Risk Analysis for Top 10 Holdings Daily Prices in the Portfolio

```
import pandas as pd
import numpy as np
from scipy.stats import norm
import yfinance as yf
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Loading the dataset
file_name = "Top 10 Holdings Daily Prices.csv"
data = pd.read_csv(file_name, index_col = 0)
```

```
# Convert index to datetime objects, handling potential errors by setting invalid dates to NaT (Not a Time)
data.index = pd.to_datetime(data.index, format="%m/%d/%Y", errors="coerce")

# Remove rows with NaT (invalid) dates in the index
if data.index.isnull().any():
    data = data[data.index.notnull()]

# Sort the DataFrame by date (ascending)
data = data.sort_index()

# Calculate daily returns for each asset, skipping the first row (NaN due to pct_change)
returns = data.pct_change(fill_method=None).dropna()

# Calculate the daily volatility (standard deviation) of each asset's returns
sigma = returns.std()

# Define the start and end dates for the analysis based on available data
start_date = returns.index.min()
end_date = returns.index.max()

# Format the start and end dates as strings for yfinance
start_date_str = start_date.strftime('%Y-%m-%d')
end_date_str = end_date.strftime('%Y-%m-%d')

# Download historical data for the S&P 500 (^GSPC) from Yahoo Finance within the specified date range
sp500_data = yf.download("^GSPC", start=start_date_str, end=end_date_str)


# Check if the S&P 500 data was downloaded successfully.  Raise an error if not.
if sp500_data.empty:
    raise ValueError("Failed to download ^GSPC data. Please check the date range and internet connection.")

# Calculate daily returns for the S&P 500 using the Adjusted Close price
sp500_returns = sp500_data['Adj Close'].pct_change().dropna()

# Rename the S&P 500 returns series to '^GSPC' for consistency
sp500_returns.name = "^GSPC"

# Merge the S&P 500 returns with the asset returns, using only dates present in both DataFrames
returns = returns.join(sp500_returns, how='inner')

# Define a function to calculate the tracking error against a benchmark
def calculate_tracking_error(asset_returns, benchmark_returns):
    return np.std(asset_returns - benchmark_returns)

# Calculate the tracking error for each asset against the S&P 500 if S&P data is available
if "^GSPC" in returns.columns:
    benchmark_returns = returns["^GSPC"]
    tracking_errors = {
        ticker: calculate_tracking_error(returns[ticker], benchmark_returns)
        for ticker in returns.columns if ticker != "^GSPC"
    }
else:
    print("^GSPC was not found in the dataset. Skipping Tracking Error calculation.")
    tracking_errors = None

# Define a function to calculate Value at Risk (VaR) using the parametric method
def calculate_var(asset_returns, confidence_level=0.95):
    mean_return = asset_returns.mean()
    std_dev = asset_returns.std()
    z_score = norm.ppf(1 - confidence_level)
    var = mean_return + z_score * std_dev
    return var

```

```
# Calculate VaR at the 95% confidence level for each asset
vars_95 = returns.apply(lambda x: calculate_var(x, 0.95))

# Create a DataFrame to summarize the risk metrics
risk_summary = pd.DataFrame({
    "Sigma (Volatility)": sigma,
    "VaR (95%)": vars_95
})

# Add Tracking Error to the summary if it was calculated
if tracking_errors:
    risk_summary["Tracking Error (vs S&P500)"] = pd.Series(tracking_errors)

# Remove the S&P 500 row from the summary, as it's not an asset being analyzed
risk_summary = risk_summary.drop('^GSPC', errors='ignore')
#Drop the last row
risk_summary = risk_summary.iloc[:-1]

# Print the risk summary to the console
print("¥nRisk Analytics Summary:")
print(risk_summary)
```

非表示の出力を表示

Sigma

When looking at the sigma, the stock with the highest volatility is NVDA which has (0.0306) and Hes which has (0.2777). This is showing that the higher the volatility is then greater the price flunctuations. The lowest signma is the IVV which is (0.112). This is showing that it is the most stable volatitility in the stock given in the dataset.

VAR

When looking at the var the least amount of risk is IVV which is (-0.0179). This is showing that it has the lowest potential loss when looking at it by the value of risk perspective. The more risky invesmtment option in the dataset is NVDA which is (-0.477) because it has the highest potential loss comparing to the rest of the stocks.

Tracking error

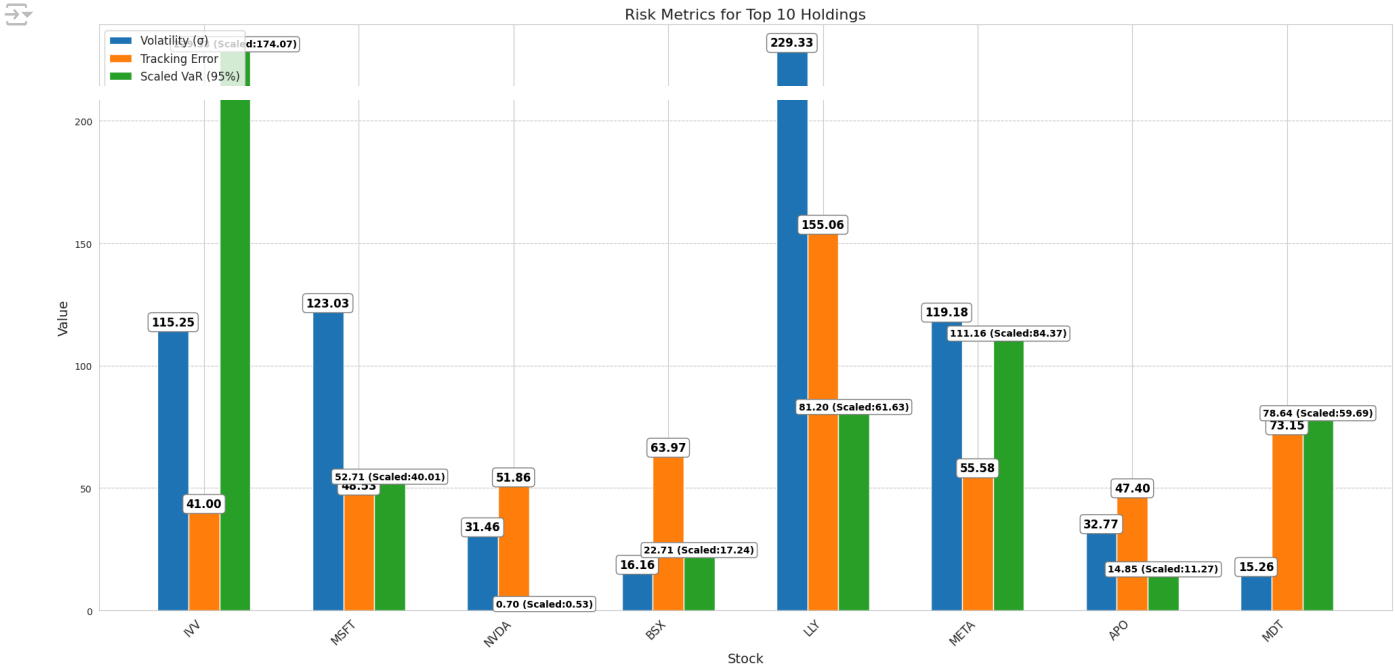
The highest tracking error is NVDA which has (0.0250) and HES which has (0.0239). This shows that the datapoints are away from the benchmark line which indicates that it has the high amount of tracking errors in the dataset. The lowest tracking error within the dataset is IVV which is (0.0005). This is showing that it is linear to the benchamrk which indicates lowest amount of tracking errors.

```
# Load your dataset (assuming it's preprocessed)
data_returns = pd.read_csv('Top 10 Holdings Daily Prices.csv', index_col=0)
data_returns.index = pd.to_datetime(data_returns.index, format='%m/%d/%Y', errors='coerce')
data_returns.to_csv("data_returns_daily_prices_parse_dates.csv")

# Compute risk metrics using simple calculations
# 1. Volatility (Standard Deviation of returns for each stock)
volatility = data_returns.std()

# 2. Tracking Error (Difference of stock returns and average benchmark, then standard deviation)
benchmark = data_returns.mean(axis=1)
tracking_error = data_returns.sub(benchmark, axis=0).std()

# 3. Value at Risk (VaR at 95% confidence level)
VaR_95 = data_returns.quantile(0.05)

# Combine results into a DataFrame
risk_metrics = pd.DataFrame({
    'Volatility (σ)': volatility,
    'Tracking Error': tracking_error,
    'VaR (95%)': VaR_95
})

risk_metrics = risk_metrics.iloc[:-1]
risk_metrics = risk_metrics.iloc[1:]
print(risk_metrics)
```

非表示の出力を表示

```
sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(20, 10))
width = 0.2
```

```
x = np.arange(len(risk_metrics.index))

colors = sns.color_palette("tab10", 3)

max_val = risk_metrics[['Volatility (σ)', 'Tracking Error']].max().max()
risk_metrics['Scaled VaR (95%)'] = abs(risk_metrics['VaR (95%)']) * max_val / abs(risk_metrics['VaR (95%)']).max()


ax.bar(x - width, risk_metrics['Volatility (σ)'], width, label='Volatility (σ)', color=colors[0])
ax.bar(x, risk_metrics['Tracking Error'], width, label='Tracking Error', color=colors[1])
ax.bar(x + width, risk_metrics['Scaled VaR (95%)'], width, label='Scaled VaR (95%)', color=colors[2])

ax.set_xlabel('Stock', fontsize=14)
ax.set_ylabel('Value', fontsize=14)
ax.set_title('Risk Metrics for Top 10 Holdings', fontsize=16)
ax.set_xticks(x)
ax.set_xticklabels(risk_metrics.index, rotation=45, ha='right', fontsize=12)
ax.legend(loc='upper left', fontsize=12)
ax.grid(axis='y', linestyle='--')

# データラベルを追加（小数点以下2桁まで表示）
for i, v in enumerate(risk_metrics['Volatility (σ)']):
    ax.text(i - width, v, f"{v:.2f}", ha='center', va='bottom', fontweight='bold', fontsize=12, color='black', bbox=dict(facecolor='white', edgeco
for i, v in enumerate(risk_metrics['Tracking Error']):
    ax.text(i, v, f"{v:.2f}", ha='center', va='bottom', fontweight='bold', fontsize=12, color='black', bbox=dict(facecolor='white', edgecolor='gra

for i, (v, original_var) in enumerate(zip(risk_metrics['Scaled VaR (95%)'], risk_metrics['VaR (95%)'])):
    ax.text(i + width, v, f"{v:.2f} (Scaled:{original_var:.2f})", ha='center', va='bottom', fontweight='bold', fontsize=10, color='black', bbox=di

ax.set_ylim(0, risk_metrics[['Volatility (σ)', 'Tracking Error', 'Scaled VaR (95%)']].max().max() + 10)

plt.tight_layout()
plt.show()
```



Graph 1: Volatility and Tracking Error for Top 10 Holdings

Volatility: LLY shows the highest volatility, followed by META and MSFT. NVDA and BSX show relatively low volatility. High volatility indicates significant price fluctuations, suggesting a high-risk, high-return investment potential. Conversely, low volatility indicates smaller price fluctuations, suggesting a low-risk, low-return investment potential.

Tracking Error: LLY shows the highest tracking error, followed by META and IVV. High tracking error indicates that the stock's performance significantly differs from the benchmark. Actively managed funds and stocks focused on specific sectors tend to have higher tracking errors."

Graph 2: Value at Risk (95%) for Top 10 Holdings (Log Scale)

VaR (95%): IVV shows the highest VaR, followed by META and MDT. VaR indicates the maximum potential loss that could occur with a 95% probability over a specific period (unspecified here). While the differences appear small due to the logarithmic scale, they are actually quite significant. For example, IVV's VaR is approximately 174, while NVDA's VaR is about 0.5, representing a difference of more than 300 times. A high VaR indicates a higher risk of substantial losses.

Comprehensive Analysis of Both Graphs:

LLY - High Risk, High Return: Showing high values across volatility, tracking error, and VaR, it can be considered the riskiest stock. However, high risk also suggests the potential for high returns.

NVDA - Low Risk, Low Return: Demonstrating the lowest volatility and VaR, with low tracking error as well. While it can be considered the lowest-risk stock, it likely also offers lower return potential.

