ابتدا هر سوال به صورت جداگانه نوشته شده و در نهایت کنار هم دیگر مرتب شده اند.

در بخش ابتدایی کد کتابخانه های لازم برای اجرا هر ۴ سوال ایمپورت شده

```
import os
import nltk
import spacy
import pandas as pd
from nltk import ne_chunk
from summa import summarizer
from nltk.corpus import stopwords
from nltk.tag import StanfordNERTagger
from sklearn.metrics.pairwise import linear_kernel
from nltk.tokenize import word_tokenize, sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
```

در قسمت بعد مواردی که از کتابخانه nltk لازم بوده دانلود شده:

```
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

سوال ۱

```
def q_1():
    text = "Natural language processing is fun! This text is a sample text."

    tokens = nltk.word_tokenize(text)
    print("Tokenization:")
    print(tokens)

    pos_tags = nltk.pos_tag(tokens)
    print("\nPOS Tagging:")
    print(pos_tags)

    chunk_grammar = r"""
        NP: {<DT>?<JJ>*<NN>}
    """

    chunk_parser = nltk.RegexpParser(chunk_grammar)
    noun_phrases = chunk_parser.parse(pos_tags)

    print("\nNoun Phrase Chunking:")
    print(noun_phrases)

    noun_phrases.draw()

    text_vp = "She decided to take a stroll in the park."

    tokens_vp = nltk.word_tokenize(text_vp)
    pos_tags_vp = nltk.pos_tag(tokens_vp)

    chunk_grammar_vp = r"""
    VP: {<PRP>?<VBD>?<TO>?<VB>?<DT>?<JJ>*<NN>?}
    """

    chunk_parser_vp = nltk.RegexpParser(chunk_grammar_vp)
```

```
    verb_phrases = chunk_parser_vp.parse(pos_tags_vp)

    print("\nVerb Phrase Chunking:")
    print(verb_phrases)

    verb_phrases.draw()

```

-۱-۱

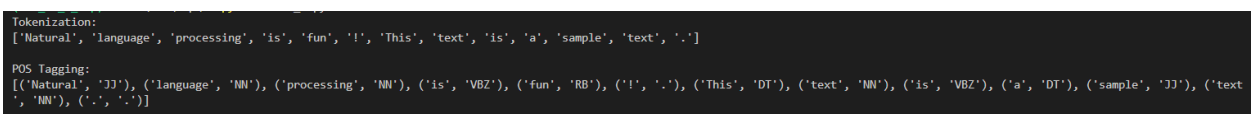
در پردازش زبان طبیعی (NLP)، Chunking یکی از مراحل مهم است که به تجزیه و تحلیل جملات به قطعات یا "چانک‌ها" کمک می‌کند. چانک‌ها مجموعه‌ای از کلمات هستند که به دلایل معنایی یا گراماتیکی با یکدیگر مرتبط هستند و به صورت یک واحد معنایی در نظر گرفته می‌شوند. این فرایند به تجزیه جملات به اجزای معنایی کمک کرده و اطلاعات مهمی را از جملات استخراج می‌کند.

برای مثال، در جمله "من یک کتاب جالب خواندم"، Chunking می‌تواند چانک‌هایی را مشخص کند که تشکیل‌دهنده‌های معنایی این جمله هستند. به عنوان مثال، "من" می‌تواند یک چانک باشد که نشان‌دهنده فاعل است، و "یک کتاب جالب" نیز یک چانک می‌تواند باشد که نشان‌دهنده مفعول است. با استفاده از Chunking، جمله به اجزای معنایی تقسیم شده و درک محتوای آن تسهیل می‌شود.

فرض کنید جملهٔ دیگری مثل "پسر کوچک با یک گلابی در دست بازی می‌کند" را داشته باشیم. Chunking می‌تواند اجزای معنایی این جمله را به صورت گروه‌هایی مانند "پسر کوچک"، "یک گلابی"، "در دست" تشخیص دهد که هرکدام اطلاعات معنایی خاصی از جمله را به ما می‌دهند.

-۱-۲

با استفاده از word_tokenize توکن بندی و با استفاده از pos_tag عملیات تگ زدن را انجام دادیم که خروجی کار میشود:

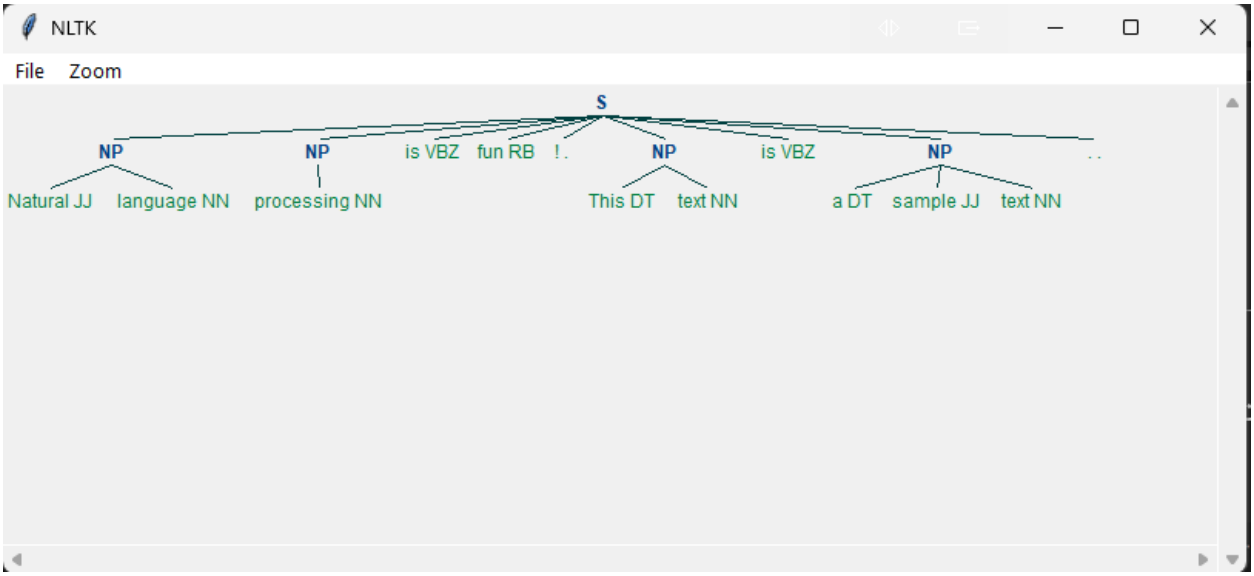



```
1. Tokenization:
2. ['Natural', 'language', 'processing', 'is', 'fun', '!', 'This', 'text', 'is', 'a', 'sample',
'text', '.']
3.
4. POS Tagging:
5. [('Natural', 'JJ'), ('language', 'NN'), ('processing', 'NN'), ('is', 'VBZ'), ('fun', 'RB'), ('!',
'.'), ('This', 'DT'), ('text', 'NN'), ('is', 'VBZ'), ('a', 'DT'), ('sample', 'JJ'), ('text', 'NN'),
('.', '.')]
```

-۱-۳

```
Noun Phrase Chunking:
(S
  (NP Natural/JJ language/NN)
  (NP processing/NN)
  is/VBZ
  fun/RB
  !/.
  (NP This/DT text/NN)
  is/VBZ
  (NP a/DT sample/JJ text/NN)
  ./.)
```

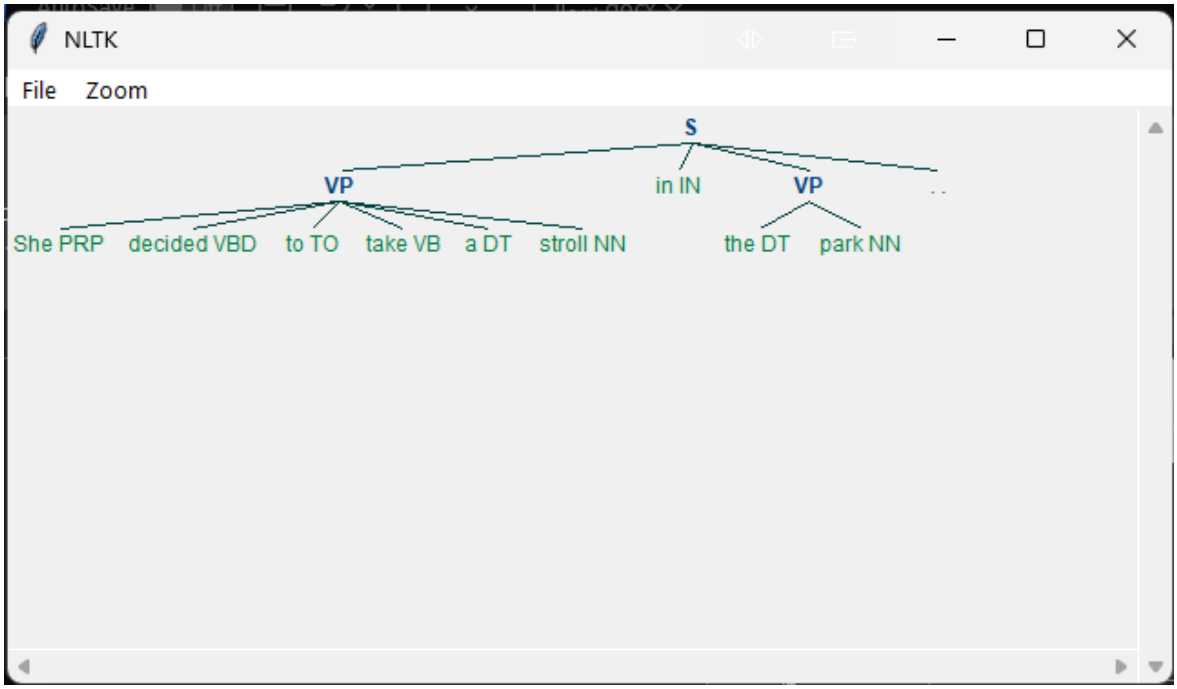
-۱-۴



۱-۵- برای verb phrase chunking از گرامر زیر استفاده شده:

```
1. chunk_grammar_vp = r"""
2.     VP: {<PRP>?<VBD>?<TO>?<VB>?<DT>?<JJ>*<NN>?}
3.     """
```

```
Verb Phrase Chunking:
(S
  (VP She/PRP decided/VBD to/TO take/VB a/DT stroll/NN)
  in/IN
  (VP the/DT park/NN)
  ./.)
```



سوال ۲

۲-۱-

```
with open("Rock.txt", "r", encoding="utf-8") as file:
     rock_text = file.read()

chunk_grammar_iob = r"""
    NP: {<DT>?<JJ>*<NN>}
"""

chunk_parser_iob = nltk.RegexpParser(chunk_grammar_iob)

tokens_iob = word_tokenize(rock_text)
pos_tags_iob = nltk.pos_tag(tokens_iob)

noun_phrases_iob = ne_chunk(pos_tags_iob, binary=True)

print("\nIOB Encoding for Noun Phrases:")
print(noun_phrases_iob)

```

ابتدا فایل را باز کردیم توکنایز و تگ زدن را انجام دادیم سپس با ne_chunk انکود را انجام دیدم(مقدار خروجی زیاد بود و بخشی از خروجی نمایش داده شده):

```
IOB Encoding for Noun Phrases:
(S
  (NE Rock/NNP)
  music/NN
  ,/,
  characterized/VBN
  by/IN
  its/PRP$
  powerful/JJ
  guitar/JJ
  riffs/NNS
  and/CC
  dynamic/JJ
  rhythms/NN
  ,/,
  has/VBZ
  been/VBN
  a/DT
  cultural/JJ
  force/NN
  since/IN
  the/DT
  mid-20th/JJ
  century/NN
  ./.
  One/CD
  of/IN
  the/DT
  most/RBS
  influential/JJ
  bands/NNS
  in/IN
  rock/NN
  history/NN
  is/VBZ
  The/DT
  (NE Beatles/NNP)
  ,/,
  with/IN
  their/PRP$
  groundbreaking/NN
```

-۲-۲

برا این قسمت نیاز به دانلود فایل های مربوطه (stanford-ner.jar و english.all.۳class.distsim.crf.ser.gz) و نصب و کانفیگ محیط مجازی برای جاوا بود در نهایت تگ زدن انجام شد و 'PERSON', 'LOCATION' جدا شده و چاپ شد:

```
    stanford_ner_path = "stanford-ner.jar"
    stanford_ner_model = "english.all.3class.distsim.crf.ser.gz"

    stanford_tagger = StanfordNERTagger(stanford_ner_model, stanford_ner_path)

    java_path = r"C:\Program Files\Java\jdk-21\bin"
    os.environ['JAVAHOME'] = java_path

    sentences_stanford = sent_tokenize(rock_text)
    tokens_stanford = [word_tokenize(sentence) for sentence in sentences_stanford]

    named_entities_stanford = stanford_tagger.tag_sents(tokens_stanford)

    named_entities_stanford = [ent for sent in named_entities_stanford for ent in sent if ent[1]
in ['PERSON', 'LOCATION']]

    print("\nNamed Entities (Stanford NER):")
    for entity in named_entities_stanford:
        print(entity)

```

که خروجی به شرح زیر است:

```
Named Entities (Stanford NER):
('Cleveland', 'LOCATION')
('Ohio', 'LOCATION')
('Elvis', 'PERSON')
('Presley', 'PERSON')
('Woodstock', 'LOCATION')
('Bethel', 'LOCATION')
('New', 'LOCATION')
('York', 'LOCATION')
('Jimi', 'PERSON')
('Hendrix', 'PERSON')
('Janis', 'PERSON')
('Joplin', 'PERSON')
('London', 'LOCATION')
('Gibson', 'PERSON')
('Lester', 'PERSON')
('Bangs', 'PERSON')
('Greil', 'PERSON')
('Marcus', 'PERSON')
('London', 'LOCATION')
('David', 'PERSON')
('Crosby', 'PERSON')
('Stephen', 'PERSON')
('Stills', 'PERSON')
('Paul', 'PERSON')
('Kantner', 'PERSON')
('Seattle', 'LOCATION')
```

```
Named Entities (Stanford NER):
('Cleveland', 'LOCATION')
('Ohio', 'LOCATION')
('Elvis', 'PERSON')
('Presley', 'PERSON')
('Woodstock', 'LOCATION')
('Bethel', 'LOCATION')
('New', 'LOCATION')
('York', 'LOCATION')
('Jimi', 'PERSON')
('Hendrix', 'PERSON')
('Janis', 'PERSON')
('Joplin', 'PERSON')
('London', 'LOCATION')
('Gibson', 'PERSON')
('Lester', 'PERSON')
('Bangs', 'PERSON')
('Greil', 'PERSON')
('Marcus', 'PERSON')
('London', 'LOCATION')
('David', 'PERSON')
('Crosby', 'PERSON')
('Stephen', 'PERSON')
('Stills', 'PERSON')
('Paul', 'PERSON')
('Kantner', 'PERSON')
('Seattle', 'LOCATION')
('Washington', 'LOCATION')
('Pearl', 'LOCATION')
('Jam', 'LOCATION')
('Soundgarden', 'LOCATION')
('Seattle', 'LOCATION')
('Coachella', 'LOCATION')
('Glastonbury', 'LOCATION')
('John', 'PERSON')
('Lennon', 'PERSON')
('London', 'LOCATION')
('Carnaby', 'LOCATION')
('Street', 'LOCATION')
('Jim', 'PERSON')
('Morrison', 'PERSON')
('Jimi', 'PERSON')
('Hendrix', 'PERSON')
('Kurt', 'PERSON')
('Cobain', 'PERSON')
('New', 'LOCATION')
('York', 'LOCATION')
('Los', 'LOCATION')
('Angeles', 'LOCATION')
('Eric', 'PERSON')
('Clapton', 'PERSON')
('Woodstock', 'LOCATION')

```

۳-۲-

برای این قسمت ابتدا از کتابخانه spacy en_core_web_sm لود شده و پردازش روی متن انجام شده و 'PERSON', 'GPE' استخراج شده و در نهایت چاپ شده:

```
nlp = spacy.load("en_core_web_sm")
    doc_spacy = nlp(rock_text)

    named_entities_spacy = [(ent.text, ent.label_) for ent in doc_spacy.ents if ent.label_ in ['PERSON', 'GPE']]

    print("\nNamed Entities (Spacy NER):")
    for entity in named_entities_spacy:
        print(entity)

```

خروجی:

```
Named Entities (Spacy NER):
('Cleveland', 'GPE')
('Ohio', 'GPE')
('Elvis Presley', 'PERSON')
('Led Zeppelin', 'PERSON')
('Nirvana', 'GPE')
('Bethel', 'GPE')
('New York', 'GPE')
('Jimi Hendrix', 'PERSON')
('Janis Joplin', 'PERSON')
('London', 'GPE')
('Gibson', 'PERSON')
('London', 'GPE')
('The Beatles', 'GPE')
('Wooden Ships', 'PERSON')
('David Crosby', 'PERSON')
('Stephen Stills', 'PERSON')
('Paul Kantner', 'PERSON')
('Seattle', 'GPE')
('Washington', 'GPE')
('Nirvana', 'GPE')
('Pearl Jam', 'PERSON')
('Soundgarden', 'PERSON')
('Seattle', 'GPE')
("John Lennon's", 'PERSON')
('London', 'GPE')
('Jim Morrison', 'PERSON')
```

```
Named Entities (Spacy NER):
('Cleveland', 'GPE')
('Ohio', 'GPE')
('Elvis Presley', 'PERSON')
('Led Zeppelin', 'PERSON')
('Nirvana', 'GPE')
('Bethel', 'GPE')
('New York', 'GPE')
('Jimi Hendrix', 'PERSON')
('Janis Joplin', 'PERSON')
('London', 'GPE')
('Gibson', 'PERSON')
('London', 'GPE')
('The Beatles', 'GPE')
```

```
('Wooden Ships', 'PERSON')
('David Crosby', 'PERSON')
('Stephen Stills', 'PERSON')
('Paul Kantner', 'PERSON')
('Seattle', 'GPE')
('Washington', 'GPE')
('Nirvana', 'GPE')
('Pearl Jam', 'PERSON')
('Soundgarden', 'PERSON')
('Seattle', 'GPE')
("John Lennon's", 'PERSON')
('London', 'GPE')
('Jim Morrison', 'PERSON')
('Jimi Hendrix', 'PERSON')
('Kurt Cobain', 'PERSON')
('The Beatles', 'GPE')
('Beatles', 'PERSON')
('New York', 'GPE')
('Los Angeles', 'GPE')
('Eric Clapton', 'PERSON')
('Led Zeppelin', 'PERSON')
```

۴-۲-

۱-تگ های نهاد های نامی:

در Stanford NER، مکان ها با تگ "LOCATION" شناخته می شوند، در حالی که Spacy از "GPE" (سیاست-مکان-انتیتی) برای تشخیص مکان ها استفاده می کند.

Stanford NER از تگ "PERSON" برای افراد استفاده می کند، در حالی که Spacy از تگ "PERSON" استفاده می کند.

۲-تشخیص افراد:

Stanford NER در برخی موارد افراد را به صورت جداگانه تشخیص داده است (مانند 'Elvis' و 'Presley')، در حالی که Spacy آنها را به عنوان یک نهاد نامی تشخیص داده است ('Elvis Presley').

تفاوت های دیگر در تشخیص افراد نیز دیده می شود.

۳-کیفیت تشخیص:

هر دو ابزار توانمندی خوبی در تشخیص نهادهای نامی از جمله مکان ها و افراد دارند.

مثال های مشابه را با دقت شناسایی کرده اند.

سوال ۳

۳-۱-

ابتدا متن را باز کردیم و استاپ ورد ها و مواردی که الفابت نبودند را حذف کردیم:

```
with open("Rock.txt", "r", encoding="utf-8") as file:
    rock_text = file.read()

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

tokens = word_tokenize(rock_text)

filtered_tokens = [word.lower() for word in tokens if word.isalpha() and word.lower() not in stop_words]
preprocessed_text = ' '.join(filtered_tokens)

```

۳-۲-

ابتدا وکتور را با ۱۰ ویژگی ایجاد کردیم و با فیت ترنسفورم داده ها را در وکتور قرار دادیم با استفاده از get_feature_names_out نام ها را استخراج کردیم و در نهایت کلمه ها را پیدا کرده و چاپ کردیم:

```
tfidf_vectorizer = TfidfVectorizer(max_features=10)
tfidf_matrix = tfidf_vectorizer.fit_transform([preprocessed_text])

feature_names = tfidf_vectorizer.get_feature_names_out()

top_keywords = [feature_names[idx] for idx in tfidf_matrix.indices]

print("\nTop 10 Key Words Extracted with TF-IDF:")
print(top_keywords)

```

خروجی:

```
Top 10 Key Words Extracted with TF-IDF:
['influence', 'new', 'like', 'artists', 'musicians', 'fame', 'beatles', 'bands', 'music', 'rock']
```

۳-۳-

برای خلاصه سازی ابتدا توکن بندی را انجام دادیم و سپس جملاتی که حاوی کلمات کلیدی مرحله قبل بودند را استخراج کردیم و در نهایت ۵ جمله اول را چاپ کردیم:

```
sentences = sent_tokenize(rock_text)

key_sentences = [sentence for sentence in sentences if any(keyword in sentence for keyword in top_keywords)]
key_sentences = key_sentences[:5]

print("\nSummary Based on Key Sentences:")
print(' '.join(key_sentences))
```

خروجی:

```
1. Summary Based on Key Sentences:
Rock music, characterized by its powerful guitar riffs and dynamic rhythms, has been a cultural
force since the mid-20th century. One of the most influential bands in rock history is The Beatles,
with their groundbreaking sound and global impact. The Rock and Roll Hall of Fame, located in
Cleveland, Ohio, honors the achievements of notable musicians, bands, and industry figures.
Established in 1983, the Hall of Fame has inducted iconic artists like Elvis Presley, Led Zeppelin,
and Nirvana. Woodstock, the legendary music festival held in 1969, became a symbol of the
counterculture movement.
```

۳-۴-

الگوریتم TextRank یک روش مبتنی بر گراف برای استخراج مهم ترین عبارات یا کلمات کلیدی از یک متن است. این الگوریتم بر اساس اصل اهمیت گراف ها و ارتباطات بین کلمات در یک متن عمل می کند. در این الگوریتم، ابتدا متن به جملات تقسیم شده و کلمات در هر جمله استخراج می شوند. سپس یک گراف جهت دار ساخته می شود که هر گره آن به یک کلمه از متن نسبت دارد.

وزن هر یال در گراف بر اساس اهمیت نسبی دو کلمه در متن محاسبه می شود. این اهمیت معمولاً با استفاده از معیارهایی مانند فراوانی همزمان ظاهر شدن کلمات یا احتمال انتخاب یک کلمه به عنوان کلمه مرکزی در جملات محاسبه می شود. سپس با اعمال الگوریتم ترتیب دهی گراف، کلمات با بیشترین اهمیت به عنوان کلمات کلیدی شناخته می شوند.

الگوریتم TextRank به دلیل سادگی پیاده سازی و کارایی در استخراج کلمات کلیدی از متون طولانی، به عنوان یک روش محبوب در حوزه پردازش متن شناخته می شود.

۳-۵-

برای پیاده سازی این قسمت از کتابخانه summa استفاده شده که از الگوریتم textRank استفاده میکند:

```
1.      text_rank_summary = summarizer.summarize(rock_text, ratio=0.15)
2.
3.      print("\nSummary Based on TextRank:")
4.      print(text_rank_summary)
5.
```

Ratio برای کنترل طول خروجی می باشد که به صورت تقریبی ست شده.

خروجی:

