IT350 - Assignmnet-2 (Similar Item set- Min-hashing)

Name : Ritesh Sharma

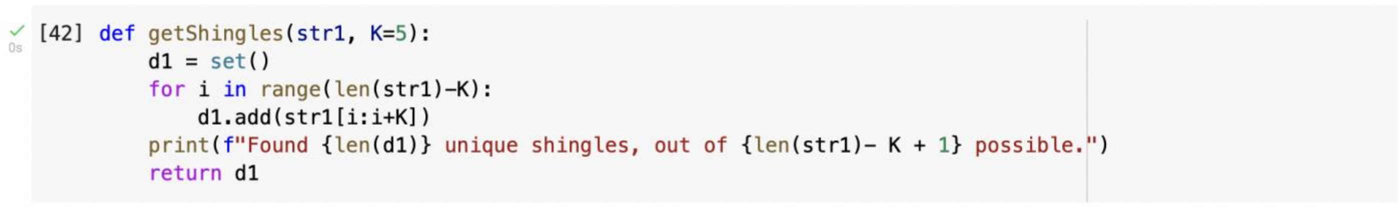
Roll No : 191IT142

Google colab link: https://colab.research.google.com/drive/1yx5KfgZ0w7px0-rSPnHog-KZQAgK2OLF?usp=sharing

Construct several types of k-shingles for all documents.

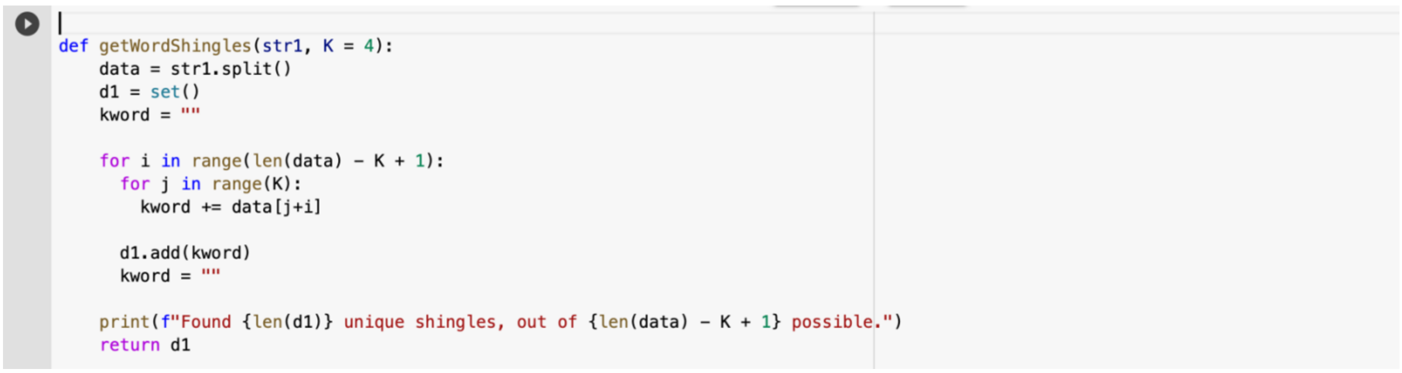
Sub Tasks:
[1] Construct 5-shingles based on characters, for all documents.
[2] Construct 8-shingles based on characters, for all documents.
[3] Construct 4-shingles based on words, for all documents.

Shingles on the basis of characters:



```
[42] def getShingles(str1, K=5):
         d1 = set()
         for i in range(len(str1)-K):
             d1.add(str1[i:i+K])
         print(f"Found {len(d1)} unique shingles, out of {len(str1)- K + 1} possible.")
         return d1
```

Shingles on basis of words:



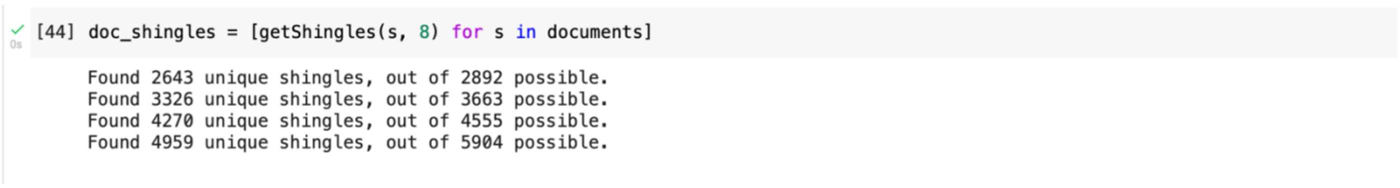
```
def getWordShingles(str1, K = 4):
    data = str1.split()
    d1 = set()
    kword = ""

    for i in range(len(data) - K + 1):
      for j in range(K):
        kword += data[j+i]

      d1.add(kword)
      kword = ""

    print(f"Found {len(d1)} unique shingles, out of {len(data) - K + 1} possible.")
    return d1
```

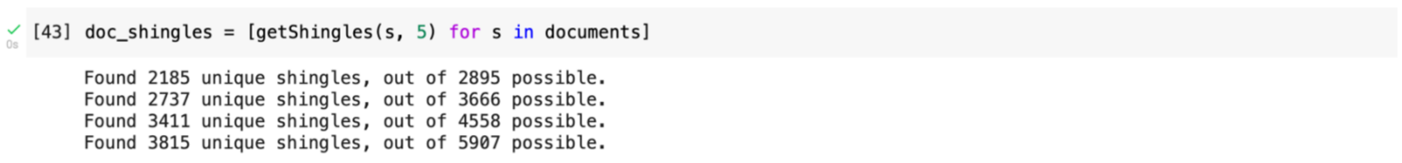
A: How many distinct shingles are there for each document with each type of shingle?
Distinct shingles for k=8 (character based)



```
[44] doc_shingles = [getShingles(s, 8) for s in documents]

     Found 2643 unique shingles, out of 2892 possible.
     Found 3326 unique shingles, out of 3663 possible.
     Found 4270 unique shingles, out of 4555 possible.
     Found 4959 unique shingles, out of 5904 possible.
```

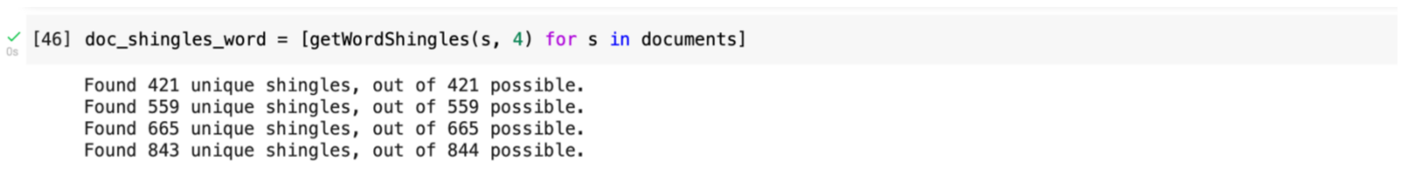
Distinct shingles for k=5 (character based):



```
[43] doc_shingles = [getShingles(s, 5) for s in documents]

     Found 2185 unique shingles, out of 2895 possible.
     Found 2737 unique shingles, out of 3666 possible.
     Found 3411 unique shingles, out of 4558 possible.
     Found 3815 unique shingles, out of 5907 possible.
```

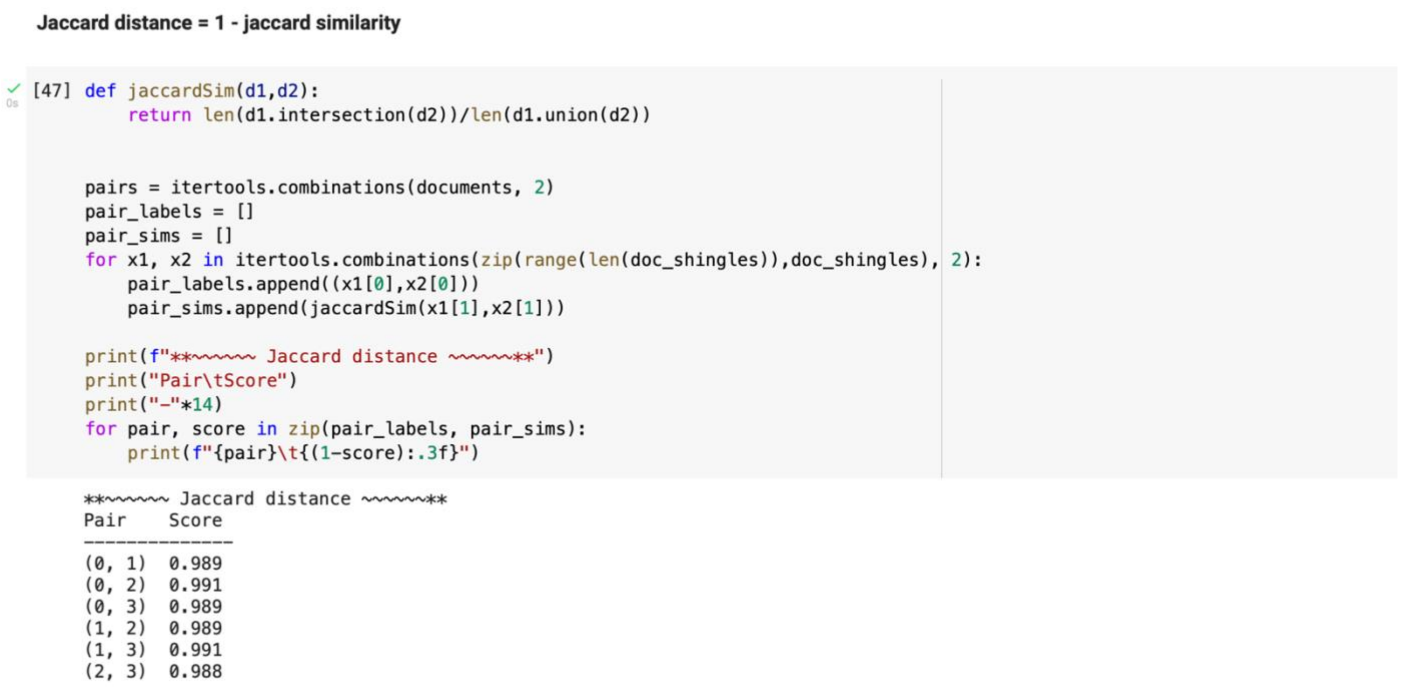
Distinct shingles for k=4 (word based):



```
[46] doc_shingles_word = [getWordShingles(s, 4) for s in documents]

     Found 421 unique shingles, out of 421 possible.
     Found 559 unique shingles, out of 559 possible.
     Found 665 unique shingles, out of 665 possible.
     Found 843 unique shingles, out of 844 possible.
```

B: Compute the Jaccard distance between all pairs of documents for each type of shingling.



Jaccard distance = 1 - jaccard similarity

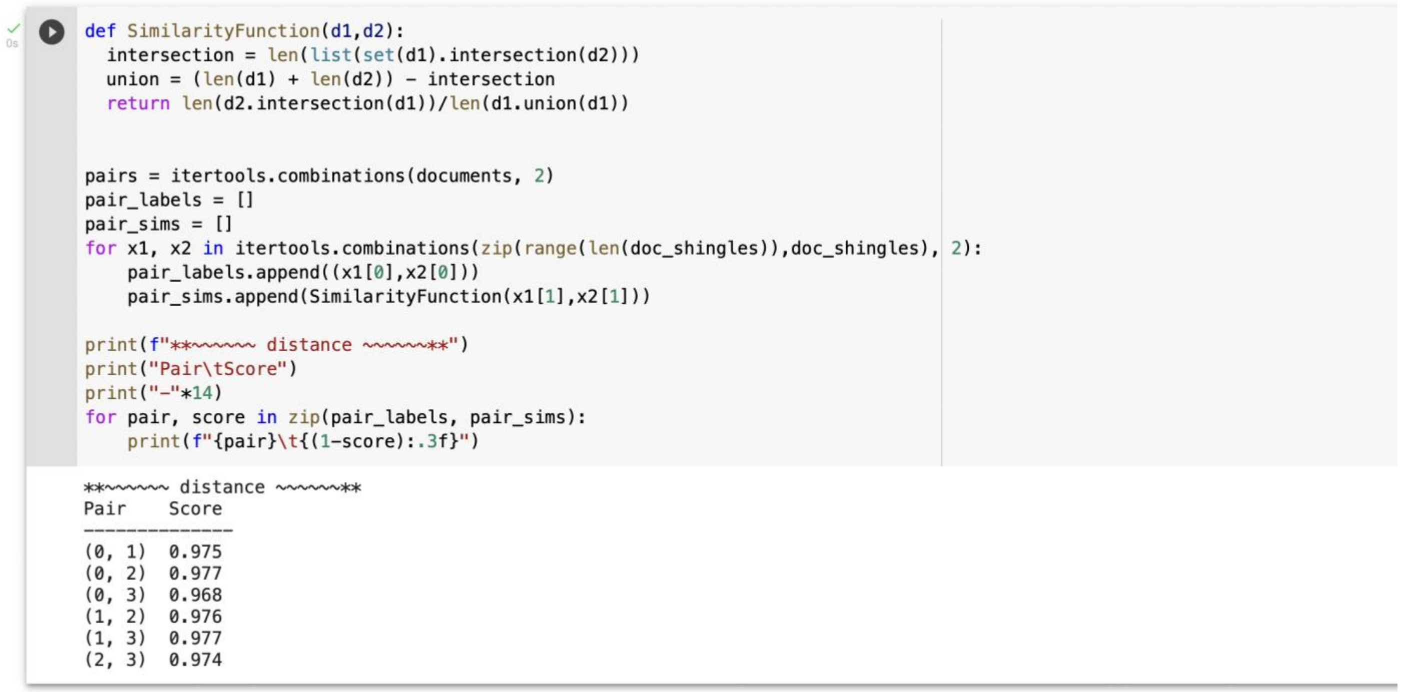
```
[47] def jaccardSim(d1,d2):
         return len(d1.intersection(d2))/len(d1.union(d2))


     pairs = itertools.combinations(documents, 2)
     pair_labels = []
     pair_sims = []
     for x1, x2 in itertools.combinations(zip(range(len(doc_shingles)),doc_shingles), 2):
         pair_labels.append((x1[0],x2[0]))
         pair_sims.append(jaccardSim(x1[1],x2[1]))

     print(f"**~~~~~~ Jaccard distance ~~~~~~**")
     print("Pair\tScore")
     print("-"*14)
     for pair, score in zip(pair_labels, pair_sims):
         print(f"{pair}\t{(1-score):.3f}")

     **~~~~~~ Jaccard distance ~~~~~~**
     Pair    Score
     --------------
     (0, 1)  0.989
     (0, 2)  0.991
     (0, 3)  0.989
     (1, 2)  0.989
     (1, 3)  0.991
     (2, 3)  0.988
```

Question C. Change to any Similarity Function (use any recent similarity distance) and check the distance.



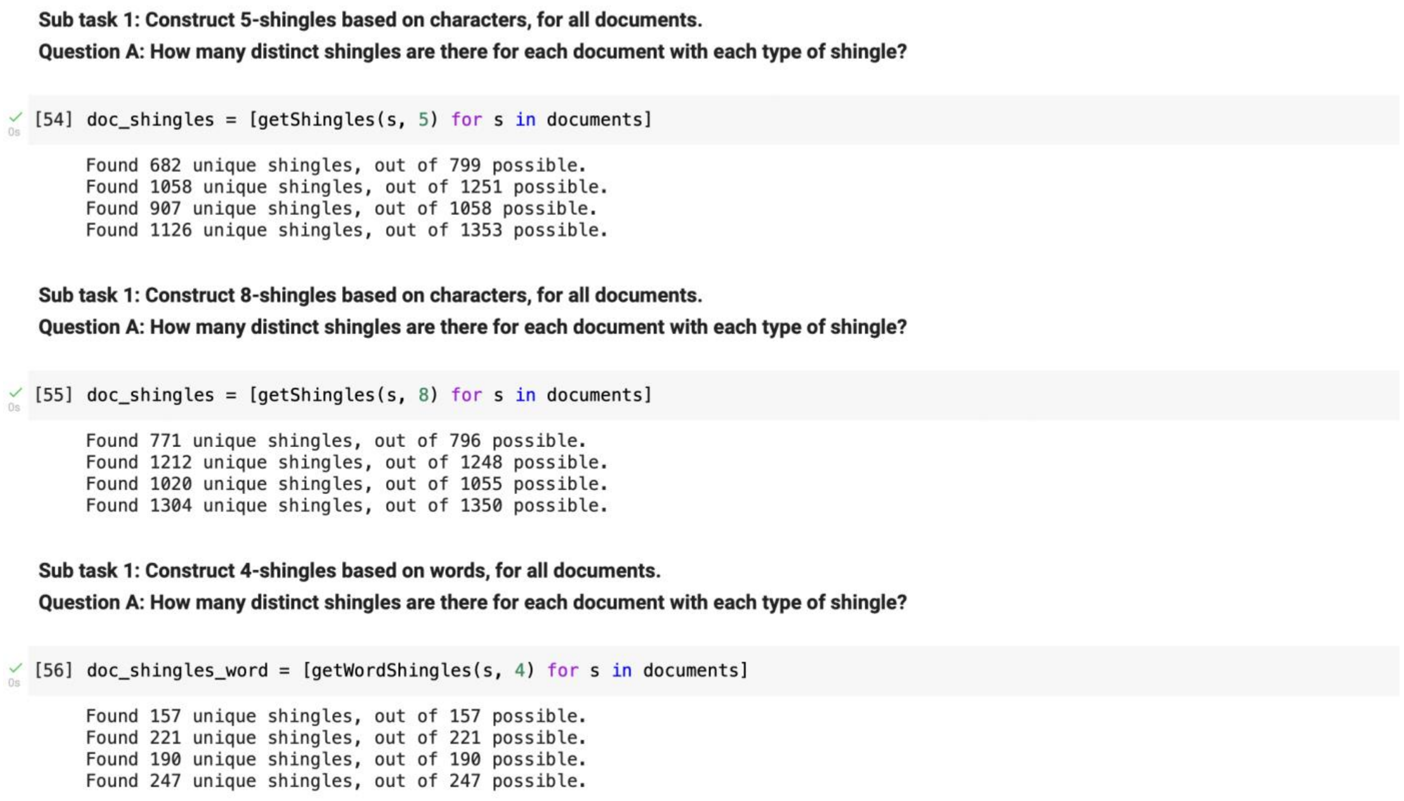
```
def SimilarityFunction(d1,d2):
  intersection = len(list(set(d1).intersection(d2)))
  union = (len(d1) + len(d2)) - intersection
  return len(d2.intersection(d1))/len(d1.union(d1))


pairs = itertools.combinations(documents, 2)
pair_labels = []
pair_sims = []
for x1, x2 in itertools.combinations(zip(range(len(doc_shingles)),doc_shingles), 2):
    pair_labels.append((x1[0],x2[0]))
    pair_sims.append(SimilarityFunction(x1[1],x2[1]))

print(f"**~~~~~~ distance ~~~~~~**")
print("Pair\tScore")
print("-"*14)
for pair, score in zip(pair_labels, pair_sims):
    print(f"{pair}\t{(1-score):.3f}")
```

```
**~~~~~~ distance ~~~~~~**
Pair    Score
--------------
(0, 1)  0.975
(0, 2)  0.977
(0, 3)  0.968
(1, 2)  0.976
(1, 3)  0.977
(2, 3)  0.974
```

