Setting up

In [1]:
```
import yfinance as yf
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
```

Define functions

In [2]:
```
def compute_summary_stats(returns):
    summary_stats = returns.describe()
    summary_stats['Mean'] = returns.mean()
    summary_stats['Median'] = returns.median()
    summary_stats['Maximum'] = returns.max()
    summary_stats['Standard Deviation'] = returns.std()
    return summary_stats

def compute_annualized_stats(returns):
    annualized_mean = returns.mean() * 250
    annualized_std = returns.std() * (250 ** 0.5)
    return annualized_mean, annualized_std

def compute_cumulative_returns(returns):
    cumulative_returns = (1 + returns).cumprod()-1
    return cumulative_returns


def compute_regression_stats(returns, benchmark):
    X = sm.add_constant(benchmark.values)
    model = sm.OLS(returns.values, X).fit()
    beta = model.params[1]
    r_squared = model.rsquared
    return beta, r_squared

def return_summary_stats(selected_start_date, selected_end_date, selected_interval, target_list):
    all_stats = pd.DataFrame()
    cumulative_returns_df = pd.DataFrame()
```

```python
    daily_returns_df = pd.DataFrame()

    # Download the benchmark data (S&P/TSX Composite index)
    benchmark_data = yf.download(tickers=benchmark_ticker,
                                 start=selected_start_date,
                                 end=selected_end_date,
                                 interval=selected_interval,
                                 progress=False)

    #simple retrun
    benchmark = benchmark_data['Adj Close'].pct_change().dropna()

    for ticker in target_list:
        # Download data for the current ticker
        data = yf.download(tickers=ticker,
                           start=selected_start_date,
                           end=selected_end_date,
                           interval=selected_interval,
                           progress=False)

        #simple retrun
        all_returns = data['Adj Close'].pct_change().dropna()


        daily_returns_df[ticker] = all_returns


        #simple retrun cumulative return
        cumulative_returns = compute_cumulative_returns(all_returns)

        # Compute summary stats for the current ticker
        summary_stats = compute_summary_stats(all_returns)

        # Compute annualized statistics
        annualized_mean, annualized_std = compute_annualized_stats(all_returns)

        # Compute regression statistics
        beta, r_squared = compute_regression_stats(all_returns, benchmark)

        # Add annualized and regression statistics to the summary stats DataFrame
        summary_stats.loc['Annualized Mean'] = annualized_mean
        summary_stats.loc['Annualized Std'] = annualized_std
        summary_stats.loc['Beta'] = beta
        summary_stats.loc['R-squared'] = r_squared
```

```python
        all_stats = pd.concat([all_stats, summary_stats], axis=1)
        cumulative_returns_df[ticker] = cumulative_returns

    # Plot cumulative returns for all target stocks
    cumulative_returns_df.plot(figsize=(10, 6))
    plt.xlabel('Date')
    plt.ylabel('Cumulative Returns')
    plt.title('Cumulative Returns for Selected Tickers')
    plt.legend()
    plt.show()

    all_stats.columns = target_list
    return all_stats, cumulative_returns_df, daily_returns_df

def compute_portfolio_composition(initial_investment, portfolio_weights, daily_returns_df):
    # Add initial date and value
    initial_date = daily_returns_df.index[0] - pd.DateOffset(days=1)
    initial_value = np.array(portfolio_weights) * initial_investment
    initial_row = pd.DataFrame(index=[initial_date], columns=daily_returns_df.columns, data=[initial_value])
    daily_returns_df = pd.concat([initial_row, daily_returns_df])

    # Compute portfolio composition
    portfolio_value = pd.DataFrame(index=daily_returns_df.index, columns=daily_returns_df.columns)
    portfolio_value.iloc[0] = initial_value
    for i in range(1, len(portfolio_value)):
        portfolio_value.iloc[i] = portfolio_value.iloc[i-1] * np.exp(daily_returns_df.iloc[i])

    # Calculate the sum of the four securities for each date
    portfolio_value['Portfolio Value'] = portfolio_value.sum(axis=1)

    # Calculate the percentage of each security for each date
    portfolio_percentage = portfolio_value.iloc[:, :-1].div(portfolio_value['Portfolio Value'], axis=0)


    # Plot the stacked bar chart for the portfolio composition
    plt.figure(figsize=(10, 6))
    portfolio_percentage.plot(kind='bar', stacked=True)
    plt.xlabel('Year')
    plt.ylabel('Percentage')
    plt.title('Portfolio Composition')
    plt.legend()

    years = portfolio_percentage.index.year
    months = portfolio_percentage.index.strftime('%b')
```

```
    # Select a subset of years and months to display on the x-axis
    step_size = max(len(years) // 10, 1)
    visible_years = years[::step_size]
    visible_months = months[::step_size]

    # Set the x-axis tick labels to display the selected years and months
    plt.xticks(range(0, len(years), step_size), [f'{year}\n{month}' for year, month in zip(visible_years, visible_month

    plt.show()


    # Plot portfolio composition percentage for all target stocks
    portfolio_percentage.plot(figsize=(10, 6))
    plt.xlabel('Date')
    plt.ylabel('Percentage')
    plt.title('portfolio composition')
    plt.legend()
    plt.show()
    # Calculate log returns of the portfolio
    returns = (portfolio_value['Portfolio Value'] / portfolio_value['Portfolio Value'].shift(1)).dropna()

    portfolio_stats = compute_summary_stats(returns)
    annualized_mean, annualized_std = compute_annualized_stats(returns)
    portfolio_stats.loc['Annualized Mean'] = annualized_mean
    portfolio_stats.loc['Annualized Std'] = annualized_std


    return portfolio_stats,portfolio_value, portfolio_percentage, returns


def compute_constant_portfolio_composition(portfolio_percentage, daily_returns_df):
    # Extract the portfolio weights from the last row of the portfolio_percentage DataFrame
    portfolio_weights = portfolio_percentage.iloc[-1]

    # simple returns
    simple_returns_df = daily_returns_df

    # Compute portfolio composition and daily portfolio value
    portfolio_value = 100000 * (1 + (simple_returns_df * portfolio_weights).sum(axis=1)).cumprod()

    # Add the initial date and value to the DataFrame
    initial_date = pd.to_datetime('2018-05-01')
    portfolio_value = pd.concat([pd.Series([100000], index=[initial_date]), portfolio_value])

    # Calculate the return of the portfolio
```

```
    returns = (portfolio_value / portfolio_value.shift(1)).dropna()


    portfolio_stats = compute_summary_stats(returns)
    annualized_mean, annualized_std = compute_annualized_stats(returns)
    portfolio_stats.loc['Annualized Mean'] = annualized_mean
    portfolio_stats.loc['Annualized Std'] = annualized_std

    print(portfolio_stats)
```

In [3]:

```
target_list = ["RY.TO", "SHOP.TO", "CNQ.TO", "VCN.TO"]
benchmark_ticker = "^GSPTSE"
```

Summary statistics including regression resulte such as beta and R2

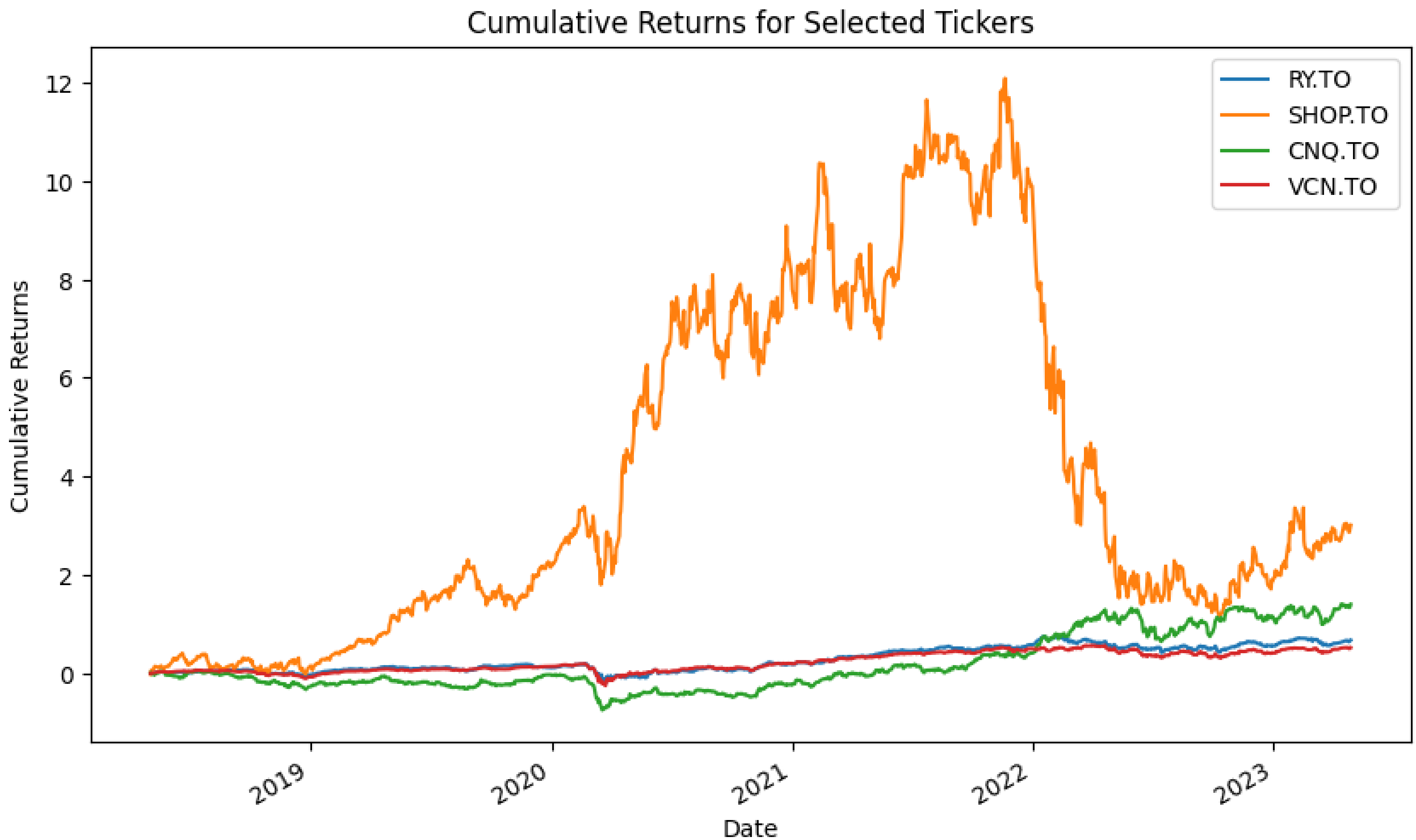
In [4]:

```
# Call return_summary_stats to get summary statistics, cumulative returns, and daily returns for all target stocks
summary_stats, cumulative_returns_df, daily_returns_df = return_summary_stats(selected_start_date='2018-05-01',
                                                                              selected_end_date='2023-04-29',
                                                                              selected_interval='1d',
                                                                              target_list=target_list)


# Print the summary statistics for all target stocks
print('--- Summary Statistics ---')
print(summary_stats)

# Print the daily returns and cumulative returns for all target stocks
print('\n--- Daily Returns ---')
print(daily_returns_df)

print('\n--- Cumulative Returns ---')
print(cumulative_returns_df)
```


Cumulative Returns for Selected Tickers
RY.TO
SHOP.TO
CNQ.TO
VCN.TO
Cumulative Returns
Date
0
2
4
6
8
10
12
2019
2020
2021
2022
2023

```
--- Summary Statistics ---
                         RY.TO      SHOP.TO       CNQ.TO       VCN.TO
count              1253.000000  1253.000000  1253.000000  1253.000000
mean                  0.000494     0.001892     0.001130     0.000401
std                   0.012792     0.039509     0.029027     0.011271
min                  -0.105383    -0.171028    -0.291804    -0.119372
25%                  -0.003937    -0.018862    -0.012035    -0.003623
50%                   0.000922     0.002810     0.000791     0.000943
75%                   0.005585     0.021776     0.014561     0.004869
max                   0.148963     0.171313     0.226124     0.122080
Mean                  0.000494     0.001892     0.001130     0.000401
Median                0.000922     0.002810     0.000791     0.000943
Maximum               0.148963     0.171313     0.226124     0.122080
Standard Deviation    0.012792     0.039509     0.029027     0.011271
Annualized Mean       0.123591     0.472888     0.282541     0.100141
Annualized Std        0.202257     0.624685     0.458949     0.178205
Beta                  0.944065     1.510540     1.690419     0.987463
R-squared             0.705511     0.189342     0.439303     0.994273

--- Daily Returns ---
               RY.TO   SHOP.TO    CNQ.TO    VCN.TO
Date
2018-05-02  0.000409 -0.015352  0.009776  0.000630
2018-05-03 -0.000511  0.077645 -0.021730 -0.000945
2018-05-04  0.001741  0.008415  0.004618  0.007881
2018-05-07  0.009506  0.020577 -0.008319  0.004379
2018-05-08  0.001215  0.038981 -0.019205  0.002180
...              ...       ...       ...       ...
2023-04-24 -0.003807 -0.018862  0.010125  0.000000
2023-04-25 -0.012812 -0.025838 -0.015279 -0.011927
2023-04-26  0.003567 -0.000947 -0.003600 -0.003380
2023-04-27  0.011798  0.021966  0.007724  0.008479
2023-04-28  0.005382  0.014999  0.020645  0.005285

[1253 rows x 4 columns]

--- Cumulative Returns ---
               RY.TO   SHOP.TO    CNQ.TO    VCN.TO
Date
2018-05-02  0.000409 -0.015352  0.009776  0.000630
2018-05-03 -0.000102  0.061101 -0.012166 -0.000315
2018-05-04  0.001638  0.070031 -0.007604  0.007564
2018-05-07  0.011160  0.092049 -0.015859  0.011976
2018-05-08  0.012389  0.134618 -0.034760  0.014182
...              ...       ...       ...       ...
```

```
2023-04-24  0.664419  2.976758  1.383694  0.526815
2023-04-25  0.643094  2.874006  1.347272  0.508604
2023-04-26  0.648955  2.870336  1.338823  0.503505
2023-04-27  0.668409  2.955351  1.356888  0.516253
2023-04-28  0.677388  3.014679  1.405546  0.524265

[1253 rows x 4 columns]
```

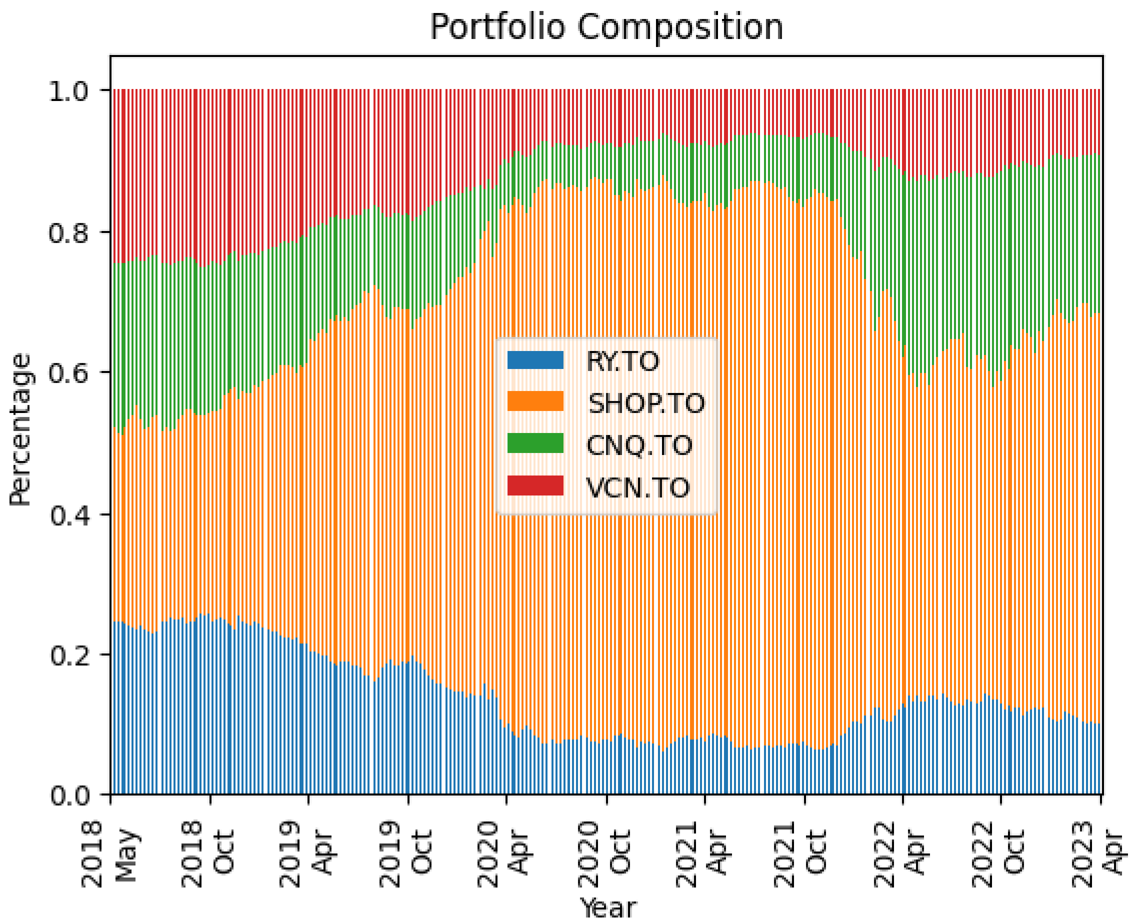
portfolio of $10,000 split equally

In [5]:

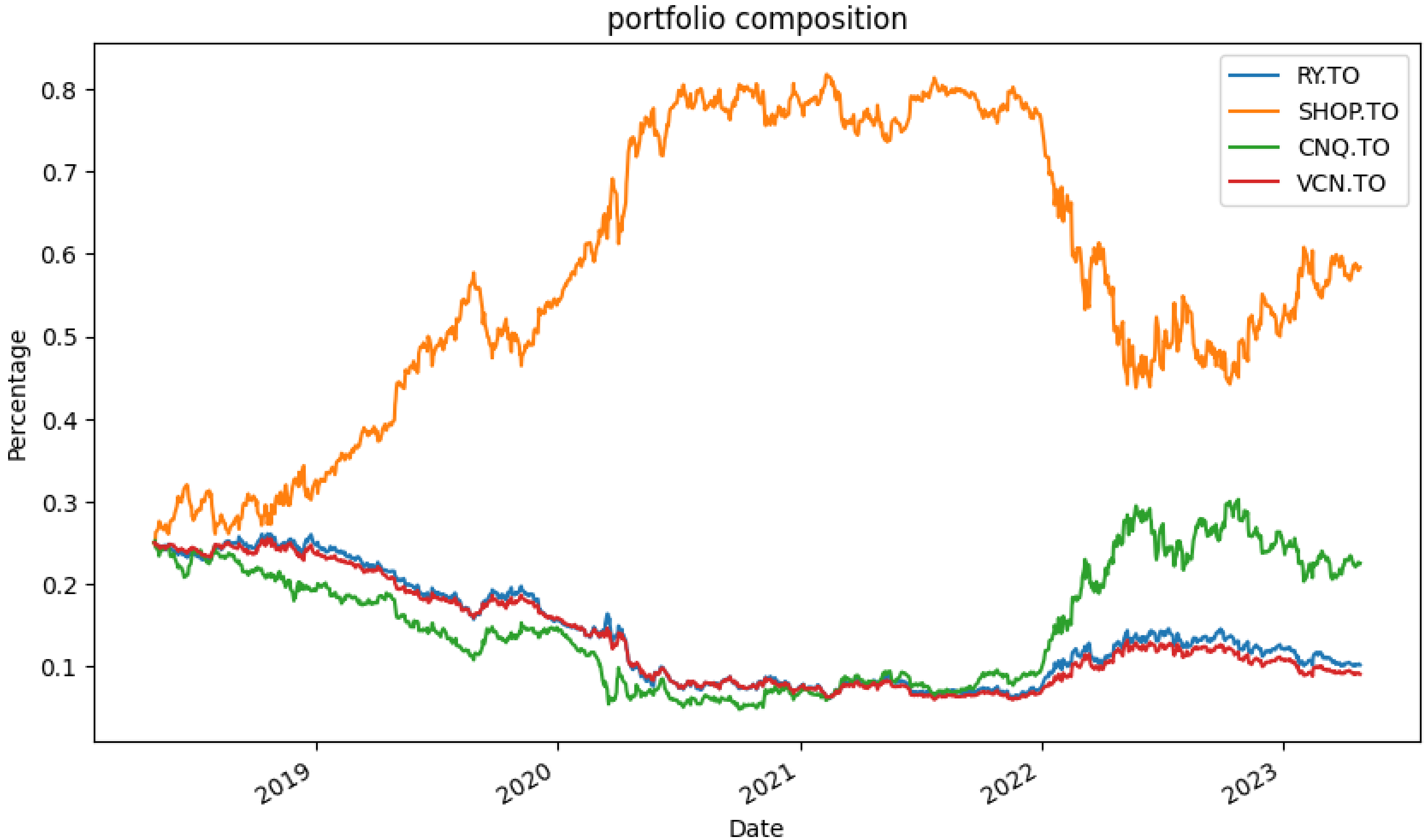
```
initial_investment = 100000
portfolio_weights = [0.25, 0.25, 0.25, 0.25]

portfolio_stats,portfolio_composition, portfolio_percentage, returns = compute_portfolio_composition(initial_investment
print("\n---Portfolio Stats---")
print(portfolio_stats)
print("\n---Portfolio Composition---")
print(portfolio_composition)
print("\n---Percentage of Each Security---")
print(portfolio_percentage)
print("\n---Log Returns of the Portfolio---")
print(returns)
```

```
<Figure size 1000x600 with 0 Axes>
```



Portfolio Composition
RY.TO
SHOP.TO
CNQ.TO
VCN.TO
Percentage
0.0
0.2
0.4
0.6
0.8
1.0
2018 May
2018 Oct
2019 Apr
2019 Oct
2020 Apr
2020 Oct
2021 Apr
2021 Oct
2022 Apr
2022 Oct
2023 Apr
Year



portfolio composition
RY.TO
SHOP.TO
CNQ.TO
VCN.TO
Percentage
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
2019
2020
2021
2022
2023
Date

```
---Portfolio Stats---
count                 1253.000000
mean                     1.001536
std                      0.025347
min                      0.863931
25%                      0.988477
50%                      1.001657
75%                      1.013139
max                      1.160309
Mean                     1.001536
Median                   1.001657
Maximum                  1.160309
Standard Deviation       0.025347
Annualized Mean        250.383959
Annualized Std           0.400764
Name: Portfolio Value, dtype: float64

---Portfolio Composition---
                   RY.TO        SHOP.TO         CNQ.TO        VCN.TO  \
2018-05-01       25000.0        25000.0        25000.0       25000.0
2018-05-02  25010.232081   24619.139696   25245.608261  25015.767231
2018-05-03   24997.44366   26606.851201   24702.938759  24992.139375
2018-05-04  25040.994458   26831.705021    24817.29245  25189.887373
2018-05-07  25280.173582   27389.546997   24611.700461   25300.44233
...                  ...            ...            ...           ...
2023-04-24  46080.124901  264756.488471  102052.086504  41360.615091
2023-04-25  45493.512698  258003.291118  100504.648357  40870.217098
2023-04-26   45656.08713  257759.002804  100143.503403  40732.308888
2023-04-27  46197.904766  263483.542175  100920.006263  41079.136429
2023-04-28  46447.194515  267465.384343  103025.183097  41296.797233

            Portfolio Value
2018-05-01    100000.000000
2018-05-02     99890.747270
2018-05-03    101299.372995
2018-05-04    101879.879303
2018-05-07    102581.863370
...                     ...
2023-04-24    454249.314966
2023-04-25    444871.669270
2023-04-26    444290.902225
2023-04-27    451680.589633
2023-04-28    458234.559188

[1254 rows x 5 columns]
```

```
---Percentage of Each Security---
               RY.TO   SHOP.TO    CNQ.TO    VCN.TO
2018-05-01      0.25      0.25      0.25      0.25
2018-05-02  0.250376  0.246461  0.252732  0.250431
2018-05-03  0.246768  0.262656  0.243861  0.246716
2018-05-04  0.245789  0.263366  0.243594  0.247251
2018-05-07  0.246439  0.267002  0.239923  0.246637
...              ...       ...       ...       ...
2023-04-24  0.101442  0.582844  0.224661  0.091053
2023-04-25  0.102262   0.57995  0.225918   0.09187
2023-04-26  0.102762  0.580158  0.225401  0.091679
2023-04-27   0.10228   0.58334  0.223432  0.090947
2023-04-28  0.101361  0.583687  0.224831  0.090122

[1254 rows x 4 columns]

---Log Returns of the Portfolio---
2018-05-02    0.998907
2018-05-03    1.014102
2018-05-04    1.005731
2018-05-07    1.006890
2018-05-08    1.006888
                ...
2023-04-24    0.990862
2023-04-25    0.979356
2023-04-26    0.998695
2023-04-27    1.016633
2023-04-28    1.014510
Name: Portfolio Value, Length: 1253, dtype: float64
```

using the last day's weight as rebalancing mertic

In [6]:
```
compute_constant_portfolio_composition(portfolio_percentage, daily_returns_df)
```

```
count                  1253.000000
mean                      1.001444
std                       0.025716
min                       0.828769
25%                       0.988752
50%                       1.002305
75%                       1.014958
max                       1.145056
Mean                      1.001444
Median                    1.002305
Maximum                   1.145056
Standard Deviation        0.025716
Annualized Mean         250.361095
Annualized Std            0.406605
dtype: float64
```