In [1]:
```
from quickfs import QuickFS
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json
import yfinance as yf
from datetime import date
```

In [2]:
```
# Forward Testing Years
year2020 = '2020'
year2015 = '2015'

start_yf = year2015 + '-01-01'
end_yf = year2020 + '-01-01'
```

In [3]:
```
exchange_input = input('Choose either NASDAQ, NYSE, NYSEAMERICAN: ').strip().upper()
```

```
Choose either NASDAQ, NYSE, NYSEAMERICAN: NYSE
```

In [4]:
```
exchange_input
```

Out[4]:
```
'NYSE'
```

In [5]:
```
#csv_name = exchange_input  + '_' + year2010 + '-' + year2015 + '.csv'
```

In [6]:
```
#sv_name = 'NYSE_ROE_2010-2015.csv'
```

In [7]:
```
csv_name = 'NYSE_ROE_2015-2020.csv'
```

In [8]:
```
print(csv_name)
```

```
NYSE_ROE_2015-2020.csv
```

In [9]:
```
def read_csv(filename=csv_name):
    df = pd.read_csv(csv_name, converters={'roe_median': pd.eval,
                                       'price_to_sales': pd.eval,
                                       'roic_5yr_avg': pd.eval,
                                      'revenue_cagr_10': pd.eval,
                                       }
                                        ,index_col=0)
    return df
```

In [10]:
```
df1 = read_csv()
```

In [11]:
```
def filter1_list(df_clean):
    df_clean['roic_5yr_avg'] = df_clean['roic_5yr_avg'].apply(np.mean)
    df_clean['roe_median'] = df_clean['roe_median'].apply(np.mean)
    df_clean['mean_ps'] = df_clean['price_to_sales'].apply(np.mean)
    df_clean['revenue_cagr_10'] = df_clean['revenue_cagr_10'].apply(np.mean)

    mid_caps = df_clean[df_clean['roe_median'] > 0.2].copy()
    mid_caps = mid_caps[(mid_caps['mean_ps']>0) & (mid_caps['mean_ps'] < 1)]

    sorted_mid_caps = mid_caps[['roe_median',
```

```
                            'mean_ps',
                            'roic_5yr_avg',
                            'revenue_cagr_10',
                            ]].sort_values('mean_ps', ascending=True).copy()

    sorted_mid_caps[sorted_mid_caps['mean_ps'] < 1]
    sorted_mid_caps.reset_index(inplace=True)
    sorted_mid_caps.rename(columns={'index':'stocks'}, inplace=True)

    sorted_mid_caps = sorted_mid_caps[sorted_mid_caps['revenue_cagr_10']>0.01]
    sorted_mid_caps = sorted_mid_caps[sorted_mid_caps['roic_5yr_avg']>0.2]
    sorted_mid_caps['stocks'] = np.where(sorted_mid_caps.stocks.str.contains(':US') ==
    sorted_mid_caps.set_index(['stocks'], inplace=True)
    yf_stocks = sorted_mid_caps.index.tolist()

    return yf_stocks
```

In [12]:

```
filteredOnce = filter1_list(df1)
```

In [13]:

```
len(filteredOnce)
```

Out[13]:

```
16
```

In [14]:

```
print(filteredOnce)
```

```
['BXC', 'SYX', 'ABC', 'IDT', 'BCC', 'NSP', 'BLDR', 'RS', 'GNE', 'AMN', 'MATX', 'OLN',
'IIIN', 'AFG', 'DAC', 'DKS']
```

In [15]:

```
def filter2_cagr_list(filtered_list):
    print(F'Getting CAGR ticker data for year {start_yf} to {end_yf}')
    close = yf.download(filtered_list, start=start_yf, end=end_yf)['Adj Close']
    close = close.ffill()
    #close.dropna(axis=1, inplace=True)
    log_returns = np.log(close.div(close.shift(1)))
    #print(log_returns)
    CAGR = np.exp(log_returns.mean() *252*5 - 1) #multiply by 5 because 5 years from s
    #print(CAGR)
    CAGR = CAGR.sort_values(ascending=False)[:].index
    CAGR = CAGR.tolist()
    return CAGR
```

