Indexing reports:

My code for testing it: (you can find it on `index.py`)

I have broken it into different parts to be viewed more easily:

- **Preparation:**

```
if __name__ == '__main__':
    prepreprocessed_documents_path =
project_root+'/data/IMDB_Preprocessed.json'
    with open(prepreprocessed_documents_path, 'r') as f:
        preprocessed_documents = json.load(f)

    index = Index(preprocessed_documents)
```

- **Checking add/remove:**

Code:

```
print("Cheking add/remove:")
index.check_add_remove_is_correct()
```

Output:

```
Cheking add/remove:
Add is correct
Remove is correct
```

- **Storing and loading the index files:**

Code:

```
index.store_index(project_root+'/index', Indexes.DOCUMENTS.value)
index.store_index(project_root+'/index', Indexes.STARS.value)
index.store_index(project_root+'/index', Indexes.GENRES.value)
index.store_index(project_root+'/index', Indexes.SUMMARIES.value)
index.load_index('index')
```

- **Checking if index loaded correctly:**

Code:

```
print("Checking if index loaded correctly (documents):")
print(index.check_if_index_loaded_correctly(
        Indexes.DOCUMENTS.value,
        index.index[Indexes.DOCUMENTS.value]))

print("Checking if index loaded correctly (stars):")
print(index.check_if_index_loaded_correctly(
        Indexes.STARS.value,
        index.index[Indexes.STARS.value]))

print("Checking if index loaded correctly (genres):")
print(index.check_if_index_loaded_correctly(
        Indexes.GENRES.value,
        index.index[Indexes.GENRES.value]))

print("Checking if index loaded correctly (summaries):")
print(index.check_if_index_loaded_correctly(
        Indexes.SUMMARIES.value,
        index.index[Indexes.SUMMARIES.value]))
```

Output:

```
Checking if index loaded correctly (documents):
True
Checking if index loaded correctly (stars):
True
Checking if index loaded correctly (genres):
True
Checking if index loaded correctly (summaries):
True
```

- **Checking if indexing is good (documents) for 2 different words:**

Code:

```
print("Checking if indexing is good (documents):")
print('Word: tt1160419')
print(index.check_if_indexing_is_good(Indexes.DOCUMENTS.value, 'hello'))
print('Word: tt15239678')
print(index.check_if_indexing_is_good(Indexes.DOCUMENTS.value, 'word'))
```

Output:

```
Checking if indexing is good (documents):
Word: tt1160419
Brute force time:  6.198883056640625e-05
Implemented time:  4.0531158447265625e-06
Indexing is correct
Indexing is good
True
Word: tt15239678
Brute force time:  4.8160552978515625e-05
Implemented time:  9.5367431640625e-07
Indexing is correct
Indexing is good
True
```

- **Checking if indexing is good (stars) for 2 different words:**

Code:

```
print("Checking if indexing is good (stars):")
print('Word: ben')
print(index.check_if_indexing_is_good(Indexes.STARS.value, 'ben'))
print('Word: bob')
print(index.check_if_indexing_is_good(Indexes.STARS.value, 'bob'))
```

Output:

```
Checking if indexing is good (stars):
Word: ben
Brute force time:  4.982948303222656e-05
Implemented time:  6.198883056640625e-06
Indexing is correct
Indexing is good
True
Word: bob
Brute force time:  0.0002999305725097656
Implemented time:  1.9073486328125e-06
Indexing is correct
Indexing is good
True
```

- **Checking if indexing is good (genres) for 2 different words:**

Code:

```
print("Checking if indexing is good (genres):")
print('Word: drama')
print(index.check_if_indexing_is_good(Indexes.GENRES.value, 'drama'))
print('Word: comedy')
print(index.check_if_indexing_is_good(Indexes.GENRES.value, 'comedy'))
```

Output:

```
Checking if indexing is good (genres):
Word: drama
Brute force time:  2.1457672119140625e-06
Implemented time:  7.867813110351562e-06
Indexing is correct
Indexing is bad
False
Word: comedy
Brute force time:  1.0967254638671875e-05
Implemented time:  3.814697265625e-06
Indexing is correct
Indexing is good
True
```

- **Checking if indexing is good (summaries) for 2 different words:**

Code:

```
print("Checking if indexing is good (summaries):")
print('Word: murder')
print(index.check_if_indexing_is_good(Indexes.SUMMARIES.value, 'murder'))
print('Word: happy')
print(index.check_if_indexing_is_good(Indexes.SUMMARIES.value, 'happy'))
```

Output:

```
Checking if indexing is good (summaries):
Word: murder
Brute force time:  7.104873657226562e-05
Implemented time:  2.1457672119140625e-06
Indexing is correct
Indexing is good
True
Word: happy
Brute force time:  0.0007071495056152344
Implemented time:  7.152557373046875e-07
Indexing is correct
Indexing is good
True
```

As you can see, my code on `index.py` has passed all the tests.

- Near-duplicate page detection reports:

My code for testing it: (you can find it on `LSH.py`)
Code:

```
if __name__ == '__main__':
    docs = []
    title_to_id = {}
    with open(project_root+'/Logic/core/LSHFakeData.json') as f:
        docs = json.load(f)
    summaries = []
    for doc in docs:
        temp = doc['summaries']
        summary = ''
        for t in temp:
            summary += ' ' + t
        summaries.append(summary)
        title_to_id[len(summaries) - 1] = doc['title']
    docs = []
    with open(project_root+'/data/IMDB_Crawled.json') as f:
        docs = json.load(f)
    for doc in docs:
        temp = doc['summaries']
        summary = ''
        for t in temp:
            summary += ' ' + t
        if summary == '':
            continue
        summaries.append(summary)
        title_to_id[len(summaries) - 1] = doc['title']

    num_hashes = 625
    min_hash_lsh = MinHashLSH(summaries, num_hashes)
    buckets = min_hash_lsh.perform_lsh()

    print_buckets = []
    print("Buckets:")
    for bucket_id, bucket in buckets.items():
        if len(bucket) > 1 and bucket not in print_buckets:
            print_buckets.append(bucket)
    print_buckets.sort(key=lambda x: x[0])
    for i, bucket in enumerate(print_buckets):
        print(f"Bucket {i+1}:\t", end='')
        for j, doc_idx in enumerate(bucket):
            print(f'{title_to_id[doc_idx]}(index={doc_idx})', end=' ')
            if j != len(bucket) - 1:
                print('- ', end='')
        print()

    min_hash_lsh.jaccard_similarity_test(buckets, summaries)
```

Outputs:

```
Buckets:
Bucket 1:       test1(index=0) - test2(index=1)
Bucket 2:       test7(index=6) - test8(index=7)
Bucket 3:       test13(index=12) - test14(index=13)
Bucket 4:       test15(index=14) - test16(index=15)
Bucket 5:       test17(index=16) - test18(index=17)
Bucket 6:       test19(index=18) - test20(index=19)
Bucket 7:       Apocalypse Now(index=75) - The Post(index=1273)
Bucket 8:       M(index=120) - Wild Strawberries(index=221)
Bucket 9:       Opening Night(index=791) - The Aura(index=1484)
Bucket 10:      The Batman Part II(index=1252) - We Live in
Time(index=2149)
Bucket 11:      The Hunchback of Notre Dame(index=2118) - Return to
Oz(index=2330)
```