Portfolio Optimization Model

Objective Function: Optimize return on investment for a given level of risk or minimize risk for a given level of return (will be maximizing Sharpe Ratio)

In this portfolio I will only be investing in Apple, Microsoft, Google, and Amazon. I am pulling the data using Yahoo Finance API

In [2]:
```
pip install yfinance numpy pandas scipy matplotlib seaborn PyPortfolioOpt
```

```
Requirement already satisfied: yfinance in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (0.2.40)
Requirement already satisfied: numpy in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (1.26.4)
Requirement already satisfied: pandas in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (2.2.2)
Requirement already satisfied: scipy in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (1.14.0)
Requirement already satisfied: matplotlib in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (3.9.1)
Requirement already satisfied: seaborn in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (0.13.2)
Requirement already satisfied: PyPortfolioOpt in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (1.5.5)
Requirement already satisfied: requests>=2.31 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from yfinance) (5.2.2)
Requirement already satisfied: platformdirs>=2.0.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from yfinance) (4.1.0)
Requirement already satisfied: pytz>=2022.5 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from yfinance) (2024.1)
Requirement already satisfied: frozendict>=2.3.4 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from yfinance) (2.4.4)
Requirement already satisfied: peewee>=3.16.2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from yfinance) (3.17.6)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from yfinance) (4.12.3)
Requirement already satisfied: html5lib>=1.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from yfinance) (1.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from pandas) (2024.1)
Requirement already satisfied: contourpy>=1.0.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from matplotlib) (4.53.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=8 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from matplotlib) (3.1.2)
Requirement already satisfied: cvxpy<2.0.0,>=1.1.19 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from PyPortfolioOpt) (1.5.2)
Requirement already satisfied: soupsieve>1.2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from beautifulsoup4>=4.11.1->yfinance) (2.5)
Requirement already satisfied: osqp>=0.6.2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from cvxpy<2.0.0,>=1.1.19->PyPortfolioOpt) (0.6.7.post0)
Requirement already satisfied: ecos>=2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from cvxpy<2.0.0,>=1.1.19->PyPortfolioOpt) (2.0.14)
Requirement already satisfied: clarabel>=0.5.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from cvxpy<2.0.0,>=1.1.19->PyPortfolioOpt) (0.9.0)
Requirement already satisfied: scs>=3.2.4.post1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from cvxpy<2.0.0,>=1.1.19->PyPortfolioOpt) (3.2.6)
Requirement already satisfied: six>=1.9 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from requests>=2.31->yfinance) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from requests>=2.31->yfinance) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from requests>=2.31->yfinance) (2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from requests>=2.31->yfinance) (2023.11.17)
Requirement already satisfied: qdldl in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from osqp>=0.6.2->cvxpy<2.0.0,>=1.1.19->PyPortfolioOpt) (0.1.7.post4)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```python
import yfinance as yf
import pandas as pd

# Define the tickers
tickers = ['AAPL', 'MSFT', 'GOOGL', 'AMZN']

# Download historical data
data = yf.download(tickers, start='2015-01-01', end='2024-01-01')['Adj Close']

# Display the first few rows
data.head()
```

```
[*********************100%%**********************]  4 of 4 completed
```

Out[3]:

Ticker	AAPL	AMZN	GOOGL	MSFT
Date				
2015-01-02	24.402176	15.4260	26.447147	40.305363
2015-01-05	23.714724	15.1095	25.943224	39.934731
2015-01-06	23.716951	14.7645	25.302961	39.348598
2015-01-07	24.049520	14.9210	25.228544	39.848526
2015-01-08	24.973555	15.0230	25.316446	41.020794

Calculating Returns and Covariance Matrix

- .pct_change(): pandas function that computes the percentage change between the current and prior element.
- .dropna(): removes any rows that contain NaN values.

In [4]:
```python
# Calculate daily returns
returns = data.pct_change().dropna()

# Display the first few rows of returns
returns.head()
```

Out[4]:

Ticker	AAPL	AMZN	GOOGL	MSFT
Date				
2015-01-05	-0.028172	-0.020517	-0.019054	-0.009196
2015-01-06	0.000094	-0.022833	-0.024679	-0.014677
2015-01-07	0.014022	0.010600	-0.002941	0.012705
2015-01-08	0.038422	0.006836	0.003484	0.029418
2015-01-09	0.001073	-0.011749	-0.012211	-0.008405

Covariance Matrix

A square matrix that shows the covariance between each pair of variables in a dataset. For my model, these elements represent the returns of different assets.

- The diagonal elements are the variances of the returns of each asset.
- The off-diagonal elements are the covariances between the returns of different assets.
- Positive covariance means they tend to move in the same direction, while negative covariance means they move in opposite directions.

In [5]:
```python
# Calculate mean returns
mean_returns = returns.mean()

# Calculate the covariance matrix
cov_matrix = returns.cov()

print("Mean Returns:\n", mean_returns)
print("\nCovariance Matrix:\n", cov_matrix)
```

```
Mean Returns:
 Ticker
AAPL     0.001079
AMZN     0.001230
GOOGL    0.000896
MSFT     0.001139
dtype: float64

Covariance Matrix:
 Ticker      AAPL      AMZN     GOOGL      MSFT
Ticker
AAPL    0.000335  0.000218  0.000205  0.000223
AMZN    0.000218  0.000439  0.000246  0.000241
GOOGL   0.000205  0.000246  0.000323  0.000229
MSFT    0.000223  0.000241  0.000229  0.000307
```

The Sharpe Ratio

(Expected Portfolio Return - Risk Free Rate) / Portfolio standard deviation (a measure of risk)

Measures the excess return per unit of risk of an investment asset or a portfolio.

Interpretation

- Positive Sharpe Ratio: Indicates that the investment returns exceed the risk-free rate, after adjusting for risk. Higher values are better.
- Negative Sharpe Ratio: Indicates that the investment returns are less than the risk-free rate, suggesting poor performance.

Risk Free Rate

- For long-term investments, the yield on longer-term government bonds.
- According to Ycharts.com, I chose 4% as the current yield on long-term government bonds.

Negative Sharpe Ratio Function

- Portfolio Return: calculates the expected return of the portfolio as the dot product of asset weights and their mean returns.
- Portfolio Volatility: calculates the portfolio's standard deviation (volatility). This is done by first computing the weighted covariance matrix and then taking the square root.
- Sharpe Ratio: sharpe_ratio = computes the Sharpe ratio, which is the excess return over the risk-free rate per unit of risk.

Constraints and Bounds

- Define constraints to ensure that the portfolio is fully invested, meaning that the sum of the weights of all assets must equal 1.
  - To prevent over-leveraging (allocating more than 100% of the capital) or under-investing (allocating less than 100%).
- fun: lambda x: np.sum(x) - 1 is a function that should return 0 when the constraint is satisfied.
- Each weight in the portfolio should be non-negative and should not exceed 1. This is because:
  1. Non-Negativity: Prevents short selling (assigning negative weights).
  2. Upper Bound: Ensures no single asset takes more than 100% of the capital.

Initial Guess

- Required to start the optimization process.
- The choice of initial guess can affect the convergence speed and the likelihood of finding a global optimum. In the context of portfolio optimization, a common initial guess is to equally distribute the weights among all assets.

In [6]:
```python
import numpy as np

# Define the risk-free rate
risk_free_rate = 0.04

# Objective function to minimize (negative Sharpe ratio)
def neg_sharpe(weights, mean_returns, cov_matrix, risk_free_rate):
    portfolio_return = np.dot(weights, mean_returns)
    portfolio_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    sharpe_ratio = (portfolio_return - risk_free_rate) / portfolio_volatility
    return sharpe_ratio

# Constraints: weights must sum to 1
constraints = {'type': 'eq', 'fun': lambda x: np.sum(x) - 1}

# Bounds: weights must be between 0 and 1
bounds = tuple((0, 1) for _ in range(len(tickers)))

# Initial guess: equally distributed weights
# The list comprehension [1 / len(tickers) for _ in range(len(tickers))] creates a list of equal weights.
# np.array converts this list into a NumPy array for optimization functions.
init_guess = np.array([1 / len(tickers) for _ in range(len(tickers))])
```

Sequential Least Squares Programming (SLSQP)

- Start with an initial guess for the weights $w0$
- At each iteration, solve a quadratic programming subproblem to approximate the nonlinear problem by linearizing the constraints
- Use the solution of the quadratic subproblem to update the weights
- Repeat steps 2 and 3 until convergence (changes in the weights are below a threshold)

Why SLSQP

- Gradient-based optimization method: Uses information about the slope of the objective function to find the optimal solution. This typically leads to faster and more accurate convergence compared to methods that do not use gradient information.
- Smooth Functions: SLSQP is well-suited for optimizing smooth functions such as portfolio optimization.
- Efficiency: SLSQP is generally efficient and can handle problems with a moderate number of variables and constraints relatively quickly.

Note: Simple and great explination of gradient-based optimization by Andrew Ng on Coursera "Supervised Machine Learning: Regression and Classification"

In [7]:
```python
from scipy.optimize import minimize

# Optimize
opt_results = minimize(neg_sharpe, init_guess, args=(mean_returns, cov_matrix, risk_free_rate),
                       method='SLSQP', bounds=bounds, constraints=constraints)

# Extract optimal weights
optimal_weights = opt_results.x


optimal_weights_dict = {tickers[i]: optimal_weights[i] for i in range(len(tickers))}
print("Optimal Weights:")
for ticker, weight in optimal_weights_dict.items():
    print(f"{ticker}: {weight}")

# optimal_return = np.dot(optimal_weights, mean_returns)
# optimal_volatility = np.sqrt(np.dot(optimal_weights.T, np.dot(cov_matrix, optimal_weights)))

# print(optimal_return, optimal_volatility)
```

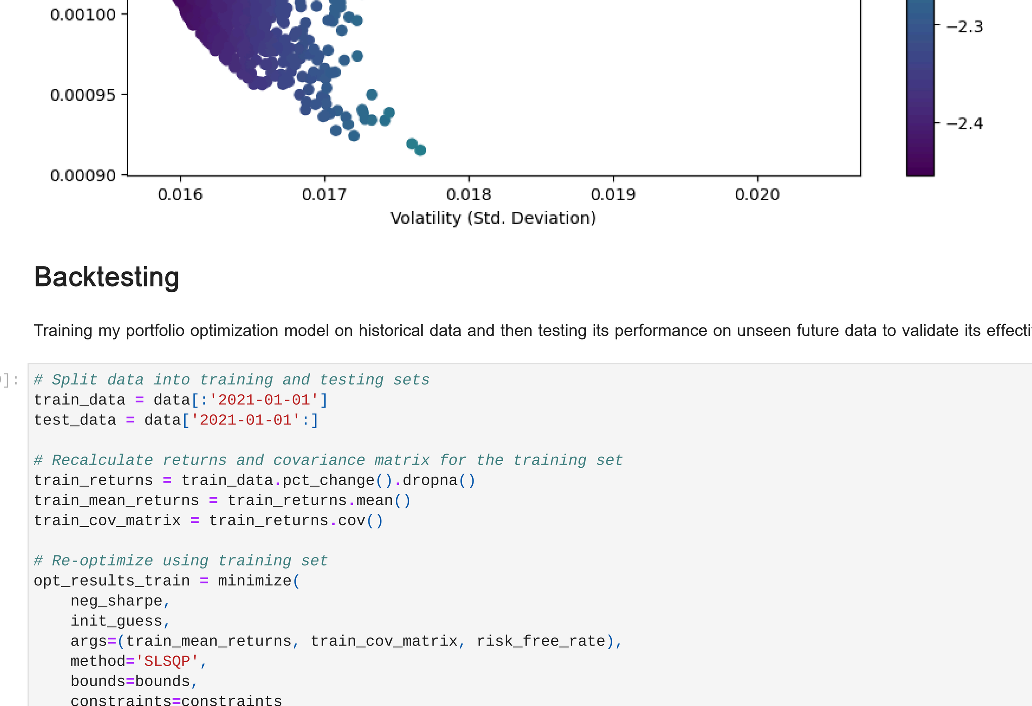
```
Optimal Weights:
AAPL: 0.31033831303470727
MSFT: 0.07518176971710266
GOOGL: 0.31886924904084285
AMZN: 0.2956106682073472
```

Efficient Frontier

Tool for visualizing the optimal portfolios

The optimal portfolio will be plotted where there return is highest for the lowest risk. (Calculated by opt_results.x)

In [8]:
```python
import matplotlib.pyplot as plt

def portfolio_performance(weights, mean_returns, cov_matrix):
    returns = np.dot(weights, mean_returns)
    volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    return returns, volatility

def generate_portfolios(mean_returns, cov_matrix, num_portfolios=20000):
    results = np.zeros((3, num_portfolios))
    weights_record = []
    for i in range(num_portfolios):
        weights = np.random.random(len(tickers))
        weights /= np.sum(weights)
        weights_record.append(weights)
        portfolio_return, portfolio_volatility = portfolio_performance(weights, mean_returns, cov_matrix)
        results[0,i] = portfolio_return
        results[1,i] = portfolio_volatility
        results[2,i] = (portfolio_return - risk_free_rate) / portfolio_volatility
    return results, weights_record

# Generate random portfolios
num_portfolios = 20000
results, weights = generate_portfolios(mean_returns, cov_matrix, num_portfolios)

# Plot the efficient frontier
plt.figure(figsize=(10, 6))
plt.scatter(results[1,:], results[0,:], c=results[2,:], cmap='viridis', marker='o')
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Volatility (Std. Deviation)')
plt.ylabel('Expected Returns')
plt.title('Efficient Frontier')

# Highlight the optimal portfolio
opt_return, opt_volatility = portfolio_performance(optimal_weights, mean_returns, cov_matrix)
plt.scatter(opt_volatility, opt_return, c='red', marker='*', s=100)  # Optimal portfolio
plt.show()
```

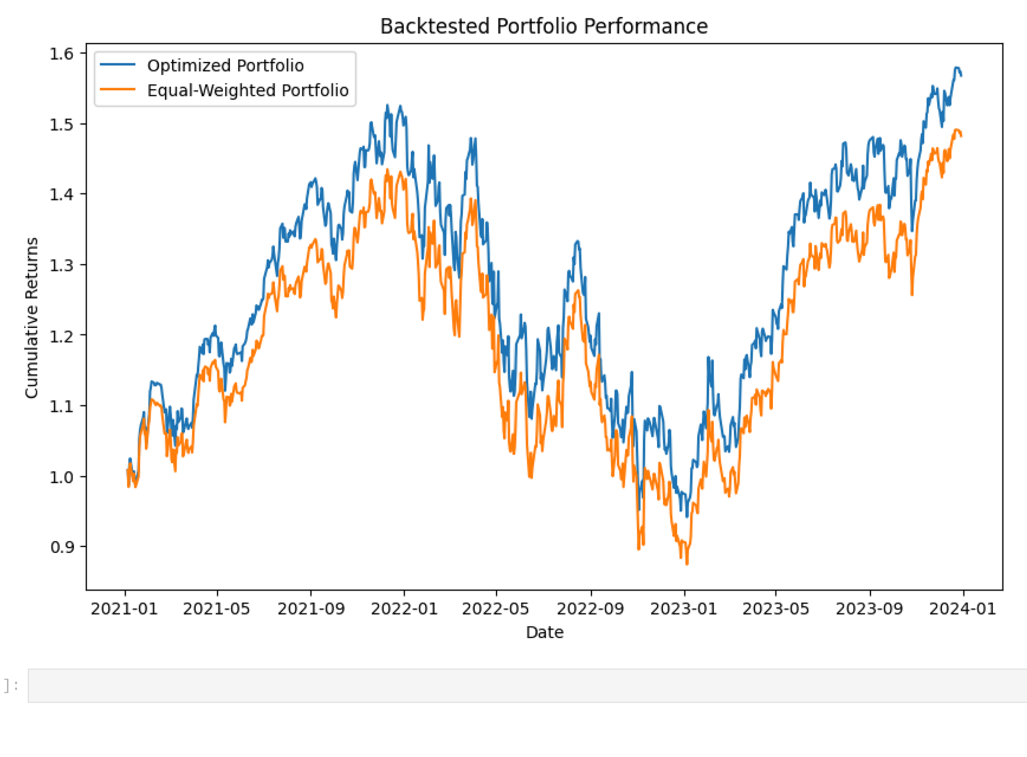


Backtesting

Training my portfolio optimization model on historical data and then testing its performance on unseen future data to validate its effectiveness.

In [9]:
```python
# Split data into training and testing sets
train_data = data[:'2021-01-01']
test_data = data['2021-01-01':]

# Recalculate returns and covariance matrix for the training set
train_returns = train_data.pct_change().dropna()
train_mean_returns = train_returns.mean()
train_cov_matrix = train_returns.cov()

# Re-optimize using training set
opt_results_train = minimize(
    neg_sharpe,
    init_guess,
    args=(train_mean_returns, train_cov_matrix, risk_free_rate),
    method='SLSQP',
    bounds=bounds,
    constraints=constraints
)

optimal_weights_train = opt_results_train.x

optimal_weights_dict_train = {tickers[i]: optimal_weights_train[i] for i in range(len(tickers))}
print("Optimal Weights:")
for ticker, weight in optimal_weights_dict_train.items():
    print(f"{ticker}: {weight}")

# Calculate test set performance
test_returns = test_data.pct_change().dropna()
test_portfolio_returns = (test_returns * optimal_weights_train).sum(axis=1)

# Align the test data index with the returns
aligned_test_index = test_returns.index

# Plot the test set performance
plt.figure(figsize=(10, 6))
plt.plot(aligned_test_index, (1 + test_portfolio_returns).cumprod(), label='Optimized Portfolio')
plt.plot(aligned_test_index, (1 + test_returns.mean(axis=1)).cumprod(), label='Equal-Weighted Portfolio')
plt.legend()
plt.title('Backtested Portfolio Performance')
plt.xlabel('Date')
plt.ylabel('Cumulative Returns')
plt.show()
```

```
Optimal Weights:
AAPL: 0.24131886540987893
MSFT: 0.12726540790151378
GOOGL: 0.42921950704511175
AMZN: 0.20219621964349554
```



In [ ]: