دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس

# کلان داده و تحلیل داده‌های حجیم

**دکتر اسدپور**

نیمسال دوم سال تحصیلی ۱۳۹۹-۱۴۰۰

# پروژه پایانی + دو راه حل نمونه

*طراحی یک سامانه بلادرنگ برای تحلیل لحظه‌ای داده‌های پیام‌رسان‌های داخلی/ توئیت‌های فارسی*

**(Elasticsearch, Kafka, Cassandra, Spark, Redis, Clickhouse, Superset)**

**طراح تمرین :**

*مجتبی بنائی*

مهلت تحویل : ۲۰ تیرماه ۱۴۰۰

## مقدمه

هدف از انجام پروژه نهایی درس کلان‌داده، آشنایی عملی با طراحی یک سامانه کاربردی پردازش داده بلادرنگ و مقیاس‌پذیر با استفاده از ابزار و کتابخانه‌های روز دنیا در حوزه بیگ‌دیتا است. انتظار می‌رود پس از انجام این پروژه دیدی تجربی و شهودی نسبت به مفاهیم زیر پیدا کنید :

1. صف‌های توزیع شده و نقش محوری آن‌ها در سامانه‌های نوین اطلاعاتی.
2. الاستیک‌سرچ و قدرت و کارآیی فوق‌العاده آن در مدیریت داده‌های متنی و جی‌سان
3. کاساندرا به عنوان یک دیتابیس سطرگسترده مقیاس‌پذیر سهل‌الوصول و کارآمد
4. اسپارک و سهولت پیاده‌سازی الگوریتم‌های پیچیده یادگیری ماشین بر روی حجم عظیم داده به کمک آن .
5. سوپرست به عنوان یک ابزار دم‌دستی و کاربردی برای بصری سازی نتایج پردازش و ساخت داشبوردهای تحلیلی
6. دیتابیس‌های تحلیلی و نقش آن‌ها در تصمیمات مدیریتی سازمانی

جزییات پروژه و مستندات مورد نیاز برای هر قسمت، در ادامه آمده است.

سعی شده است تمرکز اصلی پروژه، کار با ابزار و کتابخانه‌های ذکر شده باشد و خود کارهای پردازشی و کدهای مورد نیاز، حجم کمی را به خود اختصاص دهد.

## چشم‌انداز کلی سامانه

در این پروژه قرار است داده‌های حدود ده‌هزارکانال اطلاع‌رسانی از پیام‌رسان‌های داخلی و یا توئیت‌های فارسی را به صورت لحظه‌ای بررسی کنیم و ضمن استخراج و ذخیره اطلاعات مفید از آنها، بتوانیم برآوردی از زمان پست‌های بعدی آن‌ها و یا تعداد اشتراک‌گذاری آن‌ها داشته باشیم .

با توجه به حجم کار این پروژه، می‌توانید تیم‌های حداکثر چهار نفره تشکیل دهید که هر تیم یک مدیر یا هماهنگ کننده خواهد داشت. در صورتی که تعداد اعضای تیم شما حداکثر دونفره باشد، با هماهنگی با دستیاران آموزشی می‌توانید از انجام بخشی از کار، صرف نظر کنید.

منابع اصلی ورود داده در این پروژه از قرار زیر هستند که می‌توانید یکی از آنها را به دلخواه انتخاب نمایید :

1. پیام‌رسان‌های داخلی مانند سروش، آی‌گپ و بله خواهند بود که هر تیم، با یکی از آن‌ها کار خواهد کرد. کدهای خزش برای پیام‌رسان‌ها توسط خود اعضای تیم باید نوشته شود.
2. توئیتر و داده‌های فارسی روزانه آن .
3. توئیت‌ها و پیام‌های سایت‌های فارسی بورس ایران مانند سهامیاب و ره‌آورد۳۶۵

هدف عملیاتی این پروژه، بررسی امکان خزش و تحلیل داده‌های پیام‌رسان‌های داخلی و یا توئیت‌های فارسی، مانیتورینگ و یافتن داده‌های آماری مرتبط با هرکانال(در پیام‌رسان‌ها) و هشتگ (برای توئیت‌ها) و انجام پردازش‌های مختلف بر اساس داده‌های آن‌ها به صورت بلادرنگ و نمایش آن‌ها به کاربر از طریق داشبوردهای اطلاعاتی خواهد بود.

روند کلی پردازش داده در سامانه نهایی از قرار زیر خواهد بود :

- داده‌ها، به کمک وب‌هوک یا API های هر پیام‌رسان یا توئیتر و سایتهای فارسی بورس، دریافت و وارد کانال اولیه در کافکا می‌شوند. (هماهنگی کل پروژه و گام‌های مختلف از طریق کافکا انجام میشود که در دنیای واقعی هم همین نقش بر عهده این نرم‌افزار است)
- در گام اول (*PreProcess*)، پیش‌پردازش‌های اولیه متنی بر روی داده‌ها انجام شده، کلمات کلیدی و هشتگ‌ها استخراج می‌شوند و به عنوان متادیتا، در کنار داده‌های دریافت شده قرار میگیرند. این داده‌ها وارد کانال دوم می‌شوند.
- در گام دوم (*persistence*)، داده‌های دریافتی در الاستیک سرچ ذخیره شده، بدون انجام پردازش خاصی، وارد کانال سوم می‌شوند .
- در گام سوم (*ChannelHistory*)، داده‌ها براساس نام خبرگزاری یا ارسال کننده محتوی/توئیت، کلمات کلیدی ، هشتگ‌ها، اشخاص یا کلمات خاص، در کاساندرا ذخیره می‌شوند. هدف از این مرحله، ایجاد مکانیزمی برای

بازیابی سریع پست‌ها براساس نام کانال، کلمه کلیدی، هشتگ یا اشخاص/کلمات خاص است. سپس داده‌ها وارد کانال بعدی می‌شوند.

- در گام چهارم (*Statistics*)، اطلاعات آماری مورد نیاز مانند تعداد اخبار در یک حوزه خاص، خبرگزاری خاص، هشتگ خاص و مانند آن، به روز رسانی می‌شود. این اطلاعات در ردیس ذخیره می‌شود. سپس داده‌ها وارد کانال پنجم می‌شوند.
- در گام پنجم (*Analytics*)، داده‌های دریافت شده به غیر از خود متن دریافت شده، برای مقاصد تحلیلی وارد کلیک‌هوس می‌شوند و چرخه پردازش داده به اتمام می‌رسد.

همزمان با دریافت داده‌ها، باید بتوان :

- انواع جستجوهای متنی را روی محتوای لحظه‌ای کانال‌ها درون الاستیک سرچ انجام داد.
- آمار لحظه‌ای داده‌ها توسط یک وب اپلیکیشن و با خواندن داده‌ها از ردیس، به کاربر نمایش داده شود.
- انواع گزارش‌ها پیچیده بااتصال سوپرست به کلیک‌هوس، در لحظه قابل تولید و نمایش باشد.

علاوه بر اینها، می‌خواهیم بتوانیم برخی مدل‌های پیش‌بینی کننده را با اتصال اسپارک به کاساندرا تولید کرده، گروه بندی خودکار (هشتگ زنی خودکار) و پیش‌بینی زمان ارسال پست بعدی هر کانال را هم انجام دهیم. (**این بخش دارای امتیاز اضافی خواهد بود**). بعد از ایجاد مدل پیش‌بینی هشتگ، این مدل به گام پیش‌پردازش اضافه خواهد شد که کیفیت برچسب‌زنی و استخراج کلمات کلیدی پست‌ها، ارتقا یابد.

هر چند تأکید اصلی پروژه بر استفاده از پیام‌رسان‌های داخلی مانند سروش، بله ، آی‌گپ و مانند آن‌ها است اما برای شروع کار می‌توانید از داده‌های توئیتر استفاده کنید و پس از ساختن سامانه اصلی، منبع دریافت داده آنرا تغییر دهید.

برای استفاده از داده‌های توئیتر، می‌توانید از این آموزش (https://bit.ly/2YOiN5U) استفاده کنید و کلیدهای زیر را برای اتصال به توئیتر به کار برید :

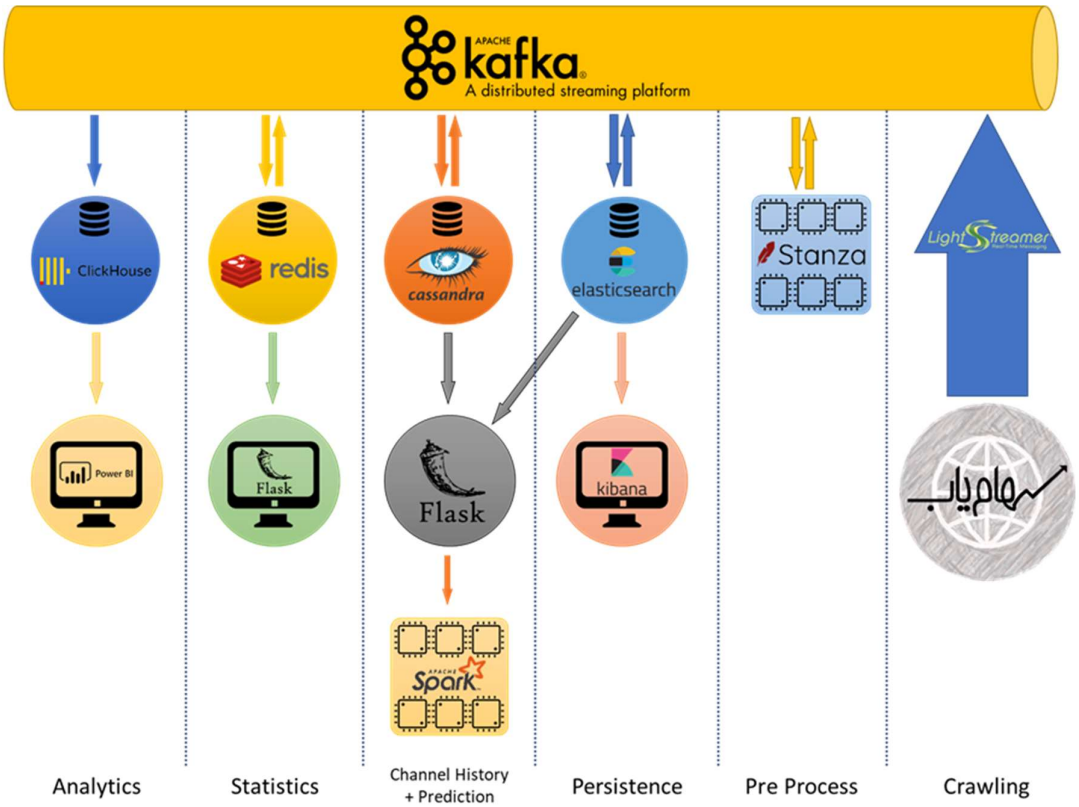
```
consumer_key = '*********'
consumer_secret = '***************'
access_token = '15257539-ERDMc7Ezn7t0tLmfBRRruyYpGmIsN43hsGSHdQS64'
access_secret = '1DH7FHDcqgHX3YxW2ZcvU91dkaZcogISXUevCw1PxScoQ'
```

در ادامه، هر یک از پنج گام پردازشی فوق و نیز الزامات کلی پروژه به تفصیل بیان خواهند شد.

## پیش‌نیازها و توضیحاتی در مورد ابزار و کتابخانه‌ها

برای هر گام از پروژه، با یک نرم‌افزار/دیتابیس کار خواهید که بهتر است آخرین نسخه آن‌ها را استفاده کنید. شالوده ارتباطی این سامانه، صف توزیع شده(کافکا) خواهد بود. **پیشنهاد ما استفاده از کافکا است اما میتوانید از RabbitMQ یا NSQ هم استفاده کنید**. تعداد اعضای هر تیم، بهتر است دو تا سه نفر باشد اما گروه‌های چهار نفره هم مجاز خواهد بود. بهتر است برای هماهنگی بیشتر، یک نفر را به عنوان مدیر تیم انتخاب کرده، هماهنگی و توزیع تسک‌ها و کارها را از طریق گیت‌لب/گیت‌هاب و از طریق مکانیزم برنچینگ و ایجاد ایشو انجام دهید.

شکل زیر شماتیک معماری این سیستم را که توسط یکی از تیم‌های سالهای گذشته این درس طراحی شده است نمایش میدهد که محوریت کافکا و نحوه تعامل بخش‌های مختلف آن به خوبی در آن قابل مشاهده است :



## روال پیشنهادی تقسیم کار

در این پروژه به مهارت‌ها و کارهای زیر نیاز است :

- خواندن اطلاعات از پیام رسان و ارسال لحظه‌ای آن‌ها به کافکا (و ساخت کانال‌های مختلف کافکا).
- پردازش اولیه متن و ذخیره اطلاعات استخراج شده در الاستیک سرچ و نمایش آن‌ها در یک داشبورد درون کیبانا.نیز ذخیره اطلاعات آماری درون ردیس و نمایش آن‌ها به کمک یک داشبورد وب که با فلسک می‌تواند پیاده‌سازی شود.
- ذخیره اطلاعات تاریخچه‌ای درون کاساندرا و ساخت یک مدل پیش‌بینی کننده زمان پست‌بعدی هر کانال و دسته بندی هر متن (هشتگ زنی خودکار) با اتصال اسپارک به کاساندرا.
- ذخیره اطلاعات تحلیلی درون دیتابیس کلیک‌هوس و اتصال آن به سوپرست و ساخت چندین داشبورد تحلیلی درون سوپرست

می‌توانید برای تقسیم کار بین اعضای تیم از بخش‌بندی فوق استفاده کنید.

## نحوه تحویل کار

گزارش نهایی پروژه توسط مدیر تیم در ایلرن به همراه آدرس ریپوزیتوری گیت پروژه(در صورت وجود)، آپلود خواهد شد. **هر فرد از اعضای تیم، گزارش آماده شده برای بخش خودش را در سامانه آپلود خواهد کرد** تا در صورت کم‌کاری یکی از اعضای تیم، فقط نمره آن فرد، تحت تأثیر قرار گیرد و نمره نهایی، براساس میزان تلاش و مشارکت هر عضو مستقل از بقیه تیم، داده شود. در جلسه تحویل آنلاین، هر نفر از اعضای تیم به صورت جداگانه کار انجام شده توسط خودش و گزارش آماده شده را تشریح کرده و تسک‌های انجام شده را توضیح خواهد داد. سپس با اجرای پروژه به صورت لوکال و به اشتراک گذاری صفحه نمایش، خروجی واقعی بخش مرتبط با خود را به دستیاران آموزشی نمایش خواهد داد.

استفاده از یک سرور (فیزیکی یا vps) و تحویل آنلاین پروژه، نمره امتیازی خواهد داشت.

## گام اول : دریافت اطلاعات و Preprocess

برای دریافت اطلاعات از پیام‌رسان‌ها، از خزشگرهایی که توسط یکی از اعضای تیم نوشته خواهد شد استفاده کنید. این اطلاعات به صورت مداوم از طریق برنامه‌ای که به صورت مداوم در حال اجراست و یا از طریق فراخوانی مداوم API ،به صورت جی‌سان وارد کانال *PreProcess* کافکا خواهد شد.

انتظار می‌رود با نوشتن یک بات و عضو کردن آن در کانالهای مختلف، به محض ارسال یک پست جدید در یک کانال، اطلاعات آن به سامانه پردازشی منتقل شود. کافی است عبارت «ساخت بات برای سروش/بله/آی‌گپ» را سرچ کنید تا بتوانید باتی برای خزش اطلاعات هر کانال طراحی کنید. بعد از ساخت این بات، لیستی از کانال‌ها تهیه کرده و این بات را به عضویت آن‌ها درآورید.

برای توئیت‌های داخلی میتوانید از روشهای مختلفی مانند فراخوانی API، Crawling و مانند آن استفاده کنید. داده‌های توئیتر نیز با فراخوانی API های استریمنگ آن، به راحتی قابل دریافت است.

با دریافت اطلاعات هر پست/ توئیت از طریق کانال *PreProcess*، فرآیند پردازش ما شروع می‌شود. ابتدا تایم استمپ زمان دریافت و یک UUID به عنوان شناسه منحصربفرد هر پست/توئیت به آن اضافه کنید. سپس هشتگ‌ها یا کلمات کلیدی آنرا استخراج کرده و به عنوان متادیتا به اطلاعات دریافت شده، اضافه کنید. اگر متن، حاوی لینک است، لینک‌های آن استخراج شده و درون یک ارایه جداگانه قرار گیرد. (متن اصلی را هیچ گاه تغییر نمیدهیم فقط اطلاعات مورد نیاز را استخراج و به صورت جداگانه ذخیره کنید)

برای استخراج کلمات کلیدی/ هشتگ، می‌توانید ایست‌واژه‌ها و افعال را حذف کنید، سپس کلماتی که tf/idf بالاتری دارند را به عنوان کلمه کلیدی در نظر بگیرید. توضیح اینکه هر پست می‌تواند یک یا چند هشتگ داشته باشد که آن‌ها را درون فیلد Hashtags ذخیره خواهید کرد. اما چه این هشتگ‌ها را داشته باشد چه نداشته باشد، **شما باید خودتان کلمات کلیدی را استخراج و درون فیلد Keywords ذخیره کنید.**

در این مرحله اگر متن دریافت شده حاوی کلمات زیر بود، **این کلمات حتماً به عنوان کلمات کلیدی باید درون آرایه Keywords قرار گیرند** :

- **بورس** - **اقتصاد** - **تحریم** - **دولت** - **حسن روحانی**
- **انتخابات** - **دلار** - **طلا** - **کرونا**
- **کوید ۱۹ (به هر شکل که نوشته شود)** - **تورم** - **دانشگاه**

در انتهای این مرحله یک json کامل از داده دریافت شده (داده‌های اصلی + متادیتای ایجاد شده ) تولید می‌شود که آماده ذخیره سازی و پردازش‌های بعدی است. این متن وارد کانال *persistence* در کافکا خواهد شد.

## گام دوم – persistence

در این مرحله، داده‌های دریافت شده مرحله قبل در الاستیک سرچ ذخیره می‌شوند.

دقت کنید که برای متون فارسی از *Persian Analyzer*[1] استفاده کنید. اگر بتوانید لیست ایست‌واژه‌ها و حتی *Tokenizer* را هم به صورت سفارشی (مثلاً استفاده از کتابخانه هضم در پردازش متون فارسی)، به الاستیک سرچ بدهید، امتیاز بیشتری خواهید گرفت.

داشبوردی در کیبانا طراحی کنید که موارد زیر را بتوان در آن مشاهده کرد :

- ابر کلمات یک کانال یا خبرگزاری خاص در یک بازه زمانی
- متن ده پست اخیری که دریافت شده است.
- تعداد پست‌های ارسال شده به ازای چند تا از کلمات کلیدی خاص که در مرحله قبل مشخص شده است در یک بازه زمانی .
- ده هشتگ بیشتر استفاده شده در پست‌های یک کانال خاص (یا تمام کانال‌ها) در یک بازه زمانی با تعداد تکرار هر هشتگ (یک نمودار ستونی) مثلاً هشتگ‌های بیشتر استفاده شده در یک روز اخیر .
- یک نمودار به انتخاب خودتان.

ضمناً در گزارش قید کنید که اگر به دنبال تمام پستهای حاوی یک کلمه خاص از یک خبرگزاری یا کانال خاص در یک بازه زمانی مشخص هستیم، چه دستوری باید بنویسیم. ( ویا یک هشتگ خاص یا یک کاربر خاص در توئیت‌ها)

اگر تعداد پستها/توئیت‌های ارسالی به ازای یک کلمه خاص را به ازای هر کانال/ یا یک هشتگ خاص در توئیت‌ها در یک بازه زمانی بخواهیم، چه دستوری باید استفاده کنیم. (این کلمه، میتواند هر کلمه‌ای در متن باشد و ممکن است جزء کلمات کلیدی هم نباشد)

[1] https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-lang-analyzer.html

## گام سوم – Channel/Hashtag History

در این مرحله، می‌خواهیم به کمک کاساندرا و مکانیزم ذخیره سازی سطرگسترده آن، تاریخچه زمانی هر کانال و هر هشتگ/کلمه کلیدی را ذخیره کنیم.

اگر کاربر نیاز داشت پستهای اخیر یک کانال یا یک هشتگ را ببیند، کافی است داده‌ها از این دو جدول کاساندرا، خوانده شده و به کاربر نمایش داده شود. با توجه به اینکه کاساندرا، هنگام ذخیره سازی، داده‌ها را به صورت مرتب (طبق تنظیماتی که در تعریف جدول آورده‌ایم)، ذخیره می‌کند و از طرفی، عملیات جوین و اتصال هم نداریم، سرعت بسیار بالایی در واکشی اطلاعات دارد.

- دقت کنید که در کاساندرا، تکرار داده‌ها یک اصل کاملاً پذیرفته شده است و به دنبال نرمال سازی نباشید.
- حداقل یک جدول برای کل پست‌ها (که بهتراست کلید هر سطر روز/ساعت دریافت هرپست باشد)، یک جدول برای هر کانال، یک جدول برای هر هشتگ/کلمه کلیدی نیاز خواهید داشت.
- کافی است فقط شناسه هر پست ذخیره شود. بعد از بازیابی اطلاعات مورد نیاز کاربر از کاساندرا، هنگام ارسال اطلاعات به کاربر، با دادن شناسه پست به الاستیک سرچ، اطلاعات کامل آنرا می‌توانید بازیابی کرده و به کاربر نشان دهید. (نوع جستجوی ids در الستیک برای همین منظور ایجاد شده است) یعنی دراین پروژه از کاساندرا بیشتر به عنوان یک اندیس سفارشی شده روی داده‌ها استفاده خواهیم کرد.

نکته : تمام این اطلاعات را الاستیک سرچ هم می‌تواند با سرعت بسیار بالا در اختیار ما قرار دهد اما هدف از این بخش، آشنایی عملی با کاساندرا و جدا کردن بخشهای مختلف منطقی سامانه از یکدیگر است.

در پایان این مرحله، داده‌ها وارد کانال *Statistics* می‌شود.

انواع دستوراتی که برای بازیابی پستها در یک ساعت اخیر، پستهای یک کانال در ۲۴ ساعت اخیر، پستهای مرتبط با یک هشتگ در بازه زمانی باید اجرا کنیم را هم در گزارش ذکر کنید.

آیا می‌توانیم اطلاعات آماری هر کانال ، هر هشتگ یا کل پستها را در یک بازه زمانی به کمک کاساندرا به دست آوریم ؟ مثلاً تمام پستهای روزانه یک کانال در یک هفته گذشته ؟ پستهای ذخیره شده در ماه گذشته ؟

## گام چهارم – Statistics

در این مرحله، اطلاعات آماری سامانه را به روز رسانی می کنیم.

به ازای هرکانال و هر هشتگ یک کلید در ردیس در نظر میگیریم و با دریافت یک کلید جدید، مقدار آنرا با یک جمع میکنیم. اما چون مثلاً بعد از گذشتن یک روز یا یک ساعت، پستهای قدیمی باید از آمار فعلی کسر شوند، بنابراین درطراحی کلیدهای ردیس دقت به خرج دهید. به ازای هر پست یا مطلب جدیدی که دریافت می‌کنید، چندین کلید را در ردیس باید به روز رسانی کنید.

راهنمایی : کلیدهایتان را به روز و ساعت مرتبط کنید و با آغاز هر ساعت جدید/ هر روز جدید، کلید جدیدی در نظر بگیرید.

در این مرحله باید بتوانید به سؤالات زیر به کمک ردیس که یک دیتابیس مقیم در حافظه بسیار سریع است جواب دهید :

- تعداد پستها/توئیت‌های ارسال شده یک کانال خاص در شش ساعت گذشته .
- تعداد کل پستها/توئیت‌های دریافت شده در یک بازه زمانی مثلاً روز گذشته .
- تعداد هشتگهای دریافت شده در یکساعت گذشته . (به صورت منحصر بفرد)
- آخرین هشتگ‌های دریافت شده . (یک لیست هزارتایی که با ورود داده‌های جدید، قدیمی‌ها حذف میشوند)
- آخرین پستها/توئیت‌های دریافت شده (یک لیست صدتایی مشابه فوق)

دقت کنید که تمام داده‌ها تا یک هفته گذشته باید در حافظه باشند و بعد از آن، باید به صورت خودکار توسط ردیس از حافظه حذف شوند .

یک وب اپلیکیشن با فلسک بنویسید که اطلاعات خواسته شده فوق را بتوان درون آن مشاهده کرد. با رفرش کردن صفحه در این اپلیکیشن، آمار آن باید به روز شود.

ردیس در این پروژه برای به روز رسانی آمار لحظه‌ای استفاده می‌شود که برای این آمارها، نیاز به کوئری زدن به دیتابیس‌های مختلف نداشته باشیم .

در پایان این مرحله، همان داده‌های دریافت شده یعنی پست جدید وارد کانال *Analytics* خواهد شد. در تمام این مراحل، داده‌های وارد شده به کانال دوم تا پنجم، همان داده‌های ایجاد شده در مرحله اول است.

## گام پنجم – Analytics

در آخرین گام از پروژه، اطلاعات آماری مورد نیاز برای تحلیل‌های آماری را درون دیتابیس **Clickhouse** ذخیره می‌کنیم.

توضیح اینکه کلیک‌هوس یک دیتابیس متن‌باز تحلیلی و بسیار سریع است که می‌توانید داده‌هایی که بعداً قرار است انواع گزارش‌گیری ها و تحلیل‌ها را روی آن‌ها انجام دهید، درون آن ذخیره کرده و انواع گزارش‌ها و تحلیل‌ها را روی هر حجمی از داده‌ها اعمال کنید . در حقیقت، به کمک این دیتابیس تحلیلی که داده‌ها را به صورت ستونی ذخیره می‌کند، نیاز به استفاده از انباره‌های داده کلاسیک را برطرف می‌کنیم و به کاربر این اجازه را می‌دهیم که هر گزارشی را با اعمال انواع فیلترها، روی هر حجمی از داده‌ها در زمانی بسیار کوتاه، مشاهده کند.

در این پروژه اطلاعات اصلی هر پست دریافت شده را درون کلیک‌هوس ذخیره می‌کنیم. البته نیازی به ذخیره متن هر پست نیست چون تحلیل‌های متنی را با الاستیک‌سرچ انجام خواهیم داد.

نکات زیر را درباره کلیک‌هوس در نظر بگیرید :

- می‌توانید کلاً از یک جدول استفاده کنید و تمام اطلاعات دریافت شده را درون آن ذخیره کنید. به دلیل مکانیزم ذخیره سازی ستونی کلیک‌هوس، فیلدهای خالی، کارآیی دیتابیس را کاهش نمی‌دهند. این امر نیاز به جوین را هم از بین می‌برد چون می‌توان تمام داده‌های مرتبط را در یک جدول ذخیره کرد (کلیک‌هوس از جوین پشتیبانی نمی‌کند)
- کلیدپارتیشن را هنگام ایجاد جدول (درون دستور ساخت جدول) با دقت انتخاب کنید چون به ازای هر پارتیشن، یک فایل ذخیره خواهد شد. بنابراین اگر کلید پارتیشن را آی‌دی هر پست بگیرید به ازای هر پست یک فایل ایجاد می‌شود و بعد از مدتی، تعداد زیاد فایلهای تولید شده، شما را به دردسر خواهد انداخت. بهتر است به ازای هر روز، یک پارتیشن درنظر بگیرید که به ازای پستهای هر روز، کلاً یک فایل ایجاد شود.
- از **dbeaver** برای کار با کلیک‌هوس می‌توانید استفاده کنید .

با انجام این مرحله، کار پردازش اطلاعات به اتمام می رسد.

## ساخت داشبوردهای مدیریتی

برای ساخت گزارش‌های تحلیلی و مدیریتی ، از آپاچی سوپرست (**Apache Superset**) استفاده کنید. کافی است سوپرست را به کلیک هوس متصل کرده ، انواع نمودارها و گزارش‌ها را به کمک آن رسم کنید.

توضیح اینکه با توجه به نیاز به تصویرسازی داده‌ها در پروژه‌های کلان‌داده، پروژه آپاچی سوپرست که بر پایه فلسک و پایتون بناشده است و به راحتی قابل تغییر و سفارشی شدن است، در این بنیاد شروع شد که اوایل سال ۲۰۲۱ نسخه ۱ آن رسما به بازار عرضه شد.

برای این پروژه، سه داشبورد مختلف به صورت زیر در نظر بگیرید :

- گزارش‌ها مرتبط با هشتگ‌ها/کلمات کلیدی
- گزارش‌ها مربوط به کانال‌ها/کاربران (در صورت استفاده از توئیت‌ها)
- گزارش‌ها عمومی سامانه مانند آمار کلی دریافت اطلاعات در یک روز و یک ساعت گذشته .
- گزارشهای مرتبط با یک کانال خاص / یک هشتگ خاص

برای هر داشبورد، از تمامی نمودارهای سوپرست می‌توانید استفاده کنید. مهم این است که یک داشبورد تحلیلی و مناسب ایجاد کنید که با یک نگاه به آن، بتوان اطلاعات مناسبی دریافت کرد.

## ساخت یک مدل پیش‌بینی کننده با اسپارک - بخش امتیازی

**توضیح :** انجام این بخش دارای امتیاز اضافه خواهد بود و انجام آن، اختیاری خواهد بود .

با اتصال اسپارک به کاساندرا و استفاده از بخش MLIB آن، دو مدل برای پیش‌بینی موارد زیر بسازید :

- **پیش‌بینی زمان ارسال پست بعدی** یک کانال با دادن یک زمان خاص در یک روز خاص از هفته . مثلاً ساعت هشت روز جمعه را به مدل میدهیم و انتظار داریم زمان ارسال پست بعدی به دقیقه را به ما بدهد.
- **پیش‌بینی هشتگ‌های یک پست/توئیت** . به ازای هر پست/توئیت و کلمات موجود در آن، کلمات کلیدی آن توسط این مدل، پیش‌بینی شود. البته برای این منظور، ابتدا باید پستها/توئیت‌های زیادی که خود حاوی هشتگ باشند را دریافت کنید و سپس مدل را طوری آموزش دهید که با دیدن یک مجموعه کلمات (یعنی هر پست) ، یک یا چند کلمه پیشنهادی برای آن ، به عنوان نتیجه برگرداند.

می‌توانید از هر روش مکاشفه‌ای که بهبود دقت مدل‌ها کمک کند، استفاده کنید .

# دو راه حل نمونه

با هدف آشنایی بیشتر علاقه‌مندان به حوزه مهندسی داده و مباحث مرتبط با طراحی زیرساخت‌های پردازش داده، دونمونه از راه حل‌ها و پروژه‌های ارائه شده با کسب اجازه از دانشجویان ارائه دهنده، در ادامه این شرح پروژه آمده است.

امید است علاقه‌مندان با بررسی گزارش و اجرای کدهایی که آدرس آنها در ریپوزیتوری‌های گیت در گزارش‌ها آمده است، به یک دید شهودی و عملی از طراحی یک خط پردازش داده مبتنی بر کافکا دست پیدا کنند.

لازم به ذکر است که پروژه نمونه اول، برای دریافت لحظه‌ای داده‌ها، از خواندن پست‌های کانال‌های تلگرامی و پروژه نمونه دوم از خواندن توئیت های فارسی توئیتر استفاده کرده است.

به نام خدا

**دانشگاه تهران**

**دانشکده مهندسی برق و کامپیوتر**

# تحلیل‌ها و سیستم‌های داده‌های حجیم

**پروژه پایانی**

| | |
|---|---|
| نام و نام خانوادگی | علیرضا حیدری – امیرپاشا معتمد |
| شماره دانشجویی | ۸۱۰۱۹۹۵۴۲ - 810198149 |
| ریپوزیتوری های پروژه | https://github.com/smbanaie/bd-project-1400<br>(آدرس اصلی پروژه در انتهای گزارش کار آمده است ) |

## گام اول: دریافت اطلاعات و پیش‌پردازش

برای انجام این پروژه، ما از پیام‌های موجود در گروه‌های عمومی و کانال‌های تلگرام استفاده می‌کنیم. برای این کار از امکانات کتابخانه Telethon استفاده کرده و اطلاعات را استخراج می‌کنیم.

در گام اول باید api_id و api_hash مربوط به اکانت خود را از آدرس https://my.telegram.org دریافت کنیم. سپس با ساخت یک کلاینت جدید امکان دسترسی به تلگرام و اطلاعات را خواهیم داشت.

```
api_id = 6502227
api_hash = 'a70e206c9d4e0160a21d058488c5524c'
phone = '+989372409384'

client = TelegramClient(phone, api_id, api_hash)
```

برای سهولت کار در مراحل بعدی کلاسی با نام client ایجاد کردیم تا ارتباط با تلگرام را مدیریت کند. در ادامه جزئیات این کلاس و توابع آن شرح داده خواهد شد:

1. تابع connect:

این تابع وظیفه اتصال به تلگرام را برعهده دارد. چنانچه برای اولین بار توسط یک کاربر (سیستم) اجرا شود یک کد تایید به اکانت تلگرام ارسال می‌شود تا از ورودهای ناخواسته جلوگیری شود. در دفعات بعدی کاربر موردنظر برای تلگرام authorized شده خواهد بود.

```
    @classmethod
    def connect(cls):
        cls.client.connect()
        if not cls.client.is_user_authorized():
            cls.client.send_code_request(cls.phone)
            cls.client.sign_in(cls.phone, input('Enter the code: '))

        return cls.client
```

2. تابع join_channels:

این تابع لیستی از Idهای کانال‌ها و گروه‌های عمومی تلگرام را گرفته و کاربر را به عضویت آن‌ها در می‌آورد. از آنجایی که تعداد کانال‌ها و گروه‌ها زیاد است این تابع با عضویت اتوماتیک در آن‌ها کار ما را بسیار ساده خواهد کرد.

```
@staticmethod
def join_channels():
    with open('channels.txt') as f:
        channels = f.read().splitlines()

    Client.connect()
    for channel in channels:
        try:
            Client.client(JoinChannelRequest(channel))
        except:
            continue
```

3. تابع get_channels:

این تابع لیستی از کانال‌ها و گروه‌هایی که کاربر در آن‌ها عضویت دارد را برمی‌گرداند و در مرحله دریافت اطلاعات استفاده خواهد شد.

```
@classmethod
def get_channels(cls):
    req = GetDialogsRequest(offset_date=None, offset_id=0, offset_peer=InputPeerEmpty(), limit=10000, hash = 0)
    channels_list = cls.client(req)
    channels_list = [PeerChannel(chat.id) for chat in channels_list.chats]

    return channels_list
```

اسکریپت fetch_data.py به دریافت اطلاعات از تلگرام می‌پردازد. تابع newMessageListener روی رخداد پیام جدید به صورت آسنکرون تنظیم می‌شود و به محض دریافت پیام روی هر یک از کانال‌هایی که کاربر عضو آن است، پیام را به سیستم منتقل می‌کند. سپس این پیام به یک دیکشنری پایتون تبدیل می‌شود و نام کانال یا کاربر ارسال کننده نیز به عنوان یک زوج کلید-مقدار جدید به آن اضافه می‌شود. پس از این کار پیام دریافت شده وارد اولین کانال کافکا یعنی preprocess می‌شود. برای کار با کافکا در محیط پایتون از امکانات کتابخانه kafka-python استفاده می‌کنیم. یک producer جدید ساخته و ساختار پیام را روی json تنظیم می‌کنیم. سپس به وسیله تابع send پیام مورد نظر را روی کانال preprocess قرار می‌دهیم. همچنین استفاده از تابع flush() به ما اطمینان خواهد داد که اطلاعات به درستی درون این کانال قرار خواهند گرفت.

```
16  producer = KafkaProducer(bootstrap_servers=['localhost:9092'], api_version=(0, 10), \
17      value_serializer=lambda v: json.dumps(v, default=date_format).encode('utf-8'))
18
19  @client.on(events.NewMessage(chats = channels_list))
20  async def newMessageListener(event):
21      new_message = event.message
22
23      channel = await client.get_entity(new_message.peer_id.channel_id)
24      new_message = new_message.to_dict()
25      new_message['sender_name'] = channel.title
26
27      producer.send('preprocess', new_message)
28      producer.flush()
```

همچنین یک اسکریپت مشابه به نام fetch_history.py ایجاد کردیم تا داده‌های کانال‌ها در روزهای اخیر را نیز وارد سیستم کند. به این ترتیب فرآیند ورودی داده ابتدا با اجرای این کد آغاز می‌شود و پس از دریافت حجم مناسبی از داده‌ها (به طور مثال ۱۰ روز اخیر) با استفاده از اسکریپت fetch_data.py، ادامه‌ی دریافت به صورت زنده و realtime ادامه می‌یابد.

در مرحله بعدی و در گام دوم این قسمت به پیش‌پردازش پیام‌ها خواهیم پرداخت. اسکریپت preprocess.py به طور مداوم روی کانال preprocess کافکا گوش می‌کند و در صورت رسیدن پیام جدید آن را برداشته و پیش‌پردازش را آغاز می‌کند.

```
consumer = KafkaConsumer('preprocess', auto_offset_reset='latest', bootstrap_servers=['localhost:9092'], \
 api_version=(0, 10), consumer_timeout_ms=1000, value_deserializer=lambda m: json.loads(m.decode('utf-8')))
```

در ابتدا با استفاده از فیلد id موجود در هر پیام و timestamp فعلی یک UUID جدید ایجاد شده و به پیام اضافه می‌شود. این UUID برای هر پیام یکتا خواهد بود.

```
new_item['UUID'] = str(uuid.uuid1(new_item['id']))
```

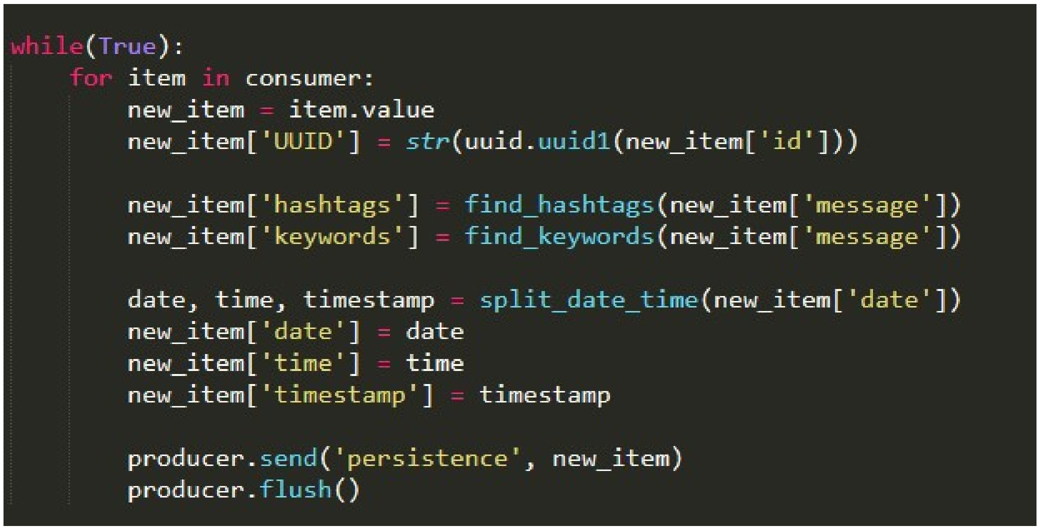
سپس درون تابع find_hashtags تمامی هشتگ‌های موجود در متن پیام به کمک regex استخراج شده و درون فیلد hashtags کنار پیام به صورت لیست ذخیره می‌شود.

```
def find_hashtags(message):
    hashtags = re.findall(r"#(\w+)", message)
    return list(set(hashtags))
```

در ادامه تابع find_keywords به کمک کتابخانه‌های hazm و yake کلمات کلید موجود در متن پیام را استخراج می‌کند. در این قسمت ابتدا تمام لینک‌ها و ایمیل‌های موجود در متن پیام کنار گذاشته می‌شوند و سپس متن باقی مانده normalize می‌شود. در ادامه stopwordها و کاراکترهای غیر الفبایی از متن حذف شده و متن آماده استخراج کلمات کلیدی می‌شود. به وسیله امکانات کتابخانه yake کلمات کلیدی استخراج شده و به صورت لیست در کنار پیام ذخیره می‌شوند.

```
def clean_message(message):
    normalizer = Normalizer()
    stop_words = []
    special_words = []

    with open("./preprocess/stopwords.txt", "r") as swf:
        sw = swf.read()
        stop_words = sw.split("\n")
    swf.close()

    message = re.sub(r'http[s]?://\S+', '', message)
    message = re.sub(r'[A-Za-z0-9]+@[a-zA-Z].[a-zA-Z]+', '', message)
    message = normalizer.normalize(message)
    message = word_tokenize(message)
    message = [word for word in message if word not in stop_words and word.isalpha()]
    special_words = [word for word in message if word in ['رو ج ن ک , ' پر بن ' ا ق ت ل ط, ' د ل ز, ' ی ر و, ' ح ر ی,
    message = ' '.join(message)

    return message, special_words


def find_keywords(message):
    message, special_words = clean_message(message)

    kw_extractor = yake.KeywordExtractor()
    custom_kw_extractor = yake.KeywordExtractor(n=1, features=None, top=10)
    keywords = custom_kw_extractor.extract_keywords(message)
    keywords = [x[0] for x in keywords if x[1] > 0.09] + special_words
    return list(set(keywords))
```

در نهایت و پس از انجام مراحل بالا تاریخ، زمان و timestamp پیام نیز با فرمت استاندارد به آن افزوده می‌شود و پیام‌ها آماده ذخیره‌سازی و پردازش می‌شوند و درون کانال persistence کافکا قرار می‌گیرند.



```
while(True):
    for item in consumer:
        new_item = item.value
        new_item['UUID'] = str(uuid.uuid1(new_item['id']))

        new_item['hashtags'] = find_hashtags(new_item['message'])
        new_item['keywords'] = find_keywords(new_item['message'])

        date, time, timestamp = split_date_time(new_item['date'])
        new_item['date'] = date
        new_item['time'] = time
        new_item['timestamp'] = timestamp

        producer.send('persistence', new_item)
        producer.flush()
```

## گام دوم: persistence

در این گام ابتدا اطلاعات پیش‌پردازش شده گام قبلی از روی کانال persistence کافکا خوانده می‌شود و درون الستیک‌سرچ ذخیره می‌شود.

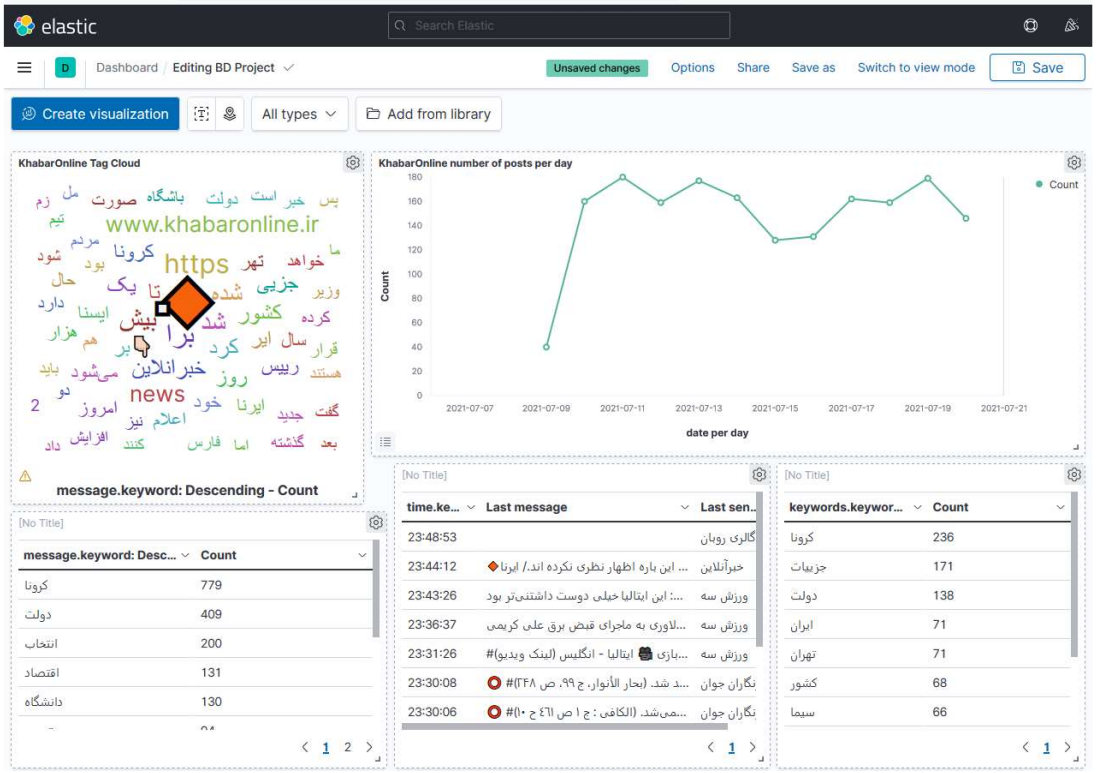
برای این کار ابتدا ایندکس جدیدی با نام تلگرام روی الستیک‌سرچ ساخته می‌شود (در صورت عدم وجود). برای فیلد message این ایندکس از analyzer فارسی parsi استفاده می‌شود و کلمات موجود در متن پیام‌ها پس از tokenize شدن و عبور از فیلترهای مختلف (مثل حذف stopwords) درون فیلد message.keyword در کنار متن اصلی پیام ذخیره می‌شود.

```
header = {"content-type":"application/json"}
new_index = {
    "mappings": {
        "properties": {
            "message": {
                "type": "text",
                "fields" : {
                    "keyword" : {
                    "type" : "text",
                    "analyzer": "parsi",
                    "fielddata": "true"
                    }
                }
            }
        }
    }
}


x = requests.head('http://localhost:9200/telegram')
if x.status_code != 200:
    requests.put('http://localhost:9200/telegram', data = json.dumps(new_index), headers=header)
```

پس از ساخت ایندکس و اعمال تنظیمات بالا، پیام‌ها یک به یک از روی کانال persistence کافکا خوانده می‌شود و با استفاده از تابع post کتابخانه requests درون الستیک‌سرچ قرار می‌گیرد. در نهایت نیز هر پیام به کانال channel_history کافکا ارسال می‌شود.

```
while(True):
    for item in consumer:
        new_item = item.value
        x = requests.post('http://localhost:9200/telegram/_doc/{}'.format(new_item['UUID']), \
            data = json.dumps(new_item), headers=header)

        producer.send('channel_history', new_item)
        producer.flush()
```

سپس بر روی کیبانا داشبورهای لازم را ایجاد می‌کنیم:




در این داشبورد، هشتگ‌های مربوط به ۱۵ روز اخیر کانال خبرآنلاین به صورت tagcloud نمایش داده شده است. همچنین تعداد پست‌های روزانه‌ی این کانال در ۱۰ روز اخیر نمایش داده شده است. آخرین ۱۰ پیام دریافتی از کل کانال ها نیز به صورت یک Data Table نمایش داده شده است. همچنین ۱۰ هشتگ پرتکرار در کانال خبرآنلاین نیز در پایین سمت راست قابل مشاهده است. کلمات کلیدی مشخص شده مانند (کرونا، دولت، اقتصاد و ...) نیز به همراه تکرارشان در پیام‌های دریافت در قابل یک Data Table در پایین سمت چپ نمایش داده است. همانطور که مشاهده می‌شود کلمه‌ی کرونا هم در کانال خبرآنلاین و هم در کلمات کلیدی مشخص شده، بیشترین تعداد تکرار را داشته است. برای پیدا کردن پست‌های مورد نظر از یک کانال خاص و یا یک هشتگ خاص و ... از دستور search در الاستیک سرچ استفاده می‌شود. برای شمارش نیز از دستور count استفاده می‌شود.

## گام سوم: channel_history

در این گام ابتدا یک keyspace جدید با نام تلگرام روی کاساندرا می‌سازیم. سپس سه جدول messages، channels و keywords را درون آن ایجاد می‌کنیم.

در جدول messages تاریخ را به عنوان partition key و UUID پیام را به همراه زمان ارسال آن به عنوان clustering key در نظر می‌گیریم. به این ترتیب تمامی پیام‌های یک روز داخل یک پارتیشن قرار می‌گیرند و براساس زمان ارسال در آن روز مرتب خواهند بود.

در جدول id channels هر کانال (channel_id) را به عنوان partition key و UUID پیام‌ها به همراه timestamp ارسال آن‌ها را به عنوان clustering key در نظر می‌گیریم. در این جدول پیام‌های مربوط به هر کانال داخل یک پارتیشن مجزا قرار می‌گیرند و بر اساس timestamp مرتب می‌شوند.

ساختار جدول keywords نیز به مانند جدول channels است و هر کلمه کلیدی به عنوان کلید پارتیشن در نظر گرفته می‌شود. سپس پیام‌هایی که حاوی آن کلمه کلیدی باشند در پارتیشن مربوطه به صورت مرتب‌شده براساس timestamp ذخیره می‌شوند.

```
cluster = Cluster()
session = cluster.connect("telegram")

query = "CREATE TABLE IF NOT EXISTS messages ( date text, time text, post_id text, \
primary key(date, time, post_id) );"
result = session.execute(query)

query = "CREATE TABLE IF NOT EXISTS channels ( channel_id text, ts float, post_id text, \
primary key(channel_id, ts, post_id)) WITH CLUSTERING ORDER BY (ts DESC, post_id DESC);"
result = session.execute(query)

query = "CREATE TABLE IF NOT EXISTS keywords ( keyword text, ts float, post_id text, \
primary key(keyword, ts, post_id)) WITH CLUSTERING ORDER BY (ts DESC, post_id DESC);"
result = session.execute(query)
```

در ادامه و پس از ساخته شدن جداول بالا، پیام‌ها از کانال channel_history کافکا خوانده‌شده و اطلاعات آن‌ها در جداول مربوط به خود ذخیره و در نهایت بدون تغییر به کانال statistics وارد می‌شوند.

```
insert_messages_query = "INSERT INTO messages (date, time, post_id) VALUES (%s, %s, %s)"
insert_channels_query = "INSERT INTO channels (channel_id, ts, post_id) VALUES (%s, %s, %s)"
insert_keywords_query = "INSERT INTO keywords (keyword, ts, post_id) VALUES (%s, %s, %s)"

while(True):
    for item in consumer:
        new_item = item.value
        print('new item recieved')

        session.execute(insert_messages_query, (new_item['date'], new_item['time'], new_item['UUID']))
        session.execute(insert_channels_query, (str(new_item['peer_id']['channel_id']), new_item['timestamp'], new_item['UUID']))

        for kw in (new_item['hashtags'] + new_item['keywords']):
            session.execute(insert_keywords_query, (kw, new_item['timestamp'], new_item['UUID']))

        producer.send('statistics', new_item)
        producer.flush()
```

برای بازیابی پست‌ها در یک ساعت اخیر ( به طور مثال بین ساعت ۵ تا ۶ روز ۱۴ ژولای ۲۰۲۱) از دستور زیر استفاده می‌کنیم:

```
select * from messages WHERE date='2021-07-14' and time>'05:00:00' and time<'06:00:00;'
```

همچنین برای دریافت پیام‌های یک کانال خاص در ۲۴ ساعت اخیر از دستور:

```
Select * from channels WHERE channel_id = '1008260193' AND ts>[timestamp]-24*3600 and ts<[timestamp]
```

که به جای timestamp، زمان فعلی را قرار می‌دهیم.

برای دریافت پست‌های اخیر هشتگ نیز از دستور زیر استفاده می‌کنیم. با توجه به اینکه clustering key به صورت نزولی در نظر گرفته شده است، نتایج نیاز به مرتب سازی ندارد:

```
Select * from keywords WHERE keyword = [keyword] LIMIT 10;
```

که به جای keyword، کلمه مورد نظر را قرار می‌دهیم.

همچنین آمار به صورت زیر قابل استخراج است:

به طور مثال تعداد پست‌های ۲۴ ساعت اخیر یک کانال:

```
Select count(*) from channels WHERE channel_id = '1008260193' AND ts>[timestamp]-24*3600 and ts<[timestamp]
```

## گام چهارم: statistics

در این گام پیام‌ها از کانال statistics کافکا خوانده و اطلاعات آماری آن‌ها درون ردیس ذخیره می‌شود. تمامی کلیدهایی که در ردیس ذخیره می‌شوند به زمان ارسال پیام‌ها مرتبط شده‌اند و پس از گذشت هفت روز از زمان دریافت حذف می‌شوند.

با دریافت هر پیام جدید ابتدا یک زوج کلید مقدار جدید به صورت (post:sender_name#time, 1) ایجاد می‌شود. در این زوج مرتب کلید از سه قسمت تشکیل شده است. کلمه پست نشان می‌دهد این کلید مربوط به یک پیام است. Sender_name نام کانالی که پیام در آن منتشر شده را مشخص می‌کند و time زمان ارسال پست را نشان می‌دهد (برحسب تاریخ و ساعت). همچنین مقدار این زوج مرتب نشان‌دهنده تعداد پیام‌های یک کانال در تاریخ و ساعت مشخص است و با هر بار دریافت پیام در آن تاریخ و ساعت مقدارش افزایش می‌یابد.

```
def channels_number_of_posts(name, date):
    ck = 'post:{}#{}'.format(name, date.split(':')[0])
    r.incr(ck)
    r.expire(ck, datetime.timedelta(days=7))
```

سپس کلید دیگری تحت عنوان recent_posts ساخته می‌شود. مقدار این کلید لیستی با طول 100 از اخرین پیام‌های دریافت شده است. هر بار پیام جدید دریافت می‌شود، متن آن در ابتدای این لیست قرار می‌گیرد و درصورتی که طول لیست بیشتر از 100 شود عنصر آخر آن (قدیمی‌ترین پیام موجود در لیست) به صورت خودکار حذف می‌شود.

```
def recent_posts(post):
    r.lpush('recent_posts', post)
    r.ltrim('recent_posts', 0, 99)
```

به طور مشابه دو قسمت قبل را برای کلمات کلیدی موجود در هر پیام نیز تکرار می‌کنیم.

```
def keywords_number(keywords, date):
    for kw in keywords:
        ck = 'keyword:{}#{}'.format(kw, date.split(':')[0])
        r.incr(ck)
        r.expire(ck, datetime.timedelta(days=7))

        r.lpush('recent_keywords', kw)
        r.ltrim('recent_keywords', 0, 999)
```

در نهایت نیز پیام‌ها به کانال analytics کافکا ارسال می‌شوند.

```
while(True):
    for item in consumer:
        new_item = item.value
        dt = new_item['date'] + "T" + new_item['time']

        channels_number_of_posts(new_item['sender_name'], dt)
        keywords_number(new_item['hashtags'] + new_item['keywords'], dt)
        recent_posts(new_item['message'])

        producer.send('analytics', new_item)
        producer.flush()
```

در ادامه اطلاعات آماری خواسته شده از redis استخراج و به کمک فلسک نمایش داده می‌شوند.

ابتدا به کمک api و کوئری زیر تعداد پیام‌های هر کانال در یک بازه زمانی که به عنوان ورودی توسط کاربر قابل تنظیم است نمایش داده می‌شود:

```
@app.route('/redis/channels/<time_range>/')
def channels_posts(time_range):
    now = datetime.datetime.now()
    now = [now - datetime.timedelta(hours=x) for x in range(0, int(time_range))]
    keys = ['post:[\u0621-\u0628\u062A-\u063A\u0641-\u0642\u0644-\u0648\u064E- \
    \u0651\u0655\u067E\u0686\u0698\u06A9\u06AF\u06BE\u06CC]*#{}' \
    .format(x.strftime("%Y-%m-%dT%H")) for x in now]
    result = {}

    for key_pattern in keys:
        for key in r.keys(key_pattern):
            try:
                k = key.split(':')[1]
                k = k.split('#')[0]
                result[k] = result.get(k, 0) + int(r.get(key))
            except:
                continue
    return json.dumps({"total_number_of_posts": result})
```

در api و کوئری بعدی نیز مشابه قسمت قبل تعداد کلمات کلیدی دریافت شده (به تفکیک کلمه) در یک بازه زمانی مشخص نشان داده می‌شود.

```
@app.route('/redis/keywords/<time_range>/')
def keywords(time_range):
    now = datetime.datetime.now()
    now = [now - datetime.timedelta(hours=x) for x in range(0, int(time_range))]
    keys = ['keyword:[\u0621-\u0628\u062A-\u063A\u0641-\u0642\u0644-\u0648\u064E- \
    \u0651\u0655\u067E\u0686\u0698\u06A9\u06AF\u06BE\u06CC]*#{}' \
    .format(x.strftime("%Y-%m-%dT%H")) for x in now]
    result = {}

    for key_pattern in keys:
        for key in r.keys(key_pattern):
            try:
                k = key.split(':')[1]
                k = k.split('#')[0]
                result[k] = result.get(k, 0) + int(r.get(key))
            except:
                continue
    return json.dumps({"total_number_of_keywords": result})
```

کوئری و api بعدی نیز مجموع کل پیام‌های ارسال شده در کانال‌های مختلف تلگرام را در یک بازه زمانی مشخص نشان می‌دهد.

```
@app.route('/redis/total_posts/<time_range>/')
def total_posts(time_range):
    now = datetime.datetime.now()
    now = [now - datetime.timedelta(hours=x) for x in range(0, int(time_range))]
    keys = ['post:[\u0621-\u0628\u062A-\u063A\u0641-\u0642\u0644-\u0648\u064E-\
    u0651\u0655\u067E\u0686\u0698\u06A9\u06AF\u06BE\u06CC]*#{}'
    .format(x.strftime("%Y-%m-%dT%H")) for x in now]

    result = 0
    for key_pattern in keys:
        for key in r.keys(key_pattern):
            try:
                result = result + int( r.get(key) )
            except:
                continue
    return json.dumps({"total_number_of_posts": result})
```

در نهایت نیز به وسیله کوئری‌ها و apiهای زیر لیست 100 پیام آخر دریافت شده و 1000 کلمه کلیدی اخیر نمایش داده می‌شود.

```
@app.route('/redis/recent_posts/')
def recent_posts():
    result = r.lrange('recent_posts', 0, -1)
    return json.dumps({"recent_posts": result})


@app.route('/redis/recent_keywords/')
def recent_keywords():
    result = r.lrange('recent_keywords', 0, -1)
    return json.dumps({"recent_keywords": result})
```

نتایج حاصل از این فاز در فرانت اند پروژه به صورت زیر قابل نمایش است:


قابلیت تنظیم بازه و همچنین به روزرسانی اطلاعات در فرانت اند تعبیه شده است که در نسخه‌ی لایو این پروژه قابل دسترسی است.

## گام پنجم: analytics

در این گام اطلاعات را از روی کانال analytics کافکا خوانده و درون clickhouse ذخیره می‌کنیم. برای این کار ابتدا دو جدول channels و keywords را به صورت زیر ایجاد می‌کنیم.

```
create_channels_query = '''
    CREATE TABLE IF NOT EXISTS telegram.channels
        (
            channel_id String,
            sender_name String,
            UUID UUID,
            views Int32,
            timestamp Float32,
            date DateTime
        )
        ENGINE = MergeTree
        PARTITION BY date ORDER BY (timestamp)
'''

create_keywords_query = '''
    CREATE TABLE IF NOT EXISTS telegram.keywords
        (
            keyword String,
            sender_name String,
            timestamp Float32,
            date DateTime
        )
        ENGINE = MergeTree
        PARTITION BY date ORDER BY (timestamp)
'''

client.execute(create_channels_query)
client.execute(create_keywords_query)
```

با این کار به‌ازای هر روز دو فایل، یکی برای اطلاعات کانال‌ها و دیگری برای کلمات کلیدی، ساخته می‌شوند. محتوای این فایل‌ها بر اساس timestamp دریافت پیام‌ها مرتب خواهد بود.

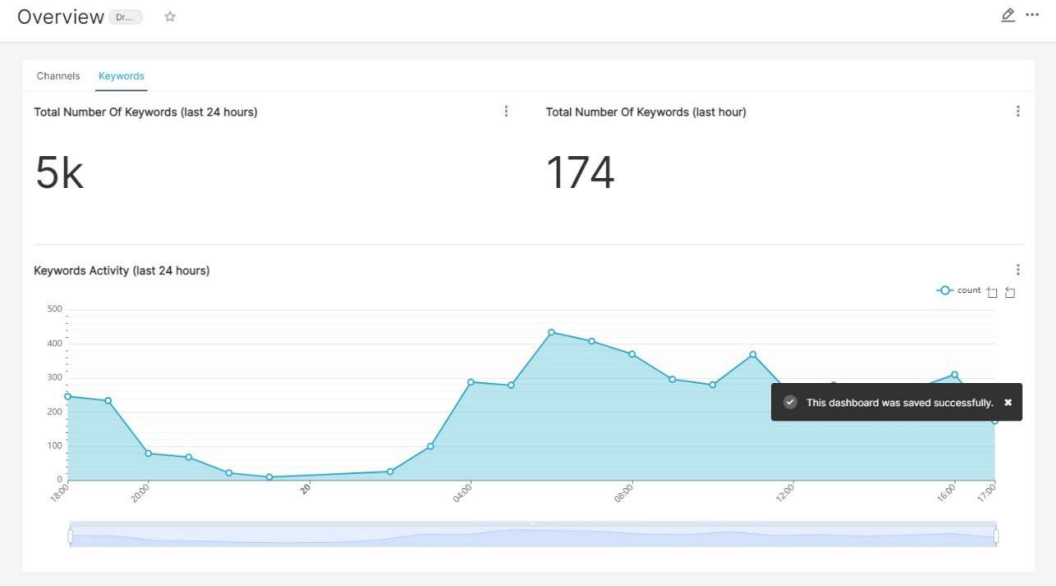
```
while(True):
    for item in consumer:
        new_item = item.value

        client.execute(
            'INSERT INTO telegram.channels (channel_id, sender_name, UUID, views, timestamp, date) VALUES',
            [( str(new_item['peer_id']['channel_id']), new_item['sender_name'], new_item['UUID'], new_item['views'], \
            new_item['timestamp'], datetime.datetime.strptime(new_item['date'] + 'T' + new_item['time'], "%Y-%m-%dT%H:%M:%S") )]
        )

        for kw in new_item['hashtags'] + new_item['keywords']:
            client.execute(
                'INSERT INTO telegram.keywords (keyword, sender_name, timestamp, date) VALUES',
                [( kw, new_item['sender_name'], new_item['timestamp'], \
                datetime.datetime.strptime(new_item['date'] + 'T' + new_item['time'], "%Y-%m-%dT%H:%M:%S") )]
            )
```

در نهایت سه داشبورد تحلیل داخل سوپرست می‌سازیم. این داشبوردها اطلاعات تحلیلی و مدیریتی سیستم را به ما نشان می‌دهند.

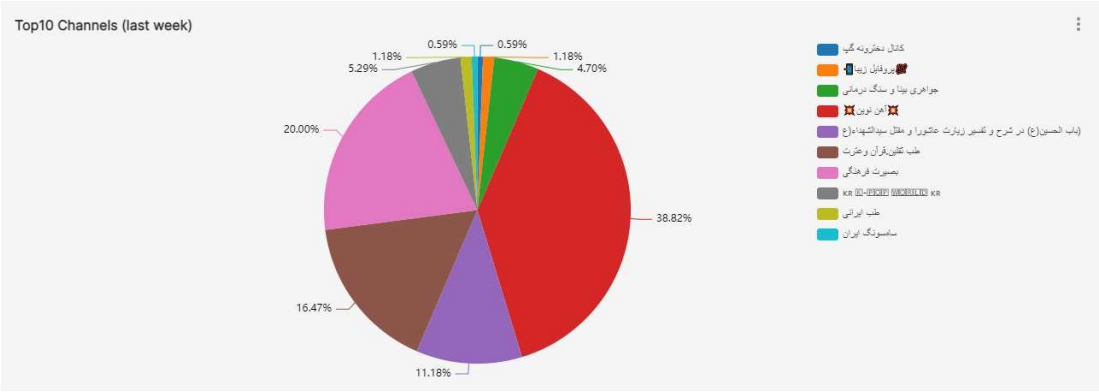
در نهایت سه داشبورد تحلیل داخل سوپرست می‌سازیم. این داشبوردها اطلاعات تحلیلی و مدیریتی سیستم را به ما نشان می‌دهند.

داشبورد اول با نام overwiew نمای کلی از سامانه در روز و ساعت گذشته را به ما نشان می‌دهد. این داشبورد شامل دو تب مجزاست. در تب اول آمار کلی دریافت پیام‌ها در روز و ساعت گذشته قابل مشاهده است. همچنین جریان دریافت داده در روز گذشته به تفکیک ساعت نیز قابل مشاهده است. با بررسی این جریان می‌توان ساعات پرکار و کم‌کار سامانه در روز گذشته را مشاهده نمود.


در تب دوم نیز همین اطلاعات را برای کلمات کلیدی می‌توان مشاهده نمود.




داشبورد دوم با نام Channels/Keywords اطلاعات کلی مربوط به کانال‌ها و هشتگ‌ها را به ما نشان می‌دهد. این داشبورد نیز دارای دو تب مجزاست. در تب اول 10 کانالی که بیشترین پیام‌ها را در طول هفته گذشته داشته‌اند به همراه درصد پیام‌های آن‌ها در یک نمودار دایره‌ای نشان داده شده است. در نمودار دیگری تعداد پیام‌های هر کانال در هفته گذشته به تفکیک نشان داده شده است.



Channels (last week)

Show All entries  Search 31 records...

| sender_name | COUNT(*) | %Percent |
|---|---|---|
| خبرآنلاین | 1.13k | 22.470% |
| باشگاه خبرنگاران جوان | 911 | 18.115% |
| خبرفوری | 910 | 18.095% |
| کانال آخرین خبر | 806 | 16.027% |
| ورزش سه | 421 | 8.371% |
| گالری روبان | 280 | 5.568% |
| مسیلۀ سبیقی | 132 | 2.625% |
| بعضاتن اشک بر امام حسین ع مادرش فاطمه س است. | 79 | 1.571% |
| 😁 روانشناسی شدن | 72 | 1.432% |
| 💥آهن اوین💥 | 66 | 1.312% |
| Totals | 5.03k | |

1 2 3 4

در تب دوم ابتدا کلمات کلیدی و هشتگ‌های پرتکرار هفته به صورت ابر کلمات نشان داده شده است. همچنین در نمودار دیگری تعداد هر هشتگ (کلمه کلیدی) در طول هفته گذشته به صورت جدول نشان داده شده است.

Hottest Keywords (last week)

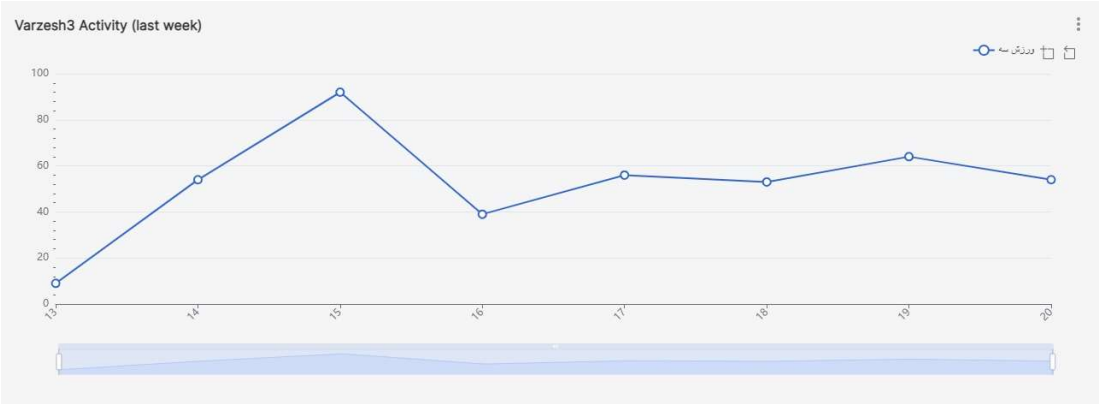

Keywords

Show All entries  Search 6797 records...

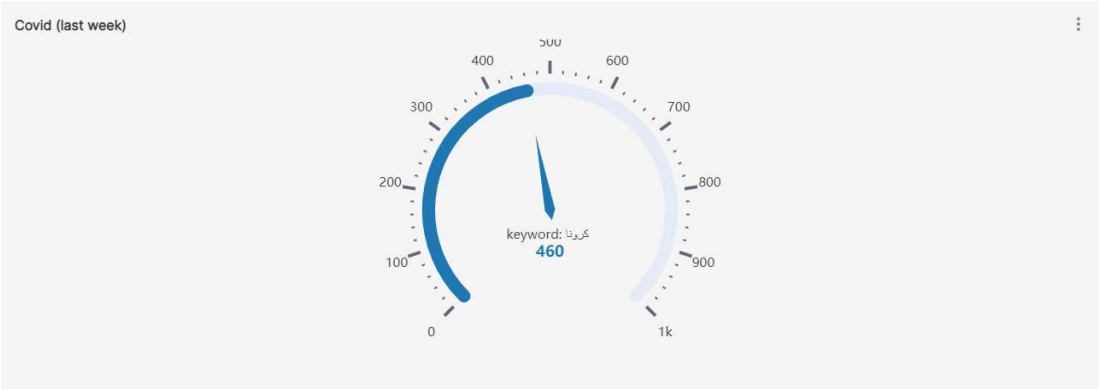
| keyword | COUNT(*) | %Percent |
|---|---|---|
| کرونا | 460 | 1.521% |
| ایران | 228 | 0.754% |
| دولت | 225 | 0.744% |
| خبرآنلاین | 192 | 0.635% |
| کشور | 150 | 0.496% |
| استقلال | 135 | 0.446% |
| تهران | 130 | 0.430% |
| پرسپولیس | 126 | 0.417% |
| سال | 126 | 0.417% |
| تیم | 120 | 0.397% |
| Totals | 30.2k | |

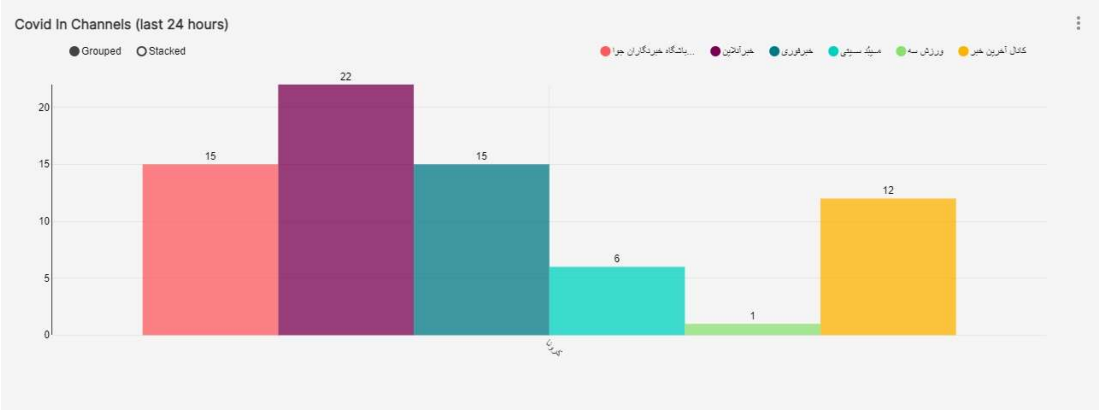
1 2 3 4 5 6 7 ... 680

داشبورد سوم با نام Speific Channels/Keywords اطلاعات مربوط به یک کانال/کلمه‌کلیدی خاص را به ما نشان می‌دهد. این داشبورد نیز مانند موارد قبلی شامل دو تب است. در تب اول اطلاعات مربوط به کانال "ورزش سه" نمایش داده شده است. در نمودارهای این تب آمار دریافت اطلاعات این کانال در روز و هفته گذشته نشان داده شده است.


در تب دوم نیز اطلاعات مربوط به کلمه کلیدی کرونا آمده است. نمودارهای موجود در این تب تعداد تکرار این کلمه را در کانال‌های مختلف در طول هفته گذشته و مجموع تکرار آن را به ما نشان می‌دهند.




Covid In Channels (last 24 hours)
Grouped
Stacked
...باشگاه خبرنگاران جوا
خبرآنلاین
خبرفوری
مسیج سیتی
ورزش سه
کانال آخرین خبر
22
15
15
6
1
12
20
15
10
5
0

## اطلاعات مربوط پروژه

آدرس گیت‌هاب و نسخه‌ی لایو پروژه به شرح زیر می‌باشد:

آدرس گیت‌هاب اصلی مربوط به پروژه:

https://github.com/aamirpashaa/bd-project-1400

آدرس گیت‌هاب مربوط به frontend پروژه:

https://github.com/aamirpashaa/bd-project-1400-frontend

مخازن به صورت خصوصی می‌باشند که به اکانت smbanaie@ دسترسی لازم داده شد.

آدرس فرانت لایو پروژه:

http://37.152.176.189:4200

username: bdproject

password: bdproject

آدرس کیبانا لایو پروژه:

http://37.152.176.189:5601

username: bdproject

password: bdproject

آدرس سوپرست لایو پروژه

http://37.152.176.189:8088

username: admin
password: admin

دانشکده ی مهندسی برق و کامپیوتر دانشگاه تهران

درس :

کلان داده  و تحلیل داده های حجیم

دکتر اسدپور

پروژه پایانی:

طراحی یک سامانه بلادرنگ برای تحلیل لحظه‌ای داده‌های توئیت‌های فارسی در توییتر

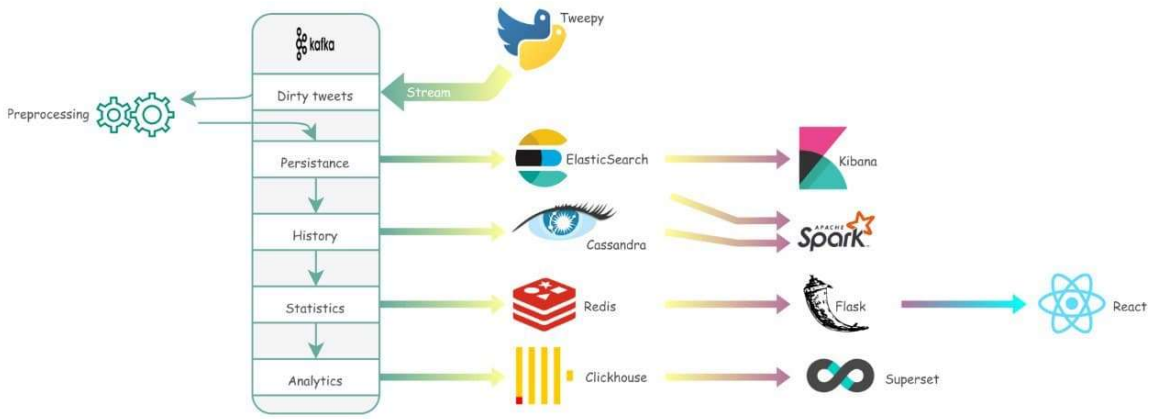
گروه :

یارا محمدی، محمدحسین حاج کاظم نیلی، آرمین آیت اللهی، زهرا حبیب زاده

*آدرس گیت‌هاب پروژه:*

*https://github.com/arminayat/bigdata-finalproject*

شمای کلی از پروژه را در تصویر زیر می بینیم که گام به گام مراحل مانند تصویر زیر توسط تیم انجام می شود.




## گام اول: دریافت اطلاعات و Preprocess

آدرس در گیتهاب:

https://github.com/arminayat/bigdata-finalproject/tree/kafka

https://github.com/smbanaie/bigdata-finalproject

### تنظیمات داکر کافکا: (آدرس فایل: ./docker-compose.yml)

تنظیمات کافکا به صورت حداقلی برای اجرا انتخاب شده. به این صورت که تمام داده ها روی یک Broker و با Replication Factor برابر ۱ ذخیره میشوند. Log های کافکا به مدت یک هفته در پوشه mount شده باقی میمانند و از پورتهای پیشفرض کافکا و zookeeper استفاده شده.

```yaml
zookeeper:
  image: confluentinc/cp-zookeeper:latest
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
  ports:
    - 22181:2181
  volumes:
    - ./docker_data/zk-data:/var/lib/zookeeper/data
    - ./docker_data/zk-txn-logs:/var/lib/zookeeper/log

kafka:
  image: confluentinc/cp-kafka:latest
  depends_on:
    - zookeeper
  ports:
    - 29092:29092
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT_HOST://localhost:29092
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    KAFKA_LOG_RETENTION_MS: 604800000
    KAFKA_LOG_RETENTION_CHECK_INTERVAL_MS: 302400000
  volumes:
    - ./docker_data/kafka-data:/var/lib/kafka/data
```

**دریافت داده از توییتر : (آدرس فایل: ./kafka stream/twitter_chnl.py)**

برای دریافت داده از توییتر از قابلیت streaming کتابخانه tweepy استفاده شده است. برای دریافت داده های فارسی به صورت stream طبق تصویر زیر، زبان فارسی در فیلد languages قرار داده شده است. فیلد track یک لیست از کلمات میگیرد و فقط توییت هایی را برمیگرداند که دارای یکی از آن کلمات باشند. از آنجایی که در تابع filter توییتر، اگر فیلد زبان مقدار داشته باشد فیلد track نمیتواند خالی باشد (یکی از عیب های این API) یک لیست ۴۰۰ تایی از stop_words های فارسی در این فیلد گذاشته ام. خوبی انتخاب stop_words ها در این فیلد آن است که اکثر توییت های فارسی این شرط را ارضا میکنند و دریافت میشوند.

```
stream = Stream(auth, l)
# Can't filter only on language
# Can't retrieve more than 1% of tweets anyway, so ok to only crawl tweets containing our stopwords
stream.filter(track = stop_words, languages=["fa"])
```

دریافت توییت ها توسط توابع event based به اسم on_data و on_error (در صورت خطا) درون کلاس stream_listener توییتر انجام میشود که در کد این بخش قابل مشاهده است.

توییت ها هر ۵ ثانیه ۱ بار گرفته میشوند و مستقیما در تاپیک "dirty_tweets" کافکا به صورت رشته produce میشوند.

کد این بخش در حین اجرا شدن، فیلد text توییت های گرفته شده و تعداد توییت های دریافت شده را

```
3
حامد اسماعیلیون به پدر #مصطفی_نعیماوی
دلم میخواست به او میگفتم آنکه هواپیمای بین‌المللی بود و همه چیزش ثبت شده بود… https://t.co/
n7FiBpsiXG
4
RT @marmarexol: رفته بودم رست ولی خبرا رو شنیدم و واقعا باورم نمیشه حتی از فندوم خودمون و اوتی ۱۲
استنا این طوری دارن رفتار میکنن.چطور چیزی…
5
RT @HazeliReza: از #اعتراضات مردم #اصفهان حمایت کنید، در #خوزستان تنها از اعتراضات #بختیاری ها
حمایت کنید. https://t.co/58xVeOPB57
6
@sajadcooper تا منزل علی لاریجانی رو گفتم 😂
```

در کنسول طبق زیر چاپ میکند:

## پیش پردازش داده : (آدرس فایل: ./kafka stream/preprocess_chnl.py)

در این بخش، یک Consumer، اطلاعات تولید شده در تاپیک "dirty_tweets" را میخواند و بعد از انجام پیشپردازش های گفته شده، یک Producer، این اطلاعات را در تاپیک "persistence" میریزد.

```
# Produce clean tweets
producer = KafkaProducer(bootstrap_servers=['localhost:29092'],
                         value_serializer=lambda x:
                         dumps(x).encode('utf-8'),
                         api_version=(0,10))

# Consume dirty tweets
consumer = KafkaConsumer(
    "dirty_tweets",
     bootstrap_servers=['localhost:29092'],
     auto_offset_reset= 'earliest', # 'earliest', # Start from last consumed, #'latest' start from last produce
     enable_auto_commit=True,
     auto_commit_interval_ms = 1000, #ms # Ok. cuz our messages come every 5 seconds
     group_id='twitter',
     value_deserializer=lambda x: loads(x.decode('utf-8')),
     api_version=(0,10))
```

تنظیمات این دو در تصویر زیر قابل مشاهده است:

در Consumer فیلد auto_offset_reset برابر 'earliest' قرار داده شده تا همیشه از آخرین داده consume شده دوباره شروع به consume کردن شود. همچنین consumer هر ثانیه تاپیک را چک میکند برای اطلاعات جدید (auto_commit_interval_ms) و از آنجایی که هر ۵ ثانیه یک بار اطلاعات توییتر گرفته میشوند، اطلاعاتی از دست نمیرود.

### پیش پردازش متن برای استخراج کلیدواژه:

```
for message in consumer:
    #print(message.value)
    #print(message.key)

    json_ = message.value
    # If tweet is longer than 140 characters, it isnt stored fully in Text field
    text = json_["text"] if not json_["truncated"] else json_["extended_tweet"]["full_text"]

    # preprocessing for keyword extraction
    text2 = preprocess_text(text)
```

در صورتی که یک توییت بیشتر از ۱۴۰ کاراکتر داشته باشد، فیلد truncated آن مقدار ۱ میگیرد و text کامل آن به جای فیلد text در فیلد extended_tweet.full_text ذخیره میشود. طبق تصویر زیر ابتدا متن کامل توییت گرفته شده سپس پیش پردازش های متن انجام میشود:

در تابع preprocess_text، ابتدا تمام underline ها با فاصله جایگذاری میشوند تا بتوان هشتگها را نیز مانند کلمات عادی پردازش کرد، سپس علایم نگارشی برداشته شده و با توابع کتابخانه Hazm عملیات های Normalize و Stem و Lemmatize انجام شده تا نیم فاصله ها ایجاد شوند، کلمات جمع تبدیل به مفرد شوند و افعال تبدیل به مصدرهایشان شوند. سپس از تابع WordTokenizer همین کتابخانه استفاده میشود تا تمام کلمات جدا شوند. این تابع فیلدهای جالبی دارد که از آنها نیز استفاده شده، به طور مثال replace_links, replace_IDs, replace_numbers, separate_emoji, replace_emails. در نهایت در لیست کلمات ایجاد شده، تمام stop_words ها و تمام افعال (لیست کلمات این دو را از گیتهاب کتابخانه Khayam دانلود کرده و درون فایل (آدرس فایل ها: /kafka stream/data/.) ذخیره کرده ام) و تمام کلماتی که حروف فارسی ندارند را حذف کرده و دوباره این کلمات را به هم میچسبانیم. خروجی این تابع، متن پیشپردازش شده است که برای بدست آوردن کلیدواژه ها استفاده میشود.

### استخراج هشتگ ها و URL ها:

در مرحله بعدی طبق کد زیر هشتگها و URL ها از متن اولیه استخراج میشود و سپس کلیدواژه ها

```
# Data to add
json_['text_k'] = text
json_["hashtags_k"], json_["urls_k"] = extract_hashtags_urls(text) # Find tags and urls
json_["keywords_k"] = keyword_extraction(text2, produce_idx)
```

ساخته میشوند.

برای بدست آوردن هشتگ و URL از regex های زیر استفاده کرده ام:

```
# Extract Hashtags
tags = re.findall("#(\w+)", text)

# Extract URLS          # Optional http(s) and www                          no persian letters     and words including .          and words including / and .
url_groups = re.findall('(http:\/\/www\.|https:\/\/www\.|http:\/\/|https:\/\/)?(?![\u0600-\u06FF\s])+([\w_-]+(?:(?:\.[\w_-]+)+))([\w.,@?^=%&:/~+#-]*[\w@?^=%&/~+#-])?' , text)
```

### استخراج کلیدواژه ها با TF-IDF:

برای بدست آوردن کلیدواژه ها ابتدا تمام کلمات متن که درون لیست keywords زیر هستند را به لیست کلیدواژه ها اضافه کرده ایم و در صورت وجود هرکدام از covid_keywords های زیر،

```
keywords = ['دانشگاه' , 'تورم' , 'کرونا' , 'طلا' , 'دلار' , 'انتخابات' , 'روحانی' , 'دولت' , 'تحریم' , 'اقتصاد' , 'بورس']
covid_keywords = ['کووید19' , 'کووید' , 'کووید19' , 'کووید19']
```

کلیدواژه "کووید۱۹" را نیز به لیست اضافه کرده ایم.

استخراج مابقی کلیدواژه ها نیاز به محاسبه TF هر توییت و IDF یک corpus دارد. از آنجایی که داده ها در لحظه تولید میشوند corpus ثابتی برای داده های توییتر نداریم و از ۲ راهکار میتوان استفاده کرد:

۱) استفاده از یک corpus بزرگ offline، مثلا یک dataset توییتر فارسی. اشکال این روش آن است که متون و موضوعات توییتر هر روز و هر هفته تغییر میکند و یک corpus ثابت از یک dataset دیگر شاید نتواند IDF های درست تولید کند.

۲) استفاده از یک corpus بزرگ online، مثلا ذخیره متن تمام توییت های دریافت شده و استفاده از آنها به عنوان corpus. ما از این روش استفاده کردیم.

درواقع هر بار که کد بخش پیشپردازش صدا زده میشود، از یک فایل corpus (آدرس فایل: kafka/stream/data/corpus.txt) تمام توییت هایی که از قبل ذخیره شده اند را میخوانیم و با هر بار consume کردن تاپیک "dirty_tweets" توییت جدید در این corpus ذخیره میشود. زمانی که تعداد توییت های این corpus به ۵۰۰۰۰ برسد، با اضافه شدن هر توییت جدید، قدیمی ترین توییت از این corpus پاک میشود.

توجه شود که به دلیل بزرگ نبودن corpus در اجراهای من، کلیدواژه های انتخاب شده بعضا بی معنی و به نظر اشتباه می آیند، ولی نکته این الگوریتم در جواب دادن بهتر آن در صورت اجرای متوالی و طولانی است.

در نهایت تمام فیلد های جدید، همراه با اطلاعات دیگر توییت توسط یک Producer به تاپیک "persistence" فرستاده میشوند و اطلاعات استخراج شده هر توییت در این گام در کنسول مانند تصویر زیر چاپ میشود:

```
text:     RT @amirebtehaj: حالا دیگه معلوم نیست کدوم گلوله از
عراقی‌ها بوده و کدوم از حکومت.
                                        https://t.co/SxLayTcacT #خوزستان
tags:     ['خوزستان']
urls:     ['https://t.co/SxLayTcacT']
keys:     ['کدوم']
************************
text:     @Shhrzaaad 😂😂😂
                                                    ناشناسم چیز جالبیه
tags:     []
urls:     []
keys:     ['ناشناسم', 'جالبیه']
************************
text:     شهر من دورود استان لرستان یک شهر صنعتی و رو به رشد بود
که بخاطر زباله بودن حکومت چ ا هر روز بدتر از روز قبل شد و امروز با
وجود پذیرش هزاران مهاجر از استانهای جنوبی و عشایر کشورمون و نبود
امکانات و زیرساخت ها و مشاغل کافی به یکی از شهرهای بسیار آسیب دیده
تبدیل شده.
                                                                #دورود
                                                              #خوزستان
tags:     ['خوزستان', 'دورود']
urls:     []
keys:     []
************************
```

### کانال های دیگر:

برای کانال های Persistance و hashtag و statistics ۳ فایل persistence_chnl.py و hashtag_history_chnl.py وredis_chnl.py در نظر گرفته شده است (آدرس فایلها: kafka/stream/) که هر کدام به ترتیب، همان داده تولید شده در مرحله پیش پردازش را ابتدا از تاپیک قبلی (به اسم های "persistence" و "hashtag" و "statistics") گرفته و در انتها به تاپیک بعدی پاس میدهد.

---

پایان بخش اول

## گام دوم: persistence

### مقدمه

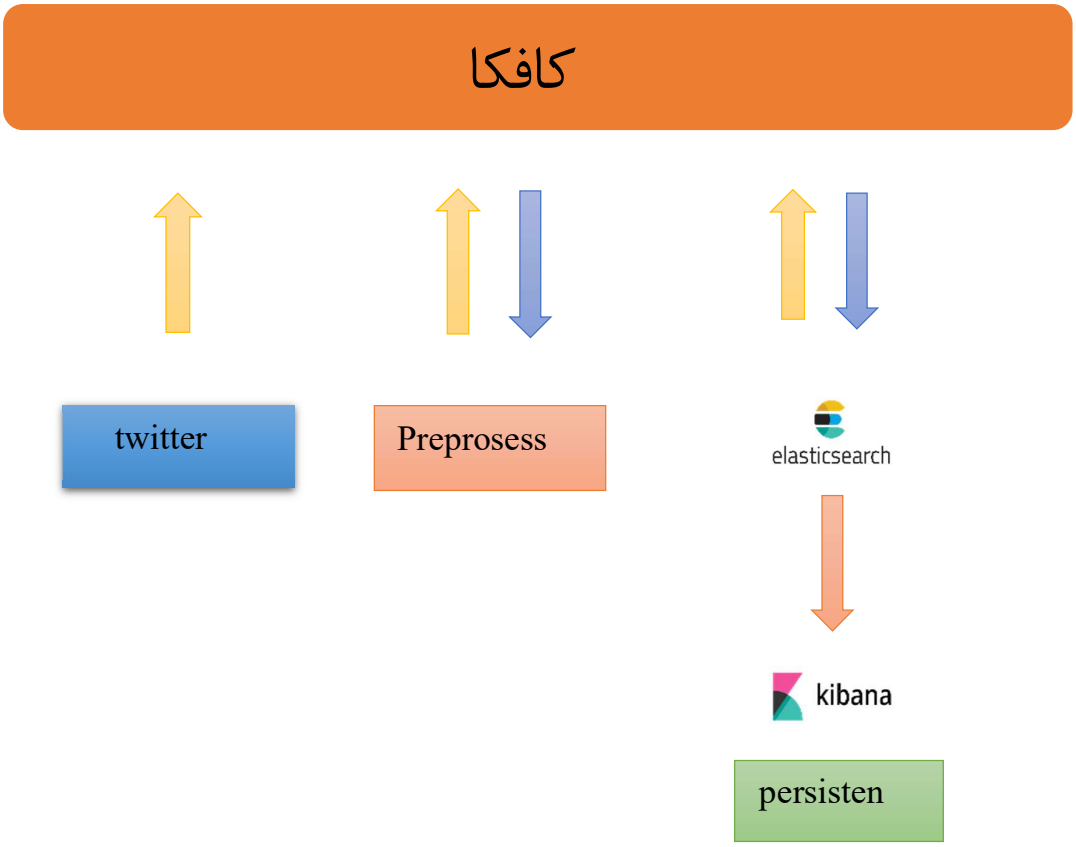
#### توضیحات کلی elastic search :

یک دیتابیس از خانواده ی NoSQL است که یکی از دیتابیس های معروف در حوزه ی متن است. توزیع بودن نرم افزار Elasticsearch سرچ باعث می شود تا حجم بسیار بالایی از داده ها را به صورت موازی پردازش کند و سریعاً بهترین جواب را در هنگام جستجو برای کاربر پیدا می کند. **Kibana** که ابزاری کاملاً مشهور در زمینه تصویری سازی و گزارش دهی است و به صورت یکپارچه با الستیک سرچ (Elasticsearch)استفاده می گردد.

#### توضیحات کلی kafka :

کافکا یک پلتفرم پیام رسان توزیع شده و اشتراکی است که برای هندل کردن داده های جریان در زمان واقعی به صورت پایپ لاین کاربرد دارد که از دو بخش producer و consumer تشکیل شده است. در پروژه ی ما کافکا با داکر بالا امده است و به این صورت عمل می کند که ابتدا داده ها از توییتر گرفته می شود که خب دیتاهای تمییز نیستند و نیاز به پیش پردازش، طبق خواسته ی مسئله در گام ۱ دارند. پس این داده های تمییز به عنوان producer به کافکا منتقل می شوند (کد را در مرحله ی بعد توضیح می دهم).حال باید پیش پردازش را انجام دهیم که پوشه ی پیش پردازش این اطلاعات را از کافکا به صورت realtime می خواند(consumer). اما من در این گام باید از این داده های انلاین استفاده کنیم پس بازهم کافکا نقش یه واسط را این وسط بین گام ۲ و گام ۱ بازی می کند به این صورت که دیتاهای پیش پردازش شده و تمییز به کافکا تحت producer منتقل می شود و این دیتاهارا تحت consumer دریافت می کنیم و کار خودم را در این گام شروع خواهم کرد و در پایان به گام ۳ به کافکا تحت producer منتقل می کنم. اینکار مزایای زیادی دارد . مثلا فرض کنید یک تیم بخواهند همزمان بر روی یک دیتای بزرگ مثل توییتر کارکنند،استفاده از کافکا ضمن اینکه سرعت کار را بالا می برد و این اجازه را به تیم می دهد که هرکدام از نفرات، از Nosql های مختلف یا ابزارهای دیگر برای کشف دیتای خود استفاده کنند، باعث می شود که به عنوان انلاین بتوان به تحلیل اطلاعات وسیعی تحت عنوان big data پرداخت.

در ادامه با اشناشدن کاربرد کافکا در این بخش، در کد بیشتر توضیح خواهم داد.(یک شمای کلی به زبان ساده از عملکرد کافکا از گام ۱ تا مرحله ۲ در این پروژه رسم خواهم کرد). برای مثال،فلشی که از persistence(elastic) وارد کافکا می شود، نشان می دهد، elastic نقش(producer) برای گام ۳ را دارد و فلشی که از کافکا به persistence (elastic)وارد می شود، نشان می دهد، elastic (consumer) است.




### بخش اول: تنظیمات داکر برای کافکا و الاستیک و کیبانا

در این گام ابتدا کافکا، کیبانا و الاستیک را در داکر بالا می اوریم که دیگر از نرم افزار کیبانا جداگانه استفاده نکنیم.(البته می توان تنها کافکا را در داکر بالا اورد و الاستیک و کیبانا را در نرم افزار خاص خودش بالا اورد، فرقی نمی کند)که در یک فایل yml تنظیمات این۳ کانتیر را ست می کنیم .سپس compose-up می کنیم تا ست شود(کل تیم، کافکا، الاستیک، کیبانا و...را در این فایل ست کردیم تا یک فایل برای الاوردن این کانتینرها در داکر ست شود) :

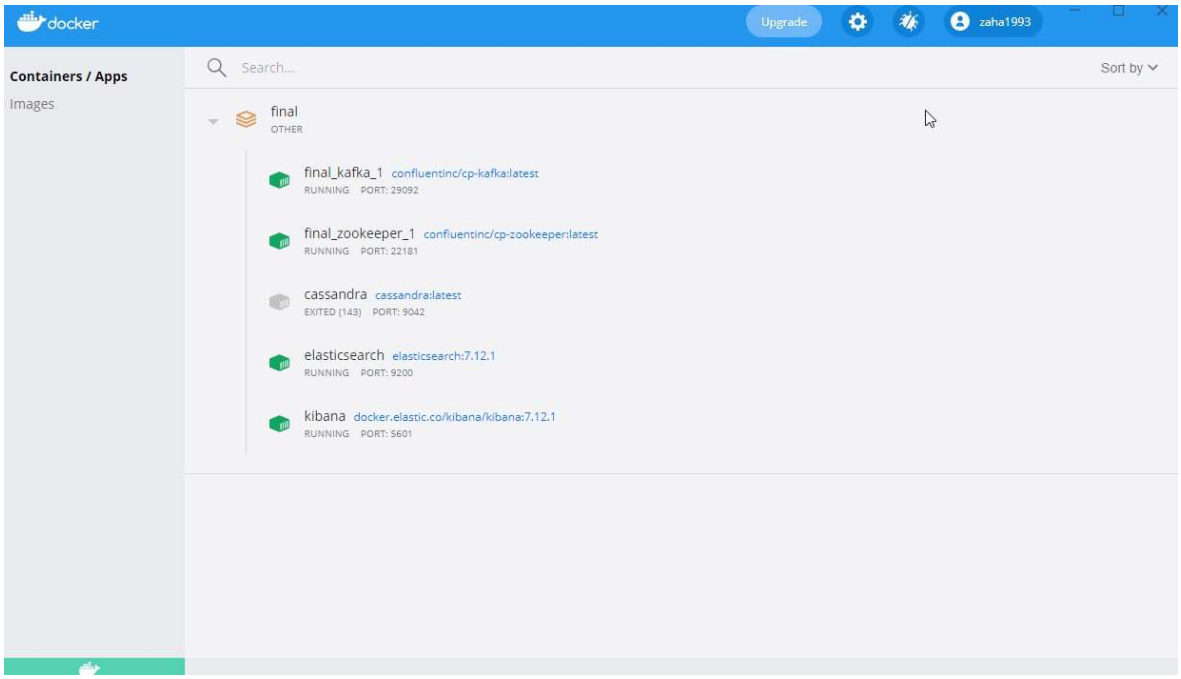
من از نسخه ی ۷.۱۲.۱ الاستیک و کیبانا برای image ها استفاده کردم.

```yaml
elasticsearch:
  image: elasticsearch:7.12.1
  container_name: elasticsearch
  environment:
    - xpack.security.enabled=false
    - discovery.type=single-node
  ulimits:
    memlock:
      soft: -1
      hard: -1
    nofile:
      soft: 65536
      hard: 65536
  cap_add:
    - IPC_LOCK
  volumes:
    -  ./docker_data/elasticsearch-data:/usr/share/elasticsearch/data
  ports:
    - 9200:9200
```

```yaml
kibana:
  image: docker.elastic.co/kibana/kibana:7.12.1
  container_name: kibana
  ports:
    - 5601:5601
  environment:
    ELASTICSEARCH_HOSTS: http://elasticsearch:9200

zookeeper:
  image: confluentinc/cp-zookeeper:latest
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
  ports:
    - 22181:2181
  volumes:
    - ./docker_data/zk-data:/var/lib/zookeeper/data
    - ./docker_data/zk-txn-logs:/var/lib/zookeeper/log
```

```yaml
zookeeper:
  image: confluentinc/cp-zookeeper:latest
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
  ports:
    - 22181:2181
  volumes:
    - ./docker_data/zk-data:/var/lib/zookeeper/data
    - ./docker_data/zk-txn-logs:/var/lib/zookeeper/log

kafka:
  image: confluentinc/cp-kafka:latest
  depends_on:
    - zookeeper
  ports:
    - 29092:29092
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT_HOST://localhost:29092
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    KAFKA_LOG_RETENTION_MS: 604800000
    KAFKA_LOG_RETENTION_CHECK_INTERVAL_MS: 302400000
  volumes:
    - ./docker_data/kafka-data:/var/lib/kafka/data
```

در ادامه داکر را با زدن docker-compose up در cmd در پوشه ای که فایل yml(تنظیمات این کانتینرها) وجود دارد، بالا میاوریم .



## بخش دوم: کدنویسی در پایتون( ست کردن consumer و producer و الاستیک)

کافکا دو تابع به اسم KafkaProducer و KafkaConsumer دارد که در این گام ابتدا تنظیمات producer و consumer را در این توابع ست می کنیم. ما در این بخش ابتدا نقش consumer را داریم که در تابع KafkaConsumer باید از دیتای تمییز شده در گام قبل که با اسم clean_tweets به این بخش توسط کافکا منتقل شد، استفاده کنیم.

```
# -*- coding: utf-8 -*-
"""
Created on Thu July 1 18:00:00 2021

@author: zaha
"""

from kafka import KafkaConsumer, KafkaProducer
from json import loads, dumps


######################### Define Kafka clients ########################


# Produce persistent data
producer = KafkaProducer(bootstrap_servers=['localhost:29092'],
                         value_serializer=lambda x:
                         dumps(x).encode('utf-8'),api_version=(0,10))


consumer = KafkaConsumer(
    'clean_tweets',
    bootstrap_servers=['localhost:29092'],
    auto_offset_reset= 'earliest', # 'earliest', # Start from last consumed, #'latest' start f
    enable_auto_commit=True,
    auto_commit_interval_ms = 1000,
    group_id='twitter',
    value_deserializer=lambda x: loads(x.decode('utf-8')),
    api_version=(0,10))


############### Insert data to elastic search here ####################
```

در ادامه کتابخانه ی الاستیک را ایمپورت می کنیم تا از تابع های این کتابخانه در پایتون از جمله Elasticsearch برای ست کردن پورت الاستیک استفاده کنیم.(من مراحل اصلی خود را در کیبانا انجام می دهم اما قبل آن می خواهم، بنابر سوالات مطرح شده، کمی پیش پردازش انجام دهم چون دیتاهایی که در consumer است، فیلدهای خیلی زیادی دارد که بنظرم می توان انها را قبل ارسال این توییت ها به الاستیک فیلتر کرد).

```
############### Insert data to elastic search here ####################

from elasticsearch import Elasticsearch

# Elastic search configuation

es = Elasticsearch(HOST="http://localhost", PORT=9200)
es = Elasticsearch()
```

### بخش سوم: Persian Analyzer در کیبانا

در این قسمت قبل از انجام پیش پردازش کلی و ارسال توییت ها به الاستیک بنابر خواسته ی مسئله باید از Persian analyzer استفاده کنیم که برای اینکار من از این راه استفاده کردم. ابتدا در کنسول کیبانا تنظیمات مربوط به انالیزها را انجام می دهیم به اینصورت که یک اندیکس به اسم توییتر(که با دستور put خودش ایجاد می شود)، ایجاد می کنیم سپس در این ایندکس، استاپ وردها، tokenizer (استاندارد انتخای می کنیم تا بیشتر punctuation ها حذف شوند)و zero_width_spaces را ست و فیلتر می کنیم.(بعد از یکبار ران کردن خاموش می کنیم چون به صورت دیفالت این تنظیمات رو ایندکس های توییت که از پایتون می ایند، ست می شود)

```
#################################persian_analize(stop_word,tokenizer)############################
put /twitttt
{
 "settings": {
   "analysis": {
     "char_filter": {
       "zero_width_spaces": {
         "type":       "mapping",
         "mappings": [ "\\u200C=>\\u0020"]
       }
     },
     "filter": {
       "persian_stop": {
         "type":       "stop",
         "stopwords":  "_persian_"
       }
     },
     "analyzer": {
       "rebuilt_persian": {
        "tokenizer":     "standard",
        "char_filter": [ "zero_width_spaces" ],
        "filter": [
          "lowercase",
          "decimal_digit",
          "arabic_normalization",
          "persian_normalization",
          "persian_stop"
        ]
       }
     }
   }
 }
}
```

### بخش چهارم: پیش پردازش توییت ها قبل از ارسال به الاستیک

ابتدا با استفاده از حلقه ی for توییت هایی که به صورت انلاین از گام پیش پردازش می ایند را دریافت می کنیم و در twit قرار می دهیم. این توییت ها فیلدهای زیادی دارند که بهتر است فقط فیلدهایی گرفته شود که در این مرحله به ان ها نیاز داریم، پس از یک حلقه ی for دیگر استفاده می کنیم و فیلدهای (created_at(زمان به عدد و تاریخ به حروف)، keywords_k (کلمات کلیدی مثل کووید و...که در پیش پردازش گام قبل گرفتیم و در این فیلد قرار دارد)، hashtags_k (هشتگ هایی که خودمان در گام قبل جدا کردیم و در این فیلد قرار دادیم)،text_k (متن توییت ها که در گام قبل جدا کردیم و در این فیلد قرار دادیم)) دریافت می کنیم. اما من می خواهم یک سری فیلد دیگر مثل روز و ساعت هم به این فیلدها قبل از ارسال به الاستیک اضافه کنم که برای اینکار پردازشی روی فیلد created_at انجام می دهم و روز و ساعت را به ۴ فیلد بالا اضافه می کنم. حال از تابع index در الاستیک در پایتون استفاده می کنم و به این توییت ها همان index را می دهم که در Persian analyzer ست کرده بودم.این حلقه ی for کلا درحال اجراست و توییت ها به صورت انلاین از توییتر توسط کافکا وارد پیش پردازش و از پیش پردازش توسط کافکا به این گام می ایند و تا زمانی که ما قطع نکنیم، دیتا درحال گرفتن است و با این تنظیمات شده در مرحله ی پیش پردازش این گام، دیتا به صورت انلاین وارد الاستیک می شود.(من توییت های خود را در بازه های زمانی متفاوت گرفتم(البته من

خیلی قبل تر از این زمانی که در گزارش می نوشتم هم  در روز های مختلف دیتا گرفته بودم و در کیبانا داشبورد ساختم اما ۲ بار داکرم پرید یعنی بالا نمیامد، که مجبور شدم ۲ بار داکر را ریست کنم که داشبورت ها پرید و اگر زیاد دیتا بگیریم، گاهی خود توییتر، بن می کند! که من یکی دوبار بن شدم))) تا ریزالت های بهتری را بتوانم در داشبورد نشان دهم.

```
for message in consumer:

    twit = message.value
    twittt = {k: twit[k] for k in ("created_at",'keywords_k','hashtags_k','text_k')}
    hour = twittt["created_at"][11:13]
    day = twittt["created_at"][8:10]
    twittt['hour'] = hour
    twittt['day_k'] = day
    index=es.index(index="twitttt", body=twittt)
    print(twittt)

    print("*************************")
```

۱.توییتر

```
*******************
33
RT @vatanamiran0002: #اهواز
                                                    مادران و دختران شجاع خوزستانی
                                    آنها میخروشند و در اعتراض به بی آبی مردم را به قیام دعوت میکنند

                                                                                #قیام_سراسری
#…
```

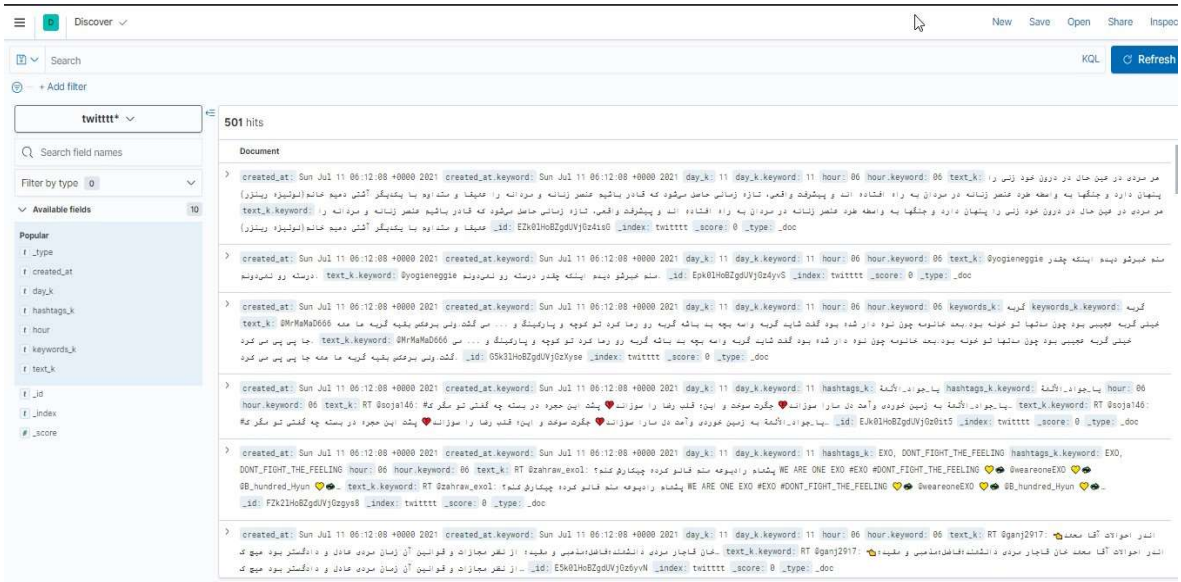
۲.preprocess

```
*************************
text:    RT @vatanamiran0002: #اهواز
                                                    مادران و دختران شجاع خوزستانی
                                    آنها میخروشند و در اعتراض به بی آبی مردم را به قیام دعوت میکنند

                                                                                #قیام_سراسری
#…
tags:    ['اهواز', 'قیام_سراسری']
urls:    []
keys:    ['دم']
*************************
```

۳.persistence (الاستیک)

```
*************************
{'created_at': 'Tue Jul 20 08:25:26 +0000 2021', 'keywords_k': ['دم'], 'hashtags_k': ['اهواز',
'قیام_سراسری'], 'text_k': 'RT @vatanamiran0002: #اهواز مادران و دختران شجاع خوزستانی\n آنها \
میخروشند و در اعتراض به بی آبی مردم را به قیام دعوت میکنند\n\n#قیام_سراسری\n#…', 'hour': '08',
'day_k': '20'}
*************************
```

۴.وارد شدن به الاستیک و دیدن این دیتاها در کیبانا در پورت ۵۶۰۱




ادامه ی مراحل من برای طراحی داشبورد و کوئری های خواسته شده در کیبانا خواهد بود اما قبل از باید با استفاده از تابع send ، گام خودم را به عنوان producer به کافکا منتقل کنم تا هم تیمیم در کاساندرا به عنوان consumer دریافت کند اما در قسمت الاستیک ذکر شده است که بدون پیش پردازش وارد مرحله ی بعد شود پس من همان توییت اصلی که فقط در گام ۱ پیش پردازش شده را به کافکا با نام History ارسال خواهم کرد.

```
producer.send('History', value=twit)
```

## بخش پنجم: طراحی داشبورد

در این بخش می خواهیم داشبوردی را طبق سوالات خواسته شده طراحی کنیم، برای اینکار مرحله به مرحله پیش می رویم و داشبورد را مرحله به مرحله طراحی می کنیم تا درنهایت به یک داشبورد کلی

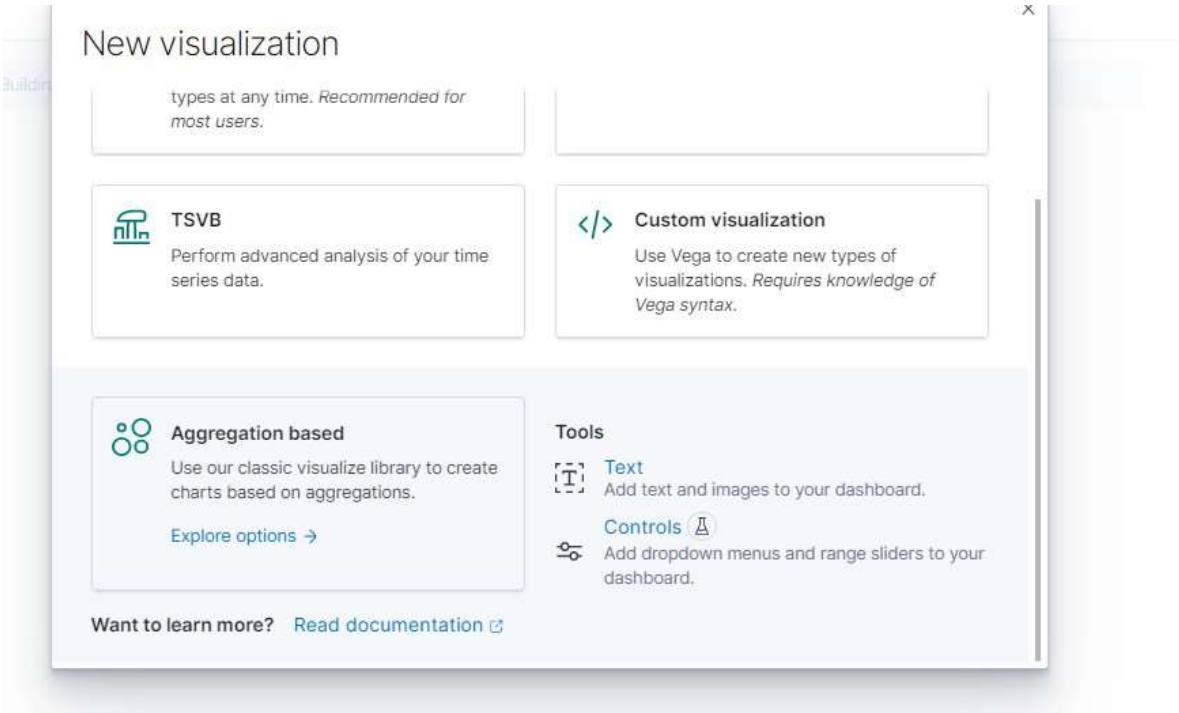
برسیم. این داشبورد به صورت انلاین زمانی که دیتا می گیریم تغییر می کند چون هر ۱ دقیقه رفرش می شود(توضیح دقیقتر نمودار ها در روز ارائه انلاین داده می شود) :

- ابرکلمات کانال (در این قسمت من بجای کانال از هشتگ استفاده می کنم چون دیتای ما از توییتر گرفته شده است و کانال نداریم) .

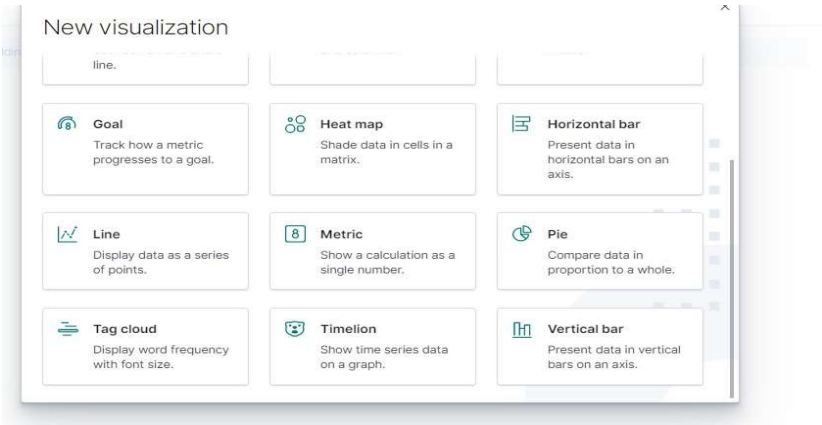
در کیبانا مراحل زیر را انجام می دهیم:

۱.ابتدا از قسمت indexmangement قسمت index pattern ، ایندکس توییت را ایجاد می کنیم. (چون چندتا ایندکس در کیبانا از قبل سیو شده است، ما با ایندکسی که برای این بخش ایجاد شده است، کار داریم)

۲.سپس از قسمت Analytics منو ، داشبورد را انتخاب می کنیم و سپس بر روی ایجاد داشبورد کلیک می کنیم، سپس از قسمت Visualize Library را انتخاب می کنیم و یک ویژولیزیشن جدید ایجاد می کنیم و گزینه ی aggregation based را انتخاب می کنیم:

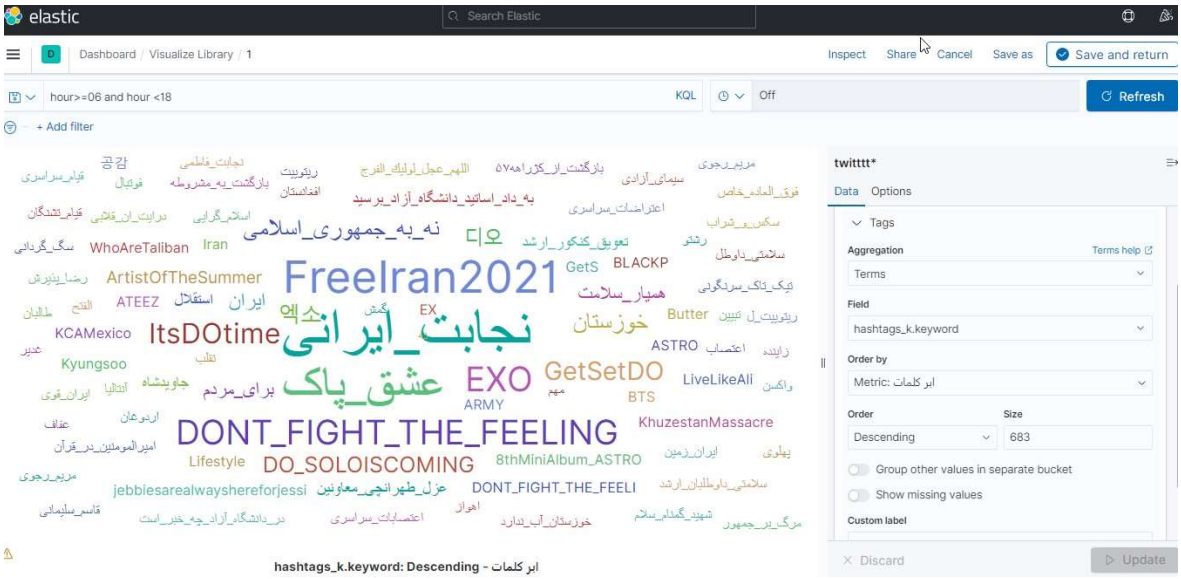



۳.سپس از قسمت aggregation based، tag cloud را انتخاب می کنیم:



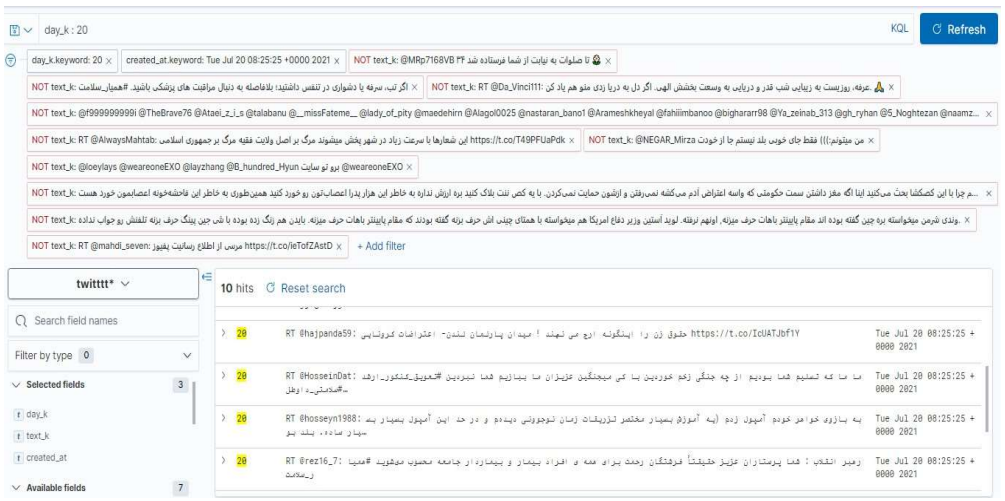

۴.سپس ایندکس توییت را انتخاب می کنیم

۵.سپس از قسمت دیتا در سمت راست تصویر پایین، تنظیمات را ست می کنیم.مثلا برای فیلد را بر اساس هشتگ توییت ها قرار می دهیم و بازه ی زمانی را از ساعت ۶ صبح تا ۶ غروب قرار می دهیم،سپس از هشتگ ها برای رسم نمودار استفاده می کنیم.



و در نهایت این ابرکلمات را به داشبوردمان به عنوان اولین ریزالت، اضافه می کنیم و گام به گام پیش می رویم تا داشبورد را کامل کنیم.همانطور که می بینیم، ابرکلمات بسته به تکرارشان، با رنگ مختلف و سایز مختلف نمایش داده می شوند که نجابت ایرانی در صدر قرار دارد.ابرکلمات فقط یک شمای کلی می دهد.


- در بخش دوم از ما خواسته شده است که ۱۰ متن اخیر را نمایش دهیم.برای اینکار ابتدا از قسمت Discover در منو، ۱۰ متن اخیرطبق زمان اخیر، دریافت می کنیم، سپس انرا سیو می کنیم تا در داشبورد از ان استفاده کنیم(در اخرین زمان ۲۱ متن قرار داشت که من طبق خواسته ی مسئله، ۱۰ تای اخیر را طبق زمان در discover فیلتر کردم انرا نمایش می دهم):



در داشبورد گزینه add from library را می زنیم و این ۱۰ متن ذخیره شده را به داشبورد اضافه می کنیم:

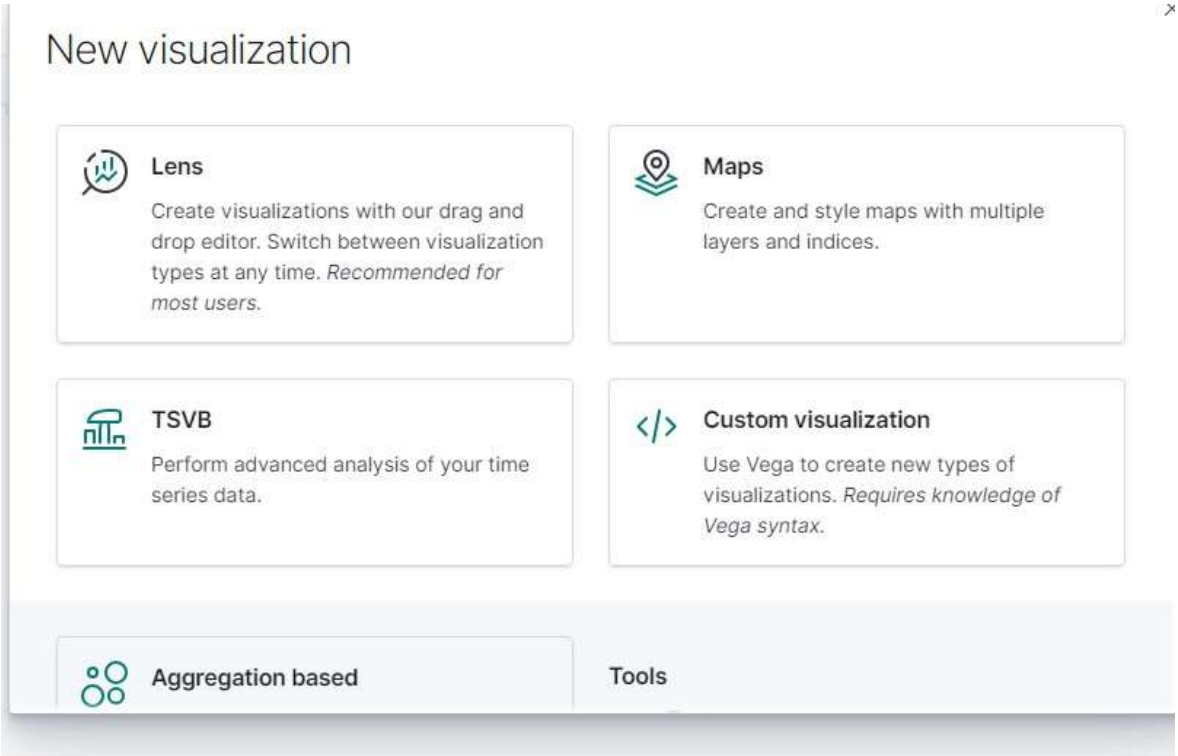
ده پست اخیر

1–10 of 10

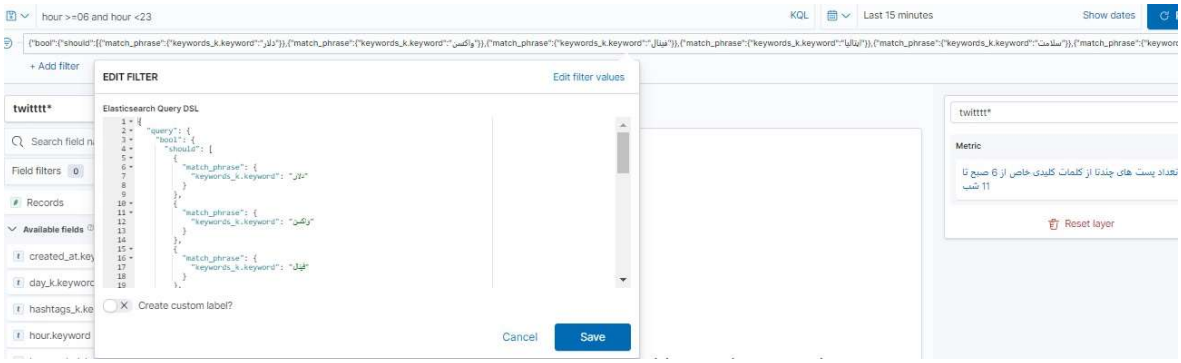
| | day_k | text_k | created_at |
|---|---|---|---|
| > | 20 | RT @par_shaa: همیشه برام سواله تو ذهنشون چی میگذره؟ 😂 !پارگی از خنده https://t.co/9Nr2KDTGqC | Tue Jul 20 08:25:25 +0000 2021 |
| > | 20 | اینکه تو تایملاین عکس یا فیلمی از حاج #قاسم_سلیمانی می‌بینم و بی‌اختیار اشکم سرازیر می‌شه غم شیرینی واسم، اما امان از دل خانواده‌ش... خدا بهشون صبر بده https://t.co/e0h4ooYn5B | Tue Jul 20 08:25:25 +0000 2021 |
| > | 20 | RT @NaeimiSana: برای کسی که اعتصاب غذا میکنه برای اینکه مطالباتِ مدِ نظرش محقق بشه خیریه نمیزن غذا جمع کنن!! سقفِ مطالباتِ #خوزستان رو تا... | Tue Jul 20 08:25:25 +0000 2021 |
| > | 20 | RT @hajpanda59: حقوق زن را اینگونه ارج می نهند ! میدان پارلمان لندن- اعتراضات کرونایی https://t.co/lcUATJbf1Y | Tue Jul 20 08:25:25 +0000 2021 |
| > | 20 | RT @HosseinDat: ما ما که تسلیم شما بودیم از چه جنگی زخم خوردین با کی میجنگین عزیزان ما ببازیم شما نبردین #تعویق_کنکور_ارشد #سلامتی_داوطل... | Tue Jul 20 08:25:25 +0000 2021 |
| > | 20 | RT @hosseyn1988: به بازوی خواهر خودم آمپول زدم (یه آموزش بسیار مختصر تزریقات زمان نوجوونی دید ه‌م و در حد این آمپول بسیار بسیار ساده، بلد بو... | Tue Jul 20 08:25:25 +0000 2021 |
| > | 20 | RT @rez16_7: رهبر انقلاب : شما پرستاران عزیز حقیقتاً فرشتگان رحمت برای همه ی افراد بیمار و بیماردار جامعه محسوب میشوید #همیار_سلامت | Tue Jul 20 08:25:25 +0000 2021 |

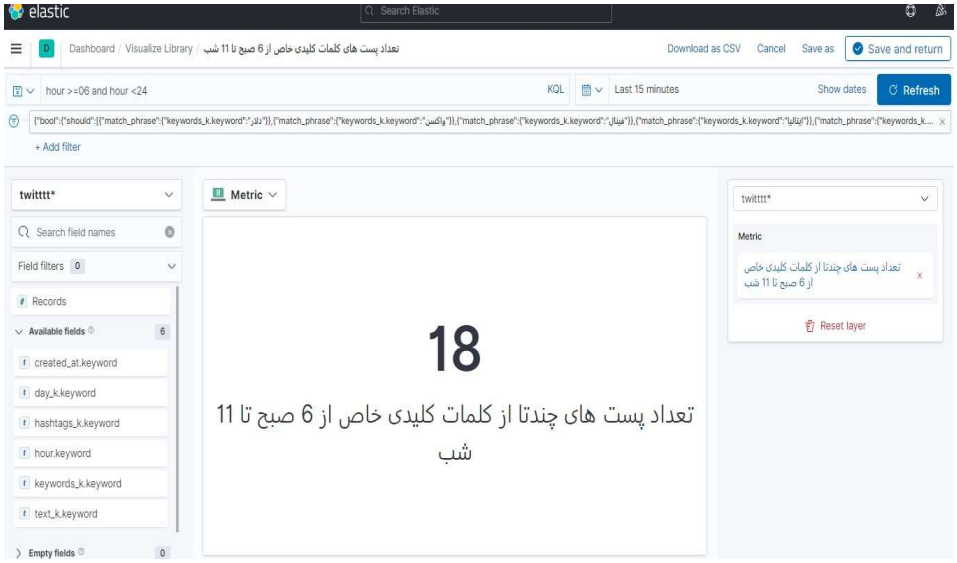
- در قسمت سوم ازما خواسته شده است که تعداد پست های ذخیره شده به ازای چندتا از کلمات کلیدی خاص را در یک بازه زمانی مشخص کنیم. از انجایی که تمام کلمات کلیدی خاص مدنظر مثل تحریم در ان بازه ای که من توییتر می گرفتم، وجود نداشت، یک سری کلمات کلیدی خاص دیگر(۱۱تا کلمه ) در کلمات کلیدی ازگام اول ست شده اند از جمله دلار، واکسن، ایران ، درمان، ایتالیا و...را شمردم.

برای این مرحله در همان داشبوردی که تا حالا ساختیم،ابتدا از قسمت Visualize library یک Visualization جدید ایجاد می کنیم و lens را انتخاب می کنیم:




برای این قسمت چون می خواهیم تعداد را نمایش دهیم از متریک استفاده می کنیم. و لازم است یکسری فیلتر انجام دهیم به این صورت که فقط یک سری از کلمات کلیدی خاص را در بازه زمانی ۶ صبح تا ۱۱ شب انتخاب می کنیم و سیو و به داشبورد اضافه می کنیم می بینیم که در بازه ی زمانی ۱۸ تا پست که شامل این کلمات کلیدی هستند، وجود دارند.



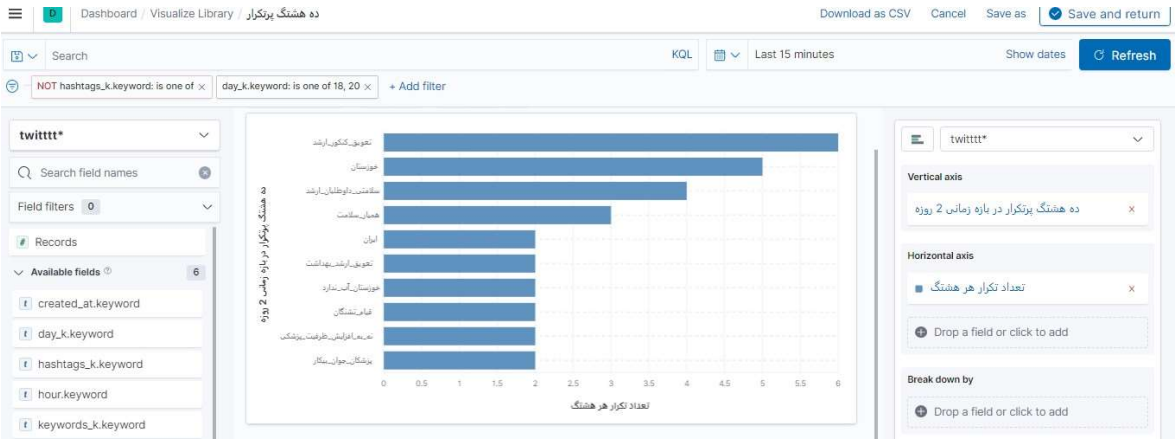




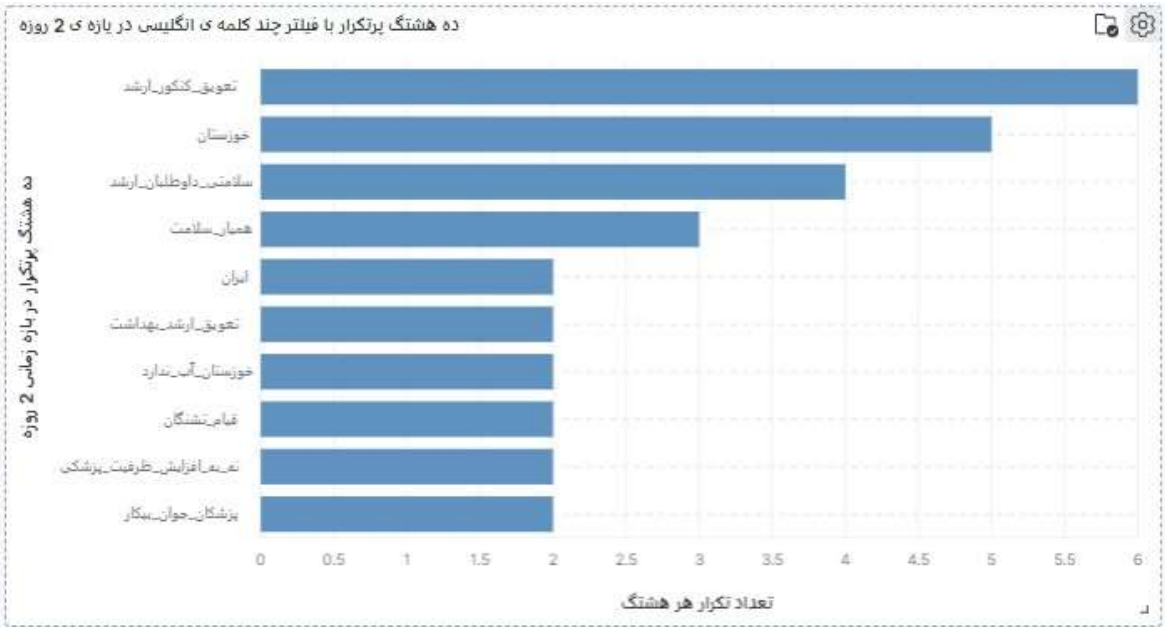
- در قسمت چهارم ازما خواسته شده است که ۱۰ هشتگ پرتکرار در یک بازه زمانی نمایش دهیم که بازه ی زمانی را ۲ روز اخیر فیلتر می کنیم و دونوع نمودار رسم می کنیم.

برای این مرحله در همان داشبوردی که تا حالا ساختیم،ابتدا از قسمت Visualize library یک Visualization جدید ایجاد می کنیم و lens را انتخاب می کنیم:

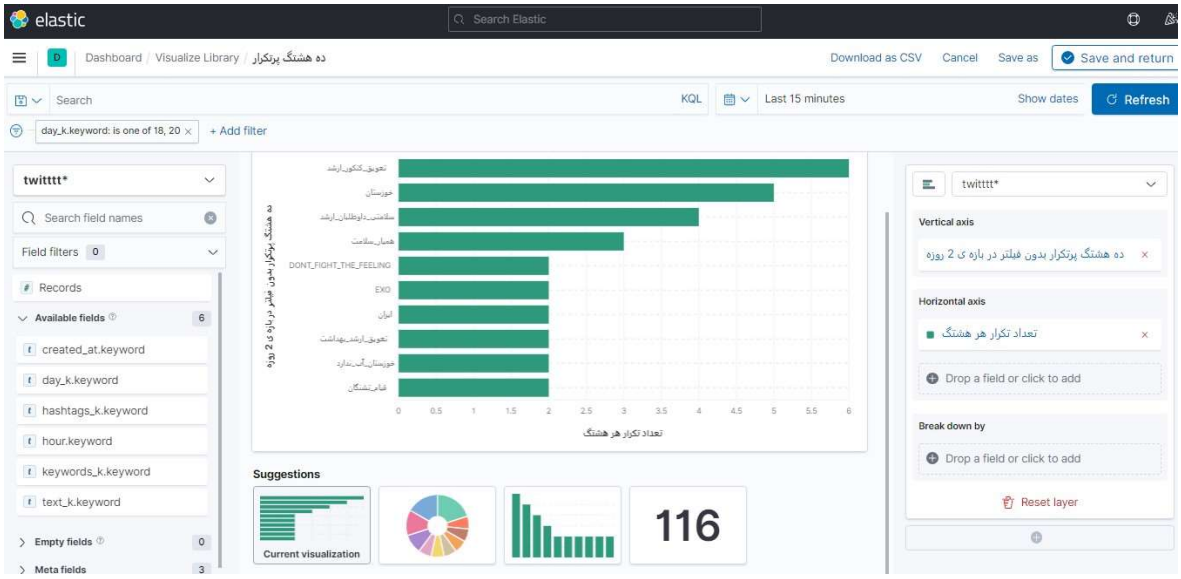
در نمودار اول، یک سری هشتگ های انگلیسی را فیلتر کردم. از بارپلات افقی استفاده کردم چون بعضی از هشتگ ها، نام های طولانی داشتند. روی هرکدام از هشتگ ها در نمودار بزنیم، تعداد را نمایش می دهد.(کلا ۱۱۶ توییت در این ۲ روز داریم)

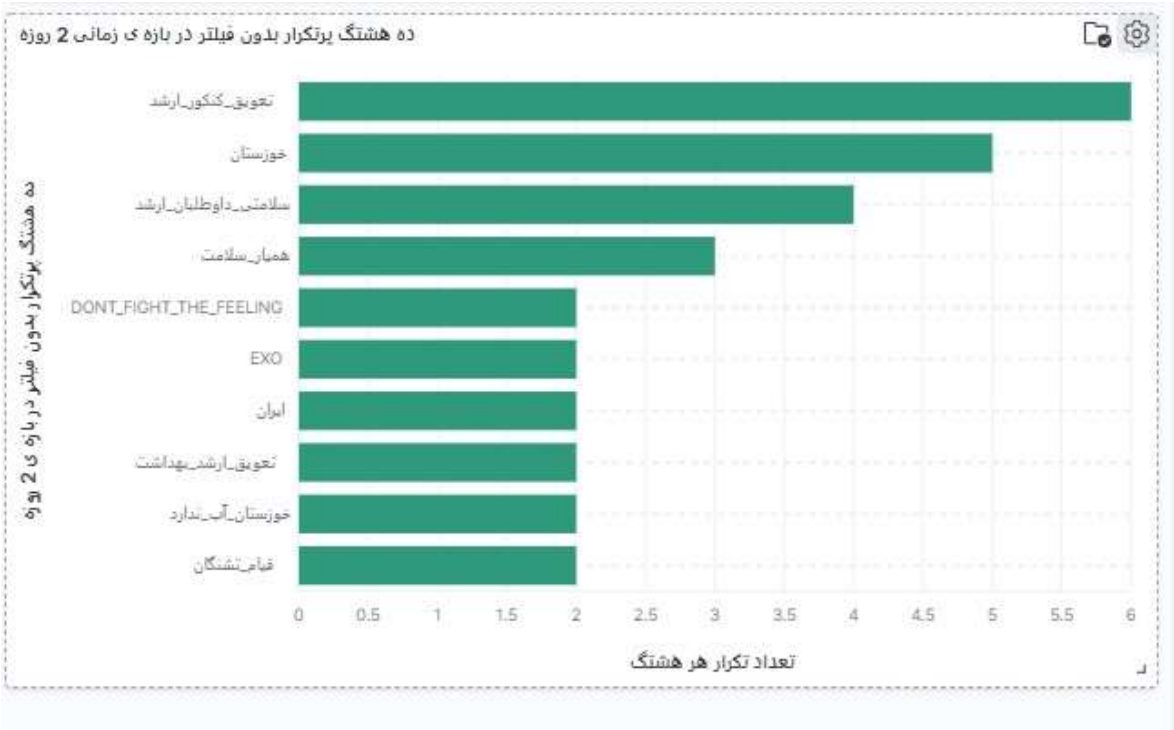



سپس این نمودار را به داشبورد اصلی اضافه می کنیم

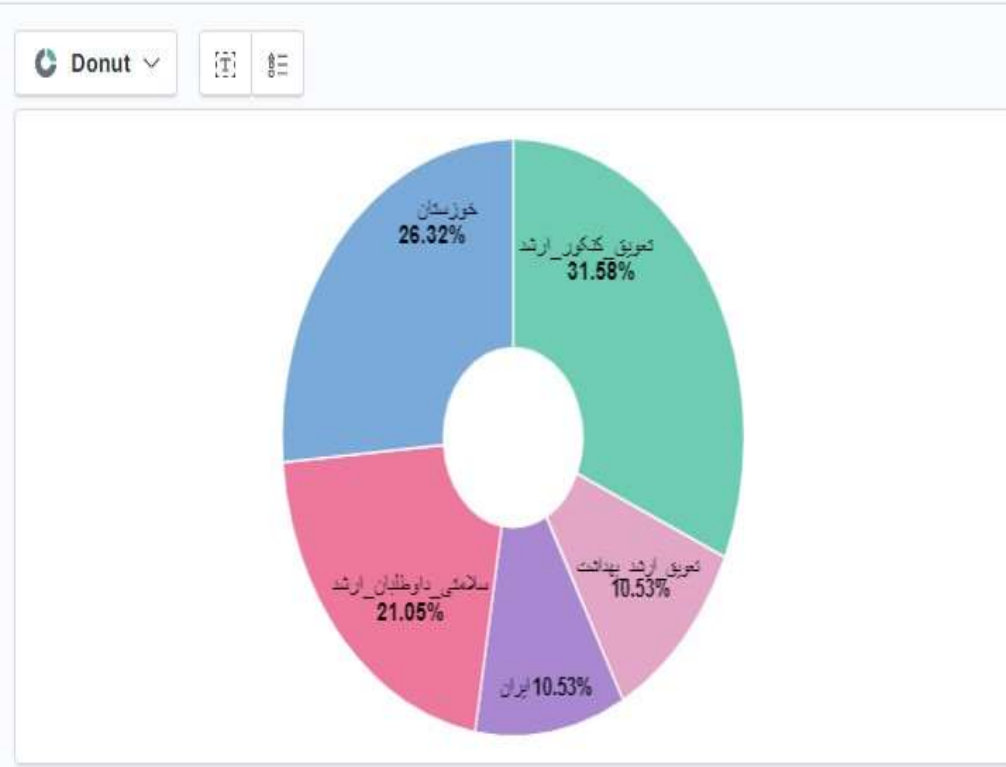


در قسمت دوم هشتگ های انگلیسی را فیلتر نمی کنیم



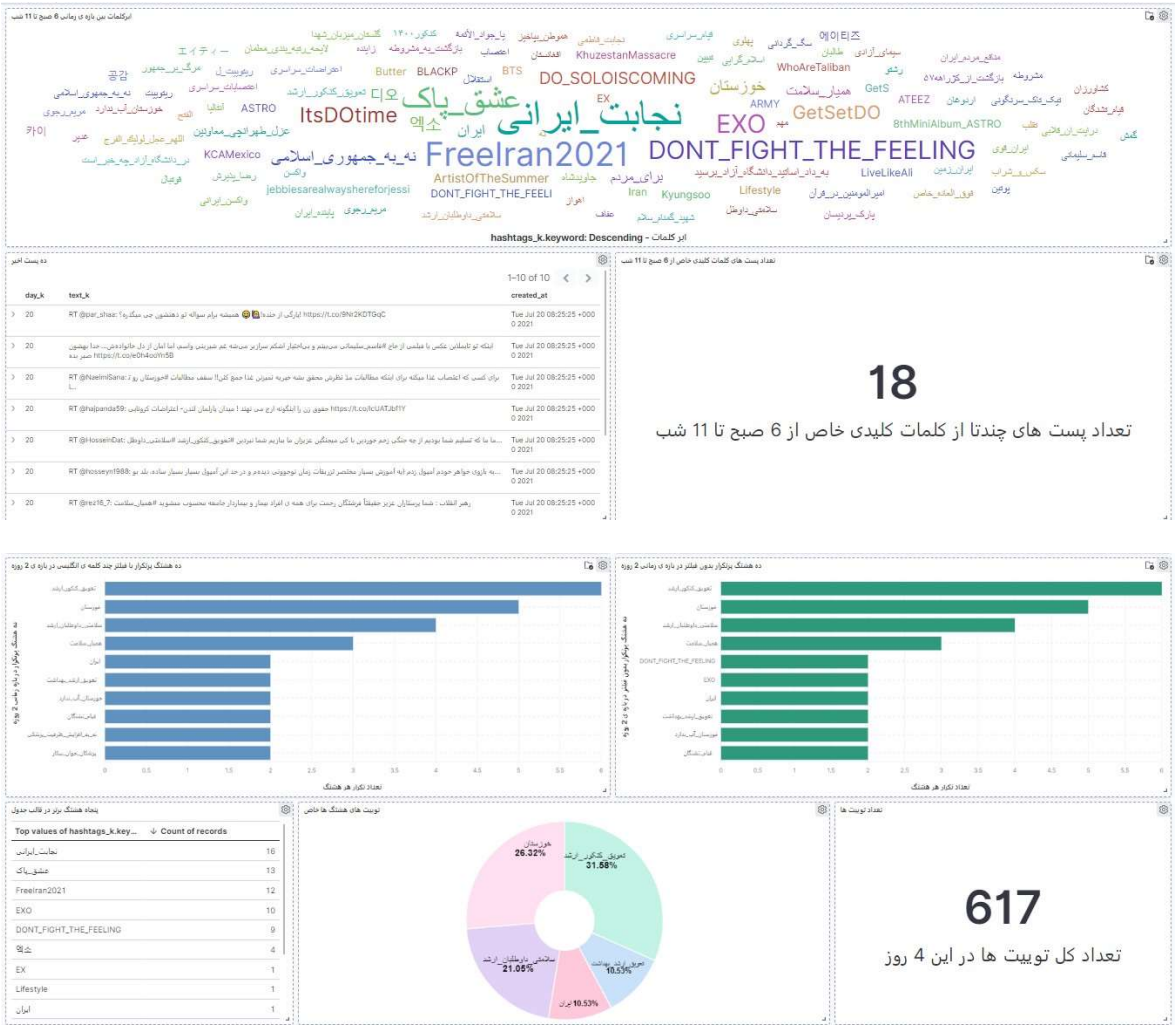
- در قسمت پنجم و اخر از طراحی داشبورد، ازما خواسته شده که یک نمودار به دلخواه انتخاب کنیم که من از donut چارت استفاده می کنم و چند هشتگ خاص را بین کل هشتگ های موجود در توییت ها نمایش می دهم که ببینیم حدودا چند درصد توییت ها به این هشتگ ها اختصاص داشت سپس به داشبورد اضافه می کنیم تا داشبورد تکمیل شود::




- می توانیم نمودار های دیگر مثل table چارت استفاده کنیم تا ۵۰ هشتگ برتر را نمایش دهیم:


- و در اخرین قسمت از داشبورد تعداد کل توییت ها را دریافت می کنیم:

:::::شمای کلی داشبورد:::::




## بخش ششم: کوئری های خواسته شده در کنسول

در بخش اخر از این گام ازما ۲ کوئری خواسته شده است::

۱. در کوئری اول می خواهیم تمام پست های حاوی یک هشتگ خاص را در یک بازه ی زمانی مشخص کنیم. برای اینکار از bool و عبارت منطقی must معادل and استفاده می کنیم چون می خواهیم ۲ شرط را چک کنیم که هم در بازه ی زمانی (range) (بین روز یازدهم و

دوازدهم) باشد و هم شامل هشتگ خاص به اسم (نجابت(ایرانی) باشد.( در این بازه ای که من دیتا میگرفتم، بحث حجاب خیلی زیاد بود)

```
###_____________________ all posts of tweets for specific hashtag _______________

GET /twitttt/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "range": {
            "day_k": {
              "gte": "11",
              "lte": "12"
            }
          }
        },
        {
          "term": {
            "hashtags_k": {
              "value": "نجابت_ایرانی"
            }
          }
        }
      ]
    }
  }
}
```

همانطور که از ریزالت زیر می بینیم ۱۶ تا پست هستند که شامل این هشتگ هستند و انها را نمایش می دهیم:

ریزالت:

```
{
  "took" : 195,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 16,
      "relation" : "eq"
    },
    "max_score" : 3.2481294,
    "hits" : [
      {
        "_index" : "twitttt",
        "_type" : "_doc",
        "_id" : "TZk9lHoBZgdUVjGzRSuA",
        "_score" : 3.2481294,
        "_source" : {
          "created_at" : "Sun Jul 11 06:12:12 +0000 2021",
          "keywords_k" : [ ],
          "hashtags_k" : [
            "نجابت_ایرانی"
          ],
          "text_k" : """RT @mohadeseh_mv: حجاب و عفاف یکی از محورهای زندگی قرآنی است.
#نجابت_ایرانی""",
          "hour" : "06",
          "day_k" : "11"
```


۲.در کوئری دوم می خواهیم تعداد پست های یک هشتگ خاص در یک بازه ی زمانی بیابیم که من تعداد پست هشتگ های ایران را در این بازه ی زمانی (بین روز یازدهم و دوازدهم) را نمایش می دهم(چون می خواهیم تعداد را نمایش دهیم از agg استفاده می کنیم و هشتگ را فیلتر می کنیم)

```
GET /twitttt/_search
{
  "query": {
    "range": {
      "day_k": {
        "gte": "11",
        "lte": "12"
      }
    }
  },
 "size": 0,
 "aggs": {
      "source" : {
        "filter" : {
          "term": {
            "hashtags_k.keyword": "ایران"}}

        }}}
```

ریزالت( می بینیم که تعداد توییت های این هشتگ ۲ تا بود)

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 501,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "source" : {
      "doc_count" : 2
    }
  }
}
```


مثلا تعداد تکرار کل هشتگ را در بازه ی زمانی بین روز یازدهم و دوازدهم هم می توانیم ببینیم::

```
###_____________________count of tweets for specific hashtag _________________

GET /twitttt/_search
{
  "query": {
    "range": {
      "day_k": {
        "gte": "11",
        "lte": "12"
      }
    }
  },

  "size":0,
  "aggs": {
    "source": {
      "terms": {
        "field": "hashtags_k.keyword",
        "size": 100000
      }
    }
  }
}
```

ریزالت را در زیر می بیینم که تعداد توییت های که به ازای هر هشتگ هستند.

```
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 501,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "source" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "نجابت_ایرانی",
          "doc_count" : 16
        },
        {
          "key" : "عشق_پاک",
          "doc_count" : 13
        },
        {
          "key" : "FreeIran2021",
          "doc_count" : 12
        },
        {
          "key" : "EXO",
          "doc_count" : 8
        },
        {
          "key" : "DONT_FIGHT_THE_FEELING",
```


.

.

.

.

```
346 *        },
347 *        {
348            "key" : "واکسن_ایرانی",
349            "doc_count" : 1
350 *        },
351 *        {
352            "key" : "یا_جواد_الأئمة",
353            "doc_count" : 1
354 *        },
355 *        {
356            "key" : "پارک_پردیسان",
357            "doc_count" : 1
358 *        },
359 *        {
360            "key" : "پاینده_ایران",
361            "doc_count" : 1
362 *        },
363 *        {
364            "key" : "پوتین",
365            "doc_count" : 1
366 *        },
367 *        {
368            "key" : "کشاورزان",
369            "doc_count" : 1
370 *        },
371 *        {
372            "key" : "کنکور۱۴۰۰",
373            "doc_count" : 1
374 *        },
375 *        {
376            "key" : "گلستان_میزبان_شهدا",
377            "doc_count" : 1
378 *        },
379 *        {
380            "key" : "گمش",
381            "doc_count" : 1
```


پایان بخش دوم

## گام سوم : کانال Hashtag_history

### بخش اول: تنظیمات داکریِ cassandra

در امتداد با بخش قبل برای اینکه داخل یک container تا حد ممکن کلاستر را بالا بیاوریم imageِ کاساندرا را فایل YML مطابق شکل زیر اضافه میکنیم :

docker-compose.yml M

docker-compose.yml

```yaml
version: '2'
services:

  cassandra:
    image: cassandra:latest
    container_name: cassandra
    ports:
      - "9042:9042"
    environment:
      - "MAX_HEAP_SIZE=256M"
      - "HEAP_NEWSIZE=128M"
    restart: always
    volumes:
      - ./docker_data/cassandra-data:/var/lib/cassandra

  # elasticsearch:
  #   image: elasticsearch:7.12.1
  #   container_name: elasticsearch
  #   environment:
  #     - xpack.security.enabled=false
```

برای انجام این گام به صورت local نیازی به داده های ذخیره شده در الستیک نداریم پس میتوانیم برای بالا رفتن سرعت سیستم الستیک و کیبانا را موقتاً غیر فعال کنیم (اگرچه در روند انجام کامل پروژه

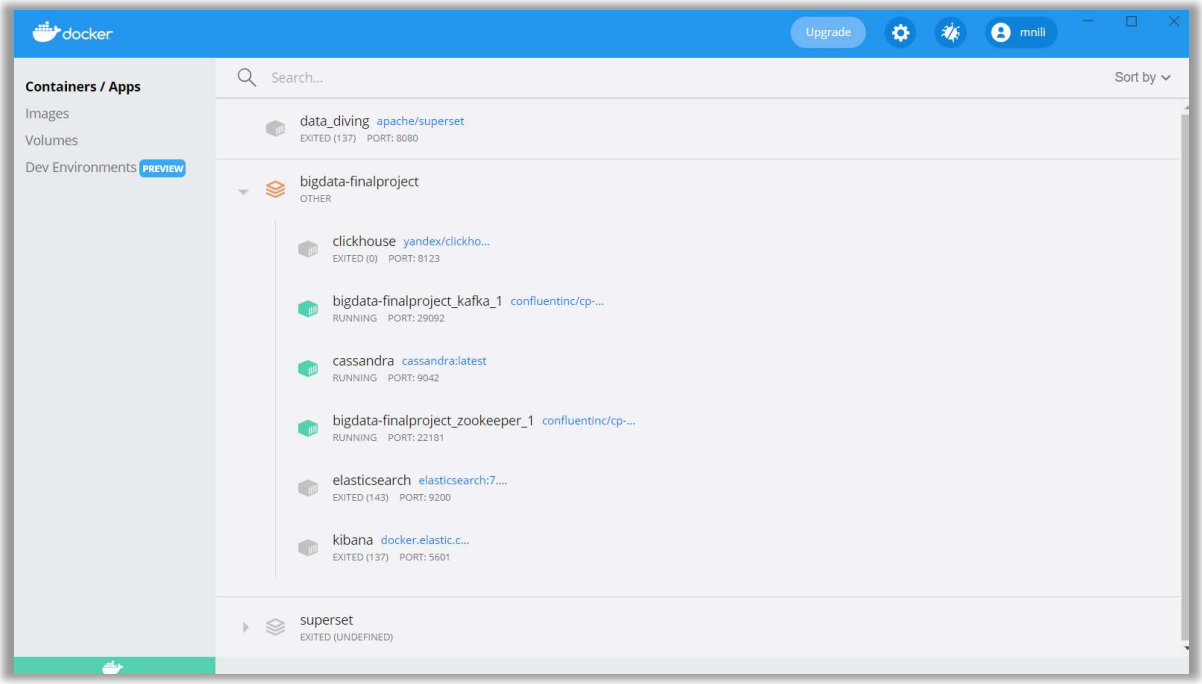
سایر بخش ها فعال هستند) ، و در نهایت با دستور docker-compose up کلاستر ما با سایر image های روی آن up میشوند مطابق با شکل زیر داریم :

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

PS C:\Users\Lenovo\Desktop\AZYN\bigdata-finalproject> docker-compose up
WARNING: Found orphan containers (elasticsearch, kibana, clickhouse) for this project. If you removed or renamed this service in your compose fi
le, you can run this command with the --remove-orphans flag to clean it up.
Starting bigdata-finalproject_zookeeper_1 ... done
Starting cassandra                        ... done
Starting bigdata-finalproject_kafka_1     ...
```

برای مشاهده ی لیست process های در حال اجرا نیز میتوانیم با دستور زیر چک کنیم :

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

PS C:\Users\Lenovo\Desktop\AZYN\bigdata-finalproject> docker ps
CONTAINER ID   IMAGE                              COMMAND                  CREATED       STATUS         PORTS
                                                 NAMES
72fa2ac12abd   confluentinc/cp-kafka:latest       "/etc/confluent/dock…"   3 days ago    Up 2 minutes   9092/tcp, 0.0.0.0:29092->29092/tcp, :::29
092->29092/tcp                                   bigdata-finalproject_kafka_1
ed4995121 6f1   cassandra:latest                  "docker-entrypoint.s…"   3 days ago    Up 2 minutes   7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.
0:9042->9042/tcp, :::9042->9042/tcp   cassandra
9252aa55b1aa   confluentinc/cp-zookeeper:latest   "/etc/confluent/dock…"   3 days ago    Up 2 minutes   2888/tcp, 3888/tcp, 0.0.0.0:22181->2181/t
cp, :::22181->2181/tcp                           bigdata-finalproject_zookeeper_1
PS C:\Users\Lenovo\Desktop\AZYN\bigdata-finalproject>
```

همانطور که گفته شد با غیر فعال کردن موقتیِ پورت هایی که در این گام نیاز نداریم تصویری مطابق زیر مشاهده میکنیم :




## بخش دوم: کدنویسی در پایتون ( ست کردن consumer و producer برای کاساندرا)

همانطور که در بخش قبل توضیح داده شد ، به کمک تابع consumer از داده های stream شده از گام قبل (persistence ) داده ها را داخل کانالِ History دریافت کرده و به کمک تابع producer میتوانیم جریان داده ها را حفظ کنیم و برای گام بعد داده تولید کنیم ( بدین معنی که روی داده های دریافتی پردازشی انجام شود و به مرحله بعد برود یا اینکه بدون تغییر همان را به مرحله بعد بفرستیم )

kafka stream > hashtag_history_chnl.py

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 30 12:05:01 2021

@author: nilam
"""

from kafka import KafkaConsumer, KafkaProducer
from cassandra.cluster import Cluster
from datetime import datetime, timezone
from json import loads, dumps
import json
import pprint

######################### Define Kafka clients #########################

# Produce persistent data
producer = KafkaProducer(bootstrap_servers=['localhost:29092'],
                         value_serializer=lambda x:
                         dumps(x).encode('utf-8'),
                         api_version=(0,10))


consumer = KafkaConsumer(
    'History',
    bootstrap_servers=['localhost:29092'],
    auto_offset_reset= 'earliest', # 'earliest', # Start from last consumed, #'latest' start from last produce
    enable_auto_commit=True,
    auto_commit_interval_ms = 1000,
    group_id='twitter',
    value_deserializer=lambda x: loads(x.decode('utf-8')),
    api_version=(0,10))
```

برای حفظ جریان داده ها ما بدون انجام پردازش و تغییری روی داده ها ، داده ها را روی کانال Statistics میفرستیم :

```python
producer.send('Statistics', value=tweet)
```

در ابتدا مطابق با host و port مشخص شده برای image داکر در فایل YML و به کمک تابع Cluster از کتابخانه ی Cassandra.cluster مطابق شکل زیر اتصال برقرار میشود .

```python
from cassandra.cluster import Cluster
```

kafka stream > hashtag_history_chnl.py

```python
######################### cassandra connection ########################

def cassandra_connection():
    cluster = Cluster(['localhost'], port=9042)
    session = cluster.connect()

    session.execute("""CREATE KEYSPACE IF NOT EXISTS big_data_twits
                        WITH REPLICATION ={ 'class' : 'SimpleStrategy', 'replication_factor' : 1 }""")

    #___ posts table « partition keys :(year,month,day) , cluster keys: (hour,minute,second,id) »
    session.execute("""CREATE TABLE IF NOT EXISTS big_data_twits.posts
                        (year int,month int,day int,hour int,minute int,second int, id text,
                        PRIMARY KEY((year,month,day),hour,minute,second,id))""")

    #___ hashtags table « partition keys :(hashtag) , cluster keys: (year,month,day,hour,minute,second,id) »
    session.execute("""CREATE TABLE IF NOT EXISTS big_data_twits.hashtags
                        (hashtag text,year int,month int,day int,hour int,minute int,second int, id text,
                        PRIMARY KEY((hashtag),year,month,day,hour,minute,second,id))""")

    #___ key_words table « partition keys :(keyword) , cluster keys: (year,month,day,hour,minute,second,id) »
    session.execute("""CREATE TABLE IF NOT EXISTS big_data_twits.key_words
                        (keyword text,year int,month int,day int,hour int,minute int,second int, id text,
                        PRIMARY KEY((keyword),year,month,day,hour,minute,second,id))""")

    return session, cluster
```

پیش از هرچه در دیتابیس کاساندرا به ساخت KeySpace یا فضای کلیدِ big_data_twits ( که مطابق با DataBase در دیتابیس های SQL است میپردازیم ، تکرار داده ها را به دلیل اجرا روی local برابر ۱ قرار میدهیم امّا برای اجرا روی سرور بهتر است برای محافظت از داده ها وابسته به فضای موجود تغییر داد .

