**Benchmark General Information Generator**

**Developer Documentation**

**Overview**

This Python module generates benchmark general information for target date funds. It creates a dataset that can be used to populate the AST\_MULTIASSET\_DB.DBO.BENCHMARKGENERALINFORMATION table. The generator produces realistic benchmark names and assigns them to specified benchmark codes (typically GSPC for equity and AGG for fixed income benchmarks).

**Table of Contents**

* Dependencies
* Setup and Configuration
* Data Flow
* Module Components
  + 4.1 Benchmark Name Generation
  + 4.2 Benchmark General Information Generation
  + 4.3 Name Pattern Analysis
  + 4.4 Main Execution Function
* Key Algorithms and 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
* random: For generating random elements of benchmark names
* logging: For structured logging (rather than print statements)

Ensure these dependencies are installed:

bash

CopyEdit

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.

python

CopyEdit

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

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

**3. Data Flow**

The module follows a clear data generation process:

* Define the benchmark codes to use (typically GSPC and AGG)
* Calculate how many records to generate for each benchmark code (fair distribution)
* For each benchmark code:
  + Generate the required number of records
  + Set all symbols to "BLANK"
  + Generate realistic benchmark names
  + Set ISBEGINOFDAYPERFORMANCE to False
* Validate the generated data
* Analyze name pattern distribution
* Return the complete benchmark general information DataFrame

**4. Module Components**

**4.1 Benchmark Name Generation**

* **Function:** generate\_name()
* **Purpose:** Generates realistic benchmark names for target date funds.
* **Process:**
  + Define multiple benchmark naming patterns:
    - Standard market benchmarks (e.g., "S&P Target Date 2045 Index")
    - Equity/Fixed Income blend benchmarks (e.g., "60% MSCI ACWI / 40% Bloomberg Agg")
    - Composite benchmarks (e.g., "Composite Target Date 2050 Index")
    - Common industry benchmark combinations (e.g., "Russell 3000 Target Date Benchmark")
    - Risk-based benchmark allocations (e.g., "Balanced 60% Equity Benchmark")
    - Standard industry glide path benchmarks (e.g., "S&P Target Date 2040 Index")
  + Randomly select one of these patterns
  + Generate a specific name by filling in the pattern with random components
* **Design Decisions:**
  + Uses lambda functions to define patterns for clearer structure
  + Incorporates randomization for variety in generated names
  + Includes realistic target date years (2025–2065)
  + Incorporates actual benchmark providers and indices used in the industry

**4.2 Benchmark General Information Generation**

* **Function:** generate\_benchmark\_general\_information(num\_records, benchmark\_codes)
* **Purpose:** Creates a DataFrame with benchmark general information records.
* **Process:**
  + Set default benchmark codes if none provided
  + Implement fair distribution algorithm to allocate records across benchmark codes
  + For each benchmark code:
    - Generate the allocated number of records
    - Set all symbols to "BLANK"
    - Generate realistic names for each record
    - Set ISBEGINOFDAYPERFORMANCE to False
  + Return all records as a pandas DataFrame
* **Key Parameters:**
  + num\_records: Total number of benchmark records to generate
  + benchmark\_codes: List of benchmark codes to use (default is ["GSPC"])
* **Fair Distribution Algorithm:**  
  The function uses an optimal distribution algorithm to allocate records across benchmark codes:
  + Calculate base records per code using integer division (num\_records // len(benchmark\_codes))
  + Find remaining records using modulo operation (num\_records % len(benchmark\_codes))
  + Distribute remaining records one at a time to each code
* **Rationale:**
  + Guarantees the most even possible distribution of records
  + Efficient with O(n) complexity
  + Produces deterministic results with the same inputs
  + Industry standard approach for resource allocation
  + Simple to understand and maintain

**4.3 Name Pattern Analysis**

* **Function:** analyze\_name\_patterns(df)
* **Purpose:** Analyzes the distribution of different patterns in the generated benchmark names.
* **Process:**
  + Initialize counters for different name patterns
  + Iterate through all benchmark names
  + Count occurrences of each pattern
  + Return a dictionary mapping patterns to counts
* **Design Decision:**  
  This function was added to provide insights into the distribution of generated benchmark name patterns, ensuring appropriate variety and realism.

**4.4 Main Execution Function**

* **Function:** main()
* **Purpose:** Orchestrates the benchmark general information generation process.
* **Process:**
  + Define benchmark codes and record count
  + Generate benchmark general information
  + Verify all symbols are set to "BLANK"
  + Log record count by benchmark code
  + Analyze and log name pattern distribution
  + Return the generated DataFrame

**5. Key Algorithms and Design Decisions**

**Fair Distribution Algorithm**  
The module uses an efficient algorithm to distribute records evenly across benchmark codes. This approach guarantees that:

* Each benchmark code gets approximately the same number of records
* The difference between any two benchmark codes is at most 1 record
* The distribution is deterministic and efficient

**Benchmark Name Generation**  
The benchmark name generation uses a pattern-based approach with randomization to create realistic benchmark names. The patterns were selected based on common industry practices for target date fund benchmarks.

**Symbol Standardization**  
All symbols are set to "BLANK" as requested. This design decision ensures consistency across all benchmark records.

**Performance Flag Standardization**  
All ISBEGINOFDAYPERFORMANCE flags are set to False, matching the observed pattern in sample data.

**6. Data Structure**

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| BENCHMARKCODE | String | Benchmark code (e.g., "GSPC", "AGG") |
| SYMBOL | String | Symbol (always "BLANK") |
| NAME | String | Generated benchmark name |
| ISBEGINOFDAYPERFORMANCE | Boolean | Performance timing flag (always False) |

**7. Error Handling**

The module implements the following error handling strategies:

* **Symbol Verification:**  
  Verifies that all generated symbols are set to "BLANK" and logs a warning if any unexpected values are found.
* **Default Values:**  
  Provides default values for function parameters to ensure the code runs even with minimal input.

**8. Testing Considerations**

When testing this module, consider the following:

**Unit Testing**

* Test the generate\_name() function for appropriate variety and realism
* Test the fair distribution algorithm with different record counts and benchmark code lists
* Verify that all symbols are correctly set to "BLANK"
* Check that all ISBEGINOFDAYPERFORMANCE flags are set to False

**Integration Testing**

* Test the complete data generation pipeline
* Verify that the generated DataFrame has the expected structure
* Check that the name pattern distribution is appropriate

**Edge Cases**

* Test with a single benchmark code
* Test with many benchmark codes
* Test with different record counts (0, 1, large numbers)

**9. Future Enhancements**

The module could be extended in several ways:

**Name Generation Enhancements**

* Add more sophisticated name generation patterns
* Implement context-aware name generation based on benchmark code
* Add support for custom name patterns

**Data Source Extensions**

* Read benchmark codes from configuration files
* Support external templates for benchmark names
* Add integration with actual benchmark databases

**Output Extensions**

* Add support for exporting to different formats (CSV, SQL, etc.)
* Add database integration for direct insertion
* Support more complex data structures

**10. Developer Checklist**

When working with or modifying this code, ensure you:

✓ Install all required dependencies  
✓ Understand the benchmark name generation patterns  
✓ Review the fair distribution algorithm if modifying record allocation  
✓ Test any changes with different benchmark codes and record counts  
✓ Maintain the "BLANK" symbol requirement for all records  
✓ Set ISBEGINOFDAYPERFORMANCE to False for all records  
✓ Verify the structure of the final DataFrame matches expectations  
✓ Maintain the logging approach (no print statements)  
✓ Update the name pattern analysis if adding new naming patterns  
✓ Consider the enhancements in section 9 if extending functionality