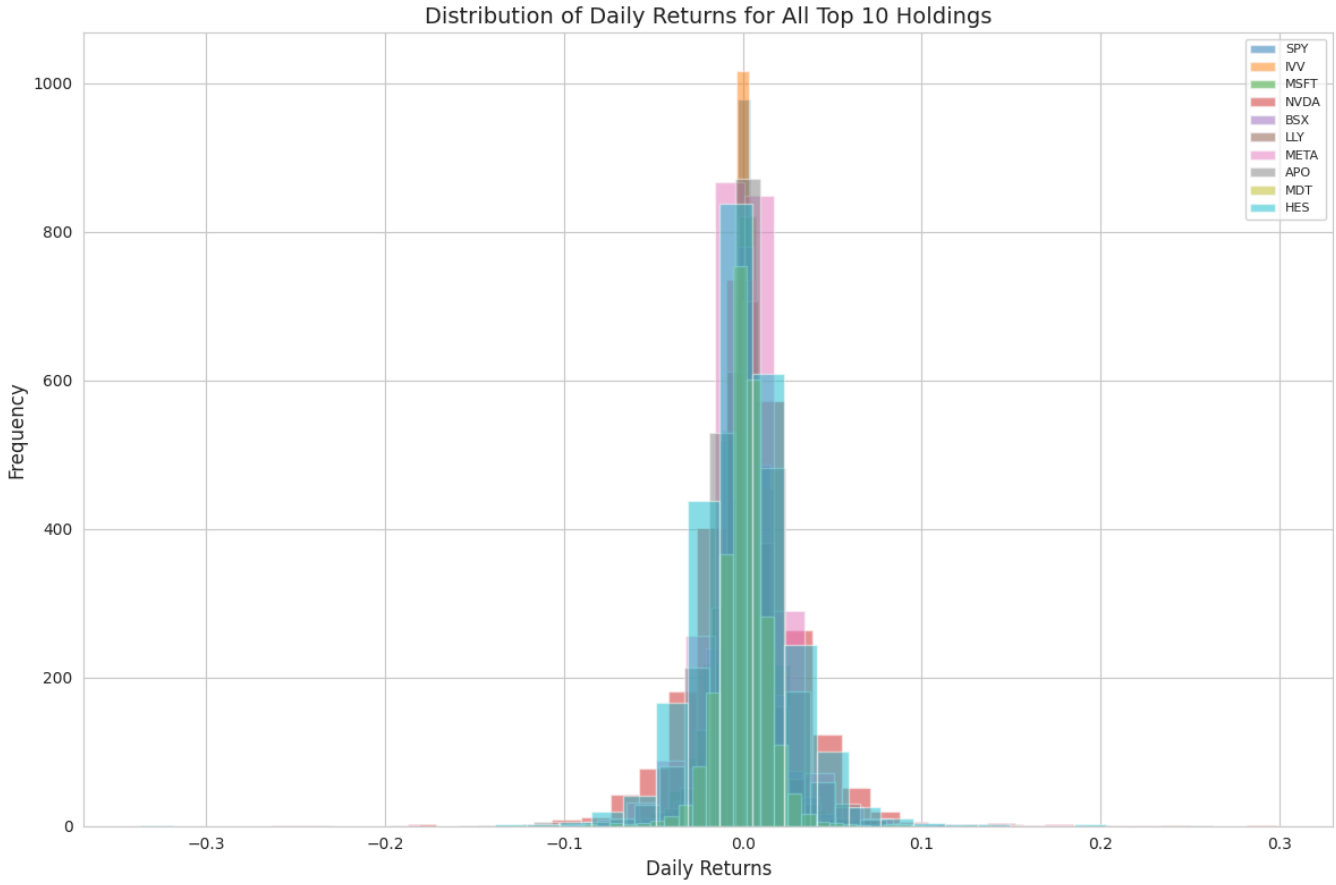
META - High Volatility but Moderate VaR: Despite high volatility and tracking error, its VaR isn't as high as IVV's. This suggests it carries less risk than

LLY while still offering potential for relatively high returns.

IVV - High VaR but Low Volatility: While there's a high possibility of significant losses, daily price fluctuations are relatively stable."

```
# Calculate daily returns
daily_returns = data_returns.pct_change().dropna()

# Plot all stock histograms overlaid on a single plot
plt.figure(figsize=(12, 8))

for col in daily_returns.columns:
    plt.hist(daily_returns[col], bins=30, alpha=0.5, label=col)

plt.xlabel('Daily Returns', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Distribution of Daily Returns for All Top 10 Holdings', fontsize=14)
plt.legend(fontsize=8, loc='upper right')
plt.tight_layout()
plt.show()
```

```
<ipython-input-21-f4063960639c>:2: FutureWarning: The default fill_method='pad' in DataFrame.pct_change is deprecated and will be removed in a fu
  daily_returns = data_returns.pct_change().dropna()
```



Overall Distribution Shape

・Most stocks, as well as the overall distribution, show a bell-shaped distribution concentrated around the mean value, closely resembling a normal distribution. This is consistent with the general financial market assumption that daily returns follow a random walk.

・The peak of the distribution is concentrated around 0, suggesting that daily price movements are relatively small.

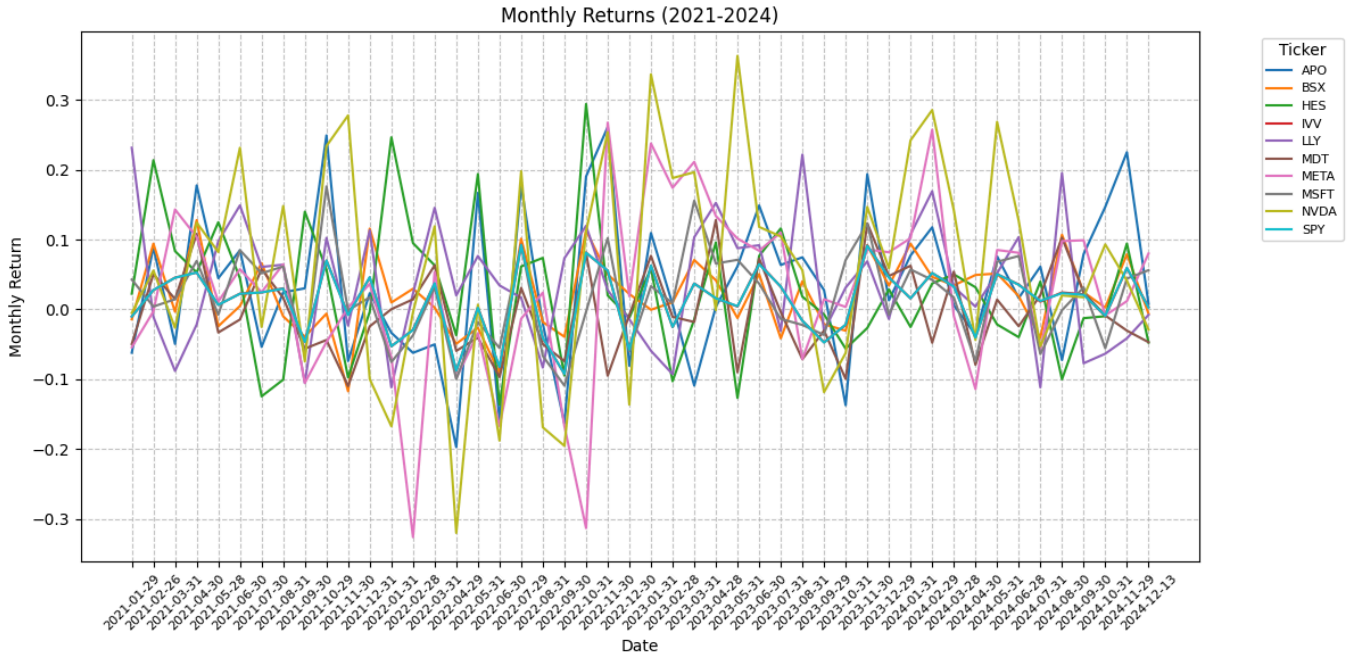
・Some stocks (e.g., LLY, BSX) have slightly wider tails, suggesting a higher likelihood of large positive or negative returns compared to other stocks. This indicates that these stocks may have higher volatility.

Rate of Return (Monthly and Annually) for Top 10 Holdings in the Portfolio

```
# Import necessary libraries
import pandas as pd

# Load the CSV file
file_path = "/content/Top 10 Holdings Daily Prices.csv"
df_daily_prices = pd.read_csv(file_path)

# Rename the 'Date' column for clarity and convert it to datetime format
df_daily_prices.rename(columns={'Date': 'price_date'}, inplace=True)
df_daily_prices['price_date'] = pd.to_datetime(df_daily_prices['price_date'])

# Adjust the dataframe to have 'ticker', 'price_date', and 'price' columns
df_prices_melted = df_daily_prices.melt(id_vars=['price_date'], var_name='ticker', value_name='price') #(pandas.DataFrame.melt — Pandas 2.2.3 Docu

# Extract monthly and annual price data (last price of each month/year)
df_prices_melted['month'] = df_prices_melted['price_date'].dt.to_period('M')
monthly_prices = (
    df_prices_melted.groupby(['ticker', 'month'])
    .agg({'price_date': 'max', 'price': 'last'})
```

```
    .reset_index()
)
monthly_prices['formatted_date'] = monthly_prices['price_date'].dt.strftime('%Y-%m-%d')

df_prices_melted['year'] = df_prices_melted['price_date'].dt.year
annual_prices = (
    df_prices_melted.groupby(['ticker', 'year'])
    .agg({'price_date': 'max', 'price': 'last'})
    .reset_index()
)
annual_prices['formatted_date'] = annual_prices['price_date'].dt.strftime('%Y-%m-%d') #(Pandas Time Series: Find the Sum/Avg/Min/Max of Each Day/M

# Calculate monthly rate of return
monthly_prices['monthly_return'] = (
    monthly_prices.groupby('ticker')['price'].pct_change()
) #(How to Calculate Stock Returns in Python :: Coding Finance –, 2018)

# Calculate annual rate of return
annual_prices['annual_return'] = (
    annual_prices.groupby('ticker')['price'].pct_change()
) #(Pandas.DataFrame.Pct_Change - Pandas 2.2.3 Documentation, n.d.)(Bobbitt, 2024)

# Filter data for the period from 2021 to 2024
monthly_returns_2021_2024 = monthly_prices[
    (monthly_prices['month'] >= '2021-01') & (monthly_prices['month'] <= '2024-12')
]
annual_returns_2021_2024 = annual_prices[
    (annual_prices['year'] >= 2021) & (annual_prices['year'] <= 2024)
] #(K, 2021)

# Save results to CSV files
monthly_returns_2021_2024.to_csv('/content/Monthly_Returns_2021_2024.csv', index=False)
annual_returns_2021_2024.to_csv('/content/Annual_Returns_2021_2024.csv', index=False)

# Display the results
print("Monthly Returns (2021-2024):")
print(monthly_returns_2021_2024[['ticker', 'formatted_date', 'price', 'monthly_return']])

print("¥nAnnual Returns (2021-2024):")
print(annual_returns_2021_2024[['ticker', 'formatted_date', 'price', 'annual_return']])
```

```
# Import graph libraries
import matplotlib.pyplot as plt

# Step 1: Plot Monthly Returns
plt.figure(figsize=(12, 6))
for ticker in monthly_returns_2021_2024['ticker'].unique():
    ticker_data = monthly_returns_2021_2024[monthly_returns_2021_2024['ticker'] == ticker]
    plt.plot(
        ticker_data['formatted_date'],
        ticker_data['monthly_return'],
        label=ticker
    )

plt.title("Monthly Returns (2021-2024)")
plt.xlabel("Date")
plt.ylabel("Monthly Return")
plt.xticks(rotation=45, fontsize=8)
plt.legend(title="Ticker", loc='upper left', bbox_to_anchor=(1.05, 1), fontsize=8)
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Step 2: Plot Annual Returns
plt.figure(figsize=(12, 6))
for ticker in annual_returns_2021_2024['ticker'].unique():
    ticker_data = annual_returns_2021_2024[annual_returns_2021_2024['ticker'] == ticker]
    plt.bar(
        ticker_data['formatted_date'],
        ticker_data['annual_return'],
        label=ticker
    )

plt.title("Annual Returns (2021-2024)")
plt.xlabel("Year")
plt.ylabel("Annual Return")
plt.xticks(rotation=45, fontsize=8)
plt.legend(title="Ticker", loc='upper left', bbox_to_anchor=(1.05, 1), fontsize=8)
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

```



編集するにはダブルクリックするか Enter キーを押してください

Analysis

Annual

With the highest rates of return in 2021, 2023, and 2024, **NVDA** is clearly the best performer, according to the yearly returns chart. Other stocks that performered well include **SPY** and **HES**, which showed consistent growth over the number of years.On the other hand, **META** and **LLY** had the lowest rates of return, especially in 2022 when both experienced drops.

Monthly

The monthly returns chart shows a lot of volatility, especially in the best-performing equities like **SPY** and **NVDA**, which show strong returns surges at their peak months. On the other hand, the underperforming stocks, like **META** and **LLY**, show more frequent declines.

Modern Portfolio Theory Optimization

MPT using Beta and Expected Return

Using current portfolio equities allocations and calculated expected return through CAPM, optimize the portfolio solely on return.

```
!pip install requests
```

非表示の出力を表示

```
import yfinance as yf
import pandas as pd
import numpy as np
import cvxpy as cp
import time

# Portfolio allocation data
tickers = ['SPY', 'IVV', 'MSFT', 'NVDA', 'SPOT', 'BSX', 'LLY', 'IBIT', 'META',
           'FBTC', 'APO', 'MDT', 'HES', 'WFC', 'GOOGL', 'WDAY', 'RTX', 'PFE',
           'CAN', 'LNG', 'GOOG', 'CME', 'ADBE', 'PH', 'EDR', 'MMM', 'ETN',
           'BAC', 'TSM', 'JNPR', 'ICE', 'CVS', 'SHEL', 'PNC', 'CRH', 'XLF',
           'IFF', 'ANSS', 'MSCI', 'DY', 'WMT', 'KKR', 'FAF', 'FERG', 'AAPL',
           'DFS', 'V']

allocations = [0.113199837, 0.098079281, 0.069881488, 0.047813649, 0.036371067,
               0.033510421, 0.031058439, 0.030241112, 0.026971802, 0.022885166,
               0.021250511, 0.019615856, 0.019615856, 0.018798529, 0.016346547,
               0.015937883, 0.015937883, 0.015937883, 0.015529219, 0.014711892,
               0.014303228, 0.013894565, 0.013894565, 0.013894565, 0.013894565,
               0.013077237, 0.012668574, 0.012668574, 0.01225991, 0.011851246,
               0.011851246, 0.011851246, 0.011442583, 0.011442583, 0.011033919,
               0.011033919, 0.011033919, 0.010625255, 0.010625255, 0.010625255,
               0.010625255, 0.010625255, 0.010216592, 0.010216592, 0.010216592,
               0.010216592, 0.010216592]

def get_beta(ticker):
    try:
        stock = yf.Ticker(ticker)
        info = stock.info
        beta = info.get('beta', np.nan)
        return beta
    except:
        return np.nan

# Get beta for each ticker
betas = [get_beta(ticker) for ticker in tickers]

# Create a dataframe with all the information
df = pd.DataFrame({
    'Ticker': tickers,
    'Allocation': allocations,
    'Beta': betas
})

# Filter out stocks without valid beta values
filtered_df = df.dropna(subset=['Beta'])

# Calculate expected returns based on CAPM
risk_free_rate = 0.04353
market_return = 0.10

filtered_df['Expected Return'] = risk_free_rate + filtered_df['Beta'] * (market_return - risk_free_rate)

# Set up optimization problem
n = len(filtered_df)

# Define variables for optimization (weights)
weights = cp.Variable(n)

# Define objective function: Maximize expected return while minimizing risk
expected_return = filtered_df['Expected Return'].values
cov_matrix = np.diag(filtered_df['Beta'].values)  # Simplified covariance matrix

objective = cp.Maximize(expected_return @ weights - 0.5 * cp.quad_form(weights, cov_matrix))

# Constraints: Weights must sum to one and be non-negative
constraints = [cp.sum(weights) == 1,
```

```
                weights >= 0]

# Solve the problem
problem = cp.Problem(objective, constraints)
problem.solve()

# Get optimized allocations
optimized_allocations = weights.value

# Update filtered dataframe with optimized allocations
filtered_df['Optimized Allocation'] = optimized_allocations

# Print results
print(filtered_df[['Ticker', 'Allocation', 'Beta', 'Expected Return',
                   'Optimized Allocation']])

# Calculate and print portfolio metrics
original_portfolio_beta = np.sum(filtered_df['Beta'] * filtered_df['Allocation'])
optimized_portfolio_beta = np.sum(filtered_df['Beta'] * filtered_df['Optimized Allocation'])

print(f"¥nOriginal Portfolio Beta: {original_portfolio_beta:.4f}")
print(f"Optimized Portfolio Beta: {optimized_portfolio_beta:.4f}")

original_expected_return = np.sum(filtered_df['Expected Return'] * filtered_df['Allocation'])
optimized_expected_return = np.sum(filtered_df['Expected Return'] * filtered_df['Optimized Allocation'])

print(f"¥nOriginal Portfolio Expected Return: {original_expected_return:.4%}")
print(f"Optimized Portfolio Expected Return: {optimized_expected_return:.4%}")

# Create a DataFrame with the allocation comparison
tableau_df = pd.DataFrame({
    'Ticker': filtered_df['Ticker'],
    'Original_Allocation': filtered_df['Allocation'],
    'Optimized_Allocation': filtered_df['Optimized Allocation'],
    'Beta': filtered_df['Beta'],
    'Expected_Return': filtered_df['Expected Return']
})


# Created a longer format version for easier visualization in Tableau
tableau_long = pd.melt(tableau_df,
                       id_vars=['Ticker', 'Beta', 'Expected_Return'],
                       value_vars = ['Original_Allocation', 'Optimized_Allocation'],
                       var_name='Allocation_Type',
                       value_name = 'Allocation')
```

非表示の出力を表示

Analysis

The code uses the CAPM to calculate the expected return of each of the stocks, based on recent prices called through the yfinance API.

