Pipeline Description for Brand Extraction

Pipeline Overview

This document provides a detailed description of the fuzzy matching pipeline, including each component's function, data flow, and processing steps. This pipeline is structured to process a large dataset efficiently by applying text extraction, transliteration, fuzzy matching, and parallelization techniques on product names.

Loading and Preprocessing Data

1. Load Data: The pipeline begins by loading the dataset containing product names and category information from CSV files (e.g., 'data.csv'). An Excel file ('pairs.xlsx') is loaded as a reference data to hold scraped brand names and categories for fuzzy matching.
2. Data Cleaning: Missing values are filled with empty strings, and duplicate rows are removed from the DataFrame.

Defining Patterns for Enclosed Value Extraction

A list of regular expression patterns is defined to identify and extract values enclosed within various symbols, such as «», ``, ", \<\<>>, etc. These patterns are passed as an argument to the processing function to identify specific keywords or terms within product names.

BrandExtractor Class

The `BrandExtractor` class contains the following methods:

1. extract_enclosed_values: Uses the defined patterns to find and extract enclosed terms from a given product name. It returns the first matched term.
2. fuzzy_match_brands: This function applies fuzzy matching to compare product names against brand names in the DataFrame. It incorporates Armenian to English transliteration, text cleaning, and uses a fuzzy matching library (RapidFuzz) based on conditions. The method returns brands with similarity scores above the threshold.
3. extract_english_in_armenian: Finds and extracts English characters embedded in Armenian text. This can be helpful for identifying English terms in product names.

Row Processing Function (process_row)

The `process_row` function is defined outside of the class and applies the following operations on each row of the DataFrame:

1. Extract enclosed values in product names using `extract_enclosed_values`.
2. Perform fuzzy matching of product names with brand names in the category using `fuzzy_match_brands`.
3. Extract English words embedded in Armenian text using `extract_english_in_armenian`.

Results are returned as a tuple containing three lists: enclosed values, exact matches, and English words.

Parallel Processing with apply_parallel_extraction

The `apply_parallel_extraction` function manages parallel processing of the dataset:

1. Chunking Data: The DataFrame is divided into chunks of rows, with each chunk processed separately.
2. Parallel Execution: Using `ProcessPoolExecutor`, the `process_row` function is applied in parallel to each row in a chunk. This maximizes efficiency for large datasets by utilizing multiple CPU cores.
3. Collecting Results: Results from all chunks are collected and assembled into a single DataFrame. A new column, 'combined', is created that stores each row’s results in a tuple format containing enclosed values, exact matches, and English words.

Saving Results

The final results are appended to the original dataset, and the combined results column is saved to a CSV file ('fuzzy_output_fixed_full.csv'). Optionally, results can also be saved to an Excel file for further analysis.

Pipeline Diagram

The following flowchart-style outline represents the data flow and operations:

1. Load Data
2. Define Patterns
3. BrandExtractor Class
- **a.** extract_enclosed_values
- **b.** fuzzy_match_brands
- **c.** extract_english_in_armenian
4. Row Processing Function (process_row)
5. Parallel Processing with apply_parallel_extraction
6. Save Results

Pipeline Description for Brand Matching

Pipeline Overview

The BrandMatcher class is designed to efficiently match brand names from a dataset to a reference list of brands. It also includes utilities for preprocessing brand names, handling extracted values, and applying advanced matching logic. This documentation provides a detailed description of the class and its methods.

Initialization (__init__)

The __init__ method initializes the BrandMatcher class with the following parameters:

- **ref_brands**(list): A list of reference brand names for matching.
- **ref_sub_categories** (list): A list of subcategories corresponding to the reference brands.

- **unnecessary_words** (list, optional): Words to be removed during preprocessing (default: None).
- **ratio** (function, optional): A similarity scoring function from rapidfuzz (default: fuzz.token_set_ratio).
- **score_threshold** (int, optional): A threshold score for determining additional matching criteria (default: 75).

Preprocessing (preprocess)

The preprocess method normalizes a brand name by:

- Converting it to lowercase.
- Removing unnecessary words (defined during initialization).
- Stripping extra spaces.

Args:
- **brand** (str): The brand name to preprocess.
Returns:
- str: The cleaned and normalized brand name.

Processing Extracted Values (process_extracted_values)

This method processes extracted values from a given row to determine the final brand name. It evaluates potential brand candidates, applies specific rules to filter valid brands, and normalizes or corrects the results based on predefined mappings.

Key Steps:

1. **Input Handling**:
2. Converts extracted_values from string to a list of lists.
3. Initializes a list of units of measurement to exclude irrelevant entries.
4. **Brand Validation**:
5. Checks if a value is a potential brand based on length and the absence of units.
6. Applies transliteration for Armenian or Cyrillic characters.
7. **Brand Selection**:
8. Prioritizes valid enclosed brands from extracted_values[0].
9. Falls back to matched brands (extracted_values[1]) no brand is identified from the first value.
10. If matched brands are also unavailable, extracted english names will be processed and selected (extracted_values[2]).
11. **Replacement Map**:
12. Corrects common brand misidentifications using a predefined dictionary.
13. **Output**:
14. Returns the final brand name, normalized and corrected as needed.

Matching (match)

The match method performs a similarity-based match for a single brand against the reference list. It uses rapidfuzz to calculate similarity scores and applies additional logic if the score exceeds a threshold.

Args:
- **brand** (str): The brand name to match.
- **sub_category** (str, optional): The subcategory to refine matching (default: None).
- **final_score** (float, optional): Precomputed score for additional logic (default: None).

Returns:
- tuple: A tuple containing the matched brand and its score.

Batch Matching (match_all)

The match_all method processes a list of brands and applies the match method to each. Progress tracking is enabled via tqdm for real-time feedback during execution.

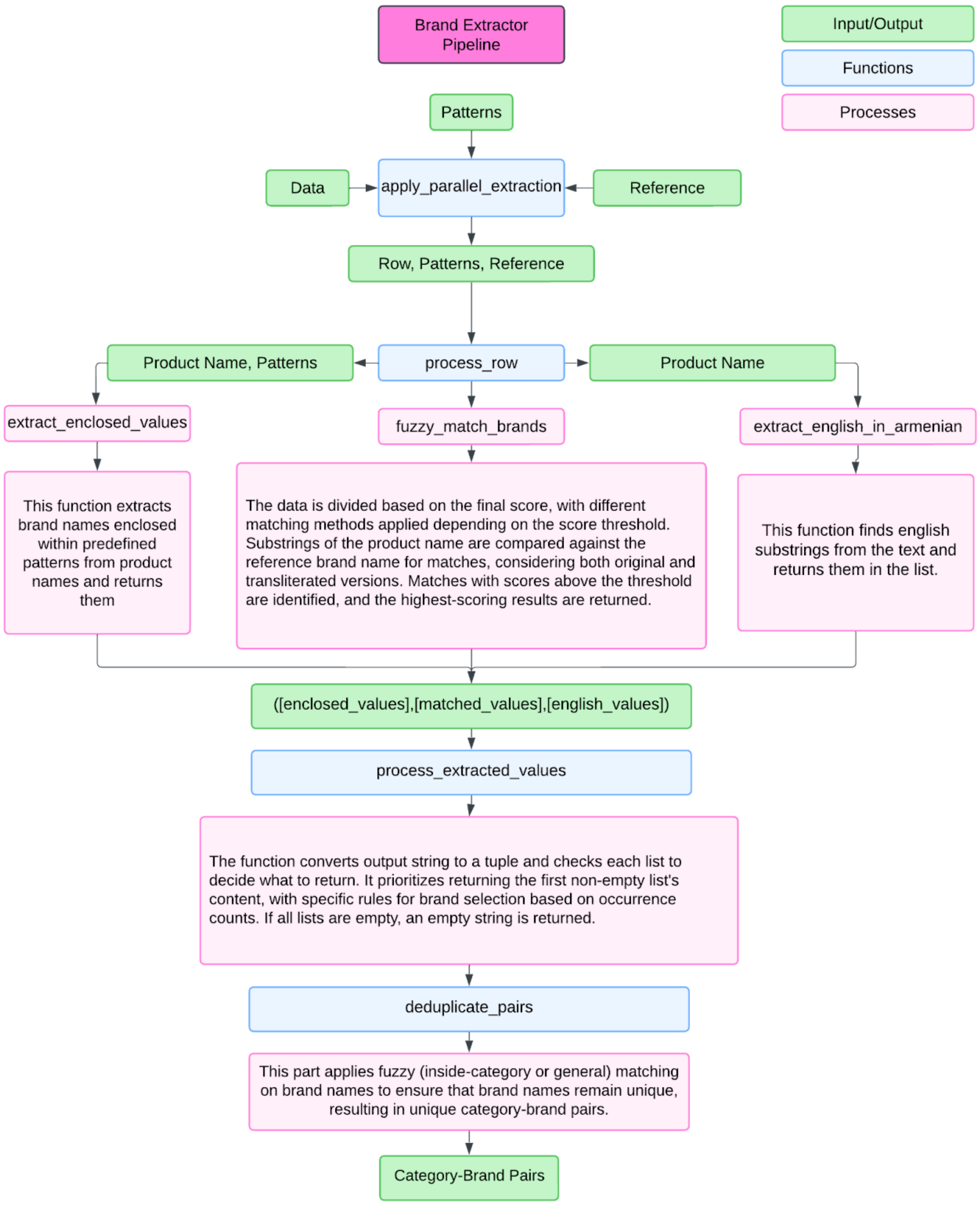
Args:
- **brands** (list): A list of brand names to match.
- **sub_categories** (list, optional): Corresponding subcategories for each brand (default: None).
- **final_scores** (list, optional): Precomputed scores for each brand (default: None).

Returns:
- list: A list of tuples containing matched brands and their scores.

Conclusion

The **BrandMatcher** class provides a robust and extensible framework for matching brand names in large datasets. It incorporates preprocessing utilities, advanced similarity scoring mechanisms, and the flexibility to handle additional logic for complex matching scenarios. The inclusion of configurable parameters such as unnecessary words, scoring functions, and thresholds makes the class highly adaptable to diverse use cases.

Through efficient batch processing capabilities, the class is well-suited for handling high-volume datasets, ensuring accurate and scalable brand matching solutions. Potential future enhancements may include additional preprocessing techniques, support for multilingual datasets, and the integration of advanced similarity algorithms to further improve performance and adaptability to various domains.



This function extracts brand names enclosed within predefined patterns from product names and returns them

The data is divided based on the final score, with different matching methods applied depending on the score threshold. Substrings of the product name are compared against the reference brand name for matches, considering both original and transliterated versions. Matches with scores above the threshold are identified, and the highest-scoring results are returned.

This function finds english substrings from the text and returns them in the list.

The function converts output string to a tuple and checks each list to decide what to return. It prioritizes returning the first non-empty list's content, with specific rules for brand selection based on occurrence counts. If all lists are empty, an empty string is returned.

This part applies fuzzy (inside-category or general) matching on brand names to ensure that brand names remain unique, resulting in unique category-brand pairs.