

سؤال ۱ :

(الف)

الگوریتمی که برای join کردن دو relation استفاده کردیم multi-step است. یعنی حداقل دو مرحله- map-reduce لازم است که در اینجا با دو مرحله انجام میدهیم. ابتدا هر ۴ relation را میخوانیم و join ها را به

$(R \text{ join } S) \text{ join } (T \text{ join } U)$ صورت مینویسیم.

R (r)

S (s)

T (t)

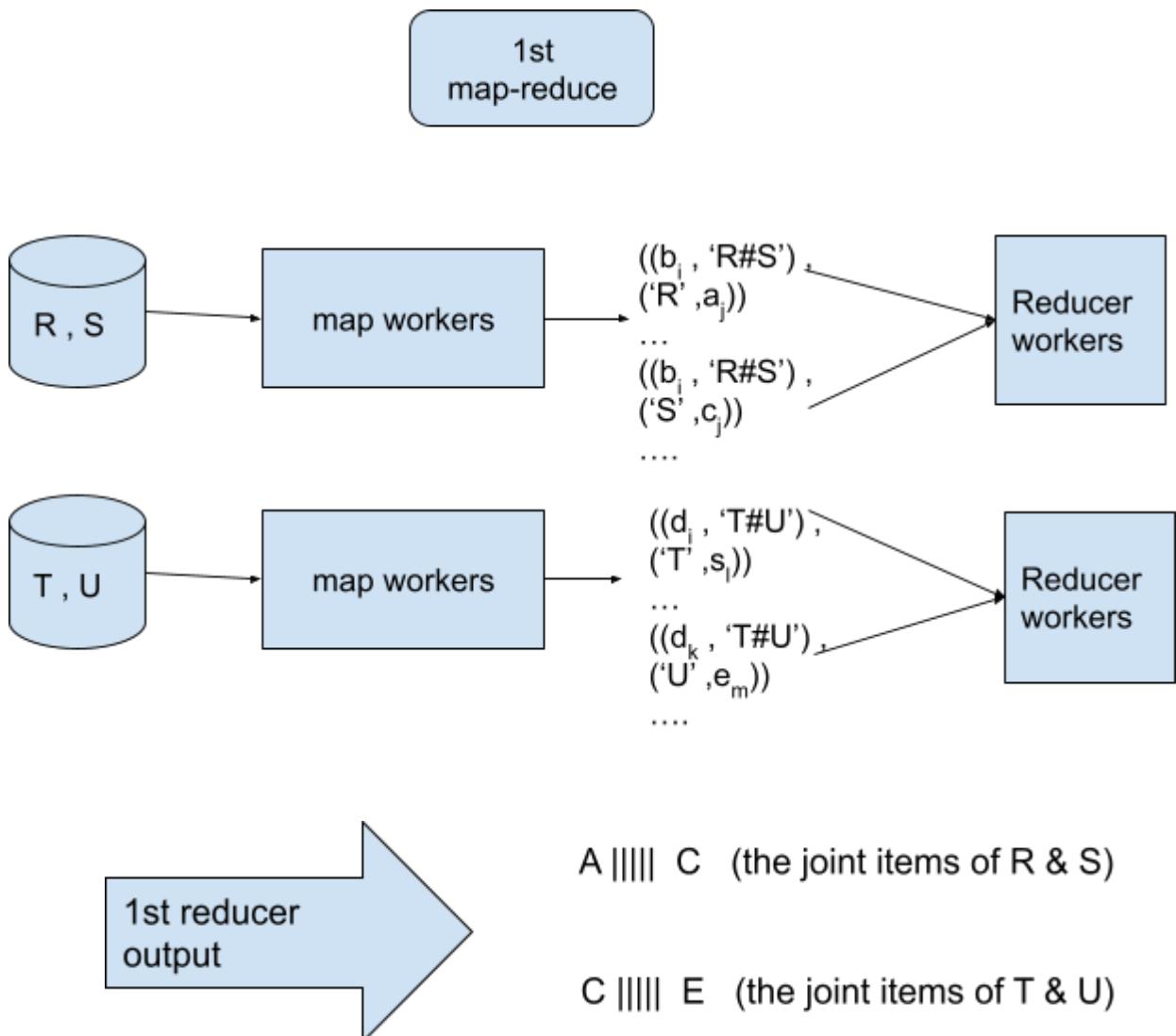
U (u)

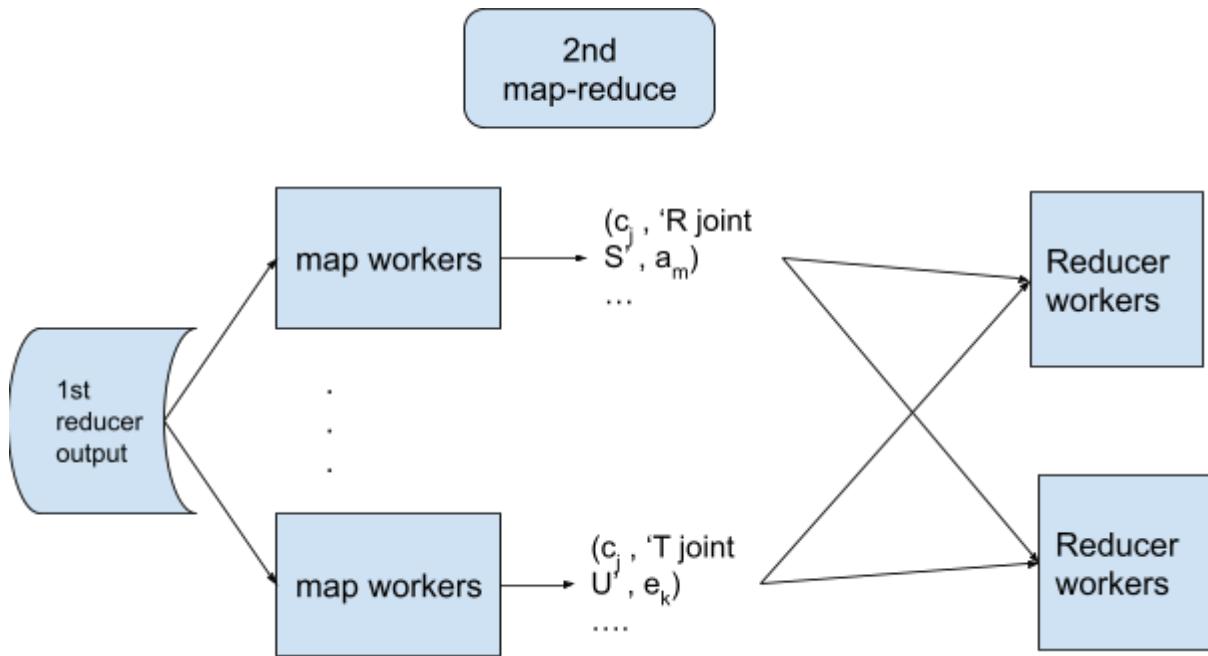
A ||| B

B |||| C

C ||| D

D ||| E





دقت کنید ای که در نظر گرفتیم تابع همانی است.

الگوریتمی که در بالا طراحی شد را از جنبه پارامترهای مختلف بررسی میکنیم :

- **Communication Cost (Important factor in Overall cost) :**

برای مرحله اول map-reduce به اندازه r . سایز ورودی ها خواندن داریم و به همان اندازه باید key

بفرستیم. پس در مرحله اول هزینه value

$$cost1 = 2 * (r + s + t + u)$$

برای مرحله دوم باید با در نظر گرفتن اینکه فقط p درصد از سایز حالتی که ماکزیمم join

داریم، ورودی های مرحله دوم را به دست بیاریم. سپس به همین تعداد باید دوباره key , value

بفرستیم. دقت کنید میتوانستیم هزینه save/write کردن در خروجی مرحله آخر را نیز در نظر

بگیریم ولی چون در حین درس، سیاق بر این نبوده است، اینجا نیز در نظر نمیگیریم (کم است).

$$cost2 = 2 * (p * r * s + p * t * u)$$

پس در مجموع هزینه کل این الگوریتم :

$$cost = O(r + s + t + u + p * r * s + p * t * u)$$

- **Replication rate (r):**

چون به ازای هر آیتم در دیتابیس دقیقاً یک key-value tuple در خروجی map ها ایجاد کردیم پس $r=1$ است.

- **Reducer size (q) :**

باید بررسی کنیم که در بدترین حالت، تعداد key هایی که به یک reducer میابند چند تاست و در واقع ماکریم سایز لیستی که در reducer درست میشود چقدر است.

در map-reduce اول باید بدترین حالتی که برای join شدن S , R پیش می آید را در نظر گرفته و حساب کنیم. در این join شدن، بدترین حالت این است که کلیدی (در سنتون مشترک، آیتمی) وجود داشته باشد که با ماکریم ترین حالت، بتواند join شود. پس (prs) در بدترین حالت از این تحلیل به دست می آید. به همین ترتیب برای U , T نیز در بدترین حالت خواهیم داشت (ptu) . و از آنجایی که هر دوی این join ها در map-reduce اول جای داده ایم و از آنجایی که تابع hash را همانی در نظر گرفته ایم، در بدترین حالت کل قسمت اول، چون کلید ها را به نحوی تعریف کردیم که مطمئنیم که key های S , R به reducer های مجازی از key هایی که از U , T تولید میشوند، میروند، باید بین prs, ptu ماکریم بگیریم .

$$q1 = \max(prs, ptu)$$

از طرفی در map-reduce دوم، باید همین تحلیل را فقط با سایز ورودی های متفاوت انجام داد. با همین تحلیل به دست می آید که در بدترین حالت به اندازه جمع سایز ورودی های map-reduce دوم، سایز لیست خواهیم داشت $(prs + ptu)$.

در نهایت برای در نظر گرفتن بدترین حالت باید ماکریم بین این ۲ را در نظر گرفت :

$$q2 = p^3 * rstu$$

- **Mapper & Reducer worker numbers:**

تعداد node های map , reduce معمولاً آزاد است و کافی است برای آن صرفاً یک حداقلی در نظر بگیریم. با توجه به آموزه های درس، مناسب است که برای load balancing به تعداد item های

دیتا، map در نظر بگیریم. (توجه کنید اینجا منظورمان هر map container است که میتواند چندتا از آنها روی یک ماشین باشد).

اول : map-reduce -

سایز ورودی اول $r+s+t+u$ است و ما هم به همین اندازه map node در نظر میگیریم. از آنجایی که را همانی در نظر گرفتیم، میتوانیم در بدترین حالت (حداکثر) به اندازه $prst + pst$ را برای reducer node ها در نظر بگیریم.

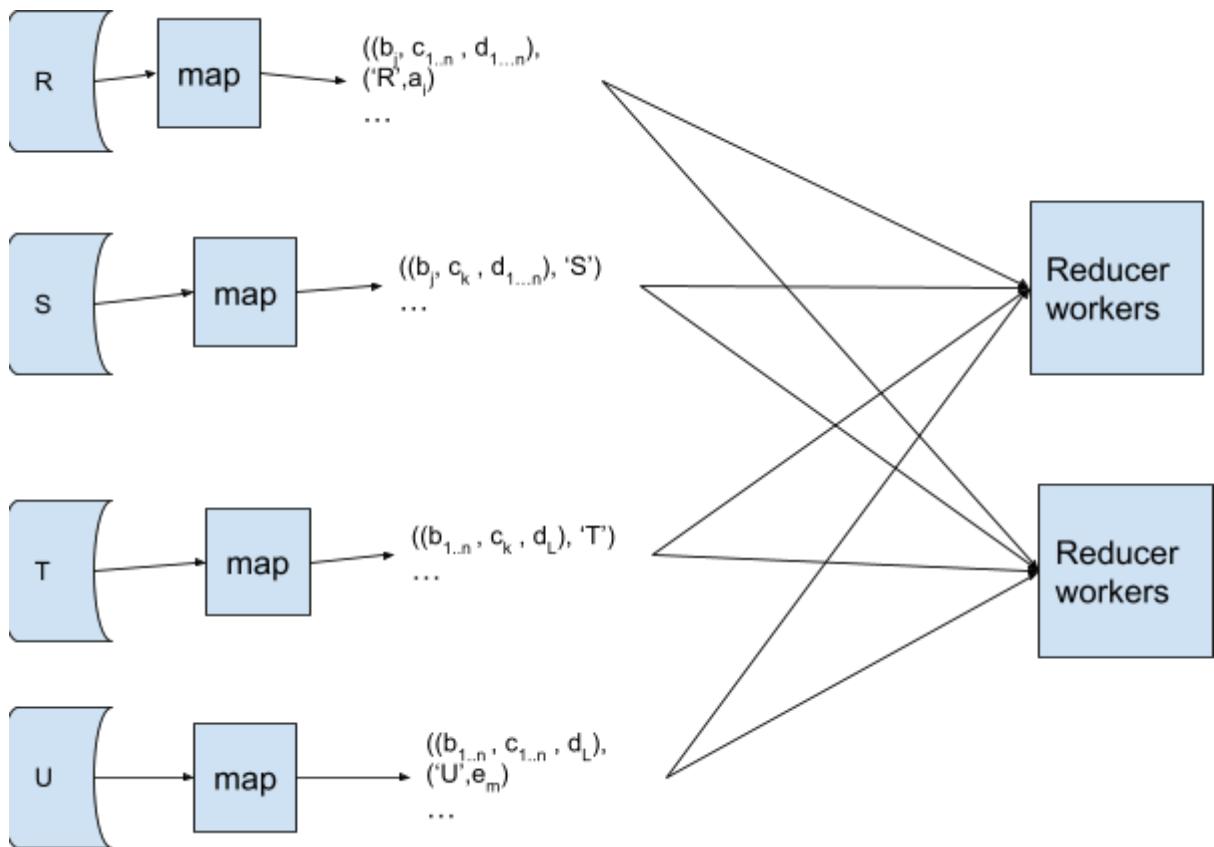
دوم : map-reduce -

سایز ورودی دوم $prst + ptu$ است. پس تعداد map ها را همین تعداد در نظر میگیریم. تعداد $p^3 * r * s * t * u$ خواهد بود.

(ب)

در این سوال بر اساس زیاد کردن key-value ها به امید آنکه بعدا مورد استفاده قرار بگیرد، عمل میکنیم. در واقع replication rate را افزایش میدهیم. الگوریتم پیشنهادی single step به شکل زیر است:





- کلید ها، یک ۳ تایی هستند که هر درایه آن، نشان از یکی از ستونهای مشترک برای join شدن می باشد (b, c,d)

- value ها در جدول هایی که فقط یک ستون از آنها جزو ستونهای مشترک نهایی هستند (U,R)، دو عنصری هستند و value ها در جدول هایی که هر دو ستون از آنها جزو ستونهای مشترک نهایی هستند (S,T)، تک عنصری هستند.

به مراحل زیر دقت کنید:

ا. وقتی جدول R را میخوانیم که شامل درایه های a_i , b_j است به ازای هر درایه، key-value هایی مشابه

زیر تولید میکنیم:

$((b_j,c1,d1),('R' , ai))$

....

$((b_j,c1,dn),('R' , ai))$

$((b_j,c2,d1),('R' , ai))$

....

((bj,c2,dn),('R' , ai))

.....

((bj,cn,dn),('R' , ai))

در واقع به ازای هر درایه که میخوانیم و یک b_i را داریم، به اندازه تعداد کل اندازه درایه های حالت موجود برای c و همینطور به اندازه تعداد کل درایه های حالت موجود برای d ، باید key-value تولید بکنیم. اگر فرض کنیم تعداد ماکریم همه درایه های موجود برای c (که مشترک بین S, T هستند)، Nc باشد و تعداد ماکریم همه درایه های موجود برای d (که مشترک بین U, T هستند) هم Nd باشد در این الگوریتم در مرحله اول به اندازه $O(N^2)$ تا key-value باید تولید کنیم.

۲. وقتی جدول S را میخوانیم که شامل درایه های b_j, c_k است به ازای هر درایه، b_j هایی مشابه زیر تولید میکنیم:

((bj,ck,d1),('S',))

....

((bj,ck,dn),('S',)))

در واقع به ازای هر درایه که میخوانیم یک c_k را داریم که هر دوی آنها در کلید مبایند. به اندازه تعداد کل اندازه درایه های حالت موجود برای d باید key-value تولید بکنیم. اگر فرض کنیم تعداد ماکریم همه درایه های موجود برای d (که مشترک بین U, T هستند) Nd باشد در این الگوریتم برای این جدول به اندازه $O(N)$ key-value باید تولید کنیم.

۳. وقتی جدول T را میخوانیم که شامل درایه های d_l, c_k است به ازای هر درایه، d_l هایی مشابه زیر تولید میکنیم:

((b1,ck,d1),('T',))

....

((bn,ck,d1),('T',))

((b1,ck,d2),('T',))

....
((bn,ck,d2),('T' ,))

.....
((bn,ck,dn),('T' ,))

در واقع به ازای هر درایه که میخوانیم یک c_k را داریم که c_k در کلید میاید. به اندازه تعداد کل اندازه درایه های حالت موجود برای d باید key-value تولید بکنیم. اگر فرض کنیم تعداد ماکریم همه درایه های موجود برای b (که مشترک بین R, S هستند) ، Nb باشد و تعداد ماکریم همه درایه های موجود برای d (که مشترک بین U, T هستند) هم Nd باشد در این الگوریتم در مرحله اول به اندازه $O(N) = r$ key-value باید تولید کنیم.

۴. وقتی جدول U را میخوانیم که شامل درایه های d_i, e_m است به ازای هر درایه، e_m هایی مشابه زیر تولید میکنیم:

((b1,c1,dl),('U' , em))

....
((b1,cn,dl),('U' , em))
((b2,c2,dl),('U' , em))

....
((bn,c2,dl),('U' , em))
.....
((bn,cn,dl),('U' , em))

در واقع به ازای هر درایه که میخوانیم و یک d_i, e_m را داریم، به اندازه تعداد کل اندازه درایه های حالت موجود برای c و همینطور به اندازه تعداد کل درایه های حالت موجود برای b ، باید key-value تولید بکنیم. اگر فرض کنیم تعداد ماکریم همه درایه های موجود برای b (که مشترک بین R, S هستند) ، Nb باشد و تعداد

ماکریم همه درایه های موجود برای C (که مشترک بین S , T هستند) هم باشد در این الگوریتم در مرحله اول به اندازه $O(N^2)$ key-value تا r باید تولید کنیم.

در نهایت در Reducer ها، در خروجی، join شده همه درایه ها بیرون میابد.

مزایا و معایب این روش:

- **Reducer size (q) :**

مزیت این روش در کاهش سایز در این روش

$$q = p^3 * r * s * t * u$$

سایز reducer در بدترین حالت خواهد بود که از سایز reducer اول در روش به multi-stage مراتب کمتر است و نیاز کمتری به حافظه در reducer هست.

ولی عیوبی که این روش دارد اینست که تعداد key-value ها را به مراتب بالا میبرد (در ارد N^2) و باعث افزایش communication cost میشود.

- **Communication Cost (Important factor in Overall cost) :**

$$O(r+s+t+u + rst + stu + rtu + urs)$$

که به مراتب از الگوریتم cost multi-stage بیشتر است.

- **Replication rate (r):**

در کمترین حالت r ، به اندازه $\min(r,s,t)$ است و در بیشترین حالت $\max(rs,st,tu)$ است.

- **Mapper & Reducer worker numbers:**

تعداد map ها را به اندازه ورودی ها در نظر میگیریم که مشابه multi-stage است. تعداد reducer ها هم در بدترین حالت میتواند به اندازه $p^3 * r * s * t * u$ خواهد بود که سایز جوین شده همه node جدولهاست.

سوال ۲:

(۱)

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes
0	Color	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore	1000
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom	40000
2	Color	Sam Mendes	602.0	148.0	0.0	161.0	Rory Kinnear	11000
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Bale	27000
4	NaN	Doug Walker	NaN	NaN	131.0	NaN	Rob Walker	131

5 rows × 28 columns

(۲)

- تعداد nan values در ستونها مختلف

```
number of all null values in df 2698
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
dtype: int64
```

:nan values درصد های -

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

director_name	2.06	number of all null values in df 1678
num_critic_for_reviews	0.99	director_name 104
gross	17.53	num_critic_for_reviews 50
genres	0.00	gross 884
actor_1_name	0.14	genres 0
movie_title	0.00	actor_1_name 7
num_voted_users	0.00	movie_title 0
num_user_for_reviews	0.42	num_voted_users 0
language	0.24	num_user_for_reviews 21
budget	9.76	language 12
title_year	2.14	budget 492
imdb_score	0.00	title_year 108
movie_facebook_likes	0.00	imdb_score 0
dtype: float64		movie_facebook_likes 0
		dtype: int64

- همانطور که میبینیم، دو ستونی که درصد Nan های آن بیشتر از ۵ درصد است طبق جدول بالا، دو

ستون budget , gross , budget به آنها است. بعد از حذف سطرهایی که ستون مربوط به

است، تعداد Nan ها و درصد آنها :

director_name	0.00	number of all null values in df 7
num_critic_for_reviews	0.03	director_name 0
gross	0.00	num_critic_for_reviews 1
genres	0.00	gross 0
actor_1_name	0.08	genres 0
movie_title	0.00	actor_1_name 3
num_voted_users	0.00	movie_title 0
num_user_for_reviews	0.00	num_voted_users 0
language	0.08	num_user_for_reviews 0
budget	0.00	language 3
title_year	0.00	budget 0
imdb_score	0.00	title_year 0
movie_facebook_likes	0.00	imdb_score 0
dtype: float64		movie_facebook_likes 0
		dtype: int64

- بعد از پر کردن ستون language با english ، تعداد و درصد Nan ها:

director_name	0.00	number of all null values in df 4
num_critic_for_reviews	0.03	director_name 0
gross	0.00	num_critic_for_reviews 1
genres	0.00	gross 0
actor_1_name	0.08	genres 0
movie_title	0.00	actor_1_name 3
num_voted_users	0.00	movie_title 0
num_user_for_reviews	0.00	num_voted_users 0
language	0.00	num_user_for_reviews 0
budget	0.00	language 0
title_year	0.00	budget 0
imdb_score	0.00	title_year 0
movie_facebook_likes	0.00	imdb_score 0
		movie_facebook_likes 0
		dtype: int64

- علت وجود Nan در دیتافریم اولیه میتواند دلایلی مختلف باشد. از جمله:

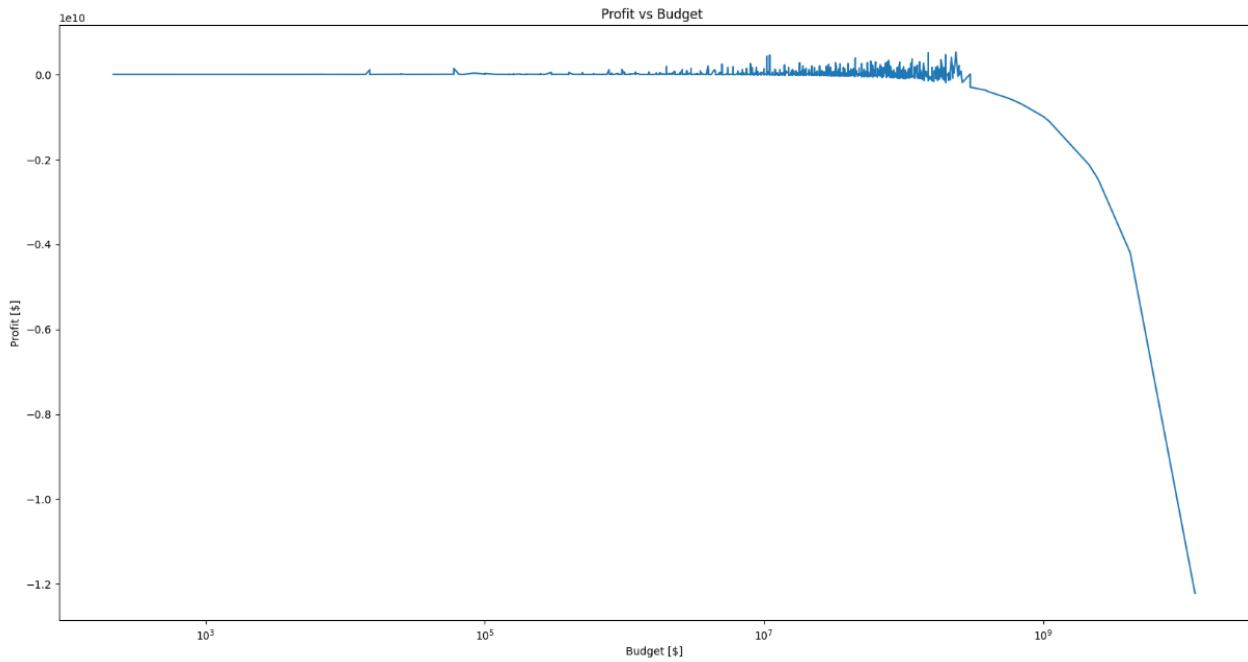
۱. در دسترس نبودن و یا محرومانه بودن دیتای ستونهایی مثل gross , budget باشد.
۲. در بعضی موارد اطلاعات وارد شده به دیتافریم از یک فایل می آید و به علت ناتوانی در decode کردن بعضی از stringها، در پایتون و دیتافریم به صورت Nan ظاهر میشود.
۳. وارد نشدن بعضی اطلاعات فیلم مثل actor_1_name به علت وارد شدن دستی اطلاعات فیلم ها توسط اپراتور های وارد کننده.

(۴)

اضافه شدن ستون : profit

profit
523505847.0
9404152.0
-44925825.0
198130642.0
-190641321.0

- نمودار gross بر حسب profit :



- در مورد این نمودار به چند مورد توجه کنید :

۱. نمودار در بعد اول آن به صورت لگاریتمی کشیده شده است (به علت گستردگی زیاد در مقدار budget ها).
۲. با تحلیل آن متوجه میشویم با افزایش درآمد فیلم ها تا حدود عدد $2e8$ دلار، افزایش بودجه فیلمها میتواند توجیه اقتصادی داشته باشد و منجر به سود بشود ولی با افزایش عدد بعد ازین عدد، سود منفی و منفی تر میشود و توجیه اقتصادی ندارد.
۳. با افزایش عدد از 0 تا حدود $1e6$ دلار، تقریبا سود اضافه نمیشود و در اردر یکسانی است ولی بعد ازین عدد تا عدد $2e8$ ، با افزایش بودجه سود هم میتواند افزایش داشته باشد.

- ۱۰ فیلم پر سود که عدد های سود و سایر اطلاعات آن در کد آمده است و در اینجا فقط اسم آنها را

نمایش میدهیم :

	director_name	num_critic_for_reviews	gross	genres	actor_1_name	movie_title	num_voted_users	num_user_for_reviews	language	budget	title_year	imdb_score
0	James Cameron	723.0	760505847.0	Action Adventure Fantasy Sci-Fi	CCH Pounder	Avatar	886204	3054.0	English	237000000.0	2009.0	7.9
29	Colin Trevorrow	644.0	652177271.0	Action Adventure Sci-Fi Thriller	Bryce Dallas Howard	Jurassic World	418214	1290.0	English	150000000.0	2015.0	7.0
26	James Cameron	315.0	658672302.0	Drama Romance	Leonardo DiCaprio	Titanic	793059	2528.0	English	200000000.0	1997.0	7.7
3024	George Lucas	282.0	460935665.0	Action Adventure Fantasy Sci-Fi	Harrison Ford	Star Wars: Episode IV - A New Hope	911097	1470.0	English	11000000.0	1977.0	8.7
3080	Steven Spielberg	215.0	434949459.0	Family Sci-Fi	Henry Thomas	E.T. the Extra-Terrestrial	281842	515.0	English	10500000.0	1982.0	7.9
17	Joss Whedon	703.0	623279547.0	Action Adventure Sci-Fi	Chris Hemsworth	The Avengers	995415	1722.0	English	220000000.0	2012.0	8.1
509	Roger Allers	186.0	422783777.0	Adventure Animation Drama Family Musical	Matthew Broderick	The Lion King	644348	656.0	English	45000000.0	1994.0	8.5
240	George Lucas	320.0	474544677.0	Action Adventure Fantasy Sci-Fi	Natalie Portman	Star Wars: Episode I - The Phantom Menace	534658	3597.0	English	115000000.0	1999.0	6.5
66	Christopher Nolan	645.0	533316061.0	Action Crime Drama Thriller	Christian Bale	The Dark Knight	1676169	4667.0	English	185000000.0	2008.0	9.0
439	Gary Ross	673.0	407999255.0	Adventure Drama Sci-Fi Thriller	Jennifer Lawrence	The Hunger Games	701607	1959.0	English	78000000.0	2012.0	7.3

(۴)

- : زانرهای مختلف

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

- تعداد فیلمهای در هر ژانر و میانگین درآمد آنها :

```
Genre name: Action
number of films: 962
average gross of films: 76584686.26611227

Genre name: Adventure
number of films: 787
average gross of films: 99223012.94790342

Genre name: Fantasy
number of films: 514
average gross of films: 86536511.92023346

Genre name: Sci-Fi
number of films: 497
average gross of films: 82580549.6861167

Genre name: Thriller
number of films: 1118
average gross of films: 48528709.706618965

Genre name: Romance
number of films: 879
average gross of films: 44139073.73606371

Genre name: Animation
number of films: 199
average gross of films: 108362167.35678393

Genre name: Comedy
number of films: 1503
average gross of films: 50102225.2401863

Genre name: Family
number of films: 451
average gross of films: 91560013.20399113
```

Genre name: Musical
number of films: 103
average gross of films: 55189001.21359223

Genre name: Mystery
number of films: 384
average gross of films: 46397270.848958336

Genre name: Western
number of films: 60
average gross of films: 43128855.78333333

Genre name: Drama
number of films: 1944
average gross of films: 36878664.63220165

Genre name: History
number of films: 154
average gross of films: 36565730.597402595

Genre name: Sport
number of films: 151
average gross of films: 43473208.98675497

Genre name: Crime
number of films: 715
average gross of films: 38779521.08111888

Genre name: Horror
number of films: 391
average gross of films: 34410688.969309464

Genre name: War
number of films: 160
average gross of films: 40029805.29375

```
Genre name: Biography
number of films: 243
average gross of films: 35645449.20576131
```

```
Genre name: Music
number of films: 159
average gross of films: 37658404.1509434
```

```
Genre name: Documentary
number of films: 67
average gross of films: 13384360.149253732
```

```
Genre name: Short
number of films: 2
average gross of films: 3926267.0
```

```
Genre name: Film-Noir
number of films: 1
average gross of films: 7927.0
```

(۶)

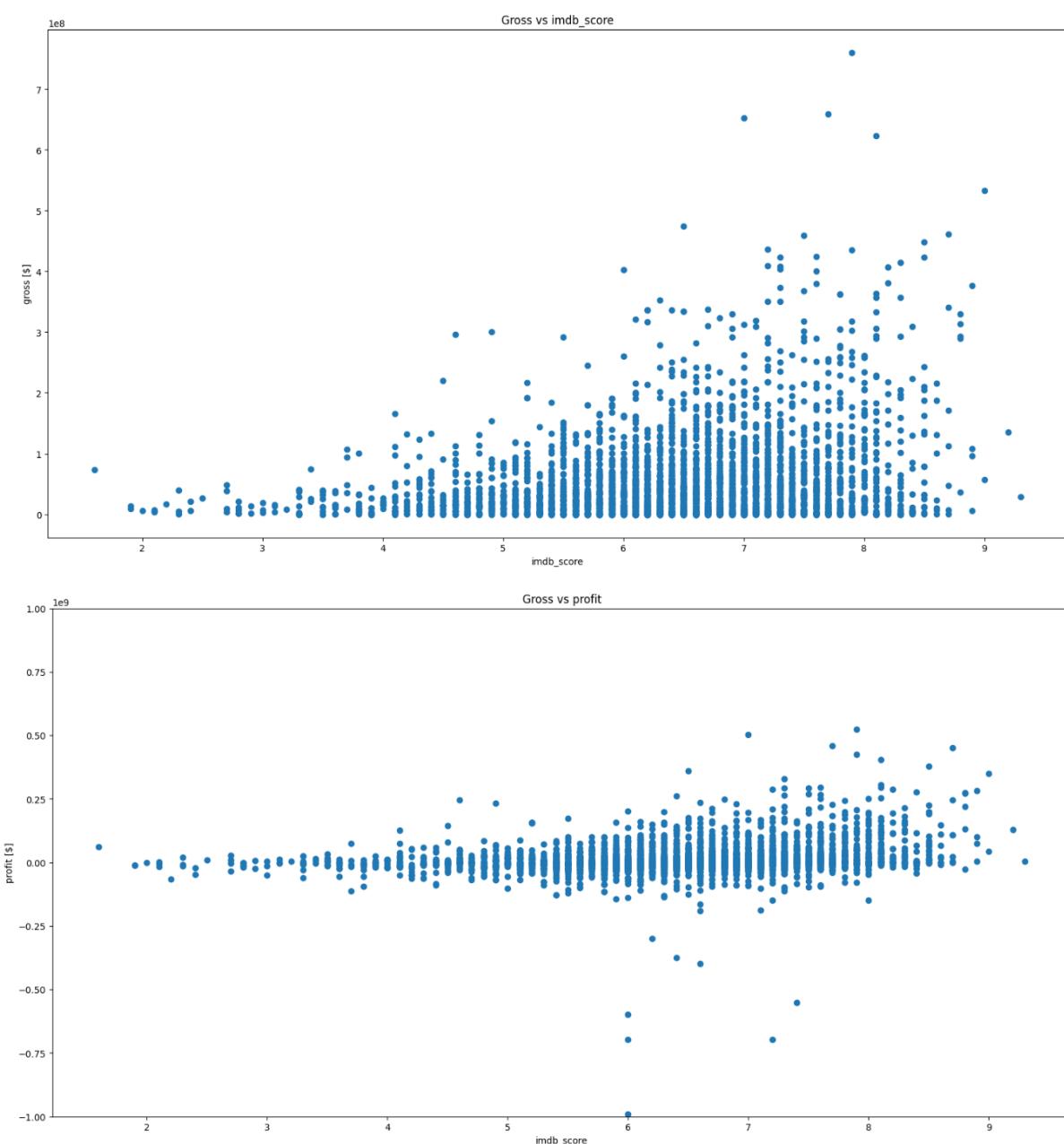
این لیست همه ژانرهای موجود در دیتابیس است که دیتای فیلمها در شرایط گفته شده صدق میکند و هر فیلم ممکن است از ژانرهای مختلف باشد :

```
.. ['Action',
 'Adventure',
 'Fantasy',
 'Sci-Fi',
 'Thriller',
 'Animation',
 'Comedy',
 'Family',
 'Musical',
 'Romance',
 'Mystery',
 'Drama',
 'Western',
 'Crime',
 'Biography',
 'History',
 'War',
 'Documentary',
 'Horror',
 'Sport',
 'Music']
```

اگر بخواهیم بدانیم هر فیلم که در شرایط گفته شده صدق میکند، چه خصوصیاتی دارد، باید به سراغ notebook برویم.

(۷)

نمودار میزان درآمد بر اساس نظر مردم (که میتواند نمایندگی آن را بکند) و همینطور نمودار میزان سود بر اساس imdb_score به شکل های زیر است :



```

dropped_df['imdb_score'].corr(dropped_df['profit'])
✓ 0.1s
0.035855880169839774

```

با مشاهده نمودارها و اعداد به دست آمده از correlation ، به نظر می آید لزوما این دو فاکتور با هم در ارتباط نیستند. یعنی با افزایش imdb_score ، اعداد سود و فروش هیچکدام لزوما افزایش نداشته اند و هم اعداد سود هم سطح و هم اعداد سود پایین تر از imdb_score های پایین تر نیز وجود دارند و این روند در همه جای بازه imdb_score وجود دارد.

دلایل این اتفاق ممکن است به شرح زیر باشد :

۱. همه افرادی که در imdb_score دادن، نقش داشته اند، لزوما فیلم را نخریده اند و در درآمد موثر نبوده اند.

۲. همه افرادی که فیلم را خریده اند، لزوما imdb_score خود را ثبت نکرده اند و یا با تاخیر زیاد ثبت کرده اند که منجر به امتیاز دهنی ناعادلانه و بایاس دار ممکن است شده باشد.

۳. با دقت در نمودار سود میتوان فهمید که با افزایش imdb_score بعد از امتیاز ۵، میانگین سود فیلم ها به طور متوسط نیز نسبتا افزایش داشته است. در نتیجه، فرضیه ای ممکن است در ذهنمان شکل بگیرد که ممکن است این دلیل یکی از دلایل موثر باشد و رابطه causality برقرار باشد، یا اینکه صرفا یک هم زمانی و correlation اتفاق افتاده است که تحت تاثیر یک عامل پنهان سومی است که اثبات یا رد این فرضیه نیازمند آزمایش است که در تمرین خواسته نشده است.

سوال ۳ :

Section 1:

در این بخش صرفا کد های دانلود کردن دیتابس و ساختن spark session را ران میکنیم. در ضمن تعداد کل article ها را چون بعدا به آن نیاز داریم، در همینجا محاسبه کردیم(num_articles) و آن را broadcast.(num_articles_b).کردیم

Section 2:

در این بخش، به اعمال مقدماتی روی متن ها می پردازیم.

۱) در ابتدا فایل های text خوانده شده را با تابع `json.loads` از کتابخانه داخلی پایتون، تبدیل به دیکشنری های `parse` شده میکنیم، در واقع بر روی هر المان `rdd` اولیه این تابع را اجرا میکنیم. این عملیات را با `map` انجام دادیم.

۲) سپس برای تمیزسازی متن، در مرحله اولیه نیاز است کاراکترهای بی معنی در تسك نهایی را حذف کنیم. کاراکترهای `punctuation` و یک سری کاراکتر های حروف نگارشی در فارسی را از `title` ها و `text` ها حذف کردیم. در نهایت با متده `strip()` در پایتون که بر روی متنها اعمال میشود، فضاهای خالی قبل و بعد همه `string` ها را پاک کردیم. در نهایت دیکشنری ای مثل ساختار دیکشنری های اولیه، خروجی میدهیم که تمیز شده اند. دقت کنید که در این مرحله، لارم نبود عملیات تمیز سازی را به صورت اعمال روی هر کاراکتر انجام بدھیم و نیازی به تبدیل به `key-value` نبود. این عملیات را با `map` انجام دادیم.

۳) در این بخش باید یک `rdd` شامل کلمات هر متن به دست بیاوریم. برای این کار، به این علت که ما در نهایت، به اندیس گذاری همه `article` ها نیازمندیم (به عنوان شناسه و ترتیب مقاله ها)، در همان ابتدا با استفاده از متده `zipWithIndex`، همه دیکشنری مقالات را با شماره آن ها به صورت `:indexed_article_rdd` درآوردهیم `key-value`

```
## (key = doc_index , value = cleaned_dic)
```

در همین مرحله چون در مراحل بعدی، در بعضی جاها صرفا از `url`، `title` متن منتظر هر متن استفاده میکردیم، برای کاهش سایز نوشتمن `key-value` ها، یک `rdd` دیگری نیز که فقط شامل این دو باشد و اندیس گذاری نیز شده باشد، نیاز داشتیم(`indexed_title_and_url_article_rdd`). در بعضی اوقات نیز فقط به متن مورد مقاله به صورتی که اندیس گذاری شده باشد نیاز داشتیم(`indexed_text_article_rdd`) و همه این ۳ تا `rdd` را همینجا ساختیم.

```
## (key = doc_index , value = cleaned_dic (title&url) )
## (key = doc_index , value = cleaned_dic (text) )
```

سپس متن مقالات موردنظر را با یک `map`، جداسازی کردیم. در نهایت `words_rdd` را خروجی دادیم.

۴) در این مرحله نیاز داریم تعداد تکرارهای هر کلمه در کل متن ها را به صورت key-value داشته باشیم. که در ابتدا (word,1) را از هر کلمه تولید کردیم و سپس با flatMap و با استفاده از reduceByKey (word,1) را از هر کلمه تولید کردیم.