```
1. Summary Based on TextRank:
Gibson and Fender, renowned guitar manufacturers, have played a crucial role in shaping the sound of rock music.
The Rock and Roll Hall of Fame Foundation, responsible for the annual inductions, ensures that the legacy of rock music is celebrated and preserved for future generations.
The Rock and Roll Hall of Fame induction ceremony is a star-studded event attended by music legends, industry insiders, and fans.
The street's influence on the fashion scene reflects the symbiotic relationship between rock music and style.
Their influence extends beyond business, shaping the trajectory of rock music as a whole.
The influence of blues music on rock is evident in the work of artists like Eric Clapton and Led Zeppelin.
```

۶-۳-

شباهت ها:

- هر دو خلاصه به تأثیرات تاریخی و فرهنگی موسیقی راک اشاره دارند.
- هر دو به مهمترین شخصیت ها و گروه های موسیقی راک اشاره دارند.
- اطلاعات مربوط به تالار مشاهیر راک اند رول در هر دو خلاصه حضور دارد.
-

تفاوت ها:

- خلاصه اول بیشتر به صفحه های تاریخی و معرفی گروه ها می پردازد، در حالی که خلاصه دوم به جزئیات فنی مرتبط با موسیقی راک می پردازد.
- خلاصه دوم به مسائل فنی مانند تأثیر تولیدکنندگان گیتار و جزئیات مراسم ورود به تالار مشاهیر راک اند رول تأکید دارد.
- محتوای خلاصه اول به نظر گسترده تر و عمیق تر می آید، در حالی که خلاصه دوم به جزئیات تخصصی محدود تری متمرکز شده است.

سوال ۴

برای این سوال ابتدا اکسل مربوطه را باز کردیم و ستون های 'title', 'genres', 'keywords' را انتخاب کردیم و ترکیب 'genres', 'keywords' را در ستون جدید combined_features قرار دادیم سپس تابع get_recommendations را فراخوانی کرده و مقادیر را به همراه امتیاز چاپ کردیم:

```
    df = pd.read_csv("tmdb_5000_movies.csv")
    selected_columns = ['title', 'genres', 'keywords']
    df = df[selected_columns]
    df['combined_features'] = df['genres'] + ' ' + df['keywords']

    recommendations_mortal_kombat = get_recommendations(df, 'Mortal Kombat')
    recommendations_flywheel = get_recommendations(df, 'Flywheel')
    recommendations_frozen = get_recommendations(df, 'Frozen')

    print("Recommendations for Mortal Kombat:")
    for title, score in recommendations_mortal_kombat:
        print(f"{title} - Similarity Score: {score}")

    print("\nRecommendations for Flywheel:")
    for title, score in recommendations_flywheel:
        print(f"{title} - Similarity Score: {score}")

    print("\nRecommendations for Frozen:")
    for title, score in recommendations_frozen:
        print(f"{title} - Similarity Score: {score}")
```

پیاده سازی تابع get_recommendations به این شکل می باشد که ابتدا وکتور tf idf را تشکیل می دهد و داده های ستون combined_features را داخل آن قرار میدهد و با شباهت کسینوسی توسط linear_kernel ۵ مورد از بالا ترین شباهت ها را استخراج میکند و در مرحله بعدی امتیاز شباهت هر فیلم را هم استخراج کرده و با هم ترکیب کرده و بازگشت میدهد:

```
def get_recommendations(df, movie_title):
    movie_index = df[df['title'] == movie_title].index[0]

    tfidf_vectorizer = TfidfVectorizer()
    tfidf_matrix = tfidf_vectorizer.fit_transform(df['combined_features'])

    cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

    sim_scores = list(enumerate(cosine_sim[movie_index]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)[1:6]

    movie_indices = [x[0] for x in sim_scores]
    recommendations = df['title'].iloc[movie_indices]

    return list(zip(recommendations.values, [round(score, 4) for _, score in sim_scores]))
```

خروجی:

```
(ven_hm_3_nlp) PS D:\sku\nlp\3> python code_1.py
Recommendations for Mortal Kombat:
Mortal Kombat: Annihilation - Similarity Score: 0.714
DOA: Dead or Alive - Similarity Score: 0.5096
In the Name of the King III - Similarity Score: 0.5013
Resident Evil: Retribution - Similarity Score: 0.4752
Street Fighter: The Legend of Chun-Li - Similarity Score: 0.467

Recommendations for Flywheel:
Fireproof - Similarity Score: 0.3565
The Apostle - Similarity Score: 0.3556
Time Changer - Similarity Score: 0.3502
Faith Like Potatoes - Similarity Score: 0.3428
Fat Albert - Similarity Score: 0.3376

Recommendations for Frozen:
Happy Feet Two - Similarity Score: 0.5036
Valiant - Similarity Score: 0.4996
Aladdin - Similarity Score: 0.4993
Legend of a Rabbit - Similarity Score: 0.4942
Jonah: A VeggieTales Movie - Similarity Score: 0.4922
```