In [16]:

```
yf_cagr_filter = filter2_cagr_list(filteredOnce)
```

```
Getting CAGR ticker data for year 2015-01-01 to 2020-01-01
[*********************100%***********************]  16 of 16 completed

1 Failed download:
- SYX: No timezone found, symbol may be delisted
```

In [17]:

```
print('List for the exchange {}'.format(exchange_input))
```

```
List for the exchange NYSE
```

In [18]:

```
print(yf_cagr_filter)
```

```
['NSP', 'BLDR', 'AMN', 'AFG', 'RS', 'GNE', 'MATX', 'BXC', 'DKS', 'IIIN', 'BCC', 'AB
C', 'OLN', 'IDT', 'DAC', 'SYX']
```

In [19]:

```
len(yf_cagr_filter)
```

Out[19]:
```
16
```

In [20]:
```
fwd_start = '2020-01-01'
fwd_end = '2023-04-15'
```

In [21]:
```
print('We should now test the performance from the time period ' + fwd_start + ' to '
```

```
We should now test the performance from the time period 2020-01-01 to 2023-04-15
```

In [22]:
```
def strategy_fwd(tickers):
    '''Calculates the performance of a ticker or list of tickers on an adjusted close
    tickers == either ticker list or a single symbol'''

    forward_test = yf.download(tickers, start=fwd_start, end=fwd_end)['Adj Close']

    returns = forward_test.pct_change()
    #returns.dropna(inplace=True)
    try:
        strategy_returns = returns.mean(axis=1)
        strategy_returns.name = 'Strategy'
    except ValueError:
        strategy_returns = returns
        strategy_returns.name = 'Benchmark'

    strategy_returns.dropna(inplace=True)


    strategy_returns = strategy_returns.add(1).cumprod().mul(100)
    return strategy_returns
```

In [23]:
```
SPY = strategy_fwd('SPY')
```

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

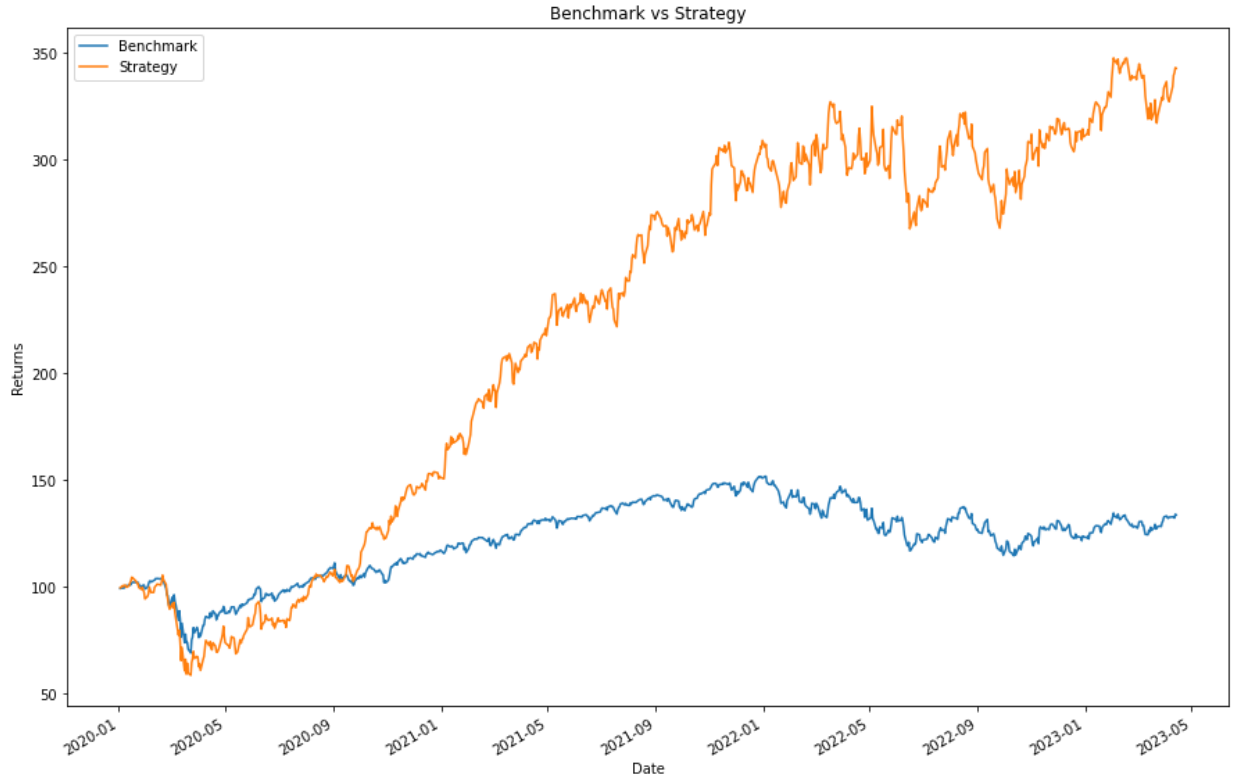
In [24]:
```
Strat = strategy_fwd(yf_cagr_filter)
```

```
[*********************100%***********************]  16 of 16 completed

1 Failed download:
- SYX: No timezone found, symbol may be delisted
```

In [25]:
```
def plot_compare(perf1, perf2):
    perf1.plot(legend=True, figsize=(15,10))
    perf2.plot(legend=True)
    plt.title("{} vs {}".format(perf1.name, perf2.name))
    plt.ylabel("Returns")
    return plt.show()
```

In [26]:
```
plot_compare(SPY, Strat)
```



In [27]:
```
Outperformance = Strat[-1] - SPY[-1]
Outperformance
```

Out[27]:
```
209.27372593526135
```

In [28]:
```
# 3. Basically 8 Years and 4/12 months = 0.33
Outperformance/8.333
```

Out[28]:
```
25.113851666298014
```

In [47]:
```
def filter2_cagr_list(filtered_list):
    print(F'Getting CAGR ticker data for year {start_yf} to {end_yf}')
    close = yf.download(filtered_list, start=start_yf, end=end_yf)['Adj Close']
    #close.dropna(axis=1, inplace=True)
    log_returns = np.log(close.div(close.shift(1)))
    #print(log_returns)
    CAGR = np.exp(log_returns.mean() *252*5 - 1) #multiply by 5 because 5 years from s
    #print(CAGR)
    CAGR = CAGR.sort_values(ascending=False)[:].index
    CAGR = CAGR.tolist()
    return CAGR
```

In [48]:
```
yf_cagr_filter = filter2_cagr_list(filteredOnce)
```

```
Getting CAGR ticker data for year 2015-01-01 to 2020-01-01
[*********************100%***********************]  16 of 16 completed

1 Failed download:
- SYX: No timezone found, symbol may be delisted
```

In [49]:
```
print('List for the exchange {}'.format(exchange_input))
```

```
List for the exchange NYSE
```

In [50]:
```
print(yf_cagr_filter[:15])
```

```
['NSP', 'BLDR', 'AMN', 'AFG', 'RS', 'GNE', 'MATX', 'BXC', 'DKS', 'IIIN', 'BCC', 'AB
C', 'OLN', 'IDT', 'DAC']
```

In [51]:
```
stock_test = yf_cagr_filter[:15]
```

In [52]:
```
print('We should now test the performance from the time period ' + fwd_start + ' to '
```

```
We should now test the performance from the time period 2020-01-01 to 2023-04-15
```

In [53]:
```
def strategy_fwd(tickers):
    '''Calculates the performance of a ticker or list of tickers on an adjusted close
    tickers == either ticker list or a single symbol'''

    forward_test = yf.download(tickers, start=fwd_start, end=fwd_end)['Adj Close']
    forward_test = forward_test.ffill()
    returns = forward_test.pct_change()
    returns.dropna(inplace=True)
    try:
        strategy_returns = returns.mean(axis=1)
        strategy_returns.name = 'Strategy'
    except ValueError:
        strategy_returns = returns
        strategy_returns.name = 'Benchmark'

    strategy_returns.dropna(inplace=True)


    strategy_returns = strategy_returns.add(1).cumprod().mul(100)
    return strategy_returns
```

In [54]:
```
SPY = strategy_fwd('SPY')
```

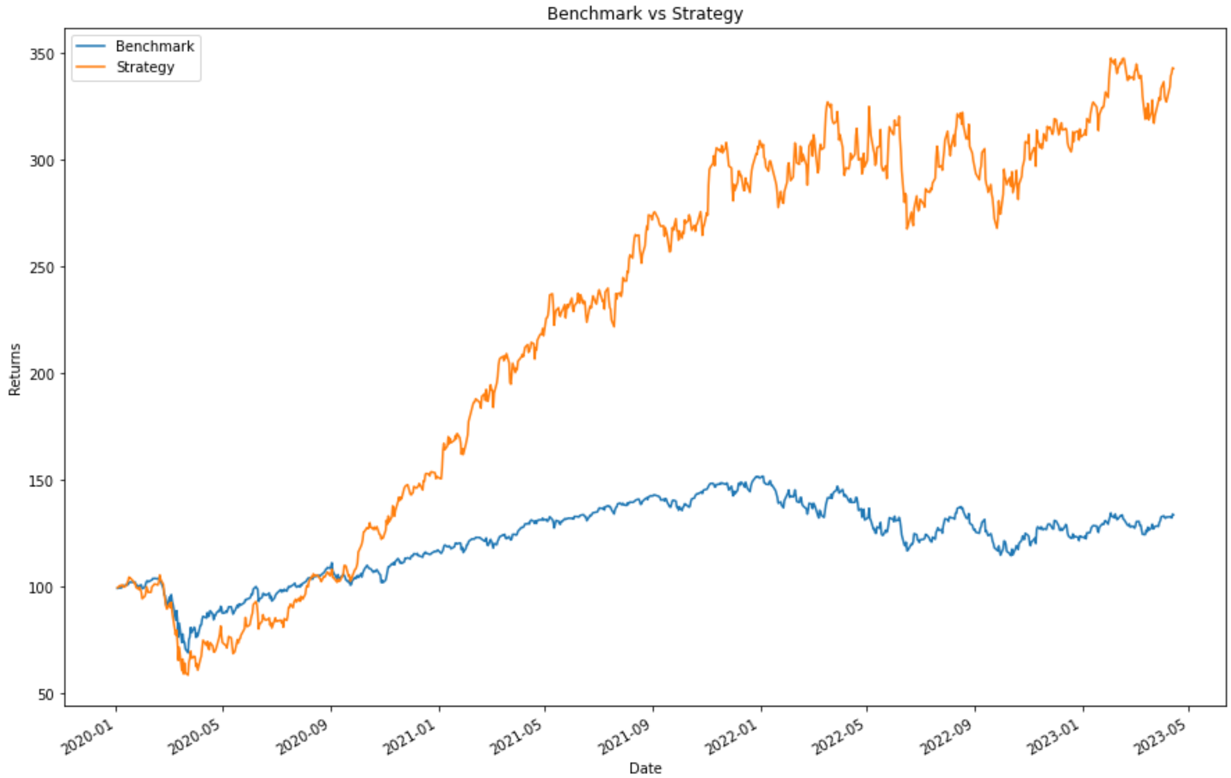
```
[*********************100%***********************]  1 of 1 completed
```

In [55]:
```
Strat = strategy_fwd(stock_test)
```

```
[*********************100%***********************]  15 of 15 completed
```

In [56]:
```
def plot_compare(perf1, perf2):
    perf1.plot(legend=True, figsize=(15,10))
    perf2.plot(legend=True)
    plt.title("{} vs {}".format(perf1.name, perf2.name))
    plt.ylabel("Returns")
    return plt.show()
```

In [57]:
```
plot_compare(SPY, Strat)
```



In [58]:
```
Outperformance = Strat[-1] - SPY[-1]
Outperformance
```

Out[58]: 209.2737455465554

In [59]:
```
# 3. Basically 8 Years and 4/12 months = 0.33
Outperformance/8.333
```

Out[59]: 25.11385401974744