Python: without numpy or sklearn

Q1: Given two matrices please print the product of those two matrices

```
Ex 1: A   = [[1 3 4]
             [2 5 7]
             [5 9 6]]
      B   = [[1 0 0]
             [0 1 0]
             [0 0 1]]
      A*B = [[1 3 4]
             [2 5 7]
             [5 9 6]]

Ex 2: A   = [[1 2]
             [3 4]]
      B   = [[1 2 3 4 5]
             [5 6 7 8 9]]
      A*B = [[11 14 17 20 23]
             [23 30 36 42 51]]

Ex 3: A   = [[1 2]
             [3 4]]
      B   = [[1 4]
             [5 6]
             [7 8]
             [9 6]]
      A*B =Not possible
```

```
A  = [[1, 2],
      [3, 4]]
B   = [[1, 2, 3, 4, 5],
      [5, 6, 7, 8, 9]]

def matrix_mul(A, B):
    # write your code
    if(len(A) == 0 or len(B)==0):
```

```
        return('Not possible')

    rows_A, rows_B =  len(A), len(B)
    cols_A, cols_B = len(A[0]), len(B[0])

    # Checking if number of columns in A is same as rows in B (if multiplication be
    if(cols_A != rows_B):
      return('Not possible')
    # initialize a result matrix of size rows_a * cols_b with 0
    res = [ [ 0 for i in range(cols_B) ] for j in range(rows_A) ] # https://www.gee

    # logic for multiplication of matrix
    for i in range(rows_A):
      for j in range(cols_B):
        for k in range(rows_B):
          res[i][j] += (A[i][k] * B[k][j])
          # print(res)
    return(res)

matrix_mul(A, B)
```

```
 [[11, 14, 17, 20, 23], [23, 30, 37, 44, 51]]
```

Q2: Select a number randomly with probability proportional to its magnitude from the given array of n elements

consider an experiment, selecting an element from the list A randomly with probability proportional to its magnitude. assume we are doing the same experiment for 100 times with replacement, in each experiment you will print a number that is selected randomly from A.

```
Ex 1: A = [0 5 27 6 13 28 100 45 10 79]
let f(x) denote the number of times x getting selected in 100 experiments.
f(100) > f(79) > f(45) > f(28) > f(27) > f(13) > f(10) > f(6) > f(5) > f(0)
```

```
# concepts understood from https://medium.com/analytics-vidhya/pure-python-exercise
from random import uniform

def pick_a_number_from_list(A):
    cumulative_sum = []
    num = 0
    # building a cumulative sum array
    for ele in A:
      cumulative_sum.append(num+ele)
      num+=ele
    print(cumulative_sum)
    # upper limit is the last numer / total sum
    u_limit = cumulative_sum[-1]

    # selecting random number between 0 and upper limit

    ran = uniform(0, u_limit)
```

```
    ran = uniform(0, u_limit)
    index = 0
    # getting index of number
    for i in range(len(cumulative_sum)):
      if(cumulative_sum[i] > ran):
        index = i
        break

    return A[i]

A = [0, 5, 27, 6, 13, 28, 100, 45, 10, 79]

def sampling_based_on_magnitued():
    for i in range(1,100):
        number = pick_a_number_from_list(A)
        print(number)

sampling_based_on_magnitued()
```

```
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
13
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
10
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
13
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
10
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
```

```
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
```

Q3: Replace the digits in the string with

consider a string that will have digits in that, we need to remove all the not digits and replace the digits with #

```
Ex 1: A = 234               Output: ###
Ex 2: A = a2b3c4            Output: ###
Ex 3: A = abc               Output:   (empty string)
Ex 5: A = #2a$#b%c%561#     Output: ####
```

```
import re
# write your python code here
# you can take the above example as sample input for your program to test
# it should work for any general input try not to hard code for only given input ex

# you can free to change all these codes/structure
# String: it will be the input to your program
def replace_digits(s):
    # write your code
    res = ''
    for e in s:
      # checking if the difference between ascii of 0 and the element 'e' in s lies
      if(ord(e) - ord('0') in range (0, 10)):
        res +='#'
    return(res) # modified string which is after replacing the # with digits

print(replace_digits('a2b3c4'))
print(replace_digits('234'))
print(replace_digits('abc'))
print(replace_digits('#2a$#b%c%561#'))
```

```
###
###
```

