In [44]:

```
# !pip3 install yfinance
```

In [ ]:

```
# Assignment 1 for MFE407
# Author: Ashutosh Ekade
```

Assignment 1

The goal of this assignment is to write a simple back-testing algorithm. It is not a test for you to come up with new strategies. I have laid out all the instructions. ChatGPT is the best friend in coding. P.S. if this is too easy, happy to give your more difficult tasks:-)

In [1]:

```
import pandas as pd
import numpy as np
import yfinance as yf
import requests
import matplotlib.pyplot as plt
from random import sample
```

```
/Users/ashutosh/Library/Python/3.9/lib/python/site-packages/urllib3/
__init__.py:34: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL
1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.
3'. See: https://github.com/urllib3/urllib3/issues/3020 (https://git
hub.com/urllib3/urllib3/issues/3020)
  warnings.warn(
```

Read in NYSE.txt into a DataFrame, which includes stock tickers and names

In [2]:

```
#read the stock tickers and names into a DataFrame
df = pd.read_csv('~/Downloads/NYSE.txt', sep='\t', header=0) ## please
#Create a list contains all tickers: iterate through stock list and ap
all_tickers_list = df.Symbol.to_list()
```

Using one stock as an example to construct buy-sell strategy

Step 1: Download data and calculate necessary summary statistics

In [3]:

```
#download stock data for the FIRST stock and place in DataFrame (using
first_stock_data = yf.download(all_tickers_list[0])
print(type(first_stock_data))
```

```
[*********************100%%***********************]  1 of 1 completed

<class 'pandas.core.frame.DataFrame'>
```

In [4]:
```
#create column to hold our 90 day rolling standard deviation
# print(first_stock_data.head(5))
first_stock_data['90_SD'] = first_stock_data['Adj Close'].rolling(90).
# print(first_stock_data.head(-10))
```

In [5]:
```
#create a column to hold our 20 day moving average
first_stock_data['20_MA'] = first_stock_data['Adj Close'].rolling(20).
```

Step 2: Create "BUY" signal according to two conditions

In [6]:
```
#BUY Condition 1: create a column which holds a TRUE value if the gap

first_stock_data['Cond_1'] =  np.where((first_stock_data['Low'].shift(
print("Condition 1 was satisfied {} times.".format(first_stock_data['C
```

```
Condition 1 was satisfied 28 times.
```

In [7]:
```
#BUY Condition 2: create a column which holds a TRUE value if the open
first_stock_data['Cond_2'] =  np.where(first_stock_data['Open'] > firs
print("Condition 2 was satisfied {} times.".format(first_stock_data['C
```

```
Condition 2 was satisfied 5522 times.
```

In [8]:
```
#"BUY" signal: create a column that holds a TRUE value if both buy cri
first_stock_data['signal'] =  np.where((first_stock_data['Cond_1'] ==
print("Buy signal {} times.".format(first_stock_data['signal'].sum()))
```

```
Buy signal 15 times.
```

Step 3: Create "SELL" signal according to two conditions

In [9]:
```
#SELL Condition 1: create a column which holds a TRUE value if the gap
first_stock_data['Sell_Cond_1'] =  np.where((first_stock_data['Open']
print("Sell condition 1 was satisfied {} times.".format(first_stock_da
```

```
Sell condition 1 was satisfied 32 times.
```

In [10]:
```
#SELL Condition 2: create a column which holds a TRUE value if the ope
first_stock_data['Sell_Cond_2'] =  np.where(first_stock_data['Open'] <
print("Sell condition 2 was satisfied {} times.".format(first_stock_da
```

```
Sell condition 2 was satisfied 539 times.
```

In [11]:
```
#"SELL" signal: create a column that holds a TRUE value if both sell c
first_stock_data['sell_signal'] =  np.where((first_stock_data['Sell_Co
print("Sell signal {} times.".format(first_stock_data['sell_signal'].s
```

```
Sell signal 0 times.
```

Step 4: Show the results of Trading Algo

In [12]:
```
#calculate daily % return series for stock
first_stock_data['daily_returns'] = first_stock_data['Adj Close'].pct_
```

A Chanllenge: create an indicator which equals to 1 if you hold one share and to -1 if you sell one share.

As a example, if at day 10, the algo tells you to buy a share, and at day 15, the algo tells you to sell a share, then you do not hold any share from day 1 to day 10, and hold one share form day 11 to day 15, and short-sell a share from day 16. How can you constrcut such an indicator in the dataset?

Step 1: Calculate shares of holding for each day

In [13]:
```
# Create a column contains your holding of shares according to "BUY" a
# first_stock_data['Share'] = np.where(first_stock_data["signal"]==Tru
buy_indices = first_stock_data[first_stock_data['signal']==True].inde
sell_indices = first_stock_data[first_stock_data['sell_signal']==True]
indicator = []
sig_to_append = 0
for idx in first_stock_data.index:
    if idx in buy_indices:
        sig_to_append = 1
    elif idx in sell_indices:
        sig_to_append = -1
    else:
         pass
    indicator.append(sig_to_append)
first_stock_data['Share'] = indicator
```

In [14]:
```
# Verify. The new column is "Share" (verificaiton is an important ste
first_stock_data[['daily_returns', 'signal', 'sell_signal', 'Share']]
```

Out[14]:

	daily_returns	signal	sell_signal	Share
Date				
2016-08-25	0.011992	False	False	1
2016-08-26	0.001270	False	False	1
2016-08-29	0.006340	False	False	1
2016-08-30	-0.010080	False	False	1

In [15]:
```
print(first_stock_data["Share"].value_counts())
```

```
Share
1    4828
0    1252
Name: count, dtype: int64
```

Step 2: Calculate strategic returns according to your holdings of shares and daily stock returns

In [16]:
```
#create a strategy return series by using the daily stock returns mutl
first_stock_data['Rets'] = first_stock_data['daily_returns'] * first_
```

In [17]:

```
# Verify again
first_stock_data[['daily_returns', 'signal', 'sell_signal', 'Share', '
```

Out[17]:

	daily_returns	signal	sell_signal	Share	Rets
Date					
2016-08-25	0.011992	False	False	1	0.011992
2016-08-26	0.001270	False	False	1	0.001270
2016-08-29	0.006340	False	False	1	0.006340
2016-08-30	-0.010080	False	False	1	-0.010080

Good Job! Apply the strategy to all stocks in stocks_list

In [18]:
```
#create empty list to hold our return series DataFrame for each stock
frames = []
# lets randomly select 200 stocks from the list
sample_stocks = sample(all_tickers_list, 200)
for stock in sample_stocks:
    try:
        ### Copy Paste Previous Code For One Stock ###
        first_stock_data = yf.download(stock)
        first_stock_data['90_SD'] = first_stock_data['Adj Close'].roll
        first_stock_data['20_MA'] = first_stock_data['Adj Close'].roll
        first_stock_data['Cond_1'] =  np.where((first_stock_data['Low'
        first_stock_data['Cond_2'] =  np.where(first_stock_data['Open'
        first_stock_data['signal'] =  np.where((first_stock_data['Conc
        first_stock_data['Sell_Cond_1'] =  np.where((first_stock_data[
        first_stock_data['Sell_Cond_2'] =  np.where(first_stock_data['
        first_stock_data['sell_signal'] =  np.where((first_stock_data[
        first_stock_data['daily_returns'] = first_stock_data['Adj Clos
        buy_indices = first_stock_data[first_stock_data['signal']==Tru
        sell_indices = first_stock_data[first_stock_data['sell_signal'
        indicator = []
        sig_to_append = 0
        for idx in first_stock_data.index:
            if idx in buy_indices:
                sig_to_append = 1
            elif idx in sell_indices:
                sig_to_append = -1
            else:
                pass
            indicator.append(sig_to_append)
        first_stock_data['Share'] = indicator
        first_stock_data['Rets'] = first_stock_data['daily_returns'] *
        #append the strategy return series to our list
        frames.append(first_stock_data['Rets'])
    except:
        pass
```

```
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed

1 Failed download:
['VR-A']: Exception('%ticker%: No timezone found, symbol may be deli
sted')
[*********************100%%**********************]  1 of 1 completed

1 Failed download:
['HFC']: Exception('%ticker%: No timezone found, symbol may be delis
ted')
[*********************100%%**********************]  1 of 1 completed

1 Failed download:
['OAK-A']: Exception('%ticker%: No timezone found, symbol may be del
isted')
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
```

Plot cummulative returns of strategy

Step 1: Calculate cumulative returns

In [19]:

```
#concatenate the individual DataFrames held in our list- and do it alc
new_df = pd.concat(frames, ignore_index=False, axis=1, keys=sample_stc
```

```
/var/folders/44/r2pt84y14r968g9_vxmvrlxh0000gn/T/ipykernel_64955/357
7510563.py:2: FutureWarning: The behavior of pd.concat with len(key
s) != len(objs) is deprecated. In a future version this will raise i
nstead of truncating to the smaller of the two sequences
  new_df = pd.concat(frames, ignore_index=False, axis=1, keys=sample
_stocks)
```

In [20]:

```
#create a column to hold the sum of all the individual daily strategy
new_df['daily_returns_sum'] = new_df.sum(axis=1, numeric_only=True)
```

