CODE

```
# The Monte Carlo method is a computational algorithm that relies on
# repeated random sampling to obtain numerical results.
# In this simulation, we will use it to model the future performance of a
# stock portfolio. We will assume the daily returns of the stocks follow a
# multivariate normal distribution.

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
import yfinance as yf

# This function retrieves historical stock data from Yahoo Finance.
# It returns the mean daily returns and the covariance matrix of the stocks.
def get_data(stocks, start, end):
  """
  Downloads historical stock data, calculates returns, and computes the
  mean returns and covariance matrix.

  Args:
    stocks (list): A list of stock ticker symbols.
    start (datetime.datetime): The start date for the data.
    end (datetime.datetime): The end date for the data.

  Returns:
    tuple: A tuple containing:
      - meanReturns (pd.Series): The mean daily returns for each stock.
      - covMatrix (pd.DataFrame): The covariance matrix of the returns.
  """
  try:
    stockData = yf.download(stocks, start, end, auto_adjust=False)
    # We are only interested in the closing price
    stockData = stockData['Close']
    # Calculate the percentage change (daily returns)
    returns = stockData.pct_change()
    # Calculate the mean of the daily returns
    meanReturns = returns.mean()
    # Calculate the covariance matrix of the daily returns
    covMatrix = returns.cov()
    return meanReturns, covMatrix
```

```
    except Exception as e:
        print(f"An error occurred while fetching data: {e}")
        # Return empty dataframes in case of an error
        return pd.Series(), pd.DataFrame()

# Define the list of stock tickers to simulate
# We use '.TO' for Toronto Stock Exchange tickers
stockList = ['MX', 'RY', 'SU', 'T', 'FTS', 'BNT']
stocks = [stock + '.TO' for stock in stockList]

# Define the date range for historical data
endDate = dt.datetime.now()
startDate = endDate - dt.timedelta(days=365)

# Retrieve the historical data
meanReturns, covMatrix = get_data(stocks, startDate, endDate)

# Check if data was successfully retrieved before proceeding
if meanReturns.empty or covMatrix.empty:
    print("Could not retrieve stock data. Please check the stock tickers or date range.")
else:
    # Generate a set of random weights for the portfolio
    # The weights represent the proportion of the initial investment in each stock
    weights = np.random.random(len(meanReturns))
    weights /= np.sum(weights) # Normalize the weights so they sum to 1

    # --- Monte Carlo Simulation Setup ---
    # Number of simulations to run
    mc_sims = 1000
    # Timeframe for the simulation in days
    T = 100

    # Create a matrix of mean returns
    # np.full() creates an array of a given shape and fills it with the specified value
    meanM = np.full(shape=(T, len(weights)), fill_value=meanReturns)
    meanM = meanM.T

    # Create a matrix to store the results of the simulations
    portfolio_sims = np.full(shape=(T, mc_sims), fill_value=0.0)

    # Define the initial investment amount
    initialPortfolio = 100000

    # --- The Monte Carlo Simulation Loop ---
```

```
for m in range(0, mc_sims):
    # This is the core of the Monte Carlo simulation
    # 1. Generate a matrix of random numbers from a standard normal distribution
    Z = np.random.normal(size=(T, len(weights)))
    # 2. Use the Cholesky decomposition of the covariance matrix to introduce correlation
    # This ensures that the simulated returns for each stock have the same
    # correlation structure as the historical data.
    L = np.linalg.cholesky(covMatrix)
    # 3. Calculate the daily returns for this simulation using the mean returns
    # and the correlated random numbers.
    dailyReturns = meanM + np.inner(L, Z)
    # 4. Calculate the portfolio value for this simulation over time
    # np.inner() calculates the dot product of weights and daily returns
    # np.cumprod() calculates the cumulative product, simulating growth over time
    portfolio_sims[:,m] = np.cumprod(np.inner(weights, dailyReturns.T)+1)*initialPortfolio

# --- Plot the results ---
plt.style.use('fivethirtyeight')
plt.figure(figsize=(10, 6))
plt.plot(portfolio_sims)
plt.ylabel('Portfolio Value ($)', fontsize=14)
plt.xlabel('Days', fontsize=14)
plt.title('Monte Carlo Simulation of a Stock Portfolio', fontsize=18)
# Save the plot as a PNG file before displaying it
plt.savefig('monte_carlo_plot.png')
plt.show()

# --- Display the final portfolio values and statistics ---
# Get the final values from the last row of the simulation matrix
final_values = portfolio_sims[-1, :]
# Calculate and print the 95% confidence interval
ci_low = np.percentile(final_values, 2.5)
ci_high = np.percentile(final_values, 97.5)
print(f"Initial Portfolio Value: ${initialPortfolio:,.2f}")
print(f"Simulations run: {mc_sims}")
print(f"Timeframe: {T} days")
print(f"Final portfolio value 95% confidence interval: [${ci_low:,.2f}, ${ci_high:,.2f}]")
print(f"Average final portfolio value: ${final_values.mean():,.2f}")
print(f"Standard deviation of final values: ${final_values.std():,.2f}")
print("\nNote: The plot has also been saved to a file named 'monte_carlo_plot.png'.")
```

OUTPUT

C:\Users\d_dem\AppData\Local\Microsoft\WindowsApps\python3.13.exe
C:\Users\d_dem\OneDrive\Desktop\MonteCarloPythonStocks.py

[*********************100%***********************] 6 of 6 completed

Initial Portfolio Value: $100,000.00

Simulations run: 1000

Timeframe: 100 days

Final portfolio value 95% confidence interval: [$82,415.68, $129,032.02]

Average final portfolio value: $104,074.78

Standard deviation of final values: $12,328.18

Note: The plot has also been saved to a file named 'monte_carlo_plot.png'.

Process finished with exit code 0