به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

تحلیل ها و سیستم های داده های حجیم

پروژه ی پایانی

نام و نام‌خانوادگی	حسین قاسمی‌رامشه مسعود رحیمی سبحان راستی
شماره دانشجویی	810198224 810198161 810198160
تاریخ ارسال گزارش	29 تیر 1400

فهرست

مقدمه

در این پروژه با نحوه ی واکشی و پردازش داده های کلان و استفاده از پایگاه داده های مختلف برای پردازش، تحلیل و نمایش اطلاعات خروجی از آن ها آشنا شدیم. در هر قسمت چالش ها و مراحل کار از ورود داده تا نمایش نهایی را توضیح داده ایم. این لینک نیز به منظور دسترسی به کدهای این پروژه است.

گام اول – دریافت اطلاعات و Preprocess

مقدمه

برای داده‌های ورودی مشتاق بودم تا بتوانم داده‌های توییتر را دریافت کرده و آن‌ها را پردازش کنم؛ پس از ایجاد کد جهت دریافت داده‌های توییتر و استفاده از APIهای داخل دستورکار، ارور گرفتم و متوجه شدم که نیاز به دریافت API از توییتر است. سپس درخواست API کردم. اما توییتر موافقت نکرد و گفت فعلا امکان‌پذیر نیست. سپس به سراغ ساخت بات برای سروش رفتم. پس از ایجاد حساب و ساخت بات برای سروش، توی گوگل سرچ کردم ولی موفق نشدم طریقه عضو کردنِ بات در کانال ها را پیدا کنم و انگلیسی هم نمی‌توانستم سرچ کنم.

سپس به سراغ تلگرام رفتم و چند روزی روی این موضوع سرچ می‌کردم. چندجایی در اینترنت خواندم که امکان عضو کردن بات در کانال‌هایی که خوم ادمین آن‌ها هستم وجود دارد و من نمی‌توانم بات را در کانال دیگران عضو کنم. سپس متوجه شدم که می‌توانم پیام‌های حساب کاربری شخصی خودم را به کمک دریافت api و کتابخانه telethon وارد پایتون کنم. به این شکل امکان دریافت پیام کانال‌های تلگرام وجود داشت. اما توی قسمتی از documentation نوشته بود که اگر سرعت دریافت پیام، علی‌الخصوص برای کشورهای حساس مثل ایران و روسیه، بالا باشد، امکان بسته شدن اکانت شما وجود دارد (لینک مطلب ذکر شده).

به همین دلیل ریسک نکردم و به سراغ توییت‌های سهام‌یاب رفتم.

بیشینه ی مثبت

ابتدا کافکا را نصب کرده و به کمک commandهای زیر server و zookeeper را اجرا می‌کنم.

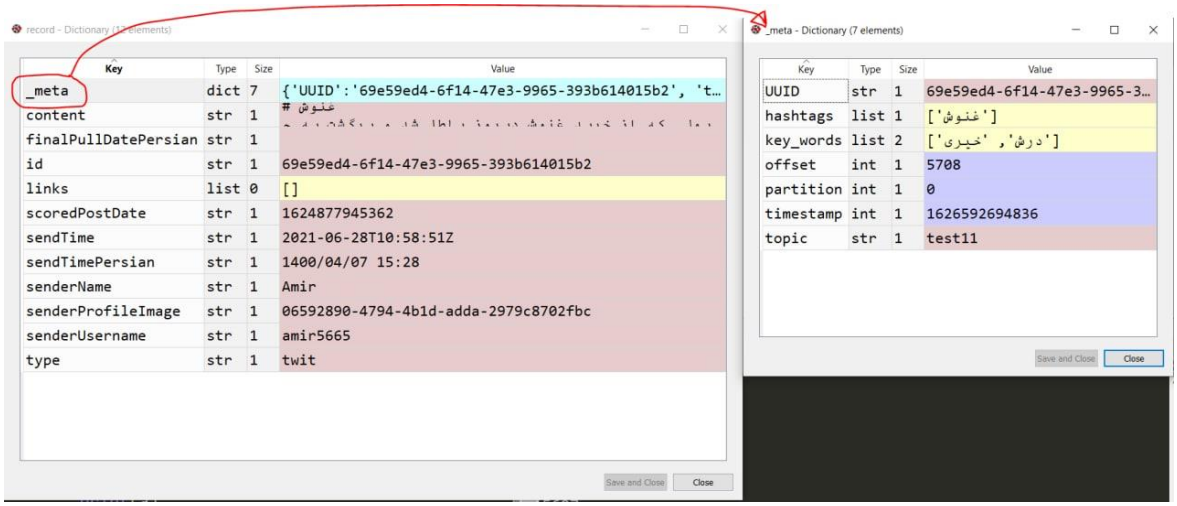
```
### 1st cmd in kafka dir
\.bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

### 2nd cmd in kafka dir
\.bin\windows\kafka-server-start.bat .\config\server.properties
```

حال با فراخوانی مداوم API داده‌ها را از سهام‌یاب دانلود می‌کنیم و سپس آن‌هایی که id تکراری ندارند را به کمک producer وارد topic کافکا به نام raw_data می‌کنیم. در همین حین، با یک consumer داده‌های خام را از تاپیک raw_data دریافت کرده و روی هر توییت پیش پردازش‌های گفته شده را اعمال می‌کنیم.

با دریافت هر توییت، ابتدا یک timestamps و UUID برای آن‌ها تولید کردم. به کمک یک تابع هشتگ‌های موجود در content هر توییت را یافته و آن را به اطلاعات همان توییت اظافه می‌کنیم. همین‌طور لینک‌ها هم استخراج شد.

جهت محاسبه کلمات کلیدی، از معیار tf/idf استفاده شده است. ابتدا لیستی از ایست‌واژه‌ها یافتم و با حذف کردن آن‌ها از متن توییت، tf/idf را محاسبه کردم. در tf/idf فراوانی هر کلمه را در هر توییت می‌یابیم و همین‌طور فراوانی همان کلمه را در همه‌ی توییت‌ها بررسی می‌کنیم. کلمه‌ای از tf/idf بالا برخوردار است (کلمه کلیدی است) که در یک توییت وجود دشته باشد اما در بقیه توییت‌ها کمتر تکرار شده باشد. سپس برای هر توییت علاوه بر کلمات خاصی که صورت سوال ذکر کرده است، دو کلمه‌ای که tf/idf بزرگتری دارند را به عنوان کلمات کلیدی در نظر گرفتم.

هر توییت ما یک دیکشنری است. در تصویر 1 key, value یک توییت را نمایش داده‌ام و در سمت راست تصویر اطلاعاتی که به عنوان متادیتا ذخیره خواهد شد، قرار دارد.



تصویر 1- کلید ها ی داده ها

در تصویر 1 می‌بینیم که partition برابر صفر و offset برابر ۵۷۰۸ ایت. یعنی ایت توییت در تاپیک test11 و در پارتیشن اول ان و همین‌طور در خانه‌ی ۵۷۰۸ پارتیشن اول قرار دارد.

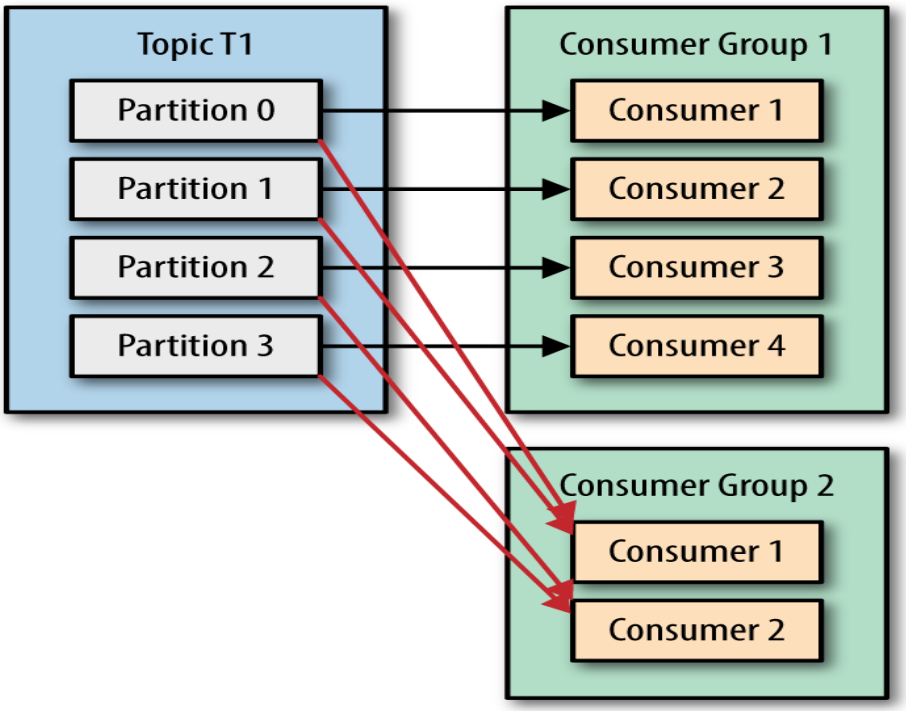
در نهایت به کمک یک producer جدید داده‌های پیش پردازش شده را در تاپیک proccesed_data قرار می‌دهیم.

