Optimization_CAPM

June 4, 2022

[164]:
```
import numpy as np
import pandas as pd
from pandas_datareader import data as wb
import matplotlib.pyplot as plt
from math import sqrt
%matplotlib inline
```

Suppose we have decided to buy 3 stocks: 1 of Apple, 1 of Credit Agricole and 1 of Amgen.

As a rational investor, we would be interested in knowing how each of these stocks performed historically before we can proceed to buy them. We divide this analysis in 2 parts.

In part 1, we first look at how each stock performed in the past by calculating their historical average returns and volatility. We combine these stocks into a portfolio and calculate the expected return of the portfolio. Since, we do not have an unlimitted amount of money, we cannot invest as much as we want in every stock. Thus, we allocate an equal amount -that we call weight- to each stock and the sum of weights must always equal 1. Hence, the expected return of the portfolio is be the sum of the weighted return of each stock.

`exp. return = weight(1)•return(1) + ... + weight(n)•return(n)`

Afterwards, we calculate the expected risk or volatility of our equally weigheted portfolio. It is simply the square of the sum of their respective weighted standard deviations.

`exp. risk = ((weight(1)risk(1) + ... + weight(n)risk(n))^2`

In part 2, instead of only considering allocating our money equally which might not be the best way to do so, we consider 10,000 ways of allocating the same money to the same portfolio but with different weights. We intend to have 10,000 different scenarios for this analysis. Once we have all these combinations, we look for two types of combinations : (1) the combination that yields the lowest risk regardless of the return (called the minimum variance portfolio) and (2) the combination that yields the highest return-risk tradeoff, that is, the combination that yields the highest return relative to its risk (called the optimal portfolio). For the latter, we calculte the sharpe ratio of all combinations and pick the one with the highest sharpe ratio. Last, we plot these combinations into graph and describe the efficient frontier.

In part 3, we draw conclusions based on our analysis.

0.1 Part 1: Risk-Return Analysis

Note : We have prepared a group of Python functions in a different file called funct that we import here.

[165]:
```python
from funct import Return, Risk, normalize, Graph
```

Let import daily stock prices for the companies we're interested in from yahoo finance. For the time period, we choose from january 2, 2017 to May 31, 2022.

[166]:
```python
tickers = ["AAPL", "ACA.PA", "AMGN","^GSPC"]
df = pd.DataFrame()

for t in tickers:
    df[t]=wb.DataReader(t, data_source ="yahoo", start = "2017-01-02", end␣
 ↪="2022-05-31")['Adj Close']
```

[167]:
```python
df.info() # Check the data for missing values
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1362 entries, 2017-01-03 to 2022-05-31
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   AAPL    1362 non-null   float64
 1   ACA.PA  1349 non-null   float64
 2   AMGN    1362 non-null   float64
 3   ^GSPC   1362 non-null   float64
dtypes: float64(4)
memory usage: 53.2 KB
```

AAPL, ACA.PA, and AMGN are the market acronyms of Apple.inc, Credit Agricole and Amgen respectively.

Our data have some missing values in column ACA.PA. We can double check it with isna() method.

[168]:
```python
df.isna().sum()
```

[168]:
```
AAPL       0
ACA.PA    13
AMGN       0
^GSPC      0
dtype: int64
```

These missing values are due to the fact that Credit Agricole is France based firm as opposed to Amgen and Apple which are in the US.

We deal with these 13 missing values with the back filling method.

[169]:
```python
df['ACA.PA']= df['ACA.PA'].fillna(method= "ffill")
```

[170]:
```python
df.isna().sum()
```

[170]:
```
AAPL      0
ACA.PA    0
AMGN      0
^GSPC     0
dtype: int64
```

No more missing values.

Next, we calcultate the return and risk of each individual stock.

[171]:
```
df.head()
```

[171]:

Date	AAPL	ACA.PA	AMGN	^GSPC
2017-01-03	27.257641	8.576064	128.340637	2257.830078
2017-01-04	27.227137	8.614586	130.162735	2270.750000
2017-01-05	27.365597	8.635597	130.256424	2269.000000
2017-01-06	27.670677	8.639098	133.491943	2276.979980
2017-01-09	27.924126	8.492020	135.245972	2268.899902

[172]:
```
year_r =Return(df[["AAPL","ACA.PA","AMGN"]], 'simple', True)
```

```
Annual average return :

 AAPL      0.359296
ACA.PA    0.087865
AMGN      0.158122
dtype: float64
```

As we mentioned early, we imported some functions that we created in a different file. Return is not a python built-in function but a customized one that we built. You can consult them at the end in annexe.

Before we proceed further, we would like to mention that we are caclulating simple returns not logarithmic returns. Both methods yield similar results but the log return is mainly used when dealing with a single stock. Since, we are dealing with several stocks and we intend to compare them, we use the simple return method.

```
simple return = Price(i)/Price(i-1) - 1, where i is a day
log return  = ln(Price(i)/Price(i-1))
```

[173]:
```
day_r =Return(df[["AAPL","ACA.PA","AMGN"]], 'simple')
```

[174]:
```
year_std = Risk(day_r,'std')
```

```
Anuual volatily

 AAPL      0.307447
ACA.PA    0.327476
```

```
AMGN      0.248428
dtype: float64
```

[175]:
```
r_corr = Risk(day_r,'corr')
```

```
Correlation


               AAPL      ACA.PA        AMGN
AAPL    250.000000   59.704093  111.842460
ACA.PA   59.704093  250.000000   50.887317
AMGN    111.842460   50.887317  250.000000
```

Next, we generate a random weight combination using the random method in Python.

[176]:
```
w = np.array([0.333, 0.333, 0.333])
w/= np.sum(w) # w = w/sum(w)
```

Expected Portfolio Return

[181]:
```
ptf_r = round(np.dot(w,year_r),5)
#ptf_r = round(np.sum(w*year_r),5)
print('Portfolio Expected return is', ptf_r)
```

```
Portfolio Expected return is 0.20176
```

Expected Portfolio Variance

Recall:

Var(ptf) = var(a) • w(a)^2 + var(b) • w(b)^2 - 2w(a)w(b)cov(a,b),
for 2 stock a and b, where Cov(a,b) = std(a) • std(b) • corr(a,b)

In terms of matrix multiplicatio, we have

```
= [Wa Wb][Var(a)     Cov(a,b)]  [Wa]
         [Cov(b,a).    Var(b)]  [Wb]

= Wa^2Var(a) + WaWbCov(a,b) + WaWbCov(a,b) + Wb^2Varb
```

We will do the dot product of the transpose weight matrix • covariance matrix • weight matrix

[179]:
```
ptf_var = round(np.dot(w.T, np.dot(day_r.cov()*250, w)),6)
print('Expect Portfolio Variance is', ptf_var)
```

```
Expect Portfolio Variance is 0.045892
```

[180]:
```
ptf_vol = round(sqrt(ptf_var),4)
print('Expected Portfolio Volatility is', ptf_vol)
```

```
Expected Portfolio Volatility is 0.2142
```

0.2 Part 2: Portfolio Optimization

In this part, we consider 10,000 weights combinations of the same portfolio we analyzed in part 1. Using Python numpy module, we generate 10,000 weight combinations that we apply to the portfolio.

[134]:
```
ptf_genreturns = [] # we will store the generated random returns here
ptf_genstds = [] # we will store the generated random standard deviations here
ptf_weight = [] # We will store the generated weights here
df2 = df[["AAPL","ACA.PA","AMGN"]]
```

[135]:
```
for x in range(10000):
    weight = np.random.random(3)
    weight/= sum(weight)
    ptf_genreturns.append(np.dot(weight,year_r))
    ptf_genstds.append(sqrt(np.dot(weight.T, np.dot(day_r.cov()*250, weight))))
    ptf_weight.append(weight)
```

[136]:
```
ptf_returns = np.array(ptf_genreturns)
ptf_stds = np.array(ptf_genstds)
```

[137]:
```
ptf_sets = pd.DataFrame({'Return':ptf_returns, 'Volatility': ptf_stds})
```

[138]:
```
for counter, symbol in enumerate(df2.columns.tolist()):
    #print(counter, symbol)
    ptf_sets[symbol+' weight'] = [w[counter] for w in ptf_weight]
```

We have successfully generated 10000 combinations of the our portfolio. We can see the first 5 entries and check if there is any missing values below.

[139]:
```
ptf_sets.head()
```

[139]:

	Return	Volatility	AAPL weight	ACA.PA weight	AMGN weight
0	0.144179	0.215320	0.078409	0.422980	0.498612
1	0.175533	0.224322	0.256439	0.486464	0.257097
2	0.192579	0.214533	0.220180	0.140028	0.639792
3	0.157130	0.218497	0.157564	0.465296	0.377139
4	0.297143	0.255193	0.713371	0.063922	0.222706

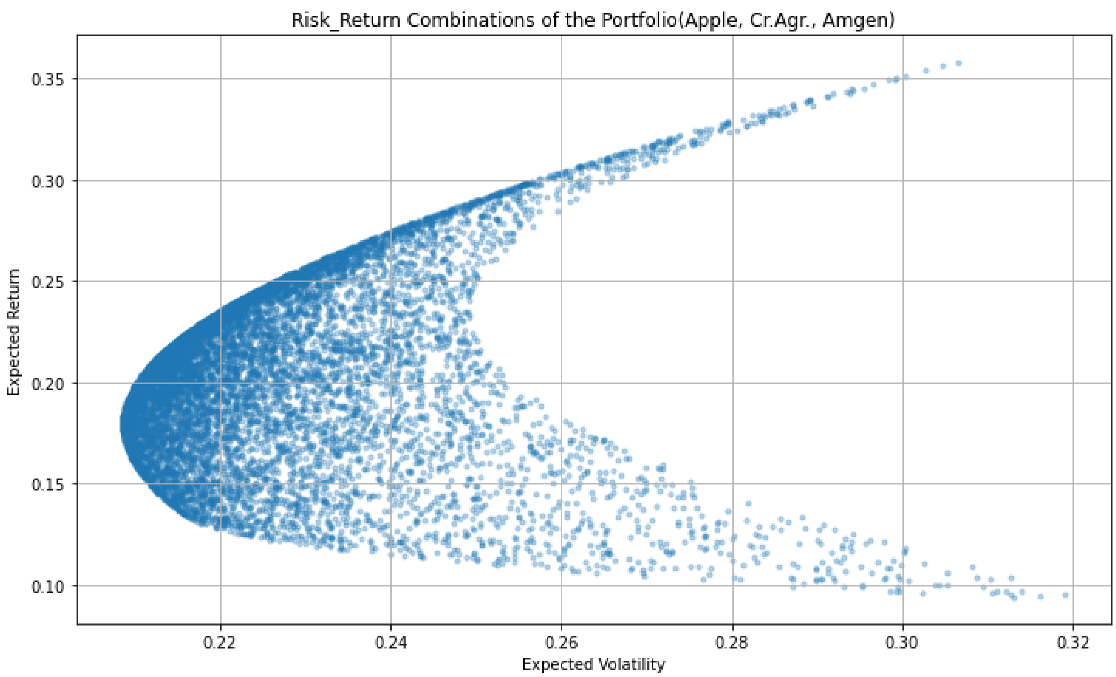
[140]:
```
ptf_sets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Return         10000 non-null  float64
 1   Volatility     10000 non-null  float64
 2   AAPL weight    10000 non-null  float64
```

```
 3   ACA.PA weight  10000 non-null  float64
 4   AMGN weight    10000 non-null  float64
dtypes: float64(5)
memory usage: 390.8 KB
```

As we can see, we have 10,000 observations for each of the variables. We create a graph to visualize them.

[141]:
```
ptf_sets.plot.scatter(x='Volatility', y='Return', marker='o', s=10, alpha=0.3,
                      grid=True, figsize=(12,7),
                      title = "Risk_Return Combinations of the Portfolio(Apple,␣
 ↪Cr.Agr., Amgen)",
                      xlabel = "Expected Volatility", ylabel = "Expected␣
 ↪Return")
plt.show()
plt.show()
```



Notice that our scatter plot appears to be a C-shape curve with all the plots inside the curve. The edge of this curve is called the efficient frontier; that is, every stock that lies on that edge is efficient. By effecient we mean, have the highest return for a given risk level.

CAPM expected return = rf + beta(rm-rf),

`where rf is risk free asset, and rm is the market return.`

According to the capm:

(1) Investors will invest some capitals in the risk free asset and some in the market portfolio.

How much one choose to invest ina risk free asset and in the market ptf depends on his risk aversion.

(2) Market portfolio is a portfolio made up of all assets in the economy (unrealistic)

(3)The line that combine the rf asset and market ptf is called the capital market line

(4) The CML is tangent to the market ptf. It touches the market ptf on the efficient frontier

(5) beta = cov(asset, market) /variance (market): determines how sensitive a stock return is to the change in the market.

(6) risk premium = r market return - risk free return

Now, we proceed by calculating the beta of each securities in the portfolio. First, we calculate the covariance between the market index and each stock. For the market index, we will use the S&P500 index since most of the stocks are US based, except Credit Agricole. Second, we calculte the variance of the market portfolio,and last we compute the beta coefficients.

[182]:
```
cov_matrix = Risk(Return(df,'simple'),'cov')
cov_array = cov_matrix[3:]

print('\n'*2,"We extract the last row of the covariance matrix above:␣
↪",'\n'*2,cov_array)
```

```
Annual Covariance

             AAPL    ACA.PA      AMGN     ^GSPC
AAPL    0.094524  0.024044  0.034169  0.046726
ACA.PA  0.024044  0.107240  0.016560  0.028726
AMGN    0.034169  0.016560  0.061717  0.028866
^GSPC   0.046726  0.028726  0.028866  0.038362


 We extract the last row of the covariance matrix above:

            AAPL    ACA.PA      AMGN     ^GSPC
^GSPC  0.046726  0.028726  0.028866  0.038362
```

[183]:
```
market_var = float(Return(df[['^GSPC']],'simple').var()*250) # verify that we␣
↪get the correct variance for SP500
print('\n', 'Thus, the Market variance is',round(market_var,6))
```

```
 Thus, the Market variance is 0.038362
```

Therefore, the beta coefficients of each stock are as follows:

[184]:
```
beta = cov_array/market_var
print(beta)
```

```
           AAPL    ACA.PA      AMGN  ^GSPC
^GSPC  1.218013  0.748821  0.752469    1.0
```

We should check that our calculations are correct. First, notice that our market variance is equal to what the covariance matrix yielded. The entry in Column GSPC and row GSPC is the variance of the market portfolio -GSPC is simply the market acronym of the S&P 500). Another thing to check is that the beta of the S&P500 to itself should equal 1. We can also double check by comparing them to the beta coefficients on yahoo or google finance.

To calculate the CAPM expected return,

(1) we assume a risk free return is 2%. Usually, we take the rate of return of the US 10-years treasury bill as an estimate for the risk free rate of return.

(2) For the risk premium, we use the commonly used proxy of 5.5% for US based firms.

[185]:
```
risk_free = .02
risk_prem = np.array([0.055,0.055,0.055]) # created this array in order to do a␣
↪dot product
```

Expected return according to CAPM

[146]:
```
capm_return = float(risk_free + np.dot(beta[["AAPL","ACA.PA","AMGN"]],␣
↪risk_prem))
print("The CAPM expected return is {}".format(round(capm_return,5)))
```

```
The CAPM expected return is 0.16956
```

[186]:
```
capm_vol = sqrt(np.dot(weight.T,np.dot(day_r.cov()*250,weight)))
w= []
for x in weight:
    w.append(str(round(x,3)))
w = ','.join(w)
print("The CAPM expected volatility for a {0} combination of Apple, Credit␣
↪Agricole and Amgen respectively is {1}".format(w,round(capm_vol,5)))
```

```
The CAPM expected volatility for a 0.366,0.378,0.256 combination of Apple,
Credit Agricole and Amgen respectively is 0.21974
```

Sharpe ratio = (return(i) - riskfree)/std(i)

Next, we calculate the sharpe ratio for each of the 10,000 combinations of our portfolio.

[187]:
```
sharpe_set = [] # alist where the generated sharpe ratio will be stored
```

[188]:
```
for x in range(len(ptf_genreturns)):
    sharpe = (ptf_genreturns[x] - risk_free)/ptf_genstds[x]
    sharpe_set.append(sharpe)
```

[189]:
```
sharpe_frame = pd.DataFrame({"Sharpe ratio": np.array(sharpe_set)})
```

[190]:
```
ptf_sets["Sharpe"] = sharpe_frame
```

The sharpe ratios are added to the table as follows:

[191]:
```
ptf_sets.head()
```

[191]:

	Return	Volatility	AAPL weight	ACA.PA weight	AMGN weight	Sharpe
0	0.144179	0.215320	0.078409	0.422980	0.498612	0.576716
1	0.175533	0.224322	0.256439	0.486464	0.257097	0.693350
2	0.192579	0.214533	0.220180	0.140028	0.639792	0.804438
3	0.157130	0.218497	0.157564	0.465296	0.377139	0.627604
4	0.297143	0.255193	0.713371	0.063922	0.222706	1.086011

Finding the minimum variance portfolio and the optimal portfolio

(1)To find the minimum variance portfolio, it suffices to calculate the min of the 10,000 volatilities.

[153]:
```
min_vol_port = ptf_sets.iloc[ptf_sets['Volatility'].idxmin()]
```

[154]:
```
print("The minimum variance portfolio is as follow", '\n'*2,min_vol_port)
```

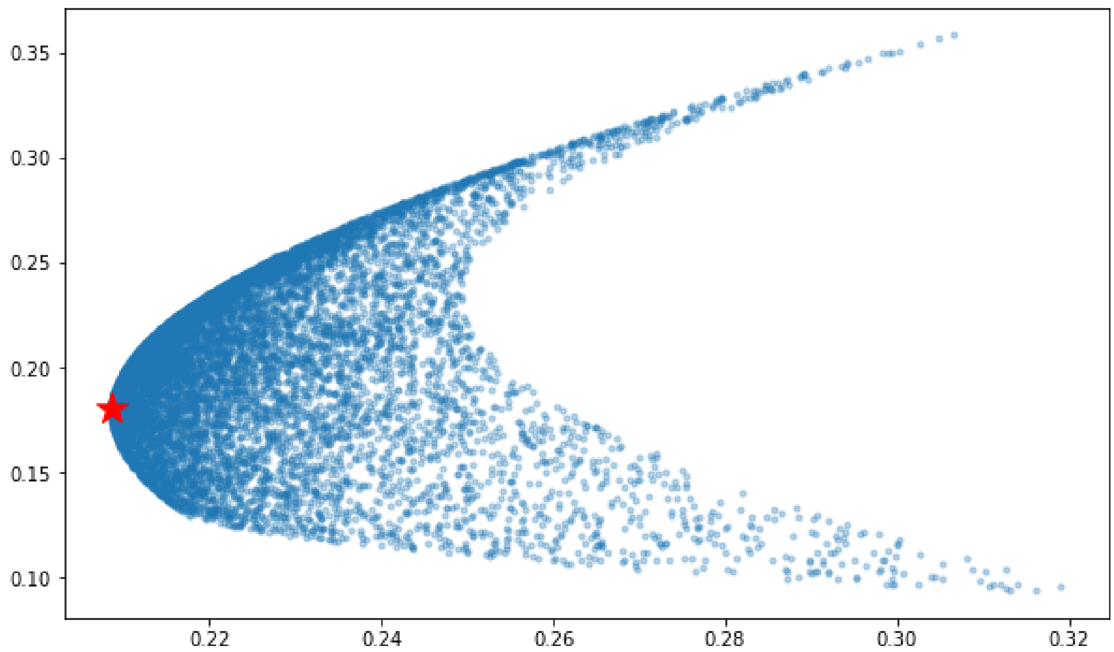
```
The minimum variance portfolio is as follow

 Return           0.180197
Volatility       0.208570
AAPL weight      0.207421
ACA.PA weight    0.279731
AMGN weight      0.512848
Sharpe           0.768071
Name: 4735, dtype: float64
```

Let visualize it in a graph.

[155]:
```
# plotting the minimum volatility portfolio

plt.subplots(figsize=[10,6])
plt.scatter(ptf_sets['Volatility'], ptf_sets['Return'],marker='o', s=10,␣
 ↪alpha=0.3)
plt.scatter(min_vol_port[1], min_vol_port[0], color='r', marker='*', s=300)
plt.show()
```



[156]:
```
w_apl,w_ca, w_amg  = min_vol_port["AAPL weight"], min_vol_port["ACA.PA␣
↪weight"], min_vol_port["AMGN weight"]
w_dic = {"Apple": round(w_apl,4), "Credit Agricole": round(w_ca,4), "Amgen":
↪round(w_amg,4)}
w_dic1 = list(w_dic.values())
w_dic2 = list(w_dic.keys())
```

The red star in the graph represents the minimum variance portfolio. That is the portfolio that yields the lowest risk regardless of the benefit. For instance, an extremely risk averse investor can hold such a portfolio becasue he is most interested in reducing his risk exposure regardless of how much he might earn as a return.

[157]:
```
print("The minimum portfolio is made of {0} shares of {1}, {2} shares of {3}␣
↪and {4} shares of {5}.".format(w_dic1[0],w_dic2[0],w_dic1[1],w_dic2[1],␣
↪w_dic1[2],w_dic2[2]))
```

```
The minimum portfolio is made of 0.2074 shares of Apple, 0.2797 shares of Credit
Agricole and 0.5128 shares of Amgen.
```

(2) To find the optimal portfolio; that is, the portfolio with the highest sharpe ratio, we simply calculate the max of the 10000 Sharpe ratios in the table.

[192]:
```
optimal_port = ptf_sets.iloc[ptf_sets['Sharpe'].idxmax()]
```

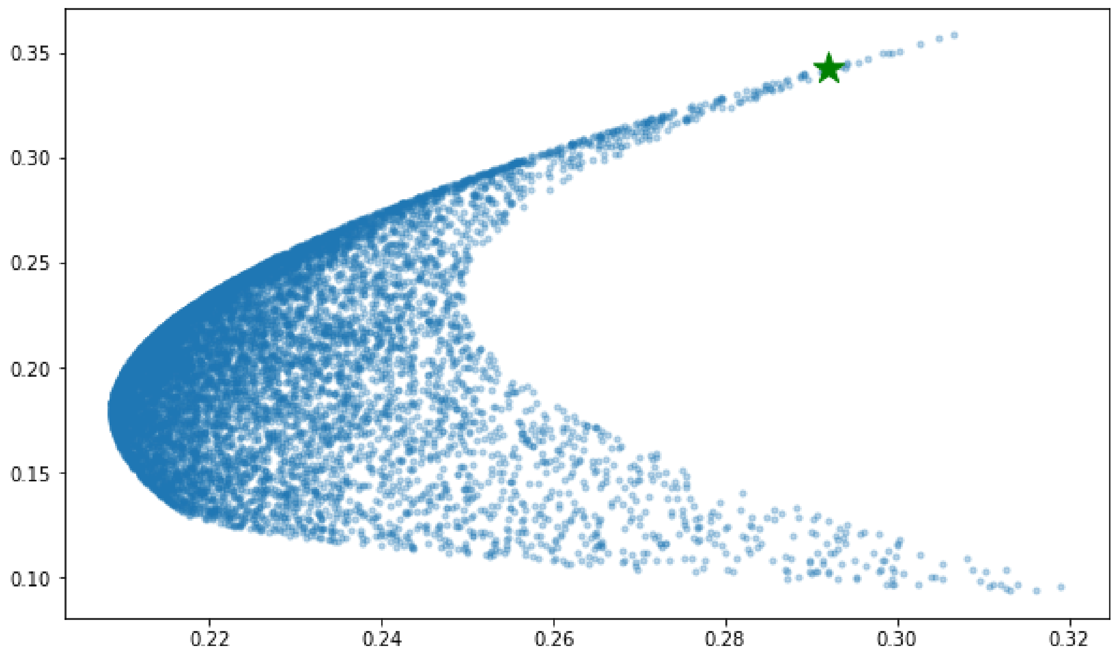
[193]:
```
print("The optimal portfolio is as follow", '\n'*2,optimal_port)
```

```
The optimal portfolio is as follow
```

```
 Return          0.342766
Volatility       0.291911
AAPL weight      0.918239
ACA.PA weight    0.001165
AMGN weight      0.080596
Sharpe           1.105701
Name: 1235, dtype: float64
```

[194]:
```
# plotting the optimal portfolio

plt.subplots(figsize=[10,6])
plt.scatter(ptf_sets['Volatility'], ptf_sets['Return'],marker='o', s=10,␣
 ↪alpha=0.3)
plt.scatter(optimal_port[1], optimal_port[0], color='g', marker='*', s=300)
plt.show()
```



[195]:
```
w_apl,w_ca, w_amg  = optimal_port["AAPL weight"], optimal_port["ACA.PA␣
 ↪weight"], optimal_port["AMGN weight"]
w_dic = {"Apple": round(w_apl,4), "Credit Agricole": round(w_ca,4), "Amgen":
 ↪round(w_amg,4)}
w_dic1 = list(w_dic.values())
w_dic2 = list(w_dic.keys())
```

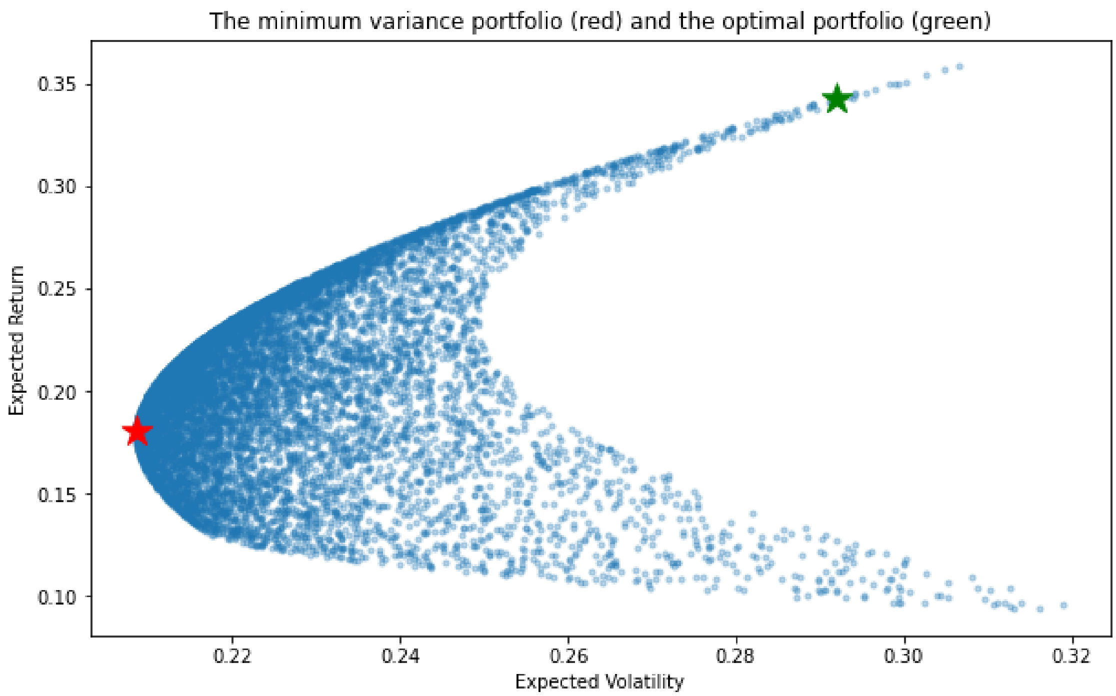
The green green represent the optimal portfolio; that is, the portfolio with the highest risk-return compensation. It yields the highest return relative to the volatility. For instance, an investor who is interested in maximizing its return while minimizing risk will be interested in holding this portfolio.

[196]:
```
print("The optimal portfolio is made of {0} shares of {1}, {2} shares of {3}␣
↪and {4} shares of {5}.".format(w_dic1[0],w_dic2[0],w_dic1[1],w_dic2[1],␣
↪w_dic1[2],w_dic2[2]))
```

```
The optimal portfolio is made of 0.9182 shares of Apple, 0.0012 shares of Credit
Agricole and 0.0806 shares of Amgen.
```

Last, we combine both portfolios in one graph

[197]:
```
plt.subplots(figsize=[10,6])
plt.scatter(ptf_sets['Volatility'], ptf_sets['Return'],marker='o', s=10,␣
↪alpha=0.3)
plt.scatter(optimal_port[1], optimal_port[0], color='g', marker='*', s=300)
plt.scatter(min_vol_port[1], min_vol_port[0], color='r', marker='*', s=300)
plt.title("The minimum variance portfolio (red) and the optimal portfolio␣
↪(green)")
plt.xlabel("Expected Volatility"); plt.ylabel("Expected Return")
plt.savefig("fig1.png")
plt.show()
```



0.3 Part 3 Conclusion

In the first part of our analysis, we considered an equally weighted portfolio pf Apple, Credit Agricole and Amgen, and we found a Portfolio expected return of 20.18% with an expected volatility of 21.42%. In the second part we considered 10,000 ways of allocating our money and performed the Markowitz portfolio optimization. This analysis yielded two results which are the minimum variance portfolio and the optimal portfolio. With these two portfolio, we mentioned that depending on how much risk an investor is willing to take, the choice of the portfolio will differ. A risk averse investor will choose to lower his risk exposure regardless of the return. For such investor, since the primary goal is to reduce risk, the minimum variance portfolio will be a good choice. On the other hand, some investors may be willing to maximize their profits while taking less risk. For this investor, the optimal portfolio will be a good choice.

Annexe

def Return(Col_index, method, Y_average = False): ”””arg: Col_index, method”””

```
#simple return method
if method == 'simple':
    s_return = (Col_index/Col_index.shift(1))-1

#log return method
elif method == 'log':
    s_return = np.log(Col_index/Col_index.shift(1))

Y_avgR = s_return.mean()*250

if Y_average == True:
    print('Annual average return :',"\n"*2, Y_avgR )
    return Y_avgR

else:
    #print("{} returns have been calculated for the variable stated above.".format(method))
    return s_return
```

def Risk(s_return, indicator =‘std’, year = True):

```
vol = s_return.std()
cov = s_return.cov()
corr = s_return.corr()
var = s_return.var()


if indicator == "cov":
    if year == False:
        print("Covariance",'\n'*2, cov)
        return cov
    else:
        print("Annual Covariance",'\n'*2, cov* 250)
        return cov* 250

elif indicator == "corr":
    if year == False:
        print("Correlation", '\n'*2,  corr)
```

```
        return corr
    else:
        print("Correlation", '\n'*2,  corr *250)
        return corr*250

elif indicator == "var":
    if year == False:
        print("Variance", '\n'*2,  var)
        return var
    else:
        print("Annual Variance", '\n'*2,  var*250)
        return var*250

else:
    if year == False:
        print("Volatily", '\n'*2, vol)
        return vol
    else:
        print("Anuual volatily", '\n'*2, vol*sqrt(250))
        return vol*sqrt(250)
```

def Graph(data, title, typ =‘line’): ””” data[index], ‘title’ ””” data.plot(figsize=(12,7), title = title, kind = typ) return (plt.show())

def normalize(data, base_price): ””“normalize prcices”"” result = data/data.iloc[0]* base_price return result

Thanks for reading. For inquiry email me at eliediwa9@gmail.com

For beautiful scatter, visit https://www.machinelearningplus.com/machine-learning/portfolio-optimization-python-example/