

Essay Grading

Feladat: 23_05d Automated Essay Scorer:

Feladat leírása:

- „Design a system that scores essays based on factors like grammar, coherence, and vocabulary.”
- „Use Natural Language Processing techniques and get assistance with libraries like NLTK or spaCy.”
- A promptokat angolul adtuk meg és minden program komment és a feladat további részei közöttük a megoldás is angol nyelven van.

Promptok és magyarázat:

#1

Design a system in Python that grades an essay based on: grammar, coherence, and vocabulary. At the end, print the result grade.

Ez megalapozta a program struktúráját, ami valahogy a kezdetben így nézett ki.

Felbontotta különböző funkciókra a különböző szempontokat, amik alapján az esszét osztályozzuk.

Majd az esszét egy sztringben tárolja, amit paraméterként ad meg minden egyes függvénynek.

Ezek a függvények pedig visszatérnek egy értékkel, ami itt még nem volt korlátok közé zárva.

```
python Copy code

import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.probability import FreqDist
from nltk.tag import pos_tag
from nltk.metrics import jaccard_distance

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')

def check_grammar(text):
    # Add your grammar checking logic here
    # Example: You might use a library like LanguageTool or GingerIt

def calculate_coherence(text):
    # Add your coherence checking logic here
    # Example: You might analyze sentence structure, coherence markers, etc.

def calculate_vocabulary(text):
    # Tokenize the text
    words = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [word.lower() for word in words if word.isalnum() and word.lower() not in stop_words]

    # Calculate vocabulary richness using Jaccard distance
    reference_text = "This is a sample reference text for vocabulary analysis."
    reference_words = word_tokenize(reference_text.lower())

    jaccard_dist = jaccard_distance(set(words), set(reference_words))
    vocabulary_score = 1 - jaccard_dist

    return vocabulary_score

def grade_essay(text):
    # Check grammar
    grammar_score = check_grammar(text)

    # Calculate coherence
    coherence_score = calculate_coherence(text)

    # Calculate vocabulary
    vocabulary_score = calculate_vocabulary(text)

    # Calculate overall grade (you can adjust the weights based on importance)
    overall_grade = (0.4 * grammar_score) + (0.3 * coherence_score) + (0.3 * vocabulary_score)

    return overall_grade

def main():
    essay_text = input("Enter the essay text: ")

    grade = grade_essay(essay_text)

    print("Grammar Score:", check_grammar(essay_text))
    print("Coherence Score:", calculate_coherence(essay_text))
    print("Vocabulary Score:", calculate_vocabulary(essay_text))
    print("Overall Grade:", grade)
```

Vocabulary (egyedi szavak aránya)

#2

What is a good and easy way to check the vocabulary of a text, and grade it. You might check sentence and paragraph transitions. You might check for advanced words and phrases.

Ez már használ tokenizációt, amely most még nem teljes, de később ez a módszer lesz célra vezető.

Az nltk-t kezdte el használni, mert ezt találta a megfelelő könyvtárnak. Így ezt külön nem kellett megadni a későbbiekben, mint feltétel.

ChatGPT-t idézve: „While there is no one-size-fits-all solution, you can use the Natural Language Toolkit (nltk) in Python to perform some basic analysis.”

python

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.probability import FreqDist
from nltk.tag import pos_tag

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
```

```
def calculate_vocabulary_richness(text):
    # Tokenize the text
    words = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [word.lower() for word in words if word.isalnum() and word.lower() not in stop_words]

    # Calculate vocabulary richness using FreqDist
    vocabulary_richness = len(set(words)) / len(words) if len(words) > 0 else 0

    return vocabulary_richness

def evaluate_sentence_transitions(text):
    # Tokenize the text into sentences
    sentences = sent_tokenize(text)

    # Perform part-of-speech tagging for each sentence
    pos_tags = [pos_tag(word_tokenize(sentence)) for sentence in sentences]

    # Check for sentence transitions (e.g., using conjunctions)
    transition_score = sum(1 for pos_tagged_sentence in pos_tags if any(tag[1] in ['CC', 'JJR', 'JJS', 'RB', 'RBR', 'RBS'] for tag in pos_tagged_sentence))

    return transition_score

def evaluate_advanced_words(text):
    # Tokenize the text
    words = word_tokenize(text)

    # Perform part-of-speech tagging
    pos_tags = pos_tag(words)

    # Check for advanced words (e.g., adjectives, adverbs)
    advanced_word_score = sum(1 for tag in pos_tags if tag[1] in ['JJ', 'JJR', 'JJS', 'RB', 'RBR', 'RBS'])

    return advanced_word_score
```

#3

Change this last code, so that the vocabulary score should be counted as how many unique words are in the essay.

