Samarth Jain 2017A7PS0067P Group 10

Prateek Sharma 2017A7PS0171P

Korukanti Shatrugna Rao 2017A7PS0118P

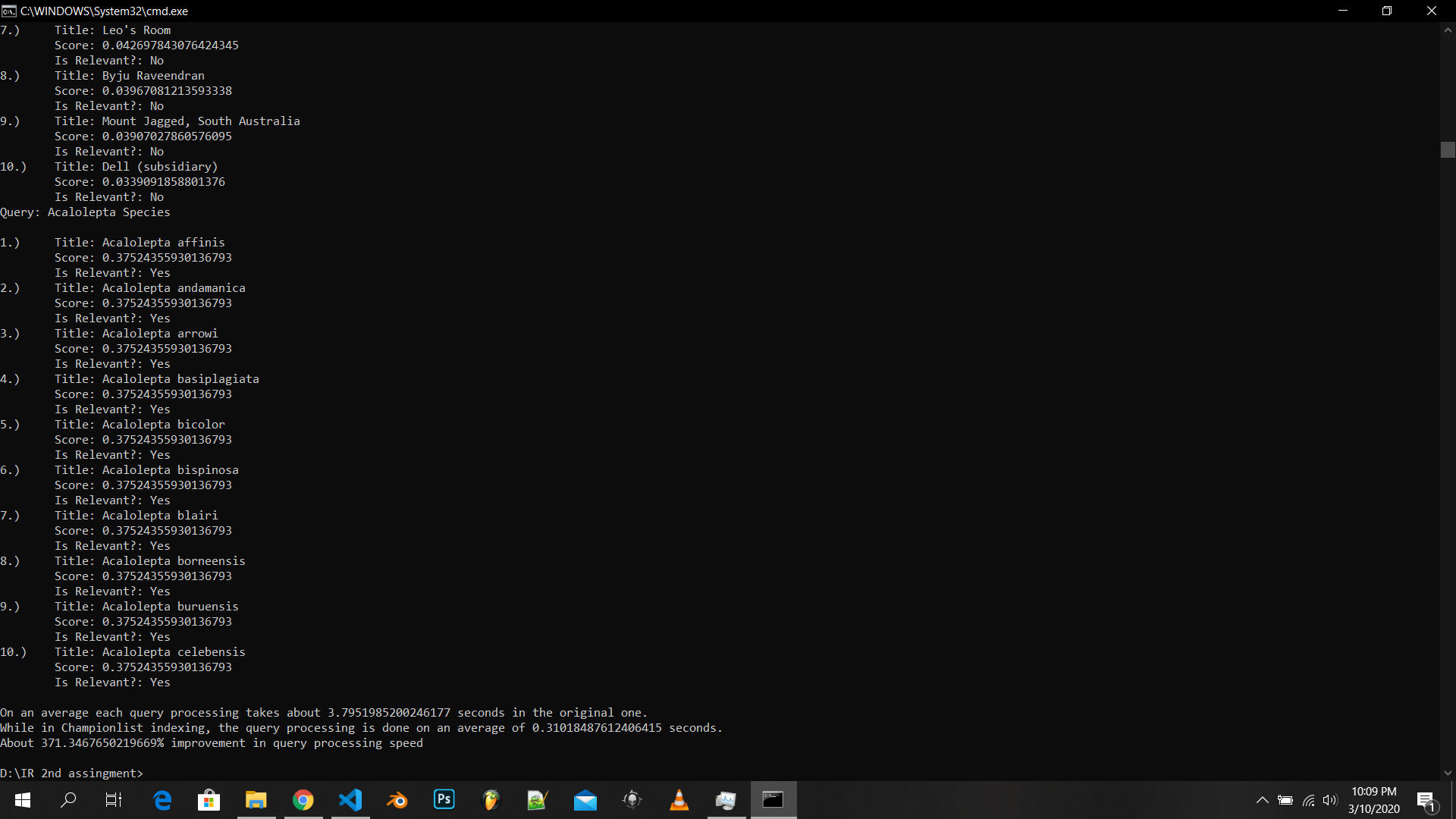
**REPORT**

**IMPLEMENTATION**

1. Library used nltk, re, csv.
2. Text cleaning and punctuation removal is done with re and the tokenization is done with nltk.
3. In part 1 an inverted index is built from the corpus in the form of a dictionary. And the doc id along with its respective title is stored in another array.
4. The processed corpus is converted to Inverted index and stored in a csv file which is then exported using csv library.
5. The doc ids and titles are exported to a text file.
6. The csv file is then imported and used for building the vector space model where then the inputted query is processed from the built vector space model, giving out results.
7. The same processing file which constructs the vector space model is again used to implement the issues correction we have noticed within the code of the implementation

**IMPROVEMENTS IN PART 1**

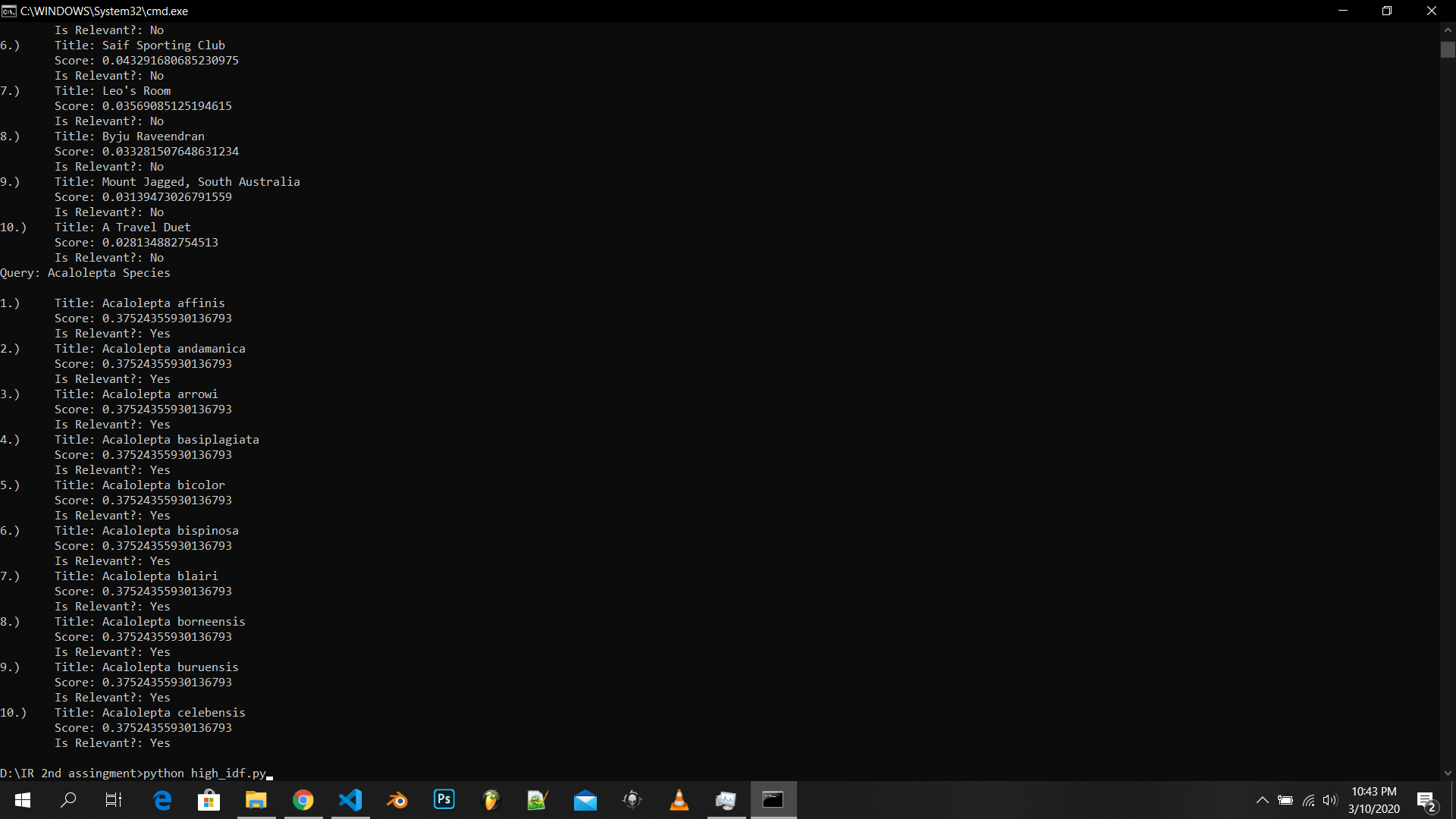
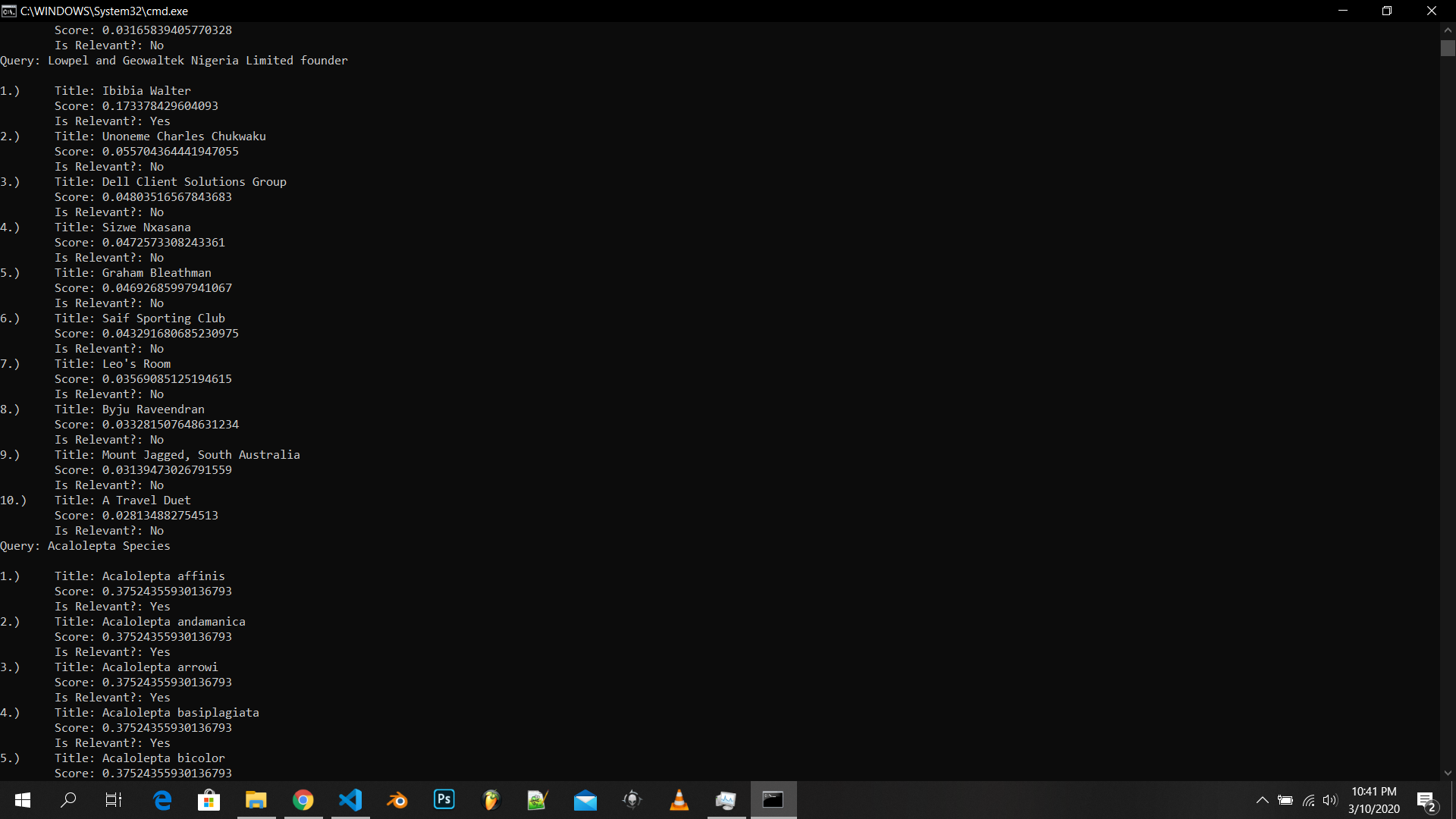
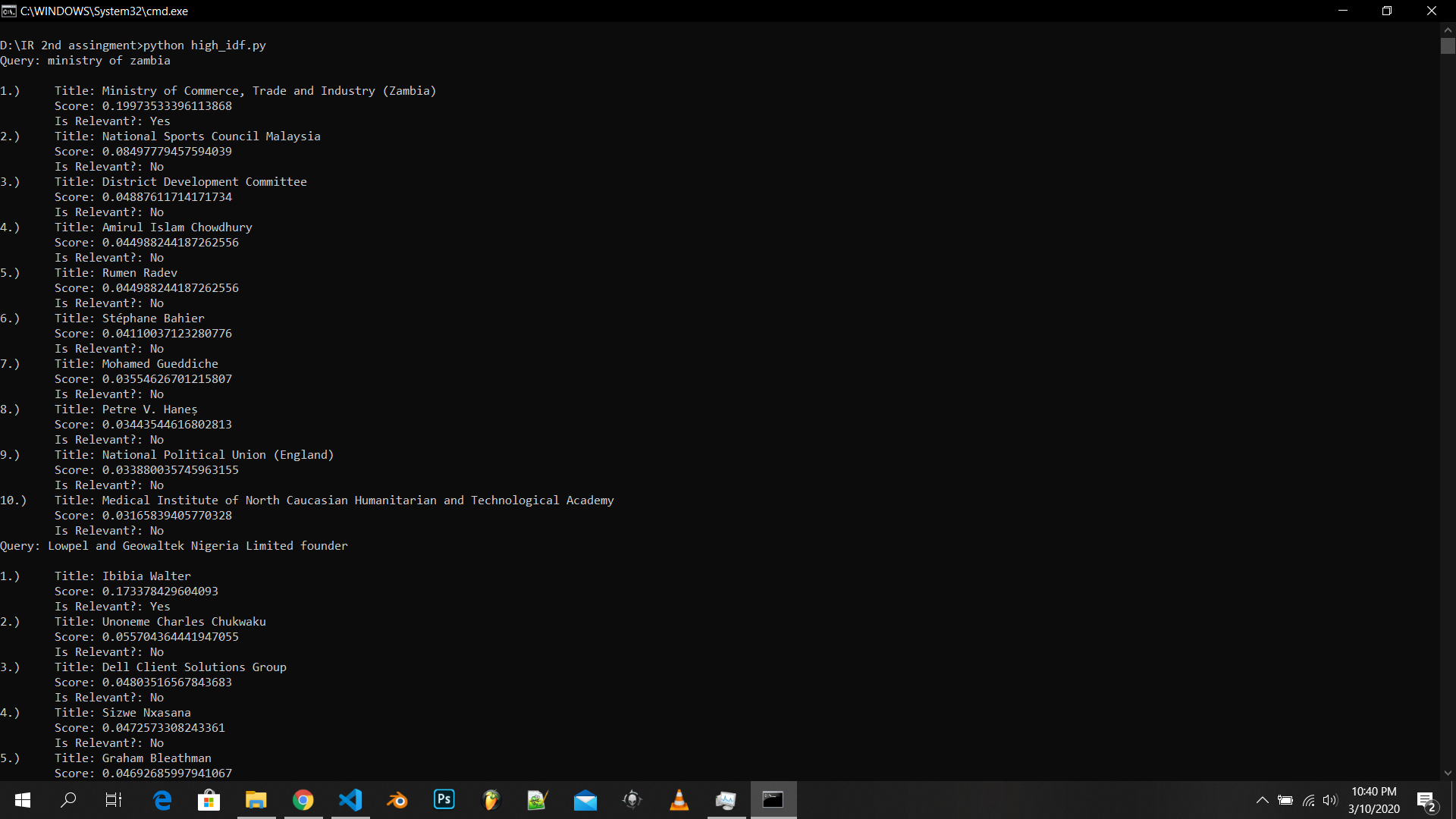
ISSUE 1

1. HIGH LATENCY - Score computation is a large (10s of %) fraction of the, CPU work on a query, Generally, we have a tight budget on latency We’ll look at ways of cutting CPU usage for scoring, without compromising the quality of results (much) .
2. CHAMPION LIST-For every term (t), store a list of r documents that have the highest score for term t, The score we are using weight score.r is fixed at the index creation time, thus it’s possible that r< K. The set of r documents are called the champion list for term t. Now, for a query, create a set of documents A from the champion list of all the terms in the query.
3. Decrease in the number of documents will result in low latency.
4. This is an inexact way of finding top r documents, corner cases might exist. example- weights are almost equal in more than r elements.
5. With the same queries in the original and modified ones, we have calculated their processing speed and checked out its improvement ratio from its predecessor. 

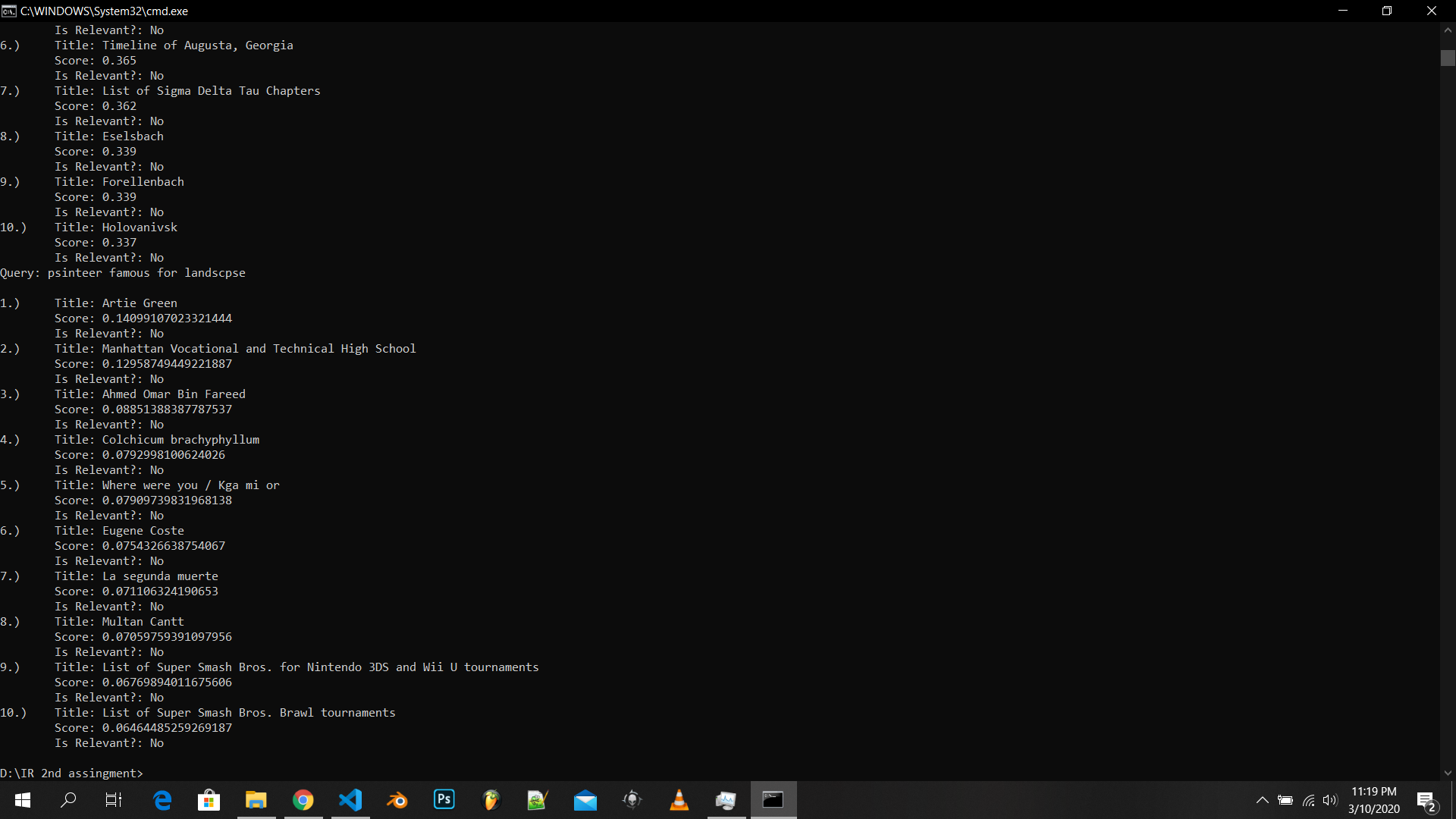
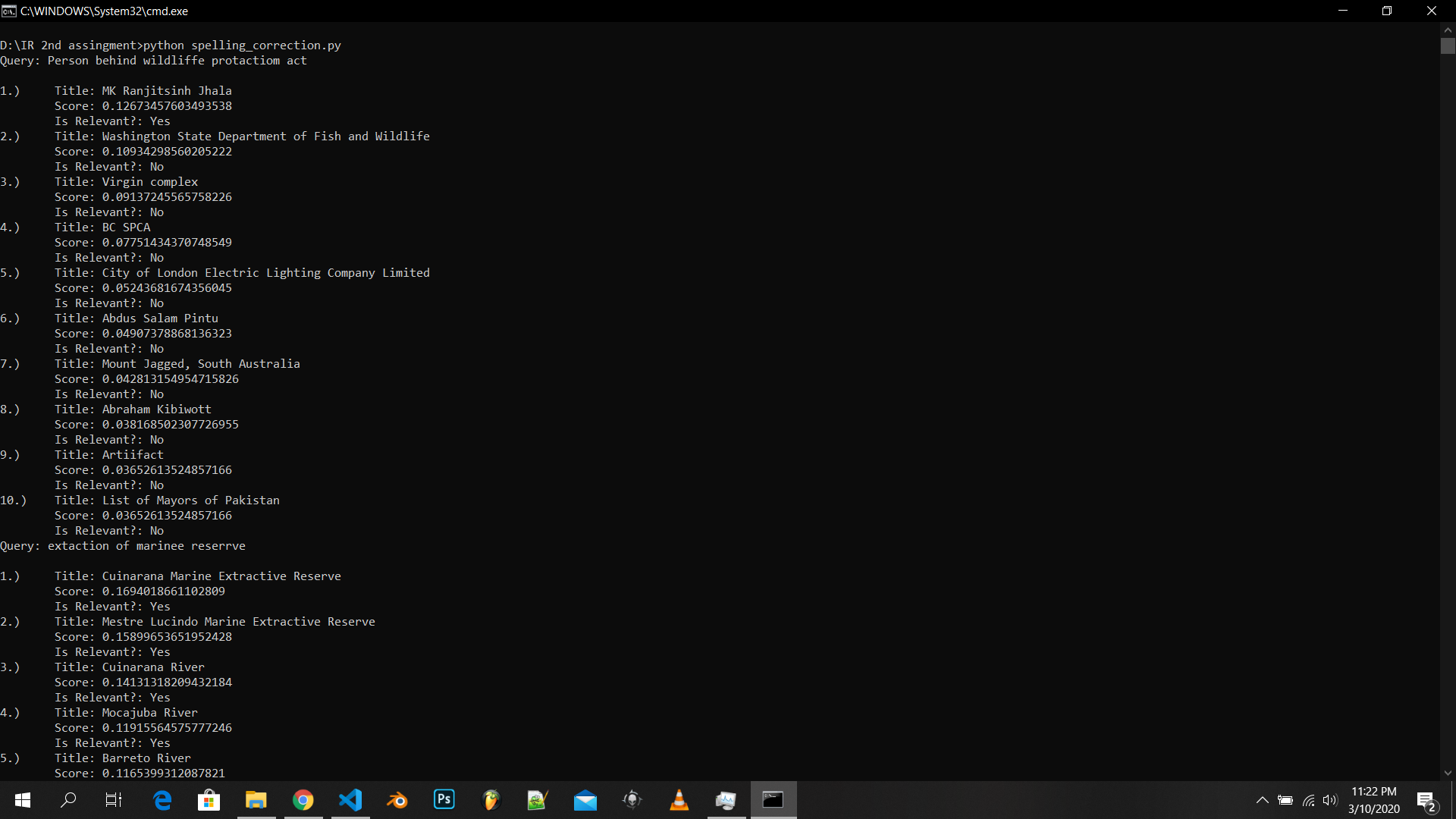
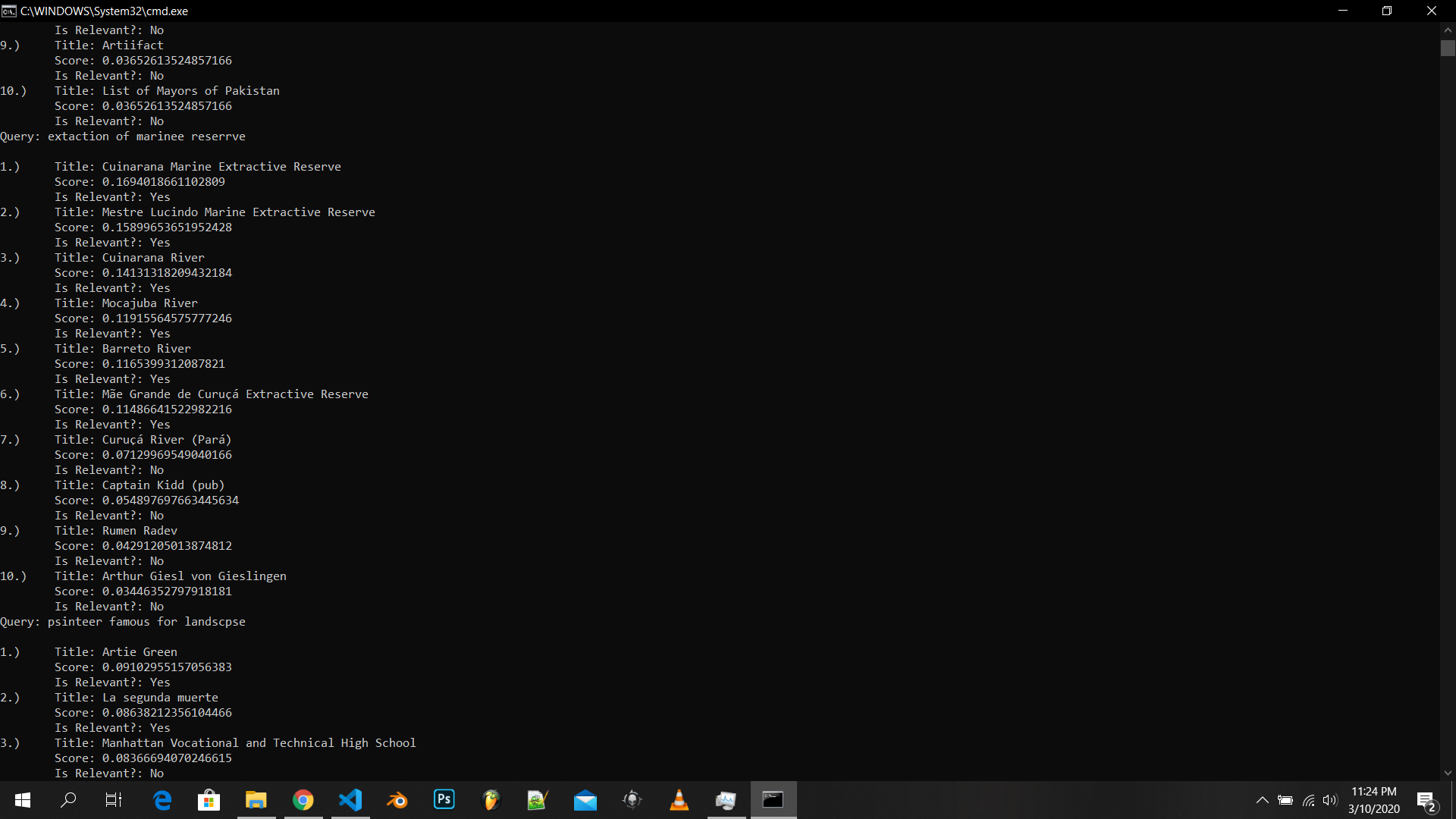
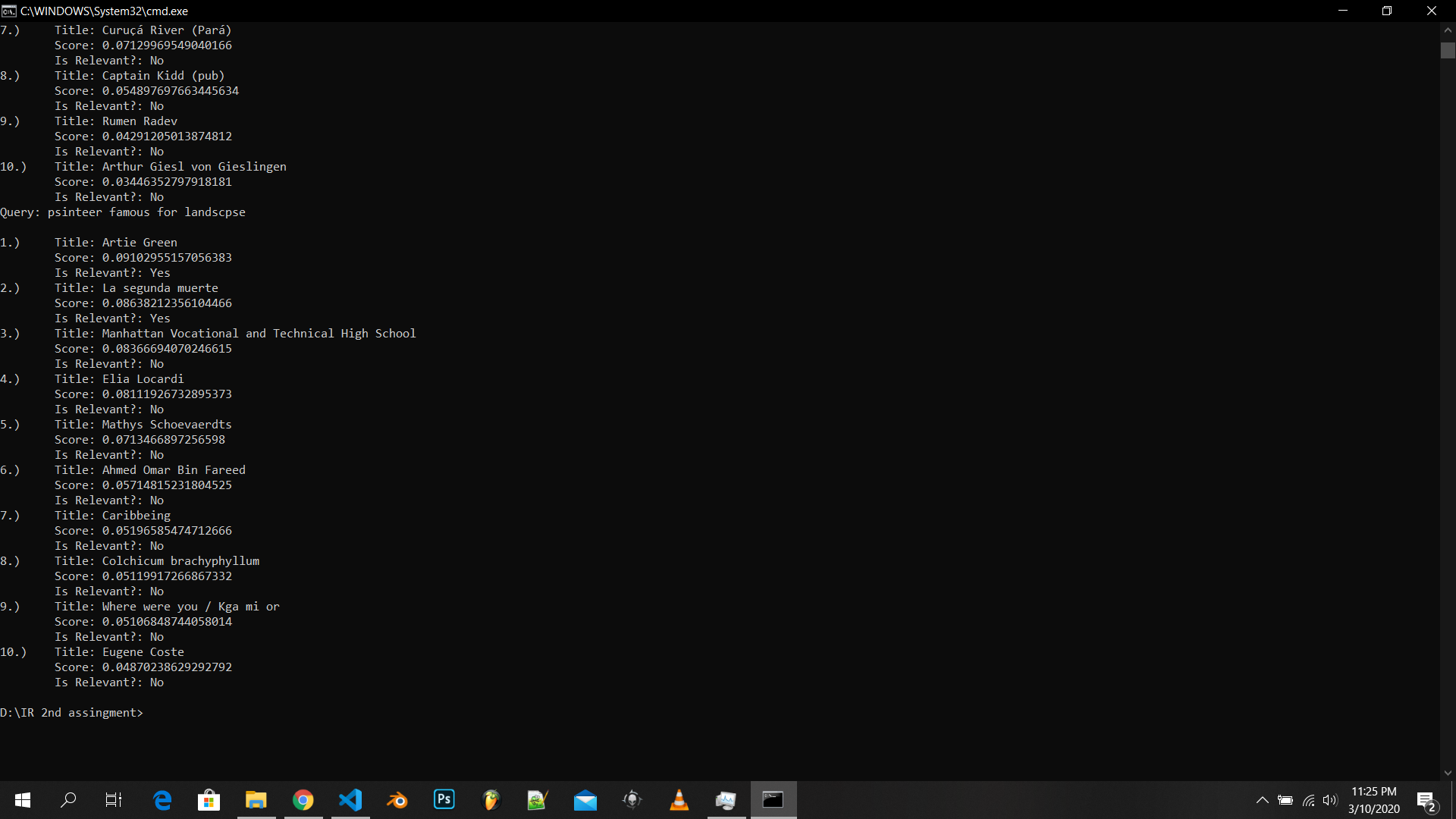
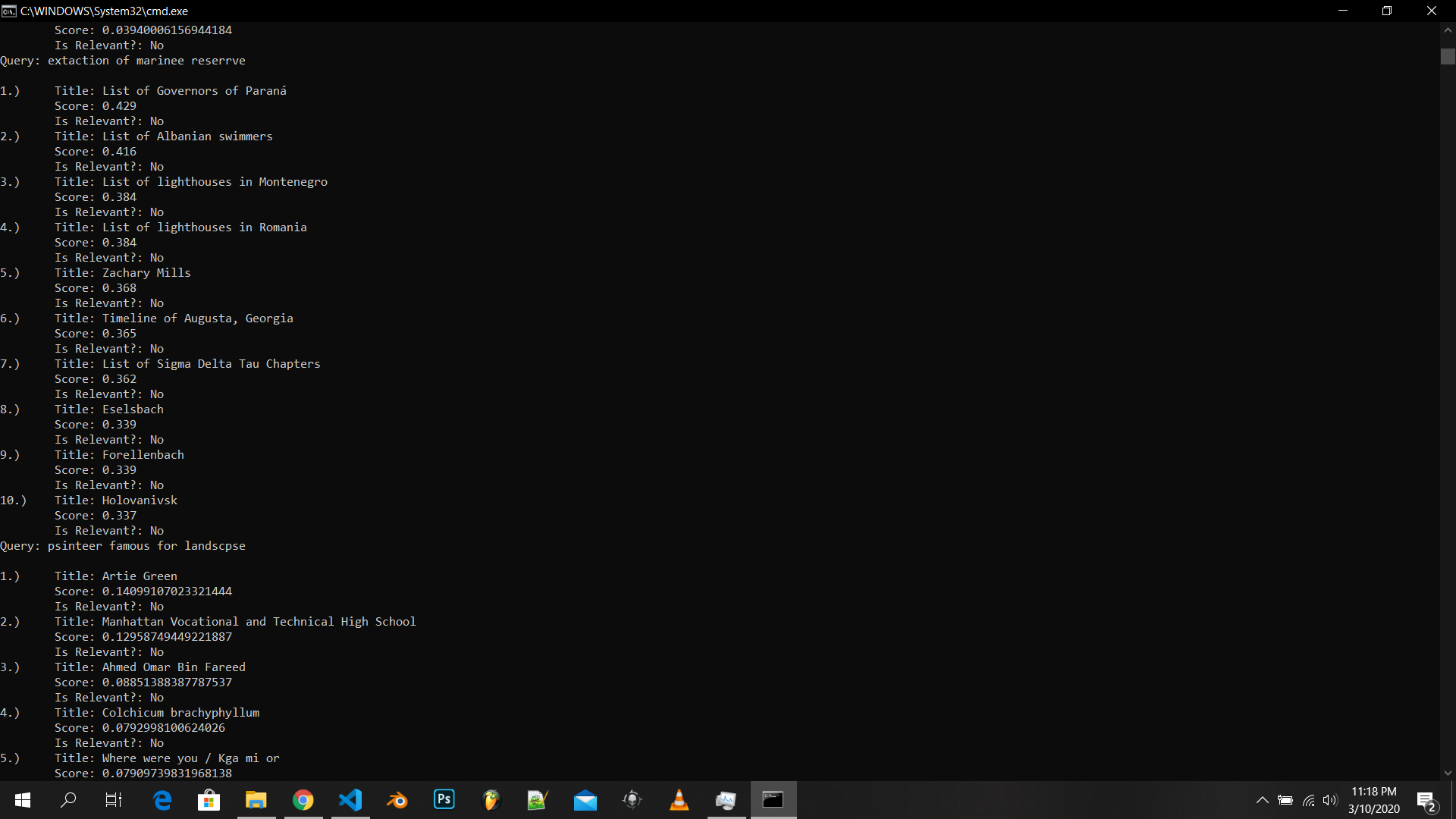
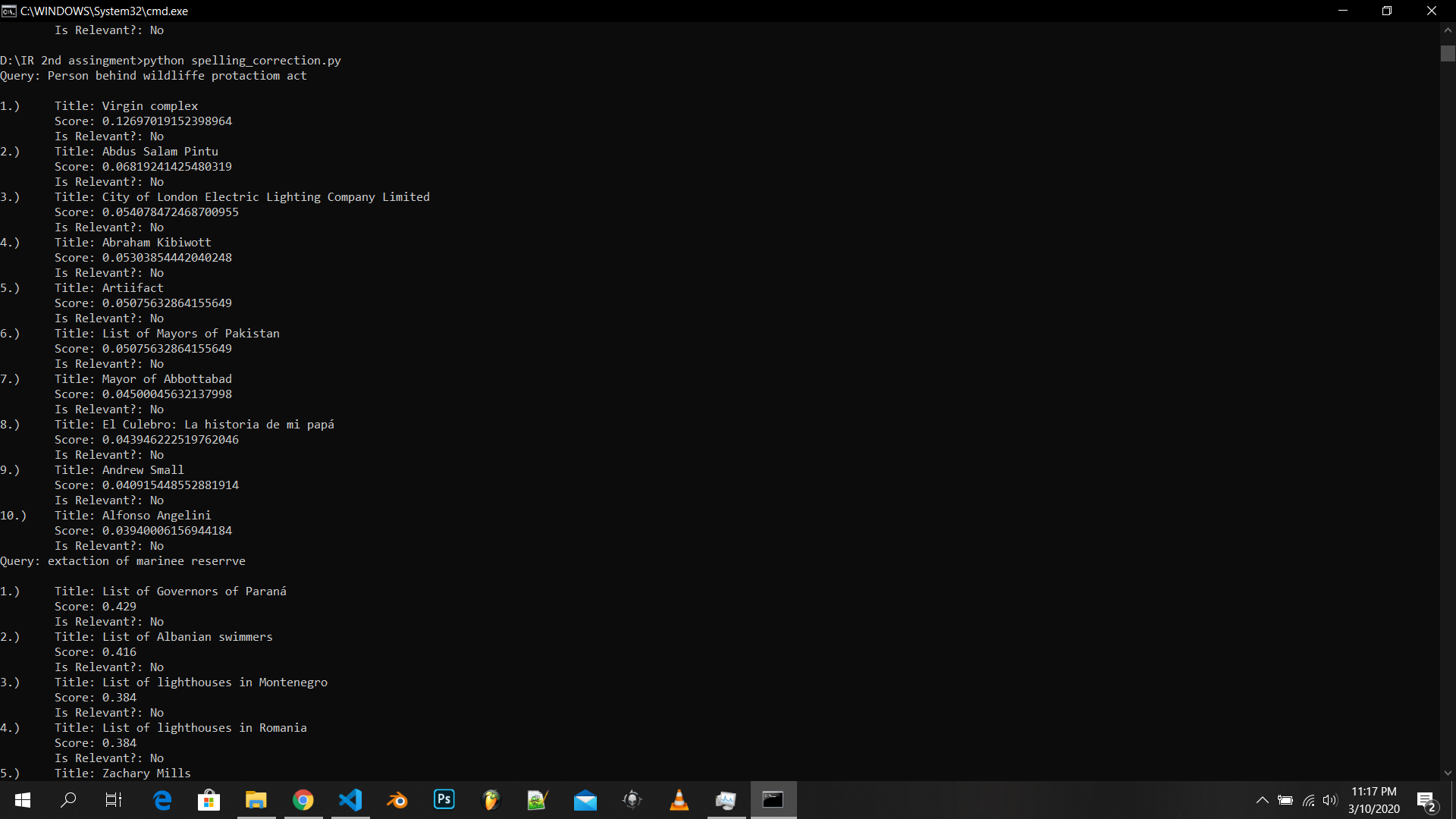
It had a 371.3 % improvement in processing speed

ISSUE 2

1. High frequency of some words such as stop words will increase the score of some documents ,since stop words are not the prime indicators of relevance of a document, the weight of stop words should be lesser than rare words.
2. In query, calculate IDF of every term, store the maximum IDF value, ignore the terms with IDF value lesser than half of the maximum value of IDF .
3. Less frequent words are given more importance than more frequent words, it will increase the probability of extracting more relevant documents .
4. Ignoring some words might cause some inconsistency in search.

Sample queries.

ISSUE 3

1. The IR system in part 1 has no way to check whether the entered term of the query is correct in spelling. This results in unexpected consequences in the final output.
2. A spell corrector can be implemented using many dependable open source libraries. Each term of the query undergoes correction if the term is not available in the dictionary.
3. The imported libraries correct the spelling of the term and undergoes the query processing to correct results.
4. It does not always yield the expected correct term of its respective incorrectly entered term. This might cause the IR system to give an overall different result to the query.
5. 

Before correction After correction

Thank You