June call RI

Elastic search

Diego Martín Ferández

UO276345

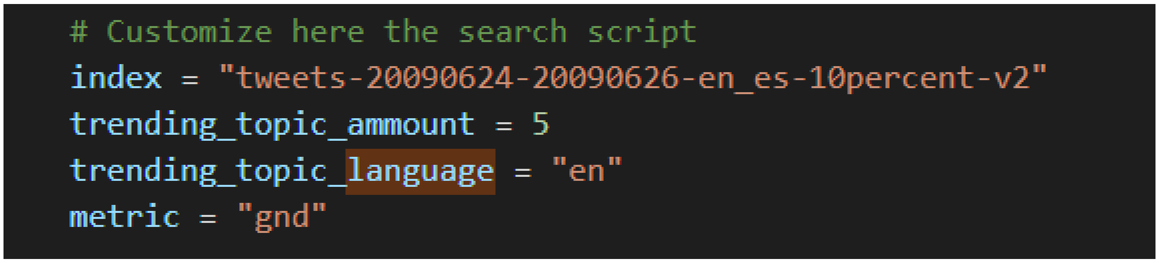
Index

Exercise 1

“Develop a script to produce a list of “trending topics” for each hour in the dataset.”

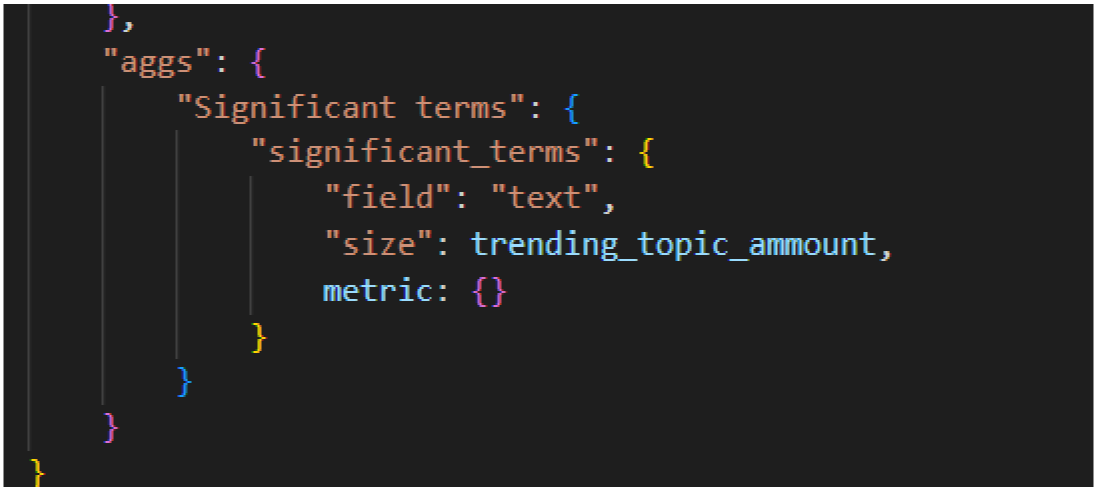
In this exercise we will use the index in the screenshot below because we want to remove the stopwords from the result which is the v2, otherwise we will use the normal index.

We will customize our script here:



