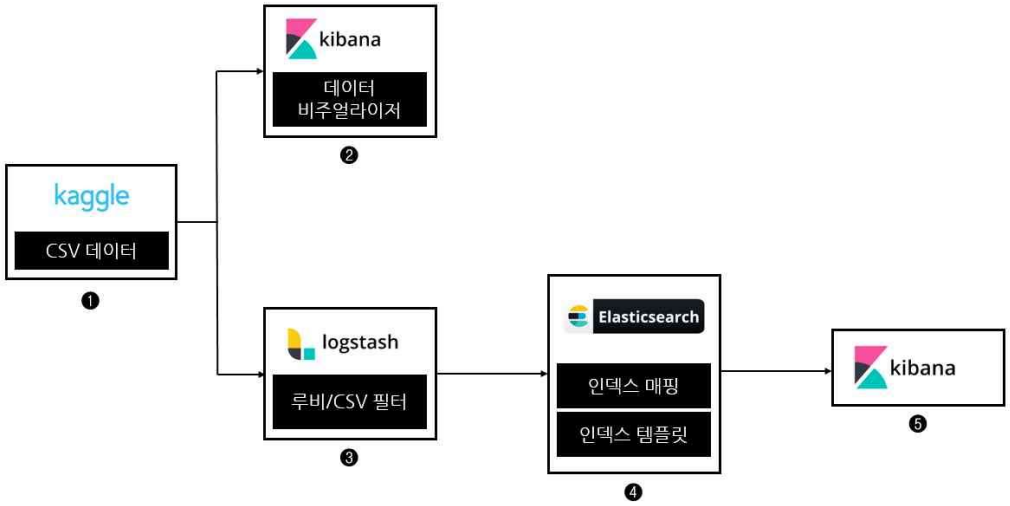
Ⅰ (실전활용) 캐글 CSV 파일을 활용한 인덱스 작성

엘라스틱서치를 이용해 분석을 하려면 가장 먼저 데이터가 필요하다. 데이터를 확보하기 위해서는 캐글(Kaggle)의 도움을 받아야 하는데, 캐글은 데이터 과학자들의 경진대회 플랫폼으로 기업이나 단체에서 해결하고 싶은 문제를 상금과 함께 등록하면 데이터 과학자들의 경쟁을 통해 더 좋은 모델이나 방법을 찾아주는 사이트이다.

이번에는 캐글에서 가져온 영화 관련 데이터를 키바나를 이용하여 손쉽게 엘라스틱서치에 저장하는 방법과 로그스태시를 거치면서 데이터를 정제한 다음 인덱스 매핑에 맞춰 엘라스틱서치에 인덱스를 생성하는 방법을 알아보자. 최종적으로 키바나에서 캐글 데이터를 분석하는 방법을 살펴본다

① 실습 진행과정



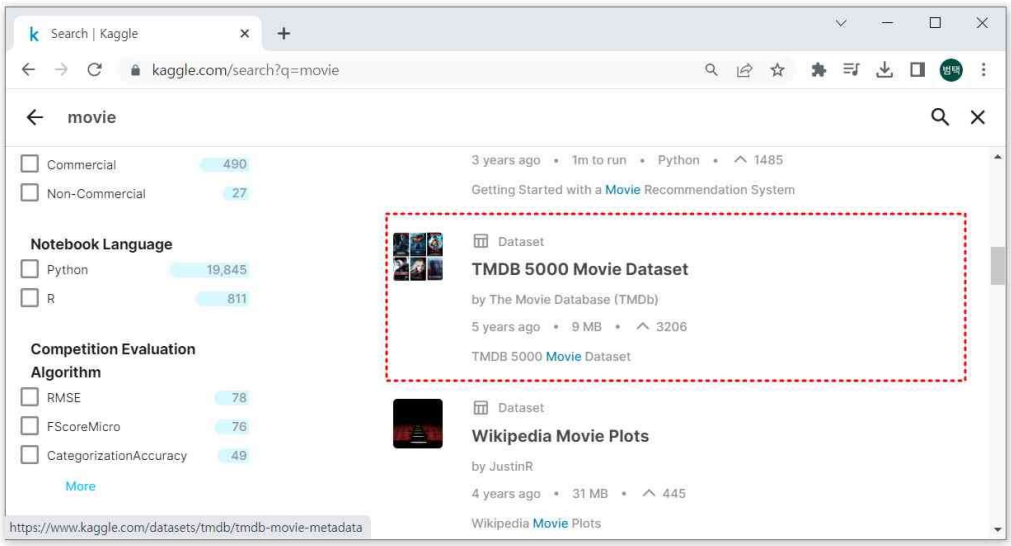
❶ 캐글 사이트에서 CSV 파일을 가져오는 방법을 통해 양질의 데이터를 검색하고 얻는 방법을 배울 수 있다
❷ 키바나 데이터 비주얼라이저를 통해 간단히 CSV 파일을 엘라스틱서치로 올려보고 빠르게 데이터를 탐색할 수 있다
❸ 로그스태시를 이용해 문자열을 파싱하기 위해 루비 필터나 CSV 필터의 세부 옵션들을 알아본다
❹ 인덱스 매핑과 인덱스 템플릿을 이용해 인덱스를 생성
❺ 키바나를 통해 시각화를 해 본다. 두 가지 시나리오를 미리 정해 그에 맞춰 데이터를정제 하는데, 10년간 가장 많이 만들어진 영화 장르가 무엇인지와 할리우드영화에서 평점과 수익/예산의 상관관계를 알아본다.

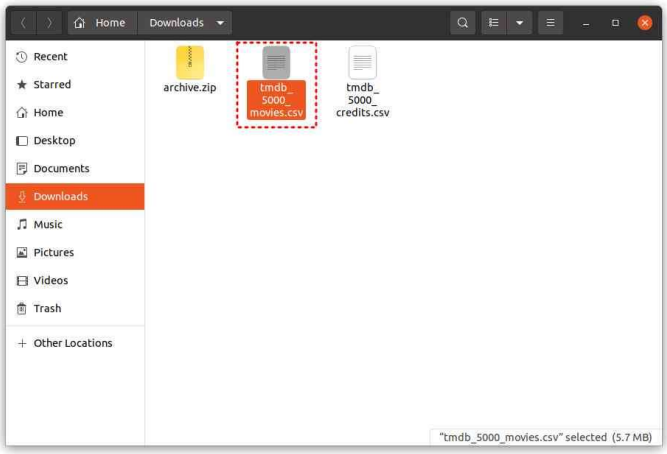
② 실습 준비 작업

1. 캐글사이트(https://www.kaggle.com) 회원가입
2. 엘라스틱서치, 키바나, 로그스태시를 설치하고 실행

③ CSV 파일 가져오기

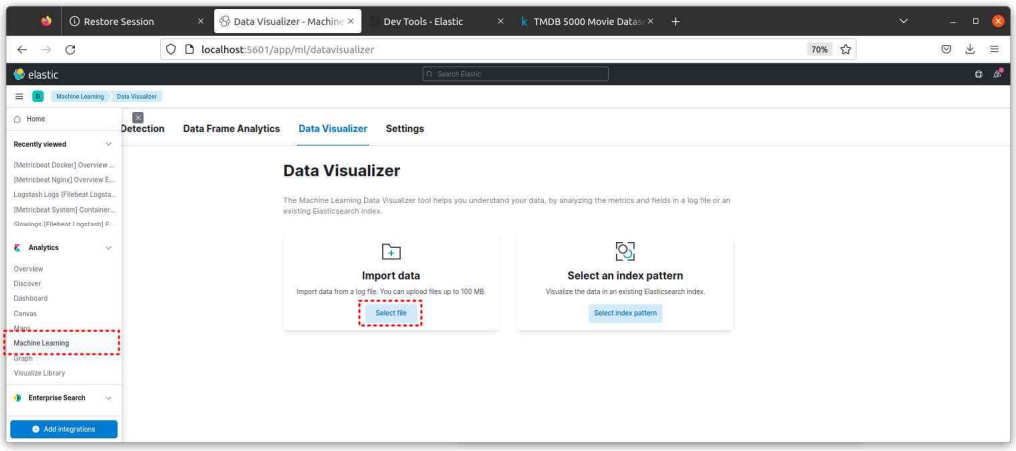
search 에서 movie 를 입력한 뒤 'TMDB 5000 Movie Dataset' 이라는 데이터를 선택한다. 다운로드 하면 2개의 파일이 압축되어 있을 것이다. 이 중 tmdb_5000_movies.csv 파일만을 사용할 것이다.





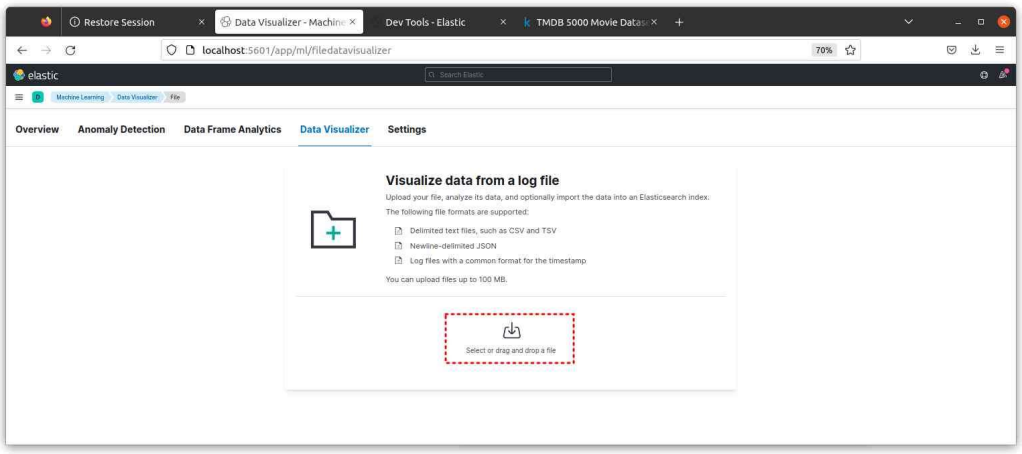
④ 키바나 데이터 비주얼 라이저를 이용한 파일 가져오기

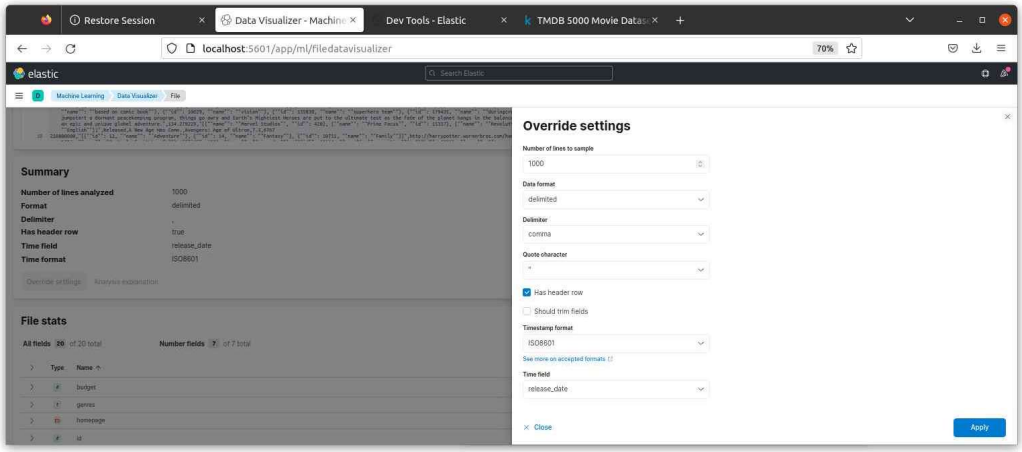
키바나에서 제공하는 Data Visualizer 메뉴를 이용하여 CSV 파일을 엘라스틱 서치에 저장해 본다. 웹 UI 가 제공하는 메뉴만을 이용해 쉽게 CSV 파일을 엘라스틱 서치에 저장할 수 있다. 데이터 비주얼라이저는 100MB 이하의 CSV(쉼표로 분리된 형식), TSV(탭으로 분리된 형식), JSON 형식의 로그파일을 지원한다. 키바나 왼족 상단의 토글 메뉴에서 키바나 메뉴를 확인할 수 있는데 Machine Learning 을 선택하면 아래와 같은 화면을 볼 수 있다.



Machine Learning 메뉴는 베이직 라이선스 이하에서는 사용할 수 없다. 미리 Stack Management > License Management 에서 30일 업데이트 버전으로 변경해 두어야 한다.

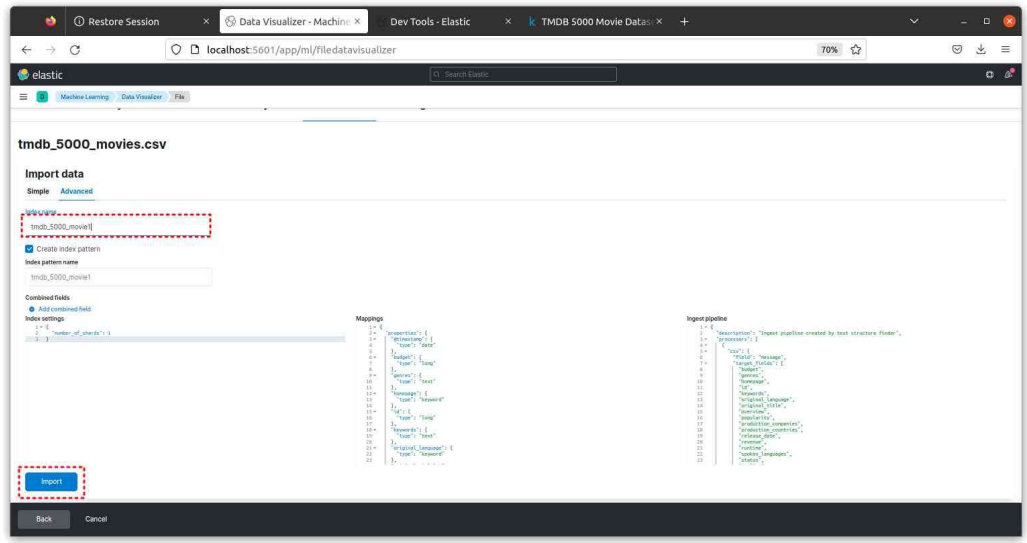
”Select File" 을 클릭한 뒤, “Select or drag and drop a file" 을 클릭하여 "tmdb_5000_movies.csv” 파일을 선택한다. 엘라스틱서치가 파일을 자체적으로 분석한다.





파일 내용과 요약정보, 각 칼럼의 통계 정보를 보여준다. 개별 칼럼의 최소/중간/최대값을 보여주고 유니크한 값의 비율도 보여준다. 데이터의 분포를 확인하거나 컬럼값의 범위 같은 간단한 확인 작업도 가능하다. Summary 의 Override settings 버튼에서는 컬럼명 변경이 가능하고 시간/날짜형 데이터 타입을 설정하고 타임 포맷도 변경할 수 있다. 키바나에서는 시간/날짜 데이터를 이용해 필터링을 자주 수행하기 때문에 시간/날짜 컬럼이 있다.면 Time field 로 설정하는 것이 좋다. 여기에서는 release_date 컬럼을 Time field 로 설정했다. 페이지 하단의 Import 버튼을 누르면 인덱스 매핑과 인덱스 설정이 가능하다.