۵) در این مرحله نیاز داشتیم تا ۱۰۰ کلمه پر تکرار متن را به دست بیاوریم و از بین آنها انتخاب کنیم که کدام ها stopwords ها هستند. صرفا از خروجی مرحله قبل و با استفاده از متدهای top و با اعمال بر روی تعداد تکرارها، ۱۰۰ تای اول را خروجی گرفتیم. بخشی از خروجی به شکل زیر است :

```
[('4859308', 'در'),
 ('4518738', 'و'),
 ('3162958', 'به'),
 ('2815381', 'از'),
 ('2181347', 'که'),
 ('1481974', 'است'),
 ('1418963', 'این'),
 ('1418066', 'را'),
 ('1137311', 'با'),
 ('960745', 'یک'),
 ('772780', 'سال'),
 ('729203', 'آن'),
 ('605514', 'برای'),
 ('528085', 'بود'),
 ('525902', 'شد'),
 ('474496', 'او'),
 ('399936', 'کرد'),
 ('382897', 'دارد'),
 ('375350', 'شدهاست'),
 ('366393', 'خود'),
 ('354456', 'تا'),
 ('344367', 'بر'),
 ('343100', 'یا'),
 ('336521', 'شده'),
 ('320442', 'ایران'),
 ('317373', 'واقع'),
 ('303084', 'میشود'),
 ('271342', 'قرار'),
 ('261968', 'فوتبال'),
 ('259079', 'نیز'),
 ('249495', 'نام'),
 ('246509', 'عنوان'),
 ('235920', 'نفر'),
 ('223708', 'دو'),
 ('218496', 'وی'),
 ('211883', 'جمعیت'),
 ('205195', 'پس'),
 ('203557', 'آنها')]
```

همانطور که میبینید در بین آنها همه نوع کلمه ای مشاهده میشود و هم حروف اضافه موجود است، هم اسم های خاص و هم فعل و حروف ربطی. از بین آنها با دقت، انتخاب کردیم که stopwords ها کدام ازینها باشند، چون کلماتی مثل (ایران) نیز در بین پر تکرار ها بودند ولی به خاطر sample ای که از ویکیپدیا گرفته ایم، پر تکرار شده اند و منطقی نیست که جزو stopword ها باشند.

۶) در این مرحله stopword ها را نوشتیم و broadcast کردیم :

```
stopwords = ['خود', 'لدهاست', 'کرد', 'او', 'شد', 'برای', 'آن', 'این', 'را', 'است', 'که', 'از', 'به', 'در', 'با', 'و', 'ادیگر', 'میکند', 'همجین', 'ایکی', 'استفاده', 'آنها', 'توسط', 'بس', 'وی', 'دو', 'بنز', 'مشود', 'لده', 'یا', 'بر', 'اما', 'کند', 'من', 'بودهاست', 'داده', 'باشد', 'بین', 'هستند', 'زیر', 'روی', 'هر', 'مورد', 'هم', 'متواتان', 'اما', '[بعد', 'بسیار', 'میباشد', 'شود', 'انجام', 'داده', 'کردهاست']
```

```
stopwords_b = sc.broadcast(stopwords)
```

۷ و ۸) در این مرحله از ما خواسته شده که stopword ها را از متنها حذف کنیم ولی به علت مطابق بودن عملیات این بخش با عملیات بخش بعد که خواسته شده کم تکرارترین uncommon_word ها را پیدا کنیم و حذف کنیم، ابتدا عملیات پیدا کردن uncommon word ها را انجام میدهیم و سپس همه آنها را با هم حذف میکنیم.

با توجه به اینکه زمان حذف کردن همه word uncommon ها با شرط تکرار کمتر از ۱۰۰، خیلی خیلی طول میکشید، تعداد پیدا کردن آن ها را به ۱۰۰ تا از کم پرتکرارترین ها تقلیل دادیم. برای این کار از استفاده کردیم و با استفاده از متند top و با استفاده از مقایسه words_count_rdd ها با هم به ۱۰۰ تا از آنها دست پیدا کردیم و آنها را در یک لیست پایتونی ریختیم و broadcast کردیم.

```
[], 'میداندمدحشگان', 'فاتیان', 'زیستیانزیمهایا', 'ZaGR', 'خودوده', 'P4', 'مولکولهای', 'دارندقلل', 'اسطورهمجموعه', 'Monaco', 'نام', 'مخصوصجهار', 'الگوریتمیمیخطهای', 'Ebbinghaus', 'میکنداقتصادمووالو', 'OMDB', 'Cantilene', 'مالدوراللیموناللاینیت', 'سریفرمیونیان', 'نامهولادیمیر', 'تارعومما', 'کمینهگراترین', 'رفتهاستبیلیبوردها', 'ساجلوندابل', 'میباشدشاه', 'میگفتندابراهیم', 'Bathybagrus', 'سقوطها', 'زدعلياصغر', 'منویسدبوان', 'lapponica', 'Европейская', 'دنبر', 'نمیگ', 'برمیانگیزندنودولس', 'کردریشتفربیا', 'Lisp', 'واقفی', 'جغرافیابیتعریفهای', 'زاپشیرها', 'نطا میهنگ'
```

که عمدتاً اسمی خیلی خاص یا کلمات خاص هستند.

حالا برای حذف کلمات از این دو دسته به ترتیب زیر عمل کردیم : ابتدا با استفاده از `split()` که متن های مقاله ها را همراه با `index` دارد، کلمات را `indexed_text_article_rdd` کردن روی کلمات هر متن، `position` هر کلمه را در آن مقاله خاص، به `value` اضافه کردیم تا `enumerate` متن مرتب شده را دوباره بازسازی کنیم. بعدها بتوانیم بر اساس این `word_position_in_doc` سپس با استفاده از `filter()` و چک کردن اینکه هیچکدام از کلمات `stopwords`، `uncommon_words` از آن رد نشوند، به مرحله بعد میرویم. در مرحلهنهایی نیاز است که کلمات هر `doc` را به صورت یکجا گردآوری کنیم و با استفاده از `position_in_doc` آن ها را `sort` کنیم و به هم بچسبانیم.

همه مراحل به شکل زیر اند :

```
# output 0 & 1
##(key = word , value= (doc_index , word_postion_in_doc) )
# output 2
```

```

##(key = doc_index , value= (word ,word_postion_in_doc))
# output 3 & 4
##(key = doc_index , value=[(word1 ,word1_postion_in_doc) ,..., (wordn
,wordn_postion_in_doc)])
# output 5
##[(key = doc_index1 , "text1") , (key = doc_index2 , "text2") ,...]

```

در نهایت برای اینکه همه doc ها را به همان صورت اولیه و ترتیب اولیه و تمیز شده داشته باشیم نیازمند این بودیم که خروجی مرحله قبل را با title & url indexed_title_and_url_article_rdd که سپس بر اساس doc_index انجام دهیم.

```

# output 0
##[(key = doc_index1 ,(title&url_dic1 , "text1") , (key = doc_index2 ,,(article_dic2 , "text2")) ,...]
# output 1
##[(key = doc_index1 , article_without_stopwords_dic1 ) ,(key = doc_index2 , article_dic2 ) ,...]

```

در نهایت برای اینکه خروجی ها به شکل مورد نظر سوال به دست بیاید باید doc_index ها را حذف میکردیم.

```

# output |
##[article_cleaned_dic1 ,article_cleaned_dic2 ,...]

```

برای نمونه خروجی اول که تمیز شده است :

```

[{'url': 'https://fa.wikipedia.org/wiki?curid=2',
'title': 'صفحه اصلی',
'text': 'امروز میلادی برابر هجری خورشیدی صفحه اصلی stylescsgt src=ltemplatestyles UTC – NOEDITSECTION'}]

```


همچنین با این شرط که طول آن کلمات حداقل باید ۳ باشد. شکل خروجی تا اینجا به این شکل است :

```
# output 2
##[(key = just_english_greater_than_3_length_word , word_length ) ,(key = just_english_greater_than_3_length_word , word_length ) ,...]
```

بعد از این مرحله باید از کلماتی که بزرگتر مساوی ۳ تا حرف انگلیسی دارند، به اندازه ای که می شود، ()**reduceByKey** تولید کرد و سپس تعداد همه tri-gram ها را در هرجا که تولید شده اند با شمرد. در نهایت ۲۰ tri-gram با بیشترین تکرار انگلیسی را به دست آورдیم :

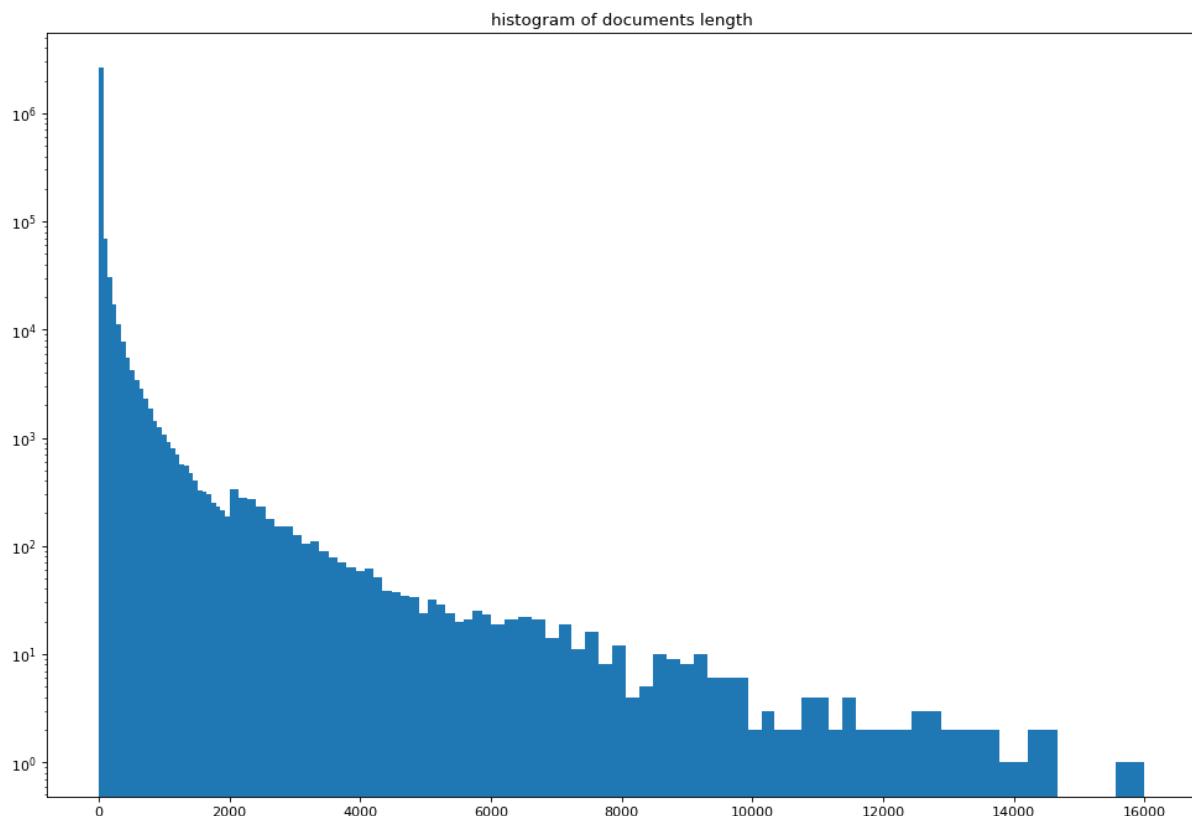
```
[('for', 47384),
 ('orm', 42877),
 ('ula', 41978),
 ('mul', 40167),
 ('rmu', 39420),
 ('ion', 30719),
 ('tio', 22093),
 ('ing', 20658),
 ('and', 18895),
 ('ati', 18649),
 ('ent', 16663),
 ('ter', 16197),
 ('tri', 15503),
 ('the', 13957),
 ('edi', 13935),
 ('dia', 13851),
 ('ist', 13635),
 ('la1', 12981),
 ('nde', 12151),
 ('all', 12012)]
```

(۳)

در این سوال از ما خواسته شده که histogram طول متنها را بکشیم. به دست آوردن طول متنها ساده است و کافی است تعداد کلمات یک متن را با یک map بگیریم. دقت کنید چون عملیات گرفتن طول یک متن در پایتون ساده می باشد و پردازش سبکی محسوب میشود، لزومی ندارد آن را دوباره به شکل map-reduce در

بیاوریم. البته شکل mpa-reduce ای آن را در قسمتهای مختلف بارها انجام داده ایم و لزومی به تکرار نیست.

هیستوگرام طول document ها با bin ۱۰۰ به شکل زیر است: (انتخاب bin ها به نحوی بوده است که در جاهایی که تراکم بیشتری داشته ایم تعداد ستونهای نیز تعییه کرده ایم تا نشانگر رزولوشن بالاتری نیز باشد).



همانطور که میبینیم، بیشتر طول داکیومنت ها در بازه بین ۰ تا ۳۰۰۰ کاراکتر هستند (نمودار عمودی به صورت لگاریتمی کشیده شده است).

- مقالات با بیشترین طول به شکل زیر آمده اند:

```
documents_length_KV_sorted[0:5]

[{'title': 'تادراشاه',
 'url': 'https://fa.wikipedia.org/wiki?curid=10787',
 'num_words': 15807},
 {'title': 'نبرد اسماولنسک',
 'url': 'https://fa.wikipedia.org/wiki?curid=2322946',
 'num_words': 14613},
 {'title': 'ایران',
 'url': 'https://fa.wikipedia.org/wiki?curid=163930',
 'num_words': 14417},
 {'title': 'نبرد مسکو',
 'url': 'https://fa.wikipedia.org/wiki?curid=1501875',
 'num_words': 13993},
 {'title': 'حافظ',
 'url': 'https://fa.wikipedia.org/wiki?curid=44114',
 'num_words': 13724}]
```

(۱۴)

در این قسمت از ما خواسته شده که کلمات topic های مورد نظر را در همه متن ها پیدا کنیم و گزارش کنیم هر کلمه در چند doc موجود است :

برای این کار، با شروع از indexed_article_words_rdd ، باید ابتدا کلمات تکراری داخل یک doc را با **reduceByKey()** حذف کنیم و سپس به ازای هر کلمه در خروجی که به ازای هر doc قطعاً از هم دیگر مجاز است، key-value هایی به شکل (word, 1) درست میکنیم که نشانگر این است که آن کلمه در آن doc وجود داشته است. سپس همه کلماتی که در topic ها وجود داشته اند (و فقط صرفاً در کوچکی و بزرگی حرفشان متفاوت بوده اند) را **filter()** میکنیم و سپس به ازای هر کلمه topic، تعداد آنها را با **reduceByKey()** میشمریم.

نتایج :

```

Medicine is : 0.0021462374668674592 %
Politics is : 0.0010015774845381475 %
LAW is : 0.00010731187334337295 %
law is : 0.0013950543534638484 %
history is : 0.0012519718556726846 %
HISTORY is : 7.154124889558197e-05 %
Engineering is : 0.00400630993815259 %
Law is : 0.0034339799469879344 %
Economics is : 0.0013235131045682665 %
economics is : 0.0004292474933734918 %
medicine is : 0.0005365593667168648 %
History is : 0.005723299911646558 %
politics is : 0.0002861649955823279 %
engineering is : 0.0008227243622991926 %

```

```

num_articles : 2795590

[('Medicine', 60),
 ('Politics', 28),
 ('LAW', 3),
 ('law', 39),
 ('history', 35),
 ('HISTORY', 2),
 ('Engineering', 112),
 ('Law', 96),
 ('Economics', 37),
 ('HISTORY', 2),
 ('economics', 12),
 ('medicine', 15),
 ('History', 160),
 ('politics', 8),
 ('engineering', 23)]

```

که نتایج طبیعی و منطقی ای می باشد. دقت کنید در شکل اول عددی که جلوی هر topic آمده است، تعداد مقالاتی هست که این کلمه در آنها وجود داشته است.

Section 4:

1) tf-idf:

- برای به دست آوردن term-frequency از هر کلمه در هر متنی، نیازمند شمردن تعداد تکرار هر کلمه در متن متناظر خودش هستیم. در نتیجه نیاز داریم با یک شناسه ای که نشان دهنده مخصوص هر متن باشد کار بکنیم که این شناسه همان doc_index است که در بخش های قبل نیز از آن استفاده کرده بودیم. در نتیجه در خروجی قسمت اول، دوست داریم tf متناظر هر کلمه در هر doc را داشته باشیم :

برای این کار با کلید قرار دادن (word, doc_index) به شکل زیر تعداد کلمات را در هر doc -

میشمریم :

```

word_tf_rdd = indexed_article_words_rdd.reduceByKey(lambda x,y:x+y)

# input:
## [(key = ('wordk' , doc_index) , value = 1 ) ,(key = ('wordk' , doc_index) , value = 1 )...]
# ouput :
## [(key = ('wordk' , doc_index) , value = tf ) , (key = ('wordk' , doc_index) , value = tf )...]

✓ 0.7s

word_tf_reordered_kv_rdd = word_tf_rdd.map(lambda x: (x[0][0] , (x[0][1], x[1])))
## (key = 'wordk' , value = (doc_index , tf))

✓ 0.4s

```

در این بخش، برای استفاده دیگری نیز، نیاز داشتیم ترتیب دیگری نیز ازین اطلاعات را بر اساس کلید word داشته باشیم.

- حالا نیاز داریم هر کلمه که در متن وجود دارد در چند document تکرار شده است و تکرارهایی که در یک document خاص از آن کلمه وجود دارد را در نظر نگیریم. برای اینکه بتوانیم تکرارهایی که از یک کلمه در همان document خاص وجود دارد را در نظر نگیریم، از خروجی خود word_tf_rdd استفاده کردیم و به ازای هر کلمه ای، آن را دور ریختیم و در value آن به df اضافه کنیم. سپس در نهایت تعداد تکرار هر کلمه را با یک reduceByKey به دست آوردیم:

```

df & idf

word_df_rdd = word_tf_rdd.map(lambda x:(x[0][0] , 1)).reduceByKey(lambda x,y: x+y)

# ouput 1:
## (key = 'wordk' , value = 1 )
# ouput 2:
## (key = 'wordk' , value = df )

word_idf_rdd = word_df_rdd.mapValues(lambda x:math.log((1+num_articles_b.value)/(1+x)))

# ouput :
## [(key = wordk , value = idf ) , (key = wordk , value = idf )....]

✓ 0.1s

```

در نهایت به جای عدد df که در value ها قرار گرفته است، عدد idf را گذاشتیم. خروجی این بخش همانطور که در قسمت سبز رنگ آخر مشاهده میکنید، می باشد.

- حالا باید key-value هایی که اطلاعات key-value را دارند، با doc_index, tf که اطلاعات join را دارند، در نهایت در خروجی، به ازای هر doc_index، کلماتی که در آن وجود داشته اند همراه با آنها را در یک دیکشنری به خروجی میدهیم :

join tf tuples & idf tuples

```

joined_words_and_docs_tf_idf_rdd = word_tf_reordered_KV_rdd.join(word_idf_rdd)
## (key = 'word' , value = ((doc_index , tf) , idf))

tfidf_rdd = joined_words_and_docs_tf_idf_rdd.map(lambda x: (x[1][0][0] ,(x[0], x[1][0][1]*x[1][1])))
## (key = doc_index , value = ('word' , tfidf))

indexed_tfidf_rdd = tfidf_rdd.groupByKey().mapValues(lambda x:dict(list(x)))

# output 1:
## (key = doc_index , value = iterable(('word1' , tfidf1),('word2' , tfidf2), ....) )
# output 2:
## (key = doc_index , value = {'word1':tfidf1 , ....})
✓ 0.1s

```

- در نهایت نیاز داریم این key_value هایی که با doc_index متناظر شدند را با یک عملیات join به متصل کنیم تا خواسته مساله برآورده شود :

Joining tfidf with the cleaned dic

```

joined_articles_and_tfidf_rdd = indexed_article_rdd.join(indexed_tfidf_rdd)
# output :
## (key = doc_index , value = (cleaned_dic ,{'word1':tfidf1 , ...} )

articles_tf_idf_vectors = joined_articles_and_tfidf_rdd.map(lambda x:{'title': x[1][0]['title'] , 'url':x[1][0]['url'] , 'text':x[1][0]['text'] , 'vector':x[1][1]})

# output :
## articles_tf_idf_vectors = [ cleaned_dic1 with added 'vector' list ,cleaned_dic2 with added 'vector' list ,..]
✓ 0.7s

```

نتایج نهایی :

```
[{"title": "صفحه اصلی",
"url": "https://fa.wikipedia.org/wiki?curid=2",
"text": "امروز میلادی برابر هجری خورشیدی stylescsgt UTC - NOEDITSECTION'،
'vector': {'5.672266446522762': 'خورشیدی',
'src14.150406910675299': 'صفحه اصلی',
'3.494866679668701': 'میلادی',
'lttemplatestyles': '10.199163192093872',
'4.082210328662131': 'اصلی',
'4.082210328662131': 'برابر',
'3.8033078316745117': '-',
'5.657199082009573': 'هجری',
'NOEDITSECTION': '14.150406910675299',
'UTC': '9.496446560517775',
'5.8628784875635365': 'امروز',
'5.8628784875635365': 'امروز : جیما ماندا انگری آدیجی',
'title': 'جيما ماندا انگری آدیجی',
"url": "https://fa.wikipedia.org/wiki?curid=149294",
"text": "نجربه دنیا آمد رمان نیمی که خورشید زرد سال ۲۰۰۷ جایزه ادبیات داستان زبان برای ... استرندگاو، ریان : استرندگاو و ۱۰.116166272522904، ۵.545935711152001، ۵.668737275328108، ادبیات : جایزه، نیجریه : ۷.578823913305377، زرد : ۶.389300699592881، دنیا : ۴.717403102621784، ... ۶.3957109779538 : خورشید، آدیجی : ۱4.150406910675299، رمان : ۶.1678198716904555، داستان : ۶.531664533004886، ... ۳۸۸۱۵۲۶۶۲۱۳۱۰۶۷۵۲۹۹"}
```

2) Searching:

- به طور کلی میتوان، برای مشابهت سنجی دو مقاله یا search کردن یک کلمه در همه مقالات موجود، از بردار tfidf هر دو استفاده کرد، به این نحو که بیاییم از هر دوی آنها که یکی متن یک کلمه ای است و یکی متن n کلمه ای، بردار های tfidf آنها را استخراج کنیم (که برای متنهای موجود این کار را کرده ایم و فقط کافی است برای متن query این کار را بکنیم) و این دو بردار را ضرب داخلی بکنیم (دقیق آنرا cosine similarity تا کل کلمات ما هست) تا این دو به دست بیاید و به ترتیب عددی که از ماکریم به مینیمم، به دست میاید آنها را sort کنیم. به علت به شدت کند اجرا شدن این الگوریتم، مجبور شدیم سرچ را محدود به یک کلمه و فقط برای ۲ یا ۳ کلمه متفاوت، آن را تست کنیم.
- الگوریتم را حالا به شکل زیر شرح میدهیم :
 ۱. میدانیم در یک متن یک کلمه ای $tf=1$ و از آنجایی که کلا یک متن فقط داریم و $df=1$ است در نتیجه idf را نیز میتوانیم ۱ در نظر بگیریم. (دقیق کنید که معیار لگاریتمی اینجا خوب نیست چون صفر میدهد و همینطور دقیق کنید این عدد به طور ذاتی اگر به صورت تک کلمه ای باشد مؤثر نیست و اهمیتی ندارد آن را چند میگیریم ولی در متن query چند کلمه ای مهم میشود.)
 ۲. حالا این کلمه را با همه کلمات همه documentها باید join کنیم تا ببینیم با کلمات یکسان در کدام document ها join میشود.

۳. بعد از join شدن، باید ضرب بین دو tfidf را محاسبه کنیم. دقت کنید که چون متن یک کلمه ای است، این حاصل ضرب، دقیقا نتیجه ضرب داخلی نیز هست.

۴. با استفاده از indexed_article_rdd ، این کلمات پیدا شده را با doc_index join بکنیم تا متناظر با این کلمات پیدا بشود.

همه این مراحل در تابع زیر انجام شده است و به همراه شکل خروجی :

```
def search_query(word):
    query_word_rdd = sc.parallelize([(word,1)])
    results_rdd = tfidf_rdd.map(lambda x:(x[1][0],(x[0], x[1][1]))).join(query_word_rdd).map(lambda x:(x[1][0][0] ,(x[0] , x[1][0][1]*x[1][1])))\n    .join(indexed_article_rdd)
    print("number of results for '{}' : {}".format(word,results_rdd.count()))
    results = results_rdd.top(2 ,lambda x: x[1][0][1])
    print("two most relevant results : \n" ,[item[1][1] for item in results])
    # input
    ## (key = doc_index , value = ('word' , tfidf))
    # output 1
    ## (key = 'word' , value = (doc_index , tfidf ))
    # output 2
    ## (key = 'word' , value = ((doc_index , tfidf),1))
    # output 3
    ## (key = doc_index , value = ('word' , cosine_similarity))
    # output 4
    ## (key = doc_index , value = (('word' , cosine_similarity) , article_dict))
```

Py

- نتایج برای کلمه : medicine

این نتایج در کد هم آمده است ولی دو نتیجه اول به شکل زیر است :

کردن دقیق کلمه و روش tfidf، اگر کلمه ای در vocabulary کلی ما نباشد، متنی از آن در نتایج سرج نخواهد آمد.

[obj]