In [1]:
```
from IPython.display import display
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
```

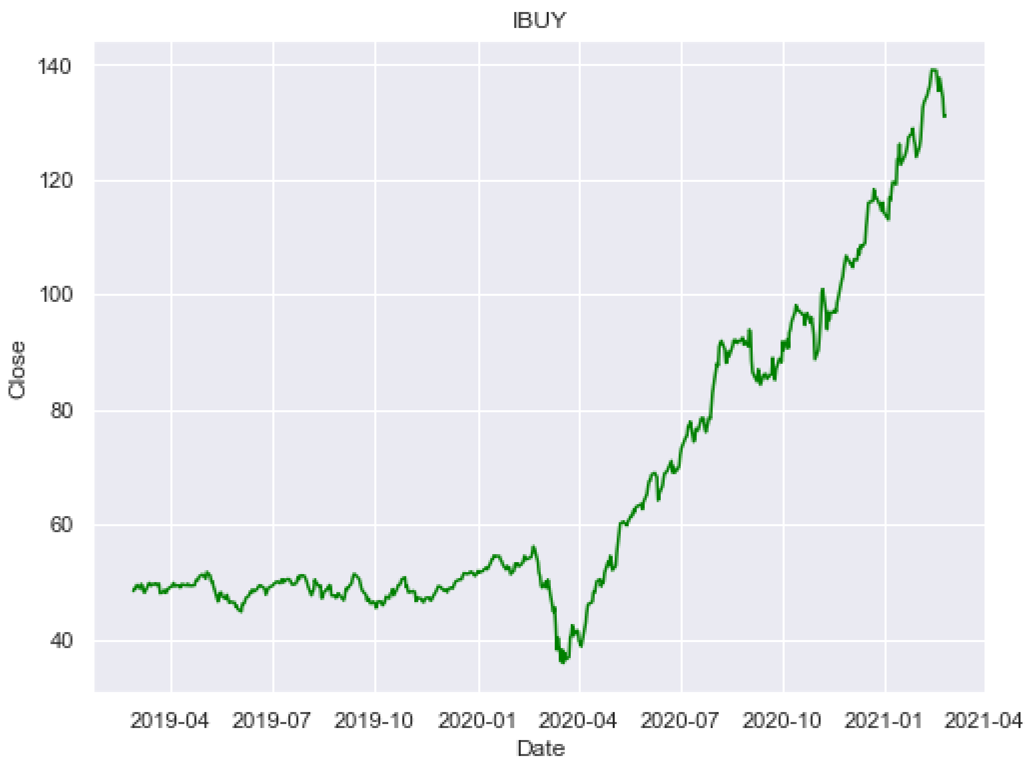
First, We will ask for a user to input their desired number of stock assets in their portfolio and then ask for them to specify which ones they are interested in.

In [3]:
```
#user input for stocks
stonks = []
n = int(input("Enter number of Stocks in Portfolio: "))
for i in range(0, n):
    istonks = input("Enter Ticker")
    stonks.append(istonks)
print(stonks)

#stonks = ['IBUY', 'ADBE', 'JNJ', 'JPM', 'J', 'NEE', 'AMAT', 'WDC', 'IIPR', 'GOOGL', 'BSX', 'WMT', 'LOW','SWBI', 'WU']
#stonks= ['LQD', 'GOVT', 'USIG', 'SHY' ]
```

