Portfolio Optimization & Performance Analysis

The below project is for Educational purposes only!

Overview: This project focuses on portfolio optimization by analyzing the performance of randomly selected technology and financial stocks in a portfolio over a 30-year period. Historical data (2000-2024) and projected data (2025-2030) have been used to calculate cumulative returns, volatility, and risk metrics. The Monte Carlo Simulation was used for forecasting future portfolio performance by simulating possible market conditions and evaluating risk and return over a given period of time. This project also applies techniques such as maximization of the Sharpe ratio to create an optimal portfolio. To effectively visualize the data, the project includes a histogram of the Monte Carlo Simulation, Actual VS Project Returns of the Portfolio, and projected cumulative returns of all stocks in the portfolio. This analysis also identifies the best and worst performing stock.

In [1]:

```python
# importing required libraries
import numpy as np
import pandas as pd

import datetime as dt
import yfinance as yf
import pandas_datareader.data as web

import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

from scipy.optimize import minimize
```

Random Sample Selection

In [2]:

```python
# random selection

# customize seed for different stocks
np.random.seed(42)

# list of all potential stocks (can be customized)
ticker_list = ['AAPL','AXP', 'BAC', 'C', 'CSCO', 'GS', 'IBM', 'INTC', 'JPM', 'MSFT', 'NVDA', 'CRM', 'QCOM', 'NOW', 'ORCL', 'AVGO', 'TXN', 'GDDY',
               'WIX', 'TSM', 'TSLA', 'SNOW', 'HUBS', 'DOCU', 'MS']

# selecting 15 random stocks
stock_list = np.random.choice(ticker_list,15,replace=False)
print(f'These are the fifteen stocks assigned to you: {" ".join(stock_list)}')

# sorting the list from A-Z
stock_list = list(np.sort(stock_list))
```

```
These are the fifteen stocks assigned to you: JPM TXN AAPL DOCU CRM MSFT NOW AXP HUBS GS BAC QCOM AVGO C CSCO
```

In [4]:

```python
# finding optimal historical portfolio since 2000 to today, using 15 stocks.
# Assume no short-selling constraints

# defining start and end date
start = dt.datetime(2000, 1, 1)
end = dt.datetime(2024, 12, 31)

# getting returns of the stocks
returns = yf.download(stock_list,start-pd.offsets.BDay(1), end+pd.offsets.BDay(1), auto_adjust=False)['Adj Close'].pct_change().dropna()

returns.head()
```

```
[*********************100%***********************]  15 of 15 completed
```

Out[4]:

Ticker	AAPL	AVGO	AXP	BAC	C	CRM	CSCO	DOCU	GS	HUBS	JPM	MSFT	NOW	QCOM	TXN
Date															
2018-04-30	0.018112	-0.014434	-0.008633	-0.007629	-0.010436	0.006405	-0.009394	-0.027687	-0.006130	-0.018081	-0.005667	-0.024004	0.018140	-0.001957	-0.010536
2018-05-01	0.023236	0.004184	-0.001721	0.001003	-0.000293	0.017522	0.012193	0.021227	-0.006965	-0.005666	0.000000	0.015826	-0.004334	-0.003725	0.019422
2018-05-02	0.044175	-0.007118	-0.009637	-0.012354	-0.003810	-0.002356	-0.021638	-0.003295	-0.011070	0.014245	-0.007906	-0.015684	0.000484	-0.011019	-0.004255
2018-05-03	0.001812	-0.015257	-0.005019	-0.012846	-0.000736	0.012946	0.013224	0.067141	-0.002734	0.010300	-0.006301	0.005989	0.019517	0.000995	0.006216
2018-05-04	0.039233	0.023396	0.012456	0.003424	0.004732	0.005707	0.019352	-0.020734	0.006555	0.019926	0.011096	0.011587	0.014757	0.043332	0.016605

In [5]:

```python
# getting monthly returns
returns_mon = returns.resample(rule = 'ME').apply(lambda x: x.add(1).prod().sub(1))
returns_mon.tail()
```

Out[5]:

Ticker	AAPL	AVGO	AXP	BAC	C	CRM	CSCO	DOCU	GS	HUBS	JPM	MSFT	NOW	QCOM	TXN
Date															
2024-08-31	0.032353	0.013319	0.022170	0.010915	-0.025235	-0.022798	0.043137	0.067231	0.008327	0.004104	0.056391	-0.001095	0.049866	-0.031224	0.051666
2024-09-30	0.017467	0.062937	0.048521	-0.019909	-0.000639	0.083985	0.053027	0.048640	-0.029672	0.065181	-0.062011	0.031548	0.046070	-0.024936	-0.036251
2024-10-31	-0.030429	-0.015826	-0.001528	0.053931	0.025080	0.064521	0.036989	0.117410	0.045808	0.043623	0.058901	-0.055659	0.043158	-0.042811	-0.010033
2024-11-30	0.051707	-0.045297	0.128110	0.136059	0.114204	0.132546	0.081066	0.148602	0.175322	0.299681	0.125270	0.044192	0.124814	-0.026049	-0.010484
2024-12-31	0.055155	0.434237	-0.025896	-0.069786	-0.006773	0.014308	-0.000169	0.128623	-0.054412	-0.033673	-0.040085	-0.004629	0.010177	-0.025903	-0.067254