D. Try the above all for anyone Indian language.
Indian Language: Hindi



Sub task 1: Construct 5-shingles based on characters, for all documents.
Question A: How many distinct shingles are there for each document with each type of shingle?

```
[54] doc_shingles = [getShingles(s, 5) for s in documents]
```

```
Found 682 unique shingles, out of 799 possible.
Found 1058 unique shingles, out of 1251 possible.
Found 907 unique shingles, out of 1058 possible.
Found 1126 unique shingles, out of 1353 possible.
```

Sub task 1: Construct 8-shingles based on characters, for all documents.
Question A: How many distinct shingles are there for each document with each type of shingle?

```
[55] doc_shingles = [getShingles(s, 8) for s in documents]
```

```
Found 771 unique shingles, out of 796 possible.
Found 1212 unique shingles, out of 1248 possible.
Found 1020 unique shingles, out of 1055 possible.
Found 1304 unique shingles, out of 1350 possible.
```

Sub task 1: Construct 4-shingles based on words, for all documents.
Question A: How many distinct shingles are there for each document with each type of shingle?

```
[56] doc_shingles_word = [getWordShingles(s, 4) for s in documents]
```

```
Found 157 unique shingles, out of 157 possible.
Found 221 unique shingles, out of 221 possible.
Found 190 unique shingles, out of 190 possible.
Found 247 unique shingles, out of 247 possible.
```

Question B: Compute the Jaccard distance between all pairs of documents for each type of shingling.

Jaccard distance = 1 - jaccard similarity

✓ 0s [57]
```python
def jaccardSim(d1,d2):
    return len(d1.intersection(d2))/len(d1.union(d2))


pairs = itertools.combinations(documents, 2)
pair_labels = []
pair_sims = []
for x1, x2 in itertools.combinations(zip(range(len(doc_shingles)),doc_shingles), 2):
    pair_labels.append((x1[0],x2[0]))
    pair_sims.append(jaccardSim(x1[1],x2[1]))

print(f"**~~~~~~~ Jaccard distance ~~~~~~~**")
print("Pair\tScore")
print("-"*14)
for pair, score in zip(pair_labels, pair_sims):
    print(f"{pair}\t{(1-score):.3f}")
```

```
**~~~~~~~ Jaccard distance ~~~~~~~**
Pair    Score
--------------
(0, 1)  0.974
(0, 2)  0.988
(0, 3)  0.993
(1, 2)  0.983
(1, 3)  0.994
(2, 3)  0.984
```

Question C. Change to any Similarity Function (use any recent similarity distance) and check the distance.

✓ 0s [58]
```python
def SimilarityFunction(d1,d2):
  intersection = len(list(set(d1).intersection(d2)))
  union = (len(d1) + len(d2)) - intersection
  return len(d2.intersection(d1))/len(d1.union(d1))


pairs = itertools.combinations(documents, 2)
pair_labels = []
pair_sims = []
for x1, x2 in itertools.combinations(zip(range(len(doc_shingles)),doc_shingles), 2):
    pair_labels.append((x1[0],x2[0]))
    pair_sims.append(SimilarityFunction(x1[1],x2[1]))

print(f"**~~~~~~~ distance ~~~~~~~**")
print("Pair\tScore")
print("-"*14)
for pair, score in zip(pair_labels, pair_sims):
    print(f"{pair}\t{(1-score):.3f}")
```

```
**~~~~~~~ distance ~~~~~~~**
Pair    Score
--------------
(0, 1)  0.934
(0, 2)  0.973
(0, 3)  0.981
(1, 2)  0.969
(1, 3)  0.988
(2, 3)  0.964
```

Question E: Build a min hash signature for the above experiment and provide your conclusions for the entire experiment

```python
class HashManager():
    def __init__(self, shingle_dict):
        self.shingle_dict = shingle_dict
        self.N = len(shingle_dict)
        self.params = None

    def _initParams(self, n_sig):
        self.params = np.random.randint(self.N, size=[n_sig,2])

    def _permuteRow(self, row):
        return (self.params@np.array([1,row]))%self.N

    def __call__(self, docs, n_sig, init=True):

        if self.params is None or len(self.params) != n_sig or init:
            self._initParams(n_sig)


        sig = np.full((n_sig, len(docs)), np.inf)


        for j, doc in enumerate(docs):
            for shingle in doc:
                orig_row = shingle_dict[shingle]
                curr_col = self._permuteRow(orig_row)
                sig[:,j] = np.minimum(sig[:,j],curr_col)
        return sig.astype(int)
```

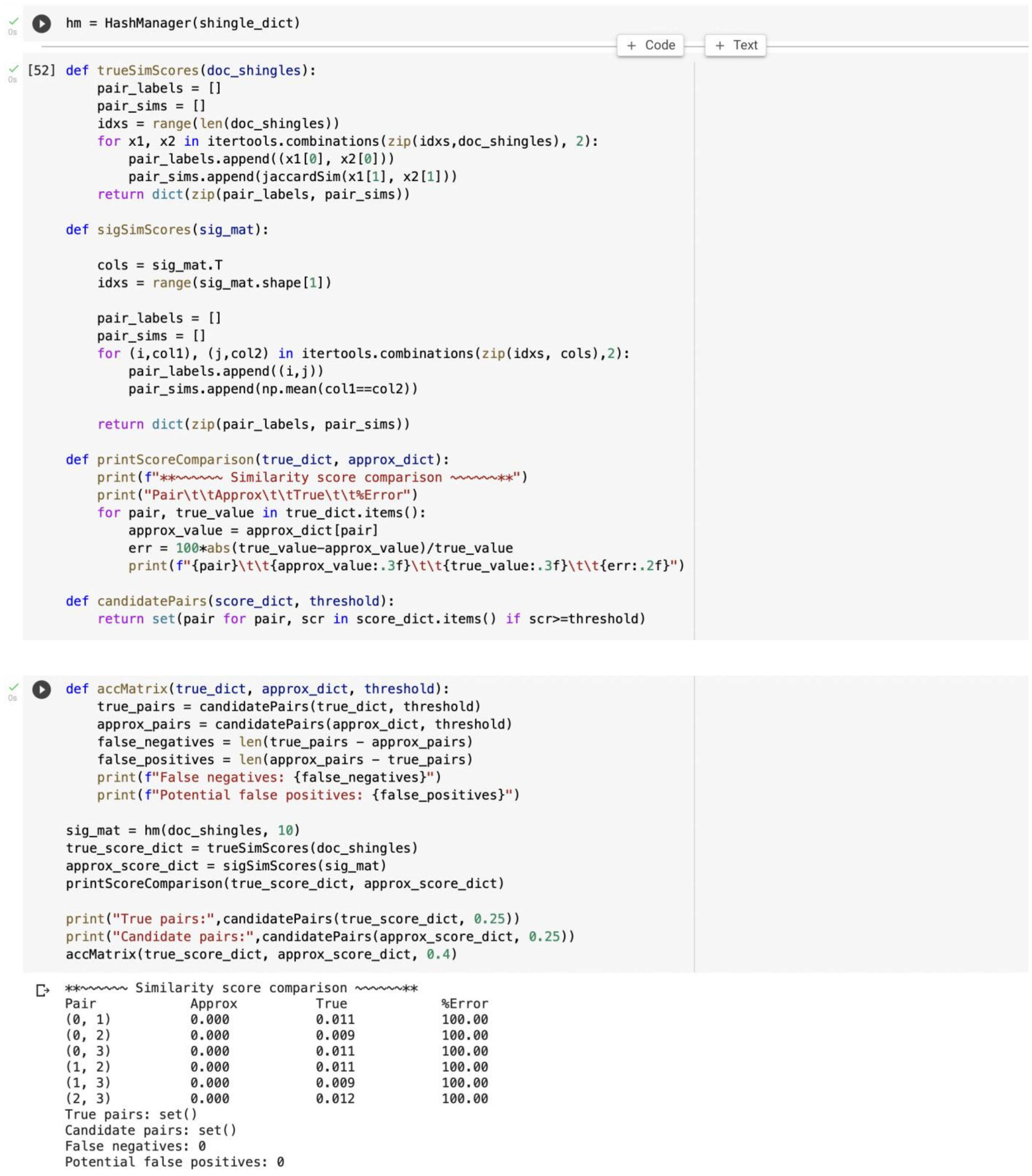
```python
try:
    print("Initialization test: ", end="")
    hm = HashManager(shingle_dict)
    print("passed")

    print("Set parameters to right size: ", end="")
    hm._initParams(n_sig=4)
    assert(hm.params.shape == (4,2))
    print("passed")

    print("Permuting a row integer returns array: ", end="")
    curr_col = hm._permuteRow(3)
    assert(curr_col.shape == (4,))
    print("passed")

    print("Compute minhashed signature matrix: ", end="")
    hm(doc_shingles, 4)
    print("passed")
except Exception as e:
    print("failure")
    print(e.args)
```

```
Initialization test: passed
Set parameters to right size: passed
Permuting a row integer returns array: passed
Compute minhashed signature matrix: passed
```



```
hm = HashManager(shingle_dict)
```

```
[52] def trueSimScores(doc_shingles):
         pair_labels = []
         pair_sims = []
         idxs = range(len(doc_shingles))
         for x1, x2 in itertools.combinations(zip(idxs,doc_shingles), 2):
             pair_labels.append((x1[0], x2[0]))
             pair_sims.append(jaccardSim(x1[1], x2[1]))
         return dict(zip(pair_labels, pair_sims))

     def sigSimScores(sig_mat):

         cols = sig_mat.T
         idxs = range(sig_mat.shape[1])

         pair_labels = []
         pair_sims = []
         for (i,col1), (j,col2) in itertools.combinations(zip(idxs, cols),2):
             pair_labels.append((i,j))
             pair_sims.append(np.mean(col1==col2))

         return dict(zip(pair_labels, pair_sims))

     def printScoreComparison(true_dict, approx_dict):
         print(f"**~~~~~~ Similarity score comparison ~~~~~~**")
         print("Pair\t\tApprox\t\tTrue\t\t%Error")
         for pair, true_value in true_dict.items():
             approx_value = approx_dict[pair]
             err = 100*abs(true_value-approx_value)/true_value
             print(f"{pair}\t\t{approx_value:.3f}\t\t{true_value:.3f}\t\t{err:.2f}")

     def candidatePairs(score_dict, threshold):
         return set(pair for pair, scr in score_dict.items() if scr>=threshold)
```

```
def accMatrix(true_dict, approx_dict, threshold):
    true_pairs = candidatePairs(true_dict, threshold)
    approx_pairs = candidatePairs(approx_dict, threshold)
    false_negatives = len(true_pairs - approx_pairs)
    false_positives = len(approx_pairs - true_pairs)
    print(f"False negatives: {false_negatives}")
    print(f"Potential false positives: {false_positives}")

sig_mat = hm(doc_shingles, 10)
true_score_dict = trueSimScores(doc_shingles)
approx_score_dict = sigSimScores(sig_mat)
printScoreComparison(true_score_dict, approx_score_dict)

print("True pairs:",candidatePairs(true_score_dict, 0.25))
print("Candidate pairs:",candidatePairs(approx_score_dict, 0.25))
accMatrix(true_score_dict, approx_score_dict, 0.4)
```

```
**~~~~~~ Similarity score comparison ~~~~~~**
Pair            Approx          True            %Error
(0, 1)          0.000           0.011           100.00
(0, 2)          0.000           0.009           100.00
(0, 3)          0.000           0.011           100.00
(1, 2)          0.000           0.011           100.00
(1, 3)          0.000           0.009           100.00
(2, 3)          0.000           0.012           100.00
True pairs: set()
Candidate pairs: set()
False negatives: 0
Potential false positives: 0
```

Conclusion:
By above experiment we can conclude that tasks like finding duplicate or similar documents etc becomes scalable since brute force solution is extremely time consuming or slow. Right representation of document for efficient similarity comparison. Valid for any language (I use Hindi as Indian language with given documents)