Report big data for company

Elasticsearch & Kibana

Yuchao QIAN Ying DAI Zhen Xing Song SHI

Part 1 Installation and data preparation

Prerequisites:

Elasticsearch 2.4+Kibana 4.6.1

Data set: IMDB Movie Dataset

In order to configure the environment easily, we need to use the tools called docker to build our working space image.

And the new version of Elasticsearch is not necessary and it has too many changes and features we will not use, so we decided to use the old version (2.4), which is much easier to use.

The first step to use docker is to write a docker file to put all the configuration into itself. So it is necessary to learn how to write a Dockerfile.

As we know there are lots of images are already on the cloud, so we can easily write a Dockerfile to have both Elasticsearch and Kibana using the following script:

```
FROM sebp/elk:es241_l240_k461
```

And we need to add the local file into the docker image, so we have to use the command ADD to add all the data and script from the local storage like this:

```
ADD ./start.sh /usr/local/bin/start.sh
```

And also we need to give the permission to run the script:

```
RUN chmod +x /usr/local/bin/start.sh
```

Then we have to expose some port for Elasticsearch and Kibana because it can be configured by restful API, so we have to use EXPOSE method like this:

```
EXPOSE 5601 9200 9300
```

Finally, we need to run the script we store in the image and it will only run once a script when the image is online the first time by adding this:

```
CMD [ "/usr/local/bin/start.sh" ]
```

After finish the Dockerfile, we need to prepare our dataset for Elasticsearch.

The example from the Elasticsearch documentation tell us that Elasticsearch can import data from JSON file by using _bulk API like the following command:

```
curl -XPUT localhost:9200/_bulk --data-binary @/data/movie.json
```

The only prerequisite is that we have to give the index, type and id for each record in the JSON file like this:

```
{"index":{"_index":"movies","_type":"record","_id":1}}
```

So we have to prepare our JSON file in advance for importing by bulk.

In order to prepare the dataset, we use ATOM and a script to create our JSON file from a csv file we download from the website. And here is the screenshot of the dataset of JSON format.

And here is part of the ATOM script to convert the CSV file to JSON file.

```
json-converter.coffee                Welcome Guide
YAML = require 'js-yaml'
converter = require('json-2-csv')
{CompositeDisposable} = require 'atom'
id = 0
unique = (array) ->
  output = {}
  output[array[key]] = array[key] for key in [0...array.length]
  value for key, value of output

isEven = (val) ->
  return val % 2 == 0

expandAction = (docs, opType='index') ->
  metaIndex = atom.config.get('json-converter.elasticIndex')
  metaType = atom.config.get('json-converter.elasticDocType')
  metaId = atom.config.get('json-converter.elasticUidField')
  metaParentId = atom.config.get('json-converter.elasticParentUidField')
  excludeFields = atom.config.get('json-converter.elasticExcludeFields')

  actions = []
  for doc in docs
    docKeys = Object.keys(doc)
    id = id+1
    action = {}
    action[opType] = {}
    action[opType]._index = metaIndex if metaIndex
    action[opType]._type = metaType if metaType
    action[opType]._id = id
    action[opType]._parent = doc[metaParentId] if metaParentId in docKeys

    if opType is 'delete'
      actions.push(JSON.stringify(action))
    else
      for field, value of doc
        delete doc[field] if field in excludeFields
      doc = {"doc": doc} if opType is 'update'
      actions.push(JSON.stringify(action))
      actions.push(JSON.stringify(doc))

  return actions
```

After all the above steps, we will have a Dockerfile, a JSON file.

So we need to add the data into the docker image by doing this:

```
ADD ./data /data
```

We need to modify the start.sh file from the example project and add the following code to run import the data.

```
curl -XPUT localhost:9200/_bulk --data-binary @/data/movie.json
```

Then we need to build our docker image and run it by the following command:

```
sudo docker build -t elakiba:1.0.1 .

sudo docker run -p 9200:9200 -p 5601:5601 elakiba:1.0.1
```

We also met the problem with the virtual machine that max_map_count is too small, so we need to set it to be 262144 as indicated.

```
sudo sysctl -w vm.max_map_count=262144
```

Part 2 Data analysis and dashboard

Then we have all the data in Elasticsearch server and we tried to use Kibana to plot the chart for visualization.

http://localhost:5601 is the url of Kibana and we should Browse http://localhost:5601 Uncheck "Index contains time-based events" Replace "logstash-*" by "mo*".

Firstly we chose to use a vertical bar chart to visualize the number of the movies per director, however, we met a big problem that the field “director_name” is analyzed, so Elasticsearch will split the name into words, so we can not have a full name of the director, which will casue a big problem for us.

So we tried to fix this problem. We go to the indices page in Kibana and we find out that all the strings are analyzed by default(like the following picture).

fields (34) | scripted fields (0)

name	type	format	analyzed	indexed
plot_keywords	string		✔	✔
country	string		✔	✔
aspect_ratio	number			✔
actor_3_facebook_likes	number			✔
num_critic_for_reviews	number			✔

We found out that we have to change the mapping before import the data to set the filed to be not_analyzed in order to let Elasticsearch not to split the word.

So we create a mapping file we set several string filed to be not_analyzed

```
{
  "mappings": {
    "record": {
      "properties": {
        "color":{"type":"string","index":"analyzed"},
        "director_name":{"type":"string","index":"not_analyzed"},
        "num_critic_for_reviews":{"type":"long"},
        "duration":{"type":"long"},
        "director_facebook_likes":{"type":"long"},
        "actor_3_facebook_likes":{"type":"long"},
        "actor_2_name":{"type":"string","index":"not_analyzed"},
        "actor_1_facebook_likes":{"type":"long"},
        "gross":{"type":"double"},
        "genres":{"type":"string"},
        "actor_1_name":{"type":"string","index":"not_analyzed"},
        "movie_title":{"type":"string"},
        "num_voted_users":{"type":"long"},
        "cast_total_facebook_likes":{"type":"long"},
        "actor_3_name":{"type":"string","index":"not_analyzed"},
        "facenumber_in_poster":{"type":"long"},
        "plot_strings":{"type":"string"},
        "movie_imdb_link":{"type":"string"},
        "num_user_for_reviews":{"type":"long"},
        "language":{"type":"string"},
        "country":{"type":"string"},
        "content_rating":{"type":"string","index":"not_analyzed"},
        "budget":{"type":"long"},
        "title_year":{"type":"long"},
        "actor_2_facebook_likes":{"type":"long"},
        "imdb_score":{"type":"double"},
        "aspect_ratio":{"type":"double"},
        "movie_facebook_likes":{"type":"long"}

      }
    }
  }
}
```

