

# تحلیل داده‌های حجمی

## تمرین اول

نام و نام خانوادگی: ناصر کاظمی

شماره دانشجویی: ۹۹۱۰۲۰۵۹

### • سوال ۱

(الف)

برای join کردن جدول‌های داده شده، می‌توان ابتدا  $R$  را با  $S$  و  $T$  را با  $U$  جوین کنیم.  
برای جوین  $(S(B, C) \text{ و } R(A, B))$  داریم:

Read  $R$ :  $(b_i, (R, a_i, b_i))$

Read  $S$ :  $(b_j, (S, b_j, c_j))$

که مرحله map را شامل می‌شود. برای  $\langle \text{key}, \text{value} \rangle$  reduce مقادیر communication می‌شود. در این فرایند replication rate مقدار MapReduce برابر ۱ است. و cost برابر است با:

$$O(r + s)$$

که

$$|R| = r, |S| = s$$

و اندازه جدول حاصل  $prs$  است که ورودی join نهایی می‌شود.

برای جوین  $(T(C, D) \text{ و } U(D, E))$  به طور مشابه عمل می‌کنیم و حاصل این دو join را با هم join می‌کنیم. برای هزینه محاسبات کل این سه عمل map و reduce داریم:

$$O(r + s + prs + t + u + ptu)$$

Replication rate در این الگوریتم ۱ است. زیرا هر ردیف از هر جدول تنها ۱ بار خوانده می‌شود. از آنجایی که در این الگوریتم دو فرآیند map و reduce اول می‌توانند همزمان اجرا شوند و سپس حاصل این مرحله به mapperهای مرحله بعد داده می‌شود، ابتدا همه nodeها را به mapperهای این مرحله اختصاص می‌دهیم. که این تعداد برابر است با:

$$r + s + t + u$$

برای تعداد reducerها نیز تعداد  $(\min\{r, s\} + \min\{t, u\}) \times p$  معقول است. زیرا به طور تقریبی به این تعداد جفت تشکیل می‌شود.

برای تعداد reducerها و mapperهای مرحله دوم نیز به طور مشابه به ترتیب مقادیر  $prs + pst$  و  $p \times \min\{prs, pst\}$

برای اندازه reducerها با در نظر گرفتن بدترین حالت (یعنی زمانی که تمام ردیف‌های جدول‌ها در join مشترک باشد) داریم:

$$|RS_{reducer}| = r + s, \quad |TU_{reducer}| = t + u, \quad |RSTU_{reducer}| = rs + pt \quad (b)$$

برای اینکه join را با یک single-step MapReduce انجام دهیم، باید تمامی ترکیب ردیف‌هایی از همه جداول را که می‌توانند به یک join منجر شوند را با هم به یک reducer بدهیم. برای این کار، چون  $\mathbb{F}$  جدول داریم، هر key به صورت یک تاپل  $^{3\times 3}$  است. فرض کنید توابع hash ستون‌های  $B, C$  و  $D$  به ترتیب  $g, h$  و  $f$  باشند. در این صورت عمل read در مرحله map به صورت زیر خواهد بود:

Read  $R: ((i, j, k), (R, a_i, b_i)) \mid i = h(b_i), \forall j \in C_{bucket}, k \in D_{bucket}$

Read  $S: ((i, j, k), (S, b_i, c_j)) \mid i = h(b_i), j = g(c_j), \forall k \in D_{bucket}$

Read  $T: ((i, j, k), (T, c_j, d_k)) \mid j = g(c_j), k = f(d_j), \forall i \in B_{bucket}$

Read  $U: ((i, j, k), (R, a_i, b_i)) \mid k = f(d_k), \forall i \in B_{bucket}, j \in C_{bucket}$

در مرحله reduce به هر باکت تمام جفت‌های  $(i, j, k)$  داده می‌شود و عمل join انجام می‌گیرد.

فرض کنید سایز باکت‌های  $D, B$  و  $C$  به ترتیب برابر با  $b, c$  و  $d$  باشد. در این صورت هزینه انتقال عناصر هر جدول به باکت‌های مربوط به صورت زیر خواهد بود:

$$R(A, B) : rcd, S(B, C) : sd, T(C, D) : tb, U(D, E) : ubc$$

همچنین برای خواندن هر ردیف از جدول‌ها برای ورودی mapperها هزینه  $u$  را خواهیم داشت. پس هزینه کل این الگوریتم برابر است با:

$$r + s + u + t + rcd + sd + tb + ubc$$

در این روش برای هر عنصر در جدول‌های داده شده مقدار replication rate برابر است با:

$$R(A, B) : cd, S(B, C) : d, T(C, D) : b, U(D, E) : bc$$

یا به طور میانگین:

$$\frac{rcd + sd + tb + ubc}{r + s + t + u}$$

برای محاسبه تعداد reducerها و mapperها فرض کنید در کل  $k$  ماشین داریم. حال باید مسئله بهینه‌سازی زیر را

$$\min \quad r + s + u + t + rcd + sd + tb + ubc$$

با قید  $k$  حل کنیم. با استفاده از روش ضرایب لاگرانژ داریم:

$$L(b, c, d, \lambda) = rcd + sd + tb + ubc - \lambda(bcd - k)$$

حال با محاسبه مشتقهای ضمنی خواهیم داشت:

$$\frac{\partial L}{\partial b} = t + uc - \lambda cd = 0$$

$$\frac{\partial L}{\partial c} = rd + ub - \lambda bd = 0$$

$$\frac{\partial L}{\partial d} = rc + s - \lambda bc = 0$$

برای سادگی حل دستگاه بالا اندازه همه جدول‌ها را برابر  $r$  در نظر می‌گیریم و خواهیم داشت:

$$\frac{\partial L}{\partial b} = r + rc - \lambda cd = 0 \quad (1)$$

$$\frac{\partial L}{\partial c} = rd + rb - \lambda bd = 0 \quad (2)$$

$$\frac{\partial L}{\partial d} = rc + r - \lambda bc = 0 \quad (3)$$

$$(1,3) \implies b = d, \lambda = \frac{2r}{b}$$

$$(1) \implies r(c+1) - 2rc = 0 \implies c = 1$$

$$\implies b = d = \sqrt{k}$$

و هزینه کل در کمترین حالت برابر با  $4r + 4r\sqrt{k}$  است.

حال اگر هزینه این روش را با این پارامترها با روش قبل مقایسه کنیم خواهیم داشت:

$$8r + 2pr^2 < 4r + 4r\sqrt{k} \implies 2 + pr < 2\sqrt{k}$$

پس با دانستن پارامترهای  $r, k$  و  $p$  روشی که هزینه کمتری دارد را انتخاب می‌کنیم.

## • سوال ۲

Importing libraries:

```
1 import pandas as pd
2 import numpy as np
3
4 from matplotlib import pyplot as plt
5 import seaborn as sns
```

(الف)

```
1 df = pd.read_csv('MOVIE_IMDB.csv')
```

ب) با استفاده از توابع `isnull` و `sum` یک بار روی ستون‌ها و بار دیگر رو سطرها تعداد null‌های هر ستون `dataframe` را بدست می‌آوریم و سپس درصد هر کدام را محاسبه می‌کنیم.

```
1 # number of nulls in each row
2 df.isnull().sum(axis=1)

0      0
1      0
2      0
3      0
4     14
...
5038    4
5039    5
5040    4
5041    2
5042    0
Length: 5043, dtype: int64
```

```
1 # number of nulls in each column
2 df.isnull().sum()

color                  19
director_name          104
num_critic_for_reviews 50
duration                15
director_facebook_likes 104
actor_3_facebook_likes  23
actor_2_name              13
actor_1_facebook_likes   7
gross                 884
genres                  0
actor_1_name              7
movie_title                0
num_voted_users               0
cast_total_facebook_likes   0
actor_3_name              23
```

facenumber_in_poster	13
plot_keywords	153
movie_imdb_link	0
num_user_for_reviews	21
language	12
country	5
content_rating	303
budget	492
title_year	108
actor_2_facebook_likes	13
imdb_score	0
aspect_ratio	329
movie_facebook_likes	0
<b>dtype: int64</b>	

```
1 # percentage of nulls in each column
2 round(df.isnull().sum() / df.shape[0] * 100, 2)
```

color	0.38
director_name	2.06
num_critic_for_reviews	0.99
duration	0.30
director_facebook_likes	2.06
actor_3_facebook_likes	0.46
actor_2_name	0.26
actor_1_facebook_likes	0.14
gross	17.53
genres	0.00
actor_1_name	0.14
movie_title	0.00
num_voted_users	0.00
cast_total_facebook_likes	0.00
actor_3_name	0.46

facenumber_in_poster	0.26
plot_keywords	3.03
movie_imdb_link	0.00
num_user_for_reviews	0.42
language	0.24
country	0.10
content_rating	6.01
budget	9.76
title_year	2.14
actor_2_facebook_likes	0.26
imdb_score	0.00
aspect_ratio	6.52
movie_facebook_likes	0.00

پ) ابتدا با دستور drop ستون‌های ذکر شده را حذف می‌کنیم. سپس فرآیند قسمت قبل را تکرار می‌کنیم

```

1 new_df = df.drop(columns=[
2     "color",
3     "director_facebook_likes",
4     "actor_1_facebook_likes",
5     "actor_2_facebook_likes",
6     "actor_3_facebook_likes",
7     "actor_2_name",
8     "cast_total_facebook_likes",
9     "actor_3_name",
10    "duration",
11    "facenumber_in_poster",
12    "content_rating",
13    "country",
14    "movie_imdb_link",
15    "aspect_ratio",
16    "plot_keywords"
17 ], axis=0)

```

```

1 # number of nulls in each row
2 cleared_df.isnull().sum(axis=1)

0      0
1      0
2      0
3      0
5      0
..    
5033   0
5034   0
5035   0
5037   0
5042   0
Length: 3891, dtype: int64

```

```

1 cleared_df.isnull().sum()

director_name      0
num_critic_for_reviews  1
gross            0
genres           0
actor_1_name      3
movie_title       0
num_voted_users   0
num_user_for_reviews  0
language          3
budget            0
title_year        0
imdb_score        0
movie_facebook_likes  0
dtype: int64

```

```

1 # percentage of nulls in each column
2 null_ratio = new_df.isnull().sum() / df.shape[0] * 100
3 round(null_ratio, 2)

director_name      2.06
num_critic_for_reviews  0.99
gross            17.53
genres           0.00
actor_1_name      0.14
movie_title       0.00
num_voted_users   0.00
num_user_for_reviews  0.42
language          0.24
budget            9.76
title_year        2.14
imdb_score        0.00
movie_facebook_likes  0.00
dtype: float64

```

حال سطرهایی که دارای عناصر null در ستون‌های با بیش از ۵ درصد null هستند را حذف می‌کنیم:

```
1 cleared_df = new_df.dropna(axis=0, subset=null_ratio[null_ratio > 5.0].index)
2 cleared_df
```

و مجدداً قسمت ب را تکرار می‌کنیم. سپس مقدار language را برای سطرهایی که NaN است را برابر قرار می‌دهیم English.

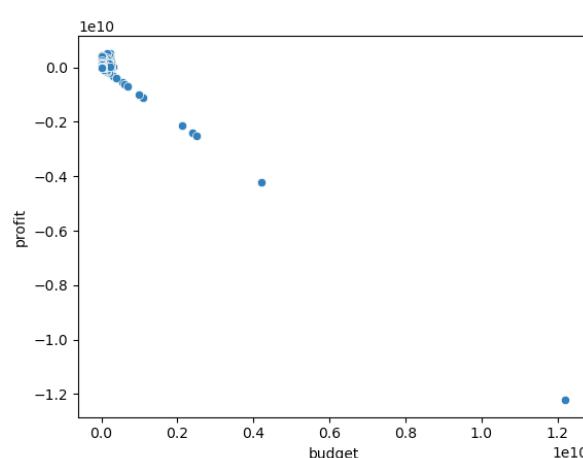
```
1 cleared_df.loc[list(cleared_df[cleared_df.language.isna()].index), 'language'] = 'English'
2 cleared_df.isna().sum()
```

```
1 round(cleared_df.isnull().sum() / cleared_df.shape[0] * 100, 2)
```

Column	Percentage
director_name	0.00
num_critic_for_reviews	0.03
gross	0.00
genres	0.00
actor_1_name	0.08
movie_title	0.00
num_voted_users	0.00
num_user_for_reviews	0.00
language	0.00
budget	0.00
title_year	0.00
imdb_score	0.00
movie_facebook_likes	0.00

dtype: float64

(ت)

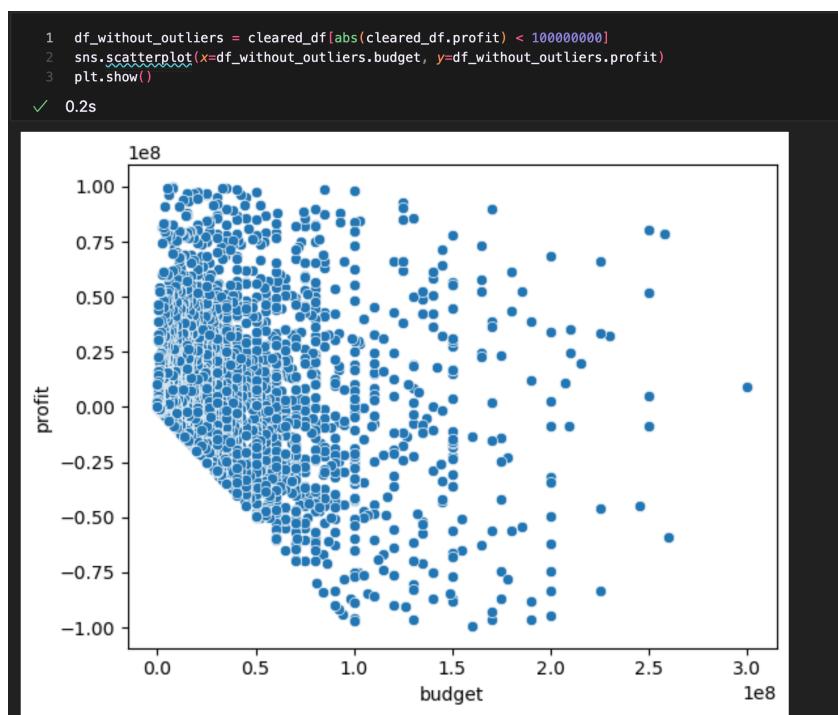


در نمودار بالا مشاهده می‌کنیم برخی از داده‌ها از سایر مقادیر بسیار دور هستند (outlier) با بررسی فیلم‌های مربوط به این داده‌ها متوجه می‌شویم که تقریباً همه فیلم‌ها خارجی از کشورهایی مثل کره هستند که مرتبه بزرگی واحد پولشان با دلار تفاوت زیادی دارد.

```
1 cleared_df[cleared_df.profit < -1000000000]
✓ 0.1s
```

	director_name	num_critic_for_reviews	gross	genres	actor_1_name	movie_title	num_voted_users
2323	Hayao Miyazaki	174.0	2298191.0	Adventure Animation Fantasy	Minnie Driver	Princess Mononoke	221552
2334	Katsuhiro Otomo	105.0	410388.0	Action Adventure Animation Family Sci-Fi Thriller	William Hootkins	Steamboy	13727
2988	Joon-ho Bong	363.0	2201412.0	Comedy Drama Horror Sci-Fi	Doona Bae	The Host	68883
3005	Lajos Koltai	73.0	195888.0	Drama Romance War	Marcell Nagy	Fateless	5603
3423	Katsuhiro Otomo	150.0	439162.0	Action Animation Sci-Fi	Mitsuo Iwata	Akira	106160
3859	Chan-wook Park	202.0	211667.0	Crime Drama	Min-sik Choi	Lady Vengeance	53508

با حذف outlierها به همچین نموداری می‌رسیم:



## لیست ۱۰ فیلم پرسود:

```
1 cleared_df.nlargest(10, "profit")
✓ 0.2s
```

	director_name	num_critic_for_reviews	gross	genres	actor_1_name	movie_title
0	James Cameron	723.0	760505847.0	Action Adventure Fantasy Sci-Fi	CCH Pounder	Avatar
29	Colin Trevorrow	644.0	652177271.0	Action Adventure Sci-Fi Thriller	Bryce Dallas Howard	Jurassic World
26	James Cameron	315.0	658672302.0	Drama Romance	Leonardo DiCaprio	Titanic
3024	George Lucas	282.0	460935665.0	Action Adventure Fantasy Sci-Fi	Harrison Ford	Star Wars: Episode IV - A New Hope
3080	Steven Spielberg	215.0	434949459.0	Family Sci-Fi	Henry Thomas	E.T. the Extra-Terrestrial
17	Joss Whedon	703.0	623279547.0	Action Adventure Sci-Fi	Chris Hemsworth	The Avengers
509	Roger Allers	186.0	422783777.0	Adventure Animation Drama Family Musical	Matthew Broderick	The Lion King
240	George Lucas	320.0	474544677.0	Action Adventure Fantasy Sci-Fi	Natalie Portman	Star Wars: Episode I - The Phantom Menace
66	Christopher Nolan	645.0	533316061.0	Action Crime Drama Thriller	Christian Bale	The Dark Knight
439	Gary Ross	673.0	407999255.0	Adventure Drama Sci-Fi Thriller	Jennifer Lawrence	The Hunger Games

:)

```
1 flatten = lambda l: [item for sublist in l for item in sublist]
2 genres = set(flatten(cleared_df.genres.str.split('|').tolist()))
3 print(genres)
✓ 0.4s
```

Python

```
{'Thriller', 'Documentary', 'Biography', 'Sci-Fi', 'Action', 'Music', 'Western', 'Crime', 'Family', 'Mystery', 'Short', 'Musical', 'Comedy', 'War', 'Drama', 'Animation', 'Horror', 'Romance', 'Sport', 'Film-Noir', 'History', 'Fantasy', 'Adventure'}
```

```
1 movies_by_genre_dict = {genre: cleared_df[cleared_df.genres.str.contains(genre)] for genre in genres}
2 genres_count = {genre: len(movies_by_genre_dict[genre]) for genre in genres}
3 print(genres_count)
✓ 0.1s
```

Python

```
{'Thriller': 1118, 'Documentary': 67, 'Biography': 243, 'Sci-Fi': 497, 'Action': 962, 'Music': 248, 'Western': 60, 'Crime': 715, 'Family': 451, 'Mystery': 384, 'Short': 2, 'Musical': 103, 'Comedy': 1503, 'War': 160, 'Drama': 1944, 'Animation': 199, 'Horror': 391, 'Romance': 879, 'Sport': 151, 'Film-Noir': 1, 'History': 154, 'Fantasy': 514, 'Adventure': 787}
```

```
1 genres_mean_profit = {genre: movies_by_genre_dict[genre].profit.mean() for genre in genres}
2 print(genres_mean_profit)
✓ 0.1s
```

Python

```
{'Thriller': 4644834.937388193, 'Documentary': 6838889.567164179, 'Biography': 7711934.790123457, 'Sci-Fi': -15577344.259557344, 'Action': 4971130.581081081, 'Music': 15271409.028225806, 'Western': 2233042.7, 'Crime': 680683.9342657343, 'Family': 21747365.980044346, 'Mystery': 10532474.111979166, 'Short': 909267.0, 'Musical': 13970168.349514563, 'Comedy': 8159866.379906853, 'War': -23010531.725, 'Drama': -3307779.098251029, 'Animation': 939052.2914572865, 'Horror': -18042006.710997444, 'Romance': 9791503.893060295, 'Sport': 11437047.59602649, 'Film-Noir': -2292073.0, 'History': -9732728.980519481, 'Fantasy': 16894707.011673152, 'Adventure': 13264381.252858957}
```

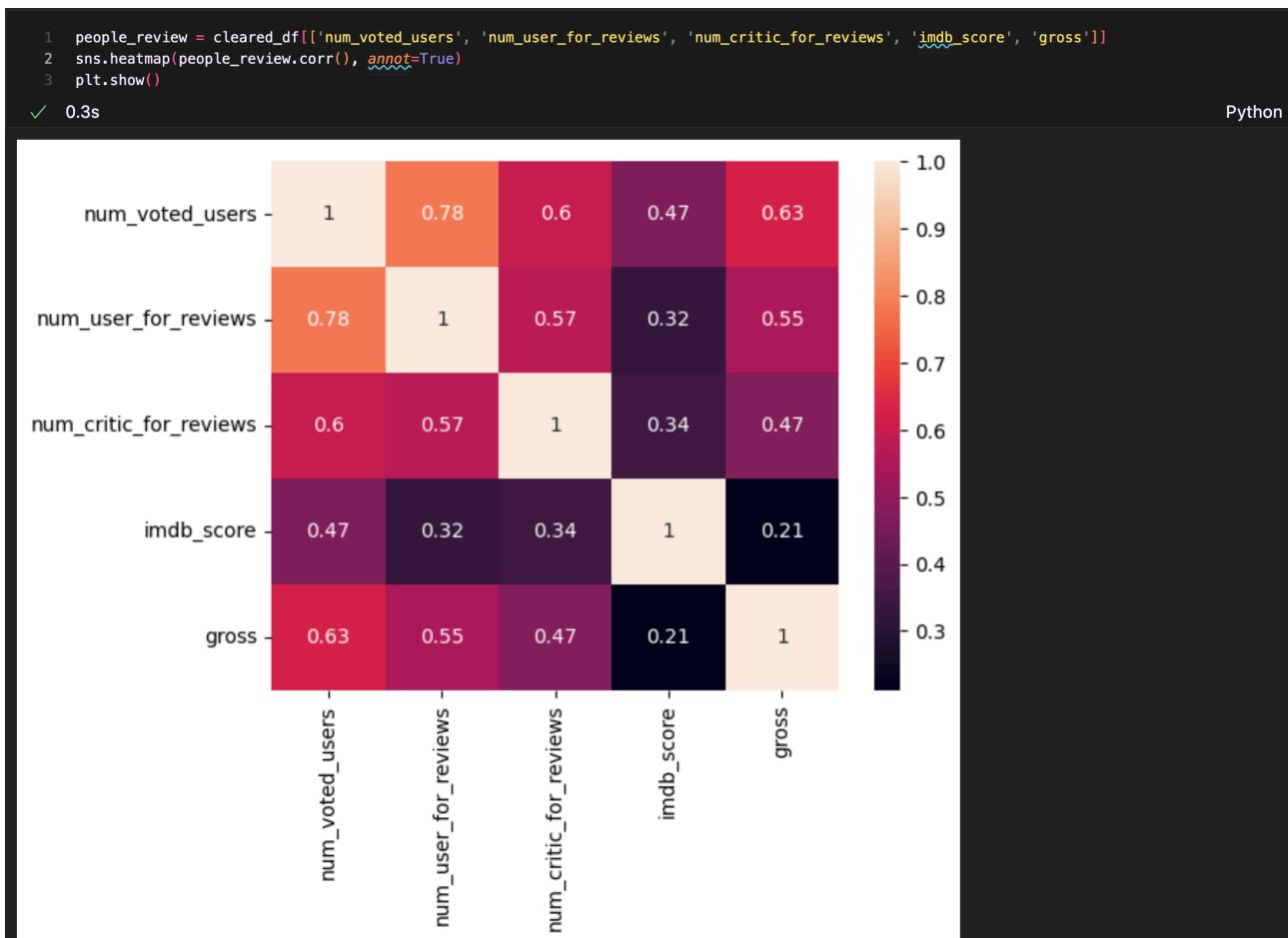
```

1 selected_movies = cleared_df[(2007 <= cleared_df.title_year) & (cleared_df.title_year <= 2015) & (7.5 <= cleared_df.imdb_score) & (
2 | | | cleared_df.imdb_score <= 8.5)]
3 selected_movies.genres
✓ 0.8s
Python

0 Action|Adventure|Fantasy|Sci-Fi
3 Action|Thriller
7 Adventure|Animation|Comedy|Family|Fantasy|Musi...
8 Action|Adventure|Sci-Fi
9 Adventure|Family|Fantasy|Mystery
...
4748 Documentary|Sport
4890 Documentary
4904 Documentary|History|Music
4922 Drama|Romance
4931 Drama|Music|Romance
Name: genres, Length: 212, dtype: object

```

چ) ستون‌هایی که به نظر مردم و فروش و محبوبیت یک فیلم ارتباط دارند را جدا می‌کنیم و دو به دو کوواریانس بین داده‌های این ستون‌ها را محاسبه می‌کنیم و به صورت نمودار heatmap نمایش می‌دهیم.



می‌بینیم که کورلیشن نسبتاً بالایی بین نظر مردم و فروش فیلم‌هاست.

سوال ۳

نتایج این گزارش براساس پردازش یک نمونه 5% از کل داده‌ها است. نتایج محاسبات بر روی کل داده‌ها در جوپیتر نوتبوک موجود است.

## بخش دوم:

ابتدا مقالات را به فرمت JSON بارگزاری و سپس کارکترهای اضافی را از متن و عنوان مقالات حذف می‌کنیم:

سپس با استفاده از تابع flatMap عنوان و متن تمام مقالات را جدا کرده و به توکن‌های کلمه می‌شکنیم. در ادامه با استفاده از map هر از هر کلمه جفت (word, 1) را می‌سازیم و با استفاده از reduceByKey تعداد دفعاتی که هر کلمه آمده است را محاسبه می‌کنیم. در نهایت ۱۰۰ کلمه با بیشترین فرکانس را بدست می‌آوریم:

```
1 words_rdd = cleansed_articles_rdd.flatMap(lambda x: (x['title'], x['text'])).flatMap(lambda x: x.split())
2 words_count_rdd = words_rdd.map(lambda x: (x, 1))
3 words_count = words_count_rdd.reduceByKey(lambda x, y: x + y)
✓ 0.1s
```

Python

---

```
1 words_count_sorted = words_count.map(lambda x : (x[1], x[0])).sortByKey(False)
```

Python

---

```
1 words_count_ls = words_count_sorted.collect()
2 top_100 = words_count_ls[:100]
3 top_100
✓ 28.3s
```

Python

سپس کلمات stopwords را با iterate کردن روی متن و عنوان هر مقاله حذف می‌کنیم:

برای حذف uncommonها، به جای اینکه روی متن‌ها iterate کنیم و هر کلمه را بررسی کنیم، کلمات common را ابتدا بدست می‌آوریم و سپس با استفاده یک تابع hash آنها را درون ۱۰۰۰ باکت می‌ریزیم و سپس براساس این درهم‌سازی کلمات را فیلتر می‌کنیم:

```
1 #TODO: remove uncommon words from articles_without_stopwords_rdd
2
3 def word_hash(word, mod):
4     tmp = 0
5     for w in word:
6         tmp += ord(w)
7     tmp *= len(word)
8     return tmp % mod
9
10
11 bucket_size = 1000
12 hashed_words = [[] for _ in range(bucket_size)]
13 for word in common_words:
14     hashed_words[word_hash(word, bucket_size)].append(word)
15
16
17 def remove_uncommon_words(article):
18     title = article['title']
19     text = article['text']
20
21     title_words = [word for word in title.split() if word != '' and word in hashed_words[word_hash(word, bucket_size)]]
22     text_words = [word for word in text.split() if word != '' and word in hashed_words[word_hash(word, bucket_size)]]
23
24     article['title'] = ' '.join(title_words)
25     article['text'] = ' '.join(text_words)
26
27 return article
```

## بخش سوم:

برای پیدا کردن کلمات ۳ حرفی به صورت زیر عمل می‌کنیم:

```
1 words_cleaned_rdd = articles_cleaned_rdd.flatMap(lambda x: (x['title'], x['text'])).flatMap(lambda x: x.split())
✓ 0.4s                                         Python

1 words_cleaned_rdd.filter(lambda x: len(x) == 3).distinct().count()
✓ 28.9s                                         Python

503
```

برای پیدا کردن trigram‌ها ابتدا یک تابع می‌نویسیم که از یک جمله ۳ تایی‌ها را جدا کند، سپس با استفاده انکودينگ utf-8 انگلیسی بودن کلمه را چک می‌کنیم:

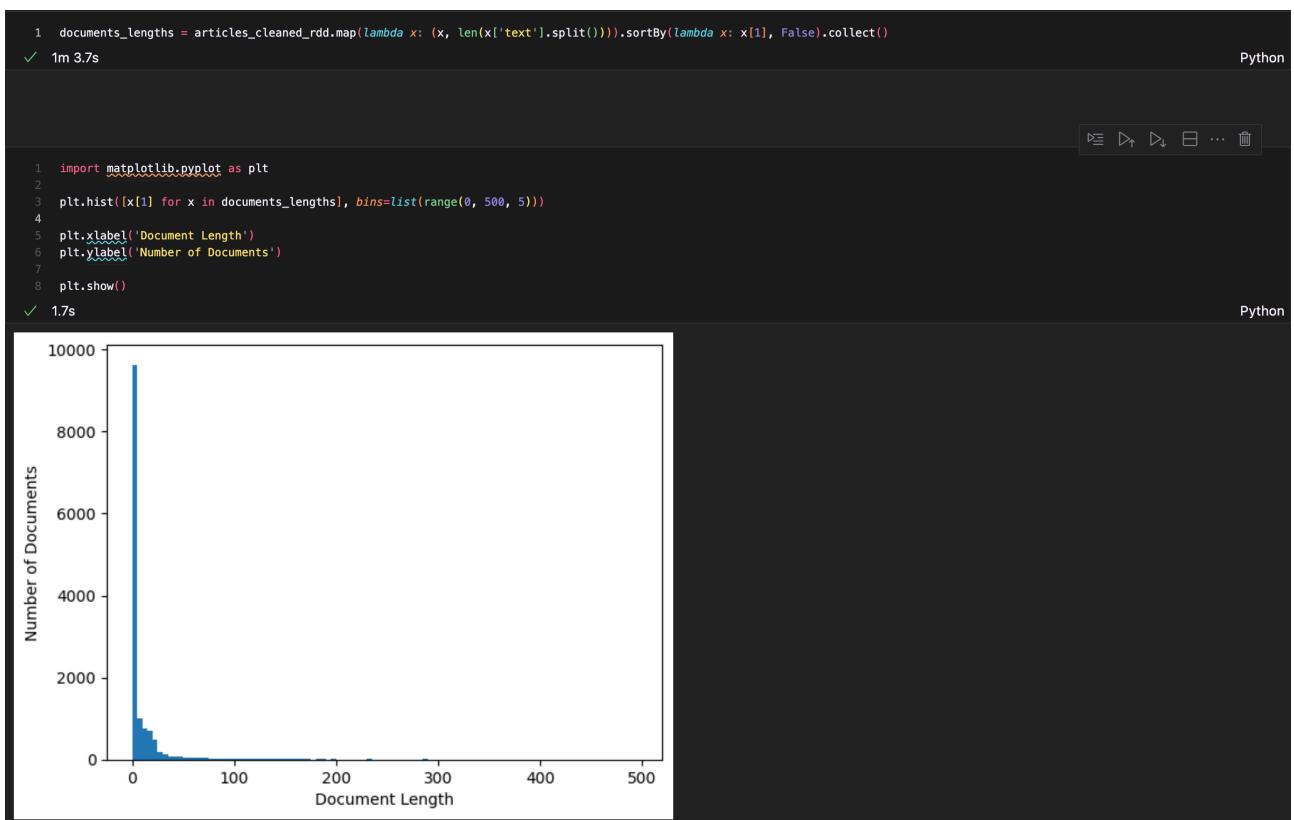
```
1 def trigram(text):
2     words = text.split()
3     trigrams = []
4     for i in range(len(words) - 2):
5         trigrams.append((words[i], words[i+1], words[i+2]))
6     return trigrams
7
8
9 def is_english(words):
10    for word in words:
11        try:
12            word.encode(encoding='utf-8').decode('ascii')
13        except UnicodeDecodeError:
14            return False
15    return True
16
17
18
19 english_trigrams_rdd = articles_cleaned_rdd.flatMap(lambda x: (x['title'], x['text'])).flatMap(lambda x: trigram(x))\
20 .filter(lambda x: is_english(x))
21
22 english_trigrams = english_trigrams_rdd.collect()
23
✓ 16.7s                                         Python
```

در نهایت با استفاده از تابع is\_english کلمات انگلیسی را فیلتر می‌کنیم و سپس ۲۰ تا اول را بدست می‌آوریم:

```
1 english_trigrams_count_rdd = english_trigrams_rdd.map(lambda x: (x, 1))
2 english_trigrams_count = english_trigrams_count_rdd.reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], False).collect()
3 top_20 = english_trigrams_count[:20]
4 top_20
✓ 54.9s                                         Python

[(('lt', 'gt', 'lt'), 50),
 (('gt', 'lt', 'gt'), 49),
 (('http', 'en', 'org'), 16),
 ('(Wikipedia', 'The', 'http'), 12),
 (('The', 'http', 'en'), 12),
 (('Wikipedia', 'Wikipedia', 'The'), 11),
 (('formula', '1', 'formula'), 10),
 (('en', 'org', 'amp'), 10),
 (('amp', 'amp', 'amp'), 9),
 (('and', 'the', 'of'), 7),
 (('x', '1', '1'), 6),
 (((formula', 'formula', 'formula'), 6),
 (('1', 'formula', '2'), 6),
 (('1', 'x', '1'), 5),
 (('x', '1', 'x'), 5),
 (('of', 'the', 'and'), 5),
 (('the', 'of', 'the'), 5),
 (('a', 'x', '1'), 5),
 (('p', 'q', 'x'), 5),
 (('lt', 'gt', 'x'), 4))
```

## برای نمودار توزیع طول مقالات داریم:



و ۵ مقاله اول با بیشترین طول عبارت اند از:

```
document title: استان کرمانشاه
document length: 6603
document URL: https://fa.wikipedia.org/wiki?curid=2603
*****
document title: ساری
document length: 5606
document URL: https://fa.wikipedia.org/wiki?curid=4094
*****
document title: تاریخ استان
document length: 5402
document URL: https://fa.wikipedia.org/wiki?curid=439490
*****
document title: آنادیت
document length: 4703
document URL: https://fa.wikipedia.org/wiki?curid=4637750
*****
document title: الکترون
document length: 4025
document URL: https://fa.wikipedia.org/wiki?curid=5237
*****
```

## بخش چهارم:

برای محاسبه  $tf$  هر کلمه مراحل reduce و map زیر را به ترتیب اجرا می‌کنیم:

$doc \rightarrow ((doc - id, word), 1), doc \rightarrow ((doc - id, word), df) :$

```
1 # TODO: calculate document frequency for each word
2 word_df_rdd = articles_cleaned_rdd.flatMap(lambda x: [((x['id'], word), 1) for word in x['text'].split()])
3 # doc -> ((doc_id, word), 1)
4 word_df_reduced_rdd = word_df_rdd.reduceByKey(lambda x, y: x + y)
5 # doc -> ((doc_id, word), df)
6 word_df = word_df_reduced_rdd.collect()
✓ 33.7s
```

Python

سپس

```
1 # (word, (doc_id, df))
2 tf_rdd = word_df_reduced_rdd.map(lambda x: (x[0][1], (x[0][0], x[1])))
3 tf = tf_rdd.collect()
✓ 11.4s
```

Python

برای محاسبه  $idf$  ابتدا  $tf$  را انجام می‌دهیم:

```
1 word_freq_rdd = idf_rdd.map(lambda x: (x[0], x[1][2])).reduceByKey(lambda x, y: x + y)
2 word_freq = word_freq_rdd.collect()
✓ 25.8s
```

Python

سپس

```
1 # (word, (doc_id, df, 1))
2 idf_rdd = word_df_reduced_rdd.map(lambda x: (x[0][1], (x[0][0], x[1], 1)))
3 pre_idf = idf_rdd.collect()
✓ 11.8s
```

Python

$$idf_t = \log\left(1 + \frac{N}{n_t}\right)$$

```
1 word_freq_rdd = idf_rdd.map(lambda x: (x[0], x[1][2])).reduceByKey(lambda x, y: x + y)
2 word_freq = word_freq_rdd.collect()
✓ 25.8s
```

Python

```
1 idf_rdd = word_freq_rdd.map(lambda x: (x[0], math.log(1 + len(articles_cleaned) / x[1])))
2 idf = idf_rdd.collect()
3 idf[:5]
✓ 37.4s
```

Python

حال rdd‌های tf و idf را با هم join می‌کنیم و مقادیر tf و idf را در هم ضرب می‌کنیم:

```
1 tf_idf_rdd = tf_rdd.join(idf_rdd)
2 tf_idf = tf_idf_rdd.collect()
✓ 1m 34.7s
Python
```

```
1 tf_idf_result_rdd = tf_idf_rdd.map(lambda x: (x[1][0][0], (x[0], x[1][0][1], x[1][1], x[1][0][1]*x[1][1]))).sortByKey()
2 tf_idf_result = tf_idf_result_rdd.collect()
✓ 1m 40.2s
Python
```

```
1 # (doc_id, (word, tf, idf, tf_idf))
2 tf_idf_result[:2]
✓ 0.4s
Python
[('1007804', ('4.933119643747322 ,4.933119643747322 ,1 ,اکتبر' ,)),
 ('1007804', ('4.581489214294248 ,4.581489214294248 ,1 ,باب_الربع'))]
```

در نهایت یک بردار tf-idf به هر article اضافه می‌کنیم که مقادیر tf-idf کلمات آن است.

```
1 # TODO: add 'vector' key to articles_cleaned_rdd dictionary with the tf_idf dictionary
2 articles_tf_idf_vectors = tf_idf_result_rdd.map(lambda x: (x[0], {x[1][0]: x[1][3]})).reduceByKey(lambda x, y: {**x, **y}).collect()
3 articles_tf_idf_vectors = {x[0]: x[1] for x in articles_tf_idf_vectors}
4
5 def add_tf_idf_vector(article):
6     article['vector'] = articles_tf_idf_vectors.get(article['id'], [])
7     return article
8
9 articles_cleaned_rdd = articles_cleaned_rdd.map(lambda x: add_tf_idf_vector(x))
✓ 44.7s
Python
```

