Documentation & Analysis for DeepQuery: A Masters' Statement of Purpose (SOP) Strength Analyzer

Group No: 30

(HOW TO RUN? – Check the README file.

DATASET? – Check the README file)

(At the end of this file, we've included all our initial work, research & hit-and-miss attempts. As in progress that we initially made and then scrapped in pursuit of better results).

Introduction

This documentation provides an in-depth explanation of a **Rule-Based Statement of Purpose (SOP) Evaluator** developed in Python. The evaluator assesses various aspects of an SOP, assigning numeric scores based on predefined evaluation metrics. Unlike machine learning models that learn patterns from data, this evaluator relies on a set of rules and keyword-based strategies to perform assessments.

Model Overview

Rule-Based vs. Learning-Based Models

Rule-Based Models and **Learning-Based Models** represent two distinct approaches in Natural Language Processing (NLP):

- **Rule-Based Models:**
    - **Definition:** Utilize predefined rules and heuristics to process and analyze text.
    - **Advantages:**
        - **Transparency:** Rules are explicit and understandable.
        - **Control:** Fine-grained control over evaluation criteria.
        - **No Training Data Required:** Operate based on rules without needing large datasets.
    - **Disadvantages:**
        - **Scalability:** Difficult to manage as complexity increases.
        - **Flexibility:** Less adaptable to nuances and variations in language.
        - **Maintenance:** Requires continuous updates to handle new patterns or edge cases.
- **Learning-Based Models (e.g., Neural Networks):**
    - **Definition:** Learn patterns and representations from large datasets using algorithms like neural networks.
    - **Advantages:**
        - **Adaptability:** Can capture complex language patterns and nuances.
        - **Scalability:** Easier to scale with data and computational resources.
        - **Performance:** Often achieve higher accuracy on complex tasks.
    - **Disadvantages:**
        - **Opacity:** Decision-making process is often a "black box."
        - **Data Dependency:** Require large, labeled datasets for training.

    - **Resource Intensive:** Need significant computational power for training and inference.

Current Model: The SOP evaluator discussed here is **rule-based**, leveraging keyword matching, semantic similarity, and various NLP techniques to assess different aspects of an SOP.

Library Installations and Imports

Installed Libraries

```
!pip install spacy nltk textblob language-tool-python transformers textstat
matplotlib seaborn
```

- **Purpose:** Install necessary libraries required for text processing, grammar checking, sentiment analysis, embedding generation, and data visualization.
- **Libraries:**
  - **SpaCy:** Advanced NLP library for tokenization, named entity recognition, and more.
  - **NLTK (Natural Language Toolkit):** Tools for text processing and linguistic data handling.
  - **TextBlob:** Simplified text processing library for sentiment analysis and more.
  - **LanguageTool-Python:** Interface to LanguageTool for grammar and spell checking.
  - **Transformers:** Library by Hugging Face for leveraging pretrained transformer models.
  - **TextStat:** Library for calculating text readability scores.
  - **Matplotlib & Seaborn:** Libraries for data visualization.

Imported Modules

```
import re
import json
from collections import defaultdict

import spacy
from textblob import TextBlob
import language_tool_python
import textstat
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import MinMaxScaler

import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger')
```

- **Standard Libraries:**
  - **re:** Regular expressions for text manipulation.
  - **json:** Handling JSON data.

    - **collections.defaultdict:** Dictionary subclass for default values.
- **NLP Libraries:**
    - **spacy:** For advanced NLP tasks like named entity recognition.
    - **TextBlob:** For sentiment analysis.
    - **language_tool_python:** For grammar and spell checking.
    - **textstat:** For readability assessments.
- **Machine Learning Utilities:**
    - **sklearn.metrics.pairwise.cosine_similarity:** To compute semantic similarity between text embeddings.
    - **sklearn.preprocessing.MinMaxScaler:** For scaling numerical scores.
- **NLTK:** Tokenizers and POS taggers required for sentence segmentation and linguistic analysis.

NLP Model Initialization

SpaCy

```
# Download the spaCy English model
!python -m spacy download en_core_web_sm

# Initialize NLP models
nlp = spacy.load("en_core_web_sm")
```

- **Purpose:** Initialize SpaCy's small English model for various NLP tasks.
- **Advantages:**
    - **Efficiency:** `en_core_web_sm` is lightweight and fast.
    - **Comprehensive Features:** Provides tokenization, part-of-speech tagging, named entity recognition, etc.
- **Disadvantages:**
    - **Accuracy:** Smaller models are less accurate than larger counterparts like `en_core_web_trf`.
    - **Limited Context Understanding:** May miss nuanced language patterns.

LanguageTool

```
tool = language_tool_python.LanguageTool('en-US')
```

- **Purpose:** Initialize LanguageTool for grammar and spell checking.
- **Advantages:**
    - **Comprehensive Error Detection:** Identifies a wide range of grammatical and stylistic errors.
    - **Ease of Use:** Simple API for integration.
- **Disadvantages:**
    - **Performance:** Can be slower for large texts due to the comprehensive rule set.
    - **Dependency on Rules:** May miss context-specific errors or advanced linguistic nuances.

BERT

```
from transformers import BertTokenizer, BertModel
```

```
import torch

# Initialize BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = BertModel.from_pretrained('bert-base-uncased')
```

- **Purpose:** Utilize BERT (Bidirectional Encoder Representations from Transformers) for generating text embeddings.
- **Advantages:**
  - **Rich Representations:** Captures contextual information and semantic meanings.
  - **Pretrained Knowledge:** Benefits from vast amounts of data used during pretraining.
- **Disadvantages:**
  - **Resource Intensive:** Requires significant computational resources for embedding generation.
  - **Static Embeddings:** Without fine-tuning, may not capture domain-specific nuances.

Preprocessing and Embedding Functions

`preprocess_text`

```
def preprocess_text(text):
    """
    Preprocess the SOP text by removing special characters, extra spaces, 
and converting to lowercase.
    """
    # Remove special characters and digits
    text = re.sub(r'[^A-Za-z\s]', '', text)
    # Convert to lowercase
    text = text.lower()
    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()
    return text
```

- **Purpose:** Clean the SOP text to ensure consistency in processing.
- **Steps:**
  1. **Remove Special Characters and Digits:** Eliminates non-alphabetic characters to focus on meaningful words.
  2. **Convert to Lowercase:** Standardizes text for uniform processing.
  3. **Remove Extra Spaces:** Ensures text is properly formatted without unnecessary whitespace.
- **Advantages:**
  - **Simplicity:** Straightforward cleaning enhances subsequent processing steps.
  - **Consistency:** Standardized text reduces variability, improving model performance.
- **Disadvantages:**
  - **Loss of Information:** Removing digits and special characters might discard context-specific information.
  - **Over-Cleaning:** Could eliminate meaningful punctuation that indicates sentence boundaries or emphasis.

`get_bert_embeddings`

```
def get_bert_embeddings(text):
    """
    Generate BERT embeddings for the given text.
    """
    inputs = tokenizer(text, return_tensors='pt', truncation=True,
max_length=512)
    with torch.no_grad():
        outputs = bert_model(**inputs)
    # Use the [CLS] token embedding as the representation of the entire
text
    cls_embedding = outputs.last_hidden_state[:, 0, :].numpy()
    return cls_embedding
```

- **Purpose:** Generate semantic embeddings for text using BERT.
- **Process:**
    1. **Tokenization:** Converts text into tokens that BERT can understand.
    2. **Embedding Generation:** Passes tokens through BERT to obtain contextual embeddings.
    3. **[CLS] Token Extraction:** Utilizes the embedding of the `[CLS]` token as a summary representation of the entire text.
- **Advantages:**
    - **Semantic Understanding:** Captures contextual and semantic relationships within the text.
    - **Versatility:** Applicable for various downstream tasks like similarity measurement.
- **Disadvantages:**
    - **Computational Overhead:** Generating embeddings for each text can be resource-intensive.
    - **Fixed Representation:** The `[CLS]` token may not capture all nuances of longer texts.

Evaluation Metrics Functions

The evaluator assesses the SOP across multiple criteria using a combination of keyword matching, semantic similarity, sentiment analysis, and other NLP techniques.

`evaluate_clear_narrative`

```
def evaluate_clear_narrative(sop, preprocessed_sop):
    """
    Evaluate the clarity and focus of the narrative by analyzing topic
consistency.
    """
    # Split SOP into sentences
    sentences = nltk.sent_tokenize(sop)
    # Vectorize sentences
    vectorizer = CountVectorizer(stop_words='english')
    sentence_vectors = vectorizer.fit_transform(sentences)

    # Apply LDA to identify topics
    lda = LatentDirichletAllocation(n_components=1, random_state=42)
    lda.fit(sentence_vectors)
    topic = lda.components_[0]
    # The highest contributing words to the topic
    top_words = [vectorizer.get_feature_names_out()[i] for i in
topic.argsort()[-10:]]

    # Calculate semantic similarity of each sentence to the main topic
    topic_text = ' '.join(top_words)
    topic_embedding = get_bert_embeddings(topic_text)
    narrative_embedding = get_bert_embeddings(preprocessed_sop)

    similarity = cosine_similarity(narrative_embedding, topic_embedding)
    # Higher similarity indicates a more focused narrative
    score = similarity[0][0] * 10
    return round(score, 1)
```

- **Purpose:** Assess the clarity and focus of the SOP by evaluating topic consistency.
- **Methodology:**
    1. **Sentence Segmentation:** Breaks the SOP into individual sentences.
    2. **Vectorization:** Converts sentences into numerical vectors using `CountVectorizer`.
    3. **Topic Modeling (LDA):** Identifies the primary topic within the SOP.
    4. **Top Words Extraction:** Retrieves the most significant words representing the topic.
    5. **Semantic Similarity:** Compares the overall narrative embedding with the topic embedding to gauge consistency.
- **Advantages:**
    - **Topic Consistency:** Ensures that the SOP maintains a clear and focused narrative.
    - **Semantic Evaluation:** Goes beyond keyword matching to understand contextual relationships.
- **Disadvantages:**
    - **Single Topic Limitation:** Assumes a single dominant topic, which might not capture multi-faceted SOPs.
    - **Sensitivity to Preprocessing:** Over-cleaning can affect the accuracy of vectorization and topic modeling.

`evaluate_customization_advanced`

```
def evaluate_customization_advanced(sop, program_keywords):
    """
    Evaluate customization by measuring semantic similarity between SOP and
program keywords.
    """
    # Join program keywords into a single string
    program_text = ' '.join(program_keywords)

    # Generate embeddings
    sop_embedding = get_bert_embeddings(preprocess_text(sop))
    program_embedding = get_bert_embeddings(preprocess_text(program_text))

    # Calculate cosine similarity
    similarity = cosine_similarity(sop_embedding, program_embedding)
    score = similarity[0][0] * 10
    return round(score, 1)
```

- **Purpose:** Measure how well the SOP is customized to the specific program or institution by evaluating semantic similarity with program-specific keywords.
- **Methodology:**
    1. **Keyword Aggregation:** Combines program-specific keywords into a single text block.
    2. **Embedding Generation:** Creates embeddings for both the SOP and the aggregated program keywords.
    3. **Similarity Measurement:** Uses cosine similarity to quantify alignment.
- **Advantages:**
    - **Semantic Alignment:** Captures nuanced similarities beyond mere keyword presence.
    - **Quantitative Assessment:** Provides a numerical score representing customization strength.
- **Disadvantages:**
    - **Dependence on Keyword List:** Effectiveness relies heavily on the comprehensiveness of `program_keywords`.
    - **Contextual Misalignment:** High similarity may not always equate to meaningful customization.

`evaluate_fit`

```
def evaluate_fit(sop, program_keywords):
    """
    Evaluate fit by measuring semantic similarity and alignment with
program keywords.
    """
    # Similar to customization
    return evaluate_customization_advanced(sop, program_keywords)
```

- **Purpose:** Assess the overall fit of the applicant with the program by reusing the customization evaluation.
- **Methodology:**
    - **Reuse of `evaluate_customization_advanced`:** Measures semantic similarity with program keywords.
- **Advantages:**
    - **Consistency:** Ensures that both customization and fit are evaluated similarly.
    - **Efficiency:** Avoids redundancy by leveraging existing functions.
- **Disadvantages:**
    - **Limited Differentiation:** May not capture distinct aspects of "fit" beyond customization.

`evaluate_specificity`

```
def evaluate_specificity(sop):
    """
    Evaluate specificity by counting named entities as proxies for detailed
examples.
    """
    doc = nlp(sop)
    num_entities = len(doc.ents)
    num_sentences = len(list(doc.sents))
    # More entities per sentence indicate higher specificity
    ratio = num_entities / num_sentences if num_sentences else 0
    score = min(ratio * 10, 10)
    return round(score, 1)
```

- **Purpose:** Gauge the level of specificity in the SOP by quantifying the presence of named entities, which often represent detailed examples.
- **Methodology:**
  1. **Named Entity Recognition (NER):** Identifies entities like names, organizations, dates, etc.
  2. **Sentence Segmentation:** Counts the number of sentences.
  3. **Ratio Calculation:** Determines the average number of entities per sentence.
  4. **Scoring:** Scales the ratio to a 0-10 range, capping at 10.
- **Advantages:**
  - **Objective Measurement:** Provides a quantitative assessment of specificity.
  - **Entity-Based Insight:** Named entities often correlate with detailed and concrete examples.
- **Disadvantages:**
  - **Entity Misinterpretation:** Not all entities indicate specificity; some might be irrelevant.
  - **Sentence Length Variability:** Longer sentences may naturally contain more entities, skewing the ratio.

`evaluate_structure`

```
def evaluate_structure(sop):
    """
    Evaluate structure by identifying distinct sections and logical flow.
    """
    intro_keywords = [
        'introduction', 'i am', 'i wish', 'my name is', 'purpose of my
application',
        'beginning my career', 'intending to', 'applying for', 'motivated
to pursue',
        'interested in', 'my journey', 'starting my academic career',
'aspire to',
        'my background', 'introduction to', 'opening statement', 'goal of',
'aim to',
        'seeking admission', 'desire to', 'reason for applying',
'background in',
        'my experience', 'commenced my studies', 'foundation of my',
'initial interest',
        'starting point', 'intended to', 'my objective', 'aiming to',
'planning to',
        'eager to', 'committed to', 'prepared to', 'ready to', 'keen on',
'excited to',
        'looking forward to', 'interested in pursuing', 'desire to engage',
'aspiring to',
        'motivated by', 'fascinated by', 'passionate about', 'drawn to'
    ]
    conclusion_keywords = [
        'thank you', 'looking forward', 'in conclusion', 'finally', 'to
summarize',
        'in summary', 'to conclude', 'in closing', 'ultimately', 'as a
final point',
        'in essence', 'in the end', 'to wrap up', 'in brief', 'to
finalize',
        'overall', 'all in all', 'in the final analysis', 'to summarize my
points',
        'to end with', 'to finish', 'to close', 'with gratitude',
'appreciate your consideration',
        'appreciate your time', 'thank you for your consideration', 'thank
you for your time',
        'appreciate your attention', 'grateful for your consideration',
'thank you for reviewing',
        'hope to hear from you', 'anticipating your response', 'awaiting
your decision',
        'eager to join', 'excited about the opportunity', 'hope to
contribute', 'looking forward to contributing',
        'enthusiastic about the possibility', 'keen to collaborate', 'ready
to embark', 'prepared to engage',
        'anticipate a positive response', 'hope to advance', 'aim to
excel', 'strive to succeed',
        'commit to excellence', 'dedicated to my goals', 'ready to pursue',
'motivated to excel',
        'prepared to contribute', 'eager to advance', 'hope to grow', 'look
forward to the challenge'
    ]

    intro = any(kw in sop.lower() for kw in intro_keywords)
    conclusion = any(kw in sop.lower() for kw in conclusion_keywords)

    score = 0
    if intro:
        score += 5
    if conclusion:
        score += 5
    return min(score, 10)
```

- **Purpose:** Assess the organizational structure of the SOP by detecting the presence of introductory and concluding sections based on keyword matching.
- **Methodology:**
    1. **Keyword Matching:** Searches for specific phrases indicative of introductions and conclusions.
    2. **Scoring:** Assigns points for the presence of introductory and concluding sections, capping the score at 10.
- **Advantages:**
    - **Simplicity:** Easy to implement and understand.
    - **Structural Insight:** Ensures that the SOP has clear beginning and end sections.
- **Disadvantages:**
    - **Rigid Criteria:** May miss structurally sound SOPs that use different phrasing.
    - **False Positives/Negatives:** Keywords may appear in contexts that don't signify section boundaries.

evaluate_authenticity

```
def evaluate_authenticity(sop):
    """
    Evaluate authenticity using sentiment analysis subjectivity.
    """
    blob = TextBlob(sop)
    subjectivity = blob.sentiment.subjectivity  # Range [0,1]
    score = subjectivity * 10
    return round(score, 1)
```

- **Purpose:** Measure the authenticity and personal touch of the SOP by analyzing sentiment subjectivity.
- **Methodology:**
  1. **Sentiment Analysis:** Utilizes TextBlob to assess the subjectivity of the text.
  2. **Scoring:** Scales the subjectivity score to a 0-10 range.
- **Advantages:**
  - **Personalization Indicator:** Higher subjectivity often correlates with personal narratives.
  - **Quantitative Assessment:** Provides a numerical measure of authenticity.
- **Disadvantages:**
  - **Overgeneralization:** High subjectivity doesn't always mean authenticity; it could indicate emotional bias.
  - **Limited Nuance:** May not capture the depth of personal experiences accurately.

`evaluate_grammar`

```
def evaluate_grammar(sop):
    """
    Evaluate grammar by counting errors identified by LanguageTool.
    """
    matches = tool.check(sop)
    num_errors = len(matches)
    sop_length = len(sop)
    error_rate = num_errors / sop_length if sop_length else 0
    score = max(1, 10 - (error_rate * 100))  # Simplistic scaling
    return round(score, 1)
```

- **Purpose:** Assess the grammatical correctness of the SOP by detecting and quantifying errors.
- **Methodology:**
  1. **Error Detection:** Uses LanguageTool to identify grammatical, spelling, and punctuation errors.
  2. **Error Rate Calculation:** Computes the ratio of errors to the total length of the SOP.
  3. **Scoring:** Inversely scales the error rate to assign a score between 1 and 10.
- **Advantages:**
  - **Automated Grammar Checking:** Efficiently detects a wide range of grammatical issues.
  - **Objective Scoring:** Provides a consistent metric based on error prevalence.
- **Disadvantages:**
  - **Error Context Ignorance:** May flag contextually correct sentences as erroneous.
  - **Simplistic Scaling:** The linear reduction based on error rate may not reflect the severity of errors.

`evaluate_balanced_content`

```
def evaluate_balanced_content(sop):
    """
    Assess balance between personal anecdotes and academic/professional
achievements.
    """
    personal_pronouns = ['i ', 'my ', 'me ', 'mine ', 'myself ']
    personal_count = sum(sop.lower().count(pron) for pron in
personal_pronouns)

    technical_keywords = ['mechanical', 'engineering', 'research',
'project', 'thesis', 'development', 'design', 'analysis']
    technical_count = sum(sop.lower().count(kw) for kw in
technical_keywords)

    if technical_count == 0:
        score = 5
    else:
        ratio = personal_count / technical_count
        # Ideal ratio is around 1 for balance
        score = max(1, min(10, 10 - abs(ratio - 1) * 10))
    return round(score, 1)
```

- **Purpose:** Evaluate the balance between personal storytelling and academic or professional accomplishments in the SOP.
- **Methodology:**
    1. **Pronoun Counting:** Counts occurrences of personal pronouns to estimate personal anecdotes.
    2. **Technical Keyword Counting:** Counts technical terms to gauge academic/professional content.
    3. **Ratio Calculation:** Computes the ratio of personal pronouns to technical keywords.
    4. **Scoring:** Assigns higher scores for ratios close to 1, indicating balance.
- **Advantages:**
    - **Balance Assessment:** Ensures that the SOP doesn't overly focus on personal or technical aspects.
    - **Quantitative Measure:** Provides a clear metric for content balance.
- **Disadvantages:**
    - **Simplistic Indicators:** Pronouns and technical keywords may not accurately reflect content balance.
    - **Contextual Limitations:** High usage of pronouns or technical terms doesn't necessarily indicate imbalance.

`evaluate_clarity_of_goals`

```
def evaluate_clarity_of_goals(sop):
    """
    Assess clarity of short-term and long-term goals.
    """
    short_term_keywords = ['immediately', 'short-term', 'in the near
future', 'next step', 'upcoming']
    long_term_keywords = ['long-term', 'ultimately', 'future', 'career',
'over the years', 'eventually']

    sop_lower = sop.lower()
    short_term = any(kw in sop_lower for kw in short_term_keywords)
    long_term = any(kw in sop_lower for kw in long_term_keywords)

    score = 0
    if short_term:
        score += 5
    if long_term:
        score += 5
    return min(score, 10)
```

- **Purpose:** Measure how clearly the applicant articulates their short-term and long-term goals within the SOP.
- **Methodology:**
    1. **Keyword Detection:** Searches for phrases indicative of short-term and long-term goals.
    2. **Scoring:** Assigns points for the presence of short-term and long-term goal indicators.
- **Advantages:**
    - **Goal Alignment:** Ensures that the SOP clearly outlines the applicant's aspirations.
    - **Simple Implementation:** Easy to implement using keyword matching.
- **Disadvantages:**
    - **Keyword Dependency:** May miss well-articulated goals that use different phrasing.
    - **False Indicators:** Presence of keywords doesn't guarantee clarity or depth in goal articulation.

`evaluate_skills`

```
def evaluate_skills(sop, skills_list):
    """
    Evaluate relevant skills using semantic similarity.
    """
    skills_text = ' '.join(skills_list)
    sop_embedding = get_bert_embeddings(preprocess_text(sop))
    skills_embedding = get_bert_embeddings(preprocess_text(skills_text))

    similarity = cosine_similarity(sop_embedding, skills_embedding)
    score = similarity[0][0] * 10
    return round(score, 1)
```

- **Purpose:** Assess the relevance and presence of the applicant's skills in the SOP by measuring semantic similarity with a predefined skills list.
- **Methodology:**
    1. **Skills Aggregation:** Combines all relevant skills into a single text block.
    2. **Embedding Generation:** Creates embeddings for both the SOP and the aggregated skills text.
    3. **Similarity Measurement:** Computes cosine similarity to quantify the alignment.
- **Advantages:**
    - **Comprehensive Skill Assessment:** Evaluates the presence and relevance of a wide range of skills.
    - **Semantic Depth:** Goes beyond keyword presence to understand contextual relevance.
- **Disadvantages:**
    - **Over-Reliance on Embeddings:** Similarity scores may not accurately reflect specific skill mentions.
    - **Skill List Exhaustiveness:** Effectiveness depends on the comprehensiveness of `skills_list`.

`evaluate_personal_attributes_advanced`

```
def evaluate_personal_attributes_advanced(sop, attributes_list):
    """
    Evaluate personal attributes using semantic similarity.
    """
    attributes_text = ' '.join(attributes_list)
    sop_embedding = get_bert_embeddings(preprocess_text(sop))
    attributes_embedding =
get_bert_embeddings(preprocess_text(attributes_text))

    similarity = cosine_similarity(sop_embedding, attributes_embedding)
    score = similarity[0][0] * 10
    return round(score, 1)
```

- **Purpose:** Measure the presence and alignment of personal attributes in the SOP through semantic similarity with a predefined attributes list.
- **Methodology:**
    1. **Attributes Aggregation:** Combines personal attributes into a single text block.
    2. **Embedding Generation:** Creates embeddings for both the SOP and the aggregated attributes text.
    3. **Similarity Measurement:** Computes cosine similarity to quantify alignment.
- **Advantages:**
    - **Comprehensive Evaluation:** Assesses a broad range of personal attributes.
    - **Semantic Understanding:** Captures nuanced expressions of attributes beyond explicit mentions.
- **Disadvantages:**
    - **Embedding Limitations:** Similarity scores may not accurately reflect the depth or context of attribute mentions.
    - **List Dependency:** Effectiveness relies on the completeness of `attributes_list`.

