In [1]:

```
import yfinance as yf
import datetime
import pandas as pd
import numpy as np
from scipy.stats import norm
import statsmodels.api as sm


# Define the stock symbols
symbols = ["RY.TO", "SHOP.TO", "CNQ.TO", "VCN.TO"]

# Define the start and end dates
start_date = datetime.datetime(2018, 5, 1)
end_date = datetime.datetime(2023, 4, 29)

# Download the stock data
data = yf.download(symbols, start=start_date, end=end_date)["Adj Close"]

# Print the downloaded data
print(data)
```

```
[*********************100%***********************]  4 of 4 completed
               CNQ.TO       RY.TO    SHOP.TO     VCN.TO
Date
2018-05-01  33.902885   79.370392  16.350000  27.247820
2018-05-02  34.234322   79.402901  16.099001  27.264994
2018-05-03  33.490417   79.362267  17.349001  27.239231
2018-05-04  33.645088   79.500404  17.495001  27.453918
2018-05-07  33.365208   80.256187  17.855000  27.574144
...               ...         ...        ...        ...
2023-04-24  80.814072  132.105591  65.019997  41.602375
2023-04-25  79.579285  130.413055  63.340000  41.106163
2023-04-26  79.292816  130.878265  63.279999  40.967224
2023-04-27  79.905273  132.422302  64.669998  41.314575
2023-04-28  81.554939  133.134949  65.639999  41.532906

[1254 rows x 4 columns]
```

In [2]:

```
# Calculate the percentage change
pct_returns = data.pct_change()

# Print the percentage change
print(pct_returns)
```

```python
# Define the initial investment for each stock
initial_investment = 25000.00

# Create an empty DataFrame to store the stock values
stock_values = pd.DataFrame(index=data.index)

# Iterate over the stock symbols and calculate the stock values
for symbol in symbols:
    stock_values[symbol] = (pct_returns[symbol] + 1).cumprod() * initial_investment
    stock_values.loc[start_date, symbol] = initial_investment

# Print the stock values
print(stock_values)
```

```
              CNQ.TO     RY.TO   SHOP.TO    VCN.TO
Date
2018-05-01       NaN       NaN       NaN       NaN
2018-05-02  0.009776  0.000410 -0.015352  0.000630
2018-05-03 -0.021730 -0.000512  0.077645 -0.000945
2018-05-04  0.004618  0.001741  0.008415  0.007882
2018-05-07 -0.008319  0.009507  0.020577  0.004379
...              ...       ...       ...       ...
2023-04-24  0.010125 -0.003807 -0.018862  0.000000
2023-04-25 -0.015279 -0.012812 -0.025838 -0.011927
2023-04-26 -0.003600  0.003567 -0.000947 -0.003380
2023-04-27  0.007724  0.011798  0.021966  0.008479
2023-04-28  0.020645  0.005382  0.014999  0.005285

[1254 rows x 4 columns]
                   RY.TO        SHOP.TO        CNQ.TO        VCN.TO
Date
2018-05-01  25000.000000   25000.000000  25000.000000  25000.000000
2018-05-02  25010.239602   24616.208800  25244.401142  25015.757010
2018-05-03  24997.440700   26527.523740  24695.845979  24992.119745
2018-05-04  25040.951200   26750.765185  24809.900221  25189.096373
2018-05-07  25279.006936   27301.221905  24603.516221  25299.404195
...                  ...            ...           ...           ...
2023-04-24  41610.475817   99418.952792  59592.325707  38170.370311
2023-04-25  41077.362851   96850.150879  58681.793336  37715.093515
2023-04-26  41223.894181   96758.405662  58470.551356  37587.616432
2023-04-27  41710.233239   98883.786942  58922.177572  37906.312639
2023-04-28  41934.701856  100366.969202  60138.641755  38106.631769

[1254 rows x 4 columns]
```

Value-at-Risk Calculations for Individual Securities

Historical Value-at-Risk at 95% and 99% Confidence Levels

In [3]:
```
# Define the confidence levels
confidence_levels = [0.95, 0.99]

# Define the initial investment amount in each security
investment_amount = 10000

# Calculate the HVaR
for symbol in symbols:
    print("Security:", symbol)
    returns = pct_returns[symbol].dropna()
    for confidence in confidence_levels:
        var = np.percentile(returns, (1 - confidence) * 100)
        hvar = -var * investment_amount
        print("1-Day HVaR ({}%):".format(confidence * 100), hvar)
    print()
```

```
Security: RY.TO
1-Day HVaR (95.0%): 158.45065509060126
1-Day HVaR (99.0%): 344.8108077840831

Security: SHOP.TO
1-Day HVaR (95.0%): 601.8191151678225
1-Day HVaR (99.0%): 1091.710540384408

Security: CNQ.TO
1-Day HVaR (95.0%): 356.39910605849576
1-Day HVaR (99.0%): 640.0848652525268

Security: VCN.TO
1-Day HVaR (95.0%): 137.63982696858247
1-Day HVaR (99.0%): 275.1044096269951
```

. Parametric Value-at-Risk at 95% and 99% Confidence Levels

In [4]:
```
# Define the confidence levels
confidence_levels = [0.95, 0.99]
```

```python
# Calculate the PVaR
for symbol in symbols:
    print("Security:", symbol)
    returns = pct_returns[symbol].dropna()
    std = returns.std()
    for confidence in confidence_levels:
        z_score = norm.ppf(1 - confidence)
        pvar = -z_score * std * investment_amount
        print("Parametric VaR ({}%):".format(confidence * 100), pvar)
    print()
```

```
Security: RY.TO
Parametric VaR (95.0%): 210.4067358659806
Parametric VaR (99.0%): 297.5822618167457

Security: SHOP.TO
Parametric VaR (95.0%): 649.8579713071781
Parametric VaR (99.0%): 919.1065303366046

Security: CNQ.TO
Parametric VaR (95.0%): 477.443579792514
Parametric VaR (99.0%): 675.2575661600388

Security: VCN.TO
Parametric VaR (95.0%): 185.38641831584874
Parametric VaR (99.0%): 262.195549232202
```

Value-at-Risk Calculations for the Portfolio

In [5]:

```python
# Calculate the portfolio value by summing the values of each stock
portfolio_values = stock_values.sum(axis=1)

# Print the portfolio values
print(portfolio_values)

# Calculate the percentage change
portfolio_returns = portfolio_values.pct_change()

# Print the portfolio return
print(portfolio_returns)

# Calculate the weight of each stock in the portfolio
```

```python
stock_weights = stock_values.div(portfolio_values, axis=0)

# Print the stock weights
print(stock_weights)
```

```
Date
2018-05-01    100000.000000
2018-05-02     99886.606555
2018-05-03    101212.930165
2018-05-04    101790.712979
2018-05-07    102483.149257
                  ...
2023-04-24    238792.124628
2023-04-25    234324.400581
2023-04-26    234040.467630
2023-04-27    237422.510393
2023-04-28    240546.944582
Length: 1254, dtype: float64
Date
2018-05-01         NaN
2018-05-02   -0.001134
2018-05-03    0.013278
2018-05-04    0.005709
2018-05-07    0.006803
                ...
2023-04-24   -0.006131
2023-04-25   -0.018710
2023-04-26   -0.001212
2023-04-27    0.014451
2023-04-28    0.013160
Length: 1254, dtype: float64
               RY.TO   SHOP.TO    CNQ.TO    VCN.TO
Date
2018-05-01  0.250000  0.250000  0.250000  0.250000
2018-05-02  0.250386  0.246442  0.252731  0.250442
2018-05-03  0.246979  0.262096  0.243999  0.246926
2018-05-04  0.246004  0.262802  0.243734  0.247460
2018-05-07  0.246665  0.266397  0.240074  0.246864
...              ...       ...       ...       ...
2023-04-24  0.174254  0.416341  0.249557  0.159848
2023-04-25  0.175301  0.413317  0.250430  0.160952
2023-04-26  0.176140  0.413426  0.249831  0.160603
2023-04-27  0.175679  0.416489  0.248174  0.159658
2023-04-28  0.174331  0.417245  0.250008  0.158417

[1254 rows x 4 columns]
```

In [6]:
```
# Define the initial investment for the portfolio
portfolio_initial_investment = 100000.00
```

```
# Get the weight of the last day (2023-04-28)
last_day_weights = stock_weights.loc['2023-04-28']

# Calculate the weighted return for each stock
stock_weighted_returns = pct_returns * last_day_weights

# Calculate the weighted return for the portfolio
portfolio_weighted_return = stock_weighted_returns.sum(axis=1)

# Create a new DataFrame for portfolio values
portfolio_adjusted_values = pd.DataFrame(index=pct_returns.index, columns=['Value'])

# Calculate the portfolio values for each day
for i in range(len(portfolio_adjusted_values)):
    if i == 0:
        # Set the initial value for the first day
        portfolio_adjusted_values.iloc[i] = portfolio_initial_investment
    else:
        portfolio_adjusted_values.iloc[i] = portfolio_adjusted_values.iloc[i-1] * (1 + portfolio_weighted_return.iloc[i

# Print the portfolio values
print(portfolio_adjusted_values)
```

```
                    Value
Date
2018-05-01       100000.0
2018-05-02   99620.994304
2018-05-03  102283.393194
2018-05-04  102919.385513
2018-05-07  103830.948923
...                   ...
2023-04-24  347209.892152
2023-04-25  340708.793103
2023-04-26  340296.943331
2023-04-27  345229.902146
2023-04-28  349785.274871

[1254 rows x 1 columns]
```

Historical Value-at-Risk of the Portfolio at 95% and 99% Confidence Levels

In [7]:
```
# Define the confidence levels
confidence_levels = [0.95, 0.99]

# Define the initial investment amount
```

```
portfolio_investment = 10000

# Calculate the HVaR
for confidence in confidence_levels:
    portfolio_returns = portfolio_returns.dropna()
    var = np.percentile(portfolio_returns, (1 - confidence) * 100)
    hvar = -var * portfolio_investment
    print("1-Day HVaR ({}%):".format(confidence * 100), hvar)
print()
```

```
1-Day HVaR (95.0%): 340.99458498769536
1-Day HVaR (99.0%): 606.2085179651153
```

Parametric Value-at-Risk of the Portfolio at 95% and 99% Confidence Levels

In [8]:

```
# Define the confidence levels
confidence_levels = [0.95, 0.99]

# Define the initial investment amount
portfolio_investment = 10000

# Calculate the PVaR
for confidence in confidence_levels:
    portfolio_returns = portfolio_returns.dropna()
    std = portfolio_returns.std()
    z_score = norm.ppf(1 - confidence)
    pvar = -z_score * std * portfolio_investment
    print("1-Day PVaR ({}%):".format(confidence * 100), pvar)
print()
```

```
1-Day PVaR (95.0%): 379.47870258766466
1-Day PVaR (99.0%): 536.7039708236819
```

Adjusted Portfolio Return (HVar & PVar)

In [9]:

```
# Define the confidence levels
confidence_levels = [0.95, 0.99]

# Define the initial investment amount
portfolio_investment = 10000

# Calculate the HVaR
for confidence in confidence_levels:
```

```
    var = np.percentile(portfolio_weighted_return, (1 - confidence) * 100)
    hvar = -var * portfolio_investment
    print("1-Day HVaR for adjusted portfolio ({}%):".format(confidence * 100), hvar)
print()
```

```
1-Day HVaR for adjusted portfolio (95.0%): 301.7430821380544
1-Day HVaR for adjusted portfolio (99.0%): 562.6241837124171
```

In [10]:

```
# Define the confidence levels
confidence_levels = [0.95, 0.99]

# Define the initial investment amount
portfolio_investment = 10000

# Calculate the PVaR
for confidence in confidence_levels:
    adjusted_std = portfolio_weighted_return.std()
    z_score = norm.ppf(1 - confidence)
    pvar = -z_score * adjusted_std * portfolio_investment
    print("1-Day PVaR for adjusted portfolio ({}%):".format(confidence * 100), pvar)
print()
```

```
1-Day PVaR for adjusted portfolio (95.0%): 344.9642165005178
1-Day PVaR for adjusted portfolio (99.0%): 487.8894744959699
```

Parametric Value-at-Risk with the S&P/TSX Composite Index as the Risk Factor

Calculating Parametric VaR at 95% and 99% Confidence Levels with S&P/TSX Composite Index

In [11]:

```
# Define the stock symbols
symbols_five= ["RY.TO", "SHOP.TO", "CNQ.TO", "VCN.TO", "^GSPTSE"]

# Define the start and end dates
start_date = datetime.datetime(2018, 5, 1)
end_date = datetime.datetime(2023, 4, 29)

# Download the stock data
data_with_SP = yf.download(symbols_five, start=start_date, end=end_date)["Adj Close"]
```

```python
# Print the downloaded data
print(data_with_SP)
```

```
[*********************100%***********************]  5 of 5 completed
               CNQ.TO      RY.TO    SHOP.TO     VCN.TO       ^GSPTSE
Date
2018-05-01  33.902874   79.370399  16.350000  27.247822  15618.900391
2018-05-02  34.234322   79.402901  16.099001  27.264996  15627.900391
2018-05-03  33.490417   79.362274  17.349001  27.239233  15621.500000
2018-05-04  33.645088   79.500404  17.495001  27.453918  15729.400391
2018-05-07  33.365200   80.256157  17.855000  27.574141  15808.599609
...               ...         ...        ...        ...           ...
2023-04-24  80.814072  132.105591  65.019997  41.602375  20676.699219
2023-04-25  79.579285  130.413055  63.340000  41.106163  20439.900391
2023-04-26  79.292816  130.878265  63.279999  40.967224  20366.699219
2023-04-27  79.905273  132.422302  64.669998  41.314575  20522.599609
2023-04-28  81.554939  133.134949  65.639999  41.532906  20636.500000

[1254 rows x 5 columns]
```

In [12]:
```python
# Calculate the returns
returns = data_with_SP.pct_change().dropna()

# Split the data into x and y variables
x_variable = returns["^GSPTSE"]
y_variables = returns[["RY.TO", "SHOP.TO", "CNQ.TO", "VCN.TO"]]

# Add a constant column to the x_variables
x_variables = sm.add_constant(x_variable)

# Perform separate regressions and extract coefficients
betas = {}
for column in y_variables:
    model = sm.OLS(y_variables[column], x_variables)
    results = model.fit()
    beta = results.params[1]  # Extract the coefficient for the independent variable (x)
    betas[column] = beta

# Print the beta coefficients
for symbol, beta in betas.items():
    print(f"Beta for {symbol}: {beta}")
```

```
Beta for RY.TO: 0.944065042615045
Beta for SHOP.TO: 1.51054031825421
Beta for CNQ.TO: 1.6904186073090786
Beta for VCN.TO: 0.9874635326790003
```

In [13]:

```
# Calculate the square of each beta and multiply by the variance of ^GSPTSE
sd_betas = {}
variance = x_variable.var()
for symbol, beta in betas.items():
    sd_beta = np.sqrt(beta ** 2 * variance)
    sd_betas[symbol] = sd_beta

# Print the adjusted beta values
for symbol, sd_beta in sd_betas.items():
    print(f"SD for {symbol}: {sd_beta}")
```

```
SD for RY.TO: 0.010744458254609002
SD for SHOP.TO: 0.017191545771496307
SD for CNQ.TO: 0.01923875086904657
SD for VCN.TO: 0.011238378952608378
```

In [14]:

```
# Define the confidence levels
confidence_levels = [0.95, 0.99]

# Define the initial investment amount
investment_amount = 10000

# Calculate the PVaR
for symbol, sd_beta in sd_betas.items():
    print("Security:", symbol)
    for confidence in confidence_levels:
        z_score = norm.ppf(1 - confidence)
        pvar = -z_score * sd_beta * investment_amount
        print("Parametric VaR ({}%):".format(confidence * 100), pvar)
    print()
```

```
Security: RY.TO
Parametric VaR (95.0%): 176.73061129722302
Parametric VaR (99.0%): 249.95347618330214

Security: SHOP.TO
Parametric VaR (95.0%): 282.77576415147945
Parametric VaR (99.0%): 399.9351595699624

Security: CNQ.TO
Parametric VaR (95.0%): 316.4492914496704
Parametric VaR (99.0%): 447.5602718340786

Security: VCN.TO
Parametric VaR (95.0%): 184.85488381252978
Parametric VaR (99.0%): 261.4437898406583
```

In [15]:

```python
weighted_beta_variance = {}

for date in returns.index:
    weighted_beta_var = 0

    for symbol, beta in betas.items():
        weight = stock_weights.loc[date, symbol]
        variance = x_variable.var()
        weighted_beta_var += beta ** 2 * weight ** 2 * variance

    weighted_beta_variance[date] = weighted_beta_var

# Calculate the average weighted beta variance
average_weighted_beta_variance = np.mean(list(weighted_beta_variance.values()))

weighted_beta_sd = np.sqrt(average_weighted_beta_variance)

print("portfolio SD:", weighted_beta_sd)
```

```
portfolio SD: 0.010265698315931836
```

In [16]:

```python
# Define the confidence levels
confidence_levels = [0.95, 0.99]

# Define the initial investment amount
portfolio_investment = 10000

# Calculate the PVaR
```

```
for confidence in confidence_levels:
    z_score = norm.ppf(1 - confidence)
    pvar = -z_score * weighted_beta_sd * portfolio_investment
    print("1-Day PVaR ({}%):".format(confidence * 100), pvar)
print()
```

```
1-Day PVaR (95.0%): 168.85571108150103
1-Day PVaR (99.0%): 238.81585452812666
```

In [17]:
```
# Calculate the squared weights
squared_adjusted_weights = np.square(last_day_weights)

# Calculate the squared betas
squared_adjusted_betas = {symbol: beta ** 2 for symbol, beta in betas.items()}

# Calculate the adjusted portfolio variance
weighted_adjusted_betas = squared_adjusted_weights * np.array(list(squared_adjusted_betas.values())) * variance

adjusted_portfolio_variance = np.sum(weighted_adjusted_betas)

# Caculate SD
adjusted_portfolio_sd = np.sqrt(adjusted_portfolio_variance)

print(f"Adjusted Portfolio SD: {adjusted_portfolio_sd}")
```

```
Adjusted Portfolio SD: 0.009014753551404539
```

Calculating Portfolio Parametric VaR with S&P/TSX Composite Index as the Risk Factor

In [18]:
```
# Define the confidence levels
confidence_levels = [0.95, 0.99]

# Define the initial investment amount
portfolio_investment = 10000

# Calculate the PVaR
for confidence in confidence_levels:
    z_score = norm.ppf(1 - confidence)
    pvar = -z_score * adjusted_portfolio_sd * portfolio_investment
    print("1-Day adjusted PVaR ({}%):".format(confidence * 100), pvar)
print()
```

```
1-Day adjusted PVaR (95.0%): 148.27950075101418
1-Day adjusted PVaR (99.0%): 209.71452759312066
```

Parametric VaR Using Exponentially Weighted Moving Average (EWMA)

Calculating EWMA Parametric VaR at 95% and 99% Confidence Levels for each Security

In [19]:

```
# Set the lambda (λ) value
lambda_ = 0.94

# Calculate the EWMA volatility for each security
ewma_volatility = pct_returns.ewm(alpha=1 - lambda_).std()

# Drop NaN values from the EWMA volatility DataFrame
ewma_volatility = ewma_volatility.dropna()

# Define the initial investment amount
investment_amount = 10000

# Calculate the PVaR
confidence_levels = [0.95, 0.99]

for security in ewma_volatility.columns:
    print("Security:", security)
    ewma_std = ewma_volatility[security].values[-1]  # Get the latest EWMA volatility
    for confidence in confidence_levels:
        z_score = norm.ppf(1 - confidence)
        ewma_pvar = -ewma_std * z_score * investment_amount
        print("EWMA PVaR ({}%):".format(confidence * 100), ewma_pvar)
    print()
```

```
Security: CNQ.TO
EWMA PVaR (95.0%): 267.793347068433
EWMA PVaR (99.0%): 378.74524117355384

Security: RY.TO
EWMA PVaR (95.0%): 129.25660111053975
EWMA PVaR (99.0%): 182.81007760948958

Security: SHOP.TO
EWMA PVaR (95.0%): 418.6481488590663
EWMA PVaR (99.0%): 592.1020661725761

Security: VCN.TO
EWMA PVaR (95.0%): 99.1750002041381
EWMA PVaR (99.0%): 140.2650953875444
```

In [20]:

```python
# Set the lambda (λ) value
lambda_ = 0.94

# Calculate the EWMA volatility for the portfolio
ewma_volatility = portfolio_returns.ewm(alpha=1 - lambda_).std()

# Drop NaN values from the EWMA volatility Series
ewma_volatility = ewma_volatility.dropna()

# Define the initial investment amount
investment_amount = 10000

# Calculate the PVaR for the portfolio
confidence_levels = [0.95, 0.99]
portfolio_ewma_pvar = {}

for confidence in confidence_levels:
    z_score = norm.ppf(1 - confidence)
    portfolio_pvar_ewma = -z_score * ewma_volatility[-1] * investment_amount
    portfolio_ewma_pvar[confidence] = portfolio_pvar_ewma

# Print the results
for confidence, pvar in portfolio_ewma_pvar.items():
    print("Portfolio EWMA PVaR ({}%):".format(confidence * 100), pvar)
```

```
Portfolio EWMA PVaR (95.0%): 219.43384957876177
Portfolio EWMA PVaR (99.0%): 310.34948100899356
```

Calculating EWMA Parametric VaR for the Portfolio at 95% and 99% Confidence Levels

In [21]:
```
# Set the lambda (λ) value
lambda_ = 0.94

# Calculate the EWMA volatility for the portfolio
ewma_volatility = portfolio_weighted_return.ewm(alpha=1 - lambda_).std()

# Drop NaN values from the EWMA volatility Series
ewma_volatility = ewma_volatility.dropna()

# Define the initial investment amount
investment_amount = 10000

# Calculate the PVaR for the portfolio
confidence_levels = [0.95, 0.99]
portfolio_ewma_pvar = {}

for confidence in confidence_levels:
    z_score = norm.ppf(1 - confidence)
    portfolio_pvar_adjusted_ewma = -z_score * ewma_volatility[-1] * investment_amount
    portfolio_ewma_pvar[confidence] = portfolio_pvar_adjusted_ewma

# Print the results
for confidence, pvar in portfolio_ewma_pvar.items():
    print("Portfolio EWMA PVaR ({}%):".format(confidence * 100), pvar)
```

```
Portfolio EWMA PVaR (95.0%): 220.5736342958308
Portfolio EWMA PVaR (99.0%): 311.9615002853422
```