معماری کافکا

کافکا شامل جمعی از record، topic، consumer، producer، broker، log و cluster است و هر record دارای key(اختیاری)، value و timestamp است.

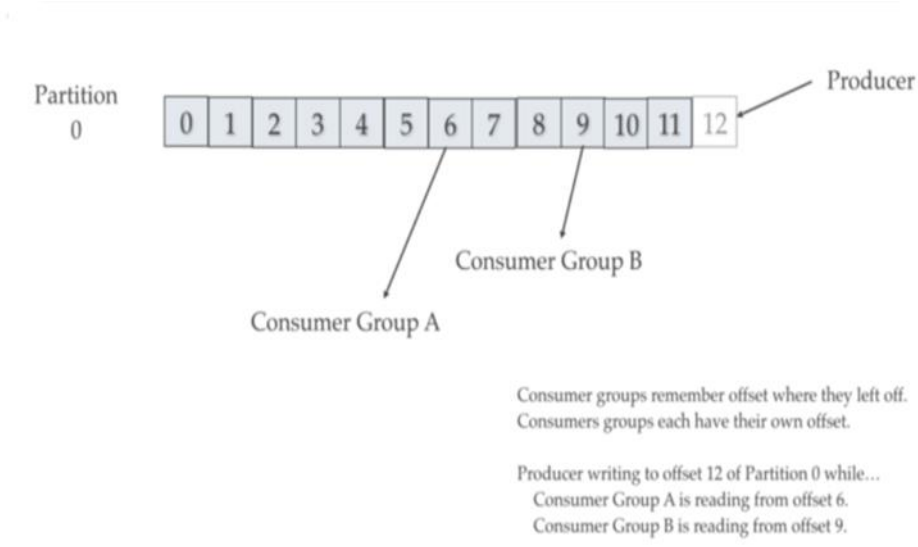
record هایی که وارد کافکا می‌شوند immutable هستند؛ یعنی producer به تنهایی فقط می‌تواند پیام‌ها را به تاپیک topic اظافه کند و امکان تغیر در recordهای موجد را ندارد.

هر تاپیک یک log دارد که محل ذخیره‌ی topic روی دیسک است. هر topic متشکل از تعدادی partition است. هر پارتیشن متشکل از تعدادی offset است (عکس زیر درک بهتری ایجاد می‌کند). هر record وارد topic مورد نظر و سپس وارد partition مورد نظر می‌شود و در آنجا به ترتیب درون offsetها ذخیره می‌شود. برای آنکه چندین consumer بتوانند همزمان record دریافت کنند، نیاز به وجود partition داریم. در عکس زیر نحوه‌ی اتصال consumerها به partition را می‌بینیم. با ایجاد consumer با group id متفاوت، می‌توان همزمان از یک تاپیک پیام دریافت کرد.



تصویر 2- معماری کافکا

هر پارتیشن بدین شکل است:



تصویر 3- پارتیشن بندی کافکا

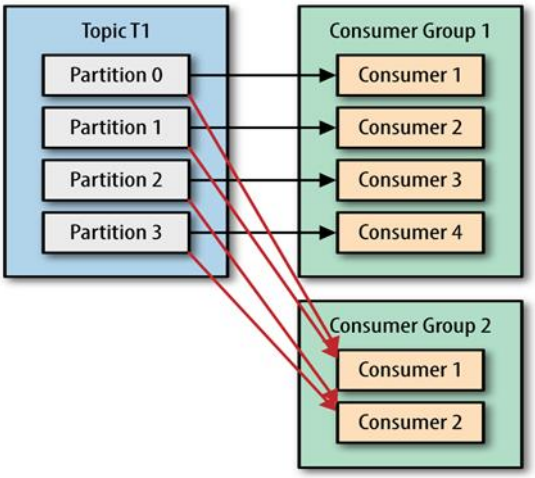
Producer برای تولید جریان داده و ارسال آن به topic استفاده می‌سود. Consumer پیام‌ها را از partition کافکا می‌خواند (هر topic شامل تعدادی partition است).

Cluster کافکا متشکل از تعدادی broker است و هر broker شامل تعدادی topic است.

گام دوم – persistence

گرفتن داده از کافکا

در این قسمت داده‌های پیش پردازش شده‌ی مرحله قبل را به کمک consumer با group id معین دریافت می‌کنیم (چون در گام‌های بعدی از تاپیک processed_data داده دریافت خواهیم کرد و در واقع داریم به صورت همزمان از partitionهای واحد توسط چند consumer پیام دریافت می‌کنیم، تعیین group id حیاتی است). این موضوع به تصویر 4 اشاره دارد:

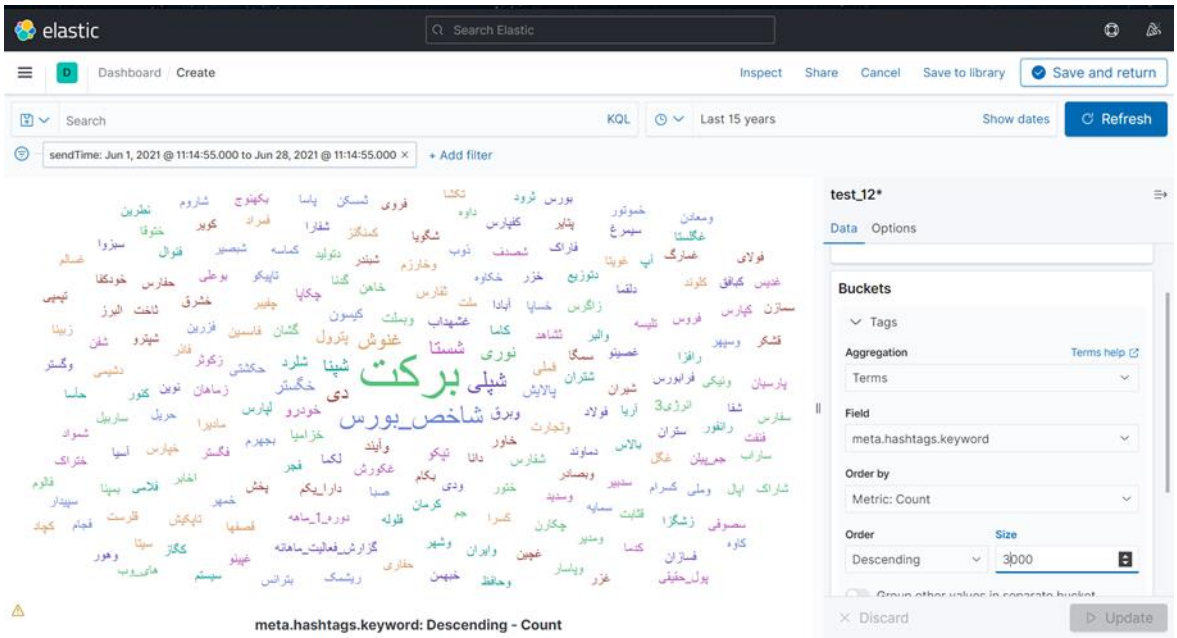


تصویر 4- **group id**

پس از دریافت داده ها از کافکا، داده‌ها را وارد index الاستیک سرچ می‌کنیم. سپس به کمک کیبانا، داشبوردهای خواسته شده را نمایش می‌دهیم.

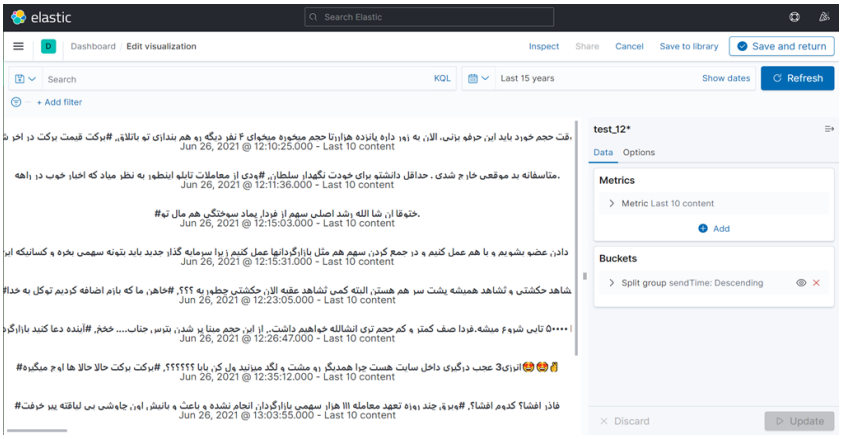
ابر کلمات یک کانال یا خبرگزاری خاص در یک بازه زمانی

از آنجایی که ما از داده‌های سهام‌یاب استفاده می‌کنیم، من ابر کلمات هشتگ‌ها در یک بازه‌ی خاص را رسم کرده‌ام.



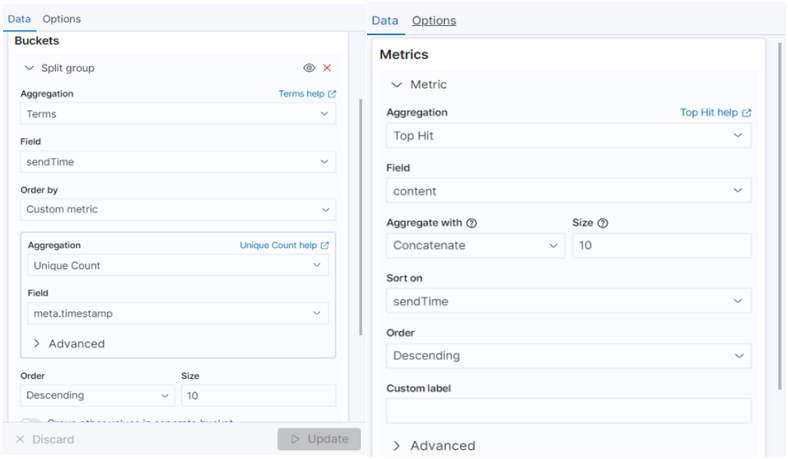
تصویر 5- ابرکلمات در الاستیک سرچ

متن ده پست اخیری که دریافت شده است



تصویر 6- متن ده پست اخیر در الاستیک سرچ

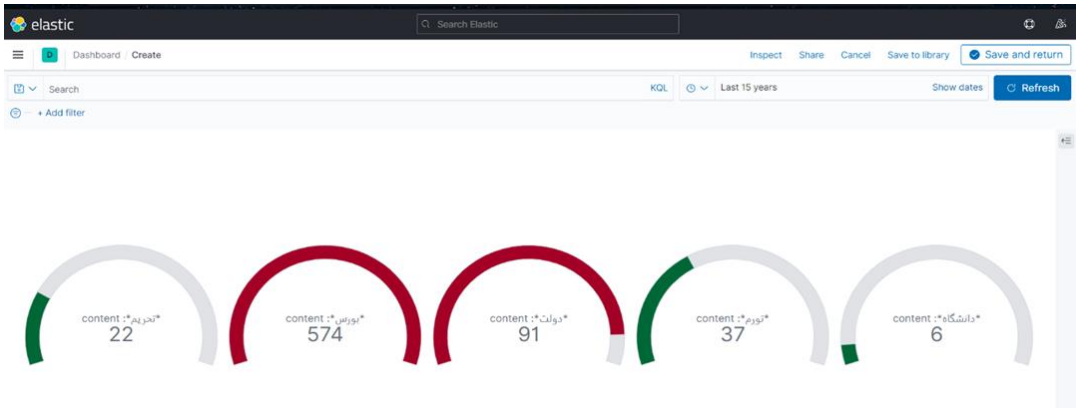
پارامترهای تعیین شده جهت نمایش داشبورد در تصاویر زیر آورده شده است (یعنی metrics و buckets).



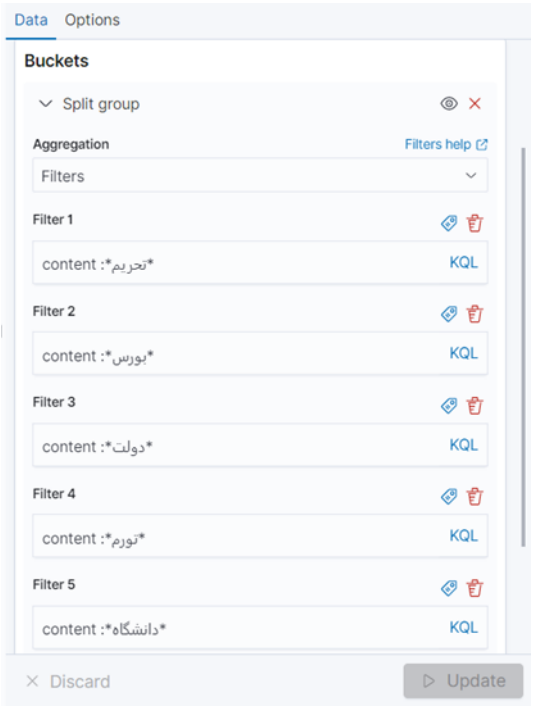
تصویر 7- پارامتر های داشبورد الاستیک سرچ

تعداد پست‌های ارسال شده به ازای چند تا از کلمات کلیدی خاص که در مرحله قبل مشخص شده است در یک بازه زمانی خاص

برای این منظور نتیجه و bucket استفاده شده را نمایش دادیم.

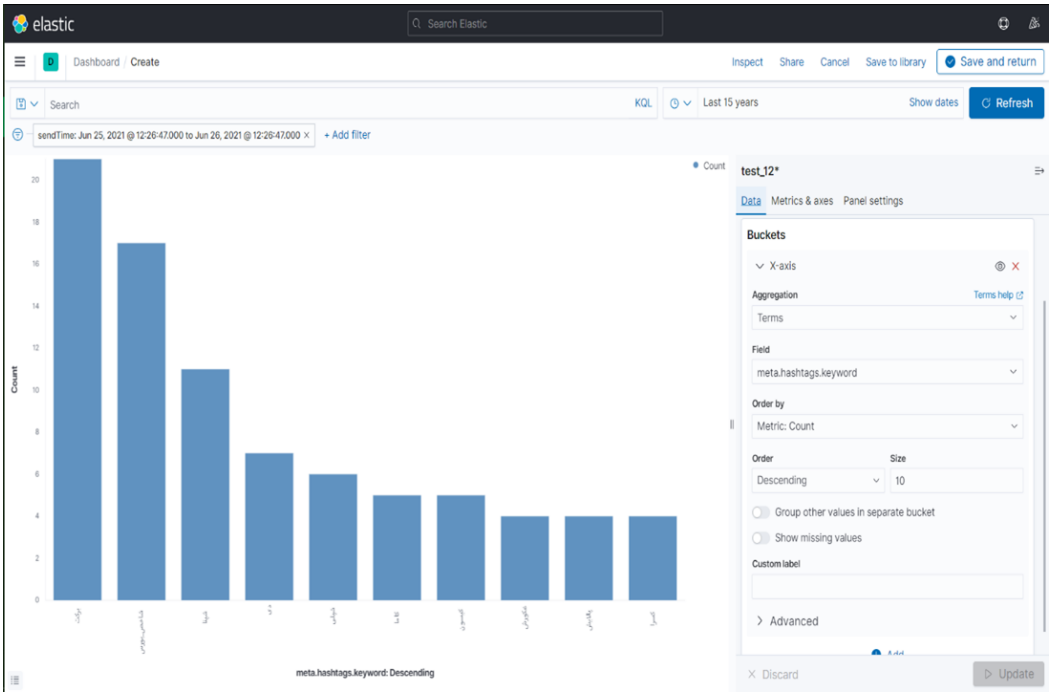


تصویر 8- آمار کلمات کلیدی در پست ها در الاستیک سرچ



تصویر 9- باکت برای آمار کلمات کلیدی در پست ها در الاستیک سرچ

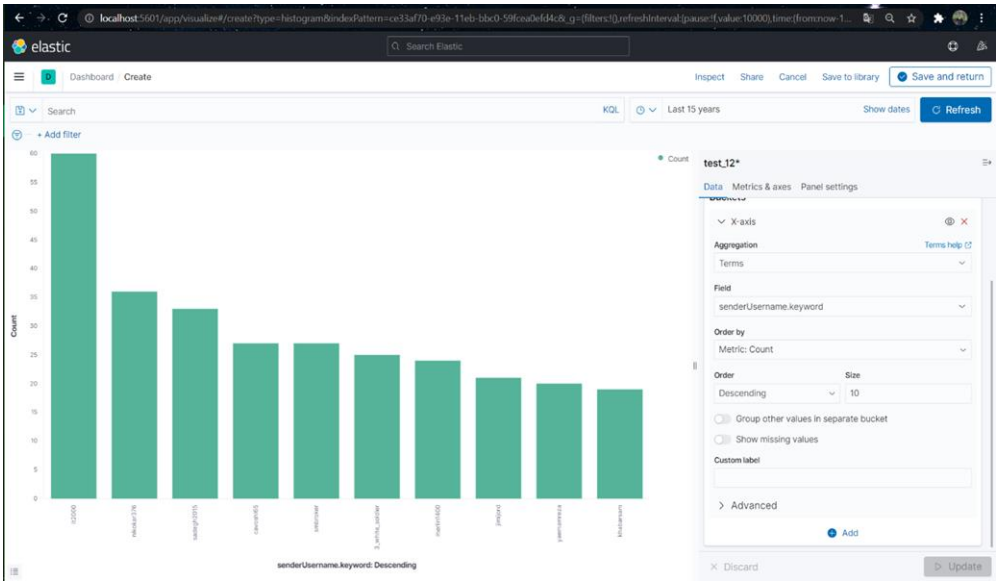
ده هشتگ بیشتر استفاده شده در پست‌های تمام کانال‌ها در یک بازه‌ی زمانی با تعداد تکرار هر هشتگ (یک نمودار ستونی) مثلا هشتگ‌های بیشتر استفاده شده در یک روز اخیر



