Start coding or generate with AI.

```
import pandas as pd
import pathlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.optimize as sci_opt

from pprint import pprint
from sklearn.preprocessing import StandardScaler

# Load the dataset
DATA = pd.read_csv('/content/Portfolio_dataset1.csv')

# Print the head of the data
print(DATA.head())

# Limit the columns to 'DATE', 'Symbol', and 'Close'
DATA = DATA[['DATE', 'Symbol', 'Close']]

# Pivot the data to reorganize it
DATA = DATA.pivot(index='DATE', columns='Symbol', values='Close')
print(DATA.head())

# Calculate the log of returns
log_return = np.log(1 + DATA.pct_change(fill_method=None))

# Define the number of symbols for weight generation
number_of_symbols = len(DATA.columns)

# Generate Random Weights
random_weights = np.random.random(number_of_symbols)

# Generate Rebalance Weights
rebalance_weights = random_weights / np.sum(random_weights)

# Calculate the Expected Returns, annualized by multiplying by 252
exp_ret = np.sum((log_return.mean() * rebalance_weights) * 252)

# Calculate the Expected Volatility, annualized by multiplying by 252
exp_vol = np.sqrt(
    np.dot(
        rebalance_weights.T,
        np.dot(log_return.cov() * 252, rebalance_weights)
    )
)

# Calculate the Sharpe Ratio
sharpe_ratio = exp_ret / exp_vol

# Put the weights into a DataFrame
weights_df = pd.DataFrame({
    'random_weights': random_weights,
    'rebalance_weights': rebalance_weights
})

print('')
print('=' * 80)
print('PORTFOLIO WEIGHTS:')
print('-' * 80)
print(weights_df)
print('-' * 80)

# Put the metrics into a DataFrame
metrics_df = pd.DataFrame({
    'Expected Portfolio Returns': [exp_ret],
    'Expected Portfolio Volatility': [exp_vol],
    'Portfolio Sharpe Ratio': [sharpe_ratio]
})

print('')
print('=' * 80)
print('PORTFOLIO METRICS:')
print('-' * 80)
print(metrics_df)
print('-' * 80)
```

```
                         DATE  Adj Close    Close     High      Low     Open  \
0  2014-01-01 00:00:00+00:00   5.156387  6.45375  7.01537  6.92356  7.82630
1  2014-01-02 00:00:00+00:00   5.219006  6.94125  7.03125  6.92625  7.03125
2  2014-01-03 00:00:00+00:00   5.371264  7.14375  7.19750  7.11125  7.14375
3  2014-01-06 00:00:00+00:00   5.292315  7.03875  7.11375  7.02125  7.11125
4  2014-01-07 00:00:00+00:00   5.271638  7.01125  7.04875  6.95625  6.97625

    Volume Symbol
0  3643211   INFY
1  3642400   INFY
2  8421600   INFY
3  4820000   INFY
4  6201600   INFY
Symbol                           BATA     INFY        TAMO      TTML
DATE
2014-01-01 00:00:00+00:00  533.825012  6.45375  370.970978  7.511293
2014-01-02 00:00:00+00:00  515.150024  6.94125  368.398560  7.319922
2014-01-03 00:00:00+00:00  520.474976  7.14375  358.850952  7.989719
2014-01-06 00:00:00+00:00  513.150024  7.03875  363.055847  7.654821
2014-01-07 00:00:00+00:00  512.275024  7.01125  361.225494  7.559135

================================================================================
PORTFOLIO WEIGHTS:
--------------------------------------------------------------------------------
   random_weights  rebalance_weights
0        0.885368           0.294202
1        0.809463           0.268979
2        0.703021           0.233609
3        0.611537           0.203210
--------------------------------------------------------------------------------

================================================================================
PORTFOLIO METRICS:
--------------------------------------------------------------------------------
   Expected Portfolio Returns  Expected Portfolio Volatility  \
0                    0.11634                       0.225065

   Portfolio Sharpe Ratio
0                0.516916
--------------------------------------------------------------------------------
```

