YAZILIM LABORATUVARI II

3.PROJE

KOCAELİ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

GONCAGÜL KOÇAK

200201108

200201108@kocaeli.edu.tr

Proje Tanımı

Bu projede verilen bir dokümandaki cümlelerin graf yapısına dönüştürülmesi ve bu graf modelinin görselleştirilmesi istenmektedir. Ardından graf üzerindeki düğümler ile özet oluşturan bir algoritma oluşturulması amaçlanmaktadır.

Giriş

Projede masaüstü uygulama yapılmıştır. Projede masaüstü uygulama geliştirmemiz gerekmektedir. Masaüstü uygulamada ilk olarak doküman yükleme işlemi gerçekleştirilecektir. Ardından yüklenen dokümandaki cümleleri graf yapısı haline getirmemiz ve bu graf yapısını görselleştirmemiz beklenmektedir. Bu grafta her bir cümle bir düğümü temsil edecektir. Cümleler arasındaki anlamsal ilişki kurulmalı, cümleler skorlanmalıdır.Bu proje python dili kullanılarak Visual Studio Code idesinde gerçekleşmiştir.

Araştırmalar ve yöntemler

Projede ilk olarak gerekli olan kütüphaneler yüklenmiştir.

```
##Gerekli kütüphaneleri import ediyorum.
import tkinter as tk
from tkinter import filedialog
import networkx as nx
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import string
```

Yukarıda projemde kullanmış olduğum kütüphaneler aşağıda kısa kısa açıklanmıştır.

tkinter: Kullanıcı arabirimi (pencere, düğme, metin kutusu gibi) oluşturmak ve göstermek için kullanılır.

filedialog: Kullanıcıya dosya seçme veya kaydetme gibi işlemler için iletişim kutuları sunar.

networkx: Grafikleri ve ağları temsil etmek ve üzerinde işlemler yapmak için kullanılır.

matplotlib: Verileri grafiklerle görselleştirmek için kullanılır.

cosine_similarity: İki vektör arasındaki benzerliği ölçmek için kullanılır.

string: Metinle ilgili işlemler yapmak için bazı hazır fonksiyonlar ve sabitler sunar.

nltk: Doğal dil işleme için kullanılan bir kütüphanedir. Metinleri analiz etmek, kelimeleri sınıflandırmak, etiketlemek gibi işlemler yapmak için kullanılır.

```
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
from nltk import pos_tag, ne_chunk
from nltk.tokenize import word_tokenize
from nltk.tree import Tree
from nltk.stem import SnowballStemmer
```

nltk kütüphanesinin çeşitli modüllerini indirmemiz gerekmektedir.

pos_tag modülü, bir metindeki kelimeleri cümledeki rollerine göre etiketler.

ne_chunk modülü, bir metindeki isimlendirilmiş varlıkları tanımlamak için kullanılır. Örneğin, kişi adları, yer adları veya organizasyon adları gibi metindeki önemli varlıkları bulmak için kullanılabilir.

word_tokenize modülü, bir metni kelimelere ayırır.

Tree modülü, ağaç yapısında metin yapılarını temsil etmek için kullanılır. Bu, dilbilgisi analizleri veya metin yapısının görselleştirilmesi gibi işlemlerde kullanılabilir.

SnowballStemmer modülü, kelime köklerini bulmak için kullanılır.

```
import re
import string
from nltk import download
from nltk.corpus import stopwords
import gensim.downloader as api
model = api.load("glove-wiki-gigaword-100")
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer
nlp = spacy.load('en_core_web_sm')
```

re kütüphanesi: Metinlerde belirli desenleri bulmak, değiştirmek veya kaldırmak için kullanılan bir kütüphanedir.

string kütüphanesi: Python'da yerleşik olarak sunulan karakter dizisi işleme işlevlerini sağlar.

download fonksiyonu: nltk kütüphanesinin gerektirdiği veri kaynaklarını indirmek için kullanılır.

stopwords modülü: Metinlerde genellikle anlam taşımayan kelimeleri içeren bir dilin durdurma kelimeleri listesini sağlar.

gensim.downloader modülü: Önceden eğitilmiş kelime gömme modellerini indirmemizi ve kullanmamızı sağlar. Bu modeller,

metinlerdeki kelime benzerliklerini veya anlamsal ilişkileri hesaplamak için kullanılır.

spacy kütüphanesi: Hızlı ve etkili bir doğal dil işleme aracıdır. Metinleri ayrıştırmak, dilbilgisi analizi yapmak ve anlamsal analizler gerçekleştirmek gibi NLP görevlerini gerçekleştirmemizi sağlar.

TfidfVectorizer modülü: Metin belgelerini TF-IDF matrisine dönüştürmek için kullanılır. Bu matris, metin belgelerindeki terimlerin önemini belirlemek için kullanılır.

nlp değişkeni: spacy kütüphanesinden yüklenen dil modelini temsil eder.

```
#Özel isim kontrolü gerçekleştiriliyor.
def count_proper_nouns(sentence):
    doc = nlp(sentence)
    proper_nouns = [token for token in doc if token.pos_ == 'PROPN']
    count = len(proper_nouns)
    length = len(doc)
    return count/length
def calculate_p1_for_all_sentences():
    global sentences
    p1_values = []
    for i, sentence in enumerate(sentences):
        p1 = count_proper_nouns(sentence)
        print(f"Sentence {i+1} - P1 value: {p1:.2f}")
        p1_values.append(p1)  # P1 değerini diziye ekle
    print(p1_values)
    return p1_values
```

count_proper_nouns fonksiyonu, bir cümledeki özel isim sayısını bulmak için kullanılır. Verilen cümle sentence parametresi olarak alınır.Cümle, nlp (SpaCy dökümantasyonuna göre bir NLP nesnesi) ile işlenir ve doküman nesnesi oluşturulur.

doc üzerinde dolaşarak her bir token'in (kelime) özelliği kontrol edilir.token.pos_ değeri 'PROPN' (özel isim) ise, bu token özel bir isimdir ve proper_nouns listesine eklenir.proper_nouns listesinin uzunluğu count olarak kaydedilir.

Cümledeki toplam kelime yani token sayısı length olarak kaydedilir.

count/length ifadesi kullanılarak özel isimlerin cümle içindeki oranı hesaplanır ve sonuç döndürülür.

calculate_p1_for_all_sentences fonksiyonu ise tüm cümleler için count_proper_nouns fonksiyonunu çağırarak P1 değerini hesaplar.

Global olarak tanımlanan sentences değişkeni üzerinde döngü oluşturulur.Her bir cümle için count_proper_nouns fonksiyonu çağrılır ve P1 değeri elde edilir.

Elde edilen P1 değeri p1_values listesine eklenir.Cümle numarası ve P1 değeri ekrana yazdırılır.

Son olarak, p1_values listesi döndürülür. Bu kod, verilen cümlelerdeki özel isimlerin cümle içindeki oranını hesaplayarak P1 değerlerini bulur. P1 değeri, her cümlenin içerdiği özel isimlerin oranını temsil etmektedir.

```python
def calculate_p2_for_all_sentences():
    global sentences
    p2_values=[]
    for i, sentence in enumerate(sentences):
        pattern = re.compile(r'\b\d+(?:s|th)?\b')
        matches = pattern.findall(sentence)

        # Cümledeki diğer kelimeleri say ve P2'yi hesapla
        tokens = word_tokenize(sentence.lower())
        words = [word for word in tokens if word not in string.punctuation
        p2 = len(matches) / len(words)
        p2_values.append(p2)  # p2_values değerini daha sonra skor hesabın
        print(f"Sentence {i+1} - P2 value: {p2:.2f}")
```

Her bir cümle için döngü oluşturulur.Verilen cümledeki sayısal ifadeleri bulmak için bir düzenli ifade (regular expression) deseni kullanılır. Bu desen, \b\d+(?:s|th)?\b şeklindedir ve bir veya daha fazla rakamdan oluşan ifadeleri eşleştirir.Eşleşen ifadeleri matches değişkenine atar.

Cümledeki diğer kelimeleri saymak için cümle önce küçük harflere dönüştürülür (lower() metodu kullanılır) ve kelimeler ayıklanır. Noktalama işaretleri dahil edilmez.P2 değeri, eşleşen ifadelerin toplam kelime sayısına oranı olarak hesaplanır: len(matches) / len(words).

Her bir cümle için hesaplanan P2 değeri p2_values listesine eklenir.

```python
    global similarity_matrix
    p3_values=[]
    threshold = threshold_scale.get()
    num_nodes = len(similarity_matrix)

    for i in range(num_nodes):
        p3 = 0
        total_connections = 0

        for j in range(num_nodes):
            if i != j and similarity_matrix[i][j] > threshold:
                p3 += 1

            if similarity_matrix[i][j] > 0:
                total_connections += 1

        p3_value = p3 / total_connections if total_connections > 0 else 0
        p3_values.append(p3_value)

        print(f"Node {i+1} - P3 value: {p3_value:.2f}")
    print(p3_values)
    return p3_values
```

similarity_matrix isimli bir matris, düğümler arasındaki benzerlikleri içerir.threshold_scale isimli bir eşik değeri kullanılır.Her bir düğüm için aşağıdaki adımlar gerçekleştirilir:p3 ve total_connections değerleri sıfıra ayarlanır.

Düğümün diğer düğümlere olan bağlantılarını kontrol eden bir döngü oluşturulur.Eğer iki düğüm birbirine eşit değilse ve benzerlik değeri eşik değerinden büyükse, p3 değeri bir artırılır.

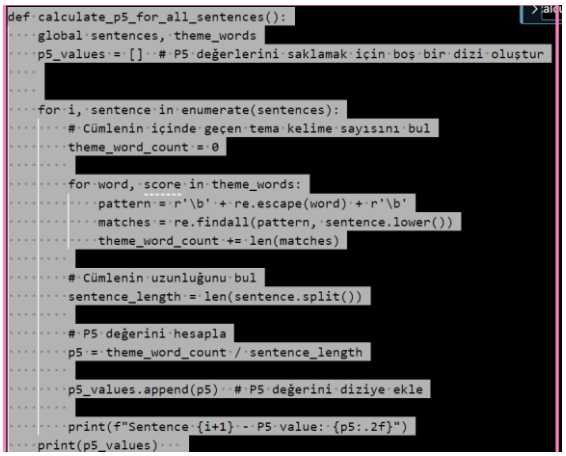
Düğümün diğer düğümlere olan bağlantısı sıfırdan büyükse, total_connections değeri bir artırılır.P3 değeri, p3 / total_connections formülüyle hesaplanır (eğer total_connections sıfırdan büyükse).Hesaplanan P3 değeri p3_values listesine eklenir.Her bir düğüm için hesaplanan P3 değeri ekrana yazdırılır.

```python
    # Başlıkta geçen kelimelerin köklerini bul
    title_stemmed_words = [stemmer.stem(word) for word in title_words]

    # Başlıkta geçen kelimelerin köklerini cümlede say
    count = sum(stemmed_word in title_stemmed_words for stemmed_word in stemmed_words)

    # Cümlenin uzunluğunu hesapla
    length = len(words)

    # P4 değerini hesapla
    p4 = count / length if length > 0 else 0
    p4_values.append(p4)
    print(f"Sentence {i+1} - P4 value: {p4:.2f}")
print(p4_values)
return p4_values
```

sentences ve title değişkenleri global olarak tanımlanmıştır. Bu değişkenler cümleleri ve başlığı içerir.Bir döngü kullanarak her bir cümle için aşağıdaki adımlar gerçekleştirilir:

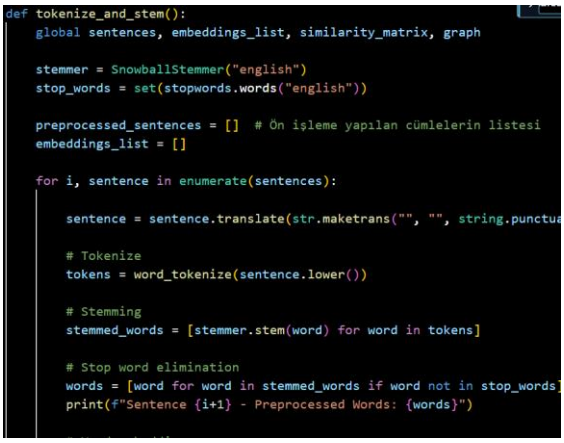
p4_values isimli bir liste oluşturulur.

Başlıktaki kelimeler ayrıştırılır ve title_words değişkenine atanır. Cümledeki köklerin başlıkta geçen köklerle eşleşmesi sayılır.Cümlenin uzunluğu hesaplanır.P4 değeri, eşleşme sayısı / cümlenin uzunluğu olarak hesaplanır (eğer cümlenin uzunluğu sıfırdan büyükse).Hesaplanan P4 değeri p4_values listesine eklenir.Her bir cümle için hesaplanan P4 değeri ekrana yazdırılır.

