Text Preprocessing Steps and Challenges

1. Steps Taken for Each Preprocessing Activity:

a) Case Normalization:

- **Step:** Converted all text to lowercase.
- **Rationale:** To ensure consistency and avoid treating the same word with different capitalizations as distinct tokens during model training.

b) Emoji Removal:

- **Step:** Employed the emoji Python library to explicitly identify and remove all emoji characters from the text.
- **Rationale:** Emojis are often non-alphanumeric and might not contribute meaningfully to the semantic content for many NLP tasks. Removing them helps in focusing on the textual information.

c) Special Character Removal:

- **Step:** Utilized SpaCy's built-in token attributes to identify and filter out punctuation marks and whitespace characters during the tokenization process.
- **Rationale:** Punctuation and excessive whitespace generally do not add significant semantic value and can clutter the data. Removing them helps in obtaining cleaner tokens.

d) Tokenization:

- **Step:** Used SpaCy's language model (en_core_web_sm) to segment the text into individual words or tokens. SpaCy's tokenizer is designed to handle various linguistic nuances.
- **Rationale:** Tokenization is a fundamental step in NLP, breaking down text into units that can be analyzed by downstream models.

e) Stopword Removal:

- **Step:** Used the default set of English stopwords provided by SpaCy. During the tokenization process, tokens identified as stopwords were filtered out.
- **Rationale:** Stopwords (common words like "the," "is," "and") often occur frequently but carry little semantic weight. Removing them helps focus on the more informative words.
- **Custom Stopword Identification (Analysis Phase):**
    - We analyzed the frequency of the most common words in the dataset after initial tokenization.
    - Based on this analysis and the specific context of the "review_text" data, we would manually identify and justify any additional words that are frequent but not informative.

- These justified custom stopwords would then be added to a list and used to further filter the tokens during the preprocessing step

2. Before and After Sample Text Examples:

Example 1:

- **Before:** "I'll start by saying this is the first of four books. I wasn't expecting it to conclude... 🤔 "
- **After (Processed Tokens):** ['start', 'saying', 'books', 'expecting', 'conclude'] (Note: Contractions are split, and the emoji is removed).

Example 2:

- **Before:** "Aggie is Angela Lansbury who carries pocketbooks instead of handguns. The story was good."
- **After (Processed Tokens):** ['Aggie', 'Angela', 'Lansbury', 'carries', 'pocketbooks', 'instead', 'handguns', 'story', 'good']

3. Challenges Encountered and Solutions Applied:

a) Emoji Handling:

- **Challenge:** Emojis are Unicode characters that are not typically handled by standard alphanumeric or punctuation removal techniques. They can appear frequently in user-generated text and might not be relevant for semantic analysis.
- **Solution:** We integrated the emoji Python library. The emoji.replace_emoji(text, replace='') function was used to explicitly identify and remove all emoji characters from the text before further processing with SpaCy.

b) Language-Specific Text:

- **Challenge:** The current preprocessing pipeline is primarily designed for English text, utilizing SpaCy's en_core_web_sm model and a standard English stopword list. If the dataset contained reviews in other languages, the tokenization and stopword removal would not be effective.
- **Solution (Potential/Considerations):**
    - Language Detection: Implement a language detection library (e.g., langdetect) to identify the language of each review.
    - Language-Specific Models and Stopwords: Load the appropriate SpaCy language model and stopword list based on the detected language. SpaCy supports multiple languages, and NLTK also provides stopword lists for various languages.
    - Translation: As a more complex solution, consider translating non-English reviews to English before applying the current pipeline, but this might introduce noise or loss of nuances.

    - The current implementation assumes English text and does not include explicit language handling.

c) Contraction Handling:

- **Challenge:** Words with contractions (e.g., "didn't," "I'll") are often split by SpaCy's tokenizer into their constituent parts (e.g., "do," "n't"; "I," "will"). This might be desirable for some linguistic analyses but could be treated differently depending on the specific NLP task.
- **Solution (Considerations):**
    - Accept Default Splitting: For many tasks, treating the parts of a contraction as separate tokens is acceptable and can be linguistically informative.
    - Custom Tokenizer Rules: To keep contractions as single tokens or expand them (e.g., "didn't" to "did not"), custom tokenizer rules could be added to SpaCy. This would require a deeper understanding of SpaCy's tokenizer API.
    - Pre-processing Replacement: Before tokenization, a dictionary of common contractions could be used to replace them with their expanded forms (as explored with NLTK).
    - The current implementation uses SpaCy's default contraction splitting behavior.