In [22]:

```
#fill 'NaNs' with zeros to allow our "count" function below to work pr
print(new_df.daily_returns_sum.sum())
```

```
297.00562770480803
```

In [23]:

```
#create a column that hold the count of the number of stocks that were
#we minus one from it so that we dont count the "Total" column we adde
new_df['count'] = new_df.count(axis=1)
```

In [24]:

```
print(new_df.head())
```

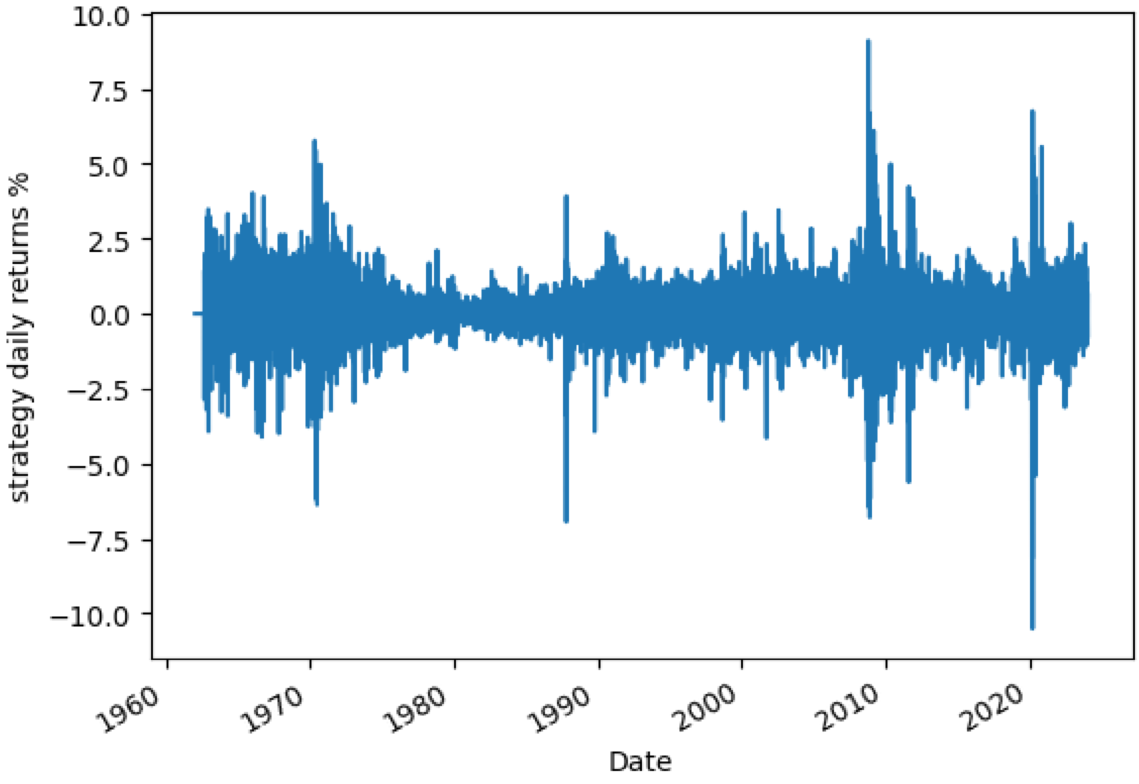
```
            BGH  VR-A  HFC  OAK-A  NKE  IHD  DDR-K  UMH  ASH  CP  
...  NGVC  \
Date                                                                
...                                                                 
1962-01-02  NaN   NaN  NaN    NaN  NaN  NaN    NaN  NaN  NaN NaN  
...   NaN
1962-01-03  NaN   NaN  NaN    NaN  NaN  NaN    NaN  NaN  NaN NaN  
...   NaN
1962-01-04  NaN   NaN  NaN    NaN  NaN  NaN    NaN  NaN  NaN NaN  
...   NaN
1962-01-05  NaN   NaN  NaN    NaN  NaN  NaN    NaN  NaN  NaN NaN  
...   NaN
1962-01-08  NaN   NaN  NaN    NaN  NaN  NaN    NaN  NaN  NaN NaN  
...   NaN

            VSH  CHH  BDC-B  SBH  PSA-Y  DNOW  GHY  daily_returns_su
m  count
Date                                                                
1962-01-02  NaN  NaN    NaN  NaN    NaN   NaN  NaN                0.
0      1
1962-01-03  NaN  NaN    NaN  NaN    NaN   NaN  NaN                0.
0      2
1962-01-04  NaN  NaN    NaN  NaN    NaN   NaN  NaN                0.
0      2
1962-01-05  NaN  NaN    NaN  NaN    NaN   NaN  NaN                0.
0      2
1962-01-08  NaN  NaN    NaN  NaN    NaN   NaN  NaN                0.
0      2

[5 rows x 122 columns]
```

In [26]:
```
#create a column that divides the "total" strategy return each day by
new_df['daily_equally_wt_returns'] = new_df['daily_returns_sum'] / nev
```

Step 2: Plot

In [35]:
```
#plot the strategy returns
(new_df['daily_equally_wt_returns'] * 100).plot(ylabel="strategy daily
```

Out[35]:
```
<Axes: xlabel='Date', ylabel='strategy daily returns %'>
```

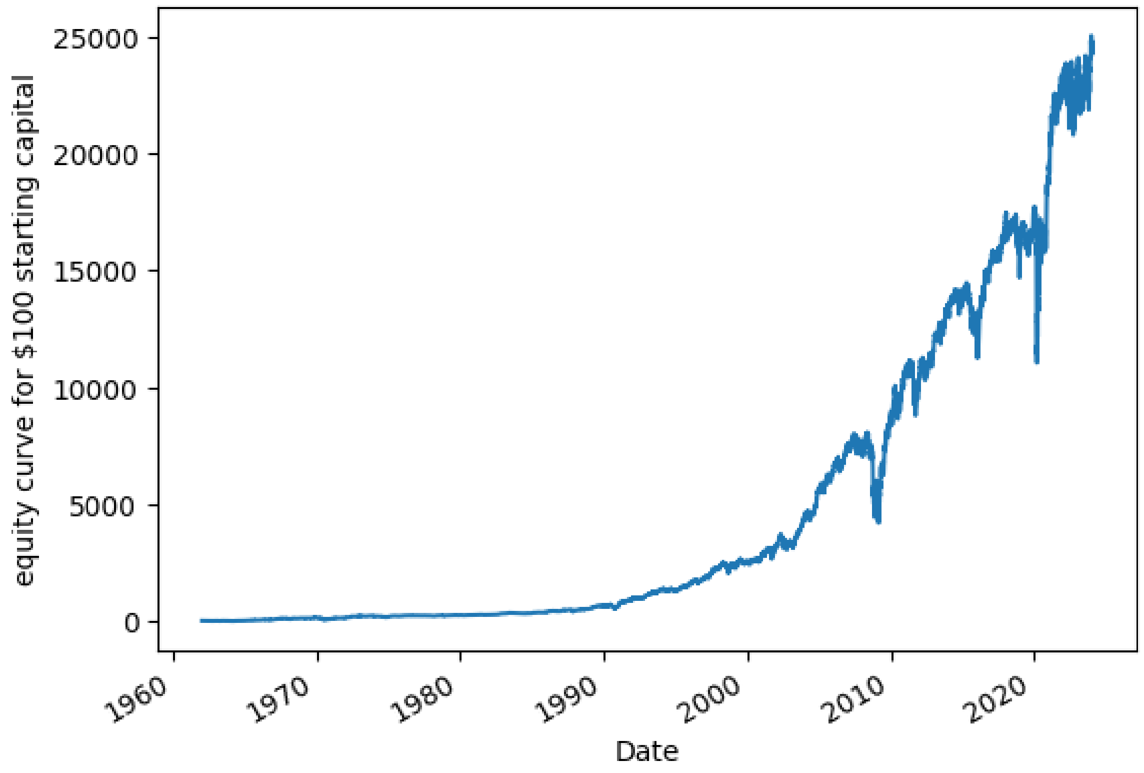


In [42]:

```
new_df['daily_cum_returns'] = (1 + new_df['daily_equally_wt_returns'])
(new_df['daily_cum_returns'] * 100).plot(ylabel="equity curve for $10(
```

Out[42]:

```
<Axes: xlabel='Date', ylabel='equity curve for $100 starting capita
l'>
```



Step 3: Calculate annual returns and sharpe ratio for your strategy

In [38]:

```
# Annual Return
annual_returns = new_df['daily_equally_wt_returns'].mean() * 255
print("Annual returns of the strategy are {:.2f} %".format(annual_retu
```

```
Annual returns of the strategy are 9.81 %
```

In [39]:

```
# Sharpe Ratio (risk free element excluded for simplicity)
def sharpe(returns):
    ann = returns.mean() * 255
    std_dev_ann = returns.std() * np.sqrt(255)
    return ann / std_dev_ann

strat_sharpe = sharpe(new_df['daily_equally_wt_returns'])
print("Strategy sharpe ratio is {:.2f}".format(strat_sharpe))
```

```
Strategy sharpe ratio is 0.77
```

Congradulation!!! You have written the first backtesting code yourself!!!