`evaluate_professionalism`

```
def evaluate_professionalism(sop):
    """
    Assess professionalism using sentiment analysis and keyword presence.
    """
    professionalism_keywords = ['prepared', 'ready', 'commitment',
'dedication', 'professional',
                                'preparedness', 'prepared to', 'ready to',
'committed to', 'dedicated to',
                                'prepared for', 'ready for', 'commitment
to', 'dedicated for']
    matches = sum(1 for kw in professionalism_keywords if kw in
sop.lower())
    score = min(matches * 2, 10)  # Each keyword adds 2 points
    return round(score, 1)
```

- **Purpose:** Evaluate the level of professionalism demonstrated in the SOP by detecting the presence of specific professionalism-related keywords.
- **Methodology:**
    1. **Keyword Matching:** Counts occurrences of predefined professionalism keywords.

2. **Scoring:** Assigns points based on the number of keyword matches, capping at 10.

- **Advantages:**
    - **Focus on Professionalism:** Ensures that the SOP conveys a professional tone.
    - **Simple and Effective:** Easy to implement and interpret.
- **Disadvantages:**
    - **Keyword Overemphasis:** May overlook professionalism demonstrated through context rather than explicit keywords.
    - **False Positives:** Keywords may appear in contexts that don't signify professionalism.

`evaluate_usp`

```
def evaluate_usp(sop, usp_phrases):
    """
    Identify unique selling propositions using semantic similarity.
    """
    usp_text = ' '.join(usp_phrases)
    sop_embedding = get_bert_embeddings(preprocess_text(sop))
    usp_embedding = get_bert_embeddings(preprocess_text(usp_text))

    similarity = cosine_similarity(sop_embedding, usp_embedding)
    score = similarity[0][0] * 10
    return round(score, 1)
```

- **Purpose:** Assess the uniqueness and standout qualities of the applicant by measuring the semantic similarity between the SOP and predefined unique selling proposition (USP) phrases.
- **Methodology:**
    1. **USP Aggregation:** Combines USP phrases into a single text block.
    2. **Embedding Generation:** Creates embeddings for both the SOP and the aggregated USP text.
    3. **Similarity Measurement:** Computes cosine similarity to quantify alignment.
- **Advantages:**
    - **Highlighting Uniqueness:** Encourages applicants to showcase unique qualities and accomplishments.
    - **Semantic Depth:** Captures nuanced expressions of USPs beyond keyword matching.
- **Disadvantages:**
    - **Embedding Sensitivity:** Similarity scores may not directly correlate with actual uniqueness.
    - **List Completeness:** Effectiveness depends on the comprehensiveness of `usp_phrases`.

`evaluate_formatting`

```
def evaluate_formatting(sop):
    """
    Evaluate adherence to formatting by checking paragraph structure.
    """
    paragraphs = sop.strip().split('\n\n')
    if len(paragraphs) < 3:
        return 4  # Not enough paragraphs
    else:
        return 10  # Assuming good formatting
```

- **Purpose:** Assess the structural formatting of the SOP by analyzing paragraph divisions.
- **Methodology:**
  1. **Paragraph Counting:** Splits the SOP into paragraphs based on double newline characters.
  2. **Scoring:** Assigns a higher score if the SOP contains at least three paragraphs, indicating a structured format.
- **Advantages:**
  - **Ensures Basic Structure:** Verifies that the SOP has a clear introduction, body, and conclusion.
  - **Simplicity:** Easy to implement without complex processing.
- **Disadvantages:**
  - **Rigid Criteria:** May penalize well-formatted SOPs that use different paragraph structures.
  - **Limited Insight:** Doesn't assess the quality or logical flow within paragraphs.

`evaluate_pitfalls`

```
def evaluate_pitfalls(sop, clichés_list):
    """
    Assess the presence of clichés and deduct points accordingly.
    """
    matches = sum(1 for c in clichés_list if c in sop.lower())
    score = max(1, 10 - (matches * 2))
    return round(score, 1)
```

- **Purpose:** Identify and penalize the use of overused or clichéd phrases in the SOP.
- **Methodology:**
  1. **Cliché Detection:** Counts occurrences of predefined cliché phrases.
  2. **Scoring:** Deducts points based on the number of clichés detected, ensuring a minimum score of 1.
- **Advantages:**
  - **Encourages Originality:** Discourages the use of generic or overused phrases.
  - **Quantitative Penalty:** Provides a clear mechanism for penalizing clichés.
- **Disadvantages:**
  - **Contextual Misinterpretation:** Clichés might be used appropriately in context.
  - **False Penalties:** Penalizes phrases that are part of a larger meaningful sentence.

`evaluate_consistency`

```
def evaluate_consistency(sop, resume_text=None, 
recommendation_letters=None):
    """
    Placeholder for evaluating consistency with other application 
materials.
    """
    # Implement consistency checks here by comparing key information
    return 5  # Neutral score
```

- **Purpose:** Intended to assess the alignment and consistency of the SOP with other application materials like resumes and recommendation letters.
- **Methodology:**
  - **Currently a Placeholder:** The function returns a neutral score and requires implementation.
- **Advantages:**
  - **Comprehensive Evaluation:** Ensures that the applicant presents a cohesive narrative across all materials.
  - **Potential for Deep Insight:** When implemented, can detect discrepancies or reinforce consistency.
- **Disadvantages:**
  - **Incomplete Implementation:** Currently non-functional, providing no actual assessment.
  - **Complexity:** Requires sophisticated text comparison techniques to effectively evaluate consistency.

`evaluate_professional_language`

```
def evaluate_professional_language(sop):
    """
    Assess the use of professional and precise language by analyzing word 
complexity.
    """
    words = sop.split()
    total_words = len(words)
    complex_words = [word for word in words if len(word) > 7]
    ratio = len(complex_words) / total_words if total_words else 0
    score = ratio * 10  # Higher ratio implies more professional language
    score = min(max(score, 1), 10)
    return round(score, 1)
```

- **Purpose:** Measure the professionalism of language used in the SOP by analyzing the complexity of words.
- **Methodology:**
  1. **Word Counting:** Splits the SOP into individual words.
  2. **Complex Word Identification:** Counts words longer than seven characters as proxies for complexity.
  3. **Ratio Calculation:** Determines the proportion of complex words to total words.
  4. **Scoring:** Scales the ratio to a 0-10 range, ensuring a minimum score of 1.
- **Advantages:**
  - **Professionalism Indicator:** Complex vocabulary often correlates with a professional tone.
  - **Quantitative Measure:** Provides a numerical score reflecting language sophistication.
- **Disadvantages:**
  - **Overemphasis on Length:** Long words aren't always more professional or precise.
  - **Contextual Limitations:** Complex words used inappropriately can detract from professionalism.

`evaluate_positive_tone`

```
def evaluate_positive_tone(sop):
    """
    Assess the positivity of the SOP using sentiment analysis.
    """
    blob = TextBlob(sop)
    polarity = blob.sentiment.polarity  # Range [-1,1]
    score = (polarity + 1) / 2 * 10  # Normalize to [0,10]
    return round(score, 1)
```

- **Purpose:** Measure the overall positivity of the SOP, ensuring a constructive and optimistic tone.
- **Methodology:**
    1. **Sentiment Analysis:** Utilizes TextBlob to determine the polarity of the text.
    2. **Normalization:** Converts the polarity score from [-1,1] to [0,10].
- **Advantages:**
    - **Tone Assessment:** Ensures that the SOP maintains a positive and constructive tone.
    - **Automated Sentiment Evaluation:** Efficiently quantifies sentiment without manual review.
- **Disadvantages:**
    - **Context Insensitivity:** Sentiment analysis may misinterpret nuanced language or mixed sentiments.
    - **Overgeneralization:** High positivity doesn't necessarily equate to effective communication.

