Analyse Stock Portfiolio for Risks and Returns

Asset allocation is the most important decision that any investor needs to face. They need to decide how to spread their total capital over certain assets (in this case, stocks). When considering the allocation, the investor wants to balance the risk and the potential reward. At the same time, the allocation depends on factors such as individual goals, risk tolerance, and the investment horizon.

The key framework used in asset allocation is the Modern Portfolio Theory (MPT), which was introduced by the Nobel Prize winner Harry Markowitz. MPT describes how investors can construct portfolios to maximize their expected returns for a given level of risk or, conversely, minimize risk for a given level of expected return. The mathematical framework used to achieve this is called mean-variance optimization.

The main insight from MPT is that investors should not evaluate an asset's performance alone. Instead, they should evaluate how it would impact the performance of a portfolio of assets. Another important takeaway is the concept of diversification, which means that owning different kinds of assets reduces risk. That is because the loss or gain of a particular security has less impact on the overall portfolio's performance.

Your task

In the dynamic realm of finance, data scientists/analysts are often tasked with finding optimal investment strategies. Imagine you're one such analyst, and you were asked to build an effective portfolio comprising FAANG stocks – Facebook (Meta), Apple, Amazon, Netflix, and Google. Your goal is to maximize returns while mitigating risk.

In this project, you are tasked to find the optimal allocation to the FAANG stocks based on historical stock price data spanning the years 2020-2023. The dataset is stored in the `faang_stocks.csv` file. For each trading day, it contains the close prices of the five tech companies.

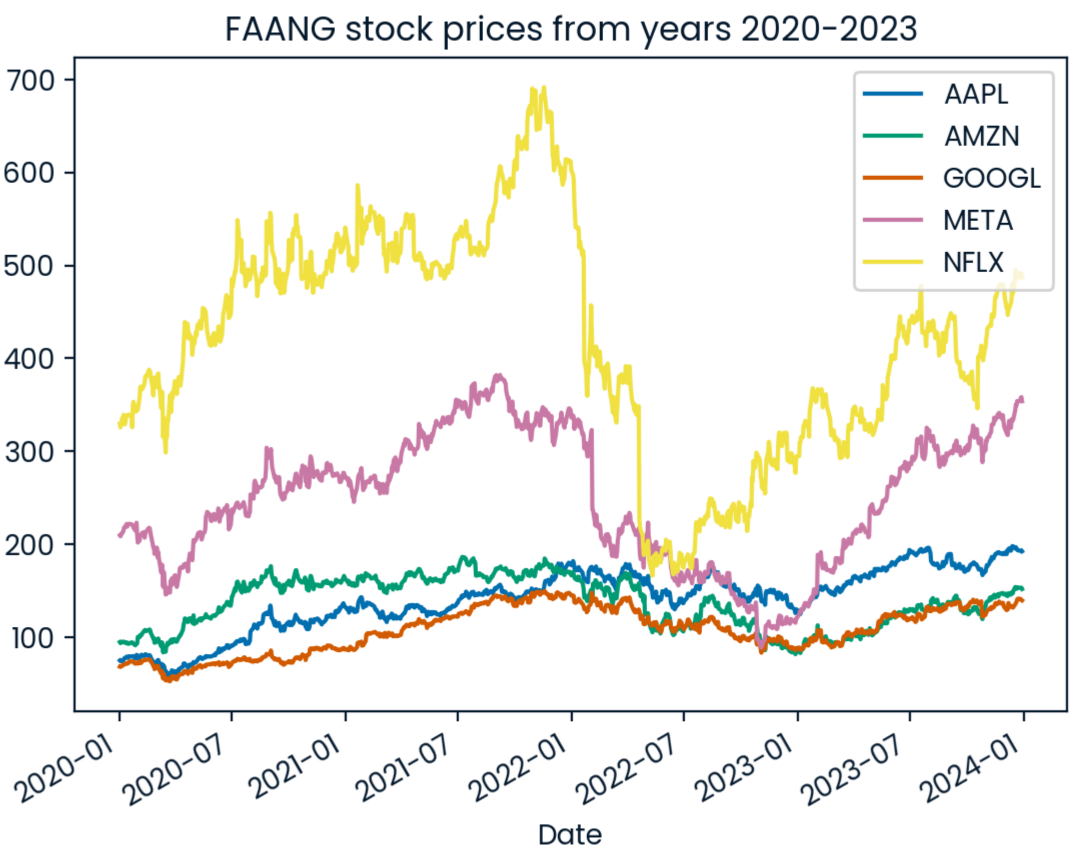
```python
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns

# Setting the plotting style to be colorblind-friendly
plt.style.use("seaborn-colorblind")

# Loading data
stock_prices_df = pd.read_csv("faang_stocks.csv", index_col="Date")

# Changing the index to a datetime type allows for easier filtering and plotting.
stock_prices_df.index = pd.to_datetime(stock_prices_df.index)
```

```python
stock_prices_df
```

Date	AAPL	AMZN	GOO
2020-01-02T00:00:00.000	75.09	94.9	
2020-01-03T00:00:00.000	74.36	93.75	
2020-01-06T00:00:00.000	74.95	95.14	
2020-01-07T00:00:00.000	74.6	95.34	
2020-01-08T00:00:00.000	75.8	94.6	
2020-01-09T00:00:00.000	77.41	95.05	
2020-01-10T00:00:00.000	77.58	94.16	
2020-01-13T00:00:00.000	79.24	94.57	
2020-01-14T00:00:00.000	78.17	93.47	
2020-01-15T00:00:00.000	77.83	93.1	
2020-01-16T00:00:00.000	78.81	93.9	
2020-01-17T00:00:00.000	79.68	93.24	
2020-01-21T00:00:00.000	79.14	94.6	
2020-01-22T00:00:00.000	79.43	94.37	
2020-01-23T00:00:00.000	79.81	94.23	

1,006 rows

```python
# Plotting the stock prices
stock_prices_df.plot(title="FAANG stock prices from years 2020-2023");
```



Task 1

```
# Calculate returns
returns_df = stock_prices_df.pct_change().dropna()

# Calculate the 1/n portfolio weights
portfolio_weights = 5 * [0.2]

# Calculate the portfolio returns of the 1/n portfolio
portfolio_returns = returns_df.dot(portfolio_weights)

# Calculate the expected portfolio return
benchmark_exp_return = portfolio_returns.mean()

# Calculate the portfolio's Sharpe ratio
benchmark_sharpe_ratio = (
    portfolio_returns.mean() / portfolio_returns.std() * np.sqrt(252)
)
```

Task 2

```
# Calculate the annualized expected returns and the covariance matrix
avg_returns = returns_df.mean() * 252
cov_mat = returns_df.cov() * 252
```

```
# Instantiate the EfficientFrontier object
ef = EfficientFrontier(avg_returns, cov_mat)

# Find the weights that maximize the Sharpe ratio
weights = ef.min_volatility()
mv_portfolio = pd.Series(weights)

# Find the minimized volatility
mv_portfolio_vol = ef.portfolio_performance(risk_free_rate=0)[1]

# Task 2 - alternative solution ----------------------------

# # Calculate the annualized expected returns and the covariance matrix
# avg_returns = returns_df.mean() * 252
# cov_mat = returns_df.cov() * 252

# # Define the function to find the portfolio volatility using the weights and the
covariance matrix
# def get_portfolio_volatility(weights, cov_mat):
#     return np.sqrt(np.dot(weights.T, np.dot(cov_mat, weights)))

# # Define the number of assets
# n_assets = len(avg_returns)

# # Define the bounds - the weights can be between 0 and 1
# bounds = tuple((0, 1) for asset in range(n_assets))

# # Define the initial guess - the equally weighted portfolio
# initial_guess = n_assets * [1.0 / n_assets]

# # Define the constraint - all weights must add up to 1
# constr = {"type": "eq", "fun": lambda x: np.sum(x) - 1}

# # Find the minimum volatility portfolio
# result = sco.minimize(
#     get_portfolio_volatility,
#     x0=initial_guess,
#     args=cov_mat,
#     method="SLSQP",
#     constraints=constr,
#     bounds=bounds,
# )

# # Store the portfolio weights
# mv_portfolio = pd.Series(result.x, index=avg_returns.index).round(2)
```