```
def calculate_vocabulary_score(text):
    # Tokenize the text
    words = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [word.lower() for word in words if word.isalnum() and word.lower() not in stop_words]

    # Calculate vocabulary score based on the count of unique words
    unique_words = set(words)
    vocabulary_score = len(unique_words)

    return vocabulary_score
```

Itt az volt a gond, hogy a score amit visszaad az az egyedi szavak száma, és az értékelés csak azt mondta meg, hogy hány értékes szó volt. Tehát minél több szó volt (egyedi) annál jobb volt az esszé. Ezért ezt a következő prompt-tal próbáltam 1 és 5-ös osztályzat közé szorítani.

(itt még a többi használatlan funkció jelen volt)

```
def evaluate_sentence_transitions(text):
    # Tokenize the text into sentences
    sentences = sent_tokenize(text)

    # You can add your logic to evaluate sentence transitions if needed

def evaluate_advanced_words(text):
    # You can add your logic to evaluate advanced words if needed

def grade_essay(text):
    # Calculate vocabulary score
    vocabulary_score = calculate_vocabulary_score(text)

    # Evaluate sentence transitions
    # transition_score = evaluate_sentence_transitions(text)

    # Evaluate advanced words
    # advanced_word_score = evaluate_advanced_words(text)

    # Calculate overall grade (you can adjust the weights based on importance)
    overall_grade = vocabulary_score # Adjust the weight as needed

    return overall_grade

def main():
    essay_text = input("Enter the essay text: ")
```

#4

it should give back a number between 1 and 5 as a grade, based on this unique-ness

```
def map_to_grade(score, max_score, min_grade=1, max_grade=5):
    # Normalize the score to be in the range [0, 1]
    normalized_score = min(1, max(0, score / max_score))

    # Map the normalized score to the grade range
    grade = min_grade + normalized_score * (max_grade - min_grade)

    return round(grade, 1)

def grade_essay(text):
    # Calculate vocabulary score
    vocabulary_score = calculate_vocabulary_score(text)

    # Map the vocabulary score to a grade between 1 and 5
    overall_grade = map_to_grade(vocabulary_score, max_score=100) # Adjust max_score

    return overall_grade
```

Ezen részen olyan probléma lépett fel, hogy egy konstanstól függött az eredmény, és nem az esszé hosszától. Ezt pedig megint ki lehetett használni, és csak hosszabb esszét kellett megadni, mint az a konstans, és jobb eredményt lehetett elérni.

(itt már nincsenek benne azok a függvények, amelyeket nem használunk)

```
def calculate_vocabulary_score(text):
    # Tokenize the text
    words = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [word.lower() for word in words if word.isalnum() and word.lower() not in stop_words]

    # Calculate vocabulary score based on the count of unique words
    unique_words = set(words)
    vocabulary_score = len(unique_words)

    return vocabulary_score

def map_to_grade(score, max_score, min_grade=1, max_grade=5):
    # Normalize the score to be in the range [0, 1]
    normalized_score = min(1, max(0, score / max_score))

    # Map the normalized score to the grade range
    grade = min_grade + normalized_score * (max_grade - min_grade)

    return round(grade, 1)

def grade_essay(text):
    # Calculate vocabulary score
    vocabulary_score = calculate_vocabulary_score(text)

    # Map the vocabulary score to a grade between 1 and 5
    overall_grade = map_to_grade(vocabulary_score, max_score=100) # Adjust max_score

    return overall_grade

def main():
    essay_text = input("Enter the essay text: ")

    grade = grade_essay(essay_text)
```

#5

It should count in the length of the essay, and not just put a fix number in the calculation.

Ebben az állapotban már az esszé hossza és az egyedi szavak aránya adta meg a jegy értékét a következő módon:

grade =

$$\text{min_grade} + ((\text{max_grade} - \text{min_grade}) / \text{total_word_count}) * \text{unique_word_count}$$

Azt részt amely megvizsgálja, hogy nulla e a szavak száma a teljes esszében, azt nem prompt-tal lett hozzáadva, hanem kézzel lett beleírva.

Itt az van kiszűrve, hogy ha 0 a szavak száma akkor 1-es jegyet adjunk az esszére, mert ezt nem lehet értékelni.

Különben pedig a képletet használva határozza meg.

Majd a return-ben generált formában a minimum és maximum osztályzatok közé kényszeríti a jegyet, így más érték nem lehet.

```
def calculate_ttr(essay):
    """
    Calculate the type-token ratio (TTR) of the given essay.

    Parameters:
    - essay (str): The input essay.

    Returns:
    - int: The number of unique words.
    """
    words = nltk.word_tokenize(essay.lower())
    unique_words = set(words)
    ttr = len(unique_words)
    return ttr

def grade_vocabulary(unique_word_count, total_word_count):
    """
    Calculate the vocabulary grade based on the TTR.

