Information Retrieval

Assignment 2  
Report

Alex Ebbage

1504283

Contents

[1 Instructions 3](#_Toc509566476)

[1.1 Installations 3](#_Toc509566477)

[1.2 Setting Up Scrapy 3](#_Toc509566478)

[1.3 Setting Up Kibana 4](#_Toc509566479)

[2 Implementation and Design Decisions 5](#_Toc509566480)

[2.1 Indexing 5](#_Toc509566481)

[2.2 Searching 7](#_Toc509566482)

[2.3 Evaluation 8](#_Toc509566483)

[2.4 Engineering a Complete System 8](#_Toc509566484)

[3 Description of the Document Collection 8](#_Toc509566485)

[4 Description and Motivation of my Evaluation Methodology 9](#_Toc509566486)

[5 Ground Truth Data and Evaluation Results 10](#_Toc509566487)

[5.1 Query 1 – “Study abroad” 10](#_Toc509566488)

[5.2 Query 2 – “Campus tours” 11](#_Toc509566489)

[5.3 Query 3 – “Masters study computer science” 12](#_Toc509566490)

[5.4 Query 4 – “International student application form” 13](#_Toc509566491)

[5.5 Query 5 – “Undergraduate scholarships” 14](#_Toc509566492)

[5.6 Query 6 – “Postgraduate research degrees” 15](#_Toc509566493)

[5.7 Query 7 – “Parking” 16](#_Toc509566494)

[5.8 Query 8 – “Entry requirements” 17](#_Toc509566495)

[5.9 Query 9 – “Project open day” 18](#_Toc509566496)

[5.10 Query 10 – “Udo Kruschwitz” 19](#_Toc509566497)

[5.11 Summary of Results 20](#_Toc509566498)

[6 Discussion of my Solution, Improvements and Extensions 20](#_Toc509566499)

# 1 Instructions

## 1.1 Installations

First, install Elasticsearch 6.2.2 and Kibana 6.2.2 to your system by following the links below and downloading them from their sites:

**ElasticSearch:** <https://www.elastic.co/downloads/elasticsearch> **Kibana:** <https://www.elastic.co/downloads/kibana>

Next, install Anaconda for Python 2.7. Anaconda comes with an array of Python packages:

**Anaconda:** <https://www.anaconda.com/download/>

Open the Anaconda prompt and write the following commands in to download the required packages:

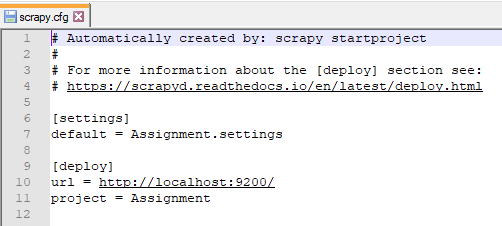
**conda install scrapy**  
**conda install elasticsearch  
conda install “ScrapyElasticSearch”**

## 1.2 Setting Up Scrapy

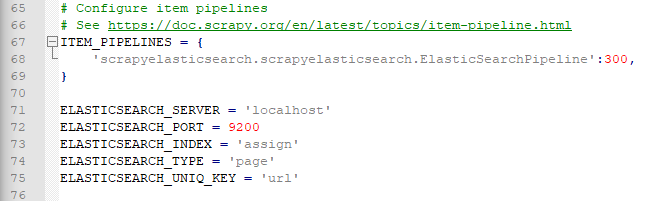
These are the Python packages which are required to start the crawler and push data to ElasticSearch. Next, create a new project with Scrapy using the following command:

**scrapy startproject <name> <directory>**

Where <name> and <directory> are left for you to choose. This will create a new Scrapy project and will prompt you to go to the newly created directory. Go there and open the *scrapy.cfg file*, modifying to look as follows:



Instead of Assignment, you will have what you named your project earlier. Save the changes then go into the folder with the name of your project and open *settings.py*. Scroll down to where it says ITEM\_PIPELINE, uncomment it by removing the # symbol and edit it too look as follows:



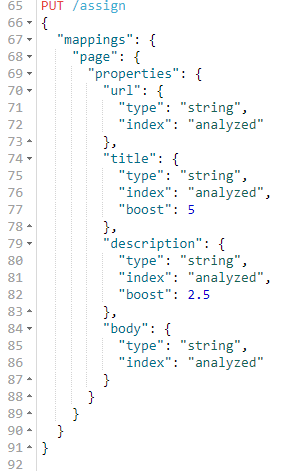
You may choose different names for the index, type and uniq\_key, however, you must keep port and server as shown. Save the changes, then go to the spiders folder and place the downloaded assignment files into the folder.

## 1.3 Setting Up Kibana

Now turn on the Kibana and Elasticsearch servers by going to where you installed them and running the batch files within their bin folders. Now go to a browser and navigate to:

**Localhost:5601**

This will take you to the Kibana site. Go to management > Index patterns and type in the name of the index you made earlier. Once complete, go to the Dev Tools and write the following code:



Hit the green arrow which appears to the side and proceed back to the Anaconda command prompt. Type in the following command:

**scrapy crawl caterpie**

This will start the crawler working. The process can take up to an hour depending on your computer but can also be stopped at any point; the only result of this is that fewer results will be passed to Elasticsearch. Once the program has completed or you have stopped it, return to the Kibana site and go to Discover. This is where you can query your newly indexed webpages.



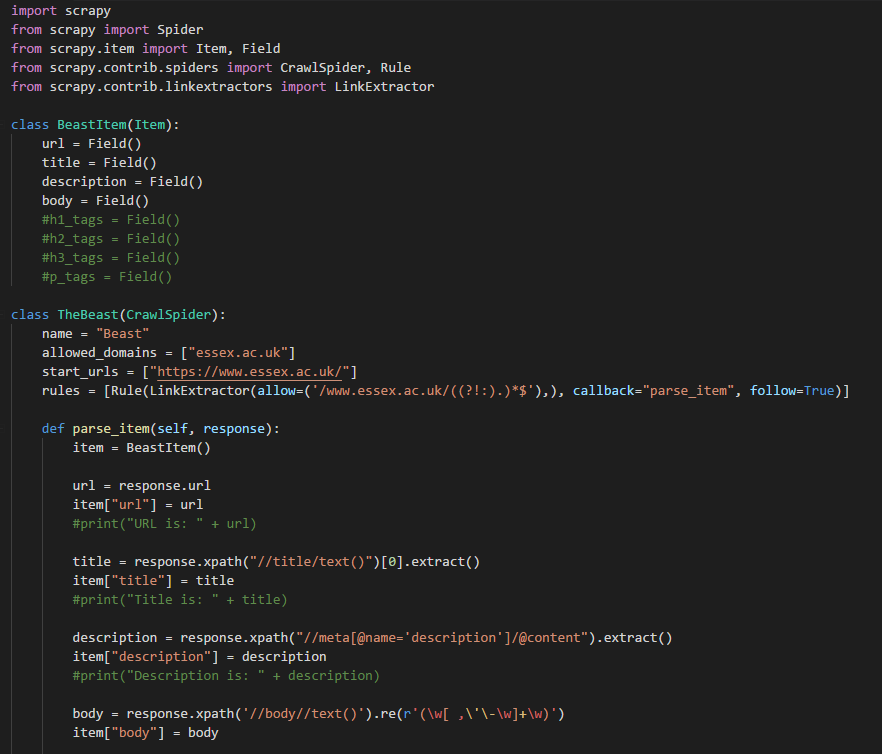
Following this, different fields can be selected to be view on the left-hand side of the screenshot above and queries can be experimented with at the top of the page.

# 2 Implementation and Design Decisions

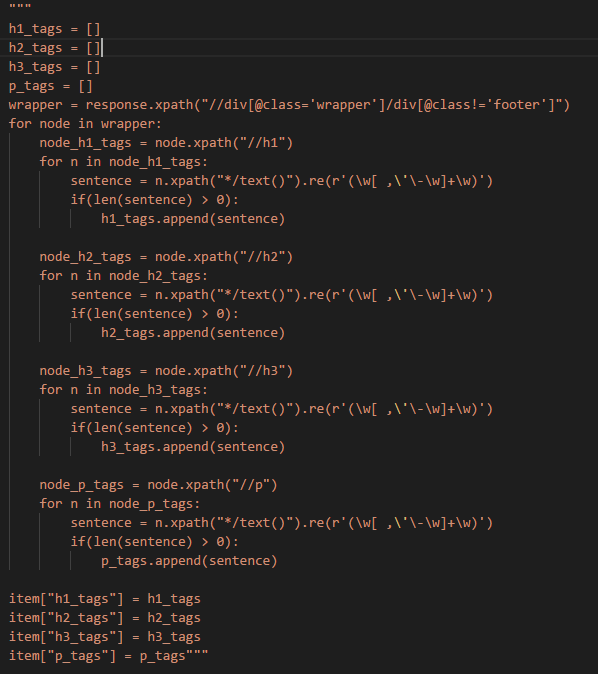
## 2.1 Indexing

When it came to identifying a document collection of my choice to index, I had the option of choosing between simply taking one of the sample datasets from the Elasticsearch tutorial in the JSon format or to collect the data myself by indexing a website. This seemed like the more interesting choice as it reflected real life sceanarios more. I began by looking into methods of web scraping and initially decided to use Beautiful soup in combination with another program to pass the data over to Elasticsearch. In the end however, I used the Python Scrappy framework, as it provides a lot of support regarding crawlers. There were other options but I felt like sticking with Scrappy and learning how to use crawlers as I found the concept of them interesting.

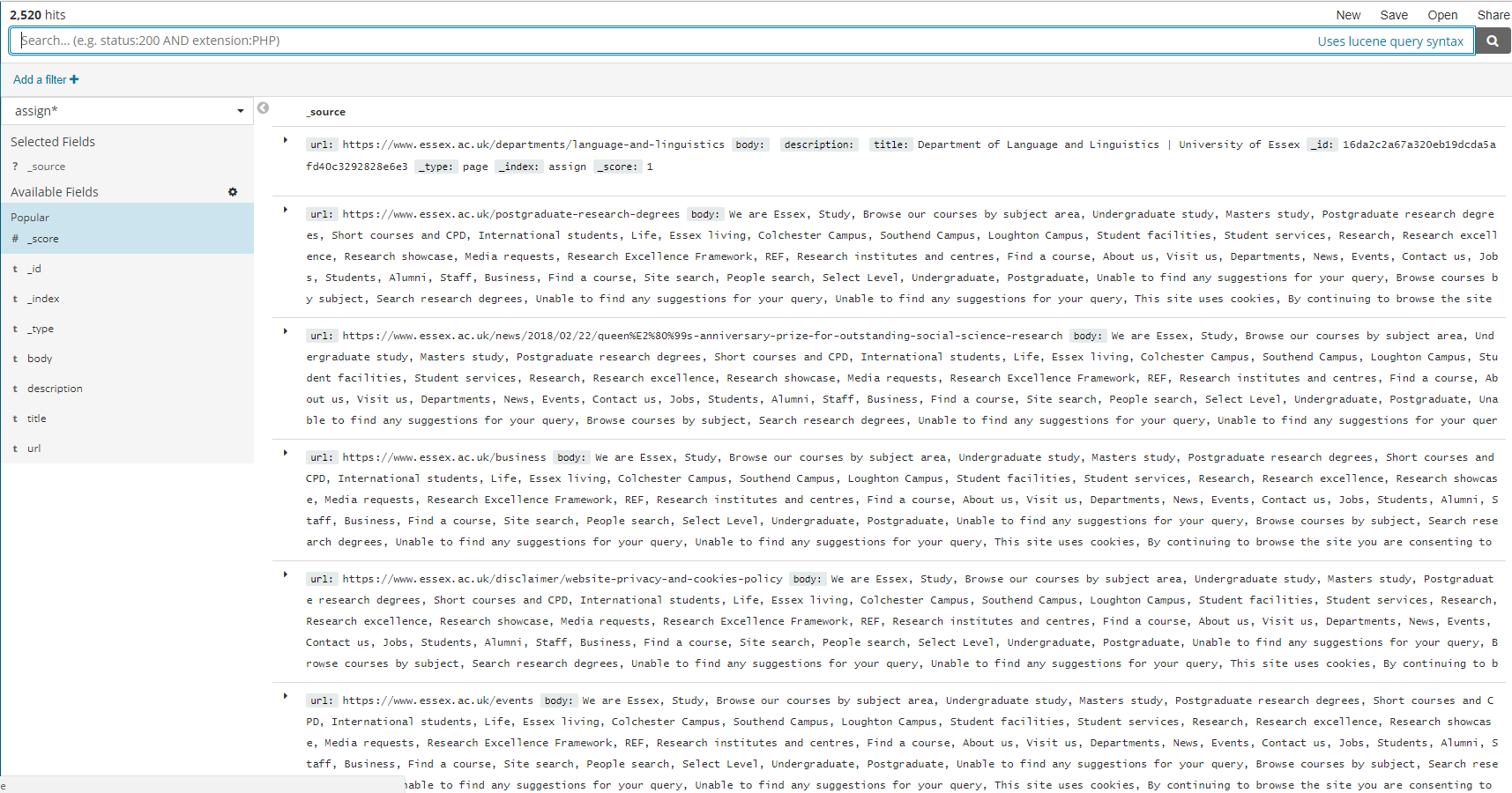
For the indexing, I used Scrapy to create a crawler which used XPath to perform web scraping. I stored the data into separate fields then which would later be used to weight the query findings. I struggled with getting Scrapy to work correctly for a long time and so didn’t reach the standard of implementation I had intended, but I believe it still performed well based on the results I attained.



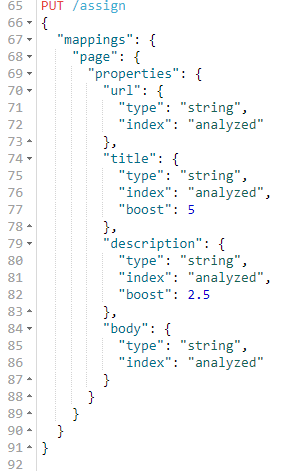
The rule in the above code uses a regular expression to extract links from the page it is on. If the expression returns a result and it is within the allowed domain, then it explores the webpage through the link and repeats the link extraction process. Each webpage that is visited by the crawler gets the URL, title, description and body of the page, places it into an item and sends it to Elasticsearch. I experimented with the code data I collected a lot, trying to produce more useful data, such as separating body into h1, h2, h3 and p tags, but it produced unusual results because of hidden content within the webpages which would produce a lot of useless information that would have an effect on the results of Kibana.



## 2.2 Searching

I used Kibana for searching as I find it to be an intuitive and powerful tool. The services it provides make it easy to do organize how the data is indexed and searched. 

## 2.3 Evaluation

The only modifications I made that would have an effect on the evaluation and querying is that I used the boost property in the index to weight the values. I gave the title the greatest boost as I found that with the document collection I used that they typically gave a good insight to the page, then I gave the descriptions a moderate boost as they weren’t on every page but they still provided a good amount of insight into the page’s contents.  


## 2.4 Engineering a Complete System

With the files setup correctly, the crawler can be run with a single command and the whole of the essex.ac.uk domain will be indexed and put into elasticsearch.



# 3 Description of the Document Collection

The documents collection I selected was one I created by performing crawling and web scraping on the Essex.ac.uk webpages. Each document is indexed with 4 additional properties: URL, title, description and body. URL is simply the URL of the given page, title is the text held between the <title> tags on a webpage, description is the contents of the meta tag with an id of “description” and body is an array of strings which have had a large amount of junk removed from them, i.e. large amounts of whitespace and newline characters. I would have liked to have refined the amount of data passed to Elasticsearch and to perform some interesting changes to the crawler but unfortunately ran out of time.

# 4 Description and Motivation of my Evaluation Methodology

The two methods of evaluation I chose to use and compare are Boolean and TF.IDF.

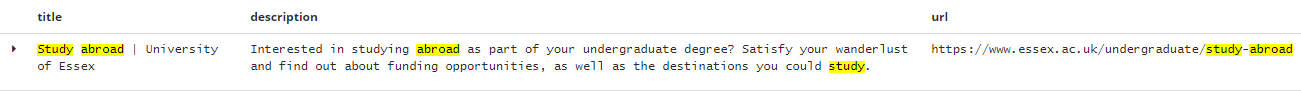
Boolean works by assuming both the documents to be searched and the user’s queries are sets of terms, producing a higher value if the documents contain the query terms.

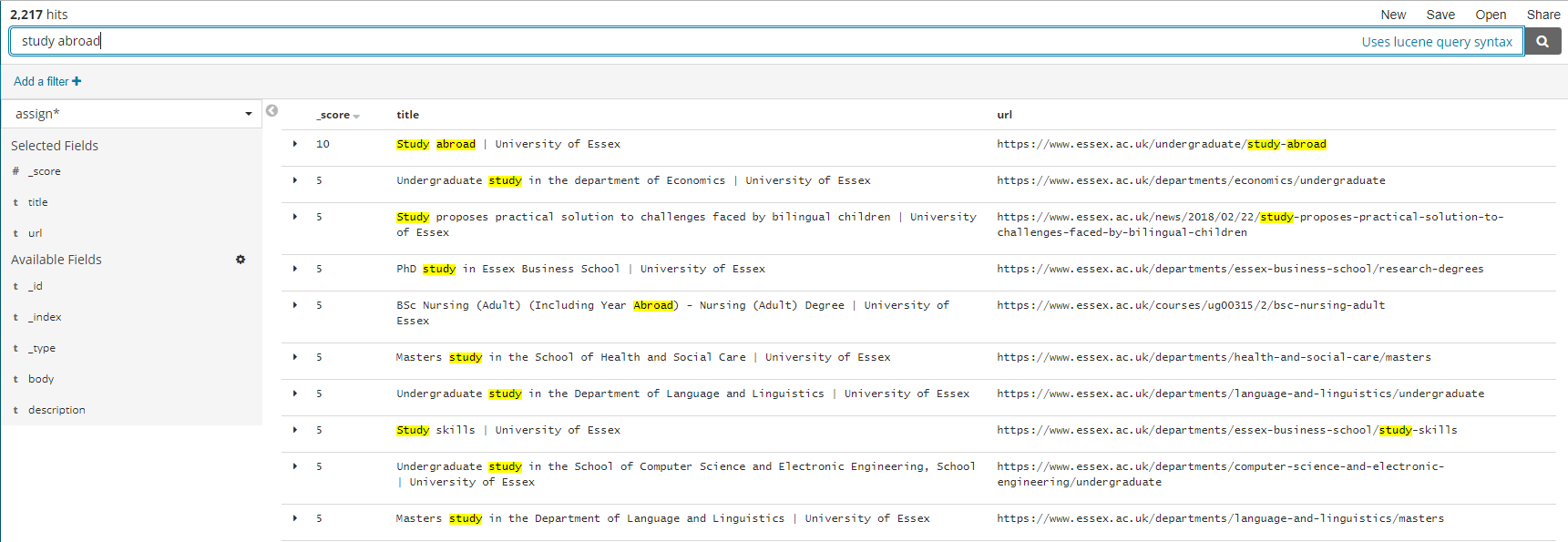
TF.IDF is supposed to reflect how important a word is to a document. The value produced increases proportionally to the number of times a word appears in the document and is offset by the words frequency in the corpus.

I chose these two because TF.IDF is one of the most popular term-weighting schemes and Boolean is one of the original IR models and I wanted to see how they compared.

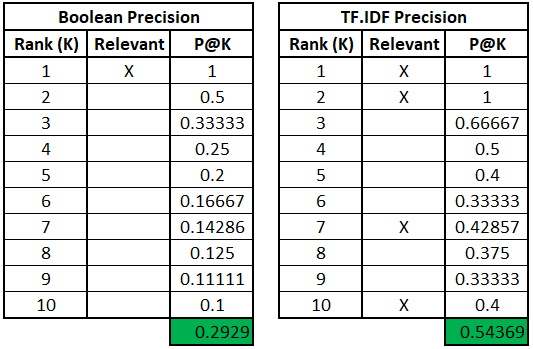
I’m using MAP (mean average precision) as a metric to determine how effective my IR system is. It uses a set of queries to find the average precision of the system.

# 5 Ground Truth Data and Evaluation Results

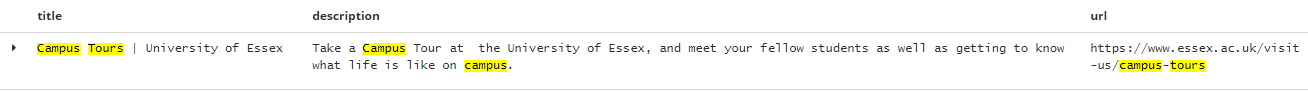
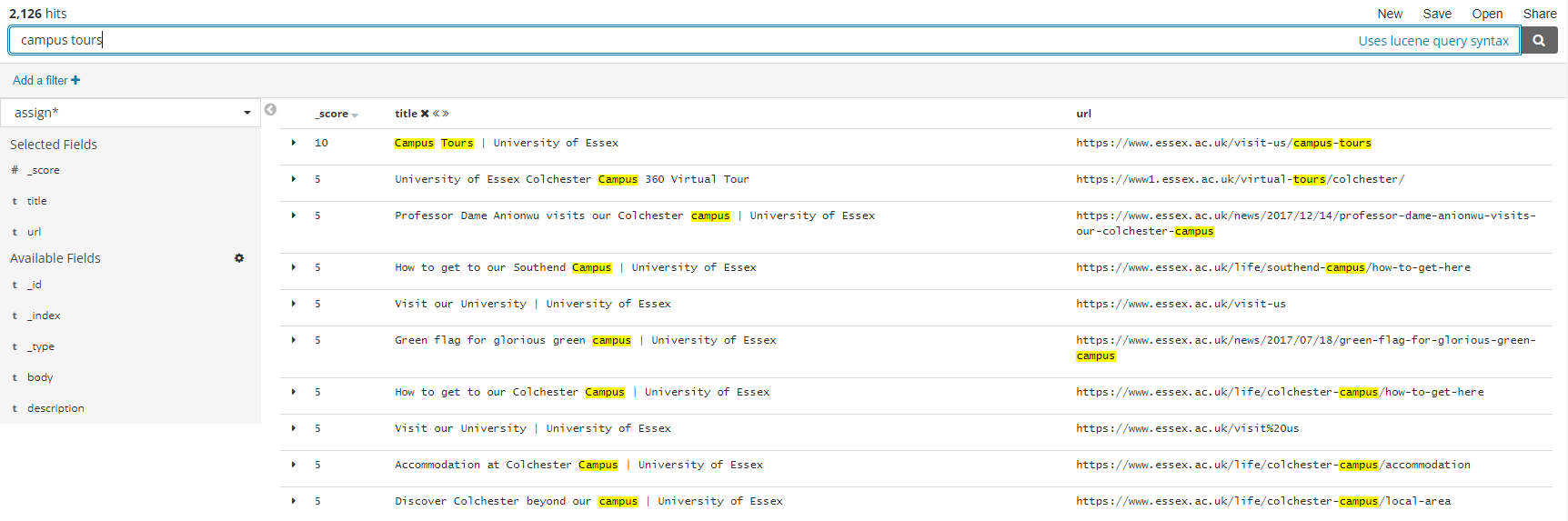
5.1 Query 1 – “Study abroad”  
**Expected result:** 

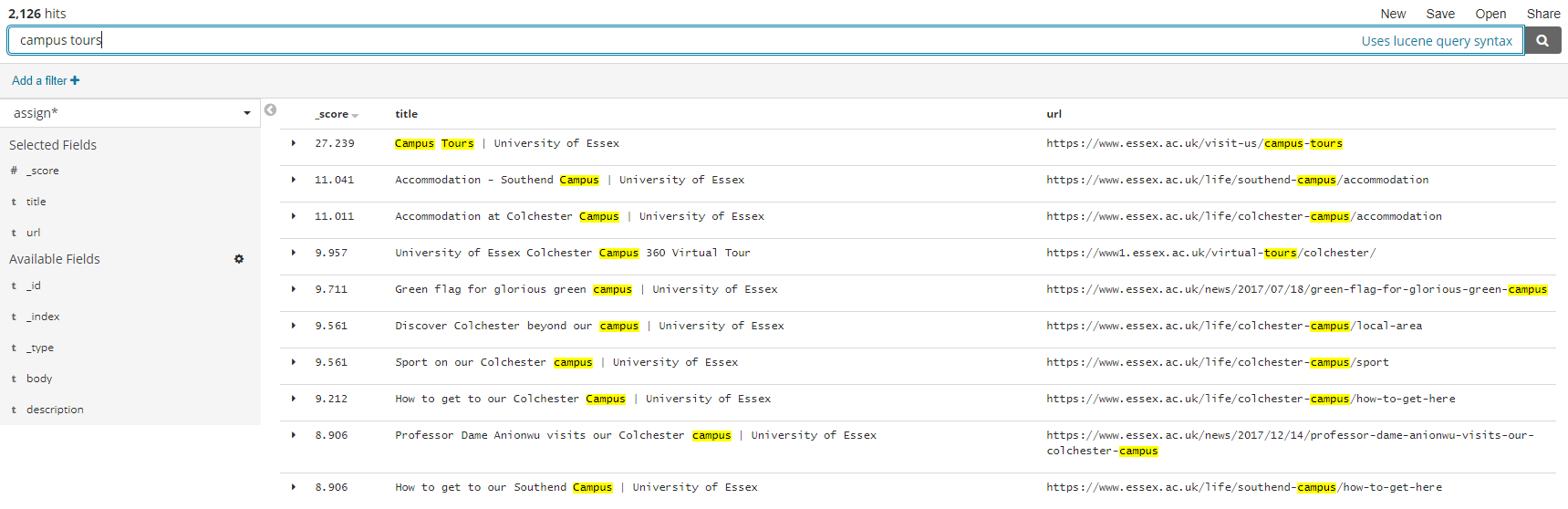
**Boolean result:** 

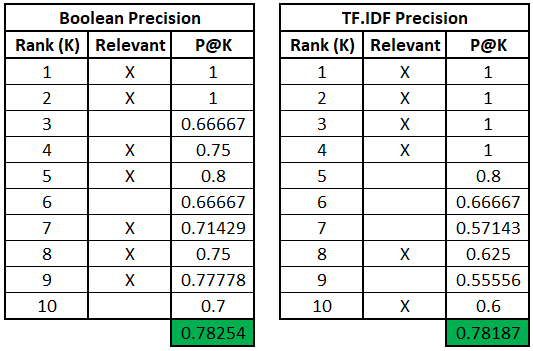
**TF.IDF result:**

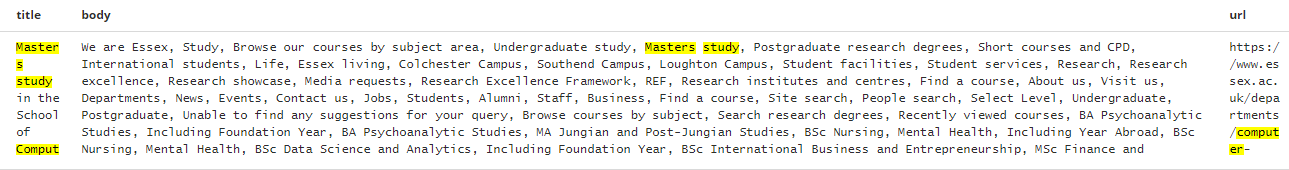
In this query, TF.IDF had a much higher precision than Boolean, however, both had the same first results which matched the expected result.

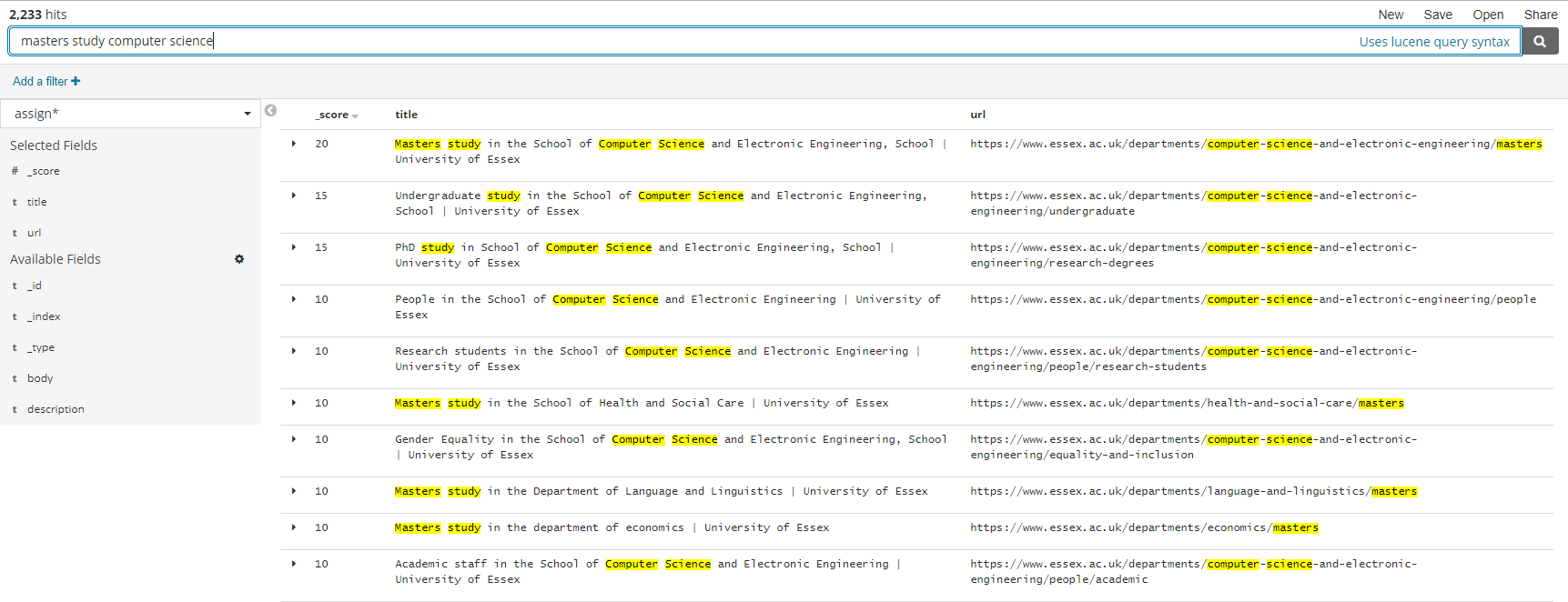
5.2 Query 2 – “Campus tours”  
**Expected result:**  
  
**Boolean result:** 

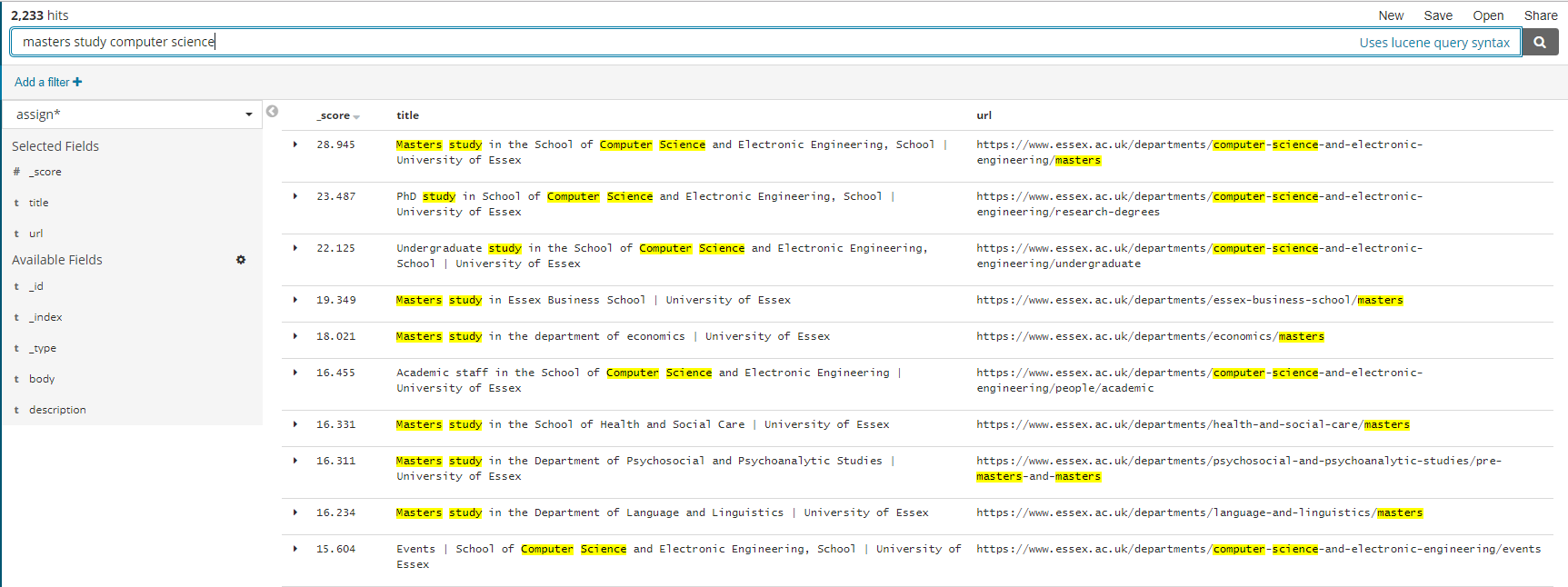
**TF.IDF result:** 

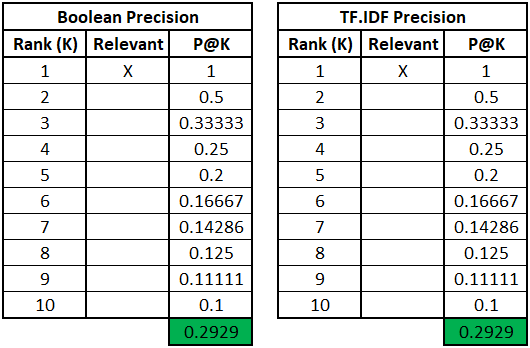


In this query, Boolean had a slightly higher precision than TF.IDF, however, both had the same first results which matched the expected result.

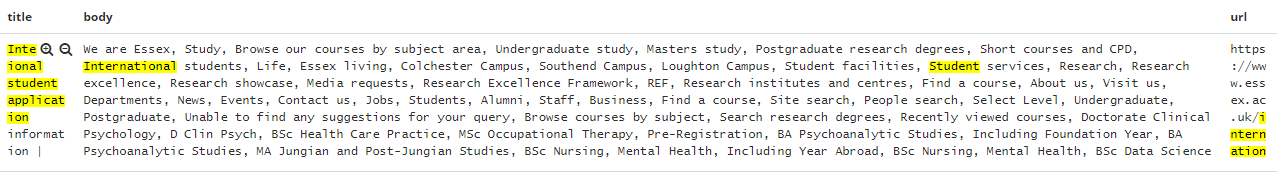
5.3 Query 3 – “Masters study computer science”  
**Expected result:** 

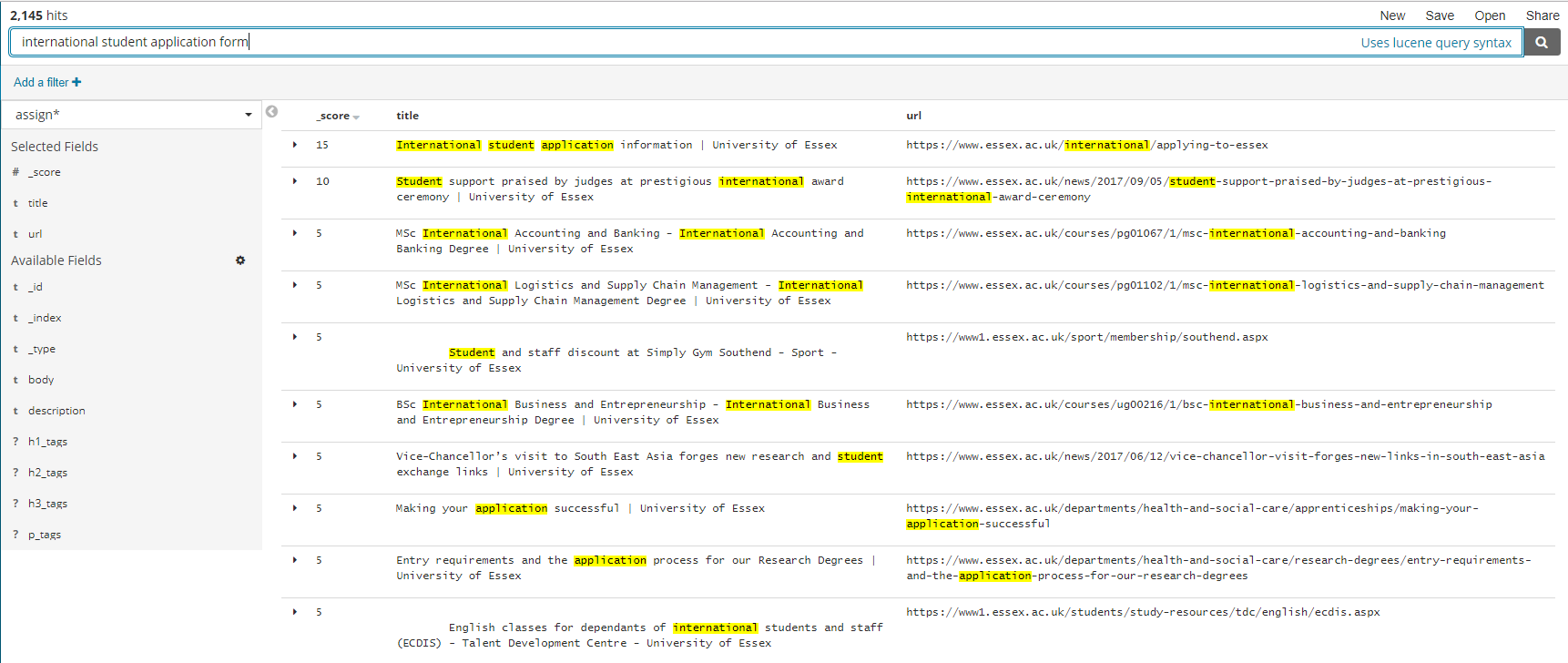
**Boolean result:** 

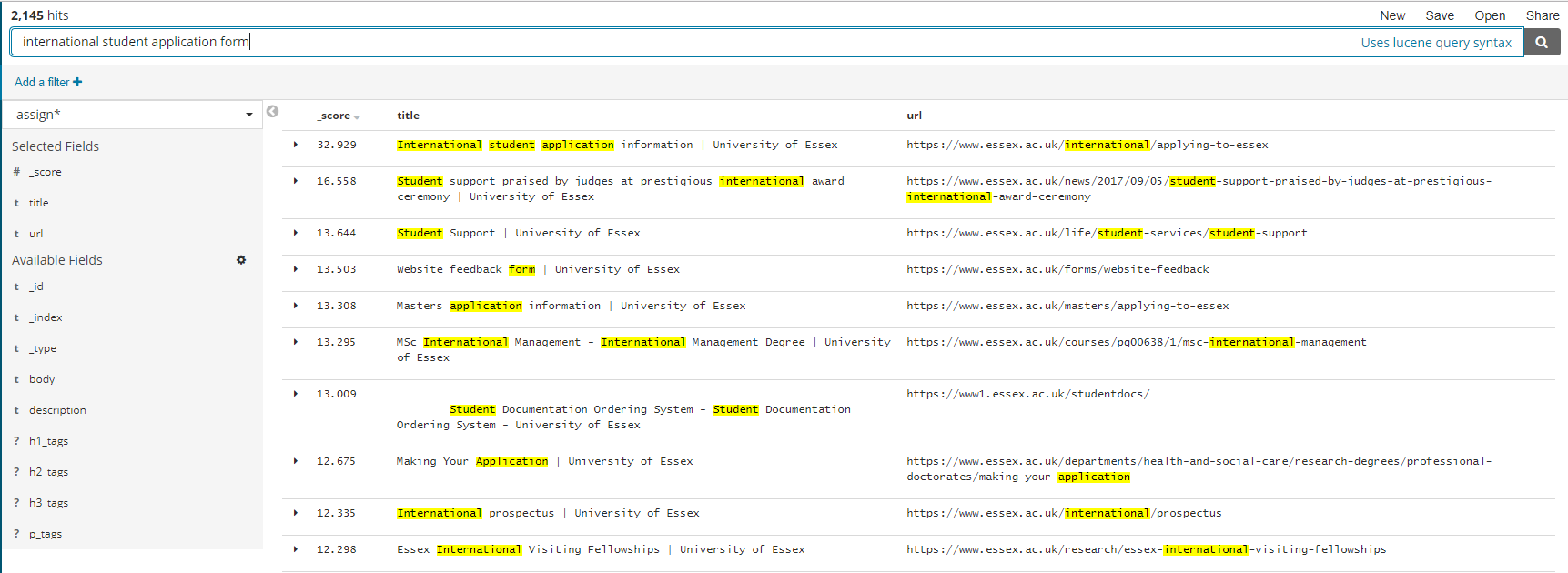
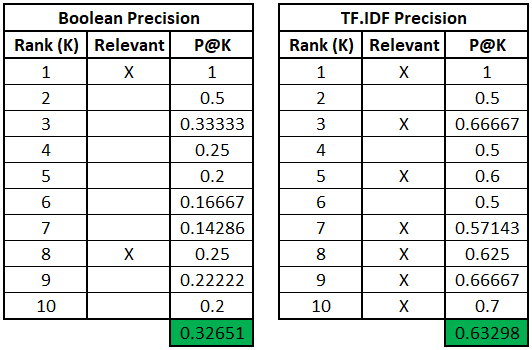
**TF.IDF result:** 



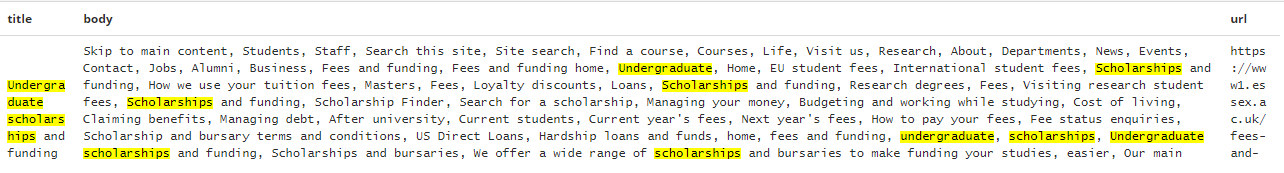
In this query, TF.IDF and Boolean performed the same, both having the same first results which matched the expected result.

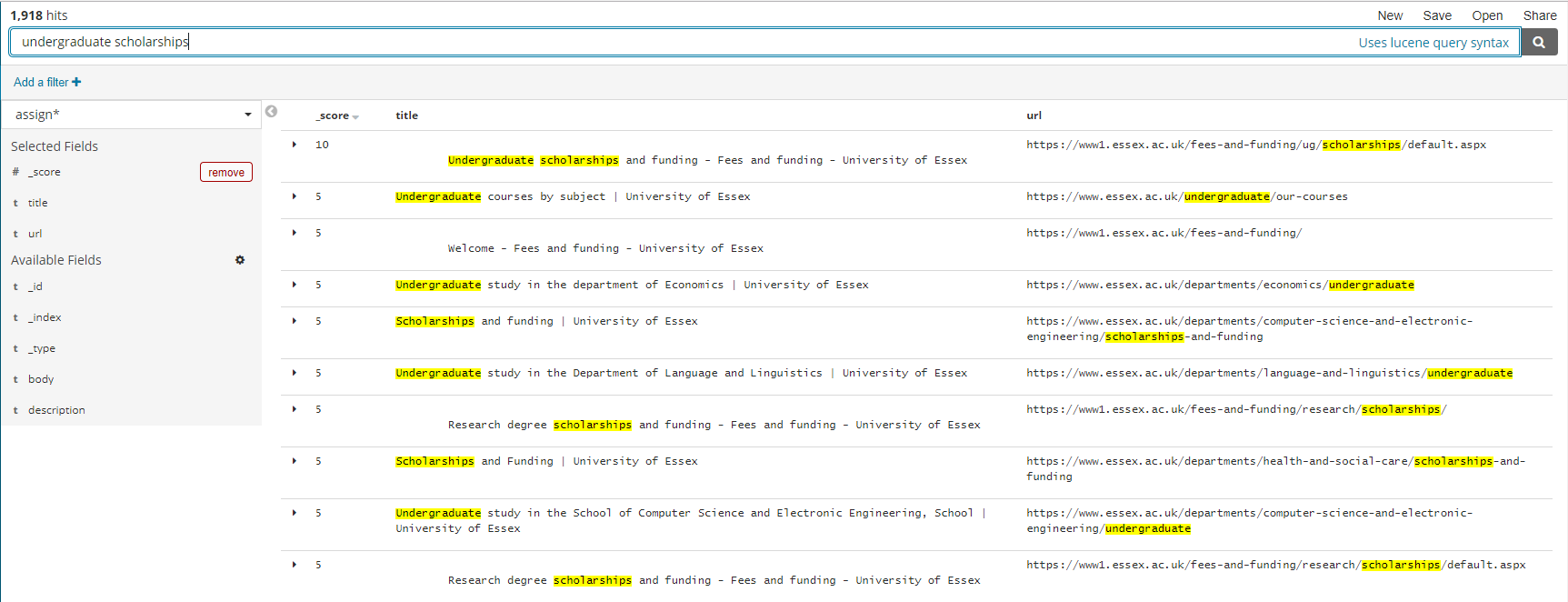
5.4 Query 4 – “International student application form”  
**Expected result:** 

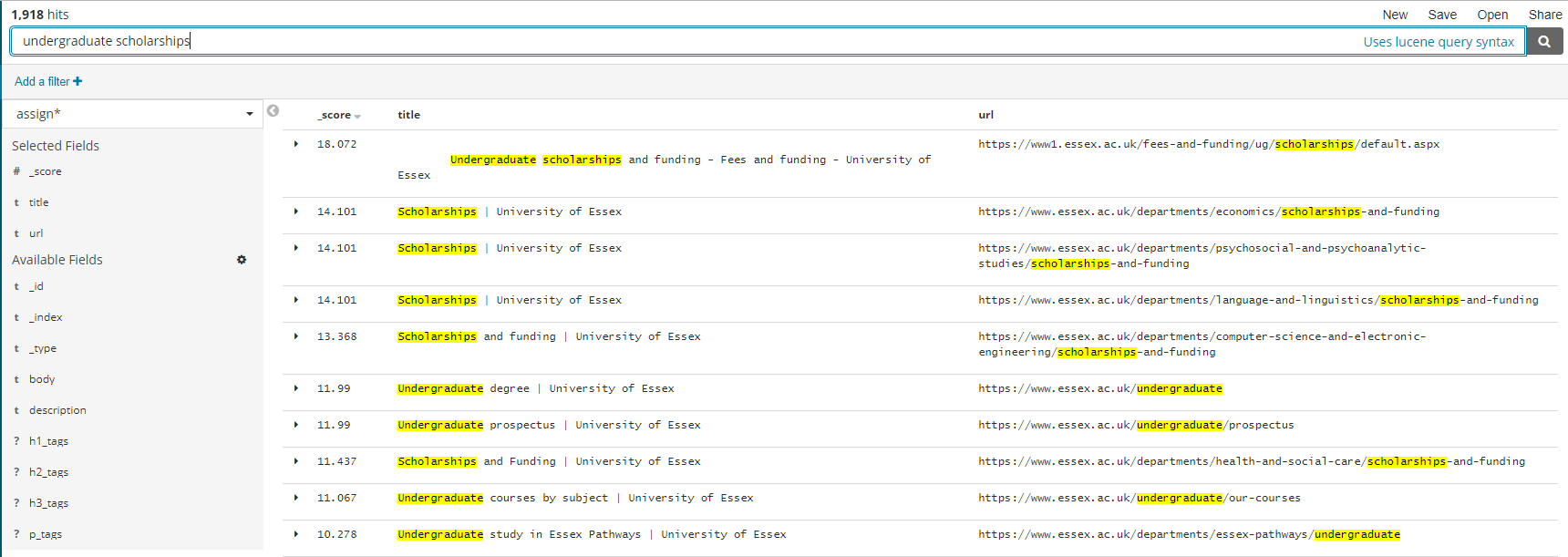
**Boolean result:** 

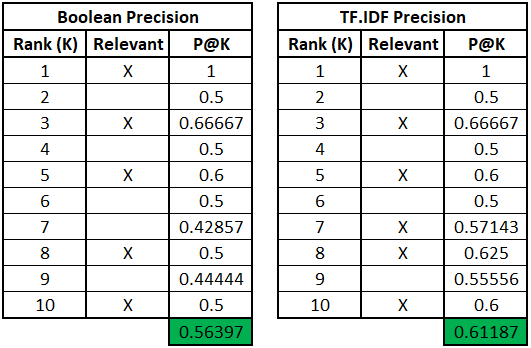
**TF.IDF result:**   


In this query, TF.IDF had a much higher precision than Boolean, however, both had the same first results which matched the expected result.

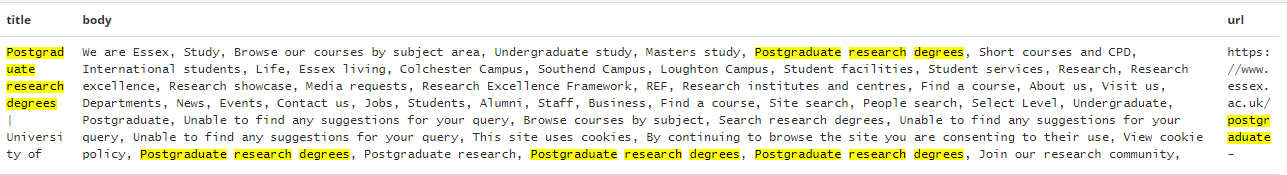
5.5 Query 5 – “Undergraduate scholarships”  
**Expected result:** 

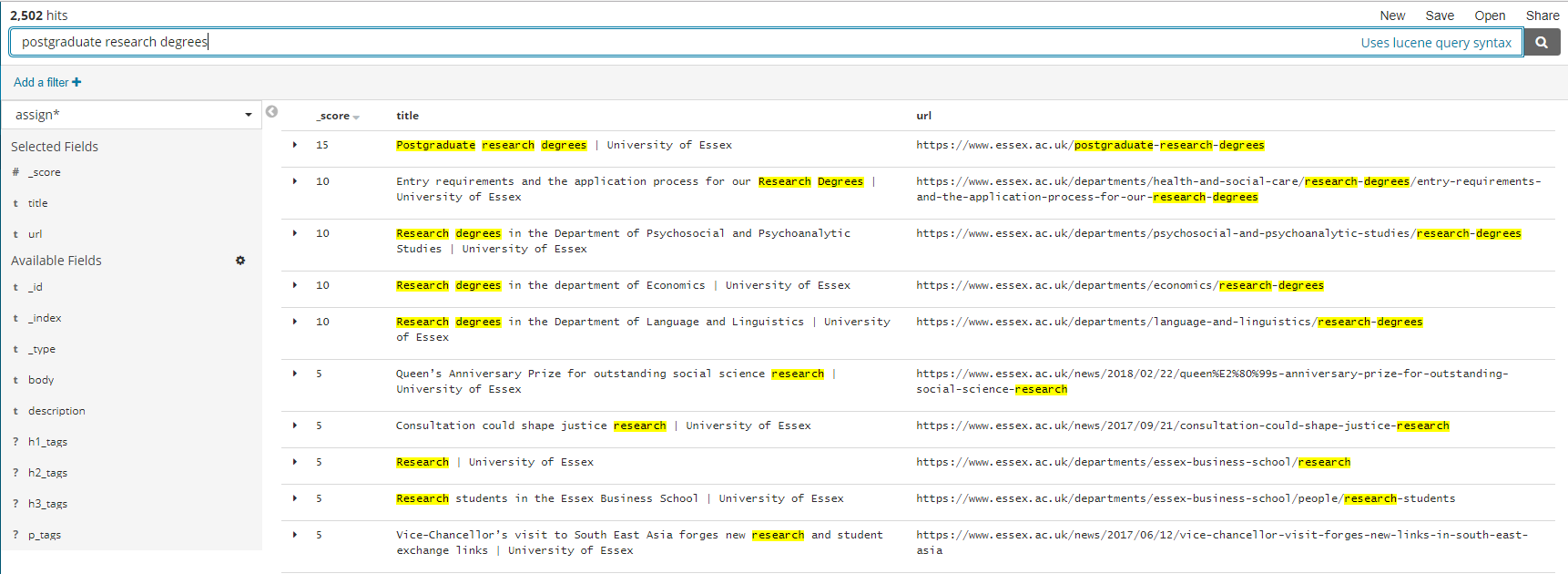
**Boolean result:** 

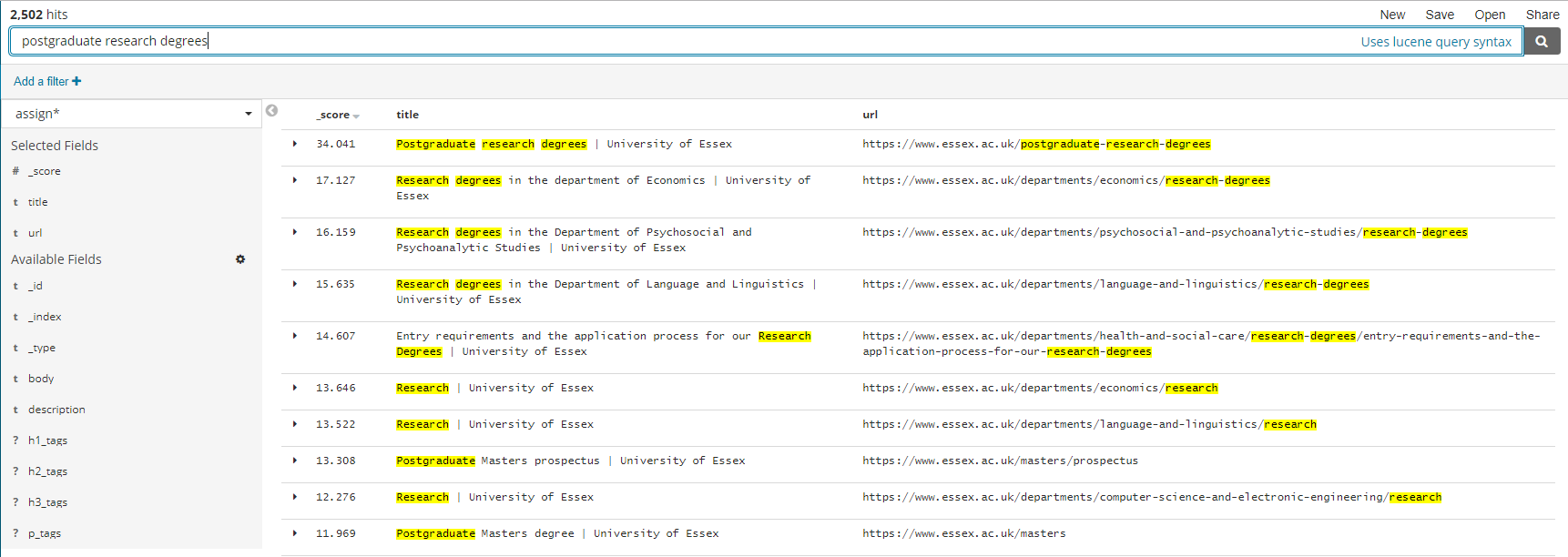
**TF.IDF result:** 

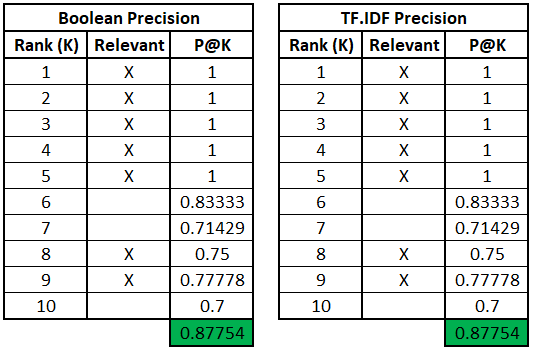


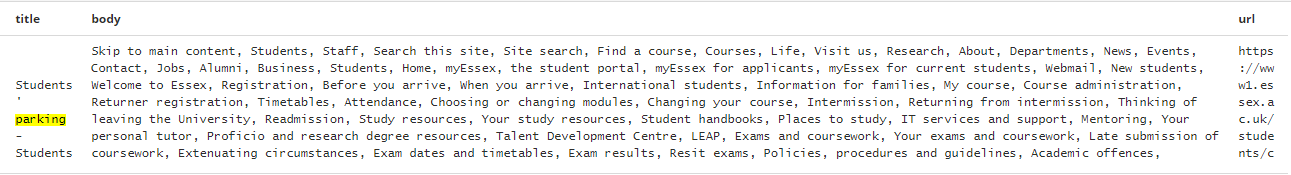
In this query, TF.IDF had a slightly higher precision than Boolean, however, both had the same first results which matched the expected result.

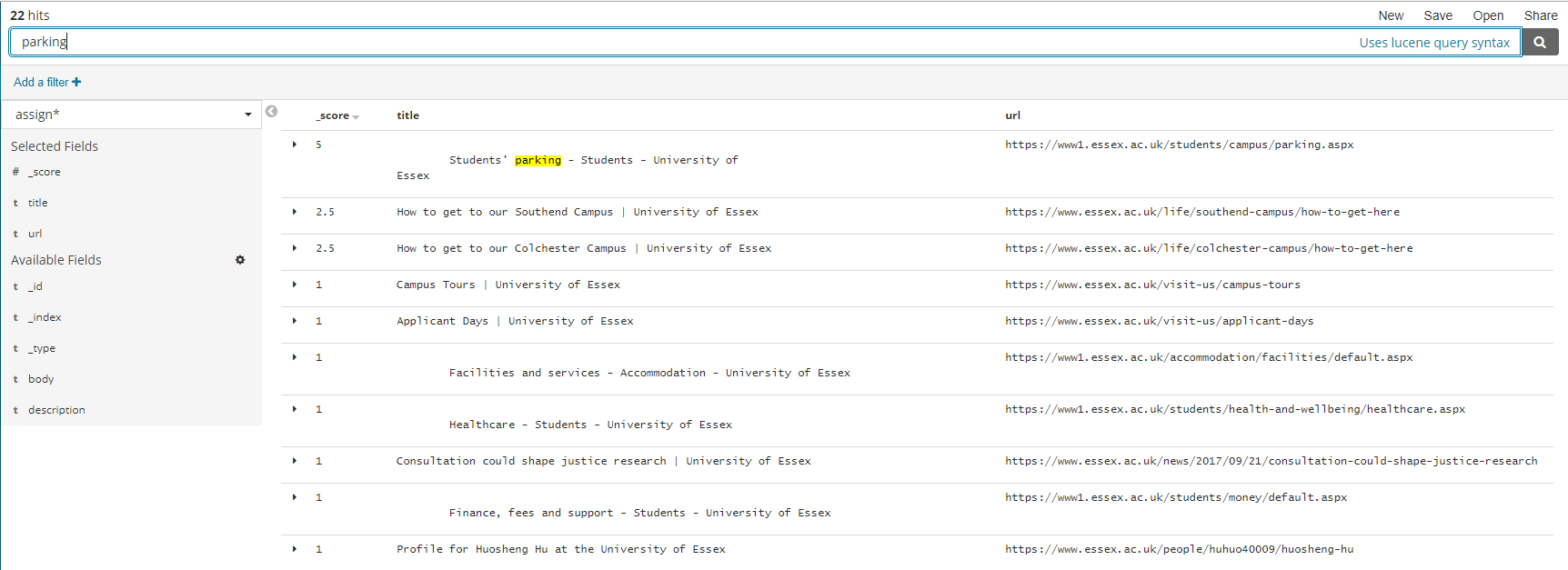
5.6 Query 6 – “Postgraduate research degrees”  
**Expected result:** 

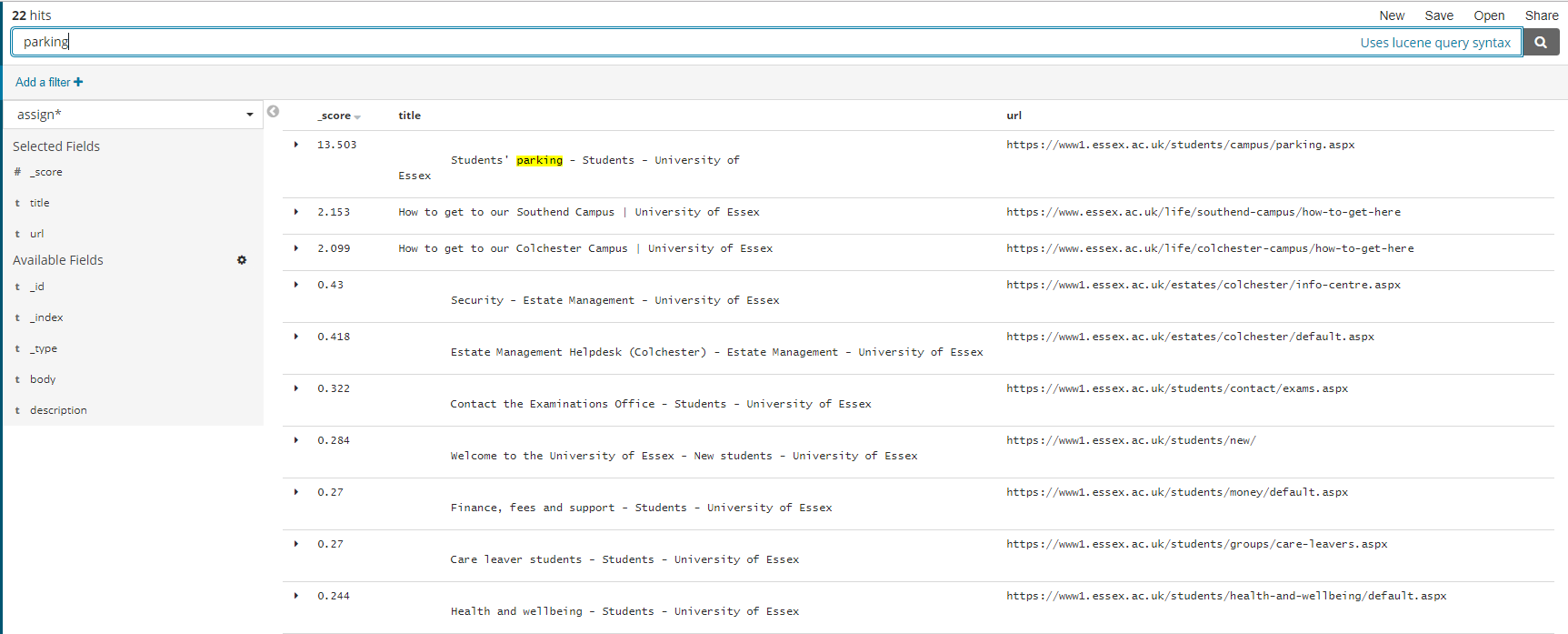
**Boolean result:** 

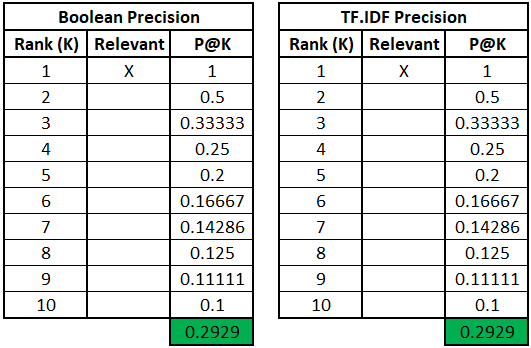
**TF.IDF result:** 

  
In this query, TF.IDF and Boolean achieved a high precision, both also had the same first results which matched the expected result.

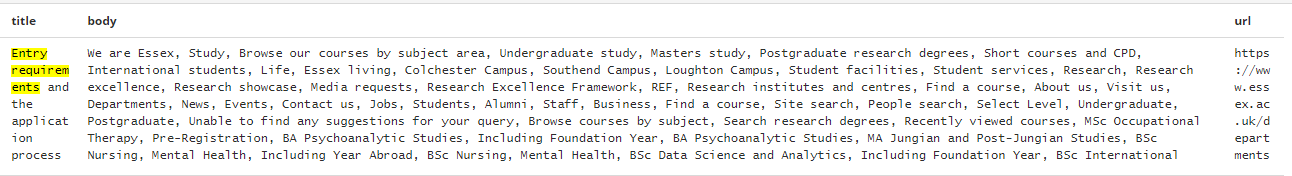
5.7 Query 7 – “Parking”  
**Expected result:** 

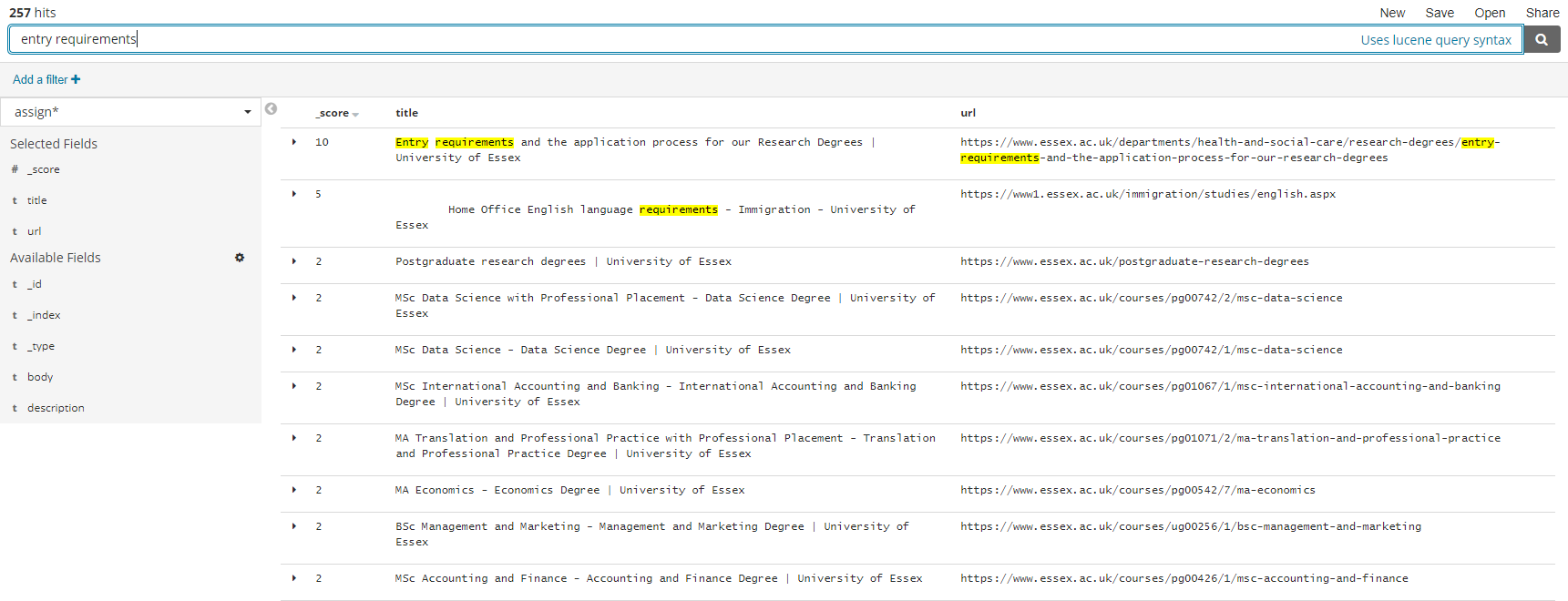
**Boolean result:** 

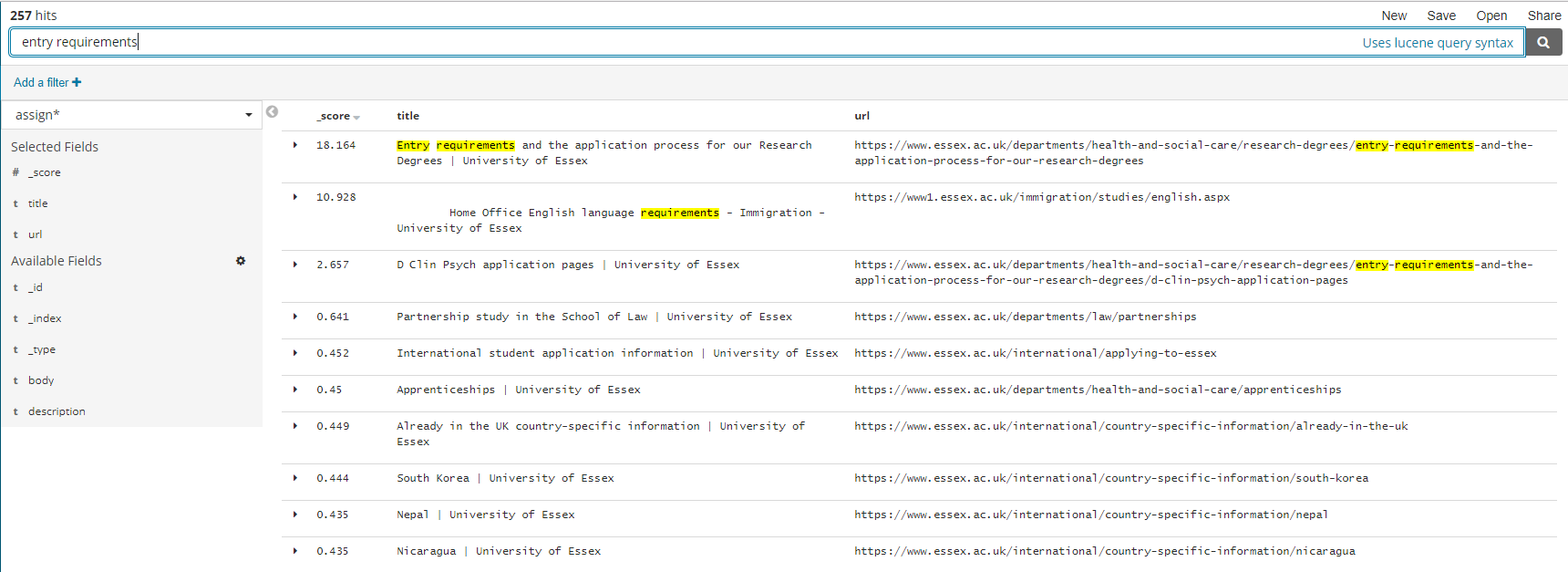
**TF.IDF result:** 

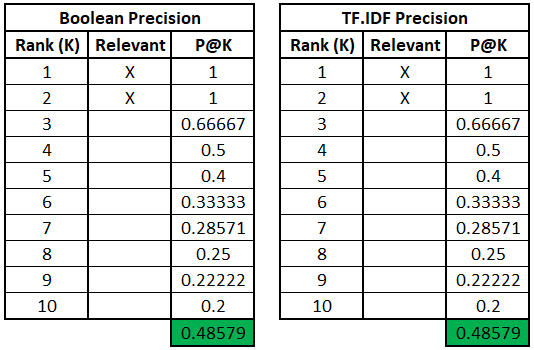


In this query, TF.IDF and Boolean had the same precision, both matching the expected result as their first and only relevant result.

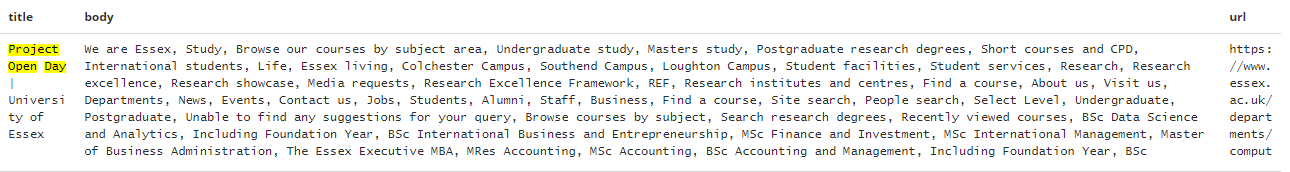
5.8 Query 8 – “Entry requirements”  
**Expected result:** 

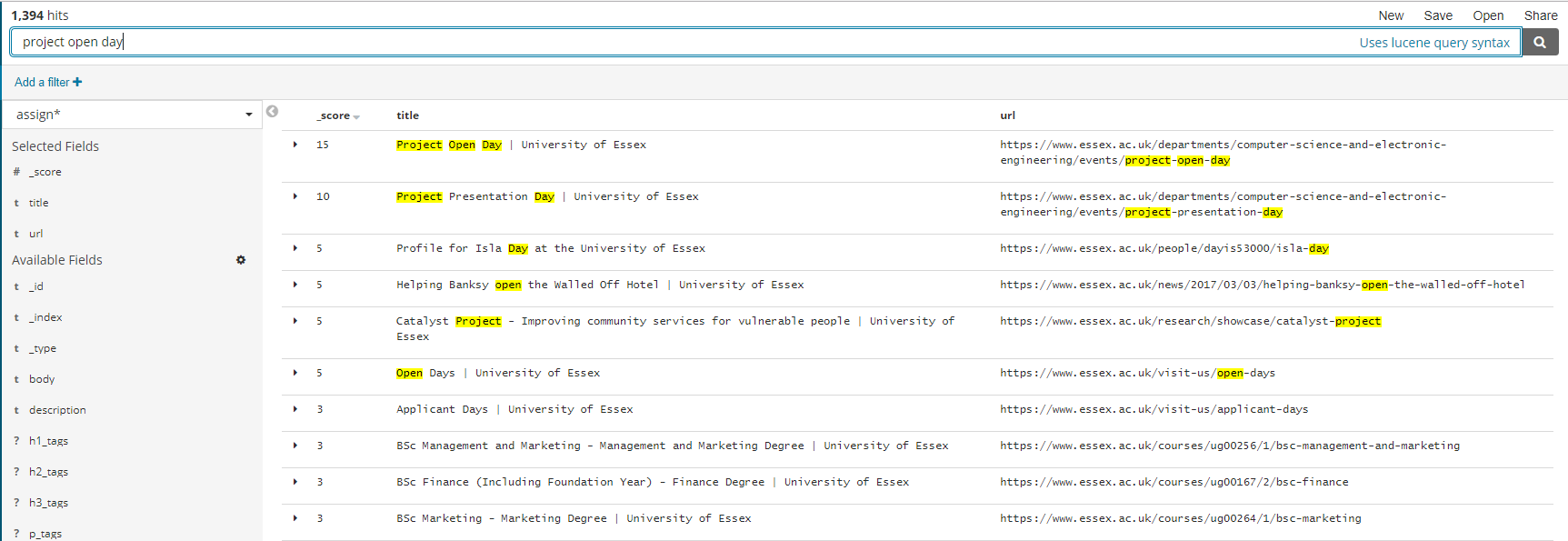
**Boolean result:** 

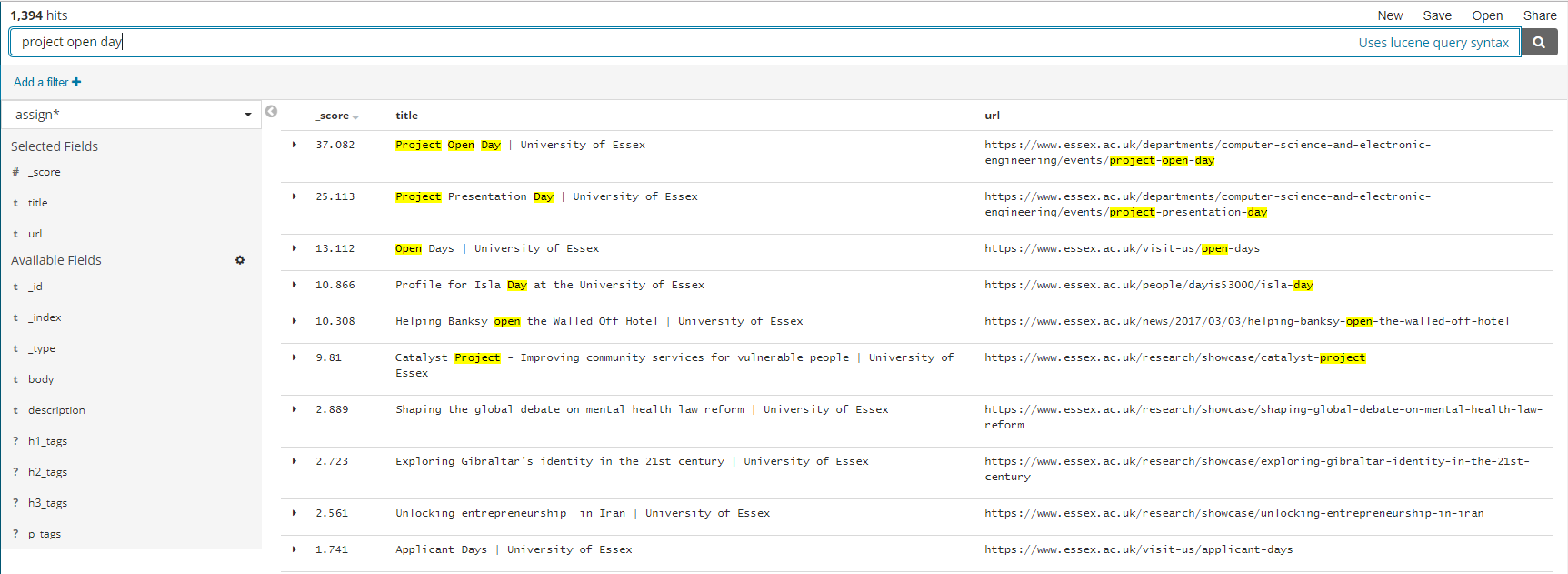
**TF.IDF result:** 

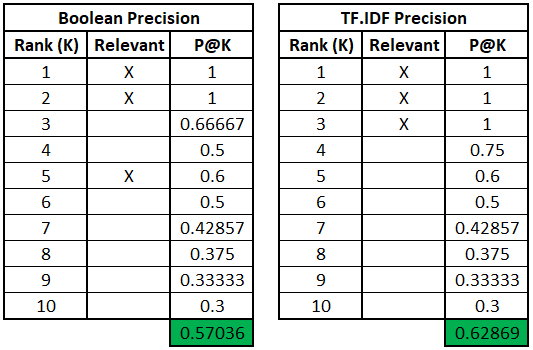


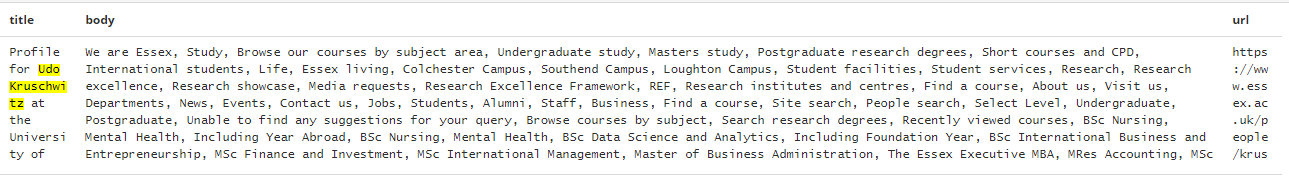
In this query, TF.IDF and Boolean had the same precision, both matching the expected result with their first results result.

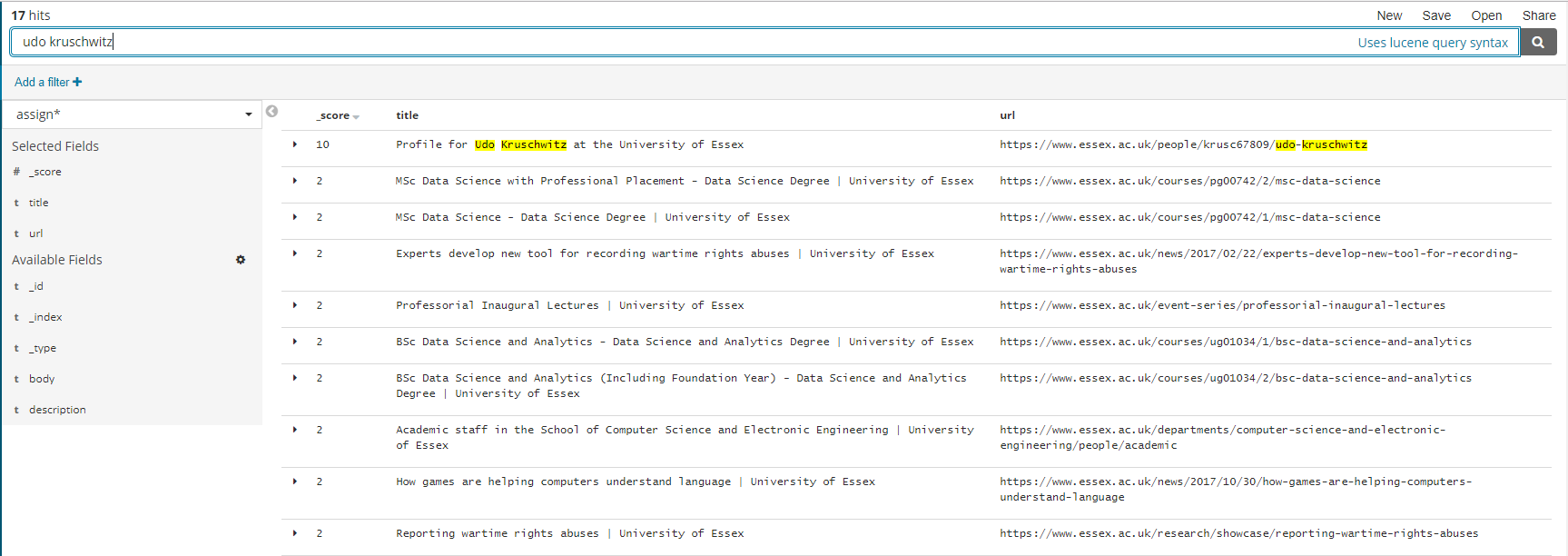
5.9 Query 9 – “Project open day”  
**Expected result:** 

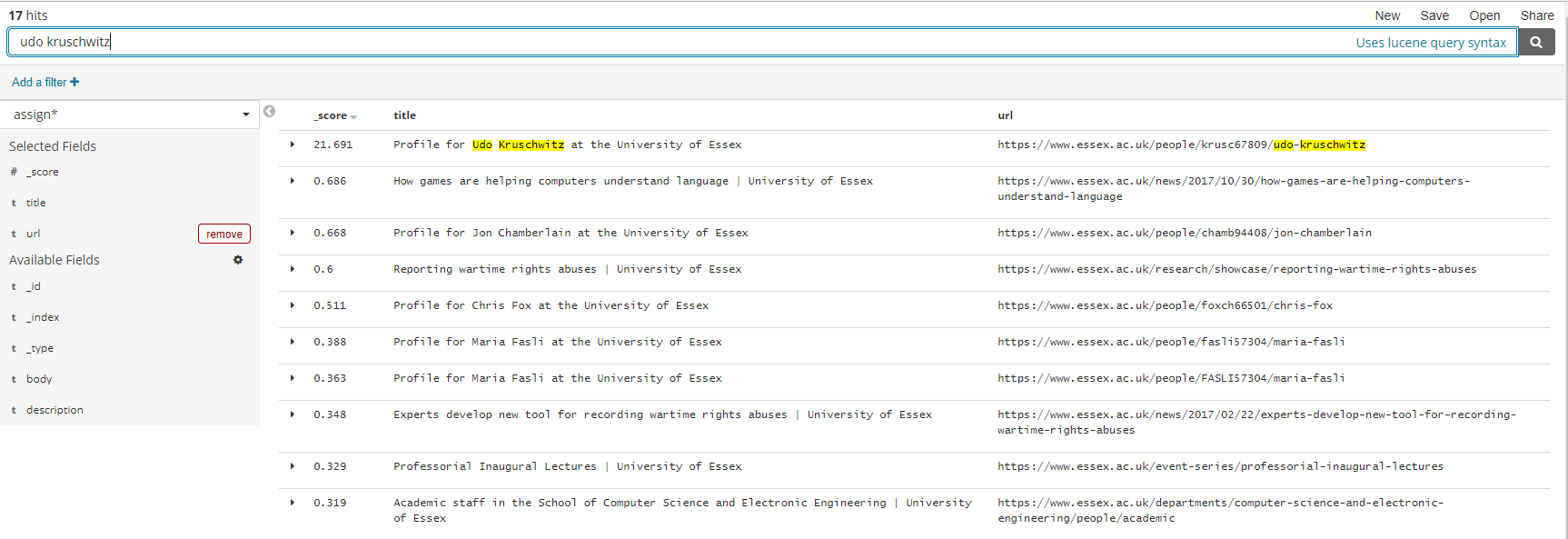
**Boolean result:** 

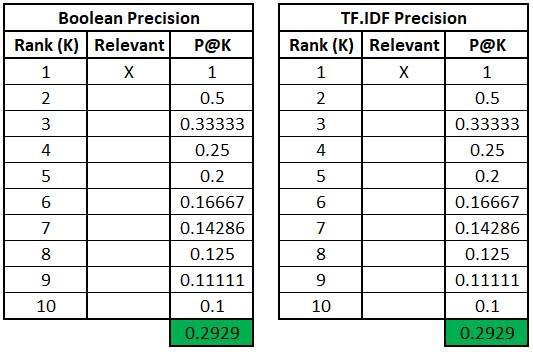
**TF.IDF result:** 

  
In this query, TF.IDF was slightly more precise than Boolean, both matching the expected result with their first result.

5.10 Query 10 – “Udo Kruschwitz”  
**Expected result:** 

**Boolean result:** 

**TF.IDF result:** 



In this query, TF.IDF and Boolean had the same precision, both matching the expected result as their first and only relevant result.

## 5.11 Summary of Results

Using the MAP metric, the results are as follows for the above set of queries:

**TF.IDF:** 0.54411 **Boolean:** 0.47783

This shows that TF.IDF has a higher average precision across the set of queries. Both types retrieved the expect result as their first answer for every query in the set, but TF.IDF produces more relevant results overall than Boolean.

# 6 Discussion of my Solution, Improvements and Extensions

I am happy with my solution and how it developed, however, I believe there is a lot of room for improvement, specifically in the indexing and the crawler. I’d like to complete the implementation I had in mind initially where there were fields for all the different tag types. On top of that, I’d like to implement a keywords field using a pipeline similar to the first assignment to find key words and phrases to identify documents with. Performing stemming and stopping would also assist with the reduction of useless content. I noticed that the layout of the webpages are very similar within the essex.ac.uk domain so I’d use that to try and get rid of irrelevant thing such as the footer and header of pages, as they’re consistent between pages and so don’t offer any unique qualities to help distinguish during querying. I’m still undecided on the usefulness of link tags as some won’t be of use such as ones found in a navigation menu, but sometimes they can offer some insight into the page’s purpose.

As an extension, it would be interesting to continually improve upon the indexing and measure the querying results to monitor improvements.