تصویر 10- نمودار ده هشتگ بیشتر استفاده شده در پست ها در الاستیک سرچ

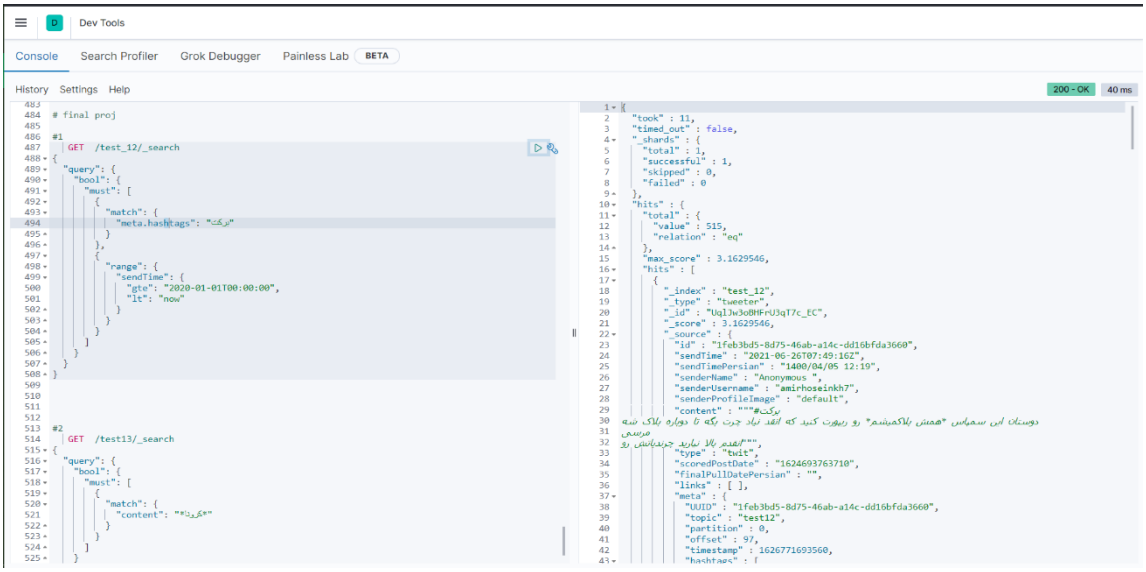
یک نمودار به انتخاب خودتان

در این قسمت تعداد توییت‌های یک کاربر را در یک نمودار ستونی نمایش داده‌ام.



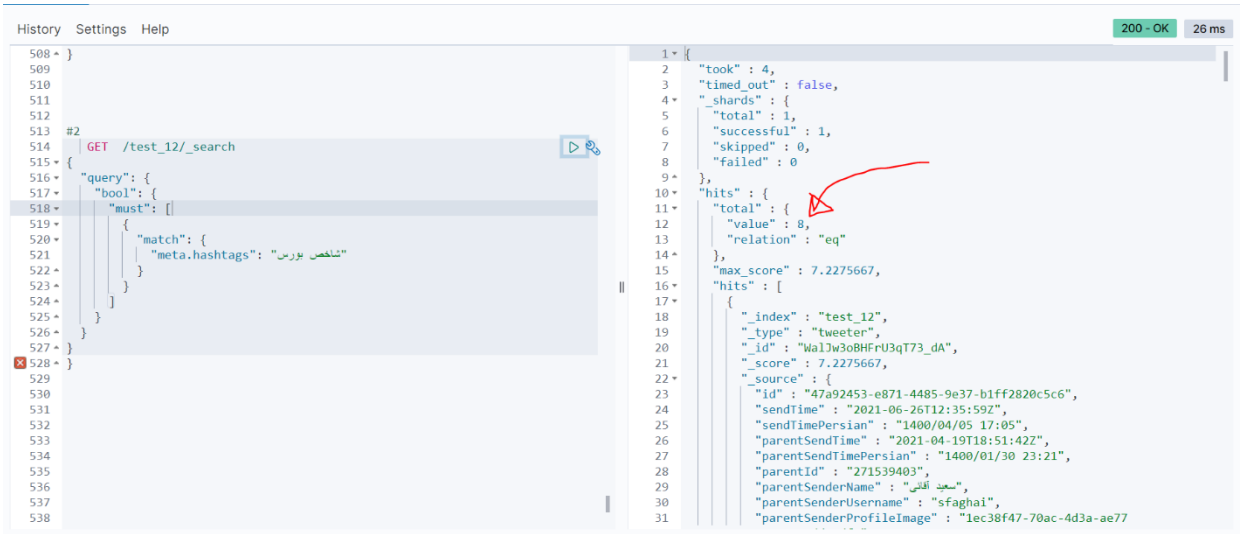
تصویر 10- نمودار تعداد توییت های کاربر در الاستیک سرچ

گر به دنبال تمام پست‌های جاوی یک کلمه خاص از یک خبرگزاری یا کانال خاص در یک بازه مشخص هستیم، چه دستوری باید بنویسیم (و یا یک هشتگ خاص یا یک کاربر خاص در توییت‌ها)



تصویر **11**- دستور هشتگ خاص در توییت ها در الاستیک سرچ

تعداد توییت‌های ارسالی به ازای یک کلمه خاص را به ازای یک هشتگ خاص در توییت‌ها در یک بازه زمانی



تصویر 12- تعداد توییت ها به ازای هشتگ خاص در توییت ها در الاستیک سرچ

گام سوم – Cassandra

مقدمه

بنده در این سوال سه جدول(البته استفاده از جدول درست نیست) ایجاد کردم یکی از آن‌ها بنابر سوالات داده شده طراحی شده است. در هر سه جدول کلید اول را کانال قرار داده‌ایم البته ما در این داده‌ها همه را از سهامیاب گرفته‌ایم پس فقط یک کانال داریم. بنده از پایتون برای اجرای کویری‌های کاساندرا استفاده کردم و این کار را با استفاده از کتابخانه cassandra انجام داده‌ام.

جدول اول

در جدول اولی چون میخواهیم به سوالات زمان ارسال پاسخ دهیم، پس sendTime را کلید بعدی قرار داده‌ایم:

```
session.execute("""
    CREATE TABLE IF NOT EXISTS fullData (
        canal text,
        sendTime timestamp,
        id text ,
        senderName text ,
        content text ,
        senderUsername text ,
        PRIMARY KEY (canal, sendTime, id) );
    """)
```

تصویر 13- جدول اول کاساندرا

جدول دوم

در جدول دوم هشتگ را کلید دوم گذاشتیم و بعد زمان ارسال را به این دلیل که بتوانیم به راحتی برای هر هشتگ در یک بازه زمانی درخواست شده داده‌های مربوطه را برگردانیم.

```
session.execute("""
    CREATE TABLE IF NOT EXISTS  hashtag (
        canal text,
        hashtag text,
        sendTime timestamp,
        id text ,
        senderName text ,
        content text ,
        senderUsername text ,
        PRIMARY KEY (canal, hashtag, sendTime, id) );
    """)
```

تصویر 14- جدول دوم کاساندرا

جدول سوم

در جدول سوم کلمات کلیدی را کلید دوم گذاشتیم و بعد زمان ارسال را به این دلیل که بتوانیم به راحتی برای هر کلمه کلیدی در یک بازه زمانی درخواست شده داده‌های مربوطه را برگردانیم.

```
session.execute("""
    CREATE TABLE IF NOT EXISTS  keyWord (
        canal text,
        keyword text,
        sendTime timestamp,
        id text ,
        senderName text ,
        content text ,
        senderUsername text ,
        PRIMARY KEY (canal, keyword, sendTime, id) );
    """)
```

تصویر 15- جدول سوم کاساندرا

کد ها و توابع پایتون

در داده‌ای که وارد میشود ابتدا آن را به کاساندرا اضافه میکنیم:

```
insert_fullData = session.prepare("""
        INSERT INTO fullData (
                        canal,
                        sendTime,
                        id,
                        senderName,
                        content,
                        senderUsername
                        )
        VALUES (?, ?, ?, ?, ?, ?)
        IF NOT EXISTS;
        """)
insert_hashtag = session.prepare("""
        INSERT INTO hashtag (
                        canal,
                        hashtag,
                        sendTime,
                        id,
                        senderName,
                        content,
                        senderUsername
                        )
        VALUES (?, ?, ?, ?, ?, ?, ?)
        IF NOT EXISTS;
        """)
insert_keyWord = session.prepare("""
        INSERT INTO keyWord (
                        canal,
                        keyword,
                        sendTime,
                        id,
                        senderName,
                        content,
                        senderUsername
                        )
        VALUES (?, ?, ?, ?, ?, ?, ?)
        IF NOT EXISTS;
        """)
```

تصویر 16- کد ورود داده کاساندرا

برای استفاده از داده‌های وارد شده بنده شش تابع نوشته‌ام که سه عدد از آن‌ها دو بازه زمانی کار میکنند که هر کدام از این سه با یک جدول کار میکند. سه تابع بعدی سوالات خواسته شده را پشتیبانی میکند به این صورت که دو ورودی دارد که یکی تعداد (ساعت، روز، هفته، ماه، سال) و دومی اگر نیاز بود هشتگ یا کلمه کلیدی مربوطه میباشد.

ابتدا کدهایی را مشاهده میفرمایید که بازه زمانی را میگیرند:

```
def all_post(timestamp1 , timestamp2, canal = 'sahamyab' ):
    cluster = Cluster(['127.0.0.1'])
    session = cluster.connect()
    session.set_keyspace(KEYSPACE)
    code = session.prepare("""
            SELECT * FROM fullData
            WHERE canal = ? AND sendTime >= ? AND sendTime <= ?;
            """)
    result = session.execute(code, [canal, timestamp1, timestamp2])
    return result

def all_post_by_hashtag(hashtag, timestamp1 , timestamp2, canal = 'sahamyab'):
    cluster = Cluster(['127.0.0.1'])
    session = cluster.connect()
    session.set_keyspace(KEYSPACE)
    code = session.prepare("""
            SELECT * FROM hashtag
            WHERE canal = ? AND hashtag = ? AND sendTime >= ? AND sendTime <= ?;
            """)
    result = session.execute(code, [canal, hashtag, timestamp1, timestamp2])
    return result

def all_post_by_keyWord(keyword, timestamp1 , timestamp2, canal = 'sahamyab' ):
    cluster = Cluster(['127.0.0.1'])
    session = cluster.connect()
    session.set_keyspace(KEYSPACE)
    code = session.prepare("""
            SELECT * FROM keyWord
            WHERE canal = ? AND keyWord = ? AND sendTime >= ? AND sendTime <= ?;
            """)
    result = session.execute(code, [canal,keyword,timestamp1, timestamp2])
    return result
```

تصویر 17- کد بازه زمانی کاساندرا

در ادامه کدهای داده‌هایی را مشاهده میکنید که تعداد میگیرد و بنا بر تعداد برای ساعت، ماه و ..... را محاسبه میکند و بر میگرداند و در کدها پیشبینی بزرگ شدن عدد ماه یعنی بزرگتر از 12 یا کوچکتر از 1 شده است:

```
def get_post(n):
    timestamp2 = datetime.today()
    timestamp1 = timestamp2 - timedelta(hours=n)
    n_hour = all_post(timestamp1 , timestamp2)

    timestamp1 = timestamp2 - timedelta(days=n)
    n_day = all_post(timestamp1 , timestamp2)

    timestamp1 = timestamp2 - timedelta(weeks=n)
    n_week = all_post(timestamp1 , timestamp2)

    m = timestamp2.month - int(n % 12)
    y = timestamp2.year - int(n / 12)
    if m <= 0:
        m = 12 + m
        y -= 1
    timestamp1 = datetime.strptime(f'{y}-{m}-{timestamp2.day}', '%Y-%m-%d')
    n_month = all_post(timestamp1 , timestamp2)

    timestamp1 = datetime.strptime(f'{timestamp2.year-n}-{timestamp2.month}-{timestamp2.day}', '%Y-%m-%d')
    n_year = all_post(timestamp1 , timestamp2)

    return n_hour, n_day, n_week, n_month, n_year
```

تصویر 18- کد محاسبه ی تعداد برای زمان مختلف کاساندرا بخش اول

```python
def get_post_by_hashtag(n, hashtag):
    timestamp2 = datetime.today()
    timestamp1 = timestamp2 - timedelta(hours=n)
    n_hour = all_post_by_hashtag(hashtag, timestamp1 , timestamp2)

    timestamp1 = timestamp2 - timedelta(days=n)
    n_day = all_post_by_hashtag(hashtag, timestamp1 , timestamp2)

    timestamp1 = timestamp2 - timedelta(weeks=n)
    n_week = all_post_by_hashtag(hashtag, timestamp1 , timestamp2)

    m = timestamp2.month - int(n % 12)
    y = timestamp2.year - int(n / 12)
    if m <= 0:
        m = 12 + m
        y -= 1
    timestamp1 = datetime.strptime(f'{y}-{m}-{timestamp2.day}', '%Y-%m-%d')
    n_month = all_post_by_hashtag(hashtag, timestamp1 , timestamp2)

    timestamp1 = datetime.strptime(f'{timestamp2.year-n}-{timestamp2.month}-{timestamp2.day}', '%Y-%m-%d')
    n_year = all_post_by_hashtag(hashtag, timestamp1 , timestamp2)

    return n_hour, n_day, n_week, n_month, n_year
```

تصویر 19- کد محاسبه ی تعداد برای زمان مختلف کاساندرا بخش دوم

```python
def get_post_by_keyWord(n, keyword):
    timestamp2 = datetime.today()
    timestamp1 = timestamp2 - timedelta(hours=n)
    n_hour = all_post_by_keyWord(keyword, timestamp1 , timestamp2)

    timestamp1 = timestamp2 - timedelta(days=n)
    n_day = all_post_by_keyWord(keyword, timestamp1 , timestamp2)

    timestamp1 = timestamp2 - timedelta(weeks=n)
    n_week = all_post_by_keyWord(keyword, timestamp1 , timestamp2)

    m = timestamp2.month - int(n % 12)
    y = timestamp2.year - int(n / 12)
    if m <= 0:
        m = 12 + m
        y -= 1
    timestamp1 = datetime.strptime(f'{y}-{m}-{timestamp2.day}', '%Y-%m-%d')
    n_month = all_post_by_keyWord(keyword, timestamp1 , timestamp2)

    timestamp1 = datetime.strptime(f'{timestamp2.year-n}-{timestamp2.month}-{timestamp2.day}', '%Y-%m-%d')
    n_year = all_post_by_keyWord(keyword, timestamp1 , timestamp2)

    return n_hour, n_day, n_week, n_month, n_year
```

تصویر 20- کد محاسبه ی تعداد برای زمان مختلف کاساندرا بخش سوم

گام چهارم – Redis

مقدمه

در این قسمت پس از نصب و راه اندازی Redis در حالیکه سرور و کلاینت Redis در حال اجرا است به وسیله ی کدهای پایتون به Redis متصل شده و داده های مورد نیاز را به Redis میفرستیم. برای دریافت آنلاین داده ها از کافکا به سراغ فایلی میرویم که همواره در حال آپدیت شدن است و داده های به روز را در فایلی در آدرس کد مد نظر ما ذخیره می کند. به منظور این که حجم این فایل از حدی بیشتر نشود بایستی صرفا داده های جدید را در فایل ذخیره کرد. در انتها و پس از انجام پردازش های مورد نیاز و گرفتن نتایج آن ها را از طریق صفحه ی HTML ای که توسط فلسک آماده شده به صورت مداوم نمایش میدهیم.

توضیحات کد پایتون پردازش داده ها

برای اینکه تمامی مراحل به صورت همروند انجام شده و نمایش داده بر روی وب اپلیکیشن هم به روز باشد از یک Thread برای گرفتن داده، ارتباط با Redis و بدست آوردن نتایج استفاده میکنیم و نمایش بر روی اپلیکیشن نیز توسط Thread اصلی انجام میگردد.

Thread پردازش تابعی به اسم process که در تصاویر زیر مشخص است را اجرا میکند. در این تابع به صورت مداوم و درون حلقه کارهای مورد نیاز انجام میگیرد. ضمنا Thread طوری طراحی شده که با تمام شدن کار در Thread اصلی آن نیز به پایان برسد. در تابع process ابتدا داده ها را از فایل مربوطه خوانده و نگه میداریم. سپس به ازای تمام توییت های دریافتی کارهای پردازشی و اضافه کردن آن ها به Redis را انجام میدهیم. برای این منظور کلیدی منحصر به آن توییت خاص و مرکب از اسم فرستنده ی توییت و زمان ارسال آن تهیه کرده و سپس در صورتی که آن کلید در Redis موجود نبود، کلید مدنظر را با یک عدد یک برای شمارش و همچنین زمان انقضای یک هفته ای به Redis اضافه میکنیم و در غیر اینصورت تعداد آن کلید خاص را تنها اضافه میکنیم. همین کار را برای کلید متشکل از زمان توییت هم انجام میدهیم. در صورتی که لیست مربوط به هشتگ های توییت خالی نبود این لیست را به همراه یک کلید مجزا ذخیره مینماییم. همین کار را برای ساخت لیست هزارتایی هشتگ ها هم انجام داده و هربار بعد از push کردن داده جدید اگر طول لیست از هزار بالاتر شد آن را Trim کرده و به اندازه ی1000 برمیگردانیم. بعد از ذخیره سازی هشتگ ها خود متن توییت ها را نیز در لیستی دیگر اما این بار با حداکثر اندازه ی 100 ذخیره میکنیم.

برای گرفتن نتایج برای سوال اول که همان مجموع توییت های دریافتی در شش ساعت اخیر است، به ازای هر ساعت، کلید ساخته شده برای شمارش این قسمت را به وسیله ی کم کردن آن ساعت از زمان فعلی بازیابی کرده و تعداد توییت موجود در Redis با آن کلید را به دست می آوریم و حاصل را جمع میکنیم.

برای سوال دوم همان کار قبلی را اینبار به ازای مقادیر زمان های خاص در یک بازه ی زمانی دلخواه مثلا یک روز گذشته تکرار میکنیم. برای سوال سوم همین کار اینبار برای هشتگ ها و برای بازه ی یک ساعت گذشته اجرا کرده و تعداد هشتگ ها را بدست می آوریم. با گرفتن موارد ذخیره شده با کلید لیست هشتگ ها، به هدف سوال چهار یعنی بدست آوردن یک لیست هزارتایی از هشتگ ها میرسیم. در نهایت به همین روش اما با کلیدی متفاوت لیست توییت ها را دریافت میکنیم. در انتها همه ی جواب ها را به همراه متن مناسب برای استفاده ی فلسک در متغیری global ذخیره میکنیم. در زیر تصاویر مربوط به کد این تابع را میبینید:

```
34  def proccess():
35      global Q1
36      global Q2
37      global Q3
38      global Q4
39      global Q5
40      global hashtagsList
41      global tweetsList
42      while True:
43          time.sleep(0.01)
44          f2 = open('preprocessed_data.json', encoding="utf8")
45          data = json.load(f2)
46          f2.close()
47
48          expirationMinutes = find_expiration_minutes(7)
49
50          for item in data:
51              # print(item)
52              key = "username: " + item["senderUsername"] + " in: " + find_date(item["sendTime"])
53              if redis.exists(key) == 0:
54                  redis.set(key, 1, ex=expirationMinutes)
55              else:
56                  redis.incr(key)
57
58              key = "tweets: " + find_date(item["sendTime"])
59              if redis.exists(key) == 0:
60                  redis.set(key, 1, ex=expirationMinutes)
61              else:
62                  redis.incr(key)
63
64              if "" not in item["_meta"]["hashtags"] and len(item["_meta"]["hashtags"]) != 0:
65                  key = "unique hashtags: " + find_date(item["sendTime"])
66                  if redis.exists(key) == 0:
67                      redis.sadd(key, *item["_meta"]["hashtags"])
68                      redis.expire(key, expirationMinutes)
69                  else:
70                      redis.sadd(key, *item["_meta"]["hashtags"])
71
72                  if redis.exists("hashtags_list") == 0:
73                      redis.lpush("hashtags_list", *item["_meta"]["hashtags"])
```

تصویر 21- بخش اول کد تابع **process** برای **Redis**

```
                    redis.expire("hashtags_list", expirationMinutes)
                else:
                    redis.lpush("hashtags_list", *item["_meta"]["hashtags"])
                    if redis.llen("hashtags_list") > 1000:
                        redis.ltrim("hashtags_list", 0, 999)

            if redis.exists("tweets_list") == 0:
                redis.lpush("tweets_list", item["content"])
                redis.expire("tweets_list", expirationMinutes)
            else:
                redis.lpush("tweets_list", item["content"])
                if redis.llen("tweets_list") > 100:
                    redis.ltrim("tweets_list", 0, 99)


        lastHours = 6
        now = datetime(2021, 7, 20, 23)

        totalTweets = 0
        for h in range(lastHours + 1):
            goalDate = now - timedelta(hours=h)
            key = "tweets: " + find_date_from_obj(goalDate)
            if redis.exists(key) > 0:
                totalTweets += int(redis.get(key).decode("utf-8"))

        Q1 = "در " + str(lastHours) +" ساعت اخیر " + str(totalTweets) + " توییت دریافت شده است."

        dateFrom = datetime(2021, 7, 1, 18)
        dateTo = datetime(2021, 7, 20, 23)

        delta = timedelta(hours=1)
        i = dateFrom
        totalTweets = 0
        while i <= dateTo:
            key = "tweets: " + find_date_from_obj(i)
            if redis.exists(key) > 0:
                totalTweets += int(redis.get(key).decode("utf-8"))
            i += delta
```

تصویر **22**- بخش دوم کد تابع **process** برای **Redis**

```
        Q2 = "از تاریخ " + str(dateFrom) +" تا " + str(dateTo) +" تعداد " + str(totalTweets) + " دریافت شده است"

        lastHours = 1
        totalHashtags = 0

        for h in range(lastHours + 1):
            goalDate = now - timedelta(hours=h)
            key = "unique hashtags: " + find_date_from_obj(goalDate)
            if redis.exists(key) > 0:
                totalHashtags += redis.scard(key)

        Q3 = "در " + str(lastHours) +" ساعت اخیر " + str(totalHashtags) +  " هشتگ زده شده است."


        indexFrom = 0
        indexTo = -1

        #hashtagsList = []
        for item in redis.lrange("hashtags_list", indexFrom, indexTo):
            hashtagsList.append(item.decode("utf-8"))

        Q4 =  str(hashtagsList)



        indexFrom = 0
        indexTo = -1
        # tweetsList = []
        #print(tweetsList)
        for item in redis.lrange("tweets_list", indexFrom, indexTo):
            tweetsList.append(item.decode("utf-8"))

        Q5 = "صد توییت آخر این ها هستند: "
        # print(tweetsList)
```

تصویر 23- بخش سوم کد تابع **process** برای **Redis**

نمایش داده ها به وسیله ی Flask

در این بخش برای نمایش نتایج از Flask کمک گرفته و نتایج ذخیره شده در متغیرهای global را با تگ های مناسب HTML ترکیب کرده و آن را برای نمایش به Flask داده و نتیجه را اجرا مینماییم. در زیر کد مربوط به این بخش و نمونه هایی از نتایج قابل مشاهده اند.

```
150  @app.route('/')
151  @app.route('/home')
152
153  def home():
154      s = ""
155      s = "<html><head><style>p.ex1 { border: 1px solid green;padding: 20px;margin: 20px;}</style></head><body>"
156      s = s + "<h2>" + Q1 +  "</h2>"
157      s = s + "<h2>" + Q2 +  "</h2>"
158      s = s + "<h2>" + Q3 +  "</h2>"
159      s = s + "<h2>" +  "هزار هشتگ آخر این ها هستند: " +  "</h2>"
160      s = s + "<p  class='ex1', style='background-color: #ffffff'>" + Q4 +  "</p>"
161      s = s + "<h1>" + Q5 +  "</h1>"
162      for x in tweetsList:
163          s = s + "<p  class='ex1', style='background-color: #ffffff'>" + str(x) +  "</p>"
164      s = s + "</tr>"
165      return "<html dir='rtl' lang='ar'><body style='background-color: lightgrey'>" + s + "</body></html>"
166
167
168  if __name__ == '__main__':  # Script executed directly?
169      proccess_thread = threading.Thread(target=proccess)
170      proccess_thread.setDaemon(True)
171      proccess_thread.start()
172      app.run(debug = True)  # Launch built-in web server and run this Flask webapp
173
```

تصویر 24- کد تابع **Flask** و اجرای برنامه برای **Redis**

تصویر 25- بخش اول نمونه ای از خروجی سوال **Redis**

تصویر 26- بخش دوم نمونه ای از خروجی سوال **Redis**

گام پنجم – Clickhouse

مقدمه

در این بخش چالش زیادی برای راه اندازی Clickhouse داشتیم، چرا که برای کار با این دیتاست نیاز به سیستم عامل لینوکس بود. البته ابزارهای دیگری در ویندوز این امکان را فراهم میکرد که خود آن ها نیز چالش های خاص خود را داشت. به هر صورت پس از نصب و راه اندازی Clickhouse در حالیکه سرور آن در حال اجراست به وسیله ی کد پایتون داده ها را در آن وارد کرده و جداول را ساختیم.

توضیحات کد پایتون و ارتباط با Clickhouse

در این بخش همانطور که در تصاویر زیر مشخص است پس از اتصال به سرور Clickhouse داده های پردازش شده و ریخته شده در فایل توسط کافکا به صورت آنلاین و مداوم را خوانده و پس از انتخاب بخش های مورد نیاز آن، حاصل را به Clickhouse اضافه مینماییم.

```python
import json
from clickhouse_driver import Client
from datetime import datetime

# client = Client('localhost')
client = Client('localhost',
                password='*****')

client.execute('DROP TABLE IF EXISTS default.sahamyab')
client.execute("""CREATE TABLE IF NOT EXISTS default.sahamyab
               (
               id String,
               sendTime DateTime,
               sendTimePersian String,
               hashtags Array(Nullable(String)),
               key_words Array(Nullable(String)),
               senderUsername String,
               senderName String,
               content String
               )
                ENGINE = MergeTree PARTITION BY toYYYYMMDD(sendTime) ORDER BY toYYYYMMDD(sendTime)
               """)


def insert(jData):
    data = [[
        jData['id'],
        datetime.strptime(jData['sendTime'], '%Y-%m-%dT%H:%M:%SZ'),
        jData['sendTimePersian'],
        jData['_meta']['hashtags'],
        jData['_meta']['key_words'],
        jData['senderUsername'],
        jData['senderName'],
        jData['content']
    ]]
```

تصویر 27- بخش اول کد اضافه کردن داده به **Clickhouse**

```
                                '('
                                'id,'
                                'sendTime,'
                                'sendTimePersian,'
                                'hashtags,'
                                'key_words,'
                                'senderUsername,'
                                'senderName,'
                                'content'
                                ')'
                                ' VALUES ', data)
    print(jData['id'])
    return True

def insert_jsons(data):
    for i in data:
        insert(i)

import json
# use the below code to read the data
f2 = open('preprocessed_data.json', encoding="utf8")
jdata = json.load(f2)
f2.close()
insert_jsons(jdata)
```

تصویر **28**- بخش دوم کد اضافه کردن داده به **Clickhouse**

تصویر 29- داده ها در محیط **Clickhouse**

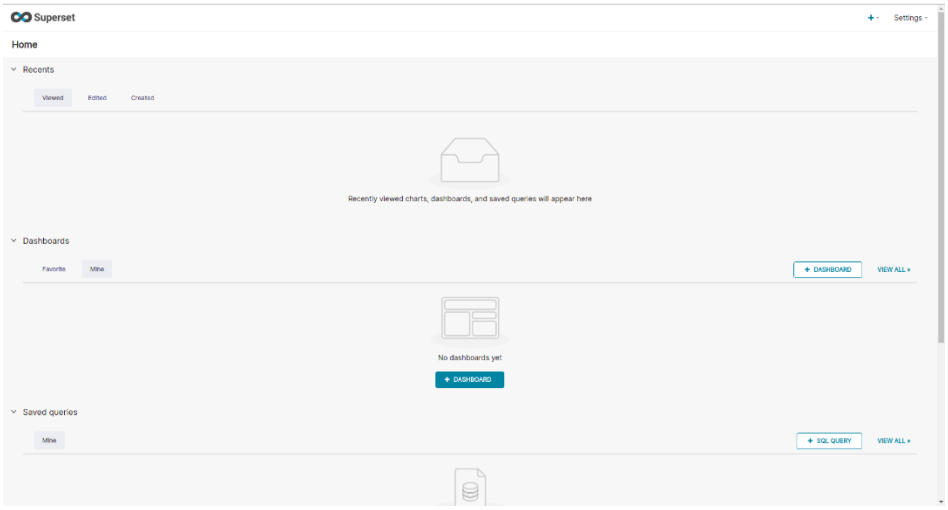
گام ششم – Apache Superset

مقدمه

در این بخش پس از اضافه شدن داده ها به Clickhouse و نصب و راه اندازی Apache Superset، Superset را به Clickhouse متصل کردیم. این اتصال در صورت داشتن رمز برای Clickhouse در ویندوز برای ما چالش برانگیز بود، چرا که نمیتوانستیم اتصال را برقرار کرده و از طرف دیگر نتوانستیم رمز را حذف نماییم. در نهایت در سیستم عامل لینوکس این مهم میسر شد.

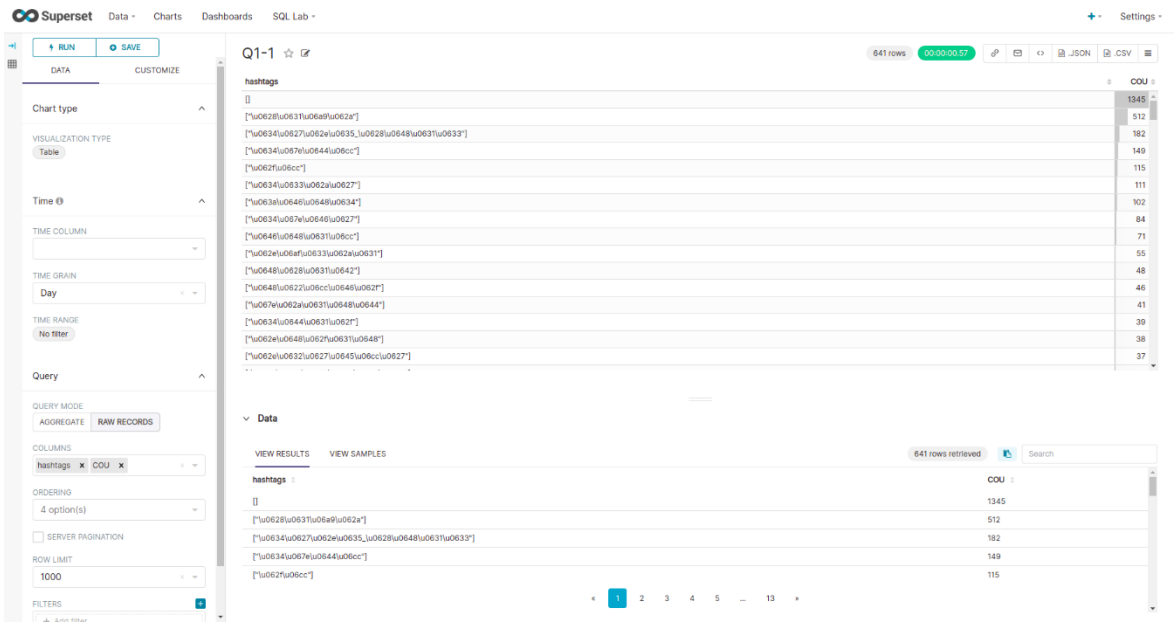
نمودارها و موارد خواسته شده

با توجه به موارد خواسته شده در هر بخش کوئری مربوطه را داده و داده هارا از Clickhouse دریافت کرده سپس بعد از explore کردن داده ها و ذخیره ی آنها با انتخاب پارامتر ها و نمودارهای مناسب نتیجه را بدست آوردیم برای مواردی نیز به وسیله ی رابط کاربری superset و محیط گرافیکی آن و بدون نیاز به کوئری نمودار های خواسته شده را بدست آوردیم. همانطور که در دستور کار خواسته شده است **سه داشبورد** از موارد خواسته شده تهیه کردیم که آنها را در ادامه می بینیم.

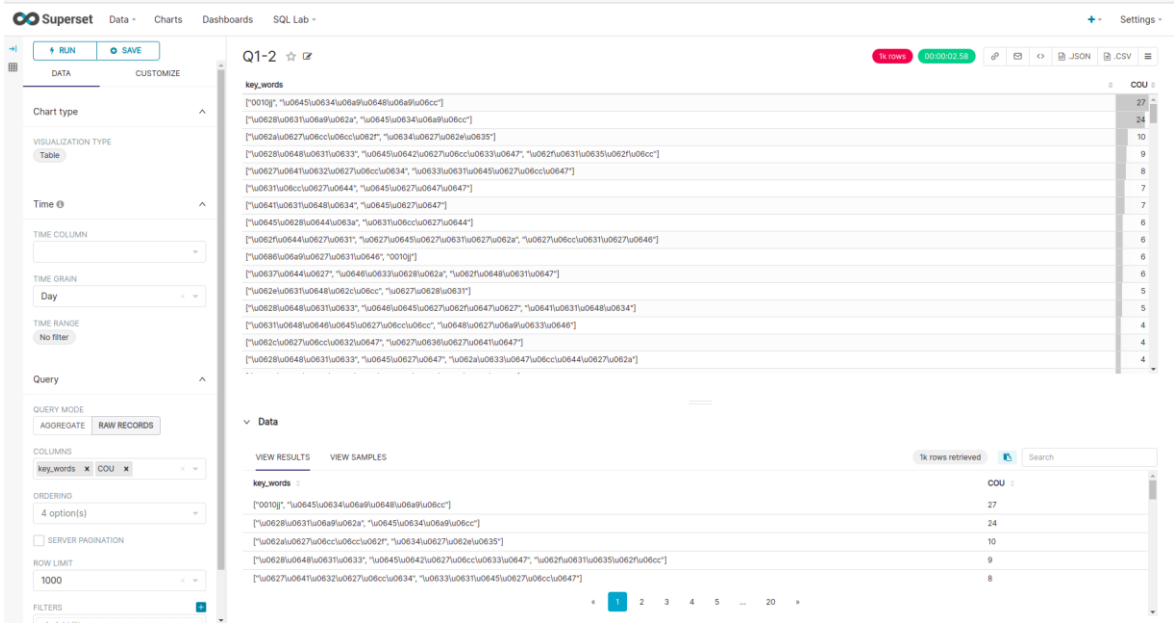


تصویر 30- محیط **superset**

در تصاویر 31 و 32 جداول بدست آمده پس از در خواست های انجام شده را می بینید. در تصویر 31 جدول و نمودار نزولی برای بیشترین تکرار در بین هشتگ ها و همین مورد را در تصویر 32 برای کلمات کلیدی می بینید.

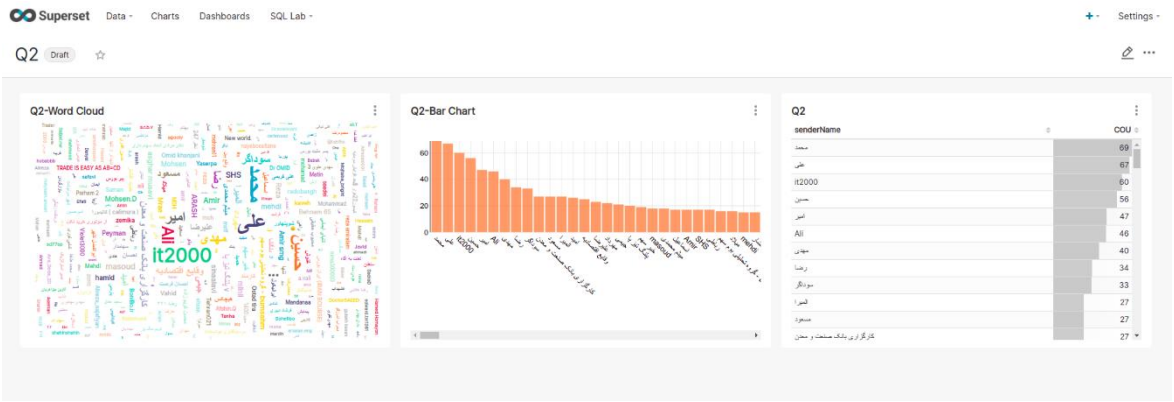


تصویر 31- هشتگ های دریافتی به همراه تعداد و نمودار آنها به صورت اسکی در محیط **superset**



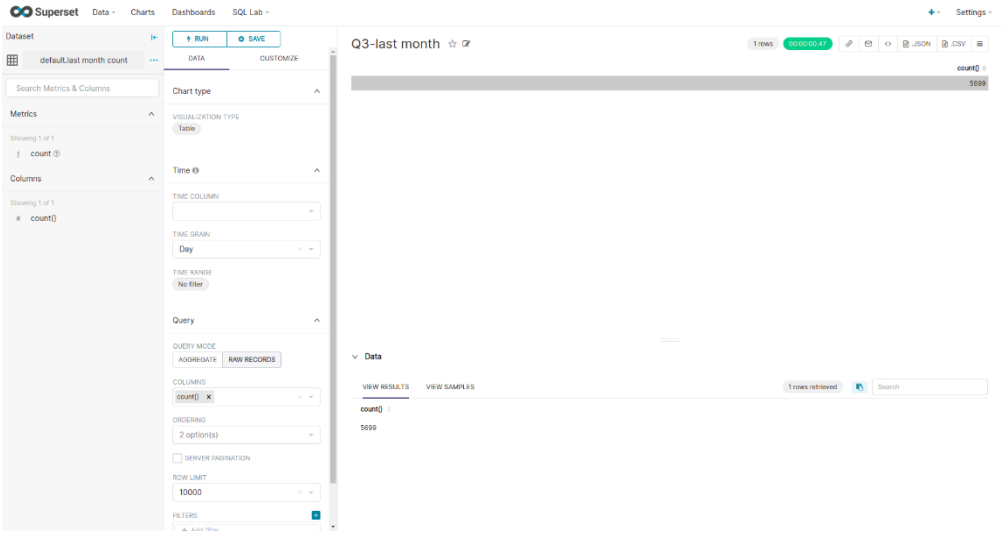
تصویر 32- کلمات کلیدی به همراه تعداد و نمودار آنها به صورت اسکی در محیط **superset**

در این بخش تعداد توییت های ارسال شده توسط هر کاربر را به دست آورده و به صورت نزولی مرتب کردیم و نمودار های مربوطه را کشیدیم. در ابر کلمات که یک ابزار کارآمد برای تحلیل داده های متنی است مشاهده میکنیم که کاربرانی با نام علی، محمد، سوداگر،it2000 از فعال ترین کاربرها در این حوزه بوده اند.

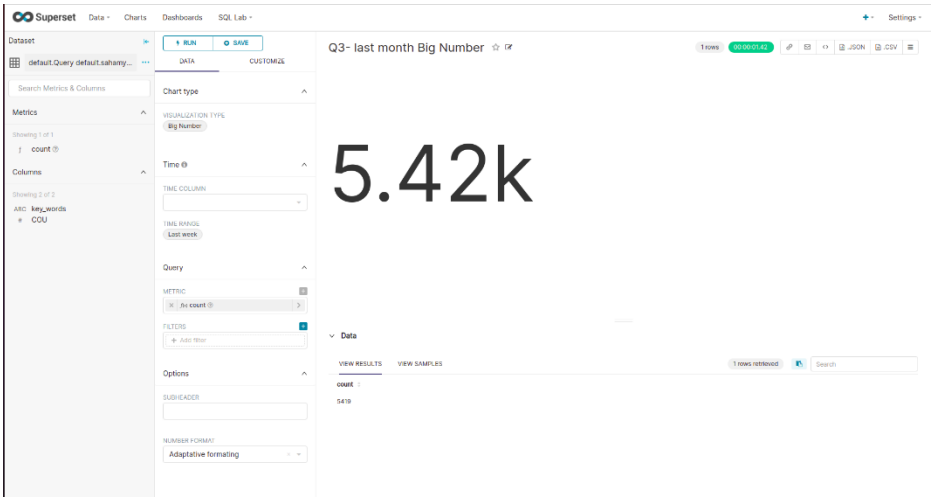


تصویر 33- تعداد توییت های کاربران به صورت نزولی در قالب جدول نمودار و ابرکلمات در محیط **superset**

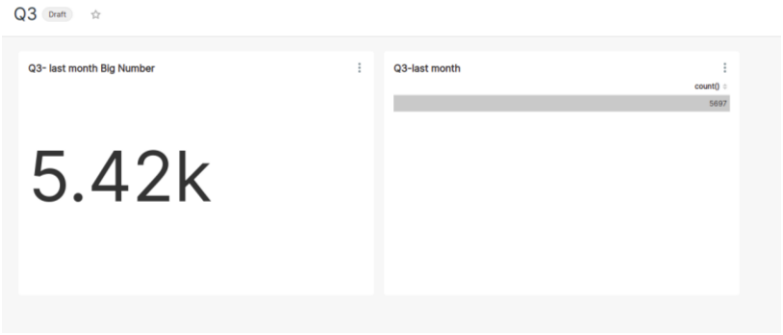
در ادامه تعداد توییت ها را در زمان ها مختلف بررسی کردیم که در تصویر 33 تعداد به دست آمده برای ماه اخیر را میبینیم و در تصاویر بعدی فرم این داده را به صورت Big Number کشیدیم.



تصویر 34- تعداد توییت ها در ماه اخیر در محیط **superset**



تصویر 35- **Big Number** تعداد توییت ها در ماه اخیر در محیط **superset**



تصویر 35- **Big Number** و تعداد توییت ها در ماه اخیر در محیط **superset**

پیوست – روند اجرای برنامه

به ازای هر قسمت کد مربوطه در پیوست آمده است.