Task 1: Text Extraction Tool

Title: Universal Text Extractor

Problem Statement: Develop a sophisticated Python tool/script to extract text from various file formats (PDF, DOCX, TXT) and images using OCR. The tool should have robust error handling and logging, and it should provide detailed validation of the extracted text.

Steps:

1. **File Handling:** Implement advanced functions to read and extract text from PDFs, DOCX, and TXT files, ensuring handling of edge cases and special characters.
2. **OCR Integration**: Use pytesseract to extract text from images, incorporating preprocessing steps to enhance OCR accuracy (e.g., noise removal, thresholding).
3. **Error Handling and Logging:** Develop a robust error handling mechanism to log detailed error messages for failed extractions.
4. **Validation:** Implement validation methods to check the completeness and accuracy of the extracted text. Include checks for language-specific characters and structure.

Prompts to Answer:

1. Explain the importance of OCR preprocessing and what techniques you used to improve OCR accuracy.
2. Discuss how you handled encoding issues in text extraction from different file formats.
3. Describe the methods you used to validate the accuracy of extracted text and why they were chosen.

Task 2: Comprehensive Text Preprocessing Pipeline

Title: NLP Text Preprocessing Pipeline

Problem Statement: Build an advanced text preprocessing pipeline that includes tokenization, removing stop words, stemming, lemmatization, and additional tasks like handling special characters, expanding contractions, and managing negations.

Steps:

1. **Tokenization:** Implement advanced tokenization techniques using NLTK and spaCy, handling punctuation and special characters.
2. **Stop Words Removal:** Create a customizable stop words list that can be extended or reduced based on specific use cases.
3. **Stemming and Lemmatization:** Use both stemming and lemmatization techniques, comparing their effectiveness on different types of text data.

4. **Additional Preprocessing Steps:** Implement functions for expanding contractions, managing negations, and handling special characters and emojis.

Prompts to Answer:

1. Compare stemming and lemmatization. When did you use one over the other, and why?
2. Explain the importance of handling contractions and negations in text preprocessing.
3. Discuss how special characters and emojis were managed in text preprocessing and the impact on subsequent analysis.

Task 3: Advanced Text Classification with Model Evaluation

Title: Text Classification with Model Evaluation

Problem Statement: Develop a sophisticated text classification model to categorize text data into multiple categories. Experiment with various algorithms and evaluate them using precision, recall, F1-score, and confusion matrix.

Steps:

1. **Data Preparation:** Preprocess a complex text dataset, ensuring balanced class distribution and handling missing values.
2. **Model Training:** Train multiple models (e.g., Logistic Regression, Decision Trees, SVM, Random Forest, and Neural Networks) using scikit-learn.
3. **Hyperparameter Tuning:** Perform hyperparameter tuning using GridSearchCV or RandomizedSearchCV.
4. **Evaluation:** Evaluate models using precision, recall, F1-score, confusion matrix, and ROC-AUC curve.
5. **Model Selection:** Select the best model based on evaluation metrics and business requirements.

Prompts to Answer:

1. Why was hyperparameter tuning important, and how did you decide which parameters to tune?
2. Explain the significance of the confusion matrix and how it helped in evaluating model performance.
3. Discuss when you preferred precision over recall and vice versa, and why.

Task 4: Sentiment Analysis Tool

Title: Sentiment Analysis System

Problem Statement: Build an advanced sentiment analysis tool that classifies text into positive, negative, or neutral categories. Incorporate feature engineering and deep learning models for improved accuracy.

Steps:

1. **Data Collection:** Collect and preprocess a sentiment analysis dataset, ensuring balanced representation of sentiments.
2. **Feature Engineering:** Use advanced feature extraction techniques (e.g., TF-IDF, word embeddings, BERT embeddings).
3. **Model Training:** Train traditional ML models (e.g., SVM, Random Forest) and deep learning models (e.g., LSTM, BERT) for sentiment classification.
4. **Evaluation:** Evaluate models using precision, recall, F1-score, and ROC-AUC curve.
5. **Model Interpretation:** Use SHAP or LIME for model interpretation to understand feature importance and decision-making process.

Prompts to Answer:

1. Compare traditional ML models and deep learning models for sentiment analysis. What are the pros and cons of each?
2. Discuss the role of word embeddings in improving text classification tasks.
3. Explain how model interpretation techniques like SHAP or LIME helped in understanding model predictions.

Task 5: Named Entity Recognition (NER) System

Title: Named Entity Recognition (NER) System

Problem Statement: Create an advanced NER system that identifies entities like names, locations, and organizations from text using NLTK and spaCy. Incorporate custom entity recognition and evaluate performance.

Steps:

1. **Data Preparation:** Collect and preprocess a dataset for NER, including custom entities relevant to a specific domain.
2. **NER Implementation:** Use NLTK and spaCy for entity recognition, incorporating custom entity rules and training custom NER models.
3. **Model Training:** Train and fine-tune spaCy models for better performance on custom entities.
4. **Evaluation:** Evaluate the models using precision, recall, F1-score, and custom evaluation metrics for domain-specific entities.
5. **Comparison:** Compare the performance of NLTK and spaCy models, providing detailed analysis and insights.

Prompts to Answer:

1. Describe how NER models work and the common challenges faced in entity recognition.
2. Explain why you needed to train a custom NER model and how you approached it.
3. Discuss the advantages of using spaCy over NLTK for NER tasks.