```
import nltk
nltk.download('punkt')
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
from nltk.tag import pos_tag
nltk.download('averaged_perceptron_tagger')
text=input("Enter text:")
print(text)

words=word_tokenize(text)
print("tokenize word")
print(words)

words=[word.lower() for word in words]

fdist=FreqDist(words)
print("word frequency")
for word,freq in fdist.items():
    print(f"{word}:{freq} ")

stop_words=set(stopwords.words("english"))
filtered_words=[words for word in words if word.casefold() not in stop_words]
print(filtered_words)

pos_tags=pos_tag(words)
print(pos_tags)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
Enter text:hhhhhhh
hhhhhhh
tokenize word
['hhhhhhh']
word frequency
hhhhhhh:1
[['hhhhhhh']]
[('hhhhhhh', 'NN')]
```

```
def print_board(board):
    for row in board:
        print(" ".join(row))

def is_safe(board,row,col):
    for i in range(col):
        if board[row][i]=="Q":
            return False

    for i,j in zip(range(row,-1,-1),range(col,-1,-1)):
```

```
            if board[i][j]=='Q':
                return False

        for i,j in zip(range(row,len(board),1),range(col,-1,-1)):
            if board[i][j]=='Q':
                return False

        return True

    def solve(board,col):
        if col>=len(board):
            return True

        for i in range(len(board)):
            if is_safe(board,i,col):
                board[i][col]='Q'
                if solve(board,col+1):
                    return True

            board[i][col]='.'
        return False

    n=int(input("Enter number of queens:"))
    board=[["."for i in range(n)]for j in range(n)]

    if solve(board,0):
        print_board(board)

    else:
        print("solution not found")
```

```
 Enter number of queens:4
 . . Q .
 Q . . .
 . . . Q
 . Q . .
```

```
    max1=int(input("Enter value of max1:"))
    max2=int(input("Enter value of max2:"))
    fill=int(input("Enter value of fill:"))

    def pour(jug1,jug2):
      print("%d \t %d"%(jug1,jug2))
      if jug2==fill:
        return

      elif jug1!=0 and jug2==0:
        pour(0,jug1)

      elif jug1==fill:
        pour(jug1,0)

      elif jug1<max1:
        pour(max1,jug2)

      elif jug1<(max2-jug2):
        pour(0,jug1+jug2)
```

```
  else:
    pour(jug1-(max2-jug2),(max2-jug2)+jug2)

print("jug1\t jug2")
pour(0,0)
```

```
     Enter value of max1:3
     Enter value of max2:4
     Enter value of fill:2
     jug1     jug2
     0        0
     3        0
     0        3
     3        3
     2        4
     2        0
     0        2
```

```
import sys
def nearest_neighbor(curr, unvisited, dist_matrix):
  nearest = sys.maxsize
  neighbor = None
  for city in unvisited:
      if dist_matrix[curr][city] < nearest:
        nearest = dist_matrix[curr][city]
        neighbor = city
  return neighbor, nearest

def tsp_nn(dist_matrix):
    n=len(dist_matrix)
    tour=[0]*5
    unvisited=set(range(1,n))
    curr_city=0

    for i in range(1,n):
        next_city,dist=nearest_neighbor(curr_city,unvisited,dist_matrix)
        tour[i]=next_city
        curr_city=next_city
        unvisited.remove(next_city)


    tour[0]=0
    cost=sum(dist_matrix[tour[i]][tour[i+1]] for i in range(n-1))
    cost+=dist_matrix[tour[n-1]][tour[0]]
    return tour,cost

import numpy as np
rows=int(input("enter rows:"))
column=int(input("enter column:"))
print("enter matrix:")
elements=list(map(int,input().split()))

dist_matrix=np.array(elements).reshape(rows,column)
print(dist_matrix)

tour,cost=tsp_nn(dist_matrix)
```

```
print("tour:",tour)
print("cost:",cost)
```

```
enter rows:4
enter column:4
enter matrix:
0 5 15 4 5 0 35 25 15 35 0 30 4 25 30 0
[[ 0  5 15  4]
 [ 5  0 35 25]
 [15 35  0 30]
 [ 4 25 30  0]]
tour: [0, 3, 1, 2, 0]
cost: 79
```