests': '1',
               'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) \
                AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99
Safari/537.36'}

    #Here we gather the pages html for the financial statements
    html_income_state = requests.get(u + ticker + '/financials', headers=headers)
    html_officers = requests.get(u + ticker + '/profile', headers=headers)
    html_balance_sheet = requests.get(u + ticker + "/financials/balance-sheet",
headers=headers)
    html_cash_flow = requests.get(u + ticker + "/financials/cash-flow",
headers=headers)

    #We now use bsf4 module to parse the html and store it into a soup variable
    soup2 = BeautifulSoup(html_income_state.content, 'html.parser')
    soup3 = BeautifulSoup(html_officers.content, 'html.parser')
    soup4 = BeautifulSoup(html_balance_sheet.content, 'html.parser')
    soup5 = BeautifulSoup(html_cash_flow.content, 'html.parser')

    cur_date = datetime.now()
    date_time = cur_date.strftime("%m_%d_%Y")
    path_main = path + "\\" + ticker + "_" + date_time

    #file structure creation to store financial data into a hierarchical structure
    try:
        os.mkdir(path_main)
    except FileExistsError as error:
        print("You already have a file with this exact name")
        return error

    path_company_folder = path_main + "\\"

    try:
        os.mkdir(path_company_folder + "excel_format")
    except FileExistsError as error:
        print("You already have a file with this exact name")
        print(error)
    path_excel = path_company_folder + "excel_format"

    #We then call the above functions to extract
    find_stock_range(soup3)
    find_officers_pid(soup3)
    find_valuation_data(soup3, ticker, path_company_folder, path_excel)
    find_statements(soup2, soup4, soup5, ticker, path_company_folder, path_excel)

    #As the officer data is identified by a unique ID we loop through a range of ids
to extract information on each officer
    for i in pid_officer:
        html_officer_data = rq.urlopen(u + ticker + '/company-profile?pid=' +
str(i))
        soup7 = BeautifulSoup(html_officer_data, 'html.parser')
        find_officer_data(soup7)

    #After we have gathered all data on this specific ticker we clear the memory
stored in the global variables for the next ticker
    organize_data(path_company_folder)
    gc.collect()
    name.clear()
    officer_title.clear()
    bio.clear()
    price_volume_data.clear()
    pid_officer.clear()




#------------------------------------------------------------------------------------
---------------------------------------------------------------


"""
market_scrape() is the main function which interacts and calls the component
function.
We prepare to call market scrape by accessing a ticker file which contains the
symbol for the
top 25 companies by most revenue.
"""


tickers_path = input('Enter the path where the ticker file is stored :\n'
                     'Or press d for default path\n'
                     'Will only work for default system\n'
                     'Input: ')
if tickers_path == "d":
    tickers_path = r'C:\Users\zuck1\PycharmProjects\pythonProject\csvoftickers_list'

top_market_cap = pd.read_csv(tickers_path + '\\' + "top25_comp_b_mcap.csv")
energy_tickers = pd.read_csv(tickers_path + '\\' + "top25Energy.csv")
financial_tickers = pd.read_csv(tickers_path + '\\' + "top25FIN.csv")
IT_tickers = pd.read_csv(tickers_path + '\\' + "top25ITcomp.csv")
real_estate_tickers = pd.read_csv(tickers_path + '\\' + "top25RealEstate.csv")
industrial_tickers = pd.read_csv(tickers_path + "\\" + "top25Industrials.csv")
```

```
#We now take the symbol from the ticker file and turn it into a list that can be
passed to the scraping funcion
top25_companies = top_market_cap['Symbol'].to_list()
top25_energy = energy_tickers['Symbol'].to_list()
top25_fin = financial_tickers['Symbol'].to_list()
top25_IT = IT_tickers['Symbol'].to_list()
top25_RE = real_estate_tickers['Symbol'].to_list()
top25_industrial = industrial_tickers['Symbol'].to_list()

user = input("Do you need to set a path to store your data ? (y / n) : ")

if user == 'n':
    my_path = r"C:\Users\zuck1\PycharmProjects\pythonProject"
elif user == 'y':
    my_path = input('Please insert the path to store your data: \n'
                    'Path=')

selection = input("Select sector to analyze:\n"
                  "a = top 25 Companies with the best market cap\n"
                  "b = top 25 Energy companies\n"
                  "c = top 25 Financial companies\n"
                  "d = top 25 IT companies\n"
                  "e = top 25 Real Estate\n"
                  "f = top 25 Industrial\n"
                  "Input: ")

start_time = time.time()

#The following calls the market_scrape function and creates a file directory for the
selected industry we are scraping data on.

if selection == "a":

    try:
        m_cap_path = my_path + "\\" + "Top 25 Companies - Highest Market Cap"
        os.mkdir(m_cap_path)

    except FileExistsError as file_exists:
        print(file_exists)

    for i in top25_companies:
        market_scrape(i, m_cap_path)
        gc.collect()

    print("--------------------")

elif selection == "b":

    try:
        energy_path = my_path + "\\" + "Top 25 Energy Companies"
        os.mkdir(energy_path)

    except FileExistsError as file_exists:
        print(file_exists)

    for i in top25_energy:
        market_scrape(i, energy_path)
        gc.collect()
    print("-------------------")

elif selection == "c":

    try:
        financial_path = my_path + "\\" + "Top 25 Financial Companies"
        os.mkdir(financial_path)

    except FileExistsError as file_exists:
        print(file_exists)

    for i in top25_fin:
        market_scrape(i, financial_path)
        gc.collect()
    print("-------------------")

elif selection == "d":

    try:
        IT_path = my_path + "\\" + "Top 25 IT Companies"
        os.mkdir(IT_path)

    except FileExistsError as file_exists:
        print(file_exists)

    for i in top25_IT:
        market_scrape(i, IT_path)
        gc.collect()
    print("-------------------")

elif selection == "e":

    try:
        RE_path = my_path + "\\" + "Top 25 Real Estate Companies"
        os.mkdir(RE_path)

    except FileExistsError as file_exists:
        print(file_exists)

    for i in top25_RE:
        market_scrape(i, RE_path)
        gc.collect()
    print("-------------------")

elif selection == "f":

    try:
        IND_path = my_path + "\\" + "Top 25 Industrial Companies"
        os.mkdir(IND_path)

    except FileExistsError as file_exists:
        print(file_exists)

    for i in top25_industrial:
        market_scrape(i, IND_path)
        gc.collect()
    print("-------------------")
```

```
else:
    print("invalid choice")
```