    Parameters:
    - unique_word_count (int): The number of unique words.
    - total_word_count (int): The total number of words.

    Returns:
    - int: The vocabulary grade.
    """
    min_grade = 1
    max_grade = 5

    if total_word_count != 0:
        grade = min_grade + ((max_grade - min_grade) / total_word_count) * unique_word_count
    else:
        grade = 1

    rounded_grade = round(grade)
    return min(max_grade, max(min_grade, rounded_grade))

sample_essay = """
How to write a great essay? This is the way to create a perfect one. In an paper it is
Some xd read b good novel? That are them ways tot created as perfect ones. Ind and pa
"""

unique_word_count = calculate_ttr(sample_essay)
total_word_count = len(nltk.word_tokenize(sample_essay.lower()))

vocabulary_grade = grade_vocabulary(unique_word_count, total_word_count)
```

#6

how not to count the (.?!) and other end or separator symbols

```
def tokenize_and_filter_punctuation(text):
    tokens = nltk.wordpunct_tokenize(text.lower())
    # Filter out punctuation marks
    tokens = [token for token in tokens if token not in string.punctuation]
    return tokens
```

Erre a megoldásra jutott a prompt alapján, mely egy ideig elegendőnek is tűnt, de később ezt még szigorítani kell.

(azt is megkérdeztük mi található ebben a string.punctuation-ben, hogy megértsük a működését)

```
!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~
```

Arra is meg lett kérve egy promptal, hogy kerekítse a jegyet, mert az eredmény megfelelt annak, hogy 1 és 5 között volt, de a nem egész számok a képlet miatt akár 6-8 tizedes jegy hosszúak is lehettek.

```
# Round the grade to the nearest whole number
rounded_grade = round(grade)
```

Aztán hosszasan a szöveg repetitívására koncentráltunk, mely később nem konzisztens és nem jó irányba vezetett. Ezért ezt a részt nem tartalmazza a jegyzőkönyv. Azonban ebben a részben egy-két jó ötlet is felmerült, mint a létigék, kötőszavak és hasonló fajtájú szavak szűrése. Ezek a szavak szükségesek szinte minden mondatba, ezért ezeket nem kellene számításba venni az egyedi szavak vizsgálatakor. (például főnévi igenév nélkül kevés mondatot lehetne az esszébe írni, ezek gyakran ismétlődnek)

Az első próbálkozás az volt, hogy a ChatGPT-nek megadva ezeket a szófajtákat adja hozzá egy listához, amelyeket nem veszünk figyelembe.

Ahogy képen is látszik ez a lista kicsit kinőtte magát, így más megoldást kellett keresni.

```
def filter_common_words(words):
    # Define a list of common pronouns, auxiliary verbs, conjunctions, articles, and
    common_words = ['i', 'me', 'my', 'mine', 'you', 'your', 'yours', 'he', 'him', 'his',
                    'we', 'us', 'our', 'ours', 'they', 'them', 'their', 'theirs',
                    'is', 'am', 'are', 'was', 'were', 'be', 'been', 'being',
                    'have', 'has', 'had', 'do', 'does', 'did',
                    'isn't', 'aren't', 'wasn't', 'weren't', 'haven't', 'hasn't', 'hadn't',
                    'to', 'and', 'in', 'on', 'for', 'with', 'at', 'by', 'about', 'of',
                    'but', 'or', 'nor', 'yet', 'so',
                    'a', 'an', 'the', # Articles
                    'could', 'can', 'should', 'shall', 'would', 'will', 'might', 'may',
                    'this', 'that', 'these', 'those', # Demonstratives
                    'here', 'there', 'where', # Location words
                    'now', 'then', 'when', 'while', # Time-related words
                    'more', 'less', 'most', 'least', # Comparison words
                    'some', 'any', 'many', 'few', # Quantity words
                    'all', 'every', 'each', 'none', # Universals
                    'which', 'who', 'whom', 'whose', # Relative pronouns
                    'what', 'whatever', 'which', 'whichever', # Interrogative pronouns
                    ]

    # Filter out common words
    filtered_words = [word for word in words if word not in common_words]

    return filtered_words
```

#7

is there a better and shorter way to filter these words rather than listing all of them in a list



ChatGPT

Yes, a more concise way to filter out common words is to use NLTK's built-in stop words list. NLTK provides a list of common words that are often excluded in text processing tasks because they are considered to be of little value in identifying content. You can use this list to filter out common words from your text. Here's an updated example:

```
python Copy code

import nltk
from nltk.corpus import stopwords
import string

def tokenize_and_filter_punctuation(text):
    tokens = nltk.word_tokenize(text.lower())
    # Filter out punctuation marks
    tokens = [token for token in tokens if token != "" or token not in string.punctuation]
    return tokens

def filter_common_words(words):
    # Use NLTK's English stop words list
    stop_words = set(stopwords.words('english'))

    # Filter out common words
    filtered_words = [word for word in words if word not in stop_words]

    return filtered_words
```

(itt is megkérdezve azt hogy mit tartalmaz a stopwords, hogy biztosan az elvárt eredményt adja majd)

Here's a snippet of what `stopwords.words('english')` contains:

```
plaintext Copy code

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
```

További módosítások után a végleges szűrés a következő képpen néz ki.

```
python Copy code

import nltk
from nltk.corpus import stopwords

def tokenize_and_filter_punctuation(text):
    tokens = nltk.word_tokenize(text.lower())

    # Filter out single characters that are punctuation marks or other unwanted symbols
    tokens = [token for token in tokens if len(token) > 1 or token.isalnum() or token in ["'", '"', "'"]]

    # Remove possessive 's from words
    tokens = [token[:-2] if token.endswith("'s") else token for token in tokens]

    # Remove double quotation marks
    tokens = [token.replace('"', '') for token in tokens]

    return tokens

def filter_common_words(words):
    # Use NLTK's English stop words list
    stop_words = set(stopwords.words('english'))

    # Filter out common words
    filtered_words = [word for word in words if word not in stop_words]

    return filtered_words
```

Ez megoldotta az olyan problémás szavakat mint a „Bob's” , ahol eddig külön szónak számított a 's.

Az értékelés kiszámítása nem változott, de a jegy adása igen. Tartományokra lett osztva 0 és 1 között, ahol attól függően melyik részbe esik ez az érték (egyedi szavak aránya az összes lehetséges egyedi szóhoz).

```
if max_unique_word_count == 0:
    unique_word_count_grade = 1
else:
    unique_word_count_ratio = len(unique_words) / max_unique_word_count

    # Ensure the grade is within the specified range
    if unique_word_count_ratio >= 0.85:
        unique_word_count_grade = 5
    elif 0.85 > unique_word_count_ratio >= 0.70:
        unique_word_count_grade = 4
    elif 0.70 > unique_word_count_ratio >= 0.55:
        unique_word_count_grade = 3
    elif 0.55 > unique_word_count_ratio >= 0.30:
        unique_word_count_grade = 2
    else:
        unique_word_count_grade = 1
```

Ennek értelmezése, hogy például ha az egyedi szavak 30%-át adják csak az összes szónak ami egyedi lehet, az csak 2-est eredményez.

Arra is megkértük a ChatGPT-t hogy adjon visszajelzést az esszéről, hogy miben lehetne fejlődni, mert amikor a végleges jegyet kapja a sok szempont alapján, akkor már nehéz lenne megmondani. Így minden funkció külön jelez hogy ha nem lett 5-ös akkor azt jelenti itt ebben a tulajdonságban lehet még javítani jobb jegy érdekében.

```
# Provide feedback to the writer
if unique_word_count_grade < 5:
    print("*** The essay could be improved with a richer vocabulary. ***")
```

A következő oldalon egyben beillesztve látható lesz a vocabulary alapján való értékelés teljes függvénye.

```
def calculate_unique_word_count(text):
    # Tokenize the text, convert to lowercase, filter punctuation
    words = tokenize_and_filter_punctuation(text)

    # Filter out common words
    filtered_words = filter_common_words(words)

    # Remove empty strings from filtered words
    filtered_words = [word for word in filtered_words if word != '']

    unique_words = set(filtered_words)

    # Calculate unique word count grade (scaled to be between 1 and 5)
    max_unique_word_count = len(filtered_words)

    if max_unique_word_count == 0:
        unique_word_count_grade = 1
    else:
        unique_word_count_ratio = len(unique_words) / max_unique_word_count
        # Ensure the grade is within the specified range
        if unique_word_count_ratio >= 0.75:
            unique_word_count_grade = 5
        elif 0.75 > unique_word_count_ratio >= 0.60:
            unique_word_count_grade = 4
        elif 0.60 > unique_word_count_ratio >= 0.45:
            unique_word_count_grade = 3
        elif 0.45 > unique_word_count_ratio >= 0.30:
            unique_word_count_grade = 2
        else:
            unique_word_count_grade = 1

    # Provide feedback to the writer
    if unique_word_count_grade < 5:
        print("*** The essay could be improved with a richer vocabulary. ***")

    return unique_word_count_grade
```

Egy jó és egy rossz esszét is generált a ChatGPT-t és azokkal ellenőrizve a következő eredményt kaptuk.

```
# Example usage
good_essay = """
Language is a powerful tool, and within its intricate web of words lies the key to effective communication. The richness of vocabulary not only embellishes our expression but also serves as a testament to the depth of human intellect and understanding.

A diverse vocabulary allows individuals to articulate their thoughts with nuance and precision. It is akin to having a palette of colors at one's disposal, enabling the creation of vivid and nuanced linguistic landscapes. Instead of settling for mundane descriptions, one can paint a picture with words, evoking emotions and painting a clear picture in the reader's mind.

Furthermore, a robust vocabulary is a gateway to intellectual exploration. It opens the doors to literature, philosophy, and various realms of knowledge. Words serve as vessels for ideas, and a rich vocabulary allows individuals to navigate the vast ocean of human thought and knowledge with ease.

In the professional sphere, a strong vocabulary is an asset that commands respect. Whether crafting a persuasive argument, composing a compelling presentation, or engaging in meaningful discourse, individuals with an extensive vocabulary possess the ability to captivate and persuade.

Moreover, a diverse vocabulary contributes to cognitive flexibility. The ability to choose the most fitting word in a given context reflects a nimble mind capable of adapting to various communication scenarios. It is this adaptability that allows individuals to express their thoughts with clarity and precision.

In conclusion, the beauty of an expansive vocabulary lies in its capacity to elevate language from a mere means of communication to an art form. It empowers individuals to convey thoughts with precision, explore the realms of knowledge, command respect in professional settings, and engage in meaningful discourse.

"""

bad_essay = """
So, like, words are a thing, and they matter, I guess. You know, they help us say stuff, and that's important. When you use different words, it can make things sound better, I think.

Having a good vocabulary is cool because, you know, it lets you express things in a way that's not boring. It's like having a bunch of colors to paint with, and that's nice. When you use different words, it's not the same as using the same words all the time.

Words are useful for, like, learning things. They help you understand stuff, I think. You can find out about stuff from books and things, and that's cool. Words help with that. So, it's like, having more words is better because then you can learn more things.

In jobs and stuff, having a good vocabulary is good, too. When you talk or write, people listen more if you use different words. It makes you sound smart, maybe. So, like, it's important to have good words for work and things.

Also, using different words is kind of flexible. You can say things in different ways, and that's useful. It's not boring when you have different words to use. So, yeah, words are kind of important, and having more of them is good.

In conclusion, words matter because they help us say things and learn stuff. Having different words is cool because it makes things not boring. It's good for jobs and stuff, and using different words is flexible. So, yeah, words are kind of important, I guess."""
```

Bad essay vocab grade: 2
Good essay vocab grade: 5

Ezek alapján neki is kezdtünk a következő szempontok kidolgozásának.

Coherence (koherencia)

A következő szempont a coherence (koherencia) volt. Kezdsnek egy új ChatGPT beszélgetést nyitottunk. Kezdsnek pedig be lett másolva az eddigi kód.

#7

similar to these functions, create a coherence checking function that is as simple it can get and work

```
python Copy code

import nltk
from nltk.corpus import stopwords
from nltk import pos_tag
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

def is_related(word1, word2):
    synsets1 = wordnet.synsets(word1)
    synsets2 = wordnet.synsets(word2)

    for synset1 in synsets1:
        for synset2 in synsets2:
            if synset1.wup_similarity(synset2) is not None and synset1.wup_similarity(synset2) > 0:
                return True
    return False
```

```
def check_coherence(text):
    tokens = nltk.word_tokenize(text.lower())

    # Remove stopwords and single characters
    tokens = [token for token in tokens if len(token) > 1 and token not in stopwords]

    # Lemmatize the tokens
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

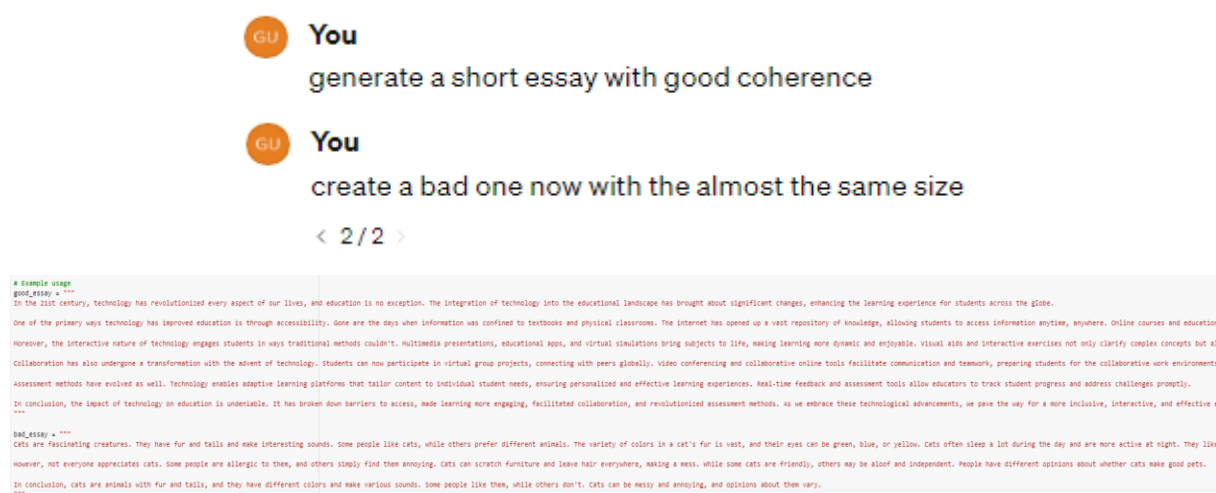
    # Parts of speech tagging
    pos_tags = pos_tag(tokens)

    # Check coherence based on the proximity of related words
    coherence_score = 0
    for i in range(len(pos_tags) - 1):
        word1, pos1 = pos_tags[i]
        word2, pos2 = pos_tags[i + 1]

        # Check if the words are related and have the same part of speech
        if is_related(word1, word2) and pos1 == pos2:
            coherence_score += 1

    return coherence_score
```

Ezután generáltattunk egy jó és egy rossz esszé példát.



Ezeket lefuttatva két számot kaptunk score eredményként, amelyet debuggolás során kerestünk ki. Majd ezeket az értékeket alapul véve generáltattunk értéktartományokat az előzőhöz hasonló képpen ez alapján adjuk a jegyet az esszére e szempont alapján.

```
def assign_grade(score):  
    print(score)  
    if score >= 16:  
        return 5  
    elif 12 <= score < 16:  
        return 4  
    elif 8 <= score < 12:  
        return 3  
    elif 4 <= score < 8:  
        return 2  
    else:  
        return 1
```

Majd ezeket az esszéket (jó és rossz) osztályoztuk koherencia alapján, és az eredményeket megfelelőnek találtuk, más generált esszével is.

```
good_essay_grade = assign_grade(check_coherence(good_essay))  
bad_essay_grade = assign_grade(check_coherence(bad_essay))  
  
print(f"Good essay coherence grade: {good_essay_grade}")  
print(f"Bad essay coherence grade: {bad_essay_grade}")
```

```
} Good essay coherence grade: 5  
Bad essay coherence grade: 2
```

A következő oldalon egyben beszúrva megtalálható a koherencia ellenőrzését adó függvényt.

```

import nltk
from nltk.corpus import stopwords
from nltk import pos_tag
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

def is_related(word1, word2):
    synsets1 = wordnet.synsets(word1)
    synsets2 = wordnet.synsets(word2)

    for synset1 in synsets1:
        for synset2 in synsets2:
            if synset1.wup_similarity(synset2) is not None and synset1.wup_similarity(synset2) > 0.6:
                return True
    return False

def check_coherence(text):
    tokens = nltk.word_tokenize(text.lower())

    # Remove stopwords and single characters
    tokens = [token for token in tokens if len(token) > 1 and token not in stopwords.words('english')]

    # Lemmatize the tokens
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Parts of speech tagging
    pos_tags = pos_tag(tokens)

    # Check coherence based on the proximity of related words
    coherence_score = 0
    for i in range(len(pos_tags) - 1):
        word1, pos1 = pos_tags[i]
        word2, pos2 = pos_tags[i + 1]

        # Check if the words are related and have the same part of speech
        if is_related(word1, word2) and pos1 == pos2:
            coherence_score += 1

    return coherence_score

```

```

def assign_grade(score):
    if score >= 16:
        return 5
    elif 12 <= score < 16:
        return 4
    elif 8 <= score < 12:
        return 3
    elif 4 <= score < 8:
        return 2
    else:
        return 1

```

Grammar (nyelvhelyesség)

Az utolsó szempont a grammar (nyelvhelyesség) volt. Ehhez is egy új ChatGPT beszélgetést nyitottunk. Kezdsnek pedig be lett másolva az eddigi kód.

#8

Similar to these functions and grading system, how to check the grammar of an essay, and grade it

1. Install LanguageTool:

Install the `LanguageTool` library, a powerful and widely-used tool for grammar and style checking in various languages. You can install it using:

```
bash
pip install language-tool-python
```

2. Grammar Checking Function:

Create a function that takes the essay text as input, checks grammar using `LanguageTool`, and returns a list of grammar issues found.

```
python
import language_tool_python

def check_grammar(text):
    tool = language_tool_python.LanguageTool('en-US')
    matches = tool.check(text)
    return matches
```

Create a function that assigns a grade based on the number of grammar issues found. You can customize the grading system based on your criteria.

```
python
def assign_grammar_grade(grammar_issues):
    total_issues = len(grammar_issues)

    if total_issues == 0:
        return 5 # Perfect grammar
    elif total_issues <= 5:
        return 4 # Minor issues
    elif total_issues <= 10:
        return 3 # Moderate issues
    elif total_issues <= 15:
        return 2 # Substantial issues
    else:
        return 1 # Numerous issues
```

4. Example Usage:

Check the grammar of your essays and assign grades.

```
python
good_essay = """Your well-written essay here."""
bad_essay = """An essay with grammar mistakes."""

grammar_issues_good = check_grammar(good_essay)
grammar_grade_good = assign_grammar_grade(grammar_issues_good)
print(f"Good essay grammar grade: {grammar_grade_good}")

grammar_issues_bad = check_grammar(bad_essay)
grammar_grade_bad = assign_grammar_grade(grammar_issues_bad)
print(f"Bad essay grammar grade: {grammar_grade_bad}")
```

5. Feedback to Writer:

Optionally, you can provide feedback to the writer based on the grammar grade, suggesting areas of improvement.

Ezekhez generált példák jó eredményt adtak, de a rossz példa nem volt elég rossz osztályzatú, így a tartományok át lettek alakítva. Az újabb rossz példa már így rosszabb értékelést kapott.

```
good_essay = """
In today's fast-paced world, people face numerous challenges. There is pressure from work, family, and society, making life very stressful.

Sometimes, individuals feel overwhelmed by the demands of daily life. There is a constant need to balance work and personal life, leading to a lot of stress. People often find it challenging to allocate time for relaxation.

Another problem is the lack of communication between people. Many individuals are always busy with their own activities, and they don't take the time to engage in meaningful conversations. This can lead to misunderstandings.

The education system also has its challenges. Students often feel stressed because they have to learn a variety of subjects. Teachers don't always have enough time to help each student individually.

In conclusion, life is filled with challenges, and people face many problems. The stress from work, lack of communication, and issues in the education system are just some of the problems people are dealing with. It's
"""
bad_essay = """
In the today fast-pace world, peoples has many challenges to facing. They're is pressures from works, famllys, and societys. The lifes can be very stressfuls.

Sometime, individuals feels overwhelmeds by the demands of the dailys life. There's constant needs to balance between works and personals life, and it can lead to a lots of stresss. Peoples don't have enough time to relaxation.

Another problems is the lack of communications between peoples. Many peoples is always busys with they're own things, and they don't has the times for talkings with each others. This can lead to misunderstandings and communication problems.

The education system also have its problems. Students is often feels stressed because they're have to learns a lots of subjects. The teachers don't always have enough times for helpings each student individually.

In conclusions, lifes is full with challengings and peoples face many problems. The stresss from works, lack of communications, and issues in the educations system are just somes of the problems peoples are dealing with.
"""
```

Good essay grammar grade: 5

Bad essay grammar grade: 1

*** The essay contains grammar issues. Consider reviewing and correcting them. ***

Rendezés, javítgatások, megjelenítés

Végül pedig egy átlagot szerettünk volna vonni, a három szempont alapján, mely a végleges jegyet adja. Ehhez is bemásoltuk az eddigi összes kódot, majd utána adtuk ki a következő prompt-ot.

#9

Create a function that gives a grade between 1 and 5 whole numbers, based on all 3 points(grammar, vocabulary, coherence). Use the average of them

```
python Copy code  
  
def calculate_overall_grade(grammar_grade, vocab_grade, coherence_grade):  
    # Calculate the average grade  
    average_grade = (grammar_grade + vocab_grade + coherence_grade) / 3  
  
    # Round the average grade to the nearest whole number  
    overall_grade = round(average_grade)  
  
    return overall_grade
```

Ezek után a teljes eddigi kód át lett adva egy prompt-ba másolva. Kód tisztítására, és rendezésére kértük, de manuálisan is rendezésre került a kód és felosztása. Végül a rossz és jó példa helyett már csak egy esszé változó maradt, ahová a tesztelendő esszét kell megadni.

#10

provide feedback should be one method that gets all 3 grades (vocab,coherence,grammar) and based on these grades give feedback after the grade was written out

```
def provide_feedback(vocab_grade, coherence_grade, grammar_grade):  
    if vocab_grade < 5:  
        print("*** The essay could be improved with a richer vocabulary  
  
    if coherence_grade < 5:  
        print("*** The essay lacks coherence. Consider improving the fl  
  
    if grammar_grade < 5:  
        print("*** The essay contains grammar issues. Consider reviewin
```

Ezt a végleges jegy kiírása után hívjuk meg, hogy visszajelzést tudjunk adni a felhasználónak, és nem csak egy számot dobunk.

#11

*the coherence score uses fixed numbers for its assign_grade function.
Instead of this make it relative to the length of the essay, as in the
vocab part*

A koherencia értékelésnél is még fix számok voltak így azt is minél jobban esszé specifikusra próbáltuk állítani, hogy az esszé hosszától függjön.

Az arányok nem voltak megfelelők egy teszt érdemi értékelésére, így ezt átírtuk példák alapján, hogy reális tartományok határozzák meg a jegyet, illetve egy másik módszerrel számoltatjuk a max_coherence_score-t is.

A kép az újabb koherencia tesztről a következő oldalon.


```

def check_coherence(text):
    print("Checking Coherence...")
    tokens = nltk.word_tokenize(text.lower())
    tokens = [token for token in tokens if len(token) > 1 and token not in stopwords.words('english')]
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    pos_tags = pos_tag(tokens)
    coherence_score = 0

    # Calculate the maximum possible coherence score based on the number of adjacent word pairs
    max_coherence_score = max(len(tokens) - 1, 1)

    for i in range(len(pos_tags) - 1):
        word1, pos1 = pos_tags[i]
        word2, pos2 = pos_tags[i + 1]

        if is_related(word1, word2) and pos1 == pos2:
            coherence_score += 1

    # Calculate coherence grade relative to the essay length
    coherence_ratio = coherence_score / max_coherence_score

    if coherence_ratio >= 0.06:
        coherence_grade = 5
    elif 0.06 > coherence_ratio >= 0.04:
        coherence_grade = 4
    elif 0.04 > coherence_ratio >= 0.02:
        coherence_grade = 3
    elif 0.02 > coherence_ratio >= 0.01:
        coherence_grade = 2
    else:
        coherence_grade = 1

    print("Coherence Check Complete✔")
    return coherence_grade

```

A nyelvhelyességnél is ugyanezt az esszéhez való arányosságot alkalmaztuk.

```

def assign_grammar_grade(grammar_issues, essay_length):
    total_issues = len(grammar_issues)

    # Calculate the ratio of grammar issues relative to the essay length
    issues_ratio = total_issues / essay_length

    print("Grammar Check Complete✔")
    # Adjust the thresholds based on the ratio
    if issues_ratio <= 0.01:
        return 5 # Perfect grammar
    elif issues_ratio <= 0.02:
        return 4 # Minor issues
    elif issues_ratio <= 0.03:
        return 3 # Moderate issues
    elif issues_ratio <= 0.04:
        return 2 # Substantial issues
    else:
        return 1 # Numerous issues

```

Az előző pár képen már látható volt, hogy valamiféle kiírások történnek. Szerettük volna, ha visszajelzést kap a felhasználó, hogy éppen mi tesztelődik és hogy áll a tesztelés. A ChatGPT ebben az élő visszajelzésre többszöri próbálkozás után sem tudott megfelelő megjelenítést kitalálni, generálni. Így végül ezzel az egyszerű megoldással jelezzük, hogy melyik teszt kezdett el vizsgálni, és hogy befejeződött e.

Az emoji-k is esztétikai szempontból lecserélték a karaktereket, szerintünk így jobban nézett ki a végleges kiírás.

```
def calculate_unique_word_count(text):  
    print("Checking Vocabulary...")  
    words = tokenize_and_filter_punctuation(text)
```

...

```
print("Vocabulary Check Complete✅")  
return unique_word_count_grade
```

```
def provide_feedback(vocab_grade, coherence_grade, grammar_grade):  
    if vocab_grade < 5:  
        print("⚠️ The essay could be improved with a richer vocabulary. ⚠️")  
  
    if coherence_grade < 5:  
        print("⚠️ The essay lacks coherence. Consider improving the flow between sentences and paragraphs. ⚠️")  
  
    if grammar_grade < 5:  
        print("⚠️ The essay contains grammar issues. Consider reviewing and correcting them. ⚠️")
```

Egy kimenet végül így néz ki:

```
Checking Vocabulary...  
Vocabulary Check Complete✅  
Vocab: 5
```

```
Checking Coherence...  
Coherence Check Complete✅  
Coherence: 5
```

```
Checking Grammar...  
Grammar Check Complete✅  
Grammar: 3
```

```
Overall essay grade: 4
```

```
⚠️ The essay contains grammar issues. Consider reviewing and correcting them. ⚠️
```

Így a kódolás részét befejeztük, hasznos képeket és prompt-okat itt megmutattuk.