And then we do the same thing to put the file into Docker image

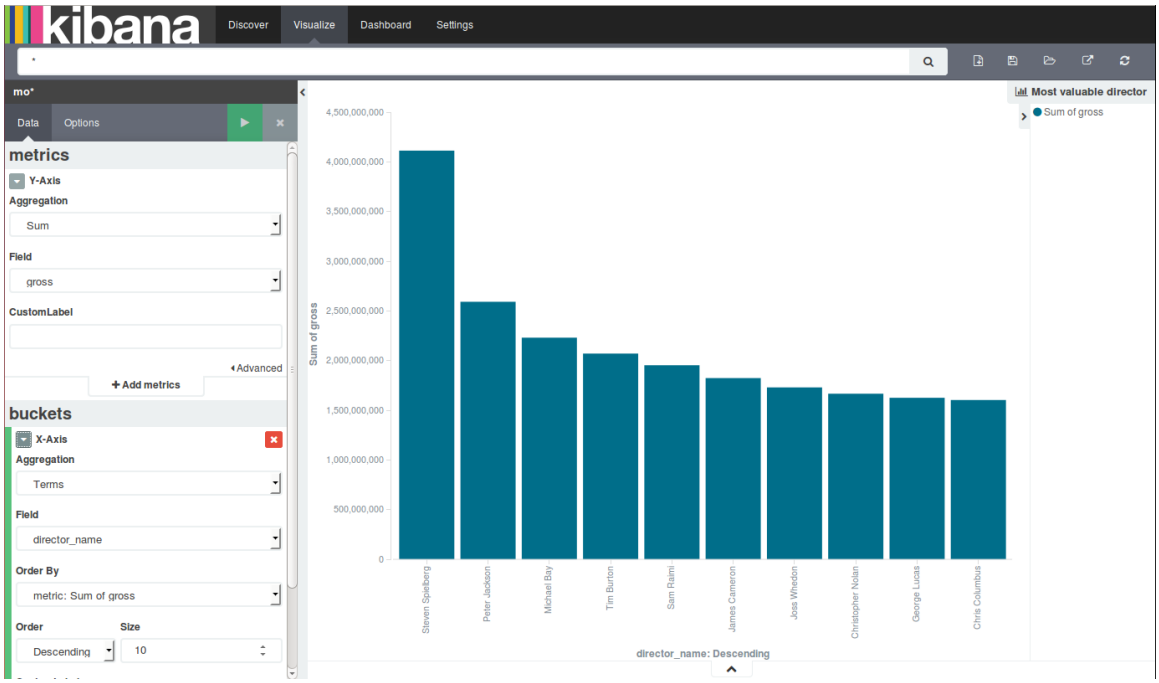
```
ADD ./mapping.json /tmp/mapping.json
```

And write following script in start.sh to put the mapping before import data.

```
curl -XPUT localhost:9200/movies -d @/tmp/mapping.json
```

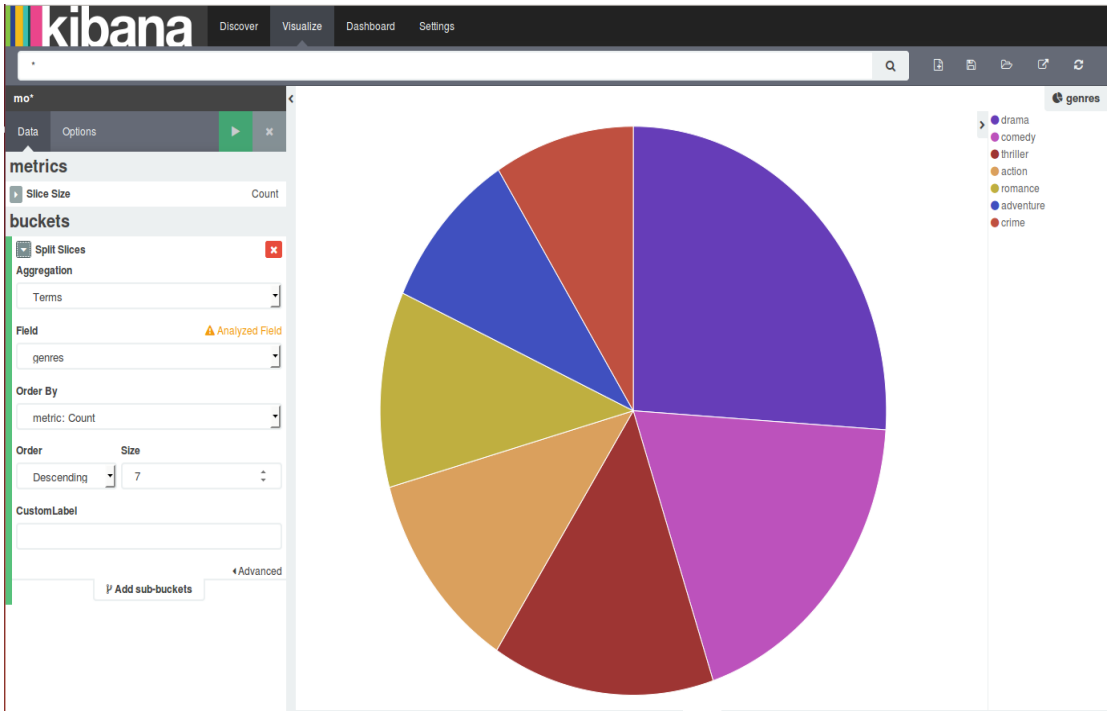
Then we can go to http://localhost:5601 again and trying to create the vertical bar chart.

We configure the visualization with the following setting and we can have a clear idea of who is the biggest money-maker during the past years.

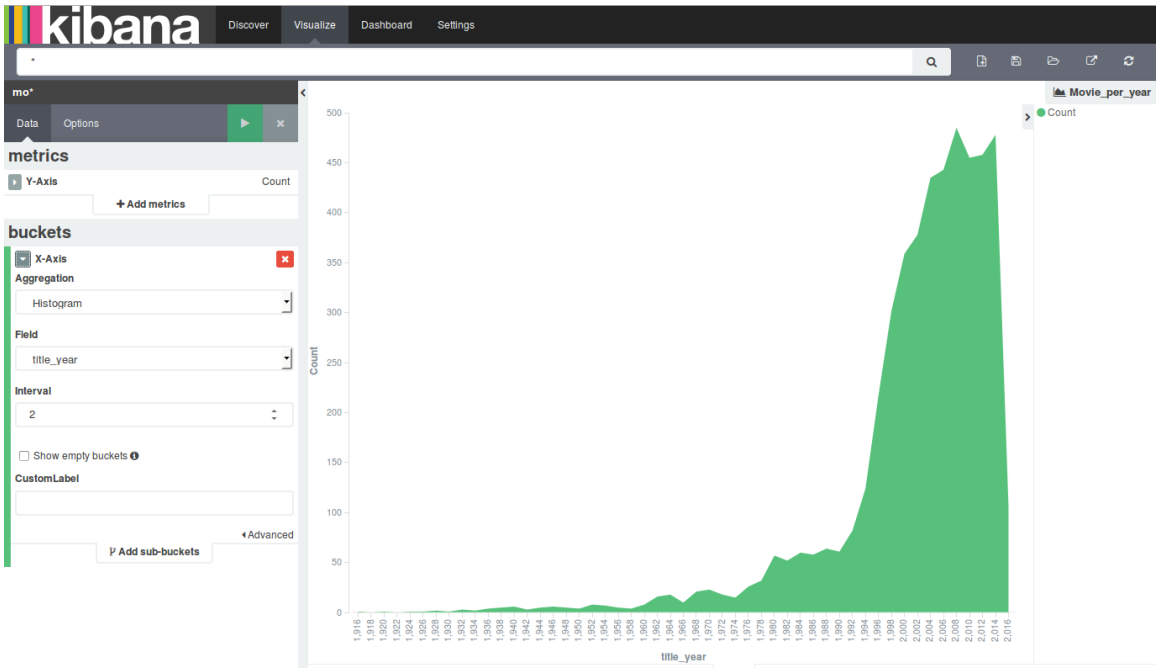


And then we create five more visualization to have a better idea of the dataset.

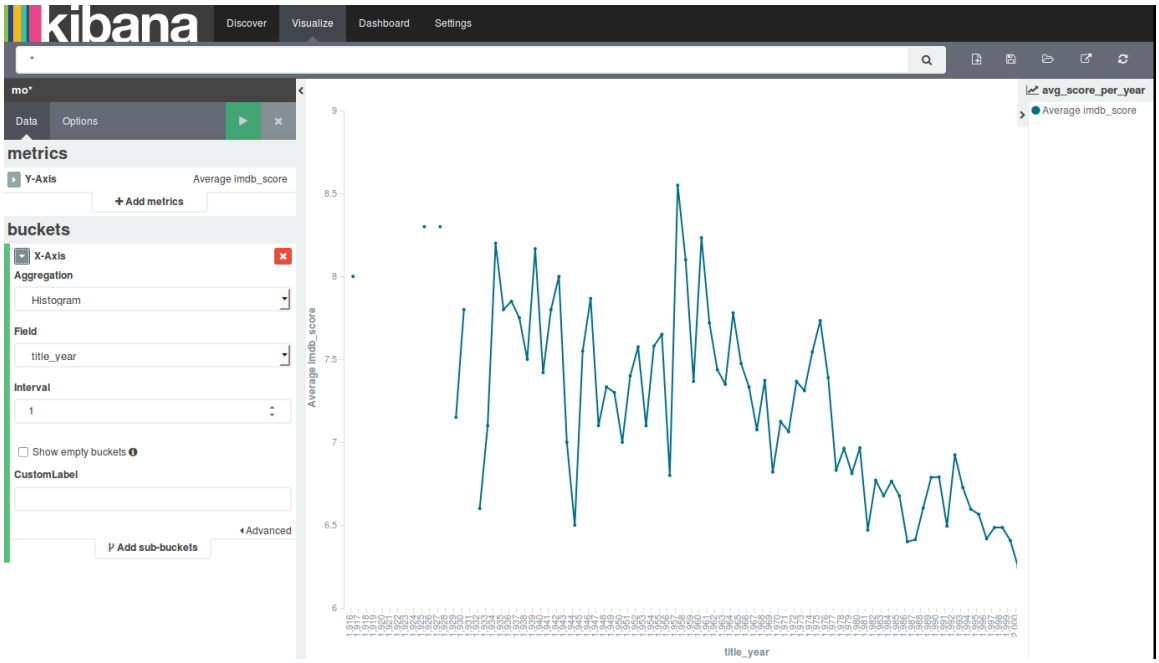
The most frequent genres of movies during the past years.



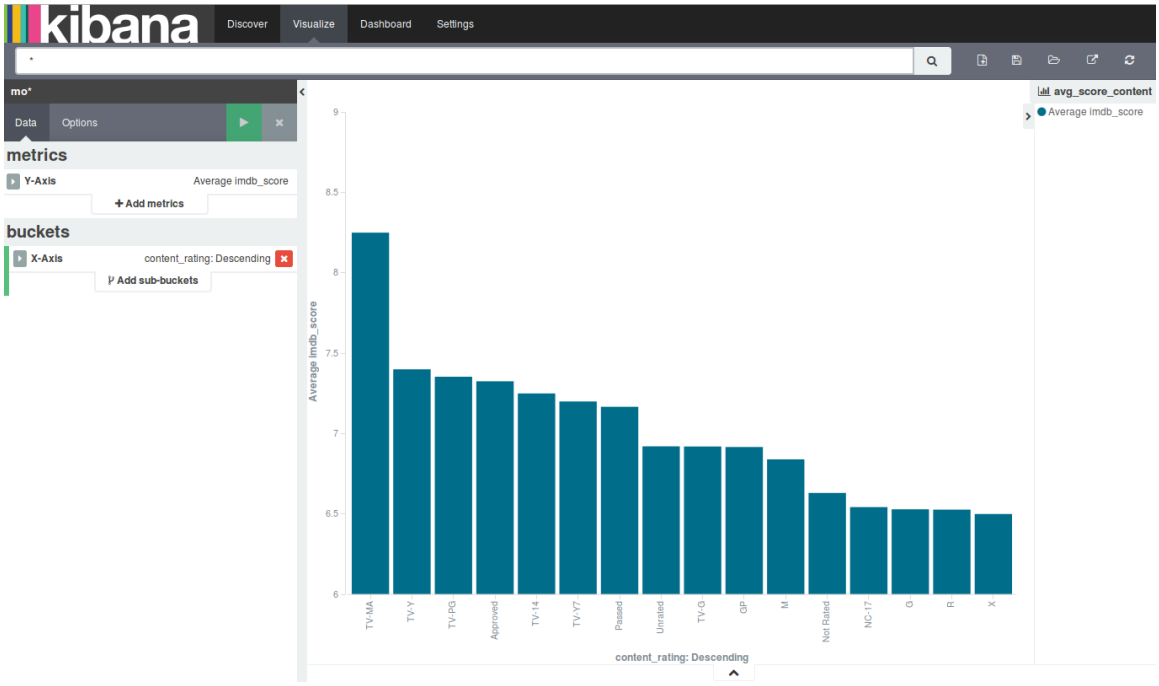
The trend of the amount of movies each year



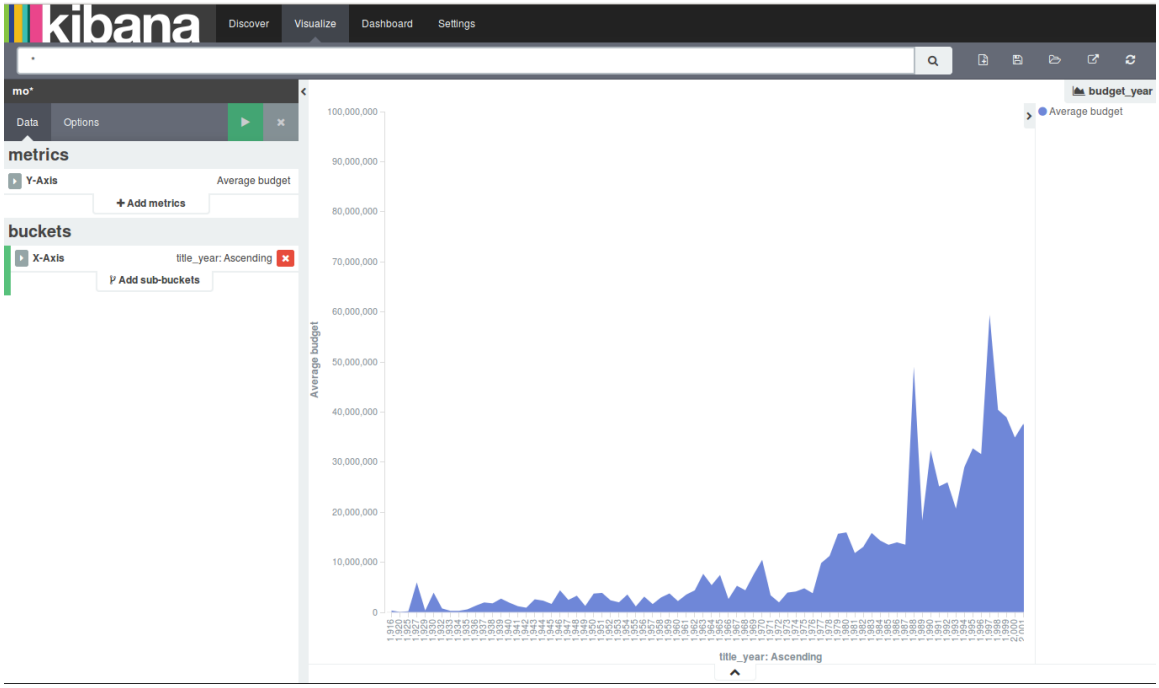
The average IMDE score by year



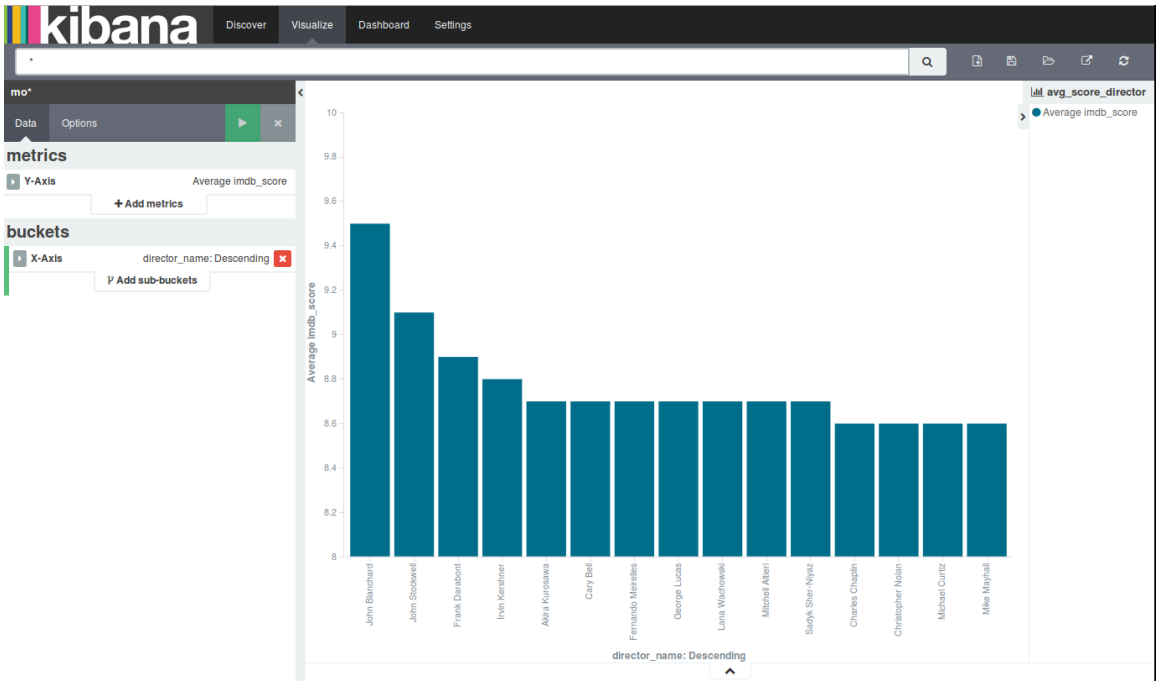
The average score by content level



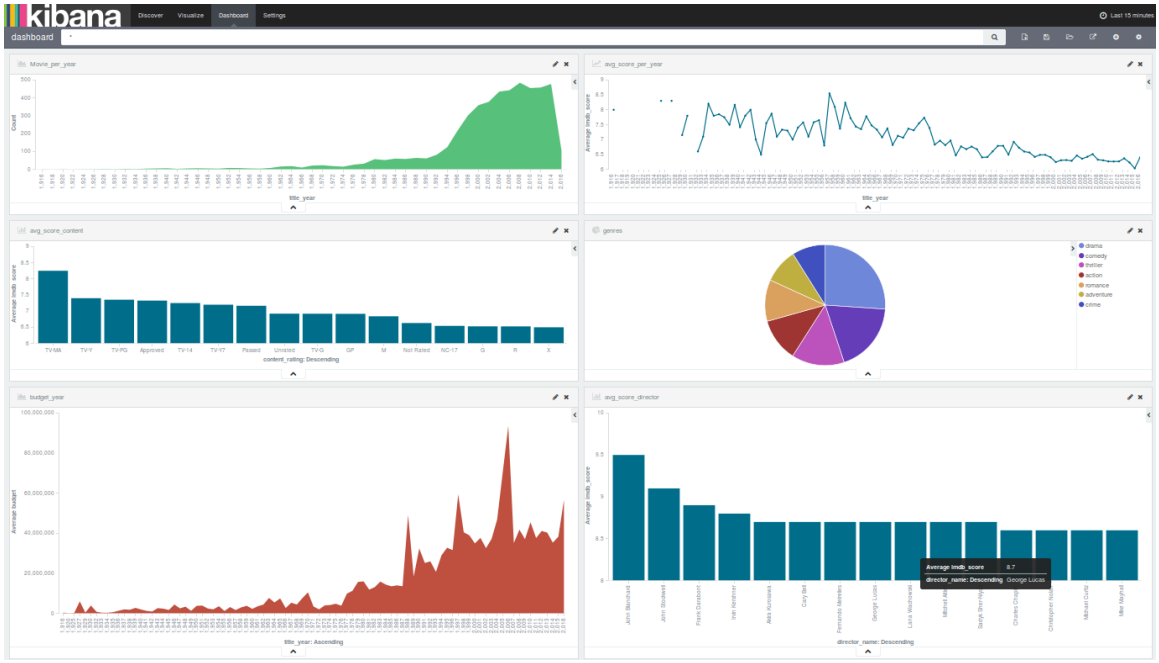
The trend of gross each year



The average IMDB score by director



Then we put all this visualization onto the dashboard.



Then the next step is to export the dashboard and visualizations also we can use the export.json to import the dashboard and visualizations.