2[with risk free]

April 11, 2021

1 Alka Santosh (M19MA003)

2 Sanyam Jain (P20QC001)

3 Sabhilesh (M19MA015)

4 1. Pick any 10 risky assets from the market. Use their 3 months closing price to obtain simple returns.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pandas_datareader as web
!wget https://raw.githubusercontent.com/qoo121314/Portfolio_Optimizer/master/
↪PortfolioOptimizer.py
```

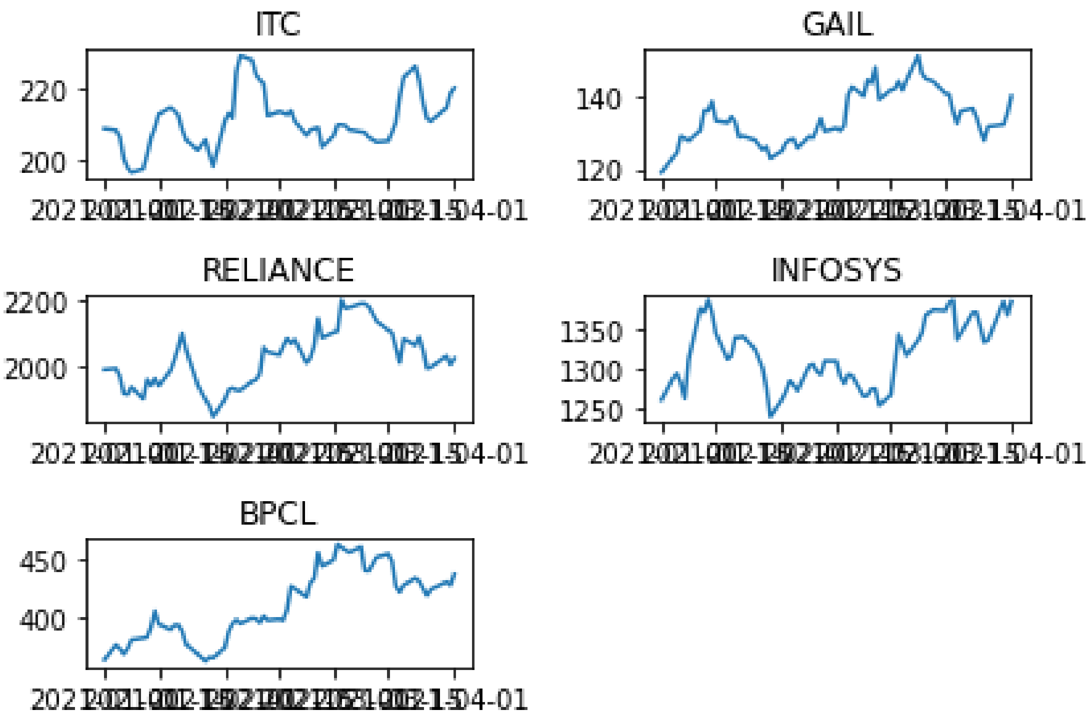
```
--2021-04-10 21:01:21--  https://raw.githubusercontent.com/qoo121314/Portfolio_O
ptimizer/master/PortfolioOptimizer.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.109.133, 185.199.110.133, 185.199.111.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 21688 (21K) [text/plain]
Saving to: 'PortfolioOptimizer.py'

PortfolioOptimizer. 100%[===================>]  21.18K  --.-KB/s    in 0s

2021-04-10 21:01:21 (49.4 MB/s) - 'PortfolioOptimizer.py' saved [21688/21688]
```

[2]:
```
tickers1 = ["ITC.NS", "GAIL.NS","RELIANCE.NS", "INFY.NS", "BPCL.NS"] #
tickers2 =  ["WIPRO.NS", "TCS.NS", "HDFCBANK.NS", "KOTAKBANK.NS", "LT.NS"]
```

[3]:
```
multpl_stocks_1 = web.get_data_yahoo(tickers1,
start = "2021-01-01",
```

```
end = "2021-03-31")
multpl_stocks_2 = web.get_data_yahoo(tickers2,
start = "2021-01-01",
end = "2021-03-31")
```

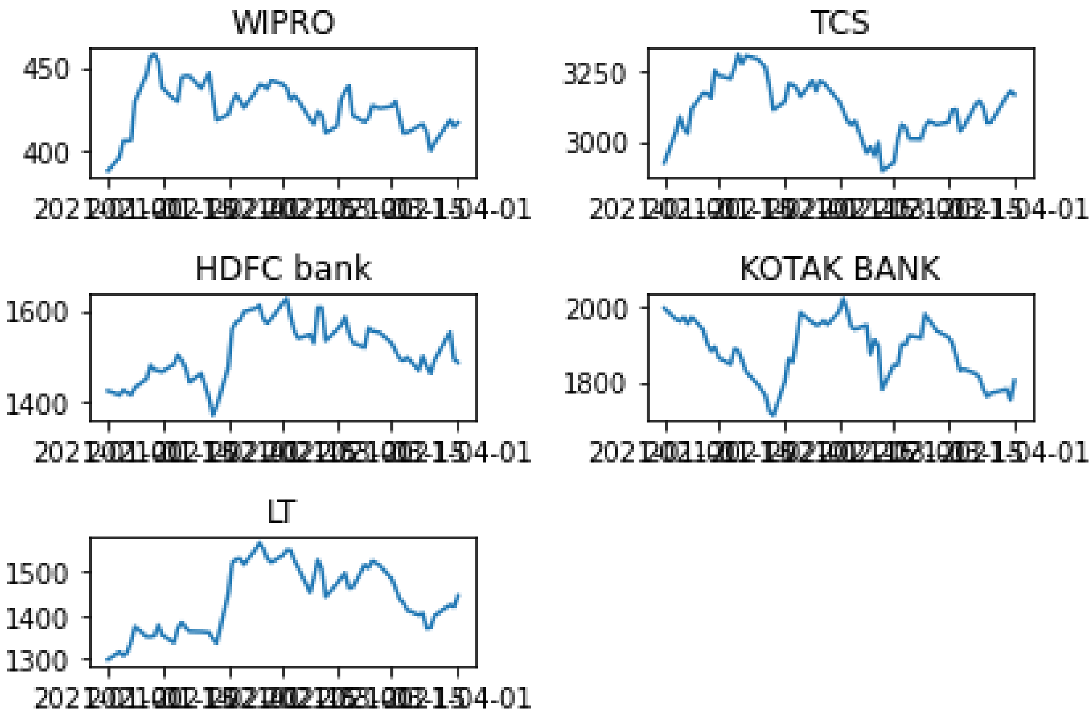
[4]:
```
fig = plt.figure()

ax1 = fig.add_subplot(321)
ax2 = fig.add_subplot(322)
ax3 = fig.add_subplot(323)
ax4 = fig.add_subplot(324)
ax5 = fig.add_subplot(325)


ax1.plot(multpl_stocks_1['Adj Close']['ITC.NS'])
ax1.set_title("ITC")

ax2.plot(multpl_stocks_1['Adj Close']['GAIL.NS'])
ax2.set_title("GAIL")

ax3.plot(multpl_stocks_1['Adj Close']['RELIANCE.NS'])
ax3.set_title("RELIANCE")

ax4.plot(multpl_stocks_1['Adj Close']['INFY.NS'])
ax4.set_title("INFOSYS")

ax5.plot(multpl_stocks_1['Adj Close']['BPCL.NS'])
ax5.set_title("BPCL")

plt.tight_layout()
plt.show()
```



[5]:

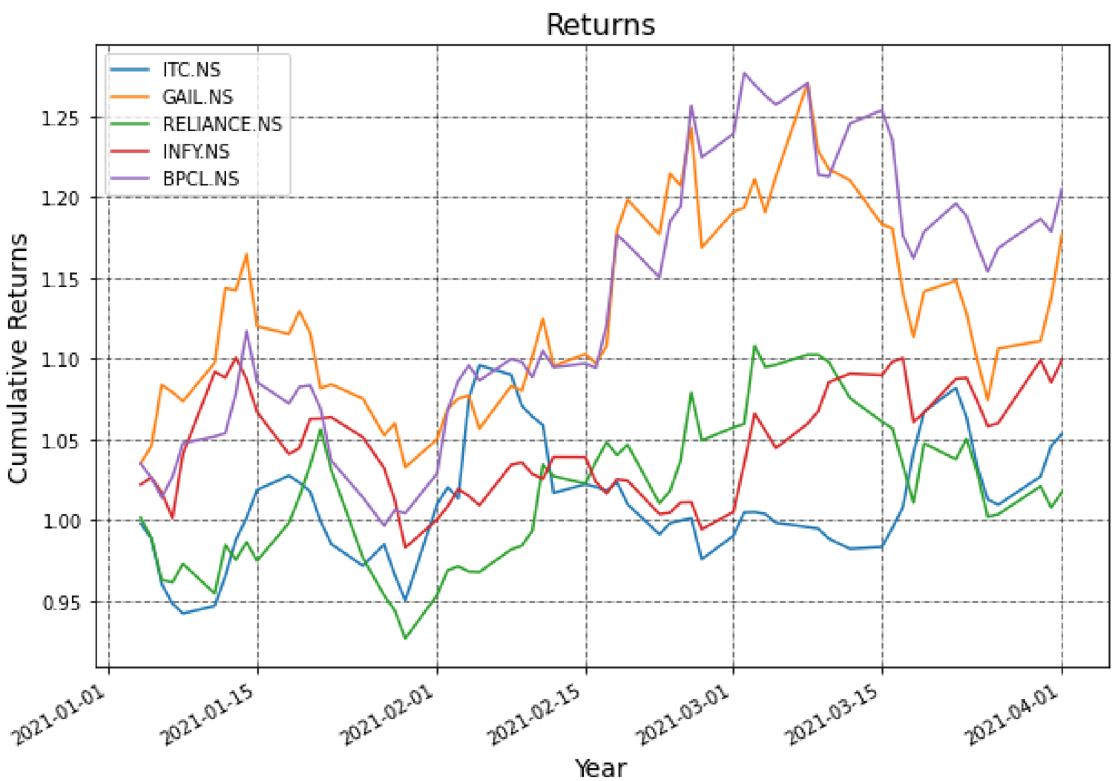
```
fig = plt.figure()
ax6 = fig.add_subplot(321)
ax7 = fig.add_subplot(322)
ax8 = fig.add_subplot(323)
ax9 = fig.add_subplot(324)
ax10 = fig.add_subplot(325)

ax6.plot(multpl_stocks_2['Adj Close']['WIPRO.NS'])
ax6.set_title("WIPRO")

