

# Portfolio Analysis & Optimization

Akshit Chopra

## KEY DEFINITION

1. **Sharpe ratio:** It's a financial metric used to measure the investment risk adjusted return by assessing how much excess return an investment generates per unit of risk.
  2. **Track Draw Down:** It is a systematic measure of the decline in the value of investment from its highest point to a subsequent lowest point before a new peak is reached
  3. **CAGR Compounded Annual Growth Rate:** Average annual rate over a period of time
  4. **Cumulative return:** It is the total return percentage of gain or loss realized on an investment over a specified period
- 

## 1. INTRODUCTION

This project aims to conduct a comprehensive analysis and optimization of an investment portfolio. The core idea is to get the best performing portfolio from the initial portfolio to the optimized portfolio derived from a broader selection of stocks.

---

## 2. METHODOLOGY

The analysis utilized a quantitative approach by sourcing one year of historical Adjusted Closing Price data for all selected stocks from Yahoo Finance.

Data Processing:

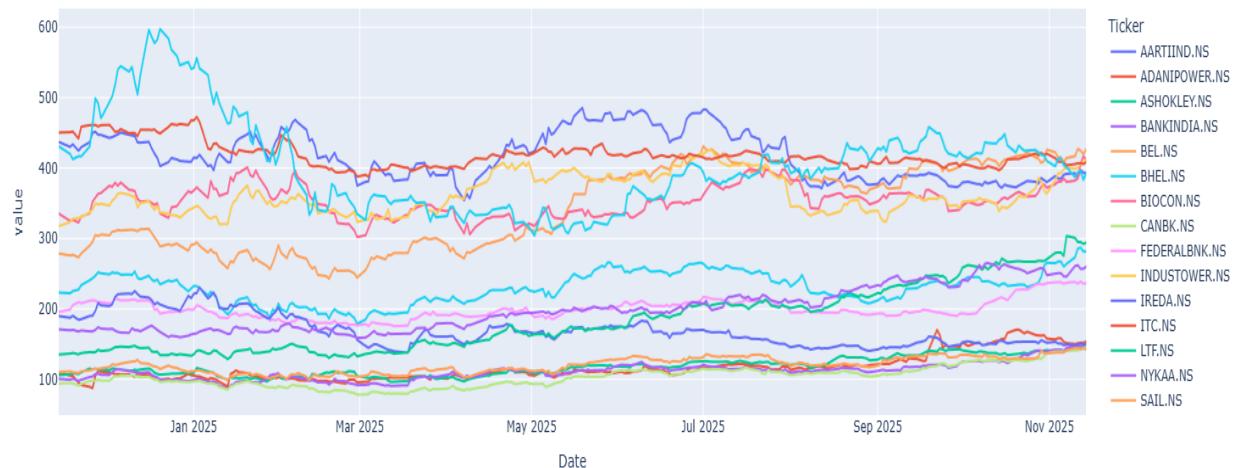
- Returns Calculation: Logarithmic returns were calculated for accurate analysis.
- Volatility: Daily and annualized standard deviation of returns (volatility) were computed.
- Risk-Adjusted Metrics: The Sharpe Ratio was calculated using a risk-free rate of 6.53% (as of November 2025).

- Performance Metrics: Cumulative Return and Compound Annual Growth Rate (CAGR) were determined for all assets.
  - Drawdown: Max Drawdown was tracked to measure maximum peak-to-trough decline.
- 

## My Portfolio snippet Code

```
from IPython.display import display
import pandas as pd
import yfinance as yf
import numpy as np
import plotly.graph_objects as go
import plotly.express as px
from scipy.optimize import minimize

portfolio = ["CANBK.NS", "BEL.NS", "ADANIPOWER.NS", "ITC.NS", "ASHOKLEY.NS", "LTF.NS", "BHEL.NS", "BIOCON.NS", "SAIL.NS",
"BANKINDIA.NS", "IREDA.NS", "INDUSTOWER.NS", "SWIGGY.NS", "NYKAA.NS", "FEDERALBNK.NS", "AARTIIND.NS"]
df_port = yf.download(portfolio, period="1y", auto_adjust=False)[["Adj Close"]]
df_port.head(5)
px.line(df_port).show()
```




---

### **3. OBJECTIVE**

- Calculate Performance Indicators: Determine the Sharpe Ratio and Track Drawdown for both the pre- and post-optimization portfolios to assess risk-adjusted returns and capital preservation.
  - Calculate Portfolio Averages: Compute the average CAGR and Cumulative Return to measure overall growth.
  - Calculate Risk Metrics: Quantify the Daily Volatility and Annual Volatility of the portfolio before and after optimization.
  - Optimize: Apply the Markowitz Model to construct an Efficient Frontier to identify optimal portfolio weights.
- 

### **4. OPTIMIZATION PROCESS**

The portfolio optimization was executed using Modern Portfolio Theory (MPT), specifically the Markowitz Model, which seeks to minimize portfolio risk for a given level of expected return.

#### **Stock Selection and Filtering:**

- A broad universe of additional Indian stocks was collected and appended to the initial portfolio.
- Filtering Criterion: A rigorous filtering step was applied to refine this universe. Stocks with a Sharpe Ratio below a benchmark of 0.99 were systematically removed from the portfolio. This step ensured that only stocks demonstrating superior risk-adjusted returns were included in the final optimization.

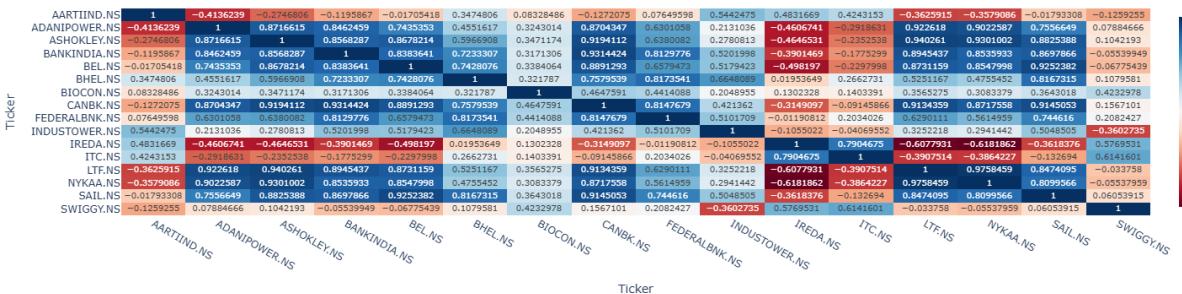
## Markowitz Model Application:

- The annualized expected returns and covariance matrix were calculated for the remaining, filtered stocks.
- The (scipy.optimize.minimize) function with the SLSQP method was used to solve the optimization problem, subject to two main constraints:
  - The sum of all asset weights must equal 1 ( $\sum w_i = 1$ ).
  - The portfolio's expected return must equal a specific target return ( $R_p = R_{\text{target}}$ ).
- By iterating this process across a range of target returns, the Efficient Frontier was mapped, representing the set of optimal portfolios.

## 5. RESULTS

Metric	Pre-Optimization	Post-Optimization
<b>Daily Volatility (%)</b>	2.06%	1.8%
<b>Annual Volatility (%)</b>	32.55%	29.38%
<b>Average Cumulative Return (%)</b>	30.11%	61.14%
<b>Average CAGR (%)</b>	0.09%	0.18%
<b>Average Sharpe Ratio</b>	0.53	1.37
<b>Average Max Drawdown (%)</b>	-24.04%	-18.79%

