Part 2

We obtained the list of references from all the files from the previous assignment and saved it as a dictionary into a text file called ‘dictionary.txt’.
Then load it into a python dictionary using the JSON load function.

```
js = {}
with open('dictionary.txt') as f:
    data = f.read()

js = json.loads(data)
```

Then for every file, we put the list of references for that file into a list. Ultimately, we obtain a nested list where every inner list contains references for that file.

```
l1 = []
for i, j in js.items():
    if len(j)!=0:
        l1.append(j)
```

Perform shingling for each set of references. We first merge all the references for one file into one string and then do word-wise shingling

```
def shingles(l1, k):
    dataset = []
    universal_list =[]

    count=0
    for document in l1:
        reference = (" ".join(document))
        ele = reference
        temp =[]

        single_ref_list = list(ele.split(" "))

        for ptr in range(0, len(single_ref_list)-k+1):
            temp_val = single_ref_list[ptr:(ptr+k):1]
            temp_val = (" ".join(temp_val))
            temp.append(temp_val)

        dataset.append(temp)
        print("dataset iteration number ======> " + str(count))
        count = count+1


    print()
    print('----------------------------')

    return dataset
```

PART 1: Using LSH, find all the candidate document pairs and plot a histogram by varying the similarity threshold and keeping the shingling parameter as fixed.

1. For this question, we have set the shingling parameter to 2

```
dataset1 = shingles(l1, 2)
```

2. Convert the shingles of the file into a set to obtain unique shingles for every document.

```
dataset = []
for i in dataset1:
    dataset.append(set(i))
```

3. Generate hash functions. The num_perm parameter indicates the number of thresholds.

```
hashFunctions = []
for i in range(0, len(l1)):
    hashFunctions.append(MinHash(num_perm=128))
```

4. Update the hash functions per the shingles for that file.
Utf8 is a 'Universal Encoding Standard Transformation Format'. And is 8 bit in size

```
for i in range(0, len(l1)):
    hashFunctions[i]= MinHash(num_perm=128)
    for token in dataset[i]:
        hashFunctions[i].update(token.encode('utf8'))
```

5. The count_list is a dictionary that stores the number of candidate pairs for each threshold value. The matched_list stores all the possible candidate pairs for each threshold in a key-value pair format. The thresh_list is a list that stores the possible threshold values which we are going to test in this question. We will start from an initial threshold of 25% and then gradually increase 2% in each iteration till we reach 61%

```
count_list ={}
matched_dict =[]
thresh_list = []
c= 0.25
while c <=0.62:
    val = round(c, 2)
    thresh_list.append(val)
    c= c+ 0.02
```

6. Initialize the threshold in every iteration. Insert the hash functions into LSH. Then query each hash function to find the references that match the current one.

```
for thr in thresh_list:
    thres = thr
    lsh = MinHashLSH(threshold=thres, num_perm=128)

    for i in range(0, len(hashFunctions)):
            temp = i
            lsh.insert(temp, hashFunctions[i])

    dict ={}
    for i in range(0, len(hashFunctions)):
        result = lsh.query(hashFunctions[i])
        if len(result)>1:
            result.remove(i)
            dict[i] = result
```

7. But if ‘A’ is similar to ‘B’ then ‘B’ is also similar to ‘A’. Hence, some candidate pairs are duplicated and must be removed from the list of candidate pairs. In the end we will get some empty sets which we have to remove as they are of no issue.

```
key = dict.keys()

for j, k in dict.items():

    val_list = dict[j]
    val_list2 = set(val_list)
    for i in k:
        if i in key:
            val = set(dict[i])
            val = val - val_list2
            val.remove(j)
            dict[i]=val
```

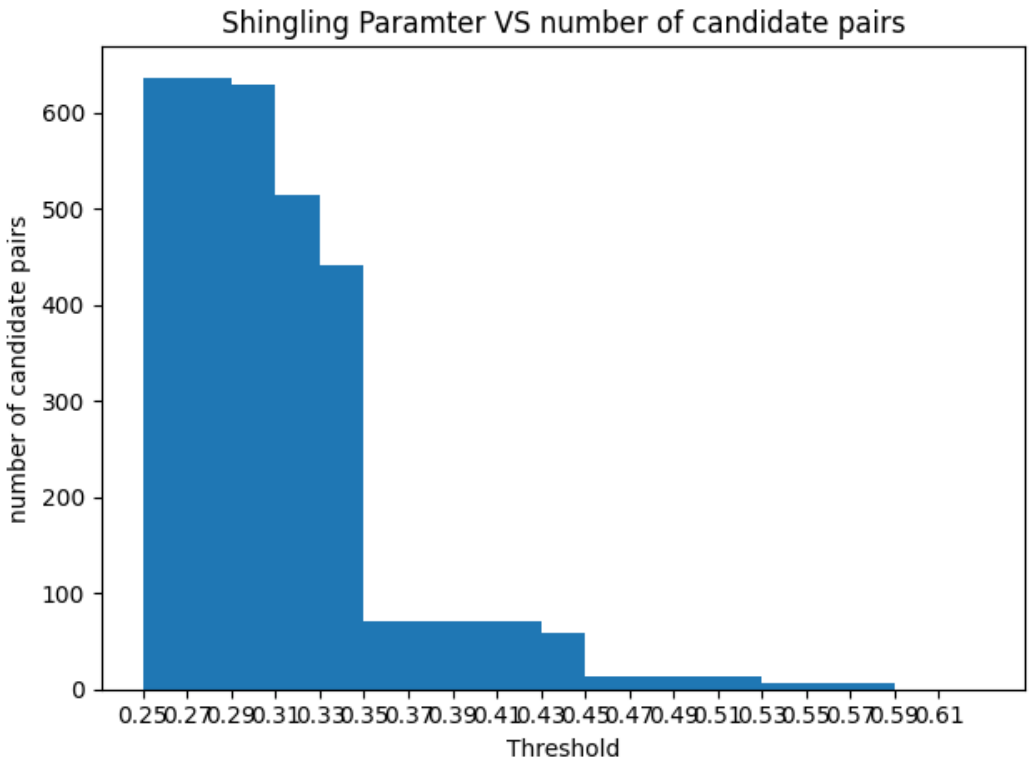
```
remove_list = []

for i, j in dict.items():
    if j == set():
        remove_list.append(i)

for i in remove_list:
    dict.pop(i)

val1 ={}
val1[thr] = dict
matched_dict.append(val1)

candidate_pair = len(dict.keys())
count_list[thres] = candidate_pair

print(thr)
```

8. Below is the dictionary which contains threshold as key and count of candidate pairs for that threshold

```
print(count_list)
```

```
{0.25: 637, 0.27: 637, 0.29: 629, 0.31: 515, 0.33: 442, 0.35: 70, 0.37: 70, 0.39: 70, 0.41: 70, 0.43: 58, 0.45: 13, 0.4
7: 13, 0.49: 13, 0.51: 13, 0.53: 6, 0.55: 6, 0.57: 6, 0.59: 0, 0.61: 0}
```

9. Plot a histogram with the above-obtained candidate pairs



PART 2: Using LSH, find all the candidate document pairs and plot a histogram by varying the shingling parameter and keeping the similarity threshold fixed.

1. Here we have kept the shingling parameter from 2 to 10, increasing the parameter by 1 in every iteration. The rest of the entire process remains the same only the shingling is performed again and again. The threshold is fixed at 35%. In the end, the count_list contains the dictionary with keys being the shingling parameter and the value being the count of the candidate pairs for that shingling parameter.
   This is the outcome.

```
count_list
```

```
{2: 70, 3: 29, 4: 10, 5: 5, 6: 5, 7: 4, 8: 1, 9: 1, 10: 2}
```

   Let us look at the histogram: