In [1]:
```
#!pip install yfinance==0.2.54
```

0. Introduction

Welcome to the Risk-Parity Portfolio Analysis project! In this project, you'll explore the concept of risk-parity, a popular portfolio management strategy that aims to allocate investments in a way that equalizes the risk contribution of each asset. This approach is widely used by hedge funds and institutional investors to create more balanced portfolios that are less sensitive to market volatility.

Throughout this project, you'll learn how to download financial data for various assets, calculate returns, and compute risk-parity weights using Python. By the end, you'll evaluate the performance of your risk-parity portfolio through key financial metrics such as annualized return, volatility, and Sharpe ratio. This hands-on experience will deepen your understanding of portfolio management and give you practical skills in financial data analysis. Whether you're new to finance or looking to enhance your quantitative finance skills, this project provides a solid foundation in risk-parity strategies and their application in real-world scenarios.

Getting Started: Tips and Instructions

1. **Familiarize Yourself with the Notebook Structure:**
    - The notebook is organized into clearly defined sections, each focusing on a specific aspect of the risk-parity portfolio analysis. Take a moment to glance through the sections to understand the overall workflow.
2. **Review the Data:**
    - Since the data is already available, start by exploring the initial few cells to understand the data structure and what each column represents. This will give you context for the calculations and analysis you'll be performing.
3. **Run Cells Sequentially:**
    - Work through the notebook by executing each code cell in order. This will help you build your analysis step by step and ensure that each part of the project is functioning as expected.
4. **Experiment with Parameters:**
    - Feel free to tweak parameters such as the rolling window size for calculating risk-parity weights or adjust the assets included in the portfolio. Experimenting will help deepen your understanding of how these factors influence portfolio performance.

5. **Leverage the Plots:**
    - Use the generated plots to visually assess the impact of your calculations. The visualizations are crucial for understanding the risk-parity approach and how different assets contribute to the portfolio.
6. **Reflect:**
    - As you work through the notebook, consider why each step is necessary and how it contributes to the overall goal of building a risk-parity portfolio. If something isn't clear, take a moment to reflect or explore additional resources.

Enjoy the process of exploring risk-parity strategies, and don't hesitate to dive deeper into the data or code to enhance your learning experience!

1. Import Libraries

In [2]:
```
# Load necessary libraries
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
```

2. Download Financial Data

We will download front-month futures data for S&P500, 10-year Treasuries, gold, and US dollar using the `yfinance` library.

In [3]:
```
# Download front-month futures data
symbols = ['ES=F', 'ZN=F', 'GC=F', 'DX=F', 'BZ=f']
data = yf.download(symbols, multi_level_index=False, auto_adjust=True)
```

```
[**********100%**********]  5 of 5 completed
```

3. Resample Data

In this section, we aim to reduce the noise in the daily financial data by resampling it to a monthly frequency. Resampling is a common technique in time series analysis, allowing us to aggregate data points over a specified time period. This helps in smoothing out short-term fluctuations and making the data more manageable for analysis.

In [4]:
```
# Resample data to monthly frequency
data = data.resample("M").last() # YOUR CODE HERE
# Convert index to datetime
data.index = pd.to_datetime(data.index) # YOUR CODE HERE
```

```
/var/folders/gx/x4w_yhkx4xddw8kdmqf3m5zw0000gn/T/ipykernel_27318/271401034
8.py:2: FutureWarning: 'M' is deprecated and will be removed in a future ver
sion, please use 'ME' instead.
  data = data.resample("M").last() # YOUR CODE HERE
```

In [5]:
```
print(type(data.index))
data.tail()
```

```
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>
```

Out[5]:

Price					Close		
Ticker	BZ=F	DX=F	ES=F	GC=F	ZN=F	BZ=F	
Date							
2024-11-30	72.940002	105.828003	6051.50	2657.000000	111.015625	73.500000	
2024-12-31	74.639999	108.295998	5935.75	2629.199951	108.750000	74.879997	
2025-01-31	76.760002	108.217003	6067.25	2812.500000	108.843750	77.080002	1
2025-02-28	73.180000	107.557999	5963.25	2836.800049	111.062500	73.739998	
2025-03-31	70.620003	103.434998	5604.00	2994.600098	110.671875	71.790001	1

5 rows × 25 columns

4. Clean and Prepare Data

In this step, we will focus on extracting the relevant data, handling missing values, and ensuring the data is ready for analysis. Specifically, we'll subset the adjusted close prices from our dataset, fill any missing values, and drop rows with unknown prices.

Steps to Clean and Prepare Data

1. **Subset Adjusted Close Prices**
2. **Fill Missing Values (NaNs)**
3. **Drop Rows with Remaining NaNs**
4. **Ensure Correct Date Formatting**

By following these steps, we will have a clean dataset of adjusted close prices that is free of missing values and properly formatted for time series analysis.

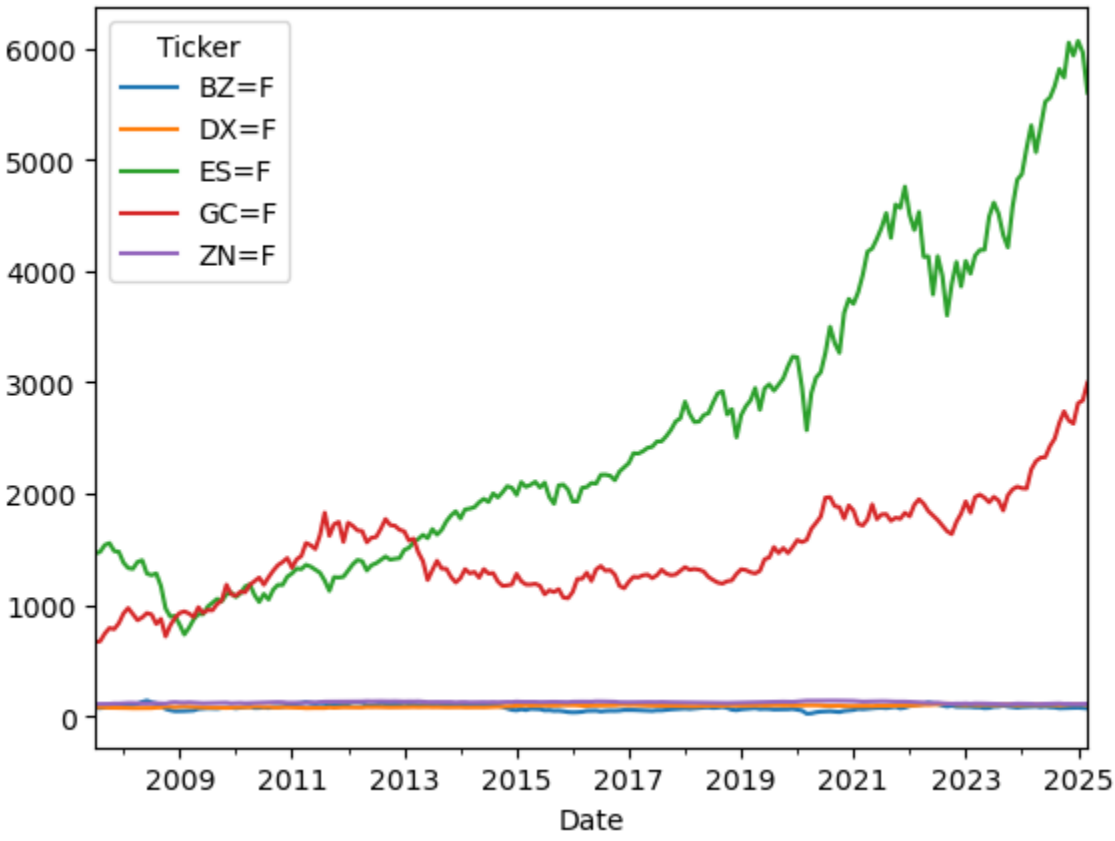
In [6]:
```
# Subset adjusted close prices and fill NaNs
prices = data['Close'].ffill().dropna() # YOUR CODE HERE
# Convert index to datetime
prices.index = pd.to_datetime(prices.index) # YOUR CODE HERE
```

In [7]:
```
print(prices)
```

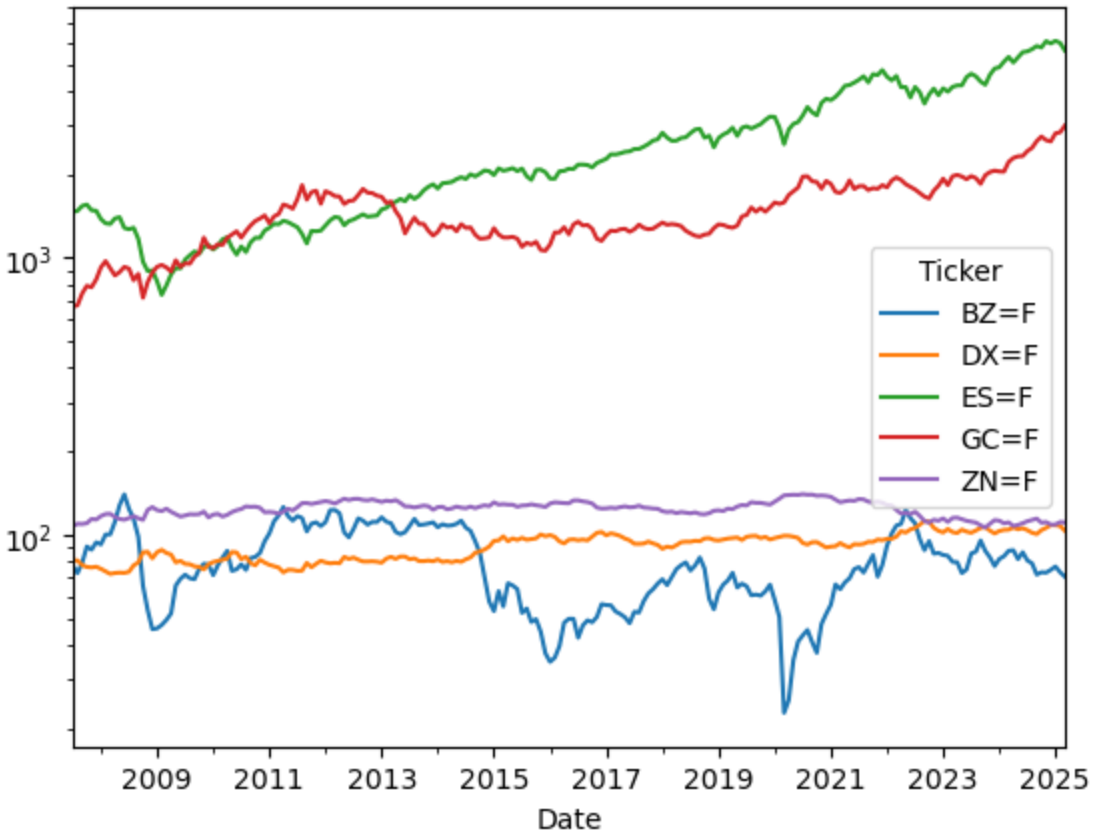
```
Ticker           BZ=F        DX=F     ES=F         GC=F        ZN=F
Date
2007-07-31  77.050003   80.660004  1462.00   666.900024  107.421875
2007-08-31  72.690002   80.745003  1476.75   673.000000  109.484375
2007-09-30  79.169998   77.625000  1538.00   742.799988  109.281250
2007-10-31  90.629997   76.459999  1555.00   792.000000  110.015625
2007-11-30  88.260002   76.169998  1483.75   782.200012  113.921875
...               ...         ...      ...         ...         ...
2024-11-30  72.940002  105.828003  6051.50  2657.000000  111.015625
2024-12-31  74.639999  108.295998  5935.75  2629.199951  108.750000
2025-01-31  76.760002  108.217003  6067.25  2812.500000  108.843750
2025-02-28  73.180000  107.557999  5963.25  2836.800049  111.062500
2025-03-31  70.620003  103.434998  5604.00  2994.600098  110.671875

[213 rows x 5 columns]
```

In [8]:
```
prices.plot()
plt.show()
```



In [9]:
```
prices.plot()
plt.yscale("log")
plt.show()
```



5. Calculate Returns

In financial analysis, one of the key metrics is the return on an investment, which shows how much the price of an asset has increased or decreased over a certain period. We will calculate the **arithmetic returns** of the adjusted close prices to measure this change over time.

Steps to Calculate Arithmetic Returns

1. **Understanding Arithmetic Returns:**

   - Arithmetic returns measure the percentage change in the price of an asset from one period to the next.
   - The formula for calculating the arithmetic return for a given period is:

     $$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

     where:

     - R_t is the return at time t,
     - P_t is the price at time t,
     - P_{t-1} is the price at the previous time period.
   - This formula can be simplified to: $R_t = \frac{P_t}{P_{t-1}} - 1$

- However, in practice, it is common to use the percentage change function provided by Pandas, which handles this calculation efficiently.

2. **Calculate Percentage Change**

3. **Handle Missing Values:**

By following these steps, we will have a series of arithmetic returns that represent the day-to-day percentage changes in the asset's price, which can be used for further analysis such as calculating cumulative returns, volatility, or risk metrics.

In [10]:
```
# Compute arithmetic returns
log_returns = np.log(prices).diff() # YOUR CODE HERE
```

In [11]:
```
print(log_returns)
```

```
Ticker          BZ=F      DX=F      ES=F      GC=F      ZN=F
Date
2007-07-31       NaN       NaN       NaN       NaN       NaN
2007-08-31 -0.058251  0.001053  0.010038  0.009105  0.019018
2007-09-30  0.085394 -0.039407  0.040639  0.098681 -0.001857
2007-10-31  0.135188 -0.015122  0.010993  0.064135  0.006698
2007-11-30 -0.026498 -0.003800 -0.046903 -0.012451  0.034891
...              ...       ...       ...       ...       ...
2024-11-30 -0.003012  0.018598  0.053108 -0.030140  0.004938
2024-12-31  0.023039  0.023053 -0.019313 -0.010518 -0.020619
2025-01-31  0.028007 -0.000730  0.021912  0.067394  0.000862
2025-02-28 -0.047762 -0.006108 -0.017290  0.008603  0.020180
2025-03-31 -0.035609 -0.039087 -0.062135  0.054134 -0.003523

[213 rows x 5 columns]
```

6. Compute Risk-Parity Weights

Risk-parity is an investment strategy that seeks to allocate portfolio weights in a way that each asset contributes equally to the overall portfolio risk. This method is especially useful in diversifying risk across different assets with varying levels of volatility.

Steps to Compute Risk-Parity Weights

1. **Understanding Risk-Parity:**

   - The idea behind risk-parity is to allocate more weight to less volatile assets and less weight to more volatile assets, thereby equalizing the risk contribution of each asset in the portfolio.
   - The weight of each asset in the portfolio is inversely proportional to its volatility.

2. **Calculate Rolling Volatility:**

   - Volatility is a statistical measure of the dispersion of returns for a given security or market index.

3. **Compute Inverse Volatility:**

4. **Normalize Weights:**

   - Once we have the inverse volatilities, we normalize them so that the sum of the weights for each time period equals 1. This ensures that we have a proper weight distribution across assets.
   - We achieve this by dividing the inverse volatility of each asset by the sum of the inverse volatilities across all assets for each time period.

5. **Shift Weights for Practical Application:**

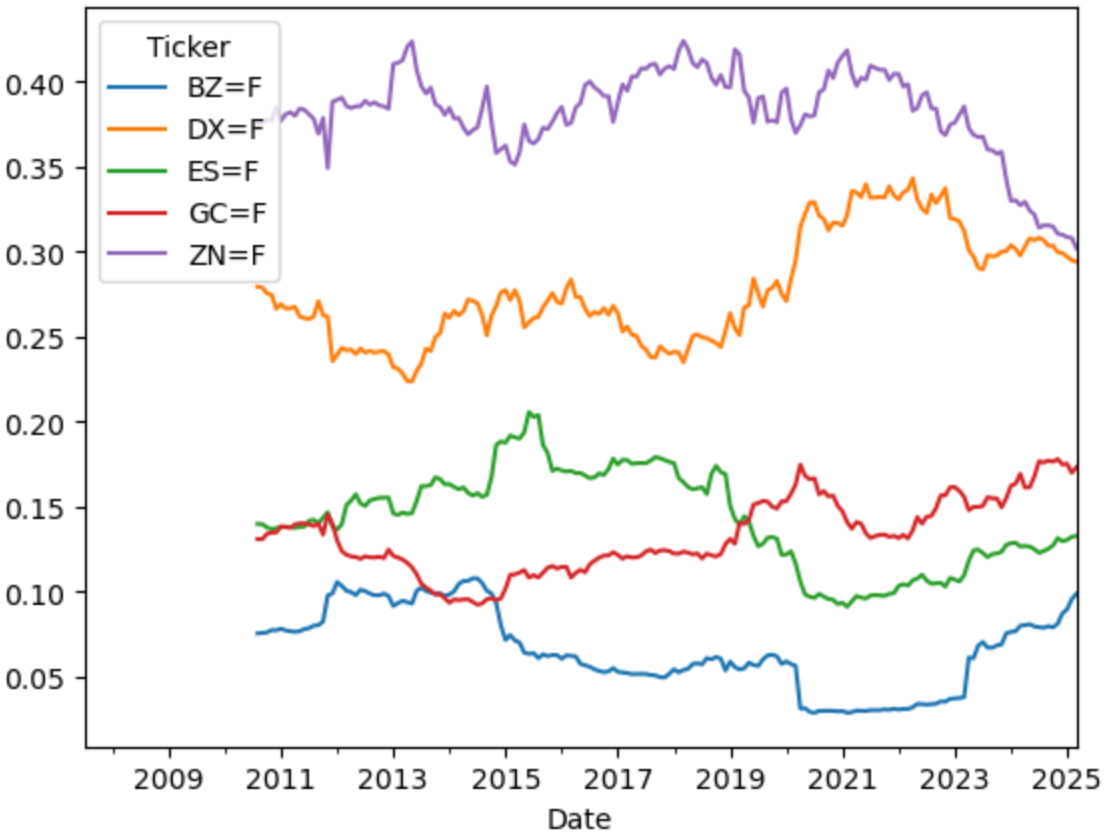
   - In a real-world scenario, we can only make investment decisions based on information that is available at the time of making the decision. Therefore, we shift the computed weights by one period to ensure that the weights at time t are based on data up to $t-1$. - This shift ensures that the weights applied are based on historical data up to the previous period, which is crucial for avoiding look-ahead bias.

By following these steps, we generate a time series of risk-parity weights that can be used to allocate assets in a way that balances the risk across the portfolio. The resulting weights adapt dynamically based on the rolling volatility, ensuring that each asset contributes equally to the overall risk over time.

In [12]:
```
def compute_risk_parity_weights(returns, window_size=36):
    # Compute volatility known at time t
    rolling_vol = returns.rolling(window_size).std()
    rolling_inverse_vol = 1/rolling_vol
    # Divide inverse volatility by the sum of inverse volatilities
    risk_parity_weights = rolling_inverse_vol.apply(lambda column: column /
    # Shift weights by one period to use only information available at time
    risk_parity_weights = risk_parity_weights.shift(1)
    return risk_parity_weights

risk_parity_weights = compute_risk_parity_weights(log_returns, 36)
```

In [13]:
```
risk_parity_weights.plot()
plt.show()
```



7. Calculate Weighted Returns

Once we have determined the risk-parity weights, the next step is to calculate the weighted returns for each asset and then aggregate them to obtain the portfolio returns. This process allows us to understand how the risk-parity allocation strategy would have performed over time.

Steps to Calculate Weighted Returns

1. **Understanding Weighted Returns:**
    - The weighted return of an asset is calculated by multiplying its individual return by the corresponding weight in the portfolio.
    - The total portfolio return at any given time is the sum of the weighted returns of all assets.
2. **Apply Weights to Returns:**
    - We will multiply the returns of each asset by its corresponding risk-parity weight for each time period. This will give us the weighted return for each asset.
3. **Handle Missing Data:**
    - We drop any rows with missing values (`NaN`) that may result from the multiplication process. This ensures that we only consider time periods with complete data.

4. **Aggregate Weighted Returns:**

    - To calculate the portfolio's return at each time period, sum the weighted returns across all assets. This gives the overall return of the portfolio for each time period.

Mathematical Representation:

Given the returns $r_{i,t}$ for asset i at time t, and the risk-parity weight $w_{i,t}$ for asset i at time t, the weighted return for asset i at time t is:

$$r_{i,t}^{weighted} = r_{i,t} \times w_{i,t}$$

The total portfolio return at time t is the sum of the weighted returns for all assets:

$$R_{portfolio,t} = \sum_{i=1}^{n} r_{i,t}^{weighted}$$

Where n is the number of assets in the portfolio.

By following these steps, we obtain the portfolio returns that reflect the performance of the risk-parity strategy over time. This provides insights into how well the strategy balanced risk across different market conditions.

In [14]:
```
# Calculate weighted returns
weighted_returns = (log_returns * risk_parity_weights).sum(axis=1) # YOUR CO
risk_parity_portfolio_returns = sum(weighted_returns) # YOUR CODE HERE
```

In [15]:
```
print(risk_parity_portfolio_returns)
```

```
0.33276780090802804
```

8. Evaluate Portfolio Performance

To assess the performance of the risk-parity portfolio, we will compute several key financial metrics. These metrics will help us understand the portfolio's return, risk, and overall performance characteristics. Below are the details of the metrics we will calculate, along with the functions used and their corresponding arguments.

1. **Annualized Mean Return**
    - **Description:** The average return of the portfolio on an annual basis. This is useful for understanding the long-term growth rate of the portfolio.
2. **Annualized Volatility**
    - **Description:** A measure of the portfolio's return volatility on an annual basis, indicating the degree of variation in returns.
    - **Formula:**
      $\text{Annualized Volatility} = \text{Standard Deviation of Monthly Returns} \times \sqrt{12}$
3. **Skewness**

- **Description:** Skewness measures the asymmetry of the return distribution. Positive skewness indicates a distribution with a longer right tail, while negative skewness indicates a longer left tail.

4. **Kurtosis**
   - **Description:** Kurtosis measures the "tailedness" of the return distribution. High kurtosis indicates a distribution with heavy tails and a sharp peak.
5. **Maximum Drawdown**
   - **Description:** The maximum observed loss from a peak to a trough of the portfolio's cumulative returns, before a new peak is attained.
   - **Formula:** $\text{Drawdown} = \frac{\text{Cumulative Return} - \text{Running Maximum}}{\text{Running Maximum}}$
6. **Sharpe Ratio**
   - **Description:** The Sharpe Ratio measures the risk-adjusted return of the portfolio, calculated as the ratio of the portfolio's excess return (over the risk-free rate, typically assumed to be 0 in this case) to its volatility.
   - **Formula:** $\text{Sharpe Ratio} = \frac{\text{Annualized Mean Return}}{\text{Annualized Volatility}}$
7. **Sortino Ratio**
   - **Description:** The Sortino Ratio is a variation of the Sharpe Ratio that penalizes only downside volatility, thus providing a better measure of risk-adjusted return for portfolios that have asymmetric return distributions.
   - **Formula:** $\text{Sortino Ratio} = \frac{\text{Annualized Mean Return}}{\text{Downside Volatility}}$
8. **Calmar Ratio**
   - **Description:** The Calmar Ratio measures the risk-adjusted return of a portfolio by comparing the annualized return to the maximum drawdown, providing insight into performance relative to the worst-case scenario.
   - **Formula:** $\text{Calmar Ratio} = \frac{\text{Annualized Mean Return}}{-\text{Maximum Drawdown}}$

After calculating these metrics, we will display the results to evaluate the performance of the risk-parity portfolio comprehensively.

In [16]:
```
# Evaluate portfolio performance
annual_mean_return = weighted_returns.mean() * 12
annual_volatility = weighted_returns.std() * np.sqrt(12)
skewness = log_returns.skew()
kurtosis = log_returns.kurtosis()

# Compute drawdown
cumulative_returns = np.exp(weighted_returns.cumsum())
running_max = cumulative_returns.cummax()
drawdown = (running_max - cumulative_returns) / running_max
max_drawdown = np.max(drawdown)

# Compute Sharpe ratio
sharpe_ratio = annual_mean_return / annual_volatility

# Compute Sortino ratio
downside_std = weighted_returns[weighted_returns<0].std() * np.sqrt(12)
```

```
sortino_ratio = annual_mean_return/downside_std

# Compute Calmar ratio
calmar_ratio = annual_mean_return / max_drawdown

# Display results
print(f"Mean Annual Return: {annual_mean_return:.4f}")
print(f"Annual Volatility: {annual_volatility:.4f}")
print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurtosis}")
print(f"Maximum Drawdown: {max_drawdown:.4f}")
print(f"Sharpe Ratio: {sharpe_ratio:.4f}")
print(f"Sortino Ratio: {sortino_ratio:.4f}")
print(f"Calmar Ratio: {calmar_ratio:.4f}")
```

```
Mean Annual Return: 0.0187
Annual Volatility: 0.0349
Skewness: Ticker
BZ=F   -2.167212
DX=F    0.190530
ES=F   -0.758093
GC=F   -0.337689
ZN=F    0.342506
dtype: float64
Kurtosis: Ticker
BZ=F    14.977169
DX=F     0.812640
ES=F     1.370629
GC=F     1.063560
ZN=F     2.001770
dtype: float64
Maximum Drawdown: 0.0596
Sharpe Ratio: 0.5376
Sortino Ratio: 0.7419
Calmar Ratio: 0.3147
```

9. Plot Results

Visualizing the performance of the risk-parity portfolio is crucial for understanding the dynamics of cumulative returns and drawdowns over time. We will create a plot that displays both the cumulative returns and the drawdowns on the same graph. This will allow us to see how the portfolio grows over time and the extent of losses from peak to trough.

Steps to Plot the Results:

1. **Initialize the Plot**:
    - We use `plt.subplots()` to create a figure and an axis object, allowing us to customize the plot.
    - The `figsize` argument specifies the size of the plot. In this case, we choose a wide format (14x7) to better display the time series data.
    - Example:

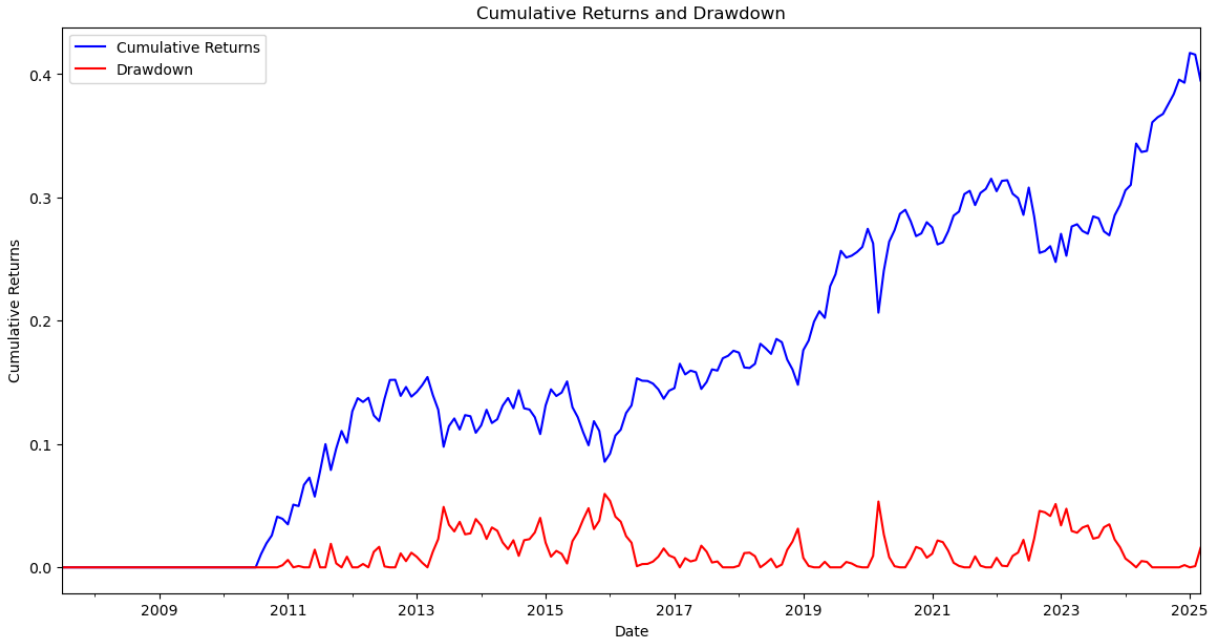
```
fig, ax = plt.subplots(figsize=(14, 7))
```

2. **Plot Cumulative Returns**:
   - The `plot()` function is used to plot the cumulative returns on the axis `ax`.
   - The `label` argument is used to create a legend entry, and `color` specifies the color of the line.
   - Example:
     ```
     cumulative_returns.plot(ax=ax, label='Cumulative Returns', color='blue')
     ```
3. **Plot Drawdown**:
   - Similarly, the `plot()` function is used to plot the drawdown on the same axis `ax`.
   - The `label` and `color` arguments distinguish this line from the cumulative returns.
   - Example:
     ```
     drawdown.plot(ax=ax, label='Drawdown', color='red')
     ```
4. **Customize the Plot**:
   - The `set_title()` function sets the title of the plot, making it clear what the graph represents.
   - The `set_ylabel()` and `set_xlabel()` functions label the y-axis and x-axis, respectively, to indicate what the axes represent (e.g., 'Cumulative Returns' and 'Date').
   - Example:
     ```
     ax.set_title('Cumulative Returns and Drawdown')
     ax.set_ylabel('Cumulative Returns')
     ax.set_xlabel('Date')
     ```
5. **Add a Legend**:
   - The `legend()` function adds a legend to the plot, helping to identify which line represents cumulative returns and which represents drawdown.
   - Example:
     ```
     ax.legend()
     ```
6. **Display the Plot**:
   - Finally, `plt.show()` is called to display the plot.
   - Example:
     ```
     plt.show()
     ```

The resulting plot will provide a clear visual representation of how the portfolio's value has evolved over time and the magnitude of any losses (drawdowns) experienced during the period. This visualization is essential for analyzing the risk and return profile of the portfolio.

In [17]:
```
# Plot portfolio performance
fig, ax = plt.subplots(figsize=(14, 7))
```

```
(cumulative_returns - 1).plot(ax=ax, label='Cumulative Returns', color='blue
drawdown.plot(ax=ax, label='Drawdown', color='red')
ax.set_title('Cumulative Returns and Drawdown')
ax.set_ylabel('Cumulative Returns')
ax.set_xlabel('Date')
ax.legend()
plt.show()
```



In [ ]: