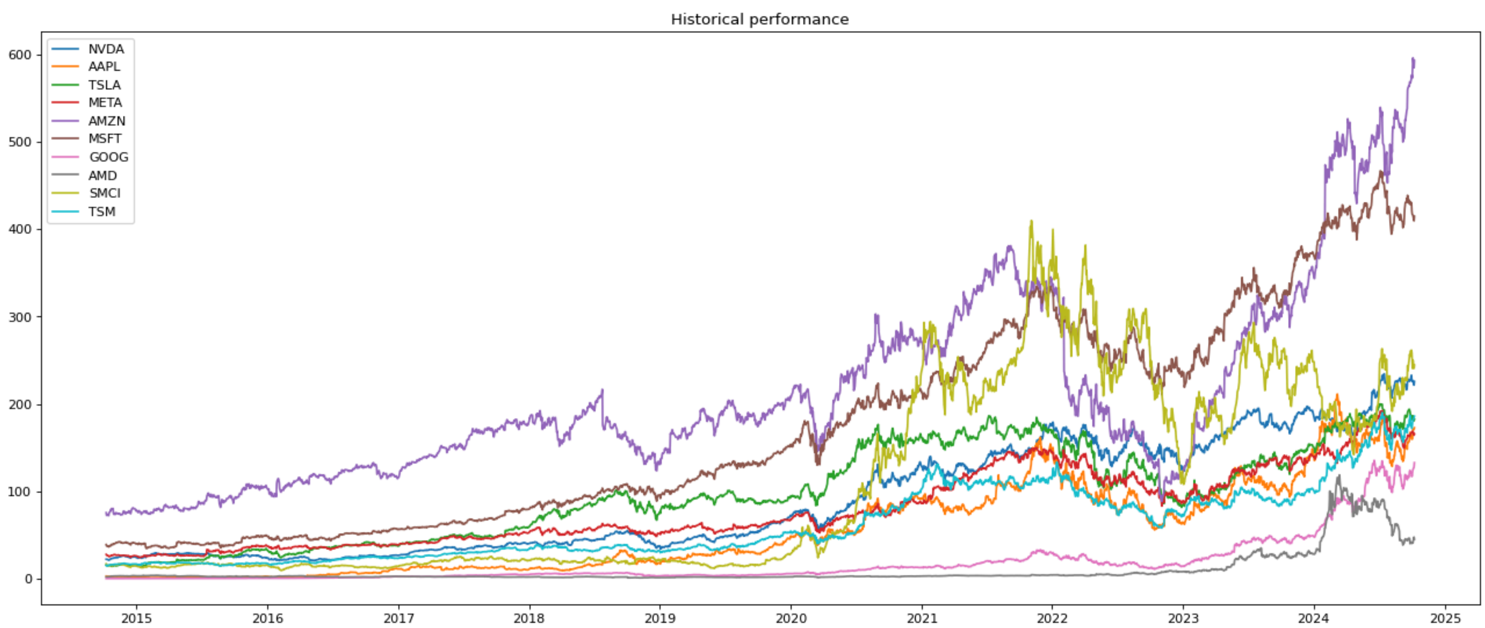
In [24]:
```python
import yfinance as yf
import pandas as pd
import numpy as np
from scipy.optimize import minimize
from matplotlib import pyplot as plt
from matplotlib.ticker import PercentFormatter
import seaborn as sns
```

a). Download stock price data for 10+ equities from free online sources.

In [25]:
```python
# Use yahoo finance to create the stock list
tickers = ['NVDA','AAPL','TSLA','META','AMZN','MSFT','GOOG','AMD','SMCI','TSM']
start_date = '2014-10-9'
end_date = '2024-10-9'

data = yf.download(tickers,start = start_date, end = end_date)['Adj Close']

print(data.head())
```

```
[*********************100%***********************]  10 of 10 completed
Ticker                          AAPL    AMD     AMZN       GOOG       META  \
Date
2014-10-09 00:00:00+00:00  22.424097  2.95  15.7685  27.898703  75.682137
2014-10-10 00:00:00+00:00  22.359726  2.72  15.5695  27.083447  72.691139
2014-10-13 00:00:00+00:00  22.155504  2.74  15.3225  26.522369  72.770897
2014-10-14 00:00:00+00:00  21.920208  2.62  15.4155  26.757645  73.369095
2014-10-15 00:00:00+00:00  21.651617  2.61  15.2985  26.364195  72.990234

Ticker                          MSFT      NVDA   SMCI       TSLA        TSM
Date
2014-10-09 00:00:00+00:00  39.202618  0.428079  2.346  17.134001  15.535735
2014-10-10 00:00:00+00:00  37.646496  0.402743  2.285  15.794000  14.754014
2014-10-13 00:00:00+00:00  37.321583  0.401309  2.310  14.972667  14.928576
2014-10-14 00:00:00+00:00  37.389980  0.410631  2.323  15.137333  15.285284
2014-10-15 00:00:00+00:00  36.953922  0.416845  2.390  15.313333  15.232158
```

In [26]:
```python
# Show the historical performance of selected datas
plt.figure(figsize=(20, 8), dpi = 80)
plt.plot(data)
plt.title('Historical performance')
plt.legend(tickers)
plt.show()
```



b). Use these stocks to compute a mean-variance efficient frontier. While mean-variance optimization is built into a lot of software packages, using a generic optimization package with the correct objective function and constraints is preferred here. How did you compute the expected return for each stock? The covariance matrix? What start/end date did you use for the return series? What frequency are the returns? Visualize the covariance matrix. Report all the expected returns and covariances as annualized quantities.

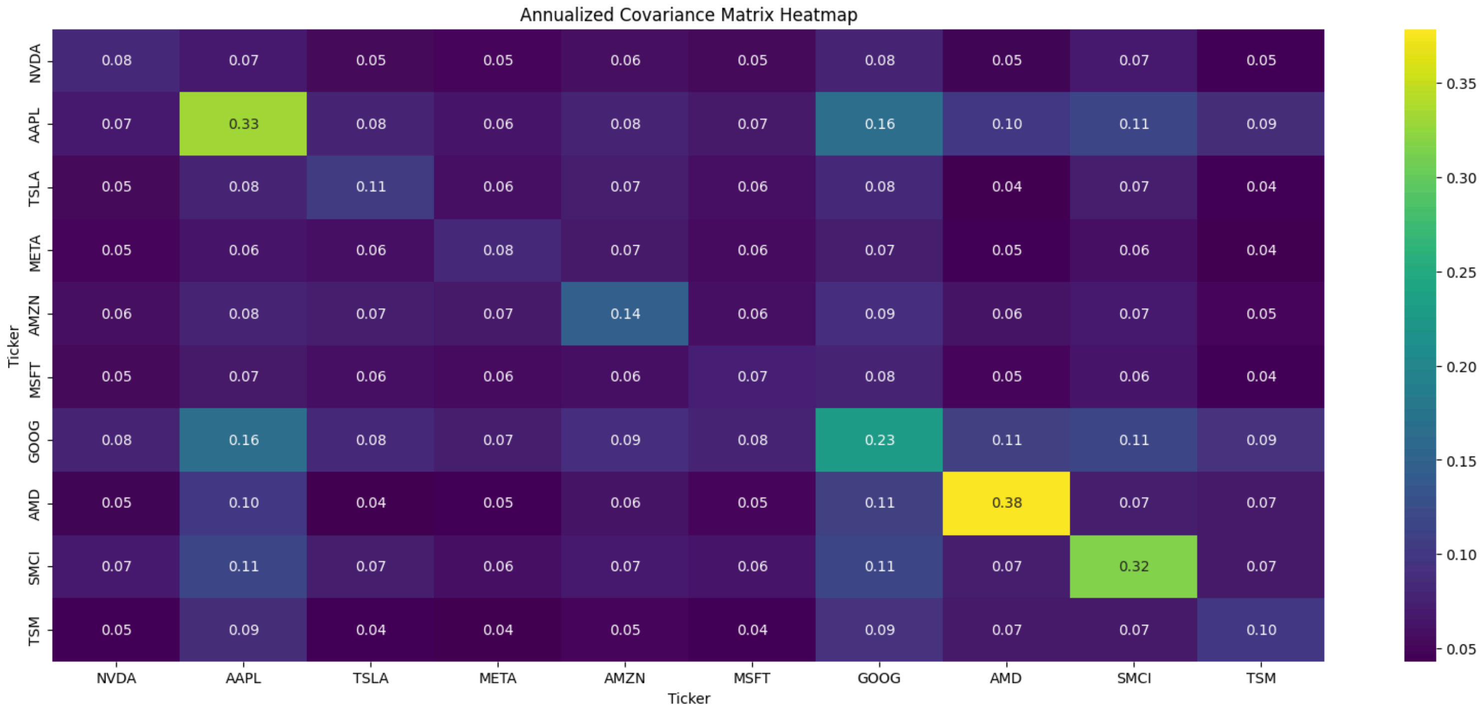
In [27]:
```python
# Daily Log Return
daily_return = np.log(data/data.shift(1)).dropna()

# Daily return mean
mean_daily_return = daily_return.mean()

# Daily covariance Matrix
daily_cov_mat = daily_return.cov()
```

In [28]:
```python
# Assume 252 trading days
trading_days = 252

# Calculate the annual return
annual_return = mean_daily_return * trading_days
```

```
# Annual covariance matrix
annual_cov_mat = daily_cov_mat * trading_days
```

In [120…

```
# Plot the covariance heat map
plt.figure(figsize=(20, 8))
sns.heatmap(annual_cov_mat, annot=True, fmt=".2f", cmap='viridis', xticklabels=tickers, yticklabels=tickers)
plt.title('Annualized Covariance Matrix Heatmap')
plt.show()
```

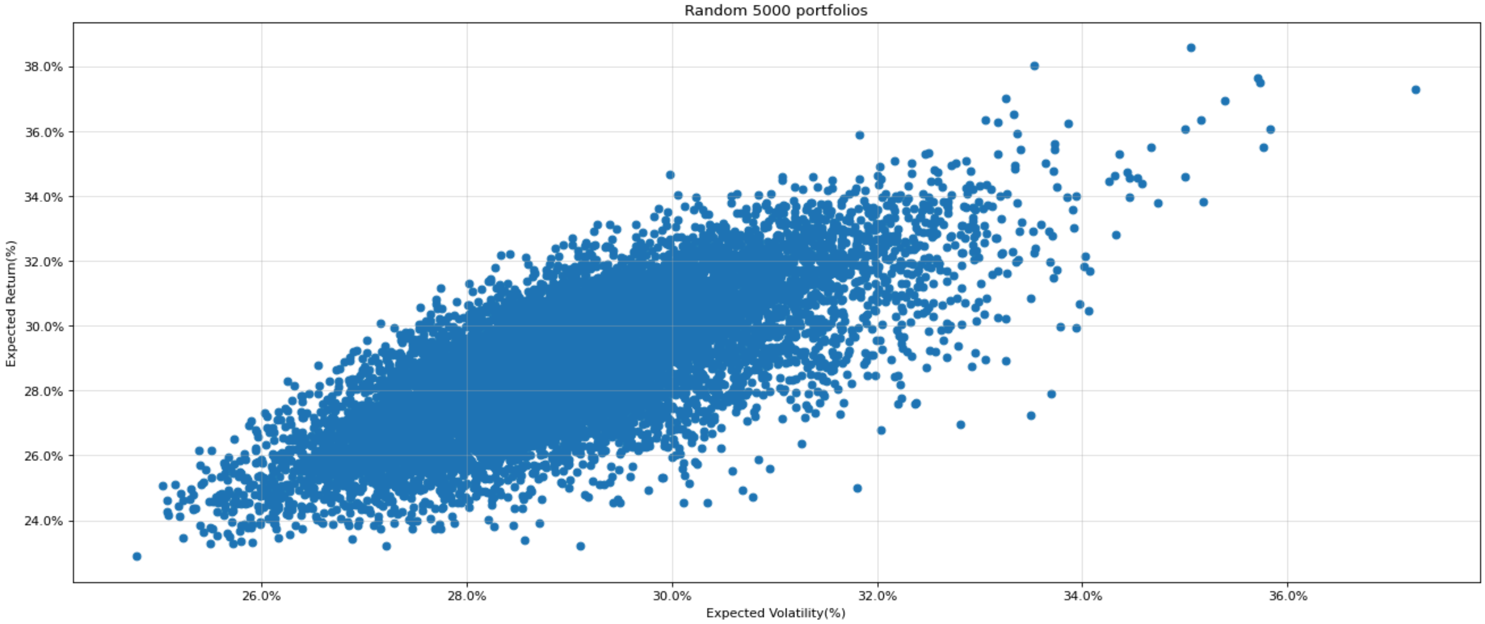


In [133…

```
# Test for random 10000 portfolios
return_list = []
volatility_list = []

for i in range(10000):
    random_weights = np.random.random(len(tickers))

    # Normalization is to ensure the total investment == 100%
    normalized_weights = random_weights / np.sum(random_weights)

    # To create a random expected return
    random_return = annual_return.dot(normalized_weights)

    return_list.append(random_return)
    random_variance = np.dot(np.dot(normalized_weights.T, annual_cov_mat), normalized_weights)
    random_std = np.sqrt(random_variance)
    volatility_list.append(random_std)
```

In [134…

```
# Create the random portfolio DataFrame
random_p = pd.DataFrame({'Return':return_list,'Volatility':volatility_list})

# Plot the 10000 portfolios in scatter
plt.figure(figsize=(20,8), dpi=80)
plt.scatter(random_p['Volatility'] * 100, random_p['Return'] * 100 )
plt.title('Random 5000 portfolios')
plt.xlabel('Expected Volatility(%)')
plt.ylabel('Expected Return(%)')
plt.gca().xaxis.set_major_formatter(PercentFormatter())
plt.gca().yaxis.set_major_formatter(PercentFormatter())
plt.grid(alpha=0.4)
plt.show()
```



Optimize the portfolio

In [135…
```
# Create the expected return and covariance of return
expected_returns = annual_return.values
covariance_returns = annual_cov_mat.values
```

In [136…
```
# Define the portfolio mean and volatility according to the weights
def mean_portfolio(weights):
    return np.dot(weights, expected_returns)

def volatility_portfolio(weights):
    return np.sqrt(np.dot(weights.T, np.dot(covariance_returns, weights)))
```

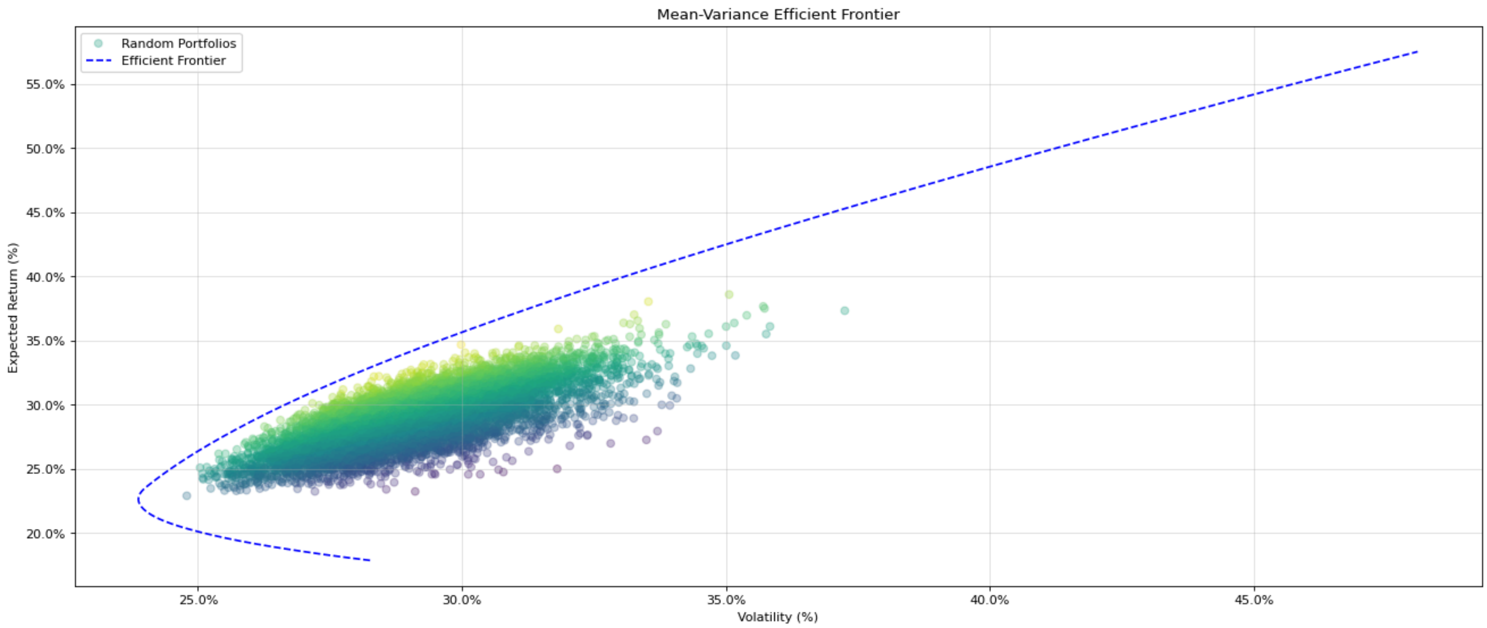
In [145…
```
# Bounds for weights, assume no short sales
# Lower bound is 0, upper bound is 1
bounds = tuple((0, 1) for _ in range(len(tickers)))

# Initial weights
random_weights = np.random.uniform(0, 1, len(tickers))
initial_weights = random_weights / np.sum(random_weights)

# Prepare to store results
portfolio_vol = []
portfolio_returns = []

# Define target returns
target_returns = np.linspace(expected_returns.min(), expected_returns.max(), 1000)

for target_return_p in target_returns:
    constraints = (
        {'type': 'eq', 'fun': lambda x: mean_portfolio(x) - target_return_p},
        {'type': 'eq', 'fun': lambda x: np.sum(x) - 1}
    )
    result_p = minimize(
        volatility_portfolio,
        initial_weights,
        method='SLSQP',
        bounds=bounds,
        constraints=constraints
    )

    # Add Success test
    if result_p.success:
        portfolio_vol.append(volatility_portfolio(result_p.x))
        portfolio_returns.append(target_return_p)
    else:
        print(f"Optimization failed for target return {target_return_p}")
```

In [146…
```
# Check if the length of data in volatility matches the length of expected return
print(f"Length of portfolio_volatility: {len(portfolio_vol)}")
print(f"Length of target_returns: {len(target_returns)}")
```

```
Length of portfolio_volatility: 1000
Length of target_returns: 1000
```

In [147…
```
# Plot the efficient frontier
plt.figure(figsize=(20, 8), dpi =80)
plt.scatter(random_p['Volatility'] * 100, random_p['Return'] * 100, c=random_p['Return']/random_p['Volatility'], marker='o', alpha= 0.3, label='Random Portfol
plt.plot(np.array(portfolio_vol) * 100, np.array(target_returns) * 100, 'b--', label='Efficient Frontier')
plt.xlabel('Volatility (%)')
plt.ylabel('Expected Return (%)')
plt.gca().xaxis.set_major_formatter(PercentFormatter())
plt.gca().yaxis.set_major_formatter(PercentFormatter())
plt.title('Mean-Variance Efficient Frontier')
plt.legend()
```

```python
plt.grid(alpha=0.4)
plt.show()
```



c). Explain what the efficient frontier means. Which portfolio would you invest in and why?

1. Each point on the frontier represents a portfolio that is optimized for either minimum risk for a certain return, or maximum return for a certain risk. \
2. The scatter of points below the frontier represents random portfolios that are not optimized

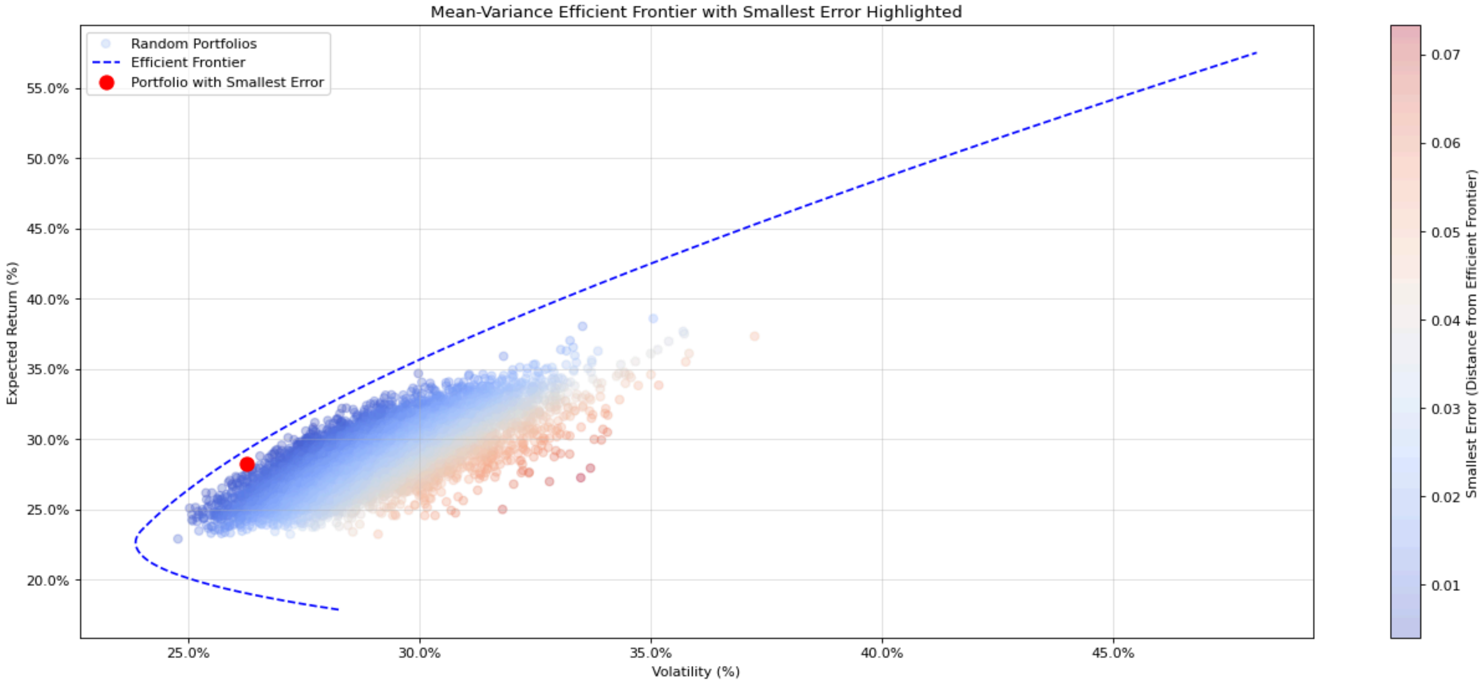
In [148…

```python
# Calculate the minimum errer between the efficient frontier and the portfolios
def calculate_distance(volatility, return_, frontier_vol, frontier_return):
    return np.sqrt((volatility - frontier_vol) ** 2 + (return_ - frontier_return) ** 2)

# Find the portfolio's return and volatility
smallest_errors = []
for index, row in random_p.iterrows():
    distances = [
        calculate_distance(row['Volatility'], row['Return'], vol, ret)
        for vol, ret in zip(portfolio_vol, portfolio_returns)
    ]
    smallest_errors.append(min(distances))

random_p['Smallest Error'] = smallest_errors

# To locate the smallest roor portfolio
min_error_idx = random_p['Smallest Error'].idxmin()
closest_portfolio = random_p.iloc[min_error_idx]

print(f"Portfolio with smallest error: \n{closest_portfolio}")
```

```
Portfolio with smallest error:
Return            0.282772
Volatility        0.262579
Smallest Error    0.003973
Name: 888, dtype: float64
```

In [149…

```python
# Plot the smallest error portfolio on the figure
plt.figure(figsize=(20, 8), dpi=80)
plt.scatter(random_p['Volatility'] * 100, random_p['Return'] * 100, c=random_p['Smallest Error'], cmap='coolwarm', marker='o', alpha=0.3, label='Random Portfo
plt.colorbar(label='Smallest Error (Distance from Efficient Frontier)')
plt.plot(np.array(portfolio_vol) * 100, np.array(target_returns) * 100, 'b--', label='Efficient Frontier')
plt.scatter(closest_portfolio['Volatility'] * 100, closest_portfolio['Return'] * 100, color='red', label='Portfolio with Smallest Error', s=100)
plt.xlabel('Volatility (%)')
plt.ylabel('Expected Return (%)')
plt.gca().xaxis.set_major_formatter(PercentFormatter())
plt.gca().yaxis.set_major_formatter(PercentFormatter())
plt.title('Mean-Variance Efficient Frontier with Smallest Error Highlighted')
plt.legend()
plt.grid(alpha=0.4)
plt.show()
```



d). Add short-sale constraints and plot the constrained efficient frontier together with the one from b). Explain any differences you see.

In [150…

```python
# Short sale is allowed; Lower bound is -1, upper bound is 1
bounds_short = tuple((-1, 1) for _ in range(len(tickers)))

# Define the initial weights
random_weights_shorts = np.random.uniform(-1, 1, len(tickers))
initial_weights_short = random_weights_shorts / np.sum(random_weights_shorts)

# Prepare to store results
portfolio_vol_shorts = []
portfolio_returns_shorts = []

# Define target returns
target_returns = np.linspace(expected_returns.min(), expected_returns.max(), 100)

for target_return_p in target_returns:
    constraints_shorts = (
        {'type': 'eq', 'fun': lambda x: mean_portfolio(x) - target_return_p},
        {'type': 'eq', 'fun': lambda x: np.sum(x) - 1}
    )
    result_p = minimize(
        volatility_portfolio,
        initial_weights_short,
        method='SLSQP',
        bounds=bounds_short,
        constraints=constraints_shorts
    )

    # Add Success test
    if result_p.success:
        portfolio_vol_shorts.append(volatility_portfolio(result_p.x))
        portfolio_returns_shorts.append(target_return_p)
    else:
        print(f"Optimization failed for target return {target_return_p}")
```

In [151…

```python
# Check if the length of data in volatility matches the length of expected return
print(f"Length of short portfolio_volatility: {len(portfolio_vol_shorts)}")
print(f"Length of short target_returns: {len(portfolio_returns_shorts)}")
```

```
Length of short portfolio_volatility: 100
Length of short target_returns: 100
```

In [152…

```python
plt.figure(figsize=(12, 6))
plt.scatter(
    random_p['Volatility'], random_p['Return'],
    c=random_p['Return'] / random_p['Volatility'],
    marker='o', alpha=0.3, label='Random Portfolios'
)
plt.plot(portfolio_vol_shorts, portfolio_returns_shorts , 'r-', label='Efficient Frontier (Short-Selling Allowed)')
plt.plot(portfolio_vol , portfolio_returns , 'b--', label='Efficient Frontier (No Short-Selling)')
plt.xlabel('Volatility ')
plt.ylabel('Expected Return')
plt.title('Mean-Variance Efficient Frontier with and without Short-Selling Constraints')
plt.legend()
plt.grid(alpha=0.4)
plt.show()
```

Differences

Without short selling: The efficient frontier without short-selling constraints is generally curved and lies above the cloud of random portfolios. This is because without short selling the portfolio optimization is more constrained, limiting the ability to reduce risk or increase return by taking negative positions in some assets.

With short selling: The efficient frontier with short-selling is higher than the without short-selling one, which means potentially reach higher return and lower volatility. Also, it offered the additional flexibility that allows for more optimal portfolios. This gives more flexibility in creating portfolios, allowing investors to potentially achieve higher returns for the same level of risk, or lower risk for the same return.

The no short-selling has more curved than short-selling, beacause it prevents negative asset allocations, which might otherwise help to offset risks from other assets.

e). If you wanted to invest in 200 equities, explain any difficulties in computing the covariance matrix and how these can be overcome.

if there are 200 equities, the matrix dimension will be 200 x 200, and the computation volume will be much larger. Also, the result will probably deviate from the actual result, which means the covariance estimate will be unreliable.

To reduce the size, we can use the fama-french model to reduce the number of covariances that need to be estimated. Moreover, we also can use the data shrinkage model to reduce the sample size for more accurate results.