Equal-Weight S&P 500 Index Fund

Introduction & Library Imports

The S&P 500 is the world's most popular stock market index. The largest fund that is benchmarked to this index is the SPDR® S&P 500® ETF Trust. It has more than US$250 billion of assets under management.

The goal of this section of the course is to create a Python script that will accept the value of your portfolio and tell you how many shares of each S&P 500 constituent you should purchase to get an equal-weight version of the index fund.

Library Imports

The first thing we need to do is import the open-source software libraries that we'll be using in this tutorial.

In [2]:
```
import numpy as np #The Numpy numerical computing library
import pandas as pd #The Pandas data science library
import requests #The requests library for HTTP requests in Python
import xlsxwriter #The XlsxWriter libarary for
import math #The Python math module
from contextlib import suppress
import matplotlib.pyplot as plt
```

Importing Our List of Stocks

The next thing we need to do is import the constituents of the S&P 500.

These constituents change over time, so in an ideal world you would connect directly to the index provider (Standard & Poor's) and pull their real-time constituents on a regular basis. 306 Paying for access to the index provider's API is outside of the scope of this course.

There's a static version of the S&P 500 constituents available here. Move this file into the `starter-files` folder so it can be accessed by other files in that directory.

Now it's time to import these stocks to our Jupyter Notebook file.

In [3]:
```
#url='https://github.com/prashantmalan/algorthmic_trading/blob/main/sp_500_stocks.csv?raw=
stocks = pd.read_csv(r'C:\Users\user\Downloads\sp_500_stocks.csv')
print (stocks)
type(stocks)
stocks['Ticker']
```

```
    Ticker
0        A
1      AAL
2      AAP
3     AAPL
4     ABBV
..     ...
496    YUM
497    ZBH
498   ZBRA
```

```
499   ZION
500    ZTS

[501 rows x 1 columns]
```

Out[3]:

```
0         A
1       AAL
2       AAP
3      AAPL
4      ABBV
       ...
496     YUM
497     ZBH
498    ZBRA
499    ZION
500     ZTS
Name: Ticker, Length: 501, dtype: object
```

Acquiring an API Token

Now it's time to import our IEX Cloud API token. This is the data provider that we will be using throughout this course.

API tokens (and other sensitive information) should be stored in a `secrets.py` file that doesn't get pushed to your local Git repository. We'll be using a sandbox API token in this course, which means that the data we'll use is randomly-generated and (more importantly) has no cost associated with it.

Click here to download your `secrets.py` file. Move the file into the same directory as this Jupyter Notebook before proceeding.

In [4]:

```
from secrets import IEX_CLOUD_API_TOKEN
```

Making Our First API Call

Now it's time to structure our API calls to IEX cloud.

We need the following information from the API:

- Market capitalization for each stock
- Price of each stock

In [5]:

```
symbol='AMZN'
print(IEX_CLOUD_API_TOKEN)
##IEX_CLOUD_API_TOKEN="059b97af715d417d9f49f50b51b1c448"
api_url = f'https://sandbox.iexapis.com/stable/stock/{symbol}/quote?token={IEX_CLOUD_API_T
data = requests.get(api_url).json()
data
```

```
Tpk_059b97af715d417d9f49f50b51b1c448
```

Out[5]:

```
{'avgTotalVolume': 67425418,
 'calculationPrice': 'close',
 'change': 0.45,
 'changePercent': 0.00338,
 'close': 138.35,
 'closeSource': 'ailcffio',
 'closeTime': 1666077041054,
 'companyName': 'Amazon.com Inc.',
 'currency': 'USD',
 'delayedPrice': 140.19,
```

```
'delayedPriceTime': 1688794081504,
'extendedChange': -0.249,
'extendedChangePercent': -0.0019,
'extendedPrice': 137.293,
'extendedPriceTime': 1711105604178,
'high': 144.311,
'highSource': 'tpuy e nedalri5ce ie1dm',
'highTime': 1701375314381,
'iexAskPrice': 0,
'iexAskSize': 0,
'iexBidPrice': 0,
'iexBidSize': 0,
'iexClose': 136.58,
'iexCloseTime': 1700196954072,
'iexLastUpdated': 1721938247689,
'iexMarketPercent': 0.024403188137940884,
'iexOpen': 135.02,
'iexOpenTime': 1701464304649,
'iexRealtimePrice': 139.5,
'iexRealtimeSize': 100,
'iexVolume': 1887782,
'lastTradeTime': 1721198022258,
'latestPrice': 135.99,
'latestSource': 'Close',
'latestTime': 'August 1, 2022',
'latestUpdate': 1672556419249,
'latestVolume': 78311736,
'low': 135.93,
'lowSource': 'maitpn1cr eledd 5eiuey ',
'lowTime': 1691565388714,
'marketCap': 1418836431063,
'oddLotDelayedPrice': 141.809,
'oddLotDelayedPriceTime': 1673459570867,
'open': 139,
'openTime': 1727762670462,
'openSource': 'fcolfiia',
'peRatio': 3.29,
'previousClose': 140.31,
'previousVolume': 150154951,
'primaryExchange': 'ASDNAQ',
'symbol': 'AMZN',
'volume': 78956116,
'week52High': 189.09,
'week52Low': 104.57,
'ytdChange': -0.18913173453118073,
'isUSMarketOpen': False}
```


Parsing Our API Call

The API call that we executed in the last code block contains all of the information required to build our equal-weight S&P 500 strategy.

With that said, the data isn't in a proper format yet. We need to parse it first.

In [6]:
```
data['latestPrice']
data['marketCap']
```

Out[6]:
```
1418836431063
```

Adding Our Stocks Data to a Pandas DataFrame

The next thing we need to do is add our stock's price and market capitalization to a pandas DataFrame. Think of a DataFrame like the Python version of a spreadsheet. It stores tabular data.

In [7]:
```
my_columns = ['Ticker', 'Price','Market Capitalization', 'Number Of Shares to Buy']
sp_dataframe = pd.DataFrame(columns = my_columns)
sp_dataframe
```

Out[7]:

Ticker	Price	Market Capitalization	Number Of Shares to Buy

In [8]:
```
sp_dataframe = sp_dataframe.append(
                                        pd.Series(['AAPL',
                                                   data['latestPrice'],
                                                   data['marketCap'],
                                                   'N/A'],
                                                  index = my_columns),
                                        ignore_index = True)
sp_dataframe
```

Out[8]:

	Ticker	Price	Market Capitalization	Number Of Shares to Buy
0	AAPL	135.99	1418836431063	N/A

Looping Through The Tickers in Our List of Stocks

Using the same logic that we outlined above, we can pull data for all S&P 500 stocks and store their data in the DataFrame using a `for` loop.

In [9]:
```
sp_dataframe = pd.DataFrame(columns = my_columns)
print(stocks)
for symbol in stocks['Ticker']:
    api_url = f'https://sandbox.iexapis.com/stable/stock/{symbol}/quote?token={IEX_CLOUD_A
    print(api_url)
    data = requests.get(api_url).json()
    print(data)
    sp_dataframe = sp_dataframe.append(
                                        pd.Series([symbol,
                                                   data['latestPrice'],
                                                   data['marketCap'],
                                                   'N/A'],
                                                  index = my_columns),
                                        ignore_index = True)
    print(sp_dataframe)
```

```
    Ticker
0        A
1      AAL
2      AAP
3     AAPL
4     ABBV
..     ...
496    YUM
497    ZBH
498   ZBRA
499   ZION
500    ZTS

[501 rows x 1 columns]
```

https://sandbox.iexapis.com/stable/stock/A/quote?token=Tpk_059b97af715d417d9f49f50b51b1c448
{'avgTotalVolume': 1390754, 'calculationPrice': 'close', 'change': -0.69, 'changePercent': -0.005, 'close': 139.96, 'closeSource': 'alcoiiff', 'closeTime': 1706984432595, 'companyName': 'Agilent Technologies Inc.', 'currency': 'USD', 'delayedPrice': 134.53, 'delayedPriceTime': 1684184874052, 'extendedChange': 0.38, 'extendedChangePercent': 0.00283, 'extendedPrice': 137.2, 'extendedPriceTime': 1724093090481, 'high': 141.25, 'highSource': 'eet1inr m 5p ed ceyaiuld', 'highTime': 1672049761709, 'iexAskPrice': 0, 'iexAskSize': 0, 'iexBidPrice': 0, 'iexBidSize': 0, 'iexClose': 133.68, 'iexCloseTime': 1687549854060, 'iexLastUpdated': 1708545593879, 'iexMarketPercent': 0.05564067690386085, 'iexOpen': 138.9, 'iexOpenTime': 1700898550550, 'iexRealtimePrice': 133.76, 'iexRealtimeSize': 1, 'iexVolume': 60417, 'lastTradeTime': 1666110003994, 'latestPrice': 139.78, 'latestSource': 'Close', 'latestTime': 'August 1, 2022', 'latestUpdate': 1693083073577, 'latestVolume': 1084364, 'low': 137.13, 'lowSource': 'mciieteprIa EXr l e', 'lowTime': 1686504222568, 'marketCap': 40208903264, 'oddLotDelayedPrice': 137.52, 'oddLotDelayedPriceTime': 1724113453623, 'open': 137.8, 'openTime': 1693782547907, 'openSource': 'ifilofac', 'peRatio': 32.68, 'previousClose': 139.5, 'previousVolume': 2259336, 'primaryExchange': 'I.TOCNNECKOH RCAEW NY S EKXG', 'symbol': 'A', 'volume': 1131748, 'week52High': 182.86, 'week52Low': 115.72, 'ytdChange': -0.16834075085377181, 'isUSMarketOpen': False}
  Ticker   Price Market Capitalization Number Of Shares to Buy
0   A 139.78   40208903264   N/A
https://sandbox.iexapis.com/stable/stock/AAL/quote?token=Tpk_059b97af715d417d9f49f50b51b1c448
{'avgTotalVolume': 37616163, 'calculationPrice': 'close', 'change': 0.59, 'changePercent': 0.04345, 'close': 14.97, 'closeSource': 'foialcif', 'closeTime': 1732063965743, 'companyName': 'American Airlines Group Inc', 'currency': 'USD', 'delayedPrice': 14.36, 'delayedPriceTime': 1692868400158, 'extendedChange': -0.06, 'extendedChangePercent': -0.0043, 'extendedPrice': 14.28, 'extendedPriceTime': 1725592521026, 'high': 14.97, 'highSource': '1ida iducy rpe5teeneml ', 'highTime': 1742003451105, 'iexAskPrice': 0, 'iexAskSize': 0, 'iexBidPrice': 0, 'iexBidSize': 0, 'iexClose': 14.8, 'iexCloseTime': 1721136664186, 'iexLastUpdated': 1721236929047, 'iexMarketPercent': 0.01965486597898694, 'iexOpen': 13.997, 'iexOpenTime': 1715879273908, 'iexRealtimePrice': 14.59, 'iexRealtimeSize': 107, 'iexVolume': 653858, 'lastTradeTime': 1661840491897, 'latestPrice': 14.5, 'latestSource': 'Close', 'latestTime': 'August 1, 2022', 'latestUpdate': 1666263074890, 'latestVolume': 33309263, 'low': 13.722, 'lowSource': 'mau yr1dcenpiti5leed e ', 'lowTime': 1691160321509, 'marketCap': 9731447657, 'oddLotDelayedPrice': 14.58, 'oddLotDelayedPriceTime': 1709883610706, 'open': 14.07, 'openTime': 1670719415681, 'openSource': 'ofalfici', 'peRatio': -5.03, 'previousClose': 13.83, 'previousVolume': 27426839, 'primaryExchange': 'QASNAD', 'symbol': 'AAL', 'volume': 33330700, 'week52High': 23.41, 'week52Low': 12.52, 'ytdChange': -0.1961433827294921, 'isUSMarketOpen': False}
  Ticker   Price Market Capitalization Number Of Shares to Buy
0   A 139.78   40208903264   N/A
1   AAL   14.50   9731447657   N/A
https://sandbox.iexapis.com/stable/stock/AAP/quote?token=Tpk_059b97af715d417d9f49f50b51b1c448
{'avgTotalVolume': 688405, 'calculationPrice': 'close', 'change': 2.41, 'changePercent': 0.01253, 'close': 200.91, 'closeSource': 'cflaofii', 'closeTime': 1693069642993, 'companyName': 'Advance Auto Parts Inc', 'currency': 'USD', 'delayedPrice': 203.36, 'delayedPriceTime': 1670582317353, 'extendedChange': 0, 'extendedChangePercent': 0, 'extendedPrice': 204.05, 'extendedPriceTime': 1701031899429, 'high': 200.91, 'highSource': ' lndiedme ut1eyep5riac ', 'highTime': 1729406387014, 'iexAskPrice': 0, 'iexAskSize': 0, 'iexBidPrice': 0, 'iexBidSize': 0, 'iexClose': 199.12, 'iexCloseTime': 1741854663011, 'iexLastUpdated': 1670983110424, 'iexMarketPercent': 0.09143692094494395, 'iexOpen': 195.62, 'iexOpenTime': 1704890307579, 'iexRealtimePrice': 203.81, 'iexRealtimeSize': 1, 'iexVolume': 49801, 'lastTradeTime': 1720453354081, 'latestPrice': 201.23, 'latestSource': 'Close', 'latestTime': 'August 1, 2022', 'latestUpdate': 1680694695307, 'latestVolume': 568912, 'low': 193.8, 'lowSource': 'e1udtiy aecid nrp5eeml', 'lowTime': 1703197240406, 'marketCap': 11951882635, 'oddLotDelayedPrice': 205.11, 'oddLotDelayedPriceTime': 1685416760715, 'open': 193.8, 'openTime': 1699661427275, 'openSource': 'fclafioi', 'peRatio': 22.7, 'previousClose': 194.66, 'previousVolume': 543164, 'primaryExchange': 'IE KOY CRGNNWXCE.NCHSTO AKE', 'symbol': 'AAP', 'volume': 553182, 'week52High': 240.69, 'week52Low': 172, 'ytdChange': -0.1743028930141297, 'isUSMarketOpen': False}
  Ticker   Price Market Capitalization Number Of Shares to Buy
0   A 139.78   40208903264   N/A
1   AAL   14.50   9731447657   N/A
2   AAP 201.23   11951882635   N/A

```
https://sandbox.iexapis.com/stable/stock/AAPL/quote?token=Tpk_059b97af715d417d9f49f50b51b1c448
{'avgTotalVolume': 75994543, 'calculationPrice': 'close', 'change': -1, 'changePercent': -0.00617, 'close': 163.23, 'closeSource': 'filicafo', 'closeTime': 1688291891733, 'companyName': 'Apple Inc', 'currency': 'USD', 'delayedPrice': 166.68, 'delayedPriceTime': 1714509033024, 'extendedChange': -0.14, 'extendedChangePercent': -0.00088, 'extendedPrice': 162.86, 'extendedPriceTime': 1710654286866, 'high': 167.83, 'highSource': 'ele5ucyepn diadi mre1 t', 'highTime': 1700655109375, 'iexAskPrice': 0, 'iexAskSize': 0, 'iexBidPrice': 0, 'iexBidSize': 0, 'iexClose': 162.05, 'iexCloseTime': 1701975128014, 'iexLastUpdated': 1697838502653, 'iexMarketPercent': 0.0213614514921501, 'iexOpen': 167.23, 'iexOpenTime': 1709521342231, 'iexRealtimePrice': 167.18, 'iexRealtimeSize': 227, 'iexVolume': 1398908, 'lastTradeTime': 1684485314673, 'latestPrice': 167.72, 'latestSource': 'Close', 'latestTime': 'August 1, 2022', 'latestUpdate': 1732169661689, 'latestVolume': 70466253, 'low': 163.23, 'lowSource': 'en dle1rmed5 utipyiae c', 'lowTime': 1686992968181, 'marketCap': 2738382373108, 'oddLotDelayedPrice': 168.889, 'oddLotDelayedPriceTime': 1707159826777, 'open': 163.56, 'openTime': 1698987382720, 'openSource': 'oficlfai', 'peRatio': 27.38, 'previousClose': 164.06, 'previousVolume': 102881347, 'primaryExchange': 'DANAQS', 'symbol': 'AAPL', 'volume': 70032289, 'week52High': 190.08, 'week52Low': 134.09, 'ytdChange': -0.09124399385222329, 'isUSMarketOpen': False}
  Ticker   Price Market Capitalization Number Of Shares to Buy
0      A  139.78           40208903264                     N/A
1    AAL   14.50            9731447657                     N/A
2    AAP  201.23           11951882635                     N/A
3   AAPL  167.72         2738382373108                     N/A
https://sandbox.iexapis.com/stable/stock/ABBV/quote?token=Tpk_059b97af715d417d9f49f50b51b1c448
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_34572/1703773792.py in <module>
      4     api_url = f'https://sandbox.iexapis.com/stable/stock/{symbol}/quote?token={IEX_CLOUD_API_TOKEN}'
      5     print(api_url)
----> 6     data = requests.get(api_url).json()
      7     print(data)
      8     sp_dataframe = sp_dataframe.append(

~\anaconda3\lib\site-packages\requests\api.py in get(url, params, **kwargs)
     73     """
     74
---> 75     return request('get', url, params=params, **kwargs)
     76
     77

~\anaconda3\lib\site-packages\requests\api.py in request(method, url, **kwargs)
     59     # cases, and look like a memory leak in others.
     60     with sessions.Session() as session:
---> 61         return session.request(method=method, url=url, **kwargs)
     62
     63

~\anaconda3\lib\site-packages\requests\sessions.py in request(self, method, url, params, data, headers, cookies, files, auth, timeout, allow_redirects, proxies, hooks, stream, verify, cert, json)
    540         }
    541         send_kwargs.update(settings)
--> 542         resp = self.send(prep, **send_kwargs)
    543
    544         return resp

~\anaconda3\lib\site-packages\requests\sessions.py in send(self, request, **kwargs)
    653
    654         # Send the request
--> 655         r = adapter.send(request, **kwargs)
    656
    657         # Total elapsed time of the request (approximately)
```

```
~\anaconda3\lib\site-packages\requests\adapters.py in send(self, request, stream, timeout,
 verify, cert, proxies)
    437         try:
    438             if not chunked:
--> 439                 resp = conn.urlopen(
    440                     method=request.method,
    441                     url=url,

~\anaconda3\lib\site-packages\urllib3\connectionpool.py in urlopen(self, method, url, bod
y, headers, retries, redirect, assert_same_host, timeout, pool_timeout, release_conn, chun
ked, body_pos, **response_kw)
    697
    698             # Make the request on the httplib connection object.
--> 699             httplib_response = self._make_request(
    700                 conn,
    701                 method,

~\anaconda3\lib\site-packages\urllib3\connectionpool.py in _make_request(self, conn, metho
d, url, timeout, chunked, **httplib_request_kw)
    443                     # Python 3 (including for exceptions like SystemExit).
    444                     # Otherwise it looks like a bug in the code.
--> 445                     six.raise_from(e, None)
    446         except (SocketTimeout, BaseSSLError, SocketError) as e:
    447             self._raise_timeout(err=e, url=url, timeout_value=read_timeout)

~\anaconda3\lib\site-packages\urllib3\packages\six.py in raise_from(value, from_value)

~\anaconda3\lib\site-packages\urllib3\connectionpool.py in _make_request(self, conn, metho
d, url, timeout, chunked, **httplib_request_kw)
    438                 # Python 3
    439                 try:
--> 440                     httplib_response = conn.getresponse()
    441                 except BaseException as e:
    442                     # Remove the TypeError from the exception chain in

~\anaconda3\lib\http\client.py in getresponse(self)
   1369         try:
   1370             try:
-> 1371                 response.begin()
   1372             except ConnectionError:
   1373                 self.close()

~\anaconda3\lib\http\client.py in begin(self)
    317         # read until we get a non-100 response
    318         while True:
--> 319             version, status, reason = self._read_status()
    320             if status != CONTINUE:
    321                 break

~\anaconda3\lib\http\client.py in _read_status(self)
    278
    279     def _read_status(self):
--> 280         line = str(self.fp.readline(_MAXLINE + 1), "iso-8859-1")
    281         if len(line) > _MAXLINE:
    282             raise LineTooLong("status line")

~\anaconda3\lib\socket.py in readinto(self, b)
    702         while True:
    703             try:
--> 704                 return self._sock.recv_into(b)
    705             except timeout:
    706                 self._timeout_occurred = True

~\anaconda3\lib\ssl.py in recv_into(self, buffer, nbytes, flags)
   1239                   "non-zero flags not allowed in calls to recv_into() on %s" %
```

```
   1240                 self.__class__)
-> 1241             return self.read(nbytes, buffer)
   1242         else:
   1243             return super().recv_into(buffer, nbytes, flags)

~\anaconda3\lib\ssl.py in read(self, len, buffer)
   1097         try:
   1098             if buffer is not None:
-> 1099                 return self._sslobj.read(len, buffer)
   1100             else:
   1101                 return self._sslobj.read(len)

KeyboardInterrupt:
```

In [24]:
```
sp_dataframe
```

Out[24]:

	Ticker	Price	Market Capitalization	Number Of Shares to Buy
0	A	134.80	40650801779	N/A
1	AAL	13.86	9003151972	N/A
2	AAP	199.95	11784743480	N/A
3	AAPL	170.63	2682163395015	N/A
4	ABBV	143.55	257378286828	N/A
...	...	...	...	...
496	YUM	124.68	35872443547	N/A
497	ZBH	115.05	23309006292	N/A
498	ZBRA	365.99	19187118594	N/A
499	ZION	54.93	8411108253	N/A
500	ZTS	189.10	88402798017	N/A

501 rows × 4 columns

Using Batch API Calls to Improve Performance

Batch API calls are one of the easiest ways to improve the performance of your code.

This is because HTTP requests are typically one of the slowest components of a script.

Also, API providers will often give you discounted rates for using batch API calls since they are easier for the API provider to respond to.

IEX Cloud limits their batch API calls to 100 tickers per request. Still, this reduces the number of API calls we'll make in this section from 500 to 5 - huge improvement! In this section, we'll split our list of stocks into groups of 100 and then make a batch API call for each group.

In [10]:
```
# Function sourced from
# https://stackoverflow.com/questions/312443/how-do-you-split-a-list-into-evenly-sized-chu
def chunks(lst, n):
    """Yield successive n-sized chunks from lst."""
    for i in range(0, len(lst), n):
        yield lst[i:i + n]
```

In [11]:

```
symbol_groups = list(chunks(stocks['Ticker'], 100))
symbol_strings = []
for i in range(0, len(symbol_groups)):
    symbol_strings.append(','.join(symbol_groups[i]))
    print(symbol_strings[i])

sp_dataframe = pd.DataFrame(columns = my_columns)

for symbol_string in symbol_strings:
#     print(symbol_strings)
        batch_api_call_url = f'https://sandbox.iexapis.com/stable/stock/market/batch/?type
        data = requests.get(batch_api_call_url).json()
        print (data.get(BaseException))

        for symbol in symbol_string.split(','):
            sp_dataframe = sp_dataframe.append(
                                            pd.Series([symbol,
                                                       data[symbol]['quote']['latestPrice'
                                                       data[symbol]['quote']['marketCap'],
                                                       'N/A'],
                                                      index = my_columns),
                                            ignore_index = True)



sp_dataframe
```

```
A,AAL,AAP,AAPL,ABBV,ABC,ABMD,ABT,ACN,ADBE,ADI,ADM,ADP,ADSK,AEE,AEP,AES,AFL,AIG,AIV,AIZ,AJ
G,AKAM,ALB,ALGN,ALK,ALL,ALLE,ALXN,AMAT,AMCR,AMD,AME,AMGN,AMP,AMT,AMZN,ANET,ANSS,ANTM,AON,A
OS,APA,APD,APH,APTV,ARE,ATO,ATVI,AVB,AVGO,AVY,AWK,AXP,AZO,BA,BAC,BAX,BBY,BDX,BEN,BF.B,BII
B,BIO,BK,BKNG,BKR,BLK,BLL,BMY,BR,BRK.B,BSX,BWA,BXP,C,CAG,CAH,CARR,CAT,CB,CBOE,CBRE,CCI,CC
L,CDNS,CDW,CE,CERN,CF,CFG,CHD,CHRW,CHTR,CI,CINF,CL,CLX,CMA,CMCSA
CME,CMG,CMI,CMS,CNC,CNP,COF,COG,COO,COP,COST,COTY,CPB,CPRT,CRM,CSCO,CSX,CTAS,CTL,CTSH,CTV
A,CTXS,CVS,CVX,CXO,D,DAL,DD,DE,DFS,DG,DGX,DHI,DHR,DIS,DISCK,DISH,DLR,DLTR,DOV,DOW,DPZ,DRE,
DRI,DTE,DUK,DVA,DVN,DXC,DXCM,EA,EBAY,ECL,ED,EFX,EIX,EL,EMN,EMR,EOG,EQIX,EQR,ES,ESS,ETFC,ET
N,ETR,EVRG,EW,EXC,EXPD,EXPE,EXR,F,FANG,FAST,FB,FBHS,FCX,FDX,FE,FFIV,FIS,FISV,FITB,FLIR,FL
S,FLT,FMC,FOX,FOXA,FRC,FRT,FTI,FTNT,FTV,GD,GE,GILD,GIS
GL,GLW,GM,GOOG,GOOGL,GPC,GPN,GPS,GRMN,GS,GWW,HAL,HAS,HBAN,HBI,HCA,HD,HES,HIG,HII,HLT,HOLX,
HON,HPE,HPQ,HRB,HRL,HSIC,HST,HSY,HUM,HWM,IBM,ICE,IDXX,IEX,IFF,ILMN,INCY,INFO,INTC,INTU,IP,
IPG,IPGP,IQV,IR,IRM,ISRG,IT,ITW,IVZ,J,JBHT,JCI,JKHY,JNJ,JNPR,JPM,K,KEY,KEYS,KHC,KIM,KLAC,K
MB,KMI,KMX,KO,KR,KSS,KSU,L,LB,LDOS,LEG,LEN,LH,LHX,LIN,LKQ,LLY,LMT,LNC,LNT,LOW,LRCX,LUV,LV
S,LW,LYB,LYV,MA,MAA,MAR,MAS,MCD,MCHP,MCK,MCO
MDLZ,MDT,MET,MGM,MHK,MKC,MKTX,MLM,MMC,MMM,MNST,MO,MOS,MPC,MRK,MRO,MS,MSCI,MSFT,MSI,MTB,MT
D,MU,MXIM,MYL,NBL,NCLH,NDAQ,NEE,NEM,NFLX,NI,NKE,NLOK,NLSN,NOC,NOV,NOW,NRG,NSC,NTAP,NTRS,NU
E,NVDA,NVR,NWL,NWS,NWSA,O,ODFL,OKE,OMC,ORCL,ORLY,OTIS,OXY,PAYC,PAYX,PBCT,PCAR,PEAK,PEG,PE
P,PFE,PFG,PG,PGR,PH,PHM,PKG,PKI,PLD,PM,PNC,PNR,PNW,PPG,PPL,PRGO,PRU,PSA,PSX,PVH,PWR,PXD,PY
PL,QCOM,QRVO,RCL,RE,REG,REGN,RF,RHI,RJF,RL,RMD,ROK,ROL,ROP
ROST,RSG,RTX,SBAC,SBUX,SCHW,SEE,SHW,SIVB,SJM,SLB,SLG,SNA,SNPS,SO,SPG,SPGI,SRE,STE,STT,STX,
STZ,SWK,SWKS,SYF,SYK,SYY,T,TAP,TDG,TDY,TEL,TFC,TFX,TGT,TIF,TJX,TMO,TMUS,TPR,TROW,TRV,TSCO,
TSN,TT,TTWO,TWTR,TXN,TXT,TYL,UA,UAA,UAL,UDR,UHS,ULTA,UNH,UNM,UNP,UPS,URI,USB,V,VAR,VFC,VL
O,VMC,VNO,VRSK,VRSN,VRTX,VTR,VZ,WAB,WAT,WBA,WDC,WEC,WELL,WFC,WHR,WM,WMB,WMT,WRB,WRK,WST,W
U,WY,WYNN,XEL,XLNX,XOM,XRAY,XRX,XYL,YUM,ZBH,ZBRA,ZION
ZTS
None
None
None
None
None
None
```

Out[11]:

	Ticker	Price	Market Capitalization	Number Of Shares to Buy
0	A	137.20	40967830681	N/A
1	AAL	14.90	9739888865	N/A

	Ticker	Price	Market Capitalization	Number Of Shares to Buy
2	AAP	205.17	11896476506	N/A
3	AAPL	168.39	2651588973400	N/A
4	ABBV	140.49	256280606512	N/A
...	...	...	...	...
496	YUM	127.26	35864619218	N/A
497	ZBH	112.15	23681921711	N/A
498	ZBRA	373.43	18915506689	N/A
499	ZION	56.30	8189878051	N/A
500	ZTS	186.23	85256452238	N/A

501 rows × 4 columns

In [ ]:

Calculating the Number of Shares to Buy

As you can see in the DataFrame above, we stil haven't calculated the number of shares of each stock to buy.

We'll do that next.

In [12]:
```
portfolio_size = input("Enter the value of your portfolio:")

try:
    val = float(portfolio_size)
except ValueError:
    print("That's not a number! \n Try again:")
    portfolio_size = input("Enter the value of your portfolio:")
```

```
Enter the value of your portfolio:1000000
```

In [13]:
```
position_size = float(portfolio_size) / len(sp_dataframe.index)
print(len(sp_dataframe.index))
print(position_size)
for i in range(0, len(sp_dataframe['Ticker'])):
    sp_dataframe.loc[i, 'Number Of Shares to Buy'] = math.floor(position_size / sp_datafra
sp_dataframe
```

```
501
1996.007984031936
```

Out[13]:

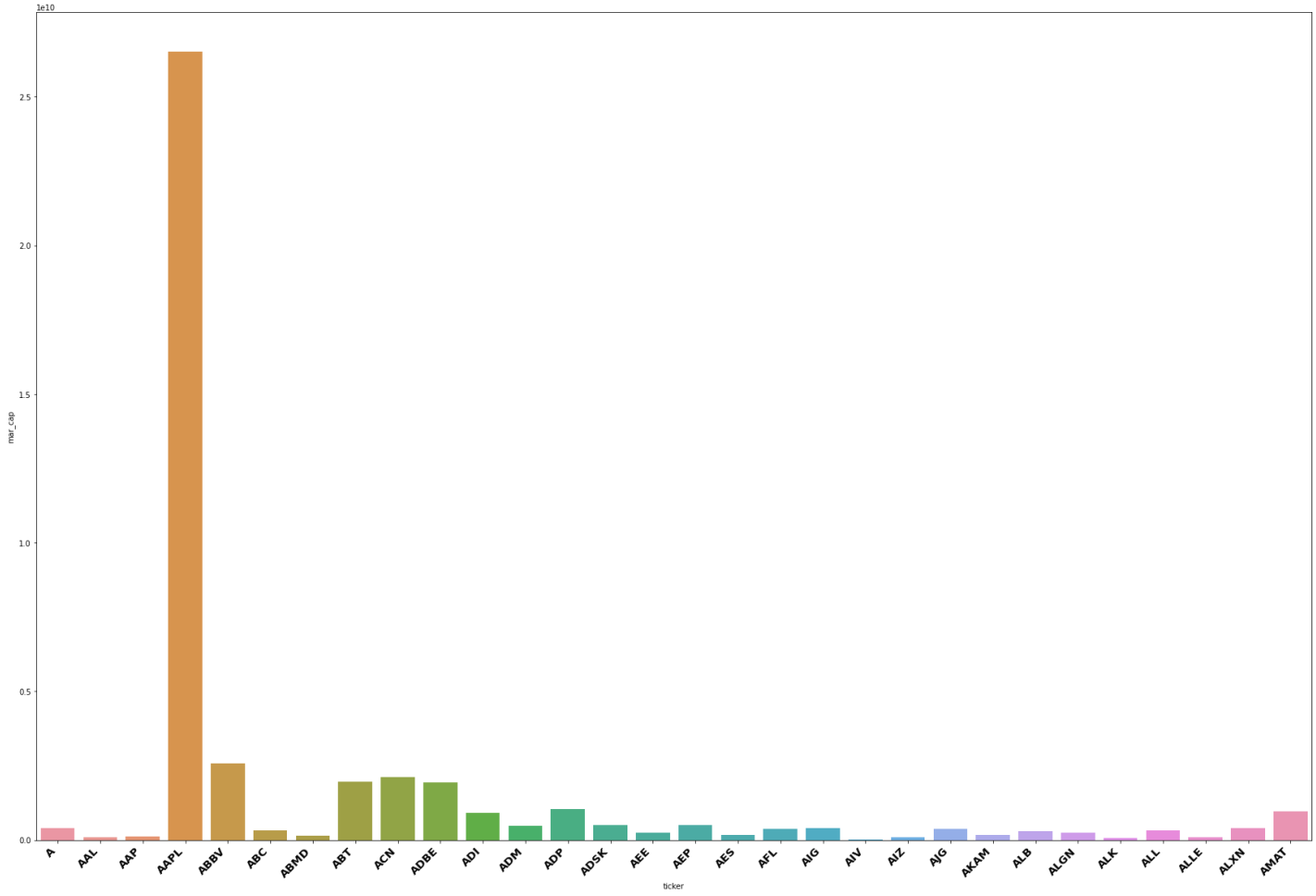
	Ticker	Price	Market Capitalization	Number Of Shares to Buy
0	A	137.20	40967830681	14
1	AAL	14.90	9739888865	133
2	AAP	205.17	11896476506	9
3	AAPL	168.39	2651588973400	11
4	ABBV	140.49	256280606512	14
...	...	...	...	...

	Ticker	Price	Market Capitalization	Number Of Shares to Buy
496	YUM	127.26	35864619218	15
497	ZBH	112.15	23681921711	17
498	ZBRA	373.43	18915506689	5
499	ZION	56.30	8189878051	35
500	ZTS	186.23	85256452238	10

501 rows × 4 columns

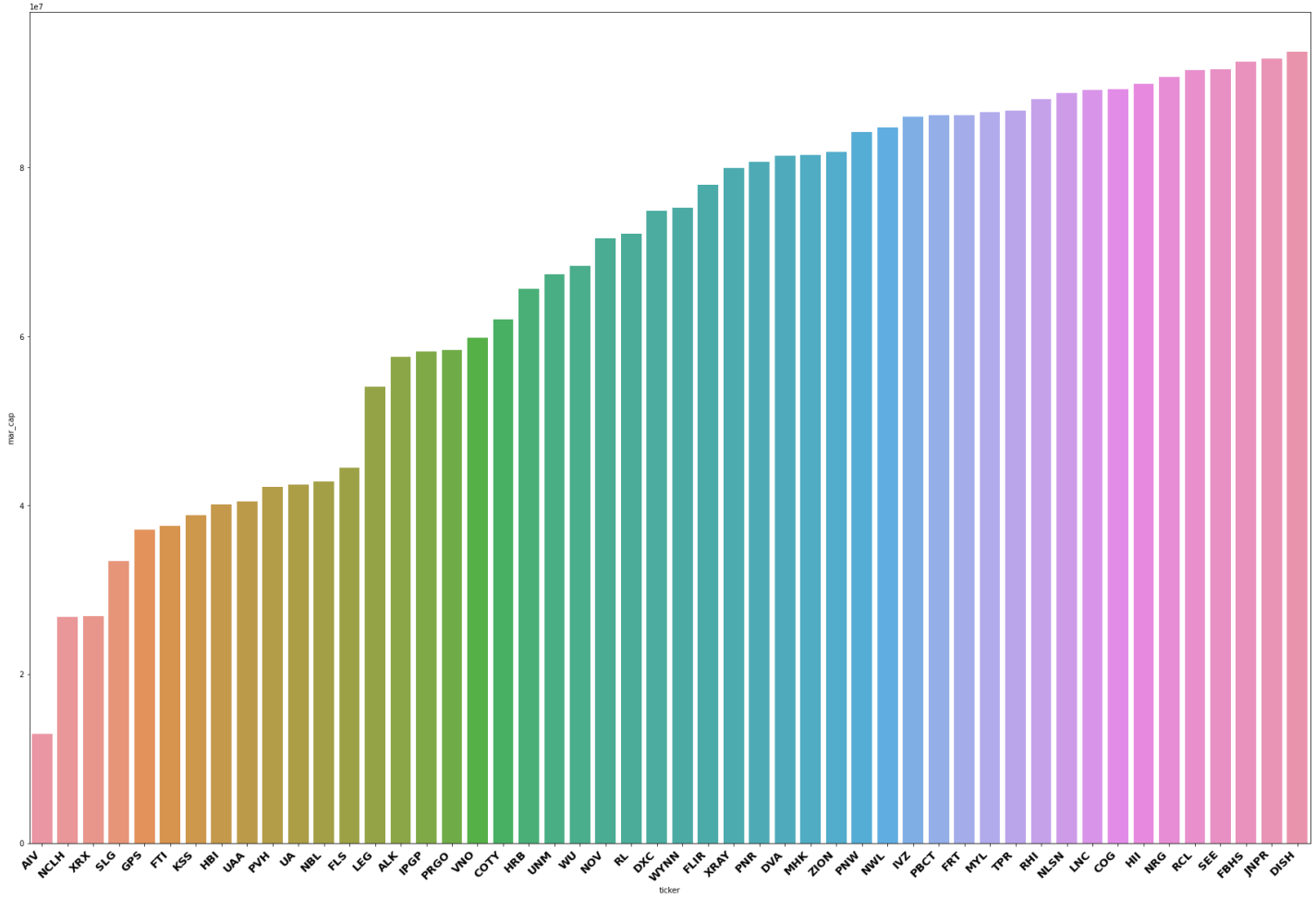
In [14]:
```
import seaborn as sns
sp_dataFrame_graph=pd.DataFrame()
sp_dataFrame_graph['ticker']=sp_dataframe["Ticker"]
sp_dataFrame_graph['mar_cap']=sp_dataframe["Market Capitalization"].astype(float)/100
sp_dataFrame_graph['nb_shares']=sp_dataframe["Number Of Shares to Buy"].astype(int)
```

In [15]:
```
#first 30 tickers
plt.figure(figsize=(30,20))
ax = sns.barplot(x="ticker",y=sp_dataFrame_graph["mar_cap"],
                 data=sp_dataFrame_graph.head(30))
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right',
    fontweight='bold',
    fontsize='x-large')
plt.show()
```



In [16]:
```
# sortted values by market cap
plt.figure(figsize=(30,20))
```

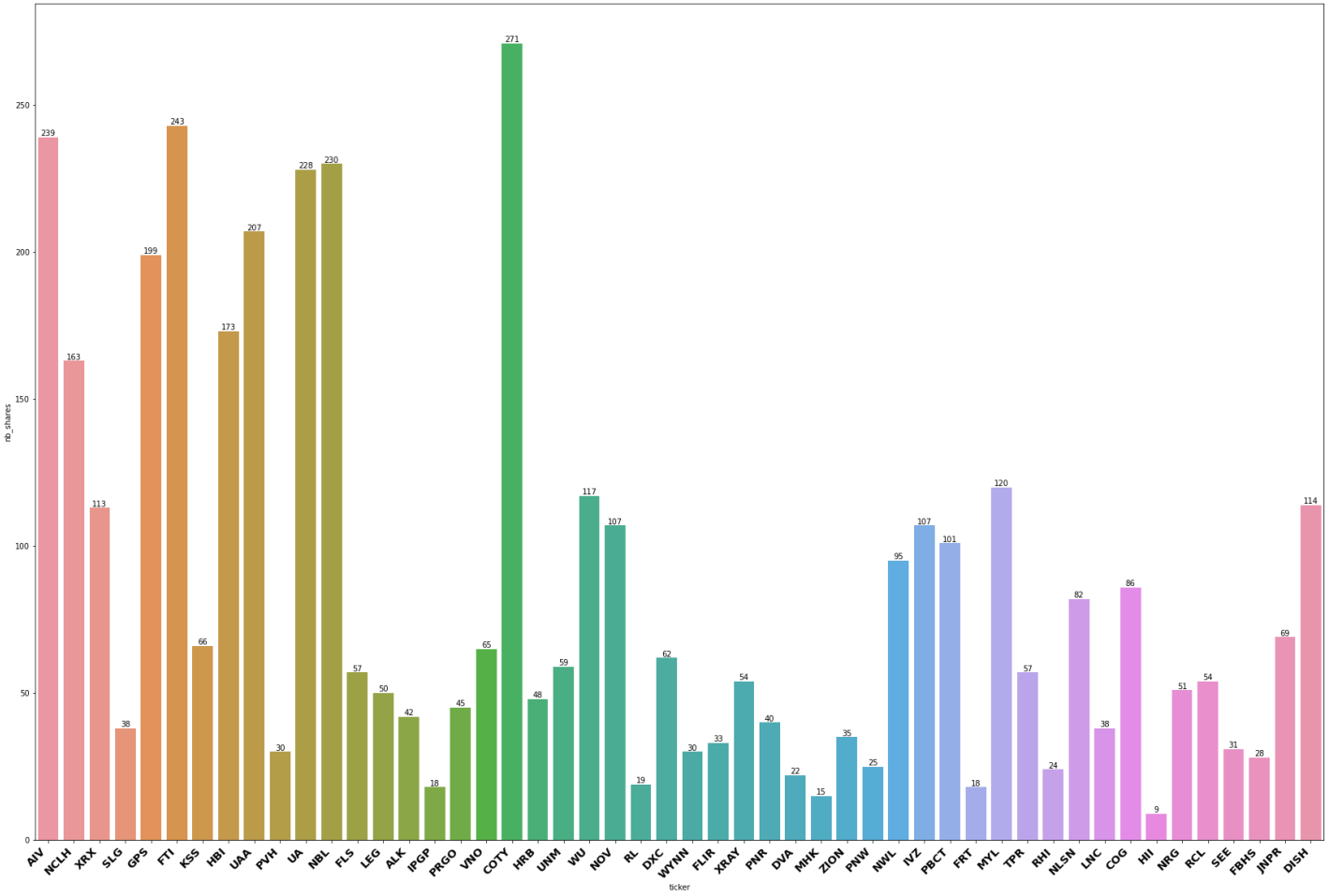
```
ax = sns.barplot(x="ticker",y=sp_dataFrame_graph["mar_cap"].astype(float),
                 data=sp_dataFrame_graph.sort_values(by="mar_cap").head(50))
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right',
    fontweight='bold',
    fontsize='x-large')
plt.show()
```



In [17]:

```
# nb_shares
plt.figure(figsize=(30,20))
ax = sns.barplot(x="ticker",y=sp_dataFrame_graph["nb_shares"].astype(float),
                 data=sp_dataFrame_graph.sort_values(by="mar_cap").head(50))


ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right',
    fontweight='bold',
    fontsize='x-large')
for i in ax.containers:
    ax.bar_label(i,)
```



Formatting Our Excel Output

We will be using the XlsxWriter library for Python to create nicely-formatted Excel files.

XlsxWriter is an excellent package and offers tons of customization. However, the tradeoff for this is that the library can seem very complicated to new users. Accordingly, this section will be fairly long because I want to do a good job of explaining how XlsxWriter works.

Initializing our XlsxWriter Object

In [18]:
```
writer = pd.ExcelWriter(r'C:\Users\user\Downloads\recommended_trades.xlsx', engine='xlsxwr
sp_dataframe.to_excel(writer, sheet_name='Recommended Trades', index = False)
```

Creating the Formats We'll Need For Our `.xlsx` File

Formats include colors, fonts, and also symbols like `%` and `$`. We'll need four main formats for our Excel document:

- String format for tickers
- $XX.XX format for stock prices
- $XX,XXX format for market capitalization
- Integer format for the number of shares to purchase

In [19]:
```
background_color = '#0a0a23'
font_color = '#ffffff'
```

```
string_format = writer.book.add_format(
        {
            'font_color': font_color,
            'bg_color': background_color,
            'border': 1
        }
    )

dollar_format = writer.book.add_format(
        {
            'num_format':'$0.00',
            'font_color': font_color,
            'bg_color': background_color,
            'border': 1
        }
    )

integer_format = writer.book.add_format(
        {
            'num_format':'0',
            'font_color': font_color,
            'bg_color': background_color,
            'border': 1
        }
    )
```

Applying the Formats to the Columns of Our `.xlsx` File

We can use the `set_column` method applied to the `writer.sheets['Recommended Trades']` object to apply formats to specific columns of our spreadsheets.

Here's an example:

```
writer.sheets['Recommended Trades'].set_column('B:B', #This tells the method to apply the format to column B
                    18, #This tells the method to apply a column width of 18 pixels
                    string_format #This applies the format 'string_format' to the column
                   )
```

In [35]:
```
# writer.sheets['Recommended Trades'].write('A1', 'Ticker', string_format)
# writer.sheets['Recommended Trades'].write('B1', 'Price', string_format)
# writer.sheets['Recommended Trades'].write('C1', 'Market Capitalization', string_format)
# writer.sheets['Recommended Trades'].write('D1', 'Number Of Shares to Buy', string_format
# writer.sheets['Recommended Trades'].set_column('A:A', 20, string_format)
# writer.sheets['Recommended Trades'].set_column('B:B', 20, dollar_format)
# writer.sheets['Recommended Trades'].set_column('C:C', 20, dollar_format)
# writer.sheets['Recommended Trades'].set_column('D:D', 20, integer_format)
```

This code works, but it violates the software principle of "Don't Repeat Yourself".

Let's simplify this by putting it in 2 loops:

In [20]:
```
column_formats = {
                    'A': ['Ticker', string_format],
                    'B': ['Price', dollar_format],
                    'C': ['Market Capitalization', dollar_format],
                    'D': ['Number of Shares to Buy', integer_format]
                    }
```

```
for column in column_formats.keys():
    writer.sheets['Recommended Trades'].set_column(f'{column}:{column}', 20, column_format
    writer.sheets['Recommended Trades'].write(f'{column}1', column_formats[column][0], str
```

Saving Our Excel Output

Saving our Excel file is very easy:

In [21]:
```
writer.save()
```

In [38]:
```
import os
current_path = os.getcwd()
print(current_path)
```

```
C:\Users\user
```

In [4]:
```
nbconvert --to webpdf --allow-chromium-download SP_500_Equal_Weight.ipynb
```

```
  File "C:\Users\user\AppData\Local\Temp/ipykernel_45644/2984715220.py", line 1
    nbconvert --to webpdf --allow-chromium-download SP_500_Equal_Weight.ipynb
                  ^
SyntaxError: invalid syntax
```

In [ ]:
```

```