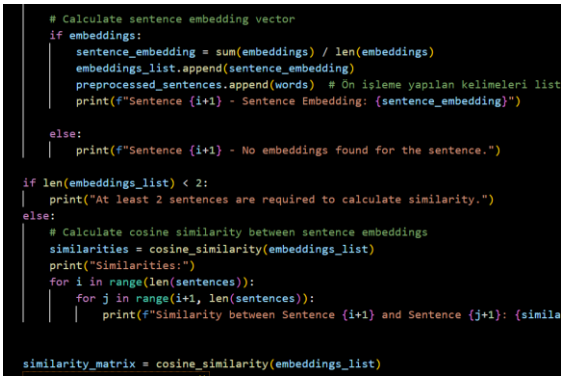


```
def calculate_p5_for_all_sentences():
    global sentences, theme_words
    p5_values = [] # P5 değerlerini saklamak için boş bir dizi oluştur

    for i, sentence in enumerate(sentences):
        # Cümlenin içinde geçen tema kelime sayısını bul
        theme_word_count = 0

        for word, score in theme_words:
            pattern = r'\b' + re.escape(word) + r'\b'
            matches = re.findall(pattern, sentence.lower())
            theme_word_count += len(matches)

        # Cümlenin uzunluğunu bul
        sentence_length = len(sentence.split())

        # P5 değerini hesapla
        p5 = theme_word_count / sentence_length

        p5_values.append(p5) # P5 değerini diziye ekle

        print(f"Sentence {i+1} - P5 value: {p5:.2f}")
    print(p5_values)
```

Her bir cümle üzerinde döngü oluşturulur.Her bir tema kelimesi için, cümlede o kelimenin kaç kez geçtiği sayılır.Cümledeki tema kelime sayısı, cümlenin toplam kelime sayısına bölünür.Elde edilen değer, cümlenin temasıyla ne kadar ilişkili olduğunu gösteren P5 değeridir.P5 değeri her bir cümle için hesaplanır ve bir liste olarak saklanır.Hesaplanan P5 değerleri ekrana yazdırılır.



```
def tokenize_and_stem():
    global sentences, embeddings_list, similarity_matrix, graph

    stemmer = SnowballStemmer("english")
    stop_words = set(stopwords.words("english"))

    preprocessed_sentences = []  # Ön işleme yapılan cümlelerin listesi
    embeddings_list = []

    for i, sentence in enumerate(sentences):

        sentence = sentence.translate(str.maketrans("", "", string.punctua

        # Tokenize
        tokens = word_tokenize(sentence.lower())

        # Stemming
        stemmed_words = [stemmer.stem(word) for word in tokens]

        # Stop word elimination
        words = [word for word in stemmed_words if word not in stop_words]
        print(f"Sentence {i+1} - Preprocessed Words: {words}")
```

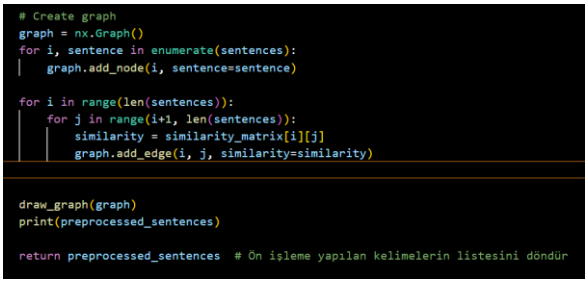


```
    # Calculate sentence embedding vector
    if embeddings:
        sentence_embedding = sum(embeddings) / len(embeddings)
        embeddings_list.append(sentence_embedding)
        preprocessed_sentences.append(words)  # Ön işleme yapılan kelimeleri list
        print(f"Sentence {i+1} - Sentence Embedding: {sentence_embedding}")

    else:
        print(f"Sentence {i+1} - No embeddings found for the sentence.")

if len(embeddings_list) < 2:
    print("At least 2 sentences are required to calculate similarity.")
else:
    # Calculate cosine similarity between sentence embeddings
    similarities = cosine_similarity(embeddings_list)
    print("Similarities:")
    for i in range(len(sentences)):
        for j in range(i+1, len(sentences)):
            print(f"Similarity between Sentence {i+1} and Sentence {j+1}: {simila

similarity_matrix = cosine_similarity(embeddings_list)
```

Kelimeleri köklerine ayırır ve İngilizce stop kelimelerini çıkarır.Her bir cümle üzerinde döngü oluşturur.Cümleyi noktalama işaretlerinden temizler, küçük harflere dönüştürür ve kelimelere ayırır.Kelimeleri köklerine ayırır ve stop kelimelerini çıkarır.Elde edilen ön işlenmiş kelimeleri gösterir.Word2Vec modeli kullanarak kelimeleri gömülerine dönüştürür.Cümle gömülerini hesaplar ve bir listeye ekler.Elde edilen gömüler arasındaki benzerliği hesaplar ve yazdırır.Benzerlik matrisini saklar.p3 değerlerini hesaplamak için calculate_p3_for_all_nodes() fonksiyonunu çağırır.

Word embedding, doğal dil işleme (NLP) alanında kullanılan bir tekniktir. Metin verilerini sayısal vektörler olarak temsil etmeyi sağlar.

Word embedding, kelime dağarcığını çok boyutlu bir sayısal uzayda temsil etmek için kullanılır. Bu uzayda, bir kelimenin anlamı ve ilişkileri, vektör uzayındaki konumuyla ifade edilir. Yani, benzer anlamlı veya bağlamsal olarak yakın kelimeler, bu uzayda birbirlerine yakın konumda yer alır.



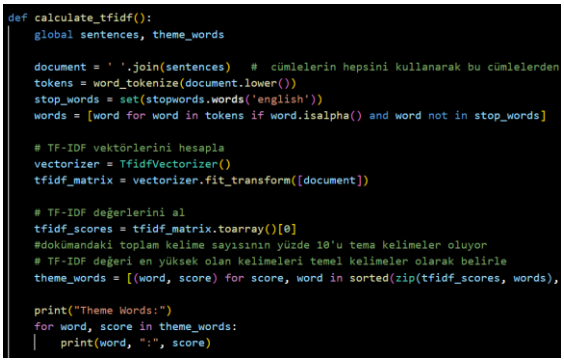
```
# Create graph
graph = nx.Graph()
for i, sentence in enumerate(sentences):
    graph.add_node(i, sentence=sentence)

for i in range(len(sentences)):
    for j in range(i+1, len(sentences)):
        similarity = similarity_matrix[i][j]
        graph.add_edge(i, j, similarity=similarity)


draw_graph(graph)
print(preprocessed_sentences)

return preprocessed_sentences  # Ön işleme yapılan kelimelerin listesini döndür
```

İlk olarak, graph adında bir boş graf oluşturulur. Daha sonra, her cümle için bir düğüm oluşturularak grafa eklenir. Düğümün kimliği (i) ve cümle metni (sentence) düğüme özellik olarak atanır.

Sonra, similarity_matrix içindeki benzerlik değerlerine dayanarak düğümler arasında kenarlar oluşturulur. İki döngü kullanılarak tüm cümle çiftleri kontrol edilir. Her bir kenarın benzerlik değeri (similarity) kenara özellik olarak eklenir.

draw_graph fonksiyonu, oluşturulan grafi görselleştirmek için kullanılabilir. Grafın görsel temsili çizilir ve ekrana görüntülenir.

Son olarak, preprocessed_sentences listesi döndürülür. Bu liste, önceden işlenmiş kelimelerin cümle bazında gruplandığı bir liste içerir,cümlelerin benzerliklerini göstermek ve grafiksel olarak temsil etmek için kullanılmaktadır.



```
def calculate_tfidf():
    global sentences, theme_words

    document = ' '.join(sentences)  #  cümlelerin hepsini kullanarak bu cümlelerden
    tokens = word_tokenize(document.lower())
    stop_words = set(stopwords.words('english'))
    words = [word for word in tokens if word.isalpha() and word not in stop_words]

    # TF-IDF vektörlerini hesapla
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform([document])

    # TF-IDF değerlerini al
    tfidf_scores = tfidf_matrix.toarray()[0]
    #dokümandaki toplam kelime sayısının yüzde 10'u tema kelimeler oluyor
    # TF-IDF değeri en yüksek olan kelimeleri temel kelimeler olarak belirle
    theme_words = [(word, score) for score, word in sorted(zip(tfidf_scores, words),

    print("Theme Words:")
    for word, score in theme_words:
        print(word, ":", score)
```

TF-IDF (Terim Sıklığı-Ters Belge Frekansı) yöntemini kullanarak bir belgedeki kelimelerin önem sırasını hesaplar.

İlk olarak, tüm cümleler birleştirilerek tek bir belge oluşturulur. Bu belgedeki kelimelerin sayısı ve önemlerini hesaplamak için TF-IDF vektörleri kullanılır.

