Mankind Matrix Product Feed Validation Project

Project Overview

This document outlines the development of a Python-based system for validating and cleaning incoming product feeds for the Mankind Matrix platform. The system ensures all product entries follow proper formats and units before integration into our database.

Background

Mankind Matrix is a platform for AI Products and Semiconductors that offers various high-tech products such as:

- GPUs (SpectraForce X Series, NovaCore Vision, etc.)
- AI and Deep Learning systems (QuantumMind Systems, EdgeNexus Platform, etc.)
- Data Center Products
- Semiconductor Chips and Components

Our platform needs robust data validation to ensure product information is consistent, accurate, and properly formatted.

Project Goals

1. Create a reliable product feed validation system that:
    - Validates required fields and data types
    - Ensures products belong to valid categories
    - Standardizes units of measurement across product specifications
    - Identifies and logs validation errors
    - Outputs cleaned, validated data ready for database import
2. Develop a system that is:
    - Maintainable and expandable by the development team
    - Configurable to adapt to evolving product categories and specifications
    - Well-documented with clear logging
    - Usable as both a standalone script and an importable module

Technical Specifications

Input Requirements

The validator should accept product feeds in the following formats:

- CSV files
- JSON files
- Excel files (XLSX/XLS)

Expected Data Structure

Each product should contain the following minimum fields:

- `product_id`: Unique identifier (alphanumeric with hyphens/underscores)
- `name`: Product name
- `category`: Must be one of our predefined categories
- `brand`: Brand/manufacturer
- `description`: Product description
- `price`: Numeric price value
- `currency`: Currency code (USD, EUR, etc.)
- `in_stock`: Boolean availability status
- `specifications`: Dictionary of product specs with values and units
- `images`: List of image URLs or identifiers

Validation Rules

The system should perform the following validations:

1. **Field presence** - Check that all required fields exist
2. **Data type validation** - Ensure fields have correct data types
3. **Format validation** - Validate formats for specific fields (e.g., product_id pattern)
4. **Category validation** - Ensure products belong to valid categories
5. **Price range validation** - Check that prices are within reasonable ranges
6. **Currency validation** - Ensure currencies are valid codes
7. **Unit standardization** - Convert and standardize units in specifications

Unit Standardization Requirements

The system should standardize the following units:

- **Memory/Storage**: Standardize to GB (converting from MB, TB, etc.)
- **Frequency/Speed**: Standardize to GHz (converting from MHz, kHz, etc.)

- **Power**: Standardize to W (converting from kW, mW, etc.)

Implementation Tasks

Task 1: Core Validator Development

- Implement the `ProductFeedValidator` class with validation logic
- Create validation methods for different field types
- Implement unit standardization functions
- Set up logging and error tracking

Task 2: File Handling and Format Support

- Implement support for reading different file formats (CSV, JSON, Excel)
- Create output functionality for saving cleaned data
- Implement validation report generation

Task 3: Configuration and Extensibility

- Create a configuration system for validation rules
- Allow for custom category and validation rule definitions
- Make the validator extensible for future product types

Task 4: Testing and Quality Assurance

- Create test cases with sample product feeds
- Test all validation rules and edge cases
- Verify unit standardization accuracy
- Document test results and validator performance

Task 5: Documentation and Integration

- Create comprehensive documentation for the validator
- Provide usage examples for different scenarios
- Document integration options with the main Mankind Matrix platform
- Create user guide for operations team

Project Timeline

1. **Week 1**: Core validator implementation
2. **Week 2**: File handling and format support
3. **Week 3**: Configuration system and extensibility
4. **Week 4**: Testing, documentation, and integration

Usage Examples

Command Line Usage

```
# Basic usage
python product_feed_validator.py input_feed.csv

# Specify output format and path
python product_feed_validator.py input_feed.json --output cleaned_feed.json --format json

# Use custom configuration
python product_feed_validator.py input_feed.csv --config custom_rules.json
```

Programmatic Usage

```
from product_feed_validator import ProductFeedValidator

# Initialize validator
validator = ProductFeedValidator(config_path="custom_rules.json")

# Validate a file
cleaned_df, validation_errors = validator.validate_file("product_feed.csv")

# Save cleaned data
validator.save_cleaned_data(cleaned_df, "cleaned_feed.csv", "csv")

# Generate validation report
validator.generate_validation_report(validation_errors, "validation_report.md")
```

Future Enhancements

1. API endpoint for validation as a service
2. Real-time validation for product uploads
3. Integration with data quality monitoring system
4. Machine learning-based anomaly detection for product data
5. Support for additional file formats and data sources

Resources

- Sample product feeds for testing
- Product category definitions
- Unit conversion tables

- API documentation for integration

Contact Information

For questions or clarifications about this project, please contact the project manager.

Note: This prototype provides a foundation that can be extended based on specific requirements and scenarios encountered in production.