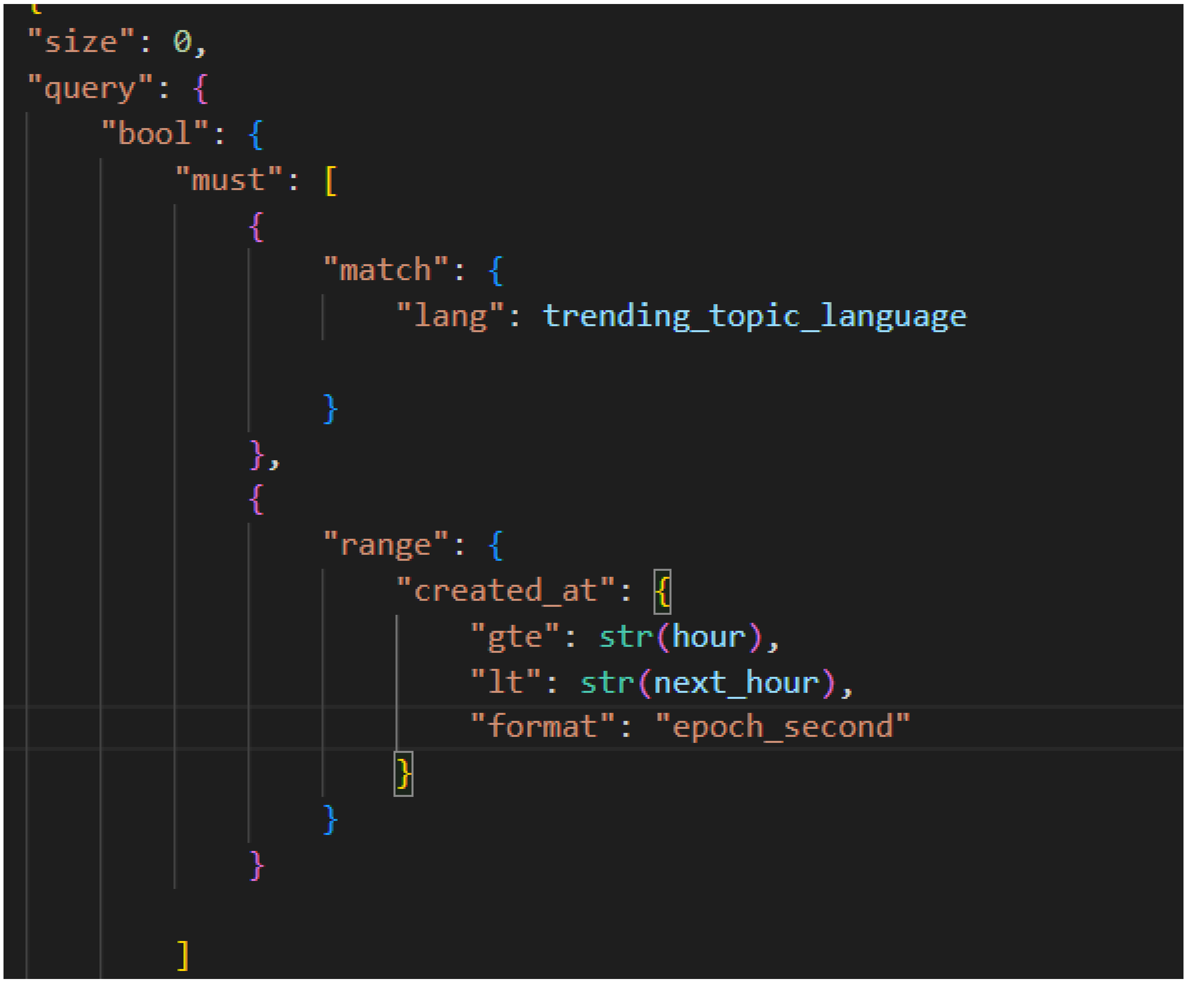
```
# Customize here the search script
index = "tweets-20090624-20090626-en_es-10percent-v2"
trending_topic_ammount = 5
trending_topic_language = "en"
metric = "gnd"
```

In “aggs” we will specify the trending topic amount, considering the customization above



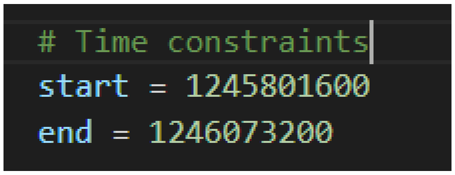
```
    },
    "aggs": {
        "Significant terms": {
            "significant_terms": {
                "field": "text",
                "size": trending_topic_ammount,
                metric: {}
            }
        }
    }
}
```

Moreover, we can also specify the conditions to select a trending topic like this, in the must condition:



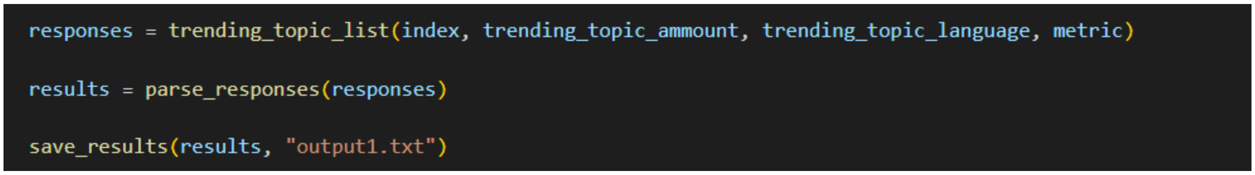
```
{
"size": 0,
"query": {
    "bool": {
        "must": [
            {
                "match": {
                    "lang": trending_topic_language

                }
            },
            {
                "range": {
                    "created_at": {
                        "gte": str(hour),
                        "lt": str(next_hour),
                        "format": "epoch_second"
                    }
                }
            }

        ]
```

Here we will set the query time range, this is because the query is executed once for every hour in the interval. It is set here using integers, it corresponds to an interval of 72 hours:



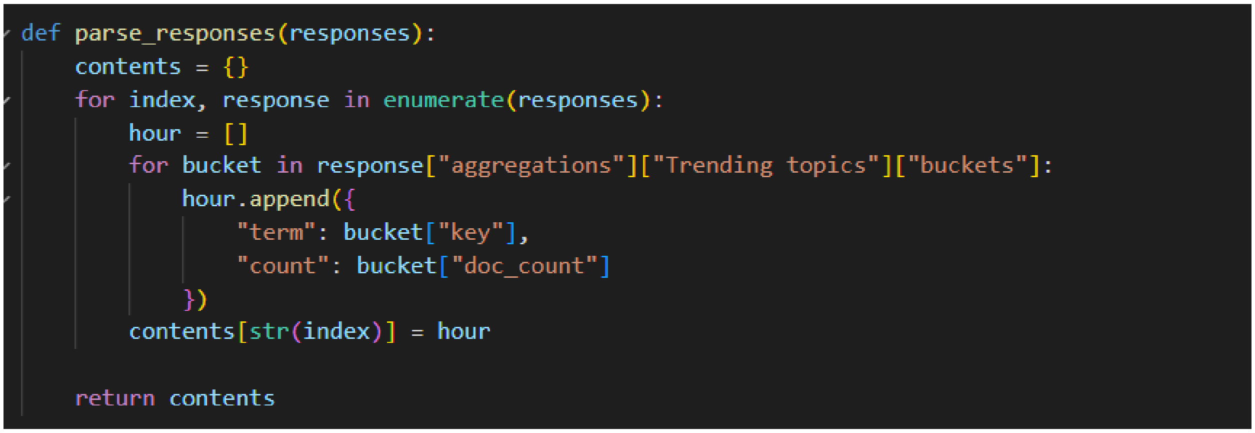
```
# Time constraints
start = 1245801600
end = 1246073200
```

For saving the results on a file we parse the response and dump it with json



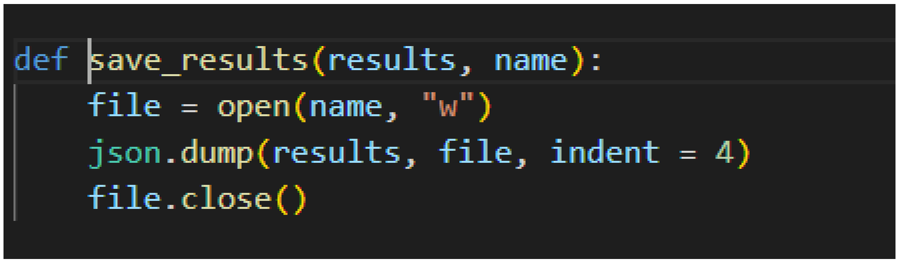
```
responses = trending_topic_list(index, trending_topic_ammount, trending_topic_language, metric)

results = parse_responses(responses)

save_results(results, "output1.txt")
```

Parsing the responses:



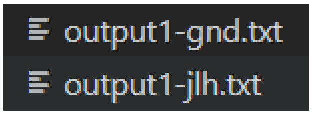
```python
def parse_responses(responses):
    contents = {}
    for index, response in enumerate(responses):
        hour = []
        for bucket in response["aggregations"]["Trending topics"]["buckets"]:
            hour.append({
                "term": bucket["key"],
                "count": bucket["doc_count"]
            })
        contents[str(index)] = hour

    return contents
```

Saving the results on a file:



```python
def save_results(results, name):
    file = open(name, "w")
    json.dump(results, file, indent = 4)
    file.close()
```

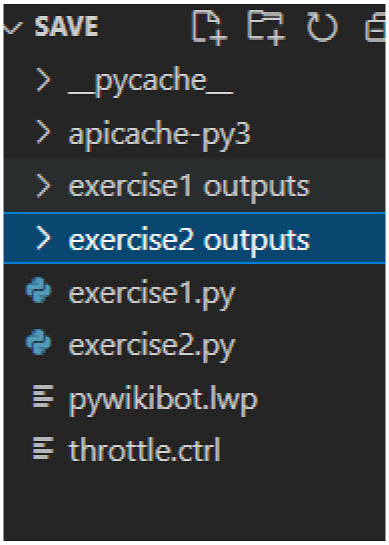
To see the different results I executed it with two different metrics. The outputs are on the folder of exercise 1 outputs:



output1-gnd.txt
output1-jlh.txt

Exercise 2

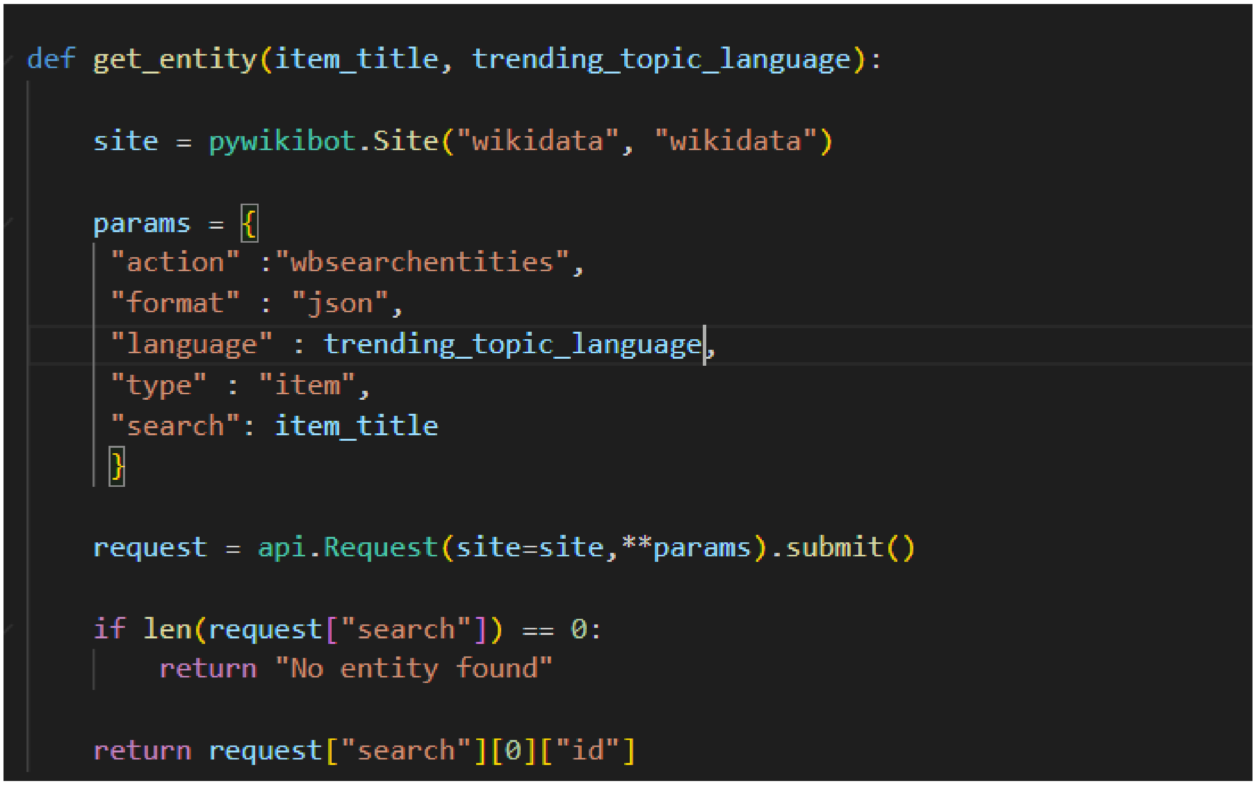
In this exercise we have to link the trending topics from exercise 1 to get their Wikidata entity. The outputs are stored in the exercise2 outputs folder with the respective metric. It takes between 8 and 10 min to finish the execution, moreover this library generates some folders that are not the output (they are removed in the delivery, but they will appear).



SAVE
- __pycache__
- apicache-py3
- exercise1 outputs
- exercise2 outputs
- exercise1.py
- exercise2.py
- pywikibot.lwp
- throttle.ctrl

The methods for parsing and storing the output from exercise1 are used here by importing them. So they must be in the same folder to work.

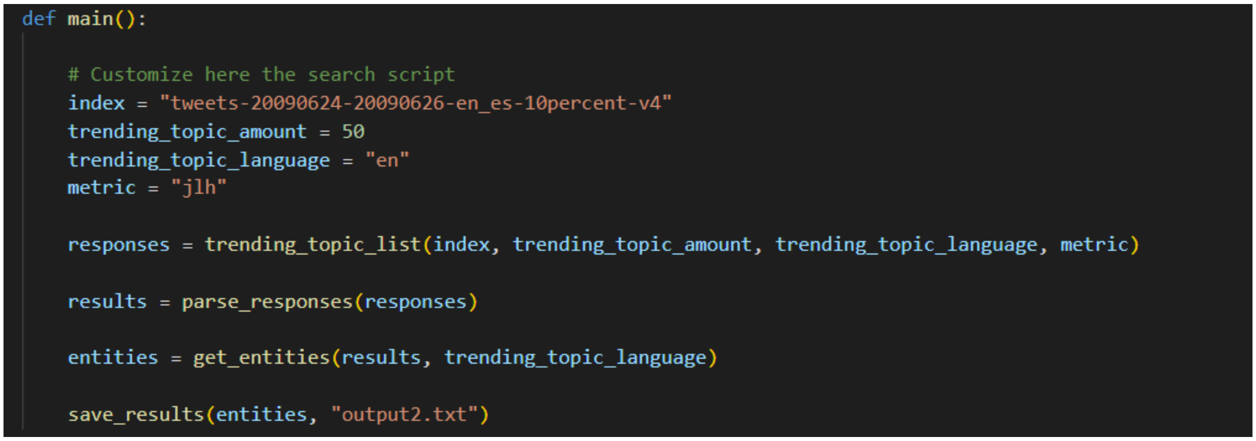
We use the pywikibot to get the entity linked to the title like this:



```python
def get_entity(item_title, trending_topic_language):

    site = pywikibot.Site("wikidata", "wikidata")

    params = {
     "action" :"wbsearchentities",
     "format" : "json",
     "language" : trending_topic_language,
     "type" : "item",
     "search": item_title
     }

    request = api.Request(site=site,**params).submit()

    if len(request["search"]) == 0:
        return "No entity found"

    return request["search"][0]["id"]
```

I got this piece of code from here:
https://www.wikidata.org/wiki/Wikidata:Pywikibot_-_Python_3_Tutorial/Quantities_and_Units

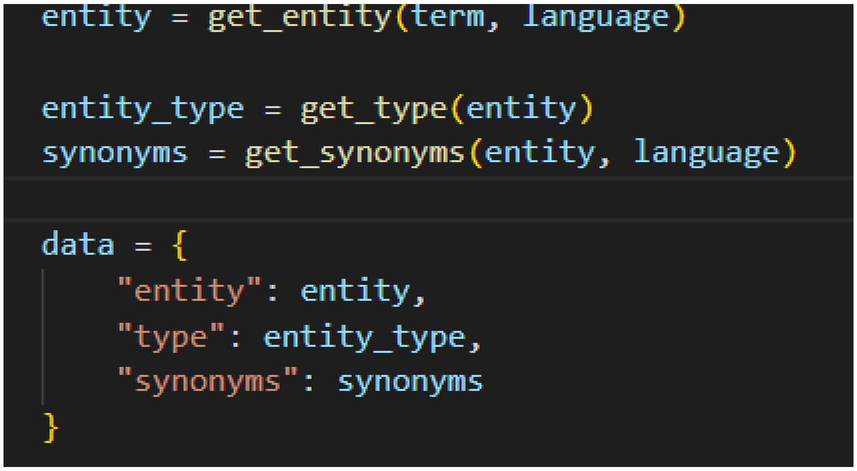
There is another method called get_entities() that calls the method from above for each trending topic and gets the result that we will parse.



```python
def main():

    # Customize here the search script
    index = "tweets-20090624-20090626-en_es-10percent-v4"
    trending_topic_amount = 50
    trending_topic_language = "en"
    metric = "jlh"

    responses = trending_topic_list(index, trending_topic_amount, trending_topic_language, metric)

    results = parse_responses(responses)

    entities = get_entities(results, trending_topic_language)

    save_results(entities, "output2.txt")
```

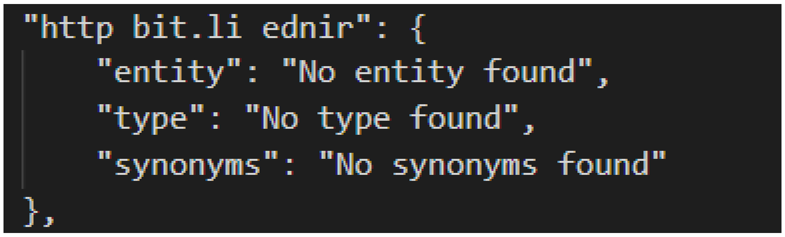
The script can be also configured from the main method.

We also have to find the type of the entity and the synonyms, so when we get the entity we execute these two methods and store it in the data variable that will be dumped in a txt.



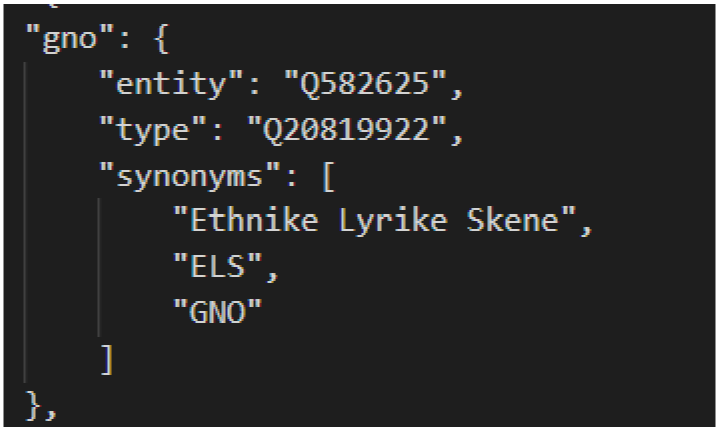
```
entity = get_entity(term, language)

entity_type = get_type(entity)
synonyms = get_synonyms(entity, language)

data = {
    "entity": entity,
    "type": entity_type,
    "synonyms": synonyms
}
```

Example of something that is not found:



```
"http bit.li ednir": {
    "entity": "No entity found",
    "type": "No type found",
    "synonyms": "No synonyms found"
},
```

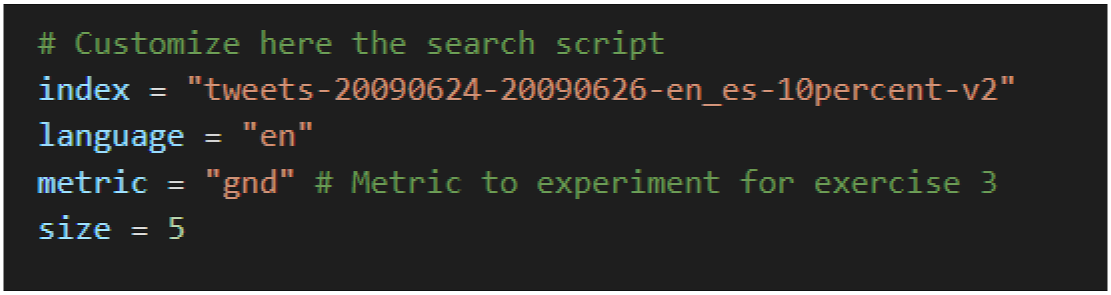
Example with synonyms



```
"gno": {
    "entity": "Q582625",
    "type": "Q20819922",
    "synonyms": [
        "Ethnike Lyrike Skene",
        "ELS",
        "GNO"
    ]
},
```

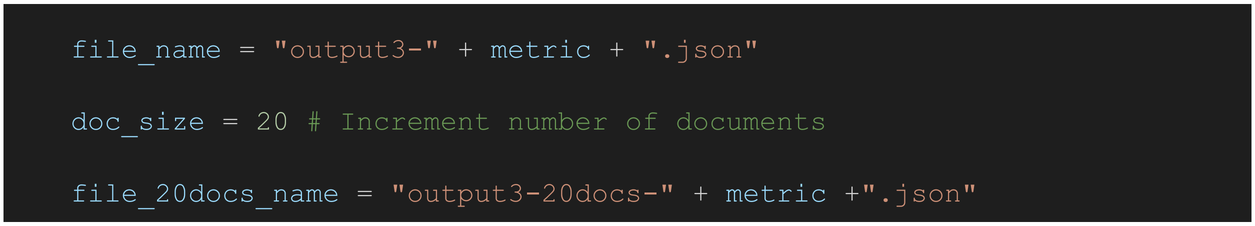
Exercise 3

Customizable part:



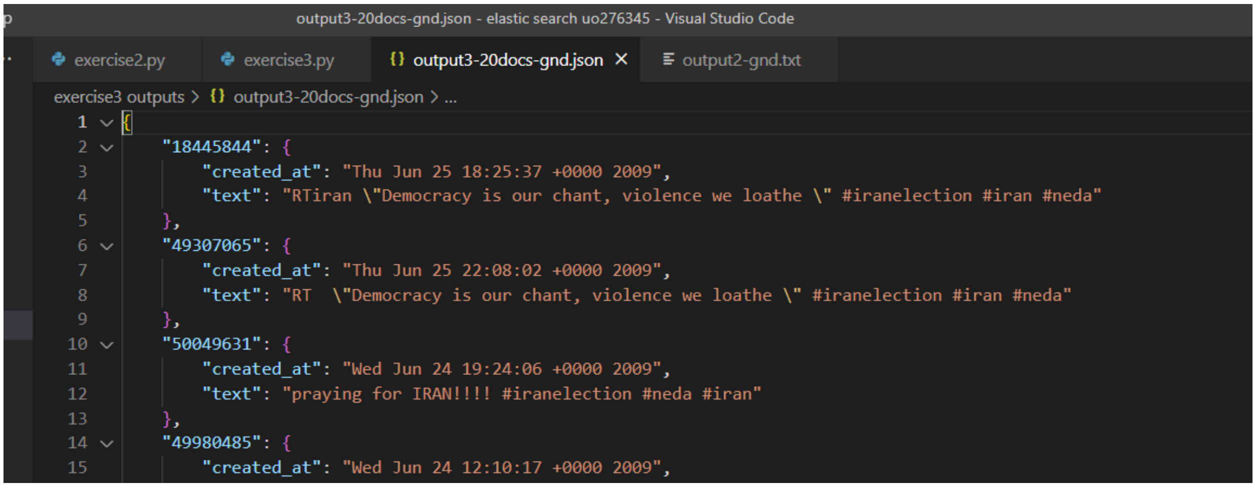
```
# Customize here the search script
index = "tweets-20090624-20090626-en_es-10percent-v2"
language = "en"
metric = "gnd" # Metric to experiment for exercise 3
size = 5
```

For this exercise we will use the index-v2. This is because we want to remove the stop words from the significant terms. The script is customizable, so you can change the parameter metric if you want to produce an output, there are two outputs for the execution



```
file_name = "output3-" + metric + ".json"

doc_size = 20 # Increment number of documents

file_20docs_name = "output3-20docs-" + metric +".json"
```

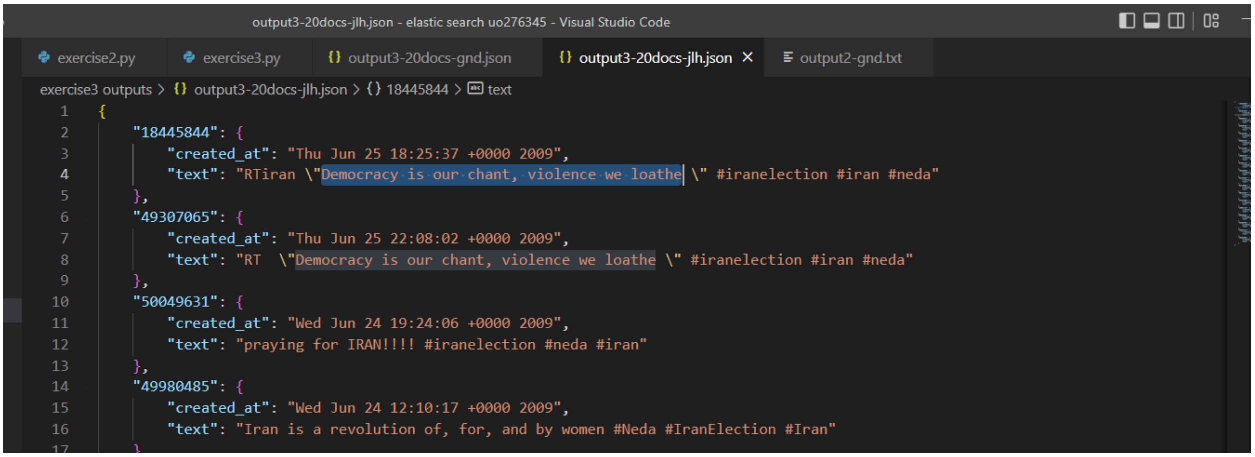
The file with the 20 docs is the one with the query expanded but limited to the doc size specified, these are the results for gnd and topic Iran.



output3-20docs-gnd.json - elastic search uo276345 - Visual Studio Code
exercise2.py
exercise3.py
output3-20docs-gnd.json
output2-gnd.txt
exercise3 outputs > output3-20docs-gnd.json > ...

```
{
    "18445844": {
        "created_at": "Thu Jun 25 18:25:37 +0000 2009",
        "text": "RTiran \"Democracy is our chant, violence we loathe \" #iranelection #iran #neda"
    },
    "49307065": {
        "created_at": "Thu Jun 25 22:08:02 +0000 2009",
        "text": "RT  \"Democracy is our chant, violence we loathe \" #iranelection #iran #neda"
    },
    "50049631": {
        "created_at": "Wed Jun 24 19:24:06 +0000 2009",
        "text": "praying for IRAN!!!! #iranelection #neda #iran"
    },
    "49980485": {
        "created_at": "Wed Jun 24 12:10:17 +0000 2009",
```

And for jlh and topic Iran.


output3-20docs-jlh.json - elastic search uo276345 - Visual Studio Code
exercise2.py
exercise3.py
output3-20docs-gnd.json
output3-20docs-jlh.json
output2-gnd.txt
exercise3 outputs > output3-20docs-jlh.json > 18445844 > text

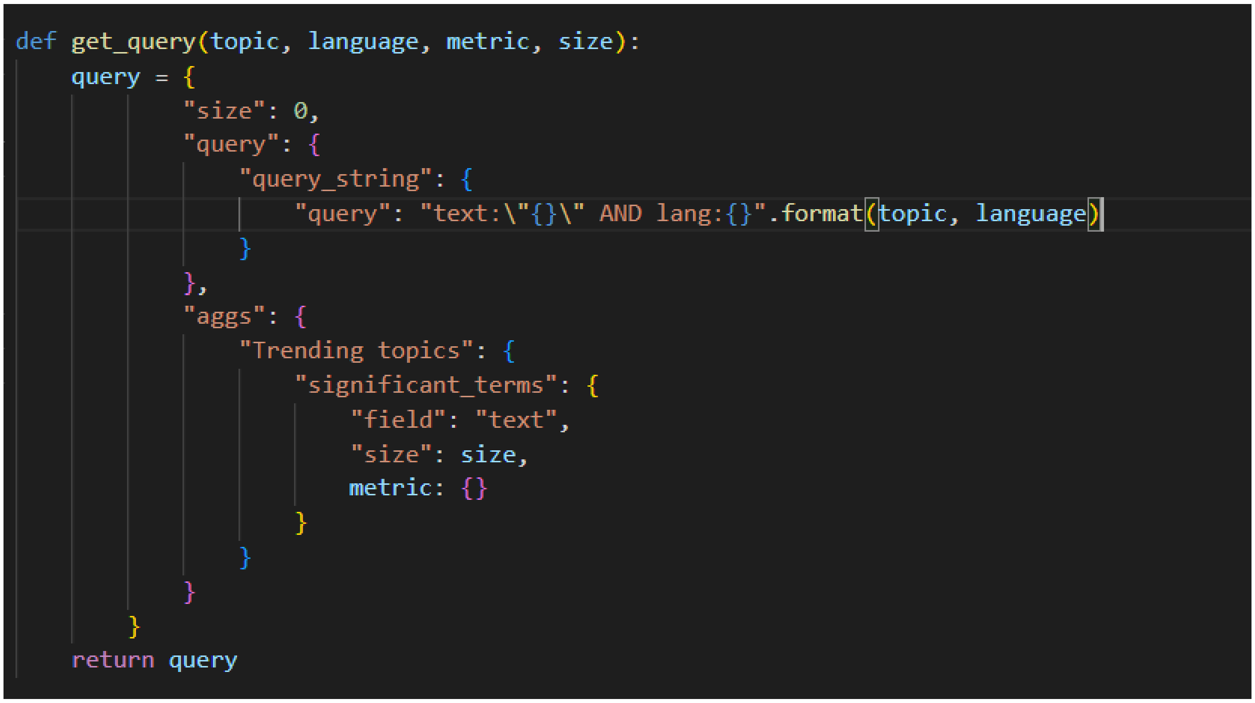
```
{
    "18445844": {
        "created_at": "Thu Jun 25 18:25:37 +0000 2009",
        "text": "RTiran \"Democracy is our chant, violence we loathe \" #iranelection #iran #neda"
    },
    "49307065": {
        "created_at": "Thu Jun 25 22:08:02 +0000 2009",
        "text": "RT  \"Democracy is our chant, violence we loathe \" #iranelection #iran #neda"
    },
    "50049631": {
        "created_at": "Wed Jun 24 19:24:06 +0000 2009",
        "text": "praying for IRAN!!!! #iranelection #neda #iran"
    },
    "49980485": {
        "created_at": "Wed Jun 24 12:10:17 +0000 2009",
        "text": "Iran is a revolution of, for, and by women #Neda #IranElection #Iran"
    }
```

They are the same despite changing the metric.

The exercise is based on using two queries and expanding them.

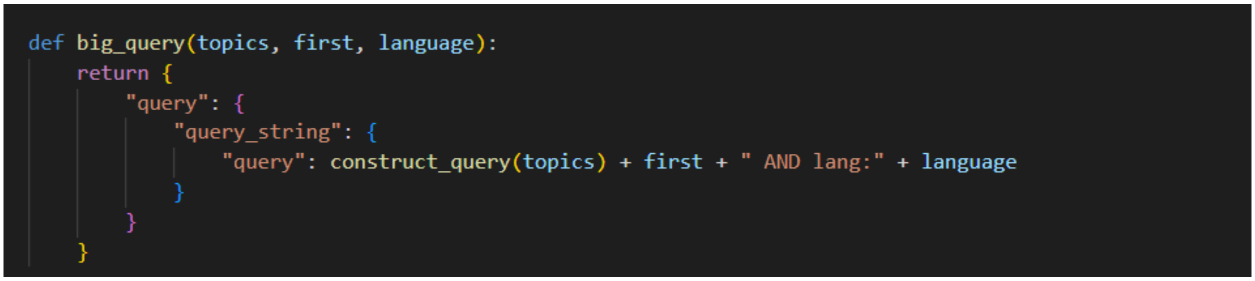
The first is for getting the results of the most significant terms related to the topic passed as a parameter. It’s used two times in order to expand the initial set of results; the first time in the main topic and the second time on the results of the first. The second query function gets the tweets that match with those significant terms.

Here is the initial query:



```
def get_query(topic, language, metric, size):
    query = {
            "size": 0,
            "query": {
                "query_string": {
                    "query": "text:\"{}\" AND lang:{}".format(topic, language)
                }
            },
            "aggs": {
                "Trending topics": {
                    "significant_terms": {
                        "field": "text",
                        "size": size,
                        metric: {}
                    }
                }
            }
        }
    return query
```

And here is the query for expanding the results of the previous one.

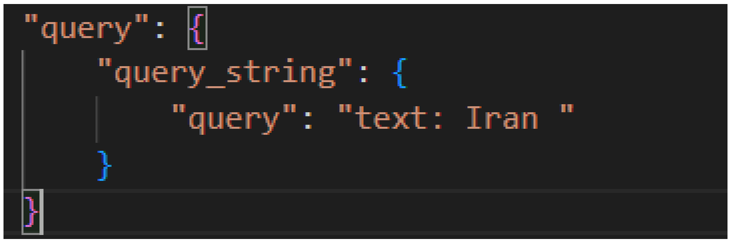


```
def big_query(topics, first, language):
    return {
        "query": {
            "query_string": {
                "query": construct_query(topics) + first + " AND lang:" + language
            }
        }
    }
```

Exercise 4

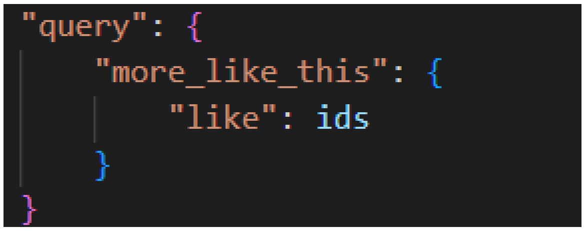
What we could do to emulate the expansion of an initial query would be this:

First, we would use an initial query to select all the documents that correspond to “Iran” for example (by using a query_string). Like this:



```
"query": {
    "query_string": {
        "query": "text: Iran "
    }
}
```

Then, we would choose the ids of the documents retrieved from this query (let's say the top 10, for example). With these _id fields we only have to execute a query of the type more_like_this to find documents similar to the ones found in the initial query :



```
"query": {
    "more_like_this": {
        "like": ids
    }
}
```