ax7.plot(multpl_stocks_2['Adj Close']['TCS.NS'])
ax7.set_title("TCS")

ax8.plot(multpl_stocks_2['Adj Close']['HDFCBANK.NS'])
ax8.set_title("HDFC bank")

ax9.plot(multpl_stocks_2['Adj Close']['KOTAKBANK.NS'])
ax9.set_title("KOTAK BANK")

ax10.plot(multpl_stocks_2['Adj Close']['LT.NS'])
ax10.set_title("LT")

plt.tight_layout()
```

```
plt.show()
```



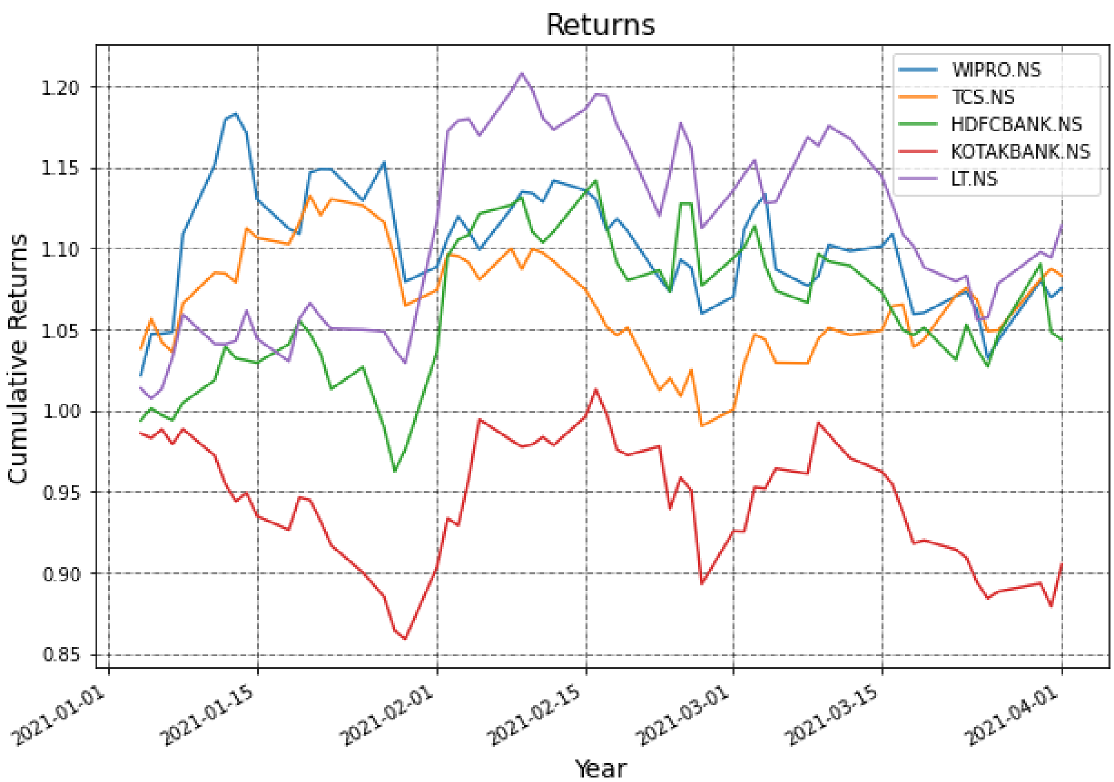
[6]:

```
# Plot all the close prices
((multpl_stocks_1['Adj Close'].pct_change()+1).cumprod()).plot(figsize=(10, 7))

# Show the legend
plt.legend()

# Define the label for the title of the figure
plt.title("Returns", fontsize=16)

# Define the labels for x-axis and y-axis
plt.ylabel('Cumulative Returns', fontsize=14)
plt.xlabel('Year', fontsize=14)

# Plot the grid lines
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



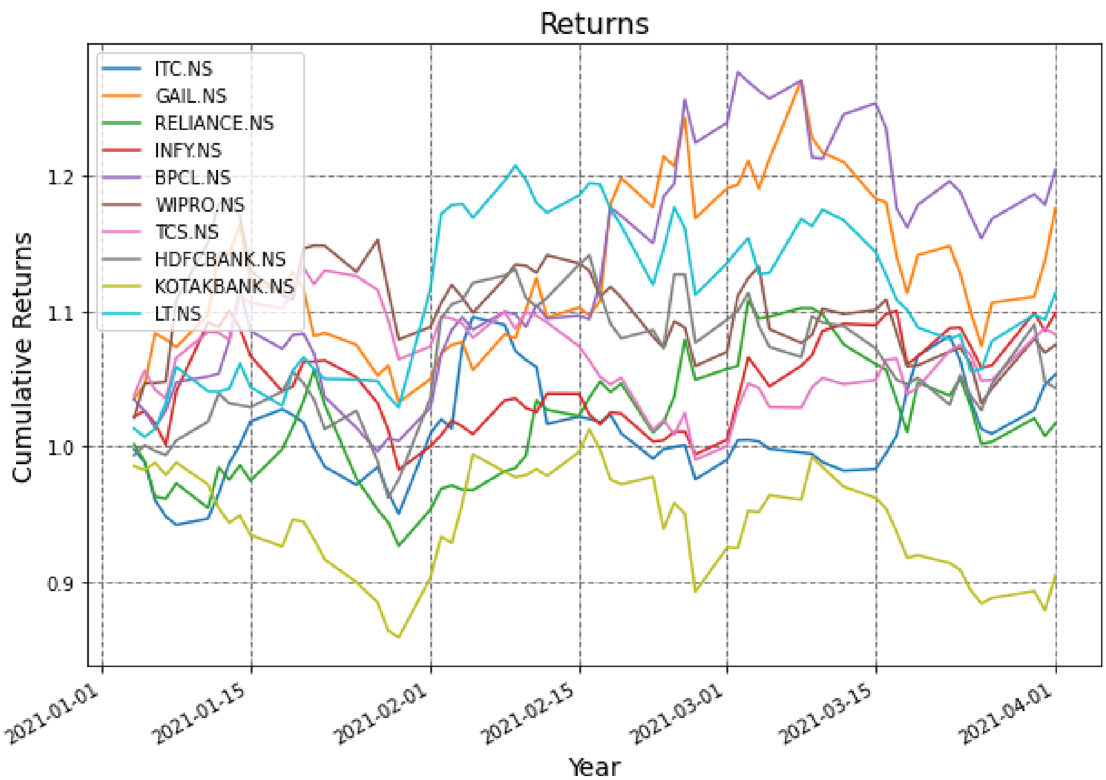
[7]:

```
# Plot all the close prices
((multpl_stocks_2['Adj Close'].pct_change()+1).cumprod()).plot(figsize=(10, 7))

# Show the legend
plt.legend()

# Define the label for the title of the figure
plt.title("Returns", fontsize=16)

# Define the labels for x-axis and y-axis
plt.ylabel('Cumulative Returns', fontsize=14)
plt.xlabel('Year', fontsize=14)

# Plot the grid lines
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



[8]:
```
tickers = ["ITC.NS", "GAIL.NS","RELIANCE.NS", "INFY.NS", "BPCL.NS", "WIPRO.NS",␣
↪"TCS.NS", "HDFCBANK.NS", "KOTAKBANK.NS", "LT.NS"]


multpl_stocks = web.get_data_yahoo(tickers,
start = "2021-01-01",
end = "2021-03-31")




# Plot all the close prices
((multpl_stocks['Adj Close'].pct_change()+1).cumprod()).plot(figsize=(10, 7))

# Show the legend
plt.legend()

# Define the label for the title of the figure
plt.title("Returns", fontsize=16)

# Define the labels for x-axis and y-axis
plt.ylabel('Cumulative Returns', fontsize=14)
```

```
plt.xlabel('Year', fontsize=14)

# Plot the grid lines
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



[8]:

```

```

5 2. Use the mean-variance theory and build the Markowitz efficient frontier.

[9]:

```
stocks = multpl_stocks['Adj Close']
riskfree = [100, 100.15, 100.3, 100.45, 100.6, 100.75, 100.9, 101.05, 101.2,␣
 ↪101.35, 101.5, 101.65, 101.8, 101.95, 102.1, 102.25, 102.4, 102.55, 102.7,␣
 ↪102.85, 103, 103.15, 103.3, 103.45, 103.6, 103.75, 103.9, 104.05, 104.2, 104.
 ↪35, 104.5, 104.65, 104.8, 104.95, 105.1, 105.25, 105.4, 105.55, 105.7, 105.
 ↪85, 106, 106.15, 106.3, 106.45, 106.6, 106.75, 106.9, 107.05, 107.2, 107.35,␣
 ↪107.5, 107.65, 107.8, 107.95, 108.1, 108.25, 108.4, 108.55, 108.7, 108.85,␣
 ↪109, 109.15]
stocks['risk_free'] = riskfree
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```

[10]:
```
stocks.head()
```

[10]:
```
Symbols         ITC.NS     GAIL.NS  …        LT.NS  risk_free
Date                                …
2021-01-01  208.898636  119.134895  …  1297.000000     100.00
2021-01-04  208.459045  123.326057  …  1314.599976     100.15
2021-01-05  206.554199  124.578583  …  1306.300049     100.30
2021-01-06  200.644272  129.106964  …  1314.000000     100.45
2021-01-07  198.104477  128.577042  …  1338.949951     100.60

[5 rows x 11 columns]
```

[11]:
```
# Converting everything to logarithmic returns is simple. Think of it as the
↪log of an arithmetic daily return (which is obtained by dividing the price
↪at day n, by the price at day n-1).
```

[12]:
```
log_returns = np.log(stocks/stocks.shift(1))
# log_returns.dropna(inplace=True)
log_returns.head()
```

[12]:
```
Symbols       ITC.NS   GAIL.NS  RELIANCE.NS  …  KOTAKBANK.NS     LT.NS
risk_free
Date                                         …
2021-01-01       NaN       NaN          NaN  …           NaN       NaN
NaN
2021-01-04 -0.002107  0.034575     0.001684  …     -0.014396  0.013479
0.001499
2021-01-05 -0.009180  0.010105    -0.012510  …     -0.002955 -0.006334
0.001497
2021-01-06 -0.029029  0.035705    -0.026726  …      0.005420  0.005877
0.001494
2021-01-07 -0.012739 -0.004113    -0.001621  …     -0.009177  0.018810
0.001492

[5 rows x 11 columns]
```

[13]:

```
# I'm going to use 6000 portfolios, but feel free to use less if your computer␣
↪is too slow. The random seed at the top of the code is making sure I get the␣
↪same random numbers every time for reproducibility.
```

[15]:
```
np.random.seed(30)
num_ports = 6000
all_weights = np.zeros((num_ports, len(stocks.columns)))
ret_arr = np.zeros(num_ports)
vol_arr = np.zeros(num_ports)
sharpe_arr = np.zeros(num_ports)

for x in range(num_ports):
    # Weights
    weights = np.array(np.random.random(11))
    weights = weights/np.sum(weights) # Wi*

    # Save weights
    all_weights[x,:] = weights

    # Expected return
    ret_arr[x] = np.sum( (log_returns.mean() * weights * 252))

    # Expected volatility
    vol_arr[x] = np.sqrt(np.dot(weights.T, np.dot(log_returns.cov()*252,␣
↪weights)))

    # Sharpe Ratio
    sharpe_arr[x] = ret_arr[x]/vol_arr[x]
```

[16]:
```
print("Max Sharpe")
print(sharpe_arr.max())
print("\n\n")
print("location in array")
print(sharpe_arr.argmax())
```

```
Max Sharpe
2.5718716857320993



location in array
3788
```

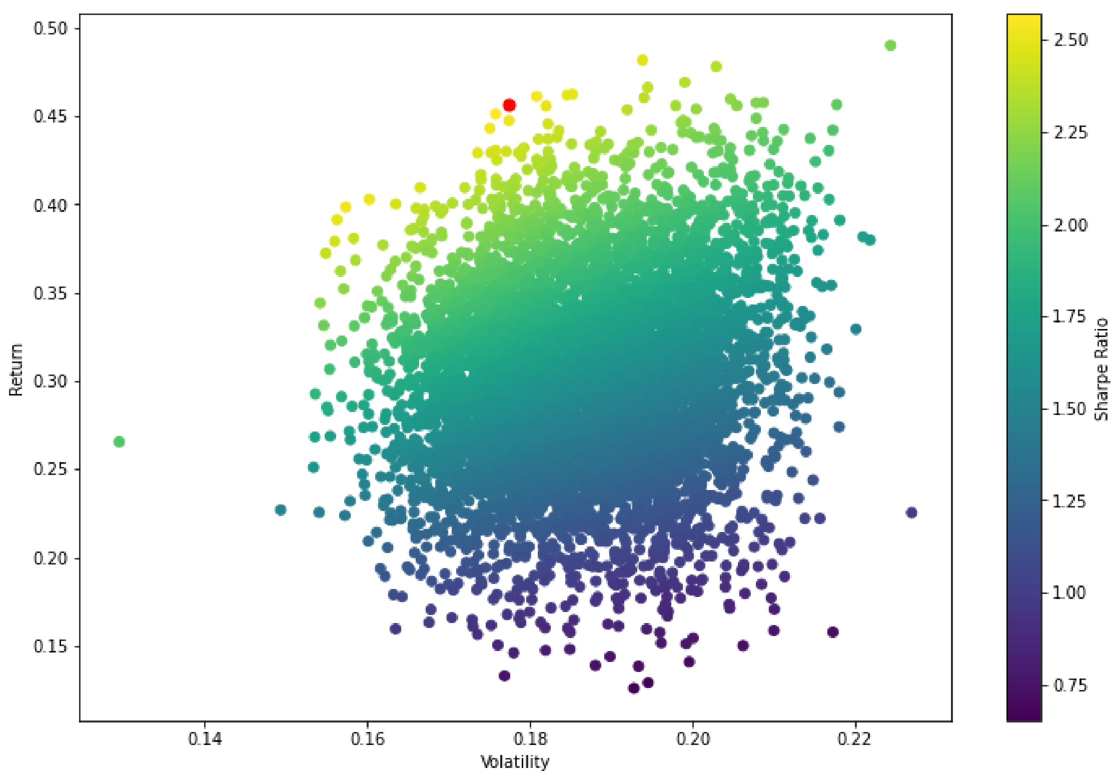
[17]:
```
# Print the allocations in the max result
print(all_weights[3788,:])
max_returns = ret_arr[sharpe_arr.argmax()]
max_vol = vol_arr[sharpe_arr.argmax()]
```

```
[0.12530348 0.16239756 0.00593159 0.1072165  0.14898168 0.00405427
 0.00566325 0.02825753 0.01251961 0.1913007  0.20837384]
```

[18]:
```
for i in range(10):
  print("The allocation for: "+str(tickers[i])+" is: "+ str(all_weights[3788,:
↪][i]))
```

```
The allocation for: ITC.NS is: 0.12530347887086948
The allocation for: GAIL.NS is: 0.16239755747954962
The allocation for: RELIANCE.NS is: 0.0059315892039434595
The allocation for: INFY.NS is: 0.10721649690309175
The allocation for: BPCL.NS is: 0.14898167714016328
The allocation for: WIPRO.NS is: 0.004054269345481703
The allocation for: TCS.NS is: 0.0056632489073559856
The allocation for: HDFCBANK.NS is: 0.028257531213158056
The allocation for: KOTAKBANK.NS is: 0.01251960906160665
The allocation for: LT.NS is: 0.19130070001718671
```

[19]:
```
plt.figure(figsize=(12,8))
plt.scatter(vol_arr, ret_arr, c=sharpe_arr, cmap='viridis')
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Volatility')
plt.ylabel('Return')
plt.scatter(max_vol, max_returns,c='red', s=50) # red dot
plt.show()
```



[20]:

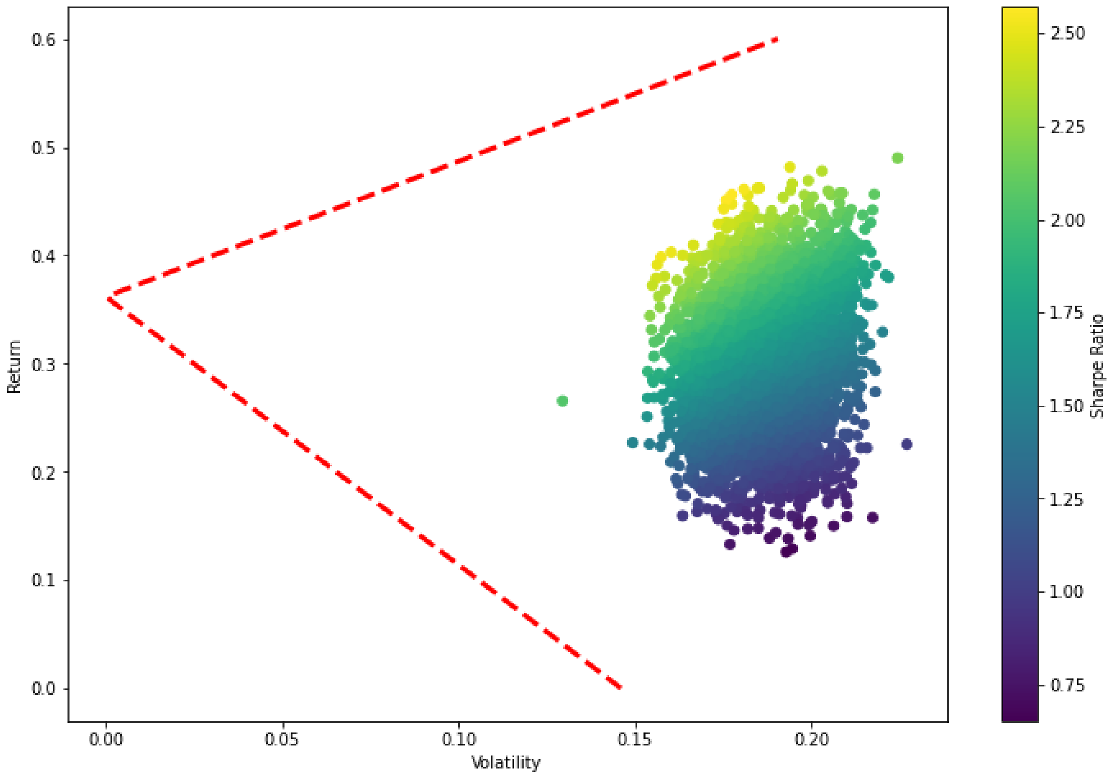
```
def get_ret_vol_sr(weights):
    weights = np.array(weights)
    ret = np.sum(log_returns.mean() * weights) * 252
    vol = np.sqrt(np.dot(weights.T, np.dot(log_returns.cov()*252, weights)))
    sr = ret/vol
    return np.array([ret, vol, sr])

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

def minimize_volatility(weights):
  return get_ret_vol_sr(weights)[1]
```

[22]:

```
frontier_x = []
frontier_y = np.linspace(0,0.6,200)
from scipy.optimize import minimize
bounds = ((0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1))
init_guess = [0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50,0.50]

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

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

[23]:

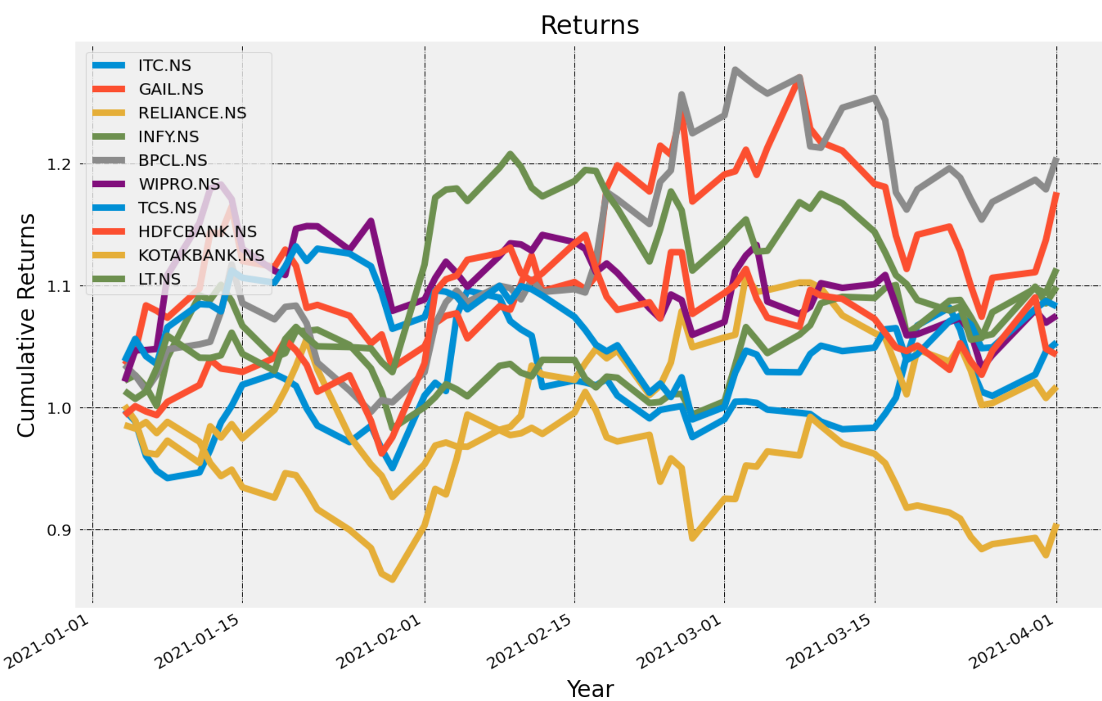
```
plt.figure(figsize=(12,8))
plt.scatter(vol_arr, ret_arr, c=sharpe_arr, cmap='viridis')
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Volatility')
plt.ylabel('Return')
plt.plot(frontier_x,frontier_y, 'r--', linewidth=3)
plt.savefig('cover.png')
plt.show()
```



6 Different Methodology

[24]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import scipy.optimize as sco

plt.style.use('fivethirtyeight')
np.random.seed(777)

%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pandas_datareader as web
tickers = ["ITC.NS", "GAIL.NS","RELIANCE.NS", "INFY.NS", "BPCL.NS", "WIPRO.NS",␣
 ↪"TCS.NS", "HDFCBANK.NS", "KOTAKBANK.NS", "LT.NS"]
```

```
multpl_stocks = web.get_data_yahoo(tickers,
start = "2021-01-01",
end = "2021-03-31")




# Plot all the close prices
((multpl_stocks['Adj Close'].pct_change()+1).cumprod()).plot(figsize=(10, 7))

# Show the legend
plt.legend()

# Define the label for the title of the figure
plt.title("Returns", fontsize=16)

# Define the labels for x-axis and y-axis
plt.ylabel('Cumulative Returns', fontsize=14)
plt.xlabel('Year', fontsize=14)

# Plot the grid lines
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()


stocks = multpl_stocks['Adj Close']
```



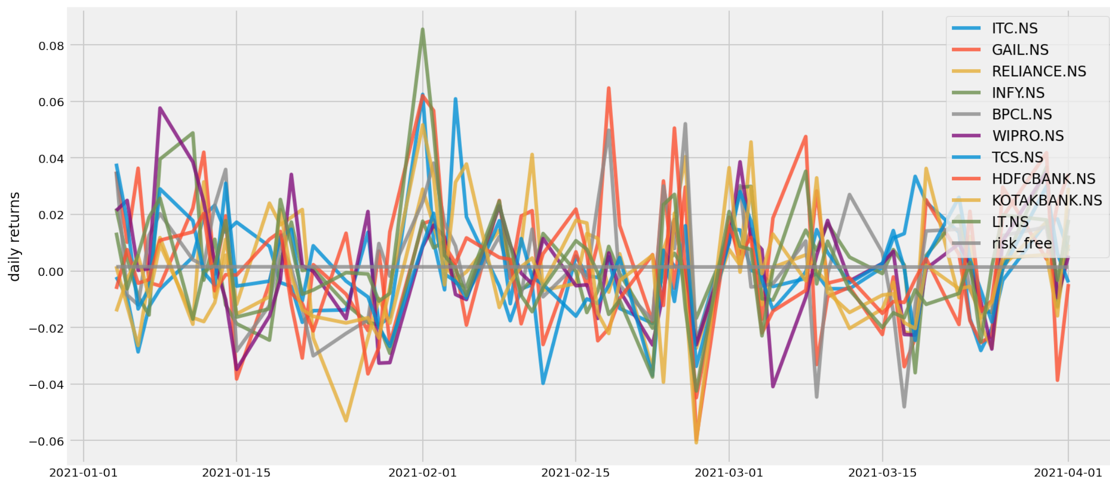
[25]:
```
riskfree = [100, 100.15, 100.3, 100.45, 100.6, 100.75, 100.9, 101.05, 101.2,␣
↪101.35, 101.5, 101.65, 101.8, 101.95, 102.1, 102.25, 102.4, 102.55, 102.7,␣
↪102.85, 103, 103.15, 103.3, 103.45, 103.6, 103.75, 103.9, 104.05, 104.2, 104.
↪35, 104.5, 104.65, 104.8, 104.95, 105.1, 105.25, 105.4, 105.55, 105.7, 105.
↪85, 106, 106.15, 106.3, 106.45, 106.6, 106.75, 106.9, 107.05, 107.2, 107.35,␣
↪107.5, 107.65, 107.8, 107.95, 108.1, 108.25, 108.4, 108.55, 108.7, 108.85,␣
↪109, 109.15]
stocks['risk_free'] = riskfree
stocks.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```

[25]:
```
Symbols          ITC.NS     GAIL.NS  …        LT.NS  risk_free
Date                                 …
2021-01-01  208.898636  119.134895  …  1297.000000     100.00
2021-01-04  208.459045  123.326057  …  1314.599976     100.15
2021-01-05  206.554199  124.578583  …  1306.300049     100.30
2021-01-06  200.644272  129.106964  …  1314.000000     100.45
2021-01-07  198.104477  128.577042  …  1338.949951     100.60

[5 rows x 11 columns]
```

[26]:
```
returns = stocks.pct_change()

plt.figure(figsize=(14, 7))
for c in returns.columns.values:
    plt.plot(returns.index, returns[c], lw=3, alpha=0.8,label=c)
plt.legend(loc='upper right', fontsize=12)
plt.ylabel('daily returns')
```

[26]:
```
Text(0, 0.5, 'daily returns')
```



[27]:
```
def portfolio_annualised_performance(weights, mean_returns, cov_matrix):
    returns = np.sum(mean_returns*weights ) *252
    std = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights))) * np.sqrt(252)
    return std, returns
def random_portfolios(num_portfolios, mean_returns, cov_matrix, risk_free_rate):
    results = np.zeros((11,num_portfolios))
    weights_record = []
    for i in range(num_portfolios):
        weights = np.random.random(11)
        weights /= np.sum(weights)
        weights_record.append(weights)
        portfolio_std_dev, portfolio_return = ␣
↪portfolio_annualised_performance(weights, mean_returns, cov_matrix)
        results[0,i] = portfolio_std_dev
        results[1,i] = portfolio_return
        results[2,i] = (portfolio_return - risk_free_rate) / portfolio_std_dev
    return results, weights_record
```

[28]:
```
returns = stocks.pct_change()
mean_returns = returns.mean()
cov_matrix = returns.cov()
num_portfolios = 25000
risk_free_rate = 0.0178
```

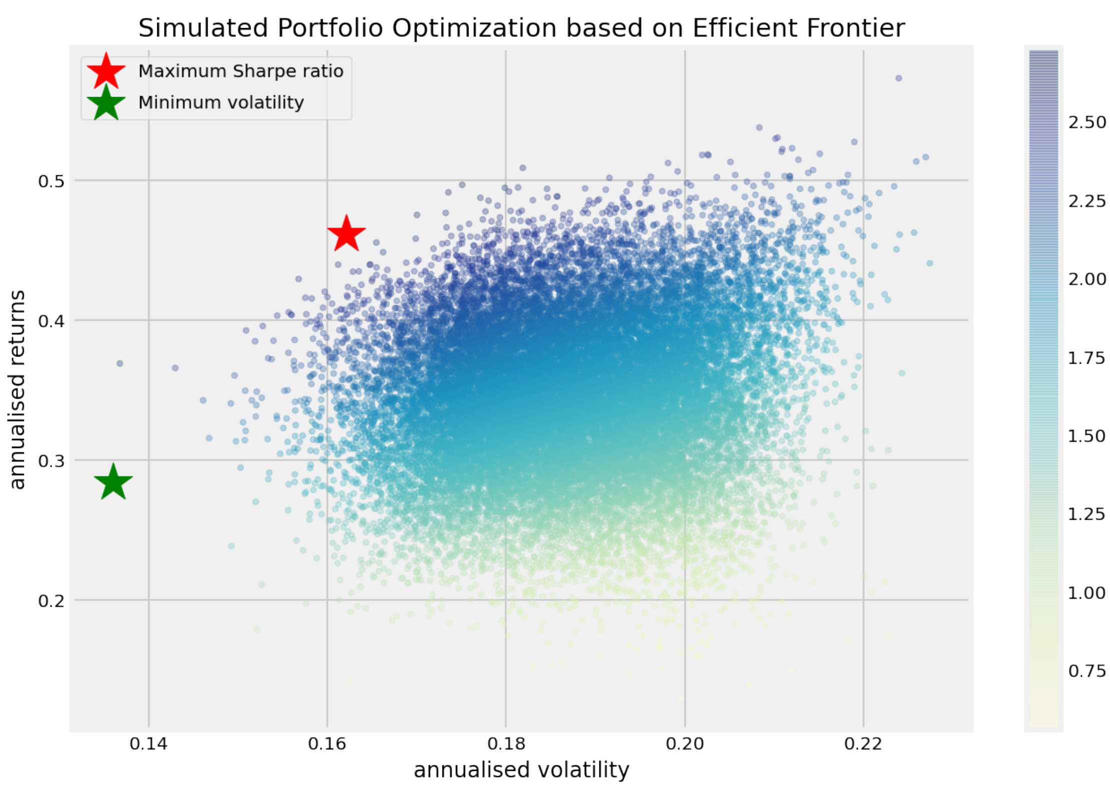
[29]:
```
def display_simulated_ef_with_random(mean_returns, cov_matrix, num_portfolios,␣
↪risk_free_rate):
    results, weights = random_portfolios(num_portfolios,mean_returns,␣
↪cov_matrix, risk_free_rate)

    max_sharpe_idx = np.argmax(results[2])
```

```
    sdp, rp = results[0,max_sharpe_idx], results[1,max_sharpe_idx]
    max_sharpe_allocation = pd.DataFrame(weights[max_sharpe_idx],index=stocks.
↪columns,columns=['allocation'])
    max_sharpe_allocation.allocation = [round(i*100,2)for i in␣
↪max_sharpe_allocation.allocation]
    max_sharpe_allocation = max_sharpe_allocation.T

    min_vol_idx = np.argmin(results[0])
    sdp_min, rp_min = results[0,min_vol_idx], results[1,min_vol_idx]
    min_vol_allocation = pd.DataFrame(weights[min_vol_idx],index=stocks.
↪columns,columns=['allocation'])
    min_vol_allocation.allocation = [round(i*100,2)for i in min_vol_allocation.
↪allocation]
    min_vol_allocation = min_vol_allocation.T

    print ("-"*80)
    print ("Maximum Sharpe Ratio Portfolio Allocation\n")
    print ("Annualised Return:", round(rp,2))
    print ("Annualised Volatility:", round(sdp,2))
    print ("\n")
    print (max_sharpe_allocation)
    print ("-"*80)
    print ("Minimum Volatility Portfolio Allocation\n")
    print ("Annualised Return:", round(rp_min,2))
    print ("Annualised Volatility:", round(sdp_min,2))
    print ("\n")
    print (min_vol_allocation)

    plt.figure(figsize=(10, 7))
    plt.scatter(results[0,:],results[1,:],c=results[2,:],cmap='YlGnBu',␣
↪marker='o', s=10, alpha=0.3)
    plt.colorbar()
    plt.scatter(sdp,rp,marker='*',color='r',s=500, label='Maximum Sharpe ratio')
    plt.scatter(sdp_min,rp_min,marker='*',color='g',s=500, label='Minimum␣
↪volatility')
    plt.title('Simulated Portfolio Optimization based on Efficient Frontier')
    plt.xlabel('annualised volatility')
    plt.ylabel('annualised returns')
    plt.legend(labelspacing=0.8)
```

[30]:
```
display_simulated_ef_with_random(mean_returns, cov_matrix, num_portfolios,␣
↪risk_free_rate)
```

```
--------------------------------------------------------------------------------
Maximum Sharpe Ratio Portfolio Allocation

Annualised Return: 0.46
```

```
Annualised Volatility: 0.16


Symbols      ITC.NS  GAIL.NS  RELIANCE.NS  …  KOTAKBANK.NS  LT.NS  risk_free
allocation      2.3    10.41         0.98  …          3.01   9.77      24.15

[1 rows x 11 columns]
--------------------------------------------------------------------------------
Minimum Volatility Portfolio Allocation

Annualised Return: 0.28
Annualised Volatility: 0.14


Symbols      ITC.NS  GAIL.NS  RELIANCE.NS  …  KOTAKBANK.NS  LT.NS  risk_free
allocation     1.38     1.36         7.05  …           9.6   9.95      34.43

[1 rows x 11 columns]
```



[31]:
```python
table = stocks
def neg_sharpe_ratio(weights, mean_returns, cov_matrix, risk_free_rate):
```

```
    p_var, p_ret = portfolio_annualised_performance(weights, mean_returns,␣
↪cov_matrix)
    return -(p_ret - risk_free_rate) / p_var

def max_sharpe_ratio(mean_returns, cov_matrix, risk_free_rate):
    num_assets = len(mean_returns)
    args = (mean_returns, cov_matrix, risk_free_rate)
    constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
    bound = (0.0,1.0)
    bounds = tuple(bound for asset in range(num_assets))
    result = sco.minimize(neg_sharpe_ratio, num_assets*[1./num_assets,],␣
↪args=args,
                        method='SLSQP', bounds=bounds, constraints=constraints)
    return result
```

[32]:
```
def portfolio_volatility(weights, mean_returns, cov_matrix):
    return portfolio_annualised_performance(weights, mean_returns,␣
↪cov_matrix)[0]

def min_variance(mean_returns, cov_matrix):
    num_assets = len(mean_returns)
    args = (mean_returns, cov_matrix)
    constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
    bound = (0.0,1.0)
    bounds = tuple(bound for asset in range(num_assets))

    result = sco.minimize(portfolio_volatility, num_assets*[1./num_assets,],␣
↪args=args,
                        method='SLSQP', bounds=bounds, constraints=constraints)

    return result
```

[33]:
```
def efficient_return(mean_returns, cov_matrix, target):
    num_assets = len(mean_returns)
    args = (mean_returns, cov_matrix)

    def portfolio_return(weights):
        return portfolio_annualised_performance(weights, mean_returns,␣
↪cov_matrix)[1]

    constraints = ({'type': 'eq', 'fun': lambda x: portfolio_return(x) -␣
↪target},
                   {'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
    bounds = tuple((0,1) for asset in range(num_assets))
    result = sco.minimize(portfolio_volatility, num_assets*[1./num_assets,],␣
↪args=args, method='SLSQP', bounds=bounds, constraints=constraints)
```

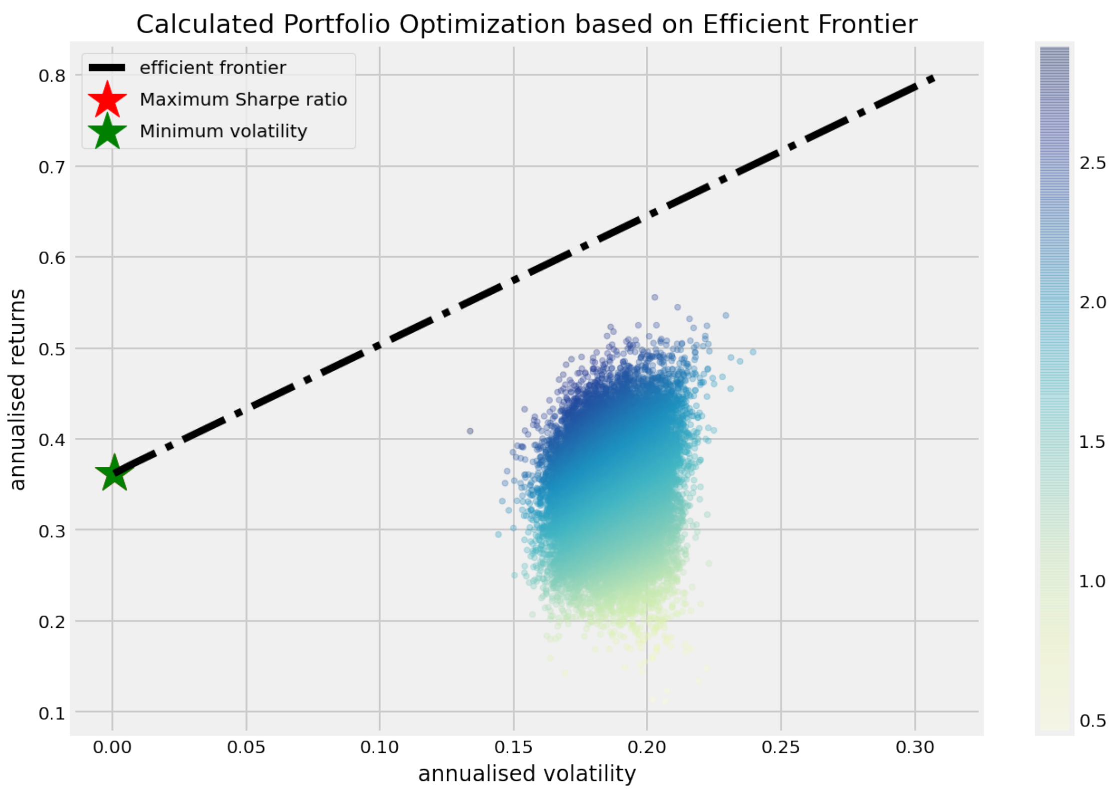
```
    return result


def efficient_frontier(mean_returns, cov_matrix, returns_range):
    efficients = []
    for ret in returns_range:
        efficients.append(efficient_return(mean_returns, cov_matrix, ret))
    return efficients
```

[34]:
```
def display_calculated_ef_with_random(mean_returns, cov_matrix, num_portfolios,␣
↪risk_free_rate):
    results, _ = random_portfolios(num_portfolios,mean_returns, cov_matrix,␣
↪risk_free_rate)

    max_sharpe = max_sharpe_ratio(mean_returns, cov_matrix, risk_free_rate)
    sdp, rp = portfolio_annualised_performance(max_sharpe['x'], mean_returns,␣
↪cov_matrix)
    max_sharpe_allocation = pd.DataFrame(max_sharpe.x,index=table.
↪columns,columns=['allocation'])
    max_sharpe_allocation.allocation = [round(i*100,2)for i in␣
↪max_sharpe_allocation.allocation]
    max_sharpe_allocation = max_sharpe_allocation.T
    max_sharpe_allocation

    min_vol = min_variance(mean_returns, cov_matrix)
    sdp_min, rp_min = portfolio_annualised_performance(min_vol['x'],␣
↪mean_returns, cov_matrix)
    min_vol_allocation = pd.DataFrame(min_vol.x,index=table.
↪columns,columns=['allocation'])
    min_vol_allocation.allocation = [round(i*100,2)for i in min_vol_allocation.
↪allocation]
    min_vol_allocation = min_vol_allocation.T

    print ("-"*80)
    print ("Maximum Sharpe Ratio Portfolio Allocation\n")
    print ("Annualised Return:", round(rp,2))
    print ("Annualised Volatility:", round(sdp,2))
    print ("\n")
    print (max_sharpe_allocation)
    print ("-"*80)
    print ("Minimum Volatility Portfolio Allocation\n")
    print ("Annualised Return:", round(rp_min,2))
    print ("Annualised Volatility:", round(sdp_min,2))
    print ("\n")
    print (min_vol_allocation)
```

```
    plt.figure(figsize=(10, 7))
    plt.scatter(results[0,:],results[1,:],c=results[2,:],cmap='YlGnBu',␣
↪marker='o', s=10, alpha=0.3)
    plt.colorbar()
    plt.scatter(sdp,rp,marker='*',color='r',s=500, label='Maximum Sharpe ratio')
    plt.scatter(sdp_min,rp_min,marker='*',color='g',s=500, label='Minimum␣
↪volatility')

    target = np.linspace(rp_min, 0.8, 50)
    efficient_portfolios = efficient_frontier(mean_returns, cov_matrix, target)
    plt.plot([p['fun'] for p in efficient_portfolios], target, linestyle='-.',␣
↪color='black', label='efficient frontier')
    plt.title('Calculated Portfolio Optimization based on Efficient Frontier')
    plt.xlabel('annualised volatility')
    plt.ylabel('annualised returns')
    plt.legend(labelspacing=0.8)
```

[35]:
```
display_calculated_ef_with_random(mean_returns, cov_matrix, num_portfolios,␣
↪risk_free_rate)
```

```
--------------------------------------------------------------------------------
Maximum Sharpe Ratio Portfolio Allocation

Annualised Return: 0.36
Annualised Volatility: 0.0


Symbols     ITC.NS  GAIL.NS  RELIANCE.NS  …  KOTAKBANK.NS  LT.NS  risk_free
allocation    0.01      0.0          0.0  …           0.0    0.0      99.99

[1 rows x 11 columns]
--------------------------------------------------------------------------------
Minimum Volatility Portfolio Allocation

Annualised Return: 0.36
Annualised Volatility: 0.0


Symbols     ITC.NS  GAIL.NS  RELIANCE.NS  …  KOTAKBANK.NS  LT.NS  risk_free
allocation    0.01      0.0          0.0  …           0.0    0.0      99.99

[1 rows x 11 columns]
```

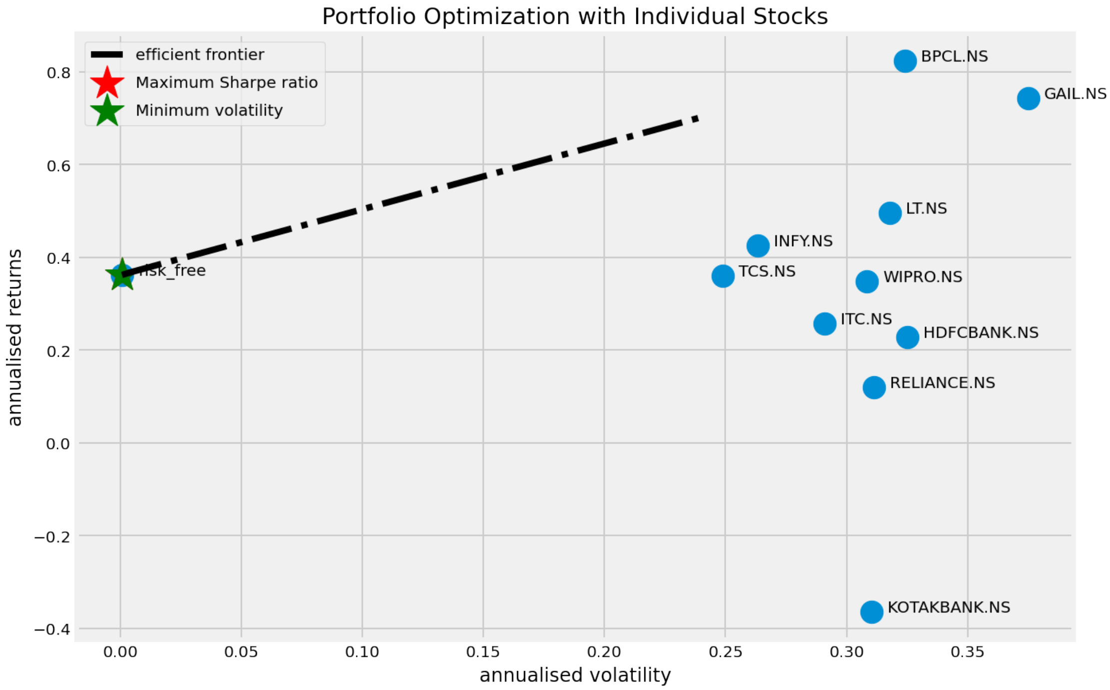


[36]:
```
def display_ef_with_selected(mean_returns, cov_matrix, risk_free_rate):
    max_sharpe = max_sharpe_ratio(mean_returns, cov_matrix, risk_free_rate)
    sdp, rp = portfolio_annualised_performance(max_sharpe['x'], mean_returns,␣
↪cov_matrix)
    max_sharpe_allocation = pd.DataFrame(max_sharpe.x,index=table.
↪columns,columns=['allocation'])
    max_sharpe_allocation.allocation = [round(i*100,2)for i in␣
↪max_sharpe_allocation.allocation]
    max_sharpe_allocation = max_sharpe_allocation.T
    max_sharpe_allocation

    min_vol = min_variance(mean_returns, cov_matrix)
    sdp_min, rp_min = portfolio_annualised_performance(min_vol['x'],␣
↪mean_returns, cov_matrix)
    min_vol_allocation = pd.DataFrame(min_vol.x,index=table.
↪columns,columns=['allocation'])
    min_vol_allocation.allocation = [round(i*100,2)for i in min_vol_allocation.
↪allocation]
    min_vol_allocation = min_vol_allocation.T

    an_vol = np.std(returns) * np.sqrt(252)
```

```
    an_rt = mean_returns * 252

    print ("-"*80)
    print ("Maximum Sharpe Ratio Portfolio Allocation\n")
    print ("Annualised Return:", round(rp,2))
    print ("Annualised Volatility:", round(sdp,2))
    print ("\n")
    print (max_sharpe_allocation)
    print ("-"*80)
    print ("Minimum Volatility Portfolio Allocation\n")
    print ("Annualised Return:", round(rp_min,2))
    print ("Annualised Volatility:", round(sdp_min,2))
    print ("\n")
    print (min_vol_allocation)
    print ("-"*80)
    print ("Individual Stock Returns and Volatility\n")
    for i, txt in enumerate(table.columns):
      print (txt,":","annuaised return",round(an_rt[i],2),", annualised␣
↪volatility:",round(an_vol[i],2))
    print ("-"*80)

    fig, ax = plt.subplots(figsize=(10, 7))
    ax.scatter(an_vol,an_rt,marker='o',s=200)

    for i, txt in enumerate(table.columns):
        ax.annotate(txt, (an_vol[i],an_rt[i]), xytext=(10,0),␣
↪textcoords='offset points')
    ax.scatter(sdp,rp,marker='*',color='r',s=500, label='Maximum Sharpe ratio')
    ax.scatter(sdp_min,rp_min,marker='*',color='g',s=500, label='Minimum␣
↪volatility')

    target = np.linspace(rp_min, 0.7, 50)
    efficient_portfolios = efficient_frontier(mean_returns, cov_matrix, target)
    ax.plot([p['fun'] for p in efficient_portfolios], target, linestyle='-.',␣
↪color='black', label='efficient frontier')
    ax.set_title('Portfolio Optimization with Individual Stocks')
    ax.set_xlabel('annualised volatility')
    ax.set_ylabel('annualised returns')
    ax.legend(labelspacing=0.8)
display_ef_with_selected(mean_returns, cov_matrix, risk_free_rate)
```

```
--------------------------------------------------------------------------------
Maximum Sharpe Ratio Portfolio Allocation

Annualised Return: 0.36
Annualised Volatility: 0.0
```

```
Symbols     ITC.NS  GAIL.NS  RELIANCE.NS  …  KOTAKBANK.NS  LT.NS  risk_free
allocation    0.01      0.0          0.0  …           0.0    0.0      99.99

[1 rows x 11 columns]
--------------------------------------------------------------------------------
Minimum Volatility Portfolio Allocation

Annualised Return: 0.36
Annualised Volatility: 0.0


Symbols     ITC.NS  GAIL.NS  RELIANCE.NS  …  KOTAKBANK.NS  LT.NS  risk_free
allocation    0.01      0.0          0.0  …           0.0    0.0      99.99

[1 rows x 11 columns]
--------------------------------------------------------------------------------
Individual Stock Returns and Volatility

ITC.NS : annuaised return 0.26 , annualised volatility: 0.29
GAIL.NS : annuaised return 0.74 , annualised volatility: 0.38
RELIANCE.NS : annuaised return 0.12 , annualised volatility: 0.31
INFY.NS : annuaised return 0.42 , annualised volatility: 0.26
BPCL.NS : annuaised return 0.82 , annualised volatility: 0.32
WIPRO.NS : annuaised return 0.35 , annualised volatility: 0.31
TCS.NS : annuaised return 0.36 , annualised volatility: 0.25
HDFCBANK.NS : annuaised return 0.23 , annualised volatility: 0.33
KOTAKBANK.NS : annuaised return -0.36 , annualised volatility: 0.31
LT.NS : annuaised return 0.5 , annualised volatility: 0.32
risk_free : annuaised return 0.36 , annualised volatility: 0.0
--------------------------------------------------------------------------------
```



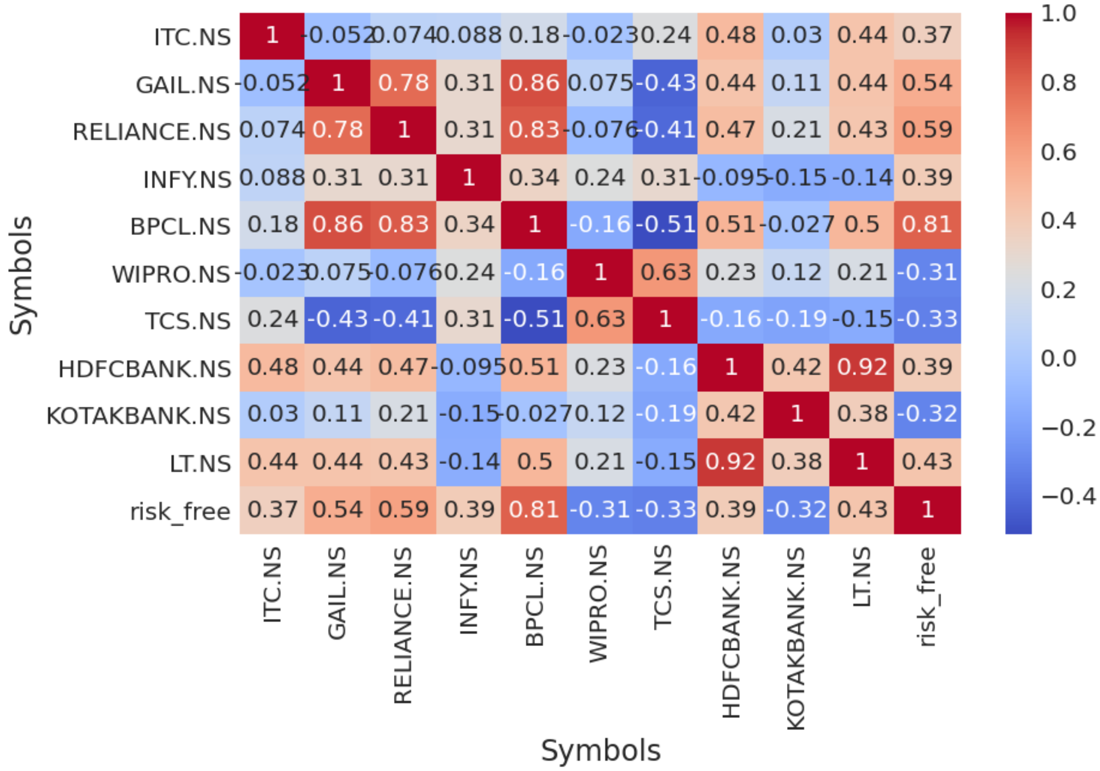
As you can see from the above plot, the stock with the least risk is TCS at around 0.38. But with portfolio optimisation, we can achieve even lower risk at 0.36, and still with a higher return than Google. And if we are willing to take slightly more risk at around the similar level of risk of TCS, we can achieve a much higher return of 0.30 with portfolio optimization.

reference: https://nbviewer.jupyter.org/github/tthustla/efficient_frontier/blob/master/Efficient%20_Frontier_in

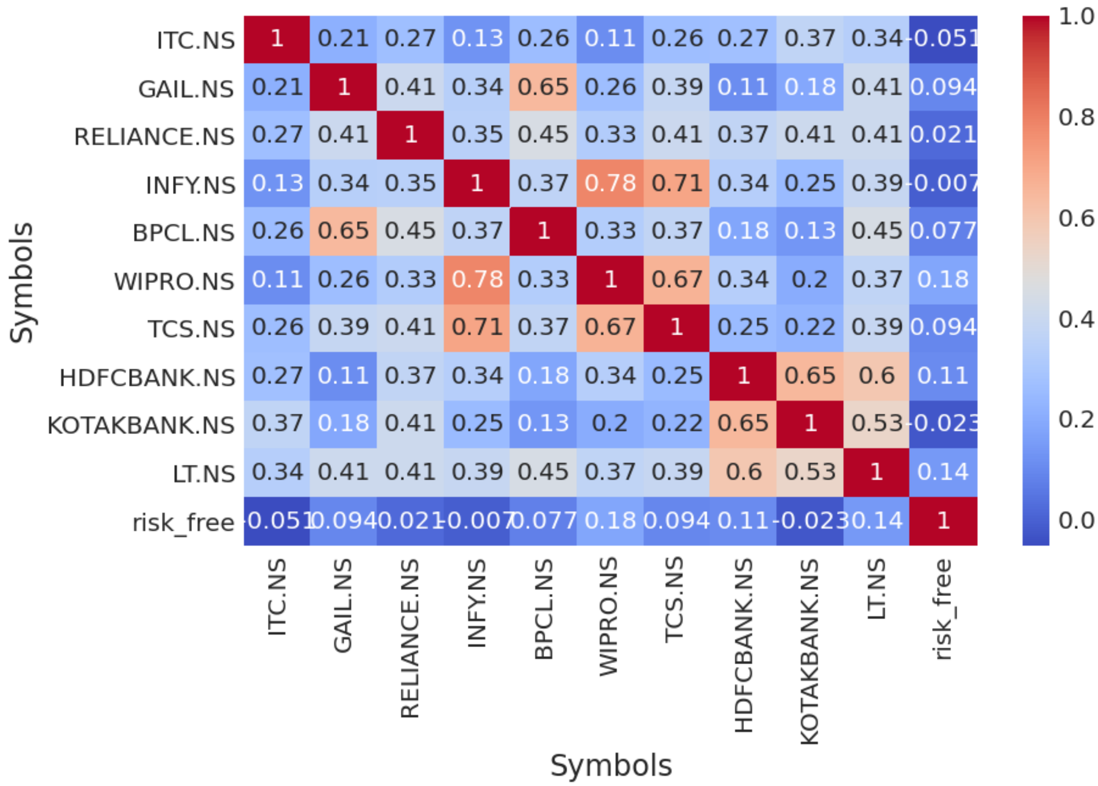
[37]:
```
from PortfolioOptimizer import *
```

[38]:
```
port = Portfolio(table)
```

[39]:
```
port.price_corr_map()
```



[40]:
```
port.return_corr_map()
```



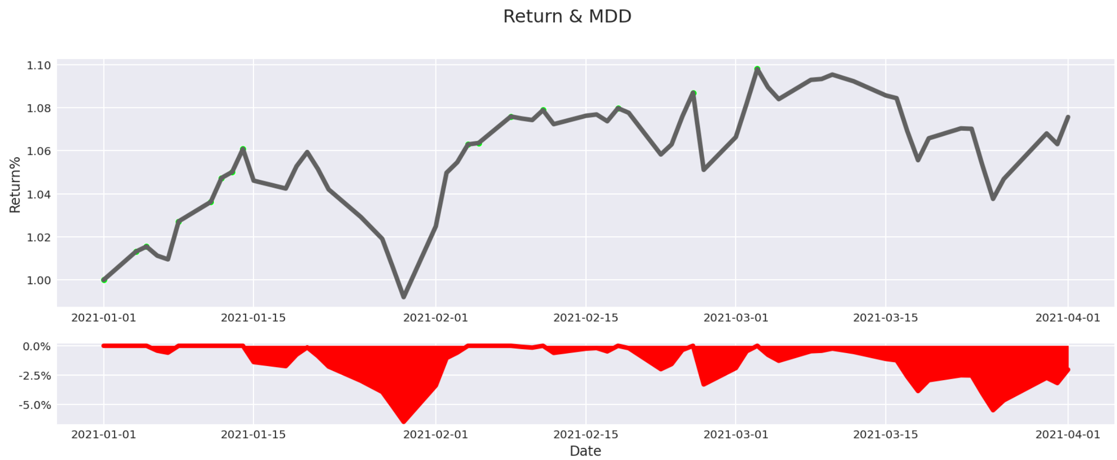
[41]:
```
port.Summary()
```

```
                             Period
From 2021-01-01  to 2021-04-01, 90 days.
------------------------------------------------------------
              Weights of Portfolio:
------------------------------------------------------------
ITC.NS                                      9.09%
GAIL.NS                                     9.09%
RELIANCE.NS                                 9.09%
INFY.NS                                     9.09%
BPCL.NS                                     9.09%
WIPRO.NS                                    9.09%
TCS.NS                                      9.09%
HDFCBANK.NS                                 9.09%
KOTAKBANK.NS                                9.09%
LT.NS                                       9.09%
risk_free                                   9.09%
------------------------------------------------------------


               Technical Indicator:
```

```
------------------------------------------------------------
Average Return :                                                   0.301
Average Standard Deviation :                       0.182
Sharpe Ratio :                                                        1.378
Sotino Ratio :                                                         1.934
Maximum Drop Down :                                      0.065
------------------------------------------------------------
```

[42]:

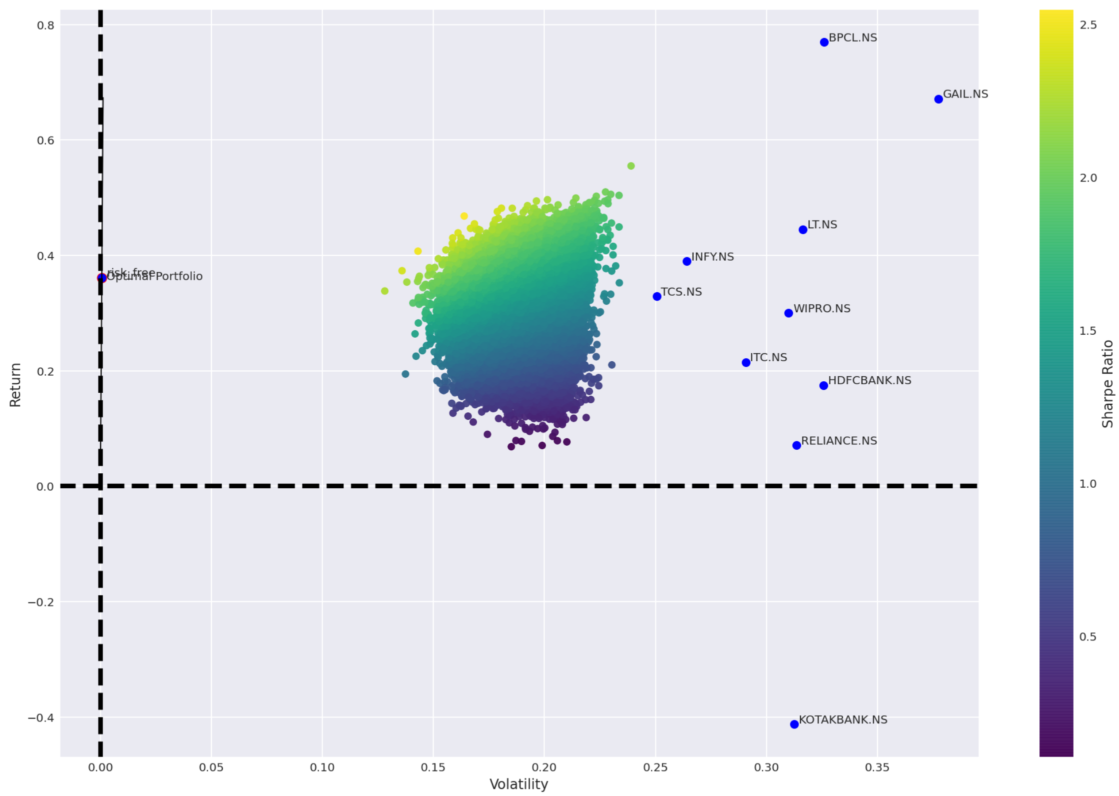
```
port.Return_Plot()
```



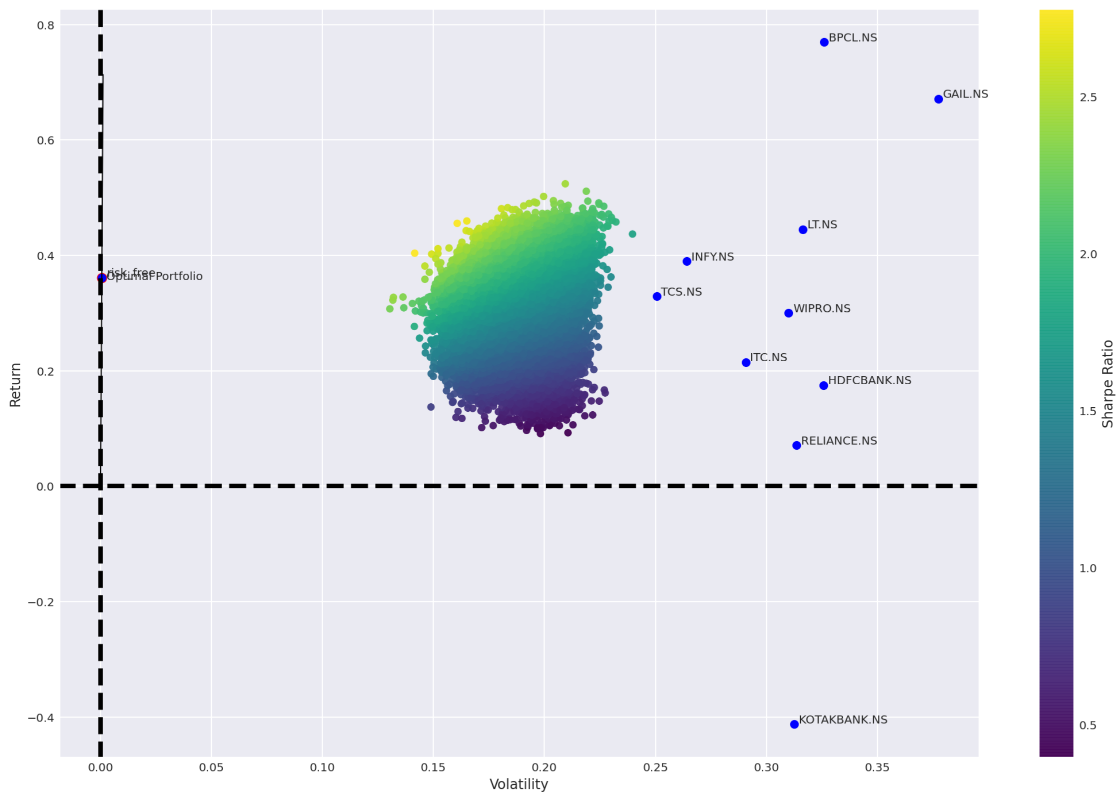
[43]:

```
port.set_optimize()
port.optimize_set()
port.Plot_Effcient_Frontier()
```

```
Begin :  2021-01-01 00:00:00
End :  2021-04-01 00:00:00
Rf :  0.05
```



[44]:
```
# CML
port.set_optimize(rf=0.01)
port.Plot_Effcient_Frontier()
```



[45]:
```
port.Get_Best_Portfolio()
```

[45]:
```
(606.5056807383834,
 [0.00010112492544409907,
  2.3502899429924264e-12,
  2.2313386469138176e-06,
  4.0054732161749244e-06,
  2.2318388158435086e-11,
  0.0,
  0.0,
  0.0,
  4.766230789861632e-06,
  3.013975053333845e-13,
  0.9999023957857084])
```

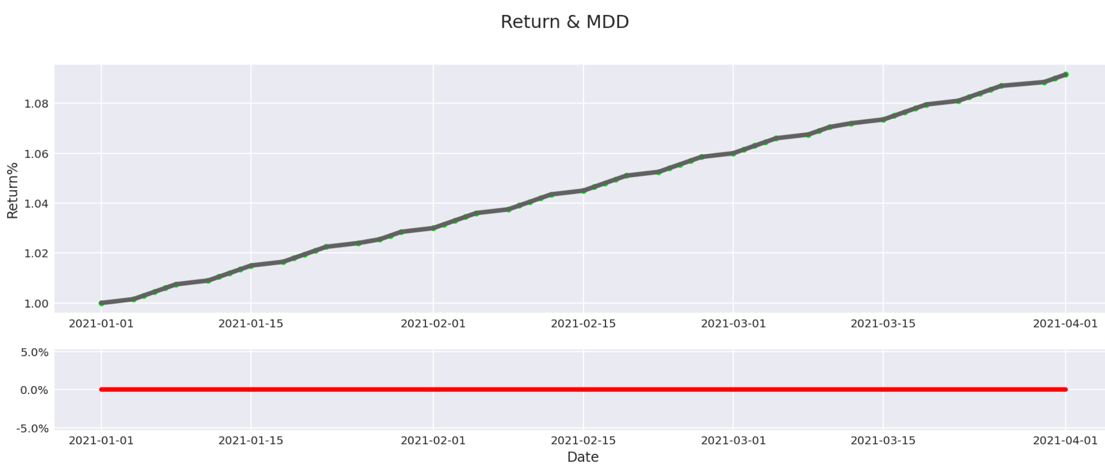
[46]:
```
port.set_weights(port.Get_Best_Portfolio()[1])
```

[47]:
```
port.Summary()
```

```
                    Period
From 2021-01-01  to 2021-04-01, 90 days.
------------------------------------------------------------
```

```
               Weights of Portfolio:
------------------------------------------------------------
ITC.NS                                          0.01%
GAIL.NS                                         0.00%
RELIANCE.NS                                     0.00%
INFY.NS                                         0.00%
BPCL.NS                                         0.00%
WIPRO.NS                                        0.00%
TCS.NS                                          0.00%
HDFCBANK.NS                                     0.00%
KOTAKBANK.NS                                    0.00%
LT.NS                                           0.00%
risk_free                                      99.99%
------------------------------------------------------------


                Technical Indicator:
------------------------------------------------------------
Average Return :                                                    0.362
Average Standard Deviation :                                0.001
Sharpe Ratio :                                                       537.522
Sotino Ratio :                                                        776.440
Maximum Drop Down :                                           0.000
------------------------------------------------------------
```

[48]:
```
port.Return_Plot()
```

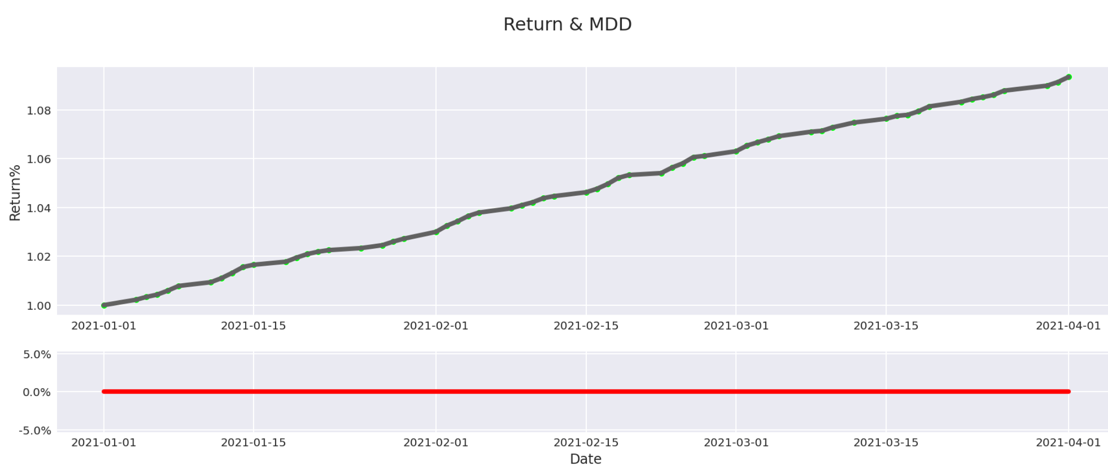


[49]:
```
port.set_weights(port.Get_Best_Portfolio(method='mdd')[1])
port.Summary()
```

```
                      Period
From 2021-01-01  to 2021-04-01, 90 days.
```

```
-----------------------------------------------------------
              Weights of Portfolio:
-----------------------------------------------------------
ITC.NS                                        0.90%
GAIL.NS                                       0.00%
RELIANCE.NS                                   0.00%
INFY.NS                                       0.00%
BPCL.NS                                       2.12%
WIPRO.NS                                      0.00%
TCS.NS                                        0.00%
HDFCBANK.NS                                   0.00%
KOTAKBANK.NS                                  0.00%
LT.NS                                         0.29%
risk_free                                    96.68%
-----------------------------------------------------------


               Technical Indicator:
-----------------------------------------------------------
Average Return :                                                0.369
Average Standard Deviation :                       0.009
Sharpe Ratio :                                                    37.319
Sotino Ratio :                                                     53.996
Maximum Drop Down :                                  0.000
-----------------------------------------------------------
```

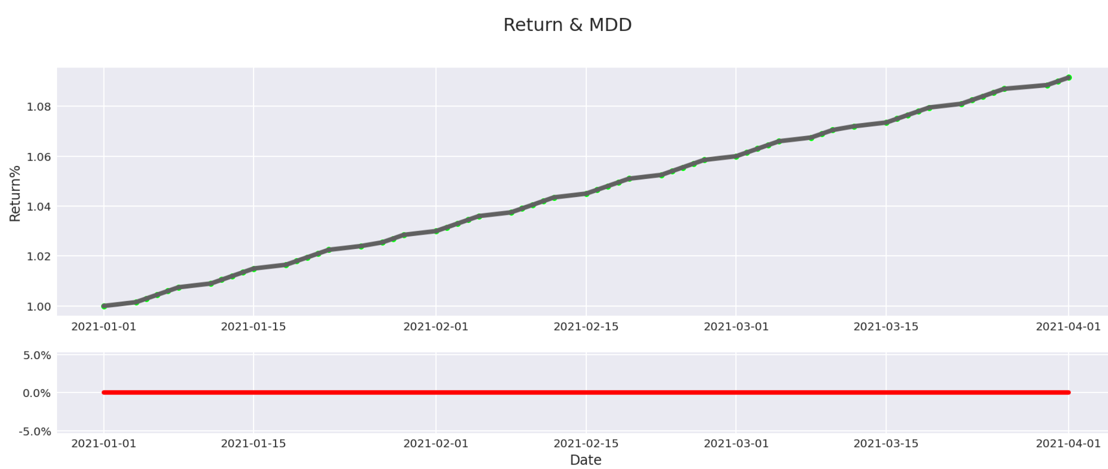
[50]:
```
port.Return_Plot()
```



[51]:
```
# Volatility
port.set_weights(port.Get_Best_Portfolio(method='std')[1])
port.Summary()
```

```
                      Period
From 2021-01-01  to 2021-04-01, 90 days.
------------------------------------------------------------
              Weights of Portfolio:
------------------------------------------------------------
ITC.NS                                  0.01%
GAIL.NS                                 0.00%
RELIANCE.NS                             0.00%
INFY.NS                                 0.00%
BPCL.NS                                 0.00%
WIPRO.NS                                0.00%
TCS.NS                                  0.00%
HDFCBANK.NS                             0.00%
KOTAKBANK.NS                            0.00%
LT.NS                                   0.00%
risk_free                              99.99%
------------------------------------------------------------


               Technical Indicator:
------------------------------------------------------------
Average Return :                                                    0.362
Average Standard Deviation :                            0.001
Sharpe Ratio :                                                       537.533
Sotino Ratio :                                                        776.264
Maximum Drop Down :                                       0.000
------------------------------------------------------------
```

[52]:
```
port.Return_Plot()
```



[53]:
```
table.head()
```

[53]:
```
Symbols          ITC.NS      GAIL.NS  …        LT.NS  risk_free
Date                                  …
2021-01-01  208.898636  119.134895  …  1297.000000     100.00
2021-01-04  208.459045  123.326057  …  1314.599976     100.15
2021-01-05  206.554199  124.578583  …  1306.300049     100.30
2021-01-06  200.644272  129.106964  …  1314.000000     100.45
2021-01-07  198.104477  128.577042  …  1338.949951     100.60

[5 rows x 11 columns]
```

[ ]: