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 4 سوال ایمپورت شده

1. import os

2. import nltk

3. import spacy

4. import pandas as pd

5. from nltk import ne\_chunk

6. from summa import summarizer

7. from nltk.corpus import stopwords

8. from nltk.tag import StanfordNERTagger

9. from sklearn.metrics.pairwise import linear\_kernel

10. from nltk.tokenize import word\_tokenize, sent\_tokenize

11. from sklearn.feature\_extraction.text import TfidfVectorizer

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

1. nltk.download('punkt')

2. nltk.download('averaged\_perceptron\_tagger')

3. nltk.download('maxent\_ne\_chunker')

4. nltk.download('words')

سوال 1

1. def q\_1():

2. text = "Natural language processing is fun! This text is a sample text."

3.

4. tokens = nltk.word\_tokenize(text)

5. print("Tokenization:")

6. print(tokens)

7.

8. pos\_tags = nltk.pos\_tag(tokens)

9. print("\nPOS Tagging:")

10. print(pos\_tags)

11.

12. chunk\_grammar = r"""

13. NP: {<DT>?<JJ>\*<NN>}

14. """

15.

16. chunk\_parser = nltk.RegexpParser(chunk\_grammar)

17. noun\_phrases = chunk\_parser.parse(pos\_tags)

18.

19. print("\nNoun Phrase Chunking:")

20. print(noun\_phrases)

21.

22. noun\_phrases.draw()

23.

24. text\_vp = "She decided to take a stroll in the park."

25.

26. tokens\_vp = nltk.word\_tokenize(text\_vp)

27. pos\_tags\_vp = nltk.pos\_tag(tokens\_vp)

28.

29. chunk\_grammar\_vp = r"""

30. VP: {<PRP>?<VBD>?<TO>?<VB>?<DT>?<JJ>\*<NN>?}

31. """

32.

33. chunk\_parser\_vp = nltk.RegexpParser(chunk\_grammar\_vp)

34. verb\_phrases = chunk\_parser\_vp.parse(pos\_tags\_vp)

35.

36. print("\nVerb Phrase Chunking:")

37. print(verb\_phrases)

38.

39. verb\_phrases.draw()

40.

1-1-

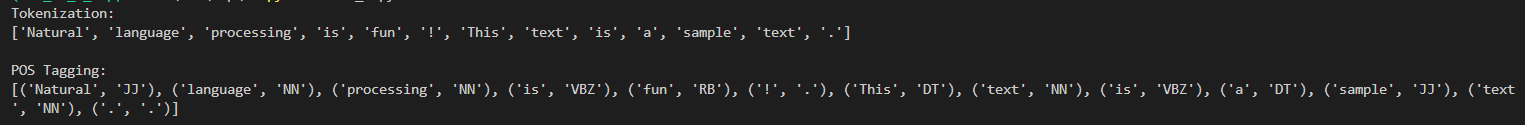
در پردازش زبان طبیعی (NLP)، Chunking یکی از مراحل مهم است که به تجزیه و تحلیل جملات به قطعات یا "چانک‌ها" کمک می‌کند. چانک‌ها مجموعه‌ای از کلمات هستند که به دلایل معنایی یا گراماتیکی با یکدیگر مرتبط هستند و به صورت یک واحد معنایی در نظر گرفته می‌شوند. این فرایند به تجزیه جملات به اجزای معنایی کمک کرده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

1-2-

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:

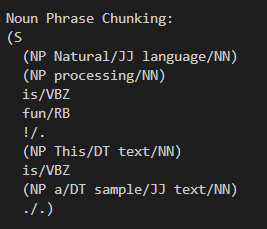
2. ['Natural', 'language', 'processing', 'is', 'fun', '!', 'This', 'text', 'is', 'a', 'sample', 'text', '.']

3.

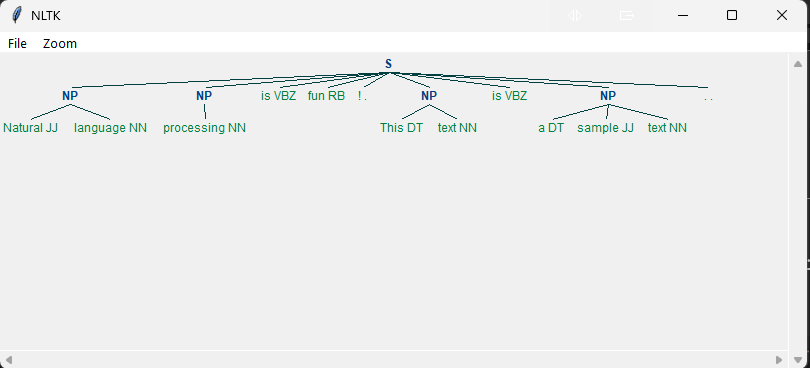
4. POS Tagging:

5. [('Natural', 'JJ'), ('language', 'NN'), ('processing', 'NN'), ('is', 'VBZ'), ('fun', 'RB'), ('!', '.'), ('This', 'DT'), ('text', 'NN'), ('is', 'VBZ'), ('a', 'DT'), ('sample', 'JJ'), ('text', 'NN'), ('.', '.')]

1-3-



1-4-

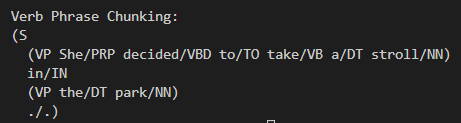


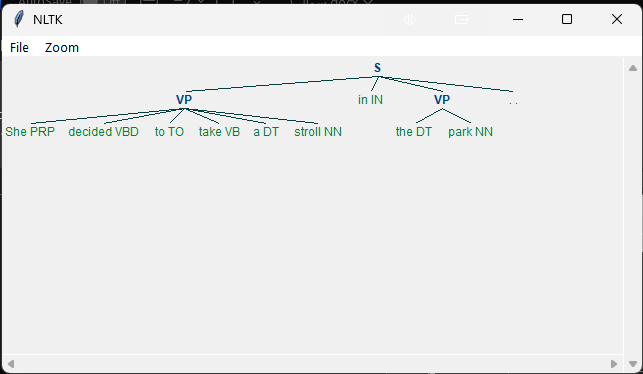
1-5- برای verb phrase chunking از گرامر زیر استفاده شده:

1. chunk\_grammar\_vp = r"""

2. VP: {<PRP>?<VBD>?<TO>?<VB>?<DT>?<JJ>\*<NN>?}

3. """





سوال 2

1-2-

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

2. rock\_text = file.read()

3.

4 .chunk\_grammar\_iob = r"""

5. NP: {<DT>?<JJ>\*<NN>}

6. """

7.

8. chunk\_parser\_iob = nltk.RegexpParser(chunk\_grammar\_iob)

9.

10. tokens\_iob = word\_tokenize(rock\_text)

11. pos\_tags\_iob = nltk.pos\_tag(tokens\_iob)

12.

13. noun\_phrases\_iob = ne\_chunk(pos\_tags\_iob, binary=True)

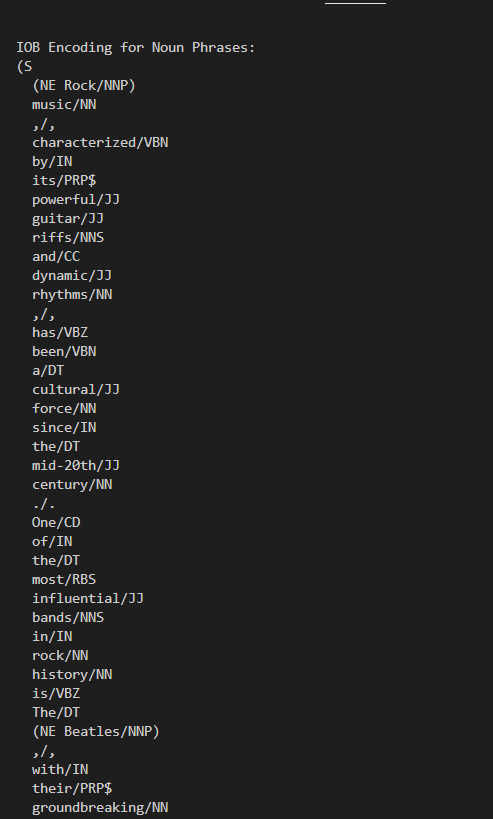
14.

15. print("\nIOB Encoding for Noun Phrases:")

16. print(noun\_phrases\_iob)

17.

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):



2-2-

برا این قسمت نیاز به دانلود فایل های مربوطه (stanford-ner.jar و english.all.3class.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:

1. stanford\_ner\_path = "stanford-ner.jar"

2. stanford\_ner\_model = "english.all.3class.distsim.crf.ser.gz"

3.

4. stanford\_tagger = StanfordNERTagger(stanford\_ner\_model, stanford\_ner\_path)

5.

6. java\_path = r"C:\Program Files\Java\jdk-21\bin"

7. os.environ['JAVAHOME'] = java\_path

8.

9. sentences\_stanford = sent\_tokenize(rock\_text)

10. tokens\_stanford = [word\_tokenize(sentence) for sentence in sentences\_stanford]

11.

12. named\_entities\_stanford = stanford\_tagger.tag\_sents(tokens\_stanford)

13.

14. named\_entities\_stanford = [ent for sent in named\_entities\_stanford for ent in sent if ent[1] in ['PERSON', 'LOCATION']]

15.

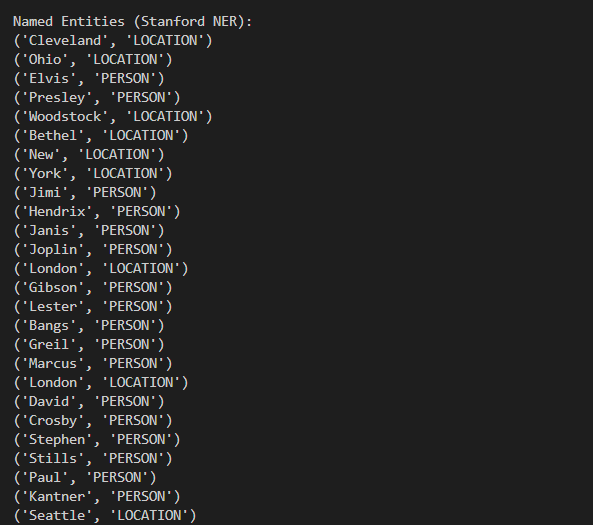
16. print("\nNamed Entities (Stanford NER):")

17. for entity in named\_entities\_stanford:

18. print(entity)

19.

که خروجی به شرح زیر است:



1. Named Entities (Stanford NER):

2. ('Cleveland', 'LOCATION')

3. ('Ohio', 'LOCATION')

4. ('Elvis', 'PERSON')

5. ('Presley', 'PERSON')

6. ('Woodstock', 'LOCATION')

7. ('Bethel', 'LOCATION')

8. ('New', 'LOCATION')

9. ('York', 'LOCATION')

10. ('Jimi', 'PERSON')

11. ('Hendrix', 'PERSON')

12. ('Janis', 'PERSON')

13. ('Joplin', 'PERSON')

14. ('London', 'LOCATION')

15. ('Gibson', 'PERSON')

16. ('Lester', 'PERSON')

17. ('Bangs', 'PERSON')

18. ('Greil', 'PERSON')

19. ('Marcus', 'PERSON')

20. ('London', 'LOCATION')

21. ('David', 'PERSON')

22. ('Crosby', 'PERSON')

23. ('Stephen', 'PERSON')

24. ('Stills', 'PERSON')

25. ('Paul', 'PERSON')

26. ('Kantner', 'PERSON')

27. ('Seattle', 'LOCATION')

28. ('Washington', 'LOCATION')

29. ('Pearl', 'LOCATION')

30. ('Jam', 'LOCATION')

31. ('Soundgarden', 'LOCATION')

32. ('Seattle', 'LOCATION')

33. ('Coachella', 'LOCATION')

34. ('Glastonbury', 'LOCATION')

35. ('John', 'PERSON')

36. ('Lennon', 'PERSON')

37. ('London', 'LOCATION')

38. ('Carnaby', 'LOCATION')

39. ('Street', 'LOCATION')

40. ('Jim', 'PERSON')

41. ('Morrison', 'PERSON')

42. ('Jimi', 'PERSON')

43. ('Hendrix', 'PERSON')

44. ('Kurt', 'PERSON')

45. ('Cobain', 'PERSON')

46. ('New', 'LOCATION')

47. ('York', 'LOCATION')

48. ('Los', 'LOCATION')

49. ('Angeles', 'LOCATION')

50. ('Eric', 'PERSON')

51. ('Clapton', 'PERSON')

52. ('Woodstock', 'LOCATION')

53.

2-3-

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:

1. nlp = spacy.load("en\_core\_web\_sm")

2. doc\_spacy = nlp(rock\_text)

3.

4. named\_entities\_spacy = [(ent.text, ent.label\_) for ent in doc\_spacy.ents if ent.label\_ in ['PERSON', 'GPE']]

5.

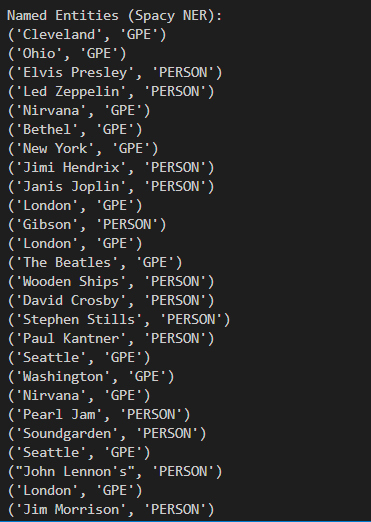
6. print("\nNamed Entities (Spacy NER):")

7. for entity in named\_entities\_spacy:

8. print(entity)

9.

خروجی:



1. Named Entities (Spacy NER):

2. ('Cleveland', 'GPE')

3. ('Ohio', 'GPE')

4. ('Elvis Presley', 'PERSON')

5. ('Led Zeppelin', 'PERSON')

6. ('Nirvana', 'GPE')

7. ('Bethel', 'GPE')

8. ('New York', 'GPE')

9. ('Jimi Hendrix', 'PERSON')

10. ('Janis Joplin', 'PERSON')

11. ('London', 'GPE')

12. ('Gibson', 'PERSON')

13. ('London', 'GPE')

14. ('The Beatles', 'GPE')

15. ('Wooden Ships', 'PERSON')

16. ('David Crosby', 'PERSON')

17. ('Stephen Stills', 'PERSON')

18. ('Paul Kantner', 'PERSON')

19. ('Seattle', 'GPE')

20. ('Washington', 'GPE')

21. ('Nirvana', 'GPE')

22. ('Pearl Jam', 'PERSON')

23. ('Soundgarden', 'PERSON')

24. ('Seattle', 'GPE')

25. ("John Lennon's", 'PERSON')

26. ('London', 'GPE')

27. ('Jim Morrison', 'PERSON')

28. ('Jimi Hendrix', 'PERSON')

29. ('Kurt Cobain', 'PERSON')

30. ('The Beatles', 'GPE')

31. ('Beatles', 'PERSON')

32. ('New York', 'GPE')

33. ('Los Angeles', 'GPE')

34. ('Eric Clapton', 'PERSON')

35. ('Led Zeppelin', 'PERSON')

2-4-

1-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.

2-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.

3-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 3

3-1-

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

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

2. rock\_text = file.read()

3.

4. stop\_words = set(stopwords.words("english"))

5.

6. tokens = word\_tokenize(rock\_text)

7.

8. filtered\_tokens = [word.lower() for word in tokens if word.isalpha() and word.lower() not in stop\_words]

9. preprocessed\_text = ' '.join(filtered\_tokens)

10.

3-2-

ابتدا وکتور را با 10 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:

1. tfidf\_vectorizer = TfidfVectorizer(max\_features=10)

2. tfidf\_matrix = tfidf\_vectorizer.fit\_transform([preprocessed\_text])

3.

4. feature\_names = tfidf\_vectorizer.get\_feature\_names\_out()

5.

6. top\_keywords = [feature\_names[idx] for idx in tfidf\_matrix.indices]

7.

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

9. print(top\_keywords)

10.

خروجی:



3-3-

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 5 جمله اول را چاپ کردیم:

1. sentences = sent\_tokenize(rock\_text)

2.

3. key\_sentences = [sentence for sentence in sentences if any(keyword in sentence for keyword in top\_keywords)]

4. key\_sentences = key\_sentences[:5]

5.

6. print("\nSummary Based on Key Sentences:")

7. 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.

3-4-

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.

3-5-

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.

3-6-

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 4

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:

1. df = pd.read\_csv("tmdb\_5000\_movies.csv")

2. selected\_columns = ['title', 'genres', 'keywords']

3. df = df[selected\_columns]

4. df['combined\_features'] = df['genres'] + ' ' + df['keywords']

5.

6. recommendations\_mortal\_kombat = get\_recommendations(df, 'Mortal Kombat')

7. recommendations\_flywheel = get\_recommendations(df, 'Flywheel')

8. recommendations\_frozen = get\_recommendations(df, 'Frozen')

9.

10. print("Recommendations for Mortal Kombat:")

11. for title, score in recommendations\_mortal\_kombat:

12. print(f"{title} - Similarity Score: {score}")

13.

14. print("\nRecommendations for Flywheel:")

15. for title, score in recommendations\_flywheel:

16. print(f"{title} - Similarity Score: {score}")

17.

18. print("\nRecommendations for Frozen:")

19. for title, score in recommendations\_frozen:

20. 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 5 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:

1. def get\_recommendations(df, movie\_title):

2. movie\_index = df[df['title'] == movie\_title].index[0]

3.

4. tfidf\_vectorizer = TfidfVectorizer()

5. tfidf\_matrix = tfidf\_vectorizer.fit\_transform(df['combined\_features'])

6.

7. cosine\_sim = linear\_kernel(tfidf\_matrix, tfidf\_matrix)

8.

9. sim\_scores = list(enumerate(cosine\_sim[movie\_index]))

10. sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)[1:6]

11.

12. movie\_indices = [x[0] for x in sim\_scores]

13. recommendations = df['title'].iloc[movie\_indices]

14.

15. return list(zip(recommendations.values, [round(score, 4) for \_, score in sim\_scores]))

خروجی:

