User Guide

Mahak Sahay

22116051

Overview:

This algorithm identifies cointegrated pairs of stocks from a given universe, applies the Golden Cross trading strategy to generate trading signals, and backtests the performance of these signals. The algorithm also provides key performance metrics and visualizations.

Instructions:

1. **Parameters**:
   - **universe**: List of stock tickers to be considered.
   - **start**: Start date for fetching historical data.
   - **end**: End date for fetching historical data.
   - **fee**: Trading fee (proportion of trade value).
   - **window**: Lookback window for calculating cointegration.
   - **t_threshold**: Threshold for t-statistic in the trading signal generation.
   - **coint_threshold**: P-value threshold for identifying cointegrated pairs.
   - **spread_threshold**: Threshold for spread to generate buy/sell signals.

2. **Data Preprocessing**:
   - Fetch historical price data for the specified universe of stocks using `yfinance`.
   - Check and handle any missing data.

3. **Identifying Cointegrated Pairs**:

   - Use the `find_cointegrated_pairs` function to identify pairs of stocks that are cointegrated based on the specified p-value threshold.

   - The `find_cointegrated_pairs` function identifies pairs of stocks from a given DataFrame that are cointegrated based on the Engle-Granger cointegration test. It initializes matrices to store the cointegration scores and p-values for all possible pairs of stocks.

   - For each pair, it extracts their price series and performs the cointegration test, storing the results in the matrices. If the p-value of a pair is below the specified threshold, the pair is considered cointegrated and is added to the list of pairs. The function returns the matrices of scores and p-values, as well as the list of cointegrated pairs.

```python
# Function to identify cointegrated pairs
def find_cointegrated_pairs(data, coint_threshold):
    n = data.shape[1]
    score_matrix = np.zeros((n, n))
    pvalue_matrix = np.ones((n, n))
    keys = data.columns
    pairs = []

    for i in range(n):
        for j in range(i+1, n):
            S1 = data[keys[i]]
            S2 = data[keys[j]]
            result = coint(S1, S2)
            score = result[0]
            pvalue = result[1]
            score_matrix[i, j] = score
            pvalue_matrix[i, j] = pvalue
            if pvalue < coint_threshold:
                pairs.append((keys[i], keys[j]))

    return score_matrix, pvalue_matrix, pairs
```

4. **Signal Generation**:
   - Apply the Golden Cross strategy to generate buy/sell signals based on the 20-day and 50-day moving averages.
5. **Backtesting**:
   - Use the `backtest_pair` function to simulate trading based on the generated signals.
   - Backtesting report calculates daily returns for `stock1` by dividing the current price by the previous price and subtracting 1. The first value is set to 0.
   - **res**: Fits an Ordinary Least Squares (OLS) regression of `S2` on `S1` (including a constant term).
   - **beta**: Slope coefficient from the regression, representing the hedge ratio.
   - **alpha**: Intercept from the regression.
   - Signals are generated when the spread exceeds predefined thresholds, indicating buy or sell actions.
   - The function then computes gross and net returns, accounting for transaction fees, and records these along with the generated signals and spread values.

- This approach helps in assessing the strategy's performance by simulating trades over the historical period, enabling analysis of profitability and risk.

- Calculate performance metrics such as annualized return, annualized volatility, Sharpe ratio, and maximum drawdown.

6. Risk Analysis

- The calculate_max_drawdown function is used for risk analysis. Maximum drawdown is a key risk metric that quantifies the largest loss experienced from a peak to a trough in a given period.
- It helps in assessing the downside risk of an investment strategy by showing the most significant drop in portfolio value before a recovery to the previous peak.

```python
# Calculate maximum drawdown
def calculate_max_drawdown(returns):
    cumulative_returns = np.cumprod(1 + returns) - 1
    running_max = np.maximum.accumulate(cumulative_returns)

    # Check for cases where running_max is zero to avoid division by zero
    if np.any(running_max == 0):
        drawdowns = np.zeros_like(cumulative_returns)
    else:
        drawdowns = (running_max - cumulative_returns) / running_max

    max_drawdown = np.nanmax(drawdowns)  # Use np.nanmax to ignore NaN values

    return max_drawdown
```

- It first calculates the cumulative returns and then determines the running maximum of these cumulative returns. To avoid division by zero, it computes drawdowns only where the running maximum is non-zero, otherwise setting drawdowns to zero.
- The function then identifies the maximum drawdown using np.nanmax to ignore any NaN values, providing a measure of the greatest potential loss from a peak to a trough in the return series.

7. Visualizations:

- **Cumulative Returns**: Plot the cumulative returns of all cointegrated pairs using the `plot_cumulative_returns` function.
- **Frequency Distribution of Final Prices**: Use the `plot_price_distribution` function to plot the distribution of the final prices of all stocks in the universe.
- Generated signals for each cointegrated stocks.

8. Interpreting Results:

- The backtesting report provides key performance metrics for each pair.
- Cumulative returns plots show the growth of the investment over time.
- Frequency distribution plots show the distribution of final stock prices.

9. Nelder-Mead

- While working on the code we have also used Nelder-Mead for optimization to find the optimal value of the coefficient b in the context of your pairs trading strategy.
- Nelder-Mead is a derivative-free optimization algorithm. This makes it suitable for problems where the objective function is not differentiable or when calculating the gradient is difficult or computationally expensive.
- Nelder-Mead is used to find the value of b that minimizes the unit_root function, i.e., the value of b that makes the residuals as stationary as possible.

10. Adjusting Parameters:

- Modify the `universe`, `start`, and `end` parameters to analyze different stocks and time periods.
- Adjust the `fee`, `window`, `t_threshold`, `coint_threshold`, and `spread_threshold` parameters to optimize the strategy based on the desired risk/return profile.