####

Q4: Students marks dashboard

consider the marks list of class students given two lists
Students =
['student1','student2','student3','student4','student5','student6','student7','student8','student9','student10']
Marks = [45, 78, 12, 14, 48, 43, 45, 98, 35, 80]
from the above two lists the Student[0] got Marks[0], Student[1] got Marks[1] and so on

your task is to print the name of students **a. Who got top 5 ranks, in the descending order of marks**
b. Who got least 5 ranks, in the increasing order of marks
d. Who got marks between >25th percentile <75th percentile, in the increasing order of marks

```
Ex 1:
Students=['student1','student2','student3','student4','student5','student6','stud
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
a.
student8  98
student10 80
student2  78
student5  48
student7  47
b.
student3 12
student4 14
student9 35
student6 43
student1 45
c.
student9 35
student6 43
student1 45
student7 47
student5 48
```

```
Students=['student1','student2','student3','student4','student5','student6','studen
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
```

```
import math
# you can free to change all these codes/structure
def sort_by_marks(student_marks):
    for i in range (len(student_marks)):
```

```
        for j in range(i+1, len(student_marks)):
          if(student_marks[i][1] > student_marks[j][1] ):
            temp = student_marks[i]
            student_marks[i] = student_marks[j]
            student_marks[j] = temp
    return student_marks

def display_dash_board(students, marks):

    student_marks = []
    # store marks and student in a list of tuple
    for i in range(len(Marks)):
      student_marks.append((Students[i], Marks[i]))

    # sort in ascending order by value of marks
    sort_by_marks(student_marks)

    # top_5_students
    top_5_students = student_marks[-5:][::-1]
    # least_5_students
    least_5_students = student_marks[0:5]
    # students within 25 and 75
    index_25 = math.floor(0.25 * len(student_marks))
    index_75 = math.floor(0.75 * len(student_marks))
    students_within_25_and_75 = student_marks[index_25: index_75]
    return top_5_students, least_5_students, students_within_25_and_75

top_5_students, least_5_students, students_within_25_and_75 = display_dash_board(St



print("Students = ", Students)
print("Marks = ", Marks)
print("a.")
for e in top_5_students:
  print(e[0], " ", e[1])
print("b.")
for e in least_5_students:
  print(e[0], " ", e[1])
print("c.")
print("a.")
for e in students_within_25_and_75:
  print(e[0], " ", e[1])
```

```
Students =  ['student1', 'student2', 'student3', 'student4', 'student5', 'stud
Marks =  [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
a.
student8   98
student10   80
student2   78
student5   48
student7   47
b.
student3   12
student4   14
student9   35
student6   43
```

```
student1   45
c.
a.
student9   35
student6   43
student1   45
student7   47
student5   48
```

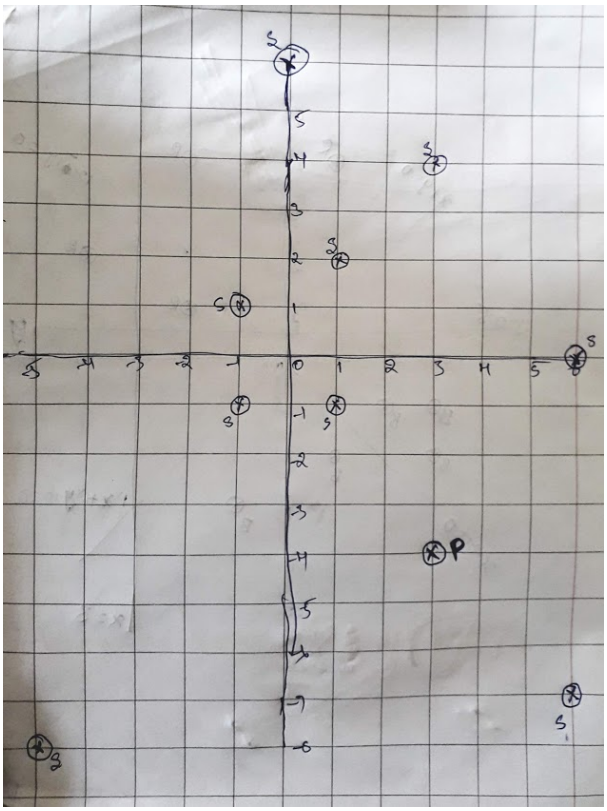
Mapping can be also be done using a dictionary which will make the code more efficient and clean

Q5: Find the closest points

consider you have given n data points in the form of list of tuples like S=[(x1,y1),(x2,y2),(x3,y3),(x4,y4),(x5,y5),..,(xn,yn)] and a point P=(p,q)
your task is to find 5 closest points(based on cosine distance) in S from P
cosine distance between two points (x,y) and (p,q) is defind as $cos^{-1}(\frac{(x\cdot p+y\cdot q)}{\sqrt{(x^2+y^2)}\cdot\sqrt{(p^2+q^2)}})$

Ex:

S= [(1,2),(3,4),(-1,1),(6,-7),(0, 6),(-5,-8),(-1,-1)(6,0),(1,-1)]
P= (3,-4)



Output:
(6,-7)
(1,-1)
(6,0)
(-5,-8)

```
(-1,-1)
```

```
import math
# function to find distance between 2 points
def find_distance(A,  B):
  dist = math.acos(((A[0]*B[0]) + (A[1] * B[1])) / ((math.sqrt(A[0]**2 + A[1]**2))
  return dist


# function to sort the array by distance - bubble sort used
def sort_dist(res):
  for i in range(len(res)):
    for j in range(i+1, len(res)):
      if(res[i][0] > res[j][0]):
        temp = res[i]
        res[i] = res[j]
        res[j] = res[i]
  return res


def closest_points_to_p(S, P):
    # write your code here
    dist = []
    for point in S:
      dist.append([find_distance(point, P), point])
    # sort ascending order by value of distance
    dist = sorted(dist)

    result = []
    for ele in dist:
      result.append(ele[1])

    return (result[:5])

S= [(1,2),(3,4),(-1,1),(6,-7),(0, 6),(-5,-8),(-1,-1),(6,0),(1,-1)]
P= (3,-4)
points = closest_points_to_p(S, P)
print(points)
```

```
[(6, -7), (1, -1), (6, 0), (-5, -8), (-1, -1)]
```

Q6: Find Which line separates oranges and apples

consider you have given two set of data points in the form of list of tuples like

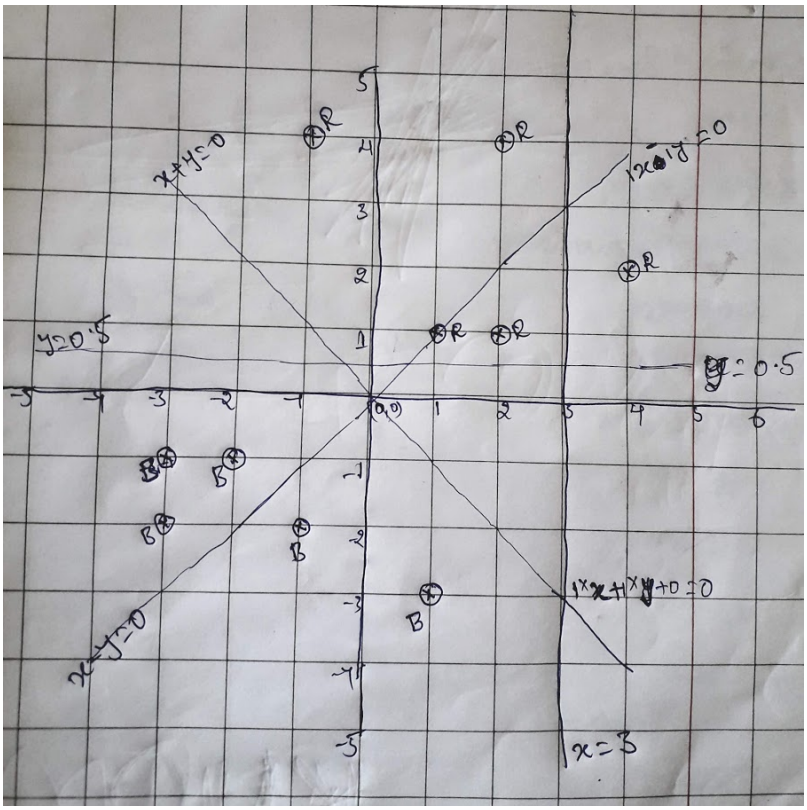
```
Red =[(R11,R12),(R21,R22),(R31,R32),(R41,R42),(R51,R52),..,(Rn1,Rn2)]
Blue=[(B11,B12),(B21,B22),(B31,B32),(B41,B42),(B51,B52),..,(Bm1,Bm2)]
```

and set of line equations(in the string formate, i.e list of strings)

Lines = [a1x+b1y+c1,a2x+b2y+c2,a3x+b3y+c3,a4x+b4y+c4,..,K lines]
Note: you need to string parsing here and get the coefficients of x,y and interce

your task is to for each line that is given print "YES"/"NO", you will print yes, if all the red points are one side of the line and blue points are other side of the line, otherwise no

Ex:
Red= [(1,1),(2,1),(4,2),(2,4), (-1,4)]
Blue= [(-2,-1),(-1,-2),(-3,-2),(-3,-1),(1,-3)]
Lines=["1x+1y+0","1x-1y+0","1x+0y-3","0x+1y-0.5"]



Output:
YES
NO
NO
YES

```
import math

def i_am_the_one(red,blue,line):

    # inserting * sign in the formula
    formula = ''
    for s in line:
      if(s == 'x' or s == 'y'):
        formula+='*'
      formula += s

    ds1, ds2= [], []
```

```
    for val in red:
      x=val[0]
      y=val[1]
      result = eval(formula) # https://stackoverflow.com/a/65696924/14049383 - refe
      ds1.append(result)
    # check if all are of the same sign
    res1 = all(item >= 0 for item in ds1) or all(item < 0 for item in ds1)


    for val in blue:
      x=val[0]
      y=val[1]
      result = eval(formula) # https://stackoverflow.com/a/65696924/14049383 - refe
      ds2.append(result)
    # check if all are of the same sign
    res2 = all(item >= 0 for item in ds2) or all(item < 0 for item in ds2)

    if(res1 and res2):
      if((ds1[0] > 0 and ds2[0]<0) or (ds1[0] < 0 and ds2[0] > 0)):
        return 'YES'
    return "NO"

Red= [(1,1),(2,1),(4,2),(2,4), (-1,4)]
Blue= [(-2,-1),(-1,-2),(-3,-2),(-3,-1),(1,-3)]
Lines=["1x+1y+0","1x-1y+0","1x+0y-3","0x+1y-0.5"]

for i in Lines:
    yes_or_no = i_am_the_one(Red, Blue, i)
    print(yes_or_no) # the returned value

     YES
     NO
     NO
     YES
```

Q7: Filling the missing values in the specified formate

You will be given a string with digits and '_'(missing value) symbols you have to replace the '_' symbols as explained

```
/4, 24/4 i.e we. have distributed the 24 equally to all 4 places

(60+40)/5,(60+40)/5,(60+40)/5,(60+40)/5 ==> 20, 20, 20, 20, 20 i.e. the sum of (60

80/5,80/5,80/5 ==> 16, 16, 16, 16, 16 i.e. the 80 is distributed qually to all 5 m


from left to right
30 to left two missing values (10, 10, 10, _, _, _, 50, _, _)
) missing values in between (10, 10, 12, 12, 12, 12, 12, _, _)
right side missing values (10, 10, 12, 12, 12, 12, 4, 4, 4)
```

for a given string with comma seprate values, which will have both missing values numbers like ex: "_, _, x, _, _, _" you need fill the missing values

Q: your program reads a string like ex: "_, _, x, _, _, _" and returns the filled sequence

Ex:

```
Input1: "_,_,_,24"
Output1: 6,6,6,6

Input2: "40,_,_,_,60"
Output2: 20,20,20,20,20

Input3: "80,_,_,_,_"
Output3: 16,16,16,16,16

Input4: "_,_,30,_,_,_,50,_,_"
Output4: 10,10,12,12,12,12,4,4,4
```

```
# reference : https://stackoverflow.com/a/60429300/14049383

def curve_smoothing(s):
    lst=s.split(',')
    for i in range(len(lst)):
        if (lst[i]!='_'):
            for j in range(i+1):
                lst[j]=int(lst[i])//(i+1)
            new_index=i
            new_value=int(lst[i])
            break
    for i in range(new_index+1,len(lst)):
        if (lst[i]!='_'):
            temp=(new_value+int(lst[i]))//(i-new_index+1)
            for j in range(new_index,i+1):
                lst[j]=temp
            new_index=i
            new_value=int(lst[i])
    try:
        for i in range(new_index+1,len(lst)):
            if not(lst[i].isdigit()):
                count=lst.count('_')
                break
        temp1=new_value//(count+1)
        for i in range(new_index,len(lst)):
            lst[i]=temp1
    except:
        pass
    return lst


S=  "_,_,_,24"
smoothed_values= curve_smoothing(S)
```

```
smoothed_values= curve_smoothing(S)
print(smoothed_values)
```

```
[6, 6, 6, 6]
```

```
#################### MY ATTEMPT ####################

def fill_res(res, i, j, val):
  for x in range(i, j+1):
    res[x] = val

def curve_smoothing(s):
    arr = s.split(',')
    res = [0]*len(arr)
    i, j = 0, 0
    cnt, right, left = 0, 0, 0
    while(i<len(arr) and j<len(arr)):
      if(arr[j]=='_'):
        cnt+=1
        j+=1
      else:
        right = int(arr[j])
        if(i==0):
          val = (left+right)/(cnt+1)
          fill_res(res, i, j, val)
        else:
          val = (left+right)/(cnt+2)
          fill_res(res, i, j, val)
        i = j
        left = val

        cnt = 0
        j+=1

    if(i<len(arr)-1):
        val = (left+0)/(cnt+1)
        fill_res(res, i, len(arr)-1, val)

    return res


S=  "_,_,_,24"
smoothed_values= curve_smoothing(S)
print(smoothed_values)
```

```
[6.0, 6.0, 6.0, 6.0]
```

Q8: Filling the missing values in the specified formate

You will be given a list of lists, each sublist will be of length 2 i.e. [[x,y],[p,q],[l,m]..[r,s]] consider its like a martrix of n rows and two columns 1. the first column F will contain only 5 uniques values (F1, F2, F3, F4, F5) 2. the second column S will contain only 3 uniques values (S1, S2, S3)

your task is to find
a. Probability of P(F=F1|S==S1), P(F=F1|S==S2), P(F=F1|S==S3)
b. Probability of P(F=F2|S==S1), P(F=F2|S==S2), P(F=F2|S==S3)
c. Probability of P(F=F3|S==S1), P(F=F3|S==S2), P(F=F3|S==S3)
d. Probability of P(F=F4|S==S1), P(F=F4|S==S2), P(F=F4|S==S3)
e. Probability of P(F=F5|S==S1), P(F=F5|S==S2), P(F=F5|S==S3)

Ex:

[[F1,S1],[F2,S2],[F3,S3],[F1,S2],[F2,S3],[F3,S2],[F2,S1],[F4,S1],[F4,S3],[F5,S1]]

a. P(F=F1|S==S1)=1/4, P(F=F1|S==S2)=1/3, P(F=F1|S==S3)=0/3
b. P(F=F2|S==S1)=1/4, P(F=F2|S==S2)=1/3, P(F=F2|S==S3)=1/3
c. P(F=F3|S==S1)=0/4, P(F=F3|S==S2)=1/3, P(F=F3|S==S3)=1/3
d. P(F=F4|S==S1)=1/4, P(F=F4|S==S2)=0/3, P(F=F4|S==S3)=1/3
e. P(F=F5|S==S1)=1/4, P(F=F5|S==S2)=0/3, P(F=F5|S==S3)=0/3

```
# you can free to change all these codes/structure
def compute_conditional_probabilites(A):
    countS1=0
    countS2=0
    countS3=0
    for ele in A:
      if(ele[1]=='S1'):
        countS1=1+countS1
      elif(ele[1]=='S2'):
        countS2=1+countS2
      else:
        countS3=1+countS3
    #finding conditional probabilities for all
    countF1S1=0
    countF1S2=0
    countF1S3=0
    countF2S1=0
    countF2S2=0
    countF2S3=0
    countF3S1=0
    countF3S2=0
    countF3S3=0
    countF4S1=0
    countF4S2=0
    countF4S3=0
    countF5S1=0
    countF5S2=0
    countF5S3=0
    for ele in A:
      if(ele[0]=='F1'):
        if(ele[1]=='S1'):
          countF1S1=1+countF1S1
        elif(ele[1]=='S2'):
```

```
            countF1S2=1+countF1S2
          else:
            countF1S3=1+countF1S3


        elif(ele[0]=='F2'):
          if(ele[1]=='S1'):
            countF2S1=1+countF2S1
          elif(ele[1]=='S2'):
            countF2S2=1+countF2S2
          else:
            countF2S3=1+countF2S3

        elif(ele[0]=='F3'):
          if(ele[1]=='S1'):
            countF3S1=1+countF3S1
          elif(ele[1]=='S2'):
            countF3S2=1+countF3S2
          else:
            countF3S3=1+countF3S3
        elif(ele[0]=='F4'):
          if(ele[1]=='S1'):
            countF4S1=1+countF4S1
          elif(ele[1]=='S2'):
            countF4S2=1+countF4S2
          else:
            countF4S3=1+countF4S3
        else:
          if(ele[1]=='S1'):
            countF5S1=1+countF5S1
          elif(ele[1]=='S2'):
            countF5S2=1+countF5S2
          else:
            countF5S3=1+countF5S3

      print("P(F=F1|S==S1)=",countF1S1 ,"/",countS1,",","P(F=F1|S==S2)=",countF1S2 ,"
      print("P(F=F2|S==S1)=",countF2S1 ,"/",countS1,",","P(F=F2|S==S2)=",countF2S2 ,"
      print("P(F=F3|S==S1)=",countF3S1 ,"/",countS1,",","P(F=F3|S==S2)=",countF3S2 ,"
      print("P(F=F4|S==S1)=",countF4S1 ,"/",countS1,",","P(F=F4|S==S2)=",countF4S2 ,"
      print("P(F=F5|S==S1)=",countF5S1 ,"/",countS1,",","P(F=F5|S==S2)=",countF5S2 ,"


A = [['F1','S1'],['F2','S2'],['F3','S3'],['F1','S2'],['F2','S3'],['F3','S2'],['F2',

compute_conditional_probabilites(A)
```

```
P(F=F1|S==S1)= 1 / 4 , P(F=F1|S==S2)= 1 / 3 , P(F=F1|S==S3)= 0 / 3
P(F=F2|S==S1)= 1 / 4 , P(F=F2|S==S2)= 1 / 3 , P(F=F2|S==S3)= 1 / 3
P(F=F3|S==S1)= 0 / 4 , P(F=F3|S==S2)= 1 / 3 , P(F=F3|S==S3)= 1 / 3
P(F=F4|S==S1)= 1 / 4 , P(F=F4|S==S2)= 0 / 3 , P(F=F4|S==S3)= 1 / 3
P(F=F5|S==S1)= 1 / 4 , P(F=F5|S==S2)= 0 / 3 , P(F=F5|S==S3)= 0 / 3
```

Q9: Given two sentances S1, S2

You will be given two sentances S1, S2 your task is to find

```
a. Number of common words between S1, S2
b. Words in S1 but not in S2
c. Words in S2 but not in S1
```

Ex:

```
S1= "the first column F will contain only 5 uniques values"
S2= "the second column S will contain only 3 uniques values"
Output:
a. 7
b. ['first','F','5']
c. ['second','S','3']
```

```
# you can free to change all these codes/structure
def string_features(S1, S2):
    S1.lower()
    S2.lower()
    s1_words = set(S1.split())
    s2_words = set(S2.split())
    a = len(s1_words.intersection(s2_words))
    b = s1_words - s2_words
    c = s2_words - s1_words
    return a, b, c

S1= "the first column F will contain only 5 uniques values"
S2= "the second column S will contain only 3 uniques values"
a,b,c = string_features(S1, S2)
print(a)
print(b)
print(c)
```

```
7
{'5', 'F', 'first'}
{'S', 'second', '3'}
```

Q10: Given two sentances S1, S2

You will be given a list of lists, each sublist will be of length 2 i.e. [[x,y],[p,q],[l,m]..[r,s]] consider its like a martrix of n rows and two columns

a. the first column Y will contain interger values

b. the second column Y_{score} will be having float values

Your task is to find the value of

$f(Y, Y_{score}) = -1 * \frac{1}{n}\Sigma_{foreachY,Y_{score}pair}(Ylog10(Y_{score}) + (1-Y)log10(1-Y_{score}))$ here n is the number of rows in the matrix

```
Ex:
[[1, 0.4], [0, 0.5], [0, 0.9], [0, 0.3], [0, 0.6], [1, 0.1], [1, 0.9], [1, 0.8]]
output:
0.4243099
```

$\frac{-1}{8} \cdot ((1 \cdot log_{10}(0.4) + 0 \cdot log_{10}(0.6)) + (0 \cdot log_{10}(0.5) + 1 \cdot log_{10}(0.5)) + \ldots + (1 \cdot log_{10}(0$

```
import math


# you can free to change all these codes/structure
def compute_log_loss(A):
    # your code
    summation_value = 0
    for li in A:
      summation_value += ((li[0] * math.log10(li[1])) + ((1-li[0]) * math.log10(1-l
    loss = -1 * summation_value / len(A)
    return loss

A = [[1, 0.4], [0, 0.5], [0, 0.9], [0, 0.3], [0, 0.6], [1, 0.1], [1, 0.9], [1, 0.8]
loss = compute_log_loss(A)
print(round(loss, 7))
```

```
0.4243099
```

✓ 0s completed at 5:29 PM