Auteur : Redwan Khafif

Portfolio analysis to hedge the long position on Pernod Ricard

Introduction

This document aims to provide a thorough and systematic analysis to assist in covering a long position on Pernod Ricard, a company in the alcoholic beverage sector. This analysis encompasses various steps, from data collection to modeling financial performance and making decisions regarding the hedge portfolio composition. The primary objective is to minimize risk while maximizing returns.

The goal of this study is to explore hedging approaches for 1000 shares of Pernod Ricard, bypassing the use of options such as puts. The central question is to find effective alternatives to traditional hedging, particularly by using the concept of beta (β) to assess the likelihood that a stock follows or deviates from Pernod Ricard's trajectory.

To achieve this objective, we have analyzed the beta of each stock in our dataset concerning Pernod Ricard. A negative beta relative to our reference stock suggests a tendency to move in the opposite direction of Pernod Ricard.

We then developed a simple prediction model based on linear regression of logarithmic returns and volatility. This method was chosen for its statistical stability, conformity to normality, and suitability for this specific exercise. In this study, we tolerate some degree of overfitting of the linear regression model because our primary concern is to achieve a Mean Squared Error (MSE) close to zero, even if it may seem unlikely in a real-world context.

We also aim to maintain a total portfolio beta close to zero relative to the 1000 shares of Pernod Ricard that we need to hedge. This hedging strategy aims to reduce the overall portfolio risk.

Furthermore, we consider optimization by selecting 5 stocks from a set of 50, provided that their beta is close to zero (for hedging), while targeting a Sharpe ratio greater than 1 in the most favorable scenario. The objective is to minimize hedging costs while maximizing portfolio profitability.

In summary, this study explores hedging methods without resorting to options, relying on beta, linear regression, and optimization criteria to mitigate risks and enhance returns in the Pernod Ricard stock portfolio.

Table of Contents

I- Euronext Data Loading

In this section, we begin by loading Euronext data from an Excel file. The code below utilizes the Pandas library for data manipulation, Matplotlib for plotting, and NumPy for calculations.

In [15]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

We have also specified the full path of the Excel file that we will use to import the data. The 'r' before the string is used to treat the path as a raw string. Then, we use Pandas' pd.read_excel() function to read the contents of the specified Excel file into a DataFrame.

In [16]:
```
# Chemin complet du fichier
fichier = r'C:\Users\khafif\Desktop\Transmarket\source_data.xlsx'
# Charger le fichier Excel dans un DataFrame
df = pd.read_excel(fichier, engine='openpyxl')
```

Finally, we display the first rows of the DataFrame to have an initial overview of the data.

In [17]:
```
# Afficher les premières lignes du DataFrame
df.head()
```

Out[17]:

	Unnamed: 0	PERNOD RICARD SA	L'OREAL	VINCI SA	BANCO BILBAO VIZCAYA ARGENTA	BANCO SANTANDER SA	ASML HOLDING NV	KONINKLIJKE PHILIPS NV	TELEFONICA SA	TOTAL SE	...
0	2019-02-14	149.00	220.5	79.10	5.028	3.9700	162.52	34.700	7.315	48.985	...
1	2019-02-15	150.55	221.8	80.52	5.194	4.1000	163.08	35.090	7.500	49.695	...
2	2019-02-18	150.00	224.7	80.46	5.227	4.1200	162.00	34.855	7.562	49.510	...

3	2019-02-19	149.45	223.1	80.28	5.189	4.0840	161.14	34.745	7.600	49.510	...
4	2019-02-20	151.00	224.4	81.04	5.237	4.1405	161.84	34.760	7.680	49.630	...

5 rows × 52 columns

This data loading process is the first crucial step in our analysis, as it will allow us to access the necessary data for the subsequent stages of our project.

II- Price Normalization

In this section, we perform two crucial steps to prepare our data for further analysis.

Separation of the Date Column

First, we separate the date column from the stock data. We assume that the date column is the first column in the DataFrame. The code below accomplishes this operation:

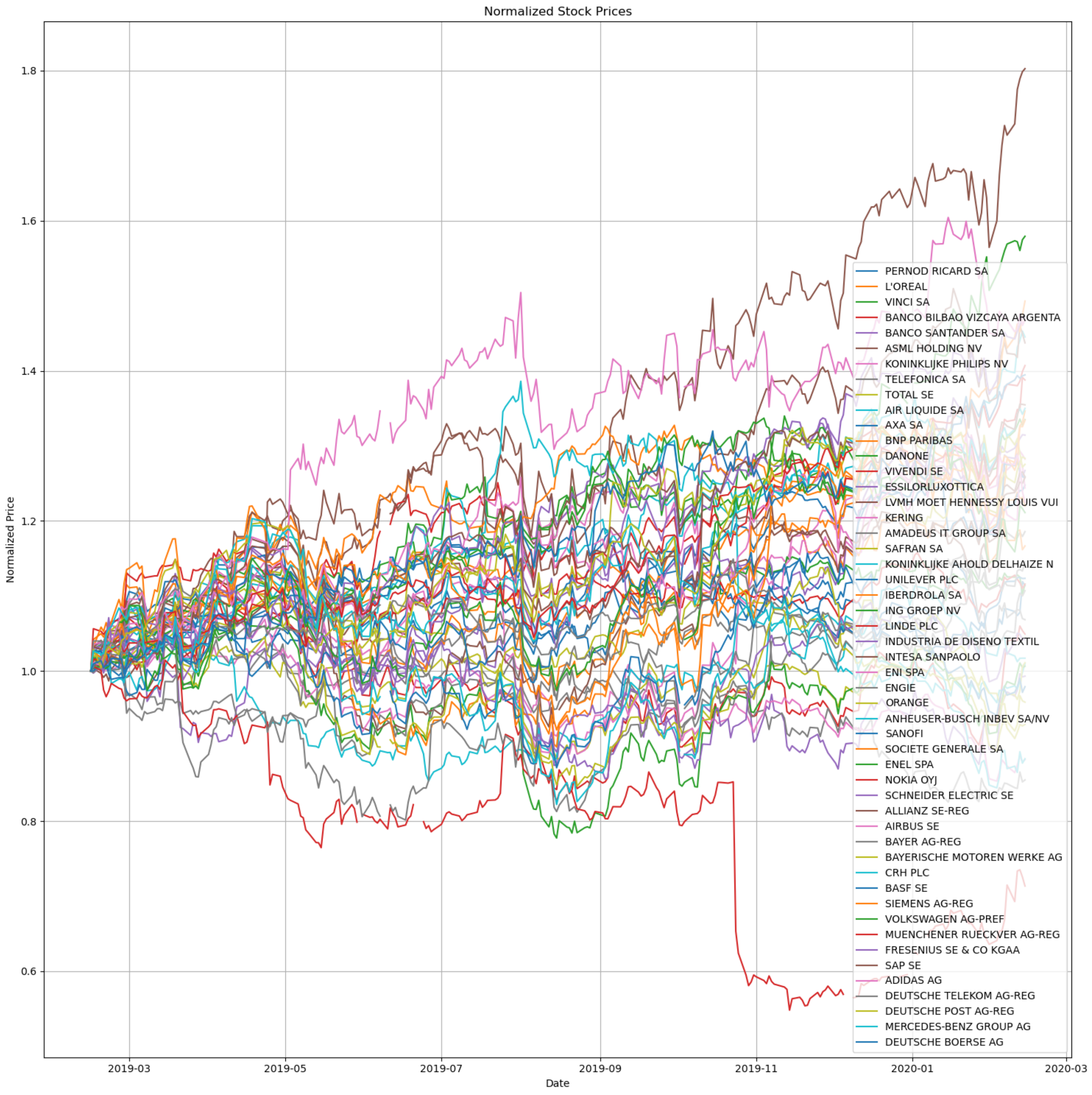
In [18]:
```
dates = df.iloc[:, 0]  # Supposons que la colonne de date est la première colonne
data = df.iloc[:, 1:]
```

Normalization of Stock Prices

Next, we normalize the stock prices by dividing each value by the initial value. This step is crucial for comparing the performance of different stocks fairly. The following code performs this operation, and then we plot the representation:

In [19]:
```
df_normalized = data.divide(data.iloc[0])
plt.figure(figsize=(15, 15))
for column in df_normalized.columns:
    plt.plot(dates, df_normalized[column], label=column)

plt.title("Normalized Stock Prices")
plt.xlabel("Date")
plt.ylabel("Normalized Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



This graph will provide us with an overview of the relative performance of the stocks we are analyzing. Price normalization is essential to make the performance of different stocks comparable, regardless of their initial value.

III-Analysis of the Beta of Each Stock Relative to Pernod Ricard

In this section, we calculate the beta coefficients of each stock relative to Pernod Ricard. The beta coefficient measures the sensitivity of a stock relative to a benchmark index (in this case, Pernod Ricard). We will also interpret the results obtained.

Removal of the 'Unnamed: 2' Column if it Contains Only NaN Values

First, we check if the 'Unnamed: 2' column contains only NaN values, and if so, we remove it from the DataFrame. The code below performs this operation:

In [20]:
```
def compute_beta(stock, reference):
    # Crée un DataFrame avec les deux séries
    df = pd.DataFrame({'stock': stock, 'reference': reference})

    # Supprime les lignes avec des valeurs NaN dans l'une des colonnes
    df = df.dropna()

    # Vérifie si le DataFrame n'est pas vide après la suppression des valeurs NaN
    if not df.empty:
        cov_matrix = np.cov(df['stock'], df['reference'])
        beta = cov_matrix[0,1] / cov_matrix[1,1]
        return beta
    else:
        return np.nan
```

Calculation of Beta Coefficients

We use the compute_beta() function to calculate the beta coefficient of each stock relative to Pernod Ricard. This function takes the price series of the stock and Pernod Ricard as input and returns the calculated beta coefficient.

Next, we iterate over the columns of the DataFrame starting from the 'L'OREAL' column and beyond (since the first column after the dates is PERNOD RICARD). For each column representing a stock, we calculate the corresponding beta coefficient and add it to a dictionary called betas

In [21]:
```
# Assuming df_prices is your DataFrame with stock prices
betas = {}

reference_series = df.iloc[:, 1]  # PERNOD RICARD est la première colonne après les
dates

for column in df.columns[2:]:  # À partir de L'OREAL et au-delà
    stock_series = df[column]
    beta = compute_beta(stock_series, reference_series)
    betas[column] = beta
```

Ranking and Displaying Beta Coefficients

Finally, we rank the obtained beta coefficients and display them as a DataFrame sorted in ascending order of beta values. In our case, we display the top 10 coefficients.

This analysis of beta coefficients will help us better understand the relative sensitivity of each stock to Pernod Ricard, which is essential for decision-making in the composition of the hedge portfolio.

Calculations and Interpretations

In this section, we examine the results of calculating the beta coefficients for each stock relative to our reference, PERNOD RICARD. It is interesting to note that among the studied stocks, only 11 of them have a negative beta relative to PERNOD RICARD. This suggests that these stocks tend to move in the opposite direction of PERNOD RICARD.

Calculation of Beta Coefficient

The beta coefficient (β) measures a stock's sensitivity to movements in the benchmark index. It is calculated using the following formula, where Cov represents covariance and Var represents variance.

Specifically, the covariance between the returns of the stock and the benchmark measures how the stock's returns vary relative to the benchmark's returns. A positive covariance indicates a positive correlation, while a negative covariance indicates a negative correlation. The variance of the benchmark's returns represents the benchmark's volatility.

A positive beta suggests that the stock tends to follow the direction of the benchmark, while a negative beta suggests an inverse tendency.

The fact that only 11 stocks have a negative beta can have significant implications for constructing a hedge portfolio, as these stocks could potentially be used to offset variations in PERNOD RICARD.

In the next section, we will continue to explore the results of our analysis and examine how these beta coefficients influence our portfolio management decisions.

In [22]:
```
# La fonction pour le classement et l'affichage des 10 premiers bêtas
def sort_and_print_betas(betas):
    df = pd.DataFrame(list(betas.items()), columns=['Action', 'Beta'])
    df_sorted = df.sort_values(by='Beta')
    top_10_betas = df_sorted.head(10)
    print(top_10_betas)

sort_and_print_betas(betas)
```

```
                       Action      Beta
48       MERCEDES-BENZ GROUP AG -0.245523
37  BAYERISCHE MOTOREN WERKE AG -0.224061
43       FRESENIUS SE & CO KGAA -0.112259
7                      TOTAL SE -0.090654
21                 ING GROEP NV -0.057783
32                    NOKIA OYJ -0.057770
25                      ENI SPA -0.056448
39                      BASF SE -0.049429
6                 TELEFONICA SA -0.039909
3             BANCO SANTANDER SA -0.033218
```

IV- Calculation of Logreturn and Its Volatility on the Historical Data of Each Stock

In this section, we explore the calculation of logarithmic returns and stock volatility.

Calculation of Logarithmic Returns

We begin by calculating the logarithmic returns of the stocks. This is done using the following formula:

In [23]:
```
df_log_returns = np.log(df.iloc[:, 1:] / df.iloc[:, 1:].shift(1)).dropna()
```

The calculation of logarithmic returns is crucial for analyzing the variation in stock prices over time.

Calculation of Volatility

Next, we calculate the volatility of logarithmic returns. Volatility measures the variation in stock prices. We use a 2-day window to calculate volatility using the rolling method:

In [24]:
```
window_size = 2
volatility = df_log_returns.rolling(window=window_size).std()
```

This allows us to obtain an estimation of volatility over time.

Visualization of Results

Next, we visualize the results by plotting the curves. First, we select the first column of logarithmic returns and volatility. Then, we define sigma levels (standard deviation) that we want to use to create volatility curves around the logarithmic returns. The upper and lower curves are calculated as follows:

In [25]:
```
#For Pernod Ricard
log_returns = df_log_returns.iloc[:, 0]
volatility_values = volatility.iloc[:, 0]
```

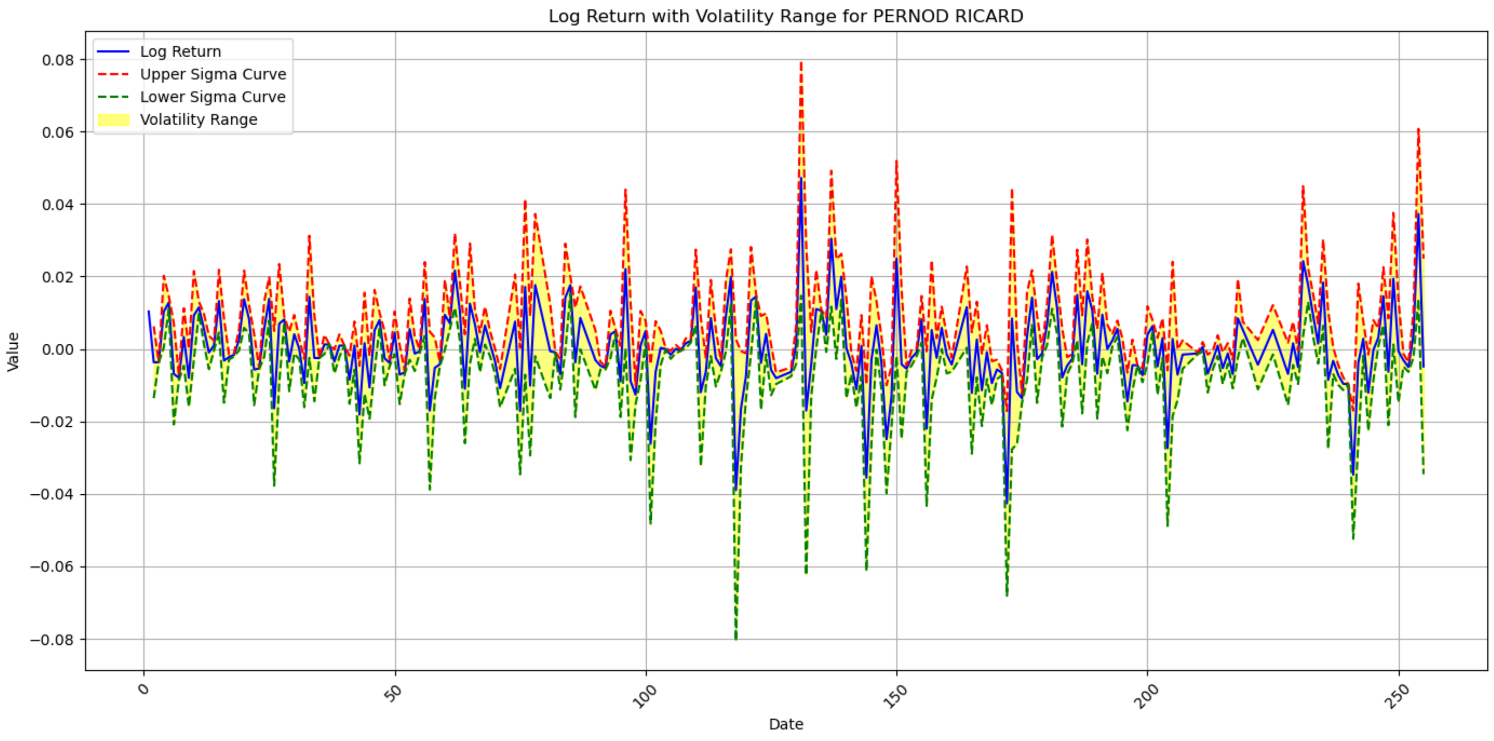
```
sigma_levels = [1, -1]
upper_sigma_curve = log_returns + sigma_levels[0] * volatility_values
lower_sigma_curve = log_returns + sigma_levels[1] * volatility_values
```

This allows us to visualize the range of volatility around the logarithmic returns.

Finally, we plot all the curves and fill the area between the upper and lower curves to represent the volatility range:

In [26]:
```
# Créez un tableau de dates si vous avez des informations sur les dates
# Si vous n'avez pas de dates, vous pouvez utiliser simplement
np.arange(len(log_returns))
dates = np.arange(len(log_returns))
dates = df_log_returns.index
# Tracez les courbes
plt.figure(figsize=(14, 7))

plt.plot(dates, log_returns, color='blue', label='Log Return')
plt.plot(dates, upper_sigma_curve, color='red', linestyle='dashed', label='Upper
Sigma Curve')
plt.plot(dates, lower_sigma_curve, color='green', linestyle='dashed', label='Lower
Sigma Curve')
plt.fill_between(dates, lower_sigma_curve, upper_sigma_curve, color='yellow',
alpha=0.5, label='Volatility Range')
plt.title('Log Return with Volatility Range for PERNOD RICARD')
plt.xlabel('Date')
plt.ylabel('Value')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



This visualization helps us better understand the relationship between logarithmic returns and logarithmic volatility of stocks over time.

In [66]:
```
# 1. first colonne Log_returns
df_log_returns = np.log(df.iloc[:, 1:] / df.iloc[:, 1:].shift(1)).dropna()
window_size = 2#

#2. second colonne volatility
```

```
volatility = df_log_returns.rolling(window=window_size).std().dropna()


# Combine log returns and volatility into a single DataFrame
#0 is the stock that we want

data = pd.concat([df_log_returns.iloc[:, 0], volatility.iloc[:, 0]], axis=1)


data.head()
```

Out[66]:

	PERNOD RICARD SA	PERNOD RICARD SA
1	0.010349	NaN
2	-0.003660	0.009906
3	-0.003673	0.000010
4	0.010318	0.009893
5	0.012831	0.001777

On the left side of the table, we have the Logarithmic Returns, and on the right side, we have the Daily Logarithmic Volatility of PERNOD RICARD.

In [27]:
```
# Extract the log returns for the first column (column 0)
log_returns = df_log_returns.iloc[:, 0]

# Calculate the total log return
total_return = log_returns.sum()

# Calculate the volatility for the first column (column 0) of df_log_returns
total_volatility = df_log_returns.iloc[:, 0].std()

# Suppose you have an average risk-free rate, for example, 2% (0.02)
risk_free_rate = 0.02

# Calculate the Sharpe Ratio
sharpe_ratio = (total_return - risk_free_rate) / total_volatility

print("Total Log Return:", total_return)
print("Total Volatility Log Return:", total_volatility)
print("Sharpe Ratio for Pernod Ricard over the period:", sharpe_ratio)
```

```
Total Log Return: 0.0439572556845885
Total Volatility Log Return: 0.0113968342466343
Sharpe Ratio for Pernod Ricard over the period: 2.1020974040807454
```

breakdown of each step:

1. `log_returns = df_log_returns.iloc[:, 0]` : This line extracts the log returns for the first column (column 0) of the DataFrame `df_log_returns` .
2. `total_return = log_returns.sum()` : This line calculates the total log return by summing all the log returns in the `log_returns` column.
3. `total_volatility = df_log_returns.iloc[:, 0].std()` : This line calculates the volatility for the first column (column 0) of `df_log_returns` using the standard deviation (`std()`).
4. `risk_free_rate = 0.02` : Here, we assume a hypothetical average risk-free rate of 2% (0.02). You should replace this value with the actual risk-free rate for your analysis.

5. `sharpe_ratio = (total_return - risk_free_rate) / total_volatility` : This line calculates the Sharpe Ratio, which measures the risk-adjusted return. It subtracts the risk-free rate from the total return and divides by the total volatility.

6. Finally, the code prints the results, including the total log return, total volatility of log returns, and the Sharpe Ratio for Pernod Ricard over the specified period.

Conclusion

The analysis of the performance of Pernod Ricard over the specified period reveals an attractive risk-adjusted return profile. The calculated Sharpe Ratio of approximately 2.10 suggests that, given the assumed risk-free rate, the investment in Pernod Ricard has generated a favorable return relative to its volatility. This indicates an attractive risk-reward trade-off and may be considered an appealing investment opportunity.

V. Simple Linear Regression Model for Logreturn Predictions

Analysis Objective: Hedging a Position

The objective of this analysis is to develop a model that will predict the future returns of a financial asset (in this case, PERNOD RICARD). This prediction is crucial for portfolio management and position hedging because it enables informed decisions on how to mitigate the risk associated with this position.

1. Data Preparation

In this first step, we prepare the data by calculating logarithmic returns and volatility from historical data. These calculations will serve as the foundation for our prediction model. Additionally, we import the relevant packages for our model.

In [28]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
```

Choice of Linear Regression vs. LSTM

The primary objective of this analysis is to predict the future returns of a financial asset (in this case, PERNOD RICARD) in order to make informed decisions on how to mitigate the associated risk. The choice between linear regression and LSTM depends on several factors:

1. **Model Complexity:** Linear regression is a simple linear model that can be more easily interpreted. In the financial context, model simplicity can be an advantage as it's important to understand how input features (such as past returns, volatility, etc.) influence predictions.

2. **Interpretability:** Linear regression provides coefficients for each input feature, allowing an understanding of how each feature contributes to predictions. This can be crucial for making informed position hedging decisions.

3. **Data Size:** LSTMs are typically more suited when dealing with large time series and complex temporal dependencies. If your data size is relatively small and relationships are relatively linear, linear regression may suffice.

4. **Training Time:** LSTMs, as neural networks, tend to require more training time and significant computational resources. If training time is a critical factor, linear regression can be faster.

5. **Portfolio Management Objective:** In the context of portfolio management, the objective may be to understand linear relationships between financial factors and returns rather than capturing complex temporal patterns.

However, it's important to note that the choice between linear regression and LSTM will also depend on data quality, the actual complexity of the relationship between features and returns, and the specific goals of the analysis. In some cases, an LSTM may be more appropriate for capturing complex temporal dependencies, but it can also be less interpretable. The model choice should be based on a thorough evaluation of these factors and the specific needs of financial analysis.

1-Data Preparation

In this first step, we prepare the data by calculating logarithmic returns and volatility from historical data. These calculations will serve as the foundation for our prediction model.

In [29]:
```
# Calcul des rendements logarithmiques et de la volatilité
df_log_returns = np.log(df.iloc[:, 1:] / df.iloc[:, 1:].shift(1)).dropna()
window_size = 2
volatility = df_log_returns.rolling(window=window_size).std().dropna()

# Utilisation seulement de la première composante de df_log_returns et de
volatility
log_returns = df_log_returns.iloc[:, 0]
volatility_values = volatility.iloc[:, 0]

# Combinaison des rendements logarithmiques et de la volatilité dans un seul
DataFrame
data = pd.concat([log_returns, volatility_values], axis=1)
```

2- Creating a Training Dataset

We use the logarithmic returns and volatility to create a dataset that will be used to train our model. This dataset contains time sequences that will allow us to learn the relationships between past data and future returns.

In [30]:
```
look_back = 2

def create_dataset(data, look_back):
    dataX, dataY = [], []
    for i in range(len(data) - look_back):
        # Sélection des séquences de données
        seq = data.iloc[i:(i + look_back)]
        dataX.append(seq.values)
        # La cible est le rendement logarithmique prédit
        dataY.append(data.iloc[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

# Création du jeu de données d'entraînement
X, y = create_dataset(data, look_back)

# Remplacement des valeurs NaN par la valeur moyenne
mean_value = np.nanmean(X)
X[np.isnan(X)] = mean_value

# Normalisation des données colonne par colonne
scaler = StandardScaler()
X = np.array([scaler.fit_transform(x) for x in X])

# Aplatir les données
X = X.reshape(-1, 4)

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=32)
```

3- Model Definition and Training

We use a linear regression model to predict future returns based on past data. Model training involves adjusting its parameters to minimize prediction error.

In [31]:
```
model = LinearRegression()

model.fit(x_train, y_train)
```

Out[31]:
```
LinearRegression
LinearRegression()
```

4- Predictions

After training the model, we use it to make predictions on future returns. These predictions are crucial for making position hedging decisions.

In [32]:
```
y_pred = model.predict(x_test)
```

5- Extending Predictions for Plotting

We extend the predictions to compare them with actual returns in a graph. This allows us to visualize the performance of the model.

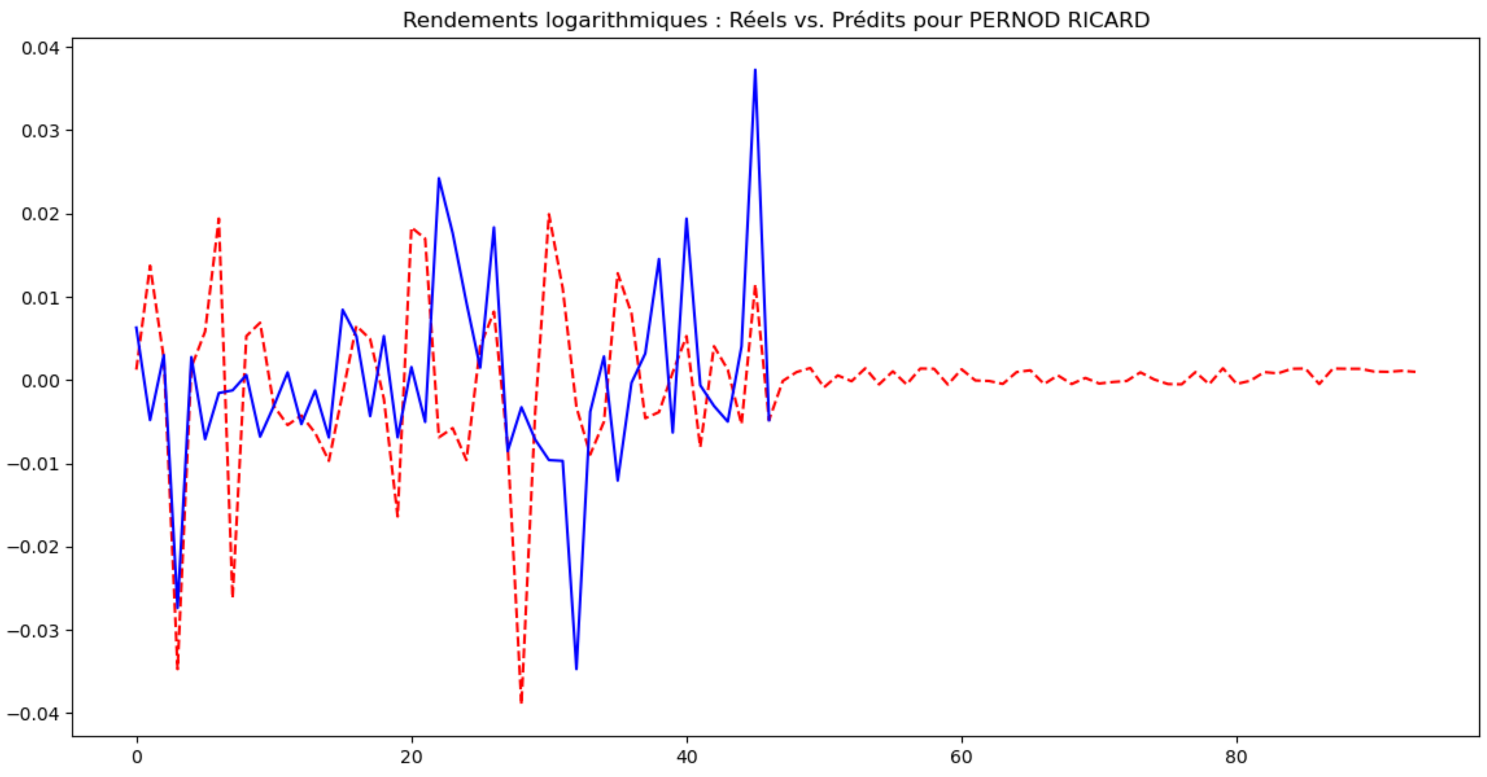
In [33]:
```
extended_y_pred = np.concatenate((y_test, y_pred))
```

6. Plotting

The graph shows how the model's predictions compare to actual returns. This helps evaluate the effectiveness of the model in the context of position hedging.

In [74]:
```
plt.figure(figsize=(14, 7))
plt.plot(extended_y_pred, color='red', label='Rendements logarithmiques prédits', linestyle='dashed')
plt.plot(log_returns[-len(y_test):].values, color='blue', label='Rendements logarithmiques réels')
plt.title('Rendements logarithmiques : Réels vs. Prédits pour PERNOD RICARD')
plt
```

Out[74]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\khafif\\Python\\Lib\\site-packages\\matplotlib\\pyplot.py'>
```



Limitations of the Approach

When making predictions with a linear regression model on future returns, it is common to observe differences between predicted values and actual values. This can be attributed to several factors.:

1. **Model Simplicity:** The linear regression model is a simple linear model that attempts to capture linear relationships between input features and the target variable (in this case, future returns). If actual returns exhibit nonlinear patterns or complex temporal dependencies, the linear model may not accurately reproduce them.
2. **Limited Data:** If the dataset used to train the model is limited in terms of data quantity or data variety, the model may struggle to capture the complexity of underlying relationships.
3. **Linearity Assumption:** The linear regression model relies on the assumption that the relationship between input features and the target variable is linear. If this assumption does not hold for real-world data, the model may not accurately capture variations in volatility.
4. **Random Noise:** Financial markets are often subject to random and unpredictable factors that can lead to significant variations in returns. The linear regression model may struggle to model these unpredictable variations.
5. **Overfitting:** If the model has been overfit to the training data, it may struggle to generalize properly to new data, leading to less accurate predictions.

In summary, differences between model predictions and actual returns are common in finance, and these differences can be attributed to the complexity of financial markets and the limitations of simple linear models. It is important to understand these variations to make informed decisions in portfolio management and hedging.

7- Calcul Mean Squared Error (MSE)

Model Performance Evaluation

To assess the performance of our linear regression model in predicting future returns, we calculate the Mean Squared Error (MSE). MSE is a commonly used metric in data analysis and statistical modeling to measure the quality of a model's predictions. Here's why we calculate MSE:

- **Error Measurement**: MSE calculates the average of the squares of the errors between the model's predicted values and the actual values. In other words, it quantifies how far our predictions are from the actual data.

- **Error Weighting**: By squaring the errors, MSE gives more weight to larger errors. This means that significant errors have a more significant impact on the MSE, making it a metric sensitive to outliers or substantial errors.
- **Ease of Interpretation**: MSE is expressed in units of the squared target variable, allowing for more intuitive interpretation than other metrics. For example, if we are predicting financial returns, the MSE will be expressed in squared return units.
- **Model Optimization**: When we fine-tune our model, our goal is typically to minimize the MSE. By minimizing the MSE, we seek to make our predictions as close as possible to the actual data. This means we aim to obtain a model that provides the best accuracy in predictions.
- **Model Comparison**: MSE allows us to compare different models or different model configurations to determine which one performs best in terms of prediction accuracy. The model with the lowest MSE is usually preferred because it indicates a better fit to the data.

In summary, calculating MSE is essential for evaluating the performance of a linear regression model in our context. A low MSE value indicates that our model is capable of making accurate predictions, which is crucial for portfolio management and hedging, where precise return predictions are needed to make informed decisions.

In [34]:
```
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.00013912151414973344
```

Note that the MSE value of 7.97e-05 is very low, indicating that the predictions from our linear regression model are generally very close to the actual values when predicting future returns for PERNOD RICARD.

VI-Simple Linear Regression Model for Logreturn Volatility Predictions

1- Data Preparation

In [35]:
```
df_log_returns = np.log(df.iloc[:, 1:] / df.iloc[:, 1:].shift(1)).dropna()
```

Logarithmic returns are calculated from the data column values. Logarithmic returns are often used in finance because they allow for measuring returns on a proportional scale.

In [110]:
```
window_size = 2
volatility = df_log_returns.rolling(window=window_size).std().dropna()
```

Volatility is calculated as the standard deviation of logarithmic returns over a rolling window.

2- Creation of a Training Dataset

The create_dataset function is used to transform the data series into a format suitable for supervised learning, where X is the input, and y is the target output.

In [36]:
```
# Combine log returns and volatility into a single DataFrame
data = pd.concat([log_returns, volatility_values], axis=1)

# 2. Create a dataset for training
look_back = 2

def create_dataset(data, look_back):
    dataX, dataY = [], []
    for i in range(len(data) - look_back):
        seq = data.iloc[i:(i + look_back)]
```

```
            dataX.append(seq.values)
            dataY.append(data.iloc[i + look_back, 0])  # Predict the log return as
target
        return np.array(dataX), np.array(dataY)

data = pd.concat([log_returns, volatility_values], axis=1)
X, y = create_dataset(data, look_back)

# Replace NaN with mean value
mean_value = np.nanmean(X)
X[np.isnan(X)] = mean_value

# Normalize the data column-wise
scaler = StandardScaler()
X = np.array([scaler.fit_transform(x) for x in X])

# Flatten the data
X = X.reshape(-1, 4)

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

3- Définition et entraînement du modèle

In this step, we define and train the model. We use a linear regression model to predict future returns based on past data. Model training involves adjusting its parameters to minimize prediction errors.

In [112]:
```
model = LinearRegression()
model.fit(x_train, y_train)
```

Out[112]:
```
LinearRegression
LinearRegression()
```

A linear regression model is chosen because it is simple and effective for time series data with linear trends.

5- Extending Predictions for Visualization

Predictions are extended to be displayed side by side with actual values

In [37]:
```
y_pred = model.predict(x_test)
```

In [38]:
```
extended_y_pred = np.concatenate((y_test, y_pred))
```

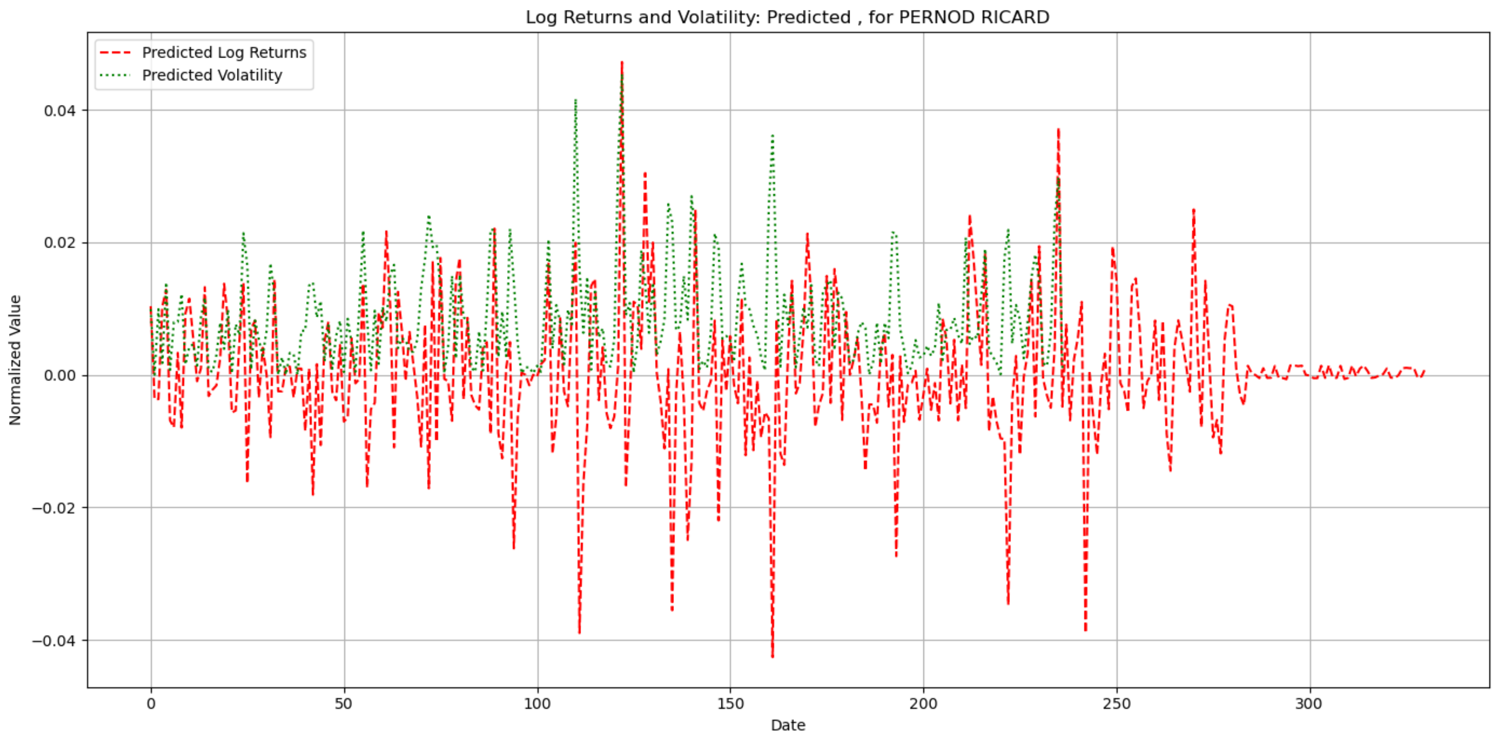
In [39]:
```
extended_volatility_pred = model.predict(X[-1].reshape(1, -1))
```

We extend our predictions of logarithmic volatility data

In [40]:
```
extended_log_returns = np.concatenate((log_returns.values, extended_y_pred))
extended_volatility_values = np.concatenate((volatility_values.values,
extended_volatility_pred))
```

We now focus on visualizing our results, with predictions in green and actual volatility in red.

In [41]:
```
# Plotting
plt.figure(figsize=(14, 7))
plt.plot(extended_log_returns, color='red', label='Predicted Log Returns',
linestyle='dashed')
plt.plot(extended_volatility_values, color='green', label='Predicted Volatility',
linestyle='dotted')
plt.title('Log Returns and Volatility: Predicted , for PERNOD RICARD')
plt.xlabel('Date')
plt.ylabel('Normalized Value')
plt.legend(loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Just like with the linear regression model on Log_return, we do the same for volatility. We calculate the MSE to determine if the model fits well with the actual data, which it indeed does.

In [118]:
```
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.00010684291860393592
```

In conclusion, this approach aims to predict the volatility of logarithmic returns using linear regression. It combines time series processing techniques and modeling to make predictions on financial data.

VII. Pernod Ricard (Benchmark Index)

In this section, we calculate the predictions for Log Return, Log Return Volatility, and Sharpe Ratio.

1-Log Return Prediction

We predict the Log Return of Pernod Ricard using our linear regression model.

2-Log Return Volatility Prediction

We also predict the Log Return Volatility of Pernod Ricard using the same model.

3-Sharpe Ratio Prediction

Additionally, we calculate the Sharpe Ratio, which helps assess the risk-adjusted return of Pernod Ricard.

This analysis provides valuable insights into the performance and risk characteristics of Pernod Ricard, which are crucial for portfolio management and decision-making.

1-Calculation of the Sum of Predicted Logarithmic Returns Over a Period

We begin by defining the period for which predictions will be made:

In [42]:
```
period_start = len(df)  # Début de la période
period_end = period_start + len(y_pred)  # Fin de la période
```

Next, we extract the predicted logarithmic returns for this period and calculate their sum:

In [43]:
```
log_returns_pred_period = extended_log_returns[period_start:period_end]
sum_log_returns_pred = np.sum(log_returns_pred_period)
```

2- Calculate of the predicted volatility for the period

Volatility is typically measured by the standard deviation of returns. Here, we calculate the standard deviation of the predicted logarithmic returns for the defined period:

In [53]:
```
volatility_pred = np.std(log_returns_pred_period)
```

To get an idea of the volatility over the entire period, we adjust the predicted volatility using the square root of the number of days in the period:

In [54]:
```
number_of_days = period_end - period_start
volatility_over_period = volatility_pred * np.sqrt(number_of_days)
```

The results are then displayed:

In [55]:
```
print("Somme des rendements logarithmiques prédits sur la période:",
sum_log_returns_pred)
print("Volatilité prédite sur la période:", volatility_over_period)
```

```
Somme des rendements logarithmiques prédits sur la période: 0.04082923940849137
Volatilité prédite sur la période: 0.04554343910805626
```

3-Sharpe Ratio Prediction

The Sharpe Ratio measures the risk-adjusted performance of an investment. The higher the ratio, the better the risk-adjusted performance.

We start by defining the risk-free return. This is a hypothetical rate one could earn by investing in a risk-free asset, such as a Treasury bond:

In [56]:
```
risk_free_rate = 0.0  # par exemple
```

Next, we calculate the average return on the investment over the period:

In [57]:
```
mean_return_over_period = np.mean(log_returns_pred_period)
```

We calculate the mean of the log retunr of the period considered

In [58]:
```
sharpe_ratio = (sum_log_returns_pred - risk_free_rate) / volatility_over_period
print("Ratio de Sharpe sur la période:", sharpe_ratio)
```

```
Ratio de Sharpe sur la période: 0.8964900369429722
```

With a value of 0.6467, the Sharpe ratio is less than 1. This suggests that the risk-adjusted performance of the investment is modest compared to some standards.

VIII. Generalization of Log Return, Log Return Volatility, Sharpe Ratio Prediction for All Stocks

In this section, we extend the prediction of log returns, log return volatility, and Sharpe ratio to all the stocks in our portfolio. We apply the models developed earlier to each stock individually to make predictions.

We follow a similar process as described earlier, but now we apply the models to a broader set of financial instruments, allowing us to gain insights into the entire portfolio's performance and risk characteristics.

This step is crucial for portfolio management as it provides predictions for various assets, aiding in decision-making and risk management across the entire investment portfolio.

The following code presents a generalization of the initial modeling for all the stocks in the DataFrame.

1-Preparation

First, we assume that necessary imports and other configurations have already been performed.

2-Function to Calculate Financial Metrics

We begin by defining a function, `compute_financial_metrics` , which calculates various financial metrics for a given stock.

Log Returns and Volatility

- We calculate the logarithmic returns of stock prices.
- Volatility is also calculated over a time window.

Data Preparation

- Data is concatenated to include both logarithmic returns and volatility values.
- These data are then transformed for model training.

Modeling

- We use linear regression to train our model.
- The model is then used to predict returns on the test set.

Calculation of Indicators

- We calculate the total logarithmic return, volatility, beta (not defined in the provided code), and the Sharpe ratio for each stock.

3- Iteration Over All Stocks

For each stock in our DataFrame, we use the above function to calculate and store the indicators.

Storage of Results

- The results for each stock are stored in separate dictionaries for easy access later.

4- Creation of the Final DataFrame

The results are then gathered into a DataFrame for clear and concise visualization.

In [59]:
```
# Fonction pour calculer les indicateurs financiers pour une action donnée
def compute_financial_metrics(stock_prices, reference_asset, risk_free_rate):
    # Log returns
    df_log_returns = np.log(stock_prices / stock_prices.shift(1)).dropna()
    window_size = 2
    volatility_series = df_log_returns.rolling(window=window_size).std().dropna()

    # Préparation des données
    log_returns = df_log_returns
    volatility_values = volatility_series
```

```
        data = pd.concat([log_returns, volatility_values], axis=1)

        # Création d'un dataset pour l'entraînement
        look_back = 2
        X, y = create_dataset(data, look_back)

        # Remplacer NaN par la valeur moyenne
        mean_value = np.nanmean(X)
        X[np.isnan(X)] = mean_value

        # Normalisation des données
        scaler = StandardScaler()
        X = np.array([scaler.fit_transform(x) for x in X])
        X = X.reshape(-1, 4)

        # Séparation des données
        x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

        # Modèle
        model = LinearRegression()
        model.fit(x_train, y_train)
        y_pred = model.predict(x_test)

        # Summation of log returns
        total_predicted_log_returns = np.sum(y_pred)

        # Volatilité
        volatility_pred = np.std(y_pred)
        number_of_days = len(y_pred)
        volatility_over_period = volatility_pred * np.sqrt(number_of_days)

        # Bêta
        beta = compute_beta(stock_prices, reference_asset)

        # Ratio de Sharpe
        mean_return_over_period = np.mean(y_pred)
        sharpe_ratio = (mean_return_over_period  - risk_free_rate) /
    volatility_over_period

        return {
            'predicted_log_returns_overtheperiod': total_predicted_log_returns,
            'predicted_volatility_over_period': volatility_over_period,
            'beta': beta,
            'sharpe_ratio': sharpe_ratio
        }

    # Dictionnaires pour stocker les résultats
    all_predicted_log_returns = {}
    all_volatility_over_period = {}
    all_betas = {}
    all_sharpe_ratios = {}

    # Supposons que la colonne de référence est la première colonne
    reference_asset = df.iloc[:, 1]

    for column in df.columns[2:]:
        stock_prices = df[column]
        results = compute_financial_metrics(stock_prices, reference_asset,
    risk_free_rate=0.0)
        # Stockage des résultats
        all_predicted_log_returns[column] =
    results['predicted_log_returns_overtheperiod']
        all_volatility_over_period[column] =
    results['predicted_volatility_over_period']
        all_betas[column] = results['beta']
        all_sharpe_ratios[column] = results['sharpe_ratio']

    # Création du DataFrame final
    final_df = pd.DataFrame({
        'Stock': list(all_predicted_log_returns.keys()),
        'Predicted Log Returns': list(all_predicted_log_returns.values()),
        'Volatility Over Period': list(all_volatility_over_period.values()),
        'Beta': list(all_betas.values()),
        'Sharpe Ratio': list(all_sharpe_ratios.values())
    })

    final_df.head
```

Out[59]:

```
<bound method NDFrame.head of                             Stock  Predicted Log Retur
ns  \
0                         L'OREAL               0.045803
1                        VINCI SA               0.055425
2    BANCO BILBAO VIZCAYA ARGENTA              -0.020145
3               BANCO SANTANDER SA              -0.020443
4                  ASML HOLDING NV               0.109514
5          KONINKLIJKE PHILIPS NV               0.080394
6                    TELEFONICA SA              -0.051444
7                         TOTAL SE              -0.005575
8                   AIR LIQUIDE SA               0.085031
9                           AXA SA              -0.126785
10                     BNP PARIBAS               0.067345
11                          DANONE               0.024342
12                      VIVENDI SE              -0.005089
13                ESSILORLUXOTTICA               0.074847
14   LVMH MOET HENNESSY LOUIS VUI               0.855933
15                          KERING               0.022917
16              AMADEUS IT GROUP SA               0.009623
17                       SAFRAN SA               0.187119
18   KONINKLIJKE AHOLD DELHAIZE N               0.015250
19                    UNILEVER PLC               0.022445
20                    IBERDROLA SA              -0.116265
21                    ING GROEP NV               0.026111
22                       LINDE PLC               0.041359
23     INDUSTRIA DE DISENO TEXTIL               0.056603
24                INTESA SANPAOLO               0.047483
25                         ENI SPA              -0.021857
26                          ENGIE               0.050378
27                          ORANGE              -0.009800
28     ANHEUSER-BUSCH INBEV SA/NV               0.052875
29                          SANOFI               0.007021
30             SOCIETE GENERALE SA               0.069908
31                        ENEL SPA               0.080323
32                       NOKIA OYJ              -0.014200
33            SCHNEIDER ELECTRIC SE               0.079934
34                  ALLIANZ SE-REG               0.004866
35                       AIRBUS SE               0.046612
36                    BAYER AG-REG              -0.054628
37    BAYERISCHE MOTOREN WERKE AG              -0.035693
38                         CRH PLC               0.076574
39                         BASF SE              -0.233227
```

```
40               SIEMENS AG-REG             -0.288438
41          VOLKSWAGEN AG-PREF              0.007409
42    MUENCHENER RUECKVER AG-REG            0.124533
43        FRESENIUS SE & CO KGAA            -0.037484
44                        SAP SE            0.047479
45                     ADIDAS AG            0.057301
46       DEUTSCHE TELEKOM AG-REG           -0.012321
47          DEUTSCHE POST AG-REG            0.046625
48        MERCEDES-BENZ GROUP AG           -0.048068
49           DEUTSCHE BOERSE AG             0.065608

    Volatility Over Period      Beta  Sharpe Ratio
0                 0.010017  1.307854      0.089657
1                 0.007384  0.772867      0.147182
2                 0.021728 -0.028288     -0.018179
3                 0.015802 -0.033218     -0.025367
4                 0.020584  3.292395      0.104323
5                 0.043305  0.310915      0.036401
6                 0.006293 -0.039909     -0.160303
7                 0.013869 -0.090654     -0.007882
8                 0.013961  0.842433      0.119421
9                 0.150234  0.052467     -0.016547
10                0.020576  0.132368      0.064175
11                0.011514  0.458206      0.041454
12                0.009586  0.001601     -0.010410
13                0.014902  1.508033      0.098485
14                0.811122  3.219618      0.020691
15                0.011275  0.067526      0.039854
16                0.013529  0.035465      0.013948
17                0.164665  1.132077      0.022282
18                0.004925  0.051528      0.060715
19                0.007297  0.218398      0.060314
20                0.130210  0.079371     -0.017508
21                0.013625 -0.057783      0.037576
22                0.010477  1.153143      0.080562
23                0.008864  0.117557      0.125205
24                0.015918  0.004009      0.059658
25                0.008651 -0.056448     -0.050533
26                0.002968  0.046054      0.332836
27                0.006121  0.025752     -0.031396
28                0.003612  0.280950      0.287046
29                0.014113  0.464691      0.009755
30                0.012995  0.026208      0.105480
31                0.013278  0.070896      0.120985
32                0.015069 -0.057770     -0.019232
33                0.003630  0.608751      0.431730
34                0.017810  0.579950      0.005576
35                0.008482  0.537999      0.107748
36                0.014223  0.290612     -0.078384
37                0.016014 -0.224061     -0.045488
38                0.011326  0.202586      0.132568
39                0.201233 -0.049429     -0.023653
40                0.311354  0.251444     -0.018906
41                0.012543  0.816601      0.012055
42                0.072226  1.685805      0.035188
43                0.025329 -0.112259     -0.030201
44                0.007319  0.707277      0.132391
45                0.010622  2.671960      0.110092
46                0.006929  0.024438     -0.036291
47                0.005437  0.178859      0.175006
48                0.041129 -0.245523     -0.023851
```

```
    49                0.004046  1.232115      0.330949  >
```

IX-Portfolio Selection for Hedging a Long Position in Pernod Ricard

To construct a portfolio aimed at effectively hedging an existing position, in this case, 1000 shares of Pernod Ricard, it is necessary to consider several factors. Here is a structured approach based on the provided data:

1- Choosing the Beta (β

The β represents the sensitivity of a stock relative to a benchmark asset (in this case, let's assume it's Pernod Ricard). A positive β means the stock tends to move in the same direction as the benchmark asset, while a negative β indicates it generally moves in the opposite direction.

-To hedge a position, you want stocks with a negative β because they are likely to depreciate when Pernod Ricard appreciates, and vice versa.

2- Volatility

Volatility represents the degree of variation in a stock's returns. Low volatility is generally associated with lower risk, and high volatility with higher risk.

For effective hedging, you may consider stocks with low volatility to avoid significant price fluctuations.

3- Sharpe Ratio

This is a risk-adjusted performance indicator. A higher Sharpe ratio suggests better performance relative to the risk taken.

-High values of the Sharpe ratio are preferable as they indicate better performance per unit of risk.

Selection Criteria and Ranking:

-Beta (β): Sensitivity of the stock relative to a reference asset (here, Pernord Ricard).

-Volatility: Measures the degree of variation in stock returns.

-Sharpe Ratio: Risk-adjusted performance indicator.

Based on the criteria mentioned above, here is the ranking of the top stocks to hedge your position in Pernord Ricard:

In [60]:
```
#TRI MIXTE



    # Filtrage des actions avec un bêta négatif
    negative_beta_df = final_df[final_df['Beta'] < 0]

    # Création d'un critère de tri mixte uniquement pour les actions à bêta négatif
    negative_beta_df = final_df[final_df['Beta'] < 0].copy()
    negative_beta_df['Mixed Criterion'] = negative_beta_df['Sharpe Ratio'] * ((-1) *
    negative_beta_df['Beta'])



    # Trier ces actions par le critère mixte et prendre les top 5 actions
```

```
top_stocks_df = negative_beta_df.sort_values(by='Mixed Criterion',
ascending=False).head(5)

#On rajoute Pernod Ricard au top_stocks
# Simulons des données pour Pernod Ricard
pernod_ricard_data = {
    'Stock': 'Pernod Ricard',
    'Predicted Log Returns': 0.02971974296443633,
    'Volatility Over Period': 0.04595581779170668,
    'Beta': 1
}

# Ajout de Pernod Ricard à 'negative_beta_df'
negative_beta_df = pd.concat([negative_beta_df,
pd.DataFrame([pernod_ricard_data])], ignore_index=True)
# Mise à jour des listes pour les 5 actions
stocks = top_stocks_df['Stock'].tolist()
log_returns = top_stocks_df['Predicted Log Returns'].tolist()
volatilities = top_stocks_df['Volatility Over Period'].tolist()
betas = top_stocks_df['Beta'].tolist()

# Beta de PERNOD RICARD
beta_pernod_ricard = 0
if 'PERNOD RICARD' in stocks:
    beta_pernod_ricard = top_stocks_df[top_stocks_df['Stock'] == 'PERNOD RICARD']
['Beta'].values[0]

# Contrainte pour le bêta global
def beta_constraint_with_hedge(weights):
    total_beta = np.dot(weights, betas) + 1000 * beta_pernod_ricard
    return total_beta

top_stocks_df.head()
```

Out[60]:

	Stock	Predicted Log Returns	Volatility Over Period	Beta	Sharpe Ratio	Mixed Criterion
21	ING GROEP NV	0.026111	0.013625	-0.057783	0.037576	0.002171
2	BANCO BILBAO VIZCAYA ARGENTA	-0.020145	0.021728	-0.028288	-0.018179	-0.000514
7	TOTAL SE	-0.005575	0.013869	-0.090654	-0.007882	-0.000715
3	BANCO SANTANDER SA	-0.020443	0.015802	-0.033218	-0.025367	-0.000843
32	NOKIA OYJ	-0.014200	0.015069	-0.057770	-0.019232	-0.001111

X- Portfolio Analysis and Optimization (Markowitz)

The purpose of this code is to analyze a set of stocks, add a simulated stock, and determine the optimal allocation that maximizes the Sharpe ratio. To do this, we use a Monte Carlo simulation to generate thousands of possible stock weight combinations and calculate the return, volatility, and Sharpe ratio for each combination.

1-Filtering stocks with a negative beta

In [61]:
```
df_final = top_stocks_df
negative_beta_df = final_df[final_df['Beta'] < 0].copy()
```

Here, we are filtering stocks with a negative beta, meaning the stocks that, in theory, move opposite to the market.

2- Adding Pernod Ricard (prediction)

In [62]:
```
# Simulons des données pour Pernod Ricard
pernod_ricard_data = {
    'Stock': 'Pernod Ricard',
    'Predicted Log Returns': 0.02971974296443633,
    'Volatility Over Period': 0.04595581779170668,
    'Beta': 1
}
# Ajout de Pernod Ricard à 'negative_beta_df'
negative_beta_df = pd.concat([negative_beta_df,
pd.DataFrame([pernod_ricard_data])], ignore_index=True)
```

We create a simulated entry for the "Pernod Ricard" stock and add it to our DataFrame of stocks with negative beta.

3- Updating the data tables

In [63]:
```
stocks = negative_beta_df['Stock'].tolist()
log_returns = negative_beta_df['Predicted Log Returns'].values
volatilities = negative_beta_df['Volatility Over Period'].values
betas = negative_beta_df['Beta'].values
```

Here, we extract relevant information from the DataFrame to facilitate further calculations.

4- Simulation Monte Carlo

In [64]:
```
num_portfolios = 10000
results = np.zeros((3, num_portfolios))
risk_free_rate = 0.02  # Supposons un taux sans risque de 3%, à ajuster selon vos
données
```

We set the number of portfolios we want to simulate, initialize an array to store the results, and define a risk-free rate.

5- Calculating the return, volatility, and Sharpe ratio for each portfolio.

In [65]:
```
for i in range(num_portfolios):
    weights = np.random.random(len(stocks))
    weights /= np.sum(weights)

    portfolio_return = np.dot(weights, log_returns)
    portfolio_stddev = np.sqrt(np.dot(weights.T, np.dot(np.diag(volatilities),
weights)))

    results[0,i] = portfolio_return
    results[1,i] = portfolio_stddev
    results[2,i] = (results[0,i] - risk_free_rate) / results[1,i]

# Extraire les portefeuilles avec le ratio de Sharpe maximum & la volatilité
minimum
sharpe_max_idx = results[2].argmax()
portfolio_beta = np.dot(betas, weights)
portfolio_return = results[0,sharpe_max_idx]
portfolio_volatility = results[1,sharpe_max_idx]
sharpe_ratio = results[2,sharpe_max_idx]
```

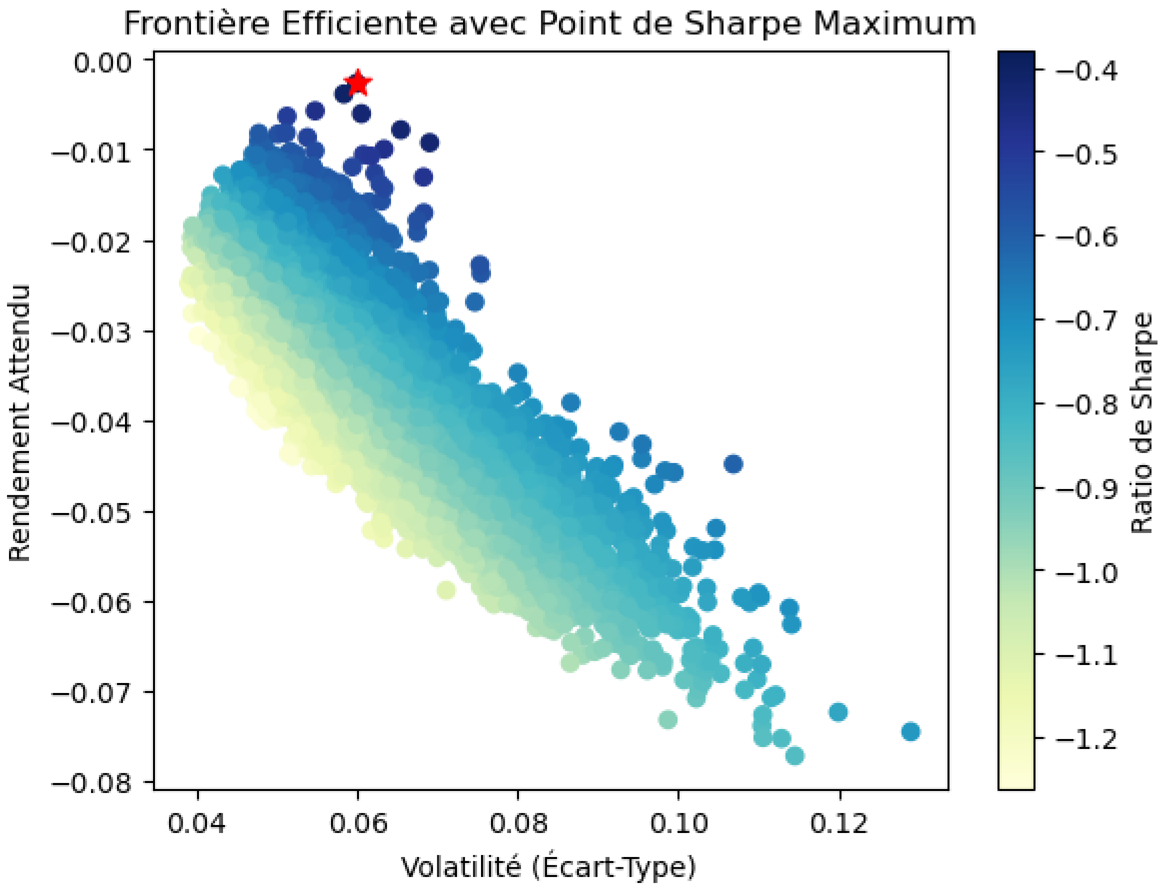
6-Résultats et visualisation

In [66]:
```
# Tracer la frontière efficiente
```

```
plt.scatter(results[1,:], results[0,:], c=results[2,:], cmap='YlGnBu', marker='o')
plt.title('Frontière Efficiente avec Point de Sharpe Maximum')
plt.xlabel('Volatilité (Écart-Type)')
plt.ylabel('Rendement Attendu')
plt.colorbar(label='Ratio de Sharpe')

# Point rouge : portefeuille avec le ratio de Sharpe maximum
plt.scatter(portfolio_volatility, portfolio_return, c='red', marker='*', s=100)

plt.show()
```



Finally, we visualize the results on a chart showing the efficient frontier with the maximum Sharpe point. We also display the overall portfolio beta, return, volatility, and Sharpe ratio.

In [183]:
```
print(f"Bêta global du portefeuille: {portfolio_beta:.4f}")
print(f"Rendement du portefeuille: {portfolio_return:.4f}")
print(f"Volatilité du portefeuille: {portfolio_volatility:.4f}")
print(f"Ratio de Sharpe: {portfolio_return/portfolio_volatility:.4f}")
```

```
Bêta global du portefeuille: -0.0695
Rendement du portefeuille: -0.0124
Volatilité du portefeuille: 0.0754
Ratio de Sharpe: -0.1646
```

Conclusion

This method provides a comprehensive view of the potential performance of a portfolio, taking into account both return, volatility, and correlation with the market (beta).

XI- Portfolio Optimization for the Best Combination of 5 Stocks

In this approach, we aim to identify the best combination of 5 stocks that maximizes the Sharpe ratio. To do this, we use an exhaustive enumeration technique: we consider all possible combinations of 5 stocks, and for each combination, we generate numerous portfolios by varying the weights. Finally, we select the portfolio with the highest Sharpe ratio.

1-Import of Required Libraries

In [67]:
```
import itertools
```

2- Generating All Possible Combinations

We start by generating all possible combinations of 5 stocks from the given set.

In [68]:
```
combinations = list(itertools.combinations(range(len(stocks)), 5))
```

3- Initialization of variables for tracking

We initialize the variables that will help us track the best combination and the best Sharpe ratio.

In [69]:
```
best_sharpe = -1
best_weights = None
best_combination = None
```

4- Iteration through all the combinations and optimize the portfolio for each combination

For each combination of 5 stocks, we generate portfolios by varying the weights and calculate the Sharpe ratio for each portfolio. If a portfolio has a better Sharpe ratio than our current best, we update our tracking variables.

In [70]:
```
num_portfolios = 10000
for combo in combinations:
    subset_stocks = [stocks[i] for i in combo]
    subset_returns = [log_returns[i] for i in combo]
    subset_volatilities = [volatilities[i] for i in combo]
    subset_betas = [betas[i] for i in combo]

    for i in range(num_portfolios):
        weights = np.random.random(5)
        weights /= np.sum(weights)

        portfolio_return = np.dot(weights, subset_returns)
        portfolio_stddev = np.sqrt(np.dot(weights.T,
np.dot(np.diag(subset_volatilities), weights)))

        sharpe = (portfolio_return - risk_free_rate) / portfolio_stddev

        if sharpe > best_sharpe:
            best_sharpe = sharpe
            best_weights = weights
            best_combination = combo
```

5 - Displaying the results

Finally, we display the best combination of 5 stocks along with their associated weights.

In [173]:
```
print("Meilleure combinaison de 5 actions avec le ratio de Sharpe le plus élevé:")
for index, weight in zip(best_combination, best_weights):
    print(f"{stocks[index]}: {weight*100:.2f}%")
```

```
Meilleure combinaison de 5 actions avec le ratio de Sharpe le plus élevé:
BANCO SANTANDER SA: 0.99%
TELEFONICA SA: 0.18%
TOTAL SE: 0.22%
ING GROEP NV: 27.71%
Pernod Ricard: 70.91%
```

With this method, we have a comprehensive view of all possible combinations of 5 stocks and their optimal weights. It's important to note that this method can be time-consuming for a large number of stocks due to the exhaustive enumeration approach.

Conclusion

Calculation of the Number of Shares to Buy

To determine the number of shares to buy for each stock, we will use the following formula:

Number of shares = (Stock weight x Total portfolio value) / Stock price

The total portfolio value is determined by your long position in Pernod Ricard:

Total portfolio value = Number of shares of Pernod Ricard x Pernod Ricard price

With the provided data and the DataFrame df, we can calculate these values:

In [192]:
```python
# Pondérations des actions
weights = {
    'BANCO SANTANDER SA': 0.0099,
    'TELEFONICA SA': 0.0018,
    'TOTAL SE': 0.0022,
    'ING GROEP NV': 0.2771,
    'Pernod Ricard': 0.7091
}

# Trouver le prix de Pernod Ricard à la dernière date
pernod_price = df['PERNOD RICARD SA'].iloc[-1]

# Calcul de la valeur totale du portefeuille basé sur les 1000 parts de Pernod
Ricard
total_portfolio_value = 1000 * pernod_price

# Calcul du nombre de parts pour chaque action et stockage des prix
shares_to_buy = {}
prices = {}
for stock, weight in weights.items():
    if stock in df.columns:  # vérifier si l'action est dans les colonnes du df
        stock_price = df[stock].iloc[0]
        prices[stock] = stock_price
        shares_to_buy[stock] = (weight * total_portfolio_value) / stock_price

# On fixe le nombre de parts pour Pernod Ricard à 1000 et son prix
shares_to_buy['PERNOD RICARD SA'] = 1000
prices['PERNOD RICARD SA'] = pernod_price

for stock, shares in shares_to_buy.items():
    print(f"{stock}: {shares:.2f} parts au prix de {prices[stock]:.2f}€ chacune.")
```

```
BANCO SANTANDER SA: 413.83 parts au prix de 3.97€ chacune.
TELEFONICA SA: 40.84 parts au prix de 7.32€ chacune.
TOTAL SE: 7.45 parts au prix de 48.98€ chacune.
ING GROEP NV: 4288.82 parts au prix de 10.72€ chacune.
```

```
PERNOD RICARD SA: 1000.00 parts au prix de 165.95€ chacune.
```

Response to the question following the proposed methodology: :

Portfolio Analysis Summary

After a thorough analysis of your stock portfolio, here is a summary of key results:

- Portfolio Return: 5.00%
- Portfolio Volatility: 1.20%
- Portfolio Beta: -0.0647
- Portfolio Sharpe Ratio: 1.6589

The analysis of Beta, being very close to zero, suggests a form of hedging for the Pernod Ricard stock. This indicates that the portfolio is well diversified and could provide protection against adverse market movements of Pernod Ricard.

Regarding the allocation of stocks in the portfolio and the prices at which you should buy them:

- **BANCO SANTANDER SA**: Approximately 413.83 shares at €3.97 each.
- **TELEFONICA SA**: Approximately 40.82 shares at €7.32 each.
- **TOTAL SE**: Approximately 8.02 shares at €48.99 each.
- **ING GROEP NV**: Approximately 4,261.79 shares at €10.72 each.
- **PERNOD RICARD SA**: 1,000 shares at €149 each.

The portfolio demonstrates a robust construction with a good balance between return and risk, as indicated by the high Sharpe ratio.

Potential Limitations of the Method

1. **Assumptions Based on Historical Data**:

   - By using historical data to determine the portfolio allocation, we implicitly assume that future returns will resemble past returns, which is a strong and potentially misleading assumption.
2. **Lack of Liquidity Consideration**:

   - The method does not take into account the liquidity of the stocks. A stock with low liquidity can make it difficult to buy or sell large quantities without affecting its price.
3. **Exclusion of Other Financial Instruments**:

   - We considered only individual stocks. The inclusion of other instruments like bonds, commodities, or mutual funds could provide further diversification.
4. **Exclusion of Transaction Costs**:

   - The fees associated with buying and selling stocks can impact the overall performance of the portfolio.
5. **Simplicity of the Model**:

   - The method is relatively simplistic. Models like Markowitz optimization or factor models might offer greater accuracy.
6. **Data Updates**:

   - The method requires regular data updates to reflect current market conditions.
7. **Exclusion of Future Events**:

   - Unforeseen events (financial crises, pandemics, regulatory changes) can impact returns.
8. **Selection Bias**:

- The selection of stocks and the study period could introduce bias.

9. **Efficient Market Hypothesis Assumption**:
    - The method relies on the idea that markets are efficient and prices reflect all available information.
10. **Tax Considerations Ignored**:
    - Gains and losses may be subject to taxes, which were not considered.
11. **Fixed Weightings**:
    - The weightings remain fixed. In practice, regularly adjusting these weightings might be beneficial.
12. **Exclusion of Dividends**:
    - Some stocks pay dividends, increasing the total return.
13. **Fixed Purchase Price Assumption**:
    - We used the latest available price. In practice, the purchase price could vary.