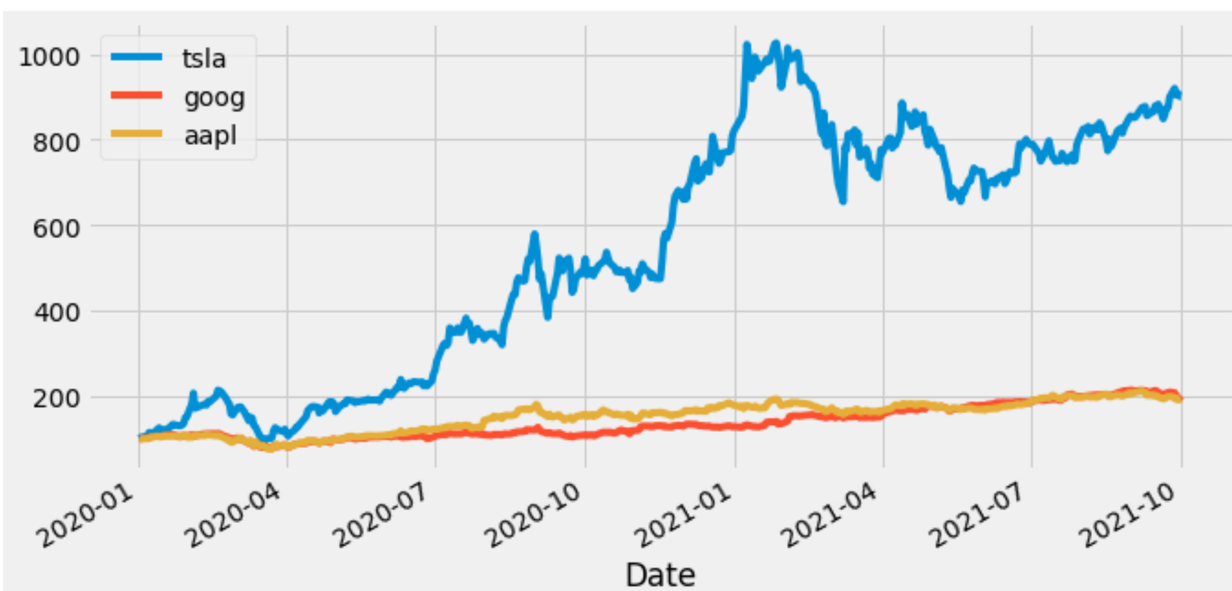
In [1]:
```
import numpy as np
import pandas as pd
import yfinance as yf
from datetime import datetime
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
from pypfopt.efficient_frontier import EfficientFrontier as ef
from pypfopt import risk_models
from pypfopt import expected_returns
from pypfopt import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns
```

In [2]:
```
#1 get data
ticker = ['tsla', 'goog', 'aapl']
df = pd.DataFrame()
for t in ticker:
    df[t] = yf.download(t, start="2020-01-01", end="2021-10-02")['Adj
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

In [3]:
```
(df / df.iloc[0] * 100).plot(figsize=(10,5))
```

Out[3]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9f8842220>
```



In [4]:
```
logreturns = np.log(df / df.shift(1))
```

In [5]:
```
logmean = logreturns.mean()
logmeanyr = logreturns.mean() * 250
```

In [6]:
```python
logreturns.cov() * 250
```

Out[6]:

	tsla	goog	aapl
tsla	0.573044	0.109173	0.145100
goog	0.109173	0.107732	0.091722
aapl	0.145100	0.091722	0.151105

In [7]:
```python
logreturns.corr()
```

Out[7]:

	tsla	goog	aapl
tsla	1.000000	0.439387	0.49310
goog	0.439387	1.000000	0.71889
aapl	0.493100	0.718890	1.00000

In [8]:
```python
# generate weights
arr = np.random.random(10)
num_ticker = len(ticker)
w = np.random.random(num_ticker)
w /= np.sum(w)
w
```

Out[8]:
```
array([0.28445229, 0.27294889, 0.44259882])
```

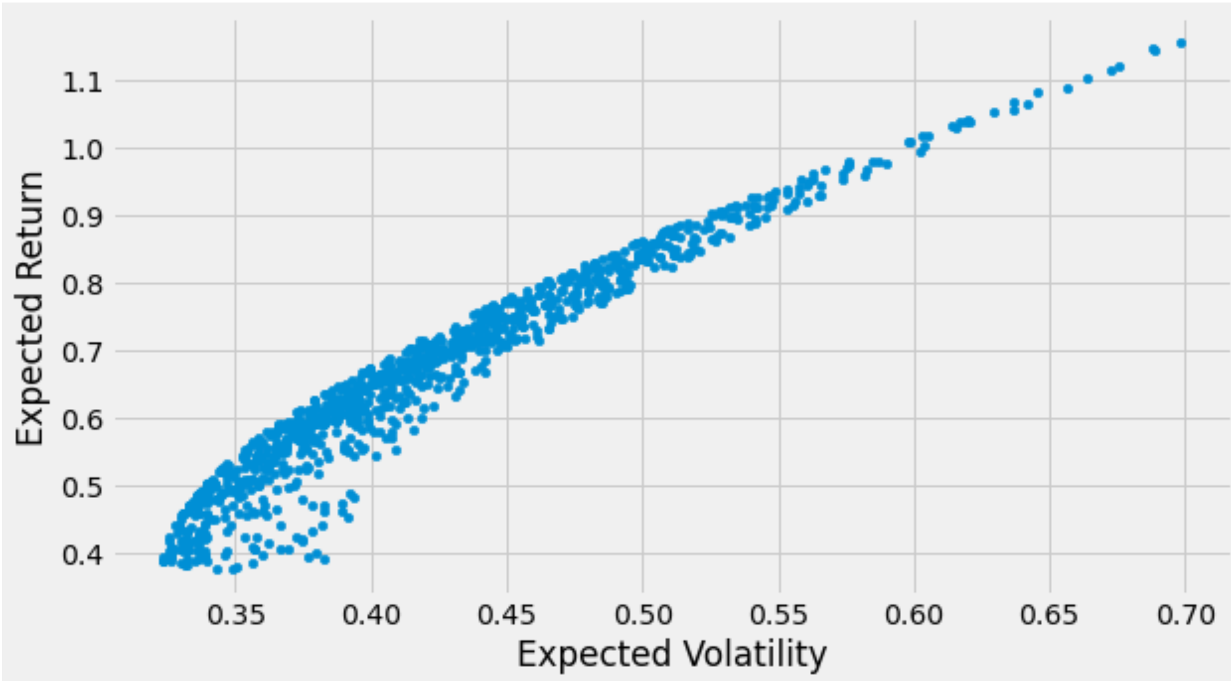
In [9]:
```python
#portfolio mean returns

mu = np.sum(w * logreturns.mean()) * 250

#portfolio volatility

std = np.dot(w.T, np.dot(logreturns.cov() * 250, w)) ** 0.5
```

In [10]:
```python
#monte carlo

pfolio_returns = []
pfolio_volatilities = []
sharpe_ratio = []

for x in range (1000):
    w = np.random.random(num_ticker)
    w /= np.sum(w)

    pfolio_returns.append(np.sum(w *logreturns.mean()) * 250)
    pfolio_volatilities.append(np.sqrt(np.dot(w.T, np.dot(logreturns.c

pfolio_returns = np.array(pfolio_returns)
pfolio_volatilities = np.array(pfolio_volatilities)


pfolio_returns,
pfolio_volatilities

portfolio = pd.DataFrame ({'Return': pfolio_returns, 'Volatility': pfc
sharpe_ratio = pfolio_returns / pfolio_volatilities
sharpe_ratio
```

```
       1.70753842, 1.65348299, 1.2029458 , 1.64473564, 1.58264621,
       1.52569879, 1.43373728, 1.58014702, 1.4210092 , 1.68258427,
       1.72186389, 1.70426931, 1.65754323, 1.63285165, 1.69995421,
       1.56053842, 1.68899949, 1.7191024 , 1.52728591, 1.51550962,
       1.69406222, 1.59814169, 1.42634473, 1.61499463, 1.38032187,
       1.64035118, 1.5139743 , 1.54817861, 1.67657378, 1.53341693,
       1.71722603, 1.35177919, 1.63530832, 1.72422543, 1.70811566,
       1.63771315, 1.42238972, 1.57009574, 1.44663864, 1.64379241,
       1.65330309, 1.61296017, 1.60873669, 1.64893417, 1.30166594,
       1.64205652, 1.46593943, 1.68496783, 1.25017561, 1.30206194,
       1.67071389, 1.41058513, 1.61302881, 1.11430477, 1.59950519,
       1.4174194 , 1.08684031, 1.6856177 , 1.70159898, 1.42208223,
       1.69583241, 1.66619008, 1.23562859, 1.57629426, 1.47340861,
       1.70763239, 1.59338937, 1.66932897, 1.67336025, 1.65709343,
       1.66169356, 1.62011465, 1.68845422, 1.61633301, 1.55888766,
       1.57076409, 1.62301514, 1.68998601, 1.64099753, 1.35428933,
       1.5942375 , 1.70602795, 1.41178134, 1.6157278 , 1.36394291,
       1.4995121 , 1.65641859, 1.61642529, 1.63720623, 1.4087025 ,
       1.45171642, 1.42121459, 1.58953989, 1.64294568, 1.52132555,
       1.26875152, 1.47474423, 1.5933191 , 1.6979904 , 1.65931326,
```

In [11]:
```
portfolio.plot(x='Volatility', y='Return', kind='scatter', figsize=(9,
plt.xlabel('Expected Volatility')
plt.ylabel('Expected Return')
```

Out[11]:
```
Text(0, 0.5, 'Expected Return')
```



In [12]:
```
mu = expected_returns.mean_historical_return(df)
S = risk_models.sample_cov(df)
```

In [13]:
```
# Optimize for maximal Sharpe ratio
ef = EfficientFrontier(mu, S)
raw_weights = ef.max_sharpe()
cleaned_weights = ef.clean_weights()
ef.save_weights_to_file("weights.csv")  # saves to file
ef.portfolio_performance(verbose=True)
print(cleaned_weights)
```

```
Expected annual return: 247.6%
Annual volatility: 75.1%
Sharpe Ratio: 3.27
OrderedDict([('tsla', 0.98235), ('goog', 0.01765), ('aapl', 0.0)])
```