**Portfolio-Benchmark Association Generator**

**Developer Documentation**

**Overview**

This Python module generates associations between portfolio codes and benchmarks for target date funds (TDFs). It creates a DataFrame that maps portfolios to their primary equity and secondary fixed income benchmarks, reflecting the dual nature of target date funds which contain both asset classes throughout their lifecycle.

**Table of Contents**

* Dependencies
* Setup and Configuration
* Data Flow
* Module Components
  + 4.1 Portfolio-Benchmark Association Generation
  + 4.2 Main Execution Function
* Key Design Decisions
* Data Structure
* Error Handling
* Testing Considerations
* Future Enhancements
* Developer Checklist

**1. Dependencies**

The module relies on the following external libraries:

* pandas: For data manipulation and DataFrame operations
* logging: For structured logging (rather than print statements)

Ensure these dependencies are installed:

pip install pandas

**2. Setup and Configuration**

**Logging Configuration**

The module uses Python's standard logging module configured at INFO level. All output uses logging rather than print statements for better production integration and flexibility.

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger(\_\_name\_\_)

**3. Data Flow**

The module follows a simple data flow:

* Define product code mapping (ticker to product code)
* Define portfolio-product relationships
* For each portfolio, create two benchmark associations:
  + Primary benchmark (GSPC - equity) with rank 1
  + Secondary benchmark (AGG - fixed income) with rank 2
* Return the associations as a pandas DataFrame

**4. Module Components**

**4.1 Portfolio-Benchmark Association Generation**

**Function:** generate\_portfolio\_benchmark\_association()  
**Purpose:** Creates associations between portfolios and their benchmarks.  
**Process:**

* Define the product mapping (ticker to product code)
* Define the portfolio data with portfolio codes and their associated product codes
* For each portfolio:
  + Add a primary equity benchmark (GSPC) with rank 1
  + Add a secondary fixed income benchmark (AGG) with rank 2
* Return the associations as a pandas DataFrame

**Data Sources:**

* Hard-coded product mapping dictionary
* Hard-coded portfolio data list

**Output Structure:**

* PORTFOLIOCODE: Portfolio identifier (e.g., PORT001)
* BENCHMARKCODE: Benchmark code (GSPC or AGG)
* RECIPIENTCODE: Set to "NULL"
* RANK: Benchmark priority (1 = primary, 2 = secondary)

**4.2 Main Execution Function**

**Function:** main()  
**Purpose:** Orchestrates the execution of the portfolio-benchmark association generation process.  
**Process:**

* Call the generate\_portfolio\_benchmark\_association() function
* Log completion message with record count
* Return the generated DataFrame

**5. Key Design Decisions**

**Dual Benchmark Approach**

Target date funds contain both equity and fixed income components throughout their lifecycle, with allocations shifting according to the glide path. Therefore, each portfolio is assigned two benchmarks:

* **S&P 500 Index (GSPC)** as the primary benchmark for the equity component
* **Bloomberg US Aggregate Bond Index (AGG)** as the secondary benchmark for the fixed income component

As noted in the code comments, if this approach is incorrect, the secondary benchmark can be removed.

**Hard-coded Data**

The module uses hard-coded data for:

* Product mapping (ticker to product code)
* Portfolio data (portfolio code to product code)

This approach was chosen for simplicity and reliability, avoiding dependencies on external data sources.

**NULL Recipient Codes**

All recipient codes are set to "NULL" in the current implementation, as per the sample data pattern.

**6. Data Structure**

The module generates a DataFrame with the following columns:

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| PORTFOLIOCODE | String | Portfolio identifier (e.g., PORT001) |
| BENCHMARKCODE | String | Benchmark code (GSPC or AGG) |
| RECIPIENTCODE | String | Set to "NULL" |
| RANK | Integer | Benchmark priority (1 = primary, 2 = secondary) |

Each portfolio has **two** rows in the DataFrame:

* Primary benchmark association (GSPC, rank 1)
* Secondary benchmark association (AGG, rank 2)

**7. Error Handling**

The module implements minimal error handling as it primarily works with static, hard-coded data. The primary error handling mechanism is the try/except block around the display() function call to handle cases where the code is run in an environment without display support (e.g., non-notebook environments).

**8. Testing Considerations**

**Unit Testing**

* ✓ Verify that each portfolio gets exactly two benchmark associations
* ✓ Check that the rank values are correctly assigned (1 for GSPC, 2 for AGG)
* ✓ Ensure the RECIPIENTCODE is correctly set to "NULL"

**Integration Testing**

* ✓ Verify that the product codes in the portfolio data match those in the product mapping
* ✓ Check that the generated DataFrame structure matches the expected output

**9. Future Enhancements**

**Data Source Extensions**

* Read product and portfolio data from external files or databases
* Implement dynamic mapping of portfolios to products
* Add support for more complex portfolio-benchmark relationships

**Benchmark Customization**

* Allow for variable benchmark assignments based on portfolio characteristics
* Support more than two benchmarks per portfolio
* Implement custom benchmark weights

**Output Extensions**

* Add support for exporting to different formats (CSV, SQL, etc.)
* Add database integration for direct insertion
* Support multiple output formats

**10. Developer Checklist**

When working with or modifying this code, ensure you:

* ✓ Install all required dependencies
* ✓ Understand the dual benchmark approach for target date funds
* ✓ Review the hard-coded data if any product or portfolio changes are needed
* ✓ Test the output DataFrame structure after any changes
* ✓ Maintain the logging approach (no print statements)
* ✓ Update documentation if changing function signatures or adding features
* ✓ Consider dynamic data sources if the number of portfolios or products grows
* ✓ Verify that each portfolio has the correct number of benchmark associations
* ✓ Check that the RANK values are correctly assigned
* ✓ Consider the enhancements in section 9 if extending functionality