In [1]:

```
import pandas as pd
import numpy as np
import yfinance  as yf
import matplotlib.pyplot as plt
from scipy.optimize import minimize, rosen, rosen_der
figsize = (12, 8)
```

In [2]:

```
symbols_list_good = ["APAM", "TMV", "PSK", "WPM", "TPL"]
symbols_list_poor = ["CIA", "CLF", "ATI", "EVR", "CRS"]
count_good = len(symbols_list_good)
count_poor = len(symbols_list_poor)
symbols_good = []
symbols_poor = []

for ticker in symbols_list_good:
    tick = yf.Ticker(ticker)
    history = tick.history(period='max')
    history['Symbol'] = ticker
    symbols_good.append(history)

for ticker in symbols_list_poor:
    tick = yf.Ticker(ticker)
    history = tick.history(period='max')
    history['Symbol'] = ticker
    symbols_poor.append(history)

df = pd.concat(symbols_good)
df = df.reset_index()
df = df[['Date', 'Close', 'Symbol']]
df = df.drop_duplicates()
price_good = df.pivot('Date','Symbol','Close').reset_index()
price_good.index = price_good.Date
price_good.drop(columns=['Date'], inplace=True)

df = pd.concat(symbols_poor)
df = df.reset_index()
df = df[['Date', 'Close', 'Symbol']]
df = df.drop_duplicates()
price_poor = df.pivot('Date','Symbol','Close').reset_index()
price_poor.index = price_poor.Date
price_poor.drop(columns=['Date'], inplace=True)
```

In [3]:

```
price_good
```

Out[3]:

Symbol	APAM	PSK	TMV	TPL	WPM
Date					
1980-03-17 00:00:00-05:00	NaN	NaN	NaN	4.629040	NaN
1980-03-18 00:00:00-05:00	NaN	NaN	NaN	4.397589	NaN
1980-03-19 00:00:00-05:00	NaN	NaN	NaN	4.496782	NaN
1980-03-20 00:00:00-05:00	NaN	NaN	NaN	4.513313	NaN
1980-03-21 00:00:00-05:00	NaN	NaN	NaN	4.529847	NaN
...	...	...	...	...	...
2023-04-28 00:00:00-04:00	34.669998	34.119999	106.459999	1477.650024	49.380001
2023-05-01 00:00:00-04:00	34.560001	33.990002	115.629997	1477.140015	48.900002
2023-05-02 00:00:00-04:00	33.400002	33.410000	107.300003	1435.979980	51.000000

In [5]:

```
price_poor
```

Out[5]:

Symbol	ATI	CIA	CLF	CRS	EVR
Date					
1973-02-21 00:00:00-05:00	NaN	NaN	0.836691	1.005395	NaN
1973-02-22 00:00:00-05:00	NaN	NaN	0.839934	1.010830	NaN
1973-02-23 00:00:00-05:00	NaN	NaN	0.836406	1.032568	NaN
1973-02-26 00:00:00-05:00	NaN	NaN	0.823336	1.005395	NaN
1973-02-27 00:00:00-05:00	NaN	NaN	0.820070	1.005395	NaN
...	...	...	...	...	...
2023-04-28 00:00:00-04:00	38.619999	1.87	15.380000	52.540001	114.070000
2023-05-01 00:00:00-04:00	37.919998	1.91	15.210000	51.599998	111.470001
2023-05-02 00:00:00-04:00	38.310001	1.87	15.280000	54.340000	108.699997
2023-05-03 00:00:00-04:00	38.000000	1.76	15.010000	52.130001	107.750000
2023-05-04 00:00:00-04:00	35.360001	1.78	14.260000	49.720001	106.839996

12662 rows × 5 columns

Monthly Price

In [6]:

```
month_price_good = price_good.resample("1 m").agg("last")
month_price_poor = price_poor.resample("1 m").agg("last")
```

In [7]:

```
month_price_good
```

Out[7]:

Symbol	APAM	PSK	TMV	TPL	WPM
Date					
1980-03-31 00:00:00-05:00	NaN	NaN	NaN	3.967750	NaN
1980-04-30 00:00:00-04:00	NaN	NaN	NaN	4.629040	NaN
1980-05-31 00:00:00-04:00	NaN	NaN	NaN	4.959685	NaN
1980-06-30 00:00:00-04:00	NaN	NaN	NaN	4.926621	NaN
1980-07-31 00:00:00-04:00	NaN	NaN	NaN	5.918558	NaN
...	...	...	...	...	...
2023-01-31 00:00:00-05:00	35.899502	35.885250	107.183395	1992.289917	45.590588
2023-02-28 00:00:00-05:00	32.970001	34.925434	124.760437	1777.014526	41.513950
2023-03-31 00:00:00-04:00	31.980000	33.427307	107.209999	1701.020020	48.160000
2023-04-30 00:00:00-04:00	34.669998	34.119999	106.459999	1477.650024	49.380001
2023-05-31 00:00:00-04:00	31.790001	31.879999	108.730003	1381.199951	51.410000

519 rows × 5 columns

In [8]:

```
month_price_poor
```

Out[8]:

Symbol	ATI	CIA	CLF	CRS	EVR
Date					
1973-02-28 00:00:00-05:00	NaN	NaN	0.816802	1.016264	NaN
1973-03-31 00:00:00-05:00	NaN	NaN	0.757993	1.010830	NaN
1973-04-30 00:00:00-04:00	NaN	NaN	0.807001	1.021700	NaN
1973-05-31 00:00:00-04:00	NaN	NaN	0.774329	0.907573	NaN
1973-06-30 00:00:00-04:00	NaN	NaN	0.803733	0.902138	NaN
...	...	...	...	...	...
2023-01-31 00:00:00-05:00	36.389999	2.40	21.350000	48.106876	129.068817
2023-02-28 00:00:00-05:00	40.650002	2.95	21.330000	48.146725	131.179993
2023-03-31 00:00:00-04:00	39.459999	3.71	18.330000	44.590260	115.379997
2023-04-30 00:00:00-04:00	38.619999	1.87	15.380000	52.540001	114.070000
2023-05-31 00:00:00-04:00	35.360001	1.78	14.260000	49.720001	106.839996

604 rows × 5 columns

In [9]:

```
month_ret_good = month_price_good.pct_change()
month_ret_good = month_ret_good.iloc[-61:-1]
month_ret_good
```

Out[9]:

Symbol Date	APAM	PSK	TMV	TPL	WPM
2018-05-31 00:00:00-04:00	0.023415	0.011268	-0.059903	0.296983	0.055784
2018-06-30 00:00:00-04:00	-0.066563	0.013760	-0.019744	-0.016060	0.009149
2018-07-31 00:00:00-04:00	0.142620	-0.002543	0.045621	0.064327	-0.050317
2018-08-31 00:00:00-04:00	-0.019694	0.013734	-0.036610	0.127729	-0.175863
2018-09-30 00:00:00-04:00	-0.022624	-0.016609	0.091691	0.033489	0.018626
2018-10-31 00:00:00-04:00	-0.154012	-0.020033	0.092256	-0.118763	-0.061143
2018-11-30 00:00:00-05:00	0.016420	-0.025133	-0.049891	-0.238028	-0.042626
2018-12-31 00:00:00-05:00	-0.188028	-0.005392	-0.157856	-0.064849	0.248721
2019-01-31 00:00:00-05:00	0.054726	0.060920	-0.007680	0.283921	0.079365
2019-02-28 00:00:00-05:00	0.203737	0.012688	0.043671	0.069326	0.032258
2019-03-31 00:00:00-04:00	-0.042966	0.013777	-0.149988	0.049050	0.094669
2019-04-30 00:00:00-04:00	0.125944	0.007298	0.064959	0.037084	-0.086337
2019-05-31 00:00:00-04:00	-0.146983	0.007748	-0.178299	-0.081424	0.024027
2019-06-30 00:00:00-04:00	0.163637	0.011470	-0.027167	0.067815	0.094117
2019-07-31 00:00:00-04:00	0.075218	0.021622	-0.005900	0.012973	0.080232
2019-08-31 00:00:00-04:00	-0.078612	0.008917	-0.279674	-0.178111	0.129703
2019-09-30 00:00:00-04:00	0.060060	0.005236	0.075544	-0.008608	-0.107786
2019-10-31 00:00:00-04:00	-0.031516	0.004777	0.029836	-0.123990	0.069741
2019-11-30 00:00:00-05:00	0.109542	-0.010002	0.012150	0.186194	-0.015319
2019-12-31 00:00:00-05:00	0.089316	0.020188	0.093399	0.157397	0.079817
2020-01-31 00:00:00-05:00	0.033416	0.009544	-0.198816	-0.032654	-0.010084
2020-02-29 00:00:00-05:00	-0.114557	-0.033228	-0.183738	-0.079276	-0.032258
2020-03-31 00:00:00-04:00	-0.248075	-0.079615	-0.307891	-0.440194	-0.030560
2020-04-30 00:00:00-04:00	0.369939	0.076904	-0.049719	0.499040	0.371595
2020-05-31 00:00:00-04:00	0.006729	0.017046	0.044225	0.029386	0.141241
2020-06-30 00:00:00-04:00	0.121850	-0.007823	-0.023823	0.014154	0.024418
2020-07-31 00:00:00-04:00	0.114769	0.041996	-0.127252	-0.103752	0.233371
2020-08-31 00:00:00-04:00	0.087315	0.013413	0.159343	-0.011764	-0.015754
2020-09-30 00:00:00-04:00	0.007233	-0.003895	-0.030245	-0.142694	-0.080570
2020-10-31 00:00:00-04:00	0.027443	-0.002310	0.100276	-0.002547	-0.060322
2020-11-30 00:00:00-05:00	0.144822	0.023716	-0.063330	0.354144	-0.148581
2020-12-31 00:00:00-05:00	0.118667	0.018670	0.034476	0.209674	0.066428
2021-01-31 00:00:00-05:00	-0.038538	-0.022743	0.108313	0.144759	-0.016052
2021-02-28 00:00:00-05:00	0.005516	-0.017816	0.172068	0.326588	-0.129778
2021-03-31 00:00:00-04:00	0.098316	0.027443	0.161203	0.443026	0.072786
2021-04-30 00:00:00-04:00	-0.023960	0.007168	-0.075488	-0.031005	0.081392
2021-05-31 00:00:00-04:00	0.020783	0.005071	-0.004912	-0.056508	0.165619
2021-06-30 00:00:00-04:00	-0.005090	0.018425	-0.130620	0.102941	-0.082257
2021-07-31 00:00:00-04:00	-0.053719	-0.002952	-0.117250	-0.067005	0.047197
2021-08-31 00:00:00-04:00	0.101573	-0.001829	0.001391	-0.088982	-0.020687
2021-09-30 00:00:00-04:00	-0.058506	-0.000230	0.081597	-0.108703	-0.165631
2021-10-31 00:00:00-04:00	0.012674	0.002079	-0.080257	0.053177	0.075306

Symbol	APAM	PSK	TMV	TPL	WPM
Date					
2021-11-30 00:00:00-05:00	-0.077783	-0.021301	-0.096510	-0.050955	0.037164
2021-12-31 00:00:00-05:00	0.065057	0.023804	0.051961	0.035428	0.027771
2022-01-31 00:00:00-05:00	-0.092989	-0.039385	0.108336	-0.139222	-0.060797
2022-02-28 00:00:00-05:00	-0.079576	-0.037768	0.037276	0.105795	0.086062
2022-03-31 00:00:00-04:00	0.032537	-0.007632	0.146941	0.140074	0.089948
2022-04-30 00:00:00-04:00	-0.183227	-0.069534	0.328366	0.010717	-0.057167
2022-05-31 00:00:00-04:00	0.221395	0.039221	0.045078	0.145902	-0.075608
2022-06-30 00:00:00-04:00	-0.073939	-0.035772	0.017755	-0.036614	-0.127814
2022-07-31 00:00:00-04:00	0.117796	0.059434	-0.080130	0.232409	-0.048016
2022-08-31 00:00:00-04:00	-0.137534	-0.047062	0.132969	0.003604	-0.106704
2022-09-30 00:00:00-04:00	-0.202310	-0.022434	0.263949	-0.032795	0.060984
2022-10-31 00:00:00-04:00	0.058671	-0.056517	0.196858	0.296326	0.010198
2022-11-30 00:00:00-05:00	0.237420	0.060848	-0.200050	0.125298	0.198869
2022-12-31 00:00:00-05:00	-0.143845	-0.039512	0.064786	-0.094680	0.001281
2023-01-31 00:00:00-05:00	0.239731	0.121877	-0.204697	-0.148612	0.170420
2023-02-28 00:00:00-05:00	-0.081603	-0.026747	0.163990	-0.108054	-0.089418
2023-03-31 00:00:00-04:00	-0.030027	-0.042895	-0.140673	-0.042765	0.160092
2023-04-30 00:00:00-04:00	0.084115	0.020722	-0.006996	-0.131315	0.025332

In [10]:

```
month_ret_poor = month_price_poor.pct_change()
month_ret_poor = month_ret_poor.iloc[-61:-1]
month_ret_poor
```

2020-12-31 00:00:00-05:00	0.243143	-0.096215	0.322434	0.191490	0.205763
2021-01-31 00:00:00-05:00	0.014311	0.055846	0.053571	0.072802	-0.004925
2021-02-28 00:00:00-05:00	0.155791	0.003306	-0.130378	0.309923	0.103220
2021-03-31 00:00:00-04:00	0.071211	-0.046129	0.507496	0.012051	0.099942
2021-04-30 00:00:00-04:00	0.104463	0.008636	-0.111885	-0.079709	0.063686
2021-05-31 00:00:00-04:00	0.052880	-0.106164	0.126540	0.272100	0.045730
2021-06-30 00:00:00-04:00	-0.148632	0.013410	0.071571	-0.160684	-0.034897
2021-07-31 00:00:00-04:00	-0.015348	0.013233	0.159555	-0.051467	-0.060879
2021-08-31 00:00:00-04:00	-0.130054	0.113806	-0.061200	-0.120675	0.061493
2021-09-30 00:00:00-04:00	-0.068869	0.040201	-0.155944	-0.018291	-0.042753
2021-10-31 00:00:00-04:00	-0.031870	0.037037	0.217062	-0.050885	0.135932
2021-11-30 00:00:00-05:00	-0.115528	-0.119565	-0.155952	-0.110104	-0.082375
2021-12-31 00:00:00-05:00	0.118680	-0.063492	0.069779	0.062227	-0.020548

Mean and Std

In [11]:

```
mean_good = month_ret_good.mean() * 12
mean_poor = month_ret_poor.mean() * 12
print(mean_good, mean_poor)
```

```
Symbol
APAM    0.199225
PSK     0.020415
TMV    -0.024964
TPL     0.375760
WPM     0.253247
dtype: float64 Symbol
ATI    0.217853
CIA   -0.181066
CLF    0.376969
CRS    0.172867
EVR    0.127372
dtype: float64
```

In [12]:

```
std_good = month_ret_good.std() * np.sqrt(12)
std_poor = month_ret_poor.std() * np.sqrt(12)
print(std_good, std_poor)
```

```
Symbol
APAM    0.420635
PSK     0.119354
TMV     0.445649
TPL     0.578978
WPM     0.370604
dtype: float64 Symbol
ATI    0.528737
CIA    0.422195
CLF    0.683722
CRS    0.552377
EVR    0.395849
dtype: float64
```

Distribution Chart - Good ESG Performance

In [13]:

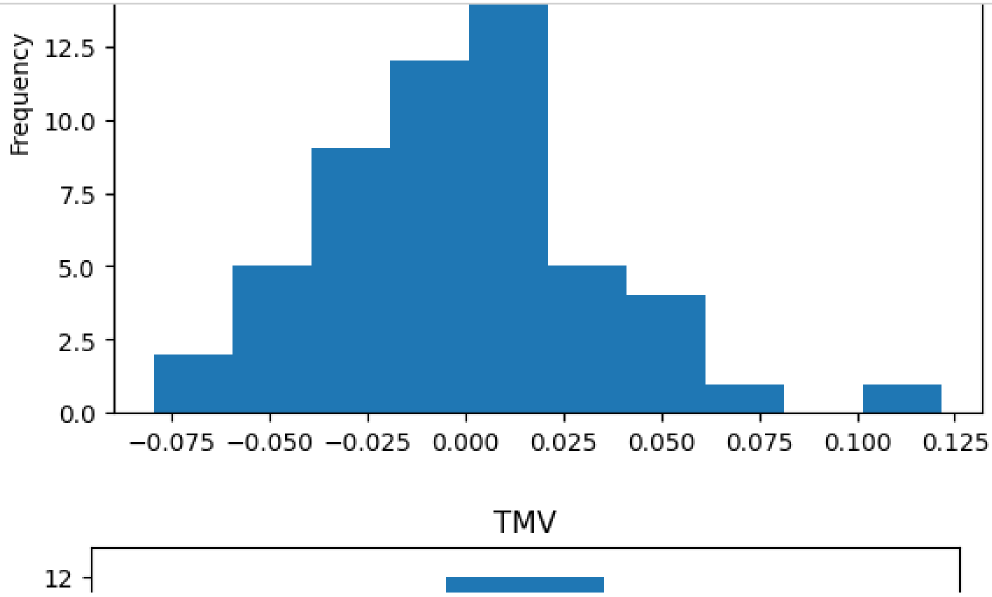
```
month_ret_good.columns
```

Out[13]:

```
Index(['APAM', 'PSK', 'TMV', 'TPL', 'WPM'], dtype='object', name='Symbol')
```

In [14]:

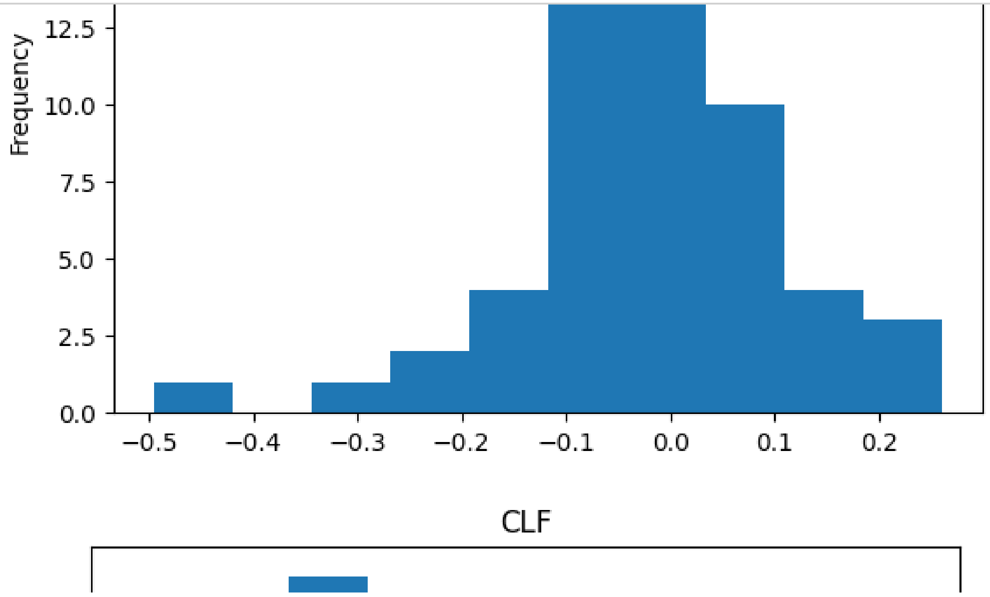
```
for symbol in month_ret_good.columns:
    month_ret_good[symbol].plot.hist()
    plt.title(symbol)
    plt.show()
```



Distribution Chart - Poor ESG Performance

In [15]:

```
for symbol in month_ret_poor.columns:
    month_ret_poor[symbol].plot.hist()
    plt.title(symbol)
    plt.show()
```



Covariance

In [16]:

```
cov_good = month_ret_good.cov()

# Alternative
# cov_good_matrix = month_ret_good.apply(lambda x: np.log(1+x)).cov()

cov_good
```

Out[16]:

Symbol Symbol	APAM	PSK	TMV	TPL	WPM
APAM	0.014744	0.003037	-0.000855	0.010675	0.003533
PSK	0.003037	0.001187	-0.001353	0.002021	0.001495
TMV	-0.000855	-0.001353	0.016550	0.006364	-0.004634
TPL	0.010675	0.002021	0.006364	0.027935	0.001989
WPM	0.003533	0.001495	-0.004634	0.001989	0.011446

In [17]:

```
cov_poor = month_ret_poor.cov()
cov_poor
```

Out[17]:

Symbol Symbol	ATI	CIA	CLF	CRS	EVR
ATI	0.023297	-0.000251	0.017828	0.019654	0.010675
CIA	-0.000251	0.014854	0.002120	-0.002786	-0.000269
CLF	0.017828	0.002120	0.038956	0.017465	0.013899
CRS	0.019654	-0.002786	0.017465	0.025427	0.012053
EVR	0.010675	-0.000269	0.013899	0.012053	0.013058

Assets

In [18]:

```
assets_good = pd.concat([mean_good, std_good], axis=1)
assets_good.columns = ['Returns', 'Volatility']
assets_good
```

Out[18]:

 Symbol	Returns	Volatility
APAM	0.199225	0.420635
PSK	0.020415	0.119354
TMV	-0.024964	0.445649
TPL	0.375760	0.578978
WPM	0.253247	0.370604

In [19]:

```
assets_poor = pd.concat([mean_poor, std_poor], axis=1)
assets_poor.columns = ['Returns', 'Volatility']
assets_poor
```

Out[19]:

	Returns	Volatility
Symbol		
ATI	0.217853	0.528737
CIA	-0.181066	0.422195
CLF	0.376969	0.683722
CRS	0.172867	0.552377
EVR	0.127372	0.395849

In [20]:

```
p_ret_good = []
p_vol_good = []
p_weights_good = []

p_ret_poor = []
p_vol_poor = []
p_weights_poor = []

num_assets_good = len(price_good.columns)
num_assets_poor = len(price_poor.columns)

num_portfolios = 10000
```

In [21]:

```
for portfolio in range(num_portfolios):
    weights = np.random.random(num_assets_good)
    weights = weights/np.sum(weights)
    p_weights_good.append(weights)
    returns = np.dot(weights, mean_good)
    p_ret_good.append(returns)
    var = np.dot(weights.T, np.dot(cov_good, weights))
    sd = np.sqrt(var)
    ann_sd = sd*np.sqrt(12)
    p_vol_good.append(ann_sd)
```

In [22]:

```
for portfolio in range(num_portfolios):
    weights = np.random.random(num_assets_poor)
    weights = weights/np.sum(weights)
    p_weights_poor.append(weights)
    returns = np.dot(weights, mean_poor)
    p_ret_poor.append(returns)
    var = np.dot(weights.T, np.dot(cov_poor, weights))
    sd = np.sqrt(var)
    ann_sd = sd*np.sqrt(12)
    p_vol_poor.append(ann_sd)
```

In [23]:

```
data_good = {'Returns':p_ret_good, 'Volatility':p_vol_good}
pd.DataFrame( data_good )
```

Out[23]:

	Returns	Volatility
0	0.190753	0.248365
1	0.114050	0.221376
2	0.166091	0.245586
3	0.143043	0.237012
4	0.069924	0.198926
...	...	...
9995	0.105569	0.177733
9996	0.164671	0.222356
9997	0.170010	0.235065
9998	0.131251	0.212400
9999	0.128720	0.192575

10000 rows × 2 columns

In [24]:

```
data_poor = {'Returns':p_ret_poor, 'Volatility':p_vol_poor}
pd.DataFrame( data_poor )
```

Out[24]:

	Returns	Volatility
0	0.182718	0.406392
1	0.122197	0.380312
2	0.231826	0.476502
3	0.219841	0.452881
4	0.170247	0.424867
...	...	...
9995	0.169072	0.394200
9996	0.108668	0.356541
9997	0.221065	0.452451
9998	0.119435	0.368271
9999	0.063551	0.309902

10000 rows × 2 columns

In [25]:

```
for counter, symbol in enumerate(price_good.columns.tolist()):
    data_good[symbol+' weight'] = [w[counter] for w in p_weights_good]

for counter, symbol in enumerate(price_poor.columns.tolist()):
    data_poor[symbol+' weight'] = [w[counter] for w in p_weights_poor]
```

In [26]:

```
rf = 0.025

portfolios_good = pd.DataFrame(data_good)
portfolios_good['Sharpe Ratio'] = (portfolios_good['Returns']-rf) / portfolios_good['Volatility']
portfolios_good.head()
```

Out[26]:

	Returns	Volatility	APAM weight	PSK weight	TMV weight	TPL weight	WPM weight	Sharpe Ratio
0	0.190753	0.248365	0.042099	0.165538	0.201436	0.280498	0.310428	0.667378
1	0.114050	0.221376	0.312501	0.316411	0.218094	0.098208	0.054786	0.402257
2	0.166091	0.245586	0.319176	0.357982	0.013466	0.140250	0.169126	0.574508
3	0.143043	0.237012	0.243020	0.265853	0.208162	0.185582	0.097383	0.498044
4	0.069924	0.198926	0.016266	0.316031	0.423288	0.072652	0.171763	0.225830

In [27]:

```
portfolios_poor = pd.DataFrame(data_poor)
portfolios_poor['Sharpe Ratio'] = (portfolios_poor['Returns']-rf) / portfolios_poor['Volatility']
portfolios_poor.head()
```

Out[27]:

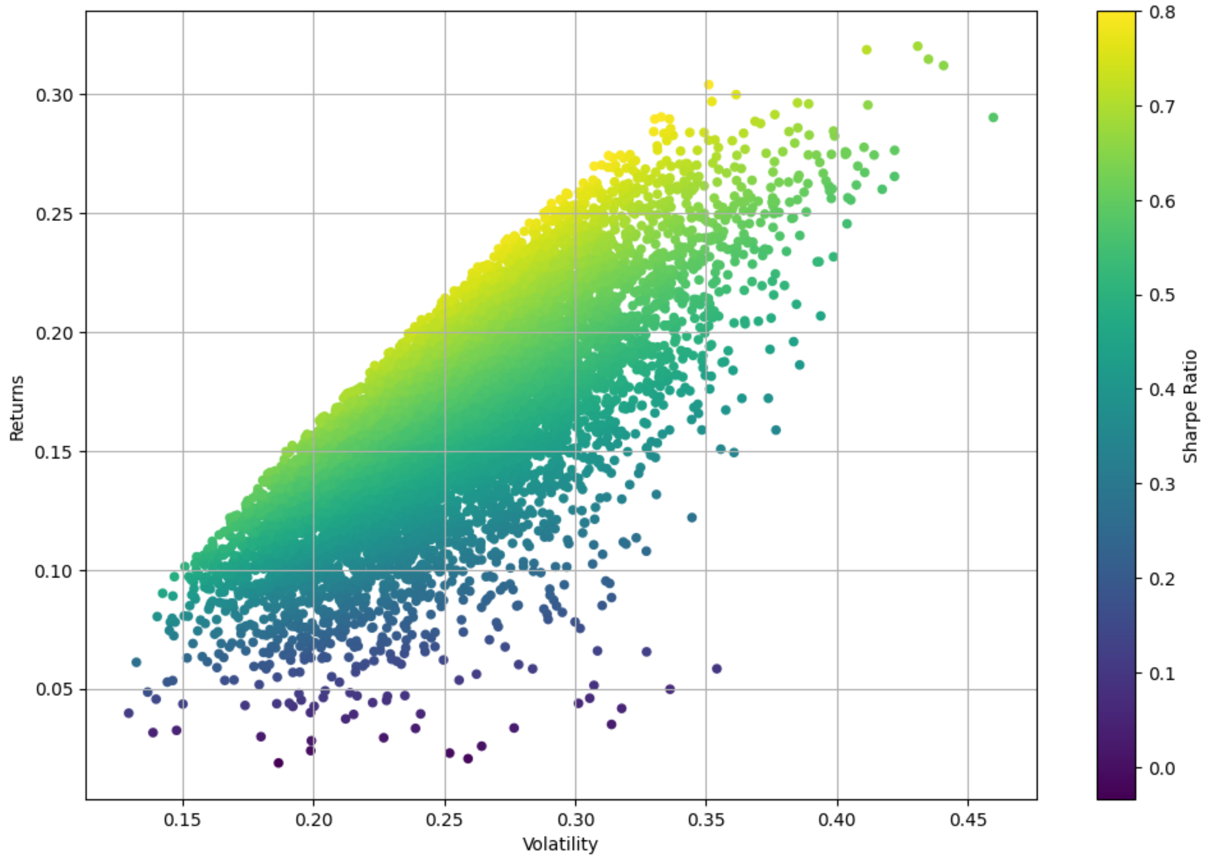
	Returns	Volatility	ATI weight	CIA weight	CLF weight	CRS weight	EVR weight	Sharpe Ratio
0	0.182718	0.406392	0.311236	0.091185	0.207856	0.075391	0.314331	0.388095
1	0.122197	0.380312	0.248329	0.197633	0.071863	0.337974	0.144202	0.255571
2	0.231826	0.476502	0.028482	0.023172	0.372945	0.350332	0.225070	0.434051
3	0.219841	0.452881	0.296855	0.053673	0.296766	0.177852	0.174853	0.430224
4	0.170247	0.424867	0.180039	0.286468	0.449943	0.057978	0.025572	0.341865

In [28]:

```
.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, figsize=figsize)
```

Out[28]:

```
<AxesSubplot:xlabel='Volatility', ylabel='Returns'>
```

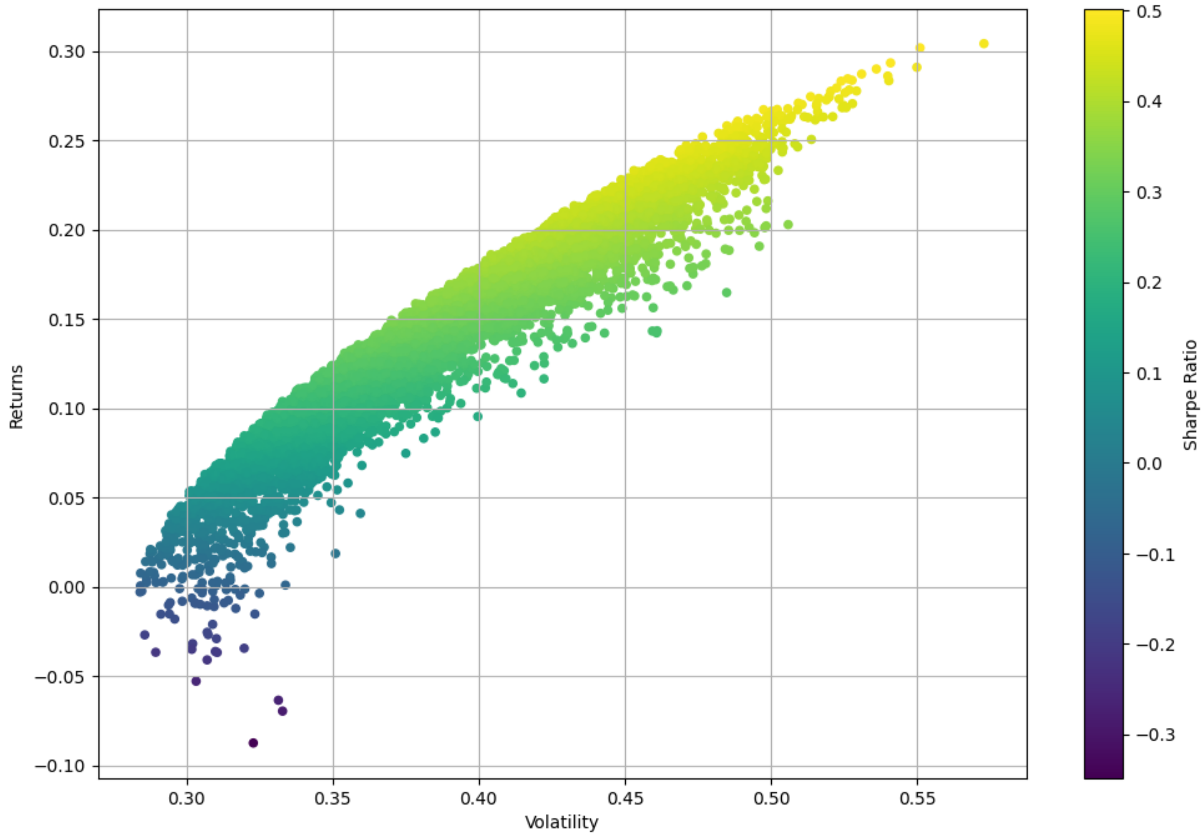


In [29]:

```
portfolios_poor.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, f
```

Out[29]:

```
<AxesSubplot:xlabel='Volatility', ylabel='Returns'>
```



Minimun Variance

In [30]:

```
portfolios_good[portfolios_good['Volatility']==portfolios_good['Volatility'].min()]
```

Out[30]:

	Returns	Volatility	APAM weight	PSK weight	TMV weight	TPL weight	WPM weight	Sharpe Ratio
7091	0.039632	0.129422	0.036122	0.614396	0.253556	0.015752	0.080173	0.113057

In [31]:

```
portfolios_poor[portfolios_poor['Volatility']==portfolios_poor['Volatility'].min()]
```

Out[31]:

	Returns	Volatility	ATI weight	CIA weight	CLF weight	CRS weight	EVR weight	Sharpe Ratio
3512	-0.002993	0.284099	0.035184	0.457749	0.019572	0.060506	0.42699	-0.098534

In [32]:

```
min_var_port_good = portfolios_good.iloc[portfolios_good['Volatility'].idxmin()]
min_var_port_good
```

Out[32]:

```
Returns          0.039632
Volatility       0.129422
APAM weight      0.036122
PSK weight       0.614396
TMV weight       0.253556
TPL weight       0.015752
WPM weight       0.080173
Sharpe Ratio     0.113057
Name: 7091, dtype: float64
```

In [33]:

```
min_var_port_poor = portfolios_poor.iloc[portfolios_poor['Volatility'].idxmin()]
min_var_port_poor
```

Out[33]:

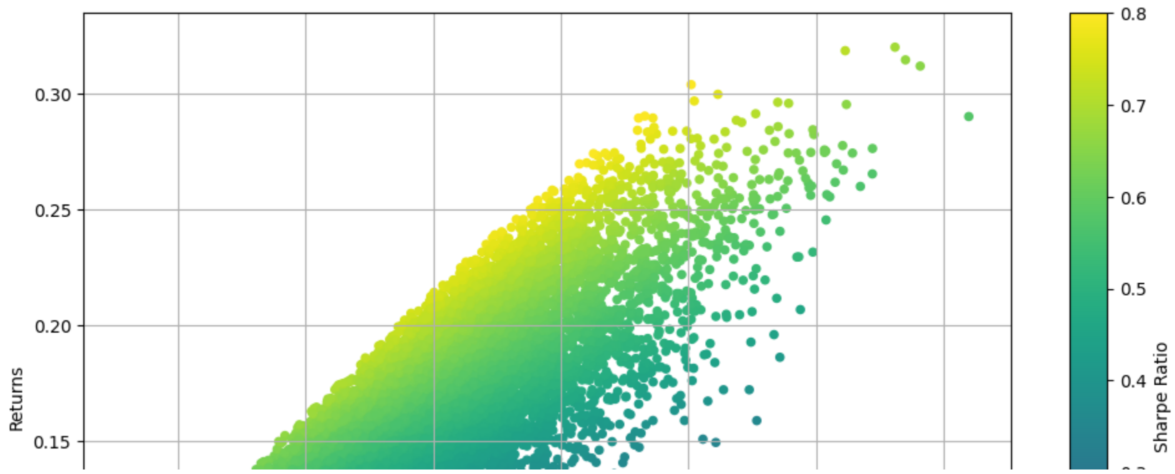
```
Returns         -0.002993
Volatility       0.284099
ATI weight       0.035184
CIA weight       0.457749
CLF weight       0.019572
CRS weight       0.060506
EVR weight       0.426990
Sharpe Ratio    -0.098534
Name: 3512, dtype: float64
```

In [34]:

```
portfolios_good.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, f
plt.scatter(min_var_port_good[1], min_var_port_good[0], color='r', s=50)
```

Out[34]:

```
<matplotlib.collections.PathCollection at 0x25ef845d0a0>
```

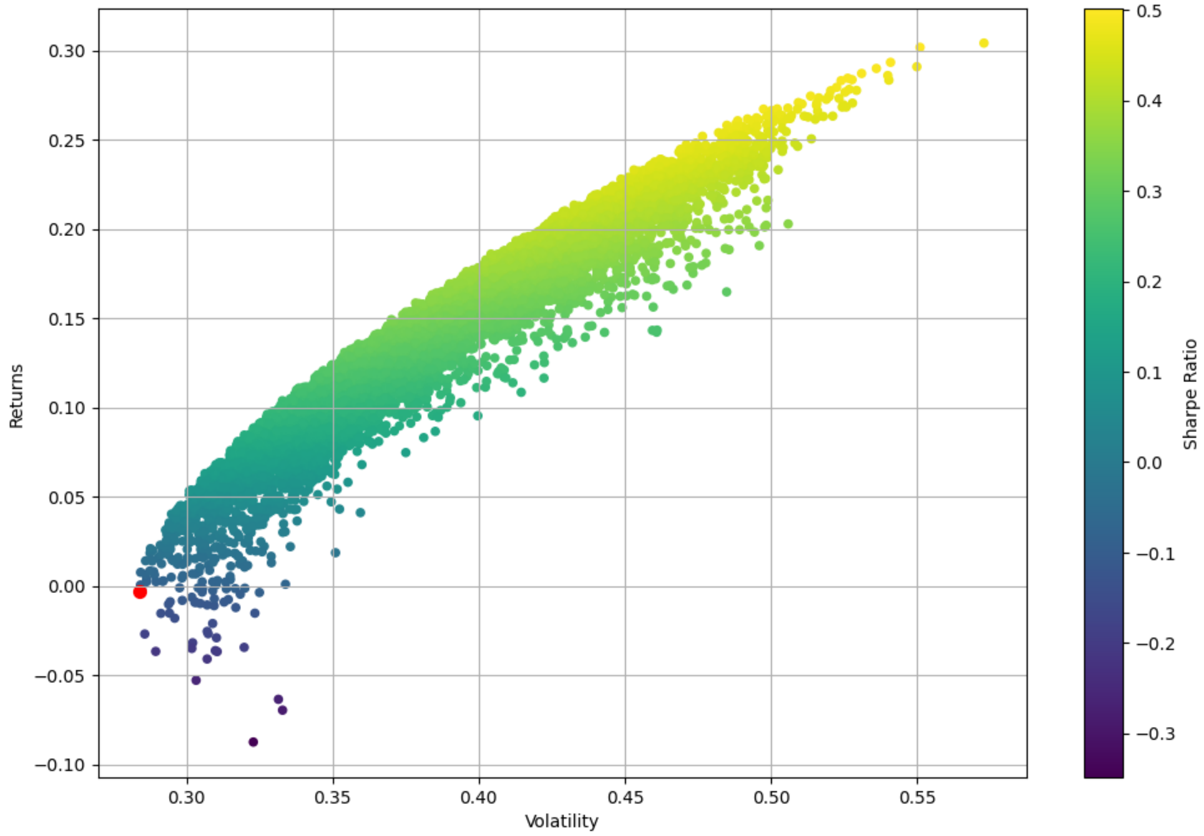


In [35]:

```
portfolios_poor.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, f
plt.scatter(min_var_port_poor[1], min_var_port_poor[0], color='r', s=50)
```

Out[35]:

<matplotlib.collections.PathCollection at 0x25ef8511df0>



Max Sharpe Ratio

$$SharpeRatio = \frac{E(R_i) - rf}{\sigma_i}$$

In [36]:

```
((portfolios_good['Returns']-rf)/portfolios_good['Volatility']).idxmax()
```

Out[36]:

4494

In [37]:

```
((portfolios_poor['Returns']-rf)/portfolios_poor['Volatility']).idxmax()
```

Out[37]:

6275

In [38]:

```
optimal_risky_port_good = portfolios_good.iloc[((portfolios_good['Returns']-rf)/portfolios_good['Volatil
optimal_risky_port_good
```

Out[38]:

```
Returns         0.289541
Volatility      0.330455
APAM weight     0.129416
PSK weight      0.006970
TMV weight      0.004008
TPL weight      0.375662
WPM weight      0.483943
Sharpe Ratio    0.800537
Name: 4494, dtype: float64
```

In [39]:

```
optimal_risky_port_poor = portfolios_poor.iloc[((portfolios_poor['Returns']-rf)/portfolios_poor['Volatil
optimal_risky_port_poor
```

Out[39]:

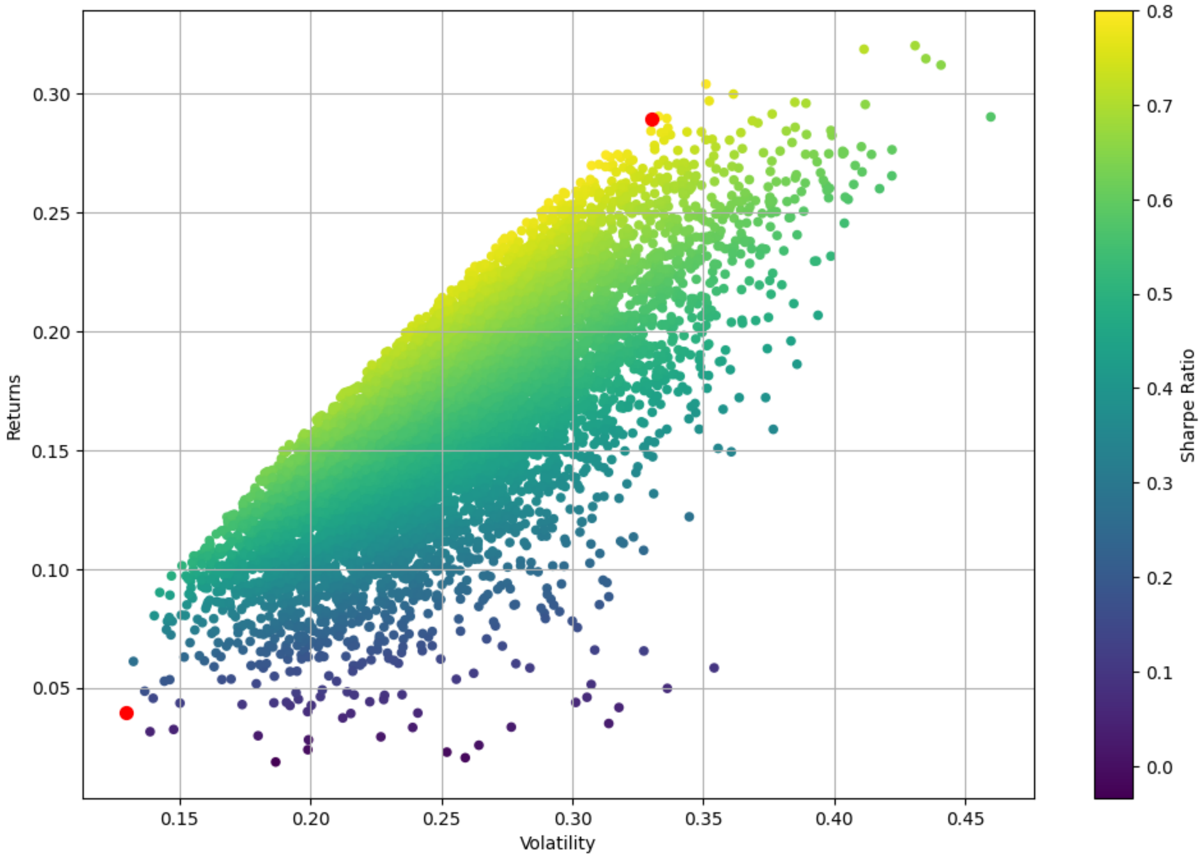
```
Returns         0.301669
Volatility      0.551034
ATI weight      0.196573
CIA weight      0.022603
CLF weight      0.654030
CRS weight      0.005238
EVR weight      0.121557
Sharpe Ratio    0.502091
Name: 6275, dtype: float64
```

In [40]:

```
portfolios_good.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, f
plt.scatter(min_var_port_good[1], min_var_port_good[0], color='r', s=50)
plt.scatter(optimal_risky_port_good[1], optimal_risky_port_good[0], color='r', s= 50)
```

Out[40]:

```
<matplotlib.collections.PathCollection at 0x25ef397e4c0>
```

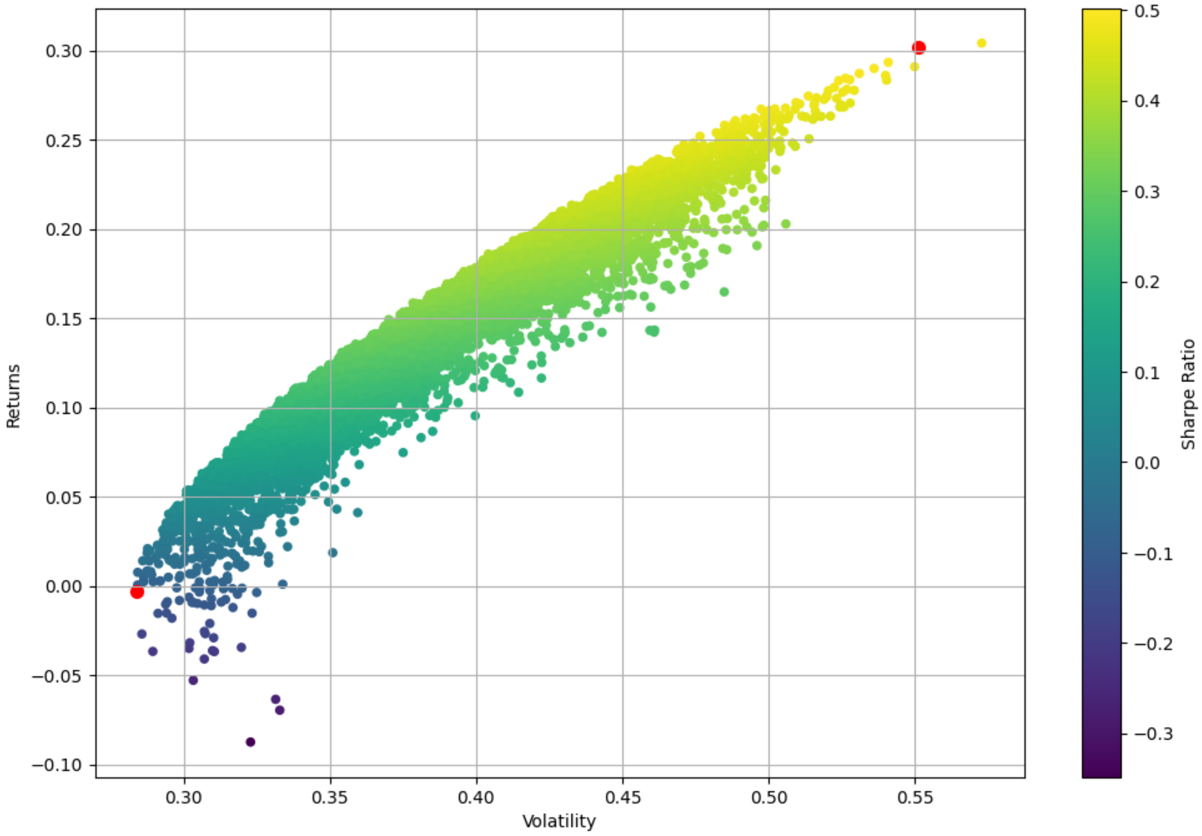


In [41]:

```
portfolios_poor.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, f
plt.scatter(min_var_port_poor[1], min_var_port_poor[0], color='r', s=50)
plt.scatter(optimal_risky_port_poor[1], optimal_risky_port_poor[0], color='r', s= 50)
```

Out[41]:

```
<matplotlib.collections.PathCollection at 0x25ef39e19a0>
```



Capital Allocation Line

$$E(R_P) = rf + \frac{E(R_i) - rf}{\sigma_i}\sigma_p$$

In [42]:

```
cal_x_good = []
cal_y_good = []

cal_x_poor = []
cal_y_poor = []
```

In [43]:

```
for er in np.linspace(rf, max(p_ret_good), 20):
    sd = (er - rf)/((optimal_risky_port_good[0]-rf)/optimal_risky_port_good[1])
    cal_x_good.append(sd)
    cal_y_good.append(er)

for er in np.linspace(rf, max(p_ret_poor), 20):
    sd = (er - rf)/((optimal_risky_port_poor[0]-rf)/optimal_risky_port_poor[1])
    cal_x_poor.append(sd)
    cal_y_poor.append(er)
```

In [44]:

```
data2_good = {'cal_y':cal_y_good, 'cal_x':cal_x_good}
cal_good = pd.DataFrame(data2_good)
cal_good.head()
```

Out[44]:

	cal_y	cal_x
0	0.025000	0.000000
1	0.040537	0.019408
2	0.056074	0.038817
3	0.071611	0.058225
4	0.087149	0.077634

In [45]:

```
data2_poor = {'cal_y':cal_y_poor, 'cal_x':cal_x_poor}
cal_poor = pd.DataFrame(data2_poor)
cal_poor.head()
```

Out[45]:

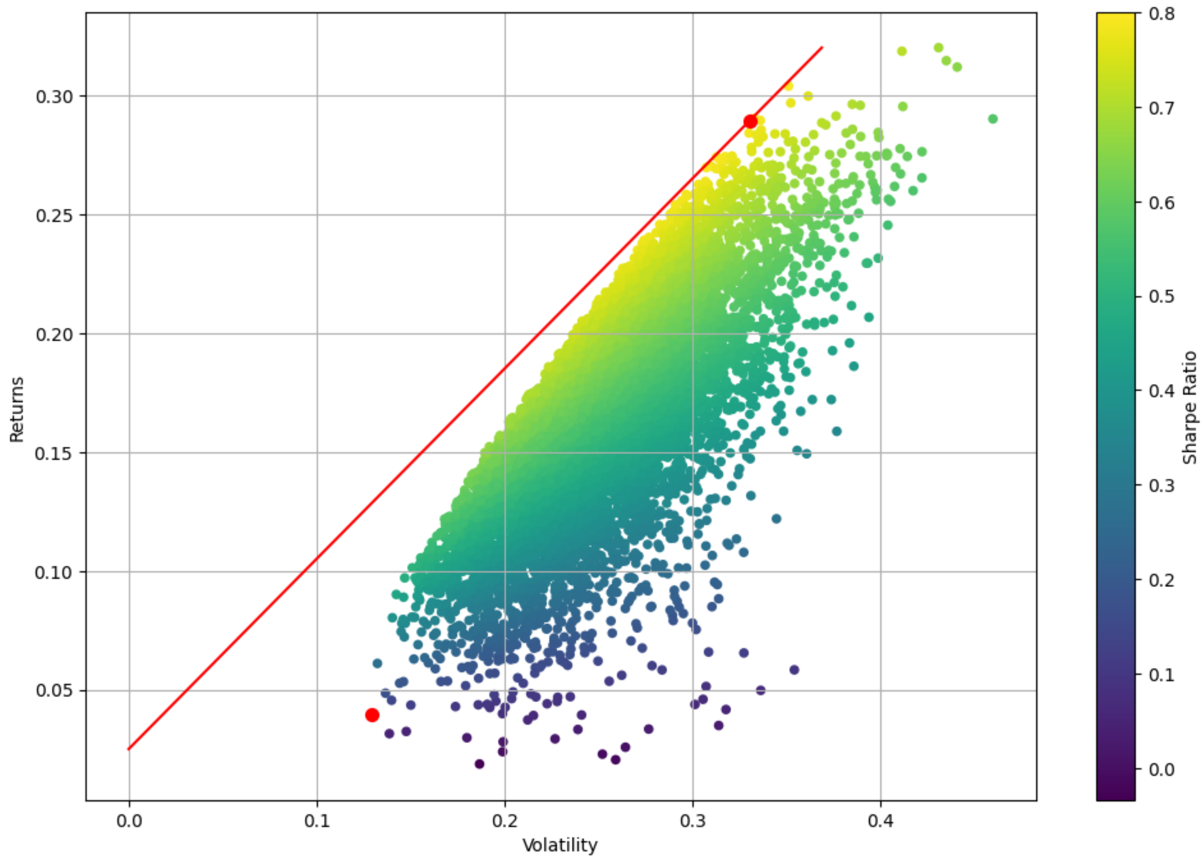
	cal_y	cal_x
0	0.025000	0.000000
1	0.039687	0.029251
2	0.054374	0.058503
3	0.069061	0.087754
4	0.083748	0.117006

In [46]:

```
portfolios_good.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, f
plt.scatter(min_var_port_good[1], min_var_port_good[0], color='r', s=50)
plt.scatter(optimal_risky_port_good[1], optimal_risky_port_good[0], color='r', s= 50)
plt.plot(cal_x_good, cal_y_good, color='r')
```

Out[46]:

```
[<matplotlib.lines.Line2D at 0x25ef3b41d60>]
```

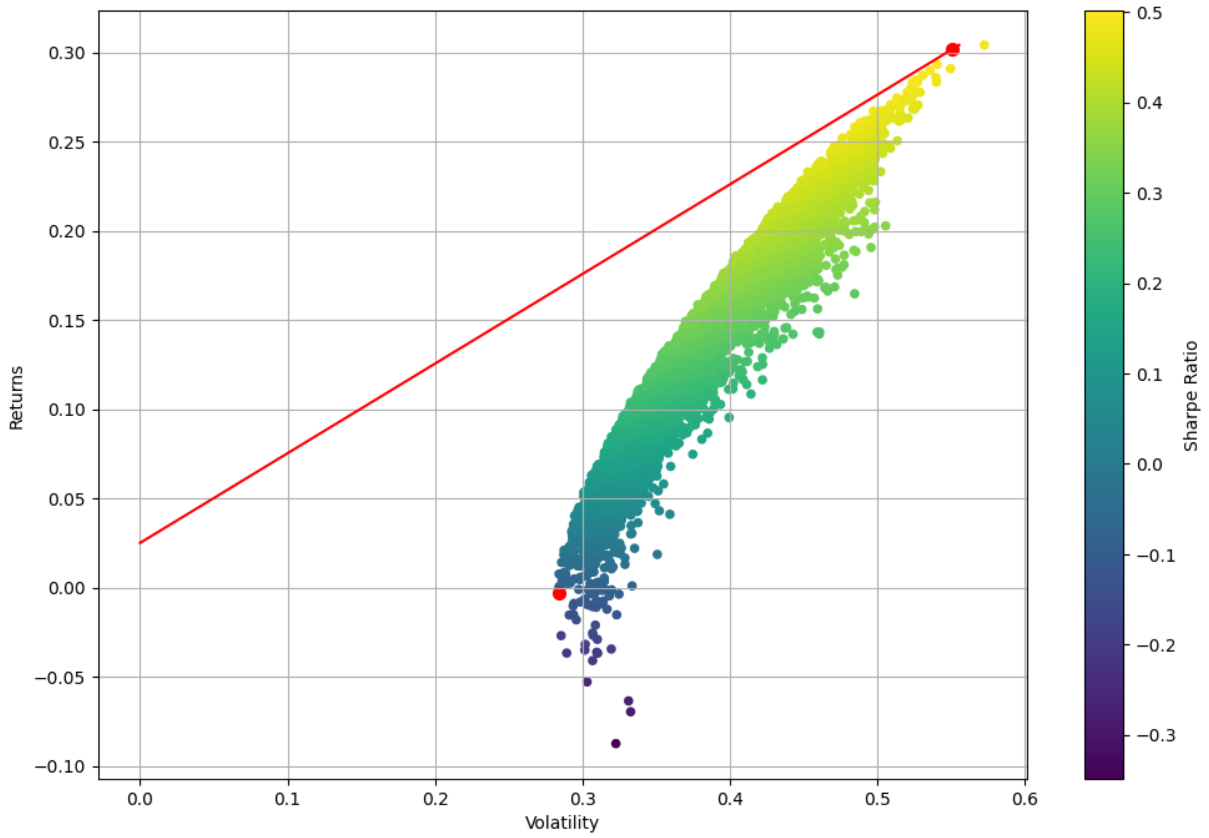


In [47]:

```
portfolios_poor.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, f
plt.scatter(min_var_port_poor[1], min_var_port_poor[0], color='r', s=50)
plt.scatter(optimal_risky_port_poor[1], optimal_risky_port_poor[0], color='r', s= 50)
plt.plot(cal_x_poor, cal_y_poor, color='r')
```

Out[47]:

```
[<matplotlib.lines.Line2D at 0x25ef8b50e50>]
```



Efficient Frontier

In [48]:

```
def get_ret_vol_sr(weights, month_ret):
    weights = np.array(weights)
    ret = np.sum(month_ret.mean() * weights) * 12
    vol = np.sqrt(np.dot(weights.T, np.dot(month_ret.cov()*12, weights)))
    sr = ret/vol
    return np.array([ret, vol, sr])

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

In [49]:

```
upper_bound_good = max(data_good["Returns"])
lower_bound_good = min(data_good["Returns"])
frontier_y_good = np.linspace(lower_bound_good - 0.05, upper_bound_good + 0.05, 50)

upper_bound_poor = max(data_poor["Returns"])
lower_bound_poor = min(data_poor["Returns"])
frontier_y_poor = np.linspace(lower_bound_poor - 0.05, upper_bound_poor + 0.05, 50)
```

In [50]:

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

In [51]:

```
bounds_good = tuple( [ (0,1) for i in range(count_good) ] )
bounds_poor = tuple( [ (0,1) for i in range(count_poor) ] )

init_guess_good = [1/count_good] * count_good
init_guess_poor = [1/count_poor] * count_poor
```

In [52]:

```
frontier_x_good = []

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

    result = minimize(minimize_volatility,init_guess_good, month_ret_good, method='SLSQP', bounds=bounds_
    frontier_x_good.append(result['fun'])


frontier_x_poor = []

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

    result = minimize(minimize_volatility,init_guess_poor, month_ret_poor, method='SLSQP', bounds=bounds_
    frontier_x_poor.append(result['fun'])
```

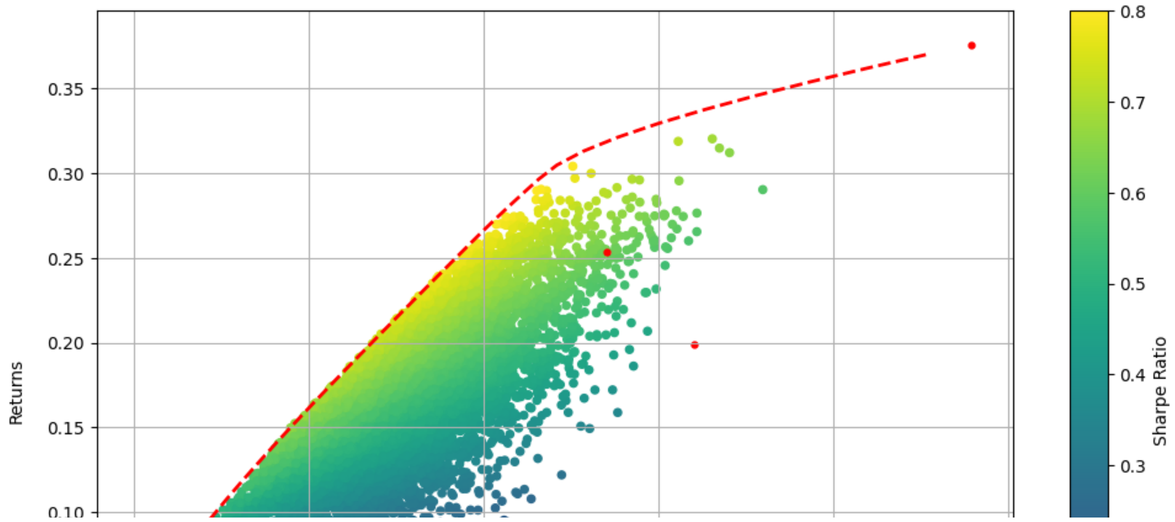
In [53]:

```
plt.figure(figsize=(12,8))
portfolios_good.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, f
plt.plot(frontier_x_good,frontier_y_good, 'r--', linewidth=2)

# plt.scatter(max_sr_vol, max_sr_ret,c='red', s=50) # red dot
# plt.scatter(min_var_vol, min_var_ret,c='red',marker='x', s=50) # red dot
# plt.scatter(max_sr_vol2, max_sr_ret2,c='purple', s=50) # red dot
# plt.scatter(min_var_vol2, min_var_ret2,c='purple',marker='x', s=50) # red dot

plt.scatter(std_good, mean_good,c='red',marker='.', s=50) # red dot
# plt.savefig('cover.png')
plt.show()
```

<Figure size 1200x800 with 0 Axes>



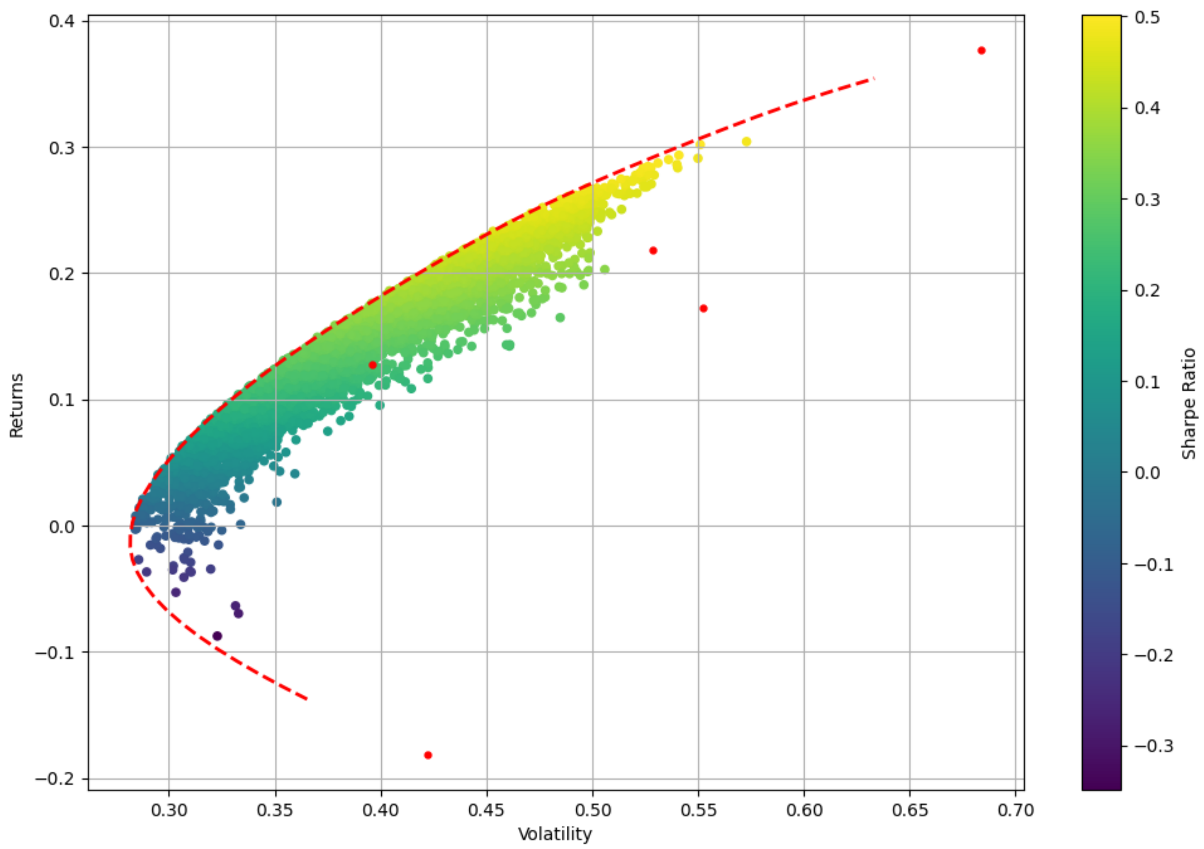
In [54]:

```
plt.figure(figsize=(12,8))
portfolios_poor.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, f
plt.plot(frontier_x_poor,frontier_y_poor, 'r--', linewidth=2)

# plt.scatter(max_sr_vol, max_sr_ret,c='red', s=50) # red dot
# plt.scatter(min_var_vol, min_var_ret,c='red',marker='x', s=50) # red dot
# plt.scatter(max_sr_vol2, max_sr_ret2,c='purple', s=50) # red dot
# plt.scatter(min_var_vol2, min_var_ret2,c='purple',marker='x', s=50) # red dot

plt.scatter(std_poor, mean_poor,c='red',marker='.', s=50) # red dot
# plt.savefig('cover.png')
plt.show()
```

<Figure size 1200x800 with 0 Axes>



Utility Function

In [55]:

```
A= 4
portfolios_good["Utility"] = portfolios_good["Returns"] - 0.5 * A * portfolios_good["Volatility"] ** 2
max_u_port_good = portfolios_good.iloc[portfolios_good['Utility'].idxmax()]
max_u_port_good
```

Out[55]:

```
Returns          0.227310
Volatility       0.262840
APAM weight      0.065354
PSK weight       0.062455
TMV weight       0.127564
TPL weight       0.225488
WPM weight       0.519138
Sharpe Ratio     0.769709
Utility          0.089140
Name: 6024, dtype: float64
```

In [56]:

```
portfolios_poor["Utility"] = portfolios_poor["Returns"] - 0.5 * A * portfolios_poor["Volatility"] ** 2
max_u_port_poor = portfolios_poor.iloc[portfolios_poor['Utility'].idxmax()]
max_u_port_poor
```

Out[56]:

```
Returns          0.103741
Volatility       0.332814
ATI weight       0.227001
CIA weight       0.209985
CLF weight       0.082365
CRS weight       0.000859
EVR weight       0.479788
Sharpe Ratio     0.236593
Utility         -0.117788
Name: 1115, dtype: float64
```

In [57]:

```
maximize_u_good = max_u_port_good["Utility"]
max_sigma_good = max(portfolios_good["Volatility"])
min_sigma_good = min(portfolios_good["Volatility"])

indifference_sigma_good = np.linspace(min_sigma_good - 0.025, max_sigma_good, 100)
indifference_return_good = maximize_u_good + 0.5 * A * np.square( indifference_sigma_good )


maximize_u_poor = max_u_port_poor["Utility"]
max_sigma_poor = max(portfolios_poor["Volatility"])
min_sigma_poor = min(portfolios_poor["Volatility"])

indifference_sigma_poor = np.linspace(min_sigma_poor - 0.025, max_sigma_poor, 100)
indifference_return_poor = maximize_u_poor + 0.5 * A * np.square( indifference_sigma_poor )
```

In [58]:

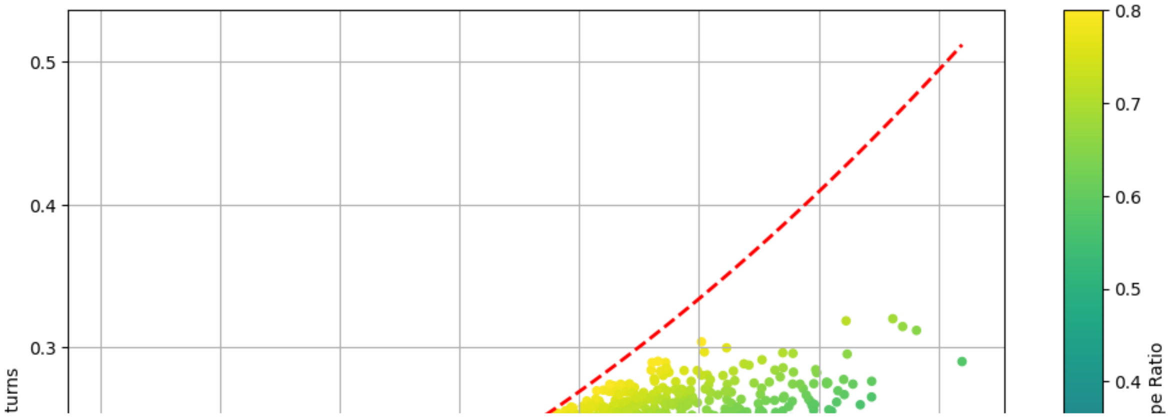
```
plt.figure(figsize=(12,8))
portfolios_good.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, f
plt.plot(indifference_sigma_good,indifference_return_good, 'r--', linewidth=2)

plt.scatter(max_u_port_good["Volatility"], max_u_port_good["Returns"],c='red',marker='.', s=50) # red do
```

Out[58]:

```
<matplotlib.collections.PathCollection at 0x25ef9c76190>

<Figure size 1200x800 with 0 Axes>
```



In [59]:

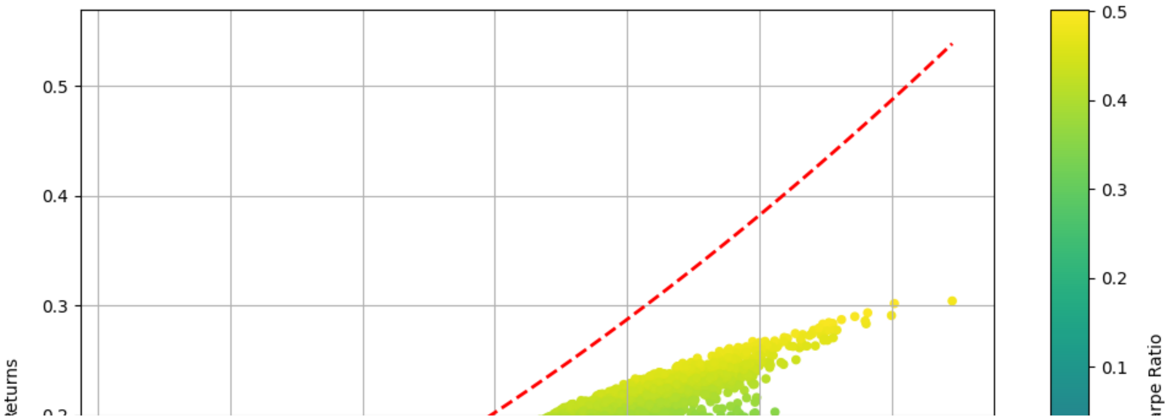
```
plt.figure(figsize=(12,8))
portfolios_poor.plot.scatter(x='Volatility', y='Returns', c="Sharpe Ratio", cmap='viridis', grid=True, f
plt.plot(indifference_sigma_poor,indifference_return_poor, 'r--', linewidth=2)

plt.scatter(max_u_port_poor["Volatility"], max_u_port_poor["Returns"],c='red',marker='.', s=50) # red do
```

Out[59]:

```
<matplotlib.collections.PathCollection at 0x25ef8b9c310>

<Figure size 1200x800 with 0 Axes>
```



Optimal compelete portfolio

In [60]:

```
optimal_y_good = (optimal_risky_port_good[0] - rf) / (A*optimal_risky_port_good[1]**2)
optimal_y_good
```

Out[60]:

```
0.6056320748042352
```

In [61]:

```
optimal_u_good = rf + optimal_y_good * (optimal_risky_port_good[0] - rf) - 0.5*A*optimal_y_good**2*optim
optimal_u_good
```

Out[61]:

```
0.10510737673757917
```

In [62]:

```
optimal_sigma_good = optimal_y_good * optimal_risky_port_good[1]
optimal_er_good = rf + optimal_y_good * (optimal_risky_port_good[0] - rf)
print("optimal deviation = ", optimal_sigma_good, "\n",
      "optimal expected return = ", optimal_er_good, sep = '')
```

```
optimal deviation = 0.20013417591403418
optimal expected return = 0.1852147534751583
```

In [63]:

```
optimal_y_poor = (optimal_risky_port_poor[0] - rf) / (A*optimal_risky_port_poor[1]**2)
optimal_y_poor
```

Out[63]:

```
0.2277950234609161
```

In [64]:

```
optimal_u_poor = rf + optimal_y_poor * (optimal_risky_port_poor[0] - rf) - 0.5*A*optimal_y_poor**2*optim
optimal_u_poor
```

Out[64]:

```
0.05651193245376958
```

In [65]:

```
optimal_sigma_poor = optimal_y_poor * optimal_risky_port_poor[1]
optimal_er_poor = rf + optimal_y_poor * (optimal_risky_port_poor[0] - rf)
print("optimal deviation = ", optimal_sigma_poor, "\n",
      "optimal expected return = ", optimal_er_poor, sep = '')
```

```
optimal deviation = 0.12552277174634408
optimal expected return = 0.08802386490753916
```

Summary Graph

In [66]:

```
# Mean-var frontiers
plt.plot(frontier_x_good,frontier_y_good, 'g', linewidth=2, label = "Mean-variance frontier(good)")
plt.plot(frontier_x_poor,frontier_y_poor, 'r', linewidth=2, label = "Mean-variance frontier(poor)")

# Cal
plt.plot(cal_x_good, cal_y_good, 'g--', linewidth=2, label = "Cal(good)")
plt.plot(cal_x_poor, cal_y_poor, 'r--', linewidth=2, label = "Cal(poor)")

# Indifference curve
plt.plot(indifference_sigma_good,indifference_return_good, 'greenyellow', linewidth=2, label = "Indiffer
plt.plot(indifference_sigma_poor,indifference_return_poor, 'tomato', linewidth=2, label = "Indifference 

# Optimal risky portfolio
plt.scatter(optimal_risky_port_good[1], optimal_risky_port_good[0], c = 'greenyellow', marker='.', s=300
plt.scatter(optimal_risky_port_poor[1], optimal_risky_port_poor[0], c = 'tomato', marker='.', s=300, zor

# Optimal complete portfolio
plt.scatter(optimal_sigma_good, optimal_er_good, c = 'g', marker='.', s=300, zorder=100, label = "Optima
plt.scatter(optimal_sigma_poor, optimal_er_poor, c = 'r', marker='.', s=300, zorder=100, label = "Optima

# Legend
plt.legend(loc="lower right", fontsize = "xx-small")
```

Out[66]:

```
<matplotlib.legend.Legend at 0x25efa9f7550>
```