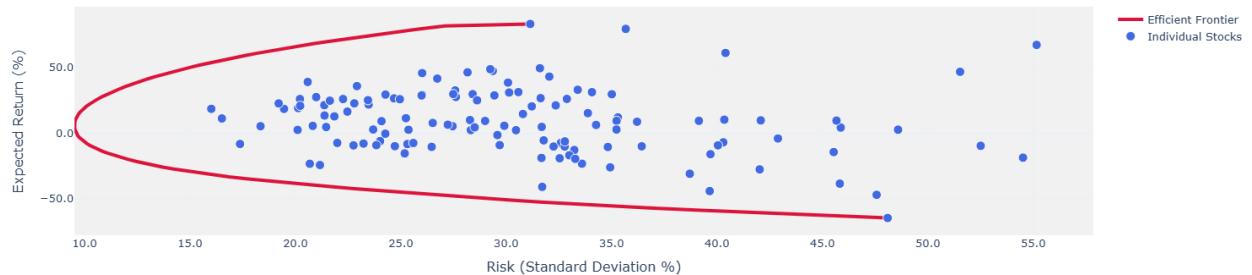
Pre Optimization Correlation Matrix



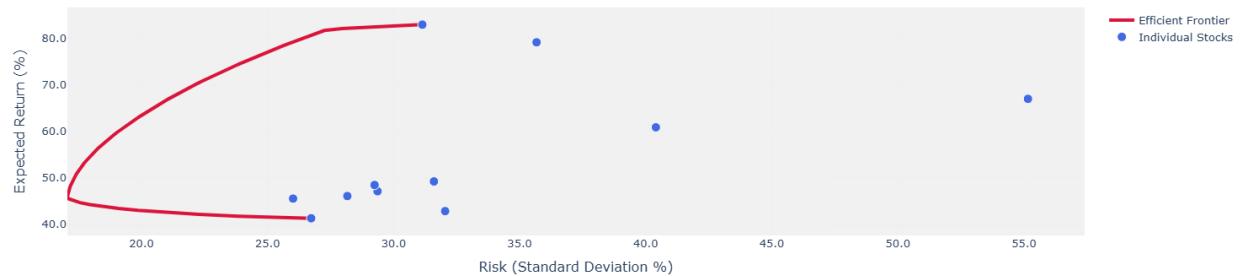
Post Optimization Correlation Matrix

Ticker	AUBANK.NS	BANKINDIA.NS	BEL.NS	CANBK.NS	EICHERMOT.NS	INDIGO.NS	LAURUSLABS.NS	LTF.NS	MARUTI.NS	NYKAA.NS	PAYTM.NS	UPL.NS	AUBANK.NS	BANKINDIA.NS	BEL.NS	CANBK.NS	EICHERMOT.NS	INDIGO.NS	LAURUSLABS.NS	LTF.NS	MARUTI.NS	NYKAA.NS	PAYTM.NS	UPL.NS	
	<b>1</b>	0.8759975	<b>0.9276504</b>	0.89408	0.7229169	0.8161083	0.8092007	0.8785191	0.6778542	0.8474887	0.7134307	0.693861													
	0.8759975	<b>1</b>	0.8383641	<b>0.9314424</b>	0.7877157	0.7257048	0.7865973	0.8945437	0.7256178	0.8535933	0.7681079	0.6539796													
	0.9276504	0.8383641	<b>1</b>	0.8891293	0.7300101	0.7880358	<b>1</b>	0.7111641	0.883879	0.9369493	0.9528147	<b>0.9407514</b>	0.8292522	0.7093396											
	0.89408	<b>0.9314424</b>	0.8891293	<b>1</b>	0.7880358	0.6908676	0.8349776	<b>0.9134359</b>	0.7476627	0.8717558	0.8530063	<b>0.5536213</b>													
	0.7229169	0.7877157	0.7300101	0.7880358	<b>1</b>	0.7111641	0.8062027	<b>1</b>	0.8062027	0.8092436	<b>0.9452344</b>	<b>1</b>	0.8999056	<b>0.9758459</b>	0.8740728	0.7705329									
	0.8161083	0.7257048	0.8707892	0.6908676	0.7111641	<b>1</b>	0.8464688	<b>0.9452344</b>	0.8092436	0.6208696	0.8210676	0.6531258	0.8427738												
	0.8092007	0.7865973	0.8170959	0.8349776	0.883879	0.8062027	<b>1</b>	0.9452344	0.6208696	0.8210676	0.9262432	0.8955849	0.7966754												
	0.8785191	0.8945437	0.8731159	<b>0.9134359</b>	0.9369493	0.8092436	<b>0.9452344</b>	<b>1</b>	0.8464688	0.9262432	0.9758459	0.8758391	<b>0.5572285</b>												
	0.6778542	0.7256178	0.6675689	0.7476627	0.9528147	<b>0.6208696</b>	0.8464688	0.8999056	<b>1</b>	0.9108668	0.8758391	<b>0.5572285</b>													
	0.8474887	0.8535933	0.8547998	0.8717558	0.9407514	0.8210676	<b>0.9262432</b>	<b>0.9758459</b>	0.9108668	<b>1</b>	0.8758391	<b>0.5572285</b>													
	0.7134307	0.7681079	0.7298625	0.8530063	0.8292522	0.6531258	0.8955849	0.8740728	0.8016849	0.8758391	<b>1</b>	<b>0.5572285</b>													
	0.693861	0.6539795	0.6717049	<b>0.5536213</b>	0.7093396	0.8427738	0.7968754	0.7705329	0.678333	0.7729419	0.5572285	<b>1</b>													
	AUBANK.NS	BANKINDIA.NS	BEL.NS	CANBK.NS	EICHERMOT.NS	INDIGO.NS	LAURUSLABS.NS	LTF.NS	MARUTI.NS	NYKAA.NS	PAYTM.NS	UPL.NS	AUBANK.NS	BANKINDIA.NS	BEL.NS	CANBK.NS	EICHERMOT.NS	INDIGO.NS	LAURUSLABS.NS	LTF.NS	MARUTI.NS	NYKAA.NS	PAYTM.NS	UPL.NS	

Pre optimization Efficient Frontier with Individual Stocks



Post optimization Efficient Frontier with Individual Stocks



Portfolio Sharpe Ratios



## Conclusion

The project successfully demonstrated the significant value added by applying quantitative portfolio optimization techniques. By employing the Markowitz Model within the framework of Modern Portfolio Theory.

- The Average Sharpe Ratio increased from 0.53 to 1.37, demonstrated higher excess return for every unit of risk taken.
- Annual Volatility decreasing from 32.55% to 29.38%, and Max Drawdown improving from -24.04% to -18.79%.

## Code Snippet

```
from IPython.display import display
import pandas as pd
import yfinance as yf
import numpy as np
import plotly.graph_objects as go
import plotly.express as px
from scipy.optimize import minimize

portfolio = ["CANBK.NS", "BEL.NS", "ADANIPOWER.NS", "ITC.NS", "ASHOKLEY.NS", "LTF.NS", "BHEL.NS", "BIOCON.NS", "SAIL.NS",
"BANKINDIA.NS", "IREDA.NS", "INDUSTOWER.NS", "SWIGGY.NS", "NYKAA.NS", "FEDERALBNK.NS", "AARTIIND.NS"]
df_port = yf.download(portfolio, period="1y", auto_adjust=False)[["Adj Close"]]
df_port.head(5)
px.line(df_port).show()

#Analysis risk and return
```

```

def portfolio_analysis(df_port, risk_free_rate):
    # Returns
    log_returns = np.log(df_port / df_port.shift(1)).dropna()

    # Daily volatility (decimal)
    daily_volatility = log_returns.std()
    port_daily_volatility = daily_volatility.mean()
    percentage_port_daily_volatility = port_daily_volatility * 100

    # Annual volatility (decimal)
    annual_volatility = daily_volatility * np.sqrt(df_port.shape[0])
    port_annual_volatility = annual_volatility.mean()
    percentage_port_annual_volatility = port_annual_volatility * 100

    # CAGR (percent)
    cagr_annual_return = ((df_port.iloc[-1] / df_port.iloc[0])***(1/len(df_port)) - 1) * 100

    # Cumulative return (decimal)
    cumulative_return = (df_port.iloc[-1] - df_port.iloc[0]) / df_port.iloc[0]
    port_cumulative_return = cumulative_return.mean()
    percentage_cumulative_return = cumulative_return * 100
    percentage_port_cumulative_return = port_cumulative_return * 100

    # Sharpe Ratio
    risk_free_rate = risk_free_rate
    portfolio_return_mean_ = log_returns.mean() * df_port.shape[0] - risk_free_rate
    sharpe_ratio = portfolio_return_mean_ / annual_volatility

    # Correlation matrix
    corr = df_port.corr()

    # Drawdown
    rolling_max = df_port.cummax()
    drawdown = (df_port - rolling_max) / rolling_max
    max_drawdown = drawdown.min() * 100

    show_graph=px.line(df_port).show()
    corr_graph = px.imshow(corr, text_auto=True, aspect="auto", color_continuous_scale='RdBu', title=' Post Optimization Correlation Matrix').show()
    return print(f"daily_volatility of the portfolio:\n{percentage_port_daily_volatility}\n"), print(f"Annual volatility of the portfolio:\n{percentage_port_annual_volatility}\n"), print(f"Average cumulative_returnin percentage:\n{percentage_port_cumulative_return}\n"), print(f"Average CAGR in percentage:\n{cagr_annual_return.mean()}\n"), print(f"Average Sharpe Ratio:\n{sharpe_ratio.mean()}\n"), print(f"Average Max Drawdown:\n{max_drawdown.mean()}\n"), print(f"sharpe ratio for each stock {sharpe_ratio}"), print(f"Max Draw Down for eahx stock{max_drawdown}"), corr_graph, print(f"Cumulative return for each stock in the portfolio in percentage:{display(percentage_cumulative_return)}\n"),show_graph

```

```

#portfolio optimization

more_stocks = ["IDFCFIRSTB.NS", "PNB.NS", "AUBANK.NS", "HDFCBANK.NS", "ICICIBANK.NS", "BAJAJFINSV.NS", "CIPLA.NS",
"DREDDY.NS", "IPCALAB.NS", "LUPIN.NS", "ZYDUSLIFE.NS", "SBICARD.NS", "ASIANPAINT.NS", "HINDUNILVR.NS", "TCS.NS",
"INFY.NS", "LT.NS", "MARUTI.NS", "SUNPHARMA.NS", "WIPRO.NS", "ADANIGREEN.NS", "ADANIPOWER.NS", "DLF.NS", "DMART.NS",
"DRREDDY.NS", "EIDPARRY.NS",
"EICHERMOT.NS", "EIHOTEL.NS", "ELGIEQUIP.NS", "EMAMILTD.NS", "FEDERALBNK.NS",
"GAIL.NS", "GLAND.NS", "GLAXO.NS", "GLENMARK.NS", "GNFC.NS",
"GODREJAGRO.NS", "GODREJCP.NS", "GODREJPROP.NS", "GRANULES.NS", "GRASIM.NS",
"HAL.NS", "HAVELLS.NS", "HCLTECH.NS", "HDFCAMC.NS", "HDFCBANK.NS",
"HEROMOTOCO.NS", "HINDALCO.NS", "HINDUNILVR.NS", "ICICIGI.NS", "ICICIBANK.NS",
"IEX.NS", "INDHOTEL.NS", "INDIGO.NS", "INDOWIND.NS", "INDUSTOWER.NS",
"INDUSINDBK.NS", "INFY.NS", "INOXGREEN.NS", "INOXWIND.NS",
"IOC.NS", "IPCALAB.NS", "IRCTC.NS", "IRFC.NS", "ITC.NS",
"JINDALSTEL.NS", "JSWENERGY.NS", "JSWSTEEL.NS", "KOTAKBANK.NS", "KPIGREEN.NS",
"LAURUSLABS.NS", "LTIM.NS", "LT.NS", "LUPIN.NS", "M&M.NS",
"MANKIND.NS", "MARICO.NS", "MARUTI.NS", "MAXHEALTH.NS",
"MOREPENLAB.NS", "MOTHERSON.NS", "MPHASIS.NS", "NATCOPHARM.NS", "NAUKRI.NS",
"NESTLEIND.NS", "NHPC.NS", "NMDC.NS", "NTPC.NS",
"OBEROIRLTY.NS", "OLECTRA.NS", "ONGC.NS", "ORCHPHARMA.NS",
"PAGEIND.NS", "PAYTM.NS", "PFC.NS", "PFIZER.NS", "PNB.NS",
"PNBHOUSING.NS", "POLICYBZR.NS", "POWERGRID.NS", "RECLTD.NS", "RELAXO.NS",
"RELIANCE.NS", "SANOFI.NS", "SBIN.NS", "SBILIFE.NS",
"SHREECEM.NS", "SIEMENS.NS", "SUNPHARMA.NS",
"SUZLON.NS", "SWSOLAR.NS", "TATACONSUM.NS", "TATAPOWER.NS", "TATASTEEL.NS",
"TCS.NS", "TECHM.NS", "TITAN.NS", "TORNTPHARM.NS", "TORNTPOWER.NS",
"TRENT.NS", "UPL.NS", "VBL.NS", "VEDL.NS",
"VGUARD.NS", "WESTLIFE.NS",
"WHIRLPOOL.NS", "WIPRO.NS", "YESBANK.NS", "ZYDUSLIFE.NS"]
portfolio.extend(more_stocks)

df_port = yf.download(portfolio, period="1y", auto_adjust=False)[ "Adj Close"]

