# Plagiarism Check tool

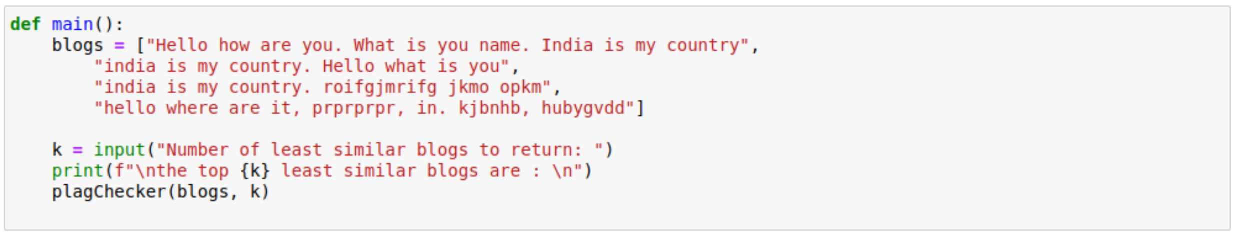
**Task:** To design a blog search tool that returns the top K least similar blogs.

**Pre-requisites:**

- Basic knowledge about python.
- Knowledge about NLP concepts like n-grams, edit-distance.

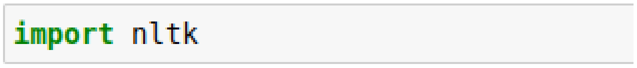
**Implementation Plan:**

- We shall create a function that gets access to all the available blogs. For simplicity purposes, we already hardcoded the blogs and their text inside the main function.



```python
def main():
    blogs = ["Hello how are you. What is you name. India is my country",
        "india is my country. Hello what is you",
        "india is my country. roifgjmrifg jkmo opkm",
        "hello where are it, prprprpr, in. kjbnhb, hubygvdd"]

    k = input("Number of least similar blogs to return: ")
    print(f"\nthe top {k} least similar blogs are : \n")
    plagChecker(blogs, k)
```

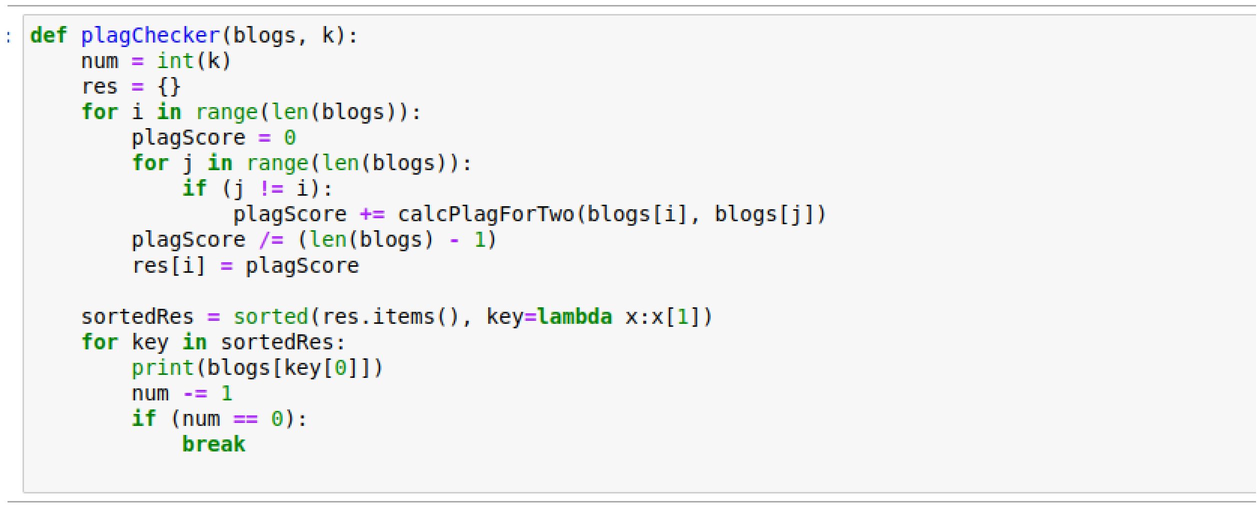
- We need to install and import all the necessary packages required for the implementation of the tool. We will be using only "NLTK" library in the code.



```python
import nltk
```

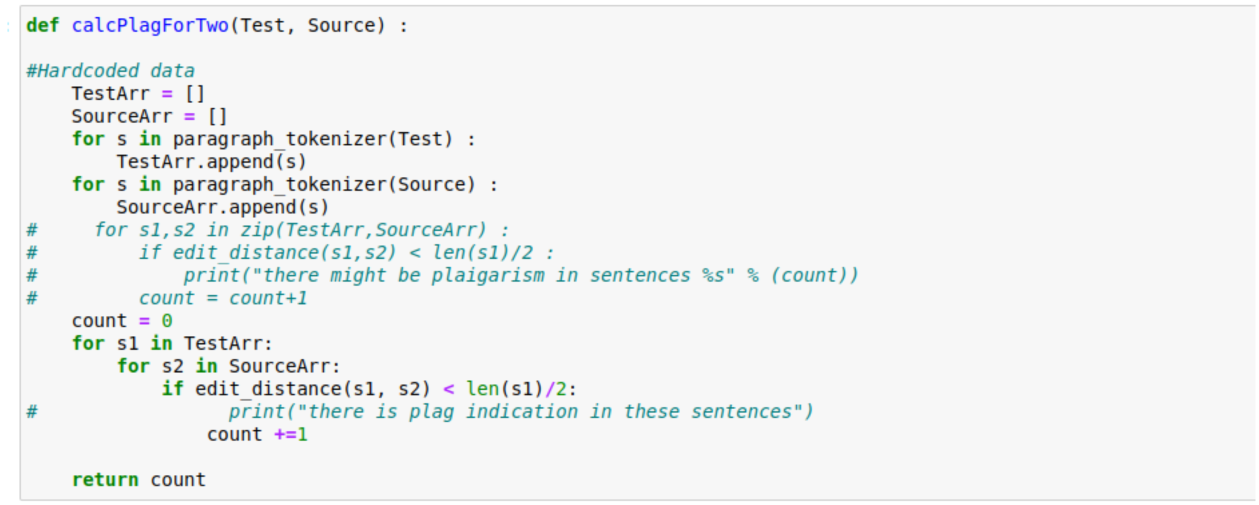
- The following is the process for finding the k least similar blogs:
    - For finding how different and unique a blog is we will have to compare it with every other blog.
    - Comparison between the blogs will happen on the basis of how many similar sentences they have in them.
    - We will be using the concept of edit-distance to find the similarity of the sentences.
    - The plagiarism score for how similar a blog is to another is the total number of sentences that are similar in the blogs divided by the total number of sentences in that particular blog.
    - The total plagiarism score for a blog can be calculated after finding the mean of the plagiarism scores of that blog with every other blog. This way we can quantify the similarity of a blog with other blogs which will help us in finding our solution.

- The **plagChecker** function takes 2 arguments, the list containing our blogs and the value “K” which is the number of blogs we have to return in the end which are the least similar.



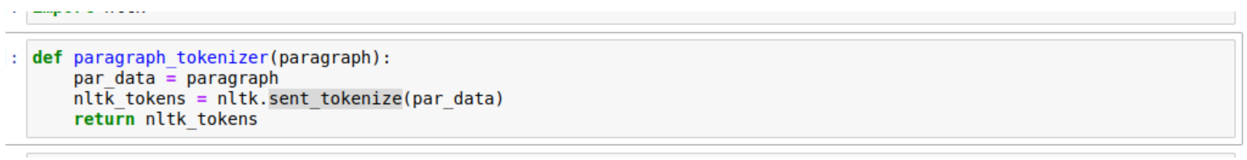
```
def plagChecker(blogs, k):
    num = int(k)
    res = {}
    for i in range(len(blogs)):
        plagScore = 0
        for j in range(len(blogs)):
            if (j != i):
                plagScore += calcPlagForTwo(blogs[i], blogs[j])
        plagScore /= (len(blogs) - 1)
        res[i] = plagScore

    sortedRes = sorted(res.items(), key=lambda x:x[1])
    for key in sortedRes:
        print(blogs[key[0]])
        num -= 1
        if (num == 0):
            break
```

- **plagScore** variable is used to find the final quantified value of plagiarism for a particular blog. **calcPlagForTwo** function is responsible for finding the plagiarism score for a particular blog when compared to another.



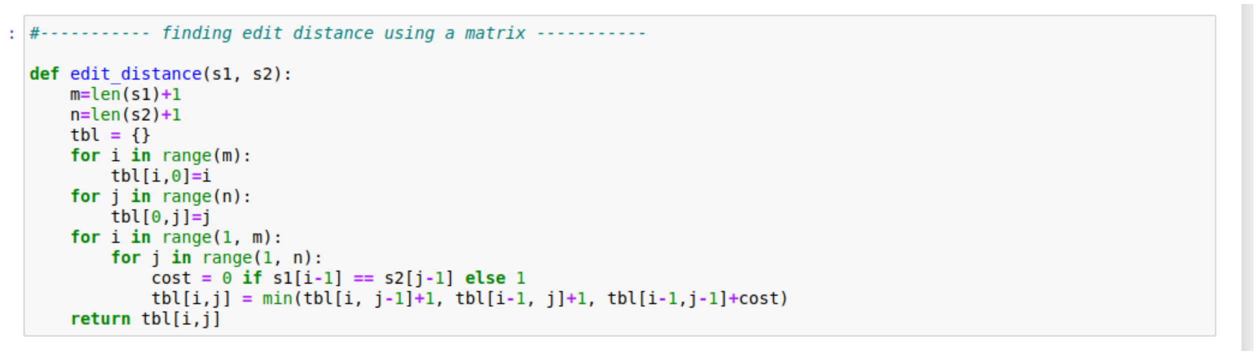
```
def calcPlagForTwo(Test, Source) :

#Hardcoded data
    TestArr = []
    SourceArr = []
    for s in paragraph_tokenizer(Test) :
        TestArr.append(s)
    for s in paragraph_tokenizer(Source) :
        SourceArr.append(s)
#     for s1,s2 in zip(TestArr,SourceArr) :
#         if edit_distance(s1,s2) < len(s1)/2 :
#             print("there might be plaigarism in sentences %s" % (count))
#         count = count+1
    count = 0
    for s1 in TestArr:
        for s2 in SourceArr:
            if edit_distance(s1, s2) < len(s1)/2:
#                 print("there is plag indication in these sentences")
                count +=1

    return count
```

- The **calPlagForTwo** function takes 2 arguments, Test and Source. Test is the blog for which we are finding the plagiarism for and Source is the blog with which we are comparing the test to.  The first step is to break the blogs into multiple sentences which later will be compared using the edit distance concept. We use the method defined in the **nltk library** which is **sent_tokenize**(). This method is used to breakdown a given text into sentences.



```python
def paragraph_tokenizer(paragraph):
    par_data = paragraph
    nltk_tokens = nltk.sent_tokenize(par_data)
    return nltk_tokens
```

- The **paragraph_tokenizer** function is being used to breakdown the blog into multiple sentences which are being stored in the **TestArr** and **SourceArr** lists. To check how similar the sentences are from each lists, we use the **edit_distance** function.



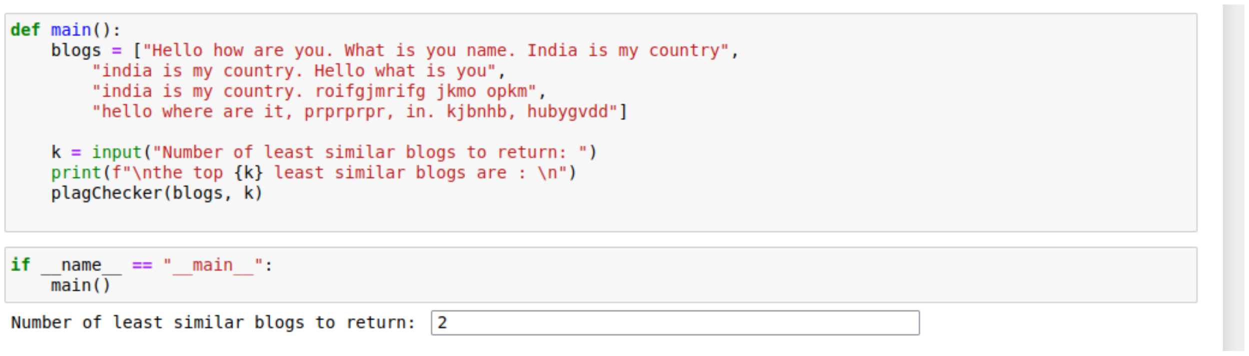
```python
#----------- finding edit distance using a matrix -----------

def edit_distance(s1, s2):
    m=len(s1)+1
    n=len(s2)+1
    tbl = {}
    for i in range(m):
        tbl[i,0]=i
    for j in range(n):
        tbl[0,j]=j
    for i in range(1, m):
        for j in range(1, n):
            cost = 0 if s1[i-1] == s2[j-1] else 1
            tbl[i,j] = min(tbl[i, j-1]+1, tbl[i-1, j]+1, tbl[i-1,j-1]+cost)
    return tbl[i,j]
```

- Edit distance is a way to measure how similar two text samples are. We implement **Levenshtein edit distance** which is a dynamic programming method. If the edit distance returned is greater than half the length of the sentence from the **TestArr**, we can consider the sentences to be similar. Following this, we can find the count of the number of sentences which are similar in the two blogs. The number of similar sentences divided by the length of the **TestArr** gives us the score of plagiarism between the two blogs. This process can be repeated until  we find the scores for a blog with every other blog.  The final score is the average of the above specified scores.

- **sortedRes** variable in the **plagChecker** function is used to store the blogs in a sorted manner based on their plagiarism scores. Now we can display the top K least similar blogs by simply iterating over the **sortedRes** list.

- The following input generated the following output:

**Input:**

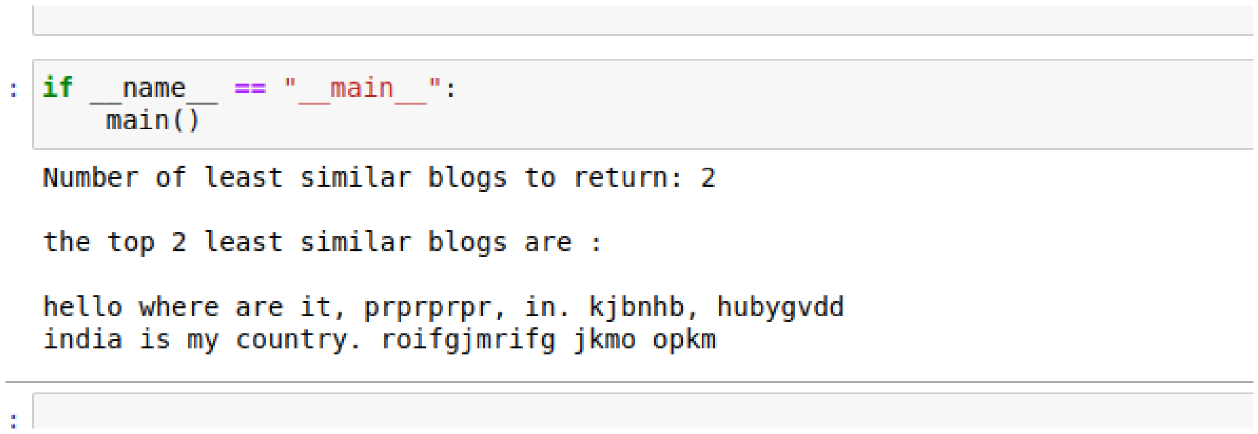


```
def main():
    blogs = ["Hello how are you. What is you name. India is my country",
        "india is my country. Hello what is you",
        "india is my country. roifgjmrifg jkmo opkm",
        "hello where are it, prprprpr, in. kjbnhb, hubygvdd"]

    k = input("Number of least similar blogs to return: ")
    print(f"\nthe top {k} least similar blogs are : \n")
    plagChecker(blogs, k)
```

```
if __name__ == "__main__":
    main()
```

```
Number of least similar blogs to return: 2
```

**Output:**



```
if __name__ == "__main__":
    main()
```

```
Number of least similar blogs to return: 2

the top 2 least similar blogs are :

hello where are it, prprprpr, in. kjbnhb, hubygvdd
india is my country. roifgjmrifg jkmo opkm
```

**References:**

1. https://en.wikipedia.org/wiki/Levenshtein_distance
2. Professor Rahul Roy's Notes