Finance | Portfolio Optimisation

Objective

Produce a financial dashboard with figures and metrics to help optimise the portfolio's

Strategy

Code measurements such as the Sharpe ratio, volatility and annual return to pick optimum stocks

Create figures to help visualise the stength of the portfolio and the stocks within

Use Python libraries to be able to create dashboards such as Panel

Output

Dashboard which allows to see the optimal weights for chosen assets, which can be rebalanced based on the historical data time frame and choice of stocks

Coding With 5 Example Stocks

In [110…
```
import yfinance as yf
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from datetime import datetime, timedelta
```

Microsoft

In [111…
```
ticker = yf.Ticker("MSFT")
info = ticker.info

print("Sector:", info.get("sector"))
print("Industry:", info.get("industry"))
print("Price:", info.get("regularMarketPrice"))

market_cap_MSFT = info.get("marketCap")
print("Market Cap (USD): ${:.2f}B".format(market_cap_MSFT / 1e9))
```

```
Sector: Technology
Industry: Software - Infrastructure
Price: 509.9
Market Cap (USD): $3790.17B
```

NVIDIA

In [112…
```
ticker = yf.Ticker("NVDA")
info = ticker.info

print("Sector:", info.get("sector"))
print("Industry:", info.get("industry"))
print("Price:", info.get("regularMarketPrice"))

market_cap_NVDA = info.get("marketCap")
print("Market Cap (USD): ${:.2f}B".format(market_cap_NVDA / 1e9))
```

```
Sector: Technology
Industry: Semiconductors
Price: 177.82
Market Cap (USD): $4329.38B
```

Taiwan Semiconductor Manufacturing

In [113…
```
ticker = yf.Ticker("TSM")
info = ticker.info

print("Sector:", info.get("sector"))
print("Industry:", info.get("industry"))
print("Price:", info.get("regularMarketPrice"))

market_cap_TSM = info.get("marketCap")
print("Market Cap (USD): ${:.2f}B".format(market_cap_TSM / 1e9))
```

```
Sector: Technology
Industry: Semiconductors
Price: 259.33
Market Cap (USD): $1345.02B
```

Apple

In [114…
```
ticker = yf.Ticker("AAPL")
info = ticker.info

print("Sector:", info.get("sector"))
print("Industry:", info.get("industry"))
print("Price:", info.get("regularMarketPrice"))

market_cap_AAPL = info.get("marketCap")
print("Market Cap (USD): ${:.2f}B".format(market_cap_AAPL / 1e9))
ev_AAPL = info.get("enterpriseValue")
print("Enterprise Value (USD): ${:.2f}B".format(ev_AAPL / 1e9))
```

```
Sector: Technology
Industry: Consumer Electronics
Price: 234.07
Market Cap (USD): $3473.69B
Enterprise Value (USD): $3520.02B
```

Alphabet Inc (Google)

In [115…
```
ticker = yf.Ticker("GOOGL")
info = ticker.info

print("Sector:", info.get("sector"))
```

```
print("Industry:", info.get("industry"))
print("Price:", info.get("regularMarketPrice"))

market_cap_GOOGL = info.get("marketCap")
print("Market Cap (USD): ${:.2f}B".format(market_cap_GOOGL / 1e9))
```

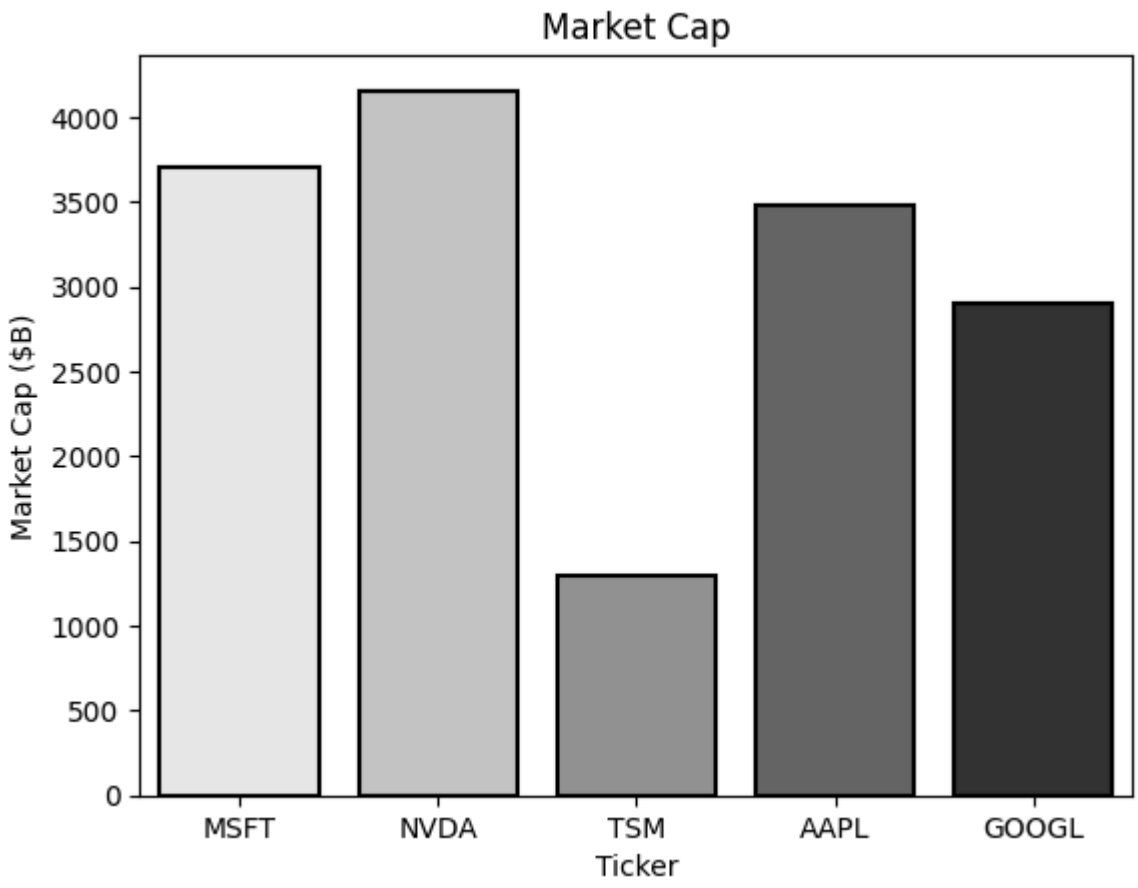
```
Sector: Communication Services
Industry: Internet Content & Information
Price: 240.8
Market Cap (USD): $2915.39B
```

Company Market Cap (Value)

In [116…
```
market_cap = [3704.77, 4157.49, 1301.40,
              3477.85, 2898.76]
tickers = ["MSFT", "NVDA", "TSM", "AAPL", "GOOGL"]
```

In [117…
```
df = pd.DataFrame({
    "tickers": tickers,
    "market_cap": market_cap
})
```

In [118…
```
sns.barplot(data=df, x="tickers", y="market_cap", linewidth=1.5,
            edgecolor="black", palette="Greys", hue="tickers"
           )

plt.title("Market Cap")
plt.ylabel("Market Cap ($B)")
plt.xlabel("Ticker")
plt.show()
```



Company Closing Price History (1Y)

In [119…
```
data = yf.download(tickers, period="1y", auto_adjust=False)
close_prices = data["Close"].reset_index()
print(close_prices.head(1))
```

```
[**********************100%**********************]  5 of 5 completed
Ticker       Date   AAPL       GOOGL        MSFT        NVDA    TSM
0      2024-09-13  222.5  157.460007  430.589996  119.099998  172.5
```

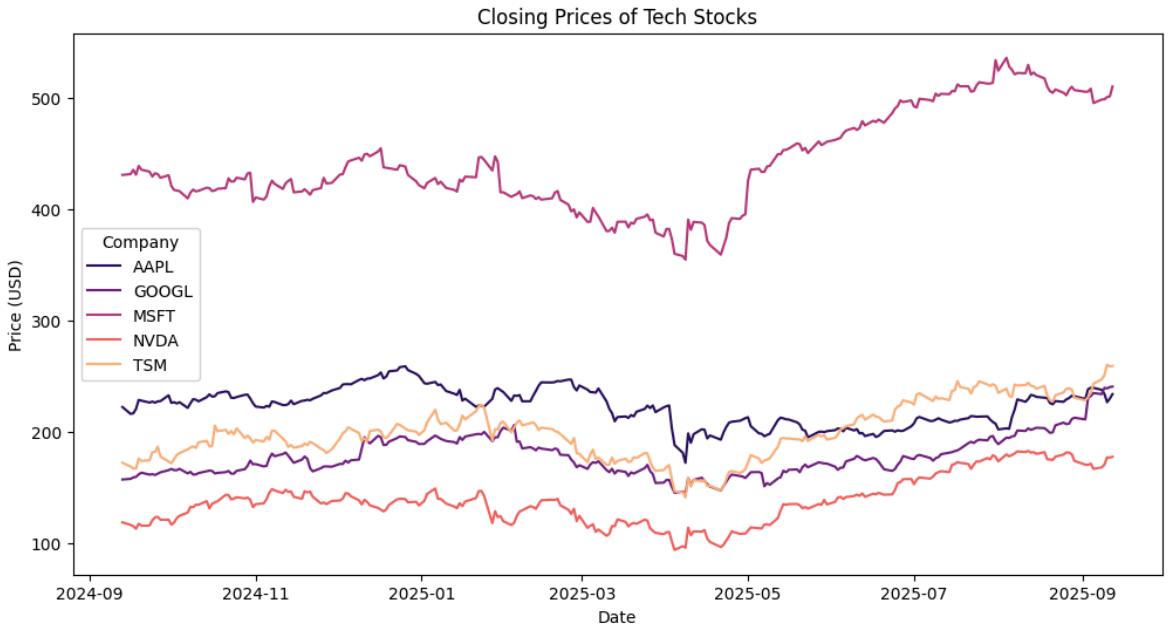
In [120…
```
df_long = close_prices.melt(id_vars="Date", var_name="Ticker", value_name
print(df_long.head(1))
```

```
        Date Ticker  Close
0 2024-09-13   AAPL  222.5
```

In [121…
```
print(df_long["Ticker"].unique())
```

```
['AAPL' 'GOOGL' 'MSFT' 'NVDA' 'TSM']
```

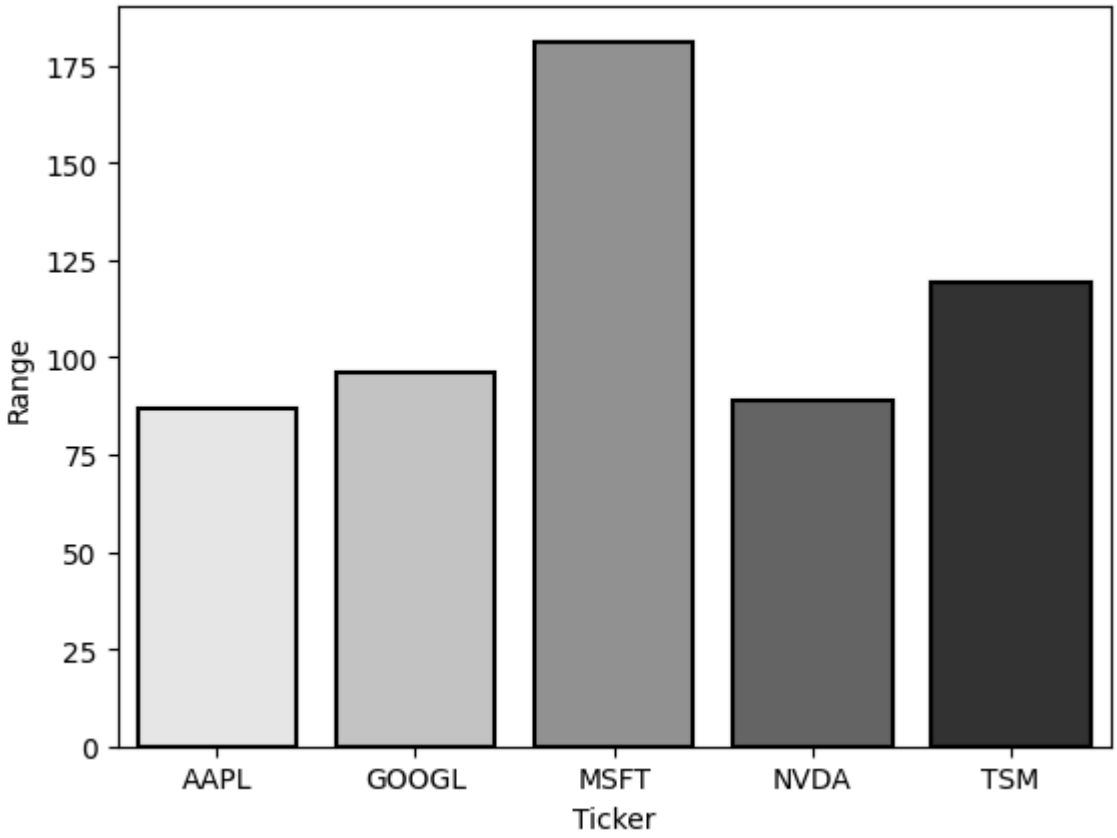
In [122…
```
plt.figure(figsize=(12,6))
sns.lineplot(data=df_long, x="Date", y="Close", hue="Ticker", palette="ma

plt.title("Closing Prices of Tech Stocks")
plt.ylabel("Price (USD)")
plt.xlabel("Date")
plt.legend(title="Company")
plt.show()
```



In [123…
```
# calculate range = max - min for each stock
price_only = close_prices.drop(columns="Date")
price_range = price_only.max() - price_only.min()
print(price_range)
```

```
Ticker
AAPL      86.599991
GOOGL     96.100006
MSFT     181.080017
NVDA      88.850006
TSM      119.070007
dtype: float64
```

In [124…
```
pr = price_range.reset_index()
pr.columns = ["Ticker", "Range"]

sns.barplot(data=pr, x="Ticker", y="Range", linewidth=1.5,
            edgecolor="black", palette="Greys", hue="Ticker")
plt.show()
```



Portfolio Optimisation

In [125…
```
end_date = datetime.today()
start_date = end_date - timedelta(days = 365)
print(start_date)
```

```
2024-09-14 13:09:41.978961
```

In [126…
```
adj_close_df = pd.DataFrame()
```

In [127…
```
for ticker in tickers:
    data = yf.download(ticker, start = start_date, end = end_date, auto_a
    adj_close_df[ticker] = data["Close"]
```

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

In [128…
```
log_returns = np.log(adj_close_df / adj_close_df.shift(1))
```

In [183…
```
log_returns.head()
```

Out[183…

Date	MSFT	NVDA	TSM	AAPL	GOOGL
2024-09-17	0.008794	-0.010242	-0.010285	0.002170	0.007940
2024-09-18	-0.010024	-0.019393	-0.000418	0.017830	0.003071
2024-09-19	0.018126	0.038926	0.052064	0.036395	0.014475
2024-09-20	-0.007827	-0.015992	-0.012218	-0.002932	0.008903
2024-09-23	-0.004052	0.002239	0.003899	-0.007610	-0.010693

In [129…
```
log_returns = log_returns.dropna()
```

Covariance Matrix

In [130…
```
cov_matrix = log_returns.cov()*252
```

Risk (Standard Deviation)

In [131…
```
def standard_deviation(weights, cov_matrix):
    variance = weights.T @ cov_matrix @ weights
    return np.sqrt(variance)
```