با توجه به پرسش های مطرح شده برای این بخش ، صرفا به ذخیره ی شناسه ی هر توییت ذخیره میکنیم ، با این کار با توجه به هر جدول (column family) پارتیشن و ترتیب موررد نظر روی توییت ها را داریم که میتوانیم با سرعت بالایی شناسه ی توییت را بازیابی کرده و به کمک الستیک با دادن شناسه به آن توییت مورد نظر را دریافت کنیم. حال تابع ساخته شده را صدا میزنیم تا اتصال برقرار شود.

```python
session, cluster = cassandra_connection()
session.set_keyspace('big_data_twits')
```

این گام نیاز به ۳ جدول ( جدول پست ها یا توییت ها ) ( جدول هشتگ ها ) ( جدول کلمات کلیدی) دارد :

posts columns family (توییت ها) :

به دلیل اینکه میخواهیم توییت های هر روز را در یک سطر گسترده ذخیره کنیم پس از ترکیب (روز و ماه و سال) به عنوان partition key استفاده میکنیم و از آنجا که با داده های stream کار میکنیم پس مرتب سازی (cluster key) را بر اساس زمان دریافت توییت ، به همراه شناسه ی منحصر به فرد توییت انجام میدهیم با این کار زمانی که توییت های یه بازه ی زمانی را به صورت مرتب شده بخواهیم با سرعت بالا در دسترس هستند. پس تعداد سطر ها حداکثر ۳۶۵*۱۲*۳۱ است

hashtags columns family :

برای دسترسی به تمامی توییت هایی که یک هشتگ خاص را دارند این جدول را میسازیم. پس به ازای هر هشتگ یک سطر گسترده خواهیم داشت به بیان دیگر partition key را برابر هشتگ قرار میدهیم و مجدداً به دلیل وجود داده های streame تاریخ و زمان را همراه شناسه به عنوان ( cluster key) در نظر میگیریم تا هشتگ ها بر اساس زمان و شناسه ی توییت ها در هر سطر مرتب شوند. با این کار میتوانیم توییت های حاوی یک هشتگ خاص را در یک بازه زمانی مشخص شده بدست آوریم. تعداد سطر ها به تعداد هشتگ های متفاوت

key_words columns family :

به دلیل اینکه با داده های توییتر کار میکنیم (کانال نداریم !) پس فرض سوال را عوض کرده و به جای کانال از کلمات کلیدی که در توییت ها استفاده شدند استفاده میکنیم و مجددا روی آنها partition key میسازیم ، cluster key را نیز ترتیب زمانی توییت ها همراه با شناسه آنها قرار میدهیم. تعداد سطر ها به تعداد کلمات کلیدی منحصر به فرد است.

## بخش سوم: Consume data

حال داده هایی که وارد کانال History شدند را به جداول کاساندرا اضافه کرده و به کمک تابع produce به کانال Statistics وارد میکنیم :

```
######################## stream twits to static channel #######################

for message in consumer:

    tweet               = message.value
    dtime               = tweet['created_at']
    new_datetime        = datetime.strftime(datetime.strptime(dtime,'%a %b %d %H:%M:%S +0000 %Y'), '%Y-%m-%d %H:%M:%S')
    date ,time          = new_datetime.split(' ')
    year,month,day      = [ int(x) for x in date.split('-')]
    hour,minute,second  = [ int(x) for x in time.split(':')]
    id_str              = tweet['id_str']
    hashtags_k          = tweet['hashtags_k']
    keywords_k          = tweet['keywords_k']

    #____________________ insert to posts table _____________
    session.execute("""INSERT INTO  posts (year,month,day,hour,minute,second,id)
                        VALUES (%s,%s,%s,%s,%s,%s,%s)""", [year,month,day,hour,minute,second,id_str])
    #____________________ insert to hashtags table _________
    if(hashtags_k!=[]):
        for tag in hashtags_k:
            session.execute("""INSERT INTO  hashtags  (hashtag,year,month,day,hour,minute,second, id)
                                VALUES (%s,%s,%s,%s,%s,%s,%s,%s)""", [tag,year,month,day,hour,minute,second,id_str])
    #____________________ insert to keywords table __________
    if(keywords_k!=[]):
        for word in keywords_k:
            session.execute("""INSERT INTO  key_words (keyword,year,month,day,hour,minute,second, id)
                                VALUES (%s,%s,%s,%s,%s,%s,%s,%s)""", [word,year,month,day,hour,minute,second,id_str])

    print("tweet id: "+id_str+" added to cassandra")

    producer.send('Statistics', value=tweet)
```

به ازای هر توییتی در ابتدا زمان آنرا به سال و ماه و روز و ساعت و دقیقه و ثانیه خرد میکنیم سپس تنها ۳ فیلد شناسه توییت و لیست هشتگ ها و لیست کلمات کلیدی را برای افزودن به جداول این بخش نیاز داریم.

به ازای هر توییت به تعداد هشتگ هایش آنرا به جدول هشتگ و سطر مورد نظر اضافه میکنیم ، ممکن است به تکرار شناسه یک توییت به ازای هشتگ های متفاوت آن یا کلمات متفاوت آن فکر کنید اما به دلیل تفاوت در هشتگ یا کلمه کلیدی در partition های جدا ذخیره میشوند و مشکلی پیش نمی آید

، مزیت این روش این است که از هشتگ ها یا کلمات کلیدی تکراری داخل یک توییت خودداری میکند و از تکرار می پرهیزد.

## بخش چهارم: نمایش خروجی پرس و جو ها (queries)

پرس و جوی اول : توییت های یک ساعت اخیر :

queries > cassandra_results_Q1.py

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 05 04:19:37 2021

@author: nilam
"""

from datetime import datetime, timezone
from cassandra.cluster import Cluster
from json import loads, dumps
import pandas as pd
import json
import pprint


def cassandra_connection():
    cluster = Cluster(['localhost'], port=9042)
    session = cluster.connect()
    session.set_keyspace('big_data_twits')

    return session, cluster

session, cluster = cassandra_connection()

###_______________ fetch current time (timezone : UTC)_______
session.set_keyspace('big_data_twits')
current_year  = int(datetime.now(timezone.utc).strftime("%Y"))
current_month = int(datetime.now(timezone.utc).strftime("%m"))
current_day   = int(datetime.now(timezone.utc).strftime("%d"))
current_hour  = int(datetime.now(timezone.utc).strftime("%H"))

###_________________________ fetch last hour tweets _________________
rows = session.execute(f"""SELECT * FROM posts where
                           year={current_year} AND month={current_month} AND day={current_day} AND
                           hour IN ({current_hour},{current_hour})""")
for row in rows:
    print(row)

###_____________________ count of fetch last hour tweets _____________
cnt = session.execute(f"""SELECT count(*) FROM posts where  year={current_year} AND month={current_month} AND day={current_day} AND
                           hour IN ({current_hour},{current_hour})""")
print(f'\n\n fetch {cnt[0][0]} rows from last hour tweets \n\n')
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Row(year=2021, month=7, day=20, hour=3, minute=37, second=4, id='1417327670912856083')
Row(year=2021, month=7, day=20, hour=3, minute=37, second=4, id='1417327671336374273')
Row(year=2021, month=7, day=20, hour=3, minute=37, second=4, id='1417327671705436163')
Row(year=2021, month=7, day=20, hour=3, minute=37, second=5, id='1417327674813558802')
Row(year=2021, month=7, day=20, hour=3, minute=37, second=5, id='1417327676822544385')
Row(year=2021, month=7, day=20, hour=3, minute=37, second=5, id='1417327676889739283')
Row(year=2021, month=7, day=20, hour=3, minute=37, second=5, id='1417327676981919745')
Row(year=2021, month=7, day=20, hour=3, minute=37, second=5, id='1417327677241847808')
Row(year=2021, month=7, day=20, hour=3, minute=37, second=5, id='1417327677388861448')
Row(year=2021, month=7, day=20, hour=3, minute=37, second=5, id='1417327677489377281')
Row(year=2021, month=7, day=20, hour=3, minute=37, second=5, id='1417327677594382363')
Row(year=2021, month=7, day=20, hour=3, minute=37, second=5, id='1417327678307377153')


 fetch 659 rows from last hour tweets
```

در ابتدا زمان فعلی بر حسب گرینویچ به سال و ماه و روز و ساعت در خطوط ۲۶-۳۰ مشخص کرده و سپس تمامی توییت های یک ساعت اخیر را از جدول tweets استخراج میکنیم و به دلیل تعداد زیاد آنها را خط به خط چاپ میکنیم که تعداد توییت های یک ساعت گذشته ۶۵۹ عدد گزارش شده است. دلیل اینکه این جدول برای این پرس و جو انتخاب شد این بود که هر سطر گستره یک روز را در این

جدول مشخص میکرد پس بهترین انتخاب برای این پرس و جو همین جدول است زیرا زمان آن نیز مرتب شده در داخل هر partition موجود است.

پرس و جوی دوم : توییت های یک روز اخیر.

```
36  ###_____________________ fetch 1 last day (last 24 h) tweets _____________
37  rows = session.execute(f"""SELECT * FROM posts where
38                             year={current_year} AND month={current_month} AND
39                             day IN ({current_day-1},{current_day}) LIMIT 100""")
40
41  print('\n\n fetch 1 last day (last 24 h) tweets : \n\n')
42  for row in rows:
43      print(row)
44
45  ###_____________ count of fetch 1 last day (last 24 h) tweets _____________
46  cnt = session.execute(f"""SELECT count(*) FROM posts where  year={current_year} AND month={current_month} AND
47                             day IN ({current_day-1},{current_day})""")
48
49  print(f'\n\n fetch {cnt[0][0]} rows from last hour tweets \n\n')
50
```

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

Row(year=2021, month=7, day=19, hour=0, minute=13, second=6, id='1416913956619395076')
Row(year=2021, month=7, day=19, hour=0, minute=13, second=7, id='1416913957114322945')
Row(year=2021, month=7, day=19, hour=0, minute=13, second=7, id='1416913957944709122')
Row(year=2021, month=7, day=19, hour=0, minute=13, second=7, id='1416913959005868037')
Row(year=2021, month=7, day=19, hour=0, minute=13, second=7, id='1416913960381685762')
Row(year=2021, month=7, day=19, hour=0, minute=13, second=7, id='1416913960813608963')
Row(year=2021, month=7, day=19, hour=0, minute=13, second=8, id='1416913962591981578')
Row(year=2021, month=7, day=19, hour=0, minute=13, second=8, id='1416913964043223043')
Row(year=2021, month=7, day=19, hour=0, minute=13, second=8, id='1416913964961869829')
Row(year=2021, month=7, day=19, hour=0, minute=13, second=9, id='1416913966958317569')


 fetch 7541 rows from last hour tweets
```

به دلیل تعداد بالای توییت های روزانه جهت نمایش و دید بهتر به ۱۰۰ عدد محدود کردیم اما تعداد کل توییت های روز اخیر ۷۵۴۱ عدد بوده است و مجدداً جدول توییت ها بهترین انتخاب است. زیرا تعداد سطر های محدودی به ازای بازه زمانی خاص را میتوانیم از جدول توییت ها انتخاب کنیم (میدانیم اولویت با جدولی است که سطر های بیشتری را برای پرس و جوی ما کنار بگذارد و با تعداد کمی از سطر ها سروکار داشته باشیم)

پرس و جوی سوم : توییت های حاوی یک هشتگ خاص در یک بازه زمانی.

```
###_____________________ fetch most common hashtag (between 1000 tweet) _________________
###        [ cause of Ram limitation to fetch all in pandas , use limit 1000]
#        ---------------------------------------------------------------------------
rows = session.execute(f"""SELECT hashtag,count(*) as cnt FROM hashtags GROUP BY hashtag LIMIT 1000""")
top_tag = pd.DataFrame(rows).sort_values('cnt',ascending=False).iloc[0]['hashtag']

###_____________________ fetch specific hashtag in time range tweets _________________
def time_range_hashtag(yy,mm,dd_s,dd_e,hh_s,hh_e,tag):
    days_interval = str(tuple(list(range(dd_s,dd_e))))
    hour_interval = str(tuple(list(range(hh_s,hh_e))))

    rows = session.execute(f"""SELECT * FROM hashtags where hashtag='{tag}' AND
                               year={yy} AND month={mm} AND day in {days_interval} AND
                               hour IN {hour_interval}""")
    for row in rows:
        print(row)

    ## ___ count ____
    cnt = session.execute(f"""SELECT count(*) FROM hashtags where hashtag='{tag}' AND
                               year={yy} AND month={mm} AND day in {days_interval} AND hour IN {hour_interval}""")


    print(f'\n\n fetch {cnt[0][0]} tweets that have #{tag} hashtag by day in :[{dd_s},{dd_e}] & hour in :[{hh_s},{hh_e}] : \n\n')

#_____________________________________________________________________________
#  Call Func:------ year ,       month    ,   day range   ,  hour range , hashtag -----
time_range_hashtag(yy=2021,mm=current_month,dd_s=17,dd_e=25,hh_s=0,hh_e=9,tag=top_tag)

```

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

Row(hashtag='دشپا_روکنک_قیوچت', year=2021, month=7, day=20, hour=6, minute=48, second=47, id='1417375918776070155')
Row(hashtag='دشپا_روکنک_قیوچت', year=2021, month=7, day=20, hour=6, minute=48, second=48, id='1417375925696671752')
Row(hashtag='دشپا_روکنک_قیوچت', year=2021, month=7, day=20, hour=6, minute=48, second=49, id='1417375928909393922')
Row(hashtag='دشپا_روکنک_قیوچت', year=2021, month=7, day=20, hour=6, minute=48, second=50, id='1417375932264951810')
Row(hashtag='دشپا_روکنک_قیوچت', year=2021, month=7, day=20, hour=6, minute=48, second=52, id='1417375938988421140')
Row(hashtag='دشپا_روکنک_قیوچت', year=2021, month=7, day=20, hour=6, minute=48, second=54, id='1417375949432238118')
Row(hashtag='دشپا_روکنک_قیوچت', year=2021, month=7, day=20, hour=6, minute=48, second=54, id='1417375950766026778')
Row(hashtag='دشپا_روکنک_قیوچت', year=2021, month=7, day=20, hour=6, minute=48, second=57, id='1417375960912048163')
Row(hashtag='دشپا_روکنک_قیوچت', year=2021, month=7, day=20, hour=6, minute=48, second=57, id='1417375962023415813')
Row(hashtag='دشپا_روکنک_قیوچت', year=2021, month=7, day=20, hour=6, minute=49, second=1, id='1417375976825237528')
Row(hashtag='دشپا_روکنک_قیوچت', year=2021, month=7, day=20, hour=6, minute=49, second=3, id='1417375986107232273')


 fetch 213 tweets that have #دشپا_روکنک_قیوچت hashtag by day in :[17,25] & hour in :[0,9] :
```

برای این بخش در ابتدا هشتگِ trend را یافته و سپس به کمک یک تابع با ورودی هشتگ و بازه زمانی آنرا در بین توییت ها جستجو می کنیم . جدول هشتگ به دلیل اینکه هر سطر حاوی یک هشتگ است پس با مشخص بودن هشتگ فقط در آن سطر خاص جستجو میکنیم و مناسب ترین جدول برای این پرسش است.

سپس تعداد کل توییت هایی که آن هشتگ را در بازه زمانی مشخص شده نیز گزارش کردیم.

همانطور که مشاهده میکنیم پرتکرار ترین هشتگ #تعویق_کنکور_ارشد است که ۲۱۳ بار در بازه زمانی ۱۲ نیمه شب تا ۹ صبح بین روز های ۱۷ ام تا ۲۵ ام تکرار شده است.

(خطوط ۴۷ و ۴۸ برای تبدیل ورودی زمان به بازه ی زمانی نوشته شدند)

پرس و جوی چهارم و پنجم : تعداد توییت به ازای هر روز تا هفته گذشته / تعداد توییت ها به ازای هر روز تا ماه گذشته

queries > cassandra_results_Q4_Q5.py

```python
#================================== Accumulative queries ====================================
###_________________________ count of tweets for each day in last week _________________
rows = session.execute(f"""SELECT year,month,day,count(*) FROM posts where
                        year={current_year} AND month={current_month} AND
                        day IN ({current_day-6},{current_day-5},{current_day-4},
                                {current_day-3},{current_day-2},{current_day-1},{current_day})
                        GROUP BY year,month,day""")
df = pd.DataFrame(rows)
print('\n\n count of tweets for each day in last week : \n\n')
print(df.to_string(index=False))

###_________________________ count of tweets for each day in last month _________________
rows = session.execute(f"""SELECT year,month,day,count(*) FROM posts where
                        year={current_year} AND month>={current_month-1} AND month<={current_month}
                        GROUP BY year,month,day
                        ALLOW FILTERING""")
df = pd.DataFrame(rows)
print('\n\n count of tweets for each day in last month : \n\n')
print(df.to_string(index=False))
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
 count of tweets for each day in last week :


year  month  day  count
2021      7   17   3170
2021      7   18   1552
2021      7   19   5842
2021      7   20   1699


 count of tweets for each day in last month :


year  month  day  count
2021      7   19   5842
2021      7   20   1699
2021      7   17   3170
2021      7   18   1552
```

به دلیل اینکه مجدداً حداکر میتوانیم  با ۶ سطر کار کنیم پس جدول توییت ها برای این بخش نیز بهترین انتخاب میباشد و به ازای روز فعلی  تا ۶ روز قبل این سطر ها برسی میشوند

برای پرسش دوم نیز اگرچه این جدول چندان مناسب نیست و بهتر بود به تعداد ماه های سال فقط سطر داشته باشیم اما بین دو جدول دیگر این جدول اولویت دارد ولی به دلیل بهینه نبود query از دستور ALLOW FILTERING استفاده میکنیم تا اجرا شود زیرا به طور تقریبی این پرس و جو بروی ۶۰ سطر انجام میشود.

---

پایان بخش سوم

## گام چهارم: Statistics

### مقدمه

در این گام، با استفاده از دیتابیس ردیس و ابزارهای فلسک و ری‌اکت، یک داشبورد ساده برای نمایش اطلاعات آماری سامانه و بروزرسانی آنها پیاده‌سازی کردیم. ابتدا به صورت خلاصه به معرفی ابزارهای استفاده شده میپردازیم:

ردیس (redis) مخفف عبارت Remote Dictionary Server است. در واقع ردیس یک نوع ساختمانِ داده است که در RAM قرار می‌گیرد و اطلاعات به صورت موقتی در آن ذخیره می‌شوند.

ردیس داده‌ها را با سیستم Key-value نگهداری می‌کند و به لطف این ویژگی از آنجایی که رابطه پیچیده‌ای میان داده‌ها ایجاد نمی‌شود، دسترسی و بازیابی این اطلاعات بسیار ساده‌تر خواهد شد.

فلَسک (Flask) عنوان یک فریموورک وب ساده و سبک و در عین حال قدرتمند برای زبان برنامه‌نویسی پایتون است. فلسک عموماً به عنوان یک ریزچارچوب شناخته می‌شود به این معنی که خصوصیاتی نظیر نگاشت شیء-رابطه‌ای، اعتبارسنجی فرم‌ها و دیگر ویژگی‌هایی که در چارچوب‌های بزرگتر مانند جنگو یافت می‌شوند را ندارد؛ اما در عوض دست توسعه‌دهنده را برای اعمال پیاده‌سازی مورد نظرش کاملاً باز می‌گذارد و در کنار آن امکان گسترش به وسیله تعداد زیادی افزونه‌های طرف سوم را داراست.

ری اکت (reactjs) اساسا یک کتابخونه open-source جاوا اسکریپتی برای ساخت رابط کاربری(user interfaces) برای single page applications(اپلیکیشن های تک صفحه ای) است. این کتابخونه برای مدیریت لایه View برای وب استفاده می شود. همچنین React این امکان رو در اختیار ما قرار می‌دهد تا reusable UI components (کامپوننت های قابل استفاده مجدد رابط کاربری) ایجاد کنیم. React در ابتدا توسط Jord an Walke یکی از مهندسین ارشد فیسبوک ایجاد شد.

جریان اطلاعات و عملیات‌های انجام شده در این گام به صورت خلاصه عبارت است از ورود جریان اطلاعات از گام قبل توسط کافکا، وارد کردن جریان داده ورودی در پایگاه داده ردیس با کلید های

مناسب جهت پاسخ به اطلاعات خواسته شده و گرفتن داده‌ها از ردیس توسط بک‌اند flask و انتقال آنها به فرانت‌اند react توسط REST API.


در طول این گام از ابزار گرافیکی redismin به عنوان محیط گرافیکی برای ردیس استفاده شده است. برای استفاده از آن، یک پروکسی خاص که توسط خود redismin ارائه شده است را در محیط داکر بالا آورده و توسط آن، ردیس موجود روی ماشین را به سایت redismin متصل کردیم.

### جریان اطلاعات و عملیات‌های انجام شده

ابتدا داده‌ها بصورت یک استریم بر بستر کافکا بوسیله گام قبل در کانال statistics تولید می‌شوند. سپس یک پردازش(redis_chnl.py) از استریم داده ورودی استفاده می‌کند و به ازای هر داده ورودی(توییت)، اطلاعات آن داده را در ردیس ذخیره می‌کند. برای ذخیره سازی داده‌های مورد نیاز در ردیس، به ازای هریک از سوالات خواسته شده، سازوکاری طراحی شده و اطلاعات(کلیدها و مقادیر آنها) بر اساس آن سازوکار در ردیس ذخیره می‌شوند. سازوکار طراحی شده برای هر بخش، در ادامه توضیح داده خواهد شد.

پس از استخراج اطلاعات و ذخیره آنها در ردیس، استریم داده ورودی در کانال an alytics برای استفاده گام بعدی قرار داده می‌شود.

برای دسترسی به اطلاعات ذخیره شده در ردیس، یک وب اپلیکیشن پیاده‌سازی شده‌است که از flask برای قسمت سمت سرور یا backend و از react برای قسمت سمت کلاینت یا frontend استفاده میکند. در زمان وارد شدن به داشبورد، react با استفاده از api های تعریف شده سمت سرور، اطلاعات مورد نیاز را در قالب Json دریافت کرده و به صورت مناسب نمایش می‌دهد.

برنامه سمت سرور نیز با هر درخواست از سمت کلاینت، اطلاعات خواسته شده را از ردیس دریافت کرده و در پاسخ به فرانت‌اند برمی‌گرداند.

تمامی مراحل ذخیره داده‌ها در ردیس در فایل kafka_stream/redis_chnl.py انجام می‌شود.

فایل بک‌اند در app/backend/app.py/ می‌باشد و تمامی مراحل دریافت داده‌ها از ردیس در این فایل انجام می‌شود.

فایلها و کد های بخش فرانت‌اند در فولدر app/frontend/ قرار دارند.

### درخواست اول: تعداد پستها/توییت‌های ارسال شده یک کانال خاص در شش ساعت گذشته

برای این درخواست، به علت وجود نداشتن کانال در محیط این پروژه (توییت‌های توییتر)، یا باید هر شخص را به عنوان کانال در نظر می‌گرفتیم یا اینکه هر کدام از کلمات کلیدی را به عنوان یک کانال در نظر می‌گرفتیم که تصمیم بر این شد تا راه دوم را انتخاب کنیم.

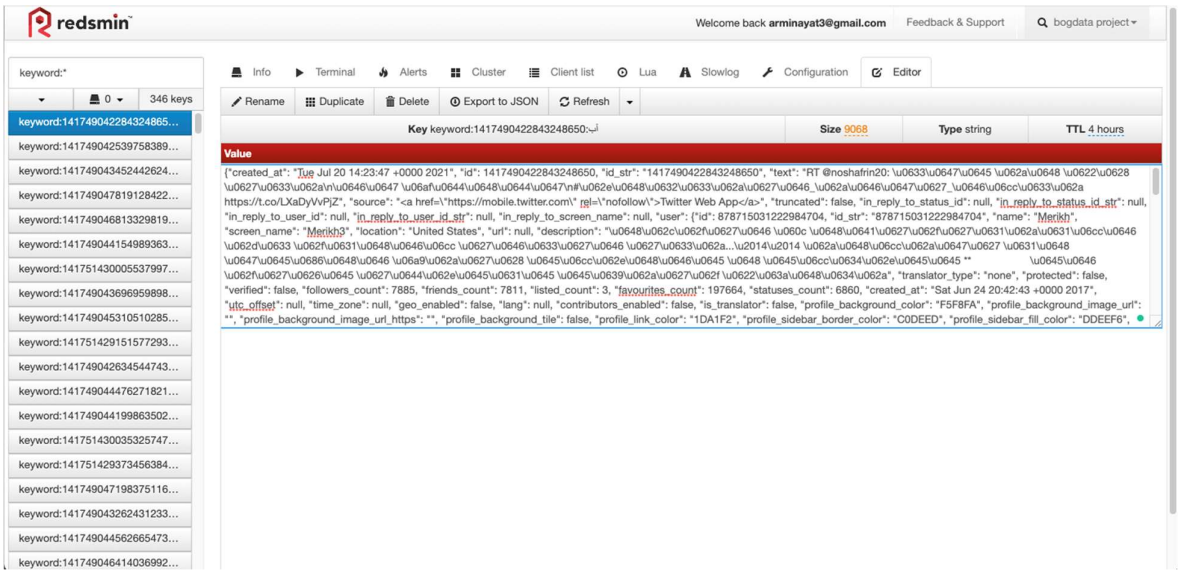
برای ذخیره کردن داده‌های مورد نیاز این درخواست از ساختار داده string در ردیس استفاده کردیم؛ به ازای هر توییت در استریم ورودی، کلمات کلیدی آن توییت و آیدی منحصر به فرد آن توییت را استخراج کردیم و به عنوان کلید آن در ردیس قرار دادیم و کل توییت را به عنوان مقدار آن کلید قرار دادیم.

همچنین از ویژگی expire time ساختار داده string در ردیس استفاده کردیم تا آن داده را بعد از ۶ ساعت از ردیس حذف کنیم.

علت استفاده از کلمه کلیدی و آیدی در کلید این است که بتوانیم هر توییت را به همراه کلمه کلیدی استفاده شده در آن به صورت منحصر به فرد (هر توییت به ازای تمام کلمات کلیدی‌اش یکبار ذخیره می‌شود) ذخیره کنیم تا هم کلمات کلیدی توییت‌ها را داشته باشیم و هم توییت‌های همه کلمات کلیدی را.

## نحوه ذخیره سازی در ردیس:

```
def store_tweet_by_keyword(tweet):
    keywords = tweet["keywords_k"]
    t_id = tweet["id"]
    for keyword in keywords:
        r.set(f"keyword:{keyword}:{t_id}", dumps(tweet), ex=60 * 60 * 6)
```




برای دریافت داده‌های این درخواست، با استفاده از دستور keys تمام کلید-مقدارهایی را که با کلمه keyword ذخیره می‌شوند را گرفته و سپس اطلاعات را از لیستِ بازگردانده شده استخراج میکنیم.

علت استفاده کلمه keyword در اول تمام کلید های این بخش نیز این است که بتوانیم به کمک آن تمامی داده‌های مختص این درخواست را یکجا دریافت کنیم.

همچنین مزیت دیگر این نوع ذخیره سازی این است که می‌توانیم علاوه بر تعداد توییت‌ها، به کل محتوای تک تک توییت‌ها نیز دسترسی داشته باشیم.

### نحوه دریافت داده از ردیس و ارسال به فرانت‌اند:

```
@app.route('/keywords')
def get_tweets_by_keyword():
    keywords = [x.decode().split(":")[1] for x in r.keys("keyword:*")]
    keywords_counts = {}
    for k in keywords:
        keywords_counts[k] = keywords.count(k)
    resp = make_response(jsonify(keywords_counts), 200)
    resp.headers['Access-Control-Allow-Origin'] = '*'
    return resp
```

### درخواست دوم: تعداد کل پستها/توییت‌های دریافت شده در یک بازه زمانی مثلا روز گذشته

برای اینکه بتوانیم در یک بازه زمانی مشخص توییت‌های ذخیره شده را دریافت کنیم ابتدا تاریخ نشر هر توییت را استخراج کردیم.

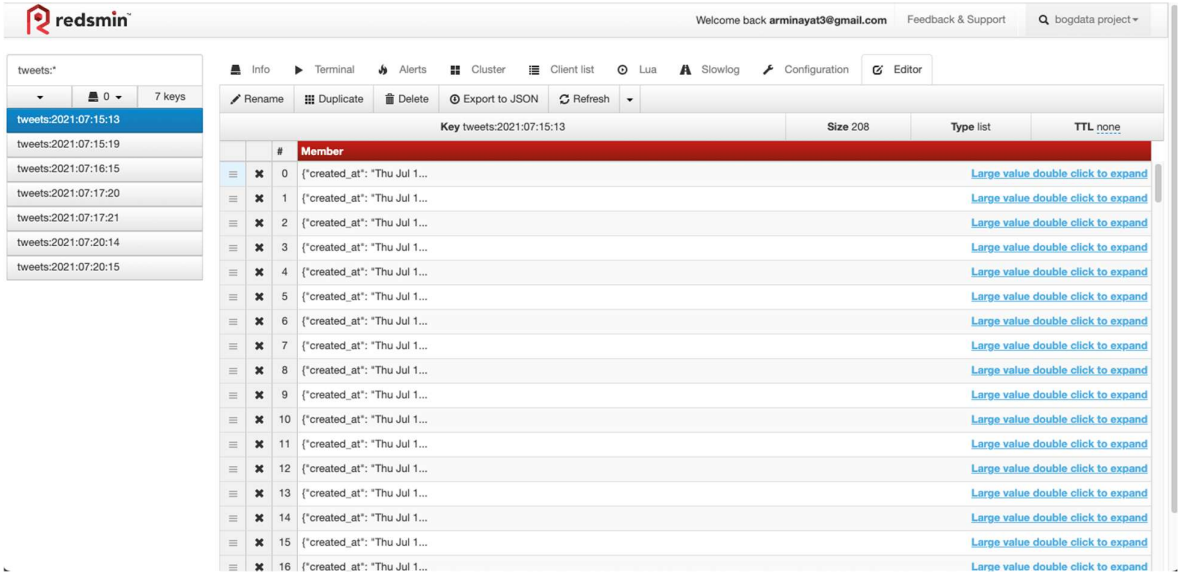
در این قسمت از ساختار داده لیست در ردیس استفاده کردیم. استفاده از لیست این امکان را به ما می‌دهد تا بتوانیم یک مجموعه از داده را به صورت مرتب شده در اختیار داشته باشیم و با استفاده از کلید آن لیست به همه داده‌های ذخیره شده در آن لیست دسترسی داشته باشیم .

زمان استخراج شده هر توییت را به فرمت Y:m:d:H برای کلید لیست‌ها استفاده کردیم. این نوع کلیدگذاری این امکان را به ما می‌دهد تا با داشتن یک بازه زمانی مشخص، بتوانیم تمام توییت های آن بازه را بدست بیاوریم؛ به این صورت که تمام بازه‌های یکساعته در بازه زمانی داده شده را بدست آورده و لیست متناظر با آنها را از ردیس میگیریم، سپس با ادغام کردن لیست ها به تمامی توییت های آن بازه زمانی می‌رسیم.

همچنین به علت استفاده از لیست، توییت های درون هر لیست نیز به ترتیب زمان وارد شده به آن لیست هستند.

### نحوه ذخیره داده ها در ردیس:

```python
def store_tweet_by_time(tweet):
    time_key = datetime.strftime(datetime.strptime(
        tweet["created_at"], '%a %b %d %H:%M:%S +0000 %Y'), '%Y:%m:%d:%H')
    r.lpush(f"tweets:{time_key}", dumps(tweet))
```




برای دریافت داده‌ها، فرانت‌اند بازه زمانی مشخص شده را به صورت پارامتر به بک‌اند می‌دهد. سپس بک‌اند تمامی بازه‌های زمانی یک‌ساعته درون بازه زمانی انتخابی کاربر را محاسبه کرده و لیست متناظر با آنها را از ردیس دریافت می‌کند. سپس با ادغام تمامی لیست‌های دریافتی، آنها را به فرانت‌اند برمی‌گرداند.

در این قسمت نیز به دلیل ذخیره کردن کل توییت‌ها به عنوان محتوا، علاوه بر تعداد توییت‌ها در هر بازه زمانی، به کل محتوای توییت‌ها نیز دسترسی داریم.

### نحوه دریافت داده‌ها:

```
@app.route('/time-filter')
def get_tweets_by_time():
    start = [int(x) for x in request.args.get("start").split("-")]

    end = [int(x) for x in request.args.get("end").split("-")]
    date = end.copy()
    query_dates = []
    while start ≤ date:
        query_dates.append(":".join(["{:02d}".format(x) for x in date]))
        app.logger.info(str(date))

        date[0] = date[0] if (date[1] ≠ 1 or date[2] ≠
                                1 or date[3] ≠ 0) else date[0] - 1
        date[1] = date[1] if (date[2] ≠ 1 or date[3] ≠
                                0) else date[1] - 1 if date[1] > 1 else 12
        date[2] = date[2] if date[3] ≠ 0 else date[2] - \
            1 if date[2] > 1 else 30
        date[3] = date[3] - 1 if date[3] > 0 else 24

    data = []
    for d in query_dates:
        data.extend(r.lrange(f"tweets:{d}", 1, -1))

    resp = make_response(jsonify([loads(x.decode()) for x in data]), 200)
    resp.headers['Access-Control-Allow-Origin'] = '*'
    return resp
```

### درخواست سوم: تعداد هشتگ‌های دریافت شده در یکساعت گذشته به صورت منحصر بفرد

در این گام از ساختار داده string در ردیس استفاده کردیم.

در این گام، چون می‌خواهیم داده‌ها منحصر به فرد باشند و هر هشتگ یکبار بیشتر ثبت نشود، فقط از خود هشتگ در کلید استفاده کردیم زیرا هرکلید باید در ردیس منحصر به فرد باشد و کلید تکراری نداریم.

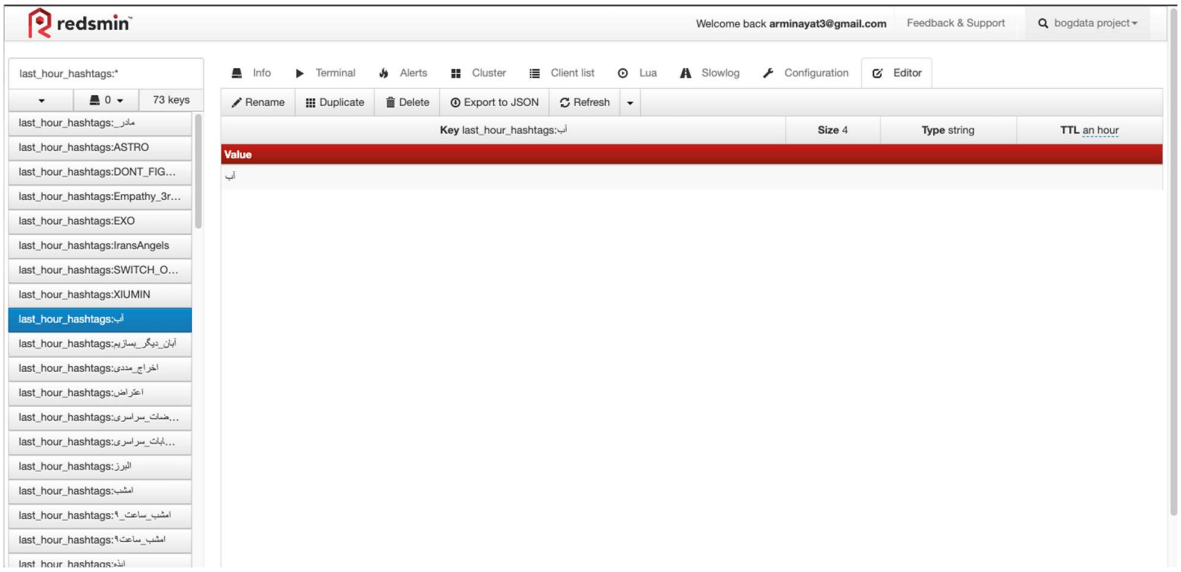
چون فقط با تعداد هشتگ‌ها کار داریم، مقدار داده در ردیس را نیز همان هشتگ گذاشتیم (می‌توانست هرچیز دیگری باشد، ما فقط از کلیدها برای شمارش استفاده می‌کنیم) و به ازای تمام هشتگ‌های هر توییت این کار را انجام دادیم.

همچنین پس از هربار ست کردن یک هشتگ، تاریخ انقضای آنرا بروزرسانی کردیم تا در یک‌ساعت بعد حذف شود.

در انتها نیز با استفاده از پارامتر nx مشخص کردیم تا اگر از قبل کلیدی با این نام وجود داشت، دیگر داده درون آن بروزرسانی نشود و تنها تاریخ انقضای آن بروزرسانی شود.

### نحوه ذخیره داده ها در ردیس:

```
def store_last_hour_hashtags(tweet):
    hashtags = tweet["hashtags_k"]
    for hashtag in hashtags:
        r.set(f"last_hour_hashtags:{hashtag}", hashtag, nx=True)
        r.expire(f"last_hour_hashtags:{hashtag}", 60 * 60)
```




برای دریافت داده‌ها نیز تمامی کلیدهایی که با کلمه last_hour_hashtags ذخیره شده‌اند را دریافت کرده و به فرانت‌اند برمی‌گردانیم.

### نحوه دریافت داده‌ها:

```
@app.route('/hashtags')
def get_last_hour_hashtags():
    keys = [x.decode().split(":")[1] for x in r.keys("last_hour_hashtags:*")]
    resp = make_response(jsonify(keys), 200)
    resp.headers['Access-Control-Allow-Origin'] = '*'
    return resp
```

درخواست چهارم: آخرین هشتگهای دریافت شده، یک لیست هزارتایی که با ورود داده‌های جدید، قدیمی‌ها حذف می‌شوند

برای این درخواست از ساختار داده لیست در ردیس استفاده شده است.

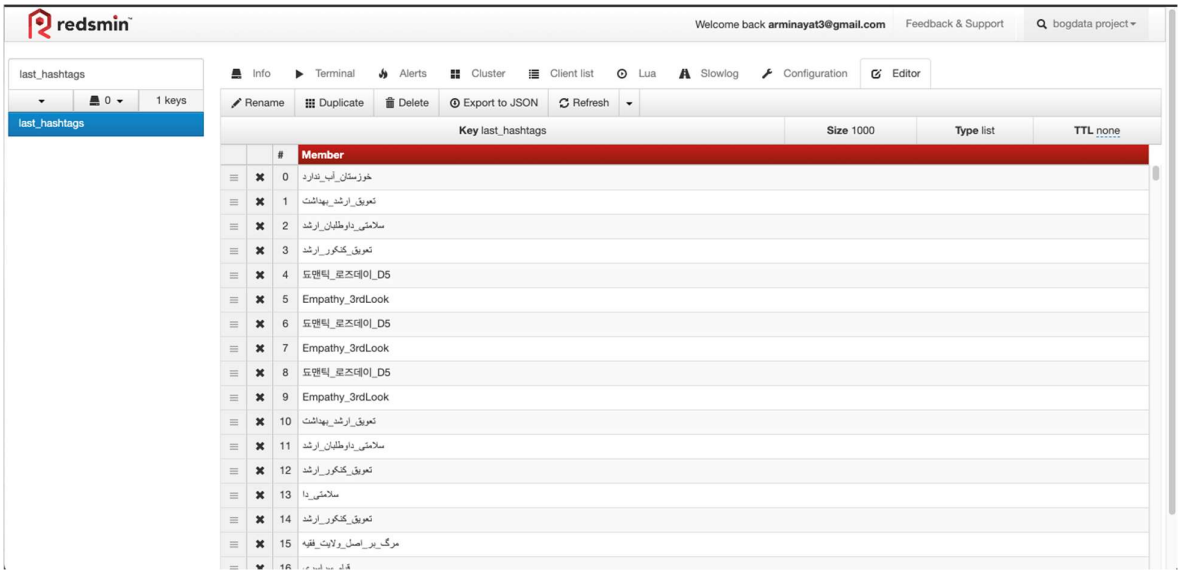
تفاوت این درخواست این است که باید همیشه ۱۰۰۰ داده آخر لیست را نگه داریم. برای این کار، پس از اضافه کردن داده جدید به لیست، هربار داده ۰ تا ۹۹۹ را از لیست نگه داشته و بقیه را حذف می‌کنیم و چون لیست ترتیب ورود را حفظ میکند، ۱۰۰۰ داده آخر نگه داشته می‌شوند.

برای ذخیره داده‌ها تنها یک لیست با کلید last_hashtags ایجاد کرده و با روش فوق، داده‌ها را در آن ذخیره می‌کنیم.

در این سوال نیز به دلیل ذخیره کردن کل توییت‌ها به عنوان محتوا، علاوه بر تعداد توییت‌ها دربازه زمانی، به کل محتوای توییت‌ها نیز دسترسی داریم اما در داشبورد تنها هشتگ‌ها نمایش داده می‌شوند.

### نحوه ذخیره داده ها در ردیس:

```
def store_last_hashtags(tweet):
    hashtags = tweet["hashtags_k"]
    for hashtag in hashtags:
        r.lpush("last_hashtags", hashtag)
        r.ltrim("last_hashtags", 0, 999)
```




برای دریافت داده‌ها کافیست با کلید last_hashtags مقادیر داخل لیست را از ردیس دریافت کرده و آنرا به فرانت‌اند برگردانیم.

### نحوه دریافت داده‌ها:

```
@app.route('/last-hashtags')
def get_last_hashtags():
    data = [x.decode() for x in r.lrange("last_hashtags", 0, -1)]
    resp = make_response(jsonify(data), 200)
    resp.headers['Access-Control-Allow-Origin'] = '*'
    return resp
```

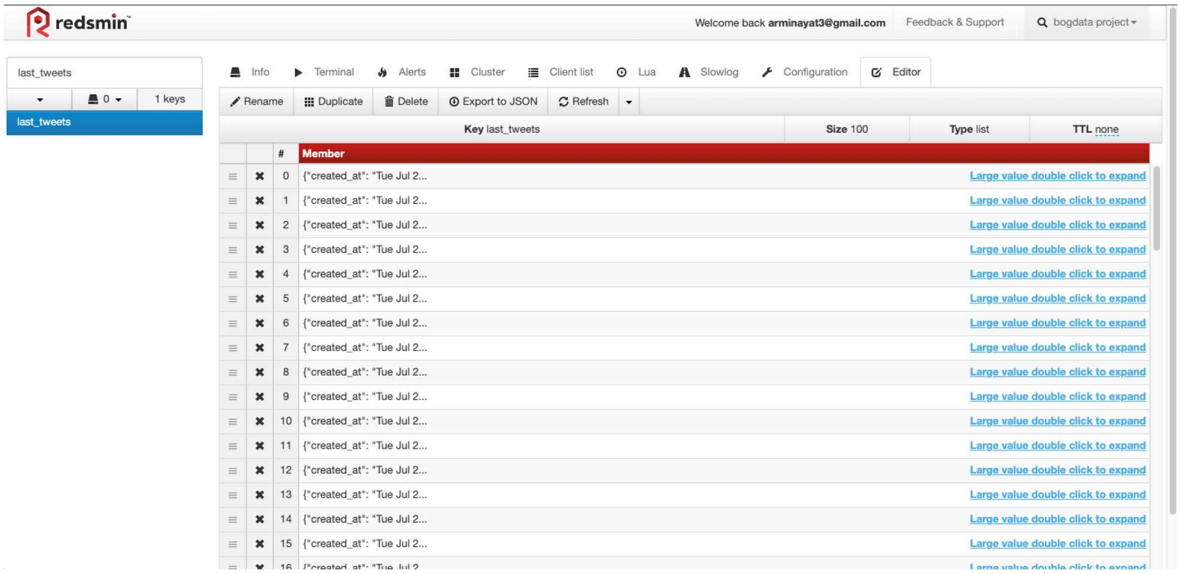
درخواست پنجم: آخرین پستها/توییت‌های دریافت شده، یک لیست صدتایی مشابه درخواست چهار

نحوه عملکرد در این بخش دقیقا همانند بخش چهار می‌باشد با این تفاوت که داده‌ها را در لیستی با کلید last_tweets ذخیره کرده و به جای نگه داشتن ۱۰۰۰ عنصر آخر، ۱۰۰ عنصر آخر را نگه داشته و برمی‌گردانیم.

در سمت فرانت‌اند نیز کل توییت نمایش داده می‌شود.

**نحوه ذخیره داده ها در ردیس:**

```
def store_last_tweets(tweet):
    r.lpush("last_tweets", dumps(tweet))
    r.ltrim("last_tweets", 0, 99)
```


redsmin
Welcome back arminayat3@gmail.com
Feedback & Support
bogdata project
last_tweets
0
1 keys
last_tweets
Info
Terminal
Alerts
Cluster
Client list
Lua
Slowlog
Configuration
Editor
Rename
Duplicate
Delete
Export to JSON
Refresh
Key last_tweets
Size 100
Type list
TTL none
#
Member
{"created_at": "Tue Jul 2...
Large value double click to expand

### نحوه دریافت داده‌ها:

```
@app.route('/last-tweets')
def get_last_tweets():
    data = [loads(x.decode()) for x in r.lrange("last_tweets", 0, -1)]
    resp = make_response(jsonify(data), 200)
    resp.headers['Access-Control-Allow-Origin'] = '*'
    return resp
```

### بخش سمت کلاینت (frontend)

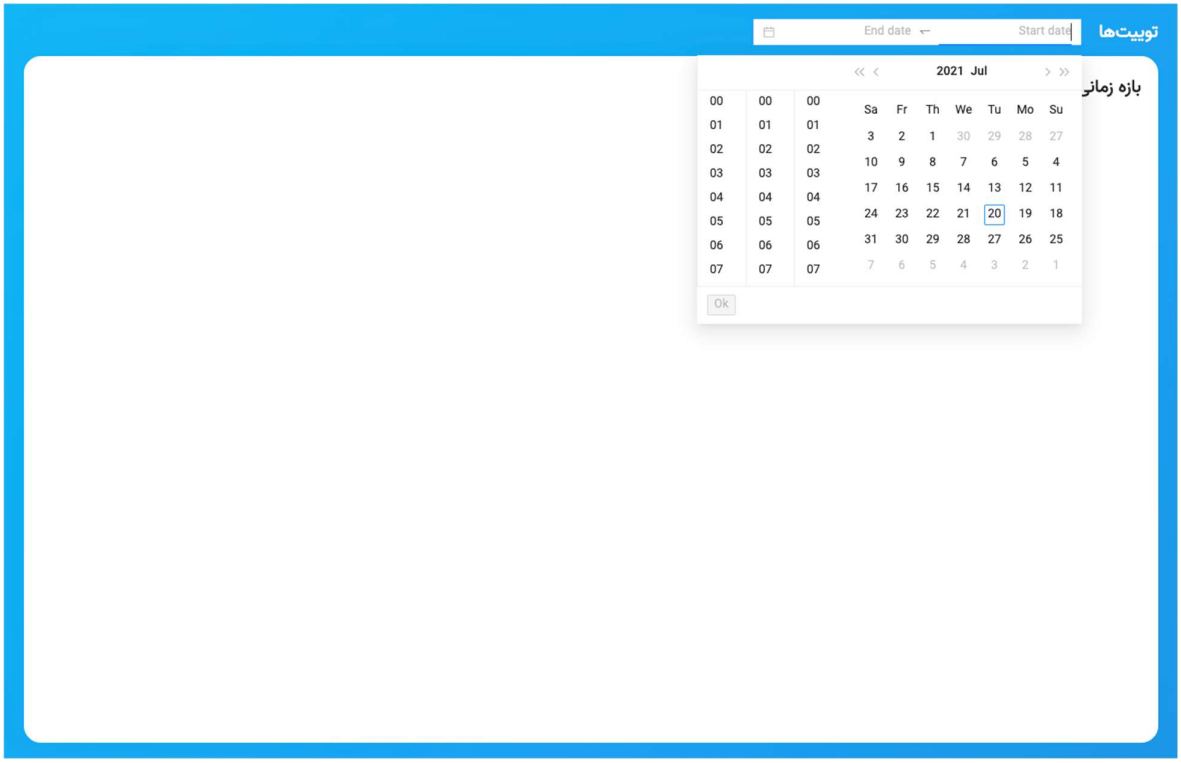
نمای کلی داشبورد به شکل زیر است:

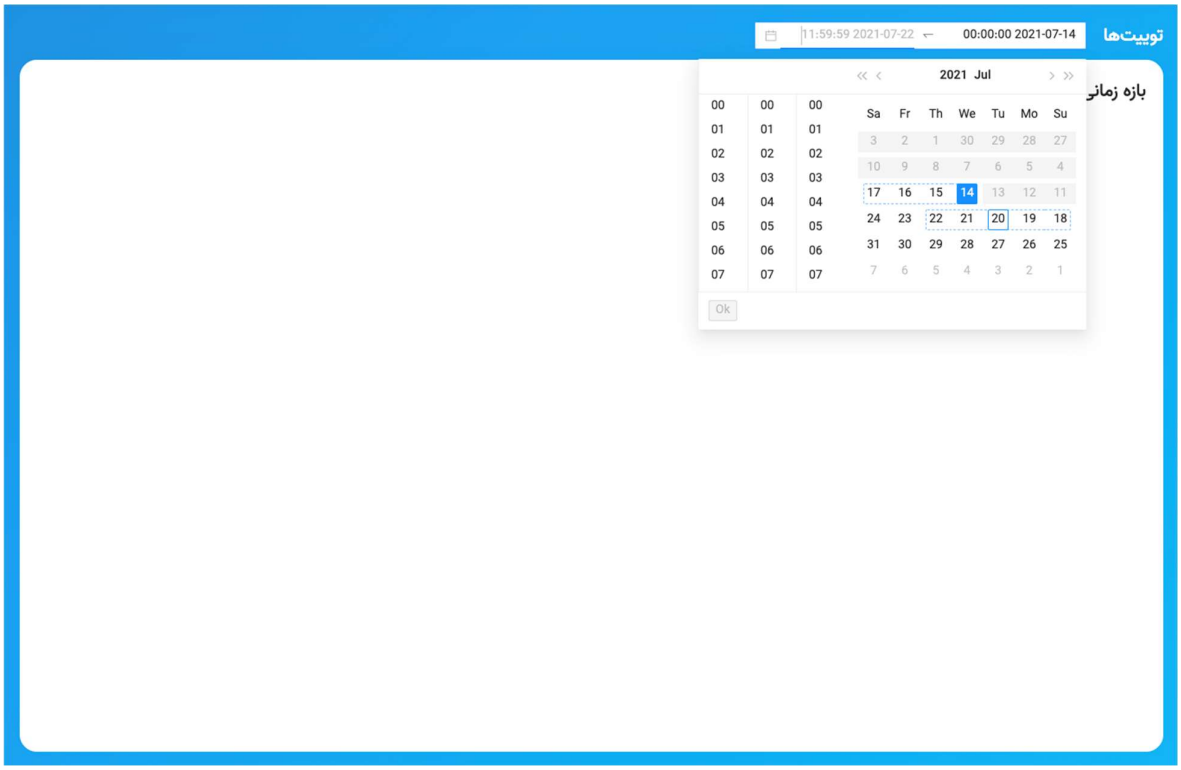

آخرین بروزرسانی در 8 ثانیه قبل
داشبورد توییتر
۱۰۰ توییت اخیر
Noori_sadat72
RT @za_shams: به آسمان نرسد کس مگر ز راه حسین... السلام علیک یاباعبدالله التماس دعا #اللهم_عجل_لولیك_الفرج #عرفه https://t.co/o61CKcofiR
amirsalar
RT @belujin2: هشتگ #همیار_سلامت میزنی ولی #تعویق_کنکور_ارشد رو نادیده میگیری و حمایتش نمیکنی؟ باور کنید هدف هر دو هشتگ در یه راستاست
محسن بیات‌زنجانی
RT @SharghDaily: واکنش یکی از مراجع تقلید قم به #اعتراضات_خوزستان آیت‌الله بیات زنجانی: تا کی بناست به مردم تذکر دهیم مواظب باشید از حرکا...
حواصیل سبز(دخترپاییز)
RT @hoseinyarloo: من حس ششم ضعیفی دارم ولی همیشه چیزارو تقریبا درست حدس میزنم با این فرمول که هرچی حسم میگه رو برعکسشو حدس میزنم🙄🤦‍♂️ htt...
ربوتیل
RT @ARDESHIR003: جوووون چه حاااالی میده تایم لاین من😍 #امشب_ساعت۹
aida.ax
@oneliinnnn @zaynpayne25 دم در منتظرم https://t.co/nlWBDhC3Vp
MoRtEzA
@Zahra_Hmi1999 کیلویی چند
عروس رویاهای مادرشوهرم
RT @iamsetaysh: واقعا فکر میکنید ما پول نداریم خودمون آب معدنی بخریم؟
artemis
RT @Miss_Maropelle: . به نام ایران به یاد تو #امشب_ساعت_۹ #خوزستان_تنها_نیست https://t.co/FTBpF21v9A
NaCl
یکیتون زن کن ادامز بشه تا هممون رو نکرده.
۱۰۰۰ هشتگ اخیر
#عرفه
#اللهم_عجل_لولیك_الفرج
#تعویق_کنکور_ارشد
#همیار_سلامت
#اعتراضات_خوزستان
#امشب_ساعت۹
#خوزستان_تنها_نیست
تعداد کلمات کلیدی شش ساعت اخیر
آب
4
آبادان
1
آخر
1
آزاد
2
آهسته
1
هشتگ‌های یکساعت اخیر
تعداد 22
#همیار_سلامت
#عید_الأضحی_المبارك
#اعتراضات_خوزستان
#سلامت_داوطلبان_ارشد
#براندازی
#نه_به_جمهوری_اسلامي
#الشعب_يريد_إسقاط_النظام
توییت‌ها
End date
Start date
بازه زمانی را انتخاب کنید

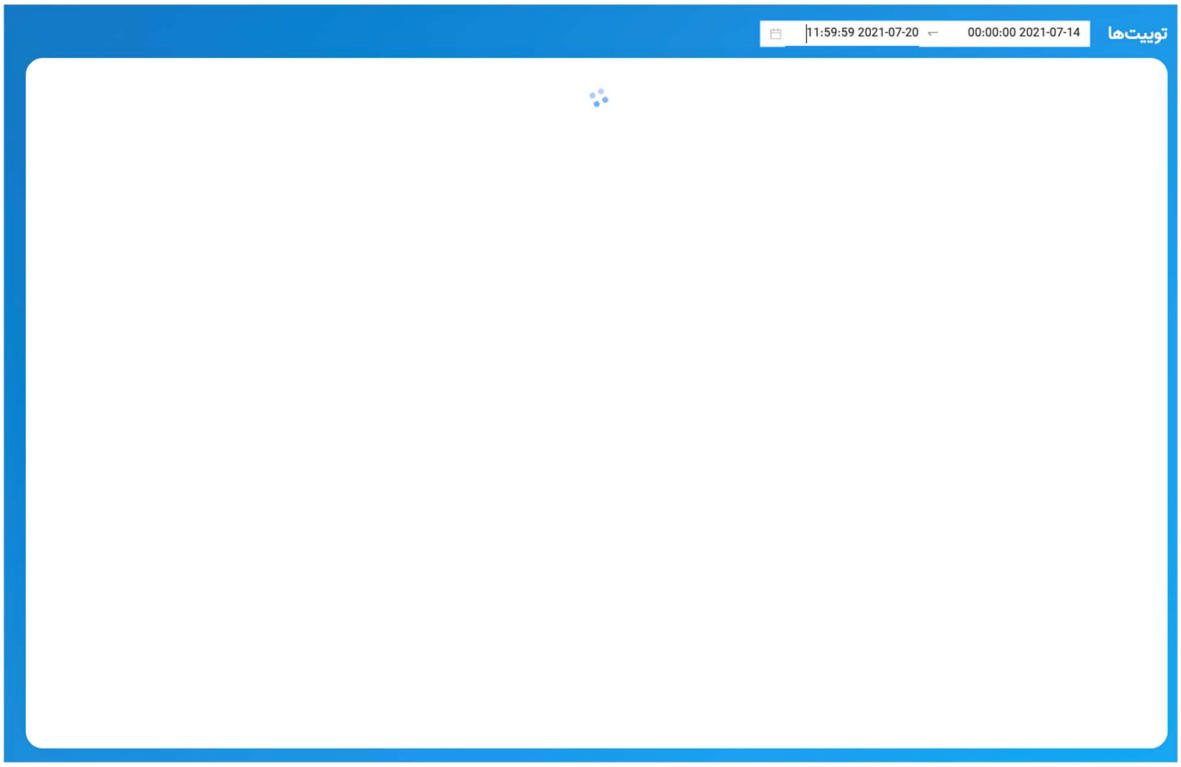
هرکدام از ماژول‌های صفحه با کوئری به API بک‌اند، داده مورد نیاز خود را دریافت کرده و به صورت مناسب نمایش می‌دهند.

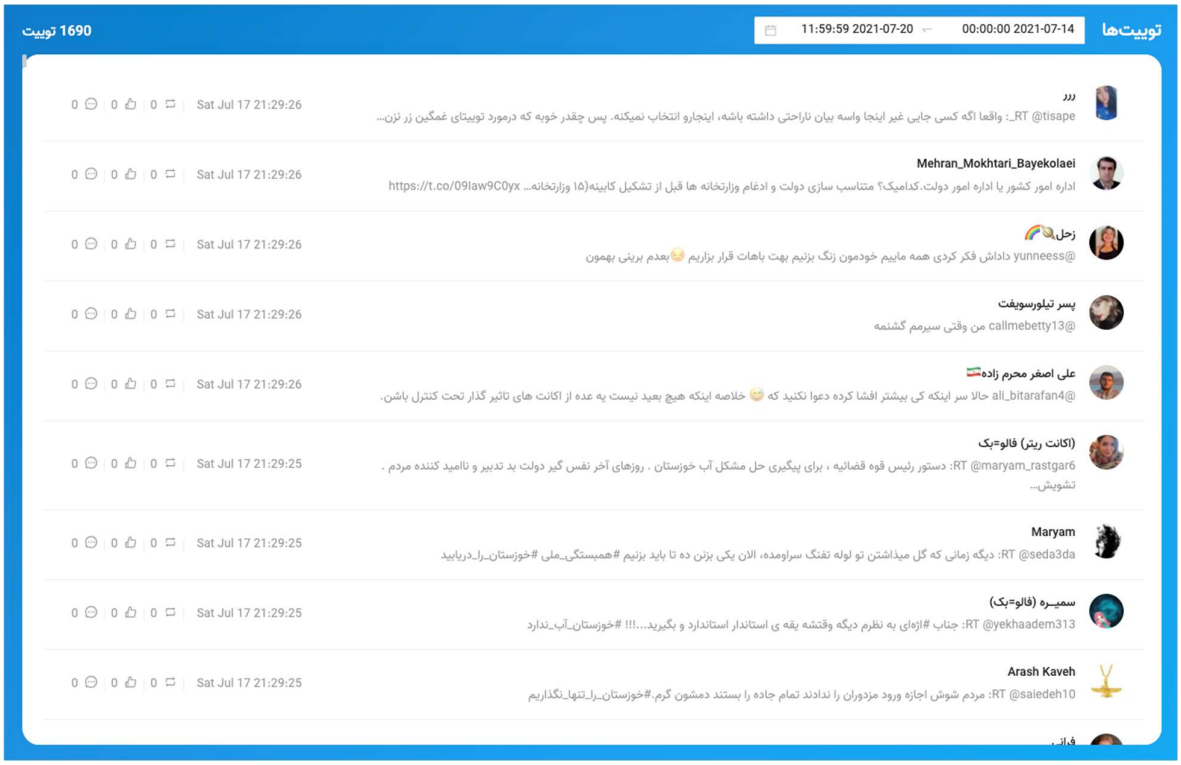
همچنین کل اطلاعات داشبورد به صورت خودکار هر ۱۰ ثانیه یکبار بروزرسانی می‌شوند.

در ماژول آخر، ابتدا کاربر بازه زمانی انتخابی خود را وارد کرده و سپس داده‌های مورد نیاز از بک‌اند دریافت شده و نمایش داده می‌شود:




توییت‌ها
00:00:00 2021-07-14
11:59:59 2021-07-22
بازه زمانی
2021 Jul
Sa Fr Th We Tu Mo Su
Ok


توییت‌ها
00:00:00 2021-07-14
11:59:59 2021-07-20




لازم به ذکر است تمامی داده‌های نمایش داده شده مانند لایک، کامنت و ریتوییت داده‌های واقعی هستند اما به علت اینکه توییت‌ها به محض منتشر شدن توسط کرالر دریافت شده و به سیستم وارد می‌شوند، طبیعتا مقدار همه آنها در لحظه دریافت داده صفر است.

برای اجرای بخش فرانت‌اند در حالت توسعه فایل README.md را در فولدر app/frontend/ مطالعه نمایید.

فرانت اند روی پورت ۵۰۰۱ اجرا می‌شود.

### بخش سمت سرور (backend)

در سمت بک‌اند، برای ارتباط با فرانت‌اند یک API پیاده سازی شده است که اطلاعات آن در ادامه آمده است:

Endpoint: /keywords

Parameters: -

Output sample:


```
// 20210720200051
// http://localhost:5000/keywords

{
  "4 :"آب,
  "1 :"آبادان,
  "1 :"آخر,
  "2 :"آزاد,
  "1 :"آهسته,
  "1 :"اب,
  "1 :"ابرآروان,
  "1 :"ادد,
  "4 :"ارشد,
  "1 :"استار,
  "1 :"استارش,
  "1 :"اسمش,
  "2 :"ال,
  "3 :"ام,
  "1 :"امتحان,
  "2 :"امشب,
  "1 :"امید,
  "1 :"انتخاب,
  "1 :"انشالله,
  "1 :"انگیزه,
  "1 :"اه,
  "1 :"اونیو,
  "2 :"اکانتم,
  "4 :"ای,
  "2 :"ایر,
  "1 :"ایران,
  "1 :"ایم,
  "1 :"اینکه,
  "1 :"ایگنور,
  "1 :"بازگو,
```

End point: /time-filter?start=[startTime]&end=[endTime]

Parameters:

startTime: start of time range in yyyy:mm:dd:hh form at

end Time: end of time range in yyyy:mm:dd:hh form at

Output sample:


```
// 20210720202731
// http://localhost:5000/time-filter?start=2021-07-14-00&end=2021-07-20-12

[
  {
    "contributors": null,
    "coordinates": null,
    "created_at": "Sat Jul 17 21:29:26 +0000 2021",
    "entities": {
      "hashtags": [

      ],
      "symbols": [

      ],
      "urls": [

      ],
      "user_mentions": [
        {
          "id": 1036708859874889728,
          "id_str": "1036708859874889728",
          "indices": [
            3,
            11
          ],
          "name": "الف لام مه",
          "screen_name": "tisape_"
        }
      ]
    },
    "favorite_count": 0,
    "favorited": false,
    "filter_level": "low",
    "geo": null,
    "hashtags_k": [

    ],
    "id": 1416510377517539328,
    "id_str": "1416510377517539328",
    "in_reply_to_screen_name": null,
    "in_reply_to_status_id": null,
    "in_reply_to_status_id_str": null,
    "in_reply_to_user_id": null,
    "in_reply_to_user_id_str": null,
    "is_quote_status": false,
```

Endpoint: /hashtags

Parameters: -

Output sample:


```
// 20210720204539
// http://localhost:5000/hashtags

[
  "엑소디오",
  "됴맨틱_로즈데이_D",
  "حس",
  "تع",
  "ASTRO",
  "Empathy_3rdLook",
  "تعویق_ارشد_بهداشت",
  "امشب_ساعت_۹",
  "اعتصابات_سراسری",
  "دلبرتون",
  "دعای_عرفه",
  "DONT_FIGHT_THE_FEELING",
  "البرز",
  "دوج_کوین",
  "محمدشهر",
  "시우민",
  "아스트로",
  "گوهردشت",
  "EXO",
  "_سکوت_بس",
  "XIUMIN",
  "کادر_درمان_را_دریابید",
  "اخراج_مددی",
  "قیام_تنها_جواب",
  "됴맨틱_로",
  "SWITCH_ON_MOODTRAILER",
  "عرفه",
  "فالو_یک",
  "خوز",
  "دیل_کارنگی",
  "تعویق_کنکور_ارشد",
  "اعتراض",
  "دردمشترک",
  "سواد_رسانه_ای",
  "خوزستان",
  "فوری",
  "مادر_",
  "کرج",
  "امشب",
  "چالوس",
  "شازده_بی",
  "됴맨틱_로즈데",
  "آئین_زندگی",
```

Endpoint: /last-hashtags

Parameters: -

Output sample:


```
// 20210720202907
// http://localhost:5000/last-hashtags

[
  "مادر",
  "تعویق_ارشد_بهداشت",
  "سلامتی_داوطلبان_ارشد",
  "تعویق_کنکور_ارشد",
  "تعویق_کنکور_ارشد",
  "뮤맨틱_로즈데이_D5",
  "Empathy_3rdLook",
  "عرفه",
  "اللهم_عجل_لولیك_الفرج",
  "تعویق_کنکور_ارشد",
  "همیار_سلامت",
  "اعتراضات_خوزستان",
  "امشب_ساعت۹",
  "خوزستان_تنها_نیست",
  "امشب_ساعت_۹",
  "گوته",
  "فاوست",
  "مرگ_بر_جمهوری_اسلامی",
  "الشعب_یرید_إسقاط_النظام",
  "عزل_طهر",
  "به_داد_اساتید_دانشگاه_آزاد_برسید",
  "در_دانشگاه_آزاد_چه_خبر_است",
  "اغتشاشات",
  "SUPERFANFRIDAY",
  "DONT_FIGHT_THE_FEELING",
  "امشب_ساعت_۹",
  "سلامت_داوطلبان_ارشد",
  "سلامتی_داوطلبان_ارشد",
  "تعویق_ارشد_بهداشت",
  "تعویق_کنکور_ارشد",
  "مولانا",
  "مرگ_بر_جمهوری_اسلامی",
  "الشعب_یرید_إسقاط_النظام",
  "تعویق_ارشد_ب",
  "تعویق_کنکور_ارشد",
  "مسعود_رجوی",
  "عرفه",
  "جنگ_رسانه",
  "گوته",
  "فاوست",
  "به_داد_اساتید_دانشگاه_آ",
```

Endpoint: /last-tweets

Parameters: -

Output sample:


```
// 20210720202924
// http://localhost:5000/last-tweets

[
  {
    "contributors": null,
    "coordinates": null,
    "created_at": "Tue Jul 20 15:58:35 +0000 2021",
    "display_text_range": [
      0,
      45
    ],
    "entities": {
      "hashtags": [

      ],
      "media": [
        {
          "display_url": "pic.twitter.com/BSWIHhmhDu",
          "expanded_url": "https://twitter.com/elmyrahoseiny/status/1417514280115572736/photo/1",
          "id": 1417514277267595264,
          "id_str": "1417514277267595264",
          "indices": [
            46,
            69
          ],
          "media_url": "http://pbs.twimg.com/media/E6wFuZCUYAAiCkc.jpg",
          "media_url_https": "https://pbs.twimg.com/media/E6wFuZCUYAAiCkc.jpg",
          "sizes": {
            "large": {
              "h": 486,
              "resize": "fit",
              "w": 512
            },
            "medium": {
              "h": 486,
              "resize": "fit",
              "w": 512
            },
            "small": {
              "h": 486,
              "resize": "fit",
              "w": 512
            },
            "thumb": {
```


برای اجرای سرور فلسک، پس از نصب تمام پیش‌نیازها با استفاده از دستور pipenv sync، به فولدر app/backend/ رفته و با دستور pipenv run flask run سرور فلسک را در حالت توسعه ران کنید.

سرور روی پورت ۵۰۰۰ اجرا می‌شود.

---

پایان بخش چهارم

## گام پنجم: Analytics

(به علت داشتن تنها دو توکن از توییتر و زمان کم، تصمیم گرفته شد علاوه بر اینکه هر کدام از ۴ نفر یک گام جدا از قبل کار کردند، گام های باقی مانده تقسیم شود که این گام به صورت اشتراکی بین من(زهرا حبیب زاده) و اقای محمد نیلی انجام شد. و این گام را به دو بخش بین خودمان تقسیم کردیم.چون توکن دیگر درگیر گام های دیگر بود)

### مقدمه

#### توضیحات کلی clickhouse :

پایگاه داده ClickHouse یک پایگاه داده ساختارمند ستونی (Column-oriented) برای پردازش تحلیلی آنلاین است که با دستورات SQL کار می‌کند که این دیتابیس در پروژه ما کاربرد دارد چون مانند دیتابیس های قبلی تحلیل انلاین انجام می دهد ما نیز داده های خودمان را به صورت انلاین از توییتر دریافت می کنیم و نیاز به دیتابیسی داریم که پردازش را با سرعت زیاد انجام دهد .این دیتابیس مزیت های خوبی دارد.

1. سرعت: در تحلیل بیگ دیتا شاید بتوان بدون اغراق گفت، سرعت پردازش از اهمیت بالایی برخوردار است.
2. ستون های موجود در کوئری را جستجو می کند و جستجو به صورت ستونی است.
3. اطلاعات را فشرده می کند که این فشرده سازی باعث می شود، حجم دیسک کم شود.
4. از خانواده ی MergeTree استفاده می کند (در مرج اطلاعات به صورت دسته ای وارد می شود یعنی تک به تک وارد نمی شود) که نیاز به دستور update نخواهیم داشت و با insert کار خودمان را انجام می دهیم.
5. و....

حال باید دیتابیس جدید ساخته وجدوال مناسب را در clickhouse مناسب با خواسته ی خود طراحی کنیم و انها را به یک اپاچی به نام superset منتقل کنیم:

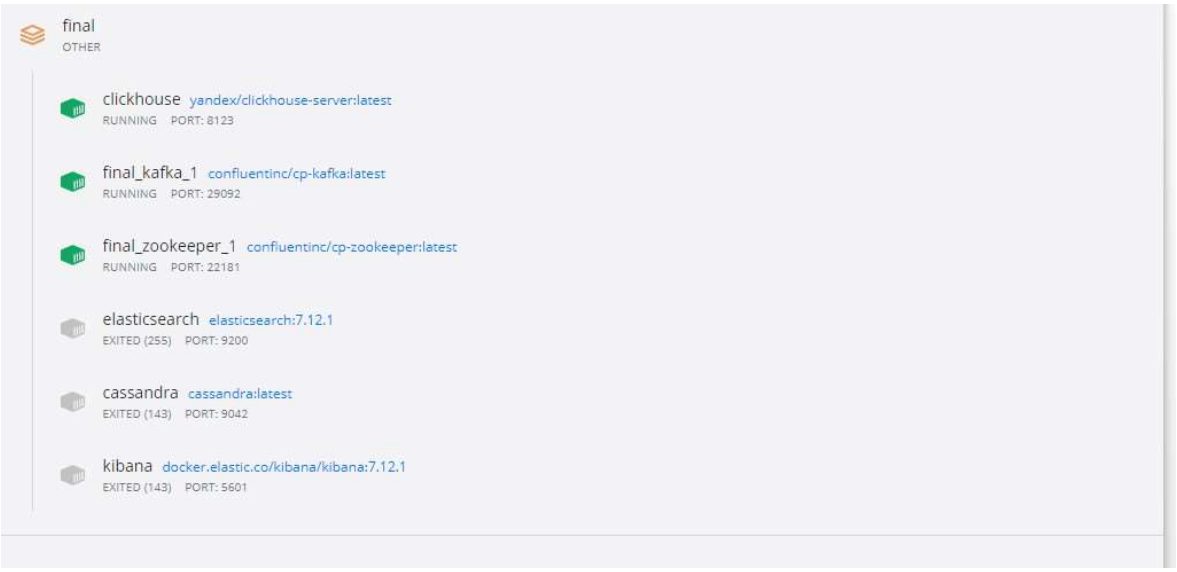
**توضیحات کلی** Apache Superset :

این اپاچی به موتور تولید داشبورد معروف است که می تواند با استفاده از ابزارهای قوی که در طراحی داشبورد دارد، اطلاعات بسیار خوبی به یک مدیر یک کامپانی یا کاربر بدهد.

**بخش اول: راه اندازی کلیک هوس در داکر**

جداول را برحسب خواسته ی سوالات در سوپرست طراحی می کنیم.طبق روش های قبل کلیک هوس را با داکر در پورت ۹۰۰۰ در همان فایل yml اصلی که بقیه کانتینرهارو بالا اوردیم، بالا می اوریم:

```
clickhouse:
  image: yandex/clickhouse-server:latest
  hostname: clickhouse
  container_name: clickhouse
  ports:
    - 8123:8123
    - 9000:9000
  ulimits:
    nofile:
      soft: 262144
      hard: 262144
  environment:
    #CLICKHOUSE_DB: Big_analytics__
    CLICKHOUSE_USER: admin
    CLICKHOUSE_PASSWORD: admin
  # depends_on:
  #   - "zookeeper"
```




## بخش دوم: کدنویسی در پایتون برای طراحی جدول در کلیک هوس

کافکا دو تابع به اسم KafkaProducer و KafkaConsumer دارد که در این گام ابتدا تنظیمات producer و consumer را در این توابع ست می کنیم. ما در این بخش ابتدا نقش consumer را داریم که در تابع KafkaConsumer باید از دیتای بخش ردیس در گام قبل که با اسم Analytics به این بخش توسط کافکا منتقل شد، استفاده کنیم.

```
from kafka import KafkaConsumer, KafkaProducer
from datetime import datetime, timezone
from clickhouse_driver import Client
from json import loads, dumps
from datetime import date

########################## Define Kafka clients ##########################

# Produce persistent data
producer = KafkaProducer(bootstrap_servers=['localhost:29092'],
                         value_serializer=lambda x:
                         dumps(x).encode('utf-8'),
                         api_version=(0,10))


consumer = KafkaConsumer(
    'Analytics',
    bootstrap_servers=['localhost:29092'],
    auto_offset_reset= 'earliest', # 'earliest', # Start from last consumed, #'latest' st
    enable_auto_commit=True,
    auto_commit_interval_ms = 1000,
    group_id='twitter',
    value_deserializer=lambda x: loads(x.decode('utf-8')),
    api_version=(0,10))
```

در ادامه باید اتصال به کلیک هوس را در پورت ۹۰۰۰ در پایتون برقرار کنیم تا جداول و دیتابیسی که می سازیم در کلیک هوس ذخیره شود تا در سوپرست از ان استفاده کنیم:

```
########################## clickhouse connection ##########################

client = Client(host='localhost',port=9000,user='admin',password='admin')
```

بخش سوم: ساخت دیتابیس و جدوال در کلیک هوس

حال گام اصلی در کلیک هوس شروع می شود ابتدا از تابع excute در کلیک هوس استفاده می کنیم تا کوئری های خود را ست کنیم.با استفاده از CREATE DATABASE IF NOT EXISTS (DATABASE_NAME) نام دیتابیس را ست می کنیم سپس با استفاده از دستور CREATE TABLE IF NOT EXISTS (DATABASE_NAME.TABLENAME) جداول را می سازیم:

```
######################## create database & tables ##########################
client.execute('CREATE DATABASE IF NOT EXISTS Big_analytics')
```

برای این دیتابیس ۴ تا جدول طبق گام سوپرست می سازیم:

### جدول اول:: جدول کلی توییت

فیلدهای مهم را طبق خواسته ی خود می سازیم و تایپش را ست می کنیم.نکته ی مهم در طراحی جدول پارتیشن کی و کلاستر هستند که با دستور ENGINE = MergeTree partition by انجام می دهیم و کلاستر کی را با دستور orderby در ادامه ی این دستور می گذاریم.طبق خواسته ی سوال پارتیشن کی را روز قرار می دهیم اما باید به این نکته توجه کرد که درست است دیتای ما برای چند روز است اما اگر گزارش ماهیانه بخواهیم ست کنیم باید فیلد دیگری را هم به این پارتیشن اضافه کنیم چون ممکن است ۲ ماه، یک روز یکسان داشته باشد.

```
##___ tweets table « partition keys :(year,month,day) , sorted keys: (hour,minute,second,id) »
client.execute("""CREATE TABLE IF NOT EXISTS Big_analytics.tweets(
    year Int32 , month Int32  , day Int32,
    hour Int32 , minute Int32 , second Int32,
    Tid_str String,
    hashtags Array(String),
    key_words Array(String)
) ENGINE = MergeTree partition by (year,month,day) order by (hour,minute,second,Tid_str);
""")
```

### جدول دوم و سوم و چهارم::

جدول هشتگ ها، جدول کیبورد و جدول کاربران(در کاربران ایتم هایی مثل نام کاربر و ای دی هم دریافت می کنیم)

```
## ___ hashtags table « partition keys :(year,month,day) , sorted keys: (hour,minute,second,id) »
client.execute("""CREATE TABLE IF NOT EXISTS Big_analytics.hashtags(
    year Int32 , month Int32  , day Int32,
    hour Int32 , minute Int32 , second Int32,
    Tid_str String,
    hashtags String
) ENGINE = MergeTree partition by (year,month,day) order by (hour,minute,second,Tid_str);
""")

## ___ keywords table « partition keys :(year,month,day) , sorted keys: (hour,minute,second,tweet_id) »
client.execute("""CREATE TABLE IF NOT EXISTS Big_analytics.keywords(
    year Int32 , month Int32  , day Int32,
    hour Int32 , minute Int32 , second Int32,
    Tid_str String,
    key_words String
) ENGINE = MergeTree partition by (year,month,day) order by (hour,minute,second,Tid_str);
""")

## ___ users table « partition keys :(year,month,day) , sorted keys: (hour,minute,second,user_id,tweet_
client.execute("""CREATE TABLE IF NOT EXISTS Big_analytics.users(
    year Int32 , month Int32  , day Int32,
    hour Int32 , minute Int32 , second Int32,
    Uname   String,
    location String,
    Uid_str  String,
    Tid_str  String
) ENGINE = MergeTree partition by (year,month,day) order by (hour,minute,second,Uid_str,Tid_str);
""")
```

## بخش چهارم: پیش پردازش فیلدها

ابتدا با استفاده از حلقه ی for توییت هایی که به صورت انلاین از گام statistics (در بالا با نام Analyticsدر consumer ست شده است)می ایند را دریافت می کنیم و در tweet قرار می دهیم. این توییت ها فیلدهای زیادی دارند که بهتر است فقط فیلدهایی گرفته شود که در این مرحله به ان ها نیاز داریم، پس از یک حلقه ی for دیگر استفاده می کنیم و فیلدهای (created_at(زمان به عدد و تاریخ به حروف)، keywords_k (کلمات کلیدی مثل کووید و...که در گام قبل گرفتیم و در این فیلد قرار دارد)، hashtags_k (هشتگ هایی که خودمان در گام قبل جدا کردیم و در این فیلد قرار دادیم اما یک سری فیلد دیگر مثل روز و ساعت هم به این فیلدها قبل از ارسال به کلیک هوس اضافه می کنیم که برای اینکار پردازشی روی فیلد created_at انجام می دهیم و سال و ماه و روز و ساعت و دقیقه و ثانیه را به به فیلد های بالا اضافه می کنیم. ( از تابع datetime برای این جداسازی استفاده

می کنیم) در ادامه فیلد idتوییت ها و id کاربران را برای جدول کاربران دریافت می کنیم و یک پیش پردازشی هم بر روی نام کاربران و مکان کاربران انجام می دهیم به این صورت که بعضی کاربران در بین نام هاشان یک سری punctuations وجود داشت که کلیک هوس نمی تواند اینها را بخواند و باید حذف شوند که با یک حلقه ی for دیگر اینکار را می کنیم.این حلقه ی for ادامه دارد یعنی در داخل این حلقه باید ولیو جدوال را insert کنیم که در بخش بعد به ان می پردازیم.

```
######################## stream twits to superset channel ########################

import pprint

for message in consumer:

    tweet               = message.value
    dtime               = tweet['created_at']
    new_datetime        = datetime.strftime(datetime.strptime(dtime,'%a %b %d %H:%M:%S +0000 %Y'))
    date ,time          = new_datetime.split(' ')
    year,month,day      = [ int(x) for x in date.split('-')]
    hour,minute,second  = [ int(x) for x in time.split(':')]
    Tid_str             = tweet['id_str']
    Uid_str             = tweet['user']['id_str']
    Uname               = str(tweet['user']['name'])
    location            = str(tweet['user']['location'])
    hashtags_k          = tweet['hashtags_k']
    keywords_k          = tweet['keywords_k']
    punclist            = ['/','\\','\'','\"']

    ##__ remove punctuations from strings_____!
    for punc in punclist:
        Uname    = Uname.replace(punc,'')
        location = location.replace(punc,'')
    ##_____________________________________________!
```

## بخش پنجم: ادامه ی حلقه ی for و insert کردن مقادیر جداول

در ادامه ی حلقه یfor بالا باید مقادیر ۴ جدول ایجاد شده را ست کنیم که برای اینکار از دستور INSERT INTO Big_analytics.table name VALUES استفاده می کنیم. فقط به این نکته باید توجه کنیم که کلیک هوس استرینگ خالی متوجه نمی شود یعنی فرمت خاصی از استرینگ باید نوشته شود که به این صورت است، '{tag}' است. برای اینکه هشتگ ها و کیبوردها در یک لیست قرار دارند و برای جداسازی انها برای هرکدام در ولیو دادن مقادیرش از یک حلقه ی for استفاده می کنیم و دقیقا استرینگ ها را به همین فرمت بالا ست می کنیم. حلقه ی for اصلی از بخش قبل،کلا درحال اجراست و توییت ها به صورت انلاین از توییتر توسط کافکا وارد پیش پردازش و از پیش پردازش توسط کافکا به persistence و از انجا توسط کافکا به گامhistory و از انجا به گام statistic و از انجا توسط کافکا به Analytics( consumer ست شده ما) می ایند و تا زمانی که ما قطع نکنیم، دیتا درحال گرفتن است و دسته ای از دیتاها در کلیک هوس به طور انلاین ذخیره می شوند.

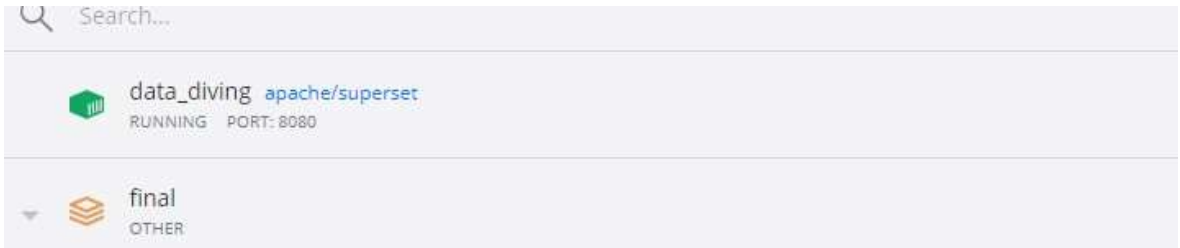
```
##_____________________________________________!

#_____________________________________________________________!
##_____________________ insert to tweets table _____________
client.execute(f"""INSERT INTO Big_analytics.tweets VALUES
                   ({year},{month},{day},{hour},{minute},{second},'{Tid_str}',{hashtags_k},{keywords_k}
                    )""")

##_____________________ insert to hashtags table _________
for tag in hashtags_k:
    client.execute(f"""INSERT INTO Big_analytics.hashtags VALUES
                    ({year},{month},{day},{hour},{minute},{second},'{Tid_str}','{tag}')""")

##_____________________ insert to keywords table _________
for word in keywords_k:
    client.execute(f"""INSERT INTO Big_analytics.keywords VALUES
                      ({year},{month},{day},{hour},{minute},{second},{Tid_str},'{word}')""")

print("tweet id: "+Tid_str+" added to clickhouse")
#_____________________________________________________________!

##_____________________ insert to users table _____________
client.execute(f"""INSERT INTO Big_analytics.users VALUES
                   ({year},{month},{day},{hour},{minute},{second},'{Uname}','{location}','{Uid_str}','{

print("user : "+Uname+" added to clickhouse")
#_____________________________________________________________!
```

```
tweet id: 1417400237837148171 added to clickhouse
user : ثکاء added to clickhouse
tweet id: 1417400238227218443 added to clickhouse
user : فاطمه حسنی پور added to clickhouse
tweet id: 1417400238227218442 added to clickhouse
user : ❄فاطمه❄ added to clickhouse
tweet id: 1417400238197837824 added to clickhouse
user : Fateme.. added to clickhouse
tweet id: 1417400238306910210 added to clickhouse
user : ✿ MoTiVationaL ✿ added to clickhouse
tweet id: 1417400238684397584 added to clickhouse
user : ماتیا. added to clickhouse
tweet id: 1417400238692786354 added to clickhouse
user : ㄹㅎ آر إچ added to clickhouse
tweet id: 1417400238822809646 added to clickhouse
user : علیرضا_کاف added to clickhouse
tweet id: 1417400238961164294 added to clickhouse
```

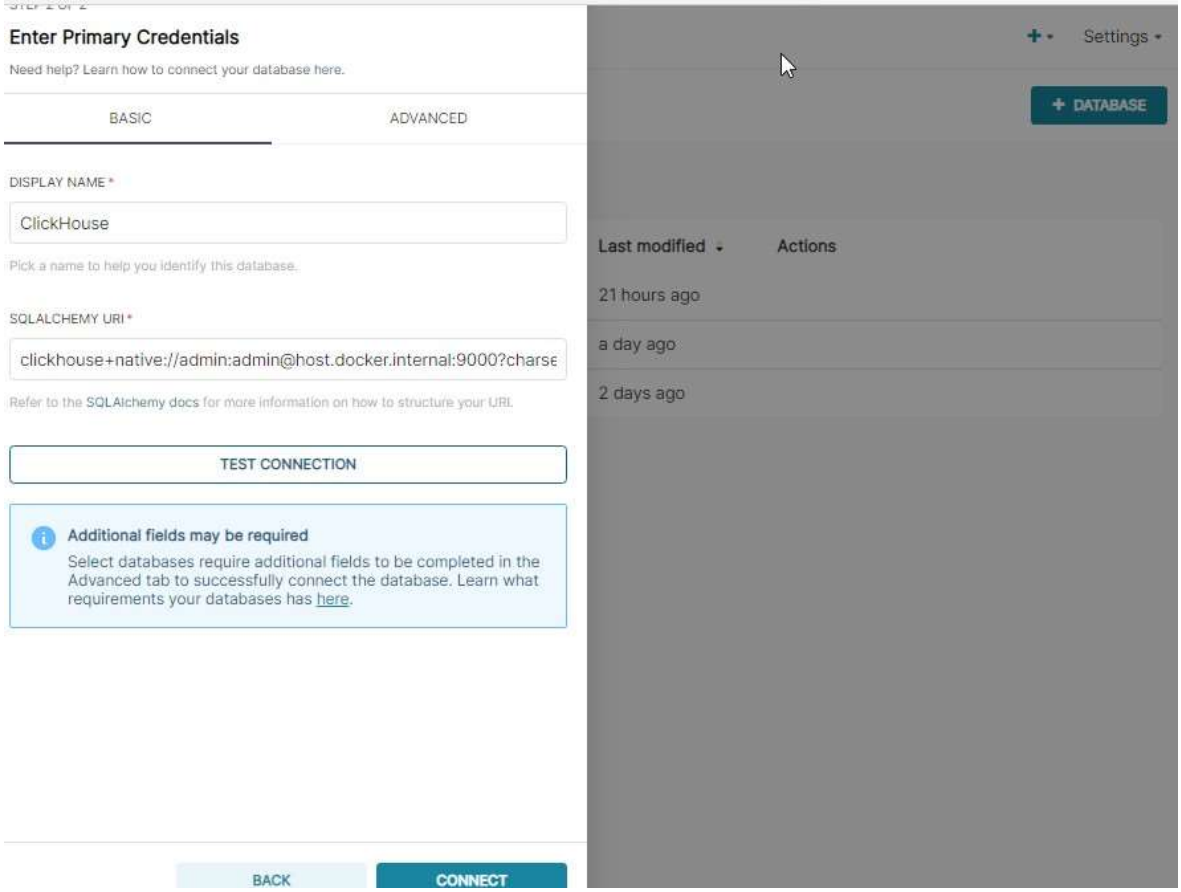
## بخش ششم: راه اندازی سوپرست در داکر و متصل کردن کلیک هوس به سوپرست

در این بخش ابتدا تنظیمات مربوط به سوپرست را در داکر انجام می دهیم. باید مراحل زیر را در cmd به ترتیب انجام دهیم.

```
WE FOLLOW THE STEPS IN BELOW LINK :

https://medium.com/geekculture/run-apache-superset-locally-in-10-minutes-30bc70ed808c

docker run -d -p 8080:8088 --name data_diving apache/superset

docker exec -it data_diving superset fab create-admin --username data_diving --firstname Admin --lastname
Admin --email admin@superset.com --password data_diving

docker exec -it data_diving superset db upgrade

docker exec -it data_diving superset load_examples

docker exec -it data_diving superset init

____________________________________________________________________________________

after a succesful instalation we need to  :

in the container we open container & :

pip install clickhouse-sqlalchemy

and we reset container but . . .
```




در پورت ۸۰۸۰ آپاچی سوپرست را بالا می اوریم و pass و user ست شده را به آن می دهیم و در نهایت وارد می شویم. در قسمت دیتابیس، دیتابیس ساخته شده در کلیک هوس را ست می کنیم و در نهایت در قسمت دیتاست، جدول ها را اضافه می کنیم. به این ترتیب اتصال برقرار شد( به نوع url دادن باید دقت کنیم) و می توانیم به سراغ ساختن داشبوردهای خواسته شده برویم:




## بخش هفتم: طراحی داشبورد در آپاچی سوپرست

ما در کل طبق خواسته ی سوال ۳ داشبورد طراحی می کنیم. در داشبورد اول، گزارشات کلی راجع به هشتگ و هشتگ خاص را می دهیم در داشبورد دوم، اطلاعات اماری را گزارش می دهیم و در داشبورد اخر، گزارشاتی راجع به کاربران را خواهیم داد.( به صورت جزیی و دقیق در روز ارائه انلاین توضیح خواهیم داد). ابتدا هر گزارش را به همراه عکس در داشبورد توضیح می دهیم سپس یک نمای کلی از داشبورد را نشان می دهیم.نکته ی دیگر این است که این دیتا به صورت انلاین به از سمت کلیک هوس به آن اضافه می شود برای همین تعداد فیلدها به طور مرتب درحال تغییر و اضافه شدن است.(ست کردیم که هر چندثانیه طبق داده ی انلاین عمل کند). ما از ابزارهای مختلف برای گزارش استفاده می کنیم تا دید بهتری به کاربر یا مدیری و...که در حال دیدن داشبورد است، بدهیم.

:::::داشبورد اول:::::

## گزارش هشتگ ها :

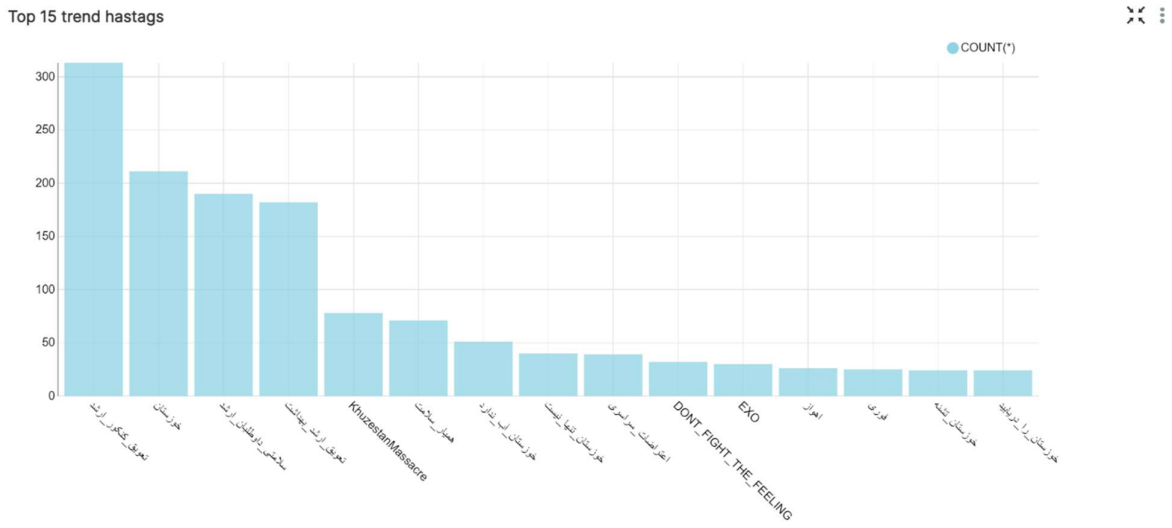
۱.در اولین گزارش، ابرکلمات را برای هشتگ ها نمایش می دهیم که می بینیم، طبق این چارت، تعویق_کنکور_ارشد که قرار است هفته ی اتی برگزار شود، بیشترین توییت را دارد چون شرایط کرونا در این روزها وخیم شده است.


۲.در دومین گزارش، از چارت pivot table استفاده می کنیم تا اطلاعات دقیقتری از ابرکلمات داشته باشیم. چون ما صرفا در بالا دیدیم که پررنگ تر است ولی تعداد را نمی دانیم که با استفاده از جدول می بینیم که ۳۱۳ هشتگ در مورد تعویق کنکور بود.

## Most Populated Hashtags

| | COUNT(*) |
|---|---|
| hashtags | |
| All | 2.58k |
| تعویق_کنکور_ارشد | 313 |
| خوزستان | 211 |
| سلامتی_داوطلبان_ارشد | 190 |
| تعویق_ارشد_بهداشت | 182 |
| KhuzestanMassacre | 78 |
| همیار_سلامت | 71 |
| خوزستان_آب_ندارد | 51 |
| خوزستان_تنها_نیست | 40 |
| اعتراضات_سراسری | 39 |
| DONT_FIGHT_THE_FEELING | 32 |
| EXO | 30 |
| اهواز | 26 |
| فوری | 25 |
| خوزستان_تشنه | 24 |
| خوزستان_را_دریابید | 24 |
| قیام_تشنگان | 23 |

۳.در سومین گزارش، از بار استفاده می کنیم تا ۱۵ هشتگ برتر را در نمودار نشان دهیم.



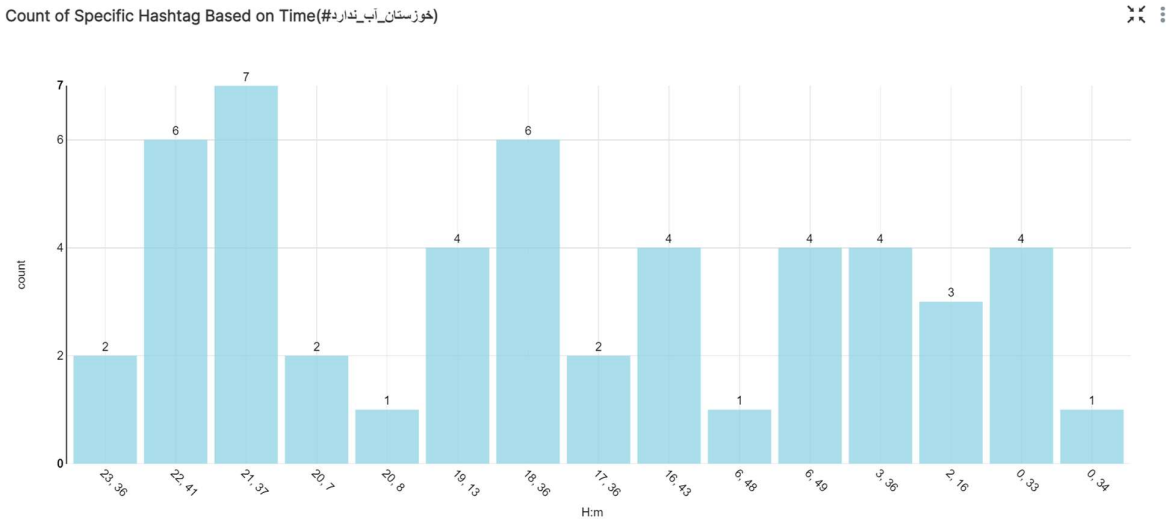

۴.در چهارمین گزارش، از چارت table استفاده می کنیم تا تعداد هشتگ ها را به صورت روزانه در ساعت ها مختلف بررسی کنیم که ببینیم کدام ساعت از روزها بیشترین هشتگ ها توییت می شود.

Count of Hastags daily(per h)

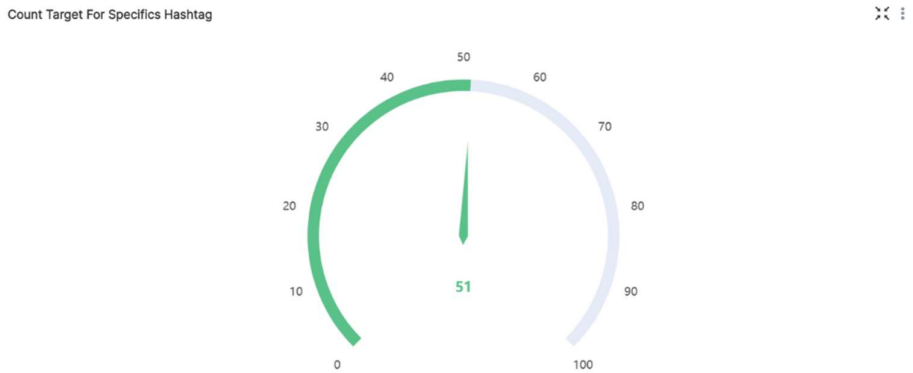
| hour | day | COUNT(*) |
|---|---|---|
| 0 | 20 | 43 |
| 22 | 19 | 37 |
| 3 | 20 | 32 |
| 6 | 20 | 29 |
| 19 | 19 | 23 |
| 17 | 19 | 19 |
| 20 | 19 | 16 |
| 18 | 19 | 16 |
| 21 | 19 | 15 |
| 23 | 19 | 9 |
| 2 | 20 | 7 |
| 16 | 19 | 7 |
| Totals | | 2.58 |

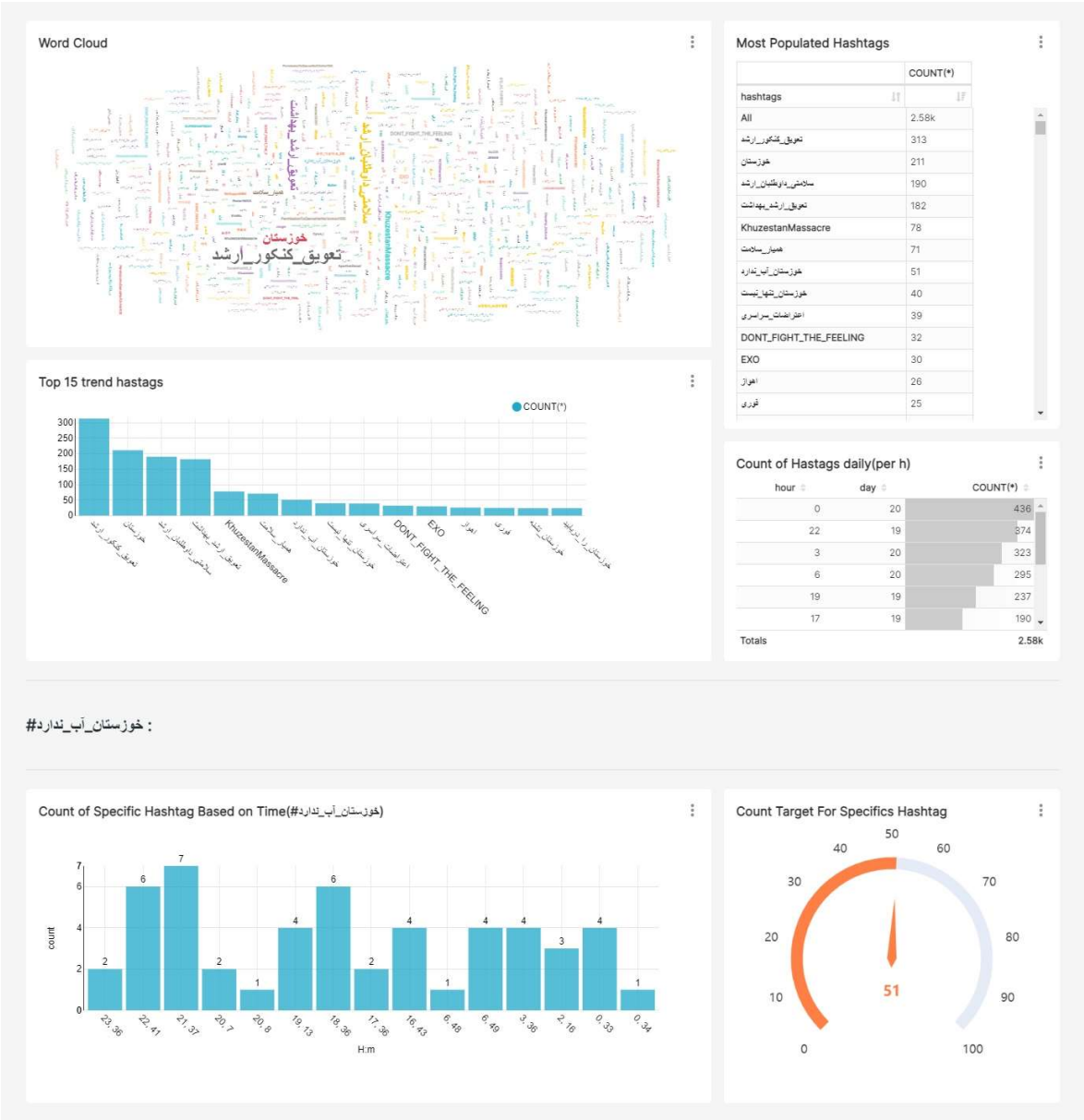
گزارش هشتگ های خاص(خوزستان_آب_ندارد)..... :

۱.در اولین گزارش، از چارت بار استفاده می کنیم تا تعداد این هشتگ را به صورت روزانه در ساعت ها مختلف بررسی کنیم که ببینیم در ساعت های مختلف از روز در ساعت های مختلف چه تعداد توییت راجع به این هشتگ زده می شود.




۲.در دومین گزارش، از چارت Gauge استفاده می کنیم تا ببینیم درتعداد کلی این هشتگ را تا بحال ببینیم. (این چارت برای هدف گذاری برای مثلا انجام یک کاری خوب است که این هشتگ نشان می دهد تعداد از ۵۰ هم بیشتر شده است و مثلا این هدف است که اگر تعداد در چند روز از ۴۰۰ بیشتر رفت این موضوع، موضوع مهمی است)





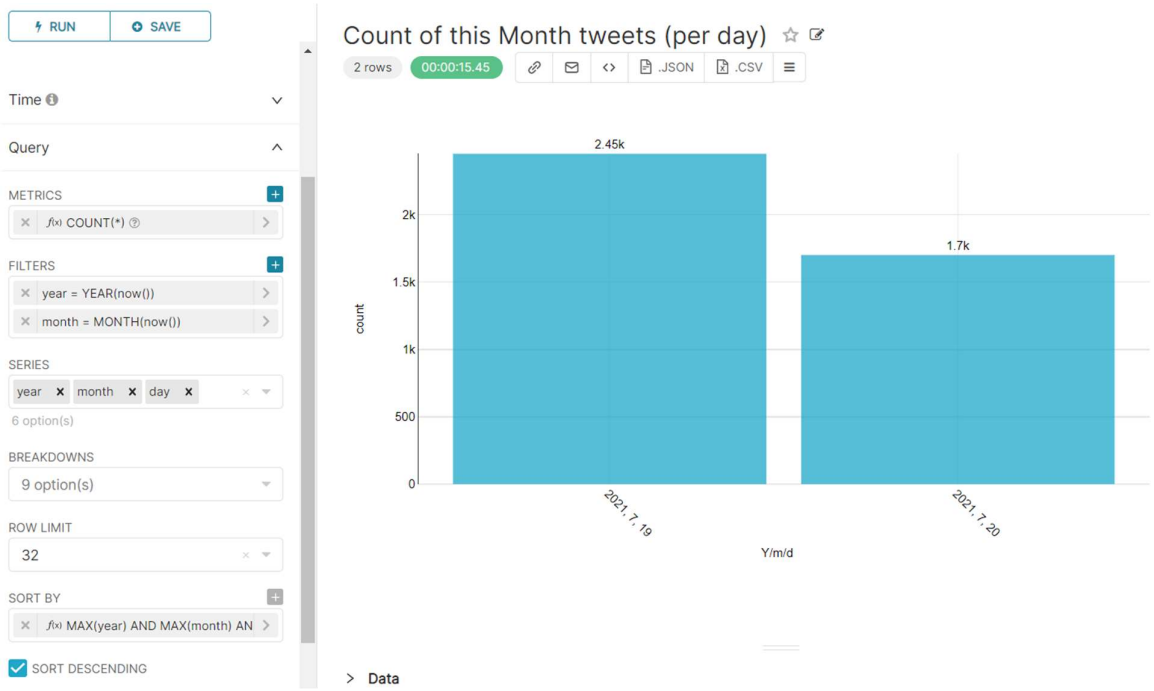

گزارش عمومی سامانه (از جدول توییت استفاده می کنیم):

۱.در اولین گزارش، از Big Number چارت استفاده می کنیم و تعداد کل توییت را برای ماه اخیر، روز اخیر و ساعت اخیر گزارش کنیم


۲.در دومین گزارش، از pivot table استفاده می کنیم و تعداد کل توییت را برای ماه اخیر بررسی می کنیم. و سپس بار پلات انرا برای درک بهتر در گزارش نمایش می دهیم

Count of this Month tweets (per day)

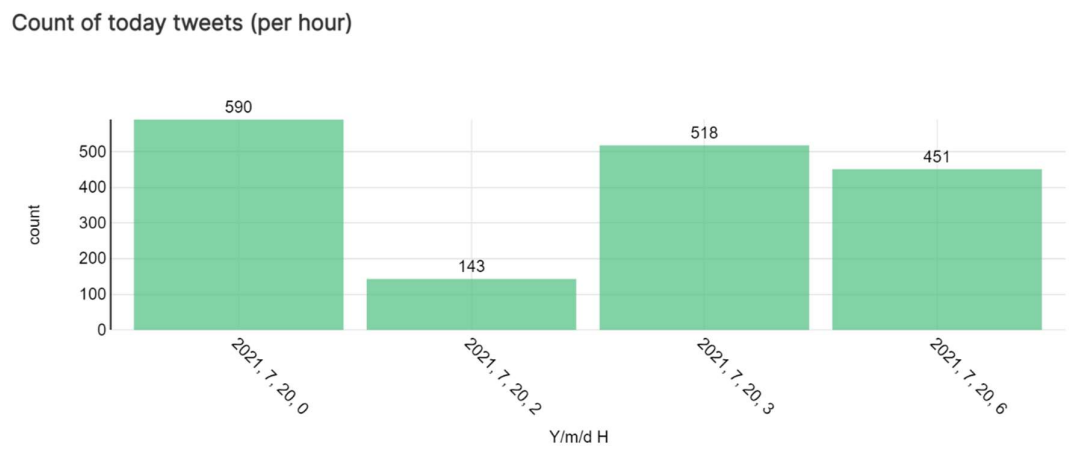
| | | | COUNT(*) |
|---|---|---|---|
| year | month | day | |
| 2021 | 7 | 19 | 2.45k |
| | | 20 | 1.7k |
| All | | | 4.16k |



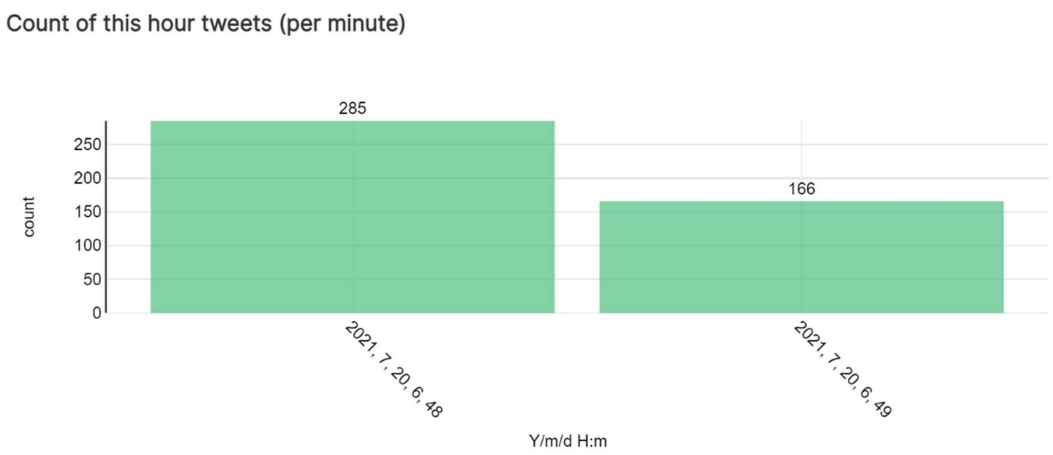

۳.در سومین گزارش، از pivot table استفاده می کنیم و تعداد کل توییت را برای روز اخیر طبق ساعت بررسی می کنیم و سپس بار پلات انرا برای درک بهتر نمایش می دهیم

Count of today tweets (per hour)

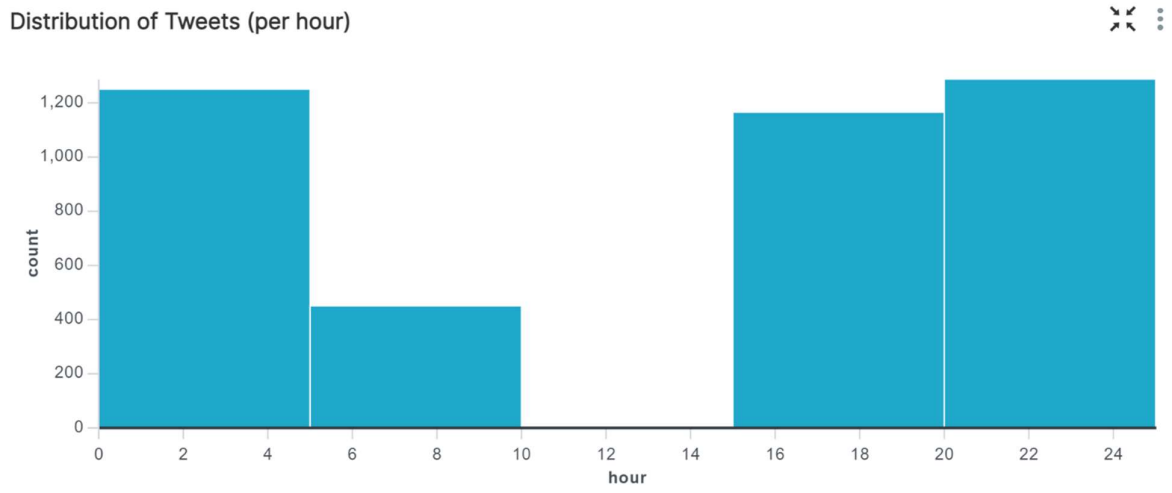
| | | | | COUNT(*) |
|---|---|---|---|---|
| year | month | day | hour | |
| 2021 | 7 | 20 | 0 | 590 |
| | | | 2 | 143 |
| | | | 3 | 518 |
| | | | 6 | 451 |
| All | | | | 590 |



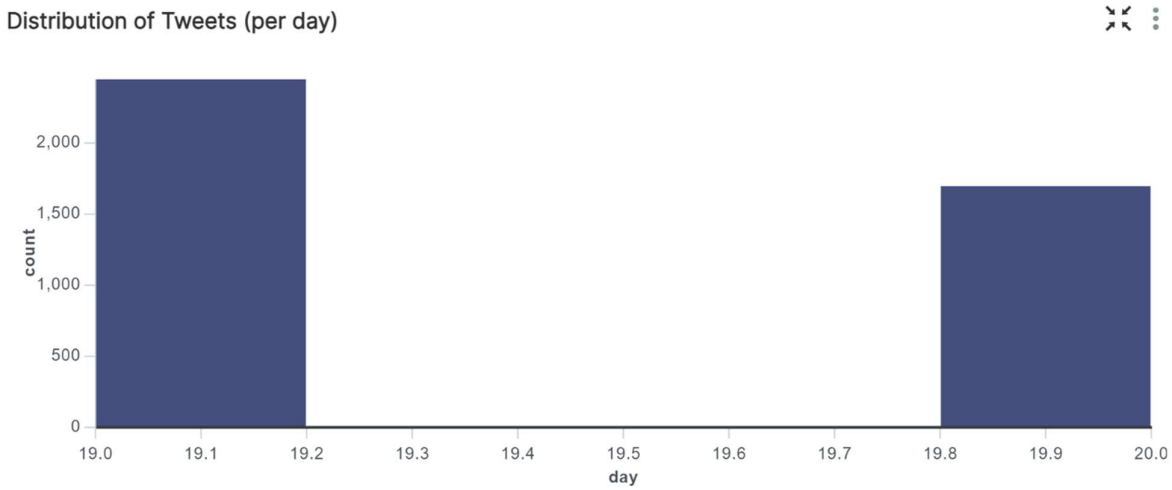

۴.در چهارمین گزارش، از بار پلات استفاده می کنیم و تعداد توییت ها را بر حسب ۱ دقیقه اخیربرای روزاخیر نشان دهیم.




۵.در پنجمین گزارش، از هسیتوگرام ستفاده می کنیم و تعداد توییت ها را بر حسب بازه ی ساعت (۲ ساعت، ۲ ساعت) برای روزاخیر نشان دهیم.( گپ خالی در هیستوگرام به این دلیل است که در این بازه، اطلاعات از سمت توییتر گرفته نشده است)




۶.در ششمین گزارش، از هسیتوگرام ستفاده می کنیم و تعداد توییت ها را بر حسب بازه ی روز برای روزاخیر نشان دهیم.( گپ خالی در هیستوگرام به این دلیل است که در این بازه، اطلاعات از سمت توییتر گرفته نشده است)



Count ALL

4.16k

Last day Tweet Count

4.16k

Last Hour Tweets Count

1.7k

Count of this Year tweets (per month)

| | | COUNT(*) |
|---|---|---|
| year | month | |
| 2021 | 7 | 4.16k |
| All | | 4.16k |

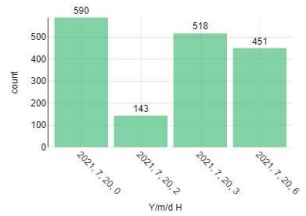
Count of this Month tweets (per day)

| | | | COUNT(*) |
|---|---|---|---|
| year | month | day | |
| 2021 | 7 | 19 | 2.45k |
| | | 20 | 1.7k |
| All | | | 4.16k |

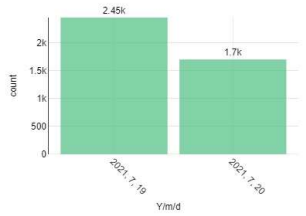
Count of today tweets (per hour)

| | | | | COUNT(*) |
|---|---|---|---|---|
| year | month | day | hour | |
| 2021 | 7 | 20 | 0 | 590 |
| | | | 2 | 143 |
| | | | 3 | 518 |
| | | | 6 | 451 |
| All | | | | 590 |

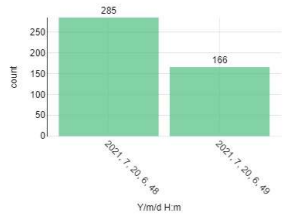
Count of today tweets (per hour)




Count of this Month tweets (per day)

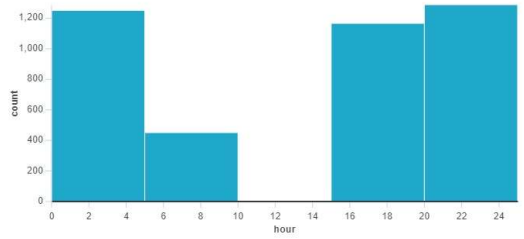



Count of this hour tweets (per minute)

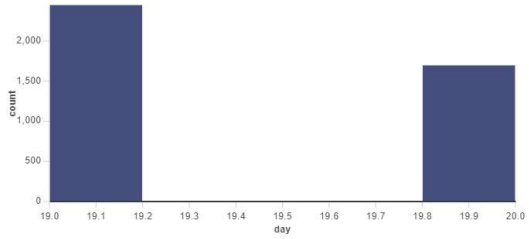



Globally :

Distribution of Tweets (per hour)



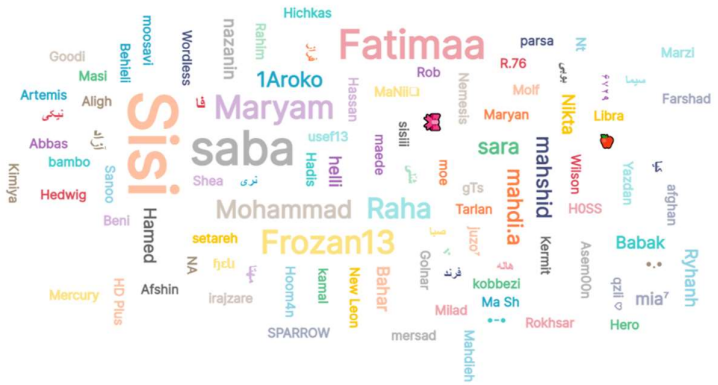

Distribution of Tweets (per day)



گزارش مربوط به کاربران (از جدول کاربران استفاده می کنیم)::

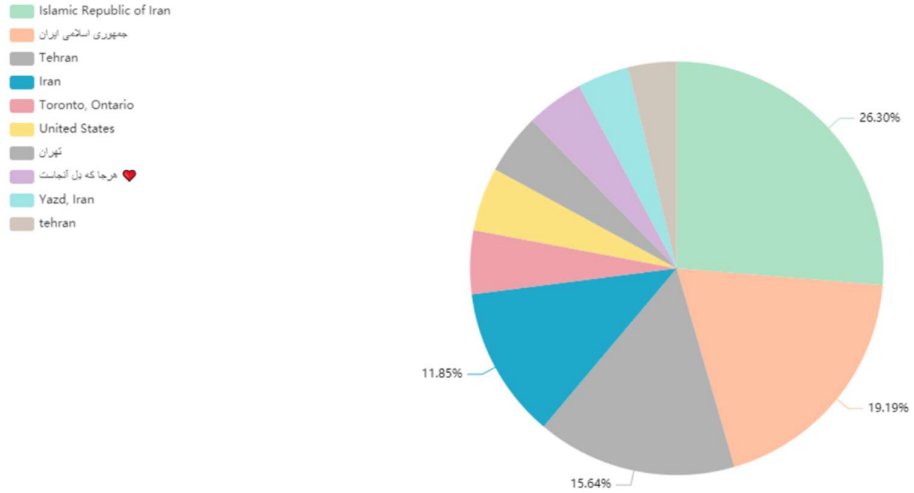
۱. در اولین گزارش، از چارت ابرکلمات استفاده می کنیم تا ۱۰۰ کاربر فعال نمایش دهیم که بینیم، طبق این چارت، کدام ۱۰۰ کاربر، بیشترین توییت را زده اند. این چارت فقط یک دید کلی به ما می دهد.

Top 100 Active Users

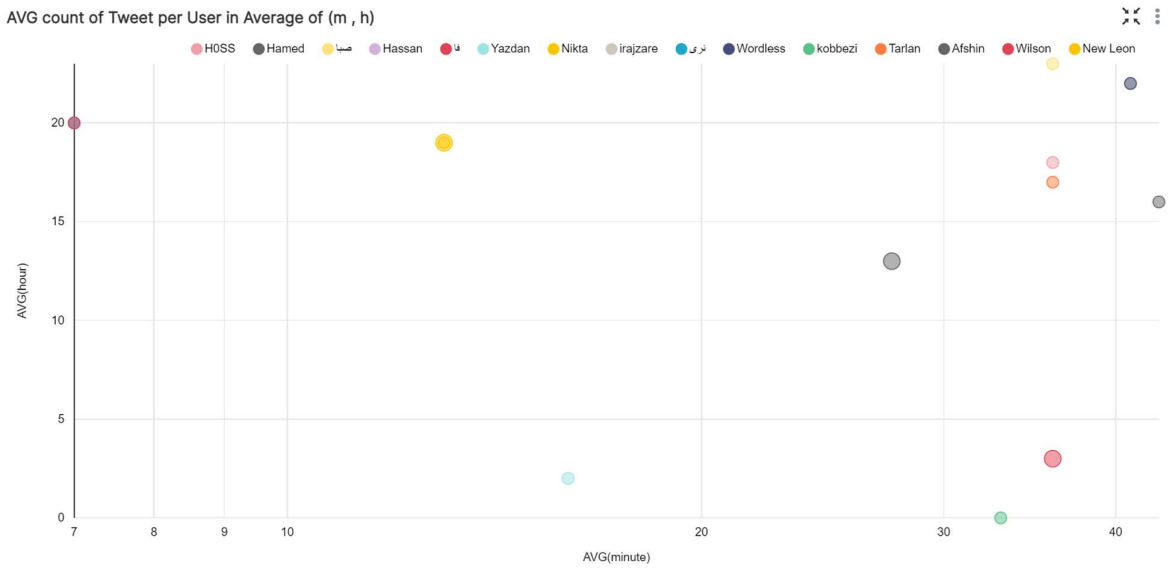



۲. در دومین گزارش، از pie چارت استفاده می کنیم تا بیشترین location مکانی ، طبق این چارت، مشاهده کنیم که می بینیم مثلا کسانی که از تهران توییت زدند، ۱۵.۶۴ درصد است.معمولا مردم location واقعی خودشان را بروز نمی دهند که احتمالا به دلایل امنیتی است.

Most used Location in User profile with persian tweets



۳. در سومین گزارش، bubble چارت استفاده می کنیم تا متوسط توییت را برای هر کاربر به طور متوسط در ساعت و دقیقه گزارش کنیم.




۴. در چهارمین گزارش، table چارت استفاده می کنیم تا تعداد توییت های هر کاربر را نمایش دهیم.

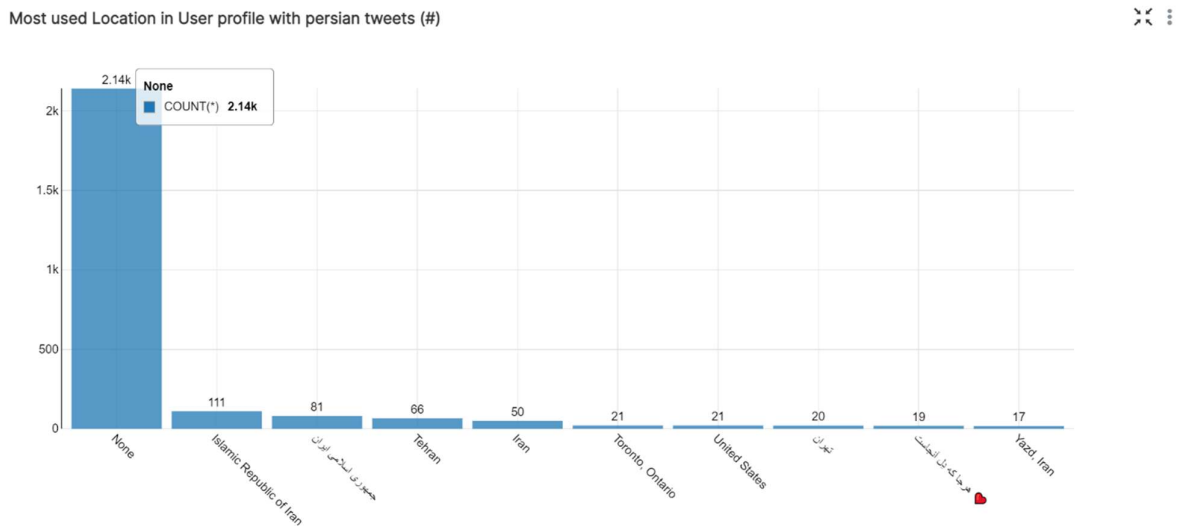
Count of Tweets per User

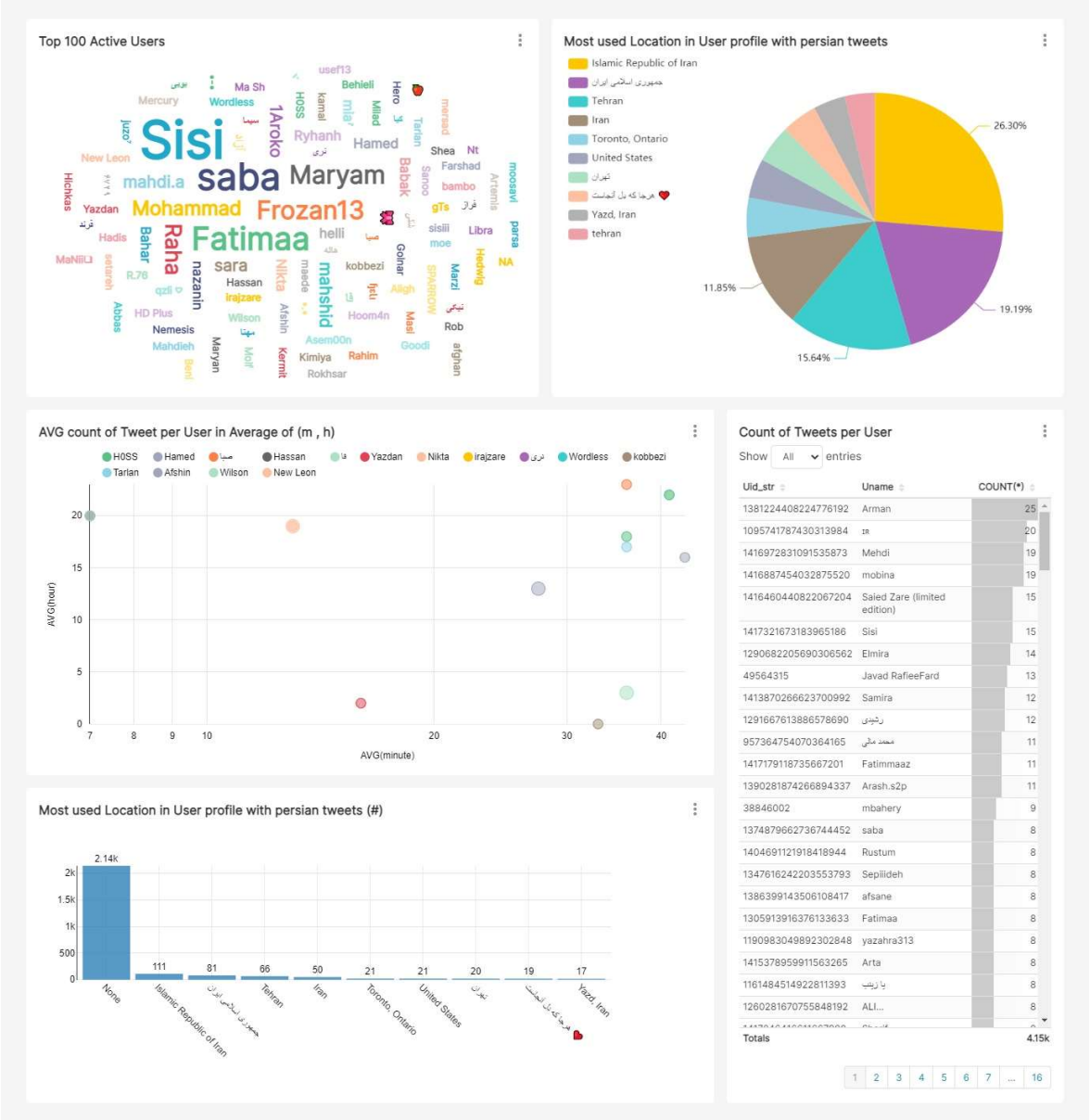
Show 200 entries

| Uid_str | Uname | COUNT(*) |
|---|---|---|
| 1381224408224776192 | Arman | 25 |
| 1095741787430313984 | IR | 20 |
| 1416972831091535873 | Mehdi | 19 |
| 1416887454032875520 | mobina | 19 |
| 1416460440822067204 | Saied Zare (limited edition) | 15 |
| 1417321673183965186 | Sisi | 15 |
| 1290682205690306562 | Elmira | 14 |
| 49564315 | Javad RafieeFard | 13 |
| 1413870266623700992 | Samira | 12 |
| 1291667613886578690 | رشیدی | 12 |
| 957364754070364165 | محمد مالی | 11 |
| 1417179118735667201 | Fatimmaaz | 11 |
| 1390281874266894337 | Arash.s2p | 11 |
| 38846002 | mbahery | 9 |
| Totals | | 4.1 |

1 2 3 4 5 6 7 ... 16

۵.در پنجمین گزارش، از بار استفاده می کنیم تا در یک شمای بهتری ببینیم چقدر برای مردم مهم است که اوکیشن واقعی خود را نشان دهند که می بینیم بیشترین تعداد توییت ها لوکیشن خود را ست نکردند.






پایان بخش پنجم

## بخش امتیازی: ساخت یک مدل پیشبینی کننده با اسپارک

آدرس گیتهاب: https://github.com/arminayat/bigdata-finalproject/tree/spark

### تنظیمات داکر

برای این بخش از pyspark-notebook استفاده کرده ام که روی پورت ۸۸۸۸ کار میکند و یک توکن ایجاد میکند که با آن میتوان به jupyter notebook متصل شد و روی محیط گرافیکی آن کد

```
spark:
  image: jupyter/pyspark-notebook
  container_name: spark
  ports:
    - 8888:8888
  volumes:
    - ./docker_data/spark-data:/home/jovyan/work
```

زد.

در گزارش این سوال، جزئیات کد و syntax توضیح داده نمیشود و برای اطلاعات بیشتر، میتوانید notebook پایتون این سوال که با markdown تقسیم بندی شده را مشاهده کنید.
(آدرس فایل ./docker_data/spark-data/spark.ipynb)

### راه اندازی کاساندرا

جدول Cassandra استخراج شده برای هر دو گام، Hashtags است که در زیر بخشی از آن مشاهده میشود. این جدول در ابتدا تبدیل به یک دیتافریم اسپارک میشود.

| | hashtag | year | month | day | hour | minute | second | id |
|---|---|---|---|---|---|---|---|---|
| **0** | 디오 | 2021 | 7 | 17 | 20 | 51 | 46 | 1416500900181458951 |
| **1** | 디오 | 2021 | 7 | 17 | 20 | 51 | 49 | 1416500911506067460 |
| **2** | 디오 | 2021 | 7 | 17 | 20 | 51 | 56 | 1416500940270673934 |
| **3** | 디오 | 2021 | 7 | 17 | 21 | 18 | 47 | 1416507698871476224 |
| **4** | سلامتی_داوطلبان_ارشد | 2021 | 7 | 17 | 17 | 51 | 0 | 1416455409250996232 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **1069** | شاوور | 2021 | 7 | 17 | 21 | 18 | 48 | 1416507702260506625 |
| **1070** | تبریز | 2021 | 7 | 19 | 21 | 54 | 19 | 1417241415944450049 |
| **1071** | خدا | 2021 | 7 | 17 | 19 | 58 | 31 | 1416487498272985094 |
| **1072** | هواپیمای_اوکراینی | 2021 | 7 | 17 | 19 | 58 | 24 | 1416487468782608384 |
| **1073** | مسیح_پولینژاد | 2021 | 7 | 17 | 19 | 58 | 24 | 1416487469462196227 |

1074 rows × 8 columns

```
root
 |-- hashtag: string (nullable = true)
 |-- year: long (nullable = true)
 |-- month: long (nullable = true)
 |-- day: long (nullable = true)
 |-- hour: long (nullable = true)
 |-- minute: long (nullable = true)
 |-- second: long (nullable = true)
 |-- id: string (nullable = true)
```

## پیشبینی زمان ارسال پست بعدی

تغییرات و تطابقات داده شده در صورت سوال: به جای کانال، هشتگ استفاده میشود و به جای روز هفته و ساعت، از ساعت و دقیقه و ثانیه استفاده میشود. این تطبیق به دلیل کمبود داده و نمونه های گرفته شده از توییتر انجام میشود.

## Exploratory Data Analysis

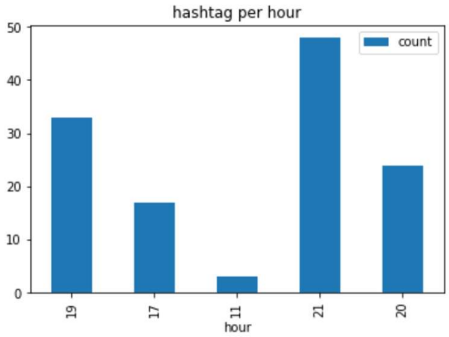
ابتدا طبق تصویر زیر (چپ) معروفترین هشتگهای ذخیره شده در اسپارک را Query کردیم و داده های پرتکرارترین هشتگ یعنی "خوزستان" را طبق تصویر زیر (وسط) برای انجام پیشبینی زمان بعدی پست نگه داشتیم. تعداد پستهای دریافت شده در هر ساعت که حاوی این هشتگ بوده اند در تصویر زیر

| | recurrence | hashtag |
|---|---|---|
| **0** | 125 | خوزستان |
| **1** | 34 | DONT_FIGHT_THE_FEELING |
| **2** | 32 | تعویق_کنکور_ارشد |
| **3** | 31 | خوزستان_آب_ندارد |
| **4** | 26 | EXO |
| **...** | ... | ... |
| **313** | 1 | اطلاعات_سپاه |
| **314** | 1 | _در_دانشگاه_آزاد_چه |
| **315** | 1 | ایرانسازپهلوی |
| **316** | 1 | خورستان |
| **317** | 1 | مبارکمون_باشه |

318 rows × 2 columns

| | id | hour | minute | second |
|---|---|---|---|---|
| **0** | 1 | 17 | 50 | 57 |
| **1** | 2 | 17 | 50 | 57 |
| **2** | 3 | 17 | 50 | 57 |
| **3** | 4 | 17 | 50 | 58 |
| **4** | 5 | 17 | 50 | 58 |
| **...** | ... | ... | ... | ... |
| **120** | 121 | 21 | 54 | 10 |
| **121** | 122 | 21 | 54 | 11 |
| **122** | 123 | 21 | 54 | 11 |
| **123** | 124 | 21 | 54 | 12 |
| **124** | 125 | 21 | 54 | 14 |

125 rows × 4 columns



(راست) قابل مشاهده است.

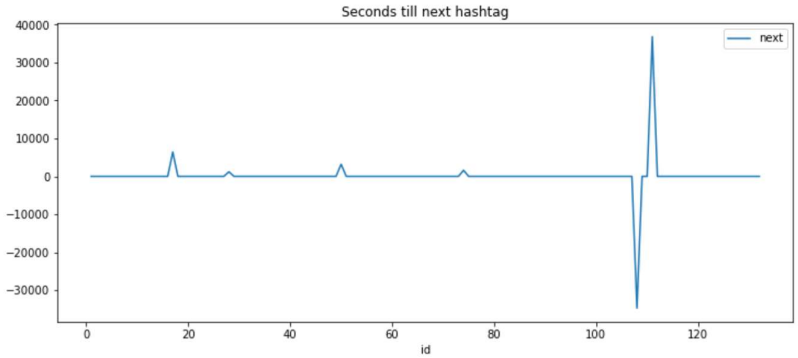
برای فرموله کردن مساله، برای هر پست، زمان روز به ثانیه را در یک ستون جدید به نام timestamp و اختلاف زمانی به ثانیه تا پست بعدی را نیز در ستون جدید next ذخیره کردیم.
دیتافریم ایجاد شده طبق تصویر مقابل است (دقت شود که به دلیل ذات ستون next، دیتافریم جدید یک سطر کمتر دارد).

| | id | hour | minute | second | timestamp | next |
|---|---|---|---|---|---|---|
| **0** | 1 | 17 | 50 | 57 | 64257 | 0 |
| **1** | 2 | 17 | 50 | 57 | 64257 | 0 |
| **2** | 3 | 17 | 50 | 57 | 64257 | 1 |
| **3** | 4 | 17 | 50 | 58 | 64258 | 0 |
| **4** | 5 | 17 | 50 | 58 | 64258 | 0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **119** | 120 | 21 | 54 | 10 | 78850 | 0 |
| **120** | 121 | 21 | 54 | 10 | 78850 | 1 |
| **121** | 122 | 21 | 54 | 11 | 78851 | 0 |
| **122** | 123 | 21 | 54 | 11 | 78851 | 1 |
| **123** | 124 | 21 | 54 | 12 | 78852 | 2 |

124 rows × 6 columns

## Outliers!!

به دلیل استخراج توییت در بازه های مقطع و متفاوت زمانی، استفاده از اختلاف زمانی، داده های پرت ایجاد میکند که در تصویر زیر قابل مشاهده است. مشاهده میشود که اکثر توییت ها اختلاف زمانی کم و نزدیک به ۰ دارند که مربوط به یک اجرا و session هستند، ولی آخرین توییت گرفته شده در یک session و اولین توییت گرفته شده در یک session دیگر دارای اختلاف بزرگی هستند (مثبت یا




منفی). میتوان با توجه به تعداد این outlier ها تعداد session های استخراج توییت را فهمید! ۶!

همچنین طبق جدول زیر مشاهده میشود که اطلاعات آماری ستون next مانند mean و STD بسیار تحت تاثیر این نقاط پرت قرار گرفته است.

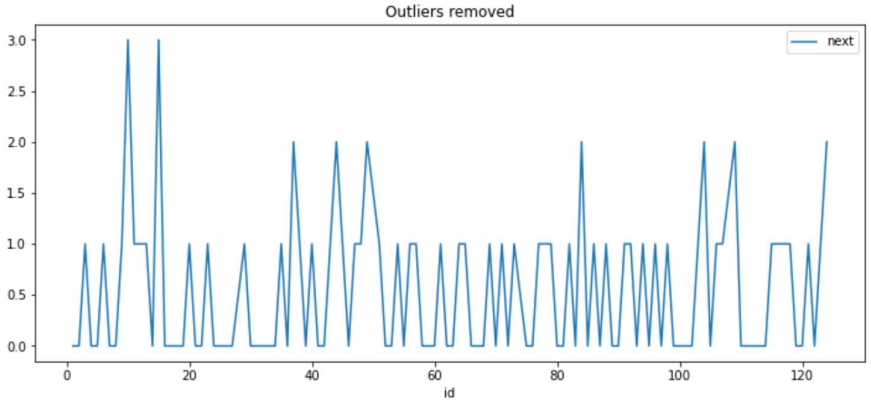
برای از بین بردن این نقاط از روش Median Absolute Deviation استفاده کرده ایم که طبق فرمول زیر، ابتدا فاصله تمام نقاط "next" از میانه شان را بدست آورده و سپس این اعداد را با میانه این اختلافات مقایسه کرده، اگر از یک threshold بیشتر باشد، آن نقطه Outlier محسوب شده و از دیتافریم حذف میشود.

$$\mathbf{MeanAbsoluteDifference_i = MAD_i = X_i - median(X)}$$

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **summary** | count | mean | stddev | min | max |
| **id** | 124 | 62.5 | 35.939764421413045 | 1 | 124 |
| **hour** | 124 | 19.483870967741936 | 1.8977259432207378 | 11 | 21 |
| **minute** | 124 | 42.016129032258064 | 15.732511969207717 | 18 | 58 |
| **second** | 124 | 34.645161290322584 | 18.561343494787696 | 0 | 59 |
| **timestamp** | 124 | 72697.54838709677 | 6546.391115715299 | 42040 | 78852 |
| **next** | 124 | 117.71774193548387 | 4608.382060155776 | -34697 | 36804 |

$$\mathbf{if\ (MAD_i - median(MAD) > Threshold)\ \rightarrow\ \ Outlier\ removal}$$

تصویر زیر مقادیر "next" بعد از حذف داده های پرت را نشان میدهد. اکثر توییت های بعدی در فاصله



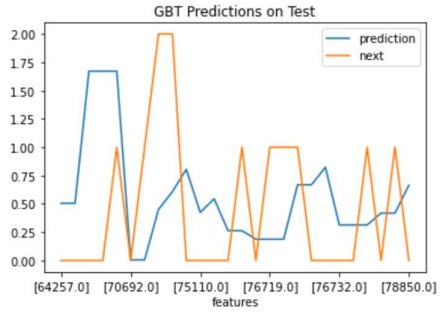

۰ و ۱ ثانیه از توییت فعلی قرار دارند.

## پیشبینی

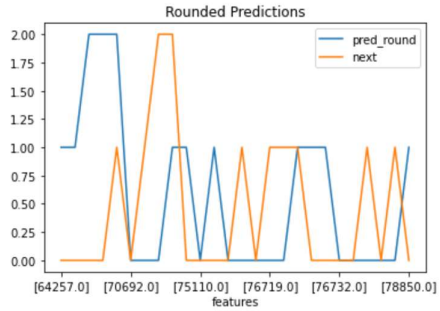
کل داده ها ۱۱۸ عدد هستند که ۹۲ تای آنها برای Train و ۲۶ تای آنها برای Test انتخاب شده اند.

برای پیشبینی از مدل GradientBoostedTree(GBT) regression model با تنظیمات پیش فرض انجام شده. نتایج روی داده های Test در دیتافریم مقابل قابل مشاهده است. "Prediction"، پیشبینی پیوسته و "P red_round" رند شده آن است.

| | prediction | pred_round | next | features |
|---|---|---|---|---|
| 0 | 0.505221 | 1.0 | 0 | [64257.0] |
| 1 | 0.505221 | 1.0 | 0 | [64258.0] |
| 2 | 1.671888 | 2.0 | 0 | [64259.0] |
| 3 | 1.671888 | 2.0 | 0 | [64259.0] |
| 4 | 1.671888 | 2.0 | 1 | [64264.0] |
| 5 | 0.005221 | 0.0 | 0 | [70692.0] |
| 6 | 0.005221 | 0.0 | 1 | [70692.0] |
| 7 | 0.450539 | 0.0 | 2 | [71905.0] |
| 8 | 0.607979 | 1.0 | 2 | [71910.0] |
| 9 | 0.804779 | 1.0 | 0 | [75108.0] |
| 10 | 0.424489 | 0.0 | 0 | [75110.0] |




در تصویر راست همان نتایج به صورت نمودار قابل مشاهده است. مدل به خوبی توانسته (با توجه به واریانس زیاد و تعداد نمونه کم) عمل پیشبینی را انجام دهد.




مقدار RMSE روی داده های تست برابر ۰.۸۲ است که با توجه به STD برابر ۰.۶۴ داده های تست مقدار قابل قبولیست.

## پیشبینی هشتگ های یک پست:

تغییرات و تطابقات داده شده در صورت سوال: به جای کلیدواژه، هشتگها با توجه به متن پیشبینی میشوند. برای سادگی، فقط پست هایی که ۱۰ هشتگ پرطرفدارتر را دارند را استفاده میکنیم و باقی پست ها را دور میریزیم.

### Exploratory Data Analysis

ابتدا ۱۰ پرطرفدارترین هشتگ و تعداد تکرار آنها طبق تصویر مقابل (چپ) استخراج شده سپس یک دیتافریم که هر سطر آن شامل یک ID و یکی از این هشتگ هاست طبق تصویر مقابل (وسط) ایجاد میشود. برخی از پست ها چندین هشتگ دارند و این مساله در این دیتافریم مشهود است (دایره قرمز). طبق صورت سوال باید مدل قادر باشد برای هر پست چند هشتگ در خروجی پیشنهاد دهد.

| | hashtag | recurrence |
|---|---|---|
| 0 | خوزستان | 133 |
| 1 | تعویق_کنکور_ارشد | 43 |
| 2 | DONT_FIGHT_THE_FEELING | 40 |
| 3 | سلامتی_داوطلبان_ارشد | 35 |
| 4 | خوزستان_آب_ندارد | 32 |
| 5 | EXO | 30 |
| 6 | باقرالعلوم | 26 |
| 7 | اعتراضات_سراسری | 26 |
| 8 | تعویق_ارشد_بهداشت | 26 |
| 9 | قیام_تشنگان | 21 |

| | hashtag | id |
|---|---|---|
| 0 | خوزستان | 1416455394214518787 |
| 1 | خوزستان | 1416455394659012610 |
| 2 | باقرالعلوم | 1416455396265500679 |
| 3 | خوزستان | 1416455397121085442 |
| 4 | اعتراضات_سراسری | 1416455398488485890 |
| ... | ... | ... |
| 407 | خوزستان_آب_ندارد | 1417241449893187586 |
| 408 | خوزستان | 1417241450438406149 |
| 409 | سلامتی_داوطلبان_ارشد | 1417241450761306120 |
| 410 | تعویق_ارشد_بهداشت | 1417241450761306120 |
| 411 | تعویق_کنکور_ارشد | 1417241450761306120 |

412 rows × 2 columns

برای این کار با عملیات Pivot، هشتگها را از حالت سطری به ستونی و گروهبندی شده روی ID درمیاوریم طبق تصویر زیر. این عملیات به نحوی مشابه با CountVectorizing است. از این هشتگ های ستونی به عنوان Label برای مدل پیشبینی استفاده خواهیم کرد. مشاهده میشود که تعداد نمونه

| | id | DONT_FIGHT_THE_FEELING | EXO | اعتراضات_سراسری | باقرالعلوم | تعویق_ارشد_بهداشت | تعویق_کنکور_ارشد | خوزستان | خوزستان_آب_ندارد | سلامتی_داوطلبان_ارشد | 엑소 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1416487448343875585 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1416455437369622531 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 2 | 1416482385781268481 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 1416507683461554176 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1416500934025244673 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 296 | 1416507735739355139 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 297 | 1416507686817046538 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 298 | 1416455433800151041 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 299 | 1416487487271313411 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 300 | 1416487444665544704 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

301 rows × 11 columns

ها از ۴۱۲ به ۳۰۱ عدد با ID های منحصر به فرد تقلیل می یابد

## اتصال به الاستیک سرچ

در این بخش، بعد از اتصال به الاستیک سرچ، متن تمام ID های موجود در دیتافریمم در اسپارک را از الاستیک سرچ استخراج کرده و آن را به دیتافریمم در اسپارک اضافه کردیم، و Duplicate های احتمالی را از دیتافریم پاک کردیم. اطلاعات استخراج شده از الاستیک سرچ طبق تصویر زیر به انتهای دیتافریم اضافه میشوند. توجه شود که این متن ها، همان متن های پردازش نشده استخراج شده از

| text_k | id | 성소 | سلامتی_داوطلبان_ارشد | خوزستان_آب_ندارد | خوزستان | تعویق_کنکور_ارشد | تعویق_ارشد_بهداشت |
|---|---|---|---|---|---|---|---|
| RT @ExoFvn_: میخوام تو تیک تاک پستش کنم اشکالی... | 1416482383772233729 | 0 | 0 | 0 | 0 | 0 | 0 |
| RT @ayyaaaar: دهها استاندار در سیستان بلوچستا... | 1416482384573173761 | 0 | 0 | 0 | 1 | 0 | 0 |
| RT @Arezou70m: خیانت یعنی از آبی که میتونی مشک... | 1416482385403785221 | 0 | 0 | 0 | 1 | 0 | 0 |

توییتر هستند.

## پیش پردازش متن

برای پیش پردازش متن از کتابخانه Spark NLP استفاده کردیم و عملیات های Tokenize و Normalize (Only keep persian words and digits) و Remove_Stopwords را در یک پایپلاین روی داده ها انجام دادیم. در نهایت، متنها به یک لیست از کلمات طبق تصویر مقابل تبدیل شدند.

| | words |
|---|---|
| 0 | [میخوام, تو, تیک, تاک, پستش, کنم, اشکالی, نداره؟] |
| 1 | ...دهها, استاندار, سیستان, بلوچستان, خوزستان, عو] |
| 2 | ...خیانت, یعنی, آبی, میتونی, مشکل, شریشو, حل ,70] |
| 3 | ...,ساله, رعایت, کردم, کرونا, نگرفتم, خدا, روشکر] |
| 4 | ...برای, یه, کصکش, بیست, چهارساله, قاتل, هشتگ, ز] |
| 5 | ...چقدر, خاطره, تلخ, مانده, ذهنت, باقرالعلوم, طی] |
| 6 | ...درود, خوزستان, حمایت, می, کنید, نگذاریم, قیام] |
| 7 | ...کاش, تو, تکبیرهای, نماز, مرگ, مسئولین, خوزست] |
| 8 | [جای, جهانآرا, خالی, ست, خوزستان] |
| 9 | ...,الحمدلله, آب, هور, حمیدیه, شادگان, رسید ,313] |

برای Embed کردن این کلمات در فضای برداری از آنجایی که در اسپارک، کتابخانه ای برای بدست آوردن Contexualized Embedding فارسی وجود ندارد (فقط BERT در SparkNLP موجود بود که حجم حافظه ای و پردازشی آن فراتر از سیستم من بود) از CountVectorizer استفاده کردیم تا به

| | features |
|---|---|
| 0 | (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, ... |
| 1 | (1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, ... |
| 2 | (1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... |
| 3 | (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, ... |
| 4 | (0.0, 0.0, 0.0, 2.0, 0.0, 1.0, 0.0, 0.0, 0.0, ... |
| 5 | (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... |
| 6 | (1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, ... |
| 7 | (1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, ... |
| 8 | (1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... |
| 9 | (1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... |

embedding متن توییت برسیم. این متن ها بعد از نگاشت به فضای برداری که طبق تصویر مقابل هستند، به عنوان Feature برای مدل استفاده خواهند شد.

## پیشبینی

۱۳۵ نمونه از داده ها برای Train و ۳۸ عدد برای Test استفاده شده اند.

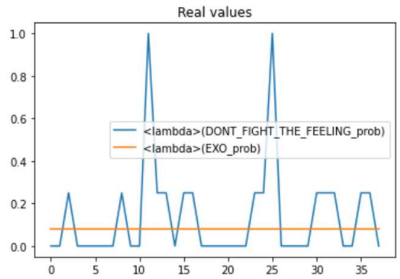
مدل استفاده شده برای پیشبینی هر کدام از ۱۰ هشتگ انتخاب شده یک DesicionTree Binary Classifier با ۲ لایه است. در واقع مدل نهایی ترکیب این ۱۰ مدل binary classifier است و در نهایت، با انتخاب یک threshold روی خروجی های تمام این مدل ها، یک لیست از هشتگ های پیشبینی شده برای هر پست بدست می آید. این لیست ممکن است خالی باشد، یک هشتگ داشته باشد، یا چند هشتگ داشته باشد. به دلیل سنگینی محاسباتی این مدل و استفاده از الاستیک سرچ، سیستم من در حین یادگیری این کلاسیفایر ها بعد از مدتی هنگ میکرد و در ادامه، خروجی مدل برای هشتگ های DON'T_FIGHT_THE_FEELING و EXO نشان داده میشود.

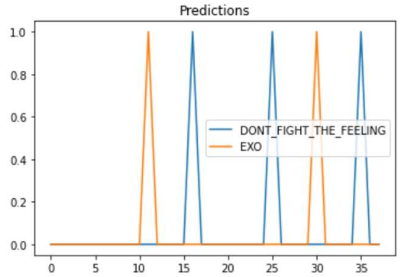
احتمال پیشبینی شده برای این ۲ هشتگ در جدول زیر (چپ) و مقدار واقعی این ۲ در جدول زیر (راست) قابل مشاهده است

| | id | <lambda>(DONT_FIGHT_THE_FEELING_prob) | <lambda>(EXO_prob) |
|---|---|---|---|
| 0 | 1416507680278130693 | 0.00 | 0.081481 |
| 1 | 1416507723357827079 | 0.00 | 0.081481 |
| 2 | 1416500926978957316 | 0.25 | 0.081481 |
| 3 | 1416507665094742020 | 0.00 | 0.081481 |
| 4 | 1416482393339224064 | 0.00 | 0.081481 |
| 5 | 1416482394354368514 | 0.00 | 0.081481 |
| 6 | 1416500902932922379 | 0.00 | 0.081481 |
| 7 | 1416500940258041860 | 0.00 | 0.081481 |
| 8 | 1416500934025244673 | 0.25 | 0.081481 |
| 9 | 1416507725626892289 | 0.00 | 0.081481 |
| 10 | 1416500911468339204 | 0.00 | 0.081481 |

| | id | DONT_FIGHT_THE_FEELING | EXO |
|---|---|---|---|
| 0 | 1416507680278130693 | 0 | 0 |
| 1 | 1416507723357827079 | 0 | 0 |
| 2 | 1416500926978957316 | 0 | 0 |
| 3 | 1416507665094742020 | 0 | 0 |
| 4 | 1416482393339224064 | 0 | 0 |
| 5 | 1416482394354368514 | 0 | 0 |
| 6 | 1416500902932922379 | 0 | 0 |
| 7 | 1416500940258041860 | 0 | 0 |
| 8 | 1416500934025244673 | 0 | 0 |
| 9 | 1416507725626892289 | 0 | 0 |
| 10 | 1416500911468339204 | 0 | 0 |

همین نتایج به صورت نمودار در تصویر مقابل نیز قابل مشاهده هستند. نمودار اول داده های واقعی و نمودار دوم داده های پیشبینی شده را نشان میدهد. مشاهده میشود که مدل توانسته برخی پیشبینی ها برای DON'T_FIGHT_THE_FEELING انجام دهد ولی نتوانسته الگویی در EXO پیدا کند.



این مدل در صورتی که برای تمام هشتگها train شود میتواند یک احتمال وجود هر هشتگی در یک پست را تخمین بزند.

همین نتایج به صورت نمودار در تصویر مقابل نیز قابل مشاهده هستند. نمودار اول داده های واقعی و نمودار دوم داده های پیشبینی شده را نشان میدهد. مشاهده میشود که مدل توانسته برخی پیشبینی ها برای DON'T_FIGHT_THE_FEELING انجام دهد ولی نتوانسته الگویی در EXO پیدا کند.

این مدل در صورتی که برای تمام هشتگها train شود میتواند یک احتمال وجود هر هشتگی در یک پست را تخمین بزند.

---

پایان بخش امتیازی