Homepage finding task focuses on getting the most appropriate result at rank-1. What metric(s) would you use to evaluate the effectiveness of an algorithm for this task? Why? [2 points]

- I would use the Mean Reciprocal Rank (MRR) metric to evaluate the effectiveness of an algorithm for the Homepage Finding Task. This is because MRR evaluates the effectiveness of the searching algorithm by focusing on the rank of the first relevant documennt/result. the MRR metric is calculated by calculating the Reciprocal Rank (RR) for a set of queries and then taking the average of the RR values. The RR value is calculated by dividing 1 by the rank of the first relevant document.

- As I mentioned before, the MRR is focused on the rank of the first relevant document. And since the Homepage Finding Task focuses on getting the most appropriate result at Rank-1 (the first relevant document), the MRR metric is the most appropriate metric to evaluate the effectiveness of an algorithm for this task.

- Relevant Equations:

    - RR = 1 ÷ Rank of First Relevant Document Retrieved
    - MRR = Σ RR ÷ Total Number of Queries

Explain the intuition behind R-precision. What are the best and the worst values of R-precision that you can get? [2 points]

- R-Precision is the precision (proportion of retrieved documents that are relevant) after the first R (number of relevant documents) documents have been retrieved. The idea behind R-Precision is to evaluate the effectiveness of a search algorithm by focusing on the precision of the first R relevant documents retrieved. The R-Precision metric is calculated by dividing the number of relevant documents retrieved by the total number of relevant documents in the collection.
    - Best Case R-Precision: 1.0 (All the first R documents retrieved are relevant)
    - Worst Case R-Precision: 0.0 (None of the first R documents retrieved are relevant)
- Relevant Equations:
    - R-Precision = (Number of Relevant Documents Retrieved) ÷ (Total Number of Relevant Documents)

Get ‘vaswani’ dataset and index it as done in the class. Prepare a dataframe with the following queries and perform retrieval using TFIDF, BM25, and language model. Compute MAP, reciprocal rank, and R-precision for a batch run experiment. Report these numbers and provide your brief comments comparing the three retrieval models. [6 points] | qid | query | |—-|———————————-| | 62 | fast transistor counters | | 63 | low pass lattice filters | | 70 | variable ultra high frequency attenuators | | 73 | transistor sweep generators | | 75 | optimising linear networks |

```
import pyterrier as pt
if not pt.started():
  pt.init()

import pandas as pd
```

```
C:\Users\Fortu\AppData\Local\Temp\ipykernel_48640\2591423586.py:2: DeprecationWarning: Call to
deprecated function (or staticmethod) started. (use pt.java.started() instead) -- Deprecated
since version 0.11.0.
  if not pt.started():
```

```
dataset = pt.get_dataset("vaswani")
indexer = pt.TRECCollectionIndexer("c:/Users/Fortu/Downloads/Aut2024/Info 376/a_6_index")
indexref = indexer.index(dataset.get_corpus())
index = pt.IndexFactory.of(indexref)
print(index.getCollectionStatistics().toString())
```

```
Number of documents: 11429
Number of terms: 7756
Number of postings: 224573
Number of fields: 0
Number of tokens: 271581
Field names: []
Positions:   false
```

```
queries = pd.DataFrame([["62", "fast transistor counters"],
                        ["63", "low pass lattice filters"],
                        ["70", "variable ultra high frequency attenuators"],
                        ["73", "transistor sweep generators"],
                        ["75", "optimising linear networks"]],columns=["qid","query"])
queries
```

	qid	query
0	62	fast transistor counters
1	63	low pass lattice filters
2	70	variable ultra high frequency attenuators
3	73	transistor sweep generators
4	75	optimising linear networks

```
print(dataset.get_qrels())
```

```
     qid  docno  label
0      1   1239      1
1      1   1502      1
2      1   4462      1
3      1   4569      1
4      1   5472      1
...   ..    ...    ...
2078  93   9875      1
2079  93   9956      1
2080  93  10497      1
2081  93  11191      1
```

```
2082  93  11318      1

[2083 rows x 3 columns]
```

```
tf_idf = pt.BatchRetrieve(index, wmodel="TF_IDF")
bm25 = pt.BatchRetrieve(index, wmodel="BM25")
Hiemstra = pt.BatchRetrieve(index, wmodel="Hiemstra_LM")
```

```
C:\Users\Fortu\AppData\Local\Temp\ipykernel_48640\1887074501.py:1: DeprecationWarning: Call to deprecated class BatchRetrieve. (use pt.terrier.Retriever() instead) -- Deprecated since version 0.11.0.
  tf_idf = pt.BatchRetrieve(index, wmodel="TF_IDF")
C:\Users\Fortu\AppData\Local\Temp\ipykernel_48640\1887074501.py:2: DeprecationWarning: Call to deprecated class BatchRetrieve. (use pt.terrier.Retriever() instead) -- Deprecated since version 0.11.0.
  bm25 = pt.BatchRetrieve(index, wmodel="BM25")
C:\Users\Fortu\AppData\Local\Temp\ipykernel_48640\1887074501.py:3: DeprecationWarning: Call to deprecated class BatchRetrieve. (use pt.terrier.Retriever() instead) -- Deprecated since version 0.11.0.
  Hiemstra = pt.BatchRetrieve(index, wmodel="Hiemstra_LM")
```

```
tf_idf.transform(queries)
```

	qid	docid	docno	rank	score	query
0	62	10356	10357	0	12.303227	fast transistor counters
1	62	8961	8962	1	10.037327	fast transistor counters
2	62	8340	8341	2	9.516181	fast transistor counters
3	62	1965	1966	3	9.121030	fast transistor counters
4	62	4710	4711	4	8.906797	fast transistor counters
...	...	...	...	...	...	...
4735	75	6715	6716	921	1.347560	optimising linear networks
4736	75	6763	6764	922	1.334390	optimising linear networks
4737	75	4820	4821	923	1.296382	optimising linear networks
4738	75	8240	8241	924	1.296382	optimising linear networks
4739	75	4929	4930	925	1.194326	optimising linear networks

4740 rows × 6 columns

```
bm25.transform(queries)
```

	qid	docid	docno	rank	score	query
0	62	10356	10357	0	22.281739	fast transistor counters

	qid	docid	docno	rank	score	query
1	62	8961	8962	1	18.164384	fast transistor counters
2	62	8340	8341	2	17.157584	fast transistor counters
3	62	1965	1966	3	16.409894	fast transistor counters
4	62	4710	4711	4	16.051816	fast transistor counters
...	...	...	...	...	...	...
4735	75	6715	6716	921	2.381950	optimising linear networks
4736	75	6763	6764	922	2.358671	optimising linear networks
4737	75	4820	4821	923	2.291488	optimising linear networks
4738	75	8240	8241	924	2.291488	optimising linear networks
4739	75	4929	4930	925	2.111093	optimising linear networks

4740 rows × 6 columns

```
Hiemstra.transform(queries)
```

	qid	docid	docno	rank	score	query
0	62	10356	10357	0	11.963904	fast transistor counters
1	62	1965	1966	1	10.578431	fast transistor counters
2	62	8961	8962	2	10.037904	fast transistor counters
3	62	4710	4711	3	9.583759	fast transistor counters
4	62	8340	8341	4	9.469847	fast transistor counters
...	...	...	...	...	...	...
4735	75	6715	6716	921	0.779075	optimising linear networks
4736	75	6763	6764	922	0.770195	optimising linear networks
4737	75	4820	4821	923	0.744756	optimising linear networks
4738	75	8240	8241	924	0.744756	optimising linear networks
4739	75	4929	4930	925	0.677779	optimising linear networks

4740 rows × 6 columns

```
pt.Experiment(
    [tf_idf, bm25, Hiemstra],
    queries,
    dataset.get_qrels(),
    ["map", "recip_rank", "Rprec"]
)
```

	name	map	recip_rank	Rprec
0	TerrierRetr(TF_IDF)	0.448187	0.840	0.455642
1	TerrierRetr(BM25)	0.449766	0.850	0.462309
2	TerrierRetr(Hiemstra_LM)	0.392385	0.825	0.352756