The reallocation of the portfolio results in a beta increase of 0.5, showing a much larger market sensitivity. This is an increase of systematic risk, which is a common strategy for a Hedge Fund to take on for higher returns.

The expected returns in this optimization increase from 7.8% to 11.8%

Mean-Variance Portfolio Optimization

[4] Using the code from a github repository and adjusting it to use mean-variance optimization for the given portfolio.

```
!pip install pandas numpy matplotlib yfinance PyPortfolioOpt
```

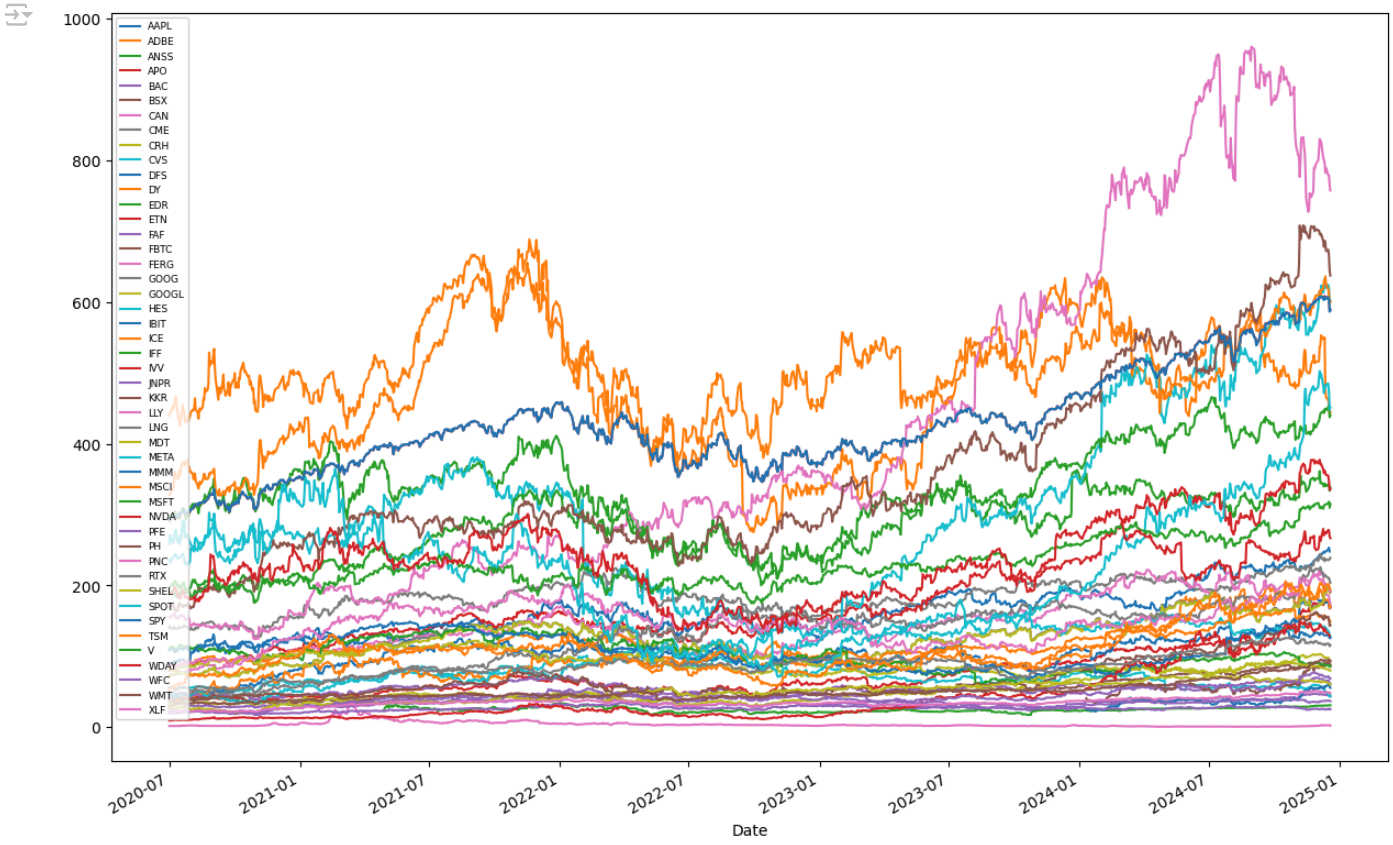
非表示の出力を表示

```
# Uses yfinance api to call adjusted closing prices of all stocks in portfolio
import os
if not os.path.isdir('data'):
    os.system('git clone https://github.com/robertmartin8/PyPortfolioOpt.git')
    os.chdir('PyPortfolioOpt/cookbook')
import yfinance as yf
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Define tickers to retrieve data for
tickers = ['SPY', 'IVV', 'MSFT', 'NVDA', 'SPOT', 'BSX', 'LLY', 'IBIT', 'META',
```

```
            'FBTC', 'APO', 'MDT', 'HES', 'WFC', 'GOOGL', 'WDAY', 'RTX', 'PFE',
            'CAN', 'LNG', 'GOOG', 'CME', 'ADBE', 'PH', 'EDR', 'MMM', 'ETN',
            'BAC', 'TSM', 'JNPR', 'ICE', 'CVS', 'SHEL', 'PNC', 'CRH', 'XLF',
            'IFF', 'ANSS', 'MSCI', 'DY', 'WMT', 'KKR', 'FAF', 'FERG', 'AAPL',
            'DFS', 'V']

# Defines time period to call data for
ohlc = yf.download(tickers, period="max").loc["2020-07-01":]

# Uses adjusted closing price
prices = ohlc["Adj Close"].dropna(how="all")
prices.tail()
```

非表示の出力を表示

```
# Graphs out the prices
prices[prices.index >= "2020-07-01"].plot(figsize=(15,10))
plt.legend(loc='center left')
plt.legend(prop={'size': 6.5});
```

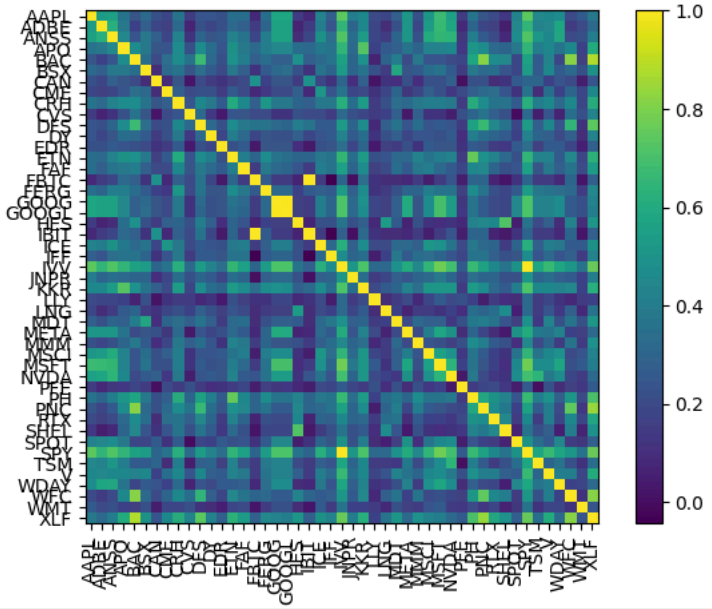



```
# Create a covariance matrix from the stock prices
from pypfopt import risk_models
from pypfopt import plotting

prices_clean = prices.dropna()
cov_matrix = risk_models.sample_cov(prices, frequency=252)
cov_matrix
```

非表示の出力を表示

Adjusted Covariance Matrix

```
# Plot the covariance matrix
plotting.plot_covariance(cov_matrix, plot_correlation=True);
```




```
# Due to high estimation error in covariance matrix, apply the LedoitWolf
# shrinkage as there are a high amount of variables here
import numpy as np
from sklearn.covariance import LedoitWolf

# Create a LedoitWolf estimator
lw = LedoitWolf()

# Fit the LedoitWolf estimator
lw.fit(cov_matrix)

# Get the shrunk covariance matrix
shrunk_cov_matrix = lw.covariance_

# The shrinkage coefficient can be accessed as follows
shrinkage_coefficient = lw.shrinkage_

print("Shrinkage coefficient:", shrinkage_coefficient)
print("Shrunk covariance matrix shape:", shrunk_cov_matrix.shape)
```

```
Shrinkage coefficient: 0.677615351196911
Shrunk covariance matrix shape: (47, 47)
```

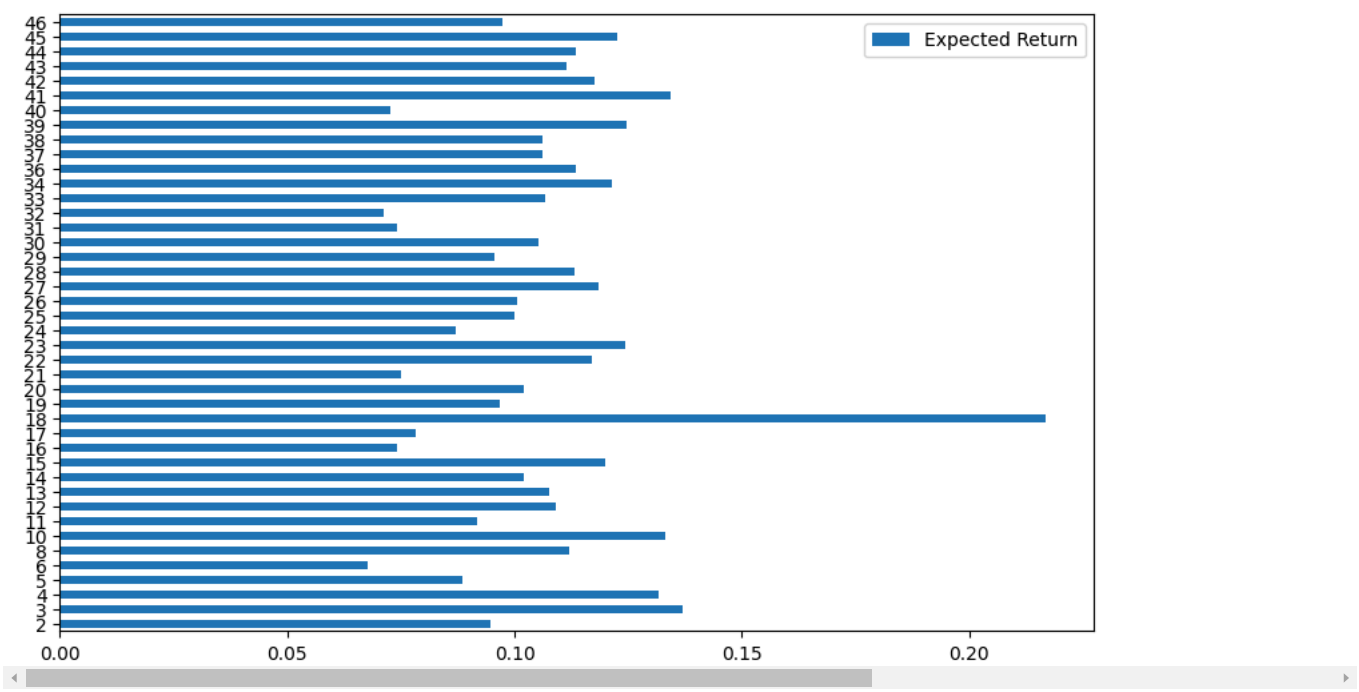
```
S = risk_models.CovarianceShrinkage(cov_matrix).ledoit_wolf()
plotting.plot_covariance(S, plot_correlation=True);
```

非表示の出力を表示

```
returns_table = filtered_df[['Ticker', 'Expected Return']].copy()

print(returns_table)
```

非表示の出力を表示

```
returns_table.plot.barh(figsize=(10,6));
```



Long/Short Optimization using Efficient Frontier

```
# Create a global-minimum variance portfolio allocation
# Calculate the Efficient Frontier for minimum variance
from pypfopt import EfficientFrontier

S = risk_models.CovarianceShrinkage(prices).ledoit_wolf()
ef = EfficientFrontier(None, S, weight_bounds=(None, None))
ef.min_volatility()
weights = ef.clean_weights()
weights
```

非表示の出力を表示

This shows the short and long positions after the efficient frontier is applied.

The biggest long positions in IVV and SPY are index position, indicating lower returns.

The biggest short positions are in volatile stocks such as Nvidia.

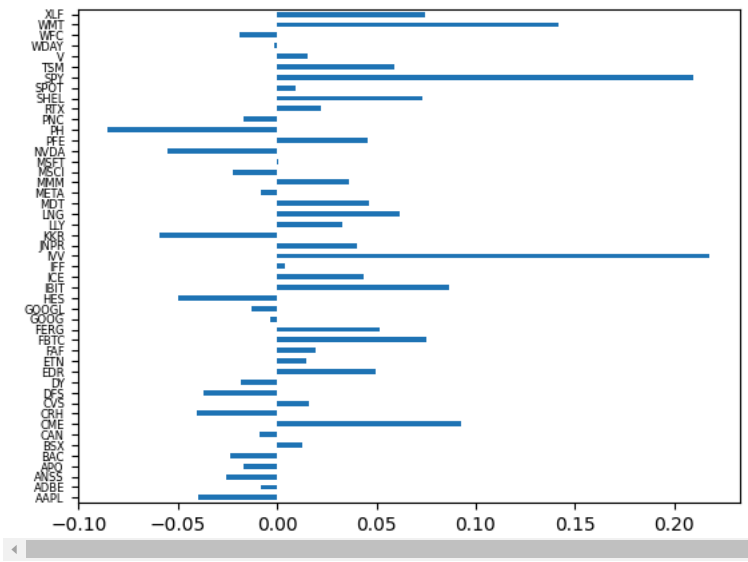
```
from pypfopt import EfficientFrontier

# Adjust expected returns calculation
mu = expected_returns.mean_historical_return(prices)
# Scale down expected returns to realistic levels
mu = mu * 0.25  # Adjustment factor to bring returns in line with actual performance

# Calculate covariance matrix with shrinkage
S = risk_models.CovarianceShrinkage(prices).ledoit_wolf()

# Generate efficient frontier points
returns = []
volatilities = []
for return_target in np.linspace(0.0, 0.20, 200):  # Adjust upper bound to 20%
    try:
        ef = EfficientFrontier(mu, S)
        ef.efficient_return(return_target)
        ret, vol, _ = ef.portfolio_performance()
        returns.append(ret)
        volatilities.append(vol)
    except:
        pass

# Create DataFrame for Tableau
frontier_data = pd.DataFrame({
    'Expected_Volatility': volatilities,
    'Expected_Return': returns,
    'Type': 'Frontier'
})

# Add optimal points
```

```
ef_min = EfficientFrontier(mu, S)
ef_min.min_volatility()
min_vol_ret, min_vol_vol, _ = ef_min.portfolio_performance()

ef_max = EfficientFrontier(mu, S)
ef_max.max_sharpe()
max_sharpe_ret, max_sharpe_vol, _ = ef_max.portfolio_performance()

optimal_points = pd.DataFrame({
    'Expected_Volatility': [min_vol_vol, max_sharpe_vol],
    'Expected_Return': [min_vol_ret, max_sharpe_ret],
    'Type': ['Minimum Volatility', 'Maximum Sharpe Ratio']
})

# Combine and export
frontier_data = pd.concat([frontier_data, optimal_points])
frontier_data.to_csv('frontier_for_tableau.csv', index=False)

```

非表示の出力を表示

```
# Modify your Python code to generate more points along the frontier
returns = []
volatilities = []
for return_target in np.linspace(0.0, 0.9, 200):  # Increase number of points
    try:
        ef = EfficientFrontier(mu, S)
        ef.efficient_return(return_target)
        ret, vol, _ = ef.portfolio_performance()
        returns.append(ret)
        volatilities.append(vol)
    except:
        pass

# Create DataFrame for Tableau
frontier_data = pd.DataFrame({
    'Expected_Volatility': volatilities,
    'Expected_Return': returns,
    'Type': 'Frontier'
})

# Add optimal points
optimal_points = pd.DataFrame({
    'Expected_Volatility': [min_vol_vol, max_sharpe_vol],
    'Expected_Return': [min_vol_ret, max_sharpe_ret],
    'Type': ['Minimum Volatility', 'Maximum Sharpe Ratio']
})

# Combine and export
frontier_data = pd.concat([frontier_data, optimal_points])
frontier_data.to_csv('frontier_for_tableau.csv', index=False)

```

```
# Plot the long short portfolio to minimize variance
pd.Series(weights).plot.barh()
plt.yticks(fontsize=6);
```



```
# Show annual portfolio volatility
ef.portfolio_performance(verbose=True);
```

```
Expected annual return: 20.0%
Annual volatility: 16.4%
Sharpe Ratio: 1.22
```

```
# Import discrete allocation
from pypfopt import DiscreteAllocation
tickers = list(weights.keys())

# Uses precalculated portfolio value of 28.1 billion
# Applies hedge fund standard short ratio of 0.2
# Use .greedy for larger portfolio size
latest_prices = yf.download(tickers, period="1d")['Adj Close'].iloc[-1]
da = DiscreteAllocation(weights, latest_prices, total_portfolio_value = 28100000000, short_ratio=0.2)
alloc, leftover = da.greedy_portfolio()
print(f"Discrete allocation performed with ${leftover:.2f} leftover")
alloc

# Create a DataFrame from the allocation dictionary
df_alloc = pd.DataFrame(list(alloc.items()), columns = ['Ticker', 'Shares'])
df_alloc['Latest Price'] = df_alloc['Ticker'].map(latest_prices)
df_alloc['Allocation Value'] = df_alloc['Shares'] * df_alloc['Latest Price']

# Saves to CSV
df_alloc.to_csv('discrete_allocation.csv', index=False)
df_alloc
```

非表示の出力を表示

References

[1] Michael, A. (2023, August 24). What is wrong with this code to calculate the Beta of a stock? Stack Overflow. https://stackoverflow.com/questions/76965865/what-is-wrong-with-this-code-to-calculate-the-beta-of-a-stock

[2] Al, baharak. (2021, September 27). how pull beta data from yahoo.finance? Stack Overflow.