Introduction

The objective of our project is to implement a program to find textual similarity documents using shingling, min-hashing, and LSH techniques. The test data is a set of documents suspected to be plagiarized, and the original texts.

Implementation Details

1. **Shingling Class:**

   This class handles generating shingles with **k** parameter as the length of each shingle. Python default `hash()` is used to generate an int representation of a shingle.

2. **CompareSets Class:**

   This class simply calculates the Jaccard similarity by dividing the intersection by the union of the 2 documents' shingles.

3. **MinHashing Class:**

   This class generates a MinHash signature for a given set of shingles, using a specified number of hash functions. The signature is taken as the smallest hash value.

4. **CompareSignatures Class:**

   This class estimate the similarity between two MinHash signatures, by calculating their proportion of matching elements as a Jaccard similarity estimate

5. **Locality-Sensitive Hashing (LSH) Class (Optional):**

   This class takes a signature matrix, similarity threshold, and number of bands. Each band of signatures is hashed by summing the rows, creating groups of documents with matching hashes as candidate pairs, which are then filtered by comparing their MinHash similarity to the threshold.

4. Instructions for Building and Running

- Download data from Kaggle (link)
  - Suspicious documents go to: data/plagiarism/suspicious
  - Original document go to: data/plagiarism/original
- Install the necessary Python package

  ```
  pip install -r requirements.txt
  ```

- Run instruction

  ```
  python main.py
  ```

5. Results and Performance Analysis

1. **Setup**

- A small corpus of documents was selected for testing, consisting of 5 original documents and 5 suspicious documents located in the "data/plagiarism" directory. This corpus provides enough variation to test the similarity detection.
- A shingle length **k=3** was chosen, meaning each document was divided into 3-character sequences to create shingles.
- The signature length for MinHash was set to **100**, balancing computational efficiency and signature uniqueness.
- LSH Parameters:
    - Threshold **s**: A similarity threshold **s = 0.55 ≈ (1/20)^(1/5)** was chosen for the LSH process. Two documents are considered similar if the LSH signature similarity is above this threshold.
    - Bands: The LSH technique was configured with **20** bands, each consisting of **5** rows, to optimise similarity detection.

2. **Results**

- Document length: ~7000 - 100000 words per document.
- Shingling, Jaccard Similarity, MinHash Generation & Comparison: The comparison is pairwise, so we only pick one original document and compare it to 5 suspicious ones. The result shows a relatively similar score between *Jaccard similarity* & *Minhash signature similarity,* which is also a condition to prove the validity of these 2 functions.

```
Total time to process all suspicious files: 0.21 seconds
                                File  Jaccard Similarity  MinHash Signature Similarity
0  suspicious-document01503.txt            0.370008                          0.36
1  suspicious-document01502.txt            0.324698                          0.36
2  suspicious-document01501.txt            0.049118                          0.04
3  suspicious-document01505.txt            0.414427                          0.41
4  suspicious-document01504.txt            0.528356                          0.54      You, 49
```

- With LSH, we collected Minhash signatures for all documents, the LSH class was run to find similar document pairs. LSH reduced the computational complexity by narrowing down potential matches based on the configured threshold and bands.

  From Jaccard & Minhash example: we compare document **source-document01503** to the rest of the suspicious docs, and **suspicious-document01504** returns the highest similarity. For LSH, we expect to see this pair appear again. Based on the trial run, we can see pair (9,4) represent the same document pair in the above example.

```
0: source-document01504.txt
1: source-document01505.txt
2: source-document01501.txt
3: source-document01502.txt
4: source-document01503.txt
5: suspicious-document01503.txt
6: suspicious-document01502.txt
7: suspicious-document01501.txt
8: suspicious-document01505.txt
9: suspicious-document01504.txt

Time to generate signatures for LSH: 0.35 seconds
Time to run LSH and find similar document pairs: 0.00 seconds
Similar document pairs: {(9, 4), (1, 3), (3, 5), (1, 5)}
```