Documentation for GoalFi MCDM Recommendation Model

Overview

This document provides detailed documentation for the code implementing multiple MCDM methods, including CRITIC, TOPSIS, and COPRAS, along with calculations for the factors within the dataset. The code aims to assist in mutual fund selection based on user-defined personal factors and risk levels.

Modules and Libraries Used

- numpy (Numerical computations)
- pandas (Data manipulation and analysis)
- yfinance (Fetching financial data)
- Mftool (Fetching mutual fund data)
- datetime and timedelta (Handling date operations)

Key Functions

1. CRITIC Weight Calculation

This function computes weights for criteria based on the CRITIC method, considering the standard deviation and conflict between criteria.

Function: calculate_critic_weights(df, criteria_cols)

Steps:

1. Normalize the decision matrix:

$$x_{ij} = \frac{x_{ij} - x_{\min,j}}{x_{\max,j} - x_{\min,j}}$$

2. Calculate standard deviation for each criterion:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

3. Compute the conflict matrix:

$$C_j = \sum_{k=1,k\neq j}^{m} (1 - r_{jk})$$

4. Calculate weights:

$$C_k = a_k \left[\sum_{i=1}^{m} (1 - r_{ik}) \right], \quad k = 1,2, \dots, m$$

$$W_k' = \frac{C_k}{\sum_{b=1}^{m} C_b}$$

Returns: CRITIC weights for each criterion.

---

2. TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution)

This function implements the TOPSIS method with CRITIC weights and allows user-defined ideal solutions.

Function: topsis(df, criteria_cols, benefit_cols, cost_cols, user_row_index)

Steps:

1. Normalize the decision matrix using the same formula as above.
2. Compute the weighted normalized decision matrix:

$$V_{ij} = W_j \cdot R_{ij}$$

3. Define the ideal and anti-ideal solutions:

$$A^+ = \{\max(x_{ij}) \mid j \in J^+\} \quad \text{(Positive Ideal Solution)}$$

$$A^- = \{\min(x_{ij}) \mid j \in J^-\} \quad \text{(Negative Ideal Solution)}$$

4. Calculate distances:

$$D_i^+ = \sqrt{\sum_{j=1}^{m} (V_{ij} - A_j^+)^2}$$

$$D_i^- = \sqrt{\sum_{j=1}^{m} (V_{ij} - A_j^-)^2}$$

5. Compute relative closeness:

$$R_i = \frac{D_i^-}{D_i^+ + D_i^-}$$

6. Rank alternatives based on the relative closeness.

Returns: Updated DataFrame with TOPSIS scores and ranks, and CRITIC weights.

3. COPRAS (Complex Proportional Assessment)

This function ranks alternatives using the COPRAS method.

Function: copras(df, benefit_criteria, weights)

Steps:

1. Normalize the decision matrix by shifting and dividing values:

$$R_{ij} = \frac{x_{ij} - \min(x_j)}{\sum_{i=1}^{m}(x_{ij} - \min(x_j))}$$

2. Compute positive and negative deviations:

$$D_i^+ = \max(0, A^+ - V_i)$$

$$D_i^- = \max(0, V_i - A^-)$$

3. Calculate the performance index:

$$P_i = \frac{Q_i}{Q_{\max}} \times 100$$

4. Rank alternatives based on Performance index

Returns: Ranked DataFrame with scheme codes, names, and performance indices.

---

4. CAGR Calculation

Calculates the Compound Annual Growth Rate.

Function: calculate_cagr(start_nav, end_nav, years)

Formula:

$$\text{CAGR} = \left(\frac{\text{Ending Value}}{\text{Beginning Value}}\right)^{\frac{1}{n}} - 1$$

---

5. Jensen's Alpha Calculation

Calculates Jensen's Alpha for a mutual fund.

Function: calculate_jensens_alpha(returns, benchmark_returns, beta, risk_free_rate)

Formula:

$$\alpha_J = R_i - \left(R_f + \beta_{iM} \cdot (R_M - R_f)\right)$$

6. User Personalization and Basket Generation

Matches user-defined factors to suitable mutual funds.

Function: get_basket(df_personal_factors, result)

Steps:

1. Divide ranked funds into 5 risk categories using pd.qcut.
2. Match the user's risk level to the corresponding basket.
3. Select the top 5 funds in the basket sorted by 3-Year CAGR.

Data Inputs

1. **Personal Factors Data**:
    - "Risk Level": User's risk tolerance (1 to 5).
    - "Monthly SIP Amount": SIP amount in INR.
    - "Age": Age of the user.
    - "Time Horizon": Investment duration in years.
    - "Debt Level as % of Income": Percentage of income allocated to debt.
2. **Mutual Fund Data**:
    - Columns: "Code", "Name", criteria metrics (e.g., "Sharpe", "Beta", "HHI").
3. **Benchmark Data**:
    - NIFTY 50 index historical data for calculating benchmark returns.

Outputs

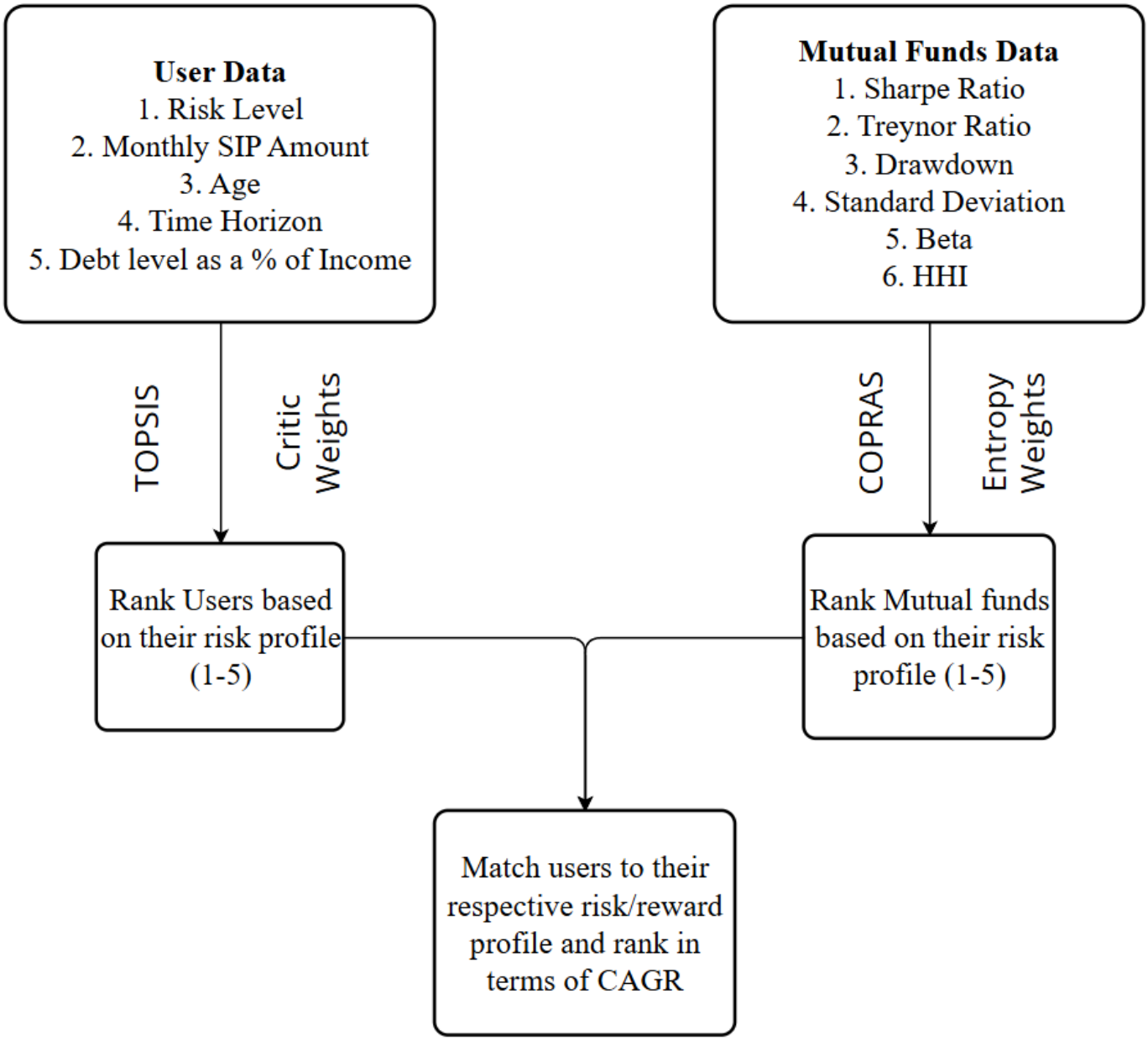
1. **CRITIC Weights**: Weights for each criterion.
2. **TOPSIS Results**: DataFrame with TOPSIS scores and ranks.
3. **COPRAS Rankings**: DataFrame with performance indices and risk categories.
4. **User Matched Basket**: Top 5 mutual funds based on user's risk profile.

Example Usage

```
# Define user inputs
criteria_cols = ["Monthly SIP Amount", "Age", "Time Horizon", "Debt Level as % of Income"]
benefit_cols = ["Monthly SIP Amount", "Time Horizon"]
cost_cols = ["Age", "Debt Level as % of Income"]

# Run TOPSIS
user_row_index = len(df_personal_factors) - 1
updated_df, critic_weights = topsis(df_personal_factors, criteria_cols, benefit_cols, cost_cols, user_row_index)

# Generate User Basket
user_basket = get_basket(df_personal_factors, result)
print(user_basket)
```


User Data
1. Risk Level
2. Monthly SIP Amount
3. Age
4. Time Horizon
5. Debt level as a % of Income
Mutual Funds Data
1. Sharpe Ratio
2. Treynor Ratio
3. Drawdown
4. Standard Deviation
5. Beta
6. HHI
TOPSIS
Critic Weights
COPRAS
Entropy Weights
Rank Users based on their risk profile (1-5)
Rank Mutual funds based on their risk profile (1-5)
Match users to their respective risk/reward profile and rank in terms of CAGR