Expected Return

In [132…
```
def expected_return (weights, log_returns):
    return np.sum(log_returns.mean()*weights)*252
```

Sharpe Ratio

In [133…
```
def sharpe_ratio (weights, log_returns, cov_matrix, risk_free_rate):
    return(expected_return (weights, log_returns)- risk_free_rate) / stan
```

In [134…
```
risk_free_rate = 0.02
```

In [135…
```
def neg_sharpe_ratio(weights, log_returns, cov_matrix, risk_free_rate):
    return -sharpe_ratio(weights, log_returns, cov_matrix, risk_free_rate
```

Set Constraints and Bounds

In [136…
```
constraints = {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}
bounds = [(0, 0.4) for _ in range(len(tickers))]
initial_weights = np.array([1/len(tickers)]*len(tickers))
```

In [137…
```
optimized_results = minimize(neg_sharpe_ratio, initial_weights,
                             args=(log_returns, cov_matrix, risk_free_rat
                             method='SLSQP', constraints=constraints, bou
```

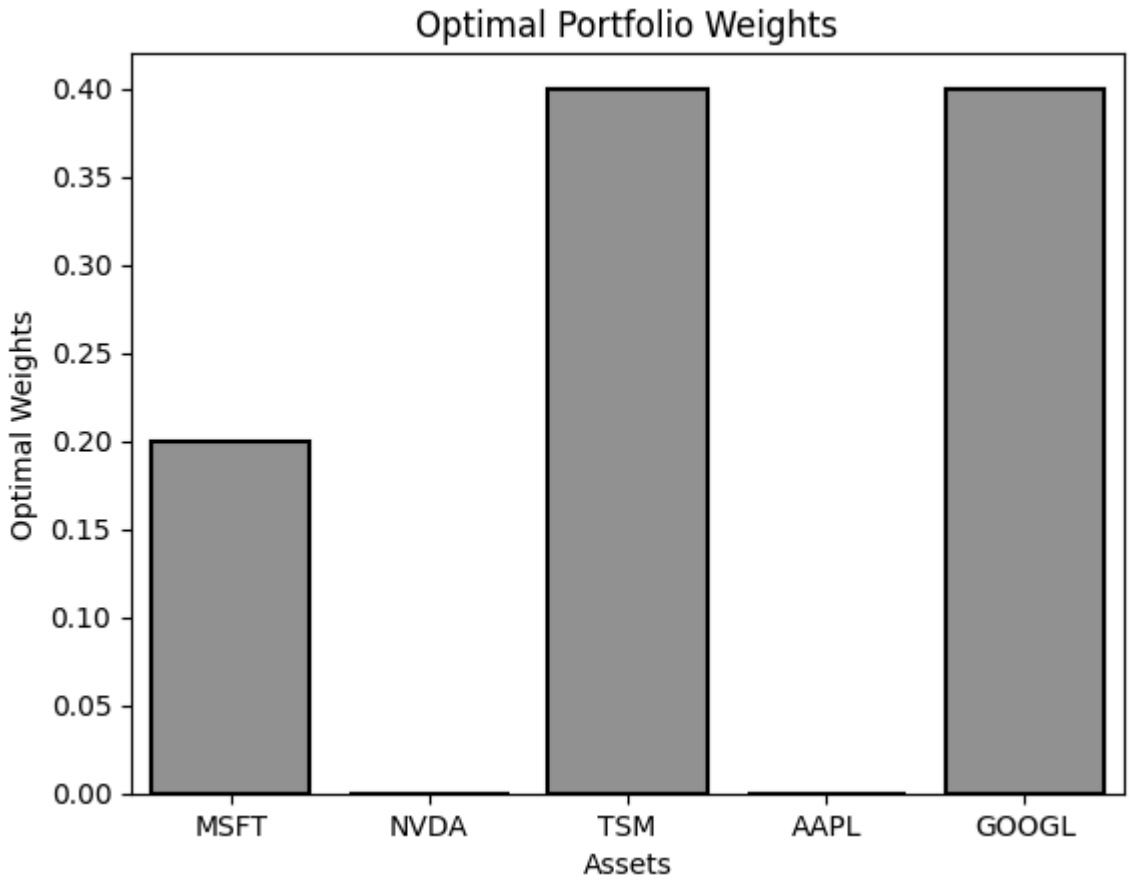
Get Optimal Weights

In [138…
```
optimal_weights = optimized_results.x
```

In [139…
```
print("Optimal Weights:")
for ticker, weight in zip(tickers, optimal_weights):
    print(f"{ticker}: {weight:.4f}")

optimal_portfolio_return = expected_return(optimal_weights, log_returns)
optimal_portfolio_volatility = standard_deviation(optimal_weights, cov_ma
optimal_sharpe_ratio = sharpe_ratio(optimal_weights, log_returns, cov_mat

print(f"Expected Annual Return: {optimal_portfolio_return:.4f}")
print(f"Expected Volatility: {optimal_portfolio_volatility:.4f}")
print(f"Sharpe Ratio: {optimal_sharpe_ratio:.4f}")
```

```
Optimal Weights:
MSFT: 0.2000
NVDA: 0.0000
TSM: 0.4000
AAPL: 0.0000
GOOGL: 0.4000
Expected Annual Return: 0.3866
Expected Volatility: 0.2841
Sharpe Ratio: 1.2907
```

In [140…
```
sns.barplot(x=tickers, y=optimal_weights, linewidth=1.5,
            edgecolor="black", palette="Greys", hue=False, legend=False)

plt.xlabel("Assets")
plt.ylabel("Optimal Weights")
plt.title("Optimal Portfolio Weights")
plt.show()
```



Portfolio Optimisation Methods Using The Pypfopt Library

In [141…
```
from pypfopt import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns
from pypfopt.discrete_allocation import DiscreteAllocation, get_latest_pr
```

In [142…
```
# Calculate expected returns and sample covariance
mu = expected_returns.mean_historical_return(adj_close_df)
S = risk_models.sample_cov(adj_close_df)
```

In [143…
```
# 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
print(cleaned_weights)
ef.portfolio_performance(verbose=True)
```

```
OrderedDict({'MSFT': 0.0, 'NVDA': 0.0, 'TSM': 0.28588, 'AAPL': 0.0, 'GOOG
L': 0.71412})
Expected annual return: 54.7%
Annual volatility: 30.1%
Sharpe Ratio: 1.82
```

Out[143…
```
(np.float64(0.5466035171002994),
 np.float64(0.3008750277995575),
 np.float64(1.816712809626879))
```

In [144…
```
latest_prices = get_latest_prices(adj_close_df)
```

In [145…
```
da = DiscreteAllocation(cleaned_weights, latest_prices, total_portfolio_v
allocation, leftover = da.greedy_portfolio()
print("Discrete allocation:", allocation)
print("Funds remaining: ${:.2f}".format(leftover))
```

```
Discrete allocation: {'GOOGL': 3, 'TSM': 1}
Funds remaining: $18.27
```

According to PyPortfolioOpt, 3 shares of GOOGL and 1 share of TSM should be bought to maximise the optimised weights as closely as possible when investing 1000 dollars. A total $30.19 remains after purchasing these shares.

Martin, R. A., (2021). PyPortfolioOpt: portfolio optimization in Python. Journal of Open Source Software, 6(61), 3066, https://doi.org/10.21105/joss.03066

Portfolio Optimisation Methods Using The Efficient Frontier

In [146…
```
from pandas_datareader import data as pdr
import datetime as dt
import scipy.optimize as sc
import plotly.graph_objects as go
```

In [147…
```
def get_data(stocks, start, end):
    stockData = yf.download(stocks, start=start, end=end)
    stockData = stockData['Close']
    returns = stockData.pct_change()
    meanReturns = returns.mean()
    covMatrix = returns.cov()
    return meanReturns, covMatrix
```

In [148…
```
def portfolioPerformance(weights, meanReturns, covMatrix):
    pReturns = np.sum(meanReturns*weights)*252
    pStd = np.sqrt(np.dot(weights.T, np.dot(covMatrix, weights))) * np.sq
    return pReturns, pStd
```

In [149…
```
def portfolioVarience(weights, meanReturns, covMatrix):
    return portfolioPerformance(weights, meanReturns, covMatrix)[1]**2
```

In [150…
```
def negativeSR(weights, meanReturns, covMatrix, riskFreeRate = 0):
    pReturns, pStd = portfolioPerformance(weights, meanReturns, covMatrix
    return - (pReturns - riskFreeRate)/pStd
```

In [151…
```
def portfolioReturns(weights, meanReturns, covMatrix):
    return portfolioPerformance(weights, meanReturns, covMatrix)[0]
```

In [153…
```
def maxSR(meanReturns, covMatrix, riskFreeRate = 0):
    numAssets = len(meanReturns)
    args = (meanReturns, covMatrix, riskFreeRate)
    constraints = ({'type':'eq', 'fun': lambda x: np.sum(x) - 1})
    bounds = tuple((0,1) for _ in range(numAssets))
    result = sc.minimize(negativeSR, numAssets*[1./numAssets], args=args,
                         method='SLSQP', bounds=bounds, constraints=const
    return result
```

In [154…
```
def minimizeVarience(meanReturns, covMatrix):
    numAssets = len(meanReturns)
    args = (meanReturns, covMatrix)
    constraints = ({'type':'eq', 'fun': lambda x: np.sum(x) - 1})
    bounds = tuple((0,1) for _ in range(numAssets))
    result = sc.minimize(portfolioVarience, numAssets*[1./numAssets], arg
    method='SLSQP', bounds=bounds, constraints=constraints)

    return result
```

In [155…
```
def efficientOpt(meanReturns, covMatrix, returnTarget):
    efficient_std = []
    numAssets = len(meanReturns)
    for target in returnTarget:
        constraints = (
            {'type':'eq', 'fun': lambda x: np.sum(x) - 1},
            {'type':'eq', 'fun': lambda x: portfolioReturns(x, meanReturn
                   )
        bounds = tuple((0,1) for _ in range(numAssets))
        result = sc.minimize(portfolioVarience, numAssets*[1./numAssets],
                        args=(meanReturns, covMatrix),
                        method = 'SLSQP', bounds=bounds, constraints=con
                        options={'maxiter': 1000})
        if result.success:
            _, std = portfolioPerformance(result.x, meanReturns, covMatri
            efficient_std.append(std)
        else:
            efficient_std.append(np.nan)
    return efficient_std
```

In [156…
```
def EF_graph(meanReturns, covMatrix):
    # Min Volatility
    minVarResult = minimizeVarience(meanReturns, covMatrix)
    minVolReturn, minVolStd = portfolioPerformance(minVarResult.x, meanRe

    # Max Sharpe Ratio
    maxSRResult = maxSR(meanReturns, covMatrix)
    maxSRReturn, maxSRStd = portfolioPerformance(maxSRResult.x, meanRetur

    # Efficient Frontier
    targetReturns = np.linspace(minVolReturn, maxSRReturn, 50)
    efficientStd = efficientOpt(meanReturns, covMatrix, targetReturns)

    # Plot
    fig = go.Figure()

     # Min Volatility
    fig.add_trace(go.Scatter(
        x=[round(minVolStd*100,2)], y=[round(minVolReturn*100,2)],
        mode='markers', name='Minimum Volatility',
        marker=dict(color='green', size=14, line=dict(width=3, color='bla
    ))

    # Max Sharpe Ratio
    fig.add_trace(go.Scatter(
        x=[round(maxSRStd*100,2)], y=[round(maxSRReturn*100,2)],
        mode='markers', name='Maximum Sharpe Ratio',
        marker=dict(color='red', size=14, line=dict(width=3, color='black
    ))
```

```
    # Efficient Frontier
    fig.add_trace(go.Scatter(
        x=[round(s*100,2) for s in efficientStd],
        y=[round(r*100, 2) for r in targetReturns],
        mode='lines', name='Efficient Frontier',
        line=dict(color='black', width=4, dash='dashdot')
    ))

    fig.update_layout(
        title="Portfolio Optimisation with Efficient Frontier",
        xaxis=dict(title="Annualised Volatility (%)"),
        yaxis=dict(title="Annualised Return (%)"),
        showlegend=True,
        width=800,
        height=600
    )

    fig.show()
```

In [157…

```
stocklist = ["MSFT", "NVDA", "TSM", "AAPL", "GOOGL"]
stocks = [stock for stock in stocklist]
endDate = dt.datetime.now()
startDate = endDate - dt.timedelta(days=365)
meanReturns, covMatrix = get_data(stocks, start=startDate , end=endDate)
EF_graph(meanReturns, covMatrix)
```

```
/var/folders/yk/lk83n4q55sg90xh1_1qbr0qm0000gn/T/ipykernel_30586/427743420
4.py:2: FutureWarning:

YF.download() has changed argument auto_adjust default to True

[**********************100%**********************]  5 of 5 completed
```

Portfolio Optimisation Method Using Monte Carlo Simulation

In [158…
```
def get_data(stocks, start, end):
    stockData = yf.download(stocks, start=start, end=end)
    stockData = stockData['Close']
    returns = stockData.pct_change()
    meanReturn = returns.mean()
    covMatrix = returns.cov()
    return meanReturn, covMatrix
```

In [159…
```
stocklist = ["MSFT", "NVDA", "TSM", "AAPL", "GOOGL"]
```

In [160…
```
stocks = [stock for stock in stocklist]
```

In [161…
```
endDate = dt.datetime.now()
startDate = endDate - dt.timedelta(days=365)
```

In [162…
```
meanReturn, covMatrix = get_data(stocks, startDate, endDate)
```

```
/var/folders/yk/lk83n4q55sg90xh1_1qbr0qm0000gn/T/ipykernel_30586/145827770
6.py:2: FutureWarning:

YF.download() has changed argument auto_adjust default to True

[**********************100%***********************]  5 of 5 completed
```