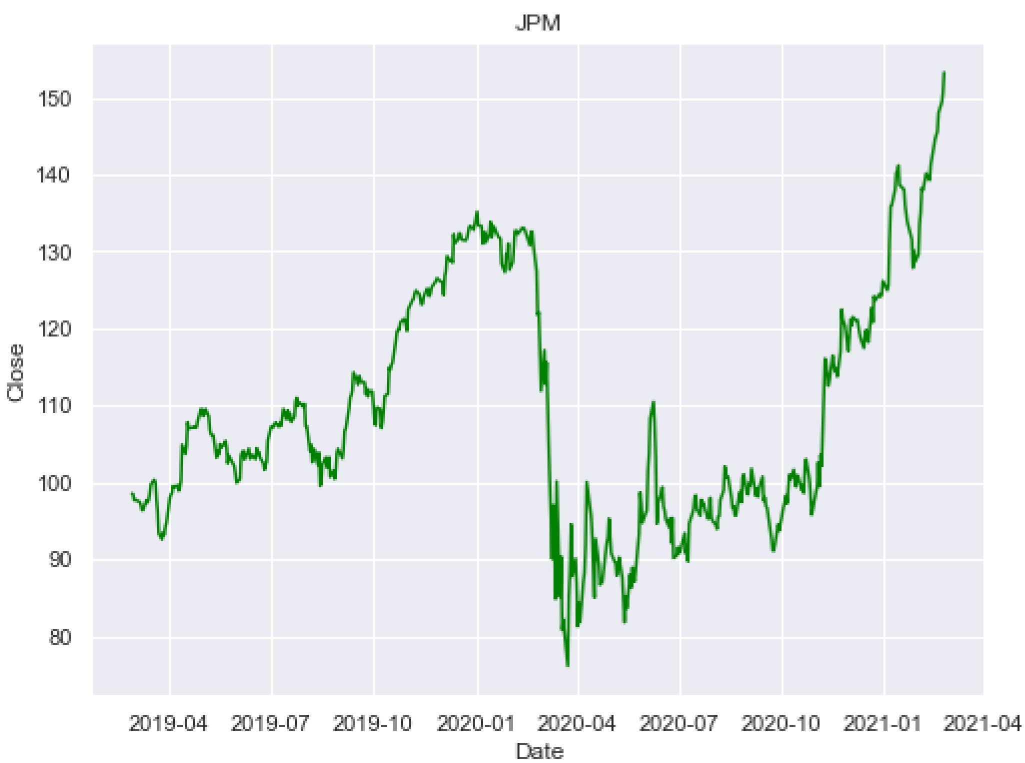
```
Enter number of Stocks in Portfolio: 4
Enter TickerIBUY
Enter TickerADBE
Enter TickerJPM
Enter TickerGOOGL
['IBUY', 'ADBE', 'JPM', 'GOOGL']
```

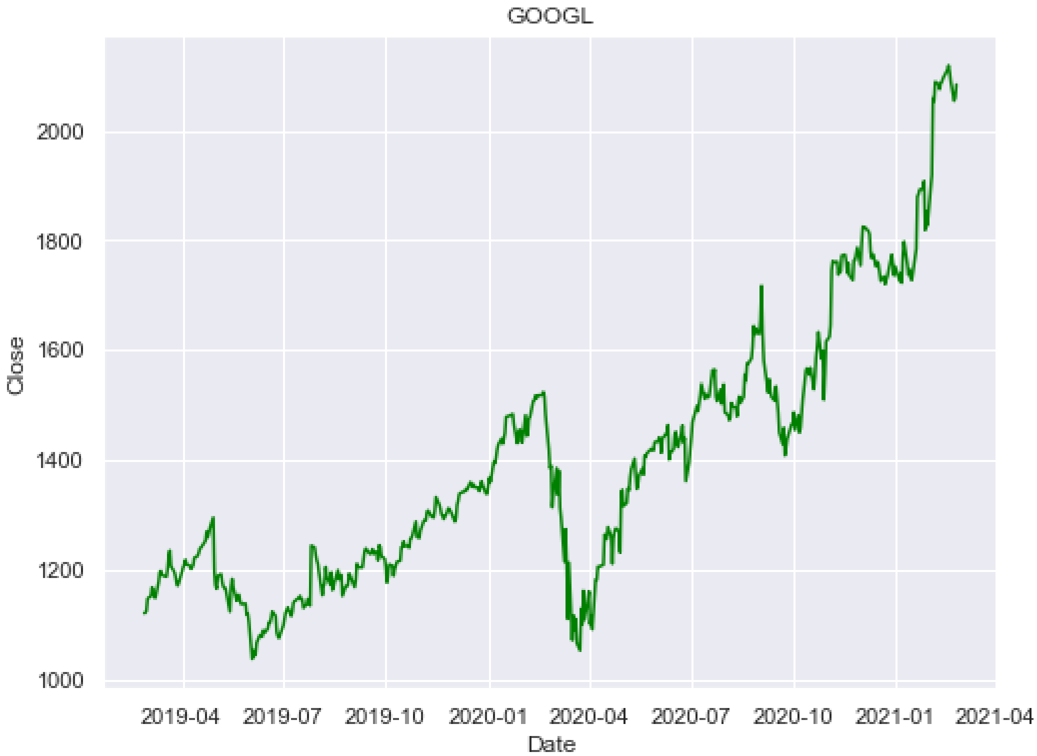
In [4]:
```
per = input('Enter the Period of Time for Stock Data: 10y, 5y, 2y,1y, 6mo, et
c')
data = yf.download(stonks, period = per, interval = "1d", group_by= 'ticker',
auto_adjust =True, prepost = False)
data[stonks[0]]
for i in range(len(stonks)):
    x= data[stonks[i]]
    x = x[~data.index.duplicated(keep='first')]
    x = x.head(-1)
    price_plt = x['Close'].reset_index()
    plt.figure(i)
    plt.figure(figsize=(8, 6))
    sns.lineplot(x= 'Date', y= 'Close', data= price_plt, color= "green").set_t
itle(stonks[i])
price_plt.info()
```

```
Enter the Period of Time for Stock Data: 10y, 5y, 2y,1y, 6mo, etc2y
[*********************100%***********************]  4 of 4 completed
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 504 entries, 0 to 503
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    504 non-null    datetime64[ns]
 1   Close   504 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 8.0 KB

<Figure size 432x288 with 0 Axes>
```