```python
# Initialize the components, to run a Monte Carlo Simulation.

# We will run 5000 iterations.
num_of_portfolios = 5000

# Prep an array to store the weights as they are generated, 5000 iterations for each of our 4 symbols.
all_weights = np.zeros((num_of_portfolios, number_of_symbols))

# Prep an array to store the returns as they are generated, 5000 possible return values.
ret_arr = np.zeros(num_of_portfolios)

# Prep an array to store the volatilities as they are generated, 5000 possible volatility values.
vol_arr = np.zeros(num_of_portfolios)

# Prep an array to store the sharpe ratios as they are generated, 5000 possible Sharpe Ratios.
sharpe_arr = np.zeros(num_of_portfolios)

# Start the simulations.
for ind in range(num_of_portfolios):

    # First, calculate the weights.
    weights = np.array(np.random.random(number_of_symbols))
    weights = weights / np.sum(weights)

    # Add the weights, to the `weights_arrays`.
    all_weights[ind, :] = weights

    # Calculate the expected log returns, and add them to the `returns_array`.
    ret_arr[ind] = np.sum((log_return.mean() * weights) * 252)

    # Calculate the volatility, and add them to the `volatility_array`.
    vol_arr[ind] = np.sqrt(
        np.dot(weights.T, np.dot(log_return.cov() * 252, weights))
    )

    # Calculate the Sharpe Ratio and Add it to the `sharpe_ratio_array`.
    sharpe_arr[ind] = ret_arr[ind]/vol_arr[ind]
    # Let's create our "Master Data Frame", with the weights, the returns, the volatility, and the Sharpe Ratio
simulations_data = [ret_arr, vol_arr, sharpe_arr, all_weights]

# Create a DataFrame from it, then Transpose it so it looks like our original one.
simulations_df = pd.DataFrame(data=simulations_data).T

# Give the columns the Proper Names.
```

```
simulations_df.columns = [
    'Returns',
    'Volatility',
    'Sharpe Ratio',
    'Portfolio Weights'
]

# Make sure the data types are correct, we don't want our floats to be strings.
simulations_df = simulations_df.infer_objects()

# Print out the results.
print('')
print('='*80)
print('SIMULATIONS RESULT:')
print('-'*80)
print(simulations_df.head())
print('-'*80)
```

```
================================================================================
SIMULATIONS RESULT:
--------------------------------------------------------------------------------
    Returns  Volatility  Sharpe Ratio  \
0  0.169009    0.335255      0.504120
1  0.110380    0.253503      0.435421
2  0.133257    0.253019      0.526667
3  0.096130    0.203121      0.473263
4  0.152730    0.336663      0.453657

                                   Portfolio Weights
0  [0.13783632235463378, 0.26088048556050364, 0.0...
1  [0.2008951597959462, 0.1934868381496411, 0.437...
2  [0.29586831758493015, 0.22006764600148707, 0.1...
3  [0.4405412140475064, 0.343491256211889, 0.1482...
4  [0.18200334838708693, 0.020679655246540005, 0....
--------------------------------------------------------------------------------
```

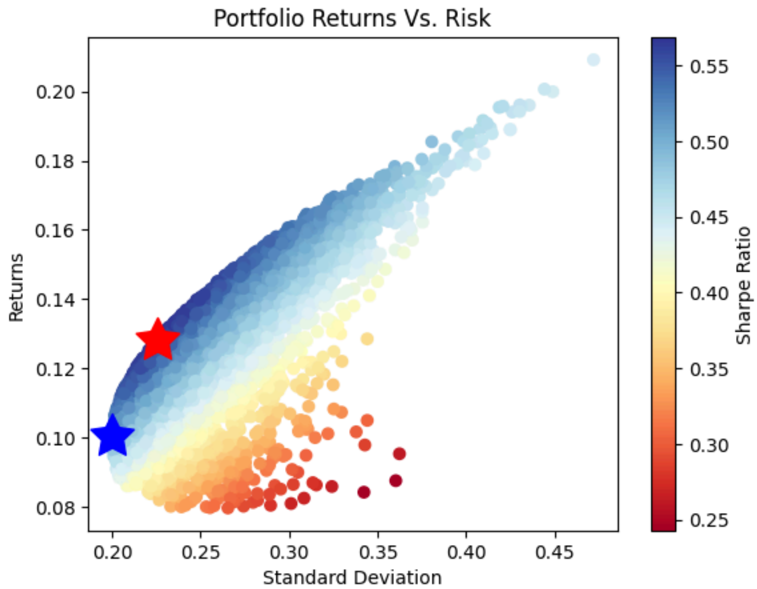
```
# Return the Max Sharpe Ratio from the run.
max_sharpe_ratio = simulations_df.loc[simulations_df['Sharpe Ratio'].idxmax()]

# Return the Min Volatility from the run.
min_volatility = simulations_df.loc[simulations_df['Volatility'].idxmin()]

print('')
print('='*80)
print('MAX SHARPE RATIO:')
print('-'*80)
print(max_sharpe_ratio)
print('-'*80)

print('')
print('='*80)
print('MIN VOLATILITY:')
print('-'*80)
print(min_volatility)
print('-'*80)
```

```
================================================================================
MAX SHARPE RATIO:
--------------------------------------------------------------------------------
Returns                                                       0.128158
Volatility                                                    0.225353
Sharpe Ratio                                                  0.568696
Portfolio Weights    [0.21763927875115546, 0.4842259600550033, 0.05...
Name: 47, dtype: object
--------------------------------------------------------------------------------

================================================================================
MIN VOLATILITY:
--------------------------------------------------------------------------------
Returns                                                       0.100403
Volatility                                                    0.200142
Sharpe Ratio                                                  0.501658
Portfolio Weights    [0.3657061768903361, 0.4368783486119645, 0.118...
Name: 3004, dtype: object
--------------------------------------------------------------------------------
```

```
%matplotlib inline

# Plot the data on a Scatter plot.
plt.scatter(
    y=simulations_df['Returns'],
    x=simulations_df['Volatility'],
```

```
        c=simulations_df['Sharpe Ratio'],
        cmap='RdYlBu'
    )

    # Give the Plot some labels, and titles.
    plt.title('Portfolio Returns Vs. Risk')
    plt.colorbar(label='Sharpe Ratio')
    plt.xlabel('Standard Deviation')
    plt.ylabel('Returns')

    # Plot the Max Sharpe Ratio, using a `Red Star`.
    plt.scatter(
        max_sharpe_ratio[1],
        max_sharpe_ratio[0],
        marker=(5, 1, 0),
        color='r',
        s=600
    )

    # Plot the Min Volatility, using a `Blue Star`.
    plt.scatter(
        min_volatility[1],
        min_volatility[0],
        marker=(5, 1, 0),
        color='b',
        s=600
    )

    # Finally, show the plot.
    plt.show()
```

```
<ipython-input-8-514522f6fe19>:19: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version,
  max_sharpe_ratio[1],
<ipython-input-8-514522f6fe19>:20: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version,
  max_sharpe_ratio[0],
<ipython-input-8-514522f6fe19>:28: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version,
  min_volatility[1],
<ipython-input-8-514522f6fe19>:29: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version,
  min_volatility[0],
```




```
    def get_metrics(weights: list) -> np.array:
        """
        ### Overview:
        ----
        With a given set of weights, return the portfolio returns,
        the portfolio volatility, and the portfolio sharpe ratio.

        ### Arguments:
        ----
        weights (list): An array of portfolio weights.

        ### Returns:
        ----
        (np.array): An array containg return value, a volatility value,
            and a sharpe ratio.
        """
```

```
        # Convert to a Numpy Array.
        weights = np.array(weights)

        # Calculate the returns, remember to annualize them (252).
        ret = np.sum(log_return.mean() * weights) * 252

        # Calculate the volatility, remember to annualize them (252).
        vol = np.sqrt(
            np.dot(weights.T, np.dot(log_return.cov() * 252, weights))
        )

        # Calculate the Sharpe Ratio.
        sr = ret / vol

        return np.array([ret, vol, sr])
    def grab_negative_sharpe(weights: list) -> np.array:
        """The function used to minimize the Sharpe Ratio.

        ### Arguments:
        ----
        weights (list): The weights, we are testing to see
            if it's the minimum.

        ### Returns:
        ----
        (np.array): An numpy array of the portfolio metrics.
        """
        return get_metrics(weights)[2] - 1

    def grab_volatility(weights: list) -> np.array:
        """The function used to minimize the Sharpe Ratio.

        ### Arguments:
        ----
        weights (list): The weights, we are testing to see
            if it's the minimum.

        ### Returns:
        ----
        (np.array): An numpy array of the portfolio metrics.
        """
        return get_metrics(weights)[1]

    def check_sum(weights: list) -> float:
        """Ensure the allocations of the "weights", sums to 1 (100%)

        ### Arguments:
        ----
        weights (list): The weights we want to check to see
            if they sum to 1.

        ### Returns:
        ----
        float: The different between 1 and the sum of the weights.
        """
        return np.sum(weights) - 1


    # Define the boundaries for each symbol. Remember I can only invest up to 100% of my capital into a single asset.
    bounds = tuple((0, 1) for symbol in range(number_of_symbols))

    # Define the constraints, here I'm saying that the sum of each weight must not exceed 100%.
    constraints = ({'type': 'eq', 'fun': check_sum})

    # We need to create an initial guess to start with,
    # and usually the best initial guess is just an
    # even distribution. In this case 25% for each of the 4 stocks.
    init_guess = number_of_symbols * [1 / number_of_symbols]

    # Perform the operation to minimize the risk.
    optimized_sharpe = sci_opt.minimize(
        grab_negative_sharpe, # minimize this.
        init_guess, # Start with these values.
        method='SLSQP',
        bounds=bounds, # don't exceed these bounds.
        constraints=constraints # make sure you don't exceed the 100% constraint.
    )

    # Print the results.
    print('')
    print('='*80)
    print('OPTIMIZED SHARPE RATIO:')
    print('-'*80)
```

```
print(optimized_sharpe)
print('-'*80)
```

```
================================================================================
OPTIMIZED SHARPE RATIO:
--------------------------------------------------------------------------------
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: -0.7937625270773627
       x: [ 6.245e-16  3.678e-16  1.000e+00  0.000e+00]
     nit: 5
     jac: [ 1.360e-01  2.380e-01 -0.000e+00  5.264e-01]
    nfev: 25
    njev: 5
--------------------------------------------------------------------------------
```

```
# Grab the metrics.
optimized_metrics = get_metrics(weights=optimized_sharpe.x)

# Print the Optimized Weights.
print('')
print('='*80)
print('OPTIMIZED WEIGHTS:')
print('-'*80)
print(optimized_sharpe.x)
print('-'*80)


# Print the Optimized Metrics.
print('')
print('='*80)
print('OPTIMIZED METRICS:')
print('-'*80)
print(optimized_metrics)
print('-'*80)
```

```
================================================================================
OPTIMIZED WEIGHTS:
--------------------------------------------------------------------------------
[6.24500451e-16 3.67761377e-16 1.00000000e+00 0.00000000e+00]
--------------------------------------------------------------------------------

================================================================================
OPTIMIZED METRICS:
--------------------------------------------------------------------------------
[0.08283264 0.40163721 0.20623747]
--------------------------------------------------------------------------------
```

```
# Define the boundaries for each symbol. Remember I can only invest up to 100% of my capital into a single asset.
bounds = tuple((0, 1) for symbol in range(number_of_symbols))

# Define the constraints, here I'm saying that the sum of each weight must not exceed 100%.
constraints = ({'type': 'eq', 'fun': check_sum})

# We need to create an initial guess to start with,
# and usually the best initial guess is just an
# even distribution. In this case 25% for each of the 4 stocks.
init_guess = number_of_symbols * [1 / number_of_symbols]

# Perform the operation to minimize the risk.
optimized_volatility = sci_opt.minimize(
    grab_volatility, # minimize this.
    init_guess, # Start with these values.
    method='SLSQP',
    bounds=bounds, # don't exceed these bounds.
    constraints=constraints # make sure you don't exceed the 100% constraint.
)

# Print the results.
print('')
print('='*80)
print('OPTIMIZED VOLATILITY RATIO:')
print('-'*80)
print(optimized_volatility)
print('-'*80)
```

```
================================================================================
OPTIMIZED VOLATILITY RATIO:
--------------------------------------------------------------------------------
 message: Optimization terminated successfully
```

```
   success: True
    status: 0
       fun: 0.19991389573809396
         x: [ 3.870e-01  4.417e-01  9.672e-02  7.452e-02]
       nit: 6
       jac: [ 1.997e-01  2.000e-01  2.000e-01  2.007e-01]
      nfev: 30
      njev: 6
--------------------------------------------------------------------------------
```

```
# Grab the metrics.
optimized_metrics = get_metrics(weights=optimized_volatility.x)

# Print the Optimized Weights.
print('')
print('='*80)
print('OPTIMIZED WEIGHTS:')
print('-'*80)
print(optimized_volatility.x)
print('-'*80)


# Print the Optimized Metrics.
print('')
print('='*80)
print('OPTIMIZED METRICS:')
print('-'*80)
print(optimized_metrics)
print('-'*80)
```

```
================================================================================
OPTIMIZED WEIGHTS:
--------------------------------------------------------------------------------
[0.38702188 0.44174147 0.09671761 0.07451904]
--------------------------------------------------------------------------------

================================================================================
OPTIMIZED METRICS:
--------------------------------------------------------------------------------
[0.09965198 0.1999139  0.49847452]
--------------------------------------------------------------------------------
```

Start coding or generate with AI.

```
from google.colab import drive
drive.mount('/content/drive')
```