Resume and Job Description Parsing System

# Overview

This system is designed to parse and extract relevant information from resumes and job descriptions, facilitating a more effective matching process. It uses a combination of custom-built functions, natural language processing (NLP) techniques, and a Flask-based microservice.

# Flask Microservice for Similarity Scoring

The Flask microservice provides HTTP endpoints to calculate various types of similarities between textual data. These endpoints accept JSON payloads and return similarity scores.

# Scoring Module Functions and Endpoints

## 1. calculate\_cosine\_similarity Endpoint

* Purpose: Computes cosine similarity between two lists of strings.
* Inputs: JSON payload with two lists (list1, list2).
* Process: Uses TfidfVectorizer to convert lists into TF-IDF vectors and calculates cosine similarity.
* Output: Floating-point number representing cosine similarity.
* Error Handling: Catches exceptions and returns a score of 0 on error.
* Usage: Comparing skill lists or textual attributes.

## 2. location\_similarity Endpoint

* Purpose: Calculates a similarity score based on geographical distances between locations in two lists.
* Inputs: JSON payload with two lists of location names.
* Process: Uses Nominatim for geocoding and calculates median geographical distance.
* Output: Normalized similarity score between 0 and 1.
* Error Handling: Handles geocoding failures and empty input lists.
* Usage: Geographical suitability assessment for job matching.

## 3. text\_similarity Endpoint

* Purpose: Computes similarity between two texts based on BERT embeddings.
* Inputs: JSON payload with two text strings (text1, text2).
* Process: Obtains BERT embeddings and calculates cosine similarity.
* Output: Similarity score based on embeddings.
* Error Handling: Logs errors during processing.
* Usage: Text-based comparisons, such as job responsibilities.

## 4. median\_similarity Endpoint

* Purpose: Calculates the median similarity score between elements of two lists.
* Inputs: JSON payload with two lists.
* Process: Compares each element pairwise and calculates the median of similarity scores.
* Output: Median similarity score.
* Error Handling: Handles empty lists and computation errors.
* Usage: Holistic similarity assessment between sets of elements.

## 5. industry\_similarity Endpoint

* Purpose: Calculates semantic similarity between industry-related texts.
* Inputs: JSON payload with two text strings (text1, text2).
* Process: Uses SentenceTransformer embeddings and computes cosine similarity.
* Output: Similarity score representing industry relevance.
* Error Handling: Catches exceptions during processing.
* Usage: Analyzing similarity between resumes and job descriptions in recruitment.

# Improvements

* Enhance Text Preprocessing: Advanced preprocessing for better handling of domain-specific language.
* Refinement of Embedding Techniques: Exploring domain-specific models for more accurate semantic analysis.
* Enhanced Geocoding Accuracy: Integrating detailed geocoding for more precise geographic matching.
* Semantic Analysis Refinement: Fine-tuning SentenceTransformer with domain-specific corpus for improved semantic alignment.