همچنین مقالاتی که دارای موضوعات داده شده هستند را بدست می‌آوریم:

```
1 # How many and what percentage of articles contain these words? [History, Politics, Medicine, Law, Economics, Engineering]
2 topics = ['سیاست', 'تاریخ', 'سیاست', 'پژوهش', 'قانون', 'اقتصاد', 'مهندسی']
3 articles_with_topic_rdd = tf_rdd.filter(lambda x: x[0] in topics)
4 articles_with_topic = articles_with_topic_rdd.collect()
Python
```

برای قسمت search، ابتدا ماتریس tf-idf مقالات را محاسبه می‌کنیم که ستون‌های آن تمام کلمات استخراج شده و ردیف‌هایش id مقالات است و هر درایه مقدار tf-idf کلمه متناظر در مقاله متناظر است.

```

1 docs = []
2 for topic in topics_dict.keys():
3     docs.extend(topics_dict[topic])
4 docs = set(docs)
5

1 articles_with_topics_rdd = articles_cleaned_rdd.filter(lambda x: x['id'] in docs)
2 articles_with_topics = articles_with_topics_rdd.collect()

1 words = word_df_reduced_rdd.map(lambda x: x[0][1]).collect()
2 words_tf_idf = {x: 0 for x in words}
3 def assign_tf_idf_to_words(article):
4     tmp = words_tf_idf.copy()
5     tmp.update(article['vector'])
6     article['vector'] = tmp
7     return (article['id'], article['vector'])

9 tf_idf_matrix_rdd = articles_with_topics_rdd.map(lambda x: assign_tf_idf_to_words(x))
10 tf_idf_matrix = tf_idf_matrix_rdd.collect()
11

✓ 52.4s

```

حال لیست بدستآمده را به دیتا فریم pandas تبدیل می‌کنیم:

```

1 import pandas as pd
2 tf_idf_matrix_df = pd.DataFrame(tf_idf_matrix, columns=['id', 'vector'])
3 tf_idf_matrix_df = tf_idf_matrix_df.set_index('id')
4 tf_idf_matrix_df = tf_idf_matrix_df['vector'].apply(pd.Series)
5 tf_idf_matrix_df = tf_idf_matrix_df.fillna(0)
6 tf_idf_matrix_df.head()

✓ 1.6s

```

id	نول	سالانه	ارتفاع	شرح	زمین	دم	کشور	نوشته	رهبری	باستان	...	گزش	کلیتوریس	هژمه	وینجستر	کوروف
2269	0.0	0.000000	0.000000	0.000000	14.141219	0.0	3.438098	0.000000	5.737575	0.000000	...	0.0	0.0	0.0	0.0	0.0
2603	0.0	25.628668	0.000000	5.164715	28.282439	0.0	147.838226	4.595333	0.000000	11.039347	...	0.0	0.0	0.0	0.0	0.0
3220	0.0	6.407167	0.000000	0.000000	4.713740	0.0	10.314295	4.595333	0.000000	16.559021	...	0.0	0.0	0.0	0.0	0.0
3345	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	9.190666	0.000000	5.519674	...	0.0	0.0	0.0	0.0	0.0
4094	0.0	0.000000	14.071269	5.164715	28.282439	0.0	79.076260	13.786000	5.737575	22.078694	...	0.0	0.0	0.0	0.0	0.0

5 rows x 3520 columns

برای اینکه یک کلمه‌ای را در article‌ها پیدا کافیست ردیف‌هایی از ماتریس که مقدار tf-idf ستون آن کلمه در آن از ۰ بیشتر است را پیدا و برای میزان مرتبط بودن مرتب کنیم. اما در بخش قبل باید کوئری داده شده را در تمام article‌ها یکی بررسی کنیم. اگر query بیشتر از یک کلمه بود می‌توانیم برای بررسی میزان شباهت از ضرب داخلی استفاده کنیم.  
برای مثال برای کلمه تاریخ داریم:

```

1 # search for articles with the word 'تاریخ' in tf_idf_matrix_df
2 related = tf_idf_matrix_df[tf_idf_matrix_df['تاریخ'] > 0]
3 # sort by tf_idf value
4 related = related.sort_values(by='تاریخ', ascending=False)
5 related

✓ 0.1s

```

id	نول	سالانه	ارتفاع	شرح	زمین	دم	کشور	نوشته	رهبری	باستان	...	گزش	کلیتوریس	هژمه	وینجستر	کوروف
439490	0.0	0.0	0.000000	5.164715	23.568699	0.00000	61.885769	22.976666	22.950302	662.360833	...	0.0	0.0	0.0	0.0	0.0
4094	0.0	0.0	14.071269	5.164715	28.282439	0.00000	79.076260	13.786000	5.737575	22.078694	...	0.0	0.0	0.0	0.0	0.0
4975637	0.0	0.0	0.000000	0.000000	0.000000	0.00000	6.876197	4.595333	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
4254	0.0	0.0	0.000000	25.823573	9.427480	24.43699	41.257179	50.548666	0.000000	11.039347	...	0.0	0.0	0.0	0.0	0.0
5634034	0.0	0.0	0.000000	0.000000	108.416016	0.00000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
672231	0.0	0.0	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
671507	0.0	0.0	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
671309	0.0	0.0	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
670875	0.0	0.0	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
6132156	0.0	0.0	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0

376 rows x 3520 columns