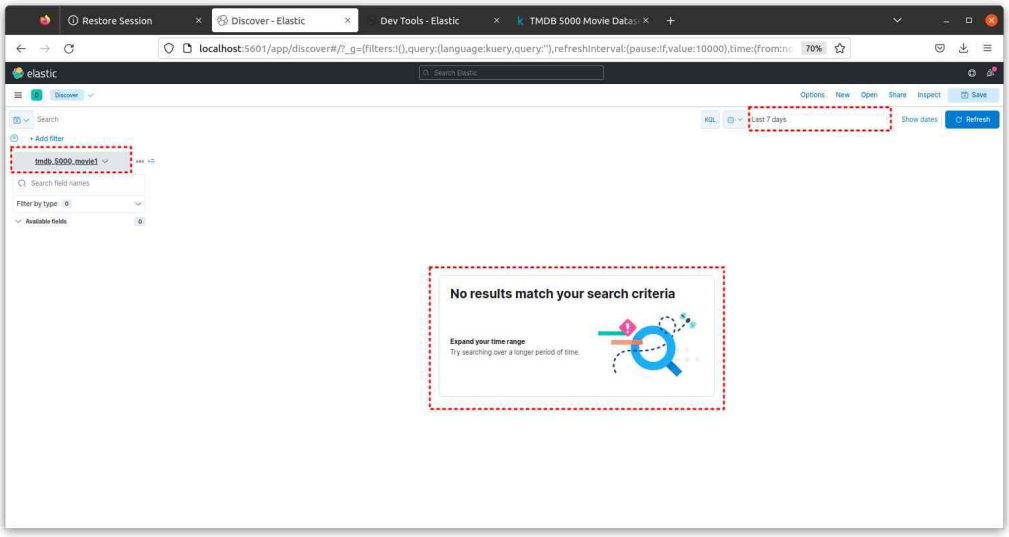
인덱스 설정에서는 간단한 설정(simple)과 고급 설정(Advanced)이 있는데 간단한 설정은 인덱스 이름 정도만 설정하면 엘라스틱이 다이내믹 매핑을 이용해 데이터를 매핑하고 인덱스를 생성한다.
고급설정은 인덱스 설정과 매핑, 그리고 인제스트 파이프라인을 이용해 데이터 변환까지 가능하다. tmdb_5000_movie1 이라는 이름으로 index name 과 index pattern name을 만들어보자. 페이지 하단의 import 버튼을 누르면 된다.



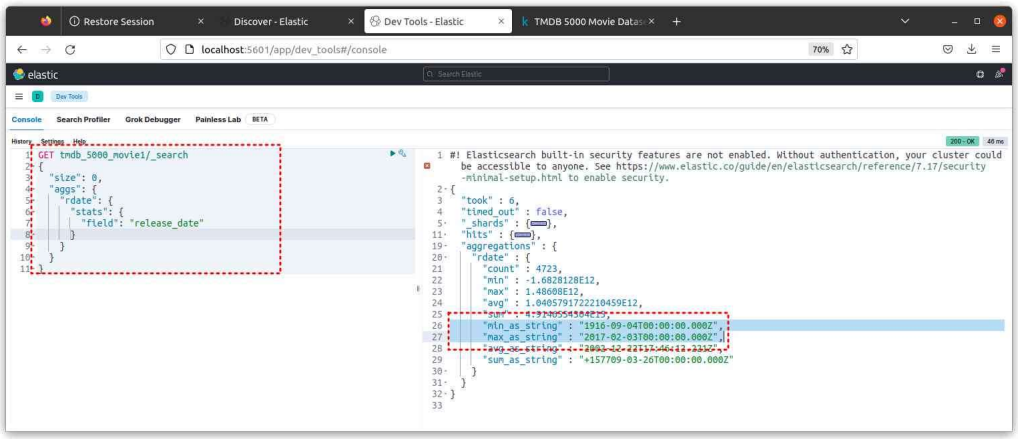
4723개의 도큐먼트가 인덱싱되었고 80개의 도큐먼트가 인덱싱 되지 못했다는 결과를 확인할 수 있다. 지금은 실패한 도큐먼트들은 무시하고 진행하도록 한다.

■ 데이터 Discover

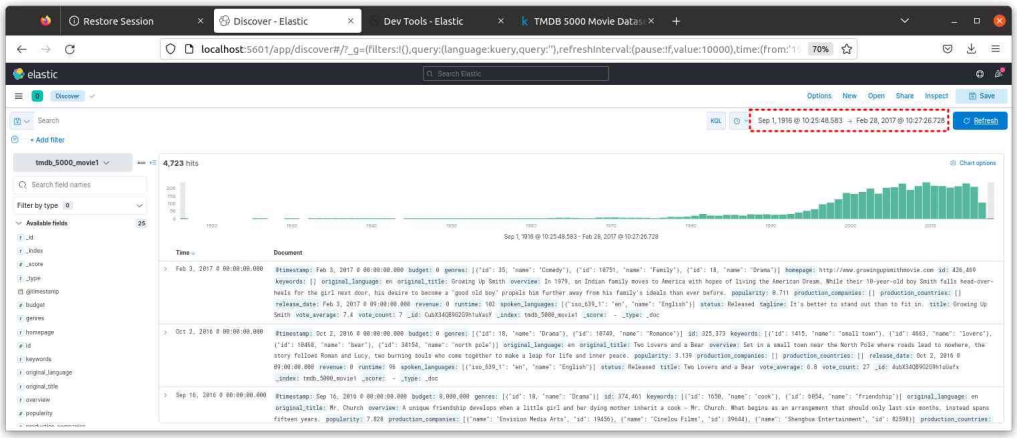
이제 인덱싱된 데이터를 확인해 보자. 먼저 키바나 Discover 메뉴를 확인해 보자.



아무런 데이터도 확인할 수 없다. 인덱스 생성시 기본 시간/날자 컬럼이 release_data 로 설정되었는데 타임피커는 "Last 7 days" 로 되어 있으므로 과거의 영화정보 데이터를 확인할 수 없는 것이다. release_date 의 범위를 확인하기 위해 키바나 콘솔에서 메트릭 집계를 통해 데이터 범위를 확인할 수 있다.

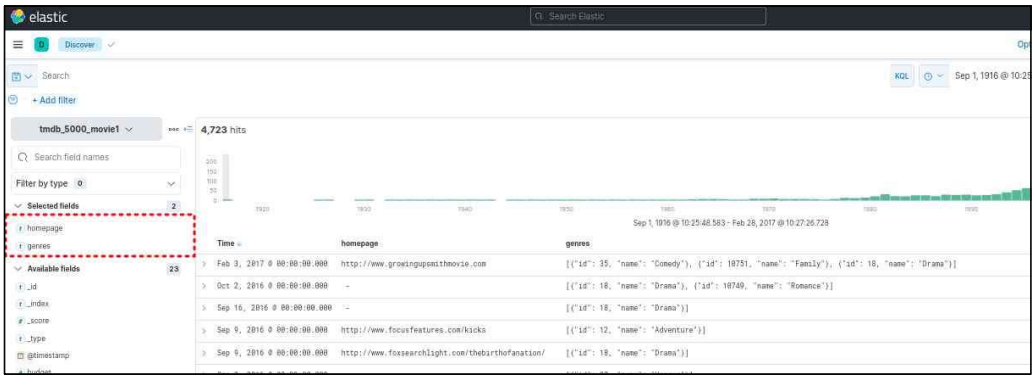


release_data 는 1916-09-04 부터 2017-02-03 까지의 범위를 갖고 있다. 아래와 같이 타임피커를 수정한다.



위와 같이 타임피커 수정후 데이터를 확인할 수 있다. 키바나를 사용할 때는 항상 날짜/시간 범위에 주의해야 한다. 이제 로그스태시를 이용해 문자열을 분석하는 작업을 진행해 본다.

⑤ 로그 스태시를 이용한 파일 가져오기



데이터 확인을 위해 homepage 〉 genres 를 선택해 보면 영화 장르가 "name": "Drama" 와 같이 JSON 형태로 되어 있는것을 확인할 수 있다. 실제 분석을 위해서는 ['Drama', 'Romance'] 와 같은 객체 배열 형태로 저장되어야 한다. 이 외에도 전처리 작업이 필요한데, 로그 스태시를 이용하여 원본 CSV 파일을 우리가 원하는 형태로 변환해 본다.

■ CSV 파일 읽기

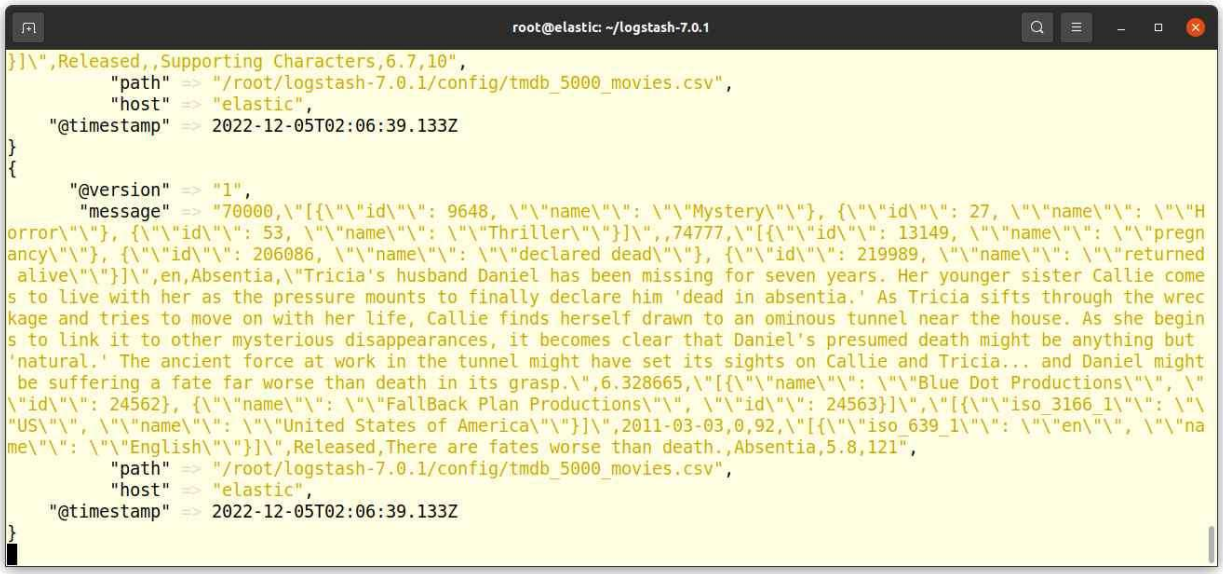
로그스태시 config 디렉터리에 logstash-tmdb.conf 파일을 생성하고 파이프라인을 작성한다

```
input {
  file {
    path => "/root/logstash-7.0.1/config/tmdb_5000_movies.csv"
    start_position => "beginning"
    sincedb_path => "nul"
  }
}

output {
  stdout {}
}
```

아래와 같이 실행해 본다.

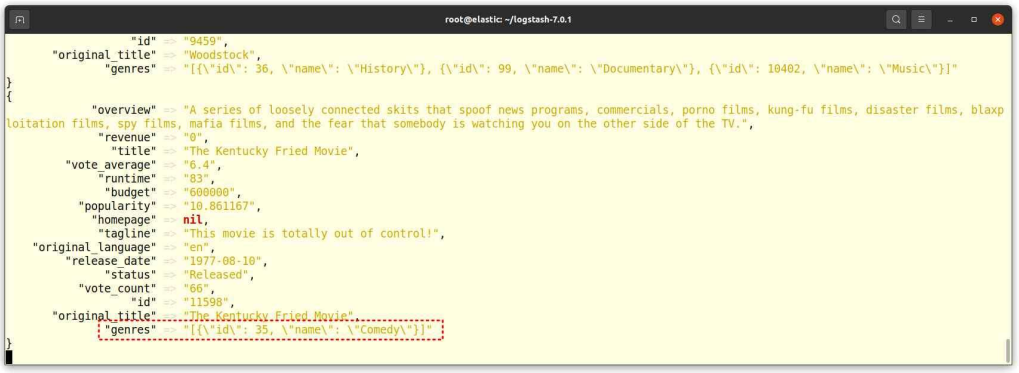
```
root@elastic:~/logstash-7.0.1# ./bin/logstash -f ./config/logstash-tmdb.conf
```



message 필드를 로그스태시 파이프라인을 이용하여 우리가 원하는 형태로 가공할 것이다. 대부분 CSV 파일의 첫라인에는 컬럼명이 적혀있는데 컬럼명 라인은 제외해야 한다.

```
root@elastic:~/logstash-7.0.1/config# head -1 tmdb_5000_movies.csv
budget,genres,homepage,id,keywords,original_language,original_title,overview,popularity,production_companies,production_countries,_release_date,revenue,runtime,spoken_languages,status,tagline,title,vote_average,vote_count
root@elastic:~/logstash-7.0.1/config# vi logstash-tmdb.conf
```

다시한번 ./bin/logstash -f ./config/logstash-tmdb.conf 를 실행하고 결과를 확인해 본다. 만약 출력이 되지 않는 상태가 지속된다면, 편집기를 이용하여 tmdb_5000_movies.csv 파일을 열고 파일 내용을 일부 수정한 뒤, 다시 원래대로 돌리고 저장, Quit 하면 출력이 진행될 것이다.



CSV 헤더 로그가 사라졌고 message 가 CSV 필터에 의해 쉼표를 기준으로 다양한 필드로 쪼개지면서 우리가 정의한 컬럼순으로 데이터가 할당된다. remove_field 에 적어두었던 6개의 필드도 보이지 않는다. 하지만 여전히 문제가 있다. genres 가 배열이 아닌 하나의 긴 텍스트 형태라는 점과 날짜/시간 데이터인 _release_data 가 날짜/시간 포맷이 아니라는 점이 문제이다.

■ 날짜/시간 타입으로 파싱

date 필터 플러그인을 사용하여 _release_date 를 텍스트 타입이 아닌 엘라스틱서치가 사용하는 날짜/시간 타입으로 변경한다.

```
root@elastic: ~/logstash-7.0.1
  }
}

filter {
  csv {
    separator => ","
    columns => ["budget","genres","homepage","id","keywords","original_language","original_title","overview","popularity","
production_companies","production_countries","release_date","revenue","runtime","spoken_languages","status","tagline","titl
e","vote_average","vote_count"]
    remove_field => ["message", "production_companies", "production_countries", "keywords", "spoken_languages", "@timestamp
", "path", "@version", "host"]
    skip_header => true
  }
  date {
    match => ["release_date", "YYYY-MM-dd"]
    target => "releasedate"
    timezone => "UTC"
    remove_field => "release_date"
  }
}

output {
  stdout {}
"config/logstash-tmdb.conf" 26L, 814C                                          20,21         83%
```

이제 다시 한번 실행한 뒤, 결과를 확인해 본다.

```
root@elastic: ~/logstash-7.0.1
}
{
             "runtime" => "80",
             "revenue" => "8000000",
                  "id" => "13282",
               "title" => "Death Race 2000",
        "vote_average" => "5.9",
            "homepage" => nil,
    "original_language" => "en",
              "status" => "Released",
          "popularity" => "6.473667",
            "overview" => "In a boorish future, the government sponsors a popular, but bloody, cross-country race in which
 points are scored by mowing down pedestrians. Five teams, each comprised of a male and female, compete using cars equipped
 with deadly weapons. Frankenstein, the mysterious returning champion, has become America's hero, but this time he has a pa
ssenger from the underground resistance.",
             "tagline" => "A Cross Country Road Wreck!",
              "genres" => "[{\"id\": 28, \"name\": \"Action\"}, {\"id\": 35, \"name\": \"Comedy\"}, {\"id\": 878, \"name\"
: \"Science Fiction\"}]",
        "releasedate" => 1975-04-27T00:00:00.000Z,
      "original_title" => "Death Race 2000",
              "budget" => "300000",
          "vote_count" => "124"
}
```

release_date 필드가 releasedate 필드로 이름이 바뀌었고 엘라스틱에서 사용 가능한 날짜/시간 타입으로 변경되었다. 이제 genres 필드를 수정해 보자.

■ 루비 필터를 이용한 파싱

genres 필드를 원래 의도했던 배열 형태로 바궈보자. 여기에는 루비 필터 플러그인을 사용할 것이다. 루비 필터 플러그인은 플러그인 내부에서 루비(Ruby)언어를 실행할 수 있게 한다. 로그스태시 설정 파일에 직접 루비 코드를 넣는 인라인 방법과 루비 파일을 만들고 이를 불러와 실행하는 방식이 있다. 우리는 복잡한 코드가 아니어서 인라인 코드로 작성할 계획이다.

두가지 방법이 있는데 하나씩 살펴보자. 먼저 루비 필터만 이용해 JSON 을 파싱하는 방법이다. logstash-tmdb.conf 파일을 수정한다.

```
    timezone => "UTC"
    remove_field => "release_date"
  }
  ruby {
    code => "
      genres = JSON.parse(event.get('genres')).map{ |genre| genre['name'] }
      event.set('genres', genres)
    "
  }
}

output {
```

루비 플러그인의 code 내부는 루비 언어이다. code 외에 루비 필터에서 사용할 수 있는 옵션은 다음과 같다

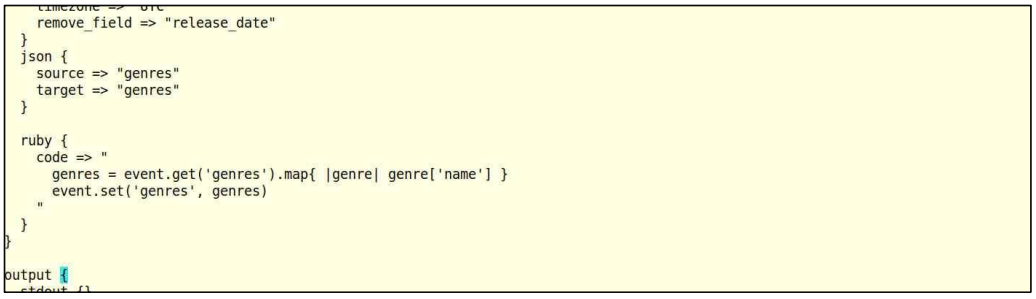
옵션	설명
code	각 이벤트의 필터링을 위한 루비 코드를 작성한다
init	로그스태시가 초기화되는 시점에서 수행할 코드. 모든 이벤트에 대해 공통으로 사용될 함수나 상수 등을 정의할 때 사용한다
path	코드를 설정에 모두 작성하는 대신 별도의 파일로 작성하고 해당 파일을 가리킬 수 있다. 이때 코드 파일에는 filter 함수가 구현되어 있어야 한다.

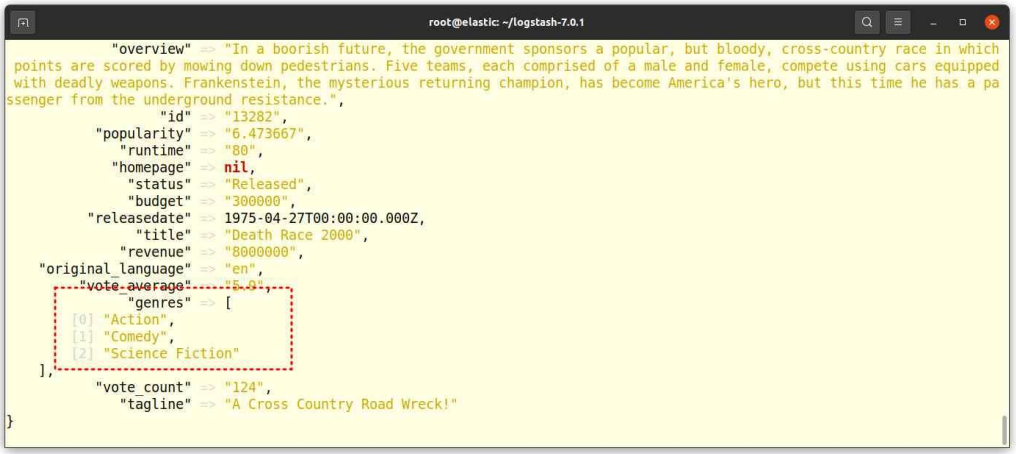
기본적으로 루비 필터를 사용하기 위해서는 루비 언어와 로그스태시에서 제공하는 이벤트 API 에 대한 기초 지식이 필요하다. JSON.parse() 는 JSON 을 파싱하는 함수이며 로그스태시에서 제공하는 API 인 event.get 으로 특정 컬럼의 문자열을 가져오고 event.set 으로 특정 컬럼에 문자열을 쓴다. map 은 루비 map 함수로, 열거형 데이터의 값을 가져올 수 있다.

이번에는 다른 방법으로 루비 필터를 사용해 보자. json 필터 플러그인으로 파싱한 후 루비 필터에서 매핑하는 방식인데, JSON 형식을 파싱할 때 검증된 필터를 사용하므로 예외 상황에 대한 대응이 수월하므로 추천하는 방식이다. json 필터 플러그인은 JSON 구문을 해석하는 플러그인으로 다음과 같이 크게 4개의 옵션이 있다.

옵션	설명
source	파싱되지 않는 JSON 문자열이 들어 있는 필드를 지정한다
target	파싱된 객체를 담을 대상 필드를 지정한다. 설정하지 않으면 루트 객체에 객체를 병합한다
skip_on_invalid_json	잘못된 형식의 JSON 문자열이 입력되었을 때 별도의 경고 없이 이를 무시한다
tag_on_failure	JSON 문자열 파싱에 실패했을 때ㅏ tags 필드에 추가할 태그를 의미한다. 기본값은 _jsonparsefailure 로 디버깅에 유용하다.

이제 기존 설정내용을 지우고 다음과 같이 작성해 본다.





이와 동일하게 genres 가 배열 형태의 데이터로 변환되어 출력된다.

⑥ 인덱스 매핑

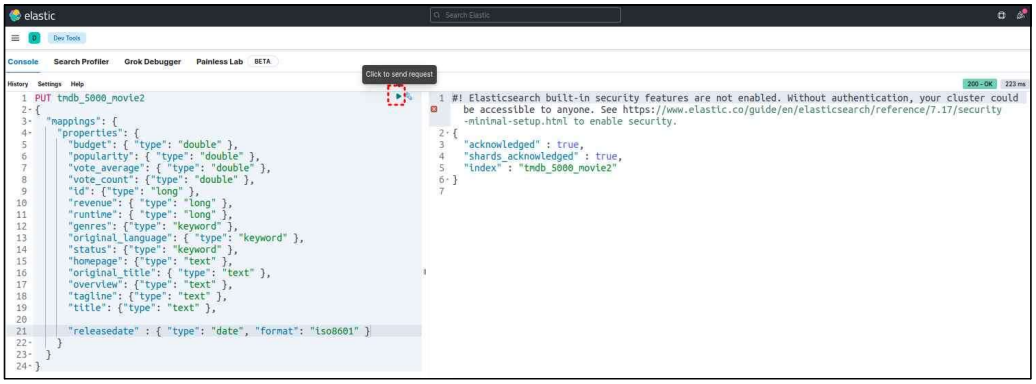
로그스태시에서 파싱된 데이터들이 엘라스틱서치로 인덱싱되기 위해서는 엘라스틱서치 매핑작업이 필요하다. 다이내믹 매핑을 지원하지만 최적의 성능을 이끌어내기 위해서는 매핑을 최적화활 필요가 있다. 여기서는 두 가지 인덱싱 방법을 적용해 볼 것이다.

■ 인덱스 매핑을 적용하여 로그스태시 데이터 저장

이제는 테스트를 위해 사용하던 표준 출력(stdout) 대신 엘라스틱서치에 직접 인덱싱을 수행한다. 기존 출력 부분을 다음과 같이 변경한다.

```
output {
  elasticsearch {
    index => "tmdb_5000_movie2"
  }
}
```

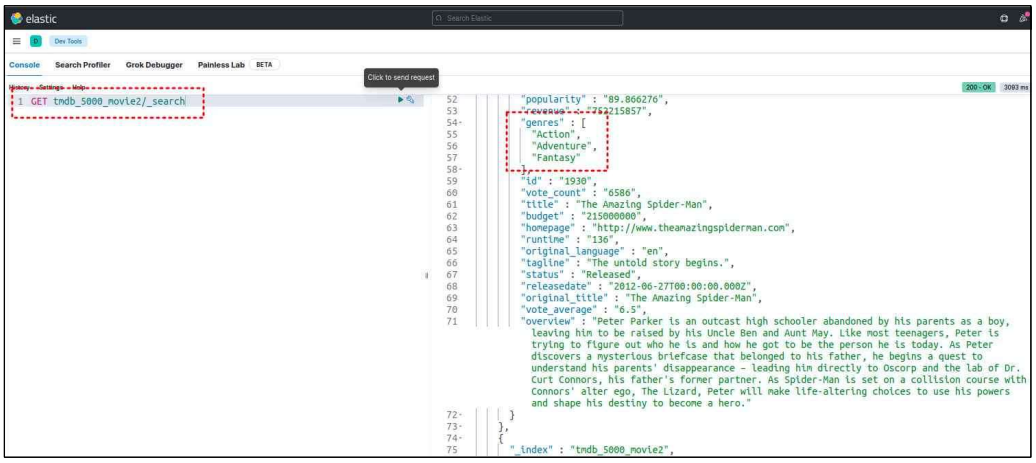
로그스태시를 실행하기 전에 한 가지 해야 할 일이 있다. 우리는 필드의 데이터 타입을 알고 있기 때문에 다이내믹 매핑보다는 명시적 매핑을 하는 것이 좋다. 다이내믹 매핑이 범용성이 뛰어나지만 필드 타입을 모두 알고 있다면 다이내믹을 할 이유는 없다. 우선 인덱스를 생성하고 원하는 형태로 매핑해야한다. 키바나 콘솔에서 tmdb_5000_movie2 인덱스를 만들려면 매핑을 적용해 보자.



text 로 매핑한 tagline, overview 같은 필드는 전문 검색이 필요해 보인다. original_title 같은 경우도 문자는 짧지만 전문 검색이 필요한 경우가 있다. 예를 들어 "엽기적인 그녀" 라는 영화를 검색할 때 '엽기', '그녀'로 검색해도 검색이 되어야 하기 때문이다. 반면 status 나 original_language 같은 경우는 범주형 데이터이

기 때문에 keyword 로 매핑했다. 그 외에 숫자 형식은 long, double 형태로 매핑되었고, 날짜/시간 데이터인 release_date 는 date 타입으로 매핑되었다.

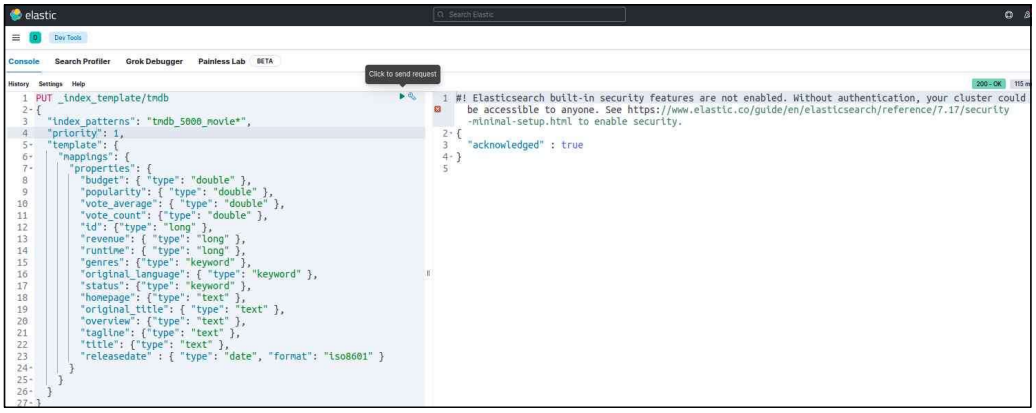
마지막으로 로그스태시를 실행해 본다.



genres 가 배열 형태로 저장되었고, releasedate 도 날짜 형식으로 잘 저장되어 있음을 확인할 수 있다.

■ 템플릿을 적용하여 로그스태시 데이터 저장

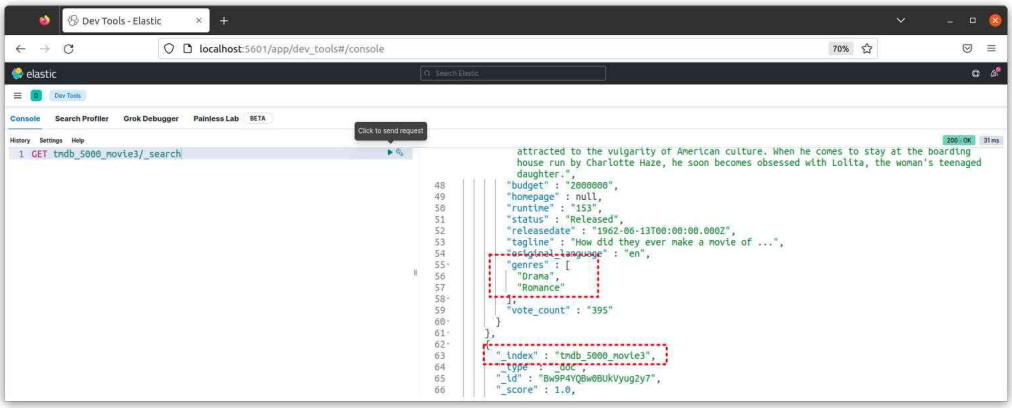
이번에는 인덱스 템플릿을 이용하여 인덱스를 매핑해본다. 인덱스 템플릿을 이용하면 설정이 동일한 복수의 인덱스를 만들 때 유리하다. 로그스태시를 이용해 대량의 실시간 데이터를 입력받는 경우 인덱스를 직접 생성하며 매핑을 적용하기 보다 인덱스 템플릿을 이용하는 경우가 많다. 일반적으로 엘라스틱서치를 이용한다면 데이터의 양이 많을 것이기에 성능 최적화를 위해 날짜별로 인덱스를 분리하기 때문이다. 키바나 콘솔에서 다음 API 를 요청해 보자.



logstash-tmdb.conf 에서 엘라스틱서치 인덱스 이름을 변경한 뒤, 로그스태시를 재 실행해 본다.

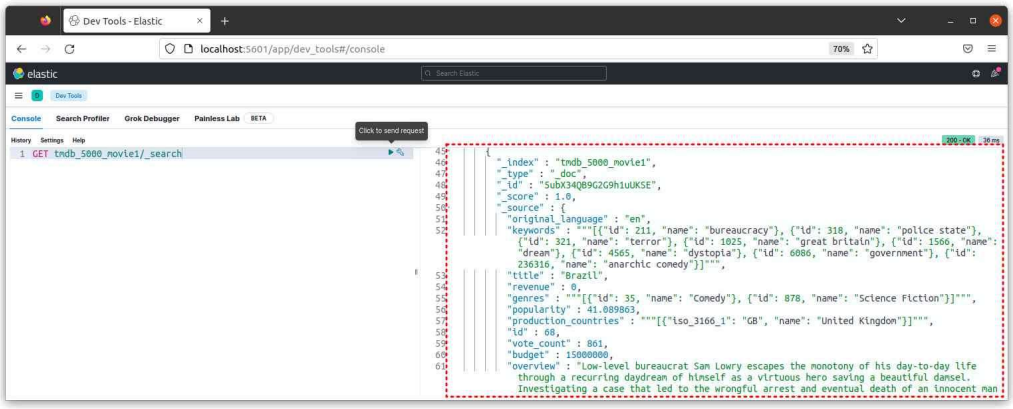
```
  }
}

output {
  elasticsearch {
    index => "tmdb_5000_movie3"
  }
}
```

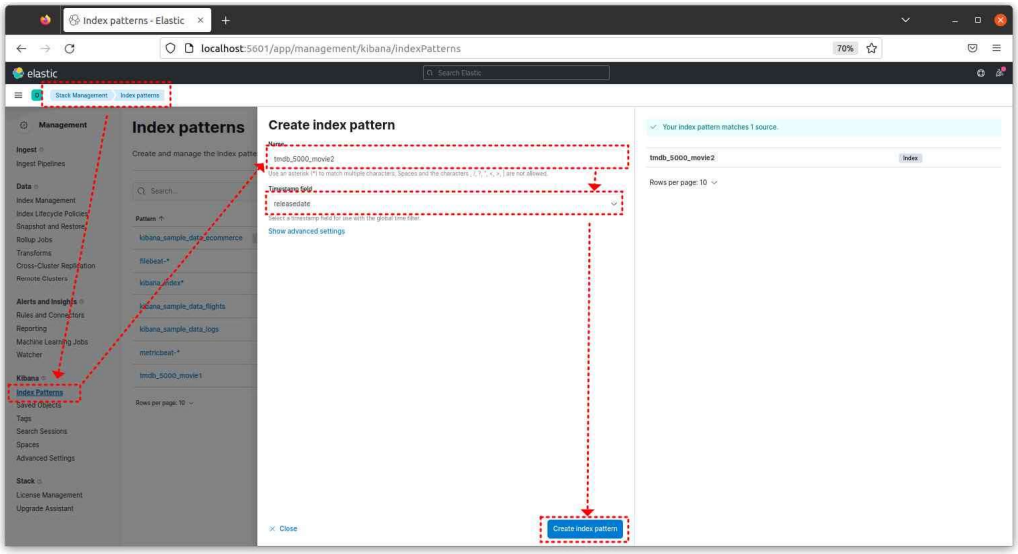


⑦ 키바나에서 간단히 분석해 보기

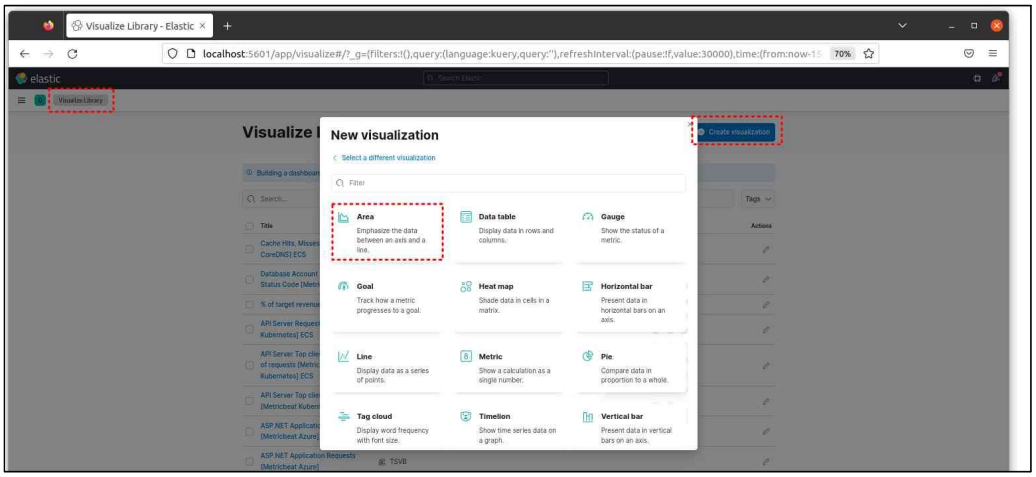
데이터를 수집했으니 키바나에서 이를 간단히 분석해 보자. 현재까지 3개의 인덱스를 만들었다. tmdb_5000_movie1 은 데이터 비주얼라이저를 이용해 만들었는데 문자 파싱이 되지 않았다.



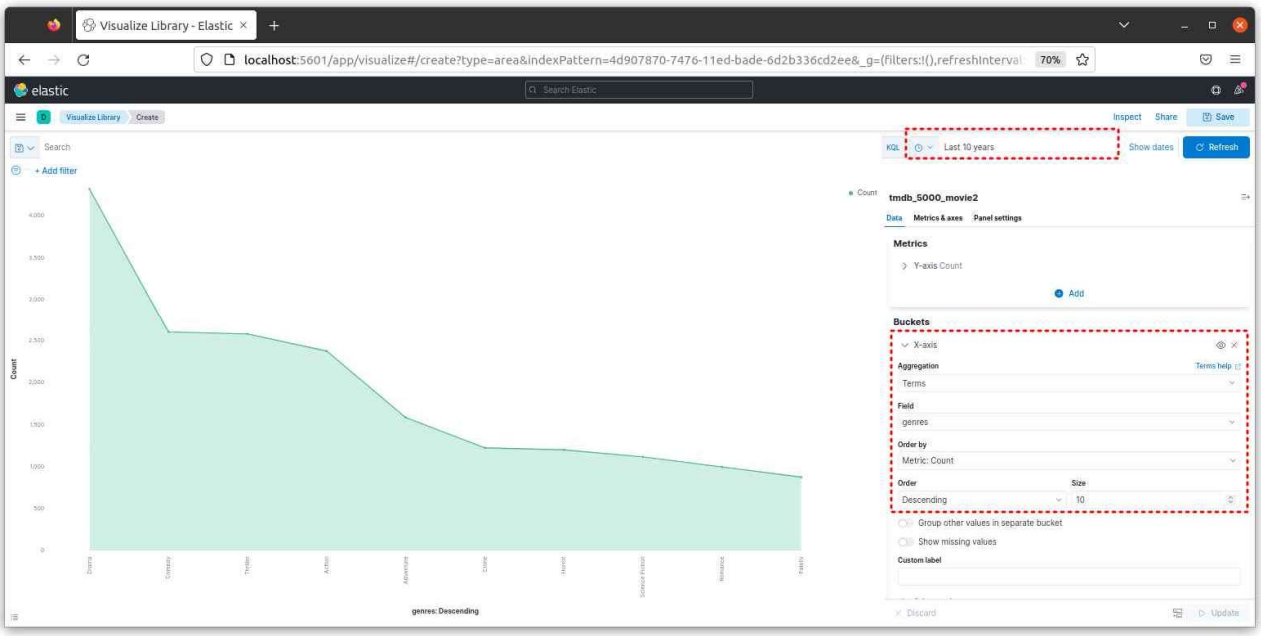
2,3 은 로그스태시에서 문자열 파싱과 전처리를 거쳤고 완벽하게 같은 형태이며 포함된 도큐먼트까지 같은 인덱스이다. 편의상 2를 사용하자. 키바나에서 엘라스틱 서치 인덱스를 이용하기 위해서는 먼저 키바나와 인덱스 패턴을 지정해야 한다.



■ 10년동안 어떤 영화 장르가 가장 많이 만들어졌는가?



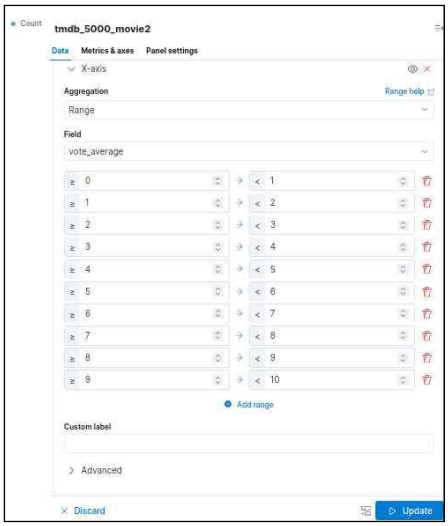
위와 같이 키바나의 시각화에서 그래프를 그려보기 위해 새로운 시각화를 만들고 시각화 타입은 Area 그래프를 선택한다. 인덱스 패턴은 "tmdb_5000_movie2" 를 선택한다. 시각화 메뉴가 나오면 먼저 타임 필터를 지정한다. 지난 10년 이므로 시간 설정을 다시해야 한다.



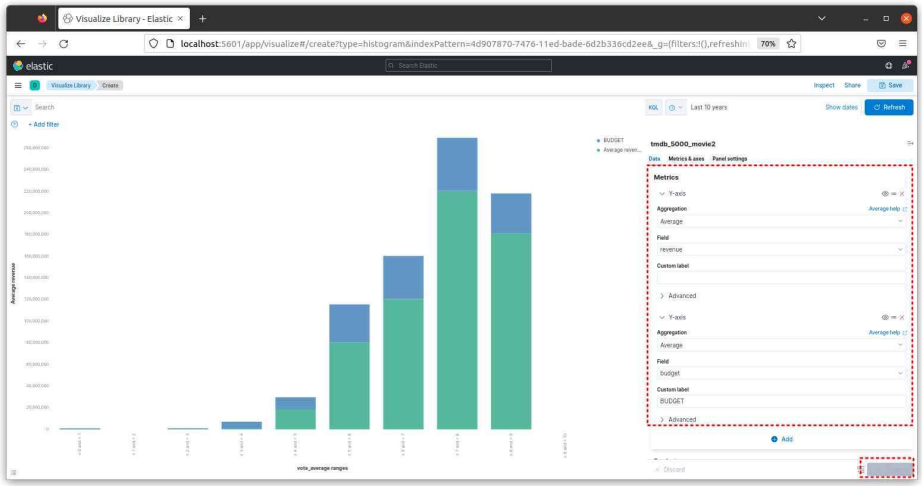
■ 할리우드 영화 중 평점 높은 영화는 수익이 났을까? 예산은?

이번에는 할리우드 영화를 기준으로 좋은 영화를 만들면 수익이 났는지 알아보자. 여기에서는 관객 평점이 높으면 좋은 영화라고 가정한다. 그리고 과거 할리우드 블록버스터 영화들은 평점이 낮은 경향이 있었는데 실제로도 그런지 알아보자. 키바나 메뉴에서 시각화를 선택하고 새로운 시각화를 생성한다. 타입은 Vertical Bar 타입을 선택하자.

인덱스 패턴은 역시 tmdb_5000_movie2 를 선택한다. 시간은 앞선 설정과 동일하게 10년으로 한다. 이제 버킷과 메트릭을 설정해야 한다. 아래와 같이 X 축 버킷을 생성한다.

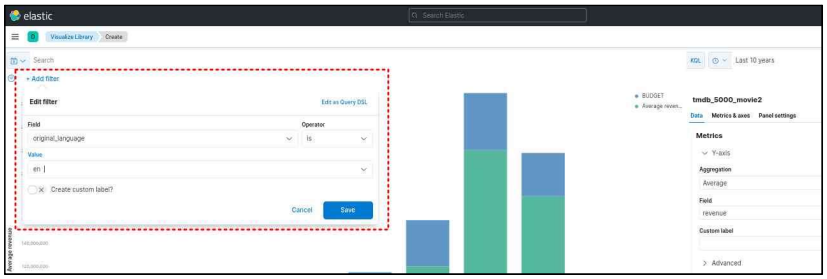


Aggregation 을 Histogram 으로 선택하고 interval 을 1 로 사용하면 될 것 같지만 이 경우 X 축이 유동적으로 변한다는 문제가 있다. 만약 8,9,10 점대 데이터가 없다면 Histogram 은 X 축에 8,9,10 이 보이지 않게 된다. Y 축에 아래와 같이 Metrics 를 설정한다.

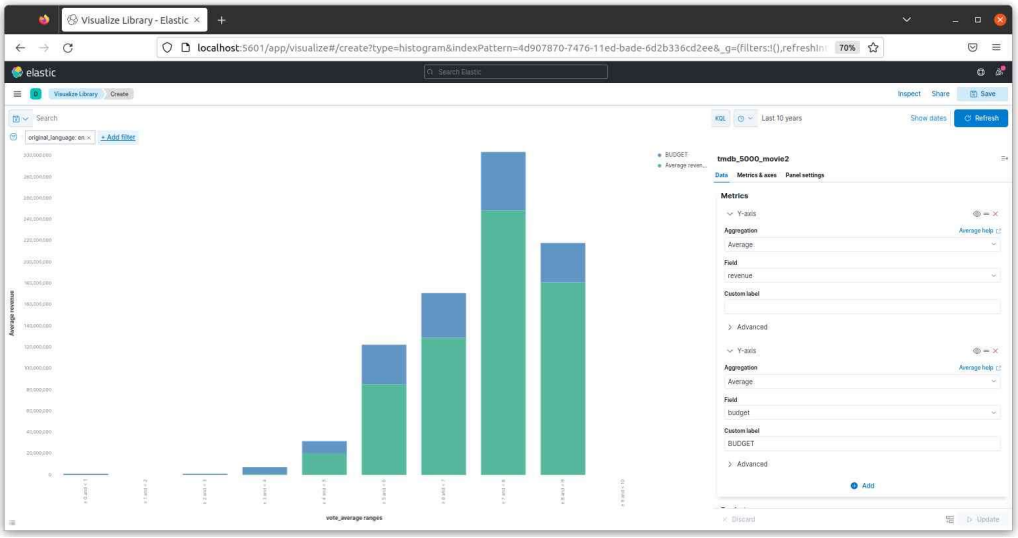


Y축에 두 가지 Metrics 를 사용할 때 두 메트릭값의 숫자 크기 차이가 큰 경우는 주의 해야 한다. 예를 들어 범위가 1~10 인 값과 1000~5000 인 값을 Y 축에서 같이 보려고 하면 작은 숫자의 그래프는 거의 보이지 않을 것이므로 주의해야 한다. 스케일이 비슷한 값들만 Y축으로 함께 보는 것이 좋다. 그리고 여기에서는 Aggregation 으로 Average 를 사용했지만 Percentile 을 이용하여 중간값 등을 같이 사용하면 더 양질의 결과를 얻을 수도 있다.

마지막으로 필터를 적용해 할리우드 영화만 필터링 해 본다.



Save 를 누르면 필터가 적용된다.



일반적으로 평점이 좋을 수록 수익이 높다는 것을 알 수 있다. 평균 평점이 5~8 점대의 작품들이 거의 비슷한 예산을 사용했음을 알 수 있다. 그럼 비 할리우드 영화에서는 이 그래프가 어떻게 바뀔까?
original_language:en 필터를 클릭하고 Exclude results 를 선택해 보자.