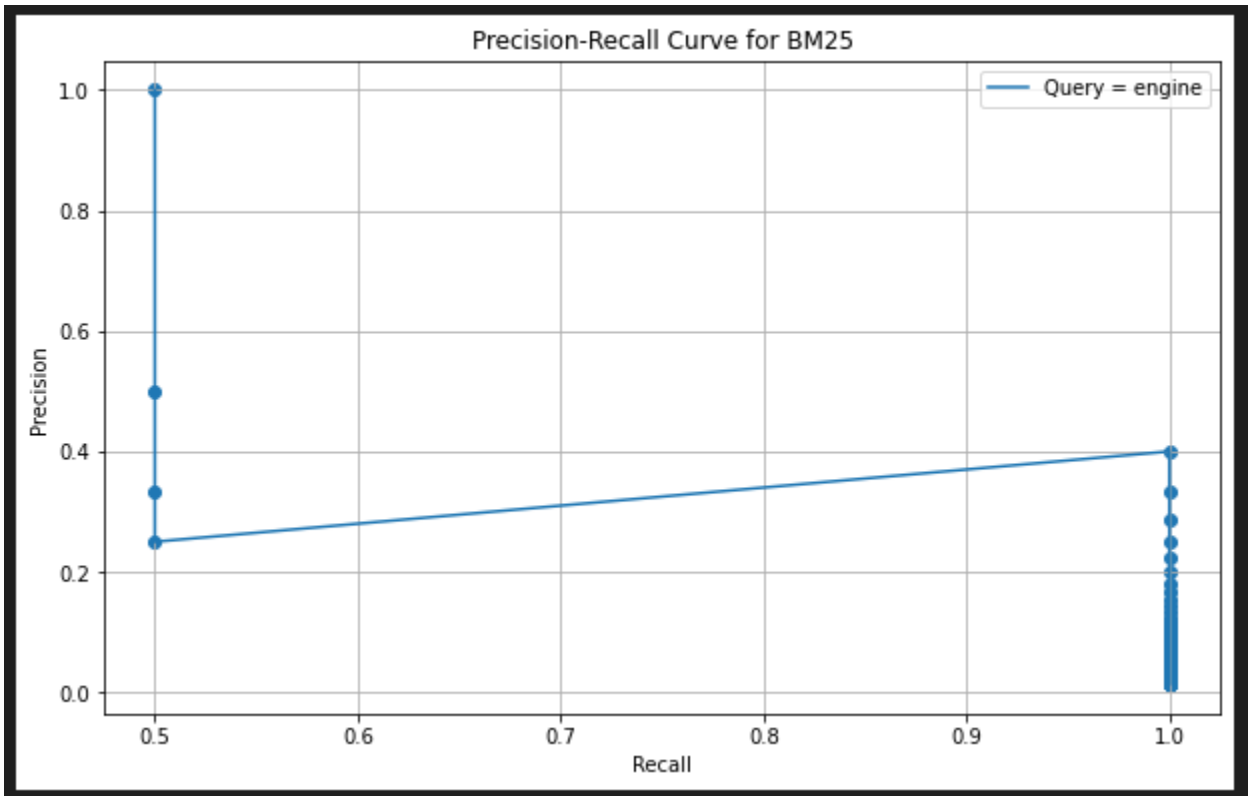
IRE Assignment 3

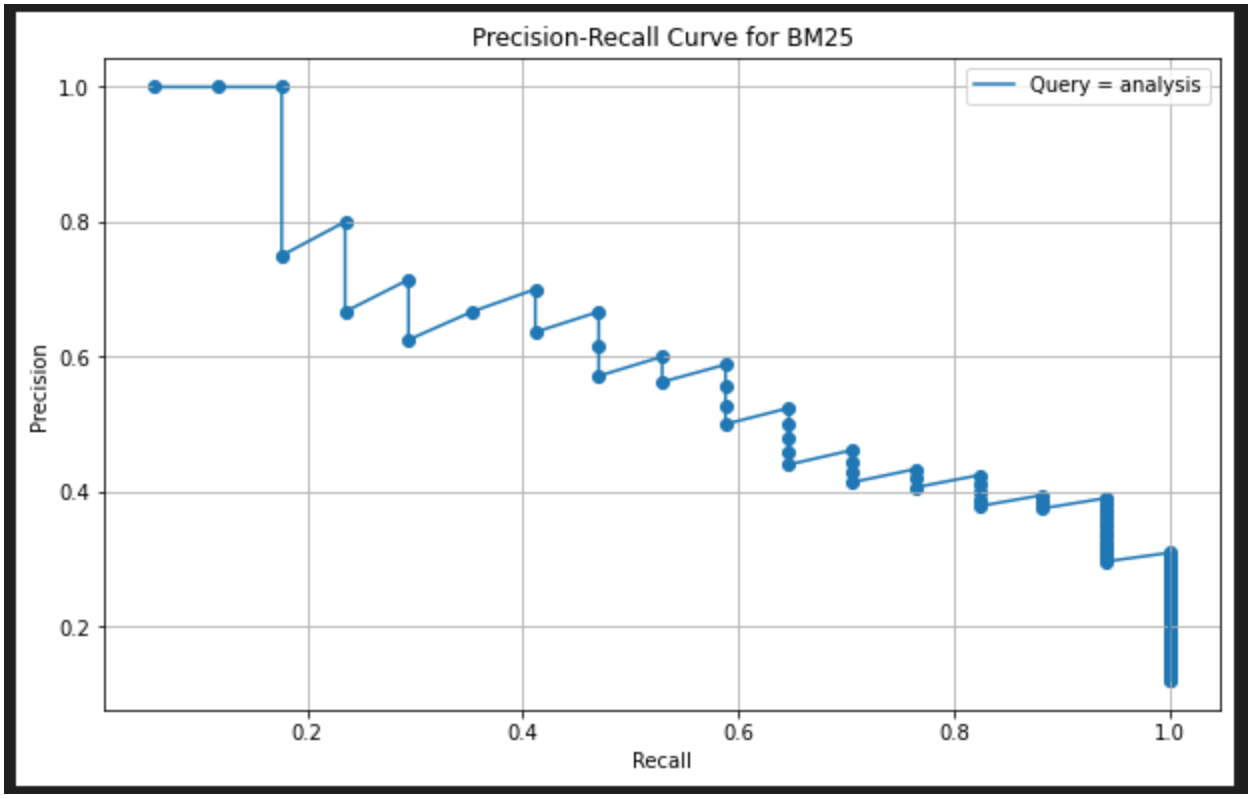
Tasks

2.1 Evaluate search engine

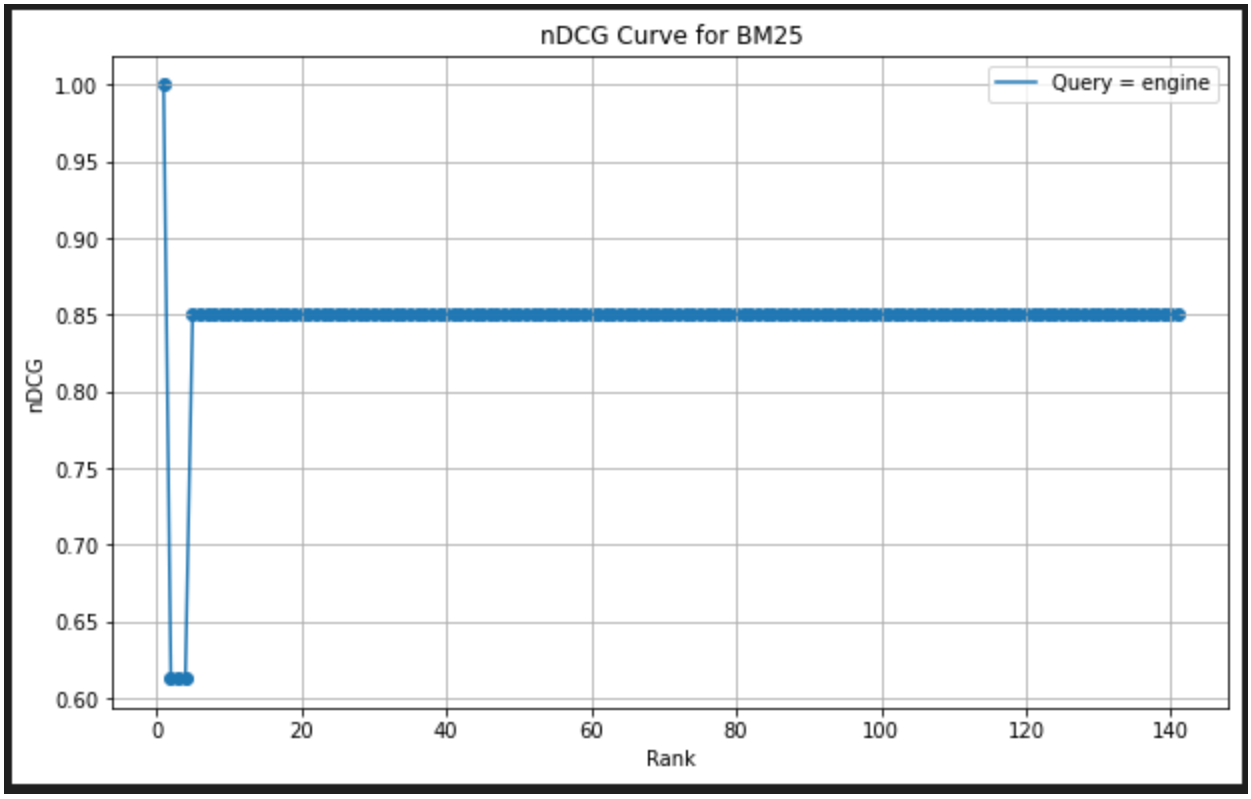
The provided code includes two functions, `generate_random_ranking` and `run_experiments`, for generating random ranked lists and relevance lists. The `generate_random_ranking` function creates random ranked and relevance lists based on a specified number of items, while `run_experiments` conducts multiple experiments with different numbers of items and collects data for each experiment. The precision_recall function calculates the precision and recall at any rank, given a ranking and relevance list.

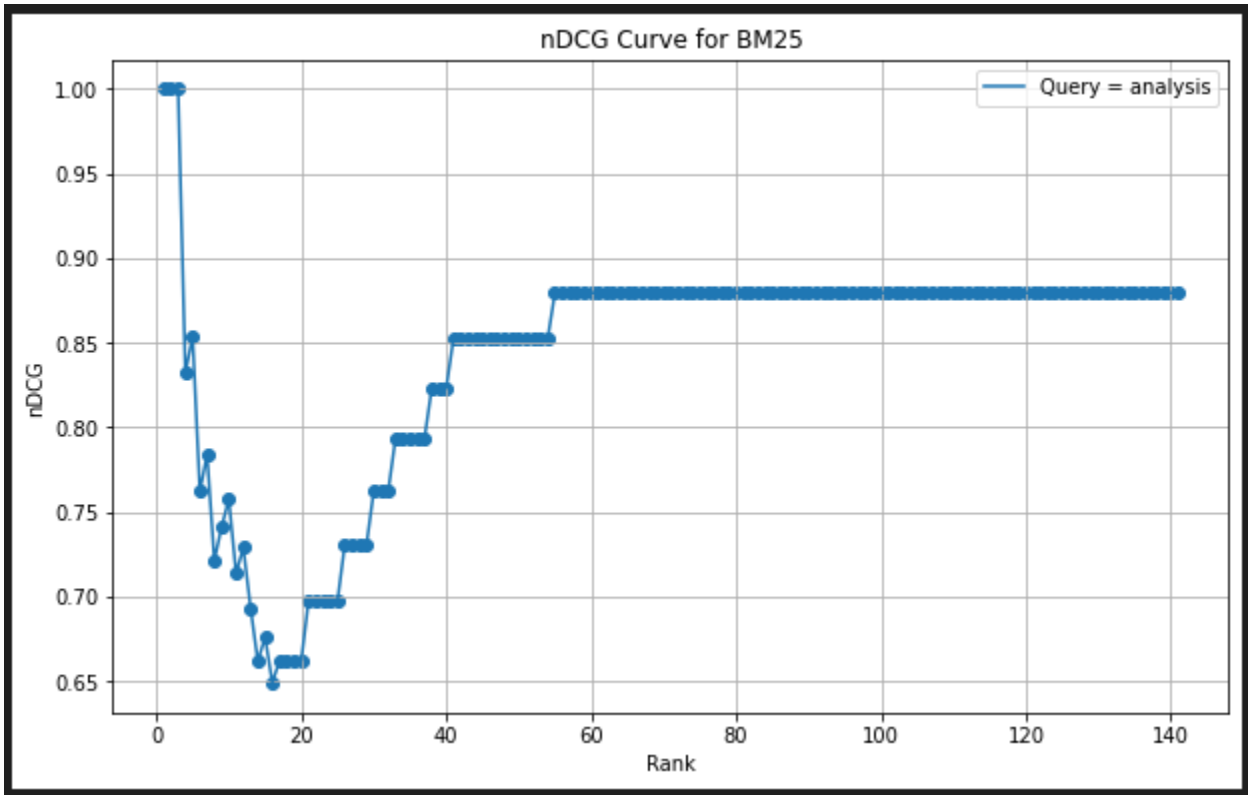
Next, I have plotted the Precision-Recall (PR) curves for a random ranking as well as for the ranking of the queries “engine” and “analysis” given by the bm25 model. For these queries, the relevant documents are determined by the grep command with regular expressions.





For the third task, I have a function to calculate the nDCG scores based on a binary grading relevance scheme. I then plot the nDCG scores of the ranking and relevance lists generated by the bm25 model for the same 2 queries as before.





Comparing nDCG graphs with precision - recall graphs, we observe that nDCG appears to capture more information about the nature of the ranking. This is because precision -

recall graphs appear to be following a general declining trend, but the nDCG graphs are more complex. In both cases, we want higher values, i.e., we would like higher values of precision, recall and nDCG at each rank. Since nDCG emphasizes higher ranks (lower rank number or left side of graph) by assigning them more importance due to the discounted gain factor, nDCG is a more complex and informative metric to evaluate the performance of a search engine compared to standard precision and recall.

For the 4th task, I use the same relevance and ranking lists generated as before and calculate the average precision until recall = 1, calculated separately for each query. The mean of the average precisions for the 2 queries is then computed and displayed.

For the 5th task, I have a function to compute the micro and macro f1 scores given a ranking list and a binary relevance list. I compute these scores for the same queries and display them.

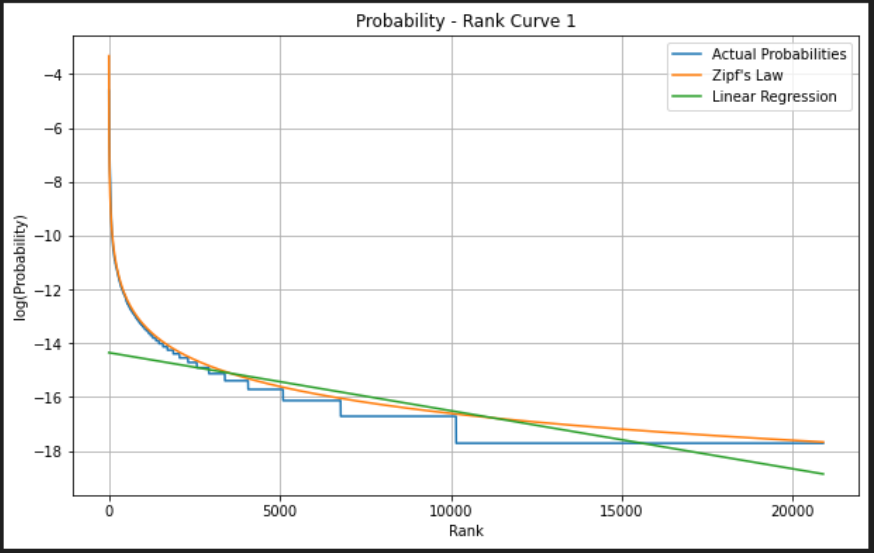
For the 6th task, I have used the scipy.stats module to compute the spearman and kendall-tau correlation coefficients given a ranked list and a relevance list.

2.2 Verify the Zipf's Law

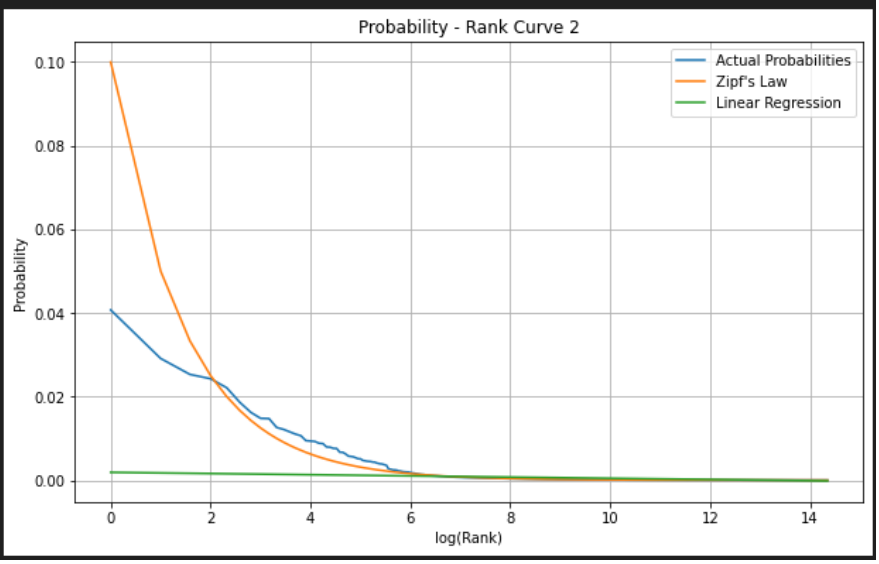
I have taken the L'homme Qui Rit french book as my document (or corpus) to verify Zipf's Law. For this, I first tokenize the content using regular expressions. This extracts all french words in the text, when the UNICODE flag option is used. I then convert the words to lowercase, find unique words and generate a frequency dictionary.

Once I have the frequency dictionary for each distinct word, I can plot the data as a bar chart and also find the probabilities for each distinct word by dividing each frequency by the total number of words. These probabilities are plotted against rank (position in reverse sorted order) along with the curve p = 0.1/r, to see the correlation between Zipf's law curve and the actual probability - rank curve. These curves are in the shape of a sharp L, which is unclear when plotted in the graph, so I have plotted log(probability) versus rank graph and probability versus log(rank) graph.

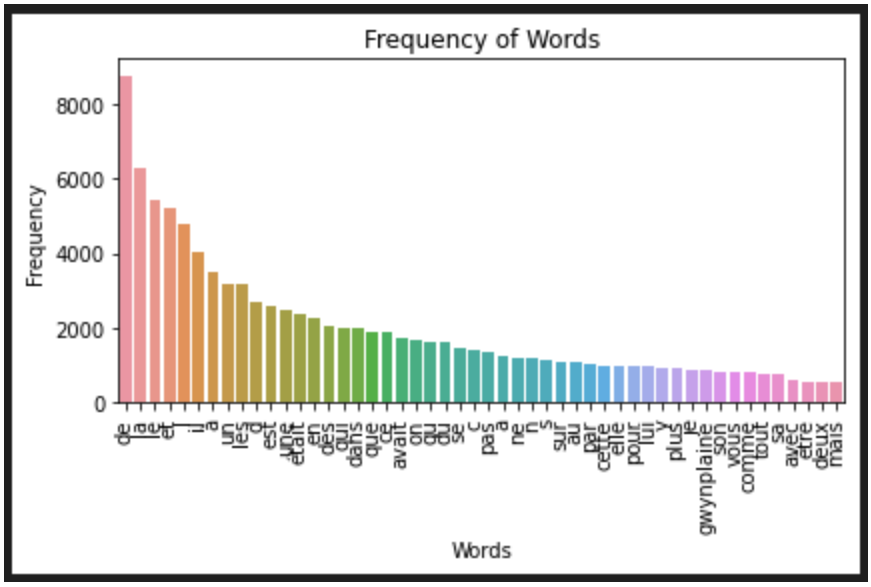
```
R-squared: 0.691603148282479
p: -0.00021589244842264092
```




```
R-squared: 0.11905528373451266
p: -0.00014196646681273207
```



From the above 2 probability - rank curves, we observe that the Zipf's law curve seems to approximate the actual probabilities at each rank, with good accuracy for a majority of the curve. The linear regression model does a better job of fitting the 2nd curve than the first. This is most likely due to the fact that the first curve follows Zipf's law well, since probability (and log(probability)) are inversely proportional to rank, and the second curve is more evened out compared to the first, since we are plotting the log of rank on the x-axis.



Finally, I found 10 examples of words with the highest frequencies, lowest frequencies (rare words) and average frequencies, from the sorted frequency list. For additional comparison, I included examples of words having frequency around the median frequency.

	Very Frequent Words	Averagely Frequent Words	Median Frequent Words	Very Rare Words
0	de	irlandais	différait	alfred
1	la	effets	dégradé	démène
2	le	lever	empiéter	jaunit
3	et	entrant	rayonnante	signalent
4	l	robert	tabouret	camuse
5	il	tigre	désespérait	trébuché
6	à	pavé	renouvellement	hume
7	un	pattes	légalité	bourchieret
8	les	couloir	coraux	pétrir
9	d	reculer	smallmouth	abordant

Intuitively, the last category of 'very rare words' would seem to be more useful in information retrieval since these rare words contain more overall information. Also, since they are sparsely located, the location or position of these words could be useful for analysis, like word association or named-entity-recognition tasks. This last category is also likely to have words with a higher tf-idf score, since idf for these words will be high in a large corpus. In smaller corpuses, the very frequent words will have high tf-idf value.