In [6]:

```python
rf = web.DataReader('F-F_Research_Data_Factors','famafrench', start, end)[0][['RF']].div(100)

rf.index = rf.index.to_timestamp(how='end').normalize()

# Align indices of returns_mon and rf['RF']
aligned_rf = rf.reindex(returns_mon.index, method='pad')
```

```
C:\Users\rishi\AppData\Local\Temp\ipykernel_31108\90602993.py:1: FutureWarning: The argument 'date_parser' is deprecated and will be removed in a future version. Please use 'date_format' instead, or read your data in a
s 'object' dtype and then call 'to_datetime'.
  rf = web.DataReader('F-F_Research_Data_Factors','famafrench', start, end)[0][['RF']].div(100)
C:\Users\rishi\AppData\Local\Temp\ipykernel_31108\90602993.py:1: FutureWarning: The argument 'date_parser' is deprecated and will be removed in a future version. Please use 'date_format' instead, or read your data in a
s 'object' dtype and then call 'to_datetime'.
  rf = web.DataReader('F-F_Research_Data_Factors','famafrench', start, end)[0][['RF']].div(100)
```

In [7]:

```python
# # Keep only the months that we have RF data from Ken French
# returns_mon = returns_mon[returns_mon.index<=rf.index[-1]]
# returns_mon.tail()
```

Optimal Historical Portfolio

In [8]:

```python
def port_ret(weights):
    """
    Portfolio Returns Function

    Parameters:
        weights (array): array of weights for each stock

    Returns:
        port_return (float): portfolio return value
    """
    port_ret = np.dot(returns_mon*12,weights).mean()
    return port_ret
```

In [9]:

```python
def port_std(weights):
    """
    Portfolio Standard Deviation Function

    Parameters:
        weights (array): array of weights for each stock

    Returns:
        port_std (float): portfolio std dev value
    """

    port_std = np.sqrt(np.dot(weights, np.dot(returns_mon.cov()*12, weights)))
    return port_std
```

In [10]:

```python
def ex_port_ret(weights):
    """
    Excess Portfolio Returns Function

    Parameters:
        weights (array): array of weights for each stock

    Returns:
        ex_port_ret (float): excess portfolio return value
    """
    ex_port_ret = (np.dot(returns_mon,weights) - aligned_rf['RF']).mean()*12
    return ex_port_ret
```

In [11]:

```python
def ex_port_std(weights):
    """
    Excess Portfolio Standard Deviation Function

    Parameters:
        weights (array): array of weights for each stock

    Returns:
        ex_port_std (float): excess portfolio std dev value
    """
    ex_port_std = (np.dot(returns_mon,weights) - aligned_rf['RF']).std()*np.sqrt(12)
    return ex_port_std
```

In [12]:

```python
def neg_SR(weights):
    """
    Sharpe Ratio Function

    Parameters:
        weights (array): array of weights for each stock

    Returns:
        SR (float): negative sharpe ratio value
    """

    SR = ex_port_ret(weights) / ex_port_std(weights)
    return (-1)*SR
```

In [13]:

```python
# by convention of minimize function it should be a function that returns zero for conditions
constraints = ({'type':'eq','fun': lambda weights: np.sum(weights) - 1})

# weights must be between 0 and 1 if short-selling not allowed
boundaries=[(0,1)]
bounds = tuple(boundaries * len(returns_mon.columns))

# initial guess (equally weighted)
init_guess = np.full(len(returns_mon.columns), 1/len(returns_mon.columns))
```

In [14]:

```python
# finding the optimal portfolio
optimal_port = minimize(neg_SR,init_guess,constraints=constraints)
optimal_port
```

Out[14]:

```
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: -1.9404704565604485
       x: [ 3.812e-01  3.622e-01 ... -1.053e-01 -9.993e-03]
     nit: 19
     jac: [-8.550e-02 -8.544e-02 ... -8.642e-02 -8.551e-02]
    nfev: 308
    njev: 19
```

In [15]:

```python
# fun (objective function) neg_SR = sharpe ratio that needs to be minimized
# for this portfolio the sharpe ratio of optimal portfolio is 1.9405. this means that this is a well-performing portfolio
# that generates significant excess returns relative to its risk

# x (array of optimal weights) = -ve due to no short selling constraints
# nit (number of iterations)

# jac (gradient (partial derivatives) of the objective function with respect to the weights at the optimal solution)
# It indicates how much the Sharpe Ratio would change with small adjustments to the portfolio weights

# nfev (number of times the objective function was evaluated)
# njev (number of times optimizer computed gradients to adjust weights)
```

In [16]:

```python
# weights of the stocks in the optimal portfolio
for stock in stock_list:
    print(stock + "  "+ str(round(optimal_port.x[stock_list.index(stock)],4)),end="  ")
```

```
AAPL  0.3812  AVGO  0.3622  AXP  0.5043  BAC  -0.5843  C  -0.5672  CRM  -0.3842  CSCO  -0.4251  DOCU  -0.005  GS  0.0567  HUBS  0.0199  JPM  0.946  MSFT  0.4075  NOW  0.4033  QCOM  -0.1053  TXN  -0.01
```

In [17]:

```python
# annualized average monthly return of the optimal portfolio
port_ret(optimal_port.x)
```

Out[17]: 0.5239130609845557

In [18]:

```python
# annualized monthly standard deviation of the optimal portfolio
port_std(optimal_port.x)
```

Out[18]: 0.2587574351555713

In [19]:

```python
# maximum Sharpe Ratio
-optimal_port.fun
```

Out[19]: 1.9404704565604485

Monte Carlo Simulation

In [20]:

```python
# simulating the portfolio performance from Jan 2025 to Dec 2030 using Monte Carlo
# initializing seed
np.random.seed(42)

# number of Monte Carlo simulations
num_simulations = 10000

# number of months 6 years (2025-2030)
num_future_months = 6 * 12

# initial amount of investment
inv_amt = 1000
```

In [21]:

```python
# average monthly returns
avg_returns = returns_mon.mean()
print("Monthly Average Returns:", avg_returns)

# covariance matrix of monthly returns
cov_matrix = returns_mon.cov()
print("Covariance Matrix: \n", cov_matrix)
```

```
Monthly Average Returns: Ticker
AAPL    0.027005
AVGO    0.035853
AXP     0.017949
BAC     0.010579
C       0.008354
CRM     0.017482
CSCO    0.008423
DOCU    0.020329
GS      0.016583
HUBS    0.031909
JPM     0.014952
MSFT    0.021164
NOW     0.026613
QCOM    0.021943
TXN     0.012168
dtype: float64
Covariance Matrix:
 Ticker      AAPL      AVGO       AXP       BAC         C       CRM      CSCO  \
Ticker
AAPL    0.007321  0.003451  0.002769  0.003149  0.003613  0.004712  0.002514
AVGO    0.003451  0.009487  0.002546  0.002424  0.003296  0.003449  0.002833
AXP     0.002769  0.002546  0.006614  0.005378  0.006381  0.003532  0.002744
BAC     0.003149  0.002424  0.005378  0.007824  0.007910  0.003284  0.002883
C       0.003613  0.003296  0.006381  0.007910  0.010341  0.004006  0.003228
CRM     0.004712  0.003449  0.003532  0.003284  0.004006  0.010008  0.001808
CSCO    0.002514  0.002833  0.002744  0.002883  0.003228  0.001808  0.005090
DOCU    0.004297  0.003207  0.002436  0.003260  0.003780  0.004904  0.002442
GS      0.003726  0.002554  0.005034  0.006761  0.007336  0.003280  0.003090
HUBS    0.004671  0.004256  0.006063  0.005805  0.006416  0.008348  0.003103
JPM     0.002266  0.002253  0.004380  0.006168  0.006513  0.002436  0.002330
MSFT    0.003446  0.002753  0.002009  0.002146  0.002540  0.003811  0.001732
NOW     0.003665  0.003241  0.002806  0.002985  0.003318  0.005802  0.002350
QCOM    0.005766  0.003960  0.003170  0.004174  0.005023  0.004010  0.001682
TXN     0.003090  0.003042  0.003052  0.003639  0.004304  0.003051  0.002159

Ticker      DOCU        GS      HUBS       JPM      MSFT       NOW      QCOM  \
Ticker
AAPL    0.004297  0.003726  0.004671  0.002266  0.003446  0.003665  0.005766
AVGO    0.003207  0.002554  0.004256  0.002253  0.002753  0.003241  0.003960
AXP     0.002436  0.005034  0.006063  0.004380  0.002009  0.002806  0.003170
BAC     0.003260  0.006761  0.005805  0.006168  0.002146  0.002985  0.004174
C       0.003780  0.007336  0.006416  0.006513  0.002540  0.003318  0.005023
CRM     0.004904  0.003280  0.008348  0.002436  0.003811  0.005802  0.004010
CSCO    0.002442  0.003090  0.003103  0.002330  0.001732  0.002350  0.001682
DOCU    0.020740  0.003344  0.008921  0.002428  0.002827  0.005386  0.004111
GS      0.003344  0.007885  0.006496  0.005662  0.002276  0.003337  0.003694
HUBS    0.008921  0.006496  0.017910  0.004743  0.004284  0.007071  0.005261
JPM     0.002428  0.005662  0.004743  0.005817  0.001674  0.002278  0.003622
MSFT    0.002827  0.002276  0.004284  0.001674  0.003724  0.003125  0.003772
NOW     0.005386  0.003337  0.007071  0.002278  0.003125  0.006873  0.002902
QCOM    0.004111  0.003694  0.005261  0.003622  0.003772  0.002902  0.013282
TXN     0.003493  0.003472  0.003888  0.002984  0.002149  0.001946  0.004607

Ticker       TXN
Ticker
AAPL    0.003090
AVGO    0.003042
AXP     0.003052
BAC     0.003639
C       0.004304
CRM     0.003051
CSCO    0.002159
DOCU    0.003493
GS      0.003472
HUBS    0.003888
JPM     0.002984
MSFT    0.002149
NOW     0.001946
QCOM    0.004607
TXN     0.004810
```

In [22]:

```python
# optimal weights of optimal historical portfolio
optimal_weights = optimal_port.x
optimal_weights
```

Out[22]:

```
array([ 0.38120663,  0.36223524,  0.50431569, -0.58431224, -0.56719437,
       -0.38418991, -0.42508533, -0.0050447 ,  0.05665578,  0.01991655,
        0.94597623,  0.40747505,  0.40330088, -0.10526248, -0.00999301])
```

In [23]:

```python
# creating an empty list to store the values from the simulation
sim_values = []

# running a monte carlo simulation
for _ in range(num_simulations):

    # randomizing monthly returns for six years
    sim_returns = np.random.multivariate_normal(avg_returns, cov_matrix, num_future_months)

    # calculating portfolio returns for each month
    portfolio_returns = np.dot(sim_returns, optimal_weights)

    # calculating portfolio value over time
    portfolio_value = inv_amt * np.cumprod(1 + portfolio_returns)

    # updating list by adding final value over six years
    sim_values.append(portfolio_value[-1])

# converting list to array
sim_values = np.array(sim_values)
print(sim_values)
```

```
[ 9604.22896545  8152.06897227 81220.35030951 ... 36101.46711378
 51095.07946834 14549.53829819]
```

In [24]:

```python
# finding mean
avg_sim = np.mean(sim_values)

# finding downside risk (10th percentile)
percentile_10 = np.percentile(sim_values, 10)

# finding upside potential (90th percentile)
percentile_90 = np.percentile(sim_values, 90)

# printing results
print(f"Average Portfolio Value (2030): ${avg_sim:,.2f}")
print(f"Downside Risk (10th Percentile): ${percentile_10:,.2f}")
print(f"Upside Potential (90th Percentile): ${percentile_90:,.2f}")
```

```
Average Portfolio Value (2030): $21,939.95
Downside Risk (10th Percentile): $8,303.68
Upside Potential (90th Percentile): $39,918.97
```

In [25]:

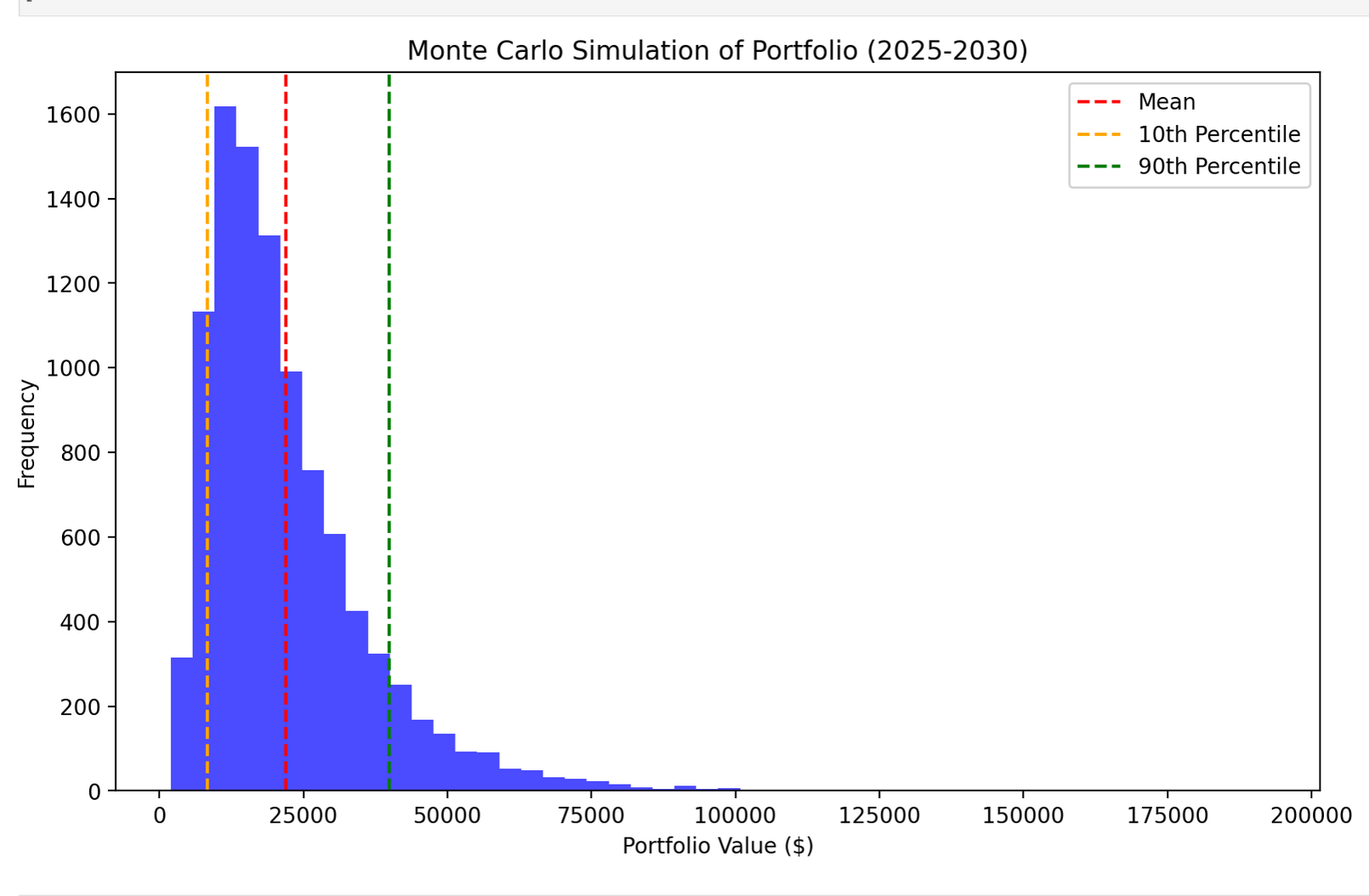
```python
# plotting the distribution of portfolio values
plt.figure(figsize=(10, 6))

# creating a histogram
plt.hist(sim_values, bins=50, color='blue', alpha=0.7)

# creating average & percentile lines
plt.axvline(avg_sim, color='red', linestyle='--', label='Mean')
plt.axvline(percentile_10, color='orange', linestyle='--', label='10th Percentile')
plt.axvline(percentile_90, color='green', linestyle='--', label='90th Percentile')

# adding plot title, axes labels, and legend
plt.title('Monte Carlo Simulation of Portfolio (2025-2030)')
plt.xlabel('Portfolio Value ($)')
plt.ylabel('Frequency')
plt.legend()

# displaying the graph
plt.show()
```



In [ ]:

60-Month Rolling Basis Optimization

In [26]:

```python
# randomizing monthly returns for six years
sim_future_returns = np.random.multivariate_normal(avg_returns, cov_matrix, num_future_months)

# converting simulated returns to a dataframe with dates
future_dates = pd.date_range(start="2025-01-31", periods=num_future_months, freq='ME')
sim_future_returns = pd.DataFrame(sim_future_returns, index=future_dates, columns=returns_mon.columns)

# combining historical and simulated data
combined_mon_returns = pd.concat([returns_mon, sim_future_returns])
combined_mon_returns
```

Out[26]:

Ticker	AAPL	AVGO	AXP	BAC	C	CRM	CSCO	DOCU	GS	HUBS	JPM	MSFT	NOW	QCOM	TXN
2018-04-30	0.018112	-0.014434	-0.008633	-0.007629	-0.010436	0.006405	-0.009394	-0.027687	-0.006130	-0.018081	-0.005667	-0.024004	0.018140	-0.001957	-0.010536
2018-05-31	0.135125	0.098727	-0.004557	-0.025446	-0.018521	0.068931	-0.035674	0.289412	-0.048887	0.144476	-0.016271	0.061467	0.069038	0.151289	0.109965
2018-06-30	-0.009418	-0.024502	-0.003052	-0.029270	0.003449	0.054666	0.007492	0.063040	-0.023508	0.034654	-0.026259	-0.002327	-0.028940	-0.034412	-0.014833
2018-07-31	0.027983	-0.086012	0.019156	0.095424	0.074268	0.005499	-0.009537	0.017941	0.076438	-0.010367	0.109162	0.075753	0.020235	0.142017	0.015296
2018-08-31	0.200422	-0.012355	0.064911	0.001619	-0.002795	0.113234	0.129582	0.158442	0.004917	0.157937	-0.003219	0.062992	0.115935	0.072086	0.009702
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2030-08-31	-0.045758	-0.041301	0.108891	0.144832	0.189205	-0.025287	0.005114	-0.117844	0.066969	-0.163973	0.108685	0.015817	-0.031849	-0.068327	-0.007843
2030-09-30	0.002052	-0.193448	-0.072993	-0.060173	-0.055971	0.035755	-0.055908	0.071530	-0.033158	0.082982	-0.049978	0.007973	0.035910	0.108039	-0.031236
2030-10-31	-0.025967	0.139970	0.044164	-0.017313	0.015149	0.214979	-0.036522	0.277805	-0.009061	0.352093	0.054291	0.060295	0.120580	-0.057369	0.025036
2030-11-30	-0.054509	-0.208296	-0.061716	-0.151959	-0.078845	0.018740	-0.059410	-0.233545	-0.059208	-0.081259	-0.050129	-0.027901	-0.052422	-0.020726	-0.048059
2030-12-31	0.090484	0.087214	0.139078	0.166335	0.187611	0.145139	0.040260	-0.211023	0.174032	0.087426	0.197061	0.131458	-0.003737	0.025108	0.064247

153 rows × 15 columns

In [27]:

```python
def optimize_portfolio(returns_subset):
    """
    Portfolio Optimization Function

    Parameters:
        returns_subset (df): dataframe of subset of returns
    """

    def neg_SR(weights):
        """
        Sharpe Ratio Function

        Parameters:
            weights (array): array of weights for each stock

        Returns:
            SR (float): negative sharpe ratio value
        """

        mean_ret = np.dot(returns_subset.mean(), weights) * 12
        std_dev = np.sqrt(np.dot(weights, np.dot(returns_subset.cov() * 12, weights)))
        return -mean_ret / std_dev

    # by convention of minimize function it should be a function that returns zero for conditions
    constraints = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1})

    # bounds for weights (no short-selling)
    bounds = [(0, 1) for _ in range(len(returns_subset.columns))]

    # initial guess (equally weighted)
    init_guess = np.full(len(returns_subset.columns), 1 / len(returns_subset.columns))

    # optimizing portfolio
    result = minimize(neg_SR, init_guess, method='SLSQP', bounds=bounds, constraints=constraints)

    # getting optimal weights
    return result.x
```

In [28]:

```python
# performing rolling optimization

# 60 months
rolling_window = 60

# creating empty list to store weights
optimal_weights_rolling = []

# creating empty list to store the end dates of rolling period
rolling_end_dates = []

for i in range(len(combined_mon_returns) - rolling_window + 1):

    # getting a 60-month rolling period return
    window_returns = combined_mon_returns.iloc[i:i+rolling_window]

    # optimizing portfolio weights for this period
    weights = optimize_portfolio(window_returns)

    # storing the weights and the end date of the rolling period
    optimal_weights_rolling.append(weights)
    rolling_end_dates.append(window_returns.index[-1])

# creating a dataframe of rolling weights
optimal_weights_df = pd.DataFrame(optimal_weights_rolling, index=rolling_end_dates, columns=combined_mon_returns.columns)

optimal_weights_df.head()
```

Out[28]:

Ticker	AAPL	AVGO	AXP	BAC	C	CRM	CSCO	DOCU	GS	HUBS	JPM	MSFT	NOW	QCOM	TXN
2023-03-31	0.235539	0.230521	1.577921e-16	0.000000e+00	1.523169e-16	2.198355e-16	2.331035e-16	1.866183e-16	2.707659e-16	0.056132	2.413705e-17	0.461560	1.624800e-02	3.955441e-16	0.000000e+00
2023-04-30	0.178418	0.180596	0.000000e+00	4.463660e-16	0.000000e+00	4.855057e-16	1.806281e-16	1.307548e-16	1.579683e-16	0.039489	0.000000e+00	0.601497	0.000000e+00	0.000000e+00	1.013729e-16
2023-05-31	0.134567	0.225100	0.000000e+00	0.000000e+00	0.000000e+00	2.046974e-16	3.330669e-16	2.896988e-16	7.632783e-17	0.041319	0.000000e+00	0.599015	1.170938e-17	1.279359e-16	0.000000e+00
2023-06-30	0.155057	0.249790	0.000000e+00	1.781344e-16	0.000000e+00	1.298874e-15	2.535949e-16	0.000000e+00	0.000000e+00	0.014294	0.000000e+00	0.550963	2.989539e-02	6.120321e-16	2.673643e-16
2023-07-31	0.197691	0.365216	0.000000e+00	0.000000e+00	2.267744e-16	0.000000e+00	2.226864e-16	0.000000e+00	1.185236e-16	0.047896	5.801837e-17	0.344145	4.505104e-02	1.306667e-16	0.000000e+00

In [29]:

```python
# combining monthly returns from 2000 to 2030
aligned_mon_returns = combined_mon_returns.loc['2000-01-01':'2030-12-31']

# expanding optimal_weights_df to match the full date range
expanded_weights = optimal_weights_df.reindex(aligned_mon_returns.index).fillna(method='bfill').fillna(method='ffill')

# calculating portfolio returns using rolling weights
projected_returns = (aligned_mon_returns * expanded_weights.shift(1).fillna(method='bfill').values).sum(axis=1)

# cumulative portfolio value starting from $100
cumulative_portfolio_value = 100 * (1 + projected_returns).cumprod()
```

```
C:\Users\rishi\AppData\Local\Temp\ipykernel_31108\3409124356.py:5: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
  expanded_weights = optimal_weights_df.reindex(aligned_mon_returns.index).fillna(method='bfill').fillna(method='ffill')
C:\Users\rishi\AppData\Local\Temp\ipykernel_31108\3409124356.py:8: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
  projected_returns = (aligned_mon_returns * expanded_weights.shift(1).fillna(method='bfill').values).sum(axis=1)
```

In [30]:

```python
#cumulative_portfolio_value.index[0]
```

In [31]:

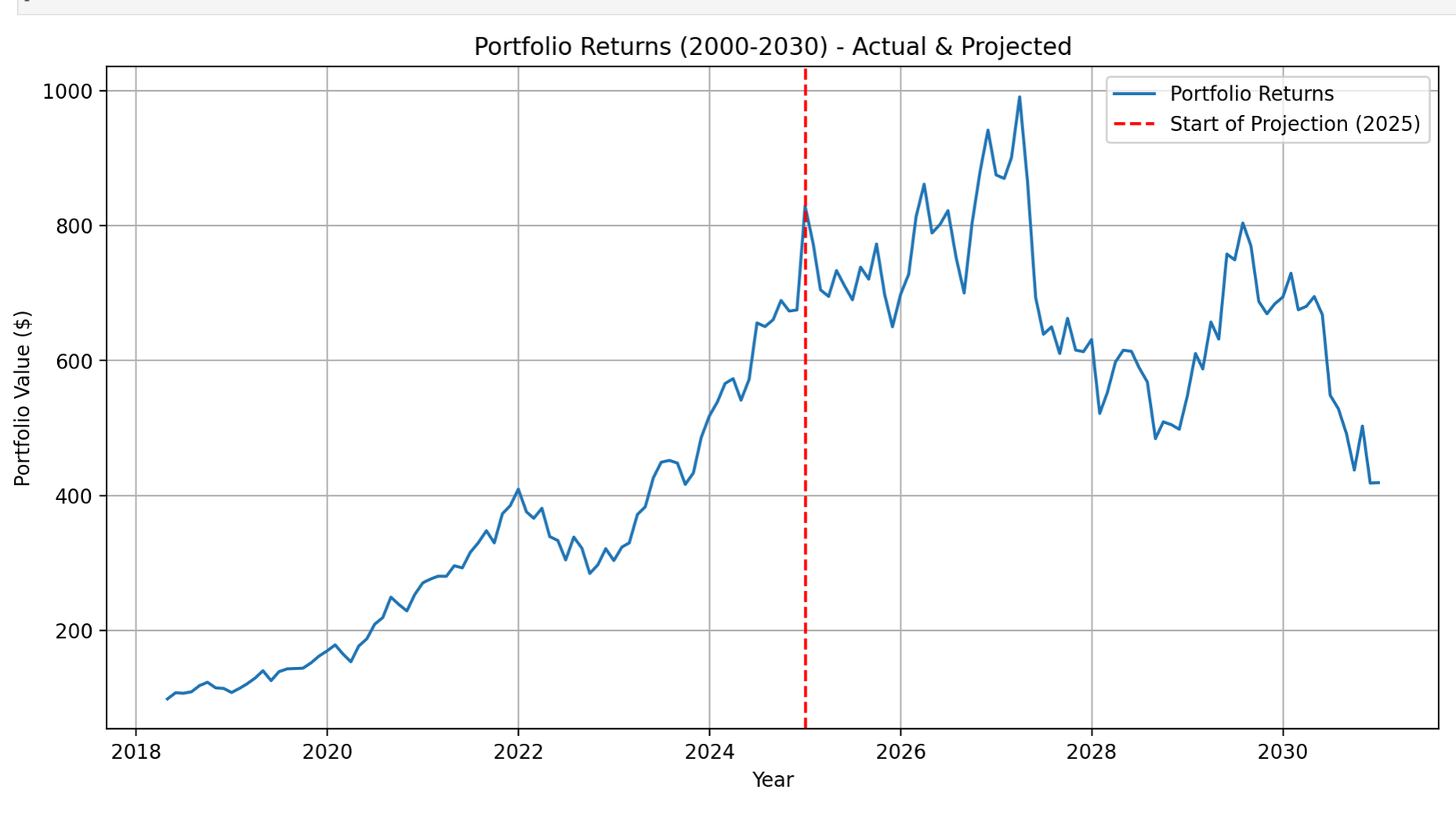
```python
# plotting a graph to show cumulative portfolio value
plt.figure(figsize=(12, 6))

# making a line graph
plt.plot(cumulative_portfolio_value.index, cumulative_portfolio_value, label='Portfolio Returns')

# adding a line to show where projected values begin
plt.axvline(pd.Timestamp("2025-01-01"), color="red", linestyle="--", label="Start of Projection (2025)")

# adding title, axes labels, legend, and grid
plt.title('Portfolio Returns (2000-2030) - Actual & Projected')
plt.xlabel('Year')
plt.ylabel('Portfolio Value ($)')
#plt.xlim(pd.Timestamp('2016-01-01'), pd.Timestamp('2031-12-31'))  # Use pd.Timestamp for x-axis limits
plt.legend()
plt.grid()

# displaying graph
plt.show()
```



In [ ]:

In [32]:

```python
# cumulative return for all stocks
cumulative_returns_all = (1 + aligned_mon_returns * expanded_weights.shift(1).bfill()).cumprod()
```

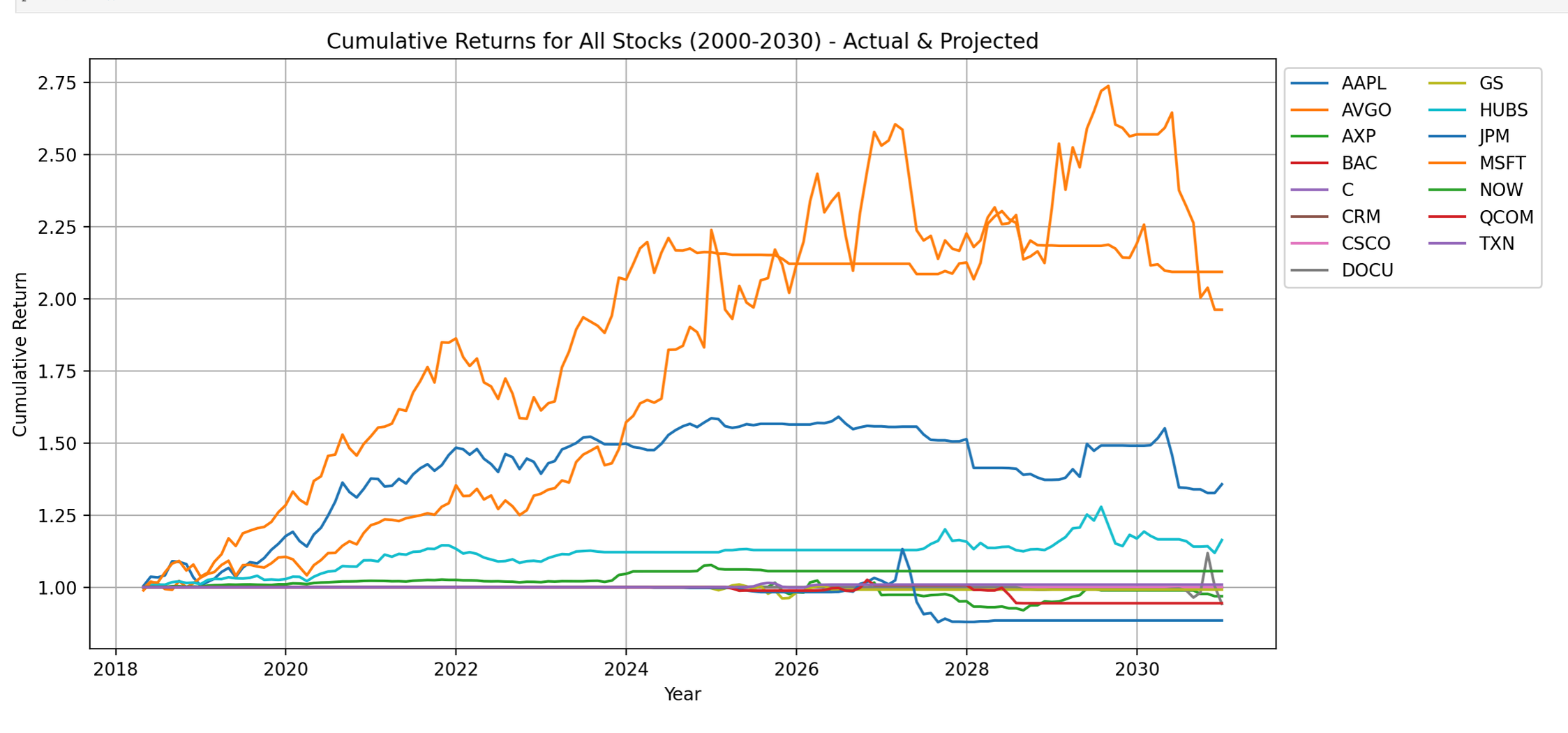
In [33]:

```python
# plotting cumulative return for the entire timeframe (2000-2030)
plt.figure(figsize=(12, 6))

# making line for each stock
for stock in cumulative_returns_all.columns:
    plt.plot(cumulative_returns_all.index, cumulative_returns_all[stock], label=stock)

# adding title, axes labels, legend, and grid
plt.title("Cumulative Returns for All Stocks (2000-2030) - Actual & Projected")
plt.xlabel("Year")
plt.ylabel("Cumulative Return")
plt.legend(loc="upper left", bbox_to_anchor=(1, 1), ncol=2)
plt.grid()

# displaying the graph
plt.show()
```



In [ ]:

In [34]:

```python
# calculating the cumulative return at the last date for all stocks
final_cumulative_returns = cumulative_returns_all.iloc[-1]

# identifying the best and worst performing portfolios
best_performing_portfolio = final_cumulative_returns.idxmax()
best_performing_return = final_cumulative_returns.max()

worst_performing_portfolio = final_cumulative_returns.idxmin()
worst_performing_return = final_cumulative_returns.min()

# printing results
print(f"Best Performing Portfolio: {best_performing_portfolio} with Cumulative Return: {best_performing_return:.2f}")
print(f"Worst Performing Portfolio: {worst_performing_portfolio} with Cumulative Return: {worst_performing_return:.2f}")
```

```
Best Performing Portfolio: MSFT with Cumulative Return: 2.09
Worst Performing Portfolio: JPM with Cumulative Return: 0.88
```

In [ ]:

In [ ]: