**Target Date Fund (TDF) Product Generator**

**Developer Documentation**

**Overview**

This Python module retrieves information about target date funds, maps ticker symbols to product codes, and generates a product master dataset. It uses Yahoo Finance as a data source and generates appropriate metadata for each fund.

**Table of Contents**

* Dependencies
* Setup and Configuration
* Data Flow
* Module Components
  + 4.1 Ticker Collection
  + 4.2 Fund Information Retrieval
  + 4.3 Name Inference
  + 4.4 Product Code Mapping
  + 4.5 Product Data Generation
* Key Algorithms and Design Decisions
* 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
* yfinance: For retrieving fund information from Yahoo Finance
* logging: For structured logging (rather than print statements)
* re: For regular expression pattern matching in fund name inference
* random: For generating random account codes
* tenacity (imported but not used): For potential future retry logic

Ensure all dependencies are installed:

bash

CopyEdit

pip install pandas numpy yfinance requests tenacity

**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 - %(name)s - %(levelname)s - %(message)s',

datefmt='%Y-%m-%d %H:%M:%S'

)

logger = logging.getLogger('tdf\_generator')

**3. Data Flow**

The module follows a clear pipeline approach:

* Retrieve target date fund ticker symbols
* Fetch fund information from Yahoo Finance for each ticker
* Create a mapping between tickers and fund names
* Generate product codes and map them to tickers
* Create product master data using all the collected information

**4. Module Components**

**4.1 Ticker Collection**

**Function**: get\_tdf\_tickers()  
**Purpose**: Returns a list of target date fund ticker symbols.  
**Design Decision**: The current implementation uses a hardcoded list of tickers. This approach was chosen for simplicity and reliability, avoiding potential issues with web scraping.  
**Future Enhancement**: This could be extended to:

* Scrape tickers from a financial website
* Read tickers from a configuration file
* Query a database for ticker information

**4.2 Fund Information Retrieval**

**Function**: get\_fund\_info\_from\_yahoo(tickers)  
**Purpose**: Retrieves detailed information about each fund from Yahoo Finance.  
**Process**:

* For each ticker in the provided list
* Use yfinance to fetch fund information
* Log success or failure for each ticker
* Return a dictionary mapping tickers to their information

**Error Handling**: Uses try/except to catch and log errors for individual tickers, ensuring the process continues even if some requests fail.  
**Design Decision**: Yahoo Finance was chosen as the data source because it's freely accessible and provides comprehensive fund information through the yfinance library.

**4.3 Name Inference**

**Functions**:

* infer\_fund\_name\_from\_ticker(ticker, info)
* create\_ticker\_to\_name\_mapping(tickers, fund\_info)

**Purpose**: These functions work together to create a reliable mapping between ticker symbols and human-readable fund names.

**Process**:

* First attempt to extract the name directly from Yahoo Finance data
* If that fails, infer the name using ticker symbol patterns and available information
* Apply pattern matching to identify:
  + Fund provider (Vanguard, Fidelity, etc.)
  + Target year (if applicable)
  + Fund type (Retirement Income, Target Date, etc.)

**Design Decisions**:

* Multiple fallback mechanisms ensure we get a name even with limited data
* Regular expressions identify target years in descriptions
* Provider inference from ticker patterns handles cases where provider info is missing

**Special Case Handling**: The code includes a hardcoded override for VSVNX (Vanguard Target Retirement 2070 Fund) due to difficulties retrieving this information from Yahoo Finance.

**4.4 Product Code Mapping**

**Functions**:

* generate\_product\_code(index)
* create\_ticker\_to\_product\_mapping(tickers)

**Purpose**: Creates a mapping between ticker symbols and product codes in the format PRD001, PRD002, etc.  
**Design Decision**: The sequential PRD### format was chosen for clarity and to ensure uniqueness. It starts from PRD001 and increments for each ticker.

**4.5 Product Data Generation**

**Functions**:

* generate\_account\_codes(account\_type)
* create\_product\_record(product\_code, fund\_name, strategy)
* generate\_product\_master\_data(ticker\_to\_name, ticker\_to\_product)

**Purpose**: These functions generate the final product master data with all necessary fields.

**Account Code Generation**:

* Uses weighted random selection between different patterns for realistic data
* Supports both performance and representative accounts
* Sometimes uses the same code for both account types (30% probability)

**Product Record Creation**:

* Creates a dictionary with all required product fields
* Uses the fund's ticker as its strategy code
* Sets appropriate defaults for fields like VEHICLECATEGORY and ASSETCLASS

**Master Data Generation**:

* Creates a DataFrame with all product records
* Ensures all expected columns are present
* Maintains 1:1 mapping between products and strategies

**5. Key Algorithms and Design Decisions**

**Name Inference Logic**

The name inference algorithm uses a cascading approach:

* First try to use Yahoo Finance data directly
* If that fails, look for patterns in description fields
* If still insufficient, use ticker symbol patterns
* Finally, fallback to a generic name  
  This multi-layered approach maximizes the chances of getting an accurate fund name.

**Account Code Generation**

Account codes are generated using weighted random selection from multiple patterns:

* Strategy-based codes (70% probability): ASTTDFCOMP, ASTTDFMF
* Numeric codes (10%): Random 4-digit numbers
* Letter-number codes (20%): LX### or OC####  
  This approach creates realistic-looking account codes while maintaining variability.

**Product Master Data Structure**

The product master data includes all fields required for the AST\_MULTIASSET\_DB.DBO.PRODUCTMASTER table:

* PRODUCTCODE: Generated PRD### code
* PRODUCTNAME: Fund name derived from Yahoo Finance or inference
* STRATEGY: The fund's ticker symbol
* VEHICLECATEGORY: "Pooled"
* VEHICLETYPE: "Mutual Fund"
* ASSETCLASS: "Multi-Asset"
* SHARECLASS: None
* PERFORMANCEACCOUNT: Generated account code
* REPRESENTATIVEACCOUNT: Generated account code
* ISMARKETED: True
* PARENTPRODUCTCODE: None

**6. Error Handling**

The module implements the following error handling strategies:

**Yahoo Finance API Errors**

* Individual ticker failures don't halt the entire process
* Errors are logged but processing continues
* The module can work with partial data if some API calls fail

**Missing Data Handling**

* Fallback mechanisms for missing fund names
* Default values for missing fields in the product data
* Column validation to ensure all expected columns exist

**7. Testing Considerations**

**Unit Testing**

* Test each function with mock data to verify correct operation
* Test name inference with various input patterns
* Test account code generation for distribution of patterns

**Integration Testing**

* Test the entire pipeline with a small set of real tickers
* Verify Yahoo Finance API connectivity
* Check that the generated DataFrame has the expected structure

**Error Case Testing**

* Test behavior when Yahoo Finance returns errors
* Test with invalid ticker symbols
* Test with missing or incomplete data from Yahoo Finance

**8. Future Enhancements**

The module could be extended in several ways:

**Data Source Extensions**

* Add alternative data sources beyond Yahoo Finance
* Implement a data source fallback mechanism
* Add caching of retrieved data to reduce API calls

**Configuration Improvements**

* Move hardcoded values to configuration files
* Add command-line arguments for flexible execution
* Implement environment variable configuration

**Performance Optimization**

* Add parallel processing for Yahoo Finance requests
* Implement request batching
* Add result caching

**Output Extensions**

* Add support for exporting to different formats (CSV, Excel, etc.)
* Add database integration for direct insertion
* Add reporting capabilities

**9. Developer Checklist**

When working with or modifying this code, ensure you:

✔️ Install all required dependencies

✔️ Understand the data flow from tickers to product master data

✔️ Review the name inference logic if modifying fund name generation

✔️ Test any changes against real target date fund tickers

✔️ Maintain the logging approach (no print statements)

✔️ Update documentation if changing function signatures or adding features

✔️ Consider retry logic for Yahoo Finance API calls if reliability is an issue

✔️ Review the hardcoded ticker list if expanding to new funds

✔️ Test with a small number of tickers before running with the full set

✔️ Check the generated account codes for proper distribution of patterns

✔️ Verify the structure of the final DataFrame matches expectations

✔️ Consider the enhancements in section 8 if extending functionality