**Task 7: Essay Scoring System**

**Introduction**This code represents a Flask-based web application that evaluates essays by assigning scores based on various factors, including grammar checks, the number of lines, and relevance to a main concept. It is designed for an educational or automated essay scoring system. The app uses Natural Language Processing (NLP) techniques via the spacy and language\_tool\_python libraries to process and assess essays.**Key Functionalities**

* **Grammar Check:** Uses language\_tool\_python to check for grammatical errors and deduct points accordingly.
* **NLP and Named Entity Recognition (NER):** Uses spacy to extract the main concept from the essay using named entity recognition
* **Line Count:** Analyzes the number of lines in the essay, rewarding longer essays with higher scores.
* **Dataset Integration:** Compares essays against a preprocessed training and validation dataset to check if a similar essay exists, providing the stored score if a match is found.
* **Scoring Logic:** Adjusts the score based on lines, grammar, relevance to the extracted concept, and dataset matches.

**Detailed Breakdown**

1. **Imports and Initial Setup**The necessary libraries are imported, including Flask for web handling, pandas for dataset management, and spacy and language\_tool\_python for NLP tasks.language\_tool\_python.LanguageTool('en-US') is initialized to handle grammar checking.spacy.load('en\_core\_web\_sm') loads the English language model for NLP tasks such as entity recognition.
2. **Dataset Loading**The load\_data function attempts to load CSV files containing preprocessed training and validation sets from the Task7 directory. The os.path.join() method is used to ensure cross-platform compatibility. If the file cannot be loaded, an error is logged, and None is returned.
3. **Training and Validation Sets**: The training and validation datasets are loaded from CSV files (preprocessed\_train\_set.csv and preprocessed\_valid\_set.csv) that contain essays and their associated scores.
4. **Preprocessing**The preprocess\_essay function processes the essay by converting it to lowercase and removing extra spaces. This ensures uniformity when matching essays to entries in the dataset.
5. **Grammar Checking**The check\_grammar function checks the essay for grammatical errors using the initialized language\_tool\_python tool. It returns the number of grammatical errors in the essay.
6. **Line Counting**The count\_lines function counts the number of non-empty lines in the essay by splitting it into lines and ignoring empty ones. This is used as a factor to adjust the score, encouraging longer and more developed essays.
7. **Named Entity Recognition (NER) and Concept Extraction**The extract\_concept function leverages spacy to extract named entities such as organizations, people, and locations from the essay. If no relevant concepts (entities like 'ORG', 'PERSON', 'GPE') are found, the essay is assumed to be generic, and no penalty is imposed.
8. **Relevance Check**The is\_essay\_relevant\_to\_concept function determines whether the essay is relevant to the extracted concept. It compares the identified concepts to the rest of the essay, checking if more than half of the sentences mention the concept. If not, the essay is considered off-topic, leading to a score deduction.
9. **Dataset Matching**The get\_score\_from\_dataset function checks if the preprocessed essay exists in the training or validation sets. If a match is found, the stored score from the dataset is returned, bypassing further processing. This functionality encourages consistency in scoring essays that have been evaluated before.
10. **Scoring Function**The score\_essay function combines all elements of the essay analysis to generate a final score. The scoring starts at a default value of 3 and is adjusted based on the following factors:

* **Number of lines:** Essays with 10 or more lines get a 2-point bonus, while shorter essays are penalized.
* **Grammar errors:** More than 10 grammar errors result in a 2-point penalty, and fewer errors add points.
* **Concept relevance:** Essays that stray from the main concept lose 3 points.
* **Dataset match:** If the essay matches an entry in the dataset, the pre-recorded score is used.

The final score is clamped between 1 and 6, ensuring it remains within a valid range. An explanation of how the score was derived is also provided.

1. **Flask Routes**The Flask app defines two routes:

* **index():** Renders the HTML template (e.g., Task7.html), providing a user interface for submitting essays.
* **submit():** Handles the form submission, processes the essay via the score\_essay function, and returns the score along with an explanation as a JSON response.

1. **App Execution**The app is run on host 0.0.0.0 at port 5001 in debug mode. This setup allows easy debugging and testing during development.

**Scoring Logic**The score calculation involves multiple factors:

1. **Essay Length**:

* Essays with 10+ lines receive a bonus of 2 points.
* Essays with 5–9 lines get a 1-point bonus.
* Essays with fewer than 5 lines lose 1 point.

1. **Grammar Check:**

* More than 10 errors result in a 2-point deduction.
* Between 6 and 10 errors result in a 1-point deduction.
* Fewer than 5 errors add 1 point to the score.

1. **Concept Relevance:**If fewer than half of the sentences in the essay mention the main concept, the essay loses 3 points.
2. **Dataset Matching:**

If the essay matches one in the dataset, the stored score is used directly.

**Strengths**

* **Clear Structure:** The modular design ensures each functionality (e.g., grammar checking, line counting, concept extraction) is isolated, improving maintainability.
* **NLP Features:** The use of spacy for NER adds an advanced layer of semantic analysis, making the scoring more sophisticated.
* **Dataset Integration:** The ability to match essays from a dataset improves consistency and allows for handling repeated submissions efficiently.
* **Customizable Scoring:** The scoring rules are flexible and can be adjusted for different use cases or evaluation criteria.

**Conclusion**This Flask-based essay scoring system combines NLP techniques, grammar checking, and dataset matching to provide an automated, customizable essay evaluation tool. While the system is well-structured and performs effectively, there are opportunities for further optimization, especially in scaling and error detection. Overall, the system offers a promising foundation for automated educational tools or essay evaluation services.