In [163…
```
weights = np.random.random(len(meanReturn))
```

In [164…
```
weights /= np.sum(weights)
```

In [165…
```
mc_sims = 100 # number of simulation
T = 30 # number of days
```

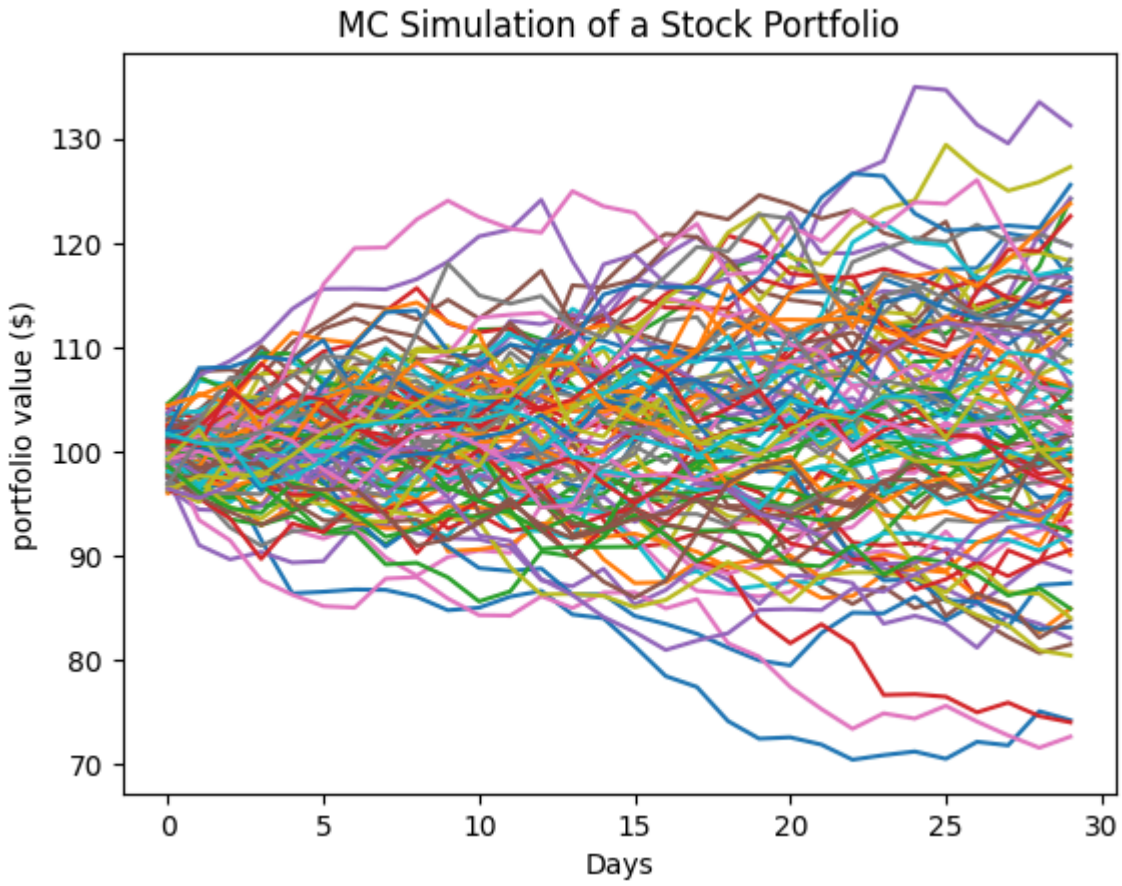
In [166…
```
meanM = np.tile(meanReturn.values, (T, 1))
```

In [167…
```
portfolio_sims = np.full(shape=(T, mc_sims), fill_value=0.0)
```

In [168…
```
initialPortfolio = 100
```

In [169…
```
for m in range(mc_sims):
    Z = np.random.normal(size=(T, len(weights)))
    L = np.linalg.cholesky(covMatrix)

    # Correlated daily returns (100,5)
    dailyReturn = meanM + (Z @ L.T)

    # Portfolio returns (100,)
    portfolio_sims[:, m] = np.cumprod(1 + dailyReturn @ weights) * initia
```

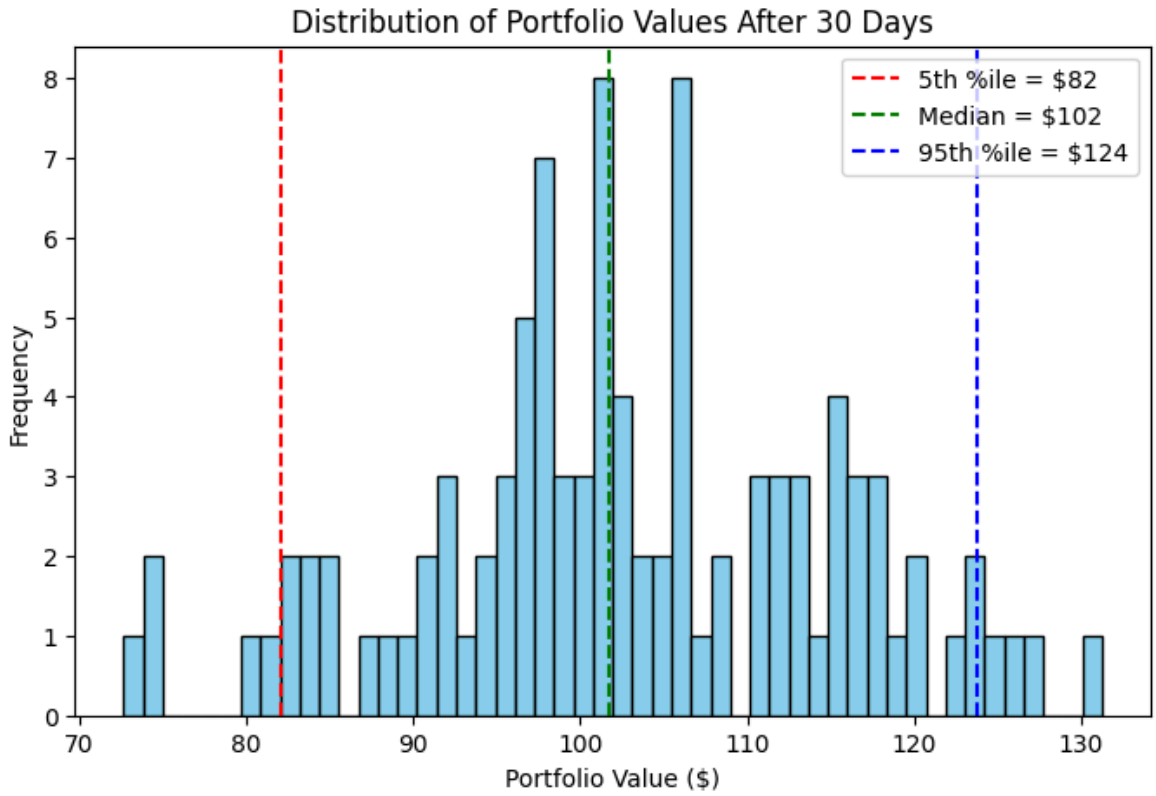
In [170…
```
plt.plot(portfolio_sims)
plt.ylabel("portfolio value ($)")
plt.xlabel("Days")
plt.title("MC Simulation of a Stock Portfolio")
plt.show()
```



In [171…

```
# Extract final portfolio values
final_values = portfolio_sims[-1, :]

# Compute percentiles
p5 = np.percentile(final_values, 5)
p50 = np.percentile(final_values, 50)  # median
p95 = np.percentile(final_values, 95)

print(f"5th percentile (worst case): ${p5:,.2f}")
print(f"50th percentile (median):    ${p50:,.2f}")
print(f"95th percentile (best case): ${p95:,.2f}")

# Plot histogram of final values
plt.figure(figsize=(8,5))
plt.hist(final_values, bins=50, color="skyblue", edgecolor="black")
plt.axvline(p5, color="red", linestyle="--", label=f"5th %ile = ${p5:,.0f
plt.axvline(p50, color="green", linestyle="--", label=f"Median = ${p50:,.
plt.axvline(p95, color="blue", linestyle="--", label=f"95th %ile = ${p95:
plt.title("Distribution of Portfolio Values After 30 Days")
plt.xlabel("Portfolio Value ($)")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

```
5th percentile (worst case): $82.02
50th percentile (median):    $101.68
95th percentile (best case): $123.79
```



Finance | Interactive Dashboard

In [220…
```
#data = yf.download(tickers, period="1y", auto_adjust=False)
#close_prices = data["Close"].reset_index()
#df_long = close_prices.melt(id_vars="Date", var_name="Ticker", value_nam
#print(df_long.head(1))
```

```
[**********************100%**********************]  5 of 5 completed
        Date Ticker  Close
0 2024-09-13   AAPL  222.5
```

In [255…
```
# Paste this into a Jupyter cell and run.
import yfinance as yf
import pandas as pd
import numpy as np
import panel as pn
import plotly.graph_objects as go
from scipy.optimize import minimize

pn.extension('plotly', 'tabulator')

# ----------------------
# Helpers: data & returns
# ----------------------
@pn.cache
def get_price_and_returns(tickers, period):
    tickers = [t.strip().upper() for t in tickers if t.strip()]
    if len(tickers) == 0:
        raise ValueError("No tickers provided.")
    price = yf.download(tickers, period=period, auto_adjust=True)['Close'
    # If single ticker, convert to DataFrame with column name
    if isinstance(price, pd.Series):
        price = price.to_frame(name=tickers[0])
```

```
    returns = price.pct_change().dropna()
    mean_daily = returns.mean()
    cov_daily = returns.cov()
    return price, returns, mean_daily, cov_daily

# ----------------------
# Efficient frontier math
# ----------------------
def compute_ef(tickers, period, rf=0.0, n_points=50, horizon="Annual"):
    price, returns, mean_daily, cov_daily = get_price_and_returns(tickers

    # Scaling factor based on horizon
    trading_days = {"Annual": 252, "Monthly": 21, "Weekly": 5}
    if horizon not in trading_days:
        raise ValueError("Invalid horizon. Choose from 'Annual', 'Monthly

    scale = trading_days[horizon]

    # expected returns and covariance matrix in chosen horizon units
    mu = mean_daily * scale
    S = cov_daily * scale

    assets = mu.index.tolist()
    n = len(mu)
    bounds = tuple((0, 1) for _ in range(n))
    cons_sum = {'type': 'eq', 'fun': lambda w: np.sum(w) - 1.0}
    init = np.repeat(1.0 / n, n)

    # objective functions (annual units)
    def portfolio_return(w):
        return float(np.dot(w, mu))

    def portfolio_vol(w):
        return float(np.sqrt(np.dot(w, S @ w)))

    def portfolio_var(w):
        return float(np.dot(w, S @ w))

    def neg_sharpe(w):
        ret = portfolio_return(w)
        vol = portfolio_vol(w)
        # small safety to avoid divide by zero
        return - (ret - rf) / (vol + 1e-12)

    # Max Sharpe
    res_sh = minimize(neg_sharpe, init, method='SLSQP', bounds=bounds, co
    w_sh = res_sh.x
    ret_sh = portfolio_return(w_sh)
    vol_sh = portfolio_vol(w_sh)
    sharpe_sh = (ret_sh - rf) / (vol_sh + 1e-12)

    # Minimum variance
    res_min = minimize(portfolio_var, init, method='SLSQP', bounds=bounds
    w_min = res_min.x
    ret_min = portfolio_return(w_min)
    vol_min = portfolio_vol(w_min)

    # Efficient frontier points (target returns between min var and max-s
    target_returns = np.linspace(ret_min, ret_sh, n_points)
    ef_vols = []
```

```
    ef_weights = []
    for target in target_returns:
        cons = (
            {'type': 'eq', 'fun': lambda w: np.sum(w) - 1.0},
            {'type': 'eq', 'fun': lambda w, target=target: float(np.dot(w
        )
        res = minimize(portfolio_var, init, method='SLSQP', bounds=bounds
        if res.success:
            ef_vols.append(np.sqrt(res.fun))
            ef_weights.append(res.x)
        else:
            ef_vols.append(np.nan)
            ef_weights.append(None)

    weights_df = pd.DataFrame([w_sh, w_min], index=["Max Sharpe", "Min Va
    perf_df = pd.DataFrame({
        "Metric": ["Annual Return (%)", "Annual Volatility (%)", "Sharpe
        "Max Sharpe": [ret_sh * 100, vol_sh * 100, sharpe_sh],
        "Min Variance": [ret_min * 100, vol_min * 100, (ret_min - rf) / (
    })

    return {
        "mu": mu, "S": S,
        "w_sh": w_sh, "ret_sh": ret_sh, "vol_sh": vol_sh, "sharpe_sh": sh
        "w_min": w_min, "ret_min": ret_min, "vol_min": vol_min,
        "ef_returns": target_returns, "ef_vols": np.array(ef_vols),
        "weights_df": weights_df, "perf_df": perf_df
    }

# ----------------------
# Plot Efficient Frontier
# ----------------------
def ef_panel(tickers_input, period, rf=0.0, horizon="Annual"):
    tickers = [t.strip().upper() for t in tickers_input.split(",") if t.s
    if len(tickers) < 2:
        return pn.pane.Markdown("**Select at least 2 tickers for Efficien
    try:
        stats = compute_ef(tickers, period, rf=rf, n_points=60, horizon=h
    except Exception as e:
        return pn.pane.Markdown(f"Error computing EF: {e}")

    # filter valid points
    mask = ~np.isnan(stats["ef_vols"])
    x = stats["ef_vols"][mask] * 100.0     # vol %
    y = stats["ef_returns"][mask] * 100.0  # return %

    fig = go.Figure()
    fig.add_trace(go.Scatter(x=x, y=y, mode='lines', name='Efficient Fron
                             line=dict(color='royalblue', width=2)))
    fig.add_trace(go.Scatter(x=[stats["vol_min"]*100], y=[stats["ret_min"
                             mode='markers', marker=dict(color='green', s
    fig.add_trace(go.Scatter(x=[stats["vol_sh"]*100], y=[stats["ret_sh"]*
                             mode='markers', marker=dict(color='red', siz

    fig.update_layout(title="Efficient Frontier (annualised)",
                      xaxis_title="Volatility (%)", yaxis_title="Return (
                      width=900, height=550)

    # ---- Weights table in percentages ----
    weights_df = pd.DataFrame({
```

```
        "Max Sharpe (%)": stats["w_sh"] * 100,
        "Min Variance (%)": stats["w_min"] * 100
    }, index=stats["mu"].index).round(2)

    # ---- Performance metrics table ----
    perf_df = pd.DataFrame({
        "Metric": [f"{horizon} Return (%)", f"{horizon} Volatility (%)",
        "Max Sharpe": [stats["ret_sh"] * 100, stats["vol_sh"] * 100, stat
        "Min Variance": [stats["ret_min"] * 100, stats["vol_min"] * 100,
                         (stats["ret_min"] - rf) / (stats["vol_min"] + 1e
    }).round(2)

    return pn.Column(
        pn.pane.Plotly(fig, config={'responsive': True}),
        pn.pane.Markdown("### Portfolio Weights (%)"),
        pn.pane.DataFrame(weights_df, width=900, height=200),
        pn.pane.Markdown("### Performance Metrics"),
        pn.pane.DataFrame(perf_df, width=900, height=150)
    )

# -----------------------
# Monte Carlo simulation
# -----------------------
def monte_carlo_panel(tickers_input, period, portfolio_value=10000, sims=
    tickers = [t.strip().upper() for t in tickers_input.split(",") if t.s
    if len(tickers) < 2:
        return pn.pane.Markdown("**Select at least 2 tickers for Monte Ca
    try:
        price, returns, mean_daily, cov_daily = get_price_and_returns(tic
    except Exception as e:
        return pn.pane.Markdown(f"Error fetching returns: {e}")

    n = len(mean_daily)

    # choose weights: use EF max-Sharpe weights if requested (fallback to
    weights = None
    if use_ef_weights:
        try:
            stats = compute_ef(tickers, period)
            weights = stats["w_sh"]
        except Exception:
            weights = np.repeat(1.0 / n, n)
    else:
        weights = np.repeat(1.0 / n, n)
    weights = np.array(weights)
    weights = weights / weights.sum()

    # jitter covariance to ensure positive-definite for cholesky
    cov_daily_j = cov_daily.values + 1e-9 * np.eye(n)
    try:
        L = np.linalg.cholesky(cov_daily_j)
    except np.linalg.LinAlgError:
        # fallback: eigenvalue regularization
        evals, evecs = np.linalg.eigh(cov_daily)
        evals_clipped = np.clip(evals, 1e-8, None)
        cov_regular = (evecs @ np.diag(evals_clipped) @ evecs.T)
        L = np.linalg.cholesky(cov_regular)

    sims_matrix = np.zeros((horizon, sims))
    mu_daily = mean_daily.values
```

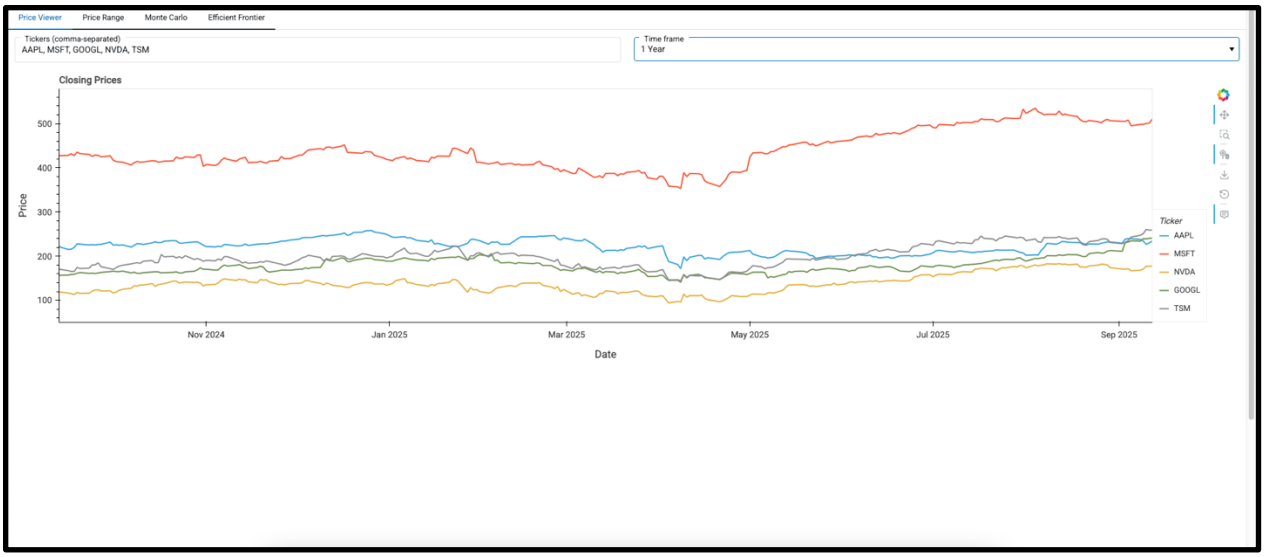
```
    for m in range(sims):
        Z = np.random.normal(size=(horizon, n))
        daily_sim = np.tile(mu_daily, (horizon, 1)) + Z @ L.T         # c
        port_daily = daily_sim @ weights                             # po
        sims_matrix[:, m] = np.cumprod(1.0 + port_daily) * portfolio_valu

    # Plot a subset of simulations (to avoid too many traces)
    n_to_plot = min(80, sims)
    idx = np.random.choice(sims, size=n_to_plot, replace=False)

    fig = go.Figure()
    for i in idx:
        fig.add_trace(go.Scatter(y=sims_matrix[:, i], mode='lines', line=
    # median path
    median_path = np.median(sims_matrix, axis=1)
    fig.add_trace(go.Scatter(y=median_path, mode='lines', line=dict(color

    fig.update_layout(title=f"Monte Carlo ({sims} sims, horizon={horizon}
                      width=900, height=450)

    # histogram of final values with percentiles
    final_vals = sims_matrix[-1, :]
    p5, p50, p95 = np.percentile(final_vals, [5, 50, 95])
    hist = go.Figure()
    hist.add_trace(go.Histogram(x=final_vals, nbinsx=40, marker=dict(colo
    hist.add_vline(x=p5, line=dict(color='red', dash='dash'), annotation_
    hist.add_vline(x=p50, line=dict(color='black', dash='dash'), annotati
    hist.add_vline(x=p95, line=dict(color='green', dash='dash'), annotati
    hist.update_layout(title="Distribution of final portfolio values", wi

    stats_table = pd.DataFrame({
        "Metric": ["Portfolio initial value", "5th percentile", "Median",
        "Value": [f"{portfolio_value:,.0f}", f"{p5:,.2f}", f"{p50:,.2f}",
    })

    return pn.Column(
        pn.pane.Plotly(fig, config={'responsive': True}),
        pn.pane.Plotly(hist, config={'responsive': True}),
        pn.pane.DataFrame(stats_table, width=900)
    )

# ----------------------
# Widgets and dashboard
# ----------------------
ticker_input = pn.widgets.TextInput(name="Tickers (comma-separated)", val
timeframe_selector = pn.widgets.Select(name="Time frame", options={"1 Mon
rf_input = pn.widgets.FloatInput(name="Risk-free rate (annual decimal)",
portfolio_value = pn.widgets.NumberInput(name="Portfolio value (currency)
mc_sims = pn.widgets.IntSlider(name="MC simulations", start=50, end=2000,
mc_horizon = pn.widgets.IntSlider(name="MC horizon (days)", start=5, end=
use_ef_weights = pn.widgets.Checkbox(name="Use Max-Sharpe weights for Mon
horizon_selector = pn.widgets.Select(name="Return Horizon", options=["Ann


ef_tab = pn.bind(ef_panel, tickers_input=ticker_input, period=timeframe_s
mc_tab = pn.bind(monte_carlo_panel, tickers_input=ticker_input, period=ti
                 portfolio_value=portfolio_value, sims=mc_sims, horizon=m

dashboard = pn.Tabs(
```

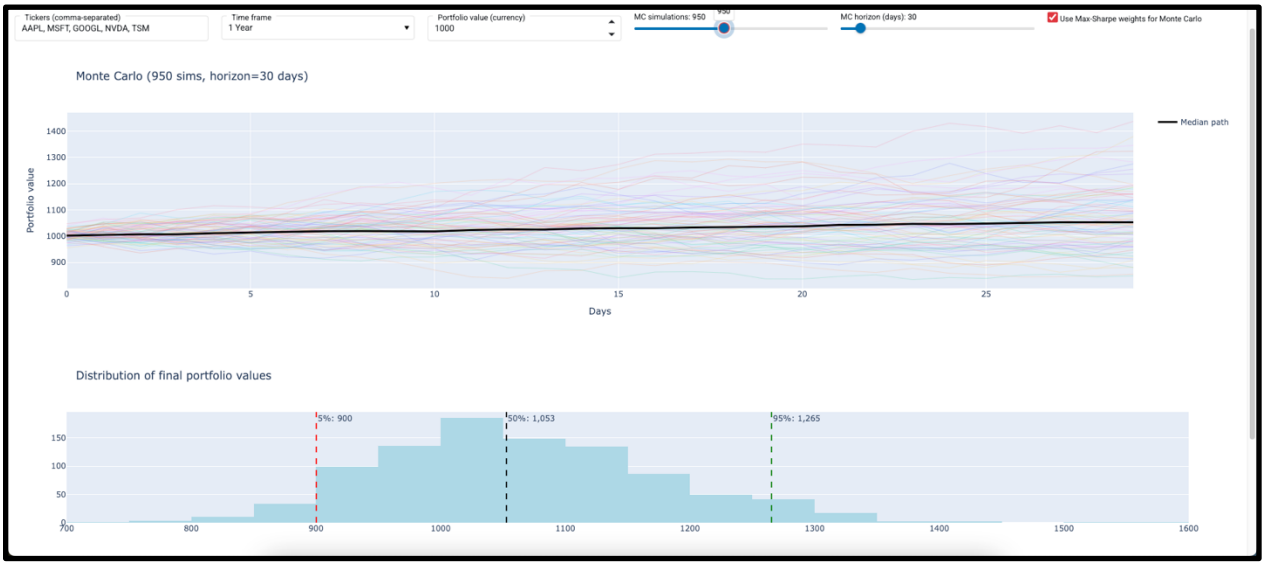
```
    ("Efficient Frontier", pn.Column(pn.Row(ticker_input, timeframe_selec
    ("Monte Carlo", pn.Column(pn.Row(ticker_input, timeframe_selector, po
)

# ----------------------
# Price Viewer
# ----------------------
import hvplot.pandas  # for interactive plotting

def price_viewer_panel(tickers_input, period):
    tickers = [t.strip().upper() for t in tickers_input.split(",") if t.s
    if len(tickers) == 0:
        return pn.pane.Markdown("**Enter at least one ticker**", width=80
    try:
        price, returns, mean_daily, cov_daily = get_price_and_returns(tic
    except Exception as e:
        return pn.pane.Markdown(f"Error fetching prices: {e}")

    df = price.reset_index().melt(id_vars="Date", var_name="Ticker", valu
    return df.hvplot.line(x="Date", y="Price", by="Ticker", width=900, he

# ----------------------
# Price Range
# ----------------------
import seaborn as sns
import matplotlib.pyplot as plt

def price_range_panel(tickers_input, period):
    tickers = [t.strip().upper() for t in tickers_input.split(",") if t.s
    if len(tickers) == 0:
        return pn.pane.Markdown("**Enter at least one ticker**", width=80

    try:
        price, returns, mean_daily, cov_daily = get_price_and_returns(tic
    except Exception as e:
        return pn.pane.Markdown(f"Error fetching prices: {e}")

    price_only = price.dropna()
    price_range = price_only.max() - price_only.min()
    pr = price_range.reset_index()
    pr.columns = ["Ticker", "Range"]

    fig, ax = plt.subplots(figsize=(7, 4))
    sns.barplot(data=pr, x="Ticker", y="Range", linewidth=1.5,
                edgecolor="black", palette="Greys", hue="Ticker", dodge=F
    ax.set_title(f"Price Range over {period}")
    ax.set_ylabel("Range")
    ax.set_xlabel("Ticker")
    ax.legend([], [], frameon=False)
    pane = pn.pane.Matplotlib(fig, tight=True)
    plt.close(fig)
    return pane

# ----------------------
# Bind tabs
# ----------------------
price_view_tab = pn.bind(price_viewer_panel, tickers_input=ticker_input,
price_range_tab = pn.bind(price_range_panel, tickers_input=ticker_input,
ef_tab = pn.bind(ef_panel, tickers_input=ticker_input, period=timeframe_s
mc_tab = pn.bind(monte_carlo_panel, tickers_input=ticker_input, period=ti
```

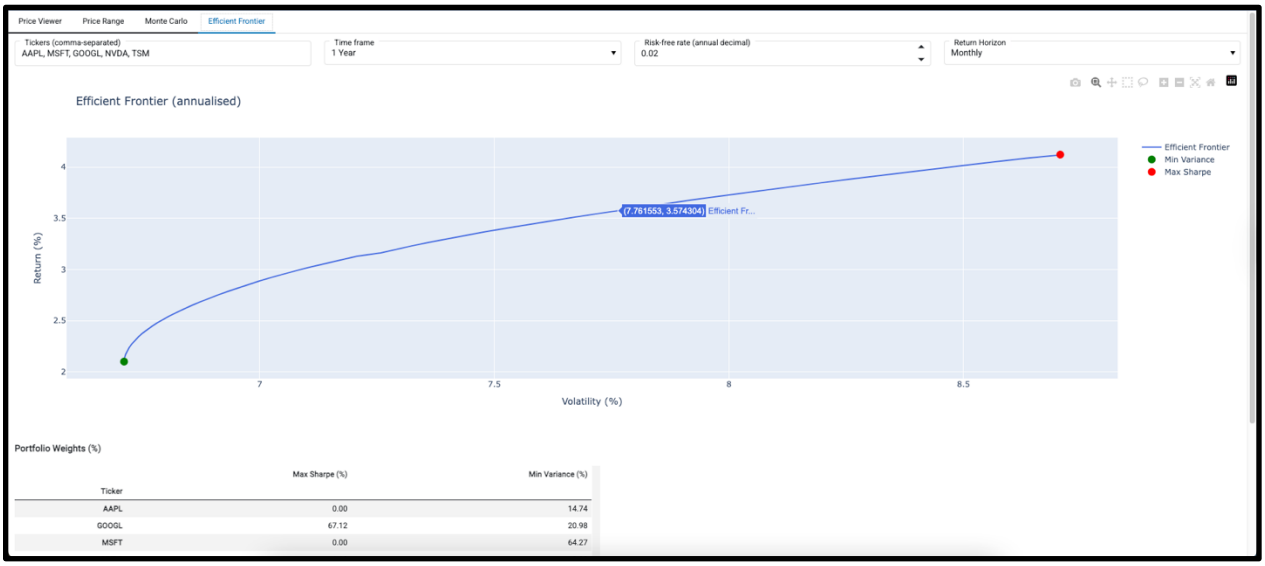
```
                portfolio_value=portfolio_value, sims=mc_sims, horizon=m

dashboard = pn.Tabs(
    ("Price Viewer", pn.Column(pn.Row(ticker_input, timeframe_selector),
    ("Price Range", pn.Column(pn.Row(ticker_input, timeframe_selector), p
    ("Monte Carlo", pn.Column(pn.Row(ticker_input, timeframe_selector, po
    ("Efficient Frontier", pn.Column(pn.Row(ticker_input, timeframe_selec
)

dashboard.servable()
dashboard

if __name__ == "__main__":
    pn.serve(dashboard, port=5008, show=True)
```

```
Launching server at http://localhost:5008
```

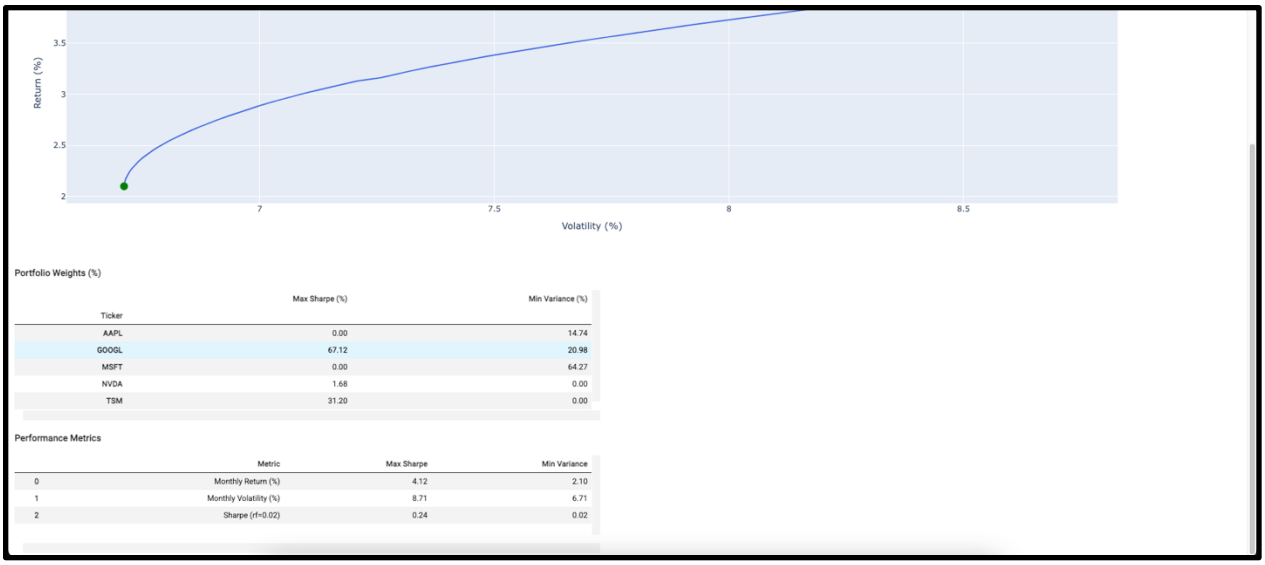
All Four Dashboard Tab Screenshots Produced on Panel







All Four Dashboard Tab Screenshots Produced on Panel



Portfolio Weights (%)

Ticker	Max Sharpe (%)	Min Variance (%)
AAPL	0.00	14.74
GOOGL	67.12	20.98
MSFT	0.00	64.27
NVDA	1.68	0.00
TSM	31.20	0.00

Performance Metrics

	Metric	Max Sharpe	Min Variance
0	Monthly Return (%)	4.12	2.10
1	Monthly Volatility (%)	8.71	6.71
2	Sharpe (rf=0.02)	0.24	0.02