`evaluate_readability_score`

```
def evaluate_readability_score(sop):
    """
    Assess readability using textstat's Flesch Reading Ease score.
    """
    flesch = textstat.flesch_reading_ease(sop)
    # Map Flesch score to 1-10 scale
    if flesch >= 90:
        score = 10
    elif flesch >= 80:
        score = 9
    elif flesch >= 70:
        score = 8
    elif flesch >= 60:
        score = 7
    elif flesch >= 50:
        score = 6
    elif flesch >= 40:
        score = 5
    elif flesch >= 30:
        score = 4
    elif flesch >= 20:
        score = 3
    else:
        score = 2
    return score
```

- **Purpose:** Evaluate the readability of the SOP to ensure it is accessible and comprehensible.
- **Methodology:**

1. **Flesch Reading Ease Calculation:** Uses `textstat` to compute the Flesch score.
2. **Scoring:** Maps the Flesch score to a 1-10 scale based on predefined ranges.

- **Advantages:**
  - **Readability Insight:** Ensures that the SOP is neither too complex nor too simplistic.
  - **Automated Evaluation:** Quickly assesses readability without manual intervention.
- **Disadvantages:**
  - **Language Complexity Limitation:** Flesch scores are based on syllable and sentence counts, not on contextual complexity.
  - **Score Mapping Subjectivity:** The chosen mapping ranges may not perfectly align with desired readability standards.

Advanced Evaluation and Visualization

`evaluate_sop_advanced`

```
def evaluate_sop_advanced(sop, program_keywords, skills_list,
attributes_list, usp_phrases, clichés_list):
    """
    Evaluate the SOP across all criteria using advanced NLP techniques.
    """
    # Preprocess SOP
    preprocessed_sop = preprocess_text(sop)

    # Evaluate each criterion
    scores = {}
    scores["Clear and Focused Narrative"] = evaluate_clear_narrative(sop,
preprocessed_sop)
    scores["Customization to the Specific Program or Institution"] =
evaluate_customization_advanced(sop, program_keywords)
    scores["Specificity and Detail with Concrete Examples"] =
evaluate_specificity(sop)
    scores["Strong Structure and Organization"] = evaluate_structure(sop)
    scores["Authenticity and Personality"] = evaluate_authenticity(sop)
    scores["Demonstrated Fit with the Program/Institution"] =
evaluate_fit(sop, program_keywords)
    scores["Error-Free Writing (Grammar, Spelling, Punctuation)"] =
evaluate_grammar(sop)
    scores["Balanced Content (Personal Anecdotes vs. Academic/Professional
Achievements)"] = evaluate_balanced_content(sop)
    scores["Clarity of Goals (Short-term and Long-term)"] =
evaluate_clarity_of_goals(sop)
    scores["Relevant Skills and Qualifications"] = evaluate_skills(sop,
skills_list)
    scores["Personal Attributes (e.g., Resilience, Adaptability)"] =
evaluate_personal_attributes_advanced(sop, attributes_list)
    scores["Professionalism and Preparedness"] =
evaluate_professionalism(sop)
    scores["Unique Selling Proposition (USP)"] = evaluate_usp(sop,
usp_phrases)
    scores["Adherence to Formatting Guidelines"] = evaluate_formatting(sop)
    scores["Avoidance of Common Pitfalls (e.g., Clichés, Irrelevant
Information)"] = evaluate_pitfalls(sop, clichés_list)
    scores["Consistency with Other Application Materials"] =
evaluate_consistency(sop)  # Placeholder
    scores["Use of Professional Language"] =
evaluate_professional_language(sop)
    scores["Positive Tone and Constructive Framing"] =
evaluate_positive_tone(sop)
    scores["Readability"] = evaluate_readability_score(sop)

    return scores
```

- **Purpose:** Aggregate all individual evaluation metrics to provide a comprehensive assessment of the SOP.
- **Methodology:**
    1. **Preprocessing:** Cleans the SOP text to ensure consistency in processing.
    2. **Metric Evaluation:** Calls each evaluation function to assess different aspects of the SOP.
    3. **Scoring Aggregation:** Compiles all individual scores into a dictionary for easy interpretation.
- **Advantages:**
    - **Comprehensive Assessment:** Evaluates multiple facets of the SOP, providing a holistic score.
    - **Modular Design:** Facilitates easy addition or modification of evaluation metrics.
- **Disadvantages:**
    - **Dependency on Individual Metrics:** The overall score is only as reliable as the individual metric evaluations.
    - **Scoring Conflicts:** Different metrics might offer conflicting assessments, complicating the interpretation.

`visualize_scores`

```
import matplotlib.pyplot as plt
import seaborn as sns

def visualize_scores(scores):
    """
    Visualize the SOP evaluation scores using a horizontal bar chart.
    """
    criteria = list(scores.keys())
    values = list(scores.values())

    plt.figure(figsize=(12, 10))
    sns.barplot(x=values, y=criteria, palette='viridis')
    plt.xlabel('Scores')
    plt.title('Advanced SOP Evaluation Scores')
    plt.xlim(0,10)
    plt.show()
```

- **Purpose:** Provide a visual representation of the SOP evaluation scores for easy interpretation and comparison.
- **Methodology:**
    1. **Data Extraction:** Retrieves evaluation criteria and their corresponding scores.

2. **Visualization:** Utilizes Seaborn's barplot to display scores in a horizontal bar chart.
3. **Customization:** Sets figure size, color palette, labels, title, and axis limits for clarity.

- **Advantages:**
  - **Enhanced Readability:** Visual charts make it easier to understand strengths and weaknesses.
  - **Comparative Insight:** Facilitates quick comparison across different evaluation criteria.
- **Disadvantages:**
  - **Static Visualization:** Does not allow for interactive exploration of data.
  - **Overcrowding:** With too many criteria, the chart can become cluttered and hard to read.

Data Definitions

Program Keywords (Refer how to change program keywords according to your program before you run the jupyter notebook in the 'HOW TO RUN?' section in the README file)

```
program_keywords = [
    'solid mechanics', 'mechanical engineering', 'finite elements', 
'computational modeling',
    'experimental mechanics', 'composite materials', 'multi-scale 
modeling', 'finite elements method',
    'mechanical properties', 'stress analysis', 'structural analysis', 
'dynamic analysis',
    'material characterization', 'thermodynamics', 'fluid mechanics', 
'materials science',
    'strain measurement', 'deformation', 'load analysis', 'structural 
integrity', 'vibration analysis',
    'nonlinear dynamics', 'simulation', 'CAD modeling', 'FEA', 'ANSYS', 
'Abaqus', 'SolidWorks',
    'thermo-mechanical analysis', 'crack propagation', 'material fatigue', 
'failure analysis',
    'plastic deformation', 'elasticity', 'yield strength', 'tensile 
testing', 'compressive testing',
    'shear testing', 'torsion testing', 'biomechanics', 'nanomechanics', 
'strain rate', 'stress-strain curve',
    'fracture mechanics', 'finite volume method', 'boundary element 
method', 'mesh generation',
    'computational fluid dynamics', 'CFD', 'material modeling', 
'viscoelasticity', 'thermal expansion',
    'heat transfer', 'energy absorption', 'impact resistance', 'material 
synthesis', 'material processing',
    'additive manufacturing', '3D printing', 'laser sintering', 'electron 
beam melting', 'casting',
    'forging', 'welding', 'machining', 'manufacturing processes', 'design 
optimization',
    'structural optimization', 'topology optimization', 'materials 
selection', 'lattice structures',
    'composite design', 'hybrid materials', 'metallurgy', 'ceramics', 
'polymers', 'biomaterials',
    'materials testing', 'X-ray diffraction', 'scanning electron
microscopy', 'transmission electron microscopy',
    'optical microscopy', 'microstructural analysis', 'fractography',
'spectroscopy', 'mechanical testing',
    'strain gauges', 'digital image correlation', 'displacement
measurement', 'structural health monitoring',
    'damage detection', 'sensor integration', 'machine learning in
mechanics', 'data-driven modeling',
    'artificial intelligence', 'optimization algorithms', 'numerical
methods', 'probabilistic modeling',
    'uncertainty quantification', 'computational efficiency', 'parallel
computing', 'high-performance computing',
    'model validation', 'experimental validation', 'model calibration',
'parameter estimation',
    'simulation-based design', 'virtual prototyping', 'finite element
analysis', 'boundary conditions',
    'initial conditions', 'load cases', 'modal analysis', 'linear
elasticity', 'plasticity',
    'viscoplasticity', 'crystal plasticity', 'homogenization',
'multiphysics simulation',
    'thermomechanical coupling', 'structural dynamics', 'buckling
analysis', 'crack initiation',
    'crack growth', 'damage mechanics', 'fatigue life prediction',
'material degradation', 'wear analysis',
    'corrosion modeling', 'environmental effects', 'thermal fatigue',
'mechanical fatigue', 'high-temperature materials',
    'low-temperature materials', 'cryogenic engineering', 'pressure vessel
analysis', 'pipeline integrity',
    'aerospace materials', 'automotive materials', 'energy materials',
'renewable energy materials',
    'sustainable materials', 'biocompatible materials', 'smart materials',
'shape memory alloys',
    'piezoelectric materials', 'magnetostrictive materials', 'electroactive
polymers', 'nanocomposites',
    'graphene', 'carbon nanotubes', 'metal matrix composites', 'polymer
matrix composites',
    'fiber-reinforced composites', 'matrix degradation', 'interface
mechanics', 'interfacial bonding',
    'mechanical anisotropy', 'elastic modulus', 'shear modulus', 'bulk
modulus', 'Poisson\'s ratio',
    'toughness', 'ductility', 'hardness', 'fracture toughness', 'impact
toughness', 'residual stress',
    'thermal stresses', 'mechanical stresses', 'stress concentration',
'strain energy density',
    'buckling load', 'critical stress', 'dynamic loading', 'static
loading', 'strain localization',
    'stress relaxation', 'creep behavior', 'fatigue crack growth',
'fracture toughness testing'
]
```

- **Purpose:** A comprehensive list of keywords relevant to the applicant's field of study and the specific program.
- **Usage:** Utilized in evaluating customization and fit by measuring semantic similarity between the SOP and these keywords.
- **Advantages:**
  - **Focused Evaluation:** Ensures that the SOP aligns with the program's focus areas.
  - **Semantic Depth:** Captures broader concepts and terminologies specific to the field.

- **Disadvantages:**
    - **Maintenance:** Requires regular updates to stay relevant with evolving program curricula.
    - **Exhaustiveness:** An incomplete or outdated list can undermine evaluation accuracy.

Skills List (Refer how to change the skills list according to your program before you run the jupyter notebook in the 'HOW TO RUN?' section in the README file)

```
skills_list = [
    'MATLAB', 'AutoCAD', 'SolidWorks', 'ANSYS', 'COSMOS Motion', 'Abaqus 
CAE', 'Python',
    'C++', 'Simulink', 'LabVIEW', 'Visual Basic', 'Turbo Pascal', 
'FORTRAN', 'CAD', 'FEA',
    'Computational Fluid Dynamics', 'CFD', 'Finite Volume Method', 
'Boundary Element Method',
    'Mesh Generation', '3D Modeling', 'Design Optimization', 'Structural 
Optimization',
    'Topology Optimization', 'Materials Selection', 'Additive 
Manufacturing', '3D Printing',
    'Laser Sintering', 'Electron Beam Melting', 'Casting', 'Forging', 
'Welding', 'Machining',
    'Thermo-Mechanical Analysis', 'Vibration Analysis', 'Dynamic Analysis', 
'Stress Analysis',
    'Structural Analysis', 'Finite Elements Method', 'Thermodynamics', 
'Heat Transfer',
    'Material Characterization', 'X-ray Diffraction', 'Scanning Electron 
Microscopy',
    'Transmission Electron Microscopy', 'Optical Microscopy', 'Digital 
Image Correlation',
    'Displacement Measurement', 'Strain Measurement', 'Fatigue Testing', 
'Impact Testing',
    'Compression Testing', 'Tensile Testing', 'Shear Testing', 'Torsion 
Testing',
    'Fracture Mechanics Testing', 'Spectroscopy', 'Fractography', 'Strain 
Gauges',
    'Structural Health Monitoring', 'Sensor Integration', 'Machine 
Learning', 'Data Analysis',
    'Artificial Intelligence', 'Numerical Methods', 'Probability Modeling', 
'Uncertainty Quantification',
    'Parallel Computing', 'High-Performance Computing', 'Model 
Calibration', 'Parameter Estimation',
    'Virtual Prototyping', 'Simulation-Based Design', 'Probabilistic 
Modeling', 'Computational Efficiency',
    'Residual Stress Analysis', 'Thermal Stress Analysis', 'Stress 
Concentration Analysis',
    'Strain Energy Density Calculation', 'Buckling Load Determination', 
'Dynamic Loading Simulation',
    'Static Loading Analysis', 'Strain Localization Detection', 'Stress 
Relaxation Studies',
    'Creep Behavior Analysis', 'Fatigue Crack Growth Prediction', 'Fracture 
Toughness Evaluation',
    'Finite Element Analysis', 'Material Degradation Studies', 'Wear 
Analysis', 'Corrosion Modeling',
    'Environmental Effects Assessment', 'Thermal Fatigue Testing', 
'Mechanical Fatigue Testing',
    'Impact Toughness Evaluation', 'Residual Stress Measurement', 'Strain
Rate Analysis',
    'Poisson\'s Ratio Calculation', 'Elastic Modulus Determination', 'Shear
Modulus Calculation',
    'Bulk Modulus Measurement', 'Toughness Testing', 'Ductility Testing',
'Hardness Testing',
    'Fracture Toughness Testing', 'Impact Resistance Testing', 'Material
Synthesis Techniques'
]
```

- **Purpose:** Enumerates relevant skills and software proficiencies pertinent to the field of study.
- **Usage:** Employed in evaluating the presence and relevance of the applicant's skills within the SOP.
- **Advantages:**
  - **Comprehensive Skill Assessment:** Covers a wide range of technical proficiencies.
  - **Quantitative Measure:** Facilitates a numerical assessment of skill relevance.
- **Disadvantages:**
  - **List Dependence:** The evaluator's effectiveness hinges on the completeness of this list.
  - **Context Insensitivity:** Simply matching skills doesn't assess the depth or proficiency.

Personal Attributes

```
attributes_list = [
    'resilience', 'adaptability', 'dedication', 'commitment', 'passion',
'flexibility',
    'productivity under stress', 'problem-solving', 'leadership',
'teamwork', 'critical thinking',
    'initiative', 'self-motivation', 'perseverance', 'creativity',
'analytical skills',
    'time management', 'attention to detail', 'communication skills',
'collaboration',
    'innovation', 'resourcefulness', 'enthusiasm', 'ethics',
'accountability', 'goal-oriented',
    'organizational skills', 'multitasking', 'dependability', 'proactive',
'strategic thinking',
    'empathetic', 'open-mindedness', 'curiosity', 'dedicated', 'focused',
'self-discipline',
    'determination', 'ability to learn', 'intellectual curiosity', 'self-
awareness',
    'conflict resolution', 'motivational', 'positive attitude',
'initiative-taking',
    'versatile', 'persistent', 'detail-oriented', 'responsible', 'dedicated
to excellence'
]
```

- **Purpose:** Lists personal qualities and attributes that are desirable in an applicant.
- **Usage:** Used to assess the presence and alignment of these attributes within the SOP through semantic similarity.
- **Advantages:**
  - **Holistic Evaluation:** Ensures that the applicant showcases desirable personal traits.

    - **Semantic Understanding:** Captures nuanced expressions of attributes beyond explicit mentions.
- **Disadvantages:**
    - **Embedding Limitations:** Similarity scores may not accurately reflect the depth or context of attribute mentions.
    - **List Completeness:** Effectiveness relies on the comprehensiveness of `attributes_list`.

Unique Selling Proposition Phrases

```
usp_phrases = [
    'first among', 'distinction with honor degree', 'STDF granted project',
'published',
    'international conference', 'innovative project', 'award-winning',
'lead research',
    'recognized by', 'top percentile', 'highly commended', 'best in class',
'pioneering work',
    'exceptional performance', 'outstanding achievements', 'state-of-the-
art', 'trailblazing',
    'groundbreaking research', 'unique expertise', 'exclusive experience',
'benchmark study',
    'elite program', 'prestigious institution', 'notable contributions',
'significant impact',
    'transformative research', 'keynote speaker', 'distinguished
researcher', 'leading expert',
    'renowned scholar', 'featured in', 'acclaimed work', 'unparalleled
experience',
    'key contributor', 'highly cited', 'exemplary record', 'exemplary
achievements',
    'commendable efforts', 'leading-edge', 'cutting-edge', 'forward-
thinking', 'strategic initiatives',
    'proven track record', 'exceptional talent', 'noteworthy
accomplishments', 'remarkable success',
    'influential work', 'preeminent researcher', 'exemplary leadership',
'innovative solutions'
]
```

- **Purpose:** Identifies standout qualities and unique achievements of the applicant.
- **Usage:** Assesses the presence and emphasis of unique selling propositions in the SOP through semantic similarity.
- **Advantages:**
    - **Highlighting Uniqueness:** Encourages applicants to showcase distinctive qualities and accomplishments.
    - **Semantic Depth:** Goes beyond keyword matching to understand contextual emphasis.
- **Disadvantages:**
    - **Embedding Sensitivity:** Similarity scores may not directly correlate with actual uniqueness.
    - **List Completeness:** Effectiveness depends on the comprehensiveness of `usp_phrases`.

Clichés List

```
clichés_list = [
    'passion for', 'long-term goals', 'strong background', 'hardworking',
'detail-oriented',
    'self-motivated', 'team player', 'go-getter', 'think outside the box',
'dedicated to',
    'extremely motivated', 'commitment to excellence', 'drive to succeed',
'innovative thinker',
    'strategic vision', 'results-driven', 'excellent communication skills',
    'highly skilled', 'proactive approach', 'exceptional problem-solving',
'dynamic individual',
    'forward-thinking', 'dedicated professional', 'passionate about',
'enthusiastic',
    'self-starter', 'able to multitask', 'highly organized',
'perfectionist', 'quick learner',
    'natural leader', 'strong work ethic', 'effective communicator',
'adaptable', 'resourceful',
    'goal-oriented', 'persistent', 'innovative solutions', 'collaborative
mindset',
    'dedicated team member', 'motivated individual', 'creative problem
solver', 'results-oriented',
    'detail-focused', 'passionate learner', 'keen interest', 'exceptional
dedication',
    'unwavering commitment', 'deeply committed', 'exceptional drive'
]
```

- **Purpose:** Detects overused or generic phrases that may detract from the originality of the SOP.
- **Usage:** Penalizes the SOP by reducing the score based on the number of clichés detected.
- **Advantages:**
  - **Promotes Originality:** Discourages the use of trite and overused expressions.
  - **Quantitative Penalty:** Provides a clear mechanism for penalizing clichés.
- **Disadvantages:**
  - **Contextual Misinterpretation:** Clichés might be used appropriately in context.
  - **False Penalties:** Penalizes phrases that are part of a larger meaningful sentence.

Project Workflow

Understanding the **Project Workflow** is crucial to grasp how the SOP evaluator processes and assesses the input text. This section outlines the step-by-step process from input to output, detailing how the SOP text is transformed and evaluated through various functions.

Step-by-Step Process

1. **Input Acquisition:**
   - **User Input:** The user provides the SOP text that needs to be evaluated.
   - **Keyword Lists:** The evaluator utilizes predefined lists for program keywords, skills, personal attributes, USP phrases, and clichés.

2. **Preprocessing:**
    - **Text Cleaning:** The SOP text is cleaned using the `preprocess_text` function, which removes special characters, digits, converts text to lowercase, and eliminates extra spaces. This standardizes the text for consistent processing.
3. **Embedding Generation:**
    - **BERT Embeddings:** The preprocessed SOP text is passed through the `get_bert_embeddings` function to generate semantic embeddings using the BERT model. These embeddings capture the contextual and semantic nuances of the text.
4. **Evaluation Metrics Execution:**
    - **Individual Assessments:** The evaluator runs multiple functions, each assessing a specific criterion of the SOP. These functions include:
        - **Narrative Clarity:** `evaluate_clear_narrative`
        - **Customization and Fit:** `evaluate_customization_advanced` and `evaluate_fit`
        - **Specificity:** `evaluate_specificity`
        - **Structure:** `evaluate_structure`
        - **Authenticity:** `evaluate_authenticity`
        - **Grammar:** `evaluate_grammar`
        - **Balanced Content:** `evaluate_balanced_content`
        - **Clarity of Goals:** `evaluate_clarity_of_goals`
        - **Skills Relevance:** `evaluate_skills`
        - **Personal Attributes:** `evaluate_personal_attributes_advanced`
        - **Professionalism:** `evaluate_professionalism`
        - **Unique Selling Proposition:** `evaluate_usp`
        - **Formatting Adherence:** `evaluate_formatting`
        - **Cliché Avoidance:** `evaluate_pitfalls`
        - **Consistency:** `evaluate_consistency` (currently a placeholder)
        - **Professional Language:** `evaluate_professional_language`
        - **Positive Tone:** `evaluate_positive_tone`
        - **Readability:** `evaluate_readability_score`
5. **Scoring Aggregation:**
    - **Dictionary Compilation:** Each evaluation function returns a score (typically between 1 and 10) for its respective criterion. These scores are aggregated into a dictionary named `scores`, where each key represents a criterion and its corresponding value is the score.
6. **Visualization:**
    - **Bar Chart Display:** The `visualize_scores` function takes the `scores` dictionary and generates a horizontal bar chart using Matplotlib and Seaborn. This visual representation aids in quickly identifying strengths and areas needing improvement.
7. **Output Generation:**
    - **Console Output:** The evaluator prints out the numerical scores for each evaluation criterion in a readable format.
    - **Visualization Display:** The bar chart is displayed, providing a graphical overview of the SOP's evaluation across different metrics.

Detailed Workflow Example

Let's walk through an example using the provided SOP text to illustrate the workflow:

1. **Input Acquisition:**
   - **SOP Text:** A multi-paragraph SOP is provided as a string.
   - **Keyword Lists:** `program_keywords`, `skills_list`, `attributes_list`, `usp_phrases`, and `clichés_list` are defined with extensive entries relevant to the mechanical engineering field.
2. **Preprocessing:**
   - **Cleaning:** The SOP text undergoes cleaning to remove special characters, digits, and extra spaces, and is converted to lowercase.
3. **Embedding Generation:**
   - **SOP Embedding:** The cleaned SOP text is embedded using BERT to obtain a semantic representation.
   - **Keyword Embeddings:** Aggregated keyword lists are similarly embedded to facilitate semantic similarity calculations.
4. **Evaluation Metrics Execution:**
   - **Narrative Clarity:** Analyzes topic consistency using LDA and cosine similarity between SOP and identified topics.
   - **Customization and Fit:** Measures semantic similarity between the SOP and program-specific keywords.
   - **Specificity:** Counts named entities to proxy the presence of detailed examples.
   - **Structure:** Checks for the presence of introductory and concluding phrases.
   - **Authenticity:** Uses sentiment subjectivity to gauge the personal touch.
   - **Grammar:** Detects grammatical errors using LanguageTool and calculates an error rate.
   - **Balanced Content:** Evaluates the balance between personal anecdotes and technical content based on pronoun and keyword counts.
   - **Clarity of Goals:** Detects phrases indicative of short-term and long-term goals.
   - **Skills Relevance:** Measures the alignment of the SOP with a predefined skills list using semantic similarity.
   - **Personal Attributes:** Assesses the presence of personal attributes through semantic similarity.
   - **Professionalism:** Counts professionalism-related keywords to assign a score.
   - **Unique Selling Proposition:** Evaluates the uniqueness of the SOP by comparing it with USP phrases.
   - **Formatting Adherence:** Checks for a minimum number of paragraphs to assess formatting.
   - **Cliché Avoidance:** Detects and penalizes the presence of clichéd phrases.
   - **Consistency:** Currently a placeholder, returns a neutral score.
   - **Professional Language:** Analyzes word complexity to assess professional language use.
   - **Positive Tone:** Measures the positivity of the SOP using sentiment analysis.
   - **Readability:** Assesses readability using the Flesch Reading Ease score mapped to a 1-10 scale.
5. **Scoring Aggregation:**
   - **Dictionary Compilation:** All individual scores are compiled into the `scores` dictionary.
6. **Visualization and Output:**

- **Bar Chart:** A horizontal bar chart is generated to visualize the scores across all criteria.
- **Console Output:** Scores are printed in a structured format, e.g.,

```
Advanced SOP Evaluation Scores:
Clear and Focused Narrative: 8.5/10
Customization to the Specific Program or Institution: 9.0/10
Specificity and Detail with Concrete Examples: 7.5/10
Strong Structure and Organization: 10/10
Authenticity and Personality: 8.0/10
Demonstrated Fit with the Program/Institution: 9.0/10
Error-Free Writing (Grammar, Spelling, Punctuation): 7.5/10
Balanced Content (Personal Anecdotes vs. Academic/Professional
Achievements): 8.0/10
Clarity of Goals (Short-term and Long-term): 10/10
Relevant Skills and Qualifications: 9.0/10
Personal Attributes (e.g., Resilience, Adaptability): 8.0/10
Professionalism and Preparedness: 10/10
Unique Selling Proposition (USP): 9.0/10
Adherence to Formatting Guidelines: 10/10
Avoidance of Common Pitfalls (e.g., Clichés, Irrelevant
Information): 8.0/10
Consistency with Other Application Materials: 5/10
Use of Professional Language: 9.0/10
Positive Tone and Constructive Framing: 8.0/10
Readability: 7/10
```

(Numbers are dummy ones shown to illustrate how the output would be, they are not to be taken as literal outputs).

Advantages and Disadvantages of our Rule-Based Model

Advantages

1. **Transparency and Interpretability:**
   - **Rule-Based Nature:** Clear rules and criteria make the evaluation process transparent and easily interpretable.
   - **Explicit Scoring Mechanism:** Users can understand how scores are derived based on specific criteria.
2. **Control Over Evaluation Criteria:**
   - **Customizable Metrics:** Easily add, modify, or remove evaluation criteria based on specific needs.
   - **Fine-Grained Assessment:** Provides detailed scores across multiple facets of the SOP.
3. **No Need for Training Data:**
   - **Immediate Deployment:** Can be used out-of-the-box without requiring large labeled datasets.
   - **Cost-Effective:** Eliminates the need for data annotation and model training expenses.
4. **Scalability for Specific Tasks:**

- **Targeted Evaluation:** Tailored to assess specific aspects relevant to SOPs, ensuring focused assessments.

Disadvantages

1. **Limited Flexibility and Adaptability:**
   - **Static Rules:** Cannot adapt to new patterns or linguistic variations without manual updates.
   - **Contextual Insensitivity:** May miss the broader context or nuanced expressions within the SOP.
2. **Maintenance Overhead:**
   - **Manual Updates Required:** Needs continuous refinement of rules and keyword lists to maintain accuracy and relevance.
   - **Scalability Issues:** As evaluation criteria expand, managing and updating rules becomes increasingly complex.
3. **Potential for Over-Simplification:**
   - **Binary Decisions:** Presence or absence of keywords may not fully capture the quality or depth of the content.
   - **Quantitative Limits:** Simplistic scoring mechanisms may not reflect the qualitative nuances of the SOP.
4. **Vulnerability to Gaming:**
   - **Keyword Stuffing:** Applicants might artificially inflate scores by overusing targeted keywords without genuine context or relevance.
5. **Incomplete Evaluation:**
   - **Placeholder Functions:** Some evaluation aspects, like consistency with other materials, are not fully implemented, limiting the comprehensiveness of the assessment.

Experiments and Results:

While we've defined the rules after looking at a 1000 or more SOPs, we've tested our ruled based model on around 100 SOPs manually, and the results were **moderately accurate**.

Since this is a rule-based model, it worked and fell short precisely in areas we've defined above in the model's advantages & disadvantages and its comparison with a 'learning' based model.

Initial Work, Research & Hit-and-miss attempts:

1. Initially, we tried to build a classification model for our SOPs (the classifier to classify to which masters' program the SOP belongs to), but didn't see success despite a lot of effort, the code is there in the **'Hit & Miss (Various initial attempts)' folder on GitHub.** Here is an explanation for the error:

With this classification model we faced a performance limitation due to the skill overlap across academic disciplines. Especially, the presence of intersecting course skills creates the issue, compromising model performance. We trained our model with 1400 SOPs that we scrapped from web and synthetically generating few.

Take for example, the programming skill of C++, which is prevalent across multiple academic disciplines such as Computer Science (CS) and Electronics and Computer Engineering (ECM). This skill commonality introduces classification complexity.

So, our model incorrectly assigns multiple classifications to a single statement of purpose (SOP). This kind of classification lacks the requirement for downstream workflow.

We could've tried to trim the dataset such that course skill that fall in same line are removed from the dataset, but it would create an issue in general classification of SOP because of insufficient data.

2. We also tried working on the Default Project mentioned in the project guidelines pdf on OneDrive. We attempted building a 'Question & Answer Model using Multi-Head Attention Transformer' using the SQuAD 2.0 dataset. We were getting incorrect outputs most of the time and correct outputs sometimes. The code is there in the **'Hit & Miss (Various initial attempts)' folder on GitHub.** Here is an explanation for the error:

Observed Issue:
During testing, the model produces overly simplistic and contextually irrelevant one-word responses like "is" or "an." This indicates a failure to comprehend the input passage and question relationship, resulting in poor inference performance.

Training Details:
● The training process was carried out for two epochs:
○ **Epoch 1:**
Training Loss: 10.2399
Validation Loss: 10.8357 (Model saved as validation loss improved)
○ **Epoch 2:**
Training Loss: 10.9672
Validation Loss: 10.9326 (No improvement observed)

● The loss values remain consistently high, suggesting underfitting and inadequate learning.

Root Causes:
1. **Model Limitations:**
The model's architecture may lack sufficient complexity to effectively capture the relationships between the passage and Question.

2. **Dataset and Training Dynamics:**
○ Potentially insufficient or poorly formatted training data, leading to limited generalization capability.
○ Two training epochs may not be sufficient for the model to Converge.

3. **Preprocessing and Tokenization:**
○ Inadequate tokenization or representation of inputs may cause the model to misinterpret context. It has limitations.

4. **Training Parameters:**
Suboptimal hyperparameters, such as learning rate or batch size, could be hindering model performance.

5. **CPU insufficiency:**
Due to cpu insufficiency and the bulk data, the whole data is not getting used in training the model and output is not as we expected.

6. **Overly Simplistic Decoding Strategy:**
Our chosen start and end tokens might not form a coherent span. They could even be the same token or unrelated tokens from different parts of the sequence. This often leads to truncated or meaningless answers.

Proposed Solutions
1. Data Enhancements:
● Use a high-quality, diverse dataset containing well-labeled passage-question-answer triples. Consider public datasets like SQuAD for additional training examples.
● Preprocess data effectively by ensuring proper passage-question concatenation and the inclusion of special tokens (e.g., [CLS],[SEP]) when using transformer models.

2. Model Improvements:
● Transition to transformer-based models such as BERT, T5, or GPT, which are pre-trained on large text corpora and designed for tasks like question answering.
● Fine-tune these models on your dataset to improve contextual understanding and generalization.

3. Training Strategies:
● Increase the number of epochs to allow sufficient learning. Use early stopping to prevent overfitting.
● Optimize hyperparameters, such as learning rate, using grid search or Bayesian optimization.
● Experiment with sequence-based loss functions, which better align with the QA task.

4. Evaluation and Debugging:

● Analyze validation predictions to pinpoint patterns in failures (e.g., repeated single-word answers).
● Employ metrics like Exact Match (EM) and F1 score to quantitatively evaluate performance.

5. Pre-trained Models as Baselines:
Leverage pre-trained models fine-tuned on similar tasks as a baseline for your project. This will provide a strong starting point and enable quicker convergence.

Next Steps:
1. Debug and verify the preprocessing pipeline to ensure tokenization and input formatting are correct.
2. Implement and train a transformer-based model, evaluating results on the validation dataset.
3. Iteratively refine the model by tweaking architecture and training parameters based on observed performance. By addressing these issues and following the proposed solutions, we aim to significantly improve the model's ability to generate meaningful answers from the given passage and question.