```

```

# Markowitz portfolio model

tickers = df_port.columns.tolist()
num_assets = len(tickers)

returns = df_port.pct_change().dropna()
mean_returns = returns.mean() * len(df_port)           # Annualized expected return
std_returns = returns.std() * np.sqrt(len(df_port))   # Annualized risk (std dev)
cov_matrix = returns.cov() * len(df_port)             # Annualized covariance matrix

# -----
# Portfolio functions
# -----
def portfolio_return(weights):
    return np.sum(weights * mean_returns)

```

```

def portfolio_variance(weights):
    return np.dot(weights.T, np.dot(cov_matrix, weights))

bounds = tuple((0, 1) for _ in range(num_assets))

target_returns = np.linspace(mean_returns.min(), mean_returns.max(), 100)
risks = []
weights_record = []

for r in target_returns:
    constraints = (
        {'type': 'eq', 'fun': lambda x: np.sum(x) - 1},
        {'type': 'eq', 'fun': lambda x: portfolio_return(x) - r}
    )
    result = minimize(portfolio_variance, num_assets * [1. / num_assets],
                      method='SLSQP', bounds=bounds, constraints=constraints)
    if result.success:
        risks.append(np.sqrt(result.fun))
        weights_record.append(result.x)

# -----
# Convert to percentage form
# -----
risks_percent = np.array(risks) * 100
returns_percent = np.array(target_returns) * 100

# -----
# DataFrame for individual stocks
# -----
stock_df = pd.DataFrame({
    'Ticker': [ticker.replace('.NS', '') for ticker in tickers],
    'Expected Return (%)': (mean_returns * 100).round(2),
    'Risk (Volatility %)': (std_returns * 100).round(2)
})

# -----
# ⑤ Plotly Visualization
# -----
fig = go.Figure()

# Efficient Frontier
fig.add_trace(go.Scatter(
    x=risks_percent, y=returns_percent,
    mode='lines',
    name='Efficient Frontier',
    line=dict(color='crimson', width=4, dash='solid'),
    hovertemplate='<b>Risk:</b> %{x:.2f}%<br><b>Return:</b> %{y:.2f}%' + extra
))

# Individual Stocks
fig.add_trace(go.Scatter(

```

```

x=std_returns * 100, y=mean_returns * 100,
mode='markers+text',
textposition='top center',
marker=dict(size=11, color='royalblue', line=dict(width=1, color='white')),
name='Individual Stocks',
hovertemplate='%{text}  
Risk: %{x:.2f}  
Return: %{y:.2f}  
extra></extra>'  

))  
  

# -----  

# Layout Aesthetic Improvements  

# -----  

fig.update_layout(  

    title=dict(text='<b> Pre optimization Efficient Frontier with Individual Stocks</b>', x=0.5, font=dict(size=22)),  

```

```

sharpe_ratio = portfolio_return_mean_ / annual_volatility

# Combined filter using & operator
filtered = sharpe_ratio < benchmark_sharpe

# Extract ticker names
optimized_stocks = list(filtered[filtered].index)

# Remove these from portfolio
portfolio = [stock for stock in portfolio if stock not in optimized_stocks]

# Download updated portfolio data
df_port = yf.download(portfolio, period="1y", auto_adjust=False)[["Adj Close"]]

print("Removed:", optimized_stocks)
print("Updated & Downloaded portfolio:", portfolio)
portfolio_analysis(df_port, risk_free_rate=0.0653) # as per 16 November 2025 the risk free rate in india is 6.53%

# Markowitz portfolio model

tickers = df_port.columns.tolist()
num_assets = len(tickers)

returns = df_port.pct_change().dropna()
mean_returns = returns.mean() * len(df_port)           # Annualized expected return
std_returns = returns.std() * np.sqrt(len(df_port))   # Annualized risk (std dev)
cov_matrix = returns.cov() * len(df_port)             # Annualized covariance matrix

# -----
# Portfolio functions
# -----
def portfolio_return(weights):
    return np.sum(weights * mean_returns)

def portfolio_variance(weights):
    return np.dot(weights.T, np.dot(cov_matrix, weights))

bounds = tuple((0, 1) for _ in range(num_assets))

target_returns = np.linspace(mean_returns.min(), mean_returns.max(), 100)
risks = []
weights_record = []

for r in target_returns:
    constraints = (
        {'type': 'eq', 'fun': lambda x: np.sum(x) - 1},
        {'type': 'eq', 'fun': lambda x: portfolio_return(x) - r}
    )
    result = minimize(portfolio_variance, num_assets * [1. / num_assets],
                      method='SLSQP', bounds=bounds, constraints=constraints)

```

```

if result.success:
    risks.append(np.sqrt(result.fun))
    weights_record.append(result.x)

# -----
# Convert to percentage form
# -----
risks_percent = np.array(risks) * 100
returns_percent = np.array(target_returns) * 100

# -----
# DataFrame for individual stocks
# -----
stock_df = pd.DataFrame({
    'Ticker': [ticker.replace('.NS', '') for ticker in tickers],
    'Expected Return (%)': (mean_returns * 100).round(2),
    'Risk (Volatility %)': (std_returns * 100).round(2)
})

# -----
# ⑤Plotly Visualization
# -----
fig = go.Figure()

# Efficient Frontier
fig.add_trace(go.Scatter(
    x=risks_percent, y=returns_percent,
    mode='lines',
    name='Efficient Frontier',
    line=dict(color='crimson', width=4, dash='solid'),
    hovertemplate='<b>Risk:</b> %{x:.2f}%<br><b>Return:</b> %{y:.2f}%' 
))
    )) 

# Individual Stocks
fig.add_trace(go.Scatter(
    x=std_returns * 100, y=mean_returns * 100,
    mode='markers+text',
    textposition='top center',
    marker=dict(size=11, color='royalblue', line=dict(width=1, color='white')),
    name='Individual Stocks',
    hovertemplate='<b>%{text}</b><br>Risk: %{x:.2f}%<br>Return: %{y:.2f}%' 
))
    )) 

# -----
# Layout Aesthetic Improvements
# -----
fig.update_layout(
    title=dict(text='<b>Post optimization Efficient Frontier with Individual Stocks</b>', x=0.5, font=dict(size=22)),
    xaxis=dict(title='Risk (Standard Deviation %)', tickformat='.1f'),
    yaxis=dict(title='Expected Return (%)', tickformat='.1f'),
    template='plotly_white',
)

```

```
    plot_bgcolor='rgba(240,240,240,0.95)',  
    paper_bgcolor='rgba(255,255,255,1)',  
    font=dict(family='Verdana', size=13),  
    hoverlabel=dict(bgcolor='white', font_size=13)  
)  
  
fig.show()  
  
# -----  
# Styled DataFrame Display  
# -----  
stock_df.style.background_gradient(cmap='RdYlGn', subset=['Expected Return (%)']) \  
    .background_gradient(cmap='Blues', subset=['Risk (Volatility %)'])
```