TF-IDF değeri, bir kelimenin belgedeki sıklığına ve diğer belgelerdeki sıklığına dayanır. Yüksek bir TF-IDF değeri, bir kelimenin belgede önemli olduğunu gösterir.

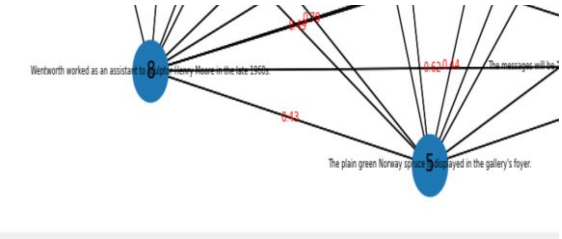
Bu kod parçasında, TF-IDF değeri en yüksek olan kelimeler temel kelimeler olarak belirlenir. Belgedeki toplam kelime sayısının yüzde 10'u kadar kelime temel kelime olarak seçilir.

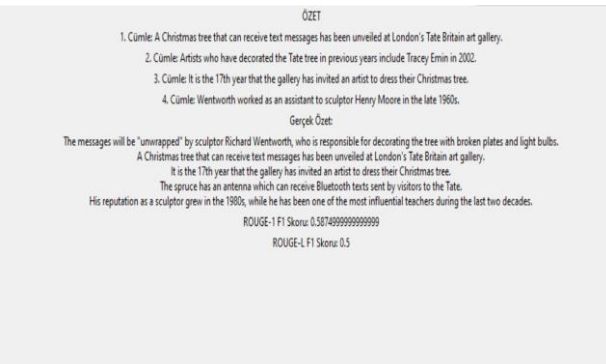
Sonuç olarak, temel kelimeler ve onlara karşılık gelen TF-IDF skorları elde edilir. Bu temel kelimeler, belgedeki önemli kelimeleri temsil eder.Bu kod parçası, belgedeki önemli kelimeleri belirlemek için TF-IDF kullanır ve temel kelimeleri theme_words olarak adlandırılan bir liste olarak döndürür.

Deneysel Sonuçlar:

Projede Python dili kullanılmıştır.VsCode geliştirme ortamında proje geliştirilmiştir. oluşturulmuştur.

Sonuç:





KABA KOD:

1. BAŞLA.
2. Gerekli kütüphaneler import edilir.
3. Özel isim kontrolü için bir fonksiyon tanımlanır: count_proper_nouns().
4. Numerik ifadelerin kontrolü için bir fonksiyon tanımlanır: calculate_p2_for_all_sentences().
5. Benzerlik matrisi hesaplamak için bir fonksiyon tanımlanır: calculate_p3_for_all_nodes().
6. Başlıkta geçen kelimelerin kontrolü için bir fonksiyon tanımlanır: calculate_p4_for_all_sentences().
7. Tema kelimelerinin kontrolü için bir fonksiyon tanımlanır: calculate_p5_for_all_sentences().
8. Cümleleri tokenize edip kök analizi yapmak için bir fonksiyon tanımlanır: tokenize_and_stem().
9. TF-IDF hesaplamak için bir fonksiyon tanımlanır: calculate_tfidf().
10. Tüm cümlelerin değerlerini hesaplamak için bir fonksiyon tanımlanır: calculate_values_for_all_sentences().
11. ROUGE skorlarını hesaplamak için bir fonksiyon tanımlanır: calculate_rouge_scores().

12. Referans metni okumak için bir fonksiyon tanımlanır: read_reference_text().
13. En iyi cümleleri göstermek için bir fonksiyon tanımlanır: show_top_sentences().
14. Kullanıcıdan bir referans dosyası seçmesi istenir.
15. Kullanıcıdan bir metin dosyası seçmesi istenir.
16. Seçilen dosyalardaki metinler okunur.
17. Metinlerin üzerinde işlem yapmak için gerekli fonksiyonlar çağrılır.
18. Özet cümleleri ve ROUGE skorlarını gösteren bir pencere oluşturulur.
19. BİTİR.

Kaynakça:

- https://mdurmuss.github.io/tf-idf-nedir/
- https://www.geeksforgeeks.org/python-word-embedding-using-word2vec/
- https://www.analyticssteps.com/blogs/word-embedding-nlp-python-code
- https://medium.com/algorithms-data-structures/tf-idf-term-frequency-inverse-document-frequency-53feb22a17c6
https://www.sciencedirect.com/topics/computer-science/cosine-similarity#:~:text=Cosine%20similarity%20measures%20the%20similarity,document%20similarity%20in%20text%20analysis.