ADBE
500
450
400
350
300
250
Close
2019-04 2019-07 2019-10 2020-01 2020-04 2020-07 2020-10 2021-01 2021-04
Date


JPM
150
140
130
120
110
100
90
80
Close
2019-04 2019-07 2019-10 2020-01 2020-04 2020-07 2020-10 2021-01 2021-04
Date


GOOGL
Close
2000
1800
1600
1400
1200
1000
2019-04
2019-07
2019-10
2020-01
2020-04
2020-07
2020-10
2021-01
2021-04
Date

In [6]:
```
data = data.head(-1)
print(data)
```

```
                  ADBE                                                    \
                  Open        High         Low       Close   Volume   
Date                                                                  
2019-02-06  255.059998  255.929993  250.710007  254.350006  2539100   
2019-02-07  251.330002  254.309998  250.279999  253.740005  2099300   
2019-02-08  251.389999  257.049988  250.639999  257.000000  2756800   
2019-02-11  258.890015  259.899994  256.190002  258.390015  3538100   
2019-02-12  260.149994  262.250000  258.670013  261.369995  2518400   
...                ...         ...         ...         ...      ...   
2021-01-29  462.170013  465.000000  455.109985  458.769989  3060400   
2021-02-01  462.279999  474.799988  459.820007  470.000000  2554800   
2021-02-02  473.649994  487.369995  472.549988  484.929993  3022000   
2021-02-03  487.089996  488.850006  479.170013  481.920013  2146900   
2021-02-04  484.220001  489.880005  481.920013  489.380005  2004000   

                  AMAT                                                ...  \
                  Open        High         Low       Close    Volume  ...   
Date                                                                  ...   
2019-02-06   38.069387   39.445271   37.982183   39.028629  21058100  ...   
2019-02-07   38.495721   39.077083   38.331004   38.679817  13395500  ...   
2019-02-08   38.079079   38.563546   37.526787   38.544167   9891300  ...   
2019-02-11   38.728264   38.941426   38.108147   38.660439   9969500  ...   
2019-02-12   39.125521   39.619675   39.028626   39.425888   8911900  ...   
...                ...         ...         ...         ...       ...  ...   
2021-01-29  100.230003  100.459999   96.070000   96.680000   7933800  ...   
2021-02-01   99.250000  102.150002   97.680000  101.209999   9083600  ...   
2021-02-02  102.989998  103.940002  101.769997  103.589996   5108600  ...   
2021-02-03  104.320000  104.410004   99.820000   99.870003   5399200  ...   
2021-02-04  100.209999  103.730003  100.209999  103.239998   6355100  ...   

                  WDC                                                  SWBI  
\
                 Open       High        Low      Close    Volume       Open   
Date                                                                          
2019-02-06  45.006200  46.202065  44.996711  45.556679   5251600   9.833775   
2019-02-07  44.882826  45.082135  43.136486  43.525616   7045600   9.543423   
2019-02-08  42.187385  44.351326  41.807745  44.332344   7119700   9.398247   
2019-02-11  44.398784  44.493696  43.288342  43.592052   4173700   9.222508   
2019-02-12  44.095074  45.395340  43.990672  44.854351   4634000   9.298916   
...               ...        ...        ...        ...       ...        ...   
2021-01-29  58.660000  60.680000  55.119999  56.430000  15480300  16.850000   
2021-02-01  56.990002  57.959999  56.430000  57.590000   4600300  16.670000   
2021-02-02  58.599998  59.009998  57.490002  57.910000   4025000  17.000000   
2021-02-03  57.980000  58.709999  57.220001  57.700001   3038500  16.830000   
2021-02-04  58.240002  59.410000  58.060001  59.259998   3299300  17.110001   


                 High        Low      Close   Volume  
Date                                                  
2019-02-06   9.864338   9.528142   9.566346   712558  
2019-02-07   9.619830   9.237789   9.390606   601582  
2019-02-08   9.413529   9.161381   9.207226   469921  
2019-02-11   9.367684   9.176663   9.260713   303523  
2019-02-12   9.482296   9.291274   9.405888   401489  
...               ...        ...        ...      ...  
2021-01-29  17.209999  16.290001  16.559999  2000000  
2021-02-01  17.030001  16.340000  16.830000  1949800  
```

```
2021-02-02  17.180000  16.420000  16.770000  1785500
2021-02-03  17.150000  16.559999  16.959999  2243200
2021-02-04  17.840000  17.020000  17.830000  2562800

[504 rows x 25 columns]
```

Below is a normalized graph of the stock performance compared with each other.

In [5]:
```
stonks_df = pd.DataFrame()
for i in range(len(stonks)):
    stonks_df[stonks[i]]= data[stonks[i]]['Close']
#print(stonks_df)
stonks_df.dropna(inplace=True)
Normalized_stonks = (stonks_df/stonks_df.iloc[0])
#print(Normalized_stonks)
Normalized_stonks.info()
Normalized_stonks.plot.line(figsize=(12, 10))
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 505 entries, 2019-02-26 to 2021-02-25
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   IBUY    505 non-null    float64
 1   ADBE    505 non-null    float64
 2   JPM     505 non-null    float64
 3   GOOGL   505 non-null    float64
dtypes: float64(4)
memory usage: 19.7 KB
```

Out[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2a8a265f5c0>
```

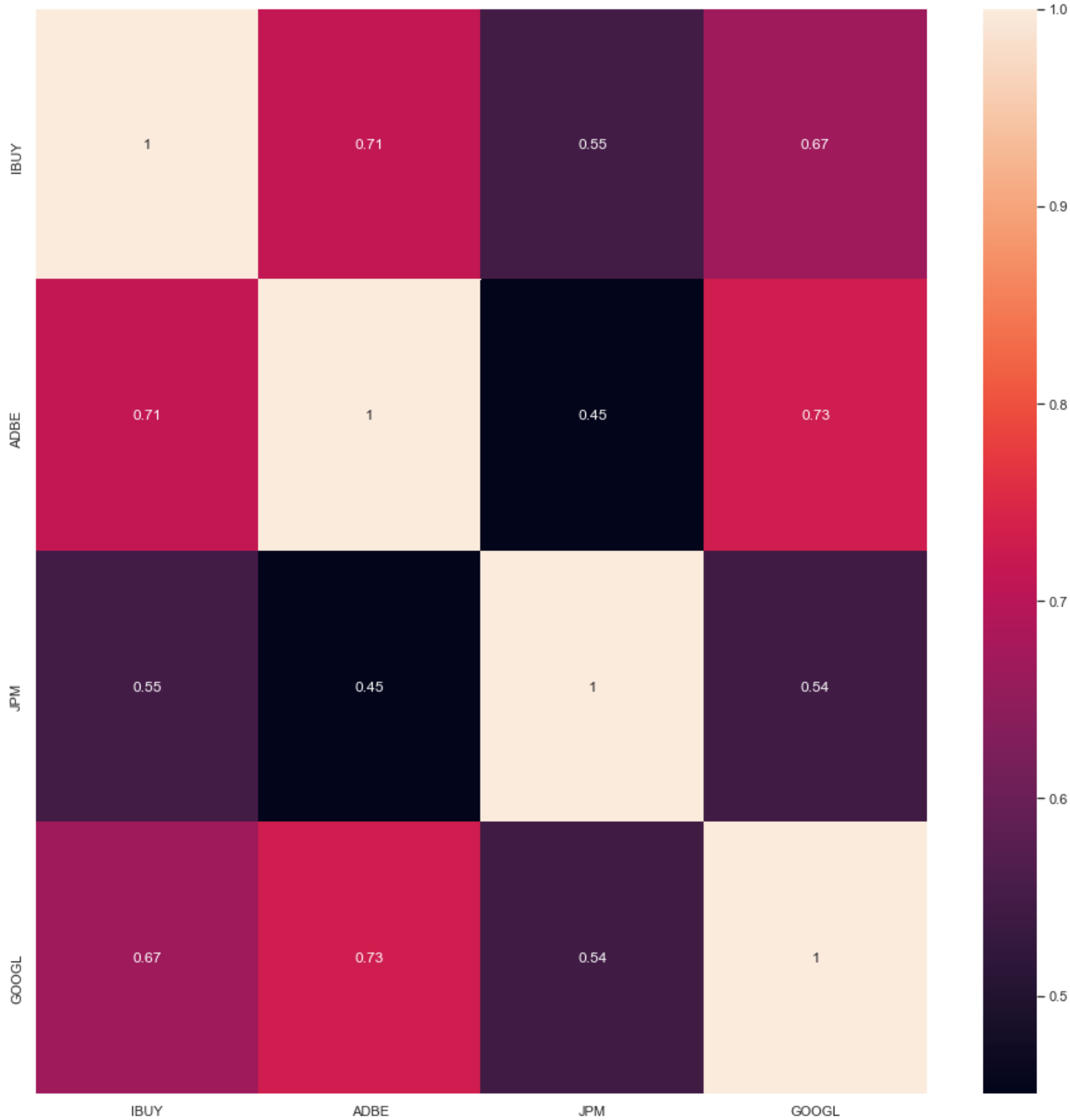
In [6]:
```
#print(stonks_df)
stonks_returns= np.log(stonks_df/stonks_df.shift(1))
stonks_returns.dropna(inplace=True)
#print(stonks_returns)
info_df= pd.DataFrame()
data_rate= 365
info_df['Annualized Returns(%)'] =stonks_returns.mean() * data_rate *100
info_df['Annualized Volatility(%)'] = stonks_returns.std() * np.sqrt(data_rate
)*100
info_df['Sharpe Ratio'] = info_df['Annualized Returns(%)'] /info_df['Annualize
d Volatility(%)']
info_df.style.bar(color=['red','green'], align='zero')
#print(stonks_returns)
```

Out[6]:

	Annualized Returns(%)	Annualized Volatility(%)	Sharpe Ratio
IBUY	69.288612	37.307076	1.857251
ADBE	40.847829	45.774983	0.892361
JPM	30.942138	49.217494	0.628682
GOOGL	42.436232	39.191927	1.082780

In [9]:
```
#Sortino Ratio
```

In [7]:
```
corr_matrix= stonks_returns.corr()
fig, ax = plt.subplots(figsize=(16,16))
sns.heatmap(corr_matrix, annot = True)
plt.show()
```



In [8]:

```
sim_num= 5000
monte_weights_df_list= pd.DataFrame()
port_return_sim = []
port_vol_sim =[]
all_weights = np.zeros((sim_num, len(stonks)))
sharpe_arr = np.zeros(sim_num)
for i in range(sim_num):
    monte_weights = np.random.random(len(stonks))
    monte_weights /= np.sum(monte_weights)
    all_weights[i,:] = monte_weights
    port_return_sim.append(np.sum(stonks_returns[stonks].mean()* monte_weights
)* 365)
    port_vol_sim.append(np.sqrt(np.dot(monte_weights.T,np.dot(stonks_returns[s
tonks].cov()*365,
 monte_weights))))
    sharpe_arr[i] = port_return_sim[i]/port_vol_sim[i]
port_return_sim = np.array(port_return_sim)
port_vol_sim=np.array(port_vol_sim)
sharpe_arr=np.array(sharpe_arr)
#print(port_return_sim)
#print(port_vol_sim)
#print(all_weights)
#print(sharpe_arr)
```

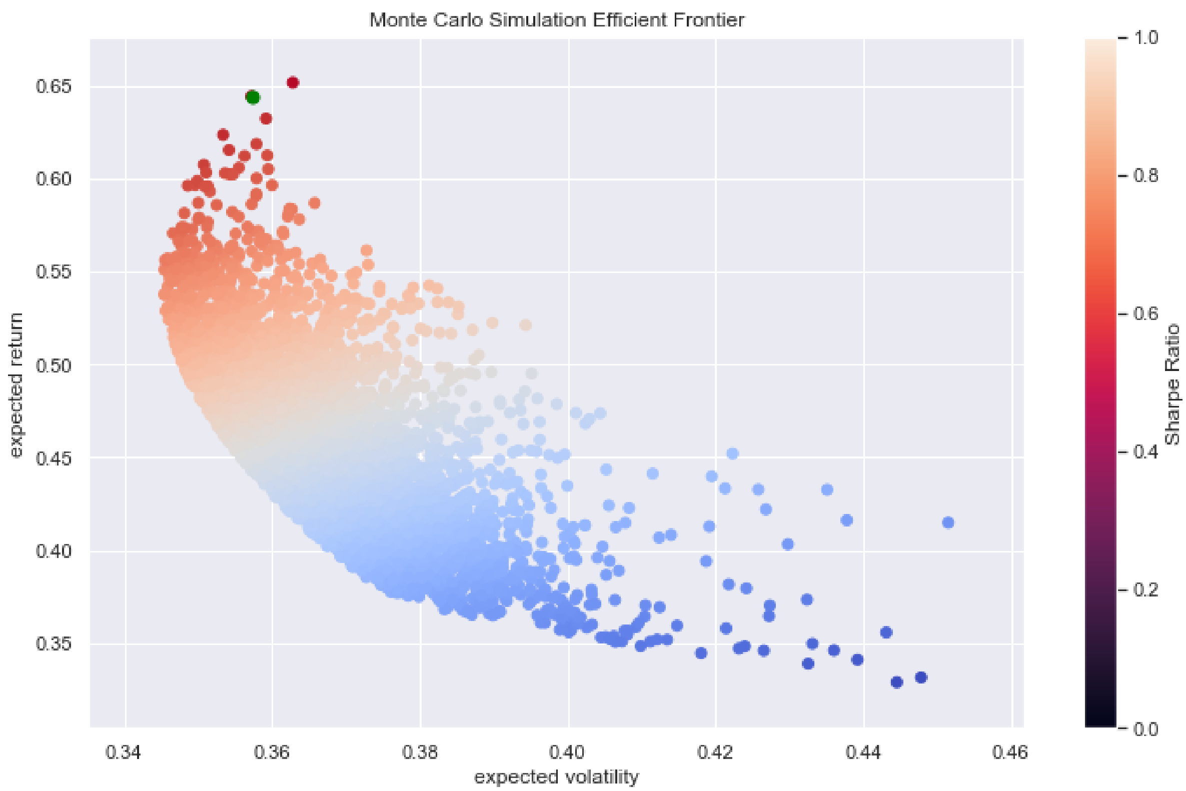
In [9]:

```
print('Max Sharpe Ratio is: {}'.format(sharpe_arr.max()))
print('Max Sharpe Ratio index is: {}'.format(sharpe_arr.argmax()))
print(stonks)
print(all_weights[sharpe_arr.argmax(),:])
max_ret=port_return_sim[sharpe_arr.argmax()]
max_vol=port_vol_sim[sharpe_arr.argmax()]
print(max_ret)
print(max_vol)
print(port_return_sim.max())
```

```
Max Sharpe Ratio is: 1.8031311244884796
Max Sharpe Ratio index is: 4909
['IBUY', 'ADBE', 'JPM', 'GOOGL']
[0.82264552 0.00783881 0.00900001 0.16051566]
0.6441032422663064
0.35721375640333897
0.6514956555192692
```

In [10]:
```
fig8 = plt.figure(figsize = (12,16))
plt.subplots_adjust(wspace=.5)
plt.subplot(212)
plt.scatter(port_vol_sim, port_return_sim, c = sharpe_arr, marker = 'o',cmap=
'coolwarm')
plt.scatter(max_vol, max_ret, c = 'green', s= 50)
plt.grid(True)
plt.xlabel('expected volatility')
plt.ylabel('expected return')
plt.colorbar(label = 'Sharpe Ratio')
plt.title('Monte Carlo Simulation Efficient Frontier')
```

Out[10]:
```
Text(0.5, 1.0, 'Monte Carlo Simulation Efficient Frontier')
```



Above is our 'efficient frontier' in which we visually see the monte carlo simulation results and the green dot represents the optimal portfolio choice when following markowitz theory and taking into acount our expected return vs our expected volatility.

In [11]:

```
"""from scipy.optimize import minimize
def get_ret_vol_sr(weights):
    weights = np.array(weights)
    ret = np.sum(stonks_returns.mean() * weights) * 252
    vol = np.sqrt(np.dot(weights.T, np.dot(stonks_returns.cov()*252, weight
s)))
    sr = ret/vol
    return np.array([ret, vol, sr])

def neg_sharpe(weights):
# the number 2 is the sharpe ratio index from the get_ret_vol_sr
    return get_ret_vol_sr(weights)[2] * -1

def check_sum(weights):
    #return 0 if sum of the weights is 1
    return np.sum(weights)-1

cons= ({'type': 'eq', 'fun':check_sum})
bounds= ((0,1),(0,1),(0,1),(0,1))
init_guess = [0.25,0.25,0.25,0.25]

opt_results = minimize(neg_sharpe, init_guess, method= 'SLSQP', bounds = bound
s, constraints= cons)
print(opt_results)

get_ret_vol_sr(opt_results.x)
frontier_y = np.linspace(0, 0.8,200)
def minimize_volatility(weights):
    return get_ret_vol_sr(weights)[1]
"""
```

Out[11]:

```
"from scipy.optimize import minimize\ndef get_ret_vol_sr(weights):\n    weigh
ts = np.array(weights)\n    ret = np.sum(stonks_returns.mean() * weights) * 2
52\n    vol = np.sqrt(np.dot(weights.T, np.dot(stonks_returns.cov()*252, weig
hts)))\n    sr = ret/vol\n    return np.array([ret, vol, sr])\n\ndef neg_shar
pe(weights):\n# the number 2 is the sharpe ratio index from the get_ret_vol_s
r\n    return get_ret_vol_sr(weights)[2] * -1\n\ndef check_sum(weights):\n    
#return 0 if sum of the weights is 1\n    return np.sum(weights)-1\n\ncons=
({'type': 'eq', 'fun':check_sum})\nbounds= ((0,1),(0,1),(0,1),(0,1))\ninit_gu
ess = [0.25,0.25,0.25,0.25]\n\nopt_results = minimize(neg_sharpe, init_guess,
method= 'SLSQP', bounds = bounds, constraints= cons)\nprint(opt_results)\n\ng
et_ret_vol_sr(opt_results.x)\nfrontier_y = np.linspace(0, 0.8,200)\ndef minim
ize_volatility(weights):\n    return get_ret_vol_sr(weights)[1]\n"
```

In [12]:
```
"""frontier_x = []

for possible_return in frontier_y:
    cons = ({'type':'eq', 'fun':check_sum},
            {'type':'eq', 'fun': lambda w: get_ret_vol_sr(w)[0] - possible_ret
urn})

    result = minimize(minimize_volatility,init_guess,method='SLSQP', bounds=bo
unds, constraints=cons)
    frontier_x.append(result['fun'])

plt.figure(figsize=(12,8))
plt.scatter(port_vol_sim, port_return_sim, c=sharpe_arr, cmap='viridis')
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Volatility')
plt.ylabel('Return')
plt.plot(frontier_x,frontier_y, 'r--', linewidth=3)
plt.savefig('cover.png')
plt.show()"""
```

Out[12]:
```
"frontier_x = []\n\nfor possible_return in frontier_y:\n    cons = ({'typ
e':'eq', 'fun':check_sum},\n            {'type':'eq', 'fun': lambda w: get_re
t_vol_sr(w)[0] - possible_return})\n    \n    result = minimize(minimize_vola
tility,init_guess,method='SLSQP', bounds=bounds, constraints=cons)\n    front
ier_x.append(result['fun'])\n    \nplt.figure(figsize=(12,8))\nplt.scatter(po
rt_vol_sim, port_return_sim, c=sharpe_arr, cmap='viridis')\nplt.colorbar(labe
l='Sharpe Ratio')\nplt.xlabel('Volatility')\nplt.ylabel('Return')\nplt.plot(f
rontier_x,frontier_y, 'r--', linewidth=3)\nplt.savefig('cover.png')\nplt.show
()"
```

In [16]:
```
# For Manually Inputing Weights
inp = 'no'
while inp == 'yes':
    inp = str(input("Would you like to continue: 'yes' or 'no'?"))
    weight=[]
    while np.sum(weight) != float(1.0):
        for i in range(len(stonks)):
            x = float(input("Enter Weight of " + stonks[i]))
            weight.append(x)
    print(weight)
    weight= np.array(weight)
    Expected_port_return= np.sum(stonks_returns.mean()*weight)*365
    Expected_port_Std = np.sqrt(weight.T.dot(stonks_returns.cov()*365).dot(wei
ght))
    port_sharpe= Expected_port_return/Expected_port_Std
    print('Annualized Returns: {:.3%}'.format(Expected_port_return))
    print('Annualized Volatility: {:.3%}'.format(Expected_port_Std))
    print('Sharpe Ratio: {:.4}'.format(port_sharpe))
    if inp == 'no':
        break
```

In [21]:
```
#weight=[0.14807922, 0.16466128, 0.07525923, 0.00480915, 0.03407212, 0.013477
4,
 #0.07040439, 0.00621632, 0.03569034, 0.14024049, 0.01212091, 0.00636116, 0.16
319165, 0.09052107, 0.02298392, 0.01191133]
weight= [0.82264552,0.00783881,0.00900001,0.16051566]
weight= np.array(weight)
print(weight)
Expected_port_return= np.sum(stonks_returns.mean()*weight)*365
Expected_port_Std = np.sqrt(weight.T.dot(stonks_returns.cov()*365).dot(weight
))
port_sharpe= Expected_port_return/Expected_port_Std
print('Annualized Returns: {:.3%}'.format(Expected_port_return))
print('Annualized Volatility: {:.3%}'.format(Expected_port_Std))
print('Sharpe Ratio: {:.4}'.format(port_sharpe))
```

```
[0.82264552 0.00783881 0.00900001 0.16051566]
Annualized Returns: 64.410%
Annualized Volatility: 35.721%
Sharpe Ratio: 1.803
```

In [ ]: