Portfolio_Analysis_Python_19_March_2018

March 19, 2018

0.1 Installations Required

- quandl
- pandas
- numpy
- matplotlib

Suggest to get NSE data from QUANDL. Yahoo API is not working with Python to scrape NSE data

Reference link - https://medium.com/python-data/effient-frontier-in-python-34b0c3043314

In [18]:
```
# import needed modules
import quandl
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# get adjusted closing prices of 5 selected companies with Quandl
quandl.ApiConfig.api_key = 'NhXyyCZDmozeVo5owgmG'
# CenterPoint Energy, Ford, Wallmart, General Electrics, Tesla
selected = ['CNP', 'F', 'WMT', 'GE', 'TSLA']
noa = len(selected)
data = quandl.get_table('WIKI/PRICES', ticker = selected,
                        qopts = { 'columns': ['date', 'ticker', 'adj_close'] },
                        date = { 'gte': '2016-1-1', 'lte': '2017-12-31' }, paginate=Tru
```

In [19]:
```
data.head()
```

Out[19]:
```
           date ticker  adj_close
None
0    2016-01-04    CNP  16.749029
1    2016-01-05    CNP  16.904876
2    2016-01-06    CNP  16.694024
3    2016-01-07    CNP  16.363994
4    2016-01-08    CNP  16.327324
```

In [20]:
```
# reorganise data pulled by setting date as index with
# columns of tickers and their corresponding adjusted prices
```

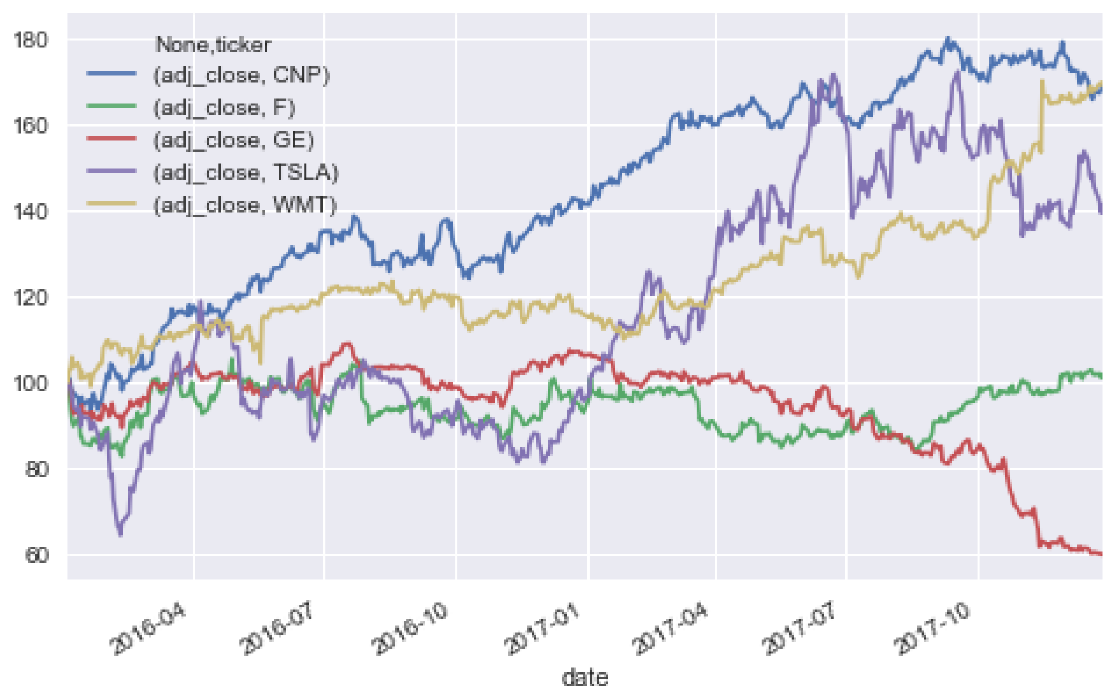
```
clean = data.set_index('date')
table = clean.pivot(columns='ticker')
table.head()
```

Out[20]:

```
              adj_close
ticker              CNP          F         GE    TSLA        WMT
date
2016-01-04    16.749029  12.366951  29.016236  223.41  58.532144
2016-01-05    16.904876  12.145638  29.044581  223.43  59.922592
2016-01-06    16.694024  11.605635  28.581606  219.04  60.522580
2016-01-07    16.363994  11.242682  27.372203  215.65  61.932075
2016-01-08    16.327324  11.101042  26.880883  211.00  60.513056
```

In [36]:

```
#Plot the graph (plot not working)
(table / table.ix[0] * 100).plot(figsize=(8, 5))
plt.show()
```



In [24]:

```
#get the mean and variance
rets = np.log(table / table.shift(1))
print("Mean ", rets.mean() * 250)
rets.cov() * 250 #some consider 252 days in a calender year
```

```
Mean            ticker
adj_close  CNP        0.262794
           F          0.004940
```

```
          GE       -0.253750
          TSLA      0.165623
          WMT       0.260986
dtype: float64
```

Out[24]:

```
                  adj_close
ticker                  CNP         F        GE      TSLA       WMT
          ticker
adj_close CNP      0.029327  0.006033  0.007128  0.010220  0.005346
          F        0.006033  0.049896  0.017384  0.017104  0.005873
          GE       0.007128  0.017384  0.034948  0.009907  0.004297
          TSLA     0.010220  0.017104  0.009907  0.135728  0.007342
          WMT      0.005346  0.005873  0.004297  0.007342  0.034084
```

In [25]:

```
# Calculating the weights: Assume 100% of investors wealth is invested
weights = np.random.random(noa)
weights /= np.sum(weights)
weights
```

Out[25]:

```
array([ 0.05437319,  0.13715337,  0.23715319,  0.19250063,  0.37881962])
```

In [26]:

```
#Expected portfolio returns
np.sum(rets.mean() * weights) * 250
```

Out[26]:

```
0.08553791335993567
```

In [27]:

```
# expected portfolio standard deviation/volatility
np.sqrt(np.dot(weights.T, np.dot(rets.cov() * 252, weights)))
```

Out[27]:

```
0.13843251430512593
```

In [31]:

```
# calculate daily and annual returns of the stocks
returns_daily = table.pct_change()
returns_annual = returns_daily.mean() * 250
print(returns_daily.head())
print(returns_annual.head())
```

```
          adj_close
ticker          CNP         F        GE      TSLA       WMT
date
2016-01-04      NaN       NaN       NaN       NaN       NaN
2016-01-05  0.009305 -0.017895  0.000977  0.000090  0.023755
2016-01-06 -0.012473 -0.044461 -0.015940 -0.019648  0.010013
2016-01-07 -0.019769 -0.031274 -0.042314 -0.015477  0.023289
2016-01-08 -0.002241 -0.012598 -0.017950 -0.021563 -0.022913
          ticker
adj_close CNP       0.277565
          F         0.029747
```

```
            GE       -0.236278
            TSLA      0.233225
            WMT       0.278283
dtype: float64
```

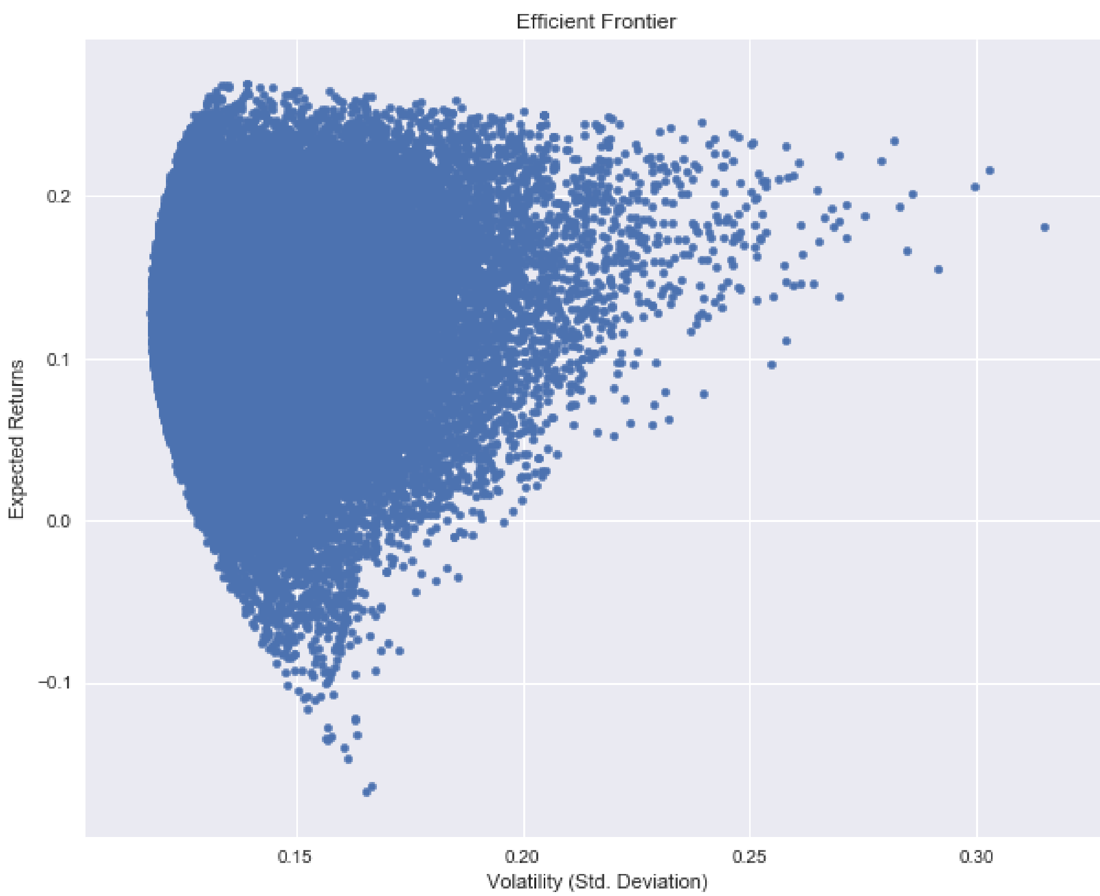
In [33]:
```python
# get daily and covariance of returns of the stock
cov_daily = returns_daily.cov()
cov_annual = cov_daily * 250
```

- Of paramount interest to investors is what risk-return profiles are possible for a given set of securities, and their statistical characteristics.
- To this end, we implement a Monte Carlo simulation to generate random portfolio weight vectors on a larger scale.

In [34]:
```python
# empty lists to store returns, volatility and weights of imiginary portfolios
port_returns = []
port_volatility = []
stock_weights = []


# set the number of combinations for imaginary portfolios
num_assets = len(selected)
num_portfolios = 50000

# populate the empty lists with each portfolios returns,risk and weights
for single_portfolio in range(num_portfolios):
    weights = np.random.random(num_assets)
    weights /= np.sum(weights)
    returns = np.dot(weights, returns_annual)
    volatility = np.sqrt(np.dot(weights.T, np.dot(cov_annual, weights)))
    port_returns.append(returns)
    port_volatility.append(volatility)
    stock_weights.append(weights)

# a dictionary for Returns and Risk values of each portfolio
portfolio = {'Returns': port_returns,
             'Volatility': port_volatility}

# extend original dictionary to accomodate each ticker and weight in the portfolio
for counter,symbol in enumerate(selected):
    portfolio[symbol+' Weight'] = [Weight[counter] for Weight in stock_weights]

# make a nice dataframe of the extended dictionary
df = pd.DataFrame(portfolio)

# get better labels for desired arrangement of columns
column_order = ['Returns', 'Volatility'] + [stock+' Weight' for stock in selected]
```

```
    # reorder dataframe columns
    df = df[column_order]
```

```
In [37]: # plot the efficient frontier with a scatter plot
         plt.style.use('seaborn')
         df.plot.scatter(x='Volatility', y='Returns', figsize=(10, 8), grid=True)
         plt.xlabel('Volatility (Std. Deviation)')
         plt.ylabel('Expected Returns')
         plt.title('Efficient Frontier')
         plt.show()
```



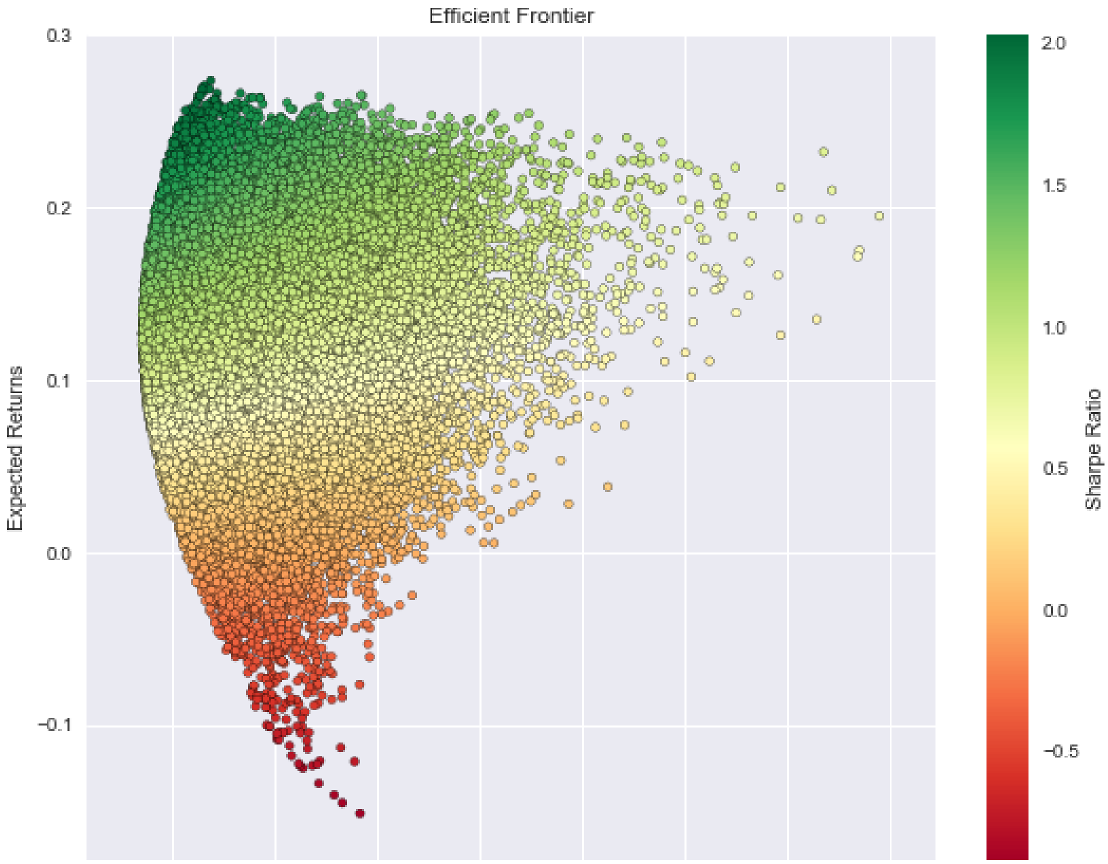
- Sharpe ratio is a measure of the performance of an investment's returns given its risk.
- This ratio adjusts the returns of an investment which makes it possible to compare different investments on a scale that incorporates risk.
- Without this scale of comparison, it would be virtually impossible to compare different investments with different combinations and their accompanying risks and returns
- One intuition of this calculation is that a portfolio engaging in "zero risk" investment, such as the purchase of U.S. Treasury bills (for which the expected return is the risk-free rate), has

a Sharpe ratio of exactly zero. Generally, the greater the value of the Sharpe ratio, the more attractive the risk-adjusted return.

Reference: https://www.investopedia.com/terms/s/sharperatio.asp#ixzz5A9hFIAaH

In [39]:
```
#set random seed for reproduction's sake
np.random.seed(101)

# empty lists to store returns, volatility and weights of imiginary portfolios
port_returns = []
port_volatility = []
sharpe_ratio = []
stock_weights = []

# populate the empty lists with each portfolios returns,risk and weights
for single_portfolio in range(num_portfolios):
    weights = np.random.random(num_assets)
    weights /= np.sum(weights)
    returns = np.dot(weights, returns_annual)
    volatility = np.sqrt(np.dot(weights.T, np.dot(cov_annual, weights)))
    sharpe = returns / volatility
    sharpe_ratio.append(sharpe)
    port_returns.append(returns)
    port_volatility.append(volatility)
    stock_weights.append(weights)

# a dictionary for Returns and Risk values of each portfolio
portfolio = {'Returns': port_returns,
             'Volatility': port_volatility,
             'Sharpe Ratio': sharpe_ratio}

# extend original dictionary to accomodate each ticker and weight in the portfolio
for counter,symbol in enumerate(selected):
    portfolio[symbol+' Weight'] = [Weight[counter] for Weight in stock_weights]

# make a nice dataframe of the extended dictionary
df = pd.DataFrame(portfolio)

# get better labels for desired arrangement of columns
column_order = ['Returns', 'Volatility', 'Sharpe Ratio'] + [stock+' Weight' for stock i

# reorder dataframe columns
df = df[column_order]

# plot frontier, max sharpe & min Volatility values with a scatterplot
plt.style.use('seaborn-dark')
df.plot.scatter(x='Volatility', y='Returns', c='Sharpe Ratio',
                cmap='RdYlGn', edgecolors='black', figsize=(10, 8), grid=True)
```

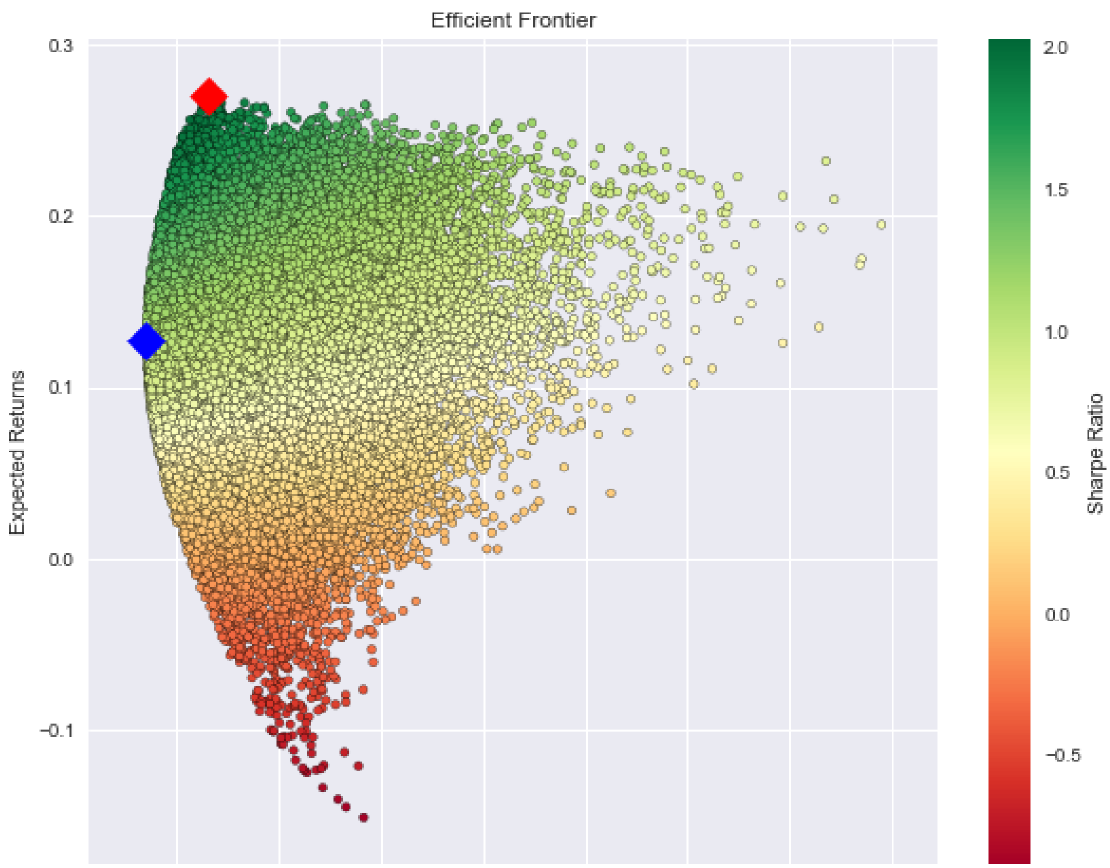
```
plt.xlabel('Volatility (Std. Deviation)')
plt.ylabel('Expected Returns')
plt.title('Efficient Frontier')
plt.show()
```



Locate the * optimal portfolio * portfolio with the minimum volatility for the most risk-averse investor

```
In [40]: # find min Volatility & max sharpe values in the dataframe (df)
         min_volatility = df['Volatility'].min()
         max_sharpe = df['Sharpe Ratio'].max()

         # use the min, max values to locate and create the two special portfolios
         sharpe_portfolio = df.loc[df['Sharpe Ratio'] == max_sharpe]
         min_variance_port = df.loc[df['Volatility'] == min_volatility]

         # plot frontier, max sharpe & min Volatility values with a scatterplot
         plt.style.use('seaborn-dark')
         df.plot.scatter(x='Volatility', y='Returns', c='Sharpe Ratio',
                         cmap='RdYlGn', edgecolors='black', figsize=(10, 8), grid=True)
         plt.scatter(x=sharpe_portfolio['Volatility'], y=sharpe_portfolio['Returns'], c='red', m
```

```
plt.scatter(x=min_variance_port['Volatility'], y=min_variance_port['Returns'], c='blue'
plt.xlabel('Volatility (Std. Deviation)')
plt.ylabel('Expected Returns')
plt.title('Efficient Frontier')
plt.show()
```




```
In [41]: # print the details of the 2 special portfolios
         print(min_variance_port.T)
         print(sharpe_portfolio.T)
```

```
                  5261
Returns       0.126757
Volatility    0.117869
Sharpe Ratio  1.075406
CNP Weight    0.317097
F Weight      0.114479
WMT Weight    0.235926
GE Weight     0.032126
TSLA Weight   0.300371
                  6124
```

```
Returns         0.270487
Volatility      0.133126
Sharpe Ratio    2.031818
CNP Weight      0.453171
F Weight        0.011244
WMT Weight      0.002400
GE Weight       0.076376
TSLA Weight     0.456810
```

0.2 Option pricing using Monte Carlo simulation

- A call option gives the holder of the option the right to buy at a known price. A call makes money if the price of the asset at maturity, denoted by ST , is above the strike price K , otherwise it's worth nothing.

$$C_T = max(0, S_T - K)$$

- Similarly, a put option is the right to sell an asset. A put makes money when the asset is below the strike price at maturity, otherwise it's worth nothing

$$P_T = max(0, K - S_T)$$

- To calculating asset prices at time TT :

$$S_T = S_t e^{(r - \frac{1}{2}\sigma^2)(T-t) + \sigma\sqrt{T-t}\epsilon}$$

- r is our risk free interest rate to discount by.
- σ is volatility, the annualised standard deviation of a stock's returns.
- (T-t) gives us the annualised time to maturity. E.g. for a 30 day option this would be 30/365=0.082...30/365=0.082...
- S at time tt . The price of the underlying asset.
- ϵ is our random value. Its distribution must be standard normal (mean of 0.0 and standard deviation of 1.0)

Reference: http://www.codeandfinance.com/pricing-options-monte-carlo.html

In [46]:
```
#European Style Option

import datetime
import random as rd
import math as ma

def generate_asset_price(S,v,r,T):
    return S * exp((r - 0.5 * v**2) * T + v * ma.sqrt(T) * rd.gauss(0,1.0))

def call_payoff(S_T,K):
    return max(0.0,S_T-K)
```

```
S = 857.29 # underlying price
v = 0.2076 # vol of 20.76%  is volatility, the annualised standard deviation of a stock
r = 0.0014 # rate of 0.14%  r is our risk free interest rate to discount by.
T = (datetime.date(2013,9,21) - datetime.date(2013,9,3)).days / 365.0  #gives us the an
# E.g. for a 30 day option this would be 30/365=0.082...

K = 860.
simulations = 90000
payoffs = []
discount_factor = ma.exp(-r * T)

for i in range(simulations):
    S_T = generate_asset_price(S,v,r,T)
    payoffs.append(
        call_payoff(S_T, K)
    )

price = discount_factor * (sum(payoffs) / float(simulations))
print("Price: %.4f", price)
```

```
